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

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. This chapter explains how to install Gentoo using one of our LiveCDs

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 so-called 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 for the x86 architecture. Each one of them has its own way of configuration. We 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?

- **Working with Gentoo**
  Learn how to work with Gentoo: installing software, altering variables, changing portage behaviour etc.
  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.

A. Installing Gentoo

1. About the Gentoo Linux Installation

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 from its users what is beneath the hood. Portage, the package maintenance system which Gentoo uses, is written in Python, meaning you can easily view and modify the source code. Gentoo's packaging system uses source code (although support for precompiled packages is included too) and configuring Gentoo happens through regular 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 ready to chroot 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, from a rescue floppy, etc. This document covers the installation using one of our LiveCDs. For help on the other installation approaches, please read our Alternative Installation Guide.

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 check the release errata, 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.

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

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

If you have a question regarding Gentoo, check out our Frequently Asked Questions, 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**

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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1 (5 of 100)9-4-2004 0:57:08) profile
- The **amd64** architecture (amd64)
- The **sparc** architecture (sparc64)
- The **ppc** architecture (ppc, G3, G4)

The [Gentoo Hardened](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1 (5 of 100)9-4-2004 0:57:08) 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

2.a. Hardware Requirements

**Introduction**

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

**The x86 architecture**
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)
You need a 486+ processor and at least 64 megabytes of memory

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's Minimal LiveCD
- Gentoo's Universal LiveCD

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 |
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.

<table>
<thead>
<tr>
<th>Stage3 Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Fastest way to get a Gentoo base system</td>
</tr>
<tr>
<td>- You cannot tweak the base system - it's built already</td>
</tr>
<tr>
<td>- You cannot brag about having used stage1 or stage2</td>
</tr>
</tbody>
</table>

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**

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.

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 so you are able to install Gentoo without a working network configuration.

Now what do these LiveCDs contain?

**Gentoo's Minimal LiveCD**

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. For example the x86 variant of this LiveCD can be found in the universal subdirectory and is called install-x86-minimal-2004.0.iso.

<table>
<thead>
<tr>
<th>Minimal LiveCD Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Smallest download</td>
</tr>
<tr>
<td>+ Suitable for a complete architecture</td>
</tr>
<tr>
<td>+ You can do a stage1, stage2 or stage3 by getting the stage tarball off the net</td>
</tr>
<tr>
<td>- Contains no stages, no portage snapshot, no GRP packages and therefore not suitable for networkless installation</td>
</tr>
</tbody>
</table>

**Gentoo's Universal LiveCD**
Gentoo's Universal LiveCD is a bootable CD suitable to install Gentoo without networking. It contains a stage1 and several stage3 tarballs (optimized for the individual subarchitectures). For example the x86 variant of this CD is called `install-x86-universal-2004.0.iso` and can be found in the `universal` subdirectory.

If you take a closer look into our `livecd` directory you will see that some architectures provide Gentoo Package CDs. This CD (which isn't bootable) only contains precompiled packages and can be used to install software after a succesfull Gentoo Installation. To install Gentoo, you only need the Universal LiveCD, but if you want OpenOffice.org, Mozilla, KDE, GNOME etc. without having to compile every single one of them, you need the Packages CD too. For example the Athlon-XP (a subarchitecture of x86) Packages CD is called `packages-athlon-xp-2004.0.iso` and can be found in the appropriate subdirectory (`athlon-xp`).

<table>
<thead>
<tr>
<th>Universal LiveCD with Packages CD</th>
<th>Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Packages CD is optimized to your subarchitecture</td>
</tr>
<tr>
<td>+</td>
<td>Packages CD provides precompiled packages for fast Gentoo installations</td>
</tr>
<tr>
<td>+</td>
<td>Contains everything you need. You can even install without a network connection.</td>
</tr>
<tr>
<td>-</td>
<td>Huge download</td>
</tr>
</tbody>
</table>

### 2.c. Download, Burn and Boot a Gentoo LiveCD

**Downloading and Burning the LiveCDs**

You have chosen to use a Gentoo LiveCD. We'll first start by downloading and burning the chosen LiveCD. We previously discussed the several available LiveCDs, but where can you find them?

You can download any of the LiveCDs (and, if you want, also a Packages CD) from one of our [mirrors](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1 (8 of 100)9-4-2004 0:57:08). The LiveCDs and Packages CDs are located in the `releases/x86/2004.0/livecd` directory.

**Warning:** The Packages CD is not a LiveCD, even though it is situated in the `livecd/` directory. You cannot boot from a Packages CD.

Inside that directory you'll find so-called ISO-files. Those are full CD images which you can write on a CD-R.

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 `install-x86-minimal-2004.0.iso.md5`). You can check the MD5 checksum with the `md5sum` tool under Linux/Unix or [md5summer](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1 (8 of 100)9-4-2004 0:57:08) for Windows.

**Note:** Please check the descriptions on the [Gentoo Store](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1 (8 of 100)9-4-2004 0:57:08) 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 :)

Booting the LiveCD(s)

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

Now place the 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:

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gentoo</td>
<td>Default kernel with framebuffer support</td>
</tr>
<tr>
<td>smp</td>
<td>Kernel with support for multiple CPUs</td>
</tr>
<tr>
<td>gentoo-nofb</td>
<td>Same as gentoo but without framebuffer support</td>
</tr>
<tr>
<td>smp-nofb</td>
<td>Same as smp but without framebuffer support</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Kernel Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acpi</td>
<td>Activate ACPI support</td>
</tr>
<tr>
<td>doataraid</td>
<td>Activate support for ATA RAID devices</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>dofirewire</td>
<td>Activate support for FireWire devices</td>
</tr>
<tr>
<td>dopcmcia</td>
<td>Activate PCMCIA support</td>
</tr>
<tr>
<td>doscsi</td>
<td>Activate support for SCSI devices</td>
</tr>
<tr>
<td>noapm</td>
<td>Deactivate APM support</td>
</tr>
<tr>
<td>nodetect</td>
<td>Deactivate hardware detection (kudzu/hotplug)</td>
</tr>
<tr>
<td>noddhcp</td>
<td>Do not use DHCP to query for an IP address</td>
</tr>
<tr>
<td>noevms</td>
<td>Deactivate EVMS support</td>
</tr>
<tr>
<td>nohotplug</td>
<td>Deactivate hotplug (kernel loading program)</td>
</tr>
<tr>
<td>nousb</td>
<td>Deactivate USB support</td>
</tr>
<tr>
<td>ide=nodma</td>
<td>Deactivate DMA support</td>
</tr>
</tbody>
</table>

**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 as kernel parameters:

**Code Listing 1: Booting a LiveCD**

```bash
boot: gentoo dopcmcia
```

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.

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 2: Listing available keymaps**

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

Now load the keymap of your choice:

**Code Listing 3: Loading a keymap**

```bash
# 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 (the SPARC LiveCDs don't even do autodetection), 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):

```
# 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):

```
# 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):

```
# hdparm -d 1 /dev/hda
# 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:

```
# 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".

```
# 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 9: 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 10: Starting the SSH daemon**

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

To be able to use sshd, you first need to setup your networking. Continue with the chapter on [Configuring your Network](#).

### 3. Configuring your Network

#### 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 Gentoo Universal LiveCDs.

**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 a Gentoo Universal LiveCD. This LiveCD includes 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 Universal LiveCD,
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**

```
# export http_proxy="http://proxy.gentoo.org:8080"
# export ftp_proxy="ftp://proxy.gentoo.org:8080"
# 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"
```

**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 configuration**

```
```
/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

```bash
# 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-set up (for regular networks) or adsl-set up (for ADSL-users).

If your installation medium does not contain any of these tools or your network doesn't function yet, continue with Manual Network Configuration.

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

Default: Using net-setup

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

Code Listing 6: Running the net-setup script

```bash
# net-set up eth0
```

net-set up 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.

### 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 it is possible that your networkcard wasn't found immediately. This means you may have to load the appropriate network modules manually.

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

```
Code Listing 8: 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 9: 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 10: Testing availability of your network card, successful
```
# ifconfig eth0
eth0 Link encapsulation: 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 11: 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.

Select one of the following sections based on your network setup:

- [Using DHCP](#) for automatic IP retrieval
- [Preparing for Wireless Access](#) if you have a wireless card
- [Understanding Network Terminology](#) explains what you need to know about networking
- [Using ifconfig and route](#) explains how to setup your networking manually

### 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 12: 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](#).

### Preparing for Wireless Access

**Note:** Not all LiveCDs have the `iwconfig` command. If yours doesn't, you can still get the extensions working by following the instructions of the [linux-wlan-ng project](#).
If you are using a wireless (802.11) card, you may need to configure your wireless settings before going any further. To see the current wireless settings on your card, you can use `iwconfig`. Running `iwconfig` might show something like:

**Code Listing 13: Showing the current wireless settings**

```bash
# iwconfig eth0
eth0      IEEE 802.11-DS  ESSID: "GentooNode"
Mode: Managed  Frequency: 2.442GHz  Access Point: 00:09:5B:11:CC:F2
Bit Rate:11Mb/s    Tx-Power=20 dBm    Sensitivity=0/65535
Retry limit:16    RTS thr:off    Fragment thr:off
Power Management:off
Link Quality:25/10  Signal level:-51 dBm  Noise level:-102 dBm
Rx invalid nw id:5901 Rx invalid crypt:0 Rx invalid frag:0 Tx
excessive retries:237 Invalid misc:350282 Missed beacon:84
```

**Note:** Some wireless cards may have a device name of `wlan0` instead of `eth0`.

For most users, there are only two settings that might be important to change, the ESSID (aka wireless network name) or the WEP key. If the ESSID and Access Point address listed are already that of your access point and you are not using WEP, then your wireless is working. If you need to change your ESSID, or add a WEP key, you can issue the following commands:

**Code Listing 14: Changing ESSID and/or adding WEP key**

```bash
(This sets the network name to "GentooNode")
# iwconfig eth0 essid GentooNode

(This sets a hex WEP key)
# iwconfig eth0 key 1234123412341234abcd

(This sets an ASCII key - prefix it with "s:")
# iwconfig eth0 key s:some-password
```

You can then confirm your wireless settings again by using `iwconfig`. Once you have wireless working, you can continue configuring the IP level networking options as described in the next section (**Understanding Network Terminology**) or use the `net-setup` tool as described previously.

**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:

<table>
<thead>
<tr>
<th>Code Listing 15: Example of an IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Address (numbers): 192.168.0.2</td>
</tr>
<tr>
<td>IP Address (bits): 11000000 10101000 00000000 00000010</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Code Listing 16: Example of network/host separation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP-address: 192 168 0 2</td>
</tr>
<tr>
<td>Netmask: 11111111 11111111 11111111 00000000</td>
</tr>
<tr>
<td>+--------------------------+--------+</td>
</tr>
<tr>
<td>Network                   Host</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Code Listing 17: Broadcast address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP-address: 192 168 0 2</td>
</tr>
<tr>
<td>Broadcast: 11000000 10101000 00000000 00000010</td>
</tr>
<tr>
<td>+--------------------------+--------+</td>
</tr>
<tr>
<td>Network                   Host</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Network Item</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your IP address</td>
<td>192.168.0.2</td>
</tr>
<tr>
<td>Netmask</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>Broadcast</td>
<td>192.168.0.255</td>
</tr>
<tr>
<td>Gateway</td>
<td>192.168.0.1</td>
</tr>
<tr>
<td>Nameserver(s)</td>
<td>195.130.130.5, 195.130.130.133</td>
</tr>
</tbody>
</table>

**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 18: 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 19: Using route**

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

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

**Code Listing 20: 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 21: /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

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**

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 x86 systems, these are called *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**

The x86 LiveCDs provide support for EVMS and LVM2. EVMS and LVM2 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 partitioning scheme we use throughout this book:

<table>
<thead>
<tr>
<th>Partition</th>
<th>Filesystem</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda1</td>
<td>ext2</td>
<td>32M</td>
<td>Boot partition</td>
</tr>
<tr>
<td>/dev/hda2</td>
<td>(swap)</td>
<td>512M</td>
<td>Swap partition</td>
</tr>
<tr>
<td>/dev/hda3</td>
<td>ext3</td>
<td>Rest of the disk</td>
<td>Root partition</td>
</tr>
</tbody>
</table>

If you are interested in knowing how big a partition should be, or even how many partitions you need, read on. Otherwise continue now with partitioning your disk by reading Using fdisk 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 webservers, mailserver, gnome, ...):

```
> 3 Filesystem usage example

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Type</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda5</td>
<td>ext3</td>
<td>509M</td>
<td>132M</td>
<td>351M</td>
<td>28%</td>
<td>/</td>
</tr>
<tr>
<td>/dev/hda2</td>
<td>ext3</td>
<td>5.0G</td>
<td>3.0G</td>
<td>1.8G</td>
<td>63%</td>
<td>/home</td>
</tr>
<tr>
<td>/dev/hda7</td>
<td>ext3</td>
<td>7.9G</td>
<td>6.2G</td>
<td>1.3G</td>
<td>83%</td>
<td>/usr</td>
</tr>
<tr>
<td>/dev/hda8</td>
<td>ext3</td>
<td>1011M</td>
<td>483M</td>
<td>477M</td>
<td>51%</td>
<td>/opt</td>
</tr>
<tr>
<td>/dev/hda9</td>
<td>ext3</td>
<td>2.0G</td>
<td>607M</td>
<td>1.3G</td>
<td>32%</td>
<td>/var</td>
</tr>
<tr>
<td>/dev/hda1</td>
<td>ext2</td>
<td>51M</td>
<td>17M</td>
<td>34M</td>
<td>36%</td>
<td>/boot</td>
</tr>
<tr>
<td>/dev/hda6</td>
<td>swap</td>
<td>516M</td>
<td>12M</td>
<td>504M</td>
<td>2%</td>
<td>&lt;not mounted&gt;</td>
</tr>
</tbody>
</table>

(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.

4.c. Using fdisk to Partition your Disk

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

<table>
<thead>
<tr>
<th>Partition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda1</td>
<td>Boot partition</td>
</tr>
<tr>
<td>/dev/hda2</td>
<td>Swap partition</td>
</tr>
<tr>
<td>/dev/hda3</td>
<td>Root partition</td>
</tr>
</tbody>
</table>

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):

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

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

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

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

```
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
```

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`:

```
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:

```
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:

```
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:

<table>
<thead>
<tr>
<th>Code Listing 8: Created boot partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command (for help): <strong>p</strong></td>
</tr>
<tr>
<td>Device Boot Start  End  Blocks  Id  System</td>
</tr>
<tr>
<td>/dev/hda1    1     14  105808+  83  Linux</td>
</tr>
</tbody>
</table>

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 `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:

<table>
<thead>
<tr>
<th>Code Listing 9: Partition listing after creating a swap partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command (for help): <strong>p</strong></td>
</tr>
<tr>
<td>Device Boot Start  End  Blocks  Id  System</td>
</tr>
<tr>
<td>/dev/hda1    *     14  105808+  83  Linux</td>
</tr>
<tr>
<td>/dev/hda2    15     81  506520  82  Linux swap</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Code Listing 10: Partition listing after creating the root partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command (for help): <strong>p</strong></td>
</tr>
<tr>
<td>Device Boot Start  End  Blocks  Id  System</td>
</tr>
<tr>
<td>/dev/hda1    *     14  105808+  83  Linux</td>
</tr>
<tr>
<td>/dev/hda2    15     81  506520  82  Linux swap</td>
</tr>
<tr>
<td>/dev/hda3    82     83  15120+  82  Linux</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda1 *</td>
<td>1</td>
<td>14</td>
<td>105808+</td>
<td>83</td>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>/dev/hda2</td>
<td>15</td>
<td>81</td>
<td>506520</td>
<td>82</td>
<td>Linux swap</td>
<td></td>
</tr>
<tr>
<td>/dev/hda3</td>
<td>82</td>
<td>3876</td>
<td>28690200</td>
<td>83</td>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

**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. 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 filesystems are chosen 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?**

The Linux kernel supports various filesystems. We'll explain ext2, ext3, ReiserFS, XFS and JFS as those filesystems are most commonly used on Linux systems.

**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 has an additional hashed b-tree indexing option that enables high performance in almost all situations. In short, 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:

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Creation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>mke2fs</td>
</tr>
<tr>
<td>ext3</td>
<td>mke2fs -j</td>
</tr>
<tr>
<td>reiserfs</td>
<td>mkreiserfs</td>
</tr>
<tr>
<td>xfs</td>
<td>mkfs.xfs</td>
</tr>
<tr>
<td>jfs</td>
<td>mkfs.jfs</td>
</tr>
</tbody>
</table>

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:

```bash
# 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:

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

To activate the swap partition, use **swapon**:

```bash
# swapon /dev/hda2
```
Create and activate the swap now.

4.e. 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 for every partition you created. As an example we mount the root and boot partition:

```bash
# 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 and then mount the filesystem:

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

Now continue with [Installing the Gentoo Installation Files](#).

5. Installing the Gentoo Installation Files

5.a. Installing a Stage Tarball

Setting the Date/Time Right

Before you continue you need to check your date/time and update it. A misconfigured clock may lead to strange results in the future!

To verify the current date/time, run `date`:

```bash
# date
Thu Apr  1 16:21:18 CEST 2004
```

If the date/time displayed is wrong, update it using the `date MDDhhmmCCYY` syntax (Month, Day, hour, minute, Century and Year). For instance, to set the date to April 1st, 16:21 in
the year 2004:

```
Code Listing 2: Setting the date/time
# date 040116212004
```

### 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 Universal 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 3: 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](http://www.gentoo.org/main/en/mirrors.xml) 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.html) stages (for the x86 architecture) are inside the subdirectory `hardened/stages/`.

```
Code Listing 4: Surfing to the mirror listing with lynx
```

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 5: Surfing to the mirror listing with links2
# links2 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 6: 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 7: 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 8: 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 9: 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`):
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 11: 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 from the LiveCD. 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)
- [Configuring the Compile Options](#) (all other installation methods)

**Installing a Portage Snapshot and Source Code from LiveCD**

There is a Portage snapshot available on the Universal 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 12: 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 13: 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:

```bash
# 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.

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.

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:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Subarchitecture</th>
<th>CHOST Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>i386</td>
<td>i386-pc-linux-gnu</td>
</tr>
<tr>
<td>x86</td>
<td>i486</td>
<td>i486-pc-linux-gnu</td>
</tr>
<tr>
<td>x86</td>
<td>i586</td>
<td>i586-pc-linux-gnu</td>
</tr>
<tr>
<td>x86</td>
<td>i686 and above (incl. athlon)</td>
<td>i686-pc-linux-gnu</td>
</tr>
<tr>
<td>alpha</td>
<td></td>
<td>alpha-unknown-linux-gnu</td>
</tr>
<tr>
<td>ppc</td>
<td></td>
<td>powerpc-unknown-linux-gnu</td>
</tr>
<tr>
<td>sparc</td>
<td></td>
<td>sparc-unknown-linux-gnu</td>
</tr>
<tr>
<td>hppa</td>
<td>(generic)</td>
<td>hppa-unknown-linux-gnu</td>
</tr>
</tbody>
</table>

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:

```
- m ar ch=ath l on- xp
```

A second one is the `-O` flag (that is a capital O, not a zero), which specifies the **gcc** optimization class flag. Possible classes are `s` (for size-optimized), `0` (zero - 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:

```
- 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:

```
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

```bash
MAKEOPTS="-j 2"
```

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

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

```bash
# 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

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

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 3: 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 4: 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/profile` will override anything defined in `/etc/make.profile/make.defaults`.
et c/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/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.

```plaintext
Code Listing 5: 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:

```plaintext
Code Listing 6: Opening /etc/make.conf

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

```plaintext
Code Listing 7: 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 and 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.

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 won't 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 8: Deselecting java support**

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

Don't forget to unset the variable after bootstrapping:

**Code Listing 9: Unsetting USE**

```bash
# 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 connection :-) 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 10: Downloading the necessary sources**

```bash
# 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.
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:

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

Now continue with the next step, [Progressing from Stage 2 to Stage 3](#).

### 6.d. Progressing from Stage 2 to Stage 3

#### 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.

```
# 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.

```
# 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 15: Building the System

```
# emerge system
```

You can for now safely ignore any warnings about updated configuration files (and running `etc-update`). When your Gentoo system is fully installed and booted, do read our documentation on [Configuration File Protection](#).

When the build process has completed, continue with [Configuring the Kernel](#).

## 7. Configuring the Kernel

### 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), *devel opmpt-sources* (vanilla 2.6 kernel source), ...

If you are performing a network-less install you can't pick any of the available kernels as we only provide sources for a few of them. If this is the case, you are restricted to using *vanilla-sources*, more specifically version 2.4.24.

Choose your kernel source and install it using `emerge`.
In the next Code Listing we show you how to install the \texttt{vanilla-sources-2.4.24} (as this is the available kernel source for networkless installations) and \texttt{gentoo-sources}. Of course substitute with your choice of sources, these are merely examples. Also note that you should install only \textit{one} kernel source.

\begin{verbatim}
Code Listing 2: Installing a kernel source

(If you are performing a networkless installation)
# emerge =vanilla-sources-2.4.24

(Otherwise just use the kernel source name you want)
# emerge gentoo-sources
\end{verbatim}

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

\begin{verbatim}
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-2.4.24
\end{verbatim}

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

\begin{verbatim}
Code Listing 4: Changing the kernel source symlink

# rm /usr/src/linux && ln -s /usr/src/linux-2.4.24 /usr/src/linux
\end{verbatim}

Now it is time to configure and compile your kernel source. You can use \texttt{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 \textbf{Default: Manual Configuration}. If you want to use \texttt{genkernel} you should read \textbf{Alternative: Using genkernel} instead.

\section*{Default: Manual Configuration}

\subsection*{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 \textit{is} true: you must know your system when you start configuring a kernel manually. Most information can be gathered by viewing the contents of \texttt{/proc/pci} (or by using \texttt{lspci} if available). You can also run \texttt{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 \texttt{make menuconfig}. This will fire up an
ncurses-based configuration menu.

```bash
# 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:

```bash
Code Listing 6: Selecting experimental 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 filesystem, /dev filesystem + Automatically mount at boot:

```bash
Code Listing 7: Selecting necessary file systems
```

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 or you are using a dial-up modem, you will need the following options in the kernel:

```bash
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.

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

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.

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 10: Compiling the kernel

(For 2.4 kernel)

# make dep && make bzImage 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-2.4.24:

Code Listing 11: Installing the kernel

# cp arch/i386/boot/bzImage /boot/kernel-2.4.24
# cp System.map /boot/System.map-2.4.24

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

```bash
# 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!

Note that, if your boot partition doesn't use ext2 or ext3 as filesystem you need to manually configure your kernel using `genkernel --menuconfig all` and add support for your filesystem in the kernel (i.e. not as a module).

```bash
# genkernel all
```

GenKernel v3.0.1_beta10
* ARCH: x86
* KERNEL VER: 2.4.24
* 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.

### Code Listing 15: Checking the created kernel image name and initrd

```
# ls /boot/kernel* /boot/initrd*
```

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 16: 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:

<table>
<thead>
<tr>
<th>Ebuild</th>
<th>Purpose</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvidia-kernel</td>
<td>Accelerated NVIDIA graphics for XFree86</td>
<td>emerge nvidia-kernel</td>
</tr>
<tr>
<td>nforce-net</td>
<td>On-board ethernet controller on NVIDIA NForce (2) motherboards</td>
<td>emerge nforce-net</td>
</tr>
<tr>
<td>nforce-audio</td>
<td>On-board audio on NVIDIA NForce(2) motherboards</td>
<td>emerge nforce-audio</td>
</tr>
<tr>
<td>e100</td>
<td>Intel e100 Fast Ethernet Adapters</td>
<td>emerge e100</td>
</tr>
<tr>
<td>e1000</td>
<td>Intel e1000 Gigabit Ethernet Adapters</td>
<td>emerge e1000</td>
</tr>
<tr>
<td>emu10k1</td>
<td>Creative Sound Blaster Live!/Audigy support</td>
<td>emerge emu10k1</td>
</tr>
<tr>
<td>ati-drivers</td>
<td>Accelerated ATI Radeon 8500+/FireGL graphics for XFree86</td>
<td>emerge ati-drivers</td>
</tr>
<tr>
<td>ati-drivers-extra</td>
<td>Graphical ATI tools</td>
<td>emerge ati-drivers-extra</td>
</tr>
<tr>
<td>xfree-drm</td>
<td>Accelerated graphics for ATI Radeon up to 9200, Rage128, Matrox, Voodoo and other cards for XFree86. Please check the I USE_VI DEO_CARDS variable in the /usr/portage/x11-base/xfree-drm ebuilds to see what you need to fill in as your card.</td>
<td>VI DEO_CARDS=&quot;yourcard&quot; emerge xfree-drm</td>
</tr>
</tbody>
</table>

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 17: View full installation package listing

```
```
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:

```
# emerge --pretend --verbose emu10k1
...
```

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:

```
# 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**

You should list the modules you want automatically loaded in `/etc/modules.autoload.d/` (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:

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

For instance, to automatically load the `3c59x.o` module, edit the `kernel-2.4` or `kernel-2.6` file and enter the module name in it.

```
# nano -w /etc/modules.autoload.d/kernel-2.4

(Example for 2.4 kernel)
# nano -w /etc/modules.autoload.d/kernel-2.4

3c59x
```

Now run `modules-update` to commit your changes to the `/etc/modules.conf` file:
Continue the installation with Configuring your System.

8. Configuring your System

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:

```bash
# 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 (such as PPC), 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:
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
```
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):

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

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

```
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:

```
(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.

```
# 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:

```bash
# 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:

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

```plaintext
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:

```plaintext
127.0.0.1     localhost
```

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

**Note:** `pcmcia-cs` is only available for x86, amd64 and ppc platforms.

PCMCIA-users should first install the `pcmcia-cs` package. The `USE="-X"` is necessary to avoid installing XFree86 at this moment:

```bash
# cd /etc/init.d
# ln -s net.eth0 net.eth1
# rc-update add net.eth1 default
```
When `pcmcia-a-cs` is installed, add `pcmcia` to the default runlevel:

```bash
# USE="-X" emerge pcmcia-a-cs

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:

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

Code Listing 22: Opening /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".

**PPC** uses x86 keymaps on most systems. Users who want to be able to use ADB keymaps on boot have to enable ADB keycode sendings in their kernel and have to set a mac/ppc keymap in `rc.conf`.

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

### 9. Configuring the Bootloader

#### 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**. For x86, Gentoo Linux provides GRUB and LILO. But before we install one of these two bootloaders, we inform you how to configure framebuffer (in case you want it of course). With framebuffer you can run the Linux command line with (limited) graphical features (such as using the nice bootsplash image Gentoo provides).

**Optional: Framebuffer**
If you have configured your kernel with framebuffer support (or you used `genkernel`'s default kernel configuration), 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.

<table>
<thead>
<tr>
<th>640x480</th>
<th>800x600</th>
<th>1024x768</th>
<th>1280x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bpp</td>
<td>769</td>
<td>771</td>
<td>773</td>
</tr>
<tr>
<td>16 bpp</td>
<td>785</td>
<td>788</td>
<td>791</td>
</tr>
<tr>
<td>32 bpp</td>
<td>786</td>
<td>789</td>
<td>792</td>
</tr>
</tbody>
</table>

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

Now continue by installing GRUB or LILO.

### 9.b. Default: Using GRUB

#### Understanding GRUB's terminology

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:

```
# emerge --usepkg grub
```

Once installed, we need to put GRUB in the MBR (Master Boot Record) of your disk so that your system activates it when you boot up. We use the `grub-install` tool to safely setup GRUB.

```
Code Listing 2: Setting up GRUB
```
# grub-install --root-directory=/boot /dev/hda

**Note:** If your system does not have any floppy drives, add the `--no-floppy` option to the above command to prevent grub from probing the (non-existing) floppy drives. Also, if you don't have a separate /boot partition, omit the `--root-directory=/boot` option.

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 3: 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-2.4.24`. 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 4: 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 :)
# Comment out if you don't have a graphics card installed
splashimage=(hd0,0)/grub/splash.xpm.gz

# Partition where the kernel image (or operating system) is located
title=Gentoo Linux 2.4.24
root (hd0,0)
kernal /kernel-2.4.24 root=/dev/hda3

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

**Code Listing 5: grub.conf for 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 :) 
# Comment out if you don't have a graphics card installed
splashimage=(hd0,0)/grub/splash.xpm.gz

# Partition where the kernel image (or operating system) is located

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

title=Gentoo Linux 2.4.24
root (hd0,0)
kernel /kernel-2.4.24 root=/dev/ram0 init=/linuxrc real_root=/dev/hda3
initrd /initrd-2.4.24

# 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 6: Adding the vga-statement as a kernel option

title=Gentoo Linux
root (hd0,0)
kernel /kernel-2.4.24 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.

If you have more questions regarding GRUB, please consult the GRUB FAQ or the GRUB Manual.

9.c. Alternative: Using LILO

Installing LILO

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 7: Installing 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.

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-2.4.24`. 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 8: Creating `/etc/lilo.conf`

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

### Code Listing 9: Example `/etc/lilo.conf`

```bash
boot=/dev/hda             # Install LILO in the MBR
prompt                    # Give the user the chance to select another section
timeout=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-2.4.24  # Name we give to this section
label=gentoo              # Start with a read-only root. Do not alter!
read-only                # Location of the root filesystem
root=/dev/hda3            # For genkernel users
image=/boot/kernel-2.4.24  # Name we give to this section
label=gentoo
read-only
root=/dev/ram0
append="init=/linuxrc real_root=/dev/hda3"
initrd=/boot/initrd-2.4.24

# The next two lines are only if you dual boot 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 `vga=788` statement to enable framebuffer:

### Code Listing 10: Using append to add kernel options

```
# For non-genkernel users
image=/boot/kernel-2.4.24
label=gentoo
read-only
root=/dev/hda3
# For genkernel users
image=/boot/kernel-2.4.24
label=gentoo
read-only
root=/dev/ram0
append="init=/linuxrc real_root=/dev/hda3"
initrd=/boot/initrd-2.4.24
# The next two lines are only if you dual boot with a Windows system
# In this case, Windows is hosted on /dev/hda6.
other=/dev/hda6
label=windows
```

```
```
gentoo

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). Keep in mind that you'll also have to rerun /sbin/lilo every time you install a new kernel!

```
Code Listing 11: Finishing the LILO Installation
#
```

Now continue with Installing Necessary System Tools.

10. Installing Necessary System 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
#
```

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 vi xi e-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 vi xi e-cron.

Code Listing 2: Installing a cron daemon

```
# emerge vi xi e-cron
# rc-update add vi xi e-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:

<table>
<thead>
<tr>
<th>File System</th>
<th>Tool</th>
<th>Install Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFS</td>
<td>xfsprogs</td>
<td>emerge xfsprogs</td>
</tr>
<tr>
<td>ReiserFS</td>
<td>reiserfsprogs</td>
<td>emerge reiserfsprogs</td>
</tr>
<tr>
<td>JFS</td>
<td>jfsutils</td>
<td>emerge jfsutils</td>
</tr>
</tbody>
</table>

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
11.a. User Administration

Setting a root Password

Before you forget, set the root password by typing:

```
# passwd
```

If you want root to be able to log on through the serial console, add `tts/0` to `/etc/securetty`:

```
# echo "tts/0" >> /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):

```
# 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. Then type in that one magical command you have been waiting for: `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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Optional%3A%20Configuring%20BootX). **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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Optional%3A%20Getting%20Gentoo%2FMIPS%20to%20Work).

GRP users can continue with [Optional: Install GRP Packages](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Optional%3A%20Install%20GRP%20Packages), all the rest can finish up with [Where to go from here?](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Where%20to%20go%20from%20here?).

### 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#BootX%20Main%20Page) for more information.

If you are a GRP users you can continue with [Optional: Install GRP Packages](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Optional%3A%20Install%20GRP%20Packages), otherwise go to [Where to go from here?](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1#Where%20to%20go%20from%20here?).

### Optional: Getting Gentoo/MIPS to Work

**Important:** This subsection is *only* for **MIPS** users! All other readers should skip this subsection.

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 change the Portage configuration to look for the prebuilt binaries from the second CD (Gentoo Packages CD). First mount this CD:

Code Listing 7: Mount the Packages CD

# mkdir /mnt/cdrom
(Put the Gentoo Packages CD in the CD tray)
# mount /mnt/cdrom

Now configure Portage to use /mnt/cdrom for its prebuilt packages:
Now install the packages you want. The Packages CD 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 equipped! Continue with [Where to go from here?](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) to learn more about Gentoo.

### 12. Where to go from here?

#### 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1).

For a full listing of all our available documentation check out our [Documentation Resources](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) 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 :)

B. Working with Gentoo

1. 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 management 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:

```plaintext
Code Listing 1: A short snippet of available USE-flags

*gtk*     - Adds support for x11-libs/gtk+ (The GIMP Tool kit)
*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:

```plaintext
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
    infjpeg kde gnome libg++ libwww mad nanmed motif mpeg ncurses nls
    ogg vorbis openogl pam pdflib png python qt quicktime readline sdl slang
    spell ssh svga tcpd truetype X xml2 xmxm 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`:

```plaintext
Code Listing 3: An example USE setting in /etc/make.conf

USE="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 environment variable

```bash
# 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

```bash
<table>
<thead>
<tr>
<th>Package</th>
<th>USE-flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnome</td>
<td>gnome-base/gnome</td>
</tr>
<tr>
<td>gtk</td>
<td>x11-libs/gtk+</td>
</tr>
<tr>
<td>qt</td>
<td>x11-libs/qt</td>
</tr>
<tr>
<td>kde</td>
<td>kde-base/kdebase</td>
</tr>
<tr>
<td>motif</td>
<td>x11-libs/openmotif</td>
</tr>
</tbody>
</table>
```

---

**Precendence**

Of course there is a certain precedence on what setting has priority over the USE setting. You don't want to declare `USE="-java"` only to see that `java` is 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

```bash
# 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 +dap
+gnome -debug +mozcalendar -mozaccess -mozxmlterm -moznoirc -moznomail
- moznocompose - moznofont
```

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
[ Col our 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

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 **ebuilder**; people interested in becoming Gentoo professionals will learn how to use **ebuilder** later on. For now, we will focus on **emerge** as it has functionality that **ebuilder** 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-php/           GNOME-office/   net-wireless/
app-benchmarks/  dev-ruby/          header.txt      net-www/
app-cdr/         dev-python/        iconfing/       net-zope/
app-crypt/       dev-tcltk/         jytHon/         packages/
app-dt/c/        dev-tex/           kde-apps/       profiles/
app-doc/         dev-tex/           kde-base/       relases/
app-editors/     dev-text/          kde-i18n/        scripts/
app-emacs/       dev-util/          kde-libs/       sec-policy/
app-emulation/   dev-util/          licenses/       skel.ChangeLog
app-games/       experimetal/       medi-a-fonts/   skel.ebuild
app-gnutp/       files/             medi-a-gfx/     skel.metadata.xml
app-i18n/        fresco-base/       medi-a-libs/    snapshots/
app-mentsc/      games-action/      medi-a-pl-ugni/ sys-apps/
app-office/      games-arcade/      medi-a-radi-o/  sys-build/
app-pda/         games-board/       medi-a-sound/   sys-cluster/
app-portage/     games-emulation/   medi-a-tv/       sys-devel/
app-sci/         games-engines/     medi-a-vdeo/     sys-fs/
app-shells/      games-fps/         medi-a-vid/     sys-frontend/
app-text/        games-ki/          medi-adat/      sys-kernel/
app-viol/        games-misc/        net-analyzer/   sys-kmodds/
app-xemacs/      games-md/          net-apache/     sys-libs/
berlin-/base/    games-puzzle/      net-di-alup/    unix-x2tcp/
dev-ada/         games-roguelike/    net-dns/        x11-base/
dev-cpp/         games-rpg/         net-firewall/   x11-lib/
dev-db/          games-server/      net-fs/         x11-lib/
dev-dotnet/      games-simulatiion/  net-ftp/        x11-misc/
dev-embedded/    games-sports/      net-i18n/       x11-pl-ugni/
dev-games/       games-strategy/    net-irc/        x11-terms/
dev-haskell/     games-util/        net-libs/       x11-themes/
dev-java/        glp/               net-mail/       xfce-base/
dev-lang/        gnome-apps/        net-misc/       xfce-extra/
dev-lib/         gnome-base/        net-news/       xfce-xtrar/
dev-libp/        gnome-extra/
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/    sc/
dia2code/    gnumeric/  lxbank/     openoffice/             scribus/   siag/
factural ux/  ical/      lyx/        openoffice-bin/        texmacs/
gaby/        kbudget/   mdbtools/    phprojekt/            
gnofin/      khacc/     mrproject/  phprojekt/          
```

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 file sizes 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   office-wapper-1.3
newstl portfix.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`:

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

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 advisable 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 `mirrorselct` to do this for you automatically. As the `SYNC` variable will be discussed later on, we will focus on using `mirrorselct`. First install `mirrorselct` by emerging it:

```
# emerge --usepkg mirrorselct
```

Now run `mirrorselct` to automatically select mirrors for you (it will also setup Portage to use a mirror for the source code):

```
# mirrorselct
```
To update Portage using rsync, simply run `emerge sync`:

```bash
# mirrorselect -a -s3
```

Updating Portage

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.

```bash
# emerge sync
```

```bash
# 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 `--empt yt re` (-e in short). This is useful with `--pret end` to display a complete tree of dependencies for any particular package. Without `--pret end`, `emerge` will (re)compile all listed packages.

```
# emerge --empt yt re --pret end 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.

```
# emerge --nodeps gnumeric
```

The opposite of `--nodeps` is `--onl ydeps`, which will have Portage install all dependencies of a given package, but not the package itself:

```
# emerge --onl ydeps 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 `--updat e` option (-u in short):

```
# emerge --updat e system
```

An identical approach can be used for the world packages:

```
# emerge --updat e world
```

Again, if you want to see what `emerge` wants to update, use the `--pret end` option together
with the \texttt{--update} option:

\textbf{Code Listing 29: Pretending to update your entire system}

\begin{verbatim}
# emerge --pretend --update world
(Some output removed to improve readability)
[ebuild U] net-misc/wget-1.9-r1 [1.9]
[ebuild UD] media-video/dvdauthor-0.5.0 [0.5.3]
[ebuild U] net-analyzer/ethereal-0.9.16 [0.9.14]
\end{verbatim}

Right next to the word "ebuild" you will notice a letter (or combination of letters) which gives you more information about the package:

- \textbf{B} (blocks) The package listed to the left is blocking the emerge of the package listed to the right.
- \textbf{N} (new) The package is new to your system and will be emerged for the first time.
- \textbf{R} (reemerge) The package isn't new, but needs to be reemerged.
- \textbf{F} (fetch) The package requires that you download the source code manually (for instance due to licencing issues).
- \textbf{U} (update) The package already exists on your system but will be upgraded.
- \textbf{UD} (downgrade) The package already exists on your system but will be downgraded.
- \textbf{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 \texttt{--upgradenonly} option (-U in short):

\textbf{Code Listing 30: Upgrading your entire system}

\begin{verbatim}
# emerge --update --upgradenonly world
\end{verbatim}

Of course, we are talking here about \textit{system} and \textit{world}, but you can perform the same actions for individual software packages.

\textbf{Removing Software}

If you want to remove software from your system, you can use the \texttt{unmerge} option (-C - capital C - in short):

\textbf{Code Listing 31: Uninstalling software}

\begin{verbatim}
# emerge unmerge gnumeric
\end{verbatim}

If you want to test a removal (but not perform it), you can use \texttt{--pretend} again:

\textbf{Code Listing 32: Pretending to uninstall software}

\begin{verbatim}
# emerge --pretend unmerge gnumeric
\end{verbatim}
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:

```
# 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:

```
[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:

```
# emerge unmerge bonobo-activation
```

### 3. Portage 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...

## 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) setting. However, if you don't use Gnome and would still like to have the monitor then you should put 'gtk' in your [USE](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) 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

```bash
# 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) variable so it contains the `distcc` keyword. Next, edit the [MAKEOPTS](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) variable so it reads `-j X` 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

```bash
# Suppose you have 2 single-CPU distccd hosts excluding this host:
MAKEOPTS="-j 4"
```

Now, still inside `/etc/make.conf`, uncomment the [PORTAGE_TMPDIR](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1) 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

```bash
# 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

```bash
# 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:

```bash
# /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](http://www.cc-cache.com/).  

**Installing `ccache`**

Installing `ccache` with Gentoo is a breeze. Just emerge it and you're done :) 

```bash
# emerge ccache
```

**Activating Portage Support**

First, edit `/etc/make.conf` and alter the `FEATURES` variable so that it contains the `ccache` keyword:

```bash
FEATURES="ccache"
```

Next, edit (or create) the `CCACHE_SIZE` variable (also in `/etc/make.conf`) so it contains the amount of disk space you want to sacrifice for `ccache`:

```bash
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:

```bash
```

# 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:

```
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:

```
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:

```
FEATURES="sfperms"
```

### 3.e. Other Features

**Portage Help**

There are several other keywords you can place in the FEATURES variable. Most of them are
4. Controlling Portage Behaviour

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 ._cf_g0000_<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 CONFI_G_PROTECT variable lists all protected directories. All subdirectories of the listed directories are protected too. The CONFI_G_PROTECT variable is defined in /etc/make.glabal.s, 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

    CONFI_G_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 CONFI_G_PROTECT_MASK variable, which also has a default value defined in /etc/make.glabal.s but should be altered by declaring it in /etc/make.conf:

Code Listing 2: An example CONFIG_PROTECT_MASK setting

    CONFI_G_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

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 useful information, enter 2. If you want to interactively update your current configuration file, enter 3.

It is not useful 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
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.glib.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.glib.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`:
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* advisable 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.conf`) 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="/path/to/overlay"
```
Distfiles

The location of the downloaded source code (so called distfiles) is defined in the \texttt{DISTDIR} variable. It defaults to \texttt{${PORTDIR}/distfiles}:

\textbf{Code Listing 19: The DISTDIR variable}

\begin{verbatim}
DISTDIR="${PORTDIR}/distfiles"
\end{verbatim}

Packages and RPMs

The location of the prebuilt packages is defined in the \texttt{PKGDIR} variable. It defaults to \texttt{${PORTDIR}/packages}:

\textbf{Code Listing 20: The PKGDIR variable}

\begin{verbatim}
PKGDIR="${PORTDIR}/packages"
\end{verbatim}

The location of the RPMs (yes, some packages are available as RPMs) is defined in the \texttt{RPMDIR} variable. It defaults to \texttt{${PORTDIR}/rpm}:

\textbf{Code Listing 21: The RPMDIR variable}

\begin{verbatim}
RPMDIR="${PORTDIR}/rpm"
\end{verbatim}

Temporary Portage Files

Portage uses a temporary location to build its ebuilds in. This location is defined in the \texttt{PORTAGE_TMPDIR} variable. It defaults to \texttt{/var/tmp}:

\textbf{Code Listing 22: The PORTAGE_TMPDIR variable}

\begin{verbatim}
PORTAGE_TMPDIR="/var/tmp"
\end{verbatim}

Per default, Portage will create a \texttt{portage} directory inside \texttt{PORTAGE_TMPDIR}. This is declared in the \texttt{BUILD_PREFIX} variable:

\textbf{Code Listing 23: The BUILD_PREFIX variable}

\begin{verbatim}
BUILD_PREFIX="${PORTAGE_TMPDIR}/portage"
\end{verbatim}

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:

```plaintext
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:

```plaintext
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:

```plaintext
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`:

```plaintext
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 \texttt{CHOST}, \texttt{CFLAGS} and \texttt{CXXFLAGS} variables, used by the compiler to compile and optimize the packages.

More information about these variables can be found in the \texttt{gcc} info pages, or online in the GCC Online Manuals.

\begin{verbatim}
Code Listing 28: Getting information on CHOST, CFLAGS and CXXFLAGS

# info gcc
(Select "Invoking gcc")
(Select "Optimize options")
\end{verbatim}

If the \texttt{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.

\begin{verbatim}
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"
\end{verbatim}

The \texttt{MAKEOPTS} variable is used by \texttt{make}, a tool used to ease the compilation of a package. It is usually defined to tell \texttt{make} to run several compilations simultaneously (especially if you have a multi-CPU system, or are using \texttt{distcc} as described previously).

To have \texttt{make} run three compilations simultaneously, set the \texttt{MAKEOPTS} variable to \texttt{-j 3}:

\begin{verbatim}
Code Listing 30: The MAKEOPTS variable

MAKEOPTS="-j 3"
\end{verbatim}

The \texttt{ROOT} variable shouldn't be set in \texttt{/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 \texttt{/}. As an example we show you how to install \texttt{gnumeric} in \texttt{/mnt/gentoo} instead of in your running system:

\begin{verbatim}
Code Listing 31: The ROOT variable

# ROOT="/mnt/gentoo" emerge gnumeric
\end{verbatim}

Output Formatting

By default, Portage colors its output to improve readability. If you do not want this, set the \texttt{NOCOLOR} variable to \texttt{true}:

\begin{verbatim}
Code Listing 32: The NOCOLOR variable

# nocolor gcc
(Select "Invoking gcc")
(Select "Optimize options")
\end{verbatim}
4.e. Resources

Man Pages

If you need a quick reference on all listed variables, please consult the *make.conf* man page:

```bash
# 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](http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?style=printable&full=1).

5. Initscripts

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 beneath `Alt-F1`, `Alt-F2`, etc.) attaching a special process called `agetty` to it. This process will then make sure you are able to log on through these terminals by running `login`.

Init Scripts

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

First, init runs all scripts from /etc/init.d that have symbolic links inside /etc/runlevels/boots. 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/boots 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/boots. 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:
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>respawn:/sbin/agetty 38400 tty1 linux</td>
</tr>
<tr>
<td>c2</td>
<td>respawn:/sbin/agetty 38400 tty2 linux</td>
</tr>
<tr>
<td>c3</td>
<td>respawn:/sbin/agetty 38400 tty3 linux</td>
</tr>
<tr>
<td>c4</td>
<td>respawn:/sbin/agetty 38400 tty4 linux</td>
</tr>
<tr>
<td>c5</td>
<td>respawn:/sbin/agetty 38400 tty5 linux</td>
</tr>
<tr>
<td>c6</td>
<td>respawn:/sbin/agetty 38400 tty6 linux</td>
</tr>
</tbody>
</table>
```

### 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

```
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

```bash
# /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

```bash
# /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

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

To also ask what dependencies the service has, you can use `iuse` or `inneed`. With `inneed` 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

```bash
# /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

```bash
# /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

```bash
# /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

```bash
# 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

```bash
# 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)
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:

```plaintext
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.

```plaintext
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

```c
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

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

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

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

```c
depend() {
  before *
}
```
Standard Functions

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

```
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:

```
# 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`:

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

Service Configuration Variables

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

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

### 5.e. Changing the Runlevel Behaviour

**Who might benefit from this?**

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

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

#### Using SOFTLEVEL

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

**Code Listing 23: Creating a runlevel directory**

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

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

**Code Listing 24: Adding the necessary init scripts**

```bash
# ls /etc/runlevels/default
acpid  domainname  local  net.eth0  netmount  postfix  syslog-ng  vixie-cron
# rc-update add acpid offline
# rc-update add domainname offline
# rc-update add local offline
# rc-update add syslog-ng offline
# rc-update add vixie-cron offline
```

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

**Code Listing 25: Adding an entry for the offline runlevel**

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

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

#### Using BOOTSLEVEL
Using **boot level** is completely analogous to **soft level**. The only difference here is that you define a second "boot" runlevel instead of a second "default" runlevel.

### 6. Environment Variables

#### 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PATH</strong></td>
<td>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 <code>ls</code>, <code>rc-update</code> or <code>emerge</code>) 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 <code>/bin/ls</code>).</td>
</tr>
<tr>
<td><strong>ROOTPATH</strong></td>
<td>This variable has the same function as <strong>PATH</strong>, but this one only lists the directories that should be checked when the root-user enters a command.</td>
</tr>
<tr>
<td><strong>LDPATH</strong></td>
<td>This variable contains a colon-separated list of directories in which the dynamical linker searches through to find a library.</td>
</tr>
<tr>
<td><strong>MANPATH</strong></td>
<td>This variable contains a colon-separated list of directories in which the <code>man</code> command searches for the man pages.</td>
</tr>
<tr>
<td><strong>INFODIR</strong></td>
<td>This variable contains a colon-separated list of directories in which the <code>info</code> command searches for the info pages.</td>
</tr>
<tr>
<td><strong>PAGER</strong></td>
<td>This variable contains the path to the program used to list the contents of files through (such as <code>less</code> or <code>more</code>).</td>
</tr>
<tr>
<td><strong>EDITOR</strong></td>
<td>This variable contains the path to the program used to change the contents of files with (such as <code>nano</code> or <code>vi</code>).</td>
</tr>
<tr>
<td><strong>KDEDIRS</strong></td>
<td>This variable contains a colon-separated list of directories which contain KDE-specific material.</td>
</tr>
<tr>
<td><strong>CLASSPATH</strong></td>
<td>This variable contains a colon-separated list of directories which contain Java classes.</td>
</tr>
<tr>
<td><strong>CONFIG_PROTECT</strong></td>
<td>This variable contains a space-delimited list of directories which should be protected by Portage during updates.</td>
</tr>
<tr>
<td><strong>CONFIG_PROTECT_MASK</strong></td>
<td>This variable contains a space-delimited list of directories which should not be protected by Portage during updates.</td>
</tr>
</tbody>
</table>

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

**Code Listing 1: Example definitions**

---

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:

```bash
Code Listing 2: /etc/conf.d/05gcc
PATH="/usr/i686-pc-linux-gnu/gcc-bi n/3.2"
ROOTPATH="/usr/i686-pc-linux-gnu/gcc-bi n/3.2"
MANPATH="/usr/share/gcc-data/i686-pc-linux-gnu/3.2/man"
IN FOPATH="/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:

```bash
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.

<table>
<thead>
<tr>
<th>Code Listing 4: Update order used by env-update</th>
</tr>
</thead>
<tbody>
<tr>
<td>00basic c</td>
</tr>
<tr>
<td>+----------------+-------------+-------------</td>
</tr>
<tr>
<td>PATH=&quot;/bin:/usr/bin:/usr/kde/3.2/bin:/usr/local/bin&quot;</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Code Listing 5: Updating the environment</th>
</tr>
</thead>
<tbody>
<tr>
<td># env-update &amp; &amp; source /etc/profile</td>
</tr>
</tbody>
</table>

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/ny_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`:

<table>
<thead>
<tr>
<th>Code Listing 6: Extending PATH for local usage in ~/.bashrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH=&quot;${PATH}:/home/ny_user/bin&quot;</td>
</tr>
</tbody>
</table>

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
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**

```bash
# export PATH="${PATH}:~/tmp/usr/bin"
```

The contents of this document are licensed under the [Creative Commons - Attribution / Share Alike](http://creativecommons.org/licenses/by-sa/4.0) license.