Getting to know GRUB

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Tutorial tips

Should I take this tutorial?

This tutorial shows you how to install and use GRUB, the Grand Unified Boot Loader. Like LILO, GRUB allows you to boot your Linux system, taking care of loading and booting the kernel. Unlike LILO, GRUB is very feature-rich, much easier to use, much more reliable and flexible, and just plain neat-o.

If you're already somewhat familiar with LILO and have a basic working knowledge of disk partitions, you know everything you need to take this tutorial. By taking this tutorial and installing GRUB, you'll improve the reliability and availability of your Linux system.

If you want just a taste of GRUB, you can take the first half of this tutorial and make a GRUB boot floppy, and then practice using it to boot your system. By doing so, you'll learn how to use GRUB to boot your system in case of an emergency.

If, however, you want more in-depth experience with GRUB, you can complete the entire tutorial, which will show you how to set up GRUB as your default boot loader.

Navigation

Navigating through the tutorial is easy:

* Use the Next and Previous buttons to move forward and backward through the tutorial.
* Use the Main menu button to return to the tutorial menu.
* If you'd like to tell us what you think, use the Feedback button.

Getting help

For technical questions about the content of this tutorial, contact the author, Daniel Robbins, at *drobbins@gentoo.org* . Bugs, suggestions, and questions directly related to GRUB can be sent to the GNU GRUB mailing list, at *bug-grub@gnu.org* .

Residing in Albuquerque, New Mexico, Daniel Robbins is the President/CEO of *Gentoo Technologies, Inc.* , the creator of **Gentoo Linux**, an advanced Linux for the PC, and the **Portage** system, a next-generation ports system for Linux. He has also served as a contributing author for the Macmillan books *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming language as well as a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at **SONY Electronic Publishing/Psygnosis**. Daniel enjoys spending time with his wife, Mary, and his new baby daughter, Hadassah.

Section 2. GRUB basics

What is GRUB?

GRUB is a boot loader -- it's responsible for loading your kernel and booting your Linux system. GRUB can also boot other operating systems, such as FreeBSD, NetBSD, OpenBSD, GNU HURD, and DOS, as well as Windows 95, 98, NT, and 2000. While booting an operating system might seem like a rather mundane and trivial task, it's actually quite important. If the boot loader doesn't do a good job or isn't very resilient, you could get locked out of your system, unable to boot your computer. Also, a good boot loader will give you flexibility, allowing you to install multiple operating systems on your computer without having to deal with unnecessary complications.

GRUB is good

Fortunately for us, GRUB is a great boot loader. It has tons of features to make the booting process very reliable. For example, it can read a Linux kernel directly from a FAT, minix, FFS, ext2, or ReiserFS partition. This means that it can find your kernel no matter what. In addition, GRUB has a special interactive console mode that allows you to hand-load a kernel and select a boot partition. This feature is invaluable: just in case your GRUB menus are configured improperly, you can still get your system to boot. Oh, yes -- GRUB has a nice color-enabled boot menu as well. And we're just scratching the surface.

Why GRUB?

You may be wondering why the world even needs GRUB -- after all, the Linux world has had the LILO boot loader for a very long time, and it has allowed millions of Linux users to boot their systems over the years. Yes, it's true, LILO does work. However, LILO is pretty "high-maintenance" and inflexible. Rather than spend a lot of time describing why GRUB is so much better, I'll show you how to create your own GRUB boot disk and how to use it to boot your system. After that, I'll explain some of GRUB's cool technical details and guide you through the process of installing GRUB to your MBR (master boot record) so that it can be your default boot loader.

If you're a little gun-shy, you have no reason to fear. You can follow the first half of this tutorial, create a GRUB boot floppy and play around with GRUB without messing with your existing boot loader. Or, you can get familiar with GRUB in its safe, "hosted" mode. So, without further ado, let's dive in.

Section 3. Go GRUB, go!

Downloading GRUB

To start exploring the wonders of GRUB, you first need to download, compile, and install it. Again, don't freak out -- your boot records will not be modified at all -- we're just going to compile and install GRUB like any other program, at which point we'll be able to create a GRUB boot disk. Again, don't worry; I'll let you know before we start doing anything that will modify your boot process.

Now, to start. Head over to *ftp://alpha.gnu.org/gnu/grub/* and download the most recent version of the GRUB tarball you can find. Right now, as I'm writing the tutorial, the most recent tarball is grub-0.5.96.1.tar.gz. After you've snagged the most recent version, it's time to install it.

Install time

Here are the commands you'll need to type to get GRUB installed from its tarball. I'll be compiling the sources in /tmp and installing everything to the /usr tree on your hard drive. As root, type the following:

```
# cd /tmp
# tar xzvf /path/to/archive/here/grub-0.5.96.1.tar.gz
# cd grub-0.5.96.1
# ./configure --prefix=/usr
# make
# make install
```

Now GRUB is installed and we're ready to start playing with it.

Making the bootdisk

To make the bootdisk, we'll perform a few simple steps. First, we'll create an ext2 filesystem on a new floppy disk. Then, we'll mount it and copy some GRUB files to it, and finally we'll run the "grub" program that'll take care of setting up the floppy's boot sector. Ready?

Making the bootdisk, part 2

OK, put a blank disk in your 1.44MB floppy drive, and type:

```
# mke2fs /dev/fd0
```

After the ext2 filesystem is created, you'll need to mount it somewhere:

```
# mount /dev/fd0 /mnt/floppy
```

Now we need to create some directories and copy some critical files (installed when we installed GRUB, earlier) to the floppy:

```
# mkdir /mnt/floppy/boot
 # mkdir /mnt/floppy/boot/grub
 # cp /usr/share/grub/i386-pc/stage1 /mnt/floppy/boot/grub
 # cp /usr/share/grub/i386-pc/stage2 /mnt/floppy/boot/grub
```

Only one more step and we'll have a working bootdisk.

Making the bootdisk, part 3

When you unpacked, compiled, and installed the GRUB source tarball, a program called grub was placed in /usr/sbin. This program is very interesting and noteworthy, because it is in fact a semi-functional version of the GRUB boot loader. Yes, even though Linux is already up and running, you can run GRUB and perform certain tasks using the exact same interface you'll see if you use the GRUB bootdisk or install GRUB to your hard drive's MBR.

It's an intriguing design strategy, and now it's time to use our hosted version of GRUB to set up the boot sector of our bootdisk. As root, type "grub". A GRUB console will start up and look something like this:

```
     GRUB  version 0.5.96.1  (640K lower / 3072K upper memory)

  [ Minimal BASH-like line editing is supported.  For the first word, TAB
    lists possible command completions.  Anywhere else TAB lists the possible
    completions of a device/filename. ]

 grub>
```

Welcome to the GRUB console. Now, for the commands.

Making the bootdisk, part 4

At the grub> prompt, type:

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

Now your bootdisk will be complete. Before we move on, let's look at the commands you just typed. The first "root" command told GRUB where to look for its auxiliary files, stage1 and stage2. By default, GRUB looks in the /boot/grub directory on the partition or disk you specify. We copied these files to the right location just a few minutes ago when the bootdisk was mounted. Next, we typed the setup command, which told GRUB to install the bootloader on the boot record of our floppy; we'll look at this in more detail later. And, then we quit. Now that we have our bootdisk, it's time to start playing around with GRUB.

Section 4. Using GRUB

Getting ready

Before using GRUB to boot your system, here's the information you'll need to know. First, make sure you know which partition holds your Linux kernel, and the partition name of your root filesystem. Then, make sure you look at your existing LILO configuration for any arguments that need to be passed to the kernel, things like "mem=128M". Once you've acquired all this information, we're ready to begin.

Starting GRUB

To start GRUB, you'll need to shut down your system and boot off of the boot disk. If for some reason you can't right now (like if you're testing out GRUB on the departmental server during working hours), then just type "grub" at the prompt and follow along. Everything will work identically, except that you won't be able to boot anything (since Linux is already running!).

First contact

When the bootdisk loads, you'll see a message at the top of the screen letting you know that stage one and two are loading. After a few seconds, you'll see a familiar screen that looks like this:

```
    GRUB  version 0.5.96.1  (640K lower / 3072K upper memory)

  [ Minimal BASH-like line editing is supported.  For the first word, TAB
    lists possible command completions.  Anywhere else TAB lists the possible
    completions of a device/filename. ]

 grub>
```

As you can see, things look exactly the same as when you ran GRUB in hosted mode under Linux -- except now we're going to use GRUB to boot Linux.

The "root"

Under Linux, when we talk about the "root" filesystem, we're normally referring to your main Linux partition. However, GRUB has its own definition of root partition. GRUB's root partition is the partition that holds your Linux kernel. This may or may not be your "official" root filesystem. For example, under Gentoo Linux, we have a separate, small partition dedicated to holding Linux kernels and boot information. We keep this partition unmounted most of the time so that it can't get messed up if the system accidentally crashes or reboots.

The "root", part 2

Now that we're in GRUB, we need to specify GRUB's root partition. When we enter the root partition, GRUB will mount this partition read-only so that we can load a Linux kernel from it. One of the really cool things about GRUB is that it can read FAT, FFS, minix, ext2, and ReiserFS partitions natively, as we'll see in a moment. But now, let's type in that root partition. Type this at the prompt, but don't hit Enter just yet:

```
grub> root (
```

Now, hit the tab key once. If you have more than one hard drive in your system, GRUB will display a list of possible completions, starting with "hd0". If you have only one hard drive, GRUB will insert "hd0," for you. If you have more than one hard drive, go ahead and type its name in ("hd2") and follow it immediately with a comma, but don't hit Enter just yet. Your partially-completed root command should look something like this:

```
grub> root (hd0,
```

The "root", part 3

Now, go ahead and hit the tab key once again. GRUB will display a list of all the partitions on that particular drive, along with their filesystem type. On my system, I get the following list when I hit tab:

```
grub> root (hd0, (tab)
 Possible partitions are:
   Partition num: 0,  Filesystem type is ext2fs, partition type 0x83
   Partition num: 1,  Filesystem type unknown, partition type 0x82
   Partition num: 2,  Filesystem type unknown, partition type 0x7
   Partition num: 4,  Filesystem type is reiserfs, partition type 0x83
   Partition num: 5,  Filesystem type is reiserfs, partition type 0x83
```

As you can see, GRUB's interactive hard drive and partition name completion feature is pretty neat. Now, all we need to do is get a good understanding of GRUB's newfangled hard drive and partition naming syntax, and we'll be ready to roll.

GRUB naming convention

Up until now, you may be a bit confused because GRUB has a hard drive/partition naming convention that's different from the one Linux uses. Under Linux, the fifth partition on the first hard drive is called "hda5". GRUB refers to the same partition as "(hd0,4)". GRUB uses numbers for both the hard drive and the partition and starts counting both from zero. In addition, the drive and partition are separated by a comma and the entire expression is enclosed in parentheses. Now, looking back at the GRUB prompt, you can see that if you wanted to boot Linux drive hda5, you'd want to type in "root (hd0,4)". Now that you understand GRUB hard drive/partition naming, you may want to adjust your current root command line so that it's pointing to the partition that holds your Linux kernel. Complete your line as follows, and hit Enter:

```
grub> root (hd0,4) (hit enter)
 Filesystem type is reiserfs, partition type 0x83
```

Loading the kernel

Now that the root filesystem is mounted, it's time to load the kernel. At the GRUB prompt, type "kernel", then a space, then the path to the kernel, then a space, and then any kernel parameters, such as the root parameter (GRUB will insert an appropriate "mem=" parameter automatically). For my system, I type:

```
grub> kernel /boot/bz2.4 root=/dev/hda5
   [Linux-bzImage, setup=0x1200, size=0xe1a30]
```

Notice the "root=" kernel parameter, which is very important. It should point to the Linux partition that holds your root filesystem. You may want to write down the commands you've typed in so far, so that you'll have them handy when I show you how create a GRUB boot menu later in the tutorial.

Root, kernel, boot!

You've mounted the root filesystem and loaded the kernel. Now, it's time to boot. Simply type "boot", and the Linux boot process will begin.

Section 5. GRUB internals

Reinvestigating the boot floppy

If all went well, you were able to boot your current Linux distribution using the GRUB bootdisk. As you've seen, GRUB is quite a powerful boot loader, allowing you to dynamically configure it to boot as you like. In a bit, I'll show you how to set up a GRUB boot menu so that you can make your OS selection from a menu rather than having to type three lines of commands to boot Linux. But before we do that, now would be a good time to get a deeper understanding of how GRUB works behind the scenes. I'll explain how the bootdisk boot process worked so that you have a greater appreciation and understanding of GRUB.

A two-stage process

To set up the boot floppy, we did two things -- copied files to the /boot/grub directory on the floppy's ext2 filesystem, and ran GRUB's setup program. When we ran GRUB setup, GRUB installed the "stage 1" loader in the floppy's boot record. It also configured the stage 1 loader to load stage2 directly from the ext2 filesystem. Typically, GRUB does this by creating a list of blocks on the floppy that contain the stage2 data, so that stage1 doesn't need to know anything about the ext2 filesystem to load stage2.

However, in most cases, GRUB will install a stage1.5 loader in the boot record, immediately after stage1. This special stage1.5 allows stage2 to be loaded from the ext2 filesystem without using a primitive blocklist, but instead using the more-flexible standard path-based approach. This ability for GRUB to understand filesystem structures directly makes GRUB a lot more rugged than LILO. If you happened to defragment your bootdisk's filesystem, for instance, stage1 (thanks to the ext2 stage1.5) would be able to find stage2. This is something that LILO cannot do. Because LILO relies exclusively on map files, it needs to be rerun every time you update your kernel or move things around physically on disk, even if their path doesn't change.

Stages 1, 1.5, and 2

You may be wondering if GRUB would work if your bootdisk was created using the FAT instead of the ext2 filesystem. Yes, it would, because when we typed "setup (fd0)", GRUB would have installed the correct stage1.5 to match the root filesystem type. And even if there were no room for a stage1.5, GRUB could load stage2 by falling back to the more primitive block-list approach.

Search and rescue

Before we move on, here's one really handy tip relating to our boot floppy. Due to GRUB's interactive nature, it makes a great boot loader for a rescue floppy. However, you can make your bootdisk even more useful by copying a good kernel over to it. That way, if the kernel on your hard drive becomes corrupt or is accidentally deleted, you can fall back to your bootdisk kernel and still get your system up and running. To copy a spare kernel to your bootdisk, do the following:

```
# mount /dev/fd0 /mnt/floppy
# cp /path/to/bzImage /mnt/floppy/boot
# umount /dev/fd0
```

Now that the floppy contains the spare kernel, you can use it to boot your Linux distribution under GRUB as follows:

```
grub> root (fd0)
grub> kernel /boot/bzImage root=/dev/hda5 (change /dev/hda5 to the partition name
grub> boot
```

Hard drive boot

OK, now how does one install GRUB onto a hard drive? The process is almost identical to the bootdisk installation process. First, you need to decide which hard drive partition will end up being your root GRUB partition. On this partition, create a /boot/grub directory, and copy the stage1 and stage2 files from /usr/share/grub/i386-pc into this directory. You can perform this next part either by rebooting your system and using the bootdisk, or by using the hosted version of GRUB. In either case, start GRUB up, and specify your root partition with the root command. For example, if you copied the stage1 and stage2 files to the /boot/grub directory on hda5, you'd type "root (hd0,4)". Now, one more step.

Hard drive boot, continued

Next, decide where you'd like to install GRUB -- either your disk's MBR, or if you're using another "master" boot loader in conjunction with GRUB, the boot record of a specific partition. If you're installing to an MBR, you can specify the entire disk without a partition, as follows (for hda):

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

If you wanted to install GRUB to /dev/hda5's boot record, you'd type:

```
grub> setup (hd0,4)
```

Now, GRUB is installed. When you boot your system, you should immediately end up in GRUB's console mode (if you installed to the MBR). Now it's time to create a boot menu so you don't have to type those commands in every time you boot your system.

The boot menu

To create a menu, all you need to do is create a simple text file in /boot/grub called menu.lst. If you put it in the right place, it'll be sitting alongside the stage1 and stage2 files on your root GRUB drive. Here's a sample menu.lst file that you can use as a basis for your own:

```
default 0
timeout 30
color white/blue blue/green

title=Boot Linux
root (hd0,4)
kernel /boot/bzImage root=/dev/hda5

title=Boot Linux using initrd
root (hd0,5)
kernel /boot/bzImage root=/dev/loop0 init=/initdisk.gz
initrd /initdisk.gz

title=Windows NT
root (hd0,3)
chainloader +1
```

I'll explain the menu.lst format in the following panels.

Understanding the boot menu

The boot menu is easy to understand. In the first three lines, we set the default menu item (item number zero, the first one), set the timeout value (30 seconds), and select some nice colors for the entire menu.

In the next three lines, we configure a "Boot Linux" menu item. To create a menu item out of your series of manual boot commands, simply add a "title=" line as the first line, and remove your "boot" command from the last line (GRUB adds this automatically).

The next four lines show you how to use GRUB to boot with an initrd (initial root disk), if you are so inclined. Now, for the last three lines...

The chainloader

Again, here are the last three lines from the example menu.lst...

```
title=Windows NT
root (hd0,3)
chainloader +1
```

Here, I added an entry to boot Windows NT. To do this, GRUB uses what is called a "chainloader." The chainloader works by loading NT's own boot loader from the boot record on partition (hd0,3), and then booting it. That's why this technique is called chain loading -- it creates a chain from one boot loader to another. This chainloading technique can be used to boot any version of DOS or Windows.

Section 6. Wrapup

GRUB's resiliency

One of GRUB's greatest strengths is its robust design -- don't forget this as you continue to use it. It's not necessary to reinstall GRUB if you update your kernel or change its location on disk. Instead, just update your menu.lst file if needed, and all will be well.

There are only a couple of situations where you'll need to reinstall the GRUB boot loader to your boot record. First, if you change the partition type of your GRUB root partition (for example, from ext2 to ReiserFS), a reinstall will be needed. Or, if you update the stage1 and stage2 files in /boot/grub so that they're from a newer version of GRUB, you'll most likely need to reinstall the boot loader. Other than that, you're all set!

An excellent GRUB resource

There's a lot more to GRUB than what we covered here. For example, you can use GRUB to do network boots, boot BSD filesystems, and more. In addition, GRUB has many configuration and security commands that you may find useful. For a complete description of all GRUB functionality, be sure to check out GRUB's excellent GNU info documentation. Just type "info grub" at your bash prompt and read away.

I hope you have enjoyed this tutorial and will continue to enjoy the power and flexibility of GRUB, the Grand Unified Boot Loader!

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

For technical questions about the content of this tutorial, contact the author, Daniel Robbins, at *drobbins@gentoo.org* . Bugs, suggestions, and questions directly related to GRUB can be sent to the GNU GRUB mailing list, at *bug-grub@gnu.org* .

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial

generator. The Toot-O-Matic tool is a short Java program that uses XSLT stylesheets to convert the XML source into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML.