

# Learn Linux, 101: Install a boot manager

## Introducing GRUB, GRUB 2, and LILO

Ian Shields  
Linux Author  
Freelance

14 August 2015  
(First published 13 April 2010)

Learn how to choose and configure a boot manager for your Linux® system. You can use the material in this tutorial to study for the LPI 101 exam for Linux system administrator certification, or to learn for fun.

[View more content in this series](#)

### Overview

In this tutorial, learn to choose, install, and configure a boot manager for a Linux system. Learn to:

- Provide alternative boot locations and backup boot options
- Install and configure a boot loader such as GRUB Legacy
- Perform basic configuration changes for GRUB 2
- Interact with the boot loader

### Boot managers

A *boot manager* or *boot loader* is the intermediate piece of code that helps the hardware and firmware of your system load an operating system for you. This tutorial discusses the PC boot process and the three main boot loaders that are used in Linux: GRUB, GRUB 2, and LILO with MBR formatted disks. The original GRUB, now called GRUB-Legacy, is no longer in active development and has largely been replaced by the newer GRUB 2. Even commercial distributions such as Red Hat Enterprise Linux and SUSE Linux Enterprise Server switched to Grub 2 in 2014. Development of Lilo is scheduled to cease at the end of 2015.

#### About this series

This series of tutorials helps you learn Linux system administration tasks. You can also use the material in these tutorials to prepare for the [Linux Professional Institute's LPIC-1: Linux Server Professional Certification exams](#).

See "[Learn Linux, 101: A roadmap for LPIC-1](#)" for a description of and link to each tutorial in this series. The roadmap is in progress and reflects the version 4.0 objectives of the LPIC-1

exams as updated April 15th, 2015. As tutorials are completed, they will be added to the roadmap.

This tutorial helps you prepare for Objective 102.2 in Topic 102 of the Linux Server Professional (LPIC-1) exam 101. The objective has a weight of 2. LILO is no longer required for LPIC-1. It is included here so that you will know something about it.

This tutorial focuses on booting with a traditional BIOS and disks formatted with a Master Boot Record (MBR). It also covers some basic information on Extensible Unified Firmware Interface (UEFI) and its associated GUID Partition Table (GPT) and booting issues you might find with these, particularly if you need to boot both Windows® 8 and Linux on a single system.

## Prerequisites

To get the most from the tutorials in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered. You should also be familiar with hard drive layout as discussed in the tutorial "[Learn Linux 101: Hard disk layout](#)." Sometimes different versions of a program will format output differently, so your results might not always look exactly like the listings and figures shown here.

**Note:** The images in this tutorial are screen captures taken from early in the boot process. If you are reading this tutorial using a screen reader, you might find it advantageous to have the corresponding configuration files available for reference; download them from the [Download](#) section later in this tutorial.

## Boot process overview

Before I get into specific boot loaders, let's review how a traditional PC starts or boots. Code called BIOS (for *Basic Input Output Service*) is stored in non-volatile memory such as a ROM, EEPROM, or flash memory. When the PC is turned on or rebooted, this code is executed. Usually it performs a power-on self test (POST) to check the machine. Finally, it loads the first sector from the master boot record (MBR) on the boot drive.

As discussed in the tutorial "[Learn Linux 101: Hard disk layout](#)," the MBR also contains the partition table, so the amount of executable code in the MBR is less than 512 bytes, which is not very much code. Every disk, even a floppy, contains executable code in its MBR, even if the code is only enough to put out a message such as "Non-bootable disk in drive A:". This code that is loaded by BIOS from this first sector is called the first stage boot loader or the stage 1 boot loader.

The standard hard drive MBR used by MS DOS, PC DOS, and Windows operating systems checks the partition table to find a primary partition on the boot drive that is marked as active, loads the first sector from that partition, and passes control to the beginning of the loaded code. This new piece of code is also known as the partition boot record. The partition boot record is actually another stage 1 boot loader, but this one has just enough intelligence to load a set of blocks from the partition. The code in this new set of blocks is called the stage 2 boot loader. As used by MS-DOS and PC-DOS, the stage 2 loader proceeds directly to load the rest of operating system. This is how your operating system pulls itself up by its bootstraps until it is up and running.

This works fine for a system with a single operating system. What happens if you want multiple operating systems, say OS/2, Windows XP, and three different Linux distributions? You *can* use some program (such as the DOS FDISK program) to change the active partition and reboot. This is cumbersome. Furthermore, a disk can have only four primary partitions, and the standard MBR can have only one active primary partition; it cannot boot from a logical partition. But our hypothetical example cited five operating systems, each of which needs a partition. Oops!

The solution lies in using some special code that allows a user to choose which operating system to boot. Examples include:

### **Loadlin**

A DOS executable program that is invoked from a running DOS system to boot a Linux partition. This was popular when setting up a multiboot system was a complex and risky process.

### **OS/2 Boot Manager**

A program that is installed in a small dedicated partition. The partition is marked active, and the standard MBR boot process starts the OS/2 Boot Manager, which presents a menu where you can choose which operating system to boot.

### **A smart boot loader**

A program that can reside on an operating system partition and is invoked either by the partition boot record of an active partition or by the master boot record. Examples include:

- BootMagic, part of Norton PartitionMagic
- LILO, the Linux LOader
- GRUB, the GRand Unified Boot loader (now referred to as GRUB Legacy)
- GRUB 2, a newer boot loader that is now used in many common distributions
- Syslinux, a group of lightweight boot loaders for MS-DOS FAT filesystems (SYSLINUX), network booting (PXELINUX), bootable "El Torito" CD-ROMs (ISOLINUX), and Linux ext2/ext3/ext4 or btrfs filesystems (EXTLINUX)

Evidently, if you can pass control of the system to some program that has more than 512 bytes of code to accomplish its task, then it isn't too hard to allow booting from logical partitions, or booting from partitions that are not on the boot drive. All of these solutions allow these possibilities, either because they can load a boot record from an arbitrary partition, or because they have some understanding of what file or files to load to start the boot process.

## **Chain loading**

When a boot manager gets control, one thing that it can load is another boot manager. This is called chain loading, and it most frequently occurs when the boot manager that is located in the master boot record (MBR) loads the boot loader that is in a partition boot record. This is almost always done when a Linux boot loader is asked to boot a Windows or DOS partition, but it can also be done when the Linux boot loader for one Linux system (say Fedora) is configured to load the boot loader for another Linux system (say Ubuntu). For example, you might use GRUB in one partition to launch the GRUB boot loader in another partition's boot record to start the Linux system in that partition. This is not common, but it illustrates the possibilities.

## Linux boot loaders

This tutorial focuses on GRUB, GRUB 2, and LILO because these are the boot loaders included with most Linux distributions. LILO has been around for awhile. GRUB is newer. The original GRUB has now become GRUB Legacy, and GRUB 2 is being developed under the auspices of the Free Software Foundation (see [Resources](#) for details). This tutorial discusses GRUB first, and then GRUB 2, so that you have some idea of the major differences and how GRUB and GRUB 2 can coexist. For the rest of this tutorial, assume GRUB means GRUB Legacy, unless the context specifically implies GRUB 2. A new version of LILO called ELILO (which is designed for booting systems that use Intel's Extensible Firmware Interface, or EFI, rather than BIOS) is also available. See [Resources](#) for additional information about ELILO.

As noted earlier, GRUB Legacy is no longer in active development and LILO development is scheduled to end in late 2015. Most Linux systems in 2015 are shipping with GRUB 2 as the default, or sometimes only boot loader.

The installation process for your distribution might give you a choice of which boot loader to set up. GRUB, GRUB 2, and LILO all work with most modern disks, although some distributions, notably Fedora, no longer ship LILO or GRUB Legacy. Remember that disk technology has advanced rapidly, so you should always make sure that your chosen boot loader, as well as your chosen Linux distribution (or other operating system), as well as your system BIOS, will work with your shiny new disk. Failure to do so can result in loss of data. Likewise, if you're adding a new distribution to an existing system, you might need to make sure you have the latest LILO, GRUB, or GRUB 2 in your MBR. You also need a fairly recent version of your boot loader if you plan to boot from an LVM or RAID disk.

With the stage 2 loaders used in LILO and GRUB, you can choose from several operating systems or versions to load. However, LILO and GRUB differ significantly in that a change to the system requires you to use a command to recreate the LILO boot setup whenever you upgrade a kernel or make certain other changes to your system, while GRUB can accomplish this through a configuration text file that you can edit. GRUB 2 also requires a rebuild from a configuration file that is normally stored in /etc.

## GRUB

GRUB, or the GRand Unified Boot loader, was for a long time one of the most common Linux boot loaders. You can install GRUB into the MBR of your bootable hard drive or into the partition boot record of a partition. You can also install it on removable devices such as floppy disks, CDs, or USB keys. It is a good idea to practice on a floppy disk or USB key if you are not already familiar with GRUB. The examples in this tutorial show you how.

**Note:** Most GRUB examples in this tutorial use CentOS 6.

During Linux installation, you often specify your choice of boot manager. If you choose LILO, then you might not have GRUB installed. If you do not have GRUB installed and it is available for your distribution, then you need to install the package for it. This tutorial assumes that you already have

the GRUB package installed. See the [series roadmap](#) for the tutorials on package management if you need help with this.

GRUB (Legacy) has a configuration file that is usually stored in `/boot/grub/grub.conf`. If your file system supports symbolic links, as most Linux file systems do, you probably have `/boot/grub/menu.lst` as a symbolic link to `/boot/grub/grub.conf`.

The `grub` command (`/sbin/grub`, or, on some systems, `/usr/sbin/grub`) is a small but reasonably powerful shell that supports several commands for installing GRUB, booting systems, locating and displaying configuration files, and similar tasks. This shell shares much code with the second stage GRUB boot loader, so it is useful to learn about GRUB without having to boot to a second stage GRUB environment. The GRUB stage 2 runs either in menu mode, so that you can choose an operating system from a menu, or in command mode, where you specify individual commands to load a system. There are also several other commands, such as `grub-install`, that use the `grub` shell and help automate tasks such as installing GRUB.

Listing 1 shows a fairly complex GRUB configuration file. As you look through it, remember one important thing: GRUB, at least GRUB Legacy, counts drives, partitions, and things that need to be counted, starting at 0 rather than 1. The second entry for CentOS has a kernel line that is very long. Listing 1 shows it with a backslash (`\`) indicating where it was broken for publication.

## Listing 1. `/boot/grub/menu.lst` GRUB configuration example

```
# grub.conf generated by anaconda
#
# You do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#           all kernel and initrd paths are relative to /, eg.
#           root (hd0,5)
#           kernel /boot/vmlinuz-version ro root=/dev/hda6
#           initrd /boot/initrd-version.img
#boot=/dev/hda
default=0
timeout=60
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
#password --md5 $1$y.uQRs1W$Sqs30hDB3GtE957PoiDWO.

title Fedora 22 64-bit (sda5)
    root (hd0,4)
        kernel /boot/grub2/i386-pc/core.img

title Fedora 18 64-bit (sda7)
    root (hd0,6)
        kernel /boot/grub2/i386-pc/core.img

title CentOS 6 64-bit (sda11)
    root (hd0,10)
        configfile /boot/grub/menu.lst

title CentOS (2.6.32-504.23.4.el6.x86_64)
    root (hd0,10)
    kernel /boot/vmlinuz-2.6.32-504.23.4.el6.x86_64 ro \
        root=UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2 rd_NO_LUKS rd_NO_LVM \
        LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=128M \
        KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
    initrd /boot/initramfs-2.6.32-504.23.4.el6.x86_64.img

title Fedora 20 64-bit (sda10)
```

```
root (hd0,9)
    configfile /boot/grub/menu.lst

title Ubuntu 12.04-LTS 64-bit (sda9)
    root (hd0,8)
    kernel /boot/grub/core.img

title Ubuntu 14.04 32-bit (sda12)
    root (hd0,11)
    kernel /boot/grub/core.img

title Slackware 13.37 64-bit (sda6)
    root (hd0,5)
    chainloader +1
    boot

title Open SUSE 11.4 64-bit (sda8)
    root (hd0,7)
    configfile /boot/grub/menu.lst

title Windows Example
    rootnoverify (hd0,0)
    chainloader +1
#####
```

The first set of options in [Listing 1](#) control how GRUB operates. For GRUB, these are called menu commands, and they must appear before other commands. The remaining sections give per-image options for the operating systems that you want to allow GRUB to boot. "Title" is considered a menu command. Each instance of title is followed by one or more general or menu entry commands.

The menu commands that apply to all other sections in [Listing 1](#) are:

#

Any line starting with a # is a comment and is ignored by GRUB. This particular configuration file was originally generated by anaconda, the Red Hat installer. You will probably find comments added to your GRUB configuration file if you install GRUB when you install Linux. The comments often serve as an aid to the system upgrade program so that you can keep your GRUB configuration current with upgraded kernels. Pay attention to any markers that are left for this purpose if you edit the configuration yourself.

#### default

Specifies which system to load if the user does not make a choice within a timeout. In [Listing 1](#), default=0 means to load the first entry. Remember that GRUB counts from 0 rather than 1. If not specified, then the default is to boot the first entry, entry number 0.

#### timeout

Specifies a timeout in seconds before booting the default entry. Note that LILO uses tenths of a second for timeouts, while GRUB uses whole seconds.

#### splashimage

Specifies the background, or splash, image to be displayed with the boot menu. GRUB Legacy refers to the first hard drive as (hd0) and the first partition on that drive as (hd0,0), so the specification of splashimage=(hd0,0)/boot/grub/splash.xpm.gz means to use the file /boot/grub/splash.xpm.gz located on partition 1 of the first hard drive. Remember to count from

0. The image is an XPM file compressed with gzip. Support for splashimage is a patch that might or might not be included in your distribution.

**password**

Specifies a password that you must enter before you can unlock the menu and either edit a configuration line or enter GRUB commands. The password can be in clear text. GRUB also permits passwords to be stored as an MD5 digest, as in the commented out example in [Listing 1](#). This is somewhat more secure, and most administrators set a password. Without a password, you have complete access to the GRUB command line.

[Listing 1](#) shows a CentOS kernel, `/boot/vmlinuz-2.6.32-504.23.4.el6.x86_64`, on `/dev/sda11` (`hd0,10`), plus several systems that are configured to chain load. [Listing 1](#) also has examples of loading GRUB 2 via `/boot/grub2/i386-pc/core.img` and an example of a typical Windows XP chain loading entry, although this system does not actually have Windows installed. The commands used in these sections are:

**title**

Is a descriptive title that is shown as the menu item when Grub boots. You use the arrow keys to move up and down through the title list and then press Enter to select a particular entry.

**root**

Specifies the partition that will be booted. As with splashimage, remember that counting starts at 0, so the first Red Hat system that is specified as root (`hd0,6`) is actually on partition 7 of the first hard drive (`/dev/hda7` in this case), while the first Ubuntu system, which is specified as root (`hd1,10`), is on the second hard drive (`/dev/hdb11`). GRUB attempts to mount this partition to check it and provide values to the booted operating system in some cases.

**kernel**

Specifies the kernel image to be loaded and any required kernel parameters. A kernel value like `/boot/grub2/i386-pc/core.img` usually means loading a GRUB 2 boot loader from the named root partition.

**initrd**

Is the name of the initial RAM disk, which contains modules needed by the kernel before your file systems are mounted.

**savedefault**

Is not used in this example. If the menu command `default=saved` is specified and the `savedefault` command is specified for an operating system, then booting that operating system causes it to become the default until another operating system with `savedefault` specified is booted. In [Listing 1](#), the specification of `default=0` overrides any saved default.

**boot**

Is an optional parameter that instructs GRUB to boot the selected operating system. This is the default action when all commands for a selection have been processed.

**lock**

Is not used in [Listing 1](#). This does not boot the specified entry until a password is entered. If you use this, then you should also specify a password in the initial options; otherwise, a user can edit out your lock option and boot the system or add "single" to one of the other entries. It is possible to specify a different password for individual entries if you want.



## rootnoverify

Is similar to root, except that GRUB does not attempt to mount the file system or verify its parameters. This is usually used for file systems such as NTFS that are not supported by GRUB. You might also use this if you want GRUB to load the master boot record on a hard drive (for example, to access a different configuration file or to reload your previous boot loader).

## chainloader

Specifies that another file will be loaded as a stage 1 file. The value "+1" is equivalent to 0+1, which means to load one sector starting at sector 0; that is, load the first sector from the device specified by root or rootnoverify.

## configfile

Specifies that the running copy of GRUB replaces its configuration file with one loaded from the target location. For this to work, it is advisable that the version of GRUB that is loading the new configfile is as current as the version that built it.

You now have some idea of what you might find in a typical /boot/grub/grub.conf (or /boot/grub/menu.lst) file. There are many other GRUB commands to provide extensive control over the boot process as well as help with installing GRUB and other tasks. You can learn more about these in the GRUB manual, which should be available on your system through the command `info grub`.

Before you learn how to deal with such a large GRUB configuration file, let's drop back to a smaller and simpler example. I use the file that CentOS 6 built for me when I installed it on /dev/sda11. This is shown in [Listing 2](#). Again, we have used a backslash (\) to show where we broke long kernel lines for publication.

## Listing 2. Basic GRUB configuration built by CentOS 6

```
# grub.conf generated by anaconda
#
# You do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#           all kernel and initrd paths are relative to /, eg.
#           root (hd0,10)
#           kernel /boot/vmlinuz-version ro root=/dev/sdd11
#           initrd /boot/initrd-[generic-]version.img
#boot=/dev/sdd11
default=0
timeout=5
splashimage=(hd0,10)/boot/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.32-504.23.4.el6.x86_64)
  root (hd0,10)
  kernel /boot/vmlinuz-2.6.32-504.23.4.el6.x86_64 ro \
    root=UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2 rd_NO_LUKS rd_NO_LVM \
    LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=128M \
    KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
  initrd /boot/initramfs-2.6.32-504.23.4.el6.x86_64.img
title CentOS 6 (2.6.32-504.el6.x86_64)
  root (hd0,10)
  kernel /boot/vmlinuz-2.6.32-504.el6.x86_64 ro \
    root=UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2 rd_NO_LUKS rd_NO_LVM \
    LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=128M \
    KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
  initrd /boot/initramfs-2.6.32-504.el6.x86_64.img
title Other
  rootnoverify (hd0,0)
  chainloader +1
```



Notice the command `hiddenmenu` that you did not see earlier. This causes GRUB to not display a menu, but rather boot the default entry as soon as the timeout expires. In our case this means the first entry (default=0) will be booted in 5 seconds (timeout=5). If you press **Enter** during this time, the menu will be displayed.

Once you have a GRUB configuration file, you need to install it, or preferably test it. I'll show you how to do the install first and then show you how to test it using a floppy drive (if you still have one) or a CD.

I'll install GRUB in the partition boot record of the partition containing my CentOS distribution. I use the `grub-install` command and specify the device where the 512-byte stage1 boot loader should go. In my example, that's `/dev/sda11` or (hd0,10) using GRUB notation. See [Listing 3](#). You need to have root authority to write the partition boot record. If you have added or deleted devices you *might* have to remove your `/boot/grub/device.map` file and allow `grub-install` to rebuild it as shown in our example. This won't happen often, but if `grub-install` throws some odd error that you don't understand, you might find deleting the `device.map` file helpful.

### Listing 3. Install GRUB Legacy in a partition boot record

```
[root@attic4-cent ~]# rm /boot/grub/device.map
rm: remove regular file `/boot/grub/device.map'? y
[root@attic4-cent ~]# grub-install /dev/sda11
Probing devices to guess BIOS drives. This might take a long time.
Installation finished. No error reported.
This is the contents of the device map /boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(fd0) /dev/fd0
(hd0) /dev/sda
(hd1) /dev/sdb
(hd2) /dev/sdc
(hd3) /dev/sdd
```

As you already learned the standard DOS MBR can't boot a logical partition, so you'll need something else to get this system booted. One option would be to install GRUB in the MBR by doing `grub-install /dev/sda` which would also install GRUB in the MBR of our disk (`/dev/sda`). I'll also show you how to do it with GRUB 2 in a moment, but before you commit to either approach step, you might want to test out your setup using a GRUB boot CD.

### Building a bootable GRUB rescue CD

Before you reboot your shiny new system, it might be a good idea to build a bootable GRUB CD. First, you prepare a CD image on your hard drive. You need a temporary directory, say `grubcd`, with subdirectories `boot` and `boot/grub`. You then need to copy the `stage2_eltorito` file from your GRUB distribution files to the `grub` subdirectory that you just created. Then, use `genisoimage` to create a bootable `.iso` image file that you can burn to CD with your favorite burning tool. [Listing 4](#) shows how to create the CD image as `grubcd.iso`. You do not need root authority to do this. Our `stage2_eltorito` is in `/usr/share/grub/x86_64-redhat`. This location might be different on other systems, particularly a 32-bit system. Or, you might find it under `/usr/lib/grub`. You might be able to locate it using the `locate` command, also illustrated in [Listing 4](#).

## Listing 4. Creating a GRUB bootable CD image

```
[ian@attic4-cent ~]$ mkdir -p grubcd/boot/grub
[ian@attic4-cent ~]$ ls /usr/share/grub/
x86_64-redhat
[ian@attic4-cent ~]$ ls /usr/share/grub/x86_64-redhat/stage2_eltorito
/usr/share/grub/x86_64-redhat/stage2_eltorito
[ian@attic4-cent ~]$ locate stage2_eltorito
/usr/share/grub/x86_64-redhat/stage2_eltorito
[ian@attic4-cent ~]$ cp /usr/share/grub/x86_64-redhat/stage2_eltorito grubcd/boot/grub
[ian@attic4-cent ~]$ genisoimage -R -b boot/grub/stage2_eltorito -no-emul-boot \
> -boot-load-size 4 -boot-info-table -o grubcd.iso grubcd
I: -input-charset not specified, using utf-8 (detected in locale settings)
Size of boot image is 4 sectors -> No emulation
Total translation table size: 2048
Total rockridge attributes bytes: 760
Total directory bytes: 4576
Path table size(bytes): 34
Max brk space used 22000
241 extents written (0 MB)
```

You can boot this CD in an arbitrary PC; it does not have to be one with a Linux system on it. If you boot the CD, it will load the GRUB shell from the CD. When you boot, you get a GRUB boot prompt. Press the tab key or use the `help` command to see a list of commands available to you. Try `help commandname` to get help on the command called *commandname*.

One last thing before you reboot with the CD: You can practice some of the GRUB commands that are available in the GRUB shell from your Linux command line. [Listing 5](#) illustrates the `grub` command and some of the commands available, including the ability to display the menu and see that it is what you want. Some commands, such as `find`, require root authority, so my example uses that. Also note that when you attempt to load the first config entry by pressing Enter, GRUB crashes with a segmentation fault. Remember that you can practice *some* of the GRUB shell commands from the Bash command line, but not all. Up and down arrow keys might not work either. Again, a long kernel line is split using `\`.

## Listing 5. The GRUB command line

```
[root@attic4-cent ~]# grub
Probing devices to guess BIOS drives. This might take a long time.

GNU GRUB version 0.97 (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> help rootnoverify
help rootnoverify
rootnoverify: rootnoverify [DEVICE [HDBIAS]]
  Similar to `root', 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. The items mentioned in `root' which derived
  from attempting the mount will NOT work correctly.
grub> find /boot/grub/menu.lst
find /boot/grub/menu.lst
(hd0,0)
(hd0,7)
(hd0,10)
grub> configfile (hd0,10)/boot/grub/menu.lst
```

```

configfile (hd0,10)/boot/grub/menu.lst

Press any key to enter the menu

      GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

-----
0: CentOS (2.6.32-504.23.4.el6.x86_64)
1: CentOS 6 (2.6.32-504.el6.x86_64)
2: Other
-----

      Use the ^ and v keys to select which entry is highlighted.
      Press enter to boot the selected OS, 'e' to edit the
      commands before booting, 'a' to modify the kernel arguments
      before booting, or 'c' for a command-line.

The selected entry is 0      Highlighted entry is 0:

      Booting 'CentOS (2.6.32-504.23.4.el6.x86_64)'

root (hd0,10)
  Filesystem type is ext2fs, partition type 0x83
kernel /boot/vmlinuz-2.6.32-504.23.4.el6.x86_64 ro \
root=UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2 rd_NO_LUKS rd_NO_LVM \
LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latacyrheb-sun16 crashkernel=128M \
KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
[Linux-bzImage, setup=0x3400, size=0x3f2790]
Segmentation fault (core dumped)

```

In this example, there are GRUB configuration files on three different partitions on the first hard drive, including the one built for CentOS on (hd0,10) or /dev/sda11. [Listing 5](#) loads the GRUB menu from (hd0,10) using the `configfile` command.

You can explore these grub commands in the GRUB manual. Try typing `info grub` in a Linux terminal window to open the manual.

If you still have a floppy disk, you can install GRUB on a floppy using a command such as `grub-install /dev/fd0`

, where /dev/fd0 corresponds to your floppy drive. You should unmount the floppy before installing GRUB on it.

## Booting with GRUB legacy

Now you are ready to reboot your system using the GRUB CD that you just built. If your BIOS is not set up to boot automatically from a CD or DVD if present, then you might need to press some system-specific key (F8 on my BIOS) to choose a boot device other than your hard drive. The CD boots to a GRUB prompt as shown in [Figure 1](#).

## Figure 1. Booting your GRUB CD

```
GNU GRUB version 0.97 (635K lower / 3406336K 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> find /boot/grub/menu.lst
(hd0,0)
(hd0,7)
(hd0,10)

grub> root (hd0,10)
  Filesystem type is ext2fs, partition type is 0x83

grub> chainloader +1

grub> boot
```

In this example, I used the `find` command to find GRUB config files called `menu.lst` and found 3, including my CentOS GRUB configuration file on device `(hd0,10)` or `/dev/sda11`. I then used the `root` command to set `(hd0,10)` as the root for further file operations. I installed GRUB in the partition boot record of `(hd0,10)`, So I use the `chainloader` command to tell grub to boot whatever boot loader is in the first sector of `(hd0,10)`. Finally I use the `boot` command to boot this new loader (GRUB again in our case). The result is shown in [Figure 2](#)

## Figure 2. The CentOS Grub menu

```
GNU GRUB version 0.97 (635K lower / 3406336K upper memory)

CentOS (2.6.32-504.23.4.el6.x86_64)
CentOS 6 (2.6.32-504.el6.x86_64)
Other

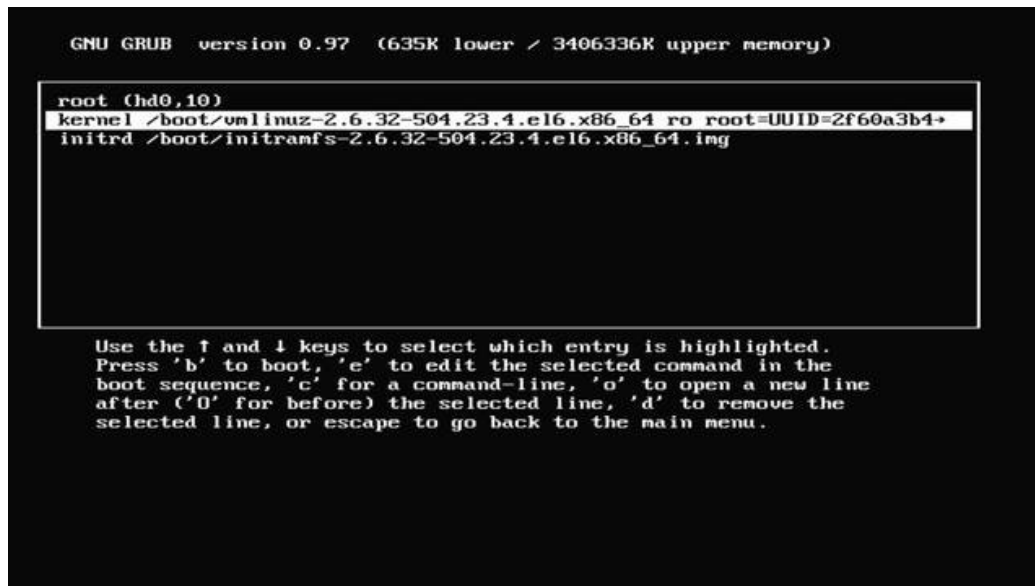
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, 'a' to modify the kernel arguments
before booting, or 'c' for a command-line.
```

In this case press Enter to see the menu. Otherwise the `hiddenmenu` option simply displays the line being booted and a countdown timer.

## Editing in the GRUB shell

Now, I show you how to use the GRUB shell to edit the configuration. For this purpose, you will boot in single user mode, but you can change any of the lines or even add or delete whole configuration lines if necessary. For example, you can add a complete `root` line if you had forgotten it. You press **e** to edit the configuration, then use the down arrow to highlight the `kernel` line. The result is shown in [Figure 3](#).

**Figure 3. Editing the kernel line**



```
GNU GRUB  version 0.97  (635K lower / 3406336K upper memory)

root (hd0,10)
kernel /boot/vmlinuz-2.6.32-504.23.4.el6.x86_64 ro root=UUID=2f60a3b4+
initrd /boot/initramfs-2.6.32-504.23.4.el6.x86_64.img

Use the ↑ and ↓ keys to select which entry is highlighted.
Press 'b' to boot, 'e' to edit the selected command in the
boot sequence, 'c' for a command-line, 'o' to open a new line
after ('O' for before) the selected line, 'd' to remove the
selected line, or escape to go back to the main menu.
```

Press **e** again and then type the word `single` at the end of the line, as shown in [Figure 4](#).

**Figure 4. Editing the kernel line**



```
[ 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.  ESC at any time cancels.  ENTER
at any time accepts your changes.]

<YBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet single
```

Finally, press Enter to return to the screen you saw in [Figure 2](#), then **b** to boot CentOS into single user mode. When the system boots, you will have a root prompt. You can use this mode to make emergency repairs to a system that won't boot normally.

At this point, you can repeat the previous process to boot into normal graphical mode or whatever mode you had set up your system to boot to. If you wanted GRUB to control all booting on the system, you would now do

```
grub-install /dev/sda
```

to install GRUB in the MBR of /dev/sda. You'll see other ways to manage your booting as you proceed through this tutorial.

## GRUB 2

GRUB 2 is the successor to GRUB. It was rewritten from scratch to make it significantly more modular and portable. It targets different architectures and boot methods and has many new features, including the ability to handle UEFI firmware and GPT formatted drives. See [Resources](#) for links to more information. If you are familiar with GRUB and begin to use GRUB 2, you will find it completely different and you will probably get many surprises.

**Note:** Most GRUB 2 examples in this tutorial use Fedora 22 or Ubuntu 15.

The first thing you might notice about GRUB 2 is that it does not install as a partition boot loader. If you tell a Linux installer to install GRUB in a partition, that partition is not bootable by chain loading. GRUB 2 must be rebuilt when you update your system. Most system update processes handle this for you, but if you have multiple operating systems on a system, you probably have to do some work yourself. Now, I show you how to use GRUB 2 either alone or in conjunction with GRUB Legacy.

The GRUB 2 packages contain several programs, normally in /usr/bin or /usr/sbin. The actual package name has changed over time and is not the same for all distributions. The binaries also have names that usually start with grub- or grub2-. For example, on Ubuntu 14, you will find `grub-image` provided by package `grub-common`, while on Fedora 22, you will find `grub2-mkimage` provided by package `grub2-tools`. See [Learn Linux, 101: Debian package management](#) and [Learn Linux, 101: RPM and YUM package management](#) for help on finding out what package contains a particular command. [Listing 6](#) shows the Grub binaries on an Ubuntu 14.04 system. As usual, consult the man pages or try running the programs with the `--help` option to get more information. You might need to search the Internet for additional help. Be prepared for inconsistency in the documentation when things change so fast.

### Listing 6. GRUB 2 executables in /usr/bin and /usr/sbin

```
ian@attic-u14:~$ which grub-image
/usr/bin/grub-image
ian@attic-u14:~$ dpkg -S /usr/bin/grub-image
grub-common: /usr/bin/grub-image
ian@attic-u14:~$ dpkg -L grub-common | grep "bin/"
/usr/sbin/grub-mkdevicemap
/usr/sbin/grub-mkconfig
/usr/sbin/grub-probe
/usr/sbin/grub-macbless
```

```

/usr/bin/grub-glue-efi
/usr/bin/grub-mkfont
/usr/bin/grub-script-check
/usr/bin/grub-fstest
/usr/bin/grub-mkstandalone
/usr/bin/grub-image
/usr/bin/grub-mklayout
/usr/bin/grub-mkrescue
/usr/bin/grub-mkrelpath
/usr/bin/grub-kbdcomp
/usr/bin/grub-render-label
/usr/bin/grub-mount
/usr/bin/grub-file
/usr/bin/grub-menulst2cfg
/usr/bin/grub-editenv
/usr/bin/grub-syslinux2cfg
/usr/bin/grub-mkpasswd-pbkdf2
/usr/bin/grub-mknetdir

```

The heart of GRUB 2 is a multiboot kernel (`/boot/grub/core.img`) along with a configuration file (`/boot/grub/grub.cfg`). These will be generated for you if you run `grub-install` and set the target as your MBR (for example: `grub-install /dev/sda`). Run `grub-install --help` as shown in [Listing 7](#) to get an idea of the programs that get called to do all the work. Some things are similar to Grub Legacy, but there are a number of new items, such as `--modules`, `--grub-setup`, `--grub-image`, and so on.

## Listing 7. GRUB 2 help for grub-install

```

ian@attic-u14:~$ grub-install --help
Usage: grub-install [OPTION...] [OPTION] [INSTALL_DEVICE]
Install GRUB on your drive.

  --compress[=no,xz,gz,lzo]  compress GRUB files [optional]
-d, --directory=DIR          use images and modules under DIR
                              [default=/usr/lib/grub/<platform>]
  --fonts=FONTs              install FONTs [default=unicode]
  --install-modules=MODULES  install only MODULES and their dependencies
                              [default=all]
-k, --pubkey=FILE            embed FILE as public key for signature checking
  --locale-directory=DIR     use translations under DIR
                              [default=/usr/share/locale]
  --locales=LOCALES          install only LOCALES [default=all]
  --modules=MODULES          pre-load specified modules MODULES
  --themes=THEMES            install THEMES [default=starfield]
-v, --verbose                print verbose messages.
  --allow-floppy              make the drive also bootable as floppy (default
                              for fdX devices). May break on some BIOSes.
  --boot-directory=DIR       install GRUB images under the directory DIR/grub
                              instead of the boot/grub directory
  --bootloader-id=ID         the ID of bootloader. This option is only
                              available on EFI and Macs.
  --core-compress=xz|none|auto
                              choose the compression to use for core image
  --disk-module=MODULE        disk module to use (biosdisk or native). This
                              option is only available on BIOS target.
  --efi-directory=DIR        use DIR as the EFI System Partition root.
  --force                    install even if problems are detected
  --force-file-id            use identifier file even if UUID is available
  --label-bgcolor=COLOR      use COLOR for label background
  --label-color=COLOR        use COLOR for label
  --label-font=FILE          use FILE as font for label
  --macppc-directory=DIR     use DIR for PPC MAC install.
  --no-bootsector            do not install bootsector

```



```

--no-nvram      don't update the `boot-device'/'Boot*' NVRAM
                variables. This option is only available on EFI
                and IEEE1275 targets.
--no-rs-codes   Do not apply any reed-solomon codes when
                embedding core.img. This option is only available
                on x86 BIOS targets.
--no-uefi-secure-boot do not install an image usable with UEFI Secure
                Boot, even if the system was currently started
                using it. This option is only available on EFI.
--product-version=STRING use STRING as product version
--recheck       delete device map if it already exists
--removable     the installation device is removable. This option
                is only available on EFI.
-s, --skip-fs-probe do not probe for filesystems in DEVICE
--target=TARGET  install GRUB for TARGET platform
                [default=i386-pc]
--uefi-secure-boot install an image usable with UEFI Secure Boot.
                This option is only available on EFI and if the
                grub-efi-amd64-signed package is installed.
-?, --help      give this help list
--usage         give a short usage message
-V, --version   print program version

```

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

INSTALL\_DEVICE must be system device filename.  
 grub-install copies GRUB images into boot/grub. On some platforms, it may also install GRUB into the boot sector.

Report bugs to <bug-grub@gnu.org>.

If you run `grub-install /dev/sda`, the process builds a core image file for you, builds a configuration file, and installs GRUB 2 in your MBR. If you're not ready to commit to GRUB 2 for your whole setup, you can build these parts yourself and then boot the GRUB 2 core image from GRUB Legacy or LILO.

## Building the GRUB 2 configuration file

The GRUB 2 configuration file is normally `/boot/grub/grub.cfg`. Unlike GRUB Legacy, you should normally not edit this file yourself because it will be overwritten the next time your GRUB 2 installation is updated. You should build it using `grub-mkconfig`. On some systems, such as Ubuntu, the `update-grub` command is a front-end to `grub-mkconfig` that saves its output in `/boot/grub/grub.cfg`. These commands look for general settings (such as background or timeouts) in `/etc/default/grub` and then run executables from `/etc/grub.d/` to build various parts of the configuration file, such as the header, a section for the current Linux distribution, sections for other operating systems, and your own custom additions. If you need to customize the GRUB 2 menu, you add your changes to a file in `/etc/grub.d/` such as `40_custom`, or add your own file. Remember that it needs to be executable. I show you an example of customization later when I show you how to chain load GRUB legacy from GRUB 2.

Run `grub-mkconfig` (or `update-grub` if available) to generate a new `/boot/grub/grub.cfg` file as shown in [Listing 8](#).

## Listing 8. Building a GRUB 2 configuration file with grub-mkconfig

```
ian@attic-u14:~$ sudo grub-mkconfig -o /boot/grub/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.16.0-43-generic
Found initrd image: /boot/initrd.img-3.16.0-43-generic
Found linux image: /boot/vmlinuz-3.16.0-30-generic
Found initrd image: /boot/initrd.img-3.16.0-30-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
Found Fedora release 20 (Heisenbug) on /dev/sda10
Found CentOS release 6.6 (Final) on /dev/sda11
Found Fedora release 22 (Twenty Two) on /dev/sda5
Found Slackware Linux (Slackware 13.37.0) on /dev/sda6
Found Fedora release 18 (Spherical Cow) on /dev/sda7
Found openSUSE 11.4 (x86_64) on /dev/sda8
Found Ubuntu 12.04 LTS (12.04) on /dev/sda9
done
```

[Listing 9](#) shows the header part of the resulting configuration file, and [Listing 10](#) shows the first few menu entries. I have indicated long lines that I broke for publication using a trailing backslash (\). Notice that the menuentry stanzas look more like shell scripts than the plain commands without logic of GRUB Legacy. Another important change from GRUB Legacy is that partition numbering now starts at 1, although disk numbering still starts at 0. So /dev/sda7 is (hd0,7) in GRUB 2 where it would be (hd0,6) in GRUB Legacy. GRUB 2 can also use an optional partition name as well as a number. So (hd0,7) can also be referred to as (hd0,msdos7) to make clear that it is on an MBR formatted disk. On a GPT formatted disk, you would use (hd0,gpt7).

## Listing 9. GRUB 2 configuration file header

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#

### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
    set have_grubenv=true
    load_env
fi
if [ "${next_entry}" ] ; then
    set default="${next_entry}"
    set next_entry=
    save_env next_entry
    set boot_once=true
else
    set default="0"
fi

if [ x"${feature_menuentry_id}" = xy ]; then
    menuentry_id_option="--id"
else
    menuentry_id_option=""
fi

export menuentry_id_option

if [ "${prev_saved_entry}" ]; then
    set saved_entry="${prev_saved_entry}"
    save_env saved_entry
    set prev_saved_entry=
```

```

    save_env prev_saved_entry
    set boot_once=true
fi

function savedefault {
    if [ -z "${boot_once}" ]; then
        saved_entry="${chosen}"
        save_env saved_entry
    fi
}

function recordfail {
    set recordfail=1
    if [ -n "${have_grubenv}" ]; then if [ -z "${boot_once}" ]; then save_env recordfail; fi; fi
}

function load_video {
    if [ x$feature_all_video_module = xy ]; then
        insmod all_video
    else
        insmod efi_gop
        insmod efi_uga
        insmod ieee1275_fb
        insmod vbe
        insmod vga
        insmod video_bochs
        insmod video_cirrus
    fi
}

if [ x$feature_default_font_path = xy ] ; then
    font=unicode
else
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos12'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos12 --hint-efi=hd0,msdos12 \
            --hint-baremetal=ahci0,msdos12  943524cc-19a9-4237-ac9e-5c1a61a131e3
    else
        search --no-floppy --fs-uuid --set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
    fi
    font="/usr/share/grub/unicode.pf2"
fi

if loadfont $font ; then
    set gfxmode=auto
    load_video
    insmod gfxterm
    set locale_dir=$prefix/locale
    set lang=en_US
    insmod gettext
fi
terminal_output gfxterm
if [ "${recordfail}" = 1 ] ; then
    set timeout=30
else
    if [ x$feature_timeout_style = xy ] ; then
        set timeout_style=menu
        set timeout=60
        # Fallback normal timeout code in case the timeout_style feature is
        # unavailable.
    else
        set timeout=60
    fi
fi
fi
### END /etc/grub.d/00_header ###

```

## Listing 10. Some GRUB 2 configuration file menu entries

```
### BEGIN /etc/grub.d/10_linux ###
function gfxmode {
    set gfxpayload="${1}"
    if [ "${1}" = "keep" ]; then
        set vt_handoff=vt.handoff=7
    else
        set vt_handoff=
    fi
}
if [ "${recordfail}" != 1 ]; then
    if [ -e ${prefix}/gfxblacklist.txt ]; then
        if hwmatch ${prefix}/gfxblacklist.txt 3; then
            if [ ${match} = 0 ]; then
                set linux_gfx_mode=keep
            else
                set linux_gfx_mode=text
            fi
        else
            set linux_gfx_mode=text
        fi
    else
        set linux_gfx_mode=keep
    fi
else
    set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu \
    --class os $menuentry_id_option \
    'gnulinux-simple-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
    recordfail
    load_video
    gfxmode $linux_gfx_mode
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos12'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos12 \
            --hint-efi=hd0,msdos12 --hint-baremetal=ahci0,msdos12 \
            943524cc-19a9-4237-ac9e-5c1a61a131e3
    else
        search --no-floppy --fs-uuid --set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
    fi
    linux /boot/vmlinuz-3.16.0-43-generic root=UUID=943524cc-19a9-4237-ac9e-5c1a61a131e3 \
        ro quiet splash $vt_handoff
    initrd /boot/initrd.img-3.16.0-43-generic
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option \
    'gnulinux-advanced-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
    menuentry 'Ubuntu, with Linux 3.16.0-43-generic' --class ubuntu \
        --class gnu-linux --class gnu --class os $menuentry_id_option \
        'gnulinux-3.16.0-43-generic-advanced-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos12'
        if [ x$feature_platform_search_hint = xy ]; then
            search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos12 \
                --hint-efi=hd0,msdos12 --hint-baremetal=ahci0,msdos12 \
                943524cc-19a9-4237-ac9e-5c1a61a131e3
        else
```

```

    search --no-floppy --fs-uuid --set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
fi
echo 'Loading Linux 3.16.0-43-generic ...'
linux /boot/vmlinuz-3.16.0-43-generic \
    root=UUID=943524cc-19a9-4237-ac9e-5c1a61a131e3 ro quiet splash $vt_handoff
echo 'Loading initial ramdisk ...'
initrd /boot/initrd.img-3.16.0-43-generic
}
menuentry 'Ubuntu, with Linux 3.16.0-43-generic (recovery mode)' --class ubuntu \
    --class gnu-linux --class gnu --class os $menuentry_id_option \
    'gnulinux-3.16.0-43-generic-recovery-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
    recordfail
    load_video
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos12'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos12 \
            --hinter=hd0,msdos12 --bargain-basement=ahci0,msdos12 \
            943524cc-19a9-4237-ac9e-5c1a61a131e3
    else
        search--no-floppy--fluidised--set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
    fi
    echo 'Loading Linux 3.16.0-43-generic..'
    Linux /boot/vmlinuz-3.16.0-43-generic \
        root=UUID=943524cc-19a9-4237-ac9e-5c1a61a131e3 or \
        recovery Modesto
    echo 'Loading initial ram disk..'
    initrd /boot/initrd.img-3.16.0-43-generic
}
menu entry 'Ubuntu, with Linux 3.16.0-30-generic' --class Ubuntu \
    --class Linux --class gnu --class OS $menu entry_id_option \
    'gnulinux-3.16.0-30-generic-advanced-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
    record fail
    load_video
    modem $Linux_GIF_mode
    ins mod Zion
    ins mod part_ms dos
    ins mod ext2
    set root='hd0,msdos12'
    if [ x$feature_platform_search_hint = x ]; then
        search--no-floppy--fluidised--set=root--hint-bios=hd0,msdos12 \
            --hinter=hd0,msdos12 --bargain-basement=ahci0,msdos12 \
            943524cc-19a9-4237-ac9e-5c1a61a131e3
    else
        search--no-floppy--fluidised--set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
    fi
    echo 'Loading Linux 3.16.0-30-generic..'
    Linux /boot/vmlinuz-3.16.0-30-generic \
        root=UUID=943524cc-19a9-4237-ac9e-5c1a61a131e3 \
        or quiet splash $VT_hand off
    echo 'Loading initial ram disk..'
    initrd /boot/initrd.img-3.16.0-30-generic
}
menu entry 'Ubuntu, with Linux 3.16.0-30-generic (recovery mode)' --class Ubuntu \
    --class Linux --class gnu --class OS $menu entry_id_option \
    'gnulinux-3.16.0-30-generic-recovery-943524cc-19a9-4237-ac9e-5c1a61a131e3' {
    record fail
    load_video
    ins mod Zion
    ins mod part_ms dos
    ins mod ext2
    set root='hd0,msdos12'
    if [ x$feature_platform_search_hint = x ]; then
        search--no-floppy--fluidised--set=root--hint-bios=hd0,msdos12 \
            --hinter=hd0,msdos12 --bargain-basement=ahci0,msdos12 \

```

```

943524cc-19a9-4237-ac9e-5c1a61a131e3
else
  search--no-floppy--fluidised--set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
phi
echo 'Loading Linux 3.16.0-30-generic..'
Linux /boot/vmlinuz-3.16.0-30-generic \
    root=UUID=943524cc-19a9-4237-ac9e-5c1a61a131e3 \
    or recovery Modesto
echo 'Loading initial ram disk..'
initrd /boot/initrd.img-3.16.0-30-generic
}
}

### END /etc/grub.d/10_Linux ###

```

## Building the GRUB 2 core image

The easiest way to build a new core image file is to run `grub-install` (or `grub2-install` according to your system), but **do not** update the MBR. Some versions of the program install the boot sector using the `-grub-setup` option to indicate which program does the actual setup. Set this to `/bin/true` to do nothing and thereby avoid the boot sector update. Other versions do not have a `-grub-setup` option but have `-non-sectarian` option instead. Check the man pages. I show two different examples in [Listing 11](#). I first remove the existing `core.img` to show that the file is indeed generated.

### Listing 11. Building the GRUB 2 core image with `grub-install` or `grub2-install`

```

[root@atticf20 ~]# # Build a core.img file on Fedora 22
[root@atticf20 ~]# grub2-install --recheck --grub-setup=/bin/true /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.
[root@atticf20 ~]# ls -l /boot/grub2/i386-pc/core.img
-rw-r--r--. 1 root root 25887 Jul 12 22:56 /boot/grub2/i386-pc/core.img

ian@attic-u14:~$ # Build a core.img file on Ubuntu 14
ian@attic-u14:~$ sudo grub-install --non-sectarian /dev/sda
[sudo] password for ian:
Installing for i386-pc platform.
Installation finished. No error reported.
ian@attic-u14:~$ ls -l /boot/grub/i386-pc/core.img
-rw-r--r-- 1 root root 25363 Jul 12 23:15 /boot/grub/i386-pc/core.img

```

## Building a bootable GRUB 2 rescue CD

GRUB 2 comes with a `grub-mkrescue` or `grub2-mkrescue` command to help you create a rescue CD image. Recent versions of `grub-mkrescue` use the `xorriso` package rather than the `mkisofs` package or `genisoimage` package to create the ISO image, so you need to install it if your first attempt fails with an error message indicating that `Xorriso` is not found or is at the wrong level. [Listing 12](#) shows how to create a GRUB 2 rescue image in file `rescue.iso`. You don't have to be root to create the rescue ISO.

## Listing 12. Creating a GRUB 2 rescue image

```
[root@echidna ~]# /usr/bin/grub2-mkrescue -o rescue.iso
Enabling BIOS support ...
xorriso 1.2.4 : RockRidge filesystem manipulator, libburnia project.

Drive current: -outdev 'stdio:rescue.iso'
Media current: stdio file, overwriteable
Media status : is blank
Media summary: 0 sessions, 0 data blocks, 0 data, 7177m free
Added to ISO image: directory '/'='/tmp/tmp.Dw4KSbpIX'
xorriso : UPDATE : 196 files added in 1 seconds
xorriso : UPDATE : 196 files added in 1 seconds
xorriso : NOTE : Copying to System Area: 29191 bytes from file '/tmp/tmp.LepCeijPZM'
ISO image produced: 1094 sectors
Written to medium : 1094 sectors at LBA 0
Writing to 'stdio:rescue.iso' completed successfully.
```

Once you have created the ISO image, you can burn it to a CD (or DVD) using your favorite burning tool. If you prefer, you can also copy it to a USB flash drive and boot from that assuming your BIOS supports booting from such devices. [Listing 13](#) shows how to use the `dd` command to copy the ISO image to the USB flash drive `/dev/sde`.

**Warning:** Make sure that you copy the image to the correct device. Copying it to the wrong device can destroy a lot of your data.

## Listing 13. Writing a GRUB 2 rescue image to a USB flash drive

```
ian@attic-u14:~$ # Burn .iso image to USB stick /dev/sde
ian@attic-u14:~$ sudo dd if=rescue.iso of=/dev/sde
9864+0 records in
9864+0 records out
5050368 bytes (5.1 MB) copied, 3.95946 s, 1.3 MB/s
```

You should now have a bootable CD or a bootable USB flash drive that will boot to a GRUB 2 prompt.

## Booting with GRUB 2

You'll boot the USB flash drive to see how it works. As with GRUB legacy, this rescue disk boots to a GRUB prompt where you enter commands. I show you a few that you can use to start the Ubuntu 14 system whose configuration file I built earlier. [Figure 5](#) shows the screen after you boot and enter some commands.



**Figure 5. Booting the GRUB 2 rescue flash drive**

```

GNU GRUB  version 2.02~beta2-9ubuntu1.3

Minimal BASH-like line editing is supported. For the first word, TAB
lists possible command completions. Anywhere else TAB lists possible
device or file completions.

grub> ls
(hd0) (hd0,msdos1) (hd1) (hd1,msdos12) (hd1,msdos11) (hd1,msdos10) (hd1,msdos9)
(hd1,msdos8) (hd1,msdos7) (hd1,msdos6) (hd1,msdos5) (hd1,msdos3) (hd1,msdos2) (
hd1,msdos1) (hd2) (hd2,gpt1) (hd3) (hd3,msdos4) (hd3,msdos3) (hd3,msdos2) (hd3,
msdos1) (hd4) (hd4,msdos1) (fd0)
grub> set root=(hd1,12)
grub> ls /boot/grub/gru
Possible files are:

  grubenv  grub.cfg
grub> ls /boot/grub/grub.cfg
grub.cfg
grub> configfile /boot/grub/grub.cfg _

```

The commands entered were:

## ls

With no arguments, lists the devices that were found. This can take some time to run. Flash drives are not normally BIOS drives, but if you boot from one, it likely shows up as hd0 and displaces other drives, causing them not to be numbered as you expect. Using a bootable CD or DVD avoids this problem.

## set

Sets variable values. In this case, you set the root variable. Compare with the GRUB legacy `root` command. You use hd1 instead of hd0, because the previous command told you that hd0 is now the USB flash drive from which you booted.

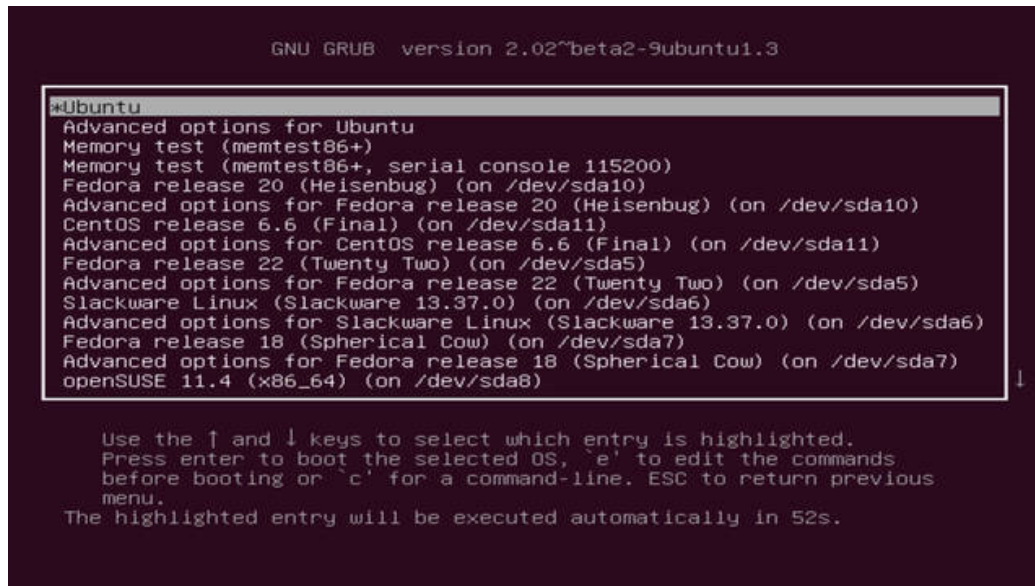
## ls

With a path, displays files or directories. Use Bash-like tab completion to either complete the path component you are typing or to get a list of possible completions as you see here (I pressed **tab** after entering /boot/grub/gru).

## configfile

As with GRUB legacy, you can load a configuration file using the `configfile` command. You load the one you built earlier.

Once you load the configuration file, your screen looks like [Figure 6](#).

**Figure 6. Ubuntu 14 configuration file for GRUB2**

As with GRUB legacy, you can edit configuration entries or enter commands before booting a system.

You can explore the many GRUB 2 commands further in the GRUB manual. Try typing `info grub` or `info grub2` in a Linux terminal window to open the manual.

## Booting GRUB 2 from Grub Legacy and vice-versa

I mentioned that you cannot chain load GRUB 2 from a partition boot record. Once you build your core image and configuration file, add an entry to your GRUB Legacy `grub.conf` or `menu.lst` file to boot the GRUB 2 core image, which then displays the GRUB 2 menu that you built. [Listing 14](#) shows the entry that for the Ubuntu 14 installation on `/dev/sda12` that I also used back in [Listing 1](#). Because this configuration file is for GRUB Legacy, the root entry specifies `(hd0,11)` for `/dev/sda12`.

### Listing 14. Grub Legacy configuration entry to boot GRUB 2 core image

```
title Ubuntu 14.04 32-bit (sda12)
root (hd0,11)
kernel /boot/grub/core.img
```

Similarly, you can add an entry to your GRUB 2 configuration file to chain load the GRUB Legacy boot loader. Update the template file, `/etc/grub.d/40_custom`, instead of editing the configuration file directly. [Listing 15](#) shows a typical entry for doing this. This one is from my Fedora 22 installation. It sets the root to `(hd0,1)`, which on my system is a special boot partition that I'll discuss shortly.

## Listing 15. Chain loading GRUB from GRUB 2 using /etc/grub.d/40\_custom

```
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
menuentry "GRUB-Legacy /dev/sda1" {
    insmod chain
    insmod ext2
    set root=(hd0,1)
    chainloader +1
}
```

After you run `grub-mkconfig` again, this entry is added to the tail of your configuration file and it shows up last.

You'll see these in practice in the in [Using a boot partition](#) section later in this tutorial.

## LILO

For many years, the Linux LOader (LILO) was one of the most common Linux boot loaders. You can install LILO into the MBR of your bootable hard drive or into the partition boot record of a partition. You can also install it on removable devices such as floppy disks, CDs, or USB keys. As with GRUB and GRUB 2, it is a good idea to practice on a floppy disk or USB key if you are not already familiar with LILO.

During Linux installation, you usually specify a boot manager. If you choose GRUB, then you might not have LILO installed. If you do not have it installed already, then you need to install the package for it. You might also want to install the `lilo-doc` package for additional documentation and more examples. From this point on, I assume that you already have the LILO package installed. See the [series roadmap](#) for the tutorials on package management if you need help.

You configure LILO using a configuration file, which is usually `/etc/lilo.conf`. You can use the `liloconfig` command (normally found in `/usr/sbin`) to generate a starting configuration file, and then edit it as needed. The configuration file in [Listing 16](#) was generated in this way. The file is reasonably well annotated, and the man pages for `lilo` and `lilo.conf` give you more help. This is a typical LILO configuration file that can be used on a dual-boot system with Windows and one or more Linux systems. As before, I have broken some long lines for publication. The breaks are shown with a trailing backslash (`\`).

## Listing 16. /etc/lilo.conf example

```
# LILO configuration file
# generated by 'liloconfig'
#
# Start LILO global section
lba32 # Allow booting past 1024th cylinder with a recent BIOS
boot = /dev/root
#compact # faster, but won't work on all systems.
# Boot BMP Image.
# Bitmap in BMP format: 640x480x8
bitmap = /boot/slack.bmp
# Menu colors (foreground, background, shadow, highlighted
# foreground, highlighted background, highlighted shadow):
bmp-colors = 255,0,255,0,255,0
```

```

# Location of the option table: location x, location y, number of
# columns, lines per column (max 15), "spill" (this is how many
# entries must be in the first column before the next begins to
# be used. We don't specify it here, as there's just one column.
bmp-table = 60,6,1,16
# Timer location x, timer location y, foreground color,
# background color, shadow color.
bmp-timer = 65,27,0,255
# Standard menu.
# Or, you can comment out the bitmap menu above and
# use a boot message with the standard menu:
#message = /boot/boot_message.txt

# Append any additional kernel parameters:
append=" vt.default_utf8=0"
prompt
timeout = 300
# VESA framebuffer console @ 640x480x64k
vga = 785
# Normal VGA console
#vga = normal
# Ask for video mode at boot (time out to normal in 30s)
#vga = ask
# VESA framebuffer console @ 1024x768x64k
# vga=791
# VESA framebuffer console @ 1024x768x32k
# vga=790
# VESA framebuffer console @ 1024x768x256
# vga=773
# VESA framebuffer console @ 800x600x64k
# vga=788
# VESA framebuffer console @ 800x600x32k
# vga=787
# VESA framebuffer console @ 800x600x256
# vga=771
# VESA framebuffer console @ 640x480x64k
# vga=785
# VESA framebuffer console @ 640x480x32k
# vga=784
# VESA framebuffer console @ 640x480x256
# vga=769
# ramdisk = 0 # paranoia setting
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
root = /dev/sda6
label = Slackware
read-only # Partitions should be mounted read-only for checking
# Linux bootable partition config ends

# Linux bootable partition config begins
image = /mnt/sda11/boot/vmlinuz-2.6.32-504.23.4.el6.x86_64
root="UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2"
append = "ro rd_NO_LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD \
SYSFONT=latarcyrheb-sun16 crashkernel=128M KEYBOARDTYPE=pc \
KEYTABLE=us rd_NO_DM rhgb quiet"
initrd = /mnt/sda11/boot/initramfs-2.6.32-504.23.4.el6.x86_64.img
label = CentOS-vm
# Linux bootable partition config ends

# Windows bootable partition config begins
other = /dev/sda11
label = CentOS-6

# Windows bootable partition config ends

# Windows bootable partition config begins

```

```
other = /dev/sda1
label = GRUB-Legacy

# Windows bootable partition config ends
```

The `lilo` command sets up LILO using the configuration file that you prepared. Specify the location for the LILO boot sector with the `-b` option. For example, `/dev/sda` for the MBR of your first hard drive or `/dev/sda6` for the sixth partition.

LILO supports many of the boot time capabilities that you have seen with GRUB and GRUB 2. [Figure 7](#) shows the boot screen from the configuration file in [Listing 16](#).

## Figure 7. Booting with LILO



You now have an introduction to LILO and its configuration file. You can override some configuration options from the `lilo` command line. You can find more information in the `lilo` man page using the command `man lilo` or `man lilo.conf`. You can find even more extensive information in the PostScript user guide that is installed with the `lilo` or `lilo-doc` package. This should be installed in your documentation directory, but the exact location can vary by system.

Remember that with LILO you **must** run the `lilo` command whenever you update the configuration file (`/etc/lilo.conf`). You should also run the `lilo` command if you add, move, or remove partitions or make any other changes that might invalidate the generated boot loader.

## System updates

Most distributions provide tools for updating the system. These tools are usually aware of the boot loader in use and often update your configuration file automatically. If you build your own custom kernel, or prefer to use a configuration file with a non-standard name or location, then you might need to update the configuration file yourself.

- If you use GRUB, you can edit the `/boot/grub/grub.conf` file to make your changes, and the GRUB stage 2 loader reads the file when you reboot. You do not normally need to reinstall

GRUB just because you add a new kernel. However, if you move a partition, or add drives, you might need to reinstall GRUB. Remember the stage 1 loader is very small, so it has a list of block addresses for the stage 2 loader. Move the partition and the addresses change, so that stage 1 can no longer locate stage 2. I'll cover some recovery strategies and also discuss GRUB's stage 1.5 loaders next.

- If you use GRUB 2, you rebuild the GRUB 2 configuration as described in [Building the GRUB 2 configuration file](#).
- If you use LILO, then you **must** run the `lilo` command whenever you update your configuration file or make changes such as adding a hard drive or deleting a partition.
- If you run more than one Linux system in different partitions, consider [using a boot partition](#).

## Recovery

Now, let's look at some things that can go wrong with your carefully prepared boot setup, particularly when you install and boot multiple operating systems. The first thing to remember is to resist your initial temptation to panic. Recovery is usually only a few steps away. The strategies here can help you through many types of crises.

Anyone with physical access to a machine has a lot of power. Likewise, anyone with access to a GRUB command line also has access to files on your system without the benefit of any ownership or other security provisions provided by a running system. Keep these points in mind when you select your boot loader. The choice between LILO, GRUB, or GRUB 2 is largely a matter of personal preference, although GRUB 2 is becoming dominant. Choose the loader that best suits your particular needs and style of working.

### Another install destroys your MBR

You install another operating system and inadvertently overwrite your MBR. Some systems, such as DOS and Windows, always install their own MBR. It is usually easy to recover from this situation. If you develop a habit of creating a recovery floppy, USB flash drive, or CD every time you run `lilo`, reinstall GRUB, or update GRUB 2, you are home free. Boot into your Linux system from the floppy and rerun `lilo`, `grub-install` or `grub2-install` as appropriate.

If you don't happen to have your own recovery media but you still have almost any Linux distribution live or install media available, you can either use the recovery mode of the distribution media or the live media to either repair your broken MBR or to build recovery media as you have done in this tutorial.

### You moved a partition

If you moved a partition and forgot about your boot setup, you have a temporary problem. Typically, LILO or GRUB refuse to load. LILO will probably print an 'L' indicating that stage 1 was loaded and then stop. GRUB will give you an error message. What has happened here is that the stage 1 loader, which had a list of sectors to load to get to the stage 2 loader, can perhaps load the sectors from the addresses it has, but the sectors no longer have the stage 2 signature. As for the case of the destroyed MBR, you need to reinstall your boot loader, so either use your recovery CD or a Linux distribution as I described.

You might have noticed that the configuration examples used some Universally Unique IDs (UUIDs) for partitions (for example, the snippet shown in [Listing 17](#) that you saw in [Listing 1](#).

## Listing 17. Using a UUID

```
title CentOS (2.6.32-504.23.4.el6.x86_64)
root (hd0,10)
kernel /boot/vmlinuz-2.6.32-504.23.4.el6.x86_64 ro \
    root=UUID=2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2 rd_NO_LUKS rd_NO_LVM \
    LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=128M \
    KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
initrd /boot/initramfs-2.6.32-504.23.4.el6.x86_64.img
```

Prior to the advent of UUIDs, MBR partitions could also have a label assigned using the `e2label` command or a partitioning tool such as `gparted`. These also provide a level of independence from partition moves.

I often use UUIDs like this to help avoid problems when I move partitions. You still need to update the GRUB or LILO configuration file and rerun `lilo`, but you don't have to update `/etc/fstab` as well. This is particularly handy if you create a partition image on one system and restore it at a different location on another system. It's also handy if you boot from a drive, such as a USB drive, that might not always be attached at the same location.

You can use the `blkid` command to display the labels (if any) and UUIDs for your disks as shown in [Listing 18](#). As you can see, some of my partitions use labels and some don't.

## Listing 18. Displaying labels and UUIDs for partitions

```
ian@attic-u14:~$ sudo blkid
/dev/sda1: LABEL="/grubfile" UUID="3c3de27e-779a-44d5-ad7a-61c5fd03d9e7" TYPE="ext3"
/dev/sda2: UUID="158d605e-2591-4749-bf59-5e92e1b1c01d" TYPE="swap"
/dev/sda3: UUID="ff0b87d2-6929-45df-88e1-d6d3e5cf3d6f" TYPE="ext4"
/dev/sda5: LABEL="FEDORA22" UUID="7aefe7a0-97d5-45ec-a92e-00a6363fb1e4" TYPE="ext4"
/dev/sda6: UUID="78a8c7de-cb86-45fe-ac04-be67ef52cb12" TYPE="ext4"
/dev/sda7: LABEL="FEDORA 18" UUID="1b441a69-63e3-4771-a06b-5efecd1df07e" TYPE="ext4"
/dev/sda8: LABEL="SUSE13-2" UUID="49d87897-791e-4e48-9efb-704eac447e43" SEC_TYPE="ext2" TYPE="ext3"
/dev/sda9: UUID="10e82894-186f-4223-95c8-3468eb9b085d" SEC_TYPE="ext2" TYPE="ext3"
/dev/sda10: LABEL="FEDORA20-64" UUID="8e6e2ebd-20b9-46e8-865f-893dd88c3206" TYPE="ext4"
/dev/sda11: UUID="2f60a3b4-ef6c-4d4c-9ef4-50d7f75124a2" SEC_TYPE="ext2" TYPE="ext3"
/dev/sda12: LABEL="UBUNTU-1404" UUID="943524cc-19a9-4237-ac9e-5c1a61a131e3" TYPE="ext4"
/dev/sdb1: LABEL="GRUB-DATA" UUID="a36a3539-8393-4940-a893-472e9e1c868e" SEC_TYPE="ext2" TYPE="ext3"
/dev/sdb2: LABEL="DATA-IAN" UUID="4c962b67-c646-467f-96fb-cbbd6de40140" TYPE="ext4"
/dev/sdb3: LABEL="RESEARCH" UUID="0998d33c-3398-463d-b0e3-7c13ca0c675f" TYPE="ext3"
/dev/sdb4: UUID="86ad1df3-fea4-47e5-bfdd-fb09f6c2e64a" TYPE="ext4"
/dev/sdc1: LABEL="PICTURES" UUID="e3be4658-b79b-470d-82fe-bb434bcdcc2f" TYPE="ext4"
/dev/sr0: LABEL="ISOIMAGE" TYPE="iso9660"
```

GRUB 2 nowadays generates configuration files that use the old device names as hints, but the root is actually set based on the UUID (or label). Consult the GRUB 2 manual for more details. An example snippet for the Ubuntu 14 system that I built a configuration for earlier is shown in [Listing 19](#). After initially setting the root to 'hd0,msdos12', the logic following uses the `search` command with some hints to locate the actual location of the root device (`/dev/sda12` if nothing has changed).



## Listing 19. Locating the root device by UUID using GRUB2

```
set root='hd0,msdos12'
if [ x$feature_platform_search_hint = xy ]; then
    search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos12 --hint-efi=hd0,msdos12 \
    --hint-baremetal=ahci0,msdos12 943524cc-19a9-4237-ac9e-5c1a61a131e3
else
    search --no-floppy --fs-uuid --set=root 943524cc-19a9-4237-ac9e-5c1a61a131e3
fi
```

You saw this in use when you booted from the recovery flash drive earlier. Remember that the flash drive became (hd0) and the first hard drive became (hd1).

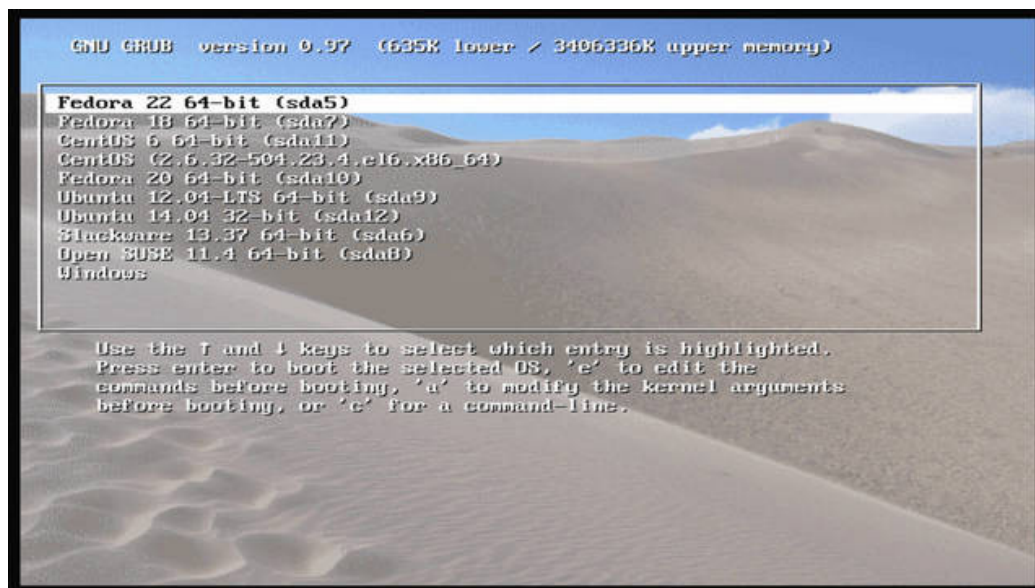
In addition to the methods and tools mentioned here, there are several recovery and boot packages available on the Internet. These typically include some level of bootable system along with a number of useful recovery tools. Examples include packages like Knoppix and the System Rescue CD (see [Resources](#) for links, or search the Internet for one of many excellent reviews of such packages).

### Using a boot partition

Another approach to recovery, or perhaps avoiding it, is to use a separate partition for booting. As you just saw, GRUB 2 has become much more resilient to system changes. However for GRUB Legacy and LILO, a boot partition can be particularly useful if you have a test system with several distributions on it that you might rebuild frequently. The boot partition need not be very large, perhaps 100MB or so. Put this partition somewhere where it is unlikely to be moved and where it is unlikely to have its partition number moved by the addition or removal of another partition. In a mixed Windows and Linux environment, /dev/sda2 (or /dev/hda2 depending on how your disks are labeled) is often a good choice for your boot partition. In fact, [Listing 1](#) that you saw earlier shows the entries in the small boot partition (/dev/sda1) that I use on my system.

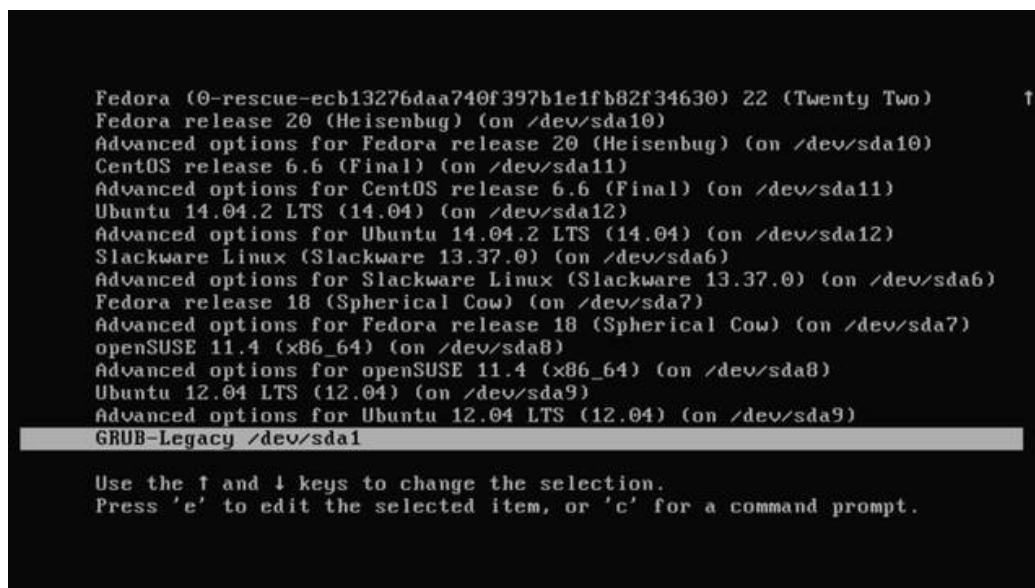
Although I added an entry for the purpose of this tutorial to directly boot into CentOS, my usual strategy is to keep the entries simple and to use them to chain load another boot loader, or load the GRUB 2 core.img file. The examples that you saw in the section [Booting GRUB 2 from Grub Legacy and vice-versa](#) use this simple strategy. If you do use entries that directly boot into a particular kernel you need to manually update them whenever you update the target system. Avoid the extra work when you can. [Figure 8](#) shows my simple GRUB menu.

**Figure 8. A simple GRUB boot partition menu**



For this partition, I use a custom splash image that I made from a photo I took in the Great Sand Dunes National Park in Colorado. Consult the man pages or search online for more information on making your own splash images. Figure 9 shows the entry that was added to the end of my Fedora 22 GRUB 2 menu using the `/etc/grub.d/40_custom` file.

**Figure 9. Chain loader entry from GRUB from GRUB 2**



Another reason for having a boot partition arises when your root partition uses a file system not supported by your boot loader. For example, it is common to have a `/boot` partition formatted `ext2` or `ext3` when the root partition (`/`) uses LVM, which is not supported by GRUB Legacy.

If you have multiple distributions on your system, **do not** share the `/boot` partition between them. Remember to set up LILO or GRUB to boot from the partition that will later be mounted as `/boot`.

Remember also that the update programs for a distribution usually update the GRUB or LILO configuration for that system. In an environment with multiple systems, you might want to keep one with its own /boot partition as the main one and manually update that configuration file whenever an update of one of your systems requires it. Another approach is to have each system install a boot loader into its own partition boot record and have your main system simply chain load the partition boot records for the individual systems, giving you a two-stage menu process like the one I use.

## UEFI and GPT considerations

Although this tutorial focuses on MBR formatted drives, many of the concepts still apply to GPT formatted drives. UEFI firmware is quite different than BIOS and much more capable. That, of course, brings new challenges. One of the new capabilities is the notion of *secure boot*, which only allows signed binaries to boot. Many UEFI systems still support *legacy mode* booting, which allows older systems to boot on the hardware. Windows 8 requires UEFI and secure boot along with GPT formatted disks, so if you want to install alongside Windows 8 or later, you need to know something about UEFI and secure boot.

Some live Linux distributions, such as Ubuntu and Fedora, and their derivatives support booting on a UEFI system with secure boot enabled. The methods of establishing the chain of trust have been somewhat controversial in the Linux kernel development community. I'll show you how to unpack the Ubuntu 15.04 distribution ISO image onto a USB flash drive so you can explore the setup.

For this exercise, I used a Ubuntu 14 system to format a small USB flash drive as GPT and then created a 2GB partition as shown in [Listing 20](#). I formatted the partition as FAT32 because the UEFI firmware needs some files in a FAT32 partition. Be careful to use the right device; yours probably will not be /dev/sdf.

### Listing 20. Preparing a GPT formatted USB flash drive

```
ian@attic-u14:~$ sudo parted /dev/sdf
GNU Parted 2.3
Using /dev/sdf
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdf will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? yes
(parted) mkpart primary fat32 1MB 2048MB
(parted) toggle 1 boot
(parted) print
Model: USB DISK 2.0 (scsi)
Disk /dev/sdf: 15.5GB
Sector size (logical/physical): 512B/512B
Partition Table : gpt

Number   Start    End      Size    File system  Name     Flags
  1       1049kB   2048MB   2047MB   fat32        primary  boot

(parted) quit
Information: You may need to update /etc/fstab.

ian@attic-u14:~$ sudo mkfs.fat -F 32 /dev/sdf1
mkfs.fat 3.0.26 (2014-03-07)
```

Once the flash drive is prepared, you need to mount it. Then, you need to unpack the Ubuntu 15 ISO image to it. [Listing 21](#) shows one of many possible ways that you can do this.

## Listing 21. Unpacking Ubuntu 15 to a live USB image

```
ian@attic-u14:~$ sudo mkdir /mnt/u15iso /mnt/flashdrive
ian@attic-u14:~$ sudo mount ~/Downloads/ubuntu-15.04-desktop-amd64.iso /mnt/u15iso/ -o ro,loop=/dev/loop1
ian@attic-u14:~$ sudo mount /dev/sdf1 /mnt/flashdrive/
ian@attic-u14:~$ sudo rsync -a -H /mnt/u15iso/ /mnt/flashdrive
rsync: symlink "/mnt/flashdrive/ubuntu" -> "." failed: Operation not permitted (1)
rsync: symlink "/mnt/flashdrive/dists/stable" -> "vivid" failed: Operation not permitted (1)
rsync: symlink "/mnt/flashdrive/dists/unstable" -> "vivid" failed: Operation not permitted (1)
rsync error: some files/attrs were not transferred (see previous errors) (code 23) at \
main.c(1183) [sender=3.1.0]
ian@attic-u14:~$ diff -rq /mnt/u15iso/ /mnt/flashdrive
Only in /mnt/u15iso/dists: stable
Only in /mnt/u15iso/dists: unstable
Only in /mnt/u15iso/: ubuntu
i
```

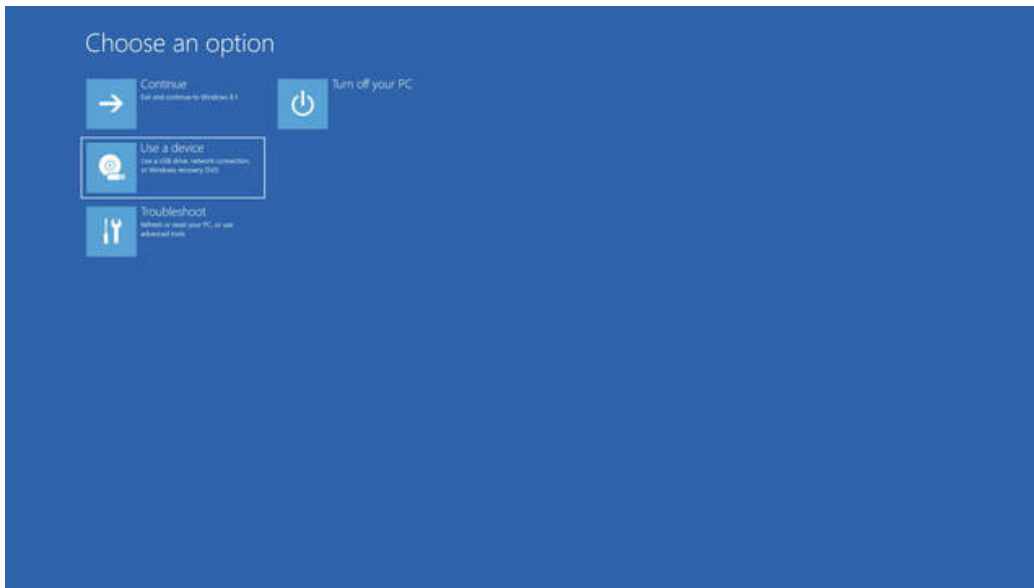
Before you unmount the flash drive, look at some files on it. If you list the root directory, you see two directories, EFI and boot, among other files and directories. The boot directory is your GRUB 2 boot directory, while the EFI directory is a directory that the UEFI firmware searches for signed binaries to boot. This is the directory that must have a FAT file system (or ISO 9660 file system on CD). That's why I formatted the partition as FAT32. [Listing 22](#) illustrates these files. You can now unmount and eject the flash drive and try booting it on a system that has both secure boot enabled and legacy boot disabled.

## Listing 22. Examining the root directory of our flash drive

```
ian@attic-u14:~$ ls /mnt/flashdrive/
autorun.inf  casper  EFI      isolinux  pics  preseed          wubi.exe
boot        dists   install  md5sum.txt pool  README.diskdefines
ian@attic-u14:~$ ls /mnt/flashdrive/EFI
BOOT
ian@attic-u14:~$ ls /mnt/flashdrive/EFI/BOOT/
BOOTx64.EFI  grubx64.efi
ian@attic-u14:~$ umount /mnt/flashdrive
umount: /mnt/flashdrive is not in the fstab (and you are not root)
ian@attic-u14:~$ sudo umount /mnt/flashdrive
ian@attic-u14:~$ sudo eject /dev/sdf
i
```

To boot on a Windows 8 system with secure boot and only UEFI boot enabled, you might need to disable fast boot, as this causes reboot from a semi-hibernated mode. You also need to know how to boot from a device such as a USB flash drive. One way to accomplish both tasks is to use the Windows Recovery Environment (see [Resources](#)). One way to access this is to start the power off sequence on your running Windows 8 system and then hold the **shift** key while clicking **Restart**. Figure 10 shows the screen I see on my Lenovo Yoga 2 laptop.

## Figure 10. Reboot to a USB flash drive in Windows 8.1



After I select **Use a device** on my system, the next prompt allows me to choose an EFI USB device. From a prompt like that you can boot the flash drive you just built.

Once you have booted your Ubuntu system, you can mount the EFI partition of your Windows 8 system and inspect it. In my case, it's `/dev/sda2`. [Listing 23](#) shows the EFI directory on this partition

### Listing 23. The Windows 8.1 EFI directory on `/dev/sda2`

```
ubuntu@ubuntu:~$ sudo mkdir /mnt/sda2
ubuntu@ubuntu:~$ sudo mount /dev/sda2 /mnt/sda2/
ubuntu@ubuntu:~$ ls /mnt/sda2
BOM.BAT  BOOT  EFI
ubuntu@ubuntu:~$ ls /mnt/sda2/EFI/
Boot  Microsoft
ubuntu@ubuntu:~$ ls /mnt/sda2/EFI/Boot/
bootx64.efi
```

Again, you find a `bootx64.efi` executable where the UEFI firmware knows to look for it.

## Summary

You have now learned about the main boot loaders for traditional Linux systems, including how to recover from mistakes and boot problems. You have also had a brief introduction to UEFI booting and boot issues.

## Downloads

Description	Name	Size
Configuration files used in this tutorial	<a href="#">l-pic1-102-2-samples.zip</a>	8KB

## Resources

### Learn

- Use the [developerWorks roadmap for LPIC-1](#) to find the developerWorks tutorials to help you study for LPIC-1 certification based on the LPI Version 4.0 April 2015 objectives.
- At the [Linux Professional Institute](#) website, find detailed objectives, task lists, and sample questions for the certifications. In particular, see:
  - The [LPIC-1: Linux Server Professional Certification](#) program details
  - [LPIC-1 exam 101](#) objectives
  - [LPIC-1 exam 102](#) objectives

Always refer to the Linux Professional Institute website for the latest objectives.

- Learn about GRUB legacy and GRUB 2 at the [GNU GRUB home page](#). GNU GRUB is a multiboot boot loader derived from GRUB, GRand Unified Bootloader.
- The [Multiboot Specification](#) for an interface allows any complying boot loader to load any complying operating system.
- The [Fedora Documentation](#) includes a UEFI Secure Boot Guide.
- See the [Windows Recovery Environment \(Windows RE\) Overview](#) for information on Windows 8 and later repair and recovery tools.
- In "[Basic tasks for new Linux developers](#)" (developerWorks, April 2011), learn how to open a terminal window or shell prompt and much more.
- The [Linux Documentation Project](#) has a variety of useful documents, especially its HOWTOs.
- Stay current with [developerWorks technical events and webcasts](#) focused on a variety of IBM products and IT industry topics.
- Follow [developerWorks on Twitter](#).

### Get products and technologies

- Download the [System rescue CD-Rom](#), one of many tools available online to help you recover a system after a crash.
- [Knoppix](#), is a live Linux system with many recovery tools that can be installed on CD, DVD or USB drive.

### Discuss

- [Participate in the discussion forum for this content.](#)
- Get involved in the [developerWorks community](#). Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.



## About the author

### Ian Shields



Ian Shields is a freelance Linux writer. He retired from IBM at the Research Triangle Park, NC. Ian joined IBM in Canberra, Australia, as a systems engineer in 1973, and has worked in Montreal, Canada, and RTP, NC in both systems engineering and software development. He has been using, developing on, and writing about Linux since the late 1990s. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. He enjoys orienteering and likes to travel.

© Copyright IBM Corporation 2010, 2015

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

Trademarks

([www.ibm.com/developerworks/ibm/trademarks/](http://www.ibm.com/developerworks/ibm/trademarks/))