NDG Introduction to Linux I - Chapter 2: Using the Shell

Objectives

Chapter 2: Using the Shell

This chapter will cover the following exam objectives:

103.1: Work on the command line v2

Weight: 4

Candidates should be able to interact with shells and commands using the command line. The objective assumes the Bash shell.

Key Knowledge Areas:

> Use single shell commands and one line command sequences to perform basic tasks on the command line.
> | Section 2.1 | Section 2.2 | Section 2.2.1 | Section 2.2.2 | Section 2.3 | Section 2.3.1 | Section 2.3.1.1 | Section 2.3.2 | Section 2.4 | Section 2.5 | Section 2.6 | Section 2.7 | Section 2.8

Key Terms

Quoting

Enclosing special characters in quotes will prevent the shell from interpreting special characters. Double quotes will prevent the shell from interpreting some of these special characters; single quotes prevent the shell from interpreting any special characters.

Section 2.3.1.1

echo

Echo the STRING(s) to standard output. Useful with scripts.

Section 2.3.1.1

man

An interface to the on-line reference manuals.

Section 2.9 | Section 2.9.1 | Section 2.9.2 | Section 2.9.3 | Section 2.9.4 | Section 2.9.5 | Section 2.9.6 | Section 2.9.6.1 | Section 2.9.6.2 | Section 2.9.6.3 | Section 2.9.6.4

pwd

Print the name of the current working directory.

Section 2.6

uname

Print certain system information such as kernel name, network node hostname, kernel release,kernel version, machine hardware name, processor type, hardware platform, and operating system, depending on options provided.

Section 2.5

Content

2.1 Command Line Interface

Most consumer operating systems are designed to shield the user from the “ins and outs” of the CLI. The Linux community is different in that it positively celebrates the CLI for its power, speed, and ability to accomplish a vast array of tasks with a single command line instruction.

When a user first encounters the CLI, they can find it challenging because it requires memorizing a dizzying amount of commands and their options. However, once a user has learned the structure of how commands are used, where the necessary files and directories are located, and how to navigate the hierarchy of a file system, they can be immensely productive. This capability provides more precise control, greater speed, and the ability to automate tasks more easily through scripting.

Furthermore, by learning the CLI, a user can easily be productive almost instantly on ANY flavor or distribution of Linux, reducing the amount of time needed to familiarize themselves with a system because of variations in a GUI.

2.2 Commands

What is a command? The simplest answer is that a command is a software program that when executed on the command line, performs an action on the computer.

When you consider a command using this definition, you are really considering what happens when you execute a command. When you type in a command, a process is run by the operating system that can read input, manipulate data, and produce output. From this perspective, a command runs a process on the operating system, which then causes the computer to perform a job.

However, there is another way of looking at what a command is: look at its source. The source is where the command "comes from" and there are several different sources of commands within the shell of your CLI:

- **Internal Commands**: Also called built-in commands, these commands are built-in to the shell itself. A good example is the `cd` (change directory) command as it is part of the Bash shell. When a user types the `cd` command, the Bash shell is already executing and knows how to interpret that command, requiring no additional programs to be started.
- **External Commands**: These commands are stored in files that are searched by the shell. If you type the `ls` command, then the shell searches through a predetermined list of directories to

try to find a file named ls that it can execute. These commands can also be executed by typing the complete path to the command.

- **Aliases**: An alias can override a built-in command, function, or a command that is found in a file. Aliases can be useful for creating new commands built from existing functions and commands.
- **Functions**: Functions can also be built using existing commands to either create new commands, override commands built-in to the shell or commands stored in files. Aliases and functions are normally loaded from the initialization files when the shell first starts, discussed later in this section.

2.2.1 External Commands

Commands that are stored in files can be in several forms that you should be aware of. Most commands are written in the C programming language, which is initially stored in a human-readable text file. These text source files are then compiled into computer-readable binary files, which are then distributed as the command files.

Users who are interested in seeing the source code of compiled, GPL licensed software can find it through the sites where it originated. GPL licensed code also compels distributors of the compiled binaries, such as RedHat and Debian, to make the source code available. Often it is found in the distributors' repositories.

Note

It is possible to view available software packages, binary programs that can be installed directly at the command line. Type the following command into the terminal to view the available source packages (source code that can be modified before it's compiled into binary programs) for the GNU Compiler Collection:

```
sysadmin@localhost:~$ apt-cache search gcc | grep source
gcc-4.8-source - Source of the GNU Compiler Collection
gcc-5-source - Source of the GNU Compiler Collection
gcc-6-source - Source of the GNU Compiler Collection
gcc-7-source - Source of the GNU Compiler Collection
gcc-8-source - Source of the GNU Compiler Collection
gcc-arm-none-eabi-source - GCC cross compiler for ARM Cortex-A/R/M processors (source)
```

The apt-cache command allows us to display information from the APT database cache. It is commonly used to find information about programs you wish to install and the components required to make them work.

Although there are a tremendous number of free and open source programs available, quite often the binary code you will need as a Linux administrator won't exist for the particular distribution you are running. Since open source licensing gives you access to the code for these programs, one of your tasks will be compiling, and sometimes modifying that code into executable programs that can be installed on the systems you manage. The Free Software Foundation (FSF) distributes the GNU Compiler Collection (GCC) to make this process easier. The GCC provides a compiler system (the special programs used to convert source code into usable binary programs) with front ends for many different programming languages. In fact, the FSF doesn't limit these tools to just Linux. There are versions of the GCC that run on Unix, Windows, MacOS, and many other systems including specific microcontroller environments.

Linux package management will be covered in greater detail later in the course.

Note

Command files can also contain human-readable text in the form of script files. A script file is a collection of commands that is typically executed at the command line.

The ability to create your own script files is a very powerful feature of the CLI. If you have a series of commands that you regularly find yourself typing in order to accomplish some task, then you can easily create a Bash shell script to perform these multiple commands by typing just one command: the name of your script file. You simply need to place these commands into a file and make the file executable.

2.2.2 Aliases

An alias can be used to map longer commands to shorter key sequences. When the shell sees an alias being executed, it substitutes the longer sequence before proceeding to interpret commands.

For example, the command `ls -l` is commonly aliased to `l` or `ll`. Because these smaller commands are easier to type, it becomes faster to run the `ls -l` command line.

To determine what aliases are set on the current shell use the `alias` command:

```
sysadmin@localhost:~$ alias
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'
```

The aliases from the previous examples were created by initialization files. These files are designed to make the process of creating aliases automatic.

New aliases can be created using the following format, where name is the name to be given the alias and command is the command to be executed when the alias is run.

```
alias name=command
```

For example, the `cal 2030` command displays the calendar for the year 2030. Suppose you end up running this command often. Instead of executing the full command each time, you can create an alias called `mycal` and run the alias, as demonstrated in the following graphic:

```
sysadmin@localhost:~$ alias mycal="cal 2030"
sysadmin@localhost:~$ mycal
                                2030

      January               February              March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5           1  2              1  2
 6  7  8  9 10 11 12  3  4  5  6  7  8  9  3  4  5  6  7  8  9
13 14 15 16 17 18 19 10 11 12 13 14 15 16 10 11 12 13 14 15 16
20 21 22 23 24 25 26 17 18 19 20 21 22 23 17 18 19 20 21 22 23
27 28 29 30 31       24 25 26 27 28       24 25 26 27 28 29 30
                                          31

      April                 May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6           1  2  3  4                    1
 7  8  9 10 11 12 13  5  6  7  8  9 10 11  2  3  4  5  6  7  8
14 15 16 17 18 19 20 12 13 14 15 16 17 18  9 10 11 12 13 14 15
21 22 23 24 25 26 27 19 20 21 22 23 24 25 16 17 18 19 20 21 22
28 29 30             26 27 28 29 30 31    23 24 25 26 27 28 29
                                          30

      October               November              December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                 1  2     1  2  3  4  5  6  7
 6  7  8  9 10 11 12  3  4  5  6  7  8  9  8  9 10 11 12 13 14
13 14 15 16 17 18 19 10 11 12 13 14 15 16 15 16 17 18 19 20 21
20 21 22 23 24 25 26 17 18 19 20 21 22 23 22 23 24 25 26 27 28
27 28 29 30 31       24 25 26 27 28 29 30          29 30 31
```

Aliases created this way only persist while the shell is open. Once the shell is closed, the new aliases are lost. Additionally, each shell has its own aliases, so, aliases created in one shell won't be available in a new shell that's opened.

2.3 Basic Command Syntax

To execute a command, the first step is to type the name of the command. Click in the terminal on the right. Type `ls` and hit **Enter**. The result should resemble the example below:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

Note

By itself, the `ls` command lists files and directories contained in the current working directory. At this point, you shouldn't worry too much about the output of the command, but instead, focus on understanding how to format and execute commands.

The `ls` command will be covered in greater detail later in the course.

Many commands can be used by themselves with no further input. Some commands require additional input to run correctly. This additional input comes in two forms: ***options*** and ***arguments***. Commands typically follow a simple pattern of syntax:

```
command [options…] [arguments…]
```

When typing a command that is to be executed, the first step is to type the name of the command. The name of the command is often based on what the command does or what the developer who created the command thinks will best describe the command's function.

For example, the `ls` command displays a listing of information about files. Associating the name of the command with something mnemonic for what it does may help you to remember commands more easily.

Keep in mind that every part of the command is normally case-sensitive, so `LS` is incorrect and will fail, but `ls` is correct and will succeed.

2.3.1 Specifying Arguments

```
command [options] [arguments]
```

An argument can be used to specify something for the command to act upon. Following a command, any desired arguments are allowed or are required depending on the command. For example, the *touch* command is used to create empty files or update the timestamp of existing files. It requires at least one argument to specify the file name to act upon.

```
touch FILE...
```

```
sysadmin@localhost:~$ touch newfile
```

The *ls* command, on the other hand, allows for a path and/or file name to be specified as an argument, but it's not required.

```
ls [FILE]...
```

An example of a scenario where an argument is allowed but not required is the use of the *ls* command. If the *ls* command is used without an argument, it will list the contents of the current directory:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

If the *ls* command is given the name of a directory as an argument, it will list the contents of that directory. In the following example, the /etc/ppp directory is used as an argument; the resulting output is a list of files contained in the /etc/ppp directory:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d  ip-up.d
```

The *ls* command also accepts multiple arguments. To list the contents of both the /etc/ppp and /etc/ssh directories, pass them both as arguments:

```
sysadmin@localhost:~$ ls /etc/ppp /etc/ssh
/etc/ppp:
ip-down.d  ip-up.d

/etc/ssh:
moduli                ssh_host_ecdsa_key        ssh_host_rsa_key
ssh_config            ssh_host_ecdsa_key.pub    ssh_host_rsa_key.pub
ssh_host_dsa_key      ssh_host_ed25519_key      ssh_import_id
ssh_host_dsa_key.pub  ssh_host_ed25519_key.pub  sshd_config
```

Some commands, like the *cp* command (*copy* file) and the *mv* command (*move* file), always require at least two arguments: a source file and a destination file.

```
cp SOURCE... DESTINATION
```

In the example below, we will copy the public ssh rsa key called ssh_host_rsa_key.pub which resides in the /etc/ssh directory, to the /home/sysadmin/Documents directory and verify that it is there:

```
sysadmin@localhost:~$ cp /etc/ssh/ssh_host_rsa_key.pub /home/sysadmin/Documents
sysadmin@localhost:~$ ls ~/Documents
School            alpha.txt    linux.txt     profile.txt
Work              animals.txt  longfile.txt  red.txt
adjectives.txt    food.txt     newhome.txt   spelling.txt
alpha-first.txt   hello.sh     numbers.txt   ssh_host_rsa_key.pub
alpha-second.txt  hidden.txt   os.csv        words
alpha-third.txt   letters.txt  people.csv
```

Consider This

An ssh rsa key is a file that contains an authentication credential (similar to a password) used to verify the identity of a machine that is being logged into using the ssh command. The ssh command allows users on a system to connect to another machine across a network, log in, and then perform tasks on the remote machine.

The ssh command is covered in greater detail later in the **NDG Introduction to Linux 2**.

2.3.1.1 Quoting

Arguments that contain unusual characters like spaces or non-alphanumeric characters will usually need to be quoted, either by enclosing them within double quotes or single quotes. Double quotes will prevent the shell from interpreting some of these special characters; single quotes prevent the shell from interpreting any special characters.

In most cases, single quotes are considered safer and should probably be used whenever you have an argument that contains characters that aren't alphanumeric.

To understand the importance of quotes, consider the `echo` command. The `echo` command displays text to the terminal and is used extensively in shell scripting.

```
echo [STRING]...
```

Consider the following scenario in which you want to list the contents of the current directory using the `ls` command and then use the `echo` command to display the string hello world!! on the screen.

You might first try the `echo` command without any quotes, unfortunately without success:

```
sysadmin@localhost:~$ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

sysadmin@localhost:~$ echo hello world!!

echo hello worldls

hello worldls
```

Using no quotes failed because the shell interprets the !! characters as special shell characters; in this case, they mean "replace the !! with the last command that was executed". In this case, the last command was the `ls` command, so `ls` replaced !! and then the `echo` command displayed hello worldls to the screen.

You may want to try the double quote " characters to see if they will block the interpretation (or expansion) of the exclamation !! characters. The double quotes block the expansion of some special characters, but not all of them. Unfortunately, double quotes do not block the expansion of the exclamation !! characters:

```
sysadmin@localhost:~$ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

sysadmin@localhost:~$ echo "hello world!!"

echo "hello worldls"

hello worldls
```

Using double quotes preserves the literal value of all characters that they enclose except metacharacters such as the $ dollar sign character, the ` backquote character, the \ backslash character and the ! exclamation point character. These characters, called wild cards, are symbol characters that have special meaning to the shell. Wild card characters are used for globbing and are interpreted by the shell itself before it attempts to run any command. Glob characters are useful because they allow you to specify patterns that make it easier to match file names in the command line. For example, the command `ls e??` would list all files in that directory that start with an e and have any two characters after it. However, because glob characters are interpreted differently by the shell, they need to be enclosed appropriately to be interpreted literally.

Note
Globbing will be covered in greater detail later in the course.

If you enclose text within the ' single quote characters, then all characters have their literal meaning:

```
sysadmin@localhost:~$ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

sysadmin@localhost:~$ echo 'hello world!!'

hello world!!
```

2.3.2 Options

command [**options**] [arguments]

Options can be used with commands to expand or modify the way a command behaves. If it is necessary to add options, they can be specified after the command name. Short options are specified with a hyphen - followed by a single character. Short options are how options were traditionally specified.

In the following example, the `-l` option is provided to the `ls` command, which results in a long display output:

```
sysadmin@localhost:~$ ls -l
total 0
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Desktop
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Documents
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Downloads
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Music
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Pictures
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Public
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Templates
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Videos
```

Often the character is chosen to be mnemonic for its purpose, like choosing the letter l for long or r for reverse. By default, the `ls` command prints the results in alphabetical order, so adding the `-r` option prints the results in reverse alphabetical order.

```
sysadmin@localhost:~$ ls -r
Videos  Templates  Public  Pictures  Music  Downloads  Documents  Desktop
```

In most cases, options can be used in conjunction with other options. They can be given as separate options like `-l -r` or combined like `-lr`. The combination of these two options would result in a long listing output in reverse alphabetical order:

```
sysadmin@localhost:~$ ls -l -r
total 0
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Videos
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Templates
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Public
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Pictures
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Music
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Downloads
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Documents
drwxr-xr-x 1 sysadmin sysadmin   0 Sep 18 22:25 Desktop
```

Multiple single options can be either given as separate options like `-a -l -r` or combined like `-alr`. The output of all of these examples would be the same:

```
ls -l -a -r

ls -rla

ls -a -lr
```

Generally, short options can be combined with other short options in any order. The exception to this is when an option requires an argument.

For example, the `-w` option to the `ls` command specifies the width of the output desired and therefore requires an argument. If combined with other options, the `-w` option can be specified last, followed by its argument and still be valid, as in `ls -rtw 40`, which specifies an output width of 40 characters. Otherwise, the `-w` option cannot be combined with other options and must be given separately.

```
sysadmin@localhost:~$ ls -lr -w 40

total 32

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Videos

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Templates

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Public

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Pictures

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Music

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Downloads

drwxr-xr-x 4 sysadmin sysadmin 4096 Feb 22 16:32 Documents

drwxr-xr-x 2 sysadmin sysadmin 4096 Feb 22 16:32 Desktop
```

If you are using multiple options that require arguments, don't combine them. For example, the `-T` option, which specifies tab size, also requires an argument. In order to accommodate both arguments, each option is given separately:

```
sysadmin@localhost:~$ ls -w 40 -T 12

Desktop Music   Templates

Documents  Pictures  Videos

Downloads  Public
```

Some commands support additional options that are longer than a single character. Long options for commands are preceded by a double hyphen -- and the meaning of the option is typically the name of the option, like the `--all` option, which lists all files, including hidden ones. For example:

```
sysadmin@localhost:~$ ls --all
.               .bashrc   .selected_editor  Downloads  Public
..              .cache    Desktop          Music      Templates
.bash_logout    .profile  Documents        Pictures   Videos
```

For commands that support both long and short options, execute the command using the long and short options concurrently:

```
sysadmin@localhost:~$ ls --all --reverse -t
.profile        Templates  Music        Desktop         ..
.bash_logout    Public     Downloads    .selected_editor  .cache
Videos          Pictures   Documents    .bashrc           .
```

Commands that support long options will often also support arguments that may be specified with or without an equal symbol (the output of both commands is the same):

```
ls --sort time

ls --sort=time
```

```
sysadmin@localhost:~$ ls --sort=time
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

A special option exists, the lone double hyphen `--` option, which can be used to indicate the end of all options for the command. This can be useful in some circumstances where it is unclear whether some text that follows the options should be interpreted as an additional option or as an argument to the command.

For example, if the `touch` command tries to create a file called --badname:

```
sysadmin@localhost:~$ touch --badname
touch: unrecognized option '--badname'
Try 'touch --help' for more information.
```

The command tries to interpret --badname as an option instead of an argument. However, if the lone double hyphen `--` option is placed before the file name, indicating that there are no more options, then the file name can successfully be interpreted as an argument:

```
sysadmin@localhost:~$ touch -- --badname
sysadmin@localhost:~$ ls
--badname  Documents  Music     Public  Videos
Desktop Downloads  Pictures  Templates
```

Consider This

The file name in the previous example is considered to be bad because putting hyphens in the beginning of file names, while allowed, can cause problems when trying to access the file. For example, using the `ls` command without any options to list the path of the `--badname` file above, will result in an error

```
sysadmin@localhost:~$ ls --badname
ls: unrecognized option '--badname'
Try 'ls --help' for more information.
```

To remove the --badname file from the home directory, use the rm remove command:

```
sysadmin@localhost:~$ rm -- --badname
```

The rm command will be covered in greater detail later in the course.

A third type of option exists for a select few commands. While the options used in the **AT&T** version of UNIX used a single hyphen and the GNU port of those commands used two hyphens, the **Berkeley Software Distribution (BSD)** version of UNIX used options with no hyphen at all.

This no hyphen syntax is fairly rare in most Linux distributions. A couple of notable commands that support the BSD UNIX style options are the ps and tar commands; both of these commands also support the single and double hyphen style of options.
In the terminal below, there are two similar commands, the first command is executed with a traditional UNIX style option (with single hyphens) and the second command is executed with a BSD style option (no hyphens).

```
sysadmin@localhost:~$ ps -u sysadmin
  PID TTY          TIME CMD
   79 ?        00:00:00 bash
  122 ?        00:00:00 ps
sysadmin@localhost:~$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
sysadmin    79  0.0  0.0  18176  3428 ?        S    20:23   0:00 -bash
sysadmin   120  0.0  0.0  15568  2040 ?        R+   21:26   0:00 ps u
```

2.4 Scripts

One exception to the basic command syntax used is the exec command, which takes another command to execute as an argument. What is special about the commands that are executed with exec is that they replace the currently running shell.

A common use of the exec command is in what is known as wrapper scripts. If the purpose of a script is to simply configure and launch another program, then it is known as a wrapper script.

A wrapper script often uses the following as the last line of the script to execute another program.

```
exec program
```

A script written this way avoids having a shell continue to run while the program that it launched is running, the result is that this technique saves resources (like RAM).

Although redirection of input and output to a script are discussed in another section, it should also be mentioned that the exec command can be used to cause redirection for one or more statements in a script.

2.5 Displaying System Information

The uname command displays system information. This command will output Linux by default when it is executed without any options.

```
sysadmin@localhost:~$ uname
Linux
```

The uname command is useful for several reasons, including when you need to determine the name of the computer as well as the current version of the kernel that is being used.

To display additional information about the system, you can use one of the many options available for the uname command. For example, to display all the information about the system, use the -a option with the uname command:

```
sysadmin@localhost:~$ uname -a
Linux localhost 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
```

To display information about what kernel version the system is running, use the -r option:

```
sysadmin@localhost:~$ uname -r
4.4.0-72-generic
```

The options for the uname command are summarized below:

Short Option	Long Option	Prints
-a	--all	All information
-s	--kernel-name	Kernel name
-n	--node-name	Network node name
-r	--kernel-release	Kernel release
-v	--kernel-version	Kernel version
-m	--machine	Machine hardware name
-p	--processor	Processor type or unknown
-i	--hardware-platform	Hardware platform or unknown
-o	--operating-system	Operating system
	--help	Help information
	--version	Version information

2.6 Current Directory

One of the simplest commands available is the pwd command, which is an acronym for print working directory. When executed without any options, the pwd command will display the name of the directory where the user is currently located in the file system. When a user logs into a system, they are normally placed in their home directory where files they create and control reside. As you navigate around the file system it is often helpful to know what directory you're in.

```
sysadmin@localhost:~$ pwd

/home/sysadmin
```

Notice our virtual machines employ a prompt that displays the current working directory, emphasized with the color blue. In the first prompt above, the blue tilde ~ character is equivalent to /home/sysadmin, representing the user's home directory:

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

After changing directories, the new location can also be confirmed in the new prompt by using the pwd command, and is again shown in blue:

```
sysadmin@localhost:~$ cd Documents/

sysadmin@localhost:~/Documents$ pwd

/home/sysadmin/Documents
```

To get back to the home directory after changing to a new location, use the cd change directory command without any arguments:

```
sysadmin@localhost:~/Documents$ cd

sysadmin@localhost:~$
```

2.7 Command Information

The type command displays information about a command type. For example, if you entered type ls at the command prompt, it will return that the ls command is actually an alias for the ls --color=auto command:

```
sysadmin@localhost:~$ type ls

ls is aliased to `ls --color=auto'
```

Using the -a option with the type command will return all locations of the files that contain a command; also called an executable file:

```
sysadmin@localhost:~$ type -a ls

ls is aliased to `ls --color=auto'

ls is /bin/ls
```

In the output above, the /bin/ls file path is the file location of the ls command.

This command is helpful for getting information about commands and where they reside on the system. For internal commands, like the pwd command, the type command will identify them as shell builtins:

```
sysadmin@localhost:~$ type pwd

pwd is a shell builtin
```

For external commands like the ip command, the type command will return the location of the command, in this case, the /sbin directory:

```
sysadmin@localhost:~$ type ip

ip is /sbin/ip
```

Consider This

The `/bin` directory contains executable programs needed for booting a system, commonly used commands, and other programs needed for basic system functionality.

The `/sbin` directory also contains executable programs; mainly commands and tools designed for system administration.

If a command does not behave as expected or if a command is not accessible, that should be, it can be beneficial to know where the shell is finding the command.

The which command searches for the location of a command in the system by searching the PATH variable.

```
sysadmin@localhost:~$ which ls

/bin/ls
```

Note

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

The PATH variable will be covered in greater detail later in the course.

2.8 Command Completion

A useful tool of the Bash shell is the ability to complete commands and their arguments automatically. Like many command line shells, Bash offers command line completion, where you type a few characters of a command (or its file name argument) and then press the **Tab** key. The Bash shell will complete the command (or its file name argument) automatically for you. For example, if you type ech and press **Tab**, then the shell will automatically complete the command echo for you.

There will be times when you type a character or two and press the **Tab** key, only to discover that Bash does not automatically complete the command. This will happen when you haven't typed enough characters to match only one command. However, pressing the **Tab** key a second time in this situation will display the possible completions (possible commands) available.

A good example of this would be if you typed ca and pressed **Tab**; then nothing would be displayed. If you pressed **Tab** a second time, then the possible ways to complete a command starting with ca would be shown:

```
sysadmin@localhost:~$ ca
cal                 capsh               cat                 cautious-launcher
calendar            captoinfo           catchsegv
caller              case                catman
sysadmin@localhost:~$ ca
```

Another possibility may occur when you have typed too little to match a single command name uniquely. If there are more possible matches to what you've typed than can easily be displayed, then the system will use a pager to display the output, which will allow you to scroll through the possible matches.

For example, if you just type c and press the **Tab** key twice, the system will provide you with many matches that you can scroll through:

```
c_rehash                    cksum
cal                         clear
calendar                    clear_console
caller                      cmp
capsh                       codepage
captoinfo                   col
case                        colcrt
cat                         colrm
catchsegv                   column
catman                      comm
cautious-launcher           command
cd                          compgen
cfdisk                      complete
chage                       compopt
chardet3                    compose
chardetect3                 continue
chattr                      coproc
chcon                       corelist
chcpu                       cp
chfn                        cpan
chgpasswd                   cpan5.26-x86_64-linux-gnu
chgrp                       cpgr
chmem                       cpio
--More--
```

You may not wish to scroll through all these options; in that case, press **Q** to exit the pager. In a situation like this, you should probably continue to type more characters to achieve a more refined match.

A common mistake when typing commands is to misspell the command name. Not only will you type commands faster, but you will type more accurately if you use command completion. Using the **Tab** key to complete the command automatically helps to ensure that the command is typed correctly.

Note that completion also works for arguments to commands when the arguments are file or directory names.

2.9 Getting Help

As previously mentioned, UNIX was the operating system from which the foundation of Linux was built. The developers of UNIX created help documents called man pages (short for manual page).

Referring to the man page for a command will provide you with the basic idea behind the purpose of the command, as well as details regarding the options of the command and a description of its features.

2.9.1 Viewing Man Pages

To view a man page for a command, execute the `man` command in a terminal window.

```
man command
```

For example, the following graphic shows the partial man page for the `cal` command:

```
sysadmin@localhost:~$ man cal

CAL(1)                    BSD General Commands Manual                    CAL(1)

NAME
     cal, ncal -- displays a calendar and the date of Easter

SYNOPSIS
     cal [-3hjy] [-A number] [-B number] [[month] year]
     cal [-3hj] [-A number] [-B number] -m month [year]
     ncal [-3bhjJpwySM] [-A number] [-B number] [-s country_code] [[month]
         year]
     ncal [-3bhJeoSM] [-A number] [-B number] [year]
     ncal [-CN] [-H yyyy-mm-dd] [-d yyyy-mm]

DESCRIPTION
     The cal utility displays a simple calendar in traditional format and ncal
     offers an alternative layout, more options and the date of Easter.  The
     new format is a little cramped but it makes a year fit on a 25x80 termi-
     nal.  If arguments are not specified, the current month is displayed.

     The options are as follows:

     -h      Turns off highlighting of today.
```

Consider This

It is possible to print man pages from the command line. If you wanted to send a man page to a default printer from a local machine, you would execute the following command:

```
man -t command | lp
```

The command above will not function in our virtual environment.

2.9.2 Controlling the Man Page Display

The man command uses a pager to display documents. Typically, this pager is the less command, but on some distributions, it may be the more command. Both are very similar in how they perform and will be discussed in more detail in a later chapter.

To view the various movement commands that are available, use the **H** key or **Shift+H** while viewing a man page. This will display a help page.

Note

If you are working on a Linux distribution that uses the more command as a pager, your output will be different from the partial example shown here.

Pagers will be covered in greater detail later in the course.

```
                  SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
  d  ^D             *  Forward  one half-window (and set half-window to N).
  u  ^U             *  Backward one half-window (and set half-window to N).

  ESC-)  RightArrow *  Left  one half screen width (or N positions).
  ESC-(  LeftArrow  *  Right one half screen width (or N positions).
  F                    Forward forever; like "tail -f".
  r  ^R  ^L            Repaint screen.
  R                    Repaint screen, discarding buffered input.
        ---------------------------------------------------
        Default "window" is the screen height.
        Default "half-window" is half of the screen height.
 ---------------------------------------------------------------------------
```

To exit the SUMMARY OF LESS COMMANDS, type **Q**.

If your distribution uses the less command, you might be a bit overwhelmed with the large number of "commands" that are available. The following table provides a summary of the more useful commands:

Command	Function
`Return` (or **`Enter`**)	Go down one line
`Space`	Go down one page
`/term`	Search for *term*
`n`	Find next search item
`1G`	Go to the beginning of the page
`G`	Go to the end of the page
`h`	Display help
`q`	Quit man page

2.9.3 Sections Within Man Pages

Each man page is broken into sections. Each section is designed to provide specific information about a command. While there are common sections that you will see in most man pages, some developers also create sections that you will only see in a specific man page.

The following table describes some of the more common sections that you will find in man pages:

NAME

Provides the name of the command and a very brief description.

```
NAME
       ls - list directory contents
```

SYNOPSIS

A brief summary of the command or function's interface. A summary of how the command line syntax of the program looks.

```
SYNOPSIS
       ls [OPTION]... [FILE]...
```

DESCRIPTION

Provides a more detailed description of the command.

```
DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.
```

OPTIONS

Lists the options for the command as well as a description of how they are used. Often this information is found in the DESCRIPTION section and not in a separate OPTIONS section.

```
-a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              scale sizes by SIZE before printing them; e.g., '--block-size=M'
              prints sizes in units of 1,048,576 bytes; see SIZE format below
```

FILES

Lists the files that are associated with the command as well as a description of how they are used. These files may be used to configure the command's more advanced features. Often this information is found in the DESCRIPTION section and not in a separate FILES section.

AUTHOR

Provides the name of the person who created the man page and (sometimes) how to contact the person.

```
AUTHOR
       Written by Richard M. Stallman and David MacKenzie.
```

REPORTING BUGS

Provides details on how to report problems with the command.

```
REPORTING BUGS
       GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
       Report ls translation bugs to <http://translationproject.org/team/>
```

COPYRIGHT

Provides basic copyright information.

```
COPYRIGHT
       Copyright (C) 2017 Free Software Foundation, Inc.  License GPLv3+:  GNU
       GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
       This  is  free  software:  you  are free to change and redistribute it.
       There is NO WARRANTY, to the extent permitted by law.
```

SEE ALSO

Provides you with an idea of where you can find additional information. This often includes other commands that are related to this command.

```
SEE ALSO
       Full documentation at: <http://www.gnu.org/software/coreutils/ls>
       or available locally via: info '(coreutils) ls invocation'
```

2.9.4 Man Pages Synopsis

The SYNOPSIS section of a man page can be difficult to understand, but it is a valuable resource since it provides a concise example of how to use the command. For example, consider an example SYNOPSIS for the `cal` command:

```
SYNOPSIS
     cal [-3hjy] [-A number] [-B number] [[month] year]
```

The square brackets [ ] are used to indicate that this feature is not required to run the command. For example, [-3hjy] means you can use the options `-3`, `-h`, `-j`, `-y`, but none of these options are required for the `cal` command to function properly.

The second set of square brackets [-A number] allows you to specify the number of months to be added to the end of the display.

The third set of square brackets in the [-B number] allows you to specify the number of months to be added to the beginning of the display.

The fourth set of square brackets in the [[month] year] demonstrates another feature; it means that you can specify a year by itself, but if you specify a month you must also specify a year.

Another component of the SYNOPSIS that might cause some confusion can be seen in the SYNOPSIS of the date command:

```
SYNOPSIS

       date [OPTION]... [+FORMAT]

       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

In this SYNOPSIS there are two syntaxes for the date command. The first one is used to display the date on the system while the second is used to set the date.

The ellipses ... following [OPTION], indicate that [OPTION] may be repeated.

Additionally, the [-u|--utc|--universal] notation means that you can either use the -u option or the --utc option, or the --universal option. Typically, this means that all three options really do the same thing, but sometimes this format (use of the | character) is used to indicate that the options can't be used in combination, like a logical or.

2.9.5 Searching Within a Man Page

In order to search a man page for a term, type the / character followed by the term and hit the **Enter** key. The program will search from the current location down towards the bottom of the page to try to locate and highlight the term.

If the term is not found, or you have reached the end of the matches, then the program will report Pattern not found (press Return). If a match is found and you want to move to the next match of the term, press **N**. To return to a previous match of the term, press **Shift+N**.

> **Note**
>
> If you haven't already, notice here that pager commands are viewed on the final line of the screen.

2.9.6 Man Page Sections

Until now, we have been displaying man pages for commands. However, sometimes configuration files also have man pages. Configuration files (sometimes called system files) contain information that is used to store information about the operating system or services.

Additionally, there are several different types of commands (user commands, system commands, and administration commands) as well as other features that require documentation, such as libraries and kernel components.

As a result, there are thousands of man pages on a typical Linux distribution. To organize all of these man pages, the pages are categorized by sections, much like each individual man page is broken into sections.

By default, there are nine sections of man pages:

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions, e.g. /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)
9. Kernel routines [non-standard]

When you use the `man` command, it searches each of these sections in order until it finds the first match. For example, if you execute the command `man cal`, the first section (Executable programs or shell commands) is searched for a man page called cal. If not found, then the second section is searched. If no man page is found after searching all sections, you will receive an error message:

```
sysadmin@localhost:~$ man zed
No manual entry for zed
```

Consider This

Typically, there are many more sections than the standard nine. Custom software designers make man pages, using non-standard section names, for example "3p".

2.9.6.1 Determining Which Section

To determine which section a specific man page belongs to, look at the numeric value on the first line of the output of the man page. For example, if you execute the command `man cal`, you will see that the `cal` command belongs to the first section of man pages:

```
sysadmin@localhost:~$ man cal
```

```
CAL(1)                    BSD General Commands Manual                    CAL(1)
```

2.9.6.2 Specifying a Section

In some cases, you will need to specify the section in order to display the correct man page. This is necessary because sometimes there will be man pages with the same name in different sections.

For example, there is a command called `passwd` that allows you to change your password. There is also a file called passwd that stores account information. Both the command and the file have a man page.

The `passwd` command is a user command, so the following command will display the man page for the `passwd` command, located in the first section, by default:

```
sysadmin@localhost:~$ man passwd
PASSWD(1)                         User Commands                         PASSWD(1)

NAME
       passwd - change user password
```

To specify a different section, provide the number of the section as the first argument of the `man` command. For example, the command `man 5 passwd` will look for the passwd man page in section 5 only:

```
PASSWD(5)                 File Formats and Conversions                 PASSWD(5)

NAME
       passwd - the password file
```

2.9.6.3 Searching by Name

Sometimes it isn't clear which section a man page is stored in. In cases like this, you can search for a man page by name.

The `-f` option to the `man` command will display man pages that match, or partially match, a specific name and provide a brief description of each man page:

```
sysadmin@localhost:~$ man -f passwd
passwd (1ssl)        - compute password hashes
passwd (1)           - change user password
passwd (5)           - the password file
```

Note that on most Linux distributions, the `whatis` command does the same thing as `man -f`. On those distributions, both will produce the same output.

```
sysadmin@localhost:~$ whatis passwd
passwd (1ssl)        - compute password hashes
passwd (1)           - change user password
passwd (5)           - the password file
```

2.9.6.4 Searching by Keyword

Unfortunately, you won't always remember the exact name of the man page that you want to view. In these cases, you can search for man pages that match a keyword by using the `-k` option to the `man` command.

For example, what if you knew you wanted a man page that displays how to change your password, but you didn't remember the exact name? You could run the command `man -k password`:

```
sysadmin@localhost:~$ man -k password
chage (1)            - change user password expiry information
chgpasswd (8)        - update group passwords in batch mode
chpasswd (8)         - update passwords in batch mode
cpgr (8)             - copy with locking the given file to the password or gr...
cppw (8)             - copy with locking the given file to the password or gr…
```

Warning

When you use this option, you may end up with a large amount of output. The preceding command, for example, provided over 60 results.

Recall that there are thousands of man pages, so when you search for a keyword, be as specific as possible. Using a generic word, such as "the" could result in hundreds or even thousands of results.

Note that on most Linux distributions, the `apropos` command does the same thing as `man -k`. On those distributions, both will produce the same output.

> Consider This
>
> Want to learn more about man pages? Execute the following command:

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