gentoo linux™ (/) Wiki

Gentoo Linux amd64 Handbook: Installing Gentoo

From Gentoo Wiki
< Handbook:AMD64 (/wiki/Special:MyLanguage/Handbook:AMD64) | Full (/wiki/Special:MyLanguage/Handbook:AMD64/Full)
Jump to:navigation Jump to:search

The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

Contents

Introduction

Welcome

Welcome to Gentoo! Gentoo is a free operating system based on Linux that can be automatically optimized and customized for just about any application or need. It is built on an ecosystem of free software and does not hide what is running beneath the hood from its users.

Openness

Gentoo's premier tools are built from simple programming languages. Portage (/wiki/Portage), Gentoo's package maintenance system, is written in Python (https://gitweb.gentoo.org/proj/portage.git/). Ebuilds, which provide package definitions for Portage are written in bash (https://gitweb.gentoo.org/repo/gentoo.git). Our users are encouraged to review, modify, and enhance the source code for all parts of Gentoo.

By default, packages are only patched when necessary to fix bugs or provide interoperability within Gentoo. They are installed to the system by compiling source code provided by upstream projects into binary format (although support for precompiled binary packages is included too). Configuring Gentoo happens through text files.

For the above reasons and others: *openness* is built-in as a *design principal*.

Choice

Choice is another Gentoo *design principal.*

When installing Gentoo, choice is made clear throughout the Handbook. System administrators can choose two fully supported init systems (Gentoo's own OpenRC (/wiki/OpenRC) and Freedesktop.org's systemd (/wiki/Systemd)), partition structure for storage disk(s), what file systems to use on the disk(s), a target system profile (/wiki/Profile), remove or add features on a global (system-wide) or package specific level via USE flags, bootloader, network management utility, and much, much more.

As a development philosophy, Gentoo's authors (https://www.gentoo.org/inside-gentoo/developers/) try to avoid forcing users onto a specific system profile or desktop environment. If something is offered in the GNU/Linux ecosystem, it's likely available in Gentoo. If not, then we'd love to see it so. For new package requests please file a bug report (https://bugs.gentoo.org) or create your own ebuild repository.

Power

Being a source-based operating system allows Gentoo to be ported onto new computer instruction set architectures (https://en.wikipedia.org/wiki/instruction_set_architecture) and also allows all installed packages to be tuned. This strength surfaces another Gentoo *design principal*: *power*.

A system administrator who has successfully installed and customized Gentoo has compiled a tailored operating system from source code. The entire operating system can be tuned at a binary level via the mechanisms included in Portage's `make.conf (/wiki//etc/portage/make.conf)` file. If so desired, adjustments can be made on a per-package basis, or a package group basis. In fact, entire sets of functionality can be added or removed using USE flags.

It is very important that the Handbook reader understands that these design principals are what makes Gentoo unique. With the principals of great power, many choices, and extreme openness highlighted, diligence, thought, and intentionality should be employed while using Gentoo.

How the installation is structured

The Gentoo Installation can be seen as a 10-step procedure, corresponding to the next set of chapters. Each step results in a certain state:

Step	Result

1	The user is in a working environment ready to install Gentoo.
2	The Internet connection is ready to install Gentoo.
3	The hard disks are initialized to host the Gentoo installation.
4	The installation environment is prepared and the user is ready to chroot (/wiki/Chroot) into the new environment.
5	Core packages, which are the same on all Gentoo installations, are installed.
6	The Linux kernel is installed.
7	Most of the Gentoo system configuration files are created.
8	The necessary system tools are installed.
9	The proper boot loader has been installed and configured.
10	The freshly installed Gentoo Linux environment is ready to be explored.

Whenever a certain choice is presented the handbook will try to explain the pros and cons of each choice. Although the text then continues with a default choice (identified by "Default: " in the title), the other possibilities will be documented as well (marked by "Alternative: " in the title). Do not think that the default is what Gentoo recommends. It is, however, the choice that Gentoo believes most users will make.

Sometimes an optional step can be followed. Such steps are marked as "Optional: " and are therefore not needed to install Gentoo. However, some optional steps are dependent on a previously made decision. The instructions will inform the reader when this happens, both when the decision is made, and right before the optional step is described.

Installation options for Gentoo

Gentoo can be installed in many different ways. It can be downloaded and installed from official Gentoo installation media such as our bootable ISO images. The installation media can be installed on a USB stick or accessed via a netbooted environment. Alternatively, Gentoo can be installed from non-official media such as an already installed distribution or a non-Gentoo bootable disk (such as Knoppix (https://www.knopper.net/knoppix/index-en.html)).

This document covers the installation using official Gentoo Installation media or, in certain cases, netbooting.

> **Note**
> For help on the other installation approaches, including using non-Gentoo bootable media, please read our Alternative installation guide (/wiki/Installation_alternatives).

We also provide a Gentoo installation tips and tricks (/wiki/Gentoo_installation_tips_and_tricks) document that might be useful.

Troubles

If a problem is found in the installation (or in the installation documentation), please visit our bug tracking system (https://bugs.gentoo.org) and check if the bug is known. If not, please create a bug report for it so we can take care of it. Do not be afraid of the developers who are assigned to the bugs - they (generally) don't eat people.

Although this document is architecture-specific, it may contain references to other architectures as well, because large parts of the Gentoo Handbook use text that is identical for all architectures (to avoid duplication of effort). Such references have been kept to a minimum, to avoid confusion.

If there is some uncertainty whether or not the problem is a user-problem (some error made despite having read the documentation carefully) or a software-problem (some error we made despite having tested the installation/documentation carefully) everybody is welcome to join the `#gentoo (ircs://irc.libera.chat/#gentoo)` (`webchat (https://web.libera.chat/#gentoo)`) channel on irc.libera.chat. Of course, everyone is welcome otherwise too as our chat channel covers the broad Gentoo spectrum.

Speaking of which, if there are any additional questions regarding Gentoo, check out the Frequently Asked Questions (/wiki/FAQ) article. There are also FAQs (https://forums.gentoo.org/viewforum.php?f=40) on the Gentoo Forums (https://forums.gentoo.org).

Hardware requirements

Before we start, we first list what hardware requirements are needed to successfully install Gentoo on a amd64 box.

AMD64 livedisk hardware requirements

	Minimal CD	LiveDVD
CPU	Any x86-64 (https://en.wikipedia.org/wiki/X86-64) CPU, both AMD64 and Intel 64	
Memory	2 GB	
Disk space	8 GB (excluding swap space)	
Swap space	At least 2 GB	

The AMD64 project (/wiki/Project:AMD64) is a good place to be for more information about Gentoo's **amd64** support.

Gentoo Linux installation media

Tip
It is okay to use other, non-Gentoo installation media, although official media is recommended. Gentoo installation media ensures the necessary tools are around. When using non-Gentoo media, skip to Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

Minimal installation CD

The Gentoo minimal installation CD is a bootable image: a self-contained Gentoo environment. It allows the user to boot Linux from the CD or other installation media. During the boot process the hardware is detected and the appropriate drivers are loaded. The image is maintained by Gentoo developers and allows

anyone to install Gentoo if an active Internet connection is available.

The Minimal Installation CD is called `install-amd64-minimal-<release>.iso`.

The occasional Gentoo LiveDVD

Occasionally, a special DVD image is crafted which can be used to install Gentoo. The instructions in this chapter target the Minimal Installation CD, so things might be a bit different when booting from the LiveDVD. However, the LiveDVD (or any other official Gentoo Linux environment) supports getting a root prompt by just invoking **`sudo su -`** or **`sudo -i`** in a terminal.

What are stages then?

A stage3 tarball is an archive containing a profile specific minimal Gentoo environment. Stage3 tarballs are suitable to continue the Gentoo installation using the instructions in this handbook. Previously, the handbook described the installation using one of three stage tarballs (/wiki/Stage_tarball). Gentoo does not offer stage1 and stage2 tarballs for download any more since these are mostly for internal use and for bootstrapping Gentoo on new architectures.

Stage3 tarballs can be downloaded from `releases/amd64/autobuilds/` on any of the official Gentoo mirrors (https://www.gentoo.org/downloads/mirrors/). Stage files update frequently and are not included in official installation images.

Downloading

Obtain the media

The default installation media that Gentoo Linux uses are the *minimal installation CDs*, which host a bootable, very small Gentoo Linux environment. This environment contains all the right tools to install Gentoo. The CD images themselves can be downloaded from the downloads page (https://www.gentoo.org/downloads/) (recommended) or by manually browsing to the ISO location on one of the many available mirrors (https://www.gentoo.org/downloads/mirrors/).

If downloading from a mirror, the minimal installation CDs can be found as follows:

1. Go to the `releases/` directory.
2. Select the directory for the relevant target architecture (such as **`amd64/`**).
3. Select the `autobuilds/` directory.
4. For **`amd64`** and **`x86`** architectures select either the `current-install-amd64-minimal/` or `current-install-x86-minimal/` directory (respectively). For all other architectures navigate to the `current-iso/` directory.

> **Note**
> Some target architectures such as **`arm`**, **`mips`**, and **`s390`** will not have minimal install CDs. At this time the Gentoo Release Engineering project (/wiki/Project:RelEng) does not support building `.iso` files for these targets.

Inside this location, the installation media file is the file with the `.iso` suffix. For instance, take a look at the following listing:

CODE **Example list of downloadable files at releases/amd64/autobuilds/current-iso/**

```
[DIR] hardened/                                    05-Dec-2014 01:42    -
[   ] install-amd64-minimal-20141204.iso           04-Dec-2014 21:04  208M
[   ] install-amd64-minimal-20141204.iso.CONTENTS  04-Dec-2014 21:04  3.0K
[   ] install-amd64-minimal-20141204.iso.DIGESTS   04-Dec-2014 21:04  740
[TXT] install-amd64-minimal-20141204.iso.asc       05-Dec-2014 01:42  1.6K
[   ] stage3-amd64-20141204.tar.bz2                04-Dec-2014 21:04  198M
[   ] stage3-amd64-20141204.tar.bz2.CONTENTS       04-Dec-2014 21:04  4.6M
[   ] stage3-amd64-20141204.tar.bz2.DIGESTS        04-Dec-2014 21:04  720
[TXT] stage3-amd64-20141204.tar.bz2.asc            05-Dec-2014 01:42  1.5K
```

In the above example, the `install-amd64-minimal-20141204.iso` file is the minimal installation CD itself. But as can be seen, other related files exist as well:

- A `.CONTENTS` file which is a text file listing all files available on the installation media. This file can be useful to verify if particular firmware or drivers are available on the installation media before downloading it.
- A `.DIGESTS` file which contains the hash of the ISO file itself, in various hashing formats/algorithms. This file can be used to verify if the downloaded ISO file is corrupt or not.
- A `.asc` file which is a cryptographic signature of the ISO file. This can be used to both verify if the downloaded ISO file is corrupt or not, as well as verify that the download is indeed provided by the Gentoo Release Engineering team and has not been tampered with.

Ignore the other files available at this location for now - those will come back when the installation has proceeded further. Download the `.iso` file and, if verification of the download is wanted, download the `.iso.asc` file for the `.iso` file as well. The `.CONTENTS` file does not need to be downloaded as the installation instructions will not refer to this file anymore, and the `.DIGESTS` is not needed if the signature in the `.iso.asc` file is verified.

Verifying the downloaded files

Note
This is an optional step and not necessary to install Gentoo Linux. However, it is recommended as it ensures that the downloaded file is not corrupt and has indeed been provided by the Gentoo Infrastructure team (/wiki/Project:Infrastructure).

The `.asc` file provides a cryptographic signature of the ISO. By validating it, one can make sure that the installation file is provided by the Gentoo Release Engineering team and is intact and unmodified.

Microsoft Windows based verification

To first verify the cryptographic signature, tools such as GPG4Win (https://www.gpg4win.org/) can be used. After installation, the public keys of the Gentoo Release Engineering team need to be imported. The list of keys is available on the signatures page (https://www.gentoo.org/downloads/signatures/). Once imported, the user can then verify the signature in the `.asc` file.

Linux based verification

On a Linux system, the most common method for verifying the cryptographic signature is to use the app-crypt/gnupg (https://packages.gentoo.org/packages/app-crypt/gnupg) software. With this package installed, the following command can be used to verify the cryptographic signature in the `.asc` file.

First, download the right set of keys as made available on the signatures page (https://www.gentoo.org/downloads/signatures/):

```
user $ gpg --keyserver hkps://keys.gentoo.org --recv-keys 0xBB572E0E2D182910
```

```
gpg: requesting key 0xBB572E0E2D182910 from hkp server pool.sks-keyservers.net
gpg: key 0xBB572E0E2D182910: "Gentoo Linux Release Engineering (Automated Weekly Release
Key) <releng@gentoo.org>" 1 new signature
gpg: 3 marginal(s) needed, 1 complete(s) needed, classic trust model
gpg: depth: 0  valid:   3  signed:  20  trust: 0-, 0q, 0n, 0m, 0f, 3u
gpg: depth: 1  valid:  20  signed:  12  trust: 9-, 0q, 0n, 9m, 2f, 0u
gpg: next trustdb check due at 2018-09-15
gpg: Total number processed: 1
gpg:         new signatures: 1
```

Alternatively you can use instead the WKD (/wiki/WKD) to download the key:

```
user $ gpg --auto-key-locate=clear,nodefault,wkd --locate-key releng@gentoo.org
```

```
gpg: key 0x9E6438C817072058: public key "Gentoo Linux Release Engineering (Gentoo Linux
Release Signing Key) <releng@gentoo.org>" imported
gpg: key 0xBB572E0E2D182910: public key "Gentoo Linux Release Engineering (Automated Wee
kly Release Key) <releng@gentoo.org>" imported
gpg: Total number processed: 2
gpg:               imported: 2
gpg: public key of ultimately trusted key 0x58497EE51D5D74A5 not found
gpg: public key of ultimately trusted key 0x1F3D03348DB1A3E2 not found
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
pub   dsa1024/0x9E6438C817072058 2004-07-20 [SC] [expires: 2024-01-01]
      D99EAC7379A850BCE47DA5F29E6438C817072058
uid                  [ unknown] Gentoo Linux Release Engineering (Gentoo Linux Release
Signing Key) <releng@gentoo.org>
sub   elg2048/0x0403710E1415B4ED 2004-07-20 [E] [expires: 2024-01-01]
```

Or if using official Gentoo release media, import the key from `/usr/share/openpgp-keys/gentoo-release.asc` (provided by sec-keys/openpgp-keys-gentoo-release (https://packages.gentoo.org/packages/sec-keys/openpgp-keys-gentoo-release)):

```
user $ gpg --import /usr/share/openpgp-keys/gentoo-release.asc
```

```
gpg: directory '/home/larry/.gnupg' created
gpg: keybox '/home/larry/.gnupg/pubring.kbx' created
gpg: key DB6B8C1F96D8BF6D: 2 signatures not checked due to missing keys
gpg: /home/larry/.gnupg/trustdb.gpg: trustdb created
gpg: key DB6B8C1F96D8BF6D: public key "Gentoo ebuild repository signing key (Automated S
igning Key) <infrastructure@gentoo.org>" imported
gpg: key 9E6438C817072058: 3 signatures not checked due to missing keys
gpg: key 9E6438C817072058: public key "Gentoo Linux Release Engineering (Gentoo Linux Re
lease Signing Key) <releng@gentoo.org>" imported
gpg: key BB572E0E2D182910: 1 signature not checked due to a missing key
gpg: key BB572E0E2D182910: public key "Gentoo Linux Release Engineering (Automated Weekl
y Release Key) <releng@gentoo.org>" imported
gpg: key A13D0EF1914E7A72: 1 signature not checked due to a missing key
gpg: key A13D0EF1914E7A72: public key "Gentoo repository mirrors (automated git signing
key) <repomirrorci@gentoo.org>" imported
gpg: Total number processed: 4
gpg:               imported: 4
gpg: no ultimately trusted keys found
```

Next verify the cryptographic signature:

```
user $ gpg --verify install-amd64-minimal-20141204.iso.asc
```

```
gpg: Signature made Fri 05 Dec 2014 02:42:44 AM CET
gpg:                using RSA key 0xBB572E0E2D182910
gpg: Good signature from "Gentoo Linux Release Engineering (Automated Weekly Release Ke
y) <releng@gentoo.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 13EB BDBE DE7A 1277 5DFD  B1BA BB57 2E0E 2D18 2910
```

To be absolutely certain that everything is valid, verify the fingerprint shown with the fingerprint on the Gentoo signatures page (https://www.gentoo.org/downloads/signatures/).

Burning a disk

Of course, with just an ISO file downloaded, the Gentoo Linux installation cannot be started. The ISO file needs to be burned on a CD to boot from, and in such a way that its *content* is burned on the CD, not just the file itself. Below a few common methods are described - a more elaborate set of instructions can be found in Our FAQ on burning an ISO file (/wiki/FAQ#How_do_I_burn_an_ISO_file.3F).

Burning with Microsoft Windows 7 and above

Versions of Microsoft Windows 7 and above can both mount and burn ISO images to optical media without the requirement for third-party software. Simply insert a burnable disk, browse to the downloaded ISO files, right click the file in Windows Explorer, and select "Burn disk image".

Burning with Linux

The **cdrecord** utility from the package app-cdr/cdrtools (https://packages.gentoo.org/packages/app-cdr/cdrtools) can burn ISO images on Linux.

To burn the ISO file on the CD in the `/dev/sr0` device (this is the first CD device on the system - substitute with the right device file if necessary):

```
user $ cdrecord dev=/dev/sr0 install-amd64-minimal-20141204.iso
```

Users that prefer a graphical user interface can use K3B, part of the kde-apps/k3b (https://packages.gentoo.org/packages/kde-apps/k3b) package. In K3B, go to `Tools` and use `Burn CD Image`.

Booting

Booting the installation media

Once the installation media is ready, it is time to boot it. Insert the media in the system, reboot, and enter the motherboard's firmware user interface. This is usually performed by pressing a keyboard key such as `DEL`, `F1`, `F10`, or `ESC` during the Power-On Self-test (POST) process. The 'trigger' key varies depending on the system and motherboard. If it is not obvious use an internet search engine and do some research using the motherboard's model name as the search keyword. Results should be easy to determine. Once inside the motherboard's firmware menu, change the boot order so that the external bootable media (CD/DVD disks or USB drives) are tried *before* the internal disk devices. Without this change, the system will most likely reboot to the internal disk device, ignoring the external boot media.

Important
When installing Gentoo with the purpose of using the UEFI interface instead of BIOS, it is recommended to boot with UEFI immediately. If not, then it might be necessary to create a bootable UEFI USB stick (or other medium) once before finalizing the Gentoo Linux installation.

If not yet done, ensure that the installation media is inserted or plugged into the system, and reboot. A boot prompt should be shown. At this screen, `Enter` will begin the boot process with the default boot options. To boot the installation media with custom boot options, specify a kernel followed by boot options and then hit `Enter`.

Note
In all likelihood, the default gentoo kernel, as mentioned above, without specifying any of the optional parameters will work just fine. For boot troubleshooting and expert options, continue on with this section. Otherwise, just press `Enter` and skip ahead to Extra hardware configuration (/wiki/Handbook:AMD64/Installation/Media#Extra_hardware_configuration).

At the boot prompt, users get the option of displaying the available kernels (`F1`) and boot options (`F2`). If no choice is made within 15 seconds (either displaying information or using a kernel) then the installation media will fall back to booting from disk. This allows installations to reboot and try out their installed environment without the need to remove the CD from the tray (something well appreciated for remote installations).

Specifying a kernel was mentioned. On the Minimal installation media, only two predefined kernel boot options are provided. The default option is called **gentoo**. The other being the *-nofb* variant; this disables kernel framebuffer support.

The next section displays a short overview of the available kernels and their descriptions:

Kernel choices

gentoo
Default kernel with support for K8 CPUs (including NUMA support) and EM64T CPUs.
gentoo-nofb
Same as *gentoo* but without framebuffer support.
memtest86
Test the local RAM for errors.

Alongside the kernel, boot options help in tuning the boot process further.

Hardware options

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

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

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

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

doapm
This loads APM driver support. This also requires that `acpi=off`.

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

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

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

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

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

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

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

nodmraid
Disables support for device-mapper RAID, such as that used for on-board IDE/SATA RAID controllers.

nofirewire
This disables the loading of Firewire modules. This should only be necessary if your Firewire hardware is causing a problem with booting the CD.

nogpm
This disables gpm console mouse support.

nohotplug
This disables the loading of the hotplug and coldplug init scripts at boot. This is useful for debugging a failing CD or driver.

nokeymap

This disables the keymap selection used to select non-US keyboard layouts.

nolapic

This disables the local APIC on Uniprocessor kernels.

nosata

This disables the loading of Serial ATA modules. This is used if the system is having problems with the SATA subsystem.

nosmp

This disables SMP, or Symmetric Multiprocessing, on SMP-enabled kernels. This is useful for debugging SMP-related issues with certain drivers and motherboards.

nosound

This disables sound support and volume setting. This is useful for systems where sound support causes problems.

nousb

This disables the autoloading of USB modules. This is useful for debugging USB issues.

slowusb

This adds some extra pauses into the boot process for slow USB CDROMs, like in the IBM BladeCenter.

Logical volume/device management

dolvm

This enables support for Linux's Logical Volume Management.

Other options

debug

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

docache

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

doload=X

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

dosshd

Starts sshd on boot, which is useful for unattended installs.

passwd=foo

Sets whatever follows the equals as the root password, which is required for *dosshd* since the root password is by default scrambled.

noload=X

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

nonfs

Disables the starting of portmap/nfsmount on boot.

nox

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

scandelay

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

scandelay=X

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

> **Note**
> The bootable media will check for `no*` options before `do*` options, so that options can be overridden in the exact order specified.

Now boot the media, select a kernel (if the default **gentoo** kernel does not suffice) and boot options. As an example, we boot the **gentoo** kernel, with `dopcmcia` as a kernel parameter:

```
boot: gentoo dopcmcia
```

Next the user will be greeted with a boot screen and progress bar. If the installation is done on a system with a non-US keyboard, make sure to immediately press `Alt` + `F1` to switch to verbose mode and follow the prompt. If no selection is made in 10 seconds the default (US keyboard) will be accepted and the boot process will continue. Once the boot process completes, the user is automatically logged in to the "Live" Gentoo Linux environment as the *root* user, the super user. A root prompt is displayed on the current console, and one can switch to other consoles by pressing `Alt` + `F2`, `Alt` + `F3` and `Alt` + `F4`. Get back to the one started on by pressing `Alt` + `F1`.

Extra hardware configuration

When the Installation medium boots, it tries to detect all the hardware devices and loads the appropriate kernel modules to support the 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 needed by the system. If the PCI auto-detection missed some of the system's hardware, the appropriate kernel modules have to be loaded manually.

In the next example the 8139too module (which supports certain kinds of network interfaces) is loaded:

```
root # modprobe 8139too
```

Optional: User accounts

If other people need access to the installation environment, or there is need to run commands as a non-root user on the installation medium (such as to chat using **irssi** without root privileges for security reasons), then an additional user account needs to be created and the root password set to a strong password.

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

```
root # passwd
```

```
New password: (Enter the new password)
Re-enter password: (Re-enter the password)
```

To create a user account, first enter their credentials, followed by the account's password. The **useradd** and **passwd** commands are used for these tasks.

In the next example, a user called *john* is created:

```
root # useradd -m -G users john
root # passwd john
```

```
New password: (Enter john's password)
Re-enter password: (Re-enter john's password)
```

To switch from the (current) *root* user to the newly created user account, use the **su** command:

```
root # su - john
```

Optional: Viewing documentation while installing

TTYs

To view the Gentoo handbook during the installation, first create a user account as described above. Then press `Alt` + `F2` to go to a new terminal (TTY).

During the installation, the **links** command can be used to browse the Gentoo handbook - of course only from the moment that the Internet connection is working.

```
user $ links https://wiki.gentoo.org/wiki/Handbook:AMD64
```

To go back to the original terminal, press `Alt` + `F1`.

> **Tip**
> When booted to the Gentoo minimal or Gentoo admin environments, seven TTYs will be available. They can be switched by pressing `Alt` then a function key between `F1` - `F7`. It can be useful to switch to a new terminal when waiting for job to complete, to open documentation, etc.

GNU Screen

The Screen (/wiki/Screen) utility is installed by default on official Gentoo installation media. It may be more efficient for the seasoned Linux enthusiast to use **screen** to view installation instructions via split panes rather than the multiple TTY method mentioned above.

Optional: Starting the SSH daemon

To allow other users to access the system during the installation (perhaps to support during an installation, or even do it remotely), a user account needs to be created (as was documented earlier on) and the SSH daemon needs to be started.

To fire up the SSH daemon on an OpenRC init, execute the following command:

```
root # rc-service sshd start
```

> **Note**
> If users log on to the system, they will see a message that the host key for this system needs to be confirmed (through what is called a fingerprint). This behavior is typical and can be expected for initial connections to an SSH server. However, later when the system is set up and someone logs on to the newly created system, the SSH client will warn that the host key has been changed. This is because the user now logs on to - for SSH - a different server (namely the freshly installed Gentoo system rather than the live environment that the installation is currently using). Follow the instructions given on the screen then to replace the host key on the client system.

To be able to use sshd, the network needs to function properly. Continue with the chapter on Configuring the network (/wiki/Handbook:AMD64/Installation/Networking).

Automatic network detection

Maybe it just works?

If the system is plugged into an Ethernet network with a DHCP server, it is very likely that the networking configuration has already been set up automatically. If so, then the many included network-aware commands on the installation media such as **ssh**, **scp**, **ping**, **irssi**, **wget**, and **links**, among others, will work immediately.

Determine interface names

ifconfig command

If networking has been configured, the **ifconfig** command should list one or more network interfaces (besides `lo`). In the example below `eth0` shows up:

root # `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
```

As a result of the shift towards predictable network interface names (http://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/), the interface name on the system can be quite different from the old `eth0` naming convention. Recent installation media might show regular network interfaces names like `eno0`, `ens1`, or `enp5s0`. Look for the interface in the **ifconfig** output that has an IP address related to the local network.

Tip
If no interfaces are displayed when the standard **ifconfig** command is used, try using the same command with the `-a` option. This option forces the utility to show all network interfaces detected by the system whether they be in an up or down state. If **ifconfig -a** produces no results then the hardware is faulty or the driver for the interface has not been loaded into the kernel. Both situations reach beyond the scope of this Handbook. Contact #gentoo (ircs://irc.libera.chat/#gentoo) (webchat (https://web.libera.chat/#gentoo)) for support.

ip command

As an alternative to **ifconfig**, the **ip** command can be used to determine interface names. The following example shows the output of **ip addr** (of another system so the information shown is different from the previous example):

root # `ip addr`

```
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group defa
ult qlen 1000
    link/ether e8:40:f2:ac:25:7a brd ff:ff:ff:ff:ff:ff
    inet 10.0.20.77/22 brd 10.0.23.255 scope global eno1
       valid_lft forever preferred_lft forever
    inet6 fe80::ea40:f2ff:feac:257a/64 scope link
       valid_lft forever preferred_lft forever
```

The interface name in the above example directly follows the number; it is `eno1`.

In the remainder of this document, the handbook will assume that the operating network interface is called `eth0`.

Optional: Configure any proxies

If the Internet is accessed through a proxy, then it is necessary to set up proxy information during the installation. It is very easy to define a proxy: just define a variable which contains the proxy server information.

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

To set up an HTTP proxy (for HTTP and HTTPS traffic):

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

To set up an FTP proxy:

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

To set up an RSYNC proxy:

```
root # export RSYNC_PROXY="proxy.gentoo.org:8080"
```

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

CODE **Adding username/password to the proxy variable**

```
http://username:password@proxy.gentoo.org:8080
```

Testing the network

Try pinging your ISP's DNS server (found in `/etc/resolv.conf`) and a web site of choice. This ensures that the network is functioning properly and that the network packets are reaching the net, DNS name resolution is working correctly, etc.

```
root # ping -c 3 www.gentoo.org
```

If this all works, then the remainder of this chapter can be skipped to jump right to the next step of the installation instructions (Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks)).

Automatic network configuration

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

If the installation medium does not contain any of these tools, continue with the Manual network configuration (/wiki/Handbook:AMD64/Installation/Networking#Manual_network_configuration).

- Regular Ethernet users should continue with Default: Using net-setup (/wiki/Handbook:AMD64/Installation/Networking#Default:_Using_net-setup)

- ADSL users should continue with Alternative: Using PPP (/wiki/Handbook:AMD64/Installation/Networking#Alternative:_Using_PPP)
- PPTP users should continue with Alternative: Using PPTP (/wiki/Handbook:AMD64/Installation/Networking#Alternative:_Using_PPTP)

Default: Using net-setup

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

```
root # net-setup eth0
```

net-setup will ask some questions about the network environment. When all is done, the network connection should work. Test the network connection as stated before. If the tests are positive, congratulations! Skip the rest of this section and continue with Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

If the network still doesn't work, continue with Manual network configuration (/wiki/Handbook:AMD64/Installation/Networking#Manual_network_configuration).

Alternative: Using PPP

Assuming PPPoE is needed to connect to the Internet, the installation CD (any version) has made things easier by including ppp. Use the provided **pppoe-setup** script to configure the connection. During the setup the Ethernet device that is connected to your ADSL modem, the username and password, the IPs of the DNS servers and if a basic firewall is needed or not will be asked.

```
root # pppoe-setup
root # pppoe-start
```

If something goes wrong, double-check that the username and password are correct by looking at `etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets` and make sure to use the right Ethernet device. If the Ethernet device does not exist, the appropriate network modules need to be loaded. In that case continue with Manual network configuration (/wiki/Handbook:AMD64/Installation/Networking#Manual_network_configuration) as it will explain how to load the appropriate network modules there.

If everything worked, continue with Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

Alternative: Using PPTP

If PPTP support is needed, use **pptpclient** which is provided by the installation CDs. But first make sure that the configuration is correct. Edit `/etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets` so it contains the correct username/password combination:

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

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

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

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

```
root # pptp <server ipv4 address>
```

Now continue with Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

Manual network configuration

Loading the appropriate network kernel modules

When the Installation CD boots, it tries to detect all the hardware devices and loads the appropriate kernel modules (drivers) to support the 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 needed to communicate properly with the present network hardware.

If **net-setup** or **pppoe-setup** failed, then it is possible that the network card wasn't found immediately. This means users may have to load the appropriate kernel modules manually.

To find out what kernel modules are provided for networking, use the **ls** command:

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

If a driver is found for the network device, use **modprobe** to load the kernel module. For instance, to load the pcnet32 module:

```
root # modprobe pcnet32
```

To check if the network card is now detected, use **ifconfig**. A detected network card would result in something like this (again, eth0 here is just an example):

```
root # ifconfig eth0
```

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

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

```
root # ifconfig eth0
```

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

The available network interface names on the system can be listed through the /sys file system:

```
root # ls /sys/class/net
```

```
dummy0  eth0  lo  sit0  tap0  wlan0
```

In the above example, 6 interfaces are found. The eth0 one is most likely the (wired) Ethernet adapter whereas wlan0 is the wireless one.

Assuming that the network card is now detected, retry **net-setup** or **pppoe-setup** again (which should work now), but for the hardcore people we explain how to configure the network manually as well.

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

- Using DHCP (/wiki/Handbook:AMD64/Installation/Networking#Using_DHCP) for automatic IP retrieval
- Preparing for wireless access (/wiki/Handbook:AMD64/Installation/Networking#Preparing_for_wireless_access) if a wireless network is used

- Understanding network terminology (/wiki/Handbook:AMD64/Installation/Networking#Understanding_network_terminology) explains the basics about networking
- Using ifconfig and route (/wiki/Handbook:AMD64/Installation/Networking#Using_ifconfig_and_route) explains how to set up 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 a DHCP server is in the network (or if the ISP provider provides a DHCP service). To have a network interface receive this information automatically, use **dhcpcd**:

```
root # dhcpcd eth0
```

Some network administrators require that the hostname and domainname provided by the DHCP server is used by the system. In that case, use:

```
root # dhcpcd -HD eth0
```

If this works (try pinging some Internet server, like Google's 8.8.8.8 or Cloudflare's 1.1.1.1), then everything is set and ready to continue. Skip the rest of this section and continue with Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

Preparing for wireless access

Note
Support for the **iw** command might be architecture-specific. If the command is not available see if the net-wireless/iw (https://packages.gentoo.org/packages/net-wireless/iw) package is available for the current architecture. The **iw** command will be unavailable unless the net-wireless/iw (https://packages.gentoo.org/packages/net-wireless/iw) package has been installed.

When using a wireless (802.11) card, the wireless settings need to be configured before going any further. To see the current wireless settings on the card, one can use **iw**. Running **iw** might show something like:

```
root # iw dev wlp9s0 info
```

```
Interface wlp9s0
        ifindex 3
        wdev 0x1
        addr 00:00:00:00:00:00
        type managed
        wiphy 0
        channel 11 (2462 MHz), width: 20 MHz (no HT), center1: 2462 MHz
        txpower 30.00 dBm
```

To check for a current connection:

```
root # iw dev wlp9s0 link
```

```
Not connected.
```

or

```
root # iw dev wlp9s0 link
```

```
Connected to 00:00:00:00:00:00 (on wlp9s0)
        SSID: GentooNode
        freq: 2462
        RX: 3279 bytes (25 packets)
        TX: 1049 bytes (7 packets)
        signal: -23 dBm
        tx bitrate: 1.0 MBit/s
```

Note
Some wireless cards may have a device name of `wlan0` or `ra0` instead of wlp9s0. Run **`ip link`** to determine the correct device name.

For most users, there are only two settings needed to connect, the ESSID (aka wireless network name) and, optionally, the WEP key.

- First, ensure the interface is active:

```
root # ip link set dev wlp9s0 up
```

- To connect to an open network with the name *GentooNode*:

```
root # iw dev wlp9s0 connect -w GentooNode
```

- To connect with a hex WEP key, prefix the key with `d:` :

```
root # iw dev wlp9s0 connect -w GentooNode key 0:d:1234123412341234abcd
```

- To connect with an ASCII WEP key:

```
root # iw dev wlp9s0 connect -w GentooNode key 0:some-password
```

Note
If the wireless network is set up with WPA or WPA2, then **`wpa_supplicant`** needs to be used. For more information on configuring wireless networking in Gentoo Linux, please read the Wireless networking chapter (/wiki/Handbook:AMD64/Networking/Wireless) in the Gentoo Handbook.

Confirm the wireless settings by using **`iw dev wlp9s0 link`**. Once wireless is working, continue configuring the IP level networking options as described in the next section (Understanding network terminology (/wiki/Handbook:AMD64/Installation/Networking#Understanding_network_terminology)) or use the **`net-setup`** tool as described previously.

Understanding network terminology

Note
If the IP address, broadcast address, netmask and nameservers are known, then skip this subsection and continue with Using **`ifconfig`** and **`route`** (/wiki/Handbook:AMD64/Installation/Networking#Using_ifconfig_and_route).

If all of the above fails, the network will need to be configured manually. This is not difficult at all. However, some knowledge of network terminology and basic concepts might be necessary. After reading this section, users will know what a gateway is, what a netmask serves for, how a broadcast address is formed and why

systems need nameservers.

In a network, hosts are identified by their IP address (Internet Protocol address). Such an address is perceived as a combination of four numbers between 0 and 255. Well, at least when using IPv4 (IP version 4). In reality, such an IPv4 address consists of 32 bits (ones and zeros). Let's view an example:

CODE **Example of an IPv4 address**

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

Note
The successor of IPv4, IPv6, uses 128 bits (ones and zeros). In this section, the focus is on IPv4 addresses.

Such an IP address is unique to a host as far as all accessible networks are concerned (i.e. every host that one wants to be able to reach must have a unique IP address). In order to distinguish between hosts inside and outside a network, the IP address is divided in two parts: the network part and the host part.

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

CODE **Example of network/host separation**

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

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

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

CODE **Broadcast address**

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

To be able to surf on the Internet, each computer in the network 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).

Previously we stated that every host has its own IP address. To be able to reach this host by a name (instead of an IP address) we 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, the necessary name servers need to be defined in `/etc/resolv.conf`.

In some cases, the gateway also serves as a nameserver. Otherwise the nameservers provided by the ISP need to be entered in this file.

To summarize, the following information is needed before continuing:

Network item	Example
The system IP address	192.168.0.2
Netmask	255.255.255.0
Broadcast	192.168.0.255
Gateway	192.168.0.1
Nameserver(s)	195.130.130.5, 195.130.130.133

Using ifconfig and route

Employing tools from the sys-apps/net-tools (https://packages.gentoo.org/packages/sys-apps/net-tools) package, setting up the network manually generally consists of three steps:

1. Assign an IP address using the **ifconfig** command.
2. Set up routing to the gateway using the **route** command.
3. Finish up by placing valid nameserver IPs in the `/etc/resolv.conf` file.

To assign an IP address, the IP address, broadcast address, and netmask are needed. Execute the following command, substituting *${IP_ADDR}* with the target IP address, *${BROADCAST}* with the target broadcast address, and *${NETMASK}* with the target netmask:

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

To configure routing using **route**, substitute the *${GATEWAY}* value with the appropriate gateway IP address:

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

Now open the `/etc/resolv.conf` file using a text editor:

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

Fill in the nameserver(s) using the following as a template substituting *${NAMESERVER1}* and *${NAMESERVER2}* with nameserver IP addresses as necessary. More than one nameserver can be added:

FILE `/etc/resolv.conf` **Default resolv.conf template**

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

Now test the network by pinging an Internet server (like Google's 8.8.8.8 or Cloudflare's 1.1.1.1). Once connected, continue with Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks).

Introduction to block devices

Block devices

Let's take a good look at disk-oriented aspects of Gentoo Linux and Linux in general, including block devices, partitions, and Linux filesystems. Once the ins and outs of disks are understood, partitions and filesystems can be established for installation.

To begin, let's look at block devices. SCSI and Serial ATA drives are both labeled under device handles such as: `/dev/sda`, `/dev/sdb`, `/dev/sdc`, etc. On more modern machines, PCI Express based NVMe solid state disks have device handles such as `/dev/nvme0n1`, `/dev/nvme0n2`, etc.

The following table will help readers determine where to find a certain type of block device on the system:

Type of device	Default device handle	Editorial notes and considerations
IDE, SATA, SAS, SCSI, or USB flash	`/dev/sda`	Found on hardware from roughly 2007 until the present, this device handle is perhaps the most commonly used in Linux. These types of devices can be connected via the SATA bus (https://en.wikipedia.org/wiki/Serial_ATA), SCSI (https://en.wikipedia.org/wiki/SCSI), USB (https://en.wikipedia.org/wiki/USB) bus as block storage. As example, the first partition on the first SATA device is called `/dev/sda1`.
NVM Express (NVMe)	`/dev/nvme0n1`	The latest in solid state technology, NVMe (https://en.wikipedia.org/wiki/NVM_Express) drives are connected to the PCI Express bus and have the fastest transfer block speeds on the market. Systems from around 2014 and newer may have support for NVMe hardware. The first partition on the first NVMe device is called `/dev/nvme0n1p1`.
MMC, eMMC, and SD	`/dev/mmcblk0`	embedded MMC (https://en.wikipedia.org/wiki/MultiMediaCard#eMMC) devices, SD cards, and other types of memory cards can be useful for data storage. That said, many systems may not permit booting from these types of devices. It is suggested to **not** use these devices for active Linux installations; rather consider using them to transfer files, which is their design goal. Alternatively they could be useful for short-term backups.

The block devices above represent an abstract interface to the disk. User programs can use these block devices to interact with the disk without worrying about whether the drives are SATA, SCSI, or something else. The program can simply address the storage on the disk as a bunch of contiguous, randomly-accessible 4096-byte (4K) blocks.

Partition tables

Although it is theoretically possible to use a raw, unpartitioned disk to house a Linux system (when creating a btrfs RAID for example), this is almost never done in practice. Instead, disk block devices are split up into smaller, more manageable block devices. On **amd64** systems, these are called partitions. There are currently two standard partitioning technologies in use: MBR (sometimes also called DOS disklabel) and GPT; these are tied to the two boot process types: legacy BIOS boot and UEFI.

GUID Partition Table (GPT)

The *GUID Partition Table (GPT)* setup (also called GPT disklabel) uses 64-bit identifiers for the partitions. The location in which it stores the partition information is much bigger than the 512 bytes of the MBR partition table (DOS disklabel), which means there is practically no limit on the amount of partitions for a GPT disk. Also the *size* of a partition is bounded by a much greater limit (almost 8 ZiB - yes, zebibytes).

When a system's software interface between the operating system and firmware is UEFI (instead of BIOS), GPT is almost mandatory as compatibility issues will arise with DOS disklabel.

GPT also takes advantage of checksumming and redundancy. It carries CRC32 checksums to detect errors in the header and partition tables and has a backup GPT at the end of the disk. This backup table can be used to recover damage of the primary GPT near the beginning of the disk.

Important
There are a few caveats regarding GPT:

- Using GPT on a BIOS-based computer works, but then one cannot dual-boot with a Microsoft Windows operating system. The reason is that Microsoft Windows will boot in UEFI mode if it detects a GPT partition label.
- Some buggy (old) motherboard firmware configured to boot in BIOS/CSM/legacy mode might also have problems with booting from GPT labeled disks.

Master boot record (MBR) or DOS boot sector

The *Master boot record (https://en.wikipedia.org/wiki/Master_boot_record)* boot sector (also called DOS boot sector or DOS disk label) was first introduced in 1983 with PC DOS 2.x. MBR uses 32-bit identifiers for the start sector and length of the partitions, and supports three partition types: primary, extended, and logical. Primary partitions have their information stored in the master boot record itself - a very small (usually 512 bytes) location at the very beginning of a disk. Due to this small space, only four primary partitions are supported (for instance, `/dev/sda1` to `/dev/sda4`).

In order to support more partitions, one of the primary partitions in the MBR can be marked as an *extended* partition. This partition can then contain additional logical partitions (partitions within a partition).

Important
Although still supported by most motherboard manufacturers, MBR boot sectors and their associated partitioning limitations are considered legacy. Unless working with hardware that is pre-2010, it best to partition a disk with GUID Partition Table (https://en.wikipedia.org/wiki/GUID_Partition_Table). Readers who must proceed with setup type should knowingly acknowledge the following information:

- Most post-2010 motherboards consider using MBR boot sectors a legacy (supported, but not ideal) boot mode.
- Due to using 32-bit identifiers, partition tables in the MBR cannot address storage space that is larger than 2 TiBs in size.
- Unless an extended partition is created, MBR supports a maximum of four partitions.

- This setup does not provide a backup boot sector, so if something overwrites the partition table, all partition information will be lost.

That said, MBR and BIOS boot is still frequently used in virtualized cloud environments such as AWS.

The Handbook authors suggest using GPT whenever possible for Gentoo installations.

Advanced storage

The **amd64** Installation CDs provide support for Logical Volume Manager (LVM). LVM increases the flexibility offered by the partitioning setup. It allows to combine partitions and disks into volume groups and define RAID groups or caches on fast SSDs for slow HDs. The installation instructions below will focus on "regular" partitions, but it is good to know LVM is supported if that route is desired. Visit the LVM (/wiki/LVM) article for more details. Newcomers beware: although fully supported, LVM is outside the scope of this guide.

Default partitioning scheme

Throughout the remainder of the handbook, we will discuss and explain two cases: 1) GPT partition table and UEFI boot, and 2) MBR partition table and legacy BIOS boot. While it is possible to mix and match, that goes beyond the scope of this manual. As already stated above, installations on modern hardware should use GPT partition table and UEFI boot; as an exception from this rule, MBR and BIOS boot is still frequently used in virtualized (cloud) environments.

The following partitioning scheme will be used as a simple example layout:

Partition	Filesystem	Size	Description
/dev/sda1	fat32 (UEFI) or xfs (BIOS - aka Legacy boot)	1GB	Boot/EFI system partition
/dev/sda2	(swap)	RAM size * 2	Swap partition
/dev/sda3	xfs	Rest of the disk	Root partition

If this suffices as information, the advanced reader can directly skip ahead to the actual partitioning.

Both **fdisk** and **parted** are partitioning utilities. **fdisk** is well known, stable, and recommended for the MBR partition layout. **parted** was one of the first Linux block device management utilities to support GPT partitions, and provides an alternative. Here, **fdisk** is used since it has a better text-based user interface.

Before going to the creation instructions, the first set of sections will describe in more detail how partitioning schemes can be created and mention some common pitfalls.

Designing a partition scheme

How many partitions and how big?

The design of disk partition layout is highly dependent on the demands of the system and the file system(s) applied to the device. If there are lots of users, then it is advised to have /home on a separate partition which will increase security and make backups and other types of maintenance easier. If Gentoo is being installed to perform as a mail server, then /var should be a separate partition as all mails are stored inside the /var directory. Game servers may have a separate /opt partition since most gaming server software is installed therein. The reason for these recommendations is similar to the /home directory: security, backups, and maintenance.

In most situations on Gentoo, `/usr` and `/var` should be kept relatively large in size. `/usr` hosts the majority of applications available on the system and the Linux kernel sources (under `/usr/src`). By default, `/var` hosts the Gentoo ebuild repository (located at `/var/db/repos/gentoo`) which, depending on the file system, generally consumes around 650 MiB of disk space. This space estimate *excludes* the `/var/cache/distfiles` and `/var/cache/binpkgs` directories, which will gradually fill with source files and (optionally) binary packages respectively as they are added to the system.

How many partitions and how big very much depends on considering the trade-offs and choosing the best option for the circumstance. Separate partitions or volumes have the following advantages:

- Choose the best performing filesystem for each partition or volume.
- The 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 realized 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 certain disadvantages as well:

- If not configured properly, the system might have lots of free space on one partition and little free space on another.
- A separate partition for `/usr/` may require the administrator to boot with an initramfs to mount the partition before other boot scripts start. Since the generation and maintenance of an initramfs is beyond the scope of this handbook, **we recommend that newcomers do not use a separate partition for `/usr/`**.
- There is also a 15-partition limit for SCSI and SATA unless the disk uses GPT labels.

> **Note**
> Installations that intend to use systemd as the service and init system must have the `/usr` directory available at boot, either as part of the root filesystem or mounted via an initramfs.

What about swap space?

There is no perfect value for swap space size. The purpose of the space is to provide disk storage to the kernel when internal memory (RAM) is under pressure. A swap space allows for the kernel to move memory pages that are not likely to be accessed soon to disk (swap or page-out), which will free memory in RAM for the current task. Of course, if the pages swapped to disk are suddenly needed, they will need to be put back in memory (page-in) which will take considerably longer than reading from RAM (as disks are very slow compared to internal memory).

When a system is not going to run memory intensive applications or has lots of RAM available, then it probably does not need much swap space. However do note in case of hibernation that swap space is used to store *the entire contents of memory* (likely on desktop and laptop systems rather than on server systems). If the system requires support for hibernation, then swap space larger than or equal to the amount of memory is necessary.

As a general rule, the swap space size is recommended to be twice the internal memory (RAM). For systems with multiple hard disks, it is wise to create one swap partition on each disk so that they can be utilized for parallel read/write operations. The faster a disk can swap, the faster the system will run when data in swap space must be accessed. When choosing between rotational and solid state disks, it is better for performance to put swap on the SSD. Also, swap files can be used as an alternative to swap partitions; this is mostly interesting for systems with very limited disk space.

What is the EFI System Partition (ESP)?

When installing Gentoo on a system that uses UEFI to boot the operating system (instead of BIOS), then it is important that an EFI System Partition (ESP) is created. The instructions below contain the necessary pointers to correctly handle this operation. **The EFI system partition is not required when booting in BIOS/Legacy mode.**

The ESP *must* be a FAT variant (sometimes shown as *vfat* on Linux systems). The official UEFI specification (http://www.uefi.org/sites/default/files/resources/UEFI%202_5.pdf) denotes FAT12, 16, or 32 filesystems will be recognized by the UEFI firmware, although FAT32 is recommended for the ESP. After partitioning, format the ESP accordingly:

```
root # mkfs.fat -F 32 /dev/sda1
```

Important
If the ESP is not formatted with a FAT variant, the system's UEFI firmware will not find the bootloader (or Linux kernel) and will most likely be unable to boot the system!

What is the BIOS boot partition?

A BIOS boot partition is only needed when combining a GPT partition layout with GRUB2 in BIOS/Legacy boot mode. **It is not required when booting in EFI/UEFI mode, and also not required when using a MBR table.** It is a very small (1 to 2 MB) partition in which boot loaders like GRUB2 can put additional data that doesn't fit in the allocated storage. It will not be used in this guide.

Partitioning the disk with GPT for UEFI

The following parts explain how to create the example partition layout for a GPT / UEFI boot installation using **fdisk**. The example partition layout was mentioned earlier:

Partition	Description
/dev/sda1	EFI system (and boot) partition
/dev/sda2	Swap partition
/dev/sda3	Root partition

Change the partition layout according to personal preference.

Viewing the current partition layout

fdisk is a popular and powerful tool to split a disk into partitions. Fire up **fdisk** against the disk (in our example, we use /dev/sda):

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

Use the `p` key to display the disk's current partition configuration:

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

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 21AAD8CF-DB67-0F43-9374-416C7A4E31EA

Device        Start      End  Sectors  Size Type
/dev/sda1      2048   526335   524288    1G EFI System
/dev/sda2    526336  2623487  2097152    1G Linux swap
/dev/sda3   2623488 19400703 16777216    8G Linux filesystem
/dev/sda4  19400704 60549086 41148383 19.6G Linux filesystem
```

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

Creating a new disklabel / removing all partitions

Type `g` to create a new GPT disklabel on the disk; this will remove all existing partitions.

Command (m for help): g

```
Created a new GPT disklabel (GUID: 87EA4497-2722-DF43-A954-368E46AE5C5F).
```

For an existing GPT disklabel (see the output of `p` above), alternatively consider removing the existing partitions one by one from the disk. Type `d` to delete a partition. For instance, to delete an existing `/dev/sda1`:

Command (m for help): d

```
Partition number (1-4): 1
```

The partition has now been scheduled for deletion. It will no longer show up when printing the list of partitions (`p`), but it will not be erased until the changes have been saved. This allows users to abort the operation if a mistake was made - in that case, type `q` immediately and hit `Enter` and the partition will not be deleted.

Repeatedly type `p` to print out a partition listing and then type `d` and the number of the partition to delete it. Eventually, the partition table will be empty:

Command (m for help): p

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 87EA4497-2722-DF43-A954-368E46AE5C5F
```

Now that the in-memory partition table is empty, we're ready to create the partitions.

Creating the EFI system partition (ESP)

> **Note**
> A smaller ESP is possible but not recommended, especially given it may be shared with other OSes.

First create a small EFI system partition, which will also be mounted as /boot. Type `n` to create a new partition, followed by `1` to select the first partition. When prompted for the first sector, make sure it starts from 2048 (which may be needed for the boot loader) and hit `Enter`. When prompted for the last sector, type +1G to create a partition 1 GByte in size:

Command (m for help): n

```
Partition number (1-128, default 1): 1
First sector (2048-60549086, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-60549086, default 60549086): +1G

Created a new partition 1 of type 'Linux filesystem' and of size 1 GiB.
```

Mark the partition as EFI system partition:

Command (m for help): t

```
Selected partition 1
Partition type (type L to list all types): 1
Changed type of partition 'Linux filesystem' to 'EFI System'.
```

Creating the swap partition

Next, to create the swap partition, type `n` to create a new partition, then type `2` to create the second partition, `/dev/sda2`. When prompted for the first sector, hit `Enter`. When prompted for the last sector, type +4G (or any other size needed for the swap space) to create a partition 4GB in size.

Command (m for help): n

```
Partition number (2-128, default 2):
First sector (526336-60549086, default 526336):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (526336-60549086, default 60549086): +4G

Created a new partition 2 of type 'Linux filesystem' and of size 4 GiB.
```

After all this is done, type `t` to set the partition type, `2` to select the partition just created and then type in *19* to set the partition type to "Linux Swap".

Command (m for help): t

```
Partition number (1,2, default 2): 2
Partition type (type L to list all types): 19

Changed type of partition 'Linux filesystem' to 'Linux swap'.
```

Creating the root partition

Finally, to create the root partition, type `n` to create a new partition. Then type `3` to create the third partition, `/dev/sda3`. When prompted for the first sector, hit `Enter`. When prompted for the last sector, hit `Enter` to create a partition that takes up the rest of the remaining space on the disk. After completing these steps, typing `p` should display a partition table that looks similar to this:

Command (m for help): p

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 87EA4497-2722-DF43-A954-368E46AE5C5F

Device       Start      End  Sectors  Size Type
/dev/sda1     2048   526335   524288    1G EFI System
/dev/sda2   526336  8914943  8388608    4G Linux swap
/dev/sda3  8914944 60549086 51634143 24.6G Linux filesystem
```

Saving the partition layout

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

Command (m for help): w

With the partitions created, it is now time to put filesystems on them.

Partitioning the disk with MBR for BIOS / legacy boot

The following explains how to create the example partition layout for a MBR / BIOS legacy boot installation. The example partition layout mentioned earlier is now:

Partition	Description
/dev/sda1	Boot partition
/dev/sda2	Swap partition
/dev/sda3	Root partition

Change the partition layout according to personal preference.

Viewing the current partition layout

Fire up **fdisk** against the disk (in our example, we use `/dev/sda`):

root # fdisk /dev/sda

Use the `p` key to display the disk's current partition configuration:

Command (m for help): p

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 21AAD8CF-DB67-0F43-9374-416C7A4E31EA

Device        Start      End  Sectors  Size Type
/dev/sda1      2048   526335   524288    1G EFI System
/dev/sda2    526336  2623487  2097152    1G Linux swap
/dev/sda3   2623488 19400703 16777216    8G Linux filesystem
/dev/sda4  19400704 60549086 41148383 19.6G Linux filesystem
```

This particular disk was until now configured to house two Linux filesystems (each with a corresponding partition listed as "Linux") as well as a swap partition (listed as "Linux swap"), using a GPT table.

Creating a new disklabel / removing all partitions

Type `o` to create a new MBR disklabel (here also named DOS disklabel) on the disk; this will remove all existing partitions.

Command (m for help): o

```
Created a new DOS disklabel with disk identifier 0xe04e67c4.
The device contains 'gpt' signature and it will be removed by a write command. See fdisk
(8) man page and --wipe option for more details.
```

For an existing DOS disklabel (see the output of `p` above), alternatively consider removing the existing partitions one by one from the disk. Type `d` to delete a partition. For instance, to delete an existing `/dev/sda1`:

Command (m for help): d

```
Partition number (1-4): 1
```

The partition has now been scheduled for deletion. It will no longer show up when printing the list of partitions (`p`), but it will not be erased until the changes have been saved. This allows users to abort the operation if a mistake was made - in that case, type `q` immediately and hit `Enter` and the partition will not be deleted.

Repeatedly type `p` to print out a partition listing and then type `d` and the number of the partition to delete it. Eventually, the partition table will be empty:

Command (m for help): p

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe04e67c4
```

Now we're ready to create the partitions.

Creating the boot partition

First, create a small partition which will be mounted as /boot. Type `n` to create a new partition, followed by `p` for a primary partition and `1` to select the first primary partition. When prompted for the first sector, make sure it starts from 2048 (which may be needed for the boot loader) and hit `Enter`. When prompted for the last sector, type +1G to create a partition 1 GB in size:

Command (m for help): n

```
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-60549119, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-60549119, default 60549119): +1G

Created a new partition 1 of type 'Linux' and of size 1 GiB.
```

Creating the swap partition

Next, to create the swap partition, type `n` to create a new partition, then `p`, then type `2` to create the second primary partition, `/dev/sda2`. When prompted for the first sector, hit `Enter`. When prompted for the last sector, type +4G (or any other size needed for the swap space) to create a partition 4GB in size.

Command (m for help): n

```
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (526336-60549119, default 526336):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (526336-60549119, default 60549119): +4G

Created a new partition 2 of type 'Linux' and of size 4 GiB.
```

After all this is done, type `t` to set the partition type, `2` to select the partition just created and then type in *82* to set the partition type to "Linux Swap".

Command (m for help): t

```
Partition number (1,2, default 2): 2
Hex code (type L to list all codes): 82

<!--T:179-->
Changed type of partition 'Linux' to 'Linux swap / Solaris'.
```

Creating the root partition

Finally, to create the root partition, type `n` to create a new partition. Then type `p` and `3` to create the third primary partition, `/dev/sda3`. When prompted for the first sector, hit `Enter`. When prompted for the last sector, hit `Enter` to create a partition that takes up the rest of the remaining space on the disk. After

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

Command (m for help): p

```
Disk /dev/sda: 28.89 GiB, 31001149440 bytes, 60549120 sectors
Disk model: DataTraveler 2.0
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe04e67c4

Device     Boot   Start      End  Sectors  Size Id Type
/dev/sda1          2048   526335   524288    1G 83 Linux
/dev/sda2        526336  8914943  8388608    4G 82 Linux swap / Solaris
/dev/sda3       8914944 60549119 51634176 24.6G 83 Linux
```

Saving the partition layout

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

Command (m for help): w

Now it is time to put filesystems on the partitions.

Creating file systems

⚠ Warning
If using an SSD or NVMe drive, please check if it needs a firmware upgrade. Some Intel SSDs in particular (600p and 6000p) require a firmware upgrade for critical bug fixes (https://bugzilla.redhat.com/show_bug.cgi?id=1402533) avoid data corruption induced by XFS I/O usage patterns (though not through any fault of the filesystem). **smartctl** can help check the model and firmware version.

Introduction

Now that the partitions have been created, it is time to place a filesystem on them. In the next section the various file systems that Linux supports are described. Readers that already know which filesystem to use can continue with Applying a filesystem to a partition (/wiki/Handbook:AMD64/Installation/Networking#Applying_a_filesystem_to_a_partition). The others should read on to learn about the available filesystems...

Filesystems

Linux supports several dozen filesystems, although many of them are only wise to deploy for specific purposes. Only certain filesystems may be found stable on the amd64 architecture - it is advised to read up on the filesystems and their support state before selecting a more experimental one for important partitions. **XFS is the recommended all-purpose, all-platform filesystem.** The below is a non-exhaustive list:

btrfs (/wiki/Btrfs)
Newer generation filesystem. Provides advanced features like snapshotting, self-healing through checksums, transparent compression, subvolumes, and integrated RAID. Kernels prior to 5.4.y are not guaranteed to be

safe to use with btrfs in production because fixes for serious issues are only present in the more recent releases of the LTS kernel branches. RAID 5/6 and quota groups unsafe on all versions of btrfs.

ext4 (/wiki/Ext4)

Ext4 is a reliable, all-purpose all-platform filesystem, although it lacks modern features like reflinks.

f2fs (/wiki/F2fs)

The Flash-Friendly File System was originally created by Samsung for the use with NAND flash memory. It is a decent choice when installing Gentoo to microSD cards, USB drives, or other flash-based storage devices.

XFS (/wiki/XFS)

Filesystem with metadata journaling which comes with a robust feature-set and is optimized for scalability. It has been continuously upgraded to include modern features. The only downside is that XFS partitions cannot yet be shrunk, although this is being worked on. XFS notably supports reflinks and Copy on Write (CoW) which is particularly helpful on Gentoo systems because of the amount of compiles users complete. XFS is the recommended modern all-purpose all-platform filesystem. Requires a partition to be at least 300MB.

VFAT (/wiki/VFAT)

Also known as FAT32, is supported by Linux but does not support standard UNIX permission settings. It is mostly used for interoperability/interchange with other operating systems (Microsoft Windows or Apple's macOS) but is also a necessity for some system bootloader firmware (like UEFI). Users of UEFI systems will need an EFI System Partition (/wiki/EFI_System_Partition) formatted with VFAT in order to boot.

NTFS (/wiki/NTFS)

This "New Technology" filesystem is the flagship filesystem of Microsoft Windows since Windows NT 3.1. Similarly to VFAT, it does not store UNIX permission settings or extended attributes necessary for BSD or Linux to function properly, therefore it should not be used as a root filesystem for most cases. It should *only* be used for interoperability or data interchange with Microsoft Windows systems (note the emphasis on *only*).

More extensive information on filesystems can be found in the community maintained Filesystem article (/wiki/Filesystem).

Applying a filesystem to a partition

> **Note**
> Please make sure to emerge the relevant package for the chosen filesystem later on in the handbook, before rebooting at the end of the install process.

To create a filesystem on a partition or volume, there are user space utilities available for each possible filesystem. Click the filesystem's name in the table below for additional information on each filesystem:

Filesystem	Creation command	On minimal CD?	Package
btrfs (/wiki/Btrfs)	`mkfs.btrfs`	✔ Yes	sys-fs/btrfs-progs (https://packages.gentoo.org/packages/sys-fs/btrfs-progs)
ext4 (/wiki/Ext4)	`mkfs.ext4`	✔ Yes	sys-fs/e2fsprogs (https://packages.gentoo.org/packages/sys-fs/e2fsprogs)
f2fs (/wiki/F2FS)	`mkfs.f2fs`	✔ Yes	sys-fs/f2fs-tools (https://packages.gentoo.org/packages/sys-fs/f2fs-tools)

xfs (/wiki/XFS)	**mkfs.xfs**	✔ Yes	sys-fs/xfsprogs (https://packages.gentoo.org/packages/sys-fs/xfsprogs)
vfat (/wiki/FAT)	**mkfs.vfat**	✔ Yes	sys-fs/dosfstools (https://packages.gentoo.org/packages/sys-fs/dosfstools)
NTFS (/wiki/NTFS)	**mkfs.ntfs**	✔ Yes	sys-fs/ntfs3g (https://packages.gentoo.org/packages/sys-fs/ntfs3g)

For instance, to have the EFI system partition partition (`/dev/sda1`) as FAT32 and the root partition (`/dev/sda3`) as xfs as used in the example partition structure, the following commands would be used:

```
root # mkfs.vfat -F 32 /dev/sda1
root # mkfs.xfs /dev/sda3
```

If using ext4 on a small partition (less than 8 GiB), then the file system must be created with the proper options to reserve enough inodes. This can be done using one of the following commands, respectively:

```
root # mkfs.ext4 -T small /dev/<device>
```

This will generally quadruple the number of inodes for a given file system as its "bytes-per-inode" reduces from one every 16kB to one every 4kB.

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

Activating the swap partition

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

```
root # mkswap /dev/sda2
```

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

```
root # swapon /dev/sda2
```

Create and activate the swap with the commands mentioned above.

Mounting the root partition

Tip

Users of non-Gentoo installation media will need to create the mount point by running:

```
root # mkdir --parents /mnt/gentoo
```

Now that the partitions have been initialized and are housing a filesystem, it is time to mount those partitions. Use the **mount** command, but don't forget to create the necessary mount directories for every partition created. As an example we mount the root partition:

```
root # mount /dev/sda3 /mnt/gentoo
```

Note

If `/tmp/` needs to reside on a separate partition, be sure to change its permissions after mounting:

```
root # chmod 1777 /mnt/gentoo/tmp
```

This also holds for `/var/tmp`.

Later in the instructions the proc filesystem (a virtual interface with the kernel) as well as other kernel pseudo-filesystems will be mounted. But first we install the Gentoo installation files (/wiki/Handbook:AMD64/Installation/Stage).

Installing a stage tarball

Setting the date and time

Before installing Gentoo the clock must be set correctly. Due to Gentoo's web based services using security certificates, it might not be possible to download the installation files if the system clock is too far skewed. Also, files saved with a date in the future may cause strange errors after the initial installation has completed if the clock is corrected later.

Verify the current date and time by running the **date** command:

```
root # date
```

```
Mon Oct  3 13:16:22 PDT 2021
```

If the date/time displayed is more than few minutes off, it should be updated in accuracy using one of the methods below.

Automatic

Most readers will desire to have their system update the time automatically using a time server.

> **Important**
> Some motherboards do not include a Real-Time Clock (RTC), which will keep relatively accurate time even while the system is powered off. It is very important for these systems to automatically sync the system clock with a time server at every system start and on a regular internal thereafter. This is just as important for systems that *do* include a RTC, but have a failed battery.

Official Gentoo live environments include the **chronyd** command (available through the net-misc/chrony (https://packages.gentoo.org/packages/net-misc/chrony) package) and a configuration file pointing to ntp.org time servers. It can be used to automatically sync the system clock to UTC time using a time server. Using this method requires a working network configuration (/wiki/Handbook:AMD64/Installation/Networking) and may not be available on all architectures.

> **Warning**
> Automatic time sync comes at a price. It will reveal the system's IP address and related network information to a time server (in the case of the example below ntp.org). Users with privacy concerns should be aware of this *before* setting the system clock using the below method.

```
root # chronyd -q
```

Manual

For systems that do not have access to a time server, the **date** command can also be used to set the system clock. It will use the following format as an argument: `MMDDhhmmYYYY` syntax (**M**onth, **D**ay, **h**our, **m**inute and **Y**ear).

UTC time is recommended for all Linux systems. A timezone will be defined later in the installation which will modify the clock to display local time.

For instance, to set the date to October 3rd, 13:16 in the year 2021, issue:

```
root # date 100313162021
```

Choosing a stage tarball

Note
Not every architecture has a multilib option. Many only run with native code. Multilib is most commonly applied to **amd64**.

Multilib (32 and 64-bit)

Choosing a base tarball for the system can save a considerable amount of time later on in the installation process, specifically when it is time to choose a system profile (/wiki/Handbook:AMD64/Installation/Base#Choosing_the_right_profile). The selection of a stage tarball will directly impact future system configuration and can save a headache or two later on down the line. The multilib tarball uses 64-bit libraries when possible, and only falls back to the 32-bit versions when necessary for compatibility. This is an excellent option for the majority of installations because it provides a great amount of flexibility for customization in the future. Those who desire their systems to be capable of easily switching profiles should download the multilib tarball option for their respective processor architecture.

Most users should not use the 'advanced' tarballs options; they are for specific software or hardware configurations.

No-multilib (pure 64-bit)

Selecting a no-multilib tarball to be the base of the system provides a complete 64-bit operating system environment. This effectively renders the ability to switch to multilib profiles improbable, although still technically possible.

Warning
Readers who are just starting out with Gentoo should *not* choose a no-multilib tarball unless it is absolutely necessary. Migrating from a no-multilib to a multilib system requires an extremely well-working knowledge of Gentoo and the lower-level toolchain (it may even cause our Toolchain developers (/wiki/Project:Toolchain) to shudder a little). It is not for the faint of heart and is beyond the scope of this guide.

OpenRC

OpenRC (/wiki/OpenRC) is a dependency-based init system (responsible for starting up system services once the kernel has booted) that maintains compatibility with the system provided init program, normally located in `/sbin/init`. It is Gentoo's native and original init system, but is also deployed by a few other Linux distributions and BSD systems.

OpenRC does not function as a replacement for the `/sbin/init` file by default and is 100% compatible with Gentoo init scripts. This means a solution can be found to run the dozens of daemons in the Gentoo ebuild repository.

systemd

systemd is a modern SysV-style init and rc replacement for Linux systems. It is used as the primary init system by a majority of Linux distributions. systemd is fully supported in Gentoo and works for its intended purpose. If something seems lacking in the Handbook for a systemd install path, review the systemd article (/wiki/Systemd) *before* asking for support.

Note
It is technically possible to switch a running Gentoo installation from OpenRC to systemd and back. However, switching requires some effort and is outside the scope of this installation manual. Before

downloading a stage tarball, decide whether OpenRC or systemd will be used as the target init system and download the relevant stage tarball.

Downloading the stage tarball

Go to the Gentoo mount point where the root file system is mounted (most likely `/mnt/gentoo`):

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

Graphical browsers

Those using environments with fully graphical web browsers will have no problem copying a stage file URL from the main website's download section (https://www.gentoo.org/downloads/#other-arches). Simply select the appropriate tab, right click the link to the stage file, then `Copy Link` to copy the link to the clipboard, then paste the link to the **wget** utility on the command-line to download the stage tarball:

```
root # wget <PASTED_STAGE_URL>
```

Command-line browsers

More traditional readers or 'old timer' Gentoo users, working exclusively from command-line may prefer using **links** (www-client/links (https://packages.gentoo.org/packages/www-client/links)), a non-graphical, menu-driven browser. To download a stage, surf to the Gentoo mirror list like so:

```
root # links https://www.gentoo.org/downloads/mirrors/
```

To use an HTTP proxy with **links**, pass on the URL with the `-http-proxy` option:

```
root # links -http-proxy proxy.server.com:8080 https://www.gentoo.org/downloads/mirrors/
```

Next to **links** there is also the **lynx** (www-client/lynx (https://packages.gentoo.org/packages/www-client/lynx)) browser. Like **links** it is a non-graphical browser but it is not menu-driven.

```
root # lynx https://www.gentoo.org/downloads/mirrors/
```

If a proxy needs to be defined, export the *http_proxy* and/or *ftp_proxy* variables:

```
root # export http_proxy="http://proxy.server.com:port"
root # export ftp_proxy="http://proxy.server.com:port"
```

On the mirror list, select a mirror close by. Usually HTTP mirrors suffice, but other protocols are available as well. Move to the `releases/amd64/autobuilds/` directory. There all available stage files are displayed (they might be stored within subdirectories named after the individual sub-architectures). Select one and press `d` to download.

After the stage file download completes, it is possible to verify the integrity and validate the contents of the stage tarball. Those interested should proceed to the next section (/wiki/Handbook:AMD64/Installation/Stage#Verifying_and_validating).

Those not interested in verifying and validating the stage file can close the command-line browser by pressing `q` and can move directly to the Unpacking the stage tarball (/wiki/Handbook:AMD64/Installation/Stage#Unpacking_the_stage_tarball) section.

Verifying and validating

> **Note**
> Most stages are now explicitly suffixed (https://www.gentoo.org/news/2021/07/20/more-downloads.html) with the init system type (`openrc` or `systemd`), although some architectures may still be missing these for now.

Like with the minimal installation CDs, additional downloads to verify and validate the stage file are available. Although these steps may be skipped, these files are provided for users who care about the legitimacy of the file(s) they just downloaded.

- A `.CONTENTS` file that contains a list of all files inside the stage tarball.
- A `.DIGESTS` file that contains checksums of the stage file in different algorithms.

Use **openssl** and compare the output with the checksums provided by the `.DIGESTS` file.

For instance, to validate the SHA512 checksum:

```
root # openssl dgst -r -sha512 stage3-amd64-<release>-<init>.tar.xz
```

Another way is to use the **sha512sum** command:

```
root # sha512sum stage3-amd64-<release>-<init>.tar.xz
```

To validate the Whirlpool checksum:

```
root # openssl dgst -r -whirlpool stage3-amd64-<release>-<init>.tar.xz
```

Compare the output of these commands with the value registered in the `.DIGESTS` files. The values need to match, otherwise the downloaded file might be corrupt (or the digests file is).

Just like with the ISO file, it is also possible to verify the cryptographic signature of the `tar.xz` file using **gpg** to make sure the tarball has not been tampered with:

```
root # gpg --verify stage3-amd64-<release>-<init>.tar.xz{.asc,}
```

The fingerprints of the OpenPGP keys used for signing release media can be found on the release media signatures page (https://www.gentoo.org/downloads/signatures/) of the Gentoo webserver.

Unpacking the stage tarball

Now unpack the downloaded stage onto the system. Use the **tar** utility to proceed:

```
root # tar xpvf stage3-*.tar.xz --xattrs-include='*.*' --numeric-owner
```

Verify the same options (`xpf` and `--xattrs-include='*.*'`) are used in the command. The `x` stands for e**x**tract, the `p` for **p**reserve permissions and the `f` to denote that we want to extract a **f**ile (not standard input). `--xattrs-include='*.*'` is to include preservation of the the extended attributes in all namespaces stored in the archive. Finally, `--numeric-owner` is used to ensure that the user and group IDs of the files being extracted from the tarball will remain the same as Gentoo's release engineering team intended (even if adventurous users are not using official Gentoo live environments).

Now that the stage file is unpacked, proceed with Configuring compile options (/wiki/Handbook:AMD64/Installation/Stage#Configuring_compile_options).

Configuring compile options

Introduction

To optimize the system, it is possible to set variables which impact the behavior of Portage, Gentoo's officially supported package manager. All those variables can be set as environment variables (using **export**) but setting via **export** is not permanent.

> **Note**
> Technically variables can be exported via the shell's (/wiki/Shell) profile or rc files, however that is not best practice for basic system administration.

Portage reads in the `make.conf (/wiki/Make.conf)` file when it runs, which will change runtime behavior depending on the values saved in the file. `make.conf` can be considered the primary configuration file for Portage, so treat its content carefully.

> **Tip**
> A commented listing of all possible variables can be found in

`/mnt/gentoo/usr/share/portage/config/make.conf.example`. Additional documentation on `make.conf` can be found by running **`man 5 make.conf`**.

For a successful Gentoo installation only the variables that are mentioned below need to be set.

Fire up an editor (in this guide we use **nano**) to alter the optimization variables we will discuss hereafter.

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

From the `make.conf.example` file it is obvious how the file should be structured: commented lines start with `#`, other lines define variables using the `VARIABLE="value"` syntax. Several of those variables are discussed in the next section.

CFLAGS and CXXFLAGS

The *CFLAGS* and *CXXFLAGS* variables define the optimization flags for GCC C and C++ compilers respectively. Although those are defined generally here, for maximum performance one would need to optimize these flags for each program separately. The reason for this is because every program is different. However, this is not manageable, hence the definition of these flags in the `make.conf` file.

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

We will not explain all possible optimization options. To understand them all, read the GNU Online Manual(s) (https://gcc.gnu.org/onlinedocs/) 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=` or `-mtune=` flag, which specifies the name of the target architecture. Possible options are described in the `make.conf.example` file (as comments). A commonly used value is *native* as that tells the compiler to select the target architecture of the current system (the one users are installing Gentoo on).

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 even 3 for more speed-optimization flags (every class has the same flags as the one before, plus some extras). `-O2` is the recommended default. `-O3` is known to cause problems when used system-wide, so we recommend to stick to `-O2`.

Another popular optimization flag is `-pipe` (use pipes rather than temporary files for communication between the various stages of compilation). It has no impact on the generated code, but uses more memory. On systems with low memory, gcc might get killed. In that case, do not use this flag.

Using `-fomit-frame-pointer` (which doesn't keep the frame pointer in a register for functions that don't need one) might have serious repercussions on the debugging of applications.

When the *CFLAGS* and *CXXFLAGS* variables are defined, combine the several optimization flags in one string. The default values contained in the stage3 archive that is unpacked should be good enough. The following one is just an example:

CODE **Example *CFLAGS* and *CXXFLAGS* variables**

```
# Compiler flags to set for all languages
COMMON_FLAGS="-march=native -O2 -pipe"
# Use the same settings for both variables
CFLAGS="${COMMON_FLAGS}"
CXXFLAGS="${COMMON_FLAGS}"
```

Tip
Although the GCC optimization (/wiki/GCC_optimization) article has more information on how the various compilation options can affect a system, the Safe CFLAGS (/wiki/Safe_CFLAGS) article may be a more practical place for beginners to start optimizing their systems.

MAKEOPTS

The *MAKEOPTS* variable defines how many parallel compilations should occur when installing a package. As of Portage version 3.0.31[1], if left undefined, Portage's default behavior is to set the *MAKEOPTS* value to the same number of threads returned by **nproc**.

A good choice is the smaller of: the number of threads the CPU has, or the total amount of system RAM divided by 2 GiB.

Warning
Using a large number of jobs can significantly impact memory consumption. A good recommendation is to have at least 2 GiB of RAM for every job specified (so, e.g. `-j6` requires *at least* 12 GiB). To avoid running out of memory, lower the number of jobs to fit the available memory.

Tip
When using parallel emerges (`--jobs`), the effective number of jobs run can grow exponentially (up to make jobs multiplied by emerge jobs). This can be worked around by running a localhost-only distcc configuration that will limit the number of compiler instances per host.

FILE `/etc/portage/make.conf` **Example *MAKEOPTS* declaration**

```
# If left undefined, Portage's default behavior is to set the MAKEOPTS value to the s
ame number of threads returned by `nproc`
MAKEOPTS="-j4"
```

Search for *MAKEOPTS in* ***man 5 make.conf*** *for more details.*

Ready, set, go!

Update the `/mnt/gentoo/etc/portage/make.conf` file to match personal preference and save (nano users would hit `Ctrl`+`o` to write the change and then `Ctrl`+`x` to quit).

Then continue with Installing the Gentoo base system (/wiki/Handbook:AMD64/Installation/Base).

References

1. https://gitweb.gentoo.org/proj/portage.git/commit/?id=5d2af567772bb12b073f1671daea6263055cbdc2 (https://gitweb.gentoo.org/proj/portage.git/commit/?id=5d2af567772bb12b073f1671daea6263055cbdc2)

Chrooting

Optional: Selecting mirrors

Distribution files

Tip
It is safe to skip this step when using non-Gentoo installation media. The app-portage/mirrorselect (https://packages.gentoo.org/packages/app-portage/mirrorselect) package can be emerged later within the stage3 (after Entering the new environment (/wiki/Handbook:AMD64/Installation/Base#Entering_the_new_environment)) and the actions defined in this section can be performed at that point.

In order to download source code quickly it is recommended to select a fast mirror. Portage will look in the `make.conf` file for the *GENTOO_MIRRORS* variable and use the mirrors listed therein. It is possible to surf to the Gentoo mirror list and search for a mirror (or mirrors) that is close to the system's physical location (as those are most frequently the fastest ones). However, we provide a nice tool called **mirrorselect** which provides users with a nice interface to select the mirrors needed. Just navigate to the mirrors of choice and press `Spacebar` to select one or more mirrors.

```
root # mirrorselect -i -o >> /mnt/gentoo/etc/portage/make.conf
```

Gentoo ebuild repository

A second important step in selecting mirrors is to configure the Gentoo ebuild repository via the `/etc/portage/repos.conf/gentoo.conf` file. This file contains the sync information needed to update the package repository (the collection of ebuilds and related files containing all the information Portage needs to download and install software packages).

Configuring the repository can be done in a few simple steps. First, if it does not exist, create the `repos.conf` (`/wiki//etc/portage/repos.conf`) directory:

```
root # mkdir --parents /mnt/gentoo/etc/portage/repos.conf
```

Next, copy the Gentoo repository configuration file provided by Portage to the (newly created) `repos.conf` directory:

```
root # cp /mnt/gentoo/usr/share/portage/config/repos.conf
/mnt/gentoo/etc/portage/repos.conf/gentoo.conf
```

Take a peek with a text editor or by using the **cat** command. The inside of the file should be in `.ini` format and look like this:

FILE **/mnt/gentoo/etc/portage/repos.conf/gentoo.conf**

```
[DEFAULT]
main-repo = gentoo

[gentoo]
location = /var/db/repos/gentoo
sync-type = rsync
sync-uri = rsync://rsync.gentoo.org/gentoo-portage
auto-sync = yes
sync-rsync-verify-jobs = 1
sync-rsync-verify-metamanifest = yes
sync-rsync-verify-max-age = 24
sync-openpgp-key-path = /usr/share/openpgp-keys/gentoo-release.asc
sync-openpgp-key-refresh-retry-count = 40
sync-openpgp-key-refresh-retry-overall-timeout = 1200
sync-openpgp-key-refresh-retry-delay-exp-base = 2
sync-openpgp-key-refresh-retry-delay-max = 60
sync-openpgp-key-refresh-retry-delay-mult = 4
```

The default *sync-uri* variable value listed above will determine a mirror location based on a rotation. This will aid in easing bandwidth stress on Gentoo's infrastructure and will provide a fail-safe in case a specific mirror is offline. It is recommended the default URI is retained unless a local, private Portage mirror will be used.

Tip
The specification for Portage's plug-in sync API can be found in the Portage Sync article (/wiki/Project:Portage/Sync).

Copy DNS info

One thing still remains to be done before entering the new environment and that is copying over the DNS information in `/etc/resolv.conf`. This needs to be done to ensure that networking still works even after entering the new environment. `/etc/resolv.conf` contains the name servers for the network.

To copy this information, it is recommended to pass the `--dereference` option to the **cp** command. This ensures that, if `/etc/resolv.conf` is a symbolic link, that the link's target file is copied instead of the symbolic link itself. Otherwise in the new environment the symbolic link would point to a non-existing file (as the link's target is most likely not available inside the new environment).

```
root # cp --dereference /etc/resolv.conf /mnt/gentoo/etc/
```

Mounting the necessary filesystems

In a few moments, the Linux root will be changed towards the new location.

The filesystems that need to be made available are:

- `/proc/` is a pseudo-filesystem. It looks like regular files, but is generated on-the-fly by the Linux kernel
- `/sys/` is a pseudo-filesystem, like `/proc/` which it was once meant to replace, and is more structured than `/proc/`
- `/dev/` is a regular file system which contains all device. It is partially managed by the Linux device manager (usually **udev**)
- `/run/` is a temporary file system used for files generated at runtime, such as PID files or locks

The `/proc/` location will be mounted on `/mnt/gentoo/proc/` whereas the others are bind-mounted. The latter means that, for instance, `/mnt/gentoo/sys/` will actually *be* `/sys/` (it is just a second entry point to the same filesystem) whereas `/mnt/gentoo/proc/` is a new mount (instance so to speak) of the filesystem.

```
root # mount --types proc /proc /mnt/gentoo/proc
root # mount --rbind /sys /mnt/gentoo/sys
root # mount --make-rslave /mnt/gentoo/sys
root # mount --rbind /dev /mnt/gentoo/dev
root # mount --make-rslave /mnt/gentoo/dev
root # mount --bind /run /mnt/gentoo/run
root # mount --make-slave /mnt/gentoo/run
```

Note
The `--make-rslave` operations are needed for systemd support later in the installation.

Warning
When using non-Gentoo installation media, this might not be sufficient. Some distributions make `/dev/shm` a symbolic link to `/run/shm/` which, after the chroot, becomes invalid. Making `/dev/shm/` a proper tmpfs mount up front can fix this:

```
root # test -L /dev/shm && rm /dev/shm && mkdir /dev/shm
root # mount --types tmpfs --options nosuid,nodev,noexec shm /dev/shm
```

Also ensure that mode 1777 is set:

```
root # chmod 1777 /dev/shm /run/shm
```

Entering the new environment

Now that all partitions are initialized and the base environment installed, it is time to enter the new installation environment by chrooting into it. This means that the session will change its root (most top-level location that can be accessed) from the current installation environment (installation CD or other installation medium) to the installation system (namely the initialized partitions). Hence the name, *change root* or *chroot*.

This chrooting is done in three steps:

1. The root location is changed from `/` (on the installation medium) to `/mnt/gentoo/` (on the partitions) using chroot
2. Some settings (those in `/etc/profile`) are reloaded in memory using the **source** command
3. The primary prompt is changed to help us remember that this session is inside a chroot environment.

```
root # chroot /mnt/gentoo /bin/bash
root # source /etc/profile
root # export PS1="(chroot) ${PS1}"
```

From this point, all actions performed are immediately on the new Gentoo Linux environment.

Tip
If the Gentoo installation is interrupted anywhere after this point, it *should* be possible to 'resume' the installation at this step. There is no need to repartition the disks again! Simply mount the root partition (/wiki/Handbook:AMD64/Installation/Disks#Mounting_the_root_partition) and run the steps above starting with copying the DNS info (/wiki/Handbook:AMD64/Installation/Base#Copy_DNS_info) to re-

enter the working environment. This is also useful for fixing bootloader issues. More information can be found in the chroot (/wiki/Chroot) article.

Mounting the boot partition

Now that the new environment has been entered, it is necessary to mount the boot partition. This will be important when it is time to compile the kernel and install the bootloader:

```
root # mount /dev/sda1 /boot
```

Configuring Portage

Installing a Gentoo ebuild repository snapshot from the web

Next step is to install a snapshot of the Gentoo ebuild repository. This snapshot contains a collection of files that informs Portage about available software titles (for installation), which profiles the system administrator can select, package or profile specific news items, etc.

The use of **emerge-webrsync** is recommended for those who are behind restrictive firewalls (it uses HTTP/FTP protocols for downloading the snapshot) and saves network bandwidth. Readers who have no network or bandwidth restrictions can happily skip down to the next section.

This will fetch the latest snapshot (which is released on a daily basis) from one of Gentoo's mirrors and install it onto the system:

```
root # emerge-webrsync
```

Note
During this operation, **emerge-webrsync** might complain about a missing `/var/db/repos/gentoo/` location. This is to be expected and nothing to worry about - the tool will create the location.

From this point onward, Portage might mention that certain updates are recommended to be executed. This is because system packages installed through the stage file might have newer versions available; Portage is now aware of new packages because of the repository snapshot. Package updates can be safely ignored for now; updates can be delayed until after the Gentoo installation has finished.

Optional: Updating the Gentoo ebuild repository

It is possible to update the Gentoo ebuild repository to the latest version. The previous **emerge-webrsync** command will have installed a very recent snapshot (usually recent up to 24h) so this step is definitely optional.

Suppose there is a need for the last package updates (up to 1 hour), then use **emerge --sync**. This command will use the rsync protocol to update the Gentoo ebuild repository (which was fetched earlier on through **emerge-webrsync**) to the latest state.

```
root # emerge --sync
```

On slow terminals, like some framebuffers or serial consoles, it is recommended to use the `--quiet` option to speed up the process:

```
root # emerge --sync --quiet
```

Reading news items

When the Gentoo ebuild repository is synchronized, Portage may output informational messages similar to the following:

```
 * IMPORTANT: 2 news items need reading for repository 'gentoo'.
 * Use eselect news to read news items.
```

News items were created to provide a communication medium to push critical messages to users via the Gentoo ebuild repository. To manage them, use **eselect news**. The **eselect** application is a Gentoo-specific utility that allows for a common management interface for system administration. In this case, **eselect** is asked to use its `news` module.

For the `news` module, three operations are most used:

- With `list` an overview of the available news items is displayed.
- With `read` the news items can be read.
- With `purge` news items can be removed once they have been read and will not be reread anymore.

```
root # eselect news list
root # eselect news read
```

More information about the news reader is available through its manual page:

```
root # man news.eselect
```

Choosing the right profile

Tip
Desktop profiles are not exclusively for *desktop environments*. They are still suitable for minimal window managers like i3 or sway.

A *profile* is a building block for any Gentoo system. Not only does it specify default values for *USE*, *CFLAGS*, and other important variables, it also locks the system to a certain range of package versions. These settings are all maintained by Gentoo's Portage developers.

To see what profile the system is currently using, run **eselect** using the `profile` module:

```
root # eselect profile list
```

```
Available profile symlink targets:
  [1]   default/linux/amd64/17.1 *
  [2]   default/linux/amd64/17.1/desktop
  [3]   default/linux/amd64/17.1/desktop/gnome
  [4]   default/linux/amd64/17.1/desktop/kde
```

Note
The output of the command is just an example and evolves over time.

Note
To use **systemd**, select a profile which has "systemd" in the name and vice versa, if not

There are also desktop subprofiles available for some architectures.

Warning
Profile upgrades are not to be taken lightly. When selecting the initial profile, use the profile corresponding to **the same version** as the one initially used by stage3 (e.g. 17.1). Each new profile

version is announced through a news item containing migration instructions. Follow the instructions before switching to a newer profile.

After viewing the available profiles for the amd64 architecture, users can select a different profile for the system:

```
root # eselect profile set 2
```

No-multilib

In order to select a pure 64-bit environment, with no 32-bit applications or libraries, use a no-multilib profile:

```
root # eselect profile list
```

```
Available profile symlink targets:
  [1]   default/linux/amd64/17.1 *
  [2]   default/linux/amd64/17.1/desktop
  [3]   default/linux/amd64/17.1/desktop/gnome
  [4]   default/linux/amd64/17.1/desktop/kde
  [5]   default/linux/amd64/17.1/no-multilib
```

Next select the *no-multilib* profile:

```
root # eselect profile set 5
root # eselect profile list
```

```
Available profile symlink targets:
  [1]   default/linux/amd64/17.1
  [2]   default/linux/amd64/17.1/desktop
  [3]   default/linux/amd64/17.1/desktop/gnome
  [4]   default/linux/amd64/17.1/desktop/kde
  [5]   default/linux/amd64/17.1/no-multilib *
```

Note
The `developer` subprofile is specifically for Gentoo Linux development and is not meant to be used by casual users.

Updating the @world set

At this point, it is wise to update the system's @world set (/wiki/World_set_(Portage)) so that a base can be established.

This following step is *necessary* so the system can apply any updates or USE flag changes which have appeared since the stage3 was built and from any profile selection:

```
root # emerge --ask --verbose --update --deep --newuse @world
```

Tip
If a full scale desktop environment profile has been selected this process could greatly extend the amount of time necessary for the install process. Those in a time crunch can work by this 'rule of thumb': the shorter the profile name, the less specific the system's @world set

(/wiki/World_set_(Portage)); the less specific the @world set, the fewer packages the system will require. In other words:

- Selecting `default/linux/amd64/17.1` will require very few packages to be updated, whereas
- Selecting `default/linux/amd64/17.1/desktop/gnome/systemd` will require many packages to be installed since the init system is changing from OpenRC to systemd, and the GNOME desktop environment framework will be installed.

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 support for GTK+ or with support for Qt. 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 users can define what options a package should be compiled with. This is where *USE* comes into play.

In the *USE* variable users 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 -qt5` will compile programs with GNOME (and GTK+) support, and not with KDE (and Qt) support, making the system fully tweaked for GNOME (if the architecture supports it).

The default USE settings are placed in the `make.defaults` files of the Gentoo profile used by the system. Gentoo uses a (complex) inheritance system for its profiles, which we will not dive into at this stage. The easiest way to check the currently active *USE* settings is to run **`emerge --info`** and select the line that starts with USE:

```
root # emerge --info | grep ^USE
```

```
USE="X acl alsa amd64 berkdb bindist bzip2 cli cracklib crypt cxx dri ..."
```

Note
The above example is truncated, the actual list of *USE* values is much, much larger.

A full description on the available USE flags can be found on the system in `/var/db/repos/gentoo/profiles/use.desc`.

```
root # less /var/db/repos/gentoo/profiles/use.desc
```

Inside the **`less`** command, scrolling can be done using the `↑` and `↓` keys, and exited by pressing `q`.

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

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

FILE **`/etc/portage/make.conf`** **Enabling flags for a KDE/Plasma-based system with DVD, ALSA, and CD recording support**

```
USE="-gtk -gnome qt5 kde dvd alsa cdr"
```

When a *USE* value is defined in `/etc/portage/make.conf` it is *added* to the system's USE flag list. USE flags can be globally *removed* by adding a `-` minus sign in front of the value in the the list. For example, to disable support for X graphical environments, `-X` can be set:

FILE **`/etc/portage/make.conf`** **Ignoring default USE flags**

```
USE="-X acl alsa"
```

⚠ Warning
Although possible, setting `-*` (which will disable all USE values except the ones specified in `make.conf`) is *strongly* discouraged and unwise. Ebuild developers choose certain default USE flag values in ebuilds in order to prevent conflicts, enhance security, and avoid errors, and other reasons. Disabling *all* USE flags will negate default behavior and may cause major issues.

CPU_FLAGS_*

Some architectures (including AMD64/X86, ARM, PPC) have a USE_EXPAND (/wiki/USE_EXPAND) variable called CPU_FLAGS_ARCH (/wiki/CPU_FLAGS_X86) (replace ARCH with the relevant system architecture as appropriate).

This is used to configure the build to compile in specific assembly code or other intrinsics, usually hand-written or otherwise extra, and is **not** the same as asking the compiler to output optimized code for a certain CPU feature (e.g. `-march=`).

Users should set this variable in addition to configuring their *COMMON_FLAGS* as desired.

A few steps are needed to set this up:

```
root # emerge --ask app-portage/cpuid2cpuflags
```
Inspect the output manually if curious:

```
root # cpuid2cpuflags
```
Then copy the output into `package.use`:

```
root # echo "*/* $(cpuid2cpuflags)" > /etc/portage/package.use/00cpu-flags
```

VIDEO_CARDS

The *VIDEO_CARDS USE_EXPAND* variable should be configured appropriately depending on the available GPU(s). The Xorg guide (/wiki/Xorg/Guide#Make.conf_configuration) covers how to do this. Setting *VIDEO_CARDS* is not required for a console only install.

Optional: Configure the ACCEPT_LICENSE variable

The licenses of a Gentoo package are stored in the *LICENSE* variable in the ebuild. The accepted specific licenses or groups of licenses of a system are defined in the following files:

- System wide in the selected profile.
- System wide in the `/etc/portage/make.conf` file.
- Per-package in a `/etc/portage/package.license` file.
- Per-package in a `/etc/portage/package.license/` *directory* of files.

Portage looks up in the *ACCEPT_LICENSE* which packages to allow for installation. In order to print the current system wide value run:

```
user $ portageq envvar ACCEPT_LICENSE
```

```
@FREE
```

Optionally override the system wide accepted default in the profiles by changing `/etc/portage/make.conf`.

FILE **`/etc/portage/make.conf`** **Example how to accept licenses with ACCEPT_LICENSE system wide**

```
ACCEPT_LICENSE="-* @FREE @BINARY-REDISTRIBUTABLE"
```

Optionally one can also define accepted licenses per-package as shown in the following directory of files example. Note that the `package.license` directory will need created if it does not already exist:

```
root # mkdir /etc/portage/package.license
```

FILE `/etc/portage/package.license/kernel` **Example how to accept licenses per-package**

```
app-arch/unrar unRAR
sys-kernel/linux-firmware @BINARY-REDISTRIBUTABLE
sys-firmware/intel-microcode intel-ucode
```

Important
The *LICENSE* variable in an ebuild is only a guideline for Gentoo developers and users. It is not a legal statement, and there is no guarantee that it will reflect reality. So don't rely on it, but check the package itself in depth, including all files that have been installed to the system.

The license groups defined in the Gentoo repository, managed by the Gentoo Licenses project (/wiki/Project:Licenses), are:

Group Name	Description
@GPL-COMPATIBLE	GPL compatible licenses approved by the Free Software Foundation [a_license 1]
@FSF-APPROVED	Free software licenses approved by the FSF (includes @GPL-COMPATIBLE)
@OSI-APPROVED	Licenses approved by the Open Source Initiative [a_license 2]
@MISC-FREE	Misc licenses that are probably free software, i.e. follow the Free Software Definition [a_license 3] but are not approved by either FSF or OSI
@FREE-SOFTWARE	Combines @FSF-APPROVED, @OSI-APPROVED and @MISC-FREE
@FSF-APPROVED-OTHER	FSF-approved licenses for "free documentation" and "works of practical use besides software and documentation" (including fonts)
@MISC-FREE-DOCS	Misc licenses for free documents and other works (including fonts) that follow the free definition [a_license 4] but are NOT listed in @FSF-APPROVED-OTHER
@FREE-DOCUMENTS	Combines @FSF-APPROVED-OTHER and @MISC-FREE-DOCS
@FREE	Metaset of all licenses with the freedom to use, share, modify and share modifications. Combines @FREE-SOFTWARE and @FREE-DOCUMENTS
@BINARY-REDISTRIBUTABLE	Licenses that at least permit free redistribution of the software in binary form. Includes @FREE
@EULA	License agreements that try to take away your rights. These are more restrictive than "all-rights-reserved" or require explicit approval

1. https://www.gnu.org/licenses/license-list.html (https://www.gnu.org/licenses/license-list.html)
2. https://www.opensource.org/licenses (https://www.opensource.org/licenses)

3. https://www.gnu.org/philosophy/free-sw.html (https://www.gnu.org/philosophy/free-sw.html)
4. https://freedomdefined.org/ (https://freedomdefined.org/)

Optional: Using systemd as the init system

The remainder of the Gentoo handbook will provide systemd steps alongside OpenRC (/wiki/OpenRC) (the traditional Gentoo init system) where separate steps or recommendations are necessary. System administrators should also consult the systemd (/wiki/Systemd) article for more details on managing systemd as the system and service manager.

Timezone

Note
This step does not apply to users of the musl libc. Users who do not know what that means should perform this step.

Select the timezone for the system. Look for the available timezones in `/usr/share/zoneinfo/`:

```
root # ls /usr/share/zoneinfo
```

Suppose the timezone of choice is *Europe/Brussels*.

OpenRC

We write the timezone name into the `/etc/timezone` file.

```
root # echo "Europe/Brussels" > /etc/timezone
```

Please avoid the `/usr/share/zoneinfo/Etc/GMT*` timezones as their names do not indicate the expected zones. For instance, `GMT-8` is in fact `GMT+8`.

Next, reconfigure the sys-libs/timezone-data (https://packages.gentoo.org/packages/sys-libs/timezone-data) package, which will update the `/etc/localtime` file for us, based on the `/etc/timezone` entry. The `/etc/localtime` file is used by the system C library to know the timezone the system is in.

```
root # emerge --config sys-libs/timezone-data
```

systemd

A slightly different approach is employed when using systemd. A symbolic link is generated:

```
root # ln -sf ../usr/share/zoneinfo/Europe/Brussels /etc/localtime
```

Later, when systemd is running, the timezone and related settings can be configured with the **timedatectl** command.

Configure locales

Note
This step does not apply to users of the musl libc. Users who do not know what that means should perform this step.

Locale generation

Most users will want to use only one or two locales on their system.

Locales specify not only the language that the user should use to interact with the system, but also the rules for sorting strings, displaying dates and times, etc. Locales are *case sensitive* and must be represented exactly as described. A full listing of available locales can be found in the `/usr/share/i18n/SUPPORTED` file.

Supported system locales must be defined in the `/etc/locale.gen` file.

```
root # nano -w /etc/locale.gen
```

The following locales are an example to get both English (United States) and German (Germany/Deutschland) with the accompanying character formats (like UTF-8).

FILE `/etc/locale.gen` **Enabling US and DE locales with the appropriate character formats**

```
en_US ISO-8859-1
en_US.UTF-8 UTF-8
de_DE ISO-8859-1
de_DE.UTF-8 UTF-8
```

Warning
Many applications require least one UTF-8 locale to build properly.

The next step is to run the **locale-gen** command. This command generates all locales specified in the `/etc/locale.gen` file.

```
root # locale-gen
```

To verify that the selected locales are now available, run **locale -a**.

On systemd installs, **localectl** can be used, e.g. **localectl set-locale ...** or **localectl list-locales**.

Locale selection

Once done, it is now time to set the system-wide locale settings. Again we use **eselect** for this, now with the `locale` module.

With **eselect locale list**, the available targets are displayed:

```
root # eselect locale list
```

```
Available targets for the LANG variable:
  [1]  C
  [2]  C.utf8
  [3]  en_US
  [4]  en_US.iso88591
  [5]  en_US.utf8
  [6]  de_DE
  [7]  de_DE.iso88591
  [8]  de_DE.iso885915
  [9]  de_DE.utf8
  [10] POSIX
  [ ]  (free form)
```

With **eselect locale set <NUMBER>** the correct locale can be selected:

```
root # eselect locale set 9
```

Manually, this can still be accomplished through the `/etc/env.d/02locale` file and for Systemd the `/etc/locale.conf` file:

FILE `/etc/env.d/02locale` **Manually setting system locale definitions**

```
LANG="de_DE.UTF-8"
LC_COLLATE="C.UTF-8"
```

Setting the locale will avoid warnings and errors during kernel and software compilations later in the installation.

Now reload the environment:

```
root # env-update && source /etc/profile && export PS1="(chroot) ${PS1}"
```

For additional guidance through the locale selection process read also the Localization guide (/wiki/Localization/Guide) and the UTF-8 (/wiki/UTF-8) guide.

Optional: Installing firmware and/or microcode

Firmware

Before getting to configuring kernel sections, it is beneficial to be aware that some hardware devices require additional, sometimes non-FOSS compliant, firmware to be installed on the system before they will operate correctly. This is often the case for wireless network interfaces commonly found in both desktop and laptop computers. Modern video chips from vendors like AMD, Nvidia, and Intel, often also require external firmware files to be fully functional. Most firmware for modern hardware devices can be found within the sys-kernel/linux-firmware (https://packages.gentoo.org/packages/sys-kernel/linux-firmware) package.

It is recommended to have the sys-kernel/linux-firmware (https://packages.gentoo.org/packages/sys-kernel/linux-firmware) package installed before the initial system reboot in order to have the firmware available in the event that it is necessary:

```
root # emerge --ask sys-kernel/linux-firmware
```

Note
Installing certain firmware packages often requires accepting the associated firmware licenses. If necessary, visit the license handling section (/wiki/Handbook:AMD64/Working/Portage#Licenses) of the Handbook for help on accepting licenses.

It is important to note that kernel symbols that are built as modules (M) will load their associated firmware files from the filesystem when they are loaded by the kernel. It is not necessary to include the device's firmware files into the kernel's binary image for symbols loaded as modules.

Microcode

In addition to discrete graphics hardware and network interfaces, CPUs also can require firmware updates. Typically this kind of firmware is referred to as *microcode*. Newer revisions of microcode are sometimes necessary to patch instability, security concerns, or other miscellaneous bugs in CPU hardware.

Microcode updates for AMD CPUs are distributed within the aforementioned sys-kernel/linux-firmware (https://packages.gentoo.org/packages/sys-kernel/linux-firmware) package. Microcode for Intel CPUs can be found within the sys-firmware/intel-microcode (https://packages.gentoo.org/packages/sys-firmware/intel-mic

rocode) package, which will need to be installed separately. See the Microcode article (/wiki/Microcode) for more information on how to apply microcode updates.

Kernel configuration and compilation

Now it is time to configure and compile the kernel sources. For the purposes of the installation, three approaches to kernel management will be presented, however at any point post-installation a new approach can be employed.

Ranked from least involved to most involved:

Full automation approach: Distribution kernels (/wiki/Handbook:AMD64/Installation/Kernel#Distribution_kernels)
A Distribution Kernel (/wiki/Project:Distribution_Kernel) is used to configure, automatically build, and install the Linux kernel, its associated modules, and (optionally, but enabled by default) an initramfs file. Future kernel updates are fully automated since they are handled through the package manager, just like any other system package. It is possible provide a custom kernel configuration file (/wiki/Project:Distribution_Kernel#Modifying_kernel_configuration) if customization is necessary. This is the least involved process and is perfect for new Gentoo users due to it working out-of-the-box and offering minimal involvement from the system administrator.
Hybrid approach: Genkernel (/wiki/Handbook:AMD64/Installation/Kernel#Alternative:_Genkernel)
New kernel sources are installed via the system package manager. System administrators use Gentoo's **genkernel** tool to generically configure, automatically build and install the Linux kernel, its associated modules, and (optionally, but ***not*** enabled by default) an initramfs file. It is possible provide a custom kernel configuration file if customization is necessary. Future kernel configuration, compilation, and installation require the system administrator's involvement in the form of running **eselect kernel**, **genkernel**, and potentially other commands for each update.
Full manual approach (/wiki/Handbook:AMD64/Installation/Kernel#Alternative:_Manual_configuration)
New kernel sources are installed via the system package manager. The kernel is manually configured, built, and installed using the **eselect kernel** and a slew of **make** commands. Future kernel updates repeat the manual process of configuring, building, and installing the kernel files. This is the most involved process, but offers maximum control over the kernel update process.

The core around which all distributions are built is the Linux kernel. It is the layer between the user's programs and the system hardware. Although the handbook provides its users several possible kernel sources, a more comprehensive listing with more detailed descriptions is available at the Kernel overview page (/wiki/Kernel/Overview).

Distribution kernels

Distribution Kernels (/wiki/Project:Distribution_Kernel) are ebuilds that cover the complete process of unpacking, configuring, compiling, and installing the kernel. The primary advantage of this method is that the kernels are updated to new versions by the package manager as part of @world upgrade. This requires no more involvement than running an **emerge** command. Distribution kernels default to a configuration supporting the majority of hardware, however two mechanisms are offered for customization: savedconfig and config snippets. See the project page for more details on configuration. (/wiki/Project:Distribution_Kernel#Modifying_kernel_configuration)

Installing the correct installkernel package

Before using the distribution kernels, please verify that the correct installkernel package for the system has been installed. When using systemd-boot (/wiki/Systemd-boot) (formerly gummiboot) as the bootloader, install:

```
root # emerge --ask sys-kernel/installkernel-systemd-boot
```

When using a traditional a `/boot` layout (e.g. GRUB, LILO, etc.), the *gentoo* variant should be installed by default. If in doubt:

```
root # emerge --ask sys-kernel/installkernel-gentoo
```

Installing a distribution kernel

To build a kernel with Gentoo patches from source, type:

```
root # emerge --ask sys-kernel/gentoo-kernel
```

System administrators who want to avoid compiling the kernel sources locally can instead use precompiled kernel images:

```
root # emerge --ask sys-kernel/gentoo-kernel-bin
```

Upgrading and cleaning up

Once the kernel is installed, the package manager will automatically update it to newer versions. The previous versions will be kept until the package manager is requested to clean up stale packages. To reclaim disk space, stale packages can be trimmed by periodically running emerge with the `--depclean` option:

```
root # emerge --depclean
```

Alternatively, to specifically clean up old kernel versions:

```
root # emerge --prune sys-kernel/gentoo-kernel sys-kernel/gentoo-kernel-bin
```

Post-install/upgrade tasks

Distribution kernels are capable of rebuilding kernel modules installed by other packages. `linux-mod.eclass` provides the `dist-kernel` USE flag which controls a subslot dependency on virtual/dist-kernel (https://packages.gentoo.org/packages/virtual/dist-kernel).

Enabling this USE flag on packages like sys-fs/zfs (https://packages.gentoo.org/packages/sys-fs/zfs) and sys-fs/zfs-kmod (https://packages.gentoo.org/packages/sys-fs/zfs-kmod) allows them to automatically be rebuilt against a newly updated kernel and, if applicable, will re-generate the initramfs accordingly.

Manually rebuilding the initramfs

If required, manually trigger such rebuilds by, after a kernel upgrade, executing:

```
root # emerge --ask @module-rebuild
```

If any kernel modules (e.g. ZFS) are needed at early boot, rebuild the initramfs afterward via:

```
root # emerge --config sys-kernel/gentoo-kernel
root # emerge --config sys-kernel/gentoo-kernel-bin
```

Installing the kernel sources

> **Note**
> This section is only relevant when using the following **genkernel** (hybrid) or manual kernel management approach.

When installing and compiling the kernel for amd64-based systems, Gentoo recommends the sys-kernel/gentoo-sources (https://packages.gentoo.org/packages/sys-kernel/gentoo-sources) package.

Choose an appropriate kernel source and install it using **emerge**:

```
root # emerge --ask sys-kernel/gentoo-sources
```

This will install the Linux kernel sources in `/usr/src/` using the specific kernel version in the path. It will not create a symbolic link by itself without `USE=symlink` being enabled on the chosen kernel sources package.

It is conventional for a `/usr/src/linux` symlink to be maintained, such that it refers to whichever sources correspond with the currently running kernel. However, this symbolic link will not be created by default. An easy way to create the symbolic link is to utilize eselect's kernel module.

For further information regarding the purpose of the symlink, and how to manage it, please refer to Kernel/Upgrade (/wiki/Kernel/Upgrade).

First, list all installed kernels:

```
root # eselect kernel list
```

```
Available kernel symlink targets:
  [1]   linux-6.1.38-gentoo
```

In order to create a symbolic link called `linux`, use:

```
root # eselect kernel set 1
root # ls -l /usr/src/linux
```

```
lrwxrwxrwx    1 root   root    12 Oct 13 11:04 /usr/src/linux -> linux-6.1.38-gentoo
```

Alternative: Genkernel

If an entirely manual configuration looks too daunting, system administrators should consider using **genkernel (/wiki/Genkernel)** as a hybrid approach to kernel maintenance.

Genkernel provides a generic kernel configuration file, automatically ***gen***erates the ***kernel***, initramfs, and associated modules, and then installs the resulting binaries to the appropriate locations. This results in minimal and generic hardware support for the system's first boot, and allows for additional update control and customization of the kernel's configuration in the future.

Be informed: while using **genkernel** to maintain the kernel provides system administrators with more update control over the system's kernel, initramfs, and other options, it *will* require a time and effort commitment to perform future kernel updates as new sources are released. Those looking for a hands-off approach to kernel maintenance should use a distribution kernel (/wiki/Handbook:AMD64/Installation/Kernel#Alternative:_Using_distribution_kernels).

For additional clarity, it is a *misconception* to believe genkernel automatically generates a *custom* kernel configuration for the hardware on which it is run; it uses a predetermined kernel configuration that supports most generic hardware and automatically handles the **make** commands necessary to assemble and install the kernel, the associate modules, and the initramfs file.

Binary redistributable software license group

If the linux-firmware package has been previously installed, then skip onward to the to the installation section (/wiki/Handbook:AMD64/Installation/Kernel#Installation).

As a prerequisite, due to the `firwmare` USE flag being enabled by default for the sys-kernel/genkernel (https://packages.gentoo.org/packages/sys-kernel/genkernel) package, the package manager will also attempt to pull in the sys-kernel/linux-firmware (https://packages.gentoo.org/packages/sys-kernel/linux-firmware) package. The binary redistributable software licenses are required to be accepted before the linux-firmware will install.

This license group can be accepted system-wide for any package by adding the `@BINARY-REDISTRIBUTABLE` as an *ACCEPT_LICENSE* value in the `/etc/portage/make.conf` file. It can be exclusively accepted for the linux-firmware package by adding a specific inclusion via a `/etc/portage/package.license/linux-firmware` file.

If necessary, review the methods of accepting software licenses (/wiki/Handbook:AMD64/Installation/Base#Optional:_Configure_the_ACCEPT_LICENSE_variable) available in the Installing the base system (/wiki/Handbook:AMD64/Installation/Base) chapter of the handbook, then

make some changes for acceptable software licenses.

If in analysis paralysis, the following will do the trick:

```
root # mkdir /etc/portage/package.license
```

FILE **`/etc/portage/package.license/linux-firmware`** **Accept binary redistributable licenses for the linux-firmware package**

```
sys-kernel/linux-firmware @BINARY-REDISTRIBUTABLE
```

Installation

Explanations and prerequisites aside, install the sys-kernel/genkernel (https://packages.gentoo.org/packages/sys-kernel/genkernel) package:

```
root # emerge --ask sys-kernel/genkernel
```

Generation

Compile the kernel sources by running **`genkernel all`**. Be aware though, as **`genkernel`** compiles a kernel that supports a wide array of hardware for differing computer architectures, this compilation may take quite a while to finish.

> **Note**
> If the root partition/volume uses a filesystem other than ext4, it may be necessary to manually configure the kernel using **`genkernel --menuconfig all`** to add built-in kernel support for the particular filesystem(s) (i.e. not building the filesystem as a module).

> **Note**
> Users of LVM2 should add `--lvm` as an argument to the **`genkernel`** command below.

```
root # genkernel --mountboot --install all
```

Once genkernel completes, a kernel and an initial ram filesystem (initramfs) will be generated and installed into the `/boot` directory. Associated modules will be installed into the `/lib/modules` directory. The initramfs will be started immediately after loading the kernel to perform hardware auto-detection (just like in the live disk image environments).

```
root # ls /boot/vmlinu* /boot/initramfs*
root # ls /lib/modules
```

Alternative: Manual configuration

Introduction

Manually configuring a kernel is often seen as the most difficult procedure a Linux user ever has to perform. Nothing is less true - after configuring a couple of kernels no one remembers that it was difficult!

However, one thing is true: it is vital to know the system when a kernel is configured manually. Most information can be gathered by emerging sys-apps/pciutils (https://packages.gentoo.org/packages/sys-apps/pciutils) which contains the **`lspci`** command:

```
root # emerge --ask sys-apps/pciutils
```

> **Note**
> Inside the chroot, it is safe to ignore any pcilib warnings (like *pcilib: cannot open /sys/bus/pci/devices*) that **`lspci`** might throw out.

Another source of system information is to run **lsmod** to see what kernel modules the installation CD uses as it might provide a nice hint on what to enable.

Now go to the kernel source directory and execute **make menuconfig**. This will fire up menu-driven configuration screen.

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

The Linux kernel configuration has many, many sections. Let's first list some options that must be activated (otherwise Gentoo will not function, or not function properly without additional tweaks). We also have a Gentoo kernel configuration guide (/wiki/Kernel/Gentoo_Kernel_Configuration_Guide) on the Gentoo wiki that might help out further.

Enabling required options

When using sys-kernel/gentoo-sources (https://packages.gentoo.org/packages/sys-kernel/gentoo-sources), it is strongly recommend the Gentoo-specific configuration options be enabled. These ensure that a minimum of kernel features required for proper functioning is available:

KERNEL **Enabling Gentoo-specific options**

```
Gentoo Linux --->
  Generic Driver Options --->
    [*] Gentoo Linux support
    [*]   Linux dynamic and persistent device naming (userspace devfs) support
    [*]   Select options required by Portage features
        Support for init systems, system and service managers  --->
          [*] OpenRC, runit and other script based systems and managers
          [*] systemd
```

Naturally the choice in the last two lines depends on the selected init system (OpenRC (/wiki/OpenRC) vs. systemd (/wiki/Systemd)). It does not hurt to have support for both init systems enabled.

When using sys-kernel/vanilla-sources (https://packages.gentoo.org/packages/sys-kernel/vanilla-sources), the additional selections for init systems will be unavailable. Enabling support is possible, but goes beyond the scope of the handbook.

Enabling support for typical system components

Make sure that every driver that is vital to the booting of the system (such as SATA controllers, NVMe block device support, filesystem support, etc.) is compiled in the kernel and not as a module, otherwise the system may not be able to boot completely.

Next select the exact processor type. It is also recommended to enable MCE features (if available) so that users are able to be notified of any hardware problems. On some architectures (such as x86_64), these errors are not printed to **dmesg**, but to `/dev/mcelog`. This requires the app-admin/mcelog (https://packages.gentoo.org/packages/app-admin/mcelog) package.

Also select *Maintain a devtmpfs file system to mount at /dev* so that critical device files are already available early in the boot process (*CONFIG_DEVTMPFS* and *CONFIG_DEVTMPFS_MOUNT*):

KERNEL **Enabling devtmpfs support (*CONFIG_DEVTMPFS*)**

```
Device Drivers --->
  Generic Driver Options --->
    [*] Maintain a devtmpfs filesystem to mount at /dev
    [*]   Automount devtmpfs at /dev, after the kernel mounted the rootfs
```

Verify SCSI disk support has been activated (*CONFIG_BLK_DEV_SD*):

KERNEL **Enabling SCSI disk support (*CONFIG_SCSI*, *CONFIG_BLK_DEV_SD*)**

```
Device Drivers --->
  SCSI device support  --->
    <*> SCSI device support
    <*> SCSI disk support
```

KERNEL **Enabling basic SATA and PATA support (*CONFIG_ATA_ACPI*, *CONFIG_SATA_PMP*, *CONFIG_SATA_AHCI*, *CONFIG_ATA_BMDMA*, *CONFIG_ATA_SFF*, *CONFIG_ATA_PIIX*)**

```
Device Drivers --->
  <*> Serial ATA and Parallel ATA drivers (libata)  --->
    [*] ATA ACPI Support
    [*] SATA Port Multiplier support
    <*> AHCI SATA support (ahci)
    [*] ATA BMDMA support
    [*] ATA SFF support (for legacy IDE and PATA)
    <*> Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support (ata_piix)
```

Verify basic NVMe support has been enabled:

KERNEL **Enable basic NVMe support for Linux 4.4.x (*CONFIG_BLK_DEV_NVME*)**

```
Device Drivers  --->
  <*> NVM Express block device
```

KERNEL **Enable basic NVMe support for Linux 5.x.x (*CONFIG_DEVTMPFS*)**

```
Device Drivers --->
  NVME Support --->
    <*> NVM Express block device
```

It does not hurt to enable the following additional NVMe support:

KERNEL **Enabling additional NVMe support (*CONFIG_NVME_MULTIPATH*, *CONFIG_NVME_MULTIPATH*, *CONFIG_NVME_HWMON*, *CONFIG_NVME_FC*, *CONFIG_NVME_TCP*, *CONFIG_NVME_TARGET*, *CONFIG_NVME_TARGET_PASSTHRU*, *CONFIG_NVME_TARGET_LOOP*, *CONFIG_NVME_TARGET_FC*, *CONFIG_NVME_TARGET_FCLOOP*, *CONFIG_NVME_TARGET_TCP*)**

```
[*] NVMe multipath support
[*] NVMe hardware monitoring
<M> NVM Express over Fabrics FC host driver
<M> NVM Express over Fabrics TCP host driver
<M> NVMe Target support
  [*]   NVMe Target Passthrough support
  <M>   NVMe loopback device support
  <M>   NVMe over Fabrics FC target driver
  < >     NVMe over Fabrics FC Transport Loopback Test driver (NEW)
  <M>   NVMe over Fabrics TCP target support
```

Now go to File Systems and select support for the filesystems that will be used by the system. Do not compile the file system that is used for the root filesystem as module, otherwise the system may not be able to mount the partition. Also select *Virtual memory* and */proc file system*. Select one or more of the following options as needed by the system:

KERNEL **Enable file system support (*CONFIG_EXT2_FS*, *CONFIG_EXT3_FS*, *CONFIG_EXT4_FS*, *CONFIG_BTRFS_FS*, *CONFIG_XFS_FS*, *CONFIG_MSDOS_FS*, *CONFIG_VFAT_FS*, *CONFIG_PROC_FS*, and *CONFIG_TMPFS*)**

```
File systems --->
  <*> Second extended fs support
  <*> The Extended 3 (ext3) filesystem
  <*> The Extended 4 (ext4) filesystem
  <*> Btrfs filesystem support
  <*> XFS filesystem support
  DOS/FAT/NT Filesystems  --->
    <*> MSDOS fs support
    <*> VFAT (Windows-95) fs support
  Pseudo Filesystems --->
    [*] /proc file system support
    [*] Tmpfs virtual memory file system support (former shm fs)
```

If PPPoE is used to connect to the Internet, or a dial-up modem, then enable the following options (*CONFIG_PPP*, *CONFIG_PPP_ASYNC*, and *CONFIG_PPP_SYNC_TTY*):

KERNEL **Enabling PPPoE support (*PPPoE*, *CONFIG_PPPOE*, *CONFIG_PPP_ASYNC*, *CONFIG_PPP_SYNC_TTY***

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

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 ppp when configured to do kernel mode PPPoE.

Don't forget to include support in the kernel for the network (Ethernet or wireless) cards.

Most systems also have multiple cores at their disposal, so it is important to activate *Symmetric multi-processing support* (*CONFIG_SMP*):

KERNEL **Activating SMP support (*CONFIG_SMP*)**

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

Note
In multi-core systems, each core counts as one processor.

If USB input devices (like keyboard or mouse) or other USB devices will be used, do not forget to enable those as well:

KERNEL **Enable USB and human input device support (*CONFIG_HID_GENERIC*, *CONFIG_USB_HID*, *CONFIG_USB_SUPPORT*, *CONFIG_USB_XHCI_HCD*, *CONFIG_USB_EHCI_HCD*, *CONFIG_USB_OHCI_HCD*, *(CONFIG_HID_GENERIC, CONFIG_USB_HID, CONFIG_USB_SUPPORT, CONFIG_USB_XHCI_HCD, CONFIG_USB_EHCI_HCD, CONFIG_USB_OHCI_HCD, CONFIG_USB4)***

```
Device Drivers --->
  HID support  --->
    -*- HID bus support
    <*>   Generic HID driver
    [*]   Battery level reporting for HID devices
      USB HID support  --->
        <*> USB HID transport layer
  [*] USB support  --->
    <*>     xHCI HCD (USB 3.0) support
    <*>     EHCI HCD (USB 2.0) support
    <*>     OHCI HCD (USB 1.1) support
  <*> Unified support for USB4 and Thunderbolt  --->
```

Architecture specific kernel configuration

Make sure to select IA32 Emulation if 32-bit programs should be supported (*CONFIG_IA32_EMULATION*). Gentoo installs a multilib system (mixed 32-bit/64-bit computing) by default, so unless a no-multilib profile is used, this option is required.

KERNEL **Selecting processor types and features**

```
Processor type and features  --->
   [ ] Machine Check / overheating reporting
   [ ]   Intel MCE Features
   [ ]   AMD MCE Features
   Processor family (AMD-Opteron/Athlon64)  --->
      ( ) Opteron/Athlon64/Hammer/K8
      ( ) Intel P4 / older Netburst based Xeon
      ( ) Core 2/newer Xeon
      ( ) Intel Atom
      ( ) Generic-x86-64
Binary Emulations --->
   [*] IA32 Emulation
```

Enable GPT partition label support if that was used previously when partitioning the disk (*CONFIG_PARTITION_ADVANCED* and *CONFIG_EFI_PARTITION*):

KERNEL **Enable support for GPT**

```
-*- Enable the block layer --->
   Partition Types --->
      [*] Advanced partition selection
      [*] EFI GUID Partition support
```

Enable EFI stub support, EFI variables and EFI Framebuffer in the Linux kernel if UEFI is used to boot the system (*CONFIG_EFI*, *CONFIG_EFI_STUB*, *CONFIG_EFI_MIXED*, *CONFIG_EFI_VARS*, and *CONFIG_FB_EFI*):

KERNEL **Enable support for UEFI**

```
Processor type and features  --->
    [*] EFI runtime service support
    [*]   EFI stub support
    [*]     EFI mixed-mode support

Device Drivers
    Firmware Drivers  --->
        EFI (Extensible Firmware Interface) Support  --->
            <*> EFI Variable Support via sysfs
    Graphics support  --->
        Frame buffer Devices  --->
            <*> Support for frame buffer devices  --->
                [*]   EFI-based Framebuffer Support
```

Compiling and installing

With the configuration now done, it is time to compile and install the kernel. Exit the configuration and start the compilation process:

```
root # make && make modules_install
```

Note
It is possible to enable parallel builds using **make -jX** with `X` being an integer number of parallel tasks that the build process is allowed to launch. This is similar to the instructions about `/etc/portage/make.conf` earlier, with the *MAKEOPTS* variable.

When the kernel has finished compiling, copy the kernel image to `/boot/`. This is handled by the **make install** command:

```
root # make install
```

This will copy the kernel image into `/boot/` together with the `System.map` file and the kernel configuration file.

Optional: Building an initramfs

In certain cases it is necessary to build an initramfs - an **i**nitial **ram**-based **f**ile **s**ystem. The most common reason is when important file system locations (like `/usr/` or `/var/`) are on separate partitions. With an initramfs, these partitions can be mounted using the tools available inside the initramfs.

Without an initramfs, there is a risk that the system will not boot properly as the tools that are responsible for mounting the file systems require information that resides on unmounted file systems. An initramfs will pull in the necessary files into an archive which is used right after the kernel boots, but before the control is handed over to the init tool. Scripts on the initramfs will then make sure that the partitions are properly mounted before the system continues booting.

Important
If using **genkernel**, it should be used for both building the kernel *and* the initramfs. When using **genkernel** only for generating an initramfs, it is crucial to pass `--kernel-config=/path/to/kernel.config` to **genkernel** or the generated initramfs **may not work** with a manually built kernel. Note that manually built kernels go beyond the scope of support for the handbook. See the kernel configuration (/wiki/Kernel/Configuration) article for more information.

To install an initramfs, install sys-kernel/dracut (https://packages.gentoo.org/packages/sys-kernel/dracut) first, then have it generate an initramfs:

```
root # emerge --ask sys-kernel/dracut
root # dracut --kver=6.1.38-gentoo
```

The initramfs will be stored in `/boot/`. The resulting file can be found by simply listing the files starting with *initramfs*:

```
root # ls /boot/initramfs*
```

Now continue with Kernel modules (/wiki/Handbook:AMD64/Installation/Kernel#Kernel_modules).

Kernel modules

Listing available kernel modules

> **Note**
> Hardware modules are optional to be listed manually. **udev** will normally load all hardware modules that are detected to be connected in most cases. However, it is not harmful for modules that will be automatically loaded to be listed. Modules cannot be loaded twice; they are either loaded or unloaded. Sometimes exotic hardware requires help to load their drivers.

The modules that need to be loaded during each boot in can be added to `/etc/modules-load.d/*.conf` files in the format of one module per line. When extra options are needed for the modules, they should be set in `/etc/modprobe.d/*.conf` files instead.

To view all modules available for a specific kernel version, issue the following **find** command. Do not forget to substitute "<kernel version>" with the appropriate version of the kernel to search:

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

Force loading particular kernel modules

To force load the kernel to load the `3c59x.ko` module (which is the driver for a specific 3Com network card family), edit the `/etc/modules-load.d/network.conf` file and enter the module name within it.

```
root # mkdir -p /etc/modules-load.d
root # nano -w /etc/modules-load.d/network.conf
```

Note that the module's `.ko` file suffix is insignificant to the loading mechanism and left out of the configuration file:

FILE **`/etc/modules-load.d/network.conf`** **Force loading 3c59x module**

```
3c59x
```

Continue the installation with Configuring the system (/wiki/Handbook:AMD64/Installation/System).

Filesystem information

About fstab

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

Creating the fstab file

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

1. The first field shows the block special device or remote filesystem to be mounted. Several kinds of device identifiers are available for block special device nodes, including paths to device files, filesystem labels and UUIDs, and partition labels and UUIDs.
2. The second field shows the mount point at which the partition should be mounted.
3. The third field shows the type of filesystem used by the partition.
4. The fourth field shows the mount options used by **`mount`** when it wants to mount the partition. As every filesystem has its own mount options, so system admins are encouraged to read the mount man page (**`man mount`**) for a full listing. Multiple mount options are comma-separated.
5. The fifth field is used by dump to determine if the partition needs to be dumped or not. This can generally be left as `0` (zero).
6. 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` if a filesystem check is not necessary).

> **Important**
> The default `/etc/fstab` file provided in Gentoo stage files is *not* a valid `fstab` file but instead a template that can be used to enter in relevant values.

```
root # nano /etc/fstab
```

In the remainder of the text, the default `/dev/sd*` block device files will be used as partition identifiers.

Filesystem labels and UUIDs

Both MBR (BIOS) and GPT include support for *filesystem* labels and *filesystem* UUIDs. These attributes can be defined in `/etc/fstab` as alternatives for the **`mount`** command to use when attempting to find and mount block devices. Filesystem labels and UUIDs are identified by the *LABEL* and *UUID* prefix and can be viewed with the **`blkid`** command:

```
root # blkid
```

> **Warning**
> If the filesystem inside a partition is wiped, then the filesystem label and the UUID values will be subsequently altered or removed.

Because of uniqueness, readers that are using an MBR-style partition table are recommended to use UUIDs over labels to define mountable volumes in `/etc/fstab`.

Important
UUIDs of the filesystem on a LVM volume and its LVM snapshots are identical, therefore using UUIDs to mount LVM volumes should be avoided.

Partition labels and UUIDs

Users who have gone the GPT route have a couple more 'robust' options available to define partitions in `/etc/fstab`. Partition labels and partition UUIDs can be used to identify the block device's individual partition(s), regardless of what filesystem has been chosen for the partition itself. Partition labels and UUIDs are identified by the *PARTLABEL* and *PARTUUID* prefixes respectively and can be viewed nicely in the terminal by running the **blkid** command:

```
root # blkid
```

While not always true for partition labels, using a UUID to identify a partition in `fstab` provides a guarantee that the bootloader will not be confused when looking for a certain volume, even if the filesystem would be changed in the future. Using the older default block device files (`/dev/sd*N`) for defining the partitions in `fstab` is risky for systems that are restarted often and have SATA block devices added and removed regularly.

The naming for block device files depends on a number of factors, including how and in what order the disks are attached to the system. They also could show up in a different order depending on which of the devices are detected by the kernel first during the early boot process. With this being stated, unless one intends to constantly fiddle with the disk ordering, using default block device files is a simple and straightforward approach.

Let us take a look at how to write down the options for the `/boot/` partition. This is just an example, and should be modified according to the partitioning decisions made earlier in the installation. In our amd64 partitioning example, `/boot/` is usually the `/dev/sda1` partition, with xfs as filesystem. It needs to be checked during boot, so we would write down:

FILE `/etc/fstab` **An example /boot line for /etc/fstab**

```
# Adjust any formatting difference from the Preparing the disks step
/dev/sda1   /boot     vfat    defaults        0 2
```

Some users don't want their `/boot/` partition to be mounted automatically to improve their system's security. Those people should substitute *defaults* with *noauto*. This does mean that those users will need to manually mount this partition every time they want to use it.

Add the rules that match the previously decided partitioning scheme and append rules for devices such as CD-ROM drive(s), and of course, if other partitions or drives are used, for those too.

Below is a more elaborate example of an `/etc/fstab` file:

FILE `/etc/fstab` **A full /etc/fstab example**

```
# Adjust any formatting difference and additional partitions created from the Preparing
the disks step
/dev/sda1   /boot        vfat    defaults    0 2
/dev/sda2   none         swap    sw                  0 0
/dev/sda3   /            xfs    defaults,noatime              0 1

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

When `auto` is used in the third field, it makes the **mount** command guess what the filesystem would be. This is recommended for removable media as they can be created with one of many filesystems. The `user` option in the fourth field makes it possible for non-root users to mount the CD.

To improve performance, most users would want to add the `noatime` mount option, which results in a faster system since access times are not registered (those are not needed generally anyway). This is also recommended for systems with solid state drives (SSDs). Users may wish to consider `lazytime` instead.

> **Tip**
> Due to degradation in performance, defining the `discard` mount option in `/etc/fstab` is not recommended. It is generally better to schedule block discards on a periodic basis using a job scheduler such as **cron** or a timer (systemd). See Periodic fstrim jobs (/wiki/SSD#Periodic_fstrim_jobs) for more information.

Double-check the `/etc/fstab` file, save and quit to continue.

Networking information

It is important to note the following sections are provided to help the reader quickly setup their system to partake in a local area network.

For systems running OpenRC, a more detailed reference for network setup is available in the advanced network configuration (/wiki/Handbook:AMD64/Networking/Introduction) section, which is covered near the end of the handbook. Systems with more specific network needs may need to skip ahead, then return here to continue with the rest of the installation.

For more specific systemd network setup, please review see the networking portion (/wiki/Systemd#Network) of the systemd (/wiki/Systemd) article.

Hostname

One of the choices the system administrator has to make is name their PC. This seems to be quite easy, but lots of users are having difficulties finding the appropriate name for the hostname. To speed things up, know that the decision is not final - it can be changed afterwards. In the examples below, the hostname *tux* is used.

Set the hostname (OpenRC or systemd)

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

systemd

To set the system hostname for a system currently *running* systemd, the **hostnamectl** utility may be used. During the installation process however, systemd-firstboot (/wiki/Handbook:AMD64/Installation/System#systemd_2) command must be used instead (see later on in handbook).

For setting the hostname to "tux", one would run:

```
root # hostnamectl hostname tux
```

View help by running **`hostnamectl --help`** or **`man 1 hostnamectl`**.

Network

There are *many* options available for configuring network interfaces. This section covers a only a few methods. Choose the one which seems best suited to the setup needed.

DHCP via dhcpcd (any init system)

Most LAN networks operate a DHCP server. If this is the case, then using the dhcpcd program to obtain an IP address is recommended.

To install:

```
root # emerge --ask net-misc/dhcpcd
```

To enable and then start the service on OpenRC systems:

```
root # rc-update add dhcpcd default
root # rc-service dhcpcd start
```

To enable and start the service on systemd systems:

```
root # systemctl enable --now dhcpcd
```

With these steps completed, next time the system boots, dhcpcd should obtain an IP address from the DHCP server. See the Dhcpcd (/wiki/Dhcpcd) article for more details.

netifrc (OpenRC)

Tip
This is one particular way of setting up the network using Netifrc (/wiki/Netifrc) on OpenRC. Other methods exist for simpler setups like Dhcpcd (/wiki/Dhcpcd).

Configuring the network

During the Gentoo Linux installation, networking was already configured. However, that was for the live environment itself and not for the installed environment. Right now, the network configuration is made for the installed Gentoo Linux system.

Note
More detailed information about networking, including advanced topics like bonding, bridging, 802.1Q VLANs or wireless networking is covered in the advanced network configuration (/wiki/Handbook:AMD64/Networking/Introduction) section.

All networking information is gathered in `/etc/conf.d/net`. It uses a straightforward - yet perhaps not intuitive - syntax. Do not fear! Everything is explained below. A fully commented example that covers many different configurations is available in `/usr/share/doc/netifrc-*/net.example.bz2`.

First install net-misc/netifrc (https://packages.gentoo.org/packages/net-misc/netifrc):

```
root # emerge --ask --noreplace net-misc/netifrc
```

DHCP is used by default. For DHCP to work, a DHCP client needs to be installed. This is described later in Installing Necessary System Tools.

If the network connection needs to be configured because of specific DHCP options or because DHCP is not used at all, then open `/etc/conf.d/net`:

```
root # nano /etc/conf.d/net
```

Set both *config_eth0* and *routes_eth0* to enter IP address information and routing information:

Note
This assumes that the network interface will be called eth0. This is, however, very system dependent. It is recommended to assume that the interface is named the same as the interface name when booted from the installation media if the installation media is sufficiently recent. More information can be found in the Network interface naming (/wiki/Handbook:AMD64/Networking/Advanced#Network_interface_naming) section.

FILE `/etc/conf.d/net` **Static IP definition**

```
config_eth0="192.168.0.2 netmask 255.255.255.0 brd 192.168.0.255"
routes_eth0="default via 192.168.0.1"
```

To use DHCP, define *config_eth0*:

FILE `/etc/conf.d/net` **DHCP definition**

```
config_eth0="dhcp"
```

Please read `/usr/share/doc/netifrc-*/net.example.bz2` for a list of additional configuration options. Be sure to also read up on the DHCP client man page if specific DHCP options need to be set.

If the system has several network interfaces, then repeat the above steps for *config_eth1*, *config_eth2*, etc.

Now save the configuration and exit to continue.

Automatically start networking at boot

To have the network interfaces activated at boot, they need to be added to the default runlevel.

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

If the system has several network interfaces, then the appropriate `net.*` files need to be created just like we did with `net.eth0`.

If, after booting the system, it is discovered the network interface name (which is currently documented as `eth0`) was wrong, then execute the following steps to rectify:

1. Update the `/etc/conf.d/net` file with the correct interface name (like `enp3s0` or `enp5s0`, instead of `eth0`).
2. Create new symbolic link (like `/etc/init.d/net.enp3s0`).
3. Remove the old symbolic link (**`rm /etc/init.d/net.eth0`**).
4. Add the new one to the default runlevel.
5. Remove the old one using **`rc-update del net.eth0 default`**.

The hosts file

An important next step may be to inform this new system about other hosts in its network environment. Network host names can be defined in the `/etc/hosts` file. Adding host names here will enable host name to IP addresses resolution for hosts that are not resolved by the nameserver.

```
root # nano /etc/hosts
```

FILE `/etc/hosts` **Filling in the networking information**

```
# This defines the current system and must be set
127.0.0.1     tux.homenetwork tux localhost

# Optional definition of extra systems on the network
192.168.0.5   jenny.homenetwork jenny
192.168.0.6   benny.homenetwork benny
```

Save and exit the editor to continue.

System information

Root password

Set the root password using the **passwd** command.

```
root # passwd
```

Later an additional regular user account will be created for daily operations.

Init and boot configuration

OpenRC

When using OpenRC with Gentoo, it uses `/etc/rc.conf` to configure the services, startup, and shutdown of a system. Open up `/etc/rc.conf` and enjoy all the comments in the file. Review the settings and change where needed.

```
root # nano /etc/rc.conf
```

Next, open `/etc/conf.d/keymaps` to handle keyboard configuration. Edit it to configure and select the right keyboard.

```
root # nano /etc/conf.d/keymaps
```

Take special care with the *keymap* variable. If the wrong keymap is selected, then weird results will come up when typing on the keyboard.

Finally, edit `/etc/conf.d/hwclock` to set the clock options. Edit it according to personal preference.

```
root # nano /etc/conf.d/hwclock
```

If the hardware clock is not using UTC, then it is necessary to set `clock="local"` in the file. Otherwise the system might show clock skew behavior.

systemd

First, it is recommended to run **systemd-firstboot** which will prepare various components of the system are set correctly for the first boot into the new systemd environment. The passing the following options will include a prompt for the user to set a locale, timezone, hostname, root password, and root shell values. It will also assign a random machine ID to the installation:

```
root # systemd-firstboot --prompt --setup-machine-id
```

Next users should run **systemctl** to reset all installed unit files to the preset policy values:

```
root # systemctl preset-all --preset-mode=enable-only
```

It's possible to run the full preset changes but this may reset any services which were already configured during the process:

```
root # systemctl preset-all
```

These two steps will help ensure a smooth transition from the live environment to the installation's first boot.

System logger

OpenRC

Some tools are missing from the stage3 archive because several packages provide the same functionality. It is now up to the user to choose which ones to install.

The first tool to decision is a logging mechanism for the system. Unix and Linux have an excellent history of logging capabilities - if needed, everything that happens on the system can be logged in a log file.

Gentoo offers several system logger utilities. A few of these include:

- app-admin/sysklogd (https://packages.gentoo.org/packages/app-admin/sysklogd) - Offers the traditional set of system logging daemons. The default logging configuration works well out of the box which makes this package a good option for beginners.
- app-admin/syslog-ng (https://packages.gentoo.org/packages/app-admin/syslog-ng) - An advanced system logger. Requires additional configuration for anything beyond logging to one big file. More advanced users may choose this package based on its logging potential; be aware additional configuration is a necessity for any kind of smart logging.
- app-admin/metalog (https://packages.gentoo.org/packages/app-admin/metalog) - A highly-configurable system logger.

There may be other system logging utilities available through the Gentoo ebuild repository as well, since the number of available packages increases on a daily basis.

Tip
If syslog-ng is going to be used, it is recommended to install and configure logrotate (/wiki/Logrotate). syslog-ng does not provide any rotation mechanism for the log files. Newer versions (>= 2.0) of sysklogd however handle their own log rotation.

To install the system logger of choice, emerge it. On OpenRC, add it to the default runlevel using **rc-update**. The following example installs and activates app-admin/sysklogd (https://packages.gentoo.org/packages/app-admin/sysklogd) as the system's syslog utility:

```
root # emerge --ask app-admin/sysklogd
root # rc-update add sysklogd default
```

systemd

While a selection of logging mechanisms are presented for OpenRC-based systems, systemd includes a built-in logger called the **systemd-journald** service. The systemd-journald service is capable of handling most of the logging functionality outlined in the previous system logger section. That is to say, the majority of installations that will run systemd as the system and service manager can *safely skip* adding a additional syslog utilities.

See **man journalctl** for more details on using **journalctl** to query and review the systems logs.

For a number of reasons, such as the case of forwarding logs to a central host, it may be important to include *redundant* system logging mechanisms on a systemd-based system. This is a irregular occurrence for the handbook's typical audience and considered an advanced use case. It is therefore not covered by the handbook.

Optional: Cron daemon

OpenRC

Although it is optional and not required for every system, it is wise to install a cron daemon.

A cron daemon executes commands on scheduled intervals. Internals could be daily, weekly, or monthly, once every Tuesday, once every other week, etc. A wise system administrator will leverage the cron daemon to automate routine system maintenance tasks.

All cron daemons support high levels of granularity for scheduled tasks, and generally include the ability to send an email or other form of notification if a scheduled task does not complete as expected.

Gentoo offers several possible cron daemons, including:

- sys-process/cronie (https://packages.gentoo.org/packages/sys-process/cronie) - cronie is based on the original cron and has security and configuration enhancements like the ability to use PAM and SELinux.
- sys-process/dcron (https://packages.gentoo.org/packages/sys-process/dcron) - This lightweight cron daemon aims to be simple and secure, with just enough features to stay useful.
- sys-process/fcron (https://packages.gentoo.org/packages/sys-process/fcron) - A command scheduler with extended capabilities over cron and anacron.
- sys-process/bcron (https://packages.gentoo.org/packages/sys-process/bcron) - A younger cron system designed with secure operations in mind. To do this, the system is divided into several separate programs, each responsible for a separate task, with strictly controlled communications between parts.

cronie

The following example uses sys-process/cronie (https://packages.gentoo.org/packages/sys-process/cronie):

```
root # emerge --ask sys-process/cronie
```

Add cronie to the default system runlevel, which will automatically start it on power up:

```
root # rc-update add cronie default
```

Alternative: dcron

```
root # emerge --ask sys-process/dcron
```

If dcron is the go forward cron agent, an additional initialization command needs to be executed:

```
root # crontab /etc/crontab
```

Alternative: fcron

```
root # emerge --ask sys-process/fcron
```

If fcron is the selected scheduled task handler, an additional emerge step is required:

```
root # emerge --config sys-process/fcron
```

Alternative: bcron

bcron is a younger cron agent with built-in privilege separation.

```
root # emerge --ask sys-process/bcron
```

systemd

Similar to system logging, systemd-based systems include support for scheduled tasks out-of-the-box in the form of *timers*. systemd timers can run at a system-level or a user-level and include the same functionality that a traditional cron daemon would provide. Unless redundant capabilities are necessary, installing an additional task scheduler such as a cron daemon is generally unnecessary and can be safely skipped.

Optional: File indexing

In order to index the file system to provide faster file location capabilities, install sys-apps/mlocate (https://packages.gentoo.org/packages/sys-apps/mlocate).

```
root # emerge --ask sys-apps/mlocate
```

Optional: Remote shell access

Tip
opensshd's default configuration does not allow root to login as a remote user. Please create a non-root user (/wiki/FAQ#How_do_I_add_a_normal_user.3F) and configure it appropriately to allow access post-installation if required, or adjust `/etc/ssh/sshd_config` to allow root.

To be able to access the system remotely after installation, **sshd** must be configured to start on boot.

OpenRC

To add the `sshd` init script to the default runlevel on OpenRC:

```
root # rc-update add sshd default
```

If serial console access is needed (which is possible in case of remote servers), **agetty** must be configured.

Uncomment the serial console section in `/etc/inittab`:

```
root # nano -w /etc/inittab
```

```
# SERIAL CONSOLES
s0:12345:respawn:/sbin/agetty 9600 ttyS0 vt100
s1:12345:respawn:/sbin/agetty 9600 ttyS1 vt100
```

systemd

To enable the SSH server, run:

```
root # systemctl enable sshd
```

To enable serial console support, run:

```
root # systemctl enable getty@tty1.service
```

Time synchronization

It is important to use some method of synchronizing the system clock. This is usually done via the NTP (/wiki/NTP) protocol and software. Other implementations using the NTP protocol exist, like Chrony (/wiki/Chrony).

To set up Chrony, for example:

```
root # emerge --ask net-misc/chrony
```

OpenRC

On OpenRC, run:

```
root # rc-update add chronyd default
```

systemd

On systemd, run:

```
root # systemctl enable chronyd.service
```

Alternatively, systemd users may wish to use the simpler **systemd-timesyncd** SNTP client which is installed by default.

```
root # systemctl enable systemd-timesyncd.service
```

Filesystem tools

Depending on the filesystems used, it may be necessary to install the required file system utilities (for checking the filesystem integrity, (re)formatting file systems, etc.). Note that ext4 user space tools (sys-fs/e2fsprogs (https://packages.gentoo.org/packages/sys-fs/e2fsprogs) are already installed as a part of the @system set (/wiki/System_set_(Portage)).

The following table lists the tools to install if a certain filesystem tools will be needed in the installed environment.

Filesystem	Package
XFS	sys-fs/xfsprogs (https://packages.gentoo.org/packages/sys-fs/xfsprogs)
ext4	sys-fs/e2fsprogs (https://packages.gentoo.org/packages/sys-fs/e2fsprogs)
VFAT (FAT32, ...)	sys-fs/dosfstools (https://packages.gentoo.org/packages/sys-fs/dosfstools)
Btrfs	sys-fs/btrfs-progs (https://packages.gentoo.org/packages/sys-fs/btrfs-progs)
ZFS	sys-fs/zfs (https://packages.gentoo.org/packages/sys-fs/zfs)
JFS	sys-fs/jfsutils (https://packages.gentoo.org/packages/sys-fs/jfsutils)
ReiserFS	sys-fs/reiserfsprogs (https://packages.gentoo.org/packages/sys-fs/reiserfsprogs)

It's recommended that sys-block/io-scheduler-udev-rules (https://packages.gentoo.org/packages/sys-block/io-scheduler-udev-rules) is installed for the correct scheduler behavior with e.g. nvme devices:

```
root # emerge --ask sys-block/io-scheduler-udev-rules
```

Tip
For more information on filesystems in Gentoo see the filesystem article (/wiki/Filesystem).

Networking tools

If networking was previously configured in the Configuring the system (/wiki/Handbook:AMD64/Installation/System) step and network setup is complete, then this 'networking tools' section can be safely skipped. In this case, proceed with the section on Configuring a bootloader (/wiki/Handbook:AMD64/Installation/Bootloader).

Installing a DHCP client

Important
Most users will need a DHCP client to connect to their network. If none was installed, then the system might not be able to get on the network thus making it impossible to download a DHCP client afterwards.

A DHCP client obtains automatically an IP address for one or more network interface(s) using netifrc scripts. We recommend the use of net-misc/dhcpcd (https://packages.gentoo.org/packages/net-misc/dhcpcd) (see also dhcpcd (/wiki/Dhcpcd)):

```
root # emerge --ask net-misc/dhcpcd
```

Optional: Installing a PPPoE client

If PPP is used to connect to the internet, install the net-dialup/ppp (https://packages.gentoo.org/packages/net-dialup/ppp) package:

```
root # emerge --ask net-dialup/ppp
```

Optional: Install wireless networking tools

If the system will be connecting to wireless networks, install the net-wireless/iw (https://packages.gentoo.org/packages/net-wireless/iw) package for Open or WEP networks and/or the net-wireless/wpa_supplicant (https://packages.gentoo.org/packages/net-wireless/wpa_supplicant) package for WPA or WPA2 networks. **iw** is also a useful basic diagnostic tool for scanning wireless networks.

```
root # emerge --ask net-wireless/iw net-wireless/wpa_supplicant
```

Now continue with Configuring the bootloader (/wiki/Handbook:AMD64/Installation/Bootloader).

Selecting a boot loader

With the Linux kernel configured, system tools installed and configuration files edited, it is time to install the last important piece of a Linux installation: the boot loader.

The boot loader is responsible for firing up the Linux kernel upon boot - without it, the system would not know how to proceed when the power button has been pressed.

For **amd64**, we document how to configure either GRUB or LILO for BIOS based systems, and GRUB or efibootmgr for UEFI systems.

In this section of the Handbook a delineation has been made between *emerging* the boot loader's package and *installing* a boot loader to a system disk. Here the term *emerge* will be used to ask Portage (/wiki/Portage) to make the software package available to the system. The term *install* will signify the boot loader copying files or physically modifying appropriate sections of the system's disk drive in order to render the boot loader *activated and ready to operate* on the next power cycle.

Default: GRUB

By default, the majority of Gentoo systems now rely upon GRUB (/wiki/GRUB) (found in the sys-boot/grub (https://packages.gentoo.org/packages/sys-boot/grub) package), which is the direct successor to GRUB Legacy (/wiki/GRUB_Legacy). With no additional configuration, GRUB gladly supports older BIOS ("pc") systems. With a small amount of configuration, necessary before build time, GRUB can support more than a half a dozen additional platforms. For more information, consult the Prerequisites section (/wiki/GRUB#Prerequisites) of the GRUB (/wiki/GRUB) article.

Emerge

When using an older BIOS system supporting only MBR partition tables, no additional configuration is needed in order to emerge GRUB:

```
root # emerge --ask --verbose sys-boot/grub
```

A note for UEFI users: running the above command will output the enabled *GRUB_PLATFORMS* values before emerging. When using UEFI capable systems, users will need to ensure `GRUB_PLATFORMS="efi-64"` is enabled (as it is the case by default). If that is not the case for the setup, `GRUB_PLATFORMS="efi-64"` will need to be added to the `/etc/portage/make.conf` file *before* emerging GRUB so that the package will be built with EFI functionality:

```
root # echo 'GRUB_PLATFORMS="efi-64"' >> /etc/portage/make.conf
```

```
root # emerge --ask sys-boot/grub
```

If GRUB was somehow emerged without enabling `GRUB_PLATFORMS="efi-64"`, the line (as shown above) can be added to `make.conf` and then dependencies for the world package set (/wiki/World_set_(Portage)) can be re-calculated by passing the `--update --newuse` options to **emerge**:

```
root # emerge --ask --update --newuse --verbose sys-boot/grub
```

The GRUB software has now been merged to the system, but not yet installed.

Install

Next, install the necessary GRUB files to the `/boot/grub/` directory via the **grub-install** command. Presuming the first disk (the one where the system boots from) is `/dev/sda`, one of the following commands will do:

- When using BIOS:

```
root # grub-install /dev/sda
```

- When using UEFI:

Important
Make sure the EFI system partition has been mounted *before* running **grub-install**. It is possible for **grub-install** to install the GRUB EFI file (grubx64.efi) into the wrong directory **without** providing *any* indication the wrong directory was used.

```
root # grub-install --target=x86_64-efi --efi-directory=/boot
```

Note
Modify the `--efi-directory` option to the root of the EFI System Partition. This is necessary if the `/boot` partition was not formatted as a FAT variant.

Important
If **grub-install** returns an error like `Could not prepare Boot variable: Read-only file system`, it may be necessary to remount the efivars special mount as read-write in order to succeed:

```
root # mount -o remount,rw,nosuid,nodev,noexec --types efivarfs efivarfs /sys/firmware/efi/efivars
```

Some motherboard manufacturers seem to only support the `/efi/boot/` directory location for the .EFI file in the EFI System Partition (ESP). The GRUB installer can perform this operation automatically with the `--removable` option. Verify the ESP is mounted before running the following commands. Presuming the ESP is mounted at `/boot` (as suggested earlier), execute:

```
root # grub-install --target=x86_64-efi --efi-directory=/boot --removable
```

This creates the default directory defined by the UEFI specification, and then copies the grubx64.efi file to the 'default' EFI file location defined by the same specification.

Configure

Next, generate the GRUB configuration based on the user configuration specified in the `/etc/default/grub` file and `/etc/grub.d` scripts. In most cases, no configuration is needed by users as GRUB will automatically detect which kernel to boot (the highest one available in `/boot/`) and what the root file system is. It is also possible to append kernel parameters in `/etc/default/grub` using the *GRUB_CMDLINE_LINUX* variable.

To generate the final GRUB configuration, run the **grub-mkconfig** command:

```
root # grub-mkconfig -o /boot/grub/grub.cfg
```

```
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-6.1.38-gentoo
Found initrd image: /boot/initramfs-genkernel-amd64-6.1.38-gentoo
done
```

The output of the command must mention that at least one Linux image is found, as those are needed to boot the system. If an initramfs is used or **genkernel** was used to build the kernel, the correct initrd image should be detected as well. If this is not the case, go to `/boot/` and check the contents using the **ls** command. If the files are indeed missing, go back to the kernel configuration and installation instructions.

Tip
The **os-prober** utility can be used in conjunction with GRUB to detect other operating systems from

attached drives. Windows 7, 8.1, 10, and other distributions of Linux are detectable. Those desiring dual boot systems should emerge the sys-boot/os-prober (https://packages.gentoo.org/packages/sys-boot/os-prober) package then re-run the **grub-mkconfig** command (as seen above). If detection problems are encountered be sure to read the GRUB (/wiki/GRUB) article in its entirety *before* asking the Gentoo community for support.

Alternative 1: LILO

Emerge

LILO, the **LI**nux**LO**ader, is the tried and true workhorse of Linux boot loaders. However, it lacks features when compared to GRUB. LILO is still used because, on some systems, GRUB does not 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 bootloaders.

Installing LILO is a breeze; just use emerge.

```
root # emerge --ask sys-boot/lilo
```

Configure

To configure LILO, first create `/etc/lilo.conf`:

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

In the configuration file, sections are used to refer to the bootable kernel. Make sure that the kernel files (with kernel version) and initramfs files are known, as they need to be referred to in this configuration file.

Note
If the root filesystem is JFS, add an `append="ro"` line after each boot item since JFS needs to replay its log before it allows read-write mounting.

FILE `/etc/lilo.conf` **Example LILO configuration**

```
boot=/dev/sda             # 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
compact                   # This drastically reduces load time and keeps the map file
smaller; may fail on some systems

image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo            # Name we give to this section
  read-only               # Start with a read-only root. Do not alter!
  root=/dev/sda3          # Location of the root filesystem

image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo.rescue     # Name we give to this section
  read-only               # Start with a read-only root. Do not alter!
  root=/dev/sda3         # Location of the root filesystem
  append="init=/bin/bb"   # Launch the Gentoo static rescue shell

# The next two lines are for dual booting with a Windows system.
# In this example, Windows is hosted on /dev/sda6.
other=/dev/sda6
  label=windows
```

Note
If a different partitioning scheme and/or kernel image is used, adjust accordingly.

If an initramfs is necessary, then change the configuration by referring to this initramfs file and telling the initramfs where the root device is located:

FILE `/etc/lilo.conf` **Adding initramfs information to a boot entry**

```
image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo
  read-only
  append="root=/dev/sda3"
  initrd=/boot/initramfs-genkernel-amd64-6.1.38-gentoo
```

If additional options need to be passed to the kernel, use an `append` statement. For instance, to add the `video` statement to enable framebuffer:

FILE `/etc/lilo.conf` **Adding video parameter to the boot options**

```
image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo
  read-only
  root=/dev/sda3
  append="video=uvesafb:mtrr,ywrap,1024x768-32@85"
```

Users that used **genkernel** should know that their kernels use the same boot options as is used for the installation CD. For instance, if SCSI device support needs to be enabled, add `doscsi` as kernel option.

Now save the file and exit.

Install

To finish up, run the `/sbin/lilo` executable so LILO can apply the `/etc/lilo.conf` settings to the system (i.e. install itself on the disk). Keep in mind that `/sbin/lilo` must be executed each time a new kernel is installed or a change has been made to the `lilo.conf` file in order for the system to boot if the filename of the kernel has changed.

```
root # /sbin/lilo
```

Alternative 2: efibootmgr

On UEFI based systems, the UEFI firmware on the system (in other words the primary bootloader), can be directly manipulated to look for UEFI boot entries. Such systems do not need to have additional (also known as secondary) bootloaders like GRUB in order to help boot the system. With that being said, the reason EFI-based bootloaders such as GRUB exist is to *extend* the functionality of UEFI systems during the boot process. Using **efibootmgr** is really for those who desire to take a minimalist (although more rigid) approach to booting their system; using GRUB (see above) is easier for the majority of users because it offers a flexible approach when booting UEFI systems.

Remember sys-boot/efibootmgr (https://packages.gentoo.org/packages/sys-boot/efibootmgr) application is not a bootloader; it is a tool to interact with the UEFI firmware and update its settings, so that the Linux kernel that was previously installed can be booted with additional options (if necessary), or to allow multiple boot entries. This interaction is done through the EFI variables (hence the need for kernel support of EFI vars).

Be sure to read through the EFI stub (/wiki/EFI_stub) kernel article *before* continuing. The kernel must have specific options enabled to be directly bootable by the system's UEFI firmware. It might be necessary to recompile the kernel. It is also a good idea to take a look at the **efibootmgr (/wiki/Efibootmgr)** article.

> **Note**
> To reiterate, **efibootmgr** is *not* a requirement to boot an UEFI system. The Linux kernel itself can be booted immediately, and additional kernel command-line options can be built-in to the Linux kernel (there is a kernel configuration option called *CONFIG_CMDLINE* that allows the user to specify boot parameters as command-line options. Even an initramfs can be 'built-in' to the kernel.

Those that have decided to take this approach must install the software:

```
root # emerge --ask sys-boot/efibootmgr
```

Then, create the `/boot/efi/boot/` location, and then copy the kernel into this location, calling it `bootx64.efi`:

```
root # mkdir -p /boot/efi/boot
root # cp /boot/vmlinuz-* /boot/efi/boot/bootx64.efi
```

Next, tell the UEFI firmware that a boot entry called "Gentoo" is to be created, which has the freshly compiled EFI stub kernel:

```
root # efibootmgr --create --disk /dev/sda --part 2 --label "Gentoo" --loader
"\efi\boot\bootx64.efi"
```

If an initial RAM file system (initramfs) is used, add the proper boot option to it:

```
root # efibootmgr -c -d /dev/sda -p 2 -L "Gentoo" -l "\efi\boot\bootx64.efi"
initrd='\initramfs-genkernel-amd64-6.1.38-gentoo'
```

Note
The use of `\` as directory separator is mandatory when using UEFI definitions.

With these changes done, when the system reboots, a boot entry called "Gentoo" will be available.

Alternative 3: Syslinux

Syslinux is yet another bootloader alternative for the **amd64** architecture. It supports MBR and, as of version 6.00, it supports EFI boot. PXE (network) boot and lesser-known options are also supported. Although Syslinux is a popular bootloader for many it is unsupported by the Handbook. Readers can find information on emerging and then installing this bootloader in the Syslinux (/wiki/Syslinux) article.

Rebooting the system

Exit the chrooted environment and unmount all mounted partitions. Then type in that one magical command that initiates the final, true test: **reboot**.

```
root # exit
cdimage ~# cd
cdimage ~# umount -l /mnt/gentoo/dev{/shm,/pts,}
cdimage ~# umount -R /mnt/gentoo
cdimage ~# reboot
```

Do not forget to remove the bootable CD, otherwise the CD might be booted again instead of the new Gentoo system.

Once rebooted in the freshly installed Gentoo environment, finish up with Finalizing the Gentoo installation (/wiki/Handbook:AMD64/Installation/Finalizing).

User administration

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.

The groups the user is member of define what activities the user can perform. The following table lists a number of important groups:

Group	Description
audio	Be able to access the audio devices.
cdrom	Be able to directly access optical devices.
floppy	Be able to directly access floppy devices.
games	Be able to play games.

portage	Be able to access portage restricted resources.
usb	Be able to access USB devices.
video	Be able to access video capturing hardware and doing hardware acceleration.
wheel	Be able to use **su**.

For instance, to create a user called larry (/wiki/User:Larry) who is member of the *wheel*, *users*, and *audio* groups, log in as root first (only root can create users) and run **useradd**:

```
Login: root
```

```
Password: (Enter the root password)
```

```
root # useradd -m -G users,wheel,audio -s /bin/bash larry
root # passwd larry
```

```
Password: (Enter the password for larry)
Re-enter password: (Re-enter the password to verify)
```

If a user ever needs to perform some task as root, they can use **su -** to temporarily receive root privileges. Another way is to use the **sudo (/wiki/Sudo)** (app-admin/sudo (https://packages.gentoo.org/packages/app-admin/sudo)) or **doas** (app-admin/doas (https://packages.gentoo.org/packages/app-admin/doas)) utilities which are, if correctly configured, very secure.

Disk cleanup

Removing tarballs

With the Gentoo installation finished and the system rebooted, if everything has gone well, we can now remove the downloaded stage3 tarball from the hard disk. Remember that they were downloaded to the `/` directory.

```
root # rm /stage3-*.tar.*
```

Where to go from here

Not sure where to go from here? There are many paths to explore... Gentoo provides its users with lots of possibilities and therefore has lots of documented (and less documented) features to explore here on the wiki and on other Gentoo related sub-domains (see the Gentoo online (/wiki/Handbook:AMD64/Installation/Finalizing#Gentoo_online) section below).

Additional documentation

It is important to note that, due to the number of choices available in Gentoo, the documentation provided by the handbook is limited in scope - it mainly focuses on the basics of getting a Gentoo system up and running and basic system management activities. The handbook intentionally excludes instructions on graphical environments, details on hardening, and other important administrative tasks. That being stated, there are more sections of the handbook to assist readers with more basic functions.

Readers should definitely take a look at the next part of the handbook entitled Working with Gentoo (/wiki/Handbook:AMD64/Working/Portage) which explains how to keep the software up to date, install additional software packages, details on USE flags, the OpenRC init system, and various other informative topics relating to managing a Gentoo system post-installation.

Apart from the handbook, readers should also feel encouraged to explore other corners of the Gentoo wiki to find additional, community-provided documentation. The Gentoo wiki team also offers a documentation topic overview (/wiki/Main_Page#Documentation_topics) which lists a selection of wiki articles by category. For instance, it refers to the localization guide (/wiki/Localization/Guide) to make a system feel more at home (particularly useful for users who speak English as a second language).

The majority of users with desktop use cases will setup graphical environments in which to work natively. There are many community maintained 'meta' articles for supported desktop environments (DEs) (/wiki/Desktop_environment) and window managers (WMs) (/wiki/Window_manager). Readers should be aware that each DE will require slightly different setup steps, which will lengthen add complexity to bootstrapping.

Many other Meta articles (/wiki/Category:Meta) exist to provide our readers with high level overviews of available software within Gentoo.

Gentoo online

Important
Readers should note that all official Gentoo sites online are governed by Gentoo's code of conduct (/wiki/Project:Council/Code_of_conduct). Being active in the Gentoo community is a privilege, not a right, and users should be aware that the code of conduct exists for a reason.

With the exception of the Libera.Chat hosted internet relay chat (IRC) network and the mailing lists, most Gentoo websites require an account on a per site basis in order to ask questions, open a discussion, or enter a bug.

Forums and IRC

Every user is welcome on our Gentoo forums (https://forums.gentoo.org) or on one of our internet relay chat channels (https://www.gentoo.org/get-involved/irc-channels/). It is easy to search for the forums to see if an issue experienced on a fresh Gentoo install has been discovered in the past and resolved after some feedback. The likelihood of other users experiencing the installation issues by first-time Gentoo can be surprising. It is advised users search the forums and the wiki before asking for assistance in Gentoo support channels.

Mailing lists

Several mailing lists (https://www.gentoo.org/get-involved/mailing-lists/) are available to the community members who prefer to ask for support or feedback over email rather than create a user account on the forums or IRC. Users will need to follow the instructions in order to subscribe to specific mailing lists.

Bugs

Sometimes after reviewing the wiki, searching the forums, and seeking support in the IRC channel or mailing lists there is no known solution to a problem. Generally this is a sign to open a bug on Gentoo's Bugzilla site (https://bugs.gentoo.org).

Development guide

Readers who desire to learn more about developing Gentoo can take a look at the Development guide (https://devmanual.gentoo.org/). This guide provides instructions on writing ebuilds, working with eclasses, and provides definitions for many general concepts (https://devmanual.gentoo.org/general-concepts/index.html) behind Gentoo development.

Closing thoughts

Gentoo is a robust, flexible, and excellently maintained distribution. The developer community is happy to hear feedback on how to make Gentoo an even *better* distribution.

As a reminder, any feedback for *this handbook* should follow the guidelines detailed in the How do I improve the Handbook? (/wiki/Handbook:Main_Page#How_do_I_improve_the_Handbook.3F) section at the beginning of the handbook.

We look forward to seeing how our users will choose to implement Gentoo to fit their unique use cases and needs.