the GRUB manual

The GRand Unified Bootloader, version 0.96, 20 September 2004.

Gordon Matzigkeit
Yoshinori K. Okuji

Copyright © 1999,2000,2001,2002,2004 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies. Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by Free Software Foundation.

Table of Contents

1 Introduction to GRUB

1.1 Overview

Briefly, a *boot loader* is the first software program that runs when a computer starts. It is responsible for loading and transferring control to an operating system *kernel* software (such as Linux or GNU Mach). The kernel, in turn, initializes the rest of the operating system (e.g. a GNU system).

GNU GRUB is a very powerful boot loader, which can load a wide variety of free operating systems, as well as proprietary operating systems with chain-loading[1]. GRUB is designed to address the complexity of booting a personal computer; both the program and this manual are tightly bound to that computer platform, although porting to other platforms may be addressed in the future.

One of the important features in GRUB is flexibility; GRUB understands filesystems and kernel executable formats, so you can load an arbitrary operating system the way you like, without recording the physical position of your kernel on the disk. Thus you can load the kernel just by specifying its file name and the drive and partition where the kernel resides.

When booting with GRUB, you can use either a command-line interface (see Section 12.1 [Command-line interface], page 33), or a menu interface (see Section 12.2 [Menu interface], page 34). Using the command-line interface, you type the drive specification and file name of the kernel manually. In the menu interface, you just select an OS using the arrow keys. The menu is based on a configuration file which you prepare beforehand (see Chapter 5 [Configuration], page 17). While in the menu, you can switch to the command-line mode, and vice-versa. You can even edit menu entries before using them.

In the following chapters, you will learn how to specify a drive, a partition, and a file name (see Chapter 2 [Naming convention], page 5) to GRUB, how to install GRUB on your drive (see Chapter 3 [Installation], page 7), and how to boot your OSes (see Chapter 4 [Booting], page 11), step by step.

Besides the GRUB boot loader itself, there is a *grub shell* `grub` (see Chapter 15 [Invoking the grub shell], page 55) which can be run when you are in your operating system. It emulates the boot loader and can be used for installing the boot loader.

1.2 History of GRUB

GRUB originated in 1995 when Erich Boleyn was trying to boot the GNU Hurd with the University of Utah's Mach 4 microkernel (now known as GNU Mach). Erich and Brian Ford designed the Multiboot Specification (see section "Motivation" in *The Multiboot Specification*), because they were determined not to add to the large number of mutually-incompatible PC boot methods.

Erich then began modifying the FreeBSD boot loader so that it would understand Multiboot. He soon realized that it would be a lot easier to write his own boot loader from scratch than to keep working on the FreeBSD boot loader, and so GRUB was born.

---

[1] *chain-load* is the mechanism for loading unsupported operating systems by loading another boot loader. It is typically used for loading DOS or Windows.

Erich added many features to GRUB, but other priorities prevented him from keeping up with the demands of its quickly-expanding user base. In 1999, Gordon Matzigkeit and Yoshinori K. Okuji adopted GRUB as an official GNU package, and opened its development by making the latest sources available via anonymous CVS. See Appendix A [Obtaining and Building GRUB], page 69, for more information.

1.3 GRUB features

The primary requirement for GRUB is that it be compliant with the *Multiboot Specification*, which is described in section "Motivation" in *The Multiboot Specification*.

The other goals, listed in approximate order of importance, are:

- Basic functions must be straightforward for end-users.
- Rich functionality to support kernel experts and designers.
- Backward compatibility for booting FreeBSD, NetBSD, OpenBSD, and Linux. Proprietary kernels (such as DOS, Windows NT, and OS/2) are supported via a chain-loading function.

Except for specific compatibility modes (chain-loading and the Linux *piggyback* format), all kernels will be started in much the same state as in the Multiboot Specification. Only kernels loaded at 1 megabyte or above are presently supported. Any attempt to load below that boundary will simply result in immediate failure and an error message reporting the problem.

In addition to the requirements above, GRUB has the following features (note that the Multiboot Specification doesn't require all the features that GRUB supports):

Recognize multiple executable formats
: Support many of the *a.out* variants plus *ELF*. Symbol tables are also loaded.

Support non-Multiboot kernels
: Support many of the various free 32-bit kernels that lack Multiboot compliance (primarily FreeBSD, NetBSD, OpenBSD, and Linux). Chain-loading of other boot loaders is also supported.

Load multiples modules
: Fully support the Multiboot feature of loading multiple modules.

Load a configuration file
: Support a human-readable text configuration file with preset boot commands. You can also load another configuration file dynamically and embed a preset configuration file in a GRUB image file. The list of commands (see Chapter 13 [Commands], page 35) are a superset of those supported on the command-line. An example configuration file is provided in Chapter 5 [Configuration], page 17.

Provide a menu interface
: A menu interface listing preset boot commands, with a programmable timeout, is available. There is no fixed limit on the number of boot entries, and the current implementation has space for several hundred.

Have a flexible command-line interface
: A fairly flexible command-line interface, accessible from the menu, is available to edit any preset commands, or write a new boot command set from scratch. If no configuration file is present, GRUB drops to the command-line.

  The list of commands (see Chapter 13 [Commands], page 35) are a subset of those supported for configuration files. Editing commands closely resembles the Bash command-line (see section "Command Line Editing" in *Bash Features*), with ⟨TAB⟩-completion of commands, devices, partitions, and files in a directory depending on context.

Support multiple filesystem types
: Support multiple filesystem types transparently, plus a useful explicit blocklist notation. The currently supported filesystem types are *BSD FFS*, *DOS FAT16 and FAT32*, *Minix fs*, *Linux ext2fs*, *ReiserFS*, *JFS*, *XFS*, and *VSTa fs*. See Chapter 11 [Filesystem], page 31, for more information.

Support automatic decompression
: Can decompress files which were compressed by `gzip`. This function is both automatic and transparent to the user (i.e. all functions operate upon the uncompressed contents of the specified files). This greatly reduces a file size and loading time, a particularly great benefit for floppies.[2]

  It is conceivable that some kernel modules should be loaded in a compressed state, so a different module-loading command can be specified to avoid uncompressing the modules.

Access data on any installed device
: Support reading data from any or all floppies or hard disk(s) recognized by the BIOS, independent of the setting of the root device.

Be independent of drive geometry translations
: Unlike many other boot loaders, GRUB makes the particular drive translation irrelevant. A drive installed and running with one translation may be converted to another translation without any adverse effects or changes in GRUB's configuration.

Detect all installed RAM
: GRUB can generally find all the installed RAM on a PC-compatible machine. It uses an advanced BIOS query technique for finding all memory regions. As described on the Multiboot Specification (see section "Motivation" in *The Multiboot Specification*), not all kernels make use of this information, but GRUB provides it for those who do.

Support Logical Block Address mode
: In traditional disk calls (called *CHS mode*), there is a geometry translation problem, that is, the BIOS cannot access over 1024 cylinders, so the accessible space is limited to at least 508 MB and to at most 8GB. GRUB can't universally solve this problem, as there is no standard interface used in all machines.

[2] There are a few pathological cases where loading a very badly organized ELF kernel might take longer, but in practice this never happen.

However, several newer machines have the new interface, Logical Block Address (*LBA*) mode. GRUB automatically detects if LBA mode is available and uses it if available. In LBA mode, GRUB can access the entire disk.

Support network booting
: GRUB is basically a disk-based boot loader but also has network support. You can load OS images from a network by using the *TFTP* protocol.

Support remote terminals
: To support computers with no console, GRUB provides remote terminal support, so that you can control GRUB from a remote host. Only serial terminal support is implemented at the moment.

1.4 The role of a boot loader

The following is a quotation from Gordon Matzigkeit, a GRUB fanatic:

> Some people like to acknowledge both the operating system and kernel when they talk about their computers, so they might say they use "GNU/Linux" or "GNU/Hurd". Other people seem to think that the kernel is the most important part of the system, so they like to call their GNU operating systems "Linux systems."
>
> I, personally, believe that this is a grave injustice, because the *boot loader* is the most important software of all. I used to refer to the above systems as either "LILO"[3] or "GRUB" systems.
>
> Unfortunately, nobody ever understood what I was talking about; now I just use the word "GNU" as a pseudonym for GRUB.
>
> So, if you ever hear people talking about their alleged "GNU" systems, remember that they are actually paying homage to the best boot loader around... GRUB!

We, the GRUB maintainers, do not (usually) encourage Gordon's level of fanaticism, but it helps to remember that boot loaders deserve recognition. We hope that you enjoy using GNU GRUB as much as we did writing it.

[3] The LInux LOader, a boot loader that everybody uses, but nobody likes.

2 Naming convention

The device syntax used in GRUB is a wee bit different from what you may have seen before in your operating system(s), and you need to know it so that you can specify a drive/partition.

Look at the following examples and explanations:

`(fd0)`

First of all, GRUB requires that the device name be enclosed with '`(`' and '`)`'. The '`fd`' part means that it is a floppy disk. The number '`0`' is the drive number, which is counted from *zero*. This expression means that GRUB will use the whole floppy disk.

`(hd0,1)`

Here, '`hd`' means it is a hard disk drive. The first integer '`0`' indicates the drive number, that is, the first hard disk, while the second integer, '`1`', indicates the partition number (or the PC slice number in the BSD terminology). Once again, please note that the partition numbers are counted from *zero*, not from one. This expression means the second partition of the first hard disk drive. In this case, GRUB uses one partition of the disk, instead of the whole disk.

`(hd0,4)`

This specifies the first *extended partition* of the first hard disk drive. Note that the partition numbers for extended partitions are counted from '`4`', regardless of the actual number of primary partitions on your hard disk.

`(hd1,a)`

This means the BSD '`a`' partition of the second hard disk. If you need to specify which PC slice number should be used, use something like this: '`(hd1,0,a)`'. If the PC slice number is omitted, GRUB searches for the first PC slice which has a BSD '`a`' partition.

Of course, to actually access the disks or partitions with GRUB, you need to use the device specification in a command, like '`root (fd0)`' or '`unhide (hd0,2)`'. To help you find out which number specifies a partition you want, the GRUB command-line (see Section 12.1 [Command-line interface], page 33) options have argument completion. This means that, for example, you only need to type

```
root (
```

followed by a ⟨TAB⟩, and GRUB will display the list of drives, partitions, or file names. So it should be quite easy to determine the name of your target partition, even with minimal knowledge of the syntax.

Note that GRUB does *not* distinguish IDE from SCSI - it simply counts the drive numbers from zero, regardless of their type. Normally, any IDE drive number is less than any SCSI drive number, although that is not true if you change the boot sequence by swapping IDE and SCSI drives in your BIOS.

Now the question is, how to specify a file? Again, consider an example:

`(hd0,0)/vmlinuz`

This specifies the file named '`vmlinuz`', found on the first partition of the first hard disk drive. Note that the argument completion works with file names, too.

That was easy, admit it. Now read the next chapter, to find out how to actually install GRUB on your drive.

3 Installation

In order to install GRUB as your boot loader, you need to first install the GRUB system and utilities under your UNIX-like operating system (see Appendix A [Obtaining and Building GRUB], page 69). You can do this either from the source tarball, or as a package for your OS.

After you have done that, you need to install the boot loader on a drive (floppy or hard disk). There are two ways of doing that - either using the utility `grub-install` (see Chapter 16 [Invoking grub-install], page 59) on a UNIX-like OS, or by running GRUB itself from a floppy. These are quite similar, however the utility might probe a wrong BIOS drive, so you should be careful.

Also, if you install GRUB on a UNIX-like OS, please make sure that you have an emergency boot disk ready, so that you can rescue your computer if, by any chance, your hard drive becomes unusable (unbootable).

GRUB comes with boot images, which are normally put in the directory '`/usr/lib/grub/i386-pc`'. If you do not use grub-install, then you need to copy the files '`stage1`', '`stage2`', and '`*stage1_5`' to the directory '`/boot/grub`', and run the `grub-set-default` (see Chapter 19 [Invoking grub-set-default], page 65) if you intend to use '`default saved`' (see Section 13.1.1 [default], page 35) in your configuration file. Hereafter, the directory where GRUB images are initially placed (normally '`/usr/lib/grub/i386-pc`') will be called the *image directory*, and the directory where the boot loader needs to find them (usually '`/boot/grub`') will be called the *boot directory*.

3.1 Creating a GRUB boot floppy

To create a GRUB boot floppy, you need to take the files '`stage1`' and '`stage2`' from the image directory, and write them to the first and the second block of the floppy disk, respectively.

Caution: This procedure will destroy any data currently stored on the floppy.

On a UNIX-like operating system, that is done with the following commands:

```
# cd /usr/lib/grub/i386-pc
# dd if=stage1 of=/dev/fd0 bs=512 count=1
1+0 records in
1+0 records out
# dd if=stage2 of=/dev/fd0 bs=512 seek=1
153+1 records in
153+1 records out
#
```

The device file name may be different. Consult the manual for your OS.

3.2 Installing GRUB natively

Caution: Installing GRUB's stage1 in this manner will erase the normal boot-sector used by an OS.

GRUB can currently boot GNU Mach, Linux, FreeBSD, NetBSD, and OpenBSD directly, so using it on a boot sector (the first sector of a partition) should be okay. But

generally, it would be a good idea to back up the first sector of the partition on which you are installing GRUB's stage1. This isn't as important if you are installing GRUB on the first sector of a hard disk, since it's easy to reinitialize it (e.g. by running '`FDISK /MBR`' from DOS).

If you decide to install GRUB in the native environment, which is definitely desirable, you'll need to create a GRUB boot disk, and reboot your computer with it. Otherwise, see Section 3.3 [Installing GRUB using grub-install], page 8.

Once started, GRUB will show the command-line interface (see Section 12.1 [Command-line interface], page 33). First, set the GRUB's *root device*[1] to the partition containing the boot directory, like this:

```
grub> root (hd0,0)
```

If you are not sure which partition actually holds this directory, use the command `find` (see Section 13.3.11 [find], page 43), like this:

```
grub> find /boot/grub/stage1
```

This will search for the file name '`/boot/grub/stage1`' and show the devices which contain the file.

Once you've set the root device correctly, run the command `setup` (see Section 13.3.34 [setup], page 48):

```
grub> setup (hd0)
```

This command will install the GRUB boot loader on the Master Boot Record (MBR) of the first drive. If you want to put GRUB into the boot sector of a partition instead of putting it in the MBR, specify the partition into which you want to install GRUB:

```
grub> setup (hd0,0)
```

If you install GRUB into a partition or a drive other than the first one, you must chain-load GRUB from another boot loader. Refer to the manual for the boot loader to know how to chain-load GRUB.

After using the setup command, you will boot into GRUB without the GRUB floppy. See the chapter Chapter 4 [Booting], page 11 to find out how to boot your operating systems from GRUB.

3.3 Installing GRUB using grub-install

Caution: This procedure is definitely less safe, because there are several ways in which your computer can become unbootable. For example, most operating systems don't tell GRUB how to map BIOS drives to OS devices correctly—GRUB merely *guesses* the mapping. This will succeed in most cases, but not always. Therefore, GRUB provides you with a map file called the *device map*, which you must fix if it is wrong. See Section 15.3 [Device map], page 56, for more details.

If you still do want to install GRUB under a UNIX-like OS (such as GNU), invoke the program `grub-install` (see Chapter 16 [Invoking grub-install], page 59) as the superuser (*root*).

[1] Note that GRUB's root device doesn't necessarily mean your OS's root partition; if you need to specify a root partition for your OS, add the argument into the command `kernel`.

The usage is basically very simple. You only need to specify one argument to the program, namely, where to install the boot loader. The argument can be either a device file (like '`/dev/hda`') or a partition specified in GRUB's notation. For example, under Linux the following will install GRUB into the MBR of the first IDE disk:

```
# grub-install /dev/hda
```

Likewise, under GNU/Hurd, this has the same effect:

```
# grub-install /dev/hd0
```

If it is the first BIOS drive, this is the same as well:

```
# grub-install '(hd0)'
```

Or you can omit the parentheses:

```
# grub-install hd0
```

But all the above examples assume that GRUB should use images under the root directory. If you want GRUB to use images under a directory other than the root directory, you need to specify the option '`--root-directory`'. The typical usage is that you create a GRUB boot floppy with a filesystem. Here is an example:

```
# mke2fs /dev/fd0
# mount -t ext2 /dev/fd0 /mnt
# grub-install --root-directory=/mnt fd0
# umount /mnt
```

Another example is when you have a separate boot partition which is mounted at '`/boot`'. Since GRUB is a boot loader, it doesn't know anything about mountpoints at all. Thus, you need to run `grub-install` like this:

```
# grub-install --root-directory=/boot /dev/hda
```

By the way, as noted above, it is quite difficult to guess BIOS drives correctly under a UNIX-like OS. Thus, `grub-install` will prompt you to check if it could really guess the correct mappings, after the installation. The format is defined in Section 15.3 [Device map], page 56. Please be quite careful. If the output is wrong, it is unlikely that your computer will be able to boot with no problem.

Note that `grub-install` is actually just a shell script and the real task is done by the grub shell `grub` (see Chapter 15 [Invoking the grub shell], page 55). Therefore, you may run `grub` directly to install GRUB, without using `grub-install`. Don't do that, however, unless you are very familiar with the internals of GRUB. Installing a boot loader on a running OS may be extremely dangerous.

3.4 Making a GRUB bootable CD-ROM

GRUB supports the *no emulation mode* in the El Torito specification[2]. This means that you can use the whole CD-ROM from GRUB and you don't have to make a floppy or hard disk image file, which can cause compatibility problems.

For booting from a CD-ROM, GRUB uses a special Stage 2 called '`stage2_eltorito`'. The only GRUB files you need to have in your bootable CD-ROM are this '`stage2_eltorito`' and optionally a config file '`menu.lst`'. You don't need to use '`stage1`' or '`stage2`', because El Torito is quite different from the standard boot process.

[2] El Torito is a specification for bootable CD using BIOS functions.

Here is an example of procedures to make a bootable CD-ROM image. First, make a top directory for the bootable image, say, ‘`iso`’:

```
$ mkdir iso
```

Make a directory for GRUB:

```
$ mkdir -p iso/boot/grub
```

Copy the file ‘`stage2_eltorito`’:

```
$ cp /usr/lib/grub/i386-pc/stage2_eltorito iso/boot/grub
```

If desired, make the config file ‘`menu.lst`’ under ‘`iso/boot/grub`’ (see Chapter 5 [Configuration], page 17), and copy any files and directories for the disc to the directory ‘`iso/`’.

Finally, make a ISO9660 image file like this:

```
$ mkisofs -R -b boot/grub/stage2_eltorito -no-emul-boot \
    -boot-load-size 4 -boot-info-table -o grub.iso iso
```

This produces a file named ‘`grub.iso`’, which then can be burned into a CD (or a DVD). *`mkisofs`* has already set up the disc to boot from the *`boot/grub/stage2_eltorito`* file, so there is no need to setup GRUB on the disc. (Note that the *`-boot-load-size 4`* bit is required for compatibility with the BIOS on many older machines.)

You can use the device ‘`(cd)`’ to access a CD-ROM in your config file. This is not required; GRUB automatically sets the root device to ‘`(cd)`’ when booted from a CD-ROM. It is only necessary to refer to ‘`(cd)`’ if you want to access other drives as well.

4 Booting

GRUB can load Multiboot-compliant kernels in a consistent way, but for some free operating systems you need to use some OS-specific magic.

4.1 How to boot operating systems

GRUB has two distinct boot methods. One of the two is to load an operating system directly, and the other is to chain-load another boot loader which then will load an operating system actually. Generally speaking, the former is more desirable, because you don't need to install or maintain other boot loaders and GRUB is flexible enough to load an operating system from an arbitrary disk/partition. However, the latter is sometimes required, since GRUB doesn't support all the existing operating systems natively.

4.1.1 How to boot an OS directly with GRUB

Multiboot (see section "Motivation" in *The Multiboot Specification*) is the native format supported by GRUB. For the sake of convenience, there is also support for Linux, FreeBSD, NetBSD and OpenBSD. If you want to boot other operating systems, you will have to chain-load them (see Section 4.1.2 [Chain-loading], page 11).

Generally, GRUB can boot any Multiboot-compliant OS in the following steps:

1. Set GRUB's root device to the drive where the OS images are stored with the command `root` (see Section 13.3.31 [root], page 47).
2. Load the kernel image with the command `kernel` (see Section 13.3.20 [kernel], page 46).
3. If you need modules, load them with the command `module` (see Section 13.3.25 [module], page 47) or `modulenounzip` (see Section 13.3.26 [modulenounzip], page 47).
4. Run the command `boot` (see Section 13.3.2 [boot], page 42).

Linux, FreeBSD, NetBSD and OpenBSD can be booted in a similar manner. You load a kernel image with the command `kernel` and then run the command `boot`. If the kernel requires some parameters, just append the parameters to `kernel`, after the file name of the kernel. Also, please refer to Section 4.2 [OS-specific notes], page 12, for information on your OS-specific issues.

4.1.2 Load another boot loader to boot unsupported operating systems

If you want to boot an unsupported operating system (e.g. Windows 95), chain-load a boot loader for the operating system. Normally, the boot loader is embedded in the *boot sector* of the partition on which the operating system is installed.

1. Set GRUB's root device to the partition by the command `rootnoverify` (see Section 13.3.32 [rootnoverify], page 48):

   ```
   grub> rootnoverify (hd0,0)
   ```

2. Set the *active* flag in the partition using the command `makeactive`[1] (see Section 13.3.22 [makeactive], page 46):

   ```
   grub> makeactive
   ```

[1] This is not necessary for most of the modern operating systems.

3. Load the boot loader with the command `chainloader` (see Section 13.3.4 [chainloader], page 42):

   ```
   grub> chainloader +1
   ```

   '`+1`' indicates that GRUB should read one sector from the start of the partition. The complete description about this syntax can be found in Section 11.3 [Block list syntax], page 32.
4. Run the command `boot` (see Section 13.3.2 [boot], page 42).

However, DOS and Windows have some deficiencies, so you might have to use more complicated instructions. See Section 4.2.6 [DOS/Windows], page 13, for more information.

4.2 Some caveats on OS-specific issues

Here, we describe some caveats on several operating systems.

4.2.1 GNU/Hurd

Since GNU/Hurd is Multiboot-compliant, it is easy to boot it; there is nothing special about it. But do not forget that you have to specify a root partition to the kernel.

1. Set GRUB's root device to the same drive as GNU/Hurd's. Probably the command `find /boot/gnumach` or similar can help you (see Section 13.3.11 [find], page 43).
2. Load the kernel and the module, like this:

   ```
   grub> kernel /boot/gnumach root=hd0s1
   grub> module /boot/serverboot
   ```
3. Run the command `boot` (see Section 13.3.2 [boot], page 42).

4.2.2 GNU/Linux

It is relatively easy to boot GNU/Linux from GRUB, because it somewhat resembles to boot a Multiboot-compliant OS.

1. Set GRUB's root device to the same drive as GNU/Linux's. Probably the command `find /vmlinuz` or similar can help you (see Section 13.3.11 [find], page 43).
2. Load the kernel:

   ```
   grub> kernel /vmlinuz root=/dev/hda1
   ```

   If you need to specify some kernel parameters, just append them to the command. For example, to set '`vga`' to '`ext`', do this:

   ```
   grub> kernel /vmlinuz root=/dev/hda1 vga=ext
   ```

   See the documentation in the Linux source tree for complete information on the available options.
3. If you use an initrd, execute the command `initrd` (see Section 13.3.17 [initrd], page 44) after `kernel`:

   ```
   grub> initrd /initrd
   ```
4. Finally, run the command `boot` (see Section 13.3.2 [boot], page 42).

Caution: If you use an initrd and specify the '`mem=`' option to the kernel to let it use less than actual memory size, you will also have to specify the same memory size to GRUB. To let GRUB know the size, run the command `uppermem` *before* loading the kernel. See Section 13.3.37 [uppermem], page 49, for more information.

4.2.3 FreeBSD

GRUB can load the kernel directly, either in ELF or a.out format. But this is not recommended, since FreeBSD's bootstrap interface sometimes changes heavily, so GRUB can't guarantee to pass kernel parameters correctly.

Thus, we'd recommend loading the very flexible loader '`/boot/loader`' instead. See this example:

```
grub> root (hd0,a)
grub> kernel /boot/loader
grub> boot
```

4.2.4 NetBSD

GRUB can load NetBSD a.out and ELF directly, follow these steps:

1. Set GRUB's root device with `root` (see Section 13.3.31 [root], page 47).
2. Load the kernel with `kernel` (see Section 13.3.20 [kernel], page 46). You should append the ugly option '`--type=netbsd`', if you want to load an ELF kernel, like this:

   ```
   grub> kernel --type=netbsd /netbsd-elf
   ```
3. Run `boot` (see Section 13.3.2 [boot], page 42).

For now, however, GRUB doesn't allow you to pass kernel parameters, so it may be better to chain-load it instead. For more information, please see Section 4.1.2 [Chain-loading], page 11.

4.2.5 OpenBSD

The booting instruction is exactly the same as for NetBSD (see Section 4.2.4 [NetBSD], page 13).

4.2.6 DOS/Windows

GRUB cannot boot DOS or Windows directly, so you must chain-load them (see Section 4.1.2 [Chain-loading], page 11). However, their boot loaders have some critical deficiencies, so it may not work to just chain-load them. To overcome the problems, GRUB provides you with two helper functions.

If you have installed DOS (or Windows) on a non-first hard disk, you have to use the disk swapping technique, because that OS cannot boot from any disks but the first one. The workaround used in GRUB is the command `map` (see Section 13.3.23 [map], page 46), like this:

```
grub> map (hd0) (hd1)
grub> map (hd1) (hd0)
```

This performs a *virtual* swap between your first and second hard drive.

Caution: This is effective only if DOS (or Windows) uses BIOS to access the swapped disks. If that OS uses a special driver for the disks, this probably won't work.

Another problem arises if you installed more than one set of DOS/Windows onto one disk, because they could be confused if there are more than one primary partitions for DOS/Windows. Certainly you should avoid doing this, but there is a solution if you do want to do so. Use the partition hiding/unhiding technique.

If GRUB *hide*s a DOS (or Windows) partition (see Section 13.2.5 [hide], page 37), DOS (or Windows) will ignore the partition. If GRUB *unhide*s a DOS (or Windows) partition (see Section 13.2.17 [unhide], page 42), DOS (or Windows) will detect the partition. Thus, if you have installed DOS (or Windows) on the first and the second partition of the first hard disk, and you want to boot the copy on the first partition, do the following:

```
grub> unhide (hd0,0)
grub> hide (hd0,1)
grub> rootnoverify (hd0,0)
grub> chainloader +1
grub> makeactive
grub> boot
```

4.2.7 SCO UnixWare

It is known that the signature in the boot loader for SCO UnixWare is wrong, so you will have to specify the option '`--force`' to `chainloader` (see Section 13.3.4 [chainloader], page 42), like this:

```
grub> rootnoverify (hd1,0)
grub> chainloader --force +1
grub> makeactive
grub> boot
```

4.2.8 QNX

QNX seems to use a bigger boot loader, so you need to boot it up, like this:

```
grub> rootnoverify (hd1,1)
grub> chainloader +4
grub> boot
```

4.3 How to make your system robust

When you test a new kernel or a new OS, it is important to make sure that your computer can boot even if the new system is unbootable. This is crucial especially if you maintain servers or remote systems. To accomplish this goal, you need to set up two things:

1. You must maintain a system which is always bootable. For instance, if you test a new kernel, you need to keep a working kernel in a different place. And, it would sometimes be very nice to even have a complete copy of a working system in a different partition or disk.
2. You must direct GRUB to boot a working system when the new system fails. This is possible with the *fallback* system in GRUB.

The former requirement is very specific to each OS, so this documentation does not cover that topic. It is better to consult some backup tools.

So let's see the GRUB part. There are two possibilities: one of them is quite simple but not very robust, and the other is a bit complex to set up but probably the best solution to make sure that your system can start as long as GRUB itself is bootable.

4.3.1 Booting once-only

You can teach GRUB to boot an entry only at next boot time. Suppose that your have an old kernel '`old_kernel`' and a new kernel '`new_kernel`'. You know that '`old_kernel`' can boot your system correctly, and you want to test '`new_kernel`'.

To ensure that your system will go back to the old kernel even if the new kernel fails (e.g. it panics), you can specify that GRUB should try the new kernel only once and boot the old kernel after that.

First, modify your configuration file. Here is an example:

```
default saved        # This is important!!!
timeout 10

title the old kernel
root (hd0,0)
kernel /old_kernel
savedefault

title the new kernel
root (hd0,0)
kernel /new_kernel
savedefault 0         # This is important!!!
```

Note that this configuration file uses '`default saved`' (see Section 13.1.1 [default], page 35) at the head and '`savedefault 0`' (see Section 13.3.33 [savedefault], page 48) in the entry for the new kernel. This means that GRUB boots a saved entry by default, and booting the entry for the new kernel saves '`0`' as the saved entry.

With this configuration file, after all, GRUB always tries to boot the old kernel after it booted the new one, because '`0`' is the entry of `the old kernel`.

The next step is to tell GRUB to boot the new kernel at next boot time. For this, execute `grub-set-default` (see Chapter 19 [Invoking grub-set-default], page 65):

```
# grub-set-default 1
```

This command sets the saved entry to '`1`', that is, to the new kernel.

This method is useful, but still not very robust, because GRUB stops booting, if there is any error in the boot entry, such that the new kernel has an invalid executable format. Thus, it it even better to use the *fallback* mechanism of GRUB. Look at next subsection for this feature.

4.3.2 Booting fallback systems

GRUB supports a fallback mechanism of booting one or more other entries if a default boot entry fails. You can specify multiple fallback entries if you wish.

Suppose that you have three systems, '`A`', '`B`' and '`C`'. '`A`' is a system which you want to boot by default. '`B`' is a backup system which is supposed to boot safely. '`C`' is another backup system which is used in case where '`B`' is broken.

Then you may want GRUB to boot the first system which is bootable among '`A`', '`B`' and '`C`'. A configuration file can be written in this way:

```
default saved        # This is important!!!
timeout 10
fallback 1 2         # This is important!!!

title A
root (hd0,0)
kernel /kernel
savedefault fallback # This is important!!!

title B
root (hd1,0)
kernel /kernel
savedefault fallback # This is important!!!

title C
root (hd2,0)
kernel /kernel
savedefault
```

Note that '`default saved`' (see Section 13.1.1 [default], page 35), '`fallback 1 2`' and '`savedefault fallback`' are used. GRUB will boot a saved entry by default and save a fallback entry as next boot entry with this configuration.

When GRUB tries to boot '`A`', GRUB saves '`1`' as next boot entry, because the command `fallback` specifies that '`1`' is the first fallback entry. The entry '`1`' is '`B`', so GRUB will try to boot '`B`' at next boot time.

Likewise, when GRUB tries to boot '`B`', GRUB saves '`2`' as next boot entry, because `fallback` specifies '`2`' as next fallback entry. This makes sure that GRUB will boot '`C`' after booting '`B`'.

It is noteworthy that GRUB uses fallback entries both when GRUB itself fails in booting an entry and when '`A`' or '`B`' fails in starting up your system. So this solution ensures that your system is started even if GRUB cannot find your kernel or if your kernel panics.

However, you need to run `grub-set-default` (see Chapter 19 [Invoking grub-set-default], page 65) when '`A`' starts correctly or you fix '`A`' after it crashes, since GRUB always sets next boot entry to a fallback entry. You should run this command in a startup script such as '`rc.local`' to boot '`A`' by default:

```
# grub-set-default 0
```

where '`0`' is the number of the boot entry for the system '`A`'.

If you want to see what is current default entry, you can look at the file '`/boot/grub/default`' (or '`/grub/default`' in some systems). Because this file is plain-text, you can just `cat` this file. But it is strongly recommended **not to modify this file directly**, because GRUB may fail in saving a default entry in this file, if you change this file in an unintended manner. Therefore, you should use `grub-set-default` when you need to change the default entry.

5 Configuration

You've probably noticed that you need to type several commands to boot your OS. There's a solution to that - GRUB provides a menu interface (see Section 12.2 [Menu interface], page 34) from which you can select an item (using arrow keys) that will do everything to boot an OS.

To enable the menu, you need a configuration file, '`menu.lst`' under the boot directory. We'll analyze an example file.

The file first contains some general settings, the menu interface related options. You can put these commands (see Section 13.1 [Menu-specific commands], page 35) before any of the items (starting with `title` (see Section 13.1.5 [title], page 36)).

```
#
# Sample boot menu configuration file
#
```

As you may have guessed, these lines are comments. Lines starting with a hash character ('`#`'), and blank lines, are ignored by GRUB.

```
# By default, boot the first entry.
default 0
```

The first entry (here, counting starts with number zero, not one!) will be the default choice.

```
# Boot automatically after 30 secs.
timeout 30
```

As the comment says, GRUB will boot automatically in 30 seconds, unless interrupted with a keypress.

```
# Fallback to the second entry.
fallback 1
```

If, for any reason, the default entry doesn't work, fall back to the second one (this is rarely used, for obvious reasons).

Note that the complete descriptions of these commands, which are menu interface specific, can be found in Section 13.1 [Menu-specific commands], page 35. Other descriptions can be found in Chapter 13 [Commands], page 35.

Now, on to the actual OS definitions. You will see that each entry begins with a special command, `title` (see Section 13.1.5 [title], page 36), and the action is described after it. Note that there is no command `boot` (see Section 13.3.2 [boot], page 42) at the end of each item. That is because GRUB automatically executes `boot` if it loads other commands successfully.

The argument for the command `title` is used to display a short title/description of the entry in the menu. Since `title` displays the argument as is, you can write basically anything there.

```
# For booting GNU/Hurd
title  GNU/Hurd
root   (hd0,0)
kernel /boot/gnumach.gz root=hd0s1
module /boot/serverboot.gz
```

This boots GNU/Hurd from the first hard disk.

```
# For booting GNU/Linux
title  GNU/Linux
kernel (hd1,0)/vmlinuz root=/dev/hdb1
```

This boots GNU/Linux, but from the second hard disk.

```
# For booting Mach (getting kernel from floppy)
title  Utah Mach4 multiboot
root   (hd0,2)
pause  Insert the diskette now^G!!
kernel (fd0)/boot/kernel root=hd0s3
module (fd0)/boot/bootstrap
```

This boots Mach with a kernel on a floppy, but the root filesystem at hd0s3. It also contains a `pause` line (see Section 13.3.27 [pause], page 47), which will cause GRUB to display a prompt and delay, before actually executing the rest of the commands and booting.

```
# For booting FreeBSD
title  FreeBSD
root   (hd0,2,a)
kernel /boot/loader
```

This item will boot FreeBSD kernel loaded from the 'a' partition of the third PC slice of the first hard disk.

```
# For booting OS/2
title OS/2
root  (hd0,1)
makeactive
# chainload OS/2 bootloader from the first sector
chainloader +1
# This is similar to "chainload", but loads a specific file
#chainloader /boot/chain.os2
```

This will boot OS/2, using a chain-loader (see Section 4.1.2 [Chain-loading], page 11).

```
# For booting Windows NT or Windows95
title Windows NT / Windows 95 boot menu
root        (hd0,0)
makeactive
chainloader +1
# For loading DOS if Windows NT is installed
# chainload /bootsect.dos
```

The same as the above, but for Windows.

```
# For installing GRUB into the hard disk
title Install GRUB into the hard disk
root    (hd0,0)
setup   (hd0)
```

This will just (re)install GRUB onto the hard disk.

```
# Change the colors.
title Change the colors
color light-green/brown blink-red/blue
```

In the last entry, the command `color` is used (see Section 13.2.2 [color], page 36), to change the menu colors (try it!). This command is somewhat special, because it can be used both in the command-line and in the menu. GRUB has several such commands, see Section 13.2 [General commands], page 36.

We hope that you now understand how to use the basic features of GRUB. To learn more about GRUB, see the following chapters.

6 Downloading OS images from a network

Although GRUB is a disk-based boot loader, it does provide network support. To use the network support, you need to enable at least one network driver in the GRUB build process. For more information please see '`netboot/README.netboot`' in the source distribution.

6.1 How to set up your network

GRUB requires a file server and optionally a server that will assign an IP address to the machine on which GRUB is running. For the former, only TFTP is supported at the moment. The latter is either BOOTP, DHCP or a RARP server[1]. It is not necessary to run both the servers on one computer. How to configure these servers is beyond the scope of this document, so please refer to the manuals specific to those protocols/servers.

If you decided to use a server to assign an IP address, set up the server and run `bootp` (see Section 13.2.1 [bootp], page 36), `dhcp` (see Section 13.2.4 [dhcp], page 37) or `rarp` (see Section 13.2.11 [rarp], page 38) for BOOTP, DHCP or RARP, respectively. Each command will show an assigned IP address, a netmask, an IP address for your TFTP server and a gateway. If any of the addresses is wrong or it causes an error, probably the configuration of your servers isn't set up properly.

Otherwise, run `ifconfig`, like this:

```
grub> ifconfig --address=192.168.110.23 --server=192.168.110.14
```

You can also use `ifconfig` in conjuction with `bootp`, `dhcp` or `rarp` (e.g. to reassign the server address manually). See Section 13.2.6 [ifconfig], page 38, for more details.

Finally, download your OS images from your network. The network can be accessed using the network drive '`(nd)`'. Everything else is very similar to the normal instructions (see Chapter 4 [Booting], page 11).

Here is an example:

```
grub> bootp
Probing... [NE*000]
NE2000 base ...
Address: 192.168.110.23    Netmask: 255.255.255.0
Server: 192.168.110.14     Gateway: 192.168.110.1

grub> root (nd)
grub> kernel /tftproot/gnumach.gz root=sd0s1
grub> module /tftproot/serverboot.gz
grub> boot
```

6.2 Booting from a network

It is sometimes very useful to boot from a network, especially when you use a machine which has no local disk. In this case, you need to obtain a kind of Net Boot ROM, such as a PXE ROM or a free software package like Etherboot. Such a Boot ROM first boots the machine, sets up the network card installed into the machine, and downloads a second stage

[1] RARP is not advised, since it cannot serve much information

boot image from the network. Then, the second image will try to boot an operating system actually from the network.

GRUB provides two second stage images, '`nbgrub`' and '`pxegrub`' (see Chapter 10 [Images], page 29). These images are the same as the normal Stage 2, except that they set up a network automatically, and try to load a configuration file from the network, if specified. The usage is very simple: If the machine has a PXE ROM, use '`pxegrub`'. If the machine has an NBI loader such as Etherboot, use '`nbgrub`'. There is no difference between them except their formats. Since the way to load a second stage image you want to use should be described in the manual on your Net Boot ROM, please refer to the manual, for more information.

However, there is one thing specific to GRUB. Namely, how to specify a configuration file in a BOOTP/DHCP server. For now, GRUB uses the tag '`150`', to get the name of a configuration file. The following is an example with a BOOTP configuration:

```
.allhost:hd=/tmp:bf=null:\
        :ds=145.71.35.1 145.71.32.1:\
        :sm=255.255.254.0:\
        :gw=145.71.35.1:\
        :sa=145.71.35.5:

foo:ht=1:ha=63655d0334a7:ip=145.71.35.127:\
        :bf=/nbgrub:\
        :tc=.allhost:\
        :T150="(nd)/tftpboot/menu.lst.foo":
```

Note that you should specify the drive name `(nd)` in the name of the configuration file. This is because you might change the root drive before downloading the configuration from the TFTP server when the preset menu feature is used (see Chapter 8 [Preset Menu], page 25).

See the manual of your BOOTP/DHCP server for more information. The exact syntax should differ a little from the example.

7 Using GRUB via a serial line

This chapter describes how to use the serial terminal support in GRUB.

If you have many computers or computers with no display/keyboard, it could be very useful to control the computers through serial communications. To connect one computer with another via a serial line, you need to prepare a null-modem (cross) serial cable, and you may need to have multiport serial boards, if your computer doesn't have extra serial ports. In addition, a terminal emulator is also required, such as minicom. Refer to a manual of your operating system, for more information.

As for GRUB, the instruction to set up a serial terminal is quite simple. First of all, make sure that you haven't specified the option '`--disable-serial`' to the configure script when you built your GRUB images. If you get them in binary form, probably they have serial terminal support already.

Then, initialize your serial terminal after GRUB starts up. Here is an example:

```
grub> serial --unit=0 --speed=9600
grub> terminal serial
```

The command `serial` initializes the serial unit 0 with the speed 9600bps. The serial unit 0 is usually called '`COM1`', so, if you want to use COM2, you must specify '`--unit=1`' instead. This command accepts many other options, so please refer to Section 13.2.12 [serial], page 39, for more details.

The command `terminal` (see Section 13.2.14 [terminal], page 41) chooses which type of terminal you want to use. In the case above, the terminal will be a serial terminal, but you can also pass `console` to the command, as '`terminal serial console`'. In this case, a terminal in which you press any key will be selected as a GRUB terminal.

However, note that GRUB assumes that your terminal emulator is compatible with VT100 by default. This is true for most terminal emulators nowadays, but you should pass the option '`--dumb`' to the command if your terminal emulator is not VT100-compatible or implements few VT100 escape sequences. If you specify this option then GRUB provides you with an alternative menu interface, because the normal menu requires several fancy features of your terminal.

8 Embedding a configuration file into GRUB

GRUB supports a *preset menu* which is to be always loaded before starting. The preset menu feature is useful, for example, when your computer has no console but a serial cable. In this case, it is critical to set up the serial terminal as soon as possible, since you cannot see any message until the serial terminal begins to work. So it is good to run the commands `serial` (see Section 13.2.12 [serial], page 39) and `terminal` (see Section 13.2.14 [terminal], page 41) before anything else at the start-up time.

How the preset menu works is slightly complicated:

1. GRUB checks if the preset menu feature is used, and loads the preset menu, if available. This includes running commands and reading boot entries, like an ordinary configuration file.
2. GRUB checks if the configuration file is available. Note that this check is performed **regardless of the existence of the preset menu**. The configuration file is loaded even if the preset menu was loaded.
3. If the preset menu includes any boot entries, they are cleared when the configuration file is loaded. It doesn't matter whether the configuration file has any entries or no entry. The boot entries in the preset menu are used only when GRUB fails in loading the configuration file.

To enable the preset menu feature, you must rebuild GRUB specifying a file to the configure script with the option '`--enable-preset-menu`'. The file has the same semantics as normal configuration files (see Chapter 5 [Configuration], page 17).

Another point you should take care is that the diskless support (see Section 6.2 [Diskless], page 21) diverts the preset menu. Diskless images embed a preset menu to execute the command `bootp` (see Section 13.2.1 [bootp], page 36) automatically, unless you specify your own preset menu to the configure script. This means that you must put commands to initialize a network in the preset menu yourself, because diskless images don't set it up implicitly, when you use the preset menu explicitly.

Therefore, a typical preset menu used with diskless support would be like this:

```
# Set up the serial terminal, first of all.
serial --unit=0 --speed=19200
terminal --timeout=0 serial

# Initialize the network.
dhcp
```

9 Protecting your computer from cracking

You may be interested in how to prevent ordinary users from doing whatever they like, if you share your computer with other people. So this chapter describes how to improve the security of GRUB.

One thing which could be a security hole is that the user can do too many things with GRUB, because GRUB allows one to modify its configuration and run arbitrary commands at run-time. For example, the user can even read '`/etc/passwd`' in the command-line interface by the command `cat` (see Section 13.3.3 [cat], page 42). So it is necessary to disable all the interactive operations.

Thus, GRUB provides a *password* feature, so that only administrators can start the interactive operations (i.e. editing menu entries and entering the command-line interface). To use this feature, you need to run the command `password` in your configuration file (see Section 13.2.10 [password], page 38), like this:

```
password --md5 PASSWORD
```

If this is specified, GRUB disallows any interactive control, until you press the key ⓟ and enter a correct password. The option '`--md5`' tells GRUB that '`PASSWORD`' is in MD5 format. If it is omitted, GRUB assumes the '`PASSWORD`' is in clear text.

You can encrypt your password with the command `md5crypt` (see Section 13.3.24 [md5crypt], page 47). For example, run the grub shell (see Chapter 15 [Invoking the grub shell], page 55), and enter your password:

```
grub> md5crypt
Password: **********
Encrypted: $1$U$JK7xFegdxWH6VuppCUSIb.
```

Then, cut and paste the encrypted password to your configuration file.

Also, you can specify an optional argument to `password`. See this example:

```
password PASSWORD /boot/grub/menu-admin.lst
```

In this case, GRUB will load '`/boot/grub/menu-admin.lst`' as a configuration file when you enter the valid password.

Another thing which may be dangerous is that any user can choose any menu entry. Usually, this wouldn't be problematic, but you might want to permit only administrators to run some of your menu entries, such as an entry for booting an insecure OS like DOS.

GRUB provides the command `lock` (see Section 13.3.21 [lock], page 46). This command always fails until you enter the valid password, so you can use it, like this:

```
title Boot DOS
lock
rootnoverify (hd0,1)
makeactive
chainload +1
```

You should insert `lock` right after `title`, because any user can execute commands in an entry until GRUB encounters `lock`.

You can also use the command `password` instead of `lock`. In this case the boot process will ask for the password and stop if it was entered incorrectly. Since the `password`

takes its own *PASSWORD* argument this is useful if you want different passwords for different entries.

10 GRUB image files

GRUB consists of several images: two essential stages, optional stages called *Stage 1.5*, one image for bootable CD-ROM, and two network boot images. Here is a short overview of them. See Appendix D [Internals], page 75, for more details.

'`stage1`' This is an essential image used for booting up GRUB. Usually, this is embedded in an MBR or the boot sector of a partition. Because a PC boot sector is 512 bytes, the size of this image is exactly 512 bytes.

All '`stage1`' must do is to load Stage 2 or Stage 1.5 from a local disk. Because of the size restriction, '`stage1`' encodes the location of Stage 2 (or Stage 1.5) in a block list format, so it never understand any filesystem structure.

'`stage2`' This is the core image of GRUB. It does everything but booting up itself. Usually, this is put in a filesystem, but that is not required.

'`e2fs_stage1_5`'
'`fat_stage1_5`'
'`ffs_stage1_5`'
'`jfs_stage1_5`'
'`minix_stage1_5`'
'`reiserfs_stage1_5`'
'`vstafs_stage1_5`'
'`xfs_stage1_5`'

These are called *Stage 1.5*, because they serve as a bridge between '`stage1`' and '`stage2`', that is to say, Stage 1.5 is loaded by Stage 1 and Stage 1.5 loads Stage 2. The difference between '`stage1`' and '`*_stage1_5`' is that the former doesn't understand any filesystem while the latter understands one filesystem (e.g. '`e2fs_stage1_5`' understands ext2fs). So you can move the Stage 2 image to another location safely, even after GRUB has been installed.

While Stage 2 cannot generally be embedded in a fixed area as the size is so large, Stage 1.5 can be installed into the area right after an MBR, or the boot loader area of a ReiserFS or a FFS.

'`stage2_eltorito`'

This is a boot image for CD-ROMs using the *no emulation mode* in El Torito specification. This is identical to Stage 2, except that this boots up without Stage 1 and sets up a special drive '`(cd)`'.

'`nbgrub`' This is a network boot image for the Network Image Proposal used by some network boot loaders, such as Etherboot. This is mostly the same as Stage 2, but it also sets up a network and loads a configuration file from the network.

'`pxegrub`' This is another network boot image for the Preboot Execution Environment used by several Netboot ROMs. This is identical to '`nbgrub`', except for the format.

11 Filesystem syntax and semantics

GRUB uses a special syntax for specifying disk drives which can be accessed by BIOS. Because of BIOS limitations, GRUB cannot distinguish between IDE, ESDI, SCSI, or others. You must know yourself which BIOS device is equivalent to which OS device. Normally, that will be clear if you see the files in a device or use the command `find` (see Section 13.3.11 [find], page 43).

11.1 How to specify devices

The device syntax is like this:

```
(device[,part-num][,bsd-subpart-letter])
```

'`[]`' means the parameter is optional. *device* should be either '`fd`' or '`hd`' followed by a digit, like '`fd0`'. But you can also set *device* to a hexadecimal or a decimal number which is a BIOS drive number, so the following are equivalent:

```
(hd0)
(0x80)
(128)
```

part-num represents the partition number of *device*, starting from zero for primary partitions and from four for extended partitions, and *bsd-subpart-letter* represents the BSD disklabel subpartition, such as '`a`' or '`e`'.

A shortcut for specifying BSD subpartitions is `(device,bsd-subpart-letter)`, in this case, GRUB searches for the first PC partition containing a BSD disklabel, then finds the subpartition *bsd-subpart-letter*. Here is an example:

```
(hd0,a)
```

The syntax '`(hd0)`' represents using the entire disk (or the MBR when installing GRUB), while the syntax '`(hd0,0)`' represents using the first partition of the disk (or the boot sector of the partition when installing GRUB).

If you enabled the network support, the special drive, '`(nd)`', is also available. Before using the network drive, you must initialize the network. See Chapter 6 [Network], page 21, for more information.

If you boot GRUB from a CD-ROM, '`(cd)`' is available. See Section 3.4 [Making a GRUB bootable CD-ROM], page 9, for details.

11.2 How to specify files

There are two ways to specify files, by *absolute file name* and by *block list*.

An absolute file name resembles a Unix absolute file name, using '`/`' for the directory separator (not '`\`' as in DOS). One example is '`(hd0,0)/boot/grub/menu.lst`'. This means the file '`/boot/grub/menu.lst`' in the first partition of the first hard disk. If you omit the device name in an absolute file name, GRUB uses GRUB's *root device* implicitly. So if you set the root device to, say, '`(hd1,0)`' by the command `root` (see Section 13.3.31 [root], page 47), then `/boot/kernel` is the same as `(hd1,0)/boot/kernel`.

11.3 How to specify block lists

A block list is used for specifying a file that doesn't appear in the filesystem, like a chainloader. The syntax is `[`*`offset`*`]+`*`length`*`[,[`*`offset`*`]+`*`length`*`]...`. Here is an example:

```
0+100,200+1,300+300
```

This represents that GRUB should read blocks 0 through 99, block 200, and blocks 300 through 599. If you omit an offset, then GRUB assumes the offset is zero.

Like the file name syntax (see Section 11.2 [File name syntax], page 31), if a blocklist does not contain a device name, then GRUB uses GRUB's *root device*. So `(hd0,1)+1` is the same as `+1` when the root device is '`(hd0,1)`'.

12 GRUB's user interface

GRUB has both a simple menu interface for choosing preset entries from a configuration file, and a highly flexible command-line for performing any desired combination of boot commands.

GRUB looks for its configuration file as soon as it is loaded. If one is found, then the full menu interface is activated using whatever entries were found in the file. If you choose the *command-line* menu option, or if the configuration file was not found, then GRUB drops to the command-line interface.

12.1 The flexible command-line interface

The command-line interface provides a prompt and after it an editable text area much like a command-line in Unix or DOS. Each command is immediately executed after it is entered[1]. The commands (see Section 13.3 [Command-line and menu entry commands], page 42) are a subset of those available in the configuration file, used with exactly the same syntax.

Cursor movement and editing of the text on the line can be done via a subset of the functions available in the Bash shell:

⟨C-f⟩
⟨PC right key⟩
: Move forward one character.

⟨C-b⟩
⟨PC left key⟩ Move back one character.

⟨C-a⟩
⟨HOME⟩ Move to the start of the line.

⟨C-e⟩
⟨END⟩ Move the the end of the line.

⟨C-d⟩
⟨DEL⟩ Delete the character underneath the cursor.

⟨C-h⟩
⟨BS⟩ Delete the character to the left of the cursor.

⟨C-k⟩ Kill the text from the current cursor position to the end of the line.

⟨C-u⟩ Kill backward from the cursor to the beginning of the line.

⟨C-y⟩ Yank the killed text back into the buffer at the cursor.

⟨C-p⟩
⟨PC up key⟩ Move up through the history list.

⟨C-n⟩
⟨PC down key⟩
: Move down through the history list.

[1] However, this behavior will be changed in the future version, in a user-invisible way.

When typing commands interactively, if the cursor is within or before the first word in the command-line, pressing the ⟨TAB⟩ key (or ⟨C-i⟩) will display a listing of the available commands, and if the cursor is after the first word, the ⟨TAB⟩ will provide a completion listing of disks, partitions, and file names depending on the context. Note that to obtain a list of drives, one must open a parenthesis, as `root (`.

Note that you cannot use the completion functionality in the TFTP filesystem. This is because TFTP doesn't support file name listing for the security.

12.2 The simple menu interface

The menu interface is quite easy to use. Its commands are both reasonably intuitive and described on screen.

Basically, the menu interface provides a list of *boot entries* to the user to choose from. Use the arrow keys to select the entry of choice, then press ⟨RET⟩ to run it. An optional timeout is available to boot the default entry (the first one if not set), which is aborted by pressing any key.

Commands are available to enter a bare command-line by pressing ⟨c⟩ (which operates exactly like the non-config-file version of GRUB, but allows one to return to the menu if desired by pressing ⟨ESC⟩) or to edit any of the *boot entries* by pressing ⟨e⟩.

If you protect the menu interface with a password (see Chapter 9 [Security], page 27), all you can do is choose an entry by pressing ⟨RET⟩, or press ⟨p⟩ to enter the password.

12.3 Editing a menu entry

The menu entry editor looks much like the main menu interface, but the lines in the menu are individual commands in the selected entry instead of entry names.

If an ⟨ESC⟩ is pressed in the editor, it aborts all the changes made to the configuration entry and returns to the main menu interface.

When a particular line is selected, the editor places the user in a special version of the GRUB command-line to edit that line. When the user hits ⟨RET⟩, GRUB replaces the line in question in the boot entry with the changes (unless it was aborted via ⟨ESC⟩, in which case the changes are thrown away).

If you want to add a new line to the menu entry, press ⟨o⟩ if adding a line after the current line or press ⟨O⟩ if before the current line.

To delete a line, hit the key ⟨d⟩. Although GRUB unfortunately does not support *undo*, you can do almost the same thing by just returning to the main menu.

12.4 The hidden menu interface

When your terminal is dumb or you request GRUB to hide the menu interface explicitly with the command `hiddenmenu` (see Section 13.1.3 [hiddenmenu], page 35), GRUB doesn't show the menu interface (see Section 12.2 [Menu interface], page 34) and automatically boots the default entry, unless interrupted by pressing ⟨ESC⟩.

When you interrupt the timeout and your terminal is dumb, GRUB falls back to the command-line interface (see Section 12.1 [Command-line interface], page 33).

13 The list of available commands

In this chapter, we list all commands that are available in GRUB.

Commands belong to different groups. A few can only be used in the global section of the configuration file (or "menu"); most of them can be entered on the command-line and can be used either anywhere in the menu or specifically in the menu entries.

13.1 The list of commands for the menu only

The semantics used in parsing the configuration file are the following:

- The menu-specific commands have to be used before any others.
- The files *must* be in plain-text format.
- '`#`' at the beginning of a line in a configuration file means it is only a comment.
- Options are separated by spaces.
- All numbers can be either decimal or hexadecimal. A hexadecimal number must be preceded by '`0x`', and is case-insensitive.
- Extra options or text at the end of the line are ignored unless otherwise specified.
- Unrecognized commands are added to the current entry, except before entries start, where they are ignored.

These commands can only be used in the menu:

13.1.1 default

`default` *num* [Command]

Set the default entry to the entry number *num*. Numbering starts from 0, and the entry number 0 is the default if the command is not used.

You can specify '`saved`' instead of a number. In this case, the default entry is the entry saved with the command `savedefault`. See Section 13.3.33 [savedefault], page 48, for more information.

13.1.2 fallback

`fallback` *num...* [Command]

Go into unattended boot mode: if the default boot entry has any errors, instead of waiting for the user to do something, immediately start over using the *num* entry (same numbering as the `default` command (see Section 13.1.1 [default], page 35)). This obviously won't help if the machine was rebooted by a kernel that GRUB loaded. You can specify multiple fallback entry numbers.

13.1.3 hiddenmenu

`hiddenmenu` [Command]

Don't display the menu. If the command is used, no menu will be displayed on the control terminal, and the default entry will be booted after the timeout expired. The user can still request the menu to be displayed by pressing ⟨ESC⟩ before the timeout expires. See also Section 12.4 [Hidden menu interface], page 34.

13.1.4 timeout

`timeout` *sec* [Command]

Set a timeout, in *sec* seconds, before automatically booting the default entry (normally the first entry defined).

13.1.5 title

`title` *name ...* [Command]

Start a new boot entry, and set its name to the contents of the rest of the line, starting with the first non-space character.

13.2 The list of general commands

Commands usable anywhere in the menu and in the command-line.

13.2.1 bootp

`bootp` [‘`--with-configfile`’] [Command]

Initialize a network device via the *BOOTP* protocol. This command is only available if GRUB is compiled with netboot support. See also Chapter 6 [Network], page 21.

If you specify ‘`--with-configfile`’ to this command, GRUB will fetch and load a configuration file specified by your BOOTP server with the vendor tag ‘`150`’.

13.2.2 color

`color` *normal* [*highlight*] [Command]

Change the menu colors. The color *normal* is used for most lines in the menu (see Section 12.2 [Menu interface], page 34), and the color *highlight* is used to highlight the line where the cursor points. If you omit *highlight*, then the inverted color of *normal* is used for the highlighted line. The format of a color is `foreground/background`. *foreground* and *background* are symbolic color names. A symbolic color name must be one of these:

- black
- blue
- green
- cyan
- red
- magenta
- brown
- light-gray

  These below can be specified only for the foreground.

- dark-gray
- light-blue
- light-green
- light-cyan

- light-red
- light-magenta
- yellow
- white

But only the first eight names can be used for *background*. You can prefix `blink-` to *foreground* if you want a blinking foreground color.

This command can be used in the configuration file and on the command line, so you may write something like this in your configuration file:

```
# Set default colors.
color light-gray/blue black/light-gray

# Change the colors.
title OS-BS like
color magenta/blue black/magenta
```

13.2.3 device

`device` *drive file* [Command]

In the grub shell, specify the file *file* as the actual drive for a BIOS drive *drive*. You can use this command to create a disk image, and/or to fix the drives guessed by GRUB when GRUB fails to determine them correctly, like this:

```
grub> device (fd0) /floppy-image
grub> device (hd0) /dev/sd0
```

This command can be used only in the grub shell (see Chapter 15 [Invoking the grub shell], page 55).

13.2.4 dhcp

`dhcp` [*–with-configfile*] [Command]

Initialize a network device via the *DHCP* protocol. Currently, this command is just an alias for `bootp`, since the two protocols are very similar. This command is only available if GRUB is compiled with netboot support. See also Chapter 6 [Network], page 21.

If you specify '`--with-configfile`' to this command, GRUB will fetch and load a configuration file specified by your DHCP server with the vendor tag '`150`'.

13.2.5 hide

`hide` *partition* [Command]

Hide the partition *partition* by setting the *hidden* bit in its partition type code. This is useful only when booting DOS or Windows and multiple primary FAT partitions exist in one disk. See also Section 4.2.6 [DOS/Windows], page 13.

13.2.6 ifconfig

`ifconfig` [‘`--server=server`’] [‘`--gateway=gateway`’] [Command]
[‘`--mask=mask`’] [‘`--address=address`’]

Configure the IP address, the netmask, the gateway, and the server address of a network device manually. The values must be in dotted decimal format, like ‘`192.168.11.178`’. The order of the options is not important. This command shows current network configuration, if no option is specified. See also Chapter 6 [Network], page 21.

13.2.7 pager

`pager` [*flag*] [Command]

Toggle or set the state of the internal pager. If *flag* is ‘`on`’, the internal pager is enabled. If *flag* is ‘`off`’, it is disabled. If no argument is given, the state is toggled.

13.2.8 partnew

`partnew` *part type from len* [Command]

Create a new primary partition. *part* is a partition specification in GRUB syntax (see Chapter 2 [Naming convention], page 5); *type* is the partition type and must be a number in the range `0-0xff`; *from* is the starting address and *len* is the length, both in sector units.

13.2.9 parttype

`parttype` *part type* [Command]

Change the type of an existing partition. *part* is a partition specification in GRUB syntax (see Chapter 2 [Naming convention], page 5); *type* is the new partition type and must be a number in the range 0-0xff.

13.2.10 password

`password` [‘`--md5`’] *passwd* [*new-config-file*] [Command]

If used in the first section of a menu file, disable all interactive editing control (menu entry editor and command-line) and entries protected by the command `lock`. If the password *passwd* is entered, it loads the *new-config-file* as a new config file and restarts the GRUB Stage 2, if *new-config-file* is specified. Otherwise, GRUB will just unlock the privileged instructions. You can also use this command in the script section, in which case it will ask for the password, before continuing. The option ‘`--md5`’ tells GRUB that *passwd* is encrypted with `md5crypt` (see Section 13.3.24 [md5crypt], page 47).

13.2.11 rarp

`rarp` [Command]

Initialize a network device via the *RARP* protocol. This command is only available if GRUB is compiled with netboot support. See also Chapter 6 [Network], page 21.

13.2.12 serial

`serial` [‘`--unit=unit`’] [‘`--port=port`’] [‘`--speed=speed`’] [‘`--word=word`’] [‘`--parity=parity`’] [‘`--stop=stop`’] [‘`--device=dev`’] [Command]

Initialize a serial device. *unit* is a number in the range 0-3 specifying which serial port to use; default is 0, which corresponds to the port often called COM1. *port* is the I/O port where the UART is to be found; if specified it takes precedence over *unit*. *speed* is the transmission speed; default is 9600. *word* and *stop* are the number of data bits and stop bits. Data bits must be in the range 5-8 and stop bits must be 1 or 2. Default is 8 data bits and one stop bit. *parity* is one of ‘`no`’, ‘`odd`’, ‘`even`’ and defaults to ‘`no`’. The option ‘`--device`’ can only be used in the grub shell and is used to specify the tty device to be used in the host operating system (see Chapter 15 [Invoking the grub shell], page 55).

The serial port is not used as a communication channel unless the `terminal` command is used (see Section 13.2.14 [terminal], page 41).

This command is only available if GRUB is compiled with serial support. See also Chapter 7 [Serial terminal], page 23.

13.2.13 setkey

`setkey` [*to_key from_key*] [Command]

Change the keyboard map. The key *from_key* is mapped to the key *to_key*. If no argument is specified, reset key mappings. Note that this command *does not* exchange the keys. If you want to exchange the keys, run this command again with the arguments exchanged, like this:

```
grub> setkey capslock control
grub> setkey control capslock
```

A key must be an alphabet letter, a digit, or one of these symbols: ‘`escape`’, ‘`exclam`’, ‘`at`’, ‘`numbersign`’, ‘`dollar`’, ‘`percent`’, ‘`caret`’, ‘`ampersand`’, ‘`asterisk`’, ‘`parenleft`’, ‘`parenright`’, ‘`minus`’, ‘`underscore`’, ‘`equal`’, ‘`plus`’, ‘`backspace`’, ‘`tab`’, ‘`bracketleft`’, ‘`braceleft`’, ‘`bracketright`’, ‘`braceright`’, ‘`enter`’, ‘`control`’, ‘`semicolon`’, ‘`colon`’, ‘`quote`’, ‘`doublequote`’, ‘`backquote`’, ‘`tilde`’, ‘`shift`’, ‘`backslash`’, ‘`bar`’, ‘`comma`’, ‘`less`’, ‘`period`’, ‘`greater`’, ‘`slash`’, ‘`question`’, ‘`alt`’, ‘`space`’, ‘`capslock`’, ‘`FX`’ (‘`X`’ is a digit), and ‘`delete`’. This table describes to which character each of the symbols corresponds:

‘`exclam`’ ‘`!`’

‘`at`’ ‘`@`’

‘`numbersign`’ ‘`#`’

‘`dollar`’ ‘`$`’

‘`percent`’ ‘`%`’

‘`caret`’ ‘`^`’

‘`ampersand`’ ‘`&`’

‘asterisk’
‘*’

‘parenleft’
‘(’

‘parenright’
‘)’

‘minus’ ‘-’

‘underscore’
‘_’

‘equal’ ‘=’

‘plus’ ‘+’

‘bracketleft’
‘[’

‘braceleft’
‘{’

‘bracketright’
‘]’

‘braceright’
‘}’

‘semicolon’
‘;’

‘colon’ ‘:’

‘quote’ ‘’’

‘doublequote’
‘"’

‘backquote’
‘‘’

‘tilde’ ‘~’

‘backslash’
‘\’

‘bar’ ‘|’

‘comma’ ‘,’

‘less’ ‘<’

‘period’ ‘.’

‘greater’ ‘>’

‘slash’ ‘/’

‘question’
‘?’

‘space’ ‘ ’

13.2.14 terminal

`terminal` [‘`--dumb`’] [‘`--no-echo`’] [‘`--no-edit`’] [‘`--timeout=secs`’] [‘`--lines=lines`’] [‘`--silent`’] [‘`console`’] [‘`serial`’] [‘`hercules`’] [Command]

Select a terminal for user interaction. The terminal is assumed to be VT100-compatible unless ‘`--dumb`’ is specified. If both ‘`console`’ and ‘`serial`’ are specified, then GRUB will use the one where a key is entered first or the first when the timeout expires. If neither are specified, the current setting is reported. This command is only available if GRUB is compiled with serial support. See also Chapter 7 [Serial terminal], page 23.

This may not make sense for most users, but GRUB supports Hercules console as well. Hercules console is usable like the ordinary console, and the usage is quite similar to that for serial terminals: specify ‘`hercules`’ as the argument.

The option ‘`--lines`’ defines the number of lines in your terminal, and it is used for the internal pager function. If you don’t specify this option, the number is assumed as 24.

The option ‘`--silent`’ suppresses the message to prompt you to hit any key. This might be useful if your system has no terminal device.

The option ‘`--no-echo`’ has GRUB not to echo back input characters. This implies the option ‘`--no-edit`’.

The option ‘`--no-edit`’ disables the BASH-like editing feature.

13.2.15 terminfo

`terminfo` ‘`--name=name`’ ‘`--cursor-address=seq`’ [‘`--clear-screen=seq`’] [‘`--enter-standout-mode=seq`’] [‘`--exit-standout-mode=seq`’] [Command]

Define the capabilities of your terminal. Use this command to define escape sequences, if it is not vt100-compatible. You may use ‘`\e`’ for ⟨ESC⟩ and ‘`^X`’ for a control character.

You can use the utility `grub-terminfo` to generate appropriate arguments to this command. See Chapter 18 [Invoking grub-terminfo], page 63.

If no option is specified, the current settings are printed.

13.2.16 tftpserver

`tftpserver` *ipaddr* [Command]

Caution: This command exists only for backward compatibility. Use `ifconfig` (see Section 13.2.6 [ifconfig], page 38) instead.

Override a TFTP server address returned by a BOOTP/DHCP/RARP server. The argument *ipaddr* must be in dotted decimal format, like ‘`192.168.0.15`’. This command is only available if GRUB is compiled with netboot support. See also Chapter 6 [Network], page 21.

13.2.17 unhide

`unhide` *partition* [Command]

Unhide the partition *partition* by clearing the *hidden* bit in its partition type code. This is useful only when booting DOS or Windows and multiple primary partitions exist on one disk. See also Section 4.2.6 [DOS/Windows], page 13.

13.3 The list of command-line and menu entry commands

These commands are usable in the command-line and in menu entries. If you forget a command, you can run the command `help` (see Section 13.3.15 [help], page 44).

13.3.1 blocklist

`blocklist` *file* [Command]

Print the block list notation of the file *file*. See Section 11.3 [Block list syntax], page 32.

13.3.2 boot

`boot` [Command]

Boot the OS or chain-loader which has been loaded. Only necessary if running the fully interactive command-line (it is implicit at the end of a menu entry).

13.3.3 cat

`cat` *file* [Command]

Display the contents of the file *file*. This command may be useful to remind you of your OS's root partition:

```
grub> cat /etc/fstab
```

13.3.4 chainloader

`chainloader` [‘`--force`’] *file* [Command]

Load *file* as a chain-loader. Like any other file loaded by the filesystem code, it can use the blocklist notation to grab the first sector of the current partition with ‘`+1`’. If you specify the option ‘`--force`’, then load *file* forcibly, whether it has a correct signature or not. This is required when you want to load a defective boot loader, such as SCO UnixWare 7.1 (see Section 4.2.7 [SCO UnixWare], page 14).

13.3.5 cmp

`cmp` *file1 file2* [Command]

Compare the file *file1* with the file *file2*. If they differ in size, print the sizes like this:

```
Differ in size: 0x1234 [foo], 0x4321 [bar]
```

If the sizes are equal but the bytes at an offset differ, then print the bytes like this:

```
Differ at the offset 777: 0xbe [foo], 0xef [bar]
```

If they are completely identical, nothing will be printed.

13.3.6 configfile

`configfile` *file* [Command]
Load *file* as a configuration file.

13.3.7 debug

`debug` [Command]
Toggle debug mode (by default it is off). When debug mode is on, some extra messages are printed to show disk activity. This global debug flag is mainly useful for GRUB developers when testing new code.

13.3.8 displayapm

`displayapm` [Command]
Display APM BIOS information.

13.3.9 displaymem

`displaymem` [Command]
Display what GRUB thinks the system address space map of the machine is, including all regions of physical RAM installed. GRUB's *upper/lower memory* display uses the standard BIOS interface for the available memory in the first megabyte, or *lower memory*, and a synthesized number from various BIOS interfaces of the memory starting at 1MB and going up to the first chipset hole for *upper memory* (the standard PC *upper memory* interface is limited to reporting a maximum of 64MB).

13.3.10 embed

`embed` *stage1_5 device* [Command]
Embed the Stage 1.5 *stage1_5* in the sectors after the MBR if *device* is a drive, or in the *boot loader* area if *device* is a FFS partition or a ReiserFS partition.[1] Print the number of sectors which *stage1_5* occupies, if successful.

Usually, you don't need to run this command directly. See Section 13.3.34 [setup], page 48.

13.3.11 find

`find` *filename* [Command]
Search for the file name *filename* in all mountable partitions and print the list of the devices which contain the file. The file name *filename* should be an absolute file name like `/boot/grub/stage1`.

13.3.12 fstest

`fstest` [Command]
Toggle filesystem test mode. Filesystem test mode, when turned on, prints out data corresponding to all the device reads and what values are being sent to the low-

[1] The latter feature has not been implemented yet.

level routines. The format is '`<partition-offset-sector, byte-offset, byte-length>`' for high-level reads inside a partition, and '`[disk-offset-sector]`' for low-level sector requests from the disk. Filesystem test mode is turned off by any use of the `install` (see Section 13.3.18 [install], page 44) or `testload` (see Section 13.3.35 [testload], page 49) commands.

13.3.13 geometry

`geometry` *drive* [*cylinder head sector* [*total_sector*]] [Command]
Print the information for the drive *drive*. In the grub shell, you can set the geometry of the drive arbitrarily. The number of cylinders, the number of heads, the number of sectors and the number of total sectors are set to CYLINDER, HEAD, SECTOR and TOTAL_SECTOR, respectively. If you omit TOTAL_SECTOR, then it will be calculated based on the C/H/S values automatically.

13.3.14 halt

`halt` '`--no-apm`' [Command]
The command halts the computer. If the '`--no-apm`' option is specified, no APM BIOS call is performed. Otherwise, the computer is shut down using APM.

13.3.15 help

`help` '`--all`' [*pattern* ...] [Command]
Display helpful information about builtin commands. If you do not specify *pattern*, this command shows short descriptions of most of available commands. If you specify the option '`--all`' to this command, short descriptions of rarely used commands (such as Section 13.3.35 [testload], page 49) are displayed as well.

If you specify any *patterns*, it displays longer information about each of the commands which match those *patterns*.

13.3.16 impsprobe

`impsprobe` [Command]
Probe the Intel Multiprocessor Specification 1.1 or 1.4 configuration table and boot the various CPUs which are found into a tight loop. This command can be used only in the Stage 2, but not in the grub shell.

13.3.17 initrd

`initrd` *file* ... [Command]
Load an initial ramdisk for a Linux format boot image and set the appropriate parameters in the Linux setup area in memory. See also Section 4.2.2 [GNU/Linux], page 12.

13.3.18 install

`install` [‘`--force-lba`’] [‘`--stage2=os_stage2_file`’] *stage1_file* [‘`d`’] *dest_dev stage2_file* [*addr*] [‘`p`’] [*config_file*] [*real_config_file*] [Command]

This command is fairly complex, and you should not use this command unless you are familiar with GRUB. Use `setup` (see Section 13.3.34 [setup], page 48) instead.

In short, it will perform a full install presuming the Stage 2 or Stage 1.5[2] is in its final install location.

In slightly more detail, it will load *stage1_file*, validate that it is a GRUB Stage 1 of the right version number, install in it a blocklist for loading *stage2_file* as a Stage 2. If the option ‘`d`’ is present, the Stage 1 will always look for the actual disk *stage2_file* was installed on, rather than using the booting drive. The Stage 2 will be loaded at address *addr*, which must be ‘`0x8000`’ for a true Stage 2, and ‘`0x2000`’ for a Stage 1.5. If *addr* is not present, GRUB will determine the address automatically. It then writes the completed Stage 1 to the first block of the device *dest_dev*. If the options ‘`p`’ or *config_file* are present, then it reads the first block of stage2, modifies it with the values of the partition *stage2_file* was found on (for ‘`p`’) or places the string *config_file* into the area telling the stage2 where to look for a configuration file at boot time. Likewise, if *real_config_file* is present and *stage2_file* is a Stage 1.5, then the Stage 2 *config_file* is patched with the configuration file name *real_config_file*. This command preserves the DOS BPB (and for hard disks, the partition table) of the sector the Stage 1 is to be installed into.

Caution: Several buggy BIOSes don’t pass a booting drive properly when booting from a hard disk drive. Therefore, you will unfortunately have to specify the option ‘`d`’, whether your Stage2 resides at the booting drive or not, if you have such a BIOS. We know these are defective in this way:

Fujitsu LifeBook 400 BIOS version 31J0103A

HP Vectra XU 6/200 BIOS version GG.06.11

Caution2: A number of BIOSes don’t return a correct LBA support bitmap even if they do have the support. So GRUB provides a solution to ignore the wrong bitmap, that is, the option ‘`--force-lba`’. Don’t use this option if you know that your BIOS doesn’t have LBA support.

Caution3: You must specify the option ‘`--stage2`’ in the grub shell, if you cannot unmount the filesystem where your stage2 file resides. The argument should be the file name in your operating system.

13.3.19 ioprobe

`ioprobe` *drive* [Command]

Probe I/O ports used for the drive *drive*. This command will list the I/O ports on the screen. For technical information, See Appendix D [Internals], page 75.

[2] They’re loaded the same way, so we will refer to the Stage 1.5 as a Stage 2 from now on.

13.3.20 kernel

`kernel` [‘`--type=type`’] [‘`--no-mem-option`’] *file* . . . [Command]

Attempt to load the primary boot image (Multiboot a.out or ELF, Linux zImage or bzImage, FreeBSD a.out, NetBSD a.out, etc.) from *file*. The rest of the line is passed verbatim as the *kernel command-line*. Any modules must be reloaded after using this command.

This command also accepts the option ‘`--type`’ so that you can specify the kernel type of *file* explicitly. The argument *type* must be one of these: ‘`netbsd`’, ‘`freebsd`’, ‘`openbsd`’, ‘`linux`’, ‘`biglinux`’, and ‘`multiboot`’. However, you need to specify it only if you want to load a NetBSD ELF kernel, because GRUB can automatically determine a kernel type in the other cases, quite safely.

The option ‘`--no-mem-option`’ is effective only for Linux. If the option is specified, GRUB doesn’t pass the option ‘`mem=`’ to the kernel. This option is implied for Linux kernels 2.4.18 and newer.

13.3.21 lock

`lock` [Command]

Prevent normal users from executing arbitrary menu entries. You must use the command `password` if you really want this command to be useful (see Section 13.2.10 [password], page 38).

This command is used in a menu, as shown in this example:

```
title This entry is too dangerous to be executed by normal users
lock
root (hd0,a)
kernel /no-security-os
```

See also Chapter 9 [Security], page 27.

13.3.22 makeactive

`makeactive` [Command]

Set the active partition on the root disk to GRUB’s root device. This command is limited to *primary* PC partitions on a hard disk.

13.3.23 map

`map` *to_drive from_drive* [Command]

Map the drive *from_drive* to the drive *to_drive*. This is necessary when you chain-load some operating systems, such as DOS, if such an OS resides at a non-first drive. Here is an example:

```
grub> map (hd0) (hd1)
grub> map (hd1) (hd0)
```

The example exchanges the order between the first hard disk and the second hard disk. See also Section 4.2.6 [DOS/Windows], page 13.

13.3.24 md5crypt

`md5crypt` [Command]
Prompt to enter a password, and encrypt it in MD5 format. The encrypted password can be used with the command `password` (see Section 13.2.10 [password], page 38). See also Chapter 9 [Security], page 27.

13.3.25 module

`module` *file ...* [Command]
Load a boot module *file* for a Multiboot format boot image (no interpretation of the file contents are made, so the user of this command must know what the kernel in question expects). The rest of the line is passed as the *module command-line*, like the `kernel` command. You must load a Multiboot kernel image before loading any module. See also Section 13.3.26 [modulenounzip], page 47.

13.3.26 modulenounzip

`modulenounzip` *file ...* [Command]
The same as `module` (see Section 13.3.25 [module], page 47), except that automatic decompression is disabled.

13.3.27 pause

`pause` *message ...* [Command]
Print the *message*, then wait until a key is pressed. Note that placing ⟨^G⟩ (ASCII code 7) in the message will cause the speaker to emit the standard beep sound, which is useful when prompting the user to change floppies.

13.3.28 quit

`quit` [Command]
Exit from the grub shell `grub` (see Chapter 15 [Invoking the grub shell], page 55). This command can be used only in the grub shell.

13.3.29 reboot

`reboot` [Command]
Reboot the computer.

13.3.30 read

`read` *addr* [Command]
Read a 32-bit value from memory at address *addr* and display it in hex format.

13.3.31 root

`root` *device* [*hdbias*] [Command]
Set the current *root device* to the device *device*, then attempt to mount it to get the partition size (for passing the partition descriptor in `ES:ESI`, used by some chain-loaded boot loaders), the BSD drive-type (for booting BSD kernels using their native

boot format), and correctly determine the PC partition where a BSD sub-partition is located. The optional *hdbias* parameter is a number to tell a BSD kernel how many BIOS drive numbers are on controllers before the current one. For example, if there is an IDE disk and a SCSI disk, and your FreeBSD root partition is on the SCSI disk, then use a '`1`' for *hdbias*.

See also Section 13.3.32 [rootnoverify], page 48.

13.3.32 rootnoverify

`rootnoverify` *device* [*hdbias*] [Command]

Similar to `root` (see Section 13.3.31 [root], page 47), but don't attempt to mount the partition. This is useful for when an OS is outside of the area of the disk that GRUB can read, but setting the correct root device is still desired. Note that the items mentioned in `root` above which derived from attempting the mount will *not* work correctly.

13.3.33 savedefault

`savedefault` *num* [Command]

Save the current menu entry or *num* if specified as a default entry. Here is an example:

```
default saved
timeout 10

title GNU/Linux
root (hd0,0)
kernel /boot/vmlinuz root=/dev/sda1 vga=ext
initrd /boot/initrd
savedefault

title FreeBSD
root (hd0,a)
kernel /boot/loader
savedefault
```

With this configuration, GRUB will choose the entry booted previously as the default entry.

You can specify '`fallback`' instead of a number. Then, next fallback entry is saved. Next fallback entry is chosen from fallback entries. Normally, this will be the first entry in fallback ones.

See also Section 13.1.1 [default], page 35 and Chapter 19 [Invoking grub-set-default], page 65.

13.3.34 setup

`setup` ['`--force-lba`'] ['`--stage2=os_stage2_file`'] ['`--prefix=dir`'] *install_device* [*image_device*] [Command]

Set up the installation of GRUB automatically. This command uses the more flexible command `install` (see Section 13.3.18 [install], page 44) in the backend and installs

GRUB into the device *install_device*. If *image_device* is specified, then find the GRUB images (see Chapter 10 [Images], page 29) in the device *image_device*, otherwise use the current *root device*, which can be set by the command `root`. If *install_device* is a hard disk, then embed a Stage 1.5 in the disk if possible.

The option '`--prefix`' specifies the directory under which GRUB images are put. If it is not specified, GRUB automatically searches them in '`/boot/grub`' and '`/grub`'.

The options '`--force-lba`' and '`--stage2`' are just passed to `install` if specified. See Section 13.3.18 [install], page 44, for more information.

13.3.35 testload

`testload` *file* [Command]

Read the entire contents of *file* in several different ways and compare them, to test the filesystem code. The output is somewhat cryptic, but if no errors are reported and the final '`i=X, filepos=Y`' reading has X and Y equal, then it is definitely consistent, and very likely works correctly subject to a consistent offset error. If this test succeeds, then a good next step is to try loading a kernel.

13.3.36 testvbe

`testvbe` *mode* [Command]

Test the VESA BIOS EXTENSION mode *mode*. This command will switch your video card to the graphics mode, and show an endless animation. Hit any key to return. See also Section 13.3.38 [vbeprobe], page 49.

13.3.37 uppermem

`uppermem` *kbytes* [Command]

Force GRUB to assume that only *kbytes* kilobytes of upper memory are installed. Any system address range maps are discarded.

Caution: This should be used with great caution, and should only be necessary on some old machines. GRUB's BIOS probe can pick up all RAM on all new machines the author has ever heard of. It can also be used for debugging purposes to lie to an OS.

13.3.38 vbeprobe

`vbeprobe` [*mode*] [Command]

Probe VESA BIOS EXTENSION information. If the mode *mode* is specified, show only the information about *mode*. Otherwise, this command lists up available VBE modes on the screen. See also Section 13.3.36 [testvbe], page 49.

14 Error messages reported by GRUB

This chapter describes error messages reported by GRUB when you encounter trouble. See Chapter 15 [Invoking the grub shell], page 55, if your problem is specific to the grub shell.

14.1 Errors reported by the Stage 1

The general way that the Stage 1 handles errors is to print an error string and then halt. Pressing ⟨CTRL⟩-⟨ALT⟩-⟨DEL⟩ will reboot.

The following is a comprehensive list of error messages for the Stage 1:

Hard Disk Error
: The stage2 or stage1.5 is being read from a hard disk, and the attempt to determine the size and geometry of the hard disk failed.

Floppy Error
: The stage2 or stage1.5 is being read from a floppy disk, and the attempt to determine the size and geometry of the floppy disk failed. It's listed as a separate error since the probe sequence is different than for hard disks.

Read Error
: A disk read error happened while trying to read the stage2 or stage1.5.

Geom Error
: The location of the stage2 or stage1.5 is not in the portion of the disk supported directly by the BIOS read calls. This could occur because the BIOS translated geometry has been changed by the user or the disk is moved to another machine or controller after installation, or GRUB was not installed using itself (if it was, the Stage 2 version of this error would have been seen during that process and it would not have completed the install).

14.2 Errors reported by the Stage 1.5

The general way that the Stage 1.5 handles errors is to print an error number in the form `Error` *`num`* and then halt. Pressing ⟨CTRL⟩-⟨ALT⟩-⟨DEL⟩ will reboot.

The error numbers correspond to the errors reported by Stage 2. See Section 14.3 [Stage2 errors], page 51.

14.3 Errors reported by the Stage 2

The general way that the Stage 2 handles errors is to abort the operation in question, print an error string, then (if possible) either continue based on the fact that an error occurred or wait for the user to deal with the error.

The following is a comprehensive list of error messages for the Stage 2 (error numbers for the Stage 1.5 are listed before the colon in each description):

1 : Filename must be either an absolute filename or blocklist
: This error is returned if a file name is requested which doesn't fit the syntax/rules listed in the Chapter 11 [Filesystem], page 31.

2 : Bad file or directory type
: This error is returned if a file requested is not a regular file, but something like a symbolic link, directory, or FIFO.

3 : Bad or corrupt data while decompressing file
: This error is returned if the run-length decompression code gets an internal error. This is usually from a corrupt file.

4 : Bad or incompatible header in compressed file
: This error is returned if the file header for a supposedly compressed file is bad.

5 : Partition table invalid or corrupt
: This error is returned if the sanity checks on the integrity of the partition table fail. This is a bad sign.

6 : Mismatched or corrupt version of stage1/stage2
: This error is returned if the install command points to incompatible or corrupt versions of the stage1 or stage2. It can't detect corruption in general, but this is a sanity check on the version numbers, which should be correct.

7 : Loading below 1MB is not supported
: This error is returned if the lowest address in a kernel is below the 1MB boundary. The Linux zImage format is a special case and can be handled since it has a fixed loading address and maximum size.

8 : Kernel must be loaded before booting
: This error is returned if GRUB is told to execute the boot sequence without having a kernel to start.

9 : Unknown boot failure
: This error is returned if the boot attempt did not succeed for reasons which are unknown.

10 : Unsupported Multiboot features requested
: This error is returned when the Multiboot features word in the Multiboot header requires a feature that is not recognized. The point of this is that the kernel requires special handling which GRUB is probably unable to provide.

11 : Unrecognized device string
: This error is returned if a device string was expected, and the string encountered didn't fit the syntax/rules listed in the Chapter 11 [Filesystem], page 31.

12 : Invalid device requested
: This error is returned if a device string is recognizable but does not fall under the other device errors.

13 : Invalid or unsupported executable format
: This error is returned if the kernel image being loaded is not recognized as Multiboot or one of the supported native formats (Linux zImage or bzImage, FreeBSD, or NetBSD).

14 : Filesystem compatibility error, cannot read whole file
: Some of the filesystem reading code in GRUB has limits on the length of the files it can read. This error is returned when the user runs into such a limit.

15 : File not found
: This error is returned if the specified file name cannot be found, but everything else (like the disk/partition info) is OK.

16 : Inconsistent filesystem structure
: This error is returned by the filesystem code to denote an internal error caused by the sanity checks of the filesystem structure on disk not matching what it expects. This is usually caused by a corrupt filesystem or bugs in the code handling it in GRUB.

17 : Cannot mount selected partition
: This error is returned if the partition requested exists, but the filesystem type cannot be recognized by GRUB.

18 : Selected cylinder exceeds maximum supported by BIOS
: This error is returned when a read is attempted at a linear block address beyond the end of the BIOS translated area. This generally happens if your disk is larger than the BIOS can handle (512MB for (E)IDE disks on older machines or larger than 8GB in general).

19 : Linux kernel must be loaded before initrd
: This error is returned if the initrd command is used before loading a Linux kernel.

20 : Multiboot kernel must be loaded before modules
: This error is returned if the module load command is used before loading a Multiboot kernel. It only makes sense in this case anyway, as GRUB has no idea how to communicate the presence of such modules to a non-Multiboot-aware kernel.

21 : Selected disk does not exist
: This error is returned if the device part of a device- or full file name refers to a disk or BIOS device that is not present or not recognized by the BIOS in the system.

22 : No such partition
: This error is returned if a partition is requested in the device part of a device- or full file name which isn't on the selected disk.

23 : Error while parsing number
: This error is returned if GRUB was expecting to read a number and encountered bad data.

24 : Attempt to access block outside partition
: This error is returned if a linear block address is outside of the disk partition. This generally happens because of a corrupt filesystem on the disk or a bug in the code handling it in GRUB (it's a great debugging tool).

25 : Disk read error
: This error is returned if there is a disk read error when trying to probe or read data from a particular disk.

26 : Too many symbolic links
: This error is returned if the link count is beyond the maximum (currently 5), possibly the symbolic links are looped.

27 : Unrecognized command
: This error is returned if an unrecognized command is entered on the command-line or in a boot sequence section of a configuration file and that entry is selected.

28 : Selected item cannot fit into memory
: This error is returned if a kernel, module, or raw file load command is either trying to load its data such that it won't fit into memory or it is simply too big.

29 : Disk write error
: This error is returned if there is a disk write error when trying to write to a particular disk. This would generally only occur during an install of set active partition command.

30 : Invalid argument
: This error is returned if an argument specified to a command is invalid.

31 : File is not sector aligned
: This error may occur only when you access a ReiserFS partition by block-lists (e.g. the command `install`). In this case, you should mount the partition with the '`-o notail`' option.

32 : Must be authenticated
: This error is returned if you try to run a locked entry. You should enter a correct password before running such an entry.

33 : Serial device not configured
: This error is returned if you try to change your terminal to a serial one before initializing any serial device.

34 : No spare sectors on the disk
: This error is returned if a disk doesn't have enough spare space. This happens when you try to embed Stage 1.5 into the unused sectors after the MBR, but the first partition starts right after the MBR or they are used by EZ-BIOS.

15 Invoking the grub shell

This chapter documents the grub shell `grub`. Note that the grub shell is an emulator; it doesn't run under the native environment, so it sometimes does something wrong. Therefore, you shouldn't trust it too much. If there is anything wrong with it, don't hesitate to try the native GRUB environment, especially when it guesses a wrong map between BIOS drives and OS devices.

15.1 Introduction into the grub shell

You can use the command `grub` for installing GRUB under your operating systems and for a testbed when you add a new feature into GRUB or when fixing a bug. `grub` is almost the same as the Stage 2, and, in fact, it shares the source code with the Stage 2 and you can use the same commands (see Chapter 13 [Commands], page 35) in `grub`. It is emulated by replacing BIOS calls with UNIX system calls and libc functions.

The command `grub` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version number of GRUB and exit.

'`--verbose`'
Print some verbose messages for debugging purpose.

'`--device-map=`*file*'
Use the device map file *file*. The format is described in Section 15.3 [Device map], page 56.

'`--no-floppy`'
Do not probe any floppy drive. This option has no effect if the option '`--device-map`' is specified (see Section 15.3 [Device map], page 56).

'`--probe-second-floppy`'
Probe the second floppy drive. If this option is not specified, the grub shell does not probe it, as that sometimes takes a long time. If you specify the device map file (see Section 15.3 [Device map], page 56), the grub shell just ignores this option.

'`--config-file=`*file*'
Read the configuration file *file* instead of '`/boot/grub/menu.lst`'. The format is the same as the normal GRUB syntax. See Chapter 11 [Filesystem], page 31, for more information.

'`--boot-drive=`*drive*'
Set the stage2 *boot_drive* to *drive*. This argument should be an integer (decimal, octal or hexadecimal).

'`--install-partition=`*par*'
Set the stage2 *install_partition* to *par*. This argument should be an integer (decimal, octal or hexadecimal).

‘`--no-config-file`’
: Do not use the configuration file even if it can be read.

‘`--no-curses`’
: Do not use the screen handling interface by the curses even if it is available.

‘`--batch`’ This option has the same meaning as ‘`--no-config-file --no-curses`’.

‘`--read-only`’
: Disable writing to any disk.

‘`--hold`’ Wait until a debugger will attach. This option is useful when you want to debug the startup code.

15.2 How to install GRUB via `grub`

The installation procedure is the same as under the *native* Stage 2. See Chapter 3 [Installation], page 7, for more information. The command `grub`-specific information is described here.

What you should be careful about is *buffer cache*. `grub` makes use of raw devices instead of filesystems that your operating systems serve, so there exists a potential problem that some cache inconsistency may corrupt your filesystems. What we recommend is:

- If you can unmount drives to which GRUB may write any amount of data, unmount them before running `grub`.
- If a drive cannot be unmounted but can be mounted with the read-only flag, mount it in read-only mode. That should be secure.
- If a drive must be mounted with the read-write flag, make sure that no activity is being done on it while the command `grub` is running.
- Reboot your operating system as soon as possible. This is probably not required if you follow the rules above, but reboot is the most secure way.

In addition, enter the command `quit` when you finish the installation. That is *very important* because `quit` makes the buffer cache consistent. Do not push ⟨C-c⟩.

If you want to install GRUB non-interactively, specify ‘`--batch`’ option in the command-line. This is a simple example:

```
#!/bin/sh

# Use /usr/sbin/grub if you are on an older system.
/sbin/grub --batch <<EOT 1>/dev/null 2>/dev/null
root (hd0,0)
setup (hd0)
quit
EOT
```

15.3 The map between BIOS drives and OS devices

When you specify the option ‘`--device-map`’ (see Section 15.1 [Basic usage], page 55), the grub shell creates the *device map file* automatically unless it already exists. The file name ‘`/boot/grub/device.map`’ is preferred.

If the device map file exists, the grub shell reads it to map BIOS drives to OS devices. This file consists of lines like this:

```
device file
```

device is a drive specified in the GRUB syntax (see Section 11.1 [Device syntax], page 31), and *file* is an OS file, which is normally a device file.

The reason why the grub shell gives you the device map file is that it cannot guess the map between BIOS drives and OS devices correctly in some environments. For example, if you exchange the boot sequence between IDE and SCSI in your BIOS, it gets the order wrong.

Thus, edit the file if the grub shell makes a mistake. You can put any comments in the file if needed, as the grub shell assumes that a line is just a comment if the first character is '`#`'.

16 Invoking grub-install

The program `grub-install` installs GRUB on your drive using the grub shell (see Chapter 15 [Invoking the grub shell], page 55). You must specify the device name on which you want to install GRUB, like this:

```
grub-install install_device
```

The device name *install_device* is an OS device name or a GRUB device name.

`grub-install` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version number of GRUB and exit.

'`--force-lba`'
Force GRUB to use LBA mode even for a buggy BIOS. Use this option only if your BIOS doesn't work properly in LBA mode even though it supports LBA mode.

'`--root-directory=dir`'
Install GRUB images under the directory *dir* instead of the root directory. This option is useful when you want to install GRUB into a separate partition or a removable disk. Here is an example in which you have a separate *boot* partition which is mounted on '`/boot`':

```
grub-install --root-directory=/boot hd0
```

'`--grub-shell=file`'
Use *file* as the grub shell. You can append arbitrary options to *file* after the file name, like this:

```
grub-install --grub-shell="grub --read-only" /dev/fd0
```

'`--recheck`'
Recheck the device map, even if '`/boot/grub/device.map`' already exists. You should use this option whenever you add/remove a disk into/from your computer.

17 Invoking grub-md5-crypt

The program `grub-md5-crypt` encrypts a password in MD5 format. This is just a frontend of the grub shell (see Chapter 15 [Invoking the grub shell], page 55). Passwords encrypted by this program can be used with the command `password` (see Section 13.2.10 [password], page 38).

`grub-md5-crypt` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version information and exit.

'`--grub-shell=`*`file`*'
Use *file* as the grub shell.

18 Invoking grub-terminfo

The program `grub-terminfo` generates a terminfo command from a terminfo name (see Section 13.2.15 [terminfo], page 41). The result can be used in the configuration file, to define escape sequences. Because GRUB assumes that your terminal is vt100-compatible by default, this would be useful only if your terminal is uncommon (such as vt52).

`grub-terminfo` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version information and exit.

You must specify one argument to this command. For example:

```
grub-terminfo vt52
```

19 Invoking grub-set-default

The program `grub-set-default` sets the default boot entry for GRUB. This automatically creates a file named '`default`' under your GRUB directory (i.e. '`/boot/grub`'), if it is not present. This file is used to determine the default boot entry when GRUB boots up your system when you use '`default saved`' in your configuration file (see Section 13.1.1 [default], page 35), and to save next default boot entry when you use '`savedefault`' in a boot entry (see Section 13.3.33 [savedefault], page 48).

`grub-set-default` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version information and exit.

'`--root-directory=`*dir*'
Use the directory *dir* instead of the root directory (i.e. '`/`') to define the location of the default file. This is useful when you mount a disk which is used for another system.

You must specify a single argument to `grub-set-default`. This argument is normally the number of a default boot entry. For example, if you have this configuration file:

```
default saved
timeout 10

title GNU/Hurd
root (hd0,0)
...

title GNU/Linux
root (hd0,1)
...
```

and if you want to set the next default boot entry to GNU/Linux, you may execute this command:

```
grub-set-default 1
```

Because the entry for GNU/Linux is '`1`'. Note that entries are counted from zero. So, if you want to specify GNU/Hurd here, then you should specify '`0`'.

This feature is very useful if you want to test a new kernel or to make your system quite robust. See Section 4.3 [Making your system robust], page 14, for more hints about how to set up a robust system.

20 Invoking mbchk

The program `mbchk` checks for the format of a Multiboot kernel. We recommend using this program before booting your own kernel by GRUB.

`mbchk` accepts the following options:

'`--help`' Print a summary of the command-line options and exit.

'`--version`'
Print the version number of GRUB and exit.

'`--quiet`' Suppress all normal output.

Appendix A How to obtain and build GRUB

Caution: GRUB requires binutils-2.9.1.0.23 or later because the GNU assembler has been changed so that it can produce real 16bits machine code between 2.9.1 and 2.9.1.0.x. See http://sources.redhat.com/binutils/, to obtain information on how to get the latest version.

GRUB is available from the GNU alpha archive site ftp://alpha.gnu.org/gnu/grub or any of its mirrors. The file will be named grub-version.tar.gz. The current version is 0.96, so the file you should grab is:

ftp://alpha.gnu.org/gnu/grub/grub-0.96.tar.gz

To unbundle GRUB use the instruction:

```
zcat grub-0.96.tar.gz | tar xvf -
```

which will create a directory called '`grub-0.96`' with all the sources. You can look at the file '`INSTALL`' for detailed instructions on how to build and install GRUB, but you should be able to just do:

```
cd grub-0.96
./configure
make install
```

This will install the grub shell '`grub`' (see Chapter 15 [Invoking the grub shell], page 55), the Multiboot checker '`mbchk`' (see Chapter 20 [Invoking mbchk], page 67), and the GRUB images. This will also install the GRUB manual.

Also, the latest version is available from the CVS. See http://savannah.gnu.org/cvs/?group=grub for more information.

Appendix B Reporting bugs

These are the guideline for how to report bugs. Take a look at this list below before you submit bugs:

1. Before getting unsettled, read this manual through and through. Also, see the GNU GRUB FAQ.
2. Always mention the information on your GRUB. The version number and the configuration are quite important. If you build it yourself, write the options specified to the configure script and your operating system, including the versions of gcc and binutils.
3. If you have trouble with the installation, inform us of how you installed GRUB. Don't omit error messages, if any. Just '`GRUB hangs up when it boots`' is not enough.

   The information on your hardware is also essential. These are especially important: the geometries and the partition tables of your hard disk drives and your BIOS.
4. If GRUB cannot boot your operating system, write down *everything* you see on the screen. Don't paraphrase them, like '`The foo OS crashes with GRUB, even though it can boot with the bar boot loader just fine`'. Mention the commands you executed, the messages printed by them, and information on your operating system including the version number.
5. Explain what you wanted to do. It is very useful to know your purpose and your wish, and how GRUB didn't satisfy you.
6. If you can investigate the problem yourself, please do. That will give you and us much more information on the problem. Attaching a patch is even better.

   When you attach a patch, make the patch in unified diff format, and write ChangeLog entries. But, even when you make a patch, don't forget to explain the problem, so that we can understand what your patch is for.
7. Write down anything that you think might be related. Please understand that we often need to reproduce the same problem you encounterred in our environment. So your information should be sufficient for us to do the same thing—Don't forget that we cannot see your computer directly. If you are not sure whether to state a fact or leave it out, state it! Reporting too many things is much better than omitting something important.

If you follow the guideline above, submit a report to the Bug Tracking System. Alternatively, you can submit a report via electronic mail to `bug-grub@gnu.org`, but we strongly recommend that you use the Bug Tracking System, because e-mail can be passed over easily.

Once we get your report, we will try to fix the bugs.

Appendix C Where GRUB will go

We started the next generation of GRUB, GRUB 2. This will include internationalization, dynamic module loading, real memory management, multiple architecture support, a scripting language, and many other nice feature. If you are interested in the development of GRUB 2, take a look at the homepage.

Appendix D Hacking GRUB

This chapter documents the user-invisible aspect of GRUB.

As a general rule of software development, it is impossible to keep the descriptions of the internals up-to-date, and it is quite hard to document everything. So refer to the source code, whenever you are not satisfied with this documentation. Please assume that this gives just hints to you.

D.1 The memory map of various components

GRUB consists of two distinct components, called *stages*, which are loaded at different times in the boot process. Because they run mutual-exclusively, sometimes a memory area overlaps with another memory area. And, even in one stage, a single memory area can be used for various purposes, because their usages are mutually exclusive.

Here is the memory map of the various components:

0 to 4K-1 BIOS and real mode interrupts

0x07BE to 0x07FF
: Partition table passed to another boot loader

down from 8K-1
: Real mode stack

0x2000 to ?
: The optional Stage 1.5 is loaded here

0x2000 to 0x7FFF
: Command-line buffer for Multiboot kernels and modules

0x7C00 to 0x7DFF
: Stage 1 is loaded here by BIOS or another boot loader

0x7F00 to 0x7F42
: LBA drive parameters

0x8000 to ?
: Stage2 is loaded here

The end of Stage 2 to 416K-1
: Heap, in particular used for the menu

down from 416K-1
: Protected mode stack

416K to 448K-1
: Filesystem buffer

448K to 479.5K-1
: Raw device buffer

479.5K to 480K-1
: 512-byte scratch area

480K to 512K-1
: Buffers for various functions, such as password, command-line, cut and paste, and completion.

The last 1K of lower memory
: Disk swapping code and data

See the file ‘`stage2/shared.h`’, for more information.

D.2 Embedded variables in GRUB

Stage 1 and Stage 2 have embedded variables whose locations are well-defined, so that the installation can patch the binary file directly without recompilation of the stages.

In Stage 1, these are defined:

`0x3E`
: The version number (not GRUB's, but the installation mechanism's).

`0x40`
: The boot drive. If it is 0xFF, use a drive passed by BIOS.

`0x41`
: The flag for if forcing LBA.

`0x42`
: The starting address of Stage 2.

`0x44`
: The first sector of Stage 2.

`0x48`
: The starting segment of Stage 2.

`0x1FE`
: The signature (`0xAA55`).

See the file ‘`stage1/stage1.S`’, for more information.

In the first sector of Stage 1.5 and Stage 2, the block lists are recorded between `firstlist` and `lastlist`. The address of `lastlist` is determined when assembling the file ‘`stage2/start.S`’.

The trick here is that it is actually read backward, and the first 8-byte block list is not read here, but after the pointer is decremented 8 bytes, then after reading it, it decrements again, reads, and so on, until it is finished. The terminating condition is when the number of sectors to be read in the next block list is zero.

The format of a block list can be seen from the example in the code just before the `firstlist` label. Note that it is always from the beginning of the disk, but *not* relative to the partition boundaries.

In the second sector of Stage 1.5 and Stage 2, these are defined:

`0x6`
: The version number (likewise, the installation mechanism's).

`0x8`
: The installed partition.

`0xC`
: The saved entry number.

`0x10`
: The identifier.

`0x11`
: The flag for if forcing LBA.

`0x12`
: The version string (GRUB's).

`0x12` + *the length of the version string*
: The name of a configuration file.

See the file ‘`stage2/asm.S`’, for more information.

D.3 The generic interface for filesystems

For any particular partition, it is presumed that only one of the *normal* filesystems such as FAT, FFS, or ext2fs can be used, so there is a switch table managed by the functions in '`disk_io.c`'. The notation is that you can only *mount* one at a time.

The block list filesystem has a special place in the system. In addition to the *normal* filesystem (or even without one mounted), you can access disk blocks directly (in the indicated partition) via the block list notation. Using the block list filesystem doesn't effect any other filesystem mounts.

The variables which can be read by the filesystem backend are:

`current_drive`
: The current BIOS drive number (numbered from 0, if a floppy, and numbered from 0x80, if a hard disk).

`current_partition`
: The current partition number.

`current_slice`
: The current partition type.

`saved_drive`
: The *drive* part of the root device.

`saved_partition`
: The *partition* part of the root device.

`part_start`
: The current partition starting address, in sectors.

`part_length`
: The current partition length, in sectors.

`print_possibilities`
: True when the `dir` function should print the possible completions of a file, and false when it should try to actually open a file of that name.

`FSYS_BUF` Filesystem buffer which is 32K in size, to use in any way which the filesystem backend desires.

The variables which need to be written by a filesystem backend are:

`filepos` The current position in the file, in sectors.

Caution: the value of *filepos* can be changed out from under the filesystem code in the current implementation. Don't depend on it being the same for later calls into the backend code!

`filemax` The length of the file.

`disk_read_func`
: The value of *disk_read_hook only* during reading of data for the file, not any other fs data, inodes, FAT tables, whatever, then set to `NULL` at all other times (it will be `NULL` by default). If this isn't done correctly, then the `testload` and `install` commands won't work correctly.

The functions expected to be used by the filesystem backend are:

`devread` Only read sectors from within a partition. Sector 0 is the first sector in the partition.

`grub_read`
If the backend uses the block list code, then `grub_read` can be used, after setting *block_file* to 1.

`print_a_completion`
If *print_possibilities* is true, call `print_a_completion` for each possible file name. Otherwise, the file name completion won't work.

The functions expected to be defined by the filesystem backend are described at least moderately in the file '`filesys.h`'. Their usage is fairly evident from their use in the functions in '`disk_io.c`', look for the use of the *fsys_table* array.

Caution: The semantics are such that then '`mount`'ing the filesystem, presume the filesystem buffer `FSYS_BUF` is corrupted, and (re-)load all important contents. When opening and reading a file, presume that the data from the '`mount`' is available, and doesn't get corrupted by the open/read (i.e. multiple opens and/or reads will be done with only one mount if in the same filesystem).

D.4 The generic interface for built-ins

GRUB built-in commands are defined in a uniformal interface, whether they are menu-specific or can be used anywhere. The definition of a builtin command consists of two parts: the code itself and the table of the information.

The code must be a function which takes two arguments, a command-line string and flags, and returns an '`int`' value. The *flags* argument specifies how the function is called, using a bit mask. The return value must be zero if successful, otherwise non-zero. So it is normally enough to return *errnum*.

The table of the information is represented by the structure `struct builtin`, which contains the name of the command, a pointer to the function, flags, a short description of the command and a long description of the command. Since the descriptions are used only for help messages interactively, you don't have to define them, if the command may not be called interactively (such as `title`).

The table is finally registered in the table *builtin_table*, so that `run_script` and `enter_cmdline` can find the command. See the files '`cmdline.c`' and '`builtins.c`', for more details.

D.5 The bootstrap mechanism used in GRUB

The disk space can be used in a boot loader is very restricted because a MBR (see Section D.9 [MBR], page 79) is only 512 bytes but it also contains a partition table (see Section D.10 [Partition table], page 79) and a BPB. So the question is how to make a boot loader code enough small to be fit in a MBR.

However, GRUB is a very large program, so we break GRUB into 2 (or 3) distinct components, *Stage 1* and *Stage 2* (and optionally *Stage 1.5*). See Section D.1 [Memory map], page 75, for more information.

We embed Stage 1 in a MBR or in the boot sector of a partition, and place Stage 2 in a filesystem. The optional Stage 1.5 can be installed in a filesystem, in the *boot loader* area in a FFS or a ReiserFS, and in the sectors right after a MBR, because Stage 1.5 is enough small and the sectors right after a MBR is normally an unused region. The size of this region is the number of sectors per head minus 1.

Thus, all Stage1 must do is just load Stage2 or Stage1.5. But even if Stage 1 needs not to support the user interface or the filesystem interface, it is impossible to make Stage 1 less than 400 bytes, because GRUB should support both the CHS mode and the LBA mode (see Section D.8 [Low-level disk I/O], page 79).

The solution used by GRUB is that Stage 1 loads only the first sector of Stage 2 (or Stage 1.5) and Stage 2 itself loads the rest. The flow of Stage 1 is:

1. Initialize the system briefly.
2. Detect the geometry and the accessing mode of the *loading drive*.
3. Load the first sector of Stage 2.
4. Jump to the starting address of the Stage 2.

The flow of Stage 2 (and Stage 1.5) is:

1. Load the rest of itself to the real starting address, that is, the starting address plus 512 bytes. The block lists are stored in the last part of the first sector.
2. Long jump to the real starting address.

Note that Stage 2 (or Stage 1.5) does not probe the geometry or the accessing mode of the *loading drive*, since Stage 1 has already probed them.

D.6 How to probe I/O ports used by INT 13H

FIXME: I will write this chapter after implementing the new technique.

D.7 How to detect all installed RAM

FIXME: I doubt if Erich didn't write this chapter only himself wholly, so I will rewrite this chapter.

D.8 INT 13H disk I/O interrupts

FIXME: I'm not sure where some part of the original chapter is derived, so I will rewrite this chapter.

D.9 The structure of Master Boot Record

FIXME: Likewise.

D.10 The format of partition tables

FIXME: Probably the original chapter is derived from "How It Works", so I will rewrite this chapter.

D.11 Where and how you should send patches

When you write patches for GRUB, please send them to the mailing list bug-grub@gnu.org. Here is the list of items of which you should take care:

- Please make your patch as small as possible. Generally, it is not a good thing to make one big patch which changes many things. Instead, segregate features and produce many patches.
- Use as late code as possible, for the original code. The CVS repository always has the current version (see Appendix A [Obtaining and Building GRUB], page 69).
- Write ChangeLog entries. See section "Change Logs" in *GNU Coding Standards*, if you don't know how to write ChangeLog.
- Make patches in unified diff format. '`diff -urN`' is appropriate in most cases.
- Don't make patches reversely. Reverse patches are difficult to read and use.
- Be careful enough of the license term and the copyright. Because GRUB is under GNU General Public License, you may not steal code from software whose license is incompatible against GPL. And, if you copy code written by others, you must not ignore their copyrights. Feel free to ask GRUB maintainers, whenever you are not sure what you should do.
- If your patch is too large to send in e-mail, put it at somewhere we can see. Usually, you shouldn't send e-mail over 20K.

Index

S

T

U

V