Chapter 4

Debian releases and archives

> Look, this is Debian. They don't release things until you have to fire rockets at the thing to stop it from working.
> — MrNemesis on Slashdot

Probably the two most common facts to hear about Debian is that it is hopelessly outdated and stable as a rock. In the Debian world, these two traits are actually one and the same, and it would be difficult to argue against either one. Already at the time of release of a new Debian version, the software it contains is usually not current. In the world of free software, where improvements, fixes, and new features are added to projects on a daily basis, this may have negative consequences. However, in productive environments, new features and improvements can often backfire. Thus, the Debian ***stable*** release focuses on software stability, rather than trying to surf the cutting edge with possibly buggy and untested software. Only security-related bug fixes are allowed in.

Debian *stable* is not the only Debian release. In addition, the archive provides two other ones: *testing* and *unstable*. While these are not really released in the way that *stable* is frozen and termed official, they are publicly available and in use by many people[1]. Before inspecting each of the Debian releases in turn, it is important to define what stability means with respect to Debian, or what instability the name *unstable* is trying to coin.

In the context of a software and distribution archives, stability can refer to one of three aspects:

Software runtime stability
: Most commonly, the term stability is used to refer to the reliability and robustness of software contained in the archive. Stable software is mature software with an extremely low number of bugs (there is no such thing as bug-free software). Runtime stability is what keeps users happy.

Software feature stability
: Stability may also refer to the feature set provided by a software. In this definition, stable software does not introduce drastic changes or radical new features from one release to the next. Administrators appreciate feature stability because it allows them to fix bugs with newer versions without risking unwanted changes to the behaviour.

Archive stability
: A software distribution archive can be termed stable if the set of packages or pieces of software it provides does not fluctuate. Furthermore, archive stability also includes the relationships among the contained packages. A stable software distribution archive does not grow or shrink in size, and updates only affect individual packages, not larger parts of the archive. Archive stability allows for official releases to happen.

The canonical Debian release names "*stable*" and "*unstable*" refer to the second and third definition of stability, although the first sense of stability is implicit to a certain extent. While Debian developers upload new packages to *unstable* on a daily basis, and drastic changes to the packages and pieces of software they provide are possible (albeit rare), once a Debian release becomes *stable*, no packages will be added or removed to or from the set. Furthermore, as a function of Debian's security update policy (see chapter 7), updates to individual packages are limited to security-grade bug fixes and must not affect the feature set (or fix non-security) bugs. Fixes to inconvenience bugs, new versions, and new software as a whole are held back until the next Debian release is promoted to Debian *stable*.

The first of the above three aspects of stability results from the Debian release cycle, which we shall unfold in an instant. For a package to be included in *stable*,

[1] The term "release" is frequently used to refer to self-contained archives in the domain of software development.

it must be free of critical bugs and have received several months worth of testing. While the runtime stability of a software is purely in the hands of the upstream author, the rigorous testing and quality control applied throughout the Debian release cycle ensures an acceptable level of runtime stability across all programmes included in Debian *stable*.

The three archives, *stable*, *testing*, and *unstable* are naturally related. A normal package traverses all three (in reverse order). To help understand the process, it is useful to look at a package life cycle, from the moment the maintainer finishes and uploads it until it is immortalised on the media of an official Debian release.

4.1 Structure of the Debian archive

First, let us identify the different directory hierarchies and their purpose in the Debian archive. The archive is split into two main hierarchies, rooted at /pool and /dists. All the packages and source files reside under /pool, whereas the index files are located in /dists. This separation was instituted when *testing* was introduced (which happened between the release of *potato* and *woody*). Some packages have equivalent versions in multiple releases and it is less of a waste of space to store packages in a common pool and reference them individually from the release indices.

An excerpt of the structure of a Debian mirror is shown in the tree diagram in figure 4.1.

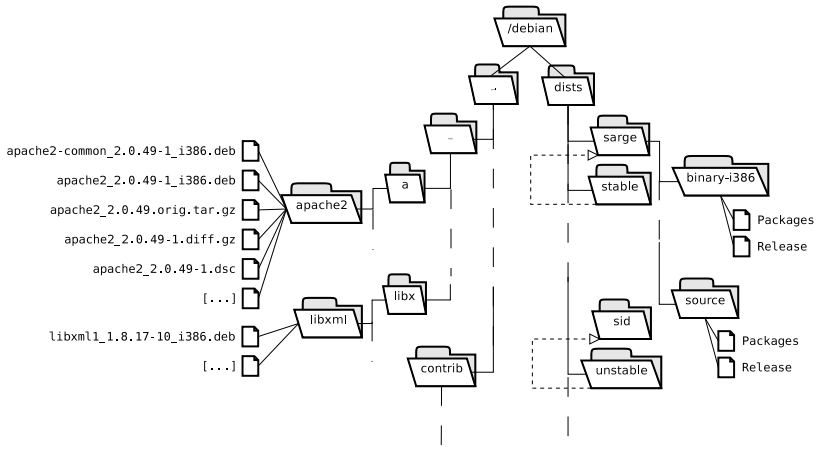

Figure 4.1: A tree diagram showing excerpts of the Debian archive

4.1.1 The package pool

The **/pool** hierarchy is divided up into three sections: *main*, *contrib*, and *non-free*. The hierarchy is further subdivided at the next level into subtrees according to the first letter of the contained packages. Within each single-letter directory there are directories for each Debian source package. For instance, files related to **apache2** are located in **/pool/main/a/apache2**. An exception is made for libraries, which sort into different subtrees, rooted at **lib?** (where the question mark is a wildcard). For example, binary packages generated from the **libxml** source package are found below **/pool/main/libx/libxml**.

At this point it is useful to identify the two different types of package found in Debian: source and binary packages. At the same time, there are native and non-native or external packages. It will all become clear in an instant! The maintainer transforms a software into a source package. Source packages are not The Debian package format (DEB) files but rather the combination of their source files. In the case of an external (non-native) package, a source package is made up of:

***.orig.tar.gz**
: The **.orig** file is a **tarball** containing the software in the way its (upstream) author released it.

***.diff.gz**
: The **diff** file encapsulates the changes needed to debianise a software. After applying the patch (a **diff** file is a patch), the software can be packaged for Debian with standard Debian tools.

***.dsc**
: The **dsc** file provides the essential information to describe a source package, including the **MD5** sums of the **orig** and **diff** files. It is signed by the maintainer and authenticates an upload[2].

Software that was specifically written for Debian does not need to be debianised. Therefore, the **diff** file does not exist and the **orig** file is replaced by a **tarball**, which, when unpacked, can be used directly to produce a Debian binary package.

With the information stored in the **./debian** subdirectory of a debianised source package, the Debian maintainer tools can produce a DEB file containing the software installable on and tailored for a Debian system. A DEB file is always a single binary package. A Debian source package can produce more than (but at least) one binary package. For instance, many libraries are split across three binary packages all generated from the same source package: **libfoo1**, **libfoo-dev**, and **libfoo-doc**.

[2] In combination with the **buildd**'s, the **dsc** file serves to identify an upload entity. For architectures other than the maintainer's native one, the **dsc** file is signed by the administrator of the **buildd** (see chapter 4.2).

The Debian archive currently contains about 15 000 binary packages generated from about 10 000 source packages.

4.1.2 Package indices

The **/dists** hierarchy provides the index files needed for **APT** to work and find DEB files to download[3]. A separate index is provided for each combination of the following four parameters:

- the release name, such as *stable* or *sarge*.
- the section, such as *main*.
- the target architecture.
- package type: source or binary.

The archive uses subdirectories to map these parameters to files, so finding the appropriate index file is a matter of climbing down the directory tree rooted at **/dists** based on these parameters.

On the first level there are the different releases with symlinks for the canonical names. For instance, when *sarge* is released, **stable** will be a symlink to **sarge**. Additional directories at that level include **experimental** and **stable-proposed-updates**. We will return to these in chapter 4.4.1 and chapter 4.4.4 respectively.

Below each release directory there are subtrees for the three sections which resemble the **/pool** hierarchy. The separation of all files within each release according to their degree of freedom is an important prerequisite to being able to produce or deploy archive snapshots with specific licence requirements. Also in the release directories are the **Contents** files, which map the files installed on the filesystem to the providing package. Tools such as **apt-file** (see chapter 5.4.4) use this information, and **grep** can usually extract all necessary information from this file as well.

In each section's directory, there are several subdirectories for the indices of binary packages as well as the directory for the source index. The index file is called **Packages** in all cases and contains the information of all available packages in the part of the archive identified by the four parameters.

For instance, **/dists/stable/main/binary-i386/Packages** contains the package descriptions for all binary packages in *main*, which can be installed as part of the *stable* distribution on the **i386** architecture. Similarly, **/dists/sid/contrib/source/Packages** references all source packages in *contrib* which are contained in *sid*. The architecture does not matter for source files.

[3]The package indices are not to be confused with the **/indices** directory found on the mirror; the latter indexes file in the mirror filesystem, while package indices index Debian packages stored therein.

4.1.3 The Release files

The /dists directory of a Debian mirror is home to the index files for the various releases provided by the mirror. Each such release is additionally described by a Release file, which contains important data about the release. The Release file of the *woody*'s third release looks like this:

```
~# cat Release
Origin: Debian
Label: Debian
Suite: stable
Version: 3.0r3
Codename: woody
Date: Mon, 25 Oct 2004 17:56:29 UTC
Architectures: alpha arm hppa i386 ia64 m68k mips mipsel powerpc s390 sp
arc
Components: main contrib non-free
Description: Debian 3.0r3 Released 25th October 2004
MD5Sum:
[...]
```

The Release file is used mainly by APT, which determines the architectures and components available from the mirrors specified in /etc/apt/sources.list using these files. Also, when mixing releases (see chapter 8.2), the various data can be used to specify criteria for pins. Finally, the file contains the checksums of all index files associated with the release. As shown in chapter 7.5, these checksums can be used to verify the integrity of packages downloaded from a Debian mirror.

4.2 The package upload

A Debian package has a life cycle, and a long way to go before it is distributed as part of the Debian *stable* release. Figure 4.2 illustrates how the different archives and components of the Debian infrastructure work together. You need not understand it all, but it may come in as a handy reference.

Following the debianisation process, a maintainer transfers the source files (along with the DEB file for the build architecture[4]) to one of the available upload queues. On the side of the accepting server, the Debian queue daemon moves the files to the unchecked directory at regular intervals. This directory is the domain of katie and friends[5], which verify that the uploaded package is signed with a trusted signature, and run a number of sanity checks on the package. On successful verification,

[4]This is required to make sure that no maintainer uploads without building the package locally first. At time of writing, the binary packages created by the maintainer directly propagated into the *unstable* archive. For all other (applicable) architectures, the build daemons are expected to generate the binary package(s) from the source package. Please see chapter 7.5 for security implications.

[5]katie and friends are a set of scripts named after female celebrities which work hand

the upload is moved to the **incoming** directory, which is accessible over the Web[6], but which should not be used as a package source except in special circumstances.

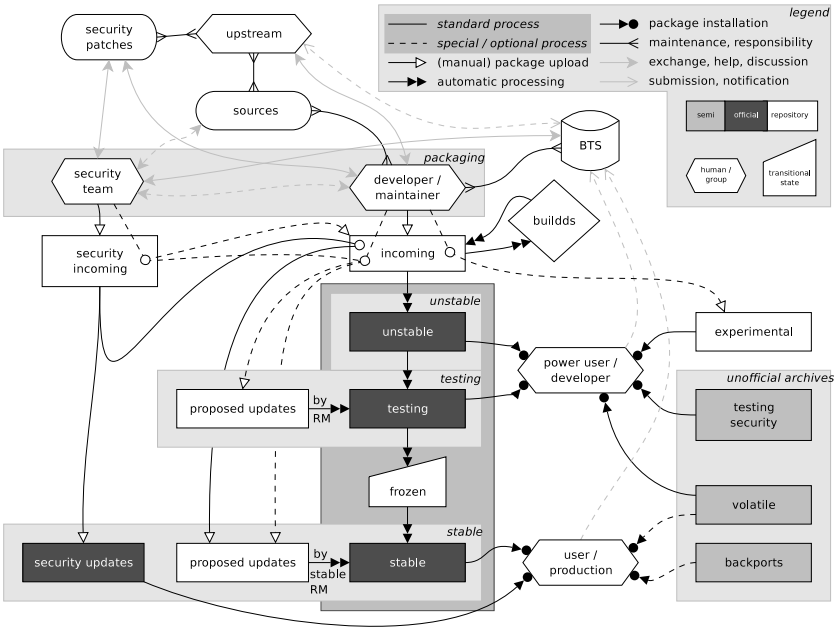

Figure 4.2: The life cycle of a Debian package (based on the work of Kevin Mark)

When an upload hits **incoming**, the build daemons (referred to as **buildd**) are notified[7]. There is at least one build daemon for each architecture that Debian supports (see chapter 4.5), and its job is to compile the software and produce a DEB file specific to the respective architecture. The resulting DEB is accompanied by a file describing it (the **.changes** file[8], which has to be signed by the administrator of the **buildd**). Finally, the package file is submitted to the upload queue and trickles into unstable as previously noted.

On a daily basis, **dinstall** moves available package files from **incoming** to the appropriate locations of the Debian pool (the **/pool** directory of every Debian mirror). It then updates the index files of the archive. Subsequently, the new packages are available from the *unstable* archive via **APT**.

in hand on the various tasks surrounding the management of the Debian archive. See **http://cvs.debian.org/dak/?cvsroot=dak**.

[6] **http://incoming.debian.org**

[7] The status of the individual **buildd**s is available at **http://www.buildd.net**.

[8] The **.changes** file is generated as part of the build process for each architecture and identifies a (set of) binary package(s). It must be cryptographically signed by a Debian developer for the package(s) to be considered for inclusion in the Debian archive. See chapter 9.2.12.

4.3 The official releases

Each of the three official Debian releases — *stable*, *testing*, and *unstable* — has specific traits related to the role the release plays during the package life cycle and the overall project. As a package usually enters the *stable* release by way of the *unstable* and *testing* archives, the following sections provide an overview of the three official releases in the same order that a new package encounters them on its way into the Debian system.

A Debian system can be installed and maintained using any of the three releases as package sources. In chapter 5.4.1 you see how a single release is selected, and chapter 8.2.1 describes how they can be combined. Note that all three of the official releases give you archive signatures for the index files of the corresponding archive (see chapter 7.5). In appendix C.1.1 you can find information to help you verify the keys used for the signatures.

4.3.1 The unstable release

As previously mentioned, the *unstable* release is in a state of continuous change. *unstable*, which is also called *sid*[9], is the workspace of Debian development. New packages percolate into the archive and become part of *sid* in a somewhat chaotic fashion. As a result, dependencies between packages break, only to be resolved later, conflicts appear and disappear, and packages possibly do not meet the quality standards of the rest of the archive. Furthermore, while maintainers take care not to inconvenience users tracking the *unstable* release, sometimes drastic changes in the packaged software hit the archive and can cause serious breakage on the target system.

The term "unstable" also applies to the packaging of software. Occasionally, a maintainer uploads a package in a rush, oversees a detail or makes a mistake in the packaging. The resulting package — if it makes it past the sanity checks — usually does not play ball with the local system, or installs horribly dysfunctional software. Even policy violations are possible. It is important to note that such policy violations are mostly restricted to misplaced files, but it should go without saying that *unstable* is *not suitable* for production environments. Having said that, most Debian developers run *unstable* on their primary machines. If the occasional failed dependency resolution is not fatal, *unstable* is quite a nice way to experience Debian — especially when there is a desire to contribute back to Debian with bug reports or interesting arguments on mailing lists.

In fact, it is unlikely for *unstable* to be more fragile than other operating systems, which are based on young software and whose developers try hard to publish the

[9] Sid is the name of the evil boy in Pixar's Toy Story who continuously breaks toys. It is thus an appropriate name for a release that can break a system. Conveniently, *sid* is also an acronym for "Still in development."

system as soon as possible. It goes without saying that short development cycles (such as Debian *unstable*) do not leave much room for testing, and therefore often result in a plethora of bugs.

Note that "unstable" refers primarily to the archive and the packages, and only indirectly to the software itself: software provided as part of Debian *unstable* may in fact be quite stable since packages in *unstable* usually correspond to official releases of the software. Thus it depends on the software author's quality standards how much runtime stability a programme needs to be part of an official upstream release. Often, a software will be available in two versions: an official release (which is often called "stable"), and a development release. If the latter is of any interest (or if the upstream authors are overly conservative with version numbering), chances are that a maintainer will provide pre-release packages for inclusion in Debian *in addition* to the official version. While not a rule, the development version usually comes in ***-snapshot** packages directly from the version control system to allow Debian users to be truly on the bleeding edge. For example, **gcc-snapshot** provides a bleeding edge version of the GNU compiler, while **gcc** provides a version deemed stable by the **gcc** developers.

As regards security updates, *unstable* enjoys a similar kind of attention as the *stable* release. While the security updates published by the security team might be restricted to the version in the *stable* release, a new and fixed version will usually become available in short time, and the maintainer will attribute special priority to uploading a fixed package to *unstable*.

Dealing with an *unstable* system is not very different from dealing with an installation of Debian *stable*. Upgrades for *unstable* are available through **APT**, but it is important to keep in mind that package upgrades in *unstable* have received considerably less testing than packages distributed as part of an official upgrade to Debian *stable*.

As the dependency information of packages in *unstable* can change, systems based on packages from the *unstable* archive should be upgraded with **apt-get --show-upgraded dist-upgrade** rather than with the plain **APT** upgrade mechanism. The **--show-upgraded** option is not needed but advisable to be able to inspect the changes proposed by **APT** before enacting them. In addition, tools such as **apt-listchanges** and **apt-listbugs** (see chapter 5.11.2 and chapter 5.11.3 respectively) are invaluable in assessing whether an upgrade is worth the trouble or involves unnecessary dangers.

4.3.2 The testing release

An upload to the Debian archive is accompanied with an urgency specification, coded into **debian/changelog** within the package. Normal uploads are of low urgency, while security updates enjoy prioritised treatment due to their high (or even

emergency) urgency. The urgency of an upload also determines when the uploaded version of a particular package moves from *unstable* to *testing*.

Depending on the urgency, a given version of a package must have been in *unstable* 10 (low), 5 (medium), 2 (high), or 0 (emergency[10]) days before being considered for testing. When a package is considered for promotion to *testing*, a number of other criteria have to be met before it is moved. If a previous version of the same package already exists in *testing*, the new version must have been built on at least all architectures supported by the previous package, and it must not have more release-critical bugs (see chapter 10.6.3) filed against it than the package in *testing*. Furthermore, all of the package's dependencies must be satisfiable within *testing*, and its declared relations cannot break another package already in *testing*.

When all these criteria have been met, the archive scripts move the package to testing, replacing any previous version[11]. *testing* is therefore generally not affected by the childhood diseases of packages as they hit *unstable*, but it is also not as current as *unstable*.

testing seems like the ideal release for all but the most critical applications. It is not on the bleeding but on the leading edge, and yet its contents has been scrutinised more carefully than the software from *unstable*. It also fluctuates less than *unstable*, which provides for easier maintenance. In the past, the major disadvantage of *testing* was the lack of security support. Security updates may already be delayed when they percolate to the *unstable* archive, and at least another two day delay is imposed before they are accepted into *testing* – provided all other requirements are met. Therefore, security updates in testing are sometimes delayed by several days, which is an important point to consider. Obviously, a home computer with a dial-up line to the Internet still qualified for a *testing* installation, but machines with a permanent Internet connection that offer services to the world, or machines that host multiple untrusted users are probably better off using *stable*, or *unstable* if that is an option.

Leading up to the release of *sarge*, the Debian testing security team has formed to address this shortcoming. At time of writing, the team is still operating unofficially, mainly coordinating through the **secure-testing-team** mailinglist hosted on **lists.alioth.debian.org**. An online record[12] with daily updates keeps track of outstanding security issues that persist in the *testing* archive. Depending on progress, *etch* could be supported with security updates while it is the *testing* release.

Similarly to *unstable*, it is advisable to use **apt-get --show-upgraded dist-upgrade** in place of **apt-get upgrade** because of the fluctuation in the set of packages provided in *testing*.

[10] Due to a limitation in the archive management script **britney**, it actually takes a day for emergency uploads to trickle into testing.

[11] Previous releases are available in the daily snapshots of the archive: **http://snapshot.debian.net**

[12] **http://merkel.debian.org/~joeyh/testing-security.html**

4.3.3 The stable release

Whenever the goals for the next release have been met[13], *testing* is frozen. During the ensuing freeze cycle, no new features are allowed to enter *testing*, and the developers concentrate on fixing bugs and providing additional translations. Especially bugs with severity above and including serious have to be fixed. These bugs are labelled RC and must be solved before a release can be made. Packages with outstanding RC bugs may be removed from the *testing* release during the freeze cycle.

Once *testing* is ready for release, the previous *stable* release is obsoleted (but archived[14], and the *stable* and *testing* symlinks changed to point to the next release generation. For this reason, it is advisable to hardcode the release codename in /etc/apt/sources.list, rather than its canonical name. Specifically, for a *sarge* system, I recommend changing all occurrences of "stable" with "sarge." While Debian release is unlikely to catch you off-guard, using the code names for the APT archive allows an upgrade to the next official release on your own schedule, and not when the symlinks in the archive change. When the next release follows, all you need to do is replace "sarge" with "etch" and then dist-upgrade as usual (see chapter 5.4.7).

As soon as a release has become the new *stable*, it becomes immutable. Security updates are kept in a separate repository (see chapter 7.2), and neither the set of packages nor the packages themselves are subject to change until the next official release comes around. It may seem a little peculiar to have security updates kept separate, but as with everything else, there is a reason for this procedure. Not every administrator wants security updates. Larger corporations frequently maintain their own internal release and have policies in place that require the ability to precisely identify the state of their machines. In such a case, fixes first need to be scrutinised before being provided internally. If the underlying archive (*stable*) were to change every other day, it would be impossible to maintain a consistent installation across hundreds of machines and simultaneously provide custom extensions and updates.

At semi-regular intervals, security and other proposed updates (such as trivial bugfixes) are merged with the last official release to create the next revision of the official release. These revisions ("stable dot releases" or simply "r-releases,") are identified by a specific suffix to the version number of the current *stable* release. For instance, when this book was written, the official Debian release was *Debian 3.0r3*, which is the third revision of the release after *woody* became *stable*. When a new dot release is published, it replaces the previous *stable* archive.

[13] http://release.debian.org

[14] http://archive.debian.org is the official archive address, and many mirrors feature /debian-archive as a sibling of /debian, which holds /dists and /pool. At time of writing, the primary site has not been reachable for a long time, and inquiries about its status have remained unanswered. Available mirrors are listed on the distribution archives web page: http://www.debian.org/distrib/archive

4.4 Unofficial APT archives

In addition to the three archives corresponding to the three official releases *stable*, *testing*, and *unstable*, a number of other APT repositories exist, and can be easily integrated with APT on systems that need them. The following sections introduce the most important of these. While it is certainly possible to run Debian systems for all purposes without these archives, the packages they contain may be needed at times. In any case, it is good to know about their existence and purpose.

4.4.1 The experimental archive

The Debian archive also hosts the *experimental* release, which contains packages that are not ready for public use, not even as part of *unstable*. Developers use this space to share packages as part of the development cycle. Unless you want to take part in this development (*e.g.* as a tester, or more actively), you can safely ignore the *experimental* archive.

The following lines in /etc/apt/sources.list enable APT to install software from *experimental* (see chapter 5.4.1). As always, please make sure you use your closest mirror instead (see chapter 5.4.1).

```
~# cat <<EOF >> /etc/apt/sources.list
deb     http://ftp.debian.org/debian experimental main
deb-src http://ftp.debian.org/debian experimental main
EOF
~# apt-get update
```

The *experimental* archive contains new major versions for some of the software found regularly in the Debian archive. For instance, APT 0.6 (see chapter 7.5.2) resides in *experimental*, while version 0.5 is available from the three release archives. The *experimental* archive is automatically deprioritised by APT so there is no need to worry about upgrading all your packages to the available experimental versions. This is accomplished with a special directive in the archive's Release file. See chapter 8.2.1 for more information:

```
~$ getfile /dists/experimental/main/binary-i386/Release
~$ grep NotAutomatic Release
NotAutomatic: yes
~$ apt-cache policy apt
apt:
  Installed: 0.5.27
  Candidate: 0.5.27
  Version Table:
     0.6.25 0
          1 http://ftp.debian.org experimental/main Packages
 *** 0.5.27 0
```

```
        500 http://ftp.debian.org sid/main Packages
        100 /var/lib/dpkg/status
```

To install software from the experimental archive, pass the **--target-release experimental** option to APT:

```
~# apt-get install --target-release experimental apt
[...]
Setting up apt (0.6.25) ...
```

4.4.2 The volatile archive

Debian's ***stable*** archive does not change beyond security updates, and these do not add new features (see chapter 4.3.3 and chapter 7). While administrators generally value this stability highly, certain types of software must change over time, even on the most stable systems. Prime candidates of such software include virus scanners, spam filters, and other tools which operate on data that is expected to change (such as whois).

While I was working on this book, a number of Debian developers started to conceive a strategy of how to deal with software that needs to change to remain usable. Such software was termed to be "volatile." A draft of the strategy is available at http://volatile.debian.net, which also hosts an APT-accessible archive for volatile software.

The goal of the ***volatile*** archive is to become a parallel to the security archive, and allow administrators to pull in updates with the same confidence with which they use the security archive. Changes will be limited to essential features and will only happen in close cooperation with the respective maintainers. Furthermore, security support for the packages in the ***volatile*** archive will be available.

To use software from the ***volatile*** archive, tell APT to use one of the mirrors found in the official mirror list[15], and update APT

```
~# cat <<EOF >> /etc/apt/sources.list
deb http://volatile.debian.net/debian-volatile sarge/volatile main
deb-src http://volatile.debian.net/debian-volatile sarge/volatile main
EOF
~# apt-get update
[...]
```

The ***volatile*** archive uses a custom version scheme designed to integrate and not conflict with the official packages from the main Debian archives (see chapter 5.7.5). All index files in the archive are signed with cryptographic signatures (see chapter 7.5), and information to validate the key used may be found in appendix C.1.2.

[15] http://volatile.debian.net/mirrors.html

4.4.3 The amd64 archive

Even though the **amd64** architecture is not yet officially supported by the Debian project, the port is ready to be used (see chapter 4.5.2). You can find installation and maintenance instructions at the port's web page[16].

Until it can be integrated with the main Debian archive, the **amd64** architecture is available from a separate **APT** repository. You can find details, as well as a list of mirrors online[17]. The archive's index files are signed with a separate key to ensure package integrity (see chapter 7.5). Information about the key may be found in appendix C.1.2.

4.4.4 The *-proposed-updates archives

The two directories **stable-proposed-updates** and **testing-proposed-updates** provide a way for developers to circumvent the normal package cycle via *unstable* and *testing* into *stable*. Packages uploaded to these directories are considered for manual inclusion by the respective release manager. Specifically, **stable-proposed-updates** serves as the basis for the next dot release of Debian (see chapter 4.3.3).

Even though both directories host proper **APT** repositories, you are herewith discouraged from using them directly. Software in either of these bypasses the regular Debian quality assurance surveillance and does not receive the same amount of testing as software that progresses via *unstable*.

4.4.5 The backports.org archive

Compared to *testing* and *unstable*, the Debian *stable* release often contains outdated software. Furthermore, many packages are not available at all because they have only been packaged recently. Even though single DEB files can be manually downloaded from newer releases, versioned dependencies make this impossible. For instance, upgrading **postfix** to version 2 (*e.g.* for policy server support) is not possible on a *woody* system without pulling in other packages from the next Debian version (*sarge*):

```
~# getfile pool/main/p/postfix/postfix_2.1.5-5_i386.deb
~# dpkg --install postfix_2.1.5-5_i386.deb
[...]
dpkg: dependency problems prevent configuration of postfix:
 postfix depends on libc6 (>= 2.3.2.ds1-4); however:
  Version of libc6 on system is 2.2.5-11.5
[...]
```

[16] http://www.debian.org/ports/amd64
[17] http://amd64.debian.net/README.mirrors.html

Undoubtedly, users of Debian *stable* are not going to be in favour of upgrading libc6; it would be a major change to a system, puting its stability at risk. An alternative would be to download the source and recompile the package against the libraries available in *stable*. If you have to do this more than once, the process becomes tedious and error-prone.

The backports.org archive[18] attempts to close this hole and distributes packages that have been recompiled in exactly this way. To get postfix version 2 installed on a *woody* system, the following line in /etc/apt/sources.list is needed. Please use the mirrors page[19] to find the mirror closest to you, and use that mirror instead of the main distribution server.

```
~# cat <<EOF >> /etc/apt/sources.list
deb http://www.backports.org/debian woody postfix
EOF
~# apt-get update
~# apt-get install postfix
[...]
Setting up postfix (2.1.4-2.backports.org.1) ...
[...]
```

As you may note, the required package is listed as part of the repository specification. The backports.org archive contains more than 450 packages, and you probably do not want all your installed packages to be upgraded to the latest backport[20]. Thus, backports.org allows you to specify precisely the set of packages you want to include. You can also specify multiple packages on a single line:

```
~# cat <<EOF >> /etc/apt/sources.list
deb http://www.backports.org/debian woody postfix subversion
EOF
```

As we will be discussing the APT sources syntax in chapter 5.4.1, you can take the above line as a way of making the *woody* backports for postfix and subversion available for direct installation with APT from the backports.org archive. Moreover, the line also ensures that backports of all dependencies can be installed with similar ease, if necessary.

Please note that the packages provided in the backports.org archive are not officially endorsed and come without any warranty. backports.org is not an official part of the Debian project, even though it is maintained and supported exclusively by official Debian developers. In particular, its packages have not undergone standard Debian quality assurance verifications, and have not received the same amount of testing as official Debian packages.

[18] http://www.backports.org
[19] http://www.backports.org/mirrors.html
[20] If you do, you can use the pseudo package name all in /etc/apt/sources.list instead.

That said, the source used to produce the packages in the **backports.org** archive comes directly from the official Debian archive and should therefore be as secure as the original version in the respective archive. Still, it is important to keep in mind that an extra delay exists for security fixes to percolate to the **backports.org** archive.

As a last note, if you are using a backported package from this archive, please refrain from reporting bugs against the Debian BTS. Instead, use the **changelog.Debian.gz** file to figure out the backporter's address to which to submit any bug reports, or send them to the **backports.org** mailing list[21]. The list is also the primary source of support for packages from the **backports.org** archive.

4.4.6 The apt-get.org directory

Setting up an **APT** repository is quite simple (as shown in chapter 9.3). Over the years, unofficial repositories have sprung up all over the place, providing useful Debian packages that are not included in Debian, or which are modified for specific purposes. The web site at **http://apt-get.org** serves as a directory for these sites.

The database can be searched by architecture and package name (or even a regular expression). The result encompasses all matching and registered repositories. For each entry, a short description, the matching package(s) (along with version information), and the necessary lines for **/etc/apt/sources.list** are provided. It is impossible to make an authoritative statement on the security, integrity, or stability of packages in the archives referenced from **apt-get.org** archive directory. If you use packages from sources listed here, you should be aware that they are packaged by people not necessarily connected to or supported by the Debian project. In particular, it would not be difficult to register an **APT** archive containing trojaned software. The directory has no guidelines, restrictions, or quality verification procedures governing the archives it lists. You have to decide for yourself which repositories you want to trust.

4.4.7 Christian Marillat's multimedia archive

Due to the freedom requirements on Debian packages, which the Debian project set in stone in the DFSG, many useful multimedia programmes cannot be distributed with the official Debian archive. Even though Debian is working hard with the respective authors to release the software under a free licence, progress is slow at times.

Christian Marillat, a Debian developer, maintains an unofficial Debian archive with prominent multimedia content. His archive, which is described on his web page[22],

[21] http://lists.backports.org
[22] http://debian.video.free.fr

is host to popular software, such as **Mplayer**, **lame**, **transcode**, and various video codecs. Christian maintains packages for this software unofficially, which is more of an indication of the level of support he can provide, than the quality of the packages themselves. Christian is an official Debian developer, and his archive is signed to allow for integrity verification (see chapter 7.5).

As a side note, **Mplayer** is actually available under a DFSG-compatible license and official packages have been prepared at time of writing of this book. Unfortunately, ***sarge*** will not include these packages.

4.5 Architecture support

Although the consumer market is full of computers powered (and heated) by derivatives of Intel's **x86** architecture, **PowerPC** machines, and the latest generation of 64 bit processors by AMD and Intel, a significant number of other architectures also profit from the support by the Linux kernel. Linux is gaining popularity as operating system for embedded devices (*e.g.* with **arm** or **mips** processors), professional servers (*e.g.* using **sparc**, **alpha**, and **hppa** chips), and entire mainframes (*e.g.* **S/390**-based). All of these architectures are supported by Debian, as well as some others.

Nevertheless, the Linux kernel does not make up an operating system by itself. The kernel is merely the interface between hardware and the user-space software. As large parts of the common user-space software (as well as the kernel itself) are written in medium-level languages (which require a compiler to generate processor-specific assembly code), sensible support for a processor architecture requires the support by the kernel as well as by the entire user-space software collection that makes up a Unix system. As the "universal operating system," Debian GNU/Linux extends the architectural support of the Linux kernel with the GNU user-space utilities on eleven different processor architectures. More supported architectures are in preparation.

To support an architecture means that all of Debian has been enabled to work on that specific architecture. Moreover, it also means that the installation feels like any other Debian system, independently of the processor architecture powering it. Therefore, the Debian operating system can be seen as a layer of abstraction, allowing unified system administration across different types of machines. With the exception of packages not applicable to all architectures (such as **memtest86**, a memory tester for the **x86** architecture), all packages available in the archive have been built for every one of the eleven supported architectures.

The combination of Debian, the underlying kernel, the user-space collection, and a processor architecture is called a "port" of Debian. The official Debian GNU/Linux ports (the architectures on which Debian GNU/Linux runs)[23] are:

[23] http://www.debian.org/ports

i386
: Being the first architecture supported by Linux, the IA-32 architecture found on **x86**-compatible chips by AMD, Cyrix, Intel, and others, is also Debian's most popular architecture.

ia64
: Together with HP, Intel finally abandoned full **x86**-(backward-)compatibility with the 64-bit IA-64 architecture. Debian started supporting **ia64** with the *woody* release. The **ia64** port allows the use of 32 bit code through software emulation.

powerpc
: Out of the cooperation between Apple, IBM, and Motorola grew the PowerPC chip, which powers IBM's RS/6000 line as well as Apple's PowerMac series. Support for the **powerpc** architecture was added in *potato*.

m68k
: The Motorola 68000 series of processors powers a wide variety of computer systems, most notably the sun3 workstation series, as well as the personal computers by Amiga, Apple Macintosh, and Atari. Debian added support for the **m68k** architecture with the *hamm* release.

sparc
: The Sun SPARC architecture powers the Sun SPARCstation workstation series as well as some models of the sun4 family. Similar to the **powerpc** port, the **sparc** architecture sports a 64 bit kernel but comes with a 32 bit userland. As an add-on to the **sparc** port, the **sparc64** sub-architectures aims to enable 64 bit user-space applications. Debian features support for **sparc** since the release of *slink*.

alpha
: Also with *slink* came support for the 64-bit Reduced Instruction Set Computer (RISC) architecture Alpha, developed by Digital (Digital Equipment Corporation, DEC).

arm
: The ARM processor is a low-power RISC chip by Acorn and Apple. Later, Digital and Intel joined to produce the improved StrongARM chips based on the **arm** architecture. First supported in *potato*, ARM processors are commonly found in mobile and embedded devices.

mips
: Used primarily in SGI machines, Cisco routers and gaming devices by Sony and Nintendo, this RISC chip has been supported since *woody*.

mipsel
: The "little-endian" brother of the **mips** architecture, found primarily in DEC-stations, also joined the Debian architectures family with *woody*'s release.

hppa
: Hewlett-Packard's PA-RISC architecture found support from Debian with the *woody* release. The **hppa** architecture is mainly found in HP machines running HP/UX (or Debian).

s390
: The IBM S/390 mainframe (reborn as **e**server zSeries in 2001) was officially adopted by Debian a short time later with the *woody* release. The 64-bit architecture power highly powerful chips optimised for parallel computing.

The advantages of supporting multiple processor architectures are self-evident. First, Debian gives a larger user base the ability to run Linux, as little to none viable user-space collections exist for users of non-Intel processor machines. Second, corporations and institutions, whose IT infrastructure has grown over years with a museum-like diversity of server architectures, are able to deploy Debian as a single operating system across all existing hardware. Thus, the costs of unifying system administration are kept as low as possible with Debian.

4.5.1 80386 — the processor

With **gcc-3.3** 1:3.3ds6-0pre6 (and also in some versions of **gcc-3.2**), the compiler started using the **bswap**, **xadd**, and **cmpxchg** instructions for code optimisation. These instructions are not available on real **80386** processors, but were added to the Intel instruction set with the **80486** processor series. With the packages for kernel versions 2.4.24 and 2.6.0, Debian added a patch to its Linux kernels to simulate these instructions in software on true **80386** processors. Unfortunately, the patch is known to be buggy and somewhat unmaintained.

The **80386** is an incredibly old and slow processor, but Debian would like to continue its support (it actively supports other architectures that are even less powerful than the **80386**, too). However, the upgrade from *woody* to *sarge* puts systems with true **80386** processors into an unfortunate catch-22 situation[24]: *sarge*'s **libc6** and **libstdc++5** both use the aforementioned instructions. Updating either of these libraries will hose the system until a new kernel is installed, but a new kernel cannot be installed due to a dependency on **modutils** (2.4 kernels) or **module-init-tools** (2.6 kernels), which in turn depend on a version of **libc6** not available in *woody*.

[24]Derived from the (excellent) book "Catch-22" by Joseph Heller, such a situation is an impossible situation where you are prevented from doing one thing until you have done another thing, but you cannot do the other thing until you have done the first.

At time of writing, the project is still discussing the possible steps to take. The suggestion for a special upgrade kernel was dismissed because of complexity and distribution issues. The preferred method to solve this would be the development of some user-space solution to emulate the missing instructions. If such a solution cannot be found, Debian will probably drop **80386** support altogether[25]. Debian *sarge* does not really run properly on one of these chips, largley due to memory requirements that cannot be fulfilled. Users of embedded **80386** machines typically have their own kernels to minimise memory usage.

Should **80386** processor support be dropped, the Debian project will look into to renaming its **i386** architecture to **i486** to indicate the change. However, the change might break existing scripts, as "i386" has been around forever. Further investigation will show. In any case, *sarge* supports the **80386** processor.

4.5.2 The amd64 architecture

While I was writing this book, Debian was ported to the **amd64** architecture. Being a very young port still, it is not distributed as an official port with Debian *sarge* nor contained in the official archive. Still, it is mostly complete and available for installation from its own archive (see chapter 4.4.3). At time of writing, four different ports exist for the **amd64** architecture:

sarge
: the 64 bit port of Debian *sarge*. The Debian **amd64** team is planning to provide security updates until the **amd64** architecture is part of the official archive.

pure64
: the 64 bit port of Debian *etch* and *sid*. This port will be integrated with the main Debian archive in the near future, and the 64 bit port of *sarge* will be merged in.

gcc3.4
: this port is identical to the **pure64** port, rebuilt with version 3.4 of the **gcc** compiler.

multi-arch
: an effort to integrate the multi-arch concept (see below) with **amd64**. Plans are to merge this port with **pure64** once it becomes part of Debian *unstable*.

Currently, the **pure64** port is the recommended port for **amd64** systems. At time of writing, it was not possible to upgrade an **i386** installation on an AMD 64 bit processor to any of the **amd64** ports. However, work is in progress to allow for this.

[25] http://lists.debian.org/debian-release/2004/10/msg00027.html

4.5.3 Multi-arch

With the advent of affordable 64 bit processors like the AMD Athlon 64, Debian has intensified its efforts to address the challenge of integrating 32 bit and 64 bit applications on the same system. Most 64 bit architectures support native or emulated 32 bit code execution, but the applications and libraries are incompatible across the two register sizes. Instead of implementing quick hacks or duplicating packages, Debian is trying to work with the LSB to come up with a method of integrating multiple architectures on a single machine in a scalable and well-designed way. Under the working title "multi-arch support", work has begun to address the challenge, and small test environments have already been put in place to help develop a policy[26].

The existing 64 bit architectures (**ia64** and **sparc64**) use separate directories to hold the 32 bit and 64 bit versions of the installed libraries. The approach is commonly referred to as "biarch" and is not free of problems. Apart from breaking the rules of the FHS (see chapter 5.7.4), the approaches differ and do not scale to other architectures, or similar changes in the future. As multi-arch reaches production status, current 64 bit architectures are expected to switch to using it. In addition, with multi-arch, Debian will be able to add full support for other 64 bit architectures, including **powerpc64**, **mips64/mipsel64**, **hppa64**, **sparc64**, and **s390x**, within a short time[27].

Until multi-arch is ready for production use, special arrangements have to be made to run 32 bit applications on 64 bit installations. One good technique is to use of a **chroot** managed by **dchroot** (see chapter 8.3.1).

[26] You can find more information about multi-arch at **http://people.debian.org/~taggart/multiarch**

[27] Some of these architectures (such as **sparc64**) are already supported, but use the deprecated **biarch** approach.