developerWorks®

Learn Linux, 101: Boot the system

Go from BIOS to running Linux system

Ian Shields
Senior Programmer
IBM

20 September 2011

Learn to guide your Linux system through the boot process. You can use the material in this article to study for the LPI 101 exam for Linux system administrator certification, or just to learn about the boot process.

View more content in this series

Overview

This article will help you understand the boot sequence from BIOS to boot completion, and will show you how to:

- Give common commands to the boot loader
- Give options to the kernel at boot time
- Check boot events in the log files

Some aspects of the boot process are common to most systems, but some hardware-related aspects are specific to a particular architecture. The material in this article is directed specifically at x86 and x86_64 architecture systems using BIOS to boot the system. A newer system using *Extensible Firmware Interface* (or *EFI*) and the *GUID Partition Table* (*GPT*) is gaining popularity for drives larger than 2TB in size. This is not currently part of the LPI objectives and is not covered in this article. See Resources for more information.

Unless otherwise noted, the examples in this article generally use Fedora 14, with a 2.6.35 kernel. Your results on other systems may differ.

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

This article helps you prepare for Objective 101.2 in Topic 101 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 3.

Prerequisites

Connect with Ian

Ian is one of our most popular and prolific authors. Browse all of Ian's articles on developerWorks. Check out Ian's profile and connect with him, other authors, and fellow readers in My developerWorks.

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here. In particular, BIOS settings vary widely between systems, and boot loader splash screens vary widely between distributions.

You should also be familiar with the material in our article, "Learn Linux 101: Hard disk layout."

Boot sequence

Before we get into boot loaders, such as LILO and GRUB, let's review how a PC starts or *boots*. Code called *BIOS* (for *B*asic *I*nput *O*utput *S*ervice) 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 and performs a power-on self test (POST) to check the machine. It also determines the boot drive from the available removable or fixed storage devices and loads the first sector from the Master Boot Record (MBR) on that drive. The drive may be a traditional hard drive, solid-state drive, USB stick or drive, or a drive with removable media such as diskette, CD, or DVD. For the remainder of this article, we'll focus on hard drives, but the process is similar for the other types of storage devices.

As discussed in the article "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. Note that every disk, even a floppy, CD, or DVD, or solid-state device such as a USB stick, 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 when you want multiple operating systems, say OS/2, Windows XP, and three different Linux distributions? You *could* 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 multi-boot 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 allowing you to 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)
- GRUB2, a new boot loader that is starting to appear in common distributions

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 can also be done when the Linux boot loader for one system is asked to load the boot loader for another system. For example, you might use LILO in one partition to chain load GRUB in another partition in order to access the GRUB menu for that partition.

Linux boot loaders

From here on, we will focus on LILO and GRUB as these are the boot loaders included with most Linux distributions. LILO has been around for a while. GRUB is newer. The original GRUB has now become *GRUB Legacy*, and GRUB2 is being developed under the auspices of the Free Software Foundation (see Resources for details). We will discuss GRUB2 briefly, to show you the major differences and how GRUB and GRUB2 can coexist. For the rest of this article, we assume GRUB means GRUB Legacy, unless the context specifically implies GRUB2. 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 GRUB2 and ELILO.

The installation process for your distribution will probably give you a choice of which boot loader to set up. Either GRUB or LILO will work with most modern disks under 2TB in size, although some distributions, notably Fedora, no longer ship LILO. 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 may result in loss of data. Likewise, if you're adding a new distribution to an existing system, you may need to make sure you have the latest LILO or GRUB in your MBR. You will also need a fairly recent version of either GRUB or LILO if you plan to boot from an LVM or RAID disk.

The stage 2 loaders used in LILO and GRUB allow you to 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. GRUB2 also requires a rebuild from a configuration file that is normally stored in /etc.

To summarize the boot process for PCs:

1. When a PC is turned on, the BIOS (*Basic Input Output Service*) performs a self test.
2. When the machine passes its self test, the BIOS loads the *Master Boot Record* (or *MBR*, usually from the first 512-byte sector of the boot drive). The boot drive is usually the first hard drive on the system, but may also be a diskette, CD, or USB key.
3. For a hard drive, the MBR loads a stage 1 boot loader, which is typically either the LILO or GRUB stage1 boot loader on a Linux system. This is another 512-byte, single-sector record.
4. The stage 1 boot loader usually loads a sequence of records called the stage 2 boot loader (or sometimes the stage 1.5 loader).
5. The stage 2 loader loads the operating system. For Linux, this is the kernel and possibly an initial RAM disk (initrd).

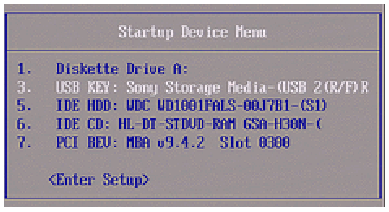
Your system should be able to install either of two popular boot loaders, LILO (the LInux LOader) or GRUB (the GRand Unified Boot loader). You should be able to use your chosen boot loader to boot normally as described above. Refer to the companion article "Learn Linux, 101: Boot managers" if you need to review boot loader installation or basic booting.

To influence your system's boot process, you can:

1. Change the device from which you boot. You may normally boot from a hard drive, but you may sometimes need to boot from a floppy disk, a USB memory key, a CD or DVD, or a network. Setting up such alternate boot devices requires your BIOS to be appropriately

configured and may require a particular keystroke during boot to display choices. The choices on one of my systems are illustrated in Figure 1. The method for setting up boot devices and selecting a boot device at startup is specific to your system and its BIOS. It is also beyond the scope of this LPI objective's requirements, so consult your system documentation.

Figure 1. Choosing a boot device



```
Startup Device Menu

1.   Diskette Drive A:
3.   USB KEY: Sony Storage Media-(USB 2(R/F)R
5.   IDE HDD: WDC WD1001FALS-00J7B1-(S1)
6.   IDE CD: HL-DT-STDVD-RAM GSA-H30N-(
7.   PCI BEV: MBA v9.4.2  Slot 0300

   <Enter Setup>
```

2. You can interact with the boot loader to select which of several possible configurations to boot. You will learn how to do this for both LILO and GRUB in this article.
3. You can use GRUB or LILO to pass parameters to the kernel to control the way that your kernel starts the system once it has been loaded by the boot loader.

LILO

The LILO configuration file defaults to /etc/lilo.conf. Listing 1 shows an example from a system that is currently running Slackware on /dev/sda10, Windows on /dev/sda1, and Fedora 14 on /dev/sdb12. The example was built using the `/sbin/liloconfig` command.

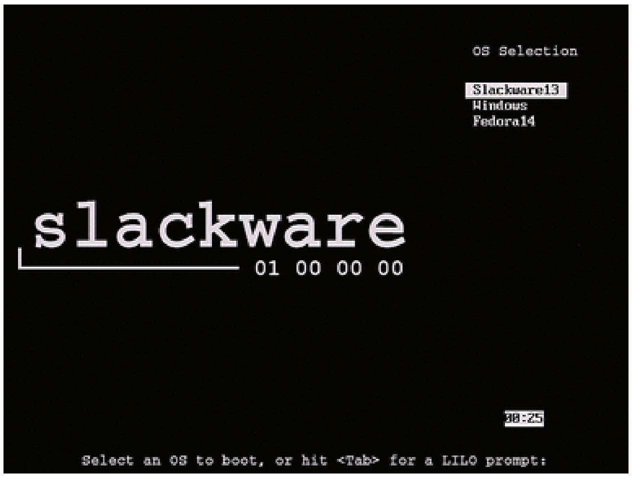
Listing 1. Sample LILO configuration

```
# 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
# Normal VGA console
vga = normal
# 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/sda10
label = Slackware13
read-only # Partitions should be mounted read-only for checking
# Linux bootable partition config ends
# Windows bootable partition config begins
other = /dev/sda1
label = Windows
table = /dev/sda
# Windows bootable partition config ends
# Linux bootable partition config begins
image = /boot/vmlinuz
root = /dev/sdb13
label = Fedora14
read-only # Partitions should be mounted read-only for checking
# Linux bootable partition config ends
```

The configuration file in Listing 1 contains the bitmap parameter set to /boot/slack.bmp. This is a bitmap image that will be displayed instead of the more traditional LILO text prompt, as shown in Figure 2. The default boot image, Slackware13, is highlighted and the other two, Windows and Fedora14, are shown below. Use the cursor keys to select a boot image and then press **Enter** to boot that choice. If the timeout in the configuration file is not 0, then the system will wait that number of tenths of a second, 300 tenths or 30 seconds in our example, and then boot the selected choice.

Figure 2. Slackware 13 LILO boot screen



```
Select an OS to boot, or hit <Tab> for a LILO prompt:
```

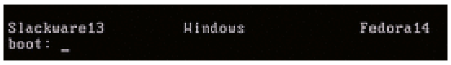
Using a text prompt

In the above example, we used a bitmap to display the boot choices. The traditional LILO boot prompt was a very terse text prompt similar to what you see if you press the **Tab** key. You can either type the name of a distribution to boot or press **Enter** to boot the first one. This is illustrated in Listing 2 and Figure 3.

Listing 2. LILO basic text prompt

```
Slackware13       Windows          Fedora14
boot:
```

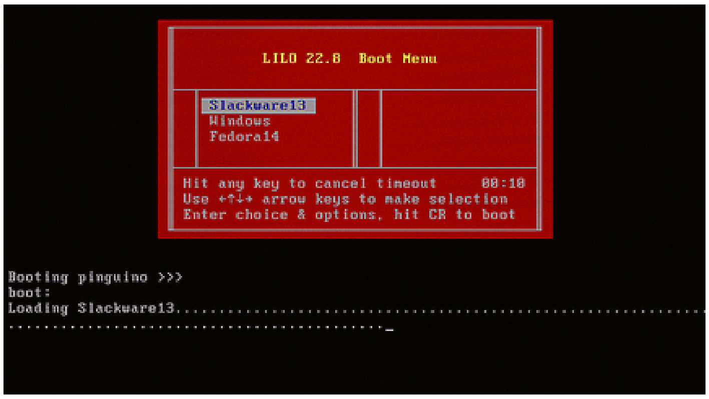
Figure 3. LILO basic text prompt



```
Slackware13         Windows             Fedora14
boot: _
```

Instead of the graphical prompt, you can get a text prompt by commenting out the `bitmap` and `bmpxxx` entries and uncommenting the `message` parameter so it points to a file containing the text message you would like to display. The text prompt is no longer as terse and now uses the curses library to display your choices. You may type parameters as before. In Figure 4, our message file contains the text "Booting pinguino >>>", and we again boot into single-user mode using the `s` parameter.

Figure 4. Slackware 13 LILO text (curses) boot screen



```
Hit any key to cancel timeout     00:10
Use ←↑↓→ arrow keys to make selection
Enter choice & options, hit CR to boot

Booting pinguino >>>
boot:
Loading Slackware13.............................................................
...........................................
```

Note: You may need to hold down the Shift key during boot to see the prompt, as a system may be configured so that it bypasses the prompt.

The lilo command

Remember that whenever you make changes to /etc/lilo.conf, or whenever you install a new kernel, you **must** run `lilo`. The `lilo` program rewrites the MBR or the partition boot record to reflect your changes, including recording the absolute disk location of the kernel. If your configuration file includes Linux images from multiple partitions, you must mount the partitions because the `lilo` command needs to access the partition to locate the image.

In addition to displaying the LILO configuration file, you may use the `-q` option of the `lilo` command to display information about the LILO boot choices. Add `-v` options for more verbose output. Two examples using the graphic configuration file of Listing 1 and the text configuration file used in Figure 4 are shown in Listing 3.

Listing 3. Displaying LILO configuration

```
                    root@echidna:~# lilo -q -C /etc/lilo-graphic.conf
Slackware13     *
Windows
Fedora14

root@echidna:~# lilo -q -v -C /etc/lilo-text.conf
LILO version 22.8, Copyright (C) 1992-1998 Werner Almesberger
Development beyond version 21 Copyright (C) 1999-2006 John Coffman
Released 19-Feb-2007 and compiled at 20:09:28 on Feb 14 2010

Reading boot sector from /dev/root
Installed:  Tue Sep  6 15:51:49 2011

Global settings:
  Delay before booting: 0.0 seconds
  Command-line timeout: 30.0 seconds
  No unattended booting
  No PC/AT keyboard hardware presence check
  Always enter boot prompt
  Boot-time BIOS data saved
  Boot-time BIOS data auto-suppress write bypassed
  Large memory (>15M) is NOT used to load initial ramdisk
  Non-RAID installation
  Boot device will not be used for the Map file
  Serial line access is disabled
  Boot prompt message is 1083 bytes
  No default boot command line
Images:
  Slackware13     *
    No password
    Boot command-line won't be locked
    No single-key activation
    VGA mode: NORMAL
    Kernel is loaded "high"
    No initial RAM disk
    No fallback
    Options: "ro root=80a  vt.default_utf8=0"
  Windows
    No password
    Boot command-line won't be locked
    No single-key activation
    No fallback
    No options
  Fedora14
    No password
    Boot command-line won't be locked
    No single-key activation
    VGA mode: NORMAL
    Kernel is loaded "high"
    No initial RAM disk
    No fallback
    Options: "ro root=81d  vt.default_utf8=0"
```

Chain loading with LILO

You may have noticed that the partition definition for Windows shows the device (/dev/sda1), while the Slackware and Fedora partition definitions show a boot image on the root partition, /dev/sda10 and /dev/sdb13, respectively. For systems where I may have several bootable partitions, I often set up a small partition containing GRUB, which can boot any of the other partitions, usually by chain loading, but by other means when, for example, the system on the partition uses a boot loader such as GRUB2. Since the partition runs GRUB Legacy, it can be chain loaded. To add a LILO entry for chain loading another partition, such as this GRUB partition, you can add an entry such as the one in Listing 4.

Listing 4. Adding a partition for chain loading GRUB

```
# Linux bootable partition config begins
other = /dev/sda2
  label = GRUB
# Linux bootable partition config ends
```

GRUB

The GRUB configuration file defaults to /boot/grub/grub.conf or /boot/grub/menu.lst. If both are present, one will usually be a symbolic link to the other. Listing 5 shows an example from the same system that you saw above for LILO. Note that we have split the three kernel definition statements into multiple lines for readability.

Listing 5. Sample GRUB configuration

```
# grub.conf generated by anaconda
#
# Note that 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 (hd1,12)
#          kernel /boot/vmlinuz-version ro root=/dev/sdb13
#          initrd /boot/initrd-[generic-]version.img
#boot=/dev/sdb13
default=0
timeout=5
splashimage=(hd1,12)/boot/grub/splash.xpm.gz
hiddenmenu
title Fedora (2.6.35.14-95.fc14.x86_64)
        root (hd1,12)
        kernel /boot/vmlinuz-2.6.35.14-95.fc14.x86_64 ro
            root=UUID=5e22a2e0-5ac2-47d5-b98a-d5b25eff585a
            rd_NO_LUKS rd_NO_LVM rd_NO_MD rd_NO_DM
            LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16
            KEYTABLE=us rhgb quiet
        initrd /boot/initramfs-2.6.35.14-95.fc14.x86_64.img
title Fedora (2.6.35.13-92.fc14.x86_64)
        root (hd1,12)
        kernel /boot/vmlinuz-2.6.35.13-92.fc14.x86_64 ro
            root=UUID=5e22a2e0-5ac2-47d5-b98a-d5b25eff585a
            rd_NO_LUKS rd_NO_LVM rd_NO_MD rd_NO_DM
            LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16
            KEYTABLE=us rhgb quiet
        initrd /boot/initramfs-2.6.35.13-92.fc14.x86_64.img
title Fedora (2.6.35.13-91.fc14.x86_64)
        root (hd1,12)
        kernel /boot/vmlinuz-2.6.35.13-91.fc14.x86_64 ro
            root=UUID=5e22a2e0-5ac2-47d5-b98a-d5b25eff585a
            rd_NO_LUKS rd_NO_LVM rd_NO_MD rd_NO_DM
            LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16
            KEYTABLE=us rhgb quiet
        initrd /boot/initramfs-2.6.35.13-91.fc14.x86_64.img
title GRUB
        rootnoverify (hd0,1)
        chainloader +1
title Slackware 13
        rootnoverify (hd0,9)
        chainloader +1
title Windows
        rootnoverify (hd0,0)
        chainloader +1
```

GRUB provides a menu interface. It can also use a password encrypted with the MD5 algorithm as opposed to the plain text password of LILO. And, perhaps most importantly, changes made to the GRUB configuration file do not require GRUB to be reinstalled in the MBR. Note that many distributions will automatically update the GRUB (or LILO) configuration file when updating to a new kernel level, but if you install a new kernel yourself or create a new initial RAM disk, you may need to edit the configuration file.

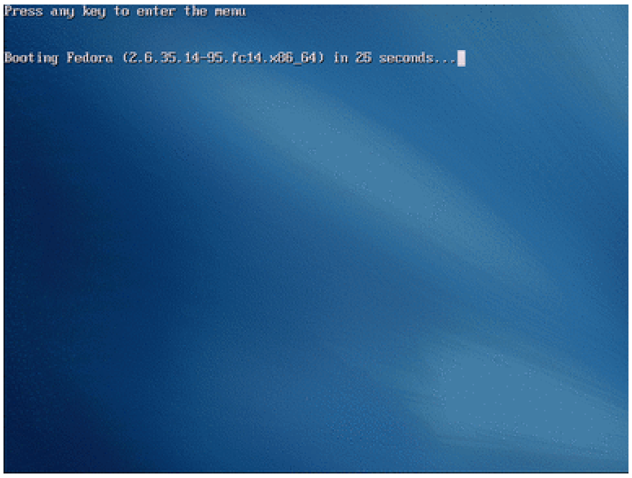
GRUB also does not require a partition to be mounted in order to configure a boot entry for it. You will notice entries such as `root (hd0,9)` and `splashimage=(hd1,12)/boot/grub/splash.xpm.gz`. GRUB refers to your hard disks as hd*n*, where *n* is an integer starting from 0. Similarly, partitions on a disk are numbered starting from 0. **Note:** GRUB2 has changed the way disks are named, so be careful if you are switching between GRUB and GRUB2.

So, on this system, (hd0,0) represents the Windows primary partition /dev/sda1, while (hd0,9) represents the Slackware logical partition /dev/sda10. A floppy drive is usually (fd0). Remember to quote these if you are invoking GRUB with parameters from a bash shell, for example, when installing GRUB onto a floppy, USB key, or your MBR.

Newer versions of GRUB allow you to use either a *label* or a *UUID* to specify the `root` element. See the section on "Labels, UUIDs, and links" in the companion article Learn Linux, 101: Control mounting and unmounting of filesystems in this series for more details on labels and UUIDs.

When you boot a system with GRUB, you will often see a default choice presented, as illustrated in Figure 5. If you do nothing, this will be booted after a configured timeout. The timeout is specified in seconds, so `timeout=5` in Listing 5 means a 5 second timeout.

Figure 5. Booting GRUB to a default choice

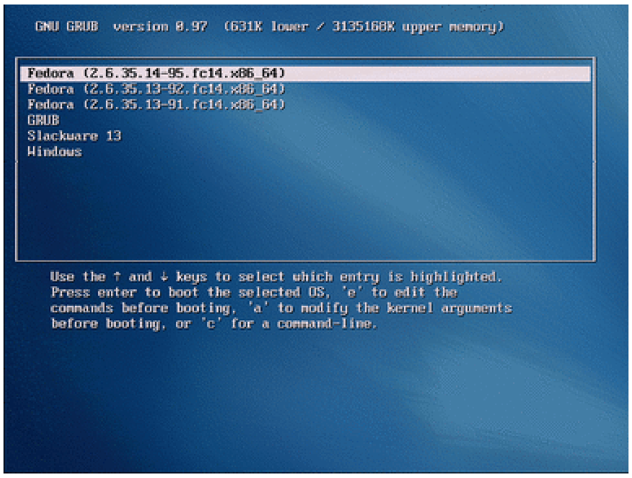


```
Press any key to enter the menu


Booting Fedora (2.6.35.14-95.fc14.x86_64) in 25 seconds...
```

If you press a key to interrupt the default boot, you will see a menu of choices similar to that in Figure 6.

Figure 6. GRUB menu choices



```
  GNU GRUB  version 0.97  (631K lower / 3135168K upper memory)

 Fedora (2.6.35.14-95.fc14.x86_64)
 Fedora (2.6.35.13-92.fc14.x86_64)
 Fedora (2.6.35.13-91.fc14.x86_64)
 GRUB
 Slackware 13
 Windows

    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.
```

When the GRUB menu is displayed, you select a boot image using the cursor movement keys to move up and down the list.

Custom backgrounds

If you want a different splash image for GRUB, you are limited to a 14-color X Window pixmap (.xpm) file that must be gzipped. Your favorite JPEG image may look a little different when reduced to 14 colors, so you will need to experiment. You can reduce to 14 colors using a graphical program such as the GIMP, where you would use the **Image->Mode->Indexed...** menu action and set the **Maximum number of colors** field to 14. Or you can use command line tools such as ImageMagick. Listing 6 shows one way of reducing an 800x531 pixel original to the required 640x480 pixels by first resizing it to 640 pixels in height, then cropping part of the image and reducing the canvas to match the image. Next we reduce the number of colors to 14 and convert the image from .jpg to .xpm. Finally we gzip the image.

Listing 6. Building a custom GRUB splash image

```
$ identify woodenbong.jpg
woodenbong.jpg JPEG 800x531 800x531+0+0 8-bit DirectClass 210KB 0.000u 0:00.000
$ convert -resize "800x640" woodenbong.jpg woodenbong2.jpg
$ convert -crop "640x480+40+0" +repage  woodenbong2.jpg  grub-in.jpg
$ convert -colors 14 grub-in.jpg grub-menu.xpm
$ gzip grub-menu.xpm
```

The GRUB shell

Unlike LILO, GRUB behaves as a small shell, with several commands that allow you to do things such as edit the commands before they are executed, find and load a configuration file, or display files using a `cat` command. From the menu, you may press **e** on an entry to edit it, **c** to switch to a GRUB command line, **b** to boot the system, **p** to enter a password, and **Esc** to return to the menu or to the previous step. There is also a `grub` command, which creates a simulated shell in which you may test your GRUB configuration or your GRUB command skills. Some basic components are available in normal user mode, but you will need root authority to perform many commands. Listing 7 illustrates how to start the grub shell as root. The grub command is usually found in /sbin or /usr/sbin.

Listing 7. Launching the GRUB shell

```
# grub

    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>
```

Within the GRUB shell, the `help` command provides a list of commands. Using `help commandname` provides help for the command named *commandname*. Listing 8 illustrates the available commands and the help for the `rootnoverify` command.

Listing 8. Using the GRUB shell

```
grub> help
blocklist FILE                         boot
cat FILE                               chainloader [--force] FILE
color NORMAL [HIGHLIGHT]               configfile FILE
device DRIVE DEVICE                    displayapm
displaymem                             find FILENAME
geometry DRIVE [CYLINDER HEAD SECTOR [ halt [--no-apm]
help [--all] [PATTERN ...]             hide PARTITION
initrd FILE [ARG ...]                  kernel [--no-mem-option] [--type=TYPE]
makeactive                             map TO_DRIVE FROM_DRIVE
md5crypt                               module FILE [ARG ...]
modulenounzip FILE [ARG ...]           pager [FLAG]
partnew PART TYPE START LEN            parttype PART TYPE
quit                                   reboot
root [DEVICE [HDBIAS]]                 rootnoverify [DEVICE [HDBIAS]]
serial [--unit=UNIT] [--port=PORT] [-- setkey [TO_KEY FROM_KEY]
setup [--prefix=DIR] [--stage2=STAGE2_ terminal [--dumb] [--no-echo] [--no-ed
terminfo [--name=NAME --cursor-address testvbe MODE
unhide PARTITION                       uppermem KBYTES
```

```
vbeprobe [MODE]

grub> 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. Note that the items mentioned in `root' which derived
    from attempting the mount will NOT work correctly.

grub>
```

As a practical example, you might continue with the previous example and use GRUB's `find` command to find configuration files. Next, you might load the configuration file from (hd1,12), which is /dev/sdb13, as illustrated in Listing 9.

Listing 9. Using GRUB to find and load a GRUB configuration file

```
grub> find /boot/grub/menu.lst
 (hd0,1)
 (hd0,5)
 (hd1,6)
 (hd1,8)
 (hd1,9)
 (hd1,10)
 (hd1,11)
 (hd1,12)

grub> configfile (hd1,12)/boot/grub/menu.lst
Press `ESC' to enter the menu... 24
```

When you load the configuration file, you might see a menu similar to that in Listing 10. Remember that this was done under the GRUB shell, which simulates the real GRUB environment and does not display the splash image. However, this is very similar to what you see superimposed on your splash image when you really boot the system using GRUB.

Note: If your menu does not look like this, and your arrow keys are echoed as something like ^[[A, then your version of the grub command may not include the curses support required for proper menu display. This is known to affect certain recent Fedora releases. In this case, you will have limited ability to test using the grub shell.

Listing 10. The GRUB menu

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

 +-------------------------------------------------------------------------+
 | Fedora (2.6.35.14-95.fc14.x86_64)                                       |
 | Fedora (2.6.35.13-92.fc14.x86_64)                                       |
 | Fedora (2.6.35.13-91.fc14.x86_64)                                       |
 | GRUB                                                                    |
 | Slackware 13                                                            |
 | Windows                                                                 |
 |                                                                         |
 |                                                                         |
 +-------------------------------------------------------------------------+
      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, or 'c' for a command-line.
```

Suppose you highlighted the third entry for `Fedora (2.6.35.13-91.fc14.x86_64)` and the pressed **e** to edit it. You would see something similar to Listing 11.

Listing 11. Editing a GRUB configuration entry

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

 +-------------------------------------------------------------------------+
 | root (hd1,13)                                                           |
 | kernel /boot/vmlinuz-2.6.35.13-91.fc14.x86_64 ro root=UUID=5e22a2e0-5>  |
 | initrd /boot/initramfs-2.6.35.13-91.fc14.x86_64.img                     |
 |                                                                         |
 |                                                                         |
 +-------------------------------------------------------------------------+
      Use the ^ and v 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.
```

Again, you use the arrow keys to select the line to be edited, and then press **e** to edit it. In this example, we'll suppose you had originally installed Fedora on /dev/sdb14 and then deleted a lower partition, say /dev/sdb8. This would drop the Fedora partition down to /dev/sdb13, or hd1,12 in GRUB notation. Use the cursor keys to move past the '3' in '(hd1,13)', then back up to delete '3' and insert '2' instead. When done, press **Enter** to accept your change, or press **Esc** to cancel. Finally, if you were really booting the system instead of running the GRUB shell, press **b** to boot the system.

When you boot into a GRUB shell, it has enough capability to display files on your filesystem, and it is run as though it were the root user, so you really need to protect your system with GRUB passwords. Remember, however, that if a user can boot from removable media, that user can provide his or her own GRUB configuration. See the security section in the GRUB manual (see Resources) for more information on GRUB security and other aspects of GRUB. You may also view it on your system using the `info grub` command.

Kernel parameters

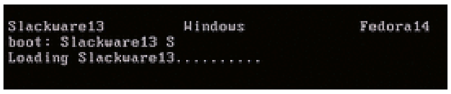
Kernel parameters (sometimes called boot parameters) supply the kernel with information about hardware parameters that it might not determine on its own, to override values that it might otherwise detect or to avoid detection of inappropriate values. For example, you might want to boot in single-user mode to repair your system, boot an SMP system in uniprocessor mode, or specify an alternate root filesystem. Some kernel levels require a parameter to enable large memory support on systems with more than a certain amount of RAM.

Sometimes you will need to add parameters to the boot process. For example, you may want to bring the system up in *single-user* mode, for which you will add the `s` parameter. Or you may need to specify a previously working kernel so you can find out why your newly built custom kernel won't boot. Press the **Tab** key to see a list of images to boot and the original terse LILO text prompt **boot:**. At this point, you may either type the name of an image, or press **Enter** to select the first entry. If you need to add parameters, type them after the image name. Listing 12 and Figure 7 show what you might see if you press the **Tab** key, type `Slackware13` to select the Slackware 13 image, add the **S** parameter to boot in single-user mode, and then press **Enter** to launch the boot process.

Listing 12. Specifying boot parameters with LILO

```
Slackware 13     Windows        Fedora 14
boot: Slackware13 S
```

Figure 7. Booting Slackware into single user mode



```
Slackware13         Windows            Fedora14
boot: Slackware13 S
Loading Slackware13..........
```

With GRUB, you could type in another set of commands for the kernel and `initrd` statements, or, preferably, you could use the edit facility described above to edit an existing entry. For example, add the **S** parameter to boot into single user mode.

The init process

When the kernel finishes loading, it usually starts `/sbin/init`. This program remains running until the system is shut down. It is always assigned process ID 1, as you can see in Listing 13.

Listing 13. The init process

```
[root@echidna ~]# ps --pid 1
  PID TTY          TIME CMD
    1 ?        00:00:00 init
```

The `init` program boots the rest of your system by running a series of scripts. These scripts typically live in /etc/rc.d/init.d or /etc/init.d, and they perform services such as setting the system's hostname, checking the filesystem for errors, mounting additional filesystems, enabling networking, starting print services, and so on. When the scripts complete, `init` starts a program called`getty`, which displays the login prompt on consoles. Graphical login screens are handled with a graphical display manager, such as GDM for Gnome.

If your system will load a kernel, but cannot run `init` successfully, you may try to recover by specifying an alternate initialization program. For example, specifying `init=/bin/sh` will boot your system into a shell prompt with root authority, from which you might be able to repair the system.

You can find out more about the available boot parameters by using the man pages for `bootparam`, or by browsing /usr/src/linux/Documentation/ramdisk.txt, which may be called /usr/src/linux-$(uname -r)/Documentation/kernel-parameters.txt on some systems.

Needless to say, if you have to apply the same set of additional parameters every time you boot, you should add them to the configuration file. Remember to rerun `lilo` if you are using LILO.

Boot events

During the Linux boot process, a large number of messages are emitted to the console, describing the kernel being booted, the hardware of your system, and other things related to the kernel.

These messages usually flash by quickly, and you probably won't be able to read them, unless there is a delay while the boot process waits for something, such as inability to reach a time server, or a filesystem that must be checked. With the advent of the Linux Bootsplash project (see Resources), these messages may be superimposed on a graphical background, or they may be hidden and replaced by a simple status bar. If your distribution supports the hidden mode, you will usually be able to switch back to displaying boot messages by pressing a key such as F2.

dmesg

It's nice to be able to go back and review the kernel messages. Since standard output is related to a process, and the kernel does not have a process identifier, it keeps kernel (and module) output messages in the *kernel ring buffer*. You may display the kernel ring buffer using the `dmesg` command, which displays these messages on standard output. Of course, you may redirect this output to a file for later analysis or forward it to a kernel developer for debugging purposes. Listing 14 illustrates some of the output that you might see.

Listing 14. Partial dmesg output

```
[root@echidna ~]# dmesg | head -n 30
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 2.6.35.14-95.fc14.x86_64 (mockbuild@x86-07.phx2.fedoraprojec
t.org) (gcc version 4.5.1 20100924 (Red Hat 4.5.1-4) (GCC) ) #1 SMP Tue Aug 16 21:01:58 U
TC 2011
[    0.000000] Command line: ro root=UUID=5e22a2e0-5ac2-47d5-b98a-d5b25eff585a rd_NO_LUKS
 rd_NO_LVM rd_NO_MD rd_NO_DM LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16 KEYTABLE=us rhgb
quiet
[    0.000000] BIOS-provided physical RAM map:
[    0.000000]  BIOS-e820: 0000000000000000 - 000000000009dc00 (usable)
[    0.000000]  BIOS-e820: 000000000009dc00 - 00000000000a0000 (reserved)
[    0.000000]  BIOS-e820: 00000000000e0000 - 0000000000100000 (reserved)
[    0.000000]  BIOS-e820: 0000000000100000 - 00000000bf6b0000 (usable)
[    0.000000]  BIOS-e820: 00000000bf6b0000 - 00000000bf6c8000 (ACPI data)
[    0.000000]  BIOS-e820: 00000000bf6c8000 - 00000000bf6cb000 (ACPI NVS)
[    0.000000]  BIOS-e820: 00000000bf6cb000 - 00000000c0000000 (reserved)
[    0.000000]  BIOS-e820: 00000000e0000000 - 00000000f0000000 (reserved)
[    0.000000]  BIOS-e820: 00000000fec00000 - 00000000fec10000 (reserved)
[    0.000000]  BIOS-e820: 00000000fed1c000 - 00000000fed20000 (reserved)
[    0.000000]  BIOS-e820: 00000000fee00000 - 00000000fee01000 (reserved)
[    0.000000]  BIOS-e820: 00000000ff000000 - 0000000100000000 (reserved)
[    0.000000]  BIOS-e820: 0000000100000000 - 0000000134000000 (usable)
[    0.000000] NX (Execute Disable) protection: active
[    0.000000] DMI present.
[    0.000000] e820 update range: 0000000000000000 - 0000000000001000 (usable) ==> (reserv
ed)
[    0.000000] e820 remove range: 00000000000a0000 - 0000000000100000 (usable)
[    0.000000] No AGP bridge found
[    0.000000] last_pfn = 0x134000 max_arch_pfn = 0x400000000
[    0.000000] MTRR default type: uncachable
[    0.000000] MTRR fixed ranges enabled:
[    0.000000]   00000-9FFFF write-back
[    0.000000]   A0000-BFFFF uncachable
[    0.000000]   C0000-C7FFF write-protect
[    0.000000]   C8000-E3FFF uncachable
```

The kernel ring buffer is also used for some events after the system is booted. These include certain program failures and hot-plug events. Listing 15 shows several entries related to plugging in a USB memory key.

Listing 15. Later events in kernel ring buffer

```
[root@echidna ~]# dmesg | tail -n 19
[70259.964953] usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[70259.964957] usb 1-4: Product: Storage Media
[70259.964960] usb 1-4: Manufacturer: Sony
[70259.964963] usb 1-4: SerialNumber: 0205093006441
[70260.228187] Initializing USB Mass Storage driver...
[70260.229608] scsi4 : usb-storage 1-4:1.0
[70260.229749] usbcore: registered new interface driver usb-storage
[70260.229752] USB Mass Storage support registered.
[70261.232195] scsi 4:0:0:0: Direct-Access     Sony     Storage Media    0100 PQ: 0 ANSI:
 0 CCS
[70261.233159] sd 4:0:0:0: Attached scsi generic sg3 type 0
[70261.237931] sd 4:0:0:0: [sdc] 1014784 512-byte logical blocks: (519 MB/495 MiB)
[70261.238809] sd 4:0:0:0: [sdc] Write Protect is off
[70261.238814] sd 4:0:0:0: [sdc] Mode Sense: 43 00 00 00
[70261.238818] sd 4:0:0:0: [sdc] Assuming drive cache: write through
[70261.243554] sd 4:0:0:0: [sdc] Assuming drive cache: write through
[70261.243566]  sdc:
[70261.317555] sd 4:0:0:0: [sdc] Assuming drive cache: write through
[70261.317561] sd 4:0:0:0: [sdc] Attached SCSI removable disk
[70261.616396] SELinux: initialized (dev sdc, type vfat), uses genfs_contexts
```

/var/log/messages

Once your system has started to the point of running `/sbin/init`, the kernel still logs events in the ring buffer as you just saw, but processes use the syslog daemon to log messages, usually in /var/log/messages. In contrast to the ring buffer, each syslog line has a timestamp, and the file persists between system restarts. This file is where you should first look for errors that occurred during the init scripts stage of booting.

Most daemons have names that end in 'd'. Listing 16 shows how to see the last few daemon status messages after a reboot.

Listing 16. Daemon messages from /var/log/messages

```
[root@echidna ~]# grep "^Sep.*d\:" /var/log/messages|tail -n 14
Sep  7 18:21:08 echidna acpid: waiting for events: event logging is off
Sep  7 18:21:12 echidna abrtd: Registered Analyzer plugin 'Kerneloops'
Sep  7 18:21:12 echidna abrtd: Registered Analyzer plugin 'Python'
Sep  7 18:21:12 echidna abrtd: Registered Reporter plugin 'Logger'
Sep  7 18:21:13 echidna abrtd: Registered Analyzer plugin 'CCpp'
Sep  7 18:21:13 echidna abrtd: Registered Action plugin 'KerneloopsScanner'
Sep  7 18:21:13 echidna abrtd: Registered Reporter plugin 'Bugzilla'
Sep  7 18:21:13 echidna abrtd: Registered Reporter plugin 'KerneloopsReporter'
Sep  7 18:21:13 echidna abrtd: Checking for unsaved crashes (dirs to check:5)
Sep  7 18:21:13 echidna abrtd: Registered Database plugin 'SQLite3'
Sep  7 18:21:13 echidna abrtd: Done checking for unsaved crashes
Sep  7 18:21:13 echidna abrtd: Init complete, entering main loop
Sep  7 18:21:26 echidna auditd[2032]: Started dispatcher: /sbin/audispd pid: 2034
Sep  7 18:21:26 echidna audispd: audispd initialized with q_depth=120 and 1 active plugins
```

You will also find logs for many other system programs in /var/log. For example, you can see the startup log for your X Window system.

This completes your introduction to guiding your Linux system through the boot process.

Resources

Learn

- Develop and deploy your next app on the IBM Bluemix cloud platform.
- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- See "Make the most of large drives with GPT and Linux" (developerWorks, July 2009) for more information on Extensible Firmware Interface (EFI) and the GUID Partition Table (GPT).
- Learn about ELILO at the ELILO: EFI Linux Boot Loader project home page.
- The Linux BootPrompt-HowTo helps you understand boot parameters.
- See the Upstart overview for more information on upstart.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

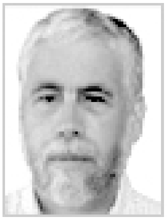
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

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

About the author

Ian Shields



Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. 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. Learn more about Ian in Ian's profile on developerWorks Community.