

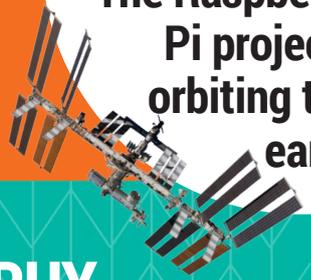


PROUDLY INDEPENDENT SINCE 2013

# LINUXVOICE

**PIs IN SPACE!**

The Raspberry Pi projects orbiting the earth



August 2016

FREE SOFTWARE | FREE SPEECH

www.linuxvoice.com

BUILD A WEBSITE

## HUGO

Generate a static site without faffing around

PUBLISHING

## EBOOKS

Get your words out there into the eternal library

PARANOIA

## STEGANOGRAPHY

Hide data where no-one will look: your holiday photos



# INSIDE THE LINUX KERNEL

- Discover the power at the heart of your Linux box
- Seize full control over your machine's hardware
- Take your first steps as a kernel hacker

**BURSTING WITH AWESOME TUTORIALS**

- GHOST** Blog quickly and cleanly with Free Software
- TAILS** Protect your privacy the easy way – install this distro!
- LIBREOFFICE** Inside the project that's taking FOSS to the masses

OPEN RIGHTS GROUP  
**JIM KILLOCK**

On spying, oversight, and how the government wants to watch us all



RAIN, RAIN, GO AWAY  
**RASPBERRY PI**

Build a virtual weather man with a Pi, some motors and a bit of coloured cardboard



**ELIXIR > RSA ENCRYPTION > KRITA & MORE!**

FREE SOFTWARE | FREE SPEECH

August 2016 £5.99 Printed in the UK

9 772054 1377001

ISSN 2054-3778

480

# FOSSTALK LIVE 2016

**A free evening of live Linux Podcasts**  
Saturday 6 August 2016

**LINUX VOICE**



**Plus Stuart Langridge and Dave MegaSlippers**

**<http://www.fosstalk.com/tickets>**

The Harrison, 28 Harrison Street, Kings Cross, London, WC1H 8JF  
Doors 5pm

# TO THE KERNEL AND BEYOND

## The August issue



### BEN EVERARD

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

**W**hat is Linux? That's the question we're tackling this issue. We already know that it's a kernel – the heart of the operating system – but what does this mean? How is it organised and what does it do? Despite using Linux for well over a decade and a half, I really only had a slight idea about what went on inside the kernel, so when we got Valentine's article in, I was particularly excited to read it. No only is this information interesting, it's also important to know. When running a Linux system – or any OS for that matter – the more you understand, the more power you have. One of the great things about using an open source OS is that we can delve down into the depths and really see what's going on. Armed with this new knowledge, I feel I'm better equipped to deal with any problems that may pop up in the future.

**Ben Everard**  
Editor, Linux Voice

### THE LINUX VOICE TEAM

Editor Ben Everard  
ben@linuxvoice.com

Deputy editor Andrew Gregory  
andrew@linuxvoice.com

Editor in hiding Graham Morrison  
graham@linuxvoice.com

Editor at large Mike Saunders  
mike@linuxvoice.com

Games editor Michel Loubet-Jambert  
michel@linuxvoice.com

Creative director Stacey Black  
stacey@linuxvoice.com

Malign puppetmaster Nick Veitch  
nick@linuxvoice.com

#### Editorial contributors:

Mark Crutch, Juliet Kemp,  
Vincent Mealing, Simon Phipps,  
Les Pounder, Mayank Sharma,  
Amit Saha, Valentine Sinitsyn

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

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

www.linuxvoice.com

## What's hot in LV#029



### ANDREW GREGORY

The landlord of my local pub hates Windows 10, and I've managed to persuade him to make the switch to Linux. I'll take this issue with me next time I go for a pint so he can choose his distro from the group test. **p50**



### GRAHAM MORRISON

GCC is one of those tools that I use all the time, but I don't know too much about what goes on inside it. This month, I've particularly enjoyed the FAQ, where Mike sheds a little light on this ubiquitous compiler. **p32**



### MIKE SAUNDERS

I use public and private keys all the time, but they just seemed like mysterious numbers to me until I read John Lane's tutorial. Now I know exactly what I need to do to keep my data safe when transferring it on the internet. **p84**





# Contents

Here on the plains of the Serengeti, a troupe of monkeys is playing...

## Regulars

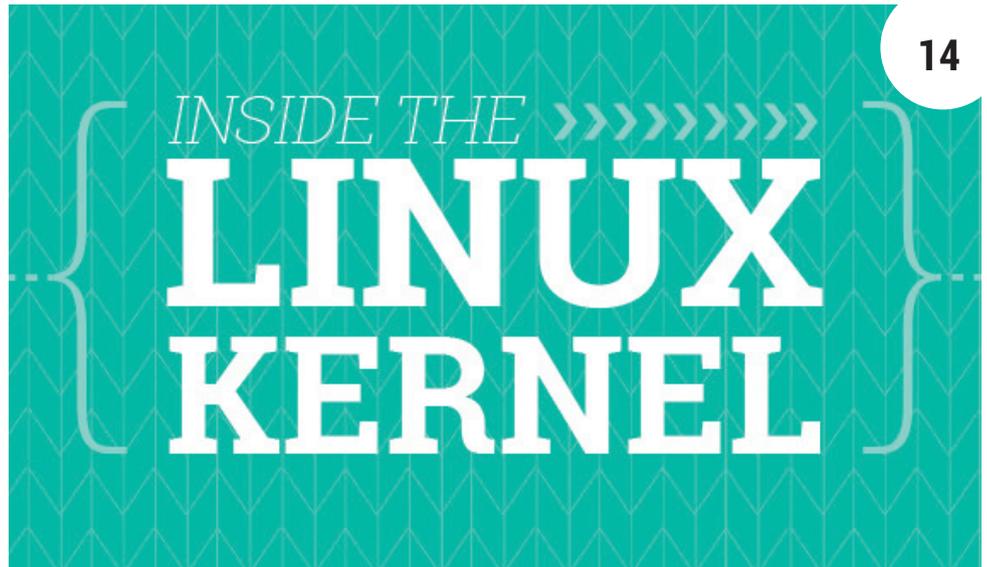
- News** 06  
Android is spreading its mechanical tentacles to the Raspberry Pi 3 and Chrome OS, OwnCloud has forked into a new project, and the Krita project is now €37,000 richer.
- Distrohopper** 08  
This month featuring a gateway distro for Windows users and a destination distro for the security-obsessed.
- Speak your brains** 10  
On the importance of facial foliage in matters Unixy, Linuxy and geeky in general.
- Subscribe!** 12/56  
Never again be the sad-faced Victorian urchin child, face pressed up against the glass of the newsagent having missed out on the last copy of Linux Voice.

- FOSSPicks** 58  
Free range software, organically grown and allowed enough space and light to grow naturally into tasty morsels for us to install on our Linux machines.

- Core Tech** 94  
Look under the hood of Linux and find out what's making that funny noise. This month: processes, threads and Systemtap.

- Geek Desktop** 98  
Nick's looking back at the olden days, which is usually our sign that we should take his Jameson's away from him.

## Cover Feature



What is it that makes Linux Linux? How does your distro speak to your hardware? Why, it's the Linux kernel of course, laid bare on page 14.

## Interview



34

### Jim Killock

The Executive director of the Open Rights Group on spying, privacy and GCHQ.

## Feature

inside **LibreOffice** 22  
The Document Foundation

### Inside LibreOffice

How a humble word processor became the figurehead of resistance to the global proprietary software empire.

## FAQ

- GCC**  
The little compiler that could – without it, everything else would be just a nice idea. Thanks GCC!

## Group Test

- 32** **Beginners' distros** 50  
Help your friends who haven't found Linux yet by recommending one of these beginner OSes.

**SUBSCRIBE ON PAGE 56**



Feature



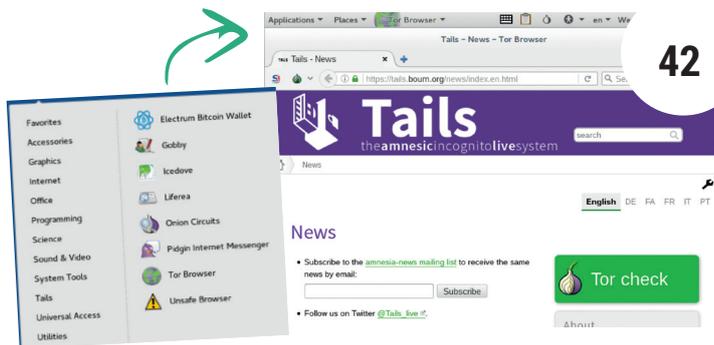
# Pi projects in space

Dip your toe in some of the code running on the ISS right now.

Reviews

# Tails

Love the idea of privacy online, but don't know where to start? Start here – Tails bundles everything you need into one easy to install Linux distribution.



## Krita 3.0

For drawing, animation and aimless doodles, *Krita* is the best Free application by a mile.

## 43 Ghost

If you have a blog and you want to keep it simple, try this elegant, easy to use solution.

## 44 Lumo

Isometric puzzle gaming for the connoisseurs – turn on and pretend it's still 1984.

## 45



## Gaming on Linux

The nights are light, the days are long and full of possibility – so stay in, pull up a chair and refine your reflexes with some fine games.

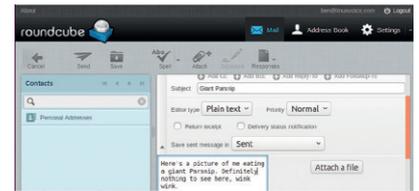


## 46

## Books

When the power fails, you need something to do. Why not read some books? Here are two good ones filled with useful information.

Tutorials



## Steganography

66

Hide data inside innocent-looking image files to keep your secrets safe from all who would do you harm (yes, we're paranoid).



## Hugo

68

Generate static websites without messing about with your own themes, CSS and HTML – *Hugo* does it all for you.

## Publishing with FOSS

74

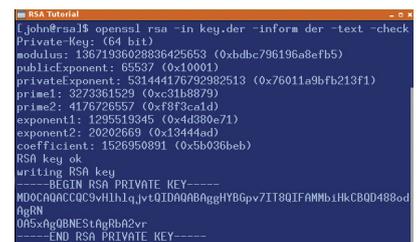
Write, edit and publish in EPUB format, then sit back and watch as the public acclaim/deafening silence floods in.

## Raspberry Pi

78

Turn a stream of data from the interwebs into a physical display, with Python, cardboard and some LEDs.

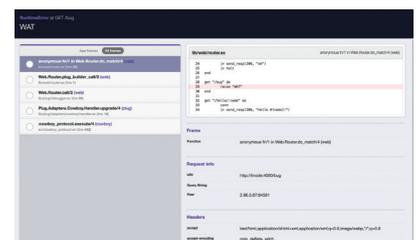
Coding



## RSA encryption

82

Cryptography is hard – find out how hard with this mind-bending advanced tutorial.



## Elixir

88

Develop a web application that's reliable, fault-tolerant and highly available.

# NEWS ANALYSIS

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

Opinion

## Liberating Java

Legal shenanigans in California make their weight felt all around the programming universe.



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

The full Java API is now available licensed under the GPL, and you can use all of or any part of it subject to those terms

The long-running lawsuit Oracle started against Google back in 2010 reached another milestone recently with a jury decision that Google's use of the Java language in Android constitutes fair use. I spent a good part of May in San Francisco and testified in court there as to how, in 2006–7, Sun Microsystems released the Java platform under the GNU General Public Licence.

I testified because of my role as Sun's head of open source at the time. Releasing Java as OpenJDK was an excellent thing to do for software freedom – Richard Stallman even agreed to say so on camera in a Sun promotional video. Among the reasons why:

- The full source code for the Java platform is now available as genuine free software under the GPL. That basic fact is itself a huge benefit.
- When we (the Sun Java and open source teams) released OpenJDK in 2006–7, we were concerned that the open source community might not trust our motives, so as well as using the unaltered GPL, we also applied the licence exception that the GNU Projects's team implementing a free Java used – the Classpath Exception

– unaltered. That means you can implement Java programs in any language. It also means that should you borrow just a few of the class files from OpenJDK, they can be linked with code under other licences.

- In particular from both of those observations, the full Java API is now available licensed under the GPL and you can use all of or any part of it subject to those terms.
- Every source file in OpenJDK includes comprehensive comments that document the specification for each class. The JavaDocs tool is able to harvest those comments to build a specification for the Java platform. Since all those comments are licensed under the GPL, it is possible to build a specification licensed only under the GPL, which places no restrictions on reimplementing.
- From the moment we announced Java would be open source, there were many, many questions in the community. We collated them all and then worked with Sun's business and legal teams to write approved answers to all the questions we could find. The result was a *magnum opus*; a comprehensive and authoritative FAQ that leaves no doubt about Sun's intent. This includes answers to key questions

like “can Sun (now renamed Oracle America) prevent anyone using parts of OpenJDK for purposes of which they don't approve” (the answer is “no”). You'll find the FAQ in the Internet Archive via [wml.me/Java-FAQ](http://wml.me/Java-FAQ) since Oracle deleted it in 2011.

- Most importantly, there is now a large community of developers able to maintain Java regardless of Oracle's strategy. Indeed, Red Hat maintains on an official basis several of the versions that would otherwise have been abandoned.

In the lawsuit, OpenJDK played a key role in proving to the jury that Google should not owe Oracle \$9 billion for using the Java programming language in Android, even before they started using it for Android N. But more relevantly for the rest of us, Java remains a platform that's suitable for open source use because OpenJDK was genuinely liberated and not just a corporate facade.



It's OK to use Java in Android. Phew!

Releasing Java as OpenJDK was an excellent thing to do for software freedom – Richard Stallman even agreed to say so on camera

Systemd • Google • Linux • Android • OwnCloud • Krita • LibreOffice

# CATCHUP

## Summarised: the biggest news stories from the last month

1

### Systemd kills background processes by default

*Systemd* has made some pretty controversial changes to Linux; now it will automatically terminate processes when you log out, so you need to be aware of it if you run a multiplexer like *Screen* or *Tmux*. Some argue that this change makes sense – that things shouldn't carry on running in the background unless you specifically say they should – whereas others argue that it's yet another change that goes against the decade-long Unix philosophy.

2

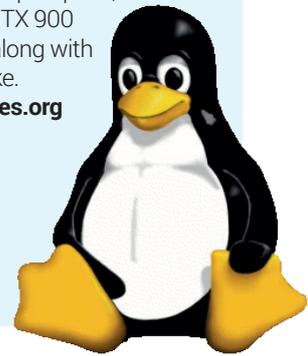
### Google brings Android support to Raspberry Pi

There are already plenty of operating systems to run on your Raspberry Pi: a zillion Linux distributions, NetBSD, and even Windows 10. Now Google is bringing a version of Android to the Pi – and specifically, the Raspberry Pi 3. In Google's Android Open Source Project (AOSP) a new device tree has popped up for the single-board computer, so Pi owners should soon be able to run the vast number of apps that have been released for Android.

3

### Linux kernel 4.6 released

Yes, kernel 4.6 is here with a bunch of improvements: OrangeFS distributed filesystem support, USB 3.1 SuperSpeed, NVIDIA GeForce GTX 900 Maxwell support along with Dell XPS 13 Skylake.  
[www.kernelnewbies.org](http://www.kernelnewbies.org)



4

### Linux: the largest software project on earth

Kernel developer Greg Kroah-Hartman has delivered a presentation explaining that the Linux kernel is the largest development project on the planet. Kernel 4.5 contains a whopping 21 million lines of source code, and in the last year the kernel project received contributions from around 4,000 developers in at least 440 different countries. And to think that in the 1990s, many people said open source software development was simply not sustainable...

5

### Android apps are coming to Chrome OS

Plenty of questions have been raised about the relationship between Google's two mobile operating systems: will Android and Chrome OS eventually merge? Or will the company kill one of them off? Well, now it looks like Chrome OS will soon be able to run Android applications – so more than a million apps and games from the Play Store. This feature is only available in the Chrome OS developer channel right now, but could make its way onto Chromebooks in the future.

6

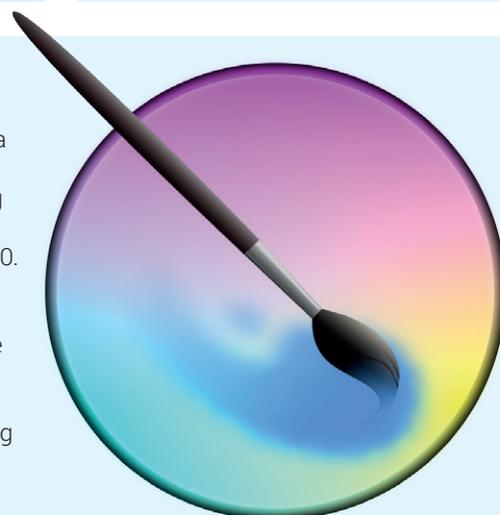
### OwnCloud gets forked: say hello to Nextcloud

Forks in FOSS are often ugly but sometimes necessary. *OwnCloud* founder Frank Karlitschek and a bunch of long-time *OwnCloud* developers have left the project to start a new one: *Nextcloud*. The reasons aren't entirely clear, but it appears that *OwnCloud* Inc. was becoming too vulnerable to the whims of corporate backers. *Nextcloud* aims to be more accountable to the community with a better long-term vision and future.  
[www.nextcloud.com](http://www.nextcloud.com)

7

### Krita achieves €37,000 Kickstarter success

Image editor *Krita* is having a fabulous time. The developers behind it have run a Kickstarter crowdfunding campaign to improve the text and vector tools, setting a target of €30,000. In the end, the final sum of donations was another €7,000 on top of that, thanks to over 1,000 backers from the community. Now the developers will be able to work on a better interface for adding and manipulating text, along with a better workflow for vector objects.



8

### LibreOffice 5.2 beta has been released

Due for release in August, *LibreOffice* 5.2 will bring a bunch of improvements across the suite: a single toolbar mode for *Writer* (ideal for low-resolution displays), new drawing tools, a selection filter in the Cross References dialog, better keyboard shortcuts in *Calc*, and new spreadsheet functions galore. If you want to try a beta release, and report any bugs you find to make the final version really shine, grab it from the website here:  
<http://tinyurl.com/h6bsemy>

# DISTROHOPPER

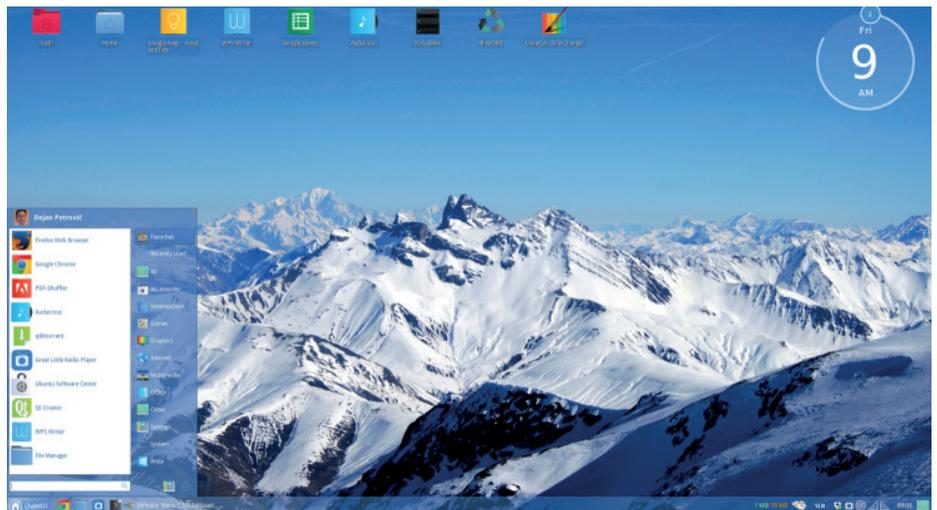
What's hot and happening in the world of Linux distros (and BSD!).

## ChaletOS 16.04

Welcome, newcomers!

**C**halet is a well put together Ubuntu-based distro aimed at capturing Windows users and keeping them on Linux, rather than scaring them off back to their proprietary comfort zone. One of the most notable things it does in this regard is take some liberties with the concepts of free software by including non-free media codecs and the like by default, sparing those trying Linux for the first time the faff of not understanding why their MP3 files don't play.

Otherwise, Chalet is more about aesthetics than anything else, since it targets those looking for a simple and familiar experience, dispelling the myth that Linux is all about headaches and terminals. As such, it offers extremely cohesive and consistent themes, including one that closely resembles Windows 10. To aid in this endeavour, the distribution comes with its own "ChaletOS Style Changer" and at first glance, an experienced Linux user would struggle to tell that it is the Xfce desktop, being used since the themes are so well implemented.



ChaletOS' eye candy, speed and simplicity woo users away from Windows.

It also does well to use Xfce considering its responsiveness without being as bare-bones as other DEs, since speed and decent boot times are other factors that are attractive to Linux newcomers. Similarly, having the stability of an LTS release will mean that this new user won't be put off by any unexpected quirks. A new addition to

the system is "Star Point", an educational/tutorial application to teach newcomers about Linux and how to use it. Though there are many distros out there aiming to grab Windows users, none really hit all the requirements as well as ChaletOS, and it's certainly one to consider when introducing others to Linux.

## Simplicity Linux 16.04

Lightweight, but with functionality.

**S**implicity Linux – a bloat-free distribution derived from Puppy Linux with some differences – has released version 16.04, since Puppy itself is now based on the Ubuntu release cycle. The two main releases of Simplicity – which both use LXDE – are called Mini (previously Netbook) and Desktop, where the former has the bare essentials and the latter has a full-blown desktop experience. Besides those, there's the X release, an experimental sandbox release where the developers try out new ideas. The previous Obsidian (an

even more bare-bones version) and Media (with *XBMC* pre-installed) seem to have been discontinued, with the latter presumably since it's not a lot of effort to install *XBMC* on the Mini build.

Though Simplicity is not ready to be used as a day-to-day desktop, its current aims are to make it more appealing to those crossing over from the world of Windows, and while this is a laudable aim, for now these efforts consist mostly of some UI tweaks and cramming in an absurd amount of software into a small space. This is an impressive feat



The standard Simplicity desktop looks rather good for a lightweight distro.

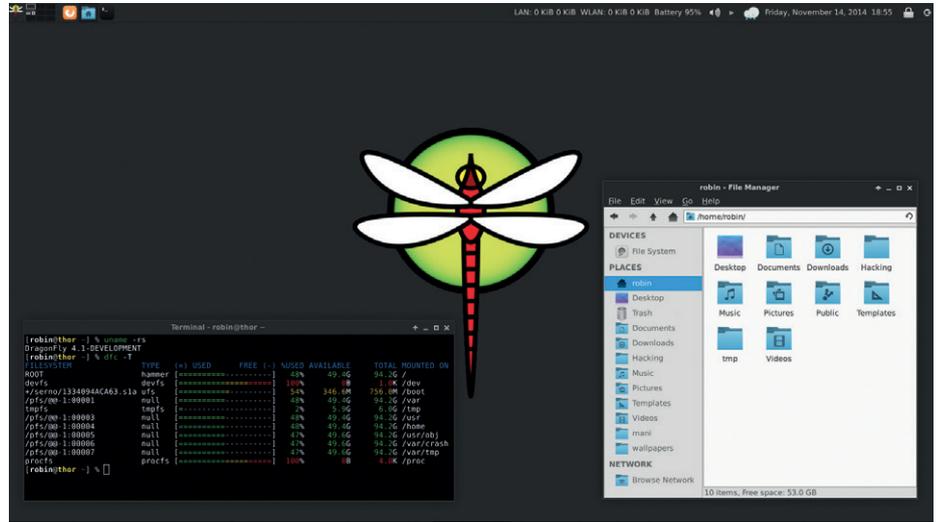
in itself, as it includes the likes of *Gimp*, *Firefox*, *Libreoffice* and *Wine*. Simplicity is one to keep an eye on, as the project seems to be at a crossroads, also having dropped 64-bit support in the two main releases, but heading in interesting directions.

# News from the \*BSD camps

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

There's been a lot of progress across the BSD camps with regard to more modern hardware support, as well as more up-to-date features. The first of this progress comes from DrangonFlyBSD, where Wayland/Weston support has been moving forward to the point where applications can run on the display server. It isn't so straightforward yet and requires some technical knowledge to pull off, however the person responsible for getting it up and running claims that even at this stage, it feels faster than X.org. There have also been advancements with drivers, with the i915 DRM driver being ported over from the Linux 4.3 kernel, replacing the outdated one from the 3.x series, thus improving stability and providing proper support for Intel Skylake CPUs.

Meanwhile, CoreCLR (the open source implementation of the .NET framework) has been ported to NetBSD, though for the time being it requires assemblies to be cross-compiled from Linux. A lot more functionality in other areas is also being added to NetBSD, most notably progress on



DragonFlyBSD can now run the Wayland compositor and relevant applications, though it's not quite ready yet for most desktop users (photo: Distrowatch).

support for audio mixing – before, sound was played on a first-come, first-served basis with only one application playing audio at a time. There is also a wide range of platform improvements being made to amd64, x86 and ARM architectures, as well

as additional hardware support such as for the Freescale i.MX7. These are some pretty positive steps, as NetBSD has been lagging behind the other BSDs for some time now.

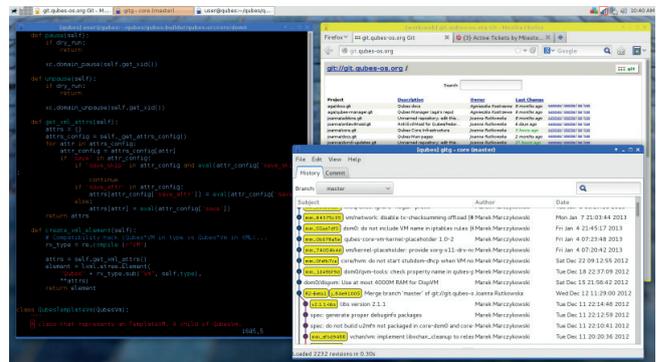
Version 0.9.0 of PC-BSD's Lumina Desktop has also been released, adding support for compositing effects like shadows and transparency, as well as a new plain text editor written in Qt 5. The Fluxbox-based desktop is making good progress leading up to its 1.0.0 release, which is expected to be released at the same time as FreeBSD 11.

## Version 0.9.0 of PC-BSD's Lumina desktop has been released, adding support for effects like shadows and transparency

### Qubes OS 3.1

Qubes is an operating system focused on security, and is hard to describe as being strictly a Linux system by design. The Linux kernel itself never interacts with the hardware directly, with the system instead making use of the Xen hypervisor to do that, using a microkernel design to run multiple instances of differing operating systems. One can see it as running a virtual machine directly on hardware, rather than running a virtual machine through an installed operating system, and this security by compartmentalisation approach is mostly what Qubes is about. The basic premise is to use separate isolated VMs for different tasks – such as one for interactions where credit card details are required and another for general browsing – to avoid cross-contamination.

Despite this seeming like a mess, the creation of the virtual machines is handled through a small and simple VM manager, which allows the allocation of labels such as "work" or "personal" and to assign each window running within that VM coloured borders, making them easily identifiable so as not to create security flaws through human error. The windows are made to be as unintrusive as possible, making the whole experience seamless rather than clunky, and even copy-pasting between VMs is possible through keyboard shortcuts. However, there are many drawbacks, since not only is running multiple VMs like this resource intensive, but Qubes runs only on a limited range of hardware. It's also 64-bit only, though given the memory requirements, this shouldn't be surprising. Actually installing the OS is also challenging, but if all you clear



Qubes running two VMs, one of which is running two windows.

those hurdles, things become straightforward. Aside from the ease of managing VMs, for the ordinary user Qubes functions exactly like a regular Linux desktop, using the familiar KDE's Plasma desktop and Fedora as the template for its VMs, though any Xen-supported OS is possible.

# YOUR LETTERS

Got an idea for the magazine? Or a great discovery? Email us: [letters@linuxvoice.com](mailto:letters@linuxvoice.com)



STAR  
LETTER

## ANGELS WITH HAIRY FACES

Ben – when I look at the photos of you and your three colleagues on LV's Welcome to Linux page, I have just realised that I can improve my Linux skills at a stroke... by growing a beard!

**Godfrey Green, Cardigan, Wales**

**Ben says:** Forget about us: look at Richard Stallman and Alan Cox, or go further back to the Unix days of beard glory and look at Brian Kernighan, Dennis Ritchie and Ken Thompson – magnificent beasts all of them. We've got a long way to go. But we're confident we can make it. Except Mike.

Let's all just ignore the sexism implied in the notion that you need to have a beard to be a leet Linux user...



## BOOTSTRAP

Bootstrap (p78 issue 28), with its 'grid-based layout system,' represents best practice in the first decade of this century and such systems remain very popular with CMS vendors and professional developers who learned their trade in the earlier part of this century.

But Apple, Mozilla and Opera proposed a flow-process approach, which was adopted – albeit in the face of great resistance from traditionalists – in 2011.

With it I have developed two websites each with a single CSS file, which enables content to be displayed on anything from a 19-inch Apple monitor to an Xperia2 smartphone in configurations suitable for each screen. For example, three images in a page display one above each other on the smartphone and side by side on the Apple because the content flows to suit the dimensions of the viewport.

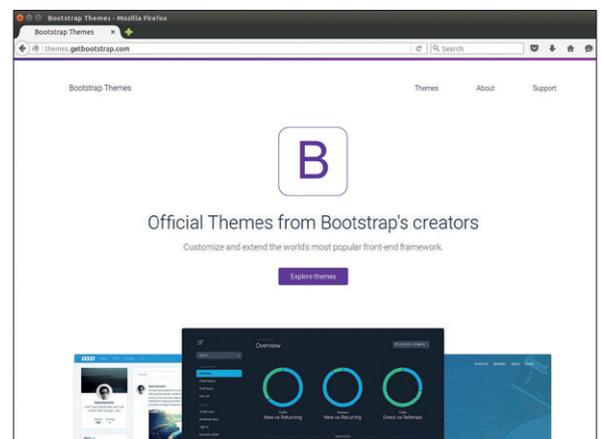
There is no need for multiple CSS files and the recently introduced v unit makes it easy to ensure that images adjust their size to the available viewport.

There is a reasonably comprehensive account of how I have done this at [http://heatholdboys.org.uk/content/HOBA\\_website\\_documentation.pdf](http://heatholdboys.org.uk/content/HOBA_website_documentation.pdf), and it might be worth considering an article on the pros and cons of these different approaches to website design.

**John**

**Ben says:** Thanks John. It seems to me that

because the web is relatively new, there's a million ways of doing things that are all the 'right' way, and things will keep changing depending on whatever technology is in vogue this week. We do know for certain though that, thanks to the web's inherent openness, the barriers to entry will remain low, so anyone can be an expert. And of course, Free Software means we'll always have lots of tools to play around with – such as Hugo, which we explore on page 68.



Whether you prefer grid- or flow-based design, Linux is our favourite platform for web and app design.

## CONVERSION

I just thought I'd drop you a line about my most recent Linux convert. My mum has an aged PC in the corner that she uses to watch YouTube videos. She has loads of precious vinyl records in the garage in tattered plastic bags that never get played because she'd rather have the convenience of playing music through the computer, including bands that seemingly only she can remember. This machine has been staggering along on Windows XP and taking increasingly long to do anything as a decade of malware has slowed it to a crawl. Anyway she finally trusted me enough to put Linux Mint on it, and it now takes a hammering every Friday night when she's had a few and wants to relive the glory nights of Motown. Another happy customer.

**Sarah Barnes, County Durham**



Mint no longer ships with all the media codecs included, but it's still easy to set up a drunken jukebox for family and friends.

## UBLOATU

One of the things that drew me to Linux was how streamlined it is. There's no need to install things you don't need, wasting time and bandwidth. I was an early Ubuntu user, and benefitted from one of the free CDs that Canonical used to post out to help with Ubuntu adoption. It also makes sense to keep a Linux distro small as they are often used in rural areas where bandwidth isn't as good as it is in the developed world. So why on earth is Ubuntu making its next release 2GB? It's going to be as big as Debian at this rate, so what's the point?

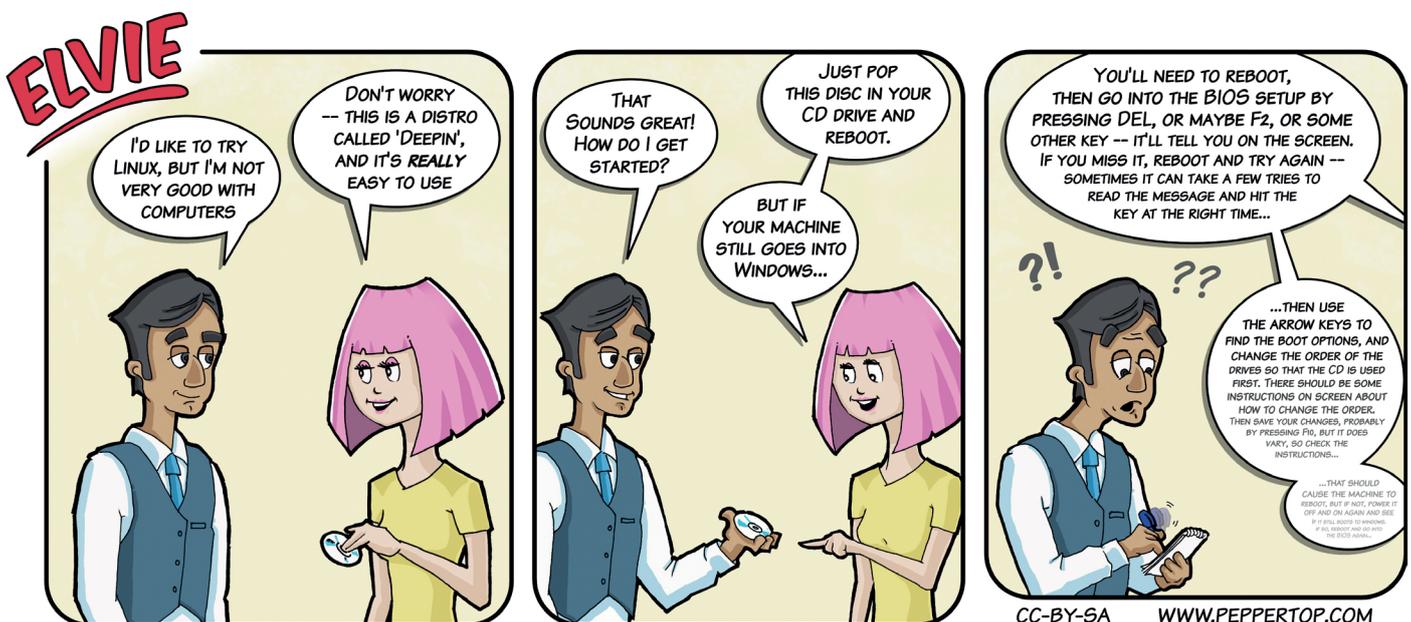
**David Wilkes, Shrewsbury**



**Andrew says:** As projects change, so their aims change. Ubuntu made its first release 12 years ago, so it's not surprising that it's shifted from its original goals. If you want light, you can still get

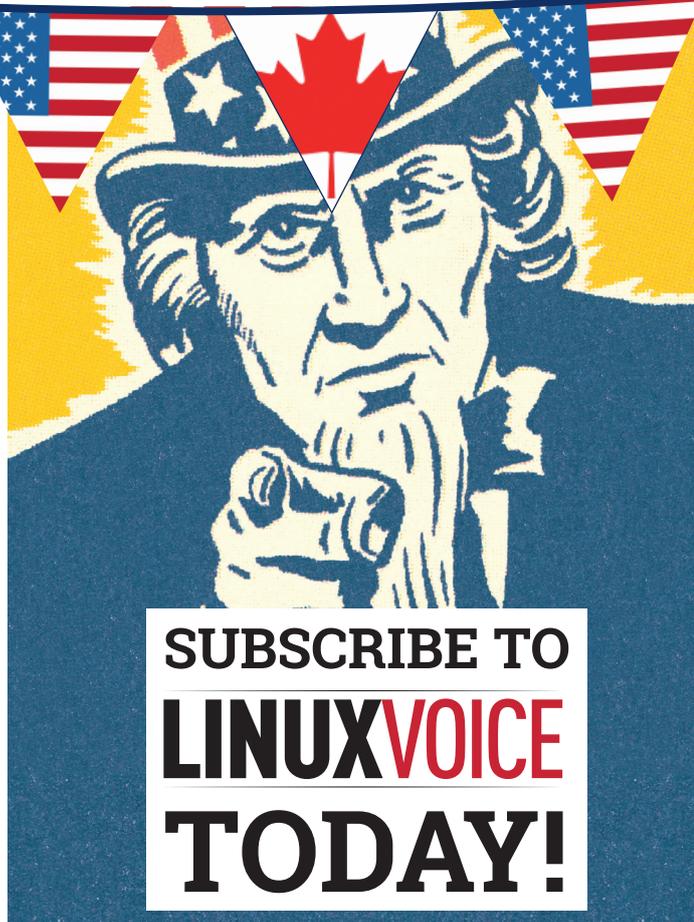
it, but maybe Ubuntu isn't the best choice for you; try Xubuntu instead and go from there. If you want a user-friendly distribution that makes sensible default choices on behalf of the the users, Ubuntu is still a good bet. 

Ubuntu, we still love you. Warty warts and all.



# Subscribe

## shop.linuxvoice.com



Get your regular dose of **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

### US/Canada subs prices

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

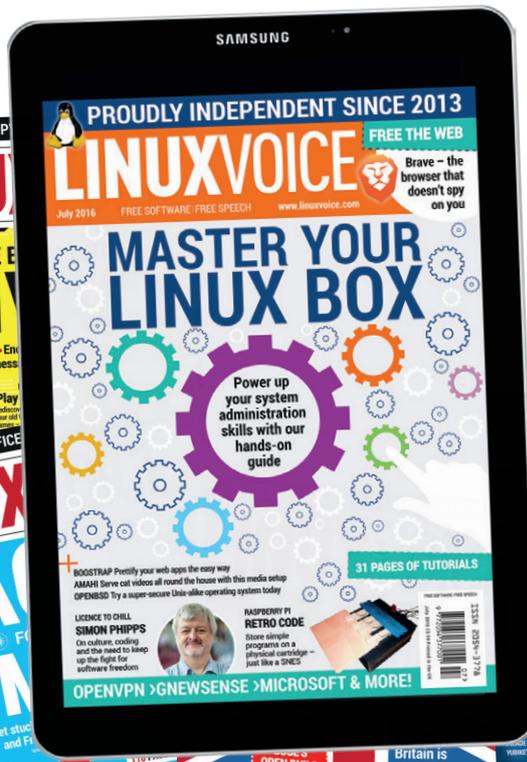
Get many pages of tutorials, features, interviews and reviews every month

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

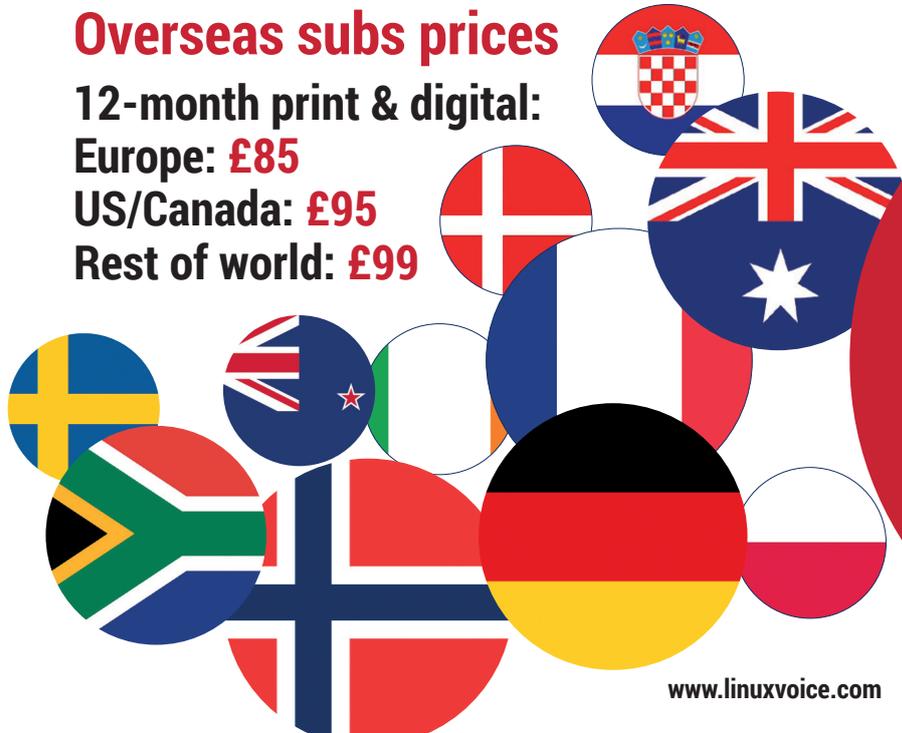
Save money on the shop price and get each issue delivered to your door

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

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



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



DIGITAL  
 SUBSCRIPTION\*  
**ONLY £38**

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



# INSIDE THE LINUX KERNEL

As any sufficiently advanced technology is indistinguishable from magic, **Dr Valentine Sinitsyn** takes his wand and a cloak to show you around the dark corridors of the Linux kernel dungeons.

Whenever you say: "I'm a Linux user," Richard Stallman raises his finger and rectifies: "GNU/Linux." His choice of nomenclature gives the GNU project credit for creating the majority of the user-level components we run every day. *Bash*? It's from GNU. *GCC*? The same. Think of a Linux command, and check its origin – most likely, you use a GNU variant.

What's Linux, then? It's a kernel – "a computer program that constitutes the central core of a computer's operating system". The kernel isn't something you interact with directly as a user. It is what gives the operating system its identity. Running *Bash* on Windows doesn't make it Linux. Even the latest Windows Subsystem for Linux just emulates the Linux kernel on top of NT one. It doesn't matter if you prefer Ubuntu, Fedora, SUSE or Arch: they may feel different on the surface, yet they are all Linuses. FreeBSD or Illumos may look familiar as they are Unix variants. Still, they are separate from Linux and each other, as they build on different kernels.

The kernel's job is often invisible. It schedules processes, manages memory and peripheral devices,

and does many other things. For us, computers "just work": we open a browser then switch to a word processor, then back again. We plug a flash memory card and upload our holiday snaps somewhere. It's intuitive, and we hardly ever think that there are complex algorithms in place to make things run smoothly.

You probably know that Linux runs on many platforms. This includes a whole range from embedded MIPS or ARM processors through commodity x86 and PowerPC to heavyweights like IBM mainframes. Linux supports all this diversity from a single codebase. To make this possible, the kernel carefully separates generic and architecture-specific parts. For example, the process scheduling algorithm is generic. The code to switch process contexts naturally is not, as each architecture has its own set of registers. So, the scheduler chooses the next process to run, then calls the architecture-specific code to apply the change. Generic parts are in C; low-level architecture-specific operations use Assembler.

C is a procedural language. Yet kernel developers employ object-oriented or even a functional style



The Linux kernel carefully separates generic and architecture specific parts

across the codebase. Many kernel-level concepts are represented as C structures with embedded function pointers. Effectively, they act as objects with virtual methods that you find in languages such as C++ or Java. Consider the **filp** (“file pointer”) usage below:

```
long vfs_ioctl(struct file *filp, unsigned
int cmd, unsigned long arg)
{
    int error = -ENOTTY;
    if (filp->f_op->unlocked_ioctl)
        goto out;
    error = filp->f_op->unlocked_
ioctl(filp, cmd, arg);
    if (error == -ENOIOCTLCMD)
        error = -ENOTTY;
out:
    return error;
}
```

Programming in the kernel is still rather different from userspace. Stack space is limited, and you can't do floating point math easily. Any bug in your code affects the whole system. You can't expect the kernel to deliver you SIGSEGV for a NULL pointer error in your code, because you are now the kernel. Many issues you don't even think about in userspace code become your responsibility in the kernel.

Perhaps the most important one is concurrency. In userspace, it's mostly a concern for multithreaded programs. The synchronous code runs line by line. You are the only owner of the data unless you share it with other processes somehow.

That's not the case in the kernel. It's asynchronous by its very nature. Imagine an interrupt occurs while the CPU is executing your code. What happens if a driver decides to update the buffer you were reading? The Linux

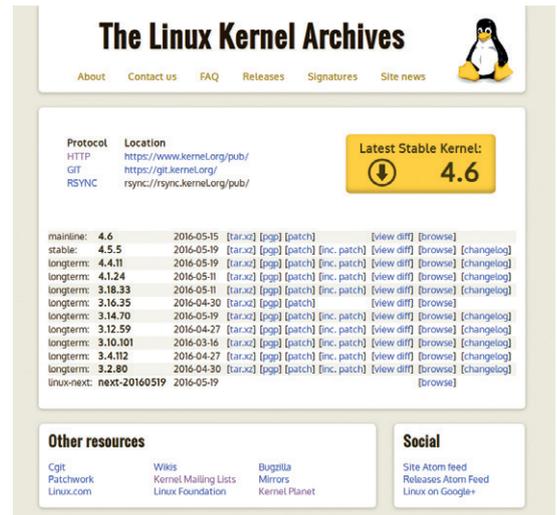
kernel is also preemptive (many other kernels aren't). This means that after the interrupt, the scheduler may decide to return control to some other code, not yours. What if that code accesses the object you were working with? These are only a few examples of what a kernel hacker should keep in mind when writing kernel-level code. Having in-kernel synchronisation the wrong way is disastrous.

## We are the kernel

The Linux kernel comprises various subsystems corresponding to different features it provides. Subsystems are logically separated from each other and typically have dedicated directories in the source tree. However, as Linux is the monolithic kernel, they aren't isolated from each other when the kernel is running.

It's difficult to identify a single “most important” subsystem, but we'd vote for the process scheduler. It selects processes to run and dictates for how long they will own the CPU. Scheduler directly affects both the system performance and responsiveness (or latency). Scheduler sources are found in the **kernel/sched** directory.

Then comes the memory. The kernel does two main things to manage your gigabytes. First, it allocates chunks called pages for the userspace and for itself. When a process wants some more RAM, it issues **brk(2)** or **mmap(2)** system calls to increase the heap (LV018). Second, the kernel configures hardware-dependent mechanisms to provide each process an isolated virtual address space. This includes swapping out pages to disk and reading them back when necessary so that a process may use more memory than physically



The Linux Kernel Archives website is the official kernel homepage. Here you get “vanilla” kernel sources.

present. Memory management lives in **mm** and **arch/.../mm**.

Perhaps the most populated directory in the kernel sources is **drivers/**. Unsurprisingly, this stores device drivers for peripherals that Linux supports (there are quite a few). Having a driver in-tree isn't a requirement, as one can also wrap it as a kernel module. Support for PCI, USB, and other buses also comes through the **drivers** subsystem.

There are many other kernel subsystems, such as networking. Many of them are worth a book on their own. We aren't going to dig that deep today, so turn over the page and let's look into process scheduler operation.

## KERNEL VERSIONING

Like any self-respecting piece of software, the Linux kernel has a version number. You can check what version your current kernel is with the following command:

```
$ uname -r
4.5.4-1-ARCH
```

The kernel version is first three dotted numbers (major, release and patch level). Everything after the dash is what your distribution adds.

Linux 2.6 was released in December 2003. The 2.6 series lasted for more than seven years. In May 2011, Linux decided 3.0.0 would follow then-current 2.6.39. The reasons weren't technical; officially, it was to celebrate the 20th anniversary of Linux. Yet we all know the real reason was that Linus can't count up to 40 (proof: <https://lkml.org/lkml/2011/5/29/204>). Linux 4.0 was released in April 2015. This was also a non-technical bump. Linux Kernel Newbies suggests that “the less you think about it, the better” ([http://kernelnewbies.org/Linux\\_4.0](http://kernelnewbies.org/Linux_4.0)).

You can learn more about Linux kernel versions at <https://www.kernel.org/category/releases.html>.



Linux Kernel Newbies is a site aimed at those who want to start hacking the kernel.

# PROCESSES AND SCHEDULING

Computers are a means to run programs. Here's how the kernel makes it happen.

```

1 [ 0.44%] Tasks: 56, 149 thr, 100 kthr, 1 running
2 [ 0.8%] Load average: 0.12 1.01 0.65
3 [ 0.8%] Uptime: 00:06:05
4 [ 1.2%]
Mem[ 860M/3.75G]
Swp[ 0K/2.00G]

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
973 root        20   0 89244  7556 6000 S  0.0  0.2  0:00.04 /usr/bin/cupsd -l
924 root        20   0 351M 17112 13400 S  0.0  0.4  0:01.23 /usr/bin/NetworkManager --no-daemon
1524 root        20   0 351M 17112 13400 S  0.0  0.4  0:00.00 /usr/bin/NetworkManager --no-daemon
1522 root        20   0 351M 17112 13400 S  0.0  0.4  0:00.00 /usr/bin/NetworkManager --no-daemon
920 root        20   0 330M  8428 6904 S  0.0  0.2  0:00.03 /usr/bin/ModemManager
1587 root        20   0 330M  8428 6904 S  0.0  0.2  0:00.00 /usr/bin/ModemManager
1484 root        20   0 330M  8428 6904 S  0.0  0.2  0:00.00 /usr/bin/ModemManager
918 root        20   0 15372  2688 2320 S  0.0  0.1  0:00.02 /usr/lib/systemd/systemd-logind
915 dbus        20   0 33848  4548 3308 S  0.0  0.1  0:00.71 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
913 avahi        20   0 41316  3616 3232 S  0.0  0.1  0:00.02 avahi-daemon: running [y55hp.local]
917 avahi        20   0 41192  316 0 S  0.0  0.0  0:00.00 avahi-daemon: chroot helper
906 systemd-t  20   0 97984  2320 2108 S  0.0  0.1  0:00.00 /usr/lib/systemd/systemd-timedatd
912 systemd-t  20   0 97984  2320 2108 S  0.0  0.1  0:00.00 /usr/lib/systemd/systemd-timedatd
215 root        20   0 36764  4844 2732 S  0.0  0.1  0:01.01 /usr/lib/systemd/systemd-udev
175 root        20   0 70476  35796 35444 S  0.0  0.9  0:00.37 /usr/lib/systemd/systemd-journald

2 root        20   0 0 0 0 S  0.0  0.0  0:00.00 kthreadd
5781 root        20   0 0 0 0 S  0.0  0.0  0:00.00 kworker/0:0
1806 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1805 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 dm-thin
1804 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1803 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1799 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1798 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 dm_bufio_cache
1791 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 kmflush
1789 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 loop1
1782 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 loop0
1773 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1771 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1770 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1768 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1766 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1765 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset
1763 root        0 -20 0 0 0 S  0.0  0.0  0:00.00 bioset

```

htop displays kernel threads alongside userspace ones. Note they are all *kthreadd* descendants.

A process is an instance of a computer program that is being executed. A typical Linux system hosts many more processes than there are CPU cores available, so arithmetic dictates that the kernel must share computing resources among these processes somehow. Even on a uniprocessor system, everything looks as if processes run in parallel.

Many operating systems permit multiple threads of execution within one process. Linux does this as well, yet it doesn't distinguish between threads and processes at the kernel level. Threads are just processes that have common memory space, file descriptors, and a few other things.

Linux creates processes with so-called "forking". One process (a parent) creates an exact copy of itself (a child), which then runs independently. The `clone()` system call implements this procedure both for processes and for threads.

Cloning (or forking) a process implies copying all of its memory. This doesn't sound lightweight. Moreover, it's often useless, as a typical child immediately issues `execve()` to run another executable. For optimisation, Linux employs the copy-on-write (COW) technique and doesn't copy memory unless necessary.

Most processes run in userspace. When they do a system call like `read()`,

the kernel executes some operations on behalf of the process. There are, however, processes (better said, threads) that run exclusively in the kernel space. They are all *kthreadd* children, and they serve various purposes. For example, *ksoftirqd* helps servicing hardware interrupts; *kswaped* swaps out memory pages; and *jbd2* flushes the filesystem journal to the disk.

## Process descriptors

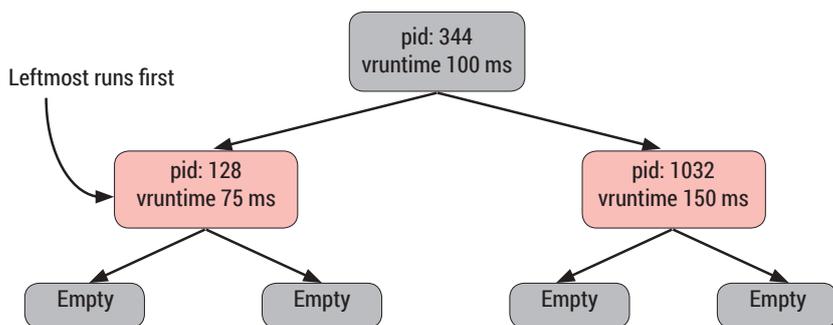
Each process has resources that the kernel needs to track. So, the kernel creates a so-called process descriptor for each process in a system. It stores the process address space, open files and pending signals, along with internal kernel bits like the process state and flags. Technically, the process descriptor is a lengthy C structure

(`struct task_struct`), defined in `linux/sched.h`.

The kernel defines the `current` macro, which returns the descriptor for the currently running process. This is a popular operation, and it must be fast. For example, the x86 architecture uses a per-CPU variable. This makes the descriptor accessible at the known offset from the `%gs` or `%fs` segment. Most other architectures put a pointer to the current process descriptor in the bottom of the kernel stack.

Every process has a userspace stack, typically located at the top of userspace memory. It is used for function calls and local variables. The kernel mode stack serves the same purposes, but it is used when a process runs in kernel mode (say, during a system call). The user mode stack is relatively large (several megabytes) and grows dynamically as needed. The kernel mode stack is small (8KB for x86 and 16KB for x86-64) and static. There are also auxiliary kernel stacks for interrupt and exception handlers, but you can ignore these details for now.

The kernel also distinguishes between process states. A process can be runnable, meaning a scheduler can pick it up for execution. Or, it can be sleeping (blocked), either interruptible or uninterruptible. The difference is that interruptible sleep can be terminated with a signal. When you can't stop a program with Ctrl+C or `kill`, it means the process is in uninterruptible sleep somewhere in the kernel. `ps` marks such processes with `D`.



Red-black tree is a self-balancing data structure that keeps its height low. CFS uses it to find the next process to run quickly.

When it's time to reschedule, the kernel selects a process to run next and does a "context switch". That is, it refills the CPU registers with new values and sets up a new address space. When the switch completes, a new process thinks it just returned from a kernel function call and continues normally.

### Calling for the scheduler

How does the kernel know it's time to reschedule? Every few milliseconds, it calls the `scheduler_tick()` function. It checks if the current process had already been running for too long. If this is the case, a special flag `TIF_NEED_RESCHED` is set on the process descriptor. Later, when the kernel returns control to the userspace after servicing an interrupt or a system call, this flag is examined. If it is set, `schedule()` is called.

`schedule()` is the scheduler's main entry point. Many kernel functions call it explicitly when they need `current` to sleep. A common reason is that a process wants to read data that isn't available at the time.

First, a process descriptor is added to the wait queue. Then, `current` is rescheduled. Later, some other code

```
schedule();
}
__set_current_state(TASK_RUNNING);
remove_wait_queue(&devp->hd_
waitqueue, &wait);
}
```

The `hpet_read()` method implements the `read()` system call for `/dev/hpet`. When an HPET interrupt occurs, its handler updates `hd_irqdata` and calls `wake_up_interruptible(&devp->hd_waitqueue)`;

Many kernels don't reschedule a process when it is in the kernel mode. Linux would do this unless kernel preemption was temporarily disabled with `preempt_disable()`. Kernel preemption occurs when Linux finishes servicing a hardware interrupt and returns control to the kernel mode code. It may also happen when the kernel enables preemption back with `preempt_enable()`.

### Schedulers galore

Linux sports different scheduling algorithms often called classes. Completely Fair Scheduler (CFS) is the default. There are also two real-time classes for higher priority processes and the Earliest Deadline First (EDF)

## OTHER KERNEL RESOURCES

If this month's cover feature has whetted your appetite, and you want to learn more, there are numerous good resources.

For starters, consider the already-mentioned Linux Kernel Newbies, available at <http://kernelnewbies.org>. It's a community website, perhaps best-known for its no-nonsense kernel changelogs. Also have a look at Kernel Planet (<http://planet.kernel.org>), which gathers blogs from many kernel developers. Many new kernel features are being discussed at Linux Weekly News (<http://lwn.net>). Note that these two resources aren't aimed at beginners, though.

There are also several good kernel books. The problem is that most of them are at least five years old. They cover Linux 2.6, which is not much different from now-current 4.x, as you already know. Ultimately, their age is not a big deal, as fundamental parts of the kernel rarely change. But of course there will be some differences between what you read and up-to-date code.

For the first read, consider the *Linux Kernel Development, 3rd Edition* by Robert Love. It's relatively short and high-level enough to build a complete picture without digging too much into details. You may also consider *Professional Linux Kernel Architecture* by Wolfgang Mauerer, which is a bit older, a bit thicker and more in-depth.

also sets some maximum value. Then, it introduces the concept of virtual running time (`vruntime`), which is a process runtime weighted by its priority (see `nice(2)`). On an ideal multitasking CPU, all processes of the same priority



## The kernel creates a so-called process descriptor for each process in a system

wakes up processes on the queue. For a blocked process, this looks like the `schedule()` function has returned. This is how it happens in `/dev/hpet` (a high-precision timer) device driver:

```
static ssize_t
hpet_read(struct file *file, char __user
*buf, size_t count, loff_t * ppos)
{
DECLARE_WAITQUEUE(wait, current);
struct hpet_dev *devp;

devp = file->private_data;
add_wait_queue(&devp->hd_waitqueue,
&wait);
for (;;) {
set_current_state(TASK_
INTERRUPTIBLE);
data = devp->hd_irqdata;
if (data)
break;
```

scheduler for real-time processes with timing constraints. You can set a scheduler class for the process with the `sched_setscheduler()` system call. EDF processes take precedence over real time processes, which run before normal ones.

CFS builds on a simple idea. Imagine "an ideal multitasking CPU" that precisely shares its 100% physical power between running tasks. This CPU would execute two tasks really in parallel, devoting 50% power to each. CFS models this on top of real hardware.

A real CPU can run only one process per core at time. Context switches are costly, so CFS sets a minimum time a process can execute. If a process doesn't yield the CPU for too long, the system becomes unresponsive, so it

would have the same `vruntime`. So, CFS picks a process with a minimum `vruntime` and runs it. Real-time scheduling classes are a bit simpler. `SCHED_FIFO` processes run until they decide to relinquish the CPU. `SCHED_RR` processes are given a timeslice and are scheduled in round-robin fashion. When a process has consumed its timeslice, it is refilled, and the process is added to the end of the queue.

The deadline scheduler (EDF) is a new guy on the block. Introduced with Linux 3.14, it ensures that a process is given a predefined time to run within each accounting period. Say, you may want a process to run for 20ms every 100ms. This is important for time-sensitive applications. The algorithm chooses the process with the earliest deadline, hence the name.



# MEMORY MANAGER

Take care of your memory – all 640KB of it.

Most architectures today provide a Memory Management Unit (MMU). You may think of the MMU as a chip that translates memory addresses and enforces memory protection. This paves the way for process-isolated virtual address spaces. MMU operates on pages. Typically, a page is 4KB, although it's possible to create huge pages spanning megabytes or even gigabytes. The idea is to increase granularity to minimise costs. If the MMU wants just one bit of metadata per memory byte, this means 12.5% overhead with a byte-level granularity, but only 0.003% with 4K pages.

Linux also needs to track memory metadata, and it also does it with page-level granularity. Each physical page has an associated **struct page** structure in the kernel. It tells if the page is free or not, or if it is dirty (that is, has some data not on the disk yet). In short, **struct page** is like **struct task\_struct** in that it stores all the information the kernel needs to manage a memory page.

Having a structure per single page may seem like a waste. If **struct page** is about 100 bytes (actually less), and

page size is 4KB, it takes a few percent of the memory available. This doesn't seem to be a prohibitive price to pay for a useful metadata **struct page**.

Pages are organised into zones. The reason for this is that not all pages are born equal. Old ISA devices, for instance, can only use memory in the lowest 16MB. 32-bit devices can't see memory above 4GB. Naturally, one doesn't allocate "normal" memory there unless absolutely necessary. Zones are what facilitates making such decisions in the kernel.

A 64-bit PC has most of its memory in the Normal and DMA32 zones. A 32-bit system is likely to have an additional HighMem zone. The kernel address space is typically 1GB on 32-bit hosts, and HighMem includes memory above that. On 64-bit systems, the address space is a huge 64TB, and everything fits in Normal.

### Meet your buddies

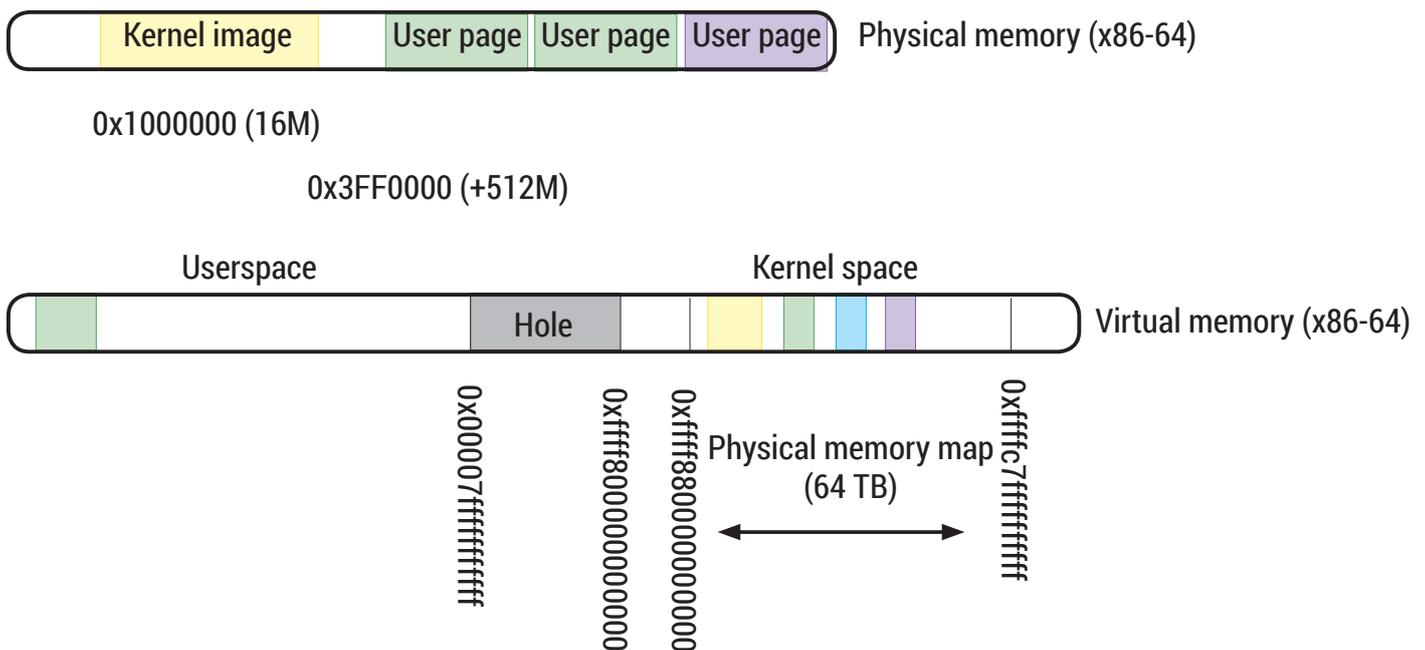
All C programmers know **malloc()**. You tell this function how many bytes of memory you want, and it finds a big enough chunk for you. **malloc()** comes

from the standard C library, which is a user-level thing. So, the kernel needs its own memory allocator.

At the lowest level in the kernel lies the page allocator. This shouldn't come as a surprise, as the page is the basic memory management unit. Yet a page is relatively big chunk of data, and no one goes to ask for 4KB if all they need is 128 bytes. So, there are other mechanisms built on top of the page allocator as well.

A memory allocator often has two main goals. First, it needs to keep memory fragmentation low in the long run. Otherwise, it wouldn't be possible to allocate a large chunk of memory, even if it is formally available. Imagine memory where all even pages are used, and all odd pages are free. One can't allocate a block larger than one page, even if half the memory is empty.

To prevent this situation, you can allocate and free up memory in large blocks. And here comes another goal: the allocator shouldn't waste too much memory. Linux addresses both points with a variation of algorithm known as a buddy system. It allocates pages in



The kernel builds virtual address spaces for userspace processes and for itself. We discussed it in LV018, now online.

power-of-two chunks: 1, 2, 4, 8 and so on. Each size has a separate free list, or list of pages (**struct page**), which are available. If someone asks for 3 pages, the allocator looks up the 4-page list. If it's empty, the allocator splits an 8-page block in two halves called "buddies" (hence the name). Then it returns the first half to the caller and adds the second to the 4-page list. Later, when the first buddy is freed, the allocator detects it and promotes a combined 8-page block back to the original list.

**cat /proc/buddyinfo** tells how many free blocks are currently available in each of your system's zones. The

corresponding kernel functions carry the **kmem\_cache\_** prefix. An object's memory comes from page-sized blocks. They are called "slabs", and are allocated via a buddy system. When the request comes, and there are no free objects in a cache, a new page is allocated and initialised. The kernel also aligns objects in a slab the way they don't step on each other's toe in CPU hardware caches.

SLAB is a classic implementation of this idea that first appeared in Solaris. SLOB is its low-footprint variant. It works best for smaller systems but isn't that good on larger ones. SLUB

```
ti = page_address(alloc_kmem_pages_node(node, THREADINFO_GFP, THREAD_SIZE_ORDER));
if (!ti)
return NULL;
...
}
```

Here, **struct task\_struct** comes from a slab; **struct thread\_info** comes from the buddy system. Finally, there is the **kmalloc()** function to allocate arbitrary-sized buffers. To facilitate it, the kernel has several caches (**kmalloc-N**) for power-of-two sized blocks smaller than two pages. For larger blocks, pages are allocated via a buddy system.



## When the kernel forks a process, it allocates a new process descriptor and kernel stack space

number of pages in the block (also called an "order") increases from left to right: 0, 1, 2 and so on.

### SLAB, SLOB, and SLUB

The kernel also provides a way to allocate specific objects, such as **struct task\_struct**. They are sometimes small and short-lived, which warrants them special treatment. To this end, the kernel has three related algorithms: SLAB, SLUB and SLOB.

The idea is to keep "ready to consume" objects and return them quickly without allocating any memory. This also serves as a cache, hence

introduced some optimisations to original SLAB, which makes it the default in recent Linux kernels.

When the kernel forks a process, it allocates a new process descriptor and kernel stack space. This occurs in **dup\_task\_struct()**:

```
static struct task_struct *dup_task_struct(struct task_struct *orig)
{
struct task_struct *tsk;
struct thread_info *ti;
tsk = kmem_cache_alloc_node(task_struct_cache, GFP_KERNEL, node);
if (!tsk)
return NULL;
```

So far, we've spoken of physical memory. However, programs and the kernel itself run in a virtual address space, so the same address may refer to different memory locations in userspace processes (the kernel has a single virtual address space). When MMU detects the program tries to touch an address which is not mapped, it calls the page fault handler in the kernel. The latter decides whether the fault should result in system panic (if it occurred within the kernel), or just **SIGSEGV**. Sometimes, the fault is not a problem at all, as the page in question has been just swapped out.

## STEP BY STEP: PEEK INTO KERNEL CACHES WITH SLABINFO

### BUILD THE TOOL

**slabinfo** comes together with the Linux kernel. Grab the latest version from [www.kernel.org](http://www.kernel.org) (it's big), untar, **cd** into **linux-X.Y.Z/tools/vm**, then do **make slabinfo**. You'll need **build-essentials** or the similar package from your Linux distribution.

```
11: 747714
12: /usr/bin/path.o
13: /usr/bin/path.o
14: /usr/bin/path.o
15: /usr/bin/path.o
16: /usr/bin/path.o
17: /usr/bin/path.o
18: /usr/bin/path.o
19: /usr/bin/path.o
20: /usr/bin/path.o
21: /usr/bin/path.o
22: /usr/bin/path.o
23: /usr/bin/path.o
24: /usr/bin/path.o
25: /usr/bin/path.o
26: /usr/bin/path.o
27: /usr/bin/path.o
28: /usr/bin/path.o
29: /usr/bin/path.o
30: /usr/bin/path.o
31: /usr/bin/path.o
32: /usr/bin/path.o
33: /usr/bin/path.o
34: /usr/bin/path.o
35: /usr/bin/path.o
36: /usr/bin/path.o
37: /usr/bin/path.o
38: /usr/bin/path.o
39: /usr/bin/path.o
40: /usr/bin/path.o
41: /usr/bin/path.o
42: /usr/bin/path.o
43: /usr/bin/path.o
44: /usr/bin/path.o
45: /usr/bin/path.o
46: /usr/bin/path.o
47: /usr/bin/path.o
48: /usr/bin/path.o
49: /usr/bin/path.o
50: /usr/bin/path.o
51: /usr/bin/path.o
52: /usr/bin/path.o
53: /usr/bin/path.o
54: /usr/bin/path.o
55: /usr/bin/path.o
56: /usr/bin/path.o
57: /usr/bin/path.o
58: /usr/bin/path.o
59: /usr/bin/path.o
60: /usr/bin/path.o
61: /usr/bin/path.o
62: /usr/bin/path.o
63: /usr/bin/path.o
64: /usr/bin/path.o
65: /usr/bin/path.o
66: /usr/bin/path.o
67: /usr/bin/path.o
68: /usr/bin/path.o
69: /usr/bin/path.o
70: /usr/bin/path.o
71: /usr/bin/path.o
72: /usr/bin/path.o
73: /usr/bin/path.o
74: /usr/bin/path.o
75: /usr/bin/path.o
76: /usr/bin/path.o
77: /usr/bin/path.o
78: /usr/bin/path.o
79: /usr/bin/path.o
80: /usr/bin/path.o
81: /usr/bin/path.o
82: /usr/bin/path.o
83: /usr/bin/path.o
84: /usr/bin/path.o
85: /usr/bin/path.o
86: /usr/bin/path.o
87: /usr/bin/path.o
88: /usr/bin/path.o
89: /usr/bin/path.o
90: /usr/bin/path.o
91: /usr/bin/path.o
92: /usr/bin/path.o
93: /usr/bin/path.o
94: /usr/bin/path.o
95: /usr/bin/path.o
96: /usr/bin/path.o
97: /usr/bin/path.o
98: /usr/bin/path.o
99: /usr/bin/path.o
100: /usr/bin/path.o
```

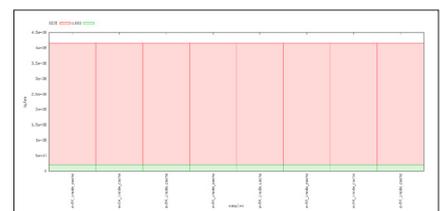
### GET SOME STATS

Now run the tool as **./slabinfo** (root permissions are required). It will produce a lengthy table containing all slab caches found in your system along with some stats. As an exercise, try to find the **kmalloc** or **task\_struct** caches. How many objects are in them? What's the object size?

```
1: 00000000 510 368 229.3K 7/7/21 22 1 25 83 *
2: 00000000 114 384 278.3K 5/8/20 21 1 8 98 *A
3: 00000012 3739 512 2.8M 92/21/31 32 2 17 94 *
4: 00000048 125 648 81.9K 8/8/5 25 2 8 97 *
5: 00000096 144 896 131.8K 8/8/4 36 3 8 98 *A
6: 000000C8 612 968 589.9K 8/8/5 36 3 8 99 *A
7: 00000124 1616 1024 1.7M 39/16/15 32 3 29 93 *
8: 00000180 812 1008 131.8K 8/8/4 28 3 8 93 *A
9: 00000240 808 2848 1.9M 45/15/14 16 3 29 91 *
10: 00000280 280 4896 1.4M 24/10/8 28 3 8 88 *
11: 00000304 15804 64 1.1M 241/77/34 64 8 26 98 *
12: 00000320 22633 192 4.5M 1869/114/38 21 8 18 94 *
13: 00000380 9485 72 819.2K 151/37/49 51 8 18 83
14: 00000400 136 888 131.8K 8/8/4 36 3 8 95 *A
15: 00000440 312 184 32.7K 8/8/8 39 8 8 99 *
16: 00000480 149552 192 27.4M 6674/89/19 21 8 98 *
17: 00000500 32 512 16.3K 8/8/1 35 2 8 108 *d
18: 00000520 38 1856 32.7K 8/8/1 38 3 8 96 *A
19: 00000540 15 2808 32.7K 8/8/1 15 3 8 95 *A
20: 00000560 128 128 16.3K 518/34/95 18 8 22 95 *
21: 00000580 51973 48 2.3M 8/8/31 64 8 8 100 *
22: 00000600 1904 64 126.9K 128/8/2 28 8 8 98 *
23: 00000620 3416 144 489.7K 3224/342/11 38 3 18 92 *
24: 00000640 95598 1856 189.2M 3224/342/11 38 3 18 92 *
```

### PLOT SOME GRAPHS

A picture is worth a thousand words. Build some totals with **while [ 1 ]; do ./slabinfo -X >> stats; sleep 1; done** (wait a dozen of seconds). Then run **bash ./slabinfo-gnuplot.sh stats** to build graphs. Green is slab size. Red is loss (how much space the slab wastes).



## SYSTEM CALLS

The kernel is here so you can focus on your job, not managing the hardware.

```

Functions calling this function: kcalloc
File      Function      Line
0 lphase.c ia_hw_type     2794
1 lanat.c  lanat_init_one 2563
2 nicstar.c ns_init_card   373
3 nicstar.c get_scq        865
4 nicstar.c get_scq        874
5 solos-pci.c fpga_probe    1294
6 suni.c   suni_init      372
7 uP98402.c uP98402_start 213
8 zatm.c   do_tx          678
9 zatm.c   start_tx       1081
a zatm.c   zatm_open     1481
b zatm.c   zatm_init_one 1594
c cfagl2864b.c cfagl2864b_init 358
d class.c class_compat_register 514
e devres.c devres_open_group 538
f dma-mapping.c dma_common_contiguous_remap 311
g firmware_class.c fw_realloc_buffer 773
h map.c   kobj_map_init 137
i platform.c platform_device_register_full 511
j core.c  _opp_set_availability 1288
k wakeup.c wakeup_source_create 93
l regcache-lzo.c regcache_lzo_prepare 48
m regcache-lzo.c regcache_lzo_compress_cache_block 78
n regcache-lzo.c regcache_lzo_decompress_cache_block 96
o regcache-rbtree.c regcache_rbtree_init 285
p regcache.c regcache_hw_init 62
q regmap-debugfs.c regmap_name_read_file 46

* Lines 156-183 of 4996, 4814 more - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find Functions called by this function:
Find Functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:

```

**cscope** is a venerable source code navigation tool. It's old enough to remembers the days of PDP-11, and it's still useful enough to be relevant in 2016. Good job, **cscope**!

Up until now, we saw the kernel as something transparent, sitting between our programs and the hardware, and providing some useful abstractions to the former. From time to time, the programs want to request some service from the kernel explicitly. Without that, it won't be possible to read and write files, or exchange data over a network.

For security reasons, the kernel is isolated from the rest of the system. This happens at hardware level (think MMU again), and this means you can't

interface are carefully chosen, and arguments you pass are thoroughly validated. So, rare-yet-possible bugs aside, system calls provide a safe way to call into kernel functions.

### The role of libc

As a programmer, you almost never deal with system calls directly. The standard C library (nine out of ten times, *Glibc*) wraps them to provide a standard API such as POSIX. Imagine you want to get some information about the system you run on. POSIX

wraps the system call of the same name. This doesn't have to be the case in all Unixes, though. **uname(3)** is POSIX standard, **uname(2)** is, in theory, Linux-specific.

Not all C library functions are simple wrappers, of course. Linux defines a few hundred system calls, and many of them are rather low-level. Others multiplex several userspace visible functions. There are system calls that are specific to hardware architecture. Often, they carry the **arch\_** prefix. There are system calls that have no C library wrappers, such as **futex(2)** or native asynchronous I/O family (LV026). No wrappers are a fat hint that you don't want to use these system calls in your programs directly. Yet it is still possible to issue them the indirect way via **syscall()**, as the C library does.

In fact, Linux distinguishes system calls not by names, but by numbers. **SYS\_uname** is a macro that expands to 63 on 64-bit x86 machines. This number is also architecture-dependent: it's 122 for 32-bit x86, for instance. Internally, the kernel recognises this system call as "new uname". There are also "old uname" and even "old old uname." It's not uncommon to have such a convoluted history in the world of system calls. From time to time,



You can think of a system call as an interrupt that you generate to draw the kernel's attention

simply call a kernel function from userspace. Instead, one uses a system call interface as a well-defined gateway to the kernel.

This isn't the only means to switch to the kernel mode: hardware interrupts incur this as well, but they are outside the programmer's control. You may think of a system call as an interrupt that you, not hardware, generate to draw the kernel's attention. Services available through the system call

defines the **uname(3)** function and associated structure definitions for this purpose. It's available on many Unixes, and its Linux implementation may be as simple as this:

```

int uname(struct utsname *uts)
{
    return syscall(SYS_uname, uts);
}

```

This code comes from *Musl* (<https://www.musl-libc.org>), an alternative C library for Linux. As you see, it simply

kernel developers make incompatible changes to data structures but leave the semantics unchanged. It seems reasonable to declare an earlier implementation as "old". Luckily, a standard C library hides all these nuances from mere mortals.

The exact method of making a system call, that is, switching to kernel mode, is also architecture-specific. On x86 computers, this used to be a software interrupt, **int \$0x80**. You

may still find it in older manuals and tutorials. This method is still supported, yet rarely used. The reason is that interrupts are quite costly. As programs do system calls very often, this may hurt the performance. So newer CPUs introduced dedicated instructions for fast switching to the kernel mode and returning the control back. Intel processors implement **sysenter/sysexit** instructions and AMD chips have **syscall/sysret**. There are also some peculiarities related to 64-bit and 32-bit modes. How does the poor C library account for all these specifics when issuing system calls?

### Nuts and bolts

In short, it doesn't. Otherwise, it won't be possible to introduce a new switching method in the kernel without touching every C library implementation in the world. This doesn't scale well. Instead, the kernel maps a "vDSO" (virtual dynamic shared object) in every process address space:

```
$ cat /proc/self/maps
```

```
...
7ffd86721000-7ffd86742000 rw-p
00000000 00:00 0 [stack]
7ffd867e2000-7ffd867e4000 r-xp
00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp
00000000 00:00 0 [vsyscall]
```

A vDSO is much like a shared library which is always present. You may think it's a waste of RAM. Again, it's not, because Linux keeps only one copy of a non-writable page (note the **r-xp** permission bits) in memory. As

a shared library, vDSO defines several symbols (think functions). One of these functions executes the preferred instruction for the system call. As the kernel builds and maps vDSO on its own, it has complete control of what this instruction would be. Other vDSO symbols may provide optimised versions for selected system calls (like **gettimeofday(2)**), which don't incur switch to the kernel at all.

Either way, when a switch to the kernel mode occurs, a system call dispatcher in the kernel ultimately runs. It analyses the system call number and calls an appropriate implementation. By convention, it carries the **sys\_** prefix. For our good old friend **uname(2)**, this will be **sys\_newuname()**. This function lives in **kernel/sys.c**:

```
SYSCALL_DEFINE1(newuname, struct
new_utsname __user *, name)
{
    int errno = 0;
    down_read(&uts_sem);
    if (copy_to_user(name,
utsname(), sizeof *name))
        errno = -EFAULT;
    up_read(&uts_sem);
    return errno;
}
```

As before, we simplified the implementation a bit to highlight the main points. The **sys\_** prefix is added in the **SYSCALL\_DEFINE1** macro. The code calls the **utsname()** kernel function, which gathers the required data from an internal kernel structure, and copies bytes to the userspace memory. Sometimes, there is a

## FINDING YOUR WAY

If you are wondering how Linux works, the sources are the ultimate answer. However, the kernel is a large program. Without dedicated navigation tools, it's very easy to get lost in its 50K+ source files.

You need to look up identifiers, like variables or function names, and search for raw text strings. You'd want to jump directly to locations where these identifiers are defined, but also list locations where they occur in the code. Quite often, you'd also want to know where some variable gets its value.

Luckily, there are several free tools to help you with all of these. A *de-facto* standard one is *Cscope*. It wasn't built specifically for the Linux kernel. In fact, it's almost as old as Unix. Born in Bell Labs, it was a part of AT&T. Santa Cruz Operations, a predecessor to the ill-fated SCO Group, released *Cscope* under the BSD licence in 2000.

Linux provides a dedicated Makefile target to generate the *Cscope* database. Running **make cscope && cscope -d** will bring you a text-based interface. Use the arrow keys to choose what you're looking for (say, a C symbol), type the search terms and hit Enter. Results are displayed in the upper half of the screen. Press Tab to switch between halves. Select the result with Up and Down keys, or press a single-letter hotkey to open the location in **\$EDITOR**. If your search yielded many results, use Space to turn pages. To exit, press Ctrl+D. The ? key brings the help page.

For a friendlier alternative, try *LXR* (Linux Cross Referencer). It's a web application built specifically for Linux, yet found its usages in other projects (*Mozilla*). You can find *LXR* online here: <http://lxr.free-electrons.com>.

First, select the kernel version. This defaults to the latest mainline. You can navigate the sources manually in the Source Navigation tab. If you want to trace the origins of some log message, *LXR* should be your first stop.

**do\_something()** kernel function that does all heavy lifting. A semaphore, **uts\_sem**, protects the structure from concurrent access (didn't I tell you that kernel programming is always asynchronous?). Also note that negative return values indicate an error. The convention in userspace is different, so the C library detects it and sets the **errno** variable appropriately.

We hope you enjoyed your trip under the Linux kernel surface. The kernel is huge, and there are many books devoted to single subsystems of it. They cover many nuances which are essential if you are serious about kernel programming. Yet the kernel is no magic, just different from what we have in a comfy userspace. It also implies a good understanding of computer architecture, even for generic pieces like the memory allocator. Not many of us write kernel code (if you have a patch accepted, please let us know!), but it's always fun to learn how gears fit together. 



## Linux Cross Reference

Free Electrons  
Embedded Linux Experts

• Source Navigation • Diff Markup • Identifier Search • Freetext Search •

Version: 2.0.40 2.2.26 2.4.37 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5 4.6

Linux/fs/iaio.c

```
1 /*
2 * An async IO implementation for Linux
3 * Written by Benjamin LaHaise <bcr@kvack.org>
4 *
5 * Implements an efficient asynchronous io interface.
6 *
7 * Copyright 2000, 2001, 2002 Red Hat, Inc. All Rights Reserved.
8 *
9 * See ../COPYING for licensing terms.
10 */
11 #define pr_fmt(fmt) "%s: " fmt, __func__
12
13 #include <linux/kernel.h>
14 #include <linux/init.h>
15 #include <linux/errno.h>
16 #include <linux/time.h>
17 #include <linux/aio_abi.h>
18 #include <linux/export.h>
19 #include <linux/syscalls.h>
20 #include <linux/backing-dev.h>
21 #include <linux/uio.h>
```

With *LXR*, you can navigate kernel sources straight from you web browser. It comes in handy when you read things like this web page.

inside

# LibreOffice

## The Document Foundation

Along with GNU/Linux and Firefox, LibreOffice is one of the biggest success stories in Free Software. Find out where it came from, and how it's developed.

**A**sk a Linux user to name some really exciting things going on in the Free Software world right now, and they will probably mention Gnome or KDE updates, or perhaps some flashy new graphical effects promised by Wayland, or maybe some awesome new games in the pipeline. Even new programming languages, container technology and *Firefox* updates can be fascinating. But office suites? Surely that's the least exciting thing on this wonderful green planet – right?

Well, no. It's true that the humble office suite is a mere tool for getting things done, a quotidian aid for productivity rather than something to wow and entertain you. But this is a topic that's well worth keeping an eye on, as it has the potential to completely change the computing landscape. We won't see Linux completely dominate the desktop market in the next few years – the days when we'd get excited about the year of Linux on the Desktop™ are over – but there's a chance that Microsoft's *de facto* monopoly

on office suites could at last be broken thanks to contenders like *LibreOffice*.

And the reason is simple: it's a much easier job to switch an office suite than it is to migrate an entire operating system. Consider how much success *Firefox* had in its early days; Microsoft's *Internet Explorer* utterly dominated the browser market, and many said it could never be beaten. But *Firefox*'s better performance, security and featureset brought it up to 30% marketshare, while *Internet Explorer* usage dropped quickly (especially thanks to competition from Google's *Chrome*).

### Reach for the prize

We could see the same with *LibreOffice*. Sure, switching office suites is a bigger job than changing browsers, but it's certainly possible. The French government has migrated hundreds of thousands of computers from *Microsoft Office* to *LibreOffice*, and the Italian military is also undergoing a major transition. And funnily



enough, while some bemoan *LibreOffice*'s "old-style" interface, many users of earlier *Microsoft Office* versions prefer switching to *LibreOffice* than having to use the ribbon that Microsoft is so proud of.

So, tens of millions of people use *LibreOffice* each day, the project is growing, and it looks to have a healthy future. But where did it come from, how does it work internally, and what's in the pipeline for the future? Over the next few pages, we'll give you all the details.

Tens of millions of people use LibreOffice each day, the project is growing, and it looks to have a healthy future

# FROM HUMBLE BEGINNINGS

Many of us cut our teeth on Linux back in the late 1990s, as it started to develop from a hacker's plaything into a viable operating system for servers and home desktops. Around the time, many computer magazines started featuring Linux distributions on their coverdiscs, along with a certain office suite called *StarOffice*. This suite had a strange look and feel: it tried to ape the Windows 95 desktop with a taskbar and Start-like button, and managed to maintain this design across the various platforms on which it ran (Windows, Solaris, Linux).

*StarOffice* came into the world in 1985 as *StarWriter*, a word processor developed by a German student for the 8-bit home computers at the time. In the following years, more components of the office suite were added and it was ported to MS-DOS, IBM's OS/2 and Microsoft Windows. *StarOffice* was a small but important player in the early 1990s, but something happened that suddenly elevated it into the spotlight.

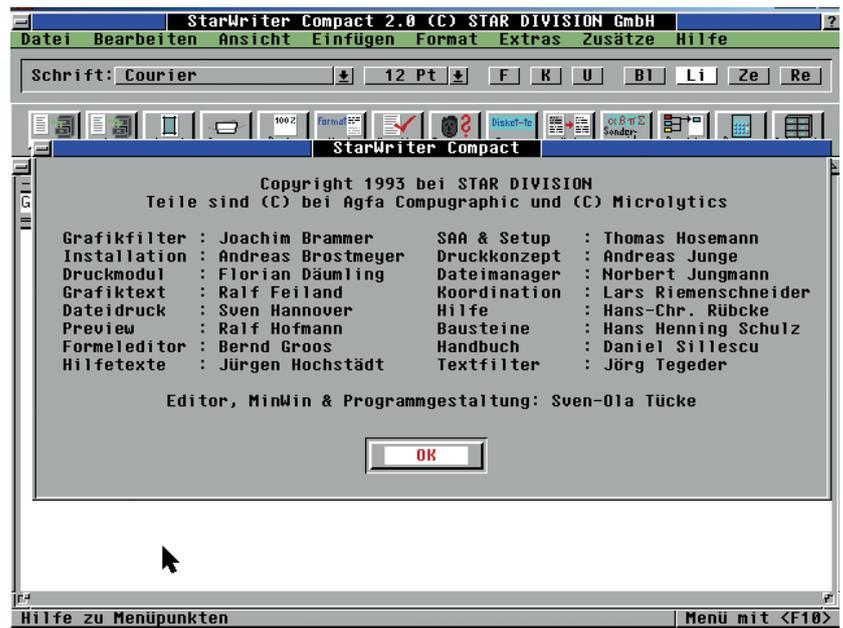
Unix giant Sun Microsystems (which is today owned by Oracle) was looking to install *Microsoft Office* on its 42,000 workstations. That's a hefty licensing cost – so Sun decided to buy *StarOffice* outright in 1999 as a cost-saving measure. *StarOffice* was also made available as a free download, so even while it was seen as a bit lacking and clunky in comparison to Microsoft's offering, many people used it as a cost-cutting measure alone.

But then Sun did something drastic: the company open sourced *StarOffice*, creating *OpenOffice.org*, which was a monumental effort. The suite was built from millions of lines of source code, many of which still contained German comments right from the early days of the suite, and getting the FOSS community on board was a difficult task. Early releases of *OpenOffice.org* were slow to use and glitchy in places but provided Linux distributions with a powerful office suite that abided by open source principles.

While Sun maintained control of the *OpenOffice.org* project, as the 2000s progressed a small community of independent developers, supporters and marketers built up around it. There was some concern that Sun had too much sway in the project, and these fears only got worse when Oracle snapped up Sun in 2010 – what would Oracle, hardly the biggest champions of open source in the world, do with the software now?

## A fork in the road

So a team of *OpenOffice.org* developers and supporters from the community forked the code and created *LibreOffice* in late 2010. This was one of the biggest splits in FOSS history, and with few developers left working on *OpenOffice.org*, Oracle decided to hand it over to The Apache Foundation (where it still survives today as *Apache Open Office*, albeit with very little development work underway).



One of the goals of *LibreOffice* was to be as independent from control of a single company as possible. To that end, the team behind it set up The Document Foundation (TDF), a non-profit entity registered in Berlin and with members across the planet. This was a bold move, especially for such a fledgling project, but it established a structure and statutes for the project to ensure democracy and transparency throughout. TDF is comprised of several bodies, including:

- **The board of directors** The main administration of

*LibreOffice's* codebase dates back to *StarOffice* in the 1980s (image credit: <http://tinyurl.com/heuv5q8>).

Early releases of *OpenOffice.org* were slow to use and glitchy in places, but provided Linux distros with a powerful open source office suite

*LibreOffice's* projects and teams.

- **The membership committee** Administers membership applications, and oversees election of the board of directors.
- **The advisory board** For companies and organisations that support *LibreOffice* to provide ideas and advice.

The statutes of TDF state the following: "The board of directors is therefore obliged to ensure, that the board of directors itself, the membership committee, and the advisory board, at maximum have one third of their members being employed by a single company, organisation, entity or their respective affiliates". This is a smart move that largely eliminates the problems of the past – too much control from a single company like Sun or Oracle.

# THE NOW AND THE FUTURE



Florian Effenberger is the executive director at The Document Foundation, managing the team that works on administration, documentation and marketing.

So *LibreOffice* has established itself, in part thanks to the efforts of The Document Foundation, as a healthy and vibrant followup to *OpenOffice.org*. But how exactly does the project work? Who's in control? Well, TDF is supported by donations, the majority of which come from end users who choose to give a bit of money when downloading the software. Many onlookers were sceptical, in the early days, that such a large project (with over 7 million lines of source code) could survive off donations – but TDF has shown that it can work.

Indeed, TDF employs a small team to help further *LibreOffice*. These aren't primarily developers, but staff working on other supporting aspects of the project, such as infrastructure, developer mentoring, documentation and marketing. Most of the development effort comes from hackers working at companies that use *LibreOffice* in their products –

Major releases such as 4.4 or 5.0 are issued every six months, and each of those releases receives a number of bugfix updates (such as 5.0.1). In addition, one release branch is named “Fresh”, meaning that it's the newest codebase which may have some features that need more testing; the other branch is called “Still” and is recommended for large-scale deployments in governments and businesses. At the time of writing, the Fresh branch is 5.1, while Still is 5.0.

## Long-term support

Speaking of deployments, while The Document Foundation strives to support each major release of *LibreOffice* for several months, it's limited in its resources and recommends that companies use Certified Developers ([www.documentfoundation.org/gethelp/developers](http://www.documentfoundation.org/gethelp/developers)) for long-term support. This is a bit like the Red Hat Enterprise Linux model, where you get the core product (in the form of CentOS) for free with some community support, but if you want to roll it out across 20,000 servers or workstations, it's probably wise to pay for commercial support as well.

As *LibreOffice* moves towards a new release, various teams inside the project hold meetings and post minutes on mailing lists. This includes the design team, engineering steering committee, documentation team, marketing team and others. The person coordinating this effort, and managing the staff at TDF, is the executive director, Florian Effenberger. We spoke to Florian way back in issue 1 for our first interview – so if you've been reading Linux Voice right from the start, dig out that issue and enjoy a look back at the state of *LibreOffice* back then!

The Document Foundation is supported by donations, the majority of which come from users who give a bit when downloading the software

such as Red Hat, Canonical and Collabora (see our interview with Michael Meeks in issue 5). Plus, of course, there are contributions from other developers in the community as well.

In order to get releases out of the door – and avoid Debian-esque “we'll release when it's ready” delays – TDF adopted a time-based release schedule.

Because the *LibreOffice* codebase is so vast, it can be difficult for new developers to get involved. In recent years, *LibreOffice* developers have done an impressive job tidying up the codebase, making it easier to navigate and build, and removing lots of those ancient German source code comments. In addition, because *LibreOffice* is designed to run on many platforms, there are large levels of abstraction in the suite, which have caused problems (especially with performance) in the past. Collabora's Michael Meeks has spent a lot of time wrestling with the internals of the software to remove old bottlenecks and make it more responsive – see <http://tinyurl.com/q72bctv> for a (very technical) description.

When a new release is available, TDF doesn't simply chuck it onto the downloads page and wait for mirrors to propagate. No, there's an increasing need to market the software professionally, making sure that end-users and journalists are fully aware of the new features and changes. In addition, TDF holds a yearly conference where developers, users and supporters meet up – this year it will be held from 7–9 September in Brno, Czech Republic, so visit <https://conference.libreoffice.org> if you're interested.

### What's coming up?

The next major release of *LibreOffice*, version 5.2, is due in early August 2016. This will include interface improvements in *Writer* (such as a single toolbar mode for low-resolution displays), new spreadsheet functions in *Calc*, along with fixes and enhancements in other areas of the suite. For companies working with sensitive data, document signing has been improved as well. And, as with every release, a lot of work has been done to make the *Microsoft Office* file format filters even more exact.

While the standalone desktop version of *LibreOffice* is coming along well, there's increasing demand (especially from businesses that want to switch to the software) for a cloud version. Progress is being made in this area: Collabora, one of the major *LibreOffice* contributors, has announced version 1.0 of its Collabora Online suite ([www.collaboraoffice.com](http://www.collaboraoffice.com)), a trimmed-down version of *LibreOffice* that's accessible in a web browser. While Collabora aims to make money providing support for its version, the company is donating code back to the main *LibreOffice* source tree, so a community-supported version should follow.



But why is a cloud version so important? The big issue is administration. Imagine you're responsible for IT in a company with 10,000 desktop PCs. When a new version of *LibreOffice* is available, you have to roll it out on every computer, make sure it's all working correctly, and that nobody has messed up their installation with some custom settings or packages. There are ways to handle this in a more automated fashion, but it's tough.

*LibreOffice* is heading to the cloud, thanks to the work of Collabora and developers in the community.

While the standalone desktop version of *LibreOffice* is coming along well, there's increasing demand for a cloud version

With a cloud version, the software is run on a server and rendered in users' web browsers – so there's only one version to worry about. Admins can upgrade and customise that single version on the server, and all end users will receive the changes via their browsers. So it makes the admin's life much easier, and also means that end users can connect to the server from different machines and locations, always interacting with the same *LibreOffice* instance.

So there's plenty to look forward to in *LibreOffice*, and the project looks to have a healthy future. There's even the possibility that *Thunderbird*, Mozilla's email client, could be integrated with *LibreOffice* some day to provide a complete productivity and collaboration solution to compete with *MS Office* and *Outlook* 

## GET INVOLVED!

Because *LibreOffice* is such a large project, the idea of getting involved may seem incredibly daunting at first. And indeed, up until a few years ago, the barriers to entry were rather high. But the *LibreOffice* team has done a lot of grunt work to simplify the build process and introduce new developers to the codebase, so there has never been a better time to contribute. Plus, saying that you've been involved in a major Free Software project like *LibreOffice* is mightily good for your CV!

The best place to start is [www.libreoffice.org](http://www.libreoffice.org). Click the Community menu on the top-right and you'll see different areas of the project: design, development, documentation, marketing, native-lang projects (for translations) and testing. So even if you're not a coder, there are plenty of ways to get involved. Even if you can only spare half an hour a week to update some documentation, translate a few interface strings or confirm a handful of bugs, all participation is greatly appreciated.

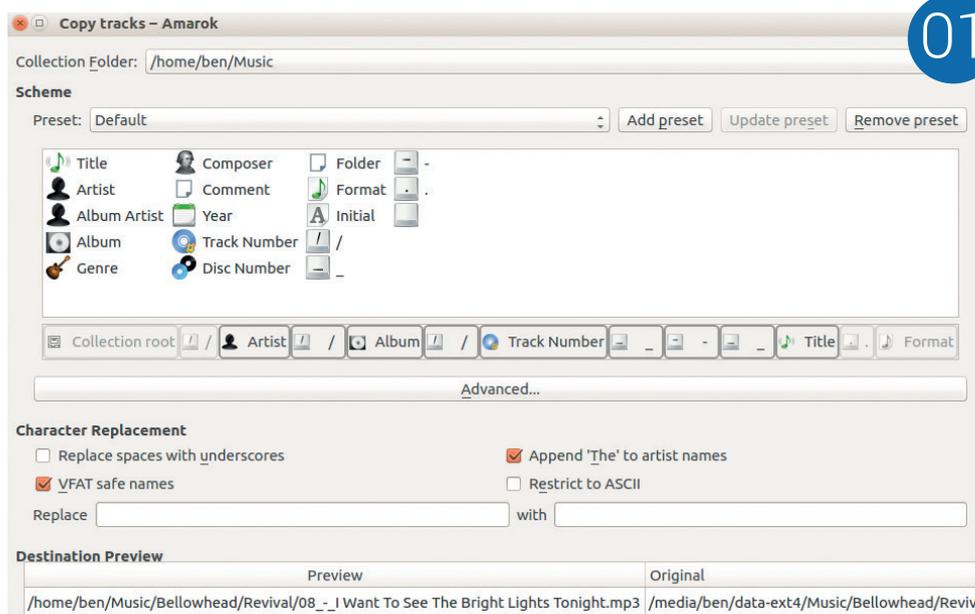
# SECRETS OF AMAROK



## Get the most out of KDE's in-house music player.

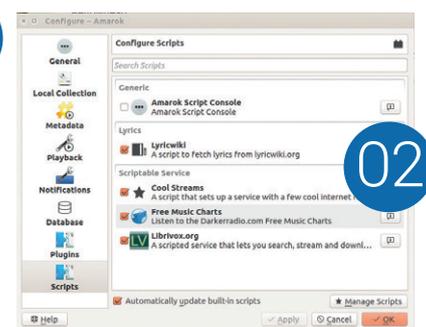
**M**usic touches a part of our souls that rational thought can't reach. It can make us smile when we're sad, help us remember events we'd forgotten and compel us to dance even when we're tired. It's a cultural universal, and every group of humans on earth has some type of music, though it comes in many different forms.

Something that's so essential to the human condition deserves to be treated as a priority on our digital setups. Mastering your music collection probably won't make you any more productive, or make your computer run faster, but it may well make you happier, and we think that's more important. Let's find out how to get the most out of *Amarok*, one of the most popular music players for Linux.



**01 Manage your tracks** If you're like us and have a mass of half-sorted MP3 files from two decades of ripping CDs and downloading from various sources (all legal of course), you'll find *Amarok's* music management features useful. It can shift your tracks into a properly structured directory system with just a couple of clicks.

**02 Scripts** *Amarok* has tons of features, but it doesn't have everything you could possibly want. If there's a particular thing that you want to do that's not yet possible, you can use JavaScript and the Qt bindings to add more functionality. There's also a large library of scripts already available to do things like add internet radio

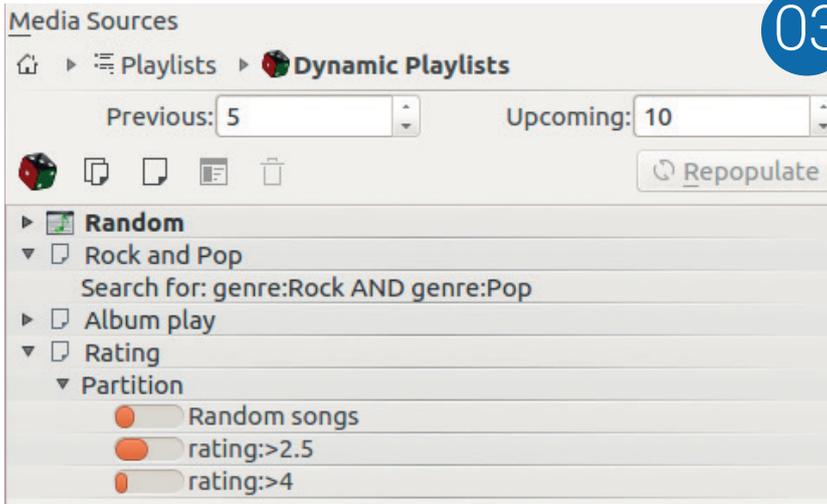


stations or lyric-streaming services. To manage your scripts go to Settings > Configure Amarok > Scripts.

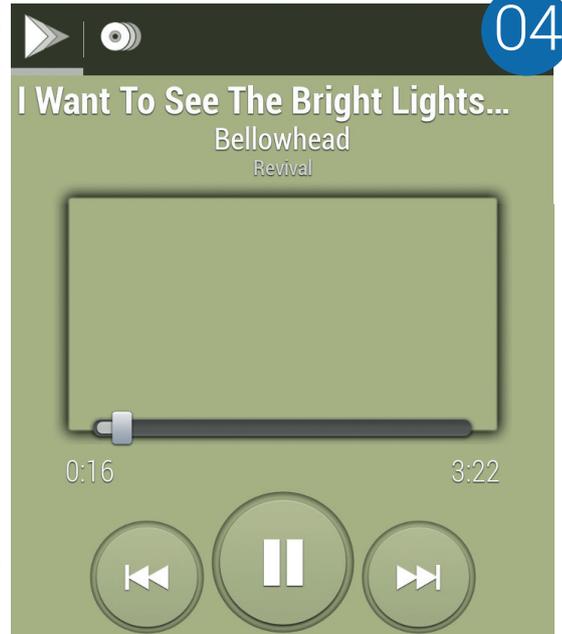
**03 Dynamic playlists** If you've got a lot of music, it can be hard to find the tracks you want to listen to. We don't mean locate a particular track, but decide the right selection of music for the moment. With *Amarok's* Dynamic Playlists, you can specify a set of rules and the software will find a selection of tracks for you to listen to. If you want 80s rock (and why wouldn't you?), just set up the Dynamic Playlist, put on your mullet wig and ripped jeans, then head-bang the night away. *Amarok* will keep your playlist topped up with more music from your collection so you won't run out.

**04 Remote control** While it isn't a core part of *Amarok*, there's additional

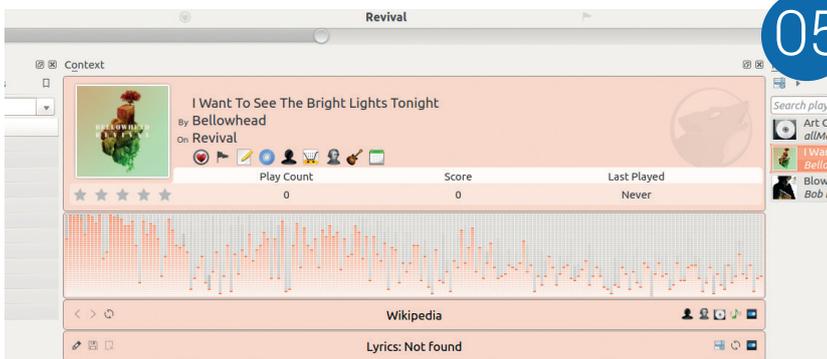
With *Amarok's* Dynamic Playlists, you can specify a set of rules and the software will find a selection of tracks for you to listen to



03



04



05

software you can use to control your *Amarok* playback while you're away from the desktop, including an Android app (*AmaroKontrol* <http://bit.ly/1TaE08Q>). *Amarok* is probably a bit too heavyweight to make it useful as an embedded music player, but this feature enables you to use your PC's sound system as a general music player without having to switch back to the mouse or keyboard regularly.

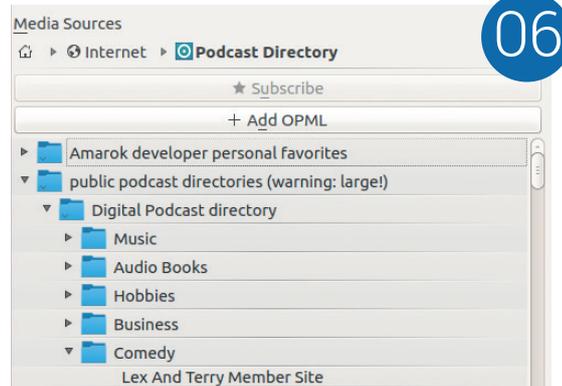
**05 File tracking** Your music player builds up a lot of information about the music files you have – how much you play them, whether you favourite them, etc. This is great until you move your collection onto a different hard drive and suddenly all this information is gone. Well, not with *Amarok*: it can still link your information to your music if it's in a different location, so you never lose this metadata.

**06 Buy music** Want to listen to some new music? *Amarok* can integrate with Amazon, Magnatune and MP3tunes (through plugins) to give you

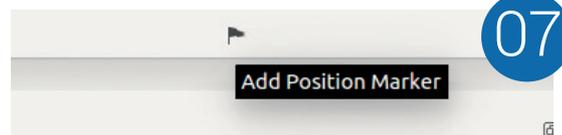
the ability to get all the latest music legally. If you're after something a little less commercial, you can use Jamendo to get music by independent artists, and if you're more interested in spoken word than tunes, there's integration with Podcast Directory and Librevox.

**07 Bookmarking** If you're listening to a podcast, audio book or long piece of music, you might want to return to a particular point. This could be because you want to restart in the same place, or there's a particularly poignant or interesting section you want to return to. *Amarok* enables you to bookmark places in your tracks so you can easily return without having to memorise the time position in the track.

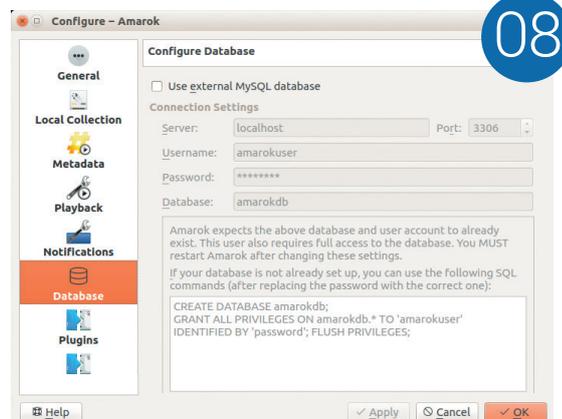
**08 Konfiguration** *Amarok* is the KDE music player, so as you would expect, it's highly customisable. In fact, just about every aspect of the application can be tweaked to your desires, even to the extent of selecting the position on the screen that notifications will appear. What other



06



07



08

music player enables you to connect to an external *MySQL* database? All this configuration works equally well on *Qt*- and *GTK*-based desktops, so you don't have to be using KDE to reap the full benefits.

# Pis In SPACE!



Les 'Spaceman' Pounder calls occupants of interplanetary craft to find out what's been going on with Linux and computing in low Earth orbit.

On December 15th 2015, astronauts from NASA, Russian Space Agency and the European Space Agency blasted off from Baikonur Cosmodrome, Kazakhstan as part of the Principia mission. Principia, or *Philosophiæ Naturalis Principia Mathematica*, Latin for "Mathematical Principles of Natural Philosophy" was the mission name chosen by Flight Engineer 4, former army major and test pilot Tim Peake. The goal of the Principia mission is to work on experiments that cannot be conducted on Earth; these include physics, biology and technology demonstrations. The location for these experiments being the International Space Station (ISS) in orbit around Earth.

In 2009, Tim Peake was appointed an ambassador for UK science and space-based careers and worked with the various agencies to promote science and engineering as career possibilities for school

children. Tim's aspect of the mission was enhanced with the use of a ubiquitous platform: the Raspberry Pi, which made its way to space because UK Space Agency wanted to engage with the public via a series of experiments and agreed that a harmonised platform – the Raspberry Pi – would be beneficial to children, offering as it would a unique opportunity to use the

Using the Sense HAT we can measure temperature, humidity, acceleration, pressure, orientation and magnetic field

same hardware as an astronaut in space. After passing every test to ensure that the equipment was ready for flight final certification for use aboard the ISS was awarded. The Astro Pi project was ready for launch.

The hardware behind Astro Pi is an add-on board for the Raspberry Pi, called Sense HAT. The Sense HAT board

comes with a plethora of sensors used to gather data; using the Sense HAT we can measure temperature, humidity, pressure, acceleration, orientation and magnetic field strength and direction. Also present on the board is a miniature joystick for basic input and an 8x8 grid of multi-colour LEDs, which can be used as a method of output.

Children from around the United Kingdom were asked to submit their project ideas for consideration, and from the many hundreds of applications seven were chosen. We're going to look at three that piqued our interest and which can be easily replicated at home.

Due to the length of these projects we have cherry-picked interesting parts of the code, but we have included a link to the project page which will provide all of the code that you will need to replicate the projects on planet Earth.

Get ready for blast off!

# PROJECT 1 – WATCHDOG

Keep astronauts safe and sound with a Pi and a mere 2,000-odd lines of code.

This project, created by Cottenham Village College, is a backup environmental system monitor designed to cross-check the ISS's own environmental control systems. *Watchdog* works by using the sensors present on the Sense HAT, chiefly the temperature, pressure and humidity. Any changes outside of mission parameters will cause the alarm to trigger, alerting the crew to an incident.

As this project is huge, totalling 2,681 lines of code, we shall take a look at sections of the code; for the full code, please visit <https://astro-pi.org/competition/winners/#watchdog>. The first thing that impressed us was the diligent use of comments to create a “table of contents” used to identify what sections of code control the various aspects of functionality. Here is a snippet to illustrate their use.

```
# 1 # CREDITS [165 - 168]
```

```
# 2 # IMPORT MODULES [171 - 180]
```

```
# 3 # SETTING UP PROGRAM [183 - 206]
```

```
# SETS ASTROPI MODULES AS FRIENDLY NAME [185 - 187]
```

```
# SETTING UP RASPBERRYPI FOR FLIGHT BUTTONS TO USE  
GPIO PINS [189 - 192]
```

```
# ASSIGNING FRIENDLY NAMES FOR GPIO PINS [195 - 202]
```

```
# FORCING PROGRAM TO RUN PROGRAM WITHIN WHILE LOOP  
[204 - 206]
```

After the comments the code starts by importing a series of libraries to enhance the project. We start by importing the **RPi.GPIO** library, this enables the ISS crew to control the project using the joystick and buttons present on the Astro Pi units. Next we import the time-logging library, which will be used as a counter. We import the **time** library and the **sleep** function to control the pace of the project. The **asctime** function is used to convert the time into a human-readable format. The other imports cover using the filesystem, the Sense HAT and the camera.

```
import RPi.GPIO as GPIO
```

```
import time, logging
```

```
from time import sleep, asctime
```

```
import datetime
```

```
import sys, os
```

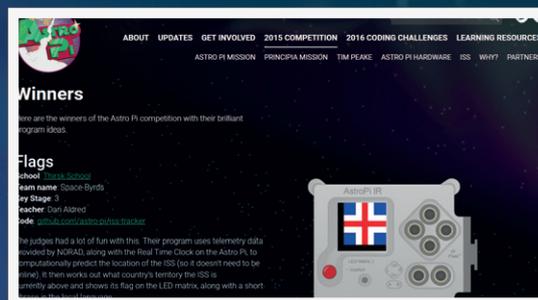
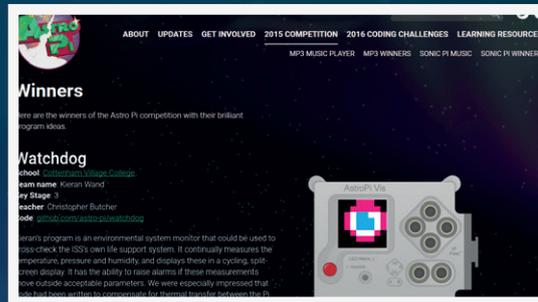
```
import astro_pi
```

```
from astro_pi import AstroPi
```

```
import picamera
```

Going down the code to line 212, we can see a time stamp used to keep an accurate reference point for data collection. From lines 233 to 248 we can see variables used to store default values for the various sensors present on the Sense HAT. Using these values the team cleverly bypass an issue where the temperature sensor returns a higher than normal value, largely due to being placed over the Pi's CPU.

The class logged all of the data to an external CSV file, which can later be imported into a spreadsheet for



**Top:** *Watchdog* is a clever project that monitors the crew's status and alerts them to any life-support issues. Including recording data to a black box for retrieval.

**Bottom:** Using some advanced calculations to plot the course of the ISS and detect which country it is over, *Flags* is a cross curricular learning project.

further investigation.

```
file = open('log/'+(str(tmstamp))+ ' watchdog-log.csv', 'w')
```

```
file.write("\nTime","\nDisplay","\nTemperature","\nTemp_Reading","\nTemp_Alarm","\nTemp_Snapshot","\nHumidity","\nHum_Reading","\nHum_Alarm","\nHum_Snapshot","\nPressure","\nPSI_Reading","\nPSI_Alarm","\nPSI_Snapshot","\nPitch","\nRoll","\nYaw"\n")
```

Going from lines 2094–2153 we see a function used to detect and react to user action, such as pressing the joystick right to show the air pressure on the LED matrix. On line 2155–2157 we see a **for** loop, cleverly configuring each of the GPIO pins used for user input.

```
for pin in [UP, DOWN, LEFT, RIGHT, A, B]:## SETUP GPIP PIN  
VALUES
```

```
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```
GPIO.add_event_detect(pin, GPIO.FALLING, callback=button_pressed, bouncetime=500)
```

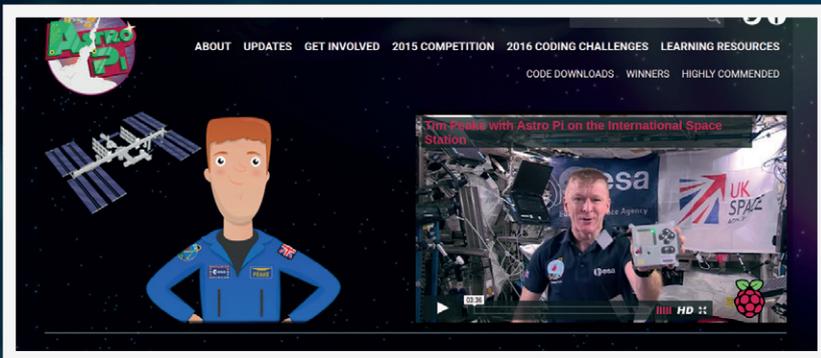
From lines 2185–2294, we see classes created. The first class, **AstroPiContinuous(AstroPi)**, contains configuration for all of the sensors including the camera. The second class, **CPUTemp**, is used to configure, read and convert the temperature taken from the temperature sensor.

From line 2299 to the end we see a **try, finally** construction used to handle reading the sensor data, write it to an external file and take pictures using the Raspberry Pi camera. There are sections of code that can mute the alarms and use the *Watchdog* project as a blackbox flight recorder. Once the project ends the code cleans up and exits to the command line

A great piece of code!

# PROJECT 2 – FLAGS

Where in the world is our intrepid team of space adventurers?



**Right:** The Astro Pi website, <https://astro-pi.org>, contains more information about the competition winners, the goal of the Principia mission, and regular blog posts from the ISS.

Team Space-Byrds from Thirsk School have created a great project to track the ISS. Typically we only hear about NORAD at Christmas, thanks to a long-running Santa Tracker, which originated via a joke communiqué in 1948 which continues to this day. But NORAD provides telemetry data for satellites in orbit, and this project uses this data to track the ISS without the need for an internet connection. The location data is then used to display the flag for the country over which the ISS is orbiting, and then scroll a phrase in the local language for that country.

All of the code for this project can be found at <https://astro-pi.org/competition/winners/#flags>.

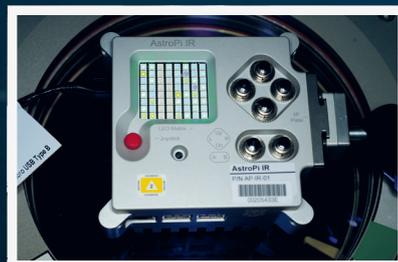
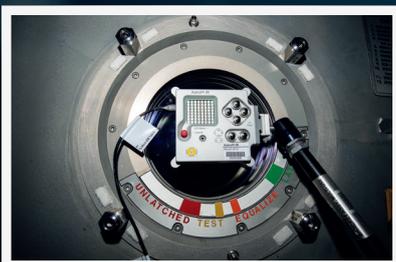
What first struck us about this project was the use of the **ephem** library. This library provides a means to perform high-precision astronomical computations.

On lines 17 and 18 we see two lines of numbers; this is satellite data for the ISS, which we shall later convert to a format for use with **ephem** on lines 33 and 34. Inside of a **while True** loop, lines 27 to 30, we see three lines of code that capture the current temperature, pressure and orientation of the Sense HAT.

```
....
temp = str(ap.get_temperature())
pressure = str(ap.get_pressure())
orientation = ap.get_orientation_degrees()
....
```

From lines 36 to 48 we see the latitude and longitude calculated and converted from a string of characters, splitting the string by identifying a

**Below:** The Astro Pi units are safely installed aboard the ISS and are being routinely used by Tim Peake who is running code written by UK school children.



delimiter and using that as a pattern. Finally the new strings are converted to float values.

Lines 54–1507 comprise a series of conditional tests used to check the longitude and latitude of the ISS against the values held for a particular country. Two variables, X and O, are used to create a list, with a layout of 8x8, to represent the LED matrix on the Sense HAT. By positioning the X and O in the correct place we can create the correct flag for a country. With the list created the pixels are then updated to show the flag. The code then pauses for six seconds before scrolling two messages across the LED matrix, in the local language for this country. Here is the example for the United Kingdom.

```
if (lati[0] <= 53 and lati[0]>= 52) and (longt[0] >= -4 and
longt[0]<= -1):
    print "United Kingdom"
    X = [255, 0, 0] # Red
    O = [255, 255, 255] # White
    UK = [
        0, 0, 0, X, X, 0, 0, 0,
        0, 0, 0, X, X, 0, 0, 0,
        0, 0, 0, X, X, 0, 0, 0,
        X, X, X, X, X, X, X, X,
        X, X, X, X, X, X, X, X,
        0, 0, 0, X, X, 0, 0, 0,
        0, 0, 0, X, X, 0, 0, 0,
        0, 0, 0, X, X, 0, 0, 0
    ]
    ap.set_pixels(UK)
    time.sleep(6)
    ap.show_message("Hello ISS, you are over the UK")
    ap.show_message("Hello ISS. How are you!", text_
colour=[255, 0, 0])
```

At the end of the conditional test we have an **else** condition, line 1509, which is used to quickly scroll the current temperature. It then uses a **while** loop that will loop and count down from 5000 to 0. While counting down random integers are chosen for the x,y co-ordinates of pixels on the LED matrix. The mix of coloured light is chosen at random, giving us the illusion of blinking LEDs computing a problem.

```
while FLASH > 0:
    x = randint(0, 7)
    y = randint(0, 7)
    r = randint(0, 0)
    g = randint(0, 100)
    b = randint(0, 255)
    ap.set_pixel(x, y, r, g, b)
    FLASH = FLASH - 1
```

The code then advises the user that their location is being computed, ready to display the correct flag. The pressure is calculated and shown on the LED matrix before the matrix runs the computer animation.

# PROJECT 3 – SPACECRAFT

## Lego, on a computer, in space...

Hannah Belshaw from Cumnor House Girls School has created a great way of visualising sensor data by using *Minecraft*. This project has many different aspects; we'll focus on the data logger and the *Minecraft* data playback scripts. All of the code for this project can be found at <https://astro-pi.org/competition/winners/#spacecraft>.

The data logger code, `astropidatalogger.py`, is rather short as it imports a lot of configuration from an user-created external library. The data logger process is called on lines 29 and 32. With line 32 the logger starts to gather data, saving the output to an external CSV file. The library contains all of the raw code that interacts with the sensors and formats the data inclusion in a CSV file. In the external library, `astropidata.py`, on line 52 we see the creation of a class designed to log the sensor data. There are a number of functions starting from line 93, which handle printing verbose information to the terminal. On line 188 we see the creation of a dictionary, used to hold specific information that can be referenced via a key, typically the name of the data that we wish to save. The names of the fields for the data spans all the way to line 231.

```
datarow = {}
datarow[DATETIME] = datetime.now()
datarow[TIME] = time()
datarow[CPU_TEMP] = cpu_temp.get_temperature()
datarow[HUMIDITY] = ap.get_humidity()
```

From lines 248–362 we see another class that reads the data from the sensors and then stores it in the appropriate section of the dictionary. Here is an excerpt from line 344–347 showing the accelerometer data being stored.

```
def get_gyroscope_raw(self):
    return {"x": float(self.data[GYRO_RAW_X]),
            "y": float(self.data[GYRO_RAW_Y]),
            "z": float(self.data[GYRO_RAW_Z])}
```

### Minecraft data playback

Playback of the data captured is handled via `mcastroplayback.py` and this code visualises the data using *Minecraft* and its many different block types. On line 39 we see the raw CSV data being imported into the code.

```
apr = AstroPiDataReader(self.filename)
```

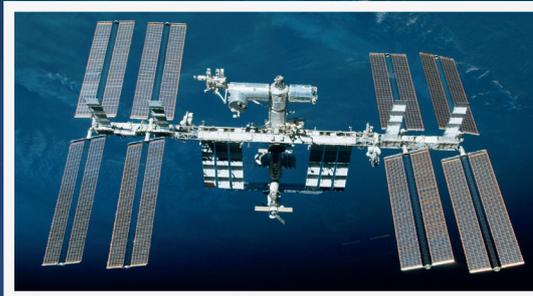
Further down the code, from line 45 onwards, see the beginnings of the *Minecraft* code; on line 50 we get the position of the player so that any visualisations are created near to their location. From lines 65–73 we see the code that will update the ISS data visualisation with the latest information.

```
isstowdisplay.update(
    apr.get_time(),
```



**Top:** As well as Astro Pi, the crew of the ISS also run other science experiments – here we see blood being taken from Tim Peake by Commander Timothy Kopra.

**Bottom:** The International Space Station is huge – measuring 109 metres by 73 metres it is the largest man-made object in Earth orbit.



```
apr.get_cpu_temperature(),
apr.get_temperature(),
apr.get_humidity(),
apr.get_pressure(),
apr.get_orientation(),
apr.get_joystick()
```

From line 113 we see the creation of another class. This time it handles a command line interface enabling the user to interact with the data via a shell interface. Commands entered into the shell can be passed arguments, extra information or configuration, in this case it is the name of the file where the data is stored.

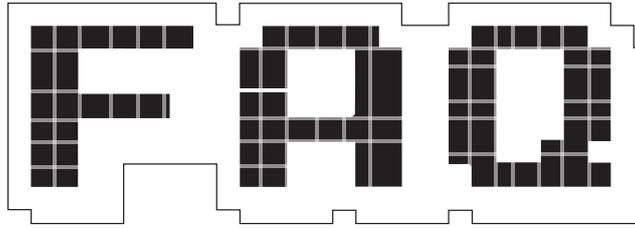
```
def __init__(self):
    Cmd.__init__(self)
    self.prompt = "SpaceCRAFT $ "
    self.intro = "Welcome to SpaceCRAFT data playback. Type
help or ? to list commands.\n"
    self.playback = None
```

From lines 181–194 we see a function that handles the playback speed of the visualisation, either real time, x1 or the value can be increased doubling each time all the way to x16 speed. The code for playback speed also incorporates error handling, in case the user provides an incorrect speed.

At the end of the code we see two lines of code that handle running the project and calling the various classes that make up the code.

```
if __name__ == "__main__":
    PlaybackCommands().cmdloop()
```

This robust project encompasses many different aspects of Python, data structures and provides an entertaining method of understanding data. 📺



# GCC

The GNU Compiler Collection may be the most important piece of free software ever developed – even more than the Linux kernel. Here's why.

MIKE SAUNDERS

**Q** Whoa, it says up there that **GCC is more important than the Linux kernel! Can I have some of what you're smoking?**

**A** No, we're serious. Yes, the Linux kernel is an incredible accomplishment, a vast body of code and effort from thousands of talented developers. It's arguably the best operating system kernel in the world. But in order to understand the importance of *GCC*, we have to step back in time a bit. Don't worry though – this isn't going to be a drawn-out history lesson.

Picture the scene in the late 1990s and early 2000s. There were several free software operating systems making good progress: GNU/Linux, FreeBSD, OpenBSD and NetBSD. Had the Linux kernel – for some reason – been made completely illegal, or

Linus Torvalds managed to erase all traces of its source code from the world and retired to a desert island, we had other options. Most of the FOSS computing world could have switched to FreeBSD, for instance, which does a similar job to GNU/Linux as a workstation and server Unix flavour.

But all of these other operating systems had something in common: they were all built with *GCC*, the GNU Compiler Collection (formerly the GNU C Compiler, but the name was changed as it evolved to support more programming languages). Had *GCC* disappeared or been made illegal back then, the Free Software community would have been up a very unpleasant creek without even the slightest hint of a paddle.

We had various kernels, text editors, libraries, window managers and other tools that could be swapped out, but *GCC* was one of a kind. Sure, other

FOSS compilers existed – but they were very primitive in comparison. *GCC* was simply years ahead, and was so dominant in the Free Software world that many programs used *GCC*-specific extensions in their source code.

**Q** OK, so *GCC* was a cornerstone of FOSS development back then. But what about today?

**A** Well, it's still a hugely significant piece of software, and the default compiler on pretty much every GNU/Linux distribution, but some competition is emerging in the form of LLVM/Clang. This is a compiler toolchain for C, C++ and Objective C(++) that's released under a more permissive licence than the GNU GPL used by *GCC* – so in other words, LLVM/Clang is more appealing to proprietary software companies. Apple has been one of the leading developers of LLVM/Clang, and other closed source companies have gotten involved as well.

Right now, *GCC* and LLVM/Clang are very close in terms of performance (both in compile times and generated code), with each project claiming a lead in certain areas. But LLVM/Clang has

GCC is still a hugely significant piece of software, and the default compiler on pretty much every GNU/Linux distribution

shown enough maturity to be the default compiler in FreeBSD 10 onwards – although *GCC* is still available as an option. By the end of 2014, LLVM/Clang was able to compile almost 95% of the Debian software archive without problems, which is mightily impressive considering that's over 40,000 packages.

**Q So is that the end for *GCC*? Is LLVM/Clang the future?**

**A** That remains to be seen. One thing that makes LLVM/Clang so appealing to developers is its modularity. The toolchain can be neatly split up so that the separate parts can be used in an IDE (integrated development environment), providing useful debugging and error messages along the compilation and linking chain. In comparison, *GCC* makes it harder for IDEs to access the code during the intermediate compilation steps, providing hurdles for seamless integration.

GNU founder Richard Stallman has said this is a feature and not a bug, and essential to keep *GCC* away from being wrapped up in proprietary IDEs. But at the end of the day, it's down to the old battle of idealism vs pragmatism. If LLVM/Clang ends up producing faster and smaller code while working much better with IDEs, many hackers may be willing to forgive its more permissive licence and just get their work done.

**Q But surely *GCC* is making progress, right? I heard that there was a new 6.1 version released just recently...**

**A** Yes, *GCC* is still going strong and has a lot of hugely talented people working on it. Version 6.1 was a major upgrade, using the C++14 standard by default (from 2014, as opposed to previous versions of *GCC* which defaulted to the 1998 standard). Support for very old architectures is being dropped, new optimisations are being rolled into the code base, and the runtime library is being worked on too.

It's also important to note that *GCC* supports a wide range of CPUs and languages, and is being updated for relatively new languages such as Go. So we don't have any deep concerns about *GCC*'s future, even with LLVM/Clang wooing lots of developers. In an

ideal world, both projects will provide healthy competition and result in even better compilers.

**Q OK, so let's get a wee bit technical. How exactly does *GCC* work?**

**A** Compilers are incredibly complicated pieces of software, and warrant entire books about their design – but we can summarise the key components here. *GCC* doesn't simply read a piece of code like `puts("Hello world");` and determine the right CPU instructions to do the job; instead, it's made up of multiple components that go through the code in a stage-by-stage basis.

To start off, the "front end" part of the toolchain looks at the human-written source code, parses it, checks for errors and generates a "syntax tree" (a tree-like representation of the code). This syntax tree is not specific to any particular language – so you have front-ends for C, C++, Ada and other languages that all generate a syntax tree in the same format for the next step of the compilation process.

This middle stage performs optimisation on the syntax tree, creating an intermediate language that looks a bit like assembly language but is still CPU-independent. Finally, the "back end" takes the intermediate language and converts it into assembly language for a specific architecture. That code is then assembled and linked, resulting in a binary executable that you can run.

Because compilation is split up into these separate stages, developers can work on the parts they like without having to know everything about every language and CPU. Someone with extensive knowledge of C++, for example, can hack on the C++ front-end without having to know anything about x86 or ARM processors. Similarly, a low-level coder who's ace at micro-optimisations on x86 chips can work on the back end without having to deal with the complexities of C++.

**Q Ooh, that actually sounds like fun. I did a bit of x86 assembly language once, thanks to a certain tutorial series in Linux Voice...**

**A** Then get involved! The *GCC* website at <https://gcc.gnu.org>



**Protip:** despite what the *GCC* logo may have you believe, gnus are not born in eggs, nor do they hatch out with fully-grown antlers.

has mountains of information; in particular, the page for new contributors at <https://gcc.gnu.org/contribute.html> is a good starting point. *GCC* is a large and mature project with some of the most experienced hackers on the planet, so it may take a while to get yourself familiar with the codebase and infrastructure, but any contributions are welcome. And as with almost every Free and open source software project, source code is just one slice of the development pie – the team also welcome bug reports, documentation updates and other non-hacking contributions as well.

If you want to get involved but don't know where to start, or you're short of ideas, see <https://gcc.gnu.org/projects> – and especially, the "projects for beginner *GCC* hackers" link. This contains a list of newbie-friendly hacking tasks, such as splitting up giant source code files and removing duplicated code. These jobs may sound trivial in the grand scheme of things, but they help to make *GCC* tidier, cleaner and more accessible for other contributors. Plus, being able to say that you contributed to a compiler is yet another thing to shout about on your geek business card... 🐮

# JIM KILLOCK

Country mice Andrew Gregory and Ben Everard venture to scary London to talk about freedom, surveillance, and why we should be bothered.

**T**he Open Rights Group: if you haven't heard of it, look it up. ORG is one of a handful of bodies dedicated to lobbying MPs, raising public awareness and fighting injustices that crop up in the modern world. Whether it's data privacy, surveillance or the right of online firms to use a level playing field, the Open Rights Group is there, making a load of good arguments with a big spoonful of common sense. We spoke to ORG's executive director, Jim Killock...

A man with glasses and a light blue striped shirt stands with his arms crossed in front of a wall painted with large, colorful geometric shapes in red, green, and white. A black skateboard is leaning against the wall to his left.

**“It doesn’t make me very comfortable that my personal internet activities are being assessed for how suspicious they look. That places me under suspicion; it places you under suspicion, it places your readers under suspicion.”**

**LV** **Let's get down to business – what is the Open Rights Group working on right now?**

**Jim Killock:** We're working on mobile data and how mobile phone companies are collecting and using people's data. Both data protection and privacy are going to be big debates in Europe over the six months, particularly privacy.

**LV** **That's if we're still in Europe – after the referendum we might have left...**

**JK:** I mean... yeah... ORG is not pro- or anti-Europe as such. But assuming we did leave, there are really big questions for digital rights. How do you do all of the telecoms regulation? What do you do with data protection? What do you do with e-privacy? Do we stay in the digital single market? And the curious thing there is that if you stay in the single market then you have to keep all these laws, which means the situation simply doesn't change. So leaving Europe might not make a great deal of difference; it might well be that a lot of these laws stay exactly the same

**LV** **That's the Norwegian model, isn't it?**

**JK:** It is. And the thing with that of course is that that might not be very politically palatable. So you've got a hard choice between the economically easy route and the politically tenable route. I think it's going to be quite hard to leave Europe and then say "well actually we're going to keep everything the same and keep adopting European laws", because that's going to make no sense. It's very hard to know which of these tendencies wins out.

I wonder how the UK parliament would cope with doing this sort of legislation. They're very detailed, technically intricate and not very interesting to voters most of the time, and I don't think that our MPs are going to have a lot of time or interest for these reasons.

What would happen in practice is that our civil servants would do all of this and it would all go through as statutory instruments. So you'd still have a democratic deficit, possibly a worse one, because there is some kind

of transparency in the EU process, and it's probably less when it's civil servants drafting SIs.

**LV** **So who, in the UK government, gets it, do you think? Tom Watson and David Davies are the obvious two that spring out. It seems to me that, there's an increasingly authoritarian streak in**

GCHQ are scoring everyone on the basis of how much risk they think we are

**mainstream politics, and there are only a few people who understand that spying on people is A Bad Thing.**

**JK** People in government in the cabinet are always under a lot of pressure to play to the gallery, and therefore even the people who might understand the problems don't often vocalise them.

In parliament at the moment we do have a shortage of attention to these



Jim tells us that it's hard to get the public interested in TTIP (how do you campaign on something that's happening in secret?), but it's most probably not going to happen anyway.



**“The first thing you’ve got to remember is that politicians are not overly familiar with all of the detail of surveillance law or what might constitute mass intrusion”.**

issues. We do have some interesting MPs who were elected on the Labour backbenches who are sort of making a bit of progress, but none of them have had time to grapple with this agenda.

I think we miss Julian Huppert [former Liberal Democrat MP for Cambridge]; he was very good, and it’s a pity that we don’t have him in parliament. Within the Conservative ranks there are a number of people who are very good. Obviously David Davis, but at least one or two of them have ended up in the cabinet, where they effectively get shut up, so you don’t hear from them.

I think the other thing with the Conservatives right now is that obviously they’re preoccupied with Europe – that’s the problem with the Investigatory Powers bill right now. A lot of the people you’d expect to have at least some kind of interest in this have not been paying attention, because they’re putting all of their energies into the referendum. And yet these next few weeks are also the critical time for the Investigatory Powers bill, making the public side of campaigning very very difficult. Attention is elsewhere.

**LV** **Labour’s rationale for abstaining on the first vote was that they would be able to**

**oppose it more effectively at a later stage. Do you think that’s a winning strategy?**

**JK** I don’t think it is particularly effective right now, but I understand why they did it. Everybody has known for a while that these laws need updating; that’s partly because of Snowden – you can see that the law is out of kilter with what’s happened, what the practice is; and it’s partly because in the 10 or 15 years since they last legislated, the world has changed. I think we can understand that legislation is needed, and if you start from that assumption, it’s going to feel like a much better option to try to improve what has been proposed rather than try to oppose it wholesale.

The other thing is that Labour themselves are quite torn about how to approach this. Part of the Labour party is very concerned about civil liberties and wants to see it do more about these issue, and part of it still very afraid of being painted as being weak on crime, weak on terror and so on, and they want to portray themselves as supporting the secret services as much as possible. So they have a problem.

Then there are some really fundamental issues that we don’t understand. One of the biggest is bulk powers. Are bulk powers justifiable? From our point of view it’s very hard

to see how something that’s entirely indiscriminate could be viewed as proportionate. But we haven’t seen it tested in the courts in quite these terms, and of course the opposing argument is kind of “How are we meant to do this? How are we meant to achieve the results we need if we aren’t sweeping it all up?”

So Labour are trying to push for a review of the powers to see whether they’re justified or not. They’re trying to in a sense get somebody else to tell them whether this is legitimate or not.

**LV** **So what could the effects of the IP bill be for the average law abiding citizen who doesn’t have anything to hide?**

**JK:** The thing you have to remember as an individual is that... GCHQ are literally scoring everybody on the basis of how much risk they think we are. You can choose to ignore it, but the fact is that we are all being evaluated. They’re piecing together information per individual, trying to correlate it with other information about individuals, then deciding whether we appear to be close to certain sorts of patterns.

Those of us who do match or seem close to certain patterns then get looked at more closely. But the fact is that machines are evaluating all of us.



Before joining ORG, Jim was the external communications chap at the Green Party.

GCHQ's argument is that this process of evaluation, correlation, matching of data doesn't matter, because it's only machines that are doing this, and none of this is being looked at by an individual. But it doesn't make me very comfortable that my personal internet activities are being assessed for how suspicious they look. That places me under suspicion; it places you under suspicion, it places your readers under suspicion.

Does using *Tor* place you more under suspicion? It kind of does, it's very likely to. Is using a VPN likely to give you a couple more suspicion points? Yes. Does moving around in certain parts of town put you at greater suspicion? Yes. Each of these things, your flight records, passport history, whatever it happens to be, all these things build a profile. And that means that sometimes some people are going to come and start asking questions. Should I read this website that provides information about extremists? Maybe not – maybe I don't want the police or GCHQ to be knowing that about me. Even as an individual there are serious concerns that people's behaviour online changes. Effectively your free expression, your right to impart and receive information, it is limited because of your fear of surveillance. We don't know the full effect of that yet, but it is inevitably

going to make people that little bit more conformist and that little but less risk-taking.

What happens with the journalist who finds it harder to persuade a whistleblower to hand evidence over? Or what happens with an individual who needs legal protection and that legal protection is known by the authorities; do they not seek legal advice? Those sorts of things are bad for us all. If the rule of law, or the ability of journalists to do their work, is limited, then the result is that we have a less democratic, less free society.

If somebody ends up in jail a little bit longer, because they've been afraid to talk to a lawyer, that undermines the rule of law for all of us. It changes our society. I think those are the reasons we try to restrain surveillance; we don't try to just say "looks, we'll deal with it when it goes wrong", which is kind of the approach that the government has when they say they'll fix it with oversight and checks and balances. You limit surveillance because you don't want everyone to feel like they're under surveillance. That's wrong.

**LV** Fair enough.

**JK:** It's not hard it is, talking to people who already agree? The problem is that the counterview is that the government has a duty to defend

people from crime, terrorism, and that individuals have a right to life. That's also a powerful argument. That's when you end up saying that you have to have some surveillance. And that's why we end up in this horrible area around the techniques that GCHQ use that really work. Because you have to ask whether they are (as GCHQ claim) the only way to reach the conclusions that they're reaching, the only way to develop these and the only way to get to these people.

**LV** We borrowed a similar point from Bruce Schneier a couple

You don't want everyone to feel like they're under surveillance. That's wrong

**of issues ago. Even with a filter that's 99.99% accurate, if you run that on 100m people that's a huge number of false positives.**

**JK:** You've got a false positives problem, but you've also got a huge need to sift everything. If your algorithms and sorting processes requires each and everybody's metadata to be sorted and then to find ways to assess all of that metadata, and to make it searchable, so you're

creating a huge amount of machinery just to deal with irrelevant information.

Maybe it's more easily solvable that I'm giving credit for, but at the very least you've got to question that use of resources.

**LV Did you do the Putin posters with the slogans over the top of them?**

**JK:** We did that with the other Don't Spy On Us groups – we raised about £15,000 to help publish those and we also got some grants in to do that. And we worked with an ad agency who wishes to remain anonymous. The little bit I did, apart from a bit of logistics, was to help them find a CC image they could use.

**LV I was wondering where that photo came from...**

**JK:** It was kindly published by the Kremlin. So if you look on the posters they all say "Image CC Kremlin.ru".

The big challenge is trying to get the public interested. I don't think it's that the public is not interested; it's just that there are so many competing problems an the EU referendum right now is overshadowing literally everything. We did the Putin posters to remind people that this issue is still there, but what you actually need is the politicians arguing it out and explaining to the public why they're taking certain positions. Until that happens, the issue isn't a real decision for people. It's just some

vague thing that's going on. It's the point at which Theresa May [the Home Secretary] stands up in parliament and says "You've got to vote for this" and someone from the other side stands up and says "what you're proposing is totalitarian". That is the point that public start to understand that this is really happening and they need to take a side.

**LV I want to ask you about Myles Jackman [who has recently joined the ORG as legal director]. How's he getting on?**

**JK:** Miles is great. He's very interesting because he's been a campaign lawyer. He thinks like a campaigner, he's someone who wants to see the law change and is capable of pinpointing the problems when they go wrong for individuals. I think he'll be a very effective member of our team.

More widely for ORG I think the legal dimension to our work is going to become more important over time, because it's one thing to argue these things out with bureaucrats who've got an excuse for everything, but if we are prepared to take these things to court that will remind politicians that if they step over the limit in these areas then they aren't going to get their way.

In this area there's been a sort of assumption that because digital is new that politicians have a kind of *carte blanche* to do whatever they like and can define the boundaries as they like. And that ignores the fact that a lot of

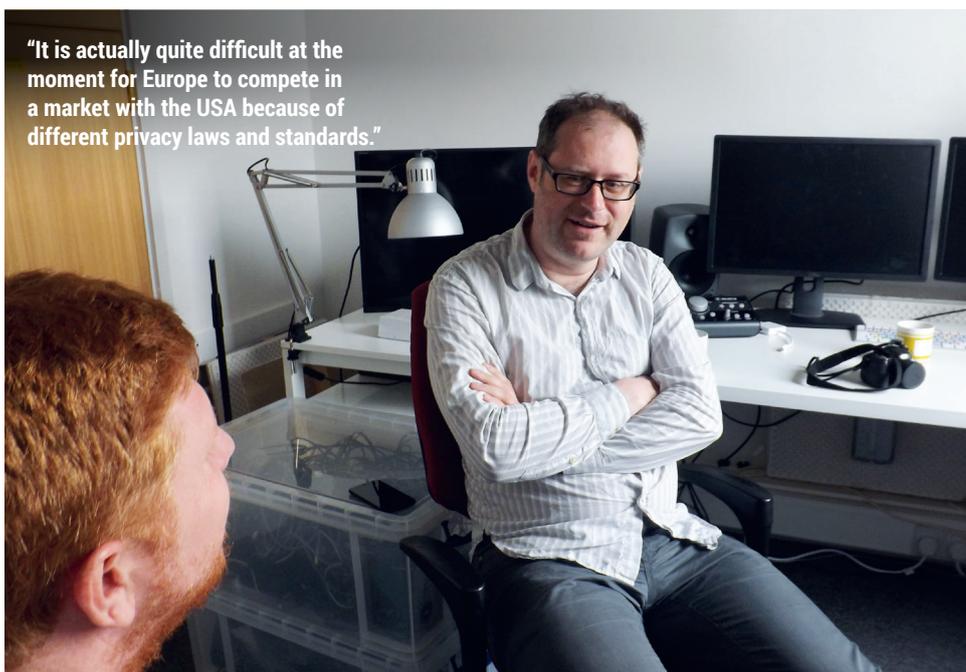
these principles have been debated in the past, and the limit online has to be the same as offline. We have the same privacy rights, we have the same free expression rights.

The dominant metaphor in the mind of a lot of politicians is the internet as publisher, so the internet should behave itself like a newspaper would, or the BBC should. So they expect the individuals online to behave as though they were a BBC journalist; not saying outrageous things, not speaking as they would in public... But at the same time, most people are talking pretty much as they would to their friends, so the metaphor at work in most people's minds when they say and do things online is that of common everyday speech, not editorialised content, and that's quite hard for politicians to grasp. You see that also in the police when they react to content that is offensive.

**LV The chap who was arrested for having a Nazi pug, for example.**

**JK:** The Hitler pug video is puerile, and I'm not going to claim that it's a great example of tasteful humour or anything like that, but the question really has to be if this man is inciting racial hatred, why isn't he being prosecuted under racial hatred laws? The reason is, I think, because the police reckon they wouldn't get away with it. He's very clear about his intentions; he says in the video that he's trying to annoy his girlfriend and that he's not actually advocating Hitlerian politics, and therefore a court would find it difficult to convict as an incitement to racial hatred. So they threatened him with offences under the Communications Act, because 'grossly offensive' is essentially an 'I don't like it very much' test that drags in anything where you can get a lot of people to shout 'I don't like this.' That's not a sufficient test for prosecuting someone.

That doesn't mean you have to like the video; but just because I think what he did wasn't an appropriate thing to do, doesn't mean that I have the right to tell him he has to go to jail. But of course for politicians a lot of the time it's easier to tap into the offence that is generated rather than try to work out whether the lines are properly drawn. I guess that's why we have courts and it's why we have ORG. 



**"It is actually quite difficult at the moment for Europe to compete in a market with the USA because of different privacy laws and standards."**

# BUY LINUXVOICE MUGS AND T-SHIRTS!



[shop.linuxvoice.com](http://shop.linuxvoice.com)

# REVIEWS

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



**Andrew Gregory**  
Is grateful to the brewers of Hebden Bridge for this month's creativity juice.

There's a bit of these running through this month's reviews: neo-Luddism. We've shunned the latest AAA first-person shooters to review *Lumo*. It's simple, it's wonderful and we like it a lot.

In the world of distributions there are all sorts of advanced features, but we've chosen to review one that has simplicity as its *raison d'être* – Tail, which makes it simple to use the Tor anonymising software. And Ben's been playing with Ghost, a blogging platform that trades on its simplicity.

### Simplicity is a feature

The reason we like simplicity isn't that we're lazy or thick: we like it because we've got better things to do than learn a whole new CMS when all we want to do is put up some allotment photographs. We don't want to have to learn a new control system when the WASD keys and Space bar can give us immediate feedback. And frankly, if someone's gone to the bother of configuring the *Tor Browser* for us, why on earth would you bother to do it for yourself? Immediacy is attractive and rewarding.  
[andrew@linuxvoice.com](mailto:andrew@linuxvoice.com)

## On test this issue . . .



**Tails** 42  
*Tor* is essential for maintaining your privacy – and Tails helps you set it up. Simple.



**Krita** 43  
New features and even more polish make KDE's drawing app a must-try for artists.



**Ghost** 44  
Bring the stark elegance of Adrian Frutiger to your online musings with this blogging engine.



**Lumo** 45  
The isometric gameplay of the 1980s coupled with the graphics performance of today. Noiiiice!

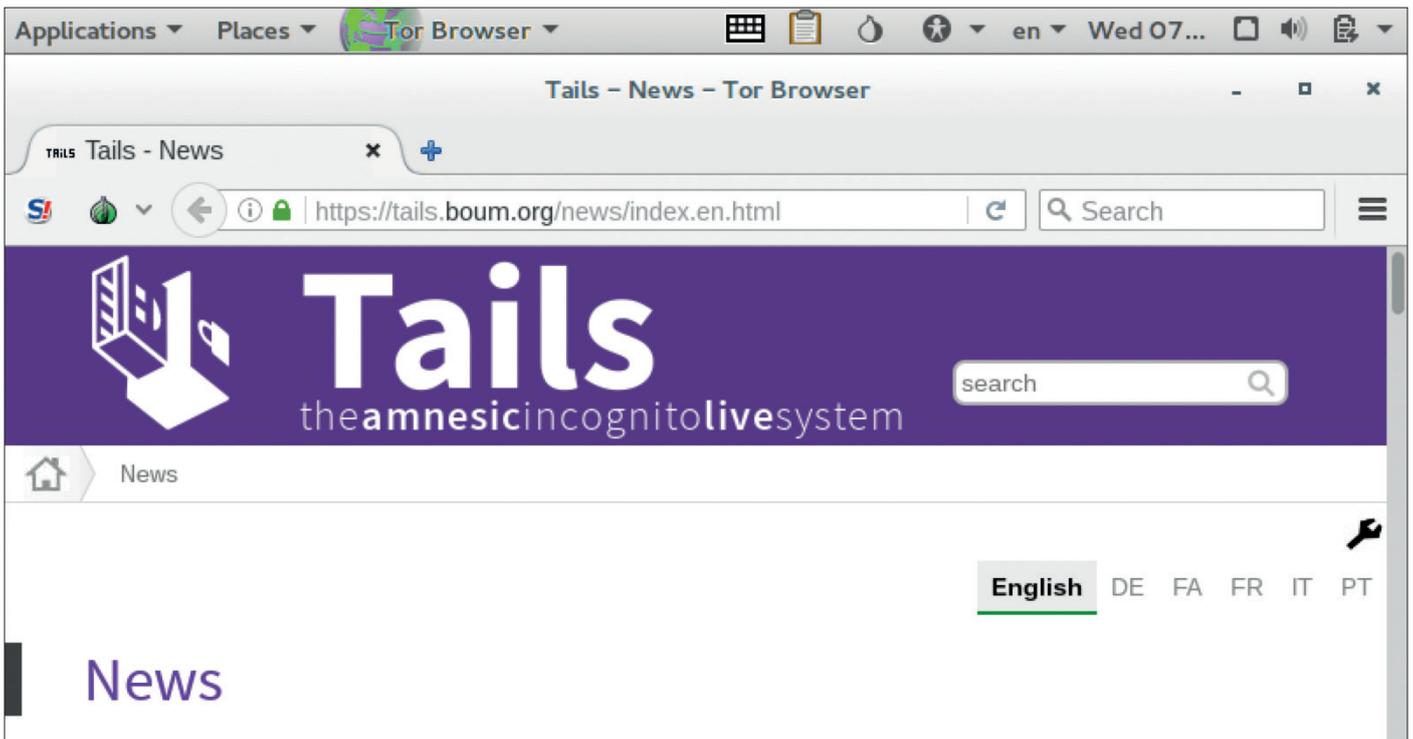
## Group test and books



**Boooooooooooooooooooooo!!!!** 48  
Presumably privacy advice for girls is the same as it is for boys. Likewise the fundamentals of Bitcoin don't change according to your gender.



**Group test – beginner distros** 50  
Emancipate yourself from mental slavery, put Windows down and install one of these fine beginner-friendly distributions.



# Tails 2.4

Browse the web completely anonymously. Well, sort-of.

**Web** <https://tails.boum.org>  
**Platforms** x86  
**Price** Free

Is there such a thing as total anonymity on the internet? Well, if you're willing to travel by foot to a foreign country with a stolen laptop, hack someone's Wi-Fi and do you work that way – quite possibly. But for most people, *Tor* provides a certain level of anonymity by routing your internet traffic through a complex network of servers, making it difficult for the end resource that you access to determine where the original request came from.

Tails (The Amnesic Incognito Live System) is a Debian-based live distro that includes everything you need to get started with *Tor* out of the box. Just boot it up and start browsing – as simple as that. If you boot

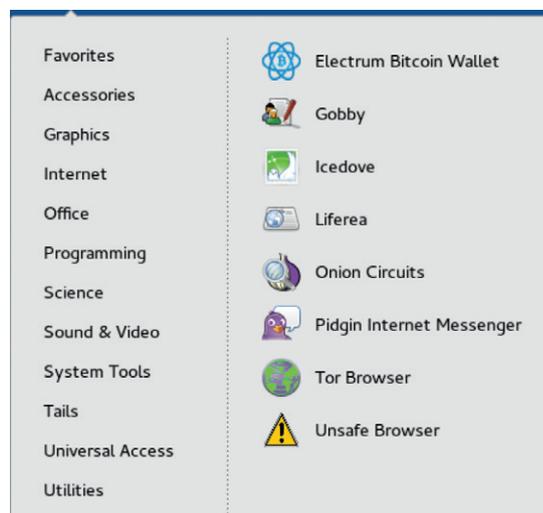
Tails in *VirtualBox*, it complains that your anonymity could be reduced (ie the virtual machine could still be monitoring your keypresses). So for maximum anonymity it's best to run it on native hardware.

Along with the *Tor Browser* – a modification of *Firefox* – Tails includes other software in its 1GB DVD download ISO, such as the *Pidgin* instant messenger, *Icedove* (a rebranded *Thunderbird*), *Gobby* (a document collaboration tool) and other useful bits and pieces. Yes, this all bulks up the download size of the distro, but we appreciate having a versatile toolset with all network traffic going through *Tor*.

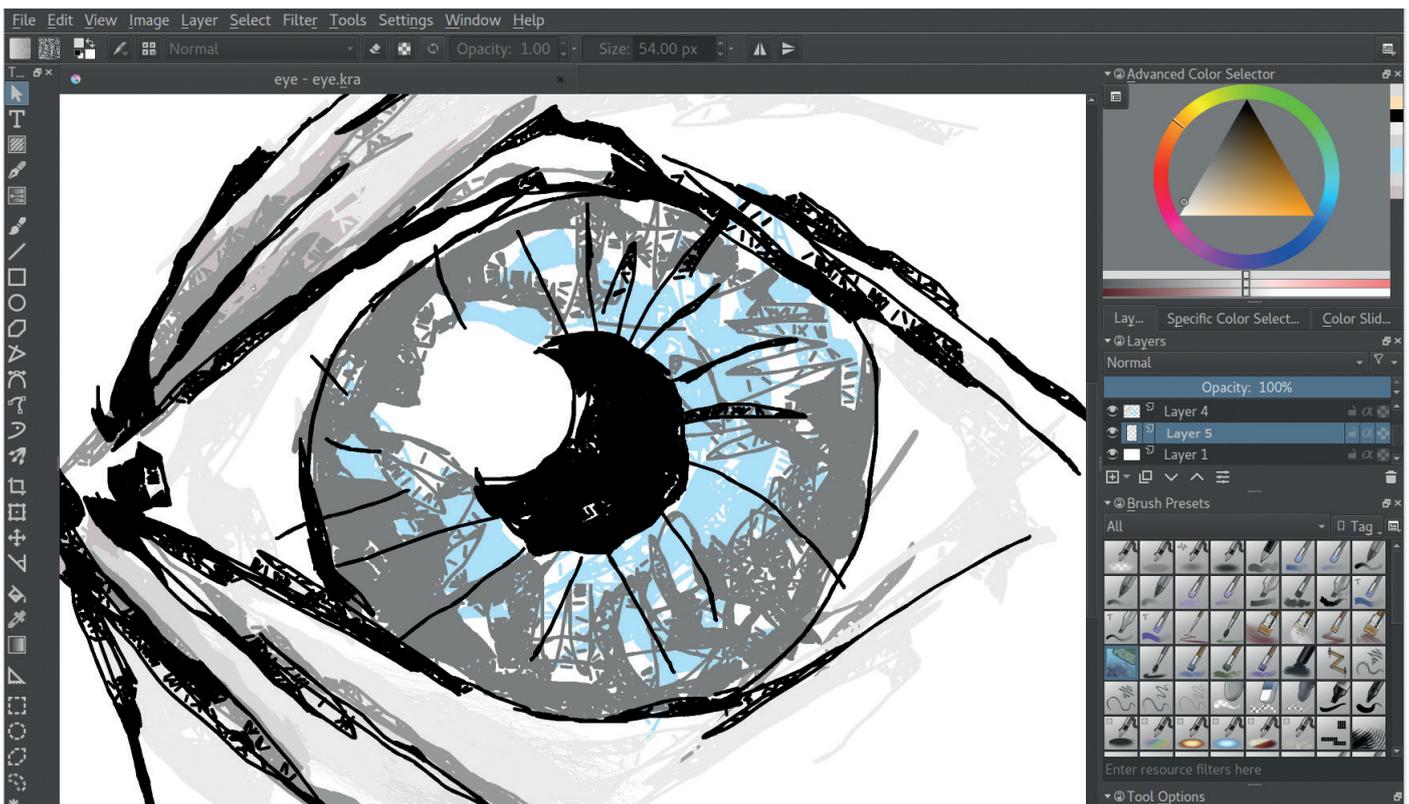
Tails 2.4 features updates for all its major software components, and the *Gnome Tweak Tool* has been removed. This is a controversial change for some, given how non-configurable *Gnome 3* is; on the other hand, we understand that it makes support much easier when end users can't pull apart their desktop beyond recognition. Another change is the removal of **#tails** on IRC as a method of communication, as the Tails team recommends using *XMPP* chat instead.

In all, it's a solid upgrade for the distro and we give credit to the Tails team for maintaining its focus and consistently delivering a good experience. It doesn't guarantee 100% anonymity, but it's a big step in the right direction. 🇵🇸

In a world where government (and corporate) spying is everywhere, Tails is a valuable tool.



While the *Tor Browser* is the main feature in Tails, plenty of other software is included as well.



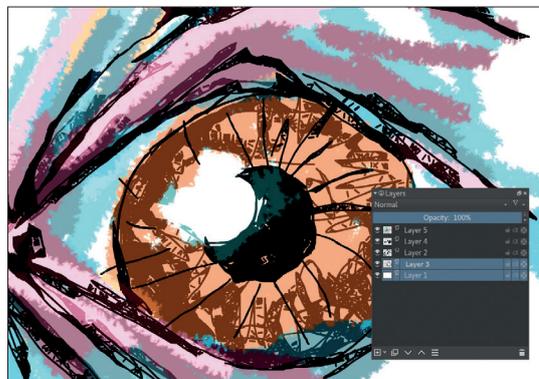
# Krita 3.0

**Graham Morrison** finds his own version of the adult colouring book phenomenon.

**B**efore we look at what's inside this major update to our favourite drawing tool, it's worth looking at what's been going on outside the code. In particular, we think the *Krita* team is doing an amazing job at managing the project, both by getting great support for its crowdfunding efforts, and via the publicity it generates through its website and media channels. We highly recommend our readers take a look at some of the time-lapse design creation videos, example files, brush presets and tutorials, as they provide a brilliant overview of what *Krita* is capable of.

## Let's get squiggling!

Animation alone is worthy of the 3.0 release moniker, literally adding an entire new dimension to *Krita*. With the new animation and timeline 'docker' panes, you can easily start making your drawings move, adding one frame after another and using onion skin overlays to preview previous and next frames. It reminds us of the ancient *Deluxe Paint* on the Amiga, and while *Krita's* implementation is obviously capable of serious results, it's just as fun. It's probably a good thing that the OpenGL 3.0 (and *Qt 5!*) performance enhancements have made it into this release too, as creating animations requires a lot of fast switching between both rasterised and vector layers, but we experienced great performance and no stability issues with the latest release on a modest 2.2GHz



**Web** <https://krita.org/>  
**Developer** The Krita team  
**Licence** GPLv2+

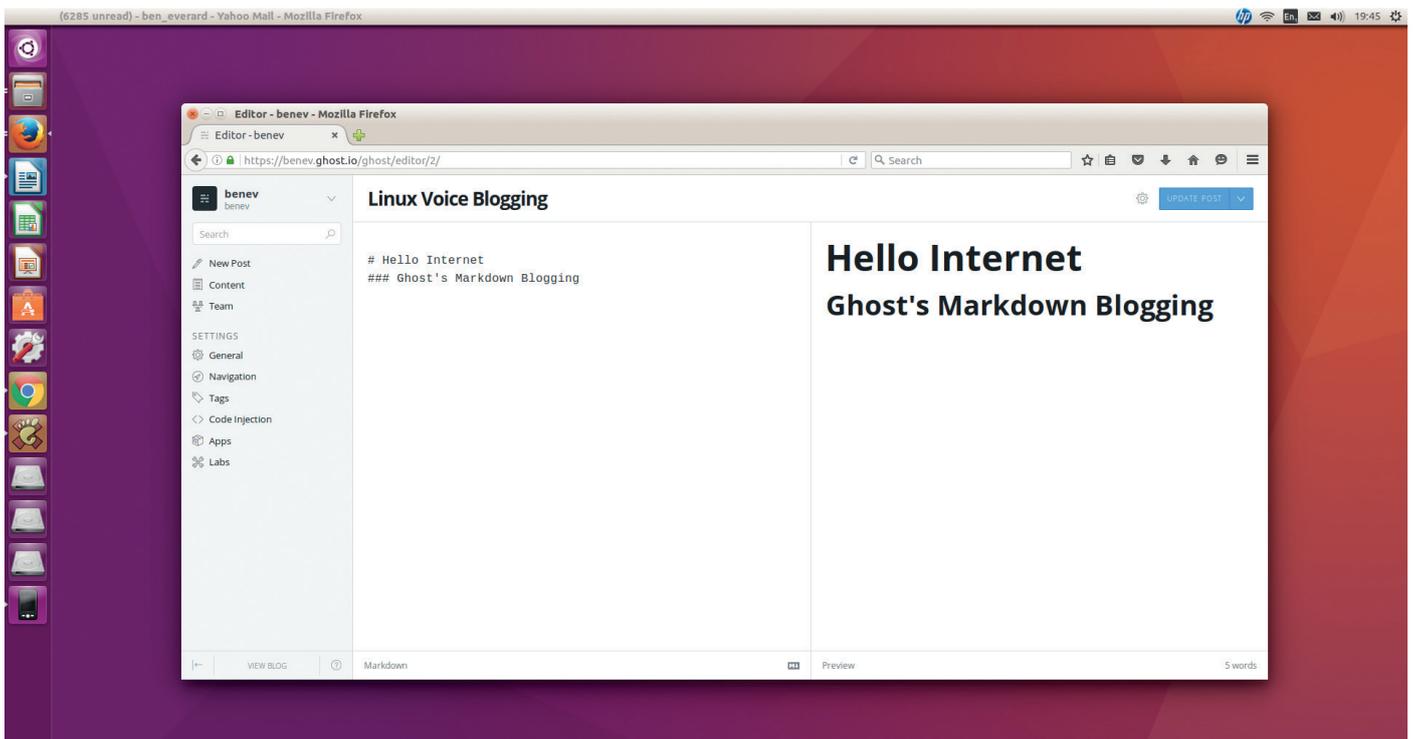
The new layer UI and multi-selection makes multi-layer blender a very creative and rewarding process.

Intel CPU with 8GB of RAM, Intel graphics and a very high resolution display. Everything behaved brilliantly.

This release is the result of a year's work, and of the great many other additions, our favourites include the grids and guides overhaul – guides are now saved with your work and the unified pane makes them easy to access – the ability to move and manipulate more than one layer at once and the new popup palette. The new update, thanks to its speed and design, feels like a genuine professional application capable of stunning results in the right hands. 📺

An absolutely brilliant drawing app, whether you're a beginner or a professional.





# Ghost 0.8

There are ghouls on the web, but **Ben Everard** ain't afraid of no Ghost.

Web [ghost.org](https://ghost.org)  
 Developer Ghost Foundation  
 Licence MIT

**G**host has all the features you'd expect from a blogging platform – you can create posts, manage them, and publish them on the web. Many people can work together to create a single blog with each publishing under their own name. The thing that sets *Ghost* apart from the other options is the user experience. Blogs are written in Markdown, and there's a live preview of the output on the right-hand side of the screen. The interface is uncluttered, and everything is stripped away to force you to focus on what's important – the content. The *Ghost* software processes the Markdown and combines it with your selected theme to produce a great-looking blog.

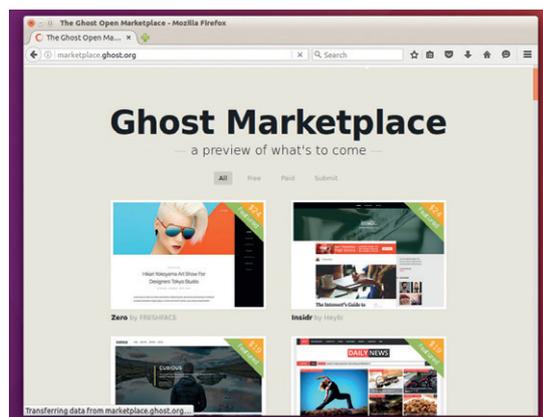
There are two options for running *Ghost* – you can host it yourself (the software is open source) or you can use the hosted option at [ghost.io](https://ghost.io). Setting the

software up is a little more involved than many web apps because it uses Node.js and has to be routed through a proxy (such as *Nginx*). There are detailed instructions on the project's web page (<https://support.ghost.org/developers>). Because of the technologies used, you'll hit some slight complications if you plan on running *Ghost* on the same box you use for *Apache/PHP* web apps.

The hosted options start from \$19 per month for up to 25,000 page views. The cost means that *Ghost* is more suited to professional writing than someone looking to write the occasional piece about a hobby.

When productivity is important, ease of use is a feature. It's difficult to make an ugly post on *Ghost* (when compared to HTML-based blogs that give you enough flexibility to create horrid-looking pages). When blogging is something that you tend to put off, these two factors may make you more likely to actually post something.

Some blogging platforms are flexible to the point that they're really content management systems that can be bent into any website. *Ghost* is not one of these – it's a blogging platform for blogging, and by focusing on one thing, it performs the task excellently. The cost might be a little high for hobbyist bloggers, but for professionals, *Ghost* has a lot to offer. 



The marketplace contains a bewildering array of themes (both free and paid-for) to suit almost any taste.

A blog platform for bloggers, but the hosted options are pricey.





# Lumo

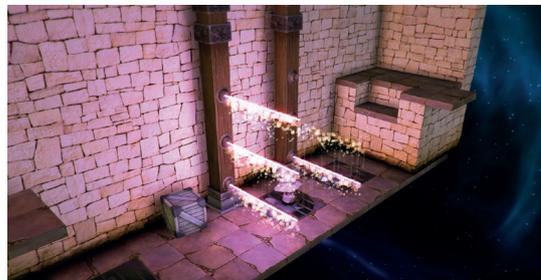
**Graham Morrison falls head over heels in love (again) with 2.5 dimensions.**

**L**umo is an isometric puzzle game. If that means anything to you, you're going to love *Lumo*. It brilliantly recreates the challenges and addictive gameplay of its forebears, rendered in a modern graphics engine running from Steam on your Linux box. If 'isometric puzzler' means nothing to you, you should note that *Lumo* isn't really isometric, nor much of a puzzler. Isometric is a reference to the way games designers built the graphics for their games back in the 1980s, placing pixels at the north west, north east, south east and south west points adjacent to another pixel. This technique limited what would now be called jaggies or aliasing, the stepped appearance of a line when it's drawn across a low resolution, and the arrival of these games on home computers caused a revolution back in 1984. 256 × 192 pixels never looked the same after Ultimate Play The Game released *Knight Lore* for the ZX Spectrum.

## Isometric alienation

*Lumo* comes with a huge chunk of nostalgia, and it does a skillful job at weaving in references from the 80s into its gameplay, from Oliver Frey to Shahid Ahmad, and retains an authentically British *Bedrooms to Billions* aesthetic (a great film if you've not seen it).

But more importantly, *Lumo* revisits a gameplay mechanic that's been lost in the era of 25GB game patches and tenth generation GPUs. This mechanic



**Web** <http://triple-eh.tumblr.com>  
**Developer** Gareth Noyce/Triple Eh Games  
**Price** £14.99 (Steam)

Controls can be configured to find the most natural fit for the view on-screen, though they never make it easy.

comes from the limitations of the display, and the language of the isometric view is a fundamental part of the challenge. Without any instructions or prompting, the player is able to make sense of their surroundings and work their way from one room to the next. There are hundreds of rooms grouped into levels, and getting through the game is going to be a serious quest. Success depends upon precision timing and enough mental dexterity to transpose the on-screen angles into keyboard or game controller movements, sometimes reversed, always under pressure. It's this unforgiving environment that makes the game so compelling and utterly addictive. We loved it. Just like its 80s counterparts. 🎮

Two thumbs up. Inane grin.



# GAMING ON LINUX

The tastiest brain candy to relax those tired neurons



## STAR TREKKIN'



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

Following last month's coverage of how the Vive VR is yet to have Linux support, the first steps to this being amended have been taken, albeit by crafty community workarounds rather than by Valve, which plans to add support at a later date. Those who bought a Vive to use with a Windows dual-boot should check out this guide (<http://tinyurl.com/z4qtgv7>) on how to get it up and running. For those looking to buy a Vive to use exclusively on Linux, it's best to wait for official Linux support.

Something briefly touched upon in Distrohopper last month was progress with running Linux on a PS4. These efforts have now begun to seriously pay off, and an install of Arch Linux on a PS4 has been able to run *Steam* in Big Picture mode. To make this feat even more impressive, the modder responsible also managed to install Radeon drivers to make full use of the console's AMD graphics. As a demonstration, a video has been making the rounds of the hit indie game *Bastion* running natively on the hardware. Needless to say, now with far more native Linux games around, the possibilities for console Linux distros are far more exciting.

On the development side of things, the *Unity Engine* is to add Vulkan support very soon, adding a major advance to one of the most widely-used game engines out there. The developers are also working on some SDL improvements, which should allow for Wayland and Mir support.

## Stellaris

Intergalactic grand strategy.

Web <http://store.steampowered.com/app/281990>  
Price £34.99

This game got a lot of people excited when it was first announced – it's reminiscent of *Master of Orion* and is brought to us by the people behind *Crusader Kings* and *Europa Universalis* (two of the finest strategy games on Linux), so it should be a seemed like a match made in heaven.

*Stellaris* certainly does a lot right. The familiar interface, ability to create and customise galactic empires, a sensible non-Hollywood approach to science, a good mix of well executed mechanics and story events that provide intrigue and mystery to the galaxy all serve to provide an extremely polished game. On top of this, *Stellaris* does well to remain grounded in sticking conventionally understood concepts like terraforming and FTL travel, instead of getting into the outlandish realms of force crystals and magical powers, which can break immersion if not done right and make decision-making arbitrary in a strategy game.

Where the game falls short is with a problem that is all-too-common in strategy games, and that's in the late game. After an incredible exploration phase in the early game filled with



The game's high level of detail enables us to see space battles and planets from up close.

small quests and the excitement of discovering neighbouring empires, followed by a strong mid-game with the imminent threat of war and strong factions, the game becomes a bit of a slog in the later stages and the new technologies and mechanics steadily unlocked throughout the game do little to remedy this.

### Bigger and better

Nevertheless, strategy games of the last 10 or so years have rarely been perfect from the start, often improving greatly with expansions. Fortunately, a tonne of free content has been announced for the coming months. *Stellaris* is still certainly worth getting now, with the knowledge that it will get even better over time.



In *Stellaris*, the Fermi paradox has been answered as the Milky Way is densely inhabited.

# Shadwen

A stylish medieval stealth game.

Web <http://store.steampowered.com/app/425210>  
Price £12.99

**S**hadwen is a stealth game that breaks a few of the genre's conventions. Other than the female protagonist and companion, the game also makes use of time pausing and rewinding. These elements mix things up a bit and make the game feel far more stealth-like than in those where the player can be caught repeatedly. Similarly, giving the choice between killing the guards or

simply evading them using craftable tranquilisers and traps adds a lot of player choice and replay value.

The story holds it all together nicely and the inclusion of Lily – a young girl rescued by Shadwen – adds more dimensions to a single-objective plot. Unfortunately, *Shadwen* feels wonky in areas with some weird physics and collision detection at times. Similarly, at the time of writing, controller support is completely absent on Linux, which is a shame since this kind of game greatly benefits from it.



Join Shadwen on her mission to kill the King while jumping about everywhere.

# The Mims Beginning

A contemporary alternative to Populous.

Web <http://store.steampowered.com/app/337820>  
Price £12.99

**A**fter some time in Early Access, *The Mims Beginning* has now seen a full release, and promises the ability to “re-live the golden age of god games”. In *The Mims*, the player is tasked with overseeing a group of aliens who have settled on floating islands in space, attending to their needs and ensuring their survival. Both the premise and the graphical style is reminiscent of this golden age, while the added benefit of hindsight ensures that the best aspects are taken from these classics.

The game has pretty much every mechanic you'd expect, a decent story mode, lots of buildings and a good interface to manage it all. There's lots to do, so expect to get many hours out of it



*The Mims* has a classic feel while using modern graphics to its advantage.

though the game doesn't have a steep learning curve.

One very unusual omission in the current build of the game is the lack of a save feature during missions, and while the rest of the game is polished and bug-free, this is something that really should have made its way into the final version. Aside from this, *The Mims* delivers what it promises.

## ALSO RELEASED...



### Overfall

This RPG with turn-based combat has it all: a strong story, pleasing art and addictive gameplay. What makes *Overfall* unique is the way in which the story is told, having *Rogue*-like permadeath elements, it occurs through its multiple runs rather than in a linear manner. It *Overfall* provides the fleshed-out world and characters of a traditional RPG.

<http://store.steampowered.com/app/402310>



### Blueprint Tycoon

Fresh out of Early Access, *Blueprint Tycoon* is a game which – despite the aesthetics – has very little to do with blueprints. It can be seen as more of a business sim game, where the objective is to gather raw materials and transform them into goods to be sold at a profit. It's pretty challenging, though also rather addictive, and involves using meticulous planning to achieve optimum efficiency.

<http://store.steampowered.com/app/454060>



### Lumo

*Lumo* brings back isometric platforming after a long absence, but now rendered in real 3D rather than hampered by the graphical limitations of the 1990s. This does a lot to offer extra charm to its already pretty – though minimalist – levels and character, and fuses modern lighting effects and the like with some classic gameplay. There are also plenty of fun minigames and secrets in this hugely enjoyable game (see our review p45).

<http://store.steampowered.com/app/345480>

# The Smart Girl's Guide To Privacy

**Ben Everard** is neither smart nor a girl, but he does protect his privacy.

**Author** Violet Blue  
**Publisher** Digita Publications Privacy  
**Price** £3.59  
**ISBN** None

**V**iolet Blue, blogger at Tiny Nibbles: Open Source Sex, isn't a typical privacy advocate. Of all her publications, this is the only one that could be considered safe for work (the others are all of an intimate nature). As a woman who speaks about sexuality online, she has had to learn how to stay safe on the internet.

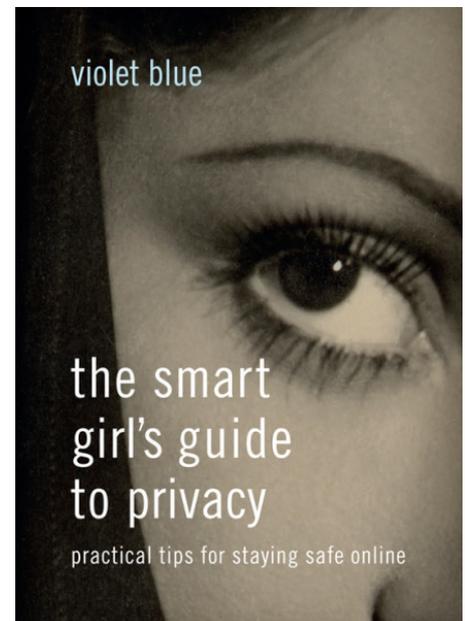
Keeping our private information private can be difficult in the digital world, but it's a challenge that we should all take seriously. *The Smart Girl's Guide To Privacy* is a great guide for women who usually find privacy advice too technical or dry to digest, and also gives real-world examples of what can go wrong if you don't take necessary precautions. *The Smart Girl's Guide To Privacy* is both practical and realistic. It accepts that

women want to have a social life online, and this may include things like internet dating. Rather than keeping your privacy by turning off your computer, this book goes through the particular risks and potential mitigations for being an active member of the online community. It gives advice for what to do should you lose control of information such as through doxxing or revenge porn.

This book is an important read if you're a woman online, and it's essential reading if you're a woman online who (for whatever reason) is particularly at risk from harassment.

The best source of online privacy advice for women we know.

★★★★★



Smart, sensible and accessible advice for staying safe online.

# Bitcoin For The Befuddled

Money money money, must be funny, in a Bitcoin world.

**Author** Conrad Barski and Chris Wilmer  
**Publisher** No Starch Press;  
**Price** £16.50  
**ISBN** 978-1593275730

**L**et's face it: Bitcoin is complicated in a huge number of ways, and if it doesn't befuddle you a bit, you're either an expert or not really thinking about it. There's the underlying processes of how a decentralised digital currency can even work, there's the practicalities of how to buy and spend it and there's the philosophical reasons for why we'd want such a thing in the first place.

All this is without getting started on what money actually is. *Bitcoin For The Befuddled* tackles these areas and even goes further to show you what to expect in the future and how to incorporate the currency into your software.

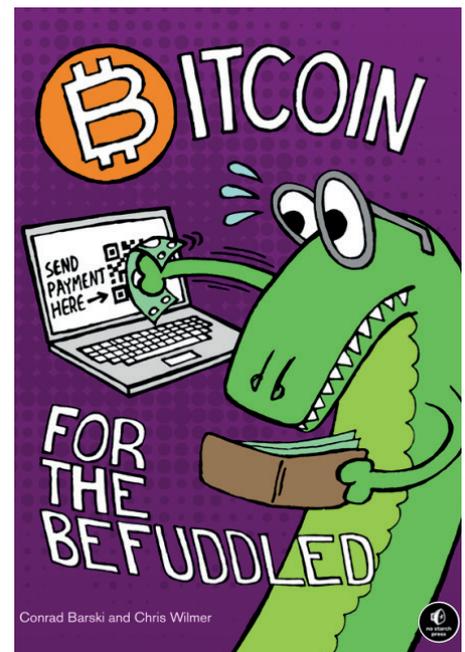
Bitcoin has attracted more than its fair share of controversy over the years, and has had wildly varying prices, but it's weathered

all the storms and seems to be as strong as ever. Bitcoin is fascinating for its own sake, but the principals behind it (such as using a blockchain as a distributed ledger) may well find applications in far more areas. Even if the actual currency of Bitcoin goes away, the technical ideas behind it won't, so it's well worth making the effort to understand it.

*Bitcoin For The Befuddled* is a great read for programmers and other technical people who want to understand what's going on with cryptocurrencies. Non-technical readers won't get quite as much out of it, but should still be able to follow most of the ideas in the book. 📖

A wide-reaching guidebook to the most popular digital currency.

★★★★★



Tired of endlessly mashing money into your screen? Then this book is for you.

# LISTEN TO THE PODCAST

# LINUXVOICE

## WWW.LINUXVOICE.COM



FREE SOFTWARE | FREE SPEECH

# GROUP TEST

Mayank Sharma helps solve the one question put to every geek – what's the best Linux distribution for those just starting their journey into Free Software?

## On test

### Antergos



URL [www.antergos.com](http://www.antergos.com)  
 Licence GPL and others  
 Latest release 2016.04.22  
*What's a rolling release doing in a Group Test of distros for newbies?*

### Deepin



URL [www.deepin.org](http://www.deepin.org)  
 Licence GPL and others  
 Latest release 15.1  
*Are the Chinese as proficient in building distros as they are with everything else?*

### Elementary OS



URL <https://elementary.io>  
 Licence GPL and others  
 Latest release 0.3.2  
*Arguably the most well known newbie distro – does it trump the competition?*

### KaOS



URL <https://kaosx.us>  
 Licence GPL and others  
 Latest release 2016.04  
*KDE for the new users? You gotta be kidding me!*

### Pinguy OS



URL [www.pinguyos.com](http://www.pinguyos.com)  
 Licence GPL and others  
 Latest release 14.04.4-1  
*It looks appealing but is that enough to top the rest?*

### Solus



URL [www.solus-project.com](http://www.solus-project.com)  
 Licence GPL and others  
 Latest release 1.1  
*Is this the solution to end all desktop geekery?*

## Beginner distros

**M**y niece recently got a laptop on her birthday and wondered if she could run Linux on it. For a second I was tempted to suggest one of either Linux Mint or Ubuntu. These distros make it easy to flesh out the installation with plugins to work with all sorts of online services and play all types of content. Ubuntu also works with a number of popular software vendors, and you can rest assured that if there's a Linux-compatible piece of code out there, it'll surely work on Ubuntu. But are these considerations enough? If anything they just make Ubuntu an ideal base for a beginner distro.

A distro that's just easy to use isn't always the best bet for helping new users traverse the open source ocean (see box below). For starters, the average Linux desktop distro

still presents too many choices that can potentially confuse someone who's just starting out. They also look almost too polished and lack the vibrancy and attractiveness that would catch the fancy of a new user. After all, Linux's legacy as a distro for uber geeks or the server room needs to be buried under a thick coat of lively interfaces and user-friendly design.

In this Group Test we'll look at distros designed to cater to the needs (and whims) of the new Linux user. We'll go off-piste and look past the usual list of desktop distros that are often pitched to the new user. Our collection of distros are designed from the ground-up to be attractive and useful to beginners. We'll analyse the unique aspects of these distros and how they help them convince a user to indulge in something new and bold.

A distro that's easy to use isn't always the best bet for helping new users traverse the open source ocean

### What makes a beginner-friendly distro?

It's a mistake to assume that a distribution designed for the average desktop user would be well suited for someone who's just starting out with Linux. A new Linux user needs to be cajoled and handheld through regular everyday computing tasks as they get familiar with the lay of the land.

To begin with, these distributions help users by making choices on their behalf. Linux and FOSS are all about choice, but it'd be a cumbersome exercise for a new user to sift through the myriad desktop

environments before they even get to a download image. These projects also take the time to tweak the components within the distro to make them easier for new users to approach them. Some even modify the code of popular apps to write customised versions of the apps that are designed to be operated by newbies.

No matter how they customise their offering, Linux distributions aimed at the inexperienced user are far easier to get started with and use than your typical desktop distro.

# Hardware requirements

## Do they burden your silicon?

What's the cost of all the glow and glitter you get with these beginner-friendly distros? Do they all need state-of-the-art hardware? The different distros on test here have different hardware requirements. The primary factor is the choice of their desktop environment. Some extend the ease of use by extending the conveniences offered by full-blown desktop environments, while others rely on

custom-built resource-friendly compositing graphical environments. Furthermore, many of these distros bundle some software that you wouldn't normally find installed by default inside a desktop distribution, primarily because of the lack of their intended audiences' ability to find these for themselves. Some of these specialised applications can be quite demanding and may take a toll on your computer.

That said, we should keep in mind that some distros are able to tune down the glitter automatically based on the hardware resources at their disposal, so you might be able to run them on a machine that's been sitting on your desk for a couple of years now. However, to enjoy these distros to the hilt and take full advantage of their beginner-friendly usability, we'll advise you to run them on a well stocked machine.

## Deepin

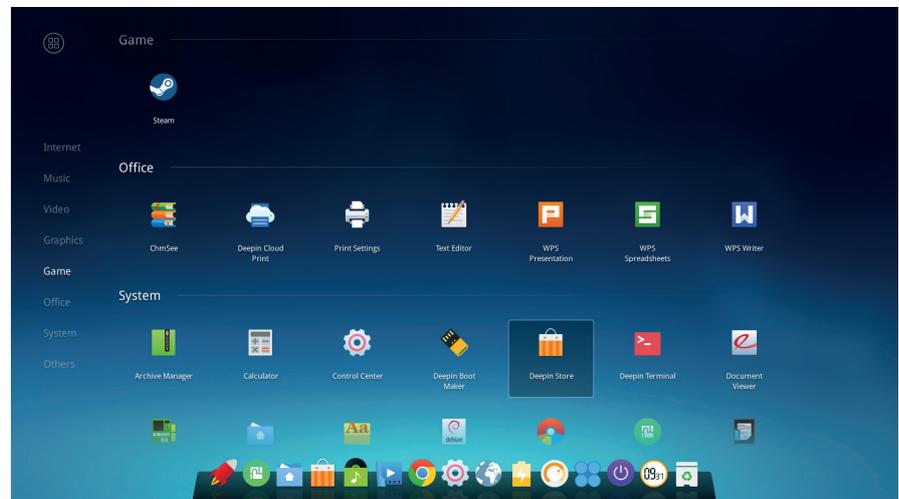
### Is beauty really just skin deep?

Give Deepin a try and you'll be amazed by its reimagined desktop experience. The Debian-based distro has customised all aspects of the distro and that has had a pleasantly positive effect on the user experience.

By default, Deepin's installer will take up all the space on the selected disk, but you can manually configure the partitioning by clicking Expert Mode. After installation you get to witness one of the major factors that sets the distro apart from other Debian derivatives; the distro's home-grown desktop – Deepin Desktop Environment (DDE). It's based on HTML 5 and *WebKit*, and uses a mix of QML and Go for its various components. DDE has a clean and clutter-free interface and tries its best to replicate the usability and aesthetics of Mac OS X.

Besides the desktop itself, notable homebrewed Deepin components include the application launcher, dock and control centre. The launcher is elegant and blazingly fast and it will either display the installed apps in alphabetical order or separate them under categories.

The desktop has hot corners and the top-left corner opens the full-screen application launcher. To search for an application you start typing the letters and the launcher will offer all matching results. The dock at the bottom of the screen has several views/modes. Besides the default stylish Fashion mode, there's the Efficient mode, which places icons on the desktop, as well as the Classic mode, which is similar to the Efficient mode but with smaller icons.



The entire distro is designed with ease of use in mind, and is well-suited for touch devices as well.

At first glance, the Deepin Store appears to be a clone of the Ubuntu Software Centre. However, it's better organised and offers a much neater experience. Besides the desktop apps, the Store also offers several web apps for installation, such as Google Drive, Hangout, Pixlr editor, and more, courtesy of the distro's collaboration with the Intel Crosswalk project.

### Kitchen-sink approach

In addition to the customised desktop, Deepin includes several custom-built apps, such as Deepin Boot Maker for creating a bootable USB, Deepin Music, Deepin Store, and more. Besides the custom apps the distro also bundles several proprietary apps such as *Google Chrome*, the *Steam* client, *WPS Office*, and *CrossOver Linux*.

The Deepin developers have gone to great lengths to make using the distro familiar to those coming from proprietary environments. For instance, removing apps

uses the same parlance as Windows and involves right-clicking an application and selecting the Uninstall option. In another departure from convention, the Control Centre is integrated into the desktop itself, instead of being offered as a distinct application. The bottom-right hot corner opens the pop-out side panel that houses the various settings under different sections. You can manage all aspects of the desktop, from the boot manager to the desktop theme from the controls here. It also helps keep track of and apply any updates. The Control Centre also enables you to access the Deepin Manual as well as the Remote Assistance feature, which uses Chrome Remote Desktop to share your desktop with other Deepin users.

### VERDICT

A well designed and elegant distro with a host of custom apps.

★★★★★

# KaOS

## Chaos?

**K**aOS isn't your typical distro designed for new users: it's designed primarily for people who are at home with the KDE desktop.

What makes KaOS beginner-friendly is its tightly integrated streamlined design. Unlike most other Linux distros that burden users with ample choices, KaOS has consciously decided to keep the options limited. The installable live distro takes a conservative approach to package management, offering only KDE as the desktop, and is available for 64-bit machines with its repositories replete with only x86\_64 packages.

Another aspect that will help new users is the use of the distro-agnostic *Calamares* installer. One of its highlights is the advanced partitioning feature, which offers several partitioning options including the ability to carve out partitions to your liking.

The distro includes all the apps you'd need for everyday desktop tasks.

Continuing with the KDE theme of the distro, there's the *Calligra* office suite, the *Qupzilla* web browser, *Clementine* audio player, *KDE Telepathy* for instant messaging, *Quassel* for IRC, *SMPlayer* and *MPV* media players, and many more including apps to stream YouTube and record the screen and the webcam. To help users flesh out their installer, KaOS uses the *Octopi* graphical application installer, which is a front-end to the powerful *Pacman* package manager. To maintain its tight integration, KaOS doesn't use any upstream repositories, choosing instead to build each of the packages in its three repositories from scratch specifically for KaOS.

### KDE for newbies

KDE is perhaps not the easiest desktop environment to get to grip with, but KaOS is so well packaged that it makes conducting everyday desktop



**KaOS has a simple repository layout that helps newbie wrap their heads around the open source universe.**

business simple and straightforward. Also helping its cause is the new KDE Plasma 5, which is pretty enough to rope in new users who wouldn't even realise that they're using a distro based on one of the geekiest flavours of Linux.

### VERDICT

Its conservative approach and tight integration give it real appeal.

★★★★★

# Antergos

## Enter the dragon.

**H**ere's another Arch-based distro that's developed with simplicity in mind. Antergos provides a fully configured environment that can be used right out of the box. The live CD boots to a customised Gnome-based desktop featuring a dock ripped out from the Activities Overview using the *Dash to Dock* extension.

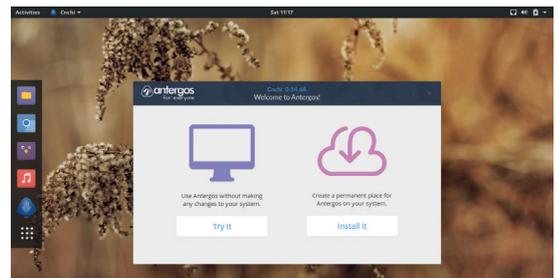
Antergos uses a custom installer called *Cnchi*, which is easy to navigate and use despite its beta status. Its partitioning step also offers several options that will appeal to both new and experienced users.

Despite its goal, Antergos passes on the burden of selecting core components to the user. One screen asks you to select a desktop environment from either Cinnamon, Gnome, KDE, Mate, Openbox or Xfce. In the next you're asked to select optional components from a list that includes *Firefox*, *LibreOffice*, *Steam*, *PlayOnLinux*

and more. The installer then downloads the components you have selected, which might take some time depending on your internet connection.

### Search for applications

Besides the apps offered in the installation screen, if you choose to install the default Gnome desktop, the distro also includes the Gnome Music and Video apps, *Pidgin*, and the *Chromium* browser. Antergos ships with a graphical package manager called *Pamac*. It's easy to navigate, and just like KaOS's *Octopi*, *Pamac* might not be pretty to look at but it gets the job done. However we'd suggest using the search function to find packages instead of relying on the available software categories, which are not always intuitive. For instance, *Firefox* is available under the browser category, while add-ons for the browser are housed under a different category.



**Like KaOS, Antergos will help instill the ways of a rolling release without intimidating a beginner.**

Also, while the email category lists just one client, a search turns up several. However, once you've got it set up, Antergos is a pleasure to use and doesn't pose any debilitating issues that are beyond the grasp of ordinary desktop users.

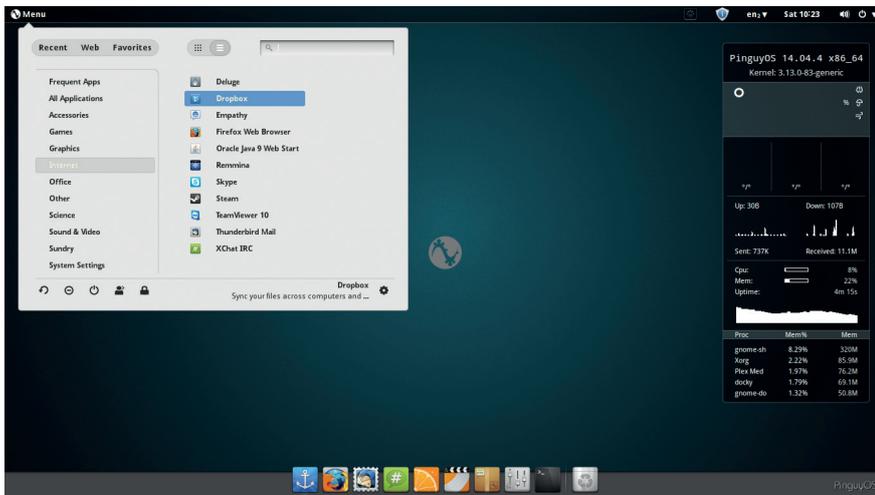
### VERDICT

The software selection screen rules it out for absolute Linux newbies.

★★★★★

# Pinguy OS

The pinup guy.



Pinguy OS also includes *Pinguy Builder*, which is a fork of the popular *Remastersys* tool to help create custom ISO images from your Pinguy OS installation.

The Pinguy OS distro gives you the option to either explore the live environment or jump straight into the installer. The distro is based on the Ubuntu LTS releases and relies on the Ubuntu installer to anchor the distro to your computer.

The distro uses a customised Gnome desktop featuring a semi-transparent panel at the top and a dock at the bottom with a *Conky*-powered applet on the right-hand side of the screen displaying vital information about the computer. The panel houses the *Gno-Menu* extension, which serves as the default application launcher. It contains a categorised list of software and also keeps track of frequently used apps and recently viewed files. You can also bring up the default Gnome 3 Activities application launcher by pressing the Win key. Move the mouse to the left-hand edge of the screen to reveal several home folders.

Pinguy is chock full of general purpose and specialised apps including *LibreOffice*, *OpenShot* video editor, *Boot Repair*, *Devede*, *Empathy*, *Handbrake*, *Shotwell*, and more. For video playback, besides *VLC*, it's also got the *Plex Media* server. The *Clementine* music player also comes equipped with access to over a dozen online music services including Grooveshark, Last.fm, Spotify, Jamendo and SoundCloud. Besides the open source apps, Pinguy OS also includes several popular proprietary ones, such as *Dropbox*, *Skype*, *TeamViewer*,

*Spotify* and *Steam* for Linux. There's also *Wine*, which you can manage with the bundled *PlayOnLinux* front-end. For customisers, the distro includes the Gnome and Ubuntu Tweak Tools.

### Shiny as a new pin

If you need more software, the distro offers the *Ubuntu Software Centre* as well as the *Synaptic* package manager. The package managers are equipped with a large number of repositories, many of which are enabled by default, including those for Linux Mint and Ubuntu. There are PPAs for themes and apps such as *Clementine*, *VLC* and Gnome. Pinguy also includes the *Y PPA Manager* to keep track of all the PPAs.

Pinguy's developer has cherry-picked the components to offer a convenient user experience. The distro uses the *Nemo* file manager along with a bunch of plugins to integrate Dropbox, Samba and the file archiver. The file manager is also equipped with scripts such as *Torrent-Video-Player*, so you can right-click on a **.torrent** file and use the *Torrent-Video-Player* option to stream it without downloading. There's also a script to help organise your downloaded TV shows and movies into folders with cover arts and subtitles.

### VERDICT

An elegant distro that's easy to use but needs a decent machine.

★★★★★

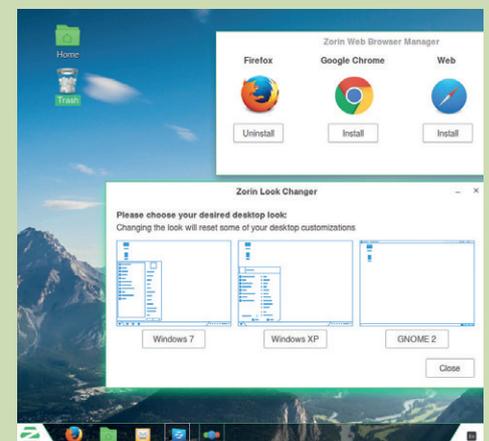
# Attack of the clones

## Missing Windows already?

Zorin OS is an unusual distro that, like the others in this group test, is aimed at easing Windows users into a Linux distro. However, what makes it unique is its Gnome desktop that's modified to resemble Windows 7. Zorin is based on Ubuntu, and its goal is to package the goodness offered by that distro into a system that "anyone can use without learning anything new thanks to its familiar interface."

To that end, the home-brewed *Zorin Look Changer* app lets you customise the distro to match different versions of Windows. The default desktop behaves pretty much like Windows 7 and although it's not an exact replica, it does provide a comfortable experience for users who only have experience with using Windows (you can also use the Look Changer to make the desktop resemble the older Windows XP, if you're feeling nostalgic). Besides the applications to modify its desktop layout, Zorin OS also includes a custom app to easily modify the theme.

The distro is available in four different versions: the Core and Lite editions are free, while the Business and Ultimate flavours cost €8.99 (about £7) and €9.99 (about £8) respectively. The paid versions come with support and a few extra features, such as the option of using interfaces that mimic Mac OS X and Windows 2000. The Core edition includes all the apps you'd find in a normal desktop Linux distribution, while the Lite edition is designed for older machines and the Ultimate edition is chock-full of all kinds of apps and games.



You can use the Zorin web browser manager to pick and install any of the four popular browsers.

# Elementary OS vs Solus

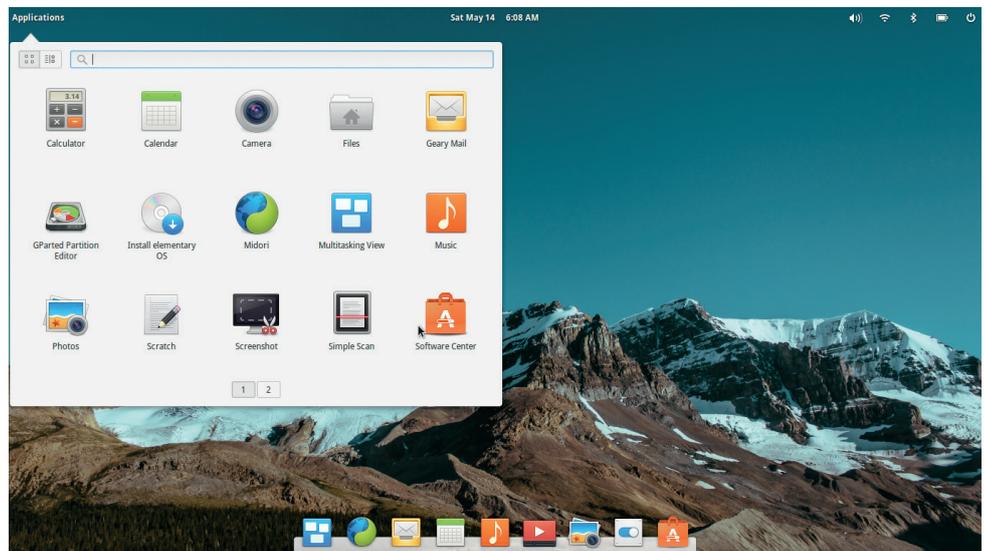
Custom desktop faceoff.

**E**lementary OS has made a name for itself as an elegant and user-friendly distro. Its custom desktop, Pantheon, takes cues from the Mac OS X desktop, and has its own *Mutter*-based window manager called *Gala*. The desktop nicely integrates the various other important elements, such as the *Plank* dock, the top panel (called *Wingpanel*) and the *Slingshot* application launcher, to present a smooth, unified, user experience. Nearly all actions on the desktop are subtly animated.

One of the strangest things about Elementary is that it comes with an unusual set of applications. The distro supplies a number of custom tools, such as the *Geary* mail client, *Scratch* text editor and *Audience* video player, which are designed to assist inexperienced users. However, Elementary OS doesn't offer many apps out of the box and doesn't include proprietary codecs or ship any non-*GTK* apps, which is why it doesn't include the likes of *LibreOffice* and *Firefox*. Instead it uses the *Midori* web browser. For package management the distro uses the Ubuntu's Software Centre and it pulls software from the Ubuntu repositories as well as PPAs of its own.

## Another solution

Unlike some of its peers, Solus is not based on another distro, which gives its developers manoeuvrability to mould all aspects of the distro to their vision, including the user experience.



Elementary OS can be used on older hardware as well because of its low system requirements.

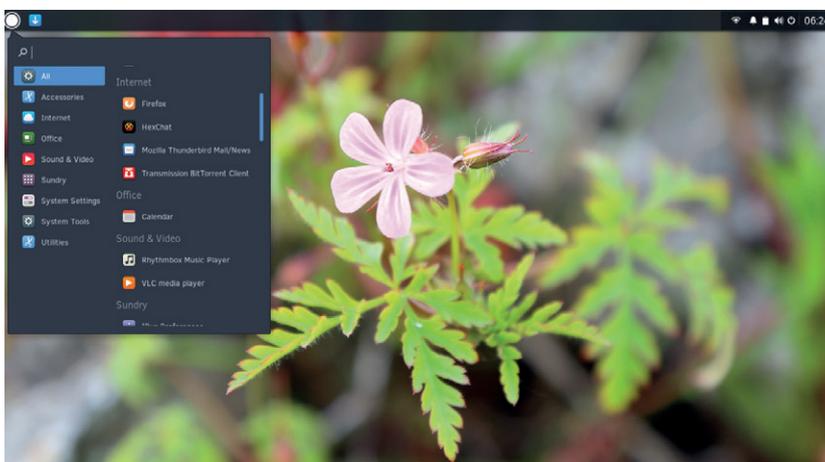
The distro boots into a Live session, and as with most of the core components, its installer too is custom built. While it's fairly intuitive and easy on the eyes, it lacks an automated partitioner. This is something that the developers will have to correct very soon, because asking first-timers to fire up *GParted* and manually craft partitions won't get them very far.

Solus uses a custom desktop called Budgie, which is based on Gnome libraries. The Budgie desktop tries its best to replicate the classic Gnome look and feel while offering several conveniences of the modern Gnome desktop. Budgie's application menu displays a categorised list of apps

and has a search box for finding apps without having to navigate the menus. The one strange aspect of the menu is that it keeps reorganising the apps within the categories depending on how often they are used. While this ensures that the frequently used apps are always at the top of each menu, it might seem confusing to not find the app at their usual places.

Another unique aspect of Solus is the all-in-one applet, notification and customisation centre called *Raven*. This houses all the information in two tabs: all the notifications are tracked in one tab, while the Applets tab displays the calendar, sound volume and sound devices. All items on the panel, including the application menu and clock etc, are applets, and you can place and reposition them.

Solus has plenty of apps for regular desktop use, including *Firefox*, *Thunderbird* and *VLC*, but lacks an office suite and any sort of games. The distro also has a custom package manager that's designed intelligently for inexperienced campaigners with a minimal, unimposing interface.



The Budgie desktop is a wonderful piece of software that blends old and new functionality.

## VERDICT

**ELEMENTARY OS** It's pleasing to the eyes but its defaults aren't the best for a newbie.

★★★★★

**SOLUS** The easy-to-use experience is hindered by the lack of an automated installer.

★★★★★

# OUR VERDICT

## Beginner distros

Recommending a distro to a new Linux user depends strongly on their proficiency and comfort level with the computer. That said, we hesitate to recommend Antergos, because it presents too many choices to the user who is probably not well equipped to make the correct decision. Then there's KaOS, which presents an intriguing option because of its tight integration, but isn't designed specifically for new users. However, those with some mileage under their belt will surely appreciate the consistency of the desktop and the coherent design.

It's a close call between Solus and Elementary OS for a position on the podium. Solus is by no means just another newbie-friendly distro. It's a project with solid foundations and leadership that has a clear vision and experience to realise it. Once the issue with the installer have been ironed out, the distro should be ready to compete with the established players.

Elementary OS has also put in quite an effort into building custom tools and libraries. Everything from

the window manager up to its apps is crafted to adhere to its design principles. We'd recommend it over Solus, but only just.

The top two distros for new Linux users are Pinguy OS and Deepin. Both distros offer a good mix of form and function and their pleasing desktop environments give access to its vast number of applications. However Pinguy's usability comes at a cost. All of its customisations consume a lot of resources, and you'll only be able to enjoy Pinguy OS on a machine that has at least 4GB of RAM.

Deepin too isn't without issues. Designed to be user friendly, the distro is ideal for beginners but doesn't offer much for advanced users. So while its installer enables you to perform all the usual partitioning operations, Deepin doesn't support LVM, and you can't create an encrypted home folder, which is something that most distros support. It's a pretty close battle between Pinguy OS and Deepin but in the end Pinguy's stringent hardware requirements help clinch the deal for Deepin.

Solus OS has solid foundations and leadership that has a clear vision and the experience to realise it

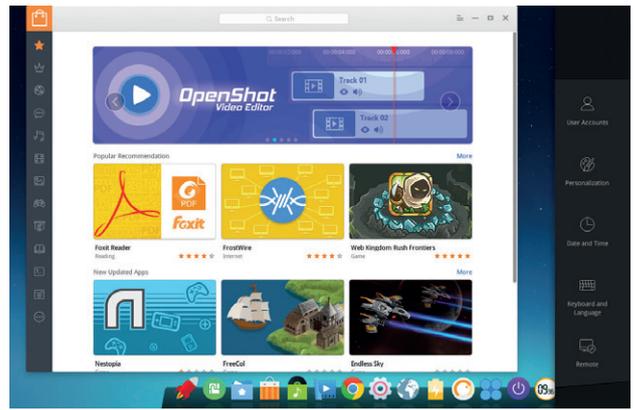
### Other user-friendly distros

There are several other projects that produce distros that are easy to use and can also double up as beginner-friendly distros. There's PCLinuxOS, which began life as a repository for improving a stock Mandrake release and then forked into a distro of its own in 2003. It uses a rolling release model with ISO releases every now and then to assist new users take the plunge.

Another useful distro is Bodhi Linux, whose minimalistic nature is in contrast to the usual approach of cramming the distro with apps. Bodhi uses the Moksha

desktop built atop the lightweight Enlightenment desktop. Like its peers, Bodhi takes the pain out of cumbersome processes like fleshing out the distro by using easy-to-operate custom apps.

You might also want to take at friendly versions of popular mainstream releases. Korora, for example, produces versions based on the official Fedora releases. Then there's the Arch-based Manjaro Linux, which produces several flavours, and the Ubuntu-based Netrunner. These distros lower the learning curve of using their parent distros.



Thanks to its focus on design and aesthetics, Deepin is easily the new benchmark for newbie-friendly distros.

### 1st Deepin

**Killer feature** Elegant-looking custom desktop and tools.  
**URL** [www.deepin.org](http://www.deepin.org)  
 Full of customisations to serve the new user well.

### 2nd PinguyOS

**Killer feature** Customised Gnome desktop and apps galore.  
**URL** <http://pinguyos.com>  
 Designed to pamper the new user, it only narrowly loses out.

### 3rd Elementary

**Killer feature** Pantheon desktop and pure GTK experience.  
**URL** <https://elementary.io>  
 It's aesthetically pleasing, but its default software selection necessitates a visit to the package manager.

### 4th Solus OS

**Killer feature** The Budgie desktop environment.  
**URL** [www.solus-project.com](http://www.solus-project.com)  
 Still early days for the distro that needs to simplify its installer.

### 5th KaOS

**Killer feature** Well integrated KDE experience.  
**URL** <https://kaosx.us>  
 The KDE-centric distro isn't specifically designed for beginners but still does a good job.

### 6th Antergos

**Killer feature** Makes Arch Linux accessible to the masses.  
**URL** [www.antergos.com](http://www.antergos.com)  
 For a newbie-friendly distro, it offers too many choices.

# Subscribe

## shop.linuxvoice.com



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 – £57
- ROW – £60

**DIGITAL SUBSCRIPTION ONLY £38**

Get 114 pages of tutorials, features, interviews and reviews every month

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

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

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

# NEXT MONTH IN LINUX VOICE

ON SALE  
THURSDAY  
21 JULY



## SAVE TIME FOR THE WIN

It's summer – so speed up your computer with our clutch of tips and tricks and spend the time saved doing awesome things.

## EVEN MORE AWESOME!



### Firefox

For so long the darling of Free Software, *Firefox* has had its ups and downs. Is it yesterday's web browser now, or can it take back the web?



OpenZFS

### ZFS

The Ferrari of filesystems has been on our radar for a while now, but we've never been brave enough to take the plunge. Well now we are – let's upgrade!



### Wayland

The graphical revolution has been a long time coming. Now it's time to set up, configure and use Wayland. How is it going to make our lives better than X?

## LINUX VOICE IS BROUGHT TO YOU BY

Editor Graham Morrison  
[graham@linuxvoice.com](mailto:graham@linuxvoice.com)  
Deputy editor Andrew Gregory  
[andrew@linuxvoice.com](mailto:andrew@linuxvoice.com)  
Technical editor Ben Everard  
[ben@linuxvoice.com](mailto:ben@linuxvoice.com)  
Editor at large Mike Saunders  
[mike@linuxvoice.com](mailto:mike@linuxvoice.com)  
Creative director Stacey Black  
[stacey@linuxvoice.com](mailto:stacey@linuxvoice.com)

Editorial consultant Nick Veitch  
[nick@linuxvoice.com](mailto:nick@linuxvoice.com)

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

Printed in the UK by  
Acorn Web Offset Ltd

Disclaimer We accept no liability for any loss of data or damage to your hardware

through the use of advice in this magazine. Experiment with Linux at your own risk! Distributed by Marketforce (UK) Ltd, 2nd Floor, 5 Churchill Place, Canary Wharf, London, E14 5HU  
Tel: +44 (0) 20 3148 3300

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

Copyright Linux is a trademark of Linus Torvalds, and is used with permission. Anything in this magazine may not be reproduced without permission of the editor, until March 2017 when all content (including our images) is re-licensed CC-BY-SA.  
©Linux Voice Ltd 2015  
ISSN 2054-3778

Subscribe: [shop.linuxvoice.com/subscriptions@linuxvoice.com](http://shop.linuxvoice.com/subscriptions@linuxvoice.com)

# FOSSpicks

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



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

Storage pruner

## QDirStat 1.0

One of our favourite Linux utilities is called *Filelight*. This is a KDE tool, now ported to other environments, that visualises the size of files stored on your drive using a multilevel pie chart (called a sunburst chart, apparently). With a simple glance, you can easily see which files were taking up the most space and where they were located. It's still the best way we've found of freeing up space by finding forgotten ISOs and games. Many of these features could also be found in KDE 3's *KDirStat*, often included in older distributions, and while its size

visualisation was an area chart, it also included a tree view of your files and folders, just like a traditional file manager. Having this functionality built right into the desktop, so you could launch the application as soon as you got your first 'Out of storage space' warnings, saved our storage bacon on several nervous occasions.

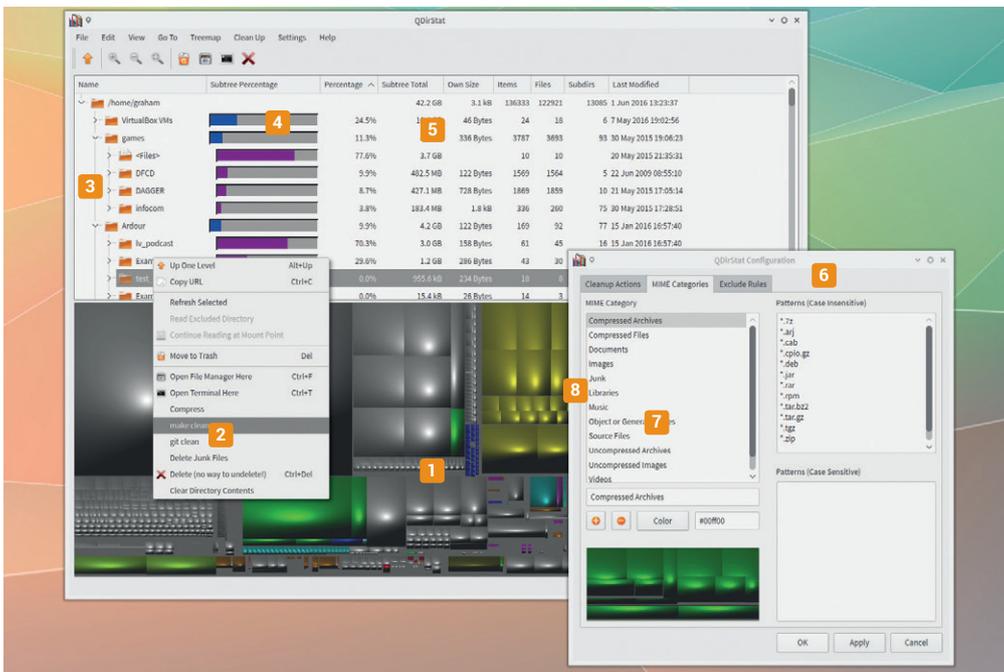
### KDE Kontinues

*QDirStat* is a continuation and port of *KDirStat* (80% of the old code has been rewritten), unlocking it from that old KDE dependency and making it available to any desktop

or operating system with a modern Qt port.

Unlike *Filelight*, which really only served one function, *QDirStat* gives you lots more information and control over how you look at your files and folder. The tree view lists the percentage of your storage that a file or folder is taking up, along with specific sizes, and a customisable colour can be defined for different file types, making them very easy to identify and filter from both the tree view and the area map (called a 'tree map' by *QDirStat*). You can even create rules for ignoring files you don't want listed.

Clicking on a file or folder nicely animates a transition in the other view so that everything stays in sync, and the same right-click context menu can be used to perform actions on your selected files and folders. Along with the expected, and dangerous, 'delete' option, you can open a terminal or a file manager, zoom in to expand a folder, and perform other actions like **make clean** and **git clean** to remove old build material from folders of source code. You can create your own cleanup actions from the Settings pane, and it's as easy as entering the command with some simple variables representing the path, file and directory name. This deceptively powerful feature makes *QDirStat* the most powerful application we've found for visualising space and folders and safely cleaning up your storage.



- 1 **Tree map** The area of each block corresponds to a file size, while the colour is for file type.
- 2 **Custom cleaning** Developers can remove unwanted build and git files, and you can add your own.
- 3 **File tree** The traditional unfoldable file view is also available.
- 4 **Percentages** Easily see exactly which files and folders are taking the most space.
- 5 **Custom columns** You can add and remove columns from the main view.
- 6 **View rules** Ignore files you don't want touched.
- 7 **MIME types** Set colours and create your own types for better visualisation.
- 8 **View rules** Ignore files you don't want touched.

Project website  
<https://github.com/shundhammer/qdirstat>

Bash Google

# Googler 2.4.1

It could be a temporary trend, but we're using the command line more and more each month. Its enforced minimalism is the opposite of GUI bloat because command line programmers are forced to think carefully about how functions are implemented and how those functions presented to the user, whether that's through a command's arguments, or through a *curses*-based interface.

This doesn't seem to happen with most desktops and graphical user interfaces, where there always seems to be room to add another feature, or where usability testing gets forgotten in all the excitement of adding new things. *Googler* is another step forward in our command line conversion, and it's one we've mentioned before. It's a tool that lets you interact with Google from the command line,

whether it's for scripts or for your own convenience. Type **googler** followed by your search terms and the results are delivered directly to your terminal.

We're happy to report that development has been prolific, and the recent 2.4.0 upgrade bundles several significant new features. The output looks much cleaner, especially for news results, and you can click on links to open them from your browser.

There are keyboard shortcuts too, and you can output the results with JSON formatting. This is excellent if you want to script your searches and integrate the results with other

Googler is a tool that lets you interact with Google from the command line

```

graham:python3 -- Konsole
File Edit View Bookmarks Settings Help
graham@mpb-arch:~$ googler --count 4 "Linux Voice"
1 Linux Voice | The magazine that gives back to the free software ...
https://www.linuxvoice.com/
Download Linux Voice Issue 19 Issue 19 of Linux Voice is nine months old, so we're releasing ... Lots and lots of Finds and a lovely Voice of the Masses.

1a Podcasts
https://www.linuxvoice.com/category/podcasts/
Podcasts Archive. podcast_image. Podcasts. Podcast Season 4 ...

1b Forums
http://forums.linuxvoice.com/
Forum: Topics: Posts: Last post. Help Having trouble with your ...

1c Creative Commons issues
https://www.linuxvoice.com/creative-commons-issues/
Creative Commons Issues. Read Linux Voice for Free! Nine ...

1d Subscriptions
http://shop.linuxvoice.com/products/subscriptions
Linux Voice is an independent GNU/Linux and Free Software ...

1e Download Linux Voice issue 1
https://www.linuxvoice.com/download-linux-voice-issue-1-with-audio/
Highlights in this issue: master Vlm, understand Systemd, solve ...

1f RSS for mp3 feed
http://www.linuxvoice.com/podcast_mp3.rss
In this episode: Good news from Qt and bad news for 32 bit Google ...

gopher (? for help)

```

We had thought that by using the command line we'd save ourselves from Google distractions. We were wrong.

applications, and it's this feature we think will be the most helpful, especially for analysis such as customised ranking reports.

**Project website**  
<https://github.com/jarun/googler/releases/tag/v2.4.1>

Search utility

# FSearch 0.1 alpha

Now that our hard drives are full and we've forgotten half the things on there, local search has become as important as online search. And yet we've yet to find the ideal solution.

There are lots of pervasive search tools – Ubuntu's works well, as does KDE's – but we're slightly distrustful of their pervasive nature. KDE's *Nepomuk* process often seems to take up huge slices of CPU and its database size can be huge. *FSearch* offers the same pervasive search functions, but constrains them to a single application where you have complete control over what is indexed, where the index lives and when you need to run it.

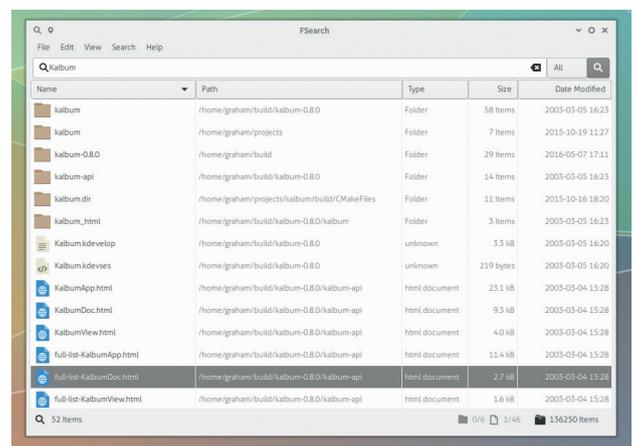
This size and scope would ordinarily make it slow, as you'd need to rebuild the database each

time you ran a search, but *FSearch* side-steps this issue by being incredibly fast. It indexed our home folder, containing 136,250 items, in under 2 seconds, and that's with a PC from five years ago.

It delivers results just as quickly, filling the results table as you type. It doesn't search within files, but that's not what *FSearch* is for. It helps you find your files instantly, whether that's via a regular expression or some vague Proustian remembrance of what the file is called.

*FSearch* is built using *GTK 3*, although the author wants to also

FSearch indexed our home folder, containing 136,250 items, in under 2 seconds



We used to think desktop integrated search was what we were looking for, but *FSearch* does it better and more quickly.

support *Qt 5*; it's tiny and takes no time to build. This is in huge contrast to many of the other search tools available, and despite this being a very early alpha, we had no issues with stability or usability.

**Project website**  
<https://cboxdoerfer.github.io/fsearch/>

C++ interpreter

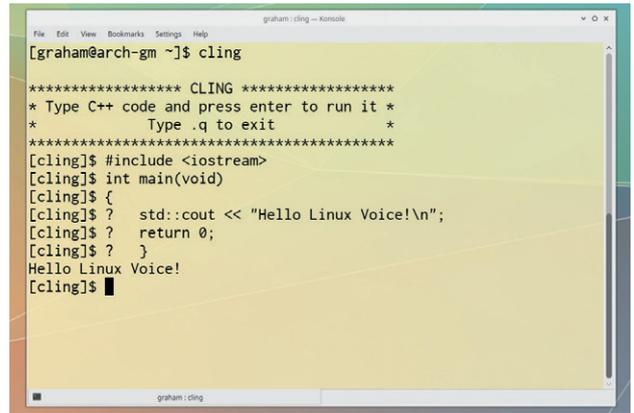
# Cling

There's a huge difference between interpreted programming languages and those that need to be compiled and run as binary objects. This difference is apparent in both the complexity of running your applications and in the approaches taken by the syntax within the languages. But the speed and capabilities of modern hardware is definitely bringing both camps closer from the perspectives both of end users and also developers.

Python and Go, for example, are interpreted languages that are used in a phenomenal number of mission critical applications, previously the domain of languages like C++. C++ itself is an old traditional language that's seen as poles apart from Python and Go. However, remarkably, C++ is also benefiting from modern hardware

with the creation of interpreters that can transform what was previously only compilable code into an executable instruction that runs immediately.

This on the fly interpretation is exactly what *Cling* does, using both the *LLVM* and *Clang* libraries. You type in C++ and it runs each line independently, building a virtual machine of your code. For those of us with a long history of C and C++, it feels like black magic. But it's also liberating, letting you try things out or run 'scripts' without the tedium of having to run **make** at every step to check your code. It's also a great educational tool, because it forces



Run C++ like it's embedded into your Commodore 64.

you to write good code, and any errors or ambiguities appear immediately.

If you've ever wanted to try C++ but have always been put off by the complexity of building, linking and running executables, the world of interpreted programming is right here waiting for you.

**Project website**  
<https://root.cern.ch/cling>

Modern hardware is bringing compiled and interpreted languages closer together

Developer package manager

# Qpm

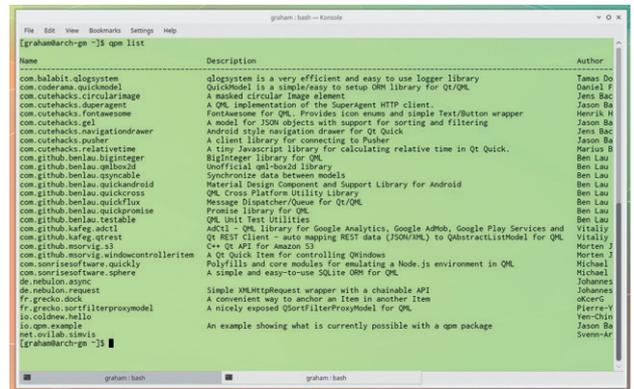
We've done some *Qt* programming in the past, and one of the things that can be annoying is trying to track down the various bits and pieces of *Qt*-related dependencies that you may need to pool together to make a single application. This is a problem that *Qpm* attempts to solve rather elegantly.

Unlike packages for your chosen distribution, *Qpm*'s packages are tailored specifically for your project, enabling you to use them and incorporate them into your code without any extra hassle. This is different from installing them through a package manager, as you're never certain which versions are installed, and that's if there's a package available. It's still early days for *Qpm*, but the packages

available already are unlikely to be available through a normal package manager, and really help with both *Qt* and *QML/QtQuick* development. It also keeps your packages and dependencies up to date, independent from your distribution package updates, and that's important when you need control over exactly what your applications are linked to.

**Developers!**

At the time of writing, there are only 34 packages being served (**qpm list** tells you this), but we're hoping many more are included in the



At least your typical developer won't be put off by using **curl** to install the package.

future, as the **qpm** command itself makes it easy to publish your own. The whole command itself is easy to use – it's installed via a simple **curl** command and typing **qpm** tells you all you need to know about searching, installing, installing and creating packages.

**Project website**  
[www.qpm.io](http://www.qpm.io)

Qpm keeps your packages and dependencies up to date independent from your distro

## Emulator front-end

# AQemu 0.9.2

Emulating old hardware is definitely fun, even if most of the fun comes from getting old games running rather than playing them. Too often we go back to our old favourites to find that without our 14-year-old reflexes, we're just cannon fodder. Emulating modern hardware isn't always fun, but it is very practical. It's the best way of experimenting with a new distribution, and for deploying a thousand servers across the cloud (sort of).

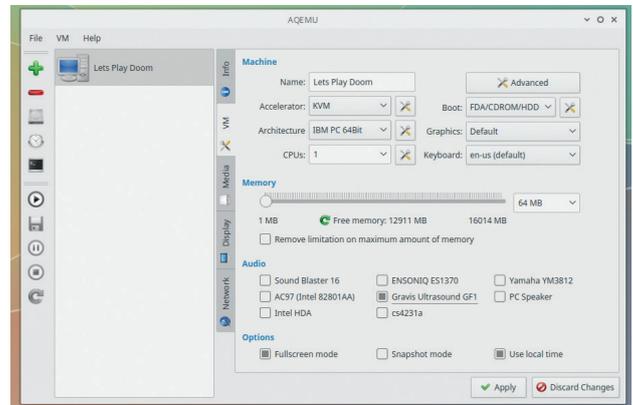
Back on the desktop, *QEmu* is one of our favourite tools, and while it's great from the command line, there's nothing wrong with accepting a little help from the GUI. *AQemu* is an alternative GUI that provides lots of help, from locating the location of the binary at startup to helping you create your first virtual hardware. We like the way,

for example, you can specify the PC hardware you'd like to conjure up by its age – a PC from 1990–95, for example, which is a great way of playing games from the golden era of DOS. Sadly, the GUI lacks the equivalent 'Turbo' button, but we may make a feature request for a fully authentic 1995 experience.

## Run Linux on Linux

For the less nostalgic, *AQemu* will also create machines using a Linux, Windows or OS X template, complete with storage and networking. As with all of the options, you still need to provide your own installation medium, and

AQemu is an alternative GUI for the QEmu emulator that provides lots of help



Why is it that a PC with a CPU speed of 66MHz and only 64MB of RAM booted faster than the futuristic machine we're typing on?

go through the installation procedure, even if this is from a disk image of *Doom* taken from a 3.5-inch floppy disk.

**Project website**  
<https://sourceforge.net/projects/aqemu>

## Disk manager

# KDE Partition Manager 2.2.0

All of us have been saved by the perennial *GParted*. It's the one application you can rely on when you need to dance that dangerous partitioning dance. When there's delicate manoeuvring to be done, we'd even prefer it over the command line equivalents because sometimes you just want to see that your data is safe.

*GParted* has never failed us, despite our own actions failing us on more than one occasion. *KDE Partition Manager* is very much like *GParted*, only with what we'd describe as a more modern user interface. Having the choice, and the competition, can only be a good thing for such an important task, and we're suckers for a nice KDE application.

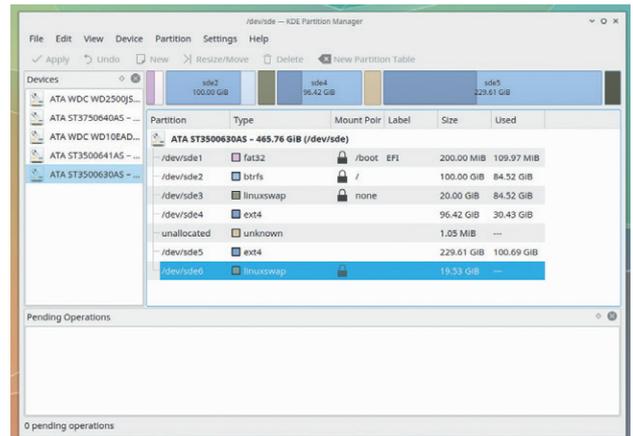
*KDE Partition Manager* looks fantastic, especially if you're not

using a *GTK*-based desktop, and offers almost all the same features as *GParted*.

But this release is significant to us for one major reason – *KDE Partition Manager 2.2.0* adds a feature we've wanted/needed from *GParted* for ages – the ability to resize encrypted LUKS partitions. This is important because more and more of us are using LUKS partitions to store our Linux and data installations.

## Killer feature

LUKS partitions stop people getting access to your data when they have access to your hardware, but because of the way LUKS hides Linux filesystems within their random noise of encryption, resizing has been a tricky problem, and being able to resize your



As you can see, with five hard drives and up to seven partition on each, we desperately needed a good partition manager.

partition rather than creating a new one and copying over the data is going to save us lots of trouble. For that feature alone, thank you *KDE Partition Manager*!

**Project website**  
<https://stikonas.eu/wordpress/2016/05/27/kde-partition-manager-2-2-0>

Webmail

# Roundcube 1.2.0

This has been a month of turmoil for one of the of the largest server-based open source projects we rely on. The *OwnCloud* project has been forked into *NextCloud*, and we genuinely hope this doesn't affect the development or momentum of what we think is one of the most important open source and Free Software projects available.

It shouldn't, as most of the developers from the old project have moved to the new project, but we don't want to imagine a future without a genuine alternative to the proprietary cloud-based behemoths of Google, Microsoft and Facebook.

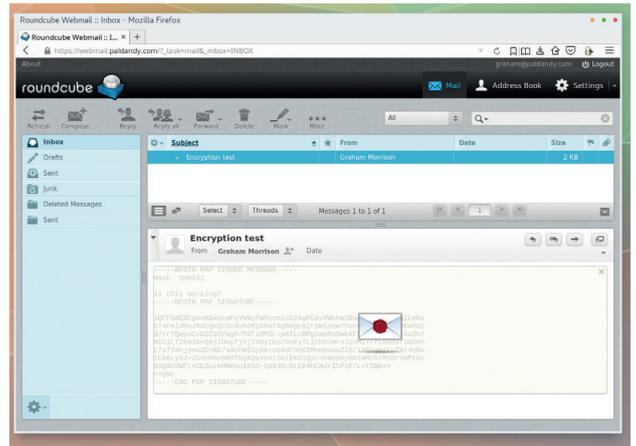
Another cloud project we used has also celebrated this month, and that's *Roundcube*, with a new release of its brilliant webmail application. The major feature for this update is that *Roundcube* now

has native support for encryption. This is significant because it's both a brilliant feature and a brilliant show of support for what should be part of every email client and email exchange.

### Keep it secret...

*Roundcube* accomplishes encryption either via a *Firefox* plugin called *Mailvelope*, or on the server via the *Enigma* plugin and *GnuPG*. From the perspective of a user, *Mailvelope* is easy to install and get to grips with, and makes grabbing remote PGG keys for decryption simple – when it detects a public key, it will automatically import this

When it detects a public key, it will import this into your web-based keychain



Finally, there's an open source webmail solution that has encryption support baked in.

into your web-based keychain so you can decrypt messages. This is exactly what you need, so there's even less of an excuse to start signing and encrypting emails. Oh, and we love the way you can now finally search between dates!

**Project website**  
[/news/2016/05/22/roundcube-webmail-1.2.0-released](https://news/2016/05/22/roundcube-webmail-1.2.0-released)

Spectrum analyser

# Cava (from git)

We have Kurraptor on our IRC channel to thank for this particular find. It's a simple but useful and slightly compulsive discovery. *Cava* is a spectrum analyser for the command line, rendering the various frequencies of an audio signal using *Curses* onto your console. Because it's for the command line, it's both tiny and quick, and being quick is a great asset if you're measuring audio in real time.

On a practical level, it can be a useful way of monitoring your surroundings before a recording. Higher and lower frequencies may be inaudible to our old ears and yet be visible on the screen. But because this is also running from the command line, you can use it to snoop, at least visually, on the audio

going into remote computers and even servers.

It feels weird SSHing into a box and then watching the little frequency bars bouncing around, but it's also surprisingly useful. This could be a visual baby monitor, for example, or a way of seeing if a conversation has finished without having to listen to the content. If you're processing audio, it's also a great way to see the range of frequencies you're recording, or have recorded.

By default, it works easily with *PulseAudio*. Just run the command and make sure *PulseAudio* is

It's a great way to see the range of frequencies you're recording, or have recorded



Now Hollywood film directors have another Linux command to use when portraying hackers on screen.

listening on the right input, and we'd recommend the excellent *Pavucontrol* for this. A few simple key commands can change the scale and colours, but that overcomplicates things. Just let yourself get mesmerised by the look of music on your console.

**Project website**  
<https://github.com/karlstav/cava>

Data rescue

# TestDisk 7.0

**P**icture the scene: you're browsing an online shopping emporium and see that there's a special offer on 1TB hard drives. This is brilliant. You can now buy several and potentially put them to good use as part of a RAID array, or just have lots of cheap storage. This is what we did a couple of years ago, except we never got around to building a RAID configuration and instead used the drives within our main PC where the drives quickly absorbed gigabytes of Kylie Ann Minogue rarities.

Two years later, we're updating Arch and installing Ubuntu, when we repartition one of those drives, certain that the drive we chose was correct. Except, because it's identical to two more 1TB drives, it isn't. It's the one filled with treasured sounds. Denial. Anger. Bargaining. Depression. Acceptance.

## Lazarus raised

Thankfully, there's *TestDisk*. This amazing utility has saved our raw data more times than we can think of. It's particularly brilliant when you've accidentally repartitioned a drive already full of data, a situation that initially looks lamentably grim. But because you've only written a new partition table to a special part of the drive, and not overwritten your actual data, there's a chance

```

PhotoRec 7.0, Data Recovery Utility, April 2015

Please select a destination to save the recovered files.
Do not choose to write the files to the same partition they were stored on.
Keys: Arrow keys to select another directory
      C when the destination is correct
      Q to quit
Directory /home/graham
>drwxr-xr-x 1000 1000 4096 6-Jun-2016 19:22 .
drwxr-xr-x 0 0 4096 25-May-2016 13:26 ..
drwxrwxr-x 1000 1000 4096 13-May-2016 11:06 Calibre Library
drwxr-xr-x 1000 1000 4096 6-Jun-2016 17:10 Desktop
drwxr-xr-x 1000 1000 4096 15-Apr-2016 16:45 Documents
drwxr-xr-x 1000 1000 4096 6-Jun-2016 17:48 Downloads
drwxr-xr-x 1000 1000 4096 15-Apr-2016 16:45 Music
drwxr-xr-x 1000 1000 4096 21-Apr-2016 17:43 Pictures
drwxr-xr-x 1000 1000 4096 15-Apr-2016 16:45 Public
drwxr-xr-x 1000 1000 4096 15-Apr-2016 16:45 Templates
drwxr-xr-x 1000 1000 4096 15-Apr-2016 16:45 Videos
drwxrwxr-x 1000 1000 4096 24-May-2016 11:53 bin
drwxrwxr-x 1000 1000 4096 17-Apr-2016 11:39 build
drwxrwxr-x 1000 1000 4096 22-Apr-2016 15:36 canonical
drwxrwxr-x 1000 1000 4096 25-Apr-2016 15:13 logs

```

Never start a partitioning operation without having *photorec* and *testdisk* handy (and a backup, of course).

– and we don't want to promise miracles – there's a chance your old table can be resurrected and rewritten to your drive, reinstating your data immediately.

If this happens to you, first make sure you don't touch your drive at all. Instead, reboot to a rescue USB stick or run *TestDisk* from a distro that doesn't touch your drive. There's a good chance it will be able to restore your old configuration and your data.

If this doesn't work, there's another part of the *TestDisk* suite that can help, and that's *PhotoRec* and its GUI counterpart, *QPhotoRec*. These scan the raw blocks of your drive looking for the telltale signatures of known file types (not

just photos, as their names imply). Dozens of file types are supported, and this major update adds more, including *.kra Krita* files. When detected, you can rip out the raw data into a usable file again, though you'll often lose the filename and any folder organisation you had.

This is a last resort rather than a fix, but it might mean you're able to keep your precious photos, and both *TestDisk* and *PhotoRec* are two the best tools to have on hand. That they're still being developed and passing major milestones like a 7.0 release is something we're all grateful for. Thanks *TestDisk!*

Project website  
[www.cgsecurity.org](http://www.cgsecurity.org)

## STEP BY STEP: RESTORE YOUR OLD PARTITION

**1** Run *testdisk* with the drive you want to scan as the single argument. You'll also need **sudo** or root privileges.

```

TestDisk 7.0, Data Recovery Utility, April 2015
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org

TestDisk is free software, and
comes with ABSOLUTELY NO WARRANTY.

Select a media (use Arrow keys, then press Enter):
/dev/sda (479 GB / 446 GiB) - CHS 58350 255 63

Proceed [C] Quit [Q]

Note: Disk capacity must be correctly detected for a successful recovery.
If a disk listed above has incorrect size, check its jumper settings, BIOS
detection, and install the latest OS patches and disk drivers.

```

**2** *TestDisk* will guess the partition type, but you need to make sure it's guessed correctly. Then select Analyse.

```

Disk /dev/sda - 479 GB / 446 GiB - CHS 58350 255 63

Please select the partition table type, press Enter when done.
[EFISYS] EFI System partition
[MAC] Mac partition table
[MAC] Apple partition map
[None] Non-partitioned media
[Sun] Sun Solaris partition
[None] Non-partitioned media
[None] Return to disk selection

Note: EFI GPT partition table type has been detected.
Note: Do NOT select 'None' for media with only a single partition. It's very
rare for a disk to be 'Non-partitioned'.

```

**3** *TestDisk* will look for the remnants of a partition table. When it finds one, select Write – then donate to *TestDisk!*

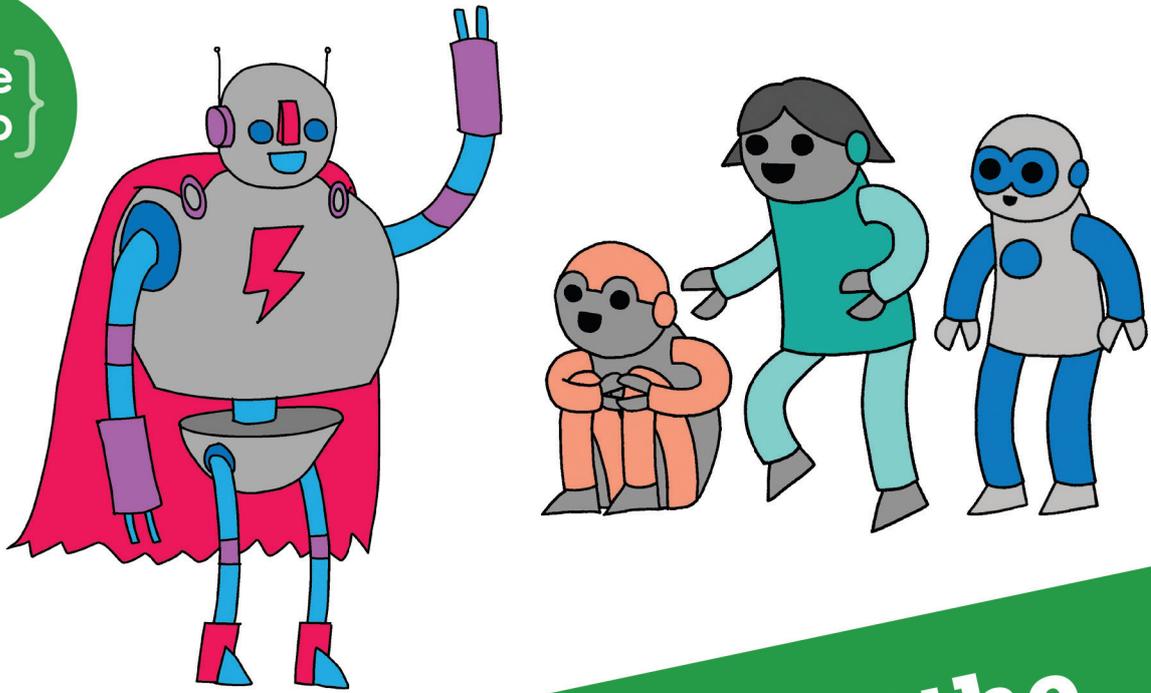
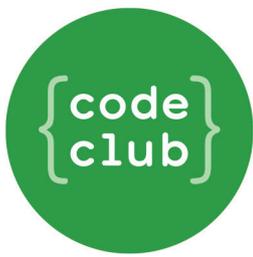
```

Disk /dev/sda - 479 GB / 446 GiB - CHS 58350 255 63

Partition Start End Size In sectors
1 P: EFI System 40 409639 409600 [EFI]
2 P: Mac HFS 409640 272847135 273437496
3 P: Mac HFS 272848136 272816472 1509936
4 P: MS Data 272816888 272701769 583808
5 P: MS Data 272702768 272702765 4096
6 P: MS Data 409238856 937392223 447162368 [arch]

[Quit] [Deeper Search] [Write]
[None] Return to disk selection

```



Can you help inspire the next generation of coders?



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

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

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

There are loads of ways to get involved!

So to find out more, join us at [www.codeclub.org.uk](http://www.codeclub.org.uk)

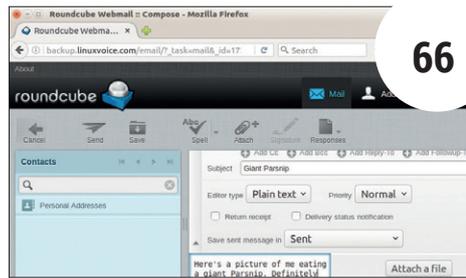
# TUTORIALS

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



**Mike Saunders**  
Makes a bomb shipping computers from Barnard's Star to Sol.

## In this issue . . .



66

### Hide data in images with Steganography

Conceal secret messages or other information inside files with **Ben Everard's** sneaky guide to the art of steganography.



68

### Build fast, clean websites with Hugo

Does the world need yet another website creation engine? Well yes, when it's as awesome as *Hugo*, as **Amit Saha** explains.

**H**ave I mentioned recently how much I love *Vim*? It has been a while since I've written about it in Linux Voice, and if it weren't for Ben's wise editorial decisions I'd do a 32-page feature every month about its raw awesomeness. But seriously, learning a powerful text editor like *Vim* was the best decision I ever made (in my computing life, at least. Combining bacon with Sriracha sauce was the best decision ever).

*Vim* scares many people away; it feels like some weird relic from the 1970s at first (and indeed, its origins lie way back in early Unix releases). But don't just think about it like an editor – it's more of a machine and a language for manipulating text. Just like a programming language, you can build up useful routines and methods for doing things, and repeat them to save you heaps of time. Yes, *Vim* may be overkill for taking a few notes, but for anything else it's superb. Check out the video I made back in 2014 to learn to really love *Vim*: [www.youtube.com/watch?v=rfl9KQb\\_HVk](http://www.youtube.com/watch?v=rfl9KQb_HVk) – and drop me a line and let me know if you become a convert too!  
[mike@linuxvoice.com](mailto:mike@linuxvoice.com)

**Markdown** 74

A blank line starts a new paragraph. A # cha the start of a new line is a top level (h1) headi, either side of a word is *italic* and two is **bold**.

**An h2 heading**

Lists begin after a blank line and items start with numbers and a period like this:

1. First things first
2. Or you can use \* or - at the start of a line for bullets

### Publish with Free Software

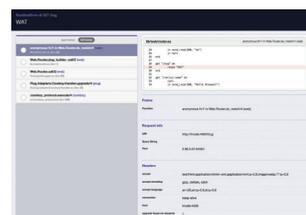
Yes, it's possible to make ebooks from scratch – **Andrew Conway** has all the tips and tricks you need to do just that.

## Coding

```

[John@rsu]$ openssl rsa -in key.der -inform der -text
Private-Key: (64 bit)
modules: 1367133628896425653 (0xdbc79619e08efb5)
publicExponent: 65537 (0x10001)
privateExponent: 531444176792982513 (0x76011a3fb21
prime1: 3273361529 (0xc31b8879)
prime2: 413728257 (0xf13e1d)
exponent1: 1295519345 (0x4d380c71)
exponent2: 20202669 (0x1344ad)
coefficient: 1526950891 (0x5b036beb)
RSA key ok
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MDCC0R9CC0C3vW1h1q.vv0IDR9R8AgglTF8GpvZIT00TFAMhB1kR
AgRN
0n5xhg0RNEStAgRbA2vr
-----END RSA PRIVATE KEY-----
    
```

**Encryption** 82  
Understand how the encryption that secures the web works. **John Lane** introduces RSA.



**Elixir** 88  
**Mihalis Tsoukalos** gets you started with this language and shows you some of its secrets.



78

### Display data on a physical interface

**Les Pounder** mixes together a Raspberry Pi Zero, stepper motors and external data sources for maximum awesomeness.

Get access to every Linux Voice tutorial ever published in our digital library of back-issues available exclusively to subscribers – turn to page p56 to join.

# HIDE CONFIDENTIAL DATA WITH STEGANOGRAPHY

Hide encrypted data in the one place the spooks will never look for it – your photos.

**BEN EVERARD**

## Why do this?

- Guard against data theft.
- Communicate securely.
- Combine your backups with your family portraits to save space.

**E**ncryption is a great way to keep private information secret; if used properly, there's no way for an attacker to break modern encryption. However, sometimes we don't want an attacker to even know we have information that's encrypted. An extreme case for this is that an attacker could blackmail you into giving them the password, but it can be useful for any occasion when we simply want to keep our encryption private.

In this tutorial we're going to look at steganography, which is the process for hiding data inside media files. Typically, this is done with image files. The technique works by subtly changing some details of the image in a way that's barely perceptible to the human eye. To the rest of the world, it looks like you have an obsession with images of fluffy kittens, but really you're safely guarding your secret family recipe that makes the fluffiest Yorkshire puddings.

## STEP BY STEP: CONCEAL ENCRYPTED DATA

### 1 Get the software

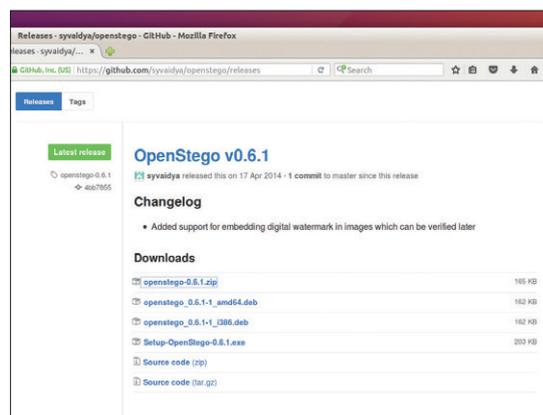
There's a wide range of steganography software available. We've opted to use *OpenStego*, because it's written in Java so should work on most systems and is easy to use. You can download the this software from <https://github.com/syvaidya/openstego/releases> – there's a Deb file for Debian-based systems or a Zip file for other Linuxes (and you should be able to run this on any system with Java). To run *OpenStego* from the Zip, just extract the contents and run the `openstego.sh` script. There's also a Windows release if you need to share your secrets with a Windows machine.

You'll need to make sure you have Java installed. *OpenStego* runs on either Oracle's version of Java or the OpenJDK. We'll also use the `gpg` command line tool for encrypting data, and you should find this in your distro's repositories if it's not installed by default.

### 2 Get an image

Any image should work, but some images work better than others. *OpenStego* can read most image formats, but the output will always be in PNG because it has to be in a lossless format – the type of compression in JPEGs, for example, could destroy the data stored in the image file. This formatting of the image can lead to some images being viewed with more suspicion than others. Photographs are rarely stored as PNGs, so an alert eavesdropper may be suspicious if they come across a large number of photos stored in PNG format. Screenshots, on the other hand, are regularly kept as PNGs, as JPEGs struggle with text.

*OpenStego* will, by default, store three bits per colour channel per pixel, so you can store one byte per pixel (it's technically 9 bits, but the maths is easier if you use 1 byte, and you rarely need to fill an image). You can split a large amount of data across several images if needed.



### 3 Encrypt your data

Steganography hides your data within images, which is good, but not perfect – if an adversary finds out you've used steganography then there's a good chance they'll be able to recover the data. *OpenStego* does enable you to password-protect your data, but it does this using the outdated DES encryption algorithm. If you're going to the trouble of hiding your data in images, the data is probably important enough to be worth encrypting using the best available encryption, so we'll first secure it using the **gpg** command. This is done with the following (enter a password when prompted):

```
gpg --cipher-algo AES256 -c <inputfile>
```

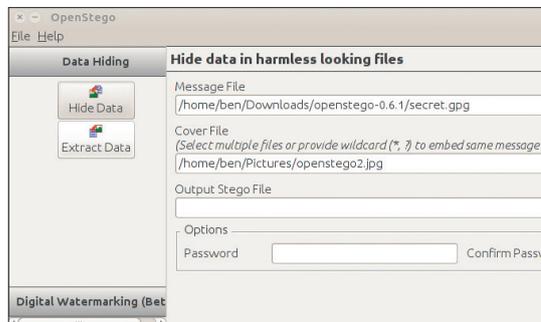
This will create a new file with the same name as **inputfile**, but with the additional suffix **.gpg**. It's this encrypted file that you will use with *OpenStego*.



### 4 Hide your secrets

Now you have your image and your encrypted file, it's time to combine the two into our steganographic secret. Start *OpenStego* and you will see the Hide Data and Extract Data options on the left of the window. Make sure Hide Data is selected, then enter the GPG file in the Message File box, the image file in the Cover File box (here you can select multiple images if you've got a large message file), and a name for the output file. There's no need to enter a password, as we've already encrypted the files and an additional layer of security won't add anything.

The original image file will remain untouched, and the data will be hidden inside the file you created in the Output File option. You can open this up in an image viewer and it should look identical (or almost identical) to the original file despite having a secret message stored inside it.

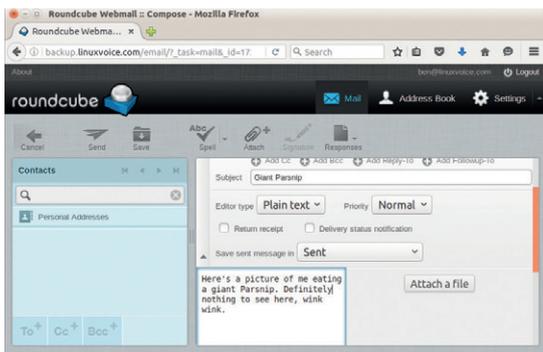


### 5 Share your secrets

Now you have your secrets hidden inside an image, it's time to do something with them. What, exactly, this step entails depends on what your secrets are and why you hid them inside a file.

If you just did it to ensure your data doesn't fall victim to any hackers that get into your computer, then the only thing to do here is get rid of the other files containing those secrets. If you did this to enable you to share your secrets then you can now send them to other people. If you post these pictures on a social network such as Facebook, they'll be re-formatted and will probably lose the information stored within them. You have to share them as files rather than as images, so anything like email attachments or DropBox should work fine.

You'll also need to make sure that anyone who should be able to read them has the password.



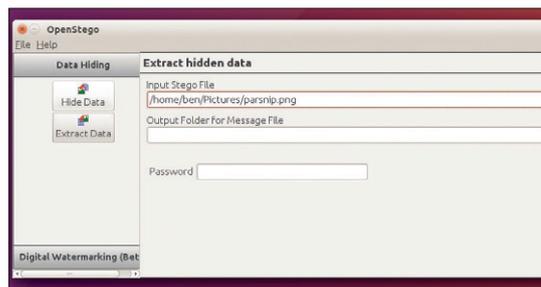
### 6 Decrypt your secrets

The final step is to extract the secrets from the image. Just like hiding them, this is a two-stage process: first we have to get the encrypted data out of the image, then we have to decrypt the data. Extracting the encrypted data is done using *OpenStego*. This time you need to select Extract Data and enter the image file and the output file. You don't need to enter the password as we'll decrypt it next. Make sure the output file has the **.gpg** suffix.

Once you've got the output as a GPG file, you can decrypt the data using the command

```
gpg <encryptedfile>
```

GPG will automatically detect the cipher type and prompt you to enter the password. You should now have recovered your secrets from the image. As you've seen, steganography is a simple way of adding an extra layer of protection to your most valuable secrets. Now go out and take loads of photos to mask your secret snaps.



# SET UP YOUR NEXT BLOG WITH HUGO

## Generate hugely configurable static websites at lightning speed!

AMIT SAHA

### Why do this?

- Get content online, fast, without having to wrestle with the complexities of a CMS.
- It's another excuse to mess around with Go!

With static website generators, we can write the content of your site as formatted text files in our favourite text editor, convert them to HTML files, copy those files to a web host and we're done. The number of static site generators available today is staggering; some of the most popular ones are *Jekyll*, *Octopress*, *Pelican*, *Nikola* and of course, *Hugo*.

*Hugo* is written in Golang, so we first have to make sure we have Golang installed and have the **GOPATH** environment variable on your system. If you don't have the **go** tools (compiler and other tools) installed, you can either use the distro's package manager to install them or download the Linux binary and follow the instructions on the install page at <https://golang.org/doc/install>. Once the installation steps are completed, open your favourite terminal emulator, type **go version** and it should print a message similar to below:

```
$ go version
go version go1.6 linux/amd64
```

We next need to set up our Golang workspace. If you already have **GOPATH** set up, you may skip ahead. Create a sub-directory **golang** in your home directory (**/home/<user>**) and two sub-directories, **src** and **bin** inside it. The directory tree for your workspace should look as follows:

```
$ tree -L 1 ~/golang
├── bin
└── src
```

### Today I Learned

The **go** compiler and other tools expect the **GOPATH** environment variable to point to the workspace directory, so set the following to your **.bashrc** or the file relevant to your shell, so that it is always set when you start a new terminal session (Replace **<user>** with your username):

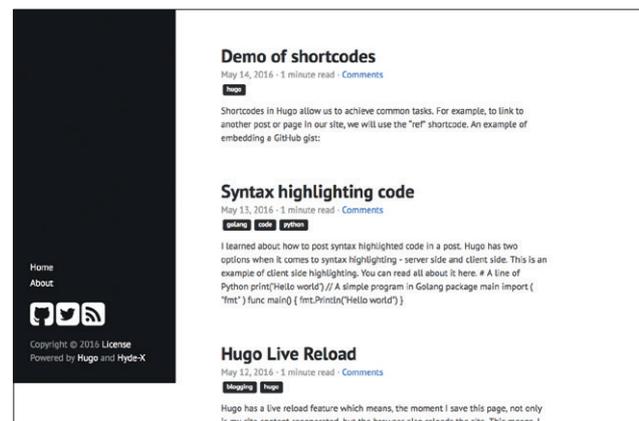
```
export GOPATH=/home/<user>/golang
```

Once you have set the above, start a new terminal session and type **go env GOPATH**:

```
$ go env GOPATH
/home/<user>/golang
```

Next, we will get the source for *Hugo* and build it:

```
$ go get -v github.com/spf13/hugo
github.com/spf13/hugo (download)
```



The site we will be building in this article – you can see it live at [https://amitsaha.github.io/linux\\_voice\\_4](https://amitsaha.github.io/linux_voice_4).

### github.com/fsnotify/fsnotify (download)

```
..
..
```

At this stage, we should have *Hugo* built and the binary placed in the **\$GOPATH/bin/** sub-directory. You can print the version of *Hugo* using **hugo version**:

```
$ $GOPATH/bin/hugo version
Hugo Static Site Generator v0.16-DEV BuildDate:
2016-04-27T18:29:22+10:00
```

If we execute **\$GOPATH/bin/hugo --help**, it will print a brief overview of the various flags and commands:

```
$ $GOPATH/bin/hugo --help
hugo is the main command, used to build your Hugo site.
Hugo is a Fast and Flexible Static Site Generator
built with love by spf13 and friends in Go.
Complete documentation is available at http://gohugo.io/
```

Usage:

```
hugo [flags]
hugo [command]
```

```
..
```

As the help message shows, *Hugo*'s functionality is available via various subcommands and flags. For example, the **hugo new site** command will be used to create a new site and **hugo server** will start a local server to serve our site's content. Appending **--help** to a *Hugo* sub-command (eg **\$GOPATH/bin/hugo new --help**) will display help message for the sub-command as well.

We will create a site called "Today I Learned". The content of the site, including the posts, pages and the configuration, will live in a single sub-directory in our filesystem. Let's say we want to create our site in a subdirectory **today-i-learned**; we'll use the **hugo new site today-i-learned** command. This will create a new sub-directory **today-i-learned** in the directory you executed the command:

```
$ GOPATH/bin/hugo new site today-i-learned
```

Congratulations! Your new Hugo site is created in "/home/amt/today-i-learned".

Just a few more steps and you're ready to go:

1. Download a theme into the same-named folder. Choose a theme from <https://themes.gohugo.io> or

create your own with the "hugo new theme <THEMENAME>" command

2. Perhaps you want to add some content. You can add single files with "hugo new <SECTIONNAME>/<FILENAME>.<FORMAT>"

3. Start the built-in live server via "hugo server"

For more information read the documentation at <https://gohugo.io>.

At this stage, *Hugo* has created a scaffold for our site:

```
$ tree today-i-learned
```

```
.
├── archetypes
├── config.toml
├── content
├── data
├── layouts
├── static
└── themes
```

The **config.toml** file is our site configuration formatted as a TOML file. The subdirectories created above are all empty and they serve different purposes:

- **archetypes** When we create new content, *Hugo* pre-fills the content's front matter with metadata such as title and date. Using archetypes, we can customise the metadata we want to be pre-filled.
- **content** This is where all the content will live
- **data** This directory can be used to load custom data from a YAML-, JSON- or TOML-formatted file and make it available at site-generation time. You can think of it as like a file based data store for your site.
- **layouts** This is where the layout of the site can be customised using templates.
- **static** This is where, we should place any custom CSS, JavaScript, images or any files we want to be a part of our site.
- **themes** We will store our theme here.

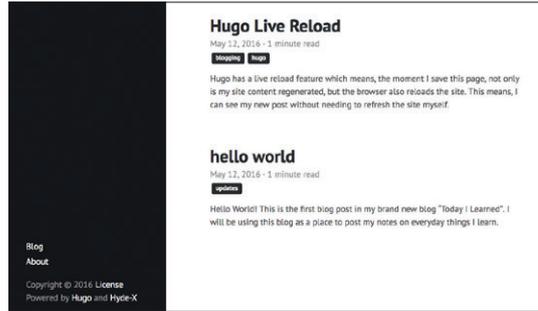
At this stage we have our site structure ready, but we are missing content and a theme. As important as they are, let's ignore them for the time being and go ahead and start a server to serve our site:

```
$ cd today-i-learned/
```

```
$ GOPATH/bin/hugo server
```

Started building site

```
=====
=====
```



Posts with categories – generated in double-quick time..

Your rendered home page is blank: /index.html is zero-length

\* Did you specify a theme on the command-line or in your

"config.toml" file? (Current theme: "")

\* For more debugging information, run "hugo -v"

```
=====
=====
```

0 draft content

0 future content

0 pages created

0 non-page files copied

0 paginator pages created

0 tags created

0 categories created

in 32 ms

Watching for changes in /home/amt/today-i-learned/ (data,content,layouts,static)

Serving pages from memory

Web Server is available at http://localhost:1313/ (bind address 127.0.0.1)

Press Ctrl+C to stop

A web server has been started for us and you can visit the URL from your browser, but you will be greeted with a blank page – which of course isn't surprising. Keep the server running, switch to a new terminal window to create our first blog post:

```
$ cd today-i-learned/
```

```
$ GOPATH/bin/hugo new post/hello-world.md
```

/home/amt/today-i-learned/content/post/hello-world.md created

A new subdirectory, **post**, has been created under the **content** directory with a file **hello-world.md** under it:

```
$ tree content
```

```
content
├── post
│   └── hello-world.md
```

At this stage, it's worth discussing the concept of "sections" in Hugo. Hugo lets you organise your site's content into any structure you please. So, for example (as we will in this article) we can categorise our site's content into being a "post" and "page", or you could just call them something else entirely.

We can now write our blog post content in the **hello-world.md** file. It currently will contain the following:

```
+++
date = "2016-05-12T08:05:42+10:00"
```

I learned about how to post syntax highlighted code in a post. Hugo has two options when it comes to syntax highlighting - server side and client side.

This is an example of client side highlighting. You can read all about it [here](#).

```
# A line of Python
print('Hello world')

// A simple program in Golang
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello world")
}
```

Figure 4: Syntax highlighting code demonstration.

```
draft = true
title = "hello world"
+++
```

The content within the `+++` is referred to as the front matter, and is the metadata. Open the file in your favourite text editor and change the content to the following:

```
+++
date = "2016-05-12T08:05:42+10:00"
title = "hello world"
+++
```

Hello World! This is the first blog post in my brand new blog "Today I Learned". I will be using this blog as a place to post my notes on everyday things I learn.

We have removed the `draft = true` line from the header so that our post will be available for consumption via our site. We now have some content for our site, but we don't have a theme yet and so we still can't view our site content. <http://themes.gohugo.io> is a showcase of a number of *Hugo* themes. The choice of theme will determine how your site's content is structured and how it looks. In addition, a theme may have additional features already available. For our site, we will use the "hyde-x" theme (<https://github.com/zyro/hyde-x>). It looks nice and makes adding social links really easy. Instead of cloning the *Git* repository for the theme, we will download a Zip archive of the theme from <https://github.com/zyro/hyde-x/archive/master.zip>, unzip it and place it in the `themes` sub-directory created for us:

```
$ cd themes
$ wget https://github.com/zyro/hyde-x/archive/master.zip
$ unzip master.zip
$ mv hyde-x-master hyde-x
$ rm master.zip
```

At this stage, our site structure looks like this:

```
$ tree -L 3
.
├── archetypes
├── config.toml
└── content
```

```
├── post
├── hello-world.md
├── data
├── layouts
├── static
├── themes
├── hyde-x
│   ├── LICENSE
│   ├── README.md
│   ├── archetypes
│   ├── images
│   ├── layouts
│   ├── static
│   └── theme.toml
```

Our next step is to tell *Hugo* that we want to use the "hyde-x" theme. We do so by changing our `config.toml` to be the following:

```
baseurl = "http://replace-this-with-your-hugo-site.com/"
languageCode = "en-us"
title = "Today I Learned"
theme = "hyde-x"
```

Note that we also changed the title of our site and specified the theme. Now, if we go back to the terminal where we left the "hugo server" running, we will see messages like:

```
Change detected, rebuilding site
2016-05-12 08:16 +1000
Config file changed: /home/amit/today-i-learned/config.toml
```

Now, if we go back to `http://localhost:1313` on your browser, you will see your site with your first post. If you click on the post title, you are led to the entire post which is available at `http://localhost:1313/post/hello-world`. If you now look at your site's directory, you will see no traces of HTML files anywhere. This is because *Hugo* serves your pages from memory.

Let's add a new post to our blog while keeping our server running:

```
$ pwd
/home/amit/today-i-learned
$ GOPATH/bin/hugo new --editor emacs post/hugo-live-reload.md
```

Passing the option `--editor` followed by the path to your editor will open your editor with the new blog post front-matter pre-filled in and ready for you to type in your post. Note how the front-matter has changed from our first post? This is because the `hyde-x` theme defines an archetype in its `archetypes` subdirectory, which overrides the default archetype. We will type in a couple of sentences:

*Hugo* has a live reload feature, which means the moment we save this page, not only our my site content regenerated, but the browser also reloads the site. This means, we can see our new post without needing to refresh the site.

As I write in the blog post above, you will see that the site you had opened on your browser has automatically been live reloaded. This is a unique feature of *Hugo*, which it achieves by utilising web sockets, and is certainly great when you're working on

PRO TIP

The `[params]` section in the `config.toml` file are made available by *Hugo* to be used in templates. In our case the templates that will be used to render our homepage are part of the theme we are using and hence any theme specific parameters will be specified in the "params" section.

## Resources

- **Hugo showcase** <https://gohugo.io/showcase>
- **Archetypes** <https://gohugo.io/content/archetypes/>
- **Sections** <https://gohugo.io/content/sections/>
- **Configuration** <https://gohugo.io/overview/configuration/>
- **Shortcodes** <https://gohugo.io/extras/shortcodes/>
- **Syntax Highlighting** <https://gohugo.io/extras/highlighting/>
- **Templates** <https://gohugo.io/templates/homepage/>
- **hyde-x theme** <https://github.com/zyro/hyde-x>
- **Data files** <https://gohugo.io/extras/datafiles#the-data-folder>
- **Static sites with docker** <http://ilkka.io/blog/static-sites-with-docker>
- **Automated site deployments** <https://gohugo.io/tutorials/automated-deployments>
- **Hugo tools** <https://gohugo.io/tools>

your site's content. To disable the live reload, we can pass the `--disableLiveReload` option to "hugo server".

## Adding pages to our blog

To add a new page to our blog, we will once again use the "hugo new" command. From within the **today-i-learned** directory, execute the following command, which will once again open a specified editor to edit the page we want to create:

```
$ GOPATH/bin/hugo new --editor emacs page/about.md
```

Type in something that you would want to be in the page, save the file and exit. You will see that a file **about.md** has been created in the **page** subdirectory under the **content** directory.

```
$ tree content/
content/
├── page
│   └── about.md
├── post
│   ├── hello-world.md
│   └── hugo-live-reload.md
```

On the terminal you have "hugo server" running, you will see messages such as:

```
adding created directory to watchlist /home/amit/
today-i-learned/content/page
Change detected, rebuilding site
2016-05-13 07:58 +1000
0 draft content
0 future content
2 pages created
1 non-page files copied
1 paginator pages created
0 tags created
0 categories created
in 7 ms
Change detected, rebuilding site
2016-05-13 07:58 +1000
```

However, if you go to your browser window, you will not see a link to the page that we just added. The reason is that menus in *Hugo* have to be explicitly configured. **hyde-x** already shows us a link to the site home, but we have to do some work to get new pages to be visible in the menu. We will edit our **config.toml**



Figure 3: Side bar showing the social links and RSS Feed

file so that it looks as follows:

```
baseurl = "http://replace-this-with-your-hugo-site.com/"
languageCode = "en-us"
title = "Today I Learned"
theme = "hyde-x"
[[menu.main]]
  name = "About"
  url = "/page/about/"
  weight = 2
```

Now, we will see the link to the About page on your homepage. The section `[[menu.main]]` indicates that we are adding this menu to the main menu, has the name "About", which is the text we see, the URL it points to and its weight. Weight decides the order of the menu items.

## Categories and tags

With *Hugo* you can classify your site's content into categories and tags. It refers to these as "taxonomies". We first have to define them in the site's configuration before we can classify our content using these. Here's our **config.toml** after defining the categories and tags:

```
baseurl = "http://replace-this-with-your-hugo-site.com/"
languageCode = "en-us"
title = "Today I Learned"
theme = "hyde-x"
[[menu.main]]
  name = "About"
  url = "/page/about/"
  weight = 2
[taxonomies]
tag = "tags"
category = "categories"
```

The "[taxonomies]" section in the site configuration defines the tag and category for our site.

Now, we can add categories and tags to our existing posts. The first post will now look like this:

```
$ cat content/post/hello-world.md
+++
date = "2016-05-12T08:05:42+10:00"
title = "hello world"
categories = ["updates"]
+++
```

Hello World! This is the first blog post in my brand new blog "Today I Learned". I will be using this blog as a place to post my notes on everyday things I learn.

Figure 5: Demonstration of using the "gist" shortcut to embed a GitHub gist



The second post is updated to the following:

```

+++
date = "2016-05-12T09:20:03+10:00"
title = "Hugo Live Reload"
categories = ["blogging", "hugo"]
tags = ["golang"]
+++

```

*Hugo* has a live reload feature, which means the moment I save this page, not only is my site content regenerated, but the browser also reloads the site. This means, I can see my new post without needing to refresh the site myself.

Now, we can see that our posts have categories assigned (Figure 2), and if you click on a category label, you will find all posts in that category.

### Social, RSS feeds and other customisation

The "hyde-x" theme makes it easy to add links to your social profiles such as your GitHub and Twitter profiles. To do so, we will add a new section "params" to our site configuration and add the following:

```

[params]
github = "https://github.com/amitsaha"
twitter = "https://twitter.com/echorand"
rss = true

```

You will see that there are links to the specified GitHub, Twitter profiles and the RSS feed for your blog contents (Figure 3). The theme also has support for various other social profiles. The RSS feed is for the entire site; if you want to refer to category-specific feeds, you can find them at <http://localhost:1313/categories/<category>/index.xml>.

There's a good chance you will have code in your blog posts, and you want it to be syntax highlighted. *Hugo* has support for two kinds of syntax highlighting – server side or render time (using *Pygments*) and client-side using JavaScript. We will see an example of the latter. Let's create a new post with this content:

```

$ cat content/post/syntax-highlighting-code.md
+++
categories = ["golang", "code", "python"]
date = "2016-05-13T17:02:38+10:00"
description = ""
keywords = []
title = "Syntax highlighting code"
+++
I learned how to post syntax highlighted code in a post.
Hugo has two

```

options when it comes to syntax highlighting - server side and client side.

This is an example of client side highlighting. You can read all about it

[here](http://gohugo.io/extras/highlighting/)(http://gohugo.io/extras/highlighting/).

```

~~~python

```

```

# A line of Python

```

```

print('Hello world')

```

```

~~~

```

```

~~~go

```

```

// A simple program in Golang

```

```

package main

```

```

import (

```

```

    "fmt"

```

```

)

```

```

func main() {

```

```

    fmt.Println("Hello world")

```

```

}

```

```

~~~

```

In addition, we will have to select a syntax highlighting scheme using the **highlight** key in the **params** section:

```

highlight = "zenburn"

```

We will see that the post has syntax highlighted code (Figure 4). There are various highlighting schemes available with **hyde-x**, which you can see in the **themes/hyde-x/static/css/highlight** sub-directory.

### Shortcodes

Shortcodes in *Hugo* enable you to do common things which Markdown doesn't allow (Figure 5). Their syntax is usually **{{< short-code parameter1 parameter2 >}}**. For example, we link to another post or page in our blog, we will use the **ref** shortcode:

```

[post]({{< ref "post/hello-world.md" >}})

```

```

[page]({{< ref "page/about.md" >}})

```

There are various other useful shortcodes – for example to embed a GitHub gist with ID 9864ec0475dd9b68c4a38be37726e552 we will use the **gist** shortcode:

```

{{< gist amitsaha 9864ec0475dd9b68c4a38be37726e552 >}}

```

The first parameter to the **gist** shortcode is the GitHub username, and the second parameter is the Gist ID.

### Hosting your content on GitHub pages

A site is perhaps never done, but I think we are at a point where we are ready to go live. We will deploy our static site using GitHub pages, because it's free and easy to set up. GitHub pages allow two kinds of sites – one of the form **<your-username>.github.io** or **<your-username>.github.io/<repo\_name>**. The first step is to create a repository on GitHub named **linux\_voice\_4** and create a branch **gh-pages** from the repository page. Next, create a local clone of the repository:

```

$ git clone git@github.com:amitsaha/linux_voice_4.git

```

```

$ cd linux_voice_4

```

```

$ git branch

```

## master

We will use the master branch of the repository to keep a copy of our site's "source", and the **gh-pages** branch will have only the generated files. Our next step is to simply move the entire directory tree of **today-i-learned** to the **linux\_voice\_4** subdirectory so that it looks like this:

```
$ tree -L 2 linux_voice_4
```

## linux\_voice\_4

```
├── LICENSE
```

```
├── README.md
```

```
├── archetypes
```

```
├── config.toml
```

```
├── content
```

```
│   └── page
```

```
│   └── post
```

```
├── data
```

```
├── layouts
```

```
├── static
```

```
├── themes
```

```
└── hyde-x
```

It's time to now modify our **config.toml** file to add our base URL:

```
baseurl = "http://<your-github-username>.github.io/
linux_voice_4/"
```

Our next step is to generate the HTML files for our content and add everything to the master branch:

```
$ GOPATH/bin/hugo
```

```
$ git add -A .
```

```
$ git commit -m "Initial version"
```

At this stage, we have our generated site in the **public** subdirectory. We now want to move the contents of that directory to our **gh-pages** branch. We will do it using a straightforward but admittedly naive approach – we will copy the contents to a directory **/tmp/hugo\_public**, checkout the **gh-pages** branch and copy the content from **/tmp/hugo\_public**, then commit everything to the **gh-pages** branch and finally push both the branches:

```
$ cp -r public/ /tmp/hugo_public
```

```
$ git checkout gh-pages
```

```
$ cp -r /tmp/hugo_public/* .
```

```
$ git add -A .
```

```
$ git commit -m "New build"
```

```
$ git push origin master gh-pages
```

If you now visit [https://<your-github-user-name>.github.io/linux\\_voice\\_4](https://<your-github-user-name>.github.io/linux_voice_4), you should see your site's content. Now that our site is live, we want to integrate Google Analytics with it for tracking and Disqus to add the ability to comment on our blog posts (Figure 6).

## Google Analytics and Disqus integration

To add Google Analytics tracking to our page, we create a property on Google Analytics, get our tracking ID and add it as **googleAnalytics = "UA-77766553-1"**. To add Disqus integration to our site, we have to first create an account on Disqus, get the shortname for this site and simply add **disqusShortname = "<you-short-name>"** in the **config.toml** file.

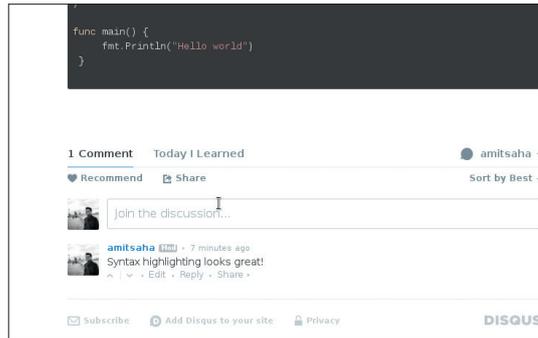


Figure 6: Hugo comes with in-built Disqus integration and the **hyde-x** theme lists the number of comments against each post.

We'll get back our master branch and add the above. The **config.toml** file now looks like this:

```
baseurl = "http://<your-github-username>.github.io/
linux_voice_4/"
```

```
languageCode = "en-us"
```

```
title = "Today I Learned"
```

```
theme = "hyde-x"
```

```
disqusShortname = "hugo-todayilearned"
```

```
googleAnalytics = "UA-77766553-1"
```

```
[[menu.main]]
```

```
name = "About"
```

```
url = "page/about/"
```

```
weight = 2
```

```
[taxonomies]
```

```
tag = "tags"
```

```
category = "categories"
```

```
[params]
```

```
highlight = "zenburn"
```

```
home = "Home"
```

```
github = "https://github.com/amitsaha"
```

```
twitter = "https://twitter.com/echorand"
```

```
rss = true
```

Once we re-publish the site by building the site from master and pushing to our **gh-pages** branch, the posts should have the Disqus commenting system and Google Analytics enabled on all the pages.

## Conclusion

As I explored Hugo while working on the article, I was initially overwhelmed by the features and the need to configure even the smallest of things. However, I have no hesitation in saying that *Hugo* stands out with its huge number of built-in features, and the enormous configurability is a good thing. It includes sensible features by default and puts the content creator in control of how the site should be written, structured and appear.

You can find the source for the "Today I Learned" site at [https://github.com/amitsaha/linux\\_voice\\_4](https://github.com/amitsaha/linux_voice_4) in addition to a set of resources to explore next. The copy of **hyde-x** in my *Git* repository is slightly changed to create the links correctly. I will investigate it further and post updates here: <https://github.com/spf13/hugo/issues/2147>. 

**Amit Saha** is the author of *Doing Math with Python* (No Starch Press) and a software engineer. He blogs at <https://echorand.me>, tweets @echorand and can be reached via email at [amitsaha.in@gmail.com](mailto:amitsaha.in@gmail.com)

# MAKE EBOOKS FROM SCRATCH WITH FOSS

## Birth your literature upon the world the best way possible – the Free way!

ANDREW CONWAY

### Why do this?

- Write an eBook with simplicity and control
- Write with FOSS tools on Linux
- Prepare for any format: eBook, web or print

It was on a dark and stormy night when, suddenly, the velvet drapes shimmered, the candle flickered and you were overcome with a desire to write a book. But how best to do it? You could use *LibreOffice Writer*, or try *Latex*, but perhaps markdown and HTML is an option. Then there's the question of how to publish: you like those old, dusty tomes piled in the corner of the study, but then the Raven quoth "Nevermore!", so an eBook it will be. But how to produce a book for these new fangled eReaders? Read on, but hurry! lest the Cthulhu devour you before you commit your thoughts to ePrint.

It is now the norm for documents to be stored in at least two distinct parts: one contains the content, such as the words in this article; and separate from that is information on style, such as what font to use for the headings. The most obvious example of this is the use of HTML for web content, and CSS to tell a web browser how the content should be displayed.

We're going to use HTML and CSS as the basis for writing our book for a few reasons. First, it means you can use any text editor to write your content, and you can keep style at arm's length while you toil over your choice of words and sculpt your prose to perfection. Also, because proofreading is usefully done using something closer to the final format, you can preview

your work using a web browser and simulate something close to what might appear on an eReader, app or even in the print version.

The third and most important advantage of HTML and CSS is that it's very easy to transform it into almost any format. In fact, one of the most common electronic formats – the EPUB – is a Zip file that includes XHTML and CSS files. Our aim in this article is to build an EPUB manually.

You could use *LibreOffice Writer* to write and format your text and then export HTML and CSS, but that way you will sacrifice both simplicity and control. You do not need to be a web designer to use the HTML and CSS needed to make a book; it's much simpler than putting together a website.

### Markdown

You can write HTML directly. However, after a while you'll start to learn why Markdown was invented. Opening every paragraph with `<p>` and closing it again with `</p>` gets irritating after you've done it a hundred times, and it's prone to error as it's almost inevitable that you'll forget to close such tags at some point.

You're spoilt for choice when it comes to converting Markdown to HTML. I use the Python `markdown_py`; another good choice would be Pandoc, which makes

### Markdown quick start

Markdown has become a favourite format for coders who tire of having to close every tag of HTML. It was created in 2004 by John Gruber with help from Aaron Swartz. In addition to being easy to write, it's designed to be readable as-is, unlike

HTML, which can look quite jumble.

The screenshot below gives you a quick introduction to the basics of Markdown. On the left is the markdown in a text editor (*Kate*), and on the right is HTML generated from it viewed in a web

browser (*Firefox*). If Markdown doesn't support the formatting you want, you can always use HTML tags. See Marco Fioretti's excellent introduction to Markdown in *Linux Voice* issue 10 for more detail and also a list of handy cheatsheets on its syntax.

#### #Markdown

A blank line starts a new paragraph. A # character at the start of a new line is a top level (h1) heading. One \* either side of a word is *italic* and two is **bold**.

##An h2 heading

Lists begin after a blank line and items start with numbers and a period like this:

## Markdown

A blank line starts a new paragraph. A # character at the start of a new line is a top level (h1) heading. One \* either side of a word is *italic* and two is **bold**.

### An h2 heading

Lists begin after a blank line and items start with numbers and a period like this:

a worthy claim to being the Swiss Army knife of text format converters.

There is however one drawback with Markdown: there is no standard. Over the years different implementations have created different syntaxes. The first problem this creates is that a web search on syntax might throw up results that aren't correct for the markdown implementation you are using. The second problem is that Markdown has portability issues: a file put through two different Markdown converters might produce different results.

The best way to avoid these problems is to stick with one Markdown converter and keep your formatting as simple as possible. If you do encounter any peculiarities of the Markdown converter you're using, you can keep your document portable by using HTML tags instead of the offending syntax.

### From laziness comes efficiency

So let's get writing. Enter some Markdown into a text editor. Now save it as **test.markdown** (you can use **.md** if you prefer, but older text editors may think you're writing Modula-2 code and highlight it incorrectly) and generate the HTML with this on the command line

```
markdown_py test.markdown
```

You'll then see the HTML produced as output to the terminal. We'd rather it went to a file, so do this

```
markdown_py test.markdown -f test.html
```

The **-f** option tells **markdown\_py** to send its output into the file **test.html**. You could use the **>** symbol to send the standard output to a file, but this will cause problems for something we'll want to do later.

Now open the file **test.html**. You can do this via your web browser's menu, though as browsers are increasingly intent on hiding menus from you, you might want to type something like this into the URL bar **file:///home/jim/somedir/test.html** – where **jim** is the username and **somedir** is the directory with **test.html** in it. Alternatively you could try dragging **test.html** from your file manager and dropping it onto your browser.

You'll now see your masterpiece rendered with formatting. It's a bit ugly – we'll come to styling CSS later. For now we wish to concentrate on content. Make some edit to **test.markdown**, save it, run **markdown\_py** on it again and then hit reload on your browser to see the results.

Now imagine doing that several hundred times a day. And count the number of times your hand goes from keyboard to mouse and back again. Yes, you can use a few keyboard shortcuts, or even avoid a mouse completely with a tiling window manager, but wouldn't it be nice to just hit Save and see the result appear instantly in your web browser? Let's go WYSIWYG. Not so much What You \*See\* Is What You Get but What You \*Save\* Is What You Get.

We're going to save our wrists from repetitive strain injury by using the nifty command **entr**. (I first heard

about this via a podcast associated with a computing magazine, Minix Vocals I think it was.) First install it if you haven't got it:

```
sudo apt-get install entr
```

Now enter the following in a terminal:

```
ls test.markdown | entr markdown_py test.markdown -f test.html
```

This says: keep an eye on the file **test.markdown**; if it changes, run the command after **entr** which, in our case, will create the **test.html** file. Now make an edit to **test.markdown**, save it and then reload the browser and you'll see your edits are rendered.

But we're only half way to our goal. We want the browser to reload automatically when we save. There

## You can write HTML directly. However, after a while you'll start to learn why Markdown was invented

are two ways to do this. The simplest method is to use *Midori*. No, not the alcoholic liqueur of green hue, but the *Midori* web browser, which you will likely find in your distro's repositories, though it is included in some distributions, notably Raspbian. It has the handy feature that you can instruct it to reload its current tab from the command line. To get this working save the following text into a file called **refresh.sh**:

```
markdown_py test.markdown -f test.html
```

```
midori -e Reload
```

Before running this script, open up **test.html** in *Midori* manually.

We can now use **entr** to run the *Bash* script **refresh.sh** whenever we change **test.markdown**:

```
ls test.markdown | entr sh refresh.sh
```

Try editing the Markdown file and, when you save it, you should see what's shown in *Midori* update accordingly.

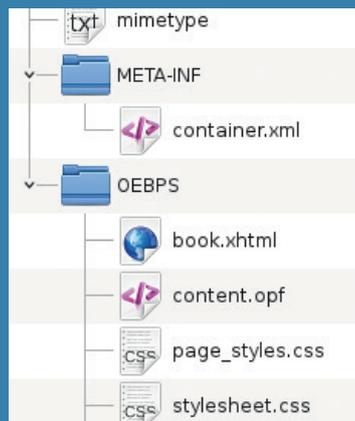
If you'd rather not use *Midori*, perhaps because you don't want to install it for this one purpose, then you can achieve the same thing with almost any web browser. To do this, go to **entr's** website **entrproject.org**, download its **reload-browser** script and use it in place of the **midori -e Reload** command. The script works by faking a Ctrl+R press in your browser window using the **xdotool** command. Bear in mind that it doesn't work in all window managers, and you might need to install **xdotool** from your distro's repositories.

One reason we use *Midori* is so we can dedicate the browser to this one task and keep it separate from all other web browsing. If you use your main browser for the job and leave it on the wrong tab and save your Markdown file, then the **reload-browser** script will merrily reload that tab. The result may be harmless, or it might be irritating, but it might cause a more serious problem if you're half way through entering a long form with important information like bank details.

## Anatomy of an EPUB

An EPUB is a Zip file with a prescribed set of files and directories, as shown here. The mimetype must be the first file in the archive and must not be compressed. This enables applications to quickly determine that the file is an EPUB. Most of the files in the EPUB are XML files, with exceptions being the CSS file and content files such as images, audio or video. The main content of the book is supplied in one or more XHTML files using HTML 5 syntax. You can have one file for an entire book, but it's more usual to split it into a file per chapter, or even per section.

The `container.xml`'s main purpose is to point to `content.opf` (though it can be given any name) which contains a manifest of all files in the EPUB, along with the order in which the main parts of the book are meant to be read, and references to navigation elements. In the current EPUB 3 format, navigation menus in applications and eReaders will use a file that's usually called `toc.xhtml`. This is a change from EPUB 2, which used an XML `.ncx` file to specify the navigation structure. As many eReaders and applications still use EPUB 2, it's still a good idea to include an NCX `.ncx`, but an EPUB 3 reader will ignore it and expect to find `toc.xhtml`. The full EPUB specification can be found at [www.idpf.org/epub](http://www.idpf.org/epub).



Now let's pull together all the bits and pieces we need to make a book. There's not space to include all the lines of XML needed in this article so we omitted the most boring ones, but you can find complete versions of all files mentioned at [github.com/mcnaul/linuxvoice-publishing](https://github.com/mcnaul/linuxvoice-publishing). First, we need to augment our script so that we create a proper HTML document with CSS styling, and also handle multiple chapters. Here we've just included two to demonstrate the principle: `chapter1.markdown` and `chapter2.markdown`. Create a text file called `make_book.sh` containing this:

```

#Insert beginning of HTML file and timestamp
cat head.html > mybook.html
echo "<p class='centre'>Generated: " >> mybook.html
date >> mybook.html
echo "</p>" >> mybook.html
#Create one big markdown file
echo "##Contents" > mybook.markdown
echo "[TOC]" >> mybook.markdown
cat chapter1.markdown >> mybook.markdown
cat chapter2.markdown >> mybook.markdown
#Create the html with a table of contents
markdown_py -x toc mybook.markdown >> mybook.html
#Finish off the HTML file
echo "</body></html>" >> mybook.html
midori -e Reload
  
```

The `head.html` should look something like this:

```

<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:epub="http://www.idpf.org/2007/ops">
<head>
<title>My Book</title>
<link rel="stylesheet" type="text/css" href="mybook.
css"/>
</head>
  
```

```

<body>
<h1 class="centre">My Book</h1>
<p class="centre">A.N. Author</p>
<p class="centre">Copyright 2016 A.N. Author </p>
  
```

This contains the tags needed at the start of a valid XHTML file, which we'll need for the EPUB later, and it specifies the character encoding as UTF-8 and the CSS file as `mybook.css` in which the `centre` class we use in `head.html` is defined. Then the body of the HTML starts and a basic title page is created with a title, author and copyright information.

The secret of writing the CSS file for an e-document is to keep it simple. Here's a CSS file that's all you need for simple but decent eBook:

```

body {margin: 3em 3em 3em 3em;}
h2 {page-break-before: always;}
h3 {page-break-after: avoid;}
.centre {text-align: center;}
p {text-indent: 1em;}
  
```

First notice that all sizes are in units of em, where 1em is equal to the font size, and 3em means three times the font size. Using pixel size px would be very naughty in this context and is likely to cause unexpected weirdness on some eReaders and unpredictable results when printed. Next, we say that there should always be a page break before the `h2` style, because it will be used for chapter headings. For headings within chapters we use `h3`, and we want to avoid breaks after them if possible. This will avoid having headings near the bottom of a page. These page-break styles will have no effect in a web browser, but will in an eReader or if you print the HTML from your browser.

The `.centre` defines a class that was used in `head.html` to centre text on the title page. The text-indent for `p` specifies that each paragraph should start with a small indent, as is conventional in print and eBooks.

We can generate a new preview of our book when any part of it is saved using the `entr` command:

```
ls *.markdown mybook.css | entr sh make_book.sh
```

This triggers a rebuild of the book if any of the markdown files change or the CSS file changes.

At this point, assuming you've written some text for chapters 1 and 2, you will have a book that you can view in a web browser. Next we'll show how to wrap this up in a valid eBook format.

## Prepare to publish

Of the very many eReader formats, probably the most widely used is EPUB, though arguably the dominance of Amazon's Kindle means that many people are using the AZW format without knowing it. AZW is based on the older MOBI format, but since 2011 Amazon has been using its newer Kindle Format 8 (`.fd8` or `.azw3`), which is based on HTML 5 and CSS.

We'll concentrate on how to manually create a version 3 EPUB. Have a look at the boxout on the anatomy of EPUB for an overview before reading on.

First, in an empty directory, create the basic structure of what needs to go into the EPUB file:

```
echo -n "application/epub+zip" > mimetype
```

```
mkdir META-INF OEBPS
```

```
cp /some/path/mybook.html OEBPS/mybook.xhtml
```

```
cp /some/path/mybook.css OEBPS
```

Notice we've used **-n**, which tells **echo** not to add a newline character. This is important because the **mimetype** file must not have more than one line. Also, notice that we renamed the **mybook.html** to **mybook.xhtml**, which is conventional in EPUB 3.

Now create a file called **container.xml** inside the **META-INF** directory that contains this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/content.opf" media-type="application/oebps-package+xml"/>
  </rootfiles>
</container>
```

The main task of **container.xml** is to point to the **content.opf** file. This is also an XML file, and its important lines are:

```
<manifest>
  <item id="toc" properties="nav" href="toc.xhtml" media-type="application/xhtml+xml" />
  <item id="template_css" href="mybook.css" media-type="text/css" />
  <item id="mybook" href="mybook.xhtml" media-type="application/xhtml+xml" />
</manifest>
<spine>
  <itemref idref="mybook" />
</spine>
```

The manifest lists the contents of the EPUB file: the table of contents (**toc**), the CSS file and the main content file. The spine is used to list the linear reading order of the document, though in our case we've opted for only one big XHTML file. If we had one **.xhtml** file per chapter, the spine section would need an **itemref** for each one.

The final file we'll need is **toc.xhtml**. The important bit of this is this:

```
<nav id="toc" epub:type="toc">
  <h1 class="frontmatter">Table of Contents</h1>
  <ol class="contents">
    <li><a href="mybook.xhtml#contents">Contents</a></li>
    <li><a href="mybook.xhtml#chapter-1">Chapter 1</a></li>
    <li><a href="mybook.xhtml#chapter-2">Chapter 2</a></li>
  </ol>
</nav>
```

You can save yourself some time by looking inside **mybook.xhtml** for the table of contents that **markdown\_py** generated. Those **<li>** items can be copy and pasted into **toc.xhtml**, although you'll have to add in **mybook.xhtml** before the **#** in each **href** link. The **toc.xhtml** is separate to the table of contents we've already put at the start of the book and will be

used to enable eReaders to offer a navigation menu no matter where you are in the book. For an eBook, there's probably no reason to have both, but it does no harm to leave it in and most eReaders will respond to the links as expected.

Now that all the files are in place we can create the EPUB file as follows:

```
zip -0X mybook.epub mimetype
```

```
zip -Xr9D mybook.epub META-INF/ OEBPS/
```

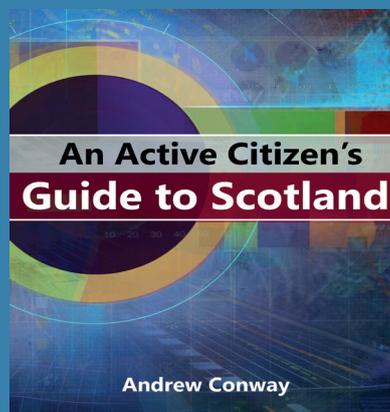
The first **zip** command is performed separately to ensure that **mimetype** is the first file in the Zip archive, and the **-0** option (that's a zero) tells Zip not to compress that file. This is important so that applications can easily find and read the mimetype. The second command compresses and stores all other directories and files in the archive. The **X** stops extra file attributes being included in the archive; **r** tells it to include files recursively in sub-directories; **9** means maximum compression and **D** omits separate entries for directories.

You can now open the EPUB in any application and check that it's displaying correctly. If you want to check you've structured your EPUB correctly, you can download the *EpubCheck* tool from <https://github.com/IDPF/epubcheck>.

## Epilogue

Hopefully this has given you an understanding of how an electronic document is structured and that by using Markdown and keeping a light touch with CSS styling, you can keep your writing process simple while staying in control of the finished product. That said, the process of creating an EPUB manually is rather fiddly. You can automate it further by writing your own scripts to copy files into the directory structure and zip up the EPUB, but another option is to use the *Calibre* application to generate the EPUB. We'll describe *Calibre* and various ways you can publish an eBook and a print version in a follow-on article. 📖

### A real example



The techniques in this article are not just some theoretical musings, but those I have used to produce a real book called *An Active Citizen's Guide to Scotland* (see [activecitizen.scot](#) if you're interested). All the text for this book was written in Markdown in KDE's *Kate* text editor, and tables and figures were mostly generated using *LibreOffice Calc* with a few being made with *Gnuplot*. Although the *Bash* script used to construct the book was a little more complex than the one shown here, it is still pretty short at 43 lines.

Some publishers may insist on their authors providing a particular format, too often **.docx**, but, if you make it as a bankable, best-selling author like JK Rowling, you could, if you so wished, submit your manuscript scrawled by hand in purple ink on the back of a thousand fag packets.

**Andrew Conway** watches the stars from his wood-panelled study. He likes open data and what you can do with it using Free Software.

# RASPBERRY PI DISPLAY DATA PHYSICALLY

Turn input from the internet into something you can hold in your hand.

LES POUNDER

## Why do this?

- Make the most out of GPIO Zero
- Control a stepper motor
- Work with external data sources

## You will need

- Any model of Raspberry Pi
- Wi-Fi connection
- Stepper motor
- LEDs
- 220Ω resistor (RED-RED-BROWN)
- Breadboard
- Male–female jumper wire
- Female–female jumper wire
- Power for your Raspberry Pi

The OpenWeatherMap website has a great suite of tools for you to research historical and forecast future weather.

Summer time is upon us in the northern hemisphere, and the weather is always a keenly discussed topic of conversation. Is it warm enough to get the barbecue out? Being the chief question. Using a little GPIO Zero code, a stepper motor and LEDs plus a Python weather module, we can build ourselves a cool weather frame that will keep us up to date. There are two parts to the hardware build. First we have the components, chiefly our stepper motor and eight LEDs. Our stepper motor has a driver board using a ULN2003 chip. The board requires 5V of power from our Raspberry Pi and a Ground (GND) connection. We can then connect the other “input” pins of the board to our Raspberry Pi. We can also attach the LEDs to the corresponding pins of our Raspberry Pi via a 220Ω resistor.

The second part of the build is the physical frame. We chose to use a cheap picture frame and replaced the glass with a piece of cardboard that fit into place with a gentle push. We then stuck some coloured card to the cardboard to hide the backing. Next we measure out a dial using an old CD and then split the circumference of the dial into segments for our temperature range. To attach the stepper motor to the cardboard we used some machine screws and nuts. But to attach the dial to our stepper we used a lollipop stick into which we cut a notch that matched the stepper motor spindle. By attaching the lollipop stick to our dial using tape and then sliding on to the spindle, we now have a reliable method of precise rotational movement. We then attached another dial to the frame, but this time with no stepper motor. In this dial we inserted 8 LEDs.



Our completed project sits inside its picture frame home ready to inform us of the next change in weather.

We start by opening a Terminal window, the icon for which is found in the top-left of the screen and looks like a black screen.

Our first task is to install the Python library for OpenWeatherMap. In the terminal type the following and press Enter.

```
$ sudo pip3 install pyowm
```

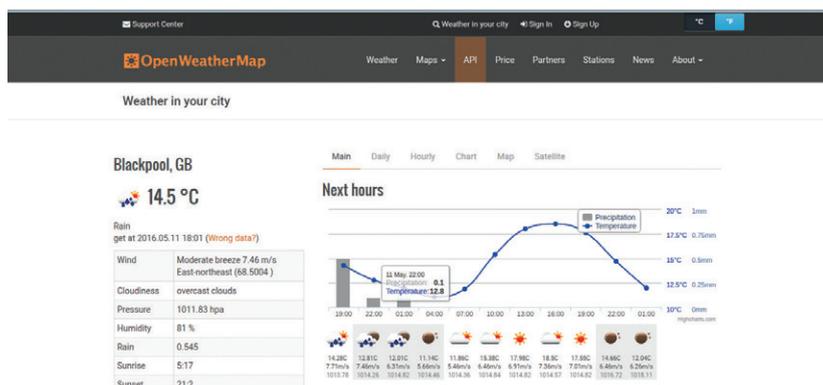
With installation complete we now need to sign up to the OpenWeatherMap service via its website. In a browser visit [https://home.openweathermap.org/users/sign\\_up](https://home.openweathermap.org/users/sign_up) and create a new account. This will generate an API key, which will enable our project to request weather data. Keep this key handy, as we will need it later in the project. Also note that this key is private and linked to your account, so do not share your API key.

## Coding the project

We start coding the project by opening the Python 3 application, found in the Programming menu. With Python 3 open click on File > File New to open a new blank document. Immediately save your work as **Weather\_Frame.py**. This will enable quick saving as we work through the code.

Our first section of Python code is a series of imports. First we import two classes from GPIO Zero, namely **LED** to handle our LED indicators, and **OutputDevice**, a class enables us to directly control GPIO pins.

```
from gpiozero import LED, OutputDevice
import pyowm
```



**import time**

We next create variables that will be used to identify the LEDs for the different weather statuses that we wish to identify. The GPIO Zero **LED** class is remarkably easy to use and requires little code for configuration.

**Sunny = LED(17)**

**Bluesky = LED(27)**

**Snow = LED(22)**

**Cloudy = LED(10)**

**Thunder = LED(9)**

**Shower = LED(11)**

**Rain = LED(5)**

**Fog = LED(6)**

Next we use another class, **OutputDevice**, from GPIO Zero. The class enables control of any GPIO pin. Here we use it to create four pins that will be used to control our stepper motor.

**Temp\_IN1 = OutputDevice(23)**

**Temp\_IN2 = OutputDevice(24)**

**Temp\_IN3 = OutputDevice(25)**

**Temp\_IN4 = OutputDevice(8)**

Finally we create a new variable called **delay** that is used to control the stepper motor speed. This is the optimum time to run the stepper smoothly.

**delay = 0.01**

We now create a function that will control our stepper motor. This function is called **cw**, short for clockwise, and it takes three arguments: the number of steps to move; the delay between each step; and the stepper motor to control. This function can control multiple stepper motors, but for this project we use just the one.

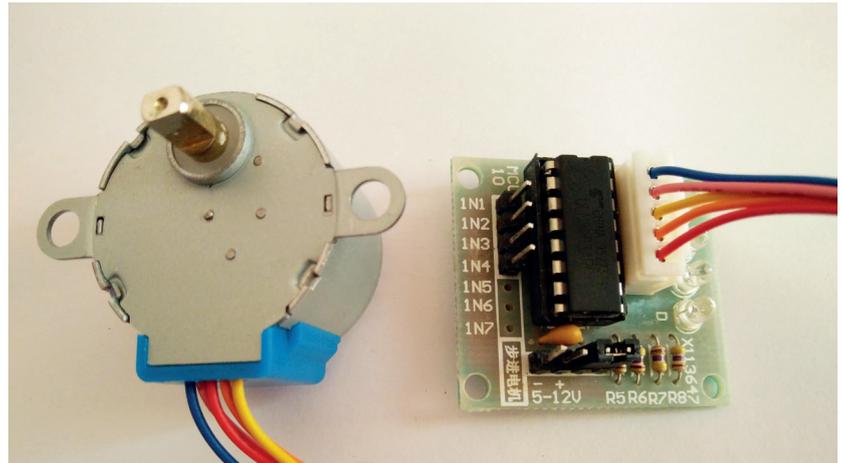
**def cw(steps,delay,stepper):**

**Work with remote data: OpenWeatherMap**

Python has many methods of working with external data, one of which is OpenWeatherMap.

OpenWeatherMap is a free resource of worldwide weather data that has an extensive API for many languages. The OpenWeatherMap website details the many uses of this project. It can be used as a simple website to query historical and forecasted weather data, but by using an API for Python, it enables our Raspberry Pi to receive remote data that will ultimately control our appliance, creating a physical appliance that can show us the data in an unusual way. By merging remote data with physical computing we can create new appliances and even use data to drive art installations. In our tutorial we used **pyowm** from <https://github.com/csparpa/pyowm>, which has now been packaged ready for use with the Python package manager **pip3**.

There are many more modules that can pull data from other services, for example eBay and newsfeeds. Another source of external data is IFTTT, short for "If This, Then That". IFTTT is a trigger- and event-based system that can link into many web services and your mobile device. You can even create a location-based trigger that will turn on your TV ready for when you get home. You can learn more about IFTTT at <https://ifttt.com>.



Inside this function we have an **if** condition that checks the name of the stepper motor to be used. In this case it is **Temp** for our temperature gauge. This condition is indented to show that it is part of the function.

**if stepper == "Temp":**

Next we create another indentation, which we use to construct a **for** loop. This loop will pulse the GPIO pins connected to our stepper motor in the correct sequence to drive the stepper clockwise. At each time one pin is turned on while the rest are off, causing the stepper to move one step. By changing the pins in quick succession we can create a smooth rotation. In the code we only partially show the sequence, which can be seen in the in the project files downloadable from our GitHub page.

**for i in range(steps):**

**Temp\_IN1.on()**

**Temp\_IN2.off()**

**Temp\_IN3.off()**

**Temp\_IN4.off()**

**time.sleep(delay)**

**Temp\_IN1.off()**

**Temp\_IN2.on()**

**Temp\_IN3.off()**

**Temp\_IN4.off()**

**time.sleep(delay)**

....

With that we end this function by making a new line under the function, and ensuring that our cursor is to the left of the window.

We now create a new function, this time called **ccw**, Counter Clockwise. This function has the same arguments as **cw**, but reverses the sequence to force the stepper to rotate counter clockwise. Again, not all of the code is shown due to its length:

**def ccw(steps,delay,stepper):**

**if stepper == "Temp":**

**for i in range(steps):**

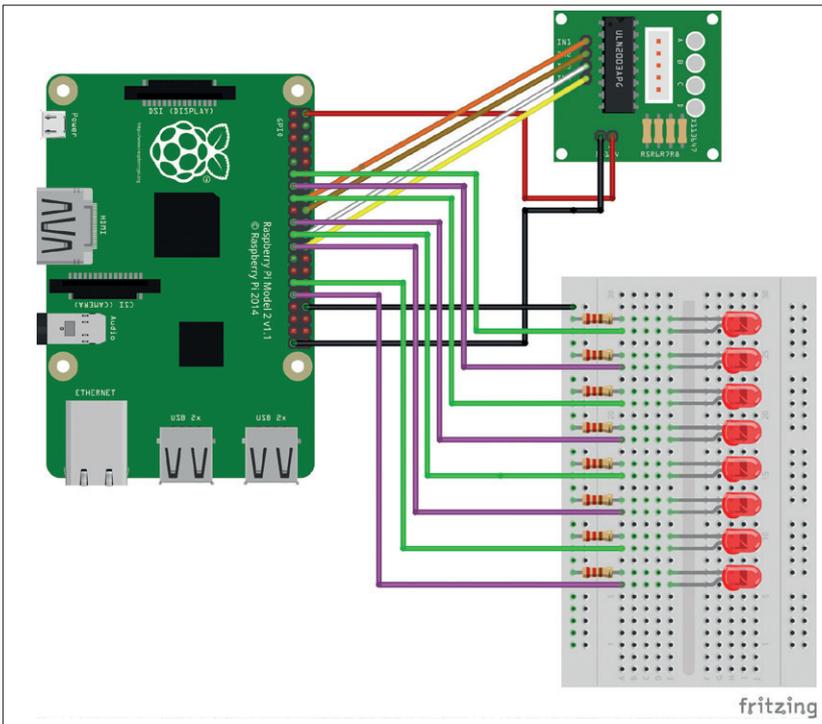
**Temp\_IN1.off()**

**Temp\_IN2.off()**

**Temp\_IN3.off()**

**Temp\_IN4.on()**

Stepper motors normally come as just a motor, but we picked these up cheaply from eBay, which come with their own controller board.



Our circuit involves quite a few wires, for best results build and test segments of the circuit as you go. It is easier to debug a hardware issue.

```

time.sleep(delay)
Temp_IN1.off()
Temp_IN2.off()
Temp_IN3.on()
Temp_IN4.off()
time.sleep(delay)
....
    
```

We now close this function and create a new function called **get\_weather**. This function uses the OpenWeatherMap API (Application Programming Interface) via a Python module. Using this module we can check the weather for any location across the globe. This function takes one argument, **n**, which represents the location where we would like to know

## Using the OpenWeatherMap API via Python we can check the weather for any location across the globe

the weather.

```

def get_weather(n):
    Indented in our function we create a variable called
    owm, and in here we store output of connecting to the
    OpenWeatherMap API using the Python module and our secret
    API key. We then create a variable called observation,
    which will get the weather data for your chosen location,
    via the n argument.
    owm = pyowm.OWM("YOUR SECRET API KEY")
    observation = owm.weather_at_place(n)
    
```

Still in the **get\_weather** function we now create a dictionary, a Python data structure that can store data with associated keys. In our project we use a dictionary to store the numerical code given by OpenWeatherMap for different weather conditions as

the key, which in turn will return the correct weather condition as a string. A dictionary uses **{}** to contain the data. Our keys are identified by a colon, **:**, and our data are strings containing weather status.

```

codes = {
211:"thunderstorm",
313:"shower rain and drizzle",
321:"shower drizzle",
500:"light rain",
...
}
    
```

Next we create a variable called **w**, which is used to temporarily store weather data.

```

w = observation.get_weather()
    From this variable we now obtain the temperature from our
    chosen location, which we then store in a new data structure
    called a. The new data structure is a dictionary and we're
    looking for the key temp, as in temperature. On the next
    line of code we wrap the extracted data in a function that
    will convert the data into an integer. All of this is then
    contained in a variable called a.
    
```

```

a = w.get_temperature('celsius')
a = int(a['temp'])
    
```

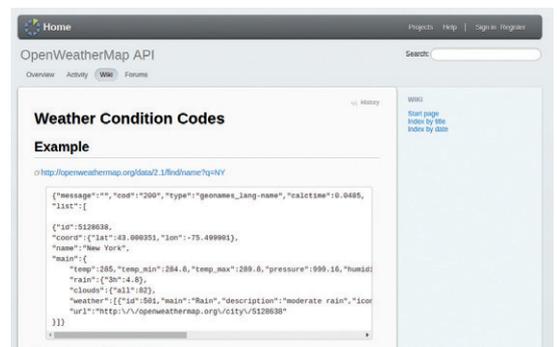
Next we create a variable called **b**, and in there we store the weather code that matches the current weather at our chosen location. This will be used later with our weather codes dictionary.

```

b = w.get_weather_code()
    To confirm that the data stored in variables a and b
    is correct we print the contents to the Python shell. This
    step can be removed once the data has been confirmed
    correct.
    print(a)
print(b)
    
```

### Else, if

Still inside our function, we now create a series of conditional tests that will check the value of the variable **a**, which is the temperature of our chosen location. If the temperature matches **b** one of the tests, then it is considered True, and the code that relates to that condition is executed. Our first test uses **if** and checks the value of **a** against the value **-5** as in **-5C**. If

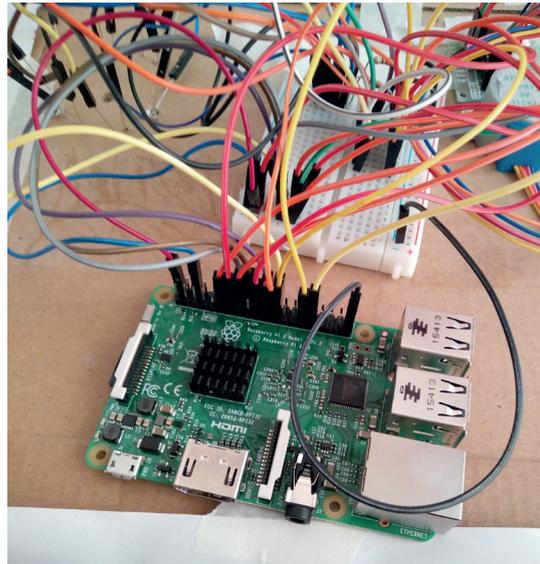


We used the weather codes from [http://bugs.openweathermap.org/projects/api/wiki/Weather\\_Condition\\_Codes](http://bugs.openweathermap.org/projects/api/wiki/Weather_Condition_Codes) to generate our dictionary of weather conditions.

## Working with motors

There are many different types of motors on the market. The first motors we typically come across are simple DC motors. These are really cheap and easy to use but not very precise. Next we have micro gear metal motors, which are typically geared to a set ratio – in other words the lower the ratio, the faster they spin. A high-ratio motor produces slow movement and plenty of pulling power for larger chassis. Stepper motors are slow motors, but they are very precise, in fact they are commonly used in DVD drives, scanners and printers, devices which require precision movement.

Every motor needs some form of controller, and should never be connected to the GPIO directly (this can cause damage to your Raspberry Pi). For the stepper motor we bought two units from eBay. These units came with their own controller boards built around the ULN2003, a high-voltage and high-current controller. The ULN2003 receives power from the Raspberry Pi and then provides it to the stepper motor at a higher rate of current, much more than the Pi can normally supply. Typically you can only power one stepper motor from the Raspberry Pi; if you need to power more, an external 5V power supply is required. You could use a 4 x AA battery box or hack a USB lead from a USB battery to provide the necessary current. Just remember to connect the GND of your battery to the GND of your Raspberry Pi



This project can be built with any Raspberry Pi. We chose to use the Pi 3 as it has WIFI built in, but this project could be built using a Pi Zero.

that condition is true then we call the **ccw** function with the arguments denoting 11 steps, delay variable, and control our "Temp" stepper motor. This will cause the stepper to spin and rotate the temperature dial for 11 steps, taking us to -5C on the dial. Then after five seconds we call the **cw** function to rotate the stepper motor back to its original position.

```
if a == -5:
    ccw(11,delay,"Temp")
    time.sleep(5)
    cw(11,delay,"Temp")
```

## Define a temperature range

We cover a temperature range of -5C to +40C and for each temperature we require a conditional test. For subsequent tests we will use "else if", shortened to **elif** in Python. If the first condition is not true, then the tests will continue until a test evaluates as **True**, or if none of the tests work then the **else** condition must be **True**, in which case it forces the code to wait for five seconds before repeating the process. Here we see a snippet of the code.

```
....
elif a == 40:
    ccw(506,delay,"Temp")
    time.sleep(5)
    cw(506,delay,"Temp")
else:
    time.sleep(5)
```

This ends the conditional tests against the temperature data. Now we create a new series of tests that will check the weather code saved in the variable **b** against the values hard-coded in our tests. Again we start with an **if** condition, but this time we control our LEDs. At the start of the code we created

variables for each of the LEDs using the **LED** class from GPIO Zero. So now we can turn an LED on or off by calling its name followed by **on** or **off**. For our first condition, if the weather code returns "211" then the LED to indicate a Thunderstorm will illuminate for five seconds before turning off.

```
if b == 211:
    Thunder.on()
    time.sleep(5)
    Thunder.off()
```

The code continues with a series of **elif** statements, each testing the value of **b**. Our final test is **else**, and we use this to indicate that it is sunny, as every other test has evaluated as **False**, so **Else** must be **True**.

```
elif b == 804:
    Cloudy.on()
    time.sleep(5)
    Cloudy.off()
Else:
    Sunny.on()
    time.sleep(5)
    Sunny.off()
```

With the functions completed we now go to our main body of code. Here we use a **While True** loop to constantly run our code. Inside the loop we call our **get\_weather** function with our location as an argument. In my case that's "Blackpool, UK". This calls all of the code that we have written previously. Once the functions have run, the loop sleeps for 15 minutes before repeating the process.

```
while True:
    get_weather("Blackpool, UK")
    time.sleep(900)
```

With the code and hardware complete, save your work and click on the Run menu, then click on Run Module to run the code. You will see the stepper motor come to life and after a few seconds the LED weather indicator will illuminate the current status. 

**Les Pounder** makes things, breaks things, and spends the rest of his time teaching teachers about the new IT curriculum.

# ROLL YOUR OWN ENCRYPTION WITH RSA

Write your own crypto tool and understand how the web stays secure.

JOHN LANE

## Why do this?

- Deepen your faith that the crypto people know what they're doing.
- Show off with some hard maths.

Finding factors isn't the way to crack RSA: it doesn't take a very long key for the time to factor it to be noticeable.

**Y**ou are protected by encryption whenever you access a secure website on the internet, and you may know this as SSL. Most secure web sessions begin with a handshake that relies on an asymmetric cryptographic algorithm called RSA. It's also used for other things, but most people will encounter it through their web browsing activities.

We explored how SSL works in issue eight (you can download it from <https://www.linuxvoice.com/issues/008/ssl.pdf>) and how it begins with an encrypted key exchange. RSA is used to perform that exchange securely, and how it does that is the subject of this tutorial.

RSA was invented in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman – hence RSA. It is the most popular asymmetric cryptosystem in use today. When we say that it's asymmetric, we mean that it uses different keys to encrypt and decrypt a message. We call these keys public and private.

The first thing to understand is that cryptography is a mathematical problem; a numbers game. When we talk about a message we mean a number. A potentially large number with hundreds of digits, but a number nonetheless. To convey a message of words, it first needs to be converted into a number. Fortunately, computers are good at that and we have codes like ASCII for specifically that purpose. So we'll put words to one side until our final example at the end and concentrate on the numbers for now. And we'll begin by using small ones to keep things simple.

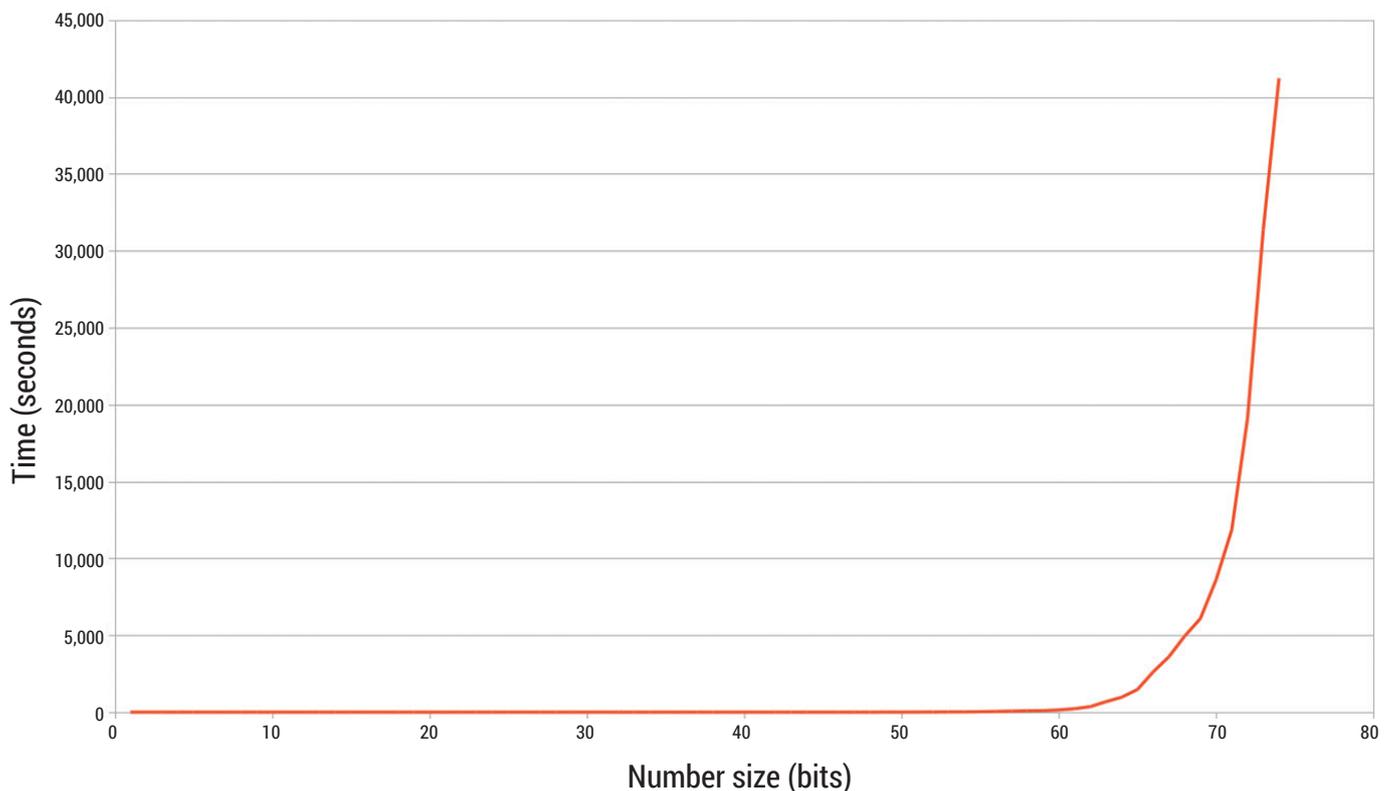
RSA provides a formula to convert one number (the plaintext) into another (the ciphertext). It looks like this:

$$y \equiv x^e \pmod{n}$$

Another formula is used to decrypt. It is very similar:

$$x \equiv y^d \pmod{n}$$

And that's it. In fact, they're the same function that we can represent in code (our examples are in Ruby):



Listing 1: the algorithms "rsa.rb"

```

0 #!/usr/bin/ruby
1 def gcd(a,b)
2   (r = a % b) == 0 ? b : gcd(b,r)
3 end
4 def phi(*args)
5   case args.length
6   when 2
7     (args[0]-1)*(args[1]-1)
8   when 1
9     n = args[0]
10    (1..n).reduce(0) { |p,i| gcd(n,i)==1 ? p+1 : p }
11 end
12 end
13 def eea(a,b)
14   return b==0 ? [1,b] : begin
15     q, r = a.divmod b
16     s, t = eea(b,r)
17     [t, s - q * t]
18 end
19 end
20 def key(e, *args)
21   case args.length
22   when 1
23     n = args[0]
24     t = phi(n)
25   when 2
26     p, q = *args
27     n = p*q
28     t = phi(p,q)
29 end
30 d = eea(e,t).first % t
31 [d, n]
32 end
33 def crypt(m, e, n)
34   m**e % n
35 end

```

def crypt(m,e,n)

```

m**e % n
end

```

You could now encrypt a plaintext message **x** into ciphertext **y** if you know what **e** and **n** are. If you sent that ciphertext **y** to someone else and they knew **d** and **n** then they would be able to decrypt **y** to recover the plaintext message **x**. But we don't yet know what those values are; this is what they mean:

- **e** is the public exponent and it's part of the public key, along with **n**.
- **d** is the private exponent and, along with **n**, forms the private key.
- **n** is called the modulus, and the **mod** in the formulae indicate they're using modular arithmetic (see the box if you need that explained).

We are free to choose what **n** is, but our choice defines the biggest number that can be encrypted. This is because all our calculations are done **mod n**, which means that we can encrypt between **0** and **n-1**. We use the size of **n** in binary digits (bits) to describe the key size. Real-world keys should be at least 2048 bits to be considered secure (that's a 617-digit number).

Before choosing **e** and **d**, think about the encryption and decryption formulae and that decryption undoes encryption. This implies that you should be able to combine them like this:

$$x \equiv (x^e)^d \pmod{n}$$

which is

$$x \equiv x^{ed} \pmod{n}$$

And you can, and you get **ed=1**. But it doesn't follow that **d=1/e**. No, if it were that simple then we wouldn't have encryption. And it isn't that simple because of the modulus, and there are no fractions in modular arithmetic.

So we need to find an integer value for **d** that gives 1 when multiplied by **e**, and we say that value **d** is the modular multiplicative inverse of **e**.

How can we do that? Well, our formula is very similar to what is known as Euler's Theorem, which looks like this:

$$x^{\phi(n)} \equiv 1 \pmod{n}$$

where  $\phi(n)$  (the Greek character  $\phi$ ) is Euler's Totient Function. We'll talk more about that later but, for now, it's interesting because we can use it to write an equation to help solve **d**. If we multiply both sides by **x** we have

$$x^{\phi(n)}x = x^{\phi(n)+1} = x^{ed} \equiv x \pmod{n}$$

Now, if you consider that the only variable is the exponent, you can say that

$$ed \equiv \phi(n) + 1 \pmod{\phi(n)}$$

Note how the modulus becomes  $\phi(n)$ , because

All our calculations are done mod **n**, which means that we can encrypt between 0 and **n-1**

that's the modulus of exponents in a modulo **n** formula (Euler's Theorem proves this if you are interested). We can replace the modulus with some multiple, **k**, of  $\phi(n)$ :

$$ed = k\phi(n) + 1$$

$$ed + k\phi(n) = 1$$

We now have a formula in the style of Bézout's Identity. This is another theorem in the elementary theory of numbers and it states that, for two integers **a** and **b**, and their greatest common divisor **c**, there exist integers **x** and **y** such that:

$$ax + by = c$$

and **x** and **y** are called the Bézout Coefficients of **a** and **b** and, if those are **e** and  $\phi(n)$  then **x** will be the private exponent, **d**, that we seek.

To make this work out for us, the theorem requires that the greatest common divisor, **c**, of **e** and  $\phi(n)$  is 1 or, in other words, that they are coprime. While we're

### Modular arithmetic

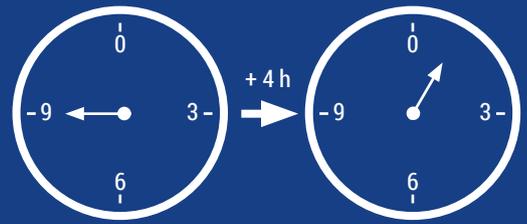
Modular arithmetic is way of counting integers where they wrap around upon reaching a value that we call the "Modulus". Time and, more specifically, clocks provide an easy to understand real-world example of everyday use.

Modular formulas are suffixed with **(mod n)** to indicate they are modulo n and the triple-bar congruence ( $\equiv$ ) symbol indicates modular equivalence, which is like equals (=) in the modulo arithmetic world.

A congruence like  $15 \equiv 3 \pmod{12}$  can be written as  $12k+15=3$  where **k** is any positive integer; any multiple (**k**) of the modulus (**12**) is congruent (equivalent) to 0 (mod 12). So the integers 3, 15, 27 and so-on are congruent in modulo 12.

Another way to look at it is as the remainder in a whole-number division, so  $9+4=13$  and  $13 / 12=1$  with a remainder of 1. There's a modulus operator that takes two arguments and returns the remainder like  $15 \pmod{12} = 3$ . Note that the **mod** operator's result is an equality (=) rather than a congruence ( $\equiv$ ). Many programming languages use a % operator for this:  $15 \% 12 = 3$ .

While we're talking division – watch out: modular arithmetic doesn't have division but, instead, has modular reduction.



Time is modulo 12 making one o'clock four hours later than nine:  $9+4 \equiv 1 \pmod{12}$

(CC BY-SA): [https://en.wikipedia.org/wiki/File:Clock\\_group.svg](https://en.wikipedia.org/wiki/File:Clock_group.svg)

Consider that  $4 \equiv 14 \pmod{10}$  but  $4/2 \not\equiv 14/2 \pmod{10}$ . Modular arithmetic is constrained to integers – there are no fractions.

Another thing to watch out for with modular reduction is that exponents must be reduced by a different modulus: the. The algorithms used by RSA are based in modular arithmetic, albeit with a very large modulus!

free to use any value for **e** within that constraint, most real-world implementations choose one of the so-called Fermat Primes (3, 5, 17, 257 and 65,537) with 65,537 being the most popular.

So, all we have to do to find **d** is to solve Bézout's Identity but, before we can do that, we need to find  $\phi(n)$ .

### The Phi Factor

In 1640, Pierre de Fermat stated Fermat's Little Theorem for prime numbers. A hundred years later, a Swiss mathematician called Leonhard Euler

We say it's a "hard" thing to do, not because the algorithm is difficult, but because it would require a disproportionate amount of time (and computer power) to complete.

But there are numbers where iteration can be avoided: prime numbers. Primes cannot be divided, so everything less than a prime is coprime to it. In other words, if **n** is prime then  $\phi(n) = n-1$  (which is what Fermat had said originally).

Still, anyone knowing **n** (remember that **n** forms part of the public key) would be able to quickly determine the private key if **n** was prime. So we really don't want **n** to be prime; we need a composite number.

We can use another property of the totient to efficiently find it for a composite number: it's multiplicative. This means, for two numbers **p** and **q**, that:

$$\phi(pq) = \phi(p)\phi(q)$$

Now, if **p** and **q** are prime then the totient of **pq** will be  $(p-1)(q-1)$ . This means that you can quickly calculate the totient of a number if you can find its prime factors. But finding factors of large numbers is also hard. So we choose the prime numbers and multiply them to make **n**. We keep them secret (as part of the private key) and that's what keeps your online banking account secure.

So we can now find the totient,  $\phi(n)$ , either iteratively for small values of **n** or, for any value that is composed as the product of known prime numbers. If we only use two primes then it'll be even more difficult to find them by factoring **n**.

We'll call our two prime factors **p** and **q**, and can obtain them from any suitable source such as

- OpenSSL, which includes a command-line tool that can generate 16-bit or larger primes: "openssl prime -generate -bits 20".

- The Wolfram Alpha computational knowledge

You can find the greatest common divisor using an algorithm that was documented over 2000 years ago

(pronounced Oiler) generalised Fermat's theorem so that it could also be applied to non-prime (composite) numbers. Euler's Theorem defines the "totient" of a number **n** to be the count of the positive integers that are less-than **n** and are coprime with **n**.

The totient can be derived by iterating over the numbers between 1 and **n** and increasing a total for any number found to be coprime with **n** – that is any number that has a greatest common divisor (gcd) of 1 with **n**. You can find the gcd using an algorithm that was documented over two thousand years

ago by the Greek mathematician, Euclid (see the ancient algorithms box).

The totient is the key to the security of the RSA algorithm because computing the totient iteratively would take too long for large numbers.

#### PRO TIP

A United States patent for RSA was granted in 1983 and expired in 2000. You can read it at <https://www.google.com/patents/US4405829>.

engine (wolframalpha.com); a request like "randomprime(1<<2047)" will produce a 2048-bit prime.

It doesn't matter where the factors come from, as long as they are prime. The bit length of **n** is the sum of the bit lengths of **p** and **q**; you can use half the desired bit length of **n** for the factors (so two 1024-bit factors would give a 2048-bit key).

Now that we have **e** and **n** (or its prime factors **p** and **q**), we can find the totient **t**. Listing 1 presents everything we need:

- **gcd** (lines 1–3) uses the Euclidean Algorithm to find the greatest common divisor of two numbers **a** and **b**, and

- **phi** (lines 4–12) returns the totient of one number by iteration or of two numbers (assumed prime) using the totient formula.

The **key** method (lines 20–32) generates a private key from a public exponent, **e**, and a modulus that can be either a number, **n**, or its prime factors **p,q**. It first uses **phi** to get the totient, **t**, of the modulus and then uses a variant of Euclid's algorithm, the Extended Euclidean Algorithm, to compute the private key.

You don't need to know how the Extended Euclidean Algorithm (**eea**, Listing 1, lines 13–19) or the other algorithms work to understand how RSA uses them, but there are many well explained resources on the web if you want to know more.

The **key** method passes, on line 30, the public exponent, **e**, and the totient, **t**, into **eea** and receives their Bézout Coefficients - the first one is the private key. That's taken modulo **t** to ensure it's a positive number (if it's negative, it'll have a positive congruence modulo **t**).

As an example, take **e=17** and **n=26**. Those numbers are small enough to iteratively compute **t** or factorise **n** into **p=2** and **q=13** and then obtain **t=(2-1)(13-1)=12**. The algorithm will return the private key **d=5**. We have our keys – we can now encrypt!

Encryption is easy. Take a message **m=20** and it's just a matter of  $n^e \bmod n = 20^{17} \bmod 26 = 24$ . And decryption is just as easy:  $24^5 \bmod 26 = 20$ . Both use the same formula so a single **crypt** method (lines 33–35) provides the implementation.

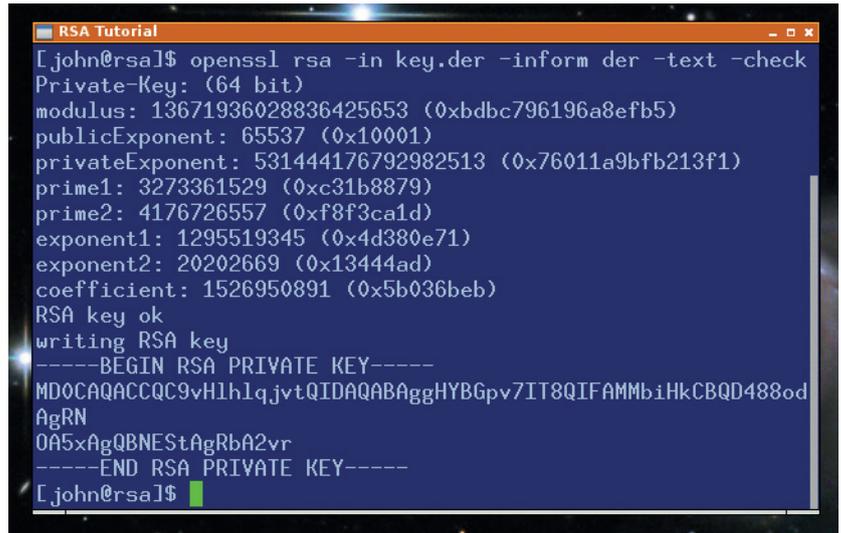
Listing 2 presents a small test that exercises the algorithms in Listing 1. We can use it to demonstrate that the message, **m**, is encrypted to ciphertext, **c** with public key **e**, which is decrypted to **mm** by private key **d**:

```
$ ./testrsa.rb 20 17 26
m:20 e:17 args:26
n:26 d:5 c:24 mm:20
```

You can also test with larger numbers and use prime factors:

```
$ ./testrsa.rb 29384 65537 199 617
m:29384 e:65537 args:199,617
n:122783 d:78209 c:17392 mm:29384
```

### Exponential Efficiencies



OpenSSL provides tools that can create and verify standard key files from the parameters generated by our test program.

The tests demonstrate that the algorithm is sound, but think about large numbers for a moment. Raising numbers to large exponents is likely to exceed the capabilities of your math library, even with a big number library (we found this happened when **n** was around 21 bits). We need another way and, because we want the result modulo **n**, modular arithmetic can provide the answer: a large exponent can be reduced to a simpler problem and one way to do this is known as the binary or squaring method. Here is one way to implement it:

**PRO TIP**  
The current factoring record was set in 2009 when the factors of a 768-bit number were found. It took two years to factor the 232-digit number.

```
def powmod(base, exponent, modulus)
  return modulus==1 ? 0 : begin
    result = 1
    base = base % modulus

    while exponent > 0
      result = result*base%modulus if exponent%2 == 1
      exponent = exponent >> 1
      base = base*base%modulus
    end
    result
  end
end
```

Add this new **powermod** method into **rsa.rb** and change **crypt** (Listing 1, line 34) to use it:

```
powmod(m,e,n)

This allows key sizes to be increased beyond 21 bits. You can try to generate a key with 2048 bits if you feel brave enough (you will need to use two 1024-bit primes). Something like this :
$ p=$(openssl prime -generate -bits 1024)
$ q=$(openssl prime -generate -bits 1024)
$ ./testrsa.rb 29384 65537 $p $q
```

The binary method only needs to perform

## Listing 2: Testing "testrsa.rb"

```

0 #!/usr/bin/ruby
1 require_relative 'rsa'
2 def rsa(m, e, *args)
3   puts "m:#{m} e:#{e} args:#{args.join(',')}"
4   d, n, *crt = key(e,*args)
5   mm = crypt(c = crypt(m,e,n),d,n)
6   puts "n:#{n} d:#{d} c:#{c} mm:#{mm}"
7   [m,c,n,e,d,*crt]
8 end
9 if ARGV.length > 0

```

```

10 m, c, *params = rsa(*ARGV.map{|i| i.to_i})
11 puts "asn1=SEQUENCE:rsa_key\n\n[rsa_key]"
12 %w(modulus pubExp privExp p q e1 e2 coeff).each do |i|
13   break if params.empty?
14   puts "#(i)=INTEGER:#{params.shift}"
15 end
16 end

```

If given no arguments this test program it will continuously generate input data to soak-test the algorithm.

multiplicative operations for each **1** bit in the exponent, so we can make further efficiencies by having as few as possible. The usual choice of **65537** for the public exponent **e** has only two **1** bits but the private exponent **d** will be a large number with a lot of **1** bits, and this can make decrypting a time-wise expensive operation.

We can employ techniques to minimise this: we can use Carmichael Numbers to find a smaller value for **d** and then use the Chinese Remainder Theorem to split the decrypting exponent operation into two smaller (and more efficient) modular exponent calculations.

## We'll use our RSA implementation in the real world to generate a key and use it with OpenSSL

The Carmichael Function (sometimes called the Reduced Totient), usually denoted as  $\lambda$  (lambda), is a drop-in replacement for Euler's Totient that can be used when the prime factors **p** and **q** are known. It results in private keys that are smaller. Smaller numbers have fewer bits and we've seen that fewer bits are less work when computing exponents. To use it, add its definition to the code:

```

def lam(p,q)
  (p-1).lcm(q-1)
end

```

and then change the **key** method (Listing 1, line 28) to use it. Change it so that **lam** is used instead of **phi** to obtain the totent value **t**:

```
t = lam(p,q)
```

The rest of the code will use the value as if it is **phi**; nothing else needs to be changed. Private keys will now be smaller!

You can test with **e=3**, **p=17** and **q=23**. If we used **phi**, we'd get **d=235** but using **lambda** it's **d=59**. What's interesting is that this reveals a little-known fact – there is more than one private key for any public key: both values of **d** will decrypt a message that was encrypted with the public exponent **e=3** and modulus **n=17\*23=391**.

The second optimisation uses the Chinese Remainder Theorem to split the computation in two. We compute three additional values when we create

the private key and let them become part of it. The values are two new exponents and a modular inverse and we extend the **key** method in Listing 1 (replace line 31 with the below) to compute them:

```
if defined?(p) && defined?(q)
```

```
  dp = d % (p-1)
```

```
  dq = d % (q-1)
```

```
  qinv = eea(q,p).first % p
```

```
  [d,n,p,q,dp,dq,qinv]
```

```
else
```

```
  [d, n]
```

```
end
```

Then, provide a new **decrypt** method that can be used with those values to decrypt a message:

```
def decrypt(c,p,q,dp,dq,qinv)
```

```
  m1 = powmod(c,dp,p)
```

```
  m2 = powmod(c,dq,q)
```

```
  h = qinv*(m1-m2) % p
```

```
  m = m2 + h*q
```

```
end
```

If we have the CRT values in our private key then we can decrypt a message more efficiently by using:

```
decrypt(c,p,q,dp,dq,qinv)
```

instead of the equally valid but less efficient

```
crypt(c,d,n)
```

## Hello, World!

Those optimisations complete the RSA implementation. We'll now show it is compatible with real-world implementations by generating a key and using it with OpenSSL. We'll also use a real text message instead of a number to demonstrate that real words can be encrypted.

First of all we will use the test program to create some key parameters (the message isn't important because we're only interested in the key; choose any public exponent and prime factors that you want - we've used 32 bits to keep them readable but 1024 bit primes will work too):

```
$ ./testrsa.rb 12345 65537 3273361529 4176726557
```

Put them in a temporary text file formatted like this:

```
asn1=SEQUENCE:rsa_key
```

```
version=INTEGER:0
```

```
modulus=INTEGER:13671936028836425653
```

```
pubExp=INTEGER:65537
```

```
privExp=INTEGER:531444176792982513
```

```
p=INTEGER:3273361529
```

```
q=INTEGER:4176726557
e1=INTEGER:1295519345
e2=INTEGER:20202669
coeff=INTEGER:1526950891
```

You may have noticed that the test program also outputs key information in this format; now you know why. Save the temporary file (with a recognisable name, say **key.cnf**), and run it through OpenSSL to create a standard **.pem** formatted key files (we have to first create a DER-formatted key and then change it to PEM):

```
$ openssl asn1parse -genconf key.cnf -out key.der
$ openssl rsa -in key.der -inform der -check -noout
RSA key ok
$ openssl rsa -in key.der -inform der -out private.pem
writing RSA key
$ openssl rsa -in key.der -inform der -out public.pem
-pubout
writing RSA key
```

That gives us valid keys in text files **private.pem** and **public.pem** that we'll now use to perform some encryption. We'll need another very small test program for that (**crypt.rb**):

```
#!/usr/bin/ruby
require_relative 'rsa'
m, mm = 0, ""
STDIN.read.each_byte { |c| m=(m<<8)+c }
m = crypt(m,*ARGV.map{|i| i.to_i})
while m>0 do
  mm.prepend (m & 255).chr
  m = m >> 8
end
print mm
```

This encrypts or decrypts a message read from standard input. First it converts the message into an integer as required by the algorithm. It then calls the **crypt** method, passing in the command-line arguments, where the exponent and modulus should be given. The integer returned by **crypt** is then returned to a string and output (**print** is used so that a newline isn't appended because this will corrupt

### Ancient algorithms

RSA depends on some algorithms that have stood the test of time. Euler's theorems, which are the foundation for RSA, were penned in the 17th century but aren't the oldest. The Chinese Remainder Theorem, which can speed up decryption, was evident as far back as the 3rd century.

But it was around 300BC when the Greek mathematician Euclid documented his Euclidean Algorithm for the greatest common divisor (or gcd) of two numbers – the largest number that evenly divides both of them. He discovered, given two numbers **a** and **b**, that **gcd(a,b)** is the same as **gcd(b, a mod b)** and that this fact can be applied recursively until **a mod b ≡ 0**; we expressed this succinctly in code (Listing 1, line 2).

ciphertext). We can encrypt with OpenSSL and decrypt with our test:

```
$ echo -n 'HiWorld!' > message.txt
$ openssl rsautl -pubin -inkey public.pem -in message.txt -encrypt -raw -out ciphertext.txt
$ ./crypt.rb < ciphertext.txt 531444176792982513 13671936028836425653
HiWorld!
```

First we place our message into a text file. We're using "raw" mode with OpenSSL, which means that it won't apply the padding that it would normally use to make the message the same size as the modulus and also more secure. Padding is external to the RSA algorithm and space prohibits covering it now, so using **-raw** disables it. Note, however that OpenSSL then insists that the input message is the same size as the modulus – hence our test message is exactly eight characters to match our 64-bit key.

With the message taken care of, we call OpenSSL to get an encrypted **ciphertext.txt** file, which we pass as input to our test program. We pass our private exponent and modulus as arguments and receive our original text as output. Finally, reversing the order, we can encrypt with our test, passing our public exponent this time, and decrypt with OpenSSL:

```
$ ./crypt.rb < message.txt 65537 13671936028836425653 > ciphertext.txt
```

OpenSSL insists that the input message is the same size as the modulus – hence our test message is 8 characters

```
$ openssl rsautl -inkey private.pem -in ciphertext.txt -decrypt -raw
HiWorld!
```

Everything works as expected, proving our RSA implementation works as it should. We'll leave it as an exercise for you to write a decrypter that uses the Chinese Remainder Theorem to decrypt the message (hint: use the **decrypt** method instead of **crypt**).

### Final words

We've shown how to write your own RSA cryptosystem to demonstrate how it works by revealing the centuries-old algorithms that it relies upon. We've created keys and then used them with OpenSSL to demonstrate that they are compatible. However, we haven't covered everything, and there are ways of using RSA that are less secure than others. Real-world implementations are aware of such potential weaknesses and apply techniques such as the padding schemes that we mentioned to mitigate them. Don't rely on a homebrew cryptosystem to protect your secrets! 

**John Lane** provides technical solutions to business problems. He has yet to find something that Linux can't solve.

# USE ELIXIR TO DEVELOP A WEB APPLICATION

Elixir can make your software reliable, fault-tolerant and highly available.

MIHALIS  
TSOUKALOS

## Why do this?

- Develop better web applications
- Use Elixir to develop fault-tolerant web applications
- Learn how to develop a website in Elixir

**E**lixir is a functional programming language created by José Valim built on top of the Erlang Virtual Machine. Elixir tries to improve the complicated parts of Erlang while keeping the good parts intact. Elixir helps you write cleaner programs using less code than Erlang. This means two things: first, that you can better understand what an Elixir program tries to implement; and second, that you can maintain an Elixir program more easily than an equivalent Erlang program. Both Elixir and Erlang are well suited for writing reliable server software.

Elixir enables you to very easily create powerful software, as we will demonstrate. After finishing this tutorial you will be able to program your own highly available, fault-tolerant web application in Elixir.

You'll need to install both Erlang and Elixir. If you're on a Debian-based system, you'll need to add the repositories for Erlang Solutions, like so:

```
wget http://packages.erlang-solutions.com/erlang-solutions_1.0_all.deb
```

```
code$ mix new web --sup
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/web.ex
* creating test
* creating test/test_helper.exs
* creating test/web_test.exs

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd web
    mix test

Run "mix help" for more commands.

code$ cd web/
web$ ls -lR
.:
total 20
drwxr-xr-x 2 mtsouk mtsouk 4096 Apr 25 22:01 config
drwxr-xr-x 2 mtsouk mtsouk 4096 Apr 25 22:01 lib
-rw-r--r-- 1 mtsouk mtsouk 680 Apr 25 22:01 mix.exs
-rw-r--r-- 1 mtsouk mtsouk 384 Apr 25 22:01 README.md
drwxr-xr-x 2 mtsouk mtsouk 4096 Apr 25 22:01 test

./config:
total 4
-rw-r--r-- 1 mtsouk mtsouk 1109 Apr 25 22:01 config.exs

./lib:
total 4
-rw-r--r-- 1 mtsouk mtsouk 579 Apr 25 22:01 web.ex
```

The `mix` utility is the Elixir build tool, that can help you create, compile, test and manage your projects' dependencies.

```
sudo dpkg -i erlang-solutions_1.0_all.deb
```

Now you can grab the packages you need:

```
sudo apt-get update
```

```
sudo apt install esl-erlang
```

```
sudo apt install elixir
```

For other distros, the process will be a little different; the details are at <http://elixir-land.org/install.html>.

You can find the version of Elixir you are using by running `elixir --version`; at the time of writing this tutorial, the latest stable version of Elixir is 1.2.4.

The following code is the Elixir version of the "Hello World!" program:

```
$ cat helloWorld.ex
```

```
defmodule LinuxVoice do
```

```
  def hello do
```

```
    IO.puts "Hello World!"
```

```
  end
```

```
end
```

```
LinuxVoice.hello
```

The reason for needing a module and a function for such a simple program is that all Elixir code must be organised in modules and functions. The last command is for automatically calling the desired function. The `.ex` extension is used for files that contain Elixir code, whereas the `.exs` extension is used for Elixir scripts.

You can execute `helloWorld.ex` using the interactive Elixir shell (`iex`) or compile it and execute it using the Elixir compiler (`elixirc`). Alternatively, you can use the Elixir script runner, named `elixir`, which is similar to `iex` but automatically exits when the Elixir script finishes. So, you can run `helloWorld.ex` as follows:

```
$ elixir helloWorld.ex
```

```
Hello World!
```

```
$ elixirc helloWorld.ex
```

```
Hello World!
```

```
$ ls -l Elixir.LinuxVoice.beam
```

```
-rw-r--r-- 1 mtsouk mtsouk 1348 Apr 25 16:44 Elixir.
```

```
LinuxVoice.beam
```

```
$ file Elixir.LinuxVoice.beam
```

```
Elixir.LinuxVoice.beam: Erlang BEAM file
```

```
$ iex
```

```
iex(1)> c("helloWorld.ex")
```

```
Hello World!
```

```

3. mtsouk@mail: ~/docs/article/working/Elixir.LV/code/web (ssh)
web$ iex -S mix
Erlang/OTP 18 [erts-7.3] [source-d2a6d81] [64-bit] [async-threads:10] [hipe] [kernel-poll:false]

Starting Web Router!

=INFO REPORT==== 30-Apr-2016::13:02:08 ===
  application: logger
  exited: stopped
  type: temporary
** (Mix) Could not start application web: Web.start(:normal, []) returned an error: shutdown: failed to start child: {:ranch_listener_sup, Web.Router.HTTP}
** (EXIT) shutdown: failed to start child: :ranch_acceptors_sup
** (EXIT) {:listen_error, Web.Router.HTTP, :eaddrinuse}
web$

```

#### [LinuxVoice]

```
iex(2)> LinuxVoice.hello
```

```
Hello World!
```

```
:ok
```

As you can see, `elixirc` automatically generates a BEAM file as does the Elixir shell after processing the `c("helloWorld.ex")` command. This happens because when you compile an Elixir program, the compiler converts the code into a BEAM (Bodgan's Erlang Abstract Machine) file, used by Erlang. The Elixir script runner doesn't generate a BEAM file which can be very convenient when experimenting with Elixir.

### Process Power

Elixir processes are very lightweight and isolated from each other. An Elixir program is implemented as a large number of small processes that do simple tasks and communicate with each other using code that has no side effects. The biggest difference between Elixir and Erlang is that in Elixir the value of a variable can change after its initial assignment whereas in Erlang this is not allowed.

Elixir offers a tool for creating new projects called **mix**. To get a list of all available **mix** options, execute

### What is OTP?

OTP stands for Open Telecom Platform, and is Erlang's collection of open source libraries and tools designed for developing big projects. OTP is about taking all the generic components, putting them into libraries, making sure they work fine and reliably and then reusing that code as often as possible. The programmer needs only to deal with things that change from application to application.

OTP enables you to supervise existing Elixir and Erlang code. In order to supervise an existing module, you will write additional Elixir code, but you will not need to make any changes to the module you want to supervise! If you are going to write real-world Elixir or Erlang software, you will eventually have to learn the OTP Framework.

**mix --help**. If you want to find more information about the **new** command, for example, you can execute **mix help new** – this works for every **mix** command.

Processes in Elixir are identified by a unique process ID (PID). A PID has the following form and its own data type, which means that you cannot manage a process ID as if it were a string:

```
#PID<0.185.0>
```

The project for the web application will be created with the help of the **mix** tool. Run **mix new web --sup** in the new directory to build the necessary files (see figure 1). The **--sup** option that was used is optional; its purpose is generating an OTP application skeleton

Here's what will happen if you try to start another instance of the same web application.

## An Elixir program is implemented as a large number of small processes that perform simple tasks

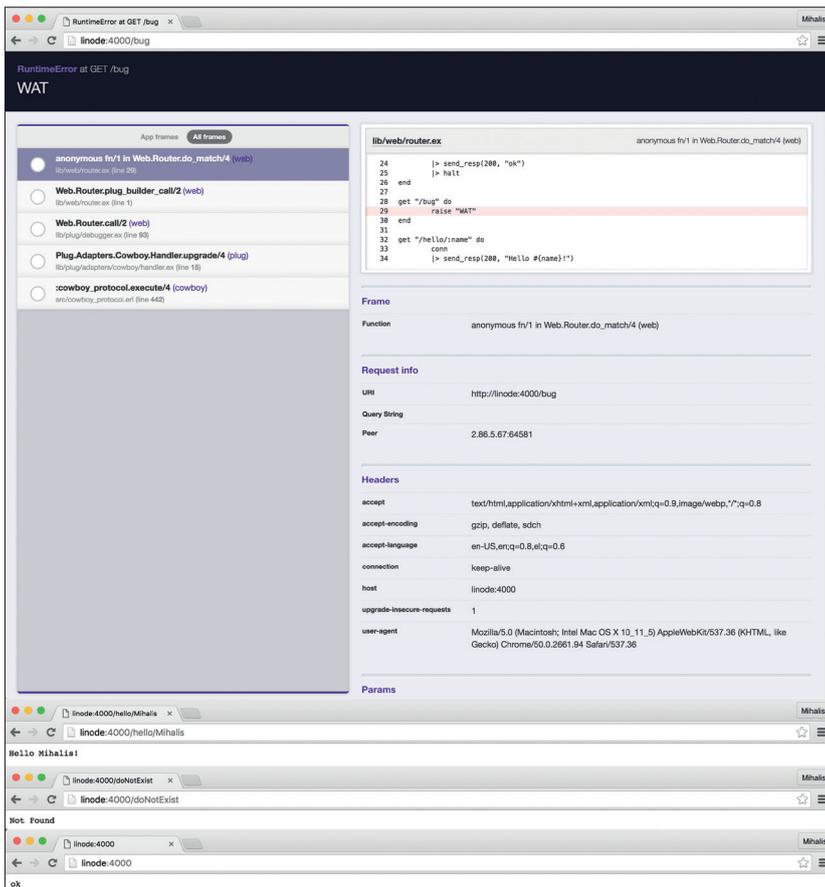
including a supervision tree (see boxout). You'll learn more about supervisors later on in this tutorial.

Other useful **mix** commands include **mix compile**, which compiles the source files of the current project, **mix clean**, which erases the generated application files and **mix run**, which is used for executing your project – running **mix run** on an empty project generates no output. The **mix test** command runs the tests of a project; when you create a new project, **mix** automatically generates some dummy tests – it is the job of the developer to create real tests. The purpose of the **mix.exs** file is to configure your project. The code for the project can be found at **./lib/web.ex** and **./lib/web/router.ex** – these are the files that you are going to edit.

You can start the web application by executing the following command:

```
$ iex -S mix
```

```
iex(1)> :application.which_applications
```



The various URLs supported by the web application. You can add as many URLs as you want!

```
{:web, 'web', '0.0.1'}, {:logger, 'logger', '1.2.3'}, {:mix, 'mix', '1.2.3'},
{:iex, 'iex', '1.2.3'}, {:elixir, 'elixir', '1.2.3'},
{:compiler, 'ERTS CXC 138 10', '6.0.3'}, {:stdlib, 'ERTS CXC 138 10', '2.8'},
{:kernel, 'ERTS CXC 138 10', '4.2'}
```

The previous output shows that the web application is running along with many other applications including **mix**, **logger**, **iex**, **elixir**, **compiler**, **stdlib** and **kernel**.

## For such a simple app, the main task of the developer is defining the supported URLs using pattern matching

In this section you're going to make the web application support multiple web pages. There exist many web frameworks that can make your life easier, including the famous *Phoenix* web framework. However, the simplicity of our application does not justify the use of *Phoenix*.

For such a simple application, the main task of the developer is defining the supported URLs using pattern matching. With pattern matching in functional programming languages, you need to provide a route or a case that matches the URLs that cannot be matched by the other routes.

The **mix.exs** file has three parts: the **project** function is used for describing the project, whereas

the **application** function is used for describing the application itself. Finally, the **deps** function is used for listing the dependencies of the project. The **application** function states the name of the Elixir module that will be used for developing the application – in this case the name of the module is **Web**, based on the argument of the **mix new** command. However, the name of the application is **:web**, as defined in the **project** function – you will use this name to start and stop the application. When you start the application using **mix**, the **Web.start/2** function is automatically called. The **/2** means that the **Web.start()** function requires two arguments.

The final version of **mix.exs** without any comments is the following:

```
defmodule Web.Mixfile do
  use Mix.Project

  def project do
    [app: :web,
     version: "0.1.1",
     elixir: "~> 1.2",
     build_embedded: Mix.env == :prod,
     start_permanent: Mix.env == :prod,
     deps: deps]
  end

  def application do
    [applications:
     [:logger, :cowboy, :plug],
     mod: {Web, []}]
  end

  defp deps do
    [{:cowboy, "~> 1.0.3"},
     {:ranch, "1.2.1"},
     {:plug, "~> 1.1.2"}]
  end
end
```

As you can see from the definition of the **deps** function, the project uses three external components named **Plug**, **Ranch** and **Cowboy**. After defining that you want to use **Plug**, **Ranch** and **Cowboy**, **mix** will get all the modules for you or give you instructions on how to install them and any other additional module dependencies when you try to run or compile the project. As you can understand from the definition of the **application** function, you also need to tell **mix** which modules your web application will actually use. As **Ranch** is not directly used by **:web**, you don't need to include it in the list.

The Elixir code of the **./lib/web.ex** file is:

```
defmodule Web do
  use Application

  def start(_type, _args) do
    import Supervisor.Spec, warn: false

    IO.puts "Starting Web Router!"

    children = [
```

```

    Plug.Adapters.Cowboy.child_spec(:http, Web.
Router, [])
  ]
  opts = [strategy: :one_for_one, name: Web.Supervisor]
  Supervisor.start_link(children, opts)
end
end

```

You will need to create a new module inside the **lib** directory that will serve all HTTP connections with the help of the **Plug** module – the name of the new module will be **Web.Router** and its code can be found inside **.lib/web/router.ex**, where you can define as many routes as you want.

```

defmodule Web.Router do
  use Plug.Router
  use Plug.Debugger

  plug Plug.Logger
  plug :match
  plug :dispatch

  def start_server do
  end

  def init(options) do
    options
  end

  def start_link do
    start_server
  end
end

```

## The Erlang philosophy

The design of Erlang follows six rules. The first rule is Isolation, which means that Erlang processes are isolated; therefore each Erlang process has its own stack and heap and is separately garbage collected. Also, processes cannot see the memory of other processes and consequently cannot harm other processes. The second rule is Concurrency, which means that Erlang processes are concurrent by design. So, in theory, all processes can run in parallel. This is an excellent property now that computers have multi-core processors, because processes can be spread over the available cores.

The third rule is Failure Detection. As failure cannot be avoided, Erlang processes can detect failures. You can also create a link between two processes; therefore, when a process dies for some reason, some other process can be informed about the failure of the first process. So, when something fails, you can let someone else fix the problem. The fourth rule is Live Code Upgrade. Put simply, Erlang can be modified as it runs! Applications can be upgraded while running without downtime! The fifth rule is Fault Identification. This means that when a process fails, the error signal contains additional data provided by the Erlang runtime system that tells exactly why the process has failed. The sixth rule is Stable Storage, which is not done in Erlang but in third-party libraries. You can use *Mnesia*, *Riak*, or another supported databases for storing data. Every process can access the data of a database because the data is shared among Erlang processes.

```

end

get "/" do
  conn
  |> send_resp(200, "ok")
  |> halt
end

get "/bug" do
  raise "WAT"
end

get "/hello/:name" do
  conn
  |> send_resp(200, "Hello #{name}!")
  |> halt
end

match _ do
  conn
  |> send_resp(404, "Not Found")
  |> halt
end
end

```

The **match \_ do** part is the case that matches what is left. Please note that the match all case must be the last one in your code because only the first match is executed.

You can start the web application with the help of the Elixir interactive shell as follows:

```
$ iex -S mix
```

You can also stop the web server from running by exiting the Elixir shell. First, you should press Ctrl+C, which will display the message:

```
iex(2)>
```

```

BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
(v)ersion (k)ill (D)b-tables (d)istribution

```

Select **a** to end the server.

Figure 2 shows the output you will get when you try to execute another instance of the same web application. This happens because a TCP port, in this case port 4000, can only be used by a single application.

Figure 3 shows the web application in action. The **/bug** address uses **Plug.Debugger** to show a very informative web page that can be used for debugging purposes.

## Under supervision

A supervisor has a very specific responsibility: supervising other processes. If something bad happens to the supervised process, the supervisor will notice and start a new process.

The good news is that you do not need to write any extra code to create and use a supervisor for your web application, as this has already be handled by **mix** when you executed **mix** with the **--sup** command line parameter. All processes included in the "children" list will be supervised. The **:one\_for\_one** option inside the **start** function of the **web.ex** file tells the Supervisor to

```

2. mtsouk@mail: ~/docs/article/working/Elixir.LV/code/web (ssh)
3. mtsouk@mail: ~ (ssh)
mtsouk@mail:~$ iex --sname n1
Erlang/OTP 18 [erts-7.3] [source-d2a6d81] [64-bit] [async-threads:10] [hipe] [kernel-poll:f
alse]

Interactive Elixir (1.2.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(n1@mail)1> Node.list
[]
iex(n1@mail)2> WantToCommunicate.sendMessage
** (UndefinedFunctionError) undefined function WantToCommunicate.sendMessage/0 (module Want
ToCommunicate is not available)
WantToCommunicate.sendMessage()
iex(n1@mail)2> Node.spawn_link :n2@mail, fn -> WantToCommunicate.sendMessage end
Message sent!
#PID<9343.79.0>
iex(n1@mail)3> █

a
web$
web$ iex --sname n2
Erlang/OTP 18 [erts-7.3] [source-d2a6d81] [64-bit] [async-threads:10] [hipe] [kernel-poll:f
alse]

Interactive Elixir (1.2.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(n2@mail)1> defmodule WantToCommunicate do
..(n2@mail)1>   def sendMessage do
..(n2@mail)1>     IO.puts "Message sent!"
..(n2@mail)1>   end
..(n2@mail)1> end
{:module, WantToCommunicate,
 <<70, 79, 82, 49, 0, 0, 5, 72, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 134, 131, 104, 2,
 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 115, 95, 118, 49, 108, 0, 0, 0
  , 4, 104, 2, ...>>,
 {:sendMessage, 0}}
iex(n2@mail)2> WantToCommunicate.sendMessage
Message sent!
:ok
iex(n2@mail)3> █

```

Look! Two Elixir shells communicating with each other!

start one process if one process crashes – this is the most important part of the Elixir code related to the supervisor.

In order to see what this really means, you should first find the process ID of the current process and kill it. The steps are the following:

```

iex(5)> children = Supervisor.which_children(Web.
Supervisor)
{{{ranch_listener_sup, Web.Router.HTTP}, #PID<0.279.0>,
:supervisor,
 [ranch_listener_sup]}
iex(6)> [{_, pid, _, _}] = children
{{{ranch_listener_sup, Web.Router.HTTP}, #PID<0.279.0>,
:supervisor,
 [ranch_listener_sup]}
iex(7)> pid
#PID<0.279.0>
iex(8)> Process.exit(pid, :kill)
true
iex(9)> children = Supervisor.which_children(Web.
Supervisor)
{{{ranch_listener_sup, Web.Router.HTTP}, #PID<0.393.0>,
:supervisor,
 [ranch_listener_sup]}
iex(10)> [{_, pid, _, _}] = children
{{{ranch_listener_sup, Web.Router.HTTP}, #PID<0.393.0>,
:supervisor,
 [ranch_listener_sup]}
iex(11)> pid
#PID<0.393.0>

```

The Supervisor automatically started a new process with **pid #PID<0.393.0>** after you killed the old process with **pid #PID<0.279.0>**.

As all Elixir processes communicate with messages, it does not matter whether all processes are running on the same machine, because they will be able to communicate with each other if needed. This section will show you how two different Elixir shells can communicate with each other. Although you will need at least two Linux machines to run a distributed application, you can simulate the process on one Linux machine. The first step includes starting two Elixir shells with different names on two different terminals on the same Linux machine. In the first shell you should execute the following command:

```
$ iex --sname n1
```

In the second shell, you should execute the following command:

```
$ iex --sname n2
```

As you can see in the image, left, the two different Elixir shells have different prompts, depending on the value of the **--sname** command line option, which helps you differentiate between them. Then, you define a new module, named **WantToCommunicate**, in shell **n2** that just contains the **sendMessage/0** function. The key point here is that shell **n1** knows nothing about the **WantToCommunicate** module and its **sendMessage/0** function, which is verified by the following output:

```

iex(n1@mail)1> WantToCommunicate.sendMessage
** (UndefinedFunctionError) undefined function
WantToCommunicate.sendMessage/0 (module
WantToCommunicate is not available)
WantToCommunicate.sendMessage()

```

However, the **n1** shell can use the **WantToCommunicate.sendMessage/0** function and get its output as follows:

```

iex(n1@mail)1> Node.spawn_link :n2@mail, fn ->
WantToCommunicate.sendMessage end
Message sent!
#PID<9332.79.0>

```

So, we called a function from a different shell on the same computer, which could have been a remote computer, without writing any extra code!

We hope that this tutorial helped you realise the advantages of Elixir and OTP – if you're developing a web application, you should definitely consider writing it in Elixir! 📖

## Bibliography

You can learn more about Elixir by reading *Programming Elixir 1.2: Functional, Concurrent, Pragmatic, Fun* from Pragmatic Bookshelf. Two other excellent books about Elixir are *Elixir in Action* from Manning Publications and *Seven More Languages in Seven Weeks* from Pragmatic Bookshelf, which contains a chapter on Elixir.

If you're interested in Erlang you'll find both *Programming Erlang, 2nd edition* from Pragmatic Bookshelf and *Learn You Some Erlang for Great Good!* from No Starch Press very useful. Finally, there's *Erlang and OTP in Action* from Manning Publications, which talks about OTP.

Mihalīs Tsoukalos is a Unix administrator, programmer and mathematician who also enjoys writing technical articles.

**Emergency**

→ **Ebola**

↑ **Gunshots**

→ **Landmine**

↗ **Cholera**

↖ **Shrapnel**

← **Maternity**



**The world's A&E Department**

Find out more at [msf.org.uk](http://msf.org.uk)



Valentine Sinitsyn develops high-loaded services and teaches students completely unrelated subjects. He also has a KDE developer account that he's never really used.

# CORE TECHNOLOGY

Prise the back off Linux and find out what really makes it tick.

## POSIX Threads

Most general-purpose OSes today run processes. They serve as units of isolation: processes have distinct address spaces, and if one of them crashes, the others remain unaffected. Processes are also units of parallelism: an OS can execute many of them simultaneously. On a uniprocessor system, this is just an illusion carefully preserved by the process scheduler. Most PCs today are multicore, though, and processes can really run in parallel on different cores. To sum up, processes are a useful yet costly abstraction.

Sometimes, you don't need isolation. You just want to run some tasks in parallel. This could make a good case for "lightweight processes", or threads. In this Core Tech, we'll look at POSIX Threads (Pthreads), and how Linux implements them.

### Hello, threads!

As a programmer, you may think of a thread as a function that executes in parallel with **main()**. For Linux, threads are just processes that share some resources. This includes memory, file descriptors, and signal handlers. Each thread has its own stack and CPU context, so the kernel can schedule them independently.

A typical desktop application may offload time-consuming jobs, such as software rendering, to

threads. This way, the main UI thread is kept ready for user-generated events (say, mouse clicks). Overall, the application remains responsive, even if it does some complex maths in the background. It's also common to spawn threads for heavy I/O tasks, yet asynchronous approaches (LV016 and LV028) may serve you better. Like processes, threads come at a price. CPU-bound tasks are usually worth it, while for I/O bound tasks, there could be cheaper alternatives.

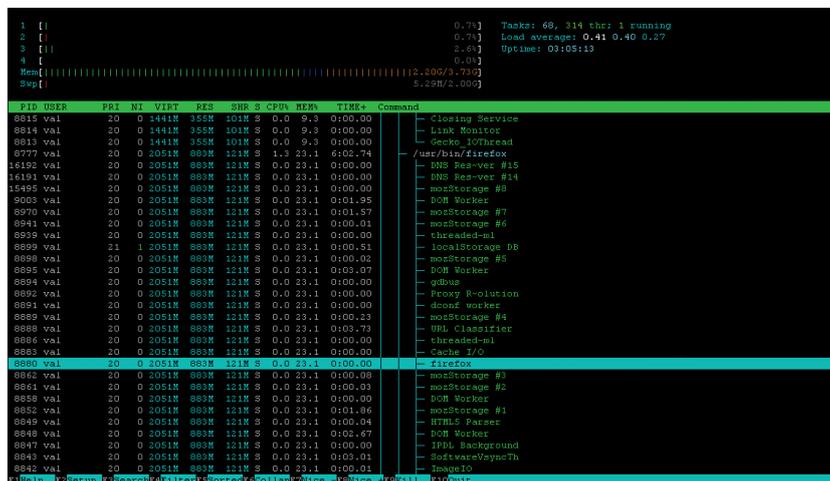
Different operating systems provide various threading APIs. For Unix (Linux included), POSIX Threads are the standard. IEEE POSIX 1003.1c defines them, and the implementation is usually available as **libpthread** which comes with the standard C library (**libc**). To enable threading support in your code, pass the **-pthread** switch to **GCC**.

Creating a thread is simple:

```
#include <pthread.h>
static void * thread_fn(void *unused)
{
    int done = 0;
    while (!done) {
        /* Do the work */
    }
    return NULL;
}
int main()
{
    pthread_t thread_id;
    void *retval;
    int err;
    err = pthread_create(&thread_id, NULL, thread_fn,
NULL);
    if (err) { /* Handle it */;
    err = pthread_join(thread_id, &retval);
    ...
    return 0;
}
```

First, we define a "thread function", **thread\_fn**. That's the code our thread will execute. Thread functions receive a sole **void \*** argument. When the function returns, or calls **pthread\_exit()**, the thread is terminated. So, it is a common pattern to loop in a

htop displays threads in a different colour, and with a custom name, settable via **pthread\_setname\_np(3)**. Note this four-core CPU runs 314 threads in total.



thread function until some condition is fulfilled.

Next, we actually spawn a thread in `main()` with `pthread_create()`. `thread_id` is like a handle to the thread. Don't confuse it with TID (see `gettid(2)`), which is a Linux process' property. `pthread_t` is just some opaque value used as a reference to a specific thread across Pthreads functions. For example, you can cancel a thread with `pthread_cancel(thread_id)`.

The second argument to `pthread_create()` defines the thread attributes. They are useful for fine-tuning, like setting a thread stack size, but that's beyond the scope of this Core Tech. The third argument is a pointer to the thread function, and the last one is the value that `thread_fn` accepts.

Quite often, you just want to spawn a thread to do some background processing, then forget about it. In this case, you should mark it as "detached" with `pthread_detach()`. When a detached thread terminates, the system reclaims its resources automatically.

If you want the result, use `pthread_join()`. This call waits for the thread to terminate, then grabs the `void *` value passed to `return/pthread_exit()`. There isn't much sense in calling `pthread_join()` just after `pthread_create()`, as we do here. But if the thread has already finished, `pthread_join()` collects its exit value and returns immediately.

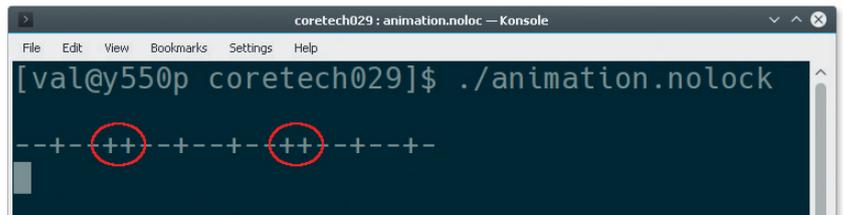
It is important to call either `pthread_detach()` or `pthread_join()`. By default, all threads are joinable, and if you forget to collect the result, you'll get a "zombie thread" wasting the system's resources. This being said, all threads typically lasts only until their parent process terminates. If you want children to outlive their parents, you probably want processes, not threads (LV019).

How many threads to spawn depends on the task's nature. Some do best with one thread per CPU core. `sysconf(_SC_NPROCESSORS_ONLN)` reports how many processors are currently online. Yet this doesn't guarantee that each thread will really run on a dedicated core. `pthread_setaffinity_np()` may help,

### Threads and signals

As you know, you can send a signal to a Unix process. What happens if you send a signal to a multithreaded application? Recall that signal handlers are shared among all threads in a process. So, the kernel delivers the signal to a random thread. It is possible, however, to set per-thread signal masks with `pthread_sigmask(3)`. This way, you can choose which threads are going to respond to which signals. Say, if your program uses `SIGHUP` to reload a config, you may want to handle it in the main thread. Just block `SIGHUP` in all worker threads, and you're done. The `kill` command will deliver this signal to the main thread.

It is also possible to send a signal to specific thread. the `tgkill(2)` system call does this. `libc` usually wraps it, or its older variant, `tkill(2)`, as the `pthread_kill(3)` function. For instance, when you cancel a thread with `pthread_cancel()`, the target thread gets a `SIGCANCEL` signal. It is usually the first POSIX real-time signal (32). This means, `SIGCANCEL` is delivered before any other real-time signal pending.



but it's rarely useful. The ultimate goal is to have just enough threads to process incoming data with no pauses, and no threads idling for too long.

Races manifest themselves in many different ways. When a writer scrolls the buffer halfway behind the reader's back, you get two pluses in a row.

### Taming concurrency

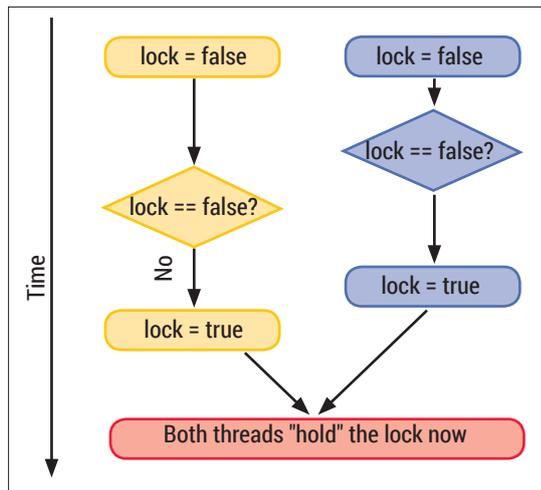
Now, imagine a thread that accepts a fixed-size buffer and continuously fills it with some symbols. That's a poor man's imitation of a complex rendering task:

```
#include <pthread.h>
struct shared_buf {
    char *data;
    long size;
};
static void *render(void *arg)
{
    struct shared_buf *sbuf = (struct shared_buf *)arg;
    char *buf = sbuf->data, c;
    int i;
    for (;;) {
        c = buf[sbuf->size - 1];
        for (i = sbuf->size - 2; i >= 0; i--) {
            buf[i + 1] = buf[i];
        }
        buf[0] = c;
    }
    return NULL;
}
```

The `main()` function initialises data in the buffer and dumps it to `stdout`, acting as a viewer:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LINE_SIZE 24
int main()
{
    struct timespec t = {0, 100000000L};
    struct shared_buf sbuf;
    pthread_t thread_id;
    char buf[LINE_SIZE + 1];
    int i;
    buf[LINE_SIZE] = '\0';
    for (i = 0; i < LINE_SIZE; i++) {
        buf[i] = (i % 3 == 0) ? '+' : '-';
    }
    sbuf.data = buf;
    sbuf.size = LINE_SIZE;
    pthread_create(&thread_id, NULL, render, &sbuf);
    pthread_detach(thread_id);
    for (;;) {
        printf("%s\n", sbuf.data);
        nanosleep(&t, NULL);
        printf("\033[1A");
    }
}
```

Non-atomic comparisons may result in two threads holding a "lock" simultaneously. Use dedicated primitives, Luke.



```

}
return 0;
}

```

The `\033[1A` ANSI sequence moves the cursor one line up. It's often used to animate progress bars in Linux console tools. Perhaps I wanted this code to produce a creeping line effect. And it really does this, but more often than not, the animation appears garbled (see the screenshot). What's wrong?

Shared data is the culprit. There is no synchronisation between threads. So, the reader may get a half-baked frame, and the writer may change the buffer while it's being read. This brings us a whole new set of issues related to atomicity and races. It's complex topic, but let us scratch the surface.

We need to ensure that only one thread accesses the buffer at given moment. Pthreads has many "synchronisation primitives" to achieve this. Perhaps the simplest one is a mutex.

Mutex stands for "mutual exclusion". It's something a thread can lock and unlock. Many threads may try to lock a mutex simultaneously, but only one will succeed. The others will be blocked until the mutex is released. A mutex is like a flag:

```

int locked = 0;
if (!locked) {
    locked = 1;
    /* Proceed safely */
}

```

However, mutex operations are atomic, while this code is not. A potential race is depicted on the figure. Races are often non-trivial to reproduce and debug. So, please use dedicated primitives, not homebrew flags, in your Pthreads code.

Fixing a race in our program is straightforward:

```

struct shared_buf {
    ...
    pthread_mutex_t mutex;
};
static void *render(void *arg)
{
    ...
    for (;;) {

```

```

pthread_mutex_lock(&sbuf->mutex);
/* The buffer is updated */
pthread_mutex_unlock(&sbuf->mutex);
}
...
}
int main()
{
    ...
    sbuf.data = buf;
    sbuf.size = LINE_SIZE;
    pthread_mutex_init(&sbuf.mutex, NULL);
    ...
    for (;;) {
        pthread_mutex_lock(&sbuf.mutex);
        printf("%s\n", sbuf.data);
        pthread_mutex_unlock(&sbuf.mutex);
        ...
    }
    pthread_mutex_destroy(&sbuf.mutex);
    return 0;
}

```

Both threads now grab the mutex before doing anything to the buffer and release it after that. Keeping locks for too long isn't a good idea, as it prevents other threads' progress. Locks can become performance bottlenecks when contended, so one should design synchronisation very carefully. The code path between the lock being acquired and released is often called a "critical section", as only one thread can execute it at the given time. This being said, it's better to think of mutexes as a way to protect specific data, not code. So we embedded the mutex in the buffer structure.

Having no synchronisation for shared data is bad, but having it wrong is worse. Imagine you have two mutexes, A and B. Thread 1 acquires mutex A and waits for B. Thread 2 does the opposite. Neither one

### Green thread

You may also come across the term "green threads". These are threads that exist completely in userspace, in some language runtime (think Java or Erlang). The name may read as if they are lightweight, which is often the case. But in fact, it came from the Green Team at Sun Microsystems, which did the threading in the Java VM.

Green threads aren't real threads as we've discussed; usually, they don't exploit multicore CPUs. They offer cooperative multitasking. This means that the thread itself yields control to the scheduler when it thinks it's time. Quite often, this is transparent to the programmer. This simplifies things a bit: don't make critical sections; simply don't reschedule when you are not ready to. Yet it also adds responsibilities: if a thread keeps running for too long, it makes the others starving. If it blocks on I/O, it blocks the whole program. This sounds similar to how asynchronous I/O works (LV016). In fact, green threads are often bound to event loops internally.

Some languages, like Java or Erlang, provide green threads out of the box. Others may have special libraries, such as Python's Gevent or Greenlet. Green threads aren't a substitute for real ones, but they do have their usages. At least, they run even if the OS lacks threading support.

is going to proceed, and they will remain in this state forever. This situation is called a deadlock. To avoid it, always acquire multiple locks in the same order.

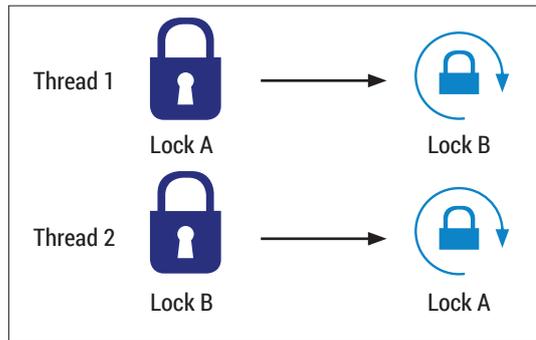
**More primitives**

Quite often, you have threads that only write shared data and threads that only read it. Any number of threads may read data simultaneously, but any writer must have exclusive access to the resource. A mutex is a bad choice here, as it won't allow multiple readers.

Read-write locks (or "rwlocks" for short) are the answer. Pthreads implements them as `pthread_lock_t` and have a bunch of functions (`pthread_rwlock_*`) to operate on them. Readers call `pthread_rwlock_rdlock()/pthread_rwlock_runlock()`, so no writer may change the data while it's being read. Writers use `pthread_rwlock_wrlock()` to obtain exclusive access.

Some may say that **rwlock** is just a specialisation of another primitive, a conditional variable (or "condvar"). An **rwlock** is basically something with two counters and a mutex to protect them. To get a read lock, you acquire the mutex and check if the number of writers is zero. If yes, you increase the readers count, release the mutex and proceed. If not, you release the mutex and block until the writers count reaches zero.

That's exactly how a conditional variable works: it is always associated with a mutex. You grab it and check the condition. If it's false, you call `pthread_cond_wait()` to atomically release the mutex and put a thread to sleep. When the condition changes, another thread signals an event to one or all waiters with `pthread_cond_signal()` or `pthread_cond_broadcast()`, respectively. In waiting threads, `pthread_cond_wait()` grabs the mutex again and returns. The thread should now re-evaluate the condition and either proceed or go back to sleep. The main guarantee that a conditional variables provide is again atomicity. Releasing a mutex and putting a thread to sleep occurs as a single operation. There's no chance for a wake-up signal to get lost in between.



Nothing lasts forever, except two threads waiting for each other to release a lock.

It is also possible (yet rather difficult) to write multithreaded programs lock-free. More formally, the code is lock-free if it keeps at least one thread progressing towards the end result. That's not the case with locks. Imagine the kernel decides to re-schedule a userspace thread holding the lock. No other thread can make progress in this case until the kernel resumes the thread and the lock is released.

Lock-free programming is made possible with low-level atomic operations, among other things. Usually, reading and writing a naturally-aligned simple type (such as a 4-byte aligned 32-bit integer) is atomic. There are also ways to increment and decrement an integer, or exchange two values atomically. Finally, many CPUs provide atomic compare-and-swap (CAS) operation. On x86, it's `lock cmpxchg`. That is, it's possible to check a target value against the expected one and update it if they match without any other code intervening. This could be used to update shared data in a lock-free fashion.

Multithread programming is hard, and lock-free programming is even harder – you don't do it on a whim. The principle that it delivers better performance isn't true in all cases either. So, you should evaluate carefully what your performance bottlenecks are – if these are locks, you may try to re-implement parts of your algorithm lock-free. How to do it properly is beyond the scope of this Core Tech, however.

# Command of the month: **stap**

In multithreaded programs, the order of statements in the sources is different from the order of execution. So, merely looking at the code doesn't reveal the sequence of the program's operations. You want some tools for live introspection, also called dynamic code analysis.

SystemTap is one of such tools, and **stap** is its main executable. SystemTap enables you to write small scripts called probes, and attach them to various events happening in userspace and in the kernel. It was designed to be safe for use on production systems. So, you can study non-trivial situations such as deadlocks, in the wild.

While SystemTap can do many things, the reason we mention it here is that it helps to determine

contended locks. SystemTap comes with many examples found in `/usr/share/doc/systemtap/examples`. One of these is the `process/futexes.stp` script. It traces the `futex` syscall and dumps how many times the thread had to sleep waiting for the lock, and for how long. Lower values are generally better. Here's what I got on my Fedora 23 box for our sample program:

```
$ sudo stap -c ./sample futex.stp
...
sample[3006] lock 0x7ffcc7301fe0 contended 79 times, 26
avg us
```

Make sure you have the kernel debug info installed.

If you want SystemTap covered in one of the future Core Teches, please drop us a line.

# /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!

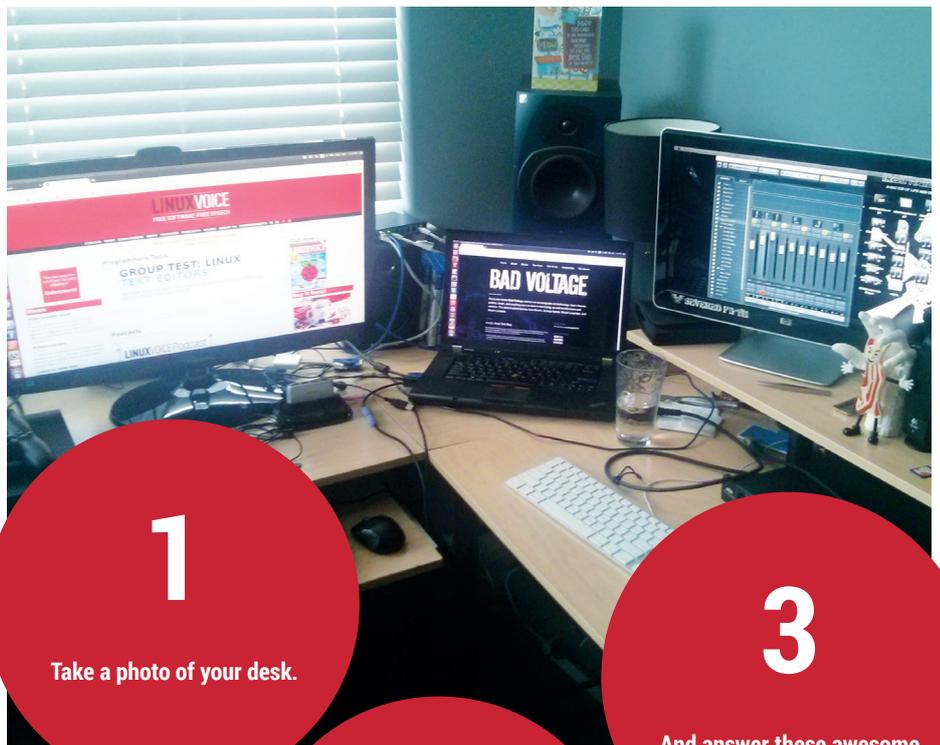
**M**any lifetimes ago, it seems to me, I was the editor of a magazine called *Amiga Format*. It was a magazine that was so stratospherically successful that we could do pretty much anything. The kind of problems about appeasing advertisers, compromises over paper quality, penny-pinching over the best writers and a myriad more that were the constant juggling act of most magazines simply didn't apply any more. We made something brilliant, and that enabled us to remain making something brilliant. It is a situation few people in a commercial environment ever find themselves in, and I was grateful to be a part of it.

It is precisely this environment that drives some of the success of open source. Make something good = people want it. More users generally leads to the ability to do even more good stuff. The trick of turning open source into a revenue-generating business seems to be adding on the money-spinning in a way that doesn't bring everything tumbling down.

The exit of Frank Karlitschek and other developers from *OwnCloud* tends to suggest this is a balancing act that isn't always easy to navigate. Though nobody has given specific reasons for most of the developers upping sticks, the comments that have been made (eg <http://goo.gl/kFYUvr>) suggest that 'investors' were felt to have undue influence over the roadmap. The rapid emergence of Frank's new project, *NextCloud*, and the employment of pretty much anyone from *OwnCloud* who wanted to come underlines where the value of an open source business really lies.

## MY LINUX SETUP

# SEND US YOUR DESKTOPS!



# 1

Take a photo of your desk.

# 3

And answer these awesome questions.



- > What version of Linux are you currently using?
- > What desktop are you using at the moment?
- > What was the first Linux setup you ever used?
- > What Free Software/open source can't you live without?
- > What do other people love but you can't get on with?

# 2

Tell us a little about the things we can see.



Then send your photos and text to:  
**[geekdesktop@linuxvoice.com](mailto:geekdesktop@linuxvoice.com)**

# LINUXVOICE

This is what we've done in the last 24 issues.  
Subscribe to the next 12 from just **£38**.



Every subscription includes access to every PDF, ePub and audio edition we've ever published.

[shop.linuxvoice.com](http://shop.linuxvoice.com)

