

Debian Repository HOWTO

Aaron Isotton

aaron@isotton.com

This document explains what a Debian repository is and how you can set up one.

1. Introduction

A Debian repository is a set of Debian packages organized in a special directory tree which also contains a few additional files containing indexes and checksums of the packages. If a user adds a repository to his `/etc/apt/sources.list` file, he can easily view and install all the packages available in it just like the packages contained in Debian.

A repository can be both online and offline (for example on a CD-ROM), although the former is the more common case.

This document explains how Debian repositories work, how to create them, and how to add them to the `sources.list` correctly.

This document's master location is <http://www.isotton.com/debian/docs/repository-howto/>.

1.1. Copyright and License

This document, *Debian Repository HOWTO*, is copyrighted (c) 2002-2006 by *Aaron Isotton*. This document is dual-licensed as follows:

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License, Version 2.

1.2. Feedback

Feedback is most certainly welcome for this document. Send your additions, comments and criticisms to the following email address: <aaron@isotton.com>.

2. Terms Used in this Document

distributions

The three Debian distributions: *stable*, *testing* and *unstable*.

index files

The `Packages.gz` and `Sources.gz` files.

3. How Repositories Work

A repository consists of at least one directory with some DEB packages in it, and two special files: `Packages.gz` for the binary packages, and `Sources.gz` for the source packages.

If your repository is listed correctly in `sources.list` (more on that later), **apt-get** will fetch the `Packages.gz` index if the binary packages are listed (with the `deb` keyword) and `Sources.gz` if the sources are listed (with the `deb-src` keyword).

`Packages.gz` contains the name, version, size, the short and the long description, and the dependencies of each package, plus some additional information which is not of interest for us. All that information is listed (and used by) the Debian package managers such as **dselect** or **aptitude**.

`Sources.gz` contains the name, version and the build dependencies (the packages needed to build) of each package (plus some information which is not of interest for us, too); that information is used by **apt-get source** and similar tools.

There's an optional `Release` file containing some informations about your repository; that is used for *Pinning*, an interesting trick I won't go into in this document. You can read more about pinning in the APT HOWTO (<http://www.debian.org/doc/manuals/apt-howto/>).

Thus, once you have set up your repository, you can list and install all of your packages together with the ones in Debian; if you update a package, it'll be upgraded when the user runs **apt-get upgrade**; and every user will be able to easily see a short description and other important information about your packages.

But there's more to it. If created properly, repositories can offer different packages for each supported distribution and for each of the (currently eleven) supported architectures; `apt` will automatically fetch the right one for a user's machine, without him even having to know about all the different architectures. It also allows you to group your packages into components, just as Debian's packages are divided into `main`, `non-free` and `contrib`. So, especially if your software is cross-platform, you'll love package repositories.

4. How to Set Up a Repository

There are two types of repositories: more complex ones where the user has only to specify the base path to the repository, the distribution and the components he wants (`apt` will automatically fetch the ones for the right architecture, if available), and simpler ones where the user has to specify an exact path (and `apt` will do no magic to find out which packages are the right ones). The former are a bit more complex to set up, but easier to use, and should always be used for complex and/or cross platform repositories; the latter are simpler to set up, but should only be used for small or single-architecture repositories.

Although it is not really correct, I'll call the former *Automatic Repositories* and the latter *Trivial Repositories*.

4.1. Automatic Repositories

The directory structure of an automatic repository with the standard Debian architectures and components looks like this:

Example 1. A Standard Debian Repository

```
(your repository root)
|
+-dists
  |
  |--stable
  |  |--main
  |  |  |--binary-alpha
  |  |  |--binary-arm
  |  |  |--binary-...
  |  |  +-source
  |  |--contrib
  |  |  |--binary-alpha
  |  |  |--binary-arm
  |  |  |--binary-...
  |  |  +-source
  |  +-non-free
  |     |--binary-alpha
  |     |--binary-arm
  |     |--binary-...
  |     +-source
```

```

|
|-testing
| |-main
| | |-binary-alpha
| | |-binary-arm
| | |-binary-...
| | +-source
| |-contrib
| | |-binary-alpha
| | |-binary-arm
| | |-binary-...
| | +-source
| +-non-free
|   |-binary-alpha
|   |-binary-arm
|   |-binary-...
|   +-source
|
+-unstable
  |-main
  | |-binary-alpha
  | |-binary-arm
  | |-binary-...
  | +-source
  |-contrib
  | |-binary-alpha
  | |-binary-arm
  | |-binary-...
  | +-source
  +-non-free
    |-binary-alpha
    |-binary-arm
    |-binary-...
    +-source

```

The free packages go into `main`; the non-free ones into `non-free`, and the free ones which depend on non-free ones into `contrib`. Debian currently supports 11 architectures; I've omitted most of them for the sake of brevity.

Each `binary-*` directory contains a `Packages.gz` and an optional `Release` file; each `source` directory contains a `Sources.gz` and an optional `Release` file. Notice that the *packages* do not have to be in the same directory as the index files, because the index files contain paths to the individual packages; in fact, they could be *anywhere* else in the repository. This makes it possible to create pools.

You are free to create as many distributions and components and to call them as you wish; the ones I used in the example are just the ones used in Debian. You could, for example, create the distributions `current` and `beta` (instead of `stable`, `testing` and `unstable`), and the components `foo`, `bar`, `baz` and `qux` (instead of `main`, `contrib` and `non-free`).

While you are free to call the components as you want, it is generally a good idea to use the standard Debian distributions, because that's what Debian users expect.

4.2. Trivial Repositories

Trivial repositories consist of one root directory and of as many subdirectories as you wish. As the users have to specify the path to the root of the repository and the relative path between the root and the directory with the index files in it, you are free to do whatever you want (even to put everything into the root of the repository; then, the relative path will be simply “/”).

Example 2. A Trivial Repository with Two Subdirectories

```
(your repository root)
|
|-binary
+--source
```

4.3. Creating the Index Files

dpkg-scanpackages generates the `Packages` file and **dpkg-scansources** the `Sources` file.

They both send their output to stdout; thus, to generate compressed files, you can use a command chain like this one: **dpkg-scanpackages arguments | gzip -9c > Packages.gz**.

The two tools work the same way; they both take two arguments (in reality there are more, but I won't go into that here; you can read the manpages if you want to know more); the first the directory under which the packages are, and the second is the *override file*. We don't need override files for simple repositories, but as it is a required argument, we simply pass `/dev/null`.

dpkg-scanpackages scans the `.deb` packages; **dpkg-scansources** scans the `.dsc` files. It is thus necessary to put the `.orig.gz`, `.diff.gz` and `.dsc` files together. The `.changes` files are not needed.

Thus, if you have a trivial repository such as the one from Example 2, you can create the two index files as follows:

```
$ cd my-repository
$ dpkg-scanpackages binary /dev/null | gzip -9c > binary/Packages.gz
$ dpkg-scansources source /dev/null | gzip -9c > source/Sources.gz
```

If you have a repository as complex as the one in Example 1, you'll have to write some scripts to automate this process.

You could also use the *pathprefix* argument of the two tools to simplify the syntax a bit; I leave this as an exercise for the reader. (It's documented in the manpages).

4.4. Creating the Release files

If you want to enable the users of your repository to use *Pinning* with your repository, you must include a `Release` file in every directory containing an index file. (You can read more about pinning in the APT HOWTO (<http://www.debian.org/doc/manuals/apt-howto/>)).

The `Release` files are simple and short text files of the following form:

```
Archive: archive
Component: component
Origin: YourCompany
Label: YourCompany Debian repository
Architecture: architecture
```

Archive

The name of the distribution of Debian the packages in this directory belong to (or are designed for), i.e. `stable`, `testing` or `unstable`.

Component

The component of the packages in the directory, for example `main`, `non-free`, or `contrib`.

Origin

The name of who made the packages.

Label

Some label adequate for the packages or for your repository. Use your fantasy.

Architecture

The architecture of the packages in this directory, such as `i386`, `sparc` or `source`.

It is important to get `Archive` and `Architecture` right, as they're most used for pinning. The others are less important.

4.5. Creating Pools

With automatic repositories, distributing the packages in the different directories will quickly lead to an unmanageable beast. It is also a waste of space and bandwidth, as there are many packages (for example documentation packages) which are the same for all architectures.

In these cases, a possible solution is a *pool*. A pool is an additional directory under the repository root containing *all* packages (the binaries for all architectures, distributions, and components, and all the sources). Through a smart combination of override files (which are not covered in this document) and of scripts many problems can be avoided. A nice example of a pooled repository is the Debian repository itself.

Pools are only useful for big repositories; I've never made one and I don't think I'll need to in the near future, and that's why I don't explain how to make one here. If you think that such a section should be added, feel free to write one, and contact me.

4.6. Tools

There are various tools to automate and ease the creation of Debian archives; I've listed the most notable of them here.

apt-ftparchive is used to move a collection of Debian package files into a proper archive hierarchy as is used in the official Debian archive. It is part of the `apt-utils` package.

apt-move is used to move a collection of Debian package files into a proper archive hierarchy as is used in the official Debian archive.

5. Using a Repository

Using a repository is very simple, but it depends on what type of repository you have made: binary or source, and automatic or trivial.

Each repository gets one line in `sources.list`; for a binary one, you use the `deb` command, and for a source one a `deb-src` command.

Each line has the following syntax:

```
deb|deb-src uri distribution [component1] [component2] [...]
```

The *uri* is the URI of the root of the repository, such as `ftp://ftp.yoursite.com/debian`, `http://yoursite.com/debian`, or, for local files, `file:///home/joe/my-debian-repository`. The trailing slash is optional.

For automatic repositories, you must specify one distribution and one or more components; the distribution must not end in a slash.

Example 3. Two Automatic Repositories from my sources.list

```
deb ftp://sunsite.cnlab-switch.ch/mirror/debian/ unstable main contrib non-free
deb-src ftp://sunsite.cnlab-switch.ch/mirror/debian/ unstable main contrib non-free
```

These two lines specify an automatic binary and source repository with root `ftp://sunsite.cnlab-switch.ch/mirror/debian/`, the distribution `unstable` and the components `main`, `contrib` and `non-free`.

If the repository is not automatic, then the *distribution* specifies the relative path to the index files and must end with a slash, and no components may be specified.

Example 4. Two Trivial Repositories from my sources.list

```
deb file:///home/aisotton/rep-exact binary/
deb-src file:///home/aisotton/rep-exact source/
```

The first of these two lines specifies a binary repository in `/home/aisotton/rep-exact/binary` on my local machine; the second specifies a source repository in `/home/aisotton/rep-exact/source`.

6. See Also

- The **apt-ftparchive** documentation.
- The **apt-get** documentation, and the documentation for `apt`.
- The **apt-move** documentation.
- <http://www.apt-get.org/> for many examples of real-world repositories.
- The APT HOWTO (<http://www.debian.org/doc/manuals/apt-howto/>).
- The **dpkg-scanpackages** documentation.
- The **dpkg-scansources** documentation.
- The `sources.list(5)` manpage.