

Chapter 5

Explaining the Boot Process

✓ Objective 1.1: Summarize Linux fundamentals.





Before you can log in and start using your Linux system, a complicated process of booting the operating system must take place. A lot happens behind the scenes in the Linux boot process. It helps to know just what all goes on in case something goes wrong.

This chapter examines the boot and startup processes in Linux systems. First, we'll look at the role the computer firmware plays in getting the process started, and then we'll discuss Linux bootloaders and how to configure them. Next, the chapter discusses the Linux initialization process, showing how Linux decides which background applications to start at bootup. The chapter ends by taking a look at some system recovery options you have available to help salvage a system that won't boot.

The Linux Boot Process

When you turn on the power to your Linux system, it triggers a series of events that eventually leads to the login prompt. Normally you don't worry about what happens behind the scenes of those events; you just log in and start using your applications.

However, there may be times when your Linux system doesn't boot quite correctly, or perhaps an application you expected to be running in background mode isn't. In those cases, it helps to have a basic understanding of just how Linux boots the operating system and starts programs so you can troubleshoot the problem.

The following sections walk through the steps of the boot process and how you can watch the boot process to see what steps failed.

Following the Boot Process

The Linux boot process can be split into three main steps:

1. The workstation firmware starts, performing a quick check of the hardware (called a *Power-On Self-Test*, or *POST*) and then looks for a bootloader program to run from a bootable device.
2. The bootloader runs and determines what Linux kernel program to load.
3. The kernel program loads into memory and starts the necessary background programs required for the system to operate (such as a graphical desktop manager for desktops or web and database servers for servers).

While on the surface these three steps may seem simple, a ballet of operations happens to keep the boot process working. Each step performs several actions as they prepare your system to run Linux.

Viewing the Boot Process

You can monitor the Linux boot process by watching the system console screen as the system boots. You'll see lots of informative messages scroll by as the system detects hardware and loads software.



Some graphical desktop Linux distributions hide the boot messages in a separate console window when they start up. Often you can press either the Esc key or the Ctrl+Alt+F1 key combination to view those messages.

Usually the boot messages scroll by quickly and it's hard to see what's happening. If you need to troubleshoot boot problems, you can review the boot time messages using the `dmesg` command. Most Linux distributions copy the boot kernel messages into a special ring buffer in memory, called the *kernel ring buffer*. The buffer is circular and set to a predetermined size. As new messages are logged in the buffer, older messages are rotated out.

The `dmesg` command displays the most recent boot messages that are currently stored in the kernel ring buffer, as shown in Listing 5.1.

Listing 5.1 The `dmesg` command output from an Ubuntu workstation

```
$ dmesg
[ 0.000000] Linux version 5.11.0-40-generic (buildd@lgw01-amd64-
010) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils
for Ubuntu) 2.34) #44~20.04.2-Ubuntu SMP Tue Oct 26 18:07:44 UTC 2021
(Ubuntu 5.11.0-40.44~20.04.2-generic 5.11.22)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.11.0-40-
generic root=UUID=5423117e-4aaf-4416-ada7-01e07073b2e1 ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Hygon HygonGenuine
[ 0.000000] Centaur CentaurHauls
[ 0.000000] zhaoxin Shanghai
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point
registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832
bytes, using 'standard' format.
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbff] usable
[ 0.000000] BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved
```

```
[ 0.000000] BIOS-e820: [mem 0x000000000000f0000-0x000000000000fffff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000000100000-0x00000000000dfffff] usable
[ 0.000000] BIOS-e820: [mem 0x00000000000dfff0000-0x00000000000dffffff] ACPI data
[ 0.000000] BIOS-e820: [mem 0x000000000fec00000-0x000000000fec00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000fee00000-0x000000000fee00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000fffc0000-0x000000000ffffffff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000100000000-0x0000000021ffffffff] usable
[ 0.000000] NX (Execute Disable) protection: active
[ 0.000000] SMBIOS 2.5 present.
[ 0.000000] DMI: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox
12/01/2006
[ 0.000000] Hypervisor detected: KVM
[ 0.000000] kvm-clock: Using msrs 4b564d01 and 4b564d00
[ 0.000000] kvm-clock: cpu 0, msr 109001001, primary cpu clock
[ 0.000000] kvm-clock: using sched offset of 3933158608 cycles
[ 0.000001] clocksource: kvm-clock: mask: 0xffffffffffffffff max_cycles:
0x1cd42e4dffb, max_idle_ns: 881590591483 ns
[ 0.000003] tsc: Detected 1497.589 MHz processor
```

Most Linux distributions also store the boot messages in a log file, usually in the `/var/log` folder. For both Debian-based and Red Hat-based systems, the file is usually `/var/log/boot.log`, but some legacy Linux systems may use `/var/log/boot`.

While it helps to be able to see the different messages generated during boot time, it is also helpful to know what generates those messages. This chapter discusses each of these three boot steps and goes through some examples showing just how they work.

The Firmware Startup

All IBM-compatible workstations and servers utilize some type of built-in firmware to control how the installed operating system starts. On older workstations and servers, this firmware was called the *Basic Input/Output System (BIOS)*. On newer workstations and servers, a method called the *Unified Extensible Firmware Interface (UEFI)* maintains the system hardware status and launches an installed operating system.

The BIOS Startup

The BIOS firmware had a simplistic menu interface that allowed you to change some settings to control how the system found hardware and define what device the BIOS should use to start the operating system.

One limitation of the original BIOS firmware was that it could read only one sector's worth of data from a hard drive into memory to run. That's not enough space to load an

entire operating system. To get around that limitation, most operating systems (including Linux and Microsoft Windows) split the boot process into two parts.

First, the BIOS runs a *bootloader* program, a small program that initializes the necessary hardware to find and run the full operating system program. It's often found at another location on the same hard drive but sometimes on a separate internal or external storage device.

The bootloader program usually has a configuration file, so you can tell it where to find the actual operating system file to run or even to produce a small menu allowing the user to boot between multiple operating systems.

To get things started, the BIOS must know where to find the bootloader program on an installed storage device. Most BIOS setups allow you to load the bootloader program from several locations:

- An internal hard drive
- An external hard drive
- A CD or DVD drive
- A USB memory stick
- An ISO file
- A network server using either NFS, HTTP, or FTP

When booting from a hard drive, you must designate the hard drive, and the partition on the hard drive, from which the BIOS should load the bootloader program. This is done by defining a *Master Boot Record (MBR)*.

The MBR is the first sector on the first hard drive partition on the system. There is only one MBR for the computer system. The BIOS looks for the MBR and reads the program stored there into memory. Since the bootloader program must fit in one sector, it must be very small, so it can't do too much. The bootloader program mainly points to the location of the actual operating system kernel file, stored in a boot sector of a separate partition installed on the system. There are no size limitations on the kernel boot file.



The bootloader program isn't required to point directly to an operating system kernel file; it can point to any type of program, including another bootloader program. You can create a primary bootloader program that points to a secondary bootloader program, which provides options to load multiple operating systems. This process is called *chainloading*.

The UEFI Startup

As operating systems became more complicated, it eventually became clear that a new boot method needed to be developed.

Intel created the *Extensible Firmware Interface (EFI)* in 1998 to address some of the limitations of BIOS. By 2005, the idea caught on with other vendors, and the Universal EFI (UEFI) specification was adopted as a standard. These days just about all IBM-compatible desktop and server systems utilize the UEFI firmware standard.

Instead of relying on a single boot sector on a hard drive to hold the bootloader program, UEFI specifies a special disk partition, called the *EFI System Partition (ESP)*, to store bootloader programs. This allows for any size of bootloader program, plus the ability to store multiple bootloader programs for multiple operating systems.

The ESP setup utilizes the old Microsoft File Allocation Table (FAT) filesystem to store the bootloader programs. On Linux systems, the ESP is typically mounted in the `/boot/efi` directory, and the bootloader files are typically stored using the `.efi` filename extension, such as `linux.efi`.

The UEFI firmware utilizes a built-in mini-bootloader (sometimes referred to as a *boot manager*) that allows you to configure which bootloader program file to launch.



Not all Linux distributions support the UEFI firmware. If you're using a UEFI system, ensure that the Linux distribution you select supports it. Also, many UEFI systems utilize a *secure boot* feature, which when enabled only loads bootloader programs digitally signed by a known signing certificate. Many of these systems only recognize Microsoft certificates, making it complicated, but not impossible, to boot a Linux system. To get around this most Linux distributions use chainloading to first load a shim bootloader, signed by Microsoft, which then points to the real Linux bootloader.

With UEFI, you need to register each individual bootloader file you want to appear at boot time in the boot manager interface menu. You can then select the bootloader to run each time you boot the system.

Once the firmware finds and runs the bootloader, its job is done. The bootloader step in the boot process can be somewhat complicated. The next section dives into covering that.

Linux Bootloaders

The bootloader program helps bridge the gap between the system firmware and the full Linux operating system kernel. In Linux there are several choices of bootloaders to use. However, the main bootloader programs that have been used by default in Linux distributions are as follows:

- Linux Loader (LILO)
- Grand Unified Bootloader (GRUB) Legacy
- GRUB2

In the original versions of Linux, the *Linux Loader (LILO)* bootloader was the only bootloader program available. It was extremely limited in what it could do, but it accomplished its purpose—loading the Linux kernel from the BIOS startup. The LILO configuration file is stored in a single file, `/etc/lilo.conf`, which defines the systems to boot.

Unfortunately, LILO doesn't work with UEFI systems, so it has limited use on modern systems and is quickly fading into history.

The first version of the GRUB bootloader (now called *GRUB Legacy*) was created in 1999 to provide a more robust and configurable bootloader to replace LILO. GRUB quickly became the default bootloader for all Linux distributions, whether they were run on BIOS or UEFI systems.

GRUB2 was created in 2005 as a total rewrite of the GRUB Legacy system. It supports advanced features, such as the ability to load hardware driver modules and using logic statements to dynamically alter the boot menu options, depending on conditions detected on the system (such as if an external hard drive is connected).



Since UEFI can load any size of bootloader program, it's now possible to load a Linux operating system kernel directly without a special bootloader program. This feature was incorporated in the Linux kernel starting with version 3.3.0. However, this method isn't common, as bootloader programs can provide more versatility in booting, especially when working with multiple operating systems.

The following sections walk through the basics of both the GRUB Legacy and GRUB2 bootloaders, which should cover just about every Linux distribution that you'll run into these days.

GRUB Legacy

The GRUB Legacy bootloader was designed to simplify the process of creating boot menus and passing options to kernels. GRUB Legacy allows you to select multiple kernels and/or operating systems using both a menu interface and an interactive shell. You configure the menu interface to provide options for each kernel or operating system you want to boot with. The interactive shell provides a way for you to customize boot commands on the fly.

Both the menu and the interactive shell utilize a set of commands that control features of the bootloader. The following sections walk through how to configure the GRUB Legacy bootloader, how to install it, and how to interact with it at boot time.

Configuring GRUB Legacy

When you use the GRUB Legacy interactive menu, you need to tell it what options to show using special GRUB *menu commands*.

The GRUB Legacy system stores the menu commands in a standard text configuration file called `menu.lst`. This file is stored in the `/boot/grub` folder (while not a requirement, some Linux distributions create a separate `/boot` partition on the hard drive). Red Hat–derived Linux distributions (such as CentOS and Fedora) use `grub.conf` instead of `menu.lst` for the configuration file.

The GRUB Legacy configuration file consists of two sections:

- Global definitions
- Operating system boot definitions

The global definitions section defines commands that control the overall operation of the GRUB Legacy boot menu. The global definitions must appear first in the configuration file. There are only a handful of global settings that you can make; Table 5.1 shows these settings.

TABLE 5.1 GRUB Legacy global commands

Setting	Description
color	Specifies the foreground and background colors to use in the boot menu
default	Defines the default menu option to select
fallback	A secondary menu selection to use if the default menu option fails
hiddenmenu	Doesn't display the menu selection options
splashimage	Points to an image file to use as the background for the boot menu
timeout	Specifies the amount of time to wait for a menu selection before using the default

For GRUB Legacy, to define a value for a command, you list the value as a command-line parameter:

```
default 0
timeout 10
color white/blue yellow/blue
```

The color command defines the color scheme for the menu. The first pair defines the foreground/background pair for normal menu entries, while the second pair defines the foreground/background pair for the selected menu entry.

After the global definitions, you place definitions for the individual operating systems that are installed on the system. Each operating system should have its own definition section. There are a lot of boot definition settings that you can use to customize how the bootloader finds the operating system kernel file. Fortunately, only a few commands are required to define the operating system. The ones to remember are listed here:

- title—The first line for each boot definition section; this is what appears in the boot menu.
- root—Defines the disk and partition where the GRUB /boot folder partition is on the system.

- `kernel`—Defines the kernel image file stored in the `/boot` folder to load.
- `initrd`—Defines the initial RAM disk file, which contains drivers necessary for the kernel to interact with the system hardware.
- `rootnoverify`—Defines non-Linux boot partitions, such as Windows.

The `root` command defines the hard drive and partition that contains the `/boot` folder for GRUB Legacy. Unfortunately, GRUB Legacy uses a somewhat odd way of referencing those values:

(*hddrive*, *partition*)

Also, unfortunately, GRUB Legacy doesn't refer to hard drives the way Linux does; it uses a number system to reference both disks and partitions, starting at 0 instead of 1. For example, to reference the first partition on the first hard drive on the system, you'd use (`hd0,0`). To reference the second partition on the first hard drive, you'd use (`hd0,1`).

The `initrd` command is another important feature in GRUB Legacy. It helps solve a problem that arises when using specialized hardware or filesystems as the root drive. The `initrd` command defines a file that's mounted by the kernel at boot time as a RAM disk, also called the *initrd*. The kernel can then load modules from the `initrd` RAM disk, which then allows it to access hardware or filesystems not compiled into the kernel itself. The `initrd` RAM disk file, located in the `/boot` directory, is called `initrd.img-kversion`, where *kversion* is the kernel version number.



If you install new hardware on your system that's required to be visible at boot time, you'll need to modify the `initrd` file. You can create a new `initrd` RAM disk image containing modules for the new hardware using the `mkinitrd` command in Red Hat–based systems. For Debian-based systems, the file is called `initramfs`, and you create it using the `mkinitramfs` command. Alternatively, you can use the `dracut` utility, which creates the `initramfs` image from a framework and copies files from the installed modules.

Listing 5.2 shows a sample GRUB configuration file that defines both a Windows and a Linux partition for booting.

Listing 5.2 Sample GRUB Legacy configuration file

```
default 0
timeout 10
color white/blue yellow/blue

title Ubuntu Linux
root (hd1,0)
kernel (hd1,0)/boot/vmlinuz
```

```
initrd /boot/initrd

title Windows
rootnoverify (hd0,0)
```

This example shows two boot options—one for an Ubuntu Linux system and one for a Windows system. The Ubuntu system is installed on the first partition of the second hard drive, whereas the Windows system is installed on the first partition of the first hard drive. The Linux boot selection specifies the kernel file to load as well as the `initrd` image file to load into memory.



You may have noticed that the kernel filename in the Listing 5.2 example was called `vmlinuz`. No, that's not a typo—the `z` at the end of the filename indicates that the kernel file is compressed using the `bzImage` compression method, a very common method in most Linux distributions. For kernel files that aren't compressed, the kernel image file is usually called `vmlinux`.

Installing GRUB Legacy

Once you build the GRUB Legacy configuration file, you must install the GRUB Legacy program in the MBR. The command to do this is `grub-install`.

The `grub-install` command uses a single parameter—the partition on which to install GRUB. You can specify the partition using either Linux or GRUB Legacy format. For example, to use Linux format you'd use

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

to install GRUB on the MBR of the first hard drive. To use GRUB Legacy format, you must enclose the hard drive format in quotes:

```
# grub-install '(hd0)'
```

If you're using the chainloading method and prefer to install a copy of GRUB Legacy on the boot sector of a partition instead of to the MBR of a hard drive, you must specify the partition, again using either Linux or GRUB format:

```
# grub-install /dev/sda1
# grub-install 'hd(0,0)'
```

You don't need to reinstall GRUB Legacy in the MBR after making changes to the configuration file. GRUB Legacy reads the configuration file each time it runs.

Interacting with GRUB Legacy

When you boot a system that uses the GRUB Legacy bootloader, you'll see a menu that shows the boot options you defined in the configuration file. If you wait for the timeout to expire, the default boot option will process. Alternatively, you can use the arrow keys to select one of the boot options and then press the Enter key to select it.

You can also edit boot options on the fly from the GRUB menu. First, arrow to the boot option you want to modify and then press the E key. Use the arrow key to move the cursor to the line you need to modify and then press the E key to edit it. Press the B key to boot the system using the new values. You can also press the C key at any time to enter an interactive shell mode, allowing you to submit commands on the fly.

GRUB2

Since the GRUB2 system was intended to be an improvement over GRUB Legacy, many of the features are the same, with a few twists. For example, the GRUB2 system changes the configuration file name to `grub.cfg` and stores it in the `/boot/grub/` folder (this allows you to have both GRUB Legacy and GRUB2 installed at the same time). Some Red Hat-based Linux distributions also make a symbolic link to this file in the `/etc/grub2.cfg` file for easy reference.

Configuring GRUB2

There are also a few changes to the commands used in GRUB2. For example, instead of the `title` command, GRUB2 uses the `menuentry` command, and you must also enclose each individual boot section with braces immediately following the `menuentry` command. Here's an example of a sample GRUB2 configuration file:

```
menuentry "Ubuntu Linux" {
    set root=(hd1,1)
    linux /boot/vmlinuz
    initrd /initrd
}
menuentry "Windows" {
    set root=(hd0,1)
}
```

Notice that GRUB2 uses the `set` command to assign values to the `root` keyword and an equal sign to assign the device. GRUB2 utilizes environment variables to configure settings instead of commands.

GRUB2 also changes the numbering system for partitions. While it still uses 0 for the first hard drive, the first partition is set to 1. So, to define the `/boot` folder on the first partition of the first hard drive, you now need to use

```
set root=hd(0,1)
```

In addition, notice that the `rootnoverify` and `kernel` commands are not used in GRUB2. Non-Linux boot options are defined the same as Linux boot options using the `root` environment variable, and you define the kernel location using the `linux` command.

The configuration process for GRUB2 is also different. While the `/boot/grub/grub.cfg` file is the configuration file that GRUB2 uses, you should never modify that file. Instead, there are separate configuration files stored in the `/etc/grub.d` folder. This allows

you (or the system) to create individual configuration files for each boot option installed on your system (for example, one configuration file for booting Linux and another for booting Windows).

For global commands, use the `/etc/default/grub` configuration file. The format for some of the global commands has changed from the GRUB Legacy commands, such as `GRUB_TIMEOUT` instead of just `timeout`.

Most Linux distributions generate the new `grub.cfg` configuration file automatically after certain events, such as upgrading the kernel. Usually the distribution will keep a boot option pointing to the old kernel file just in case the new one fails.

Installing GRUB2

Unlike with GRUB Legacy, you don't need to install GRUB2; you simply rebuild the main installation file by running the `grub2-mkconfig` program.

The `grub2-mkconfig` program reads configuration files stored in the `/etc/grub.d` folder and assembles the commands into the single `grub.cfg` configuration file.

You can update the configuration file manually by running the `grub2-mkconfig` command:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

Notice that you must use the `-o` command-line option to redirect the output of the `grub2-mkconfig` program to the `grub.cfg` configuration file. By default the `grub2-mkconfig` program just outputs the new configuration file commands to standard output.



Debian-based Linux distributions, such as Ubuntu, use GRUB2 but compile the programs to match the GRUB Legacy command names, such as the `grub-mkconfig` command instead of `grub2-mkconfig`. Needless to say, this can cause lots of confusion. You can tell which version of GRUB is being used by using the command `grub-mkconfig -V`.

There may be situations where you do need to reinstall GRUB2 on the boot drive. To do that, after creating the `grub.cfg` configuration file you can install it onto the primary hard disk using the `grub2-install` command:

```
# grub2-install /dev/sda
```



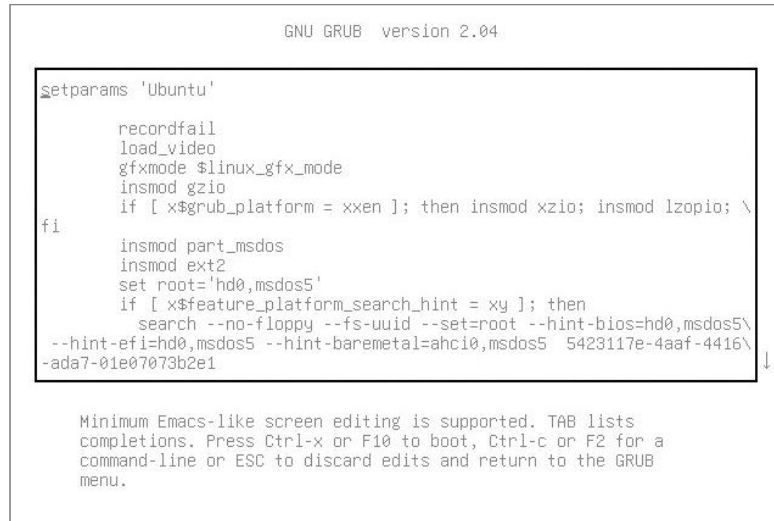
Instead of running both the `grub2-mkconfig` and `grub2-install` commands, you can use the `update-grub2` command (sometimes referred to as `grub2-update`), which is a front end that performs both operations from a single script.

Interacting with GRUB2

The GRUB2 bootloader produces a boot menu similar to the GRUB Legacy method. You can use arrow keys to switch between boot options, the `E` key to edit a boot entry, or the `C`

key to bring up the GRUB2 command line to submit interactive boot commands. Figure 5.1 shows editing an entry in the GRUB2 boot menu on an Ubuntu system.

FIGURE 5.1 Editing an Ubuntu GRUB2 menu entry



```

GNU GRUB  version 2.04

setparams 'Ubuntu'

    recordfail
    load_video
    gfxmode $linux_gfx_mode
    insmod gzio
    if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; \
fi
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos5'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos5 \
--hint-efi=hd0,msdos5 --hint-baremetal=ahci0,msdos5 5423117e-4aaf-4416 \
-ada7-01e07073b2e1
    ↓

Minimum Emacs-like screen editing is supported. TAB lists
completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
command-line or ESC to discard edits and return to the GRUB
menu.

```



Some graphical desktops (such as Ubuntu) hide the GRUB boot menu behind a graphical interface. Usually if you hold down the Shift key when the system first boots, that will display the GRUB boot menu.

Alternative Bootloaders

While GRUB Legacy and GRUB2 are the most popular Linux bootloader programs in use, you may run into a few others, depending on which Linux distributions you use.

The *Syslinux project* includes five separate bootloader programs that have special uses in Linux:

- **SYSLINUX:** A bootloader for systems that use the Microsoft FAT filesystem (popular for booting from USB memory sticks)
- **EXTLINUX:** A mini-bootloader for booting from an ext2, ext3, ext4, or btrfs filesystem
- **ISOLINUX:** A bootloader for booting from a LiveCD or LiveDVD
- **PXELINUX:** A bootloader for booting from a network server
- **MEMDISK:** A utility to boot older DOS operating systems from the other SYSLINUX bootloaders

The ISOLINUX bootloader is popular for distributions that release a LiveDVD version. The bootloader requires two files: `isolinux.bin`, which contains the bootloader program image, and `isolinux.cfg`, which contains the configuration settings.

The PXELINUX bootloader uses the *Pre-boot eXecution Environment (PXE)* standard, which defines how a network workstation can boot and load an operating system from a central network server. PXE uses DHCP to assign a network address to the workstation and BOOTP to load the bootloader image from the server. The network server must support the TFTP protocol to transfer the boot image file to the workstation.

To utilize PXELINUX, the TFTP server needs to have the PXELINUX bootloader program, stored as `/tftpboot/pxelinux.0`, available for the workstations to download. Each workstation must also have a configuration file available in the `/tftpboot/pxelinux.cfg` directory. The files are named based on the MAC address of the workstation and contain specific configuration settings required for that workstation.



Although PXE was designed to use TFTP to load the boot image, it has been modified to also load the bootloader image stored on a network server using NFS, HTTP, or even FTP.

System Recovery

There's nothing worse than starting up your Linux system and not getting a login prompt. Plenty of things can go wrong in the Linux startup process, but most issues come down to two categories:

- Kernel failures
- Drive failures

The following sections walk through some standard troubleshooting practices you can follow to attempt to recover a Linux system that fails to boot.

Kernel Failures

Kernel failures are when the Linux kernel stops running in memory, causing the Linux system to crash. This is commonly referred to as a *kernel panic*. Kernel panics often are a result of a software change, such as installing a new kernel without the appropriate module or library changes or starting (or stopping) a program at a new runlevel. Often these types of boot errors can be fixed by starting the system using an alternative method and editing the necessary files to change the system.

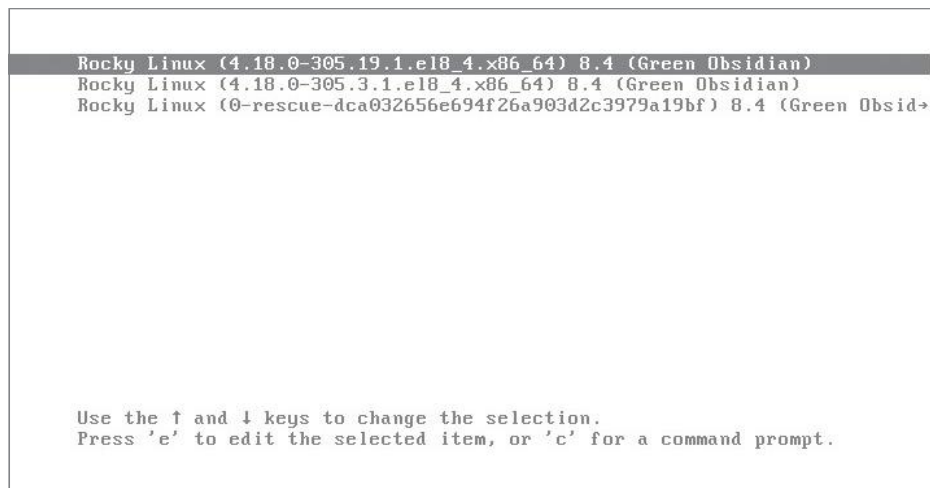
Selecting Previous Kernels at Boot

One of the biggest culprits to a failed boot is when you upgrade the Linux kernel, either on your own or from a packaged distribution upgrade. When you install a new kernel file, it's always a good idea to leave the old kernel file in place and create an additional entry in the GRUB boot menu to point to the new kernel.

By creating multiple kernel entries in the GRUB boot menu, you can select which kernel version to boot. If the new kernel fails to boot properly, you can reboot and select the older kernel version.

Most Linux distributions do this automatically when adding a new kernel, keeping the most recent older kernel available in the boot menu, as shown in Figure 5.2.

FIGURE 5.2 The Rocky Linux Grub boot menu with multiple kernel options



Single-User Mode

At times you may need to perform some type of system maintenance, such as adding a new hardware module or library file to get the system to boot properly. In these situations, you want the system to boot up without allowing multiple users to connect, especially in a server environment. This is called *single-user mode*.

The GRUB menu allows you to start the system in single-user mode by adding the `single` command to the `linux` line in the boot menu commands. To get there, press the `E` key on the boot option in the GRUB boot menu.

When you add the `single` command, the system will boot into `runlevel 1` (or for systems using the Systemd startup method, the `runlevel-1` target), which creates a single login for the root user account. Once you log in as the root user account, you can modify the appropriate modules, init scripts, or GRUB boot menu options necessary to get your system started correctly.

Passing Kernel Parameters

Besides the single-user mode trick, you can add other kernel parameters to the `linux` command in the GRUB boot menu. The kernel accepts parameters that alter the hardware modules it activates or the hardware settings it looks for with specific devices (this is

especially true for sound and network cards). You can specify the different hardware settings as additional parameters to the kernel in the `linux` command and then boot from that entry in the GRUB menu.

Root Drive Failure

Perhaps the worst feeling for a Linux system administrator is seeing that the bootloader can't read the root drive device. However, this type of error may not be fatal, because it is sometimes possible to recover from a corrupted root drive.

Using a Rescue Disk

Many Linux distributions provide a *rescue disk* for when fatal disk errors occur. The rescue disk usually boots either from the CD drive or as a USB stick and loads a small Linux system into memory. Since the Linux system runs entirely in memory, it can leave all of the workstation hard drives free for examination and repair. From the system command-line prompt, you can perform some diagnostic and repair tasks on your system hard drives.

The tool of choice for checking and fixing hard drive errors is the `fsck` command. The `fsck` command isn't a program; it's an alias for a family of commands specific to different types of filesystems (such as `ext2`, `ext3`, and `ext4`). You need to run the `fsck` command against the device name of the partition that contains the root directory of your Linux system. For example, if the root directory is on the `/dev/sda1` partition, you'd run the following command:

```
# fsck /dev/sda1
```

The `fsck` command will examine the inode table along with the file blocks stored on the hard drive and attempt to reconcile them. If any errors occur, you will be prompted on whether to repair them or not. If there are a lot of errors on the partition, you can add the `-y` parameter to automatically answer yes to all the repair questions. After a successful repair, it's a good idea to run the `fsck` command once more to ensure that all errors have been found and corrected. Continue running the `fsck` command until you get a clean run with no errors.

Mounting a Root Drive

When the `fsck` repair is complete, you can test the repaired partition by mounting it into the virtual directory created in memory. Just use the `mount` command to mount it to an available mount directory:

```
# mount /dev/sda1 /media
```

You can examine the filesystem stored in the partition to ensure that it's not corrupted. Before rebooting, you should unmount the partition using the `umount` command:

```
# umount /dev/sda1
```

After successfully unmounting the partition, you can reboot your Linux system using the standard bootloader and attempt to boot using the standard kernel and runlevels.

While booting Linux in single-user mode isn't used all that often, it comes in handy to know how to do it "just in case." Exercise 5.1 walks through the steps to help you get some practice in booting up a Linux system in single user mode.

Using Rescue Mode

This exercise will demonstrate how to start your Linux distribution in single-user mode to examine filesystems and configurations without performing a complete bootup. To use single-user mode, follow these steps:

1. First, start your Linux distribution as normal, and log in on the standard login prompt (either the graphical desktop or the command-line login) as your normal user account.
 2. Type **runlevel** to determine the default runlevel for your system. The first character returned refers to the previous runlevel (N denotes no previous runlevel since the system booted). The second character is the current runlevel. This is likely to be 2 on Debian-based systems or 3 on command-line Red Hat-based systems or 5 on graphical desktop Red Hat-based systems.
 3. Now reboot your system, and press an arrow key when the GRUB2 menu appears to stop the countdown timer. If you're using a Linux distribution that hides the GRUB2 menu (such as Ubuntu), hold down the Shift key when the system boots to display the GRUB2 menu.
 4. At the GRUB2 menu, use the arrow keys to go to the default menu entry (usually the first entry in the list) and then press the **E** key. This takes GRUB2 into edit mode.
 5. Look for either the `linux` or `linux16` menu command lines. These define the kernel used to start the session.
 6. Go to the end of the `linux` or the `linux16` line, and add the word `single`. Press **Ctrl+X** to temporarily save the change and start your system using that menu entry.
 7. The Linux system will boot into single-user mode. Depending on your Linux distribution, it may prompt you to enter the root user account or to press **Ctrl+D** to continue with the normal boot. Enter the root user account password to enter single-user mode.
 8. Now you are at the root user command prompt. Enter the command **runlevel** to view the current runlevel. It should show runlevel 1. From here you can modify configuration files, check filesystems, and change user accounts.
 9. Reboot the system by typing **reboot**.
 10. You should return to the standard boot process and GRUB2 menu options as before. Select the standard GRUB2 menu option to boot your system and then log in.
 11. At a command-line prompt, type **runlevel** to ensure that you are back to the normal default runlevel for your Linux system.
-

Summary

Although Linux distributions are designed to boot without any user intervention, it helps to know the Linux boot process in case anything does go wrong. Most Linux systems use either the GRUB Legacy or GRUB2 bootloader program. These programs both reside in the BIOS Master Boot Record or in the ESP partition on UEFI systems. The bootloader loads the Linux kernel program, which then runs the SysV init or Systemd programs to start individual background programs required for the Linux system.

No discussion on Linux startup is complete without examining system recovery methods. If your Linux system fails to boot, the most likely cause is either a kernel issue or a root device issue. For kernel issues, you can often modify the GRUB menu to add additional kernel parameters, or even boot from an older version of the kernel. For root drive issues you can try to boot from a rescue mode into a version of Linux running in memory and then use the `fsck` command to repair a damaged root drive.

Exam Essentials

Describe the Linux boot process. The BIOS or UEFI starts a bootloader program from the Master Boot Record, which is usually the Linux GRUB Legacy or GRUB2 program. The bootloader program loads the Linux kernel into memory, which in turn looks for the `init` program to run. The `init` program starts individual application programs and starts either the command-line terminals or the graphical desktop manager.

Describe the Linux GRUB Legacy and GRUB2 bootloaders. The GRUB Legacy bootloader stores files in the `/boot/grub` folder and uses the `menu.lst` or `grub.conf` configuration file to define commands used at boot time. The commands can create a boot menu, allowing you to select between multiple boot locations, options, or features. You must use the `grub-install` program to install the GRUB Legacy bootloader program into the Master Boot Record. The GRUB2 bootloader also stores files in the `/boot/grub` folder, but it uses the `grub.cfg` configuration file to define the menu commands. You don't edit the `grub.cfg` file directly but instead store files in the `/etc/default/grub` file or individual configuration files in the `/etc/grub.d` folder. Run the `grub-mkconfig` program to generate the GRUB2 configuration from the configuration files and then redirect the output to the `/etc/grub.cfg` file.

Describe alternative Linux bootloaders. The LILO bootloader is used on older Linux systems. It uses the `/etc/lilo.conf` configuration file to define the boot options. The Syslinux project has created the most popular alternative Linux bootloaders. The SYSLINUX bootloader provides a bootloader that runs on FAT filesystems, such as floppy disks and USB memory sticks. The ISOLINUX bootloader is popular on LiveCD distributions, as it can boot from a CD or DVD. It stores the bootloader program in the `isolinux.bin` file and configuration settings in the `isolinux.cfg` file. The PXELINUX bootloader program

allows a network workstation to boot from a network server. The server must contain the `pxelinux.0` image file along with the `pxelinux.cfg` directory, which contains separate configuration files for each workstation. The EXTLINUX bootloader is a small bootloader program that can be used on smaller embedded Linux systems.

Describe how to recover from a kernel panic. The GRUB bootloaders provide you with options that can help if your Linux system fails to boot or stops due to a kernel panic issue. You can press the E key at the GRUB boot menu to edit any boot menu entry, then add any additional kernel parameters, such as placing the system in single-user mode. You can also use a rescue disk to boot Linux into memory, then use the `fsck` command to repair any corrupted hard drives, and finally use the `mount` command to mount them to examine the files.

Review Questions

1. What program does the workstation firmware start at boot time?
 - A. A bootloader
 - B. The fsck program
 - C. The Windows OS
 - D. The mount command
 - E. The mkinitrd program
2. Where does the firmware first look for a Linux bootloader program?
 - A. The /boot/grub folder
 - B. The Master Boot Record (MBR)
 - C. The /var/log folder
 - D. A boot partition
 - E. The /etc folder
3. The _____ command allows us to examine the most recent boot messages.
 - A. fsck
 - B. init
 - C. mount
 - D. dmesg
 - E. mkinitrd
4. What folder do most Linux distributions use to store boot logs?
 - A. /etc
 - B. /var/messages
 - C. /var/log
 - D. /boot
 - E. /proc
5. Where does the workstation BIOS attempt to find a bootloader program? (Choose all that apply.)
 - A. An internal hard drive
 - B. An external hard drive
 - C. A DVD drive
 - D. A USB memory stick
 - E. A network server

6. Where is the Master Boot Record located? (Choose all that apply.)
 - A. The first sector of the first hard drive on the system
 - B. The boot partition of any hard drive on the system
 - C. The last sector of the first hard drive on the system
 - D. Any sector on any hard drive on the system
 - E. The first sector of the second hard drive on the system
7. The EFI System Partition (ESP) is stored in the _____ directory on Linux systems.
 - A. /boot
 - B. /etc
 - C. /var
 - D. /boot/efi
 - E. /boot/grub
8. What file extension do UEFI bootloader files use?
 - A. .cfg
 - B. .uefi
 - C. .lst
 - D. .conf
 - E. .efi
9. Which was the first bootloader program used in Linux?
 - A. GRUB Legacy
 - B. LILO
 - C. GRUB2
 - D. SYSLINUX
 - E. ISOLINUX
10. Where are the GRUB Legacy configuration files stored?
 - A. /boot/grub
 - B. /boot/efi
 - C. /etc
 - D. /var
 - E. /proc
11. Where are GRUB2 configuration files stored? (Choose all that apply.)
 - A. /proc
 - B. /etc/grub.d
 - C. /boot/grub
 - D. /boot/efi
 - E. /var

12. You must run the _____ command to generate the GRUB2 `grub.cfg` configuration file.
 - A. `mkinitrd`
 - B. `mkinitramfs`
 - C. `grub-mkconfig`
 - D. `grub-install`
 - E. `fsck`
13. What command must you run to save changes to a GRUB Legacy boot menu?
 - A. `mkinitrd`
 - B. `mkinitramfs`
 - C. `grub-mkconfig`
 - D. `grub-install`
 - E. `fsck`
14. The _____ firmware method has replaced BIOS on most modern IBM-compatible computers.
 - A. FTP
 - B. UEFI
 - C. PXE
 - D. NFS
 - E. HTTPS
15. What memory area does Linux use to store boot messages?
 - A. BIOS
 - B. The GRUB bootloader
 - C. The MBR
 - D. The `initrd` RAM disk
 - E. The kernel ring buffer
16. What command parameter would you add to the end of the GRUB2 `linux` command to force a Linux system to start in single-user mode?
 - A. `single`
 - B. `fsck`
 - C. `mkinitrd`
 - D. `mkinitramfs`
 - E. `dmesg`
17. What is the term commonly used for when the Linux system halts due to a system error?
 - A. Kernel panic
 - B. Kernel ring buffer
 - C. `initrd` RAM disk
 - D. Bootloader
 - E. Firmware

18. The _____ command generates the GRUB2 configuration used for booting.
- A. mkinitrd
 - B. grub-mkconfig
 - C. grub-install
 - D. mkinitramfs
 - E. dmesg
19. What program allows you to fix corrupted hard drive partitions?
- A. mount
 - B. umount
 - C. fsck
 - D. dmesg
 - E. mkinitrd
20. Which command allows you to append a partition to the virtual directory on a running Linux system?
- A. mount
 - B. umount
 - C. fsck
 - D. dmesg
 - E. mkinitramfs