6.2.2. Installing and Removing

With APT, packages can be added or removed from the system, respectively with **aptitude install *package*** and **aptitude 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 **aptitude purge *package*** or **apt-get purge *package*** commands involve 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 on the computers you want to update and use the following commands:

```
# dpkg --set-selections <pkg-list
# apt-get dselect-upgrade
```

The first command registers the list of packages you wish to install, then 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 **aptitude** to install certain packages and remove others on the same command line by adding a suffix. With an **aptitude install** command, add “**-**” to the names of the packages you wish to remove. With an **aptitude 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*.

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

***TIP* `apt-get --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-get`** command. The following command reinstalls *postfix* even if it is already present:

```
# apt-get --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-get --reinstall`** to restore packages modified during an attack certainly cannot recover the system as it was. *Chapter 14, Security* details the necessary steps to take with a hacked system (see *Section 14.6, "Dealing with a Compromised Machine"*).

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 **`aptitude install package=version`**, but indicating its distribution of origin (*Stable*, *Testing* or *Unstable*) — with **`aptitude 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 *GOING FURTHER Old Package Versions: `snapshot.debian.org`*).

Example 6.2. Installation of the *Unstable* Version of *spamassassin*

```
# aptitude 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: **`aptitude clean`** entirely empties the directory; **`aptitude autoclean`** only removes packages which cannot 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).

6.2.3. System Upgrade

Regular upgrades are recommended, because they include the latest security updates. To upgrade, use **`aptitude safe-upgrade`** or **`apt-get upgrade`** (of course after **`aptitude 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`** because it will refuse to install packages which were not installed beforehand.

> ***TIP* Incremental Upgrade**
> As we explained earlier, the aim of the **`aptitude 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.bz2`** for the main section of *Squeeze* takes more than 8 MB). If you wish to upgrade regularly, these downloads can take up a lot of time.
>
> A “new feature” (available since *Etch*) is that APT can now download 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 **`aptitude 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.bz2`** 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`**.

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

To tell **`aptitude`** to use a specific distribution when searching for upgraded packages, you need to use the option **`-t`** or **`--target-release`**, followed by the name of the distribution you want (for example: **`aptitude -t stable safe-upgrade`**). To avoid specifying this option every time you use **`aptitude`**, 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 **`aptitude full-upgrade`** (the option used to be named **`dist-upgrade`**, for “distribution upgrade”). With this instruction, **`aptitude`** 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.

`aptitude dist-upgrade` is still available as a synonym for **`aptitude full-upgrade`**; **`apt-get`** only recognizes the former.

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 command **`man apt.conf`** (for details on manual pages, see next chapter).

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. However, 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 himself 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. Choosing one method (**`.d`** directory used directly or a file generated from that directory) is usually imposed 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::Target-Release`** configuration directive).

You can modify the priorities by adding entries in the file **`/etc/apt/preferences`** 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 500 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.gz`** 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-parties 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 5.0 for Debian *Lenny*). 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 5.10 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 5.10*
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
```


6.2.6. Working with Several Distributions

aptitude 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 available in *Testing* or *Unstable* without diverging too much from the system's initial state.

Even if you will occasionnaly encounter problems while mixing packages from different distributions, **aptitude** 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”*).

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 **aptitude 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 (**safe-upgrade** and **dist-upgrade**) are done within *Stable* except for packages already upgraded to an other distribution: those will follow updates available in the other distributions. We'll explain this behavior with the help of the default priorities set by APT below. Do not hesitate to use **apt-cache policy** (see sidebar) 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/preferencs** 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
```

6.3. 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 **aptitude 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 **aptitude search** and **aptitude show** work in the same way.

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.

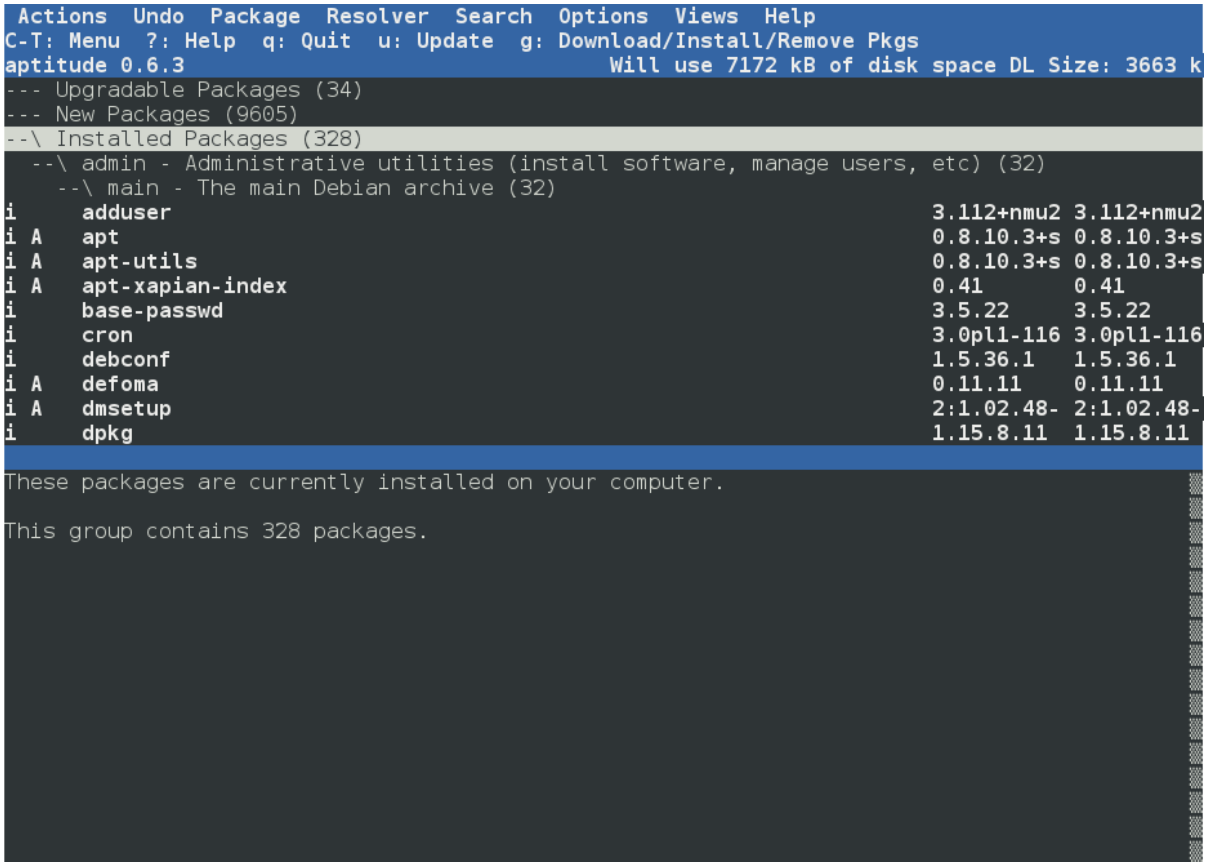

Figure 6.1. The **aptitude** Package Manager

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.

6.4.1.1. Tracking Automatically Installed Packages

One of the essential functionnalities of **aptitude** (which has also been integrated to **apt-get** since *Lenny*) is the tracking of packages installed only through dependencies. These packages are called “automatic” and are tagged with an “A” in the list of packages — they often include libraries for instance. When a package is removed, the corresponding automatic packages are also selected for removal unless another “manually installed” package depends on them. It is possible to mark a package as automatic (with **Shift**+**m**) or to remove the mark (**m** key). When maintaining a system with **aptitude**, 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. You can either navigate the list of installed packages and play with **Shift**+**m**, or apply the flag to entire sections (for example the **libs** section). This habit can help you to keep your system tidy and 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**).

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***:

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

> ***WORTH FOLLOWING*** **Recent Evolutions of apt-get and aptitude**
> Some of the advantages that **aptitude** historically had over **apt-get** have recently disappeared. For instance, since the release of *Lenny*, **apt-get** memorizes the packages that have been installed only to satisfy dependencies, just like **aptitude** has always done. It can also follow recommendations expressed by one package on another.
>
> Among the recent evolutions of **aptitude**, a new version with a graphical interface is currently being developed. Even if it's available in *Squeeze* (in the separate *aptitude-gtk* package), it's not complete yet and is subject to stability issues.

***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 run the previously suggested cleanup of “automatic” 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).

***ALTERNATIVE* `deborphan` and `debfoster`**

Before **aptitude** came to life with its tracking of 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 packages depends on. The resulting list can then serve as a basis to remove unneeded packages.

debfoster has a more elaborate approach, very similar to **aptitude**'s: 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.

6.4.1.2. 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-desktop-environment* package recommends *gnome-accessibility* (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: *gnome-accessibility* 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-desktop-environment* 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 installaion.

Suggestions between packages are also taken into account, but in a manner adapted to their specific status. For example, since *gnome-desktop-environment* suggests *gnome-audio*, 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 package 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.

6.4.1.3. 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 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 occasionaly 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.