LINUX

THE ULTIMATE BIBLE TO LEARN LINUX COMMAND LINE, LINUX ADMINISTRATION AND SHELL SCRIPTING STEP BY STEP

DAVID A. WILLIAMS

LINUX COMMAND LINE

1

DAVID A. WILLS

LINUX ADMINISTRATION

2

DAVID A. WILLS

LINUX FOR BEGINNERS

2 BOOKS IN 1

The Ultimate Bible to Learn Linux Command Line, Linux Administration and Shell Scripting Step by Step

David A. Williams

TABLE OF CONTENTS

Linux Administration
The Ultimate Beginners Guide to Learn Linux Step by Step

Linux Command

Beginners Guide to Learn Linux Commands and Shell Scripting

Linux Administration

The Ultimate Beginners Guide to Learn Linux Step by Step

David A. Williams

Introduction

The term Linux refers to a kernel or an operating system that was developed by Linus Torvals along with a few other contributors. The first time it was released publicly in September 1991. In a world where Microsoft was charging consumers for an operating system like Windows, the advantage of Linux was that it was an open-source software meaning that programmers had the option to customize it, create their own operating system out of it and use it as per their requirement. The Linux operating system code was written mostly in the C programming language.

There are literally hundreds of operating systems available today, which use the Linux kernel and the most popular among them are Ubuntu, Debian, Fedora and Knoppix. This is not the end of the list as new operating systems come up almost every year, which use the kernel from the original Linux system.

Linux was a milestone in computing and technology and most of the mobile phones, web servers, personal computers, cloud-servers, and supercomputers today are powered by Linux. The job profile of Linux System Administration refers to that of maintaining operations on a Linux based system and ensuring maximum uptime from the system, in short, making sure that the system is consumed in the most optimum possible way. In the modern world, most of your devices run on a Linux powered server or are associated with a Linux system in some way or the other because of its high stability and open source nature.

Owing to this, the job a Linux Administrator revolves around the following.

- Linux File System
- Managing the superuser on the Linux system known as Root
- Command Line using the Bash Shell
- Managing users, file and directories

You can think of it as maintaining your own personal computer, which you would do at home, but on a larger scale, in this case for an entire

organization. Linux system administration is a critical requirement for organizations in the modern world and, therefore, there is a huge demand in the market for this profile. The job description may vary from one organization to another, but the fundamentals of Linux Administration remain the same for every organization. The following responsibilities are associated with the profile of a system administrator.

- Maintaining regular backups of the data of all users on the system.
- Analyzing logs for errors to continuously improve the system.
- Maintain and enhance the existing tools for users of the system and for the Linux environment.
- Detecting problems and solving them, which range from user login issues to disaster recovery.
- Troubleshoot other problems on the system.

Perhaps, the most important skill to be found in a system admin is performing under a lot of load and stress. A system admin usually works a seven day week wherein he or she is on call two days a week and has to come online as soon as there is an issue with the system and must be quick to resolve it so that the system goes online immediately. A system or a server, if kept down for a long time can lead to losses worth thousands of dollars for an organization. As an example, take the website of the shopping giant Amazon. If the Amazon website went down even for one hour, all sales would suffer, leading to huge losses in revenue. This is where a system admin steps in and saves the day. The role of a system admin is nothing short of a superhero who saves an organization whenever the need arises.

Chapter One: Installing Red Hat Enterprise Linux on your Computer

In this chapter, we will learn how to install Red Hat Enterprise Linux 7 on your computer. It is advisable to have a Linux operating system installed before you begin with all the other chapters as doing activities mentioned in the upcoming chapters will only give you hands on experience on a Linux system, which will help you understand Linux Administration better.

We will first need to create installation media to install Red Hat Enterprise Linux 7 on your computer. Depending on whether your current computer has Windows as an operating system or Mac OS X as an operating system, the steps to create the installation media for Red Hat Enterprise Linux 7 will differ. Let us go through the steps to create installation media on both Windows and Mac OS X one by one.

Before creating the installation media, you will need to download a Red Hat Enterprise Linux 7 installation image, which you will later load onto the installation media. You can download the installation ISO from for Red Hat Enterprise Linux 7 from the following URL.

https://developers.redhat.com/products/rhel/download

Make sure that you download from the link that says DVD ISO as that is the image that contains the complete installation for Red Hat Enterprise Linux 7.

Note: You will need to register on the website before you can download the Red Hat Enterprise Linux 7 installation ISO. You can do this when you are prompted to create an account to download the ISO.

Once you have downloaded the ISO required for installation, you can proceed with creating the installation media. Installation media can be a DVD, SD card or a USB drive. The following steps will help you create installation media on a USB drive on Windows and Mac OS X.

Creating Red Hat Enterprise Linux 7 Installation Media on Windows

The process of creating installation media for Red Hat Enterprise Linux 7 on a Windows machine is simple and straightforward. There are many tools available on the Internet using, which you can create the installation media by writing the Red Hat Enterprise Linux 7 ISO image to the USB drive. We have tested many tools and we recommend that you create the installation media using a tool called Fedora Media Writer. It can be downloaded from the following URL.

https://github.com/FedoraQt/MediaWriter/releases

You can download the .EXE file available on this GitHub repository to install it on your Windows machine. You can then proceed with the steps given below to complete the creation of installation media for Red Hat Enterprise Linux 7.

1. Download Fedora Media Writer from

https://github.com/FedoraQt/MediaWriter/releases

Make sure that you download from the link that says DVD ISO as that is the image that contains the complete installation for Red Hat Enterprise Linux 7

2. By this time you would have already downloaded the ISO for Red Hat Enterprise Linux 7 from their website as mentioned earlier. If not, you can download it from https://developers.redhat.com/products/rhel/download
3. Plug the USB drive into your computer, which you intend to use as the bootable media for installing Red Hat Enterprise Linux 7
4. Launch Fedora Media Writer

5. When the main window pops up, click on Custom Image and then navigate to the path where you have downloaded the Red Hat Enterprise Linux 7 ISO and select it
6. You will see a drop-down menu, which lets you select the drive on, which you want to write the ISO. You should be able to see the USB drive that you have plugged in. Select the USB drive
7. If your USB drive is not available in the drop-down, plug it out and plug it in again and launch Fedora Media Write again
8. Click on Write To Disk, which will initiate the media creation. Kindly ensure that the USB drive is plugged in until the process is complete. The installation media creation time will depend on the size of the ISO image and the USB drive's write speed.
9. You will see a popup saying Complete when the media creation process is finished.
10. You can now click on the Safely Remove Hardware icon in the taskbar of your Windows computer and physically remove the USB drive.

And with that, you have successfully created installation media for Red Hat Enterprise Linux 7 on a USB drive using your Windows machine.

Creating Red Hat Enterprise Linux 7 Installation Media on Mac OS X

The installation media for Red Hat Enterprise Linux 7 is created using the command line utility on Mac OS X. We will be using the *dd* command to create the installation media. The steps to create the installation media are as follow.

1. Download Fedora Media Writer from

https://github.com/FedoraQt/MediaWriter/releases

Make sure that you download from the link that says DVD ISO as that is the image that contains the complete installation for Red Hat Enterprise Linux 7

2. By this time you would have already downloaded the ISO for Red Hat Enterprise Linux 7 from their website as mentioned earlier. If not, you can download it from https://developers.redhat.com/products/rhel/download
3. Plug the USB drive into your computer, which you intend to use as the bootable media for installing Red Hat Enterprise Linux 7
4. Launch the terminal
5. First, we need to identify the path for the USB drive using the *diskutil list* command. The path of the device will have a format, which will be like */dev/disknumber* where *number* denotes the number of the disk. In the example, we are using in the next step, the *disknumber* is *disk2*
6. You should see something listed like
   /dev/disk2
   #: Type Name Size Identifier
   0: Windows_NTFS SanDisk USB 8.0 GB disk2s1

You can identify your USB drive by comparing the type, name and size columns, which should give you a fair idea if it is the USB drive you plugged in or not.

7. To make sure that you have the correct device name for your USB

drive, you can unmount the USB drive by using the command *diskutil /dev/disknumber*.
Note that you will be prompted with an error message that states **failed to unmount** if you try to unmount a system drive.

8. The *dd* command of the Linux terminal can be used to start writing the image to the USB drive.
   $ sudo dd if=/pathtoISO of=/dev/rdisknumber bs=1m>

We are using *rdisknumber* instead of just *disknumber* because it is a faster method to write the ISO to the USB drive.

Note that you will need to replace the *pathtoISO* part with the actual path of the downloaded Red Hat Enterprise Linux 7 ISO on your Mac OS X machine.

9. This will now begin the writing process. You will need to wait until the writing completes, as you will not see anything until the process actually completes.
   The status of the writing progress can be displayed on the terminal by sending the Ctrl+t input from your keyboard.

load: 2.01 cmd: dd 3675 uninterruptible 0.00u 1.92s

114+0 records in

114+0 records out

116451517 bytes transferred in 125.843451 secs (1015426 bytes/sec)

10. The installation media creation time will depend on the size of the ISO image and the USB drive’s write speed.

11. Once the transfer of data is complete, the USB drive can be unplugged.

This is it. You have successfully created installation media for Red Hat Enterprise Linux 7 on a USB drive using your Mac OS X machine.

Installing Red Hat Enterprise Linux 7

There are various options for installing Red Hat Enterprise Linux 7 but the steps we have provided in this section will help you install Red Hat Enterprise Linux 7 with minimal software and no graphical interface. You do not need to panic because of the absence of a graphical interface as most tasks that need to be performed as a Linux System Administrator are done using the command line.

Let us begin with the Red Hat Enterprise Linux 7 installation using the USB media drive that you have created in the previous section.

1. Plug in the USB drive into your computer's USB port and start your computer. You should make sure that you have enabled USB boot in your computer's BIOS settings
2. Once the computer boots up, you should be able to see a list of bootable devices and one of them would be your USB drive titled Red Hat Enterprise Linux 7
3. Once the system is up, you will get an option to choose your language. Click on Continue after you have selected the required language
4. You will now be presented with the Installation Summary screen where you can customize the installation of Red Hat Enterprise Linux 7 as per your needs. Select the Date and Time to configure the locale for your system using the world map that is provided and then click on Done
5. On the next screen, choose the language for your system and for your keyboard. We would recommend that you use the English language
6. The installation source for your Red Hat Enterprise Linux 7 will be the USB drive primarily, but you can add other sources for the repositories by specifying locations on the Internet on your local network using protocols such as FTP, HTTP, HTTPS, etc. Once you have defined all your sources, click on Done. You can just leave it on the default source if you do not have any other sources to be used
7. Now, you can select the software that has to be installed along with the operating system. As we have discussed, we will only be installing

software that is essential. To do this, select on Minimal Install along with Compatibility Libraries Add-ons and click on Done

8. On the next step, we will configure partitions for the system. Click Installation Destination and then choose LVM scheme for partition, which will give optimized management for the disk space and then click on “Click here to create them automatically”

9. You will be presented with the default partition scheme, which you can edit as per your requirements. As we are going to use the Red Hat Enterprise Linux 7 operating system to learn server administration, you can use the essential partition scheme as given below

/boot partition, which should have disk space of 500 MB and should be a non-LVM partition
/root partition with a minimum disk space of 20 GB and an LVM partition
/home partition, which should be an LVM partition
/var partition with a minimum disk space of 20 GB and an LVM partition

The filesystem that you need to use is XFS, which is the world’s most advanced filesystem right now. Once you have specified the edits for the partitions, click on Update Settings and then click Done followed by Accept Changes, which will apply your edits to the system.

10. This is the final step before initiating the Red Hat Enterprise Linux 7 installation. You need to setup the network. Select Network and Hostname, which will allow you to specify a hostname for your system. You can use a short hostname for the system or use a Fully Qualified Domain Name (FQDN)

11. Once you have specified the network, you can toggle the Ethernet button on top to ON to switch on the network. If you have a router, which has a DHCP that allots IPs to devices, your IP will now be visible. If not, you can click on the Configure button to manually specify the settings for your network

12. Once you have configured the Ethernet settings, click on Done and you will be presented with the Installation screen. You will get one

last chance to review your installation settings before the setup starts writing files to your disk. After reviewing, click on the Begin Installation option to start the installation.

13. The installation will now start writing files to your hard disk. Meanwhile, you will be prompted to create a new user for your system along with a password. Click on Root Password and supply a password that is strong and has at least 8 characters with a combination of the alphabet and numbers. Click on Done

14. Next, you can create a new user other than root and provide the credentials for this user. We recommend that you make this new user a system administrator who will have privileges similar to root user by using the *sudo* command. So check the box, which says “Make this user administrator” and then click Done. Give the installation some time to complete

15. Once the installation is complete, you will see a message confirming the same and that you can now reboot the system and get ready to use it

Voila! You can now unplug your installation media, which is the USB drive and restart your computer. You will be presented with the login screen for a minimal installation of Red Hat Enterprise Linux 7. You can either use the root user or the additional user that you created to login into the system.

Chapter Two: The Linux Command Line

In this chapter, we will learn about the Linux Command Line and how to use it to perform various tasks on your Linux operating system. By the end of this chapter, you will be well versed with the basic commands that are used on the command line and also commands that are used to manage files and folders on a Linux system.

The Bash Shell

The command line utility available on Linux operating systems is known as BASH, which is short for Bourne-Again Shell. The command line utility is basically an interface, which allows you to give instructions in text-mode to the computer and its operating system. There are different types of shell interfaces that are available in the many Unix0-ike systems that have been developed over the years, but Red Hat Enterprise Linux 7 uses the bash shell. The bash shell is an evolved and improved version of the former popular shell known as Bourne Shell.

The bash shell displays a string, which implies that it is waiting for a command to be input by the user. This string that you see is called the shell prompt. For a user that is not a root user, the shell prompt ends with a $ symbol.

[student@desktop ~]$

If the root user is logged into the system, the shell prompt ends with a # symbol. The change in this symbol quickly lets the user know if he is logged in as root or a regular user so that mistakes and accidents can be avoided.

[root@desktop ~]#

If you need an easy comparison, the bash shell in Linux operating systems is similar to the command prompt utility that is available in Windows operating systems, nut you can say that the scripting language used in bash is far more sophisticated compared to the scripting that can be done in command prompt. Bash is also comparable to the power shell utility, which was made available from Windows 7 and Windows Server 2008 R2. The bash shell has been called as a very powerful tool for administration by many professionals. You can automate a lot of tasks on your Linux system using the scripting language that is provided by the bash shell. Additionally, the bash shell also has the option to perform many other tasks, which are complicated or even impossible if tried via a graphical interface.

The bash shell is accessed through a tool known as the terminal. The input to the terminal is your keyboard and the output device is your display monitor. Linux operating systems also provide something known as virtual consoles,

which can be used to access the bash shell as well. So, you will have multiple virtual consoles on the base physical console and each virtual console can act as a separate terminal. Also, you can use each virtual console to create login sessions for different users.

Basics of Shell

There are three parts to commands that are entered on the shell prompt.

1. *Command that you want to run*
2. *Options that will define the behavior of the command*
3. *Arguments, which are the command's targets*

The command basically defines that program that you want to execute. The options that follow the command can be none, one or more. The options govern the behavior of the command and define what the command will do. Options are usually used with one dash or two dashes. This is done so that we can differentiate options from arguments. Example: -a or --all

Arguments also follow the command on the command line and can be one or more like options. Arguments indicate the target on, which a command is supposed to operate.

Let us take an example command

usermod -L John

The command in this example is *usermod*

The option is -*L*

The argument is *John*

What this command does is locks the password of the user John on the system.

To be effective with the commands on the command line, it is essential for a user to know what options can be used with a particular command. If you run the --*help* option with any command, you will get a set of options, which can be passed with that particular command. So, it's not really necessary that you know all the options that are to be used by all the commands by heart. The list will also tell you what each option does.

The use of statements can sometimes seem very difficult and complicated to read. Once you get used to certain conventions, reading the statement

becomes much easier.

- Options are surrounded by square brackets [ ]
- If a command is followed by … it specifies the arbitrary length list of items belonging to that type
- If there are multiple items and if the pipe separates them | it implies that you can specify only one of them
- Variables are represented using text, which is in angle brackets <>. So if you see <filename>, you have to replace it with the filename that you wish to pass

For example, check the below command

[student@desktop ~]$ date --help

date [OPTION]... [+FORMAT]

This indicates that the command date takes the options represented by [OPTION]... with another option [FORMAT], which will be prefixed with the + symbol.

Executing Commands on the Bash Shell

The Bourne Again Shell, known as bash, does the job of interpreting commands that are input by the user on the shell prompt. We have already learned how the string that you type in at shell prompt is divided into three parts, command, options and arguments. Every word that you type into the shell is separated using blank space. The program that is already installed on the system is defined by every command that you type, and every command has options and arguments that are associated with it.

When you have typed a command on the shell prompt along with the options and arguments and are ready to run it, you can press the Enter key on the keyboard, which will execute that command. You will then see the output of that command displayed on the terminal and when the output completes, you will be presented with the shell prompt again, which is an indication that the previous command has been executed successfully. If you wish to type more than one command on a single line, you can use a semicolon to separate the commands.

Let us go through some simple commands that are used on a daily routine on the command line in Red Hat Enterprise Linux 7 and other Linux based operating systems.

The ***date*** command will display the current date and time of the system. You can also use this command to set the time of the system. If you are passing an argument with the + sign for the date command, it indicates the format in, which you want the date to be displayed on the output.

[student@desktop ~]$ date

Sat Aug 5 08:15:30 GMT 2019

[student@desktop ~]$ date +%R

08:15

[student@desktop ~]$ date +%x

08/05/2019

The ***passwd*** command can be used to change the password of a user. You

will, however, need to specify the original password for the given user before you can set a new password. The command requires you to specify a strong password, which makes it necessary to include letter belonging to lowercase and uppercase, numbers, and symbols. You also need to ensure that the password being specified is not a word in the dictionary. The root user has the option to change the password of any other user on the system.

[student@desktop ~]$ passwd

Changing password for user student.

Changing password for student.

(current) UNIX password: type old password here

New password: Specify new password here

Retype new password: Type new password again

passwd: all authentication tokens updated successfully.

File types and extensions are not specified in a Linux operating system. The ***file*** command can scan any file and tell you the kind of file it is. The file you want to classify needs to be passed as an argument to the file command.

[student@desktop ~]$ file /etc/passwd

/etc/passwd: ASCII text

If you are passing a folder/directory as an argument to the file command, it will tell you that it is a directory.

[student@desktop ~]$ file /home

/home: directory

The next set of commands are ***head*** and ***tail,*** *which* print the first ten lines and the last ten lines of a file respectively. Both these commands can be combined with the option -*n*, *which* can be used to specify the number of lines that you want to be displayed.

[student@desktop ~]$ head /etc/passwd

This will print the first 10 lines that are there in the passwd file.

[student@desktop ~]$ tail - n /etc/passwd

This will print the last 3 lines that are there in the passwd file.

The ***wc*** command is used to count lines, words and characters in a file that is passed as an argument. It supports options such as -l, -w, -c, which stands for lines, words, and characters respectively.

[student@desktop ~]$ wc /etc/passwd

30 75 2009 /etc/passwd

This shows that there are 30 lines, 75 words and 2009 characters in the file passwd.

If you pass the -l, -c, -w options along with the wc command, it will only display the count of lines, words or characters based on, which option you have passed.

The ***history*** command displays all the commands that you have typed previously along with the command number. You can use the ! mark along with the command number to expand what was typed in that command along with the output.

[student@desktop ~]$ history

1 clear

2 who

3 pwd

[student@desktop ~]$!3

/home/student

This shows that we used !3 to expand the pwd command, which shows the present working directory for, which the output was /home/student, which was the home directory of the student user.

You can use the arrow keys to navigate through the output given by the history command. Up Arrow will take you to the commands on top and Down Arrow will take you to the commands below. Using the Right Arrow

and the Left Arrow keys, you can move on the current command and edit it.

Shortcuts to Edit the Command Line

There is an editing feature available on the command line in bash when you are interacting with bash. This helps you to move around the current command that you are typing so that you can make edits. We have already seen how we can use the arrow keys with the history command to move through commands. The following list will help you use edits when you are working on a particular command.

Ctrl+a This will take your cursor to the start of the command line

Ctrl+e This will take your cursor to the end of the command line

Ctrl+u Clear the line from where the cursor is to the start of the command line

Ctrl+k Clear the line from where the cursor is to the end of the command line

Ctrl+Left Arrow Take the cursor to the start of the previous word

Ctrl+Right Arrow Take the cursor to the start of the next word

Ctrl+r Search for patterns in the history list of commands used

Managing Files using Commands on the Command Line

In this section, we will learn to execute commands that are needed to manage files and directories in Linux. You will learn how to move, copy, create, delete and organize files and directories using commands on the bash shell prompt.

Linux File System Hierarchy

Let us first understand the hierarchy of the file system in a Linux operating system. The Linux file system has a tree, which has directories forming the file system hierarchy. However, this is an inverted tree as the root starts from the top of the hierarchy in Linux and then the branches, which are directories and subdirectories extend below the root.

The root directory denoted by / sits at the top of the file system hierarchy. Although the root is denoted by / do note that the slash character / is also used to separate directories and filenames. As an example, take etc, which is a subdirectory of the root and is denoted by /etc. Similarly, if there is a file named logs in the /etc directory, we will reference it as /etc/logs

The subdirectories of root / are standard and store files based on their specific purpose. For example, files under /boot will contain files, which are needed to execute the boot process of the Linux operating system.

We will now go through the important directories in the Red Hat Enterprise Linux 7 file system hierarchy.

/usr

The shared libraries, which are installed with the software, are stored in this directory. It has subdirectories further, which are important such as

/usr/bin: User command files

/usr/sbin: Commands used in system administration

/usr/local: Files of software that has been customized locally

/etc

System configuration files are stored here.

/var

Files that change dynamically such as databases, log files, etc. are stored in this directory.

/run

This directory contains files that were created during runtime and were created since the last boot. The files created here get recreated on the next boot.

/home

This is the home directory for all users that are created on the system. The users get to store their personal data and configurations under their specific home path.

/root

This is the home directory of the root user who is also the superuser of the system.

/tmp

This is a directory used to store temporary files. Files, which are older than 10 days and have not been accessed or modified automatically get deleted. There is another directory at /var/tmp where file not accessed or modified in 30 days get deleted automatically.

/boot

The files required to start the boot process are stored here.

/dev

Contains files, which reference to hardware devices on the system.

Let us now learn how we can locate and access files on the Linux file system. In this section, we will learn to use absolute file paths, change the directory in, which we are currently working and learn commands, which will help us to determine the location and contents of a directory.

Absolute Paths

An absolute path indicates a name that is fully qualified. It begins from the root at /, which is followed by each subdirectory that is traversed through until you reach a specific file. There is an absolute path defined for every file that exists on the Linux file system, which can be identified using a simple rule. If the first character of the path is / then it implies an absolute path name.

For example, system messages are logged in a file called messages. The absolute path name for the messages file is /var/log/messages

There are relative path names in place as well since absolute path names can sometimes become very long to type.

When you first login to the Linux system, you are automatically placed in the location of your home directory. There is an initial directory in place for system processes as well. After that, both users and system processes navigate through other directories based on their requirement. The current location of a user or a system process is known as a working directory or current working directory.

Relative Paths

A relative path refers to the unique path, which is required to reach a file but this path is only from the current directory that you are in and does not start with root /

The rule, as mentioned, is simple. If the path does not begin with a / symbol, it is a relative path for the file.

For example, if you are already working in the /var directory, then the relative path for the messages file for you will be log/messages

Navigating through paths

You can use the **pwd** command, which will output the path of the current working directory you are in. Once you know this information, you can use it to traverse to different directories using relative paths.

The **ls** command when used with a directory or directory path specifies the content of that particular directory. If you do not specify a directory with it, it will list the content of the directory that you are currently working in.

[student@desktop ~]$ pwd

/home/student

[student@desktop ~]$ ls

Documents Music Downloads Pictures

You can use the **cd** command to change directories. For example, if you are in the directory /home/student and you want to go to the directory Music, you can use relative path to get there. However, if you would want to get into the Downloads directory, you will then need to use the absolute path.

[student@desktop ~]$ cd Music

[student@desktop Music]$ pwd

/home/student/Music

[student@desktop ~]$ cd /home/student/Downloads

[student@desktop Downloads]$ pwd

/home/student/Downloads

As you can see, the shell prompt will display the last part of the current directory that you are working in for convenience. If you are in /home/student/Music only Music is displayed at the shell prompt.

The **cd** command is used to navigate through directories. If you use cd with an argument of a relative path or an absolute path, your current working directory will switch to that path's end directory. You can also use **cd -, which** will take you back to the previous directory you were working in and if you use it again, you will be back to the directory that you switched from. If you just keep using it, you will keep alternating between two fixed directories.

The **touch** command is another simple command, which if applied to an existing file, updates its timestamp to the current timestamp without actually modifying the content of the file. If used by passing an argument for a filename that does not exist, it will create an empty file with that filename. This allows new users to touch and create files for practice since these files

will not harm the system in any manner.

[student@desktop ~]$ touch Documents/test.txt

This will create an empty text file called test in the Documents subdirectory.

The **ls** command lists down the files and directories of the directory that you are in. If you pass the path of a directory with the ls command, it will list down the files and directories in that path.

The ls command can be used with options, which further help listing down files.

-l will list down all the files with timestamps and permissions of the files and directories. It will also list down the owner and the group of that file.

-a can be combined with -l to additionally list down hidden files and directories.

-R used with the above two options will list down files and directories recursively for all subdirectories.

When you list down the files and directories, you will see that the first two listing are a . and ..

. denotes the current directory and .. denotes the parent directory and these are present on the system in every directory.

File Management using Command Line

When we talk about file management, we are discussing how to create, delete, copy, and move files. The same set of actions can be performed on directories as well. It is very important to know your current working directory so that when you are managing files and directories, you know if you need to specify relative paths or absolute paths.

Let us go through a few commands that can be used for file management.

Activity	Single Source	Multiple Source
Copy file	cp file1 file2	cp file1 file2 file3 dir

Move file	mv file1 file2	mv file1 file2 file3 dir
Delete file	rm file1	rm -f file1 file2 file3
Create directory	mkdir dir	mkdir -p par1/par2/dir
Copy directory	cp -r dir1 dir2	cp -r dir1 dir2 dir3 dir4
Move directory	mv dir1 dir2	mv dir1 dir2 dir3 dir4
Delete directory	rm -r dir1	rm -rf dir1 dir2 dir3

mv file1 file2

The result of this is a rename

cp -r dir1 dir2

rm -r dir1

-r is used to process the source directory recursively

mv dir1 dir2

If dir2 exists, the content of dir1 will be moved to dir2. If it does not exist, dir1 will be renamed to dir2

cp file_1 file_2 file_3 directory

mv file_1 file_2 file_3 directory

cp -r directory1 directory2 directory3 directory4

mv directory1 directory2 directory3 directory4

Make sure that the last argument to be passed in the command should be a directory

rm -f file_1 file_2 file_3

rm -rf directory1 directory2 directory3

Kindly use this carefully as the -f uses a force option will delete everything without any confirmation prompt

mkdir -p par1/par2/dir

Kindly use this carefully as using -p will keep creating directories starting from the parent and irrespective of typing errors

Let us now go through these file management commands one by one to see how they work.

Directory Creation

You can use the **mkdir** command to create a directory, or even subdirectories. If a filename already exists or if the parent directory that you have specified does not exist, you will see errors generated. When you use the mkdir command along with the option **-p** it will create all parent directories along the path that do not exist. You need to be careful while using the -p option or you will end up creating directories that are not required, as it does not check for any spelling errors.

[student@desktop ~]$ mkdir Drawer

[student@desktop ~]$ ls

Drawer

As you can see, the new directory called Drawer is created in the home directory of the user.

[student@desktop ~]$ mkdir -p Thesis/Chapter1

[student@desktop ~]$ ls -R

Thesis thesis_chapter1

As you can see, a new directory called Thesis was created and its subdirectory called Chapter1 was created at the same time.

Copy Files

The **cp** command is used to copy files. You can copy one or more files and syntax gives you the option to copy a file in the same directory or even copy a file in one directory to another file in another directory.

Note: The file that you are specifying at the destination should be a unique file. If you specify an existing file, you will end up overwriting the content of that existing file.

[student@desktop ~]$ cd Documents

[student@desktop Documents]$ cp one.txt two.txt

This will copy content of one.txt to two.txt

Similarly, you can copy from the current directory to a file in another directory as shown below.

[student@desktop ~]$ cp one.txt Documents/two.txt

This will copy content of one.txt in home directory to two.txt in the Documents sub-directory.

Move Files

The **mv** command can be used for two operations. If you are using it in the same directory, it will rename the file. If you are specifying another directory, it will move the file to the destination directory. The content of the files are retained if you rename or move the file. Also note that if the file size is huge, it may take longer to move from one directory to another.

[student@desktop ~]$ ls

Hello.txt

[student@desktop ~]$ mv Hello.txt Bye.txt

[student@desktop ~]$ ls

Bye.txt

You can see that in this example, since we were operating in the same directory, the mv command just renamed the Hello.txt file to Bye.txt

[student@desktop ~]$ ls

Hello.txt

[student@desktop ~]$ mv Hello.txt Documents

[student@desktop ~]$ ls Documents

Hello.txt

In this example, you will see that mv command moved Hello.txt file from the home directory to the Documents subdirectory.

Deleting Files and Directories

The **rm** command can be used to delete files. To delete directories, you will need to use **rm -r, which** will delete a directory, subdirectories and files in the whole path.

Note: There is nothing such as trash or recycle bin while operating from the command line. If you delete something, it is deleted permanently.

[student@desktop ~]$ ls

File1.txt Directory1

[student@desktop ~]$ rm file1

[student@desktop ~]$ ls

Directory1

[student@desktop ~]$rm -r Directory1

[student@desktop ~]$ ls

[student@desktop ~]$

The above command has demonstrated how you can delete a file and a directory using the rm command and using the -r option.

Also note that there is a **rmdir** command, which can be used to delete a directory as well provided that the directory is completely empty.

File Globbing

Management of files can become hectic if you are dealing with a large number of files. To overcome this hurdle, Linux offers a feature called file globbing, also known as path name expansion. It uses a technique called pattern matching, also known as wildcards, which is achieved with the use of meta-characters that expand and allow operations to be performed on multiple files at the same time.

*Pattern matching using * and ?*

[student@desktop ~]$ ls a*

alpha apple

As you can see, the * is used as a wildcard to match a pattern, which has any files that begin with a.

You can try this pattern placing the start at different locations such as *a and *a*

[student@desktop ~]$ ls ???

are tab map

The number of question marks defines the number of characters in a file name. The output will show all files, which have a filename of 3 characters. You can try it with additional question marks as well.

Tilde Expansion

The tilde symbol ~ followed by a slash / will point to the active user's home directory. It can be followed by a directory name and can be used with commands such as cd and ls

[student@desktop ~]$ ls ~/Documents

file.txt hello.txt

[student@desktop ~]$ cd Documents

[student@desktop Documents]$

[student@desktop Documents]$ cd ~/

[student@desktop ~]$

Brace Expansion

The brace command is used when files have something in command, and you do not want to type it repetitively. It can be used with strings, which are comma-separated, and with expressions, which have a sequential nature. You can have nested braces as well.

[student@desktop ~]$ echo {sunday, monday, tuesday}.log

sunday.log monday.logtuesday.log

[student@desktop ~]$ echo file{1..3}.txt

file1.txt file2.txt file3.txt

[student@desktop ~]$echo file{a{1, 2}, b, c}.txt

filea1.txt filea2.txt fileb.txt filec.txt

This is where we end this chapter and you have learned how to manage files and directories using simple commands using the command line interface on a Linux system. Most of these commands are generic to any flavor of a Linux operating system and not just Red Hat Enterprise Linux 7.

Chapter Three: Managing Text Files

In this chapter, we will learn how to create, view and edit text files on a Linux operating system. We will also learn how to redirect the output from one text file to another text file. We will learn to edit existing text files on the command shell prompt using a tool known as ‘Vim’.

Redirecting the Output from a File to another File or Program

In this section, we will be discussing the terms such as standard input, standard output and standard error. We will further learn how to redirect outputs from a file to another file and redirect the output from a file to another program.

Standard Input, Standard Output, and Standard Error

When a program or a process is in running state, it will take inputs from somewhere and then write the output to a file or display it on the screen. When you are using a terminal on the Linux operating system, the input is usually taken from the keyboard and the output is sent to be displayed on the screen.

A process uses a number of channels known as file descriptors, which take some input and send some output. There are at least three file descriptors in every process.

1. Standard Input, also known as Channel 0, which takes inputs from the keyboard
2. Standard Output, also known as Channel 1, which sends outputs to the screen
3. Standard error, also known as Channel 3, which sends error messages

Let us go through the channel names for file descriptors

Number	Channel Name	Description	Default Connection	Usage
0	stdin	Standard input	Keyboard	Read only
1	stdout	Standard output	Terminal	Write only
2	stderror	Standard error	Terminal	Write only

3	filename	Other files	None	Read and/or Write

Redirecting Output to a File

The input/output redirection is used to replace default output destinations with file names or other devices. The output of a command, which usually is redirected to be displayed on the terminal screen, can be redirected to a file or a device or can even be discarded with the use of redirection.

When you redirect the standard output **stdout**, it will not appear as output on the terminal screen. This does not mean that error messages **stderr** will not appear on the screen if you only redirect standard output **stdout**. If you are redirecting the standard output to a file that does not exist, it will get created in the process. If the file already exists and you use a redirect that is not an append redirect, the existing file will be overwritten. Redirecting the standard output to /dev/null will discard all the output as it is redirected to an empty file.

Let us go through the output operators that are used for redirection.

>file

This will redirect the standard output stdout to the file and will overwrite any previous content in the file.

>>file

This will redirect the standard output stdout to the file and will append to any previous content in the file.

2>file

This will redirect the standard error stderr to the file and will overwrite any previous content in the file.

2>/dev/null

This will discard the standard error stderr by redirecting it to /dev/null

>file 2>&1

&>file

This will redirect the standard output stdout and standard error stderr to overwrite the same file

>>file 2>&1

&>>file

This will redirect the standard output stdout and standard error stderr to append to the same file

Note: The order of operator is very important as changing the order can lead to a complete change in the redirection.

For example, *>file 2>&1* will redirect the standard output stdout to the file and then redirect the standard error stderr to the same file.

If you were to change the order to *2>&1 >file* it will redirect the standard error to the default output place, which is the terminal screen and only redirect the standard output to the file.

Because of this confusion, many users prefer using the alternative operators *&>file* and *&>>file* for, which merge standard output and standard error and then redirect them to the file.

Let us quickly go through some example to understand output redirection better. There are many day to day system administration tasks that can be performed using the technique of output redirection.

1. Saving the timestamp in a file for future reference

   [student@desktop ~]$ date > ~/time

   This will output the current timestamp and redirect it to the file named time in the student's home directory.

2. Copy the last 100 lines from a log file and save it in another file

   [student@desktop ~]$ tail -n 100 /var/log/messages > ~/logs

This will copy the last hundred lines from the messages log file and save it in the logs file in the student's home directory.

3. Concatenation contents of 3 files into a single file

   [student@desktop ~]$ cat file1 file2 file3 > ~/onefile

   This will concatenate contents of file1, file2 and file3 and save it in a single file called onefile in the user's home directory.

4. List the hidden directories in the home directory and save the file names in a file

   [student@desktop ~]$ ls -a > ~/hiddenfiles

   This will list the hidden directories in the user's home directory and save the directory names in the file called hiddenfiles in the user's home directory.

5. Append the out of an echo command to an existing file

   [student@desktop ~]$ echo "Hello World" >> ~/file

   This will append the string Hello World to the file in the user's home directory.

6. Direct the standard output to one file and standard error to another file

   [student@desktop ~]$ find /etc -name passwd > ~/output 2> ~/error

   This will redirect the output to the output file and the errors to the error file in the student's home directory.

7. Discarding the error messages

   [student@desktop ~]$ find /etc -name passwd > ~/output 2> /dev/null

This will redirect the errors to /dev/null, which is an empty file and discard it.

8. Redirect standard output and standard error together in one file

   [student@desktop ~]$ find /etc -name passwd > &> ~/onefile

   This will redirect the standard output and standard error to the file onefile in the student's home directory.

Using the Pipeline

A pipeline is an operator, which separates one or more commands by using a pipe operator |

The pipe basically takes the standard output of the first command and passes it as standard input to the second command.

The output will keep passing through various commands, which are separated using the pipe and only the final output will be displayed on the terminal. We can visualize it as a flow of data through a pipeline from one process to another process and that data is being modified on its way by every command it passes through.

Let us go through some examples of the pipeline, which are useful in day to day tasks of system administration.

[student@desktop ~]$ ls -l /var/log | less

This will list the files and directories located at /var/log and display it on the terminal one screen at a time because of the less command.
[student@desktop ~]$ ls | wc -l
This command will pass the output through the pipe and the wc -l command will count the number of lines in the output and just display the number of lines and not the actual output of the ls command.

Pipelines, Redirections and the Tee command

As already discussed, when you are using the pipeline, the pipeline makes

sure that all the data is processed and passed through every command and only the final output it displayed on the terminal screen. This means that if you were to use output redirection before a pipeline, the output would be redirected to the file and not to the next command in the pipeline.

```
[student@desktop ~]$ ls > ~/file | less
```

In this example, the output of the ls command was redirected to the file in the student's home directory and never passed to the less command and the final output never appeared on the terminal screen.

This is exactly where the **tee** command comes into the picture to help you work around such scenarios. If you are using a pipeline and use the tee command in it, tee will copy its standard input to standard output and at the same time will also redirect the standard output to the specified files named as arguments to the command. If you visualize data as water flowing through a pipe, tee command will be the T joint of that data, which will direct the output in two directions.

Let us go through some examples, which will help us understand how to use the tee command with pipelines.

```
[student@desktop ~]$ ls -l | tee ~/Documents/output | less
```

Using tee in this pipeline, firstly redirects the output of the ls command to the file at Documents/output in the student's home directory. After that, it also feeds the output of the ls command to the pipe as input to the less command, which is then displayed on the terminal screen.

```
[student@desktop ~]$ ls -l | less | tee ~/Documents/output
```

In this case, we see that tee is used at the end of the command. What this does is it displays the output of the commands in the pipeline on the terminal screen and saves the same output to the file at Documents/output in the student's home directory as well.

Note: You can redirect standard error while using the pipe, but you will not be able to use the merging operators &> and &>>

Therefore, if you wish to redirect both standard output and standard error while using the pipe, you will have to use it in the following manner.

[student@desktop ~]$ find -name / passwd 2>&1 | less

Using the Shell Prompt to Edit Text Files

In this section, we will learn how to use the shell prompt to create new files and edit existing files. We will also learn about Vim, which is a very popular editor used to edit files from the shell prompt.

Using Vim to edit files

One of the most interesting things about Linux is that it is designed and developed in a way where all information is stored in text-based files. There are two types of text files, which are used in linux. Flat files in, which text is stored in rows containing similar information, which you will find in the /etc directory, and Extensible Markup Language(XML) file, which have text stored using tags, which you will find in the /etc and /usr directories. The biggest advantage of text files is that they can be transferred from one system or platform to another without having the need to convert them, and they can also be viewed and edited using simple text editors.

Vim is a the most popular text editor across all Linux flavors and is an improved version of the previously popular vi editor. Vim can be configured as per the needs of a user and includes features like color formatting, split screen editing, and highlighting text for editing.

Vim works in 4 modes, which are used for different purposes.

1. Edit mode
2. Command mode
3. Visual edit mode
4. Extended command mode

When you first launch Vim, it will open in the command mode. The command mode is useful for navigation, cut and paste jobs, and other tasks related to manipulation of text. To enter the other modes of Vim, you need to enter single keystrokes, which are specific to every mode.

- If you use the i keystroke in the command mode, you will be taken to the insert mode, which lets you edit the text file. All content you type in

the insert mode becomes a part of the file. You can return to the command mode by pressing the Esc key on the keyboard

- If you use the v keystroke in the command mode, you will be taken to the visual mode, where you can manipulate text by selecting multiple characters. You can use V and Ctrl+V to select multiple lines and multiple blocks, respectively. You can exit the visual mode by using the same keystroke that is v, V or Ctrl+V.
- The : keystroke takes you to the extended command mode, which lets you save the content that you typed to the file and exit the vim editor.

There are more keystrokes that are available in vim for advanced tasks related to text editing. Although it is known to be one of the best text editors in Linux in the world, it can get overwhelming for new users. We will go through the minimum keystrokes that are essential for anyone using vim to accomplish editing tasks in Linux.

Let us go through the steps given below to get some hands-on experience of vim for new users.

1. Open a file on the shell prompt using the command *vim filename*.
2. Repeat the text entry cycle given below as many times as possible until you get used to it.
   Use the arrow keys on the keyboard to position the cursor
   Press **i** to go to insert mode
   Enter some text of your choice
   You can use **u** to undo steps taken on the current line that you are editing
   Press the **Esc** key on the keyboard to return to the command mode
3. Repeat the following cycle, which teaches you to delete text, as many times as possible until you get the hang of it.
   Position the cursor using the arrow keys on the keyboard
   Delete a selection of text by pressing **x** on the keyboard
   You can use **u** to undo steps taken on the current line that you are editing
4. You can use the following keystrokes next to save, edit, write or

discard the file.
Enter **:w** to save/write the changes you have made to the file and stay in the command mode
Enter **:wq** to save/write the changes to the file and exit Vim
Enter **:q** to discard the changes that you have made to the file and exit Vim

Rearranging the Existing Content in Vim

The tasks of copy and paste are known as yank and put in Vim. This can be achieved using the keystrokes of **y** and **p**. To start, place the cursor at the first character where you wish to copy from and then enter the visual mode. You can now use the arrow keys to expand your selection. You can then press y to copy the text to the clipboard. Next place your cursor where you wish to paste the selected content and press **p**.

Let us go through the steps given below to understand how to use the copy and paste feature using yank and put in Vim.

1. Open a file on the shell prompt using the command *vim filename*.
2. Repeat the text selection cycle given below as many times as possible until you get used to it.
   Position your cursor to the first character using the arrow keys on the keyboard
   Enter the visual mode by pressing **v**
   Position your cursor to the last character using the arrow keys on the keyboard
   Copy the selection by using yank **y**
   Position your cursor to the location where you want to paste the content using the arrow keys on the keyboard
   Paste the selection by using put **p**
3. You can use the following keystrokes next to save, edit, write or discard the file.
   Enter **:w** to save/write the changes you have made to the file and stay in the command mode
   Enter **:wq** to save/write the changes to the file and exit Vim
   Enter **:q** to discard the changes that you have made to the file and exit Vim

Note: Before you take tips from the advanced vim users, it is advisable that you get used to the basics of vim as performing advanced functions in vim without proper knowledge may lead to modification of important files and result in permanent loss of information. You can learn more about the basics

of vim by looking up the Internet for vim tips.

Using the Graphical Editor to Edit Text Files in Red Hat Enterprise Linux 7

In this section, we will learn to access, view and edit text files using a tool in Red Hat Enterprise Linux 7 known as **gedit**. We will also learn how to copy text between to or more graphical windows.

Red Hat Enterprise Linux 7 comes with a utility known as **gedit, which** is available in the graphical desktop environment of the operating system . You can launch gedit by following the steps given below.

Applications > Accessories > gedit

You can also launch gedit without navigating through the menu. You can press Alt+F2, which will open the Enter A Command dialog box. Type gedit in the text box and hit Enter.

Let us go through the basic keystrokes that are available in gedit. The menu in gedit will allow you to perform numerous tasks related to file management.

- Creating a new file: Navigate through File > New (Ctrl+n) on the menu bar or click the blank paper icon on the toolbar to start a new file
- Saving a file: Navigate through File > Save (Ctrl+s) on the menu bar or click the disk icon on the toolbar to save a file
- Open and existing file: Navigate through File > Open (Ctrl+o) on the menu bar or click on the Open icon on the toolbar. A window will open up showing you all the files on your system. Locate the file that you wish to open and select it and click on open

If you select multiple files and click on open, they will all open up and will have a separate tab under the menu bar. The tabs will display a filename for existing files or when you save a new file with a new name.

Let us now learn how to copy text between two or more graphical windows in Red Hat Enterprise Linux 7. Using the graphical environment in Red Hat Enterprise Linux 7, you can copy text between text windows, documents, and command windows. You can select the text that you want to duplicate using copy and paste, or you can move text using the cut and paste options. In

either case, the text is stored in the clipboard memory so that you can paste it to a destination.

Let us go through the steps to perform these operations.

Selecting the text:

- Left click and hold the mouse button at the first character of the text
- Drag the mouse until you have selected all the desired content and then release the button. Make sure that you do not press any mouse button again as that will result in deselection of all the text

Pasting the selected text: There are multiple methods to achieve this. This is the first one.

- Right click the mouse on the selected text at any point
- A menu will be displayed, and you will get the option to either cut or copy
- Next, open the window where you want to paste the text and place the cursor in the desired location where you wish to paste the text. Right click the mouse again and select paste on the menu that appears

There is a shorter method to achieve the same result as well.

- Firstly, select the text that you need
- Go to the window where you wish to paste the text and place the cursor at the desired location in the window. Middle click the mouse just once and will paste the selected text

This method will help you copy and paste but not cut and paste. However, to emulate a cut and paste, you can delete the original text as it remains selected. The copied text remains in the clipboard memory and can be pasted again and again.

The last method is the one using shortcut keys on the keyboard:

- After selecting the text, you can press Ctrl+x to cut or Ctrl+c to copy
- Go to the window where you want to paste the text and place the cursor

at the desired location

- Press Ctrl+v

Chapter Four: User and Group Management

In this chapter, we will learn about users and groups in Linux and how to manage them and administer password policies for these users. By the end of this chapter, you will be well versed with the role of users and groups on a Linux system and how they are interpreted by the operating system. You will learn to create, modify, lock and delete user and group accounts, which have been created locally. You will also learn how to manually lock accounts by enforcing a password-aging policy in the shadow password file.

Users and Groups

In this section, we will understand what users and groups are and what is their association with the operating system.

Who is a user?

Every process or a running program on the operating system runs as a user. The ownership of every file lies with a user in the system. A user restricts access to a file or a directory. Hence, if a process is running as a user, that user will determine the files and directories the process will have access to.

You can know about the currently logged-in user using the **id** command. If you pass another user as an argument to the id command, you can retrieve basic information of that other user as well.

If you want to know the user associated with a file or a directory, you can use the **ls -l** command and the third column in the output shows the username.

You can also view information related to a process by using the **ps** command. The default output to this command will show processes running only in the current shell. If you use the **ps a** option in the command, you will get to see all the process across the terminal. If you wish to know the user associated with a command, you can pass the **u** option with the ps command and the first column of the output will show the user.

The outputs that we have discussed will show the users by their name, but the system uses a user ID called UID to track the users internally. The usernames are mapped to numbers using a database in the system. There is a flat file stored at /etc/passwd, which stored the information of all users. There are seven fields for every user in this file.

username: password: UID: GID: GECOS: /home/dir: shell

username:

Username is simply the pointing of a user ID UID to a name so that humans can retain it better.

password:

This field is where passwords of users used to be saved in the past, but now they are stored in a different file located at /etc/shadow

UID:

It is a user ID, which is numeric and used to identify a user by the system at the most fundamental level

GID:

This is the primary group number of a user. We will discuss groups in a while

GECOS:

This is a field using arbitrary text, which usually is the full name of the user

/home/dir:

This is the location of the home directory of the user where the user has their personal data and other configuration files

shell:

This is the program that runs after the user logs in. For a regular user, this will mostly be the program that gives the user the command line prompt

What is a group?

Just like users, there are names and group ID GID numbers associated with a group. Local group information can be found at /etc/group

There are two types of groups. Primary and supplementary. Let's understand the features of each one by one.

Primary Group:

- There is exactly one primary group for every user
- The primary group of local users is defined by the fourth field in the /etc/passwd file where the group number GID is listed
- New files created by the user are owned by the primary group
- The primary group of a user by default has the same name as that of the

user. This is a User Private Group (UPG) and the user is the only member of this group

Supplementary Group:

- A user can be a member of zero or more supplementary groups
- The primary group of local users is defined by the last field in the /etc/group file. For local groups, the membership of the user is identified by a comma separated list of user, which is located in the last field of the group's entry in /etc/group
  groupname: password:GID:list, of, users, in, this, group
- The concept of supplementary groups is in place so that users can be part of more group and in turn have to resources and services that belong to other groups in the system

Getting Superuser Access

In this section, we will learn about what the root user is and how you can be the root or superuser and gain full access over the system.

The root user

There is one user in every operating system that is known as the super user and has all access and rights on that system. In a Windows based operating system, you may have heard about the superuser known as the *administrator*. In Linux based operating systems, this superuser is known as the **root** user. The root user has the power to override any normal privileges on the file system and is generally used to administer and manage the system. If you want to perform tasks such as installing new software or removing an existing software, and other tasks such as manage files and directories in the system, a user will have to escalate privileges to the root user.

Most devices on an operating system can be controlled only by the root user, but there are a few exceptions. A normal user gets to control removable devices such as a USB drive. A non-root user can, therefore, manage and remove files on a removable device but if you want to make modifications to a fixed hard drive, that would only be possible for a root user.

But as we have heard, with great power comes great responsibility. Given the unlimited powers that the root user has, those powers can be used to damage the system as well. A root user can delete files and directories, remove or modify user accounts, create backdoors in the system, etc. Someone else can gain full control over the system if the root user account gets compromised. Therefore, it is always advisable that you login as a normal user and escalate privileges to the root user only when absolutely required.

As already mentioned, the root account on Linux operating system is the equivalent of the local Administrator account on Windows operating systems. It is a practice in Linux to login as a regular user and then use tools to gain certain privileges of the root account.

Using Su to Switch Users

You can switch to a different user account in Linux using the **su** command. If you do not pass a username as an argument to the su command, it is implied that you want to switch to the root user account. If you are invoking the command as a regular user, you will be prompted to enter the password of the account that you want to switch to. However, if you invoke the command as a root user, you will not need to enter the password of the account that you are switching to.

su - <username>

[student@desktop ~]$ su -

Passord: rootpassword

[root@desktop ~]#

If you use the command su username, it will start a session in a non-login shell. But if you use the command as su - username, there will be a login shell initiated for the user. This means that using su - username sets up a new and clean login for the new user whereas just using su username will retain all the current settings of the current shell. Mostly, to get the new user's default settings, administrators usually use the su - command.

sudo and the root

There is a very strict model implemented in linux operating systems for users. The root user has the power to do everything while the other users can do nothing that is related to the system. The common solution, which was followed in the past was to allow the normal user to become the root user using the su command for a temporary period until the required task was completed. This, however, has the disadvantage that a regular user literally would become the root user and gain all the powers of the root user. They could then make critical changes to the system like restarting the system and even delete an entire directory like /etc. Also, gaining access to become the root user would involve another issue that every user switching to the root user would need to know the password of the root user, which is not a very good idea.

This is where the **sudo** command comes into the picture. The sudo command lets a regular user run command as if they are the root user, or another user, as per the settings defined in the /etc/sudoers file. While other tools like su would require you to know the password of the root user, the sudo command requires you to know only your own password for authentication, and not the password of the account that you are trying to gain access to. By doing this, it allows the administrator of the system to allow a certain list of privileges to regular users such that they perform system administration tasks, without actually needing to know the root password.

Lets us see an example where the student user through sudo has been granted access to run the **usermod** command. With this access, the student user can now modify any other user account and lock that account

[student@desktop ~]$ sudo usermod -L username

[sudo] password for student: studentpassword

Another benefit of using the sudo access is that all commands that any user runs using sudo are logged to **/var/log/secure**.

Managing User Accounts

In this section, you will learn how to create, modify, lock and delete user accounts that are defined locally in the system. There are a lot of tools available on the command line, which can be invoked to manage local user accounts. Let us go through them one by one and understand what they do.

- ***useradd*** *username* is a command that creates a new user with the username that has been specified and creates default parameters for the user in the /etc/passwd file when the command is run without using an option. Although, the command will not set any default password for the new user and therefore, the user will not be able to login until a password has been set for them.

  The useradd --help will give you a list of options that can be specified for the useradd command and using these will override the default parameters of the user in the /etc/passwd file. For a few options, you can also use the **usermod** command to modify existing users.

  There are certain parameters for the user, such as the password aging policy or the range of the UID numbers, which will be read from the /etc/login.defs file. The file only comes into picture while creating new users. Modifying this file will not make any changes to existing users on the system.

- ***usermod --help*** will display all the basic options that you can use with this command, which can be used to manage user accounts. Let us go through these in brief

-c, --comment COMMENT	This option is used to add a value such as full name to the GECOS field
-g, --gid GROUP	The primary group of the user can be specified using this option

-G, --groups GROUPS	Associate one or more supplementary groups with user
-a, --append	The option is used with the -G option to add the user to all specified supplementary groups without removing the user from other groups
-d, --home HOME_DIR	The option allows you to modify a new home directory for the user
-m, --move-home	You can move the location of the user's home directory to a new location by using the -d option
-s, --shell SHELL	The login shell of the user is changed using this option
-L, --lock	Lock a user account using this option
-U, --unlock	Unlock a user account using this option

- ***userdel*** *username* deletes the user from the /etc/passwd file but does not delete the home directory of that user.
  userdel -r username deletes the user from /etc/passwd and deletes their home directory along with its content as well.

- ***id*** displays the user details of the current user, which includes the UID of the user and group memberships.
  id *username* will display the details of the user specified, which includes the UID of the user and group memberships.

- ***passwd*** *username* is a command that can be used to set the user's initial

password or modify the user's existing password.
The root user has the power to set the password to any value. If the criteria for password strength is not met, a warning message will appear, but the root user can retype the same password and set the password for a given user anyway.
If it is a regular user, they will need to select a password, which is at least 8 characters long, should not be the same as the username, or a previous word, or a word that can be found in the dictionary.

- **UID Ranges** are ranges that are reserved for specific purposes in Red Hat Enterprise Linux 7
  UID 0 is always assigned to the root user.
  UID 1-200 are assigned by the system to system processes in a static manner.
  UID 201-999 are assigned to the system process that does not own any file in the system. They are dynamically assigned whenever an installed software request for a process.
  UID 1000+ are assigned to regular users of the system.

Managing Group Accounts

In this section, we will learn about how to create, modify, and delete group accounts that have been created locally.

It is important that the group already exists before you can add users to a group. There are many tools available on the Linux command line that will help you to manage local groups. Let us go through these commands used for groups one by one.

- ***groupadd*** *groupname* is a command that if used without any options creates a new group and assigns the next available GID in the group range and defines the group in the /etc/login.defs file
  You can specify a GID by using the option **-g GID**

  [student@desktop ~]$ sudo groupadd -g 5000 ateam

  The **-r** option will create a group that is system specific and assign it a

GID belonging to the system range, which is defined in the /etc/login.defs file.

[student@desktop ~]$ sudo groupadd -r professors

- ***groupmod*** command is used to modify the parameters of an existing group such as changing the mapping of the groupname to the GID. The **-n** option is used to specify a new name to the group.

  [student@desktop ~]$ sudo groupmod -n professors lecturers

  The **-g** option is passed along with the command if you want to assign a new GID to the group.

[student@desktop ~]$ sudo groupmod -g 6000 ateam

- ***groupdel*** command is used to delete the group.

  [student@desktop ~]$ sudo groupdel ateam

  Using groupdel may not work on a group that is the primary group of a user. Just like userdel, you need to be careful with groupdel that you check that there are no files on the system owned by the user existing after deleting the group.

- ***usermod*** command is used to modify the membership of a user to a group. You can use the command *usermod -g groupname* to achieve the same.

  [student@desktop ~]$ sudo usermod -g student student

  You can add a user to the supplementary group using the *usermod -aG groupname username* command.

  [student@desktop ~]$ sudo usermod -aG wheel student

  Using the -a option ensures that modifications to the user are done in

append mode. If you do not use it, you will be removed from all other groups and be only added to the new group.

User Password Management

In this section, we will learn about the shadow password file and how you can use it to manually lock accounts or set password-aging policies to an account. In the initial days of Linux development, the encrypted password for a user was stored in the file at /etc/passwd, which was world-readable. This was tested and found to be a secure path until attackers started using dictionary attacks on encrypted passwords. It was then that it was decided to move the location of encrypted password hash to a more secure location, which is at /etc/shadow file. The latest implementation allows you to set password-aging policies and expiration features using this new file.

The modern password hash has three pieces of information in it. Consider the following password hash:

1gCLa2/Z$6Pu0EKAzfCjxjv2hoLOB/

1. **1**: This part specifies the hashing algorithm used. The number 1 indicates that an MD5 hash has been implemented. The number **6** comes into the hash when a SHA-512 hash is used.
2. **gCLa2/Z**: This indicates the salt used to encrypt the hash. It is a randomly chosen salt at first. The combination of the unencrypted password and salt together form the encrypted hash. The advantage of having a salt is that two users who may be using the same password will not have identical hash entries in the /etc/shadow file.
3. **6Pu0EKAzfCjxjv2hoLOB/**: This is the encrypted hash.

In the event of a user trying to log in to the system, the system looks up for their entry in the /etc/shadow file. It then combines the unencrypted password entered by the user with the salt for the user and uses the hash algorithm specified to encrypt this combination. It is implied that the password typed by the user is correct of this hash matches with the hash in the /etc/shadow file. Otherwise, the user has just typed in the wrong password and their login attempt fails. This method is secure as it allows the system to determine if a user typed in the correct password without having to store the actual unencrypted password in the file system.

The format of the /etc/shadow file is as below. There are 9 fields for every user as follows.

name: password: lastchange: minage: maxage: warning: inactive: expire: blank

name: This must be a valid username on the system through, which a user logs in.

password: This is where the encrypted password of the user is stored. If the field starts with an exclamation mark, it means that password is locked.

lastchange: This is the timestamp of the last password change done for the account.

minage: This defines the minimum number of days before a password needs to be changed. If it is the number 0, it means there is no minimum age for the account.

maxage: This defines the maximum number of days before a password needs to be changed.

warning: This is a warning period that shows that the password is going to expire. If the number is 0, it means that no warning will be given before password expiry.

inactive: This is the number of days after password expiry the account will stay inactive. During this, the user can use the expired password and still log into the system to change his password. If the user fails to do so in the specified number of days for this field, the account will get locked and become inactive.

expire: This is the date when the account is set to expire.

blank: This is a blank field, which is reserved for future use.

Password Aging

Password aging is a technique that is employed by system administrators to safeguard bad passwords, which are set by users of an organization. The policy will basically set a number of days, which is 90 days by default after, which a user will be forced to change their password. The advantage of

forcing a password change implies that even if someone has gained access to a user's password, they will have it with them only for a limited amount of time. The con to this approach is that users will keep writing their password in some place since they can't memorize it if they keep changing it.

In Red Hat Enterprise Linux 7, there are two ways through, which password aging can be enforced.

1. Using the chage command on the command line
2. Using the User Management application in the graphical interface

The **chage** command with the **-M** option lets a system admin specify the number of days for, which the password is valid. Let us look at an example.

[student@desktop ~]$ sudo chage -M 90 alice

In this command, the password validity for the user alice will be set to 90 days after, which the user will be forced to reset their password. If you want to disable password aging, you can specify the -M value as 9999, which is equivalent to 273 years.

You can set password aging policies by using the graphical user interface as well. There is an application called User Manager, which you can access from the Main Menu Button > System Settings > Users & Groups. Alternatively, you can type the command **system-config-users** in the terminal window. The User Manager window will pop up. Navigate to the Users tab, select the required user from the list, and click on the Properties button where you can set the password aging policy.

Access Restriction

You can set the expiry for an account using the **chage** command. The user will not be allowed to login to the system once that date is reached. You can use the **usermod** command with the **-L** option to lock a particular user account.

[student@desktop ~]$ sudo usermod -L alice

[student@desktop ~]$ su - alice

Password: alice

su: Authentication failure

The usermod command is useful to lock and expire an account at the same time in a case where the employee might have left the company.

[student@desktop ~]$ sudo usermod -L -e 1 alice

A user may not be able to authenticate into the system using a password once their account has been locked. It is one of the best practices to prevent authentication of an employee to the system who has already left the organization. You can use the **usermod -u** *username* command later to unlock the account, in the event that the employee has rejoined the organization. While doing this, if the account was in an expired state, you will need to ensure that you set a new expiry date for the account as well.

The nologin shell

There will be instances where you want to create a user who can authenticate using a password and get a login into the system but would not need a shell to interact with the system. For example, a mail server may require a user to have an email account so that the user can login and check their emails. But it is not necessary that the user needs a login to the system to check their emails.

This is where the nologin shell comes as a solution. What we do is we specify the shell for this user to point to **/sbin/nologin**. Once this is done, the user cannot login to the system using the direct login procedure.

[root@desktop ~]# usermod - s /sbin/nologin student

[root@desktop ~]# su - student

Last login: Tue Mar 5 20:40:34 GMT 2015 on pts/0

The account is currently not available.

By using the nologin shell for the user, you are denying the user interactive login into the system but not all access to the system. The user will still be able to use certain web applications for file transfer applications to upload or download files.

Chapter Five: Accessing Files in Linux and File System Permissions

In this chapter, we will learn about the working of the Linux file system permissions model and how the permissions and ownership of files can be changed using the command line tools available to us. By the end of the chapter, we will be well versed with how file permissions affect files in Linux and how permissions can be used to employ security to files and directories.

Linux File System Permissions

File Permissions is a feature in Linux through, which the access to a file by a user can be controlled. Although the Linux file system model is simple, it is still flexible in nature, which makes it easy for a new user to understand and apply it and handle file permissions in an easy manner.

There are three categories of users to, which permissions apply with respect to a file. The three categories of file users are as follows.

1. user
2. group
3. other

The hierarchy is such that user permissions will override group permissions, which will override other permissions.

The permissions that apply to a file or directory also belong to three categories.

1. read
2. write
3. execute

Let us see their effects on a file or a directory.

Permission	Effect on Files	Effect on Directories
r (read)	Read access to file content	Contents of a directory that is filenames will be listed.
w (write)	Write access to file content	Contents of the directory that is files can be created or deleted.
x (execute)	The file can be executed as a command	Contents of the directory can be accessed subject to the

		permission of the file itself.

A user is given read and execute access to a file by default so that they can read the content of the file and execute the file if it is an executable file. However, if a user has only read only access, they will only be able to read the contents of the file but no other information at all such as permissions of the file, timestamps, etc. If there is only an execute access for a user to the file, they will not be able to list the filename down in a directory, but if they already know the name of the file, they will be able to execute the file in a command.

If a user has write permissions to a directory, they have all rights to delete any file in that directory irrespective of the actual permissions on the file itself. There is, however, an exception to this where this can be overridden by special permission, known as the sticky bit, which we will discuss later in this chapter.

Let us now see how you can view the permissions, which are assigned to a file or a directory along with the ownership. You can use the **ls** command with the **-l** option to list down the files and directories along with their permissions and ownerships.

```
[student@desktop ~]$ ls -l test

-rw-rw-r--. 1 student student 0 Feb 5 15:45test
```

Using the **ls -l directoryname** command, you can list down all the contents of the directory with additional information of permissions, ownerships, timestamps, etc. If you wish to see the listing of the parent directory itself and not descend down to the contents of the directory, you can use the **-d** option.

```
[student@desktop ~]$ ls -ld /home

drwxr-xr-x. 5 root root 4096 Feb 8 17:45/home
```

If you have been using a Windows system, you will realize that the List Folder Contents permission from Windows is the equivalent of the Linux Read permission, and
the Windows Modify permission is the equivalent of the Linux Write permission.

Windows has a feature called Full Control which is the equivalent of the power that the Root user has with respect to files and directories in Linux.

Managing File System Permissions using the Command Line

In this section, we will learn how we can use the command line in Linux to manage the permissions and ownerships for a file.

Changing the permissions of files and directories

The command we can use to change the permissions of files in Red Hat Enterprise Linux 7 from the command line is **chmod**, which is the short form for change mode since permissions are also referred to as the mode of a file or directory. The syntax of the command is followed by an instruction as to what needs to be changed and then the name of the file or directory on, which the operation needs to be executed. You can provide the instruction in two ways, that is, either numerically or symbolically.

Let us first go through the symbolic method and the syntax looks like this

chmod WhoWhatWhich files|directory

- Who is the user u, group g, other o and a for all
- What is + to add, - to remove, = to set exactly
- Which is r for read, w for write, and x for execute

You will use letter to specify the different groups that you wish to change the permissions for. u is for the user , g is for the group, o is for other, and a is for all.

When you are setting the permissions using the symbolic method, you do not need to specify a new set of permissions for the file. You can rather just make changes to the existing permissions. You can achieve this by using the three symbols +, - and =to add permissions to a set, remove permissions to a set or replace the entire set for a group of permissions respectively.

Lastly, the permissions are represented by using the letters where r is for read, w is for write, and x is for execute. Note that if you are using the chmod command with the symbolic method, and use a capital **X** as a permission flag, it will add the execute permission only if the file is a directory or already has an execute permission set for user u, group g, or other o.

Let us first go through the numeric method and the syntax looks like this

chmod ### files|directory

- Every position of the # represents an access level viz. User, group, and other
- # is the sum of read r=4, write w=2, execute x=1

In this method, we can set up permissions for files and directories using 3 digits(and sometimes 4 for special permissions) known as an octal number. A single digit can specify the number between 0-7, which shows the exact number of possibilities we can have with read, write and execute values.

Ig we understand the mapping between symbolic and numeric values, we will learn how yo do the conversion between the two as well. In the numeric representation, which is done by three digits, each digit represents permissions for a group. If we start from left to right, the first bit is for user, the second bit is for group and the third bit is for other. And for each of these groups, we can use a combination of the the read write and execute values, which are 4, 2 and 1 respectively.

Let us look at a symbolic representation of permissions, which is **-rwxr-x---**

In this representation, the user has the permissions of **rwx, which** is read write and execute. If we convert this to numeric form, it will be read r 4 + write w 2 + execute x 1, which is a total of 7.

Next for the group, we see that the permissions in symbolic form are **r-x, which** is to only read and execute. If we convert this to numeric form, it will be read r 4 + execute x 1, which is a total of 5.

The permission for others is ---, which means read write and execute are all 0. This means the other bit will be 0.

We can now say that the complete permission for all groups to this file in numeric format is represented as **750**.

We can also to a converse operation on this and do a conversion from numeric format to symbolic format.

Consider the permission **640**.

We know that the user bit is the left most bit, which is 6. The only combination of read, write and execute that will give us a 6 is that of read r 4 + write w 2. This means that the permission for the user in symbolic format is **rw-**.

Next the group bit is 4 and the only combination of read, write and execute that will give us a 4 is that of read r 4. This means that the permission for the group in symbolic format is **r--**.

Next the other bit is 0 and the only combination of read, write and execute that will give us a 0 is if values for read, write and execute are all 0. This means that the permission for the group in symbolic format is **---**.

As a whole, the permission for this file in symbolic format will look like **-rw-r-----.**

Note: You can use the **-R** option with the **chmod** command if you want to set the same permissions recursively for all files under a directory tree. It is also noteworthy that while doing this, you can use the **X** flag symbolically to set the permissions of all directories so that they are accessible and that you can skip files while doing so.

Changing the user and group ownership of files and directories

By default, if a file is newly created, it is owned by the user who created the file. The default group ownership of that file is also associated by default to the primary group of the user who created it. Since Red Hat Enterprise Linux 7 has the concept of user private groups, the group will mostly be a group, which has only one member who is the user themselves. Access to files and directories can be granted by changing their owner and group.

You can use the **chown** command to change the ownership of a file or a directory. Let us see an example.

[root@desktop ~]# chown student newfile

In the example above, we are changing the user owner of the file newfile to student.

You can also use the option **-R** with the chown command, which will recursively change the ownership of a directory and all its files and

subdirectories. The command can be used as shown below.

[root@desktop ~]# chown -R student parentdirectory

We can also use the chown command to change the group ownership of files and directories. The command is to be followed by the group name preceded by the colon :

Let us look at an example.

[root@desktop ~]# chown :admins newfile

This will change the group ownership of the file newfile to admins.

You can also use the chown command to change the user ownership and group ownership at the same time. The syntax for it is as follows.

[root@desktop ~]# chown student:admins newfile

This will change the user ownership to student and group ownership to admins for the file newfile.

The ownership of files and directories can only be changed by the root user. However, the group ownership of a file can be changed both by the root and the actual user who owns that file. Non-root users have access to provide ownership to groups that they are part of.

Note: An alternate command to the chown command is the **chgrp** command, which can be used to change the group ownership of a file or directory. The chgrp command works exactly the same way as the chown command and also works with the -R option to recursively change group ownership.

Managing File Access and Default Permissions

In this section, we are going to learn about special permissions. We will create a directory under, which files that will be created will have write access for users of the group that owns the directory by default. This will be achieved using the special permissions known as sticky bits.

Let us see how we can use the special permissions and apply them. There is a bit known as the **setuid** and **setgid** on the permissions, which allows an executable file to run as the user of that file or the group of that file and not as

the user that ran the actual command.

One such example is the **passwd** file. Let us have a look at it.

[student@desktop ~]$ ls -l /usr/bin/passwd

-rw**s**r-xr-x. 1 root root 34598 Jul 152011 /usr/bin/passwd

The sticky bit for any file in the permissions sets a restriction on file deletions. Only the user who owns the file and the root user can delete the file. An example of this is /tmp.

[student@desktop ~]$ ls -ld /tmp

drwxrwxrw**t** 39 root root 4096 Jul 102011 /tmp

Lastly, the setgit bit is a bit that allows all files created within a directory to inherit the permissions of the directory rather than getting it set by the user who created the file. Group collaborative directories mostly use this feature to change file permissions from the default private group to shared groups.

Let us go through the effects of special permissions on files and directories.

Special Permission	Effect on Files	Effect on Directories
u+s suid	The file will execute as the user who owns the file and not the user who ran the command	No effect
g+s sgid	File executes as the group that owns the file	The group owner of the newly created file in the directory will match the group owner of the directory
o+t sticky	No effect	Users who have write access on the directory can only delete files owned by them. They cannot delete or force

		writes to files owned by other users

Let us see how we can set special permissions on files and directories.

- Symbolically, setuid is u+s, setgid is g+s and sticky is o+t
- Numerically, the special permissions use the fourth bit that precedes every user's first digit. setuid is 4, setgid is 2 and sticky is 1.

Let us now try and understand what default file permissions are. The permissions set for files by default are the ones that are set by the processes that created those files. For example, if you are using a text editor like Vim to create a file, the file will have read and write access for everyone but no execute access. Shell redirection follows the same rule. Additionally, compilers can create files that are binary executable in nature. Therefore, the files will have executable permissions. The mkdir command is used to create directories and these directories have all permissions for read, write and execute.

Research and experience has shown that the permissions on files and directories are not set when they are created because the **umask** of the shell process clears these permissions. If you use the umask command without any arguments, it will show the value of the current umask of the shell.

[student@desktop ~]$ umask

0002

There is a umask for every process on the system. The umask is basically an octal bitmask that clears the permissions of newly created files and directories that are created by a process. If the umask has a bit set, the corresponding permission is cleared in newly created files.

Let us take an example. The bitmask value shown above is 0002 where the bit for other users is 2. We know by this that the special, user and group permissions will not be cleared since those bits are all 0. Therefore permissions for other users will be cleared since the corresponding umask bit is 2. We assume that there are zeroes that lead if the umask is less than three digits.

The default umask values in the system for users of bash shell are defined at /etc/profile and /etc/bashrc file. The system defaults can be overridden by the users in their .bash_profile and .bashrc files.

Chapter Six: Linux Process Management

In this chapter, we will learn how to monitor and manage processes that run on Red Hat Enterprise Linux 7. By the end of this chapter, we will be able to list processes and interpret basic information about them on the system, use bash job control to control processes, use signals to terminate processes, and monitor system resources and system load caused by processes.

Processes

In this section, we will define the cycle of a typical process and understand the different states of a process. We will also learn to view and interpret processes.

What is a process?

An executable program in a state where it is running after being launched is called a process. A process has the following features.

- Allocated memory that points to an address space
- Properties with respect to security, which include ownership privileges and credentials
- Program code that contains one or more executable threads
- The state of the process

The process environment has the following features

- Variables that are both local and global in nature
- A current scheduling context
- System resources allocated to it, which include network ports and file descriptors

An existing process is known as a parent process, which splits and duplicates its address space to create a child process. For security and tracking, a unique process ID known as PID is assigned to every new process. The PID and the parent process's ID known as PPID together make the environment for the child process. A child process can be created by any process. All the processes in the system descend from the very first process of the system, which is known as **systemd** on Red Hat Enterprise Linux 7.

As the child process splits from a parent process through a fork, properties such as previous and current file descriptors, security identities, port privileges, resource privileges, program code, environment variables are all inherited by the child process. Once these properties have been inherited, the child process can then execute its own program code. When a child process

runs, the parent process goes to sleep by setting a request to a wait flag until the child process completes. Once the child process completes, it leaves the system and releases all system resources and environment it has previously locked, and what remains of it is known as a zombie. Once the child process leaves, the parent process wakes up again and clean the remaining bit and starts to run its own program code again.

Process States

Consider an operating system, which is capable of multitasking. If it has hardware with a CPU that has multiple cores, every core can be dedicated to one process at a given point in time. During runtime, the requirements of CPU and other resources keep changing for a given process. This leads to processes being in a state, which changes as per the requirements of the current circumstance.

Let us go through the states of a process one by one by looking at the table given below.

Name	Flag	State name and description
Running	R	TASK_RUNNING: The process is waiting or executing on the CPU. The process could be executing routines for the user or the kernel. It could also be in a queued state where it is getting ready to run known as the Running state.
Sleeping	S	TASK_INTERRUPTIBLE: The process is waiting for a condition such as system resources access, hardware request, or a signal. When the condition is met by an event or signal, the process will get back to Running.
	D	TASK_UNINTERRUPTIBLE: The process is in the Sleeping state here as well, but unlike **S**, in this it will not respond to any signals. It is used only in specific

		conditions where an unpredictable device state can be caused due to process interruption.
	K	TASK_KILLABLE: It is much like the uninterruptible **D** state, but the task that is waiting can respond to a signal to be killed. Killable processes are displayed as the **D** state by utilities.
Stopped	T	TASK_STOPPED: The process is in a Stopped state because of another signal or process. Another signal can, however, send the process back into the Running state.
	T	TASK_TRACED: A process is in a state of being debugged and is therefore in a Stopped state. It shares the same **T** flag.
Zombie	Z	EXIT_ZOMBIE: A child process is complete, and it leaves the system and lets the parent process know about it. All resources held by the child process are released except for it process ID PID.
	X	EXIT_DEAD: The parent process has cleaned up the remains of the child process after it has exited, the child process has now been released completely. This state is rarely observed in utilities that list processes.

Listing processes

The current processes in the system can be listed using the **ps** command at shell prompt. The command provides detailed information about processes, which include:

- The UID user identification, which determines the privileges of the process

- The unique process ID PID
- The real time usage of the CPU
- The allocated memory by the process in various locations of the system
- The location of the process STDOUT standard output, known as the controlling terminal
- The current state of the process

The option **aux** can be used with the ps command, which will display detailed information of all the processes. It includes columns, which are useful to the user and also shows processes, which are without a controlling terminal. If you use the long listing option **lax**, you will get some more technical details, but it may display faster skipping the lookup of the username.

If you run the ps command without any options, it will display processes, which have the same effective user ID EUID as that of the current user and associated with the same terminal where the ps command was invoked.

- The ps listing also shows zombies, which are either exiting or defunct
- ps command only shows one display. You can alternatively use the top command, which will keep repeating the display output in realtime
- Process, which have round brackets are usually the ones run by kernel threads. They show up at the top of the listing
- The ps command can display a tree format so that you can understand the parent and child process relationships
- The default order in, which the processes are listed is not sorted. They are listed in a manner where the first process started, and the rest followed. You may feel that the output is chronological, but there is no guarantee unless you explicitly use options like -O or --sort

Controlling Jobs

In this section, we will learn about the terms such as foreground, background and the controlling terminal. We will also learn about using job control, which will allow us to manage multiple command line tasks.

Jobs and Sessions

Job control is a feature in shell through, which multiple commands can be managed by a single shell instance.

Every pipeline that you enter at the shell prompt is associated with a job. All processes in this pipeline are a part of the job and are members of the same process group. A minimal pipeline is when only a single command is entered on the shell prompt. In such a case, that command ends up being the only member of the job.

At a given time, inputs given to the command line from a keyboard can be read by only one job. That terminal is known as the controlling terminal and the processes that are a part of that job are known as foreground processes.

If there is any other job associated with that controlling terminal of, which it is a member, it is known as the background process of that controlling terminal. Inputs given from the keyboard to the terminal cannot be read by background processes, but they can still write to the terminal. A background job can be in a stopped state or a running state. If a background process tries to read from the terminal, the process gets automatically suspended.

Every terminal that is running is a session of its own and can have processes that are in the foreground and the background. A job is a part of one session only, the session that belongs to its controlling terminal.

If you use the **ps** command, the listing will show the name of the device of the controlling terminal of a process in a column named **TTY**. There are some processes started by the system, such as system daemons, which are not a part of the shell prompt. Therefore, these processes are not part of a job, or they do not have a controlling terminal and will never come to the foreground. Such processes, when listed using the ps command shows **?** mark in the TTY column.

Running Background Jobs

You can add an ampersand **&** to the end of a command line, which will run the command in the background. There will be a unique job number assigned to the job, and a process ID PID will be assigned to the child process, which is created in bash. The shell prompt will show up again after the command is executed as the shell will not wait for the child process to complete since it is running in the background.

[student@desktop ~]$ sleep 10000 &

[1] 5683

[student@desktop ~]$

Note: When you are putting a pipeline in the background with an ampersand **&**, the process ID PID that will show up in the output will be that of the last command in the pipeline. All other command that precede will be a part of that job.

[student@desktop ~]$ example_command | sort |mail -s “sort output” &

[1] 5456

Jobs are tracked in the bash shell, per session, in the output table that is shown by using the **jobs** command.

[student@desktop ~]$ jobs

[1]+ Running sleep 10000 &

[student@desktop ~]$

You can use the **fg** command with a job ID(*%job number*) to bring a job from the background to the foreground.

[student@desktop ~]$ fg %1

sleep 10000

-

In the example seen above, we brought the sleep command, which was

running in the background to the foreground on the controlling terminal. The shell will go back to sleep until this child process completes. This is why you will have to wait until the sleep command is over for the shell prompt to show up again.

You can send a process from the foreground to the background by pressing **Ctrl+z** on the keyboard, which will send a suspend request.

```
sleep 10000
^Z
[1]+  Stopped      sleep 10000
[student@desktop ~]$
```

The job will get suspended and will be placed in the background.

The information regarding jobs can be displayed using the **ps j** command. The display will show a PGID, which is the PID of the process group leader and refers to the first job in the pipeline of the job. The SID is the is the PID of the session leader, which with respect to a job refers to the interactive shell running on the controlling terminal.

```
[student@desktop ~]$ ps j
PPID  PID   PGID SID   TTY          TPGID    STAT UID   TIME COMM
2434 2456 2456 2456 pts/0        5677 T      1000 0:00  sleep 10000
```

The status of the sleep command is T because it is in the suspended state.

You can start a suspended process again the in background and put it into a running state by using the **bg** command with the same job ID.

```
[student@desktop ~]$ bg %1
[1]+  sleep 10000 &
[student@desktop ~]$
```

If there are jobs that are suspended and you try to exit the shell, you will get a warning that will let you know that there are suspended jobs in the background. If you confirm to leave, the suspended jobs are killed

immediately.

Killing Processes

In this section, we will learn how to use command to communicate with processes and kill them. We will understand what is a daemon process and what are its characteristics. We will also learn how to end processes and sessions owned by a user.

Using signals to control processes

A signal is an interrupt developed through software to be sent to a process. Events are sent to a program with the help of signals. These events that generate a signal can be external events, errors, or explicit requests such as commands sent using the keyboard.

Let us go through a few signals, which are useful for system admins in their routine day to day system management activities.

Signal number	Short name	Definition	Purpose
1	HUP	Hangup	This signal reports the termination of the controlling process in a terminal. Process reinitialization or configuration reload can be requested using this signal without any termination.
2	INT	Keyboard Interrupt	This signal lead to termination of a program. The signal can either be blocked or handled. The signal is sent by using the **Ctrl+c** on the keyboard known as **INTR**
3	QUIT	Keyboard quit	The signal is similar to **SIGINT** with the difference that a process dump is generated at termination. The signal is sent by using the **Ctrl+** on the keyboard known as **QUIT**

9	KILL	Kill, unblockable	This signal leads to an abrupt termination of the program.It cannot be blocked, handled, or ignored and is always fatal.
15 default	TERM	Terminate	This signal leads to termination of the program. Unlike **SIGKILL,** this signal can be clocked, ignored, or handled. This is requesting a program to terminate in a polite way, which results in proper clean-up.
18	CONT	Continue	This signal is sent to a process that is in a stopped state such that it resumes. The signal cannot be blocked, and process is resumed even if the signal is handled.
19	STOP	Stop, Unblockable	This signal leads to suspension of the process and cannot be handled or blocked.
20	TSTP	Keyboard stop	Unlike **SIGSTOP,** this signal can be blocked, handled, or ignored. The signal is sent by using the **Ctrl+z** on the keyboard known as **SUSP**

Note: The number of signal number can change based on the hardware being used for the Linux operating system, but the signal names and their purposes are standardized. Therefore, it is advisable that you use the signal names instead of the signal number on the command line. The signal numbers that we discussed above are only for systems that are associated with the Intel x86 architecture.

There is a default action associated with every signal, which corresponds to one of the following.

Term - The program is asked to exit or terminate at once.

Core - The program is asked to terminate but is asked to also save a memory image or a core dump before terminating.

Stop - the program is suspended or asked to stop and will have to wait to resume again.

Expected event signals can be tackled by programs by implementing routines for handlers so that they can replace, ignore, or extend the default action of a signal.

Commands used to send signals through explicit requests

Processes that are running in the foreground can be signaled using the keyboard by users, wherein control signals are sent to the process using keys like Ctrl+z for suspend, Ctrl+c for kill, and Ctrl+\ for getting a core dump. If you want to send signals to processes that are running in the background or are running in a different session altogether, you will need to use a command to send signals.

You can either use signal names(-HUP or -SIGHUP) to signal numbers(-1) to specify a signal. Processes, which are owned by a user can be killed by the users themselves, but processes owned by others will need root user privileges to be killed.

- The **kill** command can be sent to a process using the process ID PID. However, irrespective of the name, the kill command can be used to send other signals to a process as well and not just for sending a signal to terminate the process.

  [student@desktop ~]$ kill PID
  [student@desktop ~]$ kill -signal PID

- The **killall** command can be used to send a signal to multiple processes, which may match a given criteria such as processes owned by a particular user, command name, all system processes.

  [student@desktop ~]$ killall command_pattern

[student@desktop ~]$ killall -signal command_pattern
[student@desktop ~]$ killall -signal -u username command_pattern

- Just like the **killall** command, there is another command called **pkill, which** can be used to signal multiple processes at the same time. The selection criteria used by pkill is advanced in comparison to killall and contains the following combinations.

  Command - Pattern that is matched using the command name
  UID - Processes that belong to a particular user matched using UID
  GID - Processes that belong to a particular group matched using GID
  Parent - Child processes that belong to a particular parent process
  Terminal - Processes that are running on a particular controlling terminal

  [student@desktop ~]$ pkill command_pattern
  [student@desktop ~]$ pkill -signal command_pattern
  [root@desktop ~]# pkill -G GID command_pattern
  [root@desktop ~]# pkill -P PPID command_pattern
  [root@desktop ~]# pkill -t terminal_name -U UID command_pattern

Administratively logging out users

The **w** command lists down all the users that are logged into the system and the processes that are being run by these users. You can determine the location of the users by analyzing the **FROM** and the **TTY** columns.

Every user is associated with a controlling terminal, which is indicated by **pts/N** while working on a graphical interface or **ttyN** while working on a system console where **N** is the number of the controlling terminal. Users who have connected remotely to the system will be displayed in the **FROM** column when you use the **-f** option.

[student@desktop ~]$ w -f

12:44:34 up 25 min, 1 users, load average: 0.06, 0.45, 0.55

USER TTY FROM LOGIN@ IDLE JCPU PCPUW

student pts/0 :0 12:32 2.02s 0.07s 0.07s w -f

The session login time will let you know as to how long a user has been on the system. The CPU resources that are utilized by current jobs, including the child processes and background jobs are shown in the JCPU column. CPU utilization for foreground processes are shown in the PCPU column.

If a user is violating security of the system, or over-allocating resources, they can be forced out of the system. Therefore, if the system admin is requesting a user to close processes that are not required, close command shells that are unused, exit login sessions, they are supposed to follow the system admin.

In situations where a user is out of contact and has a ongoing sessions, which are putting a load on the system by consuming resources, a system admin may need to administratively end their session.

Note: The signal to be used in this case is **SIGTERM** but most system admins use **SIGKILL, which** can be fatal. The SIGKILL signal cannot be handled or ignored, it is fatal. Processes are forced to terminate without completing clean-up routines. Therefore, we recommend that you send the SIGTERM signal first before trying the SIGKILL signal when the process is not responding.

Signal can be sent individually or collectively to terminal or processes. You can use the **pkill** command to terminate all processes for a particular user. If you want to kill all the processes of a user and all their login shells, you will need to use the **SIGKILL** signal. This is because the session leader process, which is the initial process in a session, can handle session termination requests and other signals coming from the keyboard.

[root@desktop ~]# pgrep -l -u alice

6787 bash

6789 sleep

6999 sleep

7000 sleep

[root@desktop ~]# pkill -SIGKILL -u alice

[root@desktop ~]# pgrep -l -u alice

[root@desktop ~]#

If you need certain processes by a user and only want to kill a few of their other processes, it is not necessary to kill all their processes. Use the **w** command and figure out the controlling terminal for the session and then use the terminal ID to kill processes from a terminal, which is not required. The session leader, which is the bash login shell will survive the termination command unless you use the SIGKILL command, but this will terminate all other session processes.

[root@desktop ~]# pgrep -l -u alice

6787 bash

6789 sleep

6999 sleep

7000 sleep

[root@desktop ~]# w -h -u alice

alice tty3   18:545:07   0.45s 0.34s -bash

[root@desktop ~]# pkill -t tty3

[root@desktop ~]# pgrep -l -u alice

6787 bash

[root@desktop ~]# pkill SIGKILL -t tty3

[root@desktop ~]# pgrep -l -u alice

The criteria of terminating processes selectively can also be applied by using arguments of relationships between parent and child processes. The **pstree** command can be used in this case. The pstree command shows a process tree for a user or for the system. You can kill all its child processes by passing the parent process's parent ID PID. The bash login shell of the parent process still remains since only the child processes are terminated.

```
[root@desktop ~]# pstree -u alice
bash(8341) sleep(8454)
        sleep(8457)
        sleep(8459)
[root@desktop ~]# pkill -P 8341
[root@desktop ~]# pstree -l -u alice
bash(8341)
[root@desktop ~]# pkill -SIGKILL -P 8341
[root@desktop ~]# pstree -l -u alice
bash(8341)
[root@desktop ~]#
```

Process Monitoring

In this section, we will learn how to monitor processes in real time and how to interpret load averages on the CPU of the system.

Load Average

The Linux kernel is capable of calculating a **load average** metric, which is the **exponential moving average** of the **load number**, a cumulative count of the CPU that is kept in accordance with the system resources that are active in that given instance.

- Threads that are currently running or threads that are waiting for input or output are counted as the active requests in the CPU queue. Meanwhile, the kernel keeps track of the activity of process resources and the changes in the state of the process.
- The calculation routine run by default in the system at an interval of every five seconds is known as load number. The load number will accumulate and average out all the active requests into one single number for every CPU.
- The mathematical formula used to smoothen the highs and lows of trending data, the increase in significance of current activity, and decrease in the quality of aging data is known as the exponential moving average.
- The result of the routine load number calculation is known as load average. It refers to the display of 3 figures, which show the load averages for 1, 5 and 15 minutes.

Let us try and understand how the load average calculation works in Linux systems.

The load average is a perception of load received by the system over a period of time. Along with CPU, the load average calculation also takes into consideration the disk and the network input and output.

- Linux systems do not just count processes. The threads of a process are also counted individually and account as different tasks. The

requests to CPU queues for running threads(nr_running) and threads that are waiting for I/O resources(nr_iowait) correspond to the process states of R Running and D Uninterruptible Sleeping. Tasks that may be sleeping are waiting for responses from disk and networks are included in tasks waiting for Input/Output I/O.

- All the CPUs of the system are taken into consideration and there the load number is known as the global counter for calculation. We cannot have counts that are accurate per CPU as tasks, which were initially sleeping, may be assigned to a different CPU when they resume. Therefore, we go for a count that has cumulative accuracy. The load average that is displayed represents all the CPUs.
- Linux will count each physical core of the CPU and microprocessor hyperthread as an execution unit, and therefore as an individual CPU. The request queues for each CPU is independent. You can check the /proc/cpuinfo file, which has all the information about the CPUs.

```
[root@desktop ~]# grep “model name” /proc/cpuinfo
model name: Intel(R) Core(TM) i5 CPU    M 2600 @ 2.60GHz
model name: Intel(R) Core(TM) i7 CPU    M 2600 @ 3.60GHz
model name: Intel(R) Core(TM) i7 CPU    M 2600 @ 3.60GHz
model name: Intel(R) Core(TM) i7 CPU    M 2600 @ 3.60GHz
[root@desktop ~]# grep “model name” /proc/cpuinfo |wc -l
4
```

- Previously known UNIX systems used to consider only CPU load or the length of the run queue to calculate the system load. But soon it was realized that a system would have CPUs that may be idle, but the other resources like disk and network could be busy and it was factored into the load average shown in modern Linux systems. If the load average is high despite minimal CPU activity, you may want to have a look at the disk and the network.

Let us now learn how we can interpret the values shown for load averages. This is an important part of being a system admin. As we have already seen, you will see three values, which are the load values over a time period of 1, 5, and 15 minutes. Having a quick look at these three values is enough to understand whether the load on the system is increasing or decreasing. We can then calculate the approximate value for per CPU load, which will let us know if the system is experiencing severe wait time.

- You can use the command line utilities of **top, uptime, w** and **gnome-system-monitor** to display values of average load.
- [root@desktop ~]# uptime
  15:30:45 up 14 min, 2 users, load average: 2.56, 4.56, 5.76
- You can now divide the load average values that you see by the number of logical CPUs that are present in the system. If the result shows a value below 1, it implies that resources utilization and wait times are minimal. If the value is above 1, it indicates that resources are saturated and that there is waiting time.
- If the CPU queue is idle, then load number will be 0. Threads that are waiting or ready will add a count of 1 to the queue. If the total count on the queue is 1, resources of CPU, disk and network are busy, but there is no waiting time for other requests. With every additional request, the count increases by 1, but since many requests can be executed simultaneously, the resource utilization goes up but there is no wait time for other requests.
- The load average in increases by processes that may be in the sleeping state since they are waiting for input or output, but the disk and the network are busy. Although this does not mean that the CPU is being utilized, it still means that there are processes and users waiting for system resources.
- The load average will stay below 1 until all the resources begin to get saturated as tasks are seldom found to be waiting in the queue. It is only when requests start getting queued and are counted by the calculation routine that the load average starts spiking up. Every additional request

coming in will start experiencing wait time when the resource utilization touches 100 percent.

Process monitoring in Real time

Much like the **ps** command, the **top** command gives a dynamic view of the processes in the system, which shows a header summary and list of threads and processes. The difference is that the output in the ps command is static in nature and just gives a one time output. The output of the top command is dynamic and keeps refreshing the values in real time. The interval at, which the values refresh can be customized. You can also configure other things such as sorting, column reordering, highlighting, etc. and these user configurations can be saved and are persistent.

The default output columns are as follows.

- The process ID **PID**
- The process owner that is the user name **USER**
- All the memory that is used by a process **VIRT**, which includes memory used by shared libraries, resident set, and memory pages that may be mapped or swapped.
- The physical memory used by a process known as resident memory **RES, which** includes memory used by shared objects.
- The state of the process **S** displays as
  D: Uninterruptible Sleeping
  R: Running or Runnable
  S: Sleeping
  T: Traced or Stopped
  Z: Zombie
- The total processing time since the process began is known as CPU time **TIME**. It can be toggled so as to show the cumulative time of all the previous child processes.
- The command name process **COMMAND**

Let us now go through some keystrokes that are helpful for system admins while using the top display.

Key	Purpose
? or h	Display the help section
l, t, m	Header lines of memory, load and threads are toggled
1	Toggle to show individual CPU or all CPUs
s	Change the refresh rate of the screen in seconds
b	The default for running process is a bold highlight. This toggles reverse highlighting
B	Bold can be enabled in the header, display, and for running processes
H	Used to toggle threads to show individual threads or a summary of the processes
u, U	Used to filter for a username
M	Processes are sorted by memory usage in descending order
P	Processes are sorted by processor usage in descending order
k	Kills a process. When prompted, enter PID and signal
r	Renice a process. When prompted, enter PID and nice_value

w	Save or write the current display configuration when you launch top again
q	Quit

Chapter Seven: Services and Daemons in Linux

In this chapter, we will learn now to control and monitor network related services and system daemons using the systemd utility. By the end of this chapter, you will be able to list system daemons and network services, which are started by the systemd service and socket units. You will also be able to control networking services and system daemons using the systemctl command line utility.

Identifying System Processes Started Automatically

In this section, we will learn about system processes such as system daemons and network services that are automatically invoked by the Linux system when it initiates the **systemd** service and socket units.

What is systemd?

When your Linux system boots up, all the processes that are invoked at startup are managed by **systemd**, which is the System and Service Manager. The program includes methods that call and activate system resources, server daemons, and other relevant processes, both when the system is booting up and then later running.

Processes that are waiting or running in the background and performing different tasks are known as **daemons**. Generally, daemons are invoked automatically during the boot process. and they shut down only when the system shuts down, or if they are stopped exclusively. The naming convention for any daemon maintains that the name of the daemon ends with the letter **d**.

A socket is something that is used by a daemon to listen to connections. A socket is the primary channel for communication with both local and remote clients. A daemon can create a socket and can be separated from the socket as well such that they get created by other processes like systemd. When a connection is established with the client, the daemon takes control over the socket.

A service implies one or more daemons. However, the state of the system may be changed as a one time process by starting or stopping a process. This does not involve keeping a daemon process in the running state afterward. This is known as **oneshot**.

Let us go through some history about how systemd was created. For many years now, the process ID 1 in Linux system was dedicated to a process known as **init**. This is the process, which was invoked first during boot up and was responsible for starting all other processes and services on the system. The term “init system” got its origin from this process. The daemons that would be needed frequently would be initiated on the system at boot up

by using the LSB init scripts. These scripts are shell scripts and you can expect variations based on the Linux system that you are on. If there were daemons, which were seldom used, they would be started by other services such as initd or xinetd, which listen to connections from clients. These previous systems had a lot of limitations and were later addressed by introducing systemd.

In Red Hat Enterprise Linux 7, the process ID 1 is assigned to systemd, which is the modern initd process. Let us go through the features of systemd one by one.

- The boot speed of the system was increased because of parallel processes.
- Daemons could be started on demand without needing other services to start them.
- Linux control groups, which made way for tracking related processes together.
- Service dependency management was automated, which helped reduce timeouts by preventing a network service from starting when it was not available.

systemd and systemctl

Different types of systemd objects known as units can be managed using the systemctl command. You can use the **systemctl -t help** command to list down all the unit types. Let us go through a few common unit types.

- Service units, which represent services of the system have a .service extension. This unit will be used to start daemons like a web server, which are frequently accessed.
- The inter process communication sockets IPC are represented y socket units, which have the .service extension. When a connection is made by the client, the control of the socket is transferred to the daemon. At booth time, the start of a service can be delayed using socket units or to start services, which are not used very frequently.
- Path units, which have the extension .path are used to delay the

initiation of a service until a desired change occurs in the file system. This is used for services, which use the spool directory such as the printer service.

Service states

The command **systemctl service name.type** can be used to view the status of a service. If the tpe of the unit is not provided, systemctl will show the status of a service unit.

The output of the command has certain keywords that are interesting to a system admin, which will indicate the state of a service. Let us go through these keywords one by one.

Keyword	Description
loaded	The unit configuration file has been processed
active (running)	The service is running and has one or more processes continuing
active (exited)	A one time configuration has been executed successfully
active (waiting)	Service is in running state but is waiting for an event
inactive	Not running
enabled	The service will start at boot time
disabled	The service will not start at boot time
static	Can not be enabled but can be started automatically by another enable unit

Note: The **systemctl status NAME** command replaces the command **service NAME status, which** was used in the previous version of Red Hat Enterprise Linux.

Let us go through an example where we will list files using the systemctl command

1. Verify the system startup by querying the state of all units

   [root@desktop ~]# systemctl

2. Query the state of service units only

   [root@desktop ~]# systemctl --type=service

3. Check units, which are in the maintenance or failed state. Use the -l option to show full output

[root@desktop ~]# systemctl status rngd.service –l

4. The status argument can be passed to see if a service is active or if it will be made active during the boot process. There are alternative commands to show the active or enabled states as well

[root@desktop ~]# systemctl is-active sshd
[root@desktop ~]# systemctl is-enabled sshd

5. List the active state of all units that are currently loaded. You can also filter the type of unit. Additionally, you can use the --all option to show unites that are inactive as well

   [root@desktop ~]# systemctl list-units --type=service
   [root@desktop ~]# systemctl list-units --type=service --all

6. View the setting for all units with respect to enabled or disabled. As an option, filter the type of unit

[root@desktop ~]# systemctl list-unit-files --type=service

7. Show only failed units

```
[root@desktop ~]# systemctl --failed --type=service
```

Controlling System Services

In this section, we will learn how to control network services and system daemons using the systemctl command line utility.

Starting and Stopping system daemons

When you make changes to a configuration file of a service, it is necessary that you restart the service for those changes to be reflected in the system. A service that you do not wish to use anymore can be stopped before you uninstall the software that is related to the service. As a system admin, you may sometimes want to start a service manually only when it is needed. We will go through an example, which will show you how to start, stop, and restart a service.

1. Firstly, let us see the status of a service

   [root@desktop ~]# systemctl service sshd.service

2. Let us check if the service is in the running state

   [root@desktop ~]# ps -up PID

3. Let us now stop the service and check its status

   [root@desktop ~]# systemctl stop sshd.service
   [root@desktop ~]# systemctl status sshd.service

4. Let us now start the service again and check its status. Also notice that the process ID PID would have changed for the service

   root@desktop ~]# systemctl start sshd.service
   root@desktop ~]# systemctl status sshd.service

5. Stop the service and then start it again using just one command

   root@desktop ~]# systemctl restart sshd.service
   root@desktop ~]# systemctl status sshd.service

6. Without making the service stop and start again completely, make the

service read the new configuration in the configuration file. This will not change the process ID PID

root@desktop ~]# systemctl reload sshd.service
root@desktop ~]# systemctl status sshd.service

Some service may start as dependencies of other services. If there is a socket unit that is enabled, but a service unit is not available with the same name, the service will start automatically when there is a request made at the network socket. When a condition for file system is met, services may also get triggered by path units. For example, if you are placing a file in the print spool directory, it will automatically start the cups service if it was not already running.

root@desktop ~]# systemctl stop cups.service

Warning: Stopping cups, but it can be activated by:

cups.path

cups.socket

You will need to stop all three units in order to stop printing on the system completely. You can disable the service, which will, in turn, disable its dependencies. The command **systemctl list-dependencies UNIT** can be used to print out a tree, which will show all the other units that need to be started in order for the specified unit to work. Depending upon the need, the dependency may need to already be in a running state or start after the specified uni has started. Conversely, if you use the **--reverse** option with a specified unit, you will come to know, which other units need the specified unit as a dependency for them to run.

There will be times when there are conflicting services that are installed on the system. For example, networks can be managed via different methods such as NetworkManager and network. Also, firewalls can be managed using a couple of services such as firewalld and iptales. A network service can be masked to prevent it from starting accidentally by a system admin. When you mask a service, a link is created for the service in its configuration directory such that nothing happens even if you launch it.

root@desktop ~]# systemctl mask network

Ln -s ‘/dev/null’ ‘/etc/systemd/system/network.service’

root@desktop ~]# systemctl unmask network

Rm ‘/etc/systemd/system/network.service’

Note: A services that has been disabled will not launch automatically at boot process. It needs to be started manually. Also, a masked service cannot be started manually or automatically.

Enabling System Daemons to Start or Stop at Boot

When you start a service on a system that is already up and running does not guarantee that the service will start automatically again when you restart the system. Conversely, if you manually stop a service when the system was up would not mean that will not start again automatically if the system is restarted. When you create appropriate links in the systemd configuration directories, services are automatically started during the system's boot process. The systemctl command is used to create and delete these links.

Let us go through some examples, which will give us an idea about how to make system daemons start or stop during the boot process.

1. Let us first view the status of a service
   [root@desktop ~]# systemctl service sshd.service
2. Let us now disable the service and check its status. The service does not stop if you disable it

root@desktop ~]# systemctl disable sshd.service
root@desktop ~]# systemctl status sshd.service

3. Let us enable the service again and check its status
   root@desktop ~]# systemctl enable sshd.service
   root@desktop ~]# systemctl is-enabled sshd.service

Let us summarize all the **systemctl** commands that we have learned. Systemctl command helps to start or stop a service or enable or disable a service during boot time.

Task	Command
Get a unit's state's detailed information	systemctl status UNIT
Stop a service on a running system	systemctl stop UNIT
Start a service on a running system	systemctl start UNIT
Restart a service on a running system	systemctl restart UNIT

Reload the configuration file of a running service	systemctl reload UNIT
Disable a service from starting at boot or manually	systemctl mask UNIT
Make a masked service available	systemctl unmask UNIT
Enable a service to start at boot	systemctl enable UNIT
Disable a service from starting at boot	systemctl disable UNIT
List dependencies of a particular unit	systemctl list-dependencies UNIT

Chapter Eight: OpenSSH Service

In this chapter, we will learn how to configure and secure the openSSH service. The openSSH service is used to access Linux systems using the command line. By the end of this chapter, we will learn how to log into a remote system using SSH and how to run a command on the shell prompt of the remote system. We will also learn how to configure the SSH service to implement password free login between two systems by using a private key file for authentication. We will learn how to make SSH secure by configuring it to disable root logins and to disable password based authentication as well.

Using SSH to Access the Remote Command Line

In this section, we will learn how we can log in to a remote system using ssh and run commands on the shell prompt of the remote system.

What is OpenSSH secure shell (SSH)?

The term OpenSSh refers to the software implementation in Linux systems known as Secure Shell. The terms OpenSSH, ssh, Secure Shell, which are synonymous with each other is an implementation that lets you run shell on a remote system in a very secure manner. If you have a user configured for you on a remote Linux system, which also has SSH services, you can remotely login to the system using ssh. You can also run a single command on a remote Linux system using the ssh command.

Let us go through some example of the secure shell. They will give you an idea of the syntax used for remote logins and how to run commands on the remote shell.

- Login using ssh on a remote shell with the current user and then use the exit command to return to your original shell

  ```
  [student@desktop ~]$ ssh remotesystem
  stident@remotesystem's password:
  [student@remotesystem ~]$ exit
  Connection to remotesystem closed.
  [student@desktop ~]$
  ```

- Connect to a remote shell as a different user (remoteuser) on a remote system

  ```
  [student@desktop ~]$ ssh remoteuser@remotesystem
  remoteuser@remotesystem's password:
  [remoteuser@remotesystem ~]$
  ```

- Execute a single command on the remote system as a remote user

```
[student@desktop ~]$ ssh remoteuser@remotesystem hostname
remoteuser@remotesystem's password
remotesystem.com
[student@desktop ~]$
```

The **w** command that we learned about previously displays all the users that are currently logged into the system. The FROM column of the output of this command will let you know if the user who is logged in is from the local system or from a remote system.

SSH host keys

The communication between two systems via ssh is secured through public key encryption. A copy of the public key is sent by the server to the client before the ssh client connects to the server. This method is used to complete the authentication of the server to the client and also to set up a connection using secure encryption.

When you try to ssh into a remote system for the first time, the ssh command stores the public key of the server in your ~/.ssh/known_hosts file. Every time after this, when you try to login to the remote system, the server sends a public key and compares it to the public key that is stored in you ~/.ssh/known_hosts file. If the keys match, a secure connection is established. If the keys do not match, it is assumed that the connection attempt was altered by some hijacking the connection and connection is closed immediately.

If the public key of the server is changed for reasons such as loss of data on the hard drive or if the hard drive was replaced for a genuine reason, you will need to remove the old entry of the server' public key from ~/.ssh/known_hosts file and replace it with the new public key.

- Host IDs are stored on your local system at ~/.ssh/known_hosts
  You can **cat** this file to see all the public keys of remote hosts stored on your local system.
- The keys of the host are stored on the SSH server at /etc/ssh/ssh_host_key*

SSH Based Authentication

In this section, we will learn how to setup a secure login via ssh without using password based authentication and by enabling key based logins using the private key authentication file.

SSH key based authentication

There is a way to authenticate ssh logins without using passwords through a method known as public key authentication. A private-public key pair scheme can be used by users to authenticate their ssh logins. There are two keys generated. One is a private key and a public key. The private key file must be kept secret and in a secure location as it is like a password credential. The public key is copied to the system that a user may want to login to and is used to verify and match with the private key. There is no need for the public key to be a secret. An SSh server, which has your public key stored on it can issue a challenge, which will only be met by a system that has your private key on it. Therefore, when you log in from your system to a server, your private key will be present on your system, which will match the public key on the server resulting into a secure authentication. This is a method that is secure and does not require you to type your password to login every time.

You can use the **ssh-keygen** command on your local system to generate your private and public key pair. The private key is then generated and kept at ~/.ssh/id_rsa and the public gey is generated and kept at ~/.ssh/id_rsa.pub.

Note: When you are generating your keys, you are given an option to set a passphrase as well. In the event that someone steals your private key, they will not be able to use your private key without a passphrase since only you would know the passphrase. This additional security measure will give you enough time to set a new key pair before the attacker cracks your private key, knowing that your existing private key is stolen.

Once you have generated the SSH keys, they will be stored in the user's home directory under /.ssh. You then need to copy your public key to the destination system with, which you want to establish key based authentication. You can do this by using the **ssh-copy-id** command.

[student@system1 ~]$ ssh-copy id student@system2

When you use the command **ssh-copy-id** to copy your public key from your system to another system, it automatically copies your public key from ~/.ssh/id_rsa.pub

Customizing the SSH Configuration

In this section, we will learn how to customize the sshd configuration such that we can restrict password based logins or direct logins.

It is not really necessary to configure the openSSH service but there are options available to customize it. All the parameters of the sshd service can be configured in the file that is located at /etc/ssh/sshd_config.

Prohibiting root user logins from SSH

With respect to security, it is advisable that we restrict the root user to login directly using the ssh service.

- The user name root is available on every Linux system. If you allow root logins, an attacker only needs to know the root user's password to be able to login as root via ssh. Therefore, it is good practice to not allow root logins via ssh at all.
- The root user is a super user with unlimited privileges, and therefore, it makes sense to not allow the root user to login using ssh.

The ssh configuration file has a line, which we can comment out to restrict root user logins. You need to edit the file /etc/ssh/sshd_config and comment out the following line:

#PermitRootLogin yes

When to comment out the line for permitting root login, the root user will not be able to login using the ssh service once the sshd service has been restarted.

PermitRootLogin no

For the changes in the configuration file to come into effect, you will need to restart the sshd service

root@desktop ~]# systemctl restart sshd

Another option available it to allow only key based login where you can edit the following line into the file.

PermitRootLogin without-password

Prohibiting password authentication during ssh

There are many advantages of allowing only key based logins to a remote system.

- The length of an SSH key is longer than a password and therefore, it is more secure.
- Once you have completed the initial setup, there is hardly any time taken for the future logins.

You need to edit the file /etc/ssh/sshd_config where there is a line that allows password authentication by default.

PasswordAuthentication yes

To stop password authentication, you need to edit this line to no and then restart the sshd service.

PasswordAuthentication no

Always make sure that after you have modified the sshd service configuration file at /etc/ssh/sshd_config you will need to restart the sshd service

root@desktop ~]# systemctl restart sshd

Chapter Nine: Log Analysis

In this chapter, we will learn how to locate logs in the Linux system and interpret them for system administration and troubleshooting purposes. We will describe the basic architecture of syslog in Linux systems and learn to maintain synchronization and accuracy for the time zone configuration such that timestamps in the system logs are correct.

Architecture of System Logs

In this section, we will learn about the architecture of system logs in Red Hat Enterprise Linux 7 system.

System logging

Events that take place as a result of processes running in the system and the kernel of the operating system need to be logged. The logs will help in system audits and to troubleshoot issues that are faced in the system. As a convention, all the logs in Linux based systems are stored at **/var/log** directory path.

Red Hat Enterprise Linux 7 has a system built for standard logging by default. This logging system is used by many programs. There are two services **systemd-journald** and **rsyslog, which** handle logging in Red Hat Enterprise Linux 7.

The **systemd-journald** collects and stores logs for a series of process, which are listed below.

- Kernel
- Early stages of the boot process
- Syslog
- Standard output and errors of various daemons when they are in the running state

All these activities are logged in a structural pattern. Therefore, all these events get logged in a centrally managed database. All messages relevant to syslog are also forwarded by **systemd-journald** to **rsyslog** to be processed further.

The messages are then sorted by rsyslog based on facility or type and priority and then writes them to persistent files in **/var/log** directory.

Let us go through all the types of logs, which are stored in /var/log based on the system and services.

Log file	Purpose
/var/log/messages	Most of the syslog messages are stored in this file with the exception of messages related to email processing and authentication, cron jobs and debugging related errors
/var/log/secure	Errors related to authentication and security are stored in this file
/var/log/maillog	Mail server related logs are stored in this file
/var/log/cron	Periodically executed tasks known are known as cron. Related logs are stored in this file
/var/log/boot.log	Messages that are associated with boot up are stored here

Syslog File Review

In this section, we will learn how to review system logs, which can help a system admin to troubleshoot system related issues.

Syslog files

The syslog protocol is used by many programs in the system to log their events. The log message is categorized by two things.

- Facility, which is the type of message
- Priority, which is the severity of the message

Let us go through an overview of the priorities one by one.

Code	Priority	Severity
0	emerg	The state of the system is unusable
1	alert	Immediate action needs to be taken
2	crit	The condition is critical
3	err	The condition is non-critical with errors
4	warning	There is a warning condition
5	notice	The event is normal but significant
6	info	There is an informational event
7	debug	The message is debugging-level

The method to handle these log messages is determined by the priority and the type of the message by rsyslog. This is already configured in the file at /etc/rsyslog.conf and by other conf files in /etc/rsyslog.d. As a system admin, you can overwrite this default configuration and customize the way rsyslog

file to be able to handle these log messages as per your requirement. A message that has been handled by the rsyslog service can show up in many different log files. You can prevent this by changing the severity field to none so that messages directed to this service will not append to the specified log file.

Log file rotation

There is **logrotate** utility in place in Red Hat Enterprise Linux 7 and other linux variants so that log files do not keep piling up the /var/log file system and exhaust the disk space. The log file gets appended with the date of rotation when it is rotated. For example, and old file named /var/log/message will change to /var/log/messages-20161023 if the file was rotated on October 23, 2016. A new log file is created after the old log file is rotated and it is notified to the relevant service. The old log file is usually discarded after a few days, which is four weeks by default. This is done to free up disk space. There is a cron job in place to rotate the log files. Log files get rotated on a weekly basis, but this may vary based on the size of the log file and could be done faster or slower.

Syslog entry analysis

The system logs, which are logged by the rsyslog program have the oldest log message at the top of the file and the latest message at the end of the file. There is a standard format that is used to maintain log entries that are logged by rsyslog. Let us go through the format of the /var/log/secure log file.

Feb 12 11:30:45 localhost sshd[1432] Failed password for user from 172.25.0.11 port 59344 ssh2

- The first column shows the timestamp for the log entry
- The second column shows the host from, which the log message was generated
- The third column shows the program or process, which logged the event
- The final column shows the message that was actually sent

Using the tail command to monitor log files

It is a common practice for system admins to reproduce the issue so that error logs for the issue get generated in real time. The **tail -f /path/to/file** command can be used to monitor logs that are generated in real time. The last 10 lines of the log file are displayed with this command while it still continues to print new error logs that are generated in real time. For example, if you wanted to look for real time logs of failed login attempts, you can use the following tail command, which will help you see real time logs.

[root@desktop ~]# tail -f /var/log/secure

…

Feb 12 11:30:45 localhost sshd[1432] Failed password for user from 172.25.0.11 port 59344 ssh2

Using logger to send a syslog message

You can send messages to the rsyslog service by using the **logger** command. The command useful when you have made some changes to the configuration file of rsyslog and you want to test it. You can execute the following command, which will send a message that gets logged at /var/log/boot.log

[root@desktop ~]# logger -p local7.notice “Log entry created”

Reviewing Journal Entries for Systemd

In this section, we will learn to review the status of the system and troubleshoot problems by analyzing the logs in the systemd journal.

Using journalctl to find events:

The systemd journal uses a structured binary file to log data. Extra information about logged events is included in this data. For syslog events, this contains the severity and priority of the original message.

When you run **journalctl** as the root user, the complete system journal is shown, starting from the oldest log entry in the file.

Messages of priority notice or warning are highlighted in bold by the journalctl command. The higher priority messages are highlighted in red color.

You can use the journalctl command successfully to troubleshoot and audit is to limit the output of the command to only show relevant output.

Let us go through the various methods available to limit output of the journalctl command to show only desired output.

You can display the last 5 entries of the journal by using the following command.

[root@server ~]# journalctl -n 5

You can use the priority criteria to filter out journalctl output to help while troubleshooting issues. You can use the **-p** option with the journalctl command to specify a name or a number of the priority levels, which shows the entries that are of high level. journalctl know the priority levels such as info, debug, notice, err, warning, crit, emerg, and alert.

You can use the following command to achieve the above mentioned output.

[root@server ~]# journalctl -p err

There is command for journalctl similar to tail -f, which is **journalctl -f**. This will again list the last 10 lines of journal entry and then keep printing log entries in real time.

[root@server ~]# journalctl -f

You can also use some other filters to filter out journalctl entries as per your requirement. You can pass the following options of **--since** and **--until** to filters out journal entries as per timestamps. You need to then pass the arguments such as **today, yesterday** or an actual timestamp in the format **YYYY-MM-DD hh:mm:ss**

Let us look at a few examples below.

[root@server ~]# journalctl --since today

[root@server ~]# journalctl --since “2015-04-23 20:30:00” --until “2015-05-23 20:30:00”

There are more fields attached to the log entries, which will be visible only if you use the verbose option by using **verbose** with the journalctl command.

[root@server ~]# journalctl -o verbose

This will print out detailed journalctl entries. The following keywords are important for you to know as a system admin.

- _COMM, which is the name of the command
- _EXE show the executable path for the process
- _PID will show the PID of the process
- _UID will show the user associated with the process
- _SYSTEMD_UNIT show the systemd unit, which started the process

You can combine one or more of these options to get an output from the journalctl command as per your requirement. Let us have a look at the example below, which will print journal entries that contain the systemd unit file sshd.service bearing the PID 1183.

[root@server ~]# journalctl _SYSTEMD_UNIT=sshd.service _PID=1183

Systemd Journal Preservation

In this section, we will learn how to make changes to the **systemd-journald** configuration such that the journal is stored on the disk instead of memory.

Permanently storing the system journal

The system journal is kept at **/run/log/journal** by default, which means that when the system reboots, the entries are cleared. The journal is a new implementation in Red Hat Enterprise Linux 7.

We can be sure that if we create a directory as **/var/log/journal**, the journal entries can be logged there instead. This will give us an advantage that historical data will be available even after a reboot. However, even though we will have a journal that is persistent, we cannot have any data that can be kept forever. There is a log rotation, which is triggered by a journal on a monthly basis. Also, by default, the journal is not allowed to have a dusk accumulation of more than 10% of the file system it occupies, or even leave less than 15% of the file system free. You can change these values as per your needs in the configuration file at **/etc/systemd/journald.conf** and one the process for systemd-journald starts, the new values will come into effect and will be logged.

As discussed previously, the entries of the journal can be made permanent by creating a directory at /var/log/journal

[root@server ~]# mkdir /var/log/journal

You will need to make sure that the owner of the /var/log/journal directory is root and the group owner is systemd-journal, and the directory permission is set to 2755.

[root@server ~]# chown root:systemd-journal /var/log/journal

[root@server ~]# chmod 2755 /var/log/journal

For this to come into effect, you will need to reboot the system or as a root user, send a special signal **USR1** to the systemd-journald process.

[root@server ~]# killall -USR1 systemd-journald

This will make the systemd journal entries permanent even through system reboots, you can now use the command **journalctl -b** to show minimal output as per the latest boot.

[root@server ~]# journalctl -b

If you are investigating an issue related to system crash, you will need to filter out the journal output to show entries only before the system crash happened. That will ideally be the last reboot before the system crash. In such cases, you can combine the **-b** option with a negative number, which will indicate how many reboots to go back to limit the output. For example, to show outputs till the previous boot, you can use **journalctl -b -1**.

Maintaining time accuracy

In this section, we will learn how to make sure that the system time is accurate so that all the event logs that are logged in the log files show accurate timestamps.

Setting the local time zone and clock

If you want to analyze logs across multiple systems, it is important that the clock on all those systems is synchronized. The systems can fetch the correct time from the Internet using the Network Time Protocol NTP. There are publicly available NTP projects on the Internet like the Network pool Project, which will allow a system to fetch the correct time. The other option is to maintain a clock made up of high quality hardware to serve time to all the local systems.

To view the current settings for date and time on a Linux system, you can use the **timedatectl** command. This command will display information such as the current time, the NTP synchronization settings and the time zone.

[root@server ~]# timedatectl

The Red Hat Enterprise Linux 7 maintains a database with known time zones. It can be listed using the following command.

[root@server ~]# timedatectl list-timezones

The names of time zones are based on zoneinfo database that IANA maintains. The naming convention of time zones is based on the ocean or continent. This is followed by the largest city in that time zone or region. For example, if we look at the Mountain Time in the USA, it is represented as “America/Denver”.

It is critical to select the correct name of the city because sometimes even regions within the same time zone may maintain different settings for daylight savings. For example, the US mountain state of Arizona does not have any implementation of daylight savings and therefore falls under the time zone of “America/Phoenix”.

The **tzselect** command is used to identify zone info time zone names if they

are correct or not. The user will get question prompts about their current location and mostly gives the output for the correct time zone. While suggesting the time zone, it will not automatically make any changes to the current time on the system. Once you know, which timezone, you should be using, you can use the following command to display the same.

[root@server ~]# timedatectl set-timezone America/Phoenix

[root@server ~]# timedatectl

If you wish to change the current date and time for your system, you can use the **set-time** option with the timedatectl command. The time and date can be specified in the format “”YYYY-MM-DD hh:mm:ss”. If you just want to set the time, you can omit the date parameters.

[root@server ~]# timedatectl set-time 9:00:00

[root@server ~]# timedatectl

You can use the automatic time synchronization for Network Time Protocol using the **set-ntp** option with the timedatectl command. The argument to be passed along is **true** or **false, which** will turn the feature on or off.

[root@server ~]# timedatectl set-ntp true

The Chronyd Service

The local hardware clock of the system is usually inaccurate. The **chronyd** service is used to keep the local clock on track by synchronizing it with the configured Network Time Protocol NTP servers. If the network is not available it synchronizes the local clock to the RTC clock drift that is calculated and recorded in the **driftfile, which** is maintained in the configuration file at **/etc/chrony.conf.**

The default behavior of the chronyd service is to use the clocks from the NTP network pool project to synchronize the time and no additional configuration is needed. It is advisable to change the NTP servers if your system happens to be on an isolated network.

There is something known as a **stratum** value, which is reported by an NTP time source. This is what determines the quality of the NTP time source. The stratum value refers to the number of hops required for the system to reach a high performance clock for reference. The source reference clock has a stratum value of 0. An NTP server that is attached to the source clock will have a stratum value of 1, while a system what is trying to synchronize with the NTP server will have a stratum value of 2.

You can use the **/etc/chrony.conf** file to configure two types of time sources, **server** and **peer**. The stratum level of the server is one level above the local NTP server. The stratum level of the peer is the same as that of the local NTP server. You can specify one or more servers and peers in the configuration file, one per line.

For example, if your chronyd service synchronizes with the default NTP servers, you can make changes in the configuration file to change the NTP servers as per your need. Every time you change the source in the configuration file, you will need to restart the service for the change to take effect.

[root@server ~]# systemctl restart chronyd

The chronyd service has another service known as **chronyc, which** is a client to the chronyd service. Once you have set up the NTP synchronization, you may want to know if the system clock synchronizes correctly to the NTP

server. You can use the **chronyc sources** command or if you want a more detailed output, you use the command **chronyc sources -v** with the verbose option.

[root@server ~]# chronyc sources –v

Chapter Ten: Archiving Files

In this system, we will learn how to compress files and archive them. We will also learn to extract the compressed file. We will learn the different compression techniques that are available in Linux and why compression is necessary and how it is useful to make the life a system admin easy.

Managing Compressed Archives

In this section, we will learn about the tar command and how it is used to compress and archive files. We will also learn how to use the tar command to extract data from existing archived files.

What is tar?

Compression and archiving of files is useful for taking backups of data and for transferring huge files from one system to another over a network. The **tar** command can be used to achieve this, which is the most common and one of the oldest methods used to archive and compress files on the Linux system. With the tar command, you can compress an archive using the **gzip, xz,** or **bzip2** compression.

The tar command is accompanied by one of the following three actions.

- **c, which** is used to create an archive
- **t, which** is used to list the content of an archive
- **X, which** is used to extract from an existing archive

The options that are commonly used along with the tar command are as follows.

- **f file name, which** will be the file that you want to use
- **v** stands for verbosity, which shows the list of files getting archived or being extracted

Using tar to archive files and directories

If you are looking to create a tar archive, you need to ensure that there is no existing archive with the same file name as the one that you intend to create because the tar command will not give you any prompt and will overwrite the existing archive file.

You will need to use the **c** option to create a new archive followed by **f file name**.

```
[root@server ~]# tar cf archive.tar file1 file2 file3
```

This command will create an archive file named archive.tar, which will contain the files file1, file2, and file3.

Listing contents of a tar file

The **t** and **f** options are used with the tar command to list the contents of a tar file.

[root@server ~]# tar tf /root/etc/etc.tar

etc/

etc/fstab

etc/mtab

…

Using tar to extract an existing archive

The **x** and **f** options are used with the tar command to list the contents of a tar file.

[root@server ~]# tar xf /root/etc/etc.tar

Adding a **p** to the option ensures that all permissions are preserved after extraction

[root@server ~]# tar xpf /root/etc/etc.tar

Creating a compressed tar archive

There are three types of compression techniques that can be used with the tar command. They are as follows.

- **z, which** is used for a gzip compression with filename.tar.gz or filename.tgz extension
- **j** used for bzip2 compression with a filename.tar.bz2 extension
- **J** used for a xz compression with a filename.tar.xz extension

You can pass one of these options with the regular tar create command to get the required compressed archive.

Example: If you wish to create a tar archive with a gzip compression, you can use the following command.

[root@server ~]# tar czf new.tar.gz /etc
This will create a compressed archive of the content of the /etc directory and name it *new.tar.gz.*

Extracting a compressed tar archive

You can use the **x** option with the tar command and pass one of the compression options along with it to extract the contents of a tar archive.

Example: If you wish to extract a tar archive file, which has a gzip compression, you can use the following command.

[root@server ~]# tar xzf /root/etc/etc.tar.gz

This will extract all the contents from the compressed archive at */root/etc/etc.tar.gz* and place all its files in the home directory of the root user since that is the present working directory of the root user.

Conclusion

User data is the most expensive entity in the world today. Compromise in data can result in huge losses for an organization. You can maintain a computer at home with a reasonable amount of security such as a simple antivirus software. However, given the amount of data that is present on business related systems on the Internet, they are more prone to attackers, and therefore, the level of effort to maintain security on business machines is way more than a personal computer. But Linux operating systems have proved to be a secure platform for a choice of the operating system on server systems for big organizations. Given the open-source nature of the Linux operating system development, security patches come faster for Linux than they come out for any other commercial operating systems, making Linux the most ideal platform with respect to security.

All this said and done, what comes into the spotlight is the job profile of a Linux system administrator. There is a huge demand for this profile in all the major organizations worldwide, which work on Linux systems. This book provides a beginner’s course to the Linux system and we hope that it will encourage you to learn advanced Linux system administration in the future.

Linux Command Line

Beginners Guide to Learn Linux Commands and Shell Scripting

David A. Williams

Introduction

I decided to write this book to ease the hurdles that newbie Linux users face. I want you to understand your computer in a better way than you did before. This book contains a complete package for beginners to understand what the Linux operating system is and how it differs from other operating systems. Linux offers you a great learning experience.

You can understand the system configuration by skimming a couple of text files that are fully readable. Just remember the individual status of each component and its role when you put together the bigger picture. That's it. Confused. Don't worry. Keep reading until the end of the book and you will understand.

Why should Linux be preferred over other operating systems?
This is what should be answered at the beginning so that you can have the stimulus while you walk through the upcoming chapters of this book. Look, we are not living in the '80s or the '90s. The world has changed so much over the past few years and with it has changed the cyber world. Now all the continents are connected to one another through computer networks. From the oldest to the youngest, all users have access to the Internet. In the nooks and crannies of this cyber world there are humongous data bases that are developed by big businesses. There are billions of web pages that are processing information on a per second basis.

Coupled with these facts are the dangers linked to this ubiquitous connectivity. This is the age when you need a computer on which you have customized applications and complete control. Would you hand over control of your computer system to other companies who keep making a profit by marketing how easy they have made the use of computers? Or would you rather have more freedom and control over your own computer? You deserve freedom to customize control over your computer. You deserve to build your own software for your computer systems. That's why you should prefer Linux over all the other operating systems. With Linux on your computer, you can direct your computer to do as you wish. It will act on your

commands.

Command line is the best option

We are trapped in the ease of using computers. We are given attractive graphical user interfaces to deal with which have made us lazy and robbed us of innovation and creativity. This should be done away with. The mouse is not the way to run a computer. It is the keyboard which makes you an expert in using computers. Linux differs from other operating systems in a sense that it offers the Command Line Interface (CLI) instead of the Graphical User Interface. When we talk of the Command Line Interface (CLI), the first thing that comes to the mind of most people is a dark screen. For most people, it is a horrible thought as compared to using a graphical user interface, but this is where your power starts. The CLI allows you greater freedom to talk to your computer and direct it to do certain tasks. You enter commands on the CLI and the computer interprets them and executes them. While the command line interface seems difficult to use - which it no doubt is - it makes difficult tasks easier and faster.

Who should read this book?

You don’t have to be a master of programming to read this book. This book is for beginners who are thinking of dipping their toes into the world of Linux. What I expect from you before you go on to the next chapters is a basic understanding of computers. It means that you should be able to tinker with the graphical user interface and finish some key tasks. You should know about booting, startup, log in, files and directories. You should know how to create, save and edit documents and how to browse the web space. But the most important thing of all is your will to learn new things. I’ll take you from there.

This book also is for those who are fed up with other operating systems and want to switch to a smarter operating system. I’ll tell you about Sylvia, my friend, in the latter chapters, who was forced by her boss to learn Linux for execution of key office tasks. I’ll tell the tale of how hard it was for her and how she achieved such a gigantic goal.

If you have just come to know about Linux and want to switch to this unique operating system, this book is definitely for you. Read it, learn the command line and get started.
Before starting the journey, you should bear in mind that there is no shortcut to mastering Linux. Like all great and exciting things, learning Linux takes time. It may turn out to be challenging at times, and may also take great effort on your part. The command line is simple. Also, you can easily remember the commands and the syntax.

What makes it tough to master is its broadness. It is so wide to grasp in a short time. But as with all big things, practice makes you perfect. If you keep persevering, you will be able to learn its use and apply it accordingly. The only thing I demand from you as a beginner determination, the ability to tackle failure, and a responsible attitude. A casual approach is not the best way to tackle Linux when you are trying to learn.

What this book has to offer

This book is filled with material that is easy to read and practice, and perfectly suits starters. It is like learning from a tutor. Every step is explained with the help of a command line exercise and a solution for you to understand the process.

- The first section explains what the shell is. You will learn about the different components of the Linux system. In addition, there will be various commands to enter in the shell window for your immediate practice. This section also explains how to navigate the filesystem and different directories in a Linux operating system.

- The second section will take you further into the command line by explaining more about the commands. You will also be able to create your own commands.

- The third section is packed with more details about the Linux environment and system configuration. It will explain how GRUB works on Linux. You will learn about the system startup, the *init* and different runlevels of Linux.

- The fourth section talks about the package management, the repositories and the dependencies. It educates on managing the existing file systems and creating new ones. In addition, it fully explains how you can handle the storage devices using a Linux operating system. You will find an example of editing a partiti0n and altering its type with the help of command line tools.

- The fifth section carries details on the Linux environment variables.

- In the sixth section, you will learn the basics of shell scripting. This is like writing your own software. You will learn about key scripting techniques by which you can control your computer like teaching your computer about decision-making. It is almost akin to artificial intelligence. Your computer will act on its own following a set of instructions. Furthermore, you will learn about some important commands like the case statements, the break statement and the continue statement.

- The final section will take you to the advanced level of shell scripting.

Pre-requisites for reading the book

You will need an operational Linux system on your computer before starting

to read this book because it carries practical exercises and their solutions for you.

You can install Linux on your computer. There are a number of distributions, like Fedora, OpenSUSE and Ubuntu. Installing the distribution system is very easy if done in the right way. Of course, there are some specifications that you must have on your computer. For instance, at least 256 MB RAM and around 6GB free space on the hard disk. The higher the specifications of the system the better it is for the operations. It is recommended that you use a personal computer and a landline internet connection. Wireless systems are not suitable for the job.

You can also use the Linux distribution system from a live CD, so there is no need to install it on the system. That's less messy, to say the least. It is easier to do that. On the startup, enter the setup menu and change the boot settings. Switch your boot option to boot from a CD instead of the hard disk. After that insert the CD and you are all ready to use the Linux distribution system. You can run Ubuntu and Fedora from a live CD.

How this book can help you

This book is pretty helpful for those who want to wrap up administrative tasks in a faster way using the Linux environment. You will be able to write your own scripts in order to accomplish specific tasks. After reading this book, you will be able to automate certain administrative tasks with the help of shell scripts. File management, statistical data management and complex arithmetic functions will be a lot easier after you understand some key Linux commands.

Chapter 1: Starting with the Linux Shell

Linux is everywhere from our cell phones to computers. Linux got started in the mid ‘90s. Within a very small window of time, it spread across many industries. Stock markets and super computers also use this operating system. Its popularity is unparalleled. Linux is basically an operating system just like any other, which means it manages the software you install on your system and manages the communication between different pieces of hardware. Seems interesting yet! Let’s roll on.

What is Linux?

Before moving on to the complex parts, it is better that you sail through the basic world of Linux first. For beginners, understanding Linux can be perplexing if they don’t know what Linux actually is and what it offers.

Though it may appear complex and undoable thing at the start, in reality, it is easier to learn than you may think. The Linux operating system consists of the following parts.
Bootloader: This manages the boot process of a computer. You might have seen a splash screen coming and going in the blink of an eye before you boot into the operating system.

The Kernel: This is also dubbed as the core of the system. It manages the entire system by deploying software and hardware when they are needed. In addition, the kernel manages system memory, as well as the file system.

The Shell: You can give directions to your computer by typing commands in the form of texts. This command line is dubbed as the shell which is the daunting part of Linux. People just don’t want to venture into it. My friend Sylvia never liked Linux until her boss pushed her into this uncanny tech world. Once she got into it, she loved it more than Windows and Mac. The control the command line gave her made her quite comfortable with it at her office. The shell allowed her to copy, move and rename files with the help of the command prompt. All the struggle she had to go through was memorizing

the text commands and the rest of it was the easy part. The command prompt does everything.

You can type in a program name to start it. The shell sends the relevant info to the kernel, and it does the rest of the execution.

Getting Started

After you have installed Linux, create a regular user to run as your personal account. Log in to the account.

The Shell Window

The shell is an integral part of the command line. The command line, in reality, is the shell itself. In simple words it takes keyboard commands as input and then processes them toward the operating system for execution. Different Graphical User Interfaces (GUIs) use different terminal emulators which allow you to put in text commands through the shell. Some newbies get confused by the large number of emulators available online. To clear your mind, just remember that the basic job of all terminal emulators is to give you a gateway to the shell. One more thing: a terminal emulator is also known as the shell window.

Unix also uses a shell system, but Linux has the same system, but it’s better. This enhanced version of the shell is dubbed as “bash” which comes as a default shell on most of the Linux distributions.
After you have launched the emulator, the screen will show you something like the following:

For Ubuntu users it will look like: aka@localhost ~$

For Fedora users it appears like: [aka@localhost ~]$

This text is known as the shell prompt in which aka is the username while ‘localhost’ denotes the machine name. Their values may differ. Usually, the $ sign accompanies you all the time you are in the shell window. If you see the sign of #, it means you are logged in as a root user or the emulator you are using offers super user privileges. If this is your first time with Linux, just open the DOS command prompt on the windows. You will have the same experience while using the shell window. There is not much difference except for the commands. The environment is more or less alike.

Do you want to type something on your screen? Let’s do that.
[aka@localhost ~]$ tera

Did you write your name? Write it down. The shell will analyze the command and respond with the following:

bash: tera : command not found
[aka@localhost ~]$

You cannot use a mouse on the emulator. Instead, the keyboard arrows will help you do some magic. Strike the upper arrow key to scroll through the history of your commands. The downward arrow will bring you to the latest command. Also, the previous command disappears while you get down to the new one. Isn’t it like the DOS Command prompt? The right and left arrows will help you to position the cursor in order to edit the command if need be.

Just stay right there and try some simple commands.

[aka@localhost ~]$ date
Thus Aug 15 12:18:01 UTC 2019
[aka@localhost ~]$

If you type down ‘cal’ in place of ‘date,’ a full month’s calendar will be displayed on the screen.

Basic Commands

If you are looking forward to operating Linux just like Windows, you are wrong. Linux is smarter. You get to work with a command set. It is just like coding. Enter the command to get the job done, and that's it. Let's see which of these you can operate. The fun is going to start from here. Get ready!

The marvel of the echo command is as under.

```
[root@localhost ~]# echo My love
My love
```

The # sign indicates that I am logged in as a *root* user. I think we are done with that. Now enter the following:
Let's try the cat comman-d and see what it can do.

```
[aka@localhost ~]$ cat/etc/passwd
```

A huge number of rows and columns will appear below the command. This command is meant to display the contents of the above file.

Do you want to move files from one place to another? Check out the *mv* command.

```
[aka@localhost ~]$ mv downloads documents
[aka@localhost ~]$ mv documents downloads programs lib
```

With the help of the second command, you can move multiple files at the same time into a particular directory. In the above command, *lib* denotes the name of the directory.
The cp command is used to copy files in Linux from one place to another like from downloads to documents.

```
[aka@localhost ~]$ cp downloads documents
[aka@localhost ~]$ cp file1 file2 file3 file4 fileN dir
```

The second command is to copy content from multiple files into a single directory.
With the help of the *touch* command, you can create a file. If the file already exists on your computer, its time stamp will be modified.

[aka@localhost ~]$ touch video
[aka@localhost ~]$ ls -1 video

As you enter the second command, it will show the modified time of the respective file.
In the end, *rm* command is used to delete files from the system. Is the recycle bin going through your mind? Forget it. You don't have to search the file manually to dispatch it to the recycle bin. Just type *rm* and the file name. It will be removed.

[aka@localhost ~]$ rm downloads

The text 'downloads' is the name of the file.
Just like the *touch* command, you can also create a new directory by typing *mkdir* and the new directory's name **.**

[aka@localhost ~]$ mkdir lib
[aka@localhost ~]$ cd /lib
[aka@localhost lib]$

Congratulations! You have succeeded in creating a new file. Here *mk* denotes make.
rmdir command allows you to remove a directory. Here *rm* denotes remove.

[aka@localhost ~]$ rmdir lib

The file will be removed from the system right away. You must keep in mind that the *lib* directory should not be empty at the time of removal, otherwise, the command will fail. This is perfect for you if you are one of those people who are sick of deleting individual files after searching out hundreds of folders and files. It will delete entire directories.
'What if I want to delete the sub directories that exist in the main directory?' asked Sylvia, who had by now learned enough to be able to craft a question after hours of brainstorming. For that purpose, you can add -rf to the command was my reply.

[aka@localhost ~]$ rm –rf lib

Although this command is a really helpful one for your businesses or personal work, this can also be a very lethal one. It can delete humongous amounts of data in the blink of an eye. Make sure you are deleting the right directory before executing the command. Once you enter it, it is all gone.

Let's take a look the options we have with the *rm* command.

rm documents : it helps you delete a particular file named 'documents.'
Rm -i documents : type this in the window and you will be asked for confirmation before deleting the file
Rm -r documents lib : it will delete the *documents* file and *lib* directory along with the contents.
rm -rf documents lib : this also deletes the *documents* file and *lib* directory even if one of them doesn't exist. The point is that it doesn't fail considering the fact that what you wrote as a file name didn't exist in the system.

The *In* command

This command creates new links. You can use it in the following forms:

[aka@localhost ~]$ In file *link*
[aka@localhost ~]$ 1s –s *item link* (for hard link)

Managing the System's Memory

Memory in Linux is divided into blocks, technically known as pages. When you enter a command to locate a particular page or general information on the memory of the system, the kernel starts working to gather the blocks into columns with each section designation a particular kind of memory like total memory, used memory and free memory. There is a process dubbed as swapping out, executed by the kernel in which it accesses the memory pages that remain out of access and brings them down to the swap space.

Now the question on your mind may be whether the kernel does that when you are out of memory. Well, no, it doesn't. It does that randomly. So, the swap space remains filled all the time. If you run a program that needs a block which the kernel has swapped out, it starts creating space for it. Yes, you understood this. The kernel swaps out some other page to bring in the required page.

If you want to check the memory on your Linux system, there are specific commands to do that. Some of them are as shown below, both for checking the swap as well as the RAM memory:

The free command: Let's start right away. You will see the following table:

```
[aka@localhost ~]$ free
        total  used  free          buffers       cached
Mem:        7976          5000          2976          749           1918
-/+ buffers/cache:        xxx           yyy
Swap        xxx           yyy           zzz
```

The table is self-explanatory. Everything is explained with the help of columns. You get to know about the total, used and free memory. It also shows memory consumption by buffers and cache, and also the status of the swap memory.

The df command: You can easily see the free space on your hard drive by the following simple command.

```
[root@localhost ~]# df
Filesystem   1k-blocks   used     Available Use% Mounted on
```

```
/dev/root       1048576    207640   840936       20%/
devtmpfs        125948       0        125948      0% /dev
tmpfs           125988       8        125980      0% /run
[root@localhost ~]#
```

The proc/ meminfo file: Another popular method to check the memory status is reading the proc/meminfo file.

Type in the command: '$ cat/proc/meminfo' and you will see a long list of options along with details of the memory consumed by each of them. Let's see. The following columns will pop up on your screens when you put in the commands.

```
[root@localhost ~]# $ cat/proc/meminfo
MemTotal:          xxx kb
MemFree:          xxx kb
Buffers:           xxx kb
Cached:            xxx kb
SwapCached:              xxx kb
Active:            xxx kb
Inactive:          xxx kb
Active(anon):             xxx kb
Inactive (anon):          xxx kb
Active(file):       xxx kb
Inactive(file):           xxx kb
Unevictable:              xxx kb
Mlocked:           xxx kb
SwapTotal:         xxx kb
SwapFree:         xxx kb
Dirty:        xxx kb
Writeback:         xxx kb
AnonPages:         xxx kb
Mapped:            xxx kb
Shmem:             xxx kb
Slab:         xxx kb
SReclaimable:             xxx kb
```

SUnrelcaim: xxx kb
PageTables: xxx kb
NFS_Unstable: xxx kb
Bounce: xxx kb
Commitlimit: xxx kb
Committed_AS: xxx kb
VmallocTotal: xxx kb
VmallocUsed: xxx kb
VmallocChunk: xxx kb
HubPages_Total: xxx kb
HubPages_Free: xxx kb
HubPages_Rsvd: xxx kb
HubPages_Surp: xxx kb
Hubpagesize: xxx kb

#
This memory detail shows the total memory line of physical memory that the Linux server has. How much is used and where it is used will be shown. You can see all the memory pages which can differ from system to system. Your Linux system might not have created some of the above-mentioned pages.

An Overview of Navigation in the Shell

You will easily get bored with Linux if you get stuck in a single spot. You must be able to navigate through different files and directories to enjoy your experience with this one of a kind operating system. You need to get an idea of where you are and what things you have access to and with which command you can access them. First of all, you should know where your feet are. To learn this, use the following command.

[aka@localhost ~]$ pwd

In most cases, people land in the home directory of their accounts. Now, what is the home directory? Let me explain it all. A home directory is a place where you can store your files in addition to building up directories. You are

the boss here. It is like a real home for you. Feel free to store your data here and also create multiple directories. In the command ‘pwd’ p means print, w means working, and d is for directory.

Is it a tree?

It doesn’t look like one, but it is one, for sure. As in the Windows operating system Linux arranges all its files and folders into a hierarchical structure. Yes, it looks like a tree if you could somehow materialize it into something concrete. Hence the need for the word ‘tree.’ You may be fascinated to know that things are mostly the same among different operating systems such as Windows and Linux. The creators changed some names to differentiate these systems for the ease of users. For example, what is called a directory in Linux, is commonly known as a folder in Windows.
Let me take you further into the world of Linux directories. First of all, these directories become a bit different from Windows’ folders in a way that they can carry a large number of files as well as subdirectories. The very first directory you are confronted with is the ‘root’ directory.

Where Linux doesn’t agree with Windows

Well, disagreements happen in the world. If you are a Windows user, you are most used to an individual filing system for the hard drive, USB storage device and DVD drives. But Linux has defied this system. It offers a single treelike filing system for all the external and internal storage devices.

Windows made things easier for an average user that detached us from the relish of complex technical commands. Look, Windows presents the filing system, of course, in the form of a tree, in a graphical fashion. There are many plus signs and negative signs. With the pluses, you can further explore the branches of the tree while the minuses help you close up those branches and simplify the view. The root always remains at the top.
If you are thinking about something as cool as a graphical outlook, stop thinking right now. Linux makes it fun by offering you the potential to navigate through different files and folders with the help of putting in text

commands. In the Linux terminal or shell window, you are always in one or another directory that is called the current directory. To know your exact position ‘pwd’ command helps you as I have explained earlier.

Do you want to know about the contents of the directory? Everybody does. I believe that you have already learned, as well as practiced knowing the directory you are currently in. Time to delve deeper into the system. You should know how to explore the contents of the directory with a single command. Either you can stay in the home directory and explore it, or you can move to your favorite one. To explore the directory you are currently working in, try the following command.

```
[aka@localhost ~]$ ls
Pictures Videos Games Documents
```

My friend Sylvia loves to jump from one folder to another at the speed of light. While it is fun to do this in Windows, it is faster and easier to do in Linux. Just remember the ‘cd’ factor. Sylvia learned it, though the hard way, by jotting it down in her notebook and memorizing the entire text. Now she just types ‘cd’ and the pathname of the directory she wants to jump into.
Let me confess it is not as simple as it looks. The pathname is like a magic circle in which we jump to reach the directory we want to be in. Pathnames fall into two categories: relative and absolute. Shall we deal with the ‘absolute’ first?

Absolute pathnames: Also known as absolute path or a full path. A pathname is usually made up with the sequence of characters, containing the name of the object. The name of the directory in which it rests is also part of the name. There can be a directory that contains software and programs. In this directory, there are other directories to which you need access. Your path will be like the following.

```
[aka@localhost ~]$ cd/ usr/lib
[aka@localhost lib]$
```

You can at any time confirm if you have actually moved to the desired

directory by applying the ‘pwd’ command.

Relative Pathnames: You can start from the current directory and move to the parent directory with a simple command. Do you love to put in dots like (….). A single dot refers to the current working directory whole the double (..) leads you to the parent directory.

[aka@localhost ~]$ cd/ usr/lib
[aka@localhost lib]$

I want to go to the /usr. You can do that either the long way or the shorter way. Let’s do this first the longer but safer way.

[aka@localhost lib]$ cd/ usr
[aka@localhost usr]$ pwd
/usr

Now do this the shorter and the faster way.
[aka@localhost lib]$ cd..
[aka@localhost usr]$ pwd
/usr

Done. You have switched it to the parent directory. Wait. We are not all done on this. Let’s get back to the working directory once again by two methods. We are currently in the parent directory. Remember that. Your screen is currently showing the following.

[aka@localhost usr]$

Type the following.

[aka@localhost usr]$ cd/ usr/lib
[aka@localhost lib]$

Let’s do this in a shorter and faster way. Do you remember what single dot

was supposed to do?

[aka@localhost usr]$ cd./bin
[aka@localhost lib]$

Sylvia just hated the dots. She always put doubles where she had to insert the single and single where a double was needed. One day, when I was taking a quick walk by her office, I dropped in and found her messed up with these simple commands. A friend in need is a friend indeed. I told her a magic formula for switching from parent directory to the working directory. Let's try that.

[aka@localhost usr]$ cd bin
[aka@localhost bin]$

Not only are the dots omitted but also the slashes.
A trick of the trade: cd is an abbreviation of current and directory. If you try out just the cd command, it will bounce you back to the directory in which you first started after you logged in.

[aka@localhost lib]$ cd
[aka@localhost ~]$

More on the 'cd' command: Let's try out the (-). It is supposed to change the working directory to the one you left behind. Suppose you are currently in lib after switching from the 'usr'.

[aka@localhost lib]$ cd –
/usr
[aka@localhost usr]$

Another important command to know is cd ~ username. If you have another user name like john, you can put in the name in the place of username to switch to that account's directory.

Food for thought: It is important to keep in mind that in Linux you need to take care that you are using the right case when you are entering a file name in the shell window. In this operating system downloads and Downloads are two different things.
If at any time you want to go back up where you started, just type *cd* and press enter. Then enter *pwd* to know where you are at the moment.

What you can already do with the ls Command

It is now time to learn more commands as you already know the basics. You now know that ls is used to list directories. Also, you can view the content of the directories.

```
[aka@localhost ~]$ ls
bin  games  include  lib  lib64  libexec  local  sbin  share  src  tmp
```

In the above command, you can see four different directories. Now I want you to take a dive into your brain and bring out a specific directory name you want to list. Let me dive in first using mine. I'll use # for this command - that is the face of the super user.

```
[aka@localhost ~]$ ls /usr
```

Let us take a look at some more options for the 1s command.
There is an option (- a) which can be extended to --all. It will list all files including the ones starting from a period that usually remain hidden from other commands.

```
[aka@localhost ~]$ ls –a
. .. bin  games  include  lib  lib64  libexec  local  sbin  share  src  tm
```

The second option is -d that can be extended to --directory. If used alone, this command will show you a list of your directory's contents. You can pair it up with the -1 in order to view the content details.

The third option to consider is the -1 command. You can see the outcomes in the long form.

```
[aka@localhost ~]$ ls –1
bin
games
include
lib
lib64
libexec
local
sbin
share
src
tmp
```

The fourth option is the -S. It sorts out the results with respect to the size of the file. Here S denotes the word size. If you remember the signs by assigning them a full, relevant and familiar word, it will help you memorize the commands faster than normal.

The fifth option is -h with the long form --human-readable. This helps you to get a full-size display of directories and files that is human-readable instead of just showing bytes.

The sixth option is -r with the long form --reverse. It displays your results in a reverse angle that means in the descending alphabetical order.

```
[aka@localhost ~]$ ls –r
tmp  src  share  sbin  local  libexec  lib64  lib  include  games  bin
```

The seventh option is -t. As apparent from the use of the letter ‘t’ this option offers to view the results with respect to modification of time.

```
[aka@localhost ~]$ ls –t
sbin  lib  lib64  libexec  share  include  src  local  games  tmp  bin
```

You can see that the position of the files has been changed with respect to the modification of time.

How to make out a File’s Type

Now that we are going deeper into the system, it is time to know how to determine the type of a particular file. Actually, Linux does not make it that much easy for the users as Windows does. There are not familiar and most likely type file names that you could easily guess. It makes it rough and tough. To determine a file type, you should type in the following command.

```
[aka@localhost ~]$ file downloads
```

This is known as the file command. When you execute it, it is likely to print the following information.

```
[aka@localhost ~]$ file downloads
images.jpg: JFIF standard 1.01
```

In Linux there are lots of files. As you explore the system, you get to know more and more types of the files. So, stay tuned! Keep reading.

The *Less* Command

There is another important command in the arsenal of Linux that is the *less* command. It is a full program for those who are interested in viewing text files. This command helps you view the human-readable files in your system. It tracks them out in no time and allows you to view them.

Wait! As I write, Sylvia knocks the door and jumps in the lounge where I was working on my laptop. She is always curious about learning the Linux command line since the day her boss made it mandatory for her to do that. I must admit that what was earlier on a big burden on her nerves and shoulders had now turned into a passion. She loves the way Linux made it easier to wrap up office work in lesser time. The time she saves at the office allows her to drop in at her friends' homes. I hope you have stopped thinking about why she is here in my home. Let's welcome her.

Sylvia, when learning the *less* command, was always curious why do we need this command in the first place? In fact, why do we need to see the text files? Its answer is that many files are not just ordinary files. They are system files that contain *configuration files*. They are stored in your system in this special format.

If you are thinking like Sylvia, you might say that's not a big deal. Why do I have to view the configuration files in the first place? There is an answer to that as well. This command not only shows the system file but also lots of real programs that your system is using. These programs are technically called *scripts.* They are always stored in the system in this specific format.

I am using the word *text* for quite some time. Information can be displayed on a computer in a variety of forms. It is represented in the form of pictures, videos, numbers and alphabets in Windows operating system. Normally the language that the computer understands is in numerical form like the binary digits.

If we look at text as the representation system, it feels simpler and easier than

all the other methods. The human brain processes the information faster than in this form. Numbers are converted into text one-by-one. This system is very compact. This text is different from that of the one in word processors. I am telling you about this because in Linux a number of files are stored in the system in the form of text files. Let's see how can we use the *less* command.

[aka@localhost ~]$ less downloads
This program helps users to navigate through a specific file. Some files are longer and some are shorter. For longer files, such as the one containing system information, it can be scrolled up and down. If you want to exit the less command just strike the Q key. Some files can be really longer like the following one.
[aka@localhost ~]$ less /etc/passwd

Things to Remember

You can use the Page UP key to scroll back a page. Use 'b' instead if you don't like to use Page UP.
Use the PageDn key to scroll on to the next page. For the same purpose, you can also use spacebar.

The up arrow and down arrow keys are used for moving to one line above and below respectively.
If you are in the middle of the text and want to jump to the end of the text file, press G. To reach the start of the text file, press 1G or g.
'h' can be used to get the help screen displayed.
'q' is always there to quit the *less* command.

Common Directory Names in Linux

‘/’ denotes the root. There are usually no files in this directory.
‘/lib/ denotes the library directory to store library files.
/root takes you to the home directory. A root user is also called the super user.
/etc contains configuration files
/bin stores GNU user-level utilities. Also known as the binary directory.
/opt is executed to store optional software.
/dev is known as the device directory.
/usr is the user installed software directory
/var is the variable directory. It changes frequently.

Chapter 2: Exploring the Realm of Commands

I hope that by now, you have gained sufficient knowledge about how you can navigate through the shell window using simple commands. That's easy, you see. A little bit of extra consideration will help you get through most of the work in no time which in the Windows operating system might have taken hours of exhaustive work. One more interesting thing is that the commands are easy to memorize. You can also create your own commands and that's what we will learn in this chapter. Also, by getting acquainted with more commands, you will feel more comfortable in using them in the real shell window.

There are different types of commands:

It can be a shell function, a kind of shell scripts that are incorporated in the environment. Commands constitute executable programs just like software engineers do while working on C language or Python and Ruby. Or a command can be an alias which is your self-made command.

Do you know how to know a command's type? Let' see how it is done.

```
[aka@localhost ~]$ type cd
cd is a shell buildin
[aka@localhost ~]$ type cp
cp is /bin/cp
[aka@localhost ~]$ type rm
rm is /bin/rm
[aka@localhost ~]$ type mkdir
mkdir is /bin/mkdir
```

Now we know the type of each of our commands.

The *Which* Command

You can use this command to track the executable file. On Linux systems, unlike the Windows operating system, more than one version of programs are installed. This command tells us where to find the program.
Let's look at the syntax:
-/bin/cd
[aka@localhost ~]$ which rm
/bin/rm

It doesn't work on built in programs. See the following syntax.
[aka@localhost ~]$ which ls
/usr/bin/which: no 1s in (/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin)

The info which command: this command displays the help information.

[aka@localhost ~]$ info which
Next: Which Program, Prev (dir), Up: (dir)
'which': Show the full path of commands
****
The 'which' program shows the full path of (shell) commands.

This file documents 'which' version 2.21, updated 20 March 2015.

*Menu:

*Which Program:: The 'which' Program
*Invoking Which:: How to invoke 'which'
*Option Summary:: Overview of commandline options
*Return Value:: The return value of 'which'
*Example:: Examples of invocation
*Bugs:: Known bugs
*See Also:: Related UNIX commands
*Index:: Index

-----Info: (which)Top, 20 lines --All----------
Welcome to Infor version 6.5. Type H for help, h for tutorial.

If you type H, the following details show up on your screen.

Basic Info command keys

H Close this help window
q Quit Info altogether
h Invoke the Info tutorial

Up Move up the Info tutorial
Down Move down one line
PgUp Scroll backward one screenful.
PgDn Scroll forward one screenful.
Home Go to the beginning of this node.
End Go to the end of this node.

-----Info: *Info Help*, 302 lines --Top----------

Now type *h* and the following details will appear on your screens.

Next: Help-P, Prev: Help-Small-Screen, Up: Getting Started

The *Help* Command

You have learned lots of commands and their solutions. It is time to know the command that can get you out of the mess if you are stuck in the middle of the course and don't know what to do. Sylvia loved this feature while she was in the learning phase. One day I was planning to celebrate my daughter's birthday at a beach-side resort, Sylvia phoned me. She was panicked as she had messed up some command at her office computer. She tried the *help* command but to no avail. When the *help* command failed to do its job, she panicked. Actually, the *help* command is only created for the shell built-ins.

[aka@localhost ~]$ help cd

cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

An interesting thing about the *help* command is to get to know what the command has to offer inside the shell window. Let's see the syntax.

[aka@localhost ~]$ help help
help: help [-dms] [pattern ...]
    Display information about builtin commands.

The --*help* Option

This command will help you explore more information about executable programs. This one is really amazing because it shows different options pertaining to the program and also the supported syntax. It becomes easier to learn and exercise more commands on the go. You don't need to consult a book for small hurdles.

[aka@localhost ~]$ cp --help
The help feature can also be used along with different options to get more information along the way.
Using it with *–d*:

[aka@localhost ~]$ help –d help
help – Displays information about built in commands

Using it with *–s*:
[aka@localhost ~]$ help –s help
help: help [-dms] [pattern …]

The *man* command: In most cases, an executable program offers a formal piece of documentation that is dubbed as a *man page*. You can view the document with the help of the following command.

[aka@localhost ~]$ man program

Here program refers to the command that you want to view. The pages of

man are displayed as such that there is a title. There also is a description that explains why should you use a particular command? Beside that there will be lists along with the description of the command's options. Let's dissect a *man printf* command.

[aka@localhost ~]$ man printf

There will be a name of the document. Below which there will be the heading of SYNOPSIS. Then comes the description. This section carries in-depth details about the options available for the particular program. For the *printf*, *--help* and *–version* are available.

That was about the full page of the manual. It takes a lot of time to skim through the entire pages to locate a particular piece of information. Don't worry. You can type a targeted command to read a particular section of the manual. The *man* command offers you to specify the section number. Let's see how to do it.

[aka@localhost ~]$ man section search_term
[aka@localhost ~]$ man 9 passwd

When you execute the command, you will reach the specific section of the man page.

The *whatis* Command

If you don't know what a specific command has to offer, you can run the *whatis* command to have a brief detail about it. In most cases, the detail consists of a one liner but self-explanatory so that you learn about the function of the command faster.

[aka@localhost ~]$ whatis Is
Is (1) - list directory contents

The *info* Command

For the users of GNU Project, there is an alternative to man dubbed as the *info pages*. These pages come with a reader program named as *info*. The syntax runs as follows:
[aka@localhost ~]$ info option menu-item

An example: the syntax is as under.

[aka@localhost ~]$ info –d cvs

As I have already told you that Linux stores files in a tree like hierarchical structure. The *info files* in a Linux also are in the form of a tree. They are built into individual nodes. Each one of them contains a single topic. There also are hyperlinks for easy navigation to another node. Well, they are not dyed in blue, just as in Windows. They can be identified by their leading asterisk. Move over the cursor and strike the Enter key to open it.

The *info* command is very interesting to use. You can use it from multiple angles. If you want information on the physical location of the file, the syntax is given as under:

[aka@localhost ~]$ info –w cvs
/usr/share/info/cvs/info.gz

In case you need information on the options pertaining to a particular command, you can do this by making a simple change in the syntax.

[aka@localhost ~]$ info -O mv
Next: rm invocation, Prev: install invocation, Up: Basic operations
It displays the options node for the *mv* command. You can learn a lot with the help of *info* command. When Sylvia was learning Linux command line, she used to keep a notebook with her to jot down important things they came across while running the shell window. This helped her a lot. She picked up important pieces of information she knew.

Let’s look at the options the *info* command has to offer.

- -O takes us to the options available for a particular command line.

- -d, d for directory, helps us add directory to the INFOPATH
- -k finds out STRING in all the indices available in all the manuals.
- -o is about offering selected nodes to the file.
- -version offers users information about the version.

The *apropos* Command

For a normal human being, using Linux is not a child's play. It is not that it is something ethereal which the earthmen don't understand, but it is its commands that make it nothing less than a horrible thought. It is almost a nightmare for me to even think that I had to memorize each command to be a Linux pro. If you are just like me and cannot remember a ton of commands that Linux has to offer, perhaps you need a comprehensive lecture on this very command. It just saves me from lots of hassle of remembering so many commands.

The skilled hands behind the development of Linux also understand this. That's why they have created the *apropos* command. Just a keyword of a command is enough for *apropos* to transport it in front of your eyes. This command helps users to be more efficient and effective when they are at the shell window. Shall we see the syntax now?

[aka@localhost ~]$ apropos keyword
[aka@localhost ~]$ apropos option keyword

You can use one keyword at a time.

[aka@localhost ~]$ apropos email

It will display all the necessary information about the email. To add more spice, you can use the option -d which will trigger the terminal to return path& man directories etc. pertaining to the keyword you enter.

[aka@localhost ~]$ apropos –d email

The list goes on. I have just written a little less than half of the details that the

window returned in response to the apropos command.

Are you a Google user? If you are, you might know that google cannot see you in pain of thinking the exact keywords that you need to search. You enter a word and a list drops down containing most relevant search phrases. Most often, I go with one of the phrases from the list. Just imagine if you could avail yourself of a similar facility in Linux. Thankfully, you can. Use the -e option. Let's jump to the syntax right away.

[aka@localhost ~]$ apropos –e compress

You will have a list of commands starting from the word 'compress.'

A Trick of the Trade

Is it possible to combine different commands and get results? Absolutely, yes. You can do that. I'll write down the syntax.

[aka@localhost ~]$ cd /usr; ls; cd -
bin games include lib lib64 libexec local sbin share src tmp
/usr

We have successfully executed three consecutive commands in a go; changing the directory to /usr, listing the directory, and then returning to the directory in which we were in before the execution of the command. All three in a command.

Shall we try creating our own commands?

We can do that with the help of the command we have just tried and tested. Think about a possible name and confirm if it has not been taken.

[aka@localhost ~]$ type xin
Sh: type; xin; not found

So, xin is not there. Here is the syntax to create the alias.

[aka@localhost ~]$ alias xin=’cd/usr; ls; cd –‘

Just remember the string and finding out a name that has not already been taken. Execute the command. Great! It seems you have already created your own command.
Let’s try the new command.

[aka@localhost ~]$ xin
bin games include lib lib64 libexec local sbin share src tmp
/usr

Congratulations! To confirm run the *type* command to see to which your command *xin* has been aliased to.

[aka@localhost ~]$ type xin
xin is aliased to `cd/usr; ls; cd -'

If you desire to take a look at all the aliases in the environment, follow the syntax as under.

[aka@localhost ~]$ alias
egrep='egrep --color=auto'
fgrep='fgrep --color=auto'
grep='grep --color=auto'
l.='ls -d .* --color=auto'
ll='ls -l --color=auto'
ls='ls --color=auto'
mc='. /usr/libexec/mc/mc-wrapper.sh'
which='(alias; declare -f) | /usr/bin/which --tty-only --read-alias --read-funct
ions --show-tilde --show-dot'
xin='cd/usr; ls; cd -'
xzegrep='xzegrep --color=auto'
xzfgrep='xzfgrep --color=auto'
xzgrep='xzgrep --color=auto'
zegrep='zegrep --color=auto'

zfgrep='zfgrep --color=auto'
zgrep='zgrep --color=auto'

Your screen can show a slightly different result as per the environment you are using. Just be sure you have the information about aliases that are present in your system.

The I/O Saga

We are going to learn about the input and output in the shell window. You will be using meta characters. Basically, this is responsible for rerouting the I/O between the files. In addition, it is used to create command pipelines.

Getting acquainted with the Standard Stream: Almost all computer-based programs deliver output in reaction to the input you place. For example, if you are browsing through a website and you enter the back button, a window pops up forbidding you from getting back, or you will lose your data. The same kind of window pops up when you try to close a word file that you have not yet saved. You enter keys on the keyboard just like you are going to do right now; its results show on the desktop in a word file or in the Google search bar or in the terminal emulator of Linux. That's how the I/O system runs in general. You do have some error messages placing themselves like a frowning genie at the door of a cave.

Dissecting the I/O System

Standard input: It is given the '0' number. Your keyboard is used for the input. In short form, it will look like this (stdin).
Standard output: It has the (1) number. Your guess is right. Of course, the output is displayed on the screen before your eyes. No doubt about that, you see. In short, it is displayed as (stdout.)
Standard error: Give it the number (2). If you miss something in the input or don't exactly know the command, the bash shell will throw off an error box. It is displayed as (stderr) in short form.
There is nothing complex about the redirection process regarding the relationship between a keyboard and the Windows operating system. You will understand what it is all about.

I want to Redirect toward a File

Redirection is a little bit technical than the other commands, and also a bit different. You will be using > and < brackets. Sometimes you will have to use >> or << brackets for the desired

output. But don't worry, we will learn to use this.
Let's roll on. You will have to use the > symbol along with the desired filename. You can dispatch the output of the command to the file you want. Let's learn it in a simple way. You are working in the Microsoft Word. You type in your name but don't want it to be displayed on the screen and instead desire it to be moved to a particular file.

[aka@localhost ~]$ echo Hi there. > test.txt
[aka@localhost ~]$ cat test.txt
Hi everyone.
[aka@localhost ~]$

The file returns back to you only when you demand.

I want to Erase the Output

Yes, you can. Commands in Linux definitely are like magic wands. With a single and simple command, you can do overwrite the command. Let's do that.

[aka@localhost ~]$ cat test.txt
Hi everyone.
[aka@localhost ~]$ leo > test.txt
Sh: leo: command not found
[aka@localhost ~]$ cat test.txt
[aka@localhost ~]$

You can see that the file is not found still it was overwritten and erased. When I run the cat command after that, there was no return.

I don't want the file to be erased like that: Do you mean it? Of course, you mean it. There is a way to do that. We have the noclobber option to save deletion due to overwriting of files. Let's learn the syntax .

```
[aka@localhost ~]$ echo I love blue sky. > mydocuments.txt
[aka@localhost ~]$ cat mydocuments.txt
I love blue sky.
[aka@localhost ~]$ set –o noclobber
[aka@localhost ~]$ echo the sky is azure. > mydocuments.txt
sh: mydocuments.txt: cannot overwrite existing file
[aka@localhost ~]$ set +o noclobber
[aka@localhost ~]$ echo the sky is azure. > mydocuments.txt
[aka@localhost ~]$ cat mydocuments.txt
the sky is azure.
```

The first command kept the output from getting overwritten, but the second one reverted it back to the prior position.
The noclobber can be overruled:

Suppose we have the noclobber in place that is preventing the overwriting process. Try this one out.

```
[aka@localhost ~]$ echo the sky is azure. > mydocuments.txt
sh: mydocuments.txt: cannot overwrite existing file
[aka@localhost ~]$ echo the sky is azure. >| mydocuments.txt
[aka@localhost ~]$ cat mydocuments.txt
the sky is azure.
```

The sign >| does the magic. Clearly, the noclobber has been overruled.
If you want to redirect two file content without overwriting the first, there is a command for this. As I said, we have magic signs for the purpose. So, let’s do that.

```
[aka@localhost ~]$ cat mydocuments.txt
I love blue sky.
[aka@localhost ~]$ echo the sky is azure. >> mydocuments.txt
[aka@localhost ~]$ cat mydocuments.txt
I love blue sky
the sky is azure
```

[aka@localhost ~]$

This process is known as *appending*. It makes sense as two file contents are combined in the process.

Redirecting the Error Message

Let’s practice this first in the terminal emulator.
[aka@localhost ~]$ echo hi everyone
hi everyone
[aka@localhost ~]$ echo hi everyone 2> /dev/null
hi everyone
[aka@localhost ~]$ zcho hi everyone 2> /dev/null
[aka@localhost ~]$
[aka@localhost ~]$ zcho hi everyone
sh: zcho: command not found
[aka@localhost ~]$
So, you can see how this command can make our display less messy. There won’t be lengthy explanations if you type it wrong. Make sure to keep your display clean in the future.

Moving on to the Pipelines

Your commands are redirected through a network of pipelines. The standard inputs and outputs are received and displayed, respectively, with the help of the pipelines.

| is the operator. It is like a wall that helps pass one output to some other virtual place. It is just like you are transferring some physical commodity. Executing the pipeline command will help you push one output to another command. No activity on the display screen is required neither is required the dull exchange through temp files. A pipe does all the process in the background in complete silence. One important thing to remember is that you can only send the data. You cannot receive it from the same pipe. The information flows in just one direction. I am

using the less command because it displays the standard output of almost all commands. Test it and see the results. The results may differ from user to user.

```
[aka@localhost ~]$ ls -1 /usr/bin | less
[
ack
addr2line
ag
alias
animalsay
animate
annocheck
appliance-creator
applydeltarpm
appstreamcli
apropos
ar
arch
aria2c
arpaname
as
aserver
aulast
aulastlog
ausyscall
authvar
auvirt
awesome
awesome-client
awk
axel
b2sum
```

Redirecting Standard Input

The cat command—Concatenate files
This command is special in that it tends to read one and more than one files. It also moves files and copies them to standard output. Let's see how it runs.

[aka@localhost ~]$ cat 1s-output.txt

It is important to mention here that some users consider cat being similar to the TYPE command. But the reality is different. Files can be displayed with the help of the cat command without being paged first. The above command displays the contents of ls-output.txt file. The output will come in the form of short text files and not in the form of pages. It speeds up the process. Not only does paging take considerable time, but it is also more complex and time consuming to display it in the form of pages on the screen. You can add multiple files to the cat command as arguments. This command can also be used to append multiple files. If we have multiple files named snazzy.jpeg.01 snazzy.jpeg.02 …… snazzy.jpeg.99
Now you want to join all these files. Let's try the following command.

[aka@localhost ~]$ cat snazzy.jpeg.0* > snazzy.jpeg

We will have the files in the right order. The cat command welcomes standard input that is linked to our typing something on the keyboard.

[aka@localhost ~]$ cat > azure_sky.txt
The sky is turning azure.

You have just created a file by using the cat command. You have also put some value in the file. When you are done with the above two steps, press ctrl + d to reach the end of the file. You can bring back the content of the file with the help of the same command.

[aka@localhost ~]$ cat azure_sky.txt
the sky is turning azure.
[aka@localhost ~]$

Pipelines: Commands can surf through the standard input to read data and then send it to the standard output. This is done with the help of a special shell feature dubbed as pipelines.

```
[aka@localhost ~]$ ls -1 /usr/bin | less
[
ack
addr2line
ag
alias
animalsay
animate
annocheck
appliance-creator
applydeltarpm
appstreamcli
apropos
ar
arch
aria2c
arpaname
as
aserver
aulast
aulastlog
ausyscall
authvar
auvirt
awesome
awesome-client
awk
axel
b2sum
b43-fwcutter
```

This command can be used for a page by page display of what a command sends to the standard output.

***Filters*:** You can utilize pipelines to perform other operations connected to data on your system. You have the freedom to join together multiple commands in a single pipeline which will then be known as *filters*. Let's try one filter. For this one, we are appending together /bin and /usr/bin to see what they have got for us.

```
[aka@localhost ~]# ls /bin /usr/bin | sort | less

[
[
ack
ack
addr2line
addr2line
ag
ag
alias
alias
animalsay
animalsay
animate
animate
annocheck
annocheck
appliance-creator
appliance-creator
applydeltarpm
applydeltarpm
appstreamcli
appstreamcli
apropos
apropos
ar
ar
```

arch
arch
aria2c
aria2c
arpaname
arpaname
as
as
aserver
aserver
aulast
aulast
aulastlog
aulastlog
ausyscall
ausyscall
authvar
:
[aka@localhost ~]$
This is how filters can help you produce a single sorted listed. It is the magic of the *sort* command that we put into the pipeline. Otherwise, there would have been two lists for each directory. It saves times and is more efficient to view multiple directories.

The grep

This also is one of the powerful programs that are used to find text patterns in the files. Let's look at how it is used.

[aka@localhost ~]$ grep pattern [file……]
This command is used to hunt down matching pattern lines in the files. Let's try out the command.
[aka@localhost ~]$ ls /bin /usr/bin | sort | uniq | grep zip

The head/tail

This command is used to retrieve a specific part of the information. As

apparent from the command name, you can print the head or the tail of a particular file. It means you can get the first few lines of a file printed or the last few lines of the same file. Ten lines from the start or ten lines from the last. That's how it goes. The number of lines can be altered by applying the -n option. Let's scroll to the syntax of the command.

```
[aka@localhost ~]$ head -n 8 ls-filename.txt
```

This command can be used in the pipelines as well.

```
[aka@localhost ~]$ ls /usr/bin | tail -n 10
```

With the help of the tail option, you can view the files in real time, and review the progress of the log files while they are in the midst of being written. The following command helps you in reviewing the messages file.

```
[aka@localhost ~]$ tail -f /var/log/messages
```

The option *-f* is used to monitor real time messages. The command will only show the tail while the messages are being written. They keep popping up on your screen as long as you keep the screen open. To stop it you need to press ctrl + C.

Exploring the tee Command

Linux also has the tee command. Well, you might be thinking of golf at the moment. Just image you are actually in a golf club to tee off. Golf is an amazing sport, but Linux is still better than that when it comes to the thrill and magic of the moment. Another analogy is with plumbing. You have to fix a 't' on a pipe to guide it through your washrooms to the water tank.

This program has a special job of reading the standard input. It reads it and then copies it to the standard output. As a result, the data flows downwards. You can capture what is being passed in the pipeline. Let's see the syntax.

```
[aka@localhost ~]$ ls /usr/bin | tee ls.txt | grep zip
bunzip2
bzip2
bzip2recover
```

funzip
gunzip
gzip
unzip
unzipsfx
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
[aka@localhost ~]$

The *uniq* command

This command is used to sort a file. Both uniq and sort are applied together. To read more about uniq, please take a look at its man page. I have already briefed you about the man page. The following command will not sort the duplicates. You can see by running it on the terminal.

[aka@localhost ~]$ ls /bin /usr /bin | sort | uniq -d | less

[
ack
addr2line
ag
alias
animalsay
animate
annocheck
appliance-creator
applydeltarpm
appstreamcli
apropos

ar
arch
aria2c
arpaname
as
aserver
aulast
aulastlog
ausyscall
authvar
auvirt
awesome
awesome-client
awk
axel

To sort the files you have to remove the -d option from the syntax. Now try it on.
[root@localhost ~]$ ls /bin /usr /bin | sort | uniq | less

The *wc* Command

As visible from the abbreviation, this one is to count the words in a file content. It also counts how many lines there are in the file content in addition to bytes.

[root@localhost ~]$ wc mydocuments.txt
4598 64577 537456 mydocuments.txt

It will tell you the number of lines, number of words and total bytes that belong to the mydocuments.txt file. Unlike the Windows operating system, you don't have to search each file content and check out the weight of the file in bytes. The pipeline command will retrieve the information you need.

The Magic of Echo

We know that the echo command displays the text as it is written in the shell.

What you don’t know is that you can pair up echo with different commands and get marvelous results.

Expansion of some wildcard entries like the (*)

The shell window tends to expand simple commands like the *. While it appears nothing to us users, for the shell it can have lots of meanings. This process is called expansion. The shell interprets such symbols and expands them into words or phrases easy enough to understand. The echo alone is just a print out command. You enter a sentence and the echo prints it out without changing it a bit. Let’s see.

```
[aka@localhost ~]$ echo I kept looking at the azure sky for five minutes.
I kept looking at the azure sky for five minutes.
[aka@localhost ~]$
```

If we add * after the echo, the result will not be the same. The shell takes * as a wildcard which has a job to match characters in the names of files. It will expand the * into the names of the files that are kept in the directory in which you are working at the moment.

```
[aka@localhost ~]$ echo *
bench.py hello.c
[aka@localhost ~]$
```

The pathname expansion

Let’s try something more interesting with the echo command.

```
[aka@localhost ~]$ ls
Desktop ls-output.txt Pictures Templates
Documents Music Public Videos

[aka@localhost ~]$ echo D*
Desktop Documents
and
[aka@localhost ~]$ echo *s
Documents Pictures Templates Videos

[aka@localhost ~]$ echo [[:upper:]]*
Desktop Documents Music Pictures Public Templates Videos
```

Arithmetic expansion: The echo command can also be used to execute arithmetic functions, some quite complex. In this way, the shell prompt does the job of a calculator.

```
[aka@localhost ~]$ echo $((2*2))
4
[aka@localhost ~]$
```

This command can be used for addition, multiplication, subtraction, division, modulo and exponentiation. Either you can do simple mathematics or move on to conduct some complex mathematical operations by nesting different expression in a single script. Let’s see how it goes on.

```
[aka@localhost ~]$ echo $(($((5*75))+1000))
1375
```

Chapter 3: The Linux Environment

Now that you are well aware of the shell, its usage, and some of the basic commands to execute different tasks, I will move on to the use of kernel. You will get to know how to use the kernel like how it boots and how it starts. Let's take a look at the normal booting process of the kernel. I have divided it into some basic simplified steps.

1. The first step is the BIOS. In this phase, some integrity of the system is thoroughly analyzed. BIOS located the boot loader and executes it. The boot loader can be on a CD, a floppy or a hard drive. After BIOS loads it up on the system, the MBR boot loader has full control. Wondering what is MBR? Let's jump to the next step to understand what MBR is.
2. MBR is the short form of Master Boot Record. MBR can be traced back to the 1st sector of the hard drive or any other bootable disk. The sector in the hard drive is titled as /dev/had or /dev/sda. MBR is a very light program when it comes to size. Its major component is the primary boot loader which is just shy of 512 bytes. This information in the form of a few bytes is mainly about GRUB. Sylvia learned it in a chain like system like BIOS loads the MBR then MBR load the GRUB.
3. What is GRUB ? It is the short form of Grand Unified Bootloader. GRUB also is a loader for kernel images. If you there is one kernel image on your system, GRUB will take it as the default image and loads it right away. If there are multiple images, GRUB will bring out a splash screen that has multiple images. You have got to choose one for loading. If you don't go for one, GRUB will load the default image as per settings in the configuration file.
4. The fourth step is the kernel itself. Kernel runs the */sbin/init* program which has 1 as the process id. The term *init* can be expanded to *initrd* that can be further expanded into initial RAM Disk. Kernel uses it for a short window of time from being booted to mounting of root file system.
5. After *init* starts running, you reach the *user space start.* The i*nit* takes over from there by running the system.

The Startup

A wide range of diagnostic messages are produced at the booting time originating from the kernel in the start then from the processes after *init* takes over the operations of the system. The messages don't run in a sequence and also, they are not consistent. For a new user, they can be confusing and even misleading. Some of the Linux distribution mechanisms try to slash them with the help of splash screens and changes in boot options. You can see and read these messages by the following commands. Let's see. I have logged in as a superuser. Let's look at the result of the command.

```
[aka@localhost ~]$ dmesg
[    0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[                0.000000] Linux version 4.15.0-00049-ga3b1e7a-
dirty (bellard@localhost.loca
ldomain) (gcc version 7.3.0 (Buildroot 2016.08-git-
svn30683)) #11 Thu Nov 8 20:3
0:26 CET 2018
```

It makes use of the ring buffer of the kernel. The above is a short version of what you will see on the page. It may not be exactly the same as above, but it will be nearly the same. Let's explore the dmesg command with different options to get more information.

```
[aka@localhost ~]$ dmesg | less
```

You can use other options with the *dmesg* command. Let's take a look at different options that can be explored in Linux. I will not go into the details of what you will see on your screens. But I will definitely tell you what can you expect by entering each command. Let's see the syntax. This time I have switched the user.

```
[aka@localhost ~]$ dmesg –C          You can use this command to clear
the kernel ring buffer
[aka@localhost ~]$ dmesg –c          You can apply this command to clear all
the messages.
```

[aka@localhost ~]$ dmesg –k This is used to read kernel messages.
[aka@localhost ~]$ dmesg –L It colorizes the content of messages.
Other options are the following. Enter them in the terminal and see what they may bring to you.

[aka@localhost ~]$ dmesg –D
[aka@localhost ~]$ dmesg –r
[aka@localhost ~]$ dmesg –s
[aka@localhost ~]$ dmesg –n
[aka@localhost ~]$ dmesg –F
[aka@localhost ~]$ dmesg -e

What Boot Options Do You have When Kernel Initializes

Kernel initialization in Linux consists of some key steps. It starts with the examination of the Central Processing Unit (CPU). After that, memory is examined. Then comes the device is discovered, root filesystem goes operational and in the end user space starts.
The Parameters: When the kernel starts running, it receives its *parameters* from the boot loader. These parameters guide the kernel on how should it start. These parameters carry specifications on the diagnostic output, as well as options, on the drivers of the device. The parameters can be viewed by a dedicated command. Let’s look at the syntax.

[aka@localhost ~]$ cat /proc/cmdline

Not all parameters are important but one which is the root parameter. The kernel is unable to operate without it because it just cannot locate *init*. You might be thinking of adding something in the parameter that it doesn’t understand? Well, Sylvia added it for sure. She added the -s in the parameters. The kernel saved the unknown phrase and moved it to *init*. The kernel interprets it as the command to start in the single user mode.

The bootloaders: A bootloader is a program that is responsible for loading the tasks necessary for at the boot time. Additionally, it loads the operating system. In the world of technology, a bootloader is also called boot manager. As I have already explained, this program is kickstarted after the BIOS are done with performing the initial checks on hardware. Now we know that the kernel and parameters lie on root filesystem.
Let's take a brief look at what is the job of a bootloader.

A bootloader has to select one kernel from multiple options. Secondly, with the aid of a bootloader, users can manually override names of kernel images, as well as parameters. It also allows you to switch between different kernel parameters.

Bootloaders are now not the same as before. They have become more advanced with added features like the option of history, as well as menu. However, experts still believe that the basic job of the bootloader is to offer different options regarding the kernel image as well as selection of parameters.

Let's take a look at multiple bootloaders that you may come across when you enter the field. GRUB, LOADLIN, LILO, coreboot are some of the most popular bootloaders of this age. Let's explore GRUB.

Getting into the Details of GRUB

Bootloaders are one of the reasons due to which users run away from Linux operating systems. If you are Windows OS user, you don't have to bother about the bootloader, but this is not the case with Linux operating system. As a Windows user, you switch on the laptop or your personal computer and that's it. If there is a problem in the booting system, you can simply run the Windows Recovery program to repair the damage and bring the operating system to its normal health. You just have to click on the button and the rest is the job of the operating system itself. That's quite amazing, but the biggest problem with it is that you cannot learn anything about the insides of the operating system. Sylvia had been a Windows user for ten years and she

didn’t have the faintest idea about what a bootloader is, what is its job, and how can you use it?

Linux takes the tough course. It makes users learn the intricate details about complex things in the operating system. It reroutes our minds toward the background technology of the operating system. Let’s move on to the bootloader itself to understand how it processes. So, how does it work?

You make your computer boot and as I have explained, the BIOS, after running some checks, passes the control of the machine to the boot device that differs from user to user. It can be a floppy or a hard disk drive. We already know that the hard disk has different sectors. The first of which is known as Master Boot Record (MBR).

GRUB drives out the MBR code and fills in its own. It helps users navigate the file system, which facilitates users to select the desired kernel image as well as configuration. GRUB is the short form of *Grand Unified Boot Loader*. Just explore the menu of the GRUB to learn more about it. Mostly the bootloader is hidden from the users and in order to access the bootloader, you should hold the SHIFT button for a while immediately after the BIOS startup shows up. There also is an automatic reboot problem. To avoid that, press ESC and disable the timeout of automatic reboot which starts counting right after the GRUB menu shows up. If you want to view the configuration commands of the boot loader, press e to enter the default boot option.

Getting familiarized with GRUB

Let’s get to know the GRUB notation. Let’s take a look at the GRUB entry.
(hd0,1)
If you look at the entry, you will know that the brackets are an integral part of the entry. You remove them, you disturb the syntax. All the devices in GRUB have brackets on them. The hd means hard disk. If the device is other than the hard disk, the value will be replaced with fd (floppy disk) and cd for CD-ROM. Please note that there also are two numbers. The first number refers to the number of the hard drive in the system. Here it denotes the first hard drive. Similarly, 1 denotes the second hard drive while 2 denotes the third har

drive. That’s how it goes on.
You can change the value of the integers by putting in the number of the hard disk that you want to explore. It is evident from the entry that GRUB cannot differentiate between different types of drives like SCSI and IDE. In the GRUB menu the primary partitions can be identified from integers from 0 to 3 where 0 is for the first hard disk while 3 is for the fourth partition. Logical partitions move from 4 to upwards.

GRUB can search all partitions for a UUID to locate the kernel. If you want to see how the GRUB works on the Linux operating system, just hit C while you are at the editor and you will reach the GRUB prompt. Something similar to the following will appear on the screen of the editor.
grub>

This is the GRUB command line. Get started by entering any command here. In order to get started, you need to run a diagnostic command.

```
grub> ls
(hd0) (h0,msdos1)
```

The result will be a list of devices that are already known to the GRUB. You can see that there is just one disk, the hd0, and one partition, (hd0,msdos1). The information we can get from this is that the disk contains an MBR partition table.
To delve deeper into more information, enter 1s -1 on the GRUB command line.

```
grub> ls -1
```
This command tells us more about the partitions on the disk.
You can easily navigate files on the GRUB system. Let’s take a look at its filesystem navigation. First of all, you need to find out about the GRUB root.

```
grub> echo $root
hd0,msdos1
```

You should use the ls command in order to list the files as well as directories

in the root. Let’s see how to do that.

grub> ls ($root)/
You may expect a list of directory names that are on that particular partition. The filesystem and the directories can eb listed in the form of *etc/,bin/*, and *dev/*.

How to Configure GRUB

The configuration directory has the central file titled *grub.cfg*. In addition, these multiple modules that are loaded on the system are also part of the configuration system. These modules are named as *.mod*. This should be the beginning of the configuration system.

BEGIN /etc/grub.d/00_header

All the files in the directory /etc/grub.d are multiple shell script programs. By combining they make up the central configuration file that is grub.cfg. This is the default shape of the GRUB configuration. Now let’s move on to the function which allows you to alter the command. The short answer is that you can do that easily. You can customize it according to your needs. You know that a central file for configuration exists. You just have to create another file for customization purposes. You can name the file as *customized.cfg*. This will be added to the configuration directory of GRUB. You can find it following this path: */boot/grub/custom.cfg.*

There are two options concerning customization. The directory of configuration can be accessed at /etc/grub.d. This directory offers 40_customa and 41_custom. 40_custom can be edited on your own without any aid, but it is prone to be weak and unstable. The other one that is 41_custom is much simple than the previous one. This file contains a string of commands ready to be loaded when the GRUB starts. A small example is that Ubuntu, a Linux distribution network, allows users to edit the configuration settings by adding memory tester options (memtest86+).

If you want to write a fresh configuration and also want to install it on the

system, you can do that by writing the configuration to the directory with the help of using the -0 option. Let’s see the syntax to conclude the configuration process.

grub-mkconfig -o /boot/grub/grub.cfg.

For users of the Ubuntu Linux distribution system, things are pretty much simple and easy. All you need is to run the following command.
install-grub

How to Install GRUB

Before you move on to the installer, please read what this bootloader requires for installation purpose. Please determine the following.

- GRUB directory, as seen by the current system, should be properly analyzed. /boot/grub is the directory’s name usually. The name can be something else if you are installing GRUB on another system.
- You have to determine what device the GRUB target disk is using.

As GRUB is a modular system, it needs to read the filesystem which is contained in the GRUB directory.

#grub-install /dev/sda
In this command, grub-install is the command used to install the GRUB, and /dev/sda is the current hard disk that I am using.

If you do it wrong, that can cause many problems for you because of its power to alter the bootup sequence on the system. This increase the importance of this command. You need to have an emergency bootup plan if something actually goes wrong along the way.

Install GRUB on an External Device

You can also install GRUB on an external device. You have to specify the directory first on which it will install. For example, if you have */dev/sda* on your system. Your system will see the GRUB files in */mnt/boot/grub*. You

have to guide the system by entering the following command.

grub-install –boot-directory=/mnt/boot /dev/sda

Let’s see how GRUB works

It consists of multiple steps. Let’s take them on one by one.

- We now know the drill of how the Linux operating system starts. BIOS kicks off in the start. When it has run necessary checks on the device, it initializes the device’s hardware. After that, it tries to trace out the boot-order in order to get the boot code.
- When it has found the boot code, the next step is the loading of BIOS or the firmware. Then comes its execution. After that is the time for the start of GRUB.
- GRUB starts to load on your Linux operating system.
- First of all, the core kickstarts. Now GRUB is in a better position to access your hard disks and the filesystems that you have stored on the system.
- Then GRUB moves on to identify the partition from which it has to boot. Next step is to load the configuration.
- Now, as a user, this is the time when you have the power to alter the configuration if you need it to be.
- If you don’t want to alter it, the timeout will end shortly. Now GRUB will execute the default or altered configuration.
- While trying to execute the configuration, GRUB loads some additional code in the form of modules in the partition from which it has booted.
- Now GRUB runs the boot command and execute the kernel.

I share my weekend routine with Sylvia. I am an enthusiast of outdoor activities like hunting by a crossbow out in the wild. Sylvia likes cycling in rough and hilly terrains. We really enjoy when we are together on the weekends. It was rare for her to miss a weekend. But finally, she missed it not one but three in a row. So, I phoned her to know what was the matter. To which she responded that her limited knowledge of Linux was not helping her at the office. She was worried because she was highly pressed for

wrapping up her work while most of the time struggled with the learning process. This time it was the user space and the environment. Though it is not that difficult to learn, it takes time before you get used to it
Not that much time, I must admit, that Sylvia has consumed. With the right guidance, you can move through it like a pro. The more you practice the better understanding you will have on it. User space refers to the point where the kernel starts. How the user space moves on. Let's see in the following steps.

- init
- low level services start like udevd as well as syslogd
- The network is configured.
- Services like printing go on.
- Next come the login shells, the Graphical User Interface (GUI).
- Other apps start running that you might have installed on the Linux operating system.

The init

This is a user space program which can be located in the /sbin directory. You can locate the directory if you run the PATH command. PATH is a variable. I'll discuss this variable in-depth in the upcoming chapters. System V init, Upstart and System V init are implementations of init in its Linux distributions. Android has its own unique *init*.
System V init triggered a sequence that required just one task while the system starts up. This system appears easy, but seriously hampers the performance of the kernel. In addition, advanced users abhor this kind of simplified startup. The system starts up following a fixed set of services, so if you want any new service to run on the system, no standardized way is available here in order to accommodate any new components. So that's a minus.
On the other hand, Upstart as well as systemd rectify the performance issue. They accommodate several services to take a start in parallel to pace up the boot. The systemd is for the people who are looking out for a goal-oriented system. You will have the flexibility of defining your target and the time span

you need to achieve the target. In addition, it resolves the target. Another attractive option is that you can delay the start of a particular service until you want it to load.

Upstart receives different events and runs different tasks that result in the production of more events. Consequently, Upstart runs more jobs. As a user, you can get an advanced way to track services. These latest *init* systems are free of scripts.

Runlevels in Linux

When you boot a Linux operating system, it moves into the default runlevel and also tends to run the scripts that are attached to the specific runlevel. Runlevels are of different types and you can also switch between them off and on. To quote an example, there is a runlevel specifically designed for system recovery and tasks related to maintenance of the operating system. As an example, System V init scripts are used as default runlevel. Ubuntu uses the same.

We are well aware of the fact by now that *init*, that is launched by Linux in the start, in turn, launches other system processes. If you want to control what *init* should launch and what not, you should edit the startup script which the *init* reads. An interesting thing about *init* is that it has the ability to run different runlevels like one for networking, another for the graphical user interface. Switching between the runlevels is also easy. All it takes is a single command to jump from the graphical desktop to the text console. It saves lots of time. Let’s move further into the runlevels to understand them better.

System V Runlevels

There are different sets of processes running in the system like *crond.* These runlevels are categorized from 0 to 6. A system starts with a specific runlevel but shuts down using another runlevel that has a particular set of script in order to properly close all the programs in a proper way and also to stop the kernel. If you have a Linux system in place you can enter the following

command and check in which runlevel you are at the moment.

[aka@localhost ~]$ who -r

Your screen will tell you the stage of your runlevel on a scale from 0 to 6, and also the date and time at which the runlevel was created on the system. Runlevels in the Linux operating system has many jobs to do, such as differentiating among shutdown, the startup, the text console mode and the different user modes. The seven numbers from 0 to 6 can be allocated to different channels. Linux operating systems that are based on Fedora allocate 5 to the graphical user interface and 4 to the text console.

In a standard runlevel o is allocated to shutting down of the system, 6 is allocated to the reboot while 1 is allocated to the single user mode.

I know you are finding these runlevels quite interesting, but they are now going obsolete.

system init

The system init can help you in better handling of the system bootup and also aids you in handling services like *cron* and *inetd*. One feature that users love it is its ability to delay the start of operating system and other services until they are imperative. The full form of systemd is System Management Daemon. The *systemd* is now replacing *init*, and there are some pretty solid reasons behind this shakeup. Let’s see how *init* starts the system.

One task starts when the last startup returns successful and gets loaded up on the memory of the operating system. This causes undue delay in the booting time of the system, which resulted in frustration for the users. Now it is quite logical to think that system adds some fuel to speed up the process, but sorry you are wrong. Well, it does decrease the booting time, not by putting in some extra gas, but by removing the speed breakers, like finishing the necessary tasks neatly.

It does so much on your system that its responsibilities are somewhat difficult to grasp in a single discussion, but I'll try my best. Let's see what it does on your system.

- It improves efficiency by properly ordering the tasks, which makes the booting process simpler.
- It loads up the configuration of your system.
- The default boot goal, along with its dependencies, is determined.
- Boot goal is activated.

The difference between *systemd* and *init* is the flexibility which the former has to offer to the user. The *systemd* just avoids unnecessary startup sequences.

The Linux Environment

It was the fall. Yellowish red leaves were falling from the trees like the angels who were banished from the heavens. Pale and defeated, they rested on the pathway and the greenbelt along the road. An old woman with a cane in hand was trying to cross the road. Luckily, she got help from a young blondie lady who happened to be passing by. It was Sunday. The sun was far into the thick clouds that were hovering over the city's skyscrapers, but it was balmy, the kind of weather you like to visit a beach. To the beach did I go. But wait a minute. Did I say something about the young blondie lady who was helping the lady? Oooops! That's Sylvia. And yes, she did accompany me to the beach.

"Lovely weather today? Mind a drink?" and she pulled out a bottle of peach pulp. "I don't mind that." I replied with a smile. "Well, well, well, finally I have started getting acquainted myself with Linux," she started. I stared at her with a little frown that wasn't matured yet on my face. "For God sake. We are on the weekend. If by a stroke of the ill fate, you have joined me on this beach, please don't spoil the moment." But she was unstoppable as are with newbie learners of Linux and programming languages.

I gave in. She was curious about the Linux environment on that particular day. Well, the subject was a juicy one, so I went on trying to educate her more about it. By imaging about the environment of Linux, and the beach where I was out sunbathing, the image before your eyes might be something interesting. Perhaps your brain is trying to structure an analogy between the environment for Linux and the general environment that surrounds us.

Let me explain. Yes, it is more or less the same. Just like we live in the environment; the shell window is also surrounded by an environment of its own. Whatever data that you store in the Linux environment is accessible to programs for the determination of facts about the configuration of the system. It is pertinent here to mention that the environment can be used to customize the shell experience. You can find the environment and shell variables in the environment. We will be using two commands to examine the environment; the set command and the printenv command. The latter command shows only the environment variables. I will not expand it here as you are going to see lots of it in the next chapter. So, let's run the set command. In order to do away with a long list of variables and other info, I'll pipe the output by pairing up the set command with the less command.

```
[aka@localhost ~]$ set | less
BASH=/bin/sh
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extqu
ignore:histappend:hostcomplete:interactive_comments:login_shell:progcomp:
ars:sourcepath
BASHRCSOURCED=Y
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
```

```
BASH_VERSINFO=
([0]="4" [1]="4" [2]="23" [3]="1" [4]="release" [5]="riscv64-koji-
linux-gnu")
BASH_VERSION='4.4.23(1)-release'
COLUMNS=80
CVS_RSH=ssh
DIRSTACK=()
EUID=0
GROUPS=()
HISTCONTROL=ignoredups
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=localhost
HOSTTYPE=riscv64
IFS='
'
'
LANG=en_US.UTF-8
[aka@localhost ~]$
```

The set command distinguishes itself from the rest of the lot by displaying the information in a near alphabetical order which is quite user-friendly. In addition, to that you can use the echo command to pull out information on the variable you need.

```
[aka@localhost ~]$ echo $HOSTNAME
localhost
[aka@localhost ~]$ echo $HOSTTYPE
riscv64
[aka@localhost ~]$ echo $HOME
/root
```

Moving on with the exploration of the shell environment I'll test the alias command.

```
[root@localhost ~]$ alias
egrep='egrep --color=auto'
fgrep='fgrep --color=auto'
grep='grep --color=auto'
l.='ls -d .* --color=auto'
ll='ls -l --color=auto'
ls='ls --color=auto'
mc='. /usr/libexec/mc/mc-wrapper.sh'
which='(alias; declare -f) | /usr/bin/which --tty-only --read-alias --read-funct
ions --show-tilde --show-dot'
xzegrep='xzegrep --color=auto'
xzfgrep='xzfgrep --color=auto'
xzgrep='xzgrep --color=auto'
zegrep='zegrep --color=auto'
zfgrep='zfgrep --color=auto'
zgrep='zgrep --color=auto'
[aka@localhost ~]$
```

Some of the other important variables include HOME, LANG, SHELL, DISPLAY, PAGER and OLD_PWD. Each has its own function like HOME leads you to the pathname of the home directory. EDITOR tells you about what kind of program you are using to edit the shell scripts. Similarly, DISPLAY shows the name of the display just in case you are running a graphical environment.

Some additional variables that are interesting as well as very handy are TZ which is used to specify the timezone, USER which tells your username, PS1 which explores the contents of the shell, and the PWD which explores the directory in which you are currently working.

Chapter 4: Package Management & Storage on Linux Systems

When it comes to Linux distribution, most newbie or casual users are more concerned about the color schemes and different other unnecessary features when selecting which distribution best suits them. What they forget to take into consideration is the fact that the most important thing is the packaging system of Linux in combination with the support community of each distribution system. We know it is open source. The software is dynamic and not static like the Windows. It just keeps on changing by contributions from the community members.

With the help of package management, we can install and maintain software on our personal computer systems. The following section will tell you about the tools to deals with package management. All these tools are linked to the command line. Some people may question if Linux distributors offer a graphical user interface why should we be forced into the command line yet again. The answer is simple. Command line tools offer us the completion of some impossible tools that the graphical user interface find impossible to conclude.

The Packaging System

The packaging systems differ from various distribution systems, and they are also distribution specific. You cannot use one packaging system on multiple distributions. In general, there are two main packaging systems:

- The Debian (.deb)
- The Red Hat (.rmp)

Most Linux distributions fall under these two main camps, but the packaging system is not limited to the two. There are some important exceptions, like Slackware and Gentoo.

Ubuntu, Linspire and Debian use the (.deb) camp. Mandriva, Fedora, Red

Hat Enterprise Linux, and PCLinuxOS use the (.rpm) camp.

How it Works

Usually software is bought from the market and then installed from a disk. You run the installation wizard and get the app on your system. This is a standard process on a Windows system. Linux is different. You don't have to buy anything as almost all software can be found on the web space for free. The distribution vendors offer the code compiled and stuffed in package files or they put it as source code which you have to install manually. The latter demands pro-level skills.

A package file can be dubbed as the basic unit of Linux software. It can be further explored into sets of files that together make up the software package. A bigger question is, what does a package have? It has multiple programs in addition to including metadata which shows the text description of the file as well as the contents. There are package maintainers who create the package files. These package maintainers can be but not always are employees of the vendors who distribute the Linux system. The package maintainers receive the source code for the software and produces the metadata and installation scripts. They can opt to contribute to the source code or leave it as it is received.

Repositories

The distribution vendors do most of the work related to creating these packages. When they have been prepared, vendors or any third parties who created them place them in the central repositories. The central repositories are like mines of package files. The package files in the repositories are specifically created for the distribution.

There can be more than one repository in a Linux distribution. Let's take a look at a few:

- Testing repository: this repository is filled with packages that are usually used by skilled Linux users who are looking forward to testing bugs in the packages before they reach a common user.
- Development repository: another repository is the development

repository, which contains the packages under development and will soon be included in the next release by the distribution.

- Third party repository: well, that's what it is called. This repository contains packages that are created by third parties but cannot be made a part of the distribution's major releases out of legal bars. Countries that have pretty relaxed laws related to software patents allow the third-party repositories to be released by Linux distribution. Although they can be included in the distribution's release, yet they are not part of the distribution and must be included manually in the configuration files.

The Dependencies

Linux programs need support from several software components to work efficiently. Some packages demand a shared source to get the work done. This shared source, such as a shared library, is dubbed as the dependency for the package file. Therefore, most of the Linux distribution systems also contain their dependencies in the package so that the software functions smoothly.

The tools to manage packages

There usually are two types of tools for package management:

- The low-end tools are used to tackle basic tasks like installation and uninstallation of package files on the system.
- The high-end tools are used to conduct meta-data search and resolve the dependency problem.

Let's take a look at some common tools that come with Linux distribution systems.

Low-end tools

The dpkg is considered as a low-end tool for the Debian style while rpm is a low-end tool for Red Hat Enterprise, CentOS and Fedora.

High-end tools

The aptitude and apt-get are some high-end tools for the Debian style while yum is considered as a high-end tool for Red Hat Enterprise, CentOS and Fedora.

You can use these tools to wrap up different key tasks, such as locating a particular package from the repositories.

You can install a package

You can use the following commands to install a package in the Debian style.

- *apt-get install package_name*
- *apt-get update*

For Red Hat style, use the following command.

- *yum install package_name*

Also, you can install a package right from a particular package file. Let's see how to do that. This is for the package files that are not part of a repository and are installed directly. Let's see the commands:

For the users of the Debian style, the following command is used.

- *dpkg –install package_file*

For the users of the Red Hat style, the following command is used.

- rpm -i package_file

You can also delete a particular package from the system by using these simple tools. Let's take a look at the commands
For the Debian style users, the following command tool is the best.

- *Apt-get remove package-name*

For the Red Hat style users, the following command tool works well.

- *yum erase package-name*

You can do more with the help of command tools. Suppose you have installed a package system and now you want to update it. You can do that with the following tools. As with the above-mentioned tasks, the tools for the Debian and the Red Hat were different, the same is the case with the updating tools.

- For the Debian style users, the command tool for updating the package system is *apt-get update*. You can also replace *update* with *upgrade*. Both do the same job.
- The Red Hate style users can with the command *yum update*.

If you have downloaded an updated version of a package from a non-repository source, you can install it by replacing the previous version. The commands for the job are as under:

- The Debian style users should enter *dpkg –install package-file.*
- The Red Hat style users should enter *rpm -U package-file.*

Replace package-file with the name of the file that you want to update. When you have installed more than one packages over a particular course of time, and you want to view them in the form of a list, you can use the following tool to see the record.

- For the Debian style users *dpkg --list* is the tool to view the list.
- For the Red Hat style users *rpm -qa* should be entered.

When you have installed a package, you check its status any time to make sure it is there on your Linux operating system. There are some handy tools available for the purpose. Let’s check out.

- The Debian style users should enter type *dpkg --status package-name.*
- The Red Hat style users may use *rpm -q package-name.*

In addition to checking the status of an installed package, you also can explore more information about it with the help of the following commands.

- The Debian style users may enter *apt-cache show package-name.*
- The Red Hat style users can type *yum info package-name.*

Storage Media

We know about hard disk drives, floppy disks, and CD drives from the Windows operating system. If you think Windows makes it easy for you to handle them, Linux has more to offer. You can handle all the storage devices like the hard disk drive, the virtual storage such as the RAID, the network storage, and Logical Volume Manager (LVM). I’ll explain some key commands that you can use to handle all the storage devices easily as well as efficiently.

The commands for mounting and dismounting storage devices

You can mount and dismount a storage device with the help of some simple commands. If you are using a non-desktop system, you will have to do that manually owing to the fact that servers have some complex requirements for system configuration.

There are some steps involved to accomplish the objective like linking the storage device with the filesystem tree. Programmers call it mounting. After this, you the storage device starts participating in the functions of the operating system. We have already learned about the filesystem tree earlier on. All the storage devices are connected to the filesystem tree. This is where it differs from the Windows system, which has separate trees for each device. As an example, see the C:\ and D:\ drives. You can explore the list of the devices with the help of the command */etc/fstab*.

If you want to view the list of the filesystems that you already have mounted on the Linux computer system. We call it the mount command. By entering it on the command line, you will get the list of the mounted filesystems.

[aka@localhost ~]$ mount

The listing will show the mount-point type as well as the filesystem-type.
You can determine names of the devices

Although determining names of the devices is considered as a difficult task to do, yet there are some easy ways to master it. To make out the names of the devices, we have to first list the devices on the Linux operating system.

[aka@localhost ~]$ ls /dev

Let’s analyze the names of different devices.

/dev/hd* : These are PATA disks on the older systems. The typical motherboards have a couple of IDE connectors. Each of the IDEs had a cable

and a couple of points for drives to get attached to them. The first drive is dubbed as the master drive while the other one is named as the slave drive. You can find them by referring them as /dev/hda for the master drive and as /dev/hdb for the slave drive. These names are given to the drives in the first channel. For the second channel, you can refer the master drive as /dev/hdc and the list goes on. If you see a digit at the end of the titles, remember that it refers to the partition. To quote an example, /dev/hda1 will be considered as the 1st partition to on the first hard drive. In this scenario where you have partitions, the name /dev/had will refer to the full drive.

/dev/fd*: This name refers to the floppy disk drives.
/dev/sd*: This refers to the SCSI disks, which includes all PATA and SATA hard disk drives, the flash drives, the USB mass storage drives like the portable music players and the ports to connect digital cameras.

How to create new filesystems?
On Linux, you have the freedom to create new filesystems. Suppose you want to reformat the flash drive with the help of a Linux native filesystem instead of the FAT32 system. You can do that by following two simple steps, such as the following:

- You can create a fresh layout for a new partition if you don't like the current partition.
- Secondly, you can create a new, but empty, filesystem.

This command will format your current hard disk partitions so remember to use it on a spare disk and not on the one that stores your important data. A single mistake in the name of the drive can result in erasing data on the wrong drive.

You can use the *fdisk* program to interact with the hard disk drive and other flash drives on your Linux computer system. The *fdisk* program allows you to edit, create or delete partitions on one particular device. In this case, I'll be dealing with the flash drive.

[aka@localhost ~]$ sudo umount /dev/sdb1
[aka@localhost ~]$ sudo fdisk /dev/sdb

The first command will unmount the flash drive and with the second command you can invoke the *fdisk* program. The following will appear on the screen.

Command (m for help):

When you see that on the screen, enter ‘m’ on the keyboard which will display a menu that will prompt a command action from the user. By pressing *b,* you can edit the disk. By pressing *d,* you can delete a particular partition of the device. In this particular case, we are into the flash drive so the *d* key will delete a partition from the flash drive. You can press *l* to list the partitions that are not yet known. You can print the menu again any time by pressing *m*. Also, you can add a new partition to the device by pressing *n*. By pressing *q,* you can quit the menu without saving the necessary changes and by pressing *t,* you can change the system id of a partition.

First of all, you are required to see the layout for a particular partition. Press *p* to see the partition table for the flash drive device.

Now suppose we have a dummy storage device of 32 MB having one partition, and this device can be identified as Windows 95 FAT32. From the menu, you can see that there is an option for listing the known partition types when you press *l*. Now you can see all the possible types. It is pertinent to mention here that the b in the *sdb* refers to the partition id. In the list, you can recognize your partition with the help of the system id *‘b’*. You can change the system id for your particular partition by entering *t* on the command line. The changes have been stored on the system until now. Remember that you have not touched the device as of now. Now enter *w* to modify the partition table to the device and make an exit. You have successfully altered the partition. If at any time you decide to leave the partition without altering it, press *q* and exit the command line.

Now that you have learned to edit the flash drive device partition, we should move on to create a new filesystem with the help of *mkfs* command. Read it as make filesystem. An interesting thing is that you can create filesystems in

multiple formats.
If you want to create ext3, add -t option to the command mkfs. Let’s see the syntax for the command.

```
[aka@localhost ~]$ sudo mkfs -t ext3 /dev/sdb1
```

There will be plenty of information displayed on your screen. You can get back to the other types by the following commands. To reformat the device to the FAT32 filesystem, you can draft the following command and enter on the command line.

```
[aka@localhost ~]$ sudo mkfs -t /dev/sdb1
```

You can see that editing a partition, formatting it, and creating a new filesystem are pretty easy on the Linux operating system than that of the Windows system. You can repeat the following the above-mentioned simple steps whenever you want to add a new device to your computer system. This kind of freedom of altering the type of a storage device and editing it is unavailable in the Windows operating system. In the above example, I described the entire process with the help of a dummy flash drive. You have the liberty to practice the above commands on a hard disk drive or any other removable storage. Whatever suits you, you can go for it.

Chapter 5: Linux Environment Variables

When talking about the bash shell, it is relevant to know about its features like the environment variables. These variables come handy in storing information about a particular shell session as well as the environment in which you have been working. With the help of these variables, you can feed information in the memory which can be accessed through running a script or simply by running a program.

Global Variables

This is simple to learn. These are the variables that can be accessed globally. In simple words, you can access them at any phase of the program. When you have declared a variable, it is fed into the memory of the system while you run the program. You can offer alterations in any function that may affect the entire program. Global variables are always displayed in capital letters.

```
[aka@localhost ~]$ printenv
```

You have had a full list of global environment variables. Most of them are set during the login process. If you want to track down values of individual variables, you can do that with the help of *echo* command. Just don't forget to add the $ sign before the variable to get its value. Let's look at the syntax.

```
[aka@localhost ~]$ echo $PWD
/root
[aka@localhost ~]$
```

Local Environment Variables

These can only be seen in the local process. Both global environment variables and local environment variables are equally valuable. It is difficult to get a list of the local environment variables because you can't just run a single command for the purpose. The set command shows the environment variables set for a particular purpose.

```
[aka@localhost ~]$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extqu
ignore:histappend:hostcomplete:interactive_comments:progcomp:promptvars:
```

th
BASHRCSOURCED=Y
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=
([0]="4" [1]="4" [2]="23" [3]="1" [4]="release" [5]="riscv64-koji-linux-gnu")
BASH_VERSION='4.4.23(1)-release'
COLUMNS=80
CVS_RSH=ssh
DIRSTACK=()
EUID=0
GROUPS=()
HISTCONTROL=ignoredups
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=localhost
HOSTTYPE=riscv64
IFS=$' \t\n'
LANG=en_US.UTF-8
LESSOPEN='||/usr/bin/lesspipe.sh %s'
LINES=30
Please take into account the fact that all the global variables also are included in the *set* command details.

You can set your own local environment variables
You can set your own variables in the bash shell. You can assign value to a variable by using the equal sign.
[aka@localhost ~]$ test=world
[aka@localhost ~]$ echo $test

```
world
[aka@localhost ~]$
```

The above example is fit for assigning simple values. In order to assign a string value with spaces between words, you need to try something different. Please note that you have to use lower case letters in order to create a new variable. This is important because you can get confused by seeing the environment variables in the capital case. Let’s take a look at the following syntax:

```
[aka@localhost ~]$ test=theskyisazure
[aka@localhost ~]$ echo $test
theskyisazure
[aka@localhost ~]$ test=the sky is azure
bash: sky: command not found
[aka@localhost ~]$ test='the sky is azure'
[aka@localhost ~]$ echo $test
the sky is azure
```

You can see that the difference lies in the use of single quotation marks.

How to remove environment variables

You can remove the variables with a simple step. Let’s see the syntax of the *unset* command.

```
[aka@localhost ~]$ echo $test
theskyisazure
[aka@localhost ~]$ unset test
[aka@localhost ~]$ echo $ test
[aka@localhost ~]$
```

If you remove the environment variable from the child process, it still remains in the parent process. You have to remove it from the parent process separately.

```
[aka@localhost ~]$ test=azure
[aka@localhost ~]$ export test
[aka@localhost ~]$ bash
[aka@localhost ~]$ echo $test
azure
```

[aka@localhost ~]$ unset test
[aka@localhost ~]$ echo $test

[aka@localhost ~]$ $exit
[aka@localhost ~]$ echo $test
azure
[aka@localhost ~]$

You can clearly see that the environment variable was first exported so that it may become a global variable. The unset command was applied while I was still in the child process. When I switched to the parent shell, the command was still valid. That's why you need to delete it from the parent shell as well.
Check out the default shell variables
The bash shell contains environment variables that originate from the shell itself. Let's check out some of the variables.

[aka@localhost ~]$ echo $PATH
/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin

You can see different directories in this command. The shell looks for commands in these directories. More directories can be added to the command just by inserting colon and then adding the directory. A default value is assigned to the variables. Let's take a look at the following table.
PATH As I have shown you by an example, it carries the list of the directories.
HOME This refers to the home directory of the user.
IFS This brings out a list of characters
CDPATH You will get a list of directories that are separated by a colon.
Importance of the PATH Variable

This is perhaps the most important variable in the Linux system because it guides the shell to locate the commands for execution right after you enter on the command line. In case it fails to locate it, an error message will be displayed which can look similar to the one given as under:
[aka@localhost ~]$ newfile

-bash : newfile: command not found
[aka@localhost ~]$

It is an environmental variable in the Linux operating system. It becomes active as you enter a command or a shell script. A shell script is just like a mini program that offers text-only user interface for the users of Unix-like systems. It can read commands and then execute them accordingly. It is important to consider here that PATH with all capitals must not be replaced with a path with all the lower-case letters.

The path variable is a completely different thing as it is the address of a directory. Relative path alludes to the address in relation to the directory you are currently in. There also is an absolute path as already discussed, which is the address in relation to the root directory.

On the other hand, the PATH will turn out a series of paths distinguished by colons and stored in the form of plain text files. For revision and explanation purposes let's take a look at how it is executed.
[aka@localhost ~]$ echo $PATH
/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin

You can add new directories to this string. Let's try it.
[aka@localhost ~]$ echo $PATH
/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin
[aka@localhost ~]$ PATH=$PATH:/home/rich/azuresky
[root@localhost ~]# echo $PATH
/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/home/rich/azuresky
[root@localhost ~]$

It is important to mention that each user in the Linux operating system has a different PATH variable than others. Upon installation of an operating system, a default variable for your PATH is set. After that, you can add more to it or completely change as it suits you.

Note: It is important to mention that the root user's PATH variable has more directories than any other user. An interesting thing about the PATH variable

is that either you can change it just for the current session or on a permanent basis.

How to find Environment Variables of the Linux System

There are multiple variables that a system uses for identification purpose in the shell scripts. With the help of system variables, you can easily procure important information for the programs. There are different startup methods in the Linux system, and in each system, the bash shell executes the startup files in a different way.
The Login Shell

This locates four distinct startup files from where it could process its commands. The very first file that is executed is the /*etc*/*profile*. You can dub it as the default file for the bash shell. Each you time you log in, this command will be executed.

[aka@localhost ~]# cat /etc/profile
/etc/profile

I have run this command while logged in as a superuser. The result on your window can be slightly different.
Just take a look at the export command stated as under:

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCON
You can clearly see that the export command ensures that the variables remain accessible to the child processes.

The $HOME
This one is a user-specific variable. Let’s run it to know what it has got.
[aka@localhost ~]# cat .bash_profile
.bash_profile

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

```
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
```

Interactive Shell

If you happen to start the bash shell without first logging into the system, you kickstart the interactive shell. You have the command line to enter command here. With the start from an interactive shell, the system does not load the /etc/profile file. It locates another file called .bashrc in the home directory of the user. Let's see how this startup file looks in the Linux operating system.

```
[aka@localhost ~]# cat .bashrc
```

It conducts a couple of key tasks; first is checking for the bashrc file in the /etc directory. The second one is to allow the user to enter aliases.

```
[aka@localhost ~]$ cat /etc/bashrc
```

Variable Arrays

You can use variables as arrays. The fun thing with arrays is their capacity to hold multiple values which you can reference for a complete array. You can simply cram multiple values in a single variable by listing them with single spacing in parenthesis. Let's try it.

```
[aka@localhost ~]$ myworld=(one two three four five)
[aka@localhost ~]$ echo $myworld
one
[aka@localhost ~]$ echo ${myworld[2]}
three
[aka@localhost ~]$ echo ${myworld[1]}
two
[aka@localhost ~]$ echo ${myworld[4]}
five
```

You can see how easy it is to build up a variable array. You can bring out each value in the array with a special command. Now try the following to bring out the entire array.

```
[aka@localhost ~]$ echo ${myworld[*]}
one two three four five
```

In addition to this you can also unset a specific value from the array.

```
[aka@localhost ~]$ unset myworld[3]
[aka@localhost ~]$ echo ${myworld[*]}
one two three five
```

If you want to get rid of the entire array, you can do that by unsetting it.

```
[aka@localhost ~]$ unset myworld[*]
[aka@localhost ~]$ echo ${myworld[*]}

[aka@localhost ~]$
```

Variable arrays are not portable to other shell environments, that’s why they are not the foremost option to be used with Linux operating system users.

Chapter 6: The Basics of Shell Scripting

Sylvia was too much excited to learn the basics of the Linux operating system and the command line. In fact, she found it quite amazing and satisfying to get the job done by entering a short command, getting immediate results, and just then I told her that the major part of the Linux system has yet to come. This is the toughest part for newbies. “So, will I have to code?” she asked when I alluded to the hard part of Linux. I replied in the affirmative. I will walk you through the world of shell scripting in this chapter. Shell scripting is much like coding. It is exciting, thrilling and creative.

You have learned by now to use the command line interface (CLI). You have entered commands and viewed the command results. Now the time is ripe for an advanced level of activity on Linux that is shell scripting in which you have to put in multiple commands and get results from each command. You can also transfer the result of one command to another. In short, you can pair up multiple commands in a single step.
Let’s how it is done.

Draft syntax like the following.

[aka@localhost ~]$ date ; who

Now run this command. You will have the date and time first and the details of the user who is currently logged in the system right under this command. You can add commands up to 255 characters. That’s the final limit.

A tip: A problem will definitely get in your way when you try to combine multiple commands: remember them each time you have to enter them. An easy way to get rid of this issue is to save multiple commands into a text file so that you can copy and paste them when you need them. This definitely works for sure. You can also run the text file.

How to create a text file: You can create your customized file in the shell

window and place shell command in the file. Open a text editor like vim to create a file. We have already practiced how to create a text file in Linux command line. Let's revise it.

```
[aka@localhost ~]$ cat > file.txt
The Statue Of Liberty Was Gifted By France.
```

So, get ready to draft your first shell script. Let's roll on. Open a bash text editor and draft the script as under or something similar like that.

```
#!bin/bash
# Script writing is interesting.
echo 'The sky is azure!'
```

The second with # in the start looks like a comment. In the third line, we can see the echo command. Remember that you have to ignore everything after the # sign.

Let's try it on in the command line.

```
[aka@localhost ~]$ echo 'The sky is azure!' #I have entered a comment
The sky is azure!
```

Well, by now you might be thinking about the comments and their relation with the # sign. Then your mind will be distracted and perplexed to think about the first line of the script that also starts with the # sign. No, that's not a comment but rather a special line commonly known as *shebang*. This should come at the start of each script. You can save your file now with any name you desire. I saved it as *azure_sky*.

Displaying Messages

When you enter something in the command line, an output is definitely expected out of it that is displayed on the screen. You can also add text messages to let the user know what is going on in the script. The echo command does the wonder for you. It can show the string of simple text on the screen.

```
[aka@localhost ~]$ echo I hope you are writing the code in the right way.
I hope you are writing the code in the right way.
[aka@localhost ~]$
```

Now let’s try something exciting to see how it can go wrong and how to rectify what goes wrong.

```
[aka@localhost ~]$ echo I'll be happy if you don't go for swimming.
Ill be happy if you dont go for swimming.
[aka@localhost ~]$ echo "I'll be happy if you don't go for swimming."
I'll be happy if you don't go for swimming.
[aka@localhost ~]$ echo 'John says "I'll be happy if you don't go for swimmir
'"
John says "Ill be happy if you dont go for swimming."
[aka@localhost ~]$
```

The command went wrong, and the display was not what we had expected. So. We had to rectify it with the help of quotation marks.

Shall I Execute it Now?

It is time to execute the file you have just saved. Let’s run the command.

```
[aka@localhost ~]$ azure_sky
bash: azure_sky: command not found
$
```

You have to guide the bash shell in locating your shell script file.

```
[aka@localhost ~]$ 1s -1 azure_sky
-rw-r—r—1 me     me     63 2019-08-26 03:17 azure_sky
[aka@localhost ~]$ chmod 755 azure_sky
[aka@localhost ~]$ 1s -1 azure_sky
rwxr-xr-x 1 me me        63 2019-08-26 03:17 azure_sky
```

But how will the command line find the script that you have just written? The script should have an explicit path name. In case you fail to do that, you are likely to see the following:

```
[aka@localhost ~]$ azure_sky
bash: azurer_sky: command not found
```

You must tell the command line the exact location of your script. Otherwise, you won’t be able to execute it. There are different directories in the Linux system, and all of them are in the PATH variable. I have already shown you how to see what directories are on the system, explore the PATH variable.

```
[aka@localhost ~]$ echo $PATH
```

/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/home/rich/azuresky
[aka@localhost ~]$

Now that we have got the list of the directories you can see and put your script in the directory you want to. Follow the following steps to save your script in a specific directory.
[aka@localhost ~]$ mkdir bin
[aka@localhost ~]$ mv azure_sky bin
[aka@localhost ~]$ azure_sky
azure_sky

If your PATH variable misses the directory, you can add it by the following method:
[aka@localhost ~]$ export PATH=~/bin:"$PATH"

More ways to format scripts: The ultimate goal of a good script writer should be to write a good script and maintain it smoothly. You should be able to easily edit the script like adding more to a particular script or removing from it. Also, it should be formatted in a way that it should be easy to read as well as understand.
Go for long option names
We have seen many commands both with long and short options name. ls command that is used to list directories in your Linux system has lots of options such as the following:
[aka@localhost ~]$ ls -ad
[aka@localhost ~]$ ls –all --directory

Both these commands are equivalent when it comes to the results. Users prefer short options for ease of use so that they may not have to remember long option names. Almost every option that I have stated in previous chapters has a long form. Though short form options are easy to remember, it is always recommended to go for the full names when it comes to script writing because it is more reader-friendly.

Indentation
Now long form commands are also not very easy to handle. You have to take

care to write them in the right format so that it may confuse the reader. You can use commas between many lines.

How to use Quotation Marks and Literals

This is quite technical. As a Linux user, you must know how and where to use quotation marks, commas and period. Why does a specific punctuation mark necessary for the script? Let us print the following string.

```
[aka@localhost ~]$ echo $100
00
```

The output should have been 100, but it was instead a double zero. If you are thinking that the command line couldn't understand the script and executed it what part it succeeded in comprehending, you are right. It got it all wrong. So, what should be done now? First, we need to understand why did it get it all wrong? Its reason is that the shell only saw 00. It considered $1 as a shell variable. That's what it is. You failed to make it easily readable. Perhaps you should add the quotation marks. Add them right away.

```
[aka@localhost ~]$ echo "$100"
00
```

You are getting it wrong. Still you get the same result. Are you frustrated by now? We have a solution.

```
[aka@localhost ~]$ echo '$100'
$100
```

We have got the result that we desired.

When you add the quotation marks to a script, you are on your way to creating a literal, which is a kind of string that runs through the command line fully intact. When you are looking forward to writing a script, you need to keep in mind that the shell analyzes the variables in a command before it runs it. It also performs substitutions if they must appear along the way, and when it is done with that, it forwards the results to the command line.

Let's see what problems may come your way when you are dealing with the literals. They can be more complicated than you might have thought. Let's assume you are looking into the /etc/passwd directory in order to locate the entries that match r.*t. This will help you locate usernames like the root and robot. See the command you will be using to execute your search.

```
[aka@localhost ~]# grep r.*t /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
systemd-coredump:x:999:996:systemd Core Dumper:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon
/null:/sbin/nologin
polkitd:x:998:995:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
pesign:x:994:991:Group for the pesign signing daemon:/var/run/pesign:/sbin/n
in
```

Well, it worked perfectly. When you execute the command, you will see that specific files are colored orange. It means you have got the information you were looking for. When it seems to be working fine, it fails sometimes for reasons unknown. That's why this trigger panic. If you are on the verge of getting panicked as Sylvia did when she did it all wrong, stop right there. The problem lies in your directories. Review them, and you will find the solution. In fact, let me walk you through the solution. If your directory has names like r.output and r.input, the command will not be able to interpret the command and the result will be like the one given as under:

```
[aka@localhost ~]$ grep r.output r.input /etc/passwd
```

If you want to keep a good distance from this kind of simple problems, you need to identify the characters that are likely to land you in trouble and also master the art of using quotation marks. A single mistake can put you away from the results you need.

Learning the use of single quotation marks*:*

As a Linux user, it might be frustrating to know that the shell is not going to leave the string alone. You might get fed up the constant interruptions of the shell toward the string. Single quotes, fortunately, can help you kill this frustration. Applying single quotes on your strings can solve the matter. Let's

jump to the example.
[aka@localhost ~]$ grep ‘r.*t’ /etc/passwd
Why these single quotes matter much in solving the problem? The shell takes all the characters, including spaces inside the single quotes as a single parameter. This why if you enter the command without single quotes, it will not work. That’s why when you are about to use a literal, you must turn to single quotes. Only in this way the shell will not attempt for substitutions.
As a matter of fact, the single quote marks operate as protectors for whatever you put inside them. You can say that it just kills the special meaning of the characters. In addition, the shell will be unable to accommodate variables and wildcards.

Mastering the art of the double quote marks.
Iin general, the double quotes just work like the single quotes. The only difference between the single and double quotes is that the latter is more flexible than the previous. For example, if you want to expand any variable inside the string, you should put the string inside the double quotation marks. Let’s do it.
[aka@localhost ~]$ echo "There is no * on my path: $PATH"
There is no * on my path: /usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/loc
al/bin:/home/rich/azuresky
[aka@localhost ~]$

You can see that the double quotation marks have allowed the $PATH to expand but kept the (*) from the same act. So, whatever has a $ sign before it will be expanded when you run the command.

Food for thought: If you want only variable and command substitution to run, you should go for the double quotation marks. Please keep in mind that wildcards will still remain blocked.

The Backlash
The third option is the backlash. You can use it to alter the meaning that the characters carry. In addition, you can also use the option to escape any kind of special characters that are within the text, including the quote marks.
Most users find it tricky when they are about to pass the single quote to the

command. To avoid any unexpected result, you should insert the backlash before the single quote marks.

```
[aka@localhost ~]$ "Path is \$PATH"
bash: Path is $PATH: command not found
[aka@localhost ~]$ "Path is $PATH"
bash: Path is /usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/home
/rich/azuresky: No such file or directory
[aka@localhost ~]$
```

The important thing to keep in mind is that the backslash, as well as the quote, should remain outside the single quotes.

```
[aka@localhost ~]$ echo I don\'t like Italian food
I don't like Italian food
[aka@localhost ~]$ echo "I don't like Italian food"
I don't like Italian food
[aka@localhost ~]$ echo I don'\"t like Italian food'
I don\t like Italian food
[aka@localhost ~]$
```

Now you know how to get away from a syntax error with the help of backlash.

Decision Making in Linux: The Conditionals

For conditionals, the Bourne shell has special structures like *if/then/else* and *case* statements. The conditionals are all about decision making process in Linux. You can set Linux to make some decisions using bash shell. You have to set the conditions in order to make the system take certain decisions. Generally, a condition is an expression after the evaluates a certain statement as true or false.

Let’s take a look at the *if* statement. If the system finds that the conditions that were set by the programmer are well met, it will allow the program to execute. Otherwise, it won’t allow it. Let’s take a look at the syntax of the *if* statement.

```
#!/bin/sh
if [$1 = the sky is definitely azure]; then
```

```
        echo ‘The first argument was “the sky is definitely azure”’
else
        echo -n ‘The first argument was “the sky is definitely azure”— ‘
        echo It was ‘”’$1’”’
fi
```

The above statement is for the single decision. For single decision, the statement can be as the following:

```
#!/bin/sh
if [$1 = the sky is definitely azure]; then
        echo ‘The first argument was “the sky is definitely azure”’
fi
```

For multiple decisions you can use the following statement.

```
#!/bin/sh
if [$1 = the sky is definitely azure]; then
        echo ‘The first argument was “the sky is definitely azure”’
then
[the condition 2]
else
        echo -n ‘The first argument was “the sky is definitely azure”— ‘
        echo It was ‘”’$1’”’
fi
```

How it goes can be understood by the following steps:

- First comes the *if* command. The shell runs it and collects the code you enter in the command.
- Remember the 1 and the 0. If the exit code is 0, the command stands executed and you will see the then keyword, ending up at the *else* or *fi* keyword. The whole process ends in *fi*.

Let’s take a look at the expanded version of the *if* command. I’ll add *elif* (else-if) in the expanded version.

```
NAME=”AHSAN”
if [“$NAME” = “SYLVIA”]; then
   echo “SYLVIA ADAMS”
elif [“$NAME” = “AHSAN”]; then
   echo “AHSAN SHAH”
```

```
else
   echo “It boils down to Adam and Jack”
fi
```

The problem with the conditional

Now we should talk about a slight glitch in the conditional statement. When you enter the above conditional statement, it may not run as you think just because $1 could be empty. You might have missed setting a parameter which results in aborting the command with an error altogether, but you can fix the problem by taking the following measures.

```
if [“$1” = the sky is azure]
then
if [ x”$1” = x”hi”]
then
```

The *if* command is versatile when it comes to using different commands.

Testing the conditionals with other commands

When you enter *if* in the shell window, whatever comes next to it is considered as a command. This helps us understand why we put a semicolon before *then*. If we don’t do that, we will have to write on the next line. Let’s add some other commands into the conditionals.

```
[aka@localhost ~]# if grep -q daemon /etc/passwd; then
> echo The user is hidden in the password file.
> else
> echo Look somewhere else. It is not in the password file.
> fi
```

&& and ||

The first one as expected means *and* while the second construct means *or*. Let’s see how you can run these commands.

firstcommand && secondcomman

Suppose the shell runs *command1* and the exit code turns out to be zero, the shell moves on to the second command in the construct and runs it.

The || construct is different from && in a way that it runs the second command even if the first command returns an exit code with a nonzero value.

```
[aka@localhost ~]# if [ahsan] && [sylvia]; then
> echo "both can go to the concert"
> elif [ahsan] || [sylvia]
> echo "only one can go to the concert"
> fi
```

More if-then features

The *if* command has gone an extra mile to fulfill users' expectations. You can add much more to the logical decision making. Some of the advanced features include the following:

- Double parenthesis
- Double square brackets

How and where to use double parenthesis

Double parenthesis command is amazing in a sense that it allows Linux users to use mathematical formulas in the script. You can add some advanced mathematical symbols in the script just like other programmers do while coding. This allows users more freedom while composing scripts. Let's move on right away to the syntax to make things clear.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #using double parenthesis
[aka@localhost ~]$ val1=5
[aka@localhost ~]$ if ((val1 ** 4 > 50)); then
> ((val2=$val1**2))
> echo "The square of $val1 is $val2"
> fi
The square of 5 is 25
[aka@localhost ~]$
```

Other command symbols you can use in the above script are given as under:

- Pre-increment : ++val
- Logical negation: !
- Right bitwise shift: >>
- Left bitwise shift: <<
- Post-increment: val++
- Bitwise negation: ~
- Logical AND: &&

- Logical OR: ||

The double brackets

Now the next special feature is the use of double brackets. This command is used in string comparisons. Its special use is in pattern matching.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #using pattern bashing
[aka@localhost ~]$ if [[$USER == r*]]
> then
> echo "Hey $USER"
> else "I cannot recognize you. Come back later."
> fi
```

If the command inside the double bracket matches the $USER variable and finds that it starts with the letter *r*, which in this case is *rae*, then the shell obviously executes the *then* section part of the script.

The Loops

Sylvia happens to be a Marvel fan. She never misses a single movie. One of her favorite characters is Doctor Strange, the accidental wizard. Well, I don't like these fantasy things but why I am talking about it is the fact that one day Sylvia half narrated and half described by gestures a scene from one his movies in which he fabricates by his magic a time loop to defeat the villain Dormamu. I loved the tale because of its sprightliness. Sylvia loved it more because she connected it with the loops in Linux. Doctor Strange traps Dormamu in a time loop in which he was supposed to remain endlessly until he agreed to Dr. Strange's bargain. Again, and again Dr. Strange appears and is killed by Dormamu because of the time loop which keeps repeating itself.

Sylvia linked the two loops and fed the lesson in her mind quite successfully. A smart thought though she could have understood it without Marvel. Let's see how it goes. Jump right away to the script.

```
[aka@localhost ~]$ #!/bin/sh
[aka@localhost ~]$ for str in earth sun mars moon saturn;do
> echo $str
> done
```

```
earth
sun
mars
moon
saturn
[aka@localhost ~]$
```

In the above, you can notice that the above script is a combination of different words. While some may seem familiar to you, others are difficult to interpret. Out of the above *for, done, in,* and *do* are shell keywords. Let me explain in words how the loop works.
There is a variable *str* in the above script. When you enter the above text in the shell window, the shell fills the variable with the value of the first of the space-delimited values that can be seen above after the shell keyword *in.* The first word here is *earth*. Then the shell executes the echo command. After that, the shell once again returns back to the *for* line to set fill in the variable with the next value that is in this case *sun*. It repeats the same exercise with the second value. The loop goes on until there is no value left written after the keyword *in.*

The *for* command fills in the variable with whatever the next values are in the list. We can use the *str* variable like any other variable in shell scripting. When the last of the iterations are done, the *str* variable keeps the last value it was filled with. Here an interesting thing is that you can change that. Let's see how to do that.

```
$ cat testfile
#!/bin/bash
# testing the for variable after the looping
for str in earth sun mars saturn
do
echo "The next planet is $str"
done
echo "The next planet to visit is $str"
str=Uranus
echo "We are going to $str"
$ ./testfile
```

```
The next state is earth
The next state is sun
The next state is mars
The next state is saturn
The next planet to visit is Uranus
We are going to Uranus
$
```

The *for* loop is always not that easy to use. Simple problems in the syntax can push you over the edge and you will land in utter darkness with no clue on what to do. Do you want to see an example? Let's do that.

```
$ cat testfile
#!/bin/bash
# example of the wrong use of the for command
for test in I was pretty sure it'll work but I don't know what went wrong
do
echo "word:$str"
done
$ ./testfile
word:I was pretty sure
word:itll work but I dont
word:know what went wrong
$
```

The single quote marks did the damage in the above script. This kind of error really catches the programmer off-guard. To resolve the problem, you can use backslash or double quotation marks to properly define the values. Another problem is the use of multiword.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #another wrong use of the for command
[aka@localhost ~]$ for str in The Netherlands Australia The United States Of
> do
> echo "I will visit $str"
> done
I will visit The
```

I will visit Netherlands
I will visit Australia
I will visit The
I will visit United
I will visit States
I will visit Of
I will visit America
I will visit Scotland
I will visit The
I will visit British
[aka@localhost ~]$

Now let’s see how to do it right by making use of the double quotation marks in the script. You will be astonished to see how simple it is to rectify it.

[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #How to do it right
[aka@localhost ~]$ for str in "The Netherlands" Australia "The United States America" Scotland "The British"; do echo "I will visit $str"; done
I will visit The Netherlands
I will visit Australia
I will visit The United States Of America
I will visit Scotland
I will visit The British
[aka@localhost ~]$

How about reading a list with the help of a variable?
Suppose you have a list of values that are stored in a variable. You can use the *for* command to iterate through the list.
[akalocalhost ~]$ #!/bin/bash
[aka@localhost ~]$ #we will use the for command to deal with the list
[aka@localhost ~]$ list="Hands Face Hair Feet Limbs Elbows"
[aka@localhost ~]$ list=$list"neck"
[aka@localhost ~]$ for parts in $list
> do
> echo "I have washed my $list"

> done
I have washed my Hands
I have washed my Face
I have washed my Hair
I have washed my Feet
I have washed my Limbs
I have washed my Elbows
I have washed my neck
[aka@localhost ~]$

The While Loop

There is another loop named as the *while* loop. Let's look at the syntax of the bash while loop.

```
[aka@localhost ~]# while [fill the condition in the brackets]
do
        sample commandx
        sample commandy
        sample commandz
done
```

All the commands in between *do* and *done* are executed repeatedly until the condition stands true.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ x=1
[aka@localhost ~]$ while [$x -1e 10]
> do
> echo "the sky is azure $x times"
> x=$(($x+1))
> done
```

The Case Statement

The *case* statement is dissimilar to a loop. There is no such thing as repetition in the *case* command. You can test simple values such as integers as well as characters. In a *case* command, the bash shell analyzes the condition and accordingly manages the program flow. The case statement at first expands whatever expression you include in it and then it tries to match the same

against the patterns that are included in the script. If it founds a match, all the statements ending up at the double semicolon (;;) get executed by the shell. When it is done, the case stands terminated with the exit status that you had given to the last command. If the *case* command finds no match, the exit status stands at zero.

```
$ cat testfile
#!/bin/bash
# using the case command
case $USER in
sylvia | john)
echo "Welcome, $USER"
echo "Have a nice time";;
testing)
echo "Special testing account";;
adam)
echo "Keep in mind to log off when you have finished the task";;
*)
echo "You cannot enter";;
esac
$ ./testfile
Welcome, sylvia
Have a nice time
$
```

In the above script, we can clearly see that a variable is compared against different sets of patterns. If there is a match between the variable and the pattern, the related command is executed. The catch here is that the *case* command offers us a smoother way of integrating different options for a specific variable. Let’s write the same script using the *if-then* statement.

```
$ cat testfile
#!/bin/bash
# looking for a possible value
if [ $USER = "sylvia" ]
then
echo "Welcome $USER"
echo "Have a nice time"
elif [ $USER = john ]
```

```
then
echo "Welcome $USER"
echo "Have a nice time"
elif [ $USER = testing ]
then
echo "Special testing account"
elif [ $USER = adam ]
then
echo " Keep in mind to log off when you have finished the task "
else
echo "You cannot enter"
fi
$ ./testfile
Welcome sylvia
Please enjoy your visit
$
```

I hope you have understood by now the difference between both scripts. The *case* command just puts all the values in a single list form which the variable checks it for a match. You don't have to write *elif* statements. The repetition is eliminated.

So, what have we learned so far with different scripts? The above commands are also known as structured commands. We can alter the normal program flow in the shell script with the help of these commands. The most basic of these commands is the *if-then* statement. You can evaluate a command and move on to execute other commands on the basis of the result of the evaluated command.

You have the option of connecting different *if-then* statements with the help of *elif* statement. The *elif* is the short of *else if* which means it is another *if-then* statement. The *case* command can be dubbed as the shortest route to achieving the same results as we have with the help of using lengthy *if-then* statements.

Nesting loops

Now this is simple. You can combine different loops into one or you can nest

other loops in an already established one. The good thing is that you can add as many loops as you can for nesting.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #Testing nested loops
[aka@localhost ~]$ for ((b=1; b<=6; b++))
> do
> echo "Starting loop $b:"
> for ((d=1; d<=6; d++))
> do
> echo "Inside loop: $d"
> done
> done
Starting loop 1:
Inside loop: 1
Inside loop: 2
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
Starting loop 2:
Inside loop: 1
Inside loop: 2
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
Starting loop 3:
Inside loop: 1
Inside loop: 2
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
Starting loop 4:
Inside loop: 1
Inside loop: 2
```

```
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
Starting loop 5:
Inside loop: 1
Inside loop: 2
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
Starting loop 6:
Inside loop: 1
Inside loop: 2
Inside loop: 3
Inside loop: 4
Inside loop: 5
Inside loop: 6
[aka@localhost ~]$
```

So you can see the nested loop inside the main loop tends to iterate through its values each time as the outer loop or main loop iterates. Don't get confused between the *dos* and *dones* of the script. Bash shell differentiates the two and refers them to the inner and outer loops. In the above script I blended together two *for* loops. Now let's move on to pair up a *for* and a *while* loop.

```
#!/bin/bash
# filling in a for loop inside a while loop
a=5
while [ $a -ge 0 ]
do
echo "Outer loop: $a"
for (( b = 1; $b ‹ 3; b++ ))
do
c=$[ $a * $b ]
```

```
echo " Inner loop: $a * $b = $c"
done
a=$[ $a - 1 ]
done
$ ./test15
Outer loop: 5
Inner loop: 5 * 1 = 5
Inner loop: 5 * 2 = 10
Outer loop: 4
Inner loop: 4 * 1 = 4
Inner loop: 4 * 2 = 8
Outer loop: 3
Inner loop: 3 * 1 = 3
Inner loop: 3 * 2 = 6
Outer loop: 2
Inner loop: 2 * 1 = 2
Inner loop: 2 * 2 = 4
Outer loop: 1
Inner loop: 1 * 1 = 1
Inner loop: 1 * 2 = 2
Outer loop: 0
Inner loop: 0 * 1 = 0
Inner loop: 0 * 2 = 0
$
```

The break Command

The break command, as is evident from the name, is used to break out of the loop by terminating the same. It is applicable on the *for*, *while* and *until* loop. It can also be dubbed as the escape command.

```
#!/bin/bash
# Time to break out of a loop
for a in 1 2 3 4 5 6 7 8 9 10
do
```

```
if [ $a -eq 8 ]
then
break
fi
echo "Iteration number: $a"
done
echo "The for loop is completed"
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
Iteration number: 6
Iteration number: 7
The for loop is completed
$
```

The loop was broken from the very point I asked it to. Now I'll break out of the inner loop I have written.

```
$ cat testfile
#!/bin/bash
# I am breaking out of an inner loop
for (( x = 1; a ‹ 4; a++ ))
do
echo "Outer loop: $x"
for (( y = 1; y ‹ 100; y++ ))
do
if [ $y -eq 5 ]
then
break
fi
echo " Inner loop: $y"
done
done
$ ./testfile
Outer loop: 1
Inner loop: 1
```

```
Inner loop: 2
Inner loop: 3
Inner loop: 4
Outer loop: 2
Inner loop: 1
Inner loop: 2
Inner loop: 3
Inner loop: 4
Outer loop: 3
Inner loop: 1
Inner loop: 2
Inner loop: 3
Inner loop: 4
$
```

Let’s see how to break out of an outer loop.

```
$ cat testfile
#!/bin/bash
# How to break out of an outer loop
for (( e = 1; e ‹ 4; e++ ))
do
echo "Outer loop: $a"
for (( f = 1; f ‹ 100; f++ ))
do
if [ $f -gt 4 ]
then
break 2
fi
echo " Inner loop: $f"
done
done
$ ./testfile
Outer loop: 1
Inner loop: 1
Inner loop: 2
```

```
Inner loop: 3
Inner loop: 4
$
```

The Continue Command

This command is somewhat related to the break command, as it also terminates the processing of the loop, but differs from the break command because it doesn't exit the loop. You can set your own conditions in the script to direct the loop to stop where you want it to be.

```
$ cat testfile
#!/bin/bash
# testing the continue command
for (( x = 1; x ‹ 15; x++ ))
do
if [ $x -gt 5 ] && [ $x -lt 10 ]
then
continue
fi
echo "The Numeric Digit: $x"
done
$ ./testfile
The Numeric Digit: 1
The Numeric Digit: 2
The Numeric Digit: 3
The Numeric Digit: 4
The Numeric Digit: 5
The Numeric Digit: 10
The Numeric Digit: 11
The Numeric Digit: 12
The Numeric Digit: 13
The Numeric Digit: 14
$
```

Processing the Output of a Loop

The shell gives you an opportunity to redirect the output of your loop to a particular file. You have to fill in the shell window with the script you have

in mind. You have to create a file when the loop ends. Your output from the loop script is secured inside a text file that you create. It will look easier when you will practice it. Let’s jump right into the shell window.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #redirecting the output of the loop toward a file
[aka@localhost ~]$ for ((x=1 ; x<10;x++))
> do
> echo "the numeric digit is $x"
> done > azuresky.txt
[aka@localhost ~]$ echo "the command is finished."
the command is finished.
[aka@localhost ~]$ cat azuresky.txt
the numeric digit is 1
the numeric digit is 2
the numeric digit is 3
the numeric digit is 4
the numeric digit is 5
the numeric digit is 6
the numeric digit is 7
the numeric digit is 8
the numeric digit is 9
[aka@localhost ~]$
```

A similar technique can be used to pipe the output of your loop script to the command you want to. Let’s experiment with this.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #Piping out the output
[aka@localhost ~]$ for country in "The Netherlands" Australia Pakistan Spair
> do
> echo "This summer I'll visit $country"
> done | sort
This summer I'll visit Australia
This summer I'll visit Pakistan
This summer I'll visit Spain
This summer I'll visit The Netherlands
This summer I'll visit The United States of America
```

```
[aka@localhost ~]$
```

Chapter 7: Moving On To The Advanced Level In Shell Scripting

When you get used to writing shell scripts, you will be able to use your own scripts somewhere else to execute a program. A small code can be integrated into another script to get the desired result. Writing large scripts can be an exhaustive exercise to do that's why the shell offers programmers a convenient way to do script writing. There are user-defined functions that can make script writing easy and fun.

The Shell Functions

Shell functions save you from repeat writing the same code for different tasks. Displaying messages and doing mathematical calculations can be tedious for you when you have to do that over and over again.

With shell functions, you can write once and then use the block of code over and over again.

How to Create Function

You can use two formats to create functions in bash shell scripts. The first format can be dissected into the keyword 'function' and the name that you assign to the block of the code. See the syntax as under:

```
function name that you will assign to the block code {
        commands
}
```

The name means the unique name you assign to your particular function. The commands can be a single or multiple bash command. You can add many or as little command as you desire. Just call the function and it will execute the commands in the order you in which you place them in the script. The function command doesn't add to or cut from the performance of the script. It just flows in the normal order.

How to Use the Function Script

When you are about to use the function command, don't forget to assign it a name. Let's see an example.

```
$ cat testfile
#!/bin/bash
# Let’s see how to use a function in a script
function functest1 {
echo "Let’s see how function works"
}
count=1
while [ $count -le 5 ]
do
functest1
count=$[ $count + 1 ]
done
echo "The loop ends here"
functest1
echo "this makes the end of this script"
$ ./testfile
Let’s see how function works
Let’s see how function works
Let’s see how function works
Let’s see how function works
Let’s see how function works
The loop ends here
Let’s see how function works
this makes the end of this script
$
```

You can see that each time you refer back to the functest1, the function that you have named, the bash shell gets back to the same to execute the commands you had left in there. That means you are saved from the hassle of repeating the script in the command line. Just remember the unique name that you assign to the function. I hope you have understood by now why the name is so important.

You don’t have to write the shell function in the start. Instead, you can also write in in the middle of the script. But you have to define the function before using it, otherwise you will get an error message. Let’s see an example.

```
$ cat testfile
#!/bin/bash
# put the shell function in the middle of the script
count=1
echo "This line is placed before the function definition"
function functest1 {
echo "Now this is just a function"
}
while [ $count -le 5 ]
do
functest2
count=$[ $count + 1 ]
done
echo "The loop has reached its end"
func2
echo "This heralds the end of the script"
function func2 {
echo "Now this is just a function example"
}
$ ./testfile
This line comes before the function definition
Now this is just a function example
Now this is just a function example
Now this is just a function example
Now this is just a function example
Now this is just a function example
The loop has reached its end
./testfile: functest2: command not found
This heralds the end of the script
$
```

I defined the functest1 later in the script and when I used it, it ran okay, but I didn’t define functest2 before using it, so it results in an error message. A thing which you should be careful about is the name of the functions. You should not assign the same name to different functions. Each function must

have a unique name, otherwise, the shell will not be able to identify them separately, and will override the previous definition with the new one. Remember, there won't be any error messages to alert you about the mistake you are committing.

```
$ cat testfile
#!/bin/bash
# using the same name for different functions
function functest1 {
echo "I am defining the function with a unique name"
}
functest1
function functest1 {
echo "I have repeated the function name assigning it to another function"
}
functest1
echo "The script ends here"
$ ./testfile
I am defining the function with a unique name
I have repeated the function name assigning it to another function
The script ends here
$
```

How to return a value using shell functions

If we talk about the default exit status, it is the exit status which is returned by the last command of the function.

```
$ cat testfile
#!/bin/bash
# learning about the exit status of a shell function
functest1() {
echo "I am attempting to display a file which is non-existent"
ls -l badfile
}
echo "it is time to test the function:"
functest1
```

```
echo "The exit status is: $?"
$ ./testfile
it is time to test the function:
I am attempting to display a file which is non-existent
ls: badfile: No such file or directory
The exit status is: 1
$
```

The exit status turns out to be 1 since the last command has failed.

```
$ cat testfile
#!/bin/bash
# time to test the exit status of a shell function
func1() {
ls -l badfile
echo "We should now test a bad command"
}
echo "shall we test the function now:"
func1
echo "The exit status is: $?"
$ ./test4b
shall we test the function now:
ls: badfile: No such file or directory
We should now test a bad command
The exit status is: 0
$
```

How to use the function output

We have seen how to capture and process the output of a shell variable. Now I'll explain how you can capture a function's output.

```
$ cat testfile
#!/bin/bash
# I will use the echo command to return a value
function dbl {
read -p "Enter a value: " value
```

```
echo $[ $value * 5 ]
}
result=`dbl`
echo "The new value is $result"
$ ./testfile
Enter a value: 300
The new value is 1500
$ ./testfile
Enter a value: 500
The new value is 2500
$
```

The echo will display the result of the mathematical calculation. The script gets the value of dbl function instead of locating the exit status as the final answer. So, we have redirected the shell to capture the output of the function.

How to Pass Parameters to a Shell Function

Functions can use the standard parameter environment variables for the representation of any parameters which are passed on to the function on the CLI. $0 variable is used to represent the definition of the function. Other parameters are $1, $2,$3 and so on are used to define the parameters on the command line. There also is a special variable $# in order to determine the total number of parameters on the command line. The parameters should be written on the same command line on which you are writing the function.

```
$ cat testfile
#!/bin/bash
# how to pass parameters to a function
function addem {
if [ $# -eq 0 ] || [ $# -gt 2 ]
then
echo -1
elif [ $# -eq 1 ]
then
echo $[ $1 + $1 ]
else
```

```
echo $[ $1 + $2 ]
fi
}
echo -n "Adding 20 and 30: "
value=`addem 20 30`
echo $value
echo -n "Shall we try to add only one number: "
value=`addem 20`
echo $value
echo -n "This time try to add no numbers: "
value=`addem`
echo $value
echo -n "Let’s add three numbers this time: "
value=`addem 20 30 40`
echo $value
$ ./test6
Adding 20 and 30: 50
Let’s try adding just one number: 40
Now trying adding no numbers: -1
Finally, try adding three numbers: -1
$
```

The shell acted as it was told. Where there were more than two parameters, it returns the value of -1. Where there was one parameter, it added the figure to itself. Where there were two parameters, it added them together to get the result.

How to use global variables in a shell function

The variables have a versatile use which makes them often confusing to learn. You can also use them in the shell functions. They have a different role here. You can use the following variables in the shell functions.

- **Global**
- **Local**

Global variables are valid within the shell script. Even if you define its value in the main script, you can retrieve its value in the function.

```
$ cat testfile
```

```
#!/bin/bash
# how to use a global variable to pass a value
function dbl {
vle=$[ $vle * 2 ]
}
read -p "Enter a vle: " vle
dbl
echo "The new value is: $vle"
$ ./testfile
Enter a vle: 300
The new vle is: 600
$
```

Don’t get confused I have used vle instead of value. The variable $vle is defined here in the main script but is still valid inside the function.

```
$ cat testfile
#!/bin/bash
# let’s see how things can go real bad by the wrong use of variables
function functest1 {
temp=$[ $value + 5 ]
result=$[ $temp * 2 ]
}
temp=4
value=6
functest1
echo “We have the result as $result”
if [ $temp -gt $value ]
then
echo “temp is larger”
else
echo “temp is smaller”
fi
$ ./badtest2
The result is 22
temp is larger
```

$
Local variables are also used in functions. Local variables are mostly used internally. Put the keyword *local* before the variable declaration. You can make use of the local variable while you are assigning a value to the variable. With the help of the keyword, it becomes easier for the shell to identify the local variable inside the function script. If any variable of the same name appears outside of the function script, the shell considers it of separate value.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #I will attempt to an array variable
[aka@localhost ~]$ function functest1 { echo "The parameters are: $@"; thisa
}
[aka@localhost ~]# myarray=(1 2 3 4 5 6 7 8 9)
[aka@localhost ~]# ehco "the original array is ${myarray[*]}”
functest1 $myarray
$ ./testfile
The original array is: 1 2 3 4 5 6 7 8 9
The parameters are: 1
./testfile: thisarray[*]: bad array subscript
The received array is
$
```

You must allot the array variable its individual values and then use those values as function parameters.

```
[aka@localhost ~]$ #!/bin/bash
[aka@localhost ~]$ #I will attempt to an array variable
[aka@localhost ~]$ function functest1 {
local newarray
newarray=(‘echo “$@”’)
echo "The value for the array is: ${newarray[*]}"
}
[aka@localhost ~]# myarray=(1 2 3 4 5 6 7 8 9)
[aka@localhost ~]# ehco "the original array is ${myarray[*]}”
functest1 ${myarray[*]}
$ ./testfile
```

```
The original array is: 1 2 3 4 5 6 7 8 9
The new array is: 1 2 3 4 5 6 7 8 9
$
```

How to create a library using functions

This can be really handy if you are an administrative assistant at an office. You will save plenty of time that would otherwise have been spent on typing repeated scripts. You can create a library file to use as many times as you deem fit.

```
$ cat libfile
# creating a library file using shell functions
function addem {
echo $[ $2 + $3 ]
}
function multem {
echo $[ $5 * $2 ]
}
function divem {
if [ $2 -ne 0 ]
then
echo $[ $1 / $2 ]
else
echo -1
fi
}
$
```

You can now fill in the library file name in the script and get the desired output.

```
$ cat testfile
#!/bin/bash
# how to use functions you have defined in the library file
. ./libfile
val1=20
val2=10
```

```
result1=`addem $val1 $val2`
result2=`multem $val1 $val2`
result3=`divem $val1 $val2`
echo "The result of adding them is: $result1"
echo "The result of multiplying them is: $result2"
echo "The result of dividing them is: $result3"
$ ./testfile
The result of adding them is: 30
The result of multiplying them is: 200
The result of dividing them is: 2
$
```

Use the Functions on the Command Line

Let’s learn using the functions on the command line.

```
$ function divem { echo $[ $1 / $2 ]; }
$ divem 200 10
20
$
```

```
$ function doubleit { read -p "Enter val: " val; echo $[
$ val * 5 ]; }
$ doubleit
Enter value: 50
225
$
```

```
$ function multem {
> echo $[ $1 * $2 ]
> }
$ multem 100 100
10000
$
```

Shell functions are a great way to place script code in a single place so that you can it repeatedly whenever needed. It eliminates the rewriting practice. If you have to use a lot of functions in order to deal with some heavy workload, you have the option to create function libraries.

How to create text menus in the shell

It is time to make shell scripting more interesting by making scripting interactive. This is pure programming. You can offer your customers an interactive menu display to choose from if you are tired of dealing with them all day at the office. Linux can make this fun.

Obviously, you need to have the layout first for then menu. You can add what you want to be included in the menu. Before creating the menu, it is a good idea to run the clear command. After that, you can enter the echo command to display different elements of your menu.

You can add newline characters and the tab with the help of -e command. The command line by default displays only printable characters.

```
clear
echo
echo -e "\t\t\tWelcome to the Admin Menu\n"
echo -e "\t1. Her will be displayed the disk space"
echo -e "\t2. Here will be displayed the logged on users"
echo -e "\t3. Here will be displayed the memory usage"
echo -e "\t0. Exit\n\n"
echo -en "\t\tEnter your option: "
```

With the -en option in the last line, there will be no newline character in the end of the display. This will allow the users to enter their input. You can retrieve and read the input left by the customer with the help of the following command.

```
read -n 1 option
```

You can assign functions to the menu. These functions are pretty fun to do. The key is to create separate functions for each item in your menu. In order to save yourself the hassle, create stub functions so that you know what you have to put in there while you work smoothly in the shell. A full function will

not be kept from running in the shell, which will interrupt your working. Working would be smoother if you put the entire menu in a function script.

```
function menu {
clear
echo
echo -e "\t\t\tWelcome to the Admin Menu\n"
echo -e "\t1. Her will be displayed the disk space"
echo -e "\t2. Here will be displayed the logged on users"
echo -e "\t3. Here will be displayed the memory usage"
echo -e "\t0. Exit\n\n"
echo -en "\t\tEnter your option: "
read -n 1 option
}
```

As I have already discussed, this will help you to view the menu anytime by just recalling the function command. Now you need the case command to integrate the layout and the function to make the menu work in real time.

```
$ cat menu1
#!/bin/bash
# simple script menu
function diskspace {
clear
df -k
}
function whoseon {
clear
who
}
function memusage {
clear
cat /proc/meminfo
}
function themenu {
clear
echo
```

```
echo -e "\t\t\tWelcome to the Admin Menu\n"
echo -e "\t1. Her will be displayed the disk space"
echo -e "\t2. Here will be displayed the logged on users"
echo -e "\t3. Here will be displayed the memory usage"
echo -e "\t0. Exit\n\n"
echo -en "\t\tEnter your option: "
read -n 1 option
}
while [ 1 ]
do
menu
case $option in
0)
break ;;
1)
diskspace ;;
2)
whoseon ;;
3)
memusage ;;
*)
clear
echo "Sorry, you went for the wrong selection";;
esac
echo -en "\n\n\t\t\tHit any key on the keyboard to continue"
read -n 1 line
done
clear
$
```

Conclusion

Sylvia is an avid learner by now. She admits that what she had hated all her life has become her lifeline. The ease of use, the speed, the power over her computer, and the fun that Linux gave her was matchless. She has now installed Linux on her personal computer at home. Sylvia is now on her way to becoming a Linux pro. I wonder if she would soon be able to advise me on the command line because of the way she practices the shell. A trick she would never tell anyone is that she kept a diary on which she wrote all the important commands in order to easily invoke them when she needed them. Eventually, it helped her memorize the commands.

Linux is an operating system just like you have Windows or Mac OS X. Some say that it floats between the software and the hardware form of the operating system. Linux is considered better than the Windows operating system by programmers and white hat hackers. On the other hand, ordinary folks who have to deal with routine office work or play games prefer the Windows operating system. Linux is definitely better than Windows and there are reasons behind the notion. Let’s analyze them so that you have a clearer mind with respect to different operating systems when you finish reading this book.

It is pertinent to cite an example. What if you buy a high-end phone but are unable to see what is inside it and how does it operate? Windows operating system is just like that phone. You can use it, enjoy it, but you cannot see how it works, how it is powered, and how it is wired. On the contrary, Linux is open-source. You can get into its source code any time you like.

The world is moving very fast when it comes to technology. While there are good people across the world who are adding positive things to the cyber world, such as operating systems and applications that can really help you deal with day to day activities, there is no shortage of the bad ones who are consistently trying to sneak into your personal computer to rob you of any important piece of information that could help them get some each money. It

is now your foremost priority to protect yourself from the people who harbor such heinous designs. The basic thing to have that protection is to get the operating system that is secure against any such attacks.

Windows OS is extremely vulnerable to hacking attacks. I cannot claim that Linux is absolutely impenetrable but still it is much better than the Windows. Its features such as the package management and repositories make it securer than Windows. If you have Windows OS installed on your personal computer, you consistently run the risk of catching virus on your system that's why the first thing you do after installing the Windows OS is to purchase an antivirus program in order to keep your computer well-protected. However, with Linux on your personal computer, this need is eliminated.

Windows OS is very sensitive about the specifications of the system on which it is to be installed. For example, to get Windows 10 installed on your system, you need to update your RAM and HDD capacity. My friend had a laptop with Windows 7 installed on it. Suddenly, his system started showing the message that Microsoft Windows had stopped supporting Windows 7 and that's he must install Windows 10 on the computer. The message also read a caution that it was better to purchase a new computer on which Windows 10 was already installed by the company. That was under the heading of recommended action. As most people would have done, he bought a new computer and sold the old one.

This is absolutely not the case with Linux. All you need is to meet up the minimum system requirements and start the operating system without any fear of its expiration. Linux has the power to revive old computer systems. If you have a system with 256 MB RAM and an old processor, that is enough to run Linux. Now compare that with Windows 10 which demands 8GB RAM for smooth functioning. If you give the same system specifications to the Linux operating system, it would surely give you an edge over the Windows OS.

In addition, Linux is best for programmers. It supports all programming languages such as Java, Python, Ruby, Perl and C++. Additionally, there is a wide range of apps that suit programmers. Especially the package manager

on Linux is what the programmer needs to get things done smoothly.

Linux offers a wide range of software updates, which are faster to install, unlike Windows which restarts multiple times just to install latest updates. Apart from the all benefits, the greatest of all is the option of customization. Whatever you get in the Windows is the ultimate truth. You cannot change it for your ease of use. With Linux things change dramatically. You can customize different features. You can add or delete different features of the operating system at will. Keep what you need and delete what you don't like because it is an open source system. Also, you can add or dispose of various wallpapers and icon theme on your system in order to customize its look.
Perhaps the greatest benefits with Linux is that you can get it for free online. Unlike the Windows OS you don't have to purchase it.

This book helps users in learning the basics of Linux command line interface, which makes Linux different from all the other operating systems. Now that you have made it till the end, I hope that you have learned what makes Linux fun to use. I hope that the dread of the command line interface, the dark screen I talked about in the introduction, has vanished into thin air.

It is not that boring. You can do more work with Linux in a short timeframe which makes it fun to use. Linux provides you flexibility with respect to components you need to install. You can what you require and leave what you don't. For example, you can do away with unnecessary programs like paint and boring calculators in Windows. Instead, you can take any important and relevant program from the open source, write it on the command line and run it on the system. You can add and delete programs and applications as many as you like.

If the Windows OS catches a malware, it corrupts leaving you at the mercy of luck. If the system survives the attack, you get your data back. If it doesn't, you can only mourn your loss and do nothing. You don't have any backup for the Windows OS in any other partition; a backup that could keep forming an image of what you are doing on the operating system to save it for the day you get the OS corrupted by an attack. But Linux offers you the perfect solution. You can keep Linux file in multiple partitions. If one of them

corrupts, you can access your data from the other partitions. That’s simple and handy.

I hope this book has provided you with the basic knowledge you need to move further on the ladder in the world of Linux. I cannot promise that a single read of this book will make you an expert on Linux, but it will definitely equip you with the knowledge base needed to become a pro in the Linux operating system.

Resources

https://www.linux.com/what-is-linux/
https://blog.learningtree.com/how-to-practice-linux-skills-for-free/
https://www.binarytides.com/linux-command-check-memory-usage/
https://www.digitalocean.com/community/tutorials/basic-linux-navigation-and-file-management
https://www.javatpoint.com/linux-error-redirection
https://www.geeksforgeeks.org/basic-shell-commands-in-linux/
https://www.guru99.com/hacking-linux-systems.html
https://www.tecmint.com/how-to-hack-your-own-linux-system/
https://opensource.com/business/16/6/managing-passwords-security-linux
https://www.computerhope.com/jargon/i/init.htm
https://www.thegeekstuff.com/2011/02/linux-boot-process/
https://www.linuxtechi.com/10-tips-dmesg-command-linux-geeks/
https://www.techopedia.com/definition/3324/boot-loader
https://www.dedoimedo.com/computers/grub.html#mozTocId616834
https://www.tecmint.com/understand-linux-shell-and-basic-shell-scripting-language-tips/
https://medium.com/quick-code/top-tutorials-to-learn-shell-scripting-on-linux-platform-c250f375e0e5
https://bash.cyberciti.biz/guide/Quoting
https://askubuntu.com/questions/591787/how-can-display-a-message-on-terminal-when-open-it
https://www.learnshell.org/en/Shell_Functions
https://linuxacademy.com/blog/linux/conditions-in-bash-scripting-if-statements/
https://www.cyberciti.biz/faq/bash-while-loop/
https://www.geeksforgeeks.org/break-command-in-linux-with-examples/
https://likegeeks.com/bash-scripting-step-step-part2/#Nested-Loops
https://www.javatpoint.com/linux-init
https://www.howtogeek.com/119340/htg-explains-what-runlevels-are-on-linux-and-how-to-use-them/
Linux Command Line and Shell Scripting Bible by Richard Blum

THE LINUX COMMAND LINE by William E . Shotts , J r .

How Linux Works 2nd Edition What Every Superuser Should Know by Brian Ward