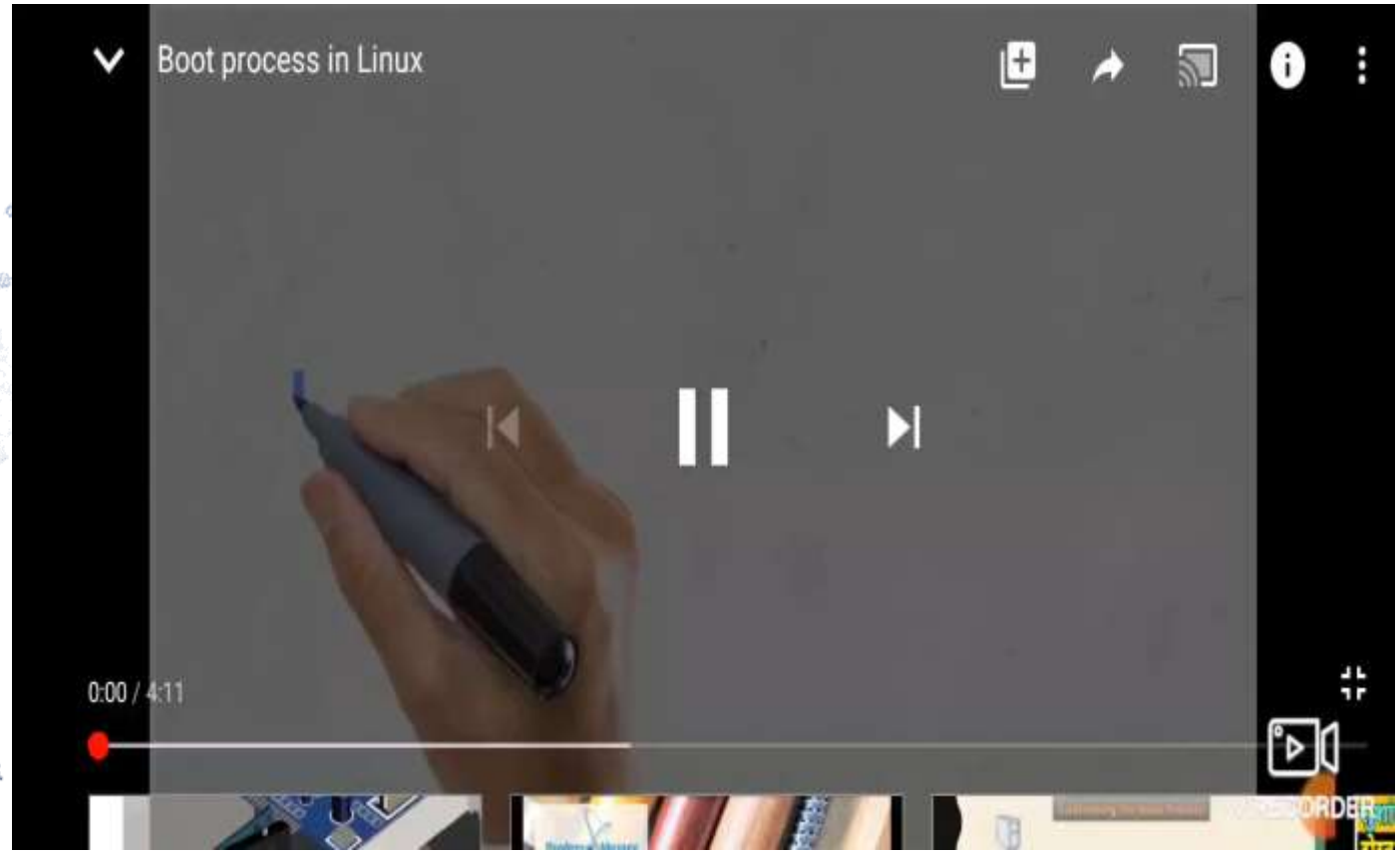
Booting and shutting down processes



BOOT LOADERS

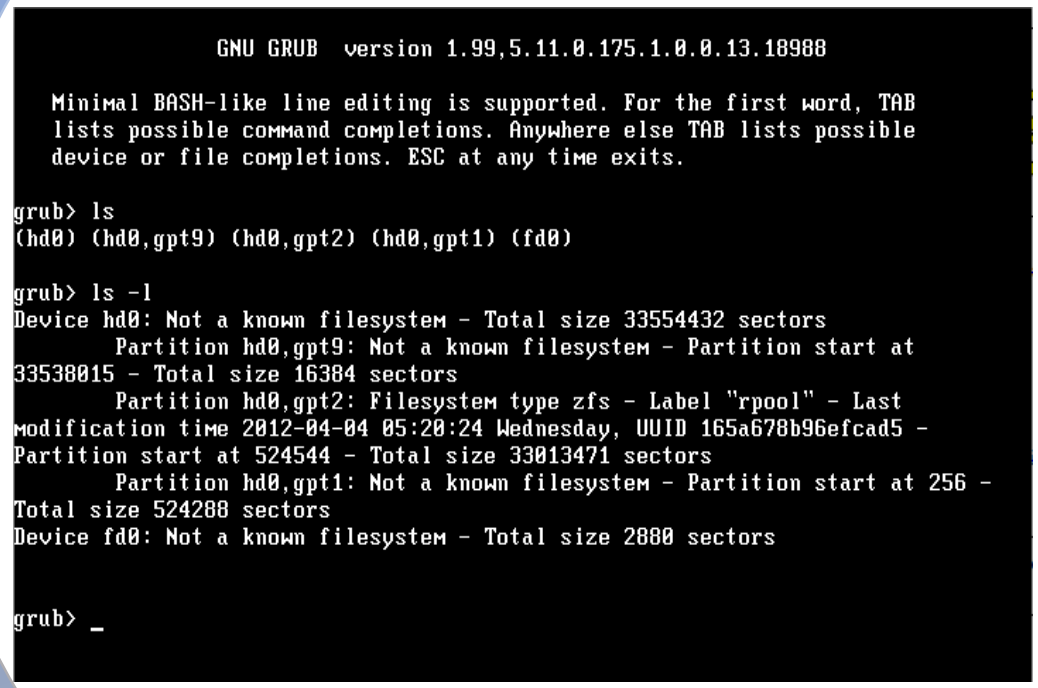
- For any operating system to boot on standard PC hardware, you need what is called a ***boot loader***.
- If you have only dealt with Windows on a PC, you have probably never needed to interact directly with a boot loader.
- The boot loader is the first software program that runs when a computer starts.
- It is responsible for handing over control of the system to the operating system.
- Typically, the boot loader will reside in the Master Boot Record (MBR) of the disk, and it knows how to get the operating system up and running.
- The main choices that come with Linux distributions are GRUB (the Grand Unified Bootloader) and LILO (Linux Loader).

GRUB

- Most modern Linux distributions use ***GRUB*** as the default boot loader during installation.
- GRUB is the default boot loader for Fedora, Red Hat Enterprise Linux (RHEL), OpenSUSE, Mandrake, Ubuntu, and a host of other Linux distributions.
- GRUB aims to be compliant with the Multiboot Specification and offers many features.
- The GRUB boot process happens in stages. Each stage is taken care of by special GRUB image files, with each preceding stage helping the next stage along.
- Two of the stages are essential, and any of the other stages are optional and dependent on the particular system setup.
    1. **Stage 1**
    - The image file used in this stage is essential and is used for booting up GRUB in the first place. It is usually embedded in the MBR of a disk or in the boot sector of a partition. The file used in this stage is appropriately named **stage1**. A Stage 1 image can next either load Stage 1.5 or load Stage 2 directly.
    2. **Stage 2**
    - The Stage 2 image is actually consist of two types of images: the intermediate (optional image) and the actual **stage2** image file. To further blur things, the optional images are called Stage 1.5. The Stage 1.5 images serve as a bridge between Stage 1 and Stage 2. The Stage 1.5 images are file system–specific; that is, they understand the semantics of one file system or the other.
    - The Stage 1.5 images have names of the form—*x*_stage_1_5 —where *x* can be a file system of type e2fs, reiserfs, fat, jfs, minix, xfs, etc.
- For example, the Stage 1.5 image that will be required to load an operating system (OS) that resides on a File Allocation Table (FAT) file system will have a name like **fat_ stage1_5**.
- The Stage 1.5 images allow GRUB to access several file systems. When used, the Stage 1.5 image helps to locate the Stage 2 image as a file within the file system.
- Next comes the actual **stage2** image. It is the core of GRUB. It contains the actual code to load the kernel that boots the OS, it displays the boot menu, and it also contains the GRUB shell from which GRUB commands can be entered. The GRUB shell is interactive and helps to make GRUB flexible. For example, the shell can be used to boot items that are not currently listed in GRUB's boot menu or to bootstrap the OS from an alternate supported medium.
- Other types of Stage 2 images are the **stage2_eltorito** image, the **nbgrub** image, and the **pxegrub** image. The **stage2_eltorito** image is a boot image for CD-ROMs. The **nbgrub** and **pxegrub** images are both network-type boot images that can be used to bootstrap a system over the network (using Bootstrap Protocol [BOOTP], Dynamic Host Configuration Protocol [DHCP], Preboot Execution Environment [PXE], Etherboot, or the like). A quick listing of the contents of the **/boot/grub** directory of most Linux distributions will show some of the GRUB images.

Conventions Used in GRUB

- GRUB has its own special way of referring to devices (CD-ROM drives, floppy drives, hard disk drives, etc.).
- The device name has to be enclosed in parentheses: “( )”.
- GRUB starts numbering its devices and partitions from zero, not from one.
- Therefore, GRUB would refer to the master Integrated Drive Electronics (IDE) hard drive on the primary IDE controller as (hd0), where “hd” means “hard disk” drive and the number zero means it is the primary IDE master.
- In the same vein, GRUB will refer to the fourth partition on the fourth hard disk (i.e., the slave on the secondary IDE controller) as “(hd3,3).”
- To refer to the whole floppy disk in GRUB would mean “(fd0)”—where “fd” means “floppy disk.”



Installing GRUB

- Most Linux distributions will give you a choice to ***install and configure the boot loader*** during the initial operating system installation.
- Thus, you wouldn't normally need to manually install GRUB during normal system use.
- However, there are times, either by accident or by design, that you don't have a boot loader.
- It could be by accident if you, for example, accidentally overwrite your boot sector or if another operating system accidentally wipes out GRUB.
- It could be by design if, for example, you want to set up your system to dual-boot with another operating system (Windows or another Linux distribution).
- This section will walk you through getting GRUB installed (or reinstalled) on your system. This can be achieved in several ways.
- You can do it the easy way from within the running OS using the ***grub-install*** utility or using GRUB's native command-line interface. You can get to this interface using what is called a GRUB boot floppy, using a GRUB boot CD, or from a system that has the GRUB software installed.
- **Backing Up the MBR**
- Before proceeding with the exercises that follow, it is a good idea to make a backup of your current "known good" MBR. It is easy to do this using the **dd** command. Since the MBR of a PC's hard disk resides in the first 512 bytes of the disk, you can easily copy the first 512 bytes to a file (or to a floppy disk) by typing

```
[root@fedora-serverA ~]# dd if=/dev/sda of=/tmp/COPY_OF_MBR bs=512 count=1
1+0 records in
1+0 records out
```

- This command will save the MBR into a file called COPY_OF_MBR under the **/tmp** directory.

Creating a Boot/Rescue CD

- Another precautionary measure to take before performing any operation that can render a system unbootable is to *create a rescue CD*. The CD can then be *used to boot the system in case of accidents*
- The boot CD is *system-specific* and is automatically built from current information extracted from your system.
- We will use the **mkbootdisk** command to generate an ISO image that can be burned to a blank CD-ROM.
- If you don't have the **mkbootdisk** utility already installed, you can use **yum** to install it on a Fedora system by typing

  yum install mkbootdisk.

- To generate an ISO image named BOOT-CD.iso for your running kernel and save the image file under the **/tmp** directory, type

  [root@fedora-serverA ~]# **mkbootdisk --device /tmp/BOOT-CD.iso --iso `uname -r`**

- You will next need to find a way to burn/write the created CD image onto a blank CD. If you have a CD burner installed on the Linux box, you can use the **cdrecord** utility to achieve this by issuing the command

  [root@fedora-serverA ~]# **cdrecord speed=4 -eject --dev=/dev/sr0 /tmp/BOOT-CD.iso**

- You should then date and label the disc accordingly with a descriptive name.

Installing GRUB from the GRUB Shell

In this section, you will learn how to install GRUB natively using GRUB's command shell from inside the running Linux operating system.

You will normally go this route if, for example, you currently have another type of boot loader (such as LILO or the NT Loader, NTLDR) but you wish to replace or overwrite that boot loader with GRUB.

1. *Launch GRUB's shell by issuing the **grub** command. Type*

```
[root@fedora-serverA ~]# grub
```

To display GRUB's current root device. Type

```
grub> root
(fd0): Filesystem type unknown, partition type 0x0
```

The output shows that GRUB will, by default, use the first floppy disk drive (fd0) as its root device, unless you tell it otherwise.

2. Set GRUB's root device to the partition that contains the boot directory on the local hard disk. Type

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

NOTE The boot directory may or may not be on the same partition that houses the root (/) directory. During the OS installation on our sample system, the **/boot** directory was stored on the **/dev/sda1** partition, and hence, we use the GRUB (hd0,0) device.

3. Make sure that the **stage1** image can be found on the root device. Type

```
grub> find /grub/stage1
(hd0,0)
```

The output means that the **stage1** image was located on the (hd0,0) device.

4. Finally, install the GRUB boot loader directly on the MBR of the hard disk. Type

```
grub> setup (hd0)
Checking if "/boot/grub/stage1" exists... no
Checking if "/grub/stage1" exists... yes
Checking if "/grub/stage2" exists... yes
Checking if "/grub/e2fs_stage1_5" exists... yes
Running "embed /grub/e2fs_stage1_5 (hd0)"... 16 sectors are embedded.
succeeded
Running "install /grub/stage1 (hd0) (hd0)1+16 p (hd0,0)/grub/stage2 /
grub/grub.conf"... succeeded
Done.
```

5. Quit the GRUB shell. Type

```
grub> quit
```

You are done. But you should note that you really didn't make any serious changes to the system, because you simply reinstalled GRUB to the MBR (where it used to be).

You would normally reboot at this point to make sure that everything is working as it should.

The GRUB Boot Floppy

- Let's create a GRUB floppy. This will allow you to boot the system using the floppy disk and use GRUB to write or install itself to the MBR.
- This is especially useful if your system does not currently have a boot loader installed but you have access to another system that has GRUB installed.
- The general idea behind using a GRUB boot floppy is that it is assumed that you currently have a system with an unbootable, corrupt, or unwanted boot loader—and since the system cannot be booted by itself from the hard disk, you need another medium to bootstrap the system with.
- For this, you can use a GRUB floppy disk or a GRUB CD.
- You want *any* means by which you can gain access to the GRUB shell so that you can install GRUB into the MBR and then boot the OS.
- You need to first locate the GRUB images, located by default in **/usr/share/grub/i386-redhat/** directory on a Fedora/Red Hat system (OpenSuSE stores the GRUB image files in the **/usr/lib/grub/** directory, and the images are stored under **/usr/lib/grub/i386-pc/** on an Ubuntu system).
- Use the **dd** command to write the **stage1** and **stage2** images to the floppy disk.

1. Change to the directory that contains the GRUB images on your system. Type

```
[root@fedora-serverA ~]# cd /usr/share/grub/i386-redhat/
```

2. Write the file **stage1** to the first 512 bytes of the floppy disk. Type

```
[root@fedora-serverA i386-redhat]# dd if=stage1 of=/dev/fd0 bs=512 count=1
1+0 records in
1+0 records out
```

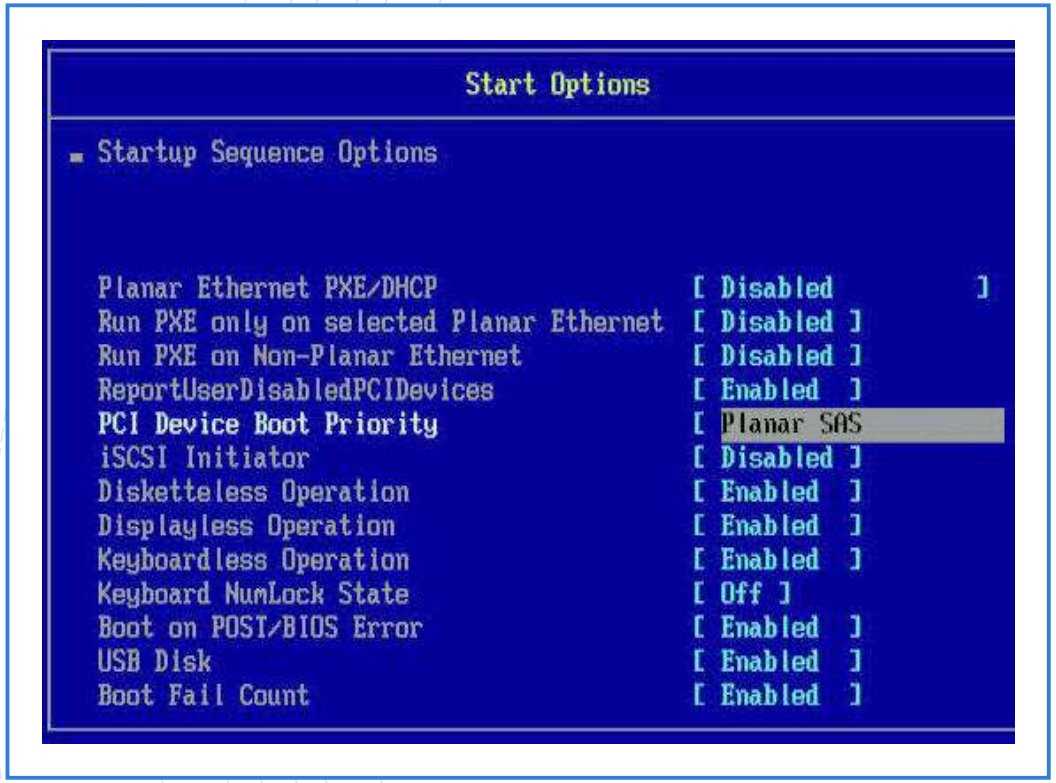
3. Write the **stage2** image right after the first image. Type

```
[root@fedora-serverA i386-redhat]# dd if=stage2 of=/dev/fd0 bs=512
seek=1
202+1 records in
202+1 records out
```

TIP You can also use the **cat** command to do the same thing as in the last two steps in a single shot. The command to do this will be

```
[root@fedora-serverA i386-redhat]# cat stage1 stage2 > /dev/fd0
```

Your GRUB floppy is now ready. You can now boot off of this floppy so that you can install the GRUB boot loader.



Installing GRUB on the MBR Using a GRUB Floppy

- Make sure that the GRUB floppy you created is inserted into the floppy disk drive.
- Reboot the system and use the floppy as your boot medium (adjust the BIOS settings if necessary).
- After the system has booted off the GRUB floppy, you will be presented with a **grub>** prompt.
- Set the root device for GRUB to your boot partition (or the partition that contains the **/boot** directory).
- On our sample system, the **/boot** directory resides on the **/dev/sda1 (hd0,0)** partition.
- To do this, type the following command:

grub> **root (hd0,0)**

- Now you can write GRUB to the MBR by using the setup command:

grub> **setup (hd0)**

- That's it, you are done. You may now reboot the system *without* the GRUB floppy.
- This is a good way to let GRUB reclaim management of the MBR, if it had previously been overwritten by another boot manager.

Configuring GRUB

- Since you only have to install GRUB once on the MBR or partition of your choice, you have the luxury of simply editing a text file, (**/boot/grub/menu.1st**), in order to make changes to your boot loader.
- When you are done editing this file, you can reboot and select any new kernel that you added to the configuration.
- The configuration file looks like the following (please note that line numbers 1–16 have been added to the output to aid readability):

```
[root@fedora-serverA ~]# cat /boot/grub/menu.lst
1) # grub.conf generated by anaconda
   # Note that you do not have to re-run grub after making changes to this file
3) # NOTICE: You have a /boot partition. This means that
4) # all kernel and initrd paths are relative to /boot/, eg.
5) # root (hd0,0)
6) # kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
7) # initrd /initrd-version.img
8) #boot=/dev/sda
9) default=0
10) timeout=5
11) splashimage=(hd0,0)/grub/splash.xpm.gz
12) hiddenmenu
13) title Fedora (2.6.25-14.fc9.i686)
14) root (hd0,0)
15) kernel /vmlinuz-2.6.25-14.fc9.i686 ro root=UUID=7db5-4c27 rhgb quiet
16) initrd /initrd-2.6.25-14.fc9.i686.img
```

The entries in the preceding sample configuration file for GRUB are discussed here:

▼ Lines 1–8 All lines that begin with the pound sign (#) are ***comments*** and are ignored.

■ Line 9, **default** This directive tells GRUB which entry to automatically boot. The numbering starts from zero. The preceding sample file contains only one entry—the entry titled **Fedora (2.6.25-14.fc9.i686)**.

■ Line 10, **timeout** This means that GRUB will automatically boot the default entry after five seconds. This can be interrupted by pressing any key on the keyboard before the counter runs out.

■ Line 11, **splashimage** This line specifies the name and location of an image file to be displayed at the boot menu. This is optional and can be any custom image that fits GRUB's specifications.

■ Line 12, **hiddenmenu** This entry hides the usual GRUB menu. It is an optional entry.

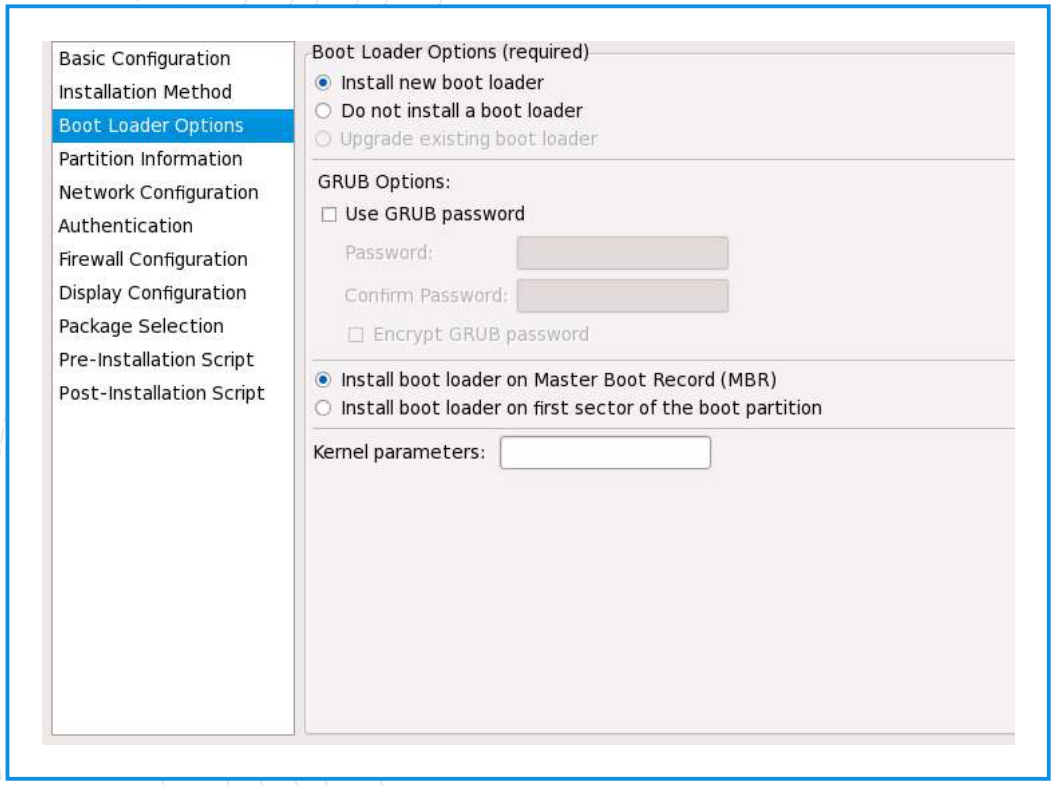
■ Line 13, **title** This is used to display a short title or description for the following entry it defines. The title field marks the beginning of a new boot entry in GRUB.

■ Line 14, **root** You should notice from the preceding listing that GRUB still maintains its device-naming convention (e.g., (hd0,0) instead of the usual Linux /dev/sda1).

■ Line 15, **kernel** Used for specifying the path to a kernel image. The first argument is the path to the kernel image in a partition. Any other arguments are passed to the kernel as boot parameters.

Note that the path names are relative to the **/boot** directory, so, for example, instead of specifying the path to the kernel to be "/boot/vmlinuz-2.6.25-14.fc9.i686," GRUB's configuration file references this path as "/vmlinuz-2.6.25-14.fc9.i686."

▲ Line 16, **initrd** The **initrd** option allows you to load kernel modules from an image, not the modules from **/lib/modules**. See the GRUB info pages, available through the **info** command, for more information on the configuration options.



Adding a New Kernel to Boot with GRUB

- In this section, you will learn ***how to manually add a new boot entry to GRUB's configuration file.***
- If you are compiling and installing a new kernel by hand, you will need to do this so that you can boot into the new kernel to test it or use it.
- If, on the other hand, you are installing or upgrading the Linux kernel using a prepackaged Red Hat Package Manager (RPM), this is usually automatically done for you.
- Because you don't have any new Linux kernel to install on the system, you will only add a dummy entry to GRUB's configuration file in this exercise.
- The new entry will not do anything useful—it is only being done for illustration purposes.
- Here's a summary of what we will be walking you through:
    1. You will make a copy of the current default kernel that your system uses, and call the copy **duplicate-kernel**.
    2. You will also make a copy of the corresponding **initrd** image for the kernel, and name the copy **duplicate-initrd**.
    3. Both files should be saved into the **/boot** directory.
    4. You will then create an entry for the supposedly new kernel and give it a descriptive title, such as "**The Duplicate Kernel**."

In addition to the preceding boot entry, you will create another entry that does nothing more than change the foreground and background colors of GRUB's boot menu.

Let's begin:

1. Change your current working directory to the **/boot** directory. Type

```
[root@fedora-serverA ~]# cd /boot
```

2. Make a copy of your current kernel, and name the copy **duplicate-kernel**. Type

```
[root@fedora-serverA boot]# cp vmlinuz-2.6.25-14.fc9.i686 duplicate-kernel
```

3. Make a copy of the corresponding **initrd** image, and name the copy **duplicateinitrd**.

Type

```
[root@fedora-serverA boot]# cp initrd-2.6.25-14.fc9.i686.img duplicate-initrd.img
```

4. Create an entry for the new pseudo-kernels in the **/boot/grub/menu.1st** configuration file, using any text editor you are comfortable with (the **vim** editor is used in this example). Type the following text at the end of the file:

```
title The Duplicate Kernel
color yellow/black
root (hd0,0)
kernel /duplicate-kernel ro root=UUID=7db5-4c27
initrd /duplicate-initrd.img
```

NOTE The value of "UUID" used above was obtained from the existing entry in the **menu.1st** file that we are duplicating. The exact partition or volume on which the root file system (**/**) resides can also be specified; for example, we could have the kernel entry in the **menu.1st** file as kernel /vmlinuz-2.6.25-14.fc9.i686 ro root=/dev/VolGroup00/LogVol00 rhgb quiet

5. Create another entry that will change the foreground and background colors of the menu when selected. The menu colors will be changed to yellow and black when this entry is selected. Enter the following text at the end of the file (beneath the entry you created in the preceding step):

```
title The change color entry
color yellow/black
```

6. Comment out the **splashimage** entry at the top of the file. The presence of the splash image will prevent your new custom foreground and background colors from displaying properly. The commented-out entry for the splash image will look like this:

```
# splashimage=(hd0,0)/grub/splash.xpm.gz
```

7. Finally, comment out the **hiddenmenu** entry from the file so that the Boot menu will appear, showing your new entries instead of being hidden. The commented out entry should look like

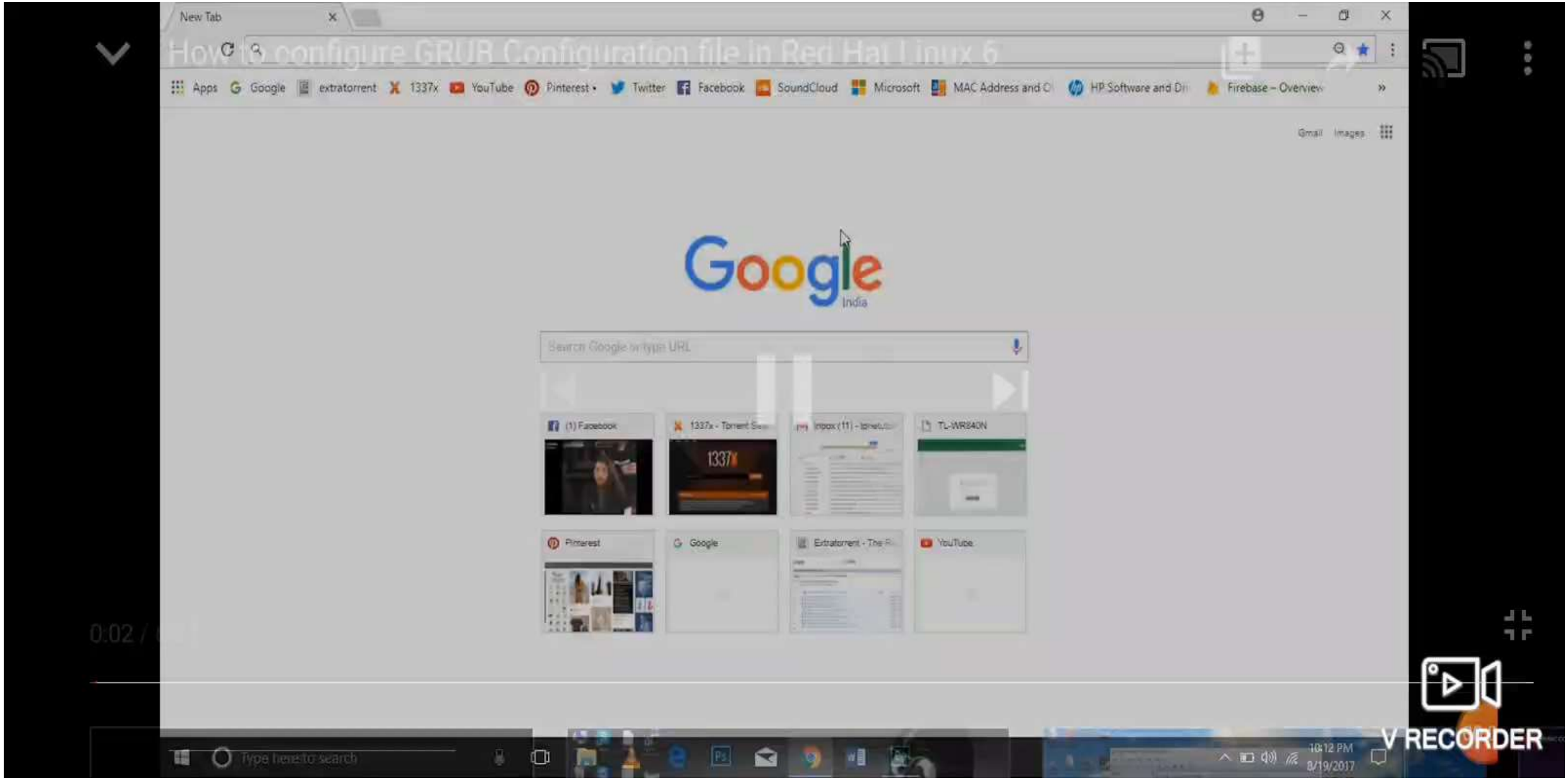
```
#hiddenmenu
```

8. Save the changes you made to the file, and reboot the system.
The final **/boot/grub/menu.1st** file (with some of the comment fields removed) will resemble the one shown here:

```
[root@fedora-serverA boot]# cat /boot/grub/menu.lst
# grub.conf generated by anaconda
default=0
timeout=5
#splashimage=(hd0,0)/grub/splash.xpm.gz
# hiddenmenu
title Fedora (2.6.25-14.fc9.i686)
root (hd0,0)
kernel /vmlinuz-2.6.25-14.fc9.i686 ro root=UUID= 7db5-4c27 rhgb quiet
initrd /initrd-2.6.25-14.fc9.i686.img
title The Duplicate Kernel
color yellow/black
root (hd0,0)
kernel /duplicate-kernel ro root=UUID=7db5-4c27
initrd /duplicate-initrd.img
title The change color entry
color yellow/black
```

When the system reboots, you can test your changes by following the next steps while at the initial grub screen.
9. After the GRUB menu appears, select The Change Color Entry, and press enter. The color of the menu should change to the color you specified in the **menu.1** file using the **color** directive.
10. Finally, verify that you are able to boot the new kernel entry that you created, that is, the "The Duplicate Kernel" entry. Select "The Duplicate Kernel" entry and then press enter.

Demo Of Adding a New Kernel to Boot with GRUB



LILO

- LILO, short for ***Linux Loader***, is a boot manager.
- It allows you ***to boot multiple operating systems, provided each system exists on its own partition.*** (Under PC-based systems, the *entire* boot partition must also exist beneath the 1024-cylinder boundary.)
- In addition to booting multiple operating systems, with LILO, you can choose various kernel configurations or versions to boot. This is especially handy when you're trying kernel upgrades before adopting them.
- Configuring LILO is straightforward:
    - A configuration file (**/etc/lilo.conf**) specifies which partitions are bootable and, if a partition is Linux, which kernel to load.
    - When the **/sbin/lilo** program runs, it takes this partition information and rewrites the boot sector with the necessary code to present the options as specified in the configuration file.
    - At boot time, a prompt (usually **lilo:**) is displayed, and you have the option of specifying the operating system. (Usually, a default can be selected after a timeout period.)
    - LILO loads the necessary code, the kernel, from the selected partition and passes full control over to it.
- LILO is what is known as a two-stage boot loader. The first stage loads LILO itself into memory and prompts you for booting instructions with the **lilo:** prompt or a colorized boot menu. Once you select the OS to boot and press enter, LILO enters the second stage, booting the Linux operating system.
- As was stated earlier in the chapter, LILO has somewhat fallen out of favour with most of the newer Linux distributions. Some of the distributions do not even give you the option of selecting or choosing LILO as your boot manager!

Bootstrapping

This section will cover the process of bootstrapping the operating system. We'll begin with the Linux boot loader (usually GRUB for PCs).

1. Kernel Loading

Once GRUB has started and you have selected Linux as the operating system to boot, the first thing to get loaded is the kernel. Keep in mind that no operating system exists in memory at this point, and PCs (by their unfortunate design) have no easy way to access all of their memory.

Thus, the kernel must load completely into the first megabyte of available random access memory (RAM). In order to accomplish this, the kernel is compressed.

The head of the file contains the code necessary to bring the CPU into protected mode (thereby removing the memory restriction) and decompress the remainder of the kernel.

2. Kernel Execution

With the kernel in memory, it can begin executing.

It knows only whatever functionality is built into it, which means any parts of the kernel compiled as modules are useless at this point.

At the very minimum, the kernel must have enough code to set up its virtual memory subsystem and root file system (usually, the **ext3** file system). Once the kernel has started, a hardware probe determines what device drivers should be initialized.

From here, the kernel can *mount* the root file system.

The kernel mounts the root file system and starts a program called **init**, which is discussed in the next section.

THE INIT PROCESS

- The **init** process is the first non-kernel process that is started, and, therefore, it always gets the ***process ID number of 1***.
- **init** reads its configuration file, **/etc/inittab**, and determines the ***runlevel*** where it should start.
- Essentially, a runlevel dictates the system's behavior. Each level (designated by an integer between 0 and 6) serves a specific purpose.
- A runlevel of **initdefault** is selected if it exists; otherwise, you are prompted to supply a runlevel value.
- The runlevel values are as follows:

  0 Halt the system
  1 Enter single-user mode
  2 Multiuser mode, but without Network File System (NFS)
  3 Full multiuser mode (normal)
  4 Unused
  5 Same as runlevel 3, except using an X Window System login rather than a text-based login
  6 Reboot the system

- When it is told to enter a runlevel, **init** executes a script, as dictated by the **/etc/ inittab** file.
- The default runlevel that the system boots into is determined by the **initdefault** entry in the **/etc/inittab** file. If, for example, the entry in the file is

  id:3:initdefault:

- This means that the system will boot into runlevel 3. But if, on the other hand, the entry in the file is

  id:5:initdefault:

- This means the system will boot into runlevel 5, with the X Window subsystem running with a graphical login screen.

.

RC SCRIPTS

- In the preceding section, we mentioned that the **/etc/inittab** file specifies which scripts to run when runlevels change. These ***scripts are responsible for either starting or stopping the services*** that are particular to the runlevel.
- Because of the number of services that need to be managed, **rc** scripts are used. The main one, **/etc/rc.d/rc**, is responsible for calling the appropriate scripts in the correct order for each runlevel.
- For each runlevel, a subdirectory exists in the **/etc/rc.d** directory. These runlevel subdirectories follow the naming scheme of **rc *X* .d**, where *X* is the runlevel. For example, all the scripts for runlevel 3 are in **/etc/rc.d/rc3.d**.
- In the runlevel directories, symbolic links are made to scripts in the **/etc/rc.d/init.d** directory. Instead of using the name of the script as it exists in the **/etc/rc.d/init.d** directory, however, the symbolic links are prefixed with an **S**, if the script is to start a service, or with a **K**, if the script is to stop (or kill) a service.
- Note that these two letters are case sensitive. You must use uppercase letters, or the start up scripts will not recognize them.
- In many cases, the order in which these scripts are run makes a difference. (For example, you can't start services that rely on a configured network interface without first enabling and configuring the network interface!)
- To enforce order, a two-digit number is suffixed to the **S** or **K**. Lower numbers execute before higher numbers; for example, **/etc/rc.d/rc3.d/ S10network** runs before **/etc/rc.d/rc3.d/S55sshd** (**S10network** configures the network settings, and **S55sshd** starts the Secure Shell [SSH] server).
- The scripts pointed to in the **/etc/rc.d/init.d** directory are the workhorses; they perform the actual process of starting and stopping services.

ENABLING AND DISABLING SERVICES

- At times, you may find that you simply don't need a particular service to be started at boot time.
- This is especially important if you are configuring the system as a server and need only specific services and nothing more.
- As described in the preceding sections, you can cause a service not to be started by simply renaming the symbolic link in a particular runlevel directory; rename it to start with a **K** instead of an **S**.
- Once you are comfortable working with the command line, you'll quickly find that it is easy to enable or disable a service.
- The startup runlevels of the service/program can also be managed using the **chkconfig** utility.
- To view all the runlevels in which the **carpald.sh** program is configured to start up, type

```
[root@fedora-serverA ~]# chkconfig --list carpald
Carpald 0:off 1:off 2:off 3:on 4:off 5:on 6:off
```

- To make the **carpald.sh** program start up automatically in runlevel 2, type

```
[root@serverA ~]# chkconfig --level 2 carpald on
```

- If you check the list of runlevels for the **carpald.sh** program again, you will see that the field for runlevel 2 has been changed from 2:off to 2:on. Type

```
[root@fedora-serverA ~]# chkconfig --list carpald
carpald 0:off 1:off 2:on 3:on 4:off 5:on 6:off
```

- ***Graphical user interface (GUI) tools*** are available that will help you manage which services start up at any given runlevel.
- In Fedora and other Red Hat–type systems (including RHEL), one such tool is the **system-config-services** utility.
- To launch the program, type

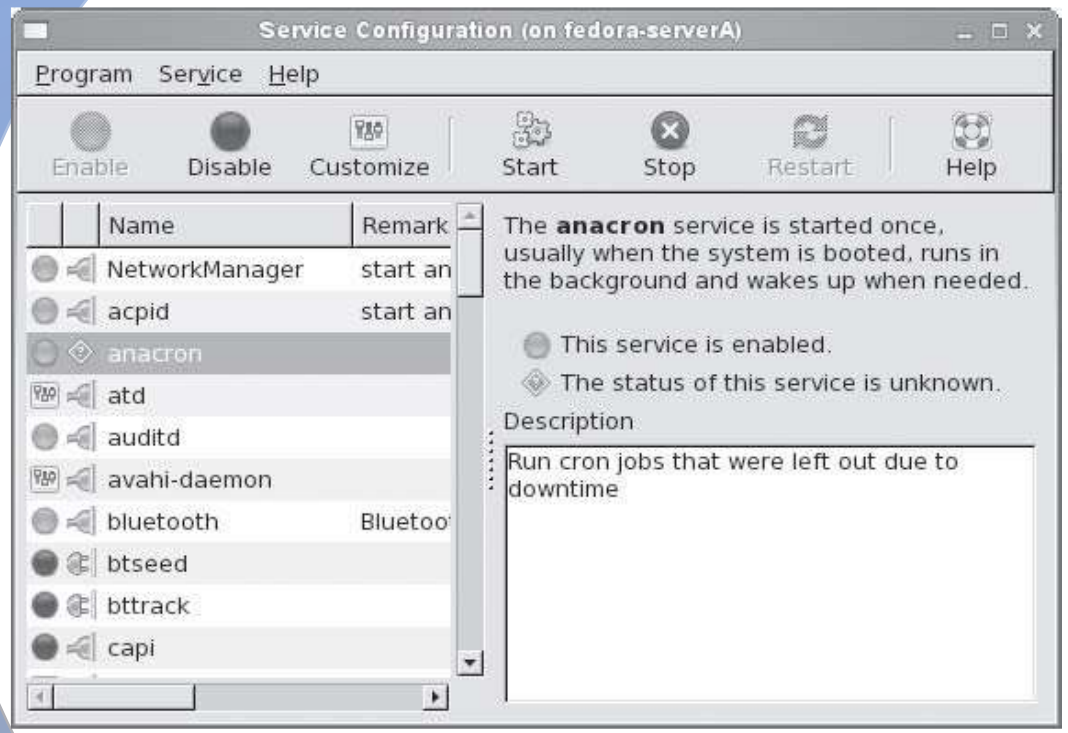
```
[root@fedora-serverA ~]# system-config-services
```

- On a system running OpenSuSE Linux, the equivalent GUI program (see Figure 6-2) can be launched by typing:

```
suse-serverA:~ # yast2 runlevel
```

- On an Ubuntu system, the equivalent GUI tool (see Figure 6-3) can be launched by typing:

```
yyang@ubuntu-serverA:~$ sudo services-admin
```



Disabling a Service

- To completely disable a service, you must, at a minimum, know the name of the service. You can then use the **chkconfig** tool to permanently turn it off, thereby preventing it from starting in all runlevels.
- For example, to disable our "life-saving" **carpald.sh** program, you could type

```
[root@fedora-serverA ~]# chkconfig carpald off
```

- If you check the list of runlevels for the **carpald.sh** program again, you will see that it has been turned off for all runlevels. Type

```
[root@fedora-serverA ~]# chkconfig --list carpald
carpald 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

- To permanently remove the **carpald.sh** program from under the **chkconfig** utility's control, you will use **chkconfig**'s delete option. Type

```
[root@fedora-serverA ~]# chkconfig --del carpald
```

- We are done with our sample **carpald.sh** script, and to prevent it from flooding us with e-mail notifications in the future (in case we accidentally turn it back on), we can delete it from the system for good. Type

```
[root@fedora-serverA ~]# rm -f /usr/local/sbin/carpald.sh
```

- And those are the ABC's of how services start up and shut down automatically in Linux. Now go out and take a break.