

LINUX VOICE

The magazine that gives back to the Free Software community

June 2014

PYTHON

ASTRONOMY

Search the blackness of space for comets

RASPBERRY PI

HARDWARE

Build a distress beacon for when the zombies come

LILYPAD

FASHION

Embed blinkenlights in your cycling jacket

**114 PAGES
OF NEURAL
ENHANCEMENT!**

I'VE GOT THE

POWER

UNLEASH THE HIDDEN STRENGTH OF THE LINUX COMMAND LINE

FREE SOFTWARE | FREE SPEECH

32+ PAGES OF TUTORIALS

LIBVIRT Manage virtual machines with Python
LXDE Upgrade your desktop from dowdy to dashing
GOOGLE APP SCRIPTING Share even more data with Google

OLDE CODE

TURING

Programmer, code breaker, genius



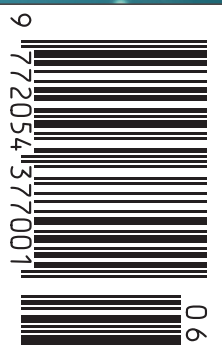
FIDEI DEFENSOR

FREEDOM!

Inside the Free Software Foundation Europe



June 2014 £5.99 Printed in the UK



ISSN 2054-3778

As the stewards of the Open Source Definition (OSD) and the community-recognized body for reviewing and approving licenses as OSD-conformant, the OSI facilitates Open Source community-building, education, and public advocacy to promote awareness, adoption and the importance of non-proprietary software.



The OSI, with global reach, champions Open Source software and projects, meeting with developers, users and communities as well as with executives from the public and private sectors to explain how Open Source technologies, licensing & models can provide economic, strategic and societal advantages.

Open Source Initiative welcomes Linux Voice to the global Open Source community.

Join the Open Source Initiative now
and be a part of the future of Open Source.

[<opensource.org/members>](http://opensource.org/members)



ELECTRONIC FRONTIER FOUNDATION

Help EFF Defend Your Rights in the Digital World eff.org/join

Command and conquer

The **June** issue

LINUX VOICE

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

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

2 No later than nine months after first publication, we will relicence 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 (you again).

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 Liam Dawe
liam@linuxvoice.com

Creative director Stacey Black
stacey@linuxvoice.com

Malign puppetmaster Nick Veitch
nick@linuxvoice.com

Editorial contributors:
Mark Crutch, Juliet Kemp, John Lane,
Vincent Mealing, Simon Phipps,
Jonathan Roberts,
Mayank Sharma, Valentine Sinitsyn



GRAHAM MORRISON

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

It's truly remarkable that despite an almost infinite expansion in bandwidth and computing power since the 1970s, the humble command line has remained relevant, and perhaps, become even more relevant. We were promised voice input and Minority Report-style gesture control, and in some technology that's what we've got. But whatever advancements have come along, whether it's the mouse or drag-and-drop, the direct and indivisible connection between your words and the command prompt cannot be beaten.

Which is why there has never been a better time to take the plunge. It's not difficult, and you can experiment safely from a new user account or a virtual machine without any worry of wayward arguments deleting important files. Even when you go back to your desktop, there's something very empowering knowing that, if you needed to, you could drop back to a prompt, regardless of your distro, hardware or connection, and perform almost any task. And there's no other operating system that gives you that kind of power.

Graham Morrison
Editor, Linux Voice

**SUBSCRIBE
ON PAGE 64**

What's hot in LV#003



ANDREW GREGORY

Attack ships off the Shoulder of Orion! Well, almost. Discover comets from the comfort of your Linux box **p86**



BEN EVERARD

Even though it's something I've written, I'm really pleased with my Arduino-based cycling jacket. I've not had a crash yet! **p80**



MIKE SAUNDERS

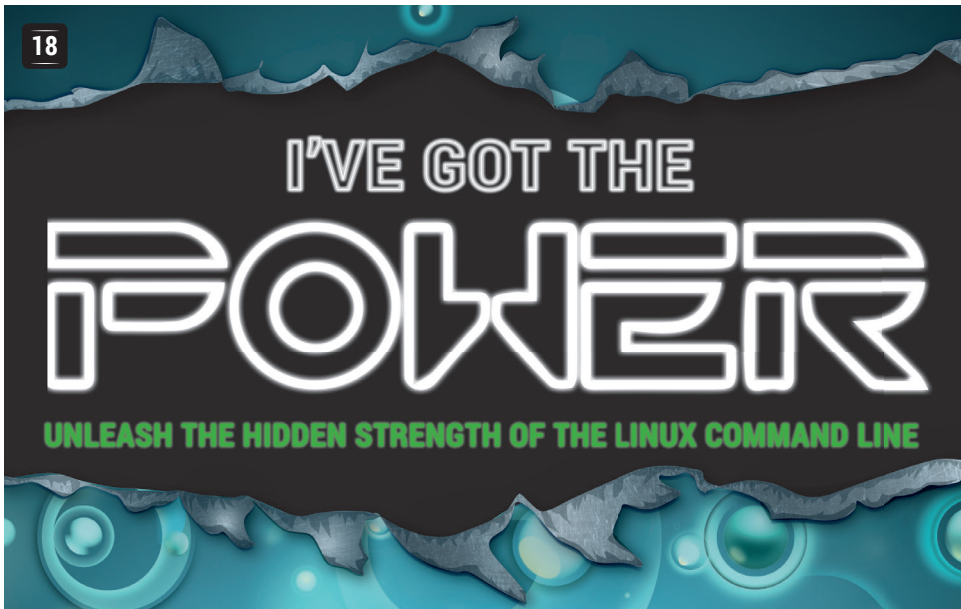
Looking for a new Linux PC? We went into real shops and asked them about Linux support. Their responses were surprising **p26**



CONTENTS

June LV003

We are all in the gutter, but some of us are searching for comets

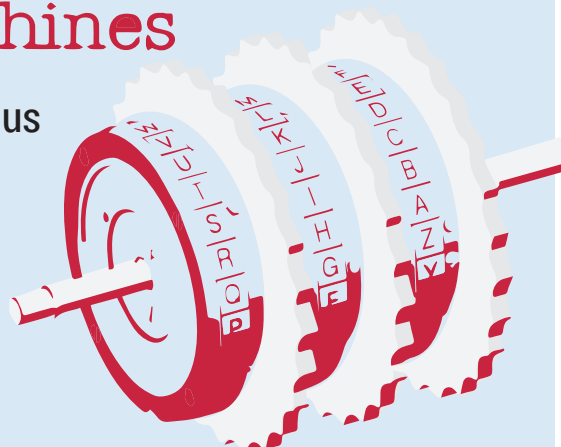


Master the command line

Peek under the surface of your Linux box and reveal the power within

Alan Turing, Colossus and Turing Machines

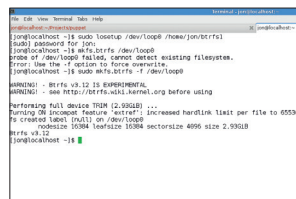
How one ordinary genius came to develop the single most important computer ever made.



26 SHOPPING
How to buy a Linux box on the high street.



42 FAQ Bad guys want to DDOS your server. But why, and how?



66 SYSADMIN
Jon Roberts finds a better filesystem.

REGULARS

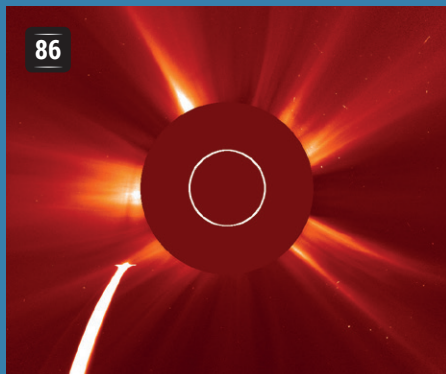
- 06 News**
Bitesize facts to lodge in our receptive brains.
- 08 Distrohopper**
What's rolling off the Linux production line this month?
- 10 Gaming**
Portal 2, The Dark Mod, Oil Rush, Wasteland 2 and more.
- 12 Speak your brains**
Ideas, criticisms, suggestions and book review confusion.
- 16 LV on tour**
Brighton, Bristol, Manchester and Munich get a visitation.
- 34 Open Cores**
A hardware architecture that's open to hack about with.
- 44 Interview**
Damian Conway – one of the founding fathers of Perl.
- 58 Group test**
Reach out and touch your shiny new Linux desktops.
- 110 Masterclass**
Get deep into Inkscape and ImageMagick. Arty!
- 114 My Linux desktop**
Mr KDE Plasma tablet Aaron Seigo shows us around.

TUTORIALS



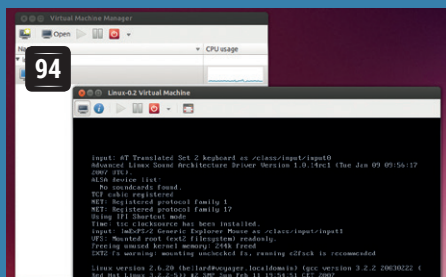
Customise the LXDE desktop

Make your Raspberry Pi a lot prettier, by enhancing its default desktop environment.



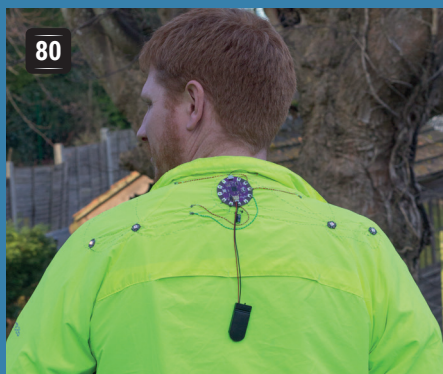
Hunt comets with Python and open data

Filter image data in the search for Thargoids comets, from the comfort of your Linux machine.



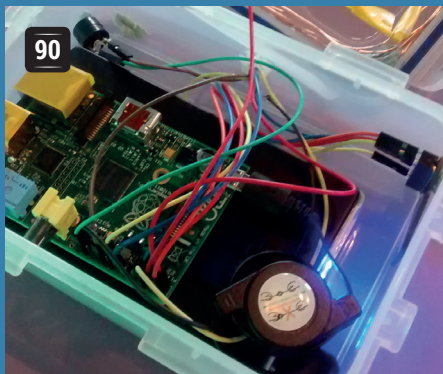
Control virtual machines with Python and libvirt

Dispense with the GUI for the awesome power of virtual machines commanded by Python.



Make smart clothes with an Arduino Lilypad

Sew a wearable circuit into clothing to turn your clothing into an electronic canvas.



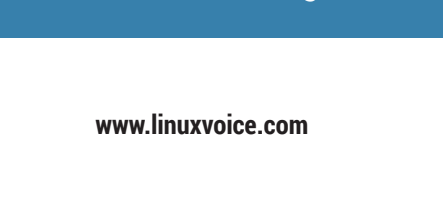
Raspberry Pi: build an emergency beacon

Stay safe in the event of disaster by broadcasting the theme from Star Wars from a lunch box.



Code 101 Genetic algorithms: create life with Python

Programming done backwards, for people like Ben and NASA.



106 Ruby: Why absolutely, positively, everything is an object

Get your head around object-oriented to the nth degree.

REVIEWS



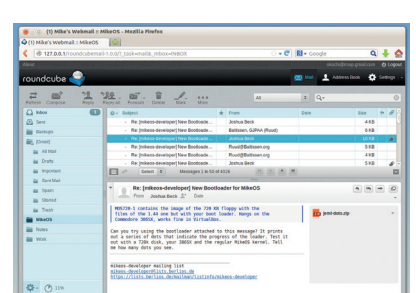
50 Bitwig 1.0 In one bound, audio production and editing on Linux became a lot more interesting, and cheaper, and better. Win!



52 Gnome 3.12 This 'next generation' desktop came in for a lot of criticism on its 3.0 release – has it improved?

53 Udoo Do you love playing with your Raspberry Pi, but feel like it's lacking in grunt? If so, this quad-core ARM device awaits.

54 Roundcube 1.0 If you think webmail is all about selling your details to advertisers, you could be in for a pleasant surprise.



55 Pibrella In our day toys were lumps of junk that taught only disappointment. The world has got a lot better since then.

56 Books Ada Lovelace, Nine Algorithms That Changed the Future, and the scribbles of some chap called Everard.

NEWS ANALYSIS

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

Opinion

Birth of a party

Calm down Mr Farage: there's some actual politics happening in Brussels!



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

As society changes, so can European-level politics. The formation of the European Pirate Party this weekend was thus an event of unique interest. Responding to the emergence of the meshed society, geeks of all flavours realised there was no one speaking up for the reality they saw.

The treatment of the Pirate Bay was the catalyst. Instead of considering its existence as civil disobedience arising from market failure, politicians sided with the legacy media intermediaries in treating it as organised crime akin to drug dealing or currency counterfeiting.

The result was outrage, at least in the more socially coherent communities of Scandinavia and Germany. Political parties formed almost spontaneously in those countries, dedicated to bringing about a just society for the digital age. The smirks of derision of the incumbent politicians in those countries faded as they discovered this was not just a protest movement but was electorally viable. In 2009 Christian Engström was elected representing Sweden in the European Parliament, as was the brilliant Amelia Andersdotter. Then in

Germany, the party gained seats in various elections. The movement spread, chaotic rabble though it often appeared, until this year it became clear it was present throughout Europe.

Which brings us to here. This weekend, the grassroots movement from all of Europe assembled in Brussels for the next stage of organisation. Some present were concerned that the whole new thing was getting too much like the old thing it wanted to replace. But those few bit their lip and savoured the moment as a rabble of smart geeks (many an open source committer among them) became the newest political party in Europe.

Bright beginnings

Not all was shiny. The opening keynotes for the event were frankly disappointing for an outsider, with little sign of a coherent intellectual framework being propounded by the speakers and no speech from Amelia Andersdotter, the clear leader of the party at this juncture. But there were flashes of promising insight. Keynote speaker (and later Plenary chair) Julia Rede said "I want a Europe more like the internet – connected, collaborative and a community of peers." This is a visionary statement, which could provide a prototype for a policy framework.

Rede also recognised that a Pirate Party needs to be more than just a movement comprising Linux sysadmins, as well as tackling current issues like the Transatlantic Trade and Investment Partnership (TTIP): "The greatest fear of TTIP is that it will raise corporations to the same status as states". She is a candidate in the upcoming



The European Pirate Party was founded in March 2014, and should have a great future.

European Elections and it's clear that the European Pirate Party needs her elected.

And that name. It turns out that people with English as a second language have fewer problems with it, picking up the humour and Disneyesque romance. But the negative implications, both from the real meaning of the word "pirate" and its cynical misuse in connection with unauthorised copyright usage, tend to make it become a slur to many with English as a first language. This doesn't have to be a problem though. It's a word true to the origins of the movement, and there is a historic precedent for adopting the slurs of one's detractors as a name. The Tories did it (the term originally meaning a mugger); before them the Whigs did it; even Christianity did it. With time, the name Pirate Party will become a strength rather than a weakness.

This is not a political movement funded from the deep pockets of corporations or unions. There was no expensive partying, no glitz or glamour, no side meetings funded by sponsors, no exhibition to gain fees. But there was broad attendance from across Europe by party members and supporters of all ages. The Pirate Party is a voice we all need in Europe. Welcome!

"The Pirate Party needs to be more than just a movement comprising Linux sysadmins."

CATCHUP

Summarised: the eight biggest news stories from the last month

1

Windows XP EOL pushing companies to try Linux

It's official: Windows XP is dead. The OS had an incredibly long life, but even Microsoft got sick of supporting it. However, with an estimated 70% of businesses still running at least one XP machine, what can they do? Buy new hardware and upgrade to the UI nightmare that is Windows 8? Or switch to Linux? Well, various surveys are suggesting that between 10 and 30% of businesses are now considering moving to Linux. That's millions of new users coming...

2

Quake III ported to the Raspberry Pi

Hang on – Quake III came out in 1999. So that's hardly impressive, eh? Well, it is when it's part of a \$10,000 bounty. In February, Broadcom announced a wealth of documentation for the graphics chip in the Pi, and provided a driver for a similar chip. Then the Raspberry Pi foundation offered \$10,000 to the first person to port the driver to the Pi's own GPU and show Quake III running silky-smooth on it. Long-time Pi hacker Simon Hall managed it, and bagged the loot.

3

QR codes considered for kernel crash messages

Kernel panic text isn't useful to many people, but if non-technical users could send a QR code containing the information to the relevant developers, that could help everyone.



4

GOG.com announces plan to support Linux

Formerly known as Good Old Games, GOG.com is a rather spiffing site that sells (mainly older) games without DRM and other such horrible nonsense. The company has announced plans to support Linux in the near future, focusing on Ubuntu and Mint, with 100 games in the pipeline. There aren't many other details right now, but for some gamers, it will be one fewer reason to reboot into Windows. Read the full announcement here: <http://tinyurl.com/kcdozjz>

5

Major GnuTLS bug leaves "secure" websites open to eavesdropping

It's easy to become complacent as a Linux user, given the overall reliability and security of the operating system. But vulnerabilities do happen, and now a Red Hat researcher has found a problem with GnuTLS, a library that implements TLS and SSL support for websites. The vulnerability affects certificate verification, potentially showing unsecure, spied-on connections as secure. Distros are scrambling to push out fixed packages.

6

Linus Torvalds refuses to accept any more code from prominent hacker

Linus doesn't mince his words, especially when someone breaks his beloved kernel. Kay Sievers, one of the developers of systemd, managed to bring Linus's blood to a boil over a complication with the "debug" flag on the kernel command line. (As if systemd weren't controversial enough...) "I will not be merging any code from Kay into the kernel" until fixes are made, said Linus. Check out the the full email: <http://tinyurl.com/linusrage>

7

Microsoft Office released for Linux – well, Android

But Android is a flavour of Linux, right? It was an unusually cold day in Hell when Microsoft announced this, but yes, you can now get mini versions of Word, Excel and PowerPoint for your Android device. The reviews so far aren't very positive, with users complaining of problems opening desktop Office files. Which is especially ironic, as Microsoft always championed its supposed cross-app and platform file compatibility. Anyway, the suite is also available for iOS.



The Giant Panda

The giant panda, which lives only in China outside of captivity, has captured the hearts of people of all ages across the globe. From their furry black and white body to their shy and docile nature, they are considered one of the most loved animals. |



8

\$2,400 'Introduction to Linux' course made free

This is jolly generous of The Linux Foundation. Its 'Introduction to Linux' course, previously available for a wallet-bursting \$2,400, will soon be freely available for everyone. The Foundation plans to release it as part of a MOOC (Massively Open Online Course), so if you're fairly new to Linux and still trying to fathom your way around the operating system, it could help you enormously. Here's the full text of the announcement: <http://tinyurl.com/lskhx8u>

DISTROHOPPER

We've tapped GCHQ's communications to find out what's going on in distro land.

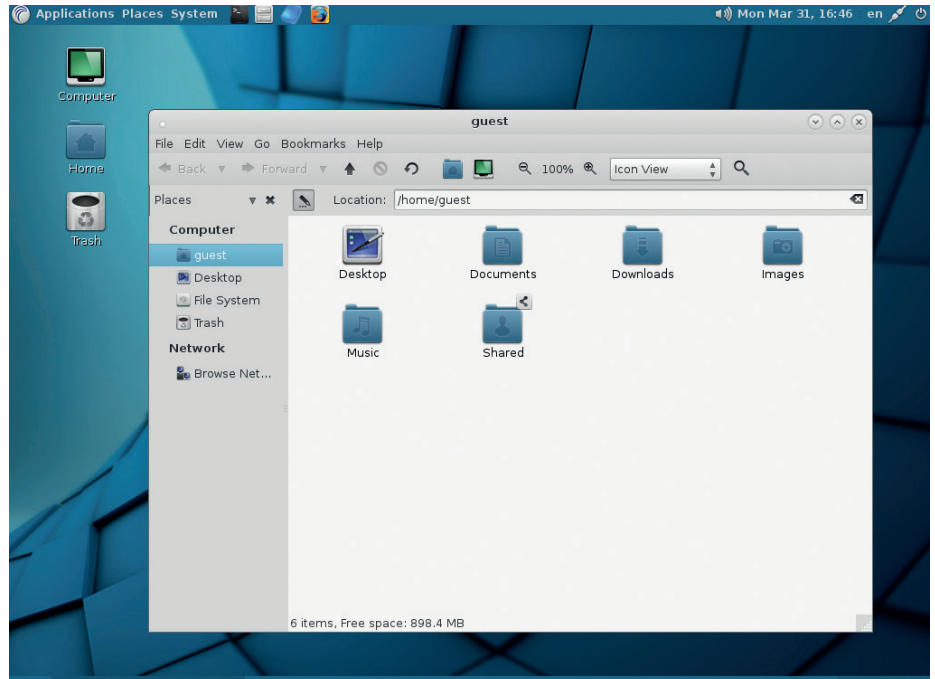
Porteus

Lightning-fast live Linux.

Porteus is a distribution designed for running directly off a USB stick. Big deal – almost every distribution can do that these days, so you may wonder if there's still room for a distro like Porteus.

However, there's plenty that's unusual about it. Porteus doesn't have a download link like you may expect, but a build service where you can customise your own version of Porteus. You can select things like the desktop environment, the web browser and word processor, and in the Advanced options you can set things like the system passwords and boot behaviour.

The second unusual thing about Porteus is how quickly it boots. On our test system, Porteus Mate got from Grub to the desktop in under 10 seconds, and this was in a virtual machine. The VM was allocated two cores of the i7 processor, but to put it into context, OpenSuse took just over a minute. This speed is incredible for a live distro. We ran the test again with the KDE version of Porteus, and even with the heavyweight



Porteus Kiosk has a bounty scheme where you can donate money towards certain features.

distro, it took just 15 seconds. On the down side there is quite a limited choice for the applications you can install, and there's little else other than those you select in the build system. This means that Porteus is

targeted very firmly at people who want a minimal live distro. Of course, this isn't really a problem, because there are loads of alternative options for people who want a heavyweight live desktop.

Kali 1.0.6

Hindu Goddess and pen-tester's best friend.

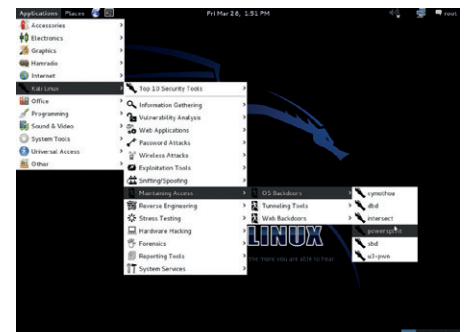
Kali Linux is undoubtedly the premiere penetration testing Linux distro. It's packed full of all manner of security-focused software, such as the Metasploit Framework. It's also got a lot of software that is not usually thought of as security focused, but has its place in the hacker's arsenal, such as the Arduino IDE.

Much of the security software is command line-based, but the developers have still included it in the applications menu to make it easy to find. When you click on the menu item, it opens a terminal window

and outputs the help from that command. It's a nice way of doing things that combines the power of the command line with the discoverability of a graphical system, and is a trick that quite a few other distros would do well to learn.

In addition to 32- and 64-bit PCs, there are builds available for Amazon Machine Images (AMI), Google's Compute Engine, and nine different ARM systems (including the Raspberry Pi).

Finally, we can't help but wonder if the artwork in Kali has been deliberately



If you're not a security expert, browsing the Kali menu can be an eye-opening experience.

designed to make it look edgy. The stylised dragon on the desktop seems to be there to remind you that the software running is the digital equivalent of a private ninja army.

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons



TENTACLES OF CTHULU



Liam Dawe is our Games Editor and the founder of gamingonlinux.com, the home of Tux gaming on the web.

SteamOS, from Valve, is a Linux distribution aimed solely at gaming, with Valve's own patches included to increase performance of various systems within the distribution. SteamOS was one of the major announcements that Valve made in September 2013, alongside its own gaming controller (the Steam Controller), and also its console-like Steam Machines. The distribution will be pre-installed on Steam Machines, removing a barrier to Linux adoption.

SteamOS has gained positive feedback from quite a few big names in the indie gaming scene, including Markus "Notch" Persson, creator of Minecraft, and DICE, the creator of the Battlefield series of first person shooters. But the really fantastic thing about Valve pushing Linux so profusely is that it has led the leading graphics card manufacturers (Nvidia and AMD) to improve their graphics driver performance.

Valve has recently open sourced two internal projects, the first being its Direct3D to OpenGL translation layer, which could prove useful for other big name developers in saving time when porting to Linux.

The second of these big projects is the new OpenGL debugger named "Vogl", which should help developers boost performance and find bottlenecks in their OpenGL rendering. To put it simply, performance of games for us Linux gamers in future should improve if they use Vogl. Onwards and upwards! Liam Dawe, gamingonlinux.com

Portal 2

Prepare to return to the Aperture Science Labs.

Look out puzzle fans – here comes a big one! Valve, our new benevolent gaming overlord, has unleashed the Linux beta version of its highly praised sequel to Portal, and appropriately named this beast Portal 2. It was originally released in 2007, and Linux users now get to join in on the fun as Valve pushes more games our way.

Something important to remember is the fact that it is in beta, so there will be certain bugs and issues you may come across, but it is still well worth a look.



Portal 2 is a first-person puzzle game involving a special gun that opens portals to jump through and overcome obstacles. This sequel includes

a co-op mode too, so you can solve puzzles with your friends – we suggest playing the single-player first. <http://store.steampowered.com/app/620>

The Dark Mod

Did you just see something move? Nah, just my imagination...

Open-source developers are at it again! The Dark Mod has finally released a stable version and it is simply fantastic. The Dark Mod is inspired by the Thief series of games and plays a lot like the older games in the series. It was originally a mod for Doom 3, before it became a standalone game.

The game is all about stealth: you play a hooded ninja-like figure who is on a mission hunting valuables. You have to be careful though – the swordsmen of the land don't take too kindly to thieves.

The Dark Mod is set in a Gothic steampunk city and



looks simply beautiful. It is easily one of the best-looking free and open-source games around at the moment.

You can even create and download mission packs for

The Dark Mod, as it doubles as a friendly toolkit as well as a game, so be sure to share your favourite packs with us on the Linux Voice forum. www.thedarkmod.com/main

Wasteland 2

Massive robo-scorpions and other nasties await you.

RPG fans, get ready to be excited, as Wasteland 2 – one of the biggest crowdfunded projects ever, having raised over \$2.9m on Kickstarter from 61,290 backers – is making its way onto a computer near you. Wasteland 2 is also the project that helped to get a Linux export option in the Unity3D game toolkit, so the developers do of course plan a Linux version as well as OS X and Windows versions.

Wasteland 2 is a post-apocalyptic turn-based RPG game with beautiful graphics and solid-looking gameplay. We don't have many good RPG games like it, so it's a welcome addition to our game library. It is a direct sequel to the 1988 game Wasteland, which also has a Linux version now.

The Linux version of Wasteland 2 hasn't landed yet, but it is expected soon, as it has only recently become part of Steam's Early Access program, although if you do



The original Wasteland was released in 1988, and has since been re-released for Linux.

buy it now and consider it a pre-order of sorts you will get a copy of Wasteland 1 for free, which is quite the classic.

<http://wasteland.inxile-entertainment.com>

Oil Rush

Prepare your tactical warfare skills for when the oil runs out.



I hope you don't get sea-sick! Oil Rush may have been out for a while, but it's still not the best-known game. It's sad because it's actually quite brilliant.

Oil Rush is a mix of a real-time strategy and a tower defence game, set in a world ruined by nuclear war changing the planet forever. It was an early supporter of the Linux platform, and when it came out it was one of the most graphically intensive games we had (it is still easily in the top 10 in that respect).

There is no micro-management to be had; you don't even control the units directly during battle, so it's easy to get into, but hard to master.

<http://oilrush-game.com>

OpenXcom

An open source engine for one of the greatest games ever made.



OpenXcom is why we love open source. It has revived a true classic – UFO: Enemy Unknown, a strategy game from another era in computing, originally released in 1994 for DOS and Amiga.

You will need the game files from somewhere like gog.com, but it will be well worth the minor effort involved.

UFO: Enemy Unknown pits humans vs aliens in a mix between real-time strategy on a planetary view and turn-based tactical combat when you shoot a UFO down. You will need to capture and research the aliens while keeping the world happy with funding you raise from the nations of the world.

<http://openxcom.org>

ALSO RELEASED...



Paper Dungeons

Paper Dungeons is a brand-new mix of RPG, Rogue-like and a board game. It mixes dice throwing into combat, and it's a hard game to master. It features 155 levels, three different game-modes, five different classes and more content! It manages to keep things fresh with unlockable sets of dice and characters, and it also gives you the ability to create and download levels too.

www.desura.com/games/paper-dungeons



Stranded

You like pointing & clicking, right? Good. Stranded has just been released, promising to make you scratch your head while you wave your mouse around.

It has a nice sci-fi setting where you have wound up 'Stranded' on an unnamed planet with nothing to guide you but your own will. Join us in being confused – you know you want to!

www.petermoorhead.com/stranded



Natural Selection 2

Dust off those graphics cards: Natural Selection 2 is a seriously demanding hybrid of first person shooter and real time strategy games that pits humans vs aliens.

You can play as a traditional shooter or be a team commander, whereupon the game turns into a real time strategy in which you command other members of your team.

www.naturalselection2.com

LINUX VOICE YOUR LETTERS



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

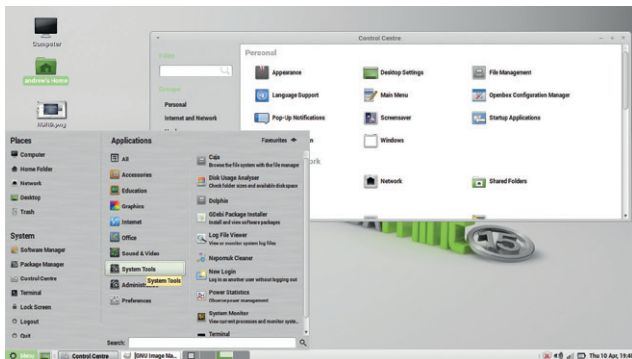
LINUX VOICE STAR LETTER

MIGRATION HERO

I found it interesting listening to various podcast discussions regarding installing Linux for other users of Windows and would like to share my experiences.

I have installed Linux on behalf of about 12 users in

an attempt to spread the good word. I usually install Mint (Cinnamon), Mint (Mate) or Xubuntu, all LTS versions, depending on the computer type, power of machine and if there are any installation difficulties.



Linux Mint (in Cinnamon or Mate) makes a great desktop system for users who just want to do the basics with their computer.

It is important to check with the user as to their expectations and requirements – Linux is not the best for syncing iTunes! I always put the programs I think the user will use on the desktop along with a ReadMe file detailing the Linux equivalents of the Windows software, how to find and install software, update the computer and details of where and how to look for help should they need it. I like to run through the setup with them so that they know how to connect to the internet etc, and this gives them the opportunity to ask any questions.

Once I have packed them off home, I have been surprised by the total lack of support that has been

requested, the only exception being the odd printing issue but once that's resolved I get nothing. I ask them how's it going only to be told that it's all fine and how much quicker it is now. It just does what they need. I get far more cries for help from Windows users than Linux ones.

My point is that should anybody be deciding whether or not to install Linux for a family member or friend and is worried that the user will need constant support my experience indicates the exact opposite, just make sure you do your homework first.

Charlie Ogier, Guernsey

Andrew says: As Roy Keane once said: "Failure to prepare is preparing to fail." You're completely right to ask what your users want out of their Linux boxes, and we salute you for your work.

I'M STICKING WITH YOU

I've just got a copy of Linux Voice issue 2. You are rightfully receiving many plaudits for the work you have done to set up this magazine, however, one thing seems to have been missed in the congratulations – it is the glue. You have managed to get a glue that firmly holds the DVD to the magazine, but when you take the DVD off, the remnants of the glue roll off nicely leaving no residue at all. This is a fantastic achievement, keep up the good work.

Charles Barnwel, Birmingham, UK

Andrew says: Like penicillin and WD-40, sometime the best things are discovered by accident. We can't claim that Mike has been working on a secret formula for the perfect glue; it's just what we were given by default by our distributors. Things do seem to be working out well so far, but if anyone has feedback about the magazine or the physical quality of the magazine please do get in touch.



When there's a great distro, we'll put it on the DVD; when there aren't any big releases, we won't bother with the DVD.

UN SOUND

I am by no means a newbie since I have been using Linux as my operating system of choice since 2000, but sound always seems to be a problem. Banshee and other players work OK and hide the difficulties from you but if you ever step outside the music player area it is a nightmare. Sound settings in PulseAudio and ALSA are arcane.

Despite this I thought I would try Ubuntu Studio as a dual boot option on my main PC. I assumed that the good folks at Canonical would have everything set up just so... I started Audacity and tried recording my voice. Selecting an input device proved to be the first problem. Audacity listed 26 input devices, and when I decided to Rescan Audio Devices the list of input devices increased to 36. I have no idea which of these options refer to the microphone I plugged into the sound card.

I can eliminate the USB options since I suspect that they refer to my webcam and I could try each of the other options till I find which one gets a response on the meter but life is short, especially when you get to my advanced years. And then I would have to do the same with the output devices.

I doubt if you can sort the problem but it would be nice to have some sort of guidance on



how to know which 'device' corresponds to which piece of hardware.

John Paton

Graham says: We have a lot of sympathy for your situation, John. One of our audio devices is a Focusrite Saffire Pro 40, and this lists 20 separate inputs and 20 separate outputs, with no indication of which channel does what or goes where. The only solution is trial and error.

The reason for this is lack of manufacturer support – many Linux drivers are created blind by developers trying their best to bring as wide hardware compatibility as possible. This is wonderful, but they often don't even have access to the hardware they're writing drivers for. What's needed, we think, is some way to create sensible default configurations for hardware. And perhaps that's something we, as a community, could help with.

Some audio apps, such as Bitwig, are fantastic – it's the underlying sound architecture in Linux that causes the problems.

GEDIT

I very much enjoyed the text editor comparison in the May issue. However, I have an addition and an objection.

The objection first: take a look at TextAdept. You are really missing a nice cross-platform editor. I use it on Linux and MacOS and I always have the installer ready in the event that I need to work on a Windows machine. It's open, free and completely extensible. What more do you want?

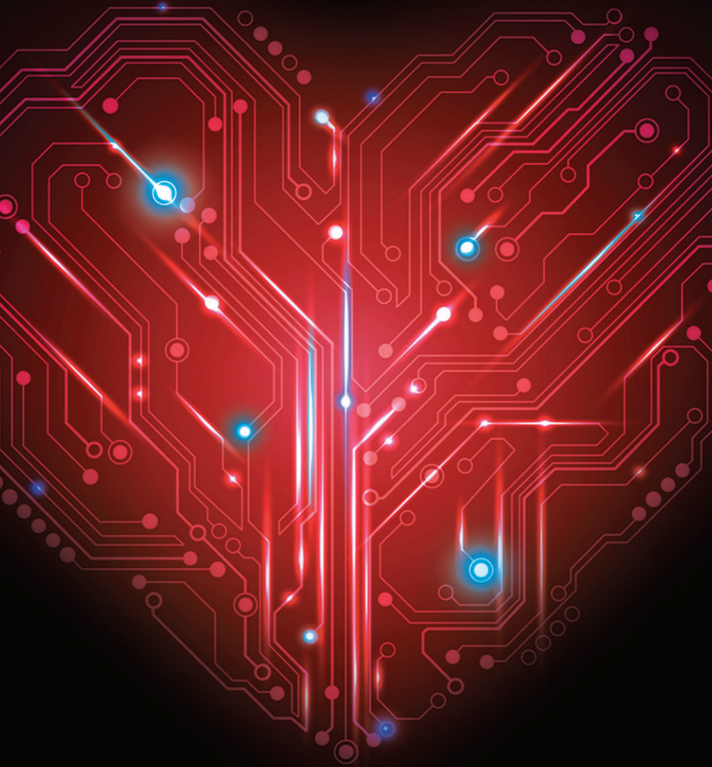
The addition: I really like Gedit. In fact, I used the text editor control to learn Python. I got a nice clone with additional features I missed. I was even thinking of embedding the Evince viewer in order to have a clone for TexWorks. It should not be that difficult, being as there is a lot of code floating around. That could be an idea for a Python programming tutorial that produces useful tools along the road...

Pedro A Aranda, Madrid

Ben says: Hi Pedro! We did include one cross-platform application in the shape of Sublime Text, which is a superbly polished piece of software and could easily have been declared the Group Test winner. As for Gedit, we like it too. We're just spoiled for choice when it comes to text editors.



WHEN IT'S CRITICAL...



WE WILL KEEP YOUR DATABASE ALIVE

2ndQuadrant's Platinum Production Support for open source PostgreSQL provides a guaranteed 15 minute (human) response, 4 hour workaround and guaranteed bug fix within 24 hours. 24 hours a day, 365 days a year.

We'll keep your business alive.



2ndQuadrant
Professional PostgreSQL

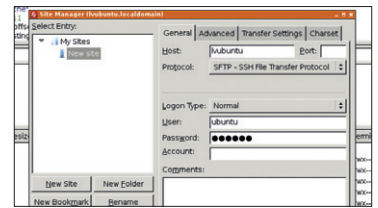
+44 (0)870 766 7756
2ndquadrant.com/support

PLAIN TEXT PASSWORDS

Congratulations on your second issue; you seem to have the contents about right for the greatest number of readers.

I would just like to mention that, although an excellent piece of software in many respects, Filezilla does have a potential security problem for some users. The login information that is entered into the Site Manager is stored in an XML file in `~/filezilla` as plain text, including any passwords. Probably not a problem for a home user, but something others may need to be aware of.

Chris Whelan



Security concern aside, FileZilla is a fantastic FTP tool.

Ben says: For home users, as you say, this isn't such a big deal, but it does serve as yet another reminder not to use the same password on multiple sites, because it only takes one to leak and you're compromised all over the place.

WHAT NAS BOX?

Having just read the article from issue 2 on "Filing Effective Bug Reports," I thought I'd comment on how you're striking quite a nice balance between newb-friendly content and more advanced stuff. The bug report article has probably assuaged many readers' fears on the topic and it might have done many open source projects quite a nice favour.

Speaking of newb-friendly content, I have a question: Over the couple of years that I've been messing around with Linux, I've seen a few articles on NAS, often on FreeBSD and how to set up an NAS device. For me, this begs that question, what is the difference between using, say, a 2TB box and putting NAS4Free on it, vice

putting a full-fledged distro like Debian on it? In both cases, can't the box be used for exactly the same thing, network storage? Wouldn't the difference just be in setting up the Debian distro as a file server?

Congratulations on the magazine, and keep the great content coming.

Roy Birk, Maryland, USA

Andrew says: Debian is an excellent OS for a NAS, and it's even used by some commercial products. NAS4Free is based on FreeBSD, so it's the best way to get the advanced ZFS filesystem. You can't go wrong with either – just make sure to uninstall any services you're not using, as each one is a potential vulnerability.

Debian running without a graphical server is a perfect choice for a network attached storage (NAS) box.



MANY IDEAS

You have mentioned in both magazines about running linux on MAC. Just wondering if you will throw a tutorial in for installing Linux on the late 2013 macbook pro(11,1) in an upcoming issue? I'd really like to get Linux up and running on my new laptop and use it as my main OS. Any help would be greatly appreciated.

Brian Meyers

Andrew says: I'm a big fan of the construction quality of Apple hardware, in particularly the battery life of the laptops. Linus Torvalds uses a Macbook running Fedora, Mike uses one running Xubuntu, and all the geeks at OSCON seem to have a shiny Apple device running some sort of Linux. It would be remiss of us not to try some sort of dual-booting tutorial, so look out for it soon.

MORE IDEAS!

It's amazing to look at the historical code tutorials and understand exactly what's going on. But after Ada Lovelace in LV001, Grace Hopper in LV002: who's next?

Brian Meyers

Andrew says: Alan Turing's next, that's who, but as the Bletchley Park stuff isn't all that hands-on we've moved him into the features section this issue; he'll be reinstated to the tutorials when Juliet gets onto the work he did around prime numbers. There's also Von Neumann and Konrad Zuse in the pipeline, so stay tuned.



If you've ever dabbled in Assembly language, Ada Lovelace's code for the Analytical Engine should look familiar – even though she wrote it in the 1840s.

PAST AGES

For the first time in my life I am writing to a magazine, which is mind blowing considering I've been collecting various mags since the late 80s when my parents opted to give me pocket money. While my friends bought sweets I saved mine and bought Metal Hammer Magazine or some other publication in that vein. but as I plod my way through my 30s I've started collecting magazines once more. I literally started buying them again last month with Linux Voice as previously I just used the internet for all my needs from the mid 2000s to now.

The last publication on Linux I bought was Linux Format back in September 2005. While I don't have that magazine I recently found the DVD that came with it in an old box of odds and sods that I found at my parents' place. I include a picture of the DVD cover for no real reason other than nostalgia. In short your magazine is so good it has rekindled my interest in Linux and got me buying magazines again.

I had an idea for a long-term tutorial that could be covered from issue to issue where you detail the building of a Linux distribution, using Linux From Scratch. Each month you dedicate a few pages (maybe 10 or so) to building the Linux Voice Distro (or whatever you want to call it). This could be really cool as a tutorial for everyone and get a lot of people



involved in creating the distro. It could be a real community project and the timing is good with the release of Ubuntu 14.04 imminent, so there's a nice fresh LTS Distro to fork from.

The tutorial could cover a lot of issues and really generate interest among the Linux Voice reader base, as learning how to build an OS from nothing is something I'm sure everyone would like to know how to do. I'm sure you would also see a jump in users on the forums.

Think about it. How cool would it be to put your own distro out on a future DVD release and also to see it on Distrowatch.com?

Well I hope you take the time to read this and would love if you ran with my idea and did it.

PJ

Ben says: That's an awesome idea, thanks PJ. We'll think of you when we've taken over the world and we're drinking fresh mango juice.

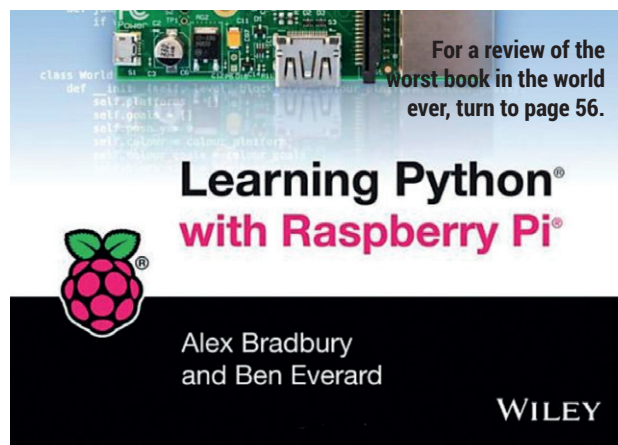
Ah, the Debian flying pig. A joke devised by our esteemed Mr Nick Veitch, who now dwells on page 114.

ILLITERACY

Issue 2 is great. But oh no – there's a gaffe on page 27!

The author of *A Computer Called LEO* is Georgina Ferry. However, on the front cover, at the bottom, there's a quote by a Brenda Maddox. Keep Ben off of the cider this month! I shall not be cancelling my subscription however as I consider it money well spent!

John Evans



LUGS ON TOUR

FLOSSUK Spring Conference

Josette Garcia likes to be beside the seaside; specifically, Brighton.

In 1783, the Prince Regent, played by Hugh Laurie in *Blackadder*, (later King George IV) made his first visit. He then spent much of his leisure time in the town and constructed the Royal Pavilion during the early part of his Regency.

So what's a bunch of Unix System administrators doing in Brighton? They did not come for the German sausage – they came to learn, network and find out different ways to improve their workload. It seems they are always looking out for more automation that will reduce human error.

Tutorials and talks

The conference started with a day of tutorials, covering LDAP for Linux Admins, Ansible, Learning Perl Together, and Caching and tuning fun for high scalability. There was also a full-day Google Workshop – Large-Scale System Design.

The wide selection of talks (covering three tracks over the two-day conference) covered topics

from configuration management and automation, security, performance improvements, as well as updates on the development of key free software projects.

The opening keynote was delivered by Paul Downey from Government Digital Services – “Make things open, it makes them better”. I feel this philosophy is not only for techies but is applicable to plenty of other situations.

Kris Buytaert presented two talks: ‘Building and Deploying MediaSalsa’ and ‘Continuous Delivery of your Infrastructure’. The premise of this was that software developers are adopting continuous delivery for their software, and infrastructure people can do the same.

Other subjects covered included the latest news from the PostgreSQL database project, a look at some of the new projects going on in Perl and the wonderfully named ‘Incident Response or When you find that you’ve lost your paddle and you are up the creek’. No conference can be without



the dinner, which was held at the Alfresco Bar and Restaurant on the seafront. After-dinner entertainment was provided by the LHS Bikeshed, members of the London Hackspace. The LHS Bikeshed is an interactive sci-fi space shuttle simulator – not for the faint hearted! You can find out more on lhsbished.tumblr.com.

Watch out for the videos on <http://flossuk.org>. The next FlossUK Spring Conference will be held in March 2015 in York, which I am told is a wonderful city!

Dammit Blackadder, Why is it that no matter how many millions of pairs of socks I buy, I never seem to have any?

Open-Source-Treffen

Mike Saunders strapped on his Lederhosen for a meetup in Munich.

Strictly speaking, the Open-Source-Treffen isn't a LUG, as it covers much more than just Linux. But like any good LUG, it meets up regularly (the last Friday of each month), everyone brings laptops to do some impromptu hacking, and there's plenty of beer to go round. It's an informal event held in a cafe near Munich's main train station – just a few minutes' walk away – and newbies are

welcome. Although most of the regulars have been using Linux and Free Software for many years, we had a good chinwag with some toe-dipping experimenters who were there to see how open source could benefit them.

In March's meetup, Maximilian Batz gave a great presentation (in German) about “the outlook for development of computers and users”. It started off looking at

general technological advances to come – eg more powerful hardware and better cryptography – and then focused on medical and social issues, such as artificial organs and implants to track people.

Although most of the discussion was in German, everyone was well-versed in English too. So if you're ever in that neck of the woods and fancy some FOSS banter, pop by: www.opensourcetreffen.de.

Digimakers Bristol

Ben Everard goes to Bristol's four-times-a-year children's geek-fest.

Every three months, Digimakers comes to @Bristol (the awkwardly named science exhibition centre). It's a day of geeky fun for all the family (actually, it's designed for children and teenagers, but in our experience, the adults enjoy it just as much). The basic premise is simple: the University of Bristol takes over a floor of the @Bristol centre and invites local techies to run sessions to help kids with tech. It's not exclusively Linux focused, but there's a strong Raspberry Pi focus to many of the events.

For the past two Digimakers, Linux Voice's Ben Everard has run sessions first on the Raspberry Pi camera module, and secondly on Scratch programming. There's a very hands-on approach, with sessions on Arduino robotics, making Lego Mindstorms dance,

shrimping, Sonic Pi... the list goes on and on. As well as teaching sessions, there are a few stands showing off some of the latest tech. The MagPi team are usually present (and usually have sweets), and earlier in the year, you could try out an Oculus Rift. Recent events have proved popular with local teachers as well who have been keen to brush up on the latest skills.

Get involved!

The next event is on 14 June 2014 (unfortunately, Linux Voice won't be attending this one). Keep up to date with what's happening on their Facebook page (www.facebook.com/digimakersbristol). You have to keep an eye on this, as spaces in the sessions are limited and require attendees to sign up beforehand. They usually fill up a few days before the event.



Ben taught 'Interactive cartoons with Scratch' at the March Digimakers to 54 fledgling Linux users.

Manchester Raspberry Jam

Jack Kelly reports on a rapidly growing success story.

This month's Manchester Raspberry Jam was a little different from the norm. Thanks to the University of Manchester's School of Computer Science – where I am an undergraduate student – we traded our usual chaotic assembly for a lab fully set up for 70 Raspberry Pi's and a lecture hall for talks.

We had our usual hack session; People of all ages gathered to make and share their creations in Scratch, Python and even Minecraft. We had Simon Walters showing off another set of new Scratch-powered robots, and Simon Duffy demonstrated various methods of home network hacking with Pi's. Pete Lomas from the Pi Foundation also attended, giving talks about the design of the Pi and the objectives of the Foundation, as well as answering questions from the event's attendees.

As always, we continue to be astounded by the projects that attendees bring to our Jams, as well as their willingness to share and help others. As an event organised primarily by myself, it's great to see people so enthusiastic about introducing people to the Raspberry Pi.

The challenge we face running the Jam from here is adapting to a changing range of attendants. While the Manchester Jam started as a simple user group, where enthusiasts could share their projects and ideas, more and more attendees turn up at Jams with intrigue alone – having never owned or even used a Pi before, wondering what all the fuss is about – and we're still figuring out how to deal with that in the best way. But we look forward to the challenge; our goal is to make sure everybody who attends the Jam



There must be something in the water in the North West of England – it's a hive of hacker activity.

gets the absolute most they can out of their Pi.

Anybody and everybody is welcome to a Raspberry Jam. If you're interested in attending, our Jam runs monthly throughout the year. To see when our next Jam will be held visit mcraspjam.org.uk or follow @McrRaspJam on Twitter.

I'VE GOT THE

POWER

UNLEASH THE HIDDEN STRENGTH OF THE LINUX COMMAND LINE

If you haven't mastered the command line, you're missing out on the most powerful features of Linux. **Mike Saunders** has tips galore for both newbies and old-timers...

"The command line can do certain jobs much more efficiently than GUI apps."

You know how in films, when they want to portray a computer genius/nerd/hacker at work, they always show someone tapping incomprehensible gobbledygook into a command line? Sometimes it'll be a green-on-black text terminal from the 1980s, accompanied by various beeping noises, just to add to the mystique. And thanks to stereotypes like these, many non-technical people assume that the command line is a weird and arcane tool, only to be used if there's no pointy-clicky GUI goodness at hand.

Now, as a Linux user, you already know

that that's nonsense, and the command line interface has its benefits. But have you really delved deeply into it? Have you discovered all its hidden tricks? And have you been able to ditch the mouse and start working more quickly? Over the next seven pages we'll show you how the command line interface (CLI) can do certain jobs much more efficiently than GUI applications, making your day-to-day Linux life smarter, easier and faster. Even if you've been using the CLI for a while, you'll find plenty of new gems in here, so let's get started.

Better file management

GUI file managers are clunky and slow. Here's how to work at light speed.

Dolphin, Nautilus, Thunar and co. are OK for simple drag-and-drop jobs, but in all honesty they don't compare to the CLI. As soon as you need to do something complicated, you end up with a horribly long workflow involving countless mouse clicks until your wrists get overloaded with RSI.

Let's take a complex job and see how it can be made simpler with some CLI magic. Even though you might not need this specific command on a day-to-day basis, you can break out the component parts and use them on your Linux travels in future, saving you heaps of time. So: imagine that you have a bunch of files without any extensions, and you don't know what's inside them. (You can see hundreds of these in Firefox's cache, for example.) They look like this:

```
3F7DFd01
E64C7d01
C42F9d01
F0887d01
...
```

Let's say you want to open the 10 biggest JPEG files in Gimp to have a look at them. Think about how you'd do that in your graphical file manager – if it's even possible. Providing that your file manager can peek inside the contents of files to determine their format, you might be able to click around and somehow sort the list by file format and size simultaneously (very few file managers can do that), and then click and drag to select the top 10, and right-click on the selection to open them in a program, then click down the list to find Gimp... Ugh.

Now check this command out:

```
file * | grep JPEG | sed s/:.*// | xargs ls -S | head -n10 | xargs gimp
```

It's a beast, isn't it? But actually, it's a bunch of smaller commands linked together, done in such a way that you can understand what each part does.

First, the **file *** part looks at every file in the current directory, and works out the filetype from the bytes contained inside. So you get

```
mike@mike-megabox: ~/images
C2B7Fd01: JPEG image data, JFIF standard 1.01
C2B7Fm01: raw G3 data, byte-padded
C7931m01: raw G3 data, byte-padded
C8DE5d01: JPEG image data, JFIF standard 1.01
C8DE5m01: raw G3 data, byte-padded
C99A9d01: gzip compressed data, from Unix
C99A9m01: raw G3 data, byte-padded
CBD2Fd01: JPEG image data, JFIF standard 1.01
D0488m01: raw G3 data, byte-padded
DB54Ad01: gzip compressed data, max compression
DB54Am01: raw G3 data, byte-padded
DBDAFd01: data
DBDAFm01: raw G3 data, byte-padded
DE504d01: HTML document, UTF-8 Unicode text, UTF-8 charset
DF382m01: raw G3 data, byte-padded
E64C7d01: JPEG image data, JFIF standard 1.01
EC359d01: PNG image data, 1 bit color, non-interlaced
F0887d01: gzip compressed data, max compression
F1A22m01: raw G3 data, byte-padded
F1CDBm01: Minix filesystem data, volume 01
FD615d01: ASCII text, with line numbers
FF52Dd01: gzip compressed data, max compression
mike@mike-megabox:~/images$ file * | grep JPEG | sed s/:.*// | xargs ls -S | head -n10 | xargs gimp
DBDAFd01
EEA2d01
90A57d01
B0584d01
FD615d01
1DECf01
8E410d01
75EDDd01
36C7Ed01
EC359d01
mike@mike-megabox:~/images$
```

Using pipes and multiple commands, you can narrow down to just the filenames you need, all without hundreds of tedious mouse clicks.

lines like this:

```
CBD2Fd01: JPEG image data, JFIF standard 1.01
D0488m01: raw G3 data, byte-padded
DB54Ad01: gzip compressed data, max compression
```

We only want JPEG files, so we take the output from the **file *** command via a pipe (**|**), then **grep** to just retrieve lines containing **JPEG**. After this point we don't need any information other than the filename, so we pipe the text to **sed**, the stream editor, which does a replacement. It takes a colon **:** followed by any sequence of characters **(.*)** and replaces it with nothing **//** – ie nothing between the slashes). So it gets rid of everything but the filenames.

Then, using **xargs ls**, we bundle together all the filenames we've got so far and list them, sorting by size **(-S)**. The **head** part retrieves the top 10 items of the list, and then using **xargs** again, we bundle up all the filenames into a single string and tell Gimp to open them.

It might take a few re-reads to really grok all this, but once you have your head around

it, you can see how powerful the CLI is for working with files. (For instance, you could replace **xargs gimp** with **xargs rm** to delete those 10 biggest JPEG files.) Try adding your own components to the command, and making new ones using parts of it.

Essential tips

If you're new to the command line, here are some things you absolutely need to know. In most Linux distros, the CLI is accessible in your desktop's program menu as Terminal, XTerm or Konsole.

- **ls/cd/rm/mv** The most common commands (list files, change directory, remove file and move/rename file). Each command has a manual page (eg **man ls** – hit **Q** to quit the viewer). Many commands have extra options; for example, **ls -la** lists all files, including hidden ones, with details.
- **Tab** Hit the Tab key to automatically complete a filename or directory. If you want to delete **foobarlongfile.txt**, for instance, enter **rm foo** and hit Tab, and it should be completed.
- **History** Use the up and down cursor keys to navigate through previous commands. You can edit them as well.
- **~** Your home directory (eg **/home/bob/**)
- **>** and **>>** sends output of a command to a file, overwriting (**>**) or appending (**>>**). Eg **ls -l > list.txt**.

“Once you have your head around it, you can see how powerful the CLI is for working with files.”

Better editing

Learning a good text editor on the command line is essential.

We can't stress enough how important it is to learn a good text editor. It really makes a vast difference to how you work – even if you're not a programmer. Some GUI text editors are well-specced with plenty of features, but when you're working with plain text, why should you keep moving your hands away from the keyboard to grab the mouse?

The two most notable text editors are Emacs and Vim. They both have their strengths and weaknesses, but we'll focus on the latter here because it's installed in high-on every Linux distro by default.

This isn't a guide to the basics – we did that in issue 1. If you don't have that issue and you've never used Vim before, see the "Micro guide" box and then do the **vimtutor**. Here we'll explain why it's well worth learning, and if you're a regular Vim user, we'll show you some tricks that you might not have come across.

How to love Vim

Many people try Vim and come away frustrated, because they don't spend time getting into the right mindset. Some people use it often but never learn to enjoy it. That's fair enough – it's not a very welcoming program. But bear these in mind and you'll learn to love it:

- Always switch back command mode (with Esc) straight after editing. Make command mode your default mode. Vim is a modal editor, which means sometimes you're editing text, and sometimes you're giving commands. You should only be in insert mode when you're editing text, so always hit Esc as soon as you're done. You'll learn commands better this way, instead of accidentally adding them to your text.

- Use the H, J, K and L keys to navigate. These are on the home row, ie under your fingers, so you don't have to move your hands down to the cursor keys. They really help you work faster – although it might take a few days to get used to them.

- Treat commands as a language. At first,

```

219
220 Returns:
221     The number of zero bits on the right hand side of the number.
222
223 """
224 if number == 0:
225     return bits
226 for i in range(bits):
227     if (number >> i) & 1:
228         return i
229 # All bits of interest were zero, even if there are more in the numb
er
230 return bits
231
232
233 def summarize_address_range(first, last):
234     """Summarize a network range given the first and last IP addresses.
235
236     Example:
237     >>> list(summarize_address_range(IPv4Address('192.0.2.0'),
238     ...                               IPv4Address('192.0.2.130')))
239     ...                               #doctest: +NORMALIZE_WHITESPA
CE
ipaddress.py [+][RO] 234,34 10%

```

A well-tuned `~/.vimrc` file makes Vim more attractive, informative and welcoming.

Vim's commands look weird and cryptic, but when you piece them together they make more sense. For instance, in Vim-speak **d** is delete, **a** means around an object, and **P** refers to a paragraph. Hit **dap** and voila: delete text around a paragraph (that is, text inside the paragraph and trailing spaces).

It's also worth customising the `.vimrc` file in your home directory to make the editor a bit friendlier. Here's what we have:

```
set number ruler laststatus=2 hlsearch ignorecase
title
```

syntax on

This adds: line numbering; a ruler showing the current line number; a status line with the filename; highlighted searches; case-insensitive searches; more information in the terminal window title; and syntax highlighting for various programming languages.

Advanced Vim tricks

Vim is chock-full of keyboard shortcuts and commands that make life easier. Want to search for the next instance of the word under the cursor? Hit ***** (asterisk). Doing

some programming, and want to find a matching bracket or brace? Move the cursor over the bracket and hit **%**. Want to quickly go back to the last place you entered text? Use **gi**.

Earlier we mentioned using **dap** to delete a whole paragraph. This is an example of using a text object, and this is where Vim is ridiculously powerful. For instance, **das** deletes a sentence, while **ci"** (C-I-double quotes) changes text inside a pair of double quotes. Say you have this text:

```

```

To change the filename: move the cursor anywhere inside the quotes, and hit **ci"**. Vim will remove all text inside the quotes and place you in insert mode, so you can type in some new text and hit Esc when you're done. And this opens up possibilities for the equally awesome **.** (dot) command.

Basically, **.** repeats the last text editing action – both the command(s) you used in Vim and the text you typed. So if you move the cursor into another **<img src=...** line, between the quotes, and hit **.** then the text will be replaced again, exactly like the first time.

This is tremendously useful if you need to do a lot of quick replacements: do the command once, then jump around to other places and tap **.** where necessary. Because **.**

"Vim is chock-full of keyboard shortcuts and commands that make life easier."

includes a whole text action, it can even be used to repeat editing operations with backspaces inside.

Imagine you have some function prototypes copied from a header file:

```
int foo(int a, int b);
void bar(char *d);
void baz(int a, bool d);
```

Now you want to implement the functions themselves. Go to the first line (**int foo**) and tap A (capital) to append text onto the end of the line. Hit Backspace to remove the semi-colon, Enter (for a newline), {, Enter again, }, and Enter once more. Then hit Esc to get back to command mode. Now the first line has changed into this:

```
int foo(int a, int b)
{
}
```

So, we've converted a prototype into a proper function. Now move the cursor to the second prototype line (void bar), hit . and *voilà*, it is also converted, using the exact same editing action as before. You can then hit . again to convert the third line. It's a massive time saver.

Globalisation

Another hugely powerful (and not well known) command is **:g** – the global command. Take this for instance:

```
:g/someword/m0
```

This takes all lines containing the word **someword** and moves them to line zero, ie the top of the file. Or you could use **:g/someword/d** to delete all lines containing **someword**.

An especially useful add-on option for **:g** is **norm**, which puts Vim into normal

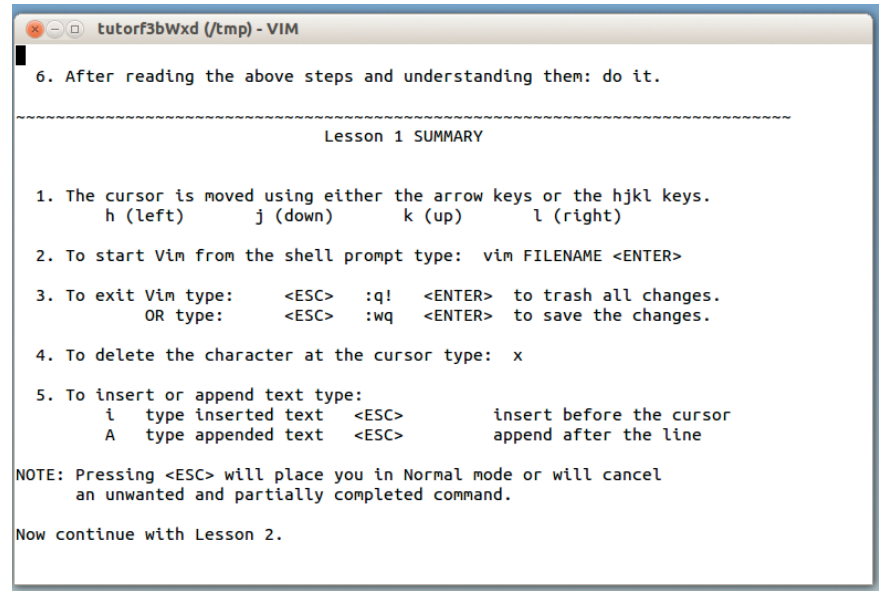
New to Vim? Here's a micro guide

Enter **vim newfile.txt** to edit a new file. Hit **I** and you'll see -- **INSERT** -- at the bottom, which means you're in the Vim mode for adding text. Type in a few lines. When you're done, hit Esc to return to command mode.

Use the **H/J/K/L** keys to move around. Hit **X** to delete a character under the cursor and **DD** to delete a line. Use **0** (zero) to go to the start of a line, and **\$** to jump to the end. Type a number

and press **Shift+G** to go to that line. Hit **Ctrl+G** to view the current line number and **U** to undo an operation.

To save, make sure you're in command mode (hit Esc to be sure) and type **:W**. To quit, use **:Q**. To quit without saving, **:Q!**. Those are the basics – now enter **vimtutor** and follow the more detailed guide, which will take about 20 minutes. Then you'll be ready to use the tips here.



Although the Vimtutor doesn't make you an expert in Vim, it gets you well-versed with the basics of this powerful, flexible text editor (and its many offshoots).

(command) mode, and then executes the commands as written. For instance, say you have some Python code and you want to comment out all lines containing **DEBUG** by putting hash marks at the start:

```
:g/DEBUG/norm 0i#
```

Here, for each line containing **DEBUG**, Vim executes **0i#** – that is, go to the start of the line, switch to insert mode, and add a hash. How cool is that? And then you can even add Esc keystrokes when entering a **:g** command by tapping **Ctrl+V** and then Esc.

Save time and energy

Imagine you want to add C-like comments to **DEBUG** lines, so that:

```
printf("DEBUG: Blah blah");
```

becomes:

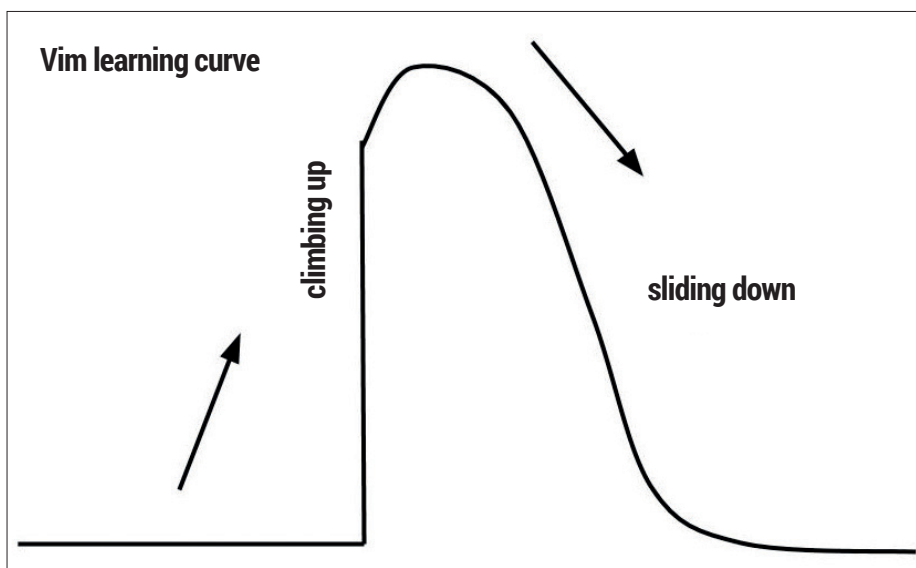
```
/* printf("DEBUG: Blah blah"); */
```

Use this:

```
:g/DEBUG/norm 0i/* ^[A */
```

(Again, use **Ctrl+V** followed by Esc to input the **^[** [escape character here.] This goes to the start of the line, inserts **/*** followed by a space, then Escapes back out of insert mode, goes to the end, and adds ***/**.

Just like we said – ridiculously powerful. And more fun than using dreaded regular expressions. (PS: Turn to the inside back cover of this very magazine for a great cheat-sheet for Vim commands!)



Vim's learning curve, jocularly depicted at <http://tinyurl.com/nfbj3mv> ("Why I use Vim" by Pascal Precht). There's a huge amount to learn at the start, but it all makes sense with time.

Better image editing

Huh? The CLI is good at editing pictures? Surely you can't be serious...

I am serious, and don't call me Shirley. Yes, there are many cases where it's easier and faster to edit images at the command line, rather than doing them in pointy-click fashion via a graphical tool like Gimp. This is especially true if you want to perform editing or processing operations on multiple files at the same time – in other words, batch processing.

The suite of programs we're using here is ImageMagick, which you've probably heard of if you've been on the Linux scene for a while, because it has been in development since the early 90s. ImageMagick is monumentally versatile and supports over 100 different file formats – so it will handle nigh-on anything you throw at it.

The most commonly used tool in ImageMagick is **convert**, which works like this example command:

```
convert image.png image.jpg
```

Pretty simple, right? This just makes a JPEG

version of the PNG file. Of course, when you're generating JPEGs you'll often want to alter the quality:

```
convert -quality 90 image.png image.jpg
```

Resizing is possible as well. The first command here specifies a percentage of the original size, while the second uses exact dimensions:

```
convert -resize 75% image.png image.jpg
```

```
convert -resize 300x300 image.png image.jpg
```

Don't be so square

Something interesting happens with the second command, and it's to do with aspect ratios. If the source image isn't a square, the resulting image will be 300 pixels wide and however many pixels tall to match the original aspect ratio. If you want to force the image to be 300 x 300 pixels and ignore the aspect ratio, add **blackslash-exclamation** to the dimensions, like so:

```
convert -resize 300x300\! image.png image.jpg
```

Along with file format conversions and resizing, another common job is to crop an image – that is, only save a portion of the original. This is fairly straightforward too:

```
convert -crop 250x100+20+40 image.png image.jpg
```

This takes a 250-pixel-wide and 100-pixel-high chunk of the original picture, from 20 pixels across and 40 pixels down, and saves it into **image.jpg**. For crop operations you might not want to convert the file – instead, you just want to overwrite the original. You can do this by changing the **convert** command to **mogrify** and omitting the destination file:

```
mogrify -crop 250x100+20+40 image.png
```

Another useful option is **rotate**, which takes the amount of degrees (clockwise):

```
mogrify -rotate 90 image.png
```

Lightning fast batch jobs

So far so good, but these commands aren't much quicker than doing the same jobs in a graphical editor. But! When we add some command line scripting into the mix, it all becomes a lot more efficient. Say you have 200 **.png** files in the current directory, and you want to shrink them all to 50% of their original sizes:

```
for f in *.png; do mogrify -resize 50% $f; done
```

Here we create a loop, saying that for every file in the current directory that has a **.png** extension, we perform a **mogrify** operation on that file (the filename is contained within the **\$f** variable).

What about if you want to convert all the PNGs to a different format? You could do this:

```
for f in *.png; do convert $f $f.jpg; done
```

But the resulting filenames are a bit ugly here – **foo.png** becomes **foo.png.jpg**, **blah.png** becomes **blah.png.jpg**, and so forth.

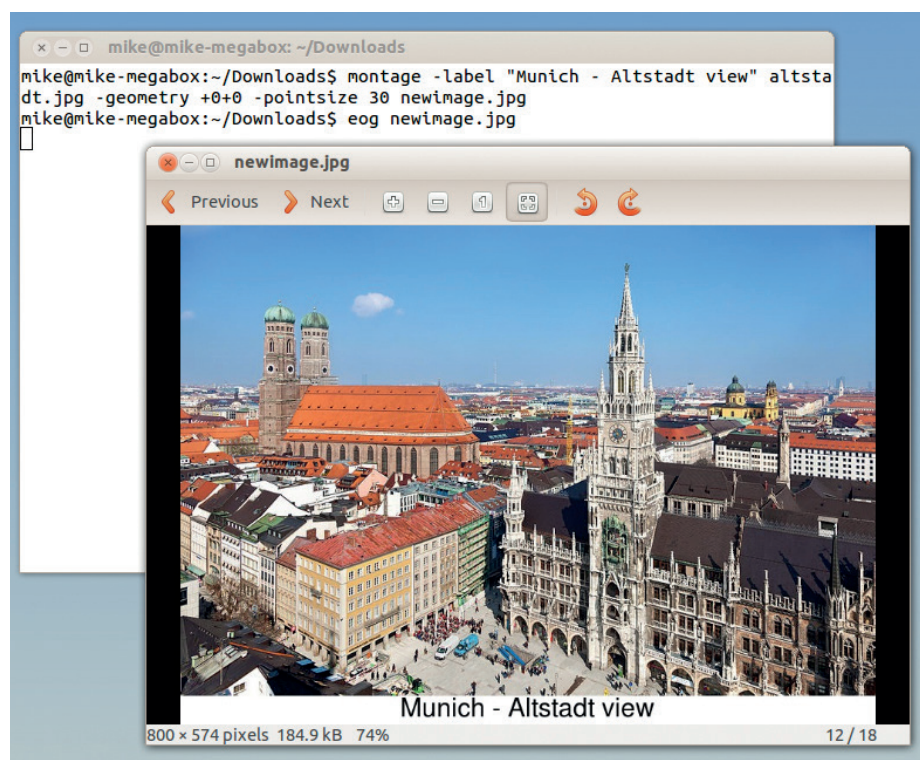
However, using a command line trick called parameter substitution, we can remove the **.png** from the destination filenames:

```
for f in *.png; do convert $f ${f%.png}.jpg; done
```

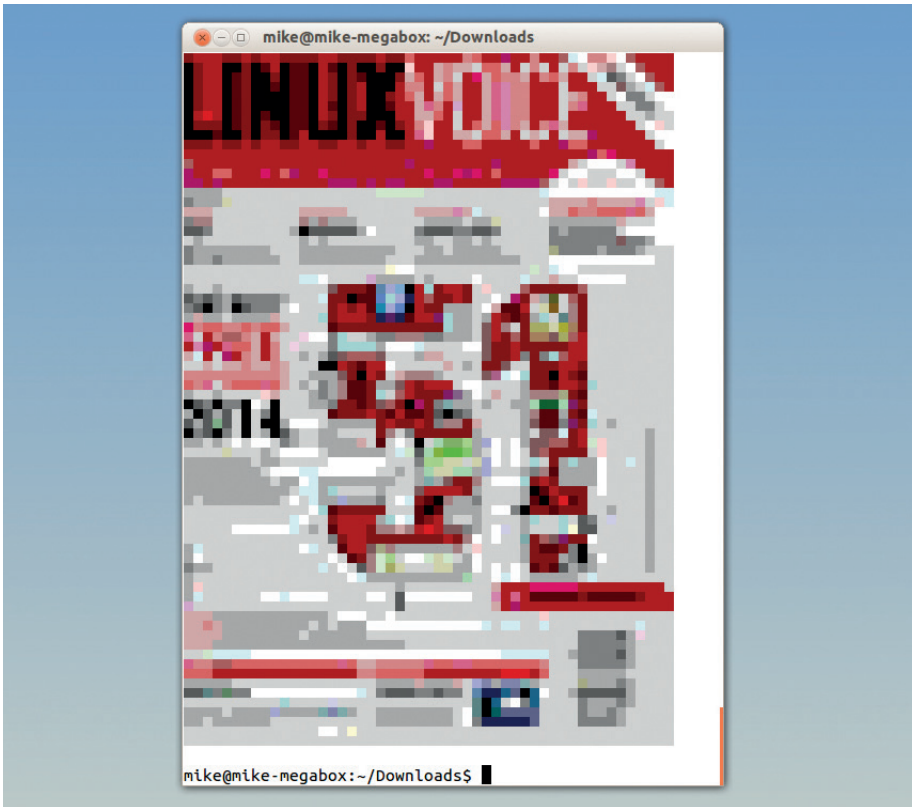
Here, the **\${f%.png}.jpg** bit does the clever work, removing **.png** and then adding **.jpg** on to the filename stem. (You can also use **mogrify** to convert images and replace their extensions, but it's worth knowing these tricks for the future.)

So, with the **convert** tool and some command line scripting, you can do

“Image Magick is monumentally versatile and supports over 100 different file formats.”



Convert, resize, flip, crop and add captions to hundreds of images in seconds, thanks to ImageMagick.



Use shellpic (<https://github.com/larsjsol/shellpic>) to view images in the terminal – handy if you’re SSHed into a remote server and want a quick preview of an image file.

conversion, resizing and cropping jobs on hundreds of images in a matter of seconds. If you had to do all the alterations by hand, it’d take hours or even days. ImageMagick has more cunning features though, so let’s take a closer look.

If you’re working with batches of photos, you’ll often need to correct their brightness and contrast settings. The **convert** and

mogrify tools have an option for this:

mogrify -brightness-contrast 20x-30 image.jpg

This improves the brightness of the image by 20%, and reduces the contrast by 30%. Again, you could include this **mogrify** command in a ‘for’ loop as discussed earlier, to fix hundreds of images at once.

ImageMagick is packed full of filters, such as blurring:

mogrify -blur 5x2 image.jpg

The first number here is the radius, while the second is the sigma (the actual amount of blurring). Try playing around with different values. You may not think it, but you can even turn pictures into charcoal drawings with a single command:

mogrify -charcoal 5 image.jpg

Another tool included in ImageMagick is **montage**, which creates a single image from a bunch of images. It’s also useful for adding captions onto images, like so:

montage -label “My caption” image.jpg -geometry +0+0 -pointsize 30 newimage.jpg

This adds the words “My caption” in 30 point font to the bottom of the picture, without resizing the picture (hence the +0+0), and writes out the result to **newimage.jpg**.

Command-line line drawing

One of ImageMagick’s most powerful features is its set of drawing commands. You can add all kinds of shapes to images via the command line, which is also useful when you’re doing batch processing jobs and want to add labels or diagrams to individual images. Take a look at this simple example:

mogrify -fill white -stroke black -draw “rectangle 30,10 200,100” file.png

This creates a white rectangle with a 1-pixel black border, 200 pixels wide and 100 pixels tall, and places it at 30 pixels across and 10 pixels down on **file.png**. Many other options are available for drawing circles, polygons and Bézier curves – see the full list at www.imagemagick.org/script/command-line-options.php.

Better calculating

Doing calculations at the command line makes much more sense than clicking loads of little buttons, over and over and over. With Qalc, part of the Qalculate suite (which also includes GUI tools) you can do some very funky stuff. On Debian/Ubuntu/Mint-based systems, grab the command line tool with **sudo apt-get install qalc**. The program’s manual page is disappointingly small, and there’s little else in the way of documentation, so the best way to learn it is via examples. Like so:

qalc “((78*30)+(13*19))/2”

Fair enough, that’s a normal calculation. But Qalc is capable of a lot more:

qalc “addDays(2014-06-18, 50)”

This asks Qalc to perform its internal **addDays** routine - you can guess what that does. In our case, we tell it to add 50 days onto the 18th of June, and it spits out the result:

addDays(“2014-06-18”, 50) = “2014-08-07”

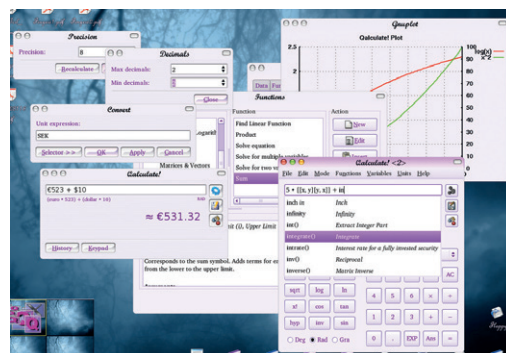
When you first run Qalc it downloads exchange rate information from the internet, so you can do:

qalc “500 EUR to GBP”

The program also understands lots of other units and conversions:

qalc “1300 feet to metres”

qalc “70 mph to kmh”



Qalc is part of Qalculate, a bigger suite of tools that include fancy GUI front-ends.

It’s especially useful for doing bandwidth calculations:

qalc “10Gibyte / 300(Kibyte/second) to hours”

This tells us how many hours it will take to download 10GB at 300k/sec. Qalc’s data files are stored in the **/usr/share/qalculate/** directory, so it’s well worth having a nosey in there to see what other units are supported. You can even do calculations with planets and atomic elements...

Better email

“All mail clients suck. This one just sucks less.” This is the motto for Mutt...

Given that most emails are plain text, you don't lose much by switching from a GUI to a CLI mail client. And indeed you gain a lot more, especially if you choose a client like Mutt. Like many of the programs we've covered in this feature, Mutt has been around for most of Linux's history – its first release was in 1995. And although it might look old-fashioned and complicated, in the right hands it's a superb program, and it's available in almost every distribution's package repositories.

Before starting Mutt for the first time, you'll need to create a `.muttrc` file in your home directory. This contains the program's settings, and an example for connecting to an IMAP server (with SMTP for sending) is:

```
set spoolfile="imaps://user:password@server.com/Inbox"
set folder="imaps://server.com/Inbox"
set smtp_url="smtp://user:password@server.com:25"
set ssl_starttls=yes
set from="name@domain.com"
set use_from=yes
set record="=Sent"
set postponed="=Drafts"
```

Change `user`, `server.com`, `name` and `domain.com` here to match your mail server settings. If you access your mail via POP3, see the relevant section of the Mutt documentation at <http://tinyurl.com/64j7tzip>.

Now enter `mutt` to start the program, and it'll retrieve the headers for your emails. Right away you can see that Mutt does a decent job given the limits of text mode: it uses colours and highlighting effectively, and even displays threaded conversations via red arrow symbols.

To select a message, use the up and down cursor keys (or J and K in proper Vi fashion) and then hit Enter. The mail contents will be displayed – hit D to delete the mail, R to reply, and I to go back to the message list. Use / (forward slash) and enter a word to search for a mail, and N to repeat the search. In the main list view, tapping Q quits the program and returns you to the command line. And in most views,

“Mutt has loads of keyboard shortcuts, so you don't have to mess around with the mouse.”

```
mike@mike-megabox: ~
q:Quit d:Del u:Undel s:Save m:Mail r:Reply g:Group ?:Help
1101 r T Mar 26 Andrew ( 93K) Latest EPOS
1102 C Mar 26 ben (2.0K)
1103 C Mar 26 Graham Morrison (3.0K)
1104 T Mar 27 Andrew Gregory (2.1K)
1105 + Mar 27 stacey (1.1K)
1106 + Mar 27 Andrew (1.2K)
1107 r + Mar 27 stacey (1.1K)
1108 + Mar 27 stacey (1.6K)
1109 r T Mar 27 wolf2525@gmx.de (1.7K) Linux Voice #1 received: still a question
1110 + Mar 27 wolf2525@gmx.de (1.3K)
1111 + Mar 30 Graham Morrison (2.5K) Re: Single issue 1s - next batch
1112 T Mar 31 Graham Morrison (1.3K) podcast recording
1113 C Mar 31 Andrew (1.6K)
1114 C Apr 01 ben (1.6K)
1115 T Mar 31 Graham Morrison (40K) Fwd: Article proposition?
1116 T Mar 31 Graham Morrison (5.2K) Fwd: best magazine EVER!!! (And volunteer writer)
1117 T Apr 01 Andrew Gregory ( 68K) Fwd: Linux Voice Ltd
1118 T Apr 01 Andrew Gregory ( 18K) Fwd: Sainsbury - Issue 3
1119 C Apr 01 Graham Morrison (2.8K)
1120 C Apr 01 Andrew Gregory (8.1K)
1121 + Apr 01 RaspberryPi (7.1K) Pi Store - Customer Invoice
1122 T Apr 01 Andrew (1.0K) Sysadmin
--Mutt: imaps://mike@mail.linuxvoice.com/Inbox [Msgs:1129 Old:4 125M]---(threads/date)--(99%)
```

The Mutt email client makes decent use of colour in the terminal, and like everything in this venerable application, these colours are highly configurable.

you'll see keyboard shortcuts displayed at the top of the screen.

Interestingly, Mutt doesn't include its own editor; instead, it uses one already installed on your system. So if you reply to a mail (or hit m in the message list to create a new mail), you'll be thrown into Vim by default. But you can change the editor in your `~/.muttrc` like so:

```
set editor="nano"
```

(Or you could change that to “`emacs`”, or even “`gedit`” if you need some GUI love.)

Macho macros

So Mutt is great: it's lightning fast, looks good, and has loads of keyboard shortcuts so you don't have to mess around with the mouse. But it has some brilliant advanced features too. Instead of using / to search, hit L and then type a word. This is the “limit” command, and it narrows down the displayed messages to match your specifications. You can set some very specific limits:

```
~N|~d<7d
```

This tells Mutt to display only new messages (`~N`) or messages less than 7 days old (`~d<7d`). The pipe (|) character is used in the middle to create the “or” part. To switch back to the full message list, hit L and then type all. (Mutt's documentation has a detailed list of all the options – see <http://tinyurl.com/yzwbrur>.)

Additionally, Mutt has excellent support for macros – that is, pre-determined sequences of actions. For instance, in the message composition view, after you've entered the text in your editor and Mutt is asking if you're ready to send, you can hit the A key to attach a file. Enter a filename, hit Enter, and the file will be attached. But you could create a macro for this in your `.muttrc`:

```
macro compose \cb '<attach-file>file.txt<enter>'
```

This means: in the compose view, if the user hits Ctrl+B, the `attach-file` command will be executed. The word `file.txt` is inserted automatically, and a virtual Enter key is pressed. So Ctrl+B now does the whole action at once – useful if you frequently attach the same file to a message.

This is just one example; Mutt supports hundreds of functions that you can use in your macros, and really speed up your day-to-day work. See <http://tinyurl.com/677feer> for the full list.

Better administration

Keep tabs on your Linux boxes, wherever in the world they are.

It goes without saying that the command line is the best way to administer a Linux box. Sure, there are some decent GUI tools, but if you're working with mail, web or database servers, chances are they don't have anything graphical installed and you're logged in via SSH. Or even on your desktop Linux box, if X goes down you'll need some way to fix and monitor things.

Slurm (<https://github.com/matthias/slurm>) is a great little network bandwidth monitor. Start it by providing the name of a network interface, eg:

```
slurm -i eth1
```

If you don't know the name of the network interface(s) on your Linux box, enter `ifconfig` for a list. Slurm displays textual information about the current data send and receive rates, along with the total number of transmitted packets and megabytes. It also shows a colourful graph of bandwidth using ASCII characters – so if you're administering multiple machines, you can leave it running in an SSH session on one, and quickly check it to see if it's being maxed out.

Monitor machine activity

Htop (<http://htop.sf.net>), meanwhile, is a souped-up version of the `top` utility. Like `top`, it displays information about currently running processes, but with much more flair and interactivity. As the program is running, hit F4 to filter processes based on name – or hit F5 to switch to a tree view, so you can see which processes were launched by other ones.

A series of bar charts at the top shows the current usage of your RAM banks and CPU cores, and you can hit F2 to configure various settings in the program. Once you've

```

root@mike-megabox:~# htop
  1  [ ]  0.7%  Tasks: 120, 268 thr: 1 running
  2  [ ]  2.0%  load average: 0.20, 0.20, 0.28
  3  [ ]  3.4%  mem: 01:46:31
  4  [ ]  0.7%
  mem:||||| 711/70203
  swap: 0/80580

2175 mike  20  0.155M 81792 1460 5 2.0 1.0 2:39.52 complz
6152 root   20  0.012M 228 1552 2.0 0.0 0:03.46 htop
6363 mike  20  0.088M 16072 1464 5 0.7 0.2 0:00.41 gnome-screenshot
2154 mike  20  0.172M 2326 1218 5 0.7 0.2 0:18.33 gnome-terminal
2538 mike  20  0.104M 9768 2316 5 0.7 1.2 0:01.58 /usr/lib/libreoff
2149 mike  20  0.174M 2790 2152 5 0.7 0.0 0:00.04 /usr/lib/dconf/dconf
2164 mike  20  0.295M 2284 8116 5 0.0 0.2 0:03.05 /usr/bin/gtk-wineid
1368 www-data 20  0.364M 2376 772 5 0.0 0.0 0:02.48 /usr/sbin/apache2
2154 mike  20  0.023M 936 764 5 0.0 0.0 0:03.71 syndaemon -l 1.0
4139 mike  20  0.555M 1550 1648 5 0.0 0.1 0:00.33 /usr/lib/xdm/xdm-l
1951 mike  20  0.553M 2556 1516 5 0.0 0.4 0:09.66 /usr/lib/unity/unity
1803 mike  20  0.044M 880 832 5 0.0 0.0 0:02.58 dbus-daemon --for
1892 mike  20  0.251M 1548 692 5 0.0 0.2 0:05.02 /usr/lib/xb8_04-l
2182 mike  20  0.156M 81792 1460 5 0.0 1.0 0:02.02 complz
1901 mike  20  0.346M 320 1628 5 0.0 0.1 1:00.43 /usr/lib/ibus/dan
f1m1f p2 3000  F3000F4 4110F5300 F6000F7 7100F8p8000 F94110F10

```

Htop is a process monitor like the standard `'top'` command, but literally a jillion times better.

```

mike@mike-megabox: ~
== slurm 0.4.0 on mlke-megabox ==

          X
          X XX
          XXXXX
X          XXXXXX
XXX  XXX  XX  XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXX  XXX
XXX  XXX  XXX XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXX  XXX
          XXX  XX
          X  XX
          X
          X
          X

Active Interface: eth1
Interface Speed: unknown

Current RX Speed: 304.19 KB/s
Graph Top RX Speed: 538.05 KB/s
Overall Top RX Speed: 538.05 KB/s
Received Packets: 57264
MBytes Received: 62.510 MB
Errors on Receiving: 0

Current TX Speed: 7.52 KB/s
Graph Top TX Speed: 30.49 KB/s
Overall Top TX Speed: 30.49 KB/s
Transmitted Packets: 45047
MBytes Transmitted: 6.065 MB
Errors on Transmission: 0

```

A sprinkling of ASCII art provides an at-a-glance overview of network activity in Slurm.

tried it, you'll never go back to using plain old `top` again.

After all of the command line goodness of the last seven pages, wouldn't it be great if you could record your favourite tricks and share them with others? You could use some screen recording software and upload the results to YouTube, but a more elegant solution is Asciiinema (www.asciinema.org). You can get it on Debian/Ubuntu like so:

```
sudo apt-get install python-pip
```

```
sudo pip install --upgrade asciinema
```

(The package might have another name than `python-pip` in other distros.) Now enter `asciinema rec`, do some work, and type `exit` when you're done. Asciiinema will offer to automatically upload the recording of your session to its website, and provide you with an URL you can then share with others. For instance, here's a recording of us demonstrating the mighty power of Figlet:

<http://asciinema.org/a/8746>

Better downloads

The download dialogs included in web browsers are very limited, and although more featureful standalone GUI alternatives exist, sometimes it's best to go straight to the CLI. Aria2 is arguably the best command line download manager in existence, supporting a gigantic range of features and options. For instance, say you want to grab an ISO image that's hosted on two servers, and they're both rather slow:

```
aria2c -s2 http://foo.com/blah.iso http://another.com/blah.iso
```

Here Aria2 downloads one half of the file from `foo.com`, and the other half from `another.com`, simultaneously, so you get the file much more quickly than you would using a single connection to one server.

It's possible to limit download speeds, so adding `--max-download-limit=100K` to the command line will restrict Aria2 to using 100KB/second of your bandwidth. And you can even tell it to give up if a connection becomes too slow:

```
--lowest-speed-limit=10K
```

(So if the bandwidth drops to less than 10KB/sec, Aria2 quits.) Other useful options include `--on-download-complete=command`, which automatically performs a command after a file has been downloaded. There's also the `--on-download-error` argument, which is handy for dealing with connection failures.

See Aria2's website at <http://aria2.sf.net> for the full documentation – it's immensely powerful when you include it in Bash scripts.



Can you buy a Linux computer on the high street? Ben Everard investigates.

Fifteen years ago, buying a Linux computer was a challenge. Back then, few people had heard of it, and even fewer knew what installing it entailed. The web was still in its youth, internet shopping wasn't as popular as it now is, Linux still came in boxed sets and hardware support was patchy.

There hasn't been a single watershed moment, but things have gradually improved. We went undercover to find out how much things have changed.

Since the world isn't yet fully digital, we started by visiting the local shopping complex to see if we could

get any help there. The only major computer chain left in the UK is PC World, so we wandered in and started poking around at PCs until one of the sales staff approached us.

"Hello there, do you need any help?"

"Yes, I was wondering if you had any machines that ran Linux."

"Oh, um, hang on, I'll just go and ask someone."

At this point, the sales assistant (a young man in his mid twenties) scampered off to the support area where a group of similarly aged young men poked around at the innards of computers. A few minutes

"We wandered into PC World and poked around at PCs until one of the sales staff approached."

later he came back with the information that we were looking for.

"Any of these computers will run Linux, but you might not get all of the features of the hardware on some of them. The best thing to do is have a look online to see which manufacturers have the best Linux support, and go with that. You'll also have to install Linux yourself."

"Is that difficult?"

"Very. I wouldn't do it myself. In fact, there's only one person in the store I'd trust to do it."

This was a bit of an overstatement. Installing a modern distro shouldn't be too much of a headache as long as your hardware is supported, but the rest of the information I'd received was pretty good.

Bare-bones option

"Is it possible to get one of these machines without Windows? I don't really want to pay for it if I'm not going to use it."

"We do bare-bones computers without operating systems, but we don't have them in store. They're only on the website. I can show you..."

We went over to one of the Windows machines and he pointed the web browser to the PC World website, but there was a problem with the internet connection. We continued chatting for a bit.

"Do many people come in asking for Linux?"

"Not many, but a few do. We get a lot of people asking about alternative operating systems."

"What do they want?"

"Anything other than Windows 8."

"What do they get?"

"Windows 8. It's the only option."

We left it there, and went on our way. We looked up the bare bones PCs on the PC World website back in the office. There are only four options listed, and of these, just one is available for sale (a low-end desktop for £209). Not particularly impressive since PC World has 50 Windows PCs.

Second-hand computers

Unless you need cutting-edge performance, a second-hand machine can also be a good choice. The majority of second-hand machines are old corporate laptops by HP, Dell or Lenovo. There are some great bargains to be had, especially if battery life isn't critical to you.

Perhaps the best thing is that these slightly older laptops have been tried and tested with Linux for a few years, so it's easy to find out what's compatible with Linux. A quick web search should bring up what works and what doesn't on a particular make and model.

Some companies sell refurbished computers with Linux installed, and one of these that deserves a particular mention is gluglug.co.uk. It sells refurbished ThinkPad laptops with Linux installed, but not only that, it flashes the firmware with Libreboot, so the laptop has no proprietary software on it. It's the first (and at the time of writing, only) computer vendor to earn the Free Software Foundation's 'Respects Your Freedoms' certification. Laptops start at £168 including shipping to Europe, USA or Canada.



PC World in Gloucester. It's not glamorous and they don't sell Linux computers, but the staff there did offer some useful advice.

PC World may be the last of the major PC retailers in the UK, but it's not the only place to get computer parts. The next stop on my shopping trip was Maplin. This is an electronics retailer that sells everything from remote control cars to transistors, and it has been a major retailer of the Raspberry Pi. Maplin doesn't sell assembled PCs, but it does have 'bundles', which include motherboard, processor, memory, hard drive, etc. Essentially, a flat-packed computer. Perhaps they'll guide us to a Linux desktop.

Again, we waited to be approached.

"Can I help you?"

"Yes. I'm hoping to build a Linux computer, and I was wondering if you knew which components would work best."

"Ah. Umm. I don't actually know too much about Linux. Hmm. Let's go and ask the manager. He knows a lot about most things"

"This gentleman wants to build a Linux computer. Do you know parts he should use?"

"All the parts we have should work under Linux, but the drivers for some of them might not be as good as the Windows ones. It's been a few years since I last used Linux, so I don't know what's best. There's loads of advice on the internet. If you do a search, you should find some forums where people can give you better advice than I can."

So far this was almost identical to our experience at PC World, though the chap in the shop did suggest that we should be able to install Linux "easily enough". It would be nice to have people more knowledgeable about Linux in shops, but the advice they were giving was actually pretty good.

Heading online

Of course, fewer and fewer of us do our shopping in real shops any more, so we decided to see what the state of the online Linux computer market place is now. If you're in the US, ZaReason and System76 both



build computers specifically for Linux, so you can be sure that the hardware will be well supported. Unfortunately for us this side of the pond, ZaReason doesn't ship across the Atlantic, though it has promised us a UK store soon. System76 does ship internationally, though the shipping costs and import duty add to the bill, and not everyone is happy about ordering expensive items like computers internationally, as it complicates the returns process if anything goes wrong.

There aren't any specialist Linux vendors in the UK, but we do have a few companies that custom build computers, such as pcspecialist.co.uk and cyberpowersystems.co.uk. While these won't ensure your system is fully compatible with Linux, or even install Linux for you, they do enable you to select everything that goes into your computer so that you can make sure that everything's Linux compatible.

Perhaps a little surprisingly, the desktop PC niche is the hardest computing market in which to find Linux support. On Amazon, the top three best-selling laptops all run Linux. They're running it in the form of

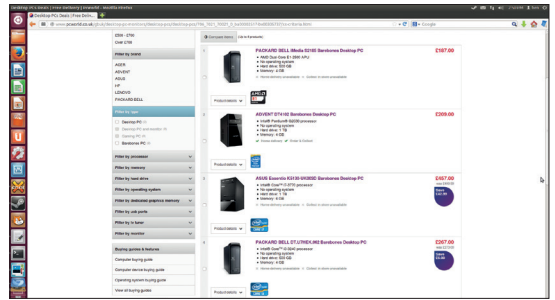
ChromeOS rather than a more traditional GNU/Linux system, but from a hardware perspective this isn't a problem, because it means that all of the

hardware will work with the kernel, so will work with other distributions as well, and there are a few projects – such as Crouton – that make installing Linux on Chromebooks easy.

The flexibility of Linux makes it a great system for building non-desktop, or slightly unusual desktop hardware. In the last few years, we've seen plenty of new takes on computing where manufacturers have created something a little different from a traditional desktop or laptop computer. More often than not, these computers have either run Linux exclusively, or supported it as an option. Closed source systems can't hope to have the same level of flexibility as open source ones, and this flexibility is essential when designing things that don't fit into traditional modes of computing.

“On Amazon, the top three best-selling laptops all run Linux, in the form of ChromeOS.”

Most NUC sellers offer Windows as an option rather than installing it as standard. Is this a sign that the once powerful Windows-Intel alliance is faltering?



Three of the four 'No OS' computers from PC World are not available. Perhaps they're sold out because all the other computers run Windows 8.

Intel has been pushing smaller computers with its Next Unit of Computing (NUC). With motherboards measuring just 4 x 4 inches, these almost fit into the palm of your hand, yet can be kitted out with a spec that would put many desktops to shame.

From a Linux user's perspective, there are two great things about the NUC: it uses Intel hardware, so is well supported, and operating systems cost extra. This last point is good because it makes people see the cost of having Windows installed on their computer, and it means that you don't have to pay for Windows if you don't want it.

Intel was far from the first company to make computers in this form factor though. There have been small home servers for several years, and even computing heavyweights such as Apple (the Mini) built little computers.

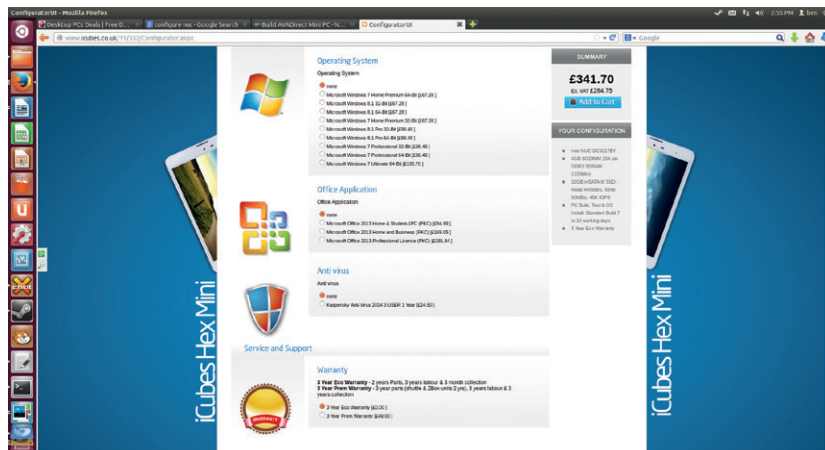
Mint machines

Embedded PC makers Compulab teamed up with Linux Mint to create the Mint Box, a small fanless PC. It's been available in America for some time, but has only just launched in Europe. This would be another great option for Linux users this side of the Atlantic, except it was such a good option that it promptly sold out. The Mint team assure us that more units are on their way, so should be for sale (on Amazon.de and possibly Amazon.co.uk) by the time you read this.

These smaller computers make great Linux boxes. The one major downside to the form factor is that they're not as upgradeable as traditional ATX machines. You usually have some ability to put in more memory, or replace the hard drives, but typically not the processor. Whether or not this is a problem really depends on you. Here at Linux Voice Mansions, we can't remember the last time we upgraded a CPU, so are inclined to say it's not an issue. However, other people may feel differently.

There's a special type of small form-factor computer that's been around for a long time – the games console. While Linux has run on some of them, none of the previous ones could have been called Linux Friendly. All this is about to change with the anticipated launch of SteamBoxes later in 2014.

This has excited gamers of all OS persuasions, but it could also be of interest to Linux computer



Open hardware

Some people like the flexibility of using Linux; some people like the concept of open source software; and some people like freedom. For these latter people, free software is only part of the solution. In a truly free computer, both the hardware and the software should be free. This not only means no binary drivers, but also that the full schematics of the computer should be available too. This way, the user has the same freedoms with the hardware that they have with free software.

Perhaps the biggest success story of open hardware is the Arduino microcontroller board. While these are several orders of magnitude less complex than PCs, their success shows the power of the idea. Because they combine open hardware and open software, people have been able to take the idea and convert it into new boards.

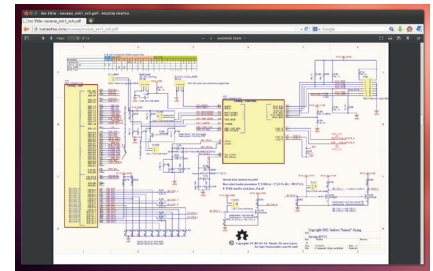
Back in 2012, Bunnie Huang set out to build a laptop based on the same principals. He aimed for a

working computer built from plans that anyone could download and copy. These plans include everything from the mainboard upwards. Early in 2014, he showed off the laptop in a working state. He plans to launch a crowdfunding campaign for people looking for a similar computer this year.

Bunnie's laptop is impressive, but it doesn't go all the way to truly free hardware, since it still relies on commercial chips with closed designs.

It might seem crazy to try to design an open source chip, especially one as complex as a CPU. Perhaps it is crazy, but that hasn't stopped people from doing it. The OpenRISC 1000 is an open source CPU written in Verilog. Verilog is a hardware description language that can be loaded onto Field Programmable Gate Arrays (FPGAs).

The OpenCore community, which developed OpenRISC, is hoping to go one better and is currently running a fundraiser to develop an



We have no idea what this is, but it's part of the schematic for Bunnie's laptop.

OpenRISC-based SoC and implement it in an Application Specific Integrated Circuit (ASIC). This, for the first time, will mean you can buy a completely open source board that will run Linux. See www.opencores.org/donation for more details.

shoppers. After all, SteamOS is built on Debian, so anything that works with SteamOS should also work with other Linuxes (at this stage, we use the word 'should' because it's still possible that Valve will do something to stop this working – though this seems unlikely). There is really no difference between a SteamBox and a PC except the form-factor. There still aren't many hard details on the specifications of the SteamBoxes, but several of the ones announced have been listed as having configurable hardware. We'll have to wait to see exactly what this means, but it looks like it could soon be another excellent option for buying a computer that's guaranteed to work with Linux regardless of whether you want to use it as a games console.

Perhaps the most famous small computers haven't been high-powered x86 machines, but lower-powered ARM ones. The Raspberry Pi is an obvious example, but it's not alone in this category. The Udoo (reviewed on page 53) and the Cubie board are just two more of an ever-growing range. Almost without exception,



While they don't have the raw processing power of x86, ARM chips are getting more powerful and are starting to become an option for desktop computing.

these run Linux (either a traditional desktop, or Android). There really isn't a credible alternative OS for small ARM boards. The disadvantage here, though, is that the hardware isn't as standardised as it is on x86 systems, so just because it runs one version of Linux, you can't be sure it'll run another. This is why, for example, you can run Ubuntu on the Udoo, but not the Raspberry Pi, and vice versa with Raspbian. It's also why you can't easily replace Android with desktop Linux on most devices.

“In the modern world, it's the PC shops that are struggling to keep up, not Linux.”

Welcome to the new reality

While it can sometimes be a little depressing to walk into a PC shop and see Windows machines all around you, the truth is that, in the modern world, it's these PC shops that are struggling to keep up, not Linux. If we really are moving into a post-PC world, then it's a world that Linux is poised to dominate. No other OS has the depth of hardware support and flexibility to enable it to run on so many different devices. It could be Android on embedded devices, an XBMC system running on a Home Theatre PC (HTPC), or an NUC that dual-boots SteamOS and desktop Linux. It's hard to see any other OS catching up to the lead Linux has when you look across all these computing platforms.

This writer, though, is highly sceptical of the term 'post-PC world'. It seems that the new devices that are coming out almost every day aren't replacing PCs and laptops, but supplementing them. If this is the case, Linux support is bound to increase as more of the non-PC computing devices use Linux, but it'll take a long time to supplant Microsoft on the desktop.

Whatever happens in the future, right now is a great time to buy a new Linux computer, and it's only likely to get better in the future.





FREE SOFTWARE FOUNDATION: EUROPE

We discover what this bastion of digital rights in Europe is doing for all of us.

“Ever since FSFE was founded in 2001, creating public awareness for Free Software has been at the heart of what we do. Today, there are more groups than ever before that really understand Free Software, and that are working to promote it. We talk to a lot of different audiences: developers, public sector people, businessfolks, students, police, church groups, and even the military – basically, we’ll go anywhere where people need to hear about Free Software. And on most of those occasions, we still need to start off by explaining what Free Software is and why it matters.”

These are the words of Karsten Gerloff in his reply to a question about how you measure the success of the Free Software Foundation, and more specifically, its European counterpart, the FSFE.

“Being able to reach all these audiences is a huge success. Seeing all the groups that have sprung up to promote Free Software in their specific environment is very satisfying. But there is so much left to do.”

In the beginning...

Richard Stallman has done many great things. Without him, there’d be no GNU, no GPL and no Emacs, and arguably, no Linux in the way that we know it. If the free software ecosystem existed without RMS, it wouldn’t be half as effective without

his idealism, insight and intelligence. Which is why, among those other accomplishments, he also founded the Free Software Foundation in early October 1985, just as Dire Straits’ *Money for Nothing* was to lose its reign at the top of the US singles chart.

But what many people don’t realise is that there are several sister organisations to the Free Software Foundation, including the Free Software Foundation Europe – or FSFE, as it’s better known. FSFE is far more than a local mirror for the US-based FSF. It’s been a significant third party in the European Union’s antitrust case against Microsoft by helping to put the case forward that competition in the file/print server market (thanks to Samba) is essential. At the end of March, the foundation also published an open letter to the EU Parliament and the European Commission asking for the support of open standards.

We had a chance to speak to Karsten Gerloff and Matthias Kirschner, President and Vice President of the Free Software Foundation Europe, about the European branch of their organisation, and Sam Tuke, its Campaign Manager, who writes a great report on this year’s Document Freedom Day over the page.

First, we asked whether there are any policy differences between the FSF and the FSF Europe. .

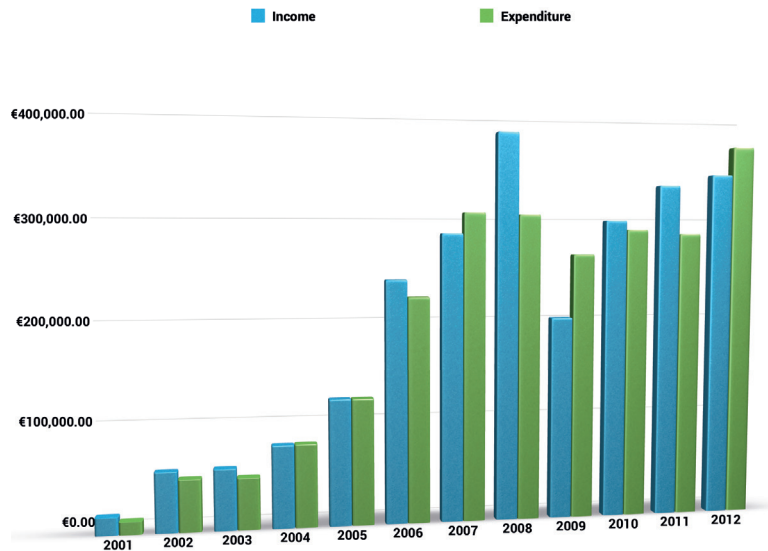
“No”, answered Karsten, “We share the goal of promoting Free Software, and do so by similar means.”

Income and expenses

2012

INCOME	
Donations	€197,420.59
Fellowship/membership contribs	€106,411.08
Paid services	€26,036.85
Merchandise	€13,493.84
Interest/currency exchange gains	€15.16
Total	€343,377.52

EXPENDITURE	
Basic infrastructure costs	€92,830.45
Public awareness	€96,626.77
Fellowship	€47,411.87
Legal work	€69,877.86
Policy work	€50,158.26
Merchandise	€14,262.61
Total	€371,167.82



For us, that begs the question of why a European-centric FSF is important.

“Being based and rooted in a specific region, in our case in Europe, makes it easier to be effective as advocates, by adapting to the local cultures and circumstances.”

Shared goals

“We frequently work together [with the FSF, FSF India and FSF Latin America] to come up with shared approaches to fundamental issues. These internal discussions tend to be very thorough, and quite productive. In organisational terms, however, all FSFs are fully independent of each other. FSFE has its own legal entity, employees, independent governance structure, and raises its own funds.”

Raising your own funding under the umbrella of a perhaps more widely known organisation must be a difficult task. We wanted to know how you measure the success of the FSFE, rather than the wider organisation, to be able to attract funding.

“That’s a tough call to make. We have had a lot of success in all three areas!” Karsten told us.

“In the legal field, FSFE has achieved something unique. Through eight years of careful work, we have built the world’s largest network of legal experts on Free Software. There are currently over 340 lawyers and engineers in this group, including many of the very best people in the field. They are helping each other learn more about Free Software and about how their respective companies

“We have built the world’s largest network of legal experts on Free Software.”

and organisations are using it, and remove fear, uncertainty, and doubt. This has given us a great set of contacts to many of the companies that build Free Software. For the companies, this exchange of

Help the FSFE

We asked Karsten how best we the community might help.

“There are many different ways how people can contribute to FSFE.” he told us.

“We have a page (<http://fsfe.org/contribute/contribute.en.html>) that outlines most of them. If you’re passionate about Free Software, and want to work with others who care for the same thing as you, there will be a place for you. One of the reasons FSFE is doing so well is that we try very hard to identify each person’s speciality, and help them use it for Free Software in the most effective way. In the UK especially, we would love to get more local groups going, with regular meetings and occasional events. People who are interested should write to fellowship@fsfeurope.org, or simply contact anyone they know in FSFE!



Karsten Gerloff, protector of Free Software and President of FSFE.

knowledge means that they better understand their obligations when using Free Software, leading to fewer licence violations.”

“We have always invested a lot of time and effort in policy work, whether it was about keeping software patents out of Europe, helping the European Commission to bring Microsoft to book for its anti-competitive behaviour, promoting Open Standards like ODF, or changing public procurement. These issues often take many years to bring to a conclusion – that is, if they ever end. FSFE is one of very few organisations, and almost the only one specialising in Free Software, that has the skills and resources to stay on the ball for as long as it takes.”

We love the way Karsten says “for as long as it takes”, as we think that’s the vital component in the FSF’s strategy – an unrelenting approach to Free Software advocacy and adoption. You know it’s not going to sell out or dilute its vision in the face of commercial pressure. And that’s an important differentiator between Free Software and other development models. It’s channelling spirit of Stallman through the foundation he created 30 years ago.

“When companies lose the fear of putting Free Software into their products, and ideally give users the possibility to change it on their devices, that is success for us.” Karsten told us.

This is just as important today, now that Windows XP is no longer supported, as it was when Microsoft was more confrontational and the FSF was trying to side-step the FUD being thrown. But times have definitely changed.

“When the European Commission gives Microsoft a record antitrust fine, along with a clear message that their behaviour isn’t acceptable around here, that is success for us,” Karsten began.

“When the UK government goes ahead and opts for ODF as a default format for its documents, in the face of fierce resistance from the incumbent IT suppliers, that is success for us. We don’t always get everything we want. But often we’ll get most of what we push for.”

“The biggest challenge for the next 10 years will be making sure that we can be in control of our own computing. That’s really what a lot of the fights we are fighting today are about: Can you be trusted to control your own computer?”

Document Freedom Day 2014

Freedom to read, write, and create requires freedom of formats, writes **Sam Tuke**. Here’s how one campaign brought open standards to 51 locations in 22 countries.

Balloons litter the floor, cream smeared plates pile high on tressel tables, beside which the crowd of participants file out of the auditorium door. “Give us a chance – choose Open Standards” reads the Spanish posters adorning the walls, and a typical Document Freedom Day event ends.

An animated band of students and professors have been debating the video files that Grenada University uses for publishing research in Southern Spain. They’ve been here all afternoon, and in a few hours their pictures will join hundreds of others that have been streaming out of cafes, lecture halls and hackerspaces around the world the last 24 hours.

As well as baking cakes, volunteers took the word of document freedom out into the streets.

Document Freedom Day is when people celebrate freedom from data format tyranny and the systems that preserve it. On the last Wednesday of March every year groups like GALPon in Granada University (“Grupo de Amigos de Linux de Pontevedra”) take the opportunity to run events that explain why these freedoms are important, and share the knowledge and tools necessary for citizens to take them back.

Some events are big, like the 300-attendee speech at Istanbul Turkey. Some are small, like the group of sixteen friends who discussed Open Document Format (ODF) in Yuli Township, Taiwan. Others are run by governments, like the Brazilian Federal Government CISL Committee. And a few take place in schools, like Maltepe Nezahat Aslan Ekşioğlu Primary in Istanbul. All are organised at a local level by independent community leaders.

Open as standard

While DFD is about people, events, and not a little cake, many of the issues addressed are necessarily technical. What you can do with a file once it’s been saved depends on the format of the data inside. What governs such formats are loosely called standards. And similarly to software applications, some standards protect freedoms while others prohibit them. Unlike software however, generic licences like





the GPL don't exist for standards. The two serve fundamentally different purposes, and what makes a file format useful is more nuanced than what makes software free. For example, the future development of a standard can be just as important as its past. When LibreOffice forked OpenOffice, the result was two separate, independently useful applications. But if a fork of their native file format, ODF, had also been made, it would have been far less useful, as no other applications would have been able to understand it. Standards provide a platform of data compatibility upon which software is built and competes.

So instead of a license, criteria are used to identify freedom-respecting standards. The ones that pass the test are "Open Standards", and the rest are "closed". Some governments, including the European Union, have their own definition. Document Freedom Day uses FSFE's five-point version, and while the variations are the subject of heated political debate, most agree on core requirements that the standard may be used by everyone, that technical details are freely available, and that modifications to the standard are set by an impartial group.

While the politics of file extensions may be fascinating to power-users and freedom fighters, getting the message to everyone else can be challenging. Highlighting the importance of Open Standards to mainstream society is a core goal of Document Freedom Day, and talking direct is a fun and effective way to achieve this.

In 2012 we raised eyebrows and headlines by sending steel handcuffs to politicians and public figures whose websites endorsed closed standards. European Commission Vice President Neelie Kroes

even showed hers off during a keynote speech (www.guardian.co.uk/technology/2012/apr/19/digital-handcuffs-ec-vice-president).

Last year netizens reported popular websites that used Adobe Flash instead of HTML5 technology for streaming video. Educational packs were duly delivered to the appropriate webmasters, together with a pair of blacked-out "plugin required" glasses illustrating the downgraded experience of users missing closed standard browser extensions.

In March, MEPs were challenged to reconsider Parliamentary security when a panel of experts debated surveillance in the European Parliament for Document Freedom Day 2014, and on the same day an open letter to EU Institutions confronted their captivity to Microsoft.

My job as campaign manager is to empower local people to serve their own communities. Our small team in Berlin can't be in 51 places at once, but our network of freedom fighters can. Nor could we generate the enormous creative energy that marks the campaign each year. A monkey hitting a typewriter infinite times may finally write Shakespeare, but it's doubtful the otherworldly sounds produced in last year's DFD Zurich open audio jam could have come from a centralised campaign.

And when getting involved can be so much fun, it's easy to ask others to participate, so why not run your own event? Join us next year in the campaign for document freedom! 🐒

"Highlighting the importance of open standards to mainstream society is a core goal of DFD."

All this year's DFD materials and source code are hosted online under copyleft licenses. Recipes, origami, certificate templates and forms for claiming back expenses are all in public repositories.



Designing and implementing your own CPU or System-on-Chip brings benefits to thousands of researchers and forward-looking businesses, and is being adopted by a growing number of hobbyists. **Richard Smedley** finds freedom in configurable silicon.

For some years (the need for a few binary blobs in the kernel excepted) many readers have run an entirely Free Software stack on their servers, laptops, desktops, and even tablets and phones. But at the silicon level it's another story, with open source hardware limited to a few embedded boards like the Arduino. The good news is that not only are there open source designs for CPUs and Systems-on-Chip (SoC) nowadays, but that it's not too hard to learn to design and make your own. Indeed, there are projects designed to get you started doing just this.

One such of these is OpenCores, which bills itself as "the #1 community within open source hardware IP-cores", backing the claim with a statistic of more than 200,000 registered users. It hosts projects ranging from relatively simple UARTs (universal asynchronous receiver/transmitter) and Ethernet MAC (Media Access Control) LAN implementations right up to the complexity of full OpenRISC chips.

That reference to "IP-cores", rather than CPU cores is an abbreviation for so-called "intellectual property", and is a telling reflection of the proprietary nature of

Space RISC

OpenCore has gone beyond earth-bound applications, after students at San Jose State University – funded by NASA's Ames Research Center – designed a 1U satellite, TechEdSat, to evaluate ÅAC Microtec's implementation of OpenRISC, and perform communications experiments.

The satellite, which was deployed from the International Space Station in 2012, cost less than US\$30,000 to build thanks to the combination of OpenRISC and off-the-shelf hardware selected to be rugged enough for space use.

According to engineers from ÅAC Microtec, the standard OpenRISC design was modified with fault-tolerant features and toolchain modifications invisible to the end-user software as different from standard OpenRISC spec. The great thing about using an open specification is that these modifications have no barrier in terms of licensing or configuration information, while the flexibility of FPGAs makes prototyping quick and (relatively) easy.

At these prices it's now conceivable that with savvy sponsorship, even schools could launch a satellite with their own custom CPU. However, don't forget you can send a

Raspberry Pi to near-space from your school for 1% of this cost, as David Akerman did when he launched his Pi and a camera on a balloon into the skies over Berkshire.



OpenCores in space: the OpenRISC powered TechEdSat is deployed from the International Space Station.

most work cast into silicon. The fast growth of OpenCores shows that there's enthusiasm and a business need for a more open alternative. Naturally, the opportunity that OpenRISC presents to gives playing with the design of a full-blown modern microprocessor makes OpenRISC useful in universities, and the freedom to explore means another field opened to hobbyists. But what's really driving development is a number of businesses taking advantage of a flexible, cost-effective route to specialist markets.

RISCing it

So, why pursue open CPU architecture, and why go the RISC route? The latter question is the simplest to answer. The case for RISC (Reduced Instruction Set Computing) was well made by IBM researchers in the late 1970s, and producers of the first RISC1 chip at the University of California, Berkeley 30 years ago. Reducing the operation code instructions in silicon (by a factor of 10 at the time), not only simplifies design but frees up space for more registers and cache. Efficient compiler design of the time took away the need for most instruction operation code, and the situation is unchanged today.

OpenRISC is a family of 32- and 64-bit processors with optional floating point and vector processing support. It's a free, open source RISC architecture with DSP (digital signal processor) features and a complete set of free, open source software development tools, libraries, operating systems and applications. The reference design, the snappily titled OR1K (OpenRISC 1000) is implemented as OpenRISC 1200 (OR1200), a synthesisable CPU core released under the GNU Lesser General Public Licence (LGPL).

Writing your own design (for an OpenRISC chip) consists of using a Hardware Description Language (such as Verilog) to describe the chip at the most basic level. Then comes synthesis – conversion to the

list of logic gates and connections used in your chosen FPGA. This latest acronym is a Field Programmable Gate Array, which is a kind of chip that isn't yet set in stone. One FPGA costs a lot more to make than the equivalent processor, but the extra flexibility means that if the design doesn't work the way you want it to, you can simply change it (that's the Field Programmable part).

Free as in almost

The netlist produced is a gate level description, which then usually uses the chip manufacturer's proprietary software to produce the programmed FPGA. For anyone wanting 100% Free and Open Source hardware and design there doesn't seem to be a way around this at the moment. As Embecosm founder and OpenCores stalwart Dr Jeremy Bennett told Linux Voice: "The back-end tools are proprietary to the FPGA manufacturers. Since these tools depend on intimate

Dr Jeremy Bennett of Embecosm showing the OpenRISC SoC implementation on FPGA at an Open Source Hardware Users Group meeting in 2011.

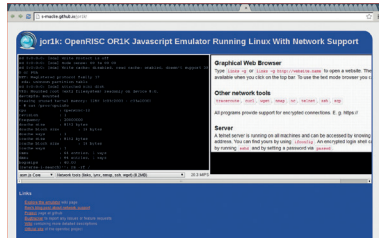


Browsing the hardware

While the idea of building your own CPU appeals to many of us, perhaps you are looking for a way of testing the waters without all the kit. Sebastian Macke of simulationcorner.net has written `gor1k` – the JavaScript OpenRisc 1000 emulator – which gives you the chance to try out open hardware design in that most familiar and comfortable environment, your web browser. `gor1k` works with Firefox and Chrome, though if running locally with the latter you need to run the browser with the command `--disable-web-security`.

The emulated OpenRISC CPU is around 1000 lines of code – a neat introduction to emulation, the OpenRISC architecture, and JavaScript programming all in one! It's also a handy sandbox to test OpenRISC ports, and you could try modifying the emulator to test out ideas for modifying OpenRISC away from the standard implementation.

The project's GitHub pages – <https://github.com/s-macke/gor1k/> – include a wiki with useful and interesting notes on some of the JavaScript optimisations used in the code, as well as speed differences between browsers and a list of the many demonstrations available in the Linux image on the emulator.



If your emulated OpenRISC goes wrong you can just scrap it and start again.

knowledge of the device, it is hard to see how there could be a free and open source implementation, unless the manufacturer chose to do so."

Given the growth of understanding in the advantages of open source methodology, this is not an impossible wish. Meanwhile, we accept that we live in an imperfect world, and continue to make it better – or at least more interesting – to the best of our abilities. At least the Linux-compatibility of the tools is good.

Fabulous Fables

Designing and fabricating semiconductors is an expensive business. You don't get many opportunities to create prototypes of designs that have tens of millions of transistors in them, and this has led to notable bugs such as the Pentium FDIV bug, which caused the processor to return incorrect results in floating point calculations (Intel eventually had to

recall the chip, but not before considerable damage to its reputation). With even giants like Intel having rationalised its range of offerings in the last decade to concentrate resources on the most profitable lines, OpenRisc is a disruptive

technology, enabling semiconductor companies to develop chips for embedded markets like network devices, personal entertainment hardware, and niche industrial applications – without having to spend money on operating their own factories.

Much of the active development on OpenRISC comes from companies like Swedish design house ORSoC, which also sponsors the OpenRISC project

directly. Many other small companies make chips and boards based on OR1K, including AAC Microtec, which has had its product put into orbit. The fast development offered by open hardware also makes it great for larger companies playing in fast-moving markets: Samsung ships OpenRISC chips in the system-on-chips used in its digital TVs.

Any curious hacker or maker can experiment with FPGAs and OpenRISC. Delving into chip design enables you to grapple with all sorts of tasty problems involving Fused Multiple Accumulator (FMAC) arithmetic, bus design, and optimal register numbers. If you've ever programmed at a low level, and cursed the decisions made by chip designers at Intel, now is your chance to show the world a better way!

Anyone wanting to join in the fun will find many resources online, but also meetings and chances to learn the process of programming your own FPGA through the Open Source Hardware Users Group (OSHUG), which conducts meetings in and around London but also ventured north for last year's Open Source Hardware Camp at the Wuthering Bytes festival in Hebden Bridge.

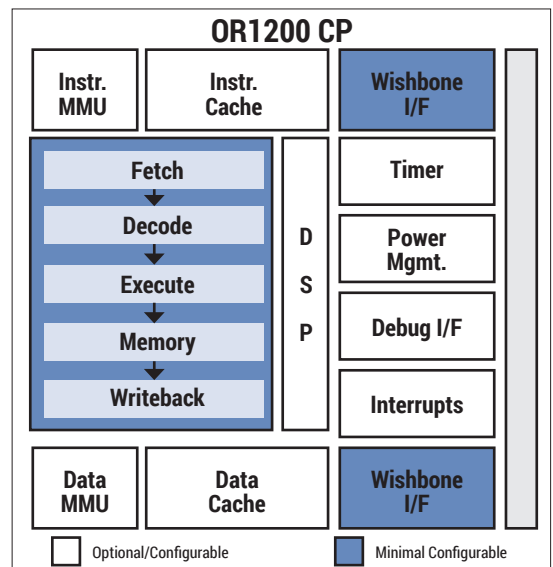
Open to all

Working with OpenCore designs is challenging but rewarding. "Inexperienced users should be warned that the OpenRISC processor is quite a difficult processor," warns Patrick Pelgrims of the Belgian De Nayer Instituut, in his tutorial on designing and implementing an OpenRISC-based embedded system. But we don't want that to put you off – the reference design is a good place to start, and as with learning programming through playing with existing, working code, so with hardware.

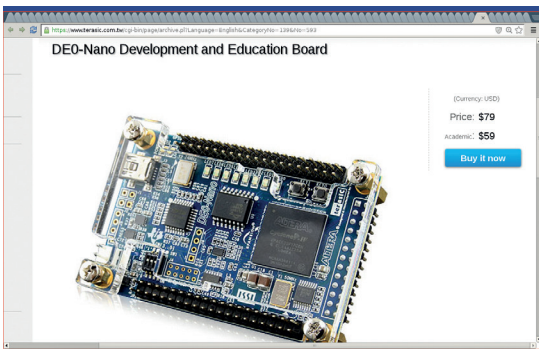
We asked Dr Bennett about the difficulties involved. He pointed out that it's "a relatively simple and well documented architecture. It has a pipeline (more difficult), but only a five-stage pipeline in the standard

"It is well within the grasp of a competent hobbyist. And of course modifying an existing design is always easier than designing one from scratch!"

Dr Jeremy Bennett.



Inside the OpenRisc 1200 CPU – configuration at the silicon level, with Free and Open Source Hardware.



You'll need an FPGA development board to get started which, while not cost-free, is orders of magnitude less expensive than building a CPU plant!

implementation (so not that difficult)." He summed it up as: "more complicated than some, but a lot less complicated than many. The bottom line is that processor design is not trivial. On the other hand it is well within the grasp of a competent hobbyist. And of course modifying an existing design is always easier than designing one from scratch!"

Chip Hack & getting involved

OSHUG runs an annual event called Chip Hack, which is a weekend of learning to create embedded hardware, building and making, and taking home your own OpenRISC SoC:

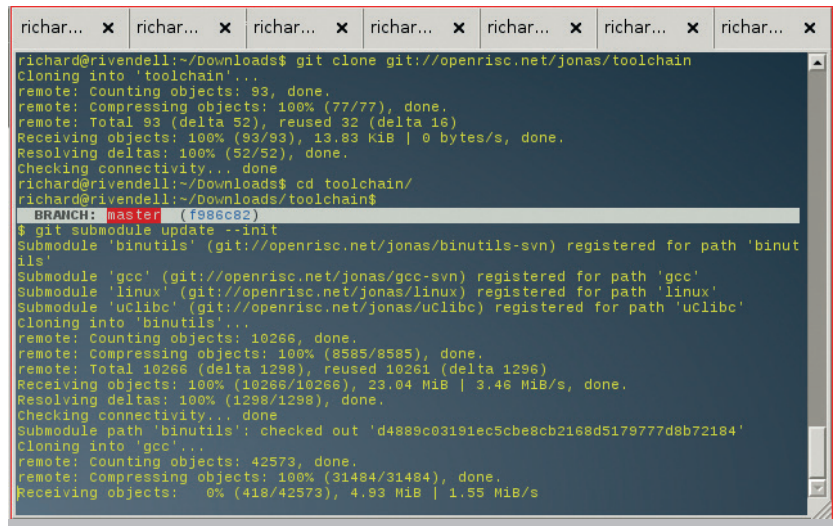
If you can't get to Chip Hack you can still give it a try yourself: you could use a browser-based emulator to explore OpenRISC (see boxout, above-left), but getting the toolchain installed on your PC to get started can be as simple as:

```
$ git clone git://openrisc.net/jonas/toolchain
$ cd toolchain
$ git submodule update --init
$ make -j3 PREFIX=~/.openrisc/toolchain
$ export PATH=PREFIX=~/.openrisc/toolchain/bin:$PATH
```

For an optimum `make -j` value, double your number of CPU cores, and add one. The destination directory can be anywhere you have permission to put it. After setting up your cross-compile environment and building a Linux kernel (see <http://openrisc.net/>

DIY chips on the web

- Introduction to FPGA programming event. <http://chiphack.org>
 - Good beginners' introduction. www.rte.se/blog/blogg-modesty-corex/openrisc-1200-soft-processor
 - Julius Baxter's Masters Thesis on the OpenRISC Project. <http://juliusbaxter.net>
 - Paper covering all the chips from OpenSPARC to the European Space Agency's LEON project: <http://ur1.ca/gyitc>
 - jor1k – OpenRisc 1000 in your browser. <http://s-macke.github.com/jor1k>
 - Open Source Hardware User Group <http://oshug.org>
- Also, there's a supportive community on the #opencores channel on freenode IRC.



`toolchain-build.html`) you can test-run your OpenRISC environment in a VM with:

```
or1ksim -f arch/openrisc/or1ksim.cfg vmlinux
```

Next, you'll need an FPGA development board. This year's Chip Hack event will use the DEO-Nano board, but there are plenty of others listed on the **OpenCores.org** website, including some recent developments. As noted earlier, you will need proprietary Quartus software from Altera installed to turn the Verilog HDL file into something that can be loaded onto the FPGA. Before that you'll need the Verilog file itself – the OpenRISC site has an OpenRISC Reference Platform System-on-Chip in the flavour you need.

Environmentally friendly

Low power consumption has always been an important selling point for RISC chips, enabling them to quietly conquer the embedded space in the 1990s, and thus be the big winners in the rise of the mobile device. Given the huge power consumption of data centres on a worldwide scale, it's no surprise to find OpenCore developers at UK-based Embecosm, which did much of the work on GCC and the GNU toolchain for ORSoC. Adapteva (developers of "a revolutionary many-core embedded computing platform for applications requiring ultra high floating-point performance with minimal power consumption"), is also doing work funded by the UK Technology Strategy Board (a UK government innovation agency), to optimise GCC for compiling binaries with a lower power draw.

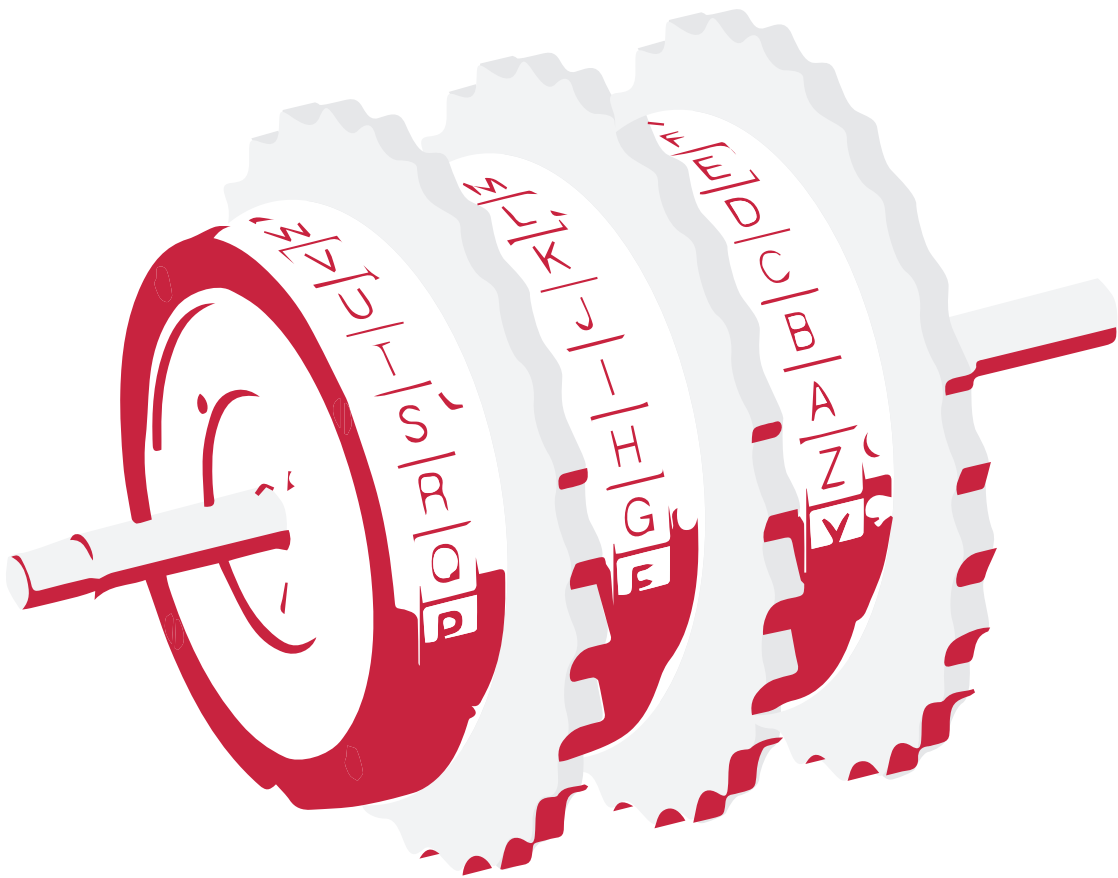
"Low power consumption has always been an important selling point for RISC chips."

As we go to press, the Chip Hack Cambridge event, providing an introduction to FPGA programming, is already sold out "but a key part of the idea is that he resources are open, so others can run the course themselves," Dr Bennett tells us. Get on the Chip Hack mailing list, and you should get early news of other events and training opportunities.

You could build an entire OpenRISC toolchain yourself, but as the hard work's already been done, just grab it with Git and get on with the fun.

Alan Turing, Colossus and Turing Machines

We're taking a break from our Olde Code series this issue to get some background on a man who was in the news as recently as last year. **Juliet Kemp** looks back at the early works of Alan Turing.



Alan Turing was born in London in 1912, studied mathematics at King's College, Cambridge, and was elected a fellow there in 1935. He worked on the Entscheidungsproblem, then spent two years studying maths and cryptology at Princeton. While there he also built part of an electro-mechanical binary multiplier. This type of machine – a computer, but not a programmable one – was the state of the art in computer hardware at the time. In 1938, Konrad Zuse would complete the Z1, the first mechanical binary programmable computer, in Berlin, although the Z1 was not a general purpose machine as it had no loop capacity.

When WWII began, Turing, who was already working part-time with the Government Code and

Cypher School (GC&CS), reported to Bletchley Park to work full time on cryptanalysis. His work on deciphering Enigma, on the Bombes, and his contributions to the development of Colossus, were a hugely valuable part of the history of early computing; they also remained secret until the 1970s.

Turing machines

In 1928, mathematician David Hilbert posed the Entscheidungsproblem (Decision Problem): does an effective procedure exist that would demonstrate whether or not a given mathematical statement is provable from a given set of axioms? In 1931, the Austrian mathematician Kurt Gödel demonstrated that any arithmetic system must be incomplete (that

“A universal Turing machine is one that can compute any computable sequence. Modern computers are all universal Turing machines, and we take this idea for granted now.”

is, it is possible to construct a statement that can be neither proved nor disproved), but did not tackle the provability problem.

In 1936, Turing wrote “On Computable Numbers with an Application to the Entscheidungsproblem”, which showed that no such procedure exists. He used the idea of an “a-machine”, now known as a “Turing machine”. This is a hypothetical computing device, which reads an infinite tape. At any one moment the machine reads a single symbol from the tape. It may alter that symbol, and the symbol may (combined with the machine’s instruction table) affect its behaviour. The tape moves backwards and forwards, so any symbol may eventually be read by the machine. Turing equated the problem of deciding whether a Turing machine halts on a given algorithm to the Entscheidungsproblem. He proved that some processes will never halt; and thus that the answer to the Entscheidungsproblem is ‘no’.

A universal Turing machine is one that can compute any computable sequence. Modern computers are all universal Turing machines, and we take this idea for granted now. At the time, however, it was a huge breakthrough, which arguably led to the idea of stored-program computers.

Bombes and Enigma

One of the simplest ciphers is a substitution cipher: each letter is substituted with another letter. This is readily breakable, but it becomes less so if you use a different substitution alphabet for each letter of the message. The Enigma machine, invented at the end of WWI, had a system of rotors which both mechanised this process (making it easier to operate), and increased the number of cipher alphabet options. Each rotor had 26 positions, and they were connected in series, so each letter was transformed multiple times. Further, the rings stepped onwards for every letter (how often the rotors stepped varied, but at least one rotor would step at least once for each letter). This meant that each letter was encrypted with an entirely new cipher from a huge number of options. Finally, a ‘plugboard’ swapped some pairs of letters before they were output to confuse matters further (though in fact, contrary to expectation, this made breaking it slightly simpler).

Before WWII, the Poles had already had some success with breaking Enigma messages with their bomba cryptologiczna machine. However, as the



Germans introduced more rotors and more plugboard settings to their Enigmas, these machines couldn’t keep up. The Bletchley Park codebreakers needed to up their game to decrypt Enigma traffic.

On the shoulders of Polish giants

Turing took the bomba cryptologiczna and improved on it to create the Bombe. A standard British bombe contained rotors to the tune of 36 Enigma equivalents, enabling it to work very rapidly. The bombe input was a crib (a fragment of probable plaintext), which was tested against the ciphertext. The bombe electronically performed various logical deductions based on this, and if a contradiction arose (which it would do with most possible settings), that crib could be discarded. Only a few settings would then be left for the cryptanalysts to look at in more detail. This was an electronic computer in one sense, but it wasn’t programmable; it performed just one task. The cribs were obtained by taking advantage of various regularities in the messages transmitted, such as weather reports and message setting information.

Turing’s colleague, Gordon Welchman, later implemented the ‘diagonal board’ improvement. The

Turing was also an accomplished athlete, with a personal best marathon time of 2 hours 46 minutes.

bombes were immensely successful, but initially Turing and his colleagues could not get the resources for more bombes and more people. Eventually they went against military procedure and contacted Churchill directly; resulting in Churchill giving the highest priority to support for the codebreaking team at Bletchley Park.

Turing also worked specifically on naval Enigma, which he started on "because no one else was interested in it so I could have it all to myself". The chief difficulty here was that the sender enciphered the message settings (the information about which rotor settings the message was encrypted with) twice, once by Enigma and once by hand using 'bigram tables' (tables of letter pairs). Turing deduced how this system worked, but was not able to move further

before the Royal Navy got hold of some actual bigram tables.

The Colossus machine was designed by Tommy Flowers to help with the cryptanalysis of the Lorenz cipher. The Lorenz cipher machines

were used for high-level wireless traffic between German High Command in Berlin and army commands throughout Europe, during World War II, whereas, as discussed above, the more portable Enigma machines were used for other German army and naval messages. It has often been stated that Turing was involved with the development of Colossus; in fact, this was true only in that his statistical methods were part of its ancestry and the cryptanalysis methods that prompted its building.

"Information about Colossus began to emerge in the late 1970s, and GCHQ released a 1945 report on the breaking of the Tunny cipher in 2000."

Rotor encryption

Lorenz worked in a similar way to Enigma, with multiple moving wheels producing a ciphertext. Similarly, one of the routes in to the cryptanalysis was the 'indicator' (showing the start position of the

wheels) sent at the start of the message. Being able to identify when two messages had used the same wheel settings gave the codebreakers access to messages in 'depth', which is crucial in codebreaking.

A colleague of Turing's, John Tiltman, managed to identify the cipher used (the Vernam stream cipher) from studying intercepted ciphertexts, and after this, Bill Tutte worked out the logical structure of the machine from further ciphertext study, without ever seeing a Lorenz machine – a hugely impressive achievement. Tutte and his colleagues correctly determined that the machine had two sets of wheels, chi and psi. The chi wheels all moved on one position with each character. The five psi wheels also all moved together, but at a different rate, controlled by two 'mu' motor wheels. This gave a huge number of machine settings and cipher alphabets, which changed in a complicated way.

Turing's main contribution was a process known as Turingery. This revolved around 'differencing', in which he XORed successive characters to emphasise any points at which the characters moved away from a uniform distribution. The next step was a complex statistical analysis of the ciphertext, in which the cryptanalyst tried out a huge number of possibilities and compared the resulting patterns with one another. Eventually, the chi wheel settings could be deduced, and from there the psi and mu settings.

Turingery was a hand method, and slow. Tutte used it as a basis for his own '1+2 break in'. This required trying all possible combinations of the chi wheels against the ciphertext, and looking for subtle statistical evidence of non-uniformity. This used differencing to amplify the effect, and it worked well – but it was only practical if it could be automated.

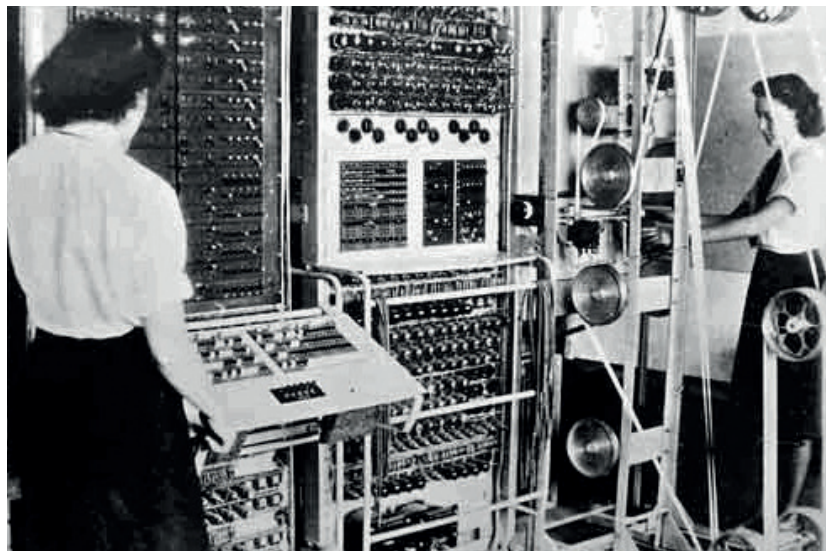
Construction time again

So in 1943 Max Newman, with Frank Morrell and Tommy Flowers from the Post Office Research Station, produced a machine known as 'Heath Robinson', using valves and paper tape. It was Flowers who realised that this could be improved on by building an entirely electronic machine (using an electronic key stream rather than reading off paper tape, although the message was still fed in on tape). Most people argued that this would be far too unreliable to be useful, but, supported by the Controller of Research at the Post Office, Flowers went ahead and built it. It first ran in December 1943 and was operational by early February 1944.

This was, then, Colossus, the first programmable (but not general purpose) digital computer. It had four main parts:

- **Tape transport and reading mechanism** Read the message tape in at 5000 characters per second, using the sprocket holes as a clock signal.
- **Key generation unit** Generated an electronic key (chi) stream.
- **Combining unit** Implemented the logic of the 2+1 method.

Colossus Mark 2 being operated by Dorothy Du Boisson and Elsie Booker. The tape transport is shown on the right of the photo.



■ **Counting unit** Counted the dots in the output and printed it out if it was over a given total.

It was so successful that they immediately began building more in place of the Robinsons. The Colossuses, and Colossus II (in operation from June 1944, the week before the Normandy landings), were vital for the remainder of the war effort, but after the war, all evidence of the project, physical and paper notes, were destroyed, for security reasons. Despite this, the number of people who had worked on the project and who went on to work in early computers meant that Colossus and the other Bletchley projects did have a significant indirect impact.

Post-war: the Pilot ACE

After the war, Turing worked on the design of the Automatic Computing Engine (ACE) at the National Physical Laboratory. The resultant paper, in 1946, was the first detailed design of a stored-program computer. The ACE implemented subroutines, and even something called Abbreviated Computer Instructions, which was a sort of programming language.

In the ACE, instead of a CPU, memory locations and temporary stores had specific logical functions associated with them. So transferring two numbers to a particular memory location, for example, added them. To speed up executing, Turing suggested that instructions should be stored at specific locations, with each instruction pointing to the next, in such a way as to optimise instruction access. (Experienced UNIVAC programmers did something similar to get around the limitations of mercury delay line memory – see LV002's tutorial on Grace Hopper and UNIVAC.) He also included a small fast-access memory for storing frequently used numbers or ones that needed to be stored temporarily.

Turing and his team, as well as sketching versions of the ACE, also wrote 'instruction tables'. Their aim was for programmers to be able to select groups of standard instructions and link them together with other cards, and they prepared in detail 'instruction routines' including division, extracting square roots, and logarithms. This echoes the work later done by Grace Hopper on UNIVAC, but sadly in Turing's case, his instructions only ever existed on paper.

Turing knew that what he proposed was feasible, and wanted Tommy Flowers to be involved. However, since no one who hadn't been at Bletchley knew about the Colossus, everyone else thought his proposal far too ambitious, and Flowers wasn't recruited. Instead, they eventually built the smaller Pilot Model ACE. This had 1,450 vacuum tubes and 12 mercury delay lines as its memory (each storing 32 bits – again, see the UNIVAC for more on mercury delay lines). Its clock speed was 1MHz, which at the time it first ran (10 May, 1950) was the fastest in the world, and around 10 times faster than its contemporary, the Manchester Mark 1.

The ACE design was also used for the MOSAIC (1952) which calculated aircraft trajectories from



The Turing Bombe rebuild project at Bletchley Park. Photograph by Mike Peel (www.mikepeel.net).

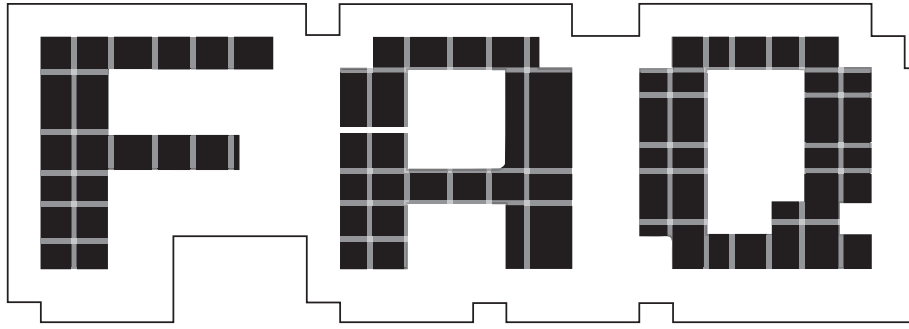
radar data (further information is still classified). The first personal computer (well, arguably; it was a small single-user machine), the G-15, built in 1954, also used ACE principles. The commercial version of the Pilot ACE, the DEUCE, was available from 1955 until 1964, and used various languages including one called GEORGE (1957) which used reverse Polish notation and had a 12-position stack. It was sold with an extensive program library of subroutines – perhaps the descendants of Turing's on-paper versions for his full ACE.

ACE program example

Wilkinson's Progress Report on the Automatic Computing Engine (April 1948) includes this code example to calculate squares and cubes of n by iteration until $n = m$ (m is stored in tank TS1):

```
A1 zeros+ zeros -> TS2 lmm 1, A
A3 zeros+ zeros -> TS3 lmm 1, A
A5 zeros+ zeros -> TS5 lmm 1, A
-----
A7 TS2 + TS3 -> TS3 lmm 1, A   i.e. n2 + n -> TS3
A9 TS3 + TS4 -> TS4 lmm 1, A   i.e. n3 + 3(n2 + n)
-> TS4
A13 TS2 + P1 -> TS2 lmm 1, A   i.e. n + 1 -> TS2
A15 TS3 + TS2 -> TS3 lmm 1, A   i.e. (n2 + n) + (n + 1) -> TS3
A17 TS4 + P1 -> TS4 lmm 1, A   i.e. n3 + 3(n2 + n)
+ 1 -> TS4
A19 TS2 != TS1 -> DISC lmm 18, A If new n = m, return to A7,
else A6
-----
A6 END
```

The temporary TS tanks are used for quick access. TS2 holds n , TS3 holds n^2 , and TS4 holds n^3 . The first three instructions simply zero the tanks TS2, TS3, and TS4 (so n , n^2 , and n^3 are all, correctly, zero). The following steps calculate the values iteratively, as described; by step A19, TS2, TS3, and TS4 will all hold their new values. (These are all discarded rather than saved, as this code was for demonstration only.) lmm 1 means an immediate transfer with timing number 1, that is, it goes straight to the next instruction. For more detailed information, check out the full paper. And that, dear readers, is where we'll have to leave him for now. We'll get to Turing's work in Manchester in a forthcoming issue of Linux Voice. 📖



DDOS

The internet attack of choice for gangsters, governments and bored geeks.

BEN EVERARD

Q Another acronym! First things first, how do I pronounce it? Dos? D'dos? Dee-dos? And how is it different from MS-DOS?

A For once, there seems to be a fairly accepted pronunciation: Dee-dos. It stands for Distributed Denial Of Service, and it's a way that bad people attempt to mess with your computer systems – so it's nothing at all to do with Microsoft's venerable Disk Operating System.

The idea behind a denial of service attack is that a bad guy wants to interrupt your service. Typically, this means 'take your website offline', but it could mean stop users from accessing anything such as email or the database back-end for a mobile app.

It's still possible in some cases for a single computer to take a website offline, but most of the time, denial of service attacks are carried out by large numbers of computers spread out

“You may remember Anonymous's DDOS attacks on financial institutions.”

across the world. These are distributed denial of service attacks.

Often these DDOS attacks are carried out by networks of PCs infected with malware (botnets), but not always. For a while it became common for people to volunteer to use their computers to DDOS sites for Anonymous.

Q Right, I think I understand what it is, but how do the bad guys go about doing it?

A Whatever server you use to provide your service has a number of finite limitations. It only has so much bandwidth, memory, CPU power, etc. If you can overload any one of these, then the server will no longer be able to function properly.

Perhaps the simplest form of DDOS is to overload the network. In this sort of attack, you just send loads and loads of data to the server. The aim is simply to clog up their network port so much that legitimate traffic starts to time-out.

Q But surely most big servers can cope with so much traffic that a few virus-infected PCs won't have any impact?

A True. However, the targets aren't always the largest sites. Also, a cunning DDOS attacker can use what's called an amplification attack. This is

where they use some way of increasing the amount of data that your computer can send. A DNS amplification attack is quite a common way of doing this.

A Domain Name Server (DNS) is what computers use to lookup information about a particular domain name. For example, if you type **www.google.com** into your browser, it sends a request to your DNS server asking what IP address is associated with **www.google.com**; then it sends an HTTP request to that IP address. However, DNS servers can be asked to return more than just the IP address. There's also a text field associated with domain names, which can hold up to 4,000 bytes. A DNS amplification attack works like this:

- A malicious computer sends a DNS request that will return a 4,000-byte text field to a DNS server, but spoofs the IP address.
- The DNS server responds with the 4,000-byte file. It doesn't send it to the malicious computer, but to the spoofed IP address (the victim server). These two steps take a 60-byte DNS request, and turn it into a 4,000 byte packet that's sent to the server. These DNS packets won't make any sense to the server, and it'll just reject them once they arrive, but the damage will have already been done.

This form of amplification allows a fairly modest collection of computers to exert a huge force on a server.

An alternative is to work smart instead of hard. In this, you don't overwhelm the server with so much data that it can't function, but you use data such that a small amount can do a very large amount of damage.

Perhaps the most famous attack of this kind is the SYN flood. Whenever you start a connection to a web server, you do a three-way handshake. This is a simple way of establishing a TCP connection over which you can send and receive data. It has three steps. Firstly you send a SYN packet to the server, then the server responds with a SYN-ACK packet, then finally, you respond to that with an ACK packet.

A SYN flood abuses this process. The attacking computers send loads of SYN packets with the IP address spoofed. The server then responds with a SYN-ACK packet, but it doesn't respond to the malicious computer's IP, instead it sends it to the spoofed IP. This computer won't respond, because it didn't send the SYN packet.

However, the server will hold this half-open TCP connection while it waits for a response. This half-open connection will lock up some of the resources of the server. If there are enough of them, even if the network isn't overloaded, the server will stop accepting new TCP connections.

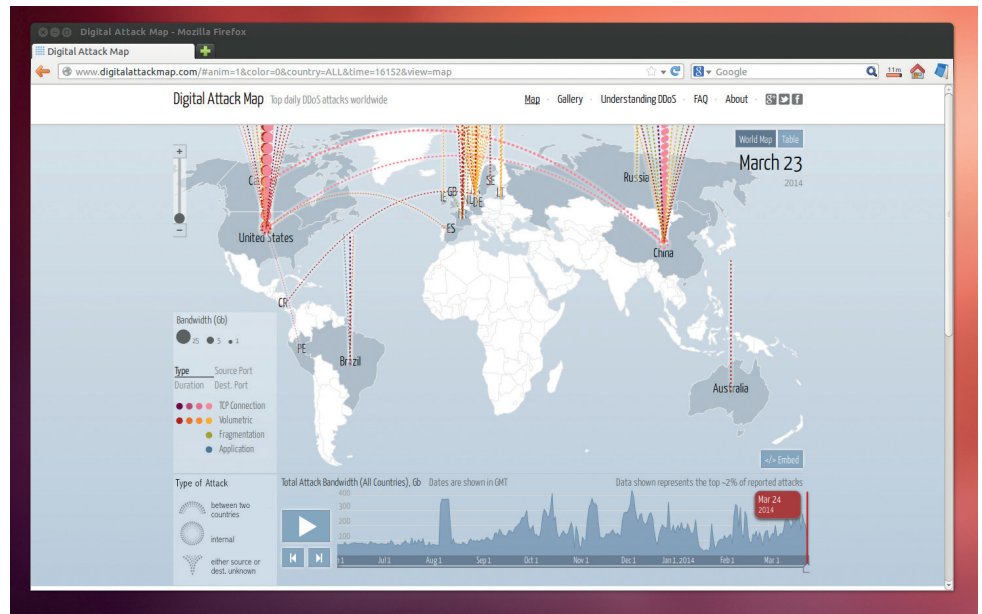
These are examples – there are many more ways to lock up resources and stop a server working properly.

Q But why do it? What's in it for the attackers?

A That varies. The most famous attacks have been politically motivated and were a show of force to try and punish organisations that the attackers felt were harming them. For example, you may remember Anonymous's DDOS attacks on financial institutions that refused to do business with Wikileaks.

One increasing area is digital extortion. In this, some internet bandits launch a DDOS attack against a site, and then tell the site that they'll only stop the attack if the site pays.

Other times, it's a business trying to cripple a competitor, or just bored geeks with a grudge. There are lots of reasons.



www.digitalattackmap.com shows a live feed of DDOS attacks, and can replay big ones from the past.

Q Hang on; people are setting up botnets to target their competition? That's a bit extreme!

A Actually no (well, a few people probably are). You can rent botnets set up for DDOS attacks, or pay people to do the DDOS for you. It's becoming quite a large industry.

Q Wow. That's scary. How bad can these attacks be?

A That really depends on how you define 'bad'. They can quite easily cripple even quite large operations. These days, a moderate volumetric attack is measured in gigabits per second, a large one in tens of gigabits per second, and a huge one in hundreds of gigabits per second. Once they get to that size, they can be pretty damaging.

Another way of looking at it is how long they last. The largest attacks burn themselves out, because few people can sustain that level of bandwidth for long. However, experience shows that there are botnets capable of sustaining large attacks for several days or longer, which is long enough to dent the finances of a web-based company.

Q What can you do to stop these from happening?

A If (for example) you're under a DNS amplification attack, you need to filter out all the rogue DNS packets, but you need to do this as far upstream as possible. The internet isn't just a randomly connected web; it has

some structure, and different connections have more bandwidth than others. The key to mitigating a network volume attack is to block it before it gets to a bottleneck. This means adjusting the filtering rules on routers at the data centre, or sometimes even at internet exchange level.

If it's some other form of attack, it means making sure that you don't waste resources on malicious packets, and again, this means identifying them and filtering them out before they do damage. Sometimes you can do this at server level, but it often means getting help from the people running the datacentre or your internet connection.

Q But my server isn't in some fancy datacentre. Is there anything else I can do?

A There is another way, and that's to route all your traffic through a very high bandwidth router that does the filtering and sends on the appropriate requests with the malicious traffic filtered out. While this may sound exactly the same as the option above, the difference is that the router doesn't have to be physically between your server and the internet.

This is known as a scrubbing centre, and it's part of what a content delivery network (CDN) does (there's much more as well). There are a few that you can use without having to change the way you host your site, such as CloudFlare, Incapsula, and SkyFaster.

DAMIAN CONWAY

We meet the creator of a programming language based on Klingon and one of the architects of Perl 6. If only we could tell them apart...

Damian Conway is one of the Guardians of Perl (our term) and one of Perl 6's chief architects. But he's chiefly a computer scientist, a brilliant communicator and an educator. His presentations are often worth crossing continents for. He was the Adjunct Associate Professor in the Faculty of Information Technology at Melbourne's Monash University between 2001 and 2010,

and has run courses on everything from Regular Expressions for Bioinformatics to Presentation Aikido (and of course, lots of Perl). Which is why, when we discovered he was making a keynote at this year's QCon conference in London in March, we braved train delays and the sardine travelling classes of the London Underground to meet him opposite Westminster Abbey.

LV The main reason we wanted to talk to you is that we want to try to simplify people's experience of programming and computers. John Horton Conway said recently that his Game of Life is the blight of his life because he had gone on to do so much more interesting and important work. But what struck us by what he said about the attraction to the game is its simplicity and the fact that that goes on to teach things that you could not possibly imagine. So with that in mind, is there something like that for programming, how does that fit with Perl, and is Perl for people that think like that in the first place?

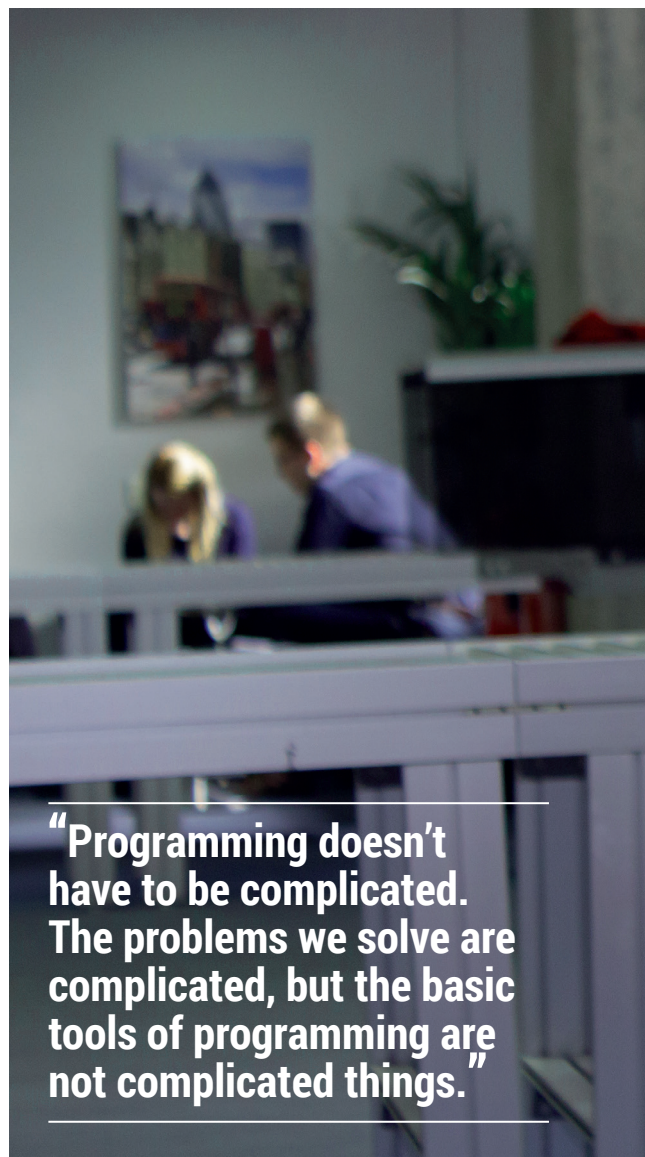
Damian: That is a huge question! There is almost an industry in making programming seem more difficult than it is. Programming doesn't have to be really complicated. The problems we solve are complicated, and at the scale we have to code things become complicated, but the basic tools of programming are not complicated things. And learning the patterns of use of those tools that work, that scale, that are robust, reliable and maintainable, isn't really that difficult. This is really not rocket science. This is not quantum mechanics. This is not that difficult.

LV But it can be. There was pride in the Perl community when you showed the Turing machine running in this much [gesticulation to show a tiny thing] code.

Damian: Sure, but that's a game. To me, that's just that I make this happen in that kind of way. It's been very interesting for me. I've recently been starting to put together classes on Perl 6, the new language in the Perl family. And the thing about Perl 6 is that it just feels like it's a lot more polished and smooth than Perl 5 ever was.

I mean, I love Perl 5 dearly, I do almost all my work in Perl 5, but Perl 6 has all of the same features but with the rough edges kind of knocked off of them. And what it gives you is the same thing that Perl 5 has always given, which is exactly the right tools to do the job you want to do and not get in your way. What I find when I change to programming in JavaScript or C++ or C is that the language itself gets in the way of my using the language.

I spend all my time coding around either limitations in the language or a particular mindset that makes you do it in one particular way, and that's equally true in Perl 5 on occasion. Perl 5 has got real deficiencies that are only just, in this very year, finally being addressed.



“Programming doesn't have to be complicated. The problems we solve are complicated, but the basic tools of programming are not complicated things.”

It's insane, for example, that in Perl 5, until the release that's probably coming out in May, we haven't had parameter lists. Now this is an advanced technology that was pioneered, what, 60 years ago, and we still haven't got them. And so everyone who's writing subroutines in Perl spends most of their time simulating the behaviour necessary for a parameter list. So finally, with Perl 5.20 coming out this year, we have parameter lists.

Every language that I code in, I find these issues. A really good one is, this afternoon I'm talking about regular expressions, and I went through 20 different languages that supply regular expression mechanisms. And in about 18 of them, the regular expression mechanism is bolted on the side, so you can't write a regular expression, you have to write a string, which then gets translated into a regular expression.



And that irritation leads to mistakes too. You don't put the right number of backslashes in, 'cause it's a string, and you've got to backslash all the backslashes to get a single backslash.

LV But, to many of us, Perl looks like a regular expression.

Damian: [laughs] Yeah, but this is kind of the same thing. If I had just gotten up on stage this morning and just shown you Klingon sentences without explaining the structure of them, the syntax of them and how they come together, then it would just look like line noise. Alphabetic line noise, but line

noise. And the thing about Perl is, in the very early design of Perl, a decision was made that there would be lots of syntactic differentiation. In most programming languages, there's only a relatively small amount of syntax. There are identifiers, there are a couple of operators and there's probably a method call mechanism, and then we do everything with that.

In Lisp it's even more extreme. In Lisp there's just comments and atoms, basically. But in Perl the decision was made very early on that we would use as much of the keyboard as possible, so that once you knew what a particular element in the Perl syntax meant, it would stand out for you immediately. So when I read Lisp, and I can read Lisp and write Lisp, and I've taught Lisp, but there's always this mental gear shift that has to go on because the language isn't helping me see what the different

components are. And I find that equally true in Python, which is a lovely language and has many many benefits. But to me, in Python, everything looks like a method call, because everything is a method call. Losing that syntactic distinction makes it really really hard for me to pick up on what's going on.

Now, the problem with that is that it only works if you know the distinction in the syntax. So people coming into Perl get lost in this sea of ampersands and stars and all sorts of other symbols that we use in the language. And until you get past and it sort of goes into your hind brain and it just translates immediately, 'ah yes, that's a scalar variable', 'ah yes, that's a type blah, blah, blah', it doesn't make sense. It looks like line noise, and I fully agree.

LV So do you think it's better for people who want to learn

“In most programming languages there's a relatively small amount of syntax.”



“Never settle for just being a Perl programmer or just being a Java programmer or just being whatever.”

programming to dive into Perl straight away?

Damian: I don't think it is. To be perfectly honest, I think Perl 5 at least is a lousy first language. And the reason I think that is that learning to program isn't just about learning syntax. It's about learning at six or seven different levels at the same time. So the purely lexical level of what character do I type here, the syntactic level of what that means, the semantic level of what does the construct that this represents mean, the algorithmic level of how do I put these things together to make things work... for me it's like when I was learning to juggle or to drive a car or any other complicated multi-level activity. If you think about learning to drive a car, it's not just about how do I steer or how do I push the accelerator pedal, it's also about how aware I am on the road, how I'm aware of what the car is doing, how do I anticipate what's happening next, how do I navigate at the same time and how do I listen to the radio as well. And for me, coding is exactly like that.

For nearly a decade, I taught the introductory programming class at our university, and I was forced to teach it in C and C++ and Java and whatever it

was. But the key is always the same. You have to give them a way of focusing on one level of abstraction at a time. And so the more syntax that the language that they're using has, the harder it is for them to focus on the level of what does this mean, what does it do and how do I make it do what I want. I think from that point of view there have been many CS programs over time that have taught Lisp as their first language. I think, in one sense, that's a really good thing, because I can tell you the syntax of Lisp in three minutes, and from then on it's just trying to understand how the mechanisms work and how the algorithms work.

So I don't think Perl 5 is a good language for that. I think Perl 6 is a better language because Perl 6 doesn't need as much syntax to get the basic stuff done. There's of acres of syntax in the background but you don't need it early on.

LV The UK government has decreed this year as the Year of Code. Its representative said that it was possible to learn some code in an hour. Talking to Robert

Lefkowitz on the subject, he thought that programming is at a similar stage to when spaces were introduced between words in Latin script, which opened up reading to more people. And, similarly, stirrups were fundamental to the feudal system because they enabled riders to wield a sword and shield.

Damian: Or the zero in the number system.

LV Yes, exactly. So is that a relevant question for Perl, or is it better suited to Python or JavaScript, say, and should we just be teaching people concepts before we teach abstraction?

Damian: Wow!

LV Sorry, I've had too much coffee this morning.

Damian: No, these are fantastic and deep and important questions. Let's go back to the very beginning. Anyone who believes you can teach programming in an hour has no idea about what programming is. I think that I finally thought that I was a confident programmer maybe about four or five years ago, so after about a quarter of a century of coding. I felt that I was an ordinary good programmer by that stage. I don't think you can even teach HTML in an hour, to be brutally honest.

LV That's one of the very examples they gave.

Damian: No, no. So there's a fundamental misunderstanding about how complicated a task it is that we do when we do programming and how quickly one ought to be able to do that task. And I think we do a disservice if we try and throw people in at the deep end. And a lot of language choices throw people in the deep end. I would, for example, put JavaScript or Java in that same category.

If you try to teach people Java, just think about the Java 'Hello World' program, you see it online all the time. The Java 'Hello World' program has a class declaration and then it has a method declaration, it has the loading of libraries that make the thing work, it then has the method call chain to actually do that. And in order to even understand the presumably simplest of all programs, you have to understand

Java at about four different levels of abstraction. You have to understand a lot of very sophisticated concepts, including things as simple as what's the difference between static and non-static. Now, a lot of good programmers would not be able to tell you what the difference between static and non-static really is. So, a language like that, which is often touted as being a relatively simple language, actually isn't.

“Anyone who believes you can teach programming in an hour has no idea about programming.”

LV So you can just dive in a change things?

Damian: Yeah, and that's what people do. They don't learn to program, they learn to evolve or mutate existing programs, and that's not the same skill set. And, frankly, a lot of Perl developers are like that as well. Their only exposure to Perl is in existing large-scale scripts on which their entire organisation depends. And all that they're asked to do is go in and make a small change to that. They're not asked to develop, to design, to build, to implement. It's strictly about “let's twiddle”.

When you're looking for a language to actually get people up and running, you need a language that doesn't get in their way, that allows them to think

about the abstractions of how to express this series of instructions clearly and unambiguously. In Perl or Perl 6, Hello World is literally “say ‘Hello World’”. The thing is, I can teach someone to do that in 30 seconds, not an hour, and I can go from there if I'm very very careful about what I introduce them to next. There are other languages where you don't have to be quite as careful because there just aren't that many constructs, and they have pitfalls as well.

What's important is that we do need good programmers, we do need people who can do this stuff, because our entire society will utterly fall apart if we do not have people that can maintain our software. We are not a society that can survive if our software goes down. But to think that we can teach them in an hour, or a day, or a week, or a month or even a year, or the three years of the standard program, is highly optimistic.

LV Does that mean that, in some way, computer science has failed if we still want people to become expert scientists, when the future promised us some pseudo-code that we could just transfer our thoughts to the computer?

Damian: Yes. The future promised us a lot of things, didn't it! I'm still waiting for my flying car.

But should programming become a commodity? Eventually, for a large number of people, it will be. We will find ways whereby people can set up their environments and have the behaviour they want. But there's a fundamental mistake there in thinking things that are that complicated can be reduced down to something so simple.

LV Considering the context of the conversation, what do you think is the ideal path? Is there an ideal language to start with? How would you recommend people get started if you want to take them to Perl nirvana?

Damian: Perl nirvana! I can probably only go by my own path and by the paths that I've shown to my of students over time. And for me, the most important thing was diversity. Not being stuck thinking this is one way that we code. And I don't care if it's the one way of Python, or the one way of Ruby, or

the one way of JavaScript, or of Java, or C, or C++ or anything. I think the important thing is that if you want to become an experienced programmer, you need to be exposed to an enormously wide range of ways of thinking about coding. You need to be exposed to functional programming systems and imperative programming systems and object-oriented programming systems and declarative programming systems and concurrent programming systems. Because it's only by opening up your mind to these different views on the same reality that you really see.

It's like back in the early days of physics where everyone either just thought of light as particles or just as waves, and there was this enormous fight over which one is it. Well, the answer is both. And is programming a purely functional activity or a purely object-oriented activity or a purely imperative activity or a declarative activity? It's all of them. And what I try to do in all of the syllabuses that I ever put together and what I try to do for myself in my own ongoing learning is find new ways of thinking about what it is that I do. How can I do functional programming in C, for example? How do I do object orientation in C? Well you can do that it. It's not easy, but you can do it. So, for me, it doesn't matter what tool I'm looking at, what I want to know is how can I think of this problem in a way that makes the solution obvious and simple and correct and robust. And often that's just I need to look at it entirely differently. And so what I would encourage every young programmer, and every old programmer as well, is never give up.

Look at the new languages that are coming out. Look at the Clojures, and the Scalas and the Darts and the Gos, and all of the different languages that are constantly coming up. See what they have to give you in the way of insights about what programming actually is. Because the only way you're going to eventually understand what this elephant looks like is if you feel the various parts of it individually and realise that they are simply parts of a greater whole.

LV **Brilliant. Thanks Damian.**
Damian: My pleasure. LV



“Ruby on Rails makes it possible for not very strong developers to build fairly sophisticated systems.”



LIBREOFFICE 4.2

FAST, COMPATIBLE, INTEGRATED

LibreOffice 4.2 offers performance and interoperability improvements, that are particularly appealing for power and enterprise users. **You can read and write old and new Microsoft Office files, and import Microsoft Publisher, Microsoft Visio, Corel Draw and Apple Keynote documents.** And you can do it faster than ever!



Test our Impress Remote Control for Android and iPhone/iPad available on Google Play Store and iTunes Store



Support The Document Foundation with a donation at <http://donate.libreoffice.org>



Download LibreOffice 4.2: <http://www.libreoffice.org/download/>



LibreOffice
The Document Foundation

AND THANK YOU FOR USING LIBREOFFICE!

www.libreoffice.org



LISTEN TO THE PODCAST

LINUX VOICE

WWW.LINUXVOICE.COM



LINUX VOICE REVIEWS



Andrew Gregory

//ART NOTE//Please run the shave and haircut filter over this photo//END NOTE//

Last week, I fixed my mum's laptop. Being of adequate intelligence with a good standard of linguistic comprehension I can normally muddle through most IT fixes, but this question (non-functioning wireless, which had apparently turned itself off) had me stumped. Google eventually told me that you can get a Start menu in Windows 8 by pressing the Windows key and X together, which is appalling – not since the early days of Gnome 3 have we seen such a silly, non-intuitive way to perform an essential function.

If you need an instructional video telling your users how to turn a machine off (<http://windows.microsoft.com/en-gb/windows-8/how-shut-down-turn-off-pc>), there's something seriously wrong with your design. At least with Linux, users have a choice, and a voice, and developers have to listen very quickly unless they want to see their projects abandoned. Gnome 3 came very close to irrelevance last year, but Darwinian inevitability has driven the project to improve. Fixes are implemented quickly, rather than grudgingly as with Microsoft's belated restoration of the Start menu. Gnome 3 is back to where it was: a usable desktop for anyone. I just need to persuade my mum to start using it. andrew@linuxvoice.com

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

On test this issue...



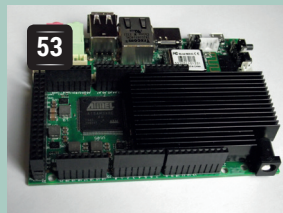
50 Bitwig Studio 1.0

We say it often, but this really is a game-changer. Audio editing and production on Linux just got deliciously better.



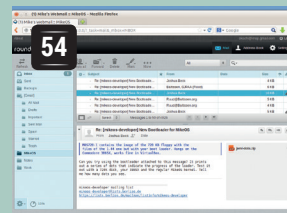
52 Gnome 3.12

If you have a shiny new Dell XPS 13 and you need a desktop to take advantage of your lovely pixel density, try this.



53 Udo

Four times the power of a Raspberry Pi, at around 3.5 times the price. Value?



54 Roundcube 1.0

The convenience of Gmail without the ads or NSA surveillance. Email bliss.



55 Pibrella

A big red button, a buzzer and three LEDs for Pi hacking. What's not to like?

BOOKS AND GROUP TEST

Touchscreens are the future. I know, because Tom Cruise told me so in *Minority Report*, and a Mexican friend of ours still makes swishing noises and waves his hands about any time he's trying to convey anything futuristic. You too can be just like Tom Cruise by using a touchscreen Linux – just read our Group Test first to find out which one's right for you.

Books are the past. I know, because self-appointed media 'experts' keep telling me. Still, they remain a popular medium for information – none more so than the excellent *Learning Python with Raspberry Pi*.



Bitwig Studio 1.0

Graham Morrison pulls himself out of rapture to write not nearly as many words as he wanted to.

DATA

Web
www.bitwig.com
Developer
Bitwig GmbH
Price
£259.99

This is a day-one Linux release of a market changing desktop music composition application. On Apple's OS X and Microsoft Windows, Bitwig Studio is causing a stir because it implements a similar workflow to an industry standard, Ableton Live. Instead of creating music by recording onto tracks, an arrangement is created by recording a sequence of triggered loops of audio and MIDI, often live and augmented with bucketloads of effects, automation and processing.

There's simply nothing like this for Linux. Bitwig Studio is a refined, minimally styled and powerful application. It's capable of full-blown music production and is a joy to work with. It's occasionally frustrating and slightly unstable – mostly because this is the first release – but it's constantly capable of the kind of audio gymnastics that a certain kind of music producer can't live without.

Jack your body

The only officially supported Linux platform is Ubuntu 12.04 LTS, which is unambitious but understandable. You can activate up to three installations, the idea being that you have a workstation at the studio and a laptop for travel, and you can activate an instance for a single session, which is useful if you're using someone else's machine.

With our i5 CPU, the Pro 40 and the Jack audio layer, we got ultra-low latencies of 2.9ms running with a Frames/Period buffer of 64. That was fast enough for realtime effects processing and monitoring on incoming audio, plus MIDI software synthesiser playback, without any hint of latency, at least to our ears. A slightly larger buffer did significantly reduce the CPU overhead of polling the audio interface, which we'd recommend if your CPU is a few years old. We also tested latency on the Dell XPS 13's internal audio, and found it perfectly acceptable for playback, being in the region of 12ms, making Bitwig and Linux a great giggering combination.

The brilliant thing about Bitwig Studio is that it features an arrangement view and a clip view that can be opened side-by-side. The arrangement view is how Audacity, Ardour, Rosegarden, Cubase, Cakewalk, Apple's Logic and countless others manage their multitrack productions. Each track of MIDI or audio is a different horizontal bar on the screen, and a mixer view usually handles effect sends and processing for each track. The clip view is where you create loops of either audio or MIDI, putting variations on the same track and arranged into groups you want triggered at the same time.

Thanks to Jack, we found the process of creating audio input and output channels, send and return

Partner packages provide an instant hit of sounds that can be dragged and dropped into your own compositions.

THE BITWIG INTERFACE

Transport control
Play, rewind, record, and enable the various latch modes for controlling the application with remote hardware.

Clip view
Uniquely, Bitwig can place the clips alongside the arrangement view.

Arrangement
Traditional DAW functional for audio, MIDI and hybrid tracks.

Browser
All instruments, clips, loops and effects are accessible through a single panel. You can also build your own collection of sounds and presets and access them from here.

Channel inspector
Colorise and manage device assignment for each channel, as well as the effects sends, volume and channel I/O.

Edit mode
Switch between the arrangement, clip and edit views and create screen sets.



Device panel
Create chains of devices made up from instruments and effects then control their parameters using your mouse or an external MIDI device. It's also possible to script your own controllers.

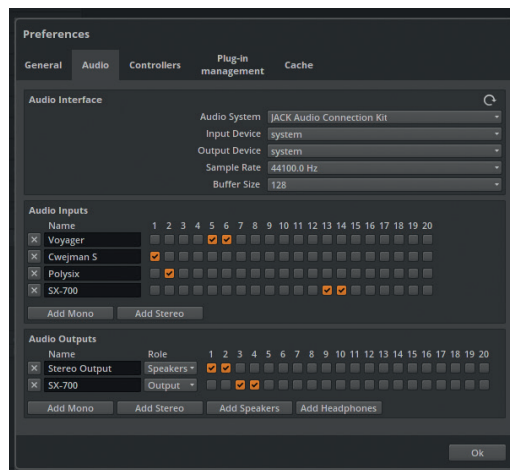
Sound preview
Listen to clips, sounds and instruments through separate output.

Hardware configuration

Bitwig Studio enabled us to make best use of our audio hardware by allowing us to create virtual inputs and output that pointed to their hardware counterparts. We would have appreciated some input monitoring to make assignments without the guesswork, but reassignments could be done without starting the audio engine.

MIDI was picked up and handled automatically, but you have to create a generic MIDI device to handle standard USB ports before they'll work. We've also got two MIDI controllers – a Behringer BC2000 and BCF2000. These are devices with lots of knobs and sliders, and Bitwig Studio comes with two patches that can be loaded into both to allow you to control the application remotely.

With the MIDI devices enabled, one of Bitwig's best features is its remote control provision. You can right-click on almost any control on the screen and then move a remote MIDI controller to make a quick and easy assignment. This is perfect for adjusting the equaliser without looking at the screen, for example, or turning the BCR2000 into a hardware controller for the built-in synth, and it obviously helps when it comes to creating a clip-launching environment, which many performers will want to do for their live work.



Our audio device has 20 individual inputs and outputs, but Bitwig is able to make sense of this configuration with only a little help from us.

effects channels, external instruments and MIDI surprisingly straightforward. Tracks can be armed for recording and enabled for input monitoring. When recording and editing loops, the markers used to delineate duration and loop points were intuitive and easy to modify. The pitch-shifting algorithm sounds good, though maybe not that creative at extremes. We missed MIDI recording quantisation, but post-recording quantisation worked well, and the automation control on tracks was brilliant, right down to the per-note level for built-in devices.

Wired for sound

Bitwig Studio comes with a lot of sounds, instrument and effects devices, all navigable through a context search enabled browser panel on the right of each view. We don't particularly enjoy the building block approach to creating music, where bass, drums and melody loops are dragged into the clip view to be reconstituted into a new piece of music, but it works well in Bitwig, and you'll be plundering commercial sample banks and libraries before you can say 'Rhythm Is A Dancer'.

Far better for us were the drum machine and synth samples, which can be used to construct your own kits using a drum kit device or quickly added to your project as a pre-built configuration. Every drum in a kit can have its own send control and dynamism, and you often find you can unfold instruments to reveal further parameters, and you can fold racks together in the same way. We found the remaining sounds a little uninspiring, but as karma dictates you should build everything from the ground-up using your own loops and samples in the comprehensive sampler device, this isn't problem for us. We're also certain that it won't be long before Bitwig's community starts sharing sound packs.

Alongside the samples, the loops, drum kits and effects, there's also a small selection of virtual instruments. There's a lovely analogue-styled polyphonic synth, a four operator FM synth, several drum sound generators and a drawbar organ, all with a considerable amount of control. Combined with the effects modularity, you can create both bread-and-butter sounds and more experimental timbres. But this is where we hit the biggest problem for this first release – Bitwig only supports plugins compiled against Steinberg's VST API, even on Linux. This is good for VST

developers who want to create cross-platform versions of their instruments and effects, but not so good for native Linux

developers. We'd love to see LV2 supported in a future release, without having to go through Jack re-routing hoops, and can we also ask for a loop-building effects plugin and a way of filtering MIDI input by channel?

We've barely scratched the surface. All we can say is that there's nothing like Bitwig Studio on Linux. If you have any interest in music composition or production, you need to try out the demo. It is relatively expensive (though not compared with its competitor), but it's the result of many years of beta testing and development. From a music production perspective, Bitwig feels like, finally, Linux has come in from the cold. 🐧

“If you have any interest in music composition or production, you need to try the demo.”

LINUX VOICE VERDICT

Amazing. A top-tier audio workstation released for Linux. Look out for a Paldandy gig near you.



Gnome 3.12

Do you have a shiny touchscreen laptop? **Ben Everard** thinks he may have found the right desktop environment for you.

DATA

Web
www.gnome.org
Developer
The GNOME Project
Price
Free under the GPL

Gnome 3.12 is the latest stable incarnation of the GNOME 3 desktop environment, which includes the GNOME Shell desktop and a range of core applications, most of which have seen some form of improvement.

One of these integrated apps is the Videos application, which has been given a thoroughly modern look with floating controls and links to online video sources. In fact, integration with online services seems to be a focus point for the GNOME team at the moment. Support for the Pocket app, which is a way for users to save online content for later perusal, has been added in a number of places, and Photos now supports importing pictures from online sources such as Facebook.

Better integration

It seemed to take a long time to migrate the Gedit text editor from the old GNOME 2.x style to GNOME 3, but it's there now. This means it fits in better with the rest

of GNOME 3 and users of the desktop environment should feel more at home in it. Of course, it does mean that Gedit now looks out of place on other desktop environments, but as it's part

of the GNOME suite, we can't really criticise it for that.

The GNOME lust for simple names has really reached a zenith (or nadir, depending on your point of view) with the naming Software. This isn't a generic term for the stuff that runs on your computer; instead it's the name of the GNOME software centre, which

has received an overhaul and is getting closer in terms of function to the Ubuntu Software Centre, which in our opinion is the best of such applications for Linux. It still has a little way to go, but it's looking good so far.

Support for high resolution (HiDPI) screens is significantly better than in 3.10, and it's certainly worth taking a look at if you're struggling to get the best out of your expensive monitor. This comes as the GNOME team have been doing excellent work in making the desktop look more beautiful. We may even go so far as to say that GNOME 3.12 has the most attractive default state of any Linux desktop.

Tip-top for tablets

Using the GNOME 3 live CD released by the GNOME foundation should be the best way to try out the desktop environment. However, unfortunately we found it a terrible experience. It completely crashed on us on several occasions forcing a reboot. Hopefully this won't be an issue once it's made its way onto mainstream distros, but we can't yet say for sure.

GNOME 3 divided opinion on its release. This reviewer really didn't like GNOME 3 when it first came out, but it's now maturing into a usable system. It's at its best on touchscreen devices (desktops and laptops rather than tablets and phones), which might seem a little excessive now but given the large proportion of computers selling now with touch screens, it's good to see that one Linux desktop environment is taking on the challenge.

Of course, the majority of Linux machines still have traditional non-touch screens. The GNOME team have managed to create a desktop that works well with both mouse and touch. Unfortunately, it's usually compared to desktops that work really well on one or the other (such as Cinnamon with a mouse or Android with touch). Those have both had a head start, but GNOME 3 is catching up in both areas. For some people, it's already the best option and we can see why, but none of us at Linux Voice are quite impressed enough to make the switch yet. Everything still seems to take one or two more clicks than it does on other desktops, and that's still enough to put us off. However, it has reached the stage where we're now considering it as an option, and that's something we didn't think would happen when GNOME 3 came out.

"GNOME 3.12 has the most attractive default state of any Linux desktop."

Application folders in GNOME 3.12 could herald the start of a shift back to a hierarchical application menu.



LINUX VOICE VERDICT

GNOME 3.12 doesn't contain enough to persuade us to switch, but users will appreciate the improvements.



Udoo

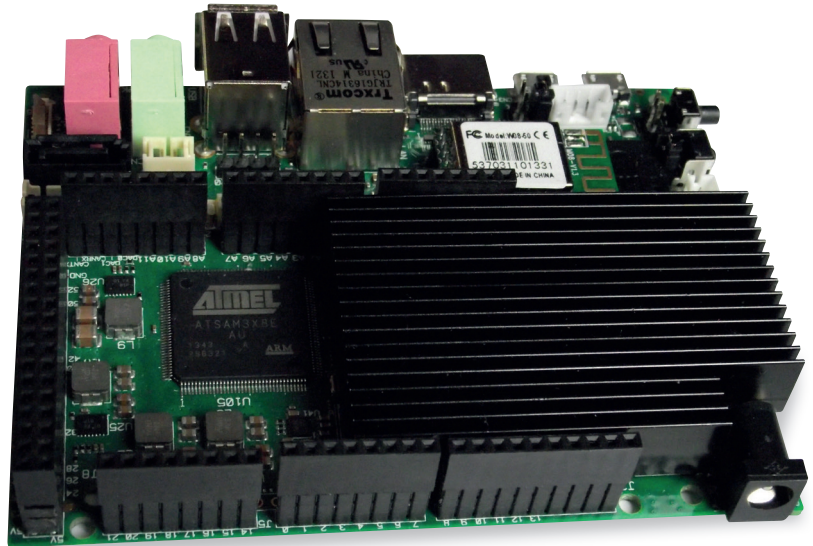
You do like small ARM computers? So does everyone these days, it seems, including Ben Everard.

The Udoo is a small ARM-based machine that runs Linux (both traditional desktop Linux and Android), and has some programmable input/output pins exposed. If that sounds familiar, it's because that's exactly what the Raspberry Pi is. The Pi has proven extremely popular, but for all its uses, it's a little lacking in hardware grunt. That's the niche that the Udoo is aimed at: simple, accessible Linux-based hacking, on a board that packs a little more punch than its fruit-based counterpart.

The CPU is a quad-core 1GHz ARM v7 (a dual-core version is also available). See the box below for benchmarks – these show that each of the Udoo's cores is more powerful than the Raspberry Pi on its own. While benchmarks provide quantitative data, qualitative data about computer performance is harder to capture. The Udoo has enough power to make the desktop feel snappy, and tasks that swamp the Pi (like browsing JavaScript-heavy websites or unzipping packages) are handled with relative ease. Put simply, it feels an order of magnitude quicker than the Pi. However, it's still no match for most x86 machines.

The Udoo uses a separate microcontroller to handle the inputs and outputs. In fact, the microcontroller and pin layout is identical to the Arduino Due, with 76 IOs including 12 analogue inputs and two analogue outputs. However, unlike most Arduino boards, the Due (and Udoo) use 3.3 volts rather than 5, so hardware designed for 5V boards won't work.

Connectivity doesn't just come in the form of IO pins: the Udoo also has a SATA connector (quad-core version only) to allow regular hard drives to connect; an LVDS connector for touchscreens (especially good if you want to build your own tablet – Udoo sells 7- and 15-inch screens); and a USB OTG connection. It also has a camera connection (camera module sold




separately). Network access is accounted for with Wi-Fi and Gigabit Ethernet on the Quad-core version.

The extra power of the Udoo comes at a cost. It's more expensive, bigger and draws more power than the Pi. All of these make it a significantly worse option for projects where the board will be included into the project physically.

An Udoo is almost exactly twice the size of a Raspberry Pi, and it packs many more connectors into that space.

Desktop replacement?

The comparison to the Pi, though, is a bit unfair. The Udoo is more than three times the cost, and while it is still cheap compared to a PC at around £110 (including taxes and shipping to the UK), that takes it out of the impulse buy range for many people.

Boards like the Udoo live or die based on whether they get enough mindshare. If there are plenty of tutorials and books available, it becomes easy to work around the limitations and compromises that are essential to all small board computers. If they don't, using them becomes more hassle than not. A quick Google search brings up about 40,000 results for 'Udoo tutorial', compared with 180,000 for 'Beaglebone tutorial' and over seven million for 'Raspberry Pi tutorial'. That's a lot less, but then the Udoo is the youngest of the three. The Udoo website explicitly pitches it as a competitor to the Raspberry Pi, and it's hard to ignore that, but we can definitely see a useful future for this device on its own merits. 

DATA

Web
www.udoo.org
Developer
SECO USA Ltd and Adilab
Price
€73–99

Udoo vs RPi performance

Benchmark	Udoo	RPi normal	RPi max overclocked
Blowfish	47.43	99.00	68.58
Cryptohash	22.39	9.07	13.28 *
Fibonacci	11.49	26.07	18.16
N Queens	41.64	84.97	69.07
FFT	48.94	149.16	101.19
Raytracing	49.02	130.38	89.84

Hardinfo benchmarks. This compares a single core of the four-core Udoo against the only core on the Raspberry Pi.

* More is better. For all others, less is better

LINUX VOICE VERDICT

The Udoo is good value for money if you're ready for a home hacking board with more power than a Pi.



Roundcube 1.0

How well does this webmail client fare against desktop apps? Mike Saunders investigates.

DATA

Web
www.roundcube.net
Developer
Roundcube team
Price
Free (open source licences)

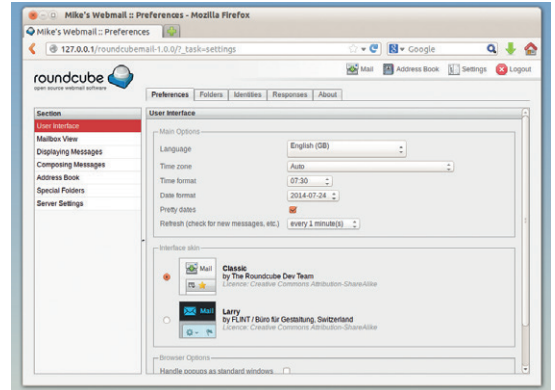
In the early 2000s, webmail used to be regarded as pretty rubbish compared to desktop applications. Searching facilities were limited, the interfaces were slow and clumsy, and you couldn't read your mail when you were disconnected. Then Google Mail came along and changed everything: it was fast, had excellent searching capabilities, and its Ajax-heavy interface made it feel somewhat like a native app. But for all its strengths, Google Mail is closed source and proprietary. If you want to implement your own webmail system, you'll need an alternative – and one of the best is Roundcube. Here at Linux Voice HQ, we've been using Roundcube extensively for the last few months, pushing it to the limits from different corners of Europe. Now version 1.0 has finally arrived, so what does it have to offer?

Fortunately, installation is straightforward: the main requirements are a web server (eg Apache) with PHP enabled. With the files in place and the right permissions set, we pointed our web browser at the

installer/ directory, which guided us effortlessly through the setup process. Roundcube can use MySQL to store its data, or SQLite as an

easier-to-set-up alternative.

Because Roundcube is just a webmail client, and not a complete solution with a mail transfer agent, you need to point it to an IMAP server for retrieving messages, along with an SMTP server for sending. Configuring the client to talk to Google's mail servers



If you're not a fan of the stock dark theme, a lighter (and more retro looking) alternative is available.

was a doddle – we were up and running in seconds. On the whole, the installation is impressively quick and polished, so Roundcube scores full points here.

Interface and docs

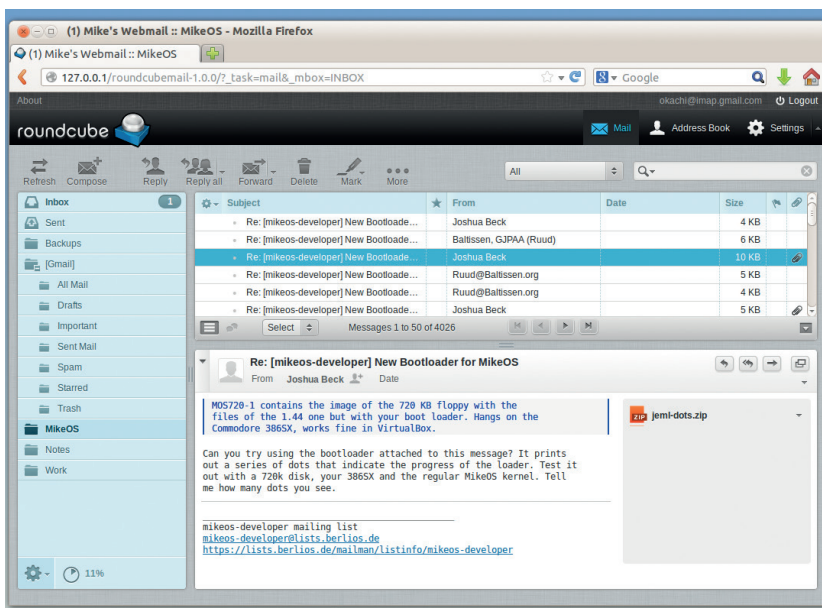
By and large, Roundcube works like a typical desktop mail client: there's a folder list down the left, message list on the right, and a toolbar on top (with buttons for composing, deleting, marking as unread) and so forth. The interface doesn't make much use of large displays by default, though, as you have to double-click a message to view it. But via Settings > Mailbox View > Show Preview Pane you can see the contents of messages from the main screen.

Feature-wise, Roundcube includes: decent search facilities (based on subject, from, message body and other fields); spell checking; drag-and-drop for moving messages between folders; both plain text and HTML composition; and an address book that can import contacts in vCard and CSV formats, or hook up to an LDAP server. It has pretty much everything you'd need in a desktop client, and a plugin system is available for some bolt-on features.

What lets Roundcube down, however, is its documentation. For an end user, it's pretty bad and limited to some scraps of information on the project's wiki. There's no comprehensive handbook or getting started guide – and this caught us out when trying to enable certain features. It's a shame, because otherwise Roundcube is a superb piece of work.

“Installation is impressively quick and polished, so Roundcube scores points here.”

This is how Roundcube looks with the Preview Pane enabled; otherwise you just get a message list in the default setup.



LINUX VOICE VERDICT

Easy to install, polished and loaded with impressive features – but the lack of documentation is a problem.



Pimoroni & Cyntech Pibrella

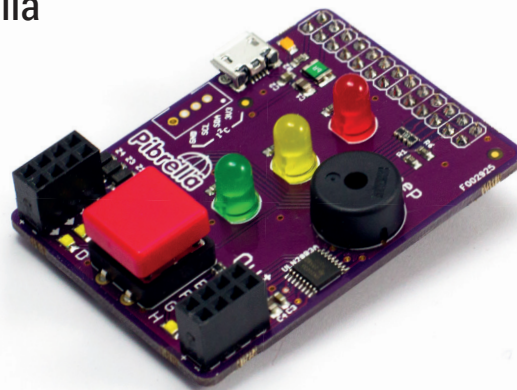
No, it has nothing to do with Rihanna, but Les Pounder lets us stand under his Pibrella... ella

Pimoroni, the Sheffield-based company of makers and tinkerers, has become the place to go to for Raspberry Pi-related kit. Fresh from the success of its popular PiGlow add-on board comes the new kid on the block: the Pibrella, which is a partnership with Cyntech Components, the company behind the Raspberry Pi logo-shaped hub.

The Pibrella is a simple add-on board that is placed on to the GPIO pins of the Raspberry Pi, and provides the user with extra methods of input and output. At first glance you can easily see two forms of output in the shape of a red, a yellow and a green LED, as well as one buzzer. There's also a big red button that looks as though it's come from the leftovers of a nuclear decommissioning programme.

If you look a little closer you'll see two banks of female connections on either side of the button. The left bank provides four extra methods of input, whereas the right bank provides four methods of output. These extra IO ports provide you with an easy way to extend the functionality of the Pibrella via the use of sensors and motors.

Pibrella can be used with two programming languages – Python and Scratch – using Simon Walters' ScratchGPIO (<http://cymplecy.wordpress.com>). Pibrella comes with its own Python library courtesy of Pimoroni's GitHub repo (<https://github.com/pimoroni/pibrella>). The library is a sheer delight



DATA

Web
<http://pibrella.com>
Developer
 Pimoroni vs Cyntech
Price
 £10

You can use the Pibrella's extra IO methods to create simple projects such as traffic lights or a reaction timer.


to use, and it really helps newcomers quickly hack together a project with minimal fuss.

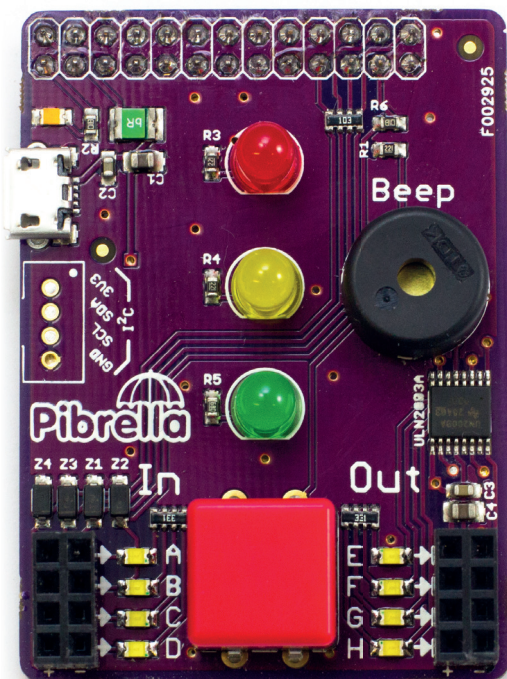
Python library

The board itself is simple to use. By importing the library into your Python code, you can easily turn lights on and off by colour using a simple line of code, so to turn the red light on and off you can use `pibrella.light.red.on()` and `pibrella.light.red.off()`. The library also provides ingenious ways to make LEDs blink and use pulse width modulation (PWM) to create a fading effect. The use of the extra IO ports is also handled via the library, and when a port is in use the corresponding LED is lit up to indicate as such, enabling you to quickly diagnose any faults.

“Pibrella can be used via two programming languages – Python and Scratch.”

It's possible to connect motors, servos and solenoids to the Pibrella, but they require a little more power than a standard Raspberry Pi can provide – have no fear though, as the team have considered this issue and incorporated a separate micro USB port to provide the additional power.

The Pibrella is the answer to a lot of our problems. It provides an easy-to-use device that enables anyone to create fun projects in Python and Scratch. The expansion possibilities are tremendous, and we can see this board becoming very popular indeed – particularly in education, as it will easily slot into the UK's secondary school curriculum. 



There's easily £10 worth of weekend hardware hacking packed into this little Raspberry Pi add-on.

LINUX VOICE VERDICT

The right features for all levels of users coupled with the right price makes this a must-have piece of kit.



Learning Python with Raspberry Pi

Graham Morrison absolutely hates this book. It's rubbish. Honestly.

First, a little disclosure. We have a vested interest in the failure of this book. Its co-author, Ben Everard, is a co-founder of this very magazine. He's one of its most technical, most entertaining and most erudite authors. If this book is a success, he's going to want to write another, and another, and another. He'll no longer have time to sew LEDs into his bike jacket, or brew alcoholic ginger beer, or cycle across minefields. Before we know it, he'll be packing his bags and jumping on the first stage coach out of the Shire to make his fortune in Wolverhampton. And Linux Voice will have lost one of its best contributors.

Despite all this, we can't help but admit that *Learning Python With Raspberry Pi* has its moments. To start with, it hits the potential target audience straight on the head; you've bought your first Raspberry Pi, you want to start using it for your own projects. Python is to the Raspberry Pi what BASIC was to Acorn's BBC. It's the *lingua franca* of the Pi generation, which we know isn't a coincidence.

Python has a similar immediacy to BASIC, and rewards experimentation. It's fun and it's forgiving. And like the Pi itself, Python can scale far beyond humble beginnings. Just take a look at our guide to controlling virtualisation (p94), or Ben's own tutorial on genetic algorithms (p104) – both use Python because it's the best tool for the job. Python may flatter by starting off simple, but there's no limit to where the language might take you.

The (very) few good bits

We love the way the book jumps straight into practical examples, forgoing the ceremonial respect usually given by describing a language by its syntax and conditional statements. Within the first two dozen pages, you're writing code that does stuff and draws things on screen, while at the same time, teaching you essential concepts about programming. We can only imagine this was Alex Bradbury's idea.

It's also the way the majority of the book continues. Any theory is always backed up by practical examples, which slowly get more advanced as they dive into more and more of the Raspberry Pi's potential.

For us, this is the best way of learning a language, because there's very little theory without an example, and as a reader, you want to expand upon what you've learnt. Each new concept comes as part of a project that teaches you something about the Raspberry Pi; develop your own web browser, write a platform game, generate OpenGL 3D graphics and script Minecraft.

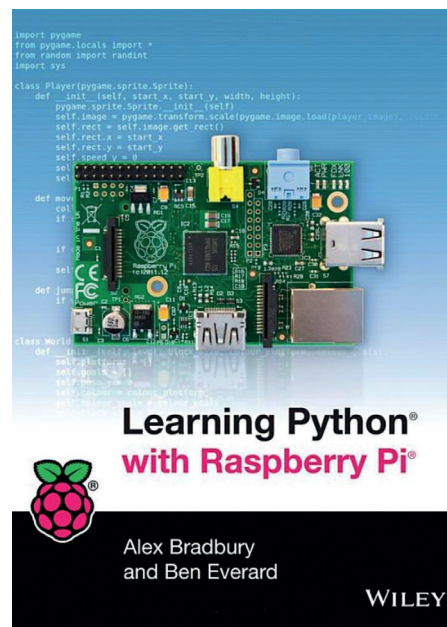
Later chapters deal with networking, hardware interfaces and debugging, basically covering every aspect to programming in Python without labouring in theory or too much detail. Each chapter finishes with some suggestions for taking things further, and sometimes a few exercises, as well as a summary of what's been covered. Even if you've never done any programming before, you should find everything easy to follow, and we also think the book will work well if you go through the examples with an older child, for example, or as part of a Python and Pi primer course.

Lousy food, and such small portions

If there's a criticism, it's that we think the book could go further. It won't take too long to work your way through its 270 pages, and there's perhaps a little too much emphasis on gaming. But it's something of a compliment to say you want each chapter to give you more. It's not enough to write a speech recognition program in 10 lines of code - we want to spend Friday evenings chatting politics with our Raspberry Pi! We want to play chess! We want more than a paragraph on robots!

It also ends quite abruptly, and while both the internet and this very magazine are full of new projects to try, we can't help but feel a little sympathy for the Python beginner who dutifully works their way through the book only to be dumped unceremoniously out of the end with little more than a link to <http://docs.python.org/3> for comfort.

But really, we only wrote that section to inject a little pseudo objectivity into the review. What we want from a book like this is for it to pique your interest without scaring you off, and to capture the essence of what both the Raspberry Pi and Python are capable of. *Learning Python With Raspberry Pi* does both and leaves you



Whatever you do, don't buy this book.

wishing the authors had written more, which we suppose leaves open the potential for slightly more advanced sequel.

Don't give up the day job

Until now, there wasn't an easy, entertaining and educational resource that would do justice to the pairing of Python and the Raspberry Pi. It's perhaps no coincidence that one of the early chapters in this book deals with Turtle graphics. We fondly remember getting an Acorn Electron in the early 80s, and it came with two books – one was *Start Programming with the Electron*, and the other was entitled *Turtle Graphics*. Together they encapsulated the same sense of wonder and exploration as Ben and Alex's book, nurturing a new generation of coders in the process. This book might do the same.

Just don't tell Ben.

LINUX VOICE VERDICT

Author Alex Bradbury and Ben Everard
 Publisher Wiley
 ISBN 978-1-118-71705-9
 Price £17.99/US \$29.99/CAN \$35.99

An excellent book for beginners to both the Raspberry Pi and the Python language.



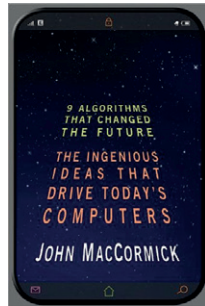
Nine algorithms that changed the future: The ingenious ideas that drive today's computers

Ben Everard wonders if all nine are required to look at Facebook.

This book is written to bring the idea of algorithms to the masses. Without assuming any computing knowledge, John MacCormick takes the reader through nine algorithms such as PageRank, Zip compression, and digital signatures in just over two hundred pages.

To achieve this the book simplifies – a lot – and sometimes it just changes algorithms to suit his purpose. For example, the chapter on public key cryptography actually deals with the Diffie-Hellman key exchange, which is private key cryptography.

Nine Algorithms... tries hard to deliver a simple description of how algorithms work to a non-technical audience. In this aim, it succeeds. It walks the reader through the algorithm and explains their basic function in a very readable manner without challenging the reader much. However, the writing is so overtly non-technical that it's off-putting to people with



How does John MacCormick know that the future's been changed?

even a slight interest in computing.

As an introduction to a fascinating subject, though, it's great.

LINUX VOICE VERDICT

Author John MacCormick
 Publisher Princeton University Press
 ISBN 978-0-691-15819-9
 Price £11.95

An easy to read introduction to algorithms for a non-technical audience



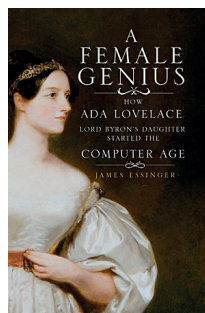
A Female Genius: How Ada Lovelace, Lord Byron's daughter, started the computer age

Ben Everard is building a difference engine out of papier maché.

This is a book about Ada Lovelace the person, not her contributions to computer science. This isn't a criticism, just a fact that we feel needs stating, especially as the subtitle *How Ada Lovelace, Lord Byron's Daughter, Started the Computer Age* implies otherwise. It actually covers almost everything about her life other than the technical details of her work.

Linux Voice readers, of course, will already be well versed in the mathematical aspects of her life from a tutorial in issue 1, so this book is the perfect companion to that. It chronicles her life from a baby growing up in the shadow of her father's scandalous life, to her untimely death. In doing this, it adds some colour and context to the cold, dry mathematics for which she is most famous.

Some people may feel that the details of a scientist's life aren't important, and only their contribution to the subject should be



Ada Lovelace was simply a genius. It's a little unfortunate that this book feels the need to qualify this with the adjective 'female'.

considered – this isn't a book for people like that. We, however, enjoyed it greatly.

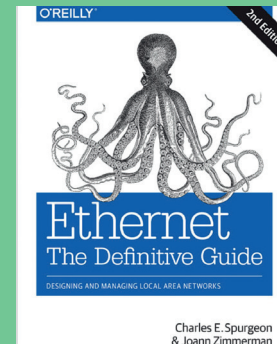
LINUX VOICE VERDICT

Author James Essinger
 Publisher Gibson Square
 ISBN 9781908096067
 Price £14.99

A thorough exploration of the Countess of Lovelace's life, this book tells the story of how the computer age almost started early.



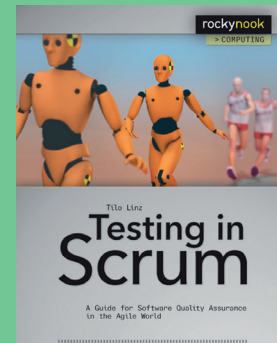
ALSO RELEASED...



Learn all about Ethernet and give yourself job security for life.

Ethernet: The Definitive Guide

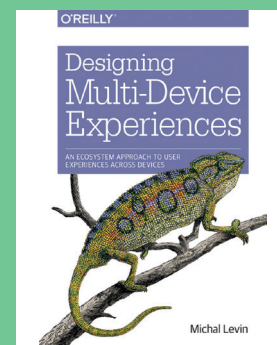
Cities, airports and rail stations are so often judged on their free WiFi, but the backbone to all networking is still Ethernet and that's not about to change. Good job that its definitive guide has just had an upgrade then, and the 2nd edition should be a great bit of research.



Scrum: the chance to laugh at co-workers every single day.

Testing in Scrum

Agile is one of those ideas that seems to have found itself into all kinds of management structures, regardless of whether they have anything to do with development. This is book promises practical help on testing and QA, and includes several case studies. Now sit down.



Linux is everywhere. Unfortunately, KDE has yet to follow.

Designing Multi-Device Experiences

These days, we expect our web experience to be similar to the supporting app experience, which is similar again to the desktop experience on all platforms. That's a tough challenge, and this book promises a practical approach to developing your own framework.

LINUX VOICE

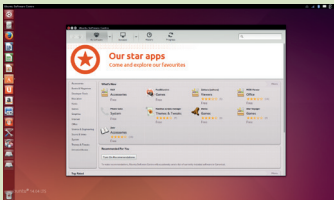
TOUCH
DESKTOPS

GROUP TEST

Graham Morrison and his trusty touch laptop explore the cutting edge of Linux desktops.

On Test

Unity



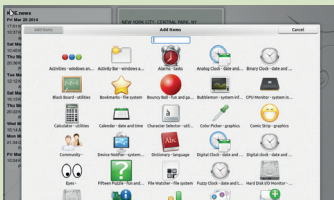
URL www.ubuntu.com
Version 7.1.2
Licence L/GPL v3
Promised touch enhancements didn't make it into 14.04, but does Unity still do enough?

Gnome



URL www.gnome.org
Version 3.9
Licence L/GPL
Gnome looks a little like Unity, but without the tablet and smartphone emphasis.

Plasma Active



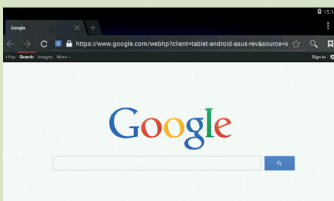
URL <http://plasma-active.org>
Version 4 (from Kubuntu daily)
Licence GPL
It's KDE with added touch and less KDE.

KDE



URL www.kde.org
Version 4.13 beta (Kubuntu)
Licence L/GPL
It's KDE.

Android x86



URL www.android-x86.org
Version 4.4 RC1
Licence Apache 2
It's just like the phone and tablet OS, only it's running off your laptop.

Touch Desktops

Don't let your touchscreen go to waste.

This is a technology that is still on the very cutting edge of what Linux desktops can do – desktops designed to be used with a touchscreen. There are several important reasons why we're doing this now, rather than waiting an indeterminate amount of time for the technology to mature.

The first is that it's fun. New technology, and new ways of interacting with it, is exciting, and Linux is going to have to find a way to work with touch. The second important reason is that the technology is already here, not just in the form of Android tablets, but increasingly in our laptops. Thanks to Microsoft's emphasis on touch for Windows 8, many new laptops now come with a touchscreen by default, and if you install Linux on one of these devices, you'll want to know which desktop is going to

work best. The third reason is that the touch interfaces of Apple's iOS and Google's Android have shaken up the old launch menu and file management desktop metaphor, and many newer Linux desktops have incorporated some of their features already.

Even if they're not designed specifically for touching, it's good to know whether the new style of design works with new hardware, or whether touch gets in the way. Which is exactly the challenge we've set ourselves for this group test.

We spent a few weeks with our multi-boot system playing with each desktop as we would a desktop in a real production environment. That meant we missed the latest GNOME release (see p52 for our review), but it also meant we took a pretty ruthless view on whether touch worked.

“New technology, and new ways of interacting with it, is fun.”

THE CRUCIAL CRITERIA

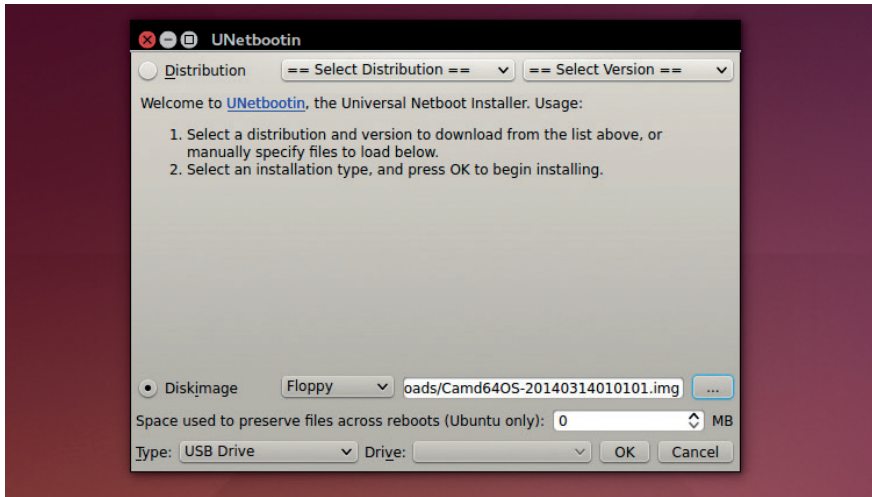
Our target is the standard x86 PC, rather than pure touch devices. We're using Dell's XPS 13, as reviewed last issue and as lent to us by www.apt-net.co.uk (thanks Alan!).

As such, any desktop can be made to work with a touchscreen, but we're not going to look at every desktop. We're going to pick those we've found to be the most effective. GNOME 3.x and Ubuntu's Unity are two obvious choices, because while they're not designed

specifically for touch, they borrow heavily from the full-screen design of Android and iOS. We can also look at Android itself. The latest x86 release of Android 4.4 works brilliantly and offers the other side of the touch coin – a touch desktop shoehorned into a laptop. For the others, we're going to use a base of Ubuntu as this ensures hardware configuration isn't the differentiator – only the way the desktops are designed to interact with touch input.

Installation and configuration

Working at the cutting edge isn't always easy.



Plasma Active is tricky to install on x86. We used Unetbootin and a recent Kubuntu daily image.

The Plasma Active and Android desktops we're looking at are so cutting-edge that they can't be installed in the way you may be used to. Not only that, you're going to need a more traditional Linux distribution installed alongside for those times when you don't want to be dealing with what are 'works in progress'. You won't have these problems with Unity, Gnome or KDE, but it is something you have to deal with when installing Plasma Active and Android, as both are different to most Linux desktops.

Plasma Active is best described as a remix of KDE for touchscreens. But it's not just a skin. It takes over the entire system and doesn't work particularly well installed alongside other KDE Plasma workspaces (as they're called), at least not in the way it's currently distributed.

Needs attention

Plasma Active in general suffers from a lack of love, despite early success, and it's a struggle to find a working configuration. As such, installation is best done through a custom Kubuntu re-spin, or by adding package repositories to vanilla Ubuntu or OpenSUSE. We went for the Kubuntu spin written to a USB stick before committing it to a section of the hard drive. Similarly, we dropped the USB image of Android 4.4 onto a USB stick booting with Easy2Boot,

and it worked amazingly well from there, including both multitouch from the screen, keyboard control and WiFi (an important consideration for Android), as well as touch control when needed.

For other desktops, the challenge is getting the touchscreen to work well, as most will be able to use touch as a mouse cursor. The best strategy is to find the very latest version of a distribution, as this will include the latest drivers. This approach worked for the Haswell XPS 13 for all the distributions and desktops we tried, especially as the XPS 13 Developer Edition originally shipped with Ubuntu 12.04, but this will also depend on your hardware.

Our touchscreen presents itself to the system as a multitouch Synaptic touchpad, which means it works out of the box. But this can also add complications if you're using a genuine Synaptic touchpad alongside the screen. When you combine these two aspects together, Ubuntu's Unity is the only desktop to have taken both the installation and configuration into consideration, by its nature, with Android coming a close second.

VERDICT

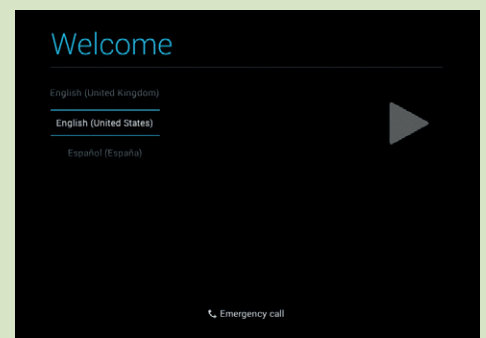
Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

Touch input

You've got the touch.

The Android-x86 build has come a long way since we first tried it a couple of years ago. It might initially seem counter-intuitive to install something designed for tablets onto what is essentially a touchscreen laptop, but we really enjoyed the results. It's like a very fast Nexus with built in keyboard. All the gestures from your smartphone work instantly, from sliding down notifications, or swiping across desktops, to pinch zoom, rotation with no further configuration. It's tough to write about Android as a legitimate alternative to a more traditional Linux distribution, but if it brings extended functionality to your touchscreen and you enjoy using it, we don't see the problem. It's still Linux.

Second to Android this time is Unity. This is because it does some sensible things to take into account the touch input. You can scroll up and down lists, for example, resize a window with three fingers and the cursor is hidden when you touch the screen. Those features alone put it in a different class. Plasma Active is pretty good, as you'd imagine, and KDE is acceptable, but not without modifications. Gnome running off both Fedora 20 and Ubuntu almost manages it. But only some window title bars register a touch, leaving certain windows unable to move without resorting to the touchpad.



Android comes closest to just working.

VERDICT

Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

Customisation

If it's not great, just what can you do to make things better?

The point of this group test is to see which desktop environments have implemented features that best work with touch. But it's also possible to change a great deal about how they look and behave even if they don't support touch. KDE comes out best because there's just so much you can change. You can dramatically increase the width of the scrollbars making them much easier to grab and move with your fingers. You can change the size of the title bar, and replace the icons with much more finger-friendly options.

A cut-down number of options is also available to Plasma Active users. Android is seriously restricted by not running traditional Linux applications, but this can be helped a little by using an open source repository such as F-Droid, or dual-booting your machine. Gnome has plenty of plugins and themes that can help, while there's not too much you can change in Unity.

Plasma Active also has a complete set of widgets that can be added to a specifically designed fullscreen background. These widgets, like the ones you find in Android, are a great way of creating something of the tablet experience with a Linux desktop, and they work well with touch and the widescreen form factor. We should also be able to pull down a task manager in Plasma Active, but in the three different x86 installations we tried, this doesn't work. Instead, we had to rely on KRunner to launch and configure the desktop.

In Unity, however, swiping down over the top-right corner of the display was enough to reveal the options of whatever icon we happened to touch. This was probably a side effect of touching the icon, but it's very similar in behaviour to the latest Ubuntu Touch builds running on phones, where you can slide down and horizontally to switch between the settings for those widgets running. Unity definitely has the best potential, and we just hope that the work that's gone into the mobile version isn't lost on x86 users, even if it's through a third-party repository.

VERDICT	
Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

Usability

Does a touchscreen actually add anything?

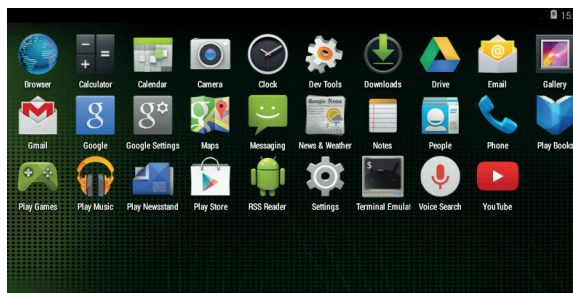
Here's the rub (sorry!) – just because your hardware has a touchscreen, it doesn't mean you should feel duty-bound to use it. If you are going to use it, the desktop has to make it worthwhile. We have used touch and keyboard devices for a few months, especially when travelling, and

our conclusion is that a touchscreen can genuinely help in some very specific cases, and the amount that they help is down to the desktop. In Android, for example, a gesture to open the settings makes a lot of sense. And Unity is obviously working on phones. But that's where all the others need most work.

Android ★★★★★

Not surprisingly, Android excels at touch. After all, input has been designed around fingers rather than mouse and keyboard input. But what most surprised us is that it feels very natural behind a laptop form-factor. You find yourself automatically launching apps and swiping through running processes by touching the screen, while at the same time using the keyboard or even reverting to the

touchpad (it works!). And using the computer in this way is quicker and more efficient than doing similar gestures the old fashioned way, or even though a launcher such as Gnome Do. As a laptop operating system, it feels much more mature than Chrome/Chromium OS. Though we're loath to use the word, there's some synergy between the touchscreen and the keyboard.

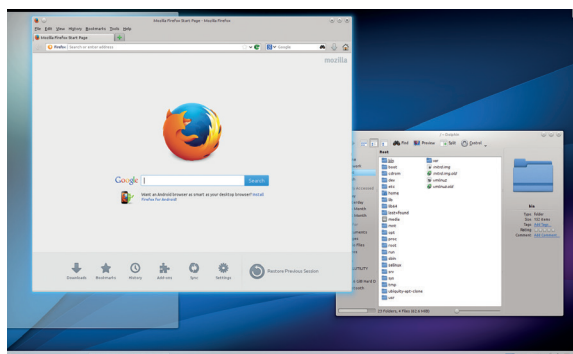


Android x86's cutting-edge nature means not everything works – Netflix, for example. But the touch experience is second to none.

KDE ★★★★★

By default, most things in KDE are small. This makes them difficult to use from a touchscreen. Clicking on an app to launch from the menu is difficult, for example, and resizing windows is almost impossible, although moving windows is slightly easier and is the only mouse function with any touch advantage potential. The single-click option for

launching an application associated with a file is useful, and KDE is perhaps the strongest desktop when it comes to configuration options. You can change almost anything about the desktop to make it more touch friendly. But perhaps because it's an area the KDE team might feel is covered by Plasma Active, concessions to touch in KDE are very few.



Without customisation, the touch KDE experience is difficult and clunky. You're better off with a mouse.

Unity 14.04 ★★★★★

Hitting the scroll bars at the side of the Unity desktop is a problem, as too is resizing a window from the bottom right corner. However, one of Unity's best features is multitouch support, and this works on the screen. Place three fingers on a window and you can move it around,

resize it using large anchor points or maximise and minimise without any difficulty. Tap three fingers and you can switch applications. A four finger tap will open the dash or bring the launcher back from a hidden state. These gestures transform Unity's touch possibilities.

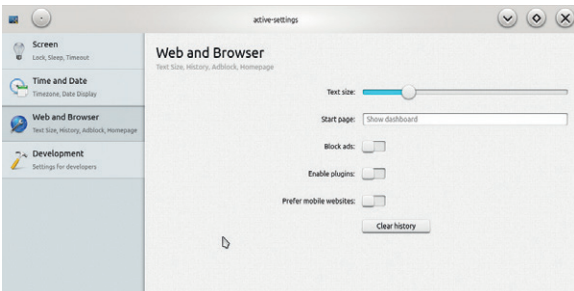


Most icons in Unity are large enough to prod, especially when it comes to restarting or shutting down your machine.

Plasma Active ★★★★★

There are many things to like about Plasma Active. It's the only big Linux desktop, bar Android, to have been designed for touch devices, so it has the potential to be the best of both worlds. By default, window management is easy thanks to the large icons and scroll bars, although there's nothing as

comprehensive as Unity's multitouch support. The large widgets you can place on the background can go some way to make Plasma Active feel like Android, and there's a primitive settings dialog that presents its options in large, slidable controllers. There's a customised version of KDE's web browser too.



Plasma Active has plenty of potential, but a lack of development is holding it back from being a usable desktop.

Gnome ★★★★★

Gnome has some of the advantages of Unity, thanks to its launcher and panel. Applications and icons are easy to locate, and the application launcher view, along with its containers for other applications, is an excellent mechanism for navigating to the tools you want to run. It's difficult to close and resize windows without

changing the theme, and Gnome is really missing the multitouch features of Unity, but it's better than KDE because the shell makes more sense from a touch perspective. Thanks to the click zones being up against the edge of the top panel, you can also swipe down on the screen to do things like logout or open the settings.



Gnome is hampered by some of the window borders not working in the same way that others do.

Touch potential

Can touch help you do more?

This is going to depend on what you need to do. Android, for instance, is hobbled by not being a traditional Linux distribution. You can't install many of the applications you may be used to, and many tasks are impossible. There are no tools for specific kinds of software development, or 3D animation, or any number of other tasks. But there are many everyday tasks such as web browsing, writing words, staying on top of emails and playing Angry Birds that are arguably better accomplished on Android than on a Linux desktop.

Even the Rotate Screen feature was useful, as it was a perfect way of proofing pages for this very magazine. We'd love to see an Android mash-up where you had X11 desktop functionality and package compatibility. However, touch on other desktops can be more productive. Web browsing in Plasma Active, file management and app launching from Unity, and cursor placement in all of them makes a touchscreen worthwhile.

Plasma Active has a lock screen designed for touch, so you can easily slide open the screen when you want to resume a session. Android does the same, obviously, but it feels like the right thing to do – much better than flicking the Caps Lock on the keyboard, which is what we usually do to resume a sleeping laptop. The problem with Plasma Active is that everything is just too half-finished to be useful. Which is disappointing, because when we tried earlier versions actually running on a tablet, usability was top-notch. The problem we had then was with performance and efficiency, not usability, while running on an ARM CPU.

Gnome had a problem where we couldn't move the settings windows with the touch screen. This might have been a hardware issue, but as we experienced the same problem on both Fedora and Ubuntu, it might be a deeper issue with the different ways Gnome handles window management, or it might be interpreting touch differently.

VERDICT

Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

Productivity

Is touch a gimmick, or can it help you do real work?

During the course of this and the previous issue, we spent a considerable period of time with each desktop, doing the stuff that we normally do. We wanted to see whether the presence of a touchscreen would change the way we interact with the Linux desktop.

We've probably written more words about Android in this group test than we have about the other desktops, and that's because Android has surprised us by being remarkably productive. When you pull it out of the CPU- and memory-restrictive environment of a smartphone or a tablet, and put it onto a fully fledged Intel Core i7 CPU with 8GB of RAM, it flies.

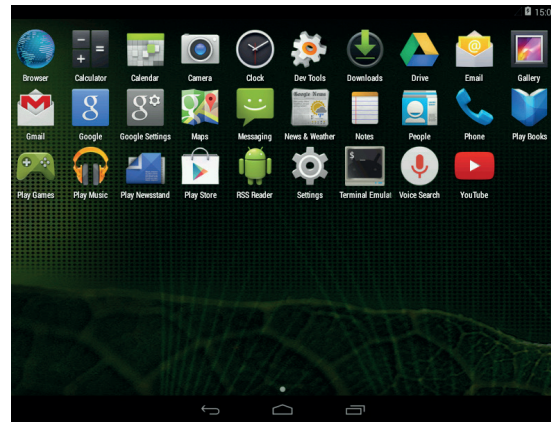
Of course, the problem is that it's only going to be good for certain tasks. In particular, it doesn't make much sense when you can pick up a tablet for very little money in comparison to a laptop. It's not a stable operating system, and has quite specific hardware requirements, so it's not going to be a good choice for most people. If you're using a computer with a touchscreen,

the chances are you want to do some serious computing with it, so while Android is great at certain mobile-friendly jobs, it wouldn't be our desert island desktop.

Multitouch FTW

Thanks to its multitouch support, the issues of resizing and moving windows don't affect Unity, as you can just push three fingers across the screen to do what needs to be done. The same configuration can be made to work with other desktops, but that requires some tinkering. One slight hitch might be whether multitouch makes its way to the Mir display server, but considering Canonical's big push on mobile and tablets, it's more likely that touch will get better rather than worse for future versions of Unity.

We were able to work with both KDE and Gnome, but these desktops were used almost 99% of the time through the mouse and keyboard, reverting to touch only for cursor placement and occasionally to launch an app or editor in KDE. It's still better than nothing, but



Android made a shockingly good, and surprisingly productive, laptop operating system.

hardly worth buying a laptop with a touchscreen for. Finally, there's Plasma Active. The biggest problem we had with this was stability, so while some aspects were better than other desktops – such as the browser and settings support – we couldn't rely on the desktop enough to do any real work. Our time was better spent trying to make KDE look more like Plasma Active.

VERDICT	
Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

Third-party support

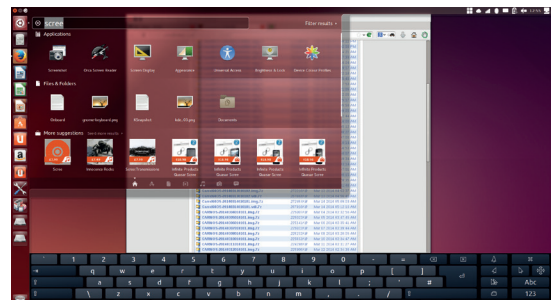
Can we make the touch experience more accessible?

There are many things you can do to make any of these desktops more friendly to touch. Gnome, Unity and KDE can quickly improve the touch experience in Firefox by installing the 'Grab & Drag' extension, for example. This replaces Firefox's default action of left-click text selection with a hand that grabs and scrolls the web page. It can even add momentum for the full tablet experience, and there's simple support for page up and page down gestures. This extension alone transforms the non-Android desktops. Even an on-screen keyboard can help, when you don't want to move your hand down to the keyboard.

Our favourite was Ubuntu's, which can be launched by typing **onboard** in

the shell. It was touch enabled without any further modification and can be latched onto the edge of the screen. It also has some useful features such as snippets and typing assistance/ auto-correction. The keyboard can also be hidden automatically, and quickly brought back to life by keeping a hovering icon on-screen.

We also liked Gnome's 'Florence' keyboard. It's scalable, has a touchscreen input mode, and you can clearly see when you've hit the key you're aiming for thanks to the focus zoom feature. Both Florence and Onboard can be set to have a transparency, so they don't have to get in the way of the remainder of the display, and while it may sound slightly crazy when there's a perfectly usable



Ubuntu does a great job at making touchscreen input a central part of its desktop experience.

real keyboard beneath the display, we found an on-screen keyboard to be almost essential for some tasks. Finally, we'd describe KDE's own on-screen keyboard as functional rather than useful for touchscreen users.

VERDICT	
Android	★★★★★
Unity	★★★★★
Gnome	★★★★★
KDE	★★★★★
Plasma Active	★★★★★

OUR VERDICT

Touch desktops

Touch hasn't changed our desktops in the way we thought it might, but touch input is something developers and users still need to consider. Microsoft, for example, got things spectacularly wrong with its unified touch interface with Windows 8, backtracking to a more traditional appearance with each update. The open source community has taken a more pragmatic and sober approach, which we think has paid off, despite early versions of both Gnome 3.x and Unity seemingly

had to check to make sure it was still being developed. There's no easy way to install it, and very little documentation on using it. It could be a great initiative, but unless there's some reason for developers to get behind it – such as Aaron Siego's wonderful Spark tablet idea – it's not going to happen.


KDE comes next, although a properly configured KDE desktop for Linux would score higher. This result is purely because there's very little evidence of any changes being made to accommodate touch. Next

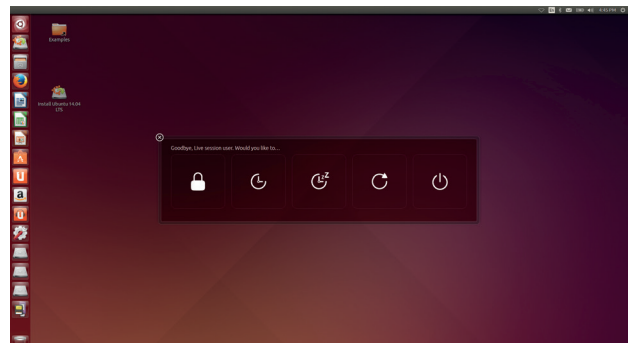
“The Unity launcher, on-screen keyboard and panel all work well with touch.”

embracing the idea. Touch isn't going to change the desktop overnight, but nor is it going to be a passing fad. The technology is seeping into standard PC/laptop/hardware, and Microsoft's hardware partners are determined to push for more tablet/PC convergence. As such, there's still a long way to go for the Linux desktop to be considered touch friendly.

We had highest hopes for Plasma Active, because it seems to be the only mainstream based Linux desktop taking touch seriously. But so little has happened over the last 12 months that we

comes Gnome. Finding and launching applications is good, as too is the on-screen keyboard, but there are some gotchas – such as some of the windows not responding in the same way to touch control.

Despite our falling for Android x86 4.4, we've only ranked it second. This is because Android x86 has no crossover with a traditional Linux distro. Unity wins because there were no big problems, the launcher, on-screen keyboard and panel all work well with touch, and the multitouch module makes all the difference. 



Unity is still easily the best Linux desktop for a touchscreen.

1st Unity

Licence GPL & LGPL v3 Version 7.1.2

www.ubuntu.com

Ubuntu is perhaps the distribution with the greatest motivation to make touch a central part of its desktop. And it shows.

2nd Android x86

Licence Apache 2 Version 4.13 beta (Kubuntu)

www.kde.org

We loved Android on a laptop. If only there were a way of installing the applications we're more used to, it would win.

3rd KDE

Licence GPL & LGPL Version 4.13 beta (Kubuntu)

www.kde.org

Just pulls ahead of Gnome because it's more configurable, and when you make the scrollbars large, it's very usable.

4th Gnome

Licence GPL & LGPL Version 3.9

www.gnome.org

Despite looking like Unity, there's no particular touch consideration in the GUI, but it has lots of potential.

5th Plasma Active

Licence GPL Version 4 (from Kubuntu daily)

<http://plasma-active.org>

We feel bad putting this last. We've used a build on an ARM tablet that worked brilliantly, but x86 is lacking love.

YOU MAY ALSO WISH TO TRY...

Any Linux desktop is going to be malleable enough to work with a touch interface. They can all be configured to use larger buttons, or place large launch icons in places that fingers will find easier to hit. Mate and Cinnamon, for example, both work well in our tests, as would XFCE. But there's nothing specifically touch friendly about them, which is why we

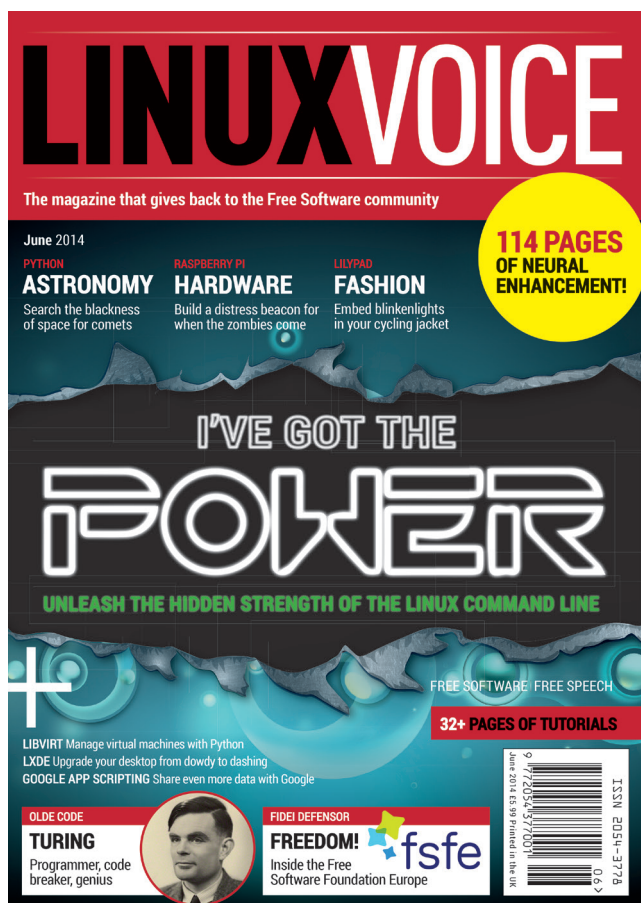
didn't make them part of this group test. Google recently announced a new Chromebook, the Acer C720P, and it comes with a touchscreen. It's running Google's own browser-centric Chrome OS, an OS that falls into the same category as Android for being non-standard in the way other Linux desktops are. But there is an open source build of the

operating system and we spent a considerable amount of time getting this to work on the XPS 13. As almost everything is browser based, touch helps when scrolling around and hitting links, but doesn't offer anything beyond Firefox with a touch plugin, but it may be worth a try if Android has given you a taste of cutting-edge touch desktops.

SUBSCRIBE

shop.linuxvoice.com

Not all Linux magazines are the same



Introducing **Linux Voice**, the magazine that:

LV Gives 50% of its profits back to Free Software

LV 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 – £55
- ROW – £60

DIGITAL
SUBSCRIPTION
ONLY £38

Each month **Linux Voice** includes 114 pages of in-depth tutorials, features, interviews and reviews, all written by the most experienced journalists in the business.

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

NEXT MONTH IN

LINUX VOICE

ON SALE
THURSDAY
29 MAY



THE BEST UBUNTU RESPIN

The world's favourite Linux distro has evolved to the next level. We look at the spin-offs that have grown up and flown the nest – Kubuntu, Elementary OS, Trisquel and more.

EVEN MORE AWESOME!



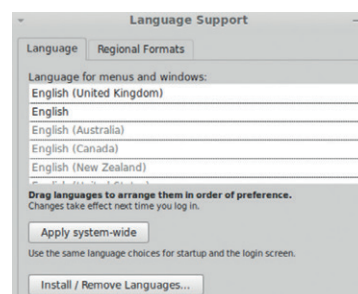
Old Code

Random numbers, DNA, nuclear weapons, the first reprogrammable computer and more – John von Neumann had his fingers in many pies.



Life on Mars

Build a Mars Rover with a Raspberry Pi*; seek out new words and civilisations. (*This is an artist's impression of Ben's Mars Rover.)



No speako lingo

Internationalisation is a massive reason for Linux adoption – get involved, translate something and quite literally spread the word.

LINUX VOICE IS BROUGHT TO YOU BY

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

Editorial consultant Nick Veitch
nick@linuxvoice.com

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

Printed in the UK by
Acorn Web Offset Ltd

Disclaimer We accept no liability for any loss of data or damage to your hardware

through the use of advice in this magazine. Experiment with Linux at your own risk! Distributed by Marketforce (UK) Ltd, Blue Fin Building, 110 Southwark Street, London, SE1 0SU
Tel: +44 (0) 20 3148 3300

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

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 January 2015 when all content (including images) is re-licensed CC-BY-SA. ©Linux Voice Ltd 2014
ISSN 2054-3778

Subscribe: [shop.linuxvoice.com](http://shop.linuxvoice.com/subscriptions@linuxvoice.com)
subscriptions@linuxvoice.com

SYSADMIN

System administration technologies brought to you from the coalface of Linux.



Jonathan Roberts dropped out of an MA in Theology to work with Linux. A Fedora advocate and systems administrator, we hear his calming tones whenever we're stuck with something hard.

Among developers, 'test driven development' has become trendy once more. It's certainly not a new idea, as similar practices are described in *The Mythical Man Month*, which was originally published in 1975, but it is as popular now as it has ever been.

The idea is simple. Developers write automated tests to check that functions and features they've implemented work. Usually, these tests take the form of simple functions that call the code to be tested, and compare the output to an expected value, using an assert statement or similar. If the expected and actual value match, the test passes; if they're different, the test fails.

In TDD, this is taken one step further, and advocates argue that the developer should first write a failing test for the function they're about to implement, and then they can keep working until it passes.

What's this got to do with system administration? Well, I would argue that operations teams should take a similar approach when building out the infrastructure for a new product.

Tests can take the form of checks in monitoring software such as Check_MK or Nagios. Ensure that, after your provisioning servers, your monitoring server is the first thing you install. Then, for each subsequent server to be installed, first add it and all necessary checks (process checks for Apache, MySQL server status checks etc) to your monitoring software.

Then you can begin building it. At first, all the checks will be red. But as you boot it, and then run your configuration management recipes, you'll see check after check turn green. If any stay red by the time Puppet etc has finished, you know you need to tweak your recipes.

Btrfs

The filesystem that's better (or butter) in so many ways.

In our third and final look at new technologies making their way to Linux, we're going to explore Btrfs (which in my head I'm pronouncing butter-eff-ess). Btrfs is a new copy-on-write filesystem for Linux, which aims to deliver advanced features such as volume management, snapshots, checksums and send/receive of subvolumes.

If you're not already a filesystem expert, many of those terms might sound alien to you, but continue reading and we'll do our best to explain what these features do, why you want them, and when you'll get them.

Stability

Let's start with that final question, as any one who's paid even a little attention to news about Btrfs has heard horror stories about it destroying data and may well think it's a long way from production.

As it stands now, OpenSUSE plans to be the first major distribution to use it by default in its November 13.2 release, indicating that they believe it's stable enough for daily use. Facebook, too, which has recently hired many Btrfs developers, has announced plans to begin using Btrfs in its production web tier, where it can test

performance and stability on real, albeit easily recoverable, workloads.

This anecdotal support from distributions and large production environments, along with the official wiki claiming that the on-disk format is now stable, suggests that if you want to start testing, now is the time to do so. It might not be making its way in to the next round of enterprise distribution releases as default, but it will be there for those users who really need it.

So, if you want to try some of the features we'll describe in the rest of this article, make sure you have automatic and tested backups running. Do that, and even if the 'experimental' status of Btrfs does lead to data loss, you won't be left cursing.

Getting started

With that word on stability out of the way, let's get to work and create a new Btrfs filesystem. Once we have the filesystem in place, we'll start working through some of its core features, showing what they do, why they're great and how to use them.

For our simple experiments, we're going to use some plain files mounted as loop devices. So, to start, first create an empty 3GB file, use **losetup** to create a new loop

```

Terminal - jon@localhost:~
File Edit View Terminal Tabs Help
jon@localhost:~/Projects/puppet
jon@localhost ~]$ sudo losetup /dev/loop0 /home/jon/btrfs1
[sudo] password for jon:
jon@localhost ~]$ mkfs.btrfs /dev/loop0
probe of /dev/loop0 failed, cannot detect existing filesystem.
Error: Use the -f option to force overwrite.
jon@localhost ~]$ sudo mkfs.btrfs -f /dev/loop0

WARNING! - Btrfs v3.12 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

Performing full device TRIM (2.93GiB) ...
Turning ON incompat feature 'extref': increased hardlink limit per file to 65536
fs created label (null) on /dev/loop0
      nodesize 16384 leafsize 16384 sectorsize 4096 size 2.93GiB
Btrfs v3.12
jon@localhost ~]$ █

```

Creating a btrfs filesystem is just like any other – easy. In this tutorial, we've used loopback mounts to experiment, but this would all work just as well on a real disk.

device, and then create a new Btrfs filesystem:

```
dd if=/dev/zero of=/home/jon/btrfs1 bs=1024
count=3072000
```

```
losetup /dev/loop0 /home/jon/btrfs1
```

```
mkfs.btrfs /dev/loop0
```

That's all there is to it. You can then mount `/dev/loop0` as you would any other filesystem, examine it with tools like `df` etc.

As with any filesystem, there are a host of options you can specify at mount time to change the way that it works. With Btrfs, one useful option is `compress`, which enables you to turn on compression using either `zlib` or `lzo`:

```
mount -o compress=lzo /dev/loop0 /mnt/btrfs
```

While compression brings the obvious advantage of letting you store more data on disk, in some circumstances it can also bring a performance benefit too. On most systems without solid state storage, there are often CPU cycles to spare, while disk I/O can be a real bottleneck. By asking the disks to pull back less data, but asking the CPU to do some more work uncompressing that data, you can improve your performance.

Subvolumes

Now that you have a Btrfs filesystem available, let's look at the second (after transparent compression) feature of interest: subvolumes. A Btrfs filesystem can be divided into multiple roots that can each be treated as a filesystem in its own right (unlike logical volumes, these independent roots are not separate block devices):

```
btrfs subvolume create /mnt/btrfs/images
```

If you inspect the mounted filesystem at this point, you'll see what appears to be a new directory. You can `cd` in to it, you can create files within it etc. What happens, however, if you try to create a hard link between a file in this subvolume and the parent `btrfs` file volume?

```
ln /mnt/btrfs/images/screen1.png /mnt/btrfs/
screen1.png
```

That operation fails, just as if you'd tried to create a hard link between two different mount points.

OK, so what can you do with subvolumes? Well, when creating a subvolume, you can make it a snapshot of another Btrfs volume:

```
btrfs subvolume snapshot /mnt/btrfs/images /mnt/
btrfs/ss-images
```

Because Btrfs is a copy-on-write filesystem, this snapshot could be an exact replica of a 300GB filesystem and it would still have been created instantly. Btrfs only needs to copy data when information in the snapshot or the original volume actually

Some of the tools you've used in the past, such as `df`, won't take into account metadata and other features of Btrfs, so it has its own tools, such as `btrfs filesystem df /path` (the substitute for `df`).

changes, making them fast to create and remove as well as extremely space efficient.

What really makes subvolumes useful, however, is that you can mount individual subvolumes without mounting their parent. First, list all of the subvolumes in your Btrfs volumes to find out their 'subvolume IDs':

```
btrfs subvolume list /mnt/btrfs
```

Then, assuming the `ss-images` snapshot created above has volume id 258, `umount` the Btrfs filesystem before remounting with the following options:

```
mount -o subvolumeid=258 /dev/loop0 /mnt/btrfs
```

When you list the contents of `/mnt/btrfs`, you'll only see the contents of that subvolume. This feature is particularly important because it means, for example, you can snapshot your root volume before

striped across a pair of drives, which is in turn mirrored to another pair of drives.

Aims to give the benefits of RAID 0 and 1.

- **RAID 5 and 6** Stripe data, as in RAID 0, but sacrifice some space for 'parity' information. This parity information allows the array to lose one disk in RAID 5 or two disks in RAID 6.

To set up a multi-device Btrfs filesystem like this, first create a second loop device:

```
dd if=/dev/zero of=/home/jon/btrfs2 bs=1024
count=3072000
```

```
losetup /dev/loop1 /home/jon/btrfs2
```

Then use the `mkfs.btrfs` command again, but with the following options:

```
mkfs.btrfs -d raid0 /dev/loop0 /dev/loop1
```

You can check the man page for `mkfs.btrfs` to see other options for the `-d` switch.

“A Btrfs filesystem can be divided into roots that can each be treated as a filesystem in its own right.”

an upgrade, and if things go awry, remount the snapshot as your root and get back to a working state straight away.

Multiple volumes

As well as having these LVM-like features, Btrfs also shares features with traditional RAID, too. A Btrfs filesystem can be spread across multiple devices, and you can configure it to distribute the data across the devices according to one of several common RAID levels:

- **RAID 0** Striping, in which data is striped across disks, leading to improved read and write speeds. Btrfs also supports an extension of RAID 0 in which disks do not have to be the same size, known as 'single'.
- **RAID 1** Mirroring, in which data is mirrored across two or more disks of the same size. Can be faster for reads, but will slow down writes, as data has to be written twice.
- **RAID 10** Mirrored striped, in which data is

Self healing

We're close to the end of this month's overview, and there's so much we haven't touched on – file cloning, filesystem mirroring with send/receive, online rebalancing (aka changing RAID levels) and much more. Before finishing this month's section, there's one other aspect of Btrfs that I'd like to draw to your attention: Btrfs aims to be self healing.

Btrfs records checksums for each block that it writes. When it reads the data, it compares the data to its checksum, and if there's a difference, it automatically tries to re-read the data from one of your redundant copies or parity information – eg if you're using RAID1/10, 5 or 6 and Btrfs reads a bad block, you'd never know it happened unless you were to take a look in the logs.

That's all we have space for, but I hope you'll start thinking about all you could do with Btrfs when it does eventually hit your favourite distribution. 📦

CLOUDADMIN

Running make-believe boxes is virtually compulsory, suggests Nick Veitch.

Virtualisation: the options

There are many shades of hypervisor.

Virtualisation is a key technology that helped give birth to the modern cloud as we understand it. It helps run the services on the cloud and often helps build clouds too. But virtualisation is also important to developing tools to run clouds. As our foray into the 'dev' part of devops has already led us to look at how continuous integration is used (see LV002) we should also take a look at the virtualisation technologies commonly in use, their alternatives, and how they may differ from your experience on the desktop.

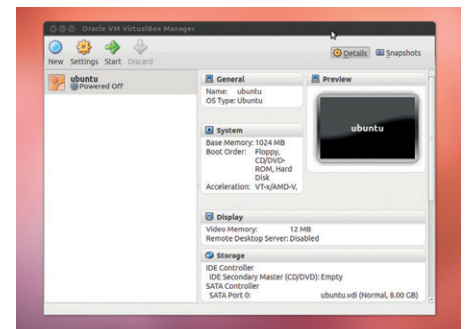
KVM

KVM works on top of Qemu, so for the purists, when we talk about KVM here we mean KVM/Qemu. KVM is a Linux-only virtualisation technology, parts of which are

included in the mainline kernel. The software relies on kernel modules to interface with the host CPU's virtualisation extensions – as such it will only run on CPUs that support (for example) Intel VT or AMD-V extensions (there is also an ARM port).

Popularity of KVM has not been driven by the desktop – it still lacks a lot of the snazzy configuration tools of VirtualBox – but it is very very popular for 'serious' use due to factors such as kernel integration and the unambiguous open source nature of the code (not to mention that it works very well).

Although it lacks somewhat in terms of graphical tools, VMs are controllable from the command line (and therefore, also easily by scripts and other software which uses the libvirt API – see our tutorial on page 94) to a greater degree than pretty much anyone



A popular choice for desktop users, Oracle's VirtualBox has some good things going for it in the developer space too.

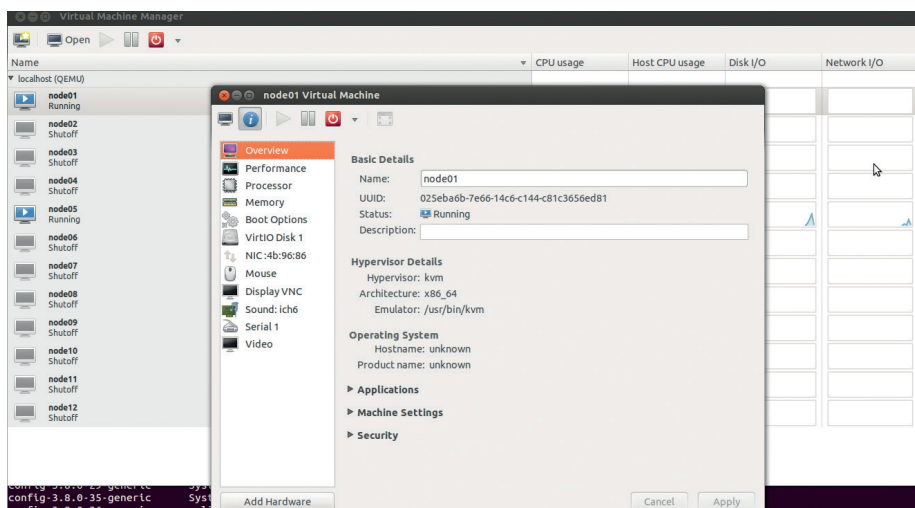
could ever want. As well as being a great tool for development, it is also widely used for spinning up VMs within clouds (eg OpenStack).

VirtualBox

VirtualBox came to prominence by virtue of it being a very featureful, well performing VM hypervisor that worked cross-platform and had an easy to understand management interface. The software has a colourful history – the original company that created it, Innotek, was acquired by Sun Microsystems, before many parts of the disintegrating Sun empire were snapped up by Oracle. As a largely open source project (there is a non-open source version, which makes use of proprietary device drivers for

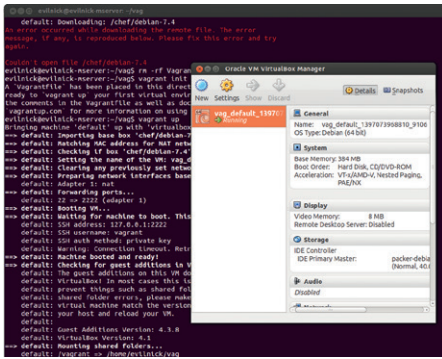
VMWare

No, we didn't forget about VMWare. Although it has been in the vanguard of virtualisation technologies for some time, VMWare is not open source. Although that doesn't exclude it from consideration in the world at large, it does tend to make it less relevant to the emerging cloud platforms, and certainly a little out of the scope of this FLOSS-loving publication.



Virt-manager is a useful graphical front-end for KVM, but you should really familiarise yourself with the virsh commandline tools, especially if you need tricky network setups.

“Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors.”



Vagrant is an effective tool for provisioning VMs and working collaboratively.

graphics, and the open source version seems to be stuck on an LGPLv2 licence) it sits a little ill at ease in the Oracle stable.

Nevertheless it is a mature and competent environment for running VMs. It relies a lot on paravirtualisation – special drivers that allow a more efficient throughput of data to and from the host OS. These do bring performance benefits, but rely somewhat on the co-operation of the guest OS, so if you are running custom kernels on strange distros you may not reap the full benefits.

It does have the huge advantage of also running on Mac OSX and Windows (and even Solaris), which can be beneficial in some collaborative environments.

Vagrant

Vagrant isn't a virtualisation engine, but it is most definitely worth talking about. The idea behind vagrant is that it becomes a sort of meta-manager for virtualised instances. Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors. Once you have installed Vagrant, you can fetch virtual machine images (which in Vagrant terms are known as 'boxes') and use them to bring up virtual machines.

You may ask yourself "What on earth is the difference between this and just creating a machine in VirtualBox?". The answer, at least at the system level, is "not much". Start up your box, and it behaves pretty much like any other VM you have initiated with

VirtualBox (or Qemu/KVM if you use that as the back-end).

The real difference is in provisioning. If you spend your life testing software, bringing up a clean VM is only part of the day-to-day grind. You then have to prepare that system for use. This ranges from the mundane installation of dependency packages to the more annoying repeatedly setting up options like host configuration or adding SSH keys so you can access the VM you created.

Yes, you can do this once in something like VirtualBox and create a snapshot image. Before you know it though you have half a dozen different snapshots that all differ in subtle ways, and aside from taking up loads of disk space, it can get pretty confusing. By using Vagrant to provision systems from a common set of boxes, you can reduce changes to your install to just changing some options in the Vagrant file that the software uses to bring up the VM. Of course, you can still create your own boxes, and there is a new service specifically for sharing those images in the cloud, so collaboration is much easier than trying to shift gigabytes of VM filestorage around.

Linux containers (LXC)

LXC is not a hypervisor for virtual machines. It is better than that; well, at a lot of things anyhow. LXC uses some very useful user-mode kernel features to containerise an implementation of a Linux OS – think chroot, but taken to extremes. Like a virtual machine, the container is able to carry out its business independently of the host OS, even to the point of running a different OS entirely. What you get is a self-contained running instance that is separate from the host OS, but which can dynamically share resources – there is no need to pre-allocate RAM and disk space for example, because the LXC container will simply consume what it needs just like any other process. LXC also plays nicely with libvirt, so you can use the same tools to

Further reading

- Why Vagrant? <https://docs.vagrantup.com/v2/why-vagrant/index.html>
- Stephane Graber's LXC primer <https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series>
- Using KVM with Ubuntu <https://help.ubuntu.com/community/KVM>
- VirtualBox homepage <https://www.virtualbox.org>

control containers as you may use with VMs (though to be fair, there are some peculiarities of LXC that aren't adequately addressed by libvirt, but you can also use the comprehensive LXC command line tools).

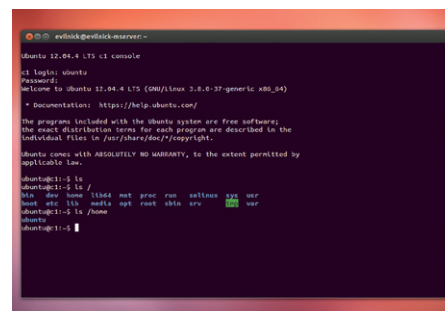
As there is no CPU or hardware virtualisation, there is a much lower overhead to running containers than VMs – there are no virtualisation layers to go through, so things like file access are much faster, and the scalable resources also mean that more efficient use can be made of hardware.

There are of course, disadvantages to using containers. For a start, you can only run Linux-based containers. It can even sometimes be tricky to run completely different distributions without additional tinkering. Added to that, the lack of virtualisation also means no virtual hardware – which can be a pain when it comes to configuring networking. For simple networks, LXC makes use of a bridged driver, which means the container can access an external network through the host's network setup, but complicated VLAN topologies become more troublesome. There can also be some nagging suspicions that what may work in a container could behave differently on real hardware. Gosh, not that anyone has real hardware!

LXC arguably has the most mature support on Debian and derivative distros, and is well worth experimenting with.

The virtual future

It sometimes seem mad that we run an OS on virtual machines through cloud software, which itself can be running on virtual machines, which themselves can be running on the very same OS. Don't think about it too much, it hurts. The point is that VMs (and containerisation) provide the essential flexibility of cloud implementations, and as the overhead associated with them gets smaller, they become more and more important enablers of future technologies. 📌



Linux containers are not a VM, and that is the whole point!

Docker
 Missing from this VM get-together is Docker. Like LXC, Docker is a containerisation solution, and we have left it out because we will be having a very detailed look at it next issue!

FOSSpicks

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



Mike Saunders has spent a decade mining the internet for free software treasures. Here's the result of his latest haul...

Create images from ASCII diagrams

Asciidia 0.3.2

Plain text is excellent, and much more versatile than a lot of people give it credit for. Take Markdown syntax, for one simple example:

```

Heading
=====
Some **bold** text
* This
* is a
* list
    
```

Although this is good old plain text, the extra formatting here is easy to understand, and the Markdown tools convert this text into decent HTML with just one command.

Asciidia does a similar job, but it focuses on creating diagrams from plain text files. In other words, you create images in a plain text editor using regular characters, run Asciidia, and get a proper vector (or bitmap) version.

As Asciidia is written in PHP, you'll need the **php5-cli** package installed to run it. Extract the tarball, **cd** into

the resulting directory, and there you'll see the program: **asciidia.php**. The best way to learn how Asciidia works is with an example file, and fortunately a few are supplied in the test directory. Have a peek at **test/diagram.txt**, for instance – it looks like classic ASCII art (see the screenshot), but Asciidia is clever enough to work out the shapes and signs contained therein.

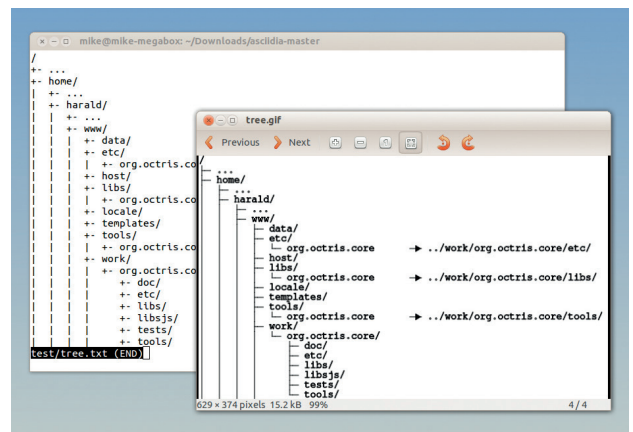
Words and pictures

To convert **test/diagram.txt** into a vector (SVG) image, enter this:

```
./asciidia.php -t diagram -i test/diagram.txt -o svg:diagram.svg
```

Asciidia doesn't provide much in the way of feedback, but its silence at the command line shows that the conversion has worked. Now

“Asciidia does a similar job to Markdown, but focuses on creating diagrams from plain text files.”



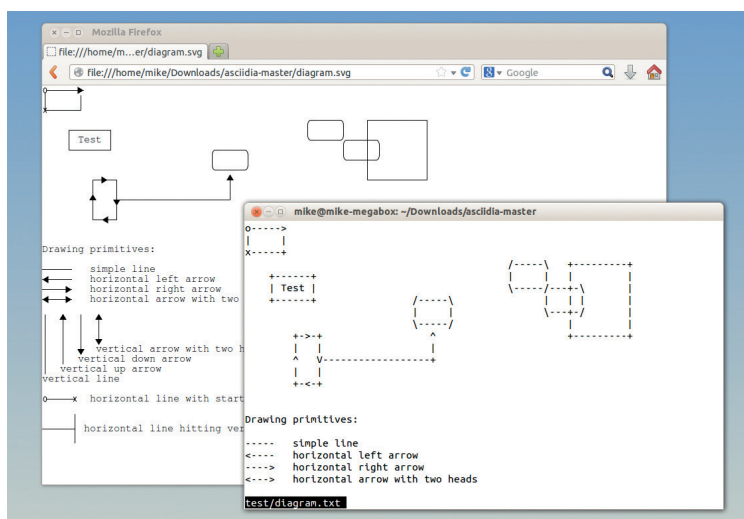
Here's **diagram.txt** being viewed in a plain text terminal, and Asciidia's SVG conversion shown in Firefox. Nicely done.

open **diagram.svg** in a vector graphics editor such as Inkscape – or you can even open it in Firefox if you don't have a vector editor to hand. And *voilà*: there's a fancier version of the ASCII art diagram, with everything in its right place.

Asciidia can also generate bitmap images providing that the **convert** tool from ImageMagick is installed. For some reason, on our Ubuntu 13.10 test box Asciidia complained that the **MAGICK_HOME** environment variable wasn't set, so we had to run the program like this:

```
export MAGICK_HOME=`which convert`
./asciidia.php -t diagram -i test/diagram.txt -o png:diagram.png
```

Note the **-o png** option here to produce PNG output. Asciidia can generate files in other formats too – see **README.md** for the details.



Asciidia can convert other types of plain text diagram, such as directory trees.

PROJECT WEBSITE
<https://github.com/aurora/asciidia>

Image manager

Phototonic 0.93

Zawinski's Law of software development, from Mozilla and XEmacs hacker Jamie Zawinski, states: "Every program expands until it can read mail." While this law is used in jest to mock programs like Emacs that have stretched way beyond their original purposes, it makes a good point: bloat and feature creep are everywhere in free software.

In contrast, we love it when a program has a very clear goal, as in the case of Phototonic. It manages your images – nothing else. It doesn't try to link to social media websites, or provide advanced editing facilities, or wash your car. All it does is help to view and organise your pictures, and consequently it's fast and reliable.

Phototonic is written in C++ with Qt 4 as the front-end, so you'll need the **libqt4-dev** and **qt4-qmake** packages installed (that's in Debian and Ubuntu – in other distros they may have different names). After extracting the tarball and **cd**ing into the new directory, enter:

```
qmake-qt4
```

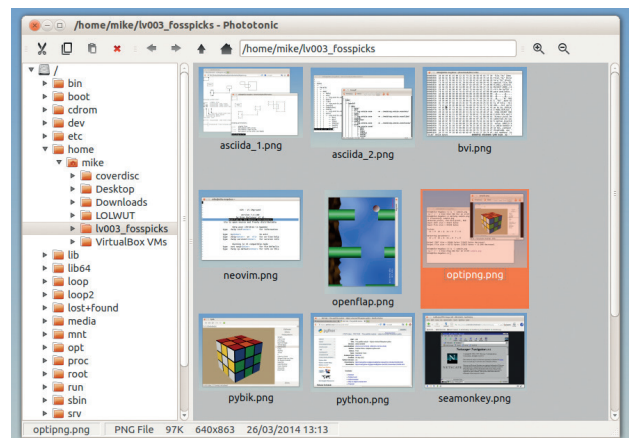
```
make
```

This builds the source code, and you can enter **sudo make install** to copy the binaries into your filesystem. Then just enter **phototonic** to start the program.

At first glance, Phototonic looks somewhat like a regular file manager, with a tree view of the filesystem down the left, and a pane containing thumbnails on the right. Navigate into a directory containing images, and you'll see them on the right-hand side. You can right-click on directories and images to rename them; this is how you're meant to organise your images in Phototonic. It doesn't try to do anything fancy with tags or metadata, but trusts that you can achieve what you want with a good old-fashioned directory tree.

Click on an image to highlight it (11 file formats are supported), and information about the image's file format, file size and dimensions is displayed in the bottom status bar. Double-click a picture to view it close-up, and press Escape to go back to the main screen.

Under the View menu you'll find options to change the sizes of the thumbnails along with the sort order, while the toolbar at the top of the window lets you navigate like in a web browser.



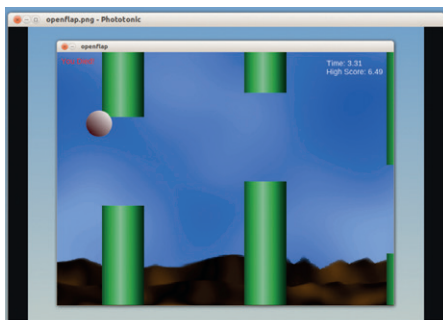
Use the plus and minus magnifying glass buttons in the top-right of the window to change the thumbnail sizes.

Phototonic includes basic image editing facilities (rotating, flipping and cropping) along with a slideshow view that shows a new image every five seconds. This default duration, along with other aspects of the program including default keybindings, is configurable in the Preferences panel. In all, it's exactly what a standalone image manager should be: simple, fast, stable and not overloaded with buggy features that should be implemented in separate apps.

"Photonic is exactly what a standalone image manager should be: simple, fast and stable."

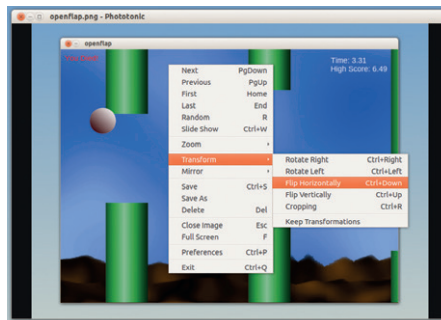
PROJECT WEBSITE
<http://oferkv.github.io/phototonic>

How it works: Editing an image



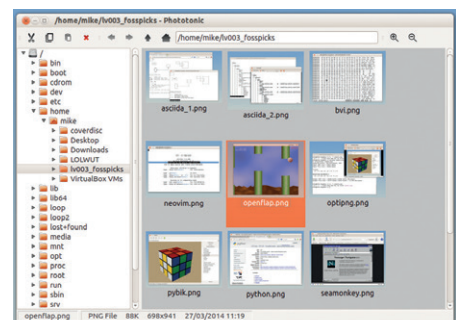
1 Select

Double-click on an image to display it in full. In this view, use the Page Up and Page Down keys to navigate through images.



2 Transform

Right-click on the image and go into the Transform menu. There you'll find options for rotating, flipping and cropping the image.



3 Save

Right-click again and choose Save, then press Esc to return to the thumbnail view. Hit F5 to refresh the thumbnails to reflect your changes.

Vi-like hex editor

Bvi 1.4.0rc

Nobody forgets their first encounter with the Vi(m) text editor. Compared with most "normal" editors, where you can simply type in text and press a key combination to save your work, Vi initially seems bizarre, with its arcane system of modes and commands like **:wq**. Yet after spending a while with Vi, many people find it to be incredibly efficient and powerful – hence its huge army of dedicated fans.

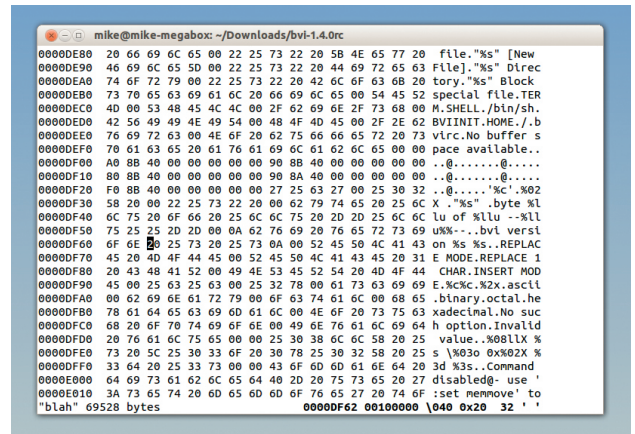
If you're one such Vi supporter, you might like Bvi: it's a similar editor geared towards working with binary files. To install it, you'll need GCC and the ncurses development files (eg the **libncurses5-dev** package in Debian/Ubuntu). After you've built the source code you can open a file with **bvi filename**.

Now, Bvi splits the screen into four sections: the left-hand panel

shows the addresses (ie locations) of bytes in the file, in hexadecimal (base 16) format. The middle panel shows the contents of those bytes in hexadecimal format, while the right-hand panel shows the ASCII representations of those bytes. Finally, a status line at the bottom shows the filename along with the currently highlighted byte in binary, octal, hex and decimal formats.

Vi-like commands

Move around using the cursor keys (or in more traditional Vi style, H/J/K/L), and hit Tab to switch between the hex and ASCII panels. To replace a byte, hit R. By default you can't insert or delete bytes - use the **:set memmove** command to enable these operations, and then **i** and **d** to do them. Saving a file and quitting is just like in Vi as well, with **:w** and **:q** respectively.



It's not pretty, but it's extremely lightweight (68k for the executable) and very useful when you need to poke around inside binary files.

Bvi has many more commands taken from Vi, along with a few useful extras such as the ability to edit a specified range of bytes in a file, instead of the whole file. Its minimal requirements mean that, like Vi, it will run almost everywhere, so it's on our list of "things to install by default in a new distro" now.

PROJECT WEBSITE
<http://bvi.sourceforge.net>

PNG file compressor

OptiPNG 0.7.5

Many of the new Free Software programs that get released are easy to ignore, performing piffling little jobs that perhaps only the developer finds useful. At first glance, one such is OptiPNG. Wow, so it reduces PNG file sizes by 10–15%; who cares? In this day and age of terabyte hard drives and blazingly fast internet connections, do such trivial reductions really matter?

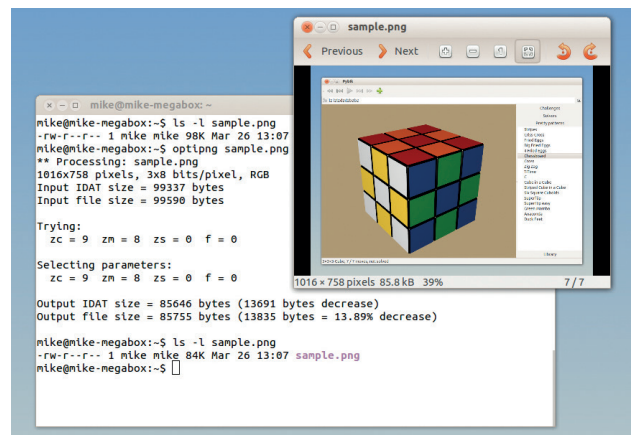
Well, they do in some circumstances – and they matter a lot. If you're hosting a very popular website serving up thousands of PNG images every minute, even the smallest reductions can add up over time, making your website faster and ultimately reducing your bandwidth costs.

OptiPNG performs lossless compression on PNG images; that

is, it tries to reduce their file sizes using various compression algorithms and without removing any pixel data. The end result looks exactly the same as the original image. Using it is very simple:

optipng file.png

OptiPNG spits out some information as it works, overwrites the original file with the smaller one, and shows you the reduction percentage. We did a bunch of tests using screenshots from this very FOSSpicks section, as generated by Gimp. Occasionally we saw impressive reductions of 25–30%, but by and large the shrinkage was in the 10–15% range.



Here's the FOSSpicks screenshot from the Pybik review, now 14% smaller thanks to OptiPNG.

While the default options do a decent job, OptiPNG has some extra settings for choosing the optimisation level and PNG delta filters. The program's website also has an excellent explanation of how PNG optimisation works under the hood: <http://tinyurl.com/a9wprt>.

PROJECT WEBSITE
<http://optipng.sourceforge.net>

“Even the smallest reductions can make your website faster.”

Internet suite

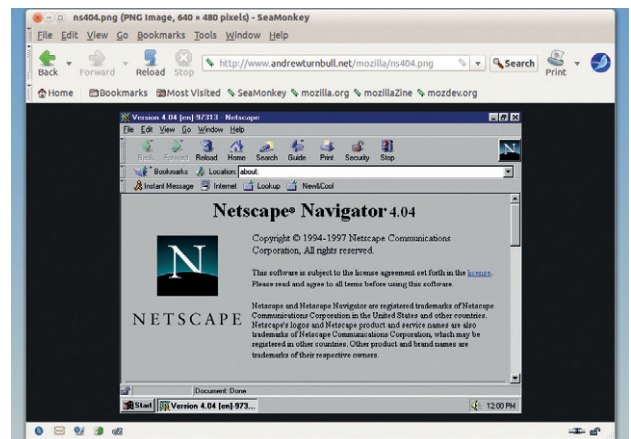
SeaMonkey 2.25

If you were around on the web in the late 90s, you'll certainly remember Netscape, a suite of programs including a browser (Navigator), email client and web page editor. After Netscape's demise, much of its source code was refactored by the Mozilla project, and today we see the results most famously in the form of Firefox. But while Firefox is a standalone browser, the internet suite project has also continued under the name SeaMonkey. In recent years it looked like SeaMonkey development was stagnating, but as more users become dissatisfied with Firefox the suite is getting more attention.

And it's really easy to try: grab the 33MB **.tar.bz2** file from the SeaMonkey website, extract it, jump into the resulting directory and run the **seamonkey** executable inside.

That's it – you don't need to install it system-wide if you don't want to. (Note that the program stores its data in **.mozilla/seamonkey/** in your home directory.) Prepare for a blast of nostalgia when you first start the app, because the interface has hardly changed since Netscape 4. A few things have been removed or clumped together, but otherwise it's quite similar.

Because SeaMonkey shares the same underlying HTML and JavaScript engines as Firefox, it works largely the same for web browsing, albeit with a more traditional interface. But we like the fact that it has been consistent over the years, not whimsically



Here's SeaMonkey from 2014, showing a screenshot Netscape 4.04 from 1997. Not a lot has changed, has it?

integrating every questionable change from self-styled “user experience designers” (ugh).

New features in version 2.25 include VP9 video decoding, support for Opus audio in WebM, and the Gamepad API (so that web games and apps can access USB or Bluetooth joypads).

“We like the fact that SeaMonkey has been consistent over the years.”

PROJECT WEBSITE
www.seamonkey-project.org

Editor reborn “for the 21st century”

Neovim 2014-Mar-23

There's something of a Vi(m) theme to this month's FOSSpicks, what with Bv's appearance on the facing page. Now we have Neovim, a fork of the regular Vim editor. But given that Vim is under active development, has masses of fans and a well respected lead developer, what kind of madman would want to fork it?

Well, that madman is Thiago de Arruda Padilha from Brazil, and he has some compelling arguments: Vim's codebase is old, complicated and full of cruft that could be removed, he says. It's difficult for new contributors to get involved. So it's time for Vim to undergo a major source code overhaul.

This isn't an easy task, and Padilha has set up a crowdfunding project to get started. At the time of writing, he had raised over \$32,000

to fund his work on Neovim – an impressive sum, given that his original goal was \$10,000.

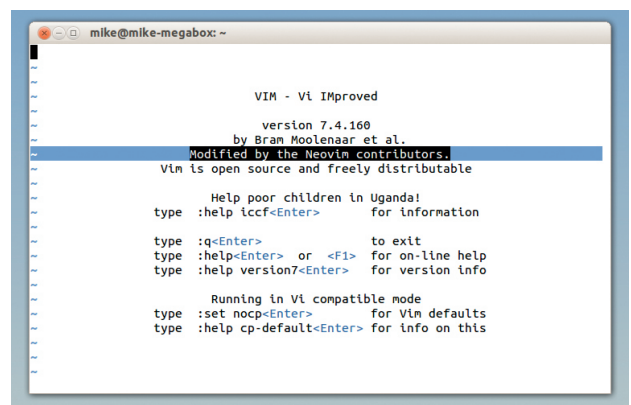
Plus ça change...

Although there's still a huge amount of work to be done on the editor, you can try it out right now to get a feel for it. To get the dependencies on Debian/Ubuntu:

```
sudo apt-get install libtool autoconf
automake cmake libncurses5-dev g++
```

Then grab the master **.zip** file from GitHub (<https://github.com/neovim/neovim>), extract it and run **make cmake && sudo make install** in the resulting directory. After compilation, you'll be able to run the editor with **nvim**.

Right now it looks, feels and smells like regular Vim – and that's the intention. Neovim won't look too different on the surface, as all of



So far there's not much to distinguish Neovim from the original version, apart from an extra line on the startup screen.

the important work will take place under the hood. Padilha wants Neovim to have a simpler build system, more platform-independent code, and a more versatile plugin system. It should also be easier to embed the editor into other programs, and create GUI front-ends for it on multiple platforms.

PROJECT WEBSITE
www.neovim.org

Programming language

Python 3.4

Maintaining a programming language is a tricky business. Python 3 was a bold step, breaking compatibility with Python 2 programs, but it was regarded as a necessary move to clean away a lot of cruft that had built up. There was plenty of controversy at the time, but Python 3 is maturing well, and now we have the 3.4 release. A bunch of new modules have been included, such as **pathlib**, which provides an object-oriented API for filesystem paths. For instance:

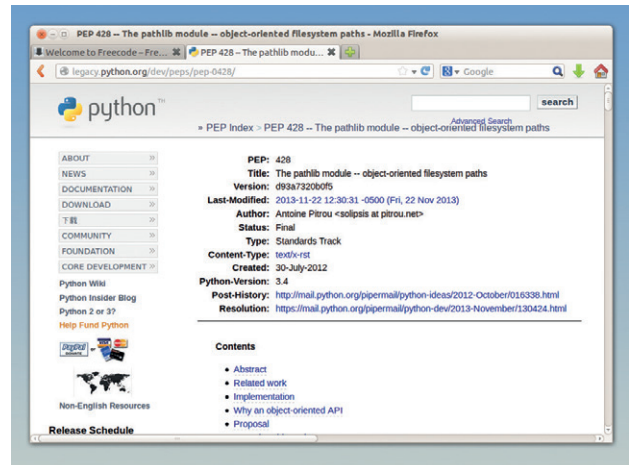
```
p = Path('/home/mike/foo/bar.py')
```

With this, **p.name** contains **bar.py**, **p.suffix** contains **.py**, and **p.parts** contains a tuple with every element in the path. You can join, split and compare paths, and query them (eg to find out if a path is relative). If you're doing cross-platform coding, there are ways to

handle Windows paths as well, including the drive letter and colon combinations at the beginning.

Another new module is **asyncio**, which provides asynchronous I/O via a pluggable event loop and coroutines. Then there's **enum** (provides support for enumerated data types), **tracemalloc** (a debugging tool to trace memory allocation) and **ensurepip** (a cross-platform way to install the PIP package manager into an existing Python setup).

Many security improvements have been implemented as well: for instance, there's now TLS 1.1 and 1.2 support in the SSL module. These are just the main new



Many Python features begin life as PEPs, or "Python Enhancement Proposals", which are discussed and reviewed by the community.

features in 3.4, and lots of work has been done on other modules to squish bugs and add general improvements. The changes have been very well documented too – something that's often lacking with new language release, so kudos to the Python team.

"Python 3 is maturing well, and now we have the 3.4 release."

PROJECT WEBSITE
www.python.org

Information organiser

TreeLine 1.9.4

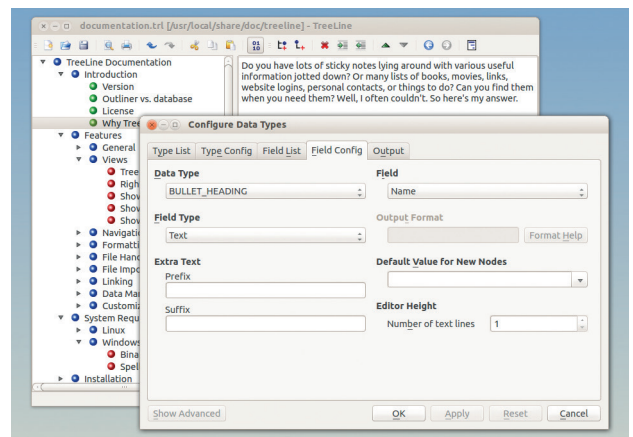
At the start of this issue's FOSSpicks we heaped spoonfuls of praise on plain text, but sometimes it can get out of hand. Take **notes.txt** for instance, the text file used for planning this section of the magazine: it started off well, with clearly defined sections and tidy presentation. But after a while it ended up as a morass of ideas, links, compilation instructions and other random bits and bobs. What we really need is a souped-up note taking and list compilation app – and thank \$DEITY, TreeLine provides it.

Essentially, TreeLine could be described as an outliner, note-taker or PIM program. It's hugely versatile and isn't designed to work with specific types of information; it will handle anything you can put in a tree-like structure.

TreeLine 1.9.4 is a development release on the road to 2.0, and is a complete rewrite of the earlier 1.4 series. It's written in Python with Qt providing the interface, so you'll need PyQt installed to run it. When you start the app, you're presented with a two-pane view: the left-hand side contains the tree of items, and the right-hand side shows the data for individual items.

It's in the trees! It's coming!

The best way to understand TreeLine is to open the documentation, which, brilliantly, is made in TreeLine itself. Click Help > Full Documentation in the menu, and a new TreeLine window will open. Click the arrows on the left-hand side to open up nodes in the tree, and green or red items to read them on the right.



TreeLine stores its data in (optionally encrypted) XML format, and can export to HTML and plain text.

TreeLine supports custom data types with a range of fields (eg text, numbers, dates) so you can easily customise it for nigh-on any type of information. It's hugely configurable and didn't exhibit any major bugs in our testing, so from now on it's our go-to app for FOSSpicks planning.

PROJECT WEBSITE
<http://treeline.bellz.org>

FOSSPICKS Brain Relaxers

Flappy Bird-like game

Openflap 1.0

Frontier: Elite II provided a giant universe to explore, together with open-ended gameplay where you could trade, mine, fight or work for the military. It provided months of fantastic entertainment – and all on the Amiga. That’s an incredibly rich game running on a 7MHz CPU backed with 1MB of RAM.

Today, people carry mobile phones that are several thousand times more powerful than the Amiga 600, and yet the most popular mobile games are so utterly trivial it makes us rage. Take Flappy Bird, for instance: it’s brain-shutdowningly tedious, and yet it was making its author \$50,000 a day at its peak.

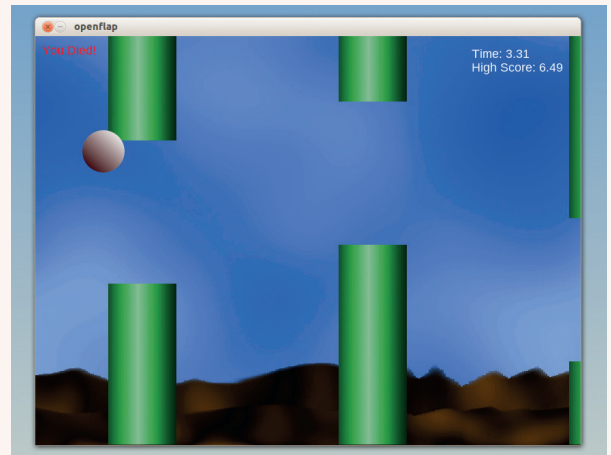
Anyway, in true FOSS style, someone has come up with

a free clone: Openflap. The gameplay is equally as minimal, and compiling it actually takes more brainpower (see the README file for instructions). The only dependencies you need are SDL and its various bolt-on libraries – so once you have it installed, just enter **openflap** to play.

Tapping tedium

If you’ve never seen Flappy Bird (or its million clones) before, here’s how you play: tap Space. A ball is falling from the top of the screen, and tapping space bounces it upwards. But! The ball is also moving to the right, and you have to tap space to help it through gaps in pipes. And that’s all there is to it.

Your high score is shown on the right, as an incentive to keep



Bounce the ball, miss the gaps – that’s it. But the code is useful for learning how to do graphics, sound and input in SDL.

playing, but what really prompted us to include Openflap is the source code. It’s clear and easy to read – a good resource for snippets and ideas if you want to make your own C++/SDL game.

PROJECT WEBSITE
<https://github.com/jazztickets/openflap>

Rubik’s Cube game

Pybik 1.1.1

Some people find Rubik’s Cubes maddeningly frustrating; others find them a good form of grey-matter stimulation. If you enjoy the odd session of “cubing” (as the cool kids call it) but you don’t want to have a cube by your desk in case your boss thinks you’re wasting time, then you can get by with a computerised version.

Pybik is written in Python, with its interface provided by Qt. So you’ll need **PyQt4** to install it – see the **INSTALL** file in the tarball for full instructions.

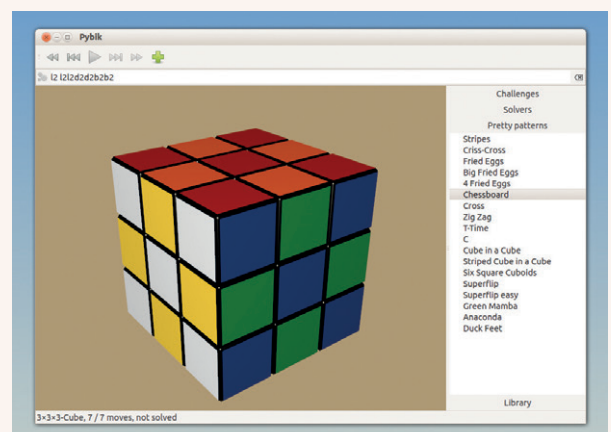
Once you’ve built it, you can run it in place with **./pybik**. The presentation is gorgeous: the cube is rendered in shiny 3D, with lovely light effects and impressive

detail to show the plastic parts onto which the coloured labels are stuck. OK, you might argue that visual frills are unimportant in a game like this, but if you’re going to be staring at the cube for a while as you solve it, why not make it look good?


Pretty colours³

Click and drag on the cream area around the cube to rotate it, and hover the mouse cursor over a piece to see how it will rotate when you click it (the mouse pointer changes direction).

A number of challenges are available, eg prompting you to solve a cube in under 10 moves, while the “Pretty patterns” list generates funky-looking cube layouts for you to solve. There’s also a library of



The lighting effects are a nice touch, although at some angles the glare is a bit over the top.

moves along with some solving algorithms. It’s a remarkably good substitute for a real cube, although you can’t throw it against the wall when you get annoyed with it. Oh well. 

PROJECT WEBSITE
<https://launchpad.net/pybik>



YOUR AD
HERE



Email andrew@linuxvoice.com to advertise here

TUTORIALS

Dip your toe into a pool full of Linux knowledge with eight tutorials lovingly crafted to expand your Linux consciousness.



Ben Everard
has his first batch of alcoholic ginger ale bottled and ready to drink.

Free software is an ideal close to many of our hearts, but it's not an isolated cause – it's under the wider umbrella of digital rights. After all, what advantage is having control of the software on your computer if the authorities block the content you want to see? This is exactly the issue facing internet users in Turkey and China. What good is a free office suite if the documents you need to access are in an obfuscated proprietary format?

Free software is booming, but other areas of digital freedoms aren't doing as well. There have been some successes in open document standards, and the Snowden revelations have shocked a few people, but they still haven't had as much attention as FOSS. Now is the time for the weight of the free software movement as a whole to come down on the forces that try to restrict our freedoms, whether that's in closing off our ability to modify software, or closing off other basic digital rights.

Linux succeeded despite some of the largest companies in the world trying to stop it, so there's no reason to think that we can't get similar levels of success with other digital freedoms. However, we have to fight for them. If we just focus on free software and not on other digital rights, we may end up winning one battle but losing the war.

ben@linuxvoice.com

In this issue...



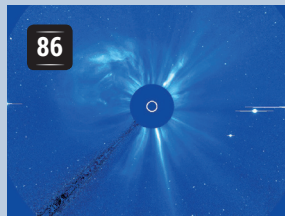
LXDE

Bored with the default LXDE desktop? Beautify this lightweight environment by following **Ben Everard's** easy steps.



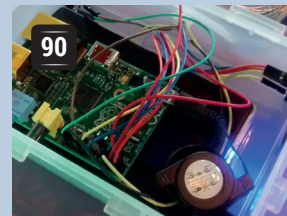
Wearables

The Jean Paul Gaultier of Linux, **Ben Everard**, sews electronics into clothes to make himself super cool.



Hunt Comets

Using Python and public data, **Andrew Conway** chases down comets.



Pi Beacon

Les Pounder, the BA Baracus of Linux, escapes trouble with a Ras Pi.



Libvirt

Valentine Sinitsyn masters his virtual machines using Python scripts.

PROGRAMMING

Ruby

98 Everything is an object! It's a popular saying among object-orientated programmers, but what does it mean, and how does it affect the way you program? We delve in and find the answer, or at least the Ruby version of the answer, and then set you readers a challenge.

Genetic algorithms

104 Programming involves breaking a problem down into its constituent parts, then assembling a step-by-step method of finding the answer. Wouldn't it be much easier if you could just say what answer you wanted and let the computer work it out? With genetic algorithms, you can.

Google script

106 The cloud offers loads of options for running software. EC2 or VPSes allow you to run virtual machines almost anywhere in the world, but sometimes you want something a bit lighter. Google Apps Scripting is a way of running simple programs on Google's servers.

CUSTOMISE THE LXDE DESKTOP

Get a fantastic desktop environment without overloading your system's hardware.



The Lightweight X11 Desktop Environment – or LXDE as it's more commonly known – is popular for its ease of use and low use of system resources. It's the desktop of choice for the Raspberry Pi, and is an excellent option for replacing Windows XP on older machines. However, in its default form it is a little ugly. Everything works as you expect it to, but it doesn't show off the Linux desktop experience as well as it could. Fortunately, it's quite easy to whip the default configuration into something that looks good and is a little more user friendly.

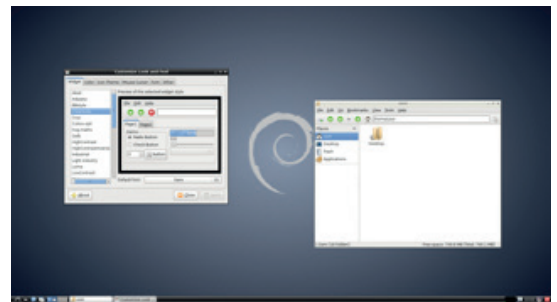
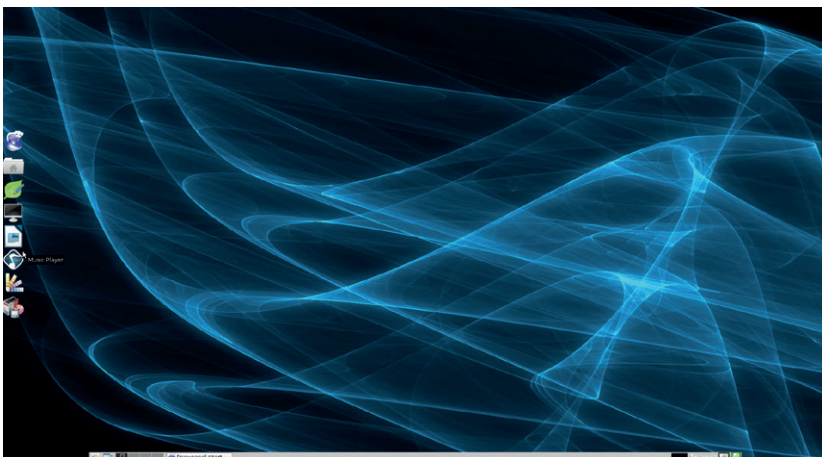
A desktop environment has a large stack of things that are really just images. These are the icons, the bits that make up the widgets (such as buttons), and the desktop background. These can all be easily swapped around provided you have new images to go in their place.

Get new wallpaper

There's no one single place for LXDE themes, but there is for Gnome, and they're mostly compatible. Head to www.gnome-look.org to see a fantastic range of user-submitted work. There are some great-looking things on there, and there are some truly terrible ones too, so take a little time to find ones you like. By default, the website shows the most recently added items, and the quality is variable. You usually need to switch to Highest Rated or Most Downloaded to find the good choices.

To switch desktop wallpapers, just save the image file that you want to use, then right-click on the desktop and choose Desktop Preferences in the menu. This will then give you the option to browse to the image file you want.

This is our LXDE desktop after tweaking. You may notice we've also changed the menu icon. This is done by right-clicking on the old icon and selecting Menu Settings.



The standard LXDE desktop: it's functional and easy to use, but with a little effort we can do much better.

Icons and themes take a little more to change, but are still quite straightforward, since there's a tool called LXAppearance to help. First you need to download the theme. We started with the Elementary icons at www.gnome-look.org/content/show.php/elementary+icons?content=73439, though most icon themes should work.

Follow the download link to DeviantArt, then download the Zip file. In principal, it is possible to install the icon theme with LXAppearance, but in practice it's a little awkward since it only supports **tar.gz** and **tar.bz** files. We found it quite unstable when installing anything. All installing does, though, is place the files in the appropriate directories, so it's quite easy to do it without an automatic installer.

Install new icons

Icon themes should be placed in a folder called **.icons** in the user's home folder. The easiest way to do this is with the PCManFM file manager that comes with LXDE. Just open up your home folder and make sure hidden folders are displayed (you should tick the box in View > Show Hidden). If there isn't already a folder called **.icons**, you need to create it (right-click > Create New > Folder). Then just unzip the icon theme that you've downloaded (right-click it in the file manager, then select Extract To and in the folder path enter **/home/ben/.icons** – with your username instead of **ben**).

To activate the icons, you'll need to use LXAppearance. Depending on your setup, you might find this in the Applications menu under Preferences > Customise Look And Feel. If it's not there, you'll have to run it by typing **lxappearance** in the terminal. In the Icon Theme tab, you should now find the Elementary theme (or whichever Icon theme you installed).

The same basic method can also be used to add new widget themes. In [gnome-look.org](http://www.gnome-look.org), these are under the GTK 2.x menu in the left-hand column of the screen. We went for BSM Simple (www.gnome-look.org/content/show.php/BSM+Simple?content=121685) These have to be downloaded and extracted into the folder `.themes`, and then they'll appear in the Widget tab in LXAppearance.

The eagle-eyed of you may notice that after installing, it looks a little different to how the theme looks on the main website. We'll come back to that in a minute, but for now, we'll go on with adding a dock.

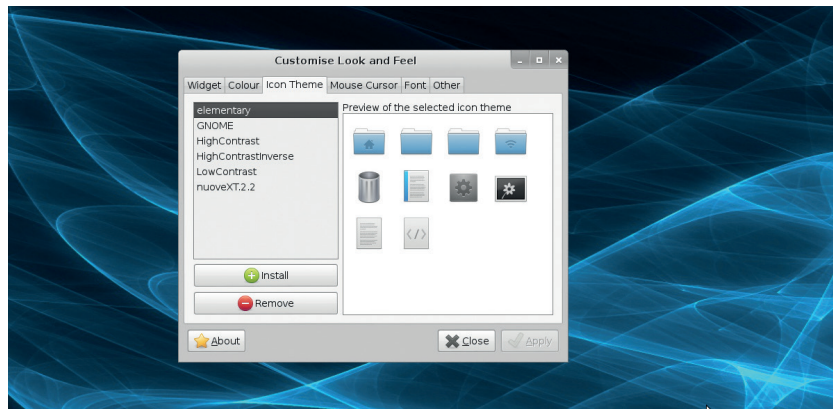
Building a dock

LXDE comes with a panel along the bottom that holds most of the basic desktop utilities, such as the applications menu, window list and system tray. It can get a little cluttered, so we like to have an application launcher on the side of the desktop to provide quick access to the programs we use most frequently.

This is really just another panel, but we'll use a few tricks to make it function better for our needs. First, right-click on the bottom panel and select Create New Panel. This will add the new panel and open the Panel Preferences window. The first thing to do is get it in position on the left side. We put ours in the middle of the left-hand edge of the screen, taking up 40% of the edge, 54 pixels wide with icons 50 pixels big.

In the Panel Applets tab, add an Application Launcher Bar, then double-click on the entry in the list to open Add Applications To The Launcher. Once you've selected your favourites, you can set the appearance. In Appearance, select Solid Colour (with Opacity), then click on the colour and scroll the opacity down to 0. The final thing to keep it out of the way is to select Minimise Panel When Not In Use in the Advanced tab.

This means it won't take up any screen space normally, but you can just move the mouse to the left edge of the screen when you want to open up an application, enabling you to have nice big application launcher buttons without spoiling the look of the bottom panel with loads of clutter.



We think that the nice-looking Metacity windows are well worth the extra few clock cycles they take to render.

On [gnome-look.org](http://www.gnome-look.org), you'll see that most themes have rounded corners on the windows, but when you install them, you get square corners. This isn't a huge deal, but you'll also find a few other things that don't quite look as well as they could. The reason for this is the window manager.

Under new management


By default, LXDE uses the Openbox window manager. This is lightweight, and serves most purposes quite well. Openbox looks its best with very minimal windows, and a very clean design. A lot of people like this, but there's also a place for slightly more substance to the windows. For this, a better look can be achieved with other window managers.

Our favourite is Metacity. This is the Gnome 2 window manager. Of course, there's a trade off to this. Metacity will use a little more screen space than Openbox, and a little more CPU and memory. The difference shouldn't be much though: we tested both, and Openbox used about 0.5–1 % of the CPU time, and 1% of the memory, while Metacity used 2–3% of the CPU and 2% of the memory. By comparison, in both cases, the underlying X Windows System used 10–15% of the CPU and 6% of the memory, so while Metacity does increase the window management overhead, in most cases it won't be significant.

To switch to Metacity, first make sure it's installed. On Debian-based systems, this is done by typing the following at the terminal:

sudo apt-get install metacity

You can then make the change. Go to the Applications Menu > Preferences > Desktop Session Settings, and in the Advanced tab, change Window Manager to Metacity. You'll need to log out and back in again (or reboot) for the changes to take effect.

As you've seen, there are loads of things you can do to improve the default look of LXDE. None of these things really change the way you use the system, but they can make it a little more pleasant. We've shown you how we like it, but with a bit of experimentation, you should find a setup that works well for you. 

Configuration files

Almost all of the configuration we've done has been either by installing work other people have done, or via point-and-click settings. This is a simple way of getting access to a huge range of settings, and you can create wonderful desktops doing just this. However, the ultrageeks among you may be itching to exert ultimate control over everything on your desktop. Fortunately, you can.

If you want to change the appearance of the windows, you'll need to dive into the theme. Creating a new theme from scratch is a daunting task, but it's pretty straightforward to modify an existing one. The Gnome wiki has details of what the various bits are (<https://wiki.gnome.org/Attic/GnomeArt/Tutorials/GtkThemes>), and you'll find everything in text files in the folder that you extracted into the `.themes` directory.

Ben Everard is a Pi enthusiast and the co-author of the best-selling *Learning Python With Raspberry Pi*.

BEN EVERARD

MAKE SMART CLOTHES WITH AN ARDUINO LILYPAD

Add a microcontroller to your cycling jacket and take one more step along the road to pervasive computing.

WHY DO THIS?

- Be the best-dressed cyclist in town
- Learn how to program clothing
- Get started with the neopixel and add colourful LEDs to your projects

Over the last decade, it has become much easier to make electronic gadgets. The Arduino revolution has made the micro-controllers easier to use, and at the same time, much more hardware and software has been created. You can now plug a few shields into an Arduino and get a mobile phone, or a GPS navigator with a touchscreen, or, well, almost anything you can imagine.

Most of the time, these gadgets use traditional circuit-making methods, such as PCBs, breadboards and strip boards. However, that doesn't have to be the case. With a little ingenuity, you can create circuits in all sorts of ways – such as by sewing conductive thread into clothes. As an example, we'll create a cycling jacket that has some LEDs to make it a bit more visible than most, but the techniques we use could easily be used to make all manner of items such as light jewellery, or digital art.

There's nothing to stop you stitching any circuit board into clothing. In fact, you could be forgiven for thinking that a small headerless microcontroller board such as an Arduino Pro is ideal. However, there are a few disadvantages to using general-purpose boards.

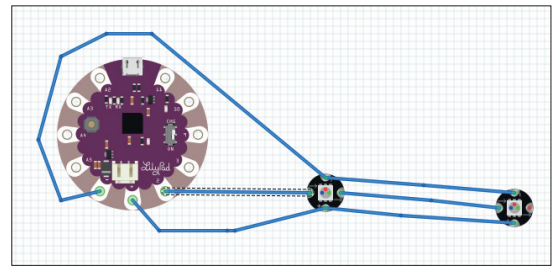


Fig 1: The first row of neopixels can be attached entirely with conductive thread.

The connections tend to be too close together (thread is less precise than soldered wire) and they have smaller contacts, which can be troublesome for use with e-textiles.

There are two lines of microcontroller boards that are designed to correct these problems, and both are perfect for wearable projects: the Lilypad and the Flora, from Adafruit Industries. They're both based on the Arduino, and are mostly compatible with each other in terms of code and hardware. The biggest difference between them from a Linux user's perspective is that the Lilypad boards work on Linux with the official Arduino IDE, while the Flora (and the smaller Adafruit Gemma board) don't. There is some guidance on the Adafruit website that may help you get the Flora working under Linux, but it's known to have some problems (particularly the Gemma variant). Because of this, we opted to base our project on the Lilypad.

Choose your controller

There are a few different types of Lilypad. Most don't come with USB integration, and need an external FTDI board in order to program them. The original Lilypad is the largest. There is also the smaller Lilypad Simple, and the Lilypad SimpleSnap, which can easily be removed to allow the clothing to be washed. The Lilytiny is the smallest, though it is a little harder to program. The easiest to get started with is the Lilypad USB, which has everything onboard, and this is the one we've used in our project.

The Lilypad USB is supported by the Arduino IDE from version 1.0.2 onwards, though we used version 1.5 in this project. If your distro comes with an earlier version, you'll need to download the latest from www.arduino.cc. Once you've got it, you just need to unzip the archive, then run the `arduino` script in the

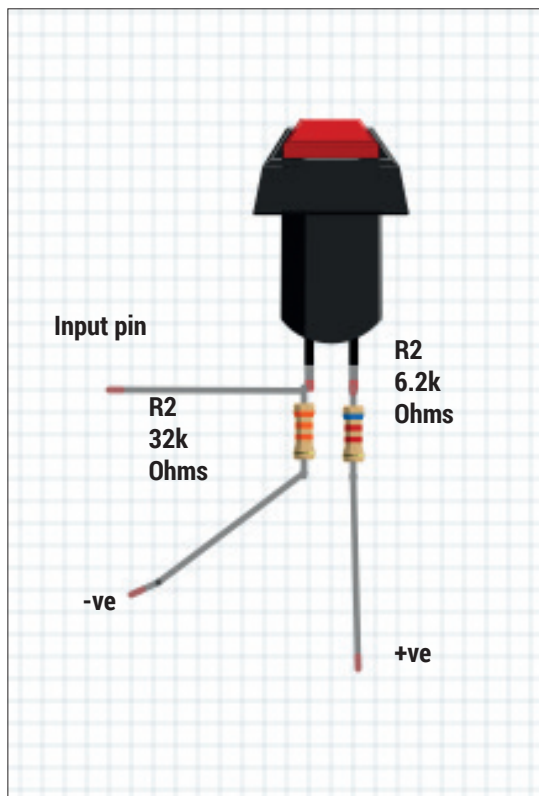


Fig 2: These two resistors make the switch work, and they're the most fiddly bit to fit into the jacket.

Power supplies

The easiest way to power the Lilypad USB is through the JST connector. This can take a lithium polymer (LiPo) battery with an output of 3.7V, and run everything off that. There's even a charging circuit built into the Lilypad USB, so you can recharge the battery by plugging the Lilypad into your computer. This means you can tuck the battery into some inaccessible place and not worry about it.

There are other batteries that can connect via JST; for example, you can get holders for three AA or AAA batteries, or for two CR2032s.

A third option is to use a USB power source. You can get battery packs designed to give mobile phones extra power, and these should work when plugged into the USB port. It's probably only worth doing this if you happen to have one of these lying around, as they're bigger and more expensive than the alternatives without having any real advantages.

new directory, and everything should work as long as you've got Java installed.

The microcontroller is the brains of the project, but it's useless without additional components for input and output. We're going to use a few extra pieces to give us the functionality we need.

Flora neopixels from Adafruit give us light. As you may have guessed from the name, they're designed to work with the Flora board, but you can equally use them with the Lilypads (or, for that matter, other Arduino-compatible boards). Neopixels are chainable RGB LEDs, which means that one pin on the controller board can drive many lights – an especially useful feature on sewable boards, as these tend to have fewer pins than most.

Neopixels take power separately from the data input. See figure 1 for details of how to wire them up. In order to use them you'll need the library from Adafruit. You can find details of how to install this on the official website (<http://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library>).

The best way to prototype wearable projects is with alligator clip leads – these are the breadboards of the wearables world. In order to make sure everything's working properly, you can connect up your neopixels as shown in figure 1, and run the strand test example sketch that comes with the neopixel library. You'll need to adjust the number of neopixels, and the pin that they're on in order to run it. It's best to use just two neopixels in a test, for reasons we'll explore later.

You'll see in a bit that we actually split the neopixels up into two strips of two. This is just to make the sewing a little easier.

Connecting the circuit

The purpose of this project is to create a cycling jacket with improved visibility. We used four neopixels sewn into the back of the jacket to flash red. In addition, we added switches to enable the outermost of the pixels to be turned into indicator lights. In order to do this, we need a way to tell the microcontroller that we want to turn. The easiest way to do this is switches. There are

all manner of switches available, but we needed some that are on-off (that is, you press them once to go on, and a second time to go off), and suitable for wearables. The best ones we found were from Adafruit at www.adafruit.com/products/1092.

You can't just put a switch between positive voltage and an input pin on the microcontroller and use it as an input. When it's on you'll get too much current flowing into the input pin, and you could damage the microcontroller. When it's off there's no input to the pin. You may think that no input is the same as an off input, but it's not. No input is a sort of floating state that can go either way, and while it'll usually go to off, it'll flash on, and create all sorts of problems. The solution to both of these problems is a resistor, though in slightly different ways.

Wire up the switch

Take a look at figure 2 for details of how to wire up a switch. When the switch is open R1 stops too much current damaging the input pin. R2 allows a little current to leak away, but since it's quite a large resistor, this isn't too much. When the switch is closed, R2 connects the pin to ground, and this is enough to make sure that the input always reads off.

The final piece of input and output hardware we'll use is a piezo buzzer. This will buzz to let the wearer know when there's an indicator on, so they don't forget to turn it off. You can get sewable buzzers, but we used an ordinary piezo element. It's not very loud, but it doesn't need to be, because it's just there to remind the cyclist that the indicator's on.

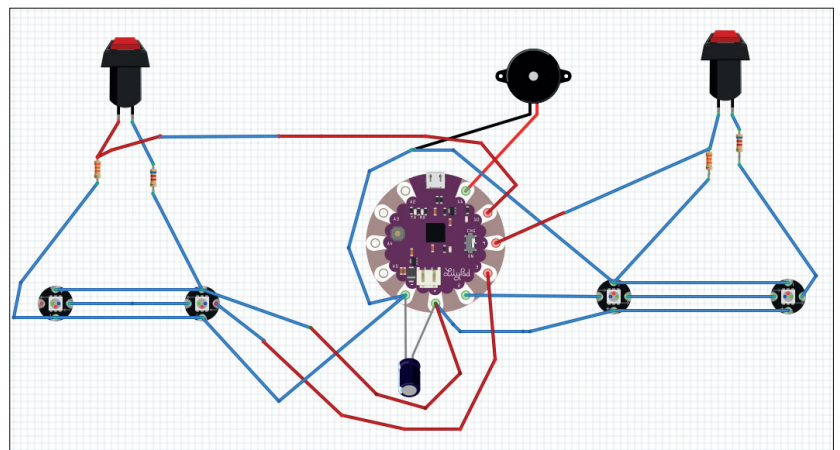
The most unusual thing about wearable electronics isn't the hardware, it's how they're connected together. You could use wires, and sew them onto the fabric, but the downside to this is that it'll make the fabric stiff, particularly if there are a lot of wires. Most wearable projects use some form of conductive sewable. There are sewable ribbon cables and conductive fabrics, but by far the most popular option is conductive thread.

Sewable thread comes in different grades, and most should work with this project. This is a really small wire of twisted stainless steel strands. You don't need any special equipment to use it, as it can be cut

LV PRO TIP

A multimeter will make your life a lot easier when testing the integrity of your circuit.

Fig 3: The blue wires are conductive thread sewn in; all others are wire.





The first neopixels sewn in. We got the alignment a little wrong so the conductive thread takes a longer path than it needs to, but it still works.

with scissors and sewn with ordinary needles. We used three-ply and got through about 30 feet (including wastage and mistakes).

Perhaps the biggest consideration when laying out a wearable circuit with conductive thread is that none of the connections can cross, because the wire isn't insulated. If you're using thick fabric you could try crossing on opposite sides of the cloth, but there's a pretty good chance that you'll run into problems. Good circuit design should minimise the number of times that two threads need to cross, and in simple circuits, it may not have to happen.

We solved the problem by using short lengths of insulated wire when paths had to cross. In principal, you could probably get away with lengths as

short as an inch just to act as a bridge if flexibility is critical, though we used lengths a few inches long to make it simpler.

In terms of circuitry, our design is simple. Perhaps the most important decision for layout is where to place the Lilypad itself, because this will affect how everything else connects together. Since we're going

“The circuit can be built up bit by bit – the first step is to get the lights working properly.”

Introducing Arduino

If you've not heard of the Arduino, then you're missing out on a revolution in microcontrollers. They're simple boards that allow a wealth of input and output options. The exact options depend on the board, but range from 20 IO pins on the Uno and Micro to over 50 on the Due and Mega.

They don't have full CPUs, but instead AVR microcontrollers. You can think of these a bit like really simple System On Chips (SoCs). They have a bit of flash storage for programs, and a bit of RAM to hold variables, and a simple processing core. It's not enough to run an operating system, so instead you program them directly with no OS underneath.

The real innovation of the Arduino system was in making them really easy to program. There's a huge library of code that you can use to quickly build quite advanced projects, and they can be programmed directly from USB with no special hardware.

Arduinos are programmed in a dialect of C++. All programs have at least two functions: `setup()` and `loop()`. `setup()` is called at the start, then `loop()` runs in, well, a loop. If you're at all familiar with C or C++, you should find it easy to pick up from looking at the examples that come with the IDE. If you're not, then there are loads of great books and online resources to help you get started.

to have components symmetrically laid out over both sides, we opted to put it in the middle. The four neopixels are in a line across our upper shoulders. This makes them more visible to drivers, and also shows the width of the cyclist. There's a very bright light on the front of our bike, so we didn't add any additional LEDs to the front, though you could easily do this. The buttons are on either side of the chest making them easy to press with either hand.

You could put the buzzer anywhere on the jacket, but we added it to the collar so it is close to the Lilypad and easy to hear.

Assemble the wearable circuit

The full circuit can be built up bit by bit. The first step is to get the lights working properly, and to do this you need to decide where the LEDs should be. This may sound simple, but it can be surprisingly confusing to work out what goes where when the jacket isn't being worn. It's easiest to put the jacket on, and get an assistant to mark the right places with a pen or pencil.

Because we've arranged the neopixels in two strips of two, the wiring gets a little convoluted right from the start. If you want, you could simplify this by having a single strip, and have the Lilypad on one side of the jacket, though this may cause complications with the buttons. See figure 3 for details of how we laid it out.

The sewing itself is straightforward. If you're feeling fancy, you can alternate the lengths of the stitches so that those on the outside are shorter than those on the inside, to make the conductive thread less visible. However, we wear our electronics like a badge of honour. Similarly, we've mounted all the circuitry on the outside; this could go inside, but it could chafe if you weren't careful with placement.

The tricky parts of sewing is making the connections at either end – the key is to loop through the hole several times, and make sure it's tight. We used a drop of glue on to stop the thread coming loose, but better stitchers may not need this. Be careful not to use too much glue, as it can get between the thread and the contact and be counter-productive. On the neopixel positive and negative points, you need to continue the rail after the first pixel. It's easiest to do the full rail in one thread, and continue after sewing in the first neopixel. This is because the holes are quite small, and it can be hard to sew in a second time. We did manage to sew in again when we needed to, so it's not too big a problem to do it in two threads.

Make sure that you trim the ends quite short, as it will cause problems if two threads touch each other. Beyond these minor points, it's no different from sewing anything else, so if you know a good sewer, you may wish to ask for a little help as they will be able to keep it neat.

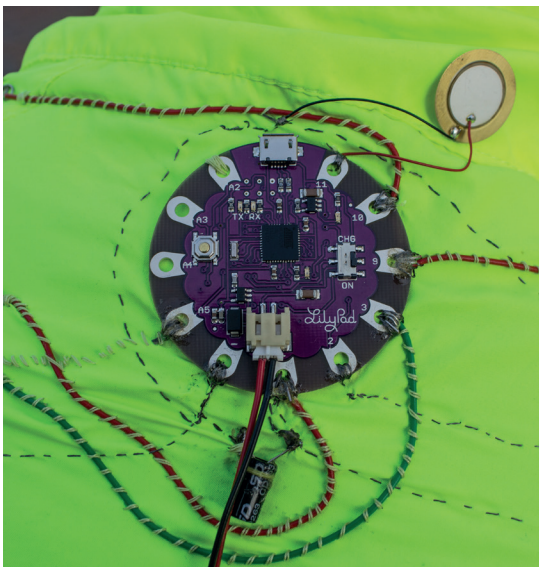
Already there are sections that need wire, and we have a few options: you could solder the wires onto the Lilypad before you start sewing, or you could always take thread off the Lilypad, then loop into the

wires later. We chose to sew the wires onto the board. This was easy to do and gave the wires more flexibility than if they'd been soldered on. First we stripped about half to three-quarters of an inch of the wire, then we looped this through the hole on the Lilypad, making sure that the end of the wire poked away from the person wearing the jacket. Then we took some thread and looped it through to make sure there was always a good contact between the wire and the Lilypad.

To keep the wire in place, we then sewed it in with some cotton (non-conductive) thread along its whole length. We bent one end of the wire into a circle (you could add a drop of solder to help it stay in shape), and stitched in the thread. These were the most troublesome contacts, so make sure you loop the thread around the wire a few times as well as sewing it in. If you find your circuit isn't working at any point, use a multimeter to make sure all the contacts are good.

Programming your jacket

A word of warning before we get started. There are three LEDs in a neopixel (for red, green and blue). Each of these can draw 20mA on full brightness. So, for full white light, that's 60mA per pixel or 240mA altogether. The regulator on the Lilypad can cope with a peak current of 500mA, but a continuous current of only 200mA, and this has to supply the microcontroller, buttons and buzzer. This means that if you put all the pixels on white, there's a good chance you'll burn out the controllers. There are two solutions to this. Either you can power the neopixels separately with another battery (or separate leads from the same battery that don't go into the Lilypad), or you could program the Arduino to not have too many of them on at once. We've gone for the latter approach to keep the design as simple as possible, and we've kept our code quite cautious. If you want to experiment with brighter lights, either power the neopixels separately, or be careful not to blow your regulator.



The wires coming off the Lilypad make it a little messy, but you can't feel this when you wear the jacket.



The complete setup with the battery hanging down. This lights the cyclist higher up than traditional bike lights and make the rider much more visible at night.

With that warning in place, let's get started programming the jacket. If you've not used an Arduino before, take a look at the boxout on the facing page.

The following code will simply test that everything's working properly, and cycle through a few colours.

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel strip1 = Adafruit_NeoPixel(2, 2, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel strip2 = Adafruit_NeoPixel(2, 3, NEO_GRB + NEO_KHZ800);

void setup() {
  strip1.begin();
  strip1.show();
  strip2.begin();
  strip2.show();
}

void loop() {
  strip1.setPixelColor(0,50,0,0);
  strip1.setPixelColor(1,50,0,0);
  strip2.setPixelColor(0,0,50,0);
  strip2.setPixelColor(1,0,50,0);
  strip1.show();
  strip2.show();
  delay(1000);
  strip1.setPixelColor(0,0,50,0);
  strip1.setPixelColor(1,0,50,0);
  strip2.setPixelColor(0,50,0,0);
  strip2.setPixelColor(1,50,0,0);
  strip1.show();
```

Washable and weather-proof

None of the parts we've used are officially weather-proof or washable. That means if you get them wet, and they break, you can't return them. That said, there's nothing that should get into much trouble if it gets a bit damp (the piezo buzzer may not fare too well, and the battery should be kept as dry as possible). If you do encounter a spot of rain, just turn it off, and hopefully, it will survive. Let it drip dry fully (including the inside of the switches) before turning it back on.

Waterproofing isn't easy, but it should be possible to make it at least stand up to some rain. The first stage would be waterproof

housing for the battery and buzzer. Waterproof switches are available, or you could put the ones we've used inside some flexible plastic cases.

With this done, you would still need to power it off during rain because the water could short out some of the connections.

The Lilypad and neopixels should stand up to a dunking (though there aren't any guarantees). Adafruit is working on making fully waterproof wearables (see a test here: www.youtube.com/watch?v=P42MzjuEPig) though at the time of writing, there isn't anything available for purchase.

```
strip2.show();
```

```
delay(1000);
```

You'll find it this code at www.linuxvoice.com/code/wearable.tar.gz as **jacket_test**.

To upload the code, first plug the Lilypad into your computer, then go to Tools > Boards and select Lilypad Arduino USB (It must have USB at the end). If that's not an option, it means you don't have the latest version of the Arduino software. You'll need to update this before continuing.

The first line of the code just includes the library (make sure you've installed this first – instructions above). You then need to set up the strips with a call to `Adafruit_NeoPixel()`. The first parameter is the number of pixels in the strip, the second parameter is the pin number they're on, and the final parameter is set depending on the version of the neopixels you're using. The above is for version two, which are the only ones currently available.

There are three methods that you can call on the strips that you've set up: `begin()` has to be called at the start to set everything up; `show()` has to be called

any time you make a change to a pixel's colour, otherwise the change won't be sent to the pixel; and

`setPixelColor()` is used to change the colour of the pixel. This last method takes four parameters: the pixel number (starting with 0, the closest to the Lilypad), and the R,G and B values respectively.

At this point, we found that our board emitted a high-pitched hum due to a noisy power supply. It wasn't a huge problem, but it was a little annoying. We added a 220µF capacitor between the positive and negative rails to stop this.

Add buttons to the circuit

Once you've got everything working, it's time to move on to the second stage: adding buttons. These are slightly more difficult because you need to solder on the resistors first. See figure 2 for details about how to

solder them. Other than that, it's just a case of sewing them in place. It's best to position them in such a way that the resistors won't get bent repeatedly, as this could lead to metal fatigue and breakage.

Once this is done, you can upload the final code. Even though the hardware isn't quite finished yet (we haven't added the buzzer), the rest of the code will work, and the buzzer will start working as soon as it's put in place.

The code is fairly simple, though a bit long-winded:

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel strip1 = Adafruit_NeoPixel(2, 2, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel strip2 = Adafruit_NeoPixel(2, 3, NEO_GRB + NEO_KHZ800);
int count;

void setup() {
  strip1.begin();
  strip2.begin();
  strip2.show();
  strip1.show();
  pinMode(10, INPUT);
  pinMode(9, INPUT);
  pinMode(11, OUTPUT);
  count = 0;
}

void loop() {
  if(digitalRead(9)){
    if(count < 8){
      analogWrite(11,100);
      #left indicator on
    }
    else {
      analogWrite(11,0);
      #left indicator off
    }
  }
  else if(digitalRead(10)){
    if(count < 8){
      analogWrite(11,100);
      #right indicator on
    }
    else {
      analogWrite(11,0);
      #right indicator off
    }
  }
  if(digitalRead(9)==LOW && digitalRead(10)==LOW){
    analogWrite(11,0);
    if(count < 4){
      #flash one red light
    }
    else if(count < 8){
      #flash next red light
    }
    else if(count < 12){
      #flash next red light
    }
  }
}
```

“Once you’ve got everything working, it’s time to move on to the second stage: adding buttons.”

```

    }
    else {
        #flash final red light
    }
}

if (count < 16) {
    count++;
}
else{
    count = 0;
}
delay(100);
}
    
```

Some of the code has been replaced with comments for brevity. You can find the full code at www.linuxvoice.com/code/wearable.tar.gz as **jacket_final**. Each of the sections with comments is replaced by a section of **setPixelColour()** and **show()** calls to the various strips.

The loop uses the variable **count** to keep track of things flashing. The two new pieces in this are the digital inputs and the analog writes. You should be able to see what's going on here. You have to first set the **pinMode()** in setup with the pin number and the mode you want the pin in. This allows you to read or write to the pins.

You should now have a working cycling jacket!

Make some noise

The buzzer was simple to attach. We used a drop of glue to attach it to the collar of the jacket, then sewed the positive lead onto pin 11 and the negative lead onto a ground thread. Sewing onto an already stitched thread is just like sewing onto a wire loop or a resistor.

Equipment

You need surprisingly little equipment to produce wearable computers. In fact, it's possible that you could do it with nothing but a needle and conductive thread. We only used two pieces of electronics equipment in producing the tutorial: a soldering iron and a multimeter.

There's a wide range of soldering irons available in a wide range of price brackets. The soldering in this project is about as simple as it comes, so any old iron should do the job. If getting a soldering iron for the first time, it's well worth getting a stand and tip cleaner as well. They shouldn't cost much, and make soldering a lot easier.

Usually in electronics tutorials, you'll see multimeters listed as useful but not essential equipment. However, in wearable projects using conductive thread, getting contacts is far more problematic than in most projects. Without a multimeter, trying to find what's causing the problem would have taken us a long time. Because of this, we're inclined to say that a multimeter is an essential tool for wearables. A good multimeter will have a continuity indicator that beeps if there's a connection between two points. This enables you find the problems with contacts without having to keep looking at the screen. This isn't essential, as you can use the resistance meter to do the same job, though the latter way requires you to look away from the circuit to get a reading.



The author has yet to be hit by a car when wearing the jacket despite cycling around the mean streets of Gloucester at night. NB: Linux Voice strongly recommends wearing a helmet while cycling, as brains are soft and squishy.

The **analogWrite()** function that we've used to control the buzzer is a bit misnamed. It's not really setting an analogue value, but a digital pulse width modulation (PWM) value. This means it emits a square wave that's on for the proportion of time you set it to be (out of 255). So **analogWrite(11,0)** sets pin 11 to be off. **AnalogWrite(11,1)** sets pin 11 to switch on for one 255th of the cycle. **analogWrite(11,100)**, then, sets pin 11 to be on for almost half of the cycle. The frequency of the PWM is dependent on the timers of the Arduino. These can be changed, but it's a little complicated and can have effects on other functions.

Since we just want to make a noise to alert the cyclist to the fact that the indicators are on, we won't bother interfering with it. The code as written should produce a high-pitched beep. If you want something a little more tuneful, there are some example of coding melodies in Files > Examples > Digital in the Arduino IDE. The buzzer makes it much easier to check you haven't accidentally left the indicator on.

We've created a cycling jacket, but exactly the same techniques could be used to produce all sorts of wearable designs. If you're a pop star embarking on a world tour and need something to wear, or feel like making your own Tron costume, this project is an excellent place to start. 📺

Ben Everard cycled across Somalia once, and says it wasn't as dangerous as the time he cycled across Wales.

HUNT COMETS WITH PYTHON AND OPEN DATA

ANDREW CONWAY

Hunt for celestial bodies from the comfort of your own home, with the SOHO satellite and the power of Python.

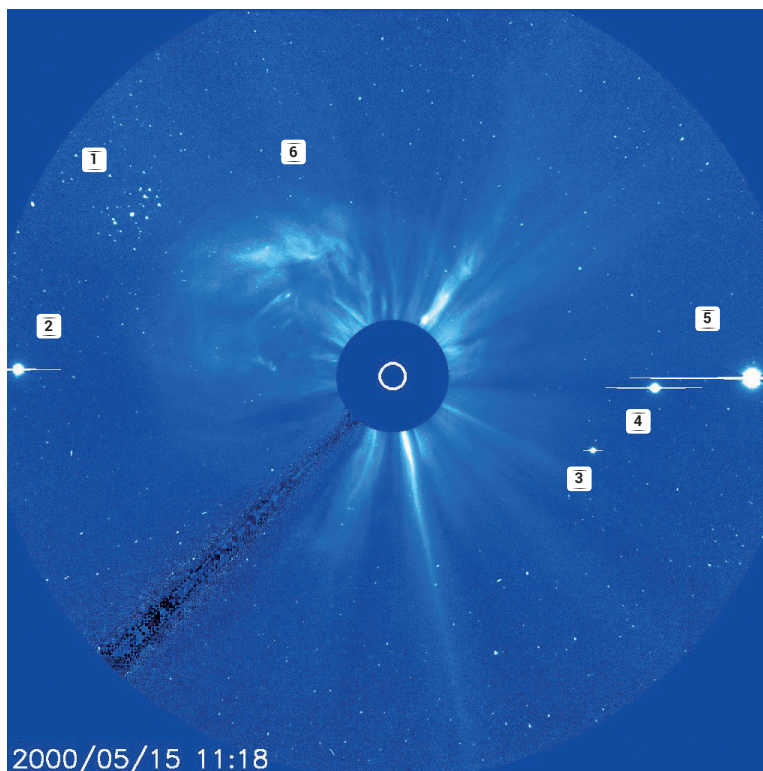
Would you like to discover a comet? Of course you would. But perhaps the thought of staring into the void with giant binoculars or a telescope, night after freezing night, for years on end, to find just one, tiny smudge might be less appealing. How about discovering a comet while sitting in a warm room wearing only your underwear, or better still, getting your computer to do it?

It may surprise you, but we cannot predict when comets will appear in our skies. Halley's Comet, and a few others, are exceptions to the rule. Most comets are spotted by chance as faint specks moving through the stars, and that's what we'll be looking for using a proven source of images: the LASCO instrument on the SOHO satellite (SOlar and Heliospheric Observatory). Its image data is released under public

domain, as with almost all NASA data, and although it's only looking at a few degrees of the sky around the Sun, this is a good place to find comets, as explained in the Sungrazers boxout, right. LASCO actually has several cameras, but we'll be using its C3 camera, as its smaller field of view makes it easier to work with.

In a typical LASCO image, there's a circle in the centre representing the disk of the Sun (called the photosphere in astronomers' lingo) but that's deliberately blotted out by a larger disk so we can see fainter objects around the Sun. The fuzzy stuff is the corona, the outer atmosphere of the Sun and the start of the solar wind – LASCO's main purpose is to study that. The SOHO spacecraft is in orbit around the Sun, and LASCO keeps it in the centre of its view, which means that stars, planets and comets will all be seen moving across the image.

MY GOD... IT'S FULL OF STARS



A view from SOHO's LASCO C3 camera that shows many stars, including the Pleiades star cluster (1) along with four planets, which are overexposed with horizontal lines running through them. From left to right: Mercury (2), Saturn (3), Jupiter (4) and Venus (5). Also, the Sun is blowing off a Coronal Mass Ejection (CME) to the top left (6). Most of the blobs are not stars or planets or comets, but are in fact cosmic rays striking the detector.

Spot the difference

Finding a comet does not involve frightening physics – it's more like a game of spot the difference using many images. It's tricky because there are lots of objects that can be confused with a comet.

The easiest objects to rule out are planets. Mercury, Venus, Mars, Jupiter and Saturn are all very bright and so easy to spot, as shown in the blue LASCO C3 image (left). Uranus and Neptune and a host of other objects such as Pluto and asteroids are much fainter, but they too can be ruled out because we know where they are going to be at any time. The Earth doesn't make an appearance in SOHO images because it is always behind the satellite.

Stars can be easily identified because their movement over time is predictable: they march across the image in formation from left to right, at about three pixels per hour in LASCO C3. Comets usually move diagonally, and at a different rates.

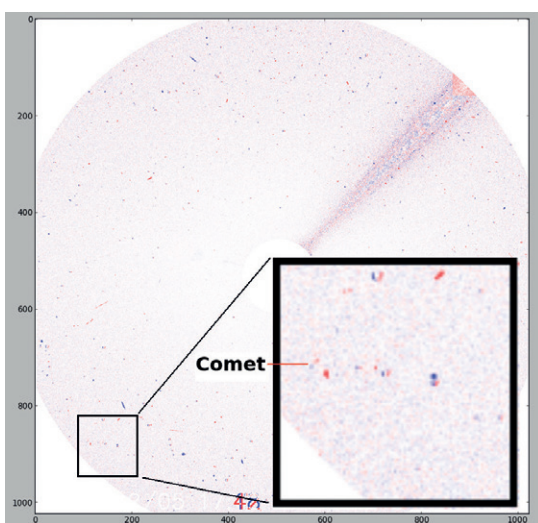
So once stars and planets are ruled out, anything that's left must be a comet, yes? Unfortunately not. There are many comet-like smudges on all SOHO images that are caused by cosmic rays. These are high energy particles from anywhere in the cosmos that strike the detector and fool it into thinking that light has been detected. Fortunately, these are easy to rule out because they only affect one image. If the smudge is present in one image, but completely gone in the next, then it's a cosmic ray.

Before automating any task, it's informative to try it manually. Thankfully that's easy to do here because

some test examples are available at the sungrazer comet page at the US Naval Research Lab (yes, the US military let their staff research comets... but why is a long story!). If you go to <http://sungrazer.nrl.navy.mil/index.php?p=guide> and scroll down you'll find a section called Strategy And Tips and in that is a list of Zip files that you can download so you can hone your comet-hunting skills. Inside each Zip file you will find a series of LASCO images, and a cheat-sheet telling you where the comet is (you're not going to peek first, are you?) Download the Zip file and open up the first image in the series using your image viewer. Click on the Next button (the default image viewers in Ubuntu/Unity and Slackware/KDE both have one) and look at the sequence of images. Unless you have the visual acuity of Robocop, you will not see a comet, but instead gain an appreciation for how difficult it can be to find one, even when you know it's there!

Manual experience

Take a deep breath. Pour yourself a relevant beverage (I like coffee or Raspberry Pi brewed beer) and read the instructions on the sungrazer page more carefully. There's one important clue that will narrow down your search: most comets approach the Sun from a particular direction that depends on time of year. Have a look at this page to get an idea of where to look and when http://sungrazer.nrl.navy.mil/index.php?p=comet_tracks. Even with this information, you might still find yourself tearing your hair out, because some comets are very faint. Try the example named **soho1264**, because that comet is relatively bright. If you flick back and forth between the images taken at 1718 and 1742, you should be able to see the comet in the bottom-left corner moving towards the centre of image. (Did you have to peek in the cheat-sheet? It's OK, I did too first time round.)



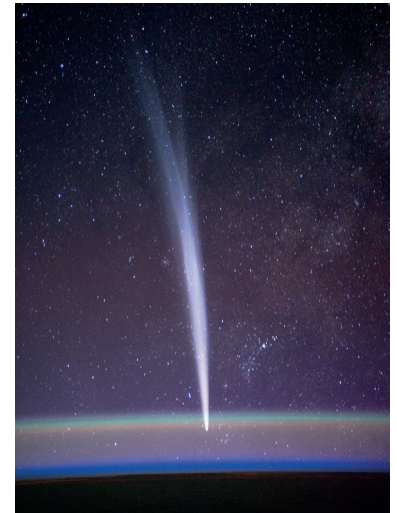
This image shows the difference between images taken at 17.18 and 17.42 on 5th Feb 2007 by SOHO LASCO/C3. Red blobs show features present at 17.42 but not at 17.18 and vice versa for blue blobs. The broad line in the top-right of the image is the pylon holding the central coronagraph disk in place. The inset shows the area around the comet.

Comets and sungrazers

Comets are often described as dirty snowballs. They are lumps of loosely bound ice, rock and dust, left over from the formation of the Solar System. Most of them hang around in what's called the Oort cloud, which is well beyond the orbit of all the Sun's planets. Once in a while, something disturbs the cloud and a comet is sent into the inner Solar System, and then we might see it.

Some comets, called sungrazers, pass very close to the Sun, which has a surface temperature of about 5500°C and is chucking out energy in the form of electromagnetic radiation (ie light) with a power of about $3.8 \times 10^{26} \text{W}$ (yes, that W means watts, the same unit used for lightbulbs!) Each square metre of the solar surface emits energy at a rate equal to 62,000,000 W – think 62,000 bars of an electric fire. Even if these numbers boggle your mind, I'm sure it's clear that this is going to cause a problem for an icy object like a comet. In fact, many comets don't survive a close encounter with the Sun. In December 2013, Comet ISON looked promising, but it perished in the intense solar radiation. Other sungrazers fare better, but are much disrupted, such as comet Lovejoy in 2011, pictured. Luckier ones will be fragmented into many small pieces, and each one will become a comet in its own right.

It's thought that a big comet broke up back in the year 1106 AD and fragments of that have provided us with many great sungrazing comets over the centuries. This group is called the Kreutz sungrazers, and 85% of comets found by SOHO are in this group.



Kreutz sungrazer comet Lovejoy only just survived its close encounter with the Sun in late 2011.

You should now be able to appreciate our plan: 1) load a pairs of images; 2) difference them; 3) clean the differenced image; 4) identify objects; 5) repeat and track objects in subsequent images. We'll concentrate on 1–4, because if these are done right, step 5 is relatively easy.

Automating with numpy, scipy and matplotlib

First, install the new Python modules we'll need. On Debian-based distros:

```
sudo apt-get install python-numpy python-scipy python-matplotlib
```

Numpy is a numeric library for Python that provides lots of new ways to work with arrays. Scipy is a library that performs all kinds of science-related data processing, and Matplotlib will make short work of displaying the images. We're going to use numpy and Scipy to load up an image file and turn it into a 2D array of numbers. You can put the following commands in a file called **comet.py**, save it and enter **python comet.py** on the command line, or you can just enter **python** on the command line and type them in line by line. First, we'll load up the first image of the soho1264 that shows the comet:

```
import scipy
image1=scipy.misc.imread('full_soho1264_070205_1718.gif',
flatten=1)
import matplotlib.pyplot as plt
imgplt=plt.imshow(image1)
imgplt.set_cmap('gray')
```

LV PRO TIP

These techniques are useful for things besides comet hunting, such as image processing.

plt.show()

You should now see a LASCO C3 image in a Matplotlib window. We've loaded the image using **imread** and flattened it, which means each pixel becomes a brightness value with no colour information. Each value will be a float between 0.0 and 255.0 inclusive and is stored in the Numpy array called **image1** which has dimensions 1024 by 1024. We then display the image with the **'gray'** colour map.

Next, we'll take a difference of two images. Close the first image window and enter the following in the same interactive Python session (or into your **.py** file):

```
image2=scipy.misc.imread('full_soho1264_070205_1742.gif',
flatten=1)
```

```
import numpy as np
```

```
diff=np.subtract(image2,image1)
```

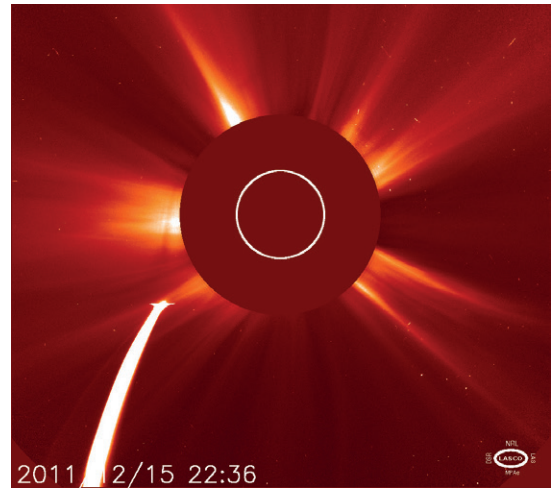
```
imgplot=plt.imshow(diff)
```

```
imgplot.set_cmap('bwr')
```

```
plt.show()
```

We've loaded the image taken 24 minutes later at 17.42, then used Numpy's **subtract** function, which takes each pixel in the second image and subtracts the value of the pixel at the same co-ordinates in the first image and returns the result to a new array we call **diff**. We then display **diff** using the colour map **bwr**, which stands for blue-white-red. This means that features that only appear in the second image show as red; features that only in the first image show as blue; and areas of no difference are white.

If you look closely at the difference image, you'll see that there are many isolated blue or red blobs that correspond to cosmic ray artefacts only present in one or other image. In a few places there is a red spot immediately to the right of a blue spot – these are stars. If you look very carefully at the bottom-left of the image, and if your monitor is very clean, you might just see the comet: a faint red smudge above and to the right of the a similar blue smudge. The fact that this smudge is moving diagonally across the image



Comet Lovejoy (officially C 2011 W3) nearing the Sun, as seen by SOHO LASCO's C2 camera.

towards the Sun is strongly suggestive of a comet, but based on two images alone we can't be sure that it's not just a happy coincidence of cosmic rays.

Clean and identify

Starting with the **diff** image we obtained above, we'll now produce a cleaned image containing only objects that showed up blue:

```
x=diff[824:924,100:200].astype(int)
```

```
xt=np.where(x<-50, x,0)
```

```
d1=np.where(xt==0, xt,-1)
```

First we convert the **diff** array to type **int** and select a 100 by 100 square in the lower-left corner. This may seem like a cheat, but the sungrazer site tells us that's where a Kreutz sungrazer would enter the image in February. On the next line we use Numpy's **where** command to set all pixels that are greater than -50 in value to zero. It works by testing each pixel for the condition specified in the first argument, **x<-50**: if true, the second argument is used to fill the value in new pixel array, and if not, the third argument is used. The resulting array will only contain strong blue blobs, that is, features prominent in image1 but not image2. We then use the **where** command again to set all remaining non-zero pixels to -1, which will make identifying the blobs much easier. We are being rather brutal here and throwing away a lot of data, eg assuming pixels between -50 and 0 are uninteresting noise, but we can fine-tune parameters later if we suspect we're missing comets.

We now have an image **d1** in which each pixel is either 0 or -1. Next, we use Scipy's cunning label function to identify all blue blobs, which are just groups of pixels with value -1:

```
from scipy.ndimage import label
```

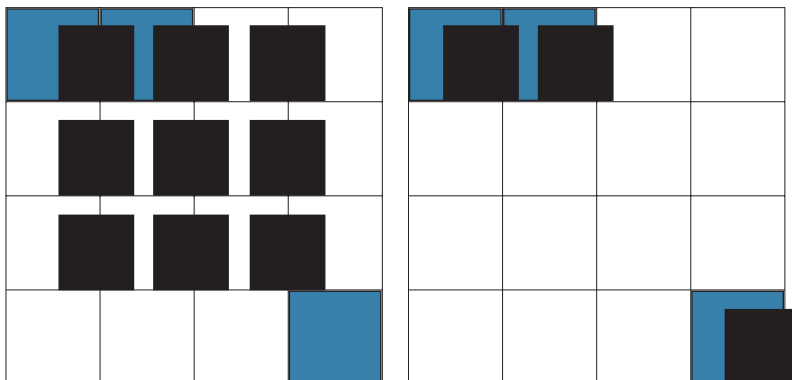
```
l1, n1 = label(d1, scipy.ones((3,3)))
```

There's a lot going on in that second line. We give the label function the cleaned differenced image **d1** and also **scipy.ones((3,3))**, which is a 3 by 3 array in which all elements are 1. This is asking **label** to look at all possible 3 by 3 grids within the image, and if it finds

Scipy's label function

Left: A 4 by 4 image, in which three pixels (shown in cyan) have the same value, is given to label to be scanned with a 3 by 3 grid. Right: No 3 by 3 grid can be drawn

containing the top left two pixels and the one at bottom right, so label will return a 4 by 4 array labelling them as two separate blobs, here labelled as 5 and 6.



The label function groups adjacent pixels with the same value into numbered blobs.

two pixels with the same non-zero value inside a 3 by 3 grid, it assigns them to the same blob.

Next, we repeat all of the above to label red blobs, except with a threshold of +50:

```
xt=np.where(x>50, x,0)
```

```
d2=np.where(xt==0, xt,1)
```

```
l2, n2 = label(d2, scipy.ones((3,3)))
```

The end result is that $n1=11$ (11 blue blobs) and $n2=15$ (15 red blobs). The **l1** array is a 100 by 100 array in which each element is zero (nothing there), or is a number between 1 and 11 indicating which blue blob that pixel belongs to, with the **l2** array being similar except that it contains 15 blobs.

Great success

We've now narrowed down our search from many thousands of blobs to about a dozen. That's pretty good going!

It's worth visualising our cleaned difference images to appreciate how good (or brutal) our clean-up has been. To do this, add together the cleaned red and blue images with `imshow(d1+d2)` and use the **bwr** colour map, as described above. You should be able to see a few pairs of red and blue blobs that are stars, and another pair moving diagonally – our comet!

We now need to pair red and blue blobs that are within a certain radius of each other. The sungrazer website says that Kreutz group comets typically move less than 10 pixels per hour in C3 images, and we know that stars move even more slowly than that, so let's set our search radius to a little more than that, at 15 pixels per hour. There's a 24-minute time difference between our two images, so our search radius will be $(24/60)*15=6$. Next we're going to look at all pairs of blobs and see which red and blue blobs are within our search radius:

```
import scipy.ndimage.measurements
```

```
pairs=list()
```

```
centres1=scipy.ndimage.measurements.center_of_
mass(d1,l1,range(1,n1+1))
```

```
centres2=scipy.ndimage.measurements.center_of_
mass(d2,l2,range(1,n2+1))
```

```
for c1 in centres1:
```

```
    for c2 in centres2:
```

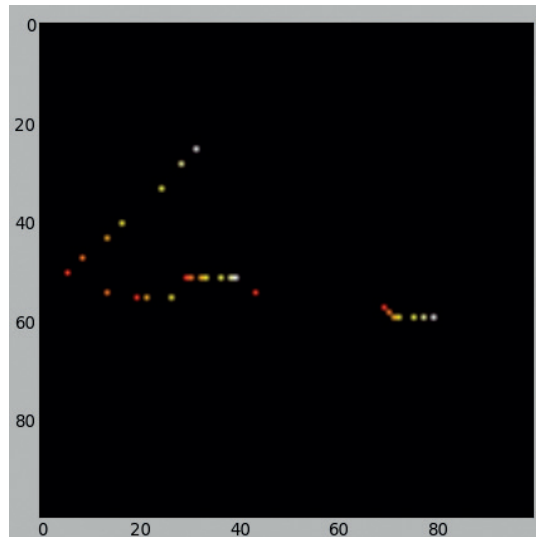
```
        if (c1[0]-c2[0])**2 + (c1[1]-c2[1])**2 < 6*6:
```

```
            pairs.append((c1,c2))
```

```
print len(pairs)
```

This code uses Scipy's `center_of_mass` function to calculate the centres of all the blobs. Then it loops through all possible pairs and if two blobs are within a circle of radius 6 pixels they're appended to the pairs list. The result is that there are 10 pairs.

To investigate further we'd need to repeat the above procedure for the next two images in the sequence, generating a new list of pairs. Since our new `image1` is just our old `image2`, we can expect the new blue blobs to have the same centres as our old red blobs. In this way, we can match up new and old pairs and track objects as they move from image to image. After we've tracked them over several images, all



Tracks of objects for LASCO C3 images on 5 Feb 2007. Dots shown show positions starting at 17.18 (light red) and ending at 20.42 (white). The time intervals vary, eg there's an hour between the fourth and fifth image. The comet is moving diagonally, and stars horizontally.

cosmic ray coincidences should be ruled out and we'll be left with tracks of stars and, hopefully, comets.

With just a few more lines of code it's possible to produce the tracks shown based on seven images from 17.18 to 20.42. The comet is now pretty obvious because of its diagonal motion. The code we've outlined above could do with a lot of refining because it's probably doing too good a job of rejecting false positives, to the point where it might be missing real comets. The best way to improve it is to try it out on other image sequences with known comets in them and experiment with some choices we've made, such as the noise threshold of 50, the 3 by 3 label search grid and the 100 by 100 sub-image.

Go discover comets, and more...

You can download SOHO data from here

<http://sohowww.nascom.nasa.gov/data/realtime-images.html> for any time period, including near real-time images. Images are now provided as JPEG files rather than GIFs, but all the code above will still work. If you do think you've spotted a comet, read the instructions on the sungrazer comets page on how to report it. In the same way that a well-constructed bug report is more likely to get attention from a developer, professional scientists are more likely to accept your discovery if it's presented to them in a way that shows you know your stuff.

Don't stop at comets; you can apply the principles introduced here to look in other data sets, to hunt for asteroids or sunspots, for example. You could also analyse satellite images of the Earth's surface, or even turn your attention to medical images. The human race is drowning in data, especially image data, and so there's every chance that, with a bit of hard work, you could make a real contribution to research by honing and applying basic image processing skills. 📺

Andrew Conway is interested in computers, science, writing and humans, and has been a happy Linux user since 1995.

RASPBERRY PI: BUILD AN EMERGENCY BEACON

Combine simple Python modules with hardware programming to build your own emergency distress beacon.

WHY DO THIS?

- Keep relatively safe from natural disasters.
- Program components connected to the Raspberry Pi's GPIO pins.
- Learn code concepts including loops, data storage and conditional statements.

YOU WILL NEED:

- Raspberry Pi (Model A or B can be used).
- Battery with integrated solar cell (or you could use the Pi powered from the mains).
- PiGlow (Available from Pimoroni.com).
- Buzzer/piezo speaker.
- Soldering iron (optional – I've breadboarded the example diagram for this tutorial).
- Jumper wire (female to male, from Pi to breadboard and male to male for breadboard connections).
- 100 ohm resistor
- Momentary switch (push button).
- Breadboard.
- Insulation tape.
- Micro USB to USB lead (to power the Pi).
- 20cm of wire (shielded, but you could use a female to male jumper wire).
- An FM radio tuned in to 103.3MHz.

The finished PiBeacon project encased inside its protective lunchbox shell.

The background to this project is that I've been working with a class at Mereside Primary School in Blackpool. The children were learning about natural disasters such as tsunamis and earthquakes. During the course of their lessons they learnt that one of the first issues faced by the victims was a loss of communication as mobile phone towers were quickly damaged. The children worked as a team to understand the impact that this would have and how they could make a difference.

Their idea was to create a beacon that attracts help in three ways.

- An FM radio transmitter, that can be tuned to work on many different frequencies.
- An LED unit, to visually attract people to the beacon.
- A buzzer, to attract people using audio output.

The beacon must be completely self supporting and have its own self-charging power source. To accomplish this we found a cheap USB battery pack with a built-in solar cell on Amazon, but for the purposes of this tutorial you can just plug into the mains.

To keep the project as simple as possible we'll use only one method of input, which is a single push button that when pressed will launch the Python code. Finally, the project must be weatherproof, and at this

prototype stage the best solution was every Raspberry Pi hacker's best friend, a plastic lunchbox.

The PiBeacon was entered into PA Consulting's Pi Awards event on 2 April 2014. I am proud to say that my team came second in their year group and really proved how far they had come in such a short time. I'd like to say a very big "well done" to the hackers from Mereside Primary School.

Pin reference

Throughout this tutorial, I will refer to the GPIO pins of the Raspberry Pi via their board reference. With pin 1 being the top-left pin, nearest the SD card slot, and pin 2 being directly to pin 1's right. Please refer to the guide, right, for the location of 3.3V, 5V and ground pins. Don't use these pins unless instructed to do so, but you can use any other pin in your program.

The only user with permission to use the GPIO pins in Raspbian is root, so in order for you to use the GPIO in Idle, open a terminal and type

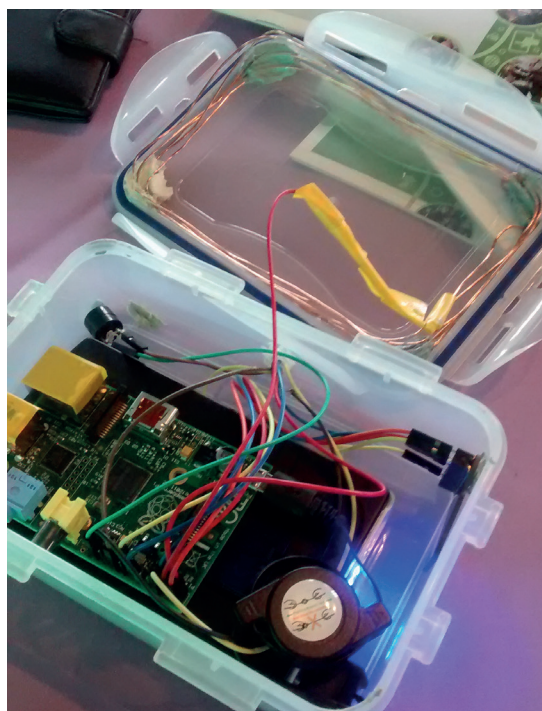
sudo idle

Type in your password (by default in Raspbian this is **raspberry**) and press Enter. In a few seconds the editor for our Python code will be on the screen. By launching Idle in this manner you will be able to access the GPIO pins – just remember to open any Python programs using the File > Open menu option.

Building the project

This build is not complex but it does have four areas that need to be carefully wired together. If you are unsure about your wiring, please ask someone to check before you connect any power to your Pi or attached components.

- **Antenna** This is the most simple section of the build. All you will need to do is attach a maximum of 20cm of wire to pin 4 of your Raspberry Pi. The greater the length of wire, the larger your antenna, but also the greater your signal may become. Please refer to the section on radio transmissions for safety instructions.
- **Button** I used a momentary switch, attached to pin 8 to act as the only method of input. The switch is attached to 3V power from pin 1 and a resistor is used inline with Ground to ensure that the switch does not accidentally trigger from a slight press.
- **Buzzer** A simple buzzer is attached to pin 26 and Ground (pin 20). This buzzer is used as an audio output that will send a message in Morse Code.



■ **PiGlow** Rather than use just one LED, we used 18 super-bright LEDs courtesy of Pimoroni's tiny board.

Normally this board covers all the GPIO pins, but thanks to a phone call with Jon and the team we worked out the minimum number of pins necessary, and these are as follows:

- **Pin 1** 3V3 Logic level voltage.
- **Pin 2** 5V LED source current.
- **Pin 3** SDA i2c Communications.
- **Pin 5** SCL i2c Communication.
- **Pin 14** Ground (GND).
- **Pin 17** Logic level voltage.

Remember when inserting the wires into the PiGlow that you will need to work out where each pin should be inserted. When the board is attached to the GPIO, the "P" of PiGlow should be near the SD card slot. Once you have located Pin 1 of PiGlow, insert a red jumper wire to help you remember that Pin 1 is 3.3V power, and refer to the diagram for more information.

Set up PiGlow, i2c and PiFM

PiGlow uses something called i2c to control the 18 onboard LEDs, and by using i2c PiGlow is able to use far fewer wires than a conventional series of 18 LED would require. I2c was developed by Philips in the 1980s as a means to send data to multiple devices using the a minimal number of wires. It's useful, but the Raspberry Pi does not have i2c set up by default.

To set up i2c on your Raspberry Pi, download a copy of Michael Rimmican's excellent setup script from GitHub: <https://github.com/heed/pi2c>.

Open a terminal, navigate to where you downloaded the file and then used **chmod** to make it executable:

```
chmod +x pi2c.sh
```

Then run the script using **sudo** or as root:

```
sudo .pi2c.sh
```

After a few minutes your Pi will be reconfigured to use i2c; at this time it would be prudent to reboot your Pi to ensure that the configuration is complete.

Now you will need to download the Python library for PiGlow, and luckily Jason Barnett has created a great library for us to use, which is available here: <https://github.com/Boeerb/PiGlow>.

3.3V	1	2	5V
	3	4	5V
	5	6	GND
	7	8	
GND	9	10	
	11	12	
	13	14	GND
	15	16	
3.3V	17	18	
	19	20	GND
	21	22	
	23	24	
GND	25	26	

Pin diagram for Model B Raspberry Pi.

For this project, **piglow.py** will need to be in the same directory as our **beacon.py** code. With these files downloaded, try out some of the examples to ensure that your PiGlow board is working correctly.

Our final requirement is PiFM, a library of code that we can easily drop in to our project to add an FM transmitter. You can download the library from www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter. Extract the files to the same directory as your **beacon.py** and **piglow.py** files. I kept the example audio file – the Star Wars theme – as the audio to play over the airwaves. You could also use any 16-bit mono WAV file.

Coding the project

You can download the code for this project from my GitHub repository: <https://github.com/lesp/PiBeacon>.

We coded this project in Python 2.7 due to its mature collection of libraries and documentation. Libraries enable us to reuse code that other people have written. I used four libraries in my code: **PiFM** to control the radio transmitter; **RPI.GPIO** for GPIO access; **time** to add a delay function to my code; and **PiGlow** to control the PiGlow LED board.

Import the libraries into our code like so:

```
import PiFm
import RPi.GPIO as GPIO
from piglow import PiGlow
from time import *
```

LV PRO TIP
Project files for the PiBeacon are available at <https://github.com/lesp/PiBeacon>.

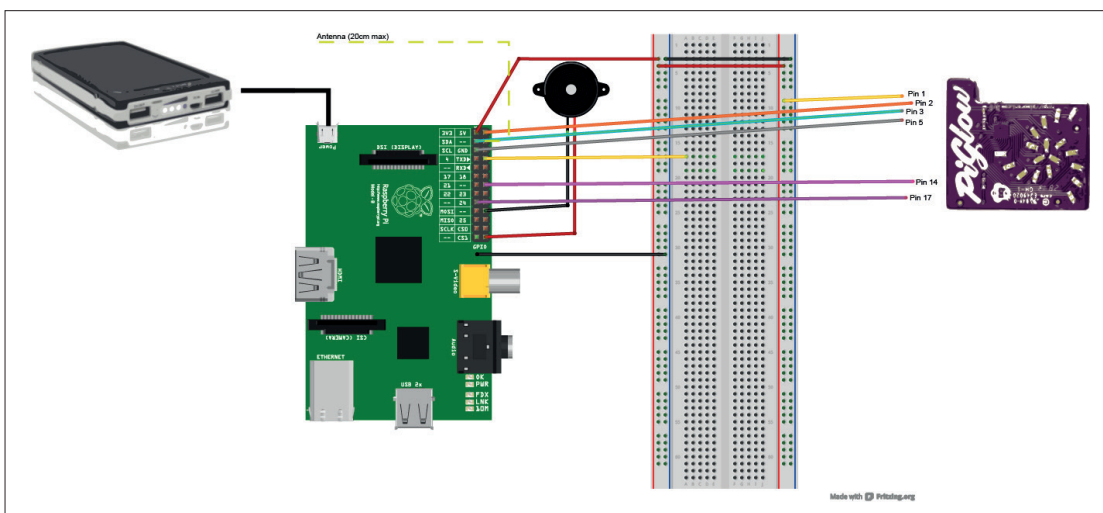


Diagram of the completed setup. Remember to pay careful attention to the GPIO pins for PiGlow.



Next I created two variables: **button_pin** and **buzzer**, and in each one I stored the value of the GPIO pin used for each, respectively 8 and 26. Variables are great, as they enable our program to retain information and act as a data storage system. Variables are used to replace hard coded values in our code. For example I could've used the integers 8 and 26 throughout my code, but if I wanted to change those numbers to something else, then I would have to go through every line of code to make the change. Because we're using a variable, we can simply change the value of that variable once and that

change is reflected whenever we refer to the variable name.

In order to use the GPIO we need to tell Python how we want to use it:

“Variables enable our program to retain information and act as a data storage system.”

GPIO.setmode(GPIO.BOARD)

This tells the Pi that I wish to use the numbering as per the earlier diagram.

GPIO.setup(button_pin , GPIO.IN)

GPIO.setup(buzzer , GPIO.OUT)

These two lines tell the Pi that our button, attached to pin 8, is an input and that our buzzer on pin 26 is an output. Remember that the variables **button_pin** and **buzzer** both contain the pin reference for each.

To make it easier for me to use the PiGlow function, **PiGlow()**, I next create a variable called **piglow**:

piglow = PiGlow()

Later on I use the code

piglow.all(128)

to set all of the LEDs to half brightness, but I'll cover that in more detail later.

Now we come to the main part of the program. In order to control the program we use an infinite loop, which in Python is '**while True:**'. This is the simplest kind of loop, and for the purpose of this project, is the most practical. Any code contained in this loop will run over and over until it is stopped.

The next line is a conditional statement that checks to see if the button has been pressed. This, coupled with our infinite loop, enables the program to constantly check for user input via the button:

while True:

if GPIO.input(button_pin)==1:

So now that we have a conditional statement, what do we want it to do if the condition is true? Well firstly I want it to print "Button Pressed", for debugging purposes, so that I can see that the code has worked. Then I want the code to start PiFm and play the Star Wars theme. The code is as so:

print("BUTTON PRESSED")

PiFm.play_sound("/home/pi/sound.wav")

Once PiFm has finished playing the audio I want to then start a loop that iterates three times. Inside this loop I want the buzzer and PiGlow to provide output in the form of Morse code – more specifically the internationally recognised SOS message (... --- ...).

To create the iterated loop I used a 'for' loop with a range that starts at zero and ends before three, so it goes 0,1,2. A 'for' loop is a loop that will iterate through a list, range or tuple until complete, giving us a the limited number of loops that we require. This gives us the three iterations that we require. Here's the code:

for i in range(0,3):

You might be wondering where the **i** came from? Well, this is a variable that we've declared "on the fly". You could replace **i** with **x**, **y** or **z** if you wished. The **range(0,3)** bit instructs the for loop to start at 0 and count to 2, as 3 is the limit of our range. By counting from 0 to 2 we have 3 loops.

Send signals

Now to make the buzzer and PiGlow come to life. We have to tell the GPIO to send electricity to the buzzer, and to do that we use the Boolean term "True" to say that we want to turn the power on. Remember I earlier set up the GPIO pin 26 as an output and used a variable called **buzzer** to represent this. So now to send the power to the pin I use the following code.

GPIO.output(buzzer, True)

To turn the buzzer off I change the **True** to **False**.

For PiGlow it is a little bit different but by no means a challenge. To illuminate all of the LED on the board I use **piglow.all**. Now as you will see in the code there is a number contained in brackets. This number is the brightness of the LED, with 0 being off and 255 being full brightness. I used 128, which is the halfway point between the two. A word of warning: PiGlow is extremely bright, so be careful with your eyes. Here's how to turn the LED on.

piglow.all(128)

Radio transmissions

This project uses a Python library called PiFM, which is available from www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter. This library is what powers the PiBeacon's radio transmissions. It's very versatile, with extra functionality such as broadcasting in stereo and using a microphone connected to your Pi to broadcast live audio over the airwaves.

Transmitting radio signals is not to be taken lightly, and great care should be taken when using this project. Make sure that you are not operating

on any frequencies that are reserved for emergency services or aviation, otherwise you will get in trouble with the authorities. Please refer to the official guidance available from <http://stakeholders.ofcom.org.uk/enforcement/spectrum-enforcement/law>, as there are certain regulations that must be followed when using radio transmitters.

The FM transmitter is also very powerful – so powerful in fact that if used incorrectly it can cause interference. Best practice would be to reduce the length of wire used in the build so that the effect is

localised. The use of SOS audio messages or SOS Morse code is also not to be broadcast on the radio spectrum, so please just play the theme from Star Wars or Transformers and save the emergency for the real thing.

If you are still unsure then the best resource to use is your local amateur radio group (basically a LUG for those interested in radio related topics). A quick Google search will find your local group, who will be able to answer any questions that you may have. Remember: hack responsibly.

And to turn off the LED we create a new line, which is identical to before but with the (128) changed to (0).

To control which letter is being communicated in Morse I used a delay function, which in Python is called `sleep()`. To create a dot, which is a short beep in Morse I kept the delay to a minimum and set it to 0.5, which is half a second. To create a dash, which is a longer sound, I used a delay of 1, which is 1 second. In code the delays look like this.

```
sleep(0.5) # For a DOT
```

```
sleep(1) # For a DASH
```

The last section of code is the else statement. When using a conditional such as `if`, we can use an `else` statement to capture any unexpected conditions. In this case the `else` statement is used when no user input is detected, it will print "Waiting for input" over and over. As soon as user input is detected, the `else` condition is no longer true and the `if` condition, when the button is pressed, is now true.

Before you test your project it would be prudent to check all of the connections and wiring before you start the program. Once you're happy that everything is as it should be, run your code. You can do this in Idle via the Run > Run Module menu item.

Grab your radio and tune in to 103.3MHz FM, which is the default frequency that we will be using for this project. You should now see the shell printing "Waiting for input" so go ahead and press the button. A moment later you should hear the theme from Star Wars playing through your FM radio. A few minutes later, once the music has finished, your buzzer and PiGlow will start emitting a message in Morse code. Congratulations: you have built a working PiBeacon!

Bonus points – change your message

In this project we use `sleep()` to control the delay for our beeps and flashes, with half a second for a dot and one second for a dash. So using just dots and dashes we can communicate text and numbers.

Instead of broadcasting SOS, let's say "Linux Voice". First of all we'll refer to a chart of Morse Code.

L	DOT DASH DOT DOT
I	DOT DOT
N	DASH DOT
U	DOT DOT DASH
X	DASH DOT DOT DASH
V	DOT DOT DOT DASH
O	DASH DASH DASH
I	DOT DOT
C	DASH DOT DASH DOT
E	DOT

Why don't you try altering the example code to output this message instead?

Here's how to write L in Morse using Python

```
#The letter L in Morse code.
```

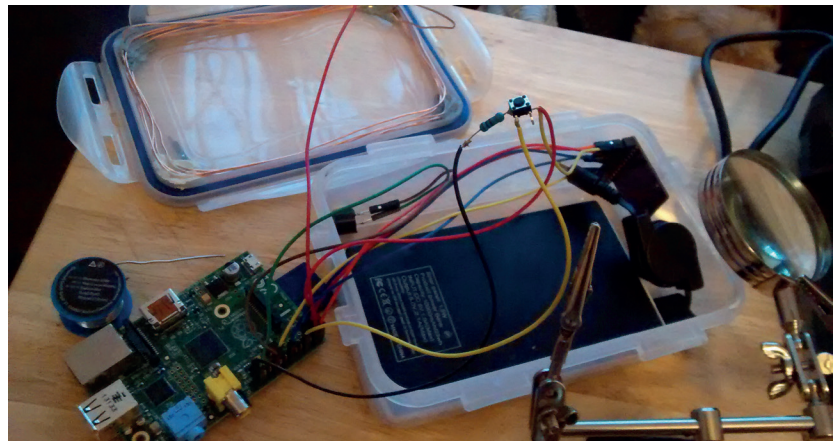
```
#DOT
```

```
GPIO.output(buzzer, True)
```

```
piglow.all(128)
```

```
sleep(0.5)
```

```
GPIO.output(buzzer, False)
```



Assembling the final prototype and soldering the connections was essential to qualify for the PA Consulting competition.

```

piglow.all(0)
#End of DOT, now a 1 second pause
sleep(1)
#DASH
GPIO.output(buzzer, True)
piglow.all(128)
sleep(1)
GPIO.output(buzzer, False)
piglow.all(0)
sleep(1)
#End of DASH, now a 1 second pause
#DOT
GPIO.output(buzzer, True)
piglow.all(128)
sleep(0.5)
GPIO.output(buzzer, False)
piglow.all(0)
#End of DOT, now a 1 second pause
sleep(1)
#DOT
GPIO.output(buzzer, True)
piglow.all(128)
sleep(0.5)
GPIO.output(buzzer, False)
piglow.all(0)
#End of DOT, now a 1 second pause
sleep(1)

```

So what have we accomplished here?

- We have built the hardware that powers our project.
- Using Python and libraries from external sources we have created the code that controls the components in the beacon.

We also used programming concepts such as Loops, to control the flow of our program and to repeat repetitive tasks; variables, to store the values of GPIO pins in one section of code, enabling us to quickly make changes to one value that are reflected throughout the program; and conditionals to control the flow of our program by using logic. The next step is to play with the lights on the PiGlow – you could even create an animation. 📺

Les Pounder is a maker and hacker specialising in the Raspberry Pi and Arduino. Les travels the UK training teachers in the new computing curriculum and Raspberry Pi.

CONTROL VIRTUAL MACHINES WITH PYTHON AND LIBVIRT

VALENTINE SINITSYN

Learn ways to automate VM management when GUIs and simple shell scripts aren't enough.

WHY DO THIS?

- Automate virtual machine maintenance and management processes.
- Batch-create virtual appliances for clouds, integration testing and so forth.
- Get to know the de-facto standard virtualisation toolkit for Linux.

If you read Linux Voice, you are probably a Linux user. And if you use Linux, you most likely know what virtualisation is. Many mainstream distributions include KVM and virt-manager these days, and you can easily install Oracle VM VirtualBox, Xen or such like. Usually, they provide some form of GUI, so why on the Earth would you want to try virtualisation from a Python script?

If you just want to try out a new distro, you probably wouldn't. However, if you use several virtual machine managers (VMMs, or hypervisors) in parallel, or create pre-configured virtual machine appliances (say, for a cloud deployment), Python may come in handy.

Meet libvirt

Born at Red Hat as an open-source project, libvirt has become an industrial-grade toolkit that provides a generic management layer on top of different hypervisors, using XML as a mediation language. It's been adopted by many Linux vendors (if you have virt-manager, you have libvirt) and has bindings for many programming languages, including Python (version 2 and, starting with libvirt-python 1.2.1, Python 3). Libvirt can create (or "define", in its parlance), run ("create") and destroy virtual machines (called "domains" here), provide them with storage, connect them to virtual networks that are protected by network filters, migrate them between nodes and do other smart things.

However, libvirt has no convenient tools to work with XML, so you'll need to know the format (described at libvirt's website, www.libvirt.org) and use `xml.etree` or similar. Let's see it in action. Install libvirt's Python bindings (usually called `python-libvirt`

or alike) and open an interactive Python shell (>>> denotes prompts in the listings below). No root privileges are initially required, but you may be asked to obtain them when needed.

```
$ python
>>> import libvirt
>>> conn = libvirt.openReadOnly('qemu:///system')
```

Here, we import the `libvirt` module and open a connection to the hypervisor specified by the URI (note the three slashes). In this tutorial we'll work with Qemu/KVM, which is probably the most 'native' VMM for libvirt. `/system` means we connect to a local system-level hypervisor instance. You may also use `qemu:///session` to connect to the local per-user Qemu instance, or `qemu+ssh://` for secure remote connections. We are not going to define new domains now, so the restricted read-only connection will suffice.

For starters, let's check what your host is capable of when it comes to the virtualisation:

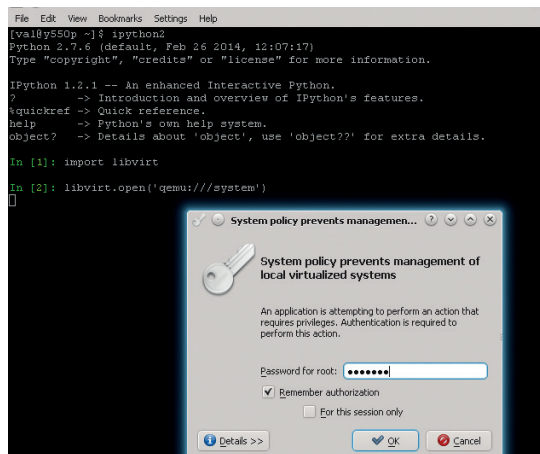
```
>>> xml = conn.getCapabilities()
>>> print xml
<capabilities>
<host>
<uuid>20873631-dad7-dd11-885a-08606eda31ae</uuid>
<cpu>
<arch>x86_64</arch>
<model>Westmere</model>
<vendor>Intel</vendor>
<topology sockets='1' cores='4' threads='1'/>
<feature name='vmx'/>
...
</capabilities>
```

You see how the XML is used to describe the host's capabilities. Libvirt identifies objects (hosts, guests, networks etc) by UUIDs. My host is a 64-bit quad-core Intel Core i5 with hardware virtualisation (VMX) support. Your results will likely be different.

The XML is quite long (note the ellipsis). Here's how you can use `xml.etree` to get supported guest domain types and corresponding architectures from it:

```
>>> from xml.etree import ElementTree
>>> for guest in tree.findall('guest'):
...     arch = guest.find('arch').get('name')
...     domain_type = guest.find('arch/domain').get('type')
```

My stock Ubuntu 13.10 supports Qemu domains only. However, since Qemu is a generic emulator, I can virtualise almost anything including s390x or SPARC (albeit at a performance penalty). x86_64 and i686 are of course supported, too.



Depending on the settings, you may be asked to enter the root password to use a system connection.

It's good to know that you can create a domain for any conceivable architecture, but how do you actually do it? First of all, you'll need some XML to describe the domain. For simple cases, it may look like this:

```
<?xml version="1.0"?>
<domain type='qemu'>
  <name>Linux-0.2</name>
  <uuid>ce1326f0-a9a0-11e3-a5e2-0800200c9a66</uuid>
  <memory>131072</memory>
  <currentMemory>131072</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type>hvm</type>
    <boot dev='hd'>
  </os>
  <devices>
    <disk type='file' device='disk'>
      <source file='/path/to/linux-0.2.img/'>
        <target dev='hda'>
      </disk>
    <interface type='network'>
      <source network='default'>
    </interface>
    <graphics type='vnc' port='5900'>
  </devices>
</domain>
```

Speak the domain language

Here, we create a Qemu/KVM (hvm) virtual machine with one CPU and 128MB of RAM. It has a hard disk at IDE primary master (**hda**), from which it boots (I've used the tiny Linux 0.2 image from the Qemu Testing page). It is connected to the "default" network (NAT-enabled 192.168.122.0/24 attached to virbr0 at the host side), and you can use VNC at port 5900/tcp to access its screen (try **vinagre localhost:5900** or similar). Note that the `<source file="..."/>` must contain an absolute path to the image, and the image format must be supported by the hypervisor. libvirt is not a tool to create disk images, however you can use `pyparted`, `ubuntu-vm-builder` or similar to automate this process with Python.

Domains in libvirt are either transient or persistent. The former exist only until the guest is stopped or the

host is restarted. Persistent domains last forever and must be defined before start. A transient domain will do for now, but as we are going to create something, a read-only connection is no longer sufficient.

```
import libvirt
xml = """domain definition here"""
conn = libvirt.open('qemu:///system')
domain = conn.createXML(xml)
```

Yeah, that's all. However, if you try to execute this script, you may get this response:

```
libvirt: QEMU Driver error : internal error: Network 'default' is not active.
```

This is because the XML references the "default" network, which won't be active unless there are domains using it already running, or you have marked it as autostarted with `virsh net-autostart default` command. Insert the following code just before `conn.createXML()` call to start the network if it is not already active:

```
net = conn.networkLookupByName('default')
if not net.isActive():
    net.create()
```

First, we get an object representing the "default" network. libvirt can look up objects by names, UUID strings (`ce1326f0-a9a0-11e3-a5e2-0800200c9a66`) or UUID binary values (`UUID("ce1326f0-a9a0-11e3-a5e2-0800200c9a66").bytes`). Corresponding method names start with the object's type (except for domains) followed by "LookupByName", "LookupByUUIDString" or "LookupByUUID", respectively.

Network objects provide other methods you may find useful. For instance, you can mark a network as autostarted with `net.setAutostart(True)`. Or, you can get an XML definition for the network (or any other libvirt object) with `XMLDesc()`:

```
>>> print net.XMLDesc()
<network>
  <name>default</name>
  <uuid>9d3c0912-6683-4128-86df-72f26847d9d3</uuid>
  ...
</network>
```

If we were going to create a persistent domain, we'd change `conn.createXML()` to:

```
domain = conn.defineXML(xml)
domain.create()
```

There and back again

libvirt is essentially a sophisticated translator from a high-level XML to low-level configurations specific to hypervisors. Sometimes you may want to see what libvirt generates from your definitions. You can do this with:

```
>>> print conn.domainXMLToNative('qemu-argv', xml)
LC_ALL=C PATH=... QEMU_AUDIO_DRV=none /usr/bin/
qemu-system-x86_64 -name Linux-0.2 ... -m 128 ... -smp
1,sockets=1,cores=1,threads=1 -uuid ce1326f0-a9a0-11e3-a5e2-
0800200c9a66 ... -vnc 127.0.0.1:0 -vga cirrus...
```

Other times, you may be unsure how to express

some VM configuration in XML, or you may have the configuration autogenerated by another front-end. libvirt can convert a native domain configuration to the XML with:

```
>>> argv="LC_ALL=C PATH=... QEMU_AUDIO_DRV=none /usr/bin/
qemu-system-x86_64 -name Linux-0.2 ... -m 128 ... -smp
1,sockets=1,cores=1,threads=1 -uuid ce1326f0-a9a0-11e3-a5e2-
0800200c9a66..."
>>> print conn.domXMLFromNative('qemu-argv', argv)
```

```
<domain type='qemu' xmlns:qemu='http://libvirt.org/schemas/
domain/qemu/1.0'>
```

```
<name>Linux-0.2</name>
<uuid>ce1326f0-a9a0-11e3-a5e2-0800200c9a66</uuid>
<memory unit='KiB'>131072</memory>
<currentMemory unit='KiB'>131072</currentMemory>
<vcpu placement='static'>1</vcpu>
<os>
  <type arch='x86_64' machine='pc'>hvm</type>
</os>
...
</domain>
```

You can also use `virsh domxml-to-native` and `virsh domxml-from-native` commands for the same purposes.

(remember that persistent domain creation is a two-phase process). To gracefully reboot or shutdown the domain, use `domain.reboot()` and `domain.shutdown()`, respectively. However, the guest can ignore these requests. `domain.reset()` and `domain.destroy()` do the same, albeit without guest OS interaction. When the domain is no longer needed, you can remove (undefine) it like this:

```
try:
    domain = conn.lookupByUUIDString('ce1326f0-a9a0-11e3-
a5e2-0800200c9a66')
    domain.undefine()
except libvirt.libvirtError:
    print 'Domain not found'
```

`lookup*()` throws `libvirtError` if no object was found; many libvirt functions do the same. If the domain is running, `undefine()` will not remove it immediately. Instead, it will make the domain transient. It is an error to undefine a transient domain.

When you are done interacting with the hypervisor, don't forget to close the connection with `conn.close()`. Connections are reference-counted, so they aren't really closed until the last client releases them.

Get'em all

A libvirt system may have many domains defined, and there are several ways to enumerate them. First, `conn.listDomainsID()` returns integer identifiers for the domains currently running on a libvirt system (unlike UUID, these IDs aren't persisted between restarts):

```
for id in conn.listDomainsID():
    domain = conn.lookupByID(id)
    ...
```

If you need all domains regardless of state, use the `conn.listAllDomains()` method. The following code mimics the behaviour of the `virsh list --all` command:

```
print ' Id Name State'
print '-' * 52
for dom in conn.listAllDomains():
    print "%3s %-31s%s" % \
        (dom.ID() if dom.ID() > 0 else '-',
         dom.name(),
         state_to_string(dom.state()))
```

For domains that aren't running, `dom.ID()` returns -1. `dom.state()` yields a two-element list: `state[0]` is a current state (one of `libvirt.VIR_DOMAIN_* constants`), and `state[1]` is the reason why the VM has moved to this state. Reason codes are defined per-state (see `virDomain*Reason enum` in the C API reference for the symbolic constant names). The custom `state_to_string()` function (not shown here) returns a string representation of the code.

Domain objects provide a set of `*stats()` methods to obtain various statistics:

```
cpu_stats = dom.getCPUStats(False)
for (i, cpu) in enumerate(cpu_stats):
    print 'CPU #%d Time: %.2lf sec' % (i, cpu['cpu_time'] /
1000000000.)
```

This way, you get a CPU usage for the domain (in nanoseconds). My host has four CPUs, so there are

Your mileage may vary

You may expect libvirt to abstract all hypervisor details from you. It does not. The API is generic enough, but there are nuances. First, you'll need your guest images in a hypervisor-supported format (use `qemu-img(1)` to convert them). Second, hypervisors vary in their support level. Qemu/KVM and Xen are arguably the best supported options, but we had some issues (like version mismatch or inability to create a transient domain) with libvirt-managed VirtualBox on our Arch Linux and Ubuntu boxes.

The bottom line: libvirt is great, but don't think you can change the hypervisor transparently.

four entries in the `cpu_stats` array. `dom.getCPUStats(True)` aggregates the statistics for all CPUs on the host:

`dom.getCPUStats(True)`

```
>>> print dom.getCPUStats(True)
[{'cpu_time': 10208067024L, 'system_time': 1760000000L,
'user_time': 5830000000L}]
```

Disk usage statistics are provided by the `dom.blockStats()` method:

```
rd_req, rd_bytes, wr_req, wr_bytes, err = dom.blockStats('/path/
to/linux-0.2.img')
```

The returned tuple contains the number of read (write) requests issued, and the actual number of bytes transferred. A block device is specified by the image file path or the device bus name set by the `devices/disk/target[@dev]` element in the domain XML.

To get the network statistics, you'll need the name of the host interface that the domain is connected to (usually `vnetX`). To find it, retrieve the domain XML description (libvirt modifies it at the runtime). Then, look for `devices/interface/target[@dev]` element(s):

```
tree = ElementTree.fromstring(dom.XMLDesc())
iface = tree.find('devices/interface/target').get('dev')
rx_bytes, rx_packets, rx_err, rx_drop, tx_bytes, tx_packets, tx_err,
tx_drop = dom.interfaceStats(iface)
```

The `dom.interfaceStats()` method returns the number of bytes (packets) received (transmitted), and the number of reception/transmission errors.

A thousand words' worth

Imagine you are making a step-by-step guide for an OS installation process. You'll probably do it in the virtual machine, taking the screenshots periodically. At the end of the day you will have a pack of screenshots that you'll need to crop to remove VM window borders. Also, it's pretty boring to have to sit there pressing `PrtSc`. Luckily, there is a better way.

libvirt provides a means to take a snap of what is currently on the domain's screen. The format of the image is hypervisor-specific (for Qemu, it's PPM), however, you can use the Python Imaging Library (PIL) to convert it to anything you want. To transfer image data from the VM, you'll need an object called `stream`. This provides a generic way to exchange data with libvirt, and is implemented by the `virStream` class. Streams are created with the `conn.newStream()` factory function, and they provide `recv()` and `send()`

methods to receive and send data. To get a stream containing the screenshot, use:

```
stream = conn.newStream()
dom = conn.lookupByUUID(UUID('ce1326f0-a9a0-11e3-a5e2-0800200c9a66')).bytes)
if dom.isActive():
    dom.screenshot(stream, 0)
```

Here, we lookup the domain by a binary UUID value, not a string (the UUID class comes from the `uuid` module). We check that the domain is active (otherwise it has no screen) and ignore other possible errors. Now we need to pump the data to the Python side. `virStream` provides a shortcut method for this purpose:

```
buffer = StringIO()
stream.recvAll(writer, buffer)
stream.finish()
```

Here, we create a StringIO file-like object to store image data. `stream.recvAll()` is a convenience wrapper that reads all data available in the stream. `writer()` function is defined as:

```
def writer(stream, data, buffer):
    buffer.write(data)
```

Its third argument is the same as the second argument in `recvAll()`. It can be an arbitrary value, and here we use it to pass the `StringIO` buffer object.

All that remains is to save the screenshot in a convenient format, like PNG:

```
from PIL import Image
buffer.seek(0)
image = Image.open(buffer)
image.save('screenshot.png')
```

PIL is clever enough to autodetect the source image type. However, it expects to see the image data from byte one, that's why we use `buffer.seek(0)`.

You can easily wrap this screenshotting code into a function and call it periodically, or when something interesting happens to the VM.

You've got a message

When something happens to a domain, for example it is defined, created, destroyed, rebooted or crashed, libvirt generates an event that you can subscribe to and act appropriately. To be able to receive these events, you'll need some event loop in your code. libvirt provides a default one, built on top of the blocking `poll(2)` system call. However, you can easily integrate with Tornado `IOLoop` (LV1) or `glib MainLoop` (LV2), if needed.

Default event loop is registered at the very beginning, even before the connection to libvirt daemon is opened:

```
libvirt.virEventRegisterDefaultImpl()
conn = libvirt.open('qemu:///system')
```

Next, you subscribe to the events you are interested in. Let's say we want to receive events of any type:

```
cb_id = conn.domainEventRegisterAny(None, libvirt.VIR_DOMAIN_EVENT_ID_LIFECYCLE, event_callback, None)
```

The first argument is the domain we want to monitor; `None` means any. The second argument

```
SeaBIOS (version 1.7.3-20130708_231806-aatxe)
Machine UUID ce1326f0-a9a0-11e3-a5e2-0800200c9a66

iPXE (http://ipxe.org) 00:03:0 C900 PCI2.10 PnP PMM+07FC2C60+07
...
Booting from Hard Disk...
LIL0 boot: _
```

specifies the event "family" to subscribe to. Here, we are interested in lifecycle events (started, stopped, etc), but there are many others (removable device changed, power management occurs, watchdog fired, and so on). The last argument is an arbitrary value to be passed to the `event_callback()` function (remember `stream.recvAll()` and `writer()` we saw earlier?).

Event handler is defined as follows:

```
def event_callback(conn, domain, event, detail, opaque):
    print 'Event #%d (detail #%d) occurred in %s' % (event, detail, domain.name())
    event and detail are integer codes describing what happened. For lifecycle events, they are defined in the virDomainEventType and virDomainEvent*DetailType enums; the constants (libvirt.VIR_DOMAIN_EVENT_STARTED etc) are named the same as enum fields.
while True:
    libvirt.virEventRunDefaultImpl()
```

This is the main loop. In a real application, you will probably run it in a separate thread. The call blocks until a subscribed event (or a timeout) occurs, so even exiting with `Ctrl+C` takes some time.

When the subscription is no longer needed, you can terminate it with:

```
conn.domainEventDeregisterAny(cb_id)
```

Events notification opens many interesting possibilities. For instance, you can start domains in the particular order (one after another), or use the Tornado framework to create a lightweight web-based `virt-manager` alternative.

And there's more...

This concludes our quick tour of the features of libvirt. We've barely scratched the surface, and there is much more than we've seen so far: storage pools, encryption, network filters, migrations, nodes, Open vSwitch integration and the rest. However, the APIs you've learned today form a solid foundation to build more advanced libvirt skills for your next project. Let the computer do the repetitive work for you, and have fun with Python in the meantime! 🐍

Dr Valentine Sinityn has committer rights in KDE but spends his time mastering virtualisation and doing clever things with Python.

You can take a screenshot of the VM as early as you want, even before a guest kernel is booted.

EVERYTHING IS AN OBJECT IN RUBY

Everything in Ruby is an object. Even procs, includes, lambdas, classes, singleton classes, superclasses and everything else...

Ruby is an interpreted object-oriented programming language that has gained popularity in recent years, perhaps due to the popularity of the Ruby on Rails web application framework. Its basic syntax is easy to learn and programmers adopting the language quickly become comfortable writing Ruby code.

As a new Ruby programmer (Rubyist), however, you're immersed in a world of classes, objects, methods and class methods, instance variables and class instance variables, class variables, singleton and proxy classes, procs, lambdas and blocks. You can be forgiven for appearing dazed and confused.

Some of the basic concepts are easy to understand: terms like classes and instances are easily understood by anyone with some object oriented programming experience. But all this terminology boils down to one basic model, and understanding that model can bring clarity and understanding of everything else, making you a better Rubyist.

We're going to focus on one key fact about Ruby – that everything is an object – and learn what this means for us as Ruby programmers. By understanding what an object is we can unleash Ruby's power and understand some of those concepts that might at first appear confusing.

If you have done your homework you will know about two kinds of object: an instance and a class; the latter being like a blueprint used to create the former.

The four freedoms

All objects have four characteristics in common: identity, state, being an instance of some class and the ability to receive messages. We'll explain each of these in turn.

Each object has a unique identity, it's object ID, which you can reveal it with its `object_id` method.

```
> "hello".object_id
```

```
=> 24750480
> 0.object_id
=> 1
```

Inside Ruby the object ID is a pointer to the location in memory where the object is stored. Following a pointer has more overhead than accessing data directly so, for performance reasons, there are a few kinds of object that encode their data within their object ID. They're explained in the Small Objects boxout on page 100. For most objects, however, the ID refers to a location in memory that contains some internal flags, a table of variables and a pointer to the class that the object is an instance of.

Ruby variables contain object IDs, and this means that they behave like pointer variables in other languages, as the following example illustrates:

```
a = "abc"
b = a
a.upcase!
puts b # => "ABC"
```

The object's state is represented by its table of variables. You can peek into any object and see these instance variables:

```
puts myobject.instance_variables
```

The object's class pointer points to the class that the object is an instance of. You can ask an object what its class is:

```
puts "abc".class # => String
```

An object receives a message when a method is called on it. In this sense, we talk about the object as the receiver and express the message as a method call like this:

```
myobject.a_method(parameters)
```

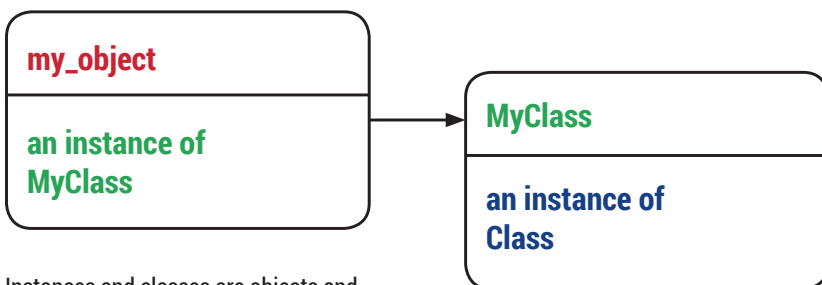
But our object definition didn't mention methods. What happens when an object receives a message is that it follows its class pointer to find the method.

The method is defined on the class so that all its instances can use it. Despite this, it is usual to say that "myobject has a method called my_method" instead of "MyClass has a method called my_method" even though that's the reality.

To clarify that methods defined on a class are called on an instance, we call them instance methods and this can be expressed in Ruby:

```
"my string".methods == String.instance_methods
```

We can summarise our understanding of an object as being an instance of a class with its own instance variables and the use of the instance methods defined by its class.



Instances and classes are objects and all objects are instances of a class.

Going back to our ‘everything is an object philosophy’, it follows that a class is an object too. This means that everything that we have said so far also applies to classes: a class is an instance of another class. It can, therefore, have its own instance variables and can also use instance methods defined by its class.

A class is an instance

Ruby allows a class to be derived from another: it supports single inheritance. Such a class is a subclass of the one it inherits, which is its superclass. Its upward chain of superclasses is called its ancestors and you can peek into a class to see them:

```
puts String.ancestors
```

The chain of ancestors is used to locate a method when an object receives a message. It looks first in its class and, if it finds nothing in there, iterates through its ancestors until it is found. Thus, a class can define new methods or redefine methods already defined by one of its ancestors. If no method is found then the object is sent a new message called **method_missing**.

When a method executes, it does so in the context of the receiver, which we call the current object, and uses its instance variables (as you would expect). The current object has a special name in Ruby: **self**.

Summarising our understanding of classes, we can say that a class is an object that can be instantiated and can have instance methods and a superclass. But there’s another very important kind of class that you need to understand – the Singleton class.

In a class of its own

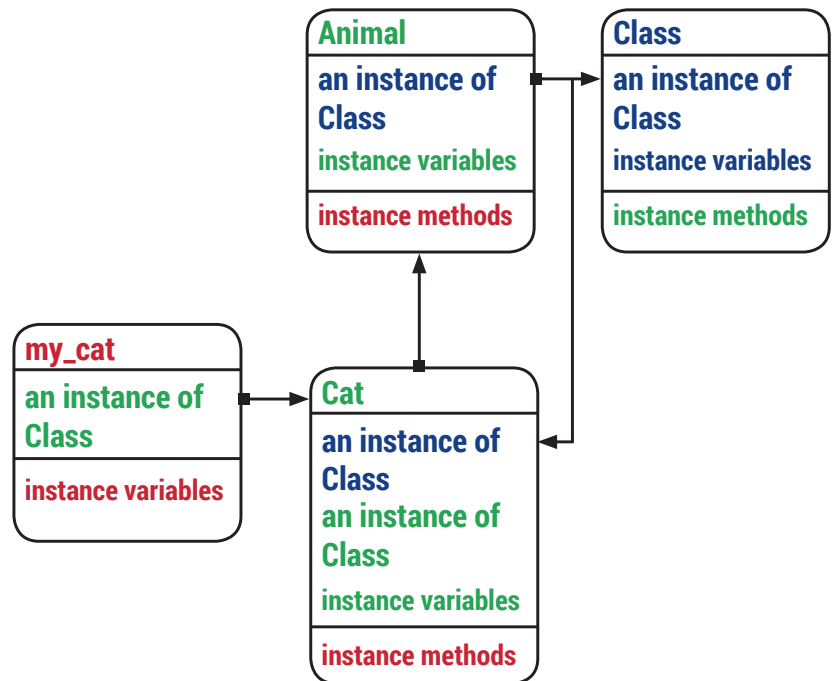
We explained that a class can define new methods or redefine those already defined by its ancestors. The singleton class enables an instance to do the same, allowing it to have methods that other instances of its class do not have. You can do this:

```
a = "abc"
def a.rot13
  tr 'A-Za-z','N-ZA-Mn-za-m'
end
a.rot13 # => nop
```

We just defined a method called **rot13** on an object, **a**, which is an instance of the **String** class. We know that the methods accessible to an object are in its class, which is **String**. But if we defined the method in the class then it would be accessible to its instances, which isn’t the case:

Everything is an object

Some experienced Rubyists may prefer to say that almost everything is an object because you can’t define something entirely in terms of itself. Ruby’s syntax is not an object and variables aren’t objects. So while it may technically make more sense to say that every value is an object, for the purpose of understanding objects, this fact can be overlooked. ruby-lang.org says that everything is an object so that’s good enough for us.



A class can be a subclass as well as an instance. They can have instance variables and instance methods. An object’s accessible instance variables and instance methods are shown in the same colour as its name. If a method isn’t defined by its class, move upwards through its ancestors to find it.

```
b = "def"
b.rot13 # => NoMethodError: undefined method `rot13' for
"def":String
```

So, where is it? Well, it’s in a singleton class. What Ruby does is create a new anonymous class that becomes the object’s class and its actual class becomes the superclass of the anonymous class. The singleton class is invisible, however:

```
a.class # => String
```

Singleton classes are everywhere in Ruby; just very well hidden. It’s common practice to create class methods using a construct like this:

```
class MyClass
  def self.class_method
    "hello"
  end
end
MyClass.class_method # => hello
```

This is exactly the same as our **rot13** example – a method called **class_method** is created in the singleton class for the object, **MyClass**, which lies between that object and its visible superclass, **Class**.

The singleton class is sometimes referred to as a metaclass, virtual class or eigenclass; Eigen being a German word meaning one’s self. That fact helps explain the way we just used the **self** keyword.

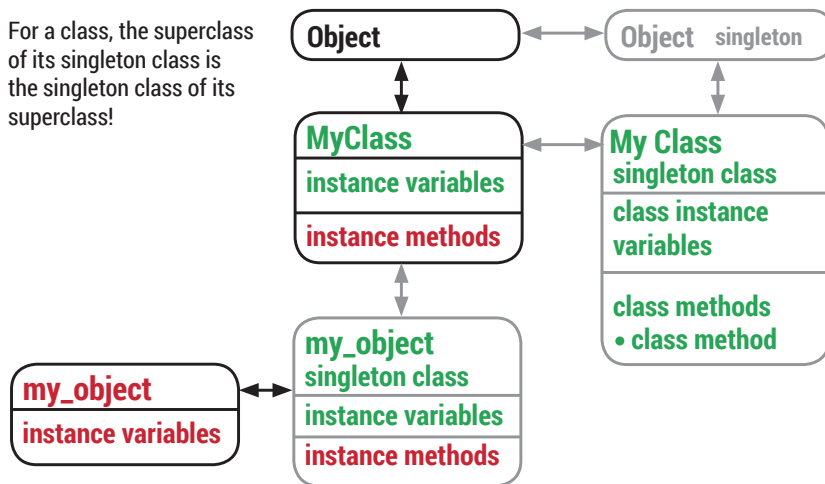
We already mentioned that **self** refers to the current object. We now extend our definition to say that the current object becomes the class being defined when inside a class definition. In the example above, **self** refers to the class, **MyClass**, that is being defined.

Another common idiom in Ruby is to open a singleton class and code directly inside it. We could

PRO TIP

You can try the code snippets yourself in **irb** or try <http://rubyfiddle.com>.

For a class, the superclass of its singleton class is the singleton class of its superclass!



have written the above example like this:

```
class MyClass
  class << self
    def class_method
      "hello"
    end
  end
end
```

which you may prefer when defining lots of class methods together. And an example that really shows off **self**'s changing persona is this **eigenclass** method:

```
class Object
  def eigenclass
    class << self
      self
    end
  end
end
```

It adds a new method to the **Object** class, an ancestor of all classes and a good place to define methods that should be available everywhere. Inside the **eigenclass** instance method, **self** is the current

object, the one that the method was called on. It uses **class << self** to open the current object's singleton class, and the second use of **self** is inside that and, therefore, the method returns the singleton class. Ruby 1.9 introduced a new method called **singleton_class** into the **Object** class that does this.

The magic of modules

There is another kind of object that is very similar to a class; so similar in fact that the only real difference is that it cannot be instantiated: the Module. Everything we've said about instance variables and instance methods apply equally to modules, but they differ from classes in the way they are used.

You don't inherit modules – you **include** them, and you can include many modules. This is how you can achieve the effect of multiple inheritance in Ruby. When a module is used within a class, it is called a **mix-in**, because it is mixed in to the class.

```
class MyClass
  include UsefulModule
  include AnotherUsefulModule
  ...
end
```

Including a module is another occasion when Ruby creates an anonymous class that, in this case, is known as an include or proxy class. It is inserted into the ancestor chain of the including class, becoming its superclass. The proxy class points to the module's instance methods. Proxy classes enter the ancestor chain in the order that modules are included into the class, so this ordering is significant in the event that modules contain methods with the same name.

The fact that we said instance methods is important – when you include a module you don't get class methods, because although, like any other class, a module has a singleton class, it does not get included. That is to say, if you do

```
module MyModule
  def self.class_method
  end
end
```

and include that module in a class, the method will not be available. Any object can, however, include a module's instance methods into its singleton class. When a class does this they become class methods.

```
module MyModule
  def class_method
    "hello"
  end
end
class MyClass
  class << self
    include MyModule
  end
end
puts MyClass.class_method # => hello
```

Because this is a common idiom, Ruby provides the **extend** method to do just that. This is more succinct.

```
class MyClass
```

Small objects

Some kinds of objects are not represented by an object structure, because the data they contain takes up less space than a reference to an object structure would. Instead, they encode their data within their object ID, and they do so for performance reasons.

Because memory addresses are always aligned with the word length of the CPU (32-bit systems have a 4-byte word length, and it's 8 bytes on 64-bit systems), all the least significant bits of a pointer are always zero. Ruby uses this space to identify small objects, which is demonstrated nicely by small integers:

```
> 20.object_id
=> 41
> 20.object_id >> 1
=> 20
```

By small integer, we mean any signed integer that can be held in the space reserved for an object ID, less one bit (so on

a 64-bit machine this means a 63-bit signed integer). The reserved bit is used as a flag that tells Ruby what the object is. The value is stored bit-shifted left and with the least-significant bit set to 1. In this way, all object IDs that are odd numbers represent integer objects. In Ruby, these numbers are the **FixNum** class. Ruby 2.0 introduced a similar encoding on 64-bit architectures for some small Float values that it calls Flonums.

Another small object is the symbol. These are used every day in Ruby, as keys to hashes among other things. They are immutable strings, meaning that they cannot be modified once created.

Ruby creates symbol objects whenever new symbols are referenced. Variables and other references to the same symbol all refer to the same symbol object. Comparing symbols is more efficient than strings, because it's only necessary to compare their object IDs.


```

extend MyModule
end

```

If you want both class and instance methods in the same module then you have to resort to some trickery that uses callbacks and a sub-module. The idea is that you create a sub-module and put the to-be class methods there. You then rely on an included callback (a method in your code that is invoked when the module is included in a class) to extend the class with the sub-module:

```

module MyModule
  def self.included(including_class)
    including_class.extend ClassMethods
  end
  module ClassMethods
    def class_method
      "class method"
    end
  end
  def instance_method
    "instance method"
  end
end
class MyClass; include MyModule; end
puts MyClass.class_method # => class method
my_object = MyClass.new
puts my_object.instance_method # => instance_method

```

Another use for modules is name-spacing, which allows classes or modules with the same names to be used without conflict. The `::` scope resolution operator allows constant definitions (classes and modules have object IDs that are constants) to be nested.

```

module Custom
  class String
  end
end
puts Custom::String == String# => false

```

You may also see modules used to implement the singleton pattern, which is often confused with the singleton classes we've previously explained, but just means a class that can only be instantiated once. Modules can be used to model these, and a typical example is a logging object.

```

module Logger
  def self.log(msg)
    @@log ||= File.open("log.txt", "a")
    @@log.puts(msg)
  end
end

```

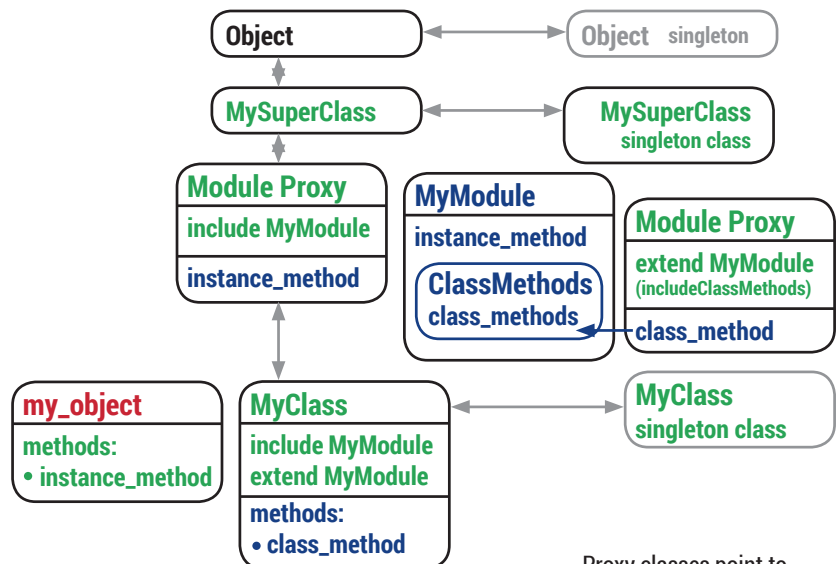
You can then **include** `Logger` wherever logging is needed and get a `log` method at your disposal.

An object you can call

The last group of objects that we need to cover are callable objects. You've seen this construct before:

```
(0..10).each { |n| puts n }
```

The bit between the braces is a block (you can also use `do` and `end`) – a chunk of Ruby code that can be passed into a method. A method can take one block that can use the mystical `yield` to execute it. Here is a



Proxy classes point to module methods.

simple example:

```

def my_method
  yield if block_given?
end
my_method { puts "I'm a block!" }

```

We use `block_given?` to ensure we have a block before we `yield` to it. If we didn't do this, Ruby would raise a `LocalJumpError` if the method was invoked without one. The block must be given after, and outside, the method's argument list. It can optionally take arguments.

A block is not an object, but it can be converted into one called a **Proc**. This happens if the method specifies a last parameter that is preceded with an ampersand (`&`). This assigns the block to a variable, which requires that it is an object. **Proc** objects may be assigned to variables but blocks can't, hence the conversion. Because they aren't converted to objects, blocks enjoy a slight performance advantage. They respond to `yield`, whereas a **Proc** must be called:

```

def my_method(&proc)
  proc.call
end
my_method { puts "I'm a block!" }

```

You can explicitly create **Proc** objects and pass them as regular parameters.

```

def my_method(proc)
  proc.call
end
proc = Proc.new { puts "I'm a proc!" }
my_method(proc)

```

A neat feature of a block or **Proc** object is that it's a closure – a fancy term which means that it remembers the environment that existed when it was created, even if that environment no longer exists. That means that the state of all variables in scope when it was created are remembered, even if they have since ceased to exist.

A variation on a **Proc** is the **lambda** or anonymous method, created like this:

```
l = lambda { |n| puts "Hello #{n}, I'm a lambda" }
```

Or, since Ruby 1.9, with an alternative syntax:

```
l = ->(n){puts "Hello #{n}, I'm a lambda" }
```

The one you use is a matter of personal taste. Although still **Proc** objects, there are a couple of behavioural differences to be aware of.

First, you will notice that a **Proc** is not sensitive to the number of arguments given – if insufficient are given, the missing ones become **nil**. Lambdas, on the other hand will raise an error in this scenario (as any method would).

The other difference is how they behave when called. A **Proc** behaves like it has been inserted into the calling method and, so, a **return** in a **Proc** will cause the surrounding method to return. Because the lambda is a method, a return inside one will just return control to the surrounding method. To summarise their differences, remember that “**Proc**” behaves like a block and “**lambda**” behaves like a method.

The final object we’re going to mention is one we’ve been using all along. Methods are objects too. You can get any method as an object and use it in place of a **proc** or **lambda**.

```
def foo() "foo" end  
def bar(m) m.call end  
m = method(:foo) # => #<Method: Object#foo>  
puts bar(m) # => foo
```

And that’s the object of Ruby. Understanding what Ruby’s objects are and how they work will help to clarify what makes the language tick and, hopefully, will lead to some wonderful coding experiences. Here be no dragons! 🐉

John Lane is a technology consultant with a penchant for Linux. He helps new business start-ups make the most of open source.

Challenges

Test your skills by writing Ruby code to solve the following puzzle

This is a small Ruby program to test knowledge of the Ruby Object Model. It creates one superclass, two subclasses and two instances of each of those. The superclass includes a module that defines instance variables and their accessors. In total there are eight objects and eight variables. All the variables have the same name, **@var**, and accessors called **var** and **var=**.

The variables are

- 1 Module instance variable.
- 2 Class instance variables.
- 4 Instance variables.

The module assigns default values to each variable. There is a different default value for each of the three types. This program refers to those defaults as **DEF_MIV**, **DEF_CIV** and **DEF_IV** and tests their assignment.

The program uses the accessors to assign values to the variables and applies a series of tests to check which variables are the same or different. The test methods are coded in a separate file, **tests.rb**.

The module is coded in **my_module.rb**. It is your challenge to write this module. Extra brownie points can be scored if the program still works after changing the superclass to use **extend** instead of **include** (line 35) without modifying anything else, including your module.

Ruby 1.9 or later should be fine. This was written with **ruby-2.0.0-p353**.

```
require_relative 'my_module'  
require_relative 'tests'  
  
class MySuperClass; include MyModule; end  
class MyClassA < MySuperClass; end  
class MyClassB < MySuperClass; end
```

```
my_object_a1 = MyClassA.new  
my_object_a2 = MyClassA.new  
my_object_b1 = MyClassB.new  
my_object_b2 = MyClassB.new
```

```
Show default values  
dump(MyModule,MySuperClass,MyClassA,MyClassB,my_object_a1,my_object_a2,my_object_b1,my_object_b2)
```

```
Test default values  
different(MyModule::DEF_MIV,MyModule::DEF_CIV,MyModule::DEF_IV)  
same(MyModule::DEF_MIV,MyModule.var)  
same(MyModule::DEF_CIV,MySuperClass.var,MyClassA.var,MyClassB.var)  
same(MyModule::DEF_IV,my_object_a1.var,my_object_a2.var,my_object_b1.var,my_object_b2.var)
```

```
Assign values  
MyModule::var = "this is a module instance variable"  
MySuperClass.var = "this is a class instance variable in MySuperClass"  
MyClassA.var = "this is a class instance variable in MyClassA"  
MyClassB.var = "this is a class instance variable in MyClassB"
```

```
my_object_a1.var = "this is an instance variable in my_object_a1"  
my_object_a2.var = "this is an instance variable in my_object_a2"  
my_object_b1.var = "this is an instance variable in my_object_b1"  
my_object_b2.var = "this is an instance variable in my_object_b2"
```

```
Show assigned values  
dump(MyModule,MySuperClass,MyClassA,MyClassB,my_object_a1,my_object_a2,my_object_b1,my_object_b2)
```

```
Test assigned values  
different(my_object_a1.var,my_object_a2.var,my_object_b1.var,my_object_b2.var,MyModule::var,MySuperClass.var,MyClassA.var,MyClassB.var)  
same(my_object_a1.class.var,my_object_a2.class.var)  
same(my_object_b1.class.var,my_object_b2.class.var)  
different(my_object_a1.class.var,my_object_b1.class.var)  
different(my_object_a2.class.var,my_object_b2.class.var)  
same(my_object_a1.class.superclass.var,my_object_a2.class.superclass.var,my_object_b1.class.superclass.var) 🐉
```

LINUX VOICE ISSUE 1 COMPETITION RESULTS

The fastest and smallest solutions to our crossword-solving puzzle from Linux Voice issue 1



When we set a competition in Linux Voice issue 1, we had no idea what would happen. While it seemed like a good idea to us, there was always the chance we'd get no entries, and have to sheepishly admit that here instead of announcing results. These fears were quickly assuaged when the first entry rolled in the same day the issue first arrived with subscribers (Thanks Jose!).

To refresh your memory, the competition was to write a Bash script that used grep to solve a crossword. There were two categories: the smallest script, and the fastest execution. The best entry in each category wins an exclusive Linux Voice winner's T-shirt.

The smallest

The prize for the shortest solution goes to Steve Engledow, creator of this ingenious one-liner.

```
s=`cat -`;for i in `cat /usr/share/dict/words|grep ^...`;do echo $s|grep -o "$i";done
```

To make it a bit more readable, here it is with some white space:

```
s=`cat -`
for i in `cat /usr/share/dict/words|grep ^...`
do
    echo $s | grep -o "$i"
done
```

The first line concatenates all the input lines into a single string. Then the for loop goes through every line in the words file that's longer than three letters.

The grep match is a little different to how most people solved the problem. Rather than create a regex based on the text in the wordsearch and try to find a match in the words file, it uses the **words** file and tries to find a match in the wordsearch. Normally this wouldn't work, since grep would output the line in the wordsearch, which would include text either side of the word. However,

this program gets around this by using the **-o** flag. This means it just outputs the section of the text that matches the regular expression, and not anything else.

The fastest

There was one script that ran faster than the rest. Much faster. It was:

```
#!/bin/bash
for line in `cat | python gen_regexs.py`; do
    egrep "^$line$" /usr/share/dict/words
done
exit 0
```

This needs the file **gen_regexs.py**, which is here:

```
import sys

def get_regex(word):
    regexs = []
    for i in range(0, len(word)-2):
        for j in range(i+3, len(word)+1):
            regexs.append(word[i:j])
    return regexs

def make_regex(word):
    return("(%s)" % ("|".join(get_regex(word))))

if __name__ == "__main__":
    for word in sys.stdin:
        print make_regex(word.strip())
```

This is obviously not just a Bash script; it's a Bash script and a Python script. This is certainly stretching the rules quite a bit, so we had to decide whether or not to allow it. The rules stated that the program had to be a Bash script that matched using a form of grep. There's no denying that this is what this does. The Python code only generates the regular expression against which the Bash script looks for matches.


After much deliberation, we decided that while this stretched the rules, it didn't actually break them. After all, sed, perl and awk scripts are regularly included in shell

scripts, so why not Python? It's within the spirit of Bash programming to include other tools in this way.

This entry ran in a little under 0.1 seconds on our machine. This was more than 2,000 times faster than the shortest entry, and a couple of times faster than its nearest rival.

The panel of expert judges were delighted to see two very different approaches to the problem solve it in two very different ways, and get very different results because they were optimised for different things. Congratulations to Richey Delaney for winning this aspect of the competition.

In future competitions, feel free to e-mail us if you're unsure about a point on the rules, and we'll give a ruling.

We'll end with a massive thank you to everyone who entered, with such diverse approaches to the problem. We hope that everyone had some fun and learned a bit along the way. 



This could be yours! The exclusive Linux Voice leet T-shirt is only available to the winners of the Linux Voice challenges. Take a look at the opposite page for details of how to enter this month's competition.

GENETIC ALGORITHMS: CREATE LIFE WITH PYTHON

Everything's easy in Python. Even things that aren't easy to solve can be evolved with a little generic magic.

Computers follow a series of instructions step by step until they get to the end. This series of instructions is called a program. However, what if something can't be calculated with a series of step-by-step instructions? Or what if the series of step-by-step instructions would take so long to complete that running them is impractical?

In these cases, we need a method that side-steps the main problem, but still attempts to find an answer. One way to do this is to use genetic algorithms. This mimics the natural process of evolution to attempt to solve a problem through a combination of randomisation, selection and combination.

The basic method goes like this:

- Create a random set of data in the right format to solve the problem.
- Apply some test to see which of the data solve the problem best.
- Combine the best pieces of data, throw in a little randomness and go back to step two.

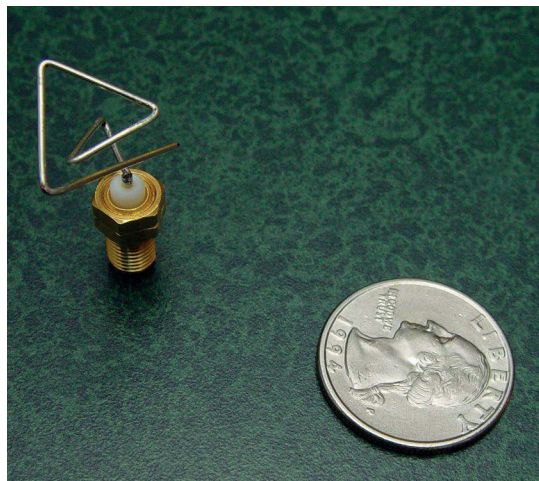
In the real world, this is how we became us. Initially, there were some primeval organisms with some DNA

and not much else.

This was step 1. The weakest of these organisms died off leaving only the strongest. This was step 2. These

remaining organisms reproduced. This is step 3. The final two steps have been repeating ever since life on earth started, and we are the result, as are all the other living things.

“Genetic algorithms mimic the natural process of evolution to solve a problem.”



NASA use genetic algorithms to find the best antenna designs for spacecraft.

To model this computationally, the key thing we need is a fitness selector. This is the test that we'll apply to the data to see if it should pass on its characteristics to the next generation, or if it should be pruned from the evolutionary tree leaving stronger data to go forward.

Essentially, it's this function that defines your genetic algorithm and what data it will search for – it turns programming around, so that you write a program specifying what the solution should look like, then leave the computer to work out what it is.

Genetic square roots

Let's take a look at an example. There isn't actually an easy way to calculate square roots, however, it is very easy to go the other way around and calculate the square of a number. So, this is the sort of problem that genetic algorithms are good at. We'll program it in Python using the **pyevolve** module. This may be in your distro's repositories in the **python-pyevolve** package, or you can get it with

pip install stallion

The evaluation function for our square root finder is:

```
def eval_func(chromosome):
```

```
    score = 0.0
```

```
    for value in chromosome:
```

```
        score += 100000000-abs(((square_root_of-(value*value))))
```

```
    return score
```

This function will be passed a list of data that represent the organism that you're evaluating. In the case of our square root calculator, this will have just a single value, but in other cases, it could hold many values representing different aspects of the organism.

It returns a value that the genetic algorithm will attempt to maximise. In this case, it will try to maximise **100000000-abs(((square_root_of-(value*value))))**. **abs()** returns the absolute value of a number – this means that it just removes the negative sign on negative numbers, so **abs(10)** is 10, and **abs(-5)** is 5. The **abs()** call in this function, then, will return a larger number the further the value is from the actual square root. However, our algorithm will try to maximise the result, so we want this number to get smaller the further it is from the square root. To do this, we take the result away from 100000000.

We said that this function effectively defines the genetic algorithm, and this is true. However, we do need a bit more code to define the environment that the evolution will take place in. Since genetic

algorithms rely on a certain amount of randomness to find the right values, there's no guarantee that they will ever find the right value. You can increase the chances of them working correctly by tweaking the environment for the particular problem you're trying to solve. The full code we've used is as follows:

```

from pyevolve import *

square_root_of = 1000

def eval_func(chromosome):
    score = 0.0
    for value in chromosome:
        score += 100000000-abs(((square_root_of-(value*value))))
    return score

genome = G1DList.G1DList(1)
genome.evaluator.set(eval_func)
genome.setParams(rangemin=0, rangemax=int(square_root_of/2))
genome.crossover.set(Crossovers.G1DListCrossoverUniform)

ga = GSimpleGA.GSimpleGA(genome)
ga.setPopulationSize(square_root_of)
ga.setGenerations(50)
ga.evolve(freq_stats=10)

print ga.bestIndividual()
    
```

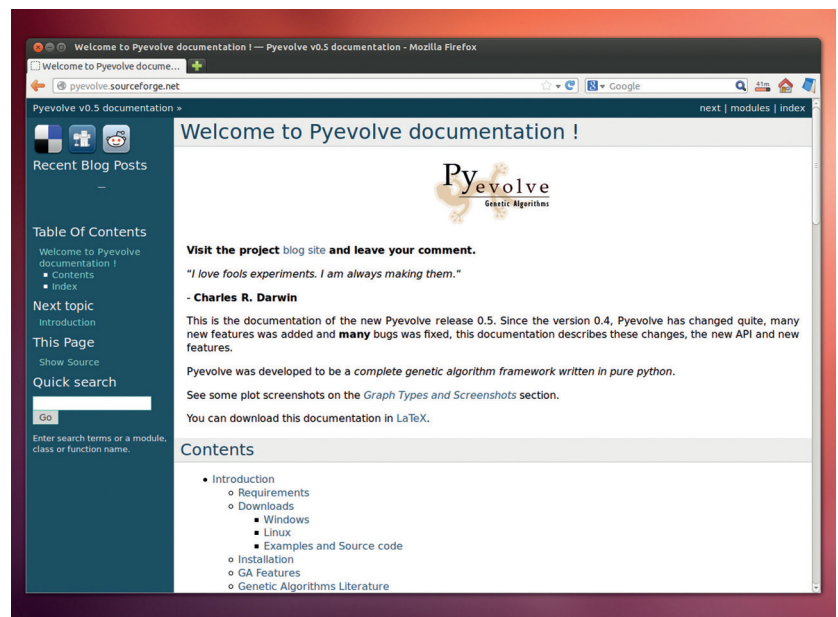
This code will attempt to find the square root of 1000, which is a little unfair since the software only works with integers. If it works correctly, it should find the closest whole number to the square root of 1000.

The variable **genome** holds an instance of **G1DList**. The parameter we gave when creating this is the number of items in the list. Once this variable is created, you can set certain attributes about it. The only thing that has to be there is the call to **evaluator.set()**. This tells the genome what function to use to test the fitness.

The other two things we've set aren't essential for it to work, but make it much more efficient. We've limited the range to between 0 and half of the number we're trying to find the square root of. The smaller we can keep the range, the less work the genetic algorithm will have to go in order to find the square root. Since we're dealing with integers, and this rounds up, it doesn't stop us getting the right answer.

The crossover is the way in which strong pieces of data are combined. There are quite a few options in **pyevolve**, but not all of them work with lists containing just one element.

The final block of code creates a genetic algorithm that takes this genome and evolves it. Again, there are some settings we can tweak to make the environment conducive for getting the right answer. The key value here is the population size. This is the number of organisms we create each cycle by combining the most successful from the previous cycle (and adding some mutations). We found that larger square roots required larger population sizes because the number



If you want to experiment further with genetic algorithms, the **pyevolve** module is well documented at <http://pyevolve.sourceforge.net>.

of values in the range is larger. Therefore, we set the population size to be the number of which we're trying to find the square root. There wasn't any clever calculation that drew us to this setting. We just tried a few different options, and this one seemed to work out the best.

You can also change the number of generations. This is, pretty obviously, the number of times you repeat the selection and recombination. Again, we came across the setting for this after a bit of trial and error. When you run the code, you'll see that it outputs the fitnesses every 10 generations, so you can easily see how quickly it's getting to the solution (or getting stuck at the wrong solution).

Now go and clone some dinosaurs!

That's all there is to it! This code is quite general-purpose, and you should be able to adapt it to your own problems. There is a certain amount of science/art/luck/witchcraft in finding the right values for the environment to produce a good result, and even with a good environment, there's no guarantee of getting the right answer. In fact, if you run this a few times, you'll probably get the wrong answer occasionally.

Genetic algorithms aren't great at every problem, but they can produce surprisingly good results to very complex problems as long as a good fitness function can be created. Essentially, it's a method of searching through a data set that's too large to exhaustively search, and where a simpler search (like binary search) won't work. Incidentally, binary searches do work well for finding square roots, and we've only used genetic algorithms here as an example. 🐳

Ben Everard is the best-selling co-author of the best-selling *Learning Python With Raspberry Pi*.

GOOGLE SCRIPT YOUR GROCERY BUDGET

Forget boring accounting software. Code your own cloud-enabled budgeting script instead.

We're the first to admit we feel uncomfortable with the amount of data that Google is gathering on every aspect of our lives. Many of us on the team are making a concerted effort to move away from some of their services – especially when it comes to location tracking, context searches and personal information (facial recognition, social interaction and profile analysis). But we're also not the types to throw babies out with their bath water. Google has done, and still does, many good things for Free Software, and many of its services are genuinely useful.

And one of the most useful is its scripting engine, known colloquially as Google Apps Script, and there are two reasons why we think it's worth the effort of

using. The first is that the scripts themselves are easy to write. The language is very similar to JavaScript, and while we accept that JavaScript is just as difficult as any other language if you want to become a master, for casual use it can be straightforward, quick and easy. It's widespread enough that many people will have come across it while hacking their own websites, and Google has also done a good job at documenting the various APIs that allow its scripting engine to access and process your data.

The second redemptive excuse we're offering is that you can schedule scripts to run automatically at any time, and unlike your network attached storage box, your Raspberry Pi or low-end-Linux machine, Google's servers rarely suffer outages and come for free.

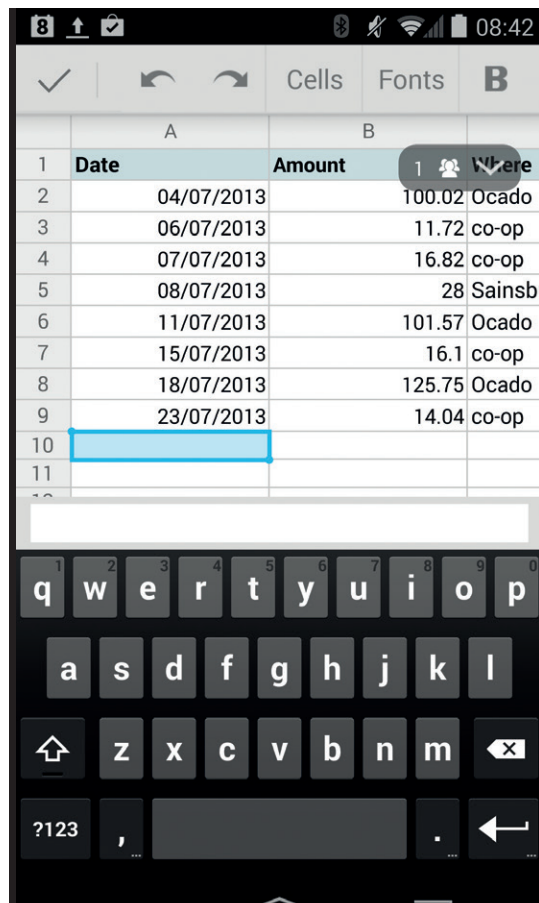
1 PROJECT ORIENTATION

To help illustrate what Google's Apps Script is capable of, and how you might best be able to use it, we're going to create a budgeting system for managing grocery expenditure. The idea is simple; set yourself a budget for each month, and whenever you go to the shops and buy something from your budget, you log the amount. The remaining budget is calculated and is sent via a weekly report telling you how much you've spent and how much you've got left to spend.

Thanks to Google, lots of the complexity is handled for us. To log spending we'll use a Google spreadsheet. These work extremely well from most smartphones, and from any Android device in particular, so it's no hassle adding totals as you go along. As it's a Google spreadsheet, you can also share it with other people, who will then be able to add and manage spending themselves. This is a great solution for a typical household.

We'll construct the spreadsheet in such a way that the data we place into it is easily accessible (through Google's API) to the script we'll write to tie everything together. We'll then write the script to take the important parts of this data, such as the total, the budget, when the cash was spent and how much you'd like to spend, and then write some simple logic around the calculation before outputting a verdict on your spending. The whole script can then be scheduled to run and email one or more people with the results at a specific time.

We feel bad writing this, but you'll need a Google account first. From there, you'll need to click on your



You can keep on top of your finances from your phone, your tablet or your laptop with equal ease.

Google Drive button or go to <http://drive.google.com>. This is Google's shared storage service that is now the central repository for Google Docs too. We imagine most readers will have a Google account already,

so this shouldn't be too much of an issue, but we promise to revisit the subject if enough people would like to see a solution using an open source service, such as OwnCloud, rather than a Google service.

2 CREATING THE SPREADSHEET

From the Google Drive page, click on the large Create button at the top-left and select 'Spreadsheet'. A few moments later, a blank untitled spreadsheet will appear in your browser window. We've called ours simply **lv_groceries** by clicking on the unnamed value at the top. Our solution has two sheets – one for logging day-to-day expenditure and the other for making the various calculations and for holding our budget values. The first sheet is very easy to create, and the best place to start is by giving the first three columns a title each – 'Date', 'Amount' and 'Where'. You might also want to highlight these cells by changing the justification, using a bold font or perhaps a different background colour. This is the page you're going to use to enter your expenditure, sometimes from your laptop and sometimes from your phone or tablet, so a clear layout will help you to be accurate.

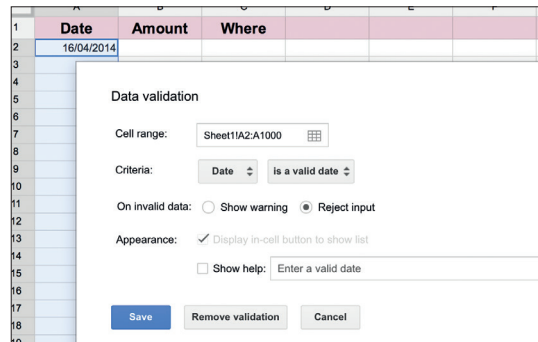
Arrange your data

As you can tell from the three column names, the first column is going to hold the date of the purchase. Google spreadsheets have a data validation feature that does two things for you.

1 It will only allow a valid date to be entered. This stops any rogue data creeping into our scripts, obviating the need to write code to handle the subsequent errors.

2 The convenience of date formatting as you get a pop-up calendar from which you can choose your date. This is much easier to use than typing in a date manually, and avoids any confusion over how a date should be formatted.

To enable data validation, Shift+select every cell in the first column beneath the title, and either right-click and select Data Validation or select the Validation option from the Data menu. A window will appear



showing the cell range you've selected so you can make sure the selection is correct (the first column is 'A', and beneath this you need to select your criteria for validation). Click on the first pop-up menu button and select Date from the short number of input formats that can be validated. Secondly, in the second pop-up menu, make sure that 'Is A Valid Date' is the logical operation automatically set for you. On the following line, you can now choose to either show a warning if a value isn't a date, or reject the input. We went for the first option, as the second can be a little restrictive, especially if you just want to delete a date completely, as this isn't accepted as a standard date.

We don't make any formatting constraints for the other two columns, although theoretically we could for the second column, which is going to hold the value of each expenditure. We don't use the third column, 'Where', but we find this information is useful for monitoring where you spend the most money and for problem solving if you need to cross-check a purchase against a bank statement. This is only the first sheet, however, and we're going to create a lot more functionality in the second sheet, which you can create by clicking on the '+' symbol at the bottom of the page.

Use the 'show help' field to give your friends a little clue about what you want entered.

PRO TIP

We used the new version of Google's spreadsheet for this tutorial – released early March, but it should also work on the older version.

3 DATA PROCESSING

Before moving on to creating the second sheet, we need to give them both names that are going to make moving between them easier. We called our first sheet Receipts, as the values were mostly read off grocery receipts after buying something, and Budget for the second sheet, which is what we're going to explain now. You rename sheets by right-clicking on the tab at the bottom of the current spreadsheet.

We're going to create four columns in the second sheet, all of which are going to be for the convenience of our script rather than for direct use – although they'll also provide a good overview of your annual

and monthly spending. To make sure everything works, we'd highly recommend creating some dummy data back on the first sheet so that when we add some calculations (and eventually the script), they'll have some real numbers to work on and you can judge on the feedback whether everything is working correctly. After you've done that, switch back to the second sheet. The first two columns will hold a reformatted month string and the total expenditure during that month, and we can create both using a formula. Double-click on the A1 cell (the first one on the sheet), and enter the following:

A	B	C	D	E	F	G	H
=query(index(Receipts!A:B), "select year(A)+(month(A)+1)/100,sum(B) where A is not null group by year(A)+(month(A)+1)/100 label year(A)+(month(A)+1)/100 'Month',sum(B) 'Total' ")							
2014.02	11 389		400	February			

Google doesn't provide much error feedback, so you need to make sure all brackets and quotation marks are correct when entering a query.

```
=query(index(Receipts!A:B), "select year(A)+(month(A)+1)/100,sum(B) where A is not null group by year(A)+(month(A)+1)/100 label year(A)+(month(A)+1)/100 'Month',sum(B) 'Total' ")
```

If you're familiar with spreadsheets, and Google's in particular, you'll know that you can access data contained within its sheets using a 'select' statement, just as you would a database. And that's exactly what we're doing here. The reason why we're doing it this way is because it gives us greater flexibility in how we handle the return values. Here's what it does, broken down into chunks of functionality:

```
=query(index(Receipts!A:B),
```

This basically grabs an array of values from both the A and B columns of the 'Receipts' sheet on your spreadsheet. 'Receipts' need to be the same the name of your first sheet. The data from the two columns, A and B, is then passed on to the 'select' statement, for first part of which we'll tackle next:

```
select year(A)+(month(A)+1)/100,sum(B) where A is not null
```

Date formatting

For our eventual script to work without any extra effort, we need the month to be formatted in a specific way: 2014.05, for May 2014, for example. Not only does this help with sorting, but it's easier to process as it appears as a floating point number.

The above command creates that formatting by taking the year and month from the first column (A), and pushing the numeric value for the month through a division by 100 to push the two digits to the right of the decimal place. We're also selecting the corresponding value in the adjacent column.

```
group by year(A)+(month(A)+1)/100 label year(A)+(month(A)+1)/100 'Month',sum(B) 'Total' ")
```

This is the remainder of the query. The **group by** makes sure that the same months are grouped together and with a label that's the same as the calculation – this will be the value itself. And to the right of this we place the total **sum(B)** for all the expenditure from that month, along with two titles for the two columns that are created. If you've created dummy data on the first sheet, you should see an entry in the first column for each month of expenditure, along with a total for that month in column B.

We now need to add three extra columns. In column 6, or 'E' on the sheet, we're going to type the word for each month starting with January in E2 and ending with December in E13. This is a cheat, so we can email the word for the month from the script. In the column to the left of this, 'D', enter the budget you want your spenders to adhere to for each month. We've done this for each month separately in case you wanted more budget for Christmas or birthdays. Finally, to the left of the budget column, we're going to enter a calculation to work out how much money you've got to spend in each month. This is as simple as subtracting the contents of the cell to its left (the total for the month), from the contents of the cell to its right (the overall budget for that month). To do that, double-click in the second cell in column C and type **=SUM(D2-B2)**. You can easily copy and paste the formula so that it changes to reflect the left and right cells of each new position by dragging the blue border surrounding the cell down the column.

PRO TIP
You can show all formulae running on a spreadsheet by selecting the option from the 'View' menu, making problem solving a little easier.

4 WRITING THE CODE

The final step is to write the JavaScript-like code to take the data from our spreadsheet and email it to ourselves. To create a script from the spreadsheet, choose 'Script Editor' from the Tools menu. This opens a new editor window containing a simple template function called 'myFunction'. Here's the first bit of code – place all this code between the curly brackets of the function:

```
var sheet = SpreadsheetApp.openById("1dWqQha3E");
var budget = sheet.getSheetByName("Budget");
```

All this code is doing is opening the spreadsheet we opened earlier. The value within the double quotes is the reference to the spreadsheet, and you need to get this from its URL – it's the value that appears where the ******** is in the following line, but this might depend on the version of sheets that you're using. Either way, the unique identifier for your spreadsheet should be

fairly obvious within your spreadsheet's URL:

```
https://docs.google.com/spreadsheets/d/****/edit#gid=1426067592
```

The next few lines of code are going to make a few assignments to get the current date and implement an offset. We're assuming your budgeting starts in January, but if it doesn't, change the first **startmonth** value to your start month number. You'll also need to offset the word list on the spreadsheet.

```
var startmonth = 0;
var date = new Date();
var month = date.getMonth();
month = (month - startmonth) + 2;
```

We're now going to grab some data from the spreadsheet, first by using the **getRange** method with the **month** variable to specify the row and '3' for the 'Remaining' column. This value will then be appended

to a string we'll use as the subject line in the email, as well as within the body of the email later:

```
var dataRange = budget.getRange(month,3);
var data = dataRange.getValues();
var remaining = parseFloat(data);
remaining = remaining.toFixed(2);
var subject = "Grocery Budget Remaining: " + remaining;
```

We'll cheekily use the same trick to add the text string for the month, taken from the fifth column in the spreadsheet:

```
dataRange = budget.getRange(month,5);
data = dataRange.getValues();
var message = "Month: " + data + "\n";
```

Add to the body of the message by grabbing the total spend value and putting this in along within the message before adding the total budget for the month:

```
dataRange = budget.getRange(month,2);
data = dataRange.getValues();
var total = data;
dataRange = budget.getRange(month,4);
data = dataRange.getValues();
message = message + "Total spend: " + total + "\n\n";
message = message + "Budget: " + data + "\n\n";
```

Now we've got all the variables together, we can write a quick conditional expression that changes the text of the message depending on whether you're under or over budget, leaving the final step to be the sending of the email itself. This is remarkably simple from Google App Script, as you simply call the `sendEmail` method from `MailApp`, using an email address with both the subject and message variables to handle everything else. Obviously, you'll want the

```
function sendGroceryEmails() {
  var sheet = SpreadsheetApp.openById("*****");
  var budget = sheet.getSheetByName("Budget");
  var startmonth = 0;
  var date = new Date();
  var month = date.getMonth();
  month = (month - startmonth) + 2;
  var dataRange = budget.getRange(month,3);
  var data = dataRange.getValues();
  var remaining = parseFloat(data);
  remaining = remaining.toFixed(2);
  var subject = "Grocery Budget Remaining: " + remaining;
  var message;
  dataRange = budget.getRange(month,5);
  data = dataRange.getValues();
  message = "Month: " + data + "\n";
  dataRange = budget.getRange(month,2);
  data = dataRange.getValues();
  var total = data;
  dataRange = budget.getRange(month,4);
  data = dataRange.getValues();
  message = message + "Total spend: " + total + "\n\n";
  message = message + "Budget: " + data + "\n\n";
  if (total < data)
    message = message + "Great work! We're under budget!\n";
  else
    message = message + "Oh no! We've gone over budget!!\n";
  MailApp.sendEmail("graham@linuxvoice.com",
    subject, message);
}
```

Google's script editor has syntax highlighting and an effective debugger, which can help if you find any errors.

email address to be your own, entered carefully, because the nightmare of being blacklisted for mail bombing your budgets from Google's servers isn't worth the potential embarrassment:

```
if (total < data)
  message = message + "Great work! We're under budget!\n";
else
  message = message + "Oh no! We've gone over budget!!\n";
MailApp.sendEmail("graham@linuxvoice.com",subject,
message);
```

LV PRO TIP

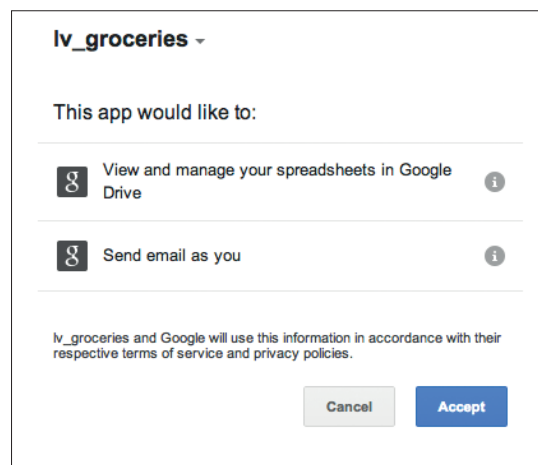
Use the 'Share' button to allow other people to add expenditure, and don't forget to add their emails to the script if you want them to get a notification.

5 RUNNING THE SCRIPT

You're now at the point where you can run the script. To do this, just click on the small black 'Play' button in the script editor toolbar. The first time you run the script, you'll be asked to authorise its access to the spreadsheet and to your email account, which is where the email will appear to originate from. With a bit of luck, a few moments after validation the script will execute and you should see an email like this:

```
Subject: Grocery Budget Remaining: 217.50
Month: April
Total spend: 182.5
Budget: 400
Great work! We're under budget!
```

Congratulations! It works! All that's now left to do is schedule the script to run at a time that makes best sense for you. This is accomplished through Google's trigger system, which can be enabled by going to the script editor, clicking on the 'Resources' menu and selecting 'Current Project's Triggers'. A wide window will include the text 'Click Here To Add One Now', and when you click on that, you can select a 'Time-driven' event to run on a 'Week Timer', 'Every Sunday' at a specific time, or whatever day/time work best for you.



You'll need to give your script permission to access your spreadsheet and to use your email account.

You can even use a trigger to send an email whenever the spreadsheet is opened or changed, giving you the awesome cloud control for your budget, and ultimately, more money to spend on beer. 🍺

Graham Morrison left eBay off this budget spreadsheet to hide the amount he spends on vintage synthesizers.

LINUX VOICE MASTERCLASS

Essential Linux tools explained – this month, say hello to the Inkscape vector editor and the ImageMagick suite.

GET TO KNOW INKSCAPE

Get into scalable vector graphics with the best of its class on Linux.

JOHN LANE

To quote its website, Inkscape is professional-quality vector graphics software that is used by design professionals and hobbyists worldwide to create a wide variety of graphics such as illustrations, icons, logos, diagrams, maps and web graphics. Inkscape uses the W3C open standard SVG (Scalable Vector Graphics) as its native format, and is free and open-source software.

Let's unravel that. Scalable Vector Graphics is an alternative to raster (also called bitmap) graphics, in which images are composed from dots, or pixels. SVG's main advantage is that it is scalable, meaning that enlarging a drawing does not reduce its quality, unlike raster images that become blocky and lose focus and sharpness as their size increases.

A vector image is defined in terms of geometrical elements such as lines, curves and polygons that are themselves based on mathematical expressions that can be rendered at any size without loss of detail.

Open file format

The name SVG also refers to a file format and markup language for describing two-dimensional graphics. The XML-based file format is royalty-free, vendor-neutral and defined as an open standard by the World Wide Web Consortium (W3C). It is Inkscape's primary file format. Inkscape is an open-source alternative to Adobe Illustrator and it's available in the repositories of most distributions.

Learn more

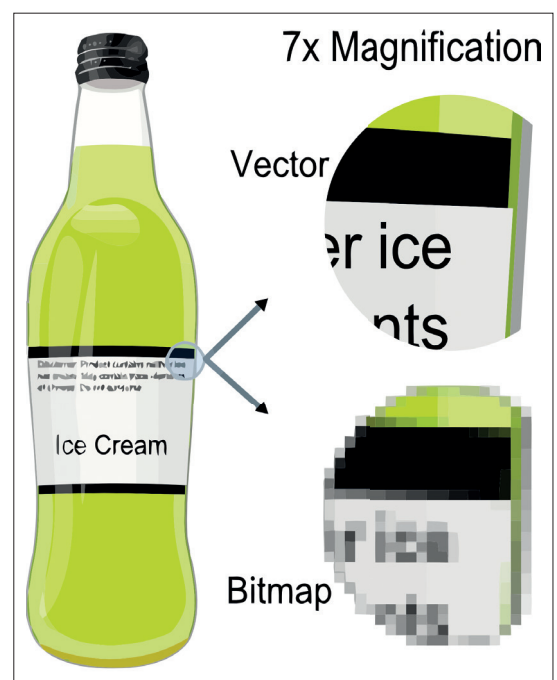
Inkscape's drawing tools create paths: a series of two or more nodes connected by Bézier curves. The Pencil (F6) and Calligraphy (Ctrl+F6) tools allow you to do this with a free hand but the Pen (Shift+F6) is somewhat different because, instead of drawing, you place nodes and use handles to shape the curves.

Handles are the points at the end of control lines that extend from a node. There is one handle for each curve connected to the node. The curve bulges towards the handle before

arriving at the node lined up with the control line (geometrically, the control line is the curve's tangent line at the node).

Intermediate nodes can be smooth, making curves flow together, or cusp, which gives a hard corner. When a node is smooth, its control lines form one straight line.

Getting to grips with Bézier curves will take a little experimentation and practice. If you really want to understand the underlying mathematics, <http://bit.ly/bezcurve> uses interactive examples that explain it well.



source: <http://en.wikipedia.org/wiki/File:VectorBitmapExample.svg>

What enlargement does to an image – Vector vs Bitmap.

When launched, Inkscape presents a single window with the ubiquitous menu and toolbar across its top and the canvas, where you will draw, presented in the centre between further toolbars on the left and right-hand sides as well as the status bar at the bottom. All of these areas are optional – anything that you don't need can be hidden with View > Show/Hide. It's worth keeping the status bar, because it often displays hints that can make the many tools easier to use.

You can zoom the canvas in and out and, when it's zoomed larger than the window, you can move it (or pan) around, either using the scroll bars, dragging with the middle mouse button or, using the keyboard, with Ctrl+arrow keys.

The Toolbox is displayed down the left-hand side of the main window and contains the main drawing and editing tools. The Tool Controls toolbar displays the controls for the selected tool beneath the menu bar. You'll find the usual tools for drawing shapes

and lines, adding text, filling and erasing, but the way they work will feel strange if you're new to this way of drawing. Hover the mouse over each one to reveal a descriptive tooltip and the associated keyboard shortcut. We'll mention the keyboard shortcuts as we tour the application.

In addition to the toolbars, you can display various sub-windows (or modeless dialogs, meaning that you can do other things while they are visible). You can leave the ones that you frequently use on the screen so they are accessible and you can quickly toggle all of those on or off with the F12 key.

Cutting shapes

Let's start by creating some shapes with the rectangle (F4), circles and ellipses (F5), stars and polygons (*) and spirals (F9) tools. They all work in a pretty intuitive way: click on the canvas and drag the mouse to the size of shape that you want. The shape has handles that you can drag to resize or alter its appearance; you can easily round a rectangle's corners or reduce a circle into a segment. Each shape that you lay down is a separate object with its own attributes, like colour.

Use Fill and Stroke (Shift+Control+F) to work with colour. Fill is the colour inside an object and Stroke is the colour of its outline. You can specify colours in various ways including RGB and CMYK and there are tools for gradient-fill and stroke styling that allow control over stroke thickness, style (solid or broken lines) and end-points like arrow-heads.

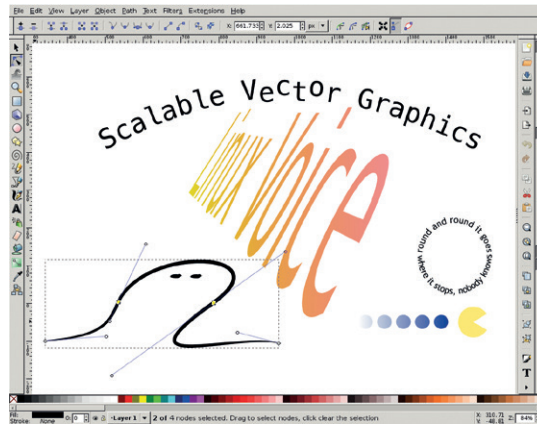
Use the Text (F8) tool to create text objects. These enable you to type text onto the canvas and select fonts, sizes and effects. You can manually kern, which is to adjust the spacing between characters, and move them vertically too. You can also make text flow along a path or within a bounding box.

The pencil and pen are the tools for drawing lines, which Inkscape calls paths. Start by drawing freehand with the pencil (F6) using the same click-and-drag action that worked for shapes. Alternatively, click two points to get a straight line between them. Each path is a separate object.

A Path is actually a series of nodes connected by Bézier curves, and you can see these nodes with the Edit Paths By Nodes (F2) tool. You can click and drag each node to adjust the line or simplify (Ctrl+L) it, reducing the number of nodes. You can edit paths with F2, and it's also possible to convert non-path objects, even text, into paths and gain better control over them.

Once you've laid down some objects, be they shapes, text or paths, you will probably want to tweak, adjust and otherwise edit them. Inkscape comes to the rescue and places a number of tools at your disposal, the most useful probably being Select and Transform (F1 or Space).

Click any object with this tool selected and you'll see it highlighted by a bounding box with handles at its corners that you can drag to resize the shape. If you click again, the handles change to enable you to rotate



Inkscape does text, and not always in straight lines. And straight lines can be reshaped with Bézier handles.

and shear. In each case, holding down Ctrl restricts certain movements (eg to lock on to horizontal or vertical). You can use the Controls toolbar to fine-tune the object's size and position.

All of these transformations can be applied to a group of objects that have been selected together. A Shift+click adds an item to those already selected. You can drag the selected objects to move them or use Align and Distribute (Shift+Ctrl+A) to control their relative positioning.

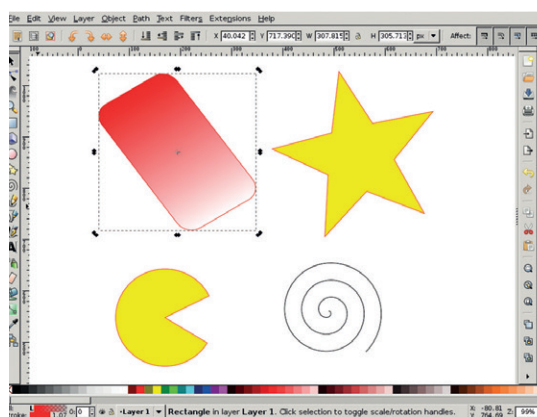
Group objects

As a drawing becomes more complex, it can be useful to logically separate or combine related objects. You can Group (Ctrl+G) and work them as one or you can use layers to separate related objects – imagine the canvas as a stack of transparencies. You can control the stacking, or Z-order, of layers, groups and individual objects, moving them above or below others.

Inkscape is packed with features – so many that you can be perfectly productive without even knowing half of them. But if you ever find yourself wanting for more, there are many extensions available (<http://bit.ly/inkext>) and, if they aren't enough, you can always write your own or even edit your drawing's underlying XML directly. Shift+Ctrl+X opens a live XML editor - any changes made, either via Inkscape's tools or by editing the XML by hand, are reflected in the other. That is the ultimate flexibility, albeit a little more than most users would ever need.

LV PRO TIP
The notifications area displays hints as you edit paths.

LV PRO TIP
There are keyboard shortcuts for just about every action. See <http://inkscape.org/doc/keys.html>.



Inkscape's main window showing some basic shapes.

GET TO KNOW IMAGEMAGICK

Who said you need a GUI to edit images...

Working with images is a task that you'd naturally expect would require a graphical user interface. But there are some tasks that can be completed more efficiently without one – as you'll have seen in our cover feature! We only scratched the surface of ImageMagick there though, so let's go deeper now.

ImageMagick, or just "IM", is a collection of command-line tools and APIs that can be used to work with images in various formats. They enable you to perform many editing operations without a graphical environment. It's been available for many years and should be a straightforward install from your usual package repository.

Because a GUI isn't required, IM is well suited to a server environment and can support web and other applications that receive and process image files. It can be used in scripts to quickly and efficiently perform similar operations on large numbers of images. Tasks like bulk format conversion or thumbnail creation are easy to perform when handled this way.

The API supports many popular programming languages including C, C++, Java, PHP, Perl, Python and Ruby as well as a good number of lesser-known ones. If you need image processing in an application, the APIs most likely have you covered.

The ImageMagick suite comprises 11 command-line tools that accept similar command-line arguments and parameters, so knowledge of one tool is transferable to the others. The best tool to start with is called **convert**.

convert image.jpg image.png

That's probably the simplest ImageMagick command and it does as it says: converts the image from JPEG format into PNG, leaving the original file intact.

ImageMagick supports myriad formats including those you're most likely to need (PNG, JPEG and GIF). You can list the supported formats:

convert -list format

Some formats are real whereas others are pseudo formats prepared via an algorithm or input/output



A montage of resized images. We used **identify**, **convert**, **montage** and **mogrify** to produce this image.

devices. We'll see some of these in action as we look at various commands.

Another useful command is **identify**. It displays information about an image in a one-line summary

identify my_image.png

```
my_image.png PNG 320x240 320x240+0+0 16-bit sRGB
71.4KB 0.000u 0:00.000
```

Add the **-verbose** option to get a very detailed information report.

Geometric argument

You perform transformations while converting by giving appropriate command-line arguments. Resizing is a common task of this kind.

convert image.jpg -resize 600x400 image.png

This gives you a PNG image that is a maximum of 600 pixels wide and 400 pixels high. It preserves the aspect ratio and won't stretch, compress or otherwise distort the image.

You can specify image size, called geometry, in various ways: as a percentage, either applied uniformly to the width and height (50%) or separately (50x75%), but doing so does not preserve aspect ratio.

Explicit values, as we saw, are maxima by default but you can use a caret (^) to reverse this:

convert image.jpg -resize 600x400^ image.png

In this case, the resized image will be at least 600 pixels wide and 400 pixels high and the aspect ratio is preserved. You can use an exclamation mark (!) to force the geometry and ignore the aspect ratio. You can give maximum (>) or minimum (<) sizes and the image will only resize if they are exceeded.

Geometries can also be specified as one dimension. ImageMagick calculates the other dimension so that the image's aspect ratio is maintained. A single dimension is assumed to be the width unless prefixed with an **x**. You can even specify a maximum number of pixels with the **@** operator.

Transmogrification sequence initiated

If you want to overwrite the original file, say because you're resizing but not changing the format, you can use **mogrify**. This is similar to **convert** except that it doesn't require an output file but instead overwrites the original.

mogrify -comment "LinuxVoice example" *.png

You would use **mogrify** for tasks where you want to modify the original file, which will happen unless processing changes the format. It's better to use **mogrify** for simple in-place image processing tasks, especially when batch processing. But **convert** is more suitable for more complex image processing and transformations, because it can perform multiple commands and works well in command-line pipes.

As well as resizing, you can perform image transformations (like crop, chop, rotate, shear and roll) and enhancements (colour, contrast and brightness) or apply various special effects. All of these are controlled using command-line options and they are described in full on the ImageMagick website (<http://bit.ly/imopts>).

Magick Draw

You can use **convert** to create images from scratch. To draw an image you start with a blank canvas that you can create using a pseudo image file format called **xc** (a historical reference to X Window Colour) with an optional colour that defaults to white:

size 100x100 xc: # white (default)

size 100x100 xc:wheat # off-white

size 100x100 xc:none # transparent

As well the plain **xc** canvas, other choices are **gradient** and **plasma**.

When an option requires a colour, you can supply it in a number of ways, either as a colour name or in decimal or hexadecimal notation like this **rgb(255,128,0)** or **#EF9CB0**. You can list colour names with

convert -list color

The documentation describes other supported colour systems like hue-saturation and Lab colour space.

You use drawing primitives to 'draw' on the canvas; there are various options including circles, rectangles, irregular polygons, Bézier curves and text. You use them with the **draw** command-line option:

**convert -size 100x60 xc: -stroke black -fill red -strokewidth 2 **
-draw 'circle 50,30 50,55' circle.png

You can also draw using SVG primitives read from a SVG file; **convert** can render from SVG.

I didn't know you could do that...

ImageMagick has a few extras that you may find useful. There are pseudo formats that acquire images from a scanner, either the default

convert SCANX: image.png

or a specific one:

convert SCAN:'hpaio:/net/scanner?ip=192.168.1.5' image.png

You can embed secrets inside images. This is called Steganography, and hides a smaller image inside a

An alternative...

GraphicsMagick is a fork of ImageMagick that was made in 2002 and continues under active development. It has an MIT licence instead of one based on the Apache 2.0 licence. It cites higher performance and multiprocessor support as advantages along with name-spaced commands equivalent to ImageMagick's but with a **gm** prefix (such as **gm convert...**). GM is used by large sites including www.flickr.com.

Because the fork was over a decade ago and there is no collaboration or code-sharing between the two, divergence is unavoidable and you will notice minor variations between the two. GraphicsMagick recognises fewer colour systems.

All of our examples work in GraphicsMagick – just prefix the command with **gm**.



Draw onto an image with graphics primitives. In this example, we drew some simple shapes, some text and a Bézier curve.

larger one. Just for fun, however: it's hardly secure and it's very brittle (don't try it with a lossy format like JPEG). First, make a message:

convert label:"Linux Voice" message.png

and embed it with **composite**. This overlays images to produce a single composite. Here, we use **rose**, which is one of a few default images that ImageMagick can generate, but you could also use your own. The **-stegano** option does the embedding and starts a given number of pixels into the image:

composite message.png rose: -stegano +27 rose_message.png

Decoding requires knowledge of both the offset and the dimensions of the embedded image (make a note of this with **inquire message.png**).

display -size 66x15+27 stegano:rose_message.png

Beyond the command line

There are some commands that require a graphical environment because they display images – useful if you want to see the fruits of your efforts.

display is a very basic

X.Org application that displays one or more images. A left-click opens a menu with commands for viewing and modifying the image and the right button pops up giving quick access to the main ones. It's by no means a replacement for Gimp or Inkscape but does give you a way to visually complete some tasks.

animate is for viewing image sequences (slideshows). Still basic, but can be useful to see something quickly:

animate -delay 100 *.png

import takes screenshots straight into the format of your choice. It has known bugs, sometimes showing black areas when capturing windows above others.

import screenshot.png

ImageMagick also has its own Magick Scripting Language, or MSL, that you can execute using the **conjure** command. The only other command that we haven't mentioned is **compare** and it shows differences between images. Try it with our **stegano** example:

compare -metric PAE rose: rose_message.png rose_diff.png

John Lane is a technology consultant with a penchant for Linux. He helps new business start-ups make the most of open source.

“As well as resizing, you can perform image transformations and enhancements.”

/DEV/RANDOM/

Final thoughts, musings and reflections



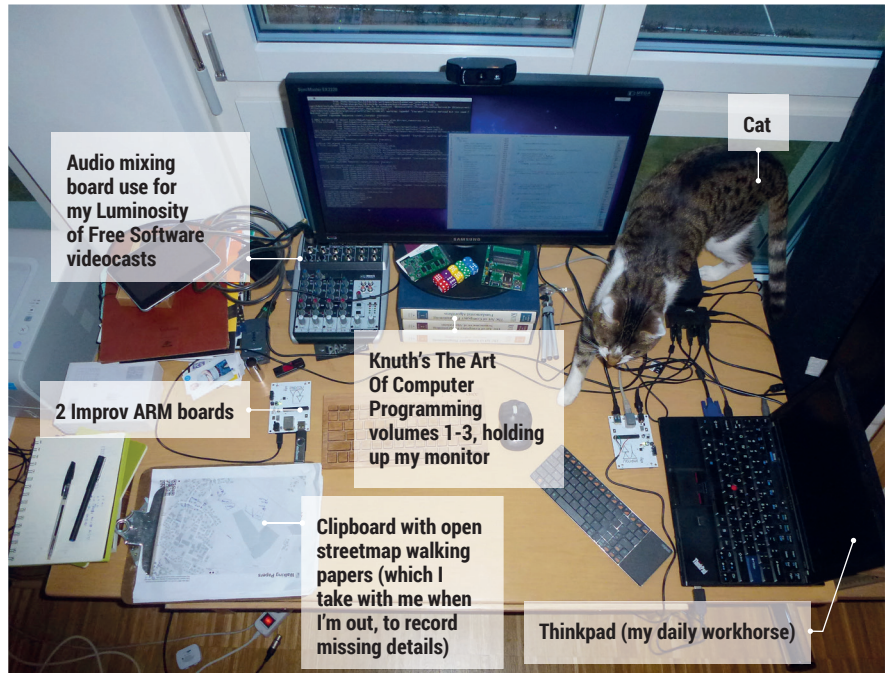
Nick Veitch was the original editor of *Linux Format*, a role he played until he got bored and went to work at Canonical instead. Splitter!

I have a box full of cables under my bed. To be specific, it is full of USB cables. Well, cables and associated adaptors, extensions, converters, powered hubs, unpowered hubs. And chances are if I am looking for a specific cable, I can untangle it from in there somewhere. The rest of the time, the cables sleep, like some Gorgon's nest of failure, a monument to failed ideals and shattered promises.

I understand that the 'U' in USB stands for 'Universal' in terms of the communication bus itself, not the myriad plug/socket combinations required to connect any one thing to any other. But really? I still need old A-B cables for my venerable LaserJet. One of my cameras uses a mini-B plug. I have a camera that uses some off-the-wall variation (I know, non-standard so I can't blame the USB guys for that), I have a music-playing device that I now can't use because the micro-B port on it has broken in some way, probably related to me trying to attach it to the wrong cable.

Is this what the future was meant to be? Are the dying hours of humanity to be spent drunkenly trying to connect your phone charging cable the wrong way up? But lo! The type-C connector approaches (www.usb.org/developers/USB-Futures.pdf). It brings with it, apart from another indistinguishable-when-drunk format change, some good stuff – for a start it doesn't care which way up it goes. And better than that, it makes a nice click when connected.

I don't really care if I have to get a bigger box, these are the things that should have formed part of the standard in the first place – hardware designers, please note that usable-after-a-double-Hendricks is a more precious feature than speed increases, and one for which we are happy to pay extra implementation £££.



My Linux setup **Aaron J Seigo**

The project leader of KDE Plasma welcomes us to his habitat.

Q What version of Linux are you using at the moment?

A KDE Plasma, both Desktop and Active, on various devices (Intel and ARM; laptops, ARM project boards such as the Improv and tablets) on top of OpenSUSE and Mer OS.

Q What was the first Linux setup you ever used?

A Slackware. I found it on a CD in the back of a book while browsing in a bookshop. I was there to buy a book regarding the then-new Solaris, which cost CAD\$80... and it wasn't even that thick. I picked up the book titled *Slackware* and when I saw it claimed to have a full UNIX-like OS on the CD in the back I checked the price of the book and just about choked: it was half the price of the thin little Solaris book. I couldn't believe that I had a UNIX-ish system on a computer that cost a fraction of the price

of any "real" UNIX box. When I figured out it came with source code I fell off my chair with delight.

Q What Free Software/open source can't you live without?

A Kontakt, for email and calendaring; any one of a number of web browsers as so much happens on the web these days; my development toolset including GCC, GDB, Kate and Konsole; dev environments like node.js and workhorse server software like PostgreSQL (of which I am a complete fanboi ;).

Q What do other people love but you can't get on with?

A Artistically devoid pop stars, religion, bitcoin, funnelling personal information through private interests who happily violate our trust and our rights, papayas and the proliferation of desktop environment projects in the last few years.

