1. Installing Gentoo

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

Content:

1. About the Gentoo Linux Installation
   Users not familiar with Gentoo do not always know that choice is what Gentoo is all about.
2. Choosing the Right Installation Medium
   You can install Gentoo in many ways. You can use our LiveCD, an existing distribution etc.
3. Configuring your Network
   If you want to install Gentoo using the Internet, you need to setup networking.
4. Preparing the Disks
   To be able to install Gentoo, you must create the necessary partitions. This chapter describes how to partition a disk for future usage.
5. Installing the Gentoo Installation Files
   Gentoo installs work through socalled stage-files. In this chapter we describe how you extract a stage-file and configure Portage.
6. Installing the Gentoo Base System
   Independent of what stage you chose, the eventual result is that you have a Gentoo base system at your disposal. This chapter describes how to progress to that stadium.
7. Configuring the Kernel
   The Linux kernel is the core of every distribution. This chapter explains how to configure your kernel.
8. Configuring your System
   You need to edit some important configuration files. In this chapter you receive an overview of these files and an explanation on how to proceed.
9. Configuring the Bootloader
   Several bootloaders exist. Each one of them has its own way of configuration. In this chapter we'll describe all possibilities for you and step you through the process of configuring a bootloader to your needs.
10. Installing Necessary System Tools
    As mentioned before, Gentoo is about choices. In this chapter we help you choose and install some important tools.
11. Finalizing your Gentoo Installation
    You're almost done. We'll just create one (or more) users for your system and leave the installation to reboot into a brand new Gentoo system.
12. Where to go from here?
    Now you have your Gentoo system, but what's next?

1. About the Gentoo Linux Installation

Content:

- Introduction
- Prebuilt or Compile-All?

1.a. Introduction

Welcome!

First of all, welcome to Gentoo. You are about to enter the world of choices and performance. Gentoo is all about choices. 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 metadistribution with a clean and flexible design. Gentoo is built around free software and doesn't hide to 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 textfiles. In other words, openness everywhere.

It is very important that you understand that choices are what makes Gentoo run. We try not to force you onto anything you don't like. If you feel like we do, please bugreport it.

How is the Installation Structured?

The Gentoo Installation can be seen as a 10-step procedure, corresponding to chapters 2 - 11. Every step results in a certain state:

- After step 1, you are in a working environment ready to install Gentoo
- After step 2, your internet connection is ready to install Gentoo (this can be optional in certain situations)
- After step 3, your hard disks are initialized to house your Gentoo installation
- After step 4, your installation environment is prepared and you are chrooted into the new environment
- After step 5, core packages, which are the same on all Gentoo installations, are installed
- After step 6, you have compiled your Linux kernel
- After step 7, you have written most of your Gentoo system configuration files
- After step 8, your choice of bootloader has been installed and configured
- After step 9, necessary system tools (which you can choose from a nice list) are installed
- After step 10, you are logged in into your new Gentoo installation

When you are given a certain choice, 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 depending 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 LiveCDs (installation CDs), from an existing distribution, from a bootable CD (such as Knoppix), from a netbooted environment, etc.

You also have several possibilities: you can compile your entire system from scratch or install prebuilt packages to have your Gentoo environment up and running in no time.

And of course you have intermediate solutions in which you don't compile everything but start from a semi-ready system.

Troubles?

If you find a problem in the installation (or in the installation documentation), please visit our bugtracking system and check if the bug is known. If not, please create a bugreport 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.

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, available from the Gentoo Documentation. You can also view the FAQs on our forums. 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 :-)

1.b. Prebuilt or Compile-All?

What is 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 out of all packages required to have a fully functional Gentoo installation. They are not only sufficient to have a base installation up to speed in no time, but all lengthier builds (such as KDE, XFree, 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

In order for Portage to be able to install the prebuilt packages they must reside in the `/usr/portage/packages/All` directory. If you are installing Gentoo from a LiveCD that houses these packages and you wish to install Gentoo using the GRP packages, we will tell you how to copy over those packages to this location later.

However, having the packages alone isn't sufficient: 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 will have to install a Portage snapshot instead of synchronising Portage with the latest available tree if you want to use the GRP installation method.

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 this moment we provide GRP packages for the following architectures:

- The x86 architecture (x86, i686, pentium3, pentium4, athlon-xp) and the special purpose Gentoo Hardened profile
- The amd64 architecture (amd64)
- The sparc architecture (sparc64)
- The ppc architecture (ppc, G3, G4)

The Gentoo Hardened project offers their own GRP set (and stages) focused on building a proactively secure system. Anyone looking to build a server on the x86 architecture should investigate this option.

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 Choosing the Right Installation Medium.

2. Choosing the Right Installation Medium

Content:

- Hardware Requirements
- Make your Choice
- Default: Download, Burn and Boot a Gentoo LiveCD
- Alternative: Download, Burn and Boot Knoppix
- Alternative: Net Booting

2.a. Hardware Requirements

Introduction

Before we start, we first list what hardware requirements you need to successfully install Gentoo on your box. This of course depends on your architecture.

Architectures

Gentoo is officially available for seven architectures, and has experimental support for one more. The official architectures are x86 (including all subarchitectures, like Pentium, Athlon, etc.), sparc (both Sparc32 and Sparc64), ppc (PowerPC), hppa, alpha, mips and amd64. Experimental support is available for ia64.

Assuming you know your architecture, check the following requirements before you continue with the Gentoo installation:

- You need at least 1 Gb of free disk space
- If you do not use prebuilt packages, you need at least 300 Mb of memory (RAM + swap)
- For the x86 architecture, you need a 486+ processor and at least 64 megabytes of memory
- For the Alpha architecture, you should check with the Alpha/Linux FAQ
- For the hppa architecture, you should check with the PA Team website
- For the PowerPC architecture, you need at least a PowerPC (at best a G3 or G4 like the iMac, iBook, PowerBook, etc.)
- For the SPARC architecture, you should check with the UltraLinux FAQ
- For the MIPS architecture, you should check with the MIPS Hardware Requirements document
- For the AMD64 architecture, you should check with AMD64 Tech Notes

2.b. Make your Choice

Introduction

Still interested in trying out Gentoo? Well, then it is now time to choose the installation medium you want to use. Yes, you have the choice, no, they are not all equal, and yes, the result is always the same: a Gentoo base system.

The installation media we will describe are:

- Gentoo LiveCDs
- Knoppix
- Other Distribution
- Net Booting

Every single media has its advantages and disadvantages. We will list the pros and cons of every medium so you have all the information to make a justified decision. But before we continue, let’s explain our three-stage installation.

The Three Stages

Gentoo Linux can be installed using one of three stage tarball files. The one you choose depends on how much of the system you want to compile yourself. The stage1 tarball is used when you want to bootstrap and build the entire system from scratch. The stage2 tarball is used for building the entire system from a bootstrapped "semi-compiled" state. The stage3 tarball already contains a basic Gentoo Linux system that has been built for you. As we will explain later, you can also install Gentoo without compiling anything (except your kernel and some optional packages). If you want this, you have to use a stage3 tarball.

Now what stage do you have to choose?

Starting from a stage1 allows you to have total control over the optimization settings and optional build-time functionality that is initially enabled on your system. This makes stage1 installs good for power users who know what they are doing. It is also a great installation method for those who would like to know more about the inner workings of Gentoo Linux.

Stage1	Pros and Cons
+	Allows you to have total control over the optimization settings and optional build-time functionality that is initially enabled on your system
+	Suitable for powerusers that know what they are doing
+	Allows you to learn more about the inner workings of Gentoo
-	Takes a long time to finish the installation
-	If you don’t intend to tweak the settings, it is probably a waste of time

Stage2 installs allow you to skip the bootstrap process and doing this is fine if you are happy with the optimization settings that we chose for your particular stage2 tarball.

Stage2	Pros and Cons
+	You don’t need to bootstrap
+	Faster than starting with stage1
+	You can still tweak your settings
-	You cannot tweak as much as with a stage1
-	It’s not the fastest way to install Gentoo
-	You have to accept the optimizations we chose for the bootstrap

Choosing to go with a stage3 allows for the fastest install of Gentoo Linux, but also means that your base system will have the optimization settings that we chose for you (which to be honest, are good settings and were carefully chosen to enhance performance while maintaining stability). stage3 is also required if you want to install Gentoo using prebuilt packages.

Stage3	Pros and Cons
+	Fastest way to get a Gentoo base system
-	You cannot tweak the base system - it’s built already
-	You cannot brag about having used stage1 or stage2

Write down (or remember) what stage you want to use. You need this later when you decide what LiveCD (or other installation medium) you want to use. You might be interested to know that, if you decide to use different optimization settings after having installed Gentoo, you will be able to recompile your entire system with the new optimization settings.

Now take a look at the available installation media:

- Gentoo LiveCDs
  - Gentoo basic LiveCDs
  - Gentoo 2-CD LiveCD Set
  - Gentoo KDE/Gnome LiveCD
- Knoppix
- Existing Distribution

- Net Booting

Gentoo LiveCDs

The Gentoo LiveCDs are bootable CDs which contain a self-sustained Gentoo environment. They allow you to boot Linux from the CD. During the boot process your hardware is detected and the appropriate drivers are loaded. They are maintained by Gentoo developers and are available for all supported architectures (x86, alpha, sparc, ppc, hppa, amd64). Even more, some architectures have several available LiveCDs.

All LiveCDs allow you to boot, setup networking, initialize your partitions and start installing Gentoo from the Internet. However, some LiveCDs also contain all necessary source code or even precompiled packages so you are able to install Gentoo without networking.

We provide several types of LiveCDs. The following table shows you what LiveCDs are available for your architecture:

Architecture	Basic LiveCD	2-CD LiveCD Set	KDE/Gnome LiveCD
x86	+	+	
sparc	+	+	
ppc	+	+	+
hppa	+		
alpha	+		
amd64	+		

Now what do these LiveCDs contain?

Gentoo basic LiveCDs

This is a small, no-nonsense, bootable CD which sole purpose is to boot the system, prepare the networking and continue with the Gentoo installation. It does not contain any stages (or, in some cases, a single stage1 file), source code or precompiled packages.

Basic LiveCD	Pros and Cons
+	Smallest download
+	Suitable for a complete architecture
+	You can do a stage1, stage2 or stage3 by getting the stage tarball off the net
-	Contains no stages, no portage snapshot, no GRP packages and therefore not suitable for networkless installation

Gentoo 2-CD LiveCD Set

The first CD is a bootable CD suitable to install Gentoo without networking. It contains a stage1 and several stage3 tarballs (optimized for the individual subarchitectures) and the necessary sourcecode to install Gentoo from stage1 without the need for a working network connection. Some architectures also provide stage2 tarballs and a set of prebuilt packages on the first CD.

The second CD only contains precompiled packages and can be used to install software after a succesfull Gentoo Installation. To install Gentoo, you only need CD-1, but if you want OpenOffice.org, Mozilla, KDE, GNOME etc. without having to compile every single one of them, you need CD-2 too.

We provide both a "default" LiveCD set, bootable on all subarchitectures for a specific architecture, as well as optimized LiveCDs (CD 2) for the various subarchitectures (such as Athlon-XP, G4, etc.).

2-CD LiveCD Set	Pros and Cons
+	Optimized to your architecture and subarchitecture
+	Provide precompiled packages for fast Gentoo installations
+	Contains everything you need. You can even install without a network connection.

-	Huge download

Gentoo KDE/Gnome LiveCD

This cd contains a complete live Gentoo environment featuring the KDE and Gnome desktop environment, an apache webserver, several webbrowsers, irc clients, firewall/router stuff, network auditing stuff, a lot of CLI tools... It can be used to fix a broken filesystem. The kde/gnome livecd contains an entire gcc (distcc, ccache enabled) toolchain; boot all systems on your network using this cd and experience the power of distributed compilation. This cd can be used to install Gentoo, but does not contain GRP, source code, nor a portage snapshot or stages.

KDE/Gnome LiveCD	Pros and Cons
+	You can work with Gentoo without installing it on your hard disk
+	You can also use it to install Gentoo :)
-	Huge download, but doesn't contain a portage snapshot, stages, precompiled packages or source code.

Knoppix

Knoppix is a well-known bootable CD with a fully working desktop environment. It allows you to boot from CD and, without interaction with your disk, start up office applications or system tools. In our case, we will use the system and networking tools to install Gentoo.

If you plan on using Knoppix, you can download it from the Knoppix homepage.

Existing Distribution

If you start installing from another already installed Linux distribution, then you benefit from all the tools already available in your installed Linux distribution. If you want to install Gentoo from an existing distribution, you should skip the rest of this chapter and continue with Configuring your Network.

Net Booting

In some cases you do not have the possibility to boot from a CD or use an existing installation to install Gentoo from. In case your system has a network interface and BIOS/ROM capable of performing a netboot (PXE) you can have it download a small system at boot time and install from that system onwards.

In case you want to use the net boot method (MIPS users have no choice) you should already have a working DHCP and TFTP server. Information on setting up a DHCP server and TFTP server are available in the Diskless-HOWTO.

Depending on your setup, you might need to create or download a kernel image for your system.

Choosing the Installation Medium

Now make your choice of installation medium and continue with the appropriate section. As this is a Gentoo handbook, we'll default to using the Gentoo LiveCDs, but you can find the necessary information about using the other installation media as alternative sections.

- Default: Download, Burn and Boot a Gentoo LiveCD
- Alternative: Download, Burn and Boot Knoppix
- Alternative: Net Booting

2.c. Default: Download, Burn and Boot a Gentoo LiveCD

Downloading and Burning the LiveCDs

You have chosen to use a Gentoo LiveCD (if not, then you are reading the wrong section). We'll first start by downloading and burning the chosen LiveCD. We previously discussed the several available LiveCDs, but where can you find them?

The following table lists the relative paths where you can find the LiveCD ISOs (CD images). The path is relative to the main Gentoo directory on any of our mirrors.

Architecture	Path
x86	`releases/2004.0/x86/livecd`
sparc	`releases/2004.0/sparc/livecd`
ppc	`releases/2004.0/ppc/livecd`
hppa	`experimental/hppa/livecd`
alpha	`experimental/alpha/livecd`
amd64	`releases/2004.0/amd64/livecd`

Visit one of our mirrors and go to the path where the LiveCD(s) of your choice are located. Inside that directory you'll find so-called ISO-files. Those are full CD images which you can write on a CD-R. If you find several ISOs with different date-tags (such as `gentoo-2004.0-x86-20040121.iso`) take the most recent one.

In case you wonder if your downloaded file is corrupted or not, you can check its MD5 checksum and compare it with the MD5 checksum we provide (such as `gentoo-2004.0-x86-20040121.iso.md5`). You can check the MD5 checksum with the `md5sum` tool under Linux/Unix or md5summer for Windows.

Note: Please check the descriptions on the Gentoo Store to know what LiveCD (or stage) you should use for your CPU. A common mistake is choosing a LiveCD (or stage) for a CPU that is more recent than yours (such as "athlon-xp" for a regular AMD Athlon).

To burn the downloaded ISO(s), you have to select raw-burning. How you do this is highly program-dependent. We will discuss a couple of popular tools on how to do this.

- With EasyCD Creator you select `File`, `Record CD from CD image`. Then you change the `Files of type` to `ISO image file`. Then locate the ISO file and click `Open`. When you click on `Start recording` the ISO image will be burned correctly onto the CD-R.
- With Nero Burning ROM, select `File`, `Burn CD image`. Set the type of file to `*.*` and select the ISO file. Older versions of Nero will tell you they don't recognize the format -- confirm here, it does recognize it but doesn't know it yet :) In the next dialog, set the following parameters:
  - Type of image: `Data Mode 1`
  - Block size: `2048 bytes`
  - File precursor and length of the image trailer: `0 bytes`
  - Scrambled: `no`
  - Swapped: `no`

  Now click on `OK` and then `Burn` (the CD-R)
- With cdrecord, you simply type `cdrecord dev=/dev/hdc` (replace `/dev/hdc` with your CD-RW drive's device path) followed by the path to the ISO file :)
- With Mac OS X Panther, launch `Disk Utility` from `Applications/Utilities`, select `Open` from the `Images` menu, select the mounted disk image in the main window and select `Burn` in the `Images` menu.
- With Mac OS X Jaguar, launch `Disk Copy` from `Applications/Utilities`, select `Burn Image` from the `File` menu, select the ISO and click the `Burn` button.

The following subsections explain how to boot the architecture-specific LiveCDs. Be sure to pick the subsection which is relevant for your architecture.

- Booting the x86 or AMD64 LiveCD(s)
- Booting the Alpha LiveCD(s)
- Booting the HPPA LiveCD(s)
- Booting the SPARC LiveCD(s)
- Booting the PPC LiveCD(s)

Booting the x86 or AMD64 LiveCD(s)

Important: This subsection is for x86 and amd64 architectures only. Read this whole subsection before continuing, as you will not always have the time to read before acting.

Once you have burned your installation CDs, it is time to boot them. 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 installation CD in the CD-ROM drive (duh) and reboot. You should see a fancy boot screen with the Gentoo Linux logo on it. 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 LiveCDs. The default one is `gentoo`. Other kernels are `smp`, which activates support for multi-cpu systems and the `-nofb` variants which disable framebuffer.

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

Kernel	Description
gentoo	Default kernel with framebuffer support
smp	Kernel with support for multiple CPUs
gentoo-nofb	Same as `gentoo` but without framebuffer support
smp-nofb	Same as `smp` but without framebuffer support

Note: Some LiveCDs provide extra kernels, or don't provide kernels listed in this document. To list the available kernels at boot-time, press F1 when you are at the bootscreen.

You can also provide kernel options. They represent optional settings you can (de)activate at will. The following table explains all available kernel options.

Kernel Option	Description
acpi	Activate ACPI support
doataraid	Activate support for ATA RAID devices
dofirewire	Activate support for FireWire devices
dokeymap	Ask the user for his keyboard setting (default: us)
dopcmcia	Activate PCMCIA support
doscsi	Activate support for SCSI devices
noapm	Deactivate APM support
nodetect	Deactivate hardware detection (kudzu/hotplug)
nodhcp	Do not use DHCP to query for an IP address
noevms	Deactivate EVMS support
nohotplug	Deactivate hotplug (kernel loading program)
nousb	Deactivate USB support
ide=nodma	Deactivate DMA support
cdcache	Cache the entire runtime portion of the CD in memory, which allows you to unmount your CD and use another one during installation.

Note: Some LiveCDs provide extra boot options, or don't provide boot options listed in this document. To list the available kernels at boot-time, press F2 when you are at the bootscreen.

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 cdcache` as kernel parameters:

Code listing 1: Booting a LiveCD

```
boot: gentoo dopcmcia cdcache
```

You will then be greeted with another boot screen and progress bar. Once the boot process completes, you will be automatically logged in to the "Live" Gentoo Linux as "root", the super user. You should have a root ("#") prompt on the current console and can also switch to other consoles by pressing Alt-F2, Alt-F3 and Alt-F4. Get back to the one you started on by pressing Alt-F1.

Now continue with Extra Hardware Configuration.

Booting the Alpha LiveCD(s)

Important: This subsection is for alpha architectures only. Read this whole subsection before continuing, as you will not always have the time to read before acting.

When your Alpha is powered on, the first thing that gets started is the firmware. It is loosely synonymous with the BIOS software on PC systems. There are two types of firmware on Alpha systems: SRM (Systems Reference Manual) and ARC (Advanced Risc Console).

SRM is based on the Alpha Console Subsystem specification, which provides an operating environment for OpenVMS, Tru64 UNIX, and Linux operating systems. ARM is based on the Advanced RISC Computing (ARC) specification, which provides an operating environment for Windows NT.

If your Alpha system supports both SRC and ARCs (ARC, AlphaBIOS, ARCSBIOS) you should follow these instructions for switching to SRM. If your system already uses SRM, you are all set. If your system can only use ARCs (Ruffian, nautilus, xl, etc.) you will need to choose `MILO` later on when we are talking about bootloaders.

Now to boot an Alpha LiveCD, put the CD-ROM in the tray and reboot the system. You can use SRM to boot the LiveCD. If you cannot do that, you will have to use `MILO`. If you don’t have `MILO` installed already, use one of the precompiled `MILO` images available on taviso’s homepage.

Code listing 2: Booting a CD-ROM using SRM

```
(Substitute dqa0 with your CD-ROM drive device)
>>> boot dqa0 -flags 0
```

Code listing 3: Booting a CD-ROM using MILO

```
(Substitute hdb with your CD-ROM drive device)
MILO> boot hdb:boot/vmlinuz initrd=initrd.img root=/dev/ram0 init=/linuxrc
```

You should have a root ("#") prompt on the current console and can also switch to other consoles by pressing Alt-F2, Alt-F3 and Alt-F4. Get back to the one you started on by pressing Alt-F1.

Now continue with Extra Hardware Configuration.

Booting the HPPA LiveCD(s)

Important: This subsection is for HPPA architectures only. Read this whole subsection before continuing, as you will not always have the time to read before acting.

Boot your HPPA system. During the boot process, you will see a message similar to the following:

Code listing 4: HPPA boot message

```
Searching for Potential Boot Devices.
To terminate search, press and hold the ESCAPE key.
```

When this message appears, press and hold the Esc-key until an option menu appears. This can take a while, be patient. By default, you should enter the BOOT_ADMIN console. If you receive an option menu, choose `Enter Boot Administration mode` to enter the BOOT_ADMIN console. You should now have an '>' prompt.

Put the Gentoo LiveCD in the CD-ROM. If you do not know the SCSI ID of your CD-ROM drive, your PA-RISC station will search for it when you issue the `search` command.

Code listing 5: Searching for SCSI ID

```
> search
Searching for Devices with Bootable Media.
To terminate search, please press and hold the ESCAPE key.
```

Your PA-RISC station will now display all the available boot media. This is an example result of this command :

Code listing 6: Available boot media

```
Device Selection      Device Path             Device Type and Utilities
---------------------------------------------------------------------------

P0                    scsi.5.0                TOSHIBA CD-ROM XM-3301TA
                                                  IPL
P1                    scsi.2.0                COMPAQ ST32550N
                                                  IPL
P2                    lan.0010a7-06d1b6.3.6   server
                                                  IPL
```

To boot from a CD-ROM you need the accompanying Device Path. For instance, if we want to boot from the TOSHIBA CD-ROM in the above example, we would need to type the following command:

Code listing 7: Booting from a CD-ROM

```
> boot scsi.5.0 ipl

Trying scsi.5.0
```

The `ipl` keyword (Initial Program Loader) tells palo (the PA-RISC boot LOader) to enter interactive mode. This will allow you to change, for example, the kernel boot parameters.

When the boot is successful, palo will start in interactive mode:

Code listing 8: PALO Interactive Mode

```
Boot path initialized.
Attempting to load IPL.


Hard booted.
palo ipl 1.2 root@b180l.da-kot Tue Apr  8 12:43:07 CEST 2003

Boot image contains:
    0/vmlinux32 4028015 bytes @ 0x1520000
    0/ramdisk 834748 bytes @ 0xf800
Current command line:
0/vmlinux initrd=initrd.gz TERM=linux console=tty root=/dev/ram0 init=/linuxrc
  0: 0/vmlinux
  1: initrd=initrd.gz
  2: TERM=linux
  3: console=tty
  4: root=/dev/ram0
  5: init=/linuxrc

Edit which field?
(or 'b' to boot with this command line)?
```

These parameters are suitable for most situations.

If you need extra features you must add the apropriate keyword(s) to the end of the command line. To add a keyword, edit the last field, add a space and type your keyword. The only implemented keyword as of now is `cdcache` which tells the LiveCD to load itself into RAM, allowing you to unmount the CD.

Code listing 9: Adding cdcache as boot option

```
(or 'b' to boot with this command line)? 5
init=/linuxrc cdcache
```

Now that you have tweaked your kernel boot params, boot it.

Code listing 10: Booting the kernel

```
(or 'b' to boot with this command line)? b
```

You should have a root ("#") prompt on the current console and can also switch to other consoles by pressing Alt-F2, Alt-F3 and Alt-F4. Get back to the one you started on by pressing Alt-F1.

Now continue with Extra Hardware Configuration.

Booting SPARC LiveCD(s)

Important: This subsection is for SPARC architectures only. Read this whole subsection before continuing, as you will not always have the time to read before acting.

Insert the Gentoo LiveCD in the CD-ROM and boot your system. During startup, press Stop-A to enter OpenBootPROM (OBP). Once you are in the OBP, boot from the CD-ROM:

Code listing 11: Booting the LiveCD

```
ok boot cdrom
```

You will be greeted by the SILO boot manager (on the LiveCD). Type in `gentoo` (single-CPU kernel) or `smp` (multi-CPU kernel) and press enter to continue booting the system. In the following example we'll boot the `gentoo` kernel.

Code listing 12: Continue booting from the LiveCD

```
boot: gentoo
```

Once the LiveCD is booted, you will be greeted by a login prompt. Log on as `root`. There is no password, so when you are asked for one, press Enter.

Code listing 13: Logging on onto the LiveCD

```
login: root
password: (Press Enter here)
```

You should have a root ("#") prompt on the current console and can also switch to other consoles by pressing Alt-F2, Alt-F3 and Alt-F4. Get back to the one you started on by pressing Alt-F1.

Continue with Extra Hardware Configuration.

Booting the PPC LiveCD(s)

Place the LiveCD in the CD-ROM and reboot the system. Hold down the 'C' key at bootup (or run an OldWorld bootloader like BootX or quik). You will be greeted by a friendly welcome message and a boot: prompt at the bottom of the screen.

At this prompt, hit enter, and a complete Gentoo Linux environment will be loaded from the CD. If you experience problems booting, choose the `-safe` option at boot. The safe option passes the following extra arguments to the kernel: `append="video=ofonly nol3 init=/linuxrc"`.

When the LiveCD is booted, you will be greeted with a login prompt. Log on as `root` and leave the password blank (i.e. press Enter).

Code listing 14: Logging on onto the LiveCD

```
login: root
password: (Press Enter here)
```

You should have a root ("#") prompt on the current console and can also switch to other consoles by pressing Alt-fn-F2, Alt-fn-F3 and Alt-fn-F4. Get back to the one you started on by pressing Alt-fn-F1.

If you are installing Gentoo on a system with a non-US keyboard, use `loadkeys` to load the keymap for your keyboard. To list the available keymaps, execute `ls /usr/share/keymaps`.

Code listing 15: Listing available keymaps

```
# ls /usr/share/keymaps
```

To load a keymap, use `loadkeys`.

Code listing 16: Loading a keymap

```
# loadkeys be2-latin1
```

Now continue with Extra Hardware Configuration.

Extra Hardware Configuration

When the Live CD 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.

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

Code listing 17: Loading kernel modules

```
# modprobe 8139too
```

Optional: Tweaking Hard Disk Performance

If you are an advanced user, you might want to tweak the IDE hard disk performance using `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 18: 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 19: Tweaking hard disk performance

```
Activate DMA:                                    # hdparm -d 1 /dev/hda
Activate DMA + Safe Performance-enhancing 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.

To change the root password, use the `passwd` utility:

Code listing 20: Changing the root password

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

To create a user account, we first enter his 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 21: Creating a user account

```
# useradd 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 22: Changing user id

```
# su john -
```

Optional: Starting the SSH Daemon

If you want to allow other users to access your computer during the Gentoo installation (perhaps because those users are going to help you install Gentoo, or even do it for you), you need to create a user account for them and perhaps even provide them with your root password (only do that if you fully trust that user).

To fire up the SSH daemon, execute the following command:

Code listing 23: Starting the SSH daemon

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

Now continue with the next chapter on Configuring your Network.

2.d. Alternative: Download, Burn and Boot Knoppix

Important: You can only use Knoppix on x86-based systems.

You can download Knoppix from the Knoppix homepage. Knoppix too is available as ISO-files. After having downloaded the Knoppix ISO file, burn it to a CD-R. You have to use "raw" burning. We'll explain how to do this with a couple of popular tools.

- With EasyCD Creator you select `File`, `Record CD from CD image`. Then you change the `Files of type` to `ISO image file`. Then locate the ISO file and click `Open`. When you click on `Start recording` the ISO image will be burned correctly onto the CD-R.
- With Nero Burning ROM, select `File`, `Burn CD image`. Set the type of file to `*.*` and select the ISO file. Older versions of Nero will tell you they don't recognize the format -- confirm here, it does recognize it but doesn't know it yet :) In the next dialog, set the following parameters:
    - Type of image: `Data Mode 1`
    - Block size: `2048 bytes`
    - File precursor and length of the image trailer: `0 bytes`
    - Scrambled: `no`

    - Swapped: `no`

  Now click on `OK` and then `Burn` (the CD-R)
- With cdrecord, you simply type `cdrecord dev=/dev/hdc` (replace `/dev/hdc` with your CD-RW drive's device path) followed by the path to the ISO file :)

By default, Knoppix boots into a KDE 3.0 desktop. The first thing you'll have to do is set the root password for knoppix. Open a konsole and type in the following command:

Code listing 24: Setting root password for Knoppix

```
$ sudo passwd root
Password: (Enter a password)
Re-enter password: (Re-enter the password)
```

Next, `su` to root and set the root home directory to `/root`. If you do not do this, you will receive errors when emerging about "/home/root: not found".

Code listing 25: Changing root home directory

```
$ su -
Password: (Enter the root password created above)
# usermod -d /root -m root
```

Now exit the current shell by typing `exit` and `su` back into root. This will load the made changes.

Code listing 26: Loading the changes

```
# exit
$ su -
Password: (Enter the root password)
```

Create the `/mnt/gentoo` mountpoint:

Code listing 27: Creating the /mnt/gentoo mountpoint

```
# mkdir -p /mnt/gentoo
```

Now continue with Configuring your Network.

2.e. Alternative: Net Booting

To be able to install Gentoo, the image you download from the TFTP server should provide the necessary tools to create filesystems, create and mount partitions, extract a tarball and chroot. You can download existing net boot images which have all tools in them...

At this time we know of only one architecture that has net boot images available and that is the MIPS architecture.

- Net Booting a MIPS System

Net Booting a MIPS System

First download one of the available net boot images from http://dev.gentoo.org/~kumba/mips/netboot/.

Now configure your DHCP server to send this file to the booting client. SGI machines however need some minor tweaks to the host system in order for TFTP to work properly:

Code listing 28: Some fixes to SGI machines to have TFTP work properly

```
(Disable "Path Maximum Transfer Unit", otherwise SGI Prom won't find the kernel)
# echo 1 > /proc/sys/net/ipv4/ip_no_pmtu_disc
(Set the port range usable by the SGI Prom)
# echo "2048 32767" > /proc/sys/net/ipv4/ip_local_port_range
```

Now power on your machine, get into the PROM monitor and issue the command to boot the kernel over the network:

Code listing 29: Net booting a MIPS

```
        Running power-on diagnostics

System Maintenance Menu

1) Start System
2) Install System Software
3) Run Diagnostics
4) Recover System
5) Enter Command Monitor

Option? 5
Command Monitor. Type "exit" to return to the menu.
>> bootp(): root=/dev/ram0
```

Note: You may have to press the Esc key to get into the menu above, otherwise, the system will attempt to boot the system directly.

Sometimes netbooting is a tricky endeavour. The following PROM commands below may help, but this is not guaranteed. If your machine refuses to netboot, double check things on the host TFTP machine to make sure that:

- dhcpd is giving the SGI Machine an IP Address
- Permissions are set properly in your tftp folder (typically `/tftpboot`)
- Check system logs to see what the tftp server is reporting (errors perhaps)
- Pray to a Tux plushie (this may or may not work, and is not an officially supported troubleshooting technique)

Code listing 30: Net booting tricks

```
>> resetenv
>> unsetenv netaddr
>> unsetenv dlserver
>> init
>> bootp(): root=/dev/ram0
```

If all things go well, you are now dropped off at busybox' `ash` shell. You can then continue with Configuring your Network.

3. Configuring your Network

Content:

- You can do without, but...
- Automatic Network Detection
- Automatic Network Configuration
- Manual Network Configuration

3.a. You can do without, but...

Who can do without?

Depending on the medium you chose to install Gentoo from, you can or cannot continue without networking (and Internet). No, we are not playing with your mind =)

Generally you will need to setup networking (and Internet). However, Gentoo also provides the possibility to install without a network connection. This exception is only possible with the following installation media:

- Gentoo x86 2-CD Installation LiveCD sets (including x86, i686, pentium3, pentium4 and athlon-xp)
- Gentoo PPC 2-CD Installation LiveCD sets (including G3 and G4)
- Gentoo Sparc 2-CD Installation LiveCD sets

Why do I need networking?

Installing Gentoo from the Internet results in a fully updated Gentoo Installation. You'll have an installation based on the most recent Portage tree (which is the collection of packages we provide together with the tools to manage your software). This is also the reason why a network-installation is preferred. However, some people cannot or do not want to install Gentoo on a system with a running Internet connection.

If you are in this situation you will need to use the 2CD-sets. These sets include the source code, a snapshot of the portage-tree and the tools to install a Gentoo base-system and beyond. This method comes at a price: You won't have the very latest software, although differences will be minimal.

If you want to follow this networkless installation you have to use such a 2-CD set, skip the rest of this chapter and continue with Preparing the Disks. Otherwise, continue with the networking configuration sections below.

Optional: Configure Proxy

If you access the Internet through a proxy, you might need to setup proxy information during the installation. It is very easy to define a proxy: you just need to define a variable which contains the proxy server information.

In most cases, you can just define the variables using the server hostname. As an example, we assume the proxy is called proxy.gentoo.org and the port is 8080.

Code listing 1: Defining proxy servers

```
(If the proxy filters HTTP traffic)
# export http_proxy="http://proxy.gentoo.org:8080"
(If the proxy filters FTP traffic)
# export ftp_proxy="ftp://proxy.gentoo.org:8080"
(If the proxy filters RSYNC traffic)
# export RSYNC_PROXY="proxy.gentoo.org:8080"
```

If your proxy requires a username and password, you should use the following syntax for the variable:

Code listing 2: Adding username/password to the proxy variable

```
http://username:password@server
```

For instance, for HTTP proxying with our previous proxy server and a username of "john" with a password of "f00b_r" one would use:

Code listing 3: Authenticated Proxy

```
# export http_proxy="http://john:f00b_r@proxy.gentoo.org:8080"
```

Networking from non-Gentoo Installation Mediums

Most information in this section is meant for users who booted from a Gentoo LiveCD. However, you can install Gentoo from several other media. If you are in such a situation, make sure that the used medium has a working Internet connection (the information available in Using DHCP or Understanding Network Terminology might come in handy) and continue with Preparing the Disks.

3.b. Automatic Network Detection

Maybe it just works?

If your system is plugged into an Ethernet network with a DHCP server, it is very likely that your networking configuration has already been set up automatically for you. If so, you should be able to take advantage of the many included network-aware commands on the LiveCD such as `ssh`, `scp`, `ping`, `irssi`, `wget` and `links`, among others.

If networking has been configured for you, the `/sbin/ifconfig` command should list some network interfaces besides lo, such as eth0:

Code listing 4: /sbin/ifconfig for a working network card

```
# /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:BA:8F:61:7A
          inet addr:192.168.0.2  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::50:ba8f:617a/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1498792 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1284980 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1984 txqueuelen:100
          RX bytes:485691215 (463.1 Mb)  TX bytes:123951388 (118.2 Mb)
          Interrupt:11 Base address:0xe800
```

Testing the Network

You may want to try pinging your ISP's DNS server (found in `/etc/resolv.conf`) and a Web site of choice, just to make sure that your packets are reaching the net, DNS name resolution is working correctly, etc..

Code listing 5: Further network testing

```
# ping -c 3 www.yahoo.com
```

Are you able to use your network? If so, you can skip the rest of this section and continue with Preparing the Disks. If not, bad luck, you'll have to pursue a bit harder :)

3.c. Automatic Network Configuration

If the network doesn't work immediately, some installation media allow you to use `net-setup` (for regular networks), `adsl-setup` (for ADSL-users) or `pptp` (for PPTP-users).

If you installation medium does not contain any of these tools, continue with Manual Network Configuration.

- Regular Ethernet users should continue with Default: Using net-setup
- ADSL users should continue with Alternative: Using RP-PPPoE
- PPTP users should continue with Alternative: Using PPTP

Default: Using net-setup

The simplest way to set up networking if it didn't get configured automatically is to run the `net-setup` script:

Code listing 6: Running the net-setup script

```
# net-setup eth0
```

`net-setup` will ask you some questions about your network environment. When all is done, you should have a working network connection. Test your network connection as stated before. If the tests are positive, congratulations! You are now ready to install Gentoo. Skip the rest of this section and continue with Preparing the Disks.

If your network still doesn't work, continue with Manual Network Configuration.

Alternative: Using RP-PPPoE

Assuming you need PPPoE to connect to the internet, the LiveCD (any version) has made things easy for you by including `rp-pppoe`. Use the provided `adsl-setup` script to configure your connection. You will be prompted for the ethernet device that is connected to your adsl modem, your username and password, the IPs of your DNS servers and if you need a basic firewall or not.

Code listing 7: Using rp-pppoe

```
# adsl-setup
# adsl-start
```

If something goes wrong, double-check that you correctly typed your username and password by looking at `/etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets` and make sure you are using the right ethernet device. If your ethernet device doesn't exist, you will have to load the appropriate network modules. In that case you should continue with Manual Network Configuration as we explain how to load the appropriate network modules there.

If everything worked, continue with Preparing the Disks.

Alternative: Using PPTP

If you need PPTP support, you can use `pptpclient` which is provided by our LiveCDs. But first you need to make sure that your configuration is correct. Edit `/etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets` so it contains the correct username/password combination:

Code listing 8: Editing /etc/ppp/chap-secrets

```
# nano -w /etc/ppp/chap-secrets
```

Then adjust `/etc/ppp/options.pptp` if necessary:

Code listing 9: Editing /etc/ppp/options.pptp

```
# nano -w /etc/ppp/options.pptp
```

When all that is done, just run `pptp` (along with the options you couldn't set in `options.pptp`) to connect the server:

Code listing 10: Connection to a dial-in server

```
# pptp <server ip>
```

Now continue with Preparing the Disks.

3.d. Manual Network Configuration

Loading the Appropriate Network Modules

When the Live CD boots, it tries to detect all your hardware devices and loads the appropriate kernel modules (drivers) 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 `net-setup` or `adsl-setup` failed, then you can safely assume that your networkcard wasn't found immediately. This means you will have to load the appropriate kernel modules manually.

To find out what kernel modules we provide for networking, use `ls`:

Code listing 11: Searching for provided modules

```
# ls /lib/modules/‘uname -r‘/kernel/drivers/net
```

If you find a driver for your network card, use `modprobe` to load the kernel module:

Code listing 12: Using modprobe to load a kernel module

```
(As an example, we load the pcnet32 module)
# modprobe pcnet32
```

To check if your network card is now detected, use `ifconfig`. A detected network card would result in something like this:

Code listing 13: Testing availability of your network card, successful

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr FE:FD:00:00:00:00
          BROADCAST NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

If however you receive the following error, the network card is not detected:

Code listing 14: Testing availability of your network card, failed

```
# ifconfig eth0
eth0: error fetching interface information: Device not found
```

Assuming that you now have a detected network card, you can retry `net-setup` or `adsl-setup` again (which should work now), but for the hardcore people amongst you, we explain how to configure your network manually.

There are two possibilities here. Either you use DHCP (automatic IP retrieval), or you manually setup your network using the `ifconfig` and `route` commands.

Using DHCP

DHCP (Dynamic Host Configuration Protocol) makes it possible to automatically receive networking information (IP address, netmask, broadcast address, gateway, nameservers etc.). This only works if you have a DHCP server in your network (or if your provider provides a DHCP service). To have a network interface receive this information automatically, use `dhcpcd`:

Code listing 15: Using dhcpcd

```
# dhcpcd eth0
```

If this works (try pinging some internet server, like Google), then you are all set and ready to continue. Skip the rest of this section and continue with Preparing the Disks.

Understanding Network Terminology

Note: If you know your IP address, broadcast address, netmask and nameservers, then you can skip this subsection and continue with Using ifconfig and route.

If all above fails, you will have to configure your network manually. Have no fear, it is far from difficult. But we are going to explain a certain amount of networking to you as you will need it to be able to configure your network to your satisfaction. When you're done reading this, you will know what a gateway is, what a netmask serves for, how a broadcast address is formed and why you need nameservers.

In a network, hosts are identified by their IP address (Internet Protocol address). Such an address is a combination of four numbers between 0 and 255. Well, at least that is how we perceive it. In reality, such an IP address consists of 32 bits (ones and zeros). Let's view an example:

Code listing 16: Example of an IP address

```
IP Address (numbers):   192.168.0.2
IP Address (bits):      11000000 10101000 00000000 00000010
                        -------- -------- -------- --------
                           192      168       0        2
```

Such an IP address is unique to a host as far as all accessible networks are concerned (i.e. all hosts that you are able to reach must have unique IP addresses). To be able to make a distinction between hosts inside a network, and hosts outside a network, the IP address is divided in two parts: the network part and the host part.

The separation is written down with the netmask, a collection of ones followed by a collection of zeros. The part of the IP that can be mapped on the ones is the network-part, the other one is the host-part. As usual, the netmask can be written down as an IP-address.

Code listing 17: Example of network/host separation

```
IP-address:    192      168      0        2
            11000000 10101000 00000000 00000010
Netmask:    11111111 11111111 11111111 00000000
               255      255      255      0
           +--------------------------+--------+
                    Network               Host
```

In other words, 192.168.0.14 is still part of our example network, but 192.168.1.2 is not.

The broadcast address is an IP-address with the same network-part as your network, but with only ones as host-part. Every host on your network listens to this IP address. It is truely meant for broadcasting packets.

Code listing 18: Broadcast address

```
IP-address:    192      168      0        2
            11000000 10101000 00000000 00000010
Broadcast:  11000000 10101000 00000000 11111111
               192      168      0       255
           +--------------------------+--------+
                     Network              Host
```

To be able to surf on the internet, you must know which host shares the Internet connection. This host is called the gateway. Since it is a regular host, it has a regular IP address (for instance 192.168.0.1).

We previously stated that every host has its own IP address. To be able to reach this host by a name (instead of an IP address) you need a service that translates a name (such as dev.gentoo.org) to an IP address (such as 64.5.62.82). Such a service is called a name service. To use such a service, you must define the necessary name servers in `/etc/resolv.conf`.

In some cases, your gateway also serves as nameserver. Otherwise you will have to enter the nameservers provided by your ISP.

To summarise, you will need the following information before continuing:

Network Item	Example
Your IP address	192.168.0.2
Netmask	255.255.255.0
Broadcast	192.168.0.255
Gateway	192.168.0.1
Nameserver(s)	195.130.130.5, 195.130.130.133

Using ifconfig and route

Setting up your network consists of three steps. First we assign ourselves an IP address using `ifconfig`. Then we setup routing to the gateway using `route`. Then we finish up by placing the nameserver IPs in `/etc/resolv.conf`.

To assign an IP address, you will need your IP address, broadcast address and netmask. Then execute the following command, substituting `${IP_ADDR}` with your IP address, `${BROADCAST}` with your broadcast address and `${NETMASK}` with your netmask:

Code listing 19: Using ifconfig

```
# ifconfig eth0 ${IP_ADDR} broadcast ${BROADCAST} netmask ${NETMASK} up
```

Now set up routing using `route`. Substitute `${GATEWAY}` with your gateway IP address:

Code listing 20: Using route

```
# route add default gw ${GATEWAY}
```

Now open `/etc/resolv.conf` with your favorite editor (in our example, we use `nano`):

Code listing 21: Creating /etc/resolv.conf

```
# nano -w /etc/resolv.conf
```

Now fill in your nameserver(s) using the following as a template. Make sure you substitute `${NAMESERVER1}` and `${NAMESERVER2}` with the appropriate nameserver addresses:

Code listing 22: /etc/resolv.conf template

```
nameserver ${NAMESERVER1}
nameserver ${NAMESERVER2}
```

That's it. Now test your network by pinging some Internet server (like Google). If this works, congratulations then. You are now ready to install Gentoo. Continue with Preparing the Disks.

4. Preparing the Disks

Content:

- Introduction to Block Devices
- Designing a Partitioning Scheme
- Using fdisk on x86 or amd64 to Partition your Disk
- Using fdisk on Alpha to Partition your Disk
- Using fdisk on SPARC to Partition your Disk
- Using mac-fdisk on PPC to Partition your Disk
- Using fdisk on HPPA to Partition your Disk
- Using fdisk on MIPS to Partition your Disk
- Creating Filesystems
- Mounting

4.a. Introduction to Block Devices

Block Devices

We'll take a good look at disk-oriented aspects of Gentoo Linux and Linux in general, including Linux filesystems, partitions and block devices. Then, once you're familiar with the ins and outs of disks and filesystems, you'll be guided through the process of setting up partitions and filesystems for your Gentoo Linux installation.

To begin, we'll introduce block devices. The most famous block device is probably the one that represents the first IDE drive in a Linux system, namely `/dev/hda`. If your system uses SCSI drives, then your first hard drive would be `/dev/sda`.

The block devices above represent an abstract interface to the disk. User programs can use these block devices to interact with your disk without worrying about whether your drives are IDE, SCSI or something else. The program can simply address the storage on the disk as a bunch of contiguous, randomly-accessible 512-byte blocks.

Partitions and Slices

Although it is theoretically possible to use a full disk to house your Linux system, this is almost never done in practice. Instead, full disk block devices are split up in smaller, more manageable block devices. On most systems, these are called partitions. Other architectures use a similar technique, called slices.

Partitions

Partitions are divided in three types: primary, extended and logical.

A primary partition is a partition which has its information stored in the MBR (master boot record). As an MBR is very small (512 bytes) only four primary partitions can be defined (for instance, `/dev/hda1` to `/dev/hda4`).

An extended partition is a special primary partition (meaning the extended partition must be one of the four possible primary partitions) which contains more partitions. Such a partition didn't exist originally, but as four partitions were too few, it was brought to life to extend the formatting scheme without losing backward compatibility.

A logical partition is a partition inside the extended partition. Their definitions aren't placed inside the MBR, but are declared inside the extended partition.

Advanced Storage

If you are booted from a Gentoo LiveCD then you have the possibility to use EVMS or LVM2 to increase the flexibility offered by your partitioning setup. During the installation instructions, we will focus on "regular" partitions, but it is still good to know EVMS and LVM2 are supported as well.

4.b. Designing a Partitioning Scheme

Default Partitioning Scheme

If you are not interested in drawing up a partitioning scheme for your system, you can use the (non-LVM) partitioning scheme we use throughout this book:

For x86 or amd64:

Partition	Filesystem	Size	Description
`/dev/hda1`	ext2	32M	Boot partition
`/dev/hda2`	(swap)	512M	Swap partition
`/dev/hda3`	ext3	Rest of the disk	Root partition

For ppc:

Partition NewWorld	Partition OldWorld	Filesystem	Size	Description
`/dev/hda1`	(Not needed)	(bootstrap)	800k	Apple_Bootstrap
`/dev/hda2`	`/dev/hda1`	(swap)	512M	Swap partition
`/dev/hda3`	`/dev/hda2`	ext3	Rest of the disk	Root partition

For Sparc:

Sun Disklabel	Filesystem	Size	Description
`/dev/hda1`	ext2	32M	Boot partition
`/dev/hda2`	(swap)	512M	Swap partition
`/dev/hda3`	(none)	Full disk	Sun Disk Label (required)
`/dev/hda4`	ext3	Rest of the disk	Root partition

If you are installing Gentoo from an existing distribution, you should first resize your existing partitions (if you don't have any spare room left) to be able to install Gentoo. You can use GNU/Parted to resize your partitions.

If you are interested in knowing how big a partition (or logical volume) should be, or even how many partitions (or volumes) you need, read on. Otherwise continue now with partitioning your disk:

- Using fdisk on x86 or amd64 to Partition your Disk
- Using fdisk on Alpha to Partition your Disk
- Using fdisk on SPARC to Partition your Disk
- Using mac-fdisk on PPC to Partition your Disk
- Using fdisk on HPPA to Partition your Disk
- Using fdisk on MIPS to Partition your Disk

How Many and How Big?

The number of partitions is highly dependent on your environment. For instance, if you have lots of users, you will most likely want to have your `/home` separate as it increases security and makes backups easier. If you are installing Gentoo to perform as a mailserver, your `/var` should be separate as all mails are stored inside `/var`. A good choice of filesystem will then maximise your performance. Gameservers will have a separate `/opt` as most gaming servers are installed there. The reason is similar for `/home`: security and backups.

As you can see, it very much depends on what you want to achieve. Separate partitions or volumes have the following advantages:

- You can choose the most performant filesystem for each partition or volume
- Your entire system cannot run out of free space if one defunct tool is continuously writing files to a partition or volume
- If necessary, file system checks are reduced in time, as multiple checks can be done in parallel (although this advantage is more with multiple disks than it is with multiple partitions)
- Security can be enhanced by mounting some partitions or volumes read-only, nosuid (setuid bits are ignored), noexec (executable bits are ignored) etc.

However, multiple partitions have one big disadvantage: if not configured properly, you might result in having a system with lots of free space on one partition and none on another.

As an example partitioning, we show you one for a 20Gb disk, used as a demonstration laptop (containing webserver, mailserver, gnome, ...):

Code listing 1: Filesystem usage example

```
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/hda5     ext3    509M  132M  351M  28% /
/dev/hda2     ext3    5.0G  3.0G  1.8G  63% /home
/dev/hda7     ext3    7.9G  6.2G  1.3G  83% /usr
/dev/hda8     ext3   1011M  483M  477M  51% /opt
/dev/hda9     ext3    2.0G  607M  1.3G  32% /var
/dev/hda1     ext2     51M   17M   31M  36% /boot
/dev/hda6     swap    516M   12M  504M   2% <not mounted>
(Unpartitioned space for future usage: 2 Gb)
```

`/usr` is rather full (83% used) here, but once all software is installed, `/usr` doesn't tend to grow that much. For `/var`, people might think the assigned space is too much. However, Gentoo compiles all programs inside `/var/tmp/portage`, so you should have `/var` with at least 1G free if you don't want to compile big programs, up to 3G free if compiling KDE and OpenOffice.org at the same time is no big deal for you.

Now partition your disk(s) using the instructions available for your architecture as an example:

- Using fdisk on x86 or amd64 to Partition your Disk
- Using fdisk on Alpha to Partition your Disk
- Using fdisk on SPARC to Partition your Disk
- Using mac-fdisk on PPC to Partition your Disk
- Using fdisk on HPPA to Partition your Disk
- Using fdisk on MIPS to Partition your Disk

4.c. Using fdisk on x86 or amd64 to Partition your Disk

Important: Only users with x86 or amd64 based systems should read this section.

The following parts explain how to create the example partition layout described previously, namely:

Partition	Description
/dev/hda1	Boot partition
/dev/hda2	Swap partition
/dev/hda3	Root partition

Change your partition layout according to your own will.

Viewing the Current Partition Layout

`fdisk` is a popular and powerful tool to split your disk into partitions. Fire up `fdisk` on your disk (in our example, we use `/dev/hda`):

Code listing 2: Starting fdisk

```
# fdisk /dev/hda
```

Once in `fdisk`, you'll be greeted with a prompt that looks like this:

Code listing 3: fdisk prompt

```
Command (m for help):
```

Type p to display your disk's current partition configuration:

Code listing 4: An example partition configuration

```
Command (m for help): p

Disk /dev/hda: 240 heads, 63 sectors, 2184 cylinders
Units = cylinders of 15120 * 512 bytes

Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
/dev/hda2            15        49    264600   82  Linux swap
/dev/hda3            50        70    158760   83  Linux
/dev/hda4            71      2184  15981840    5  Extended
/dev/hda5            71       209   1050808+  83  Linux
/dev/hda6           210       348   1050808+  83  Linux
/dev/hda7           349       626   2101648+  83  Linux
/dev/hda8           627       904   2101648+  83  Linux
/dev/hda9           905      2184   9676768+  83  Linux

Command (m for help):
```

This particular disk is configured to house seven Linux filesystems (each with a corresponding partition listed as "Linux") as well as a swap partition (listed as "Linux swap").

Removing all Partitions

We will first remove all existing partitions from the disk. Type d to delete a partition. For instance, to delete an existing /dev/hda1:

Code listing 5: Deleting a partition

```
Command (m for help): d
Partition number (1-4): 1
```

The partition has been scheduled for deletion. It will no longer show up if you type p, but it will not be erased until your changes have been saved. If you made a mistake and want to abort without saving your changes, type q immediately and hit enter and your partition will not be deleted.

Now, assuming that you do indeed want to wipe out all the partitions on your system, repeatedly type p to print out a partition listing and then type d and the number of the partition to delete it. Eventually, you'll end up with a partition table with nothing in it:

Code listing 6: An empty partition table

```
Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

Device Boot    Start       End    Blocks   Id  System

Command (m for help):
```

Now that the in-memory partition table is empty, we're ready to create the partitions. We will use a default partitioning scheme as discussed previously. Of course, don't follow these instructions to the letter if you don't want the same partitioning scheme!

Creating the Boot Partition

We first create a small boot partition. Type n to create a new partition, then p to select a primary partition, followed by 1 to select the first primary partition. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, type +32M to create a partition 32 Mbyte in size:

Code listing 7: Creating the boot partition

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-3876, default 1): (Hit Enter)
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +32M
```

Now, when you type `p`, you should see the following partition printout:

Code listing 8: Created boot partition

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

Device Boot     Start       End     Blocks   Id  System
/dev/hda1           1        14     105808+  83  Linux
```

We need to make this partition bootable. Type `a` to toggle the bootable flag on this partition. If you press `p` again, you will notice that an `*` is placed in the "Boot" column.

Creating the Swap Partition

Let's now create the swap partition. To do this, type `n` to create a new partition, then `p` to tell fdisk that you want a primary partition. Then type `2` to create the second primary partition, `/dev/hda2` in our case. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, type `+512M` to create a partition 512MB in size. After you've done this, type `t` to set the partition type, `2` to select the partition you just created and then type in `82` to set the partition type to "Linux Swap". After completing these steps, typing `p` should display a partition table that looks similar to this:

Code listing 9: Partition listing after creating a swap partition

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

Device Boot     Start       End     Blocks   Id  System
/dev/hda1 *         1        14     105808+  83  Linux
/dev/hda2          15        81     506520   82  Linux swap
```

Creating the Root Partition

Finally, let's create the root partition. To do this, type `n` to create a new partition, then `p` to tell fdisk that you want a primary partition. Then type `3` to create the third primary partition, `/dev/hda3` in our case. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, hit enter to create a partition that takes up the rest of the remaining space on your disk. After completing these steps, typing `p` should display a partition table that looks similar to this:

Code listing 10: Partition listing after creating the root partition

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

Device Boot     Start       End     Blocks   Id  System
/dev/hda1 *         1        14     105808+  83  Linux
/dev/hda2          15        81     506520   82  Linux swap
/dev/hda3          82      3876   28690200   83  Linux
```

Saving the Partition Layout

To save the partition layout and exit `fdisk`, type `w`.

Code listing 11: Save and exit fdisk

```
Command (m for help): w
```

Now that your partitions are created, you can now continue with Creating Filesystems.

4.d. Using fdisk on Alpha to Partition your Disk

Important: Only users with Alpha based systems should read this section.

The following parts explain how to create the example slice layout described previously, namely:

Slice	Description
/dev/sdaa	Swap slice
/dev/sdab	Root slice
/dev/sdac	Full disk (required)

Change your slice layout according to your own will.

Identifying Available Disks

To figure out what disks you have running, use the following commands:

Code listing 12: Identifying available disks

```
(For IDE disks)      # dmesg | grep 'drive$'
(For SCSI disks)     # dmesg | grep 'scsi'
```

From this output you should be able to see what disks were detected and their respective `/dev` entry. In the following parts we assume that the disk is a SCSI disk on `/dev/sda`.

Now fire up `fdisk`:

Code listing 13: Starting fdisk

```
# fdisk /dev/sda
```

Deleting All Slices

We start with deleting all slices except the 'c'-slice. The following shows how to delete a slice (in the example we use 'a'). Repeat the process to delete all other slices (again, except the 'c'-slice).

Use `p` to view all existing slices. `d` is used to delete a slice.

Code listing 14: Deleting a slice

```
BSD disklabel command (m for help): p

8 partitions:
#       start       end      size     fstype   [fsize bsize   cpg]
  a:        1       235*      234*    4.2BSD     1024  8192    16
  b:      235*      469*      234*      swap
  c:        1      5290*     5289*    unused        0     0
  d:      469*     2076*     1607*    unused        0     0
  e:     2076*     3683*     1607*    unused        0     0
  f:     3683*     5290*     1607*    unused        0     0
  g:      469*     1749*     1280     4.2BSD     1024  8192    16
  h:     1749*     5290*     3541*    unused        0     0

BSD disklabel command (m for help): d
Partition (a-h): a
```

After repeating this process for all slices, a listing should show you something similar to this:

Code listing 15: Viewing an empty scheme

```
BSD disklabel command (m for help): p

3 partitions:
#       start       end      size     fstype   [fsize bsize   cpg]
  c:        1      5290*     5289*    unused        0     0
```

Creating the Swap Slice

On Alpha based systems you don't need a separate boot partition. However, the first cylinder cannot be used as the `aboot` image will be placed there.

We will create a swap slice starting at the third cylinder, with a total size of 1 Gbyte. Use `n` to create a new slice. After creating the slice, we will change its type to `1`, meaning swap.

Code listing 16: Creating the swap slice

```
BSD disklabel command (m for help): n
Partition (a-p): a
First cylinder (1-5290, default 1): 3
Last cylinder or +size or +sizeM or +sizeK (3-5290, default 5290): +1024M

BSD disklabel command (m for help): t
Partition (a-c): a
Hex code (type L to list codes): 1
```

After these steps you should see a layout similar to the following:

Code listing 17: Slice layout after creating the swap slice

```
BSD disklabel command (m for help): p

3 partitions:
#       start       end      size     fstype   [fsize bsize   cpg]
  a:        3      1003      1001       swap
  c:        1      5290*     5289*    unused        0     0
```

Create the Root Slice

We will now create the root slice, starting from the first cylinder after the swap slice. Use the `p` command to view where the swap slice ends. In our example, this is at 1003, making the root partition start at 1004.

Another problem is that there is currently a bug in `fdisk` making it think the number of available cylinders is one above the real number of cylinders. In other words, when you are asked for the last cylinder, decrease the cylinder number (in this example: 5290) with one.

When the partition is created, we change the type to `8`, for ext2.

Code listing 18: Creating the root slice

```
D disklabel command (m for help): n
Partition (a-p): b
First cylinder (1-5290, default 1): 1004
Last cylinder or +size or +sizeM or +sizeK (1004-5290, default 5290): 5289

BSD disklabel command (m for help): t
Partition (a-c): b
Hex code (type L to list codes): 8
```

Your slice layout should now be similar to this:

Code listing 19: Viewing the slice layout

```
BSD disklabel command (m for help): p

3 partitions:
#       start       end      size     fstype   [fsize bsize   cpg]
  a:        3      1003      1001       swap
  b:     1004      5289      4286       ext2
  c:        1      5290*     5289*    unused        0     0
```

Save the Slice Layout and Exit

Save `fdisk` by typing `w`. This will also save your slice layout.

Code listing 20: Save and exit fdisk

```
Command (m for help): w
```

Now that your slices are created, you can now continue with Creating Filesystems.

4.e. Using fdisk on SPARC to Partition your Disk

Important: Only users with SPARC based systems should read this section.

The following parts explain how to create the example partition layout described previously, namely:

Partition	Description
`/dev/hda1`	Boot partition
`/dev/hda2`	Swap partition
`/dev/hda3`	Sun Disk Label (required)
`/dev/hda4`	Root partition

Change your partition layout according to your own will.

Firing up fdisk

Start `fdisk` with your disk as argument:

Code listing 21: Starting fdisk

```
# fdisk /dev/hda
```

You should be greeted with the fdisk prompt:

Code listing 22: The fdisk prompt

```
Command (m for help):
```

To view the available partitions, type in `p`:

Code listing 23: Listing available partitions

```
Command (m for help): p

Disk /dev/hda (Sun disk label): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
/dev/hda1   *        1        14    105808+  83  Linux
/dev/hda2           15        81    506520   82  Linux swap
/dev/hda3            0      3876  29302528    5  Whole Disk
/dev/hda4           82      3876  28690200   83  Linux
```

Note the `Sun disk label` in the output. If this is missing, the disk is using the DOS-partitioning, not the Sun partitioning. In this case, use `s` to ensure that the disk has a sun partition table.

Deleting Existing Partitions

It's time to delete any existing partitions. To do this, type `d` and hit Enter. You will then be prompted for the partition number you would like to delete. To delete a pre-existing `/dev/hda1`, you would type:

Code listing 24: Deleting a partition

```
Command (m for help): d
Partition number (1-4): 1
```

Assuming you want to remove all existing partitions, press `p` to view the available partitions, and `d` to delete those one by one. If you feel like you made an error, press `q` immediately -- `fdisk` doesn't immediately change the partitions but keeps the changes in memory. Only when you press `w` are the partitions saved.

After deleting all partitions, you should have a partition layout similar to the following:

Code listing 25: View an empty partition scheme

```
Command (m for help): p

Disk /dev/hda (Sun disk label): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
```

Create the Sun Disk Label

Now that the in-memory partition table is empty, we're ready to create the Sun Disk Label partition. To do this, type `n` to create a new partition, then type `3` to create the partition. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, hit enter. After you've done this, type `t` to set the partition type, and then type in `5` to set the partition type to "Whole disk".

Code listing 26: Steps to create a Sun Disk Label

```
Command (m for help): n
Partition number (1-4): 3
First cylinder (1-3876, default 0): 0
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): (Press Enter)
Using default value 3876

Command (m for help): t
Partition number (1-8): 3
Hex code (type L to list codes): 5
```

After completing these steps, typing `p` should display a partition table that looks similar to this:

Code listing 27: View the partition layout

```
Command (m for help): p

Disk /dev/hda (Sun disk lable): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
/dev/hda3             0      3876  29302528    5  Whole disk
```

Creating the Boot Partition

We're ready to create a boot partition. To do this, type `n` to create a new partition, then type `1` to create the partition. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, type `+32M` to create a partition `32MB` in size. Make sure that the entire boot partition must be contained entirely within the first 2Gb of the disk. You can see output from these steps below:

Code listing 28: Creating a boot partition

```
Command (m for help): n
Partition number (1-4): 1
First cylinder (1-3876, default 1): (Press Enter)
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +32M
```

Now, when you type `p`, you should see the following partition printout:

Code listing 29: Listing the partition layout

```
Command (m for help): p

Disk /dev/hda (Sun disk label): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
/dev/hda3             0      3876  29302528    5  Whole disk
```

Creating a swap partition

Next, let's create the swap partition. To do this, type `n` to create a new partition, then `2` to create the second partition, `/dev/hda2` in our case. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, type `+512M` to create a partition 512MB in size. After you've done this, type `t` to set the partition type, and then type in `82` to set the partition type to "Linux Swap". After completing these steps, typing `p` should display a partition table that looks similar to this:

Code listing 30: Listing of available partitions

```
Command (m for help): p

Disk /dev/hda (Sun disk label): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
/dev/hda2            15        81    506520   82  Linux swap
/dev/hda3             0      3876  29302528    5  Whole disk
```

Creating the root partition

Finally, let's create the root partition. To do this, type `n` to create a new partition, then type `4` to create the third partition, `/dev/hda4` in our case. When prompted for the first cylinder, hit enter. When prompted for the last cylinder, hit enter to create a partition that takes up the rest of the remaining space on your disk. After completing these steps, typing `p` should display a partition table that looks similar to this:

Code listing 31: Listing complete partition table

```
Command (m for help): p

Disk /dev/hda (Sun disk label): 240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 bytes

Device Flag    Start       End    Blocks   Id  System
/dev/hda1          1        14    105808+  83  Linux
/dev/hda2         15        81    506520   82  Linux swap
/dev/hda3          0      3876  29302528    5  Whole disk
/dev/hda4         82      3876  28690200   83  Linux
```

Save and Exit

To save your partition layout and exit `fdisk`, type `w`:

Code listing 32: Save and exit fdisk

```
Command (m for help): w
```

Now that your partitions are created, you can now continue with Creating Filesystems.

4.f. Using mac-fdisk on PPC to Partition your Disk

At this point, create your partitions using `mac-fdisk`:

Code listing 33: Starting mac-fdisk

```
# mac-fdisk /dev/hda
```

First delete the partitions you have cleared previously to make room for your Linux partitions. Use `d` in `mac-fdisk` to delete those partition(s). It will ask for the partition number to delete.

Second, create an Apple_Bootstrap partition by using `b`. It will ask for what block you want to start. If you previously selected `3` as partition number, enter `3p`.

Now create a swap partition by pressing `c`. Again `mac-fdisk` will ask for what block you want to start this partition from. As we used `3` before to create the Apple_Bootstrap partition, you now have to enter `4p`. When you're asked for the size, enter `512M` (or whatever size you want -- 512 is recommended though). When asked for a name, enter `swap` (mandatory).

To create the root partition, enter `c`, followed by `5p` to select from what block the root partition should start. When asked for the size, enter `5p` again. `mac-fdisk` will interprete this as "Use all available space". When asked for the name, enter `root` (mandatory).

To finish up, write the partition to the disk using `w` and `q` to quit `mac-fdisk`.

Now that your partitions are created, you can now continue with Creating Filesystems.

4.g. Using fdisk on HPPA to Partition your Disk

Use `fdisk` to create the partitions you want:

Code listing 34: Partitioning the disk

```
# fdisk /dev/sda
```

PALO needs a special partition to work. You have to create a partition of at least 16Mb at the beginning of your disk. The partition type must be of type f0 (Linux/PA-RISC boot).

Important: If you ignore this and continue without a special PALO partition, your system will stop loving you and fail to start.

Also, if your disk is larger than 2Gb, make sure that the boot partition is in the first 2Gb of your disk. PALO is unable to read a kernel after the 2Gb limit.

Now that your partitions are created, you can now continue with Creating Filesystems.

4.h. Using fdisk on MIPS to Partition your Disk

Creating an SGI Disk Label

All disks in an SGI System require an SGI Disk Label, which serves a similar function as Sun & MS-DOS disklabels -- It stores information about the disk partitions. Creating a new SGI Disk Label will create two special partitions on the disk:

- SGI Volume Header (9th partition): This partition is important. It is where the kernel images will go. To store kernel images, you will utilize the tool known as `dvhtool` to copy kernel images to this partition. You will then be able to boot kernels from this partition via the SGI PROM Monitor.
- SGI Volume (11th partition): This partition is similar in purpose to the Sun Disklabel's third partition of "Whole Disk". This partition spans the entire disk, and should be left untouched. It serves no special purpose other than to assist the PROM in some undocumented fashion (or it is used by IRIX in some way).

Warning: The SGI Volume Header must begin at cylinder 0. Failure to do so means you won't be able to boot from the disk.

The following is an example excerpt from an `fdisk` session. Read and tailor it to your needs...

Code listing 35: Creating an SGI Disklabel

```
# fdisk /dev/sda

Command (m for help): x

Expert command (m for help): m
Command action
   b   move beginning of data in a partition
   c   change number of cylinders
   d   print the raw data in the partition table
   e   list extended partitions
   f   fix partition order
   g   create an IRIX (SGI) partition table
   h   change number of heads
   m   print this menu
   p   print the partition table
   q   quit without saving changes
   r   return to main menu
   s   change number of sectors/track
   v   verify the partition table
   w   write table to disk and exit

Expert command (m for help): g
Building a new SGI disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content will be unrecoverably lost.

Expert command (m for help): r

Command (m for help): p

Disk /dev/sda (SGI disk label): 64 heads, 32 sectors, 17482 cylinders
Units = cylinders of 2048 * 512 bytes

----- partitions -----
Pt#     Device  Info     Start       End   Sectors  Id  System
 9:  /dev/sda1               0         4     10240   0  SGI volhdr
11:  /dev/sda2               0     17481  35803136   6  SGI volume
----- Bootinfo -----
Bootfile: /unix
----- Directory Entries -----

Command (m for help):
```

Note: If your disk already has an existing SGI Disklabel, then fdisk will not allow the creation of a new label. There are two ways around this. One is to create a Sun or MS-DOS disklabel, write the changes to disk, and restart fdisk. The second is to overwrite the partition table with null data via the following command: `dd if=/dev/zero of=/dev/sda bs=512 count=1`.

Getting the SGI Volume Header to just the right size

Now that an SGI Disklabel is created, partitions may now be defined. In the above example, there are already two partitions defined for you. These are the special partitions mentioned above and should not normally be altered. However, for installing Gentoo, we'll need to load multiple kernel images directly into the volume header, as there is no supported SGI Bootloader available in Portage yet. The volume header itself can hold up to eight images of any size, with each image allowed eight-character names.

The process of making the volume header larger isn't exactly straight-forward -- there's a bit of a trick to it. One cannot simply delete and re-add the volume header due to odd fdisk behavior. In the example provided below, we'll create a 50MB Volume header in conjunction with a 50MB /boot partition. The actual layout of your disk may vary, but this is for illustrative purposes only.

Code listing 36: Resizing the SGI Volume Header correctly

```
Command (m for help): n
Partition number (1-16): 1
First cylinder (5-8682, default 5): 51
 Last cylinder (51-8682, default 8682): 101
(Notice how fdisk only allows Partition #1 to be re-created starting at a minimum of cylinder 5)
(Had you attempted to delete & re-create the SGI Volume Header this way, this is the same issue
 you would have encountered.)
(In our example, we want /boot to be 50MB, so we start it at cylinder 51 (the Volume Header needs to
 start at cylinder 0, remember?), and set its ending cylinder to 101, which will roughly be 50MB (+/- 1-5MB))

Command (m for help): d
Partition number (1-16): 9
(Delete Partition #9 (SGI Volume Header))

Command (m for help): n
Partition number (1-16): 9
First cylinder (0-50, default 0): 0
 Last cylinder (0-50, default 50): 50
(Re-Create Partition #9, ending just before Partition #1)
```

Final partition layout

Once this is done, you are safe to create the rest of your partitions as you see fit. After all your partitions are laid out, make sure you set the partition ID of your swap partition to `82`, which is Linux Swap. By default, it will be `83`, Linux Native.

Now that your partitions are created, you can now continue with Creating Filesystems.

4.i. Creating Filesystems

Introduction

Now that your partitions are created, it is time to place a filesystem on them. If you don't care about what filesystem to choose and are happy with what we use as default in this handbook, continue with Applying a Filesystem to a Partition. Otherwise read on to learn about the available filesystems...

Filesystems?

Several filesystems are available. Some of them are found stable on all architectures, others only on a few. The following table lists the available filesystems and the architectures they are known to work on. If an architecture is contained within "(...)" then the filesystem should work but is untested.

Filesystem	Journaled	Architectures
ext2	no	All architectures
ext3	yes	All architectures
reiserfs	yes	x86, hppa, alpha, (mips), (pcc), (amd64)
xfs	yes	x86, alpha, amd64, (ppc)
jfs	yes	x86, alpha, (mips), (amd64)

ext2 is the tried and true Linux filesystem but doesn't have metadata journaling, which means that routine ext2 filesystem checks at startup time can be quite time-consuming. There is now quite a selection of newer-generation journaled filesystems that can be checked for consistency very quickly and are thus generally preferred over their non-journaled counterparts. Journaled filesystems prevent long delays when you boot your system and your filesystem happens to be in an inconsistent state.

ext3 is the journaled version of the ext2 filesystem, providing metadata journaling for fast recovery in addition to other enhanced journaling modes like full data and ordered data journaling. ext3 is a very good and reliable filesystem. It offers generally decent performance under most conditions. Because it does not extensively employ the use of "trees" in its internal design, it doesn't scale very well, meaning that it is not an ideal choice for very large filesystems, or situations where you will be handling very large files or large quantities of files in a single directory. But when used within its design parameters, ext3 is an excellent filesystem.

ReiserFS is a B*-tree based filesystem that has very good overall performance and greatly outperforms both ext2 and ext3 when dealing with small files (files less than 4k), often by a factor of 10x-15x. ReiserFS also scales extremely well and has metadata journaling. As of kernel 2.4.18+, ReiserFS is solid and usable as both general-purpose filesystem and for extreme cases such as the creation of large filesystems, the use of many small files, very large files and directories containing tens of thousands of files.

XFS is a filesystem with metadata journaling that is fully supported under Gentoo Linux's xfs-sources kernel. It comes with a robust feature-set and is optimized for scalability. We only recommend using this filesystem on Linux systems with high-end SCSI and/or fibre channel storage and a uninterruptible power supply. Because XFS aggressively caches in-transit data in RAM, improperly designed programs (those that don't take proper precautions when writing files to disk and there are quite a few of them) can lose a good deal of data if the system goes down unexpectedly.

JFS is IBM's high-performance journaling filesystem. It has recently become production-ready and there hasn't been a sufficient track record to comment positively nor negatively on its general stability at this point.

Applying a Filesystem to a Partition

To create a filesystem on a partition or volume, there are tools available for each possible filesystem:

Filesystem	Creation Command
ext2	`mke2fs`
ext3	`mke2fs -j`
reiserfs	`mkreiserfs`
xfs	`mkfs.xfs`
jfs	`mkfs.jfs`

For instance, to have the boot partition (`/dev/hda1` in our example) in ext2 and the root partition (`/dev/hda3` in our example) in ext3 (as in our example), you would use:

Code listing 37: Applying a filesystem on a partition

```
# mke2fs /dev/hda1
# mke2fs -j /dev/hda3
```

Now create the filesystems on your newly created partitions (or logical volumes).

Activating the Swap Partition

`mkswap` is the command that is used to initialize swap partitions:

Code listing 38: Creating a Swap signature

```
# mkswap /dev/hda2
```

To activate the swap partition, use `swapon`:

Note: Knoppix users who already have a swap partition on their system may skip this as Knoppix automatically activates existing swap partitions.

Code listing 39: Activating the swap partition

```
# swapon /dev/hda2
```

Create and activate the swap now.

4.j. Mounting

Now that your partitions are initialized and are housing a filesystem, it is time to mount those partitions. Use the `mount` command. Don't forget to create the necessary mount directories:

Code listing 40: Mounting partitions

```
# mount /dev/hda3 /mnt/gentoo
# mkdir /mnt/gentoo/boot
# mount /dev/hda1 /mnt/gentoo/boot
```

Note: If you want your `/tmp` to reside on a separate partition, be sure to change its permissions after mounting: `chmod 1777 /mnt/gentoo/tmp`. This also holds for `/var/tmp`.

We also need to mount the proc filesystem (a virtual interface with the kernel) on `/proc`. We first create the `/mnt/gentoo/proc` mountpoint:

Code listing 41: Creating the /mnt/gentoo/proc mountpoint

```
# mkdir /mnt/gentoo/proc
```

If you are installing Gentoo from a LiveCD, you just need to mount `proc`:

Code listing 42: Mounting proc

```
# mount -t proc none /mnt/gentoo/proc
```

If you are not installing Gentoo from a Gentoo LiveCD, you need to bindmount `/proc`:

Code listing 43: Bind-mounting proc

```
# mount -o bind /proc /mnt/gentoo/proc
```

Now continue with Installing the Gentoo Installation Files.

5. Installing the Gentoo Installation Files

Content:

- Installing a Stage Tarball
- Default: Using a Stage from the Internet
- Alternative: Using a Stage from the LiveCD
- Installing Portage
- Configuring the Compile Options

5.a. Installing a Stage Tarball

Making your Choice

The next step you need to perform is to install the stage tarball of your choice onto your system. You have the option of downloading the required tarball from the Internet or, if you are booted from one of the Gentoo LiveCDs, copy it over from the CD itself.

- Default: Using a Stage from the Internet
- Alternative: Using a Stage from the LiveCD

5.b. Default: Using a Stage from the Internet

Downloading the Stage Tarball

Go to the Gentoo mountpoint at which you mounted your filesystems (most likely `/mnt/gentoo`):

Code listing 1: Going to the Gentoo mountpoint

```
# cd /mnt/gentoo
```

Depending on your installation medium, you have a couple of tools available to download a stage. If you have `lynx` available, then you can immediately surf to the Gentoo mirrorlist and choose a mirror close to you. Then pick the `releases/` directory, followed by the Gentoo version (`2004.0`) and your architecture (for instance `x86/`) to finish up with the `stages/` directory. There you should see all available stage files for your architecture. Select one and press `D` to download. When you're finished, press `Q` to quit the browser.

Note: The Gentoo Hardened stages (for the x86 architecture) are inside the subdirectory `hardened/stages/`.

Code listing 2: Surfing to the mirror listing with lynx

```
# lynx http://www.gentoo.org/main/en/mirrors.xml
```

If you do not have `lynx`, you should have `links2` at your disposal. `links2` is more powerful than `lynx`, but has some drawbacks. One of them is that it doesn't listen to the proxy variables we have declared previously. If you need to setup a proxy, use `links2 -http-proxy proxy.server.com:8080`. From there on, you should follow the same steps as with `lynx` as they are equivalent.

Code listing 3: Surfing to the mirror listing with links2

```
(Without proxy)    # links2 http://www.gentoo.org/main/en/mirrors.xml
(With proxy)       # links2 -http-proxy proxy.server.com:8080 http://www.gentoo.org/main/en/mirrors.xml
```

If you want to check the integrity of the downloaded stage tarball, use `md5sum` and compare the output with the MD5 checksum provided on the mirror.

Code listing 4: Checking integrity of a stage tarball

```
# md5sum stage1-x86-20030910.tar.bz2
6cda1cc745ba882731ac07fbae0dd973  stage1-x86-20030910.tar.bz2
```

Unpacking the Stage Tarball

Now unpack your downloaded stage onto your system. We use GNU's `tar` to proceed as it is the easiest method:

Code listing 5: Unpacking the stage

```
# tar -xvjpf stage?-*.tar.bz2
```

Make sure that you use the same options (`-xvjpf`). The `x` stands for Extract, the `v` for Verbose (okay, yes, this is optional), the `j` for Decompress with bzip2, the `p` for Preserve permissions and the `f` to denote that we want to extract a file, not standard input.

Now that the stage is installed, continue with Installing Portage.

5.c. Alternative: Using a Stage from the LiveCD

Extracting the Stage Tarball

The stages on the CD reside in the `/mnt/cdrom/stages` directory. To see a listing of available stages, use `ls`:

Code listing 6: List all available stages

```
# ls /mnt/cdrom/stages
```

If the system replies with an error, you may need to mount the CD-ROM first:

Code listing 7: Mounting the CD-ROM

```
# ls /mnt/cdrom/stages
ls: /mnt/cdrom/stages: No such file or directory
# mount /dev/cdroms/cdrom0 /mnt/cdrom
# ls /mnt/cdrom/stages
```

Now go into your Gentoo mountpoint (usually `/mnt/gentoo`):

Code listing 8: Changing directory to /mnt/gentoo

```
# cd /mnt/gentoo
```

We will now extract the stage tarball of your choice. We will do this with the GNU `tar` tool. Make sure you use the same options (`-xvjpf`)! In the next example, we extract the stage tarball `stage3-20031011.tar.bz2`. Be sure to substitute the tarball filename with your stage.

Code listing 9: Extracting the stage tarball

```
# tar -xvjpf /mnt/cdrom/stages/stage3-20031011.tar.bz2
```

Now that the stage is installed, continue with Installing Portage.

5.d. Installing Portage

Network or No Network?

If you don't have a working network connection, you have to install a portage snapshot provided by one of our LiveCDs. If you want to use prebuilt packages later on to speed up the installation, you must use a portage snapshot, either from the LiveCD or from one of our mirrors. Other users will download a fully updated Portage tree using `emerge` later on.

Continue with the appropriate part:

- Installing a Portage Snapshot and Source Code from LiveCD (for networkless installations or GRP installations)
- Installing a Portage Snapshot from a Gentoo Mirror (for GRP installations)
- Configuring the Compile Options (all other installation methods)

Installing a Portage Snapshot and Source Code from LiveCD

There is a Portage snapshot available on some LiveCDs. Since you are reading this, we can safely assume you are using such a LiveCD. To install this snapshot, take a look inside `/mnt/cdrom/snapshots/` to see what snapshot we have available:

Code listing 10: Checking the /mnt/cdrom/snapshots content

```
# ls /mnt/cdrom/snapshots
```

Now extract the snapshot using the following construct. Again, make sure you use the correct options to `tar`. Also, the `-C` is with a capital `C`, not `c`. In the next example we use `portage-20031011.tar.bz2` as the snapshot filename. Be sure to substitute with your snapshot.

Code listing 11: Extracting a Portage snapshot

```
# tar -xvjf /mnt/cdrom/snapshots/portage-20031011.tar.bz2 -C /mnt/gentoo/usr
```

You also need to copy over all source code from the CD:

Code listing 12: Copy over source code

```
# mkdir /mnt/gentoo/usr/portage/distfiles
# cp /mnt/cdrom/distfiles/* /mnt/gentoo/usr/portage/distfiles/
```

Now that your Portage snapshot is installed, continue with Configuring the Compile Options.

Installing a Portage Snapshot from a Gentoo Mirror

In order to succesfully use GRP, you need to download a matching Portage snapshot. Go to one of our mirrors with `lynx` (or `links2`) and download the Portage snapshot available from `releases/2004.0/your_architecture/snapshots`. Be sure you are located inside `/mnt/gentoo` so that the downloaded snapshot is saved on your disk.

Code listing 13: Downloading a Portage Snapshot

```
# cd /mnt/gentoo
# lynx http://www.gentoo.org/main/en/mirrors.xml
```

Now extract the snapshot using the following construct. Again, make sure you use the correct options to `tar`. Also, the `-C` is with a capital `C`, not `c`. In the next example we use `portage-20031011.tar.bz2` as the snapshot filename. Be sure to substitute with your snapshot.

Code listing 14: Extracting a Portage snapshot

```
# tar -xvjf /mnt/gentoo/portage-20031011.tar.bz2 -C /mnt/gentoo/usr
```

Now that your Portage snapshot is installed, continue with Configuring the Compile Options.

5.e. Configuring the Compile Options

Introduction

To optimize Gentoo, you can set a couple of variables which impact Portage behaviour. All those variables can be set as environment variables (using `export`) but that isn't permanent. To keep your settings, Portage provides you with `/etc/make.conf`, a configuration file for Portage. It is this file we will edit now.

Note: A commented listing of all possible variables can be found in `/mnt/gentoo/etc/make.conf.example`. For a successful Gentoo installation you'll only need to set the variables which are mentioned beneath.

Fire up your favorite editor (in this guide we use `nano`) so we can alter the optimization variables we will discuss hereafter.

Code listing 15: Opening /etc/make.conf

```
# nano -w /mnt/gentoo/etc/make.conf
```

As you probably notice now, the `make.conf.example` file is structured in a generic way: commented lines start with "#", other lines define variables using the `VARIABLE="content"` syntax. Several of those variables are discussed next.

CHOST

Warning: Although it might be tempting for non-stage1 users, they should not change the `CHOST` setting in `make.conf`. Doing so might render their system unusable. Again: only change this variable if you use a stage1 installation.

The `CHOST` variable defines what architecture `gcc` has to compile programs for. The possibilities are:

Architecture	Subarchitecture	CHOST Setting
x86	i386	i386-pc-linux-gnu
x86	i486	i486-pc-linux-gnu
x86	i586	i586-pc-linux-gnu
x86	i686 and above (incl. athlon)	i686-pc-linux-gnu
alpha		alpha-unknown-linux-gnu
ppc		powerpc-unknown-linux-gnu
sparc		sparc-unknown-linux-gnu
hppa	(generic)	hppa-unknown-linux-gnu
hppa	pa7000	hppa1.1-unknown-linux-gnu
hppa	pa8000 and above	hppa2.0-unknown-linux-gnu
mips		mips-unknown-linux-gnu
amd64		x86_64-pc-linux-gnu

CFLAGS and CXXFLAGS

The `CFLAGS` and `CXXFLAGS` variables define the optimization flags for the `gcc` C and C++ compiler respectively. Although we define those generally here, you will only have maximum performance if you optimize these flags for each program separately. The reason for this is because every program is different.

In `make.conf` you should define the optimization flags you think will make your system the most responsive generally. Don't place experimental settings in this variable; too much optimization can make programs behave bad (crash, or even worse, malfunction).

We will not explain all possible optimization options. If you want to know them all, read the GNU Online Manual(s) or the `gcc` info page (`info gcc` -- only works on a working Linux system). The `make.conf.example` file itself also contains lots of examples and information; don't forget to read it too.

A first setting is the `-march=` flag, which specifies the name of the target architecture. Possible options are described in the `make.conf.example` file (as comments). For instance, for the x86 Athlon XP architecture:

Code listing 16: The GCC march setting

```
-march=athlon-xp
```

A second one is the `-O` flag, which specifies the `gcc` optimization class flag. Possible classes are `s` (for size-optimized), `0` (for no optimizations), `1`, `2` or `3` for more speed-optimization flags (every class has the same flags as the one before, plus some extras). For instance, for a class-2 optimization:

Code listing 17: The GCC O setting

```
-O2
```

Other popular optimization flags are `-pipe` (use pipes rather than temporary files for communication between the various stages of compilation) and `-fomit-frame-pointer` (which doesn't keep the frame pointer in a register for functions that don't need one).

When you define the `CFLAGS` and `CXXFLAGS`, you should combine several optimization flags, like in the following example:

Code listing 18: Defining the CFLAGS and CXXFLAGS variable

```
CFLAGS="-march=athlon-xp -pipe -O2"
CXXFLAGS="${CFLAGS}"                  # Use the same settings for both variables
```

MAKEOPTS

With `MAKEOPTS` you define how many parallel compilations should occur when you install a package. The suggested number is the number of CPUs in your system plus one.

Code listing 19: MAKEOPTS for a regular, 1-CPU system

```
MAKEOPTS="-j2"
```

Ready, Set, Go!

Update your `/mnt/gentoo/etc/make.conf` to your own will and save. You are now ready to continue with Installing the Gentoo Base System.

6. Installing the Gentoo Base System

Content:

- Chrooting
- Differences between Stage1, Stage2 and Stage3
- Progressing from Stage1 to Stage2
- Progressing from Stage2 to Stage3
- Optional: Preparing for GRP

6.a. Chrooting

Optional: Selecting Mirrors

If you have booted from a Gentoo LiveCD, you are able to use `mirrorselect` to update `/etc/make.conf` so fast mirrors are used for both Portage and source code (of course in case you have a working network connection):

Code listing 1: Selecting fast mirrors

```
# mirrorselect -a -s4 -o >> /mnt/gentoo/etc/make.conf
```

If for some reason `mirrorselect` fails, don't panic. This step is completely optional. If `mirrorselect` fails, the default values suffice.

Copy DNS Info

One thing still remains to be done before we enter the new environment and that is copying over the DNS information in `/etc/resolv.conf`. You need to do this to ensure that networking still works even after entering the new environment. `/etc/resolv.conf` contains the nameservers for your network.

Code listing 2: Copy over DNS information

```
# cp /etc/resolv.conf /mnt/gentoo/etc/resolv.conf
```

Optional: Mounting /dev

Knoppix users (and people that install Gentoo from an installation medium that does not use DevFS) should now bind-mount the `/dev` structure. If you use one of our LiveCDs you can skip this step.

Code listing 3: Bind-mounting /dev

```
# mkdir /mnt/gentoo/dev
# mount -o bind /dev /mnt/gentoo/dev
```

Entering the new Environment

Now that all partitions are initialized and the base environment installed, it is time to enter our new installation environment by chrooting into it. This means that we change from the current installation environment (LiveCD or other installation medium) to your installation system (namely the initialized partitions).

This chrooting is done in three steps. First we will change the root from `/` (on the installation medium) to `/mnt/gentoo` (on your partitions) using `chroot`. Then we will create a new environment using `env-update`, which essentially creates environment variables. Finally, we load those variables into memory using `source`.

Code listing 4: Chrooting into the new environment

```
# chroot /mnt/gentoo /bin/bash
# env-update
Regenerating /etc/ld.so.cache...
# source /etc/profile
```

Congratulations! You are now inside your own Gentoo Linux environment. Of course it is far from finished, which is why the installation still has some sections left :-)

Optional: Updating Portage

If you haven't installed a Portage snapshot in the previous chapter, you must download a recent Portage tree from the Internet. `emerge sync` does this for you. Other users should skip this and continue with Configuring the USE variable.

Code listing 5: Updating Portage

```
# emerge sync
(In case you are unable to use rsync, use "emerge-webrsync" which
downloads and installs a portage snapshot for you)
# emerge-webrsync
```

If you are warned that a new Portage version is available and that you should update Portage, you can safely ignore it. Portage will be updated for you later on during the installation.

Configuring the USE variable

`USE` is one of the most powerful variables Gentoo provides to its users. Several programs can be compiled with or without optional support for certain items. For instance, some programs can be compiled with gtk-support, or with qt-support. Others can be compiled with or without SSL support. Some programs can even be compiled with framebuffer support (svgalib) instead of X11 support (X-server).

Most distributions compile their packages with support for as much as possible, increasing the size of the programs and startup time, not to mention an enormous amount of dependencies. With Gentoo you can define what options a package should be compiled with. This is where `USE` comes into play.

In the `USE` variable you define keywords which are mapped onto compile-options. For instance, ssl will compile ssl-support in the programs that support it. -X will remove X-server support (note the minus sign in front). gnome gtk -kde -qt will compile your programs with gnome (and gtk) support, and not with kde (and qt) support, making your system fully tweaked for GNOME.

The default `USE` settings are placed in `/etc/make.profile/make.defaults`. What you place in `/etc/make.conf` is calculated against these defaults settings. If you add something to the `USE` setting, it is added to the default list. If you remove something from the `USE` setting (by placing a minus sign in front of it) it is removed from the default list (if it was in the default list at all). Never alter anything inside the `/etc/make.profile` directory; it gets overwritten when you update Portage!

A full description on `USE` can be found in the second part of the Gentoo Handbook, Chapter 1: USE flags. A full description on the available USE flags can be found on your system in `/usr/portage/profiles/use.desc`.

Code listing 6: Viewing available USE flags

```
# less /usr/portage/profiles/use.desc
```

As an example we show a `USE` setting for a KDE-based system with DVD, ALSA and CD Recording support:

Code listing 7: Opening /etc/make.conf

```
# nano -w /etc/make.conf
```

Code listing 8: USE setting

```
USE="-gtk -gnome qt kde dvd alsa cdr"
```

Optional: Using Distributed Compiling

If you are interested in using a collection of systems to help in compiling your system you might want to take a look at our DistCC Guide. By using `distcc` you can use the processing power of several systems to aid you with the installation.

6.b. Differences between Stage1, Stage2 and Stage3

Now take a seat and think of your previous steps. We asked you to select a stage1, stage2 or stage3 and warned you that your choice is important for further installation steps. Well, this is the first place where your choice defines the further steps.

- If you chose stage1, then you have to follow both steps in this chapter (starting with Progressing from Stage1 to Stage2)
- If you chose stage2 you only can skip the first step and immediately start with the second one (Progressing from Stage2 to Stage3)
- If you chose stage3 (either with or without GRP) then you can skip both steps. If you want to use GRP, continue with Optional: Preparing for GRP. Otherwise continue with Configuring the Kernel

6.c. Progressing from Stage1 to Stage2

Introduction to Bootstrapping

So, you want to compile everything from scratch? Okay then :-)

In this step, we will bootstrap your Gentoo system. This takes a long time, but the result is a system that has been optimized from the ground up for your specific machine and needs.

Bootstrapping means building the GNU C Library, GNU Compiler Collection and several other key system programs. The GNU Compiler Collection even has to be built twice: first with the "generic" compiler we provide, and a second time with the compiler you then just built.

Before starting the bootstrap, we list a couple of options you might or might not want. If you do not want to read those, continue with Bootstrapping the System.

Optional: Decreasing Compilation Time

If you want to speed up the bootstrapping, you can temporarily deselect java-support. This means that the GNU Compiler Collection and the GNU C Library will be compiled without java-support (which decreases compilation time considerably). Although this means that you wont have the GNU Java Compiler (`gcj`) this does not mean that your system won't be able to use java applets and other java-related stuff.

To deselect java-support temporarily, define USE="-java" before firing up the bootstrap script.

Code listing 9: Deselecting java support

```
# export USE="-java"
```

Don't forget to unset the variable after bootstrapping:

Code listing 10: Unsetting USE

```
# unset USE
```

Optional: Downloading the Sources First

If you haven't copied over all source code before, then the bootstrap script will download all necessary files. It goes without saying that this only works if you have a working network connnection :-) If you want to download the source code first and later bootstrap the system (for instance because you don't want to have your internet connection open during the compilation) use the -f option of the bootstrap script, which will fetch (hence the letter f) all source code for you.

Code listing 11: Downloading the necessary sources

```
# cd /usr/portage
# scripts/bootstrap.sh -f
```

Bootstrapping the System

Okay then, take your keyboard and punch in the next commands to start the bootstrap. Then go amuse yourself with something else (for instance harass Gentoo developers on #gentoo), because this step takes quite some time to finish.

Code listing 12: Bootstrapping the system

```
# cd /usr/portage
# scripts/bootstrap.sh
```

If you have altered the `CHOST` setting in `/etc/make.conf` previously, you need to reinitialize some variables in order for `gcc` to work fast:

Code listing 13: Reinitialize environment variables

```
# source /etc/profile
```

Now continue with the next step, Progressing from Stage2 to Stage3.

6.d. Progressing from Stage2 to Stage3

Introduction

If you are reading this section, then you have a bootstrapped system (either because you bootstrapped it previously, or you are using a stage2). Then it is now time to build all system packages.

All system packages? No, not really. In this step, you will build the system packages of which there are no alternatives to use. Some system packages have several alternatives (such as system loggers) and as Gentoo is all about choices, we don't want to force one upon you.

Optional: Viewing what will be done

If you want to know what packages will be installed, execute `emerge --pretend system`. This will list all packages that will be built. As this list is pretty big, you should also use a pager like `less` or `more` to go up and down the list.

Code listing 14: View what 'emerge system' will do

```
# emerge --pretend system | less
```

Optional: Downloading the Sources

If you want emerge to download the sources before you continue (for instance because you don't want the internet connection to be left open while you are building all packages) you can use the --fetchonly option of emerge which will fetch all sources for you.

Code listing 15: Fetching the sources

```
# emerge --fetchonly system
```

Building the System

To start building the system, execute `emerge system`. Then go do something to keep your mind busy, because this step takes a long time to complete.

Code listing 16: Building the System

```
# emerge system
```

When the building has completed, continue with Configuring the Kernel.

6.e. Optional: Preparing for GRP

Introduction

If you are booted from a x86 or ppc CD-1 LiveCD you can relax and continue with Configuring the Kernel as the installation of prebuilt packages happens at the very end of the installation.

If you are booted from a different architecture LiveCD and you want to use the prebuilt packages provided by the LiveCD, continue with Copying over the GRP packages.

If you want to use the prebuilt packages provided by a Gentoo mirror, continue with Configuring Portage for GRP Downloads.

Copying over the GRP packages

You should now copy over the packages onto your filesystem so that Portage is able to use them. First of all, open a second terminal by pressing `Alt-F2`. This is needed as we need to work from the LiveCD, not from the chrooted environment you're currently working in.

You should be greeted by a root prompt (`#`). Copy over the packages using the following commands:

Code listing 17: Copy over precompiled packages

```
# mkdir -p /mnt/gentoo/usr/portage/packages/All
# cp /mnt/cdrom/packages/All/* /mnt/gentoo/usr/portage/packages/All/
```

After this step has completed, return to the chrooted environment by pressing `Alt-F1`.

Now pay close attention! Your Portage snapshot is in place and the GRP packages are ready to be used. However, Portage doesn't automagically use them unless you tell it to. Luckily, this is hardly difficult: every time you are asked to install a package using emerge, you must add `--usepkg` as an option:

Code listing 18: Example for Installing a GRP Package

```
(Example without GRP)
# emerge vanilla-sources

(Example with GRP)
# emerge --usepkg vanilla-sources
```

That's all there is to it. Just don't forget to add `--usepkg`. Now continue with Configuring the Kernel.

Configuring Portage for GRP Downloads

First of all, you need to edit `/etc/make.conf` and define the `PORTAGE_BINHOST` variable so that it points to the server from which you want to download the GRP packages. Please check our mirror list for the available mirrors.

Code listing 19: Editing /mnt/gentoo/etc/make.conf

```
# nano -w /etc/make.conf
```

Code listing 20: Setting the PORTAGE_BINHOST variable

```
PORTAGE_BINHOST="ftp://some.mirror.com/pub/gentoo/grp/2004/athlon-xp"
```

Save and exit (by pressing Ctrl-X and confirming). With this in place, you must now pay close attention. Portage will not automagically download the GRP packages if you don't instruct it to. However, this isn't hard: every time you are asked to install a package using `emerge`, you must add `--getbinpkg` as an option:

Code listing 21: Example for Downloading GRP Packages

```
(Example without downloading GRP)
# emerge vanilla-sources

(Example with downloading GRP)
# emerge --getbinpkg vanilla-sources
```

That's all there is to it. Just don't forget to add `--getbinpkg`. Now continue with Configuring the Kernel.

7. Configuring the Kernel

Content:

- Timezone
- Installing the Sources
- Default: Manual Configuration
- Alternative: Using genkernel
- Installing Separate Kernel Modules

7.a. Timezone

You first need to select your timezone so that your system knows where it is located. Look for your timezone in `/usr/share/zoneinfo`, then make a symlink to `/etc/localtime` using `ln`:

Code listing 1: Setting the timezone information

```
# ls /usr/share/zoneinfo
(Suppose you want to use GMT)
# ln -sf /usr/share/zoneinfo/GMT /etc/localtime
```

7.b. Installing the Sources

Choosing a Kernel

The core around which all distributions are built is the Linux kernel. It is the layer between the user programs and your system hardware. Gentoo provides its users several possible kernel sources. A full listing with description is available at the Gentoo Kernel Guide.

For x86-based systems we have, amongst other kernels, `vanilla-sources` (the default kernel source as developed by the linux-kernel developers), `gentoo-sources` (kernel source patched with performance-enhancing features), `gentoo-dev-sources` (kernel v2.6 source patched with performance-enhancing features and stability improvements), `xfs-sources` (kernel source with the latest XFS support), `gs-sources` (kernel source patched for server usage), `gaming-sources` (kernel source patched for optimal gaming performance), `development-sources` (vanilla 2.6 kernel source), ...

For alpha-based systems we have `vanilla-sources` (the default kernel source as developed by the linux-kernel developers), `alpha-sources` (kernel source optimized for alpha users) and `compaq-sources` (kernel source as used by RedHat for Alpha, maintained by Compaq).

For sparc-based systems we have `vanilla-sources` (the default kernel source as developed by the linux-kernel developers) and `sparc-sources` (kernel source optimized for SPARC users).

MIPS-based systems can choose from `mips-sources` (the default kernel source for the MIPS architecture) and `mips-prepatch-sources` (prerelease kernel tree).

For AMD64-based systems we have `gentoo-dev-sources` (kernel v2.6 source patched with amd64 specific fixes for stability, performance and hardware support).

Other architectures should use the kernel source specifically optimized for their architecture: `hppa-sources` (HPPA) or `ppc-sources` (PowerPC).

Choose your kernel source and install it using `emerge`.

In the next example we install the `vanilla-sources` (as `gentoo-sources` isn't available on all architectures). Of course substitute with your choice of sources:

Code listing 2: Installing a kernel source

```
# emerge vanilla-sources
```

When you take a look in `/usr/src` you should see a symlink called `linux` pointing to your kernel source:

Code listing 3: Viewing the kernel source symlink

```
# ls -l /usr/src/linux
lrwxrwxrwx    1 root     root           12 Oct 13 11:04 /usr/src/linux -> linux-2.4.25
```

If this isn't the case (i.e. the symlink points to a different kernel source) change the symlink before you continue:

Code listing 4: Changing the kernel source symlink

```
# rm /usr/src/linux && ln -s /usr/src/linux-2.4.25 /usr/src/linux
```

Now it is time to configure and compile your kernel source. All architectures can use `genkernel` for this, which will build a generic kernel as used by the LiveCD. We explain the "manual" configuration first though, as it is the best way to optimize your environment.

If you want to manually configure your kernel, continue now with Default: Manual Configuration. If you want to use `genkernel` you should read Alternative: Using genkernel instead.

7.c. Default: Manual Configuration

Introduction

Manually configuring a kernel is often seen as the most difficult course every Linux users ever has to go through. Nothing is less true -- after configuring a couple of kernels you don't even remember that it was difficult ;)

However, one thing is true: you must know your system when you start configuring a kernel manually. Most information can be gathered by viewing the contents of `/proc/pci` (or by using `lspci` if available). You can also run `lsmod` to see what kernel modules the LiveCD uses (it might provide you with a nice hint on what to enable).

Now go to your kernel source directory and execute `make menuconfig`. This will fire up an ncurses-based configuration menu.

Code listing 5: Invoking menuconfig

```
# cd /usr/src/linux
# make menuconfig
```

You will be greeted with several configuration sections. We'll first list some options you must activate (otherwise Gentoo will not function, or not function properly without additional tweaks).

Activating Required Options

First of all, activate the use of development and experimental code/drivers. You need this, otherwise some very important code/drivers won't show up:

Code listing 6: Selecting experimental code/drivers

```
Code maturity level options --->
  [*] Prompt for development and/or incomplete code/drivers
```

Now go to `File Systems` and select support for the filesystems you use. Don't compile them as modules, otherwise your Gentoo system will not be able to mount your partitions. Also select `Virtual memory`, `/proc file system`, `/dev file system` + `Automatically mount at boot`:

Code listing 7: Selecting necessary file systems

```
File systems --->
  [*] Virtual memory file system support (former shm fs)
  [*] /proc file system support
  [*] /dev file system support (EXPERIMENTAL)
  [*]   Automatically mount at boot

(Deselect the following unless you have a 2.6 kernel)
  [ ] /dev/pts file system for Unix98 PTYs

(Select one or more of the following options as needed by your system)
  <*> Reiserfs support
  <*> Ext3 journalling file system support
  <*> JFS filesystem support
  <*> Second extended fs support
  <*> XFS filesystem support
```

Note: Users of a 2.6 kernel will find some of the mentioned options under `Pseudo filesystems` which is a subpart of `File systems`.

If you are using PPPoE to connect to the Internet, you will need the following options in the kernel:

Code listing 8: Selecting PPPoE necessary drivers

```
Network device support --->
  <*> PPP (point-to-point protocol) support
  <*>   PPP support for async serial ports
  <*>   PPP support for sync tty ports
```

Note: Users of a 2.6 kernel will find the mentioned options under `Networking support` which is a subpart of `Device Drivers`.

The two compression options won't harm but are not definitely needed, neither does the `PPP over Ethernet` option, that might only be used by `rp-pppoe` when configured to do kernel mode PPPoE.

Note: Users of a 2.6 kernel will find the mentioned options under `Device Drivers`.

If you require it, don't forget to include support in the kernel for your ethernet card.

Now, dependent on your architecture, you might need to select more options:

- Activating x86-recommended Options
- Activating Alpha-recommended Options
- Activating HPPA-recommended Options
- Activating PPC-recommended Options
- Activating SPARC-recommended Options
- Activating MIPS-recommended Options
- Activating AMD64-recommended Options

Activating x86-recommended Options

If you have an Intel CPU that supports HyperThreading (tm), or you have a multi-CPU system, you should activate "Symmetric multi-processing support":

Code listing 9: Activating SMP support

```
Processor type and features  --->
  [*] Symmetric multi-processing support
```

When you've finished configuring the kernel, continue with Compiling and Installing.

Activating Alpha-recommended Options

The following options are recommended for Alpha-users:

Code listing 10: Alpha-specific options

```
General setup --->
  <*> SRM environment through procfs
  <*> Configure uac policy via sysctl

Plug and Play configuration --->
  <*> Plug and Play support
  <M>   ISA Plug and Play support

SCSI support --->
  SCSI low-level drivers --->
    <*> SYM53C8XX Version 2 SCSI support (NEW)
    <*> Qlogic ISP SCSI support

Network device support --->
  Ethernet (10 or 100 Mbit) --->
    <M> DECchip Tulip (dc21x4x) PCI support
    <M> Generic DECchip & DIGITAL EtherWORKS PCI/EISA
    <M> EtherExpressPro/100 support (eepro100)
    <M> EtherExpressPro/100 support (e100)
  Ethernet (1000 Mbit) --->
    <M> Alteon AceNIC
      [*] Omit support for old Tigon I
    <M> Broadcom Tigon3
  [*] FDDI driver support
  <M> Digital DEFEA and DEFPA
  <*> PPP support
    <*> PPP Deflate compression

Character devices --->
  [*] Support for console on serial port
  [*] Direct Rendering Manager

File systems --->
  <*> Kernel automounter version 4 support
  Network File Systems --->
    <*> NFS
      [*] NFSv3 client
      <*> NFS server
      [*] NFSv3 server
  Partition Types --->
    [*] Advanced partition selection
    [*] Alpha OSF partition support
  Native Language Support
    <*> NLS ISO 8859-1

Sound --->
  <M> Sound card support
    <M> OSS sound modules
      [*] Verbose initialisation
      [*] Persistent DMA buffers
      <M> 100% Sound Blaster compatibles
```

When you've finished configuring the kernel, continue with Compiling and Installing.

Activating HPPA-recommended Options

If you have a HIL mouse or keyboard, do not forget to compile in support for them.

Code listing 11: Activating HIL support

```
Input core support --->
  [*] Keyboard support
  [*] Mouse support
  [*] Event interface support
```

If you have no mouse on your HIL port, only use the basic support:

Code listing 12: Basic HIL support

```
HIL support --->
  [*] HIL Keyboard (basic) support
```

If you however want full HIL support, select the following options:

Code listing 13: Full HIL support

```
HIL support --->
  [*] HP System Device Controller i8042 Support
  [*] HIL MLC Support
  [*] HIL Keyboard (full) support
  [*] HIL Mouse & Pointer support
```

Also include display driver support:

Code listing 14: Display Driver support

```
Graphics support --->
  [*] Support for frame buffer devices
      [*] HP STI frame buffer device support
  Console display driver support --->
      [*] STI text console
```

When you're done configuring your kernel, continue with Compiling and Installing.

Activating PPC-recommended Options

First of all, disable ADB raw keycodes:

Code listing 15: Disabling ADB raw keycodes

```
Macintosh Device Drivers --->
  [ ] Support for ADB raw keycodes
```

Also choose the correct RTC support (disable the `Enhanced RTC` option):

Code listing 16: Activating the correct RTC option

```
Character devices --->
  [ ] Enhanced RTC

General setup --->
  [*] Support for /dev/rtc
```

Users of OldWorld machines will want HFS support so they can copy compiled kernels to the MacOS partition.

Code listing 17: Activating HFS support

```
File Systems --->
  [*] HFS Support
```

When you're done configuring your kernel, continue with Compiling and Installing.

Activating SPARC-recommended Options

First activate the correct bus-support:

Code listing 18: Activating SBUS/UPA

```
Console drivers --->
  Frame-buffer support --->
    [*] SBUS and UPA framebuffers
      [*] Creator/Creator3D support     (Only for UPA slot adapter used in many Ultras)
    [*] CGsix (GX,TurboGX) support      (Only for SBUS slot adapter used in many SPARCStations)
```

Of course you want support for the OBP:

Code listing 19: Activating OBP Support

```
Misc Linux/SPARC drivers --->
  [*]  /dev/openprom device support
```

You will also need SCSI-specific support:

Code listing 20: Activating SCSI-specific support

```
SCSI support --->
  SCSI low-level drivers --->
    <*> Sparc ESP Scsi Driver             (Only for SPARC ESP on-board SCSI adapter)
    <*> PTI Qlogic, ISP Driver            (Only for SBUS SCSI controllers from PTI or QLogic)
    <*> SYM53C8XX Version 2 SCSI support  (Only for Ultra 60 on-board SCSI adapter)
```

To support your network card, select one of the following:

Code listing 21: Activating networking support

```
Network device support --->
  Ethernet (10 or 100Mbit) --->
    <*> Sun LANCE support                   (Only for SPARCStation, older Ultra systems, and as Sbus option)
    <*> Sun Happy Meal 10/100baseT support  (Only for Ultra; also supports "qfe" quad-ethernet on PCI and Sbu
```

When you're done configuring your kernel, continue with Compiling and Installing. However, after having compiled the kernel, check its size:

Code listing 22: Check kernel size

```
# ls -lh vmlinux
-rw-r--r--    1 root     root         2.4M Oct 25 14:38 vmlinux
```

If the (uncompressed) size is bigger than 2.5Mb (for Sparc32) or 3.5Mb (for Sparc64), reconfigure your kernel untill it doesn't exceed these limits. One way of accomplishing this is by having most kernel drivers compiled as modules. Ignoring this can lead to a non-booting kernel.

Also, if your kernel is just a tad too big, you can try stripping it using the `strip` command:

Code listing 23: Stripping the kernel

```
# strip -R .comment -R .note vmlinux
```

Activating MIPS-recommended Options

If you are using an Indy/Indigo2 based system, you need to activate support for it.

Code listing 24: Activating SGI IP22

```
Machine selection --->
  [*] Support for SGI IP22 (Indy/Indigo2)
```

If you want to run Irix binaries, include the following option:

Code listing 25: Including IRIX Binary support

```
General setup --->
  [*] Include IRIX binary compatibility
```

If you have ISA/EISA cards in your SGI Indigo2, enable support for it.

Code listing 26: Enabling ISA/EISA support for SGI Indigo2

```
General setup --->
  [*] Indigo-2 (IP22) EISA bus support
  [*]   ISA bus support
```

If you have a SGI parallel port, you can enable support for it. If you have an ISA parallel port you should select "PC-style hardware" instead.

Code listing 27: Enabling SGI Parallel Port Support

```
Parallel port support  --->
  <*> Parallel port support
  <*>   SGI Indy/Indigo2 hardware (EXPERIMENTAL) (NEW)
  <*>   IEEE 1284 transfer modes (NEW)
```

If you want to use the Indigo2 ISA slots, enable the plug and play support.

Code listing 28: Enabling PnP support for ISA

```
Plug and Play configuration  --->
  <*> Plug and Play support
  <*>   ISA Plug and Play support
```

Don’t forget to enable SCSI support, and use the SGI WD93C93 Driver:

Code listing 29: Enabling WD93C93 Driver Support

```
SCSI low-level drivers  --->
  <*> SGI WD93C93 SCSI Driver
```

For network cards you probably need support for the SGI Seeq ethernet controller:

Code listing 30: Enabling SGI Seeq Support

```
Network device support  --->
  Ethernet (10 or 100Mbit)  --->
    [*] Ethernet (10 or 100Mbit)
    [*]   SGI Seeq ethernet controller support
```

Don’t forget to enable serial console support and enable support for the SGI Zilog85C30:

Code listing 31: Enable SGI Zilog85C30 Support

```
Character devices --->
  [*] Non-standard serial port support
  [*]   SGI Zilog85C30 serial support
```

Also don’t forget to enable the Indy/I2 Watchdog support as well as the SGI DS1286 RTC support:

Code listing 32: Enable Watchdog and RTC Support

```
Character Devices --->
  [*] SGI DS1286 RTC support
  Watchdog Cards  --->
    [*] Watchdog Timer Support
    <*>   Indy/I2 Hardware Watchdog
```

You should also enable support for SGI partitions :)

Code listing 33: Enabling Support for SGI Partitions

```
File Systems --->
  Partition Types --->
    [*] Advanced partition selection
    [*]   SGI partition support
```

If you have an SGI Newport (XL Gfx) Card and want to use it, then you’ll want to enable support for it:

Code listing 34: Enabling Support for the SGI Newport Card

```
Console drivers  --->
  <*> SGI Newport Console support (NEW)
```

If you want sound support on your Indy/Indigo2, enable support for it:

Code listing 35: Enabling Support for the SGI HAL2

```
Sound  --->
  <*> Sound card support
  <*>   SGI HAL2 sound (EXPERIMENTAL)
```

When you're done configuring your kernel, continue with Compiling and Installing.

Activating AMD64-recommended Options

If you have a multi-CPU Opteron system, you should activate "Symmetric multi-processing support":

Code listing 36: Activating SMP support

```
Processor type and features --->
  [*] Symmetric multi-processing support
```

When you've finished configuring the kernel, continue with Compiling and Installing.

Compiling and Installing

Now that your kernel is configured, it is time to compile and install it. Exit the configuration and run `make dep && make bzImage modules modules_install`:

Code listing 37: Compiling the kernel

```
(For x86-based systems, 2.4 kernel)
# make dep && make bzImage modules modules_install

(For other systems, 2.4 kernel)
# make dep && make vmlinux modules modules_install

(For 2.6 kernel)
# make && make modules_install
```

When the kernel is done compiling, copy over the kernel image to `/boot`. In the next example we assume you have configured and compiled `vanilla-sources-2.4.25` (which may not be the right kernel for your architecture!):

Code listing 38: Installing the kernel

```
(For x86-based systems)
# cp arch/i386/boot/bzImage /boot/kernel-2.4.25
# cp System.map /boot/System.map-2.4.25

(For amd64-based systems)
# cp arch/x86_64/boot/bzImage /boot/kernel-2.4.25

(For other systems)
# cp vmlinux /boot/kernel-2.4.25
# cp System.map /boot/System.map-2.4.25
```

It is also wise to copy over your kernel configuration file to `/boot`, just in case :)

Code listing 39: Backing up your kernel configuration

```
# cp .config /boot/config-2.4.25
```

If you are a MIPS user and your system doesn't boot ELF kernels, compile the kernel using `make vmlinux.ecoff` instead of `make vmlinux`. The kernel image will be saved as `arch/mips/boot/vmlinux.ecoff` instead of `vmlinux`.

Now continue with Installing Separate Kernel Modules.

7.d. Alternative: Using genkernel

If you are reading this section, you have chosen to use our genkernel script to configure your kernel for you.

Now that your kernel source tree is installed, it's now time to compile your kernel by using our genkernel script to automatically build a kernel for you. genkernel works by configuring a kernel nearly identically to the way our LiveCD kernel is configured. This means that when you use genkernel to build your kernel, your system will generally detect all your hardware at boot-time, just like our Live CD does. Because genkernel doesn't require any manual kernel configuration, it is an ideal solution for those users who may not be comfortable compiling their own kernels.

Now, let's see how to use genkernel. First, emerge the genkernel ebuild:

Code listing 40: Emerging genkernel

```
# emerge genkernel
```

Now, compile your kernel sources by running genkernel all. Be aware though, as genkernel compiles a kernel that supports almost all hardware, this compilation will take quite a while to finish!

Code listing 41: Running genkernel

```
# genkernel all
GenKernel v3.0.1_beta10
* ARCH: x86
* KERNEL VER: 2.4.25
* kernel: configuring source
* kernel: running mrproper
(Output removed to increase readability)
* Kernel compiled successfully!
* Required Kernel Params:
*   : root=/dev/ram0 init=/linuxrc real_root=/dev/$ROOT
*     where $ROOT is the devicenode for your root partition as
*     you should have specified in /etc/fstab
*
* You MUST tell your bootloader to use the generated initrd
*
* Recommended Kernel Params:
*   : vga=0x317 splash=verbose
*
* Do NOT report kernel bugs (configs included) as genkernel bugs.
* Make sure you have the latest genkernel before reporting bugs
*
* For more info see /usr/share/genkernel/README
```

Once genkernel completes, a kernel, full set of modules and initial root disk (initrd) will be created. We will use the kernel and initrd when configuring a boot loader later in this document. Write down the names of the kernel and initrd as you will need it when writing the bootloader configuration file. The initrd will be started immediately after booting to perform hardware autodetection (just like on the Live CD) before your "real" system starts up.

Now, let's perform one more step to get our system to be more like the Live CD -- let's emerge hotplug. While the initrd autodetects hardware that is needed to boot your system, hotplug autodetects everything else. To emerge and enable hotplug, type the following:

Code listing 42: Emerging and enabling hotplug

```
# emerge hotplug
# rc-update add hotplug default
```

7.e. Installing Separate Kernel Modules

Installing Extra Modules

If appropriate, you should emerge ebuilds for any additional hardware that is on your system. Here is a list of kernel-related ebuilds that you could emerge:

Ebuild	Purpose	Command
nvidia-kernel	Accelerated NVIDIA graphics for XFree86	`emerge nvidia-kernel`
nforce-net	On-board ethernet controller on NVIDIA NForce(2) motherboards	`emerge nforce-net`
nforce-audio	On-board audio on NVIDIA NForce(2) motherboards	`emerge nforce-audio`
e100	Intel e100 Fast Ethernet Adapters	`emerge e100`
e1000	Intel e1000 Gigabit Ethernet Adapters	`emerge e1000`
emu10k1	Creative Sound Blaster Live!/Audigy support	`emerge emu10k1`
ati-drivers	Accelerated ATI Radeon 8500+/FireGL graphics for XFree86	`emerge ati-drivers`
ati-drivers-extra	Graphical ATI tools	`emerge ati-drivers-extra`
xfree-drm	Accelerated graphics for ATI Radeon up to 9200, Rage128, Matrox, Voodoo and other cards for XFree86. Please check the `IUSE_VIDEO_CARDS` variable in the `/usr/portage/x11-base/xfree-drm` ebuilds to see what you need to fill in as `yourcard`.	`VIDEO_CARDS="yourcard" emerge xfree-drm`

Beware though, some of these ebuilds might deal with big dependencies. To verify what packages will be installed by emerging an ebuild, use `emerge --pretend`. For instance, for the `emu10k1` package:

Code listing 43: View full installation package listing

```
# emerge --pretend emu10k1
```

If you don't like the packages it wants to install, use `emerge --pretend --verbose` to see what USE-flags are checked when deciding the dependencies:

Code listing 44: View USE-flag usage

```
# emerge --pretend --verbose emu10k1
...
[ebuild  N    ] media-sound/aumix-2.8  +gpm +nls +gtk +gnome +alsa -gtk2
```

In the previous example you can see that one of `emu10k1`'s dependencies (`aumix`) uses the `gtk` and `gnome` USE-flags, making gtk (which depends on XFree) be compiled with it.

If you don't want all this to be compiled, deselect all USE-flags, for instance:

Code listing 45: Emerging emu10k1 with all USE-flags deselected

```
# USE="-gpm -nls -gtk -gnome -alsa" emerge --pretend emu10k1
```

When you're happy with the results, remove the `--pretend` to start installing `emu10k1`.

Configuring the Modules

If you are not using `hotplug`, you should list the modules you want automatically loaded in `/etc/modules.autoload.d/kernel-2.4` (or `kernel-2.6`). You can add extra options to the modules too if you want.

To view all available modules, run the following `find` command. Don't forget to substitute "<kernel version>" with the version of the kernel you just compiled:

Code listing 46: Viewing all available modules

```
# find /lib/modules/<kernel version>/ -type f -iname '*.o' -or -iname '*.ko'
```

For instance, to automatically load the `3c59x.o` module:

Code listing 47: /etc/modules.autoload.d/kernel-2.4 or kernel-2.6

```
3c59x
```

Now run `modules-update` to commit your changes to the `/etc/modules.conf` file:

Code listing 48: Running modules-update

```
# modules-update
```

Continue the installation with Configuring your System.

8. Configuring your System

Content:

- Filesystem Information
- Networking Information
- System Information

8.a. Filesystem Information

What is fstab?

Under Linux, all partitions used by the system must be listed in `/etc/fstab`. This file contains the mountpoints of those partitions (where they are seen in the file system structure), how they should be mounted (special options) and when (automatically or not, can users mount those or not, etc.).

Creating /etc/fstab

`/etc/fstab` uses a special syntax. Every line consists of six fields, separated by whitespace (space(s), tabs or a mixture). Each field has its own meaning:

- The first field shows the partition described (the path to the device file)
- The second field shows the mountpoint at which the partition should be mounted
- The third field shows the filesystem used by the partition
- The fourth field shows the mountoptions used by `mount` when it wants to mount the partition. As every filesystem has its own mountoptions, you are encouraged to read the mount manpage (`man mount`) for a full listing. Multiple mountoptions are comma-separated.
- The fifth field is used by `dump` to determine if the partition needs to be dumped or not. You can generally leave this as `0` (zero).
- The sixth field is used by `fsck` to determine the order in which filesystems should be checked if the system wasn't shut down properly. The root filesystem should have `1` while the rest should have `2` (or `0` in case a filesystem check isn't necessary).

So start `nano` (or your favorite editor) to create your `/etc/fstab`:

Code listing 1: Opening /etc/fstab

```
# nano -w /etc/fstab
```

Let us take a look at how we write down the options for the `/boot` partition. This is just an example, so if your architecture doesn't require a `/boot` partition, don't copy it verbatim.

In our default x86 partitioning example `/boot` is the `/dev/hda1` partition, with `ext2` as filesystem. It shouldn't be mounted automatically (`noauto`) but does need to be checked. So we would write down:

Code listing 2: An example /boot line for /etc/fstab

```
/dev/hda1    /boot      ext2    noauto        1 2
```

Now, to improve performance, most users would want to add the `noatime` option as mountoption, which results in a faster system since access times aren't registered (you don't need those generally anyway):

Code listing 3: An improved /boot line for /etc/fstab

```
/dev/hda1    /boot      ext2    noauto,noatime    1 2
```

If we continue with this, we would end up with the following three lines (for `/boot`, `/` and the swap partition):

Code listing 4: Three /etc/fstab lines

```
/dev/hda1   /boot     ext2    noauto,noatime     1 2
/dev/hda2   none      swap    sw                 0 0
/dev/hda3   /         ext3    noatime            0 1
```

To finish up, you should add a rule for `/proc`, `tmpfs` (required) and for your CD-ROM drive (and of course, if you have other partitions or drives, for those too):

Code listing 5: A full /etc/fstab example

```
/dev/hda1   /boot     ext2    noauto,noatime     1 2
/dev/hda2   none      swap    sw                 0 0
/dev/hda3   /         ext3    noatime            0 1

none        /proc     proc    defaults           0 0
none        /dev/shm  tmpfs   defaults           0 0

/dev/cdroms/cdrom0    /mnt/cdrom    auto      noauto,user    0 0
```

`auto` makes `mount` guess for the filesystem (recommended for removable media as they can be created with one of many filesystems) and `user` makes it possible for non-root users to mount the CD.

Now use the above example to create your `/etc/fstab`. If you are a SPARC-user, you should add the following line to your `/etc/fstab` too:

Code listing 6: Adding openprom filesystem to /etc/fstab

```
none        /proc/openprom  openpromfs    defaults      0 0
```

If you need `usbfs`, add the following line to `/etc/fstab`:

Code listing 7: Adding usbfs filesystem to /etc/fstab

```
none        /proc/bus/usb   usbfs         defaults      0 0
```

Reread your `/etc/fstab`, save and quit to continue.

8.b. Networking Information

Hostname, Domainname etc.

One of the choices the user has to make is name his PC. This seems to be quite easy, but lots of users are having difficulties finding the appropriate name for their Linux-pc. To speed things up, know that any name you choose can be changed afterwards. For all we care, you can just call your system `tux` and domain `homenetwork`.

We use these values in the next examples. First we set the hostname:

Code listing 8: Setting the hostname

```
# echo tux > /etc/hostname
```

Second we set the domainname:

Code listing 9: Setting the domainname

```
# echo homenetwork > /etc/dnsdomainname
```

If you have a NIS domain (if you don't know what that is, then you don't have one), you need to define that one too:

Code listing 10: Setting the NIS domainname

```
# echo nis.homenetwork > /etc/nisdomainname
```

Now add the `domainname` script to the default runlevel:

Code listing 11: Adding domainname to the default runlevel

```
# rc-update add domainname default
```

Configuring your Network

Before you get that "Hey, we've had that already"-feeling, you should remember that the networking you set up in the beginning of the gentoo installation was just for the installation. Right now you are going to configure networking for your Gentoo system permanently.

All networking information is gathered in `/etc/conf.d/net`. It uses a straightforward yet not intuitive syntax if you don't know how to setup networking manually. But don't fear, we'll explain everything :)

First open `/etc/conf.d/net` with your favorite editor (`nano` is used in this example):

Code listing 12: Opening /etc/conf.d/net for editing

```
# nano -w /etc/conf.d/net
```

The first variable you'll find is `iface_eth0`. It uses the following syntax:

Code listing 13: iface_eth0 syntaxis

```
iface_eth0="<your ip address> broadcast <your broadcast address> netmask <your netmask>"
```

If you use DHCP (automatic IP retrieval), you should just set `iface_eth0` to `dhcp`. If you use rp-pppoe (e.g. for ADSL), set it to `up`. If you need to setup your network manually and you're not familiar with all the above terms, please read the section on Understanding Network Terminology if you haven't done so already.

So let us give three examples; the first one uses DHCP, the second one a static IP (192.168.0.2) with netmask 255.255.255.0, broadcast 192.168.0.255 and gateway 192.168.0.1 while the third one just activates the interface for rp-pppoe usage:

Code listing 14: Examples for /etc/conf.d/net

```
(For DHCP)
iface_eth0="dhcp"

(For static IP)
iface_eth0="192.168.0.2 broadcast 192.168.0.255 netmask 255.255.255.0"
gateway="eth0/192.168.0.1"

(For rp-pppoe)
iface_eth0="up"
```

If you have several network interfaces, create extra `iface_eth` variables, like `iface_eth1`, `iface_eth2` etc. The `gateway` variable shouldn't be reproduced as you can only set one gateway per computer.

Now save the configuration and exit to continue.

Automatically Start Networking at Boot

To have your network interfaces activated at boot, you need to add those to the default runlevel. If you have PCMCIA interfaces you should skip this action as the PCMCIA interfaces are started by the PCMCIA init script.

Code listing 15: Adding net.eth0 to the default runlevel

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

If you have several network interfaces, you need to create the appropriate `net.eth1`, `net.eth2` etc. initscripts for those. You can use `ln` to do this:

Code listing 16: Creating extra initscripts

```
# cd /etc/init.d
# ln -s net.eth0 net.eth1
# rc-update add net.eth1 default
```

Writing Down Network Information

You now need to inform Linux about your network. This is defined in `/etc/hosts` and helps in resolving hostnames to IP addresses for hosts that aren't resolved by your nameserver. For instance, if your internal network consists of three PCs called `jenny` (192.168.0.5), `benny` (192.168.0.6) and `tux` (192.168.0.7 - this system) you would open `/etc/hosts` and fill in the values:

Code listing 17: Opening /etc/hosts

```
# nano -w /etc/hosts
```

Code listing 18: Filling in the networking information

```
127.0.0.1     localhost
192.168.0.5   jenny.homenetwork jenny
192.168.0.6   benny.homenetwork benny
192.168.0.7   tux.homenetwork tux
```

If your system is the only system (or the nameservers handle all name resolution) a single line is sufficient:

Code listing 19: /etc/hosts for lonely or fully integrated PCs

```
127.0.0.1     localhost   tux
```

Save and exit the editor to continue.

If you don't have PCMCIA, you can now continue with System Information. PCMCIA-users should read the following topic on PCMCIA.

Optional: Get PCMCIA Working

PCMCIA-users should first install the `pcmcia-cs` package:

Code listing 20: Installing pcmcia-cs

```
# emerge pcmcia-cs
```

When `pcmcia-cs` is installed, add `pcmcia` to the default runlevel:

Code listing 21: Adding pcmcia to the default runlevel

```
# rc-update add pcmcia default
```

8.c. System Information

Gentoo uses `/etc/rc.conf` for general, system-wide configuration. Open up `/etc/rc.conf` and enjoy all the comments in that file :)

Code listing 22: Opening /etc/rc.conf

```
# nano -w /etc/rc.conf
```

As you can see, this file is well commented to help you set up the necessary configuration variables. Take special care with the `KEYMAP` setting: if you select the wrong `KEYMAP` you will get weird results when typing on your keyboard.

Note: Users of USB-based SPARC systems and SPARC clones might need to select an i386 keymap (such as "us") instead of "sunkeymap".

When you're finished configuring `/etc/rc.conf`, save and exit, then continue with Configuring the Bootloader.

9. Configuring the Bootloader

Content:

- Making your Choice
- Default: Using GRUB
- Alternative: Using LILO
- Alternative: Using aboot
- Alternative: Using MILO
- Alternative: Using SILO
- Alternative: Using PALO
- Alternative: Using yaBoot
- Alternative: BootX
- Alternative: MIPS PROM

9.a. Making your Choice

Introduction

Now that your kernel is configured and compiled and the necessary system configuration files are filled in correctly, it is time to install a program that will fire up your kernel when you start the system. Such a program is called a bootloader. But before you start, consider your options...

Several bootloaders exist for Linux. However, these bootloaders only function on a small set of architectures. Therefore you must choose between the bootloaders that support your architecture.

The next table lists the architectures and the supported bootloaders. Pick a bootloader based on your architecture. For instance, if you have a Pentium IV, then your architecture is x86. You have the choice between GRUB (recommended) and LILO.

Architecture	Recommended	Alternatives
x86	GRUB	LILO
alpha	aBoot	MILO
sparc	SILO	
ppc	yaBoot (NewWorld), BootX (OldWorld)	
hppa	PALO	
mips	MIPS PROM	
amd64	GRUB	

Optional: Framebuffer

Note: This section only applies to x86, AMD64 and PPC users who have configured framebuffer support in their kernel. This includes `genkernel` users.

If you have configured your kernel with framebuffer support, you have to add a `vga`-statement to your bootloader configuration file if you require framebuffer. The next table lists the available `vga`-values you can use. In the example configuration files we use 800x600 @ 16bpp, thus 788.

	640x480	800x600	1024x768	1280x1024
8 bpp	769	771	773	775
16 bpp	785	788	791	794
32 bpp	786	789	792	795

Remember (or write down) your value; you will need it shortly hereafter.

Now select the bootloader of your choice from the table above.

9.b. Default: Using GRUB

Understanding GRUB's terminology

Important: GRUB can only be used with x86- and AMD64-based systems!

The most critical part of understanding GRUB is getting comfortable with how GRUB refers to hard drives and partitions. Your Linux partition `/dev/hda1` is called `(hd0,0)` under GRUB. Notice the parenthesis around the `hd0,0` - they are required.

Hard drives count from zero rather than "a" and partitions start at zero rather than one. Be aware too that with the hd devices, only hard drives are counted, not atapi-ide devices such as cdrom players and burners. Also, the same construct is used with scsi drives. (Normally they get higher numbers than ide drives except when the bios is configured to boot from scsi devices.)

Assuming you have a hard drive on `/dev/hda`, a cdrom player on `/dev/hdb`, a burner on `/dev/hdc`, a second hard drive on `/dev/hdd` and no SCSI hard drive, `/dev/hdd7` gets translated to `(hd1,6)`. It might sound tricky and tricky it is indeed, but as we will see, GRUB offers a tab completion mechanism that comes handy for those of you having a lot of hard drives and partitions and who are a little lost in the GRUB numbering scheme.

Having gotten the feel for that, it is time to install GRUB.

Installing GRUB

To install GRUB, let's first emerge it. Users of the x86 architecture have to install `grub`, AMD64 users will need to emerge `grub-static`:

Code listing 1: Installing GRUB

```
(For the x86 architecture)
# emerge grub

(For the AMD64 architecture)
# emerge grub-static
# cp -Rpv /usr/share/grub/i386-pc/* /boot/grub
```

To start configuring GRUB, you type in `grub`. You'll be presented with the `grub>` grub command-line prompt. Now, you need to type in the right commands to install the GRUB boot record onto your hard drive.

Code listing 2: Starting the GRUB shell

```
# grub
```

In the example configuration we want to install GRUB so that it reads its information from the boot-partition `/dev/hda1`, and installs the GRUB boot record on the hard drive's MBR (master boot record) so that the first thing we see when we turn on the computer is the GRUB prompt. Of course, if you haven't followed the example configuration during the installation, change the commands accordingly.

The tab completion mechanism of GRUB can be used from within GRUB. For instance, if you type in "`root (`" followed by a TAB, you will be presented with a list of devices (such as `hd0`). If you type in "`root (hd0,`" followed by a TAB, you will receive a list of available partitions to choose from (such as `hd0,0`).

By using the tab completion, setting up GRUB should be not that hard. Now go on, configure GRUB, shall we? :-)

Code listing 3: Installing GRUB in the MBR

```
grub> root (hd0,0)          (Specify where your /boot partition resides)
grub> setup (hd0)           (Install GRUB in the MBR)
grub> quit                  (Exit the GRUB shell)
```

Note: If you want to install GRUB in a certain partition instead of the MBR, you have to alter the `setup` command so it points to the right partition. For instance, if you want GRUB installed in `/dev/hda3`, then the command becomes `setup (hd0,2)`. Few users however want to do this.

Although GRUB is now installed, we still need to write up a configuration file for it, so that GRUB automatically boots your newly created kernel. Create `/boot/grub/grub.conf` with `nano` (or, if applicable, another editor):

Code listing 4: Creating /boot/grub/grub.conf

```
# nano -w /boot/grub/grub.conf
```

Now we are going to write up a `grub.conf`. Beneath you'll find two possible `grub.conf` for the partitioning example we use in this guide, with kernel image `kernel-2.4.25`. We've only extensively commented the first `grub.conf`.

- The first `grub.conf` is for people who have not used `genkernel` to build their kernel
- The second `grub.conf` is for people who have used `genkernel` to build their kernel

Code listing 5: grub.conf for non-genkernel users

```
# Which listing to boot as default. 0 is the first, 1 the second etc.
default 0
# How many seconds to wait before the default listing is booted.
timeout 30
# Nice, fat splash-image to spice things up :)
splashimage=(hd0,0)/grub/splash.xpm.gz

title=Gentoo Linux 2.4.25
# Partition where the kernel image (or operating system) is located
root (hd0,0)
kernel (hd0,0)/kernel-2.4.25 root=/dev/hda3

# The next three lines are only if you dualboot with a Windows system.
# In this case, Windows is hosted on /dev/hda6.
title=Windows XP
root (hd0,5)
chainloader +1
```

Code listing 6: grub.conf for genkernel users

```
default 0
timeout 30
splashimage=(hd0,0)/grub/splash.xpm.gz

title=Gentoo Linux 2.4.25
root (hd0,0)
kernel (hd0,0)/kernel-2.4.25 root=/dev/ram0 init=/linuxrc real_root=/dev/hda3
initrd (hd0,0)/initrd-2.4.25

# Only in case you want to dual-boot
title=Windows XP
root (hd0,5)
chainloader +1
```

Note: If you use a different partitioning scheme and/or kernel image, adjust accordingly. However, make sure that anything that follows a GRUB-device (such as `(hd0,0)`) is relative to the mountpoint, not the root. In other words, `(hd0,0)/grub/splash.xpm.gz` is in reality `/boot/grub/splash.xpm.gz` since `(hd0,0)` is `/boot`.

If you need to pass any additional options to the kernel, simply add them to the end of the kernel command. We're already passing one option (`root=/dev/hda3` or `real_root=/dev/hda3`), but you can pass others as well. As an example we use the `vga` statement for framebuffer we discussed previously:

Code listing 7: Adding the vga-statement as a kernel option

```
title=Gentoo Linux
  root (hd0,0)
  kernel (hd0,0)/kernel-2.4.25 root=/dev/hda3 vga=788
```

genkernel users should know that their kernels use the same boot options as is used for the LiveCD. For instance, if you have SCSI devices, you should add doscsi as kernel option.

Now save the grub.conf file and exit. As of now, GRUB is fully configured, and you can continue with Installing Necessary System Tools.

9.c. Alternative: Using LILO

Installing LILO

Important: LILO can only be used with x86-based systems!

LILO, the LInuxLOader, is the tried and true workhorse of Linux bootloaders. However, it lacks some features that GRUB has (which is also the reason why GRUB is currently gaining popularity). The reason why LILO is still used is that, on some systems, GRUB doesn't work and LILO does. Of course, it is also used because some people know LILO and want to stick with it. Either way, Gentoo supports both, and apparently you have chosen to use LILO.

Installing LILO is a breeze; just use emerge.

Code listing 8: Installing LILO

```
# emerge lilo
```

Configuring LILO

To configure LILO, you must create /etc/lilo.conf. Fire up your favorite editor (in this handbook we use nano for consistency) and create the file.

Code listing 9: Creating /etc/lilo.conf

```
# nano -w /etc/lilo.conf
```

Some sections ago we have asked you to remember the kernel-image name you have created. In the next example lilo.conf we assume the imagename is kernel-2.4.25. We also use the example partitioning scheme in this example. There are two separate parts:

- One for those who have not used genkernel to build their kernel
- One for those who have used genkernel to build their kernel

Code listing 10: Example /etc/lilo.conf

```
boot=/dev/hda             # Install LILO in the MBR
prompt                    # Give the user the chance to select another section
delay=50                  # Wait 5 (five) seconds before booting the default section
default=gentoo            # When the timeout has passed, boot the "gentoo" section
# Only if you use framebuffer. Otherwise remove the following line:
vga=788                   # Framebuffer setting. Adjust to your own will

# For non-genkernel users
image=/boot/kernel-2.4.25
  label=gentoo            # Name we give to this section
  read-only               # Start with a read-only root. Do not alter!
  root=/dev/hda3          # Location of the root filesystem

# For genkernel users
image=/boot/kernel-2.4.25
  label=gentoo
  read-only
  root=/dev/ram0
  append="init=/linuxrc real_root=/dev/hda3"
  initrd=/boot/initrd-2.4.25

# The next two lines are only if you dualboot with a Windows system.
# In this case, Windows is hosted on /dev/hda6.
other=/dev/hda6
  label=windows
```

Note: If you use a different partitioning scheme and/or kernel image, adjust accordingly.

If you need to pass any additional options to the kernel, add an `append` statement to the section. As an example, we add the `acpi=off` statement to disable ACPI support:

Code listing 11: Using append to add kernel options

```
image=/boot/kernel-2.4.25
  label=gentoo
  read-only
  root=/dev/hda3
  append="acpi=off"
```

`genkernel` users should know that their kernels use the same boot options as is used for the LiveCD. For instance, if you have SCSI devices, you should add `doscsi` as kernel option.

Now save the file and exit. To finish up, you have to run `/sbin/lilo` so LILO can apply the `/etc/lilo.conf` to your system (i.e. install itself on the disk).

Code listing 12: Finishing the LILO installation

```
# /sbin/lilo
```

Now continue with Installing Necessary System Tools.

9.d. Alternative: Using aboot

Important: aboot can only be used with alpha-based systems!

We first install aboot on our system. Of course we use `emerge` to do so:

Code listing 13: Installing aboot

```
# emerge aboot
```

The next step is to make our bootdisk bootable. This will start `aboot` when you boot your system. We make our bootdisk bootable by writing the `aboot` bootloader to the start of the disk.

Code listing 14: Making your bootdisk bootable

```
# swriteboot -f3 /dev/sda /boot/bootlx
# abootconf /dev/sda 2
```

Note: If you use a different partitioning scheme than the one we use throughout this chapter, you have to change the commands accordingly. Please read the appropriate manual pages (`man 8 swriteboot` and `man 8 abootconf`).

Additionally, you can make Gentoo boot more easily by setting up the aboot configuration file and SRM boot_osflags variable. You will need to make sure that the bootdef_dev is also properly set (easier to do at the console than from Linux).

Code listing 15: Improve booting Gentoo

```
# echo '0:2/boot/vmlinux.gz root=/dev/sda2' > /etc/aboot.conf
# echo -n 0 > /proc/srm_environment/named_variables/boot_osflags
# echo -n '' > /proc/srm_environment/named_variables/boot_file
```

If you're installing using a serial console, don't forget to include the serial console boot flag in `aboot.conf`. See `/etc/aboot.conf.example` for some further information. Additionally, you need to allow login on the serial port:

Code listing 16: Allowing login on the serial port

```
# echo tts/0 >> /etc/securetty
# echo 's0:12345:respawn:/sbin/agetty 9600 tts/0 vt100' >> /etc/inittab
```

Aboot is now configured and ready to use. Continue with Installing Necessary System Tools.

9.e. Alternative: Using MILO

Important: MILO can only be used with alpha-based systems!

Before continuing, you should decide on how to use MILO. In this section, we will assume that you want to make a MILO boot floppy. If you are going to boot from an MS-DOS partition on your hard disk, you should amend the commands appropriately.

To install MILO, we use `emerge`.

Code listing 17: Installing MILO

```
# emerge milo
```

After MILO has been installed, the MILO images should be in `/opt/milo`. The commands below make a bootfloppy for use with MILO. Remember to use the correct image for your Alpha-system.

Code listing 18: Installing MILO on a floppy

```
(First insert a blank floppy)
# fdformat /dev/fd0
# mformat a:
# mcopy /opt/milo/milo-2.2-18-gentoo-ruffian a:\milo
# mcopy /opt/milo/linload.exe a:\lilnload.exe
(Only if you have a Ruffian:
  # mcopy /opt/milo/ldmilo.exe a:\ldmilo.exe
)
# echo -ne '\125\252' | dd of=/dev/fd0 bs=1 seek=510 count=2
```

Your MILO boot floppy is now ready to boot Gentoo Linux. You may need to set environment variables in your ARCS Firmware to get MILO to start; this is all explained in the MILO-HOWTO with some examples on common systems, and examples of the commands to use in interactive mode.

Not reading the MILO-HOWTO is a bad idea.

Now continue with Installing Necessary System Tools.

9.f. Alternative: Using SILO

It is now time to install and configure SILO, the Sparc Improved boot LOader.

Code listing 19: Installing SILO

```
# emerge silo
```

Now open up your favorite editor (we use `nano` as an example) and create `/etc/silo.conf`.

Code listing 20: Creating /etc/silo.conf

```
# nano -w /etc/silo.conf
```

Beneath you find an example `silo.conf` file. It uses the partitioning scheme we use throughout this book and `kernel-2.4.25` as kernelimage.

Code listing 21: Example /etc/silo.conf

```
partition = 1         # Boot partition
root = /dev/hda4      # Root partition
timeout = 15          # Wait 15 seconds before booting the default section

image = /boot/kernel-2.4.25
  label = linux
```

If you use the example `silo.conf` delivered by Portage, be sure to comment out all lines that you do not need.

If you have a separate `/boot` partition, copy over the configuration file to `/boot` and run `/sbin/silo`:

Code listing 22: Only if you have /boot on a separate partition

```
# cp /etc/silo.conf /boot
# /sbin/silo -C /boot/silo.conf
/boot/silo.conf appears to be valid
```

If your `/boot` directory resides on your root partition, just run `/sbin/silo`:

Code listing 23: Run silo

```
# /sbin/silo
/etc/silo.conf appears to be valid
```

Now continue with Installing Necessary System Tools.

9.g. Alternative: Using PALO

Important: PALO can only be used on HPPA-based systems!

On the PA-RISC platform, the boot loader is called palo. You can find the configuration file in `/etc/palo.conf`. Here is a sample configuration:

Code listing 24: /etc/palo.conf example

```
--commandline=2/vmlinux root=/dev/sdb2 HOME=/
--recoverykernel=/vmlinux.old
--init-partitioned=/dev/sdb
```

The first line tells palo the location of the kernel and which boot parameters it must use. `2/vmlinux` means the kernel named `/vmlinux` resides on the second partition. Beware, the path to the kernel is relative to the partition, not to the root of your filesystem.

The second line indicates which recovery kernel to use. If it is your first install and you do not have a recovery kernel, please comment this out. The third line indicates on which disk palo will reside.

When configuration is done, just run `palo`.

Code listing 25: Applying the PALO configuration

```
# palo
```

Now continue with Installing Necessary System Tools.

9.h. Alternative: Using yaBoot

Introduction

Important: yaBoot can only be used on NewWorld PPC-based systems!

There are two ways to configure yaBoot for your system. You can use the new and improved `yabootconfig` included with `yaboot-1.3.8-r1` and later to automatically setup yaboot. If for some reason you do not want to run `yabootconfig` to automatically setup `/etc/yaboot.conf`, you can just edit the sample file already installed on your system.

- Default: Using yabootconfig
- Alternative: Manual yaBoot Configuration

Default: Using yabootconfig

`yabootconfig` will auto-detect the partitions on your machine and will setup dual and triple boot combinations with Linux, Mac OS, and Mac OS X.

To use `yabootconfig`, your drive must have a bootstrap partition, and `/etc/fstab` must be configured with your Linux partitions. Both of these should have been done already in the steps above. To start, ensure that you have the latest version of yaboot installed by running `emerge --update yaboot`. This is necessary as the latest version will be available via Portage, but it may not have made it into the stage files.

Code listing 26: Installing yaboot

```
# emerge --update yaboot
```

Now run `yabootconfig`. The program will run and it will confirm the location of the bootstrap partition. Type `Y` if it is correct. If not, double check `/etc/fstab`. yabootconfig will then scan your system setup, create `/etc/yaboot.conf` and run `mkofboot` for you. `mkofboot` is used to format the bootstrap partition, and install the yaboot configuration file into it.

You might want to verify the contents of `/etc/yaboot.conf`. If you make changes to `/etc/yaboot.conf` (like setting the default/boot OS), make sure to rerun `ybin -v` to apply changes to the bootstrap partition.

Now continue with Installing Necessary System Tools.

Alternative: Manual yaBoot Configuration

First make sure you have the latest `yaboot` installed on your system:

Code listing 27: Installing yaboot

```
# emerge --update yaboot
```

Below you find a completed `yaboot.conf` file. Alter it at will:

Code listing 28: /etc/yaboot.conf

```
## /etc/yaboot.conf
##
## run: "man yaboot.conf" for details. Do not make changes until you have!!
## see also: /usr/share/doc/yaboot/examples for example configurations.
##
## For a dual-boot menu, add one or more of:
## bsd=/dev/hdaX, macos=/dev/hdaY, macosx=/dev/hdaZ

## our bootstrap partition:

boot=/dev/hda9

##hd: is open firmware speak for hda
device=hd:
partition=11

root=/dev/hda11
delay=5
defaultos=macosx
timeout=30
install=/usr/lib/yaboot/yaboot
magicboot=/usr/lib/yaboot/ofboot

#################
## This section can be duplicated if you have more than one kernel or set of
## boot options
#################
image=/boot/vmlinux
  label=Linux
  sysmap=/boot/System.map
  read-only
##################

macos=/dev/hda13
macosx=/dev/hda12
enablecdboot
enableofboot
```

Once `yaboot.conf` is set up the way you want it, you run `mkofboot -v` to install the settings in the bootstrap partition. Don't forget this! If all goes well, and you have the same options as the sample above, your next reboot will give you a simple, five-entry boot menu. If you update your yaboot config later on, you'll just need to run `ybin -v` to update the bootstrap partition - `mkofboot` is for initial setup only.

For more information on yaboot, take a look at the yaboot project. For now, continue the installation with Installing Necessary System Tools.

9.i. Alternative: BootX

Important: BootX can only be used on OldWorld PPC-based systems!

If you want to use BootX, then you don't have to do anything at this stage. However, after rebooting, you will have to some configuration in the BootX control panel (inside MacOS). We discuss this later in the installation.

For now, continue with Installing Necessary System Tools.

9.j. Alternative: MIPS PROM

MIPS doesn't require that you install a bootloader. The MIPS PROM handles the booting, but you need to install your kernel as a viable option.

First, install `dvhtool`, needed to copy the kernel to the SGI Volume Header.

Code listing 29: Installing dvhtool

```
# emerge dvhtool
```

Now copy over the kernel to the SGI Volume Header. By default `dvhtool` assumes that the SGI Volume Header is on `/dev/sda`. If this is not the case (for instance when IRIX is installed on `/dev/sda` and Gentoo/MIPS on `/dev/sdb`) you need to inform `dvhtool` about it using `-d <device>`.

Code listing 30: Copying a kernel to the SGI Volume Header

```
# dvhtool --unix-to-vh <kernel name in /boot> <name you want to give in volume header>
```

If you want to see all available entries in the volume, use `--print-volume-directory`:

Code listing 31: Viewing all available entries in the volume

```
# dvhtool --print-volume-directory
```

To setup your system to boot Gentoo/MIPS you need to go tweak some settings in the MIPS PROM. We will describe this later on when the Gentoo installation has finished.

Now continue with Installing Necessary System Tools.

10. Installing Necessary System Tools

Content:

- System Logger
- Optional: Cron Daemon
- File System Tools
- Optional: Networking Tools

10.a. System Logger

When we mentioned what stage3 was, we said that it contained all necessary system tools for which we cannot provide a choice to our users. We also said that we would install the other tools later on. Well, here we are :)

The first tool you need to decide on has to provide logging facilities for your system. Unix and Linux have an excellent history of logging capabilities -- if you want you can log everything that happens on your system in logfiles. This happens through the system logger.

Gentoo offers several system loggers to choose from. There are `sysklogd`, which is the traditional set of system logging daemons, `msyslog`, a flexible system logger with a modularized design, `syslog-ng`, an advanced system logger, and `metalog` which is a highly-configurable system logger.

If you can't choose one, use `syslog-ng` as it is very powerful yet comes with a great default configuration.

To install the system logger of your choice, `emerge` it and have it added to the default runlevel using `rc-update`. The following example installs `syslog-ng`. Of course substitute with your system logger:

Code listing 1: Installing a system logger

```
# emerge syslog-ng
# rc-update add syslog-ng default
```

10.b. Optional: Cron Daemon

Next is the cron daemon. Although it is optional and not required for your system, it is wise to install one. But what is a cron daemon? A cron daemon executes scheduled commands. It is very handy if you need to execute some command regularly (for instance daily, weekly or monthly).

Gentoo offers three possible cron daemons: `dcron`, `fcron` and `vixie-cron`. Installing one of them is similar to installing a system logger. However, `dcron` and `fcron` require an extra configuration command, namely `crontab /etc/crontab`. If you don't know what to choose, use `vixie-cron`.

Code listing 2: Installing a cron daemon

```
# emerge vixie-cron
# rc-update add vixie-cron default
(Only if you have chosen dcron or fcron) # crontab /etc/crontab
```

10.c. File System Tools

Depending on what file systems you are using, you need to install the necessary file system utilities (for checking the filesystem integrity, creating additional file systems etc.).

The following table lists the tools you need to install if you use a certain file system:

File System	Tool	Install Command
XFS	xfsprogs	`emerge xfsprogs`
ReiserFS	reiserfsprogs	`emerge reiserfsprogs`
JFS	jfsutils	`emerge jfsutils`

If you don't need `rp-pppoe` to connect to the Internet, continue with Finalizing your Gentoo Installation. Otherwise continue with Optional: Networking Tools.

10.d. Optional: Networking Tools

If you need `rp-pppoe` to connect to the net, you need to install it.

Code listing 3: Installing rp-pppoe

```
# USE="-X" emerge rp-pppoe
```

The `USE="-X"` will prohibit XFree to be installed as a dependency (`rp-pppoe` has graphical tools; if you want those enabled, you can recompile `rp-pppoe` later on or have XFree installed now -- which takes a long time to compile).

Now continue with Finalizing your Gentoo Installation.

11. Finalizing your Gentoo Installation

Content:

- User Administration
- Reboot and Enjoy

11.a. User Administration

Setting a root Password

Before you forget, set the root password by typing:

Code listing 1: Setting the root password

```
# passwd
```

If you want root to be able to log on through the serial console, add `ttyS0` to `/etc/securetty`.

Code listing 2: Adding ttyS0 to /etc/securetty

```
# echo "ttyS0" >> /etc/securetty
```

Adding a User for Daily Use

Working as root on a Unix/Linux system is dangerous and should be avoided as much as possible. Therefore it is strongly recommended to add a user for day-to-day use.

For instance, to create a user called `john` who is member of the `wheel` group (be able to change to root using `su`), `users` group (default for all users) and `audio` group (be able to use audio devices):

Code listing 3: Adding a user for day-to-day use

```
# useradd john -m -G users,wheel,audio -s /bin/bash
# passwd john
Password: (Enter the password for john)
Re-enter password: (Re-enter the password to verify)
```

If this user ever needs to perform some task as root, he can use `su -` to temporarily receive root privileges. Another way is to use the `sudo` package which is, if correctly configured, very secure.

11.b. Reboot and Enjoy

Rebooting

Congratulations! Your Gentoo system is now ready. Exit the chrooted environment and unmount all mounted partitions and, in case you had to bind-mount `/mnt/gentoo/dev`, don't forget to unmount it too. Then type in that one magical command you have been waiting for: `reboot`.

Code listing 4: Rebooting the system

```
# exit
# cd /
# umount /mnt/gentoo/boot /mnt/gentoo/proc /mnt/gentoo
# reboot
```

Of course, don't forget to remove the bootable CD, otherwise the CD will be booted again instead of your new Gentoo system.

OldWorld PPC users will boot in MacOS since their bootloader isn't installed yet. Those users should read Optional: Configuring BootX. MIPS users will have to do some more tweaking in their MIPS PROM to get Gentoo to work. Those users should read Optional: Getting Gentoo/MIPS to Work.

GRP users can continue with Optional: Install GRP Packages, all the rest can finish up with Where to go from here?.

Optional: Configuring BootX

Important: This subsection is only for PPC-users who want to use BootX as bootloader. All other readers should skip this subsection.

Now your machine is booted in MacOS, open the BootX control panel. Select `Options`, and uncheck `Used specified RAM disk`. When you return to the BootX main screen, you will now find an option to specify your machine's root disk and partition. Fill these in with the appropriate values.

BootX can be configured to start Linux upon boot. If you do this, you will first see your machine boot into MacOS then, during startup, BootX will load and start Linux. See the BootX home page for more information.

If you are a GRP users you can continue with Optional: Install Extra Packages, otherwise go to Where to go from here?.

Optional: Getting Gentoo/MIPS to Work

When you are rebooted, go to the System Maintenance Menu and select Enter Command Monitor (`5`). If you want to test your new Gentoo installation, you can just run `boot -f <kernel name>`. To have your system permanently boot into the Gentoo installation, you need to set some variables in the MIPS PROM:

Code listing 5: Configuring the PROM to Boot Gentoo

```
1) Start System
2) Install System Software
3) Run Diagnostics
4) Recover System
5) Enter Command Monitor

Option? 5
Command Monitor.  Type "exit" to return to the menu.

(<root device> = Gentoo's root partition, e.g. /dev/sda3)
>> setenv OSLoadPartition <root device>

(To list the available kernels, type "ls")
>> setenv OSLoader <kernel name>
>> setenv OSLoadFilename <kernel name>

(Declare the kernel parameters you want to pass)
>> setenv OSLoadOptions <kernel parameters>

(Provide the location of the Volume Header)
>> setenv SystemPartition scsi(0)disk(1)rdisk(0)partition(8)

(Automatically boot Gentoo)
>> setenv AutoLoad Yes

(Set the timezone)
>> setenv TimeZone EST5EDT

(Use the serial console - graphic adapter users should have "g" instead of "d1" (one))
>> setenv console d1
```

Now you're ready to enjoy Gentoo!

Optional: Install GRP Packages

Important: This part is for GRP users only. Other users should skip this part and continue with Where to go from here?.

Now that your system is booted, log on as the user you created (for instance, `john`) and use `su -` to gain root privileges:

Code listing 6: Gaining root privileges

```
$ su -
Password: (Enter your root password)
```

Now we need to copy over the prebuilt binaries from the second CD (CD-2) if you have it. First mount this CD:

Code listing 7: Mount the CD-2

```
# mkdir /mnt/cdrom
(Put CD-2 in the CD tray)
# mount /mnt/cdrom
```

Now copy over all prebuilt binaries from the CD to `/usr/portage/packages`. Make sure you use the same copy-command!

Code listing 8: Copy over prebuilt binaries

```
# cp /mnt/cdrom/packages/All/* /usr/portage/packages/All/
```

Now install the packages you want. CD-2 contains several prebuilt binaries, for instance KDE:

Code listing 9: Installing KDE

```
# USE="bindist" emerge --usepkg kde
```

The `USE="bindist"` is needed when you install XFree (either directly or as a dependency). It prevents the downloading of Microsoft's core fonts (which we cannot distribute on our LiveCDs).

Be sure to install the binaries now. When you do an `emerge sync` to update Portage (as you will learn later), the prebuilt binaries might not match against the ebuilds in your updated Portage. You can try to circumvent this by using `emerge --usepkgonly` instead of `emerge --usepkg`.

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

12. Where to go from here?

Content:

- Documentation
- Gentoo Online

12.a. 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 definately 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 Desktop Configuration Guide.

For a full listing of all our available documentation check out our Documentation Resources page.

12.b. Gentoo Online

You are of course always welcome on our Gentoo Forums or on one of our many Gentoo IRC channels.

We also have several mailinglists 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 :)

2. Working with Gentoo

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

Content:

1. 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.
2. Portage and Software
   The main reason for portage is to maintain the software on your system. In this chapter you learn how to get information from a package, update your package database, install/remove/update software and more...
3. Portage Features
   Gentoo's Portage allows for several features that improve portage-related aspects, such as build time. This chapter explains the existing features.
4. Controlling Portage Behaviour
   You can tweak Portage to your own needs/environment. Learn how to protect files, select mirrors, change directory locations and more.
5. 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.
6. 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. USE flags

Content:

- What are USE-flags?
- Using USE-flags
- Package specific USE-flags

1.a. What are USE-flags?

The ideas behind 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 environment in an easy way. This forces the user into deciding what he really wants 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 or locally in `/usr/portage/profiles/use.desc`. A short (very incomplete) snippet:

Code listing 1: A short snippet of available USE-flags

```
gtk     - Adds support for x11-libs/gtk+ (The GIMP Toolkit)
gtk2    - Use gtk+-2.0.0 over gtk+-1.2 in cases where a program supports both.
gtkhtml - Adds support for gnome-extra/gtkhtml
guile   - Adds support for dev-util/guile (interpreter for Scheme)
icc     - Use the Intel C++ Compiler if the package supports it
icc-pgo - Enable PGO data generation or use when use icc.
imap    - Adds support for IMAP
```

1.b. Using USE-flags

Declare permanent 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 `/etc/make.profile/make.defaults` file. Let us take a look at this default setting:

Code listing 2: /etc/make.profile/make.defaults USE variable

```
USE="x86 oss apm arts avi berkdb crypt cups encode 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 slang
     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 the `/etc/make.profile/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 3: An example USE setting in /etc/make.conf

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

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.

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

Note: The `emerge` command will be discussed more thoroughly in Portage and Software.

Code listing 4: Using USE as evironment variable

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

Inheriting USE-flags

Some packages don't only listen to USE-flags, but also provide USE-flags. When you install such a package, the USE-flag they provide is added to your USE setting. To view the list of packages that provide a USE-flag, check `/etc/make.profile/use.defaults`:

Code listing 5: 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
```

Precendence

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

1. Default USE setting declared in `/etc/make.profile/make.defaults`
2. Inherited USE setting if a package from `/etc/make.profile/use.defaults` is installed
3. User-defined USE setting in `/etc/make.conf`
4. 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 6: Running emerge info

```
# emerge info
```

1.c. Package specific USE-flags

Viewing available USE-flags

In the next chapter on Portage and Software we will explain how to manage your installed software and how to work with `emerge`. However, we will give you a primer on `emerge` by showing you how to view what USE-flags a package uses.

Let us take the example of `mozilla`: what USE-flags does it listen to? To find out, we use `emerge` with the `--pretend` (don't really do anything) and `--verbose` (give more output) options:

Code listing 7: Viewing the used USE-flags

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

Calculating dependencies ...done!
[ebuild  N    ] net-www/mozilla-1.5-r1 +java +crypt -ipv6 -gtk2 +ssl +ldap
+gnome -debug +mozcalendar -mozaccess -mozxmlterm -moznoirc -moznomail
-moznocompose -moznoxft
```

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

Code listing 8: Installing gentoolkit

```
# emerge --usepkg gentoolkit
```

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

Code listing 9: Using etcat to view used USE-flags

```
# etcat 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)
```

2. Portage and Software

Content:

- Obtaining Package Information
- Updating Portage
- Maintaining Software
- Software Availability

2.a. Obtaining Package Information

The Lord of All Tools: emerge

The main Portage tool that most users will use is `emerge`. We have already used it during the Gentoo installation and in the previous chapter, but we just briefly explained how to use it. This chapter will elaborate on `emerge` and teach you how to use `emerge` to fix all your software-related needs.

`emerge` is the command used to install, remove, query and maintain software packages. It is a front-end for `ebuild`; people interested in becoming Gentoo professionals will learn how to use `ebuild` later on. For now, we will focus on `emerge` as it has functionality that `ebuild` lacks (such as resolving dependencies, searching the Portage tree, etc.).

Since `emerge` is the most important tool for Gentoo users, it has an extensive manpage you can read by issuing `man emerge`. You can also view the in-command help by running `emerge --help`.

Code listing 1: Retrieving help for emerge

```
# man emerge
# emerge --help
```

The Portage Tree

Before we continue describing `emerge`, let us first take a look at the Portage Tree. Go to `/usr/portage` and do a listing of the available directories. We use `ls --classify` to list the contents of a directory as it will show directories with a trailing "/".

Code listing 2: Viewing the Portage Tree

```
# cd /usr/portage; ls --classify
app-admin/       dev-ml/            gnome-libs/     net-print/
app-arch/        dev-perl/          gnome-office/   net-wireless/
app-benchmarks/  dev-php/           header.txt      net-www/
app-cdr/         dev-python/        incoming/       net-zope/
app-crypt/       dev-ruby/          jython/         packages/
app-dicts/       dev-tcltk/         kde-apps/       profiles/
app-doc/         dev-tex/           kde-base/       releases/
app-editors/     dev-util/          kde-i18n/       scripts/
app-emacs/       distfiles/         kde-libs/       sec-policy/
app-emulation/   eclass/            licenses/       skel.ChangeLog
app-games/       experimental/      media-fonts/    skel.ebuild
app-gnustep/     files/             media-gfx/      skel.metadata.xml
app-i18n/        fresco-base/       media-libs/     snapshots/
app-misc/        games-action/      media-plugins/  sys-apps/
app-office/      games-arcade/      media-radio/    sys-build/
app-pda/         games-board/       media-sound/    sys-cluster/
app-portage/     games-emulation/   media-tv/       sys-devel/
app-sci/         games-engines/     media-video/    sys-fs/
app-shells/      games-fps/         metadata/       sys-kernel/
app-text/        games-kids/        net-analyzer/   sys-kmods/
app-vim/         games-misc/        net-apache/     sys-libs/
app-xemacs/      games-mud/         net-dialup/     unix2tcp/
berlin-base/     games-puzzle/      net-dns/        x11-base/
dev-ada/         games-roguelike/   net-firewall/   x11-libs/
dev-cpp/         games-rpg/         net-fs/         x11-misc/
dev-db/          games-server/      net-ftp/        x11-plugins/
dev-dotnet/      games-simulation/  net-im/         x11-terms/
dev-embedded/    games-sports/      net-irc/        x11-themes/
dev-games/       games-strategy/    net-libs/       x11-wm/
dev-haskell/     games-util/        net-mail/       xfce-base/
dev-java/        glep/              net-misc/       xfce-extra/
dev-lang/        gnome-apps/        net-nds/
dev-libs/        gnome-base/        net-news/
dev-lisp/        gnome-extra/       net-p2p/
```

As you can see, the Portage tree has several subdirectories. Most of them are the categories in which the Gentoo packages, called ebuilds, reside. Take a look at, for instance, `app-office`:

Code listing 3: Viewing a category

```
# cd app-office; ls --classify
abiword/     gnotime/   kmymoney2/  ooodi/              plan/     timestamp.x
dia/         gnucash/   koffice/    oooqs/              qhacc/
dia2code/    gnumeric/  lxbank/     openoffice/         sc/
facturalux/  ical/      lyx/        openoffice-bin/     scribus/
gaby/        kbudget/   mdbtools/   openoffice-ximian/  siag/
gnofin/      khacc/     mrproject/  phprojekt/          texmacs/
```

Inside a category you will find the packages belonging to that category, with a separate directory for each package. Let us take a look at the `openoffice` package:

Code listing 4: Viewing a package

```
# cd openoffice; ls --classify
ChangeLog  files/        openoffice-1.0.3-r1.ebuild  openoffice-1.1.0-r2.ebuild
Manifest   metadata.xml  openoffice-1.1.0-r1.ebuild  openoffice-1.1.0.ebuild
```

Remember that we told you that a Gentoo package is called an ebuild? Well, in the example directory four of such ebuilds are stored. Their naming is almost identical: they only differ in the version name. You are free to view the contents of such a package: they are plain scripts. We will not discuss it right now as it isn't important to know if you plan on just using Gentoo.

The other files are the `ChangeLog` (which contains a listing of all the changes done to the ebuilds), `Manifest` (which contains the checksums and filesizes of all the files in the directory) and `metadata.xml` (which contains more information about the package, such as the responsible development group -- called herd -- and a more extensive description).

Inside the `files` directory you will find extra files, needed by Portage: digests (checksums and permissions of the files needed by a single version of the package), patches, example configuration files, etc.

Code listing 5: Viewing the extra files

```
# cd files; ls --classify
1.0.3/  digest-openoffice-1.0.3-r1  digest-openoffice-1.1.0-r1
1.1.0/  digest-openoffice-1.1.0     digest-openoffice-1.1.0-r2
# cd 1.1.0; ls --classify
fixed-gcc.patch      ooffice-wrapper-1.3
newstlportfix.patch  openoffice-1.1.0-linux-2.6-fix.patch
no-mozab.patch       openoffice-1.1.0-sparc64-fix.patch
nptl.patch
```

If you go back to the root of the Portage tree (`/usr/portage`) you will notice that there are other, non-category directories too. We will discuss those later in this chapter.

Search for a Package

If you are new to Linux or Gentoo, you might not know what tool you need for what job. To facilitate searching, `emerge` provides you with a way to search through the available packages on your system. There are two ways you can search through packages: by name, or by name and description.

To search through the Portage tree by name, use `emerge search`. For instance, to find out more about `mozilla`:

Code listing 6: Showing information about mozilla

```
# emerge search mozilla
Searching...
[ Results for search key : mozilla ]
[ Applications found : 5 ]
(Some output removed to improve readability)
*  net-www/mozilla
      Latest version available: 1.5-r1
      Latest version installed: 1.4-r3
      Size of downloaded files: 29,153 kB
      Homepage:    http://www.mozilla.org
      Description: The Mozilla Web Browser

*  net-www/mozilla-firebird
      Latest version available: 0.7
      Latest version installed: [ Not Installed ]
      Size of downloaded files: 37,850 kB
      Homepage:    http://www.mozilla.org/projects/firebird/
      Description: The Mozilla Firebird Web Browser
(...)
```

If you want to include a search through the descriptions too, use the `--searchdesc` argument:

Code listing 7: Search through the descriptions too

```
# emerge --searchdesc mozilla
Searching...
[ Results for search key : mozilla ]
[ Applications found : 10 ]
(Some output removed to improve readability)
*  dev-libs/nss-3.8
      Latest version available: 3.8
      Latest version installed: 3.8
      Size of downloaded files:  2,782 kB
      Homepage:    http://www.mozilla.org/projects/security/pki/nss/
      Description: Mozilla's Netscape Security Services Library that implements PKI support
```

As you can see, the output of `emerge` informs you about the category and name of the package, the available version, the currently installed version, the size of the downloaded files, the homepage and the small description.

You see something new? Yes, downloaded files. When you tell Portage to install a package, it of course needs to have the necessary sources (or precompiled packages) available. It therefore checks the contents of `/usr/portage/distfiles` (for source code) or `/usr/portage/packages/All` (for precompiled packages) to see if the necessary files are already available. If not, it downloads the necessary files and places them in those directories.

Viewing the ChangeLog

While browsing through the Portage Tree, you saw that there was a ChangeLog for each package. You can view the ChangeLog entries between the available version and the installed version with `emerge` too. Use the `--pretend --changelog` (`-pl` in short) options. As an example we will view the ChangeLog entries for `gnumeric`:

Code listing 8: Viewing the ChangeLog entries for gnumeric

```
# emerge --pretend --changelog gnumeric
(Some output removed to improve readability)
*gnumeric-1.2.2

  27 Nov 2003; foser <foser@gentoo.org> gnumeric-1.2.2.ebuild :
  New release, requested in #34492
  updated deps

  12 Nov 2003; Jason Wever <weeve@gentoo.org> gnumeric-1.2.0.ebuild:
  Marked stable on sparc, fixes bug #32405.

  14 Oct 2003; Jason Wever <weeve@gentoo.org> gnumeric-1.0.8.ebuild:
  Added ~sparc keyword.  Fixes bug #31150.
```

2.b. Updating Portage

Introduction

Searching through Portage is nice, but if you don't update your Portage Tree regularly, you will be stuck with the packages and versions available on your system. This means that your system will get outdated pretty soon and that you will be missing bugfixes and remedies for possible security problems.

There are several ways to update your Portage Tree. The most popular method is by using one of our rsync mirrors. Another one is by using a Portage snapshot (in case a firewall or unavailability of a network prohibits the use of the rsync server).

Selecting a Mirror for rsync

It is adviseable to first select a fast mirror close to you. You can do this manually (by setting the `SYNC` variable in `/etc/make.conf`) or use `mirrorselect` to do this for you automatically. As the `SYNC` variable will be discussed later on, we will focus on using `mirrorselect`. First install `mirrorselect` by emerging it:

Code listing 9: Installing mirrorselect

```
# emerge --usepkg mirrorselect
```

Now run `mirrorselect` to automatically select mirrors for you (it will also setup Portage to use a mirror for the source code):

Code listing 10: Running mirrorselect

```
# mirrorselect -a -s3
```

Updating Portage

To update Portage using rsync, simply run `emerge sync`:

Code listing 11: Updating Portage using emerge sync

```
# emerge sync
```

If this fails (due to network problems, or a firewall), you can try using `emerge-webrsync` which will download a Portage Tree snapshot using `wget`. This also means that you can use proxies if you want. We discussed how to setup your system to use proxies during the Gentoo installation.

Code listing 12: Updating Portage using emerge-webrsync

```
# emerge-webrsync
```

2.c. Maintaining Software

Building or Prebuilt?

Gentoo provides ebuilds, the Gentoo packages if you like. But when you want to install such an ebuild, you can choose between building the package and using a prebuilt package. But what are the advantages/disadvantages of both approaches, and can they be used next to each other?

As you probably have guessed, building packages takes a lot of time (especially if you have little resources or want to build big packages, such as KDE, OpenOffice.org, etc.). By building the package, you can use the `USE` setting to tweak the package to your system. Of course, you can also define high optimization options (in the `CFLAGS` and `CXXFLAGS` variables) to compile the package with.

Using prebuilt packages improves the installation time (as no more compilation is needed), but you will lose the advantages of the `USE` setting and the `CFLAGS` & `CXXFLAGS` variables.

As previously stated, prebuilt packages are stored in the `/usr/portage/packages/All` directory, while the source code of the packages is placed in `/usr/portage/distfiles`. If you have finished installing a package you can remove the package or source code from the respective directory. However, you might want to keep the package/source code of the latest version, just in case you want to reinstall the package (so you don't have to redownload it).

Installing Software from Sources

Okay, enough talking, let's cut to the chase. To install a package, you will use the `emerge` command. If you don't want to use any prebuilt packages, you can just use `emerge <package-name>` or `emerge <category>/<package-name>`. As an example we'll install `gnumeric`:

Code listing 13: Building gnumeric

```
# emerge gnumeric
```

This will download the source code for you and unpacks, compiles and installs the package on your system. It will also do the same for all the dependencies. If you want to see what dependencies will be installed with it, use the `--pretend` option (`-p` in short):

Code listing 14: Pretending to build gnumeric

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

If you want to download the source code of the package and its dependencies, but don't want to build the package, use the `--fetchonly` option (`-f` in short):

Code listing 15: Fetching sources for gnumeric

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

If you want to see where `emerge` downloads the sources from, combine the `--fetchonly` and `--pretend` options:

Code listing 16: Showing URLs of the sources for gnumeric

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

You can also opt to install a specific version of a package. For instance, if you want to install a gnumeric version older than 1.2 -- for any reason whatsoever :) you would type:

Code listing 17: Installing a specific gnumeric version

```
# emerge "<gnumeric-1.2"
```

Other possibilities are of course ">" (later version) and "=" (the exact version).

Installing Prebuilt Packages

When you want to install a prebuilt package, you should use the `--usepkg` option (`-k` in short). This will use the binary package available in `/usr/portage/packages/All` if the package and the version of the application you want to install match.

Code listing 18: Installing a prebuilt package for gnumeric

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

If you want to use the binary package, even if the versions don't match, use `--usepkgonly` (`-K` in short).

Code listing 19: Installing the prebuilt package for gnumeric

```
# emerge --usepkgonly gnumeric
```

If you don't have the prebuilt package on your system yet, you can have `emerge` download it from a mirror, defined in the `PORTAGE_BINHOST` variable declared in `/etc/make.conf`.

To download the binary package in case this package doesn't exist on your system already, use `--getbinpkg` (`-g` in short):

Code listing 20: Downloading and installing a prebuilt package for gnumeric

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

This will download the package and the package-related information for you and install it on your system, together with the dependencies. If you want to see what dependencies will be installed with it, use the `--pretend` option (`-p` in short):

Code listing 21: Pretending to download the prebuilt packages for gnumeric

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

You can also opt to download the prebuilt package (and the package-related information) without checking the information on your local system and without using the prebuilt package already on your system (if applicable), use the `--getbinpkgonly` option (`-G` in short):

Code listing 22: Installing a prebuilt package without using local information

```
# emerge --getbinpkgonly gnumeric
```

You can also opt to install a specific version of a package. For instance, if you want to install a gnumeric version older than 1.2 -- for any reason whatsoever :) you would type:

Code listing 23: Installing a specific gnumeric version

```
# emerge --usepkg "<gnumeric-1.2"
```

Other possibilities are of course ">" (later version) and "=" (the exact version).

Working with Dependencies

Portage has an extensive support for dependency handling. Although you usually don't need to even think about this (as dependencies are automatically handled by Portage) some users might want to know how you can work with `emerge` and dependencies.

For instance, if you want Portage to pretend that none of the dependencies of a package are installed, you can use `--emptytree` (`-e` in short). This is useful with `--pretend` to display a complete tree of dependencies for any particular package. Without `--pretend`, `emerge` will (re)compile all listed packages. However, `glibc` will not be listed as dependency for safety reasons.

Code listing 24: Show all dependencies of gnumeric

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

Another argument is `--nodeps`, which will ask Portage to try install the given package without taking care of the dependencies. It is trivial that this can lead to failures.

Code listing 25: Installing gnumeric without taking care of the dependencies

```
# emerge --nodeps gnumeric
```

The opposite of `--nodeps` is `--onlydeps`, which will have Portage install all dependencies of a given package, but not the package itself:

Code listing 26: Installing the dependencies of gnumeric

```
# emerge --onlydeps gnumeric
```

Updating your System

Portage knows two special tags to denote a set of software packages: system and world. You have already seen the former while installing Gentoo if you didn't use a stage3 installation. To refresh things: system is the collection of core packages, necessary to have a working Gentoo system.

The world tag consists of all software you have installed yourself on your system plus the system information. In other words, every time you emerge a package using `emerge <package-name>`, the `<package-name>` is registered in the world file (`/var/cache/edb/world`). Dependencies are not part of the world file, but we will get to that later.

If you want to update the system packages, use the `--update` option (`-u` in short):

Code listing 27: Updating the system packages

```
# emerge --update system
```

An identical approach can be used for the world packages:

Code listing 28: Updating your entire system

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

Again, if you want to see what `emerge` wants to update, use the `--pretend` option together with the `--update` option:

Code listing 29: Pretending to update your entire system

```
# emerge --pretend --update world
(Some output removed to improve readability)
[ebuild     U ] net-misc/wget-1.9-r1 [1.9]
[ebuild     U ] media-video/dvdauthor-0.5.0 [0.5.3]
[ebuild     U ] net-analyzer/ethereal-0.9.16 [0.9.14]
```

Right next to the word "ebuild" you will notice a letter (or combination of letters) which gives you more information about the package:

- B (blocks) The package listed to the left is blocking the emerge of the package listed to the right
- N (new) The package is new to your system and will be emerged for the first time
- R (reemerge) The package isn't new, but needs to be reemerged
- F (fetch) The package requires that you download the source code manually (for instance due to licencing issues)
- U (update) The package already exists on your system but will be upgraded
- UD (downgrade) The package already exists on your system but will be downgraded
- U- (slot warning) The package you have installed on your system is listed as a package that can not coexist with a different version, but your update does. The update will be installed and the older version will be removed.

In certain cases, an update may mean a downgrade (i.e. install an older version instead of a newer version). If you don't want this to happen, use the `--upgradeonly` option (`-U` in short):

Code listing 30: Upgrading your entire system

```
# emerge --update --upgradeonly world
```

Of course, we are talking here about system and world, but you can perform the same actions for individual software packages.

Removing Software

If you want to remove software from your system, you can use the `unmerge` option (`-C` - capital C - in short):

Code listing 31: Uninstalling software

```
# emerge unmerge gnumeric
```

If you want to test a removal (but not perform it), you can use `--pretend` again:

Code listing 32: Pretending to uninstall software

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

Warning: Portage doesn't verify if a package is a dependency for another installed package. It also doesn't warn you if the package is part of system, i.e. a core application necessary for the correct functioning of your system!

Once the unmerge begins you will see a long list of filenames belonging to the package. Some of these filenames will have a flag displayed to the left of the filename. The flags `!mtime`, `!empty`, and `cfgpro` specify reasons why certain files are not being removed while the package is. Files listed without any of these three flags are removed from the filesystem successfully. The three flags specify the following reasons:

- `!mtime` : The listed file has been changed since it was installed, probably by you or some tool
- `!empty` : The listed directory is not empty
- `cfgpro` : This file is located inside a protected directory and will not be touched for safety

2.d. Software Availability

ARCH or not?

Gentoo places its packages in two possible stadia called ARCH and ~ARCH. Don't take this literally: the stadia depend on the architecture you are using. In other words, for x86-based systems you have x86 and ~x86, for ppc-based systems you have ppc and ~ppc etc.

The ~ARCH stadium means that the package works for the developer in charge of the package, but that the package hasn't been tested thoroughly enough by the community to be placed in ARCH. ~ARCH packages usually go to ARCH after being bugfree for a sufficient amount of time.

Your system will use ARCH packages per default. If you want to live on the edge, don't mind having a broken package once in a while, know how to deal with a broken system and you like submitting bugreports to bugs.gentoo.org, then you can opt to use ~ARCH packages. To "move" your system to a ~ARCH-using system, edit the `ACCEPT_KEYWORDS` variable in `/etc/make.conf` so that it reads ~ARCH (again: for x86-based systems: ~x86, etc.).

Note though that it is far from trivial (if even impossible) to go back to ARCH from ~ARCH.

If you want to update your system now, you will notice that a lot of packages will be updated!

Masked Packages

When you want to install a package, you might come across the following message:

Code listing 33: Message about masked packages

```
Calculating dependencies
!!! all ebuilds that could satisfy <your package> have been masked.
```

A package can be masked due to two reasons:

1. The package is in ~ARCH while you use ARCH
2. The package is hard-masked explicitly

If the package is masked because of the first reason, and you really want to install it (knowing that there is a reason why it isn't available in ARCH), you can temporarily accept ~ARCH packages:

Code listing 34: Temporarily accepting ~ARCH packages

```
# ACCEPT_KEYWORDS="~x86" emerge gnumeric
```

A package is hardmasked if it is listed in `/usr/portage/profiles/package.mask`. If you read this file, you will also read the reason why the package is hardmasked (it is usually added as a comment). If you want to install the package nevertheless (despite all the possible warnings we could ever throw at your head about "breaking your system", "breaks other packages", or "badly needs testing"), create the `/etc/portage/package.unmask` file and list the package in it (use the same format as is used in `/usr/portage/profiles/package.mask`).

Do not alter the `/usr/portage/profiles/package.mask` file as all changes are undone the next time you update your Portage tree. If you want to hardmask a package create `/etc/portage/package.mask` and list the package in it (use the same format as mentioned above).

Blocked Packages

You have a situation when you receive the following error on your screen:

Code listing 35: Blocking package

```
[blocks B     ] gnome-base/bonobo-activation (from pkg gnome-base/libbonobo-2.4.0)
```

In the above example, the package `bonobo-activation` is blocking the emerge of `libbonobo`. To resolve this issue, remove the `bonobo-activation` package and continue:

Code listing 36: Resolving a blocking situation

```
# emerge unmerge bonobo-activation
```

3. Portage Features

Content:

- DistCC
- ccache
- Binary Packages
- Security Related Features
- Other Features

3.a. DistCC

What is DistCC?

`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 elaborate information about `distcc` (and information on how to have it work with Gentoo) in our Gentoo Distcc Documentation.

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` setting. However, if you don't use Gnome and would still like to have the monitor then you should put 'gtk' in your `USE` setting.

Installing distcc is, as is with all software available through Gentoo's Portage, extremely easy:

Note: From now on, as you now know how to install binary packages if you want, we will omit the `--usepkg` option throughout the rest of the Gentoo Handbook.

Code listing 1: Installing Distcc

```
# emerge distcc
```

Activating Portage Support

Well, if installation is easy, the rest should be easy too :) So let us quickly activate the Portage support for `distcc`.

First, open `/etc/make.conf` and edit the `FEATURES` variable so it contains the `distcc` keyword. Next, edit the `MAKEOPTS` variable so it reads `-jX` with `X` the number of CPUs that run `distccd` (including the current host) plus one:

Code listing 2: Possible MAKEOPTS setting in /etc/make.conf

```
# Suppose you have 2 single-CPU distccd hosts excluding this host:
MAKEOPTS="-j4"
```

Now, still inside `/etc/make.conf`, uncomment the `PORTAGE_TMPDIR` line and add the following line at the end of the file:

Code listing 3: Add an extra, distcc-specific variable to /etc/make.conf

```
# Don't forget to uncomment the PORTAGE_TMPDIR variable
DISTCC_DIR=${PORTAGE_TMPDIR}/portage/.distcc
```

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"
```

Of course, don't forget to run the `distccd` daemon too:

Code listing 5: Starting the distcc daemon

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

Congratulations, your system will now use distributed compiling! For more in-depth information about DistCC and Gentoo, please read our Gentoo DistCC Documentation.

3.b. ccache

What is ccache?

`ccache` is a fast compiler cache. When you compile a program, it will cache intermediate results so that, when you ever 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.

Installing ccache

Installing `ccache` with Gentoo is a breeze. Just emerge it and you're done :)

Code listing 6: Installing ccache

```
# emerge ccache
```

Activating Portage Support

First, edit `/etc/make.conf` and alter the `FEATURES` variable so that it contains the `ccache` keyword:

Code listing 7: Editing FEATURES in /etc/make.conf

```
FEATURES="ccache"
```

Next, edit (or create) the `CCACHE_SIZE` variable (also in `/etc/make.conf`) so it contains the amount of diskspace you want to sacrifice for `ccache`:

Code listing 8: Editing CCACHE_SIZE in /etc/make.conf

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

As of now, Portage will use `ccache` to speed up compilations where possible. If you are uncertain that `ccache` works, you can run `ccache -s` to view the `ccache` statistics:

Code listing 9: Viewing ccache statistics

```
# ccache -s
```

3.c. Binary Packages

Creating binary packages

We have already discussed how to work with prebuilt packages, but how do you create your own prebuilt packages?

If the package is already installed, you can use the `quickpkg` command which will make a tarball of the installed files. This is very interesting for backup purposes!

Code listing 10: Using quickpkg

```
# quickpkg gnumeric
```

If the package isn't installed yet, you can install it using `emerge` and ask to build a binary package too. `emerge` uses the `--buildpkg` option (`-b` in short) for this:

Code listing 11: Installing gnumeric and building binary packages too

```
# emerge --buildpkg gnumeric
```

If you want Portage to do this by default, you should set the `buildpkg` keyword in the `FEATURES` variable declared in `/etc/make.conf`.

Code listing 12: Automatically creating binary packages

```
FEATURES="buildpkg"
```

If you don't want to install the software, but only build the package, you can use the `--buildpkgonly` option (`-B` in short):

Code listing 13: Building a binary package for gnumeric

```
# emerge --buildpkgonly gnumeric
```

3.d. Security Related Features

Sandbox

While building and installing packages, Portage uses a sandbox to protect your live system. This means that, as long as the package isn't installed on your system, the package cannot touch any file outside the sandbox. This ensures that Portage knows what files are created and modified by a package.

When the package compilation is finished, Portage will "preinstall" the package in the sandbox, registering what files are placed and where. It will then move those files from the sandbox on your live system.

User Privileges

Portage also supports building packages as non-root user (more precisely, as user "portage", group "portage"). This improves the security during the build process. You can opt to use user privileges with or without sandboxing. Of course, it goes without saying that user privileges and sandboxing is the most preferred method :)

Activating sandbox and/or userpriv

Portage will use `sandbox` per default. If you want `userpriv`, you should add it to the `FEATURES` variable. Note that activating `userpriv` will drop `sandbox` support, unless you also activate `usersandbox`:

Code listing 14: Activating userpriv and usersandbox

```
FEATURES="userpriv usersandbox"
```

Warning: Do not remove `sandbox` from the `FEATURES` variable!

Strict Checking

Portage can be asked to react strongly to possibly dangerous conditions (such as missing or incorrect Manifest files). To activate this strict checking, add the `strict` keyword to the `FEATURES` variable:

Code listing 15: Activating strict checking

```
FEATURES="strict"
```

Smart File System Permissions

Portage can be told to automatically deal with potentially dangerous file permissions that could pose a security risk. It does this by removing the "group" and "other" readable bits on setuid files and removing the "other" readable bit on setgid files in the pre install phase. To activate the smart file permissions, add the `sfperms` keyword to the `FEATURES` variable:

Code listing 16: Activating smart file system permissions

```
FEATURES="sfperms"
```

3.e. Other Features

Portage Help

There are several other keywords you can place in the `FEATURES` variable. Most of them are targeted towards developers and less interesting for the casual user. If you are interested in learning more about these features (or Portage generally), don't forget to read the `make.conf` manpage we provide.

Code listing 17: More Portage-related information

```
# man make.conf
```

4. Controlling Portage Behaviour

Content:

- Configuration File Protection
- Networking Options
- Directory Locations
- Other Portage Options
- Resources

4.a. Configuration File Protection

Protecting?

Portage knows the concept of "protected files". This means that, when you update software, it will not immediately overwrite certain files with newer versions, but inform you that a newer version exists. This is of course very usefull for configuration files (the files that reside in `/etc`).

Instead of overwriting such files, it will create a new file called `._cfg0000_<name>` with `<name>` being the original file name. It is then up to the user to merge the necessary differences in the existing file. He can use the `etc-update` command to ease this operation. We will talk about `etc-update` later.

Declaring CONFIG_PROTECT

Portage cannot protect on a file per file basis. Instead it protects entire directories. The `CONFIG_PROTECT` variable lists all protected directories. All subdirectories of the listed directories are protected too. The `CONFIG_PROTECT` variable is defined in `/etc/make.globals`, but if you want to change it, you should declare it in `/etc/make.conf` (to keep things consistent `/etc/make.conf` is used for all Portage configuration).

Code listing 1: An example CONFIG_PROTECT setting

```
CONFIG_PROTECT="/etc /usr/share/config /usr/kde/3.1/share/config"
```

If you want a certain directory protected, but not all of its subdirectories, you can "unprotect" these directories by listing them in the `CONFIG_PROTECT_MASK` variable, which also has a default value defined in `/etc/make.globals` but should be altered by declaring it in `/etc/make.conf`:

Code listing 2: An example CONFIG_PROTECT_MASK setting

```
CONFIG_PROTECT_MASK="/etc/init.d"
```

More information about the Configuration File Protection can be found in `emerge`'s online help:

Code listing 3: Getting information about Configuration File Protection

```
# emerge --help config
```

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.

Running `etc-update` is pretty straight-forward:

Code listing 4: Running etc-update

```
# etc-update
```

After merging the trivial 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 5: 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 without performing any 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 as easy as entering the number listed to the left of that configuration file.

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

Code listing 6: 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 usefull information, enter `2`. If you want to interactively update your current configuration file, enter `3`.

It is not usefull to elaborate about 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 7: 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.

4.b. Networking Options

Mirrors

As Gentoo is becoming increasingly popular, the use of mirrors is greatly appreciated. Portage uses three variables for the mirrors: one for the rsync mirrors (which is used to synchronise your Portage Tree with), one for the distfiles (which is used to download the source code) and one for the prebuilt packages.

All possible distfiles mirrors are listed on our Gentoo mirrors page. You can also use `mirrorselect` which will ease the setup of mirrors for your system. But let us first check out the individual variables...

The SYNC variable contains a list of rsync-mirrors you wish to use. For instance, to use rsync://rsync.namerica.gentoo.org/gentoo-portage as first choice, and rsync://rsync.samerica.gentoo.org/gentoo-portage as second, you would define this in /etc/make.conf:

Code listing 8: Defining SYNC in /etc/make.conf

```
SYNC="rsync://rsync.namerica.gentoo.org/gentoo-portage
      rsync://rsync.samerica.gentoo.org/gentoo-portage"
```

The GENTOO_MIRRORS variable contains a list of distfiles mirrors you wish to use. For instance, to use ftp://ibiblio.org/pub/Linux/distributions/gentoo as first choice, and http://www.gtlib.cc.gatech.edu/pub/gentoo as second, you would define this in /etc/make.conf:

Code listing 9: Defining GENTOO_MIRRORS in /etc/make.conf

```
GENTOO_MIRRORS="ftp://ibiblio.org/pub/Linux/distributions/gentoo
                http://www.gtlib.cc.gatech.edu/pub/gentoo"
```

The PORTAGE_BINHOST variable contains a list of prebuilt package mirrors you wish to use. For instance, to use ftp://login:pass@grp.mirror.site/pub/grp/i686/athlon-xp, you would define this in /etc/make.conf:

Code listing 10: Defining PORTAGE_BINHOST in /etc/make.conf

```
PORTAGE_BINHOST="ftp://login:pass@grp.mirror.site/pub/grp/i686/athlon-xp"
```

Mirrorselect

If you want to use mirrorselect, first install it (if you haven't done so already).

Code listing 11: Installing mirrorselect

```
# emerge mirrorselect
```

You can now opt to have mirrorselect automatically select the best mirrors for you, or select the mirrors manually from a list. For more information on how to use mirrorselect, just run mirrorselect from the command line - it will give you a quick overview on mirrorselect.

Code listing 12: Running mirrorselect

```
# mirrorselect
```

Fetching

The program which Portage uses to download archive files can be specified by setting the FETCHCOMMAND and RESUMECOMMAND settings. Several examples are shown in /etc/make.conf and /etc/make.globals. Portage uses wget by default:

Code listing 13: Default FETCHCOMMAND & RESUMECOMMAND

```
FETCHCOMMAND="/usr/bin/wget -t 5 --passive-ftp -P \${DISTDIR} \${URI}"
RESUMECOMMAND="/usr/bin/wget -c -t 5 --passive-ftp -P \${DISTDIR} \${URI}"
```

The ${DISTDIR} variable is substituted with the location where downloaded files are saved (/usr/portage/distfiles), while the ${URI} variable is substituted with the file that Portage needs to download.

As Portage uses wget by default, you can configure it to use proxies by defining http_proxy and ftp_proxy (note the small caps). Although you can do this in /etc/make.conf too, you are advised to use a more general approach as the http_proxy and ftp_proxy variables are used by other tools too (/etc/make.conf is used by Portage only). Please read the chapter on Environment Variables on how to declare system-wide environment variables.

Configuring rsync

rsync is used by emerge sync to update your Portage tree. Three variables used by Portage to change rsync's behaviour are RSYNC_EXCLUDEFROM, RSYNC_RETRIES and RSYNC_TIMEOUT.

A way to "protect" ebuilds from being updated or removed by emerge sync is to use the RSYNC_EXCLUDEFROM variable. It should be set to a file that is used by rsync to exclude certain files and defaults to /etc/portage/rsync_excludes. It is not adviseable to use this method as it can break dependencies if you are not careful. We will talk about PORTDIR_OVERLAY later on, which is the recommended method. For more information, please read the rsync manpage.

Code listing 14: The RSYNC_EXCLUDEFROM variable

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

When rsync fails, it will retry a number of times before switching to the next available rsync server. The number of retries is defined in RSYNC_RETRIES and defaults to 3:

Code listing 15: The RSYNC_RETRIES variable

```
RSYNC_RETRIES="3"
```

If you are using a (very) slow rsync server, rsync can time-out if no traffic is received anymore. The amount of seconds to wait before time-out is defined in the RSYNC_TIMEOUT variable and defaults to 180:

Code listing 16: The RSYNC_TIMEOUT variable

```
RSYNC_TIMEOUT="180"
```

4.c. Directory Locations

Introduction

Everything about Portage is configurable, including the directories used for the various tasks and files needed by Portage. To change the default locations (as defined in /etc/make.globals) you need to define the correct variables in - where else :) - /etc/make.conf.

Warning: If you alter a variable to point to a different location, never end the path with a trailing / !

Portage Tree

The location of the Portage tree is defined in the PORTDIR variable. It defaults to /usr/portage:

Code listing 17: The PORTDIR variable

```
PORTDIR="/usr/portage"
```

If you want to keep a local Portage tree next to the "official" one, you need to define the PORTDIR_OVERLAY variable. The directory location(s) listed in this value are unaffected by emerge sync actions: ebuilds in those locations will not get updated or removed, but are part of your Portage tree.

Code listing 18: The PORTDIR_OVERLAY variable

```
PORTDIR_OVERLAY="/usr/local/portage"
```

Distfiles

The location of the downloaded source code (so called distfiles) is defined in the `DISTDIR` variable. It defaults to `${PORTDIR}/distfiles`:

Code listing 19: The DISTDIR variable

```
DISTDIR="${PORTDIR}/distfiles"
```

Packages and RPMs

The location of the prebuilt packages is defined in the `PKGDIR` variable. It defaults to `${PORTDIR}/packages`:

Code listing 20: The PKGDIR variable

```
PKGDIR="${PORTDIR}/packages"
```

The location of the RPMs (yes, some packages are available as RPMs) is defined in the `RPMDIR` variable. It defaults to `${PORTDIR}/rpm`:

Code listing 21: The RPMDIR variable

```
RPMDIR="${PORTDIR}/rpm"
```

Temporary Portage Files

Portage uses a temporary location to build its ebuilds in. This location is defined in the `PORTAGE_TMPDIR` variable. It defaults to `/var/tmp`:

Code listing 22: The PORTAGE_TMPDIR variable

```
PORTAGE_TMPDIR="/var/tmp"
```

Per default, Portage will create a `portage` directory inside `PORTAGE_TMPDIR`. This is declared in the `BUILD_PREFIX` variable:

Code listing 23: The BUILD_PREFIX variable

```
BUILD_PREFIX="${PORTAGE_TMPDIR}/portage"
```

If you intend to change the location, make sure this temporary directory is on a partition with a sufficient amount of free space: when compiling big software packages, the directory can grow to 2 Gb and beyond!

Logging

The `PORT_LOGDIR` is a special variable and unset by default. When you define it, Portage will create per-ebuild logs in the given directory:

Code listing 24: The PORT_LOGDIR variable

```
PORT_LOGDIR="/var/log/portage"
```

4.d. Other Portage Options

Nice Value

Portage supports building with a different nice-value (a priority-like value). If you want to have Portage build packages with a higher nice-value (resulting in a more responsive system during the building process, but which also increases the build time) you can define the `PORTAGE_NICENESS` variable with a positive number:

Code listing 25: The PORTAGE_NICENESS variable

```
PORTAGE_NICENESS="3"
```

SLOT'ed Packages and Automatic Cleaning

In several situations you want multiple different versions of a package (including libraries) to be available on your system. Portage supports this by defining the `SLOT` variable in the ebuilds. As a user, you don't have to know how `SLOT`'ing works, but it is important you know it is supported.

If you are installing a newer version of a package, Portage will check if the `SLOT` variable is declared for that package. If that is the case, and the `SLOT` variable is different for both packages (new and old), Portage will not touch the older package.

However, if the `SLOT` variable is the same (as is usually the case), the older package will be removed by default. In order for the user to interrupt this removal, Portage will count down a certain amount of seconds. This amount is defined in the `CLEAN_DELAY` variable and defaults to `5` seconds:

Code listing 26: The CLEAN_DELAY variable

```
CLEAN_DELAY="5"
```

If you don't want Portage to automatically remove the older versions (called "cleaning") you can set the `AUTOCLEAN` variable to `no`:

Code listing 27: The AUTOCLEAN variable

```
AUTOCLEAN="no"
```

Build Related Variables

We have already encountered quite a lot of variables, but we are not done yet. People who have installed Gentoo will know about the `CHOST`, `CFLAGS` and `CXXFLAGS` variables, used by the compiler to compile and optimize the packages.

More information about these variables can be found in the `gcc` info pages, or online in the GCC Online Manuals.

Code listing 28: Getting information on CHOST, CFLAGS and CXXFLAGS

```
# info gcc
(Select "Invoking gcc")
(Select "Optimize options")
```

If the `DEBUGBUILD` is defined, Portage will not strip the binaries and libraries to make debugging more easy. This slows down your system and increases the filesizes.

Code listing 29: The DEBUGBUILD variable

```
# Do not set this to "false"; instead remove the line. Portage does not
# check the value, it just checks if the variable is defined.
DEBUGBUILD="true"
```

The `MAKEOPTS` variable is used by `make`, a tool used to ease the compilation of a package. It is usually defined to tell `make` to run several compilations simultaneously (especially if you have a multi-CPU system, or are using `distcc` as described previously).

To have `make` run three compilations simultaneously, set the `MAKEOPTS` variable to `-j3`:

Code listing 30: The MAKEOPTS variable

```
MAKEOPTS="-j3"
```

The `ROOT` variable shouldn't be set in `/etc/make.conf`, but rather as environment variable. Portage will check this variable to see where a package needs to be installed. Of course, this defaults to `/`. As an example we show you how to install `gnumeric` in `/mnt/gentoo` instead of in your running system:

Code listing 31: The ROOT variable

```
# ROOT="/mnt/gentoo" emerge gnumeric
```

Output Formatting

By default, Portage colors its output to improve readability. If you do not want this, set the `NOCOLOR` variable to `true`:

Code listing 32: The NOCOLOR variable

```
NOCOLOR="true"
```

4.e. Resources

Man Pages

If you need a quick reference on all listed variables, please consult the `make.conf` man page:

Code listing 33: Consulting the make.conf man page

```
# man make.conf
```

Commented Examples

The `/etc/make.conf` file contains lots of comments, including examples you might find interesting. However, most people don't update their `/etc/make.conf` interactively and therefore miss updates to the files. You can find the latest `/etc/make.conf` file in our Online ViewCVS Repository.

5. Initscripts

Content:

- Runlevels
- Working with rc-update
- Configuring Services
- Writing Init Scripts

5.a. Runlevels

Booting your System

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 beneith `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 explained to you, 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 but that are missing:

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

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

5.b. Working with rc-update

What is 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 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
```

5.c. Configuring Services

Why the Need for Extra Configuration?

Init scripts can be quite complex. It is therefore not really interesting to have the users directly edit the init script, 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 being afraid that your configuration changes are 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).

5.d. Writing Init Scripts

Do I Have To?

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 ran 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 adviseable.

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 adviseable 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`.

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.

6. Environment Variables

Content:

- Environment Variables?
- Defining Variables Globally
- Defining Variables Locally

6.a. Environment Variables?

What they are

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. This is however wrong: 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
```

6.b. Defining Variables Globally

The /etc/env.d Directory

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/conf.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 doesn't only benefit 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 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 wrong: 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. This is why many of the files in `/etc/env.d` begin with a number.

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"
```

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
```

6.c. Defining Variables Locally

User Specific

You do not always want to define an environment variable globally. For instance, you might want to add `/home/my_user/bin` 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

```
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 those few moments 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"
```