Last month's column introduced the powerful Grand Unified Boot Loader (GRUB), a utility that enables you to boot one of many operating systems when you start your computer. That column looked at basic GRUB configuration, including setting up GRUB on a floppy disk to boot Linux.

**By Roderick Smith**
Monday, December 15th, 2003

This month's column goes further, with a look at using GRUB to boot FreeBSD, DOS, and Windows. It also includes information on hiding and unhiding partitions, changing the apparent order of your physical disks, and installing GRUB on your hard disk as a permanent boot loader.

### Booting FreeBSD

FreeBSD boots much like Linux: the boot loader must load the kernel into memory, and the kernel takes it from there. However, FreeBSD presents its own unique challenges, and comes with its own boot loader. If your system multi-boots to FreeBSD, you should first understand some additional points about GRUB’s partition-naming system.

Typically, FreeBSD takes over a primary partition (what FreeBSD calls a *slice*) and breaks it into subpartitions (what FreeBSD calls a *partition*). This scheme is very similar to the logical partitions that can reside within an extended partition in the standard x86 partitioning scheme. GRUB numbers regular x86 logical partitions starting at 4, but what about FreeBSD’s partitions? To avoid confusion, GRUB gives them letters, starting with a. These letters correspond to the final character of a FreeBSD device filename for the partition.

For instance, *(hd1,a)* refers to the first partition in whatever slice holds the FreeBSD partitions on the second physical disk. If necessary, you can also specify a slice number between the drive number and FreeBSD partition number, as in *(hd1, 1,a)*, which refers to the first partition in the second slice on the second physical disk (what FreeBSD calls /dev/ad1s2a). In most cases, the shorter form suffices, but if GRUB has problems locating the FreeBSD slice, you can use the longer form.

GRUB provides three methods of booting FreeBSD. The first is to use a chain loader, as described in the next section for DOS or Windows. This method relies upon the presence of the standard FreeBSD boot loader in the FreeBSD partition’s boot sector. In most cases, this approach works perfectly well, but it’s somewhat circuitous.

Typically, FreeBSD takes over a primary partition (what FreeBSD calls a *slice*) and breaks it into subpartitions (what FreeBSD calls a *partition*). This scheme is very similar to the logical partitions that can reside within an extended partition in the standard x86 partitioning scheme. GRUB numbers regular x86 logical partitions starting at 4, but what about FreeBSD’s partitions? To avoid confusion, GRUB gives them letters, starting with a. These letters correspond to the final character of a FreeBSD device filename for the partition.

For instance, *(hd1,a)* refers to the first partition in whatever slice holds the FreeBSD partitions on the second physical disk. If necessary, you can also specify a slice number between the drive number and FreeBSD partition number, as in *(hd1, 1,a)*, which refers to the first partition in the second slice on the second physical disk (what FreeBSD calls /dev/ad1s2a). In most cases, the shorter form suffices, but if GRUB has problems locating the FreeBSD slice, you can use the longer form.

GRUB provides three methods of booting FreeBSD. The first is to use a chain loader, as described in the next section for DOS or Windows. This method relies upon the presence of the standard FreeBSD boot loader in the FreeBSD partition’s boot sector. In most cases, this approach works perfectly well, but it’s somewhat circuitous.

The second method of booting FreeBSD from GRUB also relies on the FreeBSD boot loader, but only part of it. Instead of chaining the boot process to the FreeBSD boot sector, this approach calls the FreeBSD boot loader program *(/boot/loader)* on the FreeBSD partition as if it were a kernel. (Remember that GRUB can read FreeBSD’s FFS filesystem, even if you compile your Linux kernel without that support.) The FreeBSD boot loader then loads its configuration file and boots the FreeBSD kernel. To implement this approach, you create a menu.lst or grub.conf entry like the following:
This configuration tells the system to read the `/boot/loader` file from `(hd1,a)` and treat it as a kernel. In fact, it's not a kernel, but is instead a boot loader that will in turn load the real FreeBSD kernel.

In theory, GRUB can load the FreeBSD kernel directly by pointing to the FreeBSD kernel on the `kernel` line in the configuration file, as in `kernel /boot/kernel/kernel to boot a FreeBSD 5.0 kernel.

Unfortunately, this support is unreliable at the moment; you're much better off using the FreeBSD boot loader as a helper to GRUB.

### Booting DOS or Windows

DOS and Windows have similar boot characteristics. These operating systems have simple boot loaders on their boot partitions' boot sectors. You can tell GRUB to pass control to these simple boot loaders using an entry like this:

```
title Windows
rootnoverify (hd0,1)
makeactive
chainloader +1
boot
```

This configuration has the same `title` and `boot` elements as any other, but three elements that tell GRUB how to boot the OS are unique:

* The `rootnoverify` option is similar to the `root` option, in that it tells GRUB to apply future operations to this partition. Unlike `root`, though, `rootnoverify` doesn't give GRUB access to files on the partition in question. This is important mainly for NTFS partitions, which GRUB can't mount. By using `rootnoverify`, you guarantee that GRUB won't choke on an NTFS partition.

* The `makeactive` command sets the active flag (aka the `bootable` flag) for the partition selected via `rootnoverify`. Depending upon your configuration, this step may not be necessary, since this flag may already be set. This option is more of a safety measure to ensure that you don't accidentally try to boot a partition from which this flag has been improperly removed.

* The `chainloader +1` command tells GRUB to pass control to the boot loader on the boot sector of the partition specified via the `rootnoverify` line. The +1 option to this command tells GRUB to use the first sector of the partition (that is, the boot sector) as the code to execute. In theory, you could point to any other sector on the disk as the boot loader, but this capability is of limited value.

The procedure that's used to boot DOS or Windows is also useful to boot many other OSs that provide boot loaders in their boot partitions' boot sectors. You can use this system to boot FreeBSD, OS/2, or BeOS, for instance.

### Hiding and Unhiding Partitions

Most operating systems use partition type codes (aka partition IDs) to determine which partitions they should attempt to mount. You can set these type codes from Linux's `fdisk` by using the `t` command.

Unfortunately, partition type codes don't always uniquely identify a specific filesystem, which can cause problems. For instance, type code `0x07` is used by both Windows NT/2000/XP's NTFS and by OS/2's HPFS; and `0x82` is used both for Linux swap partitions and by the x86 version of Solaris. If you use these unlucky combinations of operating systems, you may want to change the partition type codes of some partitions when you boot particular OS's. Another reason to do this is if you simply want to hide one operating system's data from another OS — say, to prevent Windows from messing with a DOS partition's files.

Hiding and unhiding partitions is accomplished with the `hide` and `unhide` commands in the GRUB configuration file. Both of these commands accept a partition identifier as a parameter. Hiding is done by adding `0x10` to the partition type code, while unhiding is done by subtracting `0x10` from the type code. (Only certain recognized codes are hidden or unhidden in this way, though. If hiding blindly added `0x10`, selecting the same OS multiple times would continue
adding 0x10 to its hidden partition's type codes, which would cause serious problems for operating systems that need to use that partition.) Typically, you'll use these commands in pairs, with reversed versions of the pair in another OS's definition.

For instance, you might use lines like these to boot DOS and Windows on a single system:

```plaintext
title DOS
  unhide (hd0,0)
  hide (hd0,1)
  rootnoverify (hd0,0)
  makeactive
  chainloader +1
  boot

title Windows
  unhide (hd0,1)
  hide (hd0,0)
  rootnoverify (hd0,1)
  makeactive
  chainloader +1
  boot
```

This definition is suitable for a system on which DOS is installed on the first primary partition (hd0,0) and Windows is installed on the second primary partition (hd0,1). Each OS's GRUB configuration unhides its own partition and hides the partition belonging to the other OS. Without this swapping of hidden partitions, the DOS partition will appear to be C: in both DOS and Windows. By hiding each OS's partition from the other OS, each believes it resides on C:. This characteristic greatly simplifies multi-boot configurations involving multiple DOS or Windows versions.

**Changing the Boot Disk**

Some operating systems, such as DOS and Windows, must boot from the first physical disk in the computer. (Windows can actually store most of its files on another disk, but the C: partition on the first physical disk must still hold certain critical boot files.) Although this requirement isn't a problem for a single-OS installation or even for most installations with two or three OSs, as the number of OSs rise, the difficulty increases. This is particularly true when you want to boot multiple versions of DOS or Windows, because you'll soon run out of primary partitions on the first physical disk. Some boot managers allow you to store multiple versions of DOS or Windows on a single C: drive, but GRUB doesn't support this option.

One way around this problem is to make the computer believe the second or subsequent physical disk is in fact the first physical disk. You can pull off this trick using the `map` option.

The `map` command accepts two GRUB drive identifiers as parameters: the `to-drive` and the `from-drive`. Typically, you'll use this parameter to swap the first two physical disks, so you'll use two `map` commands, one to map the first physical disk as the second physical disk and the second to map the second physical disk to the first physical disk.

For instance, the following `menu.lst` or `grub.conf` entry boots Windows stored on the second partition on the second physical disk:

```plaintext
title Windows
  map (hd0) (hd1)
  map (hd1) (hd0)
  rootnoverify (hd1,1)
  makeactive
  chainloader +1
  boot
```

Note that the `map` command doesn't remap GRUB's own drive identifiers; the `rootnoverify` command still refers to the second physical disk as (hd1,1), not (hd0,1). You'd use the
Of course, to use this approach, you must first install the OS on the second physical disk. One way to do this is to physically reconfigure the drives for the installation process. That is, change the jumpers on your second drive so that it becomes the first drive, and either remove the first drive entirely or change its jumpers so that it becomes the second drive. After installing, you must then reverse your changes and set up GRUB to boot from the second physical disk. Another way to handle this problem is to create a special GRUB menu item to boot your OS installation medium while swapping drives.

For instance, to boot from an installation floppy disk, you might use the following entry:

```
title Floppy boot with drive swapping
  map (hd0) (hd1)
  map (hd1) (hd0)
  rootnoverify (fd0)
  chainloader +1
  boot
```

When you select this option from the GRUB menu, it redirects the boot process to the floppy disk, but the first and second drives will be swapped, at least from the BIOS's point of view. This should be enough to install OSs that rely upon the BIOS for drive access.

One limitation of drive swapping as implemented by GRUB is that it only affects the BIOS's view of drives. If an OS uses non-BIOS access methods, as all modern operating systems do, that operating system won't be affected by the GRUB-specified drive mapping.

Nonetheless, this technique can be useful in starting the boot process. In particular, the early stages of booting Windows rely upon the BIOS, so the mapping options can be useful in overcoming the first-drive requirements of Windows. Later in the boot process, when the more advanced Windows drivers take over disk access, you must rely on other techniques to ensure that drive letters are mapped appropriately. For instance, you can use the `hide` option to hide primary partitions from the true first physical disk that might be inappropriately mapped as the C: drive in Windows. (Logical partitions from the first physical disk should be mapped with drive letters of D: or above.)

Some older operating systems, such as DOS, rely exclusively upon the BIOS for handling drive access. Thus, the GRUB `map` options alone can be a handy way to install these OSs.

For instance, suppose you install three versions of Windows on three primary partitions on the first physical disk, but you also want to install DOS. The GRUB `map` option alone will allow you to install DOS on the second physical disk.

**Permanent GRUB Installation**

After you've finished testing GRUB, you can install it on your hard disk in much the same way you installed it on a test floppy disk.

You should first choose a partition for GRUB's configuration files. The default location is `/boot/grub`, which is on either your root partition or your `/boot` partition, depending upon how your system is partitioned. This location works well for most Linux installations, but if you want to be able to modify your GRUB configuration from other OSs, you may want to move the GRUB files elsewhere. One common choice is a FAT partition, because most OSs can read and write FAT. If you move the GRUB files to a FAT partition, do so much as you moved them to a FAT floppy disk. Be sure to put all the files in a directory called `grub` or `boot/grub` on the root of the partition in question.

After copying the GRUB files, you can use the `grub` utility to install the boot loader in the disk's MBR. The process is just like installing to a floppy disk, but you must specify the hard disk rather than the floppy disk as the target.

To install GRUB in the MBR and use GRUB configuration files stored on a `/boot` partition on `/dev/hda6`, you'd type the following commands:

```
# grub
grub> root (hd0,5)
grub> setup (hd0)
grub> quit
```

Alternatively, you might want to have GRUB use files on the Linux `/dev/hda2` root partition.
(with no separate /boot partition) and use GRUB as a secondary boot loader on that partition. Such a configuration might be handy if you're using some other advanced boot loader on the MBR or if you don't want to expose the GRUB MBR boot loader to damage by non-Linux OS installers, which sometimes replace the MBR boot loader. To install GRUB in this way on /dev/hda2, you should type the following commands:

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

Do not install GRUB (via setup) in a bootable FAT partition's boot sector, though. If you store GRUB's configuration files on a FAT partition, install the boot loader code in the MBR or on a Linux partition. If you overwrite the standard DOS or Windows boot code with the GRUB boot loader, you won't be able to boot DOS or Windows from the partition; the attempt to do so will simply load GRUB again.

One shortcut to typing several commands into the grub utility is to use the grub-install utility. This command installs GRUB automatically, using the partition on which the GRUB configuration files reside as the GRUB root partition. You must specify the device to which you want to install GRUB by passing that name after the command name, as in grub-install /dev/hda to install GRUB in the MBR of the first disk.

Before rebooting, you may need to make changes to your menu.lst or grub.conf file. In particular, if you've been using a test floppy boot of GRUB, check for references to (fd0) in this file, such as in a splashimage or keytable command. Chances are you don't want to have GRUB loading critical files from the floppy disk when you boot from the hard disk, so change these references to point to the hard disk.

Roderick W. Smith is the author or co-author of eleven books, including Advanced Linux Networking and Linux Power Tools. He can be reached at rodsmith@rodsbooks.com.

Roderick W. Smith is the author or co-author of over a dozen books, including Advanced Linux Networking and Linux Power Tools. He can be reached at rodsmith@rodsbooks.com.