Dr Brown's Administeria

Dr Chris Brown
The Doctor provides Linux training, authoring and consultancy. He finds his PhD in particle physics to be of no help in this work at all.

Esoteric system administration goodness from the impenetrable bowels of the server room.

Indistinguishable from magic

The birth of commercial radio broadcasting in the UK has almost passed out of living memory – the first BBC transmissions from 2LO in London were 85 years ago. I remember constructing a crystal set as a young boy (and a one-valve receiver in my teens) and being fascinated by the results.

But before that, in the beginning, to adjust the 'cat's whisker' of your radio, and be rewarded by hearing through your headphones for the first time ever a faint, disembodied voice plucked from nowhere, must have been a jaw-dropping experience. Nowadays, the pace of technological development has advanced to the point where it's well-nigh impossible to impress anyone (particularly the younger folk) with anything at all.

Come friendly data

It seems to me that most of the fundamentally new stuff to have emerged in the last decade relates to the digitisation of the world's information sources – its conversion into vast, structured, searchable collections of ones and zeros. GPS navigation systems can tell us where we are on the planet to within inches (and what would the navigators of the early sailing ships have given for that?) We have search engines that can instantly find the entire text of a poem based on my memory of a single line.

Given just a postcode, Google Maps pinpoints the hotel I'll be staying in next week, and provides street-level photos of the local restaurants. Google Earth can show me the pool I swam in at a friend's house in California. I can carry the complete works of Shakespeare in my pocket on a £5 memory stick. I can download a readout of the three billion base pairs of the human genome. Does any of this strike you as inspired? It sure impresses the heck out of me!

In the Moodle

Moodle If you're looking to deliver online training, consider this free learning management system.

I have recently been introduced to *Moodle*. In fact, over the last month we have enjoyed a fairly intimate relationship. *Moodle* is a free, open-source learning management system. It's entirely web-based, written in PHP, and available for Linux, OS X, and Windows.

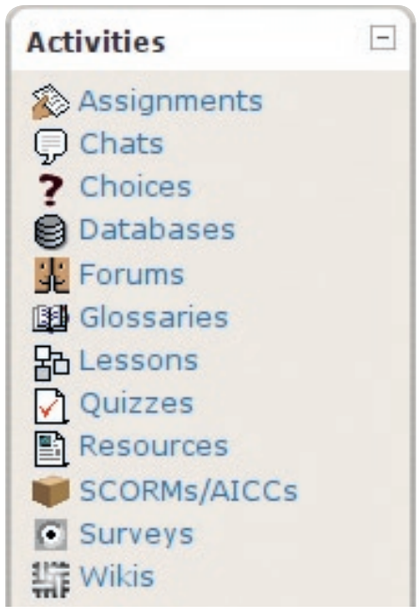
To get an idea of what moodle looks like you can go to the demo site at **http://demo.moodle.org**. Here, you can explore a real moodle course in the role of a student, teacher, or even as an administrator. Be aware, however, that *Moodle*'s appearance can be dramatically changed through the use of themes, for example to include corporate branding.

At its most basic, *Moodle* is a container for training content. Material is created in HTML, and *Moodle* includes a decent web-based WYSIWYG HTML editor. But there's much more to it than that. You can create a wide variety of activities, including:

- **Assignments**.
- **Chat windows**.
- **Forums** Where students and teachers can hold discussion threads.
- **Choices** The teacher asks a question and specifies multiple response choices.
- **Quizzes** Quizzes are created from questions provided by the course creator; they can be multiple choice, true-false or "short answer".
- **Surveys** Used to gather feedback on the students' educational experience.

Moodle isn't something you're going to install and deploy just for fun over a wet weekend. You need to get *Moodle*, PHP, *Apache* and a back-end database such as *MySQL* to play nicely together. Once it's working, you can take off your Linux geek's hat, put on your educator's hat, and figure out how to use *Moodle*'s features to deliver engaging, interactive, online training content.

If you need help, there is a program of *Moodle* conferences around the world. They are called MoodleMoots. Sounds terribly Tolkien to me...



› ***Moodle*** **supports a wide range of activities to make the training interactive and provide feedback to the teacher.**

Debian repository structure

Debian/Ubuntu Confused about where your Deb packages come from? Get to grips with how the repositories are organised.

Last month I talked about Deb files. We saw that they are the unit of packaging of software on Debian and Debian-based Linux distributions such as Ubuntu. We looked at the internal structure of a Deb file and learned how you can build your own. (If you missed that issue, scoot over to your local newsagent and bribe him for a back copy.)

This month, we'll turn our attention to Debian repositories. A repository is a collection of Deb files stored within a specific directory structure and with an accompanying package index file. Usually, the repository is made available via a web server (ie via HTTP), though it's also possible to have repositories that live in the filesystem, for example through an NFS mount from a server, or from a CD.

Packages in pockets

Ubuntu uses a two-level logical grouping of packages, which is directly reflected in the directory structure of the repositories. The upper level categories are called 'pockets', although you'll also hear them referred to as 'distributions'. For Ubuntu Hardy Heron, the pockets are:

» **hardy** This is the base product. It's frozen at the time of release of Ubuntu, and it's what you get on the CD.

» **hardy-updates** This pocket contains updates and bugfixes for the Hardy release. You will not see new versions of applications; the packages here are strictly updates to the versions published at the time of release.

» **hardy-security** Similar to the hardy-updates pocket, but for security-related updates.

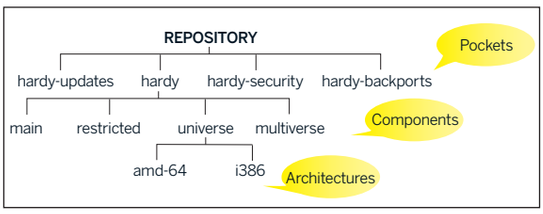
» **hardy-backports** This pocket provides new versions of already-released packages. For example, the hardy-backports pocket contains version 1.5.10 of Arts (the Linux sound system) whereas the Hardy base distribution has version 1.5.9. This pocket is maintained by the community.

If you're using another Ubuntu distro, such as Intrepid Ibex, the pockets will be named intrepid, intrepid-security etc.

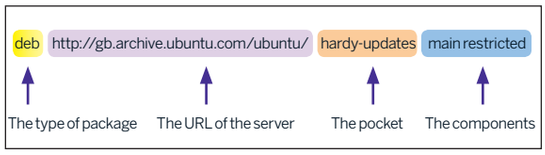
Beneath each of those pockets lies a second layer of organisation. Here, software is organised into four categories, called components, based on the level of support that is offered for the package, and its licensing terms:

» **main** These packages are fully supported by Canonical and may be freely used under open-source licences. Canonical provide security updates for these packages.

» **restricted** These packages are supported by Canonical but aren't available under open-source licences, although they are usually free. The binary drivers released by video card vendors fall into this category.



› The structure of the Ubuntu repositories. Package managers automatically choose the appropriate packages for the architecture they're running on.



› A single line from /etc/apt/sources.list. These entries map pockets and components on to server URLs.

» **universe** These packages are community-maintained open source packages. They are not supported by Canonical, and are not on the Ubuntu CD. This is the largest category.

» **multiverse** These packages typically have restrictive licences and may require registration or payment of a fee before use.

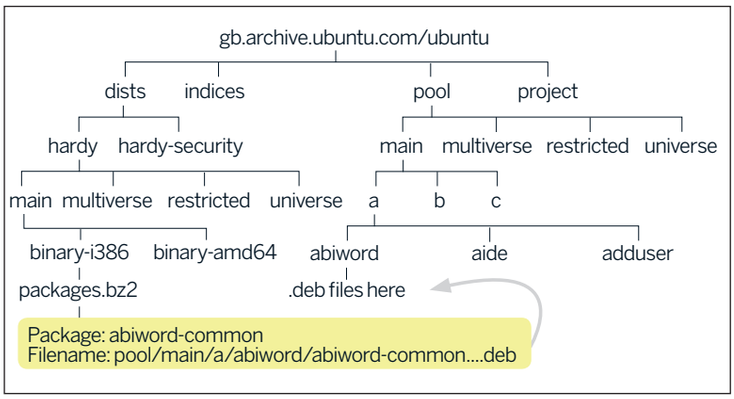
Debian includes a set of tools for interacting with these repositories, known collectively as the Advanced Packaging Tool (*APT*). This tutorial is not about *APT*; to learn more you should at least look at the man pages for *apt-get* and *apt-cache*. *APT* uses the configuration file **/etc/apt/sources.list** to tell it where its repositories are. The structure of the entries in this file reflects the pockets-and-components organisation.

Repository structure

You can look at the structure of a real repository by browsing to **http://gb.archive.ubuntu.com/ubuntu**. If you drill down from there you'll uncover a directory structure like the one I've sketched. The left-hand side of this (under **dists**) directly reflects the pocket, component and architecture structure we've discussed, but the Deb files themselves are not here. Instead you'll just find files called **Packages.bz2**. (These are the files that are fetched and cached locally when you do **sudo apt-get update**).

The actual Deb files are under the **pool** directory, which is broken down by component (but not by pocket or by architecture). To keep the structure manageable there's an extra layer of directories called **a**, **b**, **c** and so on. The **Packages.bz2** files provide the link between the two branches of this hierarchy, relating the package name to the filename within the pool. Pooled repositories avoid duplication of Deb files that are common across multiple architectures or distributions. You'll see a similar structure on the Ubuntu CD, though only for the main and restricted components.

› The filesystem layout of an Ubuntu repository. Only a few sample directories are shown.



Reflected glory

Proxies and mirrors bring your Debian or Ubuntu repositories closer to home and speed up software installation.

The easiest way to start serving Debian-style repositories is to set up a machine as an *APT* proxy. The package **apt-proxy** will do this for you. To quote the man page:

"*Apt-proxy* is a Python program designed to be run as an standalone server via *twistd*, and provides a clean, caching, intelligent proxy for *apt-get*, which speaks HTTP to *apt-get* clients, and HTTP, FTP or *rsync* to the back-end server(s). *Apt-proxy* listens by default on port 9999".

Here's how it works. The client machines on your network are configured to go to your proxy for their repositories, not to the 'real' servers. The first time the proxy is asked for, say, the *Banshee* package, the proxy will go out to the back-end server to get it. The proxy then delivers it to the client and caches it locally. When some other client asks for the same package, it will be delivered from the cache. Obviously, a caching proxy like this works well if it has many clients behind it that are likely to be installing the same stuff, for example, if they are all configured to download security updates.

> **"We're assuming that servers closer to you give better performance."**

By default, *apt-proxy* builds its cache in **/var/cache/apt-proxy**, though this location is configurable in the *apt-proxy* config file.

Making it work

Here's what I did to get *apt-proxy* up and running on my Ubuntu 8.04 server:

1 Installed the *apt-proxy* package:

```
$ sudo apt-get install apt-proxy
```

2 Tweaked the *apt-proxy* config file (**/etc/apt-proxy/apt-proxy-v2.conf**). The main thing here is to set the entries that define where the back-end servers are. I changed only a single line – just below the **[ubuntu]** section marker, I changed the **backends** definition to that of a geographically local server, so that the line looked like this:

```
backends = http://gb.archive.ubuntu.com/ubuntu
```

It would work fine even without this change, but we're assuming that servers that are physically closer to you will give better performance. It is not clear to me that this is always the case!

There are other things in the config file that you might want to adjust, for example the port number that the server will listen on, and the directory where *apt-proxy* will keep its cache, but I left all of these at their default settings.

3 Restarted the *apt-proxy* server:

```
$ sudo /etc/init.d/apt-proxy restart
```

That's it for the server.

Next, I had to make sure that my client machines used my new proxy instead of going directly to the 'real' servers. So, I edited the **/etc/apt/sources.list** file, changing every reference of **gb.archive.ubuntu.com** to **192.168.1.65:9999**. Here, 192.168.1.65 is the IP address of my *apt-proxy* server, and 9999 is the port number it's listening on.

After this, you should run:

```
$ sudo apt-get update
```

on the client to refresh the package information. As a side-effect, the proxy will download and cache this information.

Does it work? Well, outwardly things should behave pretty much as they did before. As a test, I installed the package *bind9* on a client:

```
$ sudo apt-get install bind9
```

The output from *apt-get* includes this line:

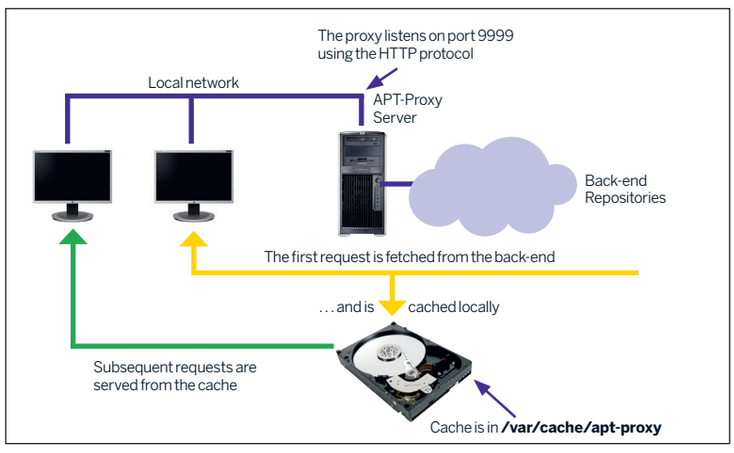
```
Get: 1 http://192.168.1.65 hardy-updates/main bind 9 ...
```

... which shows that it really is using the proxy, and if you examine the *apt-proxy* log file on the server, you should see (among other things!) entries something like:

```
[CacheEntry] start download:pool/main/b/bind9/bind9....deb
[CacheEntry] file_sent:/var/cache/apt-proxy/ubuntu/pool/main/b/bind9/bind9....deb
```

Finally, if you look in the directory **/var/cache/apt-proxy/ubuntu/pool/main** you should see that the *bind9* Deb file is now cached. In fact, *apt-proxy* is beginning to build a directory structure that matches the one on the real servers. The second time you install *bind9* on a client, the file will be served from this cache.

› ***Apt-proxy* fetches the Deb file from the back-end server first time around and subsequently serves it from the cache.**



APT for the impatient

Here are the four *APT* commands I find most useful:

1 Refresh the package metadata from all the repositories listed in sources.list. You should do this periodically, and certainly after editing sources.list:
$ sudo apt-get update.

2 Install the package foo and any other packages needed to make foo work:
$ sudo apt-get install foo.

3 Search the package metadata cache for all packages related to astronomy:
$ apt-cache search astronomy.

4 Show the package metadata for one specific package:
$ apt-cache show celestia.

If you have been using *APT* stand-alone to install packages on your server, you will find that the Deb files you installed have all been cached in **/var/cache/apt/archives**. You can import these into your *apt-proxy* cache using the script *apt-proxy-import*, which is also part of the *apt-proxy* package.

Create a mirror

The next step up the repository ladder, so to speak, is to mirror the existing repositories. The difference between proxying and mirroring is that a proxy only holds local copies of Deb files that it has previously been asked to provide, whereas a mirror actively downloads content to provide a complete local replica of a specified collection of repositories.

I feel that some sort of don't-try-this-at-home-folks warning is appropriate here. Maintaining a mirror is going to fill up several tens of gigabytes of disk space and use a lot of bandwidth while the mirror is being populated – so don't build your own mirror unless you have at least a few client machines that will benefit from it!

To create a Debian mirror, start by installing the package **apt-mirror**. The config file **/etc/apt/mirror.list** defines the repositories you want to mirror and should be adjusted to point to the appropriate repositories for the country you live in. I changed all occurrences of **archive.ubuntu.com** to **gb.archive.ubuntu.com**, so that a typical entry looks like this:

```
deb http://gb.archive.ubuntu.com/ubuntu hardy main restricted universe multiverse
```

I also decided not to mirror the source archive, so I commented out the lines beginning **deb-src**. You might also want to adjust the **nthreads** parameter in this file. This determines how aggressively *apt-mirror* will operate. I set it to 2. Now you can start to populate the new mirror:

```
$ sudo apt-mirror
Downloading 33 index files using 2 threads...
Begin time: Sun Jan 18 09:51:25 2009
[2]... [1]... [0]...
End time: Sun Jan 18 09:52:03 2009
Proceed indexes: [PPP]
25.4 GiB will be downloaded into archive.
Downloading 28204 archive files using 2 threads...
Begin time: Sun Jan 18 09:52:08 2009
[2]...
```

This will begin to download the mirror into **/var/spool/apt-mirror/mirror/gb.archive.ubuntu.com/ubuntu**. Clearly, it will take a little while...

With the mirror populated, we need to make it available to our clients. One way might be to export the mirror from the server using NFS. This approach might work well if your mirror and your clients are all on the same LAN. Alternatively, you can install a web server such as *Apache* and serve the repository through that.

I decided to go the NFS route. Assuming you have an NFS server installed, you can export the mirror by adding a line to **/etc/exports** on the server something like this:

```
/var/spool/apt-mirror          *(ro)
```

then run:

```
$ sudo exportfs -a
```

On the clients, create a suitable mount point (I chose **/repos**) and mount the mirror:

```
$ sudo mkdir /repos
$ sudo mount 192.168.1.65:/var/spool/apt-mirror /repos
```

Now, we need to tell our clients to use the mirror instead of the upstream servers. On the clients, edit the file **/etc/apt/sources.list** and change all the lines that refer to the upstream servers. For example, a line that might previously have read:

```
deb http://gb.archive.ubuntu.com/ubuntu hardy universe
```

should now read

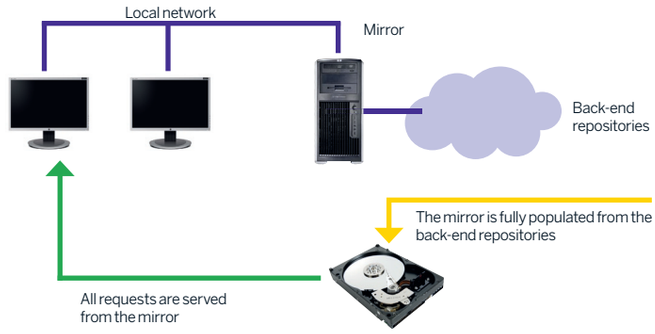
```
deb file:/repos/mirror/gb.archive.ubuntu.com/ubuntu hardy universe
```

This change will need to be made for all the repositories that you are mirroring.

Now, update the package information on the clients:

```
$ sudo apt-get update
```

and you should be all set!



› Mirroring is potentially an expensive operation. It makes a local copy of the entire back-end repository.

> **"It's not hard to create a repository and populate it with your own Debs."**

Roll your own

If you have locally developed packages you'd like to distribute within your enterprise, it's not hard to create your own repository and populate it with your own Debs. You'll need to have a web server installed, and you'll need to set up a directory structure under your web server's 'document root' broadly similar to the one in the main Ubuntu repositories (see page 63), and copy your Deb files in. With the Debs in place, the utility *dpkg-scanpackages* can be used to build the **Packages.gz** file. It generates uncompressed output on its standard output so you'll need to pipe into *gzip* or *bzip2* and send the output to the right file.

Well, I'm out of space, and must apologise to those Debian enthusiasts who are whispering urgently in my ear: "Tell 'em about feature X", for various values of X. LXF

Where to learn more

» For more on using *apt-proxy*, see **www.subvs.co.uk/apt-proxy_on_ubuntu** or **https://help.ubuntu.com/community/AptProxy**.
» For more on mirroring, see **www.howtoforge.com/local_debian_ubuntu_mirror** or **http://popey.com/Creating_an_Ubuntu_repository_mirror_with_apt-mirror**.
» The business of building your own repository is well described at **https://help.ubuntu.com/community/Repositories/Personal**.
» The Debian Policy Manual is an interesting (if slightly formal) read. Go to **www.debian.org/doc/debian-policy**.
Or of course you could also read the man pages on *apt-get*, *apt-cache*, *apt-proxy*, *apt-proxy-import*, *apt-mirror*.