Chapter 2. Debian package management

Table of Contents

Note

This chapter is written assuming the latest stable release is codename: **`jessie`**.

Debian is a volunteer organization which builds **consistent** distributions of pre-compiled binary packages of free software and distributes them from its archive.

The Debian archive is offered by many remote mirror sites for access through HTTP and FTP methods. It is also available as CD-ROM/DVD.

The Debian package management system, **when used properly**, offers the user to install **consistent sets of binary packages** to the system from the archive. Currently, there are 44893 packages available for the amd64 architecture.

The Debian package management system has a rich history and many choices for the front end user program and back end archive access method to be used. Currently, we recommend the following.

- apt-get(8) for all commandline operations, including package installation and removal, and dist-upgrades.
- aptitude(8) for an interactive text interface to manage the installed packages and to search the available packages.

Table 2.1. List of Debian package management tools

package	popcon	size	description
`apt`	V:842, I:999	3724	Advanced Packaging Tool (APT), front-end for **`dpkg`** providing "**`http`**", "**`ftp`**", and "**`file`**" archive access methods (**`apt-get`**/**`apt-cache`** commands included)
`aptitude`	V:232, I:991	4532	interactive terminal-based package manager with aptitude(8)
`tasksel`	V:45, I:968	718	tool for selecting tasks for installation on the Debian system (front-end for APT)
`unattended-upgrades`	V:73, I:455	343	enhancement package for APT to enable automatic installation of security upgrades
`dselect`	V:10, I:130	2620	terminal-based package manager (previous standard, front-end for APT and other old access methods)
`dpkg`	V:918, I:999	6656	package management system for Debian
`synaptic`	V:96, I:471	7672	graphical package manager (GNOME front-end for APT)

package	popcon	size	description
apt-utils	V:302, I:996	1393	APT utility programs: apt-extracttemplates(1), apt-ftparchive(1), and apt-sortpkgs(1)
apt-listchanges	V:385, I:790	446	package change history notification tool
apt-listbugs	V:8, I:13	520	lists critical bugs before each APT installation
apt-file	V:18, I:86	131	APT package searching utility — command-line interface
apt-rdepends	V:1, I:8	64	recursively lists package dependencies

2.1. Debian package management prerequisites

2.1.1. Package configuration

Here are some key points for package configuration on the Debian system.

- The manual configuration by the system administrator is respected. In other words, the package configuration system makes no intrusive configuration for the sake of convenience.
- Each package comes with its own configuration script with standardized user interface called debconf(7) to help initial installation process of the package.
- Debian Developers try their best to make your upgrade experience flawless with package configuration scripts.
- Full functionalities of packaged software are available to the system administrator. But ones with security risks are disabled in the default installation.
- If you manually activate a service with some security risks, you are responsible for the risk containment.
- Esoteric configuration may be manually enabled by the system administrator. This may create interferences with popular generic helper programs for the system configuration.

2.1.2. Basic precautions

Warning

Do not install packages from random mixture of suites. It probably breaks the package consistency which requires deep system management knowledge, such as compiler ABI, library version, interpreter features, etc.

The newbie Debian system administrator should stay with the **stable** release of Debian while applying only security updates. I mean that some of the following valid actions are better avoided, as a precaution, until you understand the Debian system very well. Here are some reminders.

- Do not include **testing** or **unstable** in "**/etc/apt/sources.list**".
- Do not mix standard Debian with other non-Debian archives such as Ubuntu in "**/etc/apt/sources.list**".
- Do not create "**/etc/apt/preferences**".
- Do not change default behavior of package management tools through configuration files without knowing their full impacts.
- Do not install random packages by "**dpkg -i <random_package>**".
- Do not ever install random packages by "**dpkg --force-all -i <random_package>**".
- Do not erase or alter files in "**/var/lib/dpkg/**".
- Do not overwrite system files by installing software programs directly compiled from source.
    - Install them into "**/usr/local**" or "**/opt**", if needed.

The non-compatible effects caused by above actions to the Debian package management system may leave your system unusable.

The serious Debian system administrator who runs mission critical servers, should use extra precautions.

- Do not install any packages including security updates from Debian without thoroughly testing them with your particular configuration under safe conditions.
    - You as the system administrator are responsible for your system in the end.

    - The long stability history of the Debian system is no guarantee by itself.

2.1.3. Life with eternal upgrades

Despite my warnings above, I know many readers of this document wish to run the **`testing`** or **`unstable`** suites of Debian as their main system for **self-administered Desktop environments**. This is because they work very well, are updated frequently, and offer the latest features.

> **Caution**
>
> For your **production server**, the **`stable`** suite with the security updates is recommended. The same can be said for desktop PCs on which you can spend limited administration efforts, e.g. for your mother's PC.

It takes no more than simply setting the distribution string in the "**`/etc/apt/sources.list`**" to the suite name: "**`testing`**" or "**`unstable`**"; or the codename: "**`stretch`**" or "**`sid`**". This makes you live **the life of eternal upgrades**.

The use of **`testing`** or **`unstable`** is **a lot of fun** but comes with some risks. Even though the **`unstable`** suite of the Debian system looks very stable for most of the times, there have been some package problems on the **`testing`** and **`unstable`** suite of the Debian system and a few of them were not so trivial to resolve. It may be **quite painful** for you. Sometimes, you may have a broken package or missing functionality for a few weeks.

Here are some ideas to ensure quick and easy recovery from bugs in Debian packages.

- Make the system **dual bootable** by installing the **`stable`** suite of the Debian system to another partition
- Make the installation CD handy for the **rescue boot**
- Consider installing **`apt-listbugs`** to check the Debian Bug Tracking System (BTS) information before the upgrade
- Learn the package system infrastructure enough to work around the problem
- Create a chroot or similar environment and run the latest

system in it in advance (see Section 9.10, “Virtualized system”)

(If you can not do any one of these precautionary actions, you are probably not ready for the **testing** and **unstable** suites.)

Enlightenment with the following saves a person from the eternal karmic struggle of upgrade hell and let him reach Debian nirvana.

2.1.4. Debian archive basics

Let's look into the Debian archive from a system user's perspective.

Tip

Official policy of the Debian archive is defined at Debian Policy Manual, Chapter 2 - The Debian Archive.

For the typical HTTP access, the archive is specified in the "**/etc/apt/sources.list**" file as the following, e.g. for the current **stable** = **jessie** system.

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

deb http://security.debian.org/ jessie/updates main
contrib
deb-src http://security.debian.org/ jessie/updates main
contrib
```

Please note "**ftp.XX.debian.org**" must be replaced with appropriate mirror site URL for your location, for USA "**ftp.us.debian.org**", which can be found in the list of Debian worldwide mirror sites. The status of these servers can be checked at Debian Mirror Checker site.

Here, I tend to use codename "**jessie**" instead of suite name "**stable**" to avoid surprises when the next **stable** is released.

The meaning of "**/etc/apt/sources.list**" is described in sources.list(5) and key points are followings.

- The "**deb**" line defines for the binary packages.
- The "**deb-src**" line defines for the source packages.

- The 1st argument is the root URL of the Debian archive.
- The 2nd argument is the distribution name: either the suite name or the codename.
- The 3rd and following arguments are the list of valid archive area names of the Debian archive.

The "**`deb-src`**" lines can safely be omitted (or commented out by placing "#" at the start of the line) if it is just for **`aptitude`** which does not access source related meta data. It speeds up the updates of the archive meta data. The URL can be "**`http://`**", "**`ftp://`**", "**`file://`**", ….

Tip

If "**`sid`**" is used in the above example instead of "**`jessie`**", the "**`deb: http://security.debian.org/ …`**" line for security updates in the "**`/etc/apt/sources.list`**" is not required. This is because there is no security update archive for "**`sid`**" (**`unstable`**).

Here is the list of URL of the Debian archive sites and suite name or codename used in the configuration file.

Table 2.2. List of Debian archive sites

archive URL	suite name (codename)	purpose
http://ftp.XX.debian.org/debian/	**`stable`** (**`jessie`**)	stable (jessie) release
http://ftp.XX.debian.org/debian/	**`testing`** (**`stretch`**)	testing (stretch) release
http://ftp.XX.debian.org/debian/	**`unstable`** (**`sid`**)	unstable (sid) release
http://ftp.XX.debian.org/debian/	**`experimental`**	experimental pre-release (optional, only for developer)
http://ftp.XX.debian.org/debian/	**`stable-proposed-updates`**	Updates for the next stable point release (optional)
http://security.debian.org/	**`stable/updates`**	security updates for stable release (important)

archive URL	suite name (codename)	purpose
http://security.debian.org/	**testing/updates**	security updates for testing release (important)
http://ftp.XX.debian.org/debian/	**jessie-updates**	compatible updates for spam filter, IM clients, etc. for jessie
http://ftp.XX.debian.org/debian/	**jessie-backports**	newer backported packages for jessie (optional)

Caution

Only pure **stable** release with security updates provides the best stability. Running mostly **stable** release mixed with some packages from **testing** or **unstable** release is riskier than running pure **unstable** release for library version mismatch etc. If you really need the latest version of some programs under **stable** release, please use packages from jessie-updates and http://backports.debian.org (see Section 2.7.4, "Updates and Backports") services. These services must be used with extra care.

Caution

You should basically list only one of **stable**, **testing**, or **unstable** suites in the "**deb**" line. If you list any combination of **stable**, **testing**, and **unstable** suites in the "**deb**" line, APT programs slow down while only the latest archive is effective. Multiple listing makes sense for these when the "**/etc/apt/preferences**" file is used with clear objectives (see Section 2.7.3, "Tweaking candidate version").

Tip

For the Debian system with the **stable** and **testing** suites, it is a good idea to include lines with "**http://security.debian.org/**" in the "**/etc/apt/sources.list**" to enable security updates as in the example above.

Note

The security bugs for the **`stable`** archive are fixed by the Debian security team. This activity has been quite rigorous and reliable. Those for the **`testing`** archive may be fixed by the Debian testing security team. For several reasons, this activity is not as rigorous as that for **`stable`** and you may need to wait for the migration of fixed **`unstable`** packages. Those for the **`unstable`** archive are fixed by the individual maintainer. Actively maintained **`unstable`** packages are usually in a fairly good shape by leveraging latest upstream security fixes. See Debian security FAQ for how Debian handles security bugs.

Table 2.3. List of Debian archive area

area	number of packages	criteria of package component
`main`	44118	DFSG compliant and no dependency to **`non-free`**
`contrib`	266	DFSG compliant but having dependency to **`non-free`**
`non-free`	509	not DFSG compliant

Here the number of packages in the above is for the amd64 architecture. The **`main`** area provides the Debian system (see Section 2.1.5, “Debian is 100% free software”).

The Debian archive organization can be studied best by pointing your browser to the each archive URL appended with **`dists`** or **`pool`**.

The distribution is referred by two ways, the suite or codename. The word distribution is alternatively used as the synonym to the suite in many documentations. The relationship between the suite and the codename can be summarized as the following.

Table 2.4. The relationship between suite and codename

Timing	suite = `stable`	suite = `testing`	suite = `unstable`
after the **`jessie`** release	codename = **`jessie`**	codename = **`stretch`**	codename = **`sid`**

Timing	**suite = `stable`**	**suite = `testing`**	**suite = `unstable`**
after the **`stretch`** release	codename = **`stretch`**	codename = **`buster`**	codename = **`sid`**

The history of codenames are described in Debian FAQ: 6.2.1 Which other codenames have been used in the past?

In the stricter Debian archive terminology, the word "section" is specifically used for the categorization of packages by the application area. (Although, the word "main section" may sometimes be used to describe the Debian archive area named as "main".)

Every time a new upload is done by a Debian developer (DD) to the **`unstable`** archive (via incoming processing), the DD is required to ensure uploaded packages to be compatible with the latest set of packages in the latest **`unstable`** archive.

If DD breaks this compatibility intentionally for important library upgrade etc, there is usually announcement to the debian-devel mailing list etc.

Before a set of packages are moved by the Debian archive maintenance script from the **`unstable`** archive to the **`testing`** archive, the archive maintenance script not only checks the maturity (about 10 days old) and the status of the RC bug reports for the packages but also tries to ensure them to be compatible with the latest set of packages in the **`testing`** archive. This process makes the **`testing`** archive very current and usable.

Through the gradual archive freeze process led by the release team, the **`testing`** archive is matured to make it completely consistent and bug free with some manual interventions. Then the new **`stable`** release is created by assigning the codename for the old **`testing`** archive to the new **`stable`** archive and creating the new codename for the new **`testing`** archive. The initial contents of the new **`testing`** archive is exactly the same as that of the newly released **`stable`** archive.

Both the **`unstable`** and the **`testing`** archives may suffer temporary glitches due to several factors.

- Broken package upload to the archive (mostly for **`unstable`**)
- Delay of accepting the new packages to the archive (mostly for **`unstable`**)

- Archive synchronization timing issue (both for **`testing`** and **`unstable`**)
- Manual intervention to the archive such as package removal (more for **`testing`**) etc.

So if you ever decide to use these archives, you should be able to fix or work around these kinds of glitches.

Caution

For about few months after a new **`stable`** release, most desktop users should use the **`stable`** archive with its security updates even if they usually use **`unstable`** or **`testing`** archives. For this transition period, both **`unstable`** and **`testing`** archives are not good for most people. Your system is difficult to keep in good working condition with the **`unstable`** archive since it suffers surges of major upgrades for core packages. The **`testing`** archive is not useful either since it contains mostly the same content as the **`stable`** archive without its security support (Debian testing-security-announce 2008-12). After a month or so, the **`unstable`** archive may be usable if you are careful.

Tip

When tracking the **`testing`** archive, a problem caused by a removed package is usually worked around by installing corresponding package from the **`unstable`** archive which is uploaded for bug fix.

See Debian Policy Manual for archive definitions.

- "Sections"
- "Priorities"
- "Base system"
- "Essential packages"

2.1.5. Debian is 100% free software

Debian is 100% free software because of the followings:

- Debian installs only free software by default to respect user's freedoms.

- Debian provides only free software in **`main`**.
- Debian recommends running only free software from **`main`**.
- No packages in **`main`** depend nor recommend packages in **`non-free`** nor **`contrib`**.

Some people wonder if the following 2 facts contradict or not.

-

  "Debian will remain 100% free". (First term of Debian Social Contract)
- Debian servers host some **`non-free`** and **`contrib`** packages.

These do not contradict, because of the followings.

- The Debian system is 100% free and its packages are hosted by Debian servers in the **`main`** area.
- Packages outside of the Debian system are hosted by Debian servers in the **`non-free`** and **`contrib`** areas.

These are precisely explained in the 4th and 5th terms of Debian Social Contract:

- Our priorities are our users and free software
  - We will be guided by the needs of our users and the free software community. We will place their interests first in our priorities. We will support the needs of our users for operation in many different kinds of computing environments. We will not object to non-free works that are intended to be used on Debian systems, or attempt to charge a fee to people who create or use such works. We will allow others to create distributions containing both the Debian system and other works, without any fee from us. In furtherance of these goals, we will provide an integrated system of high-quality materials with no legal restrictions that would prevent such uses of the system.
- Works that do not meet our free software standards
  - We acknowledge that some of our users require the use of works that do not conform to the Debian Free Software Guidelines. We have created "**`contrib`**" and "**`non-free`**" areas in our archive for these works. The packages in these areas are not part of the Debian system, although they have been configured for use with Debian. We encourage CD manufacturers to read the licenses of the

> packages in these areas and determine if they can distribute the packages on their CDs. Thus, although non-free works are not a part of Debian, we support their use and provide infrastructure for non-free packages (such as our bug tracking system and mailing lists).

Users should be aware of the risks of using packages in the **non-free** and **contrib** areas:

- lack of freedom for such software packages
- lack of support from Debian on such software packages (Debian can't support software properly without having access to its source code.)
- contamination of your 100% free Debian system

The Debian Free Software Guidelines are the free software standards for Debian. Debian interprets "software" in the widest scope including document, firmware, logo, and artwork data in the package. This makes Debian's free software standards very strict ones.

In order to meet this strict free software standards required for **main**, Debian unbrands Mozilla software packages such as Firefox, Thunderbird, and Seamonkey by removing their logo and some artwork data; and ships them as Iceweasel, Icedove, and Iceape, respectively.

Typical **non-free** and **contrib** packages include freely distributable packages of following types:

- Document packages under GNU Free Documentation License with invariant sections such as ones for GCC and Make. (mostly found in the **non-free/doc** section.)
- Firmware packages containing sourceless binary data such as ones listed in Section 9.9.6, “Hardware drivers and firmware” as non-free. (mostly found in the **non-free/kernel** section.)
- Game and font packages with restriction on commercial use and/or content modification.

Please note that the number of **non-free** and **contrib** packages is less than 2% of that of **main** packages. Enabling access to the **non-free** and **contrib** areas does not obscure the source of packages. Interactive full screen use of aptitude(8) provides you with full visibility and control over what packages are installed from which area to keep your system as free as you wish.

2.1.6. Package dependencies

The Debian system offers a consistent set of binary packages through its versioned binary dependency declaration mechanism in the control file fields. Here is a bit over simplified definition for them.

- "Depends"
  - This declares an absolute dependency and all of the packages listed in this field must be installed at the same time or in advance.
- "Pre-Depends"
  - This is like Depends, except that it requires completed installation of the listed packages in advance.
- "Recommends"
  - This declares a strong, but not absolute, dependency. Most users would not want the package unless all of the packages listed in this field are installed.
- "Suggests"
  - This declares a weak dependency. Many users of this package may benefit from installing packages listed in this field but can have reasonable functions without them.
- "Enhances"
  - This declares a week dependency like Suggests but works in the opposite direction.
- "Breaks"
  - This declares a package incompatibility usually with some version specification. Generally the resolution is to upgrade all of the packages listed in this field.
- "Conflicts"
  - This declares an absolute incompatibility. All of the packages listed in this field must be removed to install this package.
- "Replaces"
  - This is declared when files installed by this package

replace files in the listed packages.

- "Provides"
    - This is declared when this package provide all of the files and functionality in the listed packages.

    Note

    Please note that defining "Provides", "Conflicts" and "Replaces" simultaneously to an virtual package is the sane configuration. This ensures that only one real package providing this virtual package can be installed at any one time.

The official definition including source dependency can be found in the Policy Manual: Chapter 7 - Declaring relationships between packages.

2.1.7. The event flow of the package management

Here is a summary of the simplified event flow of the package management by APT.

- **Update** ("**`aptitude update`**" or "**`apt-get update`**"):
    1. Fetch archive metadata from remote archive
    2. Reconstruct and update local metadata for use by APT
- **Upgrade** ("**`aptitude safe-upgrade`**" and "**`aptitude full-upgrade`**", or "**`apt-get upgrade`**" and "**`apt-get dist-upgrade`**"):
    1. Chose candidate version which is usually the latest available version for all installed packages (see Section 2.7.3, “Tweaking candidate version” for exception)
    2. Make package dependency resolution
    3. Fetch selected binary packages from remote archive if candidate version is different from installed version
    4. Unpack fetched binary packages

5. Run **preinst** script
6. Install binary files
7. Run **postinst** script

- **Install** ("**`aptitude install …`**" or "**`apt-get install …`**"):
    1. Chose packages listed on the command line
    2. Make package dependency resolution
    3. Fetch selected binary packages from remote archive
    4. Unpack fetched binary packages
    5. Run **preinst** script
    6. Install binary files
    7. Run **postinst** script
- **Remove** ("**`aptitude remove …`**" or "**`apt-get remove …`**"):
    1. Chose packages listed on the command line
    2. Make package dependency resolution
    3. Run **prerm** script
    4. Remove installed files **except** configuration files
    5. Run **postrm** script
- **Purge** ("**`aptitude purge …`**" or "**`apt-get purge …`**"):
    1. Chose packages listed on the command line
    2. Make package dependency resolution
    3. Run **prerm** script
    4. Remove installed files **including** configuration files
    5. Run **postrm** script

Here, I intentionally skipped technical details for the sake of big picture.

2.1.8. First response to package management troubles

You should read the fine official documentation. The first document to read is the Debian specific "**/usr/share/doc/<package_name>/README.Debian**". Other documentation in "**/usr/share/doc/<package_name>/**" should be consulted too. If you set shell as Section 1.4.2, "Customizing bash", type the following.

```
$ cd <package_name>
$ pager README.Debian
$ mc
```

You may need to install the corresponding documentation package named with "**-doc**" suffix for detailed information.

If you are experiencing problems with a specific package, make sure to check out the Debian bug tracking system (BTS) sites, first.

Table 2.5. List of key web site to resolving problems with a specific package

web site	command
Home page of the Debian bug tracking system (BTS)	**sensible-browser "http://bugs.debian.org/"**
The bug report of a known package name	**sensible-browser "http://bugs.debian.org/<package_name>"**
The bug report of known bug number	**sensible-browser "http://bugs.debian.org/<bug_number>"**

Search Google with search words including "**site:debian.org**", "**site:wiki.debian.org**", "**site:lists.debian.org**", etc.

When you file a bug report, please use reportbug(1) command.

2.2. Basic package management operations

Repository based package management operations on the Debian system can be performed by many APT-based package management tools available on the Debian system. Here, we explain 2 basic package management tools: **apt-get** / **apt-cache** and **aptitude**.

For the package management operation which involves package installation or updates package metadata, you need to have root privilege.

2.2.1. `apt-get` / `apt-cache` vs. `aptitude`

Although **`aptitude`** is a very nice interactive tool which the author mainly uses, you should know some cautionary facts:

- The **`aptitude`** command is not recommended for the release-to-release system upgrade on the **`stable`** Debian system after the new release.
    - The use of "**`apt-get dist-upgrade`**" is recommended for it. See Bug #411280.
- The **`aptitude`** command sometimes suggests mass package removals for the system upgrade on the **`testing`** or **`unstable`** Debian system.
    - This situation has frightened many system administrators. Don't panic.
    - This seems to be caused mostly by the version skew among packages depended or recommended by a meta-package such as **`gnome-core`**.
    - This can be resolved by selecting "Cancel pending actions" in the **`aptitude`** command menu, exiting **`aptitude`**, and using "**`apt-get dist-upgrade`**".

The **`apt-get`** and **`apt-cache`** commands are the most **basic** APT-based package management tools.

- **`apt-get`** and **`apt-cache`** offer only the commandline user interface.
- **`apt-get`** is most suitable for the **major system upgrade** between releases, etc.
- **`apt-get`** offers a **robust** package dependency resolver.
- **`apt-get`** is less demanding on hardware resources. It consumes less memory and runs faster.
- **`apt-cache`** offers a **standard** regex based search on the package name and description.
- **`apt-get`** and **`apt-cache`** can manage multiple versions of

packages using **`/etc/apt/preferences`** but it is quite cumbersome.

The **`aptitude`** command is the most **versatile** APT-based package management tool.

- **`aptitude`** offers the fullscreen interactive text user interface.
- **`aptitude`** offers the commandline user interface, too.
- **`aptitude`** is most suitable for the **daily interactive package management** such as inspecting installed packages and searching available packages.
- **`aptitude`** is more demanding on hardware resources. It consumes more memory and runs slower.
- **`aptitude`** offers an **enhanced** regex based search on all of the package metadata.
- **`aptitude`** can manage multiple versions of packages without using **`/etc/apt/preferences`** and it is quite intuitive.

2.2.2. Basic package management operations with the commandline

Here are basic package management operations with the commandline using aptitude(8) and apt-get(8) /apt-cache(8).

Table 2.6. Basic package management operations with the commandline using aptitude(8) and apt-get(8) /apt-cache(8)

`aptitude` syntax	**`apt-get`/`apt-cache` syntax**	**description**
`aptitude update`	**`apt-get update`**	update package archive metadata
`aptitude install foo`	**`apt-get install foo`**	install candidate version of "**`foo`**" package with its dependencies
`aptitude safe-upgrade`	**`apt-get upgrade`**	install candidate version of installed packages without removing any other packages
`aptitude full-upgrade`	**`apt-get dist-upgrade`**	install candidate version of installed packages while removing other packages if needed

aptitude syntax	**apt-get/apt-cache syntax**	**description**
aptitude remove foo	**apt-get remove foo**	remove "**foo**" package while leaving its configuration files
N/A	**apt-get autoremove**	remove auto-installed packages which are no longer required
aptitude purge foo	**apt-get purge foo**	purge "**foo**" package with its configuration files
aptitude clean	**apt-get clean**	clear out the local repository of retrieved package files completely
aptitude autoclean	**apt-get autoclean**	clear out the local repository of retrieved package files for outdated packages
aptitude show foo	**apt-cache show foo**	display detailed information about "**foo**" package
aptitude search <regex>	**apt-cache search <regex>**	search packages which match <regex>
aptitude why <regex>	N/A	explain the reason why <regex> matching packages should be installed
aptitude why-not <regex>	N/A	explain the reason why <regex> matching packages can not be installed

Note

Although the **aptitude** command comes with rich features such as its enhanced package resolver, this complexity has caused (or may still causes) some regressions such as Bug #411123, Bug #514930, and Bug #570377. In case of doubt, please use the **apt-get** and **apt-cache** commands over the **aptitude** command.

Note

Since **apt-get** and **aptitude** share auto-installed package status (see Section 2.5.5, “The package state for APT”) after **lenny**, you can mix these tools without major troubles (see Bug #594490).

The "**aptitude why <regex>**" can list more information by "**aptitude -v why <regex>**". Similar information can be obtained by "**apt-cache rdepends <package>**".

When **aptitude** command is started in the commandline mode and faces some issues such as package conflicts, you can switch to the full screen interactive mode by pressing "**e**"-key later at the prompt.

You may provide command options right after "**aptitude**".

Table 2.7. Notable command options for aptitude(8)

command option	description
-s	simulate the result of the command
-d	download only but no install/upgrade
-D	show brief explanations before the automatic installations and removals

See aptitude(8) and "aptitude user's manual" at "**/usr/share/doc/aptitude/README**" for more.

Tip

The **dselect** package is still available and was the preferred full screen interactive package management tool in previous releases.

2.2.3. Interactive use of aptitude

For the interactive package management, you start **aptitude** in interactive mode from the console shell prompt as follows.

```
$ sudo aptitude -u
Password:
```

This updates the local copy of the archive information and display the package list in the full screen with menu. Aptitude places its configuration at "**~/.aptitude/config**".

Tip

If you want to use root's configuration instead of user's one, use "**`sudo -H aptitude …`**" instead of "**`sudo aptitude …`**" in the above expression.

Tip

`Aptitude` automatically sets **pending actions** as it is started interactively. If you do not like it, you can reset it from menu: "Action" → "Cancel pending actions".

2.2.4. Key bindings of aptitude

Notable key strokes to browse status of packages and to set "planned action" on them in this full screen mode are the following.

Table 2.8. List of key bindings for aptitude

key	key binding
`F10` or `Ctrl-t`	menu
`?`	display **help** for keystroke (more complete listing)
`F10` → Help → User's Manual	display User's Manual
`u`	update package archive information
`+`	mark the package for the **upgrade** or the **install**
`-`	mark the package for the **remove** (keep configuration files)
`_`	mark the package for the **purge** (remove configuration files)
`=`	place the package on **hold**
`U`	mark all upgradable packages (function as **full-upgrade**)
`g`	start **downloading** and **installing** selected packages
`q`	quit current screen and save changes
`x`	quit current screen and discard changes
`Enter`	view information about a package
`C`	view a package's changelog

key	key binding
`l`	change the limit for the displayed packages
`/`	search for the first match
`\`	repeat the last search

The file name specification of the command line and the menu prompt after pressing "**`l`**" and "**`//`**" take the aptitude regex as described below. Aptitude regex can explicitly match a package name using a string started by "**`~n`** and followed by the package name.

Tip

You need to press "**`U`**" to get all the installed packages upgraded to the **candidate version** in the visual interface. Otherwise only the selected packages and certain packages with versioned dependency to them are upgraded to the **candidate version**.

2.2.5. Package views under aptitude

In the interactive full screen mode of aptitude(8), packages in the package list are displayed as the next example.

```
idA   libsmbclient                                   -2220kB
3.0.25a-1  3.0.25a-2
```

Here, this line means from the left as the following.

- The "current state" flag (the first letter)
- The "planned action" flag (the second letter)
- The "automatic" flag (the third letter)
- The Package name
- The change in disk space usage attributed to "planned action"
- The current version of the package
- The candidate version of the package

Tip

The full list of flags are given at the bottom of **Help** screen shown by pressing "**?**".

The **candidate version** is chosen according to the current local preferences (see apt_preferences(5) and Section 2.7.3, “Tweaking candidate version”).

Several types of package views are available under the menu "**Views**".

Table 2.9. List of views for aptitude

view	status	description of view
`Package View`	Good	see Table 2.10, “The categorization of standard package views” (default)
`Audit Recommendations`	Good	list packages which are recommended by some installed packages but not yet installed
`Flat Package List`	Good	list packages without categorization (for use with regex)
`Debtags Browser`	Very usable	list packages categorized according to their debtags entries
`Categorical Browser`	Deprecated	list packages categorized according to their category (use **`Debtags Browser`**, instead)

Note

Please help us improving tagging packages with debtags!

The standard "**`Package View`**" categorizes packages somewhat like **`dselect`** with few extra features.

Table 2.10. The categorization of standard package views

category	description of view
`Upgradable Packages`	list packages organized as **`section`** → **`area`** → **`package`**

category	description of view
`New Packages`	, ,
`Installed Packages`	, ,
`Not Installed Packages`	, ,
`Obsolete and Locally Created Packages`	, ,
`Virtual Packages`	list packages with the same function
`Tasks`	list packages with different functions generally needed for a task

Tip

Tasks view can be used to cherry pick packages for your task.

2.2.6. Search method options with aptitude

Aptitude offers several options for you to search packages using its regex formula.

- Shell commandline:
  - "**`aptitude search '<aptitude_regex>'`**" to list installation status, package name and short description of matching packages
  - "**`aptitude show '<package_name>'`**" to list detailed description of the package
- Interactive full screen mode:
  - "**`l`**" to limit package view to matching packages
  - "**`/`**" for search to a matching package
  - "**`\`**" for backward search to a matching package
  - "**`n`**" for find-next
  - "**`N`**" for find-next (backward)

Tip

The string for <package_name> is treated as the exact string match to the package name unless it is started explicitly with "~" to be the regex formula.

2.2.7. The aptitude regex formula

The aptitude regex formula is mutt-like extended **ERE** (see Section 1.6.2, “Regular expressions”) and the meanings of the **`aptitude`** specific special match rule extensions are as follows.

Table 2.11. List of the aptitude regex formula

description of the extended match rule	regex formula
match on package name	**`~n<regex_name>`**
match on description	**`~d<regex_description>`**
match on task name	**`~t<regex_task>`**
match on debtag	**`~G<regex_debtag>`**
match on maintainer	**`~m<regex_maintainer>`**
match on package section	**`~s<regex_section>`**
match on package version	**`~V<regex_version>`**
match archive	**`~A{jessie,stretch,sid}`**
match origin	**`~O{debian,…}`**
match priority	**`~p{extra,important,optional,required,standard}`**
match essential packages	**`~E`**
match virtual packages	**`~v`**
match new packages	**`~N`**
match with pending action	**`~a{install,upgrade,downgrade,remove,purge,hold,keep}`**

description of the extended match rule	**regex formula**
match installed packages	`~i`
match installed packages with **A**-mark (auto installed packages)	`~M`
match installed packages without **A**-mark (administrator selected packages)	`~i!~M`
match installed and upgradable packages	`~U`
match removed but not purged packages	`~c`
match removed, purged or can-be-removed packages	`~g`
match packages declaring a broken dependency	`~b`
match packages declaring broken	`~B<type>`

description of the extended match rule	regex formula
dependency of <type>	
match <pattern> packages declaring dependency of <type>	**`~D[<type>:]<pattern>`**
match <pattern> packages declaring broken dependency of <type>	**`~DB[<type>:]<pattern>`**
match packages to which the <pattern> matching package declares dependency <type>	**`~R[<type>:]<pattern>`**
match packages to which the <pattern> matching package declares broken dependency <type>	**`~RB[<type>:]<pattern>`**
match packages to which some other installed packages depend on	**`~R~i`**

description of the extended match rule	**regex formula**
match packages to which no other installed packages depend on	**`!~R~i`**
match packages to which some other installed packages depend or recommend on	**`~R~i|~Rrecommends:~i`**
match <pattern> package with filtered version	**`~S filter <pattern>`**
match all packages (true)	**`~T`**
match no packages (false)	**`~F`**

- The regex part is the same **ERE** as the one used in typical Unix-like text tools using "**^**", "**.***", "**$**" etc. as in egrep(1), awk(1) and perl(1).
- The dependency <type> is one of (depends, predepends, recommends, suggests, conflicts, replaces, provides) specifying the package interrelationship.
- The default dependency <type> is "depends".

Tip

When <regex_pattern> is a null string, place "**~T**" immediately after the command.

Here are some short cuts.

- "**`~P<term>`**" == "**`~Dprovides:<term>`**"
- "**`~C<term>`**" == "**`~Dconflicts:<term>`**"
- "**`…~W term`**" == "**`(…|term)`**"

Users familiar with **`mutt`** pick up quickly, as mutt was the inspiration for the expression syntax. See "SEARCHING, LIMITING, AND EXPRESSIONS" in the "User's Manual" "**`/usr/share/doc/aptitude/README`**".

Note

With the **`lenny`** version of aptitude(8), the new **long form** syntax such as "**`?broken`**" may be used for regex matching in place for its old **short form** equivalent "**`~b`**". Now space character " " is considered as one of the regex terminating character in addition to tilde character "~". See "User's Manual" for the new **long form** syntax.

2.2.8. Dependency resolution of aptitude

The selection of a package in **`aptitude`** not only pulls in packages which are defined in its "**`Depends:`**" list but also defined in the "**`Recommends:`**" list if the menu "**`F10`** → Options → Preferences → Dependency handling" is set accordingly. These auto installed packages are removed automatically if they are no longer needed under **`aptitude`**.

2.2.9. Package activity logs

You can check package activity history in the log files.

Table 2.12. The log files for package activities

file	content
`/var/log/dpkg.log`	Log of **`dpkg`** level activity for all package activities
`/var/log/apt/term.log`	Log of generic APT activity
`/var/log/aptitude`	Log of **`aptitude`** command activity

In reality, it is not so easy to get meaningful understanding quickly out from these logs. See Section 9.2.10, “Recording changes in configuration files” for easier way.

2.3. Examples of aptitude operations

Here are few examples of aptitude(8) operations.

2.3.1. Listing packages with regex matching on package names

The following command lists packages with regex matching on package names.

```
$ aptitude search '~n(pam|nss).*ldap'
p libnss-ldap - NSS module for using LDAP as a naming
service
p libpam-ldap - Pluggable Authentication Module allowing
LDAP interfaces
```

This is quite handy for you to find the exact name of a package.

2.3.2. Browsing with the regex matching

The regex "**`~dipv6`**" in the "New Flat Package List" view with "**`l`**" prompt, limits view to packages with the matching description and let you browse their information interactively.

2.3.3. Purging removed packages for good

You can purge all remaining configuration files of removed packages.

Check results of the following command.

```
# aptitude search '~c'
```

If you think listed packages are OK to be purged, execute the following command.

```
# aptitude purge '~c'
```

You may want to do the similar in the interactive mode for fine grained control.

You provide the regex "**`~c`**" in the "New Package View" view with "**`l`**" prompt. This limits the package view only to regex matched packages, i.e., "removed but not purged". All these regex matched packages can be shown by pressing "**`[`**" at top level headings.

Then you press "**_**" at top level headings such as "Not Installed Packages". Only regex matched packages under the heading are marked to be purged by this. You can exclude some packages to be purged by pressing "**=**" interactively for each of them.

This technique is quite handy and works for many other command keys.

2.3.4. Tidying auto/manual install status

Here is how I tidy auto/manual install status for packages (after using non-aptitude package installer etc.).

1. Start **`aptitude`** in interactive mode as root.
2. Type "**`u`**", "**`U`**", "**`f`**" and "**`g`**" to update and upgrade package list and packages.
3. Type "**`l`**" to enter the package display limit as "**`~i(~R~i|~Rrecommends:~i)`**" and type "**`M`**" over "**`Installed Packages`**" as auto installed.
4. Type "**`l`**" to enter the package display limit as "**`~prequired|~pimportant|~pstandard|~E`**" and type "**`m`**" over "**`Installed Packages`**" as manual installed.
5. Type "**`l`**" to enter the package display limit as "**`~i!~M`**" and remove unused package by typing "**`-`**" over each of them after exposing them by typing "**`[`**" over "**`Installed Packages`**".
6. Type "**`l`**", to enter the package display limit as "**`~i`**"; then type "**`m`**" over "**`Tasks`**", to mark that packages as manual installed.
7. Exit **`aptitude`**.
8. Start "**`apt-get -s autoremove|less`**" as root to check what are not used.
9. Restart **`aptitude`** in interactive mode and mark needed packages as "**`m`**".
10. Restart "**`apt-get -s autoremove|less`**" as root to recheck REMOVED contain only expected packages.
11. Start "**`apt-get autoremove|less`**" as root to autoremove unused packages.

The "**`m`**" action over "**`Tasks`**" is an optional one to prevent mass package removal situation in future.

2.3.5. System wide upgrade

Note

When moving to a new release etc, you should consider to perform a clean installation of new system even though Debian is upgradable as described below. This provides you a chance to remove garbages collected and exposes you to the best combination of latest packages. Of course, you should make a full backup of system to a safe place (see Section 10.2, “Backup and recovery”) before doing this. I recommend to make a dual boot configuration using different partition to have the smoothest transition.

You can perform system wide upgrade to a newer release by changing contents of the "**`/etc/apt/sources.list`**" file pointing to a new release and running the "**`apt-get update; apt-get dist-upgrade`**" command.

To upgrade from **`stable`** to **`testing`** or **`unstable`**, you replace "**`jessie`**" in the "**`/etc/apt/sources.list`**" example of Section 2.1.4, “Debian archive basics” with "**`stretch`**" or "**`sid`**".

In reality, you may face some complications due to some package transition issues, mostly due to package dependencies. The larger the difference of the upgrade, the more likely you face larger troubles. For the transition from the old **`stable`** to the new **`stable`** after its release, you can read its new Release Notes and follow the exact procedure described in it to minimize troubles.

When you decide to move from **`stable`** to **`testing`** before its formal release, there are no Release Notes to help you. The difference between **`stable`** and **`testing`** could have grown quite large after the previous **`stable`** release and makes upgrade situation complicated.

You should make precautionary moves for the full upgrade while gathering latest information from mailing list and using common senses.

1. Read previous "Release Notes".
2. Backup entire system (especially data and configuration information).
3. Have bootable media handy for broken bootloader.
4. Inform users on the system well in advance.

5. Record upgrade activity with script(1).
6. Apply "unmarkauto" to required packages, e.g., "**`aptitude unmarkauto vim`**", to prevent removal.
7. Minimize installed packages to reduce chance of package conflicts, e.g., remove desktop task packages.
8. Remove the "**`/etc/apt/preferences`**" file (disable apt-pinning).
9. Try to upgrade step wise: **`oldstable`** → **`stable`** → **`testing`** → **`unstable`**.
10. Update the "**`/etc/apt/sources.list`**" file to point to new archive only and run "**`aptitude update`**".
11. Install, optionally, new **core packages** first, e.g., "**`aptitude install perl`**".
12. Run the "**`apt-get -s dist-upgrade`**" command to assess impact.
13. Run the "**`apt-get dist-upgrade`**" command at last.

Caution

It is not wise to skip major Debian release when upgrading between **`stable`** releases.

Caution

In previous "Release Notes", GCC, Linux Kernel, initrd-tools, Glibc, Perl, APT tool chain, etc. have required some special attention for system wide upgrade.

For daily upgrade in **`unstable`**, see Section 2.4.3, "Safeguarding for package problems".

2.4. Advanced package management operations

2.4.1. Advanced package management operations with commandline

Here are list of other package management operations for which **`aptitude`** is too high-level or lacks required functionalities.

Table 2.13. List of advanced package management operations

command	action
`COLUMNS=120 dpkg -l <package_name_pattern>`	list status of an installed package for the bug report
`dpkg -L <package_name>`	list contents of an installed package
`dpkg -L <package_name> | egrep '/usr/share/man/man.*/.+'`	list manpages for an installed package
`dpkg -S <file_name_pattern>`	list installed packages which have matching file name
`apt-file search <file_name_pattern>`	list packages in archive which have matching file name
`apt-file list <package_name_pattern>`	list contents of matching packages in archive
`dpkg-reconfigure <package_name>`	reconfigure the exact package
`dpkg-reconfigure -p=low <package_name>`	reconfigure the exact package with the most detailed question
`configure-debian`	reconfigure packages from the full screen menu
`dpkg --audit`	audit system for partially installed packages
`dpkg --configure -a`	configure all partially installed packages
`apt-cache policy <binary_package_name>`	show available version, priority, and archive information of a binary package
`apt-cache madison <package_name>`	show available version, archive information of a package
`apt-cache showsrc <binary_package_name>`	show source package information of a binary package
`apt-get build-dep <package_name>`	install required packages to build package
`aptitude build-dep <package_name>`	install required packages to build package
`apt-get source <package_name>`	download a source (from standard archive)
`dget <URL for dsc file>`	download a source packages (from other archive)

command	action
`dpkg-source -x <package_name>_<version>-<debian_version>.dsc`	build a source tree from a set of source packages ("`*.orig.tar.gz`" and "`*.debian.tar.gz`"/"`*.diff.gz`")
`debuild binary`	build package(s) from a local source tree
`make-kpkg kernel_image`	build a kernel package from a kernel source tree
`make-kpkg --initrd kernel_image`	build a kernel package from a kernel source tree with initramfs enabled
`dpkg -i <package_name>_<version>-<debian_version>_<arch>.deb`	install a local package to the system
`debi <package_name>_<version>-<debian_version>_<arch>.dsc`	install local package(s) to the system
`dpkg --get-selections '*' >selection.txt`	save **dpkg** level package selection state information
`dpkg --set-selections <selection.txt`	set **dpkg** level package selection state information
`echo <package_name> hold | dpkg --set-selections`	set **dpkg** level package selection state for a package to **hold** (equivalent to "`aptitude hold <package_name>`")

Note

For a package with the multi-arch feature, you may need to specify the architecture name for some commands. For example, use "`dpkg -L libglib2.0-0:amd64`" to list contents of the `libglib2.0-0` package for the `amd64` architecture.

Caution

Lower level package tools such as "`dpkg -i …`" and "`debi …`" should be carefully used by the system administrator. It does not automatically take care required package dependencies. Dpkg's commandline options "`--force-all`" and similar (see dpkg(1)) are intended to be used by experts only. Using them without fully understanding their effects may break your whole system.

Please note the following.

- All system configuration and installation commands require to be run from root.
- Unlike **aptitude** which uses regex (see Section 1.6.2, “Regular expressions”), other package management commands use pattern like shell glob (see Section 1.5.6, “Shell glob”).
- apt-file(1) provided by the **apt-file** package must run "**apt-file update**" in advance.
- configure-debian(8) provided by the **configure-debian** package runs dpkg-reconfigure(8) as its backend.
- dpkg-reconfigure(8) runs package scripts using debconf(1) as its backend.
- "**apt-get build-dep**", "**apt-get source**" and "**apt-cache showsrc**" commands require "**deb-src**" entry in "**/etc/apt/sources.list**".
- dget(1), debuild(1), and debi(1) require **devscripts** package.
- See (re)packaging procedure using "**apt-get source**" in Section 2.7.13, “Porting a package to the stable system”.
- **make-kpkg** command requires the **kernel-package** package (see Section 9.9, “The kernel”).
- See Section 12.11, “Making Debian package” for general packaging.

2.4.2. Verification of installed package files

The installation of **debsums** enables verification of installed package files against MD5sum values in the "**/var/lib/dpkg/info/*.md5sums**" file with debsums(1). See Section 10.3.5, “The MD5 sum” for how MD5sum works.

Note

Because MD5sum database may be tampered by the intruder, debsums(1) is of limited use as a security tool. It is only good for checking local modifications by the administrator or damage due to media errors.

2.4.3. Safeguarding for package problems

Many users prefer to follow the **unstable** release of the Debian system for its new features and packages. This makes the system more prone to be hit by the critical package bugs.

The installation of the **`apt-listbugs`** package safeguards your system against critical bugs by checking Debian BTS automatically for critical bugs when upgrading with APT system.

The installation of the **`apt-listchanges`** package provides important news in "**`NEWS.Debian`**" when upgrading with APT system.

2.4.4. Searching on the package meta data

Although visiting Debian site http://packages.debian.org/ facilitates easy ways to search on the package meta data these days, let's look into more traditional ways.

The grep-dctrl(1), grep-status(1), and grep-available(1) commands can be used to search any file which has the general format of a Debian package control file.

The "**`dpkg -S <file_name_pattern>`**" can be used search package names which contain files with the matching name installed by **`dpkg`**. But this overlooks files created by the maintainer scripts.

If you need to make more elaborate search on the dpkg meta data, you need to run "**`grep -e regex_pattern *`**" command in the "**`/var/lib/dpkg/info/`**" directory. This makes you search words mentioned in package scripts and installation query texts.

If you wish to look up package dependency recursively, you should use apt-rdepends(8).

2.5. Debian package management internals

Let's learn how the Debian package management system works internally. This should help you to create your own solution to some package problems.

2.5.1. Archive meta data

Meta data files for each distribution are stored under "**dist/<codename>**" on each Debian mirror sites, e.g., "**http://ftp.us.debian.org/debian/**". Its archive structure can be browsed by the web browser. There are 6 types of key meta data.

Table 2.14. The content of the Debian archive meta data

file	location	content
Release	top of distribution	archive description and integrity information
Release.gpg	top of distribution	signature file for the "**Release**" file signed with the archive key
Contents-<architecture>	top of distribution	list of all files for all the packages in the pertinent archive
Release	top of each distribution/area /architecture combination	archive description used for the rule of apt_preferences(5)
Packages	top of each distribution/area /binary-architecture combination	concatenated **debian/control** for binary packages
Sources	top of each distribution/area /source combination	concatenated **debian/control** for source packages

In the recent archive, these meta data are stored as the compressed and differential files to reduce network traffic.

2.5.2. Top level "Release" file and authenticity

Tip

The top level "**Release**" file is used for signing the archive under the **secure APT** system.

Each suite of the Debian archive has a top level "**Release**" file, e.g., "**http://ftp.us.debian.org/debian/dists/unstable/Release**", as follows.

```
Origin: Debian
Label: Debian
Suite: unstable
Codename: sid
Date: Sat, 14 May 2011 08:20:50 UTC
Valid-Until: Sat, 21 May 2011 08:20:50 UTC
Architectures: alpha amd64 armel hppa hurd-i386 i386 ia64
kfreebsd-amd64 kfreebsd-i386 mips mipsel powerpc s390
sparc
Components: main contrib non-free
Description: Debian x.y Unstable - Not Released
MD5Sum:
 bdc8fa4b3f5e4a715dd0d56d176fc789 18876880 Contents-
alpha.gz
 9469a03c94b85e010d116aeeab9614c0 19441880 Contents-
amd64.gz
 3d68e206d7faa3aded660dc0996054fe 19203165 Contents-
armel.gz
...
```

Note

Here, you can find my rationale to use the "suite", and "codename" in Section 2.1.4, “Debian archive basics”. The "distribution" is used when referring to both "suite" and "codename". All archive "area" names offered by the archive are listed under "Components".

The integrity of the top level "**Release**" file is verified by cryptographic infrastructure called the secure apt.

- The cryptographic signature file "**Release.gpg**" is created from the authentic top level "**Release**" file and the secret Debian archive key.
- The public Debian archive key can be seeded into "**/etc/apt/trusted.gpg**";
  - automatically by installing the keyring with the latest **base-files** package, or
  - manually by **gpg** or **apt-key** tool with the latest public archive key posted on the ftp-master.debian.org .
- The **secure APT** system verifies the integrity of the downloaded top level "**Release**" file cryptographically by this "**Release.gpg**" file and the public Debian archive key in

"**/etc/apt/trusted.gpg**".

The integrity of all the "**Packages**" and "**Sources**" files are verified by using MD5sum values in its top level "**Release**" file. The integrity of all package files are verified by using MD5sum values in the "**Packages**" and "**Sources**" files. See debsums(1) and Section 2.4.2, "Verification of installed package files".

Since the cryptographic signature verification is a much more CPU intensive process than the MD5sum value calculation, use of MD5sum value for each package while using cryptographic signature for the top level "**Release**" file provides the good security with the performance (see Section 10.3, "Data security infrastructure").

2.5.3. Archive level "Release" files

Tip

The archive level "**Release**" files are used for the rule of apt_preferences(5).

There are archive level "**Release**" files for all archive locations specified by "**deb**" line in "**/etc/apt/sources.list**", such as "**http://ftp.us.debian.org/debian/dists/unstable/main/binary-amd64/Release**" or "**http://ftp.us.debian.org/debian/dists/sid/main/binary-amd64/Release**" as follows.

```
Archive: unstable
Origin: Debian
Label: Debian
Component: main
Architecture: amd64
```

Caution

For "**Archive:**" stanza, suite names ("**stable**", "**testing**", "**unstable**", …) are used in the Debian archive while codenames ("**dapper**", "**feisty**", "**gutsy**", "**hardy**", "**intrepid**", …) are used in the Ubuntu archive.

For some archives, such as **experimental**, and **jessie-backports**, which contain packages which should not be installed automatically, there is an extra line, e.g., "**http://ftp.us.debian.org/debian/dists/experimental/main/binary-amd64/Release**" as follows.

```
Archive: experimental
Origin: Debian
Label: Debian
NotAutomatic: yes
Component: main
Architecture: amd64
```

Please note that for normal archives without "**`NotAutomatic: yes`**", the default Pin-Priority value is 500, while for special archives with "**`NotAutomatic: yes`**", the default Pin-Priority value is 1 (see apt_preferences(5) and Section 2.7.3, "Tweaking candidate version").

2.5.4. Fetching of the meta data for the package

When APT tools, such as **`aptitude`**, **`apt-get`**, **`synaptic`**, **`apt-file`**, **`auto-apt`**, … are used, we need to update the local copies of the meta data containing the Debian archive information. These local copies have following file names corresponding to the specified **`distribution`**, **`area`**, and **`architecture`** names in the "**`/etc/apt/sources.list`**" (see Section 2.1.4, "Debian archive basics").

- "**`/var/lib/apt/lists/ftp.us.debian.org_debian_dists_<distribution>_Release`**"
- "**`/var/lib/apt/lists/ftp.us.debian.org_debian_dists_<distribution>_Release.gpg`**"
- "**`/var/lib/apt/lists/ftp.us.debian.org_debian_dists_<distribution>_<area>_binary-<architecture>_Packages`**"
- "**`/var/lib/apt/lists/ftp.us.debian.org_debian_dists_<distribution>_<area>_source_Sources`**"
- "**`/var/cache/apt/apt-file/ftp.us.debian.org_debian_dists_<distribution>_Contents-<architecture>.gz`**" (for **`apt-file`**)

First 4 types of files are shared by all the pertinent APT commands and updated from command line by "**`apt-get update`**" or "**`aptitude update`**". The "**`Packages`**" meta data are updated if there is the "**`deb`**" line in "**`/etc/apt/sources.list`**". The "**`Sources`**" meta data are updated if there is the "**`deb-src`**" line in "**`/etc/apt/sources.list`**".

The "**Packages**" and "**Sources**" meta data contain "**Filename:**" stanza pointing to the file location of the binary and source packages. Currently, these packages are located under the "**pool/**" directory tree for the improved transition over the releases.

Local copies of "**Packages**" meta data can be interactively searched with the help of **aptitude**. The specialized search command grep-dctrl(1) can search local copies of "**Packages**" and "**Sources**" meta data.

Local copy of "**Contents-<architecture>**" meta data can be updated by "**apt-file update**" and its location is different from other 4 ones. See apt-file(1). (The **auto-apt** uses different location for local copy of "**Contents-<architecture>.gz**" as default.)

2.5.5. The package state for APT

In addition to the remotely fetched meta data, the APT tool after **lenny** stores its locally generated installation state information in the "**/var/lib/apt/extended_states**" which is used by all APT tools to track all auto installed packages.

2.5.6. The package state for aptitude

In addition to the remotely fetched meta data, the **aptitude** command stores its locally generated installation state information in the "**/var/lib/aptitude/pkgstates**" which is used only by it.

2.5.7. Local copies of the fetched packages

All the remotely fetched packages via APT mechanism are stored in the "**/var/cache/apt/archives**" until they are cleaned.

This cache file cleaning policy for **aptitude** can be set under "**Options**" → "**Preferences**" and it may be forced by its menu "**Clean package cache**" or "**Clean obsolete files**" under "**Actions**".

2.5.8. Debian package file names

Debian package files have particular name structures.

Table 2.15. The name structure of Debian packages

package type	name structure
The binary package (a.k.a **deb**)	**<package-name>_<epoch>:<upstream-version>-<debian.version>-<architecture>.deb**

package type	name structure
The binary package for debian-installer (a.k.a **udeb**)	**`<package-name>_<epoch>:<upstream-version>-<debian.version>-<architecture>.udeb`**
The source package (upstream source)	**`<package-name>_<epoch>:<upstream-version>-<debian.version>.orig.tar.gz`**
The **`1.0`** source package (Debian changes)	**`<package-name>_<epoch>:<upstream-version>-<debian.version>.diff.gz`**
The **`3.0 (quilt)`** source package (Debian changes)	**`<package-name>_<epoch>:<upstream-version>-<debian.version>.debian.tar.gz`**
The source package (description)	**`<package-name>_<epoch>:<upstream-version>-<debian.version>.dsc`**

Tip

Here only the basic source package formats are described. See more on dpkg-source(1).

Table 2.16. The usable characters for each component in the Debian package names

name component	usable characters (regex)	existence
`<package-name>`	**`[a-z,A-Z,0-9,.,+,-]+`**	required
`<epoch>:`	**`[0-9]+:`**	optional
`<upstream-version>`	**`[a-z,A-Z,0-9,.,+,-,:]+`**	required
`<debian.version>`	**`[a-z,A-Z,0-9,.,+,~]+`**	optional

Note

You can check package version order by dpkg(1), e.g., "**`dpkg --compare-versions 7.0 gt 7.~pre1 ; echo $?`**" .

Note

The debian-installer (d-i) uses **udeb** as the file extension for its binary package instead of normal **deb**. An **udeb** package is a stripped down **deb** package which removes few non-essential contents such as documentation to save space while relaxing the package policy requirements. Both **deb** and **udeb** packages share the same package structure. The "**u**" stands for micro.

2.5.9. The dpkg command

dpkg(1) is the lowest level tool for the Debian package management. This is very powerful and needs to be used with care.

While installing package called "**<package_name>**", **dpkg** process it in the following order.

1. Unpack the deb file ("**ar -x**" equivalent)
2. Execute "**<package_name>.preinst**" using debconf(1)
3. Install the package content to the system ("**tar -x**" equivalent)
4. Execute "**<package_name>.postinst**" using debconf(1)

The **debconf** system provides standardized user interaction with I18N and L10N (*Chapter 8, I18N and L10N*) supports.

Table 2.17. The notable files created by dpkg

file	description of contents
/var/lib/dpkg/info/<package_name>.conffiles	list of configuration files. (user modifiable)
/var/lib/dpkg/info/<package_name>.list	list of files and directories installed by the package
/var/lib/dpkg/info/<package_name>.md5sums	list of MD5 hash values for files installed by the package
/var/lib/dpkg/info/<package_name>.preinst	package script to be run before the package installation
/var/lib/dpkg/info/<package_name>.postinst	package script to be run after the package installation
/var/lib/dpkg/info/<package_name>.prerm	package script to be run before the package removal
/var/lib/dpkg/info/<package_name>.postrm	package script to be run after the package removal
/var/lib/dpkg/info/<package_name>.config	package script for **debconf** system

file	description of contents
/var/lib/dpkg/alternatives /<package_name>	the alternative information used by the **update-alternatives** command
/var/lib/dpkg/available	the availability information for all the package
/var/lib/dpkg/diversions	the diversions information used by dpkg(1) and set by dpkg-divert(8)
/var/lib/dpkg/statoverride	the stat override information used by dpkg(1) and set by dpkg-statoverride(8)
/var/lib/dpkg/status	the status information for all the packages
/var/lib/dpkg/status-old	the first-generation backup of the "**var/lib/dpkg/status**" file
/var/backups/dpkg.status*	the second-generation backup and older ones of the "**var/lib /dpkg/status**" file

The "**status**" file is also used by the tools such as dpkg(1), "**dselect update**" and "**apt-get -u dselect-upgrade**".

The specialized search command grep-dctrl(1) can search the local copies of "**status**" and "**available**" meta data.

Tip

In the debian-installer environment, the **udpkg** command is used to open **udeb** packages. The **udpkg** command is a stripped down version of the **dpkg** command.

2.5.10. The update-alternatives command

The Debian system has mechanism to install somewhat overlapping programs peacefully using update-alternatives(8). For example, you can make the **vi** command select to run **vim** while installing both **vim** and **nvi** packages.

```
$ ls -l $(type -p vi)
lrwxrwxrwx 1 root root 20 2007-03-24 19:05 /usr/bin/vi ->
/etc/alternatives/vi
$ sudo update-alternatives --display vi
...
$ sudo update-alternatives --config vi
  Selection    Command
 ----------------------------------------------
      1        /usr/bin/vim
*+    2        /usr/bin/nvi

Enter to keep the default[*], or type selection number: 1
```

The Debian alternatives system keeps its selection as symlinks in "**`/etc/alternatives/`**". The selection process uses corresponding file in "**`/var/lib/dpkg/alternatives/`**".

2.5.11. The dpkg-statoverride command

Stat overrides provided by the dpkg-statoverride(8) command are a way to tell dpkg(1) to use a different owner or mode for a **file** when a package is installed. If "**`--update`**" is specified and file exists, it is immediately set to the new owner and mode.

Caution

The direct alteration of owner or mode for a **file** owned by the package using **`chmod`** or **`chown`** commands by the system administrator is reset by the next upgrade of the package.

Note

I use the word **file** here, but in reality this can be any filesystem object that **`dpkg`** handles, including directories, devices, etc.

2.5.12. The dpkg-divert command

File **diversions** provided by the dpkg-divert(8) command are a way of forcing dpkg(1) not to install a file into its default location, but to a **diverted** location. The use of **`dpkg-divert`** is meant for the package maintenance scripts. Its casual use by the system administrator is deprecated.

2.6. Recovery from a broken system

When running **`unstable`** system, the administrator is expected to recover from broken package management situation.

Caution

Some methods described here are high risk actions. You have been warned!

2.6.1. Incompatibility with old user configuration

If a desktop GUI program experienced instability after significant upstream version upgrade, you should suspect interferences with old local configuration files created by it. If it is stable under a newly created user account, this hypothesis is confirmed. (This is a bug of packaging and usually avoided by the packager.)

To recover stability, you should move corresponding local configuration files and restart the GUI program. You may need to read old configuration file contents to recover configuration information later. (Do not erase them too quickly.)

2.6.2. Different packages with overlapped files

Archive level package management systems, such as aptitude(8) or apt-get(1), do not even try to install packages with overlapped files using package dependencies (see Section 2.1.6, “Package dependencies”).

Errors by the package maintainer or deployment of inconsistently mixed source of archives (see Section 2.7.2, “Packages from mixed source of archives”) by the system administrator may create a situation with incorrectly defined package dependencies. When you install a package with overlapped files using aptitude(8) or apt-get(1) under such a situation, dpkg(1) which unpacks package ensures to return error to the calling program without overwriting existing files.

Caution

The use of third party packages introduces significant system risks via maintainer scripts which are run with root privilege and can do anything to your system. The dpkg(1) command only protects against overwriting by the unpacking.

You can work around such broken installation by removing the old offending package, **`<old-package>`**, first.

```
$ sudo dpkg -P <old-package>
```

2.6.3. Fixing broken package script

When a command in the package script returns error for some reason and the script exits with error, the package management system aborts their action and ends up with partially installed packages. When a package contains bugs in its removal scripts, the package may become impossible to remove and quite nasty.

For the package script problem of "**`<package_name>`**", you should look into following package scripts.

- "**`/var/lib/dpkg/info/<package_name>.preinst`**"
- "**`/var/lib/dpkg/info/<package_name>.postinst`**"
- "**`/var/lib/dpkg/info/<package_name>.prerm`**"
- "**`/var/lib/dpkg/info/<package_name>.postrm`**"

Edit the offending package script from the root using following techniques.

- disable the offending line by preceding "**`#`**"
- force to return success by appending the offending line with "**`|| true`**"

Configure all partially installed packages with the following command.

```
# dpkg --configure -a
```

2.6.4. Rescue with the dpkg command

Since **dpkg** is very low level package tool, it can function under the very bad situation such as unbootable system without network connection. Let's assume **`foo`** package was broken and needs to be replaced.

You may still find cached copies of older bug free version of **`foo`** package in the package cache directory: "**`/var/cache/apt/archives/`**". (If not, you can download it from archive of http://snapshot.debian.net/ or copy it from package cache of a functioning machine.)

If you can boot the system, you may install it by the following command.

```
# dpkg -i /path/to/foo_<old_version>_<arch>.deb
```

Tip

If system breakage is minor, you may alternatively downgrade the whole system as in Section 2.7.10, “Emergency downgrading” using the higher level APT system.

If your system is unbootable from hard disk, you should seek other ways to boot it.

1. Boot the system using the debian-installer CD in rescue mode.
2. Mount the unbootable system on the hard disk to "**/target**".
3. Install older version of **foo** package by the following.

```
# dpkg --root /target -i /path/to
/foo_<old_version>_<arch>.deb
```

This example works even if the **dpkg** command on the hard disk is broken.

Tip

Any GNU/Linux system started by another system on hard disk, live GNU/Linux CD, bootable USB-key drive, or netboot can be used similarly to rescue broken system.

If attempting to install a package this way fails due to some dependency violations and you really need to do this as the last resort, you can override dependency using **dpkg**'s "**--ignore-depends**", "**--force-depends**" and other options. If you do this, you need to make serious effort to restore proper dependency later. See dpkg(8) for details.

Note

If your system is seriously broken, you should make a full backup of system to a safe place (see Section 10.2, “Backup and recovery”) and should perform a clean installation. This is less time consuming and produces better results in the end.

2.6.5. Recovering package selection data

If "**/var/lib/dpkg/status**" becomes corrupt for any reason, the Debian system loses package selection data and suffers severely. Look for the old "**/var/lib/dpkg/status**" file at "**/var/lib/dpkg/status-old**" or "**/var/backups/dpkg.status.***".

Keeping "**/var/backups/**" in a separate partition may be a good idea since this directory contains lots of important system data.

For serious breakage, I recommend to make fresh re-install after making backup of the system. Even if everything in "**/var/**" is gone, you can still recover some information from directories in "**/usr/share/doc/**" to guide your new installation.

Reinstall minimal (desktop) system.

```
# mkdir -p /path/to/old/system
```

Mount old system at "**/path/to/old/system/**".

```
# cd /path/to/old/system/usr/share/doc
# ls -1 >~/ls1.txt
# cd /usr/share/doc
# ls -1 >>~/ls1.txt
# cd
# sort ls1.txt | uniq | less
```

Then you are presented with package names to install. (There may be some non-package names such as "**texmf**".)

2.7. Tips for the package management

2.7.1. How to pick Debian packages

You can seek packages which satisfy your needs with **aptitude** from the package description or from the list under "Tasks".

When you encounter more than 2 similar packages and wonder which one to install without "trial and error" efforts, you should use some **common sense**. I consider following points are good indications of preferred packages.

- Essential: yes > no
- Area: main > contrib > non-free
- Priority: required > important > standard > optional > extra

- Tasks: packages listed in tasks such as "Desktop environment"
- Packages selected by the dependency package (e.g., **`python2.4`** by **`python`**)
- Popcon: higher in the vote and install number
- Changelog: regular updates by the maintainer
- BTS: No RC bugs (no critical, no grave, and no serious bugs)
- BTS: responsive maintainer to bug reports
- BTS: higher number of the recently fixed bugs
- BTS: lower number of remaining non-wishlist bugs

Debian being a volunteer project with distributed development model, its archive contains many packages with different focus and quality. You must make your own decision what to do with them.

2.7.2. Packages from mixed source of archives

Caution

Installing packages from mixed source of archives is not supported by the official Debian distribution except for officially supported particular combinations of archives such as **`stable`** with security updates and jessie-updates.

Here is an example of operations to include specific newer upstream version packages found in **`unstable`** while tracking **`testing`** for single occasion.

1. Change the "**`/etc/apt/sources.list`**" file temporarily to single "**`unstable`**" entry.
2. Run "**`aptitude update`**".
3. Run "**`aptitude install <package-name>`**".
4. Recover the original "**`/etc/apt/sources.list`**" file for **`testing`**.
5. Run "**`aptitude update`**".

You do not create the "**`/etc/apt/preferences`**" file nor need to worry about apt-pinning with this manual approach. But this is very cumbersome.

Caution

When using mixed source of archives, you must ensure compatibility of packages by yourself since the Debian does not guarantee it. If package incompatibility exists, you may break system. You must be able to judge these technical requirements. The use of mixed source of random archives is completely optional operation and its use is not something I encourage you to use.

General rules for installing packages from different archives are the following.

- Non-binary packages ("**`Architecture: all`**") are **safer** to install.
  - documentation packages: no special requirements
  - interpreter program packages: compatible interpreter must be available
- Binary packages (non "**`Architecture: all`**") usually face many road blocks and are **unsafe** to install.
  - library version compatibility (including "**`libc`**")
  - related utility program version compatibility
  - Kernel ABI compatibility
  - C++ ABI compatibility
  - …

Note

In order to make a package to be **safer** to install, some commercial non-free binary program packages may be provided with completely statically linked libraries. You should still check ABI compatibility issues etc. for them.

Note

Except to avoid broken package for a short term, installing binary packages from officially unsupported archives is generally bad idea. This is true even if you use apt-pinning (see Section 2.7.3, “Tweaking candidate version”). You should consider chroot or similar techniques (see Section 9.10, “Virtualized system”) to run programs from different archives.

2.7.3. Tweaking candidate version

Without the "**`/etc/apt/preferences`**" file, APT system choses the latest available version as the **candidate version** using the version string. This is the normal state and most recommended usage of APT system. All officially supported combinations of archives do not require the "**`/etc/apt/preferences`**" file since some archives which should not be used as the automatic source of upgrades are marked as **NotAutomatic** and dealt properly.

Tip

The version string comparison rule can be verified with, e.g., "**`dpkg --compare-versions ver1.1 gt ver1.1~1; echo $?`**" (see dpkg(1)).

When you install packages from mixed source of archives (see Section 2.7.2, “Packages from mixed source of archives”) regularly, you can automate these complicated operations by creating the "**`/etc/apt/preferences`**" file with proper entries and tweaking the package selection rule for **candidate version** as described in apt_preferences(5). This is called **apt-pinning**.

Warning

Use of apt-pinning by a novice user is sure call for major troubles. You must avoid using apt-pinning except when you absolutely need it.

Caution

When using apt-pinning, you must ensure compatibility of packages by yourself since the Debian does not guarantee it. The apt-pinning is completely optional operation and its use is not something I encourage you to use.

Caution

Archive level Release files (see Section 2.5.3, “Archive level "Release" files”) are used for the rule of apt_preferences(5). Thus apt-pinning works only with "suite" name for normal Debian archives and security Debian archives. (This is different from Ubuntu archives.) For example, you can do "**`Pin: release a=unstable`**" but can not do "**`Pin: release a=sid`**" in the "**`/etc/apt/preferences`**" file.

Caution

When you use non-Debian archive as a part of apt-pinning, you should check what they are intended for and also check their credibility. For example, Ubuntu and Debian are not meant to be mixed.

Note

Even if you do not create the "**`/etc/apt/preferences`**" file, you can do fairly complex system operations (see Section 2.6.4, “Rescue with the dpkg command” and Section 2.7.2, “Packages from mixed source of archives”) without apt-pinning.

Here is a simplified explanation of **apt-pinning** technique.

The APT system choses the highest Pin-Priority **upgrading** package from available package sources defined in the "**`/etc/apt/sources.list`**" file as the **candidate version** package. If the Pin-Priority of the package is larger than 1000, this version restriction for **upgrading** is dropped to enable downgrading (see Section 2.7.10, “Emergency downgrading”).

Pin-Priority value of each package is defined by "Pin-Priority" entries in the "**`/etc/apt/preferences`**" file or uses its default value.

Table 2.18. List of notable Pin-Priority values for apt-pinning technique.

Pin-Priority	apt-pinning effects to the package

Pin-Priority	apt-pinning effects to the package
1001	install the package even if this constitutes a downgrade of the package
990	used as the default for the **target release** archive
500	used as the default for the **normal** archive
100	used as the default for the **NotAutomatic** and **ButAutomaticUpgrades** archive
100	used for the **installed** package
1	used as the default for the **NotAutomatic** archive
-1	**never install** the package even if recommended

The **target release** archive can be set by several methods.

- "`/etc/apt/apt.conf`" configuration file with "`APT::Default-Release "stable";`" line
- command line option, e.g., "`apt-get install -t testing some-package`"

The **NotAutomatic** and **ButAutomaticUpgrades** archive is set by archive server having its archive level Release file (see Section 2.5.3, “Archive level "Release" files”) containing both "`NotAutomatic: yes`" and "`ButAutomaticUpgrades: yes`". The **NotAutomatic** archive is set by archive server having its archive level Release file containing only "`NotAutomatic: yes`".

The **apt-pinning situation** of <package> from multiple archive sources is displayed by "`apt-cache policy <package>`".

- A line started with "`Package pin:`" lists the package version of **pin** if association just with <package> is defined, e.g., "`Package pin: 0.190`".
- No line with "`Package pin:`" exists if no association just with <package> is defined.
- The Pin-Priority value associated just with <package> is listed right side of all version strings, e.g., "`0.181 700`".
- "`0`" is listed right side of all version strings if no association just with <package> is defined, e.g., "`0.181 0`".
- The Pin-Priority values of archives (defined as "`Package: *`" in the "`/etc/apt/preferences`" file) are listed left side of all archive paths, e.g., "`100 http://ftp.XX.debian.org /debian/ jessie-backports/main Packages`".

2.7.4. Updates and Backports

There are jessie-updates and backports.debian.org archives which provide updgrade packages for **stable** (**jessie**).

In order to use these archives, you list all required archives in the "**/etc/apt/sources.list**" file as the following.

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

There is no need to set Pin-Priority value explicitly in the "**/etc/apt/preferences**" file. When newer packages become available, the default configuration provides most reasonable upgrades (see Section 2.5.3, “Archive level "Release" files”).

- All installed older packages are upgraded to newer ones from **jessie-updates**.
- Only manually installed older packages from **jessie-backports** are upgraded to newer ones from **jessie-backports**.

Whenever you wish to install a package named "**<package-name>**" with its dependency from **jessie-backports** archive manually, you use following command while switching target release with "**-t**" option.

```
$ sudo apt-get install -t jessie-backports <package-name>
```

2.7.5. Blocking packages installed by "Recommends"

If you wish not to pull in particular packages automatically by "Recommends", you must create the "**/etc/apt/preferences**" file and explicitly list all those packages at the top of it as the following.

```
Package: <package-1>
Pin: version *
Pin-Priority: -1

Package: <package-2>
Pin: version *
Pin-Priority: -1
```

2.7.6. Tracking `testing` with some packages from `unstable`

Here is an example of **apt-pinning** technique to include specific newer upstream version packages found in **`unstable`** regularly upgraded while tracking **`testing`**. You list all required archives in the "**`/etc/apt/sources.list`**" file as the following.

```
deb http://ftp.us.debian.org/debian/ testing main contrib
non-free
deb http://ftp.us.debian.org/debian/ unstable main
contrib non-free
deb http://security.debian.org/ testing/updates main
contrib
```

Set the "**`/etc/apt/preferences`**" file as as the following.

```
Package: *
Pin: release a=unstable
Pin-Priority: 100
```

When you wish to install a package named "**`<package-name>`**" with its dependencies from **`unstable`** archive under this configuration, you issue the following command which switches target release with "**`-t`**" option (Pin-Priority of **`unstable`** becomes 990).

```
$ sudo apt-get install -t unstable <package-name>
```

With this configuration, usual execution of "**`apt-get upgrade`**" and "**`apt-get dist-upgrade`**" (or "**`aptitude safe-upgrade`**" and "**`aptitude full-upgrade`**") upgrades packages which were installed from **`testing`** archive using current **`testing`** archive and packages which were installed from **`unstable`** archive using current **`unstable`** archive.

Caution

Be careful not to remove "**`testing`**" entry from the

"**/etc/apt/sources.list**" file. Without "**testing**" entry in it, APT system upgrades packages using newer **unstable** archive.

Tip

I usually edit the "**/etc/apt/sources.list**" file to comment out "**unstable**" archive entry right after above operation. This avoids slow update process of having too many entries in the "**/etc/apt/sources.list**" file although this prevents upgrading packages which were installed from **unstable** archive using current **unstable** archive.

Tip

If "**Pin-Priority: 1**" is used instead of "**Pin-Priority: 100**" in the "**/etc/apt/preferences**" file, already installed packages having Pin-Priority value of 100 are not upgraded by **unstable** archive even if "**testing**" entry in the "**/etc/apt/sources.list**" file is removed.

If you wish to track particular packages in **unstable** automatically without initial "**-t unstable**" installation, you must create the "**/etc/apt/preferences**" file and explicitly list all those packages at the top of it as the following.

```
Package: <package-1>
Pin: release a=unstable
Pin-Priority: 700

Package: <package-2>
Pin: release a=unstable
Pin-Priority: 700
```

These set Pin-Priority value for each specific package. For example, in order to track the latest **unstable** version of this "Debian Reference" in English, you should have following entries in the "**/etc/apt/preferences**" file.

```
Package: debian-reference-en
Pin: release a=unstable
Pin-Priority: 700

Package: debian-reference-common
Pin: release a=unstable
Pin-Priority: 700
```

Tip

This apt-pinning technique is valid even when you are tracking **stable** archive. Documentation packages have been always safe to install from **unstable** archive in my experience, so far.

2.7.7. Tracking unstable with some packages from experimental

Here is another example of **apt-pinning** technique to include specific newer upstream version packages found in **experimental** while tracking **unstable**. You list all required archives in the "**/etc/apt/sources.list**" file as the following.

```
deb http://ftp.us.debian.org/debian/ unstable main
contrib non-free
deb http://ftp.us.debian.org/debian/ experimental main
contrib non-free
deb http://security.debian.org/ testing/updates main
contrib
```

The default Pin-Priority value for **experimental** archive is always 1 (<<100) since it is **NotAutomatic** archive (see Section 2.5.3, “Archive level "Release" files”). There is no need to set Pin-Priority value explicitly in the "**/etc/apt/preferences**" file just to use **experimental** archive unless you wish to track particular packages in it automatically for next upgrading.

2.7.8. Automatic download and upgrade of packages

The **apt** package comes with its own cron script "**/etc/cron.daily/apt**" to support the automatic download of packages. This script can be enhanced to perform the automatic upgrade of packages by installing the **unattended-upgrades** package. These can be customized by parameters in "**/etc/apt/apt.conf.d/02backup**" and "**/etc/apt/apt.conf.d/50unattended-upgrades**" as described in "**/usr/share**

/doc/unattended-upgrades/README".

The **unattended-upgrades** package is mainly intended for the security upgrade for the **stable** system. If the risk of breaking an existing **stable** system by the automatic upgrade is smaller than that of the system broken by the intruder using its security hole which has been closed by the security update, you should consider using this automatic upgrade with configuration parameters as the following.

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::Unattended-Upgrade "1";
```

If you are running an **unstable** system, you do not want to use the automatic upgrade since it certainly breaks system some day. Even for such **unstable** case, you may still want to download packages in advance to save time for the interactive upgrade with configuration parameters as the following.

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::Unattended-Upgrade "0";
```

2.7.9. Limiting download bandwidth for APT

If you want to limit the download bandwidth for APT to e.g. 800Kib/sec (=100kiB/sec), you should configure APT with its configuration parameter as the following.

```
APT::Acquire::http::Dl-Limit "800";
```

2.7.10. Emergency downgrading

Caution

Downgrading is not officially supported by the Debian by design. It should be done only as a part of emergency recovery process. Despite of this situation, it is known to work well in many incidents. For critical systems, you should backup all important data on the system after the recovery operation and re-install the new system from the scratch.

You may be lucky to downgrade from newer archive to older archive to recover from broken system upgrade by manipulating **candidate version** (see Section 2.7.3, “Tweaking candidate version”). This is lazy alternative to tedious actions of many "**dpkg -i <broken-package>_<old-version>.deb**" commands (see Section 2.6.4, “Rescue with the dpkg command”).

Search lines in the "**/etc/apt/sources.list**" file tracking **unstable** as the following.

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

Replace it with the following to track **testing**.

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

Set the "**/etc/apt/preferences**" file as the following.

```
Package: *
Pin: release a=testing
Pin-Priority: 1010
```

Run "**apt-get update; apt-get dist-upgrade**" to force downgrading of packages across the system.

Remove this special "**/etc/apt/preferences**" file after this emergency downgrading.

Tip

It is a good idea to remove (not purge!) as much packages to minimize dependency problems. You may need to manually remove and install some packages to get system downgraded. Linux kernel, bootloader, udev, PAM, APT, and networking related packages and their configuration files require special attention.

2.7.11. Who uploaded the package?

Although the maintainer name listed in "**/var/lib/dpkg/available**" and "**/usr/share/doc/package_name/changelog**" provide some information on "who is behind the packaging activity", the actual uploader of the package is somewhat obscure. who-uploads(1) in the **devscripts** package identifies the actual uploader of Debian source packages.

2.7.12. The equivs package

If you are to compile a program from source to replace the Debian package, it is best to make it into a real local debianized package (**`*.deb`**) and use private archive.

If you chose to compile a program from source and to install them under "**`/usr/local`**" instead, you may need to use **`equivs`** as a last resort to satisfy the missing package dependency.

```
Package: equivs
Priority: extra
Section: admin
Description: Circumventing Debian package dependencies
 This is a dummy package which can be used to create
Debian
 packages, which only contain dependency information.
```

2.7.13. Porting a package to the stable system

For partial upgrades of the **`stable`** system, rebuilding a package within its environment using the source package is desirable. This avoids massive package upgrades due to their dependencies.

Add the following entries to the "**`/etc/apt/sources.list`**" of a **`stable`** system.

```
deb-src http://http.us.debian.org/debian unstable  main
contrib non-free
```

Install required packages for the compilation and download the source package as the following.

```
# apt-get update
# apt-get dist-upgrade
# apt-get install fakeroot devscripts build-essential
$ apt-get build-dep foo
$ apt-get source foo
$ cd foo*
```

Update some tool chain packages such as **`dpkg`**, and **`debhelper`** from the backport packages if they are required for the backporting.

Execute the following.

```
$ dch -i
```

Bump package version, e.g. one appended with "**+bp1**" in "**debian/changelog**"

Build packages and install them to the system as the following.

```
$ debuild
$ cd ..
# debi foo*.changes
```

2.7.14. Proxy server for APT

Since mirroring whole subsection of Debian archive wastes disk space and network bandwidth, deployment of a local proxy server for APT is desirable consideration when you administer many systems on LAN. APT can be configure to use generic web (http) proxy servers such as **squid** (see Section 6.10, “Other network application servers”) as described in apt.conf(5) and in "**/usr/share/doc/apt/examples/configure-index.gz**". The "**$http_proxy**" environment variable can be used to override proxy server setting in the "**/etc/apt/apt.conf**" file.

There are proxy tools specially for Debian archive. You should check BTS before using them.

Table 2.19. List of the proxy tools specially for Debian archive

package	popcon	size	description
approx	V:1, I:1	4263	caching proxy server for Debian archive files (compiled OCaml program)
apt-cacher	V:1, I:2	326	Caching proxy for Debian package and source files (Perl program)
apt-cacher-ng	V:3, I:5	1022	Caching proxy for distribution of software packages (compiled C++ program)

Caution

When Debian reorganizes its archive structure, these specialized proxy tools tend to require code rewrites by the package maintainer and may not be functional for a while. On the other hand, generic web (http) proxy servers are more robust and easier to cope with such changes.

2.7.15. Small public package archive

Here is an example for creating a small public package archive compatible with the modern **secure APT** system (see Section 2.5.2, “Top level "Release" file and authenticity”). Let's assume few things.

- Account name: "**foo**"
- Host name: "**www.example.com**"
- Required packages: **apt-utils**, **gnupg**, and other packages
- URL: "**http://www.example.com/~foo/**" (→ "**/home/foo/public_html/index.html**")
- Architecture of packages: "**amd64**"

Create an APT archive key of Foo on your server system as the following.

```
$ ssh foo@www.example.com
$ gpg --gen-key
...
$ gpg -K
...
sec   1024D/3A3CB5A6 2008-08-14
uid                  Foo (ARCHIVE KEY)
<foo@www.example.com>
ssb   2048g/6856F4A7 2008-08-14
$ gpg --export -a 3A3CB5A6 >foo.public.key
```

Publish the archive key file "**foo.public.key**" with the key ID "**3A3CB5A6**" for Foo

Create an archive tree called "Origin: Foo" as the following.

```
$ umask 022
$ mkdir -p ~/public_html/debian/pool/main
$ mkdir -p ~/public_html/debian/dists/unstable
/main/binary-amd64
$ mkdir -p ~/public_html/debian/dists/unstable/main/source
$ cd ~/public_html/debian
$ cat > dists/unstable/main/binary-amd64/Release << EOF
Archive: unstable
Version: 4.0
Component: main
Origin: Foo
Label: Foo
Architecture: amd64
EOF
$ cat > dists/unstable/main/source/Release << EOF
Archive: unstable
Version: 4.0
Component: main
Origin: Foo
Label: Foo
Architecture: source
EOF
$ cat >aptftp.conf <<EOF
APT::FTPArchive::Release {
  Origin "Foo";
  Label "Foo";
  Suite "unstable";
  Codename "sid";
  Architectures "amd64";
  Components "main";
  Description "Public archive for Foo";
};
EOF
$ cat >aptgenerate.conf <<EOF
Dir::ArchiveDir ".";
Dir::CacheDir ".";
TreeDefault::Directory "pool/";
TreeDefault::SrcDirectory "pool/";
Default::Packages::Extensions ".deb";
Default::Packages::Compress ". gzip bzip2";
Default::Sources::Compress "gzip bzip2";
Default::Contents::Compress "gzip bzip2";

BinDirectory "dists/unstable/main/binary-amd64" {
  Packages "dists/unstable/main/binary-amd64/Packages";
  Contents "dists/unstable/Contents-amd64";
  SrcPackages "dists/unstable/main/source/Sources";
};
```

```
Tree "dists/unstable" {
  Sections "main";
  Architectures "amd64 source";
};
EOF
```

You can automate repetitive updates of APT archive contents on your server system by configuring **dupload**.

Place all package files into "**~foo/public_html/debian/pool/main/**" by executing "**dupload -t foo changes_file**" in client while having "**~/.dupload.conf**" containing the following.

```
$cfg{'foo'} = {
  fqdn => "www.example.com",
  method => "scpb",
  incoming => "/home/foo/public_html/debian/pool/main",
  # The dinstall on ftp-master sends emails itself
  dinstall_runs => 1,
};

$cfg{'foo'}{postupload}{'changes'} = "
  echo 'cd public_html/debian ;
  apt-ftparchive generate -c=aptftp.conf aptgenerate.conf;
  apt-ftparchive release -c=aptftp.conf dists/unstable
>dists/unstable/Release ;
  rm -f dists/unstable/Release.gpg ;
  gpg -u 3A3CB5A6 -bao dists/unstable/Release.gpg
dists/unstable/Release'|
  ssh foo@www.example.com  2>/dev/null ;
  echo 'Package archive created!'";
```

The **postupload** hook script initiated by dupload(1) creates updated archive files for each upload.

You can add this small public archive to the apt-line of your client system by the following.

```
$ sudo bash
# echo "deb http://www.example.com/~foo/debian/ unstable
main" \
   >> /etc/apt/sources.list
# apt-key add foo.public.key
```

Tip

If the archive is located on the local filesystem, you can use "**deb file:///home/foo/debian/ …**" instead.

2.7.16. Recording and copying system configuration

You can make a local copy of the package and debconf selection states by the following.

```
# dpkg --get-selections '*' > selection.dpkg
# debconf-get-selections    > selection.debconf
```

Here, "*****" makes "**selection.dpkg**" to include package entries for "purge" too.

You can transfer these 2 files to another computer, and install there with the following.

```
# dselect update
# debconf-set-selections < myselection.debconf
# dpkg --set-selections  < myselection.dpkg
# apt-get -u dselect-upgrade    # or dselect install
```

If you are thinking about managing many servers in a cluster with practically the same configuration, you should consider to use specialized package such as **fai** to manage the whole system.

2.7.17. Converting or installing an alien binary package

alien(1) enables the conversion of binary packages provided in Red Hat **rpm**, Stampede **slp**, Slackware **tgz**, and Solaris **pkg** file formats into a Debian **deb** package. If you want to use a package from another Linux distribution than the one you have installed on your system, you can use **alien** to convert it from your preferred package format and install it. **alien** also supports LSB packages.

Warning

alien(1) should not be used to replace essential system packages, such as **sysvinit**, **libc6**, **libpam-modules**, etc. Practically, alien(1) should only be used for **non-free** binary-only packages which are LSB compliant or statically linked. For free softwares, you should use their source packages to make real Debian packages.

2.7.18. Extracting package without dpkg

The "**dpkg*.deb**" package contents can be extracted without using dpkg(1) on any Unix-like environment using standard ar(1) and tar(1).

```
# ar x /path/to/dpkg_<version>_<arch>.deb
# ls
total 24
-rw-r--r-- 1 bozo bozo  1320 2007-05-07 00:11
control.tar.gz
-rw-r--r-- 1 bozo bozo 12837 2007-05-07 00:11 data.tar.gz
-rw-r--r-- 1 bozo bozo     4 2007-05-07 00:11 debian-
binary
# mkdir control
# mkdir data
# tar xvzf control.tar.gz -C control
# tar xvzf data.tar.gz -C data
```

The other "***.deb**" package contents can be extracted by the dpkg-deb(1) command obtained from the "**dpkg*.deb**" package as above; or using standard ar(1) and newer GNU tar(1) with the xz(1) decompression support similarly as above.

You can also browse package content using the **mc** command.

2.7.19. More readings for the package management

You can learn more on the package management from following documentations.

- Primary documentations on the package management:
  - aptitude(8), dpkg(1), tasksel(8), apt-get(8), apt-config(8), apt-key(8), sources.list(5), apt.conf(5), and apt_preferences(5);
  - "**/usr/share/doc/apt-doc/guide.html/index.html**" and "**/usr/share/doc/apt-doc/offline.html/index.html**" from the **apt-doc** package; and
  - "**/usr/share/doc/aptitude/html/en/index.html**" from the **aptitude-doc-en** package.

- Official and detailed documentations on the Debian archive:
    - "Debian Policy Manual Chapter 2 - The Debian Archive",
    - "Debian Developer's Reference, Chapter 4 - Resources for Debian Developers 4.6 The Debian archive", and
    - "The Debian GNU/Linux FAQ, Chapter 6 - The Debian FTP archives".
- Tutorial for building of a Debian package for Debian users:
    - "Debian New Maintainers' Guide".

Chapter 1. GNU/Linux tutorials Chapter 3. The system initialization