Chapter 8

System Initialization, X Windows, and Localization

Chapter Objectives

1 Summarize the major steps necessary to boot a Linux system.

2 Outline the configuration of the GRUB boot loader.

3 Detail the UNIX SysV and Systemd system initialization processes.

4 Start, stop, and restart daemons.

5 Configure the system to start and stop daemons upon entering certain runlevels and targets.

6 Explain the purpose of the major Linux GUI components: X Windows, window manager, and desktop environment.

7 Configure X Windows settings and accessibility options.

8 Configure time, time zone, and locale information on a Linux system.

In this chapter, you investigate the boot process in greater detail. You explore how to configure boot loaders and the process used to start daemons after the kernel has loaded. Additionally, you examine the procedures used to start and stop daemons and set them to start automatically at boot time. Next, you examine the various components that comprise the Linux GUI and how to configure them using common Linux utilities. Finally, you examine the Linux tools and processes used to configure localization options, including locale, date, and time zone information.

The Boot Process

When a computer first initializes, the system BIOS performs a **Power On Self Test (POST)**. Following the POST, the BIOS checks its configuration for boot devices and operating systems to execute. Typically, computers first check for an operating system on removable media devices, such as DVDs and USB flash memory drives, because they can contain installation media for an operating system. If it fails to find an operating system on any of these options, the BIOS usually checks the MBR/GPT on the first hard disk drive inside the computer.

Note 1

Recall that you can alter the order in which boot devices are checked in the computer BIOS.

Note 2

A computer BIOS can also be configured to boot an operating system from an NFS, HTTP, or FTP server across the network, provided that the computer network interface supports the Preboot Execution Environment (PXE) standard. This process is called **netbooting** and is primarily used to boot Linux live install media on a computer from across a network in order to perform a new local Linux installation.

The MBR/GPT normally contains the first part of a **boot loader** that can then locate and execute the kernel of the operating system. Alternatively, the MBR/GPT might contain a pointer to a partition on the system that contains a boot loader on the first sector; this partition is referred to as the **active partition**. There can be only one active partition per hard disk drive.

Note 3

In addition to storing the list of all partitions on the hard disk drive, the MBR/GPT stores the location of the active partition.

If the system has a UEFI BIOS, then boot loader is not loaded from the MBR/GPT or first sector of the active partition; it is instead loaded from the UEFI System Partition by the UEFI BIOS. There can only be one UEFI System Partition per hard disk drive. If **secure boot** is enabled in the UEFI BIOS, the digital signature of the boot loader within the UEFI System Partition is first checked to ensure that it has not been modified by malware.

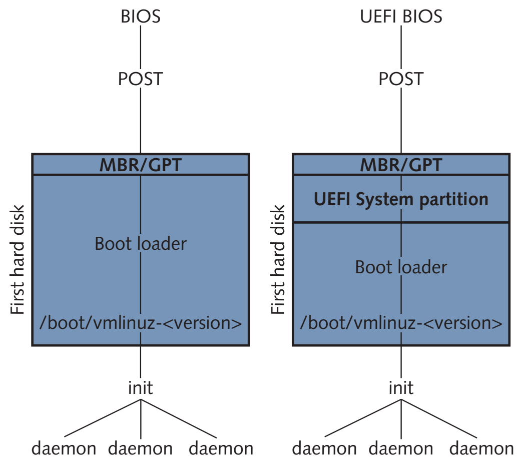
Regardless of whether the boot loader is loaded from the MBR/GPT, the first sector of the active partition, or the UEFI System Partition, the remainder of the boot process is the same. The boot loader then executes the Linux kernel from the partition that contains it.

Note 4

The Linux kernel is stored in the /boot directory and is named vmlinux-<kernel version> if it is not compressed, or vmlinuz-<kernel version> if it is compressed.

After the Linux kernel is loaded into memory, the boot loader is no longer active; instead, the Linux kernel continues to initialize the system by loading daemons into memory. Recall from Chapter 1 that a daemon is a system process that provides useful services, such as printing, scheduling, and operating system maintenance. The first daemon process on the system is called the **initialize (init) daemon**; it is responsible for loading all other daemons on the system required to bring the system to a usable state in which users can log in and interact with services. The whole process is depicted in Figure 8-1.

Figure 8-1 The boot process



Boot Loaders

As discussed in the previous section, the primary function of boot loaders during the boot process is to load the Linux kernel into memory. However, boot loaders can perform other functions as well, including passing information to the kernel during system startup and booting other operating systems that are present on the hard disk drive. Using one boot loader to boot one of several operating systems is known as **multi booting**; the boot loader simply loads a different operating system kernel based on user input.

Note 5

Unless virtualization software is used, only one operating system can be active at any one time.

The two most common boot loaders used on Linux systems are GRUB and GRUB2.

GRUB Legacy

The original **GRand Unified Bootloader (GRUB)** boot loader was created in 1999 as a replacement boot loader for the original Linux boot loader for hard disk drives that have an MBR. It supports the booting of several different operating systems, including Linux, macOS, BSD UNIX, Solaris UNIX, and Windows.

Note 6

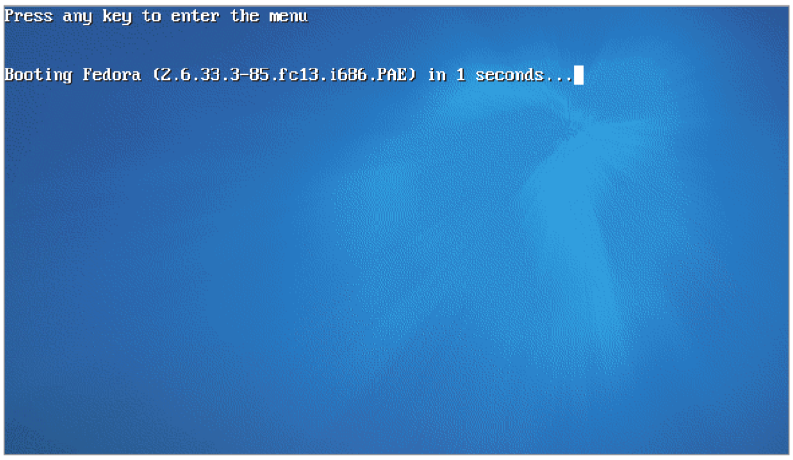
The original GRUB boot loader is called **GRUB Legacy** today as it is not used on modern Linux systems. However, because specialized Linux systems typically have a very long lifetime within production environments, it is not uncommon to see a Linux system that still uses GRUB legacy today.

The first major part of the GRUB Legacy boot loader (called Stage 1) typically resides on the MBR. The remaining parts of the boot loader are called Stage 1.5 and Stage 2. Stage 1.5 resides in the unused 30 KB of space following the MBR, and Stage 2 resides in the /boot/grub directory. Stage 1 simply points to Stage 1.5, which loads filesystem support and proceeds to load Stage 2. Stage 2 performs the actual boot loader functions and displays a graphical boot loader screen like that shown in Figure 8-2.

Note 7

Recall that the /boot directory is normally mounted to its own filesystem on most Linux systems; as a result, Stage 2 typically resides on the /boot filesystem alongside the Linux kernel.

Figure 8-2 The legacy GRUB boot screen for a Fedora system



```
Press any key to enter the menu

Booting Fedora (2.6.33.3-85.fc13.i686.PAE) in 1 seconds...
```

Source: GNU GRUB

You configure GRUB Legacy by editing a configuration file (/boot/grub/grub.conf) that is read directly by the Stage 2 boot loader. An example /boot/grub/grub.conf file for a legacy Fedora system is shown next:

```
[root@server1 ~]# cat /boot/grub/grub.conf
# NOTE: You do not have a /boot partition. This means that all
# kernel and initrd paths are relative to /, or root (hd0,0)

boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu

title Fedora (2.6.33.3-85.fc13.i686.PAE)
  root (hd0,0)
  kernel /boot/vmlinuz-2.6.fc13.i686.PAE ro root=/dev/sda1 rhgb quiet
  initrd /boot/initramfs-2.6.fc13.i686.PAE.img
[root@server1 ~]#_
```

Note 8

Alternatively, you can view and edit the /etc/grub.conf file, which is simply a symbolic link to /boot/grub/grub.conf.

Note 9

As with shell scripts, lines can be commented out of /boot/grub/grub.conf by preceding those lines with a # symbol.

Note 10

Some other Linux distributions, such as Ubuntu Linux, use a /boot/grub/menu.lst file in place of /boot/grub/grub.conf to hold GRUB Legacy configuration.

To understand the entries in the /boot/grub/grub.conf file, you must first understand how GRUB refers to partitions on hard disk drives. Hard disk drives and partitions on those hard disk drives are identified by numbers in the following format: (hd<drive#>,<partition#>). Thus, the `(hd0,0)` notation in the preceding /boot/grub/grub.conf file refers to the first hard disk drive on the system (regardless of whether it is SCSI, SATA, or PATA) and the first partition on that hard disk drive, respectively. Similarly, the second partition on the first hard disk drive is referred to as (hd0,1), and the fourth partition on the third hard disk drive is referred to as (hd2,3).

In addition, GRUB calls the partition that contains Stage 2 the **GRUB root partition**. Normally, the GRUB root partition is the filesystem that contains the /boot directory and should not be confused with the Linux root filesystem. If your system has a separate partition mounted to /boot, GRUB refers to the file /boot/grub/grub.conf as /grub/grub.conf. If your system does not have a separate filesystem for the /boot directory, this file is simply referred to as /boot/grub/grub.conf in GRUB.

Thus, the example /boot/grub/grub.conf file shown earlier displays a graphical boot screen (`splashimage=(hd0,0)/boot/grub/splash.xpm.gz`) and boots the default operating system kernel on the first hard drive (`default=0`) in 5 seconds (`timeout=5`) without showing any additional menus (`hiddenmenu`). The default operating system kernel is located on the GRUB root filesystem (`root (hd0,0)`) and is called `/boot/vmlinuz-2.6.fc13.i686.PAE`.

The kernel then mounts the root filesystem on /dev/sda1 (`root=/dev/sda1`) initially as read-only (`ro`) to avoid problems with the `fsck` command and uses a special **initramfs** disk filesystem image to load modules into RAM that are needed by the Linux kernel at boot time (`initrd /boot/initramfs-2.6.fc13.i686.PAE.img`).

Note 11

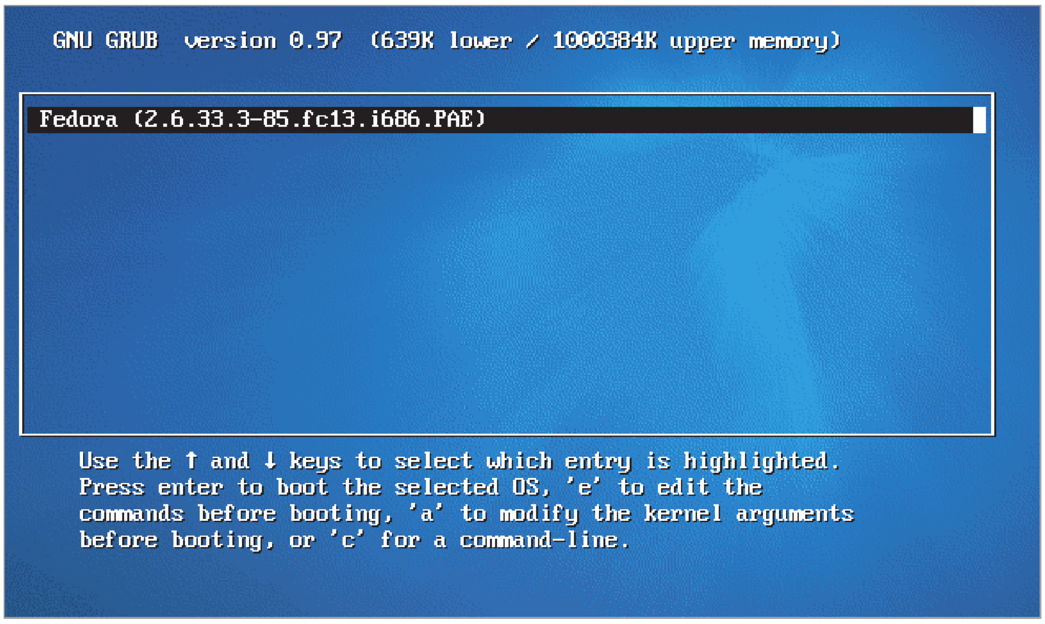
Most Linux distributions use a UUID (e.g., `root=UUID=42c0fce6-bb79-4218-af1a-0b89316bb7d1`) in place of `root=/dev/sda1` to identify the partition that holds the root filesystem at boot time.

All other keywords on the `kernel` line within /boot/grub/grub.conf are used to pass information to the kernel from Stage 2. For example, the keyword `rhgb` (Red Hat Graphical Boot) tells the Linux kernel to use a graphical boot screen as it is loading daemons, and the keyword `quiet` tells the Linux kernel to avoid printing errors to the screen during system startup. You can add your own keywords to the `kernel` line in /boot/grub/grub.conf to control how your Linux kernel is loaded. For example, appending the text `nosmp` to the `kernel` line disables Symmetric Multi-Processing (SMP) support within the Linux kernel. Alternatively, appending the text `mem=16384M` to the `kernel` line forces your Linux kernel to see 16384 MB of physical RAM in case your Linux kernel does not detect all the RAM in your computer properly.

Normally, GRUB Legacy allows users to manipulate the boot loader during system startup; to prevent this, you can optionally password protect GRUB modifications during boot time.

Recall from the /boot/grub/grub.conf file shown earlier that you have five seconds after the BIOS POST to interact with the boot screen shown in Figure 8-2. If you press any key within these five seconds, you will be presented with a graphical boot menu screen like the one shown in Figure 8-3 that you can use to manipulate the boot process. If you have several Linux kernels installed on your system (from updating your system software), you can select the kernel that you would like to boot, highlight your kernel and press `a` to append keywords to the kernel line, or press `e` to edit the entire boot configuration for the kernel listed in /boot/grub/grub.conf at boot time. You can also press `c` to obtain a grub> prompt where you can enter a variety of commands to view system hardware configuration, find and display files, alter the configuration of GRUB, or boot an operating system kernel. Typing `help` at this grub> prompt will display a list of available commands and their usage.

Figure 8-3 The legacy GRUB boot menu for a Fedora system



```
    GNU GRUB  version 0.97  (639K lower / 1000384K upper memory)

 Fedora (2.6.33.3-85.fc13.i686.PAE)

      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.
```

Source: GNU GRUB

If the GRUB Legacy boot loader becomes damaged, you can reinstall it using the **grub-install command** that is available on the system or on a live Linux system used for system rescue. To install GRUB Legacy Stage 1 on the MBR of the first SATA hard disk drive, you can type the following command:

```
[root@server1 ~]# grub-install /dev/sda
Installation finished. No error reported.
Below are the contents of the device map /boot/grub/device.map.
If lines are incorrect, fix them and re-run the script 'grub-install'.
(hd0)      /dev/sda
[root@server1 ~]#_
```

Note 12

Alternatively, you can use the `grub-install /dev/sda1` command to install GRUB Legacy Stage 1 at the beginning of the first primary partition of the same hard disk drive.

GRUB2

GRand Unified Bootloader version 2 (GRUB2) is the boot loader commonly used on modern Linux systems; it supports storage devices that use either an MBR or GPT, as well as newer storage technologies such as NVMe.

Note 13

GRUB2 has additional support for modern systems that do not use the Intel x86_64 architecture (e.g., ARM64), as well as specialized hardware systems.

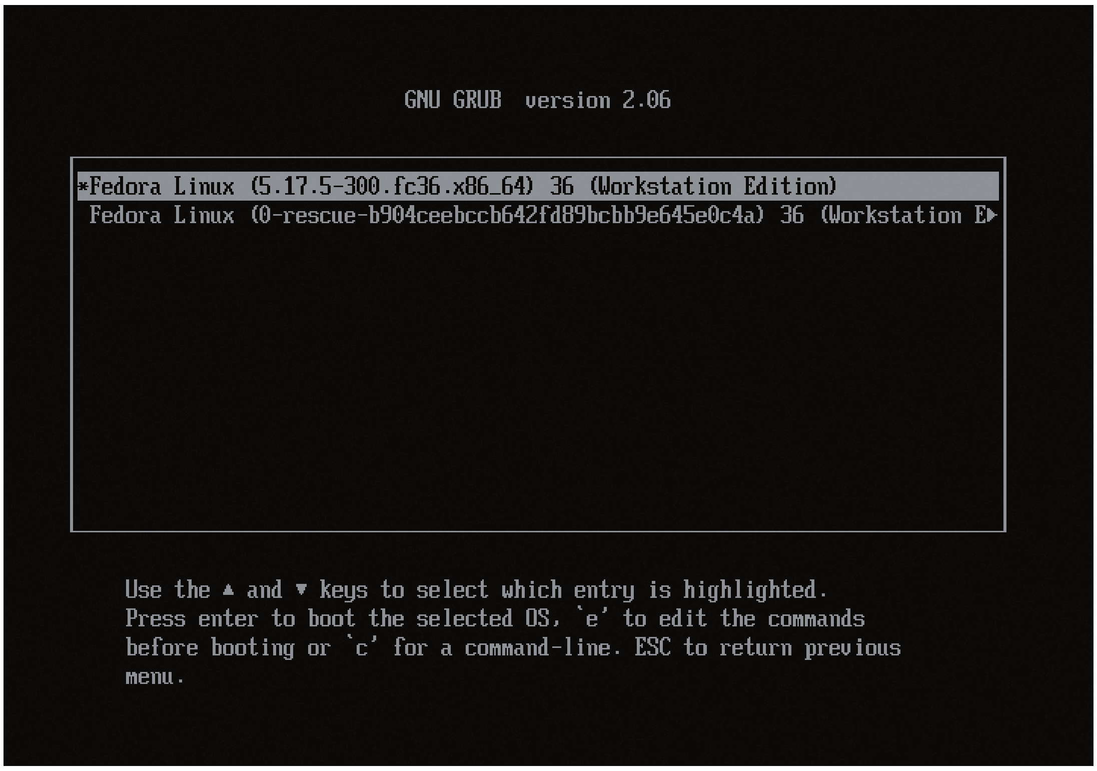
For a system that uses a standard BIOS, GRUB2 has a similar structure to GRUB Legacy. GRUB2 Stage 1 typically resides on the MBR or GPT. On MBR hard disk drives, Stage 1.5 resides in the unused 30 KB of space following the MBR, and on GPT hard disk drives, Stage 1.5 resides in a BIOS Boot partition that is created for this purpose by the Linux installation program. Stage 2 resides in the /boot/grub directory (or /boot/grub2 directory on some Linux distributions) and loads a terminal-friendly boot loader screen like that shown in Figure 8-4. As with GRUB Legacy, you can select the kernel that you would like to boot at the boot loader screen, as well as highlight your kernel and press `e` to edit the entire boot configuration for the kernel or press `c` to obtain a prompt where you can enter GRUB configuration commands.

For a system that uses a UEFI BIOS, all GRUB2 stages are stored entirely on the UEFI System Partition within a UEFI application called grubx64.efi on Intel x86_64-based systems. If it isn't already present, the UEFI System Partition is created during Linux installation, formatted using the FAT filesystem, and mounted to the /boot/efi directory during the Linux boot process. Because the UEFI System Partition can contain boot loaders for multiple different operating systems, grubx64.efi is stored underneath a subdirectory. For example, on Fedora systems, you can find grubx64.efi under the /boot/efi/EFI/fedora directory after the Linux system has booted. If the UEFI BIOS has secure boot enabled, you will also find UEFI applications within this subdirectory that store the digital signature information used to ensure that GRUB2 has not been modified by malware; for a Fedora system, these are called shim.efi and shimx64.efi.

Note 14

There is still a /boot/grub (or /boot/grub2) directory on systems that use GRUB2 alongside a UEFI BIOS, but it only contains files that are referenced by GRUB2 after it is fully loaded, such as the background image used for the GRUB boot screen.

Figure 8-4 The GRUB2 boot screen on a Fedora system



```
GNU GRUB  version 2.06

*Fedora Linux (5.17.5-300.fc36.x86_64) 36 (Workstation Edition)
 Fedora Linux (0-rescue-b904ceebccb642fd89bcbb9e645e0c4a) 36 (Workstation E▸

Use the ▴ and ▾ 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. ESC to return previous
menu.
```

Source: GNU GRUB

After grubx64.efi is executed by the UEFI BIOS, GRUB2 displays the same boot loader screen shown in Figure 8-4, where you can interact with GRUB or boot the Linux kernel but will include an additional menu option called UEFI Firmware Settings that allows you to configure UEFI BIOS settings.

The main configuration file for GRUB2 is called grub.cfg; it is stored within the /boot/grub (or /boot/grub2) directory on computers that have a standard BIOS, and within the UEFI System Partition on computers with a UEFI BIOS (e.g., in the /boot/efi/EFI/fedora directory on Fedora Linux).

The syntax of the grub.cfg file used by GRUB2 is different than the grub.conf file used by GRUB Legacy, as shown in the following excerpt:

```
set timeout=5

menuentry 'Fedora (5.17.5-300.fc36.x86_64) 36 (Workstation Edition)' --class fedora
--class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option
'gnulinux-5.17.5-300.fc36.x86_64-advanced-9ff25add-035b-4d26-9129-a9da6d0a6fa3' {
      load_video
      set gfxpayload=keep
      insmod gzio
      insmod part_gpt
      insmod ext2
      set root='hd0,gpt2'

      linuxefi /vmlinuz-5.17.5-300.fc36.x86_64 root=UUID=9ff25add-035b-4d26-9129-
      a9da6d0a6fa3 ro resume=UUID=4837de7b-d96e-4edc-8d3d-56df0a17ca62 rhgb quiet
      LANG=en_US.UTF-8

      initrdefi /initramfs-5.17.5-300.fc36.x86_64.img
}
```

Each menu entry at the graphical boot screen is identified by a `menuentry` paragraph that first loads required hardware support using `insmod` commands before accessing the GRUB root partition (mounted to /boot). The notation for identifying hard disk drives in GRUB2 is different from GRUB Legacy; partition numbers start at 1 and are prefixed by `msdos` for MBR partitions and `gpt` for GPT partitions. Thus, the `set root='hd0,gpt2'` notation in the preceding excerpt indicates that the GRUB root partition is the second GPT partition on the first hard disk drive. Next, the Linux kernel is loaded from the specified file on the GRUB root partition (`linuxefi /vmlinuz-5.17.5-300.fc36.x86_64` if the system has a UEFI BIOS, or `linux16 /vmlinuz-5.17.5-300.fc36.x86_64` if the system has a standard BIOS). The options following the Linux kernel are used to specify the UUID of the root filesystem that is initially loaded read-only (`ro`), as well as the use of a graphical boot screen (`rhgb`) that suppresses errors from being shown during the boot process (`quiet`). The `LANG=en_US.UTF-8` kernel option sets the default locale information that is discussed later in this chapter. As with GRUB Legacy, an initramfs image is used to load modules into RAM that are needed by the Linux kernel at boot time (`initrdefi /initramfs-5.17.5-300.fc36.x86_64.img` if the system has a UEFI BIOS, or `initrd16 /initramfs-5.17.5-300.fc36.x86_64.img` if the system has a standard BIOS). By default, GRUB2 stores the last menu entry chosen by the user at the graphical boot screen in the /boot/grub/grubenv (or /boot/grub2/grubenv) file and sets it as the default for the subsequent boot process if you do not select a menu entry within 5 seconds (`set timeout=5`).

Note 15

If you do not see `menuentry` paragraphs within grub.cfg, GRUB2 is configured to use the BootLoaderSpec (blscfg) module. In this case, the information for each kernel is stored within a *UUID-kernel*.conf file under the /boot/loader/entries directory, where *UUID* is the UUID of the root filesystem, and *kernel* is the version of the Linux kernel.

The grub.cfg file was not meant to be edited manually; instead, it is automatically built via entries within the /etc/default/grub file, and the output of any shell scripts stored within the /etc/grub.d directory. When you install a device driver that needs to be loaded by the boot loader (e.g., disk controller devices), the device driver package will often add a file to the /etc/grub.d directory that provides the necessary configuration. For any other settings, you should add or modify the existing lines within the /etc/default/grub file. A sample /etc/default/grub file on a Fedora system is shown here:

```
[root@server1 ~]# cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG="true"
[root@server1 ~]#_
```

From the preceding output, you could change the GRUB_TIMEOUT line to modify the number of seconds that the GRUB2 boot loader screen appears before the default operating system is loaded or add parameters that are passed to the Linux kernel by modifying the GRUB_CMDLINE_LINUX line. Also note that the /etc/default/grub file does not list any operating system information. This is because GRUB2 uses the /etc/grub.d/30_os-prober script to automatically detect available operating system kernels on the system and configure them for use with GRUB2. If you would like to manually set the default operating system kernel listed at the GRUB2 boot loader screen, you can set the GRUB_DEFAULT line to the appropriate line number, starting from 0. For example, to set the second OS line shown in Figure 8-4 (`Fedora (0-rescue-b904ceebccb642fd89bcbb9e645e0c4a)`) as the default OS to boot, set GRUB_DEFAULT=1 in the /etc/default/grub file.

After modifying the /etc/default/grub file or adding scripts to the /etc/grub.d directory, you can run the **`grub2-mkconfig` command** to rebuild the grub.cfg file. For example, to rebuild the /boot/grub2/grub.cfg file on a Fedora system, you can use the following:

```
[root@server1 ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file …
done
[root@server1 ~]#_
```

On a system with a UEFI BIOS, you instead specify the location of the grub.cfg file within the UEFI System Partition (e.g., `grub2-mkconfig -o /boot/efi/EFI/fedora/grub.cfg`).

Note 16

On some Linux distributions, you can use the **`update-grub2` command** without arguments to rebuild the grub.cfg file. The `update-grub2` command runs the appropriate `grub2-mkconfig` command to rebuild grub.cfg in the correct location on the system.

As with GRUB Legacy, if the GRUB2 boot loader becomes damaged, you can reinstall it. For systems with a standard BIOS, you can use the **`grub2-install` command**. To reinstall GRUB2 Stage 1 on the MBR/GPT of the first SATA hard disk drive on a Fedora system, you can type the following command:

```
[root@server1 ~]# grub2-install /dev/sda
Installation finished. No error reported.
[root@server1 ~]#_
```

For systems with a UEFI BIOS, you can reinstall GRUB2 by reinstalling the associated GRUB2 UEFI packages. To reinstall the GRUB2 UEFI packages, you can use the `dnf reinstall grub2-efi grub2-efi-modules shim` command on a Fedora system, or the `apt-get install --reinstall grub-efi` command on an Ubuntu system.

Note 17

If the system is unable to boot, you can boot a live Linux system used for system rescue, change to the root filesystem on the hard disk drive using the `chroot` command, and then run the appropriate command to reinstall GRUB2.

Note 18

For simplicity, some Linux distributions use `grub-mkconfig`, `grub-install` and `update-grub` in place of `grub2-mkconfig`, `grub2-install` and `update-grub2`, respectively.

Note 19

When you update your Linux operating system software, you may receive an updated version of your distribution kernel. In this case, the software update copies the new kernel to the /boot directory, creates a new initramfs to match the new kernel, and modifies the GRUB2 configuration to ensure that the new kernel is listed at the top of the graphical boot menu and set as the default for subsequent boot processes.

Note 20

Invalid entries in the GRUB2 configuration file or a damaged initramfs can prevent the kernel from loading successfully; this is called a **kernel panic** and will result in a system halt immediately after GRUB attempts to load the Linux kernel. In this case, you will need to edit the GRUB2 configuration file from a live Linux system used for system rescue or generate a new initramfs using either the **`dracut` command** or **`mkinitrd` command**.

Linux Initialization

Recall that after a boot loader loads the Linux operating system kernel into memory, the kernel resumes control and executes the init daemon, which then performs a **system initialization process** to execute other daemons and bring the system into a usable state.

Traditional Linux systems have used a system initialization process from the **UNIX SysV** standard. However, most modern Linux distributions have adopted the **Systemd** system initialization process. Systemd is completely compatible with the UNIX SysV standard yet implements new features for managing all system devices, including Linux kernel modules, daemons, processes, filesystems, and network sockets.

Note 21

Because Systemd is a recent technology, not all Linux daemons have been rewritten to use it. As a result, modern Linux distributions that have adopted Systemd may still use the UNIX SysV system initialization process to initialize some daemons.

Working with the UNIX SysV System Initialization Process

Linux systems that use the UNIX SysV system initialization process can choose from the traditional UNIX SysV system initialization process or the **upstart** system initialization process derived from UNIX SysV. In both systems, the init daemon runs a series of scripts to start other daemons on the system to provide system services and ultimately allow users to log in and use the system. Furthermore, the init daemon is responsible for starting and stopping daemons after system initialization, including stopping daemons before the system is halted or rebooted.

Runlevels

Because the init daemon must often manage several daemons at once, it categorizes the system into runlevels. A **runlevel** defines the number and type of daemons that are loaded into memory and executed by the kernel on a particular system. At any time, a Linux system might be in any of the seven standard runlevels defined in Table 8-1.

Table 8-1 Linux runlevels

Runlevel	Common Name	Description
0	Halt	A system that has no daemons active in memory and is ready to be powered off
1 s S single	Single User Mode	A system that has only enough daemons to allow one user (the root user) to log in and perform system maintenance tasks
2	Multiuser Mode	A system that has most daemons running and allows multiple users the ability to log in and use system services; most common network services other than specialized network services are available in this runlevel as well
3	Extended Multiuser Mode	A system that has the same abilities as Multiuser Mode, yet with all extra networking services started (e.g., FTP, Apache, NFS)
4	Not used	Not normally used, but can be customized to suit your needs
5	Graphical Mode	A system that has the same abilities as Extended Multiuser Mode, yet with a graphical login program; on systems that use the GNOME desktop, this program is called the **GNOME Display Manager (GDM)** and is typically started on tty1 to allow for graphical logins
6	Reboot	A special runlevel used to reboot the system

Note 22

Because the init daemon is responsible for starting and stopping daemons and, hence, changing runlevels, runlevels are often called **initstates**.

To see the current runlevel of the system and the previous runlevel (if runlevels have been changed since system startup), you can use the **`runlevel` command**, as shown in the following output:

```
[root@server1 ~]# runlevel
N 5
[root@server1 ~]#_
```

The preceding runlevel command indicates that the system is in runlevel 5 and that the most recent runlevel prior to entering this runlevel is nonexistent (N).

To change the runlevel on a running system, you can use the **`init` command**, as shown in the following output:

```
[root@server1 ~]# runlevel
N 5
[root@server1 ~]# init 1
*A list of daemons that are being stopped by the init
daemon while the system enters single user mode.*

Telling INIT to go to single user mode.
[root@server1 /]#_
[root@server1 /]# runlevel
5 1
[root@server1 /]#_
```

Note 23

The **telinit command** can be used in place of the `init` command when changing runlevels. Thus, the command `telinit 1` can instead be used to switch to Single User Mode.

Note 24

You can also pass options from the boot loader to the Linux kernel to force the system to boot to a particular runlevel. If you append the keyword `single` to the kernel line within the GRUB Legacy or GRUB2 configuration screen, you will boot to Single User Mode.

The /etc/inittab File

Unless you specify otherwise, the init daemon enters the default runlevel indicated in the /etc/inittab file. In the past, the /etc/inittab file contained the entire configuration for the init daemon. However, on systems that use the UNIX SysV system initialization process exclusively today, the /etc/inittab often contains a single uncommented line that configures the default runlevel, as shown here:

```
[root@server1 ~]# cat /etc/inittab
id:5:initdefault:
[root@server1 ~]#_
```

The line `id:5:initdefault:` in the /etc/inittab file tells the init daemon that runlevel 5 is the default runlevel to boot to when initializing the Linux system at system startup.

Note 25

Runlevel 5 is the default runlevel in most Linux distributions that have a desktop environment installed.

Runtime Configuration Scripts

During the boot process, the init daemon must execute several scripts that prepare the system, start daemons, and eventually bring the system to a usable state. These scripts are called **runtime configuration (rc) scripts**.

On Linux systems that use the traditional UNIX SysV system initialization process, the init daemon identifies the default runlevel in the /etc/inittab file and then proceeds to execute the files within the /etc/rc#.d directory that start with S or K, where # is the runlevel number. If the default runlevel is 5, then the init daemon would execute all files that start with S or K in the /etc/rc5.d directory in alphabetical order. The S or the K indicates whether to Start or Kill (stop) the daemon upon entering this runlevel, respectively. Some sample contents of the /etc/rc.d/rc5.d directory are shown in the following output:

```
[root@server1 ~]# ls /etc/rc5.d
README          S20screen-cleanup   S91apache2
S15bind9        S20zfs-mount        S92tomcat7
S19postgresql   S20zfs-share        S99grub-common
S20postfix      S70dns-clean        S99ondemand
S20rsync        S70pppd-dns         S99rc.local
[root@server1 ~]#_
```

From the preceding output, you can see that the init daemon will start the postfix daemon (S20postfix) upon entering this runlevel. To ensure that the files in the preceding directory are executed in a specific order, a sequence number is added following the S or K at the beginning of the filename. Thus, the file S19postgresql is always executed before the file S20postfix.

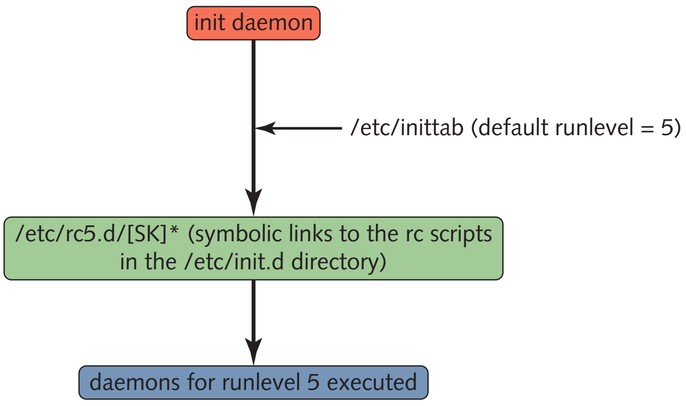
Recall that runlevel 1 (Single User Mode) contains only enough daemons for a single user to log in and perform system tasks. If a user tells the init daemon to change to this runlevel using the `init 1` command, the init daemon will execute every file that starts with S or K in the /etc/rc1.d directory. Because few daemons are started in Single User Mode, most files in this directory start with a K, as shown in the following output:

```
[root@server1 ~]# ls /etc/rc1.d
K08tomcat7   K20screen-cleanup  K80ebtables   S70dns-clean
K09apache2   K20zfs-mount       K85bind9      S70pppd-dns
K20postfix   K20zfs-share       README        S90single
K20rsync     K21postgresql      S30killprocs
[root@server1 ~]#_
```

Each file in an /etc/rc#.d directory is a symbolic link to an executable rc script in the /etc/init.d directory that can be used to start or stop a certain daemon, depending on whether the symbolic link filename started with an S (start) or K (kill/stop).

Figure 8-5 illustrates the traditional UNIX SysV system initialization process for a system that boots to runlevel 5.

Figure 8-5 A traditional UNIX SysV system initialization process


init daemon
/etc/inittab (default runlevel = 5)
/etc/rc5.d/[SK]* (symbolic links to the rc scripts in the /etc/init.d directory)
daemons for runlevel 5 executed

Note 26

After executing the runtime configuration scripts in the /etc/rc#.d directories, the init daemon executes the /etc/rc.local shell script, if present. As a result, Linux administrators often add commands to /etc/rc.local that must be run at the end of system initialization.

Note 27

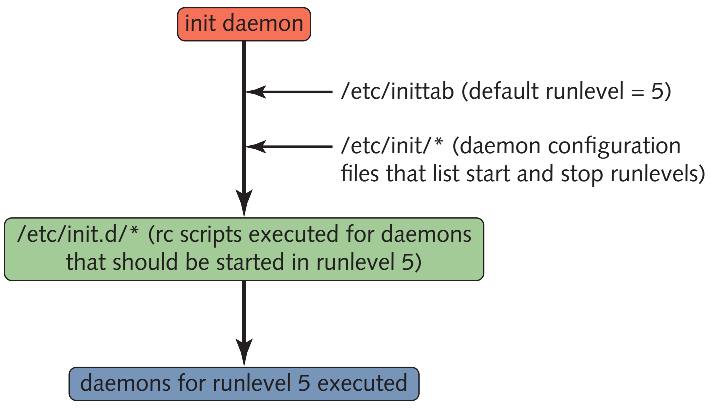
Some Linux distributions use the /etc/rc.d/rc#.d directory in place of /etc/rc#.d and use the /etc/rc.d/init.d directory in place of /etc/init.d.

On Linux systems that use the upstart init system, the /etc/rc#.d directories are not used. Instead, the init daemon identifies the default runlevel in the /etc/inittab file and then directly executes the rc scripts within the /etc/init.d directory to start or stop the appropriate daemons based on the information specified in the configuration files within the /etc/init directory. Each daemon has a separate configuration file within the /etc/init directory that uses standard wildcard notation to identify the runlevels that it should be started or stopped in. For example, the /etc/init/cron.conf file shown next indicates that the cron daemon should be started in runlevels 2, 3, 4, and 5 ([2345]) and not be stopped in runlevels 2, 3, 4, and 5 ([!2345]):

```
[root@server1 ~]# cat /etc/init/cron.conf
# cron is a standard UNIX program that runs user-specified programs at
# periodic scheduled times
description     "regular background program processing daemon"
start on runlevel [2345]
stop on runlevel [!2345]
expect fork
respawn
exec cron
[root@server1 ~]#_
```

Figure 8-6 illustrates the upstart system initialization process for a system that boots to runlevel 5.

Figure 8-6 An upstart system initialization process


init daemon
/etc/inittab (default runlevel = 5)
/etc/init/* (daemon configuration files that list start and stop runlevels)
/etc/init.d/* (rc scripts executed for daemons that should be started in runlevel 5)
daemons for runlevel 5 executed

Note 28

Some daemons are not compatible with the upstart init system. As a result, Linux distributions that use the upstart init system often host several traditional UNIX SysV daemons that are started via entries in the /etc/rc#.d directories.

Starting and Stopping Daemons Manually

Recall from the preceding section that the init daemon starts daemons at system initialization as well as starts and stops daemons afterwards when the runlevel is changed by executing the rc scripts within the /etc/init.d directory. To manipulate daemons after system startup, you can execute them directly from the /etc/init.d directory with the appropriate argument (`start`, `stop`, or `restart`). For example, to restart the cron daemon, you could run the following command:

```
[root@server1 ~]# /etc/init.d/cron restart
cron stop/waiting
cron start/running, process 3371
[root@server1 ~]#_
```

You can also use the **`service` command** to start, stop, or restart any daemons listed within the /etc/init.d directory. For example, you can restart the cron daemon using the following command:

```
[root@server1 ~]# service cron restart
cron stop/waiting
cron start/running, process 3352
[root@server1 ~]#_
```

If you modify a daemon configuration file, you often have to restart a daemon to ensure that the configuration file is reloaded by the daemon. However, some daemons allow you to simply reload configuration files without restarting the daemon; for example, to force the cron daemon to reload its configuration files, you could run the `/etc/init.d/cron reload` or `service cron reload` command. You can also see the status of a daemon at any time by using the status argument to the service command or daemon script within the /etc/init.d directory; for example, to see the status of the cron daemon, you could run the `/etc/init.d/cron status` or `service cron status` command.

The upstart init system also provides the **`stop` command** to stop a daemon, the **`start` command** to start a daemon, the **`restart` command** to restart a daemon, the **`reload` command** to reload the configuration files for a daemon, and the **`status` command** to view the status of a daemon. Thus, you could also restart the cron daemon using the following command:

```
[root@server1 ~]# restart cron
cron start/running, process 3389
[root@server1 ~]#_
```

Configuring Daemons to Start in a Runlevel

If your Linux distribution uses the upstart init system, then configuring a daemon to start or stop in a particular runlevel is as easy as modifying the associated daemon configuration file in the /etc/init directory, as shown earlier.

However, for systems that use traditional UNIX SysV system initialization, you must create or modify the symbolic links within the /etc/rc#.d directories. To make this process easier, there are commands that you can use to do this for you. The **`chkconfig` command** is available on many Linux systems, and can be used to both list and modify the runlevels that a daemon is started in. For example, the following command indicates that the postfix daemon is not started in any runlevel:

```
[root@server1 ~]# chkconfig --list postfix
postfix        0:off 1:off 2:off 3:off 4:off 5:off 6:off
[root@server1 ~]#_
```

To configure the postfix daemon to start in runlevels 3 and 5 and to verify the results, you could run the following commands:

```
[root@server1 ~]# chkconfig --level 35 postfix on
[root@server1 ~]# chkconfig --list postfix
postfix         0:off 1:off 2:off 3:on  4:off 5:on  6:off
[root@server1 ~]#_
```

You can also customize `chkconfig` to manage only the daemons you specify. For example, to remove the ability for `chkconfig` to manage the postfix daemon, you could run the `chkconfig --del postfix` command. Alternatively, the `chkconfig --add postfix` command would allow `chkconfig` to manage the postfix daemon.

Ubuntu systems use the **`update-rc.d` command** instead of `chkconfig` to configure the files within the /etc/rc#.d directories. To configure the postfix daemon, for example, you should remove any existing symbolic links within the /etc/rc#.d directories for the postfix daemon using the following command:

```
[root@server1 ~]# update-rc.d -f postfix remove
Removing any system startup links for /etc/init.d/postfix …
   /etc/rc0.d/K20postfix
   /etc/rc1.d/K20postfix
   /etc/rc2.d/S20postfix
   /etc/rc3.d/S20postfix
   /etc/rc4.d/S20postfix
   /etc/rc5.d/S20postfix
   /etc/rc6.d/K20postfix
[root@server1 ~]#_
```

Next, you can use the `update-rc.d` command to configure the appropriate symbolic links within the /etc/rc#.d directories for the postfix daemon. Because most daemons are started in runlevels 2 through 5, you can specify the `defaults` keyword to create symbolic links that start the postfix daemon in runlevels 2 through 5, as shown in the following output:

```
[root@server1 ~]# update-rc.d postfix defaults
Adding system startup for /etc/init.d/postfix …
   /etc/rc0.d/K20postfix -> ../init.d/postfix
   /etc/rc1.d/K20postfix -> ../init.d/postfix
   /etc/rc6.d/K20postfix -> ../init.d/postfix
   /etc/rc2.d/S20postfix -> ../init.d/postfix
   /etc/rc3.d/S20postfix -> ../init.d/postfix
   /etc/rc4.d/S20postfix -> ../init.d/postfix
   /etc/rc5.d/S20postfix -> ../init.d/postfix
[root@server1 ~]#_
```

Alternatively, you can specify to start the postfix daemon only in runlevels 2 and 5 using the following command, which creates the rc scripts using a sequence number of 20:

```
[root@server1 ~]# update-rc.d postfix start 20 2 5 . stop 90 1 3 4 6 .
postfix Default-Start values (2 3 4 5)
postfix Default-Stop values (0 1 6)
 Adding system startup for /etc/init.d/postfix …
   /etc/rc1.d/K90postfix -> ../init.d/postfix
   /etc/rc3.d/K90postfix -> ../init.d/postfix
   /etc/rc4.d/K90postfix -> ../init.d/postfix
   /etc/rc6.d/K90postfix -> ../init.d/postfix
   /etc/rc2.d/S20postfix -> ../init.d/postfix
   /etc/rc5.d/S20postfix -> ../init.d/postfix
[root@server1 ~]#_
```

Working with the Systemd System Initialization Process

Like the UNIX SysV init daemon, the Systemd init daemon is used to start daemons during system initialization as well as start and stop daemons after system initialization. However, Systemd can also be used to start, stop, and configure many other operating system components. To Systemd, each operating system component is called a unit. Daemons are called **service units** because they provide a system service, and runlevels are called **target units** (or **targets**). By default, each target maps to a UNIX SysV runlevel:

- Poweroff.target is the same as Runlevel 0.
- Rescue.target is the same as Runlevel 1 (Single User Mode).
- Multi-user.target is the same as Runlevel 2, 3, and 4.
- Graphical.target is the same as Runlevel 5.
- Reboot.target is the same as Runlevel 6.

Note 29

The graphical.target first loads all daemons from the multi-user.target.

Note 30

For ease, many Linux distributions create shortcuts to targets named for the UNIX SysV runlevel; for example, runlevel5.target is often a shortcut to graphical.target, and runlevel1.target is often a shortcut to rescue.target.

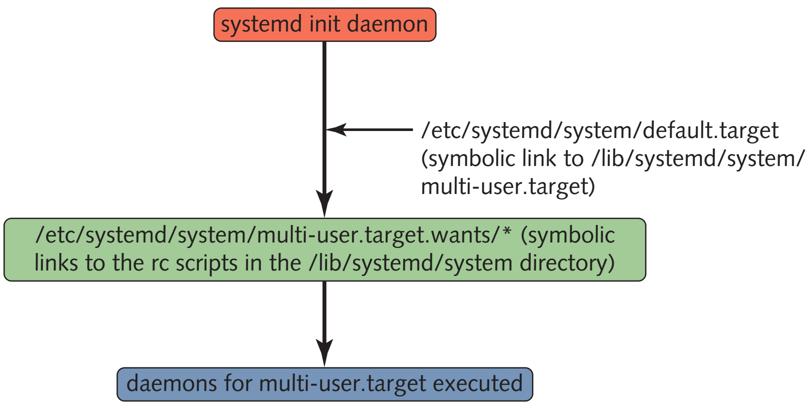
The default target on a system that has a desktop environment installed is the graphical.target. To configure your system to instead boot to the multi-user.target, you can update the /etc/systemd/system/default.target symbolic link to point to the correct runlevel using the following command:

```
[root@server1 ~]# ln -s /lib/systemd/system/multi-user.target
/etc/systemd/system/default.target
[root@server1 ~]#_
```

Most of the service units that are used by Systemd to start and stop daemons are stored in the /lib/systemd/system directory and called *daemon*.service, where *daemon* is the name of the daemon. For example, the script for the cron daemon on a Fedora system is /lib/systemd/system/crond.service. To ensure that the cron daemon is started when entering the multi-user.target, you can create a symbolic link to /lib/systemd/system/crond.service called /etc/systemd/system/multi-user.target.wants/crond.service (i.e., the multi-user target wants the crond daemon).

Figure 8-7 illustrates the Systemd system initialization process for a system that boots to multi-user.target.

Figure 8-7 A Systemd system initialization process


systemd init daemon
/etc/systemd/system/default.target (symbolic link to /lib/systemd/system/multi-user.target)
/etc/systemd/system/multi-user.target.wants/* (symbolic links to the rc scripts in the /lib/systemd/system directory)
daemons for multi-user.target executed

Note 31

The /lib/systemd/system directory stores the default configuration files for Systemd. These files are overridden by the same files in the /etc/systemd/system directory. Thus, to preserve default Systemd configuration settings, you should only create or modify files under /etc/systemd/system.

Note 32

On many Linux distributions, the /usr/lib/systemd and /lib/systemd directories are hard linked and contain identical contents as a result.

To start and stop daemons, as well as configure them to automatically start during system initialization, you can use the **`systemctl` command**. To start, stop, or restart a Systemd daemon, you can specify the appropriate action (start, stop, or restart) and name of the service unit as arguments. For example, to restart the cron daemon on Fedora Linux, you could use the following command:

```
[root@server1 ~]# systemctl restart crond.service
[root@server1 ~]#_
```

To reload the configuration files for the cron daemon on Fedora Linux, you could run the `systemctl reload crond.service` command, and to force Systemd to reload all units, you could run the `systemctl daemon-reload` command. You can also use the `systemctl` command to see detailed information about a particular Systemd daemon. For example, `systemctl status crond.service` would show detailed information about the cron daemon on Fedora Linux.

Without arguments, the `systemctl` command displays a list of all units that are currently executing in memory and their status. You can narrow this list to only services by piping the results to the `grep service` command. Moreover, to see all possible services regardless of whether they are loaded into memory or not, you can add the `–a` (or `--all`) option to the `systemctl` command. The following command displays all Systemd services and their state (dead indicates that it is currently not running):

```
[root@server1 ~]# systemctl –a | grep service | less
abrt-ccpp.service       loaded active exited    Install ABRT coredump
abrt-oops.service       loaded active running   ABRT kernel log watcher
abrt-vmcore.service     loaded active exited    Harvest vmcores for ABRT
abrtd.service           loaded active running   ABRT Automated Bug Repo
acpid.service           loaded active running   ACPI Event Daemon
alsa-store.service      loaded inactive dead    Store Sound Card State
arp-ethers.service      loaded inactive dead    Load static arp entries
atd.service             loaded active running   Job spooling tools
auditd.service          loaded active running   Security Auditing
avahi-daemon.service    loaded active running   Avahi mDNS/DNS-SD Stack
crond.service           loaded active running   Command Scheduler
cups.service            loaded active running   CUPS Printing Service
dbus-org.bluez.service error  inactive dead     dbus-org.bluez.service
dbus.service            loaded active running   D-Bus System Message Bus
:
```

You can also use the **`systemd-analyze` command** to view information about Systemd units; for example, to see a list of Systemd units sorted by the time they took to load, you could run the `systemd-analyze blame | less` command.

To configure a Systemd daemon to start in the default target (e.g., multi-user.target), you can use the `enable` argument to the `systemctl` command. For example, to ensure that the cron daemon is started at system initialization, you can run the `systemctl enable crond.service` command, which creates the appropriate symbolic link to /lib/systemd/system/crond.service within the /etc/systemd/system/multi-user.target.wants directory. Alternatively, the `systemctl disable crond.service` would prevent the cron daemon from starting when your system is booted by removing the symbolic link to /lib/systemd/system/crond.service within the /etc/systemd/system/multi-user.target.wants directory. However, another daemon started in the default target that depends on the cron daemon could potentially start the cron daemon. To fully ensure that the cron daemon cannot be started, you could run the `systemctl mask crond.service` command, which redirects the symbolic link for the cron daemon in /etc/systemd/system/multi-user.target.wants to /dev/null.

The `systemctl` command can also be used to change between targets. You can switch to multi-user.target (which is analogous to runlevel 3) by running either the `systemctl isolate multi-user.target` or `systemctl isolate runlevel3.target` command. Alternatively, you can switch to graphical.target (which is analogous to runlevel 5) by running the `systemctl isolate graphical.target` or `systemctl isolate runlevel5.target` command.

You can also create service-specific environment variables that Systemd will load into memory when it starts a particular service. To create a variable called MYVAR with the value SampleValue for use with the cron daemon on Fedora Linux, you could use the command `systemctl edit crond.service` and add the line `Environment="MYVAR=SampleValue"` into the nano editor and save your changes when finished. This line will be stored in the /etc/systemd/system/crond.service.d/override.conf file and executed each time Systemd starts the cron daemon.

Working with Systemd Unit Files

The text files under the /lib/systemd/system and /etc/systemd/system directories that provide the configuration used by Systemd are called **unit files**. You can modify unit files to alter the actions that Systemd performs or create unit files to configure additional Systemd functionality.

The following output examines sample contents of the graphical.target unit file:

```
[root@server1 ~]# cat /lib/systemd/system/graphical.target
[Unit]
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.target
After=multi-user.target display-manager.service
AllowIsolate=yes
[root@server1 ~]#_
```

The graphical.target unit file has a single `[Unit]` section that:

- `Requires` all services from the multi-user target be started; otherwise, the system will fail to switch to graphical.target.
- `Wants` the display-manager service to be started; if display-manager cannot be started, switching to the graphical target will still succeed.
- Cannot be run at the same time (`Conflicts`) as the rescue target.
- Instructs Systemd to start the multi-user target and display-manager service before entering (`After`) the graphical target (`Before` would start these services after entering the graphical target).
- Allow users to switch to the target (`AllowIsolate`) using the `systemctl` command following system initialization.

This configuration ensures that when Systemd switches to graphical.target, it will first execute the service unit files in the /lib/systemd/system/multi-user.target.wants and /etc/systemd/system/multi-user.target.wants directories. If any of these rc scripts unit files fail to execute, Systemd will not switch to graphical.target. Next, Systemd will execute the /etc/systemd/system/display-manager.service unit file. Regardless of whether the display-manager unit file executed successfully, Systemd will then execute the rc script unit files in the /lib/systemd/system/graphical.target.wants and /etc/systemd/system/graphical.target.wants directories to complete the switch to graphical.target.

Each service unit also has a service unit file that specifies the daemon it needs to start, the targets it should start and stop in, as well as any other necessary options. The following output examines sample contents of the crond.service unit file:

```
[root@server1 ~]# cat /lib/systemd/system/crond.service
[Unit]
Wants=network-online.target
After=network-online.target auditd.service systemd-user-
sessions.service

[Service]
Type=simple
User=root
EnvironmentFile=/etc/sysconfig/crond
```

```
ExecStart=/usr/sbin/crond
ExecStop=/bin/kill -INT $MAINPID
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=30s

[Install]
WantedBy=multi-user.target
[root@server1 ~]#_
```

The crond.service unit file has a `[Unit]`, `[Service]`, and `[Install]` section that instructs Systemd to:

- Ensure that the network is fully started before starting the crond service (`Wants=network-online.target`).
- Start the network-online, auditd, and systemd-user-sessions services before starting (`After`) the crond service.
- Treat the crond service as a regular daemon process (`simple`) started as the `root` user.
- Load crond-specific environment settings from a file (`EnvironmentFile`).
- Use specific commands to start, stop, and restart the crond service (`ExecStart`, `ExecStop`, and `ExecReload`).
- Restart the crond service 30 seconds after a failure (`RestartSec=30s`).
- Start the crond service when entering the multi-user target (`WantedBy`).

Note 33

Service unit files are also used to start daemons that provide server services on the network. To ensure that a daemon is only started when the system receives an associated request from a computer on the network, you can instead configure a **socket unit** file. For example, a telnet.socket unit file would instruct Systemd to start a telnet server daemon only when the system receives a telnet request from a computer on the network.

In addition to controlling the system initialization process using target units and service units, you can configure **timer units** to run programs periodically or **mount units** to mount filesystems.

Timer units are an alternative to the cron daemon (discussed in Chapter 9) that Linux systems use for scheduling programs. They are often used to schedule low-level system maintenance tasks on modern Linux systems. For example, many Linux systems use a timer unit to periodically run the **`fstrim` command** on all physical SSDs in the system to reclaim unused blocks and improve performance. Following is an example fstrim.timer unit file that instructs Systemd to automatically run fstrim.service (which in turn runs the `fstrim` command) Monday through Friday at 11:30 pm and 3600 seconds after system boot, but only if the Linux system is not run within a virtual machine or container:

```
[root@server1 ~]# cat /etc/systemd/system/fstrim.timer
[Unit]
ConditionVirtualization=!container

[Timer]
Unit=fstrim.service
OnCalendar=Mon..Fri 23:30
OnBootSec=3600
```

```
[Install]
WantedBy=timers.target
[root@server1 ~]#_
```

Mount units are an alternative to /etc/fstab for mounting filesystems. Following is an example data.mount unit file that instructs Systemd to mount an ext4 filesystem (identified by UUID) to the /data directory when the system enters the graphical.target. After creating a new mount unit, you must ensure that it is configured to start in the desired target using the `systemctl enable` command.

```
[root@server1 ~]# cat /etc/systemd/system/data.mount
[Unit]
Description=Mount data filesystem in graphical target

[Mount]
What=/dev/disk/by-uuid/4f07cfa5-fb5b-4511-8688-3305759d9006
Where=/data
Type=ext4
Options=defaults

[Install]
WantedBy=graphical.target
[root@server1 ~]# systemctl enable data.mount
[root@server1 ~]#_
```

Note 34

To mount a filesystem using a mount unit, you must specify the filesystem to mount using the UUID device file.

If you instead name the same mount unit file data.automount, the filesystem will only be mounted by Systemd when the /data directory is accessed for the first time by a user.

You can also use the **`systemd-mount` command** and **`systemd-umount` command** to manually mount and unmount filesystems using Systemd, respectively. For example, the `systemd-mount -A /dev/sda5 /data` command will instruct Systemd to mount /dev/sda5 to the /data directory, but only when the /data directory is accessed for the first time by a user (`-A`).

The X Windows System

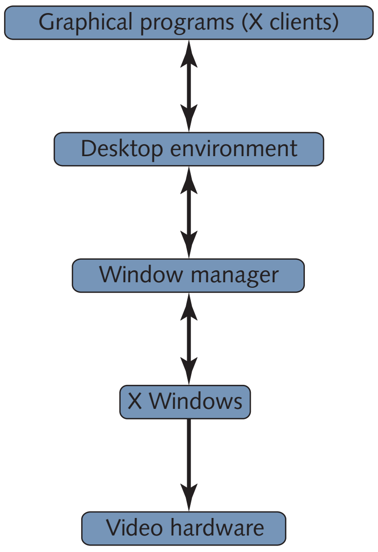
This chapter has so far focused on performing tasks using the command-line shell interface. Although most Linux administrators perform tasks exclusively within a shell, other Linux users use a GUI for running graphical programs. Thus, you need to understand the components that make up the Linux GUI. Additionally, you need to know how to start, stop, and configure them.

Linux GUI Components

The Linux GUI was designed to function consistently, no matter what video adapter card and monitor are installed on the computer system. It is composed of many components, each of which works separately from the video hardware.

A Linux installation usually includes all the GUI components listed in Figure 8-8. Together, these GUI components and related programs use over 4 GB of storage space on a typical Linux installation.

Figure 8-8 Components of the Linux GUI



X Windows

Recall from Chapter 1 that the core component of a Linux GUI is called X Windows and provides the ability to draw graphical images in windows that are displayed on a terminal screen. The programs that tell X Windows how to draw the graphics and display the results are known as **X clients**. X clients need not run on the same computer as X Windows; you can use X Windows on one computer to send graphical images to an X client on an entirely different computer by changing the DISPLAY environment variable discussed in Chapter 7. Because of this, X Windows is sometimes referred to as the server component of X Windows, or simply the **X server**.

X Windows was jointly developed by Digital Equipment Corporation (DEC) and the Massachusetts Institute of Technology (MIT) in 1984. At that time, it was code-named Project Athena and was released in 1985 as X Windows in hopes that a new name would be found to replace the X. Shortly thereafter, X Windows was sought by many UNIX vendors; and by 1988, MIT released version 11 release 2 of X Windows (X11R2). Since 1988, X Windows has been maintained by The Open Group, which released version 11 release 6 of X Windows (X11R6) in 1995. Since 2004, X Windows has been maintained as Open Source Software by the **X.org** Foundation.

Wayland is a new version of X Windows designed to replace X.org; it has additional security features and an architecture that makes graphical application development easier. While it is still currently in development, many recent Linux distributions use Wayland as the default X server but have X.org installed to ensure that Wayland-incompatible applications can still be used.

Note 35

To find out more about X Windows, visit the X.org Foundation's website at x.org. To learn more about Wayland, visit wayland.freedesktop.org.

Note 36

When Linux was first released, X Windows was governed by a separate license than the GPL, which restricted the usage of X Windows and its source code. As a result, early Linux distributions used an open source version of X Windows called XFree86.

Window Managers and Desktop Environments

To modify the look and feel of X Windows, you can use a **window manager**. For example, the dimensions and color of windows that are drawn on a graphical screen, as well as the method used to move windows around on a graphical screen, are functions of a window manager.

Note 37

Window managers that are compatible with Wayland are called **Wayland compositors**.

Many window managers are available for Linux, including those listed in Table 8-2.

Table 8-2 Common window managers

Window Manager	Description
Compiz	A highly configurable and expandable window manager that uses 3D acceleration to produce 3D graphical effects, including 3D window behavior and desktop cube workspaces
Enlightenment (E)	A highly configurable window manager that allows for multiple desktops with different settings
F Virtual Window Manager (FVWM)	A window manager, based on Tab Window Manager, that uses less computer memory and gives the desktop a 3D look
KWin	The window manager used by the K Desktop Environment; it is also a Wayland compositor
Lightweight X11 Desktop Environment (LXDE)	A window manager specifically designed for use on underpowered, mobile, and legacy systems
Metacity	The default window manager used by GNOME version 1 and 2
Mutter	The default window manager used by GNOME version 3 and later; it is also a Wayland compositor
i3	A tiling window manager commonly used by software developers and systems administrators that require a heavily customized desktop for working with code and commands
Sway	A tiling window manager that is used exclusively as a Wayland compositor; it is an alternative to i3 for Wayland users
Tab Window Manager (twm)	One of the oldest and most basic of window managers that is supported today; it provides the same features as early UNIX graphical desktops
Window Maker	A window manager that imitates the NeXTSTEP UNIX interface on which macOS is based

You can use a window manager alone or in conjunction with a desktop environment.

Recall from Chapter 1 that a desktop environment provides a standard look and feel for the GUI; it contains a standard set of GUI tools designed to be packaged together, including web browsers, file managers, and drawing programs. Desktop environments also provide sets of development tools, known as toolkits, that speed up the process of creating new software. As discussed in Chapter 1, the two most common desktop environments used on Linux are the K Desktop Environment (KDE) and the GNU Network Object Model Environment (GNOME).

KDE is the traditional desktop environment used on Linux systems. First released by Matthias Ettrich in 1996, KDE uses the KWin window manager and the Qt toolkit for the C++ programming language. The graphical interface components of KDE are collectively called the **Plasma Desktop**; KDE Plasma Desktop 5 is the latest version at the time of this writing.

Note 38

To learn more about KDE, visit kde.org.

The Qt toolkit included in KDE was created by a company called Trolltech in Norway in the 1990s. However, it took a while to build a following because, at the time, most open source developers preferred to develop in the C programming language instead of C++. Also, the fact that Qt was not released as Open Source Software until 1998 was a drawback, because most developers preferred source code that was freely modifiable.

As a result of the general dissatisfaction with the Qt toolkit, GNOME was created in 1997, and has since become the default desktop environment on most Linux distributions. GNOME 42 is the latest version of GNOME at the time of this writing; it uses the Mutter window manager and the GTK+ toolkit for the C programming language. The GTK+ toolkit was originally developed for the GNU Image Manipulation Program (GIMP); like the GIMP, it is open source. The graphical interface components of GNOME 42 are collectively called the **GNOME Shell**. Alternatives to the GNOME Shell have been developed that provide different features, including the mobile-focused **Unity**. There are also many modern desktop environments that are based on GNOME, including the **Cinnamon** desktop environment derived from GNOME 3, and the **MATE** desktop environment derived from GNOME 2.

Note 39

To learn more about GNOME, visit gnome.org.

KDE, GNOME, and GNOME-based desktop environments use system resources, such as memory and CPU time, to provide their graphical interface. As a result, most Linux administrators do not install a desktop environment on a Linux server. For Linux servers that require a desktop environment, many Linux administrators choose to install a lightweight desktop environment that uses very few system resources. **XFCE** is a common lightweight desktop environment used today.

Note 40

To learn more about XFCE, visit xfce.org.

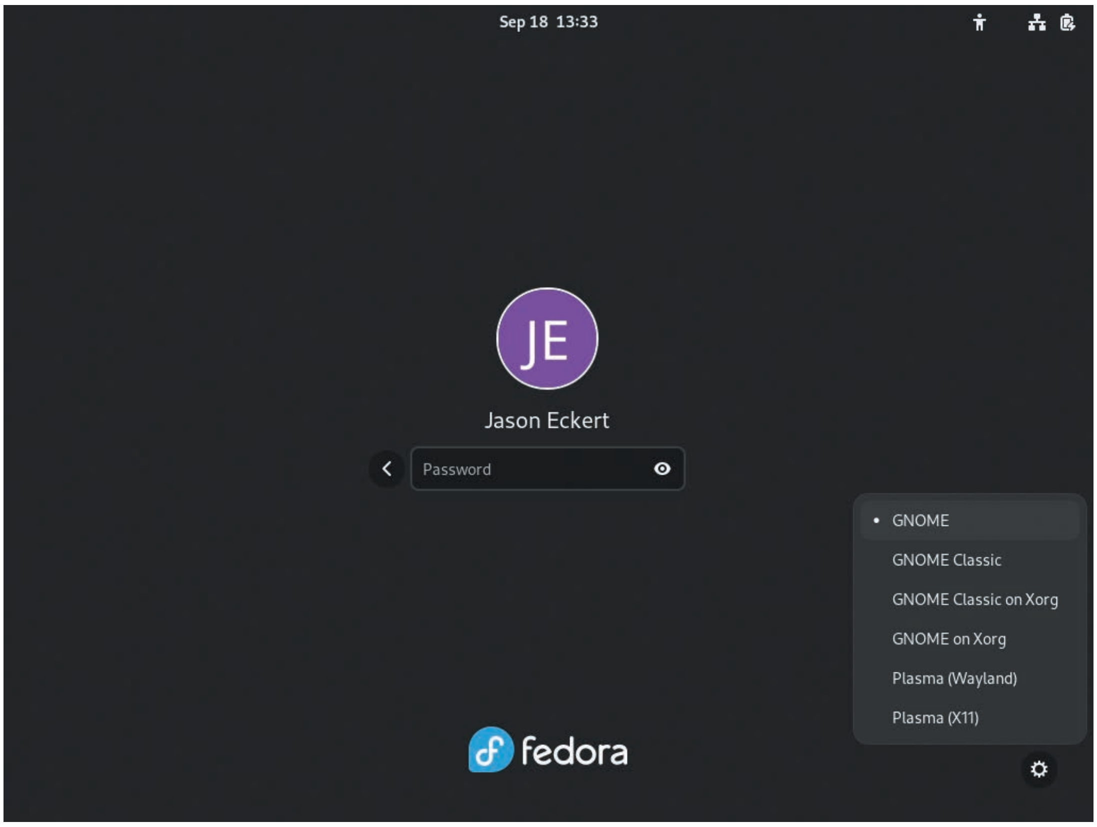
Note 41

Desktop environments are often used to run many different programs. As a result, they must provide a mechanism that allows programs to easily communicate with one another for functions such as copy-and-paste. While this functionality can be provided by Wayland compositors, most Wayland- and X.org-based systems use the **Desktop Bus (D-Bus)** software component to provide communication between graphical programs.

Starting and Stopping X Windows

As explained earlier in this chapter, when the init daemon boots to runlevel 5 or graphical.target, a program called the GNOME Display Manager (GDM) is started that displays a graphical login screen. If you click a user account within the GDM and choose the settings (cog wheel) icon, you will be able to choose from a list of installed desktop environments or window managers, as shown in Figure 8-9.

Figure 8-9 Selecting a graphical session within the GNOME Display Manager on a Fedora system



Source: Fedora Linux

Note 42

The GNOME Display Manager is a variant of the **X Display Manager (XDM)**, which displays a basic graphical login for users. Other common display managers used on Linux distributions include the **KDE Display Manager (KDM)**, the **Simple Desktop Display Manager (SDDM)**, and the **LightDM** display manager.

The default desktop environment started by the GNOME Display Manager on Fedora Linux is GNOME using Wayland. However, after you select a particular user, select Plasma using the Session menu, and the GNOME Display Manager will continue to use the KDE Plasma as the default desktop environment for the user account unless you choose otherwise.

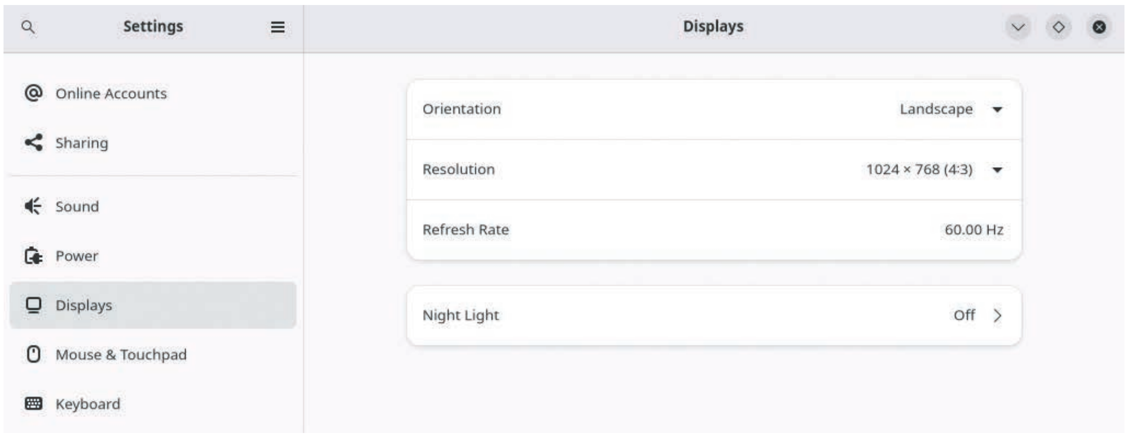
The GNOME Display Manager is the easiest way to log in and access the desktop environments that are available on your system. If, however, you use runlevel 1 (or rescue.target) or runlevel 2 through 4 (multi-user.target), the GNOME Display Manager is not started by default. In this case, you can run the **`startx` command** from a shell to start X Windows and the default window manager or desktop environment (e.g., GNOME on Wayland).

Configuring X Windows

X Windows is the component of the GUI that interfaces with the video hardware in the computer. For X Windows to perform its function, it needs information regarding the mouse, monitor, and video adapter card. This information is normally detected from the associated kernel modules that are loaded at boot time but can also be specified within configuration files. The keyboard type (e.g., English, US layout) is normally specified manually during the installation process, as shown earlier in Figure 2-8, but can be changed afterwards by accessing the Settings utility within a desktop environment, or by modifying the appropriate localization options as discussed later in this chapter.

If you use X.org, X Windows stores its configuration in the file /etc/X11/xorg.conf as well as within files stored in the /etcX11/xorg.conf.d directory. Although there is no /etc/X11/xorg.conf file in Fedora by default, it will be used if present. Instead, the keyboard type and any user-configured settings are stored in files under the /etc/X11/xorg.conf.d directory, and common settings such as the display resolution can be modified using the Settings utility within a desktop environment. In GNOME, you can navigate to Activities, Show Applications, Settings to open the Settings utility, and then select Displays to view available settings, as shown in Figure 8-10.

Figure 8-10 Selecting display settings



Orientation Landscape
Resolution 1024 × 768 (4:3)
Refresh Rate 60.00 Hz
Night Light Off

Note 43

Wayland does not have an xorg.conf equivalent; instead, all manual configuration for Wayland must be performed by the Wayland compositor.

Note 44

If you choose incompatible X Windows settings, any errors generated will be written to the ~/.xsession-errors file.

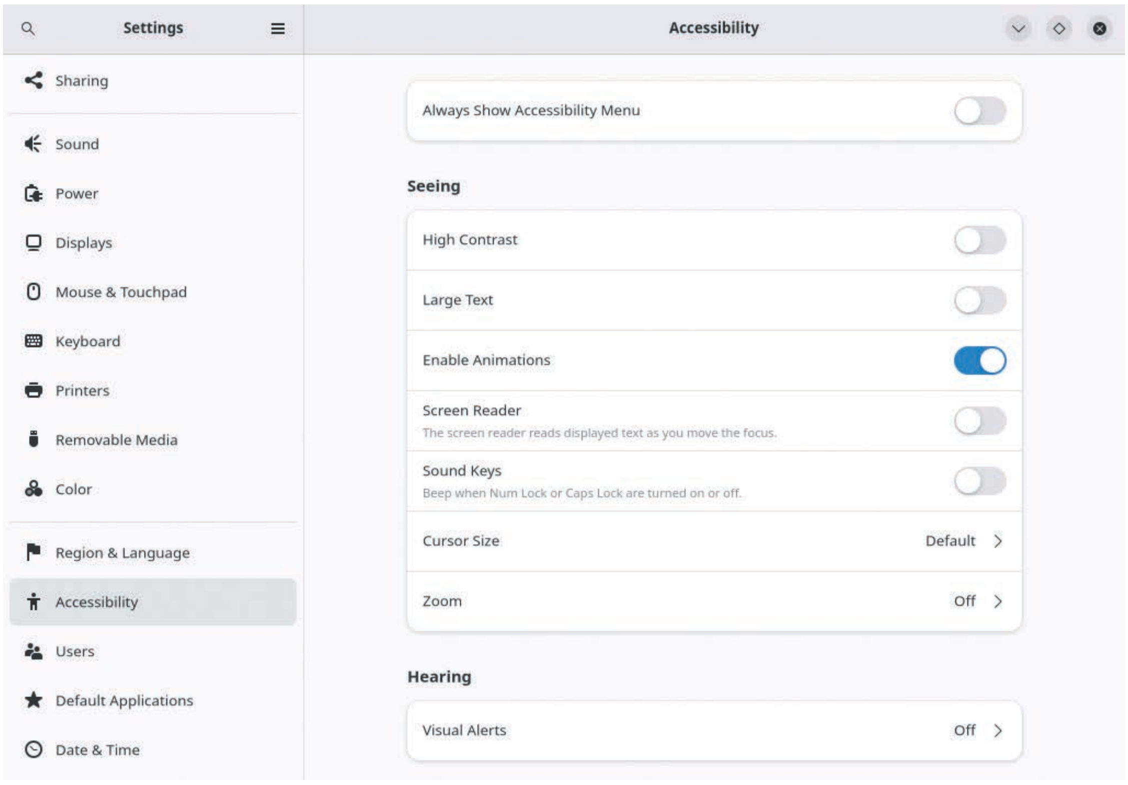
Accessibility

Many desktop environments can be configured to suit the needs of users, thus increasing the accessibility of the desktop environment. The tools used to increase accessibility are collectively called **assistive technologies**. You can configure assistive technologies using the Settings utility within a desktop environment. In GNOME, you can navigate to Activities, Show Applications, Settings to open the Settings utility, and then select Accessibility to configure assistive technologies, as shown in Figure 8-11.

If you turn on the Always Show Accessibility Menu option shown in Figure 8-11, you can enable and disable each assistive technology using the Accessibility icon in the upper-right corner of the GNOME desktop. Following is a list of the assistive technologies that you can configure within GNOME:

- High Contrast, which modifies the color scheme to suit those with low vision
- Large Text, which increases the font size of text to suit those with low vision
- Enable Animations, which allows visual effects to appear on windows and icons for easy identification

Figure 8-11 Configuring assistive technologies



Always Show Accessibility Menu

Seeing

High Contrast

Large Text

Enable Animations

Screen Reader
The screen reader reads displayed text as you move the focus.

Sound Keys
Beep when Num Lock or Caps Lock are turned on or off.

Cursor Size Default

Zoom Off

Hearing

Visual Alerts Off

- Screen Reader, which narrates the text on the active window on the screen
- Sound Keys, which beep when the Num Lock or Caps Lock keys are pressed
- Cursor Size, which enlarges the mouse pointer size for easy visibility
- Zoom, which allows you to enable magnification for parts of the screen that your mouse follows, as well as modify the color and crosshair effects during magnification
- Visual Alerts, which displays a visual alert in place of beep sounds
- Screen Keyboard, which displays an on-screen keyboard that can be used with a mouse
- Repeat Keys, which simulates multiple key presses when a single key is continually pressed
- Cursor Blinking, which adds a blinking cursor to the current text field
- Typing Assist, which provides three keyboard assistive features:
    - Sticky keys, which simulate simultaneous key presses when two keys are pressed in sequence
    - Slow keys, which add a delay following each key press
    - Bounce keys, which ignore fast duplicate key presses
- Mouse Keys, which allow the user to control the mouse using the cursor keys on the keyboard
- Locate Pointer, which will increase the size of the mouse cursor (making it easier to locate on the screen) if you move your mouse rapidly
- Click Assist, which can be used to simulate a right-click by holding down a left-click for a period of time, or trigger a left-click by hovering the mouse pointer over an area of the desktop
- Double-Click Delay, which can be used to set the time delay between two clicks that the desktop will interpret as a double-click action (e.g., to open an app)

You can also provide many other assistive technologies if your computer has the appropriate software and hardware. For example, if your system has a braille interface device, you can install braille display software that will allow it to work with the desktop environment. Similarly, you can configure voice recognition software if your system has an audio microphone, or gesture recognition software if your system has a web camera or trackpad.

Localization

Localization refers to the collective settings on a system that are specific to a specific region within the world. For example, a Linux system within Toronto, Canada will have different settings compared to a Linux system in Paris, France, including time, time zone, language, character set, and keyboard type. Localization settings are normally chosen during the installation process of the Linux distribution (as shown earlier in Figures 2-7 and 2-8) but can be changed afterwards.

Time Localization

The Linux kernel does not store time information in a format that represents the year, month, day, hour, minute and second as in the output of the `date` command in Chapter 2. Instead, the Linux kernel stores time as the number of seconds since January 1, 1970 UTC (the birth of UNIX); this is called **epoch time** and can be shown by supplying arguments to the date command, as shown in the following output:

```
[root@server1 ~]# date +%s
1535823499
[root@server1 ~]#_
```

By default, the system obtains the current time from the BIOS on your system. You can view or modify the time within the BIOS using the **`hwclock` command**. Without arguments, `hwclock` simply prints the time from the BIOS, but you can specify options to modify the BIOS time as well. For example, the `hwclock --set --date='2023-08-20 08:16:00'` would set the BIOS time to 8:16 AM on August 20, 2023.

Alternatively, you can set the Linux time using the `date` command. For example, the command `date -s "20 AUG 2023 08:16:00"` would set the Linux time to 8:16 AM on August 20, 2023. You could then use the `hwclock -w` command to set the BIOS time to match the current Linux time to ensure that the correct time is loaded from the BIOS at boot time.

Note 45

Recall from Chapter 1 that you can also obtain time from a server across a network, such as the Internet, using the Network Time Protocol (NTP). The configuration of NTP will be discussed in Chapter 13.

Time information is dependent on the regional time zone. For example, 8:16 AM in Toronto, Canada is 2:16 PM in Paris, France. Time zone information is stored in the binary /etc/localtime file, which contains the rules for calculating the time based on your time zone relative to epoch time. This file is normally a copy of, or symbolic link to, the correct file under the /usr/share/zoneinfo directory, which stores all zone information files, as shown below:

```
[root@server1 ~]# ll /etc/localtime
lrwxrwxrwx. 1 root root 37 Jul  8 20:13 /etc/localtime ->
../usr/share/zoneinfo/America/Toronto
[root@server1 ~]#_
```

To change the system time zone, you can copy the appropriate time zone information file from the /usr/share/zoneinfo directory to /etc/localtime or modify the /etc/localtime symbolic link to point to the correct zone information file under the /usr/share/zoneinfo directory.

In addition to /etc/localtime, some distributions also have an /etc/timezone text file that contains a reference to the correct time zone file path underneath the /usr/share/zoneinfo directory for use by certain applications, as shown in the following output:

```
[root@server1 ~]# cat /etc/timezone
America/Toronto
[root@server1 ~]#_
```

You can also override the default system time zone within your current shell by modifying the contents of the TZ variable. For example, running the `export TZ='America/Toronto'` command would ensure that any commands that you run in your shell display time information using the Toronto, Canada time zone. To ensure that the commands within a particular shell script use a particular time zone, set the TZ variable at the beginning of the shell script under the hashpling line. If you are unsure which time zone file you should use with the TZ variable (e.g., America/Toronto), you can run the **`tzselect` command**, which will prompt you to answer a series of questions to determine the correct time zone file name.

The **`timedatectl` command** can also be used to view and set both the time and time zone information on your system. Without arguments, `timedatectl` displays system time information. However, you could run the `timedatectl set-time "2023-08-20 08:16:00"` command to set the system time to 8:16 AM on August 20, 2023, or the `timedatectl set-timezone 'America/Toronto'` command to set the time zone to Toronto, Canada. You can also use the `timedatectl set-ntp true` command to ensure that time and time zone information is obtained using NTP.

Format Localization

In addition to time localization, different regions may have different formats used to represent data. This primarily includes the language, but also includes conventions used by the region. For example, while North America often expresses money in the format $4.50 (four dollars and fifty cents), Europe would express the same format as 4,50$. Similarly, Asia typically represents dates in the form YYYYMMDD (Year, Month, Day), whereas the United States typically represents dates in the form MMDDYYYY (Month, Day, Year).

Different languages may also have different character sets that must be represented by software as well as mapped to keyboard keys for input. The **American Standard Code for Information Interchange (ASCII)** character set used on early computers of the 1960s was specific to English characters and thus had no international localization options. It was extended to include some other languages with the introduction of the **ISO-8859** standard, but still lacked many of the extended characters used by these languages. A new standard called **Unicode** extends ASCII to allow for the representation of characters in nearly all languages used worldwide, and the **UTF-8** character set allows software to use one to four 8-bit bytes to represent the characters defined by Unicode.

Note 46

You can convert data between different character sets using the **`iconv` command**.

Moreover, the same language may have slightly different character sets, keyboard layouts, and formats for different regions; for example, a Canadian French keyboard will differ from a European French keyboard, and Canadian French date and money formats differ from European French date and money formats. As a result, the format localization on most systems includes a language, region, and character set, and is referred to as a **locale**. For example, the en_US.UTF-8 locale represents US regional English, with a UTF-8 character set. If the character set is omitted from the locale, the ASCII character set is assumed; for example, en_US represents US regional English, with an ASCII character set.

Some Linux distributions pass the correct locale to the Linux kernel as it is loaded by the GRUB2 boot loader using the LANG option on the kernel line of the GRUB2 configuration file. This instructs Linux to set the LANG variable following system startup to the correct locale. You can also view the contents of /proc/cmdline to see if your Linux kernel was loaded with the LANG option:

```
[root@server1 ~]# cat /proc/cmdline
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-5.17.5-300.fc36.x86_64
root=UUID=9ff25add-035b-4d26-9129-a9da6d0a6fa3 ro rhgb quiet
LANG=en_US.UTF-8
[root@server1 ~]#_
```

If the LANG variable was not created by the Linux kernel during boot, it is loaded from a file; on Ubuntu systems, the LANG variable is loaded from /etc/default/locale, and on Fedora systems it is loaded from /etc/locale.conf as shown below:

```
[root@server1 ~]# cat /etc/locale.conf
LANG="en_US.UTF-8"
[root@server1 ~]#_
```

You can display the values for locale variables using the **`locale` command** as shown below:

```
[root@server1 ~]# locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
[root@server1 ~]#_
```

Note from the previous output that the value of the LANG variable is set to en_US.UTF-8. Also note that other format-specific locale variables exist and are set to use the value of LANG by default. This allows users to override the default locale for a particular format type, such as units of measurement. For example, to ensure that operating system components and applications use the Canadian metric system for measurement, you could add the line `export LC_MEASUREMENT="en_CA.UTF-8"` to a shell environment file; this will use Canadian regional English with a UTF-8 character set for measurement, and US regional English for all other formats. Alternatively, to ensure that the same locale is used for all format types, you can set the value of the LC_ALL variable, which overrides LANG and all other LC variables. To see a list of all locales that you can use, run the `locale -a` command.

Note 47

The C locale is a standard locale that all UNIX and Linux systems can use. Many Linux administrators add `export LANG=C` at the beginning of shell scripts (underneath the hashpling line) to ensure that the stdout of any commands within the shell script is not dependent on the locale of the system that executes it. Most shell scripts within a Linux distribution use the C locale to ensure that any stdout can be compared to standardized online documentation for troubleshooting and learning purposes.

You can also use the **`localectl` command** to view and change locale settings on your system. Without arguments, `localectl` displays the current locale. To see a list of available locales, you can run the `localectl list-locales` command, and to set the default locale to en_CA.UTF-8, you can run the `localectl set-locale LANG=en_CA.UTF-8` command.

Normally, the keyboard type on a system matches the language and region within the locale (e.g., en_US for US English keyboard, or en_CA for Canadian English keyboard). However, there may be times when you must choose a layout that varies from the regional standard. For example, to ensure that your Linux system can use a US English Apple Macintosh keyboard, you could use the `localectl set-keymap mac-us` command. To see a list of available keyboard types, you can use the `localectl list-keymaps` command.

Summary

- The GRUB boot loader is normally loaded from the MBR or GPT of a storage device by a standard BIOS, or from the UEFI System Partition by a UEFI BIOS. GRUB Legacy supports systems with a standard BIOS and MBR storage devices, whereas GRUB2 supports both standard and UEFI BIOS systems as well as MBR and GPT storage devices.
- After the boot loader loads the Linux kernel, a system initialization process proceeds to load daemons that bring the system to a usable state.
- There are two common system initialization processes: UNIX SysV and Systemd.
- UNIX SysV uses seven runlevels to categorize a Linux system based on the number and type of daemons loaded in memory. Systemd uses five standard targets that correspond to the seven UNIX SysV runlevels.
- The init daemon is responsible for loading and unloading daemons when switching between runlevels and targets, as well as at system startup and shutdown.
- Daemons are typically executed during system initialization via UNIX SysV rc scripts or Systemd unit files. The /etc/init.d directory contains most UNIX SysV rc scripts, and the /lib/systemd/system and /etc/systemd/system directories contain most Systemd unit files.
- In addition to managing the system initialization process using target units and service units, Systemd can use socket units to manage network daemons, timer units to schedule programs, and mount units to mount filesystems.
- You can use a variety of different commands to start, stop, and restart daemons following system initialization. The `service` command is commonly used to start, stop, and restart UNIX SysV daemons, and the `systemctl` command is commonly used to start, stop, and restart Systemd daemons.
- You can use the `chkconfig` or `update-rc.d` commands to configure UNIX SysV daemon startup at boot time, as well as the `systemctl` command to configure Systemd daemon startup at boot time.
- The Linux GUI has several interchangeable components, including the X server, X clients, window manager, and optional desktop environment. You can use assistive technologies to make desktop environments more accessible to users.
- X Windows is the core component of the Linux GUI that draws graphics to the terminal screen. It comes in one of two open source implementations today: X.org and Wayland. The hardware information required by X Windows is automatically detected but can be modified using several utilities.
- You can start the Linux GUI from runlevel 3 by typing `startx` at a command prompt, or from runlevel 5 via a display manager, such as GDM.
- Linux has many region-specific settings, including time, date, and locale. Locale settings provide region-specific formats, language, and character support; they can be set for an entire system, or for a specific shell or shell script.

Key Terms

active partition
American Standard Code for Information Interchange (ASCII)
assistive technologies
boot loader
`chkconfig` command
Cinnamon
Desktop Bus (D-Bus)
`dracut` command
epoch time
`fstrim` command
GNOME Display Manager (GDM)
GNOME Shell
GRand Unified Bootloader (GRUB)
GRand Unified Bootloader version 2 (GRUB2)
GRUB Legacy
GRUB root partition
`grub-install` command
`grub2-install` command
`grub2-mkconfig` command
`hwclock` command
`iconv` command
`init` command
initialize (init) daemon
initramfs
initstate
ISO-8859
KDE Display Manager (KDM)
kernel panic
LightDM
locale
`locale` command
`localectl` command
localization
MATE
`mkinitrd` command
mount unit
multi boot