APT HOWTO (Obsolete Documentation)

Gustavo Noronha Silva `<kov@debian.org>`

1.8.11 - August 2005

Abstract

This document intends to provide the user with a good understanding of the workings of the Debian package management utility, APT. Its goal is to make life easier for new Debian users and to help those who wish to deepen their understanding of the administration of this system. It was created for the Debian project in order to help improve the support available for users of this distribution.

Copyright Notice

Contents

Chapter 1

Introduction

In the beginning there was the .tar.gz. Users had to compile each program that they wanted to use on their GNU/Linux systems. When Debian was created, it was deemed necessary that the system include a method of managing the packages installed on the machine. The name `dpkg` was given to this system. Thus the famous 'package' first came into being on GNU/Linux, a while before Red Hat decided to create their own 'rpm' system.

A new dilemma quickly took hold of the minds of the makers of GNU/Linux. They needed a rapid, practical, and efficient way to install packages that would manage dependencies automatically and take care of their configuration files while upgrading. Here again, Debian led the way and gave birth to APT, the Advanced Packaging Tool, which has since been ported by Conectiva for use with rpm and has been adopted by some other distributions.

This manual makes no attempt to address apt-rpm, as the Conectiva port of APT is known, but "patches" to this document which do so would be welcome.

This manual is based on the next Debian release, `Sarge`.

Chapter 2

Basic Configuration

2.1 The /etc/apt/sources.list file

As part of its operation, APT uses a file that lists the 'sources' from which packages can be obtained. This file is `/etc/apt/sources.list`.

The entries in this file normally follow this format:

```
deb http://host/debian distribution section1 section2 section3
deb-src http://host/debian distribution section1 section2 section3
```

Of course, the above entries are fictitious and should not be used. The first word on each line, `deb` or `deb-src`, indicates the type of archive: whether it contains binary packages (`deb`), that is, the pre-compiled packages that we normally use, or source packages (`deb-src`), which are the original program sources plus the Debian control file (`.dsc`) and the `diff.gz` containing the changes needed for 'debianizing' the program.

We usually find the following in the default Debian sources.list:

```
# See sources.list(5) for more information, especially
# Remember that you can only use http, ftp or file URIs
# CDROMs are managed through the apt-cdrom tool.
deb http://http.us.debian.org/debian stable main contrib non-free
deb http://non-us.debian.org/debian-non-US stable/non-US main contrib non-fre
deb http://security.debian.org stable/updates main contrib non-free

# Uncomment if you want the apt-get source function to work
#deb-src http://http.us.debian.org/debian stable main contrib non-free
#deb-src http://non-us.debian.org/debian-non-US stable/non-US main contrib no
```

These are the lines needed by a basic Debian install. The first `deb` line points to the official archive, the second to the non-US archive and the third to the archive of Debian security updates.

The two last lines are commented out (with a '#' in front), so apt-get will ignore them. These are `deb-src` lines, that is, they point to Debian source packages. If you often download program sources for testing or recompiling, uncomment them.

The `/etc/apt/sources.list` file can contain several types of lines. APT knows how to deal with archives of types `http`, `ftp`, `file` (local files, e.g., a directory containing a mounted ISO9660 filesystem) and `ssh`, that I know of.

Do not forget to run `apt-get update` after modifying the `/etc/apt/sources.list` file. You must do this to let APT obtain the package lists from the sources you specified.

2.2 How to use APT locally

Sometimes you have lots of packages .deb that you would like to use APT to install so that the dependencies would be automatically solved.

To do that create a directory and put the .debs you want to index in it . For example:

```
# mkdir /root/debs
```

You may modify the definitions set on the package's control file directly for your repository using an `override` file. Inside this file you may want to define some options to override the ones that come with the package. It looks like follows:

```
package priority section
```

package is the name of the package, priority is low, medium or high and section is the section to which it belongs. The file name does not matter, you'll have to pass it as an argument for `dpkg-scanpackages` later. If you do not want to write an `override` file, just use `/dev /null`. when calling `dpkg-scanpackages`.

Still in the /root directory do:

```
# dpkg-scanpackages debs file | gzip > debs/Packages.gz
```

In the above line, *file* is the `override` file, the command generates a file `Packages.gz` that contains various information about the packages, which are used by APT. To use the packages, finally, add:

```
deb file:/root debs/
```

After that just use the APT commands as usual. You may also generate a sources repository. To do that use the same procedure, but remember that you need to have the files `.orig.tar.gz`, `.dsc` and `.diff.gz` in the directory and you have to use `Sources.gz` instead of `Packages.gz`. The program used is also different. It is `dpkg-scansources`. The command line will look like this:

```
# dpkg-scansources debs | gzip > debs/Sources.gz
```

Notice that `dpkg-scansources` doesn't need an `override` file. The sources.list's line is:

```
deb-src file:/root debs/
```

2.3 Deciding which mirror is the best to include in the sources.list file: netselect, netselect-apt

A very frequent doubt, mainly among the newest users is: "which Debian mirror to include in `sources.list`?". There are many ways to decide which mirror. The experts probably have a script that measures the ping time through the several mirrors. But there's a program that does this for us: **netselect**.

To install netselect, as usual:

```
# apt-get install netselect
```

Executing it without parameters shows the help. Executing it with a space-separated list of hosts (mirrors), it will return a score and one of the hosts. This score takes in consideration the estimated ping time and the hops (hosts by which a network query will pass by to reach the destination) number and is inversely proportional to the estimated download speed (so, the lower, the better). The returned host is the one that had the lowest score (the full list of scores can be seen adding the -vv option). See this example:

```
# netselect ftp.debian.org http.us.debian.org ftp.at.debian.org download.unes
  365 ftp.debian.org.br
#
```

This means that, from the mirrors included as parameters to netselect, `ftp.debian.org.br` was the best, with an score of 365. (Attention!! As it was done on my computer and the network topography is extremely different depending on the contact point, this value is not necessarily the right speed in other computers).

Now, just put the fastest mirror found by netselect in the `/etc/apt/sources.list` file (see 'The /etc/apt/sources.list file' on page 3) and follow the tips in 'Managing packages' on page 7.

Note: the list of mirrors may always be found in the file `http://www.debian.org/mirror/mirrors_full`.

Beginning with the 0.3.ds1 version, the netselect source package includes the **netselect-apt** binary package, which makes the process above automatic. Just enter the distribution tree as parameter (the default is stable) and the `sources.list` file will be generated with the best main and non-US mirrors and will be saved under the current directory. The following example generates a sources.list of the stable distribution:

```
# ls sources.list
ls: sources.list: File or directory not found
# netselect-apt stable
(...)
# ls -l sources.list
sources.list
#
```

Remember: the `sources.list` file is generated under the current directory, and must be moved to the `/etc/apt` directory.

Then, follow the tips in 'Managing packages' on the facing page.

2.4 Adding a CD-ROM to the sources.list file

If you'd rather use your CD-ROM for installing packages or updating your system automatically with APT, you can put it in your `sources.list`. To do so, you can use the `apt-cdrom` program like this:

```
# apt-cdrom add
```

with the Debian CD-ROM in the drive. It will mount the CD-ROM, and if it's a valid Debian CD it will look for package information on the disk. If your CD-ROM configuration is a little unusual, you can also use the following options:

```
-h           - program help
-d directory - CD-ROM mount point
-r           - Rename a recognized CD-ROM
-m           - No mounting
-f           - Fast mode, don't check package files
-a           - Thorough scan mode
```

For example:

```
# apt-cdrom -d /home/kov/mycdrom add
```

You can also identify a CD-ROM, without adding it to your list:

```
# apt-cdrom ident
```

Note that this program only works if your CD-ROM is properly configured in your system's `/etc/fstab`.

Chapter 3

Managing packages

3.1 Updating the list of available packages

The packaging system uses a private database to keep track of which packages are installed, which are not installed and which are available for installation. The `apt-get` program uses this database to find out how to install packages requested by the user and to find out which additional packages are needed in order for a selected package to work properly.

To update this list, you would use the command `apt-get update`. This command looks for the package lists in the archives found in `/etc/apt/sources.list`; see 'The /etc/apt/sources.list file' on page 3 for more information about this file.

It's a good idea to run this command regularly to keep yourself and your system informed about possible package updates, particularly security updates.

3.2 Installing packages

Finally, the process you've all been waiting for! With your sources.list ready and your list of available packages up to date, all you have to do is run `apt-get` to get your desired package installed. For example, you can run:

```
# apt-get install xchat
```

APT will search it's database for the most recent version of this package and will retrieve it from the corresponding archive as specified in `sources.list`. In the event that this package depends on another – as is the case here – APT will check the dependencies and install the needed packages. See this example:

```
# apt-get install nautilus
Reading Package Lists... Done
```

```
Building Dependency Tree... Done
The following extra packages will be installed:
  bonobo libmedusa0 libnautilus0
The following NEW packages will be installed:
  bonobo libmedusa0 libnautilus0 nautilus
0 packages upgraded, 4 newly installed, 0 to remove and 1  not upgraded.
Need to get 8329kB of archives. After unpacking 17.2MB will be used.
Do you want to continue? [Y/n]
```

The package `nautilus` depends on the shared libraries cited, therefore APT will get them from the archive. If you had specified the names of these libraries on the `apt-get` command line, APT would not have asked if you wanted to continue; it would automatically accept that you wanted to install all of those packages.

This means that APT only asks for confirmation when it needs to install packages which weren't specified on the command line.

The following options to apt-get may be useful:

```
-h  This help text.
-d  Download only - do NOT install or unpack archives
-f  Attempt to continue if the integrity check fails
-s  No-act. Perform ordering simulation
-y  Assume Yes to all queries and do not prompt
-u  Show a list of upgraded packages as well
```

Multiple packages may be selected for installation in one line. Files downloaded from the network are placed in the directory `/var/cache/apt/archives` for later installation.

You can specify packages to be removed on the same command line, as well. Just put a '-' immediately after the name of the package to be removed, like this:

```
# apt-get install nautilus gnome-panel-
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  bonobo libmedusa0 libnautilus0
The following packages will be REMOVED:
  gnome-applets gnome-panel gnome-panel-data gnome-session
The following NEW packages will be installed:
  bonobo libmedusa0 libnautilus0 nautilus
0 packages upgraded, 4 newly installed, 4 to remove and 1  not upgraded.
Need to get 8329kB of archives. After unpacking 2594kB will be used.
Do you want to continue? [Y/n]
```

See section 'Removing packages' on the facing page for more details on package removal.

If you somehow damage an installed package, or simply want the files of a package to be reinstalled with the newest version that is available, you can use the `--reinstall` option like so:

```
# apt-get --reinstall install gdm
Reading Package Lists... Done
Building Dependency Tree... Done
0 packages upgraded, 0 newly installed, 1 reinstalled, 0 to remove and 1  not
Need to get 0B/182kB of archives. After unpacking 0B will be used.
Do you want to continue? [Y/n]
```

3.3 Removing packages

If you no longer want to use a package, you can remove it from your system using APT. To do this just type: `apt-get remove package`. For example:

```
# apt-get remove gnome-panel
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  gnome-applets gnome-panel gnome-panel-data gnome-session
0 packages upgraded, 0 newly installed, 4 to remove and 1  not upgraded.
Need to get 0B of archives. After unpacking 14.6MB will be freed.
Do you want to continue? [Y/n]
```

As you can see in the above example, APT also takes care of removing packages which depend on the package you have asked to remove. There is no way to remove a package using APT without also removing those packages that depend on it.

Running `apt-get` as above will cause the packages to be removed but their configuration files, if any, will remain intact on the system. For a complete removal of the package, run:

```
# apt-get --purge remove gnome-panel
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  gnome-applets* gnome-panel* gnome-panel-data* gnome-session*
0 packages upgraded, 0 newly installed, 4 to remove and 1  not upgraded.
Need to get 0B of archives. After unpacking 14.6MB will be freed.
Do you want to continue? [Y/n]
```

Note the '*' after the names. This indicates that the configuration files for each of these packages will also be removed.

Just as in the case of the `install` method, you can use a symbol with `remove` to invert the meaning for a particular package. In the case of removing, if you add a `'+'` right after the package name, the package will be installed instead of being removed.

```
# apt-get --purge remove gnome-panel nautilus+
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  bonobo libmedusa0 libnautilus0 nautilus
The following packages will be REMOVED:
  gnome-applets* gnome-panel* gnome-panel-data* gnome-session*
The following NEW packages will be installed:
  bonobo libmedusa0 libnautilus0 nautilus
0 packages upgraded, 4 newly installed, 4 to remove and 1  not upgraded.
Need to get 8329kB of archives. After unpacking 2594kB will be used.
Do you want to continue? [Y/n]
```

Note that `apt-get` lists the extra packages which will be installed (that is, the packages whose installation is needed for the proper functioning of the package whose installation has been requested), those which will be removed, and those which will be installed (including the extra packages again).

3.4 Upgrading packages

Package upgrades are a great success of the APT system. They can be achieved with a single command: `apt-get upgrade`. You can use this command to upgrade packages within the same distribution, as well as to upgrade to a new distribution, although for the latter the command `apt-get dist-upgrade` is preferred; see section 'Upgrading to a new release' on the next page for more details.

It's useful to run this command with the `-u` option. This option causes APT to show the complete list of packages which will be upgraded. Without it, you'll be upgrading blindly. APT will download the latest versions of each package and will install them in the proper order. It's important to always run `apt-get update` before you try this. See section 'Updating the list of available packages' on page 7. Look at this example:

```
# apt-get -u upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages have been kept back
  cpp gcc lilo
The following packages will be upgraded
  adduser ae apt autoconf debhelper dpkg-dev esound esound-common ftp indent
  ipchains isapnptools libaudiofile-dev libaudiofile0 libesd0 libesd0-dev
  libgtk1.2 libgtk1.2-dev liblockfile1 libnewt0 liborbit-dev liborbit0
  libstdc++2.10-glibc2.2 libtiff3g libtiff3g-dev modconf orbit procps psmisc
29 packages upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Need to get 5055B/5055kB of archives. After unpacking 1161kB will be used.
Do you want to continue? [Y/n]
```

The process is very simple. Note that in the first few lines, `apt-get` says that some packages were `kept back`. This means that there are new versions of these packages which will not be installed for some reason. Possible reasons are broken dependencies (a package on which it depends doesn't have a version available for download) or new dependencies (the package has come to depend on new packages since the last version).

There's no clean solution for this first case. For the second case, it's sufficient to run `apt-get install` for the specific package in question, as this will download the dependencies. An even cleaner solution is to use `dist-upgrade`. See section 'Upgrading to a new release' on the current page.

3.5 Upgrading to a new release

This feature of APT allows you to upgrade an entire Debian system at once, either through the Internet or from a new CD (purchased or downloaded as an ISO image).

It is also used when changes are made to the relationships between installed packages. With `apt-get upgrade`, these packages would be kept untouched (`kept back`).

For example, suppose that you're using revision 0 of the stable version of Debian and you buy a CD with revision 3. You can use APT to upgrade your system from this new CD. To do this, use `apt-cdrom` (see section 'Adding a CD-ROM to the sources.list file' on page 6) to add the CD to your `/etc/apt/sources.list` and run `apt-get dist-upgrade`.

It's important to note that APT always looks for the most recent versions of packages. Therefore, if your `/etc/apt/sources.list` were to list an archive that had a more recent version of a package than the version on the CD, APT would download the package from there.

In the example shown in section 'Upgrading packages' on the preceding page, we saw that some packages were `kept back`. We'll solve this problem now with the `dist-upgrade` method:

```
# apt-get -u dist-upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
Calculating Upgrade... Done
The following NEW packages will be installed:
  cpp-2.95 cron exim gcc-2.95 libident libopenldap-runtime libopenldap1
  libpcre2 logrotate mailx
The following packages have been kept back
```

```
  lilo
The following packages will be upgraded
  adduser ae apt autoconf cpp debhelper dpkg-dev esound esound-common ftp gcc
  indent ipchains isapnptools libaudiofile-dev libaudiofile0 libesd0
  libesd0-dev libgtk1.2 libgtk1.2-dev liblockfile1 libnewt0 liborbit-dev
  liborbit0 libstdc++2.10-glibc2.2 libtiff3g libtiff3g-dev modconf orbit
  procps psmisc
31 packages upgraded, 10 newly installed, 0 to remove and 1 not upgraded.
Need to get 0B/7098kB of archives. After unpacking 3118kB will be used.
Do you want to continue? [Y/n]
```

Note now that the packages will be upgraded, and new packages will also be installed (the new dependencies of the packages). Note too that lilo is still being `kept back`. It probably has a more serious problem than a new dependency. We can find out by running:

```
# apt-get -u install lilo
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  cron debconf exim libident libopenldap-runtime libopenldap1 libpcre2
  logrotate mailx
The following packages will be REMOVED:
  debconf-tiny
The following NEW packages will be installed:
  cron debconf exim libident libopenldap-runtime libopenldap1 libpcre2
  logrotate mailx
The following packages will be upgraded
  lilo
1 packages upgraded, 9 newly installed, 1 to remove and 31 not upgraded.
Need to get 225kB/1179kB of archives. After unpacking 2659kB will be used.
Do you want to continue? [Y/n]
```

As noted in the above, lilo has a new conflict with the package `debconf-tiny`, which means it couldn't be installed (or upgraded) without removing debconf-tiny.

To know what's keeping or removing a package you may use:

```
# apt-get -o Debug::pkgProblemResolver=yes dist-upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
Calculating Upgrade... Starting
Starting 2
Investigating python1.5
Package python1.5 has broken dep on python1.5-base
  Considering python1.5-base 0 as a solution to python1.5 0
  Holding Back python1.5 rather than change python1.5-base
Investigating python1.5-dev
Package python1.5-dev has broken dep on python1.5
  Considering python1.5 0 as a solution to python1.5-dev 0
  Holding Back python1.5-dev rather than change python1.5
 Try to Re-Instate python1.5-dev
Done
Done
The following packages have been kept back
  gs python1.5-dev
0 packages upgraded, 0 newly installed, 0 to remove and 2  not upgraded.
```

This way, it's easy to notice that the python1.5-dev package cannot be installed because of an unsatisfied dependency: python1.5.

3.6 Removing unused package files: apt-get clean and autoclean

When you install a package APT retrieves the needed files from the hosts listed in /etc/apt/sources.list, stores them in a local repository (`/var/cache/apt/archives/`), and then proceeds with installation, see 'Installing packages' on page 7.

In time the local repository can grow and occupy a lot of disk space. Fortunately, APT provides tools for managing its local repository: `apt-get`'s `clean` and `autoclean` methods.

`apt-get clean` removes everything except lock files from `/var/cache/apt/archives/` and `/var/cache/apt/archives/partial/`. Thus, if you need to reinstall a package APT should retrieve it again.

`apt-get autoclean` removes only package files that can no longer be downloaded.

The following example show how apt-get autoclean works:

```
# ls /var/cache/apt/archives/logrotate* /var/cache/apt/archives/gpm*
logrotate_3.5.9-7_i386.deb
logrotate_3.5.9-8_i386.deb
gpm_1.19.6-11_i386.deb
```

In /var/cache/apt/archives there are two files for the package `logrotate` and one for the package `gpm`.

```
# apt-show-versions -p logrotate
logrotate/stable uptodate 3.5.9-8
# apt-show-versions -p gpm
gpm/stable upgradeable from 1.19.6-11 to 1.19.6-12
```

apt-show-versions shows that logrotate_3.5.9-8_i386.deb provides the up to date version of logrotate, so logrotate_3.5.9-7_i386.deb is useless. Also gpm_1.19.6-11_i386.deb is useless because a more recent version of the package can be retrieved.

```
# apt-get autoclean
Reading Package Lists... Done
Building Dependency Tree... Done
Del gpm 1.19.6-11 [145kB]
Del logrotate 3.5.9-7 [26.5kB]
```

Finally, apt-get autoclean removes only the old files. See 'How to upgrade packages from specific versions of Debian' on page 16 for more information on apt-show-versions.

3.7 Using APT with dselect

dselect is a program that helps users select Debian packages for installation. It's considered somewhat complicated and rather boring, but with practice you can get the hang of its console-based ncurses interface.

One feature of dselect is that it knows how to make use of the capacity Debian packages have for "recommending" and "suggesting" other packages for installation. To use the program, run `dselect' as root. Choose 'apt' as your access method. This isn't truly necessary, but if you're not using a CD ROM and you want to download packages from the Internet, it's the best way to use dselect.

To gain a better understanding of dselect's usage, read the dselect documentation found on the Debian page http://www.debian.org/doc/ddp.

After making your selections with dselect, use:

```
# apt-get -u dselect-upgrade
```

as in the example below:

```
# apt-get -u dselect-upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  lbxproxy
The following NEW packages will be installed:
  bonobo console-tools-libs cpp-3.0 enscript expat fingerd gcc-3.0
  gcc-3.0-base icepref klogd libdigest-md5-perl libfnlib0 libft-perl
  libgc5-dev libgcc300 libhtml-clean-perl libltdl0-dev libsasl-modules
  libstdc++3.0 metamail nethack proftpd-doc psfontmgr python-newt talk tidy
  util-linux-locales vacation xbill xplanet-images
The following packages will be upgraded
  debian-policy
1 packages upgraded, 30 newly installed, 1 to remove and 0  not upgraded.
Need to get 7140kB of archives. After unpacking 16.3MB will be used.
Do you want to continue? [Y/n]
```

Compare with what we see when running apt-get dist-upgrade on the same system:

```
# apt-get -u dist-upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
Calculating Upgrade... Done
The following packages will be upgraded
  debian-policy
1 packages upgraded, 0 newly installed, 0 to remove and 0  not upgraded.
Need to get 421kB of archives. After unpacking 25.6kB will be freed.
Do you want to continue? [Y/n]
```

Note that many of the packages from above are being installed because other packages "suggested" or "recommended" them. Others are being installed or removed (in the case of lbxproxy, for example) per the choices we made while navigating through dselect's package listing. Dselect can be a powerful tool when used in conjunction with APT.

3.8 How to keep a mixed system

People are sometimes interested in using one of the Debian versions as its main system distribution and one or more packages from another branch.

To set up what is your main version of Debian you should edit the `/etc/apt/apt.conf` (it does not usually exist, create it if you don't have one) to contain the following line:

```
APT::Default-Release "version";
```

Where *version* is the version of Debian you want to use as the main distribution. The versions you can use are `stable`, `testing` and `unstable`. To install packages from another version, then, you must use APT in the following way:

```
# apt-get -t distribution install package
```

For that to work, though, you need at least one APT source line in your `/etc/apt/sources.list` for the distribution you want the package from, and the package must exist on that source.

You can also request a specific version of a package using the following sintax:

```
# apt-get install package=version
```

For example, the line below will install version `2.2.4-1` of the `nautilus` package.:

```
# apt-get install nautilus=2.2.4-1
```

IMPORTANT: the 'unstable' version of Debian is the version to which the newest versions of Debian packages are uploaded first. This distribution sees all of the changes that packages go through, both small ones and more drastic ones which affect many packages or the whole system. For this reason, this version of the distribution should *not* be used by inexperienced users or by those who need proven stability.

The 'testing' distribution is not necessarily better than 'unstable', because it does not receive security updates quickly. For servers and other production systems stable should always be used.

3.9 How to upgrade packages from specific versions of Debian

`apt-show-versions` provides a safe way for users of mixed distributions to upgrade their systems without getting more of the less-stable distribution than they had in mind. For instance, it is possible to upgrade just your unstable packages by running after having installed the `apt-show-versions` package:

```
# apt-get install `apt-show-versions -u -b | grep unstable | cut -d ' ' -f 1`
```

3.10 How to keep specific versions of packages installed (complex)

You may have occasion to modify something in a package and don't have time or don't want to port those changes to a new version of the program. Or, for instance, you may have just upgraded your Debian distribution to 3.0, but want to continue with the version of a certain package from Debian 2.2. You can "pin" the version you have installed so that it will not be upgraded.

Using this resource is simple. You just need to edit the file `/etc/apt/preferences`.

The format is simple:

```
Package: <package>
Pin: <pin definition>
Pin-Priority: <pin's priority>
```

Each entry must be separated from any other entries by a blank line. For example, to keep package `sylpheed` that I have modified to use "reply-to-list" at version 0.4.99, I add:

```
Package: sylpheed
Pin: version 0.4.99*
```

Note that I used an * (asterisk). This is a "wildcard"; it say that I want that this "pin" to be valid for all versions beginning with 0.4.99. This is because Debian versions its packages with a "Debian revision" and I don't want to avoid the installation of these revisions. So, for instance, versions 0.4.99-1 and 0.4.99-10 will be installed as soon as they are made available. Note that if you modified the package you won't want to do things this way.

The pin priority helps determine whether a package matching the "Packages:" and "Pin:" lines will be installed, with higher priorities making it more likely that a matching package will be installed. You can read `apt_preferences(7)` for a thorough discussion of priorities, but a few examples should give the basic idea. The following describes the effect of setting the priority field to different values in the sylpheed example above.

1001 Sylpheed version 0.4.99 will never be replaced by apt. If available, apt will install version 0.4.99 even if it would replace an installed package with a higher version. Only packages of priority greater than 1000 will ever downgrade an existing package.

1000 The effect is the same as priority 1001, except that apt will refuse to downgrade an installed version to 0.4.99

990 Version 0.4.99 will be replaced only by a higher version available from a release designated as preferred using the "APT::Default-Release" variable (see 'How to keep a mixed system' on page 15, above).

500 Any version higher than 0.4.99 of sylpheed which is available from any release will take preference over version 0.4.99, but 0.4.99 will still be preferred to a lower version.

100 Higher versions of sylpheed available from any release will take preference over version 0.4.99, as will any installed higher version of slypheed; so 0.4.99 will be installed only if no version is installed already. This is the priority of installed packages.

-1 Negative priorities are allowed as well, and prevent 0.4.99 from ever being installed.

A pin can be specified on a package's `version`, `release` or `origin`.

Pinning on a `version`, as we have seen, supports literal version numbers as well as wildcards to specify several versions at one time.

Option `release` depends on the Release file from an APT repository or from a CD. This option may be of no use at all if you're using package repositories that don't provide this file. You may see the contents of the Release files that you have on `/var/lib/apt/lists/`. The parameters for a release are: `a` (archive), `c` (components), `v` (version), `o` (origin) and `l` (label).

An example:

```
Package: *
Pin: release v=2.2*,a=stable,c=main,o=Debian,l=Debian
Pin-Priority: 1001
```

In this example, we chose version 2.2* of Debian (which can be 2.2r2, 2.2r3 – this accommodates "point releases" that typically include security fixes and other very important updates), the `stable` repository, section `main` (as opposed to `contrib` or `non-free`) and origin and label Debian. Origin (o=) defines who produced that Release file, the label (l=) defines the name of the distribution: Debian for Debian itself and Progeny for Progeny, for example. A sample Release file:

```
$ cat /var/lib/apt/lists/ftp.debian.org.br_debian_dists_potato_main_binary-i3
Archive: stable
Version: 2.2r3
Component: main
Origin: Debian
Label: Debian
Architecture: i386
```

Chapter 4

Very useful helpers

4.1 How to install locally compiled packages: equivs

Sometimes, people want to use a specific version of a program available only on source code, with no Debian package. But the packaging system can be a trouble when doing this. Suppose you want to compile a new version of your email server. All is fine, but many packages in Debian depend on an MTA (Mail Transport Agent). Since you installed something you compiled by yourself, the packaging system doesn't know about it.

That's where `equivs` enters the scene. To use it, install the package with that name. Equivs creates an empty package that fullfills dependencies, making the package system believe that the dependencies are satisfied.

Before we begin, it is good to remind you that there are safer ways of compiling a program which is already packaged for Debian with different options, and that one should not use equivs to replace dependencies if you don't know what you are doing. See section 'Working with source packages' on page 29 for more information.

Let's continue with the MTA example, you just installed your new compiled `postfix` and goes on for installing `mutt`. Suddenly you discover that `mutt` wants to install another MTA. But you already have yours.

Go to some directory (`/tmp`, for example) and run:

```
# equivs-control name
```

Replace *name* for the name of the control file you want to create. The file will be created as follows:

```
Section: misc
Priority: optional
Standards-Version: 3.0.1
```

```
Package: <enter package name; defaults to equivs-dummy>
Version: <enter version here; defaults to 1.0>
Maintainer: <your name and email address; defaults to username>
Pre-Depends: <packages>
Depends: <packages>
Recommends: <packages>
Suggests: <package>
Provides: <(virtual)package>
Architecture: all
Copyright: <copyright file; defaults to GPL2>
Changelog: <changelog file; defaults to a generic changelog>
Readme: <README.Debian file; defaults to a generic one>
Extra-Files: <additional files for the doc directory, comma-separated>
Description: <short description; defaults to some wise words>
 long description and info
 .
 second paragraph
```

We just need modify this to do what we want. Have a look at the field's format and to their descriptions, there's no need to explain each one here, let's do what's required:

```
Section: misc
Priority: optional
Standards-Version: 3.0.1

Package: mta-local
Provides: mail-transport-agent
```

Yes, that's all. `mutt` depends on `mail-transport-agent`, that is a virtual package provided by all MTAs, I could simply name the package `mail-transport-agent`, but I preferred to use the virtual package's schema, using Provides.

Now you only need to build the package:

```
# equivs-build name
dh_testdir
touch build-stamp
dh_testdir
dh_testroot
dh_clean -k
# Add here commands to install the package into debian/tmp.
touch install-stamp
dh_testdir
dh_testroot
dh_installdocs
```

```
dh_installchangelogs
dh_compress
dh_fixperms
dh_installdeb
dh_gencontrol
dh_md5sums
dh_builddeb
dpkg-deb: building package `name' in `../name_1.0_all.deb'.

The package has been created.
Attention, the package has been created in the current directory,
```

And install the resulting `.deb`.

As one can see, there are several uses for `equivs`. One can even crate a `my-favorites` package, which depends on the programs you usually installs, for example. Just free your imagination, but be careful.

It is important to note that there are example control files in `/usr/share/doc/equivs/examples`. Check them out.

4.2 Removing unused locale files: localepurge

Many Debian users use only one locale. A Brazilian Debian user, for example, usually uses the `pt_BR` locale all the time and doesn't care about the `es` one.

`localepurge` is a very useful tool for these users. You can free lots of space by having only the locales that you really use. Just `apt-get install localepurge`.

It is very easy to configure it, debconf questions guide the user in a step-by-step configuration. Be very careful on answering the first question though, wrong answers may remove all the locales files, even the ones you use. The only way to recover these files is reinstalling all the packages that provide them.

4.3 How to know what packages may be upgraded

`apt-show-versions` is a program that shows what packages in the system may be updated and several useful information. The `-u` option displays a list of upgradeable packages:

```
$ apt-show-versions -u
libeel0/unstable upgradeable from 1.0.2-5 to 1.0.2-7
libeel-data/unstable upgradeable from 1.0.2-5 to 1.0.2-7
```

Chapter 5

Getting information about packages.

There are some front-end programs for the APT system that make it significantly easier to get listings of packages that are available for installation or are already installed, as well as to find out what section a package is in, what its priority is, what its description is, etc.

But… our goal here is to learn how to use pure APT. So how can you find out the name of a package that you want to install?

We have a number of resources for such a task. We'll begin with `apt-cache`. This program is used by the APT system for maintaining its database. We'll take just a brief look at some of its more practical applications.

5.1 Discovering package names

For example, suppose that you want to reminisce about the good old days of the Atari 2600. You want to use APT to install an Atari emulator, and then download some games. You can do:

```
# apt-cache search atari
atari-fdisk-cross - Partition editor for Atari (running on non-Atari)
circuslinux - The clowns are trying to pop balloons to score points!
madbomber - A Kaboom! clone
tcs - Character set translator.
atari800 - Atari emulator for svgalib/X/curses
stella - Atari 2600 Emulator for X windows
xmess-x - X binaries for Multi-Emulator Super System
```

We find several packages related to what we're looking for, together with brief descriptions. To get more information about a specific package, I can then use:

```
# apt-cache show stella
```

```
Package: stella
Priority: extra
Section: non-free/otherosfs
Installed-Size: 830
Maintainer: Tom Lear <tom@trap.mtview.ca.us>
Architecture: i386
Version: 1.1-2
Depends: libc6 (>= 2.1), libstdc++2.10, xlib6g (>= 3.3.5-1)
Filename: dists/potato/non-free/binary-i386/otherosfs/stella_1.1-2.deb
Size: 483430
MD5sum: 11b3e86a41a60fa1c4b334dd96c1d4b5
Description: Atari 2600 Emulator for X windows
 Stella is a portable emulator of the old Atari 2600 video-game console
 written in C++.  You can play most Atari 2600 games with it.  The latest
 news, code and binaries for Stella can be found at:
 http://www4.ncsu.edu/~bwmott/2600
```

In this output you have many details about the package that you want (or don't want) to install, together with the full description of the package. If the package is already installed on your system and there is a newer version, you'll see information about both versions. For example:

```
# apt-cache show lilo
Package: lilo
Priority: important
Section: base
Installed-Size: 271
Maintainer: Russell Coker <russell@coker.com.au>
Architecture: i386
Version: 1:21.7-3
Depends: libc6 (>= 2.2.1-2), debconf (>=0.2.26), logrotate
Suggests: lilo-doc
Conflicts: manpages (<<1.29-3)
Filename: pool/main/l/lilo/lilo_21.7-3_i386.deb
Size: 143052
MD5sum: 63fe29b5317fe34ed8ec3ae955f8270e
Description: LInux LOader - The Classic OS loader can load Linux and others
 This Package contains lilo (the installer) and boot-record-images to
 install Linux, OS/2, DOS and generic Boot Sectors of other OSes.
 .
 You can use Lilo to manage your Master Boot Record (with a simple text scree
 or call Lilo from other Boot-Loaders to jump-start the Linux kernel.

Package: lilo
Status: install ok installed
Priority: important
```

```
Section: base
Installed-Size: 190
Maintainer: Vincent Renardias <vincent@debian.org>
Version: 1:21.4.3-2
Depends: libc6 (>= 2.1.2)
Recommends: mbr
Suggests: lilo-doc
Description: LInux LOader - The Classic OS loader can load Linux and others
 This Package contains lilo (the installer) and boot-record-images to
 install Linux, OS/2, DOS and generic Boot Sectors of other OSes.
 .
 You can use Lilo to manage your Master Boot Record (with a simple text scree
 or call Lilo from other Boot-Loaders to jump-start the Linux kernel.
```

Note that the first in the list is the available package and the second is the one already installed. For more general information about a package, you can use:

```
# apt-cache showpkg penguin-command
Package: penguin-command
Versions:
1.4.5-1(/var/lib/apt/lists/download.sourceforge.net_debian_dists_unstable_mai

Reverse Depends:
Dependencies:
1.4.5-1 - libc6 (2 2.2.1-2) libpng2 (0 (null)) libsdl-mixer1.1 (2 1.1.0) libs
Provides:
1.4.5-1 -
Reverse Provides:
```

And to just find out what packages it depends on:

```
# apt-cache depends penguin-command
penguin-command
  Depends: libc6
  Depends: libpng2
  Depends: libsdl-mixer1.1
  Depends: libsdl1.1
  Depends: zlib1g
```

In summary, we have a range of weapons we can use to find out the name of a package we want.

5.2 Using dpkg to find package names

One of the ways to locate the name of a package is to know the name of an important file found within the package. For example, to find the package that provides a particular ".h" file you need for compilation you can run:

```
# dpkg -S stdio.h
libc6-dev: /usr/include/stdio.h
libc6-dev: /usr/include/bits/stdio.h
perl: /usr/lib/perl/5.6.0/CORE/nostdio.h
```

or:

```
# dpkg -S /usr/include/stdio.h
libc6-dev: /usr/include/stdio.h
```

To find out the names of packages installed on your system, which is useful, for example, if you plan to clean up your hard drive, you can run:

```
# dpkg -l | grep mozilla
ii  mozilla-browse 0.9.6-7        Mozilla Web Browser
```

The problem with this command is that it can "break" the package name. In the example above, the full name of the package is `mozilla-browser`. To fix this, you can use the `COLUMNS` environment variable this way:

```
[kov]@[couve] $ COLUMNS=132 dpkg -l | grep mozilla
ii  mozilla-browser             0.9.6-7                       Mozilla Web Brows
```

or the description or part of it this way:

```
# apt-cache search "Mozilla Web Browser"
mozilla-browser - Mozilla Web Browser
```

5.3 How to install packages "on demand"

You're compiling a program and, all of a sudden, boom! There's an error because it needs a `.h` file you don't have. The program `auto-apt` can save you from such scenarios. It asks you to install packages if they're needed, stopping the relevant process and continuing once the package is installed.

What you do, basically, is run:

```
# auto-apt run command
```

Where 'command' is the command to be executed that may need some unavailable file. For example:

```
# auto-apt run ./configure
```

It will then ask to install the needed packages and call apt-get automatically. If you're running X, a graphical interface will replace the default text interface.

Auto-apt keeps databases which need to be kept up-to-date in order for it to be effective. This is achieved by calling the commands `auto-apt update`, `auto-apt updatedb` and `auto-apt update-local`.

5.4 How to discover to which package a file belongs

If you want to install a package, and you can't find out what it is called by searching with `apt-cache`, but know the filename of the program itself, or some other filename that belongs to the package, then you can use `apt-file` to find the package name. This is done like this:

```
$ apt-file search filename
```

It works just like `dpkg -S`, but will also show you uninstalled packages that contain the file. It could also be used to find what packages contain necessary include files that are missing when compiling programs, although `auto-apt` is a much better method of solving such issues, see 'How to install packages "on demand"' on the facing page.

You can also list the contents of a package, by running:

```
$ apt-file list packagename
```

`apt-file` keeps a database of which files all packages contain, just like auto-apt does and it needs to be up-to-date. This is done by running:

```
# apt-file update
```

By default, `apt-file` uses the same database `auto-apt` is using, see 'How to install packages "on demand"' on the preceding page.

5.5 How to keep informed about the changes in the packages.

Every package installs in its documentation directory (`/usr/share/doc/packagename`) a file called `changelog.Debian.gz` which contains the list of changes made to the package since the last version. You can read these files with `zless`' help, for example, but it is something not so easy, after an complete system upgrade, to start searching changelogs for every upgraded package.

There's a way to automatize this task by means of a tool called `apt-listchanges`. To begin with one needs to install the `apt-listchanges` package. During the package installation, Debconf will configure it. Some questions may not be shown to you depending on the priority you set up Debconf to use. Answer to the questions as you want.

The first question asks how you want the changes to be showed by apt-listchanges. You can have them mailed to you, which is good for automatic upgrades, or you can ask them in a pager like `less`, so you can inspect the changes before leting the upgrade continue. If you don't want `apt-listchanges` running automaticaly during upgrades you can answer `none`.

After apt-listchanges is installed, as soon as packages are downloaded (or gotten from a CD or mounted disk) by apt it will show the lists of changes made to those packages before installing them.

Chapter 6

Working with source packages

6.1 Downloading source packages

It's common in the world of free software to study source code or even make corrections to buggy code. To do this, you would need to download the source of the program. The APT system provides an easy way to obtain source code to the many programs contained in the distribution, including all the files needed to create a .deb for the program.

Another common use of Debian sources is to adapt a more recent version of a program, from the unstable distribution, for example, for use with the stable distribution. Compiling a package against stable will generate .debs with dependencies adjusted to match the packages available in this distribution.

To accomplish this, the `deb-src` entry in your `/etc/apt/sources.list` should be pointed at unstable. It should also be enabled (uncommented). See section 'The /etc/apt/sources.list file' on page 3.

To download a source package, you would use the following command:

```
$ apt-get source packagename
```

This will download three files: a `.orig.tar.gz`, a `.dsc` and a `.diff.gz`. In the case of packages made specifically for Debian, the last of these is not downloaded and the first usually won't have "`orig`" in the name.

The `.dsc` file is used by dpkg-source for unpacking the source package into the directory *packagename-version*. Within each downloaded source package there is a `debian/` directory that contains the files needed for creating the .deb package.

To auto-build the package when it's been downloaded, just add `-b` to the command line, like this:

```
$ apt-get -b source packagename
```

If you decide not to create the .deb at the time of the download, you can create it later by running:

```
$ dpkg-buildpackage -rfakeroot -uc -b
```

from within the directory that was created for the package after downloading. To install the package built by the commands above one must use the package manager directly, like this:

```
# dpkg -i file.deb
```

There's a difference between `apt-get`'s `source` method and its other methods. The `source` method can be used by normal users, without needing special root powers. The files are downloaded to the directory from which the `apt-get source package` command was called.

6.2 Packages needed for compiling a source package

Normally, specific headers and shared libraries need to be present in order for a source package to be compiled. All source packages have a field in their control files called 'Build-Depends:' that indicates which additional packages are needed for the package to be built from source.

APT has a simple way of downloading these packages. Just run `apt-get build-dep package`, where 'package' is the name of the package you're going to build. For example:

```
# apt-get build-dep gmc
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  comerr-dev e2fslibs-dev gdk-imlib-dev imlib-progs libgnome-dev libgnorba-de
  libgpmg1-dev
0 packages upgraded, 7 newly installed, 0 to remove and 1  not upgraded.
Need to get 1069kB of archives. After unpacking 3514kB will be used.
Do you want to continue? [Y/n]
```

The packages that will be installed are the packages needed in order for `gmc` to be built correctly. It's important to note that this command doesn't look for the source package of the program to be compiled. You will therefore need to run `apt-get source` separately to get it.

If all you want is checking what packages are needed to build a given package, there's a variant of the `apt-cache show` command (see 'Getting information about packages.' on page 23, which will show, among other informations, the `Build-Depends` line that lists those information, the `Build-Depends` line that lists those

```
# apt-cache showsrc package
```

Chapter 7

How to deal with errors

7.1 Common errors

Errors will always happen, many of them caused by users not paying attention. The following is a list of some of the most frequently reported errors and how to deal with them.

If you receive a message that looks like the one below when trying to run `apt-get install package...`

```
Reading Package Lists... Done
Building Dependency Tree... Done
W: Couldn't stat source package list 'http://people.debian.org unstable/ Pack
W: You may want to run apt-get update to correct these missing files
E: Couldn't find package penguineyes
```

you forgot to run `apt-get update` after your last change to the `/etc/apt/sources.list` file.

If the error looked like:

```
E: Could not open lock file /var/lib/dpkg/lock - open (13 Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root
```

when trying any `apt-get` method other than `source`, you don't have root permission, that is, you're running as a normal user.

There's an error similar to the above which happens when you run two copies of `apt-get` at the same time, or even if you try to run `apt-get` while a `dpkg` process is active. The only method that can be used simultaneously with others is the `source` method.

If an installation breaks in the middle of the process and you find that it's no longer possible to install or remove packages, try running these two commands:

```
# apt-get -f install
# dpkg --configure -a
```

And then try again. It may be necessary to run the second of the above commands more than once. This is an important lesson for those adventurers who use ‘unstable’.

If you receive the error “E: Dynamic MMap ran out of room” when running `apt-get update`, add the following line to `/etc/apt/apt.conf`:

```
APT::Cache-Limit 10000000;
```

7.2 Where can I find help?

If you find yourself plagued by doubts, consult the extensive documentation available for the Debian packaging system. `--help`’s and manpages can be an enormous help to you, as can the documentation contained in the `/usr/share/doc` directories such as `/usr/share/doc/apt`.

If this documentation fails to drive your fears away, try looking for the answer on the Debian mailing lists. You can find more information about specific user lists on the Debian website: http://www.debian.org.

Remember that these lists and resources should be used only by Debian users; users of other systems will find better support from the community resources of their own distributions.

Chapter 8

What distributions support APT?

Here are the names of some of the distributions that use APT:

Debian GNU/Linux (http://www.debian.org) - it was for this distribution that APT was developed

Conectiva (http://www.conectiva.com.br) - this was the first distribution to port APT for use with rpm

Libranet (http://www.libranet.com)

Mandrake (http://www.mandrake.com)

PLD (http://www.pld.org.pl)

Vine (http://www.vinelinux.org)

APT4RPM (http://apt4rpm.sf.net)

Alt Linux (http://www.altlinux.ru/)

Red Hat (http://www.redhat.com/)

Sun Solaris (http://www.sun.com/)

SuSE (http://www.suse.de/)

Yellow Dog Linux (http://www.yellowdoglinux.com/)

Chapter 9

Credits

A big thank you goes out to my great friends in the Debian-BR project, and in Debian itself, who are a constant help to me and always give me the strength to continue working for humanity's benefit, as well as helping me with my goal of saving the world. :)

I also want to thank the CIPSGA for the enormous help it has given to our project and to all the free projects that spring from great ideas.

And special thanks to:

Yooseong Yang <yooseong@debian.org>

Michael Bramer <grisu@debian.org>

Bryan Stillwell <bryan@bokeoa.com>

Pawel Tecza <pawel.tecza@poczta.fm>

Hugo Mora <h.mora@melix.com.mx>

Luca Monducci <luca.mo@tiscali.it>

Tomohiro KUBOTA <kubota@debian.org>

Pablo Lorenzzoni <spectra@debian.org>

Steve Langasek <vorlon@netexpress.net>

Arnaldo Carvalho de Melo <acme@conectiva.com.br>

Erik Rossen <rossen@freesurf.ch>

Ross Boylan <RossBoylan@stanfordalumni.org>

Matt Kraai <kraai@debian.org>

Aaron M. Ucko <ucko@debian.org>

Jon Åslund <d98-jas@nada.kth.se>

Chapter 10

New versions of this tutorial

This manual was created by the Debian-BR (`http://www.debian-br.org`) project, with the goal of aiding everyday use of Debian.

New versions of this document will be made available in the Debian Documentation Project's page, at `http://www.debian.org/doc/ddp`.

Comments and criticisms can be sent to me directly by email at `<kov@debian.org>`.