Debian GNU/Linux and (Large Scale) System Administration

Alexander Zangerl
az@debian.org

Abstract

This paper provides an overview of the Debian GNU/Linux distribution and some of the more important aspects of using Debian as a System Administrator. Debian presents a successful example of a very large volunteer-based community effort in the field of software development and integration, which despite its size and global nature nevertheless manages to adhere to some very strict guidelines that make the resulting distribution the most consistent system currently available.

For a system administrator consistency, automation and simplicity are crucial aspects of all software that is to be administered; all all three areas are addressed quite extensively by Debian which makes it a very good choice of an OS distribution. This paper highlights some of the advantages and benefits of using Debian in the areas of deployment and ongoing administration of server systems.

1 Introduction

1.1 What is Debian GNU/Linux?

One possible short answer is "The Universal Operating System", which is one of the project's goals. The phrase "The Distribution for Sysadmins by Sysadmins" is frequently seen, too.

The Debian GNU/Linux software distribution is the current product of the Debian Project, which includes the Linux OS kernel and thousands of prepackaged applications. Various processor types are supported to one extent or another, so the claim of being universal is not far off. Ports of Debian to other OS kernels like GNU Hurd and FreeBSD or NetBSD are currently in the works, extending the Debian methodologies to other UNIX environments.

The Debian distribution is built modularly from *packages* each providing some functionality, and so far Debian looks just like any other OS distribution. The more intriguing aspects of Debian (both distribution and project) are in the realms of philosophy and quality control, e.g. how consistency is enforced via fine-grained dependency specifications between packages.

The main goal[1] of the Debian Project is to provide a consistent high-quality operating system distribution that is composed entirely of free software (as in "free speech", not "free beer"[7]). Debian is the only significant non-commercial distributor of Linux.

A lot of development effort has been put into the "meta-levels" of software development: how to integrate software by different developers as seamlessly and as consistently as possible while still allowing the administrator to be in charge of all this.

The Debian Project is a worldwide group of volunteers, open for every software developer and user to contribute their work: anybody can join the project after going through a quality-assurance process where the applicant shows his skills and familiarity with Debian's goals and governance.

1.2 Debian for large scale environments?

Debian is very suitable for deployment in large scale environments, because it eases some of the more burdensome tasks of administration.

Keeping systems both up-to-date and consistently configured, doing updates seamlessly and remotely and offering the same environment on a multitude of architectures are some of the positive characteristics to mention; these and others will be covered throughout the remainder of this paper.

1.3 History

The Debian Project was founded by Ian Murdock in 1993[2] when he perceived the need for consistency and structured development in the important field of Linux distributions. His statement[1] of the goals and intentions still form the core of Debian's philosophy today, although the project has gone through some very massive changes.

His key point to make was that distributions are crucial to the commercial success of Linux, but that they get too little attention from developers. The reason for this is that it is neither easy nor glamorous to construct a distribution, and it also requires a great deal of ongoing effort from the creator to keep the distribution bug-free and up-to-date.

The Debian Project was intended as a non-commercial group of people dedicated to tackling this unglamorous task to the benefit of the global Linux community. Initially the project started as a small group of Free Software hackers and grew gradually to become a global, large and well-organized community of developers and users.

At the end of 1995 there were about 60 developers, and various pre-1.0 releases had been made. In June 1996 Debian started using code names for its releases, taken from the characters of the movie "Toy Story". The first such release, Debian 1.1 "Buzz" was based on the 2.0 kernel series and contained 474 packages.

Debian 2.0 "Hamm" in 1998 was the first multi-architecture release, including support for the Motorola 68000 architecture. 1500 packages were maintained by about 400 developers. By the time of the current release, Debian 3.0 "Woody" in July 2002, the lists had grown to about 8900 packages, over 900 developers and 11 different supported architectures.

1.4 Social, Philosophical and Organisational

Obviously a project as big as this could not work cohesively in a totally anarchic environment; therefore Debian developed quite some organizational and technical means to keep the project open, free and focused on its goals. These three sets of documents form the foundation and baseline:

The Debian Constitution: The constitution[3] describes the organisational structure for formal decision-making in the project, as well as the decision-making process itself; basically it explains how developers or package maintainers, the elected project leader and the technical committee interact and ascertains that Debian won't get into any kind of "hostile takeover" situation.

The Debian Social Contract: The social contract[4] is a set of commitments that the Debian developers agree to abide by, including Debian's promise to remain 100% free software. This commitment to free software is one of the absolute cornerstones of the project.

A major component of the social contract, the Debian Free Software Guidelines (DFSG), has been adopted by the free software community as the basis of the Open Source Definition[7]. The DFSG addresses the crucial issue of what can be considered free software by describing what a free software license must allow and must not restrict, and only software meeting these criteria can become part of Debian. Strict adherence to the DFSG is one of the major reasons for many developers to join the project, and occasionally Debian is also able to convince other software authors to adjust their licenses sufficiently.

As such the DFSG is a set of rules with very little potential for compromise; however, Debian acknowledges that not all software is free and provides some infrastructure to build and use non-free software in a manner that meets at least the *technical* policies. Such non-free software is not part of the Debian distribution proper, it just shares the packaging format.

Together with the constitution the DFSG provide a user with a reasonable guarantee of continuity for the distribution, a lack of which would be a very limiting factor for adoption of Debian in commercial environments.

The Debian Policy Manual: The policy manual[5] describes the policy requirements for the Debian GNU/Linux distribution. This includes design and setup issues of the operating system, technical requirements that each package must satisfy to be included and similar. The policy manual explains the environment a user of any Debian system can expect and as such is the most important technical document describing Debian. Among other things, the policy requires strict adherence to the Filesystem Hierarchy Standard[8]; compliance with the Linux Standards Base[9] is currently in the works which will improve Debian's attractiveness as platform for commercial development.

2 Debian from a System Administrator's point of view

The following is a brief listing of key characteristics of Debian and some consequences as applicable to large scale environments. Most of these are discussed in greater depth and more technically in the following sections.

2.1 The system administrator is in charge!

Debian does not treat the administrator as a moron who has to be protected from himself. This is both an advantage as well as a disadvantage, as shooting yourself into your foot *is* possible. However, as a system administrator one probably prefers the capabilities and flexibility of a mature set of tools over the fool-proof but very limited nature of a padded-cell environment.

No configuration is hidden from the administrator, and configuration data is almost exclusively kept as plain text files. There are frontends for most major maintenance activities, but one is not locked into having to use a non-automatable configuration frontend. In fact, ease of automation is very well taken care of and maintenance of multiple systems is made easy.

2.2 Debian is maintained by its users

Debian is neither an academic, ivory-tower development nor a commercial endeavour that needs to sell you something to survive. Debian is used to make Debian, and both users as well as developers contribute to what is perceived as the Best Current Practices in the field of OS distributions; as the global set of servers used for making Debian is absolutely heterogenous and of fair size, most aspects of large scale environments are experienced by the developers themselves, too.

2.3 Breadth of support

Debian is quite well supported by the usual means of communication in the Free Software community: besides lots of mailinglists dedicated to various aspects and various IRC channels there is a comprehensive and open set of documentation of Debian's inner workings. Some specialty areas are covered by Debian subprojects such as "Debian Jr.".

Most of the wide range of organizations and individuals using Debian do their share of improving and maintaining Debian's quality by providing feedback and bug reports using the extensive Bug Tracking System[10]. Also there are community websites run by users of Debian, the most useful of which are DebianPlanet[12] and DebianHelp[13].

2.4 Availability

So far Debian has been released for ten architectures using the Linux-kernel and ports to other architectures are in progress. Ports to other kernels such as GNU Hurd and FreeBSD are being developed right now. For a heterogenous set of machines this gives you the advantage of using the same system on all machines which will reduce maintenance efforts significantly.

Debian packages can be installed from all kinds of storage media as well as from a well-established network of archive mirrors which make packages available via the Internet; needing installation media for each and every system you want to deploy Debian on is not necessary which makes multiple installations less taxing.

2.5 Continuity and Consistency

A Debian system can be upgraded seamlessly, with dependencies between packages being handled automatically. You have the choice to upgrade only certain packages or to upgrade everything to the latest state. Depending on your choice of distribution stream, new versions of packages may be available every day. Upgrades work while the machine and the services run, without the need to reboot it (except for activating a new OS kernel) or to switch into single-user mode, which obviously is a major advantage for production or server machines. Upgrades losing their drastic nature and the simplicity of keeping large numbers of systems up to date are the main benefit for large environments.

Debian's package management system offers fine grained dependencies for all packages, a menu-system that connects common programs to the menus of most window managers, documentation for all packages, support for alternative programs and virtual packages providing certain functionality.

All Debian software is packaged by a fairly coherent group of developers, which means that you can be assured that issues regarding complicated dependencies have been worked out already. Several tools are used to help connecting packages to each other, presenting the user a well maintained and consistent system. This integration between packages makes a Debian system very robust.

Each Debian package comes preconfigured or with a configuration tool, but you are free and encouraged to reconfigure or fine-tune the packages to your needs, with or without using the supplied configuration interfaces.

Maybe the most important distinction of Debian is that the package management system will never throw away your local configuration changes when upgrading a package. That way Debian caters for both the specialist who needs all necessary freedom to tinker as well as for the casual user who benefits from mostly sensible defaults being provided. Together with configuration in text files that can be modified automatically this vastly aids administration of multiple machines.

2.6 Remote Maintenance

Related to easy upgradability, Debian also can be completely administered remotely. This includes configuration and package maintenance as well as installation or removal of new packages. Debian takes care of not cutting off your remote access even if you upgrade or reconfigure the very package providing the remote access.

2.7 Stability or Bleeding Edge: Your Choice

The Debian distribution consists of three streams, offering you a choice that ranges from rock-solid stability to bleeding edge software.

- The "stable" distribution is the latest released variant. Releases do not happen very often as the quality control process is time-consuming and thorough. Debian stable is updated only with fixes for serious security issues and does not change whatsoever otherwise. This also means that there is quite some difference between software versions available in stable and the latest versions which is often perceived as one of Debian's main shortcomings.

- The other extreme is "unstable", unreleased and development software. Unstable is not as bad as the name suggests[1] but it is a moving target and suffers from occasional breakage.
- Middle ground is covered by the "testing" stream, which consists of the software that is to be tested before becoming the next released distribution. Testing is updated automatically from unstable with packages that meet certain quality requirements.

An extremely useful feature of the package management system is that you can mix software from the different streams with relative ease. That way you can keep your foundation as stable as possible while including certain necessary packages in their newest versions from testing or unstable.

2.8 Quality, Stability and Security

Packages in Debian are maintained by a person or a group of people responsible for the quality of the integration with the other parts of Debian.

Maintainers (who are quite frequently also the original authors) generally have a strong personal interest in each package they maintain, as they mostly volunteered to maintain it because they wanted to use it themselves. This results in high quality work by highly motivated, and generally technically skillful people, governed by the overall policy and technical requirements.

The Debian Project pays more attention to quality and testing than to releasing often, and each released version of Debian has been well tested over a long period and all major bugs have been removed. The entire distribution is tested by at least all active developers and interested users from the beginning.

Debian has a dedicated security team whose members keep the distribution up to date with regard to evolving security isses. Security updates are issued usually for the last two released distributions, backporting fixes to the "old" versions available in the released distribution, and in very serious cases also for the testing and unstable distributions. These security updates are made available via the Internet and can be installed easily using the normal package management system (even fully automated and completely unattended, although that is not the suggested mode of operation). This system enables you to keep your systems safe with minimal hassle and without needing to migrate away from your established and trusted software versions, which is of especial importance in server and large scale environments.

2.9 Not *Too* Hard to Install

The Debian installation process may be less pleasant to look at with regard to fancy eye candy than others, but it is nevertheless very simple and is also improved constantly.

However, occasionally the finer details of administering a Debian system *efficiently* can elude the newcomer as the functionality available the an administrator is very rich.

As per Debian's overall philosophy the human is to be in charge at any time, and the installation process reflects this well: you can get a shell to do your low-level magic, you are not forced to follow the suggested sequence of steps and you can fine-tune the initial installation to your needs easily. Also the installation can be done from various media, including cdroms, floppies, a DOS-environment and also completely over the network, via `ftp`, `http`, `smb` or `nfs`. Network booting is supported for hardware architectures which offer this in some standard way.

2.10 Lots of software packaged

Debian currently comes with over 8900 different pieces of software, all of which are free and integrated quite consistenly.

For some commercial software which can't be included in Debian there are *installer packages* that will download, install and configure such software as consistent with Debian's rules as possible.

[1] Unstable has the permanent code name "Sid": in "Toy Story", Sid was the boy next door who destroyed toys.

For all packages of the distribution the whole source is available, which also includes the necessary changes and setup to rebuild the package automatically in its Debianized form. The changes that transform software into a Debian-conformant package are kept separately from the pristine original source, and all packages can be (re)built locally on any Debian system, which allows for easy customization and deployment of your customized packages in your environment.

Everything in the entire distribution is free according to the DFSG which also means that everybody can improve packages and still distribute them; in fact there is a number of distributions that use Debian as a base.

2.11 Constantly evolving procedures

Debian is constantly under development. Whether this is a benefit or nightmare depends on your point of view: if you are using Debian for a server environment where stability is paramount, you would very likely use the "stable" stream and not notice any changes in behaviour inbetween major releases, so new developments won't be visible to you until the software has matured in the testing process prior to the next major release.

If you use "unstable" you may find out that various minor aspects of administering your system do change fairly often, for example new frontends for common procedures, automation support being finetuned etc. However, these changes apply mostly to administration of your system without direct effects on end-users.

3 Packaging and Package Management

Packaging software for Debian can be a very challenging task, as it usually involves quite a lot of changes in the original software to meet Debian's policy requirements. A package consists of the actual files making up the package, control information and maintainer scripts. All this is wrapped up in a file with the extension `.deb` which is basically an `ar` archive with two tarballs inside.

All Debian packages are available in source-form which consists of the original source and the necessary Debian-specific information, code changes and integration scripts which are always kept separately from the original sources in form of a `patch` file against the original source. If a package is "Debian-native" (the Debian maintainer *is* the original author) then all information is kept in the main source archive.

Usually multiple binary packages (for the various supported architectures) are built automatically from the source packages on the machines where Debian "build daemons" run (a heterogenous set of globally distributed systems belonging to the project) and made available for download. Usually one installs such binary packages only and rarely bothers with source packages unless necessary.

Architecture-independent software like `perl` scripts comes in a single binary package and there are source-only packages without binary package at all, too: for example software that needs to be built against a specific kernel is distributed that way.

3.1 Package Information

The control information of a Debian package anchors that package in the context of the distribution and contains meta information like this:

```
Package: hello
Priority: optional
Section: devel
Installed-Size: 45
Maintainer: Adam Heath <doogie@debian.org>
Architecture: i386
Version: 1.3-16
Depends: libc6 (>= 2.1)
```

```
Filename: dists/potato/main/binary-i386/devel/hello_1.3-16.deb
Size: 19488
MD5sum: a9defe5d44eee6586a2aeac64d0d3224
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting.  It
 allows nonprogrammers to use a classic computer science tool which
 would otherwise be unavailable to them.
 .
 Seriously, though: this is an example of how to do a Debian package.
 It is the Debian version of the GNU Project's 'hello world' program
 (which is itself an example for the GNU Project).
```

The most important part of the control information are the name, the version and the dependency specification.

Debian package names have to follow a policy which occasionally makes the names arkward but works pretty well in general and serves to categorize packages. For example, `perl` packages must be called *something*-perl, library packages are always called lib*something* and so on. Software versions consist of the original version information, a hyphen and the Debian revision, which is used to distinguish the Debian version from the original software as these can differ a lot.

3.2 Package Relationships

Well-defined dependencies are one major part of making Debian's pieces integrate properly. A package must declare all other packages it needs for proper operation, packages it conflicts with or suggestions for enhancing its functionality. Certain packages are marked `Essential` and guaranteed to be available; dependencies on these need not be stated. The dependency vocabulary is quite powerful and knows the following relationships (most of which allow to specify software versions, too):

Depends and Pre-Depends: Absolute dependency on another package for the package's function. As with most other dependency keywords, ranges of versions can be used, specifying ranges for an acceptable version of the dependee. Pre-Depends is a special form which identifies another package that must be present before the respective package can be even unpacked, let alone configured and installed.

Recommends: A package that the maintainer judges would be a good idea to use with the respective package, but which you might conceivably not want to do.

Suggests: A weaker suggestion of related packages to consider, which may or may not enhance the package's functionality.

Conflicts: Conflicting packages either provide the same functionality in an incompatible way or provide the same files (a file on your system must belong to at most one package).

Replaces: A package replaces another package when files installed by the previous package are removed and (in some cases) over-written by files in the new one. This is very useful for renaming packages while providing an automatic upgrade path.

Provides: A Package provides another package when all of functionality of the provided package is incorporated. This mechanism is used to declare virtual packages like *mail-transport-agent* which specify functionality that can be provided by lots of different actual packages. Such providing packages must be sufficiently compatible so that the basic functionality is guaranteed to be available regardless of which particular package is actually chosen; for example all *mail-transport-agent*s do offer a legacy `/usr/lib/sendmail` interface.

3.3 Package Status

On a machine running Debian, for each software package available the control information of the package and some status are kept locally. The status information includes an installation state, a selection state and some flags.

A package can be in states *not-installed* (not present), *installed* (is installed and working), *half-installed* (the installation did not finish properly), *unpacked* (but not configured), *half-configured* (the configuration has not finished properly) or, finally, in state *config-files*, which means that only the configuration files are left on your machine.

Selection states are kept separate from the package status and include *install* (you want to install a package), *deinstall* (you want to remove the package's files except its configuration files) or *purge* (remove the package and the config files). Section 4 explains the fine details of the various states.

Finally there are two flags, *hold* (a package will not be touched by `dpkg` unless explicitely told to) and *reinst-required* which indicates a state so thoroughly botched that the package installation must be completed first before any other action may work again (even removal).

3.4 `dpkg`

Debian's package management consists of multiple layers of tools, of which `dpkg` is the lowest. `dpkg` implements the core functionality, it deals with installed packages and existing package files, but has no idea of different package sources, distribution streams or the like.

`dpkg` interprets the control information of packages and acts accordingly; for example, if you are installing a package that conflicts with another already installed package, `dpkg` will flag an error regarding the conflict. As usual with Debian, you can override such safeties easily and occasionally this even makes sense; section 7 will show some example uses.

`dpkg` keeps track of the state of packages on your system in the file `/var/lib/dpkg/status`, which is very likely the worst single file to lose on a Debian box.

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

- `dpkg -L` *name* shows the files belonging to package *name*.
- `dpkg -S` *file* searches for a package that contains *file*.
- `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.
- `dpkg -i` *packagefile.deb* installs the package from file *packagefile.deb*. This fails if there are unsatisfied dependencies or conflicts.
- `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*.

3.5 `dselect`

`dselect` is the old-style menu-driven interface to the package management system, on top of `dpkg`. `dselect` supports various access methods (direct, ftp, cdrom etc.) and interprets the `Recommends` and `Suggests` control fields.

Unfortunately `dselect` suffers from a very unintuitive interface and therefore has largely been replaced by `apt` and its frontends.

3.6 apt

apt stands for Advanced Package Tool, which is working on top of dpkg. apt supports various access methods, knows about networked mirrors of the Debian distribution archive, supports mixing of packages from different distributions and also private repositories.

apt is command-line driven like dpkg and does currently ignore the Recommends and Suggests package relationships; thus it is not a complete replacement for dselect, although most people use it successfully as exactly that.

apt's operation is mostly governed by a list of package source specifications in /etc/apt/sources.list. This list can specify distribution cdroms, local directories with packages, networked mirrors of the Debian archive or other software repositories that offer Debian packages. apt keeps a local cache of the lists of packages available from these sources.

The two most frequently used programs from the apt-suite are apt-cache and apt-get.

apt-cache offers operations on apt's package cache, e.g. apt-cache search *query* for doing a full text search on all available package control information, apt-cache show *name* to display information about package *name* or apt-cache policy *name* to show the available and installed versions for package *name*.

apt-get works like an extension of dpkg and as such offers operations like apt-get install *name* to install a package, apt-get remove *name* to remove or purge a package. To update the cache of available packages, one would run apt-get update, and to upgrade all installed packages to the latest available versions apt-get upgrade. Different from dpkg, apt-get follows dependency chains automatically and includes packages that are necessary for fulfilling the request you have made. Conflicts are resolved in a similar manner, by removing packages (recursively); but of course the administrator is prompted before any action is taken on packages beyond the explicitly requested ones. For the sake of automation this safety net can be switched off, too.

3.7 Frontends for apt

As mentioned before the situation for interactive package management is not perfect, as dselect is quite old by now and not very user-friendly to operate while apt is command-line based.

Therefore there are several[2] efforts under way to provide more user-friendly frontends to the proven apt layer. The two most likely successors of dselect are aptitude and deity, both offering a curses interface. Other popular frontends include graphical tools like synaptic, kpackage, gnome-apt or stormpkg.

4 Configuring and Keeping your systems up to date

The Debian Project believes firmly in software consistency and automation and the package management processes reflect this well; it is therefore very simple to keep your systems up to date and there is a lot of automation and scripting support for many administration tasks.

4.1 Package (Pre)Configuration

All Debian packages must be preconfigured for immediate use or must provide a suitable configuration interface to the user during the installation process. Of course you are not required to stick with this often basic default configuration but in fact you are encouraged to adjust the run-time configuration as much as you like. The package management system supports you in this:

- Configuration files always reside somewhere under /etc/. This is a consequence of applying the FHS[8].

[2] Free Software at work: (re)inventing wheels in parallel.

- An administrator's changes to configuration files are guaranteed to be preserved during a package upgrade. This is one of the most important features of Debian.
- Configuration files are preserved when a package is removed and only deleted when a package is purged (which must be requested explicitely). The idea behind the distinction between removed and purged packages is that you may have spent quite a lot of effort on fine tuning a particular package, and this should not be lost even if you have to remove that package. If you decide to reinstall the package later, the configuration is kept or merged with the default config the package maintainer supplies.

Debian ensures that your manual configuration changes are never lost in two different ways, both entwined with the package installation process.

A package can declare included configuration files as *conffile*s which makes `dpkg` handle preserving changes between the current state of the configuration file and the version which is supplied with the newer package.

Alternatively, the package can supply so-called *maintainer scripts*, which then bear the responsibility to correctly create, update and maintain the necessary configuration files and remove them on purge. These scripts must be idempotent and must not overwrite or otherwise mangle the user's configuration without asking.

When `dpkg` is tasked with installing a package, it first extracts the control information from the package file. If all package relationships are ok, it proceeds with running maintainer scripts if available: if an older version of the package is installed, its `prerm` script is run first. Then the `preinst` script of the new version is executed. Next step is to actually unpack the data files making up the package, while old files are backed up. When this has succeeded, an old version's `postrm` script is run.

After that `dpkg` enters the configuration phase: it unpacks the package's *conffiles*, backs up old ones and looks for differences. If differences are detected, `dpkg` prompts the administrator with a set of choices regarding that difference. The safe default is to keep your changed file and save the new different version in a separate file for later examination.

Finally, if a package supplies a `postinst` script, it is run at that stage. When all this completes successfully, `dpkg` removes the backups of old files and files that are no longer present in the new version of the package.

One small issue that bites administrators new to Debian occasionally is that the removal of a *conffile* is deemed a configuration change by the administrator and, as such, is preserved during an upgrade: removed *conffile*s will not automatically reappear when you upgrade or reinstall a package. The solutions for this are to run `dpkg` with the option `--force-confmiss` or to purge the package before reinstallation.

The package management is able to update a running program, which is possible because the kernel and filesystem support replacing files even while they are in use. Besides that Debian relies heavily on the tool `start-stop-daemon` which can be used by the maintainer scripts to stop, reload or restart daemons when necessary.

The net result of all these steps happening behind the scenes is that all you need to do to keep your system up to date is to run `apt-get update` followed by `apt-get upgrade` occasionally. `apt` will then look for packages needing updates, retrieve the appropriate packages and `dpkg` will take care of installing the newer versions without disrupting your system and keeping service downtime to an absolute minimum. This seamless operation is one of the major benefits of using Debian in environments with multiple systems.

4.2 Remote Maintenance

Debian systems can be fully administered remotely, which includes installation or removal of all kinds of packages, even as fundamental ones as `libc`, the dynamic linker or the kernel itself. You can do a full operating system upgrade remotely, without the need to reboot except for activating a new OS kernel, and in general without the services on that machine being unavailable for more than a few minutes. If

you are administering your system remotely, for example using `ssh`, then it is not a problem to upgrade your `ssh` server: the package management system takes extra care not to terminate your active session.

Such upgrades are very simple and seamless; you do not have to wait for major releases to update particular packages because that can be done easily and (almost automatically) on the fly, whenever the need arises and without risking major downtime or needing physical access. As such, most problems associated with major upgrades (like higher probability of side effects because all of the system is changed at the same time) are greatly reduced with Debian.

The package `cron-apt` offers one set of scripts for keeping your system up to date automatically, although usually you would set it up so that it only downloads new packages and alerts you.

5 Availability and Accessibility of Debian

5.1 Sources

The Debian distribution is published in two major ways: as official cdrom images for the stable distribution and online via a list of networked mirrors for all distribution streams (stable, testing and unstable).

The distribution archive for each distribution is organized similarly regardless of medium, divided in these major areas: *main*, *contrib*, *non-free* and *non-US*. Main includes software meeting the DFSG[4] in all aspects (including tools needed to build the software), while contrib holds software that is in itself free but requires some non-free software for its operation or build process. Non-free is only available via the network mirrors and is not really part of the Debian distribution: software which does not meet the DFSG goes there. Non-US holds software that must not be exported from the USA because of cryptographic algorithms (although this is changing nowadays) or because of restrictive software patents. The servers offering non-US are mostly located in countries which do not recognize patents on software algorithms.

The choice between the various sections is made by listing respective sources in `/etc/apt/sources.list`. To choose between the different distributions, most of which provide the same packages in different versions, a method called `apt-pinning` is employed: the administrator can state relative preferences of distributions in `/etc/apt/preferences`. `apt` decides on the basis of these preferences (and command-line arguments) where to get a package from.

This allows fairly easy mixing of the different streams according to the administrator's needs: using `apt-pinning` `apt` can be configured e.g. to use as much software as possible from "stable" but still allow the administrator to include specific packages from "testing". `apt` will take care of satisfying the dependencies with as little side effects as possible.

For the stable distribution there is a separate network archive which provides security updates. This server is located at `security.debian.org`, and is not (officially) mirrored to minimize the probability of Trojan horses being distributed.

There are also some other sources of Debianized packages, which are not part of the main distribution for a variety of reasons; the site `apt-get.org`[11] is dedicated to provide listings of the goods offered by such non-standard repositories.

While `apt` comes with a good configuration interface for specifying your preferred sources (after all, `apt` is just another Debian package), there are other tools that can be helpful when selecting your package sources: `apt-spy` and `netselect` for example can produce a `sources.list` file based on bandwidth tests.

One interesting tool related to availability is `alien` which allows the conversion of LSB[9], Red Hat, Stampede and Slackware packages into Debian packages (and vice versa). Apart from untranslatable Debianisms it works quite well.

5.2 Multiple Installations

Rolling out multiple systems efficiently is not hard with Debian: the initial installation step, getting the hardware set up and the most minimal basic system on the machine, is usually the only step that has to be done interactively, and after that initial step you can do everything remotely and pretty much everything in an automated fashion, too.

Debian interacts very well with a variety of different systems for multiple/batch/unattended installation to simplify that initial installation step; currently the packaged batch-installation tools include `fai`, `system imager`, `replicator` and `partimage`.

But also without using any of those specialized (and rather complex) systems you can deal with most of the necessary migration or cloning tasks with just the basic Debian tools. For example to migrate your software selection choices onto another machine there are various methods available. The simplest but least perfect is to use `dpkg` to transfer the selection status and `dselect` (or another `apt`-frontend) to act upon these selections: you would run `dpkg --get-selections` on the source machine and copy the resulting file onto the target machine, where `dpkg --set-selections` would update the selection status. Finally you would have to use `dselect` to install the selected packages.

Likely a better method is to use `jablicator`: it creates a meta-package with information about the currently installed packages. You can install this meta-package on another system, and `apt` will install any necessary packages automatically.

In a pinch you might also just `rsync` everything onto another box; in this case you'll notice positively that the task of copying just the relevant parts but not machine specific information is made easier by Debian's adherence to the FHS as configuration files won't be anywhere outside `/etc/`.

The capability of installing software made available via a network allows multiple installations without needing multiple installation media. To lighten the network load when doing batch installations, Debian offers tools to create local and/or partial mirrors for subsequent installations: `apt-move` can be used to organize a collection of package files into a proper archive hierarchy as is used in the official archive. Alternatively, `apt-proxy` automatically builds an archive mirror based the requests which pass through this proxy.

As mentioned in section 3 ongoing administration of a big set of systems running Debian is also fairly easy as you can either use remote access to administer systems individually (interactively or non-interactively at your choice) or centrally generate configurations, push them to particular machines and then rely on automation support offered by Debian to further activate these configurations if necessary.

A useful tool operating inbetween these two approaches might be `dsh`, a simple `ssh`-based noninteractive shell which is especially useful in conjunction with the non-interactive mode of operation of the `dpkg`/`apt` tool chain.

5.3 `debconf`

Besides offering replication techiques for bulk installations, Debian provides very flexible methods for batch-configuration of software, too, which mostly revolve around the `debconf` specification and tool.

If a package needs to prompt the user for input during execution of the maintainer scripts, then it can do this directly or by using `debconf`. The major problems with the directly interfacing, home-grown scripts are too little consistency and no memory of past answers.

To address these needs, `debconf` was developed: a configuration management system which allows for multiple different backend databases to store user's choices, offers a variety of user interfaces to be presented and is generally way more flexible than home-grown scripts. It offers a unified configuration interface, front-end independent configuration scripts, easy storage of the configuration information and fully supports localization.

The basic data element from `debconf`'s point of view is a question and its priority wich is based on whether there is an acceptable default value available or not. The administrator of a system can decide e.g. whether low-priority questions should be hidden and answered in the default manner or not.

`debconf` supports multiple frontends which do the actual interaction with the user, currently including `dialog` (for text-mode menus), `readline` (console- and line-oriented), `gnome`, `web` (`debconf` runs a small webserver), `editor` (`debconf` presents you with all information in a file and runs your favourite editor[3]) and `noninteractive` (`debconf` never interacts with you at all and instead uses the default answers everywhere; for non-default answers you have to pre-populate the `debconf` database), which obviously is most suitable for bulk installations.

The answers to questions are stored permanently in some kind of backend database. Supported backends currently are flat files, directories, LDAP directories and pipes from generic programs, with other drivers being developed; the default is to use the file `/var/cache/debconf/config.dat`. In addition to these direct backends `debconf` also supports stacking of backend databases: multiple sources are queried until a match is found.

`debconf` is most beneficial for multiple systems where similar configuration for a particular package is needed: one can pregenerate the database of answers, copy it to the target machine or make it available via a network (LDAP or similar) and simply run `apt-get install packagename` on the particular machine (remotely or locally) and the software will be configured in exactly the intended fashion. For software already installed a run of `dpkg-reconfigure packagename` takes care of activating your `debconf` choices.

6 The Daily Niceties

This section lists some selected properties of Debian which the author regards as very nice in day-to-day operations.

Debian guarantees (via the FHS) that `/usr/local/` is never touched by the package management system, but of course the default `PATH` settings include the canonical spots under `/usr/local/` first.

Most configuration systems in Debian are built with automation in mind: it is very rare that you have to use things like `sed` to update config files. Instead, most important packages have their configuration split over multiple files, which are then combined into a single config file by an `update-`*something* script. For example, `syslog-ng`, `logrotate`, `logcheck`, `emacs` and the `modutils` package use this approach. Other packages like `inetd` can be administered fully with their respective `update-`*something* programs. Thus one does not *have to* edit `/etc/inetd.conf` ever if one chooses so, but if one makes local changes manually these changes are guaranteed to survive all package updates.

A lot of tedious script execution is realized in such a modularized manner, too: a tool called `run-parts` is used to step through a whole directory of scripts in a defined order. The various `cron`s in Debian use this method, as does the X11 server startup and lots of other packages. This is really useful for automation purposes and when you have to adjust multiple machines: all you need is to copy a file in a particular spot without having to worry about parsing and modifying other files to include your customizations.

Default users and groups are well-chosen and the flexibility of the UNIX file access model is used to the utmost extent; for example the special files under `/dev/` have sensible and secure default ownerships and permissions, which makes things like giving a user permissions to use audio devices a single call to `adduser` to add the account to a particular group.

Debian's `syslogd`s come with logrotation pre-setup, as well as all other services that log extensively (like `apache`). Syslog facilities are setup sensibly across the distribution.

Debian uses a System V `init` setup with the policy specifying the interface that has to be offered by the startup scripts, so you can rely on daemons coming with a functional startup script (which is also a *conffile*: your changes will be preserved). `update-rc.d` is one administration interface offered for manipulation of the startup scripts (but of course manual configuration is possible, too) and their symlinks.

`apt-listchanges` is a small tool that shows new Debian changelog entries for a package when updating (but before installing) a package using `apt`. All Debian packages must come with a changelog file, and `apt-listchanges` allows you to scrutinize the changelog during installation and maybe abort the

[3] Citing the `debconf` documentation: "This is for those fanatics who have to do everything in a text editor."

installation if you forsee problems. A sibling tool, `apt-listbugs`, warns you of known critical bugs before installing a package.

7 Customization

Debian of course acknowledges that compile-time configurations can never meet everybody's requirements and therefore offers lots of tools to help one with customizing a system, ranging from tools for some fine tuning to a *ports*-like approach where everything is built locally.

7.1 Run-time configuration

The available tools for run-time customization include `dpkg-divert`, which is used to persistently override/remove/rename a file provided by some package. The idea is to let the package management system know about your changes, instead of just removing a file and wondering why it shows up again after the next package upgrade.

Occasionally it is necessary to override the package mangement with regard to file permissions and ownerships; this is done best using `dpkg-statoverride` which makes these changes persistent.

If you are mixing software from various streams extensively, then you may also need to make the package management disregard certain packages when upgrades are considered. One example situation would be if you know that no version of a package except the currently installed one works with your other tools. To address this, you can put packages "on hold" by using `dpkg` (or a suitable `apt`-frontend). The package management will not upgrade or remove that package at all except when explicitely instructed to.

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: tools whose functionality is offered by multiple packages are usually referenced by symbolic links in `/etc/alternatives/`, for example `editor` or `x-terminal-emulator`. 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.

Debian also offers a tool to install software on demand: `auto-apt` checks file accesses and automatically installs packages whose files are referenced.

If you need to fool the package management system, then this can be done using `equivs`: it allows you to build a dummy package which only contains dependency information. That way you can make the package management system accept your locally built software to provide functionality for the sake of fulfilling dependency requirements; of course you could also force `dpkg` to ignore missing dependencies, but that is drastically less elegant and stable a solution than using `equivs`.

7.2 Compile-time configuration

Debian packages are always available in source form, including all necessary information to build the packages unattended and on any machine running Debian. The maintainer of a package is not allowed to do any dirty tricks to produce an installable binary just on his own machine, but instead has to provide the appropriate infrastructure to make a local build straightforward. Debian packages therefore come not just with dependency information for the binary package but also with `Build-Depends` information which specifies what packages are necessary for building a package locally.

Building a Debian package locally involves just two simple steps: first one runs `apt-get build-dep` *packagename* to install all build-dependencies. Afterwards, the command `apt-get -b source` *packagename* downloads the sources for the package and starts the Debian build process. The result is an installable `.deb` package file in the current directory.

Rebuilding packages locally is necessary occasionally if one wants to run some software from the "testing" or "unstable" stream but without upgrading *all* the dependencies to that stream. Packages in "unstable" are mostly built against libraries available in "unstable" and specify the library version explicitely, thus installing such a package would require upgrading these libraries, too. Rebuilding the package locally binds the dependencies to the libraries available on the local machine and therefore no further upgrades are necessary. This approach does not work always, as the build-dependencies of a package might specify a minimum version of some requisite software which is not available in one's desired distribution stream.

Adjusting the build process further is not hard, either: the process is always controlled by the main Makefile `debian/rules` which must be provided and behave according to some specific rules.

For the building of the Linux kernel "the Debian way" there is the package `kernel-package`. It allows to build and install kernels in a way consistent with Debian's other policies, and is especially useful if you are often building kernels for test purposes or if your system uses kernel modules extensively. `kernel-package` wraps the customized kernel binary 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.

If you require many packages to be built locally, possibly with customizations and optimizations of your choice, then `apt-build` will likely be the tool of choice.

8 Conclusion

There can be no doubt that the Debian Project has made an enormous impact on modern software development and integration. It is one of the largest volunteer-based cooperative development projects, yet still keeps on track and in line of the overall goals: to be and stay completely, unequivocally free and non-commercial, and to provide the most stable, consistent OS distribution. Both these goals are becoming increasingly important in the distribution market as recent developments regarding other distributors have shown.

The open development process with its long test phase between releases produces to most stable distribution currently available. But bleeding edge software is available, too, and so Debian offers both rock-solid server reliability as well as fancy desktop playgrounds.

For system administrators the continuity provisions are of utmost impact, as one can upgrade a Debian system in perpetuity without ever having to reinstall from scratch. No other distribution matches Debian's upgradability, and few are as consistent to administer. None runs on so many different architectures.

In large scale environments all previously mentioned features are of course beneficial, but further features like extensive automation support, non-interactive installation and configuration and good remote administration capabilities are of major importance, as is the possibility to easily customize packages and deploy these onto a large number of systems efficiently.

That Debian is available on a multitude of different platforms ranging from handhelds like the Ipaq, via PC and workstations (HP, Sun Sparc) to "big iron" systems (IBM S/390) can also positively affect a heterogenous system environment.

The most concise summary in the author's opinion: "Debian gives you the freedom of choice."

References

[1] The Debian Manifesto. `http://www.debian.org/doc/manuals/project-history/ap-manifesto.html`

[2] Debian History. `http://www.debian.org/doc/manuals/project-history/`

[3] The Debian Constitution. `http://www.debian.org/devel/constitution`

[4] The Debian Social Contract. http://www.debian.org/social_contract

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

[6] The Debian FAQ. http://www.debian.org/doc/FAQ/

[7] The Free Software Definition. http://www.gnu.org/philosophy/free-sw.html

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

[9] The Linux Standards Base. http://www.linuxbase.org/

[10] Debian Bug Tracking System. http://bugs.debian.org/

[11] List of Unofficial Repositories. http://apt-get.org/

[12] Debian Planet. http://www.debianplanet.org/

[13] Debian Help. http://www.debianhelp.org/

[14] Debconf. http://kitenet.net/programs/debconf/