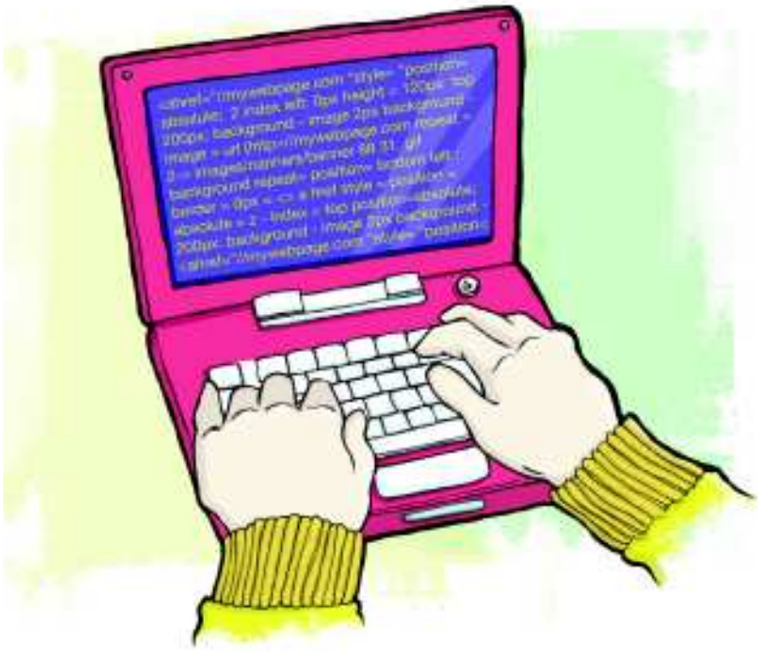
apt-get: Master

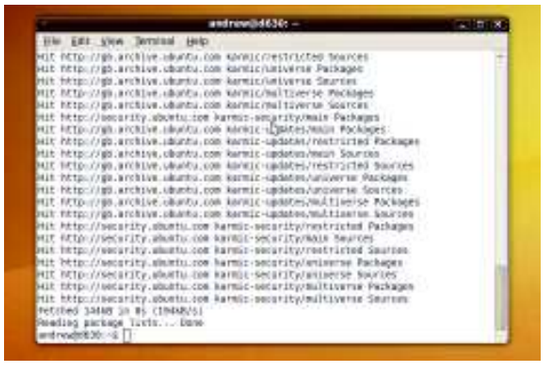
Ever wanted a one-stop place to do your package management and the associated housekeeping? **Andy Hudson** explains how, with this handy tool.



Our expert

Andy Hudson has been networking things since he was a nipper. He's written several books on Linux, and is the man we turn to if there's anything we can't fix ourselves.

Depending on which way you swing in the Linux world, you may be a fan of *portage*, *yum* or simply building your applications from source by hand. However, for the growing number of Debian-based distros, there is only the one true way to manage your packages, and that's with *apt-get*. It's not hard to understand why our Debian-loving brethren love *apt-get* so much: it's been a staple of the distro for a number of years now, is lighting fast and surprisingly easy to use, albeit with a large number of additional options for you to choose from.



› **After running a quick apt-get update, you'll hopefully see a list like this appear in your terminal window.**

Most people won't get beyond the basics, so in this tutorial we're going to show you how to get the most from this powerful package management tool, turning you from a beginner to an advanced user in a flash.

We're not going to assume that you've actually used *apt-get* before, simply because a lot of people only use *Synaptic*, the standard GUI utility of choice for the majority of Debian and Ubuntu users. So, the first thing to do is get yourself to the command line and get ready to rumble. Because *apt-get* makes system-wide changes, you either need to work as root, or with superuser powers before you're actually able to do anything. In this tutorial we'll assume that you've already moved to the root prompt; if you're using Ubuntu then you should prefix every command with **sudo** to elevate your privileges. The first thing you should type is simply:

```
apt-get update
```

This forces *apt-get* to update its information about available packages, using details gleaned from the **sources.list** file. It's important that you do this before you go any further, so as to ensure that you have the latest information about the most up-to-date packages. Once *apt-get* has finished processing, you can issue the command:

```
apt-get upgrade
```

This will download any newer versions of currently installed applications, along with their dependencies. It then does the necessary. Beyond that, the only other basic uses of *apt-get* that you need to know at this stage are:

```
apt-get install foo
```

This adds software, while the following removes it:

```
apt-get remove bar
```

So, with the essential commands out of the way, let's take a closer look at some of the more interesting things that you can do with *apt-get*.

Keep things tidy

First off, although *apt-get* is a pretty efficient package manager, it can generate a fair amount of cruft over time, which isn't necessarily handled automatically by *Apt* in the day-to-day running of your system. This cruft is mostly made up of packages that have already been processed and installed, but lie dormant in the filesystem.

Depending on how old your system is, you may never find this a real problem, particularly if you move between distros regularly. But if you're relying on a Debian install, or one of the Ubuntu LTS releases, then you'll feel the benefit of cleaning up every once in a while. Just issue the command:

```
apt-get clean
```

Apt will then clean up all the package files that are no longer needed, giving you back a fair chunk of disk space, especially

your packages

Bovine nonsense

If you've ever just run **apt-get** as a command, you'll get the usual list of available options, and more often than not there will also be a cryptic message saying 'This apt has Super Cow Powers' and not a lot else. The way to see the easter egg is to use the command:

```
apt-get moo
```

It displays an ASCII picture of a cow and a question that's bound to put a smile on your face. *Aptitude*, the higher-level companion to *apt*, has a similar function – try using:

```
aptitude moo && aptitude moo -v
```

This'll give you a taste of what's in store. Now keep adding an extra **v** on the end of the second command to see just how dry the humour is.

```
File  Edit  View  Terminal  Help
andrew@d630:~$ apt-get moo
         (__)
         (oo)
   /------\/
  / |    ||
 *  /\---/\
    ~~   ~~
....."Have you mooed today?"...
andrew@d630:~$
```

if you've been running your system for a while or have performed some major upgrades. The only potential downside with this is that if you need to do a re-install of a package, *apt-get* will need to download the package in question again. To avoid this, and if you're not too worried about disk space, then you may want to try out:

```
apt-get autoclean
```

This gets rid of any packages that are no longer required – for instance, if you've performed a couple of upgrades to a specific package on your system, then **autoclean** will remove all the extra packages, other than the one that's currently installed. If you find yourself doing a lot of re-installing, this will save you time, even if you do have a fast internet connection.

Another option, if you're particularly fastidious with your package management, is to use the **purge** switch rather than simply **remove**; doing so gets rid of the package in question, but also clears away any configuration files that were in use.

The final string in your *apt-get* maintenance armoury is:

```
apt-get autoremove
```

The **autoremove** command clears any packages that were installed to satisfy dependencies, but are no longer required.

Having now looked at how to clean up after *Apt*, what if you prefer to compile all your packages from source, *à la* Gentoo? Well, it's certainly possible with *apt-get*, and in fact, we'd argue that it's probably the easiest way to dip your toes into the compilation hot tub, particularly because it can take a lot of pain out of compiling. We'll start off by simply retrieving the source code, for which you'd fire off the command:

```
apt-get source foo
```

This will retrieve the source files, including a **.dsc** one that is used by *dpkg-source* to tell *apt-get* how to unpack the source code. This is usually into a directory under your current working folder, named after the source package. It's then ready for you to compile in the usual way.

Useful tricks

If you're a little daunted by the idea of having to work out how to compile the source, then *apt-get* has another nifty trick up its sleeve. You're able to ask it to compile the package automatically, as soon as it's finished downloading.

Of course, no package is an island, to coin a phrase, and it's likely that you would need to ensure that the dependencies are installed before you attempt any compilation. It's here that *apt-get* really starts to become handy. If you know the name of the package that you're going to be compiling, just use this command to install the dependencies that are required by said package:

```
apt-get build-dep bar
```

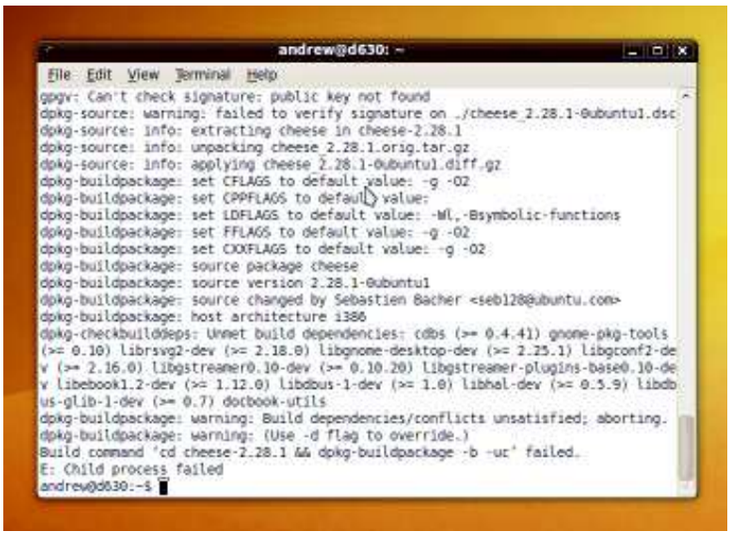
Apt-get will scan the repositories for the required dependencies and install them, but not the specified package itself at this stage. Only once this process has finished are you ready to issue the command:

```
apt-get source foo -b
```

This grabs the source code and compiles it. However, it's not actually installed yet – *apt-get* instead creates a **deb** package for you in the current working directory. All that's left to do is to use **dpkg** to install that package using the syntax:

```
dpkg -i packagename.deb
```

As you've already taken care of the dependencies, the package will be installed and you'll be ready to go. LXF



› Get your dependencies before you attempt to build your package from source.