

# The Debian Maintenance HOWTO

Joey Schulze, [joey@infodrom.org](mailto:joey@infodrom.org)

November 23rd, 2014

## **Abstract**

The aim of this document is to give users a quick introduction into Maintaining a Debian System. For further information please refer to the proper chapter.

## Copyright Notice

Copyright (c) 2001-2011,2014 joey Schulze <[joey@infodrom.org](mailto:joey@infodrom.org)> and others.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 2.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	History	1
1.2	Special Terms, Acronyms and Abbreviations	1
1.3	Thanks and Credits	2
1.4	Feedback	2
1.5	Todo	2
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Boot Medium	3
2.1.1	Floppy Images	3
2.1.2	CD Images	3
2.2	Hardware	3
2.3	Preload Modules	4
2.4	Configuring Device Driver Modules	4
2.5	Network	4
2.5.1	/etc/hostname	4
2.5.2	/etc/hosts	4
2.5.3	/etc/host.conf	5
2.5.4	/etc/resolv.conf	5
2.5.5	/etc/network/interfaces	5
2.5.6	/etc/init.d/network	5
2.6	Passwords	6
2.6.1	Shadow Passwords	6
2.6.2	DES-Encrypted Passwords	6
2.6.3	MD5-Encrypted Passwords	6
2.7	Selecting Tasks	6
2.8	Installing Packages	6
2.9	Configuring X11	6
2.9.1	Using XDM	7
2.9.2	Keyboard Layout	7
2.10	Configuring ISDN	8
2.11	Custom Boot Floppies	8

<b>3</b>	<b>Package Maintenance</b>	<b>9</b>
3.1	Package Basics	9
3.1.1	Static Package Information	9
3.1.2	General Information	9
3.1.3	Dependencies and Conflicts	10
3.2	Package Managers	10
3.3	Installed Packages	11
3.4	Information about Packages	11
3.5	Locate Files and Packages	11
3.6	New Packages	12
3.7	Package Database Update	12
3.8	APT Sources	12
3.8.1	Accessing network servers	13
3.8.2	Accessing local directories	13
3.8.3	Accessing cd-roms	13
3.8.4	Configuration for stable	13
3.8.5	Configuration for unstable	13
3.9	Package Difference Files	14
3.10	Introduction to dpkg	14
3.10.1	dpkg --install	14
3.10.2	dpkg --configure	14
3.10.3	dpkg --list	14
3.10.4	dpkg --status	15
3.10.5	dpkg --search	15
3.10.6	dpkg --listfiles	15
3.10.7	dpkg --remove	15
3.10.8	dpkg --purge	15
3.10.9	Option --force-overwrite	15
3.10.10	Option --force-depends	15
3.10.11	Option --force-conflicts	16
3.11	Introduction to apt-get	16
3.11.1	apt-get update	16
3.11.2	apt-get install	16
3.11.3	apt-get upgrade	16
3.11.4	apt-get dist-upgrade	17
3.12	Configuring APT	17
3.12.1	Translations	18
3.13	Brandnew Packages	18
3.14	Packages not in Debian	18
3.15	Virtual packages	19
3.16	Pseudo packages	19
3.17	Package Pools and Distributions	19

3.18	The testing Distribution	20
3.19	Forcing Package Installation	20
3.20	Creating Local Archives	20
3.21	Upgrading single Packages	21
3.22	Upgrading everything	21
3.23	Distribution-Upgrade	22
3.23.1	Upgrade with apt-get	22
3.23.2	Upgrade with aptitude	22
3.24	Upgrading to unstable	22
<b>4</b>	<b>Alternatives</b>	<b>25</b>
4.1	Display Alternatives	25
4.2	Configure Alternative	25
4.3	Add Alternative	25
4.4	Remove Alternative	25
<b>5</b>	<b>System Administration</b>	<b>27</b>
5.1	System cloning	27
5.2	Keyboard	27
5.3	Time and Timezone	27
5.3.1	Reconfiguring the Timezone	28
5.3.2	System Clock	28
5.3.3	Automatic Clock Adjustments	28
5.4	Systemwide environment configuration	28
5.5	Languages (locales)	28
5.6	Character sets (foreign characters)	29
5.6.1	Readline	29
5.6.2	less	29
5.6.3	Mutt	29
5.6.4	Emacs	30
5.6.5	Console modus	30
<b>6</b>	<b>Menus</b>	<b>31</b>
6.1	Menu files	31
<b>7</b>	<b>Booting and Runlevel</b>	<b>33</b>
7.1	Reloading/Restarting a service	33
7.2	Adding a new service	33
7.3	Removing a service	34
<b>8</b>	<b>Support</b>	<b>35</b>
8.1	Via Mail	35
8.2	Online	35
8.3	Personal	35

---

<b>9 Building Packages</b>	<b>37</b>
9.1 Source Packages	37
9.2 Unpacking the Source on Debian	37
9.3 Unpacking the Source somewhere	37
9.4 Rebuilding	38
9.5 Packaging recent source	38
9.6 Backporting	38
9.7 Finding new upstream source	38
9.8 New upstream source	39
9.9 Packaging	39
9.9.1 Basics	39
9.9.2 Developers Resources	40

# Chapter 1

## Introduction

This document describes many issues and techniques concerning the maintenance of a Debian GNU system. It is intended to be used as guide for new administrators as well as dictionary for long-term administrators and users who already work with the system but tend to forget how certain things work, or who would like to use otherwise occupied memory for other things.

### 1.1 History

The Debian Project was officially founded by Ian Murdock on August 16th, 1993. Ian Murdock begun it as a new distribution which would be made openly, in the sprit of Linux and GNU. No other distribution fulfilled this goal in 1993. This effort was sponsored by the FSF's GNU project for one year (November 1994 to November 1995).

Debian was meant to be carefully and conscientiously put together, and to be maintained and supported with similar care. It started as a small, tightly-knit group of Free Software hackers, and gradually grew to become a large, well-organized community of developers and users.

Debian has become the only distribution that is open for every developer and user for committing their work. It is also the only large project that has defined their own rules and policy documents which help organizing this project. In order to receive high quality and to keep it, there are a lot of regulations, automatisms and documents describing parts of the system. Debian has also become the one and only distribution with a detailed set of dependencies and other inter-package relationships.

You will find a detailed story about Debian in the Debian Project History (<http://www.debian.org/doc/manuals/project-history/>) on the main web server of Debian.

### 1.2 Special Terms, Acronyms and Abbreviations

The Debian project has introduced several terms which are only very rarely used outside the project. Add to that several terms commonly used in the Free Software Community and it is possible to construct sentences unreadable to people outside of the community or who are new.

- **DFSG** – Debian Free Software Guidelines ([http://www.debian.org/social\\_contract#guidelines](http://www.debian.org/social_contract#guidelines)). Only software compliant to these guidelines can be distributed in the Debian system. Other packages may be allowed in the non-free section.
- **downstream** – The opposite of *upstream*. The maintainer inside a distribution is called that way by the developers who have been written a given package.
- **package** – Software and its documentation accompanied by meta-information is packed together into so called packages. Source packages contain the machine-readable source code while binary package contain machine-executable programs and compiled documentation.
- **revision** – Denotes changes the package maintainer inside Debian has applied (<http://www.debian.org/doc/developers-reference/ch-pkgs>) to a particular package. The initial revision is *1*, a revision referring to a non-maintainer upload looks like *1.1* and the revision for a binary-only rebuild is similar to *1+b1*. The entire package version is composed of the version and the revision.

- **upstream** – The developers who have written the software and documentation inside a package are called upstream in order to distinguish them from the Debian maintainer. The latter usually *only* packs the software together and integrates it into the Debian system.
- **version** – The version upstream developers gave to a package when they released it. Debian maintainers add a *+d*<sub>fs</sub>*g* suffix to the version if they had to repackage the package in order to remove components not compliant with the DFSG. The entire package version is composed of the version and the revision.

## 1.3 Thanks and Credits

Thanks to the people who have founded The Debian Project and worked on it from the beginning. These people ultimately paved the way into a new generation of distribution and made current development possible.

Thanks as well to all Debian developers and contributing users.

## 1.4 Feedback

This document is a service to users of the Debian operating system, it will only be useful if it will be used. Therefore feedback is highly appreciated and, to some extend, required to keep the document up to date.

Please send comments, corrections, additions and wishes to this document to Martin Schulze <joe@debian.org>.

## 1.5 Todo

- ISDN Configuration
- deborphan, debfoster
- PPP Configuration
- Dial-on-Demand / Masquerading
- apt-proxy
- apt-move, debian-mirror
- lintian
- archive.debian.org
- packages.debian.org
- apt-zip
- useradd/userdel/addgroup
- Install on RAID/raiserfs
- Keyboard with showkey and xev
- updates while offline
- hosts.allow/hosts.deny/ssh/sshd\_conf (rootlogin)
- Printing (smbprint, hosts.lpd, printcap, cups)
- ssh -o 'ForwardX11 yes' root@localhost
- Archive-maint / Pools / ftp-master -> mirror -> mirror
- BTS, /usr/doc/debian/bug-reporting.txt, querybts, bug
- .deb = ar -x
- xdm | gdm | kdm
- signing, keyring.debian.org
- DBS
- apt-get -b source openssh
- Permissions, audio, dialout, floppy etc.
- user maint: adduser, addgroup
- apt-listchanges
- debian-mirror
- debian-cd
- equivs
- apt-ftparchive
- dlocate

–initscripts. The logging is perhaps not enabled by default and it is perhaps not detailed to the extent as suggested in the bugreport, but by running `echo 'BOOTLOGD_ENABLE=Yes' > /etc/default/bootlogd` one can enable initscript logging to `/var/log/boot`, which will show everything that was sent to the console during bootup (excluding –kernel messages).

–>



## Chapter 2

# Installation

The installation section only covers installation on intel x86 compatible systems. Except for the boot medium all chapters also apply for other architectures. Please also read the official installation documentation (<ftp://ftp.debian.org/debian/dists/stable/main/disks-i386/current/doc/install.en.html>).

### 2.1 Boot Medium

There are several ways to install Debian on your computer. While the most convenient method which runs from the network entirely requires the machine to be able to boot off of the network, only few machines can do that, especially non-intel-based ones like sparc, mips or powerpc machines. Other ways use a CD or a floppy disk.

The following will describe some ways to install Debian. Please keep in mind that you can still do a full floppy installation if this doesn't work. However, you will need to have 17 sane floppy disks available (for Debian woody). This can become a showstopper... So you may want to consider purchasing a professionally pressed CD set.

If still doesn't work because your BIOS may be too old and may not support ISOLINUX you write the Smart Boot Manager image `sbm.bin` to a floppy which you can find in the `install` directory. This will boot an OS independent boot manager. To write the image to a floppy use `dd if=sbm.bin of=/dev/fd0` (for GNU/Linux or Unix) or use `rawrite` (for MS-DOS).

If you can't boot off of your CD-ROM drive, create your own boot and root disk (`rescue.bin` and `root.bin`) and try to use the remaining (drivers, base, packages) from CD-ROM. If you can't get your CD-ROM recognized by the kernel, you can copy the entire cd to your harddisk and use it from there. Since Linux can read vfat you can copy it file by file to your windows partition or by copying the image and using `mount -t iso9660 -o loop /path/to/image /mnt` to mount it via loopback.

#### 2.1.1 Floppy Images

Proper floppy disk images are provided by debian.org at <ftp://ftp.debian.org/debian/dists/stable/main/disks-i386/current/> for various architectures. There are images for 1.2, 1.44 and 2.88 MB large disks. Images are called `*.bin`, you will have to write them to a floppy by using `dd` under unix or `rawrite` under dos.

#### 2.1.2 CD Images

The Debian project also provides readily mastered cd images for their stable distribution. They are bootable and optionally contain non-US stuff as well. Please find them on [cdimage.debian.org](http://cdimage.debian.org) ([http://cdimage.debian.org/](http://cdimage.debian.org)) or at [linuxiso.org](http://www.linuxiso.org/debian.html) (<http://www.linuxiso.org/debian.html>). Both places contain images for various architectures.

### 2.2 Hardware

Debian uses the Kernel produced and maintained by Linus Torvalds. Due to this fact there is no special Debian Hardware list available. All the hardware that will work with the same kernel version, will work with Debian as well.

Debian GNU/Linux 2.2 alias potato is shipped with Linux Kernel 2.2 and a possibility to upgrade to Linux Kernel 2.4.0. This includes support for all kind of IDE and SCSI devices. Using the newer kernel you will also be able to use USB mostly out-of-the-box.

If you occur hardware problems or want to know if something is supported by Linux, a good address seems to be the SuSE Support Database (<http://sdb.suse.de/>). Even though this database is maintained by a different vendor, much of its content applies not only to their distribution but to any, which makes it a worthwhile resource.

## 2.3 Preload Modules

From Debian GNU/Linux 2.2r2 you are able to preload modules during the installation. This step should only be required if your harddrives are not recognized or you need a network connection during installation. Your harddrive may not be automatically recognized by Linux if it uses a hardware raid devices.

The preload-step will read additional modules from a special floppy disk that you have to create. You'll need to format the floppy disk with the msdos, ext2 or minix filesystem and place all required modules in the main directory of it.

The driver modules on this special floppy disk have to match the kernel used on the boot disk. If your driver is already in the standard kernel, try to use the one from the `drivers.tgz` file. If that doesn't work, compile the module on your own.

## 2.4 Configuring Device Driver Modules

During installation you are asked to configure your device driver modules. This is just another name for kernel modules, thus the modules in question are stored in the `/lib/modules` directory. Normally you can and should skip most of the modules. Only a few are required to be configured.

The Linux kernel uses a sophisticated mechanism to automatically load modules that are to be used. This feature makes it almost superfluous to load modules at boot time. The only exception refers to modules that interact with certain hardware which is not unique (such as an ethernet card, a scsi adapter, a soundcard etc.)

In this step you should only configure your ethernet card, your ISDN card and if possible your soundcard. Leave everything else untouched, i.e don't install any filesystems and additional stuff, it will only eat up processor cycles and ram space. These modules will be loaded automatically by the kernel. On the other hand, the modules you have configured will be loaded at boot time when your machine boots. The name of these modules will be added to `/etc/modules` and may be edited at any time later.

If you feel that you have skipped too much of the device driver configuration and want to repeat this step, you can do that at any later time. The program called was `modconf` (<http://packages.debian.org/modconf>) which may be issued at any later time to configure device drivers. It will read modules from `/lib/modules`, so if you have comiled your own kernel without modules, `modconf` won't be able to do much.

## 2.5 Network

During the installation you have the opportunity to configure your network. If you plan to use the system as something else than a single system with no network access, please configure your network here. You won't be presented another simple way to configure it. If you forget to configure your network during the installation routine you have to edit the files named below manually.

### 2.5.1 /etc/hostname

This file contains the simple hostname of the machine (i.e. `klecker` instead of `klecker.debian.org`). This is the name that will be displayed at part of the login prompt, for example. You can use the fully qualified domain name (fqdn), `*getty` would use it instead.

### 2.5.2 /etc/hosts

This file contains the local mapping of ip addresses and hostnames, both simple and fqdn. Depending on your host order this file has the power to override even DNS entries. There should be at least one entry in that file like the following:

```
127.0.0.1      localhost hostname.domain.org hostname
```

If your host is connected to a network with an official IP address and an official hostname, you should leave out all names except of localhost. For details please refer to the manpage of `hosts`.

### 2.5.3 /etc/host.conf

This file controls how name lookups are performed on your system. Normally the file `/etc/hosts` is queried before nameserver lookups are performed. This is controlled by the following contents. For details please refer to the manpage of `host.conf`.

```
order hosts,bind
multi on
reorder on
```

### 2.5.4 /etc/resolv.conf

This file tells your resolver library which nameserver to query. Normally one or two nameservers are listed there, or localhost if the machine runs its own nameserver. The file's contents look like the following. For details please refer to the manpage of `resolv.conf`.

```
search infodrom.ffis.de infodrom.north.de north.de
nameserver 134.106.1.7
nameserver 127.0.0.1
```

### 2.5.5 /etc/network/interfaces

Since Debian 2.2 alias potato the network configuration is handled by this file. It is read by `/etc/init.d/network` if that is a proper start/stop (you will notice by scanning through the file and looking for a `case`-statement). In that case the file contains the entire configuration of network devices. For a machine with a real ethernet card (contrary to PCMCIA) it should look like the following example. For details please refer to the manpage of `interfaces`.

```
iface lo inet loopback

iface eth0 inet static
    address 206.246.226.45
    network 206.246.226.0
    netmask 255.255.255.0
    broadcast 206.246.226.255
    gateway 206.246.226.1
```

If you are already running the unstable distribution you need to add one more statement or no devices would be initialized at boot time:

```
auto lo eth0
```

If your machine is connected to a network that provides BOOTP or DHCP for maintaining IP addresses, this is supported as well, please read the manpage to find out the syntax.

### 2.5.6 /etc/init.d/network

This file reflects the network setup prior to Debian 2.2 alias potato. While it is a real init-script in potato, it was a plain and simple script to activate the network. If your `/etc/init.d/network`-file looks like the following and doesn't contain `start`- and `stop`-entries it should contain the entire network configuration as given in the following example:

```
ifconfig lo 127.0.0.1
route add -net 127.0.0.0 dev lo

# Configure the ethernet device or start SLIP/PPP below.
IPADDR="195.27.69.162"      # Your IP address.
NETMASK="255.255.255.240"   # Your netmask.
NETWORK="195.27.69.160"    # Your network address.
BROADCAST="195.27.69.175"   # Your broadcast address (blank if none).
GATEWAY="195.27.69.161"    # Your gateway address.

/sbin/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}
/sbin/route add -net ${NETWORK}
[ ! -z "$GATEWAY" ] && /sbin/route add default gw ${GATEWAY}
```

## 2.6 Passwords

Once the base system is installed on your new Debian machine you'll have to reboot the system and will be asked to set your root password and create a user account.

You should always create a user account since you should not overstress your root-account. Especially for new users to Unix/Linux the root account should only be used as exception. The account is dangerous since it bypasses nearly all security limitations and permissions. Even worse, programs designed to be started as user and run as different user may misbehave when started as root.

Installing Debian you'll have some possibilities to maintain passwords. The following list should explain some of the differences.

### 2.6.1 Shadow Passwords

Shadow passwords will give you a little bit more security since encrypted passwords won't be stored in a publically readable file (`/etc/passwd`) anymore but in a special file (`/etc/shadow`) that only root and certain programs may read. You should always select this option if your system is not a single-user system and you don't plan to use NIS. With NIS there are reports that it won't work with shadow passwords.

### 2.6.2 DES-Encrypted Passwords

Normally a unix system uses a variant of DES (Data Encryption Standard) algorithm for encrypting passwords. This algorithm is said to be quite secure, however exhaustive searches of its key space are possible using massively parallel computers.

### 2.6.3 MD5-Encrypted Passwords

Linux libraries support another algorithm to encrypt passwords, it is MD5. This algorithm is said to be more secure than the above DES and thus should be preferred among those.

## 2.7 Selecting Tasks

During the installation you will be provided with a list of less than 50 so called task packages. This will allow you to skip the selection of more than 3900 single packages. Each task package reflects one task that your future system will perform. Once you have selected the tasks the program manager will install the referenced packages.

If you have forgotton something you can repeat this step at any later stage by calling the program `tasksel`. This requires the similar named package to be installed already (issue `apt-get install tasksel` otherwise).

## 2.8 Installing Packages

At the end of the installation all selected packages will be installed. Below you'll find several ways to repeat package installation and information how to install new packages. Debian packages may use several ways to configure themself. During the installation some packages need interact with the user and ask a couple of questions.

You should answer these questions as good as you can. For example `exim` (or any other mail transport agent) want to know about the hostname, a smart host and others. If you are absolutely unsure, select the 'No configuration at all' item, but please run `eximconfig` (or a similar tool) as soon as you can afterwards. (Warning: I've once seen an smail going mad while not configured trying to deliver a mail to root.)

## 2.9 Configuring X11

For new users who want to run a graphical environment on their machine this is the most important task. First of all you should know what kind of graphics card is in your system. Depending on this information you'll need to select the X-server

(e.g. `xserver-svga` for Matrox G200 cards). You will be asked for every server if that server should become the default X-Server and if you want to configure it. Deny this for every wrong server.

The utilities used to configure your X-Server are `xf86config` and `XF86Setup`. If you don't manage to get your x-server configured properly during installation, please skip that and retry later by using one of these tools directly.

To find out which X-Server you need, I can only direct you to the XFree86 web site (<http://www.xfree86.org/>) at the moment. More information to be expected.

In Debian unstable or XFree86 4.x there is a new configuration utility called `dexconf` to be used with that version of XFree86.

## 2.9.1 Using XDM

Whether XDM is started upon system boot and provides a graphical login is controlled by the package `xdm`. It will only work that way if that package is installed. The same applies to GDM and KDM and their respective packages `gdm` and `kdm`.

## 2.9.2 Keyboard Layout

There are three or four places where the keyboard layout for X11 is configured. They are listed in the order of their appearance/overwrite capability.

### `/etc/X11/XF86Config`

This is the main configuration file for XFree86. To configure the keyboard a section of its own has to be edited. For example, to use an american keyboard you would use the following:

```
Section "Keyboard"
    Protocol      "Standard"
    XkbKeymap     "xfree86(us)"
EndSection
```

If you have a German keyboard and want to use it with the same keybinding, you would use something like the following configuration section:

```
Section "Keyboard"
    Protocol      "Standard"
    XkbLayout     "de"
    XkbVariant    "nodeadkeys"
EndSection
```

### `xmodmap`

You can modify your keyboard mapping with `xmodmap`. Please find an example in `/usr/X11R6/lib/X11/etc/xmodmap.std` and read about the syntax in the `xmodmap` manpage. To find out the keycode for pressed key, use the `xev` program to display.

### `xkeycaps`

This program provides a graphical interface to the keyboard and its mapping.

### KDE/Control Center

KDE uses its own method to map the keyboard, overloading the regular X11 settings. You are able to modify the keyboard mapping on each session.

## 2.10 Configuring ISDN

Many users in Europe use ISDN as a means to connect their machine to the internet in favour to slower modem connections. Once your card was detected setting up ISDN for dialin is quite easy.

First of all you need the kernel detect your isdn card. To make this happen, start the program `modconf`. The HiSax module (given that you use one of many cards that ship with that Siemens chipset) is in section `misc`. Upon configuration you'll have to specify which ISDN card you use. After that is done, the kernel should recognize the card and provide an interface to the ISDN subsystem. `modconf` will display if it was successful or not.

Once the card is detected properly install the `isdnutils` package and run the program `isdnconfig`. This program will create some template files for you that help you set up ISDN services. For dial on demand you will need to create network devices (`device.ipp0`) and synchronous ppp (`ipppd.ipp0`).

These files are only template files and you have to edit them manually in order to get ISDN configured. As a hint: the password may be stored in `/etc/ppp/chap-secrets` or `/etc/ppp/pap-secrets` instead of within the `/etc/isdn` directory where the isdn configuration resides.

## 2.11 Custom Boot Floppies

If you want to run Debian GNU/Linux on hardware that is not supported by the boot-floppies you will have to create your own boot floppies. This may be required if you want to install Linux on a hardware raid system, LVM or ReiserFS and you can't use a preloaded module with the regular installation process.

Before modifying the Debian boot-floppies you should first understand how they work. The following list should spread some light on this.

- 1 Debian boot-floppies contain a dos filesystem which make it easy to modify for anybody.
- 2 Debian uses `syslinux` to boot an intel-based pc off of floppies. This means that each boot-floppy has to contain a file `ldlinux.sys` that is used like a driver in dos-speaking.
- 3 `syslinux` is configured through the file `syslinux.cfg`. That file contains information about the boot process, i.e. additional parameters for the Linux kernel.
- 4 The boot floppy contains a file named `linux` which is a compressed (`bzip2` or `gzip`) Linux kernel.
- 5 Additionally you'll find `sys_map.gz` which is copied to `/boot/System.map` later.
- 6 The root image (`root.bin`) is an ext2 filesystem, compressed and padded with zeros.

If you have to modify the boot-floppy you'll have to mount the floppy loopback, modify it and umount it. Mounting is done with the following command.

```
mount -o loop -t msdos rescue.bin /mnt/
```

Modifying the root image (e.g. for including `lvmtools`) this process is a little bit more trickier. You'll have to stake the following stages:

- 1 Since the root image is a compressed file you'll have to decompress it. This is done by

```
zcat root.bin > /tmp/rootfs
```

- 2 Now you can mount it just like the boot image, except that you'll have to use a different filesystem.

```
mount -o loop -t ext2 /tmp/rootfs /mnt/
```

- 3 Now you can freely modify it. Please take into account that there is not much space left over on the root filesystems and that the included `glibc` is a stripped down version that may not work with regular executables. Thus it may be required to build static binaries for additional programs. Since they are larger than shared binaries, you may need a third floppy disk for supplying it.
- 4 Once you have modified the root filesystem you have to fold it back into the compressed root image. Normally it is not required to pad it with zeroes, though it's more reliable if you do. After umounting the directory the following steps will work:

```
gzip -9 < root > root.gz
dd if=/dev/zero of=root.bin bs=1k count=1440
dd if=root.gz of=root.bin bs=1k count=1440 conv=notrunc
```

## Chapter 3

# Package Maintenance

Let's assume that your system is installed properly and that you want to check which software is installed, to which package a particular file belongs, which packages require which other packages and eventually install new packages.

Debian is known to be upgradeable at any time without a reboot of the system, even from a remote terminal. A Debian system is usually only installed once in its lifetime, except the root disk suffers from hardware damage and requires a new installation. In other words, Debian systems don't get installed every now and then but upgraded to the current stable or unstable distribution instead.

### 3.1 Package Basics

Before we continue we need to draw attention to the basics of packages and management. In order to help maintaining the GNU/Linux system you have installed the Debian project splits the installed software into small partitions called packages. These packages contain the actual data or software that you install on your system.

With packages it is quite easy to install certain parts of the system and leave away others. For example, if you want to use KDE but not GNOME you don't have to install GNOME. However, thanks to sophisticated dependencies should a KDE program require a part of the GNOME system, this part will be installed as well, but only the required part.

The project distinguishes between source and binary packages. The binary packages are what the user or administrator will see most of the time. They contain pre-compiled software which is installed on a Debian system.

One binary package is always associated to exactly one source package, which — as the name suggests — contains the source for said binary package. In order to provide more sophisticated dependencies several source packages build more than only one binary package.

#### 3.1.1 Static Package Information

A binary package consists of more than just a set of files that are installed on the system. Next to the installed data are meta-information about the package and its relationship to other packages in the distribution.

This information is displayed with `apt-cache show package` and `dpkg -I package.deb` respectively if the package is locally available already. These parts are used by the packaging system to determine from where to fetch a package, which other packages are required, which other packages must not be installed at the same time, which other packages may be overwritten and so on.

Most of the information is provided by the maintainer of the given package and is subject to permanent adjustments during the lifetime of the package. Especially when talking about libraries such meta-information will change with future updates.

#### 3.1.2 General Information

Information about a package contains the name of the maintainer so you can find out who is responsible for the package and can actually get in touch with the person if you need to discuss anything with regards to the package. Several packages are maintained by a group these days and the maintainer address refers to a mailing list instead of an individual. Please don't hesitate to write to the list but keep in mind that it is most probably archived publicly so that your mail address will be exposed on the Internet.

Other information contain the name of the package, the version this entry refers to, the filename in the Debian archive and a number of signatures.

### 3.1.3 Dependencies and Conflicts

Dependency information is the most part of the meta-information of a package and is the key to Debian's package management. There are several different fields which are discussed in detail below that make the package manager decide which packages to install or remove. The names refer to fields (lines) in the meta information of a package.

#### Depends

This field defines which other packages are required for this one to run properly. The other packages always have to be installed or this one won't work. Hence, this package depends upon other packages. The package manager will also install the other packages upon installation.

This system also works the other way around. When a package is to be removed the package manager checks whether other packages still depend on it. If there are others, the package cannot be removed alone. Instead the other packages need to be removed as well in order to keep a well-working state of the system.

For example, in order to run a program for GNOME you'll need to have GNOME libraries installed at the same time. As a result of this, a GNOME package contains a dependency on several GNOME libraries. This will tell the package manager upon installation to write an error message or, if it will download files automatically, to download the libraries and install them as well.

#### Conflicts

This information is not used very often since most packages can be installed and run along with others. However, some packages are conflicting with others since they provide the same files or allocate the same port or provide the same service. These can't be installed at the same time. This field will tell the package manager to abort with an error message or to remove the other package so that the new one can be installed.

When using the `unstable` distribution it is possible that you notice `apt-get` wanting to remove half of the installed system. This often originates in a library transition that includes a new conflict. Before the other packages are recompiled to use the new library they are part of the conflict. Usually it is sufficient to just wait a few days until they have been uploaded again and are compiled against the new library.

For example there can only be one mail-transport-agent like `exim`, `sendmail`, `postfix` etc. or there can only be one development package of a library (contrary to the runtime package that may be installed with several versions).

#### Replaces

This field defines that the new package replaces (parts of) another package. With this definition it is possible that two packages that contain the same files may be installed at the same time.

This is often used when files are moved from one package to another, so that the package manager can install the new versions even though there are some file conflicts.

#### Provides

This field is used for virtual packages (see 'Virtual packages' on page 19) and when a package is renamed. It basically tells the package manager that this package provides another package. In the case of a virtual package the semantic is that this package provides a certain functionality that other packages may depend on.

In case of a real package, i.e. when another package was renamed, it means that the package manager may remove the other one if it also replaces it at the same time.

## 3.2 Package Managers

The packages on a Debian system are maintained by `dpkg` (see 'Introduction to `dpkg`' on page 14). This is the package maintenance system. There are several frontends to this system. The most commonly used and most sophisticated one is



`apt-get` (see ‘Introduction to apt-get’ on page 16). However, the older frontend `dselect` still exists and is in use by those who are used to it.

A more up-to-date text-based frontend to the package maintenance system is `aptitude` which is used to upgrade from one distribution to another these days. Those who prefer graphical programs can run Synaptic which enables you to install, upgrade and remove software packages in a user friendly way. Finally, there is also `kpackage` that can cope with Debian packages.

### 3.3 Installed Packages

There are two lists of installed packages available on Debian. The original file these lists are created from is `/var/lib/dpkg/status`. This file must not be corrupted, or otherwise your system is hosed. This is the main database for the package manager `dpkg`.

The Debian package system keeps an older copy from the last but one `dpkg` run in `/var/lib/dpkg/status-old`. In order to preserve the system for greater damage upon a crash or filesystem corrupting a daily backup of this file is created into `/var/backups` when the file differs from the last copy. The backup code is in `/etc/cron.daily/standard`.

The informal list is created by the command `dpkg -l` or by starting `dselect` and using the Select item. Or by using any of the other package manager frontends. This list consists of the name of the package (stripped down to some 40 characters), the installed version of each package and a short description. This is intended to be human readable.

For technical purpose you can make `dpkg` to generate a list of packages and their selection status (i.e. install, hold and deinstall). This is created by `dpkg --get-selections`. This output is intended to be parsed by a program again, such as `dpkg --set-selections` which will change the status of packages in its internal database. The `dselect-upgrade` action from `apt-get` will install all new packages afterwards.

### 3.4 Information about Packages

As mentioned above each package consists of installed files and metadata. To display a part of the package information you have to issue `dpkg -s package`. Among others you will see a description of the package as well as dependency information and information about configuration files used by this package.

The mentioned configuration files will not automatically be overwritten on an upgrade if you have modified them manually. You will be asked if you want to overwrite it or keep the old file. However, on an upgrade if the file was not modified and the package comes with a new version it will be overwritten.

### 3.5 Locate Files and Packages

One advantage of using package management to install packages is that it keeps information about files and packages somewhere on the local system. The benefit is that you can map any file on the local system to a package and display status information.

To find out which package contains a given file, issue the command `dpkg --search file` or `dpkg -S file`. If you are able to specify the absolute path, do that, if not `dpkg` will display all packages and files it has found.

Since this method only works for installed files and packages you’ll have to issue a different command when the file is not installed on your system or the package isn’t. This is useful especially if you are looking for a package that contains a command you only have about.

The Debian project provides a web-based search tool to look into their packages databases. On [debian.org](http://www.debian.org/distrib/packages) (<http://www.debian.org/distrib/packages>) you can search for package names and descriptions as well as for files within any package that is distributed by the Debian project.

Debian provides a file that contains a list of all files of the distribution. It is called `Contents-$arch.gz` and is found on their FTP-server in `/debian/dists/$dist/`. Since there are differences among different architectures (i386 for example has `lilo`, `sparc` has `silos` etc.) there is one file for each architecture.

If you want to use a current version of that file, there is no need to fetch it every day. Due to the time requirements for such a file to get regenerated (approximately one hour per file) these files are only generated once per week. Thus it is sufficient to update your copy once a week. New versions of packages often don’t imply a change in the Contents files, but only entirely new packages or splitted/merged packages. If you don’t need such an up-to-date copy, for the unstable distribution, it is usually sufficient to make a sporadic update or update that file once per week.

The Contents files are plain text files so you can simply use `grep` or `zgrep` respectively to locate a file or package in it. There is also a very simplistic frontend that makes use of this and is also able to update the file. Since the author uses it for the unstable distribution that distribution is hardcoded in it. Check out `findpkg` (<http://www.infodrom.org/Infodrom/tools/findpkg.html>).

## 3.6 New Packages

There exist a couple of ways to install new packages on a Debian system. First of all, the package should be in `.deb`-format. If it is not and it is a binary package, try using `alien` to convert it.

If you have already fetched a `.deb`-file you can install it with `dpkg --install file.deb` or `dpkg -i file.deb`. The package manager will then check all dependencies (see ‘Package Maintenance’ on page 9) and install the package if no problems occurred. You can add more packages to the commandline. If the dependencies are not fulfilled it will bail out.

If the package does not yet exist locally, you may want to use `apt-get install package`. APT will check if all dependencies are fulfilled and will download the other required package as well. When there are conflicts it will try to resolve them, which may result in the removal of other packages.

APT stores these files in `/var/cache/apt`, so if the connection goes down in the middle of the download, just issue the command again. If you add `-d` to the commandline, APT will also only download the packages but not install them. If you add `-s`, APT will simulate the installation and display some information.

Even if it may be a little bit confusing, to upgrade a package to the newest version in the Debian archive, you also have to use `apt-get install package`. APT will automatically use the most recent version available according to its internal database.

## 3.7 Package Database Update

Both `dpkg` and APT store information about available packages on the local system. Before you can access newer packages from Debian archives, you’ll have to update the internal database against updated sources. To update the internal database of available packages issue the command `apt-get update`.

All package sources are configured in the `sources.list` file (see ‘APT Sources’ on this page for details). After a new installation this file is initialised with proper sources, including the installation CD or DVD. When you don’t need this anymore, just comment out the respective line before executing `apt-get update`.

If you use a Debian archive provided on the Internet this will usually download several megabytes of index files. You will have to issue this command more frequently if you use the testing or unstable distribution instead of the stable one.

Subsequent calls of `apt-cache show` will show both the installed and the new version of a package if it has been updated in the meantime. All new installations of packages refer to the new database and hence will always install the most recent version of a package.

Even though you will normally not need to know the details, there may be situations in which they are helpful. APT saves the entire package index files it has downloaded from the Internet. They are placed in the directory `/var/lib/apt/lists`. This directory also contains descriptions and their translation into other languages (see ‘Translations’ on page 18).

## 3.8 APT Sources

Before giving an introduction into `apt-get` (8) (see ‘Introduction to apt-get’ on page 16) I want to provide some information about its sources first, since several frontend programs (like `dselect` and `capt`) are also using this information. You will need this information if you want to direct these programs into the proper archive direction.

Sources for APT refer to the file `/etc/apt/sources.list` which is a plain text file containing resources for apt. These resources point to a directory that contains `.deb` files and provides a `Packages` file. Usually you will use official Debian archives but you may also use unofficial archives. Each line can point to a binary cd, an HTTP- or FTP-resource or a local mirror.

For binary packages you should only use one of stable, testing and unstable. These three distributions represent sets of packages that Debian maintains as a set. As a regular user you should only use the stable distribution, which is the best tested and released distribution that Debian can provide. If you want to run bleeding-edge software or require to run most recent software, you may have to use unstable distribution. In that case please be warned that unstable may break things from time to time.

There are a couple of other non-official resources maintained by Debian developers or people who want to distribute non-official packages, e.g. nightly builds from CVS. Stephane Borzmeyer maintains a list (<http://www.internatif.org/bortzmeyer/debian/apt-sources/index-list.html>) of unofficial sources.

### 3.8.1 Accessing network servers

This is the most often used method for `apt-get`, accessing a server located somewhere on the internet and using the Debian archive there. You'll have to specify the URL until the `dists` directory occurs (here: `http://http.us.debian.org/debian`). The next argument is the distribution to be used, this is the path component until the section (`main`, `contrib` etc.) occurs (here: `stable`). Finally you have to specify all sections (here: `main`). See the example below.

```
deb http://http.us.debian.org/debian stable main
```

### 3.8.2 Accessing local directories

For local directories, not much has to be changed. The URL access method has to be changed to `file:`. The next component is the path on the local system until the `dists` directory occurs (here: `/mirror/debian`).

```
deb file:/mirror/debian stable main
```

### 3.8.3 Accessing cd-roms

CD-ROMs require special handling since they are not available all the time like ftp servers. Even worse it is quite unlikely that there is always the same cd-rom inserted in the drive. To cope with this situation `apt-cdrom(8)` was invented. When called with the `add` command it will umount the current cdrom and then ask the user to insert the Debian cd.

```
apt-cdrom add
```

The program will add proper lines to `/etc/apt/sources.list` so that later calls to `apt-get` will know about the availability of these cd-roms. `apt-cdrom` will take care of determining the structure of the cd as well as correcting for several possible mis-burns and verifying the index files. `apt-get` will not only add the cdrom to the list of APT resources but also scan them and add their index files to the internal database of apt. Thus, you must not create the resource lines by hand.

If you don't want to use the cdroms anymore (e.g. after installation is done and further updates should be used from the internet), you have to remove the respective lines from `/etc/apt/sources.list`. These lines look like the following:

```
deb cdrom:[Debian GNU/Linux 2.2 r0 _Potato_ - Official i386 Binary-2 (20000814)]
```

If you want to use a different set of cdroms, you have to remove these lines from `/etc/apt/sources.list` first and then call `apt-cdrom add` again. This will scan every new cdrom and add proper entries to the internal databases.

### 3.8.4 Configuration for stable

For the stable distribution (codenamed potato as of this writing) there should be lines in `/etc/apt/sources.list` like the following, given that you want to use the archives from the internet. If you also want to use non-free software, simply add "contrib non-free" to "main".

```
deb http://http.us.debian.org/debian stable main
```

### 3.8.5 Configuration for unstable

For the unstable or testing distribution the lines look similar:

```
deb http://http.us.debian.org/debian unstable main
```

## 3.9 Package Difference Files

Beginning with Debian 4.0 (codename etch) APT support package diff files and defaults to downloading them instead of full index files. They contain only the differences between two versions of the package index file `Packages`. When updating frequently this will save a lot of bandwidth since the differences are a lot smaller than the entire files.

The default setting is to download only differences of package index files instead of the entire file. This behaviour can be altered which may be useful when the administrator doesn't update too frequently or when there is a Debian mirror in the local network. To switch back to downloading the entire index files add the following to `/etc/apt/apt.conf`.

```
Acquire::Pdiffs "false";
```

If you would like to adjust this setting only temporarily, you can do so on the commandline as well without having to edit a configuration file. `apt-get` supports configuration on the commandline with the `-o` argument. The following command will update the internal database against the one provided by the Debian project on the Internet and temporarily disable fetching diff files:

```
apt-get update -o Acquire::Pdiffs=false
```

## 3.10 Introduction to dpkg

The main package manager on a Debian system is `dpkg` (1). It handles installation, upgrade, configuration and removal of packages as well as dependency handling. These days it is not called directly normally but through a frontend like `apt-get`, `dselect` or `capt`.

Nevertheless it is always good to understand that the actual work behind the scenes is done by `dpkg` and that you may always call it manually if you should run into trouble the frontend cannot cope with. After the most important commands you'll find a list of important options to add. For a comprehensive list of arguments, please call `dpkg --help`

### 3.10.1 dpkg --install

Add an arbitrary number of package archives (i.e. `.deb`-files) to the commandline. `dpkg` will check their dependency information and install the packages if there are no problems. When there are problems, you will have to fix them manually. This command may be abbreviated with `-i`.

### 3.10.2 dpkg --configure

With this command you are able to configure packages that failed to configure in the first stage. This usually only happens when you upgrade to the current unstable distribution and install broken packages that don't configure (i.e. whose preinst/postinst scripts produced a failure). `dpkg` will retry to configure the specified packages. If `-a` is supplied instead of a list of packages `dpkg` will check its database for any unconfigured packages and try to configure them.

### 3.10.3 dpkg --list

Add an arbitrary number of package names to the commandline. The program will display a short status for each package together with their name, version and short description. If no packages were provided `dpkg` will display information about all installed packages. This command may be abbreviated with `-l`. The output looks like:

```
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name          Version          Description
+++-----+-----+-----+
ii modconf         0.2.27           Device Driver Configuration
rc chimera         1.70p0-1         X11 World-Wide Web Client
hi apache         1.3.9-13.1       Versatile, high-performance HTTP server
```

The first three characters represent the status of the package in question. The most common states are:

- `ii` - The package is installed and will be upgraded if possible.
- `rc` - The package is removed, won't get reinstalled but still has its configuration files installed.
- `hi` - The package is installed and set on hold, thus won't be upgraded.

### 3.10.4 `dpkg --status`

Add an arbitrary number of package names to the commandline. `dpkg` will display the status and description for each supplied package. The output also contains information about registered configuration files for that package. You may want to use this information if you want to change the configuration of a package. This command may be abbreviated with `-s`. If you don't need information referring to the local installation of the package but only want to read meta data you may use `apt-cache show` instead since it is faster.

### 3.10.5 `dpkg --search`

Expand the commandline with a file or directory substring. `dpkg` will search in its database for that string and display all files that contain this substring together with the package which contains the file. This command may be abbreviated with `-S`.

### 3.10.6 `dpkg --getfiles`

Using this command you'll see which files are contained in a particular package. The output is the same as generated by `dpkg-query -f='${Package} ${File}\n'`. This command may be abbreviated with `-L`.

### 3.10.7 `dpkg --remove`

Add an arbitrary number of packages that you want to remove from your system. `dpkg` will only remove these packages that don't leave packages with broken dependencies. Thus if you want to remove a library another package depends on, you'll have to remove the other package at the same time or before. This command may be abbreviated with `-r`. If you have to remove a package regardless of other packages with broken dependencies, please continue and check the option `--force-depends`.

Please keep in mind that removing a package does only mean that binaries and variable files are removed but no configuration files from that package. If you want to remove the package entirely, i.e. including configuration files please read on and use `--purge`.

### 3.10.8 `dpkg --purge`

This command will remove all packages specified on the commandline together with their configuration files and meta data. `dpkg` will only purge the packages that won't result in other packages with broken dependencies. This command may be abbreviated with `-P`.

### 3.10.9 Option `--force-overwrite`

For the released stable distribution this flag is turned on by default, though it's turned off for the unstable distribution. If not turned on `dpkg` will break if a package is to be installed that will overwrite a file from another package while the new package does not officially replace the other package (by using the `Replaces:` information).

For the unstable distribution this is likely to happen for all packages that have to interact with other packages that contain the same files.

### 3.10.10 Option `--force-depends`

This option must not be used if you don't know what will happen. It is only documented here because it may be useful to cope with unstable and broken packages. You should not require its use on a system that runs the stable distribution.

If this option is used, `dpkg` will turn off its dependency check. This will most probably result in broken packages or packages whose dependencies are not satisfied, though some components may still work. This may be the only way to remove a broken package in order to install a fixed package.

When you force to install or remove a package with this option, `apt-get` will most probably not work anymore until you have fixed the problem.

### 3.10.11 Option `-force-conflicts`

You should never have to use this option. It will bypass the conflict check of `dpkg` and will enable you to install two or more conflicting packages. This is most probably not what you want to achieve.

## 3.11 Introduction to `apt-get`

The program `apt-get` (8) provides the most convenient interface for upgrading and installing packages. For people who don't find `dselect` (8) too much userfriendly and don't mind typing a short command in a shell, `apt-get` is most probably the best choice for their package maintenance.

The program maintains its own set of index files that are stored in a binary way and accessed using optimized methods. Thus accessing them works faster than similar messages from `dpkg`. However, it also means that if you are switching from `apt-get` to `dselect` you may have to re-transfer the index files again. If you are running the unstable distribution the index files will change about daily (except when our main archive server crashes and no new packages were processed) so you'll have to retransfer them before any other command as well.

Even if `apt-get` will perform important tasks, it is only a frontend to `dpkg`. Actual installation and upgrading of packages will still be performed by `dpkg`. However, `apt-get` is able to fetch files from the net, perform dependency checks and call `dpkg` with the correct order of packages.

If you notice that `apt-get` isn't able to finish downloads you can increase the number of retries that it should perform before it finally gives up. This is done by adding `APT::Acquire::Retries=20` to `/etc/apt/apt.conf`.

If you want to see the urls for packages that `apt-get` would download, you'll have to add `-y --print-uris` to the commandline (`-y` will bypass the first question). You can easily parse the output and create an input file for `wget` (1) or similar.

Please read all subsequent sections covering `apt-get` carefully before working with the program.

### 3.11.1 `apt-get update`

This command will upgrade the index files for all targets listed in `/etc/apt/sources.list`. Before actually downloading any file from the network it will check if it is newer than the copy already present if any. This will save you some bandwidth for less frequently updated archives.

On a host with good network connection on which the unstable distribution is running you may want to put that call into the `crontab` (5) for the user root.

### 3.11.2 `apt-get install`

This command will install all packages provided on the commandline, given that they don't produce a conflict in some way. When these packages depend on others that are not yet installed, `apt-get` will download and install them as well. If that causes conflicts old packages will probably be deconfigured so everything will work well.

If files actually need to be downloaded then `apt-get` will display the amount of data that have to be downloaded and asks for the users permission. If the connection to the internet breaks down in the middle of the download session, just restart it, `apt-get` will cope with it properly. If you only want to download files but not install them, use the `-d` switch. All packages will be downloaded into the directory `/var/cache/apt/archives`.

If you specify packages on the commandline that are already installed, `apt-get` will upgrade this package to the most recent version found in all the. So if you only want to upgrade a package, use the `install` command.

If you want to reinstall a package that is already installed in the most recent version, please add `--reinstall` to the commandline. `apt-get` will then fetch the most recent version regardlessly of the installed one.

### 3.11.3 `apt-get upgrade`

With this command you will upgrade the entire system to the next revision. It will upgrade packages one by one if a newer version is available. This should only be used if you stay with the same distribution (i.e. stable, testing, slink, potato etc.). This method is used for security updates as well.

The Debian Project provides updates to a released distribution. These are called revisions and will overwrite the former release. Say potato was released in 2000 as 2.2r0 (implicit revision 0) the next revision would be 2.2r1. Updates to once released distributions will only contain security updates and eventually very few other updates like fixed packages if they were extremely broken. Security updates from security.debian.org (<http://security.debian.org/dists/>) will be pushed into these updates.

Thus if you upgrade to the next release of a stable Debian distribution it is quite unlikely that you would download large amounts of packages or data. Though, if there were a lot of security updates, it can still result in a couple of megabytes.

Since the unstable distribution is target to a lot of changes and package reorganizations it is quite likely that a normal upgrade will not result in what you expect. Please consider running the `dist-upgrade` command instead.

### 3.11.4 apt-get dist-upgrade

This command will upgrade the entire system. In addition to what `apt-get` will do for a regular upgrade it will handle changed dependencies etc. in an intelligent way so that the upgrade is as smooth as it can be. This should be used if you don't updated the unstable distribution on a daily basis, if you are upgrading from stable to unstable or if you are upgrading from one stable version to the next one (for revisions upgrade is sufficient).

`apt-get` has a "smart" conflict resolution systems that helps with upgrading the distribution. It will upgrade the most important packages first at the expense of less important ones if necessary. If you want to upgrade the packages as well that were left out you can run `apt-get upgrade` afterwards. This should upgrade all other packages for which a newer version is available and that don't produce a conflict.

## 3.12 Configuring APT

The main configuration of APT refers to the `sources.list` file in `/etc/apt`. This file contains a list of mostly network-reachable package sources to use. This is the basis for all package installations and the information displayed by the `apt-cache` command.

The behaviour of APT can be tweaked in many other ways as well. The behaviour is configured with the `apt.conf` in the `/etc/apt` directory or with files in the `apt.conf.d` subdirectory in the same directory respectively. When the configuration is split among different files the filenames have to start with two digits.

The format of these configuration files is modelled after a format the Internet Software Consortium (<http://www.isc.org/>) uses for their tools such as BIND and DHCP daemon. All settings are organised in groups and partially even sub-groups. Lines starting with two slashes are considered comments and, hence, ignored. Each block needs to be terminated with a semicolon.

The general format for a configuration option is a key-value pair:

```
Option "value";  
Acquire::Pdiffs "false";
```

However, since options are grouped and since the APT configuration format allows setting a scope for included options, the configuration files can be written in a nicer format. In the example above the `Option Pdiffs` is part of the `Acquire` group. In the single-line format options, groups and sub-groups are delimited by double colons. In the multi-line format groups and sub-groups embrace options by using curly brackets as shown in this example:

```
Acquire {  
    Pdiffs "false";  
    Retries "0";  
};
```

This can help keep the configuration file readable and easier to read since options that belong to each other are naturally grouped together.

The general rules for configuring APT is explained in the manpage `apt.conf(5)`. However, only very few options are listed there. In `/usr/share/doc/apt/examples` you'll find all possible configuration options with valid settings in the file `configure-index.gz`.



### 3.12.1 Translations

Package descriptions and their translations are stored in separate files in order to keep Packages files small. Translations also have their source entirely outside of the package they describe. Since they are stored in a different file `apt-get` needs to be configured which language files to load during package database updates.

You can configure multiple languages, or just the one you would like to support. Later on programs decide on the environment variable `LC_MESSAGES` which language to display. The following configuration will tell `apt-get` to load at least the English and German translations from the Debian server.

```
Acquire::Languages { "environment"; "en"; de"; };
```

If you don't want `apt-get` to load any languages and thus risk `apt-cache` to display any long descriptions use the following configuration. This is also the default setting if you upgrade an old Debian system to current distributions.

```
Acquire::Languages { "none"; };
```

## 3.13 Brandnew Packages

Packages uploaded into the Debian archive by a Debian developer will only be synchronised into the official archive once a day (this will be changed to twice per day in the future). Due to this limitation there may be a 30 hour delay until a recently uploaded package hits the mirror you use.

Furthermore the Debian project distinguishes between updated and new packages. Updated packages refer to updated versions of packages that already exist in the particular suite. They will be processed by the archive software automatically. New packages, however, do not yet exist in the particular suite and require manual attention.

An ftpmaster needs to inspect the new package and see if it is suitable for the Debian archive. If the origin is a source package which is already present in the archive only the internal database needs to be update to reflect the section and priority of the new package.

If the package originates in a source package which is also not yet present in the archive, more work needs to be done. An ftpmaster will have to inspect the package and decide whether it is suited for the Debian archive. They have to read the license carefully and reject the package if it does not conform to the Debian Free Software Guidelines ([http://www.debian.org/social\\_contract#guidelines](http://www.debian.org/social_contract#guidelines)).

Furthermore the Debian project is allowed to distribute cryptographic software. It is even allowed to export cryptographic software from the U.S. if it is Free Software. The U.S. government considers cryptographic software as weapons and hence usually do not permit to export it.

However, this requires the Debian project to announce (<http://www.debian.org/legal/cryptoinmain>) all software that may be subject to cryptography to the U.S. Bureau of Export Administration (BXA). In order to comply to this regulation, all new packages are reported to the BXA with the note that they may contain or utilise cryptographic software.

This, unfortunately, requires the Debian project to not export such software before the BXA has been informed. Hence, the new queue of the incoming directory cannot be made public anymore. They will have to be kept private unless they have been properly processed, in which case they'll appear in the Debian archive at last one day later.

Packages sitting in the incoming directory will be added to the public package archive with the next archive software run. They can be fetched from the incoming directory (<http://incoming.debian.org/>) directly when immediate updates may be required. This is a direct gateway to the accepted directory in which updated packages and processed new packages are stored until the archive maintenance run.

## 3.14 Packages not in Debian

The amount of Free Software is sheer endless. Today a lot of projects and individual produce very good applications and utilities that are distributed under a Free Software license. Naturally, not all such packages are part of the distribution, not even in the case of Debian which already refers to the largest archive of integrated packages.

If you are not using the `unstable` distribution chances are that the developers have added the missing package to the archive already but it is simply not yet available in the `stable` or `testing` distribution. In that case it may be possible that you find a backport (<http://www.backports.org/>) of the package.



When the distribution you are using is not too far away from the `unstable` distribution with regards to libraries and stuff, it may be possible that you can install the package from `unstable`. You'll find a download link via [packages.debian.org](http://packages.debian.org) (<http://packages.debian.org/>). If this doesn't work, it may be possible to recompile the source on your system and install the resulting package.

Please note that neither of this is recommended by the project and that all errors that may occur lay entirely in your responsibility. The package maintainer may be a source for help but they are not required to support their package on distributions it is not designed for.

A list of unofficial Debian archives is [www.apt-get.org](http://www.apt-get.org) (<http://www.apt-get.org/>). Packages from one of the listed sources are not supported by the Debian project but only by their respective maintainers. You may, however, find software that is not yet packaged for Debian because the integration is difficult or large parts of code are duplicated.

If you are looking for a particular package that is not even part of the `unstable` distribution and can't find it in the unofficial sources mentioned above, you may have success on [Fresmeat.net](http://freshmeat.net) (<http://freshmeat.net/>) or [rpmfind.com](http://www.rpmfind.com) (<http://www.rpmfind.com/>).

### 3.15 Virtual packages

A virtual package does not really exist in the package universe of Debian. It is a name other packages can depend on if they depend on a certain functionality. Virtual packages are used when several packages provide the same functionality that other packages need. The name of the virtual package often refers to the functionality and not to a common package name.

Instead of declaring a dependency against five distinct packages it is sufficient to only declare a single dependency against the virtual package which is provided by all of these five packages.

In order to help the dependency resolver often a dependency is declared on the virtual package and one of the packages providing it. This way the resolver will prefer one particular package if none of them is installed yet.

A common example is `mail-transport-agent` which is provided by Postfix, Sendmail, Smail, Exim, Nullmailer, ssmtp and other packages that provide `/usr/sbin/sendmail` as a means to accept and deliver mail.

The list of virtual packages is documented on the Debian website (<http://www.debian.org/doc/packaging-manuals/virtual-package-names-list.txt>).

### 3.16 Pseudo packages

Unlike virtual packages pseudo packages really don't exist. They are used so that people can assign bug reports to certain parts of the distribution or the Debian project that doesn't originate in a source package. Hence, they don't refer to a source or binary package.

This way it is possible to assign bug reports to [www.debian.org](http://www.debian.org) for example. The entire website does not originate in a particular package but is maintained by webmasters who are improving the website and fixing bugs as well. Instead of contacting them directly, users may rather open a bug report against this pseudo package.

Another example would be the `installation-reports` pseudo package which is used to collect reports of new installations. These are used by the `debian-installer` (<http://www.debian.org/devel/debian-installer/>) team to check with new installations and discover bugs that need to be fixed.

The list of pseudo packages is documented on the Debian website (<http://www.debian.org/Bugs/pseudo-packages>).

### 3.17 Package Pools and Distributions

The Debian archive is organised in distributions (`stable`, `testing` and `unstable`). They refer to a directory below the `dists` directory in the archive top. In these directories the index files for source and binary packages of the particular distributions reside.

Starting with `sarge` (Debian GNU/Linux 3.1) the package files itself are stored in a different directory. Past distributions kept the source and binary packages within these directories as well. Today they only contain index files.

Instead the packages itself are stored in the `pool` directory in the archive top directory. All current versions and architectures are stored in the same directory. The index files mentioned above refer to files within this directory, so that APT is still able to download packages.

In the pool directory the package files are stored in a special directory named after the source package. This directory is placed in a directory consisting of the first letter (or `lib` in the case of libraries) in order to reduce the number of sub-directories and to reduce the delay when reading the directory.

## 3.18 The testing Distribution

This distribution has been introduced in order to help prepare the next `stable` distribution. The `testing` distribution should be releasable all the time. Packages in this distribution should work together fine and all dependencies should be resolved. Additionally, all architectures should always be in sync.

Packages uploaded into the `unstable` distribution will migrate into the `testing` distribution automatically if they have been proven stable for a number of days, all architectures are in sync and all dependencies are fulfilled in the `testing` distribution itself.

The quarantine time for a package with urgency `low` is 10 days, with urgency `medium` 5 days and with urgency `high` two days. No release critical (<http://bugs.debian.org/release-critical/>) bug must be reported during this time.

This distribution should always be in a state in which it could be frozen and declared stable. Packages in this distribution work sufficiently well and don't incorporate the problems of the `unstable` distribution. However, there is no security support available. Security updates will migrate into `testing` the same way as normal packages, however, they usually declare an urgency of `high`.

## 3.19 Forcing Package Installation

When you try to install packages from `unstable` on a `stable` distribution or try to install packages that conflict, `dpkg` will spit out error messages refusing to install these packages. This is good, since it will keep the user from destroying his system.

However, for advanced users or developers there are situations where `dpkg` is wrong and the package should be installed anyway. In that case, please call `dpkg --force-help` to find out about its interface to force things. The following is only a short list with some explanation, `dpkg` is capable of more.

- `--force-overwrite` forces `dpkg` to overwrite files that are already installed by a different packages.
- `--force-depends` forces `dpkg` to install or remove packages even if this causes some dependencies not to be fulfilled.
- `--force-conflicts` forces `dpkg` to install packages even if they conflict.

These parameters may only be used in rare exceptional cases and must not be used as default setting since they will harm the system in most cases. Only advanced users may use them if they know how the system behaves and what their use will result in.

## 3.20 Creating Local Archives

This section is most probably only useful for experienced users who need to maintain additional resources for `apt`. This may be useful if you want to provide a repository containing current versions of a software product that is not (yet) part of Debian but consist of more than one binary package (e.g. KDE, OpenOffice, GNOME etc.) or if you want to use Debian for your corporate network and need a locally maintained archive for updates.

The programs to generate your `Packages` and `Sources` files are `dpkg-scanpackages` (8) and `dpkg-scansources` (8) respectively. Both are included in the `dpkg-dev` package.

Whenever you consider providing your own archive please read the manpage for `dpkg-scanpackages` (8) carefully. Lets assume that you wish to provide an archive that will be covered by the following line for `sources.list`

```
deb http://debian.domain.org/debian unstable foo
```

and `DocumentRoot` of your webserver is `/var/www/`, consider the following directory structure

```

/var/www/debian
  ../dists
    ../unstable
      ../foo/binary-arch/
      ../foo/binary-all/
      ../foo/source/

```

Where `arch` is a valid architecture string (like `i386`, `ia32`, `m68k` etc.). By comparing the line for `apt-get` with the directory structure you should easily be able to support other architectures, other distributions and other sections by simply adding more directory trees to the proper root.

Both programs, `dpkg-scanpackages` and `dpkg-scansources`, will descend into subdirectories and handle them properly. That's the way the main debian archive works by using subsections (like `mail`, `base` etc.). Both programs will also follow symlinks which means that `binary-all` programs may indeed be symlinks to `../binary-all/program.deb`. This feature is also used on the main debian archive.

In order to generate the `Packages` file you have to issue the following two commands. It is important to change the directory to the main Debian directory first so all paths will be adjusted properly.

```

cd /var/www/debian
dpkg-scanpackages dists/unstable/foo/binary-arch /dev/null \
> dists/unstable/foo/binary-arch/Packages

```

The component `/dev/null` could be an override file (which you could fetch from [debian.org](http://ftp.debian.org/debian/indices/) ([ftp://ftp.debian.org/debian/indices/](http://ftp.debian.org/debian/indices/))) you probably don't have, thus it is ignored that way.

The above paragraphs discussed the setup for the large archives which is used at [debian.org](http://debian.org) as well. If you only plan to provide a small archive and only support one architecture you can probably skip large chunks of the above. Assume you want to provide the following line for `sources.list`:

```
deb http://debian.domain.org/debian foo/
```

If you don't forget to write the trailing slash, `apt-get` will expect the package file in the directory `/var/www/debian/foo`. As a result of this, you can place all `.deb` files in that directory directly without having to care about general directory structures. Creating the `Packages` file is easier as well:

```

cd /var/www/debian
dpkg-scanpackages foo /dev/null > foo/Packages

```

Please be warned that these scan programs may not handle multiple versions of packages properly. Thus if there are two versions of the same package it may be possible that the newer version doesn't appear in the `Packages` files but only older ones. You'd better try to avoid that situation.

## 3.21 Upgrading single Packages

To upgrade a single package, fetch the package and install it with `dpkg -i file.deb` or perform both steps with `apt-get install file`, however this requires a properly set up `sources.list` file.

When you want to install files from testing or unstable on a system that reflects the stable distribution of Debian, you may run into dependency problems. This will happen especially if you are trying to update some major or large package (like `sql`, `xfree`, `gnome`, `kde` etc.). They will probably require a newer `glibc`, newer `xlibs` etc. Due to this a regular upgrade of everything may be a good idea.

## 3.22 Upgrading everything

To upgrade from one distribution (e.g. `stable`) to a newer revision of the same distribution you only need to issue two commands. This is needed when you want to incorporate all security updates from your last upgrade or your first installation, or when the Debian project has released a new revision of the stable distribution. Your `sources.list` (see 'APT Sources' on page 12 for details) would look like this:

```

deb http://ftp.debian.org/debian stable main
deb http://security.debian.org stable/updates main

```

If you also want to use non-free parts that were packaged for Debian, you'll have to add `contrib non-free` after the word `main`. After that the following commands will do everything needed.

```
apt-get update
apt-get upgrade
```

The first command will fetch new package databases from the archive (see 'Package Database Update' on page 12 for details) and the latter will actually fetch packages and install them. If you only want to fetch them but not install them please add the `-d` switch to the second commandline. If your Internet connection goes down during the update, re-issue the command and `apt-get` will restart the process.

## 3.23 Distribution-Upgrade

Every once in a while even the Debian project releases a new `stable` distribution. These releases usually have accumulated quite a number of new and updated packages. Many libraries are available in a newer version and security support for the old stable release will end in one year after the new release.

Because many libraries have been modified and several new libraries have been introduced, simply upgrading every single package doesn't work. Instead the package manager (see 'Package Managers' on page 10) needs to calculate all dependencies and decide which package to update and which new ones to install. The installed packages also need to be grouped in small sets which also need to be calculated.

As a general rule you should always check the release notes (<http://www.debian.org/releases/stable/releasenotes>) before starting the upgrade. You should also backup all of your configuration and all important data that may suffer from the upgrade. It will mention those packages that need to be updated before the rest of the system when problems are anticipated otherwise.

If you have enough disk space on one of your systems you could also test the upgrade before actually performing it. The package `dchroot` (<http://packages.debian.org/dchroot>) allows you to switch into a chroot environment. You could clone your production system into another directory, jump into it and perform the distribution upgrade inside the cloned system. Without risking your production system you could test the upgrade and develop solutions to problems in case you should experience some.

The same applies to you if you wish to upgrade from a `stable` or `testing` distribution to `unstable`.

### 3.23.1 Upgrade with apt-get

A distribution upgrade usually works with `apt-get dist-upgrade` after the package database has been updated (see 'Package Database Update' on page 12). However, due to the complexity of the Debian distribution it is possible that APT can't cope with the dependencies as it should and will offer to remove several packages.

When this happens it is often helpful to manually upgrade certain parts of the distribution. This is done by executing `apt-get install` on some of the packages that should be removed otherwise. This will ensure that they are upgraded together with their dependencies.

Candidates for such particular upgrades are `dpkg`, `apt`, `libc6`, `apache`, several libraries, several GNOME and KDE packages. Afterwards a regular `dist-upgrade` should work again.

### 3.23.2 Upgrade with aptitude

When you choose to upgrade your distribution via `aptitude` (<http://packages.debian.org/aptitude>) you should upgrade this package manager first, either with `apt-get` or with `aptitude` itself.

Afterwards the regular upgrade of the entire system should work flawlessly. Execute `aptitude -f --with-recommends dist-upgrade` to perform the upgrade. If too many packages are to be removed, please upgrade some packages manually just as with `apt-get`.

## 3.24 Upgrading to unstable

If you plan to upgrade to the `unstable` distribution please remember the exact meaning of the word "unstable". This distribution is the current development target for all Debian developers. Packages and dependencies are expected to break from time to time.

For example, Perl upgrades are known to break everything. GNOME and KDE upgrades usually cause problems as well. Libraries that provide architecture independent packages together with architecture dependent packages will cause a problem on those architectures for which it hasn't been compiled yet.

The Debian project appreciates everybody who is using `unstable` (or `testing` for that matter) and reports bugs to the bug tracking system (<http://www.debian.org/Bugs/>). However, please don't complain because things are broken from time to time as this is to be expected on `unstable`.

Please note that the Debian project does not provide security updates for the unstable distribution. This suite is not supported by the security team. Instead, maintainers will upload security fixes on their own responsibility. Security update may be unavailable for the `unstable` distribution for all or certain architectures for a while.

In order to upgrade your system you will need to edit the lines in the `sources.list` file (see 'APT Sources' on page 12) to point to the unstable distribution like the following:

```
deb http://ftp.debian.org/debian unstable main
```

If you also want to use non-free parts that were packaged for Debian, please add `contrib non-free` after the word `main`. After that you continue as if you are performing a distribution upgrade as described in 'Distribution-Upgrade' on the facing page.

Nothing should go wrong now. However, `unstable` is called so for a reason - it is not stable and things may break. Thus, it is possible that the upgrade will die in the middle. Even though this may look confusing it is not a big problem. All you need to do is to configure all installed but not yet configured packages and restart the upgrade afterwards. This is done by the following commands:

```
dpkg --configure --pending
apt-get dist-upgrade
```



## Chapter 4

# Alternatives

Debian has developed a mechanism that will provide you with an interface to install several packages that provide the same functionality. For example take the program `vi`. There is not a single `vi` package but half a dozen of them. All can be installed to provide `/usr/bin/vi`, the admin decides which of them will actually provide this common interface.

This is achieved by making `/usr/bin/vi` a symbolic link to `/etc/alternatives/vi`. That file again is a link, which gets automatically created, to the real binary. Along with the program there is a so called slave-link for the manpage.

This mechanism is controlled by the `update-alternatives` program. It creates, removes, maintains and displays information about the symbolic links comprising the Debian alternatives system. For each major link (contrary to a slave-link) can be configured with a priority which will control which package actually provides the second link.

### 4.1 Display Alternatives

to find out which alternatives exist on your system, you have to list the directory contents of `/etc/alternatives`. You'll find a lot of links in there. Some of them are slave links that you can ignore for the following actions. Slave links are often manpages, thus the links `foo.1.gz`.

If you know the link name you can issue a direct query to the system by using `update-alternative --display vi`. The name `vi` is just a common example. The program will display all affected packages and also display the chosen one.

### 4.2 Configure Alternative

The program `update-alternative` provides a simple but sufficient configuration frontend. Simply call it as `update-alternatives --config foo` where `foo` is a major link from `/etc/alternatives`.

### 4.3 Add Alternative

A new alternative is set with `--install` as argument to `update-alternatives`. Each alternative has a priority that controls the creation of the link. The higher the priority the more probably this alternative will be used. Since you can change an alternative by re-installing it you can also change the priority at the same time. As an example, the following code forces `vi` to be a link to `vim` (using priority 900). For details please refer to the manpage for that program.

```
update-alternatives --install /usr/bin/vi vi /usr/bin/vim 900 \  
--slave /usr/share/man/vi.1.gz vi.1.gz /usr/share/man/vim.1.gz
```

### 4.4 Remove Alternative

An alternative is removed with:

```
update-alternatives --remove vi /usr/bin/vim
```





## Chapter 5

# System Administration

### 5.1 System cloning

After a crash or when a second system should be set up as a backup server, it is helpful to simply clone the first system into the second one instead of manually installing everything. This is actually possible and can be achieved without much work. All you need as additional software is `dselect` even though you may have considered it archane.

In section ‘Installed Packages’ on page 11 you have already been told how to dump the status of all packages known to the system into a file. This file needs to be transferred to the second machine and imported there via `dpkg --set-selections`.

Afterwards start `dselect` and let it update its internal database. When this is done, hit the install target and let `dselect` finish the job. It will remove those packages that were marked to be removed and install those that were marked to be installed. Alternatively, use the `dselect-upgrade` mechanism of `apt-get`.

### 5.2 Keyboard

This section only covers the keyboard layout for (virtual) text consoles. The configuration for X11 is covered in ‘Keyboard Layout’ on page 7. The keyboard translation table is controlled by the program `loadkeys(1)`. As parameter for that program you supply the keymap you want to load. All available keymaps are stored in `/usr/share/keymaps`. Upon system boot the file `/etc/kbd/default.kmap.gz` is loaded.

The `default.kmap.gz` file is a compiled and compressed keymap, generated from `/usr/share/keymap` with all `#include` statements expanded. This is configured by the program `kbdconfig(8)` which you can run as root at any time.

It may be useful when using a console at a friend who uses a different keyboard layout than you, or when he needs to type on yours, to switch the keyboard layout on the running system. For example `loadkeys us` will switch to a US layout while `loadkeys de` will switch to the German layout. These changes have impact to all virtual consoles but not for X11.

It may be valuable to remap some keys (e.g. use the Window-Key for something useful). For doing this you will have to edit the keymap yourself. Since they are plain text files (though compressed) this is done with any editor. For finding out which keycode is generated for which keypress, use the `showkey(1)` program. The program runs until 10 seconds (or the amount of time specified by the `-timeout` or `-t` option) has elapsed since the last key press or release event, or until it receives a suitable signal, like `SIGTERM`, from another process.

If you want to modify an existing keymap and use it under a different name, you may want to use the `dumpkeys(1)` program which writes the current keyboard translation table to stdout.

Please be warned not to use a keymap which was designed for a different architecture. Using a keymap designed for sun on an intel compatible system will lead to very strange keyboard mappings.

### 5.3 Time and Timezone

Upon system installation you are asked in which timezone your computer is and if it uses GMT for the system clock. This chapter informs you how the time is calculated and how you can adjust or change that.

### 5.3.1 Reconfiguring the Timezone

The program `tzconfig(8)` is used to configure the timezone. You set a continent and capitol city which is next to your location. From this information the actual timezone is calculated. For example, if you are living in Germany your timezone will most probably be `Europe/Berlin`. This information is stored in `/etc/timezone`.

### 5.3.2 System Clock

Your system clock does not need to run with the same time that is displayed when issuing the `date` command. The time is calculated from it at boot time though. Only upon system boot the Linux clock is adjusted from the hardware clock. On a Debian system it is done the other way around when the system is turned off. This is controlled by the start/stop-script `/etc/init.d/hwclock.sh`.

Your system clock may run on UTC (Universal Coordinated Time also known as Greenwich Mean Time (GMT)) or `localtime`. This is configured in `/etc/default/rcS`. When UTC is set to `yes` this tells the boot script that the system clock runs on UTC and time has to be converted into local time when the time is read or written.

### 5.3.3 Automatic Clock Adjustments

There are techniques to adjust the Linux time on a regular basis. This may be important if your clock runs too unreliable but requires some sort of permanent connection to the internet. Adjusting the time is done by synchronizing it with another host which is said to contain a reliable time.

The most common technique is to participate from timeservers that are located in various places around the world and that synchronize themselves either with a radio receiver or by using the Network Time Protocol (NTP). To participate in this network all you have to do is to install the `ntp`-package and to tell it to use one or more public NTP-servers (<http://www.eecis.udel.edu/~mills/ntp/servers.htm>).

## 5.4 Systemwide environment configuration

Sometimes it is needed to set some environment variables that should be used by all applications. It is a general rule to set as few variables as possible since this data is inherited from process to process and may waste memory. However, some variables are needed for localisation support (see below), for example. But make sure that the changes are really wished by all users before doing them!

There is NO DEFAULT WAY of setting the variables in Debian, but you may use the `/etc/environment` file that can be parsed in scripts that use bourne compatible shells (like `bash`).

```
# /etc/environment
# EXAMPLE ONLY
LANG=de_DE
LC_MESSAGES=en_US
```

Now, this file may be parsed by `/etc/profile` adding following line there:

```
. /etc/environment
```

## 5.5 Languages (locales)

Many non-native english speakers like to use their native language for program messages where possible. This is configured by the `LANG` environment variable and by other settings from `locale(7)`. The described variables may be set from each user for his environment (i.e. in `~/.bash_profile` if using `bash`) or systemwide as described above.

The rules of setting the locale variables in few words:

If nothing is set before, then setting of “`LANG`” makes `LC_*` (except of `LC_ALL`) adopt the value of “`LANG`”. If a `LC` environment variable is set before, it will keep it’s setting indepent of `LANG`.

If you set `LC_ALL`, it will allways overwrite the content of all locale variables, except of `LANG`.

So if you wish to only set certain variables (e.g. `LC_CTYPE`), set them first, then `LANG` (if wished) but don’t set `LC_ALL`.

Another example: if you wish to set everything to your country's settings, change LANG first, and then (if wished) change certain LC\_ variables.

When using current unstable (after 2.2 aka potato) you'll have to prepare the locales you want to use as well. The file `/etc/locale.gen` contains all possible locales that you can use on your system. You will have to uncomment all of them that will actually be used (or all if you are insane). After that you'll have to run the program `locale-gen`.

## 5.6 Character sets (foreign characters)

**Disclaimer:** This section lacks information. This is due to the fact that the author doesn't make heavy use of foreign characters on his machines. Help is appreciated.

You have to distinguish between input of foreign characters and proper output of them. In general, locale variables like LANG and LC\_ALL should be sufficient. Unfortunately they are not and you may have to tweak a couple of programs in order to make them work properly with non-english characters.

### 5.6.1 Readline

The readline library is responsible for interaction with programs like `bash`, `ftp` and `psql`. With recent versions of the library you don't need to make the changes described below if you have set LC\_CTYPE correctly (see locales part above).

However, with potato or a prior release of Debian, you will need additional configuration. There are two files where you may configure this: `/etc/inputrc` (system-wide) and `~/.inputrc` (per user). The following configuration should enable you to type foreign characters as well as making programs output them properly.

```
set meta-flag on
set convert-meta off
set input-meta on
set output-meta on
```

### 5.6.2 less

Less usually doesn't display foreign characters. You have to tell this program which character set to use. This can be adjusted with the environment variable LESSCHARSET. It is set in either `/etc/profile`, `~/.bash_profile` or `~/.bashrc` for a bourne shell (i.e. `bash`):

```
export LESSCHARSET=latin1
```

If you use a C-shell the relevant files are `/etc/csh.login`, `/etc/csh.cshrc` and `~/.cshrc` that have to contain the following:

```
setenv LESSCHARSET latin1
```

### 5.6.3 Mutt

With LANG=de\_DE Mutt should not only display german messages but also display german characters (setup locales as described above). If you don't like german messages but want to see german characters (i.e. umlauts) you'll have to set the following:

```
export LC_CTYPE=en_US.ISO-8859-1
```

If this is only required for Mutt you can use the following respectively:

```
alias mutt='LC_CTYPE=en_US.ISO-8859-1 mutt'
```

### 5.6.4 Emacs

Emacs normally displays non-english characters by their octal representation normally. To surpress this you have to add the following code to `~/.emacs` or add it as as separate file to `/etc/emacs/site-start.d` (I have it as `50i18n.el` there).

```
(set-input-mode (car (current-input-mode))
  (nth 1 (current-input-mode))
  0)
(standard-display-european t)
```

If you have a US keyboard layout and want to type foreign characters the minor mode `iso-accent-mode` might be interesting for you. German umlauts are typed just like with LaTeX, maybe that's already common for you.

### 5.6.5 Console modus

For special character sets that need more than simple exchanging of the codepage (e.g. Cyrillian), you will have to modify the console environment to get the proper character translation scheme. This is achieved on Debian in the file `/etc/console-tools/config`. The folowing example shows the configuration for Russian (KOI8-R) charset:

```
SCREEN_FONT=Cyr_a8x8

# for every console
APP_CHARSET_MAP_vc1=koi2alt.trans
APP_CHARSET_MAP_vc2=koi2alt.trans
APP_CHARSET_MAP_vc3=koi2alt.trans
APP_CHARSET_MAP_vc4=koi2alt.trans
APP_CHARSET_MAP_vc5=koi2alt.trans
APP_CHARSET_MAP_vc6=koi2alt.trans
APP_CHARSET_MAP_vc7=koi2alt.trans
APP_CHARSET_MAP_vc8=koi2alt.trans
APP_CHARSET_MAP_vc9=koi2alt.trans
APP_CHARSET_MAP_vc10=koi2alt.trans
```

Furthermore, if you use `SVGATextMode`, you should set the proper fonts in it's configuration file too.

## Chapter 6

# Menus

Debian (or Joost Witteveen to be precise) has developed a mechanism that enables packages to update the system menu for every windowmanager. This even works for KDE and GNOME. You'll find a detailed description by Joost in `/usr/doc/menu/`.

As an administrator you are free to add new items to the menu that will automatically appear in all menus. Menu files that are shipped together with the installed packages are put into `/usr/lib/menu`. To add new items to the menu system, just copy one file from there to `/etc/menu`, edit it and run the program `update-menus`. This will update all menu files in question.

This even works on a per user basis. As user you need to copy such a menu file to `~/.menu`, edit it and then run `update-menus`.

### 6.1 Menu files

A menu file is a plain text file (did you expect anything else?). It is edited with any editor you want to use. A menu file is only processed in the way that a new menu item appears in the Debian menu if the referring package is installed as well. This is done by the statement `?package (pkg)` where `pkg` is the name of the package that has to be installed for this menu item to be generated.

For example, if you want to provide a new system `aumix` call, this item should only be generated if `aumix` is available which means that the package `aumix` has to be installed. In this case you would write `?package (aumix)` at the top of the menu entry. You can divert your menu entries to be available under X11 only, under any real virtual text console or using a text console (i.e. `xterm`). You have to add `:needs=vc` if the program requires a text console, `:needs=x11` if the program requires X11 and `:needs=text` if the program only requires a text terminal, which can be an `xterm` as well.

Below please find a comprehensive example that makes use of this technique.

```
?package(aumix):needs=vc section=Apps/Sound title="aumix"\  
  longtitle="aumix audio mixer" command="/usr/bin/aumix"  
?package(aumix):needs=x11 section=Apps/Sound title="aumix"\  
  icon=/usr/X11R6/include/X11/pixmaps/aumix.xpm\  
  longtitle="aumix audio mixer" command="/usr/X11R6/bin/xaumix"
```



## Chapter 7

# Booting and Runlevel

The regular boot process of a Debian system does not use an initial ramdisk. This is only required if you plan to boot off of LVM or if you use a hardware RAID system that requires a special kernel module which is neither provided by the regular kernel nor provided as source so you can compile it into the kernel.. If you need an initial ramdisk you will need to set it up on your own.

By default a Debian system boots into runlevel 2. By default runlevel 2, 3, 4 and 5 are equivalent. You are free to change this to whatever you like. I have heard people who want to have runlevel 2 being multiuser without net, 3 being multiuser with net and 4 being multiuser with net and X11. There are no restrictions. Though, when installing new Debian packages, they will install itself into runlevel 2, 3, 4 and 5 equivalently. If you don't want to have them run in one of these levels, you have to remove them from these levels manually. If you want your system to boot into a runlevel different from 2, you have to edit the line `id:2:initdefault:` in `/etc/inittab`.

There are two ways to tell the system which programs are to be started in which runlevel. For each such program a start/stop script exists in `/etc/init.d/`. The SysV method uses symbolic links in `/etc/rc?.d`. The second method uses a plain text file `/etc/runlevel.conf`. Debian supports both methods by providing a unified interface for it: `update-rc.d`. When installing a new system SysV style is default. If you want to use the second method (which is much more userfriendly when it comes to editing runlevels) you will have to install the package `file-rc`.

## 7.1 Reloading/Restarting a service

All services that get started during system boot have a script in `/etc/init.d` referenced to it. These scripts understand at least five arguments (some of them understand more). These are:

- `start` - starts the service if it wasn't started already. It will result in an error message if it is already running.
- `stop` - stops the service if it is running. It may result in an error message if the service is not running already.
- `reload` - will tell the service to reload its configuration files if there are any and the service has support for reloading.
- `restart` - will restart the service, i.e. stop it and then start it again.
- `force-reload` - will make the service to reload its configuration files. If there is proper support this is just an alias for `reload`. If support for reloading is missing it's an alias for `restart`.

## 7.2 Adding a new service

You can only add a new service to the boot sequence if a script in `/etc/init.d` exists. In that case the following command will install it with default settings (foo being the name of a script in `/etc/init.d`). For details please refer to the manpage of that program.

```
update-rc.d foo defaults
```

## 7.3 Removing a service

A service may only be removed after the script in `/etc/init.d` as deleted already. If so, the following command will remove its references (foo being the name of a script in `/etc/init.d`).

```
update-rc.d foo remove
```

If you want to remove a service without removing the start/stop script as well, you may consider using the `file-rc` package and editing the runlevel configuration file `/etc/runlevel.conf`. When using the SysV method you should rename the start/stop script, then call `update-rc.d` and then rename the start/stop script back to its old name.



## Chapter 8

# Support

The Debian project is an open project consisting of more than half a hundred members living in all continents on this earth. Since we are no company there is no general Debian Support Division. Instead you'll get the best support from Debian.

### 8.1 Via Mail

There are a lot of mailing lists on [lists.debian.org](http://lists.debian.org) (<http://lists.debian.org/>). The most important one with regard to user support is `debian-user`. More than 2,500 people are reading the list, most probably somebody knows how to help you fix the problem you're working on.

With regards to software and package development there is our `debian-devel` list. Many Debian developers are reading there, more than 1,000 people are subscribed. Important development discussions take place there.

### 8.2 Online

When you are online there are a couple of resources for getting support in a realtime manner. A lot of Debian developers and users use Internet Relay Chat (IRC) at least occasionally. By joining the channels they listen to you'll get in touch with them directly.

- `#Debian` on Open Projects Network
- `#Debian.DE` (<http://channel.debian.de/>) on IRCNet
- Debian Bug Tracking System (<http://bugs.debian.org/>)
- `debianHELP` (<http://www.debianhelp.org/>)
- Debian Planet (<http://www.debianplanet.org/>)

### 8.3 Personal

The Debian project often gets invitation for conferences and exhibitions. Please check out our events (<http://www.debian.org/events/>) pages. If you want to meet Debian people or have questions that you would like to get answered directly, look for a person to interview for a magazine or just want something to be demonstrated, please stop by at the booth and meet us.



## Chapter 9

# Building Packages

This section covers binary packages and the process to build or rebuild them from proper source packages.

### 9.1 Source Packages

The Debian distribution comes in form of binary and source packages. While you would only install binary packages, source packages can be of more interest to you. You can learn a lot by reading other peoples code or patches. This can also be important if you want to use the same software on different OSes or want to use a specific Debian patch somewhere else.

A Debian source package usually consists of three files:

- `.dsc` containing meta information about the source package
- `.diff.gz` containing all patches that Debian applied to the package or `.debian.tar.gz` containing all patches that Debian applied to the package (new source format 3.0)
- `.orig.tar.gz` containing the pristine upstream source tarball

A couple of packages are Debian native packages that have no different upstream source. In that case, no `.diff.gz` but only a `.tar.gz` will be provided.

### 9.2 Unpacking the Source on Debian

On a Debian system you will use the command `dpkg-source -x file.dsc` This will automatically unpack the original source and apply all Debian patches regardless of the source format used. You will find the new source files in a directory named like `package-version` without the Debian revision in the version string ready for build binary packages.

To create a new source package from an existing source tree (i.e. build new `.diff.gz` or `.debian.tar.gz` respectively and `.dsc` files) the command `dpkg-source -b package-version` is used.

### 9.3 Unpacking the Source somewhere

Debian source packages are designed to be extractable by standard utilities that are available on any GNU/Linux (or Unix/POSIX) system. Thus you'd only need `gunzip`, `tar` and `patch` to build a proper Debian source tree. The following steps have to be taken:

- 1 Unpack the original tar file (i.e. `tar xzf file.orig.tar.gz`)
- 2 rename the resulting directory to `package-version` (e.g. `hello-1.0`). Patch will only work with properly named directories.
- 3 Create the directory `debian` within the new directory
- 4 Apply the Debian patch with `zcat file.diff.gz | patch -p0`. OR change into the package directory and untar the `.debian.tar.gz` with `tar xzf file.debian.gz`
- 5 Make the file `debian/rules` executable (i.e. `chmod +x`)

## 9.4 Rebuilding

From time to time it makes sense to rebuild a Debian package. If it was uploaded for unstable and you need it on a stable system or if it was uploaded for an architecture you don't have or if you want to modify its source it is useful to rebuild the source.

The most automated way to rebuild a Debian package is achieved by letting the Debian tools work. To do so, move into the new source directory and issue the command `dpkg-buildpackage -b`.

You can also perform the steps manually by calling `debian/rules` with proper arguments:

```
make -f debian/rules build
make -f debian/rules binary
```

At the end of both commands you'll find a freshly built `.deb`-file in the upper directory, ready for installation.

## 9.5 Packaging recent source

Packaging upstream source for Debian that is more recent than the most recent Debian package is one of the easiest tasks. However this will only work if there were no structural changes made in the upstream source. When this applies, you only have to fetch the most recent Debian source tree that build properly on your system and copy the entire `debian/`-directory into the new upstream source directory.

Once this is done, you have to edit the file `debian/changelog` which provides the version number and the address of the person who builds the package. It has a very simple structure so you should be able to duplicate it with any text editor. Once duplicated, please edit the first line (version number) and correct the version string. Please use `-0.1` as addition so that new Debian packages (which would have `-1`) will supersede this one. Then edit the mail-address at the end of that item. Now please edit the lines between version and address that contains changelog information. You could write something like *unofficial packages* to make sure you'll remember how that package was built.

After that, simply continue as if you were rebuilding a package. If you have renamed the upstream tarfile into `<package>_<upstream-version>.orig.tar.gz` and named the directory like `<package>-<upstream-version>` after a `make -f debian/rules clean` even `dpkg-source -b` should be able to build proper source packages.

## 9.6 Backporting

When new packages can't be installed on older systems (e.g. the Apache from unstable can't be installed on stable) it is time to backport the package. This may not be as easy as you expect.

For the first trial you can simply unpack the source and try to compile it. If that works, dance and be happy. If not, try to analyse the problem. If it is just due to unsupported programs like `dh_*`, then install the new `debhelper` package from unstable. Since it is binary-all (i.e. contains only perl scripts and documentation) this should work. Then try to recompile the package.

If the error message looks like the build system lacks some library routines or requests newer version of a library, try to locate the referring package and the referring `-dev` package. When trying to find the required library read the section above covering locating packages.

Before installing a `-dev` package please keep in mind that you can only install one similar `-dev` package of a certain library (i.e. only one `gtk1.1.x-dev` package).

When that new library is not installable on your system, you will have to enter a new level of recursion and backport that library first. It is possible that you'll end up in level 10 or so. If so, you may want to re-evaluate your decision not to upgrade the whole system.

## 9.7 Finding new upstream source

When a Debian package is outdated and you want to try out the new version of if you know that the new version of a certain program provides some functionality that you would like to use and the old package doesn't provide it, you'll need to check for new upstream source packages.

For finding out where to find new upstream source packages, please read `/usr/share/doc/<package>/copyright`. This file does not only contain copyright information for that particular package but contains an URL from which source packages were fetched as well.

If the file does not contain an upstream source URL or that url is out-dated there are two other common places where software for Linux is likely to be found:

- 1 For historical reasons the main archive for Linux Software is ibiblio.org (<ftp://ibiblio.org/pub/Linux/>) (formerly known as metalab, formerly known as sunsite.unc.edu).
- 2 Nowadays many new packages are not uploaded to ibiblio anymore but announced on freshmeat (<http://freshmeat.net/>) instead. They collect URLs and descriptions of packages. If you want to download the files you'll be redirected to the authors archive

## 9.8 New upstream source

If you want to package a new upstream version of an existing Debian package, you will find that packaging can be quite easy. Nevertheless, you should take a look at the next section (Packaging). Please read carefully what is written about the `changelog` file since it contains the version number the package will have later.

For packaging a new upstream version of an existing Debian package you'll need to copy the `debian/` directory from the most recent Debian source package into the new upstream source tree basically. Using only this you would end up with the same version number, thus you should increase it by editing `debian/changelog`.

This is done manually by copying the most recent entry in that file and modifying it, by using `debian-changelog-mode` or by using the `dch` program from the `devscripts` package. If you want to let your version be overwritten by a new Debian package you should add `-0.1` to the upstream version number. Since Debian packages normally use `-1` as revision appendix the Debian version would be considered newer than yours.

If you need to package a source archive that does not yet exist in Debian, the most easiest way would be to use the `debian/` directory from an existing package. In this case you should have read the following paragraph and should have some basic knowledge about Debian Packaging.

## 9.9 Packaging

This section could be larger than the entire document since Debian Packaging can be complex and complicated. However, I don't want to overstress the reader so I will try to keep it short. However#2, if you plan to maintain packages officially (by becoming an official maintainer) please read the documents listed in the Resources section.

### 9.9.1 Basics

A Debian package consists of source packages and binary packages. All binary packages are automatically built out of the source packages. Please refer to sections above covering unpacking the source and building the package.

All Debian related information and files are placed in the `debian/`-directory inside of the source directory. Basic files in there are:

- `control` - contains main meta information for a package
- `rules` - contains the main makefile to build the package
- `conffiles` - contains a list of used config files
- `changelog` - contains the packages' logfile (`debian-changelog-mode.el`)

These files are the most important ones among others. Please take into account that the version and name of a package are actually extracted from the `changelog` file. You'll have to increase the version of the top most entry in that file in order to produce a package with a higher version number as before. If you want to build a new package out of pristine source you should at least use the following steps.

- Install `debian/conffiles` into `debian/tmp/DEBIAN`

- Install `debian/preinst` (etc.) into `debian/tmp/DEBIAN`
- Install `debian/changelog` into `debian/tmp/usr/share/doc/<package>` as `changelog.Debian` and compress it with `gzip`.
- Call `dpkg-gencontrol`
- Call `dpkg-shlibdeps` on all executables in one argument list

If you use `debhelper` some work will be done by that tool. However, if you use it and it doesn't do what you have expected, don't complain, it was your choice to use it instead of building it manually.

For learning purpose it is always a good idea to look how other packages have managed their build process. Please check out the `hello` and `hello-debhelper` packages as well as some random packages like `cfingerd` or `miscfiles`.

## 9.9.2 Developers Resources

If you plan to become an official maintainer for a Debian package you should read all the documents that describe the development process and philosophical background.

- Social Contract ([http://www.debian.org/social\\_contract](http://www.debian.org/social_contract))
- Debian Constitution (<http://www.debian.org/devel/constitution>)
- Developer's Reference (<http://www.debian.org/doc/packaging-manuals/developers-reference/>)
- Debian Policy Manual (<http://www.debian.org/doc/debian-policy/>)
- Joining Debian (<http://www.debian.org/devel/join/>)
- Debian Contacts (<http://www.debian.org/intro/organization>)
- Developer's Reference (<http://www.debian.org/doc/packaging-manuals/developers-reference/>)
- Release Critical Bugs (<http://bugs.debian.org/release-critical/>)
- Work Needing and Prospective Packages (<http://www.debian.org/devel/wnpp/>)
- New Maintainers' Guide (<http://www.debian.org/doc/maint-guide/>)
- Debian Menu System (<http://www.debian.org/doc/packaging-manuals/menu-policy/>)
- Debconf Programmer's Tutorial (<http://kitenet.net/doc/debconf-doc/tutorial.html>)
- Managing Debian Packages with CVS ([http://www.debian.org/devel/cvs\\_packages](http://www.debian.org/devel/cvs_packages)) (optional)
- Developers Corner (<http://www.debian.org/devel/>)

Please read these documents carefully. You don't have to bear in mind everything which is written in them. However, you will have to remember where to find what information if you actually need them.

I advise you really to read all those documents before attempting to join the Debian project, it will cause a lot of reading but will help you not to get frustrated during packaging.