# Practical Debian Administration

Alexander Zangerl
az@debian.org

**Abstract**

Debian GNU/Linux offers a very rich and well-designed environment to a system administrator. This talk is about the most important tasks you will face on any Debian box, and how to deal with them most efficiently; I will also present some of the less well-known (but in my opinion still quite interesting) Debian-specific tools and approaches that can make administrating a Debian system very pleasant.

## 1 Debian in a Nutshell

Debian is a free, volunteer-produced Linux/Unix Operating System Distribution. As I pointed out in some detail in a previous paper[1], Debian is very well suited for experienced Unix users as well as large scale environments for a variety of reasons; we might very well call Debian "The Distribution for Sysadmins by Sysadmins".

The biggest difference[5] between Debian and other operating system distributions is that Debian uses a very extensive policy and stringent quality assurance processes which produce one well integrated operating system from thousands of disjunct little pieces of software.

Other distributions recently have adopted `apt` as Debian's "killer app" as they perceive it, but in fact the real killer apps are the policy, the bug handling rules and methods, the security team and all the other codified consistency and quality measures.

While people are using "`apt-get` into it!" as slogan for Debian, I'd rather something like "Policy makes Perfect".

All that flexibility and the project size come at a price[6], too, but mostly that price is paid by the developers. For a user things are not as harsh and overly complex even given little to moderate experience with Unix.

This talk tries to cover the most important administration tasks one usually faces on a Unix system and how Debian can be made to Do What You Mean.

## 2 You're in a maze of twisty `shell` prompts...

So somebody tricked you into installing this Debian thing on your computer, right? And you followed all the prompts of the installer, marvelling at how much weird stuff it asked you. And now, the installation done, all you have is this boring little `shell` prompt. Where is all that functionality hinted at beforehand? How do you actually make use of this new Debian system *efficiently*?

Let's look at the first steps you should take in any new environment: a brief familiarisation.

### 2.1 File Locations

Debian follows the *Filesystem Hierarchy Standard*[2] very strictly. For you this means that all configuration information will be in `/etc`, `/usr` contains all the static data and `/var` is where variable data like caches, databases and so on live.

### 2.2 Documentation

All documentation can be found on the main Debian website[3], but is also available on the actual system in the form of man pages, plain text files and info, html and pdf documents. All documentation must come at least in plain text, usually `gzip`ed to save space. The wrapper program `zmore` is always available to read these quickly.

`/usr/share/doc` is the location for documentation. Every software package has to provide a minimal set of documentation in `/usr/share/doc/`*`pkgname`*, at least including `changelog.Debian.gz` and a `copyright` file. The `changelog` file describes the revision history of the Debian port of the package and the `copyright` file tells you where the software came from originally. Very often you'll also find a `README.Debian` which outlines major differences between the upstream software and the Debian port.

The information about the Debian project and the distribution is in `/usr/share/doc/debian` (coming from the *doc-debian* package) and includes an FAQ collection and a few other Getting Started docs.

If you need information about the exact rules and policies affecting Debian's behaviour, install the package *debian-policy*: the policy documents will shine light on all intricacies of package management, file locations, program behaviour and so on that make Debian's thousands of pieces work together consistently.

The *newbiedoc* package includes a couple of useful documents for newbies to the Debian environment, whereas the *apt-howto* package (or a language-specific version like *apt-howto-en*) contains a large amount of important information about `apt`, the main package management interface.

Let's assume that you do not immediately run into trouble and bugs, which is quite likely the case if you have installed the "*stable*" distribution stream. If you do find a problem you'll want to know that bugs and problems are handled by the *Bug Tracking System*, which is easiest to access via the BTS website[4], but there is also a very useful text-mode tool called `querybts` (in the *reportbug* package).

## 3 Getting Started

If you are a Unix oldtimer, then there won't be a lot of surprises for you: you can do pretty much everything the old-style "./configure; make; make install" way. However you'll lose most of the benefits of precompiled and -configured software, so you won't usually want to take that hard route.

Debian does not come with a single configuration frontend; instead it follows the ages old Unix philosophy and provides "mechanism not policy" – in the form of the fundamental `dpkg` package manager and some tools built on top of it, most well known among them `apt`.

Everything is configured in text files, and often what I'd call "meta-configuration" files are used as templates to generate the actual configuration.

So let's look at the tools with which you can interact with Debian next, in order of appearance.

## 3.1 Which Tool for Which Purpose?

During the initial installation you won't interact with `dpkg` or `apt` but either with `tasksel` or `dselect`, an older frontend to `dpkg`.

#### `base-config`

The final phase of the actual install is controlled by `/usr/sbin/base-config`: this includes things like setting of a root password, setting the timezone, specifying whether and where from to install extra software and so on.

`base-config` can be run without major trouble later on again if you need to repeat the steps for whatever reason. What may be more interesting to you is how `base-config` is doing its job (and how to reuse the magic contained therein):

Like a lot of administration tools in Debian, `base-config` uses small stub modules or scripts that are executed in sequence to actually provide all the functionality. If you look into `/usr/lib/base-config` you will find scripts like `20passwd` which deals with adjusting the basic authentication environment.

There are no hidden depencendies or trust assumptions between these scripts, and it is perfectly safe for the administrator to run them independent of `base-config`.

In most cases in Debian the sequencing of scripts to execute is done by a tool called `run-parts` which you should use when designing such an environment yourself.

#### `tasksel`

Likely you did run `tasksel` once during the installation and were prompted with a high-level selection of "tasks" or feature sets you might want to install on your system.

If you are not happy with the choices made and want to add software then you can simply run `tasksel` again, but be aware of the fact that `tasksel` is not meant for ongoing administration: it does not deal with removal of packages at all, just with the quick and simple addition of common feature sets.

`tasksel` of course does its job by relying on the main package management tools and is just a very simple-minded frontend for them; this means among other things that you can do by yourself the same thing `tasksel` does.

The most interesting option offered by `tasksel` is in my opinion `-t`: If you run `tasksel -t` it will *not* automatically install all the packages that your selection affects but only print out the `apt-get` commandline that it would have run without `-t`.

Using this option gives you a quick indication of what packages are actually included in the particular tasks and also allows you to get rid of tasks later on: replace `apt-get install` with `apt-get remove --purge` in `tasksel`'s output and run that command.

However, as simple `tasksel` is to use for initially populating your system, quite likely you will want to use the more fine-grained controls available in other tools later on.

### dselect

`dselect` is the oldest "user-friendly" frontent for the Debian package manager. Personally I don't like it; it's very powerful but ugly to interact with. I think it suffers a bit from bad interface design and should be avoided nowadays.

The reason for `dselect` still being offered during the installation process is that the newer frontends were not in releasable state at the time the current "*stable*" distribution was released.

### dpkg

`dpkg` implements the core package manager functionality, it deals with installed packages and existing package files, but has no idea of different package sources, distribution streams or the like.

The most common usages of tools from the `dpkg`-layer for a system administrator are as follows, in rough order of importance:

- `dpkg -l` *name* or `dpkg -s` *name* show a long or short status of selected packages installed on your system. If *name* is not specified, all packages are examined.

  For an explanation of the status columns in the output of `dpkg -l` please refer to [1]; the basics you need to know are that "`ii`" is what you want to see for an installed package, "`rc`" means a removed package with config intact and "`pn`" is a purged package.

- `dpkg -i` *packagefile*`.deb` installs the package from file *packagefile*`.deb`. This fails if there are unsatisfied dependencies or conflicts.

  You will need this to install locally generated packages. The most common situation where you have to build a package on your system is a custom kernel image; more about that in section 7.

- `dpkg -r` *name* removes the package *name* but keeps the configuration files in place. Removal fails if there are packages depending on *name*.

- `dpkg -P` *name* removes the package *name* and also its configuration files.

- `dpkg-reconfigure` *name* repeats the configuration phase of package *name*; see section 4.3 for details about the configuration system.

- `dpkg -L` *name* shows the files belonging to package *name*.

- `dpkg -S` *file* searches for a package that *file* belongs to.

  Files belonging to a package include all the files that were provided in the package file, which means all executables, libraries and so on, and also configuration files that are under `dpkg`'s control[1].

After your installation finishes you should look at `dpkg -l`'s output, make yourself familiar with what is currently installed and maybe do a bit of clean up: try to `dpkg -P` packages you think should not be installed, and look at the descriptions of those packages you are not sure about. `dpkg` will complain about breaking dependencies (without doing anything bad) if you try to remove a package that is still needed by something else.

[1] There is a subtle difference between configuration files `dpkg` handles and general configuration files which is explained in Section 10.7 of the Policy Manual; for an administrator it's important to know that only the former show up as belonging to a package.

### `apt` and Other Frontends

Debian software can be installed directly from lots of Debian mirrors all over the Internet, but you need a network-aware package management tool.

`apt` is a command-line version of such a tool, but for the GUI fans (text-mode or X11) there are a couple of frontends like `aptitude` and `deity`, both offering a curses interface, and graphical tools like `synaptic`, `kpackage`, `gnome-apt` or `stormpkg`.

The next section outlines how to use `apt` to get the work done.

# 4 Getting Software Installed

First you will need to know what software is actually available. The file `/var/lib/dpkg/available` lists all packages known to your system and `grep` is your friend.

Of course there are easier ways of retrieving this information, namely `apt-cache search` *`regex`* which looks for matching package descriptions. `apt-cache show` *`pkgname`* provides you with the full description (and dependency information) about a particular package.

You may want to quickly see what dependencies a package requires and `apt-cache depends` *`pkgname`* provides just that.

To actually install software you use `apt-get install` *`pkgname...`* which will pull in all required depencies. The installation will proceed and succeed if there are no conflicts that need human reasoning to resolve; in practice you won't see such conflicts unless you try to use or mix distribution streams besides "stable" and/or inofficial package sources.

If your installation left you with incomplete configuration for `apt` to do its job, examine `/etc/apt/sources.list` and `/etc/apt/apt.conf`.The sources for Debian packages you want to use (eg. CD, website, ftpserver) are specified in `sources.list` and `apt.conf` controls the general behaviour of `apt` (eg. whether to use a web proxy and which).

`apt-setup` is an interactive user-friendly tool to produce extra `sources.list` entries if you don't want to edit that file manually. (`dpkg-reconfigure apt` does *not* go through the source selection step again.)

Likely you will want to install the tool `apt-show-versions` which can tell you about packages with updates available, outdated versions etc. This is especially important if you want to mix distribution streams as outlined in section 6.

`apt-get` can also remove or purge packages with the commands `remove` and the option `--purge`, respectively. `apt` keeps a cache of downloaded packages in `/var/cache/apt` which can grow quite large: `apt-get clean` cleans that up.

For the zealots among you the first extra tool you will install might be *vrms*, the "virtual Richard M. Stallman"; more pragmatic people would likely look for their favourite editors and shells next.

## 4.1 The Update Cycle

Debian releases seldom, and only after longish quality assurance periods where the software in the distribution is made as bug-free and well-integrated as possible. A normal installation will usually use this "stable" distribution stream as source

for software to install, but it is possible (and not *too* painful) to mix distribution streams which section 6 describes.

"stable" is always quite a bit out of date wrt. software versions, but is kept secure and stable by the efforts of the Debian security team and the individual software maintainers who backport critical and security fixes to the versions available in "stable".

Your `sources.list` should reference the archive of security updates at `http://security.debian.org/`, and you should periodically run `apt-get update` to refresh the list of available packages (which is kept locally on your system).

After that you may want to use `apt-get upgrade` which will look for packages with updates available, retrieve the appropriate files and install the newer versions without disrupting your system.

The `apt` configuration entry `Apt::Get::Show-Upgraded "true";` makes `apt` show packages that are candidates for an upgrade. Apart from that I find two packages very useful, because they provide tools that modify `apt`'s upgrade behaviour favourably:

- `apt-listchanges` is a small tool that shows new Debian changelog entries for a package when updating (but before installing) a package. All Debian packages must come with a changelog file with a strictly defined format, and `apt-listchanges` allows you to scrutinize the changelog during installation and maybe abort the installation if you detect potential problems.
- A sibling tool, `apt-listbugs`, warns you of known critical bugs before installing a package. It does so by querying the online bug tracking system and will work only if your system is networked and has web access.

To (semi-)automate the update-upgrade process you can use the package *cron-apt* which provides a set of scripts for `cron`. By default `cron-apt` will only download new packages, notify you of these and not do the actual upgrade; as there are situations where an upgrade requires operator interaction this is the sensible thing to do, but like most Debian tools you can disable such security features.

You may also want to save (most of) the state of a package with `dpkg-repack` before trying out a newer version: this tool will pack up all files belonging to an installed package into a snapshot package which you can keep around until you are satisfied that the new version really works.

There are a couple of downsides to this approach, the most severe being that the files belonging to a package will usually not include generated data files or generated config files. As an example the files under the default webserver docroot `/var/www` do *not* belong to any webserver package, similar to database files not belonging to a database application. In practice this means that you will have to capture a lot more data together with `dpkg-repack`'s result if you want to completely roll back a package update.

## 4.2 Getting Rid of Software

Over time your system may accumulate packages that you do not actually need anymore. This is a direct consequence of the very fine-grained dependency specification Debian uses, but it's also something you don't have to worry about.

There are two main approaches to facilitate dealing with unneeded packages:

- The package *deborphan* provides a tool of the same name which looks for packages that have no other packages depending on them. By default `deborphan` will only consider library packages (which have a name prefixed with `lib`) but with the option `-a` it will check all packages. The results will not be complete for end-user packages of course, but will still give you at least some indication.
- *debfoster* approaches this problem in a reactive manner: you tell `debfoster` which packages you want to to keep installed. From then onwards it detects packages that have been installed only because other packages depend on them. If these dependencies change `debfoster` will notice this and ask you about removing now unneeded packages.

### 4.3 Automatic Configuration with `debconf`

All (recent) Debian packages must use the tool `debconf` if they need to interact with the administrator during installation or upgrade. `debconf` is a system to prompt users for configuration choices and store these answers, which offers different user interfaces and multiple backend databases.

The administrator of a system can specify whether low-priority questions should be hidden and answered in the default manner when `debconf` is installed; if you need to adjust this configuration later on just run `dpkg-reconfigure debconf`.

`debconf` is very useful if you want to install software on multiple systems: you configure it on one system interactively, then make the answers available to `debconf` on the other systems. The answers are stored in the textfile `/var/cache/debconf/config.dat`, which you could copy to your target system but the better approach is to extract only the relevant answers and transfer these. To do that you'll need some tools from package `debconf-utils`, most important `debconf-communicate`.

If you want to reconfigure an installed package *pkgname* with new configuration choices, you just run `dpkg-reconfigure packagename`, optionally setting the frontend via `-f` *`type`* to the non-interactive frontend (which keeps `debconf` from prompting you at all).

## 5 Customizing the Environment

### 5.1 Alternatives

In Debian much functionality is provided in slightly varying form by different packages, e.g. editors or mail transfer agents. This diversity is handled by using "virtual packages" (allowing simple package dependency specifications without actually naming a particular package) and the "alternatives" system: common tools or generic functionality like "editor" which are implemented by multiple packages are made available by two symlinks: one from the normal `bin` directory to `/etc/alternatives/` and one from there to the currently selected tool that provides the functionality.

As more than just the path to the binary needs to be adjusted when choosing a provider of functionality (e.g. manpages), a configuration tool called `update-alternatives` is offered to make such choices.

`update-alternatives --display` *`genericname`* displays what alternatives for *genericname* are available, with option `--config` you are prompted which alternative to select and option `--auto` will revert to the default automatic state. Of

course you can change these links by hand, too; `update-alternatives` is just more convenient.

## 5.2 File Diversions, Permission Overrides

Occasionally it is desirable to move files out of the way, for example to disable some unwanted functionality without removing the whole package (like disabling unwanted plugin libraries). Of course you can just remove the offending file, but on an upgrade of the package it will be reinstalled. You can tell the package manager about your wishes with `dpkg-divert` which persistently stores your instruction to override/remove/rename a file provided by some package (in the file `/var/lib/dpkg/diversions`). `dpkg-divert --add --divert-to` *`newpath`* `--rename` *`pathname`* moves *pathname* to *newpath*, and this diversion will be kept across upgrades unless you cancel it with option `--remove`. `dpkg-divert --list` produces a list of all present diversions.

If it becomes necessary to override file permissions and ownerships, then you should use `dpkg-statoverride` which makes your changes permanent. Such status overrides are recorded in `/var/lib/dpkg/statoverride`.

## 5.3 The `init` System

Debian uses the System-V `init` style with multiple runlevels and symlinks pointing to the actual scripts in `/etc/init.d`. To make administration of the symlinks as easy as possible there is a tool called `update-rc.d` which can add, remove or change the startup links of services as well as quickly revert to the default state for a particular service.

One important thing to know about Debian initscripts is that policy does strongly suggest that configuration settings for services should not be put into the actual initscript but in `/etc/default/`*`name`*.

Most important example of such configuation settings is `/etc/default/rcS`, which controls the generic boot process. For a remotely administered server setting `FSCKFIX=yes` may be a good idea, and setting `EDITMOTD=no` should disable the automatic regeneration of `/etc/motd` on every reboot.

## 5.4 Other Useful Helpers for Automating Tasks

For quite a lot of services and programs you will find that there is a `update-`*`something`* tool available; commonly used ones include for example `update-inetd` which makes changing `inetd`'s configuration really painless (compared to using things like `sed` to effect changes).

If your system is to be a desktop system, you may want to know about `update-mime`: this tool propagates your choices about MIME handler tools from `/etc/mailcap.order` to the main MIME capabilities file `/etc/mailcap`. This file is consulted not just by `metamail` but also by the general purpose tool `run-mailcap`. Going through these settings is very useful for example if you have multiple graphics tools installed and the default choice of image handler does not meet your expectation: setting the order of tools to choose from rather than finetuning every single application that might use an image handler is a big time-saver.

For users of kernel modules `update-modules` will be important: Debian expects module options to be defined in multiple files in `/etc/modutils`, which are then combined into `/etc/modules.conf` by `update-modules`. That way packages can produce and maintain their own module option files without having to fear any clashes with other packages, and of course this feature is also available to the administrator.

# 6 Mix and Match

The default choice of distribution stream is normally "stable" which is a good basis for a server system but likely won't make a good desktop if the user expects current software versions.

`apt` can deal with mixing the available distribution streams ("stable", "testing", "unstable" and "experimental"), but there are a couple of caveats and problems to take care of:

First you need to bear in mind that security updates are done only for "stable"; other streams may or may not get updated in time or at all.

There are also some other sources of Debianized packages which are not part of the main distribution for various reasons; check the site `http://apt-get.org` which lists the packages offered by such non-standard repositories.

Eventually you need to tell `apt` about the other package sources: you need to list the sources in `/etc/apt/sources.list`, and run `apt-get update` to refresh the list of packages.

If you were to install extra software without additional instructions, then `apt` would just use the newest versions available and thus quickly upgrade your whole system to the newest versions which is likely not what you want.

A mechanism called `apt-pinning` is used to instruct `apt` which priority each source should have. The administrator needs to state preferences in `/etc/apt/preferences`. The format of this file is described badly in the manpage for `apt_preferences` and a lot better in the *apt-howto*.

`apt` uses these preferences to decide where to get a package from; also you can (and occasionally must) tell `apt` explicitely which version of a package you want: *pkgname=version* requests a particular version, while *pkgname/stream* selects the version from a particular distribution stream.

You can use `apt-cache policy` to figure out what versions of a package are available and which one `apt` would pick according to the current preferences.

The main problem you will encounter very soon when mixing distribution streams is unmatched or conflicting dependencies: newer versions of programs will either require a lot of collateral updates to fulfil the dependency requirements or may conflict with existant packages badly enough to make an installation impossible.

All possible solutions at that point are unpalatable: either choose a lot of more bleeding-edge software than you'd actually like, stick with the old versions of everything in stable, or try to build the package locally. More about the last choice in the next section.

# 7 Building Your Own Packages

## 7.1 Custom Kernel

At some point you will want to customise the kernel on your system to your specific needs. You can ignore the Debian way of doing this without major penalties, but as there's just a few extra steps you won't gain anything by doing so.

What you need is either a stock kernel from `kernel.org` or a Debianized kernel source package (these are called `kernel-source-`*`version`*).

The differences between stock and Debian kernels are occasionally substantial, but unlike some other Linux distributors Debian keeps all changes open and documented: If necessary you can install `kernel-patches-`*`version`* which includes only the diffs between the stock and the Debian kernel version.

Unpack the kernel source in `/usr/src` and configure it in your preferred way, eg. by running `make menuconfig`.

To make a kernel the Debian way, you need the package *kernel-package*: it provides a tool called `make-kpkg` which builds the kernel binary and wraps it into a normal Debian package which includes maintainer scripts that take care of updating your system environment (e.g. module configuration, boot loader configuration etc.) upon install.

You need to run `make-kpkg --revision=`*`yourrevision`* `kernel_image`, and after that finishes you install the resulting Debian package with `dpkg`. The kernel configuration is stored in `/boot/config-`*`version`*, which is very useful if you want to prime a kernel source config quickly from some other kernel image.

## 7.2 Other Packages

This paper won't go into how generic software can be Debianized (for such information see the Policy Manual[7] and the Developer Reference Handbook[8]), as these are very extensive topics.

However, rebuilding an already Debianized package locally with some fixes is a simple proposition which can also help when mixing distribution streams.

Every Debian package comes also as an automatically buildable "source package", which includes the specification of dependencies for the build process.

In the simplest case, all you need to run is `apt-get build-dep` *`pkgname`* followed by `apt-get source -b` *`pkgname`*. The first will install all the build dependencies, while the second command will download the source package and build it in the current directory. The result should be a package file *`pkgname_someversion`*`.deb` in the current directory.

Most of the time you will want to make changes to the package before it is built: `apt-get source` will leave the unpacked sources in a new directory named by package name and version. You will find a directory `debian` within that build directory, which is where all Debian-specific information regarding that package is kept.

If you need to adjust the build process, look at `debian/rules`: this makefile governs the configuration and the build process.

If you want to give the package a different version (to distinguish it from official packages), you will have to edit `debian/changelog` and add a new entry with your local version number. As the format of this file is very specific, you may want to

use one of the user-friendly frontends for that task: `debchange`. To change the version without producing unexpected side-effects, just attach a trailing number to the current version: for example you might change a package that had version "2.05a-11" to "2.05a-11.1".

To build the package, either run `debian/rules binary` or use the nice frontend `dpkg-buildpackage -uc -us`, both while being in the build directory.

Changing the package version in the way outlined above is useful because it keeps `apt` from replacing your locally built version of the package with the upstream version immediately (`apt` would prefer the upstream version, even though they have the same version number).

The second method to keep a package fixed at a particular version is to put it "on hold". `dselect` and the `apt`-frontends can do that directly, but if you are not using any of these you can also achieve this with `dpkg`: Run `dpkg --get-selections`, redirect the output to a file which you edit to change the intended state of the package from "install" to "hold", and then run `dpkg --set-selections` with the updated file as input.

# 8 Conclusion

There is a multitude of tools which can help you to use your Debian system most efficiently. This paper has outlined some of the most common and important tasks of a system administrator and covered the tools available for these tasks.

As an administrator you want to get a job done, with as little overhead as possible, and Debian caters well for that; Software maintenance was and still is a large part of system administration work and Debian makes this relatively easy.

But this easiness does not lock you into a colorful-but-inflexible environment where you *have* to use the provided tools: with Debian you can run your system "on the bare metal" as much as you like or use the available frontends *selectively* - and this is a freedom of choice I would not want to miss.

# References


[1] Alexander Zangerl. Debian GNU/Linux and System Administration
`http://people.debian.org/~az/sage-2003/`

[2] The Filesystem Hierarchy Standard.
`http://www.pathname.com/fhs/`

[3] The Debian Project
`http://www.debian.org/`

[4] Bug Tracking System
`http://bugs.debian.org/`

[5] Manoj Srivastava. Why Linux? Why Debian?
`http://people.debian.org/~srivasta/talks/why_debian/`

[6] How many Debian Developers does it take to change a lightbulb?
`http://en.wikipedia.org/wiki/Lightbulb_joke#Debian_Linux_developers`

[7] The Debian Policy Manual
http://www.debian.org/doc/debian-policy/

[8] Debian Developers Reference
http://www.debian.org/doc/developers-reference/