Boot with GRUB

Wayne Marshall in Linux Journal

Apr 30, 2001

Abstract

Especially useful for multiboot, partitioned systems, GRUB offers flexibility and convenience for startup.

1 Introduction

GRUB: it's neither larva, fast food nor the loveliest of acronyms in the GNU herd of free software. Rather, GRUB is the GNU GRand Unified Bootloader. And, it is truly the greatest loader for booting Linux and practically any other OS — open source or otherwise — you may have scattered on your platters.

GRUB is independent of any particular operating system and may be thought of as a tiny, function-specific OS. The purpose of the GRUB kernel is to recognize filesystems and load boot images, and it provides both menu-driven and command-line interfaces to perform these functions. The command-line interface in particular is quite flexible and powerful, with command history and completion features familiar to users of the bash shell.

GRUB is in its element with the multiboot, multidisk systems typical of Linux and open-source adventurers who may simultaneously test or track several Linux distributions, the BSDs, GNU/Hurd, BeOS and perhaps that vestigial partition for Mr. Bill. Even if you stick with LILO as your system's primary boot loader, it's smart to keep a GRUB boot floppy handy as the best and fastest way to get your system back if you otherwise cream your master boot record (MBR). If you have done any number of multiboot installations, you know exactly what I'm talking about. Should you need any more reasons for considering GRUB, check out the sidebar, "Why GRUB". Let's get started!

"Why GRUB" is in Section 9

2 Installation

Installation of GRUB is a two-step process. The first step is to install or build GRUB in a host OS environment, and for this we will, of course, use Linux. The second step is to install and configure GRUB as the boot loader for your system.

The first step is the usual: download the source archive, untar it, configure and make install. Assuming you have found a source mirror (see www.gnu.org/software/grub/grub.html)

and downloaded the source distribution into a suitable working directory, continue with:

```
tar -xzvf grub-0.5.96.1.tar.gz
cd grub-0.5.96.1
./configure
make
make install
```

This should create the executables: `grub`, `grub-install` and `mbchk`; install support files in `/usr/local/share/grub/i386-pc/`, and install the GNU information manual and man pages.

For the second step of installation, we will first build and work with a GRUB boot floppy. This way we can use GRUB to learn about its features while testing various configurations for our particular system. After getting comfortable with the GRUB setup on floppy, we will then install it onto the MBR of the system's first hard disk. Even if you decide not to install GRUB on your hard disk right away, no harm done: you will now have your own GRUB boot floppy available to rescue systems with trashed boot loaders.

3 Preparing a GRUB floppy

GRUB recognizes a number of different filesytem types, including Linux `ext2fs`, `Reiser`, `MINIX`, BSD's `ffs`, as well as `FAT`, so it is possible to make a GRUB boot floppy with any of these filesystems. We will stick to `FAT` for this example, however, because it is the lowest common denominator, and most OSes have tools for mounting and reading/writing files on `FAT` floppies. That way, we will always be able to get to its menu configuration file if we need to.

Scrounge around in your junk drawer for some unused floppy (a new one would be even better), and give it a fresh format and `FAT` filesystem:

```
fdformat /dev/fd0
mkfs -t msdos /dev/fd0
```

We are going to put some files on this disk, so go ahead and mount to your usual floppy mount point (here I use `/floppy`):

```
mount -t msdos /dev/fd0 /floppy
```

Now install the directories and files GRUB will need:

```
mkdir -p /floppy/boot/grub
cp /usr/local/share/grub/i386-pc/stage* /floppy/boot/grub
```

The floppy can then be unmounted, **umount /floppy**, but leave it in the drive. The GRUB floppy is prepared and ready for the final installation, which is to install the GRUB boot loader in the MBR of the floppy itself. For that,

Figure 1: GRUB in command-line mode. Note the on-line help

we will use the grub executable we have built with our Linux installation. Start the executable at the Linux command prompt: `grub`.

This brings up an emulator of GRUB's command shell environment, which looks like Figure 1. We will discuss the features of this shell in more detail a little further on. For now, enter the following series of commands at the grub prompt:

```
grub> root (fd0)
grub> setup (fd0)
grub> quit
```

And that's it! This sequence of commands completes the installation of GRUB on the floppy disk. It is now bootable and will allow us to boot any other OS on our system.

4   Demonstrating GRUB

To see how GRUB may be used to boot a multitude of different operating systems, consider this example setup:

First Hard Disk (SCSI, Linux `/dev/sda`): 1st primary partition: Win98 2nd primary partition: Linux-Slackware 3rd primary partition: Linux-Debian 4th primary partition: Linux SwapSecond Hard Disk (SCSI, Linux `/dev/sdb`) 1st primary partition: FreeBSD 2nd primary partition: OpenBSD 3rd primary partition: BeOS

Note that although GRUB and Linux are capable of dealing with installations in extended partitions, here we show a preference for using primary parti-

tions whenever possible. Filesystems in primary partitions are often mountable by other operating systems, whereas cross-OS mounting filesystems in extended partitions is often not supported.

This system has two hard disks with six different operating systems using seven partitions. As you probably know, each OS has its own nomenclature for naming devices and partitions. For example, the Slackware installation would be known to Linux as `/dev/sda2` (with swap on `/dev/sda4`), while FreeBSD would recognize its filesystem on `/dev/da1s1a`. Alternatively, if the system were configured with IDE hard disks, Slackware would be on `/dev/hda2`, and FreeBSD would refer to its root directory on `/dev/ad1s1a`. You get large helpings of this alphabet soup whenever maintaining any multiboot setup.

Since GRUB also needs to be capable of loading any of these systems, it has its own OS-neutral naming conventions for referring to devices. Hard disks are all `hd`, floppy disks are `fd`, device numbering starts from zero, partition numbering starts from zero and complete device names are enclosed in parentheses.

With these naming rules, the floppy disk is `(fd0)`, the Win98 partition is `(hd0,0)`, and GRUB recognizes the Slackware and Debian partitions respectively as `(hd0,1)` for slackware and `(hd0,2)` for debian.

The BSDs further subdivide their own partitions (or "slices" in BSD terms), and GRUB would refer to the root mount for the FreeBSD system on `(hd1,0,a)`.

Okay, ready to give GRUB a taste? Slide the GRUB floppy in the drive and reboot your system (with your system's BIOS configured to boot from `A:` drive). You should see GRUB's terse boot messages and then find yourself in the GRUB command-line environment as shown in 1.

To start, let's boot Slackware. Enter the following commands at the grub prompt:

```
grub> root (hd0,1)
grub> kernel /vmlinuz root=/dev/sda2 ro vga=791
grub> boot
```

Badda-bing, badda-boom, that postage-stamp-sized Tux appears in the upper-left corner of your screen (yes, Slackware is configured to use the framebuffer device), and Linux bootstraps its jolly way into glorious being.

Another example. Reboot the system again with the GRUB floppy, and enter the following commands at the grub prompt:

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

Now your screen turns into a vague blue cloud, and you think you have made some horrible mistake. Then you realize it's only Windows and you remind yourself to expunge this partition one day soon.

Let's take a closer look at these examples. In the Slackware boot, we first used the GRUB root command to specify the device for GRUB to access. If the

device has a filesystem recognized by GRUB (that is, one of `ext2fs`, `reiser`, `ffs`, etc.), it attempts to mount it and get its partition information, then reports its success following the command. Thus, you would see the following command/response dialog on your screen:

```
grub> root (hd0,1)
Filesystem type is ext2fs, partition type 0x83
```

Next, we used the GRUB kernel command to specify the boot image for GRUB to load. The argument to the kernel command is the filename of the boot image relative to the device specified by the root command above. The kernel image file can also be specified in explicit `(device)/filename` terms as follows:

```
grub> kernel (hd0,1)/vmlinuz
```

The kernel command gives you great flexibility for specifying the boot image you wish to load. For example, if we saved a previous version of a kernel to the file `/vmlinuz.old`, we could specify it with this command (which shows GRUB's response):

```
grub> kernel /vmlinuz.old root=/dev/sda2 ro vga=ask
[Linux-bzImage, setup=0xe00, size=0xfad30]
```

The arguments following the name of the boot image are passed to the target kernel and aren't related to GRUB. For Linux, these kernel arguments are pretty much what you would specify them to be in `lilo.conf`. In our example, we tell the kernel what device to mount for the root partition (`root=/dev/sda2 ro`), using the device nomenclature expected by Linux. Note here that we also use the `ro` flag to mount the root filesystem read-only initially while it performs its filesystem check. The other kernel argument in our example simply demonstrates setting another kernel variable (`vga=791`) to use a particular vga mode for the framebuffer display.

Finally, the last command is **`grub> boot`**. The kernel image specified is now loaded and sent rolling down the royal road to bootdom.

The second example, using Win98, demonstrates the use of GRUB's chainloading mechanism. This method of booting loads the target OS's own bootchain-loader rather than a kernel image of the OS. In this instance, we specified:

```
grub> rootnoverify (hd0,0)
grub> chainloader +1
```

First, the **`rootnoverify`** command is for OS filesystems not specifically recognized by GRUB, so that GRUB will not try to mount the partition. Next, the **`chainloader`** command will use the first sector of the partition of device (`hd0,0`) and attempt to boot whatever it finds there. This is a common means of booting OSes that install their own boot loaders in the first sector of the

partition where they are installed (this is sometimes called the partition boot sector or PBR).

Finally, the `makeactive` command sets the active flag in the partition table for the device specified by the root command, as some operating systems, like Win98, require.

The GRUB command line is easy and fun, and you should boot the different OSes on your system a few times to get the hang of it. While you are testing, be sure to keep any notes specific to getting your particular kernels successfully loaded. This information will be useful later when you configure the menu system of GRUB to perform these command-line steps automatically.

But before we leave the command line, here are a few more GRUB commands to look at.

The help command will display a list of the 40 or so commands available in GRUB. Typing the name a particular command after help will produce on-line help for that particular command. So **grub> help kernel** will tell you all about using the kernel command.

The cat command can be used to view the contents of a file. For example, **grub> cat (hd0,2)/etc/fstab** will show the contents of the **/etc/fstab** file in the Debian installation. This is a very handy way of pulling out system configuration information if your normal boot loader gets whacked. Note also as you are using the GRUB command line that, like bash, up and down arrows will scroll through command history, and a tab will complete the name of a GRUB command or filename.

Finally, you can call up a specific menu interface with the configfile command as in:

```
grub> configfile (fd0)/boot/grub/menu.lst
```

This will switch GRUB into its menu mode with an interface defined by the file, menu.lst. We haven't created that file yet, but — look out, segue coming! — that's exactly what we will do next.

5 Menu Configuration

Using the GRUB command line is cool, but after a few thousand system starts, you will probably get a little tired of entering the same commands at the GRUB prompt and long for something a little more automated. Good news from the GRUB gang: you get a fully configurable menu interface at no extra charge! The GRUB boot menu gives you point-and-shoot boot selection, unattended default boot after a configurable timeout, any number of fallback boots if previous boots fail, toggle between command-line and menu modes, and interactive editing of menu selections and password protection. These features give GRUB an ease of use to match its tremendous functionality.

When GRUB boots, it automatically looks for the **/boot/grub/menu.lst** file on its boot device (the last three letters are "ELL ess tee" and not "one
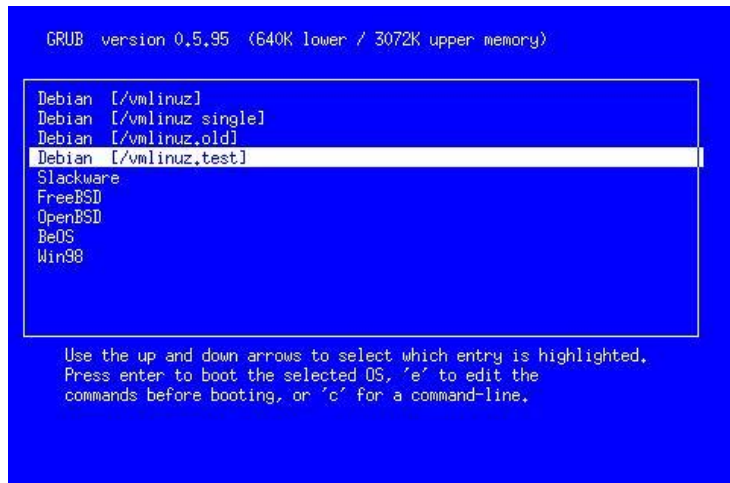
Figure 2: GRUB's Boot Menu Interface

ess tee"). If the file is found, GRUB automatically enters its menu mode and presents the user with a stunning interface, as shown in 2.

Listing 1 [found at LJ's web site] shows the configuration file responsible for this demonstration menu. As you can see, it is a simple text file typical of many UNIX configuration files, where lines starting with hashes (#) and blank lines are ignored.

The first set of commands sets general configuration variables. The `timeout` command sets the time in seconds to wait for the user to make a selection before proceeding automatically to the default boot. The default command sets which of the following boot stanzas GRUB should attempt to boot automatically. Boot stanzas are numbered implicitly, starting from zero, according to their order of appearance in the configuration file. This order is also how they will be listed in the menu.

The fallback command specifies which of the boot stanzas to load if the default should fail. It is possible to set more than one fallback, as is shown in the example.

The color command lets you breathe a bit of life into the text-mode menu screen. The syntax for the color command is

```
color  foreground/background  [ hilite-fg/hilite-bg ]
```

where each of the `foreground/background` colors is specified with a name from the set of `black`, `blue`, `green`, `cyan`, `red`, `magenta`, `brown` and `light-gray`; `dark-gray`, `light-blue`, `light-green`, `light-cyan`, `light-cyan`, `light-red`, `light-magenta`, `yellow` and `white`. Among these colors, only the first eight are used for the background. The optional highlight color pair, if specified, will be used to show the current menu selection. When not specified, GRUB will use the inverse of the normal colors.

7

The rest of the configuration file consists of the boot stanzas for our demonstration system. The title command marks the beginning of a new boot stanza, and its argument is the label that will be displayed for its entry in the menu, from the first non-white-space character to the end of the line. The remaining commands in each stanza are identical to those used when working from the GRUB command line. The exception here is that we no longer need to give a boot command; GRUB does this job without asking.

This example configuration gives only a sample of the many flexible uses of the GRUB boot loader. Besides multiple OSes, you can use GRUB to set up menu selections for test kernels, rescue kernels, different kernel options and so on.

All in all, the GRUB configuration file will be very similar to your `/etc/lilo.conf`. And after working with the GRUB command line and these examples, it should be a simple matter of firing up your favorite text editor and creating a menu configuration file suitable for your own system and preferences. Don't worry if it's not perfect the first time; you will see that you can make changes interactively, and the GRUB command line is always available as a fallback.

Once you've got your configuration file, mount your GRUB floppy again, and copy the file (say it has been saved as `mygrub.conf`) into the magic location:

```
cp  mygrub.conf /floppy/boot/grub/menu.lst
```

Now when you boot with your GRUB floppy — presto! — you will be greeted with a lovely boot menu like the one in 2. If you like, just stare at it for the few seconds it needs to count down from the timeout setting, and then it will automatically boot into your default OS. Or, use the arrow keys to highlight the OS you want to load and press return. Or, type `c` to go to the now-familiar GRUB command prompt. From the command prompt, press `ESC` to go back to the boot menu again.

It is also possible to edit the entries displayed in the menu. Typing `e` will open a simple `vi`-like editor interface for the highlighted entry. This allows you to adjust or add any settings to the configuration before booting. Any changes made here will then remain in effect for the duration of the GRUB session. To make permanent changes, you will later need to edit the configuration file on the boot disk, saving the changes with your text editor.

Play with your GRUB floppy configuration until you have it set up the way you like. After just a few system boots, you'll be slinging through GRUB like hashbrowns in a diner.

6   Hard Disk Installation

By this time you may be thinking, "Okay, GRUB has got it goin' on. But do I have to keep booting from this lame floppy all the time?" Of course not. Booting from floppy is for weenies.

The operation for installing GRUB on the master boot record of your hard disk is similar to the creation of a GRUB floppy. The one difference is that

our floppy has all the resources GRUB needs in one place. That is, the boot image, support and configuration files are all on the floppy device MBR and `/boot/grub/` directory. In a hard disk setup, you can choose where you want these resources to be.

For example, you could set up a `/boot/grub` directory on the first partition of your first hard disk and copy all GRUB's files into it as we did in our floppy setup. In our demonstration system, this would be the Win98 partition, and you may choose to do it that way if you want. But you can also set up the `/boot/grub` directory up in any `device/partition` on your machine with a filesystem supported by GRUB. In practice it is usually best to install this support directory in your most frequently used and/or most stable partition; that is, one that you aren't reinstalling all the time.

For this example, we will use the Slackware partition since this stays pretty stable, and I tend to do more tracking and installations in the Debian system. Once this decision is made, everything else is simple. First, boot into Slackware, create the `/boot/grub` directory and copy GRUB's files into it (these are all the files that the GRUB build installed in the `/usr/local/share/grub/i386-pc` directory). Make sure to put your handcrafted `menu.lst` configuration file in here, too.

Next, start GRUB, either with the grub executable you built in Linux or by rebooting with the GRUB floppy. If GRUB starts in menu mode, press c to go to command-line mode. Enter the following commands at the grub prompt:

```
grub> root (hd0,1)
grub> setup (hd0)
grub> quit
```

You're done. Your system is now fully GRUB'd, installed in the MBR of your hard disk. Type **reboot** as root (or take the floppy out and jab the keyboard with the old three-prong) and watch just how fast GRUB comes up now!

A few words of explanation about these installation commands. The first, `root (hd0,1)`, tells GRUB to mount this device, in this case the partition with the Slackware installation. All files will now be relative to this device, so the GRUB installer will know to look for its support files in the `/boot/grub` directory we created in the Slackware partition.

The second command, `setup (hd0)`, is a simplified front end to GRUB's install command. Note in particular that we specify the device as `(hd0)` and not `(hd0,0)`. Device `(hd0)` results in GRUB's installation to the master boot record, which is what we want. If we had instead used `(hd0,0)`, GRUB would be installed to the boot sector of the first partition, rather than the MBR. The difference is crucial; your technical writer makes mistakes like this so you don't have to. While each partition can have a boot sector, your hard disk will have only one master boot record the BIOS loads every time you start your machine. Unless you are doing some kind of funky boot-chaining, like using LILO to boot GRUB, you will usually want to install GRUB in the master boot record.

When GRUB installs itself on a device, it first copies a small piece of itself to the MBR, which it calls stage1. Then it follows stage1 with just enough

information about where to find the rest of GRUB. In our example, GRUB will put stage1 in the MBR, followed by a blocklist that points to the Slackware partition. GRUB will then find the rest of what it needs (its stage2 files) in the `/boot/grub` directory.

To check this setup, just edit the menu configuration file in Slackware's `/boot/grub/menu.lst` at any time. Any changes will be reflected in the next boot.

7 Error Recovery

If you should foul up the hard disk installation somehow or want to uninstall GRUB from your system, here's what you need to know.

First, if you ever want to clean your MBR from whatever is installed there, the canonical method is to use the `fdisk` program from an MS-DOS boot floppy:

```
A:> FDISK /MBR
```

Of course, this isn't necessary if you just want to go back to LILO as your system's boot manager. In that case, simply make sure your `/etc/lilo.conf` file has a line that reads **boot=/dev/hda**. Then, when the rest of the `lilo.conf` file is the way you want, just rerun LILO. This will put LILO back on the MBR of your system.

If you install GRUB in the boot sector of a partition, instead of the MBR (such as specifying **setup (hd0,0)** instead of **setup (hd0)**), you may need to reinstall that OS's boot loader. In the case of DOS/Windows, this means running the sys command from your DOS/Windows boot floppy: **A:> SYS C:**.

If, this is a Linux partition, it is again effective to rerun LILO, where `/etc/lilo.conf` has a line in the boot stanza that reads **root=/dev/hda1**.

In general, most OSes will have a way to reinstall their partition's boot sector without doing a full reinstallation from scratch. (For FreeBSD, see `boot0cfg(8)`; for OpenBSD, see `installboot(8)`.)

In practice, especially if you followed through on the GRUB floppy examples, you should find that GRUB itself is one of the best rescue and system recovery tools in your toolkit. For example, if you have ever made a screwup in your `lilo.conf` file, you know you can be in for some major pain if your system won't boot. With GRUB, you always have a miniature, self-contained operating system that can recognize and mount different filesystems, find files, locate kernels and boot them, bringing your system back to life so you can work on it. At times like these, GRUB can save your bacon.

8 Conclusions

As is typical of GNU software, GRUB is rich with capabilities beyond what are described here. Some of these include:

- Remapping disks and partition hiding, so you can even run multiple versions of DOS/Windows, on other than the first hard disk.

- Network booting with BOOTP and DHCP protocols, to support multiboot schemes across a network and diskless operation.

- Keyboard remapping, disk geometry access, memory reading, I/O port and processor probes, password protection, decompression support, etc.

See the GNU information manual for more information on these topics. GRUB is under active development, and even more features are planned for future releases.

In this brave GNU world, with vast acreage of cheap hard disk and a glut of great free OSes available, you really need a flexible and user-friendly boot loader to manage them all. Grab GRUB and give it a go.

9   Why GRUB?

Quite a few boot managers are available. Of course, Linux uses LILO, FreeBSD installs BootEasy, and you can also use OS-BS, payware like System Commander and other boot managers that ship with commercial operating systems like OS/2 and NT. So why GRUB?

OS-neutral: GRUB is not tied to any particular platform, and you don't need to run any special operating system to install and configure it. With LILO, for example, you need to be able to get into a Linux system for setup and maintenance, and remember to run the LILO binary after changing `/etc/lilo.conf`. GRUB can be set up and maintained from a number of systems, and any changes to its configuration file are immediately available.

Interactive: GRUB offers an interactive and configurable menu interface, and a command-line mode is always available. The command-line features are invaluable for getting into an OS whose boot configuration got creamed. Even boot settings within menu items can be interactively adjusted as necessary from within a GRUB session.

Powerful: If your BIOS handles LBA mode, GRUB can boot kernels beyond the 1024th cylinder, making kernel installations possible in partitions beyond 8GB. GRUB can do tricks with disk mapping and partition hiding, so you can run most OSes from whatever disk you want. GRUB also has network boot capabilities, enabling BOOTP and DHCP from remote servers.

GRUB is actively maintained and under continuing development. Its acronym expresses the goals of its authors to become the unified bootloader, so that every new OS project need not invent its own boot system. Although GRUB is primarily available for i386 systems, ports are underway and in various stages of completion for other architectures as well. By learning to use GRUB, you can be confident in having a single boot manager to handle any OS you may ever choose to install.

10   Author

Wayne Marshall (guinix@yahoo.com) is a UNIX programmer and technical consultant currently living in Guinea, West Africa. When not grubbing about with computers, he enjoys taking the pirogue for day trips to the local islands near Conakry with his wife, Paula.

A   Listing 1

Listing 1. `/boot/grub/menu.lst`

```
# /boot/grub/menu.lst
# grub boot menu configuration

# general configuration:
timeout  10
default   0
fallback  2
fallback  4
color     light-gray/blue  red/light-gray

# boot stanzas follow
# each is implicitly numbered from 0
# in the order of appearance below

# (0) Debian (default boot):
title  Debian  [/vmlinuz]
root   (hd0,2)
kernel /vmlinuz root=/dev/sda3 ro

# (1) Debian - runlevel 1
title  Debian  [/vmlinuz single]
root   (hd0,2)
kernel /vmlinuz single root=/dev/sda3 ro

# (2) Debian-old (first fallback)
title  Debian  [/vmlinuz.old]
root   (hd0,2)
kernel /vmlinuz.old root=/dev/sda3 ro

# (3) Debian-kernel testing
title  Debian  [/vmlinuz.test]
root   (hd0,2)
kernel /vmlinuz.test root=/dev/sda3 ro

# (4) Slackware (second fallback):
title  Slackware
root   (hd0,1)
kernel /vmlinuz root=/dev/sdb2 ro vga=791 mem=256M

# (5) FreeBSD:
title  FreeBSD
root   (hd1,0,a)
```

```
kernel /boot/loader

# (6) OpenBSD:
title  OpenBSD
# You would like to use:
##root    (hd1,1,a)
##kernel --type=openbsd /bsd
# But openbsd passes bios & kernel parameters with
# its own bootloader, the first stage of which is
# installed in the partition boot record, and in
# turn calls /boot, which in turn loads kernel /bsd.
# So just use the GRUB chainloader instead:
root (hd1,1)
makeactive
chainloader +1

# (7) BeOS:
title  BeOS
rootnoverify  (hd1,2)
makeactive
chainloader +1

# (8) WinDoze
title Win98
root  (hd0,0)
makeactive
chainloader +1

# end file menu.lst
```