Ubuntu Command Line Quickstart

By Andrew Hudson,Matthew Helmke,Paul Hudson,Ryan Troy

Date: Feb 3, 2011

Sample Chapter is provided courtesy of Sams Publishing.

Return to the article

This chapter looks at some of the basic commands that you need to know to be productive at the Ubuntu command line. You will find out how to get to the command line, and learn some of the commands used to navigate the file system.

In This Chapter

The Linux command line, sometimes called the terminal, is one of the most powerful tools available for computer system administration and maintenance. It is an efficient way to perform complex tasks accurately and much more easily than it would seem at a first glance. Knowledge of the commands available to you and also how to string them together will make using Ubuntu even easier for many tasks. Many of the commands were created by the GNU Project as free software analogs to previously existing proprietary UNIX commands. If you are interested, you can learn more about the GNU Project at www.gnu.org/gnu/thegnuproject.html.

This chapter looks at some of the basic commands that you need to know to be productive at the command line. You will find out how to get to the command line, and learn some of the commands used to navigate the file system. Later in the book we present a Command Line Masterclass (Chapter 30), which explores the subject in greater depth. The skills you learn in this chapter will help you get started using the command line with confidence.

What Is the Command Line?

If you spend any amount of time with experienced Linux users, you will hear them mention the command line. Some, especially those who have begun their journey in the Linux world using distributions, like Ubuntu, that make it easy to complete many tasks using a graphical user interface (GUI), may speak with trepidation about the mysteries of the text interface. Others will either praise its power or comment about doing something via the command line as if it is the most natural and obvious way to complete a task.

It is not necessary for you to embrace either extreme. You may develop an affinity for the command line when performing some tasks and prefer the GUI for others. This is where most users end up today. Some may say that you will never need to access the command line because Ubuntu offers a slew of graphical tools that enable you to configure most things on your system. There are some good reasons to acquire a fundamental level of comfort with the command line that you should consider before embracing that view.

Sometimes things go wrong, and you may not have the luxury of a graphical interface to work with. In these situations, a fundamental understanding of the command line and its uses can be a real lifesaver. Also, some tasks end up being far easier and faster to accomplish from the command line.

NOTE

Don't be tempted to skip over this chapter as irrelevant. You should take the time to work through the chapter and ensure that you are comfortable with the command line before moving on. Doing so will benefit you greatly for years to come.

Initially, some may be tempted to think of the command line as the product of some sort of black and arcane art, and in some ways it can appear to be extremely difficult and complicated to use. However, with a little perseverance, by the end of this chapter you will start to feel comfortable using the command line and you'll be ready to move on to Chapter 35, "Command Line Masterclass."

More importantly, though, you will be able to make your way around a command line–based system, which you are likely to encounter if you ever work with a Linux server since many Linux servers have no GUI and all administration is done using a command line interface via SSH.

This chapter introduces you to commands that enable you to do the following:

- **Perform routine tasks—**Logging in and out, changing passwords, listing and navigating file directories
- **Implement basic file management—**Creating files and folders, copying or moving them around the file system, renaming and deleting them
- **Execute basic system management—**Shutting down or rebooting, using text-based tools to edit system configuration files, and reading man pages, which are entries for commands included as files already on your computer in a standardized manual format

The information in this chapter is valuable for individual users or system administrators who are new to Linux and are learning to use the command line for the first time.

TIP

Those of you who have used a computer for many years will probably have come into contact with MS-DOS, in which case being presented with a black screen will fill you with a sense of nostalgia. Don't get too comfy; the command line in Linux is different from (and actually more powerful than) its distant MS-DOS cousin. Even cooler is that, whereas MS-DOS skills are transferable only to other MS-DOS environments, the skills that you learn at the Linux command line can be transferred easily to other UNIX and UNIX-like operating systems, such as Solaris, OpenBSD, FreeBSD, and even Mac OS X, which provides access to the terminal.

User Accounts

A good place to start is with the concept of user-based security. For the most part, only two types of people will access the system as users (although there will be other accounts that run programs and processes, here we are talking about accounts that represent human beings rather than something like an account created for a web server process). Most people have a regular user account. These users can change anything that is specific to their account, such as the wallpaper on the desktop, their personal preferences, the configuration for a program when it is run by them using their account, and so on. Note that the emphasis is on anything that is specific to them. This type of user is not able to make systemwide changes that could affect other users.

To make systemwide changes, you need to use super-user privileges, such as the account you created when you started Ubuntu for the first time (see Chapter 1, "Installing Ubuntu"). With super-user privileges you have access to the entire system and can carry out any task, even destructive ones! To help prevent this from happening, this user does not run with these powers enabled at all times, but instead spends most of the time with the look and feel of a regular user.

To use your super-user privileges from the command line you need to preface the command you want to execute with another command, `sudo`, followed by a space and the command you want to run. As a mnemonic device, some think of this as "super-user do." When you press Enter (after typing the remaining command) you will be prompted for your password, which you should type in followed by the Enter key. As usual on any UNIX-based system, the password will not appear on the screen while you are typing it as a security measure, in case someone is watching over your shoulder. Ubuntu will then carry out the command, but with super-user privileges.

An example of the destructive nature of working as the super-user can be found in the age-old example, `sudo rm -rf /`, which erases everything on your hard drive. If you enter a command using `sudo` as a regular user who does not have an account with super-user privileges, an error message will appear and nothing will happen because the command will not run. We recommend that you don't try this particular command as a test, though. If you enter this command using an account with super-user privileges, you will soon find yourself starting over with a fresh installation and hoping you have a current backup of all of your data. You need to be especially careful when using your super-user privileges; otherwise, you may do irreparable damage to your system.

However, the ability to work as the super-user is fundamental to a healthy Linux system and should not be feared, but rather respected and used only with focused attention. Without this ability you would not be able to install new software, edit system configuration files, or do a large number of important administration tasks. By the end of this chapter you will feel comfortable working with your super-user privileges and be able to adequately administer your system from the command line. By the way, you have already been performing operations with super-user privileges from the GUI if you have ever been asked to enter your password to complete a specific task, such as installing software updates. The difference is that most graphical interfaces limit the options that users have and make it a little more difficult to do some of the big, disruptive tasks, even the ones that are incredibly useful.

Ubuntu works slightly differently from many other Linux distributions. If you study some other Linux distros, especially older or more traditional ones, you will hear about a specific user account called `root`, which is a super-user account. In those distros, instead of typing in `sudo` before a command while using a regular user account with super-user privileges, you log in to the `root` account and simply issue the command without entering a password. In those cases, you can tell when you are using the root account at the command line because you will see a pound sign (`#`) in the command line prompt in the place of the dollar sign ($).

For example, `matthew@seymour:~#` vs the usual `matthew@seymour:~$`.

In Ubuntu the `root` account is disabled by default because forcing regular users with super-user privileges to type a specific command every time they want to execute a command as a super-user should have the benefit of making them carefully consider what they are doing when they use that power. It is easy to forget to log out of a root account, and entering a powerful command while logged in to `root` can be catastrophic. However, if you are more experienced and comfortable with the more traditional method of using super-user privileges and want to enable the root account, you can use the command `sudo passwd`. When prompted, enter your user password to confirm that your user account has super-user privileges. You will then be asked for a new UNIX password, which will be the password for the root account, so make sure and remember it. You will also be prompted to repeat the password, in case you've made any mistakes. After you've typed it in and pressed Enter, the `root` account will be active. You'll find out how to switch to `root` later on.

An alternative way of getting a root prompt, without having to enable the root account, is to issue the command `sudo -i`. After entering your password you will find yourself at a root prompt (`#`). Do what you need to do and when you are finished, type `exit` and press Enter to return to your usual prompt. You can learn more about `sudo` and `root` from an Ubuntu perspective at https://help.ubuntu.com/community/RootSudo.

Ubuntu offers a number of ways to access the command line. One way is to press the key combination Ctrl+Alt+F1, after which Ubuntu will switch to a black screen and a login prompt like this:

```
Ubuntu 10.10 maverick seymour tty1
seymour login:
```

TIP

This is one of six virtual consoles that Ubuntu provides. After you have accessed a virtual console, you can use the Ctrl+Alt + any of F1 through F6 to switch to a different console. If you want to get back to the graphical interface, press Ctrl+Alt+ F7. You can also switch between consoles by holding the Alt key and pressing either the left or the right cursor key to move down or up a console, such as tty1 to tty2.

At the login prompt enter your username and press the Enter key. You will be asked for your password, which you should enter. Note that Ubuntu does not show any characters while you are typing your password in. This is a good thing because it prevents any shoulder surfers from seeing what you've typed or the length of the password.

Pressing the Enter key drops you to a shell prompt, signified by the dollar sign:

```
matthew@seymour:~$
```

This particular prompt tells me that I am logged in as the user `matthew` on the system `seymour` and I am currently in my home directory; Linux uses the tilde (~) as shorthand for the home directory, which would usually be something like `/home/matthew`.

TIP

Navigating through the system at the command line can get confusing at times, especially when a directory name occurs in several places. Fortunately, Linux includes a simple command that tells you exactly where you are in the file system. It's easy to remember because the command is just an abbreviation of present working directory, so type **`pwd`** at any point to get the full path of your location. For example, typing **`pwd`** after following these instructions shows `/home/yourusername`, meaning that you are currently in your home directory.

Using the `pwd` command can save you a lot of frustration when you have changed directory half a dozen times and have lost track.

Another way to quickly access the terminal is to use the desktop menu option Applications, Accessories, Terminal. This opens up `gnome-terminal`, which allows you to access the terminal while remaining in a GUI environment. This time, the terminal appears as white text on an aubergine (dark purple) background. This is the most common method for most desktop users, but both are useful.

Regardless of which way you access the terminal, using the virtual tty consoles accessible at Ctrl + Alt + F1-6 or via the windowed version atop your GUI desktop, you will find the rest of the usage details that we cover work the same. As you continue to learn and experiment beyond the contents of this book, you may start to discover some subtle differences between the two and develop a preference. For our purposes, either method will work quite well.

With that, let's begin.

Navigating Through the File System

Use the `cd` command to navigate the file system. This command is generally used with a specific directory location or pathname, like this:

```
matthew@seymour:~$ cd /etc/apt/
```

The `cd` command can also be used with several shortcuts. For example, to quickly move up to the *parent* directory, the one above the one you are currently in, use the cd command like this:

```
matthew@seymour:~$ cd ..
```

To return to your home directory from anywhere in the Linux file system, use the cd command like this:

```
matthew@seymour:~$ cd
```

You can also use the `$HOME` shell environment variable to accomplish the same thing. Type this command and press Enter to return to your home directory:

```
matthew@seymour:~$ cd $HOME
```

You can accomplish the same thing by using the tilde (~) like this:

```
matthew@seymour:~$ cd ~
```

Don't forget you can always use `pwd` to remind you where you are within the file system!

The `ls` command lists the contents of the current directory. It's commonly used by itself, but a number of options (also known as switches) are available for `ls` and give you more information. For instance, the following command returns a listing of all the files and directories within the current directory, including any hidden files (denoted by a `.` prefix) as well as a full listing, so it will include details such as the permissions, owner and group, size, and last modified time and date:

```
matthew@seymour:~$ ls -la
```

You can also issue the command

```
matthew@seymour:~$ ls -R
```

This command scans and lists all the contents of the subdirectories of the current directory. This might be a lot of information, so you may want to redirect the output to a text file so you can browse through it at your leisure by using the following:

```
matthew@seymour:~$ ls -laR > listing.txt
```

TIP

The previous command sends the output of `ls -laR` to a file called `listing.txt` and demonstrates part of the power of the Linux command line. At the command line you are able to use files as inputs to commands, or generate files as outputs as shown. For more information about combining commands, see Chapter 30. In the meantime, note that you can read the contents of that text file using the command `less listing.txt`, which will let you read the file bit by bit using the arrow keys to navigate in the file (or `Enter` to move to the next line), the spacebar to move to the next page, and q to exit when done.

Table 4.1 shows some of the top-level directories that are part of a standard Linux distro.

Table 4.1. Basic Linux Directories

Name	Description
/	The root directory
/bin	Essential commands
/boot	Boot loader files, Linux kernel
/dev	Device files
/etc	System configuration files
/home	User home directories
/lib	Shared libraries, kernel modules
/lost+found	Directory for recovered files (if found after a file system check)
/media	Mount point for removable media, such as DVDs and floppy disks
/mnt	Usual mount point for local, remote file systems
/opt	Add-on software packages
/proc	Kernel information, process control
/root	Super-user (root) home
/sbin	System commands (mostly root only)
/srv	Holds information relating to services that run on your system
/sys	Real-time information on devices used by the kernel
/tmp	Temporary files
/usr	Software not essential for system operation, such as applications
/var	Variable data (such as logs); spooled files

Knowing these directories can help you find files when you need them. This knowledge can even help you partition hard drives when you install new systems by letting you choose to put certain directories on their own distinct partition, which can be useful for things like isolating directories from one another, such as a server security case like putting a directory like /boot that doesn't change often on its own partition and making it read-only and unchangeable without specific operations being done by a super-user during a maintenance cycle. Desktop users probably won't need to think about that, but the directory

tree is still quite useful to know when you want to find the configuration file for a specific program and set some program options systemwide to affect all users.

Some of the important directories in Table 4.1, such as those containing user and root commands or system configuration files, are discussed in the following sections. You may use and edit files under these directories when you use Ubuntu.

Linux also includes a number of commands you can use to search the file system. These include the following:

- `whereis` *`command`* —Returns the location of the command (e.g. /bin, /sbin, or /usr/bin/command) and its man page, which is an entry for the command included as a file already on your computer in a standardized manual format.
- `whatis` *`command`* —Returns a one-line synopsis from the command's man page.
- `locate` *`file`* —Returns locations of all matching file(s); an extremely fast method of searching your system because `locate` searches a database containing an index of all files on your system. However, this database (which is several MB in size and named `mlocate.db`, under the `/var/lib/mlocate` directory) is built daily at 6:25 a.m. by default, and does not contain pathnames to files created during the workday or in the evening. If you do not keep your machine on constantly, you can run the `updatedb` command with super-user privileges to manually start the building of the database. More advanced users can change the time the database is updated by changing a `cron` job that calls the command. There is a script in `/etc/cron.daily` called `mlocate` that does this as part of the daily maintenance for the database. You would need to either remove that script from `/etc/cron.daily` and put it the root `crontab` or change when the daily tasks are run. If you don't know how to do this right now, you should wait until you have read and understand Chapter 11, "Automating Tasks," and its discussion of `cron`, after which the preceding description will be clear.
- `apropos` *`subject`* —Returns a list of commands related to subject.
- `type` *`name`* —Returns how a name would be interpreted if used as a command. This generally shows options or the location of the binary that will be used. For example, type ls returns ls is aliased to `` `ls --color=auto' ``.

Managing Files with the Shell

Managing files in your home directory involves using one or more easily remembered commands. Basic file management operations include paging (reading), moving, renaming, copying, searching, and deleting files and directories. Here are some commands to do these tasks:

- `cat` *`filename`* —Outputs contents of filename to display
- `less` *`filename`* —Allows scrolling while reading contents of filename
- `mv` *`file1 file2`* —Renames *`file1`* to *`file2`* and may be used with full pathnames to move the file to a new directory at the same time
- `mv` *`file dir`* —Moves file to specified directory
- `cp` *`file1 file2`* —*Copies* *`file1`* and creates `file2`
- `rm` *`file`* —Deletes file
- `rmdir` *`dir`* —Deletes directory (if empty)
- `grep` *`string file`* `(s)` —Searches through files(s) and displays lines containing matching string (a string is a series of characters like letters, numbers, and symbols, such as a word)

Note that each of these commands can be used with pattern-matching strings known as *wildcards* or *expressions*. For example, to delete all files in the current directory beginning with the letters `abc`, you

can use an expression beginning with the first three letters of the desired filenames. An asterisk (`*`) is then appended to match all these files. Use a command line with the `rm` command like this:

```
matthew@seymour:~$ rm abc*
```

Linux shells recognize many types of filenaming wildcards, but this is different from the capabilities of Linux commands supporting the use of more complex expressions. You learn more about using wildcards in Chapter 11, "Automating Tasks."

NOTE

You can also learn more about using expressions by reading the grep manual pages (`man grep`).

Working with Compressed Files

Another file management operation is compression and decompression of files, or the creation, listing, and expansion of file and directory archives. Linux distributions usually include several compression utilities you can use to create, compress, expand, or list the contents of compressed files and archives. These commands include:

- *`bunzip2`*—Expands a compressed file
- *`bzip2`*—Compresses or expands files and directories
- *`gunzip`*—Expands a compressed file
- *`gzip`*—Compresses or expands files and directories
- *`tar`*—Creates, expands, or lists the contents of compressed or uncompressed file or directory archives known as *tape archives* or *tarballs*

Most of these commands are easy to use. However, the `tar` command, which is the most commonly used of the bunch, has a somewhat complex set of command-line options and syntax. This flexibility and power are part of its popularity; you can quickly learn to use `tar` by remembering a few of the simple command line options. For example, to create a compressed archive of a directory, use `tar`'s `czf` options like this:

```
matthew@seymour:~$ tar czf dirname.tgz dirname
```

The result is a compressed archive (a file ending in `.tgz`) of the specified directory (and all files and directories under it). Add the letter `v` to the preceding options to view the list of files added during compression and archiving while the archive is being created. To list the contents of the compressed archive, substitute the c option with the letter `t`, like this:

```
matthew@seymour:~$ tar tzf archive
```

However, if many files are in the archive, a better invocation (to easily read or scroll through the output) is

```
matthew@seymour:~$ tar tzf archive | less
```

TIP

In the previous code example, we used a pipe character (|). Each pipe sends the output of the first command to the next command. This is another of the benefits of the command line under Linux—you can string several commands together to get the desired results.

To expand the contents of a compressed archive, use `tar`'s `zxf` options, like so:

```
matthew@seymour:~$ tar zxf archive
```

The `tar` utility decompresses the specified archive and extracts the contents in the current directory.

Essential Commands from the `/bin` and `/sbin` Directories

The `/bin` directory contains essential commands used by the system for running and booting the system. In general, only the root operator uses the commands in the `/sbin` directory. The software in both locations is essential to the system; they make the system what it is, and if they are changed or removed, it could cause instability or a complete system failure. Often, the commands in these two directories are *statically* linked, which means that the commands do not depend on software libraries residing under the `/lib` or `/usr/lib` directories. Nearly all the other applications on your system are *dynamically* linked—meaning that they require the use of external software libraries (also known as *shared* libraries) to run. This is a feature for both sets of software.

The commands in `/bin` and `/sbin` are kept stable to maintain foundational system integrity and do not need to be updated often, if at all. For the security of the system, these commands are kept in a separate location and isolated where changes are more difficult and where it will be more obvious to the system administrator if unauthorized changes are attempted or made.

Application software changes more frequently, and applications often use the same functions that other pieces of application software use. This was the genesis of shared libraries. When a security update is needed for something that is used by more than one program, it has to be updated in only one location, a specific software library. This enables easy and quick security updates that will affect several pieces of non-system-essential software at the same time by updating one shared library, contained in one file on the computer.

Using and Editing Files in the `/etc` Directory

System configuration files and directories reside under the `/etc` directory. Some major software packages, such as Apache, OpenSSH, and `xinetd`, have their own subdirectories in `/etc` filled with configuration files. Others like crontab or fstab use one file. Some examples of system-related configuration files in `/etc` include the following:

- *`fstab`*—The file system table is a text file listing each hard drive, CD-ROM, floppy, or other storage device attached to your PC. The table indexes each device's partition information with a place in your Linux file system (directory layout) and lists other options for each device when used with Linux (see Chapter 36, "Kernel and Module Management"). Nearly all entries in `fstab` can be manipulated by root using the `mount` command.
- *`modprobe.d/`*—This folder holds all the instructions to load kernel modules that are required as part of the system startup.
- *`passwd`*—The list of users for the system, including special-purpose nonhuman users like syslog and couchdb, along with user account information.
- *`sudoers`*—A list of users or user groups with super-user access.

Protecting the Contents of User Directories—`/home`

The most important data on a Linux system resides in the user's directories, found under the `/home` directory. Segregating the system and user data can be helpful in preventing data loss and making the process of backing up easier. For example, having user data reside on a separate file system or mounted from a remote computer on the network might help shield users from data loss in the event of a system hardware failure. For a laptop or desktop computer at home, you might place `/home` on a separate partition from the rest of the file system, so that if the operating system is upgraded, damaged, or reinstalled, `/home` would be more likely to survive the event intact.

Using the Contents of the `/proc` Directory to Interact with the Kernel

The content of the `/proc` directory is created from memory and exists only while Linux is running. This directory contains special files that either extract information from or send information to the kernel. Many Linux utilities extract information from dynamically created directories and files under this directory, also known as a *virtual file system*. For example, the `free` command obtains its information from a file named **`meminfo`**:

```
matthew@seymour:~$ free

             total       used       free     shared    buffers     cached
Mem:       4055680    2725684    1329996          0     188996    1551464
-/+ buffers/cache:     985224    3070456
Swap:      8787512          0    8787512
```

This information constantly changes as the system is used. You can get the same information by using the cat command to see the contents of the `meminfo` file:

```
matthew@seymour:~$ cat /proc/meminfo
MemTotal:                4055680 KB
MemFree:                 1329692 KB
Buffers:                  189208 KB
Cached:                  1551488 KB
SwapCached:                    0 KB
Active:                  1222172 KB
Inactive:                1192244 KB
Active(anon):             684092 KB
Inactive(anon):               16 KB
Active(file):             538080 KB
Inactive(file):          1192228 KB
Unevictable:                  48 KB
Mlocked:                      48 KB
SwapTotal:               8787512 KB
SwapFree:                8787512 KB
Dirty:                       136 KB
Writeback:                     0 KB
AnonPages:                673760 KB
Mapped:                   202308 KB
Shmem:                     10396 KB
Slab:                     129248 KB
SReclaimable:             107356 KB
SUnreclaim:                21892 KB
KernelStack:                2592 KB
PageTables:                30108 KB
NFS_Unstable:                  0 KB
Bounce:                        0 KB
WritebackTmp:                  0 KB
CommitLimit:            10815352 KB
```

```
Committed_AS:               1553172 KB
VmallocTotal:           34359738367 KB
VmallocUsed:                 342300 KB
VmallocChunk:           34359387644 KB
HardwareCorrupted:                0 KB
HugePages_Total:                  0
HugePages_Free:                   0
HugePages_Rsvd:                   0
HugePages_Surp:                   0
Hugepagesize:                  2048 KB
DirectMap4k:                  38912 KB
DirectMap2M:                4153344 KB
```

The `/proc` directory can also be used to dynamically alter the behavior of a running Linux kernel by "echoing" numerical values to specific files under the `/proc/sys` directory. For example, to "turn on" kernel protection against one type of denial of service (DOS) attack known as *SYN flooding*, use the echo command to send the number 1 (one) to the following `/proc` path:

```
matthew@seymour:~$ $ sudo echo 1 >/proc/sys/net/ipv4/tcp_syncookies
```

Other ways to use the /proc directory include the following:

- Getting CPU information, such as the family, type, and speed from `/proc/cpuinfo`.
- Viewing important networking information under `/proc/net`, such as active interfaces information under `/proc/net/dev`, routing information in `/proc/net/route`, and network statistics in `/proc/net/netstat`.
- Retrieving file system information.
- Reporting media mount point information via USB; for example, the Linux kernel reports what device to use to access files (such as `/dev/sda`) if a USB camera or hard drive is detected on the system. You can use the `dmesg` command to see this information.
- Getting the kernel version in `/proc/version`, performance information such as uptime in `/proc/uptime`, or other statistics such as CPU load, swap file usage, and processes in `/proc/stat`.

Working with Shared Data in the /usr Directory

The `/usr` directory contains software applications, libraries, and other types of shared data for use by anyone on the system. Many Linux system administrators give `/usr` its own partition. A number of subdirectories under `/usr` contain manual pages (`/usr/share/man`), software package shared files (`/usr/share/` *`name_of_package`* , *such as* *`/usr/share/emacs`*), additional application or software package documentation (`/usr/share/doc`), and an entire subdirectory tree of locally built and installed software, `/usr/local`.

Temporary File Storage in the /tmp Directory

As its name implies, the `/tmp` directory is used for temporary file storage; as you use Linux, various programs create files in this directory.

Accessing Variable Data Files in the /var Directory

The `/var` directory contains subdirectories used by various system services for spooling and logging. Many of these variable data files, such as print spooler queues, are temporary, whereas others, such as

system and kernel logs, are renamed and rotated in use. Incoming electronic mail is usually directed to files under `/var/spool/mail`.

Linux also uses `/var` for other important system services. These include the top-most File Transfer Protocol (FTP) directory under `/var/ftp` (see Chapter 18, "Remote File Serving with FTP"), and the Apache web server's initial home page directory for the system, `/var/www/html`. (See Chapter 17, "Apache Web Server Management," for more information on using Apache.)

NOTE

There is a recent trend to move data that is served from /var/www and /var/ftp to /srv, but this is not universal.

Logging In to and Working with Linux

You can access and use a Linux system in a number of ways. One way is at the console with a monitor, keyboard, and mouse attached to the PC. Another way is via a serial console, either by dial-up via a modem or a PC running a terminal emulator and connected to the Linux PC via a null modem cable. You can also connect to your system through a wired or wireless network, using the `telnet` or `ssh` commands. The information in this section shows you how to access and use the Linux system, using physical and remote text-based logins.

NOTE

This chapter focuses on text-based logins and use of Linux. Graphical logins and using a graphical desktop are described in Chapter 3, "Working with GNOME."

Text-Based Console Login

If you sit down at your PC and log in to a Linux system that has not been booted to a graphical login, you see a prompt similar to this one:

```
Ubuntu 10.10 maverick seymour ttyl
seymour login:
```

Your prompt might vary, depending on the version of Ubuntu you are using and the method you are using to connect. In any event, at this prompt, type in your username and press Enter. When you are prompted for your password, type it in and press Enter.

NOTE

Your password is not echoed back to you, which is a good idea. Why is it a good idea? Well, people are prevented from looking over your shoulder and seeing your screen input. It is not difficult to guess that a five-letter password might correspond to the user's spouse's first name!

Logging Out

Use the `exit` or `logout` command or Ctrl+D to exit your session. You are then returned to the login prompt. If you use virtual consoles, remember to exit each console before leaving your PC. (Otherwise, someone could easily sit down and use your account.)

Logging in and Out from a Remote Computer

Although you can happily log in on your computer, an act known as a *local* login, you can also log in to your computer via a network connection from a remote computer. Linux-based operating systems provide a number of remote access commands you can use to log in to other computers on your local area network (LAN), wide area network (WAN), or the Internet. Note that not only must you have an account on the remote computer, but the remote computer must be configured to support remote logins—otherwise, you won't be able to log in.

NOTE

See Chapter 14, "Networking," to see how to set up network interfaces with Linux to support remote network logins and Chapter 11 to see how to start remote access services (such as `sshd`).

The best and most secure way to log in to a remote Linux computer is to use `ssh`, the Secure Shell client. Your login and session are encrypted while you work on the remote computer. The `ssh` client features many command-line options, but can be simply used with the name or IP address of the remote computer, like this:

```
matthew@seymour:~$ ssh 192.168.0.41
The authenticity of host '192.168.0.41 (192.168.0.41)' can't be established.
RSA key fingerprint is e1:db:6c:da:3f:fc:56:1b:52:f9:94:e0:d1:1d:31:50.
Are you sure you want to continue connecting (yes/no)?
```

`yes`

The first time you connect with a remote computer using `ssh`, Linux displays the remote computer's encrypted identity key and asks you to verify the connection. After you type `yes` and press Enter, you are warned that the remote computer's identity (key) has been entered in a file named `known_hosts` under the `.ssh` directory in your home directory. You are also prompted to enter your password:

```
Warning: Permanently added '192.168.0.41' (RSA) to the list of known hosts.
matthew@192.168.0.41's password:
matthew@babbage~$
```

After entering your password, you can work on the remote computer, which you can confirm by noticing the changed prompt that now uses the name of the remote computer on which you are working. Again, because you are using `ssh`, everything you enter on the keyboard in communication with the remote computer is encrypted. When you log out, you will return to the shell on your computer.

```
matthew@babbage~$ logout
matthew@seymour:~$
```

Using Environment Variables

A number of in-memory variables are assigned and loaded by default when you log in. These variables are known as shell *environment variables and* can be used by various commands to get information about your environment, such as the type of system you are running, your home directory, and the shell in use. Environment variables are used to help tailor the computing environment of your system and include helpful specifications and setup, such as default locations of executable files and software libraries. If you begin writing shell scripts, you might use environment variables in your scripts. Until then, you need to be aware only of what environment variables are and do.

The following list includes a number of environment variables, along with descriptions of how the shell uses them:

- `PWD`—provides the full path to your current working directory, used by the `pwd` command, such as `/home/matthew/Documents`
- `USER`—declares the user's name, such as `matthew`
- `LANG`—sets the default language, such as English
- `SHELL`—declares the name and location of the current shell, such as `/bin/bash`
- `PATH`—sets the default location(s) of executable files, such as `/bin`, `/usr/bin`, and so on
- `TERM`—sets the type of terminal in use, such as `vt100`, which can be important when using screen-oriented programs, such as text editors
- You can print the current value of any environment variable using `echo $VARIABLENAME`, like this:

```
matthew@seymour:~$ echo $USER

matthew
matthew@seymour:~$
```

NOTE

Each shell can have its own feature set and language syntax, as well as a unique set of default environment variables. See Chapter 15, "Remote Access for SSH and Telnet," for more information about using the different shells included with Ubuntu.

You can use the `env` or `printenv` command to display all environment variables, like so:

```
matthew@seymour:~$ env
ORBIT_SOCKETDIR=/tmp/orbit-matthew
SSH_AGENT_PID=1729
TERM=xterm
SHELL=/bin/bash
WINDOWID=71303173
GNOME_KEYRING_CONTROL=/tmp/keyring-qTEFTw
GTK_MODULES=canberra-gtk-module
USER=matt
hew
SSH_AUTH_SOCK=/tmp/keyring-qTEFTw/ssh
DEFAULTS_PATH=/usr/share/gconf/gnome.default.path
SESSION_MANAGER=local/seymour:/tmp/.ICE-unix/1695
USERNAME=matthew
XDG_CONFIG_DIRS=/etc/xdg/xdg-gnome:/etc/xdg
DESKTOP_SESSION=gnome
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/home/matt
hew
GDM_KEYBOARD_LAYOUT=us
LANG=en_US.utf8
GNOME_KEYRING_PID=1677
MANDATORY_PATH=/usr/share/gconf/gnome.mandatory.path
GDM_LANG=en_US.utf8
GDMSESSION=gnome
HISTCONTROL=ignoreboth
SPEECHD_PORT=7560
SHLVL=1
HOME=/home/matt
hew
LOGNAME=matt
hew
```

```
LESSOPEN=| /usr/bin/lesspipe %s
DISPLAY=:0.0
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/var/run/gdm/auth-for-matthew-PzcGqF/database
COLORTERM=gnome-terminal
OLDPWD=/var/lib/mlocate
_=/usr/bin/env
```

This abbreviated list shows some of the common variables. These variables are set by configuration or *resource* files contained in `the /etc, /etc/skel`, or `user /home directory`. You can find default settings for bash, for example, `in /etc/profile` and `/etc/bashrc` as well as `.bashrc` or `.bash_profile` files in your home directory. Read the man page for `bash` for details about using these configuration files.

One of the most important environment variables is `$PATH`, which defines the location of executable files. For example, if, as a regular user, you try to use a command that is not located in your `$PATH` (such as the imaginary `command` command), you will see something like this:

```
matthew@seymour:~$ command
-bash: command: command not found
```

If the command that you're trying to execute exists in the Ubuntu software repositories, but is not yet installed on your system, Ubuntu will respond with the correct command to install the command.

```
matthew@seymour:~$ command

The program 'command' is currently not installed. You can install it by typing:

sudo apt-get install command
```

However, you might know that `command` is definitely installed on your system, and you can verify this by using the `whereis` command, like so:

```
$ whereis command
command: /sbin/command
```

You can also run the command by typing its full pathname or complete directory specification, like this:

```
$ /sbin/command
```

As you can see in this example, the command `command` is indeed installed. What happened is that by default, the `/sbin` directory is not in your $PATH. One of the reasons for this is that commands under the `/sbin` directory are normally intended to be run only by root. You can add `/sbin` to your $PATH by editing the file `.bash_profile` in your home directory (if you use the bash shell by default, like most Linux users). Look for the following line:

```
PATH=$PATH:$HOME/bin
```

You can then edit this file, perhaps using the `vi` editor (discussed later in this chapter), to add the `/sbin` directory like so:

```
PATH=$PATH:/sbin$HOME/bin
```

Save the file. The next time you log in, the /sbin directory is in your $PATH. One way to use this change right away is to read in the new settings in .bash_profile by using the bash shell's source command like so:

```
$ source .bash_profile
```

You can now run commands located in the /sbin directory without the need to explicitly type the full pathname.

Some Linux commands also use environment variables, for example, to acquire configuration information (such as a communications program looking for a variable such as BAUD_RATE, which might denote a default modem speed).

To experiment with the environment variables, you can modify the PS1 variable to manipulate the appearance of your shell prompt. If you are working with bash, you can use its built-in export command to change the shell prompt. For example, if your default shell prompt looks like this:

```
matthew@seymour:~$
```

You can change its appearance by using the PS1 variable like this:

```
$ PS1='$OSTYPE r00lz ->'
```

After you press Enter, you see

```
linux-gnu r00lz ->
```

Notes

See the bash man page for other variables you can use for prompt settings.

Using the Text Editors

Linux distributions include a number of applications known as *text editors* that you can use to create text files or edit system configuration files. Text editors are similar to word processing programs, but generally have fewer features, work only with text files, and might or might not support spell checking or formatting. Text editors range in features and ease of use and are found on nearly every Linux distribution. The number of editors installed on your system depends on what software packages you've installed on the system.

Some of the more popular console-based text editors include:

- *emacs*—The comprehensive GNU emacs editing environment, which is much more than an editor; see the section "Working with emacs" later in this chapter.
- *joe*—Joe's Own Editor, a text editor, which can be used to emulate other editors.
- *nano*—A simple text editor similar to the pico text editor included with the pine email program.
- *vim*—An improved, compatible version of the vi text editor (which we call vi in the rest of this chapter because it has a symbolic link named vi and a symbolically linked manual page).

Note that not all text editors are *screen oriented*, meaning designed for use from a terminal. Some of the text editors are designed to run from a graphical desktop and which provide a graphical interface with menu bars, buttons, scrollbars, and so on, are:

- *gedit*—A GUI text editor for GNOME
- *kate*—A simple KDE text editor
- *kedit*—Another simple KDE text editor

A good reason to learn how to use a text-based editor, such as `vi or nano`, is that system maintenance and recovery operations almost never take place during X Window sessions, negating the use of a GUI editor. Many larger, more complex and capable editors do not work when Linux is booted to its single-user or maintenance mode. If anything does go wrong with your system and you can't log into the X Window system and its graphical user interface, knowledge and experience of using both the command line and text editors will turn out to be very important. Make a point of opening some of the editors and playing around with them; you never know—you might just thank me someday!

Another reason to learn how to use a text-based editor under the Linux console mode is so that you can edit text files through remote shell sessions, because many servers will not host graphical desktops.

Working with `vi`

The one editor found on nearly every UNIX and Linux system is the `vi` editor, originally written by Bill Joy. This simple-to-use but incredibly capable editor features a somewhat cryptic command set, but you can put it to use with only a few commands. Although many experienced UNIX and Linux users use `vi` extensively during computing sessions, many users who do only quick and simple editing might not need all its power and may prefer an easier-to-use text editor such as `pico` or GNU `nano`. Diehard GNU fans and programmers definitely use `emacs`.

However, learning how to use `vi` is a good idea. You might need to edit files on a Linux system with a minimal install, or a remote server without a more extensive offering of installed text editors. Chances are nearly 100 percent that `vi` will be available.

You can start an editing session by using the `vi` command like this:

```
matthew@seymour:~$ vi file.txt
```

The vi command works by using an insert (or editing) mode, and a viewing (or command) mode.

When you first start editing, you are in the viewing mode. You can use your arrow or other navigation keys (as shown later) to scroll through the text. To start editing, press the `i` key to insert text or the `a` key to append text. When you're finished, use the Esc key to toggle out of the insert or append modes and into the viewing (or command) mode. To enter a command, type a colon (`:`), followed by the command, such as `w` to write the file, and press Enter.

Although `vi` supports many complex editing operations and numerous commands, you can accomplish work by using a few basic commands. These basic `vi` commands are

- **Cursor movement**—`h, j, k, l` (left, down, up, and right)
- **Delete character**—`x`
- **Delete line**—`dd`
- **Mode toggle**—`Esc`, `Insert` (or `i`)
- **Quit**—`:q`
- **Quit without saving**—`:q!`
- **Run a shell command**—`:sh` (use `'exit'` to return)
- **Save file**—`:w`

- **Text search**—/

NOTE

Use the `vimtutor` command to quickly learn how to use `vi`'s keyboard commands. The tutorial takes less than 30 minutes, and it teaches new users how to start or stop the editor, navigate files, insert and delete text, and perform search, replace, and insert operations.

Working with `emacs`

Richard M. Stallman's GNU `emacs` editor, like `vi`, is included with Ubuntu and nearly every other Linux distribution. Unlike other UNIX and Linux text editors, `emacs` is much more than a simple text editor—it is an editing environment and can be used to compile and build programs and act as an electronic diary, appointment book, and calendar; use it to compose and send electronic mail, read Usenet news, and even play games. The reason for this capability is that `emacs` contains a built-in language interpreter that uses the Elisp (`emacs` LISP) programming language. `emacs` is not installed in Ubuntu by default. To use emacs, the package you need to install is called `emacs`. See Chapter 32, "Managing Software."

You can start an `emacs` editing session like this:

```
matthew@seymour:~$ emacs file.txt
```

TIP

If you start `emacs` when using X11, the editor launches in its own floating window. To force `emacs` to display inside a terminal window instead of its own window (which can be useful if the window is a login at a remote computer), use the `-nw` command-line option like this: `emacs -nw file.txt`.

The `emacs` editor uses an extensive set of keystroke and named commands, but you can work with it by using a basic command subset. Many of these basic commands require you to hold down the Ctrl key, or to first press a *meta* key (generally mapped to the Alt key). The basic commands are listed in Table 4.2.

Table 4.2. Emacs Editing Commands

Action	Command
Abort	Ctrl+G
Cursor left	Ctrl+B
Cursor down	Ctrl+N
Cursor right	Ctrl+F
Cursor up	Ctrl+P
Delete character	Ctrl+D
Delete line	Ctrl+K
Go to start of line	Ctrl+A
Go to end of line	Ctrl+E
Help	Ctrl+H
Quit	Ctrl+X, Ctrl+C
Save As	Ctrl+X, Ctrl+W
Save file	Ctrl+X, Ctrl+S
Search backward	Ctrl+R

Action	Command
Search forward	Ctrl+S
Start tutorial	Ctrl+H, T
Undo	Ctrl+X, U

TIP

One of the best reasons to learn how to use `emacs` is that you can use nearly all the same keystrokes to edit commands on the `bash` shell command line. Another reason is that like `vi`, `emacs` is universally available for installation on nearly every UNIX and Linux system, including Apple's Mac OS X.

Working with Permissions

Under Linux (and UNIX), everything in the file system, including directories and devices, is a file. And every file on your system has an accompanying set of permissions based on ownership. These permissions provide data security by giving specific permission settings to every single item denoting who may read, write, and/or execute the file. These permissions are set individually for the file's owner, for members of the group the file belongs to, and for all others on the system.

You can examine the default permissions for a file you create by using the `umask` command, which lists default permissions using the number system explained next, or by using the `touch` command and then the `ls` command's long-format listing like this:

```
matthew@seymour:~$ touch file
matthew@seymour:~$ ls -l file
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:06 file
```

In this example, the touch command is used to quickly create a file. The ls command then reports on the file, displaying the following (from left to right):

- **The type of file created—**Common indicators of the type of file are in the leading letter in the output. A blank (which is represented by a dash, as in the preceding example) designates a plain file, `d` designates a directory, `c` designates a character device (such as `/dev/ttyS0`), and `b` is used for a block device (such as `/dev/sda`).
- **Permissions—**Read, write, and execute permissions for the owner, group, and all others on the system. (You learn more about these permissions later in this section.)
- **Number of links to the file—**The number one (`1`) designates that there is only one file, whereas any other number indicates that there might be one or more hard-linked files. Links are created with the `ln` command. A hard-linked file is an exact copy of the file, but it might be located elsewhere on the system. Symbolic links of directories can also be created, but only the root operator can create a hard link of a directory.
- **The owner—**The account that owns the file; this is originally the file creator, but you can change this designation using the `chown` command.
- **The group—**The group of users allowed to access the file; this is originally the file creator's main group, but you can change this designation using the `chgrp` command.
- **File size and creation/modification date—**The last two elements indicate the size of the file in bytes and the date the file was created or last modified.

Assigning Permissions

Under Linux, permissions are grouped by owner, group, and others, with read, write, and execute permission assigned to each, like so:

```
Owner    Group    Others
Rwx      rwx      rxw
```

Permissions can be indicated by mnemonic or octal characters. Mnemonic characters are

- `r` indicates permission for an owner, member of the owner's group, or others to open and read the file.
- `w` indicates permission for an owner, member of the owner's group, or others to open and write to the file.
- `x` indicates permission for an owner, member of the owner's group, or others to execute the file (or read a directory).

In the previous example for the file named `file`, the owner, `matthew`, has read and write permission. Any member of the group named `matthew` may only read the file. All other users may only read the file. Also note that default permissions for files created by the root operator (while using sudo or a root account) will be different because of `umask` settings assigned by the shell.

Many users prefer to use numeric codes, based on octal (base 8) values, to represent permissions. Here's what these values mean:

- `4` indicates read permission.
- `2` indicates write permission.
- `1` indicates execute permission.

In octal notation, the previous example file has a permission setting of `664` (read + write or `4` + `2`, read + write or `4` + `2`, read-only or `4`). Although you can use either form of permissions notation, octal is easy to use quickly after you visualize and understand how permissions are numbered.

NOTE

In Linux, you can create groups to assign a number of users access to common directories and files, based on permissions. You might assign everyone in accounting to a group named `accounting` and allow that group access to accounts payable files while disallowing access by other departments. Defined groups are maintained by the root operator, but you can use the `newgrp` command to temporarily join other groups to access files (as long as the root operator has added you to the other groups). You can also allow or deny other groups' access to your files by modifying the group permissions of your files.

Directory Permissions

Directories are also files under Linux. For example, again use the `ls` command to show permissions like this:

```
matthew@seymour:~$ mkdir directory
matthew@seymour:~$ ls -ld directory
drwxr-xr-x                    2 matthew  matthew  4096 2010-06-30 13:23 directory
```

In this example, the `mkdir` command is used to create a directory. The `ls` command and its `-ld` option is used to show the permissions and other information about the directory (not its contents). Here you can

see that the directory has permission values of `755` (read + write + execute or `4 + 2 + 1`, read + execute or `4 + 1`, and read + execute or `4 + 1`).

This shows that the owner can read and write to the directory and, because of execute permission, also list the directory's contents. Group members and all other users can list only the directory contents. Note that directories require execute permission for anyone to be able to view their contents.

You should also notice that the `ls` command's output shows a leading `d` in the permissions field. This letter specifies that this file is a directory; normal files have a blank field in its place. Other files, such as those specifying a block or character device, have a different letter.

For example, if you examine the device file for a Linux serial port, you will see:

```
matthew@seymour:~$ ls -l /dev/ttyS0
crw-rw---- 1 root dialout 4, 64 2010-06-30 08:13 /dev/ttyS0
```

Here, `/dev/ttyS0` is a character device (such as a serial communications port and designated by a c) owned by root and available to anyone in the `dialout` group. The device has permissions of `660` (read + write, read + write, no permission).

On the other hand, if you examine the device file for an IDE hard drive, you see:

```
matthew@seymour:~$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 2010-06-30 08:13 /dev/sda
```

In this example, `b` designates a block device (a device that transfers and caches data in blocks) with similar permissions. Other device entries you will run across on your Linux system include symbolic links, designated by `s`.

You can use the `chmod` command to alter a file's permissions. This command uses various forms of command syntax, including octal or a mnemonic form (such as `u`, `g`, `o`, or `a` and `rwx`, and so on) to specify a desired change. The `chmod` command can be used to add, remove, or modify file or directory permissions to protect, hide, or open up access to a file by other users (except for the root account or a user with super-user permission and using sudo, either of which can access any file or directory on a Linux system).

The mnemonic forms of `chmod`'s options are (when used with a plus character, `+`, to add, or a minus sign, `-`, to remove):

- *u*—Adds or removes user (owner) read, write, or execute permission
- *g*—Adds or removes group read, write, or execute permission
- *o*—Adds or removes read, write, or execute permission for others not in a file's group
- *a*—Adds or removes read, write, or execute permission for all users
- *r*—Adds or removes read permission
- *w*—Adds or removes write permission
- *x*—Adds or removes execution permission

For example, if you create a file, such as a `readme.txt`, the file will have default permissions (set by the `umask` setting in `/etc/bashrc`) of

```
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

As you can see, you can read and write the file. Anyone else can only read the file (and only if it is outside your home directory, which will have read, write, and execute permission set only for you, the owner). You can remove all write permission for anyone by using `chmod`, the minus sign, and `aw` like so:

```
matthew@seymour:~$ chmod a-w readme.txt
matthew@seymour:~$ ls -l readme.txt
-r--r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

Now, no one can write to the file (except you, if the file is in your home or `/tmp` directory because of directory permissions). To restore read and write permission for only you as the owner, use the plus sign and the `u` and `rw` options like so:

```
matthew@seymour:~$ chmod u+rw readme.txt
matthew@seymour:~$ ls -l readme.txt
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

You can also use the octal form of the `chmod` command, for example, to modify a file's permissions so that only you, the owner, can read and write a file. Use the `chmod` command and a file permission of `600`, like this:

```
matthew@seymour:~$ chmod 600 readme.txt
matthew@seymour:~$ ls -l readme.txt
-rw------- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

If you take away execution permission for a directory, files might be hidden inside and may not be listed or accessed by anyone else (except the root operator, of course, who has access to any file on your system). By using various combinations of permission settings, you can quickly and easily set up a more secure environment, even as a normal user in your home directory.

Understanding Set User ID and Set Group ID Permissions

Two more types of permission are "set user ID", known as *suid*, and "set group ID," or *sgid*, permissions. These settings, when used in a program, enable any user running that program to have program owner or group owner permissions for that program. These settings enable the program to be run effectively by anyone, without requiring that each user's permissions be altered to include specific permissions for that program.

One commonly used program with suid permissions is the `passwd` command:

```
matthew@seymour:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 42856 2010-01-26 10:09 /usr/bin/passwd
```

This setting allows normal users to execute the command (as root) to make changes to a root-only accessible file, `/etc/passwd`.

You also can assign similar permission with the `chfn` command. This command allows users to update or change `finger` information in `/etc/passwd`. You accomplish this permission modification by using a leading `4` (or the mnemonic `s`) in front of the three octal values.

NOTE

Other files that might have suid or guid permissions include `at`, `rcp`, `rlogin`, `rsh`, `chage`, `chsh`, `ssh`, `crontab`, `sudo`, `sendmail`, `ping`, `mount`, and several UNIX-to-UNIX Copy (UUCP) utilities. Many programs (such as games) might also have this type of permission to access a sound device.

Files or programs that have suid or guid permissions can sometimes present security holes because they bypass normal permissions. This problem is compounded if the permission extends to an executable binary (a command) with an inherent security flaw because it could lead to any system user or intruder gaining root access. In past exploits, this typically happened when a user fed a vulnerable command with unexpected input (such as a long pathname or option); the command would fail, and the user would be presented a root prompt. Although Linux developers are constantly on the lookout for poor programming practices, new exploits are found all the time, and can crop up unexpectedly, especially in newer software packages that haven't had the benefit of peer developer review.

Savvy Linux system administrators keep the number of suid or guid files present on a system to a minimum. The `find` command can be used to display all such files on your system:

```
matthew@seymour:~$ sudo find / -type f -perm /6000 -exec ls -l {} \;
```

NOTE

The `find` command is quite helpful and can be used for many purposes, such as before or during backup operations. See the section "Using Backup Software" in Chapter 13, "Backing Up."

Note that the programs do not necessarily have to be removed from your system. If your users really do not need to use the program, you can remove the programs execute permission for anyone. You have to decide, as the root operator, whether your users are allowed, for example, to mount and unmount CD-ROMs or other media on your system. Although Linux-based operating systems can be set up to accommodate ease of use and convenience, allowing programs such as `mount` to be suid might not be the best security policy. Other candidates for suid permission change could include the `chsh`, `at`, or chage commands.

Working as Root

The root, or super-user account, is a special account and user on UNIX and Linux systems. Super-user permissions are required in part because of the restrictive file permissions assigned to important system configuration files. You must have root permission to edit these files or to access or modify certain devices (such as hard drives). When logged in as root, you have total control over your system, which can be dangerous.

When you work in root, you can destroy a running system with a simple invocation of the `rm` command like this:

```
matthew@seymour:~$ sudo rm -rf /
```

This command line not only deletes files and directories, but also could wipe out file systems on other partitions and even remote computers. This alone is reason enough to take precautions when using root access.

The only time you should run Linux as the super-user is when you are configuring the file system, for example, or to repair or maintain the system. Logging in and using Linux as the root operator isn't a good idea because it defeats the entire concept of file permissions.

Knowing how to run commands as the super-user (root) without logging in as root can help avoid serious missteps when configuring your system. In Ubuntu, you can use `sudo` to allow you to execute single commands as root and then quickly return to normal user status. For example, if you would like to edit

your system's file system table (a simple text file that describes local or remote storage devices, their type, and location), you can use `sudo` like this:

```
matthew@seymour:~$ sudo nano -w /etc/fstab
Password:
```

After you press Enter, you are prompted for a password that gives you access to root. This extra step can also help you "think before you leap" into the command. Enter the root password, and you are then editing `/etc/fstab`, using the nano editor with line wrapping disabled (thanks to the -w).

CAUTION

Before editing any important system or software service configuration file, make a backup copy. Then make sure to launch your text editor with line wrapping disabled. If you edit a configuration file without disabling line wrapping, you could insert spurious carriage returns and line feeds into its contents, causing the configured service to fail when restarting. By convention, nearly all configuration files are formatted for 80-character text width, but this is not always the case. By default, the `vi` and `emacs` editors don't use line wrap.

Creating Users

When a Linux system administrator creates a user, an entry in `/etc/passwd` for the user is created. The system also creates a directory, labeled with the user's username, in the `/home` directory. For example, if you create a user named `heather`, the user's home directory is `/home/heather`.

NOTE

In this chapter, you learn how to manage users from the command line. See Chapter 10, "Managing Users," for more information on user administration with Ubuntu using graphical administration utilities, such as the graphical `users-admin` client found in the menu at System, Administration, Users and Groups.

Use the `useradd` command, along with a user's name, to quickly create a user:

```
matthew@seymour:~$ sudo useradd ryan
```

After creating the user, you must also create the user's initial password with the `passwd` command:

```
matthew@seymour:~$ sudo passwd ryan


Changing password for user ryan.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Enter the new password twice. If you do not create an initial password for a new user, the user cannot log in.

You can view `useradd`'s default new user settings by using the command and its `-D` option, like this:

```
matthew@seymour:~$ useradd -D
GROUP=100
HOME=/home
```

```
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

These options display the default group ID, home directory, account and password policy (active forever with no password expiration), the default shell, and the directory containing defaults for the shell.

The `useradd` command has many command-line options. The command can be used to set policies and dates for the new user's password, assign a login shell, assign group membership, and other aspects of a user's account. See `man useradd` for more info.

Deleting Users

Use the `userdel` command to delete users from your system. This command removes a user's entry in the system's `/etc/passwd` file. You should also use the command's `-r` option to remove all the user's files and directories (such as the user's mail spool file under `/var/spool/mail`):

```
matthew@seymour:~$ sudo userdel -r andrew
```

If you do not use the -r option, you have to manually delete the user's directory under `/home`, along with the user's `/var/spool/mail` queue.

Shutting Down the System

Use the `shutdown` command to shut down your system. The `shutdown` command has a number of different command-line options (such as shutting down at a predetermined time), but the fastest way to cleanly shut down Linux is to use the `-h` or halt option, followed by the word `now` or the numeral zero (`0`), like this:

```
matthew@seymour:~$ sudo shutdown -h now
```

or

```
matthew@seymour:~$ sudo shutdown -h 0
```

To incorporate a timed shutdown and a pertinent message to all active users, use shutdown's time and message options, like so:

```
matthew@seymour:~$ sudo shutdown -h 18:30 "System is going down for maintenance this
evening at 6:30 p.m. Please make sure you have saved your work and logged out by
then or you may lose data."
```

This example shuts down your system and provides a warning to all active users 15 minutes before the shutdown (or reboot). Shutting down a running server can be considered drastic, especially if there are active users or exchanges of important data occurring (such as a backup in progress). One good approach is to warn users ahead of time. This can be done by editing the system Message of the Day (MOTD) `motd` file, which displays a message to users when they login using the command line interface, as is common on multi-user systems.

It used to be that to create a custom MOTD you only had to use a text editor and change the contents of `/etc/motd`. However, this has changed in Ubuntu as the developers have added a way to automatically

and regularly update some useful information contained in MOTD using cron. To modify how the MOTD is updated, you should install update-motd and read the man page.

You can also make downtimes part of a regular schedule, perhaps to coincide with security audits, software updates, or hardware maintenance.

You should shut down Ubuntu for only a few very specific reasons:

- You are not using the computer, no other users are logged in or expected to need or use the system, such as your personal desktop or laptop computer, and you want to conserve electrical power.
- You need to perform system maintenance that requires any or all system services to be stopped.
- You want to replace integral hardware.

TIP

Do not shut down your computer if you suspect that one or more intruders has infiltrated your system; instead, disconnect the machine from any or all networks and make a backup copy of your hard drives. You might want to also keep the machine running to examine the contents of memory and to examine system logs. Exceptions to this are when the system contains only trivial data files and non-essential services, such as a personal computer that is only used to run a web browser, and when you have no intention of trying to track down what an intruder may have changed, either to repair the damage or to try to catch them using computer forensics, but rather plan to merely wipe everything clean and rebuild or reinstall the system from scratch.

Rebooting the System

You should also use the `shutdown` command to reboot your system. The fastest way to cleanly reboot Linux is to use the `-r` option, and the word `now` or the numeral zero (`0`):

```
matthew@seymour:~$ sudo shutdown -r now
```

or

```
matthew@seymour:~$ sudo shutdown -r 0
```

Both rebooting and shutting down can have dire consequences if performed at the wrong time (such as during backups or critical file transfers, which arouses the ire of your system's users). However, Linux-based operating systems are designed to properly stop active system services in an orderly fashion. Other commands you can use to shut down and reboot Linux are the halt and reboot commands, but the shutdown command is more flexible.

Reading Documentation

Although you learn the basics of using Ubuntu in this book, you need time and practice to master and troubleshoot more complex aspects of the Linux operating system and your distribution. As with any operating system, you can expect to encounter some problems or perplexing questions as you continue to work with Linux. The first place to turn for help with these issues is the documentation included with your system; if you cannot find the information you need there, check Ubuntu's website.

Using Apropros

Linux, like UNIX, is a self-documenting system, with man pages accessible through the `man` command. Linux offers many other helpful commands for accessing its documentation. You can use the `apropos` command—for example, with a keyword such as `partition`—to find commands related to partitioning, like this:

```
matthew@seymour:~$ apropos partition
addpart       (8)    - Simple wrapper around the "add partition" ioctl
all-swaps     (7)    - Event signaling that all swap partitions have been ac...
cfdisk        (8)    - Curses/slang based disk partition table manipulator fo...
delpart       (8)    - Simple wrapper around the "del partition" ioctl
fdisk         (8)    - Partition table manipulator for Linux
gparted       (8)    - Gnome partition editor for manipulating disk partitions.
Mpartition    (1)    - Partition an MSDOS hard disk
Partprobe     (8)    - Inform the OS of partition table changes
Partx         (8)    - Telling the kernel about presence and numbering of on-...
Pvcreate      (8)    - Initialize a disk or partition for use by LVM
Pvresize      (8)    - Resize a disk or partition in use by LVM2
Sfdisk        (8)    - Partition table manipulator for Linux
```

To find a command and its documentation, you can use the `whereis` command. For example, if you are looking for the `fdisk` command, you can do this:

```
$ whereis fdisk
fdisk: /sbin/fdisk /usr/share/man/man8/fdisk.8.gz
```

Using Man Pages

To learn more about a command or program, use the `man` command, followed by the name of the command. Man pages are stored in places like `/usr/share/man` and `/usr/local/share/man`, but you don't need to know that. To read a man page, such as the one for the `rm` command, use the `man` command like this:

```
matthew@seymour:~$ man rm
```

After you press Enter, the less command (a Linux command known as a *pager*) displays the man page. The less command is a text browser you can use to scroll forward and backward (even sideways) through the document to learn more about the command. Type the letter `h` to get help, use the forward slash to enter a search string, or press `q` to quit.

NOTE

Nearly all the hundreds of commands included with Linux each have a man page; however, some do not or may only have simple pages. You may also use the `info` command to read more detailed information about some commands or as a replacement for others. For example, to learn even more about `info` (which has a rather extensive manual page), use the `info` command like this:

```
$ info info
```

Use the arrow keys to navigate through the document and press `q` to quit reading.

The following programs and built-in shell commands are commonly used when working at the command line. These commands are organized by category to help you understand the command's purpose. If you need to find full information for using the command, you can find that information under the command's man page.

- **Managing users and groups**—chage, chfn, chsh, edquota, gpasswd, groupadd, groupdel, groupmod, groups, mkpasswd, newgrp, newusers, passwd, umask, useradd, userdel, usermod
- **Managing files and file systems**—cat, cd, chattr, chmod, chown, compress, cp, dd, fdisk, find, gzip, ln, mkdir, mksfs, mount, mv, rm, rmdir, rpm, sort, swapon, swapoff, tar, touch, umount, uncompress, uniq, unzip, zip
- **Managing running programs**—bg, fg, kill, killall, nice, ps, pstree, renice, top, watch
- **Getting information**—apropos, cal, cat, cmp, date, diff, df, dir, dmesg, du, env, file, free, grep, head, info, last, less, locate, ls, lsattr, man, more, pinfo, ps, pwd, stat, strings, tac, tail, top, uname, uptime, vdir, vmstat, w, wc, whatis, whereis, which, who, whoami
- **Console text editors**—ed, jed, joe, mcedit, nano, red, sed, vim
- **Console Internet and network commands**—bing, elm, ftp, host, hostname, ifconfig, links, lynx, mail, mutt, ncftp, netconfig, netstat, pine, ping, pump, rdate, route, scp, sftp, ssh, tcpdump, traceroute, whois, wire-test

Reference

- https://help.ubuntu.com/community/UsingTheTerminal—The Ubuntu community help page for using the terminal.
- https://help.ubuntu.com/community/LinuxFilesystemTreeOverview—The Ubuntu community help page for and overview of the Linux file system tree.
- https://help.ubuntu.com/community/RootSudo—An Ubuntu community page explaining sudo, the philosophy behind using it by default, and how to use it.
- www.vim.org/—Home page for the `vim` (`vi` clone) editor included with Linux distributions. Check here for updates, bug fixes, and news about this editor.
- www.gnu.org/software/emacs/emacs.html—Home page for the FSF's GNU `emacs` editing environment; you can find additional documentation and links to the source code for the latest version here.
- www.nano-editor.org/—Home page for the GNU `nano` editor environment.

---

800 East 96th Street Indianapolis, Indiana 46240

THEORY - Advanced Shell Features

OBJECTIVES:

This chapter will cover the following exam objectives:

Customize and use the shell environment v2

Students: should be able to customize shell environments to meet users' needs. Candidates should be able to modify global and user profiles.

Key Knowledge Areas:

- Set environment variables (e.g. PATH) at login or when spawning a new shell.
- Write Bash functions for frequently used sequences of commands.
- Maintain skeleton directories for new user accounts.
- Set command search path with the proper directory.

KEY TERMS:

&&
This is an AND list. If the command on the left side of && does execute successfully, then the command on the right side of && will execute.

.
Command used to re-execute a bash shell configuration file (.profile, .bashrc, etc.) after changes are made.

/etc/bash.bashrc
Systemwide configuration file read when a new shell starts. Contains variables, commands, and aliases common to all users.

/etc/profile
Systemwide bash configuration file read at login. Contains bash settings and environment variables common to all users.

alias
alias and unalias utilities create or remove a nickname (shorthand term) for a command or series of commands.

env
Displays all environmental variables available to all shells.

export
With no arguments export displays all environmental variables available to all shells. export VARIABLE is used to export a locl variable.

function
Similar to an alias but for executing multiple commands. Typically used in bash shell scripts.

set
With no arguments set displays the values of all variables (local and environment) set in the current shell

source
Command used to re-execute a bash shell configuration file (.profile, .bashrc, etc.) after changes are made.

unset
Removes the variable from the current shell and subshells.

||
This is an OR list. If the command on the left side of || does NOT execute successfully, then the command on the right side of || is executed.

~/.bash_login
A bash user configuration file used to customize the shell environment. Sourced if ~/.bash_profile does not exist. Includes global variables not defined in /etc/profile.

~/.bash_logout
A bash user configuration file read at logout. Typically contains "clean up" commands such as clear to remove any text on the screen.

~/.bash_profile
A bash user configuration file used to customize the shell environment. If exists, it is sourced after /etc/profile. Includes global variables not defined in /etc/profile.

~/.bashrc
A bash user configuration file read each time a new shell is started. Typically contains local variables, commands and aliases.

~/.profile
A bash user configuration file used to customize the shell environment. Sourced if neither ~/.bash_profile or ~/.bash_login does not exist. Includes global variables not defined in /etc/profile.

1.1 Introduction

Even though most Linux distributions use a Graphical User Interface (GUI), administrators are more likely to work in the Command Line Interface (CLI) to perform routine tasks. The command line typically has much more powerful features, which are not available in a GUI, such as scripts (i.e., Shell scripts) that administrators can use to perform remote and advanced tasks such as setting up system services.

In general, administrators work with many servers simultaneously, and the GUI interface does not lend itself to this type of work. GUIs also utilize valuable system resources (such as RAM and CPU), which slows down the performance of other server functions (such as FTP and HTTP).

As a result, becoming comfortable with the command line environment is crucial for anyone who works with Linux on a regular basis. The primary focus of this module is to increase your comfort level and productivity level while working with a shell environment.

This chapter will address shell features and shell programming. Understanding how to customize a shell environment is an essential skill for both end-users and administrators. End-users may want to modify their shell environment to suit their specific needs. For example, a software developer may want a different environment than someone who creates and maintains websites.

Administrators can make the shell environment easier to use for entire sets of users. Knowing how to customize a shell environment and apply these customizations to multiple user accounts will provide a more user-friendly, "best practices" environment. This should increase productivity and reduce support issues.

In this chapter, you will learn a variety of shell customization features, including variables, aliases, and customization files.

1.2 Shell Variables

A variable is a name or identifier that can be assigned a value. The shell (and other commands) reads the values of these variables, which can result in altered behavior depending on the contents (value) of the variable.

Variables typically hold a single value, like `0` or `bob`. Some may contain multiple values separated by spaces like `Joe Brown` or by other characters, such as colons: `/usr/bin:/usr/sbin:/bin:/usr/bin:/home/joe/bin`.

You assign a value to a variable by typing the name of the variable, immediately followed by an equal sign `=` and then the value. For example: `name="Bob Smith"`.

Variable names should start with a letter (alpha character) or underscore and contain only letters, numbers, and the underscore character. It is important to remember that variable names are case-sensitive; `a` and `A` are different variables. Just as with arguments to commands, single or double quotes should be used when special characters are included in the value assigned to the variable to prevent shell expansion.

Valid Variable Assignments	Invalid Variable Assignments
`a=1`	`1=a`

Valid Variable Assignments	Invalid Variable Assignments
`_1=a`	`a-1=3`
`LONG_VARIABLE='OK'`	`LONG-VARIABLE='WRONG'`
`Name='Jose Romero'`	`'user name'=anything`
`sshdirectory='/etc/ssh'`	`"sshdirectory"=etc/ssh`

The Bash shell and many commands make extensive use of variables. One of the main uses of variables is to provide a means to configure various preferences for each user. The Bash shell and commands can behave in different ways, based on the value of variables. For the shell, variables can affect what the prompt displays, the directories the shell will search for commands to execute, and much more.

1.3 Local and Environment Variables

A *local* variable is only available to the shell in which it was created. An *environment* variable is available to the shell in which it was created, and it is passed into all other commands/programs started by the shell.

By convention, lowercase characters are used to create local variable names, and uppercase characters are used when naming an environment variable. For example, a local variable might be called `test`, while an environment variable might be called `TEST`. While this is a convention that most people follow, it is not a rule.

There are several ways to display the values of variables. The `set` command by itself will display all variables (local and environment):

```
clima@centos:~$ set | head
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote
:force_fignore:histappend:interactive_comments:login_shell:progcomp:promptvars:s
ourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=([0]="2" [1]="8")
BASH_LINENO=()
BASH_SOURCE=()
```

```
BASH_VERSINFO=([0]="4" [1]="4" [2]="19" [3]="1" [4]="release" [5]="x86_64-pc-
lin

ux-gnu")
```

To display only environment variables, there are several commands that provide nearly the same output: `env`, `declare -x`, `typeset -x` or `export -p`:

```
clima@centos:~$ env | tail
PWD=/home/clima
HOME=/home/clima
MAIL=/var/mail/clima
SHELL=/bin/bash
TERM=xterm
SHLVL=1
LOGNAME=clima
PATH=/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games
LESSOPEN=| /usr/bin/lesspipe %s
_=/usr/bin/env
clima@centos:~$ declare -x | tail -5
declare -x PWD="/home/clima"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x TERM="xterm"
declare -x USER="clima"
clima@centos:~$ typeset -x | tail -5
declare -x PWD="/home/clima"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x TERM="xterm"
declare -x USER="clima"
clima@centos:~$ export -p | tail -5
declare -x PWD="/home/clima"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x TERM="xterm"
```

```
declare -x USER="clima"
```

To display the value of a specific variable, use the `echo` command with the name of the variable prefixed by the dollar `$` sign. For example, to display the value of the `PATH` variable, you would execute `echo $PATH`.

```
clima@centos:~$ echo $PATH
/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
```

Consider This

Variables can also be enclosed in curly `{}` braces in order to delimit them from surrounding text. While the `echo ${PATH}` command would produce the same result as the `echo $PATH` command, curly braces set the variable apart visually, making it easier to see in scripts in some contexts.

1.4 Creating Environment Variables

By default, when a variable is assigned in the Bash shell, it is initially set as a local variable. There are a few ways that a local variable can also be made to be an environment variable. First, an existing local variable can be *exported* with the `export` command:

```
clima@centos:~$ ENVIRONMENT_A=1
clima@centos:~$ export ENVIRONMENT_A
```

Second, a new variable can be *exported* and assigned a value with a single command:

```
clima@centos:~$ export ENVIRONMENT_B=2
```

Third, the `declare` or `typeset` command can be used, to *declare* a variable to be an environment variable. These commands are synonymous and work the same way:

```
clima@centos:~$ declare -x ENVIRONMENT_C=3
clima@centos:~$ typeset -x ENVIRONMENT_D=3
```

If the commands above are all executed, there will be four new environment variables:

```
clima@centos:~$ env | grep ENVIRONMENT
ENVIRONMENT_B=2
ENVIRONMENT_C=3
ENVIRONMENT_A=1
ENVIRONMENT_D=3
```

Using the following syntax, the `env` command can also be used to temporarily set a variable:

```
env VARIABLE_NAME=TEMP_VALUE command
```

Servers are often set to Coordinated Universal Time (UTC), which is good for maintaining consistent time on servers across the planet, but can be frustrating for practical use to simply tell the time:

```
clima@centos:~$ date
Fri Sep 18 21:27:25 UTC 2020
```

To temporarily set the time zone variable `TZ` to Eastern Standard Time (EST), use the `env` command:

```
clima@centos:~$ env TZ=EST date
Fri Sep 18 16:27:54 EST 2020
```

The `TZ` variable is set only in the environment of the current shell, and only for the duration of the `date` command. The rest of the system will not be affected by this variable. In fact, running the `date` command again will verify that the `TZ` variable has reverted to UTC.

```
clima@centos:~$ date
Fri Sep 18 21:29:43 UTC 2020
```

1.5 Unsetting Variables

If the shell option `nounset` is enabled with the `set -o nounset` command, then referring to an unset variable will result in an `unbound variable` error. When the command is used within a script, referencing an unset variable will cause the script to exit.

This option may be *turned off*, using the `set +o nounset` command, where any reference to an unset variable will return a null value.

If you create a variable and then no longer want that variable to be defined, use the `unset` command to delete it:

```
clima@centos:~$ variable_1="This is a variable."
clima@centos:~$ echo $variable_1
This is a variable.
clima@centos:~$ unset variable_1
clima@centos:~$ set -o nounset
clima@centos:~$ echo $variable_1
-bash: variable_1: unbound variable
```

```
clima@centos:~$ set +o nounset
clima@centos:~$ echo $variable_1


clima@centos:~$
```

Important

Do not `unset` critical system variables like the `PATH` variable, since this could lead to a malfunctioning environment.

1.6 Understanding the `PATH` Variable

The `PATH` variable is one of the most critical environment variables for the shell, so it is important to understand the effect it has on how commands will be executed.

The following graphic displays a typical `PATH` variable, with directory names separated from each other by the colon `:` character:

```
clima@centos:~$ echo $PATH
/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
```

The `PATH` variable contains a list of directories that are used to search for commands entered by the user. When the user types a command and then presses the **Enter** key, the `PATH` directories are searched for an executable file that matches the command name. Processing works through the list of directories from left to right; the first executable file that matches what is typed is the command the shell will try to execute.

Consider This

Before searching the `PATH` variable for the command, the shell will first determine if the command is an alias or function, which may result in the `PATH` variable not being utilized when that specific command is executed.

Additionally, if the command happens to be built-in to the shell, the `PATH` variable will not be utilized.

Lastly, if the user uses an absolute path such as `/bin/ls` or a relative path such as `./ls`, then the `PATH` variable is not utilized.

The following table describes the purpose of some of the directories displayed in the `PATH` variable above:

Directory	Contents
`/home/clima/bin`	A directory for the current user `clima` to place programs. Typically used by users who create their own scripts.
`/usr/local/sbin`	Normally empty, but may have administrative commands that have been compiled from local sources.

Directory	Contents
`/usr/local/bin`	Normally empty, but may have commands that have been compiled from local sources.
`/usr/sbin`	Contains the majority of the administrative command files.
`/usr/bin`	Contains the majority of the commands that are available for regular users to execute.
`/sbin`	Contains the essential administrative commands.
`/bin`	Contains the most fundamental commands that are essential for the operating system to function.

To execute commands that are not contained in the directories that are listed in the `PATH` variable, several options exist:

- The command may be executed by typing the absolute path to the command.
- The command may be executed with a relative path to the command.
- The `PATH` variable can be set to include the directory where the command is located.
- The command can be relocated or copied to a directory that is listed in the `PATH` variable.

Absolute Path

To demonstrate *absolute paths* vs. *relative paths*, an executable script named `my.sh` is created in the home directory:

```
clima@centos:~$ echo 'echo Hello World!' > my.sh
clima@centos:~$ chmod u+x my.sh
```

Note

The `chmod` command is used to modify user permissions using the *symbolic method*. In the command above, the `u` represents the *user owner*, the plus `+` sign represents *add the permission*, and the `x` represents the `execute` permission; in other words, add the execute permission to the user owner on the file.

An absolute path specifies the location of a file or directory from the top-level directory through all of the subdirectories to the file or directory. Absolute paths always start with the `/` character representing the root directory. For example, `/usr/bin/ls` is an absolute path. It means the *ls file, which is in the bin directory, which is in the usr directory, which is in the / (root) directory*. A file can be executed using an absolute path like so:

```
clima@centos:~$ /home/clima/my.sh
Hello World!
```

Relative Path

A relative path specifies the location of a file or directory relative to the current directory. For example, in the `/home/clima` directory, a relative path of `test/newfile` would actually refer to the `/home/clima/test/newfile` file. Relative paths never start with the `/` character.

Using a relative path to execute a file in the current directory requires the use of the period `.` character, which symbolizes the current directory:

```
clima@centos:~$ ./my.sh
Hello World!
```

Modify `PATH` Variable

When a user wants their home directory added to the `PATH` variable in order to run scripts and programs without using `./` in front of the filename, they might be tempted to modify the path variable like so:

```
clima@centos:~$ echo $PATH
/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
clima@centos:~$ pwd
/home/clima
clima@centos:~$ PATH=/home/clima
```

Unfortunately, everything that was previously contained in the `PATH` variable will be lost.

```
clima@centos:~$ echo $PATH
/home/clima
```

This is because `PATH=/home/clima` resigns the value of the `PATH` variable. A correct method is included later in this section.

Programs listed outside of the `/home/clima` directory will now only be accessible by using their full pathname. For example, assuming that the home directory has not yet been added to the `PATH` variable, the `uname -a` command would behave as expected:

```
clima@centos:~$ uname -a
Linux centos 4.4.0-72-generic #93~14.04.1-Ubuntu SMP Fri Mar 31 15:05:15 UTC
2017 x86_64 x86_64 x86_64 GNU/Linux
```

After assigning the home directory to the `PATH` variable, the `uname -a` command will need its full pathname to execute successfully:

```
clima@centos:~$ PATH=/home/clima
clima@centos:~$ uname -a
Command 'uname' is available in '/bin/uname'
The command could not be located because '/bin' is not included in the PATH e
nvi
```

```
ronment variable.

uname: command not found

clima@centos:~$ /bin/uname -a

Linux centos 4.4.0-72-generic #93~14.04.1-Ubuntu SMP Fri Mar 31 15:05:15 UTC
2017 x86_64 x86_64 x86_64 GNU/Linux
```

It is possible to add to the `PATH` variable without overwriting its previous contents. Import the current value of the `$PATH` variable into the newly-defined `PATH` variable by using it on both sides of the assignment statement.

Note

If you changed the `PATH` variable like we did in the previous example and want to reset it, simply use the `exit` command to logout:

```
clima@centos:~$ exit
```

Once logged back in, the PATH variable will be reset to its original value.

```
clima@centos:~$ PATH=$PATH
```

Finish it with the value of the additional home directory path.

```
clima@centos:~$ PATH=$PATH:/home/clima

clima@centos:~$ echo $PATH

/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games:/home/clima
```

Now, scripts located in the `/home/clima` directory can execute without using a path:

```
clima@centos:~$ my.sh

Hello World!
```

Note

In general, it is a bad idea to modify the `PATH` variable. If it were to change, administrators would view it as suspicious activity. Malicious forces want to gain elevated privileges and access to sensitive information residing on Linux servers. One way to do this is to write a script that shares the name of a system command, then change the `PATH` variable to include the administrator's home directory. When the administrator types in the command, it actually runs the malicious script!

Relocate the Command

It is often simpler to copy a script to a directory that is already included in the `PATH`. For example, the user could place the script in the `/home/clima/bin` directory, since that directory is already included in the `PATH` variable.

Remember that it is possible for different commands (located in different directories) to have the same name; the shell will always try to execute the first matching command it finds. For this reason, it is best to avoid using names for scripts that are already used by other commands.

The directories listed in the `PATH` will be searched in the order they are entered, from left to right. For example, based on the output below, if a script named `userupdate` is present in both the `usr/local/bin` and the `usr/local/games` directories, the version that would be found and executed would be the one in the `usr/local/bin` directory.

```
clima@centos:~$ echo $PATH
/home/clima/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
```

Consider This

How can you tell if a command that you are creating might have the same name as another command on the system? Given that the conflict could be with a regular command, another shell script, a built-in command, an alias, or a function, it could be a bit challenging. Fortunately, the `type` command will help you discover any potential conflicts.

As demonstrated in the output below, the `type` command can determine if the command specified already exists. The final command executed in the example below, using the `junk` command, illustrates what it looks like when there is no such command:

```
clima@centos:~$ type echo
echo is a shell builtin
clima@centos:~$ type ls
ls is aliased to `ls --color=auto'
clima@centos:~$ type cal
cal is /usr/bin/cal
clima@centos:~$ type junk
-bash: type: junk: not found
```

1.7 Additional Common Variables

It is important to be familiar with the most commonly-used shell variables. Some of these variables will affect specific programs, while others will display information or change the behavior of the shell. The following describes some of the most important variables.

1.7.1 Prompt Variables

Typically, the prompt includes the user name, hostname, and current directory:

```
clima@centos:~$
```

This prompt can be changed by modifying the `PS1` (Prompt Shell 1) variable. Before making any changes to the `PS1`, back it up using a local variable so it can be restored to its original state:

```
clima@centos:~$ echo $PS1

\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\
033

[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$

clima@centos:~$ ps1=$PS1
```

Once the `PS1` is modified, changes to the prompt will be effective immediately, as seen in the following example:

```
sysadmin@localhost:~$
```

There are several useful special character sequences available to customize this prompt. These special characters start with a backslash character. For example, to display the current path in the prompt, use either the `\w` or `\W` character sequence:

```
sysadmin@localhost:~$
```

Notice that `\w` provides a full path in the prompt `(/usr/share/doc` in the example above) while `\W` provides a relative path in the prompt. To see the possible prompt character sequences, view the man page for `bash` and search for `PROMPTING`:

```
clima@centos:~$ man bash

Output Omitted...

OPTIONS

       All of the single-character shell options documented in the descriptio
n

       of the set builtin command can be used as options  when  the  shell  i
s

/PROMPTING_
```

Press the **Esc+n** key combination a couple of times to view the exact result as shown:

```
PROMPTING
```

```
       When executing interactively, bash displays the primary prompt PS1 whe
n
       it is ready to read a command, and the secondary prompt  PS2  when  it
       needs  more  input  to  complete  a  command.  Bash allows these promp
t
       strings to be customized by inserting  a  number  of  backslash-escape
d
       special characters that are decoded as follows:
              \a     an ASCII bell character (07)
              \d     the  date  in "Weekday Month Date" format (e.g., "Tue Ma
y
                     26")

              \D{format}
                     the format is passed to strftime(3)  and  the  result  i
s
                     inserted  into the prompt string; an empty format result
s
                     in a locale-specific time representation.  The braces ar
e
                     required
              \e     an ASCII escape character (033)
              \h     the hostname up to the first `.'
              \H     the hostname
              \j     the number of jobs currently managed by the shell
              \l     the basename of the shell's terminal device name
              \n     newline
              \r     carriage return
              \s     the  name  of  the shell, the basename of $0 (the portio
n
                     following the final slash)
```

Note that changes made to the prompt only affect the current shell. Closing the shell and opening a new one will result in a return to the original prompt.

Note

To make changes to a variable permanent, define it in an initialization file.

1.7.2 History Variables

Several variables affect the use of the `history` command (and associated features). Recall that the `history` command can be used to *re-execute* previously executed commands.
The `HISTSIZE` variable will determine how many commands to keep in memory for each Bash shell.

When the shell program is closed, it takes commands stored in memory and saves them into the *history file*, which is `~/.bash_history` by default. If a user wants to store the history commands in a different file, then the user can specify an absolute path as the value for the `HISTFILE` local variable:

```
HISTFILE=/tmp/history
```

By default, five hundred commands will be stored in the history file. The `HISTFILESIZE` variable will determine how many commands to write to this file.

If the values of `HISTSIZE` and `HISTFILESIZE` are different, the lowest of the two numbers is the number of commands that will actually be written to the file.

Although it is not normally set to anything by default, it may be advantageous to set a value for the `HISTCONTROL` variable in an initialization file like the `~/.bash_profile` file.
The `HISTCONTROL` variable could be set to any of the following features:

- `ignoredups`: This will prevent duplicate commands that are executed consecutively.
- `ignorespace`: This will not store any command that begins with a space. This provides the user with an easy way to execute a command that won't go into the history list.
- `ignoreboth`: This will not store consecutive duplicates or any command that begins with a space.
- `erasedups`: This will not store a command that is identical to another command in your history (actually, the previous entry of the command will be deleted from the history list).
- `ignorespace:erasedups`: This will include the benefit of `erasedups` with the advantage of `ignorespace`.

Another variable that will affect what gets stored in the history of commands is the `HISTIGNORE` variable. Most users do not want the history list cluttered with basic commands like `ls`, `cd`, `exit`, and `history`. The `HISTIGNORE` variable can be used to tell Bash not to store certain commands in the history list.

To have commands not included in the history list, include a command, like the following, in the `~/.bash_profile` file:

```
HISTIGNORE='ls*:cd*:history*:exit'
```

Consider This

Recall that the asterisk `*` character is used to indicate *anything else after the command*; so, `ls*` would match any `ls` command, such as `ls -l` or `ls /etc`.

1.8 Aliases

An alias is essentially a nickname for a command or series of commands. Some aliases may be created automatically as the result of commands executed in initialization shells. To view the shell's current aliases, use the `alias` command with no arguments:

```
clima@centos:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || e
cho
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//
'\''
')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Each line describes an alias in the format that the `alias` command is executed when creating new aliases. For example, the alias that defines the *long list* `ll` command is:

```
alias ll='ls -alF'
```

Oddly enough, even the `ls` command is an alias to itself with color support:

```
alias ls='ls --color=auto'
```

The primary purposes of creating an alias include:

- **To include a command option by default:**
    - For example, if the `-i` option is always used when executing the `rm` command, an alias can be created for `rm`, so it always runs as `rm -i`.

      ```
      clima@centos:~$ alias rm='rm -i'
      ```

- **To create a short form of a long command:**
    - For example, if the command `ls -ld .*` is used often, an alias can be created for this, perhaps called `hidden`, which will display only the hidden files in a directory. When `hidden` is run as a command, it will execute `ls -ld .*`

      ```
      clima@centos:~$ alias hidden="ls -ld .*"
      ```

```
clima@centos:~$ hidden
drwxr-xr-x 1 clima clima 4096 Sep 20 22:49 .
drwxr-xr-x 1 root     root     4096 May  2 16:18 ..
-rw------- 1 clima clima  194 Sep 20 21:58 .bash_history
-rw-r--r-- 1 clima clima  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 clima clima 3768 Apr 24 16:24 .bashrc
drwx------ 2 clima clima 4096 Sep 20 21:54 .cache
-rw------- 1 clima clima   51 Sep 20 22:49 .lesshst
-rw-r--r-- 1 clima clima  807 Apr  4  2018 .profile
-rw-r--r-- 1 clima clima   74 Apr 24 16:24 .selected_editor
-rw-r--r-- 1 clima clima    0 Apr 24 16:24 .sudo_as_admin_success
ful
```

- **To create DOS equivalent commands:**
  - Many people need to work both on Microsoft Windows systems as well as Linux. Subtle differences, such as running `cls` (DOS) instead of `clear` (Linux) to clear the terminal display, can create headaches. On Linux, create an alias called `cls` that will run the `clear` command.

```
clima@centos:~$ cls
-bash: cls: command not found
clima@centos:~$ alias cls=clear
```

```
clima@centos:~$ cls
```

Aliases that are created on the command line are specific to the shell that they are created in. If a new shell is opened, the aliases that are created in this shell will not exist in the new shell.

To make aliases permanent, the `alias` command needs to be placed in an initialization shell. This technique will be discussed later in this chapter.

To remove an alias from the current shell, use the `unalias` command:

```
clima@centos:~$ alias | grep hidden
alias hidden='ls -ld .*'
clima@centos:~$ unalias hidden
clima@centos:~$ alias | grep hidden
clima@centos:~$ hidden
hidden: command not found
```

Note

If the alias value (like `ls -l .*`) has spaces in it, make sure to enclose it within quotes.

In some cases, an administrator may just want to avoid an alias temporarily. For example, consider a situation in which the `rm` command is aliased to the `rm -i` command:

```
clima@centos:~$ alias rm='rm -i'
clima@centos:~$ rm -r Documents
rm: descend into directory 'Documents'? y
rm: remove regular file 'Documents/animals.txt'? n
rm: remove regular file 'Documents/alpha-third.txt'? n
rm: remove regular file 'Documents/longfile.txt'? n
rm: remove regular file 'Documents/red.txt'? n
rm: remove regular file 'Documents/alpha.txt'? n
rm: remove regular file 'Documents/hidden.txt'? n
rm: remove regular file 'Documents/food.txt'? n
rm: remove regular file 'Documents/adjectives.txt'? n
rm: remove regular file 'Documents/spelling.txt'? n
rm: remove regular file 'Documents/os.csv'? n
rm: remove regular file 'Documents/letters.txt'? n
rm: remove regular file 'Documents/people.csv'? n
rm: remove regular file 'Documents/letters.txt'? n
rm: remove regular file 'Documents/people.csv'? n
rm: remove regular file 'Documents/alpha-first.txt'? n
rm: remove regular file 'Documents/newhome.txt'? n
rm: remove regular file 'Documents/numbers.txt'? n
rm: remove regular file 'Documents/alpha-second.txt'? n
rm: descend into directory 'Documents/School'? n
rm: remove regular file 'Documents/profile.txt'? n
rm: remove regular file 'Documents/linux.txt'? n
rm: descend into directory 'Documents/Work'? n
rm: remove write-protected regular file 'Documents/words'? n
rm: remove regular file 'Documents/hello.sh'? n
clima@centos:~$
```

The `-i` option tells the `rm` command to prompt for each file and directory before deleting. This can be very cumbersome in large directory structures. To avoid this alias for a single execution of the command, use the back slash `\` character before the command:

```
clima@centos:~$ mkdir test
clima@centos:~$ ls
Desktop    Downloads  Pictures  Templates  my.sh
Documents  Music      Public    Videos     test
clima@centos:~$ \rm -r test
clima@centos:~$ ls
Desktop    Downloads  Pictures  Templates  my.sh
Documents  Music      Public    Videos
```

Aliases can also be avoided by using either an absolute or relative path to the command.

1.9 Functions

Functions are a bit more advanced than aliases and typically are used in Bash shell scripts. As a result, they will be introduced in this unit and explored in greater detail in the next chapter on shell scripting.

A function is much like an alias in that it provides a *nickname* for something else that requires execution. However, functions are typically designed to execute multiple commands, not a single command like an alias (although it is possible to execute multiple commands with aliases).

For those familiar with Java, C++, or several other programming languages, this will look very familiar. To create a function, use the following syntax:

```
function_name ()
{
   commands_here
}
```

In the previous example, `function_name` can be anything that the administrator desires. Many commands can also be used to replace the `commands_here` placeholder. Note the formatting, in particular, the location of the parenthesis `()` and braces `{}`, as well as the convention of using tabs to make the function more easily readable.

Consider the scenario where an administrator is constantly executing the following commands:

```
cd /usr/share/doc
echo "Document directory usage report" > /tmp/report
```

```
date >> /tmp/report
pwd >> /tmp/report


du -sh . >> /tmp/report
cd ~
```

Instead of typing these commands repeatedly, the administrator can create a function that will execute each command, one at a time:

```
clima@centos:~$ report () {
> cd /usr/share/doc
> echo "Document directory usage report" > /tmp/report
> date >> /tmp/report
> pwd >> /tmp/report
> du -sh . >> /tmp/report
> cd ~
> }
clima@centos:~$ report
clima@centos:~$ more /tmp/report
Document directory usage report
Fri Sep 20 23:01:03 UTC 2019
/usr/share/doc
15M     .
```

The `>` characters that you see in the previous output are called *secondary prompts*. The Bash shell knows that to end the function, a closing curly `}` brace is needed, so it prompts for more information each time the **Enter** key is pressed, until a closing curly brace is found.

A nice improvement that can be made to this function is to make it more flexible and allow the user to pass an argument to the function; this argument would represent the directory that the user wants to execute the `du` command on. When arguments are passed to a function, they are stored in special variables: `$1` for the first argument, `$2` for the second argument, etc.

For example, use the arrows to modify the first command of the report function to allow the user to pass in an argument of which directory to perform the operations on:

```
clima@centos:~$ report () { cd $1; echo "Document directory usage report"
> /tmp/report; date >> /tmp/report; pwd >> /tmp/report; du -sh . >> /tmp/repo
rt;cd ~; }
clima@centos:~$ report /usr/share
clima@centos:~$ cd /usr/share
```

```
clima@centos:/usr/share$ more /tmp/report
Document directory usage report
Fri Sep 20 23:01:03 UTC 2019
/usr/share/doc
15M     .
clima@centos:/usr/share$ cd
clima@centos:~$
```

Note

Note Like aliases, functions only exist in the shell that they are created in. To make them *permanent*, place them in an initialization file.

1.10 Lists

In the context of the Bash shell, a list is a sequence of commands which are separated by one of the following operators:

Operator	Meaning
;	The commands within the list are executed sequentially, where the shell will execute the first command and wait for it to terminate before executing the next command. The exit status of the list is based upon the exit status of the last command that is executed.
&	Each command within the list is executed asynchronously within a subshell, or in the background. The shell does not wait for the commands to terminate and returns an exit status of zero.
&&	This is an AND list, so if the command on the left side of `&&` does execute successfully, then the command on the right side of `&&` will execute. The exit status of the list is based upon the exit status of the last command that is executed.
||	This is an OR list, so if the command on the left side of `||` does NOT execute successfully, then the command on the right side of `||` is executed. The exit status of the list is based upon the exit status of the last command that is executed.

To make sure every command executes in order, use a simple sequential list with the commands separated by the semicolon `;` character. For example, to execute the `date` command, followed by the `who` command, and then the `uptime` command, execute something like:

```
clima@centos:~$ date;who;uptime
Fri Sep 18 23:09:31 UTC 2020
clima console      Sep 18 21:58
 23:09:31 up 9 days, 21:50,  1 user,  load average: 0.30, 0.48, 0.45
```

If the list of the commands that an administrator wants to execute will take time to complete and the administrator wants to regain access to the shell immediately, the commands may be executed asynchronously in the background as a list like this:

```
clima@centos:~$ updatedb& makewhatis&
[1] 3259
[2] 3260
```

The `updatedb` command updates the database used by the `locate` command, while the `makewhatis` command collects information from the man pages and indexes it in a database. Note that the `makewhatis` command does not exist in our VM; therefore, the output in the example above may not match the output in our virtual environment.

Notice that the shell will output a job number in square brackets, along with a process identification (PID) number for each background process. These numbers may be used by commands such as `kill` and others to control these processes or jobs.

The AND list forms a conditional chain of commands separated by the double ampersand `&&` characters. Only when the command on the left side of the `&&` succeeds will the command on the right side of the `&&` execute. If a user wanted to copy a file after creating a directory, as long as the directory did not previously exist, they could execute the following chain of commands:

```
clima@centos:~$ mkdir ./Backup && cp /etc/hosts ./Backup
clima@centos:~$ ls
Backup   Documents  Music     Public     Videos
Desktop  Downloads  Pictures  Templates  my.sh

clima@centos:~$ ls ./Backup/
hosts
```

If the first command succeeds, the second command will execute. The result will be a new folder called `Backup` with a copy of the `/etc/hosts` file. If the same command was run in a directory without appropriate permissions, the failure of the `mkdir` command would cause the `cp` command not to run.

```
clima@centos:~$ cd /usr
clima@centos:/usr$ mkdir ./Backup && cp /etc/hosts ./Backup
mkdir: cannot create directory './Backup': Permission denied
```

Using an OR list, when one command is executed, another command will be executed only if it fails. In this example, the `test` command is used to see if a directory `$HOME/bin` exists. If this test fails, then the directory is created:

```
clima@centos:~$ test -e $HOME/bin || mkdir $HOME/bin
clima@centos:~$ ls $HOME
Backup   Documents  Music     Public     Videos  my.sh
```

```
Desktop  Downloads  Pictures  Templates  bin
```

Although the examples provided of command lists, so far, have all been where all the items in the lists were separated by the same operator, this has been done to keep the examples simple. Lists can be constructed that can combine the various operators in interesting and powerful ways, especially by combining the AND and OR list types. For example, the following hybrid AND/OR list could be used within a script to make sure that the variable named `search` is either set to the first argument passed to the script, or else it is set to the current working directory:

```
test $# -eq 0 && search=`pwd` || search=$1
```

1.11 Initialization Files

When a user opens a new shell, either during login or when they run a terminal that starts a shell, the shell is customized by files called initialization (or configuration) files. These initialization files set the value of variables, create aliases and functions, and execute other commands that are useful in starting the shell.

There are two types of initialization files: *global* initialization files that affect all users on the system and *local* initialization files that are specific to an individual user.

The global configuration files are located in the `/etc` directory. Local configuration files are stored in the user's home directory.

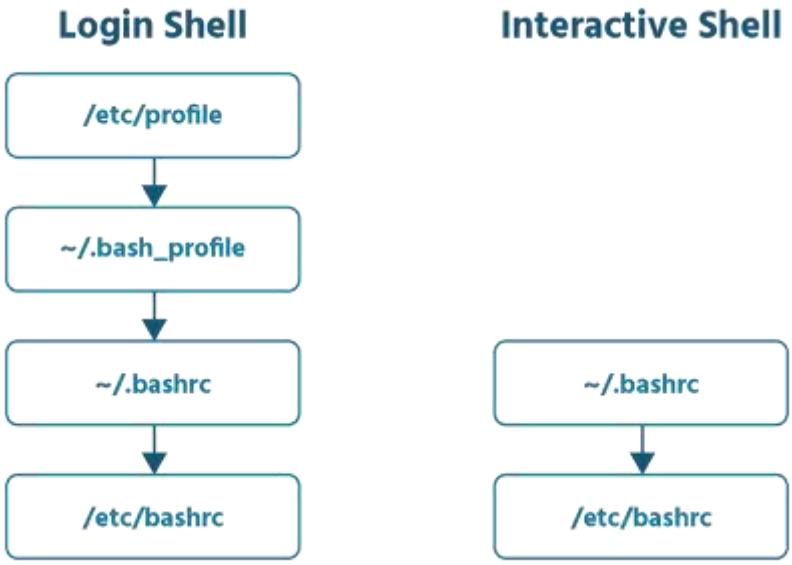
BASH Initialization Files

Each shell uses different initialization files. Additionally, most shells execute different initialization files when the shell is started via the login process (called a login shell) versus when a shell is started by a terminal (called a non-login shell or an interactive shell).

The following diagram illustrates the different files that are started with a typical login shell versus an interactive shell:

Note

The tilde `~` character represents the user's home directory and file names preceded by a period `.` character indicate hidden files.



When Bash is started as an interactive shell, it executes the `~/.bashrc` file, which may also execute the `/etc/bashrc` file, if it exists. Again, since the `~/.bashrc` file is owned by the user who is logging in, the user can prevent execution of the `/etc/bashrc` file.

With so many initialization files, a common question at this point is, "what file am I supposed to use?" The following chart illustrates the purpose of each of these files, providing examples of what commands you might place in each file:

File	Purpose
`/etc/profile`	This file can only be modified by the administrator and will be executed by every user who logs in. Administrators use this file to create key environment variables, display messages to users as they log in, and set key system values.
`~/.bash_profile`	Each user has their own `.bash_profile` file in their home directory. The purpose of this file is the same as the `/etc/profile` file, but having this file allows a user to customize the shell to their own tastes. This file is typically used to create customized environment variables.
`~/.bashrc`	Each user has their own `.bashrc` file in their home directory. The purpose of this file is to generate items that need to be created for each shell, such as local variables and aliases.
`/etc/bashrc`	This file may affect every user on the system. Only the administrator can modify this file. Like the `.bashrc` file, the purpose of this file is to generate items that need to be created for each shell, such as local variables and aliases.

1.12 Modifying Initialization Files

The way a user's shell operates can be changed by modifying that user's initialization files. Modifying global configuration requires administrative access to the system as the files under the `/etc` directory can only be modified by an administrator. A user can only modify the initialization files in their home directory.

The best practice is to create a backup before modifying configuration files as insurance against errors; a backup can always be restored if something should go wrong.

In some Linux distributions, the default `~/.bash_profile` contains the following two lines which customize the `PATH` environment variable:

```
PATH=$PATH:$HOME/bin

export PATH
```

The first line sets the `PATH` variable to the existing value of the `PATH` variable with the addition of the bin subdirectory of the user's home directory. The `$HOME` variable refers to the user's home directory. For example, if the user logging in is *joe*, then `$HOME/bin` is `/home/joe/bin`.

The second line converts the local `PATH` variable into an environment variable.

The default `~/.bashrc` file executes /etc/bashrc using a statement like:

```
. /etc/bashrc
```

The period `.` character is used to *source* a file in order to execute it. The period `.` character also does have a synonym command, the `source` command, but the use of the period `.` character is more common than using the `source` command.

Sourcing can be an effective way to test changes made to initialization files, enabling you to correct any errors without having to log out and log in again. If you update the `~/.bash_profile` to alter the `PATH` variable, you could verify that your changes are correct by executing the following:

```
. ~/.bash_profile

echo $PATH
```

1.13 BASH Exit Scripts

Just as Bash executes one or more files upon starting up, it also may execute one or more files upon exiting. As Bash exits, it will execute the `~/.bash_logout` and `/etc/bash_logout` files, if they exist. Typically, these files are used for *cleaning up* tactics as the user exits the shell. For example, the default `~/.bash_logout` executes the `clear` command to remove any text present in the terminal screen.

Consider This

When a new user is created, the administrator can automatically copy files from `/etc/skel` into the user's home directory. These files help setup a useful environment through startup files that are read when the user logs into his or her account.

Along with the files that are provided for when the user logs into his or her account, files that control tasks that happen when the user logs out are provided to the accounts from the `/etc/skel` directory when the account is created. Commonly these tasks clean up the environment that was used by the user's account during the login session.

When a new user is created, the files from the `/etc/skel` directory are automatically copied into the new user's home directory. As an administrator, you can modify the files in the `/etc/skel` directory to provide custom features to new users.

LAB - Advanced Shell Features

Introduction

In this lab, students will gain practice in customizing the shell and simple shell programming by performing the following tasks with CentOS (preferably but also works fine for Ubuntu with minor differences):

1. View and modify local and environment variables to customize the shell.
2. Create, use, and avoid aliases for executing commands within the shell.
3. Define functions that execute multiple commands.
4. Use lists of commands to logically execute commands.
5. Customize the bash initialization files.
6. Create several simple bash scripts.

Step 1

```
[clima@centos ~]$
```

Recall that shell variables contain useful information for the shell and other executable programs. Display all variables (both local and environment) by executing the `set` command:

```
set
```

```
[clima@centos ~]$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:histappend:hostcomplete:interactive_comments:login_shell:progcomp:promptvars:sourcepath
```

```
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="46" [3]="2" [4]="release" [5]="x86_64-redhat
-linux-gnu")
BASH_VERSION='4.2.46(2)-release'
COLUMNS=80
DIRSTACK=()
EUID=1000
GROUPS=()
HISTCONTROL=ignoredups
HISTFILE=/home/clima/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/clima
HOSTNAME=centos
HOSTTYPE=x86_64
```

Some output has been omitted from the terminal above for brevity.

Step 2

In some cases, you will want to distinguish between environment and local variables. To display only the environment variables, execute the `env` command:

```
env
```

```
[clima@centos ~]$ env
HOSTNAME=centos
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
USER=clima
```

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01
:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=30;42:o
w=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.t
az=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=0
1;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=
01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.
tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=0
1;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:
*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif
=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;3
5:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*
.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv
=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01
;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.
flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;
35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.
ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.flac=01;36:*.mid=01;36:*.midi=
01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:
*.axa=01;36:*.oga=01;36:*.spx=01;36:*.xspf=01;36:

MAIL=/var/spool/mail/clima

PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.loca

l/bin:/home/clima/bin

PWD=/home/clima

HISTCONTROL=ignoredups

SHLVL=1

HOME=/home/clima

LOGNAME=clima

LESSOPEN=||/usr/bin/lesspipe.sh %s

_=/bin/env
```

Step 3

Execute the following command to see an alternative method to display only the environment variables:

```
export -p
```

```
[clima@centos ~]$ export -p

declare -x HISTCONTROL="ignoredups"

declare -x HISTSIZE="1000"

declare -x HOME="/home/clima"

declare -x HOSTNAME="centos"
```

```
declare -x LESSOPEN="||/usr/bin/lesspipe.sh %s"

declare -x LOGNAME="clima"

declare -x LS_COLORS="rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35
:bd

=40;33;01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:t
w=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=
01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;3
1:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01
;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz
=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;3
1:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.z
oo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01
;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*
.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svg
z=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01
;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35
:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb
=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:
*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx
=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.flac=01;36:*.mid=01;3
6:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;36:*.w
av=01;36:*.axa=01;36:*.oga=01;36:*.spx=01;36:*.xspf=01;36:"

declare -x MAIL="/var/spool/mail/clima"

declare -x OLDPWD

declare -x PATH="/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home
/sy

sadmin/.local/bin:/home/clima/bin"

declare -x PWD="/home/clima"

declare -x SHELL="/bin/bash"

declare -x SHLVL="1"

declare -x TERM="xterm"

declare -x USER="clima"
```

Step 4

To set the `PS1` local variable, execute the following command. This local variable will change the prompt to display the history number of the current command and will display the prompt character as `$`:

```
PS1='\# \$ '
```

```
[clima@centos ~]$ PS1='\# \$ '
```

```
5 $
```

The number `5` shown reflects a running count of the commands in the command history. This number will increase with each command that is executed. The number shown in your own window may vary, depending on how many commands you have executed since starting the shell.

```
ls /
```

```
5 $ ls /

 bin  etc   lib    media  opt   root  sbin  sys  usr
dev  home  lib64  mnt    proc  run   srv   tmp  var
6 $ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
7 $
```

Step 5

Set the `PS1` local variable so that it will show the current user name, hostname, the working directory, and the `$` prompt character by executing the following command:

```
PS1='\u@\h \w \$ '
```

```
7 $ PS1='\u@\h \w \$ '
clima@centos ~ $
```

Step 6

Before you change the `COLUMNS` environment variable, notice how this variable affects the number of columns output by commands like `ls /bin`:

```
ls /bin
```

```
clima@centos ~ $ ls /bin
[                               objcopy
a2p                             objdump
addr2line                       od
alias                           oldfind
applydeltarpm                   p11-kit
```

```
apropos                 package-cleanup
ar                      passwd
arch                    paste
as                      pathchk
awk                     pcre-config
base64                  perl
basename                perl5.16.3
bash                    perlbug
bashbug                 perldoc
bashbug-64              perlthanks
bg                      pgawk
bootctl                 pgrep
busctl                  pic
c++filt                 piconv
c2ph                    pinentry
c89                     pinentry-curses
c99                     ping
ca-legacy               ping6
cal                     pinky
captoinfo               pk12util
cat                     pkg-config
catchsegv               pkill
catman                  pl2pm
cc                      pldd
cd                      pmap
certutil                pod2html
chacl                   pod2man
chage                   pod2text
chardetect              pod2usage
chcon                   post-grohtml
chfn                    pr
chgrp                   pre-grohtml
chmem                   preconv
chmod                   printenv
```

```
chown                                   printf
chrt                                    prlimit
chsh                                    ps
cksum                                   psed
clear                                   pstruct
...
```

Some output has been omitted from the terminal above for brevity.

The partial output shown nearly fills the terminal width because the default value of the `COLUMNS` variable is `80`.

Step 7

Change the `COLUMNS` variable to a value of `60`, and use the `export` command to make the variable an environment variable:

```
export COLUMNS=60
```

```
clima@centos ~ $ export COLUMNS=60
```

Step 8

Now that you have changed the `COLUMNS` environment variable, you can see a difference in the number of columns output by executing the following command:

```
ls /bin
```

```
clima@centos ~ $ ls /bin


[
a2p
addr2line
alias
applydeltarpm
apropos
ar
arch
```

```
as
awk
base64
basename
bash
bashbug
bashbug-64
bg
bootctl
busctl
c++filt
c2ph
c89
c99
ca-legacy
cal
captoinfo
cat
...
```

Some output has been omitted from the terminal above for brevity.

The output displayed above shows that fewer columns are output because the `COLUMNS` environment variable limits the number of characters that can be displayed on a single line.

Step 9

You can also display the value of an individual variable by using the `echo` command along with the name of the variable prefixed with a `$` (dollar sign). This works for both local and environment variables. Execute the following command to display the value of the `COLUMNS` variable:

```
echo $COLUMNS
```

```
clima@centos ~ $ echo $COLUMNS
60
```

When displaying the value of a variable, preface the variable with the `$` character. When setting a variable, do not use the `$` character.

Step 10

Display the `PATH` environment variable, which determines which directories will be searched for commands executed from the command line, by executing the following command:

```
echo $PATH
```

```
clima@centos ~ $ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.local/bin
:/home/clima/bin
```

Step 11

To test the `PATH` variable, first create a new directory in your home directory named `scripts` by executing the following command:

```
mkdir scripts
```

```
clima@centos ~ $ mkdir scripts
clima@centos ~ $ ls
Desktop    Downloads  Pictures  Templates  scripts
Documents  Music      Public    Videos
```

Step 12

Create a script file named `today` by executing the following commands:

```
echo '#!/bin/bash' > scripts/today
echo 'echo Today is $(date +%D)' >> scripts/today
cat scripts/today
```

```
clima@centos ~ $ echo '#!/bin/bash' > scripts/today
clima@centos ~ $ echo 'echo Today is $(date +%D)' >> scripts/today
clima@centos ~ $ cat scripts/today
#!/bin/bash
echo Today is $(date +%D)
```

Each `echo` command is taking the single-quoted text that immediately follows it and redirects that output to the `scripts/today` file. When a single `>` (greater than sign) follows the text, then that

text overwrites any existing text in the file. When a double > (greater than sign) follows the text, then that text is added to the end of the existing text in the file.

Step 13

The `bash` command can be used to execute the `today` script by providing it with the path to the script file. Execute the following command to execute this script:

```
bash scripts/today
```

```
clima@centos ~ $ bash scripts/today
Today is 10/08/25
```

Step 14

If you want to be able to execute the script directly (without the `bash` command), the permissions for the user must include the read and execute permission. Otherwise, a message about denied permission will be shown. By default, files that are created are never automatically given execute permission. Attempt to execute the script, realizing that it lacks execute permission, by typing the path to the file, `scripts/today`:

```
scripts/today
```

```
clima@centos ~ $ scripts/today
-bash: scripts/today: Permission denied
```

Step 15

Viewing the detailed listing of the file with the `ls` command will allow you to see the permissions of the file. Use `ls -l` to view the details of the script file:

```
ls -l scripts/today
```

```
clima@centos ~ $ ls -l scripts/today
-rw-rw-r-- 1 clima clima 38 Oct  8 02:20 scripts/today
```

In the example output above, the user owner has *rw (read and write)* permissions, the group owner has the same, and the others have only *read* permission. Everyone lacks the necessary *execute* permission to run the file directly.

Note

Note the second through tenth characters of the output. The second, third, and fourth characters represent the permissions belonging to the user owner of the file (you, in this case).

```
-rw-rw-r-- 1 clima clima 39 Oct 13 18:43 scripts/today
```

The fifth, sixth, and seventh characters represent the permissions belonging to the members of the group that owns the file (if they are not the user owner of it).

```
-rw-rw-r-- 1 clima clima 39 Oct 13 18:43 scripts/today
```

The eighth, ninth, and tenth characters show the permissions for any user who is not the owner of the file, nor a member of the group that owns the file.

```
-rw-rw-r-- 1 clima clima 39 Oct 13 18:43 scripts/today
```

For the three characters that represent the permissions, if they show `rwx`, then that file has *read, write,* and *execute* permission. If a dash `-` character is shown instead of the `r`, then the entity lacks the *read* permission. Likewise, if a `-` is shown instead of the `w`, then the entity lacks the *write* permission. Of special interest in this case, if a `-` is shown instead of the `x`, then the entity has no *execute* permission.

Step 16

Use the `chmod` command to change the permissions on the script so anyone with access to the scripts directory would be able to execute the script directly. Execute the following commands:

```
chmod a+x scripts/today
ls -l scripts/today
```

```
clima@centos ~ $ chmod a+x scripts/today
clima@centos ~ $ ls -l scripts/today
-rwxrwxr-x 1 clima clima 38 Oct  8 02:20 scripts/today
```

Now that the fourth, seventh, and tenth characters of the output are an `x`, this indicates that the file is executable for the user, group, and others. The green color that is shown for the path to the file is another clue that indicates the file is executable.

Step 17

With the script file executable, it can now be run with any valid relative path or absolute path. Execute the following commands to verify that this script is now executable:

```
scripts/today
cd scripts
./today
/home/clima/scripts/today
~/scripts/today
```

```
clima@centos ~ $ scripts/today
Today is 10/08/25
clima@centos ~ $ cd scripts
clima@centos ~/scripts $ ./today
Today is 10/08/25
clima@centos ~/scripts $ /home/clima/scripts/today
Today is 10/08/25
clima@centos ~/scripts $ ~/scripts/today
Today is 10/08/25
```

The path that starts with `./` represents a relative path to the file in the current directory. The path that starts with `~/` represents a path to the user's home directory.

Step 18

Although relative and absolute paths work to execute the script, using the script name only will fail because the `scripts` directory is not listed as a value in the `PATH` variable. Try to execute the script by only using the script name; it will fail because the current directory is not searched for commands:

```
today
clima@centos ~/scripts $ today
-bash: today: command not found
```

Step 19

Display the `PATH` variable again to confirm that the `scripts` directory is not contained within it. Modify the `PATH` variable so that it contains the `scripts` directory and display the `PATH` variable to verify the change:

```
echo $PATH
PATH=$PATH:$HOME/scripts
echo $PATH
clima@centos ~/scripts $ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.local/bin
:/home/clima/bin
clima@centos ~/scripts $ PATH=$PATH:$HOME/scripts
```

```
clima@centos ~/scripts $ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.local/bin
:/home/clima/bin:/home/clima/scripts
```

Step 20

Once the `PATH` variable contains the path to the directory where your script is located, you can have it execute by typing only the name of the script file, regardless of your location within the filesystem. Execute the script from your current directory. Then change to your home directory by executing the `cd` command and execute the today script again to see this work:

```
today
cd
today
```

```
clima@centos ~/scripts $ today
Today is 04/16/25
clima@centos ~/scripts $ cd
clima@centos ~ $ today
Today is 10/08/25
```

Step 21

By design, changes to variables made in the shell do not persist after the user logs out of the system. Type `exit` to log out of the system:

```
exit
```

```
clima@centos ~ $ exit
logout
This lab has two user accounts (username :: password )

   root     :: netlab123
   clima :: netlab123

Press the [Enter] key to begin...
```

```
Last login: Wed Oct  9 16:50:10 UTC 2019 on console
[clima@centos ~]$
```

Important

The virtual machine will need to be reset to continue. It will automatically log the user back in as the `clima` user after the reset is complete.

Step 22

Display the `PATH` variable to confirm that its value has reverted back to the original list of directories:

```
echo $PATH
```

```
[clima@centos ~]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.local/bin
:/home/clima/bin
```

Step 23

For a non-privileged user, the `.bash_profile` file can be used to customize the variables such as `PATH`. Execute the following command to see how the `PATH` variable is customized on line 10:

```
cat -n .bash_profile
```

```
[clima@centos ~]$ cat -n .bash_profile
     1  # .bash_profile
     2
     3  # Get the aliases and functions
     4  if [ -f ~/.bashrc ]; then
     5          . ~/.bashrc
     6  fi
     7
     8  # User specific environment and startup programs
     9
    10  PATH=$PATH:$HOME/.local/bin:$HOME/bin
```

```
    11
    12  export PATH
```

On line 10, the `PATH` variable is set to the value of the existing `PATH` variable and the `$HOME/bin` directory. Notice the colon `:` character that separates the paths.

Step 24

Use the `nano` editor to modify the `.bash_profile` file:

```
nano .bash_profile
```

```
[clima@centos ~]$ nano .bash_profile
  GNU nano 2.3.1                          File: .bash_profile


# .bash_profile


# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi


# User specific environment and startup programs


PATH=$PATH:$HOME/.local/bin:$HOME/bin


export PATH









                          [ Read 12 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Step 25

Use the **Arrow Keys** to move to the 10th line and add `$HOME/scripts` to it:

```
Down Arrow (x9)
Right Arrow (or Ctrl+E)
:$HOME/scripts
```

```
  GNU nano 2.3.1                          File: .bash_profile

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin:$HOME/scripts

export PATH




                          [ Read 12 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Step 26

Exit the `nano` editor by pressing **Ctrl+X**. Type y to save the changes, and press the **Enter** key:

```
Ctrl+X
```

```
y

Enter
```

Step 27

Instead of logging out and logging back in to test the changes to the `.bash_profile`, which is processed when the Bash shell starts during the login process, the file can be processed by using the `source` command. In some cases, you might see the use of the `.` command. The `.` command is the same as the `source` command. Execute the following command:

```
. .bash_profile
```

```
[clima@centos ~]$ . .bash_profile

[clima@centos ~]$
```

Important

It is much safer to test changes to Bash initialization files by sourcing those files instead of logging out and back in again. Potentially, an error in changing one of the initialization files might prevent a user from being able to log in again. If there is an error in sourcing the file while you are logged in, then the problem can be easily corrected.

Step 28

Display the `PATH` to verify that it includes the `scripts` directory now that you have sourced the `~/.bash_profile` file:

```
echo $PATH
```

```
[clima@centos ~]$ echo $PATH

/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/clima/.local/bin
:/home/clima/bin:/home/clima/.local/bin:/home/clima/bin:/home/clima/scripts
```

The user's `$HOME/bin` (`/home/clima/bin`) is included twice because it was added when the user first logged in and the `~/.bash_profile` file was processed, and again after sourcing the `~/.bash_profile` file. Having this path twice will not cause any problems.

Step 29

The files that are in the `/etc/skel` directory are automatically copied to a new user's home directory when their account is created. Execute the following command to view these files:

```
ls -a /etc/skel
```

```
[clima@centos ~]$ ls -a /etc/skel
.  ..  .bash_logout  .bash_profile  .bashrc
```

Step 30

To add user accounts or copy files into the `/etc/skel` directory requires the privileges of the superuser. Execute the following command to switch to the root user account. When prompted for the password, enter `netlab123`:

```
su -
```

```
[clima@centos ~]$ su -
Password:
Last login: Wed Oct  9 17:12:36 UTC 2019 on console
[root@centos ~]#
```

Step 31

Copy the customized `.bash_profile` file from the home directory of the `clima` user by executing the following command:

```
cp ~clima/.bash_profile  /etc/skel
```

When prompted to overwrite the existing file, answer `y`:

```
[root@centos ~]# cp ~clima/.bash_profile  /etc/skel
cp: overwrite `/etc/skel/.bash_profile'? y
```

By placing this new `.bash_profile` into the `/etc/skel` directory, any new `BASH` account will automatically have the contents of this file placed in the home directory of the new user account. This allows the administrator the ability to create customizations that will affect all new user accounts.

Step 32

The `useradd` command can be used to create a new user account. Execute the following command to create a new user named `testuser`:

```
useradd testuser
```

```
[root@centos ~]# useradd testuser
```

The `useradd` command will be covered in greater detail in a later module.

Step 33

View the `.bash_profile` file in the `testuser` home directory to verify that the new `.bash_profile` file has been copied to the user's home directory. Execute the following command:

```
cat ~testuser/.bash_profile
```

```
[root@centos ~]# cat ~testuser/.bash_profile
# .bash_profile


# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi


# User specific environment and startup programs


PATH=$PATH:$HOME/.local/bin:$HOME/bin:$HOME/scripts


export PATH
```

Step 34

Because the `~/.bash_profile` file is processed only once, typically when the user logs in, it is a good place to put custom environment variable assignments since these variables only need to be created once per account. The `~/ .bashrc` file is processed every time a user starts a new bash shell, such as when opening a new terminal window. The `~/.bashrc` is a good place to put aliases and functions that must be executed in order to be defined every time the user starts a new shell.

View the `.bashrc` file in the `/etc/skel` directory to see what it currently does by executing the following command:

```
cat /etc/skel/.bashrc
```

```
[root@centos ~]# cat /etc/skel/.bashrc
# .bashrc


# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi


# Uncomment the following line if you don't like systemctl's auto-paging feat
ure
:
# export SYSTEMD_PAGER=


# User specific aliases and functions
```

Recall that this file will be copied to the home directory when creating a new user account. It is executed every time a new shell is opened. The code under `# Source global definitions` checks to see if there is an `/etc/bashrc`. If that file exists, then it will be sourced. This feature allows the system administrator a place to create aliases and functions that will be applied to all user accounts.

At the bottom of the file is a comment suggesting where user-specific aliases and functions should be added.

Step 35

Before adding aliases and functions to the `/etc/skel/.bashrc` file, it is recommended that you test them in an interactive shell to make sure that they work as you would expect. First, view the aliases that are currently defined in the shell by executing the `alias` command.

```
alias
```

```
[root@centos ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
```

```
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot--show-tilde'
```

Step 36

To add an alias to your shell, use the `alias` command. Execute the following commands to define a new alias called `lt`, which will list files in reverse time order, and then display the current shell's aliases:

```
alias lt='ls -lrt'
alias
```

```
[root@centos ~]# alias lt='ls -lrt'
[root@centos ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias lt='ls -lrt'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot--show-tilde'
```

Step 37

Make sure that the `lt` alias you defined does in fact list files in reverse by time. Execute the following command to list the `/var` directory in reverse time order:

```
lt /var
```

```
[root@centos ~]# lt /var
total 72
drwxr-xr-x 2 root root 4096 Apr 11  2018 yp
drwxr-xr-x 2 root root 4096 Apr 11  2018 preserve
drwxr-xr-x 2 root root 4096 Apr 11  2018 opt
drwxr-xr-x 2 root root 4096 Apr 11  2018 nis
drwxr-xr-x 2 root root 4096 Apr 11  2018 local
drwxr-xr-x 2 root root 4096 Apr 11  2018 gopher
drwxr-xr-x 2 root root 4096 Apr 11  2018 games
drwxr-xr-x 2 root root 4096 Apr 11  2018 adm
lrwxrwxrwx 1 root root    6 Dec  4  2018 run -> ../run
lrwxrwxrwx 1 root root   11 Dec  4  2018 lock -> ../run/lock
lrwxrwxrwx 1 root root   10 Dec  4  2018 mail -> spool/mail
drwxr-xr-x 1 root root 4096 Jan 29  2019 kerberos
drwxr-xr-x 1 root root 4096 Mar 29  2019 spool
drwxr-xr-x 1 root root 4096 Mar 29  2019 empty
drwxr-xr-x 1 root root 4096 Apr 24 19:20 db
drwxr-xr-x 1 root root 4096 Jul  1 14:57 cache
drwxrwxrwt 1 root root 4096 Jul  1 14:57 tmp
drwxr-xr-x 1 root root 4096 Jul  1 14:57 log
drwxr-xr-x 1 root root 4096 Jul  1 14:58 lib
```

The dates and times are displayed in descending order in the output above. Your output may vary, but should be shown in reverse time order.

Step 38

To remove the alias definition from your current shell session, use the `unalias` command with the name of the alias. Execute the following commands to remove the `lt` alias and verify that it is removed:

```
unalias lt
alias
```

```
[root@centos ~]# unalias lt
[root@centos ~]# alias
```

```
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot--show-tilde'
```

Step 39

While aliases are typically shorter names for longer commands, functions allow you to execute multiple commands and work with arguments that are passed to these commands. To view the functions that are defined within your current Bash shell, execute the following command:

```
declare -f
```

```
[root@centos ~]# declare -f
```

The lack of any output indicates that no functions are defined.

Step 40

A function can be defined by first typing the function name to be assigned, followed by a pair of parentheses, and then the commands and options of the function contained between a pair of curly braces. For example, the following function will list files in reverse by size:

```
lS() { ls -lrS $@; }
```

The function uses the `ls` command with the options `-lrS` to list files in reverse by size and the variable `$@` to refer to any arguments that might have been passed to the function. Execute the following commands to create the function and verify it exists (note, the function name is `lS`, a lowercase `l` and a capital `S`):

```
lS() { ls -lrS $@; }
declare -f
```

```
[root@centos ~]# lS() { ls -lrS $@; }
```

```
[root@centos ~]# declare -f
lS ()
{
    ls --color=auto -lrS $@
}
```

Recall that Linux is case-sensitive. As a result, `ls` and `lS` are different commands/functions.

Step 41

Check to make sure this new function performs properly, listing the files in descending order by size, by executing the following command:

```
lS /var
```

```
[root@centos ~]# lS /var
total 64
lrwxrwxrwx 1 root root    6 Dec  4  2018 run -> ../run
lrwxrwxrwx 1 root root   10 Dec  4  2018 mail -> spool/mail
lrwxrwxrwx 1 root root   11 Dec  4  2018 lock -> ../run/lock
drwxr-xr-x 2 root root 4096 Apr 11  2018 yp
drwxrwxrwt 1 root root 4096 Jul  1 14:57 tmp
drwxr-xr-x 1 root root 4096 Mar 29  2019 spool
drwxr-xr-x 2 root root 4096 Apr 11  2018 preserve
drwxr-xr-x 2 root root 4096 Apr 11  2018 opt
drwxr-xr-x 2 root root 4096 Apr 11  2018 nis
drwxr-xr-x 1 root root 4096 Jul  1 14:57 log
drwxr-xr-x 2 root root 4096 Apr 11  2018 local
drwxr-xr-x 1 root root 4096 Jul  1 14:58 lib
drwxr-xr-x 1 root root 4096 Jan 29  2019 kerberos
drwxr-xr-x 2 root root 4096 Apr 11  2018 gopher
drwxr-xr-x 2 root root 4096 Apr 11  2018 games
drwxr-xr-x 1 root root 4096 Mar 29  2019 empty
drwxr-xr-x 1 root root 4096 Apr 24 19:20 db
drwxr-xr-x 1 root root 4096 Jul  1 14:57 cache
drwxr-xr-x 2 root root 4096 Apr 11  2018 adm
```

Step 42

The `unset` command can be used to remove a function's definition from within a shell. Execute the following commands to unset the function and verify that it no longer exists:

```
unset lS
declare -f
```

```
[root@centos ~]# unset lS
[root@centos ~]# declare -f
```

Step 43

After having tested the `lt` alias and the `lS` function, you are ready to make those available to users by adding them to the `.bashrc` file in the `/etc/skel` directory. Start the `nano` editor to edit this file by executing the following command:

```
nano /etc/skel/.bashrc
```

```
  GNU nano 2.3.1                    File: /etc/skel/.bashrc


# .bashrc


# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi


# Uncomment the following line if you don't like systemctl's auto-paging feat
ure$
# export SYSTEMD_PAGER=


# User specific aliases and functions
```

```
                              [ Read 11 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Step 44

Move the cursor to the end of the last line of the file by using the **Arrow keys** on your keyboard. Then, type the alias and function definitions as follows:

```
Down Arrow (x11)
alias lt='ls -lrt'
lS() { ls -lrS $@; }
```

```
  GNU nano 2.3.1            File: /etc/skel/.bashrc                         Modifie
d


# .bashrc


# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi


# Uncomment the following line if you don't like systemctl's auto-paging feat
ur$
# export SYSTEMD_PAGER=


# User specific aliases and functions
alias lt='ls -lrt'
lS() { ls -lrS $@; }
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Step 45

Exit the `nano` editor by pressing **Ctrl+X**. Type y to save the changes and then press the **Enter** key:

```
Ctrl+X
y
Enter
```

Step 46

Test the new `/etc/skel/.bashrc` file by sourcing it. Execute the following command:

```
source /etc/skel/.bashrc
```

```
[root@centos ~]# source /etc/skel/.bashrc
```

If any errors are shown, go back and fix them in the `/etc/skel/.bashrc` file and repeat this step.

Step 47

Verify that both the new alias and the new function are now defined within your shell by executing the following commands:

```
alias
declare -f
```

```
[root@centos ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

```
alias ls='ls --color=auto'
alias lt='ls -lrt'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show
-tilde'
[root@centos ~]# declare -f
lS ()
{
    ls --color=auto -lrS $@
}
```

Recall that the files in the `/etc/skel` directory only affect new user accounts. To affect existing users, you would either have to copy the alias and function definition into each user's `~/.bashrc` file or place these definitions in the `/etc/bashrc` file.

This lab is complete.

THEORY - Shell Scripts

```
Please read Advanced Shell Features first
```

2.1 Introduction

When you find yourself typing the same sets of commands on a regular basis, it is time to consider creating a shell script. At the most simple level, a shell script is just a set of commands placed into a text file, which, when invoked as a program, will execute each individual command in order. In the previous chapter, the following command list was used:

```
clima@centos:~$ date;who;uptime
Fri Sep 20 00:50:18 UTC 2020
sysadmin console      Sep 20 23:16
 00:50:18 up 9 days, 23:32,  1 user,  load average: 0.05, 0.15, 0.13
```

To make a script out of these commands, simply place them in a file and make it executable:

```
clima@centos:~$ echo 'date;who;uptime' > my.sh
clima@centos:~$ chmod u+x ./my.sh
clima@centos:~$ ./my.sh
Fri Sep 20 00:50:18 UTC 2020
sysadmin console      Sep 20 23:16
 00:50:18 up 9 days, 23:32,  1 user,  load average: 0.05, 0.15, 0.13
```

While this method may produce an immediate script, it is not the recommended way to build scripts that you will add to, change, or modify as time goes on.

Shell scripting can be quite complex and utilize powerful features. In addition to all of the features that the interactive shell provides (i.e., variables, aliases, functions), administrators can also make use of programming features such as *if statements*, *while loops* and *for loops*, to name a few. While not considered by some as robust as more advanced programming languages, shell scripts can be very powerful because they make direct use of shell commands.

2.2 Shell Scripting Basics

To create a shell script, use the following steps:

- Use a text editor to create a file with the commands that you want to execute.
- Place the following line at the beginning of the file to indicate that the script is to be run in the Bash shell.

```
#!/bin/bash
```

- Use the `chmod` command to make the file executable:

```
chmod a+x file_name
```

In the following example, a script has been created that will create a report of disk space usage in the `/usr/share/doc` directory. Notice that this is very similar to the function example from the preceding chapter. It is not uncommon to convert a function into a script when it becomes more complex.

```
clima@centos:~$ more report.sh
```

```
#!/bin/bash
                                                                    cd /usr
/share/doc
echo "Document directory usage report" > /tmp/report
date >> /tmp/report
pwd >> /tmp/report
du -sh . >> /tmp/report
clima@centos:~$ chmod a+x report.sh
clima@centos:~$ ./report.sh
clima@centos:~$ more /tmp/report
Document directory usage report
Mon Jun 15 06:41:17 UTC 2015
/usr/share/doc
6.6M    .
```

The example above is for demonstration purposes. The output in the example above may not match the output in our virtual environment.

Note

While the two above examples demonstrate how to create a script from a command line session, a more sustainable method for creating scripts is to open an editor, such as `nano` or `vi`, and create a script from scratch.

As an example, to create a script in `vi`, the user would decide on the script name, then open a `vi` session by typing the following:

```
clima@centos:~$ vi scriptfile1.sh
```

In the `vi` editor, the user would enter *Insert* mode by pressing the **i** key and then on the first line of the file, enter the string: `#!/bin/bash`. This line is optional, but it is good practice to include it (see the *Consider This* box which follows this example).

Next, the user would press the **Enter** key to move down two lines and begin entering the commands that are desired. Below is an example of a simple script that will `echo` some text to the screen when running and then execute several commands.

```
#!/bin/bash


echo This is the $0 script that is running
echo Today's Date and Time of Report >> system_users_report.txt
date >> system_users_report.txt
echo The users currently on the system are:  >> system_users_report.txt
w >> system_users_report.txt
echo The users are running these commands >> system_users_report.txt
who >> system_users_report.txt
cat system_users_report.txt
~
~
~
"scriptfile1.sh" 10 lines, 387 characters
```

Once the script is added, press the **ESC** key to leave *Insert* mode and press **ZZ** to save the file and exit `vi`.

The script can then be run to see the results by executing:

```
clima@centos:~$ bash system_user_report.sh
```

The report is made by the *echo'ing* of the text to the report file, each `echo` line followed by the output of the command it mentions. At the end of the script, the contents of the report file are sent to the standard output or console, so the user running the report can see the results.

The use of the double greater than `>>` characters when sending text and data to the report file will initially create the `system_users_report.text` file, but each additional time it is run, it will append the new report output to the bottom of the current file, not overwrite the report file.

Consider This

The first line of the script should include a path to the Bash shell executable. The `#!` tells the system that this first line is intended to provide this path. Typically a pound sign `#` character is used to comment a line, but when placed at the beginning of the first line and followed by an exclamation mark `!` character, it is used to tell the system which executable to use to run the program.

2.3 Script Placement

Notice that once a script has been set to be executable, such as the script in a previous example, a `./` character combination was placed before the script name to indicate the current directory. While this technique can always be used to run a script, consider placing the script in a location that is specified by the `PATH` variable.

While the exact value of the `PATH` variable will vary based on which Linux distribution is being used as well as individual customizations, a typical `PATH` variable will include the following directories:

```
clima@centos:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
```

With administration rights, a script can be placed in any of the directories listed in the `PATH` variable. However, the ideal location for scripts designed to be shared by regular users would be the `/usr/local/bin` directory. For scripts designed to be executed by administrators, place the script in the `/usr/local/sbin` directory.

Note

The PATH variable contains a list of directories that are used to search for commands entered by the user.

Regular users don't have the rights to place files in these directories. For a user's own script, the ideal place to put the script is the `/home/USER/bin` directory, where `USER` is the name of the current user account. For example, `sysadmin` should place scripts in the `/home/sysadmin/bin` directory.

By placing scripts in these directories, they will be easier to find as well as easier to execute.

2.4 Script Permissions

Note

Permissions determine the level of access that a user has on a file. When a user runs a program and the program accesses a file, then the permissions are checked to determine if the user has the correct access rights to the file.

There are three types of permissions: *read*, *write*, and *execute*. Each has a different meaning depending on whether they are applied to a *file* or a *directory*.

Read:

- On a *file*, this allows processes to read the contents of the file, meaning the contents can be viewed and copied.
- On a *directory*, file names in the directory can be listed, but other details are not available.

Write:

- A *file* can be written to by a process, so changes to a file can be saved. Note that the *write* permission usually requires the *read* permission on the file to work correctly with most applications.
- On a *directory*, files can be added to or removed from the directory. Note that the *write* permission requires the *execute* permission on the directory to work correctly.

Execute:

- A *file* can be executed or run as a process.
- On a *directory*, the user can use the `cd` command to "get into" the directory and use the directory in a path name to access files and, potentially, subdirectories under this directory.

Permissions are covered in complete detail in the NDG Introduction to Linux I course.

Typically, a script will need to have both *read* and *execute* permissions for the user who is attempting to execute it. In the previous example, the command `chmod a+x file_name` provided execute permission to all users (it was assumed that the read permission was already provided for all users).

If the script is designed for private use, consider only providing the execute permission for the user: `chmod u+x file_name`.

System administrators have historically enabled the *setuid* permission on scripts that they created in UNIX so that they could execute as the root user. When the *setuid* permission is set on an *executable binary file* (a *program*), the binary file is run as the owner of the file, not as the user who executed it. This was generally a bad idea as someone may be able to utilize an exploit and use the script to gain administrative rights to the system.

The Linux kernel does not allow for shell script files to be executed with *setuid* permission. In order to execute a shell script as the root user, an administrator can log in as root, and use `su`, or `sudo` to run the script directly. If the script must be executed with root privileges by an ordinary user, then a simple *wrapper* around the script can be created in a language that compiles to binary, and the binary executable can be *setuid*.

A wrapper is a script whose purpose is to simply configure and launch another program. One way to create a wrapper script is to use the `exec` command, which takes another command to execute as an argument. A wrapper script often uses the following as the last line of the script to execute another program.

```
exec program
```

Recall that a command is a program that, in most cases, creates a process when executed and then returns to the current process, the shell. However, the `exec` command behaves differently. The `exec` command replaces the current process with a specified command. If no command is specified, the `exec` command spawns a new shell process and returns a shell prompt. Running the `exec` command with no argument allows a user to change the current shell environment, including using redirection. For example, it is possible to use `exec` to redirect all output to a file:

```
clima@centos:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
clima@centos:~$ exec > my_commands.txt
clima@centos:~$ ls
```

Notice in the example above, there was no output for the `ls` command because the `exec` command has replaced the current process and is redirecting all output to the `my_commands.txt` file. Exit the exec session and then open a new shell:

```
clima@centos:~$  exit
clima@centos:~$ ls my_commands.txt
my_commands.txt
```

```
clima@centos:~$ cat my_commands.txt
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
my_commands.txt
clima@centos:~$ rm my_commands.txt
```

The `exec` command is also commonly used in scripts to start a new shell with a clean environment as read by the shell startup files. The following is an example of how the `exec` command can be used to change the shell prompt, which is then changed back to the original prompt from the shell startup files by using the `bash` command as an argument:

```
clima@centos:~$ PS1="NewPrompt>> "
NewPrompt>>
NewPrompt>> exec bash
clima@centos:~$
```

2.5 Command Substitution

A shell feature that is very useful when creating scripts is called *command substitution*. This allows the script to execute a shell command and redirect the output of the command into a variable instead of having the output displayed on the terminal. For example, to capture the date and time of when a script started and ended, use the following syntax:

```
clima@centos:~$ start=`date`
```

The backquote characters around `date` tell the shell to execute the command and replace `` `date` `` with the output of the `date` command. This output is then assigned to the start variable:

```
clima@centos:~$ echo $start
Fri Sep 18 03:58:34 UTC 2020
```

The backquote character method of running a subcommand has its origins in the original Bourne Shell. In the Bash shell, you can also use the syntax of $(*command*):

```
clima@centos:~$ end=$(date)
clima@centos:~$ echo $start $end
Fri Sep 18 03:58:34 UTC 2020 Fri Sep 18 03:59:22 UTC 2020
```

Both techniques accomplish the same task; most Bash shell programmers prefer to use the $(*command*) syntax as they consider it easier to read (backquotes sometimes look like single quotes).

Command substitution can also be used to insert the output of a command as another command's argument. For example:

```
clima@centos:~$ echo "Today is $(date)"
Today is Fri Sep 18 04:01:39 UTC 2020
```

This can be very useful when you have a file that contains data like user names, directory names, or file names. For example, in the following demonstration the `dirs.txt` file contains three directory names that are passed into the `du` command as arguments by using command substitution:

The examples below are for demonstration purposes. The output in the examples may not match the output in our virtual environment.

```
clima@centos:~/test$ more dirs.txt
/usr/share/doc
/usr/local
/etc/magic
clima@centos:~/test$ du -sh $(cat dirs.txt)
6.6M    /usr/share/doc
4.0K    /usr/local
4.0K    /etc/magic
```

While command substitution is often used on the command line, it is also a very powerful shell scripting feature. The following script demonstrates using command substitution:

```
clima@centos:~/test$ more report2.sh
#!/bin/bash
start=$(date)
cd /usr/share/doc
echo "Document directory usage report" > /tmp/report
du -sh $(cat ~/test/dirs.txt) >> /tmp/report
echo "Start of report: $start" >> /tmp/report
echo "End of report: $(date)" >> /tmp/report
clima@centos:~/test$ ./report2.sh
clima@centos:~/test$ more /tmp/report
Document directory usage report
6.6M    /usr/share/doc
4.0K    /usr/local
4.0K    /etc/magic
Start of report: Mon Jun 15 06:58:23 UTC 2015
End of report: Mon Jun 15 06:58:23 UTC 2015
```

2.6 read Statement

The `read` statement can be used to gather information from the user who is running the script. The user will be presented with a blinking cursor that will accept keyboard input and place what the user types into one or more variables. The basic syntax of the `read` command is:

```
read variable
```

For example, the following command will read the user's input into a variable called `name`:

```
read name
```

It would be helpful for the user to know what sort of input is being requested. The `-p` option to the `read` command allows you to issue a prompt:

```
read -p "Please enter your name: " name
```

In the following example script, user input is used to determine which directory name to use to execute the `du` command:

The example below is for demonstration purposes. The output in the example below may not match the output in our virtual environment.

```
#!/bin/bash


read -p "Enter a directory: " dir
start=$(date)
echo "Document directory usage report" > /tmp/report
du -sh $dir >> /tmp/report
echo "Start of report: $start" >> /tmp/report
echo "End of report: $(date)" >> /tmp/report
clima@centos:~/test$ ./report3.sh
Enter a directory: /etc/magic
clima@centos:~/test$ more /tmp/report
Document directory usage report
4.0K    /etc/magic
Start of report: Mon Jun 15 07:03:56 UTC 2015
End of report: Mon Jun 15 07:03:56 UTC 2015
```

2.7 test Statement

While writing shell scripts, there will be times when an administrator will want to execute some commands based on if a *conditional statement* is true or false. A conditional statement can be used to determine the following:

- If two string variables or values match (or don't match)
- If two numeric variables or values match (or don't match)
- The status of files (if file exists, if file is a directory, etc.)
- If a command completes successfully

Typically, programmers will use the `test` command as a conditional statement. The `test` command can perform numeric comparisons, string comparisons, and file testing operations.

The following chart illustrates some of the comparisons that are possible:

Type	Symbol	Example
True if length of string is zero	`-z`	`-z` *string*
True if length of string is not zero	`-n`	`-n` *string*
True if strings are equal	`=`	*string1* `=` *string2*
True if strings are not equal	`!=`	*string1* `!=` *string2*
True if integers are equal	`-eq`	*int1* `-eq` *int2*
True if integers are not equal	`-ne`	*int1* `-ne` *int2*
True if first integer is greater than second integer	`-gt`	*int1* `-gt` *int2*
True if first integer is greater than or equal to second integer	`-ge`	*int1* `-ge` *int2*
True if first integer is less than second integer	`-lt`	*int1* `-lt` *int2*
True if first integer is less than or equal to second integer	`-le`	*int1* `-le` *int2*
True if file is a directory	`-d`	`-d` *file*
True if file is a plain file	`-f`	`-f` *file*
True if file exists	`-e`	`-e` *file*
True if file has read permission for current user	`-r`	`-r` *file*
True if file has write permission for current user	`-w`	`-w` *file*
True if file has execute permission for current user	`-x`	`-x` *file*

While you can execute the `test` command specifically, within an `if` statement, you can have the test statement implicitly called by placing the arguments of the test statement within square brackets. So, instead of writing…

```
if test $var -eq 7
```

…you could write:

```
if [ $var -eq 7 ]
```

Note
When using the square bracket technique, make sure you place space characters in front of and after the brackets. If you don't, an error will occur.

2.8 if Statement

A script performs different functions based on tests, called *branching*. The `if` statement is the basic operator to implement branching.

The basic syntax of an `if` statement is:

```
if COND
then
```

```
   TRUE_COMMAND(S)
fi
```

Replace *COND* with a conditional expression, like a test statement. Replace *TRUE_COMMAND(S)* with the commands you want to execute if the *COND* statement returns true. The word `if` spelled backward, `fi`, ends the `if` statement.

In the example above, if the *COND* statement returns false, then no commands would be executed. To have commands executed if the *COND* statement returns false, include an `else` clause:

```
if COND
then
   TRUE_COMMAND(S)
else
   FALSE_COMMAND(S)
fi
```

If the conditional statement returns false, all of the commands after `else` and before `fi` will be executed.

You might also want to perform a secondary check; this can be accomplished with an `else if` statement, which in Bash is denoted with the keyword `elif`:

```
if COND1
then
   TRUE1_COMMAND(S)
elif COND2
then
   TRUE2_COMMAND(S)
else
   FALSE_COMMAND(S)
fi
```

If the first condition *COND1* is true, then the *TRUE1_COMMAND(S)* would be executed. If the first condition is false and the second condition *COND2* is true, then the *TRUE2_COMMAND(S)* is executed. If neither condition is true, then the *FALSE_COMMAND(S)* is executed.

Examples of the `if` Statement

The following examples are provided on the command line for visual demonstration purposes. Typically, you would see these statements in a Bash shell script, as will be demonstrated later.

In the first example, a conditional check is performed on the `$USER` variable that contains the current user account name:

```
clima@centos:~$ echo $USER
sysadmin
clima@centos:~$ if [ $USER = 'sysadmin' ]
>                     then
>                         echo 'hi sysadmin!'
>                     fi
```

```
hi sysadmin!
```

Notice what happens in the next example when a string is used in a numeric comparison. An error like the following would cause a Bash shell script to exit prematurely:

```
clima@centos:~$ echo $USER
sysadmin
clima@centos:~$ if [ $USER -eq 'sysadmin' ]
>                     then
>                         echo 'hi sysadmin!'
>                     fi
-bash: [: sysadmin: integer expression expected
```

In the next example, a conditional check is performed to determine if the `/etc/passwd` file is readable for the current user:

```
clima@centos:~$ ls -l /etc/passwd
-rw-r--r-- 1 root root 1602 May  2 16:18 /etc/passwd
clima@centos:~$ if [ -r /etc/passwd ]
>                     then
>                         echo "/etc/passwd is readable"
>                     fi
/etc/passwd is readable
```

The final two examples below both perform the same function, to indicate the type of operating system. The first uses two isolated `if` statements, while the second uses an `else` statement, nesting an `if` statement inside:

```
#!/bin/bash
ARCHITECTURE=`uname -m`
if [ $ARCHITECTURE = "i686" ]
then
        echo "32 Bit Operating System Detected"
fi
if [ $ARCHITECTURE = "x86_64" ]
then
        echo "64 Bit Operating System Detected"
fi
#!/bin/bash
ARCHITECTURE=`uname -m`
if [ $ARCHITECTURE = "i686" ]
then
        echo "32 Bit Operating System Detected"
else
        if [ $ARCHITECTURE = "x86_64" ]
        then
```

```
                echo "64 Bit Operating System Detected"
        fi
fi
```

The output in the example below may not match the output in our virtual environment.

```
clima@centos:~$ ./arch.sh
64 Bit Operating System Detected
```

Further examples of using the `if` statement will be provided throughout the remainder of this chapter.

2.9 Test Return Values

Every command has a return or exit status value that can be used to determine if the command succeeded or failed. This return value is a number that can only be an integer value from 0 to 255. A value of 0 means the command executed successfully while a positive integer value means the command failed.

Commands can fail for several reasons, including bad user input or file permission issues. When you feel that a command may potentially fail, it is a good idea to check the exit status of the command after the script executes that command.

To see the exit status of a command, view the `$?` variable:

```
clima@centos:~$ grep sysadmin /etc/passwd
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
clima@centos:~$ echo $?
0
clima@centos:~$ grep sysadmin /etc/shadow
grep: /etc/shadow: Permission denied
clima@centos:~$ echo $?
2
```

In the previous example, the first `grep` command executed successfully, resulting in a value of `0` in the `$?` variable. The second `grep` command failed, resulting in a value of `2` in the `$?` variable.

Typically, the developers of these commands will use different fail values for different situations. For example, for the `grep` command, a value of `2` means an error occurred, while a value of `1` means no lines matching the pattern were found. Sometimes these different values are documented in the `EXIT STATUS` section of the command's man page.

When executing a command that may fail, an administrator might want to get in the habit of discarding the command's Standard Error (STDERR). Often these error messages, like the message from the second `grep` command in the previous example, only confuse users who are executing the script. Recall that to discard STDERR, execute the command as follows:

```
clima@centos:~$ grep sysadmin /etc/shadow 2> /dev/null
```

The most common case where a command in a shell script may fail is when user input is requested. For example, in a previous example script, the following lines were executed:

```
read -p "Enter a directory:" dir
du -sh $dir >> /tmp/report
```

If the user provides the name of a directory that doesn't exist, then the `du` command will fail and produce an error message to STDERR:

```
clima@centos:~$ ./report.sh
Enter a directory: /junk
du: cannot access `/junk': No such file or directory
```

The output in the example above may not match the output in our virtual environment.

To solve this problem, do one of the following:

- Redirect the STDERR messages of the `du` command into the `/dev/null` file and then use the `if` statement to see if the value of the `$?` variable is `0` or a positive number.
- Place the `du` command with the conditional part of the `if` statement; redirection should also be used in this case. When used as a conditional statement, the return value of the command is used by the `if` statement to determine *true* or *false*.
- Before executing the `du` command, use file checking on the filename provided by the user to determine if the directory is accessible. In this case, the filename must be a directory and must have the read and execute permissions set for the current user.

Solution #1: Check the $? Variable

In the following example, the `du` command is executed, and the value of the `$?` variable is checked. If the `$?` variable is set to `0`, output is placed in the `/tmp/report` file. If the `$?` variable is set to a non-zero value, an error is displayed on the screen and in the `/tmp/report` file:

```
#!/bin/bash


read -p "Enter a directory: " dir
start=$(date)
echo "Document directory usage report" > /tmp/report
du -sh $dir >> /tmp/report 2> /dev/null
if [ $? -eq  0 ]
then
    echo "Start of report: $start" >> /tmp/report
    echo "End of report: $(date)" >> /tmp/report
else
    echo "Error, $dir could not be accessed"
    echo "Error: no report generated.  $dir not accessible" >> /tmp/report
fi
```

Solution #2: Check the Return Value of the Command

In the following example, the `du` command is executed as the condition of the `if` statement. If the `du` command is successful, the output is placed in the `/tmp/report` file. If it fails, an error is displayed on the screen and in the `/tmp/report file`:

```
#!/bin/bash


read -p "Enter a directory: " dir
```

```
start=$(date)
echo "Document directory usage report" > /tmp/report
if du -sh $dir >> /tmp/report 2> /dev/null
then
    echo "Start of report: $start" >> /tmp/report
    echo "End of report: $(date)" >> /tmp/report
else
    echo "Error, $dir could not be accessed"
    echo "Error: no report generated.  $dir not accessible" >> /tmp/report
fi
```

Solution #3: Check the Value of the Variable

In the next example, the value of the variable is checked before attempting to run the `du` command. This requires three checks:

1. Is the value of the variable a directory?
2. Does the value of the variable have read permission?
3. Does the value of the variable have execute permission?

In order to check all three of these conditions, the `-a` option (and) is used:

```
#!/bin/bash


start=$(date)
echo "Document directory usage report" > /tmp/report
read -p "Enter a directory: " dir

if [ -d $dir -a -r $dir -a -x $dir ]
then
    du -sh $dir >> /tmp/report 2> /dev/null
    echo "Start of report: $start" >> /tmp/report
    echo "End of report: $(date)" >> /tmp/report
else
    echo "Error, $dir could not be accessed"
    echo "Error: no report generated.  $dir not accessible" >> /tmp/report
fi
```

Note that `-o` can be substituted for *or* and the exclamation point `!` character can be substituted for *not*. For example, the following means *return true if* `$USER` *equals* `joe` *or* `$USER` *equals* `ted`:

```
[ $USER = "joe" -o $USER = "ted" ]
```

Another example: The following returns true if `$FILE` is not readable:

```
[ ! -r $FILE ]
```

Verify User Input

One of the biggest challenges that you will have when dealing with user input in scripts is making sure the user provides the correct type of input. For example, if the script asks the user to provide a ZIP code (assume, for this example, we are using standard United States zip codes, a 5 digit number) and the user provides `abcxyz`, this will cause problems when it tries to use the variable that the user data was assigned to:

```
clima@centos:~$ read -p "Enter your ZIP code: " zip
Enter your ZIP code: abcxyz
clima@centos:~$ if [ $zip -eq 99999 ]
>                     then
>                         echo "You are in the Mars zip code"
>                     fi
-bash: [: abcxyz: integer expression expected
```

The problem in this situation is that in order to use the `-eq` test operation, both values must be numbers. If one is not a number, then an error will occur.

To verify that the user input is an integer, use the `grep` command as demonstrated below:

```
if echo $zip | grep -E "^[0-9]$"
```

See the following for a more comprehensive example:

```
clima@centos:~$ read -p "Enter your ZIP code: " zip
Enter your ZIP code: abcxyz
clima@centos:~$ if echo $zip | grep -E '^[0-9]+$' > /dev/null 2>/dev/null
>                       then
>                           echo "thank you for the proper zip code"
>                       else
>                           echo "incorrect zip code"
>                     fi
incorrect zip code
clima@centos:~$ read -p "Enter your ZIP code: " zip
Enter your ZIP code: 99999

clima@centos:~$ if echo $zip | grep -E '^[0-9]+$' > /dev/null 2>/dev/null
>                       then
>                           echo "thank you for the proper zip code"
>                       else
>                           echo "incorrect zip code"
>                     fi
thank you for the proper zip code
```

To make the pattern as precise as possible, in this case, because a standard United States ZIP code must be only 5 digits long, the following pattern would be more precise:

```
echo $zip | grep -E '^[0-9]{5}$'
```

Or, for a modern United States ZIP code, which is 5 digits followed by a dash and four more digits:

```
echo $zip | grep -E '^[0-9]{5}-[0-9]{4}$'
```

2.10 Sending Mail to Superuser

Quite often, users will be creating scripts that are designed to be executed as the root user. Many of these scripts will also be executed automatically via a cron job, a technique that allows automatic executing of commands and scripts at specific future times.

When creating these scripts, administrators will likely wish to have information regarding the outcome of how the scripts executed sent to the email account of the root user. Typically, this will be performed based on the outcome of a conditional statement, such as the following:

```
if [ -d /data/January ]
then
     echo "January directory exists at `date`" | mail root
fi
```

2.11 while Statement

The `while` statement is used to determine if a condition is true or false; if it is true, then a series of actions take place, and the condition is checked again. If the condition is false, then no action takes place, and the program continues. The `while` statement, along with the block of statements to be executed when the condition is true, is also known as a `while` loop.

The series of statements that are to be executed when the `while` conditional test is true will be contained in a block that starts after the keyword `do` and ends at the keyword `done`. The keywords `do` and `done` are not only used to create blocks of code that may be executed repeatedly in `while` loops but are also used in `for` and `until` loops.

For example, consider the code earlier in which user input was checked to determine if it was in the correct format:

```
clima@centos:~$ read -p "Enter your ZIP code: " zip
Enter your ZIP code: abcxyz
clima@centos:~$ if echo $zip | grep -E '^[0-9]+$' > /dev/null 2>/dev/null
>                       then
>                           echo "thank you for the proper zip code"
>                       else
>                           echo "incorrect zip code"
>                   fi
incorrect zip code
clima@centos:~$ read -p "Enter your ZIP code: " zip
Enter your ZIP code: 99999
clima@centos:~$ if echo $zip | grep -E '^[0-9]+$' > /dev/null 2>/dev/null
>                       then
```

```
>                           echo "thank you for the proper zip code"
>                       else
>                           echo "incorrect zip code"
>                   fi
thank you for the proper zip code
```

If the user provides invalid input, then the program doesn't provide another chance to provide valid input. The following example (placed in an actual script), allows the user the opportunity to type the correct value after an error occurs:

```
read -p "Enter a valid ZIP code: " zip


while echo $zip | grep -E -v '^[0-9]{5}$' > /dev/null 2>/dev/null
do
   echo "$zip is not a valid zip code"
   echo "The zip code must consist of all numbers"
   read -p "Enter a valid ZIP code: " zip
done


echo "Thank you, $zip is a correct zip code"
```

The statements between the `do` and `done` keywords will be executed until the value of the `zip` variable contains exactly five digits. Once that value is entered, execution would resume at the `echo` statement following the `done` keyword.

2.12 for Statement

The `for` statement is extremely valuable when you want to perform an operation on multiple items. For example, suppose an administrator wants to create five user accounts named `ted`, `fred`, `ned`, `jed`, and *bob*. He could execute five separate commands:

```
useradd ted
useradd fred
useradd ned
useradd jed
useradd bob
```

But what if it was twenty user accounts? Or what if the administrator wanted to not only create the account, but set a password and use the `chage` command to specify some password aging rules. That would involve a lot of typing.

Using a `for` loop, this process becomes a lot easier. The basic syntax of a `for` loop is:

```
for name in list_of_values
do
    FOR_COMMANDS
done
```

The *name* is a variable that will be used to enumerate the items in the *list_of_values*. The *list_of_values* is a list, like a list of user names, file names, or directory names. With the `for` loop, the *name* variable will be assigned from the *list_of_values*, one at a time. For example, consider the following program:

The output in the example below may not match the output in our virtual environment.

```
root@localhost:~# more create.sh
#!/bin/bash
echo -e "User\tPassword" > /root/password
echo -e "----\t--------" >> /root/password

for name in ted fred ned jed bob
do
    useradd $name
    pw=$(tr -dc A-Za-z0-9_ < /dev/urandom | head -c 12 | xargs)
    echo -e "$name:\t$pw" >> /root/password
    echo $pw | passwd --stdin $name
    chage -M 90 -m 5 -W 10 $name
done
```

This small, but powerful program will create five user accounts, assign random passwords (generated by the `tr -dc A-Za-z0-9_ < /dev/urandom | head -c 12 | xargs` command line) to each account, and assigns password aging rules to each of the new accounts. Notice that the passwords are placed in the `/root/password` file for the root user to provide to the new users (note that this script needs to be executed as the root user).

What makes this program even more powerful is that it can be so flexible. For example, instead of putting the new user account names in the script itself, they could be gathered from user input:

```
#!/bin/bash
echo -e "User\tPassword" > /root/password
echo -e "----\t--------" >> /root/password

read -p "Enter account names separated by spaces: " accounts
for name in $accounts
do
    useradd $name
    pw=$(tr -dc A-Za-z0-9_ < /dev/urandom | head -c 12 | xargs)
    echo -e "$name:\t$pw" >> /root/password
    echo $pw | passwd --stdin $name
    chage -M 90 -m 5 -W 10 $name
done
```

Or, these user names could be derived from a file:

```
#!/bin/bash
echo -e "User\tPassword" > /root/password
echo -e "----\t--------" >> /root/password
```

```
for name in `cat /root/users`
do
    useradd $name
    pw=$(tr -dc A-Za-z0-9_ < /dev/urandom | head -c 12 | xargs)
    echo -e "$name:\t$pw" >> /root/password
    echo $pw | passwd --stdin $name
    chage -M 90 -m 5 -W 10 $name
done
```

2.13 seq Statement

Suppose an administrator needs to create twenty accounts for some temporary employees. The accounts don't need special names, they could be named `user1`, `user2`, `user3`, etc. For situations like this, the `seq` statement is very useful.

The `seq` statement can generate integer values very quickly. For example, the following command would generate the integer values from 1 to 20:

```
clima@centos:~$ seq 1 20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

The `seq` command can also accept three arguments: an initial value, a step value, and a final value. For example, to start at 0, and increment the value by 10 until reaching 100, you could execute:

```
clima@centos:~$ seq 0 10 100
```

```
0
10
20
30
40
50
60
70
80
90
100
```

Since step values can be negative as well, `seq` can create a sequence of decreasing numbers. For example, to create a sequence that starts at 12, and decreases by 3 until it reaches -12, execute:

```
clima@centos:~$ seq 12 -3 -12
12
9
6
3
0
-3
-6
-9
-12
```

LAB - Shell Scripts

2.1 Step 1

For this lab we had been using the CentOS PC virtual machine. (but should work fine with your Ubuntu – most of the time)

To allow this section to stand alone, the scripts that you will create will be placed in the `/home/sysadmin/bin` directory, which is already contained in the `PATH` variable. By placing scripts in this directory, you will be able to simply type the name of your script file to be able to execute it, after you have set proper permissions on it.

First, execute the `cd` command to make sure you are located in your home directory. Then, create the `bin` directory in your home directory by executing the `mkdir` command. Finally, change to the `bin` directory by executing the `cd` command:

```
cd
mkdir bin
cd bin
```

```
[clima@centos ~]$ cd
[clima@centos ~]$ mkdir bin
[clima@centos ~]$ cd bin
[clima@centos bin]$
```

2.2 Step 2

Use the `nano` editor to create a script file named `forcount.sh`. Execute the following command:

```
nano forcount.sh
```

```
[clima@centos bin]$ nano forcount.sh
  GNU nano 2.3.1              File: forcount.sh

                                [ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.3 Step 3

Every script file should begin with a shebang (also called the hashbang), specifically, the hash `#` and exclamation `!` characters. The first line of the script file should be `#!`, followed by the path the command interpreter will use to execute the script. In the case of all these bash scripts, the first line of the files should be: `#!/bin/bash`.

While the first line shebang tells the system which shell to use to run the script, any other line that begins with the `#` character will be considered a comment and will be ignored by the shell.

Type `#!/bin/bash` on the first line and press **Enter**.

```
#!/bin/bash
```

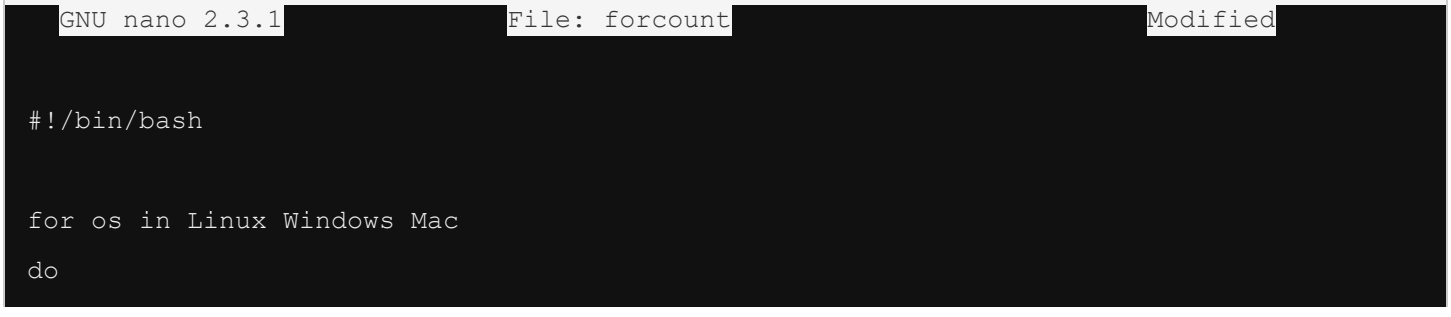
2.4 Step 4

The `for` statement can be executed in several ways, which you will explore in this script. The first line of a `for` statement sets a variable to some value. Following the first line, the do statement marks the beginning of the block of statements that will be executed for each value of the variable set on the first line.
The `done` statement marks the end of the block of statements that will be executed. Since this block of statements is executed over and over for each value of the variable, the `for` statement is sometimes known as a `for` loop. A common `for` statement may use a list of values to assign to the variable.

2.5 Step 5

In the following example, the `os` variable will be assigned each value in this list: `Linux Windows Mac`. In the `do/done` block, the `echo` command will print the value of the `os` variable. Add the following text to the script:

```
for os in Linux Windows Mac
do
  echo $os
done
```



```
  echo $os
done
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Although not required, the indentation of the statements within the `do/done` block is done to create a visual effect indicating these statements are within the block.

2.6 Step 6

Exit the `nano` editor by pressing **Ctrl+X**:

```
Ctrl+X
```

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No           ^C Cancel
```

Type `y` to save the changes and then press the **Enter** key:

```
y
Enter
```

```
File Name to Write: forcount.sh
^G Get Help         M-D DOS Format      M-A Append          M-B Backup File
^C Cancel           M-M Mac Format      M-P Prepend
```

2.7 Step 7

Change the permissions on the script, so that it can be executed directly. First, execute the following `chmod` command and then execute the command by typing in the command name and pressing the **Enter** key:

```
chmod a+x forcount.sh
forcount.sh
```

```
[clima@centos bin]$ chmod a+x forcount.sh
[clima@centos bin]$ forcount.sh
```

```
Linux
Windows
Mac
```

The three lines output above show the value of the `os` variable each time the `do/done` block executed.

2.8 Step 8

Execute the `nano` command again to edit the `forcount.sh` file:

```
nano forcount.sh
```

```
  GNU nano 2.3.1            File: forcount.sh


#!/bin/bash

for os in Linux Windows Mac
do
  echo $os
done









                          [ Read 6 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.9 Step 9

The `seq` command can be used to generate a list or sequence of numbers. For example, to generate the sequence, `1 2 3 4 5`, you would execute the `seq 1 5` command. Generally, sequences start with the first number and continue one by one, ending with the last number.

If a third argument is used with the `seq` command, then the third value becomes the last number that will be generated by the sequence, and the second value becomes the increment between each number. For example, executing the `seq 10 -1 0` command would generate the sequence `10 9 8 7 6 5 4 3 2 1 0`. The sequence still starts at the first number, ends at the third number, and decreases by one each time because the second number has a value of `-1`.

Often, it is necessary to have a `for` statement execute for a series of numbers, so the output of the `seq` command is often substituted into the `for` statement to provide these numbers. To perform command

substitution, the `seq` statement is enclosed with backquotes such as `` `seq 1 5` `` or with the dollar sign `$` and a pair of parentheses, such as `$(seq 1 5)`.

Add the following six lines to the script file:

```
for num in $(seq 10 -1 0)
do
  echo "$num seconds until blastoff"
  sleep 1
done
echo BLASTOFF
```

```
  GNU nano 2.3.1            File: forcount.sh                         Modified

#!/bin/bash

for os in Linux Windows Mac
do
  echo $os
done

for num in $(seq 10 -1 0)
do
  echo "$num seconds until blastoff"
  sleep 1
done
echo BLASTOFF




^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

In this new `for` loop, the `$(seq 10 -1 0)` command returns a list of values from `10` to `0`. Each value is assigned one at a time to the `num` variable. The `echo` command will print the value of the `num` variable, followed by the string `seconds until blastoff`. The program then pauses for one second and the loop repeats until there are no more values to assign to the `num` variable.

2.10 Step 10

Exit the `nano` editor by pressing **Ctrl+X**. Type `y` to save the changes, and press the **Enter** key:

```
Ctrl+X
y
```

```
Enter
```

2.11 Step 11

Once you have made the script executable, it stays that way, so you don't need to make it executable again. You can run the script again by executing the following command:

```
forcount.sh
```

```
[clima@centos bin]$ forcount.sh
Linux
Windows
Mac
10 seconds until blastoff
9 seconds until blastoff
8 seconds until blastoff
7 seconds until blastoff
6 seconds until blastoff
5 seconds until blastoff
4 seconds until blastoff
3 seconds until blastoff
2 seconds until blastoff
1 seconds until blastoff
0 seconds until blastoff
BLASTOFF
```

2.12 Step 12

Execute the `nano` command to create a file called `whiletest.sh`:

```
nano whiletest.sh
```

```
  GNU nano 2.3.1               File: whiletest.sh
```

```
                                [ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.13 Step 13

The `while` statement will execute a `do/done` block similar to a `for` statement, but only if the condition that follows the `while` has a value that is `true`. Often the condition that follows the `while` statement will be based upon the result of a **`test`** statement. In the next example, the `true` statement will be used as the condition; because the value for the `true` statement is always true, the example that follows could go on forever.

The **`test`** statement can also be written by enclosing the condition to test within square brackets `[ ]`, as the example below shows, where the answer variable is tested within the `if` statement to see if it is equal to `y`. Loops made with a `while` statement can be “exited early”, before the `while` condition changes to false.

In the example that follows, the `if` statement checks whether the test of `$answer" == "y"` has a true result. The **`if`** statement is terminated by the `fi` statement. If the user types `y` when prompted to by the `read` statement, then the test will have a true result, and the `break` statement will be executed, thus exiting the script without testing the `while` conditional statement.

Type the text of the script:

```
#!/bin/bash
while true
do
  read -p "Do you want to quit (y/n)? " answer
  if [ "$answer" == "y" ]
  then
    break
  fi
done
```

```
  GNU nano 2.3.1             File: whiletest.sh                        Modified

#!/bin/bash
while true
do
  read -p "Do you want to quit (y/n)? " answer
  if [ "$answer" == "y" ]
  then
    break
  fi
done
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.14 Step 14

Exit the `nano` editor by pressing **Ctrl+X**. Type `y` to save the changes, and press the **Enter** key:

```
Ctrl+X
y
Enter
```

2.15 Step 15

Make the `whiletest.sh` script executable with `chmod` and run it by typing its name, `whiletest.sh`. Answer `n` the first two times when prompted, and then answer `y`:

```
chmod a+x whiletest.sh
whiletest.sh
n
n
y
```

```
[clima@centos bin]$ chmod a+x whiletest.sh
[clima@centos bin]$ whiletest.sh
Do you want to quit (y/n)? n
Do you want to quit (y/n)? n
Do you want to quit (y/n)? y
```

2.16 Step 16

Modify the `whiletest.sh` file to include an `else` within the `if` statement and nest a `for` loop within the `while` loop. Execute the `nano` command again:

```
nano whiletest.sh
```

```
  GNU nano 2.3.1             File: whiletest.sh

#!/bin/bash
while true
do
  read -p "Do you want to quit (y/n)? " answer
  if [ "$answer" == "y" ]
  then
    break
  fi
done

                              [ Read 9 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.17 Step 17

Insert the following lines between the `break` and `fi` statements:

```
else
    for num in $(seq 5 -1 1)
    do
        echo “$num seconds until prompted again”
        sleep 1
    done
```

```
  GNU nano 2.3.1             File: whiletest.sh                         Modified

#!/bin/bash
while true
do
  read -p "Do you want to quit (y/n)? " answer
  if [ "$answer" == "y" ]
  then
```

```
    break
  else
    for num in $(seq 5 -1 1)
    do
        echo "$num seconds until prompted again"
        sleep 1
    done
  fi
done
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

2.18 Step 18

Exit the `nano` editor by pressing **Ctrl+X**. Type `y` to save the changes, and press the **Enter** key:

```
Ctrl+X
y
Enter
```

2.19 Step 19

Run the script again to see the effect of the additional loop and `else` statement by typing its name, `whiletest.sh`. Answer `n` the first time when prompted and wait for the prompt to reappear. Notice that the sequence numbering was modified from earlier to only loop 5 times. Once the prompt displays, answer `y`:

```
whiletest.sh
n
y
```

```
[clima@centos bin]$ whiletest.sh
Do you want to quit (y/n)? n
5 seconds until prompted again
4 seconds until prompted again
3 seconds until prompted again
```

```
2 seconds until prompted again
1 seconds until prompted again
Do you want to quit (y/n)? y
```

This lab is complete.

Theory (cron, at, batch)

3.1 Introduction

Scheduling tasks or executing commands at a particular time is inherently part of Linux administration. Some of the tasks (commands or shell scripts) which generally require scheduling are:

- Generating thumbnails for websites
- Computing intermediate data for processing
- Performing backups
- Synchronizing files
- Scheduling operating system and application updates
- Sending newsletter emails
- Automatically checking websites for broken links
- Housekeeping activities such as cleaning up directories that store log files

There are a wide variety of commands that can be executed at a future time. However, keep in mind that the commands should be non-interactive since the assumption is that there is no user to interact with when the commands execute. Generally speaking, the four tools described in this chapter (`cron`, `at`, `batch`, and systemd timers) will be extremely useful for scheduling tasks for execution at some later time.

3.2 Understanding cron

The `cron` command derives its name from chronos, the Greek word for time. `cron` is a utility that provides tools, including:

- A time-based job scheduler that runs as a daemon named `crond`. This daemon is used to execute repetitive tasks at a predetermined interval.
- A `crontab` (cron table) file that stores scheduled tasks (commonly referred to as jobs or cron jobs) to execute and is read by the `crond` daemon. Each user is permitted to have a personal `crontab` file by default, but the administrator can disable this feature for security purposes.
- The `crontab` command used to view and edit a user's `crontab` file (which is initially empty).

3.3 System crontab and User crontab

The system `crontab` allows scheduling of system-wide tasks such as log rotations, backups, and system database updates. Modifying the system `crontab` for scheduling tasks requires administrative privileges. Additional system `crontab` jobs that require finer control of their scheduling (verses daily, weekly, monthly tasks in `/etc/crontab`) and do not need environmental variables to be loaded before script execution are stored in the `/etc/cron.d` directory.

The system `crontab` can execute commands either as root or as any other user. For example, if a user does not have access to `cron`, but needs to have a particular script run at the end of the day on a daily basis, then they can ask the administrator to add an entry in the system `crontab`. Administrative tasks, such as backup of system files on a periodic basis, should be performed as the root user.

The following is an example of a typical system `crontab` file:

```
# /etc/crontab - root's crontab

#

#2014-03-20 17:14:40
```

```
#
#
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
#
#minute hour mday month wday who command
#
0 23 * * * root /usr/root/bkup.sh
*/30 * * * * dev /usr/dev/ant_build.sh
```

The first ten lines show comments (those beginning with the hash `#` character) and set variables (`SHELL`, `PATH`) that define the environment for the scheduled jobs when they run. The last two lines describe what command or script is executed. Each of these lines describes one command to execute. Based on the two entries in the previous example, the `/usr/root/bkup.sh` script will be executed as the `root` user and the `/usr/dev/ant_build.sh` script will be executed as the `dev` user.

In addition to the system `crontab` maintained by root, each user can have a personal `crontab` file unless prevented by the administrator for security reasons. Like the system `crontab` file, these commands will run regardless of whether the user is logged into the system.

The following is an example of a basic user `crontab` file:

```
#minute hour mday month wday who command


#
0 17 * * 5 /usr/bin/banner  "The Weekend Is Here!" > /dev/console
```

In the preceding example, the message `The Weekend Is Here!` in big letters would be displayed in an open terminal window (`/dev/console` - see `who` command output) at 5:00 pm (hour `17`) every Friday (day `5`) of the week of every month.

The user `crontab` is similar to the system `crontab` except for the 6th field, which contains the user name. This field is not included in the user `crontab` since all the entries are for that particular user only. The fields in a `crontab` entry will be discussed in detail in the next section.

3.4 Understanding crontab Entries

The `crontab` entries can be used to schedule tasks that can occur as frequently as once every minute to as infrequently as once a year. The format of a `crontab` entry is:

```
Minute Hour Day-of-month Month Day-of-week command
```

The following table provides additional details regarding the values that can be placed in each of these fields:

Field	Values
Minute	Range `0 – 59`
Hour	Range `0 – 23`
Day-of-month	Range `1 – 31`

Field	Values
Month	Range `1` – `12` (Jan-Dec)
Day-of-week	Range `0` – `7` (`0` & `7` = Sunday)
Command	The command to be executed

The first five fields can contain the following, to represent time values:

- A single value
- Multiple values such as 1,3,5 in the fifth field, which would mean Monday, Wednesday, Friday
- A range of values (such as 1 through 5 (1-5)) in the fifth field, which would mean Monday through Friday
- An asterisk `*` character means any or all values

Example crontab Entries

Example 1:

```
30 04 1 1 1 /usr/bin/some_command
```

The above entry will run `/usr/bin/some_command` at 4:30 am on January 1st, plus every Monday in January.

Example 2:

```
30 04 * * * /usr/bin/some_command
```

The above entry will run `/usr/bin/some_command` at 4:30 am on every day of every month.

Example 3:

```
01,31 04,05 1-15 1,6 * /usr/bin/some_command
```

Comma-separated values can be used to run more than one instance of a particular command within a time period. Dash-separated values can be used to run a command continuously. The above entry will `run /usr/bin/some_command` at 01 and 31 past the hours of 4:00 am and 5:00 am on the 1st through the 15th of every January and June.

Example 4:

```
00 08-17 * * 1-5 /usr/bin/some_command
```

The above entry will run `/usr/bin/some_command` every hour (on the hour) from 8AM to 5PM, Monday through Friday of every month.

Variables in crontabs

The crontab file can also contain variables; some of the most commonly found variables are:

```
PATH=/usr/local/bin:/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin
SHELL=/bin/bash
MAILTO=databaseadmin
```

The `PATH` variable is very important in the event that relative pathnames are used when specifying commands. For example, to execute the `date` command without including the `PATH` variable, the full path to the `date` command `/bin/date` must be specified.

It is important to note that the `crond` utility does not read user initialization files when it executes commands. This means that creating variables like the `PATH` variable is very important.

The `MAILTO` variable is important for any command that produces output when executed. There is no assumption that the user is logged in when the command runs, so the output of the command is sent to the user's email address by default. To specify a different email address, use the `MAILTO` variable.

Crontab Keywords

As defined in the `crontab` man page, the following special keywords can also be used to specify a time in place of the normal five fields representing time values:

Entry	Description	Equivalent to
`@reboot`	Run once, at startup	
`@yearly` `@annually`	Run once a year	`0 0 1 1 *`
`@monthly`	Run once a month	`0 0 1 * *`
`@weekly`	Run once a week	`0 0 * * 0`
`@daily` `@midnight`	Run once a day	`0 0 * * *`
`@hourly`	Run once an hour	`0 * * * *`

For example, the `crontab` entry to execute the `/home/sysadmin/bin/daily-backup` script at 00:00 (midnight) every day could be entered in one of three ways:

```
0 0 1 * *  /home/sysadmin/bin/daily-backup
```

OR

```
@daily  /home/sysadmin/bin/daily-backup
```

OR

```
@midnight  /home/sysadmin/bin/daily-backup
```

3.5 Maintaining User crontab Files

The `crontab` command is used for maintaining user `crontab` files. The user `crontab` files are stored in the `/var/spool/cron` directory, but should not be edited directly. In fact, regular users don't have any access to this directory, so a user must use the `crontab` command to display or edit his own `crontab` file. This also means that a regular user can't display `crontab` files of other users.

To edit the `crontab` file, a user can type the following command:

```
clima@centos:~$ crontab -e
no crontab for sysadmin - using an empty one
```

The `crontab` command will start a text editor program (`vi` by default) to allow for the modification of that user's `crontab` file.

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
~
"/tmp/crontab.Y9TMSH/crontab" 22L, 888C
```

To quit the `crontab` file without saving, use the `:q` command.

To specify a different editor, set the `EDITOR` variable before executing the `crontab -e` command. For example, to use the `gedit` editor (a graphical editor available on GNOME desktops), use the `export EDITOR gedit` command.

On exit from the editor, the file will be saved, and the changes will be stored in that user's `crontab` file; additionally, the `crond` daemon will automatically update the `crontab` file that it stores in memory, so the changes take effect immediately.

One of the useful features of the `crontab` command is that it performs some basic error checking. For example, if an invalid time field is specified, the `crontab` command will issue a warning and provide the opportunity to fix the error:

```
clima@centos:~$ crontab -e
no crontab for sysadmin - using an empty one
crontab: installing new crontab
"/tmp/crontab.Dwt5Vg/crontab":22: bad month
errors in crontab file, can't install.
Do you want to retry the same edit? (y/n)
```

When prompted `Do you want to retry the same edit?`, type **y** for yes and the `crontab` command will return to the editor so the error can be fixed. Answer **n** for no, and the `crontab` will quit and discard all changes that were made.

Each user can view the current contents of their `crontab` using the `crontab -l` command. Each user can also remove all of their current crontab entries, using the `crontab -r` command. An administrator logged in as the root user can view or modify another user's `crontab` file by specifying a user name with the `-u` option: `crontab -u user_name`.

The `crond` daemon looks for user `crontab` files in the specified spool area; typically, this is either the `/var/spool/cron` directory, but it may be the `/var/spool/cron/crontabs` directory on some distributions. The `crontab` files found in these locations are loaded into memory for use by the `crond` daemon.

3.6 Maintaining System crontab Files

The `crond` daemon reads the system `crontab` from the `/etc/crontab` file. The contents of this file will vary, depending on the distribution being used. For example, on Red Hat Enterprise Linux this file typically looks like the following:

```
[clima@srvUbuntu sysadmin]# cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# For details see man 4 crontabs



# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

It is important to note that on modern releases of Red Hat Enterprise Linux, the `/etc/crontab` file may contain no entries by default. On previous releases (and on some current distributions of Linux) there may be entries in this file that direct the `crond` daemon to execute commands in the following directories:

- `/etc/cron.hourly`
- `/etc/cron.daily`
- `/etc/cron.weekly`

Commands and executables in these directories are executed on an hourly, daily, or weekly basis.

The `crond` daemon also reads the files in the `/etc/cron.d` directory. This location is used by various packages to exercise fine control when scheduling system tasks. A software application can create

a `crontab` file and place it in the `/etc/cron.d` directory during installation (or removed from the directory when the package is uninstalled from the system).

For example, on an Ubuntu Linux system, the `/etc/cron.daily` directory may contain files as shown in the following screen:

```
clima@centos:~$ ls /etc/cron.*
/etc/cron.d:

/etc/cron.daily:

apt       bsdmainutils  logrotate  mlocate  quota
aptitude  dpkg          man-db     passwd   upstart

/etc/cron.hourly:

/etc/cron.monthly:

/etc/cron.weekly:
apt-xapian-index  fstrim  man-db
```

Don't be surprised if your system has different files in the various `/etc/cron.*` directories than the example that has been provided. As different software is installed (or removed) from the system, the contents of this directory may change.

The `crond` daemon "wakes up" every minute, examines all of the `crontab` files that are stored in memory, and determines if there is a particular command or script requiring execution. If such a command is found, the `crond` daemon launches that process.

The `crond` daemon keeps track of changes in `crontabs` by checking the modification time of files. It will poll for changes in the modification time of all `crontabs` every minute and reload the changed files, if any. There is no need to restart the `crond` daemon if any of the `crontabs` have been updated.

The `crond` daemon can be started, stopped, and restarted by the root user using the following commands:

```
/etc/init.d/crond start
/etc/init.d/crond stop
/etc/init.d/crond restart
```

Note

Users who remotely connect to a server may be working in different time zones. For example, a user in London who logs into a server in New York would discover that the time on the system is different from the user's local time.

As discussed at the end of this module, the `TZ` environment variable can be used to display the local time for a specific user. However, this may result in problems when that user creates `crontab` files.

The `crond` daemon always runs in a specified time zone only. So even if there are multiple users of the system who are working in different time zones, all of the jobs will run on the time zone configured on the server, irrespective of the time zones specified by users in their `crontab` files.

If the user in London has scheduled a job on the New York server (running on EST) and he wants the job to be run at midnight London time, then the user will have to make the `crontab` entry considering the adjustments with the server time zone. Setting the `TZ` environment variable in the `crontab` file does not affect when the command is executed.

3.7 Controlling Access to cron

The root user always has the ability to execute the `crontab` command, but this access can be restricted for other users. The two files used for this purpose are the `/etc/cron.allow` file and the `/etc/cron.deny` file. These files are created and maintained by the root user. They should contain a list of users, with one user specified per line. The following rules are used to determine how these files are used:

1. If both the `cron.allow` and the `cron.deny` files do not exist, the default on current Ubuntu and other Debian-based systems is to *allow all users* to use the `crontab` command.
2. If only the `cron.allow` file exists, then *only the users listed in the file can execute* the `crontab` command.
3. If only the `cron.deny` file exists, then *all users listed in this file are denied the ability to execute* the `crontab` command, and *all other users are allowed* to use the `crontab` command.
4. If both the `cron.allow` and the `cron.deny` files exist, the `cron.allow` file applies and the `cron.deny` file is ignored. As only one of these files should exist, the presence of both files is typically due to a mistake made by the administrator.

To help you understand how to determine which of these files should be created, consider the following three scenarios:

Scenario #1: 100 users, only five should be able to use the `crontab` command.

In this scenario, create the `/etc/cron.allow` file and place the five users who are allowed to use the `crontab` command in this file.

Do not create a `/etc/cron.deny` file.

Scenario #2: 100 users, only five should be denied from using the `crontab` command.

In this scenario, create the `/etc/cron.deny` file and place the five users who are denied access to the `crontab` command in this file.

Do not create a `/etc/cron.allow` file.

Scenario #3: 100 users, none should be able to use the `crontab` command.

In this scenario, do not create a `/etc/cron.deny`, but do create an empty `/etc/cron.allow` file to specify that no users have permission to access to `crontab`.

While this may vary depending on the distribution of Linux, typically only the `/etc/cron.deny` file exists by default, and it is normally empty by default. This allows all users the ability to use the `crontab` command.

If a user account is not able to use the `crontab` command, then the user will receive an error message like the following:

```
clima@centos:~$ crontab -e
You (sysadmin) are not allowed to use this program (crontab)
See crontab(1) for more information
clima@centos:~$ crontab -l
You (sysadmin) are not allowed to use this program (crontab)
See crontab(1) for more information
```

Note that if a user has been denied access to the `crontab` command, this does not prevent any previously created `crontab` entries from running. If the user already had `crontab` entries, the administrator needs to remove them, which can be done by using the `crontab -u sysadmin -r` command.

Consider This

If the system is using PAM based authentication, then there is an alternative way of configuring access to the `crontab` command. While this alternative method is beyond the scope of this course, it is mentioned here in the event that a student is working on a system that has a customized PAM configuration.

3.8 at Command

The `cron` command is a good tool for scheduling tasks that are required to run at regular intervals. For scheduling one-time tasks, the `at` and `batch` commands are more useful.

There are two prerequisites for running the `at` command: the `at` package must be installed, and `atd` service must be running.

If the `at` package is not installed, use the appropriate command for your system to install the package.

To determine if the `atd` service is running, use the following command:

The examples in this section are for demonstration purposes. The output in the examples may not match the output in our virtual environment.

```
clima@centos:~$ service atd status
atd  stop/waiting
```

If it is not running, execute the following command as root:

```
clima@srvUbuntu:~# service atd start
atd  start/running,  process   540
```

Command Usage

The `at` command executes commands at a specified time or at a time relative to the current time. The command can be used as follows:

```
at TIME
```

The `TIME` specifications supported by `at` are quite extensive and easy to use. Some of the supported formats are mentioned in the table below, assume the current date/time is 1st Mar 2025, 8:00 a.m:

Keyword/Date Format	Significance
`midnight`	12:00 a.m. 2nd Mar 2025
`noon`	12:00 p.m .1st Mar 2025
`tomorrow`	8:00 a.m. 2nd Mar 2025
`next week`	8:00 a.m. 8th Mar 2025
`1630`	4:30 p.m. 1st Mar 2025
`4:30 PM Mar 20`	4:30 p.m. 20th Mar 2025
`now + 2 hours`	10:00 a.m .1st Mar 2025
`now + 7 days`	8:00 a.m .8th Mar 2025

The exact `TIME` specifications can be found in the `at` command man page or the `timespec` help document typically found in the `/usr/share/doc/at*` directory. Consult the man page for the `at` command to determine the exact location of the help file.

When the `at` command is executed, the user is presented with the `at>` prompt.

```
clima@centos:~$ at now
at>
```

Enter the command that requires execution and then press the **Enter** key. Another `at>` prompt will be presented for another command.

When you are finished entering commands to execute, use **Ctrl+d**. The End Of Transmission (`EOT`) will be displayed, followed by the assigned job number, and the time the job will be executed. To cancel the `at` job, press **Ctrl+c** at any time (before scheduling the job with **Ctrl+d**). In the example below, `Planning meeting in 30-minutes` will display in the current terminal window (`/dev/pts/0` - see `who` command output) at 10:00 am.

Consider This

Since Linux is a multi-user system, there is a way to address each session that is logged into the system. Each user login session can be identified with the `who` command. The command below uses the address (`/dev/pts/0`) listed in the `who` command output, for the desired user to send that session an alert at the specified time.

```
clima@centos:~$ at 1000
warning:  commands will be executed using /bin/sh
at> echo "Planning meeting in 30-minutes" > /dev/pts/0
at>  <EOT>
job 2 at Mon Feb  10  09:26:00  2025
```

Note

The `at` and `batch` jobs are stored in the `/var/spool/at` directory.

Additional Examples

The following command will read the command entered at the `at>` prompt and execute it 2 hours from now:

```
clima@centos:~$ at now + 2 hours
warning:  commands will be executed using /bin/sh
at> echo "Marketing meeting in 30-minutes" > /dev/pts/0
at>  <EOT>
job 3 at Mon Feb  10  09:28:00  2025
```

The following command will read the commands from the `backup_script.txt` file (rather than prompting the user to enter commands at the `at>` prompt) at 10:00 pm on March 22nd:

```
clima@centos:~$ at -f backup_script.txt  10 pm  Mar 22
warning:  commands will be executed using /bin/sh
job 4 at Mon Feb  10  09:31:00  2025
```

The following command is the same as above, with the addition of the `-m` option which sends a mail message to the user after the job is complete:

```
clima@centos:~$ at -m -f backup_script.txt  10 pm  Mar 22
```

```
warning:  commands will be executed using /bin/sh
job 5 at Mon Feb  10  09:33:00  2025
```

Listing and Deleting at Jobs

The *at queue* `atq` command lists the user's pending jobs. If the command is being run as the root user, then it will list jobs for all the users.

The output of the `atq` command will be similar to the following:

```
clima@centos:~$ atq
2             Mon Feb 10  09:26:00  2025  a   sysadmin
3             Mon Feb 10  09:28:00  2025  a   sysadmin
4             Mon Feb 10  09:31:00  2025  a   sysadmin
5             Mon Feb 10  09:33:00  2025  a   sysadmin
```

The output includes the following fields:

Field	Example	Significance
Number	`2`	Job number allocated to this job
Date	`Mon Feb 10`	Date when this job will be executed
Hour	`09:26:00`	Hour when this job will be executed
Queue	`a`	Name of the queue. Valid queue names are from a-z, with 'a' being the default queue.
User	`sysadmin`	User name of the user who has scheduled this job

The *at remove* `atrm` command deletes `at` jobs, identified by their job number. For example, to delete job number 2, use the following command:

```
clima@centos:~$ atrm 2
clima@centos:~$ atq
3             Mon Feb 10  09:28:00  2025  a   sysadmin
4             Mon Feb 10  09:31:00  2025  a   sysadmin
5             Mon Feb 10  09:33:00  2025  a   sysadmin
```

Important

To delete a job, you must either be the owner of the job or the root user.

Configuring Access to the at Command

While the root user can always use the `at` and `batch` commands, users may not be able to use these commands.

Note

The `batch` command is covered in greater detail in a later section.

Access to the `at` and `batch` commands is controlled by the following two files:

1. The `/etc/at.allow` file
2. The `/etc/at.deny` file

The format of these files is similar to `cron.allow` and `cron.deny` files. Both the files contain a list of user names, one on each line. The rules mentioned for `cron` access are applicable for `at` access also:

1. If both files do not exist, then *all regular users are denied the ability to execute* the `at` and `batch` commands.
2. If only the `at.allow` file exists, then *only the users listed in the file can execute* the `at` and `batch` commands.
3. If only the `at.deny` file exists, then *all users listed in this file are denied the ability* to execute the `at` and `batch` commands, and *all other users are allowed to execute* the `at` and `batch` commands.
4. If both the `at.allow` and the `at.deny` files exist, the `at.allow` file applies and the `at.deny` file is ignored. As only one of these files should exist, the presence of both files is typically due to a mistake made by the administrator.

To help you understand how to determine which of these files should be created, consider the following three scenarios:

Scenario #1: 100 users, only five should be able to use the `at` and `batch` commands.

In this scenario, create the `/etc/at.allow` file and place the five users who are allowed to use the `at` and `batch` commands in this file.

Do not create an `/etc/at.deny` file.

Scenario #2: 100 users, only five should be denied from using `at` and `batch` commands.

In this scenario, create the `/etc/at.deny` file and place the five users who are denied from using the `at` and `batch` commands in this file.

Do not create a `/etc/at.allow` file.

Scenario #3: 100 users, none should be able to use the `at` and `batch` commands.

In this scenario, do not create a `/etc/at.deny` file, but do create an empty `/etc/at.allow` to specify that no normal users are allowed to run the `at` and `batch` commands.

While this may vary depending on the distribution of Linux you are working on, typically only the `/etc/at.deny` file exists by default, and it is normally empty by default. This allows all users the ability to use the `at` and `batch` commands.

If a user account is not able to use the `at` and `batch` commands, then the user will receive an error message like the following:

```
clima@centos:~$ at 1000
You do not have permission to use at.
```

Note that if a user has been denied access to the `at` and `batch` commands, this does not prevent any previously created `at` or `batch` entries from running. If the user already had `at` or `batch` entries, the administrator needs to remove those entries.

3.9 batch Command

Similar to the `at` command, the `batch` command is used to schedule one-time tasks. However, instead of specifying an execution time, `batch` commands are executed as soon as the system's load average drops

below `0.8` (80% of CPU usage). The default value of `0.8` can be changed by using the `-l` option to the `atd` command.

The `batch` command will prompt for command input. A job number is issued upon the successful completion of input. Alternatively, it can read input from a file by using the `-f` option. For example, to sort the large `marketing_data` file at the point when the system load average drops below `0.8`, execute the following:

```
clima@centos:~$ batch
at> sort  ~/marketing_data
at> <EOT>
job 5 at Mon Feb  3  09:26:00  2025
```

3.10 Systemd Timer Units

Many modern systems use systemd rather than the much older system daemon `init` for managing system services. Systemd is a service and system manager originally designed by Red Hat. Systems that have `systemd` as a replacement for the traditional `init` process provide an alternative to `crond` for scheduling jobs and managing services, called *systemd timer*.

Cron's creation dates back to 1975, which makes it much older than systemd timers. While `cron` is more widely available across systems, including non-Linux systems (such as BSD systems), systemd timers have some advantages from what has been learned from years and years of usage of cron.

For example, when systemd timers are created, they can be set to run in a specific environment (`systemd.exec`), whereas `cron` inherits the environment it is being run from. Because (`cron` is not run from a shell that loads the environment through startup files like `.bashrc`, errors may occur since variables and file paths may differ from the shell environment that is being used to create the job (i.e., a user's shell). Another advantage is that with systemd timers, dependencies can be configured for jobs, meaning that scheduled jobs can be set to be dependent on other systemd units. Finally, systemd timer jobs are logged via the systemd journal, providing output in a centralized place and format.

What is a timer? Timers are essentially files that end in `.timer`, which fall under a category of files called systemd *unit* files. These unit files contain information about services, sockets, mount points, timers, devices, and other types of systemd units. The `.timer` files will contain configuration information about the task to be executed by the systemd timer. Below is an example of a timer file created to display a greeting after the system startup:

```
[Unit]
Description=Displays greeting after boot


[Timer]
OnBootSec=10sec
Unit=greeting.service


[Install]
WantedBy=multi-user.target
```

Below is a breakdown of the file sections:

Section	Description
`[Unit]`	General information about the systemd unit file. The `Description` option creates a human-readable name for the systemd unit.

Section	Description
`[Timer]`	Contains the options that define when the timer will start and what service to execute.
`[Install]`	Contains information about how the unit will be installed. The `WantedBy=` option creates a symbolic link within the target level that "wants" to have that service running. Systemd targets are beyond the scope of this course.

Systemd uses two different types of timers; *monotonic* and *realtime*. With monotonic timers, the systemd timer allows a job to be executed after an event has occurred. This type can be used to run a job when the system boots (`OnBootSec` option) or a systemd unit is active (`OnActiveSec` option). The example `greeting.timer` file above is a monotonic timer that uses the `OnBootSec` option to run the timer ten seconds after the system boots:

```
OnBootSec=10sec
```

Realtime timers work like `cron` and execute a job when a specified time has occurred. To create a realtime timer, the `OnCalendar` option should be used in the `[Timer]` section of the `.timer` file. The format of an `OnCalendar` time entry is:

```
DayofWeek Year-Month-Day Hour:Minute:Second
```

The entry below is an example of a timer unit that is executed every day at 9:00 am:

```
OnCalendar=*-*-* 9:00:00
```

Similar to the `crontab` format, the asterisk `*` character in the example above means *any or all values*.

To specify what days of the week to run a timer, use the `DayofWeek` field. The entry below is an example of a timer unit that is executed every Monday, Wednesday, and Friday at midnight:

```
OnCalendar=Mon,Wed,Fri *-*-* 00:00:00
```

To run a timer on a specific day of the month, use the `Day` field. The following entry will run a timer on the first day of every month at 10 PM:

```
OnCalendar=*-*-01 22:00:00
```

The `OnCalendar` option also uses special expressions such as `hourly`, `daily`, `monthly`, and `yearly` as time values:

```
OnCalendar=daily
```

A `.timer` unit file should correspond to a systemd service, which is a `.service` file with the same name. For example, the `greeting.timer` unit file created above should have a corresponding service file called `greeting.service`. When the systemd timer is activated, the `.service` systemd unit is executed.

3.11 systemctl Command

To better manage systemd timers, as well as other services with systemd, some knowledge of the `systemctl` command is required. The `systemctl` command is used to interface with and control systemd units. The syntax to use the `systemctl` command is the following:

```
systemctl COMMAND UNIT_NAME [OPTION]
```

OR

```
systemctl LIST_UNIT
```

To demonstrate, let's start the `sshd` service using the `systemctl start` command as root:

The output in the examples below may not match the output in our virtual environment.

```
clima@srvUbuntu:~# systemctl start sshd
```

Another common `systemctl` command is `status`, which is used to check if a service is running. Here is an example of checking on the status of the sshd with the following:

```
clima@srvUbuntu:~# systemctl status sshd
```

To list all the services that are being run on the system, the following command can be used:

```
clima@srvUbuntu:~# systemctl list-units --type=service
UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
accounts-daemon.service       loaded active running Accounts Service
apparmor.service              loaded active exited  AppArmor initialization
blk-availability.service      loaded active exited  Availability of block devices
console-setup.service         loaded active exited  Set console font and keymap
cron.service                  loaded active running Regular background program processing
daemon
dbus.service                  loaded active running D-bus System Message Bus
getty@tty1.service            loaded active running Getty on tty1
grub-common.service           loaded active exited  LSB: Record successful boot for GRUB
keyboard-setup.service        loaded active exited  Set the console keyboard layout
kmod-static-nodes.service     loaded active exited  Create list of required static device
nodes for the current kernel
lvm2-lvmetad.service          loaded active running LVM2 metadata daemon
lvm2-monitor.service          loaded active exited  Monitoring of LVM2 PV scan on device
8:1
networkd-dispatcher.service loaded active running Dispatcher daemon for systemd-neworkd
Output Omitted...
```

To show what timers are active on you system, simply run:

```
clima@srvUbuntu:~# systemctl list-timers
NEXT                         LEFT     LAST                         PASSED          UNIT
Sun 2019-10-20 06:49:59 UTC  7h left  Sat  2019-10-19 22:07:15 UTC 1h 16min ago apt-dai
ly-upgrade.timer
Sun 2019-10-20 08:03:49 UTC  8h left  Sat  2019-10-19 22:07:15 UTC 1h 16min ago apt-dai
ly.timer
Sun 2019-10-20 11:18:19 UTC  11h left Sat  2019-10-19 22:32:58 UTC 51min ago motd-news.
timer
Sun 2019-10-20 22:22:58 UTC  22h left Sat  2019-10-19 22:22:58 UTC 1h 1min ago systemd-
tmpfile-clear.timer
```

```
Mon 2019-10-21 00:00:00 UTC  24h left Sat  2019-10-19 22:07:15 UTC 1h 16min ago fstrim.
timer


5 timers listed.

Pass --all to see loaded but inactive timers, too.

lines 1-9/9 (END)
```

To show all of the systemd timer units on a system, including the inactivate systemd timer units, execute the following:

```
clima@srvUbuntu:~# systemctl list-timers --all

NEXT                         LEFT     LAST                         PASSED          UNIT

Sun 2019-10-20 06:49:59 UTC  7h left  Sat  2019-10-19 22:07:15 UTC 1h 16min ago apt-dai
ly-upgrade.timer

Sun 2019-10-20 08:03:49 UTC  8h left  Sat  2019-10-19 22:07:15 UTC 1h 16min ago apt-dai
ly.timer

Sun 2019-10-20 11:18:19 UTC  11h left Sat  2019-10-19 22:32:58 UTC 51min ago motd-news.
timer

Sun 2019-10-20 22:22:58 UTC  22h left Sat  2019-10-19 22:22:58 UTC 1h 1min ago systemd-
tmpfile-clear.timer

Mon 2019-10-21 00:00:00 UTC  24h left Sat  2019-10-19 22:07:15 UTC 1h 16min ago fstrim.
timer

n/a                          n/a      n/a   n/a                          unreadahead-stop.time
r


6 timers listed.

lines 1-9/9 (END)
```

The systemd timers can be enabled and disabled like any other systemd unit using the following command:

```
systemctl enable name.timer

systemctl disable name.timer
```

Outside of the `.timer` files, the `systemd-run` command can be used to run a *transient* job, one which does not have a `.timer` file. The `systemd-run` command can be used to run a command or execute a systemd `.service` unit. For example, to execute the `touch /home/sysadmin/newfile` command one hour after running the `systemd-run` command, use:

```
clima@srvUbuntu:~# systemd-run --on-active="1h" /bin/touch /home/sysadmin/newfile
```

This will run the `touch` command and create the `newfile` file in the `/home/sysadmin` directory one hour after running the `systemd-run` command. To execute a systemd `.service` unit, use the `--unit` option:

```
clima@srvUbuntu:~# systemd-run --on-active="1h" --unit=greeting.service
```

The command above will execute the `greeting.service` unit one hour after running the `systemd-run` command.

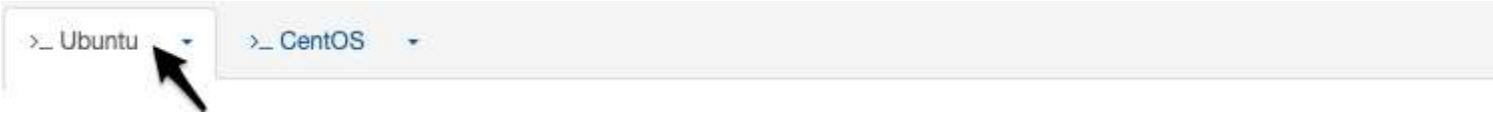
After creating a systemd timer via `systemd-run`, the name of the transient systemd job is returned to you. Executing the `systemctl list-timers` command will display the name of the systemd transient job listed in the `UNIT` column.

Lab (cron, at, batch)

Step 1

The Ubuntu VM will be used for this portion of the lab.

Click on the **Ubuntu** virtual machine tab on the right.



When prompted for `ubuntu login:`, enter `sysadmin`. When prompted for a password, enter `netlab123`:

```
Ubuntu 18.04.3 LTS ubuntu tty1

ubuntu login: sysadmin
Password:

* Documentation: https://help.ubuntu.com
* Management:    https://landscape.connonical.com
* Support:       https://ubuntu.com/advantage

sysadmin@ubuntu:~$
```

Step 2

Execute the following command to start the process of creating a crontab entry for the `sysadmin` account:

```
crontab -e
2
```

```
sysadmin@ubuntu:~$ crontab -e
no crontab for sysadmin - using an empty one

Select an editor. To change later, run 'select-editor'.
   1. /bin/nano      <---- easiest
   2. /usr/bin/vim.tiny
   3. /bin/ed
Choose 1-3 [1]: 2
```

After you hit the **Enter** key to execute the command, your screen should look like the following:

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
~
~
~
~
~
~
~
~
"/tmp/crontab.C1lFsW/crontab" 22L, 888C
```

The `-e` option allows you to edit your crontab file. By default this places you in the `vi` editor.

Step 3

Enter the following keystrokes to create a crontab entry:

```
G
o
```

```
* * * * * echo "it works" >> /tmp/cronoutput
```

```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
* * * * * echo "it works" >> /tmp/cronoutput
~
~
~
~
~
~
~
~
"/tmp/crontab.YYSUXO/crontab" 23L, 933C
```

Esc

:wq

Enter

After pressing the **Enter** key, the following message should appear:

```
crontab: installing new crontab
sysadmin@ubuntu:~$
```

Step 4

Verify your work by executing the following command:

```
crontab -l
```

The output should look like the following:

```
sysadmin@ubuntu:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
```

```
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
* * * * * echo "it works" >> /tmp/cronoutput
```

Step 5

This crontab entry should run the `echo` command **once per minute** and send output to the `/tmp/cronoutput` file. To verify the entry is working correctly, wait one minute and execute the following command:

```
more /tmp/cronoutput
```

You should have at least one line of output like the following:

```
sysadmin@ubuntu:~$ more /tmp/cronoutput
it works
```

Step 6

Remove the `sysadmin` crontab entry and verify by executing the following commands:

```
crontab -r
crontab -l
```

```
sysadmin@ubuntu:~$ crontab -r
sysadmin@ubuntu:~$ crontab -l
no crontab for sysadmin
```

Step 7

To view and modify *system* crontab files, first execute the following command to switch to the `root` account. When prompted, provide the `root` password `netlab123`:

```
su -
```

```
sysadmin@ubuntu:~$ su -
Password:
root@ubuntu:~#
```

Step 8

View the `/etc/crontab` file and notice that it already has some entries:

```
more /etc/crontab
```

```
root@ubuntu:~# more /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd /  && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/
cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/
cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/
cron.monthly )
#
```

Step 9

In addition to the entries in the `/etc/crontab` file, entries from files in the `/etc/cron.d` directory are used. View the contents of the `/etc/cron.d` directory to determine if there are any system crontab entries:

```
ls /etc/cron.d
```

```
root@ubuntu:~# ls /etc/cron.d
popularity-contest
```

Typically these files are placed when some software packages are installed.

Step 10

Create a system crontab entry in the `/etc/crond.d` directory by executing the first command below:

```
echo "* * * * * root who >> /tmp/whothere" > /etc/cron.d/who
ls /etc/cron.d
more /etc/cron.d/who
```

```
root@ubuntu:~# echo "* * * * * root who >> /tmp/whothere" > /etc/cron.d/who
root@ubuntu:~# ls /etc/cron.d
popularity-contest who
root@ubuntu:~# more /etc/cron.d/who
* * * * * root who >> /tmp/whothere
```

This new system crontab file should take the output of the `who` command (which displays who is logged into the system) and sends it to the `/tmp/whothere` file. This crontab entry runs every minute, so **if you wait a couple of minutes**, you can verify this crontab entry is working by executing the following command:

```
more /tmp/whothere
```

```
root@ubuntu:~# more /tmp/whothere
sysadmin console      Dec 11 17:33
```

Step 11

Execute the following commands to view the `/etc/cron.allow` and `/etc/cron.deny` files to verify that all users can use the `crontab` command:

```
more /etc/cron.deny
more /etc/cron.allow
```

```
root@ubuntu:~# more /etc/cron.deny
more: stat of /etc/cron.deny failed: No such file or directory
root@ubuntu:~# more /etc/cron.allow
more: stat of /etc/cron.allow failed: No such file or directory
```

On Ubuntu systems, the absence of both of these files indicates that all users can use the `crontab` command. This is not typical of most Linux distributions, but rather specific to Ubuntu based systems.

Step 12

Place the name `sysadmin` in the `/etc/cron.deny` file to prevent the `sysadmin` user from using the `crontab` command:

```
echo "sysadmin" > /etc/cron.deny
```

```
root@ubuntu:~# echo "sysadmin" > /etc/cron.deny
```

Step 13

Verify that these changes are effective by exiting from the root account shell and attempting to execute the `crontab` command:

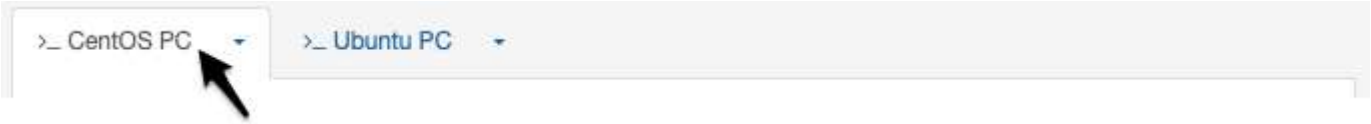
```
exit
crontab -e
```

```
root@ubuntu:~# exit
logout
sysadmin@ubuntu:~$ crontab -e
You (sysadmin) are not allowed to use this program (crontab)
See crontab(1) for more information
```

Step 14

The CentOS PC will be used for this portion of the lab.

Click on the **CentOS PC** tab to switch to the CentOS system:



When prompted for `centos login:`, enter `sysadmin`. When prompted for a password, enter `netlab123`:

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1062.el7.x86_64 on an x86_64

centos login: sysadmin
Password:

[sysadmin@centos ~]$ _
```

Step 15

Verify that the `at` package has been installed on this system by executing the following command:

```
rpm -q at
```

```
[sysadmin@centos ~]$ rpm -q at
at-3.1.13-24.el7.x86_64
```

The output that lists the package name and version information confirms that the package has been installed.

Step 16

Determine if the `atd` service is currently running by executing the following command:

```
systemctl status atd
```

```
[clima@srvUbuntu~]$ systemctl status atd service atd status
● atd.service - Job spooling tools
   Loaded: loaded (/lib/systemd/system/atd.service; enabled; vendor preset: dis
abled)
   Active: active (running) since Wed 2019-12-11 19:03:50; 0min ago
 Main PID: 638 (atd)
 CGroup: /system.slice/atd.service
           └─638 /usr/sbin/atd -f $OPTS

Dec 11 19:03:50 localhost.localdomain systemd[1]: Started Job spooling tools
```

Based on the output of the previous command, the atd service is `active (running)` and `enabled`.

The `MAIN PID` in our virtual machine environment may be different than the one in the output above.

Step 17

Execute the following command to execute a command two minutes from now:

```
at now + 2 min
```

```
[clima@srvUbuntu~]$ at now + 2 min
at>
```

When the `at>` prompt is displayed, type the following:

```
echo "test" > /tmp/test.output
Enter
Ctrl+D
```

```
at> echo "test" > /tmp/test.output
at> <EOT>
```

```
job 1 at 2015-04-20 02:50
```

Step 18

Wait at least **two minutes** before verifying that the `echo` command executed. The result should be a new file in the `/tmp` directory called `test.output`. Execute the following command to verify that the `echo` command executed:

```
cat /tmp/test.output
```

```
[clima@srvUbuntu~]$ cat /tmp/test.output
test
```

Step 19

Execute the following command to schedule a command to be executed 24 hours from now:

```
at tomorrow
```

```
[clima@srvUbuntu~]$ at tomorrow
at>
```

When the `at>` prompt is displayed, type the following:

```
echo "next day" >> /tmp/test.ouput
Enter
Ctrl+D
```

```
at> echo "next day" >> /tmp/test.output
at> <EOT>
job 2 at 2015-04-21 02:50
```

Step 20

View the jobs for the `sysadmin` account by executing the following command:

```
atq
```

```
[clima@srvUbuntu~]$ atq
2       2015-04-21 02:50 a sysadmin
```

The only job listed is job `2`. Job `1` was executed previously in this lab and once it had completed, it was removed from the job queue.

Step 21

Execute the following commands to remove job `2` from the queue and verify that it no longer exists:

```
atrm 2
atq
```

```
[clima@srvUbuntu~]$ atrm 2
[clima@srvUbuntu~]$ atq
```

Step 22

Access to the `atd` service can only be controlled by the root account. Switch to the `root` user by executing the following command:

```
su -
```

When prompted for the password, enter: `netlab123`

```
[clima@srvUbuntu~]$ su -
Password:
Last login: Wed Dec 11 19:07:33 UTC 2019 on console
[clima@srvUbuntu ~]#
```

To determine who can currently use the `at` command, execute the following commands:

```
more /etc/at.deny
more /etc/at.allow
```

```
[clima@srvUbuntu ~]# more /etc/at.deny


[clima@srvUbuntu ~]# more /etc/at.allow
/etc/at.allow: No such file or directory
```

On Redhat based systems, a non-existent `at.allow` file combined with an *empty* `at.deny` file means that any user can use the `at` command.

Step 23

Execute the following command to prevent the `sysadmin` user from using the `at` command:

```
echo "sysadmin" > /etc/at.deny
```

```
[clima@srvUbuntu ~]# echo "sysadmin" > /etc/at.deny
```

Step 24

Execute the following command to return to the `sysadmin` account:

```
exit
```

```
[clima@srvUbuntu ~]# exit
logout
```

Step 25

Attempt to use the `at` command by executing the following command:

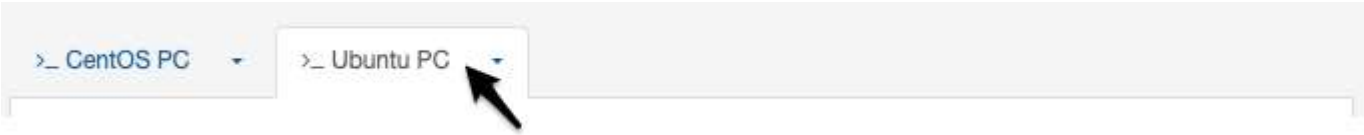
```
at now + 5 min
```

```
[clima@srvUbuntu~]$ at now + 5 min
You do not have permission to use at.
```

Step 26

The Ubuntu PC will be used for this portion of the lab.

Click on the **Ubuntu PC** virtual machine tab on the right.



Systems that have `systemd` as a replacement for the traditional `init` process provide an alternative to `crond` for scheduling jobs and managing services, called *systemd timer*.

The `systemctl` command is used to interface with and control systemd units. The syntax to use the `systemctl` command is the following:

```
systemctl COMMAND UNIT_NAME [OPTION]
```

To check what systemd-timers are active on the system, use the following command:

```
systemctl list-timers
```

```
[clima@srvUbuntu~]$ systemctl list-timers
NEXT                 LEFT          LAST                         PASSED       UNIT
Mon 2019-11-25 06:36:54 UTC  6h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily-u
pgrade.timer
Mon 2019-11-25 11:42:48 UTC  11h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily.
timer
Mon 2019-11-25 12:21:54 UTC 12h left   Mon 2019-11-25 00:03:06 UTC 2min 6s ago motd-new
s.timer
Tue 2019-11-26 00:00:00 UTC 23h left   Mon 2019-11-25 00:00:08 UTC 5min ago apt-daily.t
imer
```

```
Mon 2019-11-25 00:00:00 UTC 6 days left Mon 2019-11-25 00:00:08 5min ago fstrim.timer


5 timers listed.
Pass --all to see loaded but inactive timers, too.
lines 1-9/9 (END)
```

Type **q** to return to the prompt:

```
q
```

Step 27

To disable a timer, the `disable` command can be used. Use the following command to disable the `fstrim.timer` systemd-timer. Enter the `netlab123` password when prompted:

```
sudo systemctl stop fstrim.timer
```

```
[clima@srvUbuntu~]$ sudo systemctl stop fstrim.timer
[sudo] password for sysadmin:
[clima@srvUbuntu~]$
```

Verify the `fstrim.timer` systemd-timer has been disabled:

```
systemctl list-timers
```

```
[clima@srvUbuntu~]$ systemctl list-timers
NEXT                LEFT          LAST                          PASSED       UNIT
Mon 2019-11-25 06:36:54 UTC  6h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily-u
pgrade.timer
Mon 2019-11-25 11:42:48 UTC  11h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily.
timer
Mon 2019-11-25 12:21:54 UTC 12h left   Mon 2019-11-25 00:03:06 UTC 2min 6s ago motd-new
s.timer
Tue 2019-11-26 00:00:00 UTC 23h left   Mon 2019-11-25 00:00:08 UTC 5min ago apt-daily.t
imer


4 timers listed.
Pass --all to see loaded but inactive timers, too.
lines 1-8/8 (END)
```

Type **q** to return to the prompt:

```
q
```

Step 28

Enable the `fstrim.timer` systemd-timer using the following command:

```
sudo systemctl start fstrim.timer
```

```
[clima@srvUbuntu~]$ sudo systemctl start fstrim.timer
[sudo] password for sysadmin:
[clima@srvUbuntu~]$
```

Verify the `fstrim.timer` systemd-timer has been enabled:

```
systemctl list-timers
```

```
[clima@srvUbuntu~]$ systemctl list-timers
NEXT                         LEFT          LAST                         PASSED       UNIT
Mon 2019-11-25 06:36:54 UTC  6h left       Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily-upgrade.timer
Mon 2019-11-25 11:42:48 UTC  11h left      Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily.timer
Mon 2019-11-25 12:21:54 UTC 12h left       Mon 2019-11-25 00:03:06 UTC 2min 6s ago motd-news.timer
Tue 2019-11-26 00:00:00 UTC 23h left       Mon 2019-11-25 00:00:08 UTC 5min ago apt-daily.timer
Mon 2019-11-25 00:00:00 UTC 6 days left Mon 2019-11-25 00:00:08 5min ago fstrim.timer


5 timers listed.
Pass --all to see loaded but inactive timers, too.
lines 1-9/9 (END)
```

Type **q** to return to the prompt:

```
q
```

Step 29

Verify that `/home/sysadmin/newfile` does not exist.

```
ls /home/sysadmin/newfile
```

```
[clima@srvUbuntu~]$ ls /home/sysadmin/newfile
ls: cannot access '/home/sysadmin/newfile': No such file or directory
```

Step 30

Create a timer that will create the `/home/sysadmin/newfile` file one minute after entering this command via the `systemd-run` command. Note that creating a timer, a one time or repeating timer, requires superuser privileges on most systems.

```
sudo systemd-run --on-active="1m" /bin/touch /home/sysadmin/newfile
```

```
[clima@srvUbuntu~]$ sudo systemd-run --on-active="1m" /bin/touch /home/sysadmin/newfile
Running timer as unit: run-r99591573c1c84a689d7bc1833214b6b8.timer
Will run service as unit: run-r99591573c1c84a689d7bc1833214b6b8.timer
```

The `.timer` file name in our virtual machine environment may be different than the one in the output above.

Step 31

After **one minute**, verify that `/home/sysadmin/newfile` has been created.

```
ls /home/sysadmin/newfile
```

```
[clima@srvUbuntu~]$ ls /home/sysadmin/newfile
/home/sysadmin/newfile
```

To see how much time is left before the timer executes, execute the following command:

```
[clima@srvUbuntu~]$ systemctl list-timers
NEXT                         LEFT          LAST                         PASSED       UNIT
Wed 2019-12-11 21:36:54 UTC  23s left n/a                               n/a          run-r99591573
c1c84a689d7bc1833214b6b8.timer
Mon 2019-11-25 06:36:54 UTC  6h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily-u
pgrade.timer
Mon 2019-11-25 11:42:48 UTC  11h left  Sun 2019-11-24 23:44:52 UTC 20min ago apt-daily.
timer
Mon 2019-11-25 12:21:54 UTC 12h left   Mon 2019-11-25 00:03:06 UTC 2min 6s ago motd-new
s.timer
Tue 2019-11-26 00:00:00 UTC 23h left   Mon 2019-11-25 00:00:08 UTC 5min ago apt-daily.t
imer
Mon 2019-11-25 00:00:00 UTC 6 days left Mon 2019-11-25 00:00:08 5min ago fstrim.timer


6 timers listed.
Pass --all to see loaded but inactive timers, too.
lines 1-10/10 (END)
```

The `LEFT` field in the output above states that there is 23s left until the new timer job runs.

This lab is complete.

Networking Fundamentals

13.1 Introduction

In today's computing environment, it is rare to have a *stand-alone* machine. In almost all cases, a computer will be connected to a network in order to provide the access that users need.

Network access is critical for many reasons. Typically, updates to the operating system are made via the network using software repositories. If the system is a server that provides services, such as file-sharing or web pages, then a network connection is essential.

Providing access to the network is the responsibility of the root user on the system. Unfortunately, this can sometimes be a difficult task. Network cards are not all standardized, so additional configuration may be required. In addition, there are several components that need to be configured for the system to connect to the network, including IP address settings, gateway settings, and DNS (Domain Name Service) settings.

This chapter will focus on the fundamentals of networking, as well as how to configure your system to connect to the network and troubleshoot networking issues.

13.2 Networking Basics

Every computer and device on a network has a unique identifier. Just as you would address a letter to send via postal mail, computers use the unique identifier to send data to specific computers and devices on a network. Most networks today, including all computers on the internet, use the *TCP/IP* protocol developed by the *Internet Engineering Task Force* (IEFT) as the standard for how to communicate on the network.

In the TCP/IP model, the unique identifier for a computer is its *IP address*. An IP address can be either *static*, which is assigned manually, or *dynamic*, which is assigned by the *Dynamic Host Configuration Protocol* (DHCP) running as a service on the network.

A routing table is a small in-memory database managed by every device on the network (i.e., computer, router, etc.). It is used to calculate the optimal journey through other routers in the same or other networks for messages it is responsible for forwarding to a destination address.

TCP/IP is a combination of the two most popular protocols. *The TCP (Transmission Control Protocol)* protocol is used for reliable delivery of data between computers connected through a *LAN (Local Area Network)* or internet, and the *IP (Internet Protocol)* protocol is mainly responsible for the routing of data packets to the destination address.

There are two versions of the IP protocol used across the internet, *IP version 4 (IPv4)* and the more current *IP version 6 (IPv6)*, with IPv4 being in use on the majority of the systems today.

Consider This

No explanation of computer networking would be complete without referencing the OSI Model. Like so many other parts of computing today, the ability for systems to communicate with each other begins with an internationally accepted standard. The OSI (Open Systems Interconnection) reference model is a conceptual model created by ISO (International Organization for Standards) first defined in 1978 with the goal of standardizing how computer systems communicate with each other. It encompasses and references many different protocols and standards to achieve this and has proven to be very successful for designing and troubleshooting computer networks.

The OSI Model is comprised of seven different conceptual layers, each responsible for a different function within the network and each with its own protocol data unit (PDU), which represents the unit of data handled at that layer. The following is a summary of the layers in the OSI model:

Layer	Purpose
7. Application	User interface, Application Programming Interface (API)
6. Presentation	Data representation, encryption

Layer	Purpose
5. Session	Controls connections between hosts, maintains ports and sessions
4. Transport	Uses transmission protocols to transmit data (TCP, UDP)
3. Network	Determines path of data, IP
2. Data Link	Physical addressing (MAC), delivery of frames (Protocol Data Units (PDU))
1. Physical	Transmits raw data between physical medium

While specific knowledge of the OSI Model is not required for the LPIC-1 certification, understanding its components and how they work together will greatly enhance your ability to troubleshoot networking issues.

13.3 IPv4 Addresses

The IP address is used to identify the network interface of a device (i.e., phone, computer, etc.) on the network. Each computer on a network is assigned a unique IP address, much like each house on a street is assigned a unique number. The IP address is a numerical identifier in *decimal dotted quad* format since there are four values in an IPv4 address (for example, `192.224.10.8`). A server or PC can support more than one network interface, and each of the interfaces can be assigned a distinct IP address.

The IP address is made up of 4 *octets*, which are sets of 8-bit values. The value of each of these octets can range from decimal values `0 – 255` (or in binary, `00000000-11111111`).

```
00001010. 00001001. 00001000. 00000001
|______|  |______|  |______|  |______|
    |         |         |         |
Octet #1   Octet #2   Octet #3  Octet #4
```

Note

The 8-bit binary format can be translated to decimal value by using a multiplier. Only bits set to `1` will use a multiplier, and the multiplier is based on the location of the bit in the octet. Bits set to `1` should be assigned a value of 2, and bits set to `0` should be assigned a value of 0. The following shows what multipliers to use, based on the location of the bit:

Multiplier	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	0

To demonstrate, based on the information above, we can determine by adding the multiplied bits (8+2) that the octet `00001010` has a decimal value of 10:

Octet	0	0	0	0	1	0	1	0
Value	0	0	0	0	8	0	2	0

An IPv4 network addressing scheme has been designed on the basis of the octets. It classifies networks into 5 classes: A, B, C, D, and E.

- **Class A** – The network is denoted by the first octet, and the remaining three octets are used to create subnets (to be discussed) or identify hosts on the network. The first bit of the first octet is always `0`, so the range of values permissible is `00000001 – 01111111`, i.e., `1 – 127` in decimal value (the first *number* of an IP address cannot be `0` by the definition of IP addresses). The network ID cannot have all bits set to either `1`s or `0`s, which means the total number of class A networks available is only 127.

However, the 127 network is a special network referred to a *loopback* network, not a real class A network that is used on the internet. An example of a class A address would be `65.16.45.126`.

- **Class B** – The network is denoted by the 1st and 2nd octets, and the remaining 2 octets are used to create subnets or identify hosts. The 1st and 2nd bits of the 1st octet are set to `1` and `0` respectively, so the range of values permissible is `10000000` – `10111111`, i.e., decimals `128` - `191`. An example of a Class B address would be `165.16.45.126`.
- **Class C** – The network is denoted by the 1st, 2nd, and 3rd octets, and the last octet is used to create subnets or identify hosts. The 1st, 2nd, and 3rd bits of the 1st octet are set to `1`, `1`, and `0` respectively, so the range of values permissible is `11000000` – `11011111`, i.e., decimals `192` – `223`. An example of a Class C address would be `205.16.45.126`.
- **Class D** – These addresses are not assigned to network interfaces and are used for multicast operations such as audio-video streaming. The 1st, 2nd, 3rd, and 4th bits of the first octet are set to `1`, `1`, `1`, and `0` respectively, so the range of values permissible is `11100000` - `11101111`, i.e., decimals `224` - `239`. An example of a Class D address would be `224.0.0.6`.
- **Class E** – These addresses are reserved for future use.

13.4 Understanding Network Masks

In order for computers to communicate directly on the same network (i.e. not connected to another network via a router or gateway), all of the computers must be on the same *subnet*. A subnet is either an entire class A, B, or C network or a portion of one of these networks. To take a large class network and create a smaller portion, use a *subnet mask*.

The subnet mask is used to differentiate the network and subnet components of the IP address. The subnet mask is not an IP address in itself; it is a numeric pattern used to indicate the portion of the IP address that contains the network identifier. The service provider will allocate a network from Class A, B, or C type, and use subnets to logically partition the network so that multiple sub-networks can be created.

The addresses for Class A, B, and C have default masks as follows:

Network Class	Subnet Mask
Class A	`255.0.0.0`
Class B	`255.255.0.0`
Class C	`255.255.255.0`

For example, consider a standard Class A IP address 10.9.8.1 and its default subnet mask expressed in binary format:

- IP Address:

```
00001010. 00001001. 00001000. 00000001
```

- Subnet Mask:

```
11111111. 00000000. 00000000. 00000000
```

In the subnet mask, the octets where the mask bits are `1` represent the network ID, whereas the octets where the mask bits are `0` represent the host ID.

```
11111111. 00000000. 00000000. 00000000
|______|  |_____________________________|
   |                   |
```

```
Network ID          Host ID
```

In the previous example IP Address `10.9.8.1`, the network ID is `10` and the host ID is `9.8.1`.

For an example of a custom subnet, assume that a class C network, `202.16.8.0`, has been allocated. Network addresses with custom subnets are typically assigned by your internet service provider (ISP) and are used for systems that need to access the internet directly, such as web or email servers. In this case, you would take two bits from its default subnet mask and replace them with `1s` as follows:

- Default Mask:

  ```
  11111111. 11111111. 11111111. 00000000
  ```

- Subnet Mask:

  ```
  11111111. 11111111. 11111111. 11000000
  ```

Using the two bits will give 4 (2^2) subnets, the remaining 6 bits will give 64 (2^6) host addresses for each subnet. The address range will be as follows:

- `202.16.8.0 - 202.16.8.64`
- `202.16.8.65 - 202.16.8.128`
- `202.16.8.129 - 202.16.8.192`
- `202.16.8.193 - 202.16.8.224`

The subnet mask `255.255.255.192` has partitioned the Class C network address into 4 sub-networks, and each of these sub-networks can be assigned to a particular group of machines.

13.5 Public and Private IPv4 Addresses

There are two types of IP addresses used on a network: *public* and *private*. The **InterNIC (Network Information Center)** is the global body responsible for assigning public addresses. They assign class-based network IPs, which are always unique. The public addresses are available with internet routers so that data can be delivered correctly.

Most organizations use the internet for email and for browsing the web. To accomplish this, only systems such as email servers, web servers, and proxies (which handle intermediary requests from clients seeking resources from other servers) need direct connectivity to the internet so that users outside of the LAN can connect to these servers. On some occasions, additional services, like file-sharing servers, will need to have a public IP address for direct connection to the internet.

On the other hand, users who work on their own machines and want to connect to the internet do not need a globally unique IP address. Instead, they can be assigned private IP addresses, which are then converted to public IP addresses by the gateway/router.

According to **RFC 1918**, a portion of the IP address space has been designated as “private addresses”. This range of addresses does not overlap with the public addresses. The private addresses can be reused, which means the risk of running out of addresses on the internet has been mitigated. The private address space is not directly reachable through the internet. To access devices utilizing the private address space, a router using NAT (Network Address Translation) will need to be configured. There are three blocks of private addresses:

```
10.0.0.0/8
```

This **Class A** address allows the range of addresses from `10.0.0.1` to `10.255.255.254`. The 24 bits from the host ID are available for subnetting.

```
172.16.0.0/12
```

This **Class B** network allows the range of addresses from `172.16.0.1` to `172.31.255.254`. The 20 bits from the host ID are available for subnetting.

```
192.168.0.0/16
```

This **Class C** network allows the range of addresses from `192.168.0.1` to `192.168.255.254`. The 16 bits from the host ID are available for subnetting.

13.6 Comparing IPv4 and IPv6

The IPv4 addresses are made up of four 8-bit octets for a total of 32-bits. This means the maximum number of possible addresses is 2^{32}, which is less than 4,294,967,296.

The IPv6 addresses are based on 128-bits. Using similar calculations, as shown above, the maximum number of possible addresses is 2^{128}, which gives a massively large pool of addresses. The IPv6 addresses consist of eight 16-bit segments, which means each segment can have 2^{16} possible values. IPv6 addresses are usually expressed in hexadecimal format.

A brief comparison of IPv4 vs. IPv6:

Feature	IPv4	IPv6
Address Size	32-bit	128-bit
Address Format	Decimal dotted quad `192.168.20.8`	Hex notation `4AAE:F200:0342:AA00:0135:4680:7901:ABCD`
Number of addresses	2^{32}	2^{128}
Broadcasting	Uses broadcasting to send data to all hosts on a subnet	No broadcast addresses, uses multicast scoped addresses as a way to selectively broadcast

Note

A complete discussion of the merits of IPv6 vs. IPv4 is beyond the scope of this class. Realize that the differences between the two include many other aspects, including security, package contents, and speed of transport.

13.7 Default Route

The function of routing is to send an IP packet, consisting of a header (source and destination address) and encapsulated data, from one point to another.

All devices have *routing* tables, which contain routes used to calculate the optimal journey of the messages that it is responsible for forwarding through other routers in the same or other networks. When a computer sends packets to another computer, it consults its routing table. If a packet is being sent to a destination on the same subnet, no routing is needed, and the packet is sent directly to the computer. If a packet is being sent to the internet or another network, then the first “hop” or place a packet will go is to whatever is in the default gateway field (a network setting with the IP address of the network router) and lets the router (a network device that forwards IP packets) decide the optimal path forward.

The router for the network will have its own routing table, including its own *default route* (which router to send packets to when the destination is in another network or subnet). The routing table is a list of other routers that are connected to the current router. If the router receives a packet for a network destination that it has in its routing table (typically, this will be another local network), it simply forwards it. Otherwise, the router will send the packet to its default route (typically, this will be the way to get to the internet).

To view the existing routing table, execute the `route` command:

The examples in this chapter may not match the output in our virtual environment. Additional practice is available in the lab.

```
root@localhost:~# route
Kernel IP routing table
Destination       Gateway       Genmask         Flags Metric Ref Use Iface
default               192.168.1.1     0.0.0.0           UG    0      0    0    eth1
192.168.1.0       *           255.255.255.0     UG        0      0    0    eth1
192.168.2.0       *           255.255.255.0     UG    0      0    0    eth0
```

In the output of the kernel routing table, the first column contains the `Destination` network address. The word `default` signifies the default route. The second column contains the defined `Gateway` for the specified destination. In the event that an asterisk `*` is shown, it means that a gateway is not needed to access the destination network. The `Genmask` column shows the netmask for the destination network. In the `Flags` column, a `U` means the route is up and available, whereas the `G` means that the specified gateway should be used for this route. The `Metric` column defines the distance to the destination. This is typically listed in the number of hops (number of routers between source and destination). The `Ref` column is not used by the Linux Kernel, and the `Use` column is used to define the number of lookups for the route. Finally, the `Iface` column is used to define the exit interface for this route.

The default route in this example has been configured to use the `eth1` interface, the second network card on the system. The network device with an IP address of `192.168.1.1` is designated as the default gateway. The default gateway is a router that will pass packets from this network to another network. The address that is specified for the default gateway must be on a network to which your system is connected.

To understand the previous output, consider the following:

1. If the designation addresses of the network packet are in the `192.168.2.0/24` network, then the packet is broadcast on the network that the `eth0` network card is attached to. The source IP address of the packet will be `192.168.2.1` (the IP address assigned to the `eth0` network card).
2. If the designation addresses of the network packet are in the `192.168.1.0/24` network, then the packet is broadcast on the network that the `eth1` network card is attached to. The source IP address of the packet will be `192.168.1.106` (the IP address assigned to the `eth0` network card).
3. All other network packets are sent to the router with the IP address of `192.168.1.1` via the `eth1` network card.

All other network packets are sent to the router with the IP address of 192.168.1.1 via the eth1 network card.

```
root@localhost:~# route add default gw 192.168.1.1 eth1
```

Now, if any of the routes in the routing table do not match the specified address, then the packet will be forwarded to `192.168.1.1` (the *default route*).

Note

The `route` command has been superseded by the `ip route` command, which is part of the `iproute2` suite of utilities that will be discussed in a later chapter.

13.8 Understanding TCP

The *Transmission Control Protocol (TCP)* provides connection-oriented service between two applications exchanging data. The protocol guarantees delivery of data.

For example, consider accessing a server via a web browser. The user's computer will resolve the IP address for the web server and connect to the web server via the standard HTTP port 80. After establishing the connection, the client and server processes exchange information about the socket size used to buffer data and the initial sequence number of packets.

The sequence number mechanism in the header ensures ordered delivery of data. The web server will then service GET requests sent on the HTTP port for web pages. For error control, TCP uses the acknowledgment number in the header. The client sends the acknowledgment number to the server. If the server sends 2000 bytes of data to the client and the client acknowledges only 1000 bytes, then it indicates loss of data. The web server will then retransmit the data.

13.9 Using FTP

FTP is a protocol that uses TCP for transport and reliable delivery. The `ftp` command provides the user interface to the standard File Transfer Protocol (FTP). Using the `ftp` utility, a user can transfer files to and from remote machines. It can be used for UNIX as well as non-UNIX machines. The service is provided by the `ftpd` daemon, which uses TCP protocol and listens to the FTP ports specified in the `/etc/services` file for FTP requests.

To connect to a particular host via `ftp`, execute the command:

```
ftp ftp_server_host_name [or IP address]
```

For example:

```
sysadmin@localhost:~$ ftp 127.0.0.1
ftp>
```

If the FTP service is running on the specified host, the user will be prompted to log in using a known user name and password.

Consider This

The user name and password used to log in to an FTP server are sent over the network(s) without being encrypted. The opportunity for someone to capture this information as it crosses from network to network and host to host is very real and dangerous. Unless your FTP server is on a private network with only highly trusted users, you should not log in with your real credentials.

Most FTP servers that are internet-connected will allow you to perform an anonymous login, so none of your real credentials can be compromised. To log in anonymously, you use the user name "anonymous" with any password you want.

Once logged into the FTP server, the `ftp>` prompt will be displayed. You will find that you can view and navigate the filesystem of the FTP server similar to your own local filesystem using commands like the `ls` and `cd` commands.

The `lcd` command can be used to change your local working directory to affect where files will get downloaded or uploaded.

```
ftp>  lcd
Local Directory now /home/sysadmin
ftp>  lcd  /tmp
Local Directory now /tmp
```

To execute other commands on your local machine while logged in to the FTP server, prefix a command with an `!` exclamation point.

```
ftp> !ping 127.0.0.1
```

```
PING 127.0.0.1[127.0.0.1]  56(84)  bytes of data.

64 bytes from 127.0.0.1:  icmp_seq=0  ttl=255  time=0.5 ms

64 bytes from 127.0.0.1:  icmp_seq=1  ttl=255  time=0.3 ms



---  127.0.0.1  ping statistics ---

2 packets transmitted, 2 packets received,  0% packet loss round-trip min/avg/max = 0.3
/0.3/0.5 ms
```

For additional assistance, use the `help` command while logged in to the FTP server.

```
ftp> help

ftp> ?
```

The default file transfer mode for the FTP utility is ASCII, which is used for ordinary text files. To transfer other types of files (i.e., program file, zip, or tar files, etc.), it is recommended that the server is in binary transfer mode. To set the file transfer mode to binary, execute the command:

```
ftp> bin
```

To retrieve or download a file from the remote machine to your local machine, use the `get` command. For example, to download the `devsrccode.tar` file (binary) from the FTP server named `server1` to the `/tmp` directory on the local machine, execute the following `ftp` commands:

```
sysadmin@localhost:~$ ftp server1

ftp> bin

ftp> lcd  /tmp

Local Directory now /tmp

ftp> get devsrccode.tar

local: devsrccode.tar remote: devsrccode.tar

200 PORT command successful.  Consider using PASV.

150 Opening BINARY mode data connection for devsrccode.tar (1000 bytes).

226 Transfer complete.

ftp> !ls

devsrccode.tar
```

To send or upload a file from your local machine to the remote machine, use the `put` command. For example, to upload the `devsrccode.tar` file from the local machine to the server, execute the following command:

```
ftp> put  devsrccode.tar
```

To download multiple files, use the `mget` command. If you wanted to download all the `*.tar` files in a particular directory from the server to the local machine, execute the following command:

```
ftp> mget *.tar
```

To quit `ftp`, execute the following command:

```
ftp> quit
```

```
sysadmin@localhost:~$
```

13.10 Using Telnet

Telnet is another protocol that uses TCP for transport and reliable delivery. The Telnet Protocol is used for interactive communication with host machines using TCP/IP. The client's machine becomes a virtual terminal for the remote host. The `telnet` command provides the user interface to the standard Telnet protocol. The service is provided by the `telnetd` daemon, which services requests from clients to connect to the Telnet port specified in the `/etc/services` file.

To open a `telnet` session to the server, execute the following command:

```
telnet host_name [or IP address]
```

If either the hostname or the IP address is specified, then the `telnet` command implicitly executes the open option to open a connection to the host. If the telnet service is running on the specified host, then the user will be prompted to log in using their user name and password.

```
sysadmin@localhost:~$ telnet server1
Trying 192.9.49.10 …
Connected to server1
Escape character is '^]'.
Welcome to server1



login: root
Password:
Last login:  Mon Mar 8 8:35:15 from localhost
root@localhost:~#
```

To end a `telnet` session, log out normally with the `logout` or `exit` command. If you find yourself stuck within the `telnet` client, use **Ctrl+]** (the right square bracket key) to get to a `telnet` prompt. From the `telnet` prompt, type `help` to get assistance or `quit` to exit `telnet`.

The `telnet` protocol can be very useful as a troubleshooting tool to determine whether a daemon is listening and responding on a certain port. To specify a port other than the standard port 23, you add the port number after the host specification. For example, to see if there is a service or daemon listening on port 25 on your own host, execute:

```
root@localhost:~# telnet localhost 25
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 localhost ESMTP Sendmail 8.14.4/8.14.4/Debian-4.1ubuntu1; Mon, 8 Mar 2015
8:40:15 GMT; (No UCE/UBE) logging access from: localhost(OK)-localhost [127.0.0.
1]
```

```
^C
```

Consider This

The `telnet` protocol suffers the same defect as the `ftp`; the packets that are sent are not encrypted. This means that your user name, your password, and any data sent during a `telnet` session can easily be captured, so your credentials may become compromised. `ssh` is a safer alternative for `telnet`, though it is not in the current LPIC-1 objectives.

13.11 Understanding UDP

User Datagram Protocol (UDP) provides connectionless service between two applications exchanging data. Unlike TCP, UDP has no error control and does not guarantee the transfer of data.

UDP sends data without notifying the receiver prior to sending. As a result, it does not offer either ordered or reliable delivery. UDP is like the traditional postal system; you are not notified that a letter will be delivered to your mailbox.

The header of UDP packets is lightweight as compared to TCP packets since it does not contain sequence or acknowledgment numbers. It uses a simple, optional checksum mechanism for error checking. UDP is faster than TCP and is used in services such a VoIP, Streaming Video (Netflix), and DNS (Domain Name Service).

13.12 /etc/services File

In order to make it easy to distinguish between packets destined for different services, each service is assigned one or more *port numbers*. If you consider an IP address to be like a street number for an apartment complex, a port number is like a number for a specific residence within the apartment complex.

The `/etc/services` file is used for mapping application service names to port numbers. This file exists on each system and can be modified only by the root user. Generally, there is no need to modify this file since most of the services use their own configuration files to determine port numbers; however, this file does provide a good reference for standard port numbers.

Consider This

The `/etc/services` file is queried by programs using the `getportbyname()` Application Program Interface (API) to determine the port number that they should open a socket connection to. For example, if the `finger` command is used to do a name lookup for a user on a remote machine, then it executes a `getportbyname()` API for the `finger` service and fetches the corresponding port number, which is 79.

The use of this API is now fairly rare, typically reserved for legacy UNIX services. Most services in modern Linux use separate configuration files to specify the ports that they communicate through. However, the `/etc/services` file is useful as most default service configuration files will initially have the same port numbers as found by the `/etc/services` file. In the cases that the numbers are different, it was likely a change made by the local system administrator.

A snippet of the `/etc/services` file looks like the following:

```
tcpmux          1/tcp          # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp          sink null
discard         9/udp          sink null
systat         11/tcp          users
daytime        13/tcp
```

```
daytime         13/udp
netstat         15/tcp
qotd         17/tcp         quote
msp         18/tcp         # message send protocol
```

The first column of the file indicates the official service name. The second column indicates the port number and transport protocol (TCP (Transmission Control Protocol) or UDP (User Datagram Protocol - like TCP without error control) it uses. The third column shows aliases that may be associated with the protocol. The final column typically provides a description of the protocol.

The port numbers in the range `0` – `1023` are assigned as per the Request For Comments (RFC) 1700 standard. This RFC also mentions the standard use of ports beyond `1023`.

The mappings of some of the commonly-used TCP and UDP ports are:

Port Number	Service
20/tcp	ftp-data
21/tcp	ftp
21/udp	ftp
22/tcp	**ssh**
22/udp	**ssh**
23/tcp	telnet
25/tcp	smtp
53/tcp	domain (dns)
53/udp	domain (dns)
80/tcp	**http**
80/udp	**http**
110/tcp	pop3
110/udp	pop3
119/tcp	nntp
139/tcp	netbios-ssn
139/udp	netbios-ssn
143/tcp	imap2
143/udp	imap2
161/tcp	snmp
161/udp	snmp
443/tcp	**https**
443/udp	**https**

Port Number	Service
465/tcp	ssmtp
993/tcp	imaps
993/udp	imaps
995/tcp	pop3s
995/udp	pop3s

13.13 Querying DNS Servers

The `host` and `dig` commands are used for DNS (Domain Name Server) lookups. A DNS server provides hostname to IP address translation. DNS is covered in more detail at the end of this unit.

Although it has been deprecated for a long time, there is also a command called `nslookup`. Many Linux systems still have the `nslookup` command, but there has been no development for this tool for a long time, and some distributions have dropped it entirely.

The `host` command is used to resolve hostnames to IP addresses and IP addresses to hostnames. The utility uses UDP for transport of queries to the servers listed in the `/etc/resolv.conf` file.

To find the IP address of a host, execute the following command:

```
sysadmin@localhost:~$  host netdevgroup.com
netdevgroup.com has address 152.46.6.105
```

The `host` command also does reverse lookups of IP addresses to names by executing the command as shown:

```
sysadmin@localhost:~$  host 152.46.6.105
152.46.6.105.in-addr.arpa domain name pointer www.netdevgroup.com.
```

To find the DNS servers for a domain, do not specify the host and use the `-t` option with an argument of `ns`, execute the command:

```
sysadmin@localhost:~$  host -t ns netdevgroup.com
netdevgroup.com name server ns-1328.awsdns-38.org
netdevgroup.com name server ns-1635.awsdns-12.co.uk.
netdevgroup.com name server ns-461.awsdns-57.com
netdevgroup.com name server ns-703.awsdns-23.net
```

The `dig` (Domain Information Groper) command is used for troubleshooting the configuration of DNS servers. DNS server administrators like the output of the `dig` command because it is in the same format that the information is entered into a DNS server configuration file. The utility performs DNS lookups and displays the responses received from the name servers listed in the `/etc/resolv.conf` file.

To perform a simple lookup of a hostname, execute the following command:

```
sysadmin@localhost:~$  dig example.com
; <<>> DiG 9.9.5-3ubuntu0.1-Ubuntu <<>> example.com
;; global options: +cmd
```

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14876
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
.. OUTPUT OMITTED ...
;; ANSWER SECTION:
example.com.            86400   IN      A       192.168.1.2
;; AUTHORITY SECTION:
example.com.            86400   IN      NS      example.com.
;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Mar 11 15:23:13 UTC 2015
;; MSG SIZE  rcvd: 70
```

To view the trace of domain name servers from the servers where the lookup begins, and each name server along the way, execute the following command:

```
sysadmin@localhost:~$  dig +trace example.com
```

For a reverse lookup, using an IP address instead of hostname, execute the following command:

```
sysadmin@localhost:~$  dig -x 192.168.1.2
```

Some of the key options of this command are:

Option	Meaning
`-f filename`	Operate in batch mode by reading a list of lookups to be done from the specified file
`-p port#`	Query the specified port other than the standard DNS port
`-4`	Use IPv4 query transport
`-6`	Use IPv6 query transport

13.14 Understanding ICMP

The TCP protocol provides an error control mechanism but does not contain information about possible reasons for errors. The *Internet Control Message Protocol (ICMP)* is a diagnostic protocol used to notify about network problems that are causing delivery failures. This protocol is considered as a part of the IP protocol, though it is processed differently than normal IP packets. Some of the common types of ICMP messages are:

- Destination Unreachable
- Redirect (i.e., use an alternative router instead of this one)
- Time exceeded (i.e., IP TTL exceeded)
- Source Quench (i.e., host or router is congested)
- Echo Reply/Request (i.e., the `ping` command)

A summary of the key features and differences of TCP, UDP, IP, and ICMP is shown below but is beyond the scope of this course and is not covered on the LPIC 102 exam. It is recommended that you conduct an internet search on “Understanding Network Protocols” and “TCP/IP family of protocols” to learn more.

Feature	TCP	UDP	IP	ICMP
OSI Networking Layer	Transport	Transport	Internet	Internet
Packet Header Size	20 bytes	8 bytes	Minimum 20 bytes	8 bytes
Unit of Data	Segment	UDP Datagram	IP Datagram	Messages
Contents	Application Data	Application Data	Encapsulates TCP/UDP application data	Query and Error Messages
Type of Connection	Connection-Oriented	Connectionless	Connectionless	Connectionless

Network Configuration

14.1 Introduction

The system administrator needs to know how to configure the system for network access. This chapter discusses configuring routing tables, the manual and automatic configuration of interfaces, and common issues related to network configuration.

14.2 TCP/IP Configuration

Recall that the *Transmission Control Protocol (TCP)* provides connection-oriented service between two applications exchanging data. To configure a network port, or interface on legacy systems, use the `ifconfig` command. The `ifconfig` command stands for *interface configuration* and is used to display network configuration information.

This command is used for the following functions:

- Assigning static IP Address
- Viewing current network configuration
- Setting the netmask
- Setting the broadcast address
- Enable/disable network interfaces

The `ifconfig` command can be used without options or arguments to display all interfaces on the network, or with options and an interface name as an argument:

```
ifconfig [INTERFACE] [OPTIONS]
```

Not all network settings are covered in this course, but it is important to note from the output below that the IP address of the primary network device (`eth0`) is `192.168.1.2` and that the device is currently active (`UP`):

```
root@localhost:~# ifconfig
eth0      Link encap:Ethernet  HWaddr b6:84:ab:e9:8f:0a
          inet addr:192.168.1.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::b484:abff:fee9:8f0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:95 errors:0 dropped:4 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25306 (25.3 KB)  TX bytes:690 (690.0 B)
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:460 (460.0 B)  TX bytes:460 (460.0 B)
```

Note

The `lo` device is referred to as the *loopback* device. It is a special network device used by the system when sending network-based data to itself.

To view all the network interfaces on the system, execute the `ifconfig -a` command. Typically, this output won't be any different from the previous `ifconfig` command unless there are some interfaces that are not currently active.

To view the details of a specific interface, execute the following command:

```
sysadmin@localhost:~$ ifconfig eth0
```

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 192.168.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:c0ff:fea8:102  prefixlen 64  scopeid 0x20<link>
        ether 02:42:c0:a8:01:02  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1038 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

To assign an IP address to an interface, execute the following command:

```
sysadmin@localhost:~$ sudo ifconfig eth0 192.168.1.3
```

To assign a netmask to an interface, execute the following command:

```
sysadmin@localhost:~$ sudo ifconfig eth0 netmask 255.255.255.192
```

To assign a broadcast address to an interface, execute the following command:

```
sysadmin@localhost:~$ sudo ifconfig eth0 broadcast 192.168.1.63
```

The `ifconfig` command can be used to enable and disable an interface. To disable (deactivate) a network interface, execute the following command:

```
sysadmin@localhost:~$ sudo ifconfig eth0 down
```

To view the changes that were made, execute the following command again:

```
sysadmin@localhost:~$ ifconfig eth0
eth0: flags=4163<BROADCAST,MULTICAST>  mtu 1450
        inet 192.168.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:c0ff:fea8:102  prefixlen 64  scopeid 0x20<link>
        ether 02:42:c0:a8:01:02  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1038 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note from the output above that the `eth0` interface is no longer `UP` and `RUNNING`.

To enable (activate) a network interface, execute either of the commands:

```
sysadmin@localhost:~$ sudo ifconfig eth0 up
```

To verify the interface is enabled, execute the following command again:

```
sysadmin@localhost:~$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 192.168.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:c0ff:fea8:102  prefixlen 64  scopeid 0x20<link>
```

```
        ether 02:42:c0:a8:01:02  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1038 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The `ifconfig` command can also be used to modify network settings temporarily. Typically these changes should be permanent, so using the `ifconfig` command to make such changes is relatively rare.

14.2.1 Setting the Hostname

The *hostname* is used to identify the system by applications such as web servers. On Debian-derived and modern Red Hat-derived systems, the `/etc/hostname` file contains this information, while legacy Red Hat-derived systems store this information in the `/etc/sysconfig/network` file. This file is read at boot time to set the hostname.

The `hostname` command is used to set and view the system's host and domain name. It is the system administrator's responsibility to assign an appropriate hostname. It cannot be longer than 64 characters and can contain alphanumeric [`a-z`] [`0-9`], period `.` characters, and hyphen `-` characters only.

To view the currently assigned hostname of the system, execute the `hostname` or *short name cut at the first dot* `hostname -s`, or *full domain name* `hostname -f` command:

```
sysadmin@localhost:~$ hostname -s
localhost
```

The name displayed above is returned by the `gethostname()` application programming interface (API).

To view the fully-qualified domain name, execute the following command:

```
sysadmin@localhost:~$ hostname -f
test.example.com
```

To set the hostname of the system, the root user can execute the following command:

```
root@localhost:~# hostname example.com
root@localhost:~# hostname
example.com
```

The examples above may not match the output in our virtual environment.

Note that setting the hostname using the `hostname` command results in a change that is only persistent until the next system boot.

The `/etc/hosts` file is used for mapping hostnames with IP addresses. It is a flat-file with one record on each line. The format of the file is:

```
IP Address    Host Name    Alias
```

A sample `/etc/hosts` file will look like the following:

```
127.0.0.1       localhost
192.168.4.8     apps.sample.com      apps
```

```
192.168.4.12    vm1.sample.com        vm1
```

The `Alias` field is used for mapping short names or labels to a host.

The functionality of the `/etc/hosts` file has been relegated by DNS but is still used in the following situations:

- **Bootstrapping**: This file is referred to during system startup since the DNS service is not started at this point.
- **Isolated Nodes**: If a node is not connected to the internet, it is unlikely to use DNS. The `/etc/hosts` file is useful for such nodes.
- **NIS**: The records in the hosts file are used as input for the NIS (Network Information Services) database.

Systemd systems use an alternative to the `hostname` command, the `hostnamectl` command. Similar to the `hostname` command, the `hostnamectl` command can also be used to query and set system hostnames, but the `hostnamectl` command provides additional categories for hostnames; *static*, *pretty*, and *transient* which are described below:

- **Static**: A *static hostname* is limited to [`a-z`], [`0-9`], hyphen `-`, and period `.` characters (no spaces i.e., `localhost` or `ndg-server`). This hostname is stored in the `/etc/hostname` file. Static hostnames can be set by a user.
- **Pretty**: Hostname can be in a human-readable format using any valid UTF-8 characters and can include special characters (i.e., Sarah's Laptop or Joe's Home PC).
- **Transient**: The transient hostname is a dynamic hostname usually set by the kernel to localhost by default. A dynamic hostname can be modified if needed. The transient hostname can be modified by DHCP or mDNS at runtime.

Hostnames can be up to 64 characters, but it is recommended that static and transient hostnames are limited to only 7 bit ASCII lowercase characters with no spaces or dots and conforming to strings acceptable for DNS domain names.

To demonstrate, in order to view the current hostname, simply use the `hostnamectl` command by itself or with the `status` subcommand:

```
sysadmin@localhost:~$ hostnamectl status
      Static hostname: localhost
            Icon name: computer-vm
              Chassis: vm
           Machine ID: 5b91eb5e50594030b48b28f103cf5cd6
              Boot ID: c04fc73d939a4fc18b279cd46d08f8d7
       Virtualization: kvm
     Operating System: Ubuntu 18.04.2 LTS
               Kernel: Linux 4.15.0-45-generic
         Architecture: x86-64
```

To change the local machine hostname, use the `hostnamectl` command with the `set-hostname` subcommand, this must be done with elevated permissions:

```
sysadmin@localhost:~$ sudo hostnamectl set-hostname netlab01
[sudo] password for sysadmin:
sysadmin@localhost:~$ hostnamectl status
      Static hostname: netlab01
            Icon name: computer-vm
              Chassis: vm
```

```
            Machine ID: 5b91eb5e50594030b48b28f103cf5cd6
               Boot ID: c04fc73d939a4fc18b279cd46d08f8d7
       Virtualization: kvm
     Operating System: Ubuntu 18.04.2 LTS
               Kernel: Linux 4.15.0-45-generic
         Architecture: x86-64
```

14.2.2 Configuring DNS

The *DNS (Domain Name System)* is the mapping table for the internet, allowing any computer or device to access websites, mail servers, etc. by using a name (i.e., google.com, mail.comcast.net) instead of an IP address. The DNS implementation is based on a distributed database of network names and IP addresses and query interfaces to retrieve information.

The `/etc/resolv.conf` is the configuration file for DNS resolvers. The information in this file is normally set up by network initialization scripts. If DNS servers are like giant *phone books* of domain names and IP addresses, the `/etc/resolv.conf` file is used to tell a computer where the phone book is located on the network or internet.

A sample `/etc/resolv.conf` file looks like the following:

```
# /etc/resolv.conf
domain        sample.com
search        sample.com
# central nameserver
nameserver    191.74.10.12


sortlist 191.74.10.0 191.74.40.0
```

The format of the `/etc/resolv.conf` is:

```
directive    value1, value2…
```

The configuration directives used in this file are:

Option	Meaning
`nameserver`	IP address of the name server that the resolver will use Maximum of 3 servers can be listed
`domain`	Domain name to be used locally
`search`	Search list to be used for hostname lookup
`sortlist`	Allow addresses to be sorted The list is specified by IP addresses and optionally the netmask
`options`	Used to modify the resolver's internal variables using certain keywords i.e., `attempts: 3` will set the retry count for querying the name servers to 3

Consider This

The DNS implementation queries a name server, and if the query times out without a response, then the next name server is queried. If any server responds, then the query is finished; otherwise, all the servers in the list will be queried for the maximum number of attempts specified.

14.2.3 Name Service Switch

The Name Service Switch (NSS) is used by the system administrator to specify which name information source (i.e., local files, LDAP, etc.) to use for different categories (i.e., `passwd`, `shadow`, `hosts`, etc.), and in which order the sources are searched. The client applications query the name service database using APIs such as:

- `gethostbyname()`
- `getaddrinfo()`
- `getnetent()`

The `/etc/nsswitch.conf` file is used to store the information used for name service switching. It is a text file with columns that contain the following information:

- Database name
- Lookup order of sources
- Actions permitted for the lookup result

A process that needs to lookup host or network-related information will refer to the configuration for the required database in this file.

A sample portion of an `/etc/nsswitch.conf` file will look like the following:

```
sysadmin@localhost:~$ cat /etc/nsswitch.conf
...
passwd:         compat
group:          compat


...
hosts:          files dns
networks:       files
...
```

The first column contains the database name, followed by the services to be queried in the order of their occurrence in the file.

For example, the current sample of the `/etc/nsswitch.conf` file demonstrates the `hosts` database services configured like the following:

```
hosts:        files dns
```

When a hostname lookup is performed, the `files` entry will make use of the `/etc/hosts` file to perform the resolving. If the query does not return any results, then the query will be sent to the DNS resolver.

By changing the order of the name services listed for a particular database, like `hosts`, the administrator could change whether the local `/etc/hosts` file is consulted before or after the DNS servers listed in `/etc/resolv.conf`.

```
hosts:        dns files
```

In the event of the query not returning any results, specific actions can also be mentioned in the `/etc/nsswitch.conf` file.

```
hosts: dns [NOTFOUND=return] files
```

In the above example, the DNS resolver will try to resolve the hostname. If a match is not found, then the resolver will immediately return the `NOTFOUND` status and `/etc/host` file will not be queried.
The `/etc/host` file will only be queried if the DNS resolver service itself is unavailable due to some reason.

14.2.4 Configuring Routing Tables

As discussed in the previous chapter, routing tables are used by the kernel to store information about how to reach a network directly or indirectly. The `route` command is used to view, as well as update, the IP routing table.

Any system using the TCP/IP protocol to send network packets will have a routing table. The routing function is managed by the IP layer. The routing table will decide the forwarding IP address for the packet.

Static routes in the kernel's routing table can be set using the `route` command (note, as shown in the previous chapter, the `ip` command can also display and modify routes). To view the current routing table, execute the following command:

```
sysadmin@localhost:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.192 U     0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
```

The fields in the output are as follows:

Option	Meaning
`Destination`	Destination host or network
`Gateway`	IP address of the gateway `0.0.0.0` means "no route, broadcast on this network"
`Genmask`	Subnet mask for the destination network Set to `0.0.0.0` for the default route
`Flags`	Values include `C` (Cache entry), `G` (Use Gateway) and `U` (up) For more options, refer to the man page for route
`Metric`	The number of hops to the target, used as a measure of distance
`Ref`	Number of references to this route, this is not used in Linux
`Use`	Count of lookups done for this route
`Iface`	Interface to use for sending packets for this route

If an administrator wants to add a route, they must specify a gateway which is connected to a network that one of the system's interfaces is on. If an attempt is made to add a route that will be unusable due to an unreachable gateway, then it will be ignored and an error will be displayed. For example, because the `eth0` interface in the previous example is on the `192.168.1.0-63` network, any routes that get added for this interface must use a router on that network.

For instance, to be able to reach the `192.56.78.0/255.255.255.0` network, a router like `192.168.1.1` could be used, if that machine is connected to both the `192.168.1.0` network and the `192.56.78.0` network (it would likely have two network interfaces). To add this route to the `eth0` interface, the administrator could execute the following `sudo` command (or as root):

```
sysadmin@localhost:~$ sudo route add -net 192.56.78.0 netmask 255.255.255.0 gw 192.168.
1.1 dev eth0
```

To view the changes in the routing table, execute the following command:

```
sysadmin@localhost:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.56.78.0     192.168.1.1     255.255.255.0   UG    0      0        0 eth0
192.168.1.0     *               255.255.255.192 U     0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
```

To add a default gateway, execute the following command:

```
sysadmin@localhost:~$ sudo route add default gw 192.168.1.1
```

To view the changes in the routing table, execute the following command:

```
sysadmin@localhost:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
192.56.78.0     192.168.1.1     255.255.255.0   UG    0      0        0 eth0
192.168.1.0     *               255.255.255.192 U     0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
```

To verify the connectivity to a network that is now available via the new route, execute the following `ping` command, using an IP address of a machine that should be available on the newly accessible network:

```
sysadmin@localhost:~$ ping 192.168.1.1
```

If the `ping` command is working correctly, then the gateway has been configured correctly.

To view the kernel's cached routing information, execute the following command:

```
sysadmin@localhost:~$ route -Cn
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref    Use Iface
```

If a setup is required where a particular host is blocked when packets are routed, then execute the command:

```
sysadmin@localhost:~$ sudo route add host 192.168.1.62 reject
```

This will make the specified host unreachable.

To delete a route from the routing table, an administrator can execute a command like the one that added it, except using `del` instead of `add`. For example, to delete the route and default gateway that was added earlier, an administrator could execute the following `sudo` commands (or as root):

```
sysadmin@localhost:~$ sudo route del -net 192.56.78.0 netmask 255.255.255.0 gw 1
92.168.1.1 dev eth0
sysadmin@localhost:~$ sudo route del default gw 192.168.1.1
```

To view the changes in the routing table execute the following command:

```
sysadmin@localhost:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.192 U     0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
```

14.3 Network Interfaces

The term *network interface* refers to the point of connection between a computer and a network. It can be implemented in either hardware or software. The *network interface card (NIC)* is an example of the hardware interface, while the loopback interface (`127.0.0.1`) is an example of the software interface.

14.3.1 Network Interface Configuration

The Linux system comes with default drivers for the general network interfaces. If the *NIC (Network Interface Card)* can be loaded using the default driver, then it will be detected during initialization. If the NIC is not supported by the default driver, then the driver will have to be loaded into the kernel before the card can be used.

For example, by performing some research, the administrator has determined that the driver or kernel module that is needed for a network interface is called `veth`. To manually load this driver, execute the following `sudo` command (or as root):

```
sysadmin@localhost:~$ sudo modprobe veth
```

To view information about the driver, the *list hardware* `lshw` command can be used:

```
sysadmin@localhost:~$ sudo lshw -c network | grep veth
       configuration: autonegotiation=off broadcast=yes driver=veth driverversion=1.0 d
uplex=full ip=192.168.1.2 link=yes multicast=yes port=twisted pair speed=
10Gbit/s
```

To verify if the driver has been loaded correctly, execute the following command:

```
sysadmin@localhost:~$ lsmod | grep veth
```

```
veth                    12390  0
```

If the details of the driver are shown, then the driver has been successfully installed, and the new interface card `eth1` can be viewed using the `ifconfig` command:

```
sysadmin@localhost:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:CB:F3:51
          inet6 addr: fe80::a00:27ff:fecb:f351/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:4914 (4.7 KiB)
```

This output indicates that `eth1` has been recognized by the system. The next step is to configure and assign an IP address to `eth1`. To temporarily assign the IP address `192.168.10.12` to the `eth1` device, execute the following command:

```
sysadmin@localhost:~$ sudo ifconfig eth1 192.168.81.12 netmask 255.255.255.0
```

To verify if the NIC is working correctly, execute the command:

```
sysadmin@localhost:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:CB:F3:51
          inet addr:192.168.81.12  Bcast:192.168.81.255
      Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fecb:f351/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:9008 (8.7 KiB)
```

The `UP` status indicates that the interface has been enabled and the `RUNNING` status indicates that the configuration of the interface is complete and it is operational. The `RX` (receive) and `TX` (transmit) packet counts have increased, which indicate that network traffic is being routed using the `eth1` interface.

Some of the fields in the output significant for analyzing network errors are:

Option	Meaning
`RX errors`	Number of received packets which were damaged
`RX dropped`	Number of packets dropped due to reception errors
`RX overruns`	Number of received packets which experienced data overrun
`RX frame`	Number of received packets which experienced frame errors
`TX errors`	Number of packets which experienced data transmission errors
`TX dropped`	Number of packets dropped due to transmission errors

Option	Meaning
`TX overruns`	Number of transmitted packets which experienced data overrun
`TX carriers`	Number of transmitted packets which experienced loss of carriers
`TX collisions`	Number of transmitted packets which experienced Ethernet collisions possibly due to network congestion

14.3.1.1 RedHat Interface Configuration

On a legacy Red Hat-derived system, the `/etc/sysconfig/network` file contains host and routing details for all configured network interfaces. A sample file would look like the following:

```
NETWORKING=yes
HOSTNAME=gsource1.localdomain
GATEWAY=192.168.122.1
```

For each network interface, there is a corresponding interface configuration script file `/etc/sysconfig/network-scripts/ifcfg-<interface-name>`. A network interface can have its settings automatically assigned via a DHCP (Dynamic Host Configuration Protocol) server or statically assigned within this file. Any text following a # is considered a comment and is used for documentation.

Any `GATEWAY` specified in an interface configuration file would override the `GATEWAY` specified in the `/etc/sysconfig/network` file. A sample file `/etc/sysconfig/network-scripts/ifcfg-eth0` for the `eth0` device where the interface is configured automatically via DHCP would look like the following:

```
DEVICE="eth0"        # name of the device
NM_CONTROLLED="no"    # device is not NetworkManager managed
ONBOOT=yes        # activate interface automatically
TYPE=Ethernet        # type of interface
BOOTPROTO=dhcp        # use a DHCP to configure interface
```

On a Red Hat-derived system, a static configuration of the `/etc/sysconfig/network-scripts/ifcfg-eth0` file would look like the following:

```
DEVICE="eth0"        # name of the device
NM_CONTROLLED="no"    # device is not NetworkManager managed
ONBOOT=yes        # activate interface automatically
TYPE=Ethernet        # type of interface

BOOTPROTO=none        # use static configuration
IPADDR=192.168.0.3    # set the IP address
NETMASK=255.255.255.0    # set the subnet mask
GATEWAY=192.168.0.1    # set the default router
DNS1=192.168.0.254    # set the primary DNS server
```

14.3.1.2 Debian Interface Configuration

For Debian-derived systems, the `/etc/network/netplan` directory contains files that are used to configure the interfaces. Any text that follows a `#` is considered a comment. A sample `*.yaml` (YAML Ain't Markup Language) human-readable file for an interface that uses DHCP for address configuration would look like the following:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
    version: 2
    renderer: networkd
    ethernets:
        ens3:
            dhcp4: yes
```

A sample interfaces file for using static addresses would look like the following:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      addresses:
        - 10.10.10.2/24
      gateway4: 10.10.10.1
      nameservers:
          search: [mydomain, otherdomain]
          addresses: [10.10.10.1, 1.1.1.1]
```

14.3.2 NetworkManager

Originally developed by Red Hat, NetworkManager provides automatic detection and configuration of network interfaces on a Linux system. It works for both wired and wireless interfaces as well as having support for some modems and Virtual Private Network (VPN) connections.

The *NetworkManager* service is a GNOME project hosted at https://wiki.gnome.org/Projects/NetworkManager. NetworkManager provides a graphical utility for configuring network interfaces and is used in many Linux distributions.

NetworkManager uses the *network management command line interface* `nmcli` tool to manage network connections and display information about devices on a network. The `nmcli` command uses the following syntax:

```
nmcli [OPTIONS] OBJECT [COMMAND][ARGUMENTS...]
```

The `OPTIONS` for the `nmcli` command can be found by visiting the `nmcli` man page or by using the `nmcli -help` command. Commonly used options include the *terse* `-t` option that displays concise output and the *pretty* `-p` option, which makes the output easily readable by printing headers and aligning values.

The `OBJECT` field can be one of the following:

Object	Meaning
`general`	Display information about or modify the status of NetworkManager.
`networking`	Display information about or modify the network managed by NetworkManager.
`connection`	Display information about or modify connections managed by NetworkManager.
`device`	Display information about or modify devices managed by NetworkManager.
`radio`	Display the status of, and enable or disable, the radio switches.

To demonstrate, the following command can be used to show the existing network device:

The output in the examples below may not match the output in our virtual environment.

```
[sysadmin@centos ~]$ nmcli device
DEVICE  TYPE      STATE      CONNECTION
ens3    ethernet  connected  eth0
```

Consider This

When using the `nmcli` command, the object can be abbreviated. For example, the `nmcli device` command used in the example above can be shortened to `nmcli dev` or `nmcli d`:

```
[sysadmin@centos ~]$ nmcli dev
DEVICE  TYPE      STATE      CONNECTION
ens3    ethernet  connected  eth0
[sysadmin@centos ~]$ nmcli d
DEVICE  TYPE      STATE      CONNECTION
ens3    ethernet  connected  eth0
```

The `nmcli` command is useful for displaying additional information about a specific connection. In the example below, the `nmcli` command is used with the `con` object, the `show` subcommand and the *pretty* `-p` option, to display information about the `eth0` connection:

```
[sysadmin@centos ~]$ nmcli -p con show eth0
```

Below are the first twenty-four lines of output from the command above:

```
===============================================================================
                        Connection profile details (eth0)
===============================================================================
connection.id:                          eth0
connection.uuid:                        5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03
connection.stable-id:                   --
```

```
connection.type:                        802-3-ethernet
connection.interface-name:              --
connection.autoconnect:                 yes
connection.autoconnect-priority:        0
connection.autoconnect-retries:         -1 (default)
connection.auth-retries:                -1
connection.timestamp:                   1571100767
connection.read-only:                   no
connection.permissions:                 --
connection.zone:                        --
connection.master:                      --
connection.slave-type:                  --
connection.autoconnect-slaves:          -1 (default)
connection.secondaries:                 --
connection.gateway-ping-timeout:        0
connection.metered:                     unknown
connection.lldp:                        default
connection.mdns:                        -1 (default)
lines 1-24
```

The output above provides detailed information such as the universally unique identifier (`uuid`), type (`ethernet`), and `autoconnect` (connect when the system starts) settings for the `eth0` connection.

The `nmcli` command can also be used to create a new connection. To add a connection, the following syntax can be used:

```
nmcli con add {OPTIONS} [IP]/[NETMASK] [GATEWAY]
```

For example, to add a connection named `eth1`, define the connection as Ethernet and specify the IP address, network mask, and gateway, the following command can be used:

```
[sysadmin@centos ~]$ nmcli con add con-name eth1 ifname eth1 type ethernet \ip4 10.0.2.
18/24 gw4 10.0.2.2
```

The `nmcli` command above uses the `con` object with the `add` subcommand followed by a series of options, which are summarized below:

Options	Meaning
`con-name`	Specifies the name of the network connection. In the example above, the `con-name eth1` option adds a connection named `eth1`.
`ifname`	Name of the interface (device) that is used for the connection.
`type`	Specifies the connection type. Various types of connections exist such as ethernet, wifi, VLAN, bridge, and
`ip4`	Specify an IPv4 IP address and netmask for the connection.

Options	Meaning
`gw4`	Specify the gateway used for the connection.

The `con show` command can be used to view the new connection:

```
[sysadmin@centos ~]$ nmcli -p con show eth1
```

The output below shows the details for the new `eth1` connection:

```
===============================================================================
                        Connection profile details (eth1)
===============================================================================
connection.id:                          eth1
connection.uuid:                        a0ab5f1e-6bd0-4af0-8a5b-3621aa5ff6d2
connection.stable-id:                   --
connection.type:                        802-3-ethernet
connection.interface-name:              eth1
connection.autoconnect:                 yes
connection.autoconnect-priority:        0
connection.autoconnect-retries:         -1 (default)
connection.auth-retries:                -1
connection.timestamp:                   0
connection.read-only:                   no
connection.permissions:                 --
connection.zone:                        --
connection.master:                      --
connection.slave-type:                  --
connection.autoconnect-slaves:          -1 (default)
connection.secondaries:                 --
connection.gateway-ping-timeout:        0
connection.metered:                     unknown
connection.lldp:                        default
connection.mdns:                        -1 (default)
lines 1-24
```

14.3.3 Wireless Interfaces

As Linux makes its way onto more desktop and laptop computers, configuring wireless interfaces is becoming more important than ever. NetworkManager provides a command line utility to configure and connect with wireless networks.

To configure a wireless interface, first determine the name by using the `nmcli d` command to view interface devices.

```
sysadmin@localhost:~$ nmcli d
DEVICE   TYPE       STATE         CONNECTION
enp4s0   ethernet   connected     Wired connection 1
wlp2s0   wifi       disconnected  --
lo       loopback   unmanaged     --
```

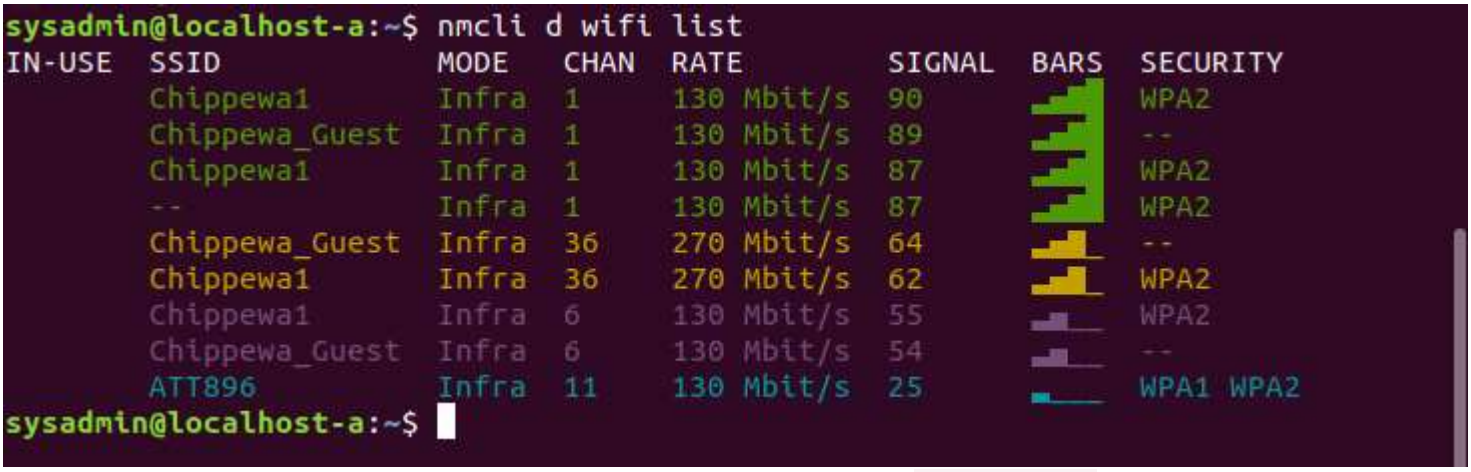
In this example, there are three interfaces, a wired Ethernet (`enp4s0`), a loopback (`lo`), and the wifi interface (`wlp2s0`). The next step is to make sure the wifi interface is active by using the `nmcli radio` command:

```
sysadmin@localhost:~$ nmcli radio wifi on
```

Note

The default state for wireless radios is `on`.

After verifying that the wifi interface is active, the next step is to select the name of the wifi network, the Service Set Identifier (SSID), to use for the connection by listing all the available wifi networks. The `nmcli` command can be used with the `wifi list` option to list the available wifi networks:



To connect to the desired SSID, in this case, `Chippewa_Guest`, use the `wifi connect` option and the password for the wifi network; for this example, the password is `1882`.

```
sysadmin@localhost:~$ nmcli d wifi connect Chippewa_Guest password 1882
Device 'wlp2s0' successfully activated with '4c9476f7-4dc7-45c8-8c26-854f0480435b'.
```

As demonstrated in the output above, the terminal will provide confirmation of successful activation.

14.3.4 iproute2 tools

It is important to mention that many of the tools used in this module, such as the `ifconfig` command, are being phased out in newer Linux distributions in favor of the `iproute2` suite of tools. The table below shows some of the configuration and troubleshooting utilities and the `ip` commands that replace them:

Legacy net-tools	Replacement iproute2 Commands	Usage
`ifconfig`	`ip address`, `ip link`, `ip -s`	Configure addresses and links
`route`	`ip route`	Manage routing tables

Legacy net-tools	Replacement iproute2 Commands	Usage
`arp`	`ip neigh`	Display and manage neighbors (hosts that share the same IP address/link)
`iptunnel`	`ip tunnel`	Manage tunnels (shared communication channel between networks)
`nameif`	`ifrename`, `ip link set name`	Manage network interface name
`ipmaddr`	`ip maddr`	Manage multicast (group of hosts on a network)
`netstat`	`ip -s`, `ss`, `ip route`	Display network information

Note

The `iproute2` commands will be covered in greater detail later in the course.

14.4 systemd-networkd

A great benefit to using modern Linux systems running systemd is the `systemd-networkd` system daemon. This background program detects and manages network configurations, automatically configuring devices as they appear, such as when a USB Ethernet connector is plugged in, or a WiFi radio is turned on. The `systemd-networkd` daemon is also useful for creating virtual devices such as the devices used with containers and other cloud objects.

The `systemd-networkd` daemon functions through configuration files, which reside in the `/usr/lib/systemd/network/`, `/run/systemd/network`, and `/etc/systemd/network` directories. Like other systemd configuration files, there are numerous options available for administrators to specify how devices should be configured on startup.

Name resolution services on systems that use systemd are handled by `systemd-resolved`. This systemd service tells local applications where to find domain name information on a network. The `systemd-resolved` service can operate in four different modes, which are:

1. Using a systemd DNS stub file located at `/run/systemd/resolve/stub-resolv.conf`.
2. Preserving the legacy `resolv.conf` file we learned about earlier in this chapter.
3. Automatic configuration with a network manager.
4. Manual, or local DNS stub mode where alternate DNS servers are provided in the `resolved.conf` file.

It is important to understand how `resolv.conf` and `systemd-resolved` interact with each other to ensure proper DNS configuration. The systemd-resolved system service creates its own DNS/DNSSEC stub resolver that local applications can use for network name resolution. It also reads the data in `/etc/resolv.conf` to discover other DNS servers configured in the system. This compatibility function only works directly on the `/etc/resolv.conf`, not on symlinks.

The `systemd-resolved` service provides a tool called `resolvectl`, which can be used for resolving domain names, IPv4 and IPv6 address, and DNS resource records and services. The syntax below shows how to use the `resolvectl` command:

```
resolvectl [OPTIONS...] {COMMAND} [NAME...]
```

To demonstrate, find the IP address for a domain name using `resolvectl` with the `query` option:

```
sysadmin@localhost:~$ resolvectl query netdevgroup.com

netdevgroup.com: 34.214.209.23
```

```
-- Information acquired via protocol DNS in 25.1ms.
-- Data is authenticated: no
```

The query above uses the `systemd-resolved.service` resolver service to find the IP address for netdevgroup.com.

DNS services and the programs which configure and use them are constantly evolving. Many legacy programs and system services have been renamed and revised to ensure backward compatibility as well as providing enhanced functionality. Administrators should strive to keep current with changes in name resolution technology as it affects much of what they do on a daily basis.

Network Troubleshooting

15.1 Introduction

Once a network has been configured, the system administrator will inevitably encounter problems that will require troubleshooting. This chapter discusses some of the common problems and the tools used for debugging.

Some of the commands in this unit have been discussed in previous units, so the focus here will be on the options used from the troubleshooting perspective. Root privileges are required in order to execute command options that update network settings.

Some commonly-observed network issues are:

1. Network interface card not detected by the system
2. IP address not assigned correctly to the system
3. Not being able to communicate with the router
4. DNS service not reachable

15.2 Display Network Status

When troubleshooting a network, it is useful to gather information about networking services such as open ports, interface statistics, routing tables, and network connections. The following sections will cover common legacy and updated Linux tools available for displaying network status.

15.2.1 Using netstat

The `netstat` command is used by the system administrator to monitor the traffic on the network, and check connections that are not trustworthy. While administrators should be familiar with the `netstat` command, it should be noted that it is a legacy command that is being phased out as new systems come online. The `ss` command, covered in the next section, should be used in most cases.

To list all ports, execute the following `netstat` command:

```
sysadmin@localhost:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  4      [ ]         DGRAM                    237543   /dev/log
unix  2      [ ]         DGRAM                    240199
unix  2      [ ]         DGRAM                    236517
```

The output of the previous command lists all ports, including those that are not currently being used. Those ports that are currently being used (active) are marked with the state `LISTEN`. To view only the listening ports, execute the `netstat -l` command.

To display a summary of details for each protocol, execute the following command:

```
sysadmin@localhost:~$ netstat -s
Ip:
    1621 total packets received
    0 forwarded
    0 incoming packets discarded
    1621 incoming packets delivered
    372 requests sent out
    4776 dropped because of missing route
Icmp:
    122 ICMP messages received
    80 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 106
        echo requests: 8
        echo replies: 8
    122 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 106
        echo request: 8
        echo replies: 8
IcmpMsg:
        InType0: 8
        InType3: 106
        InType8: 8
        OutType0: 8
        OutType3: 106
        OutType8: 8
```

```
Tcp:
    7 active connections openings
    6 passive connection openings
    1 failed connection attempts
    0 connection resets received
    0 connections established
    74 segments received
    74 segments send out
    0 segments retransmited
    0 bad segments received.
    1 resets sent
Udp:
    70 packets received
    2 packets to unknown port received.
    0 packet receive errors
    176 packets sent
UdpLite:
TcpExt:
    6 TCP sockets finished time wait in fast timer
    12 packet headers predicted
    24 acknowledgments not containing data payload received
    TCPOrigDataSent: 36
IpExt:
    InBcastPkts: 1353
    InOctets: 435603
    OutOctets: 28714
    InBcastOctets: 412665
    InNoECTPkts: 1621
```

To view the kernel's routing table, execute the `netstat -r` command:

```
sysadmin@localhost:~$ netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     *               255.255.255.192 U         0 0          0 eth0
192.168.1.0     *               255.255.255.0   U         0 0          0 eth0
```

Consider This

The interface name `eth0` follows an older interface naming scheme which is being phased out and replaced by a newer Predictable Network Interface Names standard. Updated interface names are modeled after the *biosdevname* scheme which assigns fixed names to interfaces, like `enp4s0`, that are related to firmware information. In some cases, depending on the configuration of the operating system, the `ifconfig` command may not display Predictable Network Interface Names.

To view the details of specific interfaces, use the *interface* `-i` option when executing the `netstat` command:

```
sysadmin@localhost:~$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0      1369      0      2 0            57      0      0      0 BMRU
eth0:0     1500 0       - no statistics available -                        BMRU
lo         1500 0       268       0      0 0           268      0      0      0 LRU
```

If you include the `-c` option, the `netstat` command will display the interface information continuously after an interval of one (1) second. This is useful to watch the activity on the interfaces over a period of time.

The `netstat` command is also commonly used to display open *ports*. A port is a unique number that is associated with a service provided by a host. If the port is open, then the service is available for other hosts.

For example, you can log into a host from another host using the SSH service. The SSH service is assigned port #22. So, if port #22 is open, then the service is available to other hosts.

To see a list of all currently open ports, use the following command:

```
root@localhost:~# netstat -tln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.1.2:53          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

As you can see from the output above, port #22 is *listening*, which means it is open.

In the previous example, `-t` stands for TCP (recall this protocol from earlier in this course), `-l` stands for *listening* (which ports are listening) and `-n` stands for *show numbers, not names*.

Sometimes showing the names can be more useful, this can be achieved by leaving out the `-n` option:

```
root@localhost:~# netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 cserver.example.:domain *:*                     LISTEN
tcp        0      0 localhost:domain        *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:953           *:*                     LISTEN
tcp6       0      0 [::]:domain             [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:953           [::]:*                  LISTEN
```

On some distributions you may see the following message in the man page of the `netstat` command:

```
NOTE
     This program is obsolete. Replacement for netstat is ss. Replacement for
     netstat -r is ip route. Replacement for netstat -i is ip -s link.
     Replacement for netstat -g is ip maddr.
```

While no further development is being done on the `netstat` command, it is still an excellent tool for displaying network information. The goal is to eventually replace the `netstat` command with commands such as the `ss` and `ip` commands.

15.2.2 Using ss

The `ss` command is designed to show socket statistics and supports all the major packet and socket types. Meant to be a replacement for and to be similar in function to the `netstat` command, it also shows a lot more information and has more features.

A *network socket* is a communication endpoint between nodes (devices) on a network. Sockets use a socket address to receive incoming network traffic and forward it to a process on a machine or device. The socket address commonly consists of the IP address of the node that it is "attached" to and a port number.

The main reason a user would use the `ss` command is to view what connections are currently established between their local machine and remote machines, statistics about those connections, etc. To use the `ss` command, follow the syntax below:

```
ss [options] [FILTER]
```

Similar to the `netstat` command, you can get a great deal of useful information from the `ss` command just by itself, as shown in the example below.

```
root@localhost:~# ss
Netid  State      Recv-Q Send-Q       Local Address:Port                Peer Address:Por
t
u_str  ESTAB      0      0                    * 104741                    * 104740
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 14623      * 14606
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 13582      * 13581
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 16243      * 16242
u_str  ESTAB      0      0                    * 16009                     * 16010
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 10910      * 10909
u_str  ESTAB      0      0      @/tmp/dbus-LoJW0hGFkV 15706                * 15705
u_str  ESTAB      0      0                    * 24997                     * 24998
u_str  ESTAB      0      0                    * 16242                     * 16243
u_str  ESTAB      0      0          @/tmp/dbus-opsTQoGE 15471               * 15470
```

The output is very similar to the output of the `netstat` command with no options. The columns above are:

- `Netid` - the socket type and transport protocol
- `State` - Connected and Unconnected, depending on protocol
- `Recv-Q` - Amount of data queued up for being processed having been received
- `Send-Q` - Amount of data queued up for being sent to another host
- `Local Address` - The address and port of the local host's portion of the connection
- `Peer Address` - The address and port of the remote host's portion of the connection

The format of the output of the `ss` command can change dramatically, given the options specified, such as the use of the `-s` option, which displays mostly the types of sockets, statistics about their existence and numbers of actual packets sent and received via each socket type, as shown below:

```
root@localhost:~# ss -s
```

```
Total: 1000 (kernel 0)
TCP:   7 (estab 0, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
*         0         -         -
RAW       0         0         0
UDP       9         6         3
TCP       7         3         4
INET      16        9         7
FRAG      0         0         0
```

One common use for the `ss` command is determining which ports an interface is *listening* on. By using the `-l` option to list only those ports which are listening, and the `-t` option to show only TCP ports, you can quickly determine which ports are available for TCP communications.

```
sysadmin@localhost:~$ ss -lt
State    Recv-Q    Send-Q        Local Address:Port         Peer Address:Port
LISTEN   0         128              127.0.0.11:33906             0.0.0.0:*
LISTEN   0         10             192.168.1.2:domain             0.0.0.0:*
LISTEN   0         10               127.0.0.1:domain             0.0.0.0:*
LISTEN   0         128                0.0.0.0:ssh                0.0.0.0:*
LISTEN   0         128              127.0.0.1:953                0.0.0.0:*
LISTEN   0         10                     [::]:domain                [::]:*
LISTEN   0         128                    [::]:ssh                   [::]:*
LISTEN   0         128                   [::1]:953                   [::]:*
```

When troubleshooting UDP dependent services like DNS name resolution or some video streaming applications, you can use the `-u` option to display just the UDP sockets available. The example below only lists UDP sockets which are currently listening.

```
sysadmin@localhost:~$ ss -ul
State    Recv-Q    Send-Q        Local Address:Port         Peer Address:Port
UNCONN   0         0              192.168.1.2:domain             0.0.0.0:*
UNCONN   0         0                127.0.0.1:domain             0.0.0.0:*
UNCONN   0         0              127.0.0.11:47117               0.0.0.0:*
UNCONN   0         0                      [::]:domain                [::]:*
```

Consider This

The `ss` command typically shows many rows of data, and it can be somewhat daunting to try to find what you want in all that output. Consider sending the output to the `less` command to make the output more manageable. Pagers allow the user to scroll up and down, do searches, and many other useful functions inside the parameters of the `less` command.

15.2.3 Using ip

The `ifconfig` command is becoming obsolete (deprecated) in some Linux distributions and is being replaced with a form of the `ip` command, specifically `ip address`.

The `ip` command differs from `ifconfig` in several important manners, chiefly that through its increased functionality and set of options, it can almost be a one-stop-shop for configuration and control of a system's networking. The format for the `ip` command is as follows:

```
ip [OPTIONS] OBJECT COMMAND
```

While `ifconfig` is limited primarily to the modification of networking parameters, and displaying the configuration details of networking components, the `ip` command branches out to do some of the work of several other legacy commands such as `route` and `arp`.

Note: Linux and UNIX commands don't usually just disappear when they become obsolete; they stick around as a legacy command, sometimes for many years. The number of scripts that depend on those commands, and the amount of muscle memory amongst system administrators, makes it a good idea to keep them around for the sake of compatibility.

The `ip` command can initially appear to be a little more verbose than the `ifconfig` command, but it's a matter of phrasing and a result of the philosophy behind the operation of the `ip` command.

In the example below, both the `ifconfig` command and `ip` command are used to show all interfaces on the system:

```
sysadmin@localhost:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:71:f0:bb
          inet addr:172.16.241.140  Bcast:172.16.241.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe71:f0bb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8506 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1201 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8933700 (8.9 MB)  TX bytes:117237 (117.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:285 errors:0 dropped:0 overruns:0 frame:0
          TX packets:285 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:21413 (21.4 KB)  TX bytes:21413 (21.4 KB)
sysadmin@localhost:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group def
ault qlen 1000
    link/ether 00:0c:29:71:f0:bb brd ff:ff:ff:ff:ff:ff
    inet 172.16.241.140/24 brd 172.16.241.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe71:f0bb/64 scope link
       valid_lft forever preferred_lft forever
```

Both commands show the type of interface, protocols, hardware and IP addresses, network masks, and various other various information about each of the active interfaces on the system.

Another useful option with the `ip` command is the `-statistics` or `-s` option, which shows statistics for the object referenced in the command. For example, to show statistics for active IP addresses, you can type:

```
root@localhost:~# ip -s address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.⅛ scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
   RX: bytes  packets  errors  dropped  overrun  mcast
   6680       88       0       0        0        0
   TX: bytes  packets  errors  dropped  carrier  collsns
  6680      88        0       0        0        0
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_code1 state UP group defau
lt qlen 1000
   link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
   inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic ens3
     valid_lft 85864sec preferred_lft 85864sec
   inet6 fe80::5054:ff:fe12:3456/64 scope link
     valid_lft forever preferred_lft forever
   RX: bytes  packets  errors  dropped  overrun  mcast
   11020      29       0       0        0        0
   TX: bytes  packets  errors  dropped  carrier  collsns
   4278       39       0       0        0        0
```

Viewing the information in the output above shows that the `ens3` interface on the `10.0.2.0/24` network received 29 packets and transmitted 39 packets with zero errors and zero dropped packets. Multiple dropped packets could indicate a Layer 1 issue where network cabling could be the problem.

15.2.4 Using ping

The *packet internet groper* `ping` command is used to check the connectivity to a host. It is a simple test that can be performed from the command prompt when a particular network service is not available. This utility sends the ICMP protocol's `ECHO_REQUEST` (*ping*) datagram to a host, and the host sends an `ECHO_RESPONSE` (*pong*) datagram in response. Each datagram will have an IP header, an ICMP header, a *timeval* structure, and some additional bytes used for padding.

The *Time To Live (TTL)* value is the maximum number of IP routers that may attempt to route a packet. Every time a router attempts to route the packets, its TTL count is decremented by 1. If a router receives a packet and the TTL of a packet is zero, then the packet is discarded. Typically, a packet will have a maximum TTL of 30 *hops* by default. The Linux `ping` command continuously displays the TTL value of each packet it receives until **Ctrl-C** is pressed to cancel. The *count* `-c` option stops after sending *n* packets as seen in the example below, which sends 5 packets (`-c 5`) and then stops:

```
sysadmin@localhost:~$ ping -c 5 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.058 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.068 ms

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.056/0.061/0.068/0.006 ms
```

The hostname can also be specified instead of the IP address as follows:

```
sysadmin@localhost:~$ ping -c 5 example.com
PING example.com (192.168.1.2) 56(84) bytes of data.
64 bytes from example.com (192.168.1.2): icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from example.com (192.168.1.2): icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from example.com (192.168.1.2): icmp_seq=3 ttl=64 time=1.64 ms
64 bytes from example.com (192.168.1.2): icmp_seq=4 ttl=64 time=2.62 ms
64 bytes from example.com (192.168.1.2): icmp_seq=5 ttl=64 time=0.103 ms

--- example.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4020ms
rtt min/avg/max/mdev = 0.034/0.893/2.624/1.060 ms
```

Some of the key options of the `ping` command are:

Option	Meaning
`-c count`	Stop after sending count `ECHO_REQUEST` packets
`-s packetsize`	Specifies the number of data bytes to be sent

Option	Meaning
`-t ttl`	Sets the IP Time to Live
`-w timeout`	Sets the timeout in seconds for ping to exit

Note
By default, the Windows `ping` command pings 4 times. Linux will continue pinging indefinitely if the count switch is not used. To stop pinging, an administrator will need to press **Ctrl+C**.

The `ping6` command is similar to the `ping` command, but it uses `ICMPv6 ECHO_REQUEST` to verify network connectivity. The `ping6` command can use either a hostname or an IPv6 address to request a response from remote systems. Similar to the `ping` command, the `ping6` command will continue pinging until **Ctrl+C** is typed in the terminal.

```
ping6 ipv6.google.com
```

```
sysadmin@localhost:~$ ping6 ipv6.google.com

PING ipv6.google.com(dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e)) 56 data byte
s

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=1 ttl=55 t
ime=16.10 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=2 ttl=55 t
ime=18.5 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=3 ttl=55 t
ime=18.7 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=4 ttl=55 t
ime=15.2 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=5 ttl=55 t
ime=16.2 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=6 ttl=55 t
ime=15.10 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=7 ttl=55 t
ime=14.8 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=8 ttl=55 t
ime=16.4 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=9 ttl=55 t
ime=14.6 ms

64 bytes from dfw25s34-in-x0e.1e100.net (2607:f8b0:4000:808::200e): icmp_seq=10 ttl=55
time=14.10 ms

^C

--- ipv6.google.com ping statistics ---

10 packets transmitted, 10 received, 0% packet loss, time 20ms

rtt min/avg/max/mdev = 14.563/16.227/18.664/1.388 ms
```

15.2.5 Using traceroute

The `traceroute` command is used to trace the route of packets to a specified host. This utility uses the IP header's TTL field and tries to fetch an ICMP `TIME_EXCEEDED` response from each router on the path to the

host. The probing is done by sending ICMP ping packets with a small TTL value and then checking the ICMP `TIME_EXCEEDED` response. Network administrators use this command to test and isolate network problems. This command can be run by root users only.

To trace the route to a particular host, execute the following command:

```
sysadmin@localhost:~$ traceroute example.com

traceroute to example.com (192.168.1.2), 30 hops max, 60 byte packets

 1  example.com (192.168.1.2)  0.026 ms  0.015 ms  0.013 ms
```

Some of the key options of the `traceroute` command are:

Option	Meaning
`-T`	Probe using TCP SYN
`-f first_ttl`	Specifies the initial TTL value
`-m max_ttl`	Specifies the maximum number of hops to be probed
`-w timeout`	Sets the timeout in seconds to exit after waiting for a response to a probe

The `traceroute` command, commonly used for seeing how a transmission travels between a local host machine to a remote system can also be used for IPv6 connections. To use the `traceroute6` command, which is the same as `traceroute -6` to view the IPv6 path to `ipv6.google.com` execute the following command:

```
sysadmin@localhost:~$ traceroute6 ipv6.google.com

traceroute to ipv6.l.google.com (2607:f8b0:4009:811::200e) from 2600:380:5c6b:897e:443b
:aa22:6729:5980, 30 hops max, 24 byte packets

1 2600:380:5c6b:897e:3504:968f:d7a4:fbcc (2600:380:5c6b:897e:3504:968f:d7a4:fbcc)   2.2
19 ms 1.481 ms 1.725 ms

2 * * *

3 * * *

4 2600:300:2000:2604::1 (2600:300:2000:2604::1) 72.578 ms 52.919 ms 34.129 ms

5 2600:300:2000:2622::1 (2600:300:2000:2622::1) 40.852 ms 38 ms 40.568 ms

6 * * *

7 2001:1890:ff:ffff:12:83:188:242 (2001:1890:ff:ffff:12:83:188:242) 69.811 ms 67.35 ms
87.242 ms

8 cgcil21crs.ipv6.att.net (2001:1890:ff:ffff:12:122:2:225) 61.877 ms 96.87 ms 77.492 ms

9 2001:1890:ff:ffff:12:122:22:52 (2001:1890:ff:ffff:12:122:22:78.746 ms 82.986 ms 81.59
5 ms

10 2001:1890:c02:f00::115e:4d7d (2001:1890:c02:f00::115e:4d7d) 70.623 ms 67.296 ms 43.9
87 ms

11 2607:f8b0:8289::1 (2607:f8b0:8289::1) 41.813 ms 85.667 ms 76.925 ms

12 ord38s01-in-x0e.1e100.net (2607:f8b0:4009:811::200e) 78.578 ms 53.261 ms 86.931 ms
```

15.2.6 Using tracepath

The `tracepath` command is used to trace the path to a network host, discovering MTU (maximum transmission unit) along the path. The functionality is similar to `traceroute`. It sends ICMP and UDP messages of various

sizes to find the MTU size on the path. Using UDP messages to trace the path can be useful when routers are configured to filter ICMP traffic.

To trace the path to a host, execute the following command:

```
sysadmin@localhost:~$ tracepath netdevgroup.com
1?:    [LOCALHOST]                     pmtu 1500

1:    192.168.1.1                    2.041ms
1:    192.168.1.1                    1.387ms
2:    172.72.53.1                    2.285ms
3:    sur02.englewood.co.denver.comcast.net        13.123ms
4:    edge3.Denver.Level3.net               14.657ms
5:    car2.Charlotte1.Level3.net           54.875ms
6:    rtp7600-gw-tg4-2-to-trp-crs-gw.ncren.net    58.711ms
7:    dc6500-1-10g.dcs.mcnc.org           59.248ms
13:    no reply
```

Much like the `traceroute6` command, the `tracepath6` command can also be used to determine what route communications travel between local and remote systems. The `tracepath` and `tracepath6` command use the sockets API to map out paths, which can be useful when routers are configured to filter out ICMP traffic.

15.2.7 Using ethtool

The `ethtool` utility is useful for configuring and troubleshooting network devices such as Ethernet cards and their device drivers.

```
ethtool [OPTION...] devname
```

In the example below, the `ethtool` command is used with the `-i` or `--driver` option to show the first twenty lines of driver information for Ethernet device `ens3`.

```
sysadmin@localhost:~$ sudo ethtool -i ens3 | head -n 20
MAC Registers
-------------
0x00000: CTRL (Device control register)  0x48140240
     Endian mode (buffers):              little
     Link reset:                         normal
     Set link up:                        1
     Invert Loss -Of-Signal:             no
     Receive flow control:               enabled
     Transmit flow control:              disabled
     VLAN mode:                          enabled
     Auto speed detect:                  disabled
     Speed select:                       1000MB/s
```

```
     Force speed:                          no
     Force duplex:                         no
0x00008: STATUS (Device status register) 0x80080783
     Duplex:                               full
     Link up:                              link config
   TBI mode:                               disabled
   Link speed:                             1000Mb/s
   Bus type:                               PCI
```

The `ethtool` command can also be used to display other useful troubleshooting information, such as the speed of an interface. First, an administrator could determine the name of the interface, by using the `ifconfig` or IP address command, then the following command can be executed to determine the speed of that interface:

```
sysadmin@localhost:~$ sudo ethtool ens3 | grep Speed
            Speed:1000Mb/s
```

Consider This

The `ethtool` utility may not be installed on all Linux systems.

15.2.8 Using ip neighbor

One of the most useful things to know when troubleshooting networks is what machines are on the same network segment as you. The `ip neighbor` command, part of the `iproute2` command suite, is used to add, change, or replace entries in the neighbor tables, also known as ARP (Address Resolution Protocol) cache tables. To display the ARP cache on a specific interface, use the `ip neighbor show` command with the interface name.

```
ip [OPTION...] neighbor command
```

The following command will display the contents of the local ARP cache entries (IP addresses that have been resolved to MAC addresses accessible on the network) for the `ens3` network interface:

```
root@localhost:~# ip neighbor show dev ens3
10.0.2.2 lladdr 52:55:0a:00:02:02 STALE
10.0.2.3 11addr 52:55:0a:00:02:03 STALE
```

15.2.9 Using ip link

The `ip link` command, introduced as part of the `iproute2` tools, replaces the `ifconfig` command in a previous chapter. It is useful for network troubleshooting at the Data-Link (OSI Layer 2) level. The `ip link` command is used to display and manage network interfaces.

```
ip link { COMMAND | help }
```

The `ip link` command executed by itself will display all interfaces on the network and their state:

```
root@localhost:~# ip link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_code1 state UP mode DEFAUL
T group default qlen 1000

    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
```

If you are trying to determine the status of previously configured interfaces, the `ip link show` command is one tool for doing so. To display information about a specific pre-configured interface, use the `ip link show` command followed by the interface name. The example below will display information only for the `ens3` interface:

```
root@localhost:~# ip link show ens3

2: ens3: <LOOPBACK,UP,LOWER_UP> mtu 1500 qdisc fq_code state UP mode DEFAULT group defa
ult qlen 1000

link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
```

Including the `-br (--brief)` option only prints basic information formatted in a tabular output that is easier to read.

```
sysadmin@localhost:~$ ip -br link show

lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>

eth0@if39386     UP             02:42:c0:a8:01:02  <BROADCAST,MULTICAST,UP,LOWER_UP>
```

15.2.10 Using netcat

The `netcat` utility is one of the more useful tools available for troubleshooting network issues. It is a cross-platform tool; therefore, it can be used on Windows and Mac computers as well as Linux and has many features for monitoring and debugging network connections. Some uses are transferring data, acting as a network proxy, and scanning for open ports.

```
netcat [-options] hostname port[s] [ports] ...
```

The `netcat` command can also be used in the short form, which is `nc`. To demonstrate using the `netcat` utility to find ports, the following example will scan for open ports on the local interface `192.168.1.2`, using the `netcat` command with the `-z` option, which tells it to only scan for open ports without sending any data to them, as well as with the `-v` option for verbose output.

The command below will scan ports 20 through 25 on the `192.168.1.2` interface:

```
sysadmin@localhost:~$ netcat -z -v 192.168.1.2 20-35

netcat: connect to 192.168.1.2 port 20 (tcp) failed: Connection refused
```

```
netcat: connect to 192.168.1.2 port 21 (tcp) failed: Connection refused
Connection to 192.168.1.2 22 port [tcp/ssh] succeeded!
netcat: connect to 192.168.1.2 port 23 (tcp) failed: Connection refused
netcat: connect to 192.168.1.2 port 24 (tcp) failed: Connection refused
netcat: connect to 192.168.1.2 port 25 (tcp) failed: Connection refused
```

Upon examining the results above, we can see that only port 22 (`ssh`) is open on this system.

The `netcat` command can also be used to create a communication socket between computers. Given two computers and the knowledge of one of their IP addresses, use the following commands to initiate a TCP session on port 23:

```
sysadmin@localhost:~$ sudo netcat -l 23
```

With knowledge of the IP address of the first computer, type the following on the second computer:

```
sysadmin@localhost:~$ netcat 192.168.1.2 23
```

Now, anything typed on either computer will appear on both. Congratulations, you have just simulated a Telnet session!

15.3 Troubleshooting Network Interfaces

When troubleshooting a network interface, it is important to know how to verify network connectivity systematically. By using the *Open Systems Interconnection (OSI)* model as a reference, you will be able to test interface connectivity, network addressing, gateways, routing, DNS, and more. The following sections will cover tools available for testing connectivity and modifying network interfaces.

Layer
7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

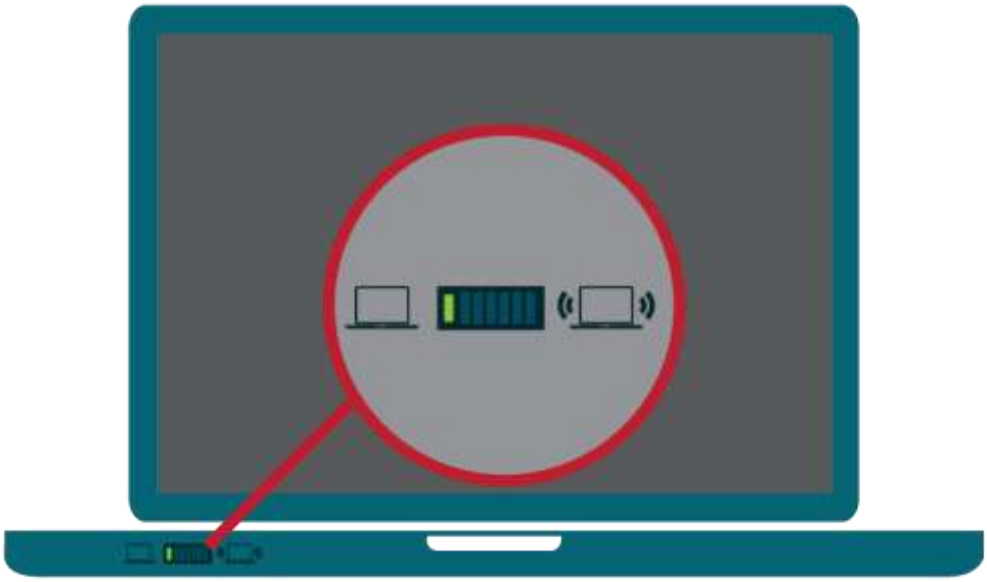
15.3.1 Physical Layer

The physical layer of the OSI model defines hardware connections and turns binary data into physical pulses (electrical, light, or radio waves).

Layer
7 Application

Layer
6 Presentation
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

The first questions that an administrator would need to answer when determining network connectivity are: “Is the device on?”, “Is my network card detected?”, and “Is the network card connected?” An example of this would be a wireless switch on a laptop.



No amount of Bash commands can turn the wireless switch on, but you can test for it by using the following command:

```
sysadmin@localhost:~$ ip link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFA
ULT group default qlen 1000

    link/ether 0c:9d:92:60:00:52 brd ff:ff:ff:ff:ff:ff

3: enx000ec6a415ca: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state D
OWN mode DEFAULT group default qlen 1000
    link/ether 00:0e:c6:a4:15:ca brd ff:ff:ff:ff:ff:ff
```

The highlighted `NO-CARRIER` message in the output above indicates that the interface is not connected to a network. In some cases, the output may not even contain the interface at all.

```
sysadmin@localhost:~$ ip link
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode
DEFAULT group default qlen 1000
    link/ether b8:27:eb:b6:76:14 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DOR
MANT group default qlen 1000
    link/ether b8:27:eb:e3:23:41 brd ff:ff:ff:ff:ff:ff
```

In this case, the `eth0` interface does not detect a carrier, but the `wlan0` interface is working fine. This is typical of a laptop that has both interfaces available. The `ifconfig` and `ip address` commands will also display this information.

It is possible that the device does not detect a network card or has not loaded a kernel driver for it. To verify that a network card is detected, use the `lspci` command.

```
sysadmin@localhost:~$ lspci | grep Ethernet

04:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Expr
ess Gigabit Ethernet Controller (rev 15)
```

In the event that the device does not have a PCI bus (Rasberry Pi™), or there is a USB to Ethernet converter installed, the `lsusb` and `lsmod` commands may be useful.

```
sysadmin@localhost:~$ lsusb

Bus 003 Device 003: ID 0b95:7720 ASIX Electronics Corp. AX88772

Bus 003 Device 002: ID 046d:c52b Logitech, Inc. Unifying Receiver

Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub

sysadmin@localhost:~$ lsmod | grep asix

asix                   45056  0

usbnet                 45056  1 asix

mii                    16384  3 r8169,usbnet,asix
```

Wireless devices may not show any network hardware using either `lspci` or `lsusb`. For instance, the Raspberry Pi™ typically has the `rfkill` module inserted in the Linux kernel. The output of this module can be searched to determine the network driver, in this case, `cfg80211`:

```
pi@localhost:~$ lsmod | grep rfkill

rfkill                 28672  6 bluetooth,cfg80211
```

After verifying that hardware switches are on and your drivers are installed correctly, if your computer is still experiencing OSI layer 1 connectivity issues, and there is no carrier, check your power, wiring, or the other end of the connection.

Consider This

Although it is beyond the scope of this course, be aware that the `iwconfig` and `iwlist` commands can be used to determine wireless connectivity. To view wireless connection details, execute the following command:

```
sysadmin@localhost:~$ iwconfig wlan0

wlan0     IEEE 802.11  ESSID:"MonitoredNetwork"

          Mode:Managed  Frequency:2.437 GHz  Access Point: 2C:30:33:90:46:0B

          Bit Rate=72.2 Mb/s   Tx-Power=31 dBm

          Retry short limit:7   RTS thr:off   Fragment thr:off
```

```
          Power Management:on
          Link Quality=59/70  Signal level=-51 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:1  Invalid misc:0   Missed beacon:0
```

15.3.2 Data-Link Layer

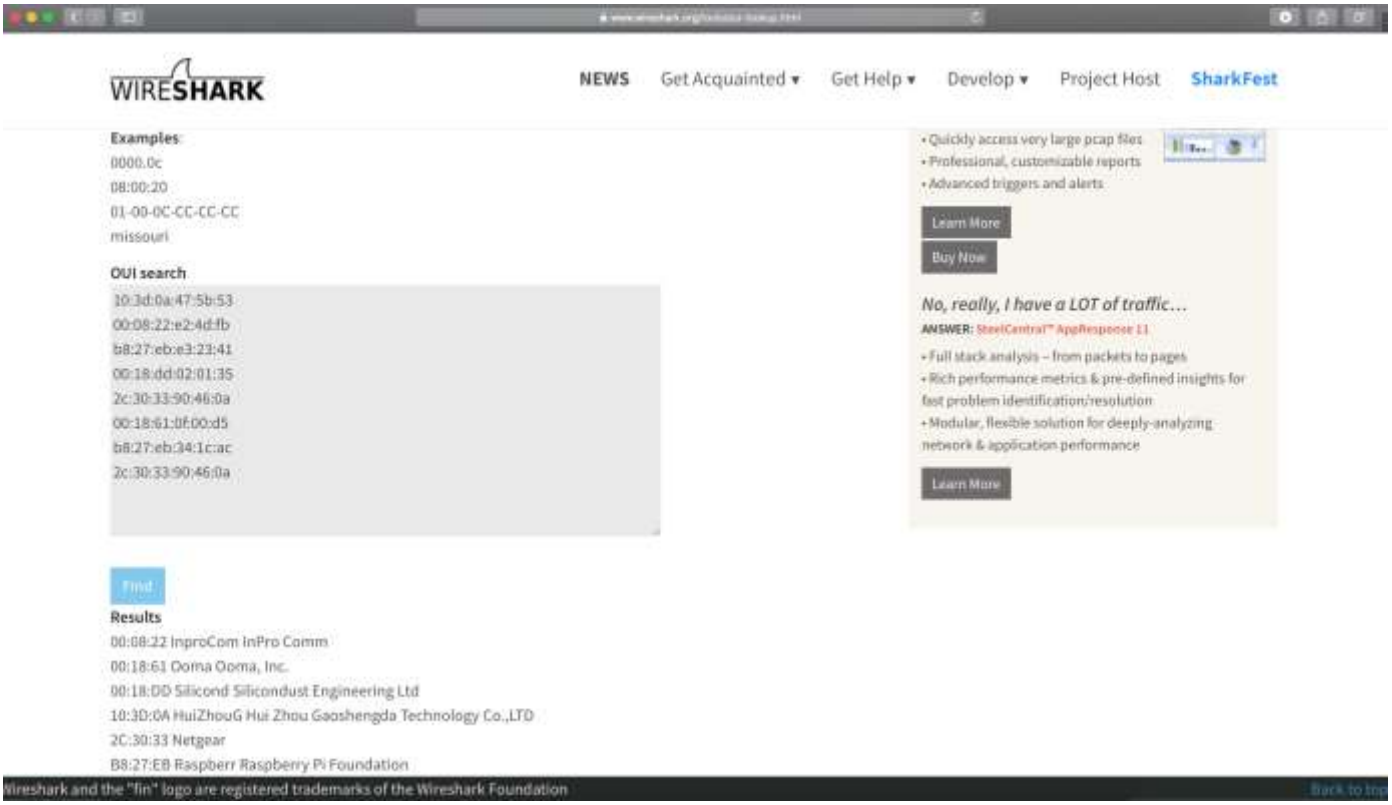
The data-link layer of the OSI model defines interface to the physical layer. It also monitors and corrects for errors that may occur in the physical layer by using a frame check sequence (FCS).

Layer
7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

The next question that an administrator might ask is, “Does this computer see any devices on the network?” As part of its function between the physical and network layers, the data-link layer keeps a table of IP address to MAC address translations. This is called the address resolution protocol (ARP) table. The `ip neighbor` command displays a list of translations:

```
sysadmin@localhost:~$ ip neighbor
192.168.0.3 dev enp4s0 lladdr 10:3d:0a:47:5b:53 STALE
192.168.0.23 dev enp4s0 lladdr 00:08:22:e2:4d:fb STALE
192.168.0.13 dev enp4s0 lladdr b8:27:eb:e3:23:41 STALE
10.10.10.156 dev enx000ec6a415ca lladdr 00:18:dd:02:01:35 STALE
192.168.0.1 dev enp4s0 lladdr 2c:30:33:90:46:0a REACHABLE
192.168.0.4 dev enp4s0 lladdr 00:18:61:0f:00:d5 STALE
192.168.0.253 dev enp4s0 lladdr b8:27:eb:34:1c:ac REACHABLE
fe80::2e30:33ff:fe90:460a dev enp4s0 lladdr 2c:30:33:90:46:0a router REACHABLE
```

From the output above, an administrator can input the MAC addresses into the Wireshark™ organizational unique identifier (OUI) tool to determine the manufacturer of the network card:



This information can be useful in determining if a particular device is found on the network.

The `ethtool` command is also useful for determining connectivity at the data-link layer along with link connection speed, duplex, and other details.

```
root@localhost:~# ethtool enp4s0 | tail
Cannot get wake-on-lan settings: Operation not permitted
    Link partner advertised FEC modes: Not reported
    Speed: 1000Mb/s
    Duplex: Full
    Port: MII
    PHYAD: 0
    Transceiver: internal
    Auto-negotiation: on
    Current message level: 0x00000033 (51)
                   drv probe ifdown ifup
    Link detected: yes
```

Consider This

The `arp -a` command performs a similar function to `ip neighbor`, as does the Windows™ `arp` command.

15.3.3 Network Layer

The network layer of the OSI model performs network routing functions, defines logical addresses, and uses a hierarchical addressing scheme. Various protocols specify packet structure and processing used to carry data from host to host.

Layer
7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

A network administrator may ask such questions as: “Does this device have an IP address?” and “Is the gateway address set on this device?” Use the `ifconfig` or `ip address` command to determine the various addresses assigned to an interface.

```
sysadmin@localhost:~$ ip address

2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group def
ault qlen 1000

    link/ether 0c:9d:92:60:00:52 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.7/24 brd 192.168.0.255 scope global dynamic noprefixroute enp4s0
       valid_lft 61212sec preferred_lft 61212sec
    inet6 2601:401:8001:45c:e9d:92ff:fe60:52/64 scope global dynamic mngtmpaddr
       valid_lft 326281sec preferred_lft 326281sec
    inet6 fe80::e9d:92ff:fe60:52/64 scope link
       valid_lft forever preferred_lft forever
```

From the highlighted output above, a network administrator can determine the MAC address of `enp4s0`, the IPv4 address and subnet mask (since /24=255.255.255), and the IPv6 address and prefix length of /64 bits.

Consider This

Though it is beyond the scope of this curriculum, be aware that the `dhclient` command requests an IP address from a DHCP server, akin to the Windows `ipconfig /renew` command.

To use the command, an administrator requires `su` or `sudo` access:

```
root@localhost:~# dhclient enp4s0 -v

Internet Systems Consortium DHCP Client 4.3.5

Copyright 2004-2016 Internet Systems Consortium.

All rights reserved.

For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp4s0/0c:9d:92:60:00:52

Sending on   LPF/enp4s0/0c:9d:92:60:00:52

Sending on   Socket/fallback

DHCPREQUEST of 192.168.0.7 on enp4s0 to 255.255.255.255 port 67 (xid=0x372c640d)

DHCPACK of 192.168.0.7 from 192.168.0.1

RTNETLINK answers: File exists

bound to 192.168.0.7 -- renewal in 42030 seconds.
```

15.3.3.1 Routing Table Testing

When checking network connectivity, ensure that your system can get to the assigned gateway. The network gateway, as defined in your network interface configuration, is the “first hop” or the first place your computer will go to when looking for resources beyond the local network. Use the `ping` command to determine connectivity to the gateway IP:

```
sysadmin@localhost:~$ ping -c 4 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=2.06 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.51 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=1.21 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=64 time=1.94 ms

--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.219/1.686/2.068/0.340 ms
```

The gateway should be configured to route network traffic out from the local network and on to the next router, which can direct your communication towards its ultimate destination. The `iproute2` suite of tools makes it possible to test the routing table with just a few commands. First, print the current list of routes available to the system with the `ip route show` command:

```
sysadmin@localhost:~$ ip route show
default via 192.168.141.1 dev wlp2s0 proto dhcp metric 600
169.254.0.0/16 dev wlp2s0 scope link metric 1000
192.168.141.0/24 dev wlp2s0 proto kernel scope link src 192.168.141.187 metric 600
```

By default, the command above will print the main routing table; other routing tables can be displayed by using the `table` parameter.

The above example shows the local interface, `wlp2s0`, as having IP address `192.168.141.187` and the default route is `192.168.141.1`. By testing that our computer can use the default route `192.168.141.187` to get to the *outside* interface at `169.254.0.0/16`, we can verify that the route from our local machine to the internet is working. The addresses to test are specified with the `get`, `to`, and `from` modifiers.

```
sysadmin@localhost:~$ ip route get to 169.254.0.0/16 from 192.168.141.187
169.254.0.0 from 192.168.141.187 dev wlp2s0 uid 1000
    cache
```

15.3.3.2 Reaching Other Networks

Once connectivity to the gateway has been reached, use the `tracepath` or `traceroute` command to determine that a system can reach beyond the network using a well known IP address.

```
sysadmin@localhost:~$ tracepath -n -m 4 1.1.1.1
 1?: [LOCALHOST]                      pmtu 1500
 1:  192.168.0.1                                           2.361ms
```

```
 1:  192.168.0.1                                          1.825ms
 2:  96.120.41.221                                       20.877ms
 3:  68.85.48.49                                         51.052ms
 4:  96.108.20.157                                       26.691ms
     Too many hops: pmtu 1500
     Resume: pmtu 1500
```

Although the number of *hops* was limited to four in this example, the output clearly demonstrates that the system can reach outside the network. If connectivity stops at the gateway, there is a network issue that may need to be resolved by the internet service provider (ISP).

15.3.3.3 Domain Name Service

Beyond network connectivity, uniform resource locator (URL) addresses need to be resolved to IP addresses using DNS. Both the `nslookup` and `host` commands can be useful when determining if DNS is working properly.

```
sysadmin@localhost:~$ nslookup netdevgroup.com
Server:         192.168.0.253
Address:    192.168.0.253#53

Non-authoritative answer:
Name:    netdevgroup.com
Address: 34.214.209.23
Name:    netdevgroup.com
Address: 64:ff9b::22d6:d117
sysadmin@localhost:~$ host netdevgroup.com
netdevgroup.com has address 34.214.209.23
netdevgroup.com has IPv6 address 64:ff9b::22d6:d117
netdevgroup.com mail is handled by 10 aspmx2.googlemail.com.
netdevgroup.com mail is handled by 10 aspmx3.googlemail.com.
netdevgroup.com mail is handled by 1 aspmx.l.google.com.
netdevgroup.com mail is handled by 5 alt1.aspmx.l.google.com.
netdevgroup.com mail is handled by 5 alt2.aspmx.l.google.com.
```

When DNS fails only for certain sites, it is possible that the DNS requests are being filtered, or that an `/etc/hosts` file is providing inaccurate information. These are both techniques used to filter out advertising and to comply with laws, such as the Children's Internet Protection Act (CIPA).

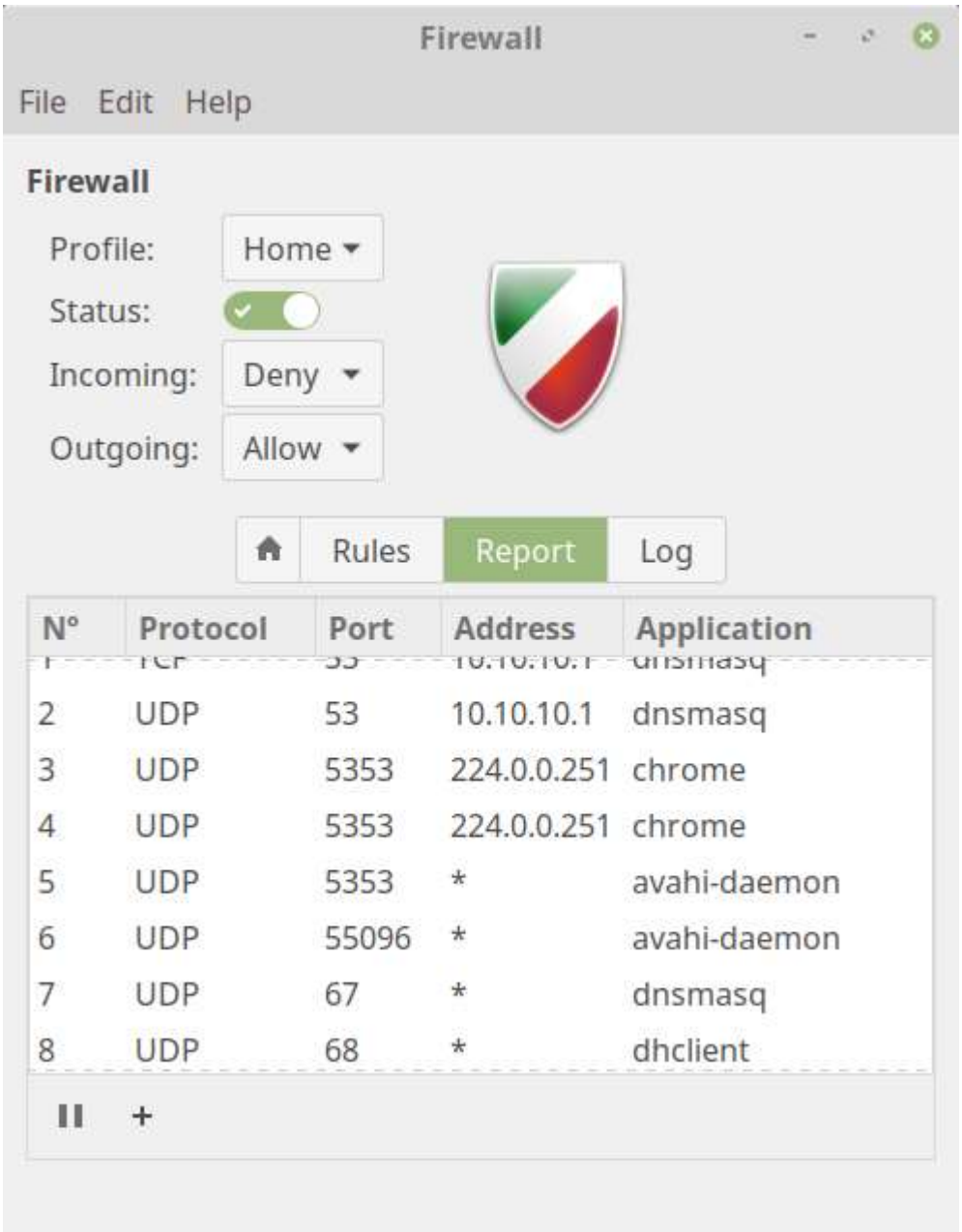
15.3.3.4 Firewalls

Firewalls are beyond the scope of the curriculum; however, they can interfere with network connectivity. Windows™ clients, in particular, do not respond to pings by default, based on their firewall settings. Linux uses IP tables to manage network traffic, which we will not cover in this curriculum. There is an easy to use tool that can

be installed called Uncomplicated Firewall. Once connectivity has been reestablished, to verify the status of the firewall, use the `ufw` command and make changes as needed.

```
root@localhost:~# ufw status
Status: inactive
root@localhost:~# ufw enable
Firewall is active and enabled on system startup
```

The `gufw` command is the graphical equivalent:



Details of the firewall logs can be found in the `/var/log/ufw.log` file.

When using a Red Hat-based distribution, `firewalld` may be blocking access. Verify that it is running using `systemctl`.

```
root@localhost:~# systemctl status firewalld | grep Active
   Active: active (running) since Sun 2019-12-01 23:27:49 EST; 8min ago
```

To stop or start `firewalld`, use the following `systemctl` options:

```
root@localhost:~# systemctl stop firewalld
```

```
root@localhost:~# systemctl disable firewalld

Removed /etc/systemd/system/multi-user.target.wants/firewalld.service.

Removed /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.

root@localhost:~# systemctl enable firewalld

Created symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service → /usr/li
b/systemd/system/firewalld.service.

Created symlink /etc/systemd/system/multi-user.target.wants/firewalld.service → /usr/li
b/systemd/system/firewalld.service.

root@localhost:~# systemctl start firewalld
```

15.3.3.5 Starting and Stopping Network Interfaces

The `ifup` and `ifdown` legacy commands are used to bring up and bring down a network interface, respectively. For example, assume there are two interfaces, `eth0` and `eth1`, configured on a system. If a test run needs to be performed using `eth0` in isolation, bring down the `eth1` device by executing the following command as the root user:

```
root@localhost:~# ifdown eth1
```

To enable the `eth1` device, execute the following command as the root user:

```
root@localhost:~# ifup eth1
```

It is also necessary to bring down a network device before assigning an IP address to the device. For example, if the IP address of the `eth1` device has to be changed, then the steps to be followed are as follows:

1. Bring down `eth1` using the `ifdown` command.
2. Use the `ifconfig` command to assign the new IP address to `eth1`.
3. Use the `ifconfig` command to view the updated IP address.
4. Bring up `eth1` again using the `ifup` command.

The `ip` command can also be used to turn the interfaces on and off.

```
root@localhost:~# ip link set enx000ec6a415ca down

root@localhost:~# ip link show enx000ec6a415ca

3: enx000ec6a415ca: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel state DOWN mode DEFAU
LT group default qlen 1000
    link/ether 00:0e:c6:a4:15:ca brd ff:ff:ff:ff:ff:ff
root@localhost:~# ip link set enx000ec6a415ca up
root@localhost:~# ip link show enx000ec6a415ca
3: enx000ec6a415ca: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0e:c6:a4:15:ca brd ff:ff:ff:ff:ff:ff
```

The highlighted portions of the output demonstrate the ability to turn interfaces on and off, at least until the next reboot!

Consider This

If the network interface is the only configured network interface, then the user should be logged in to the *system console* when the network interface needs to be brought down for any debugging purpose. If the `ifdown` command is issued through a `telnet` session, then the interface will be stopped, and the machine will no longer be available on the network unless the interface is brought up again immediately.

To be able to execute both commands successfully in a `telnet` or other remote session, the administrator could execute the following command:

```
root@localhost:~# ifdown eth1; ifup eth1
```

15.3.3.6 Deleting Network Interfaces

The network interface can be temporarily disabled by using the `ifdown` command as follows:

```
root@localhost:~# ifdown eth1
```

To make the change permanent, the configuration file for the corresponding interface should be deleted. For example, if the NIC for the `eth1` interface has been removed from the system and installed in another system, then the network configuration on the original machine should reflect this change.

Red Hat-derived

On a Red Hat-derived system, the `/etc/sysconfig/network-scripts/ifcfg-eth1` file should be moved to another directory or deleted, and the network service should be restarted using the following command:

```
root@localhost:~# /etc/init.d/network restart
```

Debian-derived

On a Debian-derived system, the `/etc/network/interfaces` file should be updated, and any references to the `eth1` interface should be commented out or removed.

The following is a sample `/etc/network/interfaces` file including `eth1` references:

```
auto lo                    #automatically activates lo
iface lo inet loopback             #lo with 127.0.0.1 address
iface enp4s0eth0 inet dhcp             #enp4s0eth0 with DHCP configuration
iface eth1 inet dhcp             #eth1 with DHCP configuration
```

Sample `/etc/network/interfaces` file with `eth1` references commented out with the hash `#` character:

```
auto lo                    #automatically activates lo
iface lo inet loopback             #lo with 127.0.0.1 address
iface enp4s0eth0 inet dhcp             #enp4s0eth0 with DHCP configuration
#iface eth1 inet dhcp             #eth1 with DHCP configuration
```

After modifying this file, the networking service should be restarted with the following command:

```
root@localhost:~# /etc/init.d/networking restart
```

15.3.4 Transport Layer

The transport layer of the OSI model performs transparent transfer of data between end users. It is responsible for error recovery and flow control and ensures complete data transfer.

Layer
7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

After establishing full network connectivity, a network administrator may want to know if a service on their server is running and if it can be reached. To find out if the service is running, we can examine the open ports using the socket statistics `ss` command.

```
sysadmin@localhost:~$ ss -tl4
State      Recv-Q Send-Q           Local Address:Port      Peer Address:Port
LISTEN     0      128             *:http-alt              *:*
LISTEN     0      128             *:ss                    *:*
sysadmin@localhost:~$ ss -tln4
State      Recv-Q Send-Q           Local Address:Port      Peer Address:Port
LISTEN     0      128             *:8080                  *:*
LISTEN     0      128             *:22                    *:*
```

We can see from the output that the server is listening for both SSH (port 22) and HTTP (port 8080) traffic. Using another computer, we can use the `netcat` command to probe the services remotely.

```
sysadmin@localhost:~$ netcat 192.168.0.13 22
SSH-2.0-OpenSSH_7.4p1 Raspbian-10+deb9u4
```

Consider This

The `ss` command replaces the deprecated `netstat` command.

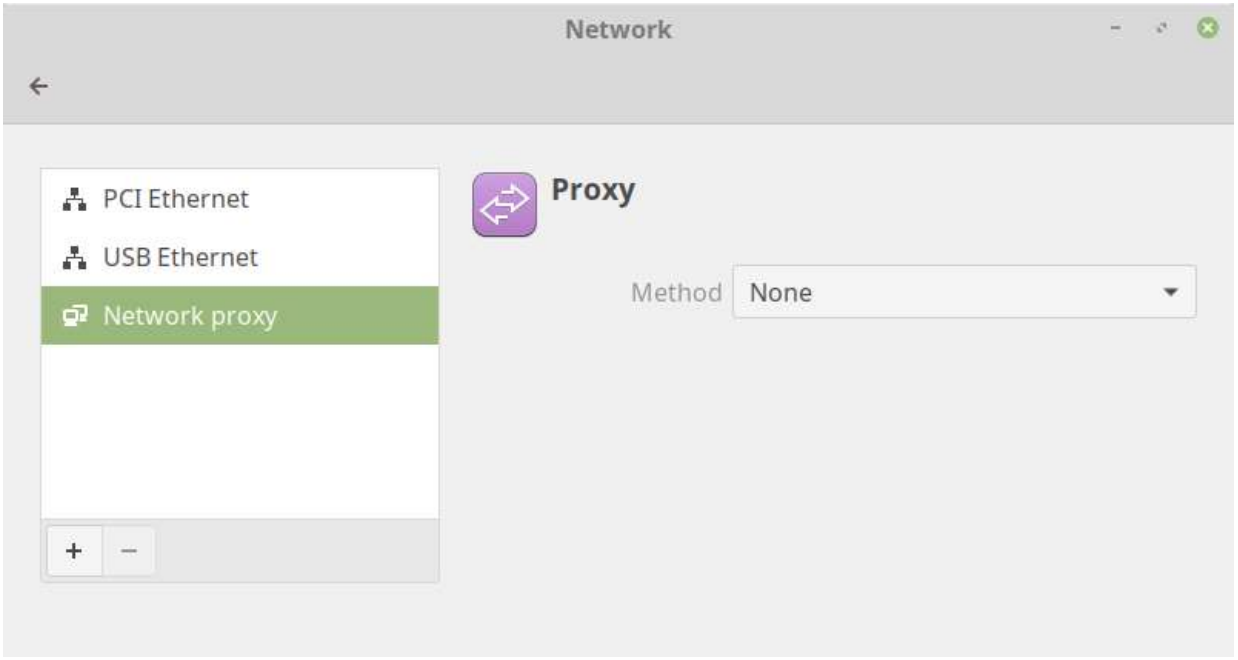
15.3.5 Remaining Layers

The session, presentation, and application layers of the OSI model are all handled by software.

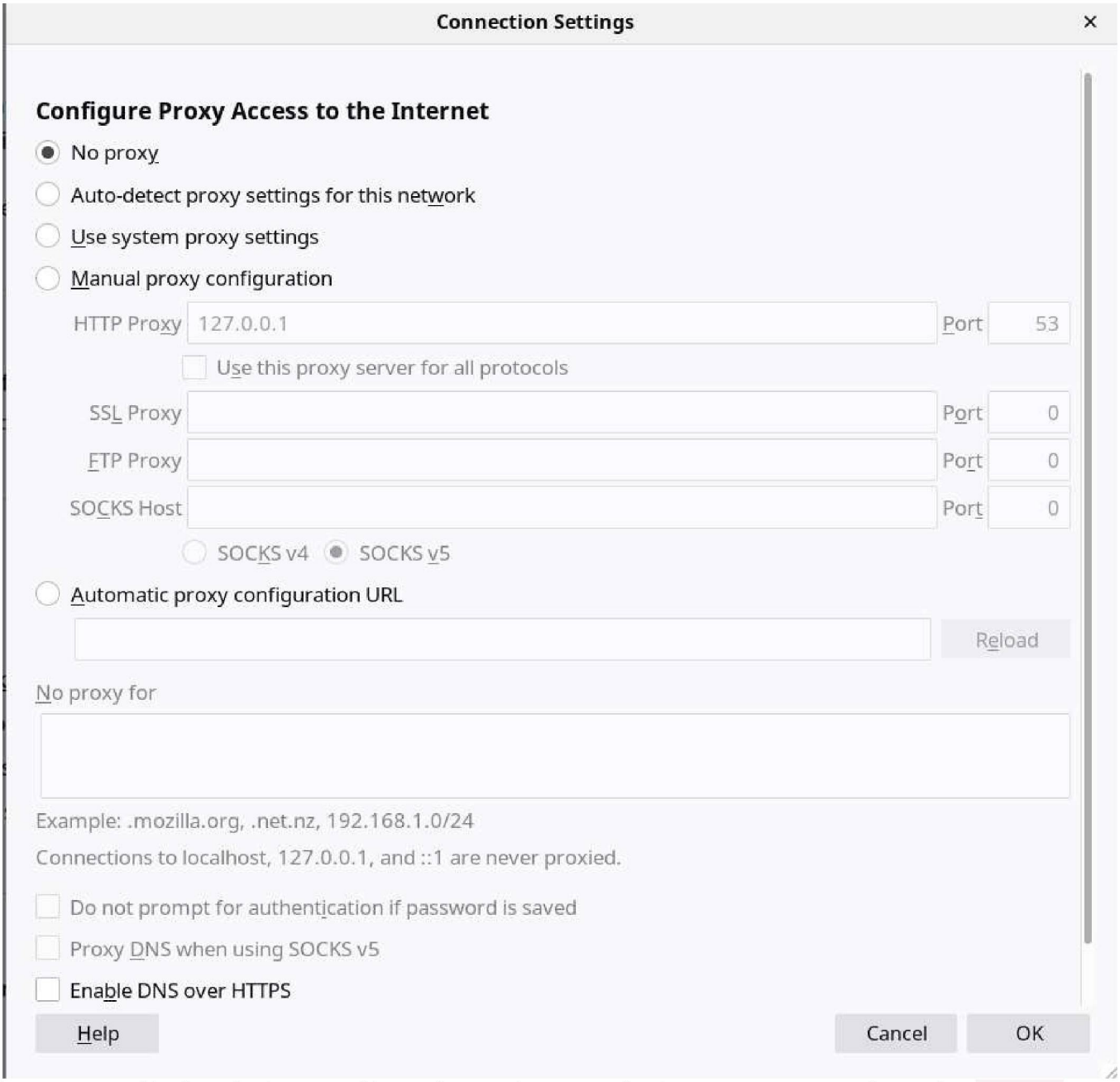
Layer
7 Application
6 Presentation

Layer
5 Session
4 Transport
3 Network
2 Data-Link
1 Physical

Once all other problems have been eliminated, a network administrator might want to examine various program settings. There could be non-standard port configurations in service configuration files or proxy settings could be incorrect for the system:



Proxy settings could also be set incorrectly in a browser:



To get a good look at the inner workings of network communications, capture some data using `tcpdump` or Wireshark—both are beyond the scope of this curriculum.

9.1 Introduction

The premier text editor for Linux and UNIX is a program called `vi`. While there are numerous editors available for Linux that range from the tiny editor `nano` to massive `emacs` editor, there are several advantages to the `vi` editor:

- The `vi` editor is available on every Linux distribution in the world. This is not true of any other editor.
- The `vi` editor can be executed both in a CLI interface and a GUI interface. While graphical editors, like `gedit` from the Gnome desktop environment or `kedit` from K desktop environment, are easier to use, they require a GUI, which servers won't always have running.
- While new features have been added to the `vi` editor, the core functions have been around for decades. This means if someone learned the `vi`editor in the 1970s, they could use a modern version without any problem. While that seems trivial, it may not seem to so trivial twenty years from now.

Consider This

The correct way to pronounce "the vi editor" is "the vee-eye editor". The letters vi stand for "**vi**sual", but it was never pronounced "vi" by the developers, but rather the letter "v" followed by the letter "i".

The original `vi` editor was written by Bill Joy, the co-founder of Sun Microsystems. Since `vi` is part of the Single UNIX Specification (SUS), it is required that conforming UNIX-based systems have it. Since the Linux Standards Base (LSB) mirrors the requirements of SUS, Linux systems that conform to LSB must also include the `vi` editor.

In reality, most Linux systems don't include the original `vi`, but an improved version of it known as `vim`, for **vi im**proved. This fact may be hidden by most Linux distributions. On some the `vi` file will link to `vim`, while on others an alias exists that will execute `vim` when the `vi` command is run:

```
[sysadmin@localhost ~]$ which vi
alias vi='vim'
        /usr/bin/vim
```

For the most part, `vim` works just like `vi`, but has additional features. For the topics that will be covered in this course, either `vi` or `vim` will work.

To get started using `vi`, simply type the command followed by the pathname to the file to edit or create:

```
sysadmin@localhost:~$ vi newfile
```

9.2 Command Mode Movement

There are three modes used in `vi`: command mode, insert mode, and ex mode. Initially, the program starts in command mode. Command mode is used to type commands, such as those used to move around a document, manipulate text, and access the other two modes. To return to command mode at any time, press the **ESC** key.

Once some text has been added into a document, to perform actions like moving the cursor, the **ESC** key needs to be pressed first to return to command mode. This seems like a lot of work, but remember that `vi` works in a terminal environment where a mouse is useless.

Movement commands in `vi` have two aspects, a motion and an optional number prefix, which indicates how many times to repeat that motion. The general format is as follows:

```
[count] motion
```

The following table summarizes the motion keys available:

Motion	Result
`h`	Left one character
`j`	Down one line
`k`	Up one line
`l`	Right one character
`w`	One word forward
`b`	One word back
`^`	Beginning of line
`$`	End of the line

Note: Since the upgrade to `vim` it is also possible to use the arrow keys `←↓↑→` instead of `hjkl` respectively.

These motions can be prefixed with a number to indicate how many times to perform the movement. For example, `5h` would move the cursor five characters to the left,`3w` would move the cursor three words to the right.

To move the cursor to a specific line number, type that line number followed by the `G` character. For example, to get to the fifth line of the file type `5G`. `1G` or `gg` can be used to go to the first line of the file, while a lone `G` will take you to the last line. To find out which line the cursor is currently on, use **CTRL-G**.

9.3 Command Mode Actions

The standard convention for editing content with word processors is to use copy, cut, and paste. The `vi` program has none of these, instead `vi` uses the following three commands:

Standard	Vi	Meaning
cut	`d`	delete
copy	`y`	yank

Standard	Vi	Meaning
paste	`P` | `p`	put

The motions learned from the previous page are used to specify where the action is to take place, always beginning with the present cursor location. Either of the following general formats for action commands is acceptable:

```
action [count] motion
[count] action motion
```

Delete

Delete removes the indicated text from the page and saves it into the buffer, the buffer being the equivalent of the "clipboard" used in Windows or Mac OSX. The following table provides some common usage examples:

Action	Result
`dd`	Delete current line
`3dd`	Delete the next three lines
`dw`	Delete the current word
`d3w`	Delete the next three words
`d4h`	Delete four characters to the left

Change

Change is very similar to delete, the text is removed and saved into the buffer, however the program is switched to insert mode to allow immediate changes to the text. The following table provides some common usage examples:

Action	Result
`cc`	Change current line
`cw`	Change current word

Action	Result
`c3w`	Change the next three words
`c5h`	Change five characters to the left

Yank

Yank places content into the buffer without deleting it. The following table provides some common usage examples:

Action	Result
`yy`	Yank current line
`3yy`	Yank the next three lines
`yw`	Yank the current word
`y$`	Yank to the end of the line

Put

Put places the text saved in the buffer either before or after the cursor position. Notice that these are the only two options, put does not use the motions like the previous action commands.

Action	Result
`p`	Put (paste) after cursor
`P`	Put before cursor

Searching in vi

Another standard function that word processors offer is find. Often, people use **CTRL+F** or look under the edit menu. The `vi` program uses search. Search is more powerful than find because it supports both literal text patterns and regular expressions.

To search forward from the current position of the cursor, use the `/` to start the search, type a search term, and then press the **Enter** key to begin the search. The cursor will move to the first match that is found.

To proceed to the next match using the same pattern, press the `n` key. To go back to a previous match, press the `N` key. If the end or the beginning of the document is reached, it will automatically wrap around to the other side of the document.

To start searching backwards from the cursor position, start by typing `?`, then type the pattern to search for matches and press the **Enter** key.

9.4 Insert Mode

Insert mode is used to add text to the document. There a few ways to enter insert mode from command mode, each differing by where the text insertion will begin. The following table covers the most common:

Input	Purpose
`a`	Enter insert mode right after the cursor
`A`	Enter insert mode at the end of the line
`i`	Enter insert mode right before the cursor
`I`	Enter insert mode at the beginning of the line
`o`	Enter insert mode on a blank line after the cursor
`O`	Enter insert mode on a blank line before the cursor

9.5 Ex Mode

Originally, the `vi` editor was called the `ex` editor. The name `vi` was the abbreviation of the **visual** command in the `ex` editor that switched the editor to "visual" mode.

In the original normal mode, the `ex` editor only allowed users to see and modify one line at a time. In the visual mode, users could see as much of the document that will fit on the screen. Since most users preferred the visual mode to the line editing mode, the `ex` program file was linked to a `vi` file, so that users could start `ex` directly in visual mode when they ran the `vi` link.

Eventually, the actual program file was renamed `vi` and the `ex` editor became a link that pointed the `vi` editor.

When the ex mode of the `vi` editor is being used, it is possible to view or change settings, as well as carry out file-related commands like opening, saving or aborting changes to a file. In order to get to the ex mode, type a `:` character in command mode. The following table lists some common actions performed in ex mode:

Input	Purpose

Input	Purpose
`:w`	Write the current file to the filesystem
`:w filename`	Save a copy of the current file as filename
`:w!`	Force writing to the current file
`:1`	Go to line number 1 or whatever number is given
`:e filename`	Open filename
`:q`	Quit if no changes made to file
`:q!`	Quit without saving changes to file

A quick analysis of the table above reveals if an exclamation mark `!` is added to a command, then attempts to force the operation. For example, imagine you make changes to a file in the `vi` editor and then try to quit with `:q`, only to discover that the "quit" command fails. The `vi` editor doesn't want to quit without saving the changes you made to a file, but you can force it to quit with the ex command `:q!`.

Consider This

While it may seem impossible, the `vi` editor can save changes to a read-only file. The command `:w!` will try to write to a file, even if it is read-only, by attempting to change the permissions on the file, perform the write to the file and then change the permissions back to what they were originally.

This means that the root user can make changes to almost any file in the `vi` editor, regardless of the permissions on the file. However, ordinary users will only be able to force writing to the files that they own. Using `vi` doesn't change the fact that regular users can't modify the permissions on file that they do not own.

Although the ex mode offers several ways to save and quit, there's also `ZZ` that is available in command mode; this is the equivalent of `:wq`. There are many more overlapping functions between ex mode and command mode. For example, ex mode can be used to navigate to any line in the document by typing `:` followed by the line number, while the `G` can be used in command mode as previously demonstrated.

Chapter 9: The vi Editor

This chapter will cover the following exam objectives:

103.8: Perform basic file editing operations using vi

Weight: 3

Description: Candidates should be able to edit text files using vi. This objective includes vi navigation, basic vi modes, inserting, editing, deleting, copying and finding text.

Key Knowledge Areas:

- Navigate a document using vi
  Section 9.2
- Use basic vi modes
  Section 9.2
- Insert, edit, delete, copy and find text
  Section 9.4

Chapter 9: The vi Editor

/, ?

This is used to search for text while in command mode. the / is used to start searching. Enter a key term and press enter to begin searching the file for the text entered. If the user would like to search backwards in the document, a ? can be used instead of the /.
Section 9.3

ZZ, :w!, :q!, :e!

These keys are used to exit the vi editor from command mode. ZZ is used to save and quit the file. It must be done for each file. :e! is used to restor the original file allow the user to start over. :w! will force the writing of the current file. :q! will exit the editor without saving changes to the current file.
Section 9.5

c, d, p, y, dd, yy

These are used to cut, copy, replace and paste text when in command mode. c is used to change a line from the the current cursor location to the end of the line with whatever the user types. d is used to cut one alphabetic word, where as dd is used to cut an entire line of text. y is used to copy one one alphabetic word, where as yy is used to copy and entire line at a time. If a number preceeds either dd or yy, this will copy that number of lines. For example if 3dd is typed this will cut 3 lines at a time.
Section 9.3

h, j, k, l

These keys are used for basic cursor movement in vi when in command mode. h moves left one character, j moves down one line, k moves up one line, and l moves right one character.
Section 9.2

i, o, a

i, o, and a are used to enter insert mode from command mode. i will allow a user to start inserting text at the current location of the cursor. o will allow a user to start inserting text a line below the current location of the cursor, and a will allow a user to insert text one postion after the current location of the cursor.
Section 9.4

vi

A screen-oriented text editor originally created for Unix operating systems. vi is also known as a modal editor in which the user must switch modes to create, edit, and search text in a file.
Section 9.1 | Section 9.2 | Section 9.3 | Section 9.5