Chapter

6

Maintenance and Updates: The APT Tools

Contents

What makes Debian so popular with administrators is how easily software can be installed and how easily the whole system can be updated. This unique advantage is largely due to the APT *program, that Falcot Corp administrators studied with enthusiasm.*

APT is the abbreviation for Advanced Package Tool. What makes this program "advanced" is its approach to packages. It doesn't simply evaluate them individually, but it considers them as a whole and produces the best possible combination of packages depending on what is available and compatible (according to dependencies).

VOCABULARY

Package source and source package

The word *source* can be ambiguous. A source package — a package containing the source code of a program — should not be confused with a package source — a repository (website, FTP server, CD-ROM, local directory, etc.) which contains packages.

APT needs to be given a "list of package sources": the file `/etc/apt/sources.list` will list the different repositories (or "sources") that publish Debian packages. APT will then import the list of packages published by each of these sources. This operation is achieved by downloading `Packages.xz` or a variant using a different compression method (such as `Packages.gz` or `.bz2`) files (in case of a source of binary packages) and `Sources.xz` or a variant (in case of a source of source packages) and by analyzing their contents. When an old copy of these files is already present, APT can update it by only downloading the differences (see sidebar "Incremental upgrade" page 110).

BACK TO BASICS

`gzip`, `bzip2`, `LZMA` and `XZ` Compression

A `.gz` extension refers to a file compressed with the `gzip` utility. `gzip` is the fast and efficient traditional Unix utility to compress files. Newer tools achieve better rates of compression but require more resources (computation time and memory) to compress and uncompress a file. Among them, and by order of appearance, there are `bzip2` (generating files with a `.bz2` extension), `lzma` (generating `.lzma` files) and `xz` (generating `.xz` files).

6.1. Filling in the `sources.list` File

6.1.1. Syntax

Each active line of the `/etc/apt/sources.list` file contains the description of a source, made of 3 parts separated by spaces.

The first field indicates the source type:

- "deb" for binary packages,
- "deb-src" for source packages.

The second field gives the base URL of the source (combined with the filenames present in the `Packages.gz` files, it must give a full and valid URL): this can consist in a Debian mirror or in any other package archive set up by a third party. The URL can start with file:// to indicate a local source installed in the system's file hierarchy, with http:// to indicate a source accessible from a web server, or with ftp:// for a source available on an FTP server. The URL can also start with cdrom: for CD-ROM/DVD-ROM/Blu-ray disc based installations, although this is less frequent, since network-based installation methods are more and more common.

The syntax of the last field depends on the structure of the repository. In the simplest cases, you can simply indicate a subdirectory (with a required trailing slash) of the desired source (this is often a simple "`./`" which refers to the absence of a subdirectory — the packages are then directly at the specified URL). But in the most common case, the repositories will be structured like a Debian mirror, with multiple distributions each having multiple components. In those cases, name the chosen distribution (by its "codename" — see the list in sidebar "Bruce Perens, a controversial leader" page 9 — or by the corresponding "suites" — stable, testing, unstable), then the components (or sections) to enable (chosen between main, contrib, and non-free in a typical Debian mirror).

VOCABULARY

The `main`, `contrib` and `non-free` archives

Debian uses three sections to differentiate packages according to the licenses chosen by the authors of each work. `Main` gathers all packages which fully comply with the Debian Free Software Guidelines.

The `non-free` archive is different because it contains software which does not (entirely) conform to these principles but which can nevertheless be distributed without restrictions. This archive, which is not officially part of Debian, is a service for users who could need some of those programs — however Debian always recommends giving priority to free software. The existence of this section represents a considerable problem for Richard M. Stallman and keeps the Free Software Foundation from recommending Debian to users.

`Contrib` (contributions) is a set of open source software which cannot function without some non-free elements. These elements can be software from the `non-free` section, or non-free files such as game ROMs, BIOS of consoles, etc. `Contrib` also includes free software whose compilation requires proprietary elements. This was initially the case for the OpenOffice.org office suite, which used to require a proprietary Java environment.

TIP

`/etc/apt/sources.list.d/*.list` files

If many package sources are referenced, it can be useful to split them in multiple files. Each part is then stored in `/etc/apt/sources.list.d/`*`filename`*`.list` (see sidebar "Directories ending in `.d`" page 111).

The cdrom entries describe the CD/DVD-ROMs you have. Contrary to other entries, a CD-ROM is not always available since it has to be inserted into the drive and since only one disc can be read at a time. For those reasons, these sources are managed in a slightly different way, and need to be added with the `apt-cdrom` program, usually executed with the `add` parameter. The latter will then request the disc to be inserted in the drive and will browse its contents looking for `Packages` files. It will use these files to update its database of available packages (this operation is usually done by the `apt update` command). From then on, APT can require the disc to be inserted if it needs one of its packages.

6.1.2. Repositories for *Stable* Users

Here is a standard `sources.list` for a system running the *Stable* version of Debian:

Example 6.1 */etc/apt/sources.list file for users of Debian Stable*

```
# Security updates
deb http://security.debian.org/ jessie/updates main contrib non-free
deb-src http://security.debian.org/ jessie/updates main contrib non-free

## Debian mirror

# Base repository
deb http://ftp.debian.org/debian jessie main contrib non-free
deb-src http://ftp.debian.org/debian jessie main contrib non-free

# Stable updates
deb http://ftp.debian.org/debian jessie-updates main contrib non-free
deb-src http://ftp.debian.org/debian jessie-updates main contrib non-free

# Stable backports
deb http://ftp.debian.org/debian jessie-backports main contrib non-free
deb-src http://ftp.debian.org/debian jessie-backports main contrib non-free
```

This file lists all sources of packages associated with the *Jessie* version of Debian (the current *Stable* as of this writing). We opted to name "jessie" explicitly instead of using the corresponding "stable" alias (stable, stable-updates, stable-backports) because we don't want to have the underlying distribution changed outside of our control when the next stable release comes out.

Most packages will come from the "base repository" which contains all packages but is seldom updated (about once every 2 months for a "point release"). The other repositories are partial (they do not contain all packages) and can host updates (packages with newer version) that APT might install. The following sections will explain the purpose and the rules governing each of those repositories.

Note that when the desired version of a package is available on several repositories, the first one listed in the `sources.list` file will be used. For this reason, non-official sources are usually added at the end of the file.

As a side note, most of what this section says about *Stable* applies equally well to *Oldstable* since the latter is just an older *Stable* that is maintained in parallel.

Security Updates

The security updates are not hosted on the usual network of Debian mirrors but on security.debian.org (on a small set of machines maintained by the Debian System Administrators). This archive contains security updates (prepared by the Debian Security Team and/or by package maintainers) for the *Stable* distribution.

The server can also host security updates for *Testing* but this doesn't happen very often since those updates tend to reach *Testing* via the regular flow of updates coming from *Unstable*.

Stable Updates

Stable updates are not security sensitive but are deemed important enough to be pushed to users before the next stable point release.

This repository will typically contain fixes for critical bugs which could not be fixed before release or which have been introduced by subsequent updates. Depending on the urgency, it can also contain updates for packages that have to evolve over time... like *spamassassin*'s spam detection rules, *clamav*'s virus database, or the daylight-saving time rules of all timezones (*tzdata*).

In practice, this repository is a subset of the proposed-updates repository, carefully selected by the Stable Release Managers.

Proposed Updates

Once published, the *Stable* distribution is only updated about once every 2 months. The proposed-updates repository is where the expected updates are prepared (under the supervision of the Stable Release Managers).

The security and stable updates documented in the former sections are always included in this repository, but there is more too, because package maintainers also have the opportunity to fix important bugs that do not deserve an immediate release.

Anyone can use this repository to test those updates before their official publication. The extract below uses the jessie-proposed-updates alias which is both more explicit and more consistent since wheezy-proposed-updates also exists (for the *Oldstable* updates):

```
deb http://ftp.debian.org/debian jessie-proposed-updates main contrib non-free
```

Stable Backports

The stable-backports repository hosts "package backports". The term refers to a package of some recent software which has been recompiled for an older distribution, generally for *Stable*.

When the distribution becomes a little dated, numerous software projects have released new versions that are not integrated into the current *Stable* (which is only modified to address the most critical problems, such as security problems). Since the *Testing* and *Unstable* distributions can be more risky, package maintainers sometimes offer recompilations of recent software applications for *Stable*, which has the advantage to limit potential instability to a small number of chosen packages.

➡ http://backports.debian.org

The stable-backports repository is now available on the usual Debian mirrors. But backports for *Squeeze* are still hosted on a dedicated server (backports.debian.org), and requires the following `sources.list` entry:

```
deb http://backports.debian.org/debian-backports squeeze-backports main contrib non-
       free
```

Backports from stable-backports are always created from packages available in *Testing*. This ensures that all installed backports will be upgradable to the corresponding stable version once the next stable release of Debian is available.

Even though this repository provides newer versions of packages, APT will not install them unless you give explicit instructions to do so (or unless you have already done so with a former version of the given backport):

```
$ sudo apt-get install package/jessie-backports
$ sudo apt-get install -t jessie-backports package
```

6.1.3. Repositories for *Testing*/*Unstable* Users

Here is a standard `sources.list` for a system running the *Testing* or *Unstable* version of Debian:

Example 6.2 */etc/apt/sources.list file for users of Debian* Testing/Unstable

```
# Unstable
deb http://ftp.debian.org/debian unstable main contrib non-free
deb-src http://ftp.debian.org/debian unstable main contrib non-free

# Testing
deb http://ftp.debian.org/debian testing main contrib non-free
deb-src http://ftp.debian.org/debian testing main contrib non-free

# Stable
deb http://ftp.debian.org/debian stable main contrib non-free
deb-src http://ftp.debian.org/debian stable main contrib non-free

# Security updates
deb http://security.debian.org/ stable/updates main contrib non-free
deb http://security.debian.org/ testing/updates main contrib non-free
deb-src http://security.debian.org/ stable/updates main contrib non-free
deb-src http://security.debian.org/ testing/updates main contrib non-free
```

With this `sources.list` file APT will install packages from *Unstable*. If that is not desired, use the APT::Default-Release setting (see section 6.2.3, "System Upgrade" page 109) to instruct APT to pick packages from another distribution (most likely *Testing* in this case).

There are good reasons to include all those repositories, even though a single one should be enough. *Testing* users will appreciate the possibility to cherry-pick a fixed package from *Unstable* when the version in *Testing* is affected by an annoying bug. On the opposite, *Unstable* users bitten by unexpected regressions have the possibility to downgrade packages to their (supposedly working) *Testing* version.

The inclusion of *Stable* is more debatable but it often gives access to some packages which have been removed from the development versions. It also ensures that you get the latest updates for packages which have not been modified since the last stable release.

The Experimental *Repository*

The archive of *Experimental* packages is present on all Debian mirrors, and contains packages which are not in the *Unstable* version yet because of their substandard quality — they are often software development versions or pre-versions (alpha, beta, release candidate...). A package can also be sent there after undergoing subsequent changes which can generate problems. The maintainer then tries to uncover them with help from advanced users who can handle important issues. After this first stage, the package is moved into *Unstable*, where it reaches a much larger audience and where it will be tested in much more detail.

Experimental is generally used by users who do not mind breaking their system and then repairing it. This distribution gives the possibility to import a package which a user wants to try or use as the need arises. That is exactly how Debian approaches it, since adding it in APT's `sources.list` file does not lead to the systematic use of its packages. The line to be added is:

```
deb http://ftp.debian.org/debian experimental main contrib non-free
```

6.1.4. Non-Official Resources: mentors.debian.net

There are numerous non-official sources of Debian packages set up by advanced users who have recompiled some software (Ubuntu made this popular with their Personal Package Archive service), by programmers who make their creation available to all, and even by Debian developers who offer pre-versions of their package online.

The mentors.debian.net site is interesting (although it only provides source packages), since it gathers packages created by candidates to the status of official Debian developer or by volunteers who wish to create Debian packages without going through that process of integration. These packages are made available without any guarantee regarding their quality; make sure that you check their origin and integrity and then test them before you consider using them in production.

Installing a package means giving root rights to its creator, because they decide on the contents of the initialization scripts which are run under that identity. Official Debian packages are created by volunteers who have been co-opted and reviewed and who can seal their packages so that their origin and integrity can be checked.

COMMUNITY

The `debian.net` sites

The *debian.net* domain is not an official resource of the Debian project. Each Debian developer may use that domain name for their own use. These websites can contain unofficial services (sometimes personal sites) hosted on a machine which does not belong to the project and set up by Debian developers, or even prototypes about to be moved on to *debian.org*. Two reasons can explain why some of these prototypes remain on *debian.net*: either no one has made the necessary effort to transform it into an official service (hosted on the *debian.org* domain, and with a certain guarantee of maintenance), or the service is too controversial to be officialized.

In general, be wary of a package whose origin you don't know and which isn't hosted on one of the official Debian servers: evaluate the degree to which you can trust the creator, and check the integrity of the package.

➨ `http://mentors.debian.net/`

GOING FURTHER

Old package versions: `snapshot.debian.org`

The `snapshot.debian.org` service, introduced in April 2010, can be used to "go backwards in time" and to find an old version of a package. It can be used for example to identify which version of a package introduced a regression, and more concretely, to come back to the former version while waiting for the regression fix.

6.1.5. Caching Proxy for Debian Packages

When an entire network of machines is configured to use the same remote server to download the same updated packages, any administrator knows that it would be beneficial to have an intermediate proxy acting as a network-local cache (see sidebar "Cache" page 116).

You can configure APT to use a "standard" proxy (see section 6.2.4, "Configuration Options" page 111 for the APT side, and section 11.6, "HTTP/FTP Proxy" page 282 for the proxy side), but the Debian ecosystem offers better options to solve this problem. The dedicated software presented in this section are smarter than a plain proxy cache because they can rely on the specific structure of APT repositories (for instance they know when individual files are obsolete or not, and thus adjust the time during which they are kept).

apt-cacher and *apt-cacher-ng* work like usual proxy cache servers. APT's `sources.list` is left unchanged, but APT is configured to use them as proxy for outgoing requests.

approx, on the other hand, acts like an HTTP server that "mirrors" any number of remote repositories in its top-level URLs. The mapping between those top-level directories and the remote URLs of the repositories is stored in `/etc/approx/approx.conf`:

```
# <name> <repository-base-url>
debian   http://ftp.debian.org/debian
security http://security.debian.org
```

approx runs by default on port 9999 via inetd (see section 9.6, "The `inetd` Super-Server" page 204) and requires the users to adjust their `sources.list` file to point to the approx server:

```
# Sample sources.list pointing to a local approx server
deb http://apt.falcot.com:9999/security jessie/updates main contrib non-free
deb http://apt.falcot.com:9999/debian jessie main contrib non-free
```

6.2. `aptitude`, `apt-get`, and `apt` Commands

APT is a vast project, whose original plans included a graphical interface. It is based on a library which contains the core application, and `apt-get` is the first front end — command-line based — which was developed within the project. `apt` is a second command-line based front end provided by APT which overcomes some design mistakes of `apt-get`.

Numerous other graphical interfaces then appeared as external projects: `synaptic`, `aptitude` (which includes both a text mode interface and a graphical one — even if not complete yet), `wajig`, etc. The most recommended interface, `apt`, is the one that we will use in the examples given in this section. Note however that `apt-get` and `aptitude` have a very similar command line syntax. When there are major differences between `apt`, `apt-get` and `aptitude`, these differences will be detailed.

6.2.1. Initialization

For any work with APT, the list of available packages needs to be updated; this can be done simply through `apt update`. Depending on the speed of your connection, the operation can take a while since it involves downloading a certain number of `Packages`/`Sources`/`Translation-`*`language-code`* files, which have gradually become bigger and bigger as Debian has developed (at least 10 MB of data for the `main` section). Of course, installing from a CD-ROM set does not require any downloading — in this case, the operation is very fast.

6.2.2. Installing and Removing

With APT, packages can be added or removed from the system, respectively with `apt install` *`package`* and `apt remove` *`package`*. In both cases, APT will automatically install the necessary dependencies or delete the packages which depend on the package that is being removed. The `apt purge` *`package`* command involves a complete uninstallation — the configuration files are also deleted.

TIP

Installing the same selection of packages several times

It can be useful to systematically install the same list of packages on several computers. This can be done quite easily.

First, retrieve the list of packages installed on the computer which will serve as the "model" to copy.

```
$ dpkg --get-selections >pkg-list
```

The `pkg-list` file then contains the list of installed packages. Next, transfer the `pkg-list` file onto the computers you want to update and use the following commands:

```
## Update dpkg's database of known packages
# avail=‘mktemp‘
# apt-cache dumpavail > "$avail"
# dpkg --merge-avail "$avail"
# rm -f "$avail"
## Update dpkg's selections
# dpkg --set-selections < pkg-list
## Ask apt-get to install the selected packages
# apt-get dselect-upgrade
```

The first commands records the list of available packages in the dpkg database, then `dpkg --set-selections` restores the selection of packages that you wish to install, and the `apt-get` invocation executes the required operations `aptitude` does not have this command.

TIP

Removing and installing at the same time

It is possible to ask `apt` (or `apt-get`, or `aptitude`) to install certain packages and remove others on the same command line by adding a suffix. With an `apt install` command, add "`-`" to the names of the packages you wish to remove. With an `apt remove` command, add "+" to the names of the packages you wish to install.

The next example shows two different ways to install *package1* and to remove *package2*.

```
# apt install package1 package2-
[...]
# apt remove package1+ package2
[...]
```

This can also be used to exclude packages which would otherwise be installed, for example due to a `Recommends`. In general, the dependency solver will use that information as a hint to look for alternative solutions.

TIP

`apt --reinstall` and `aptitude reinstall`

The system can sometimes be damaged after the removal or modification of files in a package. The easiest way to retrieve these files is to reinstall the affected package. Unfortunately, the packaging system finds that the latter is already installed and politely refuses to reinstall it; to avoid this, use the `--reinstall` option of the `apt` and `apt-get` commands. The following command reinstalls *postfix* even if it is already present:

```
# apt --reinstall install postfix
```

The `aptitude` command line is slightly different, but achieves the same result with `aptitude reinstall postfix`.

The problem does not arise with `dpkg`, but the administrator rarely uses it directly.

Be careful! Using `apt --reinstall` to restore packages modified during an attack will certainly not recover the system as it was. section 14.7, "Dealing with a Compromised Machine" page 413 details the necessary steps to take with a compromised system.

If the file `sources.list` mentions several distributions, it is possible to give the version of the package to install. A specific version number can be requested with `apt install` *`package=version`*, but indicating its distribution of origin (*Stable*, *Testing* or *Unstable*) — with `apt install` *`package/distribution`* — is usually preferred. With this command, it is possible to go back to an older version of a package (if for instance you know that it works well), provided that it is still available in one of the sources referenced by the `sources.list` file. Otherwise the snapshot.debian.org archive can come to the rescue (see sidebar "Old package versions: `snapshot.debian.org`" page 106).

Example 6.3 *Installation of the* unstable *version of* spamassassin

```
# apt install spamassassin/unstable
```

GOING FURTHER

The cache of `.deb` files

APT keeps a copy of each downloaded `.deb` file in the directory `/var/cache/apt/archives/`. In case of frequent updates, this directory can quickly take a lot of disk space with several versions of each package; you should regularly sort through them. Two commands can be used: `apt-get clean` entirely empties the directory; `apt-get autoclean` only removes packages which can no longer be downloaded (because they have disappeared from the Debian mirror) and are therefore clearly useless (the configuration parameter `APT::Clean-Installed` can prevent the removal of `.deb` files that are currently installed). Note that `apt` does not support those commands.

6.2.3. System Upgrade

Regular upgrades are recommended, because they include the latest security updates. To upgrade, use `apt upgrade`, `apt-get upgrade` or `aptitude safe-upgrade` (of course after `apt`

`update`). This command looks for installed packages which can be upgraded without removing any packages. In other words, the goal is to ensure the least intrusive upgrade possible. `apt-get` is slightly more demanding than `aptitude` or `apt` because it will refuse to install packages which were not installed beforehand.

TIP

Incremental upgrade

As we explained earlier, the aim of the `apt update` command is to download for each package source the corresponding `Packages` (or `Sources`) file. However, even after a `bzip2` compression, these files can remain rather large (the `Packages.xz` for the *main* section of *Jessie* takes more than 6 MB). If you wish to upgrade regularly, these downloads can take up a lot of time.

To speed up the process APT can download "diff" files containing the changes since the previous update, as opposed to the entire file. To achieve this, official Debian mirrors distribute different files which list the differences between one version of the `Packages` file and the following version. They are generated at each update of the archives and a history of one week is kept. Each of these "diff" files only takes a few dozen kilobytes for *Unstable*, so that the amount of data downloaded by a weekly `apt update` is often divided by 10. For distributions like *Stable* and *Testing*, which change less, the gain is even more noticeable.

However, it can sometimes be of interest to force the download of the entire `Packages` file, especially when the last upgrade is very old and when the mechanism of incremental differences would not contribute much. This can also be interesting when network access is very fast but when the processor of the machine to upgrade is rather slow, since the time saved on the download is more than lost when the computer calculates the new versions of these files (starting with the older versions and applying the downloaded differences). To do that, you can use the configuration parameter `Acquire::Pdiffs` and set it to `false`.

`apt` will generally select the most recent version number (except for packages from *Experimental* and *stable-backports*, which are ignored by default whatever their version number). If you specified *Testing* or *Unstable* in your `sources.list`, `apt upgrade` will switch most of your *Stable* system to *Testing* or *Unstable*, which might not be what you intended.

To tell `apt` to use a specific distribution when searching for upgraded packages, you need to use the -t or --target-release option, followed by the name of the distribution you want (for example: `apt -t stable upgrade`). To avoid specifying this option every time you use `apt`, you can add APT::Default-Release "stable"; in the file `/etc/apt/apt.conf.d/local`.

For more important upgrades, such as the change from one major Debian version to the next, you need to use `apt full-upgrade`. With this instruction, `apt` will complete the upgrade even if it has to remove some obsolete packages or install new dependencies. This is also the command used by users who work daily with the Debian *Unstable* release and follow its evolution day by day. It is so simple that it hardly needs explanation: APT's reputation is based on this great functionality.

Unlike `apt` and `aptitude`, `apt-get` doesn't know the `full-upgrade` command. Instead, you should use `apt-get dist-upgrade` ("distribution upgrade"), the historical and well-known command that `apt` and `aptitude` also accept for the convenience of users who got used to it.

6.2.4. Configuration Options

Besides the configuration elements already mentioned, it is possible to configure certain aspects of APT by adding directives in a file of the `/etc/apt/apt.conf.d/` directory. Remember for instance that it is possible for APT to tell `dpkg` to ignore file conflict errors by specifying DPkg:: options { "--force-overwrite";}.

If the Web can only be accessed through a proxy, add a line like Acquire::http::proxy "http://*yourproxy*:3128". For an FTP proxy, write Acquire::ftp::proxy "ftp://*yourproxy*". To discover more configuration options, read the `apt.conf(5)` manual page with the `man apt.conf` command (for details on manual pages, see section 7.1.1, "Manual Pages" page 134).

BACK TO BASICS

Directories ending in `.d`

Directories with a `.d` suffix are used more and more often. Each directory represents a configuration file which is split over multiple files. In this sense, all of the files in `/etc/apt/apt.conf.d/` are instructions for the configuration of APT. APT includes them in alphabetical order, so that the last ones can modify a configuration element defined in one of the first ones.

This structure brings some flexibility to the machine administrator and to the package maintainers. Indeed, the administrator can easily modify the configuration of the software by adding a ready-made file in the directory in question without having to change an existing file. Package maintainers use the same approach when they need to adapt the configuration of another software to ensure that it perfectly co-exists with theirs. The Debian policy explicitly forbids modifying configuration files of other packages — only users are allowed to do this. Remember that during a package upgrade, the user gets to choose the version of the configuration file that should be kept when a modification has been detected. Any external modification of the file would trigger that request, which would disturb the administrator, who is sure not to have changed anything.

Without a `.d` directory, it is impossible for an external package to change the settings of a program without modifying its configuration file. Instead it must invite the user to do it themselves and lists the operations to be done in the file `/usr/share/doc/`*`package`*`/README.Debian`.

Depending on the application, the `.d` directory is used directly or managed by an external script which will concatenate all the files to create the configuration file itself. It is important to execute the script after any change in that directory so that the most recent modifications are taken into account. In the same way, it is important not to work directly in the configuration file created automatically, since everything would be lost at the next execution of the script. The chosen method (`.d` directory used directly or a file generated from that directory) is usually dictated by implementation constraints, but in both cases the gains in terms of configuration flexibility more than make up for the small complications that they entail. The Exim 4 mail server is an example of the generated file method: it can be configured through several files (`/etc/exim4/conf.d/*`) which are concatenated into `/var/lib/exim4/config.autogenerated` by the `update-exim4.conf` command.

6.2.5. Managing Package Priorities

One of the most important aspects in the configuration of APT is the management of the priorities associated with each package source. For instance, you might want to extend one distribution with one or two newer packages from *Testing*, *Unstable* or *Experimental*. It is possible to assign a priority to each available package (the same package can have several priorities depending on its version or the distribution providing it). These priorities will influence APT's behavior: for each package, it will always select the version with the highest priority (except if this version is older than the installed one and if its priority is less than 1000).

APT defines several default priorities. Each installed package version has a priority of 100. A non-installed version has a priority of 500 by default, but it can jump to 990 if it is part of the target release (defined with the -t command-line option or the APT::Default-Release configuration directive).

You can modify the priorities by adding entries in the `/etc/apt/preferences` file with the names of the affected packages, their version, their origin and their new priority.

APT will never install an older version of a package (that is, a package whose version number is lower than the one of the currently installed package) except if its priority is higher than 1000. APT will always install the highest priority package which follows this constraint. If two packages have the same priority, APT installs the newest one (whose version number is the highest). If two packages of same version have the same priority but differ in their content, APT installs the version that is not installed (this rule has been created to cover the case of a package update without the increment of the revision number, which is usually required).

In more concrete terms, a package whose priority is less than 0 will never be installed. A package with a priority ranging between 0 and 100 will only be installed if no other version of the package is already installed. With a priority between 100 and 500, the package will only be installed if there is no other newer version installed or available in another distribution. A package of priority between 501 and 990 will only be installed if there is no newer version installed or available in the target distribution. With a priority between 990 and 1000, the package will be installed except if the installed version is newer. A priority greater than 1000 will always lead to the installation of the package even if it forces APT to downgrade to an older version.

When APT checks `/etc/apt/preferences`, it first takes into account the most specific entries (often those specifying the concerned package), then the more generic ones (including for example all the packages of a distribution). If several generic entries exist, the first match is used. The available selection criteria include the package's name and the source providing it. Every package source is identified by the information contained in a `Release` file that APT downloads together with the `Packages` files. It specifies the origin (usually "Debian" for the packages of official mirrors, but it can also be a person's or an organization's name for third-party repositories). It also gives the name of the distribution (usually *Stable*, *Testing*, *Unstable* or *Experimental* for the standard distributions provided by Debian) together with its version (for example 8 for Debian *Jessie*). Let's have a look at its syntax through some realistic case studies of this mechanism.

SPECIFIC CASE

Priority of *experimental*

If you listed *Experimental* in your `sources.list` file, the corresponding packages will almost never be installed because their default APT priority is 1. This is of course a specific case, designed to keep users from installing *Experimental* packages by mistake. The packages can only be installed by typing `aptitude install package/experimental` — users typing this command can only be aware of the risks that they take. It is still possible (though *not* recommended) to treat packages of *Experimental* like those of other distributions by giving them a priority of 500. This is done with a specific entry in `/etc/apt/preferences`:

```
Package: *
Pin: release a=experimental
Pin-Priority: 500
```

Let's suppose that you only want to use packages from the stable version of Debian. Those provided in other versions should not be installed except if explicitly requested. You could write the following entries in the `/etc/apt/preferences` file:

```
Package: *
Pin: release a=stable
Pin-Priority: 900

Package: *
Pin: release o=Debian
Pin-Priority: -10
```

a=stable defines the name of the selected distribution. o=Debian limits the scope to packages whose origin is "Debian".

Let's now assume that you have a server with several local programs depending on the version 5.14 of Perl and that you want to ensure that upgrades will not install another version of it. You could use this entry:

```
Package: perl
Pin: version 5.14*
Pin-Priority: 1001
```

The reference documentation for this configuration file is available in the manual page `apt_preferences(5)`, which you can display with `man apt_preferences`.

TIP

Comments in `/etc/apt/preferences`

There is no official syntax to put comments in the `/etc/apt/preferences` file, but some textual descriptions can be provided by putting one or more "`Explanation`" fields at the start of each entry:

```
Explanation: The package xserver-xorg-video-intel provided
Explanation: in experimental can be used safely
Package: xserver-xorg-video-intel
Pin: release a=experimental
Pin-Priority: 500
```

`apt` being such a marvelous tool, it is tempting to pick packages coming from other distributions. For example, after having installed a *Stable* system, you might want to try out a software package available in *Testing* or *Unstable* without diverging too much from the system's initial state.

Even if you will occasionally encounter problems while mixing packages from different distributions, `apt` manages such coexistence very well and limits risks very effectively. The best way to proceed is to list all distributions used in `/etc/apt/sources.list` (some people always put the three distributions, but remember that *Unstable* is reserved for experienced users) and to define your reference distribution with the APT::Default-Release parameter (see section 6.2.3, "System Upgrade" page 109).

Let's suppose that *Stable* is your reference distribution but that *Testing* and *Unstable* are also listed in your `sources.list` file. In this case, you can use `apt install package/testing` to install a package from *Testing*. If the installation fails due to some unsatisfiable dependencies, let it solve those dependencies within *Testing* by adding the -t testing parameter. The same obviously applies to *Unstable*.

In this situation, upgrades (`upgrade` and `full-upgrade`) are done within *Stable* except for packages already upgraded to another distribution: those will follow updates available in the other distributions. We will explain this behavior with the help of the default priorities set by APT below. Do not hesitate to use `apt-cache policy` (see sidebar "apt-cache policy" page 114) to verify the given priorities.

Everything centers around the fact that APT only considers packages of higher or equal version than the installed one (assuming that `/etc/apt/preferences` has not been used to force priorities higher than 1000 for some packages).

TIP — **apt-cache policy**

To gain a better understanding of the mechanism of priority, do not hesitate to execute `apt-cache policy` to display the default priority associated with each package source. You can also use `apt-cache policy` *package* to display the priorities of all available versions of a given package.

Let's assume that you have installed version 1 of a first package from *Stable* and that version 2 and 3 are available respectively in *Testing* and *Unstable*. The installed version has a priority of 100 but the version available in *Stable* (the very same) has a priority of 990 (because it is part of the target release). Packages in *Testing* and *Unstable* have a priority of 500 (the default priority of a non-installed version). The winner is thus version 1 with a priority of 990. The package "stays in *Stable*".

Let's take the example of another package whose version 2 has been installed from *Testing*. Version 1 is available in *Stable* and version 3 in *Unstable*. Version 1 (of priority 990 — thus lower than 1000) is discarded because it is lower than the installed version. This only leaves version 2 and 3, both of priority 500. Faced with this alternative, APT selects the newest version, the one from *Unstable*.If you don't want a package installed from *Testing* to migrate to *Unstable*, you have

to assign a priority lower than 500 (490 for example) to packages coming from *Unstable*. You can modify `/etc/apt/preferences` to this effect:

```
Package: *
Pin: release a=unstable
Pin-Priority: 490
```

6.2.7. Tracking Automatically Installed Packages

One of the essential functionalities of `apt` is the tracking of packages installed only through dependencies. These packages are called "automatic", and often include libraries for instance.

With this information, when packages are removed, the package managers can compute a list of automatic packages that are no longer needed (because there is no "manually installed" packages depending on them). `apt-get autoremove` will get rid of those packages. `aptitude` and `apt` do not have this command: the former because it removes them automatically as soon as they are identified, and the latter probably because the user should not have to manually run such a command. In all cases, the tools display a clear message listing the affected packages.

It is a good habit to mark as automatic any package that you don't need directly so that they are automatically removed when they aren't necessary anymore. `apt-mark auto` *`package`* will mark the given package as automatic whereas `apt-mark manual` *`package`* does the opposite. `aptitude markauto` and `aptitude unmarkauto` work in the same way although they offer more features for marking many packages at once (see section 6.4.1, "`aptitude`" page 117). The console-based interactive interface of `aptitude` also makes it easy to review the "automatic flag" on many packages.

People might want to know why an automatically installed package is present on the system. To get this information from the command line, you can use `aptitude why` *`package`* (`apt` and `apt-get` have no similar feature):

```
$ aptitude why python-debian
i   aptitude         Recommends apt-xapian-index
i A apt-xapian-index Depends    python-debian (>= 0.1.15)
```

ALTERNATIVE

`deborphan` and `debfoster`

In days where `apt`, `apt-get` and `aptitude` were not able to track automatic packages, there were two utilities producing lists of unnecessary packages: `deborphan` and `debfoster`.

`deborphan` is the most rudimentary of both. It simply scans the `libs` and `oldlibs` sections (in the absence of supplementary instructions) looking for the packages that are currently installed and that no other package depends on. The resulting list can then serve as a basis to remove unneeded packages.

`debfoster` has a more elaborate approach, very similar to APT's one: it maintains a list of packages that have been explicitly installed, and remembers what packages are really required between each invocation. If new packages appear on the system and if `debfoster` doesn't know them as required packages, they will be shown on the screen together with a list of their dependencies. The program then offers a

choice: remove the package (possibly together with those that depend on it), mark it as explicitly required, or ignore it temporarily.

6.3. The `apt-cache` Command

The `apt-cache` command can display much of the information stored in APT's internal database. This information is a sort of cache since it is gathered from the different sources listed in the `sources.list` file. This happens during the `apt update` operation.

VOCABULARY

Cache

A cache is a temporary storage system used to speed up frequent data access when the usual access method is expensive (performance-wise). This concept can be applied in numerous situations and at different scales, from the core of microprocessors up to high-end storage systems.

In the case of APT, the reference `Packages` files are those located on Debian mirrors. That said, it would be very ineffective to go through the network for every search that we might want to do in the database of available packages. That is why APT stores a copy of those files (in `/var/lib/apt/lists/`) and searches are done within those local files. Similarly, `/var/cache/apt/archives/` contains a cache of already downloaded packages to avoid downloading them again if you need to reinstall them after a removal.

The `apt-cache` command can do keyword-based package searches with `apt-cache search` *`keyword`*. It can also display the headers of the package's available versions with `apt-cache show` *`package`*. This command provides the package's description, its dependencies, the name of its maintainer, etc. Note that `apt search`, `apt show`, `aptitude search`, `aptitude show` work in the same way.

ALTERNATIVE

`axi-cache`

`apt-cache search` is a very rudimentary tool, basically implementing `grep` on package's descriptions. It often returns too many results or none at all when you include too many keywords.

`axi-cache search` *`term`*, on the other hand, provides better results, sorted by relevancy. It uses the *Xapian* search engine and is part of the *apt-xapian-index* package whichs indexes all package information (and more, like the `.desktop` files from all Debian packages). It knows about tags (see sidebar "The Tag field" page 80) and returns results in a matter of milliseconds.

```
$ axi-cache search package use::searching
105 results found.
Results 1-20:
100% packagesearch - GUI for searching packages and viewing
       package information
98% debtags - Enables support for package tags
94% debian-goodies - Small toolbox-style utilities
93% dpkg-awk - Gawk script to parse /var/lib/dpkg/{status,
      available} and Packages
```

```
93% goplay - games (and more) package browser using DebTags
[...]
87% apt-xapian-index - maintenance and search tools for a
      Xapian index of Debian packages
[...]
More terms: search debian searching strigi debtags bsearch
      libbsearch
More tags: suite::debian works-with::software:package role
      ::program interface::commandline implemented-in::c++
      admin::package-management use::analysing
'axi-cache more' will give more results
```

Some features are more rarely used. For instance, `apt-cache policy` displays the priorities of package sources as well as those of individual packages. Another example is `apt-cache dumpavail` which displays the headers of all available versions of all packages. `apt-cache pkgnames` displays the list of all the packages which appear at least once in the cache.

6.4. Frontends: `aptitude`, `synaptic`

APT is a C++ program whose code mainly resides in the `libapt-pkg` shared library. Using a shared library facilitates the creation of user interfaces (front-ends), since the code contained in the library can easily be reused. Historically, `apt-get` was only designed as a test front-end for `libapt-pkg` but its success tends to obscure this fact.

6.4.1. `aptitude`

`aptitude` is an interactive program that can be used in semi-graphical mode on the console. You can browse the list of installed and available packages, look up all the available information, and select packages to install or remove. The program is designed specifically to be used by administrators, so that its default behaviors are much more intelligent than `apt-get`'s, and its interface much easier to understand.

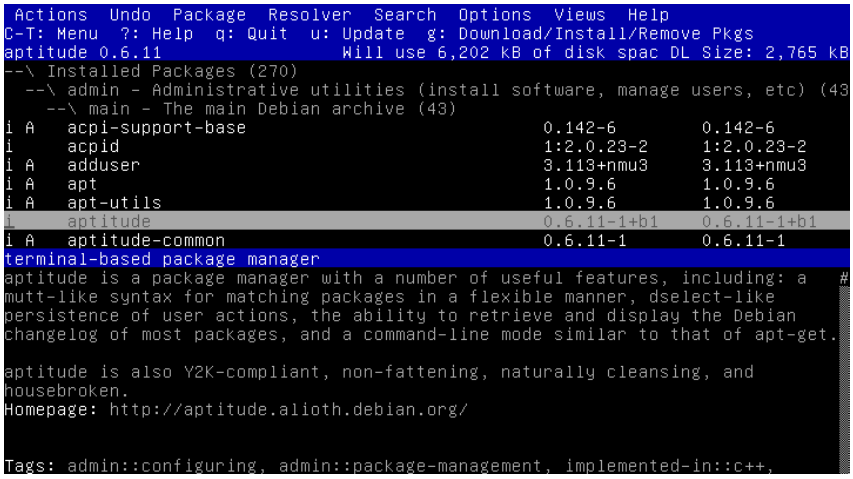

Figure 6.1 *The `aptitude` package manager*

When it starts, `aptitude` shows a list of packages sorted by state (installed, non-installed, or installed but not available on the mirrors — other sections display tasks, virtual packages, and new packages that appeared recently on mirrors). To facilitate thematic browsing, other views are available. In all cases, `aptitude` displays a list combining categories and packages on the screen. Categories are organized through a tree structure, whose branches can respectively be unfolded or closed with the Enter, [and] keys. + should be used to mark a package for installation, - to mark it for removal and _ to purge it (note than these keys can also be used for categories, in which case the corresponding actions will be applied to all the packages of the category). u updates the lists of available packages and Shift+u prepares a global system upgrade. g switches to a summary view of the requested changes (and typing g again will apply the changes), and q quits the current view. If you are in the initial view, this will effectively close `aptitude`.

DOCUMENTATION

`aptitude`

This section does not cover the finer details of using `aptitude`, it rather focuses on giving you a survival kit to use it. `aptitude` is rather well documented and we advise you to use its complete manual available in the *aptitude-doc-en* package.

➨ `file:///usr/share/doc/aptitude/html/en/index.html`

To search for a package, you can type / followed by a search pattern. This pattern matches the name of the package, but can also be applied to the description (if preceded by ~d), to the section (with ~s) or to other characteristics detailed in the documentation. The same patterns can filter the list of displayed packages: type the l key (as in *limit*) and enter the pattern.

Managing the "automatic flag" of Debian packages (see section 6.2.7, "Tracking Automatically Installed Packages" page 115) is a breeze with `aptitude`. It is possible to browse the list of installed packages and mark packages as automatic with Shift+m or to remove the mark with the m key. "Automatic packages" are displayed with an "A" in the list of packages. This feature also offers a simple way to visualize the packages in use on a machine, without all the libraries

and dependencies that you don't really care about. The related pattern that can be used with l (to activate the filter mode) is ~i⊠M. It specifies that you only want to see installed packages (~i) not marked as automatic (⊠M).

TOOL

Using aptitude on the command-line interface

Most of `aptitude`'s features are accessible via the interactive interface as well as via command-lines. These command-lines will seem familiar to regular users of `apt-get` and `apt-cache`.

The advanced features of `aptitude` are also available on the command-line. You can use the same package search patterns as in the interactive version. For example, if you want to cleanup the list of "manually installed" packages, and if you know that none of the locally installed programs require any particular libraries or Perl modules, you can mark the corresponding packages as automatic with a single command:

```
# aptitude markauto '~slibs|~sperl'
```

Here, you can clearly see the power of the search pattern system of `aptitude`, which enables the instant selection of all the packages in the `libs` and `perl` sections.

Beware, if some packages are marked as automatic and if no other package depends on them, they will be removed immediately (after a confirmation request).

Managing Recommendations, Suggestions and Tasks

Another interesting feature of `aptitude` is the fact that it respects recommendations between packages while still giving users the choice not to install them on a case by case basis. For example, the *gnome* package recommends *gdebi* (among others). When you select the former for installation, the latter will also be selected (and marked as automatic if not already installed on the system). Typing g will make it obvious: *gdebi* appears on the summary screen of pending actions in the list of packages installed automatically to satisfy dependencies. However, you can decide not to install it by deselecting it before confirming the operations.

Note that this recommendation tracking feature does not apply to upgrades. For instance, if a new version of *gnome* recommends a package that it did not recommend formerly, the package won't be marked for installation. However, it will be listed on the upgrade screen so that the administrator can still select it for installation.

Suggestions between packages are also taken into account, but in a manner adapted to their specific status. For example, since *gnome* suggests *dia-gnome*, the latter will be displayed on the summary screen of pending actions (in the section of packages suggested by other packages). This way, it is visible and the administrator can decide whether to take the suggestion into account or not. Since it is only a suggestion and not a dependency or a recommendation, the package will not be selected automatically — its selection requires a manual intervention from the user (thus, the package will not be marked as automatic).

In the same spirit, remember that `aptitude` makes intelligent use of the concept of task. Since tasks are displayed as categories in the screens of packages lists, you can either select a full task

for installation or removal, or browse the list of packages included in the task to select a smaller subset.

Better Solver Algorithms

To conclude this section, let's note that `aptitude` has more elaborate algorithms compared to `apt-get` when it comes to resolving difficult situations. When a set of actions is requested and when these combined actions would lead to an incoherent system, `aptitude` evaluates several possible scenarios and presents them in order of decreasing relevance. However, these algorithms are not failproof. Fortunately there is always the possibility to manually select the actions to perform. When the currently selected actions lead to contradictions, the upper part of the screen indicates a number of "broken" packages (and you can directly navigate to those packages by pressing b). It is then possible to manually build a solution for the problems found. In particular, you can get access to the different available versions by simply selecting the package with Enter. If the selection of one of these versions solves the problem, you should not hesitate to use the function. When the number of broken packages gets down to zero, you can safely go to the summary screen of pending actions for a last check before you apply them.

NOTE

`aptitude`'s log

Like `dpkg`, `aptitude` keeps a trace of executed actions in its logfile (`/var/log/aptitude`). However, since both commands work at a very different level, you cannot find the same information in their respective logfiles. While `dpkg` logs all the operations executed on individual packages step by step, `aptitude` gives a broader view of high-level operations like a system-wide upgrade.

Beware, this logfile only contains a summary of operations performed by `aptitude`. If other front-ends (or even `dpkg` itself) are occasionally used, then `aptitude`'s log will only contain a partial view of the operations, so you can't rely on it to build a trustworthy history of the system.

6.4.2. `synaptic`

`synaptic` is a graphical package manager for Debian which features a clean and efficient graphical interface based on GTK+/GNOME. Its many ready-to-use filters give fast access to newly available packages, installed packages, upgradable packages, obsolete packages and so on. If you browse through these lists, you can select the operations to be done on the packages (install, upgrade, remove, purge); these operations are not performed immediately, but put into a task list. A single click on a button then validates the operations, and they are performed in one go.

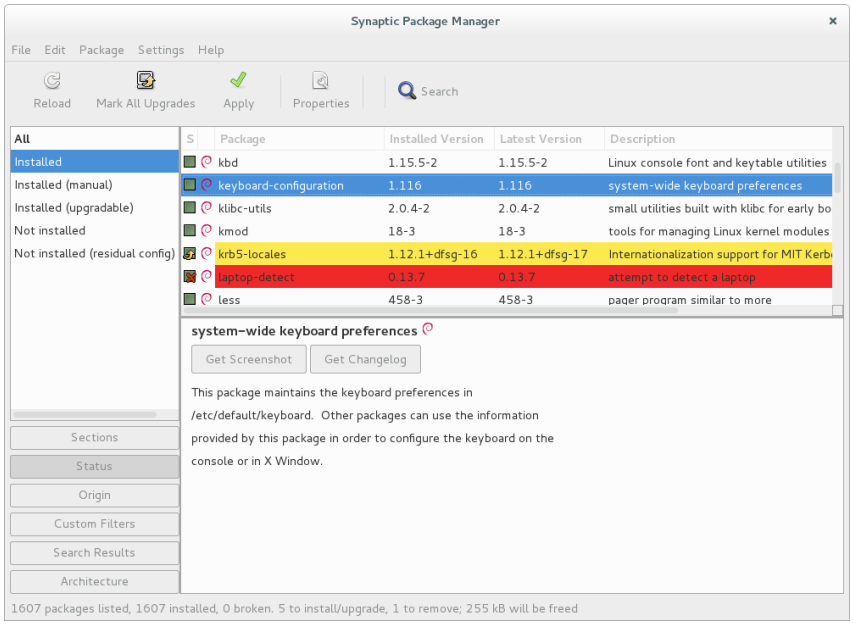

Figure 6.2 *synaptic package manager*

6.5. Checking Package Authenticity

Security is very important for Falcot Corp administrators. Accordingly, they need to ensure that they only install packages which are guaranteed to come from Debian with no tampering on the way. A computer cracker could try to add malicious code to an otherwise legitimate package. Such a package, if installed, could do anything the cracker designed it to do, including for instance disclosing passwords or confidential information. To circumvent this risk, Debian provides a tamper-proof seal to guarantee — at install time — that a package really comes from its official maintainer and hasn't been modified by a third party.

The seal works with a chain of cryptographical hashes and a signature. The signed file is the `Release` file, provided by the Debian mirrors. It contains a list of the `Packages` files (including their compressed forms, `Packages.gz` and `Packages.xz`, and the incremental versions), along with their MD5, SHA1 and SHA256 hashes, which ensures that the files haven't been tampered with. These `Packages` files contain a list of the Debian packages available on the mirror, along with their hashes, which ensures in turn that the contents of the packages themselves haven't been altered either.

The trusted keys are managed with the `apt-key` command found in the *apt* package. This program maintains a keyring of GnuPG public keys, which are used to verify signatures in the `Release.gpg` files available on the mirrors. It can be used to add new keys manually (when non-official mirrors are needed). Generally however, only the official Debian keys are needed. These keys are automatically kept up-to-date by the *debian-archive-keyring* package (which puts

the corresponding keyrings in `/etc/apt/trusted.gpg.d`). However, the first installation of this particular package requires caution: even if the package is signed like any other, the signature cannot be verified externally. Cautious administrators should therefore check the fingerprints of imported keys before trusting them to install new packages:

```
# apt-key fingerprint
/etc/apt/trusted.gpg.d/debian-archive-jessie-automatic.gpg
-----------------------------------------------------------
pub   4096R/2B90D010 2014-11-21 [expires: 2022-11-19]
      Key fingerprint = 126C 0D24 BD8A 2942 CC7D  F8AC 7638 D044 2B90 D010
uid                  Debian Archive Automatic Signing Key (8/jessie) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-jessie-security-automatic.gpg
-------------------------------------------------------------------
pub   4096R/C857C906 2014-11-21 [expires: 2022-11-19]
      Key fingerprint = D211 6914 1CEC D440 F2EB  8DDA 9D6D 8F6B C857 C906
uid                  Debian Security Archive Automatic Signing Key (8/jessie) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-jessie-stable.gpg
-------------------------------------------------------
pub   4096R/518E17E1 2013-08-17 [expires: 2021-08-15]
      Key fingerprint = 75DD C3C4 A499 F1A1 8CB5  F3C8 CBF8 D6FD 518E 17E1
uid                  Jessie Stable Release Key <debian-release@lists.debian.org>

/etc/apt/trusted.gpg.d/debian-archive-squeeze-automatic.gpg
----------------------------------------------------------
pub   4096R/473041FA 2010-08-27 [expires: 2018-03-05]
      Key fingerprint = 9FED 2BCB DCD2 9CDF 7626  78CB AED4 B06F 4730 41FA
uid                  Debian Archive Automatic Signing Key (6.0/squeeze) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-squeeze-stable.gpg
--------------------------------------------------------
pub   4096R/B98321F9 2010-08-07 [expires: 2017-08-05]
      Key fingerprint = 0E4E DE2C 7F3E 1FC0 D033  800E 6448 1591 B983 21F9
uid                  Squeeze Stable Release Key <debian-release@lists.debian.org>

/etc/apt/trusted.gpg.d/debian-archive-wheezy-automatic.gpg
---------------------------------------------------------
pub   4096R/46925553 2012-04-27 [expires: 2020-04-25]
      Key fingerprint = A1BD 8E9D 78F7 FE5C 3E65  D8AF 8B48 AD62 4692 5553
uid                  Debian Archive Automatic Signing Key (7.0/wheezy) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-wheezy-stable.gpg
-------------------------------------------------------
pub   4096R/65FFB764 2012-05-08 [expires: 2019-05-07]
      Key fingerprint = ED6D 6527 1AAC F0FF 15D1  2303 6FB2 A1C2 65FF B764
uid                  Wheezy Stable Release Key <debian-release@lists.debian.org>
```

IN PRACTICE

Adding trusted keys

When a third-party package source is added to the `sources.list` file, APT needs to be told to trust the corresponding GPG authentication key (otherwise it will keep complaining that it can't ensure the authenticity of the packages coming from that repository). The first step is of course to get the public key. More often than not, the key will be provided as a small text file, which we will call `key.asc` in the following examples.

To add the key to the trusted keyring, the administrator can run `apt-key add < key.asc`. Another way is to use the `synaptic` graphical interface: its "Authentication" tab in the Settings Repositories menu gives the possibility of importing a key from the `key.asc` file.

For people who would want a dedicated application and more details on the trusted keys, it is possible to use `gui-apt-key` (in the package of the same name), a small graphical user interface which manages the trusted keyring.

Once the appropriate keys are in the keyring, APT will check the signatures before any risky operation, so that front-ends will display a warning if asked to install a package whose authenticity can't be ascertained.

6.6. Upgrading from One Stable Distribution to the Next

One of the best-known features of Debian is its ability to upgrade an installed system from one stable release to the next: *dist-upgrade* — a well-known phrase — has largely contributed to the project's reputation. With a few precautions, upgrading a computer can take as little as a few minutes, or a few dozen minutes, depending on the download speed from the package repositories.

6.6.1. Recommended Procedure

Since Debian has quite some time to evolve in-between stable releases, you should read the release notes before upgrading.

BACK TO BASICS

Release notes

The release notes for an operating system (and, more generally, for any software) are a document giving an overview of the software, with some details concerning the particularities of one version. These documents are generally short compared to the complete documentation, and they usually list the features which have been introduced since the previous version. They also give details on upgrading procedures, warnings for users of previous versions, and sometimes errata.

Release notes are available online: the release notes for the current stable release have a dedicated URL, while older release notes can be found with their codenames:

- `http://www.debian.org/releases/stable/releasenotes`
- `http://www.debian.org/releases/wheezy/releasenotes`

In this section, we will focus on upgrading a *Wheezy* system to *Jessie*. This is a major operation on a system; as such, it is never 100% risk-free, and should not be attempted before all important data has been backed up.

Another good habit which makes the upgrade easier (and shorter) is to tidy your installed packages and keep only the ones that are really needed. Helpful tools to do that include `aptitude`, `deborphan` and `debfoster` (see section 6.2.7, "Tracking Automatically Installed Packages" page 115). For example, you can use the following command, and then use `aptitude`'s interactive mode to double check and fine-tune the scheduled removals:

```
# deborphan | xargs aptitude --schedule-only remove
```

Now for the upgrading itself. First, you need to change the `/etc/apt/sources.list` file to tell APT to get its packages from *Jessie* instead of *Wheezy*. If the file only contains references to *Stable* rather than explicit codenames, the change isn't even required, since *Stable* always refers

to the latest released version of Debian. In both cases, the database of available packages must be refreshed (with the `apt update` command or the refresh button in `synaptic`).

Once these new package sources are registered, you should first do a minimal upgrade with `apt upgrade`. By doing the upgrade in two steps, we ease the job of the package management tools and often ensure that we have the latest versions of those, which might have accumulated bugfixes and improvements required to complete the full distribution upgrade.

Once this first upgrade is done, it is time to handle the upgrade itself, either with `apt full-upgrade`, `aptitude`, or `synaptic`. You should carefully check the suggested actions before applying them: you might want to add suggested packages or deselect packages which are only recommended and known not to be useful. In any case, the front-end should come up with a scenario ending in a coherent and up-to-date *Jessie* system. Then, all you need is to do is wait while the required packages are downloaded, answer the Debconf questions and possibly those about locally modified configuration files, and sit back while APT does its magic.

6.6.2. Handling Problems after an Upgrade

In spite of the Debian maintainers' best efforts, a major system upgrade isn't always as smooth as you could wish. New software versions may be incompatible with previous ones (for instance, their default behavior or their data format may have changed). Also, some bugs may slip through the cracks despite the testing phase which always precedes a Debian release.

To anticipate some of these problems, you can install the *apt-listchanges* package, which displays information about possible problems at the beginning of a package upgrade. This information is compiled by the package maintainers and put in `/usr/share/doc/`*package*`/NEWS.Debian` files for the benefit of users. Reading these files (possibly through *apt-listchanges*) should help you avoid bad surprises.

You might sometimes find that the new version of a software doesn't work at all. This generally happens if the application isn't particularly popular and hasn't been tested enough; a last-minute update can also introduce regressions which are only found after the stable release. In both cases, the first thing to do is to have a look at the bug tracking system at https://bugs.debian.org/*package*, and check whether the problem has already been reported. If it hasn't, you should report it yourself with `reportbug`. If it is already known, the bug report and the associated messages are usually an excellent source of information related to the bug:

- sometimes a patch already exists, and it is available on the bug report; you can then recompile a fixed version of the broken package locally (see section 15.1, "Rebuilding a Package from its Sources" page 420);
- in other cases, users may have found a workaround for the problem and shared their insights about it in their replies to the report;
- in yet other cases, a fixed package may have already been prepared and made public by the maintainer.

Depending on the severity of the bug, a new version of the package may be prepared specifically for a new revision of the stable release. When this happens, the fixed package is made available in the proposed-updates section of the Debian mirrors (see section 6.1.2.3, "Proposed Updates" page 103). The corresponding entry can then be temporarily added to the `sources.list` file, and updated packages can be installed with `apt` or `aptitude`.

Sometimes the fixed package isn't available in this section yet because it is pending a validation by the Stable Release Managers. You can verify if that's the case on their web page. Packages listed there aren't available yet, but at least you know that the publication process is ongoing.

➠ `https://release.debian.org/proposed-updates/stable.html`

6.7. Keeping a System Up to Date

The Debian distribution is dynamic and changes continually. Most of the changes are in the *Testing* and *Unstable* versions, but even *Stable* is updated from time to time, mostly for security-related fixes. Whatever version of Debian a system runs, it is generally a good idea to keep it up to date, so that you can get the benefit of recent evolutions and bug fixes.

While it is of course possible to periodically run a tool to check for available updates and run the upgrades, such a repetitive task is tedious, especially when it needs to be performed on several machines. Fortunately, like many repetitive tasks, it can be partly automated, and a set of tools have already been developed to that effect.

The first of these tools is `apticron`, in the package of the same name. Its main effect is to run a script daily (via `cron`). The script updates the list of available packages, and, if some installed packages are not in the latest available version, it sends an email with a list of these packages along with the changes that have been made in the new versions. Obviously, this package mostly targets users of Debian *Stable*, since the daily emails would be very long for the faster paced versions of Debian. When updates are available, `apticron` automatically downloads them. It does not install them — the administrator will still do it — but having the packages already downloaded and available locally (in APT's cache) makes the job faster.

Administrators in charge of several computers will no doubt appreciate being informed of pending upgrades, but the upgrades themselves are still as tedious as they used to be, which is where the `/etc/cron.daily/apt` script (in the *apt* package) comes in handy. This script is also run daily (and non-interactively) by `cron`. To control its behavior, use APT configuration variables (which are therefore stored in a file under `/etc/apt/apt.conf.d/`). The main variables are:

APT::Periodic::Update-Package-Lists This option allows you to specify the frequency (in days) at which the package lists are refreshed. `apticron` users can do without this variable, since `apticron` already does this task.

APT::Periodic::Download-Upgradeable-Packages Again, this option indicates a frequency (in days), this time for the downloading of the actual packages. Again, `apticron` users won't need it.

APT::Periodic::AutocleanInterval This option covers a feature that `apticron` doesn't have. It controls how often obsolete packages (those not referenced by any distribution anymore) are removed from the APT cache. This keeps the APT cache at a reasonable size and means that you don't need to worry about that task.

APT::Periodic::Unattended-Upgrade When this option is enabled, the daily script will execute `unattended-upgrade` (from the *unattended-upgrades* package) which — as its name suggest — can automatize the upgrade process for some packages (by default it only takes care of security updates, but this can be customized in `/etc/apt/apt.conf.d/50unattended-upgrades`). Note that this option can be set with the help of debconf by running `dpkg-reconfigure -plow unattended-upgrades`.

Other options can allow you to control the cache cleaning behavior with more precision. They are not listed here, but they are described in the `/etc/cron.daily/apt` script.

These tools work very well for servers, but desktop users generally prefer a more interactive system. That is why the "Debian desktop environment" task installs *gnome-packagekit* (at least when you select GNOME as desktop environment). It provides an icon in the notification area of desktop environments when updates are available; clicking on this icon then runs `gpk-update-viewer`, a simplified interface to perform updates. You can browse through available updates, read the short description of the relevant packages and the corresponding `changelog` entries, and select whether to apply the update or not on a case-by-case basis.

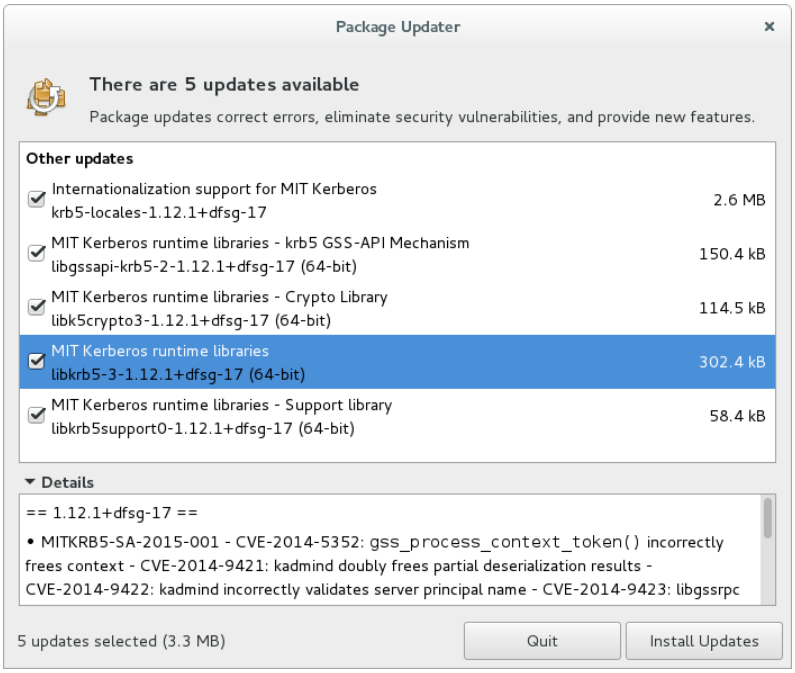

Figure 6.3 *Upgrading with `gpk-update-viewer`*

6.8. Automatic Upgrades

Since Falcot Corp has many computers but only limited manpower, its administrators try to make upgrades as automatic as possible. The programs in charge of these processes must therefore run with no human intervention.

6.8.1. Configuring `dpkg`

As we have already mentioned (see sidebar "Avoiding the configuration file questions" page 84), `dpkg` can be instructed not to ask for confirmation when replacing a configuration file (with the --force-confdef --force-confold options). Interactions can, however, have three other sources: some come from APT itself, some are handled by `debconf`, and some happen on the command line due to package configuration scripts.

6.8.2. Configuring APT

The case of APT is simple: the -y option (or --assume-yes) tells APT to consider the answer to all its questions to be "yes".

6.8.3. Configuring `debconf`

The case of `debconf` deserves more details. This program was, from its inception, designed to control the relevance and volume of questions displayed to the user, as well as the way they are shown. That is why its configuration requests a minimal priority for questions; only questions above the minimal priority are displayed. `debconf` assumes the default answer (defined by the package maintainer) for questions which it decided to skip.

The other relevant configuration element is the interface used by the front-end. If you choose `noninteractive` out of the choices, all user interaction is disabled. If a package tries to display an informative note, it will be sent to the administrator by email.

To reconfigure `debconf`, use the `dpkg-reconfigure` tool from the *debconf* package; the relevant command is `dpkg-reconfigure debconf`. Note that the configured values can be temporarily overridden with environment variables when needed (for instance, `DEBIAN_FRONTEND` controls the interface, as documented in the `debconf(7)` manual page).

6.8.4. Handling Command Line Interactions

The last source of interactions, and the hardest to get rid of, is the configuration scripts run by `dpkg`. There is unfortunately no standard solution, and no answer is overwhelmingly better than another.

The common approach is to suppress the standard input by redirecting the empty content of `/dev/null` into it with *`command`* `</dev/null`, or to feed it with an endless stream of newlines.

None of these methods is 100 % reliable, but they generally lead to the default answers being used, since most scripts consider a lack of reply as an acceptance of the default value.

6.8.5. The Miracle Combination

By combining the previous elements, it is possible to design a small but rather reliable script which can handle automatic upgrades.

Example 6.4 *Non-interactive upgrade script*

```
export DEBIAN_FRONTEND=noninteractive
yes '' | apt-get -y -o DPkg::options::="--force-confdef" -o DPkg::options::="--force-
       confold" dist-upgrade
```

IN PRACTICE

The Falcot Corp case

Falcot computers are a heterogeneous system, with machines having various functions. Administrators will therefore pick the most relevant solution for each computer.

In practice, the servers running *Jessie* are configured with the "miracle combination" above, and are kept up to date automatically. Only the most critical servers (the firewalls, for instances) are set up with `apticron`, so that upgrades always happen under the supervision of an administrator.

The office workstations in the administrative services also run *Jessie*, but they are equipped with *gnome-packagekit*, so that users trigger the upgrades themselves. The rationale for this decision is that if upgrades happen without an explicit action, the behavior of the computer might change unexpectedly, which could cause confusion for the main users.

In the lab, the few computers using *Testing* — to take advantage of the latest software versions — are not upgraded automatically either. Administrators only configure APT to prepare the upgrades but not enact them; when they decide to upgrade (manually), the tedious parts of refreshing package lists and downloading packages will be avoided, and administrators can focus on the really useful part.

6.9. Searching for Packages

With the large and ever-growing amount of software in Debian, there emerges a paradox: Debian usually has a tool for most tasks, but that tool can be very difficult to find amongst the myriad other packages. The lack of appropriate ways to search for (and to find) the right tool has long been a problem. Fortunately, this problem has almost entirely been solved.

The most trivial search possible is looking up an exact package name. If `apt show` *`package`* returns a result, then the package exists. Unfortunately, this requires knowing or even guessing the package name, which isn't always possible.

TIP

Package naming conventions

Some categories of packages are named according to a conventional naming scheme; knowing the scheme can sometimes allow you to guess exact package names. For instance, for Perl modules, the convention says that a module called `XML::Handler::Composer` upstream should be packaged as *libxml-handler-composer-perl*. The library enabling the use of the `gconf` system from Python is packaged as *python-gconf*. It is unfortunately not possible to define a fully general naming scheme for all packages, even though package maintainers usually try to follow the choice of the upstream developers.

A slightly more successful searching pattern is a plain-text search in package names, but it remains very limited. You can generally find results by searching package descriptions: since each package has a more or less detailed description in addition to its package name, a keyword search in these descriptions will often be useful. `apt-cache` and `axi-cache` are the tools of choice for this kind of search; for instance, `apt-cache search video` will return a list of all packages whose name or description contains the keyword "video".

For more complex searches, a more powerful tool such as `aptitude` is required. `aptitude` allows you to search according to a logical expression based on the package's meta-data fields. For instance, the following command searches for packages whose name contains kino, whose description contains video and whose maintainer's name contains paul:

```
$ aptitude search kino~dvideo~mpaul
p   kino  - Non-linear editor for Digital Video data
$ aptitude show kino
Package: kino
State: not installed
Version: 1.3.4-2.1+b1
Priority: extra
Section: video
Maintainer: Paul Brossier <piem@debian.org>
Architecture: amd64
Uncompressed Size: 8,472 k
Depends: libasound2 (>= 1.0.16), libatk1.0-0 (>= 1.12.4), libavc1394-0 (>=
         0.5.3), libavcodec56 (>= 6:11~beta1) | libavcodec-extra-56 (>=
         6:11~beta1), libavformat56 (>= 6:11~beta1), libavutil54 (>=
         6:11~beta1), libc6 (>= 2.14), libcairo2 (>= 1.2.4), libdv4,
         libfontconfig1 (>= 2.11), libfreetype6 (>= 2.2.1), libgcc1 (>=
         1:4.1.1), libgdk-pixbuf2.0-0 (>= 2.22.0), libglade2-0 (>= 1:2.6.4-2~),
         libglib2.0-0 (>= 2.12.0), libgtk2.0-0 (>= 2.24.0), libice6 (>=
         1:1.0.0), libiec61883-0 (>= 1.2.0), libpango-1.0-0 (>= 1.14.0),
         libpangocairo-1.0-0 (>= 1.14.0), libpangoft2-1.0-0 (>= 1.14.0),
         libquicktime2 (>= 2:1.2.2), libraw1394-11, libsamplerate0 (>= 0.1.7),
         libsm6, libstdc++6 (>= 4.9), libswscale3 (>= 6:11~beta1), libx11-6,
         libxext6, libxml2 (>= 2.7.4), libxv1, zlib1g (>= 1:1.1.4)
Recommends: ffmpeg, curl
Suggests: udev | hotplug, vorbis-tools, sox, mjpegtools, lame, ffmpeg2theora
Conflicts: kino-dvtitler, kino-timfx, kinoplus
Replaces: kino-dvtitler, kino-timfx, kinoplus
Provides: kino-dvtitler, kino-timfx, kinoplus
```

```
Description: Non-linear editor for Digital Video data
 Kino allows you to record, create, edit, and play movies recorded with DV
 camcorders. This program uses many keyboard commands for fast navigating and
 editing inside the movie.

 The kino-timfx, kino-dvtitler and kinoplus sets of plugins, formerly
 distributed as separate packages, are now provided with Kino.
Homepage: http://www.kinodv.org/

Tags: field::arts, hardware::camera, implemented-in::c, implemented-in::c++,
      interface::x11, role::program, scope::application, suite::gnome,
      uitoolkit::gtk, use::editing, use::learning, works-with::video,
      x11::application
```

The search only returns one package, *kino*, which satisfies all three criteria.

Even these multi-criteria searches are rather unwieldy, which explains why they are not used as much as they could. A new tagging system has therefore been developed, and it provides a new approach to searching. Packages are given tags that provide a thematical classification along several strands, known as a "facet-based classification". In the case of *kino* above, the package's tags indicate that Kino is a Gnome-based software that works on video data and whose main purpose is editing.

Browsing this classification can help you to search for a package which corresponds to known needs; even if it returns a (moderate) number of hits, the rest of the search can be done manually. To do that, you can use the ~G search pattern in `aptitude`, but it is probably easier to simply navigate the site where tags are managed:

➠ `http://debtags.alioth.debian.org/cloud/`

Selecting the works-with::video and use::editing tags yields a handful of packages, including the *kino* and *pitivi* video editors. This system of classification is bound to be used more and more as time goes on, and package managers will gradually provide efficient search interfaces based on it.

To sum up, the best tool for the job depends on the complexity of the search that you wish to do:

- `apt-cache` only allows searching in package names and descriptions, which is very convenient when looking for a particular package that matches a few target keywords;
- when the search criteria also include relationships between packages or other meta-data such as the name of the maintainer, `synaptic` will be more useful;
- when a tag-based search is needed, a good tool is `packagesearch`, a graphical interface dedicated to searching available packages along several criteria (including the names of the files that they contain). For usage on the command-line, `axi-cache` will fit the bill.
- finally, when the searches involve complex expressions with logic operations, the tool of choice will be `aptitude`'s search pattern syntax, which is quite powerful despite being somewhat obscure; it works in both the command-line and the interactive modes.

Keywords

Documentation
Solving problems
Log files
README.Debian
Manual
info