# Debian Policy Manual

Ian Jackson <ijackson@gnu.ai.mit.edu>

Christian Schwarz <schwarz@debian.org>

revised: David A. Morris <bweaver@debian.org>

The Debian Policy mailing List<debian-policy@lists.debian.org>

version 2.5.0.0, 1998-10-30

---

## 0.1. Abstract

This manual describes the policy requirements for the Debian GNU/Linux distribution. This includes the structure and contents of the Debian archive, several design issues of the operating system, as well as technical requirements that each package must satisfy to be included in the distribution.

---

## 0.2. Contents

**1. About this manual** **3**
1.1. Scope 3
1.2. New versions of this document 3
1.3. Feedback 3

**2. The Debian Archive** **4**
2.1. Package copyright and sections 4
2.2. Priorities 7
2.3. Binary packages 7
2.4. Source packages 10

**3. The Operating System** **12**
3.1. Filesystem hierarchy 12
3.2. Users and groups 12
3.3. System run levels 14
3.4. Cron jobs 17
3.5. Console messages 18
3.6. Menus 20
3.7. Keyboard configuration 20
3.8. Environment variables 21

**4. Files** **22**
4.1. Binaries 22
4.2. Libraries 22
4.3. Shared libraries 23
4.4. Scripts 23
4.5. Symbolic links 24
4.6. Device files 24

4.7. Configuration files 25
4.8. Log files 26
4.9. Permissions and owners 26

**5. Customized programs 28**
5.1. Architecture specification strings 28
5.2. Daemons 28
5.3. Editors and pagers 28
5.4. Web servers and applications 29
5.5. Mail transport agents 29
5.6. News system configuration 30
5.7. Programs for the X Windows system 30
5.8. Emacs lisp programs 31
5.9. Games 31

**6. Documentation 32**
6.1. Manual pages 32
6.2. Info documents 32
6.3. Additional documentation 33
6.4. Preferred documentation formats 33
6.5. Copyright information 33
6.6. Examples 34
6.7. Changelog files 34

---

## 0.3. Copyright Notice

Copyright

©1996,1997,1998 Ian Jackson and Christian Schwarz.

This manual is free software; you may redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

A copy of the GNU General Public License is available as `/usr/doc/copyright/GPL` in the Debian GNU/Linux distribution or on the World Wide Web at `http://www.gnu.org/copyleft/gpl.html`. You can also obtain it by writing to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

# 1. About this manual

## 1.1. Scope

This manual describes the policy requirements for the Debian GNU/Linux distribution. This includes the structure and contents of the Debian archive, several design issues of the operating system, as well as technical requirements that each package must satisfy to be included in the distribution.

This manual does *not* describe the technical mechanisms involved in package creation, installation, and removal. This information can be found in the *Debian Packaging Manual* and the *Debian System Administrators' Manual*.

This document assumes familiarity with these other two manuals. Unfortunately, the *System Administrators' Manual* does not exist yet.

Much of the information presented in this manual will be useful even when building a package which is to be distributed in some other way or is for local use.

## 1.2. New versions of this document

The current version of this document is always accessible from the Debian FTP server at **ftp://ftp.debian.org/debian/doc/manuals/debian-policy.html.tar.gz** or from the Debian WWW server at **http://www.debian.org/doc/manuals/debian-policy/**

In addition, this manual is distributed via the Debian package **debian-policy**

## 1.3. Feedback

As the Debian GNU/Linux system is continuously evolving this manual is changed from time to time.

While the authors of this document tried hard not to include any typos or other errors these still occur. If you discover an error in this manual or if you want to tell us any comments, suggestions, or critics please send an email to the Debian Policy List, **<debian-policy@lists.debian.org>**, or submit a bug report against the **debian-policy** package.

# 2. The Debian Archive

The Debian GNU/Linux system is maintained and distributed as a collection of *packages*. Since there are so many of them (over 2600) they are split into *sections* and *priorities* to simplify handling of them.

The effort of the Debian project is to build a free operating system, but not every package we want to make accessible is *free* in our sense (see Debian Free Software Guidelines, below), or may be imported/exported without restrictions. Thus, the archive is split into the sections *main*, *non-us*, *non-free*, and *contrib.*

The *main* section forms the *Debian GNU/Linux distribution.*

Packages in the other sections are not considered as part of the Debian distribution, though we support their use, and we provide infrastructure for them (such as our bug-tracking system and mailing lists). This Debian Policy Manual applies to these packages as well.

## 2.1. Package copyright and sections

The aims of this policy are:

- We want to make as much software available as we can.
- We want to encourage everyone to write free software.
- We want to make it easy for people to produce CD-ROMs of our system without violating any licenses, import/export restrictions, or any other laws.

### 2.1.1. The Debian Free Software Guidelines

The Debian Free Software Guidelines (DFSG) as is our definition of 'free' software.

>Free Redistribution
: The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

Source Code
: The program must include source code, and must allow distribution in source code as well as compiled form.

Derived Works
: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Integrity of The Author's Source Code
: The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (This is a compromise. The Debian group encourages all authors to not restrict any files, source or binary, from

being modified.)

No Discrimination Against Persons or Groups
: The license must not discriminate against any person or group of persons.

No Discrimination Against Fields of Endeavor
: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Distribution of License
: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

License Must Not Be Specific to Debian
: The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

License Must Not Contaminate Other Software
: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

Example Licenses
: The "GPL," "BSD," and "Artistic" licenses are examples of licenses that we consider *free*.

### 2.1.2. The main section

Every package in "main" must comply with the DFSG (Debian Free Software Guidelines).

In addition, the packages in "main"

- must not require a package outside of "main" for compilation or execution (thus, the package may not declare a "Depends" or "Recommends" relationship on a non-main package),
- must not be so buggy that we refuse to support them,
- must meet all policy requirements presented in this manual.

### 2.1.3. The contrib section

Every package in "contrib" must comply with the DFSG.

Examples of packages which would be included in "contrib" are

- free packages which require "contrib", "non-free", or "non-US" packages or packages which are not in our archive at all for compilation or execution,
- wrapper packages or other sorts of free accessories for non-free programs,
- packages which we don't want to support because they are too buggy, and
- packages which fail to meet some other policy requirements in a serious way.

### 2.1.4. The non-free section

'Non-free' contains packages which are not compliant with the DFSG or which are encumbered by patents or other legal issues that make their distribution problematic.

All packages in 'non-free' must be electronically distributable across international borders.

### 2.1.5. The non-us server

Some programs with cryptographic program code must be stored on the "non-us" server because of export restrictions of the U.S.

This applies only to packages which contain cryptographic code. A package containing a program with an interface to a cryptographic program or a program that's dynamically linked against a cryptographic library can be distributed if it is capable of running without the cryptography library or program.

### 2.1.6. Further copyright considerations

Every package must be accompanied by a verbatim copy of its copyright and distribution license in the file /usr/doc/<package-name>/copyright (see 'Copyright information', page 33 for details).

We reserve the right to restrict files from being included anywhere in our archives if
- their use or distribution would break a law,
- there is an ethical conflict in their distribution or use,
- we would have to sign a license for them, or
- their distribution would conflict with other project policies.

Programs whose authors encourage the user to make donations are fine for the main distribution, provided that the authors do not claim that not donating is immoral, unethical, illegal or something similar; otherwise they must go in contrib (or non-free, if even distribution is restricted by such statements).

Packages whose copyright permission notices (or patent problems) do not allow redistribution even of only binaries, and where no special permission has been obtained, cannot be placed on the Debian FTP site and its mirrors at all.

Note, that under international copyright law (this applies in the United States, too) *no* distribution or modification of a work is allowed without an explicit notice saying so. Therefore a program without a copyright notice *is* copyrighted and you may not do anything to it without risking being sued! Likewise if a program has a copyright notice but no statement saying what is permitted then nothing is permitted.

Many authors are unaware of the problems that restrictive copyrights (or lack of copyright notices) can cause for the users of their supposedly-free software. It is often worthwhile contacting such authors diplomatically to ask them to modify their license terms. However, this is a politically difficult thing to do and you should ask for advice on **debian-devel** first.

When in doubt, send mail to **<debian-devel@lists.debian.org>**. Be prepared to provide us with the copyright statement. Software covered by the GPL, public domain software and BSD-like copyrights are safe; be wary of the phrases 'commercial use prohibited' and 'distribution restricted'.

### 2.1.7. Subsections

The packages in the *main*, *contrib*, and *non-free* sections are grouped further into *subsections* to simplify handling of them.

The section for each package is specified in the package's *control record*. However, the maintainer of the Debian archive may override this selection to assure the consistency of the Debian distribution.

Please check the current Debian distribution to see which sections are available.

---

## 2.2. Priorities

Each package is given a certain *priority* value, which is included in the package's *control record*. This information is used in the Debian package management tool to separate high-priority packages from less-important packages.

The following *priority levels* are supported by the Debian package management system, **dpkg**.

**required**
: **required** packages are necessary for the proper functioning of the system. You must not remove these packages or your system may become totally broken and you may not even be able to use **dpkg** to put things back. Systems with only the **required** packages are probably unusable, but they do have enough functionality to allow the sysadmin to boot and install more software.

**important**
: Important programs, including those which one would expect to find on any Unix-like system. If the expectation is that an experienced Unix person who found it missing would say 'What the F*!@<+ is going on, where is **foo**', it should be in **important**. This is an important criterion because we are trying to produce, amongst other things, a free Unix. Other packages without which the system will not run well or be usable should also be here. This does *not* include Emacs or X11 or TeX or any other large applications. The **important** packages are just a bare minimum of commonly-expected and necessary tools.

**standard**
: These packages provide a reasonably small but not too limited character-mode system. This is what will install by default if the user doesn't select anything else. It doesn't include many large applications, but it does include Emacs (this is more of a piece of infrastructure than an application) and a reasonable subset of TeX and LaTeX (if this is possible without X).

**optional**
: (In a sense everything is optional that isn't required, but that's not what is meant here.) This is all the software that you might reasonably want to install if you didn't know what it was or don't have specialised requirements. This is a much larger system and includes X11, a full TeX distribution, and lots of applications.

**extra**
: This contains packages that conflict with others with higher priorities, or are only likely to be useful if you already know what they are or have specialised requirements.

Packages may not depend on packages with lower priority values. If this should happen, one of the priority values will have to be adapted.

## 2.3. Binary packages

The Debian GNU/Linux distribution is based on the Debian package management system, called **dpkg**. Thus, all packages in the Debian distribution have to be provided in the **.deb** file format.

### 2.3.1. The package name

Every package must have a name that's unique within the Debian archive.

Package names may only consist of lower case letters, digits (0-9), plus (+) or minus (-) signs, and periods (.).

The package name is part of the file name of the **.deb** file and is included in the control field information.

### 2.3.2. The maintainer of a package

Every package must have exactly one maintainer at a time. This person is responsible that the license of the package's software complies with the policy of the distribution this package is included in.

The maintainer must be specified in the **Maintainer** control field with the correct name and a working email address for the Debian maintainer of the package. If one person maintains several packages he/she should try to avoid having different forms of their name and email address in different **Maintainer** fields.

If the maintainer of a package quits from the Debian project the Debian QA Group takes over the maintainership of the package until someone else volunteers for that task. These packages are called *orphaned packages.*

### 2.3.3. The description of a package

Every Debian package should have an extended description stored in the appropriate field of the control record.

The description should be written so that it tells the user what they need to know to decide whether to install the package. This description should not just be copied from the blurb for the program. Instructions for configuring or using the package should not be included--that is what installation scripts, manual pages, Info files, etc. are for. Copyright statements and other administrivia should not be included--that is what the copyright file is for.

### 2.3.4. Dependencies

Every package has to specify the dependency information about other packages, that are required for the first to work correctly.

For example, for any shared libraries required by dynamically-linked executable binary in a package a dependency entry has to be provided.

It is not necessary for other packages to declare any dependencies they have on other packages

which are marked **Essential** (see below).

Sometimes, a package requires another package to be installed *and* configured before it can be installed. In this case, you'll have to specify a **Pre-Depends** entry for the package.

You must not specify a **Pre-Depends** entry for a package before this has been discussed on the **debian-devel** mailing list and a consensus about doing that has been reached.

### 2.3.5. Virtual packages

Sometimes, there are several packages doing more-or-less the same job. In this case, it's useful to define a *virtual package* who's name describes the function the packages have. (The virtual packages just exist logically, not physically--that's why they are called *virtual*.) The packages with this particular function will then *provide* the virtual package. Thus, any other package requiring that function can simply depend on the virtual package without having to specify all possible packages individually.

All packages must use virtual package names where appropriate, and arrange to create new ones if necessary. They must not use virtual package names (except privately, amongst a cooperating group of packages) unless they have been agreed upon and appear in the list of virtual package names.

The latest version of the authoritative list of virtual package names can be found on **ftp.debian.org** in **/debian/doc/package-developer/virtual-package-names-list.text** or your local mirror. In addition, it is included in the **debian-policy** package. The procedure for updating the list is described at the top of the file.

### 2.3.6. Base packages

The packages included in the **base** section have a special function. They form a minimum subset of the Debian GNU/Linux system that is installed before everything else on a new system. Thus, only very few packages are allowed to go into the **base** section to keep the required disk usage very small.

Most of these packages should have the priority value **required** or at least **important**, and many of them will be tagged **essential** (see below).

You must not place any packages into the **base** section before this has been discussed on the **debian-devel** mailing list and a consensus about doing that has been reached.

### 2.3.7. Essential packages

Some packages are tagged **essential**. (They have **Essential: yes** in their package control record.) This flag is used for packages that are *essential* for a system.

Since these packages can not easily be removed (you'll have to specify an extra *force option* to **dpkg**) this flag should only be used where absolutely necessary. A shared library package should not be tagged *essential*--the dependencies will prevent its premature removal, and we need to be able to remove it when it has been superseded.

You must not tag any packages **essential** before this has been discussed on the **debian-devel** mailing and a consensus about doing that has been reached.

### 2.3.8. Maintainer scripts

The package installation scripts should avoid producing output which it is unnecessary for the user to see and should rely on **dpkg** to stave off boredom on the part of a user installing many packages. This means, amongst other things, using the **--quiet** option on **install-info**.

Packages should try to minimise the amount of prompting they need to do, and they should ensure that the user will only ever be asked each question once. This means that packages should try to use appropriate shared configuration files (such as **/etc/papersize** and **/etc/nntpserver**), rather than each prompting for their own list of required pieces of information.

It also means that an upgrade should not ask the same questions again, unless the user has used **dpkg --purge** to remove the package's configuration. The answers to configuration questions should be stored in an appropriate place in **/etc** so that the user can modify them, and how this has been done should be documented.

If a package has a vitally important piece of information to pass to the user (such as "don't run me as I am, you must edit the following configuration files first or you risk your system emitting badly-formatted messages"), it should display this in the **postinst** script and prompt the user to hit return to acknowledge the message. Copyright messages do not count as vitally important (they belong in **/usr/doc/***package***/copyright**); neither do instructions on how to use a program (these should be in on line documentation, where all the users can see them).

Any necessary prompting should almost always be confined to the post-installation script, and should be protected with a conditional so that unnecessary prompting doesn't happen if a package's installation fails and the **postinst** is called with **abort-upgrade**, **abort-remove** or **abort-deconfigure**.

Errors which occur during the execution of an installation script *must* be checked and the installation *must not* continue after an error.

Note, that 'Scripts', page 23, in general applies to package maintainer scripts, too.

Do not use **dpkg-divert** on a file belonging to another package without consulting the maintainer of that package first.

In order for **update-alternatives** to work correctly all the packages which supply an instance of the 'shared' command name (or, in general, filename) must use it. You can use **Conflicts** to force the deinstallation of other packages supplying it which do not (yet) use **update-alternatives**. It may in this case be appropriate to specify a conflict on earlier versions on something--this is an exception to the usual rule that this is not allowed.

---

## 2.4. Source packages

### 2.4.1. Standards conformance

You should specify the most recent version of the packaging standards with which your package complies in the source package's **Standards-Version** field.

This value will be used to file bug reports automatically if your package becomes too much out of date.

The value corresponds to a version of the Debian manuals, as can be found on the title page or page headers and footers (depending on the format).

The version number has four components--major and minor number and major and minor patch level. When the standards change in a way that requires every package to change the major number will be changed. Significant changes that will require work in many packages will be signaled by a change to the minor number. The major patch level will be changed for any change to the meaning of the standards, however small; the minor patch level will be changed when only cosmetic, typographical or other edits which do not change the meaning are made, or changes which do not affect the contents of packages.

You should regularly, and especially if your package has become out of date, check for the newest Policy Manual available and update your package, if necessary. When your package complies with the new standards you may update the **Standards-Version** source package field and release it.

### 2.4.2. Changes to the upstream sources

If changes to the source code are made that are generally applicable please try to get them included in the upstream version of the package by supplying the upstream authors with the changes in whatever form they prefer.

If you need to configure the package differently for Debian or for Linux, and the upstream source doesn't provide a way to configure it the way you need to, please add such configuration facilities (for example, a new **autoconf** test or **#define**) and send the patch to the upstream authors, with the default set to the way they originally had it. You can then easily override the default in your **debian/rules** or wherever is appropriate.

Please make sure that the **configure** utility detects the correct architecture specification string (refer to 'Architecture specification strings', page 28 for details).

If you need to edit a **Makefile** where GNU-style **configure** scripts are used, you should edit the **.in** files rather than editing the **Makefile** directly. This allows the user to reconfigure the package if necessary. You should *not* configure the package and edit the generated **Makefile**! This makes it impossible for someone else to later reconfigure the package.

### 2.4.3. Documenting your changes

Document your changes and updates to the source package properly in the **debian/changelog** file.

A copy of the file which will be installed in **/usr/doc/***package***/copyright** should be in **debian/copyright**.

In non-experimental packages you may only use a format for **debian/changelog** which is supported by the most recent released version of **dpkg**. If your format is not supported and there is general support for it you should contact the **dpkg** maintainer to have the parser script for your format included in the **dpkg** package. (You will need to agree that the parser and its manpage may be distributed under the GNU GPL, just as the rest of **dpkg** is.)

### 2.4.4. Error trapping in makefiles

When **make** invokes a command in a makefile (including your package's upstream makefiles and

the **debian/rules**) it does so using **sh**. This means that **sh**'s usual bad error handling properties apply: if you include a miniature script as one of the commands in your makefile you'll find that if you don't do anything about it then errors are not detected and **make** will blithely continue after problems.

Every time you put more than one shell command (this includes using a loop) in a makefile command you *must* make sure that errors are trapped. For simple compound commands, such as changing directory and then running a program, using **&&** rather than semicolon as a command separator is sufficient. For more complex commands including most loops and conditionals you must include a separate **set -e** command at the start of every makefile command that's actually one of these miniature shell scripts.

### 2.4.5. Obsolete constructs and libraries

The include file **<varargs.h>** is provided to support end-users compiling very old software; the library **libtermcap** is provided to support the execution of software which has been linked against it (either old programs or those such as Netscape which are only available in binary form).

Debian packages should be ported to include **<stdarg.h>** and **ncurses** when they are built.

# 3. The Operating System

## 3.1. Filesystem hierarchy

### 3.1.1. Linux Filesystem Structure

The location of all installed files and directories must comply fully with the Linux Filesystem Structure (FSSTND). The latest version of this document can be found alongside this manual or on **tsx-11.mit.edu** in **/pub/linux/docs/linux-standards/fsstnd/**. Specific questions about following the standard may be asked on **debian-devel**, or referred to Daniel Quinlan, the FSSTND coordinator, at **<quinlan@pathname.com>**.

### 3.1.2. Site-specific programs

As mandated by the FSSTND no package should place any files in **/usr/local**, either by putting them in the filesystem archive to be unpacked by **dpkg** or by manipulating them in their maintainer scripts.

However, the package should create empty directories below **/usr/local** so that the system administrator knows where to place site-specific files. These directories should be removed on package removal if they are empty.

Note, that this applies only to directories *below* **/usr/local**, not *in* **/usr/local**. The directory **/usr/local** itself may only contain the sub-directories listed in FSSTND, section 4.8. However, you may create directories below them as you wish. You may not remove any of the directories listed in 4.8, even if you created them.

Since **/usr/local** may be mounted read-only from a remote server, these directories have to be created and removed by the **postinst** and **prerm** maintainer scripts. These scripts must not fail if either of these operations fail. (In the future, it will be possible to tell **dpkg** not to unpack files matching certain patterns, so that the directories can be included in the **.deb** packages and system administrators who do not wish these directories in /usr/local do not need to have them.)

For example, the **emacs** package will contain

```
mkdir -p /usr/local/lib/emacs/site-lisp || true
```

in the **postinst** script, and

```
rmdir /usr/local/lib/emacs/site-lisp && \
              rmdir /usr/local/lib/emacs || \
              true
```

in the **prerm** script.

If you do create a directory in **/usr/local** for local additions to a package, you must ensure that settings in **/usr/local** take precedence over the equivalents in **/usr**.

The **/usr/local** directory itself and all the subdirectories created by the package should have permissions 2775 (group-writable and set-group-id) and be owned by **root.staff**.

## 3.2. Users and groups

The Debian system can be configured to use either plain or shadow passwords.

Some user ids (UIDs) and group ids (GIDs) are reserved globally for use by certain packages. Because some packages need to include files which are owned by these users or groups, or need the ids compiled into binaries, these ids must be used on any Debian system only for the purpose for which they are allocated. This is a serious restriction, and we should avoid getting in the way of local administration policies. In particular, many sites allocate users and/or local system groups starting at 100.

Apart from this we should have dynamically allocated ids, which should by default be arranged in some sensible order--but the behaviour should be configurable.

No package except **base-passwd** may modify **/etc/passwd**, **/etc/shadow**, or **/etc/group**.

The UID and GID ranges are as follows:

0-99:
: Globally allocated by the Debian project, must be the same on every Debian system. These ids will appear in the **passwd** and **group** files of all Debian systems, new ids in this range being added automatically as the **base-passwd** package is updated.

  Packages which need a single statically allocated uid or gid should use one of these; their maintainers should ask the **base-passwd** maintainer for ids.

100-999:
: Dynamically allocated system users and groups. Packages which need a user or group, but can have this user or group allocated dynamically and differently on each system, should use '**adduser --system**' to create the group and/or user. **adduser** will check for the existence of the user or group, and if necessary choose an unused id based on the ranged specified in **adduser.conf**.

1000-29999:
: Dynamically allocated user accounts. By default **adduser** will choose UIDs and GIDs for user accounts in this range, though **adduser.conf** may be used to modify this behaviour.

30000-59999:
: Reserved.

60000-64999:
: Globally allocated by the Debian project, but only created on demand. The ids are allocated centrally and statically, but the actual accounts are only created on users' systems on demand.

  These ids are for packages which are obscure or which require many statically-allocated ids. These packages should check for and create the accounts in **/etc/passwd** or **/etc/group** (using **adduser** if it has this facility) if necessary. Packages which are likely to require further allocations should have a 'hole' left after them in the allocation, to give them room to grow.

65000-65533:
: Reserved.

65534:
: User '**nobody**.'

65535:
**(uid_t)(-1) == (gid_t)(-1)**. NOT TO BE USED, because it is the error return sentinel value.

---

## 3.3. System run levels

### 3.3.1. Introduction

The **/etc/init.d** directory contains the scripts executed by **init** at boot time and when init state (or 'runlevel') is changed (see **init(8)**).

These scripts are being referenced by symbolic links in the **/etc/rc***n***.d** directories. When changing runlevels, **init** looks in the directory **/etc/rc***n***.d** for the scripts it should execute, where *n* is the runlevel that is being changed to, or 'S' for the boot-up scripts.

The names of the links all have the form **S***mm/script* or **K***mm/script* where *mm* is a two-digit number and *script* is the name of the script (this should be the same as the name of the actual script in **/etc/init.d**. When **init** changes runlevel first the targets of the links whose names starting with a **K** are executed, each with the single argument **stop**, followed by the scripts prefixed with an **S**, each with the single argument **start**. The **K** links are responsible for killing services and the **S** link for starting services upon entering the runlevel.

For example, if we are changing from runlevel 2 to runlevel 3, init will first execute all of the **K** prefixed scripts it finds in **/etc/rc3.d**, and then all of the **S** prefixed scripts. The links starting with **K** will cause the referred-to file to be executed with an argument of **stop**, and the **S** links with an argument of **start**.

The two-digit number *mm* is used to decide which order to start and stop things in--low-numbered links have their scripts run first. For example, the **K20** scripts will be executed before the **K30** scripts. This is used when a certain service must be started before another. For example, the name server **bind** might need to be started before the news server **inn** so that **inn** can set up its access lists. In this case, the script that starts **bind** should have a lower number than the script that starts **inn** so that it runs first:

```
/etc/rc2.d/S17bind
/etc/rc2.d/S70inn
```

### 3.3.2. Writing the scripts

Packages can and should place scripts in **/etc/init.d** to start or stop services at boot time or during a change of runlevel. These scripts should be named **/etc/init.d/***package*, and they should accept one argument, saying what to do:

**start**
start the service,

**stop**
stop the service,

**restart**

stop and restart the service,

**reload**
cause the configuration of the service to be reloaded without actually stopping and restarting the service,

**force-reload**
cause the configuration to be reloaded if the service supports this, otherwise restart the service.

The **start**, **stop**, **restart**, and **force-reload** options must be supported by all scripts in **/etc/init.d**, the **reload** option is optional.

The **init.d** scripts should ensure that they will behave sensibly if invoked with **start** when the service is already running, or with **stop** when it isn't, and that they don't kill unfortunately-named user processes. The best way to achieve this is usually to use **start-stop-daemon**.

If a service reloads its configuration automatically (as in the case of **cron**, for example), the **reload** option of the **init.d** script should behave as if the configuration has been reloaded successfully.

These scripts should not fail obscurely when the configuration files remain but the package has been removed, as the default in **dpkg** is to leave configuration files on the system after the package has been removed. Only when it is executed with the **--purge** option will dpkg remove configuration files. Therefore, you should include a **test** statement at the top of the script, like this:

```
test -f program–executed–later–in–script || exit 0
```

### 3.3.3. Managing the links

A program is provided, **update-rc.d**, to make it easier for package maintainers to arrange for the proper creation and removal of **/etc/rc*n*.d** symbolic links from their **postinst** and **postrm** scripts.

You should use this script to make changes to **/etc/rc*n*.d** and *never* include any **/etc/rc*n*.d** symbolic links in the actual archive.

By default **update-rc.d** will start services in each of the multi-user state runlevels (2, 3, 4, and 5) and stop them in the halt runlevel (0), the single-user runlevel (1) and the reboot runlevel (6). The system administrator will have the opportunity to customize runlevels by simply adding, moving, or removing the symbolic links in **/etc/rc*n*.d**.

To get the default behaviour for your package, put in your **postinst** script

```
update-rc.d package default >/dev/null
```

and in your **postrm**

```
if [ purge = "$1" ]; then
            update-rc.d package remove >/dev/null
            fi
```

This will use a default sequence number of 20. If it does not matter when or in which order the script is run, use this default. If it does, then you should talk to the maintainer of the **sysvinit** package or post to **debian-devel**, and they will help you choose a number.

For more information about using **update-rc.d**, please consult its manpage **update-rc.d(8)**.

### 3.3.4. Boot-time initialisation

There is another directory, **/etc/rc.boot**, which contains scripts which are run once per machine boot. This facility is provided for initialisation of hardware devices, cleaning up of leftover files, and so forth.

For example, the **kbd** package provides a script here for initialising the keyboard layout and console font and mode.

The files in **/etc/rc.boot** should *not* be links into **/etc/init.d**--they should be the scripts themselves.

**rc.boot** should *not* be used for starting general-purpose daemons and similar activities. This should be done using the **rc***n***.d** scheme, above, so that the services can be started and stopped cleanly when the runlevel changes or the machine is to be shut down or rebooted.

### 3.3.5. Notes

*Do not* include the **/etc/rc***n***.d/*** symbolic links in the **.deb** filesystem archive! *This will cause problems!* You should create them with **update-rc.d**, as above.

*Do not* include the **/etc/rc***n***.d/*** symbolic links in **dpkg**'s conffiles list! *This will cause problems! Do*, however, include the **/etc/init.d** scripts in conffiles. (This is important since we want to give the local system administrator the chance to adapt the scripts to the local system--e.g., to disable a service without deinstalling the package, or to specify some special command line options when starting a service--while making sure her changes aren't lost during the next package upgrade.)

### 3.3.6. Example

The **bind** DNS (nameserver) package wants to make sure that the nameserver is running in multiuser runlevels, and is properly shut down with the system. It puts a script in **/etc/init.d**, naming the script appropriately **bind**. As you can see, the script interprets the argument **reload** to send the nameserver a **HUP** signal (causing it to reload its configuration); this way the user can say **/etc/init.d/bind reload** to reload the name server.

```
#!/bin/sh
                #
                # Original version by Robert Leslie
                # <rob@mars.org>, edited by iwj and cs

                test -x /usr/sbin/named || exit 0

                case "$1" in
                start)
                echo -n "Starting domain name service: named"
                start-stop-daemon --start --quiet --exec /usr/sbin/named
                echo "."
                ;;
                stop)
                echo -n "Stopping domain name service: named"
                start-stop-daemon --stop --quiet  \
                --pidfile /var/run/named.pid --exec /usr/sbin/named
                echo "."
                ;;
                restart)
                echo -n "Restarting domain name service: named"
                start-stop-daemon --stop --quiet  \
                --pidfile /var/run/named.pid --exec /usr/sbin/named
                start-stop-daemon --start --verbose --exec /usr/sbin/named
                echo "."
                ;;
                force-reload|reload)
                echo -n "Reloading configuration of domain name service: named"
                start-stop-daemon --stop --signal 1 --quiet  \
                --pidfile /var/run/named.pid --exec /usr/sbin/named
                echo "."
                ;;
                *)
                echo "Usage: /etc/init.d/bind {start|stop|restart|reload|force-reload}"
>&2
                exit 1
                ;;
                esac

                exit 0
```

Another example on which to base your **/etc/init.d** scripts is in **/etc/init.d/skeleton**.

If this package is happy with the default setup from **update-rc.d**, namely an ordering number of 20 and having named running in all runlevels, it can say in its **postinst**:

```
update-rc.d bind default >/dev/null
```

And in its **postrm**, to remove the links when the package is purged:

```
if [ purge = "$1" ]; then
                update-rc.d acct remove >/dev/null
                fi
```

## 3.4. Cron jobs

Packages may not touch the configuration file **/etc/crontab**, nor may they modify the files in **/var/spool/cron/crontabs**.

If a package wants to install a job that has to be executed via cron, it should place a file with the name if the package in one of the following directories:

**/etc/cron.daily**
**/etc/cron.weekly**
**/etc/cron.monthly**

As these directory names say, the files within them are executed on a daily, weekly, or monthly basis, respectively.

If a certain job has to be executed more frequently than 'daily,' the package should install a file **/etc/cron.d/<package-name>** tagged as configuration file. This file uses the same syntax as **/etc/crontab** and is processed by **cron** automatically. (Note, that scripts in the **/etc/cron.d** directory are not handled by **anacron**. Thus, you should only use this directory for jobs which may be skipped if the system is not running.)

All files installed in any of these directories have to be scripts (shell scripts, Perl scripts, etc.) so that they can easily be modified by the local system administrator. In addition, they have to be registered as configuration file.

The scripts in these directories have to check, if all necessary programs are installed before they try to execute them. Otherwise, problems will arise when a package was removed (but not purged), since the configuration files are kept on the system in this situation.

## 3.5. Console messages

This section describes different formats for messages written to standard output by the **/etc/init.d** scripts. The intent is to improve the consistency of Debian's startup and shutdown look and feel.

Please look very careful at the details. We want to get the messages to look exactly the same way concerning spaces, punctuation, and case of letters.

Here is a list of overall rules that you should use when you create output messages. They can be useful if you have a non-standard message that isn't covered in the sections below.

- Every message should cover one line, start with a capital letter and end with a period '.'.
- If you want to express that the computer is working on something (performing a specific task, not starting or stopping a program), we use an "ellipsis", namely three dots '…'. Note that we don't insert spaces in front of or behind the dots. If the task has been completed we write 'done.' and a line feed.
- Design your messages as if the computer is telling you what he is doing (let him be polite :-) but don't mention "him" directly. For example, if you think of saying

  **I'm starting network daemons: nfsd mountd.**

just say

```
Starting network daemons: nfsd mountd.
```

The following formats must be used

- when daemons get started. Use this format if your script starts one or more daemons. The output should look like this (a single line, no leading spaces):

  ```
  Starting <description>: <daemon-1> <daemon-2> <…> <daemon-n>.
  ```

  The <description> should describe the subsystem the daemon or set of daemons are part of, while <daemon-1> up to <daemon-n> denote each daemon's name (typically the file name of the program).

  For example, the output of /etc/init.d/lpd would look like:

  ```
  Starting printer spooler: lpd.
  ```

  This can be achieved by saying

  ```
  echo -n "Starting printer spooler: lpd"
                   start-stop-daemon --start --quiet lpd
                   echo "."
  ```

  in the script. If you have more than one daemon to start, you should do the following:

  ```
  echo -n "Starting remote filesystem services:"
                   echo -n " nfsd"; start-stop-daemon --start --quiet nfsd
                   echo -n " mountd"; start-stop-daemon --start --quiet mountd
                   echo -n " ugidd"; start-stop-daemon --start --quiet ugidd
                   echo "."
  ```

  This makes it possible for the user to see what takes so long and when the final daemon has been started. Please be careful where to put spaces: In the example above the system administrator can easily comment out a line if he don't wants to start a specific daemon, while the displayed message still looks good.

- when something needs to be configured.

  If you have to set up different parameters of the system upon boot up, you can use this format:

  ```
  Setting <parameter> to `<value>'.
  ```

  You can use the following echo statement to get the quotes right:

  ```
  echo "Setting DNS domainname to \`"value"'."
  ```

  Note that the left quotation mark (`) is different from the right (').

- when a daemon is stopped.

  When you stop a daemon you should issue a message similar to the startup message, except that 'Starting' is replaced with 'Stopping'.

  So stopping the printer daemon will like like this:

  ```
  Stopping printer spooler: lpd.
  ```

- when something is executed. There a several examples where you have to run a program at system startup or shutdown to perform a specific task. For example, setting the system's clock via 'netdate' or killing all processes when the system comes down. Your message should like this:

  ```
  Doing something very useful…done.
  ```

You should print the 'done.' right after the job has been completed, so that the user gets informed why he has to wait. You can get this behaviour by saying

```
echo -n "Doing something very useful..."
                    do_something
                    echo "done."
```

in your script.

- when the configuration is reloaded.

  When a daemon is forced to reload its configuration files you should use the following format:

  ```
  Reloading <daemon's-name> configuration…done.
  ```

- when none of the above rules apply. If you have to print a message that doesn't fit into the styles described above, you can use something appropriate, but please have a look at the overall rules listed above.

---

## 3.6. Menus

The Debian **menu** packages provides a unique interface between packages providing applications and documents, and *menu programs* (either X window managers or text-based menu programs as **pdmenu**).

All packages that provide applications that need not be passed any special command line arguments for normal operation should register a menu entry for those applications, so that users of the **menu** package will automatically get menu entries in their window managers, as well in shells like **pdmenu**.

Please refer to the *Debian Menu System* document that comes with the **menu** package for information about how to register your applications and web documents.

---

## 3.7. Keyboard configuration

To achieve a consistent keyboard configuration (i.e., all applications interpret a keyboard event the same way) all programs in the Debian distribution have to be configured to comply with the following guidelines.

Here is a list that contains certain keys and their interpretation:

**<--**
: delete the character to the left of the cursor

**Delete**
: delete the character to the right of the cursor

**Control+H**
: emacs: the help prefix

The interpretation of any keyboard events should be independent of the terminal that's used (either the console, X windows, rlogin/telnet session, etc.).

The following list explains how the different programs should be set up to achieve this:

- '**<--**' generates KB_Backspace in X.
- '**Delete**' generates KB_Delete in X.
- X translations are set up to make KB_Backspace generate ASCII DEL, and to make KB_Delete generate **ESC [ 3 ~** (this is the vt220 escape code for the 'delete character' key). This must be done by loading the resources using xrdb on all local X displays, not using the application defaults, so that the translation resources used correspond to the xmodmap settings.
- The Linux console is configured to make '**<--**' generate DEL, and 'Delete' generate **ESC [ 3 ~** (this is the case at the moment).
- X applications are configured so that Backspace deletes left, and Delete deletes right. Motif applications already work like this.
- stty erase **^?** .
- The 'xterm' terminfo entry should have **ESC [ 3 ~** for kdch1, just like TERM=linux and TERM=vt220.
- Emacs is programmed to map KB_Backspace or the 'stty erase' character to delete-backward-char, and KB_Delete or kdch1 to delete-forward-char, and **^H** to help as always.
- Other applications use the 'stty erase' character and kdch1 for the two delete keys, with ASCII DEL being 'delete previous character' and kdch1 being 'delete character under cursor'.

This will solve the problem except for:

- Some terminals have a **<--** key that cannot be made to produce anything except **^H**. On these terminals Emacs help will be unavailable on **^H** (assuming that the 'stty erase' character takes precedence in Emacs, and has been set correctly). M-x help or F1 (if available) can be used instead.
- Some operating systems use **^H** for stty erase. However, modern telnet versions and all rlogin versions propagate stty settings, and almost all UNIX versions honour stty erase. Where the stty settings are not propagated correctly things can be made to work by using stty manually.
- Some systems (including previous Debian versions) use xmodmap to arrange for both **<--** and Delete to generate KB_Delete). We can change the behaviour of their X clients via the same X resources that we use to do it for our own, or have our clients be configured via their resources when things are the other way around. On displays configured like this Delete will not work, but **<--** will.
- Some operating systems have different kdch1 settings in their terminfo for xterm and others. On these systems the Delete key will not work correctly when you log in from a system conforming to our policy, but **<--** will.

---

## 3.8. Environment variables

No program may depend on environment variables to get reasonable defaults. (That's because these environment variables would have to be set in a system-wide configuration file like /etc/profile, which is not supported by all shells.)

If a program should depend on environment variables for its configuration, the program has to

be changed to fall back to a reasonable default configuration if these environment variables are not present. If this cannot be done easily (e.g., if the source code of a non-free program is not available), the program should be replaced by a small 'wrapper' shell script which sets the environment variables and calls the original program.

Here is an example of a wrapper script for this purpose:

```
#!/bin/sh
BAR=/var/lib/fubar
export BAR
exec /usr/lib/foo/foo "$@"
```

Furthermore, as **/etc/profile** is a configuration file of the **bash** package, no other package may put any environment variables or other commands into that file.

# 4. Files

## 4.1. Binaries

It is not allowed that two packages install programs with different functionality but with the same filenames. (The case of two programs having the same functionality but different implementations is handled via 'alternatives.') If this case happens, one of the programs has to be renamed. The maintainers should report this to the developers' mailing and try to find a consensus about which package will have to be renamed. If a consensus can not be reached, *both* programs must be renamed.

Generally the following compilation parameters should be used:

```
CC = gcc
CFLAGS = -O2 -g -Wall # sane warning options vary between programs
LDFLAGS = # none
install -s # (or use strip on the files in debian/tmp)
```

Note that all installed binaries should be stripped, either by using the **-s** flag to **install**, or by calling **strip** on the binaries after they have been copied into **debian/tmp** but before the tree is made into a package.

The **-g** flag is useful on compilation so that you have available a full set of debugging symbols in your built source tree, in case anyone should file a bug report involving (for example) a core dump.

The **-N** flag should not be used. On a.out systems it may have been useful for some very small binaries, but for ELF it has no good effect.

It is up to the package maintainer to decide what compilation options are best for the package. Certain binaries (such as computationally-intensive programs) may function better with certain flags (**-O3**, for example); feel free to use them. Please use good judgment here. Don't use flags for the sake of it; only use them if there is good reason to do so. Feel free to override the upstream author's ideas about which compilation options are best--they are often inappropriate for our environment.

## 4.2. Libraries

All libraries must have a shared version in the lib package and a static version in the lib-dev package. The shared version must be compiled with **-fPIC**, and the static version must not be. In other words, each ***.c** file is compiled twice.

You have to specify the gcc option **-D_REENTRANT** when building a library (either static or shared) to make the library compatible with LinuxThreads.

Note that all installed shared libraries should be stripped with

```
strip --strip-unneeded <your-lib>
```

(The option '--strip-unneeded' makes **strip** remove only the symbols which aren't needed for relocation processing.) Shared libraries can function perfectly well when stripped, since the

symbols for dynamic linking are in a separate part of the ELF object file.

Note that under some circumstances it may be useful to install a shared library unstripped, for example when building a separate package to support debugging.

Please make sure that you use only released versions of shared libraries to build your packages; otherwise other users will not be able to run your binaries properly. Producing source packages that depend on unreleased compilers is also usually a bad idea.

## 4.3. Shared libraries

Packages involving shared libraries should be split up into several binary packages.

For a straightforward library which has a development environment and a runtime kit including just shared libraries you need to create two packages: *librarynamesoname* (*soname* is the shared object name of the shared library--it's the thing that has to match exactly between building an executable and running it for the dynamic linker to be able run the program; usually the *soname* is the major number of the library) and *librarynamesoname*`-dev`.

If you prefer only to support one development version at a time you may name the development package *libraryname*`-dev`; otherwise you may wish to use `dpkg`'s conflicts mechanism to ensure that the user only installs one development version at a time (after all, different development versions are likely to have the same header files in them, causing a filename clash if both are installed). Typically the development version will also need an exact version dependency on the runtime library, to make sure that compilation and linking happens correctly.

Packages which use the shared library should have a dependency on the name of the shared library package, *librarynamesoname*. When the *soname* changes you can have both versions of the library installed while moving from the old library to the new.

If your package has some run-time support programs which use the shared library you must *not* put them in the shared library package. If you do that then you won't be able to install several versions of the shared library without getting filename clashes. Instead, either create a third package for the runtime binaries (this package might typically be named *libraryname*`-runtime`--note the absence of the *soname* in the package name) or if the development package is small include them in there.

If you have several shared libraries built from the same source tree you can lump them all together into a single shared library package, provided that you change all their *soname*s at once (so that you don't get filename clashes if you try to install different versions of the combined shared libraries package).

Follow the directions in the *Debian Packaging Manual* for putting the shared library in its package, and make sure you include a `shlibs` control area file with details of the dependencies for packages which use the library.

Shared libraries should *not* be installed executable, since `ld.so` does not require this and trying to execute a shared library results in a core dump.

## 4.4. Scripts

All command scripts, including the package maintainer scripts inside the package and used by **dpkg**, should have a **#!** line naming the shell to be used to interpret them.

In the case of Perl scripts this should be **#!/usr/bin/perl**.

Shell scripts (**sh** and **bash**) should almost certainly start with **set -e** so that errors are detected. Every script *must* use **set -e** or check the exit status of *every* command.

The standard shell interpreter '**/bin/sh**' may be a symbolic link to any POSIX compatible shell. Thus, shell scripts specifying '**/bin/sh**' as interpreter may only use POSIX features. If a script requires non-POSIX features from the shell interpreter, the appropriate shell has to be specified in the first line of the script (e.g., '**#!/bin/bash**') and the package has to depend on the package providing the shell (unless the shell package is marked 'Essential', e.g., in the case of **bash**).

Restrict your script to POSIX features when possible so that it may use **/bin/sh** as its interpreter. If your script works with **ash**, it's probably POSIX compliant, but if you are in doubt, use **/bin/bash**.

Perl scripts should check for errors when making any system calls, including **open**, **print**, **close**, **rename** and **system**.

**csh** and **tcsh** should be avoided as scripting languages. See *Csh Programming Considered Harmful*, one of the **comp.unix.*** FAQs. It can be found on **rtfm.mit.edu** in **/pub/usenet-by-group/comp.unix.programmer/Csh_Programming_Considered_Harmful**. If an upstream package comes with **csh** scripts then you must make sure that they start with **#!/bin/csh** and make your package depend on the **c-shell** virtual package.

Any scripts which create files in world-writable directories (e.g., in **/tmp**) have to use a mechanism which will fail if a file with the same name already exists.

The Debian base distribution provides the **tempfile** and **mktemp** utilities for use by scripts for this purpose.

---

## 4.5. Symbolic links

In general, symbolic links within a toplevel directory should be relative, and symbolic links pointing from one toplevel directory into another should be absolute. (A toplevel directory is a sub-directory of the root directory '/'.)

In addition, symbolic links should be specified as short as possible, i.e., link targets like 'foo/../bar' are deprecated.

Note that when creating a relative link using **ln** it is not necessary for the target of the link to exist relative to the working directory you're running **ln** from; nor is it necessary to change directory to the directory where the link is to be made. Simply include the string that should appear as the target of the link (this will be a pathname relative to the directory in which the link resides) as the first argument to **ln**.

For example, in your **Makefile** or **debian/rules**, do things like:

```
ln -fs gcc $(prefix)/bin/cc
ln -fs gcc debian/tmp/usr/bin/cc
ln -fs ../sbin/sendmail $(prefix)/bin/runq
ln -fs ../sbin/sendmail debian/tmp/usr/bin/runq
```

A symbolic link pointing to a compressed file should always have the same file extension as the referenced file. (For example, if a file '**foo.gz**' is referenced by a symbolic link, the filename of the link has to end with '**.gz**' too, as in 'bar.gz.')

## 4.6. Device files

No package may include device files in the package file tree.

If a package needs any special device files that are not included in the base system, it has to call **makedev** in the **postinst** script, after asking the user for permission to do so.

No package should remove any device files in the **postrm** or any other script. This is left to the system administrator.

Debian uses the serial devices **/dev/tty***. Programs using the old **/dev/cu*** devices should be changed to use **/dev/tty***.

## 4.7. Configuration files

Any configuration files created or used by your package should reside in **/etc**. If there are several you should consider creating a subdirectory named after your package.

It is almost certain that any file in **/etc** that is in your package's filesystem archive should be listed in **dpkg**'s **conffiles** control area file. (See the *Debian Packaging Manual*).

Only packages that are tagged *conflicting* with each other may specify the same file as **conffile**. A package may not modify a configuration file of another package.

If two or more packages use the same configuration file, one of these packages has to be defined as *owner* of the configuration file, i.e., it has to list the file as **conffile** and has to provide a program that modifies the configuration file.

The other packages have to depend on the *owner* package and use that program to update the configuration file.

Sometimes it's appropriate to build a new package, which just provides the basic *infrastructure* for the other packages and which manages the shared configuration files. (Check out the **sgml-base** package as an example.)

Files in **/etc/skel** will automatically be copied into new user accounts by **adduser**. They should not be referenced there by any program.

Therefore, if a program needs a dotfile to exist in advance in **$HOME** to work sensibly that dotfile should be installed in **/etc/skel** (and listed in conffiles, if it is not generated and modified dynamically by the package's installation scripts).

However, programs that require dotfiles in order to operate sensibly (dotfiles that they do not create themselves automatically, that is) are a bad thing, and programs should be configured by the Debian default installation as close to normal as possible.

Therefore, if a program in a Debian package needs to be configured in some way in order to operate sensibly that configuration should be done in a site-wide global configuration file

elsewhere in **/etc**. Only if the program doesn't support a site-wide default configuration and the package maintainer doesn't have time to add it should a default per-user file be placed in **/etc/skel**.

**/etc/skel** should be as empty as we can make it. This is particularly true because there is no easy mechanism for ensuring that the appropriate dotfiles are copied into the accounts of existing users when a package is installed.

Ideally the sysadmin should not have to do any configuration other than that done (semi-)automatically by the **postinst** script.

---

## 4.8. Log files

Log files should usually be named **/var/log/***package***.log**. If you have many log files, or need a separate directory for permissions reasons (**/var/log** is writable only by **root**), you should usually create a directory named **/var/log/***package*.

Make sure that any log files are rotated occasionally so that they don't grow indefinitely; the best way to do this is to use **savelog** program in an **/etc/cron.daily**, **/etc/cron.weekly** or **/etc/cron.monthly** script. Here is a good example:

```
[ -d /var/log/apache/. ] || exit 0
            umask 022
            cd /var/log/apache
            if [ -fs access.log ]
            then
            savelog -c 7 access.log > /dev/null
            fi
```

Make sure that any log files are removed when the package is purged (but not when it is only removed), by checking the argument to the **postrm** script (see the *Debian Packaging Manual* for details).

---

## 4.9. Permissions and owners

The rules in this section are guidelines for general use. If necessary you may deviate from the details below. However, if you do so you must make sure that what is done is secure and you must try to be as consistent as possible with the rest of the system. You should probably also discuss it on **debian-devel** first.

Files should be owned by **root.root**, and made writable only by the owner and universally readable (and executable, if appropriate).

Directories should be mode 755 or (for group-writability) mode 2775. The ownership of the directory should be consistent with its mode--if a directory is mode 2775, it should be owned by the group that needs write access to it.

Setuid and setgid executables should be mode 4755 or 2755 respectively, and owned by the appropriate user or group. They should not be made unreadable (modes like 4711 or 2711 or even 4111); doing so achieves no extra security, because anyone can find the binary in the freely available Debian package--it is merely inconvenient. For the same reason you should not restrict

read or execute permissions on non-set-id executables.

Some setuid programs need to be restricted to particular sets of users, using file permissions. In this case they should be owned by the uid to which they are set-id, and by the group which should be allowed to execute them. They should have mode 4754; there is no point in making them unreadable to those users who must not be allowed to execute them.

Do not arrange that the system administrator can only reconfigure the package to correspond to their local security policy by changing the permissions on a binary. Ordinary files installed by **dpkg** (as opposed to conffiles and other similar objects) have their permissions reset to the distributed permissions when the package is reinstalled. Instead you should consider (for example) creating a group for people allowed to use the program(s) and making any setuid executables executable only by that group.

If you need to create a new user or group for your package there are two possibilities. Firstly, you may need to make some files in the binary package be owned by this user or group, or you may need to compile the user or group id (rather than just the name) into the binary (though this latter should be avoided if possible). In this case you need a statically allocated id.

You must ask for a user or group id from the base system maintainer, and must not release the package until you have been allocated one. Once you have been allocated one you must make the package depend on a version of the base system with the id present in **/etc/passwd** or **/etc/group**, or alternatively arrange for your package to create the user or group itself with the correct id (using **adduser**) in its pre- or post-installation script (the latter is to be preferred if it is possible).

On the other hand, the program may able to determine the uid or gid from the group name at runtime, so that a dynamic id can be used. In this case you must choose an appropriate user or group name, discussing this on **debian-devel** and checking with the base system maintainer that it is unique and that they do not wish you to use a statically allocated id instead. When this has been checked you must arrange for your package to create the user or group if necessary using **adduser** in the pre- or post-installation script (again, the latter is to be preferred if it is possible).

Note that changing the numeric value of an id associated with a name is very difficult, and involves searching the filesystem for all appropriate files. You need to think carefully whether a static or dynamic id is required, since changing your mind later will cause problems.

# 5. Customized programs

## 5.1. Architecture specification strings

If a program needs to specify an *architecture specification string* in some place, the following format has to be used:

**<arch>-<os>**

where '<arch>' is one of the following: i386, alpha, arm, m68k, powerpc, sparc and '<os>' is one of: linux, gnu. Use of *gnu* in this string is reserved for the GNU/Hurd operating system. .

Note, that we don't want to use '<arch>-debian-linux' to apply to the rule 'architecture-vendor-os' since this would make our programs incompatible to other Linux distributions. Also note, that we don't use '<arch>-unknown-linux', since the 'unknown' does not look very good.

## 5.2. Daemons

The configuration files **/etc/services**, **/etc/protocols**, and **/etc/rpc** are managed by the **netbase** package and may not be modified by other packages.

If a package requires a new entry in one of these files, the maintainer has to get in contact with the **netbase** maintainer, who will add the entries and release a new version of the **netbase** package.

The configuration file **/etc/inetd.conf** may be modified by the package's scripts only via the **update-inetd** script or the **DebianNet.pm** Perl module.

If a package wants to install an example entry into **/etc/inetd.conf**, the entry has to be preceded with exactly one hash character (#). Such lines are treated as 'commented out by user' by the **update-inetd** script and are not changed or activated during a package updates.

## 5.3. Editors and pagers

Some programs have the ability to launch an editor or pager program to edit or display a text document. Since there are lots of different editors and pagers available in the Debian distribution, the system administrator and each user should have the possibility to choose his/her preferred editor and pager.

In addition, every program should choose a good default editor/pager if none is selected by the user or system administrator.

Thus, every program that launches an editor or pager has to use the EDITOR or PAGER environment variables to determine the editor/pager the user wants to get started. If these variables are not set, the programs '/usr/bin/editor' and '/usr/bin/pager' have to be used, respectively.

These two files are managed through 'alternatives.' That is, every package providing an editor or pager has to call the 'update-alternatives' script to register these programs.

If it is very hard to adapt a program to make us of the EDITOR and PAGER variable, that program should be configured to use '/usr/bin/sensible-editor' and '/usr/bin/sensible-pager' as editor or pager program, respectively. These are two scripts provided in the Debian base system that check the EDITOR and PAGER variables and launches the appropriate program or falls back to '/usr/bin/editor' and '/usr/bin/pager', automatically.

Since the Debian base system already provides an editor and a pager program, there is no need for a package to depend on 'editor' and 'pager', nor is it necessary for a package to provide such virtual packages.

---

## 5.4. Web servers and applications

This section describes the locations and URLs that have to be used by all web servers and web application in the Debian system.

1. Cgi-bin executable files are installed in the directory
   **/usr/lib/cgi-bin/<cgi-bin-name>**
   and can be referred to as
   **http://localhost/cgi-bin/<cgi-bin-name>**

2. Access to html documents Html documents for a package are stored in /usr/doc/*package* and can be referred to as
   **http://localhost/doc/<package>/<filename>**

3. Web Document Root Web Applications should try to avoid storing files in the Web Document Root. Instead use the /usr/doc/<package> directory for documents and register the Web Application via the menu package. If access to the web-root is unavoidable then use
   **/var/www**
   as the Document Root. This might be just a symbolic link to the location where the sysadmin has put the real document root.

---

## 5.5. Mail transport agents

Debian packages which process electronic mail, whether mail-user-agents (MUAs) or mail-transport-agents (MTAs), *must* make sure that they are compatible with the configuration decisions below. Failure to do this may result in lost mail, broken **From**: lines, and other serious brain damage!

The mail spool is **/var/spool/mail** and the interface to send a mail message is **/usr/sbin/sendmail** (as per the FSSTND). The mail spool is part of the base system and not part of the MTA package.

All Debian MUAs and MTAs have to use the **maillock** and **mailunlock** functions provided by the **liblockfile** packages to lock and unlock mail boxes. These functions implement a NFS-safe locking mechanism. (It is ok if MUAs and MTAs don't link against liblockfile but use a *compatible* mechanism. Please compare the mechanisms very carefully!)

Mailboxes are generally 660 *user*.**mail** unless the user has chosen otherwise. A MUA may

remove a mailbox (unless it has nonstandard permissions) in which case the MTA or another MUA must recreate it if needed. Mailboxes must be writable by group mail.

The mail spool is 2775 **mail.mail**, and MUAs need to be setgid mail to do the locking mentioned above (and obviously need to avoid accessing other users' mailboxes using this privilege).

**/etc/aliases** is the source file for the system mail aliases (e.g., postmaster, usenet, etc.)--it is the one which the sysadmin and **postinst** scripts may edit. After **/etc/aliases** is edited the program or human editing it must call **newaliases**. All MTA packages should come with a **newaliases** program, even if it does nothing, but older MTA packages do not do this so programs should not fail if **newaliases** cannot be found.

The convention of writing **forward to** *address* in the mailbox itself is not supported. Use a **.forward** file instead.

The location for the **rmail** program used by UUCP for incoming mail is **/usr/sbin/rmail**, as per the FSSTND. Likewise, **rsmtp**, for receiving batch-SMTP-over-UUCP, is in **/usr/sbin/rsmtp** if it is supported.

If you need to know what name to use (for example) on outgoing news and mail messages which are generated locally, you should use the file **/etc/mailname**. It will contain the portion after the username and **@** (at) sign for email addresses of users on the machine (followed by a newline).

A package should check for the existence of this file. If it exists it should use it without comment. (An MTA's prompting configuration script may wish to prompt the user even if it finds this file exists.) If it does not exist it should prompt the user for the value and store it in **/etc/mailname** as well as using it in the package's configuration. The prompt should make it clear that the name will not just be used by that package. For example, in this situation the INN package says:

> **Please enter the 'mail name' of your system. This is the hostname portion of the address to be shown on outgoing news and mail messages. The default is** *syshostname*, **your system's host name. Mail name ['**`syshostname`**']:**

where *syshostname* is the output of **hostname --fqdn**.

---

## 5.6. News system configuration

All the configuration files related to the NNTP (news) servers and clients should be located under **/etc/news**.

There are some configuration issues that apply to a number of news clients and server packages on the machine. These are:

/etc/news/organization
: A string which shall appear as the organization header for all messages posted by NNTP clients on the machine

/etc/news/server
: Contains the FQDN of the upstream NNTP server, or localhost if the local machine is an NNTP server.

Other global files may be added as required for cross-package news configuration.

## 5.7. Programs for the X Windows system

Some programs can be configured with or without support for X Windows. Typically these binaries produced when configured for X will need the X shared libraries to run.

Such programs should be configured *with* X support, and should declare a dependency on **xlib6g** (for the X11R6 libraries). Users who wish to use the program can install just the relatively small **xlib6g** package, and do not need to install the whole of X.

Do not create two versions (one with X support and one without) of your package.

*Application defaults* files have to be installed in the directory **/usr/X11R6/lib/X11/app-defaults/**. They are considered as part of the program code. Thus, they should not be modified and should not be tagged as *conffile*. If the local system administrator wants to customise X applications globally, the file **/etc/X11/Xresources** should be used.

If you package a program that requires a non-free Motif library, it would be good if you can provide a "foo-smotif" and a "foo-dmotif" package, containing a (against Motif libraries) statically and a dynamically linked version, respectively. This way, users without Motif can use the package too, while users that have Motif installed get the advantages of a dynamically linked version.

However, if your package works reliably with lesstif, you should package it with lesstif, and not with Motif at all.

Note, that packages that require non-free Motif libraries can't go into the main section. If your package is free otherwise, it should go into contrib. Otherwise it has to go into non-free.

## 5.8. Emacs lisp programs

Please refer to the 'Debian Emacs Policy' (documented in **debian-emacs-policy.gz** of the **emacsen-common** package) for details of how to package emacs lisp programs.

## 5.9. Games

The permissions on /var/lib/games are 755 **root.root**.

Each game decides on its own security policy.

Games which require protected, privileged access to high-score files, savegames, etc., must be made set-*group*-id (mode 2755) and owned by **root.games**, and use files and directories with appropriate permissions (770 **root.games**, for example). They must *not* be made set-*user*-id, as this causes security problems. (If an attacker can subvert any set-user-id game they can overwrite the executable of any other, causing other players of these games to run a Trojan horse program. With a set-group-id game the attacker only gets access to less important game data, and if they can get at the other players' accounts at all it will take considerably more effort.)

Some packages, for example some fortune cookie programs, are configured by the upstream authors to install with their data files or other static information made unreadable so that they

can only be accessed through set-id programs provided. Do not do this in a Debian package: anyone can download the `.deb` file and read the data from it, so there is no point making the files unreadable. Not making the files unreadable also means that you don't have to make so many programs set-id, which reduces the risk of a security hole.

As described in the FSSTND, binaries of games should be installed in the directory `/usr/games`. This also applies to games that use the X windows system. Manual pages for games (X and non-X games) should be installed in `/usr/man/man6`.

# 6. Documentation

## 6.1. Manual pages

You must install manual pages in **nroff** source form, in appropriate places under **/usr/man**. You should only use sections 1 to 9 (see the FSSTND for more details). You must *not* install a preformatted 'cat page'.

If no manual page is available for a particular program, utility or function and this is reported as a bug on debian-bugs, a symbolic link from the requested manual page to the **undocumented(7)** manual page should be provided. This symbolic link can be created from **debian/rules** like this:

```
ln -s ../man7/undocumented.7.gz \
        debian/tmp/usr/man/man[1-9]/the_requested_manpage.[1-9].gz
```

This manpage claims that the lack of a manpage has been reported as a bug, so you may only do this if it really has (you can report it yourself, if you like). Do not close the bug report until a proper manpage is available.

You may forward a complaint about a missing manpage to the upstream authors, and mark the bug as forwarded in the Debian bug tracking system. Even though the GNU Project do not in general consider the lack of a manpage to be a bug, we do--if they tell you that they don't consider it a bug you should leave the bug in our bug tracking system open anyway.

Manual pages should be installed compressed using **gzip -9**.

If one manpage needs to be accessible via several names it is better to use a symbolic link than the **.so** feature, but there is no need to fiddle with the relevant parts of the upstream source to change from **.so** to symlinks--don't do it unless it's easy. Do not create hard links in the manual page directories, and do not put absolute filenames in **.so** directives. The filename in a **.so** in a manpage should be relative to the base of the manpage tree (usually **/usr/man**).

## 6.2. Info documents

Info documents should be installed in **/usr/info**. They should be compressed with **gzip -9**.

Your package must call **install-info** to update the Info **dir** file, in its post-installation script:

```
install-info --quiet --section Development Development \
        /usr/info/foobar.info
```

It is a good idea to specify a section for the location of your program; this is done with the **--section** switch. To determine which section to use, you should look at **/usr/info/dir** on your system and choose the most relevant (or create a new section if none of the current sections are relevant). Note that the **--section** flag takes two arguments; the first is a regular expression to match (case-insensitively) against an existing section, the second is used when creating a new one.

You must remove the entries in the pre-removal script:

```
install-info --quiet --remove /usr/info/foobar.info
```

If **install-info** cannot find a description entry in the Info file you will have to supply one. See **install-info(8)** for details.

## 6.3. Additional documentation

Any additional documentation that comes with the package can be installed at the discretion of the package maintainer. Text documentation should be installed in a directory **/usr/doc/***package*, where *package* is the name of the package, and compressed with **gzip -9** unless it is small.

If a package comes with large amounts of documentation which many users of the package will not require you should create a separate binary package to contain it, so that it does not take up disk space on the machines of users who do not need or want it installed.

It is often a good idea to put text information files (**README**s, changelogs, and so forth) that come with the source package in **/usr/doc/***package* in the binary package. However, you don't need to install the instructions for building and installing the package, of course!

## 6.4. Preferred documentation formats

The unification of Debian documentation is being carried out via HTML.

If your package comes with extensive documentation in a markup format that can be converted to various other formats you should if possible ship HTML versions in a binary package, in the directory **/usr/doc/***appropriate package* or its subdirectories.[1]

Other formats such as PostScript may be provided at your option.

## 6.5. Copyright information

Every package must be accompanied by a verbatim copy of its copyright and distribution license in the file /usr/doc/<package-name>/copyright. This file must neither be compressed nor be a symbolic link.

In addition, the copyright file must say where the upstream sources (if any) were obtained, and explain briefly what modifications were made in the Debian version of the package compared to the upstream one. It must name the original authors of the package and the Debian maintainer(s) who were involved with its creation.

/usr/doc/<package-name> may be a symbolic link to a directory in /usr/doc only if two packages both come from the same source and the first package has a "Depends" relationship on the second. These rules are important because copyrights must be extractable by mechanical means.

Packages distributed under the UCB BSD license, the Artistic license, the GNU GPL, and the GNU LGPL should refer to the files /usr/doc/copyright/BSD, /usr/doc/copyright/Artistic, /usr/doc/copyright/GPL, and /usr/doc/copyright/LGPL.

Do not use the copyright file as a general **README** file. If your package has such a file it should be installed in **/usr/doc/***package***/README** or **README.Debian** or some other appropriate place.

[1] The rationale: The important thing here is that HTML docs should be available in *some* package, not necessarily in the main binary package, though.

## 6.6. Examples

Any examples (configurations, source files, whatever), should be installed in a directory **/usr/doc/***package***/examples**. These files should not be referenced by any program--they're there for the benefit of the system administrator and users, as documentation only.

## 6.7. Changelog files

This installed file must contain a copy of the **debian/changelog** file from your Debian source tree, and a copy of the upstream changelog file if there is one. The debian/changelog file should be installed in **/usr/doc/***package* as **changelog.Debian.gz**. If the upstream changelog file is text formatted, it must be accessable as **/usr/doc/***package***/changelog.gz**. If the upstream changelog file is HTML formatted, it must be accessable as **/usr/doc/***package***/changelog.html.gz**. If the upstream changelog files do not already conform to this naming convention, then this may be achieved by either renaming the files or adding a symbolic link at the packaging developer's discretion.

Both should be installed compressed using **gzip -9**, as they will become large with time even if they start out small.

If the package has only one changelog which is used both as the Debian changelog and the upstream one because there is no separate upstream maintainer then that changelog should usually be installed as **/usr/doc/***package***/changelog.gz**; if there is a separate upstream maintainer, but no upstream changelog, then the Debian changelog should still be called **changelog.Debian.gz**.