Gentoo Linux 2006.0 x86 Handbook

Gentoo Foundation, Inc.

This is the Gentoo Handbook, an effort to centralise Gentoo/Linux information. This handbook contains the installation instructions for a networkless installation on x86 systems and parts about working with Gentoo and Portage.

Contents:

This handbook was processed on Wed Feb 22 16:18:26 UTC 2006
$ Header: /var/cvsroot/gentoo/xml/htdocs/doc/en/handbook/draft/2006.0/handbook-x86.xml, v1.9 2006/02/22 04:07:43 fox2mike $

Authors

Sven Vermeulen [swift@gentoo.org]
Author

Roy Marples [uberlord@gentoo.org]
Author

Daniel Robbins [drobbins@gentoo.org]
Author

Chris Houser [chouser@gentoo.org]
Author

Jerry Alexandratos [jerry@gentoo.org]
Author

Joshua Saddler [nightmorph@gentoo.org]
Author

Seemant Kulleen [seemant@gentoo.org]
Gentoo x86 Developer

Tavis Ormandy [taviso@gentoo.org]
Gentoo Alpha Developer

Jason Huebel [jhuebel@gentoo.org]
Gentoo AMD64 Developer

Guy Martin [gmsoft@gentoo.org]
Gentoo HPPA developer

Pieter Van den Abeele [pvdabeel@gentoo.org]
Gentoo PPC developer

Joe Kallar [blademan@gentoo.org]
Gentoo SPARC developer

John P. Davis [zhen@gentoo.org]
Editor

Pierre-Henri Jondot
Editor

Eric Stockbridge [stocke2@gentoo.org]
Editor

Rajiv Manglani [rajiv@gentoo.org]
Editor

Jungmin Seo [seo@gentoo.org]
Editor

Stoyan Zhekov [zhware@gentoo.org]
Editor

Jared Hudson [jhhudso@gentoo.org]
Editor

Colin Morey [peitolm@gentoo.org]
Editor

Jorge Paulo [peesh@gentoo.org]
Editor

Carl Anderson [carl@gentoo.org]
Editor

Jon Portnoy [avenj@gentoo.org]
Editor

Zack Gilburd [klasikahl@gentoo.org]
Editor

Jack Morgan [jmorgan@gentoo.org]
Editor

Benny Chuang [bennyc@gentoo.org]
Editor

Erwin [erwin@gentoo.org]
Editor

Joshua Kinard [kumba@gentoo.org]
Editor

Tobias Scherbaum [dertobi123@gentoo.org]
Editor

Xavier Neys [neysx@gentoo.org]
Editor

Shyam Mani [fox2mike@gentoo.org]
Editor

Grant Goodyear [g2boojum@gentoo.org]
Reviewer

Gerald J. Normandin Jr. [gerrynjr@gentoo.org]
Reviewer

Donnie Berkholz [spyderous@gentoo.org]
Reviewer

Ken Nowack [antifa@gentoo.org]
Reviewer

Lars Weiler [pylon@gentoo.org]
Contributor

Part 1

Installing Gentoo

In this part you learn how to install Gentoo on your system.

Contents:

About the Gentoo Linux Installation

“Users not familiar with Gentoo do not always know that choice is what Gentoo is all about.”

1: Introduction

First of all, *welcome* to Gentoo. You are about to enter the world of customization and performance. When installing Gentoo, this is made clear to you several times -- you can choose how much you want to compile yourself, how to install Gentoo, what system logger you want, etc.

Gentoo is a fast, modern meta-distribution with a clean and flexible design. Gentoo is built around free software and doesn't hide from its users what is beneath the hood. Portage, the package maintenance system which Gentoo uses, is written in Python, meaning you can easily view and modify the source code. Gentoo's packaging system uses source code (although support for precompiled packages is included too) and configuring Gentoo happens through regular text files. In other words, openness everywhere.

It is very important that you understand that *empowerment* is what makes Gentoo run. We try not to force anything on our users and try our best to empower you to make the choices you wish. If you feel a change should be made, please file a bug report [http://bugs.gentoo.org] about it.

. . : : How do I go about Installing Gentoo?

Gentoo Linux comes with two versions of an easy to use Installer. A GTK+ based installer (for use with an X based environment) and a Dialog based installer for use on the console. Chapter 3 of the handbook deals with the GTK+ based installer while Chapter 4 is for the Dialog based one.

Sometimes, you are given a certain choice in the handbook. We try our best to explain what the pros and cons are. We will continue then with a default choice, identified by "Default: " in the title. The other possibilities are marked by "Alternative: ". Do *not* think that the default is what we recommend. It is however what we believe most users will use.

Sometimes you can pursue an optional step. Such steps are marked as "Optional: " and are therefore not needed to install Gentoo. However, some optional steps are dependant on a previous decision you made. We will inform you when this happens, both when you make the decision, and right before the optional step is described.

. . : : What are my Options?

You can install Gentoo in many different ways. You can download and install from one of our Installation CDs, from an existing distribution, from a bootable CD (such as Knoppix), from a netbooted environment, from a rescue floppy, etc.

This document covers the installation using a Gentoo Linux Installation CD, a bootable CD that contains everything you need to get Gentoo Linux up and running. There are two types of Installation CDs, the InstallCD and the Installer LiveCD. The InstallCD is a minimal environment which contains only those packages necessary for installing Gentoo Linux. The LiveCD is a complete Gentoo Linux environment and can be used for multiple tasks, one of which is installing Gentoo Linux. The LiveCD is not available on all architectures at this time. If your architecture does not have a LiveCD, then this document will refer to the Universal InstallCD for you.

This installation approach however does not immediately use the latest version of the available packages; if you want this you should check out the Installation Instructions inside our Gentoo Linux Handbooks [http://www.gentoo.org/doc/en/handbook/index.xml].

For help on the other installation approaches, please read our Alternative Installation Guide [http://www.gentoo.org/doc/en/altinstall.xml]. We also provide a Gentoo Installation Tips & Tricks [http://www.gentoo.org/doc/en/gentoo-x86-tipsntricks.xml] document that might be useful to read as well. If you feel that the current installation instructions are too elaborate, feel free to use our Quick Installation Guide available from our Documentation Resources [http://www.gentoo.org/doc/en/index.xml] if your architecture has such a document available.

. . : : Troubles?

If you find a problem in the installation (or in the installation documentation), please check the errata from our Gentoo Release Engineering Project [http://www.gentoo.org/proj/en/releng/], visit our bug tracking system [http://bugs.gentoo.org] and check if the bug is known. If not, please create a bug report for it so we can take care of it. Do not be afraid of the developers who are assigned to (your) bugs -- they generally don't eat people.

Note though that, although the document you are now reading is architecture-specific, it will contain references to other architectures as well. This is due to the fact that large parts of the Gentoo Handbook use source code that is common for all architectures (to avoid duplication of efforts and starvation of development resources). We will try to keep this to a minimum to avoid confusion.

If you are uncertain if the problem is a user-problem (some error you made despite having read the documentation carefully) or a software-problem (some error we made despite having tested the installation/documentation carefully) you are free to join #gentoo on irc.freenode.net. Of course, you are welcome otherwise too :)

If you have a question regarding Gentoo, check out our Frequently Asked Questions [http://www.gentoo.org/doc/en/faq.xml], available from the Gentoo Documentation [http://www.gentoo.org/doc/en/]. You can also view the FAQs [http://forums.gentoo.org/viewforum.php?f=40] on our forums [http://forums.gentoo.org]. If you can't find the answer there ask on #gentoo, our IRC-channel on irc.freenode.net. Yes, several of us are freaks who sit on IRC :-)

2: Fast Installation using the Gentoo Reference Platform

The Gentoo Reference Platform, from now on abbreviated to GRP, is a snapshot of prebuilt packages users (that means you!) can install during the installation of Gentoo to speed up the installation process. The GRP consists of all packages required to have a fully functional Gentoo installation. They are not just the ones you need to have a base installation up to speed in no time, but all lengthier builds (such as xorg-x11, GNOME, OpenOffice, Mozilla, ...) are available as GRP packages too.

However, these prebuilt packages aren't maintained during the lifetime of the Gentoo distribution. They are snapshots released at every Gentoo release and

make it possible to have a functional environment in a short amount of time. You can then upgrade your system in the background while working in your Gentoo environment.

. . : : How Portage Handles GRP Packages

Your Portage tree - the collection of *ebuilds* (files that contain all information about a package, such as its description, homepage, sourcecode URLs, compilation instructions, dependencies, etc.) - must be synchronised with the GRP set: the versions of the available ebuilds and their accompanying GRP packages must match.

For this reason you can only benefit from the GRP packages Gentoo provides while performing the current installation approach. GRP is not available for those interested in performing an installation using the latest versions of all available packages.

. . : : Is GRP Available?

Not all architectures provide GRP packages. That doesn't mean GRP isn't supported on the other architectures, but it means that we don't have the resources to build and test the GRP packages.

At present we provide GRP packages for the following architectures:

- The **amd64** architecture (amd64)
- The **ppc** architecture (ppc32, ppc64)
- The **sparc** architecture (sparc64)
- The **x86** architecture (athlon, athlon-xp, athlon-mp, pentium-pro, pentium2, pentium3, pentium4 and pentium-m) Note: The packages are for i686 and are available on the Installer LiveCD.

If your architecture (or subarchitecture) isn't on this list, you are not able to opt for a GRP installation.

Now that this introduction is over, let's continue with Booting the Universal InstallCD/Installer LiveCD.

Booting the Installer LiveCD

"Using our Installer LiveCD you can boot up your system into a running environment that allows you to install Gentoo."

1: Hardware Requirements

Before we start, we first list what hardware requirements you need to successfully install Gentoo on your box using the Installer LiveCD.

. . : : Hardware Requirements

CPU	i686 or later
Memory	128 MB
Diskspace	1.5 GB (excluding swap space)
Swap space	At least 256 MB

2: The Gentoo Linux Installer LiveCD

Gentoo Linux can be installed using a *stage3* tarball file. Such a tarball is an archive that contains a minimal environment from which you can successfully install Gentoo Linux onto your system.

Installations using a stage1 or stage2 tarball file are not documented in the Gentoo Handbook - please read the Gentoo FAQ [http://www.gentoo.org/doc/en/faq.xml#stage12] on these matters.

. . : : Gentoo Linux Installer LiveCD

A LiveCD is a bootable medium which contains a self-sustained Gentoo environment. It allows you to boot Linux from the CD. During the boot process your hardware is detected and the appropriate drivers are loaded. The Gentoo Installation CDs are maintained by Gentoo developers.

There currently are two Installation CDs available:

- The Installer LiveCD contains everything you need to install Gentoo. It provides a graphical environment, a graphical as well as console based installer which automatically carries out the installation for you, and of course, the installation instructions for your architecture.
- The Minimal Installation CD contains only a minimal environment that allows you to boot up and configure your network so you can connect to the Internet. It does not contain any additional files and cannot be used during the current installation approach.

3: Download, Burn and Boot the Gentoo Linux Installer LiveCD

You can download the Installer LiveCDs from one of our mirrors [http://www.gentoo.org/main/en/mirrors.xml]. They are located in the `releases/x86/2006.0/livecd` directory.

Inside that directory you'll find an ISO-file. That is a full CD image which you can write on a CD-R.

After downloading the file, you can verify its integrity to see if it is corrupted or not:

- You can check its MD5 checksum and compare it with the MD5 checksum we provide (for instance with the md5sum tool under Linux/Unix or md5sum [http://www.etree.org/md5com.html] for Windows)
- You can verify the cryptographic signature that we provide. You need to obtain the public key we use (17072058) before you proceed though.

To fetch our public key using the GnuPG application, run the following command:

Code Listing 1: Obtaining the public key

```
$ gpg --keyserver subkeys.pgp.net --recv-keys 17072058
```

Now verify the signature:

Code Listing 2: Verify the cryptographic signature

```
$ gpg --verify <signature file> <downloaded iso>
```

To burn the downloaded ISO(s), you have to select raw-burning. How you do this is highly program-dependent. We will discuss cdrecord and K3B here; more information can be found in our Gentoo FAQ [http://www.gentoo.org/doc/en/faq.xml#isoburning].

- With cdrecord, you simply type cdrecord dev=/dev/hdc <downloaded iso file> (replace `/dev/hdc` with your CD-RW drive's device path).
- With K3B, select Tools > CD > Burn Image. Then you can locate your ISO file within the 'Image to Burn' area. Finally click Start.

. . : : Booting the Installer LiveCD

Important: Read this whole subsection before continuing, as you will probably not have the opportunity to read it before doing things later.

Once you have burned your LiveCD, it is time to boot it. Remove all CDs from your CD drives, reboot your system and enter the BIOS. This is usually done by hitting DEL, F1 or ESC, depending on your BIOS. Inside the BIOS, change the boot order so that the CD-ROM is tried before the hard disk. This is often found under "CMOS Setup". If you don't do this, your system will just reboot from the hard disk, ignoring the CD-ROM.

Now place the LiveCD in the CD-ROM drive and reboot. You should see a boot prompt. At this screen, you can hit Enter to begin the boot process with the default boot options, or boot the LiveCD with custom boot options by specifying a kernel followed by boot options and then hitting Enter.

Specifying a kernel? Yes, we provide several kernels on our LiveCD. The default one is gentoo. Other kernels are for specific hardware needs and the -nofb variants which disable framebuffer.

Below you'll find a short overview on the available kernels:

Kernel	*Description*
gentoo	Default 2.6 kernel with support for multiple CPUs
gentoo-nofb	Same as gentoo but without framebuffer support
memtest86	Test your local RAM for errors

You can also provide kernel options. They represent optional settings you can (de)activate at will. The following list is the same as the one you receive when you press F2 through F7 at the bootscreen.

Code Listing 3: Options available to pass to your kernel of choice

```
Hardware options:

acpi=on          This loads support for ACPI and also causes the acpid daemon to
                 be started by the CD on boot. This is only needed if your
                 system requires ACPI to function properly.  This is not
                 required for Hyperthreading support.

acpi=off         Completely disables ACPI.  This is useful on some older
                 systems, and is also a requirement for using APM.  This will
                 disable any Hyperthreading support of your processor.

console=X        This sets up serial console access for the CD.  The first
                 option is the device, usually ttyS0 on x86, followed by any
                 connection options, which are comma separated.  The default
                 options are 9600,8,n,1.

dmraid=X         This allows for passing options to the device-mapper RAID
                 subsystem.  Options should be encapsulated in quotes.

doapm            This loads APM driver support.  This requires you to also use
                 acpi=off.

dobladecenter    This adds some extra pauses into the boot process for the slow
                 USB CDROM of the IBM BladeCenter.

dopcmcia         This loads support for PCMCIA and Cardbus hardware and also
                 causes the pcmcia cardmgr to be started by the CD on boot.
                 This is only required when booting from a PCMCIA/Cardbus device.

doscsi           This loads support for most SCSI controllers.  This is also a
                 requirement for booting most USB devices, as they use the SCSI
                 subsystem of the kernel.

hda=stroke       This allows you to partition the whole hard disk even when your
                 BIOS is unable to handle large disks.  This option is only used
                 on machines with an older BIOS.  Replace hda with the device
                 that is requiring this option.

ide=nodma        This forces the disabling of DMA in the kernel and is required
                 by some IDE chipsets and also by some CDROM drives.  If your
                 system is having trouble reading from your IDE CDROM, try this
                 option.  This also disables the default hdparm settings from
                 being executed.

noapic           This disables the Advanced Programmable Interrupt Controller
                 that is present on newer motherboards.  It has been known to
                 cause some problems on older hardware.

nodetect         This disables all of the autodetection done by the CD,
                 including device autodetection and DHCP probing.  This is
                 useful for doing debugging of a failing CD or driver.

nodhcp           This disables DHCP probing on detected network cards.  This is
                 useful on networks with only static addresses.
```

nodmraid	Disables support for device-mapper RAID, such as that used for on-board IDE/SATA RAID controllers.
nofirewire	This disables the loading of Firewire modules. This should only be necessary if your Firewire hardware is causing a problem with booting the CD.
nogpm	This disables gpm console mouse support.
nohotplug	This disables the loading of the hotplug and coldplug init scripts at boot. This is useful for doing debugging of a failing CD or driver.
nokeymap	This disables the keymap selection used to select non-US keyboard layouts.
nolapic	This disables the local APIC on Uniprocessor kernels.
nosata	This disables the loading of Serial ATA modules. This is useful if your system is having problems with the SATA subsystem.
nosmp	This disables SMP, or Symmetric Multiprocessing, on SMP-enabled kernels. This is useful for debugging SMP-related issues with certain drivers and motherboards.
nosound	This disables sound support and volume setting. This is useful for systems where sound support causes problems.
nousb	This disables the autoloading of USB modules. This is useful for debugging USB issues.

Volume/Device Management:

dodevfs	This enables the deprecated device filesystem on 2.6 systems. You will also need to use noudev for this to take effect. Since devfs is the only option with a 2.4 kernel, this option has no effect if booting a 2.4 kernel.
doevms2	This enables support for IBM's pluggable EVMS, or Enterprise Volume Management System. This is not safe to use with lvm2.
dolvm2	This enables support for Linux's Logical Volume Management. This is not safe to use with evms2.
noudev	This disables udev support on 2.6 kernels. This option requires that dodevfs is used. Since udev is not an option for 2.4 kernels, this options has no effect if booting a 2.4 kernel.
unionfs	Enables support for Unionfs on supported CD images. This will create a writable Unionfs overlay in a tmpfs, allowing you to change any file on the CD.
unionfs=X	Enables support for Unionfs on supported CD images. This will create a writable Unionfs overlay on the device you specify. The device must be formatted with a filesystem recognized and writable by the kernel.

```
Other options:

debug           Enables debugging code. This might get messy, as it displays
                a lot of data to the screen.

docache         This caches the entire runtime portion of the CD into RAM,
                which allows you to umount /mnt/cdrom and mount another CDROM.
                This option requires that you have at least twice as much
                available RAM as the size of the CD.

doload=X        This causes the initial ramdisk to load any module listed, as
                well as dependencies. Replace X with the module name. Multiple
                modules can be specified by a comma-separated list.

noload=X        This causes the initial ramdisk to skip the loading of a
                specific module that may be causing a problem.  Syntax matches
                that of doload.

nox             This causes an X-enabled LiveCD to not automatically start X,
                but rather, to drop to the command line instead.

scandelay       This causes the CD to pause for 10 seconds during certain
                portions the boot process to allow for devices that are slow to
                initialize to be ready for use.

scandelay=X     This allows you to specify a given delay, in seconds, to be
                added to certain portions of the boot process to allow for
                devices that are slow to initialize to be ready for use.
                Replace X with the number of seconds to pause.
```

Now boot your CD, select a kernel (if you are not happy with the default gentoo kernel) and boot options. As an example, we show you how to boot the gentoo kernel, with dopcmcia as kernel parameters:

Code Listing 4: Booting an Installation CD

```
boot: gentoo dopcmcia
```

You will then be greeted with a boot screen and progress bar. If you are installing Gentoo on a system with a non-US keyboard, make sure you immediately press Alt-F1 to switch to verbose mode and follow the prompt. If no selection is made in 10 seconds the default (US keyboard) will be accepted and the boot process will continue. Once the boot process completes, Gnome will start up and you will be automatically logged in to the "Live" Gentoo Linux system as "gentoo" in graphical mode. You will be logged in as "root", the superuser on the other consoles and should have a root ("#") prompt there. You can switch to those consoles by pressing Alt-F2, Alt-F3, Alt-F4 Alt-F5, Alt-F6. Get back to the graphical desktop you started on by pressing Alt-F7. To switch to other consoles from within X, you must prefix the above with Ctrl. You are able to run commands as root from any terminal within the graphical environment by using the sudo application. You can even

become root within a terminal to perform multiple tasks.

Code Listing 5: Using sudo to run applications

```
(Editing the group file)
# sudo vi /etc/group
(Becoming root for a session)
# sudo su -
```

. . : : Extra Hardware Configuration

When the LiveCD boots, it tries to detect all your hardware devices and loads the appropriate kernel modules to support your hardware. In the vast majority of cases, it does a very good job. However, in some cases, it may not auto-load the kernel modules you need. If the PCI auto-detection missed some of your system's hardware, you will have to load the appropriate kernel modules manually. These tasks require root access.

In the next example we try to load the 8139too module (support for certain kinds of network interfaces):

Code Listing 6: Loading kernel modules

```
# modprobe 8139too
```

If you need PCMCIA support, you should start the pcmcia init script:

Code Listing 7: Starting the PCMCIA init script

```
# /etc/init.d/pcmcia start
```

. . : : Optional: Tweaking Hard Disk Performance

If you are an advanced user, you might want to tweak the IDE hard disk performance using hdparm. You will need root access to use hdparm. With the -tT options you can test the performance of your disk (execute it several times to get a more precise impression):

Code Listing 8: Testing disk performance

```
# hdparm -tT /dev/hda
```

To tweak, you can use any of the following examples (or experiment yourself) which use `/dev/hda` as disk (substitute with your disk):

Code Listing 9: Tweaking hard disk performance

```
Activate DMA:                       # hdparm -d 1 /dev/hda
Activate Safe Performance Options:  # hdparm -d 1 -A 1 -m 16 -u 1 -a 64 /dev/hda
```

. . : : Optional: User Accounts

If you plan on giving other people access to your installation environment or you want to chat using irssi without root privileges (for security reasons), you need to create the necessary user accounts and change the root password. You need root access to change the root password and add new users.

To change the root password, use the passwd utility:

Code Listing 10: Changing the root password

```
$ sudo su -
# passwd
New password: (Enter your new password)
Re-enter password: (Re-enter your password)
```

To create a user account, we first enter their credentials, followed by its password. We use useradd and passwd for these tasks. In the next example, we create a user called "john".

Code Listing 11: Creating a user account

```
# useradd -m -G users john
# passwd john
New password: (Enter john's password)
Re-enter password: (Re-enter john's password)
```

You can change your user id from root to the newly created user by using su:

Code Listing 12: Changing user id

```
# su - john
```

You can also change the password for the "gentoo" user in the graphical environment. This account is already suitable for use on the Internet.

Code Listing 13: Changing the gentoo password

```
$ passwd
New password: (Enter your new password)
Re-enter password: (Re-enter your password)
```

. . : : Optional: Viewing Documentation while Installing

If you want to view the Gentoo Handbook (either from-CD or online) during the installation, make sure you have created a user account (see Optional: User Accounts). Then press Alt-F2 to go to a new terminal and log in.

If you want to view the documentation on the CD you can immediately run links to read it:

Code Listing 14: Viewing the on-CD documentation

```
# links /mnt/cdrom/docs/html/index.html
```

However, it is preferred that you use the online Gentoo Handbook as it will be more recent than the one provided on the CD. You can view it using links as well, but only after having completed the *Configuring your Network* chapter (otherwise you won't be able to go on the Internet to view the document):

Code Listing 15: Viewing the Online Documentation

```
# links http://www.gentoo.org/doc/en/handbook/handbook-x86.xml
```

You can go back to your original window by pressing Alt-F7.

You can now choose to proceed by using the GTK+ based installer (which needs X) or the Dialog based installer that can be run on a console.

Using the GTK+ based Gentoo Linux Installer

"You now have an option of using our graphical installer to install Gentoo. Configure the options you need through an easy to use GUI and you're ready to go."

1: Welcome

Once the Gentoo Linux Installer (GLI) has finished loading, you will be greeted by the welcome screen. It provides a friendly introduction to the process of installing Gentoo on your computer. Remember to read each option carefully. There is detailed help available for each step of installation; just click Help in the lower left corner of the installer. We recommend that you always read the help pages before making your choices. Note that at any time during the installation process, you can save your configuration progress in case you need to resume your installation at a later time.

2: Pre-installation Configuration

In the next section, you are required to configure your network. The Installer should have already detected and set up your network for you, but if it didn't, you can manually configure your network. On the *Misc.* tab, you can specify a location of your choice or keep the default of `/var/log/installer.log` where the Installer will store its logs.

. . : : Optional: Remote Installation

If you wish to enable SSH access to the machine, you can start sshd and specify a root password.

. . : : Optional: Load Additional Kernel Modules

If you need to load more kernel modules to support your hardware, enter their names into the appropriate line, each separated by a space.

Warning: Do not change the Install mode selection to anything besides Normal. This feature is still experimental and changing it will leave you with an unbootable system!

3: Partitioning

In order to install Gentoo on your machine, you will need to prepare your disks. The *Partitioning* screen will show you a list of detected disks and allow you to specify the filesystems you would like to have on your partitions. Clicking Clear partitions will erase all previous partitions on your disk, so be careful with this option! It is also possible to resize certain partition types.

If you choose to go with the Recommended layout, the installer will create three partitions: 100MB for `/boot`, a `/swap` partition up to 512MB in size, and the rest of the available space on the disk is used for `/`, the root partition. If you have more than 4GB of unpartitioned space, using the "Recommended layout" will automatically configure your partitions without destroying any data on any existing partitions.

4: Network Mounts

This screen lets you set up and use any existing network mounts during and after installation. Click New to begin configuration. At this time, only NFS is supported.

5: Stage Selection

Since you are performing an installation *without* an internet connection, you **must** check GRP Install as well as Dynamic from the options present. Everything you need to build your system will then be generated from the files on the LiveCD.

Warning: Do not select options other than those specified above when you are installing without an internet connection.

6: Portage Tree

You must choose Snapshot on this screen. The installer will automatically install a Portage tree from the LiveCD once you select Snapshot; you do not need to specify a snapshot URI.

7: make.conf

Since you are performing a GRP/networkless install, you will not be allowed to select USE flags before installation. However, you are free to set your own USE flags in `/etc/make.conf` after you have rebooted into your finished system.

. . : : CFLAGS

You should, however, select your processor type in the *CFLAGS* section along with any custom optimizations you may want, such as -O2 and -pipe.

. . : : Other

Any other options you wish to set for future use should be selected now. *Use unstable (~arch)* will allow you to use packages from the unstable branch of the Portage tree. *Build binary packages* creates ready-to-install binary tarballs of all packages you compile on your system. *DistCC* allows you to share the burden of compiling with another computer via your network connection.*ccache* saves compiled code for later use, and thus can greatly speed up compilation time if you re-install the same package.

You will not be allowed to change your *CHOST*, as this can seriously damage your installation. In *MAKEOPTS* you define how many parallel compilations should occur when you install a package. A good choice is the number of CPUs in your system plus one, but this guideline isn't always perfect. On a uniprocessor system, -j2 might be used.

8: Kernel Sources

You must use the kernel present on the LiveCD for the GRP/networkless install. This is merely a gentoo-sources kernel compiled by genkernel, Gentoo's automated kernel compilation utility and will give you a kernel that automatically detects and configures your hardware upon boot.

If you want to have a nifty background image during system boot, select the Enable bootsplash option.

9: Bootloader

This screen allows to you choose your bootloader and, optionally, specify additional kernel parameters that will be used at bootup.

You may specify which disk to boot from by choosing the appropriate option from *Boot Drive*. In Linux, the first IDE disk in your system is called hda, the second IDE disk is hdb, and so on. If you have SATA or SCSI disks, they will be called sda, sdb, etc. Please make the correct selection for your system.

If you need to pass any additional options to the kernel, such as video and/or VGA statements, simply add them to the "Extra kernel parameters" section.

If you jumpered your harddrive because the BIOS can't handle large harddrives you'll need to append hdx=stroke or for instance, if you have SCSI devices, you should add doscsi as kernel option.

10: Timezone

Study the map and select the region closest to your actual location. Later, you will be asked to select if you want your clock to be set to UTC or local time.

11: Networking

On this screen, you will be able to configure the various network interface devices on your computer. Read the available options carefully.

On the *Hostname/Proxy Information/Other* tab, you will need to choose a hostname for your machine. You may also specify proxy server and DNS settings if needed.

12: Daemons

Cron daemons are helpful programs that run tasks at scheduled times. While you do not *need* to install one, they can be quite useful.

. . : : System logger

A system logger is a necessity for any Linux operating system. Make your selection from the available choices.

13: Extra Packages

The LiveCD contains a number of available pre-built packages. If you wish to install any of them, check the appropriate box.

Important: As you are installing Gentoo without an internet connection, you cannot add any extra packages other than those shown in the installer.

14: Startup Services

This screen allows you to choose various services to load at system boot. Study the available options and their descriptions carefully, and then select your desired services. For example, if you have chosen to install xorg-x11 and wamt to boot straight into a graphical desktop, then you would select "xdm" from the list.

15: Other Settings

Now you will be able to change various settings, including keyboard layout, graphical display manager, the default editor, and whether to set your hardware clock to UTC or local time.

16: Users

First set the root password for the system administrator (the *root* user).

We *strongly* recommend that you create a regular user for daily work. Working as root all the time is *dangerous* and should be avoided! Create your users, add them to the appropriate groups, and set their passwords. You can optionally change their

home directories, select their login shell, and set helpful comments.

17: Review

Please take the time to double-check each step of the installation process, ensuring that your system is properly configured. When you have finished reviewing, you may save your progress and exit, or click Install to begin automatically installing Gentoo.

You are free to browse around on the LiveCD while the installation proceeds. The installer window will alert you when it has finished. At that point, can close the window by clicking the x in the top right corner. When you are ready, you may log out and reboot. Make sure you remove the LiveCD during the reboot.

Congratulations, your system is now fully equipped! Continue with Where to go from here? to learn more about Gentoo.

Using the Dialog based Gentoo Linux Installer

"You also have an option of using our text based installer to install Gentoo. Configure the options you need through an easy to use set of menus and you're ready to go."

1: Welcome

After you boot the Gentoo Linux Installer LiveCD, it will attempt to load a graphical desktop. If it is unable to do so, it will instead display a command line prompt. To launch the installer, simply type:

Code Listing 1: Start the installer

```
# installer
```

Once the installer has finished loading, you will be greeted by the welcome screen. It provides a friendly introduction to the process of installing Gentoo on your computer. Remember to read each option carefully. There is detailed help available for each step of installation at the top of the screen. We recommend that you always read the help provided before making your choices. Note that at any time during the installation process, you can save your configuration progress in case you need to resume your installation at a later time. Use the Tab key (on your keyboard) to move around the menus within a screen and the Enter key to confirm an action.

2: Partitioning

In order to install Gentoo on your machine, you will need to prepare your disks. The Partitioning screen will show you a list of detected disks and allow you to specify the filesystems you would like to have on your partitions. Selecting Clear partitions will erase all previous partitions on your disk, so be careful with this option! It is also possible to resize certain partition types.

If you choose to go with the Recommended layout, the installer will create three

partitions: 100MB for `/boot`, a `/swap` partition up to 512MB in size, and the rest of the available space on the disk is used for `/`, the root partition. If you have more than 4GB of unpartitioned space, using the "Recommended layout" will automatically configure your partitions without destroying any data on any existing partitions.

3: Network Mounts

This screen lets you set up and use any existing network mounts during and after installation. Select New to begin configuration. At this time, only NFS is supported.

4: Stage Selection

Since you are performing an installation *without* an internet connection, you *must* select GRP from the stage options present. Then, on the next screen, select Create from CD. Everything you need to build your system will then be generated from the files on the LiveCD.

Warning: Do not select any options other than those specified above when you are installing without an internet connection.

5: Kernel Sources

You must use the kernel present on the LiveCD for the GRP/networkless install. This is merely a gentoo-sources kernel compiled by genkernel, Gentoo's automated compilation utility, and will give you a kernel that automatically detects and configures your hardware upon boot.

Select LiveCD kernel and continue to the next screen.

6: Bootloader

This screen allows to you choose your bootloader. The installer will automatically configure your choice.

7: Timezone

Study the list and select the region closest to your actual location.

8: Networking

On this screen, you will be able to configure the various network interface devices detected on your computer. Read the available options carefully.

The next screen gives you a choice between DHCP and manual IP address configuration. Once your network interface is properly configured, you will need to create a hostname for your system. Optionally, you may specify a domainname and any DNS server information needed.

9: Extra Packages

The LiveCD contains a number of available pre-built packages. If you wish to install any of them, check the appropriate box.

Important: As you are installing Gentoo without an internet connection, you cannot add any extra packages other than those shown in the installer.

10: Users

First set the root password for the system administrator (the *root* user).

We *strongly* recommend that you create a regular user for daily work. Working as root all the time is *dangerous* and should be avoided! Create your users, set their passwords, and add them to the appropriate groups. You can optionally change their home directories, select their login shell, and set helpful comments.

11: Review

Please take the time to double-check each step of the installation process, ensuring that your system is properly configured. When you have finished reviewing, you may save your progress and exit, or select Install to begin automatically installing Gentoo.

The installer will alert you when it has finished. It will then return you to the command prompt. All you need to do to reboot is type:

Code Listing 2: Rebooting

```
# shutdown -r now
```

Congratulations, your system is now fully equipped! Continue with Where to go from here? to learn more about Gentoo.

Where to go from here?

"Now you have your Gentoo system, but what's next?"

1: Documentation

Congratulations! You now have a working Gentoo system. But where to go from here? What are your options now? What to explore first? Gentoo provides its users with lots of possibilities, and therefore lots of documented (and less documented) features.

You should definitely take a look at the next part of the Gentoo Handbook entitled Working with Gentoo which explains how to keep your software up to date, how to install more software, what USE flags are, how the Gentoo Init system works, etc.

If you are interested in optimizing your system for desktop use, or you want to learn how to configure your system to be a full working desktop system, consult our extensive Gentoo Desktop Documentation Resources [http://www.gentoo.org/doc/en/index.xml?catid=desktop]. Besides, you might want to use our localization guide [http://www.gentoo.org/doc/en/guide-localization.xml] to make your system feel more at home.

We also have a Gentoo Security Handbook [http://www.gentoo.org/doc/en/security/] which is worth reading.

For a full listing of all our available documentation check out our Documentation Resources [http://www.gentoo.org/doc/en/index.xml] page.

2: Gentoo Online

You are of course always welcome on our Gentoo Forums [http://forums.gentoo.org] or on one of our many Gentoo IRC channels [http://www.gentoo.org/main/en/irc.xml].

We also have several mailinglists [http://www.gentoo.org/main/en/lists.xml] open to all our users. Information on how to join is contained in that page.

We'll shut up now and let you enjoy your installation :)

3: Gentoo Changes since 2006.0

Gentoo is a fast-moving target. The following sections describe important changes that affect a Gentoo installation. We only list those that have anything in common with the installation, not with package changes that did not occur during the installation.

There have been no significant changes since.

Part 2

Working with Gentoo

Learn how to work with Gentoo: installing software, altering variables, changing Portage behaviour etc.

Contents:

A Portage Introduction

“This chapter explains the "simple" steps a user definitely needs to know to maintain the software on his system.”

1: Welcome to Portage

Portage is probably Gentoo's most notable innovation in software management. With its high flexibility and enormous amount of features it is frequently seen as the best software management tool available for Linux.

Portage is completely written in Python [http://www.python.org] and Bash [http://www.gnu.org/software/bash] and therefore fully visible to the users as both are scripting languages.

Most users will work with Portage through the emerge tool. This chapter is not meant to duplicate the information available from the emerge man page. For a complete rundown of emerge's options, please consult the man page:

Code Listing 1: Reading the emerge man page

```
$ man emerge
```

2: The Portage Tree

When we talk about packages, we often mean software titles that are available to the Gentoo users through the Portage tree. The Portage tree is a collection of *ebuilds*, files that contain all information Portage needs to maintain software (install, search, query, ...). These ebuilds reside in `/usr/portage` by default.

Whenever you ask Portage to perform some action regarding software titles, it will use the ebuilds on your system as a base. It is therefore important that you regularly update the ebuilds on your system so Portage knows about new software, security updates, etc.

. . : : Updating the Portage Tree

The Portage tree is usually updated with rsync [http://rsync.samba.org/], a fast incremental file transfer utility. Updating is fairly simple as the emerge command provides a front-end for rsync:

Code Listing 2: Updating the Portage tree

```
# emerge --sync
```

If you are unable to rsync due to firewall restrictions you can still update your Portage tree by using our daily generated Portage tree snapshots. The emerge-webrsync tool automatically fetches and installs the latest snapshot on your system:

Code Listing 3: Running emerge-webrsync

```
# emerge-webrsync
```

3: Maintaining Software

To search through the Portage tree after software titles, you can use emerge built-in search capabilities. By default, emerge --search returns the names of packages whose title matches (either fully or partially) the given search term.

For instance, to search for all packages who have "pdf" in their name:

Code Listing 4: Searching for pdf-named packages

```
$ emerge --search pdf
```

If you want to search through the descriptions as well you can use the --searchdesc (or -S) switch:

Code Listing 5: Searching for pdf-related packages

```
$ emerge --searchdesc pdf
```

When you take a look at the output, you'll notice that it gives you a lot of information. The fields are clearly labelled so we won't go further into their

meanings:

Code Listing 6: Example 'emerge --search' output

```
*  net-print/cups-pdf
      Latest version available: 1.5.2
      Latest version installed: [ Not Installed ]
      Size of downloaded files: 15 kB
      Homepage:    http://cip.physik.uni-wuerzburg.de/~vrbehr/cups-pdf/
      Description: Provides a virtual printer for CUPS to produce PDF files.
      License:     GPL-2
```

. . : : Installing Software

Once you've found a software title to your liking, you can easily install it with emerge: just add the package name. For instance, to install gnumeric:

Code Listing 7: Installing gnumeric

```
# emerge gnumeric
```

Since many applications depend on each other, any attempt to install a certain software package might result in the installation of several dependencies as well. Don't worry, Portage handles dependencies well. If you want to find out what Portage *would* install when you ask it to install a certain package, add the --pretend switch. For instance:

Code Listing 8: Pretend to install gnumeric

```
# emerge --pretend gnumeric
```

When you ask Portage to install a package, it will download the necessary source code from the internet (if necessary) and store it by default in `/usr/portage/distfiles`. After this it will unpack, compile and install the package. If you want Portage to only download the sources without installing them, add the --fetchonly option to the emerge command:

Code Listing 9: Download the sourcecode for gnumeric

```
# emerge --fetchonly gnumeric
```

. . : : Finding Installed Package Documentation

Many packages come with their own documentation. Sometimes, the doc USE flag determines whether the package documentation should be installed or not. You can check the existence of a doc USE flag with the emerge -vp <package name> command.

Code Listing 10: Checking the existence of a doc USE flag

```
(alsa-lib is just an example, of course.)
# emerge -vp alsa-lib
[ebuild  N    ] media-libs/alsa-lib-1.0.9_rc3  +doc -jack 674 kB
```

You can enable or disable the doc USE flag either globally in the `/etc/make.conf` file or per package in the `/etc/portage/package.use` file. The USE Flags chapter covers this aspect in detail.

Once the package installed, its documentation is generally found in a subdirectory named after the package under the `/usr/share/doc` directory. You can also list all installed files with the equery tool which is part of the app-portage/gentoolkit package [http://www.gentoo.org/doc/en/gentoolkit.xml].

Code Listing 11: Locating package documentation

```
# ls -l /usr/share/doc/alsa-lib-1.0.9_rc3
total 28
-rw-r--r--  1 root root  669 May 17 21:54 ChangeLog.gz
-rw-r--r--  1 root root 9373 May 17 21:54 COPYING.gz
drwxr-xr-x  2 root root 8560 May 17 21:54 html
-rw-r--r--  1 root root  196 May 17 21:54 TODO.gz

(Alternatively, use equery to locate interesting files:)
# equery files alsa-lib | less
media-libs/alsa-lib-1.0.9_rc3
* Contents of media-libs/alsa-lib-1.0.9_rc3:
/usr
/usr/bin
/usr/bin/alsalisp
(Output truncated)
```

. . : : Removing Software

When you want to remove a software package from your system, use emerge --unmerge. This will tell Portage to remove all files installed by that package from your system *except* the configuration files of that application if you have altered those after the installation. Leaving the configuration files allows you to continue working with the package if you ever decide to install it again.

However, a big warning applies: Portage will *not* check if the package you want to remove is required by another package. It will however warn you when you want to remove an important package that breaks your system if you unmerge it.

Code Listing 12: Removing gnumeric from the system

```
# emerge --unmerge gnumeric
```

When you remove a package from your system, the dependencies of that package that were installed automatically when you installed the software are left. To have Portage locate all dependencies that can now be removed, use emerge's --depclean functionality. We will talk about this later on.

. . : : Updating your System

To keep your system in perfect shape (and not to mention install the latest security updates) you need to update your system regularly. Since Portage only checks the ebuilds in your Portage tree you first have to update your Portage tree. When your Portage tree is updated, you can update your system with emerge --update world. In the next example, we'll also use the --ask switch which will tell Portage to display the list of packages it wants to upgrade and ask you if you want to continue:

Code Listing 13: Updating your system

```
# emerge --update --ask world
```

Portage will then search for newer version of the applications you have installed. However, it will only verify the versions for the applications you have explicitly installed - not the dependencies. If you want to update every single package on your system, add the --deep argument:

Code Listing 14: Updating your entire system

```
# emerge --update --deep world
```

Since security updates also happen in packages you have not explicitly installed on your system (but that are pulled in as dependencies of other programs), it is recommended to run this command once in a while.

If you have altered any of your USE flags lately you might want to add --newuse as well. Portage will then verify if the change requires the installation of new packages or recompilation of existing ones:

Code Listing 15: Performing a full update

```
# emerge --update --deep --newuse world
```

. . : : Metapackages

Some packages in the Portage tree don't have any real content but are used to install a collection of packages. For instance, the kde package will install a complete KDE environment on your system by pulling in various KDE-related packages as dependencies.

If you ever want to remove such a package from your system, running emerge --unmerge on the package won't have much effect as the dependencies remain on the system.

Portage has the functionality to remove orphaned dependencies as well, but since the availability of software is dynamically dependent you first need to update your entire system fully, including the new changes you applied when changing USE flags. After this you can run emerge --depclean to remove the orphaned dependencies. When this is done, you need to rebuild the applications that were dynamically linked to the now-removed software titles but don't require them anymore.

All this is handled with the following three commands:

Code Listing 16: Removing orphaned dependencies

```
# emerge --update --deep --newuse world
# emerge --depclean
# revdep-rebuild
```

revdep-rebuild is provided by the gentoolkit package; don't forget to emerge it first:

Code Listing 17: Installing the gentoolkit package

```
# emerge gentoolkit
```

4: When Portage is Complaining...

As we stated before, Portage is extremely powerful and supports many features that other software management tools lack. To understand this, we explain a few aspects of Portage without going into too much detail.

With Portage different versions of a single package can coexist on a system. While other distributions tend to name their package to those versions (like freetype and freetype2) Portage uses a technology called *SLOT*s. An ebuild declares a certain SLOT for its version. Ebuilds with different SLOTs can coexist on the same system. For instance, the freetype package has ebuilds with SLOT="1" and SLOT="2".

There are also packages that provide the same functionality but are implemented differently. For instance, metalogd, sysklogd and syslog-ng are all system loggers. Applications that rely on the availability of "a system logger" cannot depend on, for instance, metalogd, as the other system loggers are as good a choice as any. Portage allows for *virtuals*: each system logger provides virtual/syslog so that applications can depend on virtual/syslog.

Software in the Portage tree can reside in different branches. By default your system only accepts packages that Gentoo deems stable. Most new software titles, when committed, are added to the testing branch, meaning more testing needs to be done before it is marked as stable. Although you will see the ebuilds for those software in the Portage tree, Portage will not update them before they are placed in the stable branch.

Some software is only available for a few architectures. Or the software doesn't work on the other architectures, or it needs more testing, or the developer that committed the software to the Portage tree is unable to verify if the package works on different architectures.

Each Gentoo installation adheres to a certain profile which contains, amongst other information, the list of packages that are required for a system to function normally.

. . : : Blocked Packages

Code Listing 18: Portage warning about blocked packages (with --pretend)

```
[blocks B     ] mail-mta/ssmtp (is blocking mail-mta/postfix-2.2.2-r1)
```

Code Listing 19: Portage warning about blocked packages (without --pretend)

```
!!! Error: the mail-mta/postfix package conflicts with another package.
!!!        both can't be installed on the same system together.
!!!        Please use 'emerge --pretend' to determine blockers.
```

Ebuilds contain specific fields that inform Portage about its dependencies. There are two possible dependencies: build dependencies, declared in DEPEND and run-time dependencies, declared in RDEPEND. When one of these dependencies explicitly marks a package or virtual as being *not* compatible, it triggers a blockage.

To fix a blockage, you can choose to not install the package or unmerge the conflicting package first. In the given example, you can opt not to install postfix or to remove ssmtp first.

It is also possible that two packages that are yet to be installed are blocking each other. In this rare case, you should find out why you need to install both. In most cases you can do with one of the packages alone. If not, please file a bug on Gentoo's bugtracking system [http://bugs.gentoo.org].

. . : : Masked Packages

Code Listing 20: Portage warning about masked packages

```
!!! all ebuilds that could satisfy "bootsplash" have been masked.
```

Code Listing 21: Portage warning about masked packages - reason

```
!!! possible candidates are:

- gnome-base/gnome-2.8.0_pre1 (masked by: ~x86 keyword)
- lm-sensors/lm-sensors-2.8.7 (masked by: -sparc keyword)
- sys-libs/glibc-2.3.4.20040808 (masked by: -* keyword)
- dev-util/cvsd-1.0.2 (masked by: missing keyword)
- media-video/ati-gatos-4.3.0 (masked by: package.mask)
- sys-libs/glibc-2.3.2-r11 (masked by: profile)
```

When you want to install a package that isn't available for your system, you will receive this masking error. You should try installing a different application that is

available for your system or wait until the package is put available. There is always a reason why a package is masked:

- **~arch keyword** means that the application is not tested sufficiently to be put in the stable branch. Wait a few days or weeks and try again.
- **-arch keyword** or **-* keyword** means that the application does not work on your architecture. If you believe the package does work file a bug at our bugzilla [http://bugs.gentoo.org] website.
- **missing keyword** means that the application has not been tested on your architecture yet. Ask the architecture porting team to test the package or test it for them and report your findings on our bugzilla [http://bugs.gentoo.org] website.
- **package.mask** means that the package has been found corrupt, unstable or worse and has been deliberately marked as do-not-use.
- **profile** means that the package has been found not suitable for your profile. The application might break your system if you installed it or is just not compatible with the profile you use.

. . : : Missing Dependencies

Code Listing 22: Portage warning about missing dependency

```
emerge: there are no ebuilds to satisfy ">=sys-devel/gcc-3.4.2-r4".

!!! Problem with ebuild sys-devel/gcc-3.4.2-r2
!!! Possibly a DEPEND/*DEPEND problem.
```

The application you are trying to install depends on another package that is not available for your system. Please check bugzilla [http://bugs.gentoo.org] if the issue is known and if not, please report it. Unless you are mixing branches this should not occur and is therefore a bug.

. . : : Ambiguous Ebuild Name

Code Listing 23: Portage warning about ambiguous ebuild names

```
!!! The short ebuild name "aterm" is ambiguous.  Please specify
!!! one of the following fully-qualified ebuild names instead:

    dev-libs/aterm
    x11-terms/aterm
```

The application you want to install has a name that corresponds with more than one package. You need to supply the category name as well. Portage will inform

you of possible matches to choose from.

. . : : Circular Dependencies

Code Listing 24: Portage warning about circular dependencies

```
!!! Error: circular dependencies:

ebuild / net-print/cups-1.1.15-r2 depends on ebuild /
app-text/ghostscript-7.05.3-r1
ebuild / app-text/ghostscript-7.05.3-r1 depends on ebuild /
net-print/cups-1.1.15-r2
```

Two (or more) packages you want to install depend on each other and can therefore not be installed. This is most likely a bug in the Portage tree. Please resync after a while and try again. You can also check bugzilla [http://bugs.gentoo.org] if the issue is known and if not, report it.

. . : : Fetch failed

Code Listing 25: Portage warning about fetch failed

```
!!! Fetch failed for sys-libs/ncurses-5.4-r5, continuing...
(...)
!!! Some fetch errors were encountered.  Please see above for details.
```

Portage was unable to download the sources for the given application and will try to continue installing the other applications (if applicable). This failure can be due to a mirror that has not synchronised correctly or because the ebuild points to an incorrect location. The server where the sources reside can also be down for some reason.

Retry after one hour to see if the issue still persists.

. . : : System Profile Protection

Code Listing 26: Portage warning about profile-protected package

```
!!! Trying to unmerge package(s) in system profile. 'sys-apps/portage'
!!! This could be damaging to your system.
```

You have asked to remove a package that is part of your system's core packages. It is listed in your profile as required and should therefore not be removed from the system.

USE flags

"USE-flags are a very important aspect of Gentoo. In this chapter, you learn to work with USE-flags and understand how USE-flags interact with your system."

1: What are USE-flags?

When you are installing Gentoo (or any other distribution, or even operating system for that matter) you make choices depending on the environment you are working with. A setup for a server differs from a setup for a workstation. A gaming workstation differs from a 3D rendering workstation.

This is not only true for choosing what packages you want to install, but also what features a certain package should support. If you don't need OpenGL, why would you bother installing OpenGL and build OpenGL support in most of your packages? If you don't want to use KDE, why would you bother compiling packages with KDE-support if those packages work flawlessly without?

To help users in deciding what to install/activate and what not, we wanted the user to specify his/her environment in an easy way. This forces the user into deciding what they really want and eases the process for Portage, our package managment system, to make useful decisions.

. . : : Definition of a USE-flag

Enter the USE-flags. Such a flag is a keyword that embodies support and dependency-information for a certain concept. If you define a certain USE-flag, Portage will know that you want support for the chosen keyword. Of course this also alters the dependency information for a package.

Let us take a look at a specific example: the kde keyword. If you do not have this keyword in your USE variable, all packages that have *optional* KDE support will be compiled *without* KDE support. All packages that have an *optional* KDE dependency will be installed *without* installing the KDE libraries (as dependency). If you have defined the kde keyword, then those packages *will* be compiled with KDE support, and the KDE libraries will be installed as dependency.

By correctly defining the keywords you will receive a system tailored specifically to your needs.

. . : : What USE-flags exist?

There are two types of USE-flags: *global* and *local* USE-flags.

- A *global* USE-flag is used by several packages, system-wide. This is what most people see as USE-flags.
- A *local* USE-flag is used by a single package to make package-specific decisions.

A list of available global USE-flags can be found online [http://www.gentoo.org/dyn/use-index.xml] or locally in `/usr/portage/profiles/use.desc`.

A list of available local USE-flags can be found locally in `/usr/portage/profiles/use.local.desc`.

2: Using USE-flags

In the hope you are convinced of the importance of USE-flags we will now inform you how to declare USE-flags.

As previously mentioned, all USE-flags are declared inside the USE variable. To make it easy for users to search and pick USE-flags, we already provide a *default* USE setting. This setting is a collection of USE-flags we think are commonly used by the Gentoo users. This default setting is declared in the `make.defaults` files part of your profile.

The profile your system listens to is pointed to by the `/etc/make.profile` symlink. Each profile works on top of another, larger profile, the end result is therefore the sum of all profiles. The top profile is the `base` profile (`/usr/portage/profiles/base`).

Let us take a look at this default setting for the 2004.3 profile:

Code Listing 1: Cumulative make.defaults USE variable for the 2004.3 profile

```
(This example is the sum of the settings in base, default-linux,
 default-linux/x86 and default-linux/x86/2004.3)
USE="x86 oss apm arts avi berkdb bitmap-fonts crypt cups encode fortran f77
     foomaticdb gdbm gif gpm gtk imlib jpeg kde gnome libg++ libwww mad
     mikmod motif mpeg ncurses nls oggvorbis opengl pam pdflib png python qt
     quicktime readline sdl spell ssl svga tcpd truetype X xml2 xmms xv zlib"
```

As you can see, this variable already contains quite a lot of keywords. Do **not** alter any `make.defaults` file to tailor the USE variable to your needs: changes in this file will be undone when you update Portage!

To change this default setting, you need to add or remove keywords to the USE variable. This is done globally by defining the USE variable in `/etc/make.conf`. In this variable you add the extra USE-flags you require, or remove the USE-flags you don't want. This latter is done by prefixing the keyword with the minus-sign ("-").

For instance, to remove support for KDE and QT but add support for ldap, the following USE can be defined in `/etc/make.conf`:

Code Listing 2: An example USE setting in /etc/make.conf

```
USE="-kde -qt ldap"
```

. . : : Declaring USE flags for individual packages

Sometimes you want to declare a certain USE flag for one (or a couple) of applications but not system-wide. To accomplish this, you will need to create the `/etc/portage` directory (if it doesn't exist yet) and edit `/etc/portage/package.use`.

For instance, if you don't want berkdb support globally but you do want it for mysql, you would add:

Code Listing 3: /etc/portage/package.use example

```
dev-db/mysql berkdb
```

You can of course also explicitly *disable* USE flags for a certain application. For instance, if you don't want java support in PHP:

Code Listing 4: /etc/portage/package.use 2nd example

```
dev-php/php -java
```

. . : : Declare temporary USE-flags

Sometimes you want to set a certain USE-setting only once. Instead of editing `/etc/make.conf` twice (to do and undo the USE-changes) you can just declare the USE-variable as environment variable. Remember that, when you re-emerge or update this application (either explicitly or as part of a system update) your changes will be lost!

As an example we will temporarily remove java from the USE-setting during the installation of mozilla.

Code Listing 5: Using USE as environment variable

```
# USE="-java" emerge mozilla
```

. . : : Automatic USE Flags

After certain packages are installed, additional USE flags will automatically be enabled for you if you do not explicitly disable them. To view the list of packages that trigger automatic USE-flags, check `/etc/make.profile/use.defaults` and the `use.defaults` files of the parent profiles.

Code Listing 6: A snippet from /etc/make.profile/use.defaults

```
gnome           gnome-base/gnome
gtk             x11-libs/gtk+
qt              x11-libs/qt
kde             kde-base/kdebase
motif           x11-libs/openmotif
```

. . : : Precedence

Of course there is a certain precedence on what setting has priority over the USE setting. You don't want to declare USE="-java" only to see that java is still used due to a setting that has a higher priority. The precedence for the USE setting is, ordered by priority (first has lowest priority):

1. Default USE setting declared in the `make.defaults` files part of your profile

2. Inherited USE setting if a package from profile `use.defaults` is installed
3. User-defined USE setting in `/etc/make.conf`
4. User-defined USE setting in `/etc/portage/package.use`
5. User-defined USE setting as environment variable

To view the final USE setting as seen by Portage, run emerge --info. This will list all relevant variables (including the USE variable) with the content used by Portage.

Code Listing 7: Running emerge --info

```
# emerge --info
```

. . : : Adapting your Entire System to New USE Flags

If you have altered your USE flags and you wish to update your entire system to use the new USE flags, use emerge's --newuse option:

Code Listing 8: Rebuilding your entire system

```
# emerge --update --deep --newuse world
```

Next, run Portage's depclean to remove the conditional dependencies that were emerged on your "old" system but that have been obsoleted by the new USE flags.

Warning: Running emerge --depclean is a dangerous operation and should be handled with care. Double-check the provided list of "obsoleted" packages to make sure it doesn't remove packages you need. In the following example we add the -p switch to have depclean only list the packages without removing them.

Code Listing 9: Removing obsoleted packages

```
# emerge -p --depclean
```

When depclean has finished, run revdep-rebuild to rebuild the applications that are dynamically linked against shared objects provided by possibly removed packages. revdep-rebuild is part of the gentoolkit package; don't forget to emerge it first.

Code Listing 10: Running revdep-rebuild

```
# revdep-rebuild
```

When all this is accomplished, your system is using the new USE flag settings.

3: Package specific USE-flags

Let us take the example of mozilla: what USE-flags does it listen to? To find out, we use emerge with the --pretend and --verbose options:

Code Listing 11: Viewing the used USE-flags

```
# emerge --pretend --verbose mozilla
These are the packages that I would merge, in order:

Calculating dependencies ...done!
[ebuild   R   ] www-client/mozilla-1.7.12-r2  USE="crypt gnome java mozsvg ssl
truetype xprint -debug -ipv6 -ldap -mozcalendar -mozdevelop -moznocompose
-moznoirc -moznomail -moznoxft -postgres -xinerama" 0 kB
```

emerge isn't the only tool for this job. In fact, we have a tool dedicated to package information called equery which resides in the gentoolkit package. First, install gentoolkit:

Code Listing 12: Installing gentoolkit

```
# emerge gentoolkit
```

Now run equery with the uses argument to view the USE-flags of a certain package. For instance, for the gnumeric package:

Code Listing 13: Using equery to view used USE-flags

```
# equery uses gnumeric
[ Colour Code : set unset ]
[ Legend      : (U) Col 1 - Current USE flags        ]
[             : (I) Col 2 - Installed With USE flags ]

 U I [ Found these USE variables in : app-office/gnumeric-1.2.0 ]
 - - libgda  : Adds GNU Data Access (CORBA wrapper) support for gnumeric
 - - gnomedb : unknown
 + + python  : Adds support/bindings for the Python language
 + + bonobo  : Adds support for gnome-base/bonobo (Gnome CORBA interfaces)
```

Portage Features

"Discover the features Portage has, such as support for distributed compiling, ccache and more."

1: Portage Features

Portage has several additional features that makes your Gentoo experience even better. Many of these features rely on certain software tools that improve performance, reliability, security, ...

To enable or disable certain Portage features you need to edit `/etc/make.conf`'s FEATURES variable which contains the various feature keywords, separated by white space. In several cases you will also need to install the additional tool on which the feature relies.

Not all features that Portage supports are listed here. For a full overview, please consult the `make.conf` man page:

Code Listing 1: Consulting the make.conf man page

```
$ man make.conf
```

To find out what FEATURES are default set, run emerge --info and search for the FEATURES variable or grep it out:

Code Listing 2: Finding out the FEATURES that are already set

```
$ emerge --info | grep FEATURES
```

2: Distributed Compiling

distcc is a program to distribute compilations across several, not necessarily identical, machines on a network. The distcc client sends all necessary information to the available distcc servers (running distccd) so they can compile pieces of source code for the client. The net result is a faster compilation time.

You can find more information about distcc (and how to have it work with Gentoo) in our Gentoo Distcc Documentation [http://www.gentoo.org/doc/en/distcc.xml].

. . : : Installing distcc

Distcc ships with a graphical monitor to monitor tasks that your computer is sending away for compilation. If you use Gnome then put 'gnome' in your USE variable. However, if you don't use Gnome and would still like to have the monitor then you should put 'gtk' in your USE variable.

Code Listing 3: Installing distcc

```
# emerge distcc
```

. . : : Activating Portage Support

Add distcc to the FEATURES variable inside `/etc/make.conf`. Next, edit the MAKEOPTS variable to your liking. A known guideline is to fill in "-jX" with X the number of CPUs that run distccd (including the current host) plus one, but you might have better results with other numbers.

Now run distcc-config and enter the list of available distcc servers. For a simple example we assume that the available DistCC servers are 192.168.1.102 (the current host), 192.168.1.103 and 192.168.1.104 (two "remote" hosts):

Code Listing 4: Configuring distcc to use three available distcc servers

```
# distcc-config --set-hosts "192.168.1.102 192.168.1.103 192.168.1.104"
```

Don't forget to run the distccd daemon as well:

Code Listing 5: Starting the distccd daemons

```
# rc-update add distccd default
# /etc/init.d/distccd start
```

3: Caching Compilation

ccache is a fast compiler cache. When you compile a program, it will cache intermediate results so that, whenever you recompile the same program, the compilation time is greatly reduced. In common compilations this can result in 5 to 10 times faster compilation times.

If you are interested in the ins and outs of ccache, please visit the ccache homepage [http://ccache.samba.org].

. . : : Installing ccache

To install ccache, run emerge ccache:

Code Listing 6: Installing ccache

```
# emerge ccache
```

. . : : Activating Portage Support

Open `/etc/make.conf` and add ccache to the FEATURES variable. Next, add a new variable called CCACHE_SIZE and set it to "2G":

Code Listing 7: Editing CCACHE_SIZE in /etc/make.conf

```
CCACHE_SIZE="2G"
```

To check if ccache functions, ask ccache to provide you with its statistics. Because Portage uses a different ccache home directory, you need to set the CCACHE_DIR variable as well:

Code Listing 8: Viewing ccache statistics

```
# CCACHE_DIR="/var/tmp/ccache" ccache -s
```

The `/var/tmp/ccache` location is Portage' default ccache home directory; if you want to alter this setting you can set the CCACHE_DIR variable in `/etc/make.conf`.

However, if you would run ccache, it would use the default location of `${HOME}/.ccache`, which is why you needed to set the CCACHE_DIR variable when asking for the (Portage) ccache statistics.

. . : : Using ccache for non-Portage C Compiling

If you would like to use ccache for non-Portage compilations, add `/usr/lib/ccache/bin` to the beginning of your PATH variable (before `/usr/bin`). This can be accomplished by editing `/etc/env.d/00basic`, which is the first environment file that defines the PATH variable:

Code Listing 9: Editing /etc/env.d/00basic

```
PATH="/usr/lib/ccache/bin:/opt/bin"
```

4: Binary Package Support

Portage supports the installation of prebuilt packages. Even though Gentoo does not provide prebuilt packages by itself (except for the GRP snapshots) Portage can be made fully aware of prebuilt packages.

To create a prebuilt package you can use quickpkg if the package is already installed on your system, or emerge with the --buildpkg or --buildpkgonly options.

If you want Portage to create prebuilt packages of every single package you install, add buildpkg to the FEATURES variable.

More extended support for creating prebuilt package sets can be obtained with catalyst. For more information on catalyst please read the Catalyst Reference Manual [http://www.gentoo.org/proj/en/releng/catalyst/2.x/reference.xml] and Catalyst Frequently Asked Questions [http://www.gentoo.org/proj/en/releng/catalyst/faq.xml].

. . : : Installing Prebuilt Packages

Although Gentoo doesn't provide one, you can create a central repository where you store prebuilt packages. If you want to use this repository, you need to make Portage aware of it by having the PORTAGE_BINHOST variable point to it. For instance, if the prebuilt packages are on ftp://buildhost/gentoo:

Code Listing 10: Setting PORTAGE_BINHOST in /etc/make.conf

```
PORTAGE_BINHOST="ftp://buildhost/gentoo"
```

When you want to install a prebuilt package, add the --getbinpkg option to the emerge command alongside of the --usepkg option. The former tells emerge to download the prebuilt package from the previously defined server while the latter asks emerge to try to install the prebuilt package first before fetching the sources and compiling it.

For instance, to install gnumeric with prebuilt packages:

Code Listing 11: Installing the gnumeric prebuilt package

```
# emerge --usepkg --getbinpkg gnumeric
```

More information about emerge's prebuilt package options can be found in the emerge man page:

Code Listing 12: Reading the emerge man page

```
$ man emerge
```

Initscripts

"Gentoo uses a special initscript format which, amongst other features, allows dependency-driven decisions and virtual initscripts. This chapter explains all these aspects and explains how to deal with these scripts."

1: Runlevels

When you boot your system, you will notice lots of text floating by. If you pay close attention, you will notice this text is the same every time you reboot your system. The sequence of all these actions is called the *boot sequence* and is (more or less) statically defined.

First, your boot loader will load the kernel image you have defined in the boot loader configuration into memory after which it tells the CPU to run the kernel. When the kernel is loaded and run, it initializes all kernel-specific structures and tasks and starts the init process.

This process then makes sure that all filesystems (defined in `/etc/fstab`) are mounted and ready to be used. Then it executes several scripts located in `/etc/init.d`, which will start the services you need in order to have a successfully booted system.

Finally, when all scripts are executed, init activates the terminals (in most cases just the virtual consoles which are hidden beneath Alt-F1, Alt-F2, etc.) attaching a special process called agetty to it. This process will then make sure you are able to log on through these terminals by running login.

. . : : Init Scripts

Now init doesn't just execute the scripts in `/etc/init.d` randomly. Even more, it doesn't run all scripts in `/etc/init.d`, only the scripts it is told to execute. It decides which scripts to execute by looking into `/etc/runlevels`.

First, init runs all scripts from `/etc/init.d` that have symbolic links inside `/etc/runlevels/boot`. Usually, it will start the scripts in alphabetical order, but some scripts have dependency information in them, telling the system that another

script must be run before they can be started.

When all `/etc/runlevels/boot` referenced scripts are executed, init continues with running the scripts that have a symbolic link to them in `/etc/runlevels/default`. Again, it will use the alphabetical order to decide what script to run first, unless a script has dependency information in it, in which case the order is changed to provide a valid start-up sequence.

. . : : How Init Works

Of course init doesn't decide all that by itself. It needs a configuration file that specifies what actions need to be taken. This configuration file is `/etc/inittab`.

If you remember the boot sequence we have just described, you will remember that init's first action is to mount all filesystems. This is defined in the following line from `/etc/inittab`:

Code Listing 1: The system initialisation line in /etc/inittab

```
si::sysinit:/sbin/rc sysinit
```

This line tells init that it must run /sbin/rc sysinit to initialize the system. The `/sbin/rc` script takes care of the initialisation, so you might say that init doesn't do much -- it delegates the task of initialising the system to another process.

Second, init executed all scripts that had symbolic links in `/etc/runlevels/boot`. This is defined in the following line:

Code Listing 2: The system initialisation, continued

```
rc::bootwait:/sbin/rc boot
```

Again the rc script performs the necessary tasks. Note that the option given to rc (*boot*) is the same as the subdirectory of `/etc/runlevels` that is used.

Now init checks its configuration file to see what *runlevel* it should run. To decide this, it reads the following line from `/etc/inittab`:

Code Listing 3: The initdefault line

```
id:3:initdefault:
```

In this case (which the majority of Gentoo users will use), the *runlevel* id is 3. Using this information, init checks what it must run to start *runlevel 3*:

Code Listing 4: The runlevel definitions

```
l0:0:wait:/sbin/rc shutdown
l1:S1:wait:/sbin/rc single
l2:2:wait:/sbin/rc nonetwork
l3:3:wait:/sbin/rc default
l4:4:wait:/sbin/rc default
l5:5:wait:/sbin/rc default
l6:6:wait:/sbin/rc reboot
```

The line that defines level 3, again, uses the rc script to start the services (now with argument *default*). Again note that the argument of rc is the same as the subdirectory from `/etc/runlevels`.

When rc has finished, init decides what virtual consoles it should activate and what commands need to be run at each console:

Code Listing 5: The virtual consoles definition

```
c1:12345:respawn:/sbin/agetty 38400 tty1 linux
c2:12345:respawn:/sbin/agetty 38400 tty2 linux
c3:12345:respawn:/sbin/agetty 38400 tty3 linux
c4:12345:respawn:/sbin/agetty 38400 tty4 linux
c5:12345:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

. . : : What is a runlevel?

You have seen that init uses a numbering scheme to decide what *runlevel* it should activate. A *runlevel* is a state in which your system is running and contains a collection of scripts (runlevel scripts or *initscripts*) that must be executed when you enter or leave a runlevel.

In Gentoo, there are seven runlevels defined: three internal runlevels, and four user-defined runlevels. The internal runlevels are called *sysinit*, *shutdown* and *reboot* and do exactly what their names imply: initialize the system, powering off the system and rebooting the system.

The user-defined runlevels are those with an accompanying `/etc/runlevels` subdirectory: `boot`, `default`, `nonetwork` and `single`. The `boot` runlevel starts all system-necessary services which all other runlevels use. The remaining three runlevels differ in what services they start: `default` is used for day-to-day operations, `nonetwork` is used in case no network connectivity is required, and `single` is used when you need to fix the system.

. . : : Working with the Init Scripts

The scripts that the rc process starts are called *init scripts*. Each script in `/etc/init.d` can be executed with the arguments *start*, *stop*, *restart*, *pause*, *zap*, *status*, *ineed*, *iuse*, *needsme*, *usesme* or *broken*.

To start, stop or restart a service (and all depending services), start, stop and restart should be used:

Code Listing 6: Starting Postfix

```
# /etc/init.d/postfix start
```

Note: Only the services that *need* the given service are stopped or restarted. The other depending services (those that *use* the service but don't need it) are left untouched.

If you want to stop a service, but not the services that depend on it, you can use the pause argument:

Code Listing 7: Stopping Postfix but keep the depending services running

```
# /etc/init.d/postfix pause
```

If you want to see what status a service has (started, stopped, paused, ...) you can use the status argument:

Code Listing 8: Status information for postfix

```
# /etc/init.d/postfix status
```

If the status information tells you that the service is running, but you know that it is not, then you can reset the status information to "stopped" with the zap argument:

Code Listing 9: Resetting status information for postfix

```
# /etc/init.d/postfix zap
```

To also ask what dependencies the service has, you can use iuse or ineed. With ineed you can see the services that are really necessary for the correct functioning of the service. iuse on the other hand shows the services that can be used by the service, but are not necessary for the correct functioning.

Code Listing 10: Requesting a list of all necessary services on which Postfix depends

```
# /etc/init.d/postfix ineed
```

Similarly, you can ask what services require the service (needsme) or can use it (usesme):

Code Listing 11: Requesting a list of all services that require Postfix

```
# /etc/init.d/postfix needsme
```

Finally, you can ask what dependencies the service requires that are missing:

Code Listing 12: Requesting a list of missing dependencies for Postfix

```
# /etc/init.d/postfix broken
```

2: Working with rc-update

Gentoo's init system uses a dependency-tree to decide what service needs to be started first. As this is a tedious task that we wouldn't want our users to have to do manually, we have created tools that ease the administration of the runlevels and init scripts.

With rc-update you can add and remove init scripts to a runlevel. The rc-update tool will then automatically ask the depscan.sh script to rebuild the dependency tree.

. . : : Adding and Removing Services

You have already added init scripts to the "default" runlevel during the installation of Gentoo. At that time you might not have had a clue what the "default" is for, but now you should. The rc-update script requires a second argument that defines the action: *add*, *del* or *show*.

To add or remove an init script, just give rc-update the add or del argument, followed by the init script and the runlevel. For instance:

Code Listing 13: Removing Postfix from the default runlevel

```
# rc-update del postfix default
```

The rc-update show command will show all the available init scripts and list at which runlevels they will execute:

Code Listing 14: Receiving init script information

```
# rc-update show
```

3: Configuring Services

Init scripts can be quite complex. It is therefore not really desirable to have the users edit the init script directly, as it would make it more error-prone. It is however important to be able to configure such a service. For instance, you might want to give more options to the service itself.

A second reason to have this configuration outside the init script is to be able to update the init scripts without the fear that your configuration changes will be undone.

. . : : The /etc/conf.d Directory

Gentoo provides an easy way to configure such a service: every init script that can be configured has a file in `/etc/conf.d`. For instance, the apache2 initscript (called `/etc/init.d/apache2`) has a configuration file called `/etc/conf.d/apache2`, which can contain the options you want to give to the Apache 2 server when it is started:

Code Listing 15: Variable defined in /etc/conf.d/apache2

```
APACHE2_OPTS="-D PHP4"
```

Such a configuration file contains variables and variables alone (just like `/etc/make.conf`), making it very easy to configure services. It also allows us to provide more information about the variables (as comments).

4: Writing Init Scripts

No, writing an init script is usually not necessary as Gentoo provides ready-to-use init scripts for all provided services. However, you might have installed a service without using Portage, in which case you will most likely have to create an init script.

Do not use the init script provided by the service if it isn't explicitly written for Gentoo: Gentoo's init scripts are not compatible with the init scripts used by other distributions!

. . : : Layout

The basic layout of an init script is shown below.

Code Listing 16: Basic layout of an init script

```
#!/sbin/runscript

depend() {
  (Dependency information)
}

start() {
  (Commands necessary to start the service)
}

stop() {
  (Commands necessary to stop the service)
}

restart() {
  (Commands necessary to restart the service)
}
```

Any init script *requires* the start() function to be defined. All other sections are optional.

. . : : Dependencies

There are two dependencies you can define: use and need. As we have mentioned before, the need dependency is more strict than the use dependency. Following this dependency type you enter the service you depend on, or the *virtual* dependency.

A *virtual* dependency is a dependency that a service provides, but that is not provided solely by that service. Your init script can depend on a system logger, but there are many system loggers available (metalogd, syslog-ng, sysklogd, ...). As you cannot need every single one of them (no sensible system has all these system loggers installed and running) we made sure that all these services provide a virtual dependency.

Let us take a look at the dependency information for the postfix service.

Code Listing 17: Dependency information for Postfix

```
depend() {
  need net
  use logger dns
  provide mta
}
```

As you can see, the postfix service:

- requires the (virtual) net dependency (which is provided by, for instance, `/etc/init.d/net.eth0`)
- uses the (virtual) logger dependency (which is provided by, for instance, `/etc/init.d/syslog-ng`)
- uses the (virtual) dns dependency (which is provided by, for instance, `/etc/init.d/named`)
- provides the (virtual) mta dependency (which is common for all mail servers)

. . : : Controlling the Order

In some cases you might not require a service, but want your service to be started before (or after) another service *if* it is available on the system (note the conditional - this is no dependency anymore) *and* run in the same runlevel (note the conditional - only services in the same runlevel are involved). You can provide this information using the before or after settings.

As an example we view the settings of the Portmap service:

Code Listing 18: The depend() function in the Portmap service

```
depend() {
  need net
  before inetd
  before xinetd
}
```

You can also use the "*" glob to catch all services in the same runlevel, although this isn't advisable.

Code Listing 19: Running an init script as first script in the runlevel

```
depend() {
  before *
}
```

. . : : Standard Functions

Next to the depend() functionality, you also need to define the start() function. This one contains all the commands necessary to initialize your service. It is advisable to use the ebegin and eend functions to inform the user about what is happening:

Code Listing 20: Example start() function

```
start() {
  ebegin "Starting my_service"
  start-stop-daemon --start --quiet --exec /path/to/my_service
  eend $?
}
```

If you need more examples of the start() function, please read the source code of the available init scripts in your `/etc/init.d` directory. As for start-stop-daemon, there is an excellent man page available if you need more information:

Code Listing 21: Getting the man page for start-stop-daemon

```
# man start-stop-daemon
```

Other functions you can define are: stop() and restart(). You are not obliged to define these functions! Our init system is intelligent enough to fill these functions by itself if you use start-stop-daemon.

Gentoo's init script syntax is based on the Bourne Again Shell (bash) so you are free to use bash-compatible constructs inside your init script.

. . : : Adding Custom Options

If you want your init script to support more options than the ones we have already encountered, you should add the option to the opts variable, and create a function with the same name as the option. For instance, to support an option called restartdelay:

Code Listing 22: Supporting the restartdelay option

```
opts="${opts} restartdelay"

restartdelay() {
  stop
  sleep 3    # Wait 3 seconds before starting again
  start
}
```

. . : : Service Configuration Variables

You don't have to do anything to support a configuration file in `/etc/conf.d`: if your init script is executed, the following files are automatically sourced (i.e. the variables are available to use):

- `/etc/conf.d/<your init script>`
- `/etc/conf.d/basic`
- `/etc/rc.conf`

Also, if your init script provides a virtual dependency (such as net), the file associated with that dependency (such as `/etc/conf.d/net`) will be sourced too.

5: Changing the Runlevel Behaviour

Many laptop users know the situation: at home you need to start net.eth0 while you don't want to start net.eth0 while you're on the road (as there is no network available). With Gentoo you can alter the runlevel behaviour to your own will.

For instance you can create a second "default" runlevel which you can boot that has other init scripts assigned to it. You can then select at boottime what default runlevel you want to use.

. . : : Using softlevel

First of all, create the runlevel directory for your second "default" runlevel. As an example we create the `offline` runlevel:

Code Listing 23: Creating a runlevel directory

```
# mkdir /etc/runlevels/offline
```

Add the necessary init scripts to the newly created runlevels. For instance, if you want to have an exact copy of your current default runlevel but without net.eth0:

Code Listing 24: Adding the necessary init scripts

```
(Copy all services from default runlevel to offline runlevel)
# cd /etc/runlevels/default
# for service in *; do rc-update add $service offline; done
(Remove unwanted service from offline runlevel)
# rc-update del net.eth0 offline
(Display active services for offline runlevel)
# rc-update show offline
(Partial sample Output)
               acpid | offline
          domainname | offline
               local | offline
            net.eth0 |
```

Now edit your bootloader configuration and add a new entry for the offline runlevel. For instance, in `/boot/grub/grub.conf`:

Code Listing 25: Adding an entry for the offline runlevel

```
title Gentoo Linux Offline Usage
  root (hd0,0)
  kernel (hd0,0)/kernel-2.4.25 root=/dev/hda3 softlevel=offline
```

Voilà, you're all set now. If you boot your system and select the newly added entry at boot, the offline runlevel will be used instead of the default one.

. . : : Using bootlevel

Using bootlevel is completely analogous to softlevel. The only difference here is that you define a second "boot" runlevel instead of a second "default" runlevel.

Environment Variables

"With Gentoo you can easily manage the environment variables for your system. This chapter explains how you do that, and also describes frequently used variables."

1: Environment Variables?

An environment variable is a named object that contains information used by one or more applications. Many users (and especially those new to Linux) find this a bit weird or unmanageable. However, this is a mistake: by using environment variables one can easily change a configuration setting for one or more applications.

. . : : Important Examples

The following table lists a number of variables used by a Linux system and describes their use. Example values are presented after the table.

Variable	*Description*
PATH	This variable contains a colon-separated list of directories in which your system looks for executable files. If you enter a name of an executable (such as ls, rc-update or emerge) but this executable is not located in a listed directory, your system will not execute it (unless you enter the full path as command, such as /bin/ls).
ROOTPATH	This variable has the same function as PATH, but this one only lists the directories that should be checked when the root-user enters a command.
LDPATH	This variable contains a colon-separated list of directories in which the dynamical linker searches through to find a library.
MANPATH	This variable contains a colon-separated list of directories in which the man command searches for the man pages.
INFODIR	This variable contains a colon-separated list of directories in which the info command searches for the info pages.
PAGER	This variable contains the path to the program used to list the contents of files through (such as less or more).
EDITOR	This variable contains the path to the program used to change the contents of files with (such as nano or vi).
KDEDIRS	This variable contains a colon-separated list of directories which contain KDE-specific material.
CLASSPATH	This variable contains a colon-separated list of directories which contain Java

	classes.
CONFIG_PROTECT	This variable contains a *space*-delimited list of directories which should be protected by Portage during updates.
CONFIG_PROTECT_MASK	This variable contains a *space*-delimited list of directories which should not be protected by Portage during updates.

Below you will find an example definition of all these variables:

Code Listing 1: Example definitions

```
PATH="/bin:/usr/bin:/usr/local/bin:/opt/bin:/usr/games/bin"
ROOTPATH="/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin"
LDPATH="/lib:/usr/lib:/usr/local/lib:/usr/lib/gcc-lib/i686-pc-linux-gnu/3.2.3"
MANPATH="/usr/share/man:/usr/local/share/man"
INFODIR="/usr/share/info:/usr/local/share/info"
PAGER="/usr/bin/less"
EDITOR="/usr/bin/vim"
KDEDIRS="/usr"
CLASSPATH="/opt/blackdown-jre-1.4.1/lib/rt.jar:."
CONFIG_PROTECT="/usr/X11R6/lib/X11/xkb /opt/tomcat/conf \
                /usr/kde/3.1/share/config /usr/share/texmf/tex/generic/config/ \
                /usr/share/texmf/tex/platex/config/ /usr/share/config"
CONFIG_PROTECT_MASK="/etc/gconf
```

2: Defining Variables Globally

To centralise the definitions of these variables, Gentoo introduced the `/etc/env.d` directory. Inside this directory you will find a number of files, such as `00basic`, `05gcc`, etc. which contain the variables needed by the application mentioned in their name.

For instance, when you installed gcc, a file called `05gcc` was created by the ebuild which contains the definitions of the following variables:

Code Listing 2: /etc/env.d/05gcc

```
PATH="/usr/i686-pc-linux-gnu/gcc-bin/3.2"
ROOTPATH="/usr/i686-pc-linux-gnu/gcc-bin/3.2"
MANPATH="/usr/share/gcc-data/i686-pc-linux-gnu/3.2/man"
INFOPATH="/usr/share/gcc-data/i686-pc-linux-gnu/3.2/info"
CC="gcc"
CXX="g++"
LDPATH="/usr/lib/gcc-lib/i686-pc-linux-gnu/3.2.3"
```

Other distributions tell you to change or add such environment variable definitions

in `/etc/profile` or other locations. Gentoo on the other hand makes it easy for you (and for Portage) to maintain and manage the environment variables without having to pay attention to the numerous files that can contain environment variables.

For instance, when gcc is updated, the `/etc/env.d/05gcc` file is updated too without requesting any user-interaction.

This not only benefits Portage, but also you, as user. Occasionally you might be asked to set a certain environment variable system-wide. As an example we take the http_proxy variable. Instead of messing about with `/etc/profile`, you can now just create a file (`/etc/env.d/99local`) and enter your definition(s) in it:

Code Listing 3: /etc/env.d/99local

```
http_proxy="proxy.server.com:8080"
```

By using the same file for all your variables, you have a quick overview on the variables you have defined yourself.

. . : : The env-update Script

Several files in `/etc/env.d` define the PATH variable. This is not a mistake: when you run env-update, it will append the several definitions before it updates the environment variables, thereby making it easy for packages (or users) to add their own environment variable settings without interfering with the already existing values.

The env-update script will append the values in the alphabetical order of the `/etc/env.d` files. The file names must begin with two decimal digits.

Code Listing 4: Update order used by env-update

```
         00basic        99kde-env       99local
     +-------------+----------------+-------------+
PATH="/bin:/usr/bin:/usr/kde/3.2/bin:/usr/local/bin"
```

The concatenation of variables does not always happen, only with the following variables: KDEDIRS, PATH, CLASSPATH, LDPATH, MANPATH, INFODIR, INFOPATH, ROOTPATH, CONFIG_PROTECT, CONFIG_PROTECT_MASK, PRELINK_PATH and PRELINK_PATH_MASK. For all other variables the latest defined value (in alphabetical order of the files in `/etc/env.d`) is used.

When you run env-update, the script will create all environment variables and place

them in `/etc/profile.env` (which is used by `/etc/profile`). It will also extract the information from the LDPATH variable and use that to create `/etc/ld.so.conf`. After this, it will run ldconfig to recreate the `/etc/ld.so.cache` file used by the dynamical linker.

If you want to notice the effect of env-update immediately after you run it, execute the following command to update your environment. Users who have installed Gentoo themselves will probably remember this from the installation instructions:

Code Listing 5: Updating the environment

```
# env-update && source /etc/profile
```

Note: The above command only updates the variables in your current terminal, *new* consoles, and their children. Thus, if you are working in X11, you will need to either type source /etc/profile in every new terminal you open or restart X so that all new terminals source the new variables. If you use a login manager, become root and type /etc/init.d/xdm restart. If not, you will need to logout and log back in for X to spawn children with the new variable values.

3: Defining Variables Locally

You do not always want to define an environment variable globally. For instance, you might want to add `/home/my_user/bin` and the current working directory (the directory you are in) to the PATH variable but don't want all other users on your system to have that in their PATH too. If you want to define an environment variable locally, you should use `~/.bashrc` or `~/.bash_profile`:

Code Listing 6: Extending PATH for local usage in ~/.bashrc

```
(A colon followed by no directory is treated as the current working directory)
PATH="${PATH}:/home/my_user/bin:"
```

When you relogin, your PATH variable will be updated.

. . : : Session Specific

Sometimes even stricter definitions are requested. You might want to be able to use binaries from a temporary directory you created without using the path to the binaries themselves or editing `~/.bashrc` for the short time you need it.

In this case, you can just define the PATH variable in your current session by using the export command. As long as you don't log out, the PATH variable will be using the temporary settings.

Code Listing 7: Defining a session-specific environment variable

```
# export PATH="${PATH}:/home/my_user/tmp/usr/bin"
```

Part 3

Working with Portage

"Working with Portage" provides an in-depth coverage of Portage, Gentoo's Software Management Tool.

Contents:

Files and Directories

“Once you want to know Portage in-depth you need to know where it stores its files and data.”

1: Portage Files

Portage comes with a default configuration stored in `/etc/make.globals`. When you take a look at it, you'll notice that all Portage configuration is handled through variables. What variables Portage listens to and what they mean are described later.

Since many configuration directives differ between architectures, Portage also has default configuration files which are part of your profile. Your profile is pointed to by the `/etc/make.profile` symlink; Portage' configurations are set in the `make.defaults` files of your profile and all parent profiles. We'll explain more about profiles and the `/etc/make.profile` directory later on.

If you're planning on changing a configuration variable, *don't* alter `/etc/make.globals` or `make.defaults`. Instead use `/etc/make.conf` which has precedence over the previous files. You'll also find a `/etc/make.conf.example`. As the name implies, this is merely an example file - Portage does not read in this file.

You can also define a Portage configuration variable as an environment variable, but we don't recommend this.

. . : : Profile-Specific Information

We've already encountered the `/etc/make.profile` directory. Well, this isn't exactly a directory but a symbolic link to a profile, by default one inside `/usr/portage/profiles` although you can create your own profiles elsewhere and point to them. The profile this symlink points to is the profile to which your system adheres.

A profile contains architecture-specific information for Portage, such as a list of packages that belong to the system corresponding with that profile, a list of

packages that don't work (or are masked-out) for that profile, etc.

. . : : User-Specific Configuration

When you need to override Portage's behaviour regarding the installation of software, you will end up editing files within `/etc/portage`. You are *highly recommended* to use files within `/etc/portage` and *highly discouraged* to override the behaviour through environment variables!

Within `/etc/portage` you can create the following files:

- `package.mask` which lists the packages you never want Portage to install
- `package.unmask` which lists the packages you want to be able to install even though the Gentoo developers highly discourage you from emerging them
- `package.keywords` which lists the packages you want to be able to install even though the package hasn't been found suitable for your system or architecture (yet)
- `package.use` which lists the USE flags you want to use for certain packages without having the entire system use those USE flags

More information about the `/etc/portage` directory and a full list of possible files you can create can be found in the Portage man page:

Code Listing 1: Reading the Portage man page

```
$ man portage
```

. . : : Changing Portage File & Directory Locations

The previously mentioned configuration files cannot be stored elsewhere - Portage will always look for those configuration files at those exact locations. However, Portage uses many other locations for various purposes: build directory, source code storage, Portage tree location, ...

All these purposes have well-known default locations but can be altered to your own taste through `/etc/make.conf`. The rest of this chapter explains what special-purpose locations Portage uses and how to alter their placement on your filesystem.

This document isn't meant to be used as a reference though. If you need 100% coverage, please consult the Portage and `make.conf` man pages:

Code Listing 2: Reading the Portage and make.conf man pages

```
$ man portage
$ man make.conf
```

2: Storing Files

The Portage tree default location is `/usr/portage`. This is defined by the PORTDIR variable. When you store the Portage tree elsewhere (by altering this variable), don't forget to change the `/etc/make.profile` symbolic link accordingly.

If you alter the PORTDIR variable, you might want to alter the following variables as well since they will not notice the PORTDIR change. This is due to how Portage handles variables: PKGDIR, DISTDIR, RPMDIR.

. . : : Prebuilt Binaries

Even though Portage doesn't use prebuilt binaries by default, it has extensive support for them. When you ask Portage to work with prebuilt packages, it will look for them in `/usr/portage/packages`. This location is defined by the PKGDIR variable.

. . : : Source Code

Application source code is stored in `/usr/portage/distfiles` by default. This location is defined by the DISTDIR variable.

. . : : RPM Files

Even though Portage cannot use RPM files, it is able to generate them using the ebuild command (see The Ebuild Application). The default location where Portage stores RPM files is `/usr/portage/rpm` and is defined by the RPMDIR variable.

. . : : Portage Database

Portage stores the state of your system (what packages are installed, what files belong to which package, ...) in `/var/db/pkg`. Do *not* alter these files manually! It might break Portage's knowledge of your system.

. . : : Portage Cache

The Portage cache (with modification times, virtuals, dependency tree information, ...) is stored in `/var/cache/edb`. This location really is a cache: you can clean it if you are not running any portage-related application at that moment.

3: Building Software

Portage's temporary files are stored in `/var/tmp` by default. This is defined by the PORTAGE_TMPDIR variable.

If you alter the PORTAGE_TMPDIR variable, you might want to alter the following variables as well since they will not notice the PORTAGE_TMPDIR change. This is due to how Portage handles variables: BUILD_PREFIX.

. . : : Building Directory

Portage creates specific build directories for each package it emerges inside `/var/tmp/portage`. This location is defined by the BUILD_PREFIX variable.

. . : : Live Filesystem Location

By default Portage installs all files on the current filesystem (`/`), but you can change this by setting the ROOT environment variable. This is useful when you want to create new build images.

4: Logging Features

Portage can create per-ebuild logfiles, but only when the PORT_LOGDIR variable is set to a location that is writable by Portage (the portage user). By default this variable is unset.

Configuring through Variables

"Portage is completely configurable through various variables you can set in the configuration file or as environment variable."

1: Portage Configuration

As noted previously, Portage is configurable through many variables which you should define in `/etc/make.conf`. Please refer to the `make.conf` man page for more and complete information:

Code Listing 1: Reading the make.conf man page

```
$ man make.conf
```

2: Build-specific Options

When Portage builds applications, it passes the contents of the following variables to the compiler and configure script:

- CFLAGS & CXXFLAGS define the desired compiler flags for C and C++ compiling.
- CHOST defines the build host information for the application's configure script
- MAKEOPTS is passed to the make command and is usually set to define the amount of parallelism used during the compilation. More information about the make options can be found in the make man page.

The USE variable is also used during configure and compilations but has been explained in great detail in previous chapters.

. . : : Merge Options

When Portage has merged a newer version of a certain software title, it will remove the obsoleted files of the older version from your system. Portage gives the user a 5 second delay before unmerging the older version. These 5 seconds are defined by the CLEAN_DELAY variable.

3: Configuration File Protection

Portage overwrites files provided by newer versions of a software title if the files aren't stored in a *protected* location. These protected locations are defined by the CONFIG_PROTECT variable and are generally configuration file locations. The directory listing is space-delimited.

A file that would be written in such a protected location is renamed and the user is warned about the presence of a newer version of the (presumable) configuration file.

You can find out about the current CONFIG_PROTECT setting from the emerge --info output:

Code Listing 2: Getting the CONFIG_PROTECT setting

```
$ emerge --info | grep 'CONFIG_PROTECT='
```

More information about Portage's Configuration File Protection is available through emerge:

Code Listing 3: More information about Configuration File Protection

```
$ emerge --help config
```

. . : : Excluding Directories

To 'unprotect' certain subdirectories of protected locations you can use the CONFIG_PROTECT_MASK variable.

4: Download Options

When the requested information or data is not available on your system, Portage will retrieve it from the Internet. The server locations for the various information and data channels are defined by the following variables:

- GENTOO_MIRRORS defines a list of server locations which contain source code (distfiles)
- PORTAGE_BINHOST defines a particular server location containing prebuilt packages for your system

A third setting involves the location of the rsync server which you use when you update your Portage tree:

- SYNC defines a particular server which Portage uses to fetch the Portage tree from

The GENTOO_MIRRORS and SYNC variables can be set automatically through the mirrorselect application. You need to emerge mirrorselect first before you can use it. For more information, see mirrorselect's online help:

Code Listing 4: More information about mirrorselect

```
# mirrorselect --help
```

If your environment requires you to use a proxy server, you can use the HTTP_PROXY, FTP_PROXY and RSYNC_PROXY variables to declare a proxy server.

. . : : Fetch Commands

When Portage needs to fetch source code, it uses wget by default. You can change this through the FETCHCOMMAND variable.

Portage is able to resume partially downloaded source code. It uses wget by default, but this can be altered through the RESUMECOMMAND variable.

Make sure that your FETCHCOMMAND and RESUMECOMMAND stores the source code in the correct location. Inside the variables you should use \${URI} and

\${DISTDIR} to point to the source code location and distfiles location respectively.

You can also define protocol-specific handlers with FETCHCOMMAND_HTTP, FETCHCOMMAND_FTP, RESUMECOMMAND_HTTP, RESUMECOMMAND_FTP, and so on.

. . : : Rsync Settings

You cannot alter the rsync command used by Portage to update the Portage tree, but you can set some variables related to the rsync command:

- RSYNC_EXCLUDEFROM points to a file listing the packages and/or categories rsync should ignore during the update process
- RSYNC_RETRIES defines how many times rsync should try connecting to the mirror pointed to by the SYNC variable before bailing out. This variable defaults to 3.
- RSYNC_TIMEOUT defines the number of seconds an rsync connection can idle before rsync sees the connection as timed-out. This variable defaults to 180 but dialup users or individuals with slow computers might want to set this to 300 or higher.

5: Gentoo Configuration

You can change your default branch with the ACCEPT_KEYWORDS variable. It defaults to your architecture's stable branch. More information on Gentoo's branches can be found in the next chapter.

. . : : Portage Features

You can activate certain Portage features through the FEATURES variable. The Portage Features have been discussed in previous chapters, such as Portage Features.

6: Portage Behaviour

With the PORTAGE_NICENESS variable you can augment or reduce the nice value Portage runs with. The PORTAGE_NICENESS value is *added* to the current nice value.

For more information about nice values, see the nice man page:

Code Listing 5: More information about nice

```
$ man nice
```

. . : : Output Behaviour

The NOCOLOR, which defaults to "false", defines if Portage should disable the use of coloured output.

Mixing Software Branches

“Gentoo provides software separated in several branches, depending on stability and architectural support. "Mixing Software Branches" inform you how these branches can be configured and how you can override this separation individually.”

1: Using One Branch

The ACCEPT_KEYWORDS variable defines what software branch you use on your system. It defaults to the stable software branch for your architecture, for instance x86.

We recommend that you only use the stable branch. However, if you don't care about stability this much and you want to help out Gentoo by submitting bugreports to http://bugs.gentoo.org, read on.

. . : : The Testing Branch

If you want to use more recent software, you can consider using the testing branch instead. To have Portage use the testing branch, add a ~ in front of your architecture.

The testing branch is exactly what it says - *Testing*. If a package is in testing, it means that the developers feel that it is functional but has not been thoroughly tested. You could very well be the first to discover a bug in the package in which case you could file a bugreport [http://bugs.gentoo.org] to let the developers know about it.

Beware though, you might notice stability issues, imperfect package handling (for instance wrong/missing dependencies), too frequent updates (resulting in lots of building) or broken packages. If you do not know how Gentoo works and how to solve problems, we recommend that you stick with the stable and tested branch.

For example, to select the testing branch for the x86 architecture, edit `/etc/make.conf` and set:

Code Listing 1: Setting the ACCEPT_KEYWORDS variable

```
ACCEPT_KEYWORDS="~x86"
```

If you update your system now, you will find out that *lots* of packages will be updated. Mind you though: when you have updated your system to use the testing branch there is usually no easy way back to the stable, official branch (except for using backups of course).

2: Mixing Stable with Testing

You can ask Portage to allow the testing branch for particular packages but use the stable branch for the rest of the system. To achieve this, add the package category and name you want to use the testing branch of in `/etc/portage/package.keywords`. For instance, to use the testing branch for gnumeric:

Code Listing 2: /etc/portage/package.keywords setting for gnumeric, full line

```
app-office/gnumeric ~x86
```

. . : : Test Particular Versions

If you want to use a specific software version from the testing branch but you don't want Portage to use the testing branch for subsequent versions, you can add in the version in the `package.keywords` file. In this case you *must* use the = operator. You can also enter a version range using the <=, <, > or >= operators.

In any case, if you add version information, you *must* use an operator. If you leave out version information, you *cannot* use an operator.

In the following example we ask Portage to accept gnumeric-1.2.13:

Code Listing 3: Enabling a particular gnumeric test version

```
=app-office/gnumeric-1.2.13 ~x86
```

3: Using Masked Packages

The Gentoo developers do **not** support the use of these files. Please exercise due caution when doing so. Support requests related to package.unmask and/or package.mask will not be answered. You have been warned.

When a package has been masked by the Gentoo developers and you still want to use it despite the reason mentioned in the `package.mask` file (situated in `/usr/portage/profiles` by default), add the *exact* same line in `/etc/portage/package.unmask`.

For instance, if =net-mail/hotwayd-0.8 is masked, you can unmask it by adding the exact same line in the `package.unmask` file:

Code Listing 4: /etc/portage/package.unmask

```
=net-mail/hotwayd-0.8
```

. . : : The package.mask file

When you don't want Portage to take a certain package or a specific version of a package into account you can mask it yourself by adding an appropriate line to `/etc/portage/package.mask`.

For instance, if you don't want Portage to install newer kernel sources than gentoo-sources-2.6.8.1, you add the following line to `package.mask`:

Code Listing 5: /etc/portage/package.mask example

```
>sys-kernel/gentoo-sources-2.6.8.1
```

Additional Portage Tools

"Portage comes with a few extra tools that might make your Gentoo experience even better. Read on to discover how to use dispatch-conf and other tools."

1: etc-update

etc-update is a tool that aids in merging the `._cfg0000_<name>` files. It provides an interactive merging setup and can also auto-merge trivial changes. `._cfg0000_<name>` files are generated by Portage when it wants to overwrite a file in a directory protected by the CONFIG_PROTECT variable.

Running etc-update is pretty straight-forward:

Code Listing 1: Running etc-update

```
# etc-update
```

After merging the straightforward changes, you will be prompted with a list of protected files that have an update waiting. At the bottom you are greeted by the possible options:

Code Listing 2: etc-update options

```
Please select a file to edit by entering the corresponding number.
              (-1 to exit) (-3 to auto merge all remaining files)
                           (-5 to auto-merge AND not use 'mv -i'):
```

If you enter -1, etc-update will exit and discontinue any further changes. If you enter -3 or -5, *all* listed configuration files will be overwritten with the newer versions. It is therefore very important to first select the configuration files that should not be automatically updated. This is simply a matter of entering the number listed to the left of that configuration file.

As an example, we select the configuration file `/etc/pear.conf`:

Code Listing 3: Updating a specific configuration file

```
Beginning of differences between /etc/pear.conf and /etc/._cfg0000_pear.conf
[...]
End of differences between /etc/pear.conf and /etc/._cfg0000_pear.conf
1) Replace original with update
2) Delete update, keeping original as is
3) Interactively merge original with update
4) Show differences again
```

You can now see the differences between the two files. If you believe that the updated configuration file can be used without problems, enter 1. If you believe that the updated configuration file isn't necessary, or doesn't provide any new or useful information, enter 2. If you want to interactively update your current configuration file, enter 3.

There is no point in further elaborating the interactive merging here. For completeness sake, we will list the possible commands you can use while you are interactively merging the two files. You are greeted with two lines (the original one, and the proposed new one) and a prompt at which you can enter one of the following commands:

Code Listing 4: Commands available for the interactive merging

```
ed:     Edit then use both versions, each decorated with a header.
eb:     Edit then use both versions.
el:     Edit then use the left version.
er:     Edit then use the right version.
e:      Edit a new version.
l:      Use the left version.
r:      Use the right version.
s:      Silently include common lines.
v:      Verbosely include common lines.
q:      Quit.
```

When you have finished updating the important configuration files, you can now automatically update all the other configuration files. etc-update will exit if it doesn't find any more updateable configuration files.

2: dispatch-conf

Using dispatch-conf you are able to merge updates to your configuration files while keeping track of all changes. dispatch-conf stores the differences between the configuration files as patches or by using the RCS revision system.

Like etc-update, you can ask to keep the configuration file as-is, use the new configuration file, edit the current one or merge the changes interactively. However, dispatch-conf also has some nice additional features:

- Automatically merge configuration file updates that only contain updates to comments
- Automatically merge configuration files which only differ in the amount of whitespace

Make certain you edit `/etc/dispatch-conf.conf` first and create the directory referenced by the archive-dir variable.

For more information, check out the dispatch-conf man page:

Code Listing 5: Reading the dispatch-conf man page

```
$ man dispatch-conf
```

3: quickpkg

With quickpkg you can create archives of the packages that are already merged on your system. These archives can be used as prebuilt packages. Running quickpkg is straightforward: just add the names of the packages you want to archive.

For instance, to archive curl, arts and procps:

Code Listing 6: Example quickpkg usage

```
# quickpkg curl arts procps
```

The prebuilt packages will be stored in `$PKGDIR/All` (`/usr/portage/packages/All` by default). Symbolic links pointing to these packages are placed in `$PKGDIR/<category>`.

Diverting from the Official Tree

“"Diverting from the Official Tree" gives you some tips and tricks on how to use your own Portage tree, how to synchronise only the categories you want, inject packages and more.”

1: Using a Portage Tree Subset

You can selectively update certain categories/packages and ignore the other categories/packages. We achieve this by having rsync exclude categories/packages during the emerge --sync step.

You need to define the name of the file that contains the exclude patterns in the RSYNC_EXCLUDEFROM variable in your `/etc/make.conf`.

Code Listing 1: Defining the exclude file in /etc/make.conf

```
RSYNC_EXCLUDEFROM=/etc/portage/rsync_excludes
```

Code Listing 2: Excluding all games in /etc/portage/rsync_excludes

```
games-*/*
```

Note however that this may lead to dependency issues since new, allowed packages might depend on new but excluded packages.

2: Adding Unofficial Ebuilds

You can ask Portage to use ebuilds that are not officially available through the Portage tree. Create a new directory (for instance `/usr/local/portage`) in which you store the 3rd-party ebuilds. Use the same directory structure as the official Portage tree!

Then define PORTDIR_OVERLAY in `/etc/make.conf` and have it point to the previously defined directory. When you use Portage now, it will take those ebuilds

into account as well without removing/overwriting those ebuilds the next time you run emerge --sync.

. . : : Working with Several Overlays

For the powerusers who develop on several overlays, test packages before they hit the Portage tree or just want to use unofficial ebuilds from various sources, the app-portage/gentoolkit-dev package brings you gensync, a tool to help you keep the overlay repositories up to date.

With gensync you can update all the repositories at once, or select just a few of them. Each repository should have a `.syncsource` file in the `/etc/gensync/` configuration directory which contains the repository location, name, ID, etc.

Suppose you have two additional repositories called java (for the in-development java ebuilds) and entapps (for the applications developed in-house for your enterprise). You can update those repositories with the following command:

Code Listing 3: Using gensync to update a few repositories

```
# gensync java entapps
```

3: Non-Portage Maintained Software

In some cases you want to configure, install and maintain software yourself without having Portage automate the process for you, even though Portage can provide the software titles. Known cases are kernel sources and nvidia drivers. You can configure Portage so it knows that a certain package is manually installed on your system. This process is called *injecting* and supported by Portage through the `/etc/portage/profile/package.provided` file.

For instance, if you want to inform Portage about vanilla-sources-2.6.11.6 which you've installed manually, add the following line to `/etc/portage/profile/package.provided`:

Code Listing 4: Example line for package.provided

```
sys-kernel/vanilla-sources-2.6.11.6
```

The Ebuild Application

"In "The Ebuild Application" you are informed about the steps Portage takes while installing software and how you can do this yourself using the ebuild application."

1: Emerge and Ebuild

The ebuild application is a lower level interface to the Portage system. Using this application you can execute specific actions against a given ebuild. For instance, you can perform the individual merging steps by yourself.

Using ebuild is more for development purposes; more information about ebuild can therefore be found in the Developers Handbook [http://www.gentoo.org/proj/en/devrel/handbook/handbook.xml]. However, we will explain which ebuild instances are invoked by Portage during the merge process of a certain software title, and how to invoke the post-configuration steps some ebuilds allow you to perform.

2: Manually Installing Software

Whenever you invoke ebuild against a given ebuild file, it will verify if the checksums of all involved files are equal to those given in the accompanying `Manifest` or `files/digest-<name>-<version>` file. This happens after the sources have been fetched.

To fetch the sources using ebuild, run:

Code Listing 1: Fetching the sources

```
# ebuild path/to/ebuild fetch
```

If the ebuild's md5sum does not match the one listed in the `Manifest` file, or one of the downloaded sources don't match those listed in the `files/digest-<package>` file, you will receive an error similar to this:

Code Listing 2: Ebuild checksum failure

```
!!! File is corrupt or incomplete. (Digests do not match)
>>> our recorded digest: db20421ce35e8e54346e3ef19e60e4ee
>>>  your file's digest: f10392b7c0b2bbc463ad09642606a7d6
```

The subsequent line will mention the erroneous file.

If you are certain that the sources you've fetched and the ebuild itself are valid, you can regenerate the `Manifest` and `digest-<package>` file using ebuild's digest functionality:

Code Listing 3: Regenerate Manifest and digest

```
# ebuild path/to/ebuild digest
```

. . : : Unpacking the Sources

To unpack the sources in `/var/tmp/portage` (or any other directory location you have specified in `/etc/make.conf`), run ebuild's unpack functionality:

Code Listing 4: Unpacking the sources

```
# ebuild path/to/ebuild unpack
```

This will execute the ebuild's src_unpack() function (which defaults to plain extraction if no src_unpack() function is defined). It is also in this step that all necessary patches are applied.

. . : : Compiling the Sources

The next step in the merge process is to compile the sources. The ebuild's compile functionality takes care of this step by executing the src_compile() function in the ebuild. This also includes the configure steps if appropriate.

Code Listing 5: Compiling the sources

```
# ebuild path/to/ebuild compile
```

You are advised to edit the ebuild's src_compile() function if you want to change the compilation instructions. However, you can also trick Portage into believing that the ebuild application has finished the compile steps. Run all necessary commands yourself and create an empty file called `.compiled` in the working directory:

Code Listing 6: Informing Portage about the finished compilation jobs

```
# touch .compiled
```

. . : : Installing the Files in a Temporary Location

In the next step Portage will install all necessary files in a temporary location. This directory will then contain all files that are to be merged on the live filesystem. You can accomplish this by running ebuild's install functionality, which executes the ebuild's src_install() function:

Code Listing 7: Installing the files

```
# ebuild path/to/ebuild install
```

. . : : Merging the Files onto the Live Filesystem

The final step is to merge all files onto the live filesystem and register those in the Portage backend. ebuild calls this step "qmerge" and involves the following steps:

- Execute the pkg_preinst() function if specified
- Copy over all files to the live filesystem
- Register the files in the Portage backend
- Execute the pkg_postinst() function if specified

Run ebuild's qmerge functionality to accomplish these steps:

Code Listing 8: Merging the files on the live filesystem

```
# ebuild path/to/ebuild qmerge
```

. . : : Cleaning the Temporary Directory

Finally you can clean the temporary directory using ebuild's clean functionality:

Code Listing 9: Cleaning the temporary directory

```
# ebuild path/to/ebuild clean
```

3: Additional Ebuild Features

Using ebuild's merge functionality you can run the fetch, unpack, compile, install and qmerge commands in one go:

Code Listing 10: Installing software

```
# ebuild path/to/ebuild merge
```

. . : : Performing Configuration Actions

Some applications include instructions that configure the package further on your system. These instructions can be interactive and are therefore not automatically executed. To run these configuration steps, which are enlisted in the ebuild's (optional) config() function, use ebuild's config functionality:

Code Listing 11: Configuring a package

```
# ebuild path/to/ebuild config
```

. . : : Building an (RPM) Package

You can instruct Portage to create a binary package of an ebuild or even an RPM file. Use ebuild's package or rpm functionality to create these archives. There are a few differences between those functionalities though:

- The package functionality is a lot like the merge functionality, executing all necessary steps (fetch, unpack, compile, install) before creating the package
- The rpm functionality builds an RPM package from the files created *after* having run ebuild's install functionality

Code Listing 12: Creating packages

```
(For a Portage-compatible binary package)
# ebuild path/to/ebuild package

(For an RPM package)
# ebuild path/to/ebuild rpm
```

The created RPM file however does not contain the ebuild's dependency information.

4: More Information

Please consult the following man pages for more information about Portage, the ebuild application and the ebuild files:

Code Listing 13: Man pages

```
$ man portage    (Portage itself)
$ man emerge     (The emerge command)
$ man ebuild     (The ebuild command)
$ man 5 ebuild   (The ebuild file syntax)
```

You will also find more development-related information in the Developers Handbook [http://www.gentoo.org/proj/en/devrel/handbook/handbook.xml].

Part 4

Gentoo Network Configuration

A comprehensive guide to Networking in Gentoo.

Contents:

Getting Started

"A guide to quickly get your network interface up and running in most common environments."

1: Getting started

Note: This document assumes that you have correctly configured your kernel, its modules for your hardware and you know the interface name of your hardware. We also assume that you are configuring eth0, but it could also be eth1, wlan0, etc.

Note: This document requires you are running baselayout-1.11.11 or better.

To get started configuring your network card, you need to tell the Gentoo RC system about it. This is done by creating a symbolic link from `net.lo` to `net.eth0` in `/etc/init.d`.

Code Listing 1: Symlinking net.eth0 to net.lo

```
# cd /etc/init.d
# ln -s net.lo net.eth0
```

Gentoo's RC system now knows about that interface. It also needs to know how to configure the new interface. All the network interfaces are configured in `/etc/conf.d/net`. Below is a sample configuration for DHCP and static addresses.

Code Listing 2: Examples for /etc/conf.d/net

```
# For DHCP
config_eth0=( "dhcp" )

# For static IP using CIDR notation
config_eth0=( "192.168.0.7/24" )
routes_eth0=( "default via 192.168.0.1" )

# For static IP using netmask notation
config_eth0=( "192.168.0.7 netmask 255.255.255.0" )
routes_eth0=( "default gw 192.168.0.1" )
```

Note: If you do not specify a configuration for your interface then DHCP is assumed.

Note: CIDR stands for Classless InterDomain Routing. Originally, IPv4 addresses were classified as A, B, or C. The early classification system did not envision the massive popularity of the Internet, and is in danger of running out of new unique addresses. CIDR is an addressing scheme that allows one IP address to designate many IP addresses. A CIDR IP address looks like a normal IP address except that it ends with a slash followed by a number; for example, 192.168.0.0/16. CIDR is described in RFC 1519 [http://rfc.net/rfc1519.html].

Now that we have configured our interface, we can start and stop it using the below commands

Code Listing 3: Starting and stopping network scripts

```
# /etc/init.d/net.eth0 start
# /etc/init.d/net.eth0 stop
```

Important: When troubleshooting networking, it is recommended to set RC_VERBOSE="yes" in /etc/conf.d/rc so that you get more information about what's happening.

Now that you have successfully started and stopped your network interface, you may wish to get it to start when Gentoo boots. Here's how to do this. The last "rc" command instructs Gentoo to start any scripts in the current runlevel that have not yet been started.

Code Listing 4: Configuring a network interface to load at boot time

```
# rc-update add net.eth0 default
# rc
```

Advanced Configuration

"Here we learn about how the configuration works - you need to know this before we learn about modular networking."

1: Advanced Configuration

The config_eth0 variable is the heart of an interface configuration. It's a high level instruction list for configuring the interface (eth0 in this case). Each command in the instruction list is performed sequentially. The interface is deemed OK if at least one command works.

Here's a list of in-built instructions.

Command	*Description*
null	Do nothing
noop	If the interface is up and there is an address then abort configuration successfully
an IPv4 or IPv6 address	Add the address to the interface
dhcp, adsl or apipa (or a custom command from a 3rd party module)	Run the module which provides the command. For example dhcp will run a module that provides DHCP which can be one of either dhcpcd, udhcpc, dhclient or pump.

If a command fails, you can specify a fallback command. The fallback has to match the config structure exactly.

You can chain these commands together. Here are some real world examples.

Code Listing 1: Configuration examples

```
# Adding three IPv4 addresses
config_eth0=(
  "192.168.0.2/24"
  "192.168.0.3/24"
  "192.168.0.4/24"
)

# Adding an IPv4 address and two IPv6 addresses
config_eth0=(
  "192.168.0.2/24"
  "4321:0:1:2:3:4:567:89ab"
  "4321:0:1:2:3:4:567:89ac"
)

# Keep our kernel assigned address, unless the interface goes
# down so assign another via DHCP. If DHCP fails then add a
# static address determined by APIPA
config_eth0=(
  "noop"
  "dhcp"
)
fallback_eth0=(
  "null"
  "apipa"
)
```

Note: When using the ifconfig module and adding more than one address, interface aliases are created for each extra address. So with the above two examples you will get interfaces eth0, eth0:1 and eth0:2. You cannot do anything special with these interfaces as the kernel and other programs will just treat eth0:1 and eth0:2 as eth0.

Important: The fallback order is important! If we did not specify the null option then the apipa command would only be run if the noop command failed.

Note: APIPA and DHCP are discussed later.

2: Network Dependencies

Init scripts in `/etc/init.d` can depend on a specific network interface or just net. net can be defined in `/etc/conf.d/rc` to mean different things using the RC_NET_STRICT_CHECKING variable.

Value	*Description*
none	The `net` service is always considered up
no	This basically means that at least one `net.*` service besides `net.lo` must be up. This can be used by notebook users that have a WIFI and a static NIC, and only wants one up at any given time to have the net service seen as up.
lo	This is the same as the no option, but `net.lo` is also counted. This should be useful to people that do not care about any specific interface being up at boot.
yes	For this ALL network interfaces MUST be up for the `net` service to be considered up.

But what about `net.br0` depending on `net.eth0` and `net.eth1`? `net.eth1` may be a wireless or PPP device that needs configuration before it can be added to the bridge. This cannot be done in `/etc/init.d/net.br0` as that's a symbolic link to `net.lo`.

The answer is making your own depend() function in `/etc/conf.d/net`.

Code Listing 2: net.br0 dependency in /etc/conf.d/net

```
# You can use any dependency (use, after, before) as found in current scripts
depend_br0() {
  need net.eth0 net.eth1
}
```

For a more detailed discussion about dependency, consult the section Writing Init Scripts in the Gentoo Handbook.

3: Variable names and values

Variable names are dynamic. They normally follow the structure of variable_${interface|mac|essid|apmac}. For example, the variable dhcpcd_eth0 holds the value for dhcpcd options for eth0 and dhcpcd_essid holds the value for dhcpcd options when any interface connects to the ESSID "essid".

However, there is no hard and fast rule that states interface names must be ethx. In fact, many wireless interfaces have names like wlanx, rax as well as ethx. Also, some user defined interfaces such as bridges can be given any name, such as foo. To make life more interesting, wireless Access Points can have names with non alpha-numeric characters in them - this is important because you can configure networking parameters per ESSID.

The downside of all this is that Gentoo uses bash variables for networking - and

bash cannot use anything outside of English alpha-numerics. To get around this limitation we change every character that is not an English alpha-numeric into a _ character.

Another downside of bash is the content of variables - some characters need to be escaped. This can be achived by placing the \ character in front of the character that needs to be escaped. The following list of characters needs to be escaped in this way: ", ' and \.

In this example we use wireless ESSID as they can contain the widest scope of characters. We shall use the ESSID My "\ NET:

Code Listing 3: variable name example

```
# This does work, but the domain is invalid
dns_domain_My____NET="My \"\\ NET"

# The above sets the dns domain to My "\ NET when a wireless card
# connects to an AP whose ESSID is My "\ NET
```

Modular Networking

"Gentoo provides you flexible networking - here you are told about choosing different DHCP clients, setting up bonding, bridging, VLANs and more."

1: Network Modules

We now support modular networking scripts, which means we can easily add support for new interface types and configuration modules while keeping compatibility with existing ones.

Modules load by default if the package they need is installed. If you specify a module here that doesn't have its package installed then you get an error stating which package you need to install. Ideally, you only use the modules setting when you have two or more packages installed that supply the same service and you need to prefer one over the other.

Code Listing 1: Module preference

```
# Prefer iproute2 over ifconfig
modules=( "iproute2" )

# You can also specify other modules for an interface
# In this case we prefer udhcpc over dhcpcd
modules_eth0=( "udhcpc" )

# You can also specify which modules not to use - for example you may be
# using a supplicant or linux-wlan-ng to control wireless configuration but
# you still want to configure network settings per ESSID associated with.
modules=( "!iwconfig" )
```

2: Interface Handlers

We provide two interface handlers presently: ifconfig and iproute2. You need one of these to do any kind of network configuration.

ifconfig is the current Gentoo default and it's included in the system profile. iproute2 is a more powerful and flexible package, but it's not included by default.

Code Listing 2: To install iproute2

```
# emerge sys-apps/iproute2

# To prefer iproute2 over ifconfig if both are installed
modules=( "iproute2" )
```

As both ifconfig and iproute2 do very similar things we allow their basic configuration to work with each other. For example both the below code snippet work regardless of which module you are using.

Code Listing 3: ifconfig and iproute2 examples

```
config_eth0=( "192.168.0.2/24" )
config_eth0=( "192.168.0.2 netmask 255.255.255.0" )

# We can also specify broadcast
config_eth0=( "192.168.0.2/24 brd 192.168.0.255" )
config_eth0=( "192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255" )
```

3: DHCP

DHCP is a means of obtaining network information (IP address, DNS servers, Gateway, etc) from a DHCP server. This means that if there is a DHCP server running on the network, you just have to tell each client to use DHCP and it sets up the network all by itself. Of course, you will have to configure for other things like wireless, PPP or other things if required before you can use DHCP.

DHCP can be provided by dhclient, dhcpcd, pump or udhcpc. Each DHCP module has its pros and cons - here's a quick run down.

DHCP Module	*Package*	*Pros*	*Cons*
dhclient	net-misc/dhcp	Made by ISC, the same people who make the BIND DNS software. Very configurable	Configuration is overly complex, software is quite bloated, cannot get NTP servers from DHCP, does not send hostname by default
dhcpcd	net-misc/dhcpcd	Long time Gentoo default, no reliance on outside tools	No longer maintained upstream, can be slow at times, does not daemonize when lease is infinite
pump	net-misc/pump	Lightweight, no reliance	No longer maintained upstream,

		on outside tools	unreliable, especially over modems, cannot get NIS servers from DHCP
udhcpc	net-misc/udhcp	Lightweight - smallest DHCP client around, made for embedded systems	Unproven - no distro uses it by default, cannot define a timeout beyond 3 seconds

If you have more than one DHCP client installed, you need to specify which one to use - otherwise we default to dhcpcd if available.

To send specific options to the DHCP module, use module_eth0="..." *(change module to the DHCP module you're using - ie dhcpcd_eth0)*.

We try and make DHCP relatively agnostic - as such we support the following commands using the dhcp_eth0 variable. The default is not to set any of them:

- release - releases the IP address for re-use
- nodns - don't overwrite `/etc/resolv.conf`
- nontp - don't overwrite `/etc/ntp.conf`
- nonis - don't overwrite `/etc/yp.conf`

Code Listing 4: Sample DHCP configuration in /etc/conf.d/net

```
# Only needed if you have more than one DHCP module installed
modules=( "dhcpcd" )

config_eth0=( "dhcp" )
dhcpcd_eth0="-t 10" # Timeout after 10 seconds
dhcp_eth0="release nodns nontp nonis" # Only get an address
```

Note: dhcpcd, udhcpc and pump send the current hostname to the DHCP server by default so you don't need to specify this anymore.

4: ADSL Modem

First we need to install the ADSL software.

Code Listing 5: Install the rp-pppoe package

```
# emerge net-dialup/rp-pppoe
```

Warning: baselayout-1.11.x supports PPPoE only. Hopefully future versions will support PPPoA.

Now we need to instruct configure eth0 to be an ADSL interface and enter our username.

Code Listing 6: Configure eth0 for ADSL

```
config_eth0=( "adsl" )
adsl_user_eth0="username"
```

Finally you need to define your username and password in `/etc/ppp/pap-secrets`.

Code Listing 7: sample /etc/ppp/pap-secrets

```
# The * is important
"username"  *  "password"
```

5: APIPA (Automatic Private IP Addressing)

APIPA tries to find a free address in the range 169.254.0.0-169.254.255.255 by arping a random address in that range on the interface. If no reply is found then we assign that address to the interface.

This is only useful for LANs where there is no DHCP server and you don't connect directly to the internet and all other computers use APIPA.

For APIPA support, emerge net-misc/iputils or net-analyzer/arping.

Code Listing 8: APIPA configuration in /etc/conf.d/net

```
# Try DHCP first - if that fails then fallback to APIPA
config_eth0=( "dhcp" )
fallback_eth0=( "apipa" )

# Just use APIPA
config_eth0=( "apipa" )
```

6: Bonding

For link bonding/trunking emerge net-misc/ifenslave.

Bonding is used to increase network bandwidth. If you have two network cards going to the same network, you can bond them together so your applications see just one interface but they really use both network cards.

Code Listing 9: bonding configuration in /etc/conf.d/net

```
# To bond interfaces together
slaves_bond0="eth0 eth1 eth2"

# You may not want to assign an IP to the bonded interface
config_bond0=( "null" )

# Depend on eth0, eth1 and eth2 as they may require extra configuration
depend_bond0() {
  need net.eth0 net.eth1 net.eth2
}
```

7: Bridging (802.1d support)

For bridging support emerge net-misc/bridge-utils.

Bridging is used to join networks together. For example, you may have a server that connects to the internet via an ADSL modem and a wireless access card to enable other computers to connect to the internet via the ADSL modem. You could create a bridge to join the two interfaces together.

Code Listing 10: Bridge configuration in /etc/conf.d/net

```
# Configure the bridge - "man btctl" for more details
brctl_br0=( "setfd 0" "sethello 0" "stp off" )

# To add ports to bridge br0
bridge_br0="eth0 eth1"

# You need to configure the ports to null values so dhcp does not get started
config_eth0=( "null" )
config_eth1=( "null" )

# Finally give the bridge an address - you could use DHCP as well
config_br0=( "192.168.0.1/24" )

# Depend on eth0 and eth1 as they may require extra configuration
depend_br0() {
  need net.eth0 net.eth1
}
```

Important: For using some bridge setups, you may need to consult the variable name documentation.

8: MAC Address

You don't need to emerge anything for changing the MAC address of your interface if you have sys-apps/baselayout-1.11.14 or newer and want to change to a specific MAC address. However, if you need to change to a random MAC address or have a baselayout older than the version mentioned above, you have to emerge net-analyzer/macchanger to be able to make use of this feature.

Code Listing 11: MAC Address change example

```
# To set the MAC address of the interface
mac_eth0="00:11:22:33:44:55"

# To randomize the last 3 bytes only
mac_eth0="random-ending"

# To randomize between the same physical type of connection (eg fibre,
# copper, wireless) , all vendors
mac_eth0="random-samekind"

# To randomize between any physical type of connection (eg fibre, copper,
# wireless) , all vendors
mac_eth0="random-anykind"

# Full randomization - WARNING: some MAC addresses generated by this may
# NOT act as expected
mac_eth0="random-full"
```

9: Tunnelling

You don't need to emerge anything for tunnelling as the interface handler can do it for you.

Code Listing 12: Tunnelling configuration in /etc/conf.d/net

```
# For GRE tunnels
iptunnel_vpn0="mode gre remote 207.170.82.1 key 0xffffffff ttl 255"

# For IPIP tunnels
iptunnel_vpn0="mode ipip remote 207.170.82.2 ttl 255"

# To configure the interface
config_vpn0=( "192.168.0.2 peer 192.168.1.1" )
```

10: VLAN (802.1q support)

For VLAN support, emerge net-misc/vconfig.

Virtual LAN is a group of network devices that behave as if they were connected to

a single network segment - even though they may not be. VLAN members can only see members of the same VLAN even though they may share the same physical network.

Code Listing 13: VLAN configuration in /etc/conf.d/net

```
# Specify the VLAN numbers for the interface like so
# Please ensure your VLAN IDs are NOT zero-padded
vlans_eth0="1 2"

# You can also configure the VLAN
# see for vconfig man page for more details
vconfig_eth0=( "set_name_type VLAN_PLUS_VID_NO_PAD" )
vconfig_vlan1=( "set_flag 1" "set_egress_map 2 6" )

# Configure the interface as usual
config_vlan1=( "172.16.3.1 netmask 255.255.254.0" )
config_vlan2=( "172.16.2.1 netmask 255.255.254.0" )
```

Important: For using some VLAN setups, you may need to consult the variable name documentation.

Wireless Networking

“Wireless isn't straight-forward. Hopefully we'll get you working!”

1: Introduction

Currently we support wireless setup either by wireless-tools or wpa_supplicant. The important thing to remember is that you configure for wireless networks on a global basis and not an interface basis.

wpa_supplicant is the best choice, but it does not support all drivers. For a list of supported drivers, read the wpa_supplicant site [http://hostap.epitest.fi/wpa_supplicant]. Also, wpa_supplicant can currently only connect to SSID's that you have configured for.

wireless-tools supports nearly all cards and drivers, but it cannot connect to WPA only Access Points.

Warning: The linux-wlan-ng driver is not supported by baselayout at this time. This is because linux-wlan-ng have their own setup and configuration which is completely different to everyone else's. The linux-wlan-ng developers are rumoured to be changing their setup over to wireless-tools - when this happens you may use linux-wlan-ng with baselayout.

2: WPA Supplicant

WPA Supplicant [http://hostap.epitest.fi/wpa_supplicant] is a package that allows you to connect to WPA enabled access points. It's setup is fairly fluid as it is still in beta - however it works fine for the most part.

Code Listing 1: Install wpa_supplicant

```
# emerge net-wireless/wpa_supplicant
```

Important: You have to have CONFIG_PACKET enabled in your kernel for wpa_supplicant to work.

Now we have to configure `/etc/conf.d/net` to so that we prefer wpa_supplicant over wireless-tools (if both are installed, wireless-tools is the default).

Code Listing 2: configure /etc/conf.d/net for wpa_supplicant

```
# Prefer wpa_supplicant over wireless-tools
modules=( "wpa_supplicant" )

# It's important that we tell wpa_supplicant which driver we should
# be using as it's not very good at guessing yet
wpa_supplicant_eth0="-Dmadwifi"
```

Note: If you're using the host-ap driver you will need to put the card in *Managed mode* before it can be used with wpa_supplicant correctly. You can use iwconfig_eth0="mode managed" to achieve this in `/etc/conf.d/net`.

That was simple, wasn't it? However, we still have to configure wpa_supplicant itself which is a bit more tricky depending on how secure the Access Points are that you are trying to connect to. The below example is taken and simplified from `/etc/wpa_supplicant.conf.example` which ships with wpa_supplicant.

Code Listing 3: an example /etc/wpa_supplicant.conf

```
# The below line not be changed otherwise we refuse to work
ctrl_interface=/var/run/wpa_supplicant

# Ensure that only root can read the WPA configuration
ctrl_interface_group=0

# Let wpa_supplicant take care of scanning and AP selection
ap_scan=1

# Simple case: WPA-PSK, PSK as an ASCII passphrase, allow all valid ciphers
network={
  ssid="simple"
  psk="very secret passphrase"
  # The higher the priority the sooner we are matched
  priority=5
}

# Same as previous, but request SSID-specific scanning (for APs that reject
# broadcast SSID)
network={
  ssid="second ssid"
  scan_ssid=1
  psk="very secret passphrase"
  priority=2
}

# Only WPA-PSK is used. Any valid cipher combination is accepted
network={
  ssid="example"
  proto=WPA
  key_mgmt=WPA-PSK
  pairwise=CCMP TKIP
  group=CCMP TKIP WEP104 WEP40
  psk=06b4be19da289f475aa46a33cb793029d4ab3db7a23ee92382eb0106c72ac7bb
  priority=2
}

# Plaintext connection (no WPA, no IEEE 802.1X)
network={
  ssid="plaintext-test"
  key_mgmt=NONE
}

# Shared WEP key connection (no WPA, no IEEE 802.1X)
network={
  ssid="static-wep-test"
  key_mgmt=NONE
  wep_key0="abcde"
  wep_key1=0102030405
  wep_key2="1234567890123"
  wep_tx_keyidx=0
  priority=5
}

# Shared WEP key connection (no WPA, no IEEE 802.1X) using Shared Key
```

```
# IEEE 802.11 authentication
network={
  ssid="static-wep-test2"
  key_mgmt=NONE
  wep_key0="abcde"
  wep_key1=0102030405
  wep_key2="1234567890123"
  wep_tx_keyidx=0
  priority=5
  auth_alg=SHARED
}

# IBSS/ad-hoc network with WPA-None/TKIP
network={
  ssid="test adhoc"
  mode=1
  proto=WPA
  key_mgmt=WPA-NONE
  pairwise=NONE
  group=TKIP
  psk="secret passphrase"
}
```

3: Wireless Tools

Wireless Tools [http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html] provide a generic way to configure basic wireless interfaces up to the WEP security level. While WEP is a weak security method it's also the most prevalent.

Wireless Tools configuration is controlled by a few main variables. The sample configuration file below should describe all you need. One thing to bear in mind is that no configuration means "connect to the strongest unencrypted Access Point" - we will always try and connect you to something.

Code Listing 4: Install wireless-tools

```
# emerge net-wireless/wireless-tools
```

Note: Although you can store your wireless settings in `/etc/conf.d/wireless` this guide recommends you store them in `/etc/conf.d/net`.

Important: You will need to consult the variable name documentation.

Code Listing 5: sample iwconfig setup in /etc/conf.d/net

```
# Prefer iwconfig over wpa_supplicant
modules=( "iwconfig" )

# Configure WEP keys for Access Points called ESSID1 and ESSID2
# You may configure up to 4 WEP keys, but only 1 can be active at
# any time so we supply a default index of [1] to set key [1] and then
# again afterwards to change the active key to [1]
# We do this incase you define other ESSID's to use WEP keys other than 1
#
# Prefixing the key with s: means it's an ASCII key, otherwise a HEX key
#
# enc open specified open security (most secure)
# enc restricted specified restricted security (least secure)
key_ESSID1="[1] s:yourkeyhere key [1] enc open"
key_ESSID2="[1] aaaa-bbbb-cccc-dd key [1] enc restricted"

# The below only work when we scan for available Access Points

# Sometimes more than one Access Point is visible so we need to
# define a preferred order to connect in
preferred_aps=( "ESSID1" "ESSID2" )
```

. . : : Fine tune Access Point Selection

You can add some extra options to fine-tune your Access Point selection, but these are not normally required.

You can decide whether we only connect to preferred Access Points or not. By default if everything configured has failed and we can connect to an unencrypted Access Point then we will. This can be controlled by the associate_order variable. Here's a table of values and how they control this.

Value	*Description*
any	Default behaviour
preferredonly	We will only connect to visible APs in the preferred list
forcepreferred	We will forceably connect to APs in the preferred order if they are not found in a scan
forcepreferredonly	Do not scan for APs - instead just try to connect to each one in order
forceany	Same as forcepreferred + connect to any other available AP

Finally we have some blacklist_aps and unique_ap selection. blacklist_aps works in a similar way to preferred_aps. unique_ap is a yes or no value that says if a second wireless interface can connect to the same Access Point as the first interface.

Code Listing 6: blacklist_aps and unique_ap example

```
# Sometimes you never want to connect to certain access points
blacklist_aps=( "ESSID3" "ESSID4" )

# If you have more than one wireless card, you can say if you want
# to allow each card to associate with the same Access Point or not
# Values are "yes" and "no"
# Default is "yes"
unique_ap="yes"
```

. . : : Ad-Hoc and Master Modes

If you want to set yourself up as an Ad-Hoc node if you fail to connect to any Access Point in managed mode, you can do that too.

Code Listing 7: fallback to ad-hoc mode

```
adhoc_essid_eth0="This Adhoc Node"
```

What about connecting to Ad-Hoc networks or running in Master mode to become an Access Point? Here's a configuration just for that! You may need to specify WEP keys as shown above.

Code Listing 8: sample ad-hoc/master configuration

```
# Set the mode - can be managed (default), ad-hoc or master
# Not all drivers support all modes
mode_eth0="ad-hoc"

# Set the ESSID of the interface
# In managed mode, this forces the interface to try and connect to the
# specified ESSID and nothing else
essid_eth0="This Adhoc Node"

# We use channel 3 if you don't specify one
channel_eth0="9"
```

Important: The below is taken verbatim from the BSD wavelan documentation found at the NetBSD documentation. There are 14 channels possible; We are told that channels 1-11 are legal for North America, channels 1-13 for most of Europe, channels 10-13 for France, and only channel 14 for Japan. If in doubt, please refer to the documentation that came with your card or access point. Make sure that the channel you select is the same channel your access point (or the other card in an ad-hoc network) is on. The default for cards sold in North America and most of Europe is 3; the default for cards sold in France is 11, and the default for cards sold in Japan is 14.

. . : : Troubleshooting Wireless Tools

There are some more variables you can use to help get your wireless up and running due to driver or environment problems. Here's a table of other things you can try.

Variable	*Default Value*	*Description*
iwconfig_eth0		See the iwconfig man page for details on what to send iwconfig
iwpriv_eth0		See the iwpriv man page for details on what to send iwpriv
sleep_scan_eth0	0	The number of seconds to sleep before attempting to scan. This is needed when the driver/firmware needs more time to active before it can be used.
sleep_associate_eth0	5	The number of seconds to wait for the interface to associate with the Access Point before moving onto the next one
associate_test_eth0	MAC	Some drivers do not reset the MAC address associated with an invalid one when they lose or attempt association. Some drivers do not reset the quality level when they lose or attempt association. Valid settings are MAC, quality and all.
scan_mode_eth0		Some drivers have to scan in ad-hoc mode, so if scanning fails try setting ad-hoc here
iwpriv_scan_pre_eth0		Sends some iwpriv commands to the interface before scanning. See the iwpriv man page for more details.
iwpriv_scan_post_eth0		Sends some iwpriv commands to the interface after scanning. See the iwpriv man page for more details.

4: Defining network configuration per ESSID

Sometimes, you need a static IP when you connect to *ESSID1* and you need DHCP when you connect to *ESSID2*. In fact, most module variables can be defined per ESSID. Here's how we do this.

Note: These work if you're using WPA Supplicant or Wireless Tools.

Important: You will need to consult the variable name documentation.

Code Listing 9: override network settings per ESSID

```
config_ESSID1=( "192.168.0.3/24 brd 192.168.0.255" )
routes_ESSID1=( "default via 192.168.0.1" )

config_ESSID2=( "dhcp" )
fallback_ESSID2=( "192.168.3.4/24" )
fallback_route_ESSID2=( "default via 192.168.3.1" )

# We can define nameservers and other things too
# NOTE: DHCP will override these unless it's told not too
dns_servers_ESSID1=( "192.168.0.1" "192.168.0.2" )
dns_domain_ESSID1="some.domain"
dns_search_domains_ESSID1="search.this.domain search.that.domain"

# You override by the MAC address of the Access Point
# This handy if you goto different locations that have the same ESSID
config_001122334455=( "dhcp" )
dhcpcd_001122334455="-t 10"
dns_servers_001122334455=( "192.168.0.1" "192.168.0.2" )
```

Adding Functionality

“If you're feeling adventurous, you can add your own functions to networking.”

1: Standard function hooks

Four functions can be defined which will be called surrounding the start/stop operations. The functions are called with the interface name first so that one function can control multiple adapters.

The return values for the preup() and predown() functions should be 0 (success) to indicate that configuration or deconfiguration of the interface can continue. If preup() returns a non-zero value, then interface configuration will be aborted. If predown() returns a non-zero value, then the interface will not be allowed to continue deconfiguration.

The return values for the postup() and postdown() functions are ignored since there's nothing to do if they indicate failure.

${IFACE} is set to the interface being brought up/down. ${IFVAR} is ${IFACE} converted to variable name bash allows.

Code Listing 1: pre/post up/down function examples

```
preup() {
  # Test for link on the interface prior to bringing it up.  This
  # only works on some network adapters and requires the mii-diag
  # package to be installed.
  if mii-tool ${IFACE} 2> /dev/null | grep -q 'no link'; then
    ewarn "No link on ${IFACE}, aborting configuration"
    return 1
  fi

  # Test for link on the interface prior to bringing it up.  This
  # only works on some network adapters and requires the ethtool
  # package to be installed.
  if ethtool ${IFACE} | grep -q 'Link detected: no'; then
    ewarn "No link on ${IFACE}, aborting configuration"
    return 1
  fi

  # Remember to return 0 on success
  return 0
}

predown() {
  # The default in the script is to test for NFS root and disallow
  # downing interfaces in that case.  Note that if you specify a
  # predown() function you will override that logic.  Here it is, in
  # case you still want it...
  if is_net_fs /; then
    eerror "root filesystem is network mounted -- can't stop ${IFACE}"
    return 1
  fi

  # Remember to return 0 on success
  return 0
}

postup() {
  # This function could be used, for example, to register with a
  # dynamic DNS service.  Another possibility would be to
  # send/receive mail once the interface is brought up.
       return 0
}

postdown() {
  # This function is mostly here for completeness... I haven't
  # thought of anything nifty to do with it yet ;-)
  return 0
}
```

2: Wireless Tools function hooks

Note: This will not work with WPA Supplicant - but the ${ESSID} and ${ESSIDVAR} variables are available in the postup() function.

Two functions can be defined which will be called surrounding the associate function. The functions are called with the interface name first so that one function can control multiple adapters.

The return values for the preassociate() function should be 0 (success) to indicate that configuration or deconfiguration of the interface can continue. If preassociate() returns a non-zero value, then interface configuration will be aborted.

The return value for the postassociate() function is ignored since there's nothing to do if it indicates failure.

${ESSID} is set to the exact ESSID of the AP you're connecting to. ${ESSIDVAR} is ${ESSID} converted to variable name bash allows.

Code Listing 2: pre/post association functions

```
preassociate() {
  # The below adds two configuration variables leap_user_ESSID
  # and leap_pass_ESSID. When they are both configured for the ESSID
  # being connected to then we run the CISCO LEAP script

  local user pass
  eval user=\"\$\{leap_user_${ESSIDVAR}\}\"
  eval pass=\"\$\{leap_pass_${ESSIDVAR}\}\"

  if [[ -n ${user} && -n ${pass} ]]; then
    if [[ ! -x /opt/cisco/bin/leapscript ]]; then
      eend "For LEAP support, please emerge net-misc/cisco-aironet-client-utils"
      return 1
    fi
    einfo "Waiting for LEAP Authentication on \"${ESSID//\\\\//}\""
    if /opt/cisco/bin/leapscript ${user} ${pass} | grep -q 'Login incorrect';
then
      ewarn "Login Failed for ${user}"
      return 1
    fi
  fi

  return 0
}

postassociate() {
  # This function is mostly here for completeness... I haven't
  # thought of anything nifty to do with it yet ;-)

  return 0
}
```

Note: ${ESSID} and ${ESSIDVAR} are unavailable in predown() and postdown() functions.

Network Management

"For laptop users or people who move their computer around different networks."

1: Network Management

If you and your computer are always on the move, you may not always have an ethernet cable or plugged in or an access point available. Also, we may want networking to automatically work when an ethernet cable is plugged in or an access point is found.

Here you can find some tools that helps you manage this.

Note: This document only talks about ifplugd, but there are alternatives you can look into like quickswitch.

2: ifplugd

ifplugd [http://0pointer.de/lennart/projects/ifplugd/] is a daemon that starts and stops interfaces when an ethernet cable is inserted or removed. It can also manage detecting association to Access Points or when new ones come in range.

Code Listing 1: Installing ifplugd

```
# emerge sys-apps/ifplugd
```

Configuration for ifplugd is fairly straightforward too. The configuration file is held in `/etc/conf.d/ifplugd`. Run man ifplugd for details on what the variables do.

Code Listing 2: sample ifplug configuration

```
# Define which interfaces we monitor
INTERFACES="eth0"

AUTO="no"
BEEP="yes"
IGNORE_FAIL="yes"
IGNORE_FAIL_POSITIVE="no"
IGNORE_RETVAL="yes"
POLL_TIME="1"
DELAY_UP="0"
DELAY_DOWN="0"
API_MODE="auto"
SHUTDOWN="no"
WAIT_ON_FORK="no"
MONITOR="no"
ARGS=""

# Additional parameters for ifplugd for the specified interface. Note that
# the global variable is ignored, when a variable like this is set for an
# interface
MONITOR_wlan0="yes"
DELAY_UP_wlan0="5"
DELAY_DOWN_wlan0="5"
```