The Debian GNU/Linux FAQ

'Authors' on page 67

version 3.0.2, 28 January 2003

Abstract

This document answers questions frequently asked about Debian GNU/Linux.

Copyright Notice

Copyright © 1996-2003 by Software in the Public Interest

Permission is granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this document under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this document into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

Contents

1 Definitions and overview **1**

1.1 What is Debian GNU/Linux? 1

1.2 OK, now I know what Debian is... what is Linux?! 2

1.3 What is this new "Hurd" thing? 3

1.4 What is the difference between Debian GNU/Linux and other Linux distributions? Why should I choos

1.5 How does the Debian project fit in or compare with the Free Software Foundation's GNU project? 4

1.6 How does one pronounce Debian and what does this word mean? 4

2 Getting and installing Debian GNU/Linux **5**

2.1 What is the latest version of Debian? 5

2.2 Where/how can I get the Debian installation disks? 5

2.3 How do I install the Debian from CD-ROMs? 6

2.4 I have my own CD-writer, are there CD images available somewhere? 6

2.5 Can I install it from a pile of floppy disks? 6

2.6 Can I get and install Debian directly from a remote Internet site? 7

3 Compatibility issues **9**

3.1 On what hardware architectures/systems does Debian GNU/Linux run? 9

3.2 How compatible is Debian with other distributions of Linux? 10

3.3 How source code compatible is Debian with other Unix systems? 10

3.4 Can I use Debian packages (".deb" files) on my RedHat/Slackware/... Linux system? Can I use RedHat

3.5 Is Debian able to run my very old "a.out" programs? 11

3.6 Is Debian able to run my old libc5 programs? 12

3.7 Can Debian be used to compile libc5 programs? 12

3.8 How should I install a non-Debian program? . . . 12
3.9 Why do I get "Can't find libX11.so.6" errors when I try to run `foo`? . . . 13
3.10 Why can't I compile programs that require libtermcap? . . . 13
3.11 Why can't I install AccelX? . . . 13
3.12 Why do my old XFree 2.1 Motif applications crash? . . . 13

4 Software available in the Debian system **15**
4.1 What types of applications and development software are available for Debian GNU/Linux? 15
4.2 Who wrote all that software? . . . 16
4.3 How can I get a current list of programs that have been packaged for Debian? . . . 16
4.4 What is missing from Debian GNU/Linux? . . . 16
4.5 Why do I get "ld: cannot find -lfoo" messages when compiling programs? Why aren't there any libfoo.s
4.6 (How) Does Debian support Java? . . . 17
4.7 How can I check that I am using a Debian system, and what version is it? . . . 17
4.8 How does Debian support non-English languages? . . . 17
4.9 What about the US export regulation limitations? . . . 18
4.10 Where is pine? . . . 18

5 The Debian FTP archives **19**
5.1 What are all those directories at the Debian FTP archives? . . . 19
5.2 How many Debian distributions are there in the `dists` directory? . . . 20
5.3 What are all those names like slink, potato, etc.? . . . 20
5.3.1 Which other codenames have been used in the past? . . . 20
5.3.2 Where do these codenames come from? . . . 20
5.4 What about "frozen"? . . . 21
5.5 What about "sid"? . . . 21
5.5.1 Historical notes about "sid" . . . 21
5.6 What does the stable directory contain? . . . 22
5.7 What does the testing directory contain? . . . 22
5.8 What does the unstable directory contain? . . . 22
5.9 What are all those directories inside `dists/stable/main`? . . . 23
5.10 Where is the source code? . . . 23
5.11 What's in the `pool` directory? . . . 24
5.12 What is "incoming"? . . . 24

6 Basics of the Debian package management system **25**

6.1 What is a Debian package? . . . 25
6.2 What is the format of a Debian binary package? . . . 26
6.3 Why are Debian package file names so long? . . . 26
6.4 What is a Debian control file? . . . 27
6.5 What is a Debian conffile? . . . 28
6.6 What is a Debian preinst, postinst, prerm, and postrm script? . . . 28
6.7 What is a *Required*, *Important*, *Standard*, *Optional*, or *Extra* package? . . . 29
6.8 What is a Virtual Package? . . . 29
6.9 What is meant by saying that a package *Depends*, *Recommends*, *Suggests*, *Conflicts*, *Replaces* or *Provides* and
6.10 What is meant by Pre-Depends? . . . 31
6.11 What is meant by *unknown*, *install*, *remove purge* and *hold* in the package status? . . . 31
6.12 How do I put a package on hold? . . . 31
6.13 How do I install a source package? . . . 32
6.14 How do I build binary packages from a source package? . . . 33
6.15 How do I create Debian packages myself? . . . 33

7 The Debian package management tools **35**

7.1 What programs does Debian provide for managing its packages? . . . 35
7.1.1 dpkg . . . 35
7.1.2 dselect . . . 36
7.1.3 dpkg-deb . . . 38
7.1.4 apt-get . . . 38
7.1.5 dpkg-split . . . 38
7.2 Debian claims to be able to update a running program; how is this accomplished? 39
7.3 How can I tell what packages are already installed on a Debian system? . . . 39
7.4 How can I find out what package produced a particular file? . . . 40

8 Keeping your Debian system up-to-date **41**

8.1 How can I upgrade my Debian 1.3.1 (or earlier) distribution, based on libc5, to 2.0 (or later), based on lib
8.2 How can I keep my Debian system current? . . . 42
8.2.1 APT . . . 42

8.2.2 dpkg-ftp . . . 43
8.2.3 mirror . . . 43
8.2.4 dpkg-mountable . . . 44
8.3 Must I go into single user mode in order to upgrade a package? . . . 44
8.4 Do I have to keep all those .deb archive files on my disk? . . . 44
8.5 How can I keep a log of the packages I added to the system? . . . 44

9 Debian and the kernel **45**
9.1 Can I install and compile a kernel without some Debian-specific tweaking? . . . 45
9.2 What tools does Debian provide to build custom kernels? . . . 45
9.3 How can I make a custom boot floppy? . . . 46
9.4 What special provisions does Debian provide to deal with modules? . . . 47
9.5 Can I safely de-install an old kernel package, and if so, how? . . . 47

10 Customizing your installation of Debian GNU/Linux **49**
10.1 How can I ensure that all programs use the same paper size? . . . 49
10.2 How can I provide access to hardware peripherals, without compromising security? 49
10.3 How do I load a console font on startup the Debian way? . . . 49
10.4 How can I configure an X11 program's application defaults? . . . 50
10.5 Every distribution seems to have a different boot-up method. Tell me about Debian's. 50
10.6 It looks as if Debian does not use `rc.local` to customize the boot process; what facilities are provided?
10.7 How does the package management system deal with packages that contain configuration files for othe
10.8 How do I override a file installed by a package, so that a different version can be used instead? 52
10.9 How can I have my locally-built package included in the list of available packages that the package mar
10.10 Some users like mawk, others like gawk; some like vim, others like elvis; some like trn, others like tin; h

11 Getting support for Debian GNU/Linux **55**
11.1 What other documentation exists on and for a Debian system? . . . 55
11.2 Are there any on-line resources for discussing Debian? . . . 56
11.2.1 Mailing lists . . . 56
11.2.2 Maintainers . . . 57
11.2.3 Usenet newsgroups . . . 57

11.3 Is there a quick way to search for information on Debian GNU/Linux? 57
11.4 Are there logs of known bugs? 58
11.5 How do I report a bug in Debian? 58

12 Contributing to the Debian Project **61**
12.1 How can I become a Debian software developer? 61
12.2 How can I contribute resources to the Debian project? 61
12.3 How can I contribute financially to the Debian project? 62
12.3.1 Software in the Public Interest 62
12.3.2 Free Software Foundation 62

13 Redistributing Debian GNU/Linux in a commercial product **63**
13.1 Can I make and sell Debian CDs? 63
13.2 Can Debian be packaged with non-free software? 63
13.3 I am making a special Linux distribution for a “vertical market”. Can I use Debian GNU/Linux for the g
13.4 Can I put my commercial program in a Debian “package” so that it installs effortlessly on any Debian sy

14 Changes expected in the next major release of Debian **65**
14.1 Increased security 65
14.2 Extended support for non-English users 65
14.3 More architectures 65
14.4 More kernels 66

15 General information about the FAQ **67**
15.1 Authors 67
15.2 Feedback 67
15.3 Availability 68
15.4 Document format 68

Chapter 1

Definitions and overview

1.1 What is Debian GNU/Linux?

Debian GNU/Linux is a particular *distribution* of the Linux operating system, and numerous packages that run on it.

In principle, users could obtain the Linux kernel via the Internet or from elsewhere, and compile it themselves. They could then obtain source code for many applications in the same way, compile the programs, then install them into their systems. For complicated programs, this process can be not only time-consuming but error-prone. To avoid it, users often choose to obtain the operating system and the application packages from one of the Linux distributors. What distinguishes the various Linux distributors are the software, protocols, and practices they use for packaging, installing, and tracking applications packages on users' systems, combined with installation and maintenance tools, documentation, and other services.

Debian GNU/Linux is the result of a volunteer effort to create a free, high-quality Unix-compatible operating system, complete with a suite of applications. The idea of a free Unix-like system originates from the GNU project, and many of the applications that make Debian GNU/Linux so useful were developed by the GNU project.

For Debian, free has the GNUish meaning (see the Debian Free Software Guidelines (`http://www.debian.org/social_contract#guidelines`)). When we speak of free software, we are referring to freedom, not price. Free software means that you have the freedom to distribute copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

The Debian Project was created by Ian Murdock in 1993, initially under the sponsorship of the Free Software Foundation's GNU project. Today, Debian's developers think of it as a direct descendent of the GNU project.

Debian GNU/Linux is:

- **full featured**: Debian includes more than 8250 software packages at present. Users can select which packages to install; Debian provides a tool for this purpose. You can find a

list and descriptions of the packages currently available in Debian at any of the Debian mirror sites (http://www.debian.org/distrib/ftplist).

- **free to use and redistribute**: There is no consortium membership or payment required to participate in its distribution and development. All packages that are formally part of Debian GNU/Linux are free to redistribute, usually under terms specified by the GNU General Public License.

  The Debian FTP archives also carry approximately 350 software packages (in the `non-free` and `contrib` sections), which are distributable under specific terms included with each package.

- **dynamic**: With about 900 volunteers constantly contributing new and improved code, Debian is evolving rapidly. New releases are planned to be made every several months, and the FTP archives are updated daily.

Although Debian GNU/Linux itself is free software, it is a base upon which value-added Linux distributions can be built. By providing a reliable, full-featured base system, Debian provides Linux users with increased compatibility, and allows Linux distribution creators to eliminate duplication of effort and focus on the things that make their distribution special. See 'I am making a special Linux distribution for a "vertical market". Can I use Debian GNU/Linux for the guts of a Linux system and add my own applications on top of it?' on page 64 for more information.

1.2 OK, now I know what Debian is... what is Linux?!

In short, Linux is the kernel of a Unix-like operating system. It was originally designed for 386 (and better) PCs; now, ports to other systems, including multi-processor systems, are under development. Linux is written by Linus Torvalds and many computer scientists around the world.

Besides its kernel, a "Linux" system usually has:

- a file system that follows the Linux Filesystem Hierarchy Standard http://www.pathname.com/fhs/.
- a wide range of Unix utilities, many of which have been developed by the GNU project and the Free Software Foundation.

The combination of the Linux kernel, the file system, the GNU and FSF utilities, and the other utilities are designed to achieve compliance with the POSIX (IEEE 1003.1) standard; see 'How source code compatible is Debian with other Unix systems?' on page 10.

For more information about Linux, see Michael K. Johnson's Linux Information Sheet (ftp://ibiblio.org/pub/Linux/docs/HOWTO/INFO-SHEET) and Meta-FAQ (ftp://ibiblio.org/pub/Linux/docs/HOWTO/META-FAQ).

1.3 What is this new "Hurd" thing?

The Hurd is a set of servers running on top of the GNU Mach microkernel. Together they build the base for the GNU operating system.

Currently, Debian is only available for Linux, but with Debian GNU/Hurd we have started to offer the Hurd as a development, server and desktop platform, too. However, Debian GNU/Hurd is not officially released yet, and won't be for some time.

Please see http://www.gnu.org/software/hurd/ for more information about the GNU/Hurd in general, and http://www.debian.org/ports/hurd/ for more information about Debian GNU/Hurd.

1.4 What is the difference between Debian GNU/Linux and other Linux distributions? Why should I choose Debian over some other distribution?

These key features distinguish Debian from other Linux distributions:

The Debian package maintenance system: The entire system, or any individual component of it, can be upgraded in place without reformatting, without losing custom configuration files, and (in most cases) without rebooting the system. Most Linux distributions available today have some kind of package maintenance system; the Debian package maintenance system is unique and particularly robust. (see 'Basics of the Debian package management system' on page 25)

Open development: Whereas other Linux distributions are developed by individuals, small, closed groups, or commercial vendors, Debian is the only Linux distribution that is being developed cooperatively by many individuals through the Internet, in the same spirit as Linux and other free software.

More than 900 volunteer package maintainers are working on over 8250 packages and improving Debian GNU/Linux. The Debian developers contribute to the project not by writing new applications (in most cases), but by packaging existing software according to the standards of the project, by communicating bug reports to upstream developers, and by providing user support. See also additional information on how to become a contributor in 'How can I become a Debian software developer?' on page 61.

The Bug Tracking System: The geographical dispersion of the Debian developers required sophisticated tools and quick communication of bugs and bug-fixes to accelerate the development of the system. Users are encouraged to send bugs in a formal style, which are quickly accessible by WWW archives or via e-mail. See additional information in this FAQ on the management of the bug log in 'Are there logs of known bugs?' on page 58.

The Debian Policy: Debian has an extensive specification of our standards of quality, the Debian Policy. This document defines the qualities and standards to which we hold Debian packages.

For additional information about this, please see our web page about reasons to choose Debian (http://www.debian.org/intro/why_debian).

1.5 How does the Debian project fit in or compare with the Free Software Foundation's GNU project?

The Debian system builds on the ideals of free software first championed by the Free Software Foundation (http://www.gnu.org/) and in particular by Richard Stallman (http://www.stallman.org/). FSF's powerful system development tools, utilities, and applications are also a key part of the Debian system.

The Debian Project is a separate entity from the FSF, however we communicate regularly and cooperate on various projects. The FSF explicitly requested that we call our system "Debian GNU/Linux", and we are happy to comply with that request.

The FSF's long-standing objective is to develop a new operating system called GNU, based on Hurd (http://www.gnu.org/software/hurd/). Debian is working with FSF on this system, called Debian GNU/Hurd (http://www.debian.org/ports/hurd/).

1.6 How does one pronounce Debian and what does this word mean?

The project name is pronounced Deb'-ee-en, with a short e in Deb, and emphasis on the first syllable. This word is a contraction of the names of Debra and Ian Murdock, who founded the project. (Dictionaries seem to offer some ambiguity in the pronunciation of Ian (!), but Ian prefers ee'-en.)

Chapter 2

Getting and installing Debian GNU/Linux

2.1 What is the latest version of Debian?

Currently there are three versions of Debian GNU/Linux:

release 3.0, a.k.a. the 'stable' distribution This is stable and well tested software, it changes if major security or usability fixes are incorporated.

the 'testing' distribution This is where packages that will be released as the next 'stable' are placed; they've had some testing in unstable but they may not be completely fit for release yet. This distribution is updated more often than 'stable', but not more often than 'unstable'.

the 'unstable' distribution This is the version currently under development; it is updated continuously. You can retrieve packages from the 'unstable' archive on any Debian FTP site and use them to upgrade your system at any time, but you may not expect the system to be as usable or as stable as before - that's why it's called '**unstable**'!

Please see 'How many Debian distributions are there in the `dists` directory?' on page 20 for more information.

2.2 Where/how can I get the Debian installation disks?

You can get the installation disks by downloading the appropriate files from one of the Debian mirrors (`http://www.debian.org/mirror/list`).

The installation system files are separated in subdirectories of `dists/stable/main` directory, and the names of these subdirectories correspond to your architecture like this:

`disks-arch` (*arch* is "i386", "sparc", etc, check the site for an exact list). In each of these architecture subdirectories there can be several directories, each for a version of the installation system, and the currently used one is in the 'current' directory (that's a symbolic link).

See the `README.txt` file in that directory for further instructions.

2.3 How do I install the Debian from CD-ROMs?

Linux supports the ISO 9660 (CD-ROM) file system with Rock Ridge extensions (formerly known as "High Sierra"). Several vendors (`http://www.debian.org/CD/vendors/`) provide Debian GNU/Linux in this format.

Warning: When installing from CD-ROM, it is usually not a good idea to choose dselect's `cdrom` access method. This method is usually very slow. The `mountable` and `apt` methods, for example, are much better for installing from CD-ROM (see 'dpkg-mountable' on page 44 and 'APT' on page 42).

2.4 I have my own CD-writer, are there CD images available somewhere?

Yes. To make it easier for CD vendors to provide high quality disks, we provide the Official CD images (`http://cdimage.debian.org/`).

2.5 Can I install it from a pile of floppy disks?

First of all, a warning: whole Debian GNU/Linux is way too large to be installed from media as small as a standard 1.44MB floppy disk - you may not find installing from floppies a very pleasant experience.

Copy the Debian packages onto formatted floppy disks. Either a DOS, the native Linux "ext2", or the "minix" format will do; one just has to use a mount command appropriate to the floppy being used.

Using floppy disks has these complications:

- Short MS-DOS file names: If you are trying to place Debian package files onto MS-DOS formatted disks, you will find that their names are generally too long, and do not conform to the MS-DOS 8.3 filename limitation. To overcome this, you would have to use VFAT formatted disks, since VFAT supports longer file names.
- Large file sizes: Some packages are larger than 1.44 MBytes, and will not fit onto a single floppy disk. To solve this problem, use the dpkg-split tool (see 'dpkg-split' on page 38), available in the `tools` directory on Debian mirrors (`http://www.debian.org/mirror/list`).

You must have support in the kernel for floppy disks in order to read and write to floppy disk; most kernels come with floppy drive support included in them.

To mount a floppy disk under the mount point `/floppy` (a directory which should have been created during installation), use:

- `mount -t msdos /dev/fd0 /floppy/`
  if the floppy disk is in drive A: and has an MS-DOS file system,
- `mount -t msdos /dev/fd1 /floppy/`
  if the floppy disk is in drive B: and has an MS-DOS file system,
- `mount -t ext2 /dev/fd0 /floppy/`
  if the floppy disk is in drive A: and has an ext2 (i.e., a normal Linux) file system.

2.6 Can I get and install Debian directly from a remote Internet site?

Yes. You can boot the Debian installation system from a set of files you can download from our FTP site and its mirrors.

You can download a small CD image file, create a bootable CD from it, install the basic system from it and the rest over the network. For more information please see `http://www.debian.org/CD/netinst/`.

You can also download even smaller floppy disk image files, create bootable diskettes from them, start the installation procedure and get the rest of Debian over the network. For more information, please see `http://www.debian.org/distrib/floppyinst`.

Chapter 3

Compatibility issues

3.1 On what hardware architectures/systems does Debian GNU/Linux run?

Debian GNU/Linux includes complete source-code for all of the included programs, so it should work on all systems which are supported by the Linux kernel; see the Linux FAQ (http://en.tldp.org/FAQ/Linux-FAQ/intro.html#DOES-LINUX-RUN-ON-MY-COMPUTER) for details.

The current Debian GNU/Linux release, 3.0, contains a complete, binary distribution for the following architectures:

i386: this covers PCs based on Intel and compatible processors, including Intel's 386, 486, Pentium, Pentium Pro, Pentium II (both Klamath and Celeron), and Pentium III, and most compatible processors by AMD, Cyrix and others.

m68k: this covers Amigas and ATARIs having a Motorola 680x0 processor for x>=2; with MMU.

alpha: Compaq/Digital's Alpha systems.

sparc: this covers Sun's SPARC and most UltraSPARC systems.

powerpc: this covers some IBM/Motorola PowerPC machines, including CHRP, PowerMac and PReP machines.

arm: ARM and StrongARM machines.

mips: SGI's big-endian MIPS systems, Indy and Indigo2; *mipsel*: little-endian MIPS machines, Digital DECstations.

hppa: Hewlett-Packard's PA-RISC machines (712, C3000, L2000, A500).

ia64: Intel IA-64 ("Itanium") computers.

s390: IBM S/390 mainframe systems.

The development of binary distributions of Debian for Sparc64 (UltraSPARC native) architectures is currently underway.

For further information on booting, partitioning your drive, enabling PCMCIA (PC Card) devices and similar issues please follow the instructions given in the Installation Manual, which is available from our WWW site at http://www.debian.org/releases/stable/installmanual.

3.2 How compatible is Debian with other distributions of Linux?

Debian developers communicate with other Linux distribution creators in an effort to maintain binary compatibility across Linux distributions. Most commercial Linux products run as well under Debian as they do on the system upon which they were built.

Debian GNU/Linux adheres to the Linux Filesystem Hierarchy Standard (http://www.pathname.com/fhs/). However, there is room for interpretation in some of the rules within this standard, so there may be slight differences between a Debian system and other Linux systems.

3.3 How source code compatible is Debian with other Unix systems?

For most applications Linux source code is compatible with other Unix systems. It supports almost everything that is available in System V Unix systems and the free and commercial BSD-derived systems. However in the Unix business such claim has nearly no value because there is no way to prove it. In the software development area complete compatibility is required instead of compatibility in "about most" cases. So years ago the need for standards arose, and nowadays POSIX.1 (IEEE Standard 1003.1-1990) is one of the major standards for source code compatibility in Unix-like operating systems.

Linux is intended to adhere to POSIX.1, but the POSIX standards cost real money and the POSIX.1 (and FIPS 151-2) certification is quite expensive; this made it more difficult for the Linux developers to work on complete POSIX conformance. The certification costs make it unlikely that Debian will get an official conformance certification even if it completely passed the validation suite. (The validation suite is now freely available, so it is expected that more people will work on POSIX.1 issues.)

Unifix GmbH (Braunschweig, Germany) developed a Linux system that has been certified to conform to FIPS 151-2 (a superset of POSIX.1). This technology was available in Unifix' own distribution called Unifix Linux 2.0 and in Lasermoon's Linux-FT.

3.4 Can I use Debian packages (".deb" files) on my RedHat/Slackware/...Linux system? Can I use RedHat packages (".rpm" files) on my Debian GNU/Linux system?

Different Linux distributions use different package formats and different package management programs.

You probably can: A program to unpack a Debian package onto a Linux host that is been built from a 'foreign' distribution is available, and will generally work, in the sense that files will be unpacked. The converse is probably also true, that is, a program to unpack a RedHat or Slackware package on a host that is based on Debian GNU/Linux will probably succeed in unpacking the package and placing most files in their intended directories. This is largely a consequence of the existence (and broad adherence to) the Linux Filesystem Hierarchy Standard. The Alien (`http://packages.debian.org/alien`) package is used to convert between different package formats.

You probably do not want to: Most package managers write administrative files when they are used to unpack an archive. These administrative files are generally not standardized. Therefore, the effect of unpacking a Debian package on a 'foreign' host will have unpredictable (certainly not useful) effects on the package manager on that system. Likewise, utilities from other distributions might succeed in unpacking their archives on Debian systems, but will probably cause the Debian package management system to fail when the time comes to upgrade or remove some packages, or even simply to report exactly what packages are present on a system.

A better way: The Linux File System Standard (and therefore Debian GNU/Linux) requires that subdirectories under `/usr/local/` be entirely under the user's discretion. Therefore, users can unpack 'foreign' packages into this directory, and then manage their configuration, upgrade and removal individually.

3.5 Is Debian able to run my very old "a.out" programs?

Do you actually still have such a program? :-)

To *execute* a program whose binary is in `a.out` (i.e., QMAGIC or ZMAGIC) format,

- Make sure your kernel has `a.out` support built into it, either directly (CONFIG_BINFMT_AOUT=y) or as a module (CONFIG_BINFMT_AOUT=m). (Debian's kernel-image package contains the module `binfmt_aout`.)

  If your kernel supports `a.out` binaries by a module, then be sure that the `binfmt_aout` module is loaded. You can do this at boot time by entering the line `binfmt_aout` into the file `/etc/modules`. You can do it from the command line by executing the command `insmod DIRNAME/binfmt_aout.o` where `DIRNAME` is the name of the directory where the modules that have been built for the version of the kernel now running are stored. On a system with the 2.2.17 version of the kernel, `DIRNAME` is likely to be `/lib/modules/2.2.17/fs/`.

- install the package `libc4`, found in one of the releases prior to release 2.0 (because at that time we removed the package). You might want to look for an old Debian CD-ROM (Debian 1.3.1 still had this package), or see `ftp://archive.debian.org/debian-archive/` on the Internet.

- If the program you want to execute is an `a.out` X client, then install the `xcompat` package (see above for availability).

If you have a commercial application in `a.out` format, now would be a good time to ask them to send you an `ELF` upgrade.

3.6 Is Debian able to run my old libc5 programs?

Yes. Just install the required `libc5` libraries, from the `oldlibs` section (containing old packages included for compatibility with older applications).

3.7 Can Debian be used to compile libc5 programs?

Yes. Install `libc5-altdev` and `altgcc` packages (from the `oldlibs` section). You can find the appropriate libc5-compiled `gcc` and `g++` in directory `/usr/i486-linuxlibc1/bin`. Put them in your $PATH variable to get `make` and other programs to execute these first.

If you need to compile libc5 X clients, install `xlib6` and `xlib6-altdev` packages.

Be aware that libc5 environment isn't fully supported by our other packages anymore.

3.8 How should I install a non-Debian program?

Files under the directory `/usr/local/` are not under the control of the Debian package management system. Therefore, it is good practice to place the source code for your program in /usr/local/src/. For example, you might extract the files for a package named "foo.tar" into the directory `/usr/local/src/foo`. After you compile them, place the binaries in `/usr/local/bin/`, the libraries in `/usr/local/lib/`, and the configuration files in `/usr/local/etc/`.

If your programs and/or files really must be placed in some other directory, you could still store them in `/usr/local/`, and build the appropriate symbolic links from the required location to its location in `/usr/local/`, e.g., you could make the link

```
ln -s /usr/local/bin/foo /usr/bin/foo
```

In any case, if you obtain a package whose copyright allows redistribution, you should consider making a Debian package of it, and uploading it for the Debian system. Guidelines for becoming a package developer are included in the Debian Policy manual (see 'What other documentation exists on and for a Debian system?' on page 55).

3.9 Why do I get "Can't find libX11.so.6" errors when I try to run `foo`?

This error message could mean that the program is linked against the `libc5` version of the X11 libraries. In this case you need to install the `xlib6` package, from the `oldlibs` section.

You may get similar error messages referring to libXpm.so.4 file, in which case you need to install the libc5 version of the XPM library, from the `xpm4.7` package, also in the `oldlibs` section.

3.10 Why can't I compile programs that require libtermcap?

Debian uses the `terminfo` database and the `ncurses` library of terminal interface routes, rather than the `termcap` database and the `termcap` library. Users who are compiling programs that require some knowledge of the terminal interface should replace references to `libtermcap` with references to `libncurses`.

To support binaries that have already been linked with the `termcap` library, and for which you do not have the source, Debian provides a package called `termcap-compat`. This provides both `libtermcap.so.2` and `/etc/termcap`. Install this package if the program fails to run with the error message "can't load library 'libtermcap.so.2"', or complains about a missing `/etc/termcap` file.

3.11 Why can't I install AccelX?

AccelX uses the `termcap` library for installation. See 'Why can't I compile programs that require libtermcap?' on the current page above.

3.12 Why do my old XFree 2.1 Motif applications crash?

You need to install the `motifnls` package, which provides the XFree-2.1 configuration files needed to allow Motif applications compiled under XFree-2.1 to run under XFree-3.1.

Without these files, some Motif applications compiled on other machines (such as Netscape) may crash when attempting to copy or paste from or to a text field, and may also exhibit other problems.

Chapter 4

Software available in the Debian system

4.1 What types of applications and development software are available for Debian GNU/Linux?

Like most Linux distributions, Debian GNU/Linux provides:

- the major GNU applications for software development, file manipulation, and text processing, including gcc, g++, make, texinfo, Emacs, the Bash shell and numerous upgraded Unix utilities,
- Perl, Python, Tcl/Tk and various related programs, modules and libraries for each of them,
- TeX (LaTeX) and Lyx, dvips, Ghostscript,
- the X Window System, which provides a networked graphical user interface for Linux, and countless X applications including GNOME,
- a full suite of networking applications, including servers for Internet protocols such as HTTP (WWW), FTP, NNTP (news), SMTP and POP (mail) and name server; also provided are web browsers, and development tools.

More than 7890 packages, ranging from news servers and readers to sound support, FAX programs, database and spreadsheet programs, image processing programs, communications, net, and mail utilities, Web servers, and even ham-radio programs are included in the distribution. Another 350 software suites are available as Debian packages, but are not formally part of Debian due to license restrictions.

4.2 Who wrote all that software?

For each package the *authors* of the program(s) are credited in the file `/usr/doc/PACKAGE/copyright`, where PACKAGE is to be substituted with the package's name.

Maintainers who package this software for the Debian GNU/Linux system are listed in the Debian control file (see 'What is a Debian control file?' on page 27) that comes with each package.

4.3 How can I get a current list of programs that have been packaged for Debian?

A complete list is available in two parts:

the list of packages that can be distributed everywhere from any of the Debian mirrors (`http://www.debian.org/distrib/ftplist`), in the file `indices/Maintainers`. That file includes the package names and the names and e-mails of their respective maintainers.

the list of packages that cannot be exported from the US from any of the Debian non-US mirrors (`http://www.debian.org/misc/README.non-US`), in the file `indices-non-US/Maintainers`. That file includes the package names and the names and e-mails of their respective maintainers.

The WWW interface to the Debian packages (`http://packages.debian.org/`) conveniently summarizes the packages in each of about twenty "sections" of the Debian archive.

4.4 What is missing from Debian GNU/Linux?

A list of packages which are still needed to be packaged for Debian exists, the Work-Needing and Prospective Packages list (`http://www.debian.org/devel/wnpp/`).

For more details about adding the missing things, see 'How can I become a Debian software developer?' on page 61.

4.5 Why do I get "ld: cannot find -lfoo" messages when compiling programs? Why aren't there any libfoo.so files in Debian library packages?

Debian Policy requires that such symbolic links (to libfoo.so.x.y.z or similar) are placed in separate, development packages. Those packages are usually named libfoo-dev or libfooX-dev (presuming the library package is named libfooX, and X is a whole number).

4.6 (How) Does Debian support Java?

Since the official Java Development kit from Sun Microsystems is non-free software, it cannot be included in Debian proper. However, both the JDK and several *free* implementations of Java technology are available as Debian packages. You can write, debug and run Java programs using Debian.

Running a Java applet requires a web browser with the capability to recognize and execute them. Several web browsers available in Debian, such as Mozilla or Konqueror, support Java plug-ins that enable running Java applets within them. Netscape Navigator, while non-free, is also available as a Debian package and it can run Java applets.

Please refer to the Debian Java FAQ (`http://www.debian.org/doc/manuals/debian-java-faq/`) for more information.

4.7 How can I check that I am using a Debian system, and what version is it?

In order to make sure that your system has been installed from the real Debian base disks check for the existence of `/etc/debian_version` file, which contains a single one-line entry giving the version number of the release, as defined by the package `base-files`.

The existence of the program `dpkg` shows that you should be able to install Debian packages on your system, but as the program has been ported to many other operating systems and architectures, this is no longer a reliable method of determining is a system Debian GNU/Linux.

Users should be aware, however, that the Debian system consists of many parts, each of which can be updated (almost) independently. Each Debian "release" contains well defined and unchanging contents. Updates are separately available. For a one-line description of the installation status of package `foo`, use the command `dpkg --list foo`. To view versions of all installed packages, run:

```
dpkg -l
```

For a more verbose description, use:

```
dpkg --status foo
```

4.8 How does Debian support non-English languages?

- Debian GNU/Linux is distributed with keymaps for nearly two dozen keyboards, and with utilities (in the `kbd` package) to install, view, and modify the tables.

  The installation prompts the user to specify the keyboard he will use.

- Vast majority of the software we packaged supports entering non-US-ASCII characters used in other Latin languages (e.g. ISO-8859-1 or ISO-8859-2), and a number of programs support multi-byte languages such as Japanese or Chinese.
- Currently, support for German-, Spanish-, Finnish-, French-, Hungarian-, Italian-, Japanese-, Korean- and Polish-language manual pages is provided through the `manpages-LANG` packages (where LANG is the two-letter ISO country code). To access an NLS manual page, the user must set the shell LC_MESSAGES variable to the appropriate string.

  For example, in the case of the Italian-language manual pages, LC_MESSAGES needs to be set to 'italian'. The `man` program will then search for Italian manual pages under `/usr/share/man/it/`.

4.9 What about the US export regulation limitations?

US laws place restrictions on the export of defense articles, which includes some types of cryptographic software. PGP and ssh, among others, fall into this category.

To prevent anyone from taking unnecessary legal risks, certain Debian GNU/Linux packages are only available from a non-US site `ftp://non-US.debian.org/debian-non-US/`. There are numerous mirror sites all of which are also outside of the US, see `ftp://non-US.debian.org/debian-non-US/README.non-US` for a full list.

4.10 Where is pine?

Due to its restrictive license, it's in the non-free area. Moreover, since license does not even allow modified binaries to be distributed, you have to compile it yourself from the source and the Debian patches.

The source package name is `pine`. You can use the `pine-tracker` package to be notified about when you need to upgrade.

Note that there are many replacements for both pine and pico, such as `mutt` and `nano`, that are located in the main section.

Chapter 5

The Debian FTP archives

5.1 What are all those directories at the Debian FTP archives?

The software that has been packaged for Debian GNU/Linux is available in one of several directory trees on each Debian mirror site.

The `dists` directory is short for "distributions", and it is the canonical way to access the currently available Debian releases (and pre-releases).

The `pool` directory contains the actual packages, see 'What's in the `pool` directory?' on page 24.

There are the following supplementary directories:

/*tools*/: DOS utilities for creating boot disks, partitioning your disk drive, compressing/decompressing files, and booting Linux.

/*doc*/: The basic Debian documentation, such as the FAQ, the bug reporting system instructions, etc.

/*indices*/: The Maintainers file and the override files.

/*project*/: mostly developer-only materials, such as:

***project/experimental/*:** This directory contains packages and tools which are still being developed, and are still in the alpha testing stage. Users shouldn't be using packages from here, because they can be dangerous and harmful even for most experienced people.

***project/orphaned/*:** Packages that have been orphaned by their old maintainers, and withdrawn from the distribution.

5.2 How many Debian distributions are there in the `dists` directory?

Normally there are three distributions, the "stable" distribution, the "testing" distribution, and the "unstable" distribution. Sometimes there is also a "frozen" distribution (see 'What about "frozen"?' on the next page).

5.3 What are all those names like slink, potato, etc.?

They are just "codenames". When a Debian distribution is in the development stage, it has no version number but a codename. The purpose of these codenames is to make easier the mirroring of the Debian distributions (if a real directory like `unstable` suddenly changed its name to `stable`, a lot of stuff would have to be needlessly downloaded again).

Currently, `stable` is a symbolic link to `woody` (i.e. Debian GNU/Linux 3.0) and `testing` is a symbolic link to `sarge`. This means that `woody` is the current stable distribution and `sarge` is the current testing distribution.

`unstable` is a permanent symbolic link to `sid`, as `sid` is always the unstable distribution (see 'What about "sid"?' on the facing page).

5.3.1 Which other codenames have been used in the past?

Other codenames that have been already used are: `buzz` for release 1.1, `rex` for release 1.2, `bo` for releases 1.3.x, `hamm` for release 2.0, `slink` for release 2.1 and `potato` for release 2.2.

5.3.2 Where do these codenames come from?

So far they have been characters taken from the movie "Toy Story" by Pixar.

- *buzz* (Buzz Lightyear) was the spaceman,
- *rex* was the tyrannosaurus,
- *bo* (Bo Peep) was the girl who took care of the sheep,
- *hamm* was the piggy bank,
- *slink* (Slinky Dog) was the toy dog,
- *potato* was, of course, Mr. Potato,
- *woody* was the cowboy.
- *sarge* was the sergeant of the Green Plastic Army Men.

5.4 What about “frozen”?

When the “testing” distribution is mature enough, the release manager starts ‘freezing’ it. The normal propagation delays are increased to ensure that as little as possible new bugs from “unstable” enter “testing”.

After a while, the “testing” distribution becomes truly ‘frozen’. This means that all new packages that are to propagate to the “testing” are held back, unless they include release-critical bug fixes. The “testing” distribution can also remain in such a deep freeze during the so-called ‘test cycles’, when the release is imminent.

We keep a record of bugs in the “testing” distribution that can hold off a package from being released, or bugs that can hold back the whole release. Once that bug count lowers to maximum acceptable values, the frozen “testing” distribution is declared “stable” and released with a version number.

With each new release, the previous “stable” distribution becomes obsolete and moves to the archive. For more information please see Debian archive (`http://www.debian.org/distrib/archive`).

5.5 What about “sid”?

sid or *unstable* is the place where most of the packages are initially uploaded. It will never be released directly, because packages which are to be released will first have to be included in *testing*, in order to be released in *stable* later on. sid contains packages for both released and unreleased architectures.

The name “sid” also comes from the “Toy Story” animated motion picture: Sid was the boy next door who destroyed toys :-)

5.5.1 Historical notes about “sid”

When the present-day sid did not exist, the FTP site organization had one major flaw: there was an assumption that when an architecture is created in the current unstable, it will be released when that distribution becomes the new stable. For many architectures that isn’t the case, with the result that those directories had to be moved at release time. This was impractical because the move would chew up lots of bandwidth.

The archive administrators worked around this problem for several years by placing binaries for unreleased architectures in a special directory called “sid”. For those architectures not yet released, the first time they were released there was a link from the current stable to sid, and from then on they were created inside the unstable tree as normal. This layout was somewhat confusing to users.

With the advent of package pools (see ‘What’s in the `pool` directory?’ on page 24), binary packages began to be stored in a canonical location in the pool, regardless of the distribution,

so releasing a distribution no longer causes large bandwidth consumption on the mirrors (there is, however, a lot of gradual bandwidth consumption throughout the development process).

5.6 What does the stable directory contain?

- stable/main/: This directory contains the packages which formally constitute the most recent release of the Debian GNU/Linux system.

  These packages all comply with the Debian Free Software Guidelines (http://www.debian.org/social_contract#guidelines), and are all freely usable and distributable.

- stable/non-free/: This directory contains packages distribution of which is restricted in a way that requires that distributors take careful account of the specified copyright requirements.

  For example, some packages have licenses which prohibit commercial distribution. Others can be redistributed but are in fact shareware and not freeware. The licenses of each of these packages must be studied, and possibly negotiated, before the packages are included in any redistribution (e.g., in a CD-ROM).

- stable/contrib/: This directory contains packages which are DFSG-free and *freely distributable* themselves, but somehow depend on a package that is *not* freely distributable and thus available only in the non-free section.

5.7 What does the testing directory contain?

Packages are installed into the 'testing' directory after they have undergone some degree of testing in unstable. They must be in sync on all architectures where they have been built and mustn't have dependencies that make them uninstallable; they also have to have fewer release-critical bugs than the versions currently in testing. This way, we hope that 'testing' is always close to being a release candidate.

More information about the status of "testing" in general and the individual packages is available at http://www.debian.org/devel/testing

5.8 What does the unstable directory contain?

The 'unstable' directory contains a snapshot of the current development system. Users are welcome to use and test these packages, but are warned about their state of readiness. The advantage of using the unstable distribution is that you are always up-to-date with the latest in GNU/Linux software industry, but if it breaks: you get to keep both parts :-)

There are also main, contrib and non-free subdirectories in 'unstable', separated on the same criteria as in 'stable'.

5.9 What are all those directories inside `dists/stable/main`?

Within each of the major directory trees (`dists/stable/main`, `dists/stable/contrib`, `dists/stable/non-free`, and `dists/unstable/main/`, etc.), the binary packages reside in subdirectories whose names indicate the chip architecture for which they were compiled:

- binary-all/, for packages which are architecture-independent. These include, for example, Perl scripts, or pure documentation.
- binary-i386/, for packages which execute on 80x86 PC machines.
- binary-m68k/, for packages which execute on machines based on one of the Motorola 680x0 processors. Currently this is done mainly for Atari and Amiga computers, and also for some VME based industry standard boards.
- binary-sparc/, for packages which execute on Sun SPARCStations.
- binary-alpha/, for packages which execute on DEC Alpha machines.
- binary-powerpc/, for packages which execute on PowerPC machines.
- binary-arm/, for packages which execute on ARM machines.

Please note that the actual binary packages for *woody* and subsequent releases no longer reside in these directories, but in the top level `pool` directory. The index files (Packages and Packages.gz) have been kept, though, for backwards compatibility.

See 'On what hardware architectures/systems does Debian GNU/Linux run?' on page 9 for more information.

5.10 Where is the source code?

Source code is included for everything in the Debian system. Moreover, the license terms of most programs in the system *require* that source code be distributed along with the programs, or that an offer to provide the source code accompany the programs.

Normally the source code is distributed in the "source" directories, which are parallel to all the architecture-specific binary directories, or more recently in the `pool` directory (see 'What's in the `pool` directory?' on the next page). To retrieve the source code without having to be familiar with the structure of the FTP archive, try a command like `apt-get source mypackagename`.

Some packages are only distributed as source code due to the restrictions in their licenses. Notably, one such package is `pine`, see 'Where is pine?' on page 18 for more information.

Source code may or may not be available for packages in the "contrib" and "non-free" directories, which are not formally part of the Debian system.

5.11 What's in the `pool` directory?

Historically, packages were kept in the subdirectory of `dists` corresponding to which distribution contained them. This turned out to cause various problems, such as large bandwidth consumption on mirrors when major changes were made.

Packages are now kept in a large 'pool', structured according to the name of the source package. To make this manageable, the pool is subdivided by section ('main', 'contrib' and 'non-free') and by the first letter of the source package name. These directories contain several files: the binary packages for each architecture, and the source packages from which the binary packages were generated.

You can find out where each package is placed by executing a command like `apt-cache showsrc mypackagename` and looking at the 'Directory:' line. For example, the `apache` packages are stored in `pool/main/a/apache/`. Since there are so many `lib*` packages, these are treated specially: for instance, libpaper packages are stored in `pool/main/libp/libpaper/`.

The `dists` directories are still used for the index files used by programs like `apt`. Also, at the time of writing, older distributions have not been converted to use pools so you'll see paths containing distributions such as potato or woody in the Filename header field.

Normally, you won't have to worry about any of this, as `apt` and probably `dpkg-ftp` (see 'How can I keep my Debian system current?' on page 42) will handle it seamlessly.

5.12 What is "incoming"?

After a developer uploads a package, it stays for a short while in the "incoming" directory before it is checked that it's genuine and allowed into the archive.

Usually nobody should install things from this place. However, in some rare cases of emergency, the incoming directory is available at `http://incoming.debian.org/`. You can manually fetch packages, check the GPG signature and MD5sums in the .changes and .dsc files, and then install them.

Chapter 6

Basics of the Debian package management system

6.1 What is a Debian package?

Packages generally contain all of the files necessary to implement a set of related commands or features. There are two types of Debian packages:

- *Binary packages*, which contain executables, configuration files, man/info pages, copyright information, and other documentation. These packages are distributed in a Debian-specific archive format (see 'What is the format of a Debian binary package?' on the following page); they are usually distinguished by having a '.deb' file extension. Binary packages can be unpacked using the Debian utility `dpkg`; details are given in its manual page.
- *Source packages*, which consist of a `.dsc` file describing the source package (including the names of the following files), a `.orig.tar.gz` file that contains the original unmodified source in gzip-compressed tar format and usually a `.diff.gz` file that contains the Debian-specific changes to the original source. The utility `dpkg-source` packs and unpacks Debian source archives; details are provided in its manual page.

Installation of software by the package system uses "dependencies" which are carefully designed by the package maintainers. These dependencies are documented in the `control` file associated with each package. For example, the package containing the GNU C compiler (`gcc`) "depends" on the package `binutils` which includes the linker and assembler. If a user attempts to install `gcc` without having first installed `binutils`, the package management system (dpkg) will send an error message that it also needs `binutils`, and stop installing `gcc`. (However, this facility can be overridden by the insistent user, see `dpkg(8)`.) See more in 'What is meant by saying that a package *Depends*, *Recommends*, *Suggests*, *Conflicts*, *Replaces* or *Provides* another package?' on page 30 below.

Debian's packaging tools can be used to:

- manipulate and manage packages or parts of packages,
- aid the user in the break-up of packages that must be transmitted through a limited-size medium such as floppy disks,
- aid developers in the construction of package archives, and
- aid users in the installation of packages which reside on a remote FTP site.

6.2 What is the format of a Debian binary package?

A Debian "package", or a Debian archive file, contains the executable files, libraries, and documentation associated with a particular suite of program or set of related programs. Normally, a Debian archive file has a filename that ends in `.deb`.

The internals of this Debian binary packages format are described in the `deb(5)` manual page. This internal format is subject to change (between major releases of Debian GNU/Linux), therefore please always use `dpkg-deb(8)` for manipulating `.deb` files.

6.3 Why are Debian package file names so long?

The Debian binary package file names conform to the following convention: <foo>_<VersionNumber>-<DebianRevisionNumber>.deb

Note that `foo` is supposed to be the package name. As a check, one can learn the package name associated with a particular Debian archive file (.deb file) in one of these ways:

- inspect the "Packages" file in the directory where it was stored at a Debian FTP archive site. This file contains a stanza describing each package; the first field in each stanza is the formal package name.
- use the command `dpkg --info foo_VVV-RRR.deb` (where VVV and RRR are the version and revision of the package in question, respectively). This displays, among other things, the package name corresponding to the archive file being unpacked.

The `VVV` component is the version number specified by the upstream developer. There are no standards in place here, so the version number may have formats as different as "19990513" and "1.3.8pre1".

The `RRR` component is the Debian revision number, and is specified by the Debian developer (or an individual user if he chooses to build the package himself). This number corresponds to the revision level of the Debian package, thus, a new revision level usually signifies changes in the Debian Makefile (`debian/rules`), the Debian control file (`debian/control`), the installation or removal scripts (`debian/p*`), or in the configuration files used with the package.

6.4 What is a Debian control file?

Specifics regarding the contents of a Debian control file are provided in the Debian Packaging manual, chapter 4, see 'What other documentation exists on and for a Debian system?' on page 55.

Briefly, a sample control file is shown below for the Debian package hello:

```
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)
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 Package field gives the package name. This is the name by which the package can be manipulated by the package tools, and usually similar to but not necessarily the same as the first component string in the Debian archive file name.

The Version field gives both the upstream developer's version number and (in the last component) the revision level of the Debian package of this program as explained in 'Why are Debian package file names so long?' on the preceding page.

The Architecture field specifies the chip for which this particular binary was compiled.

The Depends field gives a list of packages that have to be installed in order to install this package successfully.

The Installed-Size indicates how much disk space the installed package will consume. This is intended to be used by installation front-ends in order to show whether there is enough disk space available to install the program.

The Section line gives the "section" where this Debian package is stored at the Debian FTP sites. This is the name of a subdirectory (within one of the main directories, see 'What are all those directories at the Debian FTP archives?' on page 19) where the package is stored.

The Priority indicates how important is this package for installation, so that semi-intelligent software like dselect or console-apt can sort the package into a category of e.g. packages optionally installed. See 'What is a *Required*, *Important*, *Standard*, *Optional*, or *Extra* package?' on page 29.

The Maintainer field gives the e-mail address of the person who is currently responsible for maintaining this package.

The Description field gives a brief summary of the package's features.

For more information about all possible fields a package can have, please see the Debian Packaging Manual, section 4., "Control files and their fields".

6.5 What is a Debian conffile?

Conffiles is a list of configuration files (usually placed in `/etc`) that the package management system will not overwrite when the package is upgraded. This ensures that local values for the contents of these files will be preserved, and is a critical feature enabling the in-place upgrade of packages on a running system.

To determine exactly which files are preserved during an upgrade, run:

```
dpkg --status package
```

And look under "Conffiles:".

6.6 What is a Debian preinst, postinst, prerm, and postrm script?

These files are executable scripts which are automatically run before or after a package is installed. Along with a file named `control`, all of these files are part of the "control" section of a Debian archive file.

The individual files are:

preinst This script executes before that package will be unpacked from its Debian archive (".deb") file. Many 'preinst' scripts stop services for packages which are being upgraded until their installation or upgrade is completed (following the successful execution of the 'postinst' script).

postinst This script typically completes any required configuration of the package `foo` once `foo` has been unpacked from its Debian archive (".deb") file. Often, 'postinst' scripts ask the user for input, and/or warn the user that if he accepts default values, he should remember to go back and re-configure that package as the situation warrants. Many 'postinst' scripts then execute any commands necessary to start or restart a service once a new package has been installed or upgraded.

prerm This script typically stops any daemons which are associated with a package. It is executed before the removal of files associated with the package.

postrm This script typically modifies links or other files associated with `foo`, and/or removes files created by the package. (Also see 'What is a Virtual Package?' on the facing page.)

Currently all of the control files can be found in directory `/var/lib/dpkg/info`. The files relevant to package `foo` begin with the name "foo" and have file extensions of "preinst", "postinst", etc., as appropriate. The file `foo.list` in that directory lists all of the files that were installed with the package `foo`. (Note that the location of these files is a dpkg internal; you should not rely on it.)

6.7 What is a *Required*, *Important*, *Standard*, *Optional*, or *Extra* package?

Each Debian package is assigned a *priority* by the distribution maintainers, as an aid to the package management system. The priorities are:

- **Required**: packages that are necessary for the proper functioning of the system.

  This includes all tools that are necessary to repair system defects. You must not remove these packages or your system may become totally broken and you may probably 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** packages should be found on any Unix-like system.

  Other packages which the system will not run well or be usable without will be here. This does *NOT* include Emacs or X11 or TeX or any other large applications. These packages only constitute the bare infrastructure.

- **Standard** packages are standard on any Linux system, including a reasonably small but not too limited character-mode system.

  This is what will install by default if users do not select anything else. It does not 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 turns out to be possible without X).

- **Optional** packages include all those that you might reasonably want to install if you did not know what it was, or do not have specialized requirements.

  This includes X11, a full TeX distribution, and lots of applications.

- **Extra**: packages that either conflict with others with higher priorities, are only likely to be useful if you already know what they are, or have specialized requirements that make them unsuitable for "Optional".

6.8 What is a Virtual Package?

A virtual package is a generic name that applies to any one of a group of packages, all of which provide similar basic functionality. For example, both the `tin` and `trn` programs are

news readers, and should therefore satisfy any dependency of a program that required a news reader on a system, in order to work or to be useful. They are therefore both said to provide the "virtual package" called `news-reader`.

Similarly, `smail` and `sendmail` both provide the functionality of a mail transport agent. They are therefore said to provide the virtual package, "mail transport agent". If either one is installed, then any program depending on the installation of a `mail-transport-agent` will be satisfied by the existence of this virtual package.

Debian provides a mechanism so that, if more than one package which provide the same virtual package is installed on a system, then system administrators can set one as the preferred package. The relevant command is `update-alternatives`, and is described further in 'Some users like mawk, others like gawk; some like vim, others like elvis; some like trn, others like tin; how does Debian support diversity?' on page 53.

6.9 What is meant by saying that a package *Depends*, *Recommends*, *Suggests*, *Conflicts*, *Replaces* or *Provides* another package?

The Debian package system has a range of package "dependencies" which are designed to indicate (in a single flag) the level at which Program A can operate independently of the existence of Program B on a given system:

- Package A *depends* on Package B if B absolutely must be installed in order to run A. In some cases, A depends not only on B, but on a version of B. In this case, the version dependency is usually a lower limit, in the sense that A depends on any version of B more recent than some specified version.
- Package A *recommends* Package B, if the package maintainer judges that most users would not want A without also having the functionality provided by B.
- Package A *suggests* Package B if B contains files that are related to (and usually enhance) the functionality of A.
- Package A *conflicts* with Package B when A will not operate if B is installed on the system. Most often, conflicts are cases where A contains files which are an improvement over those in B. "Conflicts" are often combined with "replaces".
- Package A *replaces* Package B when files installed by B are removed and (in some cases) over-written by files in A.
- Package A *provides* Package B when all of the files and functionality of B are incorporated into A. This mechanism provides a way for users with constrained disk space to get only that part of package A which they really need.

More detailed information on the use of each these terms can be found in the Packaging manual and the Policy manual.

6.10 What is meant by Pre-Depends?

"Pre-Depends" is a special dependency. In the case of most packages, `dpkg` will unpack its archive file (i.e., its `.deb` file) independently of whether or not the files on which it depends exist on the system. Simplistically, unpacking means that `dpkg` will extract the files from the archive file that were meant to be installed on your file system, and put them in place. If those packages *depend* on the existence of some other packages on your system, `dpkg` will refuse to complete the installation (by executing its "configure" action) until the other packages are installed.

However, for some packages, `dpkg` will refuse even to unpack them until certain dependencies are resolved. Such packages are said to "Pre-depend" on the presence of some other packages. The Debian project provided this mechanism to support the safe upgrading of systems from `a.out` format to `ELF` format, where the *order* in which packages were unpacked was critical. There are other large upgrade situations where this method is useful, e.g. the packages with the required priority and their LibC dependency.

As before, more detailed information about this can be found in the Packaging manual.

6.11 What is meant by *unknown*, *install*, *remove purge* and *hold* in the package status?

These "want" flags tell what the user wanted to do with a package (as indicated either by the user's actions in the "Select" section of `dselect`, or by the user's direct invocations of `dpkg`).

Their meanings are:

- unknown - the user has never indicated whether he wants the package
- install - the user wants the package installed or upgraded
- remove - the user wants the package removed, but does not want to remove any existing configuration files.
- purge - the user wants the package to be removed completely, including its configuration files.
- hold - the user wants this package not to be processed, i.e., he wants to keep the current version with the current status whatever that is.

6.12 How do I put a package on hold?

There are two ways of holding back packages, with dpkg, or with dselect.

With dpkg, you just have to export the list of package selections, with:

```
dpkg --get-selections \* > selections.txt
```

Then edit the resulting file `selections.txt`, change the line containing the package you wish to hold, e.g. `libc6`, from this:

```
libc6                                           install
```

to this:

```
libc6                                           hold
```

Save the file, and reload it into dpkg database with:

```
dpkg --set-selections < selections.txt
```

With dselect, you just have to enter the [S]elect screen, find the package you wish to hold in its present state, and press the '=' key (or 'H'). The changes will go live immediately after you exit the [S]elect screen.

6.13 How do I install a source package?

Debian source packages can't actually be "installed", they are just unpacked in whatever directory you want to build the binary packages they produce.

Source packages are distributed on most of the same mirrors where you can obtain the binary packages. If you set up your APT's `sources.list(5)` to include the appropriate "deb-src" lines, you'll be able to easily download any source packages by running

```
apt-get source foo
```

To help you in actually building the source package, Debian source package provide the so-called build-dependencies mechanism. This means that the source package maintainer keeps a list of other packages that are required to build their package. To see how this is useful, run

```
apt-get build-dep foo
```

before building the source.

6.14 How do I build binary packages from a source package?

You will need all of foo_*.dsc, foo_*.tar.gz and foo_*.diff.gz to compile the source (note: there is no .diff.gz for some packages that are native to Debian).

Once you have them ('How do I install a source package?' on the facing page), if you have the `dpkg-dev` package installed, the following command:

```
dpkg-source -x foo_version-revision.dsc
```

will extract the package into a directory called `foo-version`.

If you want just to compile the package, you may cd into `foo-version` directory and issue the command

```
dpkg-buildpackage -rfakeroot -b
```

to build the package (note that this also requires the `fakeroot` package), and then

```
dpkg -i ../foo_version-revision_arch.deb
```

to install the newly-built package(s).

6.15 How do I create Debian packages myself?

For more detailed description on this, read the New Maintainers' Guide, available in the `maint-guide` package, or at http://www.debian.org/doc/devel-manuals#maint-guide.

Chapter 7

The Debian package management tools

7.1 What programs does Debian provide for managing its packages?

7.1.1 dpkg

This is the main package management program. `dpkg` can be invoked with many options. Some common uses are:

- Find out all the options: `dpkg --help`.
- Print out the control file (and other information) for a specified package: `dpkg --info foo_VVV-RRR.deb`
- Install a package (including unpacking and configuring) onto the file system of the hard disk: `dpkg --install foo_VVV-RRR.deb`.
- Unpack (but do not configure) a Debian archive into the file system of the hard disk: `dpkg --unpack foo_VVV-RRR.deb`. Note that this operation does *not* necessarily leave the package in a usable state; some files may need further customization to run properly. This command removes any already-installed version of the program and runs the preinst (see 'What is a Debian preinst, postinst, prerm, and postrm script?' on page 28) script associated with the package.
- Configure a package that already has been unpacked: `dpkg --configure foo`. Among other things, this action runs the postinst (see 'What is a Debian preinst, postinst, prerm, and postrm script?' on page 28) script associated with the package. It also updates the files listed in the `conffiles` for this package. Notice that the 'configure' operation takes as its argument a package name (e.g., foo), *not* the name of a Debian archive file (e.g., foo_VVV-RRR.deb).
- Extract a single file named "blurf" (or a group of files named "blurf*" from a Debian archive: `dpkg --fsys-tarfile foo_VVV-RRR.deb | tar -xf - blurf*`
- Remove a package (but not its configuration files): `dpkg --remove foo`.

- Remove a package (including its configuration files): `dpkg --purge foo`.
- List the installation status of packages containing the string (or regular expression) "foo*": `dpkg --list 'foo*'`.

7.1.2 dselect

This program is a menu-driven interface to the Debian package management system. It is particularly useful for first-time installations and large-scale upgrades.

`dselect` can:

- guide the user as he/she chooses among packages to install or remove, ensuring that no packages are installed that conflict with one another, and that all packages required to make each package work properly are installed;
- warn the user about inconsistencies or incompatibilities in their selections;
- determine the order in which the packages must be installed;
- automatically perform the installation or removal; and
- guide the user through whatever configuration process are required for each package.

`dselect` begins by presenting the user with a menu of 7 items, each of which is a specific action. The user can select one of the actions by using the arrow keys to move the highlighter bar, then pressing the <*enter*> key to select the highlighted action.

What the user sees next depends on the action he selected. If he selects any option but `Access` or `Select`, then `dselect` will simply proceed to execute the specified action: e.g., if the user selected the action `Remove`, then dselect would proceed to remove all of the files selected for removal when the user last chose the `Select` action.

Both the `Access` menu item and the `Select` menu item lead to additional menus. In both cases, the menus are presented as split screens; the top screen gives a scrollable list of choices, while the bottom screen gives a brief explanation ("info") for each choice.

Extensive on-line help is available, use the '?' key to get to a help screen at any time.

The order in which the actions are presented in the first `dselect` menu represents the order in which a user would normally choose `dselect` to install packages. However, a user can pick any of the main menu choices as often as needed (including not at all, depending on what one wants to do).

- Begin by choosing an **Access Method**. This is the method by which the user plans on accessing Debian packages; e.g., some users have Debian packages available on CD-ROM, while others plan to fetch them using anonymous FTP. The selected "Access Method" is stored after `dselect` exits, so if it does not change, then this option need not be invoked again.

- Then **Update** the list of available packages. To do this, `dselect` reads the file "Packages.gz" which should be included in the top level of the directory where the Debian packages to be installed are stored. (But if it is not there, `dselect` will offer to make it for you.)

- **Select** specific packages for installation on his system. After choosing this menu item, the user is first presented with a full screen of help (unless the '–expert' command line option was used). Once the user exits the Help screen, he sees the split-screen menu for choosing packages to install (or remove).

  The top part of the screen is a relatively narrow window into the list of Debian's 8250 packages; the bottom part of the screen contains description of the package or group of packages which are highlighted above.

  One can specify which packages should be operated on by highlighting a package name or the label for a group of packages. After that, you can select packages:

  to be installed: This is accomplished by pressing the '+' key.

  to be deleted: Packages can be deleted two ways:

  – removed: this removes most of the files associated with the package, but preserves the files listed as configuration files (see 'What is a Debian conffile?' on page 28) and package configuration information. This is done by pressing the '-' key.
  – purged: this removes *every* file that is part of the package. This is done by pressing the '_' key.

  Note that it's not possible to remove "All Packages". If you try that, your system will instead be reduced to the initial installed base packages.

  to be put "on hold" This is done by pressing '=', and it effectively tells `dselect` not to upgrade a package even if the version currently installed on your system is not as recent as the version that is available in the Debian repository you are using (this was specified when you set the **Access Method**, and acquired when you used **Update**).

  Just like you can put a package on hold, you can reverse such setting by pressing ':'. That tells `dselect` that the package(s) may be upgraded if a newer version is available. This is the default setting.

  You can select a different order in which the packages are presented, by using the 'o' key to cycle between various options for sorting the packages. The default order is to present packages by Priority; within each priority, packages are presented in order of the directory (a.k.a. section) of the archive in which they are stored. Given this sort order, some packages in section A (say) may be presented first, followed by some packages in section B, followed by more packages (of lower priority) in section A.

  You can also expand meanings of the labels at the top of the screen, by using the 'v' (verbose) key. This action pushes much of the text that formerly fit onto the display off to the right. To see it, press the right arrow; to scroll back to the left, press the left arrow.

If you select a package for installation or removal, e.g., `foo.deb`, and that package depends on (or recommends) another package, e.g., `blurf.deb`, then `dselect` will place the you in a sub-screen of the main selection screen. There you can choose among the related packages, accepting the suggested actions (to install or not), or rejecting them. To do the latter, press Shift-D; to return to the former, press Shift-U. In any case, you can save your selections and return to the main selection screen by pressing Shift-Q.

- Users returning to the main menu can then select the "Install" menu item to unpack and configure the selected packages. Alternatively, users wishing to remove files can choose the "Remove" menu item. At any point, users can choose "Quit" to exit dselect; users' selections are preserved by `dselect`.

7.1.3 dpkg-deb

This program manipulates Debian archive(`.deb`) files. Some common uses are:

- Find out all the options: `dpkg-deb --help`.
- Determine what files are contained in a Debian archive file: `dpkg-deb --contents foo_VVV-RRR.deb`)
- Extract the files contained in a named Debian archive into a user specified directory: `dpkg-deb --extract foo_VVV-RRR.deb tmp` extracts each of the files in `foo_VVV-RRR.deb` into the directory `tmp/`. This is convenient for examining the contents of a package in a localized directory, without installing the package into the root file system.

Note that any packages that were merely unpacked using `dpkg-deb --extract` will be incorrectly installed, you should use `dpkg --install` instead.

More information is given in the manual page `dpkg-deb(1)`.

7.1.4 apt-get

`apt-get` provides a simple way to install packages from the command line. Unlike `dpkg`, `apt-get` does not understand .deb files, it works with the packages proper name and can only install .deb archives from a source specified in `/etc/apt/sources.list`.

For more information, install `apt` package and read `apt-get(8)`, `sources.list(5)` and `/usr/share/doc/apt/guide.html/index.html`.

7.1.5 dpkg-split

This program splits large package into smaller files (e.g., for writing onto a set of floppy disks), and can also be used to merge a set of split files back into a single file. It can only be used

on a Debian system (i.e. a system containing the `dpkg` package), since it calls the program `dpkg-deb` to parse the debian package file into its component records.

For example, to split a big .deb file into N parts,

- Execute the command `dpkg-split --split foo.deb`. This will produce N files each of approximately 460 KBytes long in the current directory.
- Copy those N files to floppy disks.
- Copy the contents of the floppy disks onto the hard disk of your choice on the other machine.
- Join those part-files together using `dpkg-split --join "foo*"`.

7.2 Debian claims to be able to update a running program; how is this accomplished?

The kernel (file system) in Debian GNU/Linux systems supports replacing files even while they're being used.

We also provide a program called `start-stop-daemon` which is used to start daemons at boot time or to stop daemons when the kernel runlevel is changed (e.g., from multi-user to single-user or to halt). The same program is used by installation scripts when a new package containing a daemon is installed, to stop running daemons, and restart them as necessary.

7.3 How can I tell what packages are already installed on a Debian system?

To learn the status of all the packages installed on a Debian system, execute the command

```
dpkg --list
```

This prints out a one-line summary for each package, giving a 2-letter status symbol (explained in the header), the package name, the version which is *installed*, and a brief description.

To learn the status of packages whose names match the string any pattern beginning with "foo" by executing the command:

```
dpkg --list 'foo*'
```

To get a more verbose report for a particular package, execute the command:

```
dpkg --status packagename
```

7.4 How can I find out what package produced a particular file?

To identify the package that produced the file named `foo` execute either:

- `dpkg --search filename`

  This searches for `filename` in installed packages. (This is (currently) equivalent to searching all of the files having the file extension of `.list` in the directory `/var/lib/dpkg/info/`, and adjusting the output to print the names of all the packages containing it, and diversions.)

- `zgrep foo Contents-ARCH.gz`

  This searches for files which contain the substring `foo` in their full path names. The files `Contents-ARCH.gz` (where ARCH represents the wanted architecture) reside in the major package directories (main, non-free, contrib) at a Debian FTP site. A `Contents` file refers only to the packages in the subdirectory tree where it resides. Therefore, a user might have to search more than one `Contents` files to find the package containing the file `foo`.

  This method has the advantage over `dpkg --search` in that it will find files in packages that are not currently installed on your system.

Chapter 8

Keeping your Debian system up-to-date

A Debian goal is to provide a consistent upgrade path and a secure upgrade process. We always do our best to make upgrading to new releases a smooth procedure. In case there's some important note to add to the upgrade process, the packages will alert the user, and often provide a solution to a possible problem.

You should also read the Release Notes, document that describes the details of specific upgrades, shipped on all Debian CDs, and available on the WWW at http://www.debian.org/releases/stable/releasenotes.

8.1 How can I upgrade my Debian 1.3.1 (or earlier) distribution, based on libc5, to 2.0 (or later), based on libc6?

There are several ways to upgrade:

- Using a simple shell script called `autoup.sh` which upgrades the most important packages. After `autoup.sh` has done his job, you may use dselect to install the remaining packages *en masse*. This is probably the recommended method, but not the only one.

  Currently, the latest release of `autoup.sh` may be found on the following locations:

  - http://www.debian.org/releases/2.0/autoup/
  - http://www.taz.net.au/autoup/
  - http://debian.vicnet.net.au/autoup/

- Following closely the Debian libc5 to libc6 Mini-HOWTO (http://debian.vicnet.net.au/autoup/HOWTO/libc5-libc6-Mini-HOWTO.html) and upgrade the most important packages by hand. `autoup.sh` is based on this Mini-HOWTO, so this method should work more or less like using `autoup.sh`.

- Using a libc5-based `apt`. APT stands for A Package Tool, and it might replace dselect some day. Currently, it works just as a command-line interface, or as a dselect access method. You will find a libc5 version in the `dists/slink/main/upgrade-older-i386` directory at the Debian archives.
- Using just dselect, without upgrading any package by hand first. It is highly recommended that you do NOT use this method if you can avoid it, because dselect alone currently does not install packages in the optimal order. APT works much better and it is safer.

8.2 How can I keep my Debian system current?

One could simply execute an anonymous ftp call to a Debian archive, then peruse the directories until he finds the desired file, and then fetch it, and finally install it using `dpkg`. Note that `dpkg` will install upgrade files in place, even on a running system. Sometimes, a revised package will require the installation of a newly revised version of another package, in which case the installation will fail until/unless the other package is installed.

Many people find this approach much too time-consuming, since Debian evolves so quickly – typically, a dozen or more new packages are uploaded every week. This number is larger just before a new major release. To deal with this avalanche, many people prefer to use a more automated method. Several different packages are available for this purpose:

8.2.1 APT

APT is an advanced interface to the Debian packaging system. apt-get is the command-line tool for handling packages, and APT dselect method is an interface to APT through `dselect`. Both of these provide a simpler, safer way to install and upgrade packages.

APT features complete installation ordering, multiple source capability and several other unique features, see the User's Guide in `/usr/share/doc/apt/guide.html/index.html`.

Install the `apt` package, and edit the `/etc/apt/sources.list` file to set it up. If you wish to upgrade to the latest stable version of Debian, you'll probably want to use a source like this one:

```
http://http.us.debian.org/debian stable main contrib non-free
```

You can replace http.us.debian.org with the name of a faster Debian mirror near you. See the mirror list at `http://www.debian.org/misc/README.mirrors` for more information.

Details on this can be found in `apt-get(8)` and `sources.list(8)` manual pages, as well as in the aforementioned APT User's Guide, in `/usr/share/doc/apt/guide.html/index.html`.

Then run

```
apt-get update
```

followed by

```
apt-get dist-upgrade
```

Answer any questions that might come up, and your system will be upgraded.

To use APT with `dselect`, choose the APT access method in dselect's method selection screen (option 0) and then specify the sources that should be use. The configuration file is `/etc/apt/sources.list`, and its format is described in the `sources.list(5)` manual page.

If you want to use CDs to install packages, you can use `apt-cdrom`. For details, please see the Release Notes, section "Setting up for an upgrade from a local mirror".

Please note that when you get and install the packages, you'll still have them kept in your /var directory hierarchy. To keep your partition from overflowing, remember to delete extra files using `apt-get clean` and `apt-get autoclean`, or to move them someplace else (hint: use `apt-move`).

8.2.2 dpkg-ftp

This is an older access method for `dselect`. It can be invoked from within `dselect`, thereby allowing a user the ability to download files and install them directly in one step. To do this, select the `ftp` access method in `dselect` (option 0) and specify the remote host name and directory. `dpkg-ftp` will then automatically download the files that are selected (either in this session of `dselect` or earlier ones).

Note that, unlike the `mirror` program, `dpkg-ftp` does not grab everything at a mirror site. Rather, it downloads only those files which you have selected (when first starting up `dpkg-ftp`), and which need to be updated.

`dpkg-ftp` is somewhat obsolete. You should use the APT access method with ftp:// URLs in `sources.list` instead.

8.2.3 mirror

This Perl script, and its (optional) manager program called `mirror-master`, can be used to fetch user-specified parts of a directory tree from a specified host *via* anonymous FTP.

`mirror` is particularly useful for downloading large volumes of software. After the first time files have been downloaded from a site, a file called `.mirrorinfo` is stored on the local host. Changes to the remote file system are tracked automatically by `mirror`, which compares this file to a similar file on the remote system and downloads only changed files.

The `mirror` program is generally useful for updating local copies of remote directory trees. The files fetched need not be Debian files. (Since `mirror` is a Perl script, it can also run on

non-Unix systems.) Though the `mirror` program provides mechanisms for excluding files names of which match user-specified strings, this program is most useful when the objective is to download whole directory trees, rather than selected packages.

8.2.4 dpkg-mountable

dpkg-mountable adds an access method called 'mountable' to dselect's list, which allows you to install from any file system specified in /etc/fstab. For example, the archive could be a normal hard disk partition or an NFS server, which it will automatically mount and umount for you if necessary.

It also has some extra features not found in the standard dselect methods, such as provision for a local file tree (either parallel to the main distribution or totally separate), and only getting packages which are required, rather than the time-consuming recursive directory scan, as well as logging of all dpkg actions in the install method.

8.3 Must I go into single user mode in order to upgrade a package?

No. Packages can be upgraded in place, even in running systems. Debian has a `start-stop-daemon` program that is invoked to stop, then restart running process if necessary during a package upgrade.

8.4 Do I have to keep all those .deb archive files on my disk?

No. If you have downloaded the files to your disk (which is not absolutely necessary, see above for the description of dpkg-ftp), then after you have installed the packages, you can remove them from your system.

8.5 How can I keep a log of the packages I added to the system?

`dpkg` keeps a record of the packages that have been unpacked, configured, removed, and/or purged, but does not (currently) keep a log of terminal activity that occurred while a package was being so manipulated.

The simplest way to work around this is to run your `dpkg/dselect/apt-get`/whatever sessions within the `script(1)` program.

Chapter 9

Debian and the kernel

9.1 Can I install and compile a kernel without some Debian-specific tweaking?

Yes.

There's only one common catch: the Debian C libraries are built with the most recent *stable* releases of the **kernel** headers. If you happen to need to compile a program with kernel headers newer than the ones from the stable branch, then you should either upgrade the package containing the headers (`libc6-dev`), or use the new headers from an unpacked tree of the newer kernel. That is, if the kernel sources are in `/usr/src/linux`, then you should add `-I/usr/src/linux/include/` to your command line when compiling.

9.2 What tools does Debian provide to build custom kernels?

Users who wish to (or must) build a custom kernel are encouraged to download the package `kernel-package`. This package contains the script to build the kernel package, and provides the capability to create a Debian kernel-image package just by running the command

```
make-kpkg kernel_image
```

in the top-level kernel source directory. Help is available by executing the command

```
make-kpkg --help
```

and through the manual page `make-kpkg(8)`.

Users must separately download the source code for the most recent kernel (or the kernel of their choice) from their favorite Linux archive site, unless a kernel-source-version package is available (where "version" stands for the kernel version).

Detailed instructions for using the `kernel-package` package are given in the file `/usr/doc/kernel-package/README`. Briefly, one should:

- Unpack the kernel sources, and `cd` to the newly created directory.
- Modify the kernel configuration using one of these commands:
  - `make config` (for a tty one-line-at-a-time-interface).
  - `make menuconfig` (for an ncurses-based menu driven interface). Note that to use this option, the `libncurses5-dev` package must be installed.
  - `make xconfig` (for an X11 interface). Using this option requires that relevant X and Tcl/Tk packages be installed.

  Any of the above steps generates a new `.config` in the top-level kernel source directory.
- Execute the command: `make-kpkg -rev Custom.N kernel_image`, where N is a revision number assigned by the user. The new Debian archive thus formed would have revision Custom.1, e.g., `kernel-image-2.2.14_Custom.1_i386.deb` for the Linux kernel 2.2.14.
- Install the package created.
  - Run `dpkg --install /usr/src/kernel-image-VVV_Custom.N.deb` to install the kernel itself. The installation script will:
    * run the boot loader, LILO (if it is installed),
    * install the custom kernel in /boot/vmlinuz_VVV-Custom.N, and set up appropriate symbolic links to the most recent kernel version.
    * prompt the user to make a boot floppy. This boot floppy will contain the raw kernel only. See 'How can I make a custom boot floppy?' on this page.
  - To employ secondary boot loaders such as `grub` or `loadlin`, copy this image to other locations (e.g., to /boot/grub or to an `MS-DOS` partition).

9.3 How can I make a custom boot floppy?

This task is greatly aided by the Debian package `boot-floppies`, normally found in the `admin` section of the Debian FTP archive. Shell scripts in this package produce boot floppies in the `SYSLINUX` format. These are `MS-DOS` formatted floppies whose master boot records have been altered so that they boot Linux directly (or whatever other operating system has been defined in the syslinux.cfg file on the floppy). Other scripts in this package produce emergency root disks and can even reproduce the base disks.

You will find more information about this in the `/usr/doc/boot-floppies/README` file after installing the `boot-floppies` package.

9.4 What special provisions does Debian provide to deal with modules?

Debian's `modconf` package provides a shell script (`/usr/sbin/modconf`) which can be used to customize the configuration of modules. This script presents a menu-based interface, prompting the user for particulars on the loadable device drivers in his system. The responses are used to customize the file `/etc/modules.conf` (which lists aliases, and other arguments that must be used in conjunction with various modules) through files in `/etc/modutils/`, and `/etc/modules` (which lists the modules that must be loaded at boot time).

Like the (new) Configure.help files that are now available to support the construction of custom kernels, the modconf package comes with a series of help files (in `/usr/lib/modules_help/`) which provide detailed information on appropriate arguments for each of the modules.

9.5 Can I safely de-install an old kernel package, and if so, how?

Yes. The `kernel-image-NNN.prerm` script checks to see whether the kernel you are currently running is the same as the kernel you are trying to de-install. Therefore you can remove unwanted kernel image packages using this command:

```
dpkg --purge --force-remove-essential kernel-image-NNN
```

(replace "NNN" with your kernel version and revision number, of course)

Chapter 10

Customizing your installation of Debian GNU/Linux

10.1 How can I ensure that all programs use the same paper size?

Install the `libpaperg` package, and it will ask you for a system-wide default paper size. This setting will be kept in the file `/etc/papersize`.

Users can override the paper size setting using the `PAPERSIZE` environment variable. For details, see the manual page `papersize(5)`.

10.2 How can I provide access to hardware peripherals, without compromising security?

Many device files in the `/dev` directory belong to some predefined groups. For example, `/dev/fd0` belongs to the `floppy` group, and `/dev/dsp` belongs to the `audio` group.

If you want a certain user to have access to one of these devices, just add the user to the group the device belongs to, i.e. do:

```
adduser user group
```

This way you won't have to change the file permissions on the device.

10.3 How do I load a console font on startup the Debian way?

The `kbd` and `console-tools` packages support this, edit `/etc/kbd/config` or `/etc/console-tools/config` files.

10.4 How can I configure an X11 program's application defaults?

Debian's X programs will install their application resource data in the `/etc/X11/app-defaults/` directory. If you want to customize X applications globally, put your customizations in those files. They are marked as configuration files, so their contents will be preserved during upgrades.

10.5 Every distribution seems to have a different boot-up method. Tell me about Debian's.

Like all Unices, Debian boots up by executing the program `init`. The configuration file for `init` (which is `/etc/inittab`) specifies that the first script to be executed should be `/etc/init.d/rcS`. This script runs all of the scripts in `/etc/rcS.d/` by sourcing or forking subprocess depending on their file extension to perform initialization such as to check and to mount file systems, to load modules, to start the network services, to set the clock, and to perform other initialization. Then, for compatibility, it runs the files (except those with a '.'in the filename) in `/etc/rc.boot/` too. Any scripts in the latter directory are usually reserved for system administrator use, and using them in packages is deprecated.

After completing the boot process, `init` executes all start scripts in a directory specified by the default runlevel (this runlevel is given by the entry for `id` in `/etc/inittab`). Like most System V compatible Unices, Linux has 7 runlevels:

- 0 (halt the system),
- 1 (single-user mode),
- 2 through 5 (various multi-user modes), and
- 6 (reboot the system).

Debian systems come with id=2, which indicates that the default runlevel will be '2' when the multi-user state is entered, and the scripts in `/etc/rc2.d/` will be run.

In fact, the scripts in any of the directories, `/etc/rcN.d/` are just symbolic links back to scripts in `/etc/init.d/`. However, the *names* of the files in each of the `/etc/rcN.d/` directories are selected to indicate the *way* the scripts in `/etc/init.d/` will be run. Specifically, before entering any runlevel, all the scripts beginning with 'K' are run; these scripts kill services. Then all the scripts beginning with 'S' are run; these scripts start services. The two-digit number following the 'K' or 'S' indicates the order in which the script is run. Lower numbered scripts are executed first.

This approach works because the scripts in `/etc/init.d/` all take an argument which can be either 'start', 'stop', 'reload', 'restart' or 'force-reload' and will then do the task indicated by the argument. These scripts can be used even after a system has been booted, to control various processes.

For example, with the argument 'reload' the command

```
/etc/init.d/sendmail reload
```

sends the sendmail daemon a signal to reread its configuration file.

10.6 It looks as if Debian does not use `rc.local` to customize the boot process; what facilities are provided?

Suppose a system needs to execute script `foo` on start-up, or on entry to a particular (System V) runlevel. Then the system administrator should:

- Enter the script `foo` into the directory `/etc/init.d/`.
- Run the Debian command `update-rc.d` with appropriate arguments, to set up links between the (command-line-specified) directories rc?.d and `/etc/init.d/foo`. Here, '?' is a number from 0 through 6 and corresponds to each of the System V runlevels.
- Reboot the system.

The command `update-rc.d` will set up links between files in the directories rc?.d and the script in `/etc/init.d/`. Each link will begin with a 'S' or a 'K', followed by a number, followed by the name of the script. Scripts beginning with 'S' in `/etc/rcN.d/` are executed when runlevel `N` is entered. Scripts beginning with a 'K' are executed when leaving runlevel `N`.

One might, for example, cause the script `foo` to execute at boot-up, by putting it in `/etc/init.d/` and installing the links with `update-rc.d foo defaults 19`. The argument 'defaults' refers to the default runlevels, which are 2 through 5. The argument '19' ensures that `foo` is called before any scripts containing numbers 20 or larger.

10.7 How does the package management system deal with packages that contain configuration files for other packages?

Some users wish to create, for example, a new server by installing a group of Debian packages and a locally generated package consisting of configuration files. This is not generally a good idea, because `dpkg` will not know about those configuration files if they are in a different package, and may write conflicting configurations when one of the initial "group" of packages is upgraded.

Instead, create a local package that modifies the configuration files of the "group" of Debian packages of interest. Then `dpkg` and the rest of the package management system will see that the files have been modified by the local "sysadmin" and will not try to overwrite them when those packages are upgraded.

10.8 How do I override a file installed by a package, so that a different version can be used instead?

Suppose a sysadmin or local user wishes to use a program "login-local" rather than the program "login" provided by the Debian `login` package.

Do **not**:

- Overwrite `/bin/login` with `login-local`.

The package management system will not know about this change, and will simply overwrite your custom `/bin/login` whenever `login` (or any package that provides `/bin/login`) is installed or updated.

Rather, do

- Execute:

  ```
  dpkg-divert --divert /bin/login.debian /bin/login
  ```

  in order to cause all future installations of the Debian `login` package to write the file `/bin/login` to `/bin/login.debian` instead.

- Then execute:

  ```
  cp login-local /bin/login
  ```

  to move your own locally-built program into place.

Details are given in the manual page `dpkg-divert(8)`.

10.9 How can I have my locally-built package included in the list of available packages that the package management system knows about?

Execute the command:

```
dpkg-scanpackages BIN_DIR OVERRIDE_FILE [PATHPREFIX] > my_Packages
```

where:

- BIN-DIR is a directory where Debian archive files (which usually have an extension of ".deb") are stored.
- OVERRIDE_FILE is a file that is edited by the distribution maintainers and is usually stored on a Debian FTP archive at `indices/override.main.gz` for the Debian packages in the "main" distribution. You can ignore this for local packages.
- PATHPREFIX is an *optional* string that can be prepended to the `my_Packages` file being produced.

Once you have built the file `my_Packages`, tell the package management system about it by using the command:

```
dpkg --merge-avail my_Packages
```

If you are using APT, you can add the local repository to your `sources.list(5)` file, too.

10.10 Some users like mawk, others like gawk; some like vim, others like elvis; some like trn, others like tin; how does Debian support diversity?

There are several cases where two packages provide two different versions of a program, both of which provide the same core functionality. Users might prefer one over another out of habit, or because the user interface of one package is somehow more pleasing than the interface of another. Other users on the same system might make a different choice.

Debian uses a "virtual" package system to allow system administrators to choose (or let users choose) their favorite tools when there are two or more that provide the same basic functionality, yet satisfy package dependency requirements without specifying a particular package.

For example, there might exist two different versions of newsreaders on a system. The news server package might 'recommend' that there exist *some* news reader on the system, but the choice of `tin` or `trn` is left up to the individual user. This is satisfied by having both the `tin` and `trn` packages provide the virtual package `news-reader`. *Which* program is invoked is determined by a link pointing from a file with the virtual package name `/etc/alternatives/news-reader` to the selected file, e.g., `/usr/bin/trn`.

A single link is insufficient to support full use of an alternate program; normally, manual pages, and possibly other supporting files must be selected as well. The Perl script `update-alternatives` provides a way of ensuring that all the files associated with a specified package are selected as a system default.

For example, to check what executables provide 'x-window-manager', run:

```
update-alternatives --display x-window-manager
```

If you want to change it, run:

```
update-alternatives --config x-window-manager
```

And follow the instructions on the screen (basically, press the number next to the entry you'd like better).

If a package doesn't register itself as a window manager for some reason (file a bug if it's in error), or if you use a window manager from /usr/local directory, the selections on screen won't contain your preferred entry. You can update the link through command line options, like this:

```
update-alternatives --install /usr/bin/x-window-manager \
  x-window-manager /usr/local/bin/wmaker-cvs 50
```

The first argument to '–install' option is the symlink that points to /etc/alternatives/NAME, where NAME is the second argument. The third argument is the program to which /etc/alternatives/NAME should point to, and the fourth argument is the priority (larger value means the alternative will more probably get picked automatically).

To remove an alternative you added, simply run:

```
update-alternatives --remove x-window-manager /usr/local/bin/wmaker-cvs
```

Chapter 11

Getting support for Debian GNU/Linux

11.1 What other documentation exists on and for a Debian system?

- Installation instructions for the current release: see http://www.debian.org/releases/stable/installmanual.
- Policy manual documents the policy requirements for the distribution, i.e. the structure and contents of the Debian archive, several design issues of the operating system etc. It also includes the technical requirements that each package must satisfy to be included in the distribution, and documents the basic technical aspects of Debian binary and source packages.

  Get it from the debian-policy package, or at http://www.debian.org/doc/devel-manuals#poli
- Documentation on installed Debian packages: Most packages have files that are unpacked into /usr/doc/PACKAGE.
- Documentation on the Linux project: The Debian package doc-linux installs all of the most recent versions of the HOWTOs and mini-HOWTOs from the Linux Documentation Project (http://www.tldp.org/).
- Unix-style 'man' pages: Most commands have manual pages written in the style of the original Unix 'man' files. They are referenced by the section of the 'man' directory where they reside: e.g., foo(3) refers to a manual page which resides in /usr/share/man/man3/, and it can be called by executing the command: man 3 foo, or just man foo if section 3 is the first one containing a page on foo.

  One can learn which directory of /usr/share/man/ contains a certain manual page by executing man -w foo.

  New Debian users should note that the 'man' pages of many general system commands are not available until they install these packages:

  – man-db, which contains the man program itself, and other programs for manipulating the manual pages.

  - `manpages`, which contains the system manual pages. (see 'How does Debian support non-English languages?' on page 17).

- GNU-style 'info' pages: User documentation for many commands, particularly GNU tools, is available not in 'man' pages, but in 'info' files which can be read by the GNU tool `info`, by running `M-x info` within GNU Emacs, or with some other Info page viewer.

  Its main advantage over the original 'man' pages are that it is a hypertext system. It does *not* require the WWW, however; `info` can be run from a plain text console. It was designed by Richard Stallman and preceded the WWW.

Note that you may access a lot of documentation on your system by using a WWW browser, through 'dwww' or 'dhelp' commands, found in respective packages.

11.2 Are there any on-line resources for discussing Debian?

Yes. In fact, the main method of support Debian provides to our users is by the way of email.

11.2.1 Mailing lists

There are a lot of Debian-related mailing lists (`http://www.debian.org/MailingLists/`).

On a system with the `doc-debian` package installed there is a complete list of mailing lists in `/usr/share/doc/debian/mailing-lists.txt`.

Debian mailing lists are named following the pattern debian-*list-subject*. Examples are debian-announce, debian-user, debian-news. To subscribe to any list debian-*list-subject*, send mail to debian-*list-subject*-request@lists.debian.org with the word "subscribe" in the Subject: header. Be sure to remember to add *-request* to the email address when using this method to subscribe or unsubscribe. Otherwise your email will go to the list itself, which could be embarrassing or annoying, depending on your point of view.

If you have a forms-capable World Wide Web browser, you can subscribe to mailing lists using the WWW form (`http://www.debian.org/MailingLists/subscribe`). You can also un-subscribe using a WWW form (`http://www.debian.org/MailingLists/unsubscribe`).

The list manager's e-mail address is `<listmaster@lists.debian.org>`, in case you have any trouble.

Archives of the Debian mailing lists are available via WWW at `http://lists.debian.org/`.

What is the code of conduct for the mailing lists?

When using the Debian mailing lists, please follow these rules:

- Do not send spam. See the Debian mailing list advertising policy (http://www.debian.org/MailingLists/#ads).
- Do not flame; it is not polite. The people developing Debian are all volunteers, donating their time, energy and money in an attempt to bring the Debian project together.
- Do not use foul language; besides, some people receive the lists via packet radio, where swearing is illegal.
- Make sure that you are using the proper list. *Never* post your (un)subscription requests to the mailing list itself[1]
- See section 'How do I report a bug in Debian?' on the following page for notes on reporting bugs.

11.2.2 Maintainers

Users can address questions to individual package maintainers using email. To reach a maintainer of a package called xyz, send email to *xyz@packages.debian.org*.

11.2.3 Usenet newsgroups

Users should post non-Debian-specific questions to one of the Linux USENET groups, which are named comp.os.linux.* or linux.*. There are several lists of Linux Usenet newsgroups and other related resources on the WWW, e.g. on the Linux Online (http://www.linux.org/docs/usenet.html) and LinuxJournal (http://www.linuxjournal.com/helpdesk.php) sites.

11.3 Is there a quick way to search for information on Debian GNU/Linux?

There is a variety of search engines that serve documentation related to Debian:

- Debian WWW search site (http://search.debian.org/).
- Google Groups (http://groups.google.com/): a search engine for newsgroups.

  For example, to find out what experiences people have had with finding drivers for Promise controllers under Debian, try searching on the phrase `Promise Linux`

[1] Use the debian-*list-subject*-REQUEST@lists.debian.org address for that.

`driver`. This will show you all the postings that contain these strings, i.e. those where people discussed these topics. If you add `Debian` to those search strings, you'll also get the postings specifically related to Debian.

- Any of the common web spidering engines, such as AltaVista (`http://www.altavista.com/`) or Google (`http://www.google.com/`), as long as you use the right search terms.

  For example, searching on the string "cgi-perl" gives a more detailed explanation of this package than the brief description field in its control file.

11.4 Are there logs of known bugs?

The Debian GNU/Linux distribution has a bug tracking system (BTS) which files details of bugs reported by users and developers. Each bug is given a number, and is kept on file until it is marked as having been dealt with.

Copies of this information are available at `http://www.debian.org/Bugs/`.

A mail server provides access to the bug tracking system database via e-mail. In order to get the instructions, send an e-mail to request@bugs.debian.org with "help" in the body.

11.5 How do I report a bug in Debian?

If you have found a bug in Debian, please read the instructions for reporting a bug in Debian. These instructions can be obtained in one of several ways:

- By anonymous FTP. Debian mirror sites contain the instructions in the file `doc/bug-reporting.txt`.
- From the WWW. A copy of the instructions is shown at `http://www.debian.org/Bugs/Reporting`.
- On any Debian system with the `doc-debian` package installed. The instructions are in the file `/usr/doc/debian/bug-reporting.txt`.

You can use the packages `bug` or `reportbug` that will guide you through the reporting process and mail the message to the proper address, with some extra details about your system added automatically.

If you want to mail the report with an email program, send a message to `<submit@bugs.debian.org>`. The message's first line must be similar to

```
Package: package-name
```

(replace *package-name* with the name of the package). The next line should relate the package version number in a similar way:

```
Version: version-number
```

The version number for any package installed on your system can be obtained using the command line

```
dpkg -s package-name
```

This section is referred to as the pseudo-header. The rest of the message should contain the description of the bug (please make it moderately detailed), the Debian release you are using, and versions of other relevant packages. The Debian release number will be displayed by the command

```
cat /etc/debian_version
```

Expect to get an automatic acknowledgement of your bug report. It will also be automatically given a bug tracking number, entered into the bug log and forwarded to the debian-bugs-dist mailing list.

If you identify a bug that is common to many programs, then rather than entering dozens of very similar bug reports, you might prefer to send individual bugs to <maintonly@bugs.debian.org> (instead of the submit@... address) to reach only the respective package maintainers, and then send a summary report to debian-devel and/or debian-bugs-dist mailing lists.

Additionally, there exists a Debian package checker, called Lintian (http://www.debian.org/lintian/), which is designed to mechanically check Debian packages for policy violations and common packaging errors. Thus, if you detect a bug in a package which is likely to appear in other packages too, it might be better to get in contact with the Lintian maintainers at <lintian-maint@debian.org> so that a new check is written for Lintian instead of reporting the bug directly. This will most likely prevent the bug from appearing in future versions of the package again, or in any other package of the distribution.

You can also use <quiet@bugs.debian.org>, to submit bug reports to the BTS only, without having them sent either to debian-bugs-dist or to the maintainer. This 'quiet' address is used very rarely, e.g. when you want to send some minor data to your report, that should just be recorded in the log, or when you want to record something in the BTS log but you already sent it to the maintainer.

Chapter 12

Contributing to the Debian Project

Donations of time (to develop new packages, maintain existing packages, or provide user support), resources (to mirror the FTP and WWW archives), and money (to pay for new testbeds as well as hardware for the archives) can help the project.

12.1 How can I become a Debian software developer?

The development of Debian is open to all, and new users with the right skills and/or the willingness to learn are needed to maintain existing packages which have been "orphaned" by their previous maintainers, to develop new packages, and to provide user support.

The description of becoming a Debian developer can be found at the New Maintainer's Corner (http://www.debian.org/devel/join/newmaint) at the Debian web site.

12.2 How can I contribute resources to the Debian project?

Since the project aims to make a substantial body of software rapidly and easily accessible throughout the globe, mirrors are urgently needed. It is desirable but not absolutely necessary to mirror all of the archive. Please visit the Debian mirror size (http://www.debian.org/mirror/size) page for information on the disk space requirements.

Most of the mirroring is accomplished entirely automatically by scripts, without any interaction. However, the occasional glitch or system change occurs which requires human intervention.

If you have a high-speed connection to the Internet, the resources to mirror all or part of the distribution, and are willing to take the time (or find someone) who can provide regular maintenance of the system, then please contact <debian-admin@lists.debian.org>.

12.3 How can I contribute financially to the Debian project?

One can make individual donations to one of two organizations that are critical to the development of the Debian project.

12.3.1 Software in the Public Interest

Software in the Public Interest (SPI) is an IRS 501(c)(3) non-profit organization, formed when FSF withdrew their sponsorship of Debian. The purpose of the organization is to develop and distribute free software.

Our goals are very much like those of FSF, and we encourage programmers to use the GNU General Public License on their programs. However, we have a slightly different focus in that we are building and distributing a Linux system that diverges in many technical details from the GNU system planned by FSF. We still communicate with FSF, and we cooperate in sending them changes to GNU software and in asking our users to donate to FSF and the GNU project.

SPI can be reached at: `http://www.spi-inc.org/`.

12.3.2 Free Software Foundation

At this time there is no formal connection between Debian and the Free Software Foundation. However, the Free Software Foundation is responsible for some of the most important software components in Debian, including the GNU C compiler, GNU Emacs, and much of the C runtime library that is used by all programs on the system. FSF pioneered much of what free software is today: they wrote the General Public License that is used on much of the Debian software, and they invented the "GNU" project to create an entirely free Unix system. Debian should be considered a descendent of the GNU system.

FSF can be reached at: `http://www.fsf.org/`.

Chapter 13

Redistributing Debian GNU/Linux in a commercial product

13.1 Can I make and sell Debian CDs?

Go ahead. You do not need permission to distribute anything we have *released*, so that you can master your CD as soon as the beta-test ends. You do not have to pay us anything. Of course, all CD manufacturers must honor the licenses of the programs in Debian. For example, many of the programs are licensed under the GPL, which requires you to distribute their source code.

Also, we will publish a list of CD manufacturers who donate money, software, and time to the Debian project, and we will encourage users to buy from manufacturers who donate, so it is good advertising to make donations.

13.2 Can Debian be packaged with non-free software?

Yes. While all the main components of Debian are free software, we provide a non-free directory for programs that are not freely redistributable.

CD manufacturers *may* be able to distribute the programs we have placed in that directory, depending on the license terms or their private arrangements with the authors of those software packages. CD manufacturers can also distribute the non-free software they get from other sources on the same CD. This is nothing new: free and commercial software are distributed on the same CD by many manufacturers now. Of course we still encourage software authors to release the programs they write as free software.

13.3 I am making a special Linux distribution for a "vertical market". Can I use Debian GNU/Linux for the guts of a Linux system and add my own applications on top of it?

Yes. For example, one person is building a "Linux for Hams" distribution, with specialized programs for Radio Amateurs. He is starting with Debian as the "base system", and adding programs to control the transmitter, track satellites, etc. All of the programs he adds are packaged with the Debian packaging system so that his users will be able to upgrade easily when he releases subsequent CDs.

There are several other Debian-derived distributions already on the market, such as Corel Linux and Storm Linux, that are targeted at a different kind of audience than the original Debian GNU/Linux is, but use most of our components in their product.

Debian also provides a mechanism to allow developers and system administrators to install local versions of selected files in such a way that they will not be overwritten when other packages are upgraded. This is discussed further in the question on 'How do I override a file installed by a package, so that a different version can be used instead?' on page 52.

13.4 Can I put my commercial program in a Debian "package" so that it installs effortlessly on any Debian system?

Go right ahead. The package tool is free software; the packages may or may not be free software, it can install them all.

Chapter 14

Changes expected in the next major release of Debian

14.1 Increased security

Debian contains support for shadow passwords since release 1.3. In addition, the Linux library of Pluggable Authentication Modules (a.k.a. libpam (http://www.kernel.org/pub/linux/libs/pam/);) that allows sysadmins to choose authorization modes on an application-specific basis is available, and initially set to authenticate via shadow password.

Including full support for advanced authentication methods such as Kerberos, RSBAC and others is in progress.

14.2 Extended support for non-English users

Debian already has some support for non-English users, see 'How does Debian support non-English languages?' on page 17.

We hope to find people who will provide support for even more languages, and translate. Some programs already support internationalization, so we need message catalogs translators. Many programs still remain to be properly internationalized.

The GNU Translation Project ftp://ftp.gnu.org/pub/gnu/ABOUT-NLS works on internationalizing the GNU programs.

14.3 More architectures

Complete Debian system on other architectures such as SPARC64 or SuperH is expected soon.

14.4 More kernels

In addition to Debian GNU/Hurd, Debian is being ported to several BSD kernels, namely those from NetBSD, FreeBSD and OpenBSD.

Chapter 15

General information about the FAQ

15.1 Authors

The first edition of this FAQ was made and maintained by J.H.M. Dassen (Ray) and Chuck Stickelman. Authors of the rewritten Debian GNU/Linux FAQ are Susan G. Kleinmann and Sven Rudolph. After them, the FAQ was maintained by Santiago Vila. The current maintainer is Josip Rodin.

Parts of the information came from:

- The Debian-1.1 release announcement, by Bruce Perens (http://www.perens.com/).
- The Linux FAQ, by Ian Jackson (http://www.chiark.greenend.org.uk/~ijackson/).
- Debian Mailing Lists Archives (http://lists.debian.org/),
- the dpkg programmers' manual and the Debian Policy manual (see 'What other documentation exists on and for a Debian system?' on page 55)
- many developers, volunteers, and beta testers, and
- the flaky memories of its authors. :-)

The authors would like to thank all those who helped make this document possible.

All warranties are disclaimed. All trademarks are property of their respective trademark owners.

15.2 Feedback

Comments and additions to this document are always welcome. Please send e-mail to <doc-debian@packages.debian.org>, or submit a wishlist bug report against the doc-debian package.

15.3 Availability

The latest version of this document can be viewed on the Debian WWW pages at http://www.debian.org/doc/FAQ/.

It is also available for download in plain text, HTML, PostScript and PDF formats at http://www.debian.org/doc/user-manuals#faq. Also, there are several translations there.

The original SGML files used to create this document are also available in doc-debian's source package, or in CVS at: :pserver:anonymous@cvs.debian.org:/cvs/debian-doc/ddp/manuals.sgml/faq

15.4 Document format

This document was written using the DebianDoc SGML DTD (rewritten from LinuxDoc SGML). DebianDoc SGML systems enables us to create files in a variety of formats from one source, e.g. this document can be viewed as HTML, plain text, TeX DVI, PostScript, PDF, or GNU info.

Conversion utilities for DebianDoc SGML are available in Debian package debiandoc-sgml.