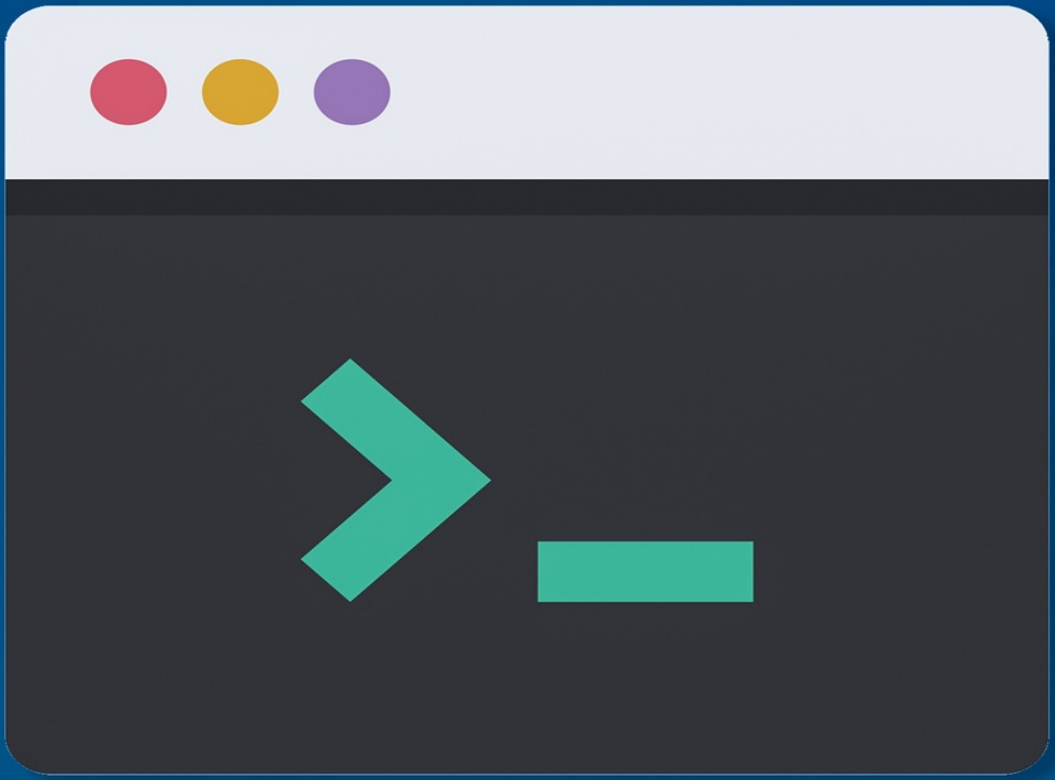
LINUX COMMAND LINE

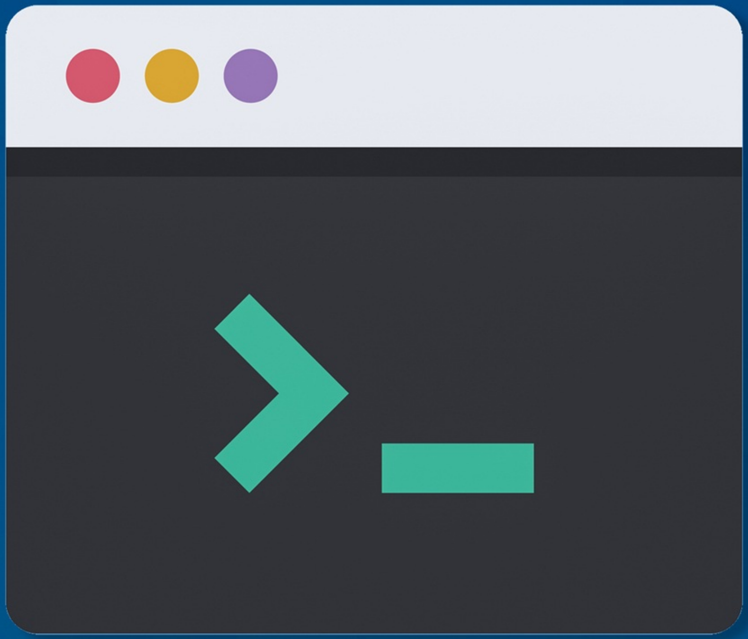
ADVANCED GUIDE TO UNDERSTAND THE BASICS OF COMMAND LINE, ADMINISTRATION, AND SECURITY FOR HACKERS. START YOUR QUICK STUDY FOR HACKING AND NETWORKING. INCLUDING THE ESSENTIALS, TIPS, AND EXERCISES



JASON BLUNT

LINUX COMMAND LINE

ADVANCED GUIDE TO UNDERSTAND THE BASICS OF COMMAND LINE, ADMINISTRATION, AND SECURITY FOR HACKERS. START YOUR QUICK STUDY FOR HACKING AND NETWORKING. INCLUDING THE ESSENTIALS, TIPS, AND EXERCISES



JASON BLUNT

LINUX COMMAND LINE:

ADVANCED GUIDE TO UNDERSTAND THE BASICS OF COMMAND LINE, ADMINISTRATION AND SECURITY FOR HACKERS. START YOUR QUICK STUDY FOR HACKING AND NETWORKING. INCLUDING THE ESSENTIALS, TIPS AND EXERCISES.

Chapter 1 Contents

Conclusion

Description

Linux is a mammoth operating system that derives its development from the millions of developers working on improving the open software. Because we cannot touch on everything, we shall start with some of the basic commands and work toward explaining the hard stuff. By getting into these basics, I assume that you have a Linux operating system. To log into the Linux system, you will require a username and password that you will use to log into the graphical user interface.

The Linux system has two main modes of operation i.e. the sober-quick in text console mode which looks like a DOS operating system with a multi-tasking, multi-user and system that can be used with a mouse. The other mode is the graphical mode, which uses more system resources but looks and feels much better. Nowadays, the graphical mode is the most common in home computers. To know if you are logging into the graphic interface, you will be required to provide a username and password in two windows .

The command line in Linux is one of the most important features because it allows you to do complex things in very simple ways so naturally you find a permanent command line window open for Linux users, although the command line is an interpretive programming language, it can be used by non-experts and programmers without any difficulty.

This guide will focus on the following:

- What Is Linux?
- The Linux Command Line
- Benefits of the Linux operating system
- How to Install Linux
- Package Management
- Linux Distributions
- File Operations
- Exploring Shell
- Archiving and Compression
- … AND MORE!!!

Introduction

Did you know that due to Android's dominance on smartphones, tablets, and other mobile devices, Linux has turned into one of the largest installed bases in all general-purpose systems?

Although it is not the top pick for desktop computers, Linux remains as an exemplary model of free and open source collaborative projects. Since it operates on embedded systems, it is very useful on television, facility automation controls, video game consoles, network routers, smartwatches, and, as mentioned, mobile devices.

Linux's simplistic design makes it a favorite of programmers. While others like a bit of complexity, some programmers prefer straightforward concepts. In its original nature, it is a leading OS (or Operating System) on mainframe computers, supercomputers, and numerous servers.

As a beginner in Linux, start learning the fundamentals. What are the essentials about Linux?

History of Linu x

In 1991, Linux was officially released. Originally, its development centered as a free OS for different Intel x8-based personal computers. Its creator, Linus Torvalds, announced that the OS' creation is partly due to a micro server kernel's unavailability back then.

As an open source collaborative project, its source code is free to use. Under its respective license's term, it can also be modified for commercial and non-commercial distribution.

Since it is compliant with POSIX (or Portable Operating System Interface), Linux is a dependable OS. It has undergone and passed assessment for API (or Application Programming Interface), standard utility interfaces, and command line shells.

Linux Components

The seamless activities of Linux are attributable to its essential components. There are seven of these.

Seven essential components :

Applications availability

Linux presents the availability of thousands of applications, and these applications are available for immediate installation. It is like *Windows Store* and *Apps Store* that lets you search for a preferred application. Once done searching, you can install the app from a centralized location.

Daemons

Daemons are Linux components that serve as background services. Examples of background services are *sound, printing, and scheduling* . These are launched either after a desktop login or during boot.

Desktop environments

Desktop environments refer to components with user interaction. Examples are *GNOME, Unity,* and *Cinnamon Enlightenment* . Each of these comes with configuration tools, file managers, calculators, web browsers, and other built-in features .

Graphical server

A graphical server is Linux's subsystem. Its primary duty is displaying graphics on your screen. You can also refer to it as "X" or the "X server".

The boot loader

Linux's management of your computer's boot process is handled by the boot loader. Usually, it is in the form of a splash screen. Once this splash screen pops up, it will slowly proceed into the booting process.

The kernel

Linux's core is called the kernel. It oversees management for the memory, peripheral devices, and CPU.

The shell

The shell is Linux's command line. It permits control via typed commands in a text interface.

The Concatenate: A Frequently Used Linux Feature

One of the reasons why a Linux OS functions smoothly is due to simple features. These features can handle several tasks simultaneously. One of these features is *concatenate* or cat.

Concatenate feature:

- */etc/pass – for displaying file contents*

Sample:

```
# cat /etc/password
```

```
root:x:0:2:root:/root:/bin
bin:x:1:1:bin:/root:/bin:/sbin/nologin
nard:x:300:400::/home/nard:/bin/bash
```

- *test test1 – for viewing contents of multiple files*

Sample:

```
# cat first test second test
Hello everybody
Hi world!
```

- *-n option – for displaying the contents of a file; uses line numbers*

Sample:

```
# cat -n song titles.txt
1  Pop
2  Snow in Summer
3  Your Black Heart
4  And I Know It Is A Home
5  And These Places Could
6  Be Much Bigger
7  Brighter Than Tomorrow
8  And Please If You Really Really Try
9  You Will Find There Is No Need
10  To Cry During This Moment
11  In This Place You Will Feel
12  There Is No Hurt Or Sorrow
```

- *-e option – for displaying multiple lines in a single line*

Sample :

```
# cat -e test
Hi there everybody, how do you do?$
Hey, am alright.$
How was your training and where was it?$
$Let us do maybe practice some more in Linux.
```

Result:

```
Hi there everybody, how do you do?
Hey, am alright. How was your training and where was it? Let us do
maybe practice some more in Linux.
```

- *-T – for displaying where "tab" should be used and a new line should begin; "^|" is an indication for the use of "tab"*

Sample:

```
# cat -T test
Hi there ^Ieverybody, how do you do?
Hey, ^Iam alright .
^I^IHow was your training ^Iand where was it?
Let us do ^Imaybe practice some more in Linux.
```

Result:

```
Hi there
everybody, how do you do?
Hey,
am alright.
How was your training
and where was it?
Let us do
maybe practice some more in Linux.
```

- *test – for displaying contents of multiple files in a single structure*

Sample:

```
# cat first test; cat second test; cat third test
```

```
This is first test file

This is second test file.

This is third file .
```

Ten Basic Commands

Before, Linux’s archaic structure, as well as its command line, is a turn-off. According to some programmers, it was rather too complex. However, with all the developments, Linux is now an OS that adapts to changes of the modern world.

Before the identifier, type “#” to use a command.

Ten commands:

1. mkdir – allows creation of directories

Sample:

```
# mkdir nameofdirectory
```

2. rm – allows file removal without a confirmation prompt

Sample:

```
# rm nameoffile
```

3. w – displays information on current users, as well as current users’ average load

Sample :

```
# w

05:33:22 uptime 33 min, 5 users, load average: 0.01, 0.00, 1.07

USERS     TTY      FROM     LOGIN   NOT IDLE   JCPU   PCPU

trs  pts/0    198.172.51.1     04:21    0.03s  0.18s  0.07s w
```

4. uptime – displays information on how long system is running, number of current users, and load average

Sample:

```
# uptime

05:33:23 uptime 33 min, 5 users, load average: 0.01, 0.00, 1.07

[trs~] uptime

procps version 2.8
```

5. Is – displays list of files in readable format; displays files since last modification

Sample:

```
# ls -l

total 120

dr-xr-x-x.   4 root root 1094 Jun 24 07:11 bin

dr-xr-x-x.   7 root root 2098 Jun 24 10:31 boot

total 39

-rw-r--r--. 2 root root 2062 Jun 24 13:11

install.log.sys

-r-rw--r--. 2 root root 2063 Jun 24 14:47 install.log

-rw-------. 2 root root 2041 Jun 24 14:47 anaconda-ks.cfg
```

6. who – displays the date, host information, time, and date

Sample:

```
# who

trs  pts/0        2016-07-11 09:21 (192.172.51.1)
```

7. less – allows the quick viewing of files; allows page up and page down options

Sample:

```
# less install.log.sys

Installing setup-3.7.18-09.e19.noarch

warning: setup-3.7.18-09.e19.noarch: Header V4 RSA/SHA198

Signature, key ID b109khz: KEY

Installing ca-certificates-2009.72-8.el9.noarch

Installing filesystem-3.7.18-09.e19.noarch.i727

Installing xml-common-0.6.3-32.el6.noarch

Installing tzdata-2009l-1.el9.noarch

Installing iso-codes-3.19-3.el9.noarch
```

8. more – allows quick viewing of files; displays percentages

Sample:

```
# more install.log.sys

Installing setup-3.7.18-09.e19.noarch

warning: setup-3.7.18-09.e19.noarch: Header V4 RSA/SHA198
```

```
Signature, key ID b109khz: KEY

Installing ca-certificates-2009.72-8.el9.noarch

Installing filesystem-3.7.18-09.e19.noarch.i727

Installing xml-common-0.6.3-32.el6.noarch

Installing tzdata-2009l-1.el9.noarch

Installing iso-codes-3.19-3.el9.noarch

--More--(20%)
```

9. top – displays real-time kernel managed tasks and processor activity; displays use of memory and processor

Sample:

```
# top trs

top - 12:22:03 up  2:20,  1 user,  load average: 0.00, 0.01, 0.01

Tasks: 120 total, 2 running, 120 sleeping, 1 stopped, 1 zombie

Cpu(s):   0.2%us,  0.4%sy,  0.4%ni,  88%id,  0.1%wa,  0.1%hi,  0.1%si,
0.1%st

Mem:  1040502k total, 90025k used, 950477k free, 10048k buffers

Swap: 48444k total, 0k used, 48444k free, 70022k cached

PID USER   PR USER NI VIRT RS SHR W %CPU %MEM TIME+

1812 trs 19 2 22301 947 912 W  0.01  0.1   0:01:30 sshd
```

```
1803 trs 19 2 22301 947 912 W  0.01  0.1   0:01:22 bash

7719 trs 19 2 22301 947 912 W  0.01  0.1   0:00:58 sshd

8811 trs 19 2 22301 947 912 W  0.01  0.1 0:00:40 bash
```

10. last – displays user activity details such as time, date, system boot, terminal, and kernel version

Sample:

```
# last

trs  pts/1        198.172.51.1     Fri Jun 24 05:35   still logged in

trs  pts/0        198.172.51.1     Fri Jun 24 04:30   still logged in

reboot   system boot  3.7.18-09.e19.i Fri Jun 24 04:28 - 11:38  (03:43)

root     pts/1        198.172.51.1     Wed Jun 22 9:36 - down   (03:53)

root     pts/0        :0.0             Wed Jun 22 9:32 - 13:09  (02:32)

root     tty1         :0               Wed Jun 22 9:03 - down   (04:26)

reboot      system  boot    3.7.18-09.e19.i  Wed  Jun  22  08:53  -  14:33
(04:35)

narad    pts/2        198.172.51.1     Tue Jun 21 06:08 - down   (01:15)
```

Chapter 1 What Is Linux?

To put it simply, Linux is an operating system. An operating system is simply the software in any piece of technology that helps manage every single hardware resource that's associated with the product. So, if you have a smartphone and you boot it up, you have all those preloaded applications you may or may not use. Those apps are called software. That software is loaded into the operating system, and it's the operating system that houses and manages the information inputted into that phone application.

If we were speaking in metaphors, the wiring of your home is its "operating system." It houses you and all the stuff you use, and it manages your electricity and water. The wiring of your home takes care of all those basic functions you rely on, and when you want your home upgraded, you must upgrade the wiring as well.

An operating system works in much the same way.

Now, an operating system has many different pieces: the bootloader is the software that takes care of the booting process of your piece of technology. When you turn your phone or laptop on, there is usually a screen that flashes on it with the logo of the piece of technology you own, and then it goes away before it loads your home or lock screen.

That's the forefront of the bootloader.

Then you have the kernel, and this is the piece in the Linux operating system that's called "Linux." The kernel is the stabilizing core of the entire system, and it helps to manage the piece of technology's memory, peripheral devices (e.g., the keyboard, mouse, or plugged-in speakers for a computer), and the CPU (the "brains" of your technological device). At its core, this is what keeps your piece of technology breathing.

It's the heart of your device.

The daemons are the next part, and these are basically all the background services that queue themselves up whenever you turn on your device. Things like the sound on your device, its printing capabilities, and any calendar events you might have programmed into it are all examples of daemons .

Then you have the shell. When you are coding on a Linux system, the shell is the command process that allows the person typing in the commands to control the computer (and its operating system) by the lines of code you type into the text interface. With the modern updates of Linux, it's no longer necessary to fiddle around with certain command lines, but if you have ever watched those television shows where a computer whiz is typing seemingly random pieces of text information into a black box on their computer screen at lightning speed, that's what we're talking about. Those individual "random pieces" are lines of code, and they control specific functions within the operating system itself.

That's what the shell consists of: those command lines of code and the "little black box" they are being typed into.

An operating system also has a graphical server. This is a fancy way of saying "the process that helps pictures display on your screen." If you ever hear someone talk about the "X server," this is what they are talking about: the process that helps images pop up on the main screen of your device.

Then you have the desktop environment, which is what people interact with. When you turn on your device, get past the bootloader screen, and log into your piece of technology, the "main screen" pops up. That's the desktop environment.

And finally, there are the applications. All those things you download from "app stores" or download from a website onto your desktop computer are all applications. They are what give your piece of technology different capabilities, from playing a game to recording the videos you watch. Each type of Linux desktop environment has its own unique and centralized application store where you can download certain things in order to customize your Linux experience.

So, why use Linux? Why not use some other operating system to learn how to do this whole coding and DIY project thing? After all, Linux is one of the most different computing environments compared to anything else that's popular on the market, and even for the most technologically-savvy person, there is always an element they have to learn from scratch.

Well, for one thing, Linux is free. All those popular operating systems, such as iOS (for Apple products) and Windows (for Microsoft products), cost money to download onto a new piece of software. Then, as they put out new and fully upgraded software updates, sometimes *those* cost money, too! Linux is completely free: there is no "yearly clean-up" fee, and there is no fee to download and update new upgrades and desktop environment layouts. You'll also not pay anything for server licensing (the monthly fee that's sometimes charged to run licensed software on a computer, such as Microsoft Word).

With the cost effectiveness of this operating system in mind, Linux is also incredibly secure. People boast of using this operating system for decades without ever running into issues with viruses, malware, or computer slowdowns as applications begin to upgrade. *That's* how stable Linux is, and you won't find that with any other operating system.

Not only that, but the only server reboot you’ll ever experience is if the kernel is updated. Remember what we said the kernel was?

It’s the heart of your technological device.

This means no randomly shutting down your computer device to reboot and no having to restart your technological device in order to upgrade it. Ever.

Linux is also distributed underneath something called an open source license. This means that the operating system can run programs for any purpose, can be studied under any type of microscope, can be altered and changed in any fashion so you can do what you wish, can be redistributed without payment to help out your neighbor, and can be distributed after being modified by you to anyone you wish.

It truly is “by the people, for the people” with this operating system. It holds the philosophy of freedom and freedom of choice to its highest standard, and that’s the one reason so many people have come together and made Linux the single greatest technological ecosystem available.

Remember when we mentioned earlier that Linux had multiple types of desktop environments? This means Linux has many different “distributions,” or different versions that tailor themselves in specific ways to fit the needs of the person using it. If you ever hear someone talk about a “distro,” they are talking about a certain distribution of Linux. These distributions can be either downloaded for free, put onto a USB thumb drive, or burned to a disk and installed on any laptop or desktop computer you wish.

Without charge, every single time.

The most popular ones are Linux Mint, Arch Linux, Deepin, Ubuntu Linux, Debian, openSUSE, and Fedora. Each of these “distros” has a different take on the layout of the desktop environment, but the differences don’t stop there. Some distributions require more skill with computers than others, some have more modern graphical aesthetics (as in, they look prettier than others), and some are built specifically for coding and learning how to program on Linux.

Finding the distribution of Linux that’s right for you is all about accessing your needs and what you find important. If you are a true beginner, you’ll want to stick with something that’s friendly to newbies. Those would be Ubuntu, Deepin, or Linux Mint. If you consider your skills to be average, then you could use a distribution like Fedora or Debian. Remember, because Linux is free to alter and distribute, as your skills progress you can secure different distributions alter and distribute, as your skills progress you can secure different distributions of the operating system in order to continuously tailor it to your needs.

But, when people finally choose the distribution of Linux they want to use, there is always the daunting task of installing the operating system in the first place. Don’t worry—Linux has one of the easiest installations of any operating system to ever hit the open market. Not only that, but many distributions that have been updated are now circulating something called a “live distribution,” which is where you insert the burned disk or the thumb drive into your technological device and run the operating system without making any major changes to the hard drive of your device.

Yes, this means you’ll get the full functionality of your chosen Linux distribution without ever having to commit to the installation in the first place. This is wonderful if you are trying out new systems, and it gives you the ability to “test drive” the product before you commit to it. Then, when you *do* find the distribution you wish to use, you simply double-click the “Install” icon located within the main file of the disk or thumb drive and walk through the prompted installation process.

Chapter 2 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 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 *Joh n*

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 command. If you run the --*help* option with any command, you will get a set of options, which can be passed with that command. So, it's not 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: director y
```

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 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 lin e

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.

/ us r

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.

/hom e

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.

/de v

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/message s

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/message s

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 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

DocumentsMusicDownloadsPictures
```

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 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 file 2

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 promp t

```
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
```

```
Drawe r
```

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
Thesisthesis_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.tx t
```

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.tx t
```

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 many 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 ???
aretabmap
```

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.txthello.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}.lo g
sunday.log monday.logtuesday.log
[student@desktop ~]$ echo file{1..3}.txt
file1.txtfile2.txtfile3.txt
[student@desktop ~]$echo file{a{1, 2}, b, c}.txt
filea1.txtfilea2.txtfileb.txtfilec.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 3 Benefits of the Linux operating system

In this chapter, we will look at the main benefits of Linux. Here not all are collected, but the main advantages of this operating system are.

1. Linux is faster

Yes, it is. Linux kernel operating systems really benefit in terms of performance ahead of Windows. That is why the web server is most often equipped with this system.

Of course, not every Distributor boasts performance. For example, the world-famous Ubuntu without settings will work much worse than the same Windows 7. But there are Xubuntu and Lubuntu, which will never let you down.

2. Linux is better customizable.

Windows, as such, does not give a special leeway to take at least pre-installed tools to change the appearance. They can do little to help. But Linux !

I use the environment – am an active Linux user, programmer and founder of the ProgHacker website. Today I would like to tell you why it is worth switching to a Linux-based operating system. E of the XFCE desktop. based on this Xubutnu.

Could you do this in Windows? Maybe with the use of third-party programs, yes, but it would give significant damage to your RAM. And the above view works on a completely inconspicuous device - 1 GB OP + Pentium 4.

3. Most distributions are free.

Something I thought... But this, in fact, is a decisive factor for organizations. Almost all distributions are distributed free of charge even for commercial use. You do not need to activate, you should just download and install.

4. Linux is an open source world.

Since Linux itself is free, it has many tools built on the principle of open source software. Not only do you not have to pay for the program, but you can also make changes to them yourself. This explains the transparency of the installed programs – in the official repositories malicious software will not work.

5. Linux is safer

Linux has its roots in prehistoric UNIX systems — and since then many of the principles have remained the same. Since all the code is low-level, it does not depend on any additional libraries, it will be difficult to hack the device. And the number of Linux users is too low for hackers to pay attention as a platform for attack.

6. No need to pull hundreds of accounts

If Microsoft Windows requires the user to bind to multiple accounts, then there is no such thing in Linux. First, the account itself is optional – the number of its applications is negligible. Well, even if it is needed, almost all the services that are somehow related to Linux are tied to Ubuntu One – a single account.

7. Choose to your taste!

Are you a designer? Writer? Painter? Translator? Then, for sure, a distribution kit is ready for you, which contains all the programs ready for work !

If you had to download everything on Windows, then many Linux distributions are based on Linux that is already customized for specific needs.

For example, Ubuntu Studio is suitable for a small (or large) studio, TuxTrans for the translator, Fedora Design Suite for the artist, KXStudio for the musician.

8. Stability

If Windows requires a lot of updates, and your computer's memory is filled every six months, this will not happen in Linux. There is no registry in the usual sense of the word, and all installed packages and programs are easily controlled. So farewell, frequent reinstallation!

9. Linux weighs little

Compare Ubuntu or Xubuntu (1.8 GB and 1.3 GB respectively) with 5 GB of Windows 7. At the same time, the set of Linux programs is not only not inferior to the Microsoft product, but, on the contrary, surpasses it! The same applies to programs – they weigh significantly less.

10. Responsive community

Linux is not Windows, where you are left alone with your problem. First, there is the official English-speaking community (https://www.linux.org/forums/).

Chapter 4 How to Install Linux

Versions

We now will have a quick word about which version of Linux to install. This is very popular distro and has many features. It also offers a lot of support from the FLOSS community. Ubuntu itself also has many "variations" however. They're basically the same Ubuntu core with a different graphic/desktop environments. You will see these listed as either XFCE, Openbox, GNOME, Unity, or KDE, just to name a few.

These different variations of the distros themselves also have different elements to them such as window managers and desktop environments.

A window manager is the software that allows all the interactions with a graphical user interface. These elements provide the different window panels, menus and taskbars, settings and appearances. If coupled, the two provide you with more options (this would apply to desktop environments such as GNOME and KDE versions of Linux). You also could download more than one environment, and switch between them when you want a different desktop environment at each boot up. Most of these environments for Ubuntu come with both the environment and with the window manager.

Desktop Installation:

Again, this version of Linux will give you the graphics interface (GUI) that many are used to and expect from using computer.

To install

Before you start you should make sure you have backed up your computer before trying either version as a precaution. Linux is not as likely as Windows to crash, but it is best to be prudent and protect your data.

Pull up the website for www.Ubuntu.com. Always use legitimate websites for your distros and other needs. Once you see the distro that you need, check your computer's requirements. Is there enough room for install? Are there hardware or software components that do not match what the distro is stated to require? At this stage, you can decide if you need to repair or add anything before moving forward. You may even decide to go with another distro .

From the webpage, select the Linux CD image (an .ISO file) that you would like, and choose to download to a disc or a USB. This is highly recommended to download to the external media over a direct install to your hard drive. It is portable, it is non-permanent, and will not make changes to your computer system. You can boot the system straight from the copy that you make. More will be said about this. The desktop version also will be a lot smaller than the server edition.

After the download, you should boot it off the CD drive or USB. This is called a live environment. The good news is that this is not going to affect or change your computer at all. However you also have the option to install the distro to your computer. Be patient during the download, as in either case, the download may take a while. Remember that this version is a bit bigger than the server version in comparison. It also depends on your hardware, the age of your system as well as your internet speed .

With an install you can easily save settings, files and other components between uses that you can't get with the USB or disc reboot. However, you could also save files to your drive to use on your next reboot with a USB drive, by enabling the "persistence" setting.

Options to Try (Use Live) or Install Ubuntu (www.linuxandubuntu.com)

For the install on your drive:

For hard installation we recommend to use the guided installation, unless you are more advanced with Linux (in which case, you probably would not be reading this guide).

Ubuntu makes the following, additional recommendations prior to installing the system:

- **Make sure you are plugged in and connected to the internet. Stay connected.**
- *Download updates while installing and choose to also install third-party software.*

You should remember that by installing Ubuntu on your entire hard disk you will erase all data that is currently on the drive. Backup all data before you install Ubuntu on your actual drive! If this scares you, you may want to look at the Live installation instead. If you want to also keep another backup copy in a location where they are safe and away from corruptible hardware. Try a cloud based program such as Jungle Disk. It does have a low cost to subscribe but the service is worth it, and it plays well with Linux.
www.ubuntu.com

For the CD/USB (Live) install:

If you wisely chose to do a live environment installation, once it has been downloaded and rebooted from the CD or USB you just follow the series of prompts. Some prompts are simple information from you such as language and preferences. Some prompts will ask questions about what you want to do with your system afterwards, or who may have access to the system. Other prompts include a question to partition (for advanced users). You may have to disable the " Secure Boot" setting if it will not allow you to boot Linux with some newer versions on Windows.

Upon rebooting, you will immediately see that your tabs include functions, applications, an internet browser, and tools like MAC or Windows. Home or My computer is your "C" Drive.

NOTE: If your computer doesn't read the CD or USB and install automatically, try the F12 key for the menu to allow you to boot.

Virtualization:

Another great feature of any distro is the virtual desktop, which enables you to toggle between operating systems. A virtualized desktop is exactly that. It is the user interface in a virtual environment, and it is stored on a remote server instead of locally. The virtual operating system runs within an application. This can be much easier and more efficient than dual booting.

You can get these in an App. Some popular ones for Linux are Microsoft Virtual PC, VMWare, VMWare Workstation, or VirtualBox. Do some research on these to explore this option as well.

VirtualBox Manager Screenshot (www.lifehacker.com)

Dual Boot Option:

So you have the choice to install Linux or to use the live environment. You also have the choice to run Linux by itself, wipe out your current operating system entirely and substitute it with Linux. You also could opt to run it side by side with your current system such as Windows.

If you chose the latter, you can partition the system to allow for both, and toggle between them as needed. Being able to troubleshoot Windows or recover data is another benefit to having the partition, or to having the live USB or disc. Some people like the option of having both types of systems, and some will keep one or the other exclusively.

One option to maintain both is to set up a dual boot configuration where you keep two operating systems. Wubi is a program that will automate installation, and won't affect partitioning (sectioning off) of your computer. If you want to explore this, go to the Wubi website at: www.wubi-installer.org to access this program.

If you are more tech savvy you can manually create your own dual boot install, create partitions. Another point to note is that if you have a MAC OS, there may be other things to consider when sharing the space. The system may need something special to help with the co-existence. The program “rEFInd” can be installed and will help you access the OS you want at start up.

The Look of Ubuntu Desktop Version

So, by now you have chosen a version you would like, and you are deciding to use the Desktop Version of Ubuntu.

This is what the Desktop Version may look like once you have it up and running. As you can see, the GUI provides an interface that is very familiar to Windows and MAC users. You will see icons, task bars, tools, and see how you could multitask by toggling the windows.

Server Version:
Again, using Ubuntu as an example for an installation, you will start by fetching and downloading the ISO file from the website www.Ubuntu.com . Chose the version you will need for the task (desktop, netbook, server, cloud, etc.), and burn the one you need after you check your computer settings. Make sure the requirements are compatible.

You will probably want to choose a larger bit if you want a server. Download it to a disc or USB, in the live environment again, which is preferable, and reboot off this and not your hard drive.

It make take some time due to its large size and internet speed. Be patient. It also won’t take up too much space. It also does not matter the age of your system. In most cases, you can go back 5-6 years and use an old system. Remember that the server version is bare bones, no graphics .

Ubuntu Installer Boot Menu Screen (www.pendrivelinux.com)

Follow any prompts. You will see something asking for a Name/ Hostname and you can just call it Server, unless you feel inclined to give in a moniker. Answer yes to any of the settings offered in the guides, except for some questions of choice when you need to provide a particular response. Some of the questions may get too technical for a beginner. It is recommended to stay only with what you are comfortable in doing.

When you do this reboot and go through the set up prompts, immediately you will notice the Shell. Remember that the server edition will make it so that you must have knowledge of commands after following the installation the guides. A Filesystem Hierarchy Standard (FHS) will help you navigate locations and commands for using Linux by Commands.

At the install you follow prompts to add your user name, password and other set up features. The option for encryption would secure the files in your home directory, but it is not suggested to do this right now until you have more knowledge about Linux. The frequency of updates also are not needed but you can later do periodic, manual updates will be fine. Unlike Windows, you will not need the frequent, automatic updates.

You will also chose what type of server you would like it to be. If you aren't sure, you should do some Google searches. For example, what type of server for a website, or files? Once you chose the type, it will install what you need, and only what you need. Again, it is a pared-down version and you will have this opportunity to choose exactly what you need, for exactly what you chose.

Example of Ubuntu Server Edition (www.pwrusr.com)

Dual Booting Servers

There is also the option to partition the Server version as well. This however, if usually much more complicated than with the process for the Desktop. It is suggested that you really do some research before you decide to partition the Server, to share space with another Operating System. It may not be something for a beginner to delve into just yet. Then again, you won't be able to learn if you cannot explore.

Adding a Database to MariaDB

In other books I have written about Linux, I had users use a web tool called phpmyadmin to add users and databases. In this book, I want to take a different approach and do this from the command line interface.

As we progress into installing WordPress, it will need a database to store its information into. In these next steps, we will create that database and a user to connect to that database.

To begin, we must login to MariaDB's root account by using the following command:

```
sudo mysql –u root –p
```

As you may have guessed, the *u* flag symbolizes user and *p* stands for password.

You will be prompted to enter the root password for MariaDB. You would have set this when we ran the command to secure the install .

Once you are logged in, the next step is to create the database. We are going to name this database ‘wordpress’.

CREATE DATABASE wordpress;

Please note that capitals do matter here as they do for other commands. Additionally all MySQL and MariaSQL statements must end with a semi-colon.

That was easy! Now we need to create a new user account that will only be able to operate with the database we created. This allows for good security and better control of database permissions.

For me, I am going to call the new user account ‘wpuser’. In the command below, you can use a different name if you wish and change ‘password’ to a secure password. We will need this password as we connect WordPress to the database.

CREATE USER wpuser@localhost IDENTIFIED BY ‘password’;

Right now you have a database and user account, but the user account cannot yet access the wordpress database. We need to grant that user access to the wordpress database with the following command.

GRANT ALL PRIVLEGES ON wordpress.* TO wpuser@localhost IDENTIFIED BY ‘password’;

Our user now has access to the database, but we need to do something called flush so that MySQL/MariaDB knows about the privilege changes we made. Use the following command:

FLUSH PRIVLEGES;

Now that we have done these commands, we can exit MySQL/MariaDB by using the following command:

exit

Easy right? There is much more to learn about using MySQL and MariaDB. I recommend you continue expanding your knowledge on these topics as they are a very in demand skillset.

Chapter 5 Installing And Setting Up Linux

There a couple of ways to install Linux from your current operating system. You can choose to download Linux from either a live CD or a USB drive, and then boot Linux from your system in order to install it. For beginners, this can get a little confusing, but it doesn't require a lot of technical knowhow and once you get through the first time, installing another system is simple if you choose to do so. For this book, we will use the example of installing Linux on a computer currently running Windows.

Installing from a Live CD

To install Linux from a live CD, you will first need to download the appropriate file. Before you do this, you will need to select your distribution and visit the homepage. From there, you can go through the downloads page and find the file that is suited to your system. There are usually a few different files for each system: some are optimized for netbooks and others for desktops, offering both 32- and 64-bit versions. After you locate the version required by your system, download the .ISO file.

Once you have the file on your computer, burn it onto a blank CD. There are programs you can download for this but generally your operating system should have some built-in programs to make this step relatively simple.

After the file is burned to the CD, restart your computer. Before the operating system boots up, you will be presented with the option to enter the boot menu. In this menu, tell the computer to boot from the CD, save your options, and restart once again. If you are unsure of where to go to select these options, consult your operating system manual and look for the boot options. This should present you with a step-by-step procedure for setting the boot options.

You are then presented with the option to either install Linux or give it a try. It is highly recommended that you go through a trial run before installing Linux to ensure that you have a good feel for the desktop and that it is what you had in mind. Once you have finished your practice run, it is easy to select the option to install.

Installing from a Live USB

If you are deciding to install from a USB, which is the more contemporary option, you will need to install an extra piece of software. Unetbootin is one option if you are running Windows. It is simple to download and get the program started. From there you can decide on your distribution from the list offered and download it to your USB. Not all the distributions are offered by

Unetbootin, but that doesn't limit your options. You can simply install the .ISO file from the download page of your distribution to your desktop and then have Unetbootin pick up on the file.

You then need to follow the same process as with the CD: restart your computer with the installed distribution on your drive connected and go into the BIOS, select the boot options and choose the USB hard drive as the first option on the list, save your options, and restart once again. You will be taken to the Unetbootin menu where you can start your Linux session just as you could with the live CD method.

The Installation Process

The way a distribution is installed on your system varies, but you are generally given a setup guide that walks you through the process. If you are deciding to install Linux alongside another operating system, such as Windows, there is a little bit of work that needs to be done during the installation process to ensure that everything runs smoothly.

The first step, in this case, is to partition your drive. This will tell your computer where you want Linux to be installed. This is a simple task that requires little effort on your part. You just need to select a space on your hard drive and allocate some of the free space you have for the installation and running of the operating system.

If you are installing Ubuntu, the program will generally do this for you by selecting its own space. From there, the rest of the installation process is quite simple.

For other distributions, you will be required to partition the space yourself and some partitioning tools can be complex. In this case, you will need to create two new partitions. One is to set aside space for the operating system itself. The recommendation is at least 10 GB for this. You should name this partition "Ext4" and set the mount point as "/". Then create a second partition for what is called "swap space**.** " This is where the memory well be held in order to run the operating system and keep things processing quickly. If you only have a small amount of RAM available, make the swap partition twice as large as your available memory. If you have a decent amount of RAM (over 3GB), make the swap partition the same size as your available RAM.

GRUB and other Bootloaders

Before Linux completes the installation process, it will install a new bootloader onto your system under the name GRUB. This will be replacing your standard bootloader and will give you the option upon start-up to open your desktop with Windows or Linux. You do not need to do any work during this part of the installation process; you only need to sit back and let your distribution install GRUB.

For those of you who have a Mac, this is one of the parts of the process that will work differently and you will need to install GRUB onto the Linux partition manually.

Windows users: make a note that if you decide to uninstall and reinstall Windows at any point, you will need to install GRUB manually once you get your operating system set up again.

If you prefer something with a more enhanced user interface, you may want to install another bootloader that is compatible with Linux. An example of this is a bootloader called Burg. However, you do not need to do this straight away and can wait until you have finished installing Linux before setting up a new bootloader.

If you have followed these steps, you are ready to go. Your Linux partition should now be installed and all you need to do is restart your computer to be taken to the GRUB menu. From there you can decide whether you would like to boot Linux or Windows. Obviously, for the purposes of this book, we want to boot Linux to get a feel for it for the first time. You will notice that there are several pre-installed apps that come with your Linux distribution. Some of these will be useful to you and some will not, but you can look at each one and figure out which ones you can use.

In some cases, installing a new operating system can cause hardware to become unstable. This is usually because the hardware does not realize that you are using Linux and some miscommunication can occur. This is especially the case for Wi-Fi. There are some steps you can take to ensure that all your hardware is compatible with your new installation.

Chapter 6 Computers and Math

Arithmetic comes naturally to computers, and fortunately, programming languages tend to make it easy. The symbols are intuitive, and you shouldn’t have much issue at all with them. I’ll list the general mathematical symbols and when you may use them:

+This symbol means, of course, addition.

-This symbol means subtraction.

*This symbol means multiplication.

/This symbol means division.

%This symbol is the *modulo* . It refers to the remainder of a given division problem. For example, 7%2 would be 1, since the remainder of 7 divided by 2 is 1.

This symbol turns a number into an exponent. For example, 22 would be two squared, or four.

//This symbol refers to floor division.

Additionally, there are some assignment operators intended to make arithmetic in programming more intuitive. Assignment operators are operators which can be used to assign values. The most common one is the equals sign, which gives one thing a certain value. However, there are more.

For example, instead of typing x = x + 1, you can just type x++. Instead of x--, you can just type x--.

There are also shorthand assignment operators. These are intended to take arithmetic operations and shorten them. For example, instead of x = x + 3, you could simply type *x += 3.*

Here are all the shorthand operators and their uses:

x += y-x = x + y

x -= y-x = x - y

x *= y-x = x * y

x /= y-x = x / y

x **= y-x = x**y

x //= y-x = x // y

Computers and Logi c

In this portion of the chapter, we’re going to start to discuss the foundations of logic. Logic is a very dense topic, but it’s also rather simple. Essentially, when it comes to statements that we make, we tend to think of them in very abstract terms. We even go as far as to make unnecessary abstractions where they really don’t need to be made. This isn’t how computers work.

As I said earlier, computers are dumb. They take things as literally as you can make them sound. If you say something to a computer, it interprets it as the literal form of the statement.

This is important, because logic is also used in pretty much every decision we make every day, and paralleling those decisions to a computer will prove very difficult if you don’t understand the different ways in which we process logic.

Logic is essentially built off *comparisons* . You may not realize this now, but every decision you make is based on a comparison, whether implied or explicit. Even if you don’t outright compare one value to another, you are almost always comparing something to some *unknown* values. For example, if you say “do we have any eggs?” you aren’t just checking to see if there *are eggs* , you’re checking to see if the number of eggs that you *have* is greater than zero. This is an implicit logical statement.

Comparisons are key to making your program branch out and do many different things. This concept is known as *control flow* . Control flow is essential because it’s the way of making your program *think* . Without control flow, your program isn’t going to *do* anything. Sure it may have basic operations it performs, but as far as internal decision making goes, there will be no such thing.

In Java, C++, and Python alike, these are the comparative operators:

< Less than

<= Less than or equal to

> Greater than

>=Greater than or equal to

== Equal to

! = Not equal to

These are used to build *statements* . Statements are logical comparisons which have one or more comparison built-in. Here is a pseudocode version of a logical comparison:

```
peaches = 6

    bananas = 3
    peaches > bananas

//Since this statement is true, “peaches > bananas” would be true.
```

Using these statements is essential to building the internals of your program. You can also group different statements together into one larger statement.

These statements are built using what are called *conditional* operators. There are three primary operators that you need to know at a lower level: *and* , *or* , and *not* .

And signifies that the two statements are connected. This means that if either statement is false, the entire statement is false. It’s only true if both statements are true.

Or signifies that the two statements are disjointed but implicitly connected. If either statement is true, then the whole statement is true. They don’t both have to be true for the statement to return as true.

Not signifies that the entire statement following is untrue. Therefore, the validity of the entire statement *including* the not operator is based upon whether the statement *without* the not operator is false. For example, if *morePeaches* is true, then *not morePeaches* is false; conversely, if *morePeaches* is false, then *not morePeaches* is true.

Conditional Statements

With all of that said, it’s time that we start to talk about how you can use statements in your program to build logical lines for the program to run along. Conditional statements are used to check to see whether a given statement is true, then act accordingly. For example, if the conditional statement were *if the number of eggs is greater than zero* , then the program would evaluate whether this was true and then execute something if it was. This is the first kind of conditional statement, known as an *if* statement.

Here is pseudocode for a basic if statement:

```
if (statement) {
```

```
// code to execute

}
```

This is what I like to refer to as a *passive* if statement. I call it so because these statements will only execute if the comparative statement is deemed to be true. However, there is no escape clause in these; there is no programmatic way of designating what should happen if it turns out the statement is false. The program would simply skip over the statement and move on to whatever was next in the code.

You can also create an *active* if statement by using what is called an *else* statement. Else statements are appendices to if statements that provide a set action to take if the initial if statement turns out not to be true. Here is an example of an active if statement :

```
if (statement) {

// code to execute

} else {

// code to execute if first block fails

}
```

When you use an active if statement, even if the statement is false, something is still going to run.

Which one you'll need to use can vary immensely depending on what you're needing at that specific moment. Deciding which one to use is a skill that will grow on you as you gain more experience.

Sometimes, though, you need to catch more than one clause. For example, you may want to do something if a number is less than 6, if a number is greater than 37, and if a number is somewhere between the two. For this, you need another part of your if clause.

So how do you go about doing this? You use the *else if* statement. Else if statements give you a way to test additional statements before moving to your catch-all else statement. They are sandwiched between your if statement and your else statement and you can have as many of them as the situation calls for:

```
if (statement1) {

// block 1
} else if (statement2) {
// block 2
} else {
// catch-all block
}
```

If statement one turned out to be true, then the first block of code would execute. If statement two turned out to be true, then the second block of code would execute. If neither turned out to be true, then the third block would execute. This is the simplest form of control flow.

Loop s

The other form of control flow is known as the *loop* . You use loops every day without even thinking about it. Much like if statements, you're constantly making comparisons in your head, you're just not thinking about how they, well, *are* comparisons.

What do loops allow you to do? Loops offer a methodical way to check whether a statement is true, then you can take a course of action if it is or isn't.

You use loops all the time in your internal logic, you just don't consider that they're loops. Think about counting out loud from 1 to 10. You start with the number zero, even though you don't say it - you increment the number by 1, then you say it out loud. This continues for as long as number *n* is less than or equal to the number 10.

There are two different primary kinds of loops: while loops and for loops. They both serve diametrically different purposes, but that won't be completely clear from the get-go. What's more is that learning how to use these properly is more of a question of experience than something that can be properly taught from a book in any way shape or form, but we can outline the bare essentials and primary differences by the two and start to delineate their uses.

While loops are primarily used for simple statement checking. That is to say that whenever you just need to run something over and over for the duration of time that a given statement is true or false, then while loops are your friend.

This gives way to them being used for something called a *game* loop. Game loops are called such because they allow you to repeat a chunk over and over until a certain variable is tagged true or false. They are common in computer games, where the variable may be something like GameIsOver = False and the loop will run over and over until GameIsOver is true. Whenever something triggers that GameIsOver = true, like the player running out of lives or the dealer winning the card game, the loop will exit and something else will happen.

The other kind of loop is the *for* loop. In C-style languages like C++ and Java, the difference between the two isn't as clear-cut as it is in a language like Python. However, by using it in Python, the primary use of it in C++ and Java becomes exceedingly clear. For loops are intended for *iteration* . What they do is allow you to define a certain set or range, create an iterator variable (which is a variable that will change in some way with every instance of the loop as the set or range are moved through), and then move through the loop until the defined set or range have been completely iterated through.

The pseudocode for a while loop is like this:

```
while (statement) {

// code executes until statement is false

}
```

And the pseudocode for a for loop is like this:

```
for (iterator variable; range definition; increment setting (in C++, Java, etc.)) {

// code executes until range has been iterated through

}
```

There is also the *do while loop* . What the do while loop does is allow you to define a block of code which will most *definitely* execute once, but it will only execute more if a given statement is false. Compare this to a normal while loop - if the initial statement that was given is untrue, then the entire loop won't execute at all. With the do while loop, you ensure that this won't happen by giving you the security to know that the given code will execute at least once.

Here is the pseudocode for a do while loop:

```
do {

// code executes at least once, more if the while condition is true

} while (statement)
```

Loops are the other essential part of control flow which allows you to break your program into different paths which can change depending upon the way that the program's execution is going. It allows you to respond to user input or any other variables. All in all, it lets you write a program that isn't a straight path from point A to point B.

Methods

Methods allow you to add a layer to your programming called *modularity*. We'll be discussing this in a more in-depth way in the section which follows this one, but modularity is essentially the ability of something to be broken down and reused over and over. Methods allow you to add a layer of modularity to your program because they let you reuse a certain chunk of code over and over. They also allow you to detach different parts of your code so that the code is more readable and manageable.

Another perk of modular programming is that it allows you to change one part of the code and have it change throughout without requiring you to go through and refactor what could amount to hundreds of lines of code in big programs. This is a huge difference and will save you a lot of pain and frustration and time spent debugging in the future .

So what are methods exactly? Well, to understand them better, it might be better to use their *other* name for a minute: functions. (Functions and methods are functionally the same thing; method is the name used in more modern languages, usually, where function is preferred in languages like C and C++ - other than the name, they are the same exact thing)

This is the second reference to high school math classes that we've had so far, and it's equally relevant to the last one. Remember back in your high school algebra classes where you'd have a function like f(x) = 3x + 4 and you would plug in the given x value? For example, if f(x) were f(3), then you would plug three into the equation?

In an abstract and very basic way, this works in an extremely similar manner to the way that functions do in programming. In the function f(x), *x* is what is called the argument. You feed something to the function. The function itself is just the name of the entire statement - the term *3x + 4* can be referred to as the *action* of the function. The action can perform operations upon the arguments of the function. The result is what the function *returns* . The end value given above of f(3) = 3(3) + 4 would be 13. The return, therefore, after all operations are finished and the action has been carried out, is the *return value* of the function. Since f(x) = 3x + 4, the return value of f(x) with the argument 3 is f(3) = ^{13} .

In programming, this can change just a little bit. For example, a function doesn't have to take any arguments at all. This is also technically true in mathematics, where function *f* (x) could simply be 14 + 4. All values of x would be *14 + 4* . If a function doesn’t have arguments, then whatever you define as the action will always take place in the same way. This is greater if you need to run the same bit of code at various times, like so:

```
function bark() {

print “bark!”

}

BARK() ;
```

Anytime that the function *bark* is called, then it will print out *bark* . Why is something like this useful? Imagine a code chunk where you must print out bark multiple times - it’s hard, but this is a tutorial book so just try, some books certainly have stranger examples. If you decided you wanted it to instead say “barkbark!” instead of “bark!”, you could just change your *one* line of code in your function instead of going to every line in your program that says “print ‘bark!’” and changing it.

Methods are relatively simple in what they’re intended to do and aren’t terribly difficult to understand. Here is the pseudocode for a method:

```
method(arguments) {

// actions

return value;

}
```

File Input and Outpu t

This is the last topic that we need to talk about before we get to the monolithic concept that is object-oriented programming. File input and output is an essential part of programming that will almost certainly be demanded of you at some point in your programming career. It’s not terribly difficult, and most file input and output operations will follow the similar line of logic.

First, you *open* the file. Opening the file means either creating it if you’re writing or reading it if you’re appending or reading. In the latter cases, you will normally have to have some sort of exception set up to catch if the file doesn’t exist. Otherwise, your program will simply crash.

Then, you *modify* the file. Reading and writing files are self-explanatory. You either take in data, write to the file, or both. You can also append to a file. This is where you go to the final point in a file and add data to the end of it.

Afterward, you *close* the file. Closing files is where you close the stream between the program and the file. This ensures that no corruption or data leaks occur. There is no point to keep resources open if you aren't using them.

These are all simple operations done with methods which are relative to the given programming languages you'll be using. We will go into more detail on them in the programming language specific chapters.

Object-Oriented Programming

Object-oriented programming is not a very *difficult* concept. However, it is an *important* concept, and it marks an important milestone in your ability as a programmer because you go from writing simple programs without a great degree of nuance to far more detached programs that can scale to much greater sizes.

There are a few different aspects to object-oriented programming. The first is *modularity* . Modularity is the idea that things should be able to be reused over and over in different contexts, and you should be able to detach and replace parts without breaking the entire system .

There is also abstraction. Abstraction is the idea of detachment; it aims to make the programmer's life easier by keeping them from doing any heavy lifting and using simple terms to refer to things within the language and within core classes. The philosophy, carried out, will lead to very readable code that is very easy to work with as well. Abstraction refers also to the idea that the programmer should be shielded from having to do any low-level operations, like directly handling the computer's memory.

Then, there is polymorphism. Polymorphism is the idea that an abstract operation can be done with different value types.

In addition to polymorphism is *inheritance* . Inheritance is the idea that something should be able to build upon the qualities of another unit to form itself.

Lastly, there is encapsulation. Encapsulation is the idea that things should be neatly tucked together under one roof .

So what exactly are *objects* ? And moreover, what is a *class?*

A class is simply a method of grouping together different values and functions under one title. This lets you create new instances of that specific class, each with unique versions of the class' variables and functions. However, they also share core functions and variables. For instance, if I were to create a class called *dog* , I could make new instances of *dog* called *corgi* , *yorkie* , and so forth. But if *dog* had a function defined as *bark* , then all the instances of *dog* would use the same *bark* function. If the dog class had *numberOfLegs* defined as 4, then all dogs would have that as their value for *numberOfLegs* .

When one says *object* , they are simply referring to an instance of a class. For example, all the instances of dog - corgi, yorkie, etc. - would all be *objects* .

Object-oriented programming refers, broadly, to languages which are geared towards using objects regularly. Of the three programming languages in this book, Java is the one that is primarily object-oriented; however, object-oriented programming is so popular that the methodology has become insidiously ingrained in other forms of programming as well. Therefore, it’s important that you know the core terms and practices. Object-oriented programming goes a little beyond the scope of this book, but both languages that we study in this book - as well as most modern languages, for that matter - will make use of object-oriented terms and structures, so it’s important you know what it is.

Chapter 7 Package Management

In today's world, software can be quite complex. Part of this complexity stems from programs being split into single components made available as single packages, such as binary data, shared libraries and documentation. This concept of software development is called "modularity", and is generally accepted and in widespread use. Debian, for instance, offers more than 60,000 different packages.

Package management covers all actions needed to handle these software packages including installation, configuration, removal, and updating packages. It also covers resolving package dependencies (packages that depend on each other).

9.1 Understanding Package Architecture

To explain the Linux package architecture, we will be using Debian for illustration. The Debian distribution is known for its stability and exceptional packaging system. The descriptions that follow are valid for all Linux distributions that are derived from Debian, such as Ubuntu, Linux Mint, Knoppix, Kali Linux, Grml and Xandros. Keep in mind that the names of packages may differ, as well as the distribution-specific package mirrors and the names of package classes.

The Debian package management ecosystem contains quite several different components such as package lists, package sources and package mirrors, different software packages, software to manage the package setup, and the package cache. These components are explained in more detail below.

Package Lists

Debian contains a list of available software packages. This list is referenced in the file /etc/apt/sources, and for specific components and personal additions in the files in the directory /etc/apt/sources.list. d. As an example, the messenger "Skype" uses the file "skype-stable. list". The lists use a text format with one package source per line:

```
deb http://ftp.de.debian.org/debian/ stretch main contrib non-fre e

deb-src http://ftp.de.debian.org/debian/ stretch main contrib non-
free
```

Each line either refers to a binary package type (deb) or to a source package type (deb-src). The package type is followed by the package source, which refers to a repository that provides the software packages. Next, the line contains the name of the Debian release, which can be specified either as the nickname of the release or the branch.

The nickname originates from a character of the film “Toy Story”. The branch refers to the development stage of the release, and can be oldoldstable (pre-previous release), oldstable (previous release), stable (current release), testing (next release), unstable (future release) or experimental. After the name of the branch, the package classes are named.

The entries for security updates of Debian 9 Stretch look as follows:

```
deb http://security.debian.org/ stretch/updates main contrib non-free

deb-src http://security.debian.org/ stretch/updates main contrib non-free
```

In order to refresh the package list from the referenced package sources, invoke the following command:

```
# apt-get update
```

Package Sources

The Debian project maintains an official list of package sources named “package mirrors”. This list is divided into primary and secondary package mirrors. Primary package mirrors offer the full spectrum of supported architectures, while secondary mirrors only offer a subset. For the United States, the official primary package mirror is:

```
ftp.us.debian.org
```

You can view the current state of your desired package mirror here:

Package Types

Debian divides its packages into three categories:

- *Main - free packages.*
- *Contrib - free packages that depend on non-free packages.*
- *Non-free - non-free packages that do not provide source code.*

Packages that belong to the non-free category include firmware drivers, Skype messenger, and the Flash plugin. Every single entry in the list of source packages specifies their category of software.

Package Management Tool s

In order to maintain the selection of software packages installed on your Linux system, several tools are available:

- *dpkg - used to manage single packages.*
- *apt, aptitude, synaptic - used to manage packages with their dependencies.*
- *tasksel - used to manage entire software collections (tasks).*

The image below depicts the different levels on which the tools operate and how they work together. The two boxes with dotted outlines refer to the shared libraries that handle the package management tasks.

On the lower level, we have “dpkg”. Its task is to install a single Debian package or to remove the contents of a package that has already been installed. In order to query packages and files, it makes use of the two tools “dpkg-deb” and “dpkg-query”. “dpkg” allows you to maintain your package base but does not handle package dependencies .

Typically, the upper level is represented by the tools “apt”, “aptitude” and “synaptic”. These tools have the task of simplifying package management by combining all actions into one application. These tools use either the shared libraries “libapt-inst” or “libapt-pkg”, or communicate with “dpkg” to perform the package management tasks. This includes handling the package dependencies.

Package Cache

The package cache is located at /var/cache/apt/archives. Whenever you install a software package via “apt-get” or “aptitude” the appropriate package is retrieved and stored locally at this path. To illustrate, below lists the cached “deb” packages using *ls* .

```
$ ls /var/cache/apt/archives/*.deb
/var/cache/apt/archives/cpuid_20140123-2_amd64.de b
/var/cache/apt/archives/libxml2-utils_2.9.1+dfsg1-
5+deb8u7_amd64.deb
/var/cache/apt/archives/python3-bitarray_0.8.1-1_amd64.deb
/var/cache/apt/archives/python-bitarray_0.8.1-1_amd64.deb
$
```

The more software you install, the more “deb” packages are cached. It is recommended to clear the package cache from time to time by using either *aptitude* or *apt-cache* :

```
#        aptitude
clean
```

```
#        apt-cache
clean
```

9.2 Day-to-Day Administration Task s

This section deals with basic package management actions needed in day-to-day life as a system administrator.

Displaying Installed Packages

The command *dpkg -l* lists the installed packages and their status. The image below shows the output.

The single columns contain the following information:

- *status - the status of the package, such as fully installed (ii) or removed and still configured (rc).*
- *name - the name of the package.*
- *version - the package version.*

- *architecture - the architecture or platform the package is built for.*
- *description - a short description of the package*

Installing a Packag e

In order to install a package, use the *apt-get* command. The output below shows this for the “htop” package.

Updating an Installed Package

Updating an installed package is quite easy and done in the following way. First, update the package list using *apt-get update* as described earlier in this chapter. Then run *apt-get install* to update the package. If an updated version of the package is available, the new package will be retrieved from the package mirror, the old package will be removed, and the new one will be installed.

Removing an Installed Package

Removing an installed package is done in a similar way as installing or updating. The *apt-get* command has the option to remove the package as shown below.

This step removes all the package content except for the configuration files. In order to entirely remove the package with its configuration files, add the switch -*-purge* as follows:

```
# apt-get remove --purge htop

...

#
```

Chapter 8 Linux Distributions

Globally, there are so many computer users these days. These computer users are different in what they like in their computers and in what they use their computers for. Linux developers understand that very well, and they have created different distributions and versions in order to suit different needs of computer users across the globe.

These distributions have been created with considerations to software packaging, installation process as well as the updating process. This is what makes each of them different from the other, and this forms the basis that users use in order to choose a distribution to go for.

Linux has a distribution for the beginners as well as the experts in computer use. These distributions have been created in order to ensure that the needs of all users have been met well and that they are having no issues as they use their computers. Distributions for the newbies are much easy to learn, understand and use; and those for computer experts are much complicated and can be challenging for a user with few computer skills.

These distributions are the different Linux versions that have already been created. They are also called *Distros* in short form. Close to all these distributions can be downloaded for free, burnt to a CD or copied to a USB flash drive and installed to as many computers as the user wants.

The most popular Linux distributions are as follows:

- Linux Mint
- Ubuntu Linux
- Deepin
- Fedora
- Arch Linux
- OpenSUSE
- Debian

Ubuntu Linux is the most modern user interface and it is the most preferred among the above distributions. There are people who prefer a much older version though like the openSUSE. Each of these distributions has its own kind of take on the desktops and other computers. Therefore, you must understand them all in order to make a perfect choice.

There are over 100 such distributions today. So for a wise choice, you have to study them all in detail, bearing in mind what you need for your computer.

For desktops and laptops, go for Ubuntu, Mint, Fedora, Debian or openSUSE.

This will depend on your expertise level. OpenSUSE is perfect for laptops of all kinds. The tool that comes with this distribution allows for wi-fi connectivity and other capabilities that are fit for laptop users, like a simplified dockingstation.There is no other distribution that can compete with openSUSE on the laptops.

The **servers** here are for instance:

- Ubuntu server
- Red Hat Enterprise Linux
- CentOS
- SUSE Enterprise Linux

Some of the server distributions are free; for instance,centOS and Ubuntu server and some of them have a price they are associatedwith like the Red Hat Enterprise Linux and SUSE Enterprise Linux. Those with a price includes support, and the cost is quite low. Therefore, you will not strain financially to acquire it if you want a distribution with a price on it.

The Most Widely Used Distributions

Ubuntu Linux is the most widely used distribution of Linux. Most of the users that have been using Windows operating system are choosing Ubuntu now. The reasons could be:

a) Because learning it is very easy when compared to other Linux distributions.

b) It has the widest range of applications for you to choose from. This means that you can do anything on your machine with the right apps in place.

c) Its application center is well developed, making it easy for the user to search through the applications that he wants to install .

Linux Mint has also been growing in popularity over the years. Many people believe that its popularity is close to threatening the dominance that Ubuntu has had for so many years. Linux was developed out of user suggestions to make Ubuntu better, and this means that users have a better experience with it.

Determining the Right Distribution

The most important thing at this level is to choose the right kind of distribution to go for. This is not a big deal at all if you know what you need. This should guide you:

a) What kind of computer user are you? Are you well skilled or a beginner? Choose a Linux distribution that matches the level of computer skills you have so that you will benefit from it. You also need a distribution that you can learn faster and use better all the time.

b) Would you prefer a standard or modern desktop interface? This varies from one person to another. There are a good number of people who prefer the modern desktop interface, and there are also a good number of computer users who go for the standard desktop interface. It is good to take time and learn about them both, the pros and cons so that you will choose a better interface for your needs.

c) Do you want a server or desktop distribution? This depends on whether you want to be the system administrator, or the system is only for your computer.

If your computer skills are just basic, it is good to stick to a beginner-friendly distribution like Ubuntu Linux and Linux Mint or Deepin. Computer users with above average skills could choose distributions like Fedora or Debian. Computer gurus who have mastered the skills of computer use and system administration can choose a distribution like Gentoo. This is a muchadvanced kind of Linux distribution.

Users that are looking for server-only distribution will have to choose between desktop interfaces or to decide whether they want to have it through a command-line only.

System administrators can choose a distribution as per the kind of features they want to go for. If you are looking for a server kind of distribution that will offer you everything that you need from a server and out of box features, go for centOS. If you are looking for a desktop distribution that will allow you to add features as you need them, you can go for Ubuntu Linux or Debian distributions.

New users could make work much easier if they compared distributions that are fit for use by newbies only. This way, you will save time in making the right choice and make the best choice in the end.

Chapter 9 Basic Shell Configuration and Customization

There are several hidden configuration files in your user directory that allow you to configure your shell. Since we are only looking at the BASH shell, we shall explore the configuration files that are used to control the BASH shell environment. Other shells will have their own configuration files that will work in a similar way but use a potentially different syntax.

Since this is just an introduction to the shell we shall not delve too deeply into these files, but more take this opportunity to make you aware of their existence and the potential customization that can be performed. These files and their customization would be beneficial for you to explore further on your own.

Remember that we can view the hidden files in your home directory, and in this case we shall request a long listing format to obtain maximum detail, by using the following command whether we are in our home directory or not

```
$ ls –al ~
```

Not all systems will have every file present by default, but if we require them we can just add them.

Since we are talking here about the editing, or modification, of files that are system configuration files, it is worth mentioning that it is always a good idea to create a safety copy of the file you are manipulating. This then allows you to go back to the original if the amended file does not work for some reason, or alternatively to use the *diff* command to look at the differences between a working copy and a non-working copy of the file.

As we have seen, environment variables are used to set up default values for properties and to control the way in which the shell functions. One of the most important roles that these startup scripts take care of is the initialization of environment variables, since this means that we do not need to set them up every time that we use the shell and that they persist..

Setting Environmental Variables at Login

It is important to understand the distinction between different ways of starting the shell session. There are Login, Non-Login, Interactive, and Non-Interactive Shell Sessions and the bash shell reads and processes different configuration files depending on how the session is started.

A login shell requires that the user is authenticated before it starts, such as when connecting using SSH, (Secure SHell). If we start a shell session from our authenticated session, such as using the *bash* command, a non-login shell session is started, since we will not be asked for authentication details.

There are also interactive and non-interactive shell sessions. When the shell session is attached to a terminal it is interactive, when it is not it is non-interactive. A normal session that begins with SSH is generally an interactive login shell. When scripts are run from the command line they are usually run in a non-interactive, non-login shell.

The type of shell session determines which files are used for initialization and in what order.

Sessions started as login sessions read configuration details from the global /etc/profile file first. They then look for the first login shell configuration file in the user's home directory to get user-specific configuration details reading the first file that it can find out of ~/.bash_profile, ~/.bash_login, and ~/.profile. Once it has found one it does not read any further files.

Conversely, non-login shell sessions will read /etc/bash.bashrc and then the user-specific ~/.bashrc file to build their environment.

Non-interactive shells read the *BASH_ENV* environmental variable and then the file there specified in order to define the new environment.

We have already looked at how to create shell and environment variables. We will use this knowledge to create them at startup of the shell session. Since we usually want to set user specific environment variables for both login and non-login shells so we will define these variables in the ~/.bashrc file.

Within these files we can set environment variables thus

```
export VARNAME=valu e
```

Note that there is no command prompt, *$* , here, since these are commands entered into a file

Once you have edited one of these files, they will be read each time a shell session is started. In order to force the current shell session to read the file immediately we can use

```
$ source ~/.bashrc
```

If we want to set system-wide variables, it is usually better to add them to /etc/profile, /etc/bash.bashrc, or /etc/environment as appropriate.

Another useful command that can be used in the bash startup scripts is the *alias* command.

alias

Adding aliases

alias allows us to create a different name for a command or a command and options that will be used when we type our simple command name. For example, it can often be useful to add an alias to the remove command, *rm* , that will ensure that interactive mode, *rm –i* , is used so that you always need to confirm any deletions.

If we add the following line to one of our startup scripts then anytime we type *rm* it will use *rm -i*

```
alias rm=”rm –i”
```

We can use the existing name of a command and change its behavior, or we can use a completely random name to represent a command string.

Although not something I would recommend, *alias* can also be used to make the switch from MS-DOS, if you have experience of that, to Linux with such aliases as

```
alias dir="ls"

alias rename="mv"

alias md="mkdir"

alias copy="cp"

alias del="rm -i"
```

and others that you decide can help you personally .

Removing Aliases

The command *unalias* can be used to remove aliases that have been created. For example if aliases have been set in the global setup scripts you may want to remove them in your personal scripts. If the *alias* for *rm* that we used above had been set globally we could remove it in our personal startup scripts thus

```
unalias rm
```

This allows us greater freedom to customize how our shell session works both globally, and on a user specific level

Our final section will deal with Software Package Management.

Chapter 10 Installing Software and Package Management

Since it is often necessary to install new packages to enhance the functionality of your system we shall take a few moments for a brief overview of how we can perform this from the command line. This is more of a system administration task than many others mentioned in this text, but is an extremely important one.

There are several different programs and commands that can be used, depending upon the system that you are using and your goal. Since we are concentrating on Ubuntu in this guide we shall look closely at the apt suite of commands, but we shall also mention a few of the other programs that you may come across with different flavors of Linux such as Red Hat, Debian, etc.

Installing Packages on the Command Line

We shall begin by looking at package management, installation, removal, and updating on Ubuntu. Packages are software packages, collections of the appropriate files, libraries, dependencies, metadata, etc. required for the installation of a piece of software or utility. We shall make use of a dummy package named JuliansPackage

Ubuntu

apt

The *apt* family is the command line tool for package management, providing a command line interface for system administration of packages. We shall look at *apt* , *apt-get* , and *apt-cache* and some of the common uses and basic commands that are available. Several front-end, graphical interfaces exist that make use of the apt family, such as aptitude and synaptic. The *apt-get* and *apt-cache* commands available are considered lower level command line options.

We can perform several system wide and package specific tasks, similar to *apt-get* , which we shall look at more fully shortly. *apt* works like a front end to *apt-get* and *apt-cache.*

Importantly we must ensure that we have *root* privileges in order to make use of these system administration commands .

So, on with our brief look at *apt* . If we want to update the APT database of packages, we can use

```
$ apt update
```

To upgrade all the installed packages on the system

```
$ apt upgrade
```

To obtain a list of available packages

```
$ apt list
```

To search the APT database of packages we can use

```
$ apt search Julian
```

This search makes use of regular expresions (using apt cache at its heart) and should reveal the package JuliansPackage.

For dealing with individual packages we can make use of the following commands.

To install packages by name

```
$ apt install JuliansPackage
```

And to remove a package

```
$ apt remove JuliansPackag e
```

Now, after a brief introduction to apt we shall look at apt-get and apt-cache more fully, due to their power and usefulness.

apt-get

apt-get is a low-level command line tool for package management and is often considered a back-end to other tools that make use of the APT library, such as aptitude, and even *apt* as a command line only tool.

There are several *apt-get* commands that apply system wide rather than being used on a single package. They allow us to update the APT package database of available packages, install upgrades to installed packages, autoremove packages that are no longer required, and even perform a full distribution upgrade when a new distribution becomes available. We shall look first at the system wide commands.

In order to update the APT package lists from the system repositories we can use

```
$ apt-get updat e
```

Once we have updated the package lists, we can upgrade all the installed packages with

```
$ apt-get upgrade
```

This then brings our installed system fully up to date.

If we want to upgrade the entire distribution to a different version, such as when a new distribution has been released, we can use

```
$ apt-get dist-upgrade
```

To autoremove any packages that are no longer required such as libraries that

To autoremove any packages that are no longer required such as libraries that were only used by one package that you have removed

```
$ apt-get autoremove
```

These deal with the system. The following commands deal with individual packages, or several named packages

To install a package, that we have called here JuliansPackage we use

```
$ apt-get install JuliansPackag e
```

The command above will require confirmation to continue. We can force the command to continue by adding the *–y* option

```
$ apt-get –y installs JuliansPackage
```

If we want to install more than one package at the same time, we can list them on the command line as follows.

```
$ apt-get install JuliansPackage JacksPackage
```

To remove a package, we use

```
$ apt-get remove JuliansPackage
```

There are also several other available commands for advanced usage, such as purge, source (to retrieve source code files) amongst others. Check man *apt-get* for a more complete list of available commands and options.

apt cache

apt cache allows a variety of options to be performed on the APT package cache. These operations are not manipulative of system state but allow the facility to search for packages and generate valuable output from the package metadata. One incredibly useful command, the *search* command, allows searching of the APT package database which we use with a regular expression search term

```
$ apt-cache search Julian
```

This allows us to retrieve the correct name of a package that we may want to install but are unsure of the name that we require for our installation command. The search above would reveal our package JuliansPackage.

Other Flavours of Linux and Package Managers

It is important to appreciate that there are alternatives depending on the Linux distribution such as *yum* , *rpm* , *dpkg*

dpkg

dpkg is the software used to install, build, remove and manage Debian, .deb, packages. *dpkg* can be used on the command line and its behaviour is controlled by commands that consist of an action and usually a parameter list .

There is also a graphical front end for *dpkg* , aptitude, which can be a more user-friendly way to access the commands and manage packages, for those that prefer to work that way. Since this book is about the command line, we shall not look at aptitude.

We shall just mention a few basic commands for the most common purpose, since Ubuntu is heavily based on Debian.

To list all the installed packages on your system use

```
$ dpkg -l
```

To install a deb package, we can use *dpkg* as follows

```
$ dpkg –i JuliansPackage.deb
```

And to remove a deb package we use the following. Note that the name that we use does not include the .deb extension.

```
$ dpkg –r JuliansPackage
```

So there are a few basic commands for using dpkg which is something that you may find that you need to do even if you are used to using an Ububtu system and the apt commands, as some packages may only be available as .deb packages.

Yum

Yum, (Yellowdog Updater, Modified), is the primary command line tool for managing Red Hat Linux RPM packages having taken over from the *rpm* command, seen below. *yum* can be used for all package management tasks such as getting, installing, deleting and querying. We shall look at a couple of the most common use of *yum* .

To install an *rpm* package using *yum* we user

```
$ yum install JuliansPackage.rpm
```

This will perform dependency checks and ask us whether we wish to continue, which we will need to confirm. We can avoid that using the *–y* option

```
$ yum –y installs JuliansPackage.rpm
```

Likewise, in order to remove a package, first with confirmation being required, then without, we use

```
$ yum remove JuliansPackage.rp m
```

```
$ yum –y removes JuliansPackage.rpm
```

We can update a package to the latest version uusing

```
$ yum update JuliansPackage.rpm
```

These are a few of the more basic commands. Check out *man yum* if you are using a Red Hat based system that makes use of *yum* as a package manager.

rpm

The RPM Package Manager, RPM, can be used for installing software on Unix-like systems that use the rpm package format, particularly Red Hat Linux. Packages consist of an archive of files, and specific package information, including name, version, and descriedqzption

First it is necessary to execute the following commands as root, either using the sudo command, or by changing to the root user.

To install a package, that we shall here call JuliansPackage.rpm, we use

```
$ rpm –i JuliansPackage.rp m
```

and to upgrade it from an earlier version we use

```
$ rpm –U JuliansPackage.rpm
```

In order to remove an installed package, we use

```
$ rpm –ev JuliansPackage.rpm
```

As usual the best way to familiarise oneself with this command is to use *man rpm.*

So that is a very brief introduction to a variety of package management techniques and command line utilities available on different flavours of Linux. We shall complete our brief tour of some of the simpler aspects of the Linux Command Line Interpreter with a few useful command line tools.

Chapter 11 Testing Network Connectivity

Sometimes your networking may not work as expected and you will need to troubleshoot the problem by testing the network connectivity.

The following commands will be covered in this section:

ping

The *ping* command is a basic troubleshooting command that can be used to verify network connectivity.

The following subsections will be covered in this section:

Loopback

If you *ping* the loopback address 127.0.0.1 it verifies that the *networking service* is working. In the example below the ping command is successful. Each line *64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.073 ms* indicates a successful ping. The *ping* command will continue until you press *Ctrl+C* .

In the example below the *ping* to the loopback address is unsuccessful as the *network service* is not running.

Default Gateway

On a network the *default gateway* is the router or device that leads out of the network usually eventually to the Internet. If your system cannot communicate with the *default gateway*, then it cannot access the Internet. The *default gateway* is normally the first useable IP address in a network but that is not a requirement. For example, if the network is 10.0.2.0/24 then the *default gateway* would be 10.0.2.1. In the example below the ping to 10.0.2.1 was successful meaning the system can communicate with the *default gateway* .

If you are unable to ping the *default gateway* it does not mean that there is anything wrong with your system as it could be a problem with the network.

IP

If you can ping the *default gateway* the next step is to ping something past the *default gateway* , preferably something on the Internet. In the example below, a ping to *8.8.8.8* is successful which means that everything is working, and the system can reach the Internet. The ping command used the *-c 3* option which cause the ping command to only do three pings and then stop vice running until pressing *Ctrl+C* .

Hostname

While pinging an IP address on the Internet will show you if your network connectivity is working it does not give you the entire picture. In order to be able to access web pages and other services by using a hostname requires DNS to be configured and working correctly.

traceroute

The *traceroute* command is used to trace the path of the IP packets from your system to another system. In the example below a the *traceroute* command is used to trace packets to *www.somewebsite.com* .

5.4 Name Resolution

Name Resolution is the process of converting hostnames to IP addresses. The way the Internet works today we mostly use hostnames and domains to connect to websites and other resources. Your system, however, needs to know the IP address of where you want to go in order to get there.

/etc/hosts

When the Internet was very young, you had to know the IP address of any site that you wanted to access. Then the *hosts* file was created and shared among the users of the Internet which worked because there were not many people on the Internet. The *hosts* file holds the manual mapping of hostnames to IP address. With the advent of DNS, the *hosts* file was no longer necessary but it still remains in Linux as a fallback *Name Resolution* method.

The following subsections are covered in this section:

Ubuntu

The example below shows the default *Ubuntu hosts* file.

If you want to add a custom IP to hostname mapping you must edit the */etc/hosts* file. The following is the syntax for entries:

ip_address hostname

Fedora

The example below shows the default *Fedora* hosts file.

If you want to add a custom IP to hostname mapping you must edit the */etc/hosts* file. The following is the syntax for entries:

ip_address hostname

Debian

The example below shows the default *Debian* hosts file.

If you want to add a custom IP to hostname mapping you must edit the */etc/hosts* file. The following is the syntax for entries:

ip_address hostname

DNS

The *Domain Name System (DNS)* is a system for naming systems on the Internet based on resource locators, hostnames, and domains and providing a mechanism to rapidly distribute updates when things change.

When you use a web browser to connect to a website you type if something like www.somewebsite.com which is known as a *Universal Resource Locator (URL)* . Starting from the right and moving left the *URL* starts with *.com* which is a *Top-Level Domain (TDL)* . There are lots of *TDLs* , .com, .net, .org, .mil, and each *TDL* has an organization that controls the *TDL* . The next part of the *URL* is the domain name, *somewebsite* . When someone registers a domain name, the domain name is registered with the appropriate *TDL* controlling authority which which will in turn advertise DNS information about the domain. The last part of the *URL* is the resource locator *www* which is the resource locator for a web page.

The following commands will be covered in this section:

nslookup

di g

nslookup

The *nslookup* command can be used to query the DNS server for information about domains.

The basic use of the *nslookup* command is from the command line as in the example below.

The output lists the IP address of the DNS server. The line *Non-authoritative answer* means that the answer is not from the main DNS server responsible for the domain but rather another DNS server that knows the answer. The last part of the output is the name you were looking for and the IP address.

If you type the command *nslookup* without putting in a domain name you will enter interactive mode. In interactive mode you will have a > character that means *nslookup* is awaiting your input.

The *server* option in interactive mode will show you which DNS server you are getting name resolution from. You can also use the *server* option to change the DNS server to query. This will not change the server which is used for normal name resolution. In the example below the DNS server is changed to *10.8.4.3* .

The *set type* option allows you to change the type of DNS record to look for. By default, *nslookup* shows *a* records which are resource records. In the example below, the *type* is set to *ns* which is for nameservers or DNS servers. The example below shows four nameservers for the domain *somewebsite.com* .

In the example below, the *type* is set to *mx* which is for mail exchangers or email servers. The example below shows five email servers for the domain *somewebsite.com* .

In the example below, the type is set to *a* which is for resource records. The example below shows the *a* record for www.somewebsite.com.

In the example below, the type is set to *aaaa* which is for IPv6 resource records. The example below shows the *aaaa* record for www.somewebsite.com.

To exit *nslookup* interactive mode, type *exit* .

dig

The *dig* is another command that can be used to query the DNS for information about domains .

The basic use of the *dig* command is from the command line as in the example below. The output lists the IP address of *www.somewebsite.com* . The flag line *AUTHORITY: 0* means that the answer is non-authoritative.

The *dig* command by default looks for *a* record. The *dig* command can also be used to lookup other types of records by using the *-t* option followed by the type of record, *ns* , *mx* , *a* , or *aaaa* .

route

The *route* command is used to view the network routes that your system knows about. On most systems with only one NIC there is not much to do with routes. The most important route that your system has is a route to the *default gateway* which is the route to to Internet.

If your system has multiple NICs then then will be multiple routes. It is important to make sure that *default gateway* is pointed in the right direction. The syntax to add a default gate is as follows:

route add default gw *ip_address* dev *interface_name*

In the example below the *route* command is used to add *192.168.56.1* as the *default gateway* which can be reached via the *enp2s0* interface.

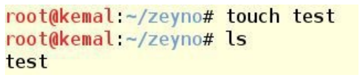
Chapter 12 File Operations

In this chapter, we will talk about creating a file, copy, move, view the contents, command output file to divert etc. Here we will work files will be usually text files.

Our subject order will be as follows: more than a categorical order, the place where the samples will be given on the use of commands when they are necessary, so the subject will provide more efficient processing. I think it would be a better narrative style.

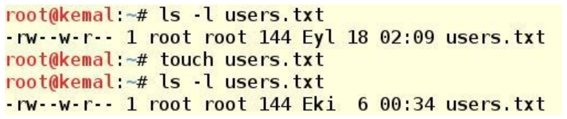
touch

One of the easiest ways to create a text file is to use the touch command. For example, with the *touch test* command, we can create a text file named as test. (you can give the name *test.txt* if you want)



```
root@kemal:~/zeyno# touch test
root@kemal:~/zeyno# ls
test
```

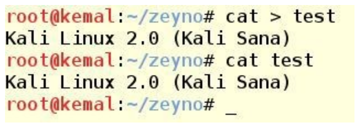
This command is not only to create a text file, you can also use it to update the existing file's date stamp.



```
root@kemal:~# ls -l users.txt
-rw--w-r-- 1 root root 144 Eyl 18 02:09 users.txt
root@kemal:~# touch users.txt
root@kemal:~# ls -l users.txt
-rw--w-r-- 1 root root 144 Eki  6 00:34 users.txt
```

cat

The "cat " command which is to use to read the file contents from the terminal screen, also used to write a in a text file. For example, to write into an empty *test* file, lets type *cat > test* command (note that the use of the > operator) and typing the phrase that we want and end out text with *CTRL + D* . Thus, what we wrote at the console will be registered into our file. To read the contents of the file we can use *cat test* command.

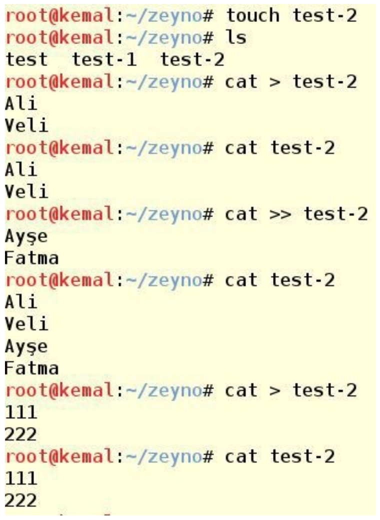


```
root@kemal:~/zeyno# cat > test
Kali Linux 2.0 (Kali Sana)
root@kemal:~/zeyno# cat test
Kali Linux 2.0 (Kali Sana)
root@kemal:~/zeyno# _
```

We wrote our text into an existing file in the above example. Even if there was no file named test, our command would be created and saved into a file created phrases specified file.

If you check the above example again, we use the operator > as we call *routing operator* . This operator is used when forwarding the files to be made. This operator can be used in many ways. Given expressions, command outputs, etc. as can be redirected to a file, it can also be taken the contents of a file as an argument.

We will give examples about these later. Right now, only we'll talk about > and >> operators. To write anything into an existing text file, the file must be blank. If the file is not empty, it will delete the old content and new content will be written. Of course, this is the case when using the > operator. If we use >> operator, without deleting the contents of the file, we enter our phrase is added from the last line in the file. Let us explain what we mean with some examples.

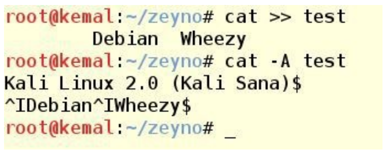


```
root@kemal:~/zeyno# touch test-2
root@kemal:~/zeyno# ls
test  test-1  test-2
root@kemal:~/zeyno# cat > test-2
Ali
Veli
root@kemal:~/zeyno# cat test-2
Ali
Veli
root@kemal:~/zeyno# cat >> test-2
Ayşe
Fatma
root@kemal:~/zeyno# cat test-2
Ali
Veli
Ayşe
Fatma
root@kemal:~/zeyno# cat > test-2
111
222
root@kemal:~/zeyno# cat test-2
111
222
```

Let us explain what we are doing:

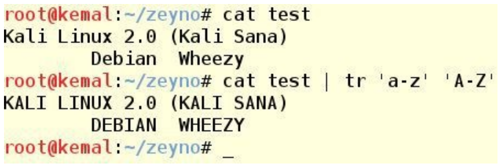
- *touch test-2* we create a new text file.
- *cat > test-2* We gave the command, and then type the expressions we want, and we finished with Ctrl + D.
- *cat test-2* we look at the file content.
- *cat >> test-2* We've written some new expressions and We finished with the file CTRL + D.
- *cat test-2* We saw what we type last is added to the file content.
- *cat > test-2* (Please note that we use > only once) we add our new content in our file.
- *cat test-2* When we checked again with the content of our files, we found that the old content is deleted and the content we enter the final written.

We learn that we can read the file content with cat command. İf we use *cat* command with -*A* the tab in the text shows spaces with ^|, and shows end of the line with *$* .



```
root@kemal:~/zeyno# cat >> test
        Debian  Wheezy
root@kemal:~/zeyno# cat -A test
Kali Linux 2.0 (Kali Sana)$
^IDebian^IWheezy$
root@kemal:~/zeyno# _
```

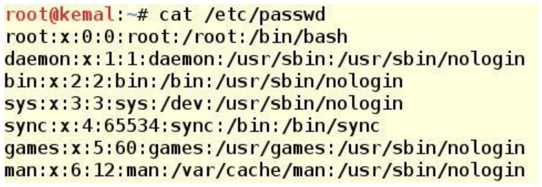
To change lowercase letters to uppercase letters in a text file that we want to see the contents or to do the opposite, we can use cat and tr command together. For example, to change lowercase letters to uppercase letters in test file, we can use cat test | tr 'a-z' 'A-Z'.



```
root@kemal:~/zeyno# cat test
Kali Linux 2.0 (Kali Sana)
        Debian  Wheezy
root@kemal:~/zeyno# cat test | tr 'a-z' 'A-Z'
KALI LINUX 2.0 (KALI SANA)
        DEBIAN  WHEEZY
root@kemal:~/zeyno# _
```

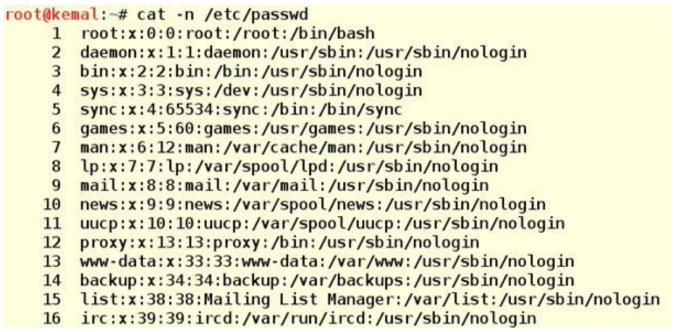
Here, I want to talk briefly about that we will use again in time, about | sign in and *piping* (pipe). As seen in the example above, with the pipe transaction an output command is given as input command. When we examine the example, by reading the contents of *test* file is given as input to *tr* command and tr command in accordance with the parameters used in the file content with its trade. Piping process is made by "|". Yes, let us keep to the examples again .

We talked about that we can read the contents of the text file with cat command to the terminal. Now, for example, let's look at the *passwd* file contents with *cat /etc/passwd* command



```
root@kemal:~# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

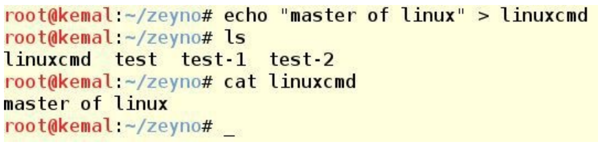
The file content shown as it is. If we want to see the content with the row numbers, we must use cat -n /etc/passwd command.



```
root@kemal:~# cat -n /etc/passwd
     1  root:x:0:0:root:/root:/bin/bash
     2  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
     3  bin:x:2:2:bin:/bin:/usr/sbin/nologin
     4  sys:x:3:3:sys:/dev:/usr/sbin/nologin
     5  sync:x:4:65534:sync:/bin:/bin/sync
     6  games:x:5:60:games:/usr/games:/usr/sbin/nologin
     7  man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
     8  lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
     9  mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
    10  news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
    11  uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
    12  proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
    13  www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
    14  backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
    15  list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
    16  irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
```

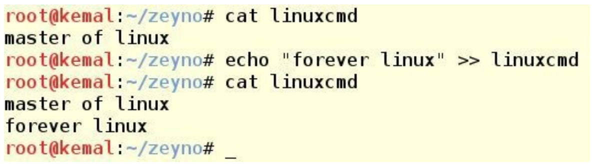
echo

Normally with this command, any desired statements sent as output to the terminal screen. Besides this it can be used to create a file and write any string into it. Now let's create a new file and expression as well as we want to let you write into the file using the echo command.



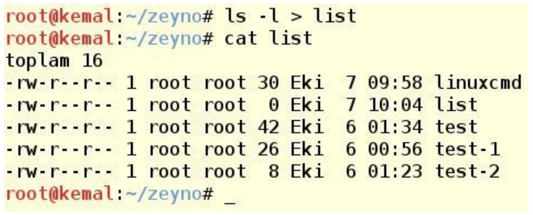
```
root@kemal:~/zeyno# echo "master of linux" > linuxcmd
root@kemal:~/zeyno# ls
linuxcmd  test  test-1  test-2
root@kemal:~/zeyno# cat linuxcmd
master of linux
root@kemal:~/zeyno# _
```

As you see with echo "master of the linux" > linuxcmd command file, we created *linuxcmd* command that did not exist before and wrote into *master of linux* . Previously existed an empty file can be written in this way. But if the file is not empty, then we can use >> operator as we did in the *cat* command before



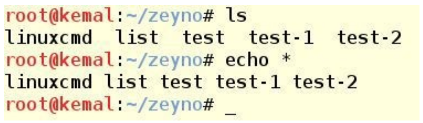
```
root@kemal:~/zeyno# cat linuxcmd
master of linux
root@kemal:~/zeyno# echo "forever linux" >> linuxcmd
root@kemal:~/zeyno# cat linuxcmd
master of linux
forever linux
root@kemal:~/zeyno# _
```

I said using the operators command output redirection could lead to a file. By the way, let's also do a small example of this. For example, lets direct ls -l command output to *list* file name that we create. Which means the contents of the command output is written into the file we named *list* . To do this, use the *ls -l > list* command is enough.



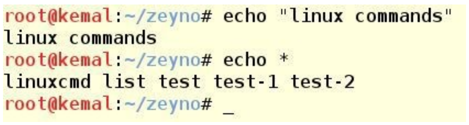
```
root@kemal:~/zeyno# ls -l > list
root@kemal:~/zeyno# cat list
toplam 16
-rw-r--r-- 1 root root 30 Eki  7 09:58 linuxcmd
-rw-r--r-- 1 root root  0 Eki  7 10:04 list
-rw-r--r-- 1 root root 42 Eki  6 01:34 test
-rw-r--r-- 1 root root 26 Eki  6 00:56 test-1
-rw-r--r-- 1 root root  8 Eki  6 01:23 test-2
root@kemal:~/zeyno# _
```

Now let's give an example about how we can use echo command in another way and have additional information. First, let's look at the contents of the directory with the *ls* command that we were in, and then use the *echo ** command, let's check if the output is the same in both commands.



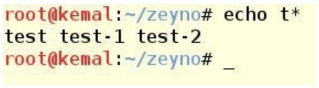
```
root@kemal:~/zeyno# ls
linuxcmd  list  test  test-1  test-2
root@kemal:~/zeyno# echo *
linuxcmd list test test-1 test-2
root@kemal:~/zeyno# _
```

As you can see the output of the two commands are the same. The reason is that, before starting the echo command shell, with * character and in */zeyno* directory all file index names to the command line get in place and give them as each to *echo* parameters. This means, with the fact that *echo ** command screen *asterix (*)* character was supposed to pop up, but it was not. So that means, asterix character is not sent as a parameter to echo.



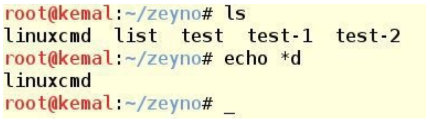
```
root@kemal:~/zeyno# echo "linux commands"
linux commands
root@kemal:~/zeyno# echo *
linuxcmd list test test-1 test-2
root@kemal:~/zeyno# _
```

After this little reminder, let us continue with our other examples. To print files names starting with t capital from the directory we are in, we can use the echo t* command.



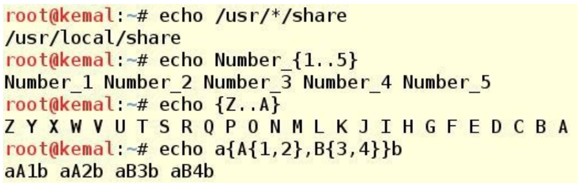
```
root@kemal:~/zeyno# echo t*
test test-1 test-2
root@kemal:~/zeyno# _
```

If you want to list the files with the last letter of d in the names, this time you can use the echo *d command.



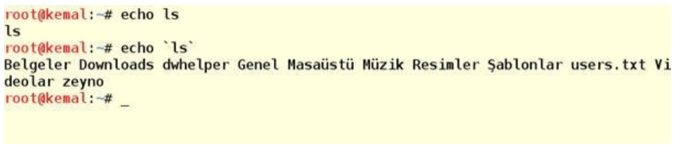
```
root@kemal:~/zeyno# ls
linuxcmd  list  test  test-1  test-2
root@kemal:~/zeyno# echo *d
linuxcmd
root@kemal:~/zeyno# _
```

By examining the following example, find out why that is. As your homework ☺



```
root@kemal:~# echo /usr/*/share
/usr/local/share
root@kemal:~# echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
root@kemal:~# echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
root@kemal:~# echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

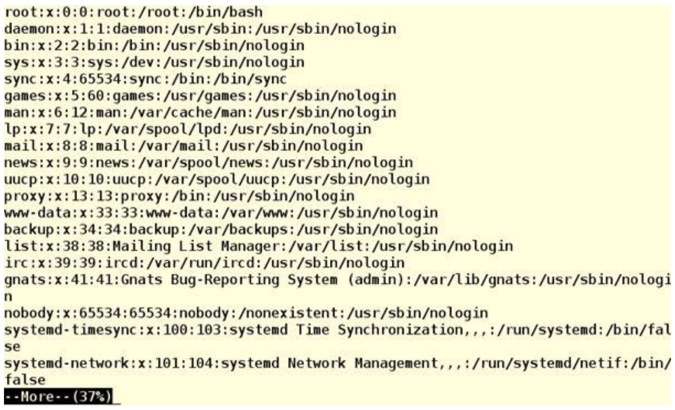
When using the terminal command, if there is a phrase in this statement between reverse quotes (` ) this perceived as a *command* by the shell. As an example, when *echo ls* command is given, naturally *ls* expression (*string* ) is printed on the screen. However, if you use the command as *echo `ls`* form, this time the *Is* that sent to the terminal will be perceived as command by the *shell* and rotating the command to print the expression on the display command output. (i.e. all strings in the output of the *ls* command will be given as a parameter to *echo* command, and therefore these statements to be printed to the screen as well)



```
root@kemal:~# echo ls
ls
root@kemal:~# echo `ls`
Belgeler Downloads dwhelper Genel Masaüstü Müzik Resimler Şablonlar users.txt Vi
deolar zeyno
root@kemal:~# _
```

more

To read text files from the terminal one of the commands that we can use is *the more* command. This command is used where command is longer than the output, for our more comfort to read. For example, when we give *more /etc/passwd* command, the longer command output appears on the screen and each time you press the *enter* key it will proceed line by line. With pressing *space* key is to scroll page by page. To return to the previous page, the *b* key is used. When we want to exit, we can press to *q* key.



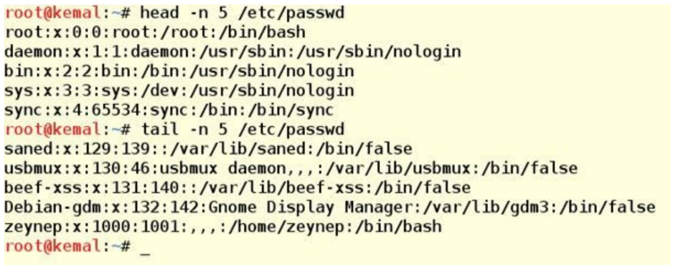
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
n
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/fal
se
systemd-network:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/
false
--More--(37%)
```

less

Once again, to read the contents of a file, another command you can use is less command. With this command as in the *more* command we display output with long files to easy to read. Using the up and down arrow keys to scroll through the pages. With *space* key, goes to next page, with *b* key goes to the previous page. To exit *q* key is used as same we mentioned before.

head-tail

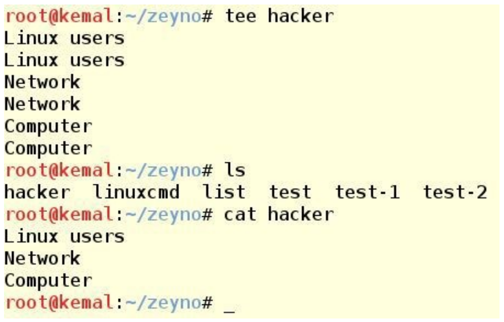
While viewing the file contents to see the first ten rows head , to view the last ten lines *tail* command is used. The use of these parameters is the case but if you want to see a certain number of rows you must specify them with *-n* parameter. For example, to see the first five lines *head -n 5* , to see the last five lines *tail -n 5* command can be used.



```
root@kemal:~# head -n 5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
root@kemal:~# tail -n 5 /etc/passwd
saned:x:129:139::/var/lib/saned:/bin/false
usbmux:x:130:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
beef-xss:x:131:140::/var/lib/beef-xss:/bin/false
Debian-gdm:x:132:142:Gnome Display Manager:/var/lib/gdm3:/bin/false
zeynep:x:1000:1001:,,,:/home/zeynep:/bin/bash
root@kemal:~# _
```

tee

To print expressions both standard output (ie, terminal/console) and you want to write to a file specified as well, you must use the tee command. If the file does not exist, it will be created and written into. When the writing is finished, exit *Ctrl + D* so you return to the command line.

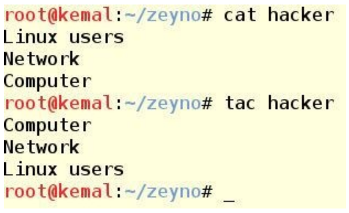


```
root@kemal:~/zeyno# tee hacker
Linux users
Linux users
Network
Network
Computer
Computer
root@kemal:~/zeyno# ls
hacker  linuxcmd  list  test  test-1  test-2
root@kemal:~/zeyno# cat hacker
Linux users
Network
Computer
root@kemal:~/zeyno# _
```

As you can see, first it created the file named hacker with *tee hacker* command, then gave out put we wrote expressions from the terminal as well as wrote to files.

tac

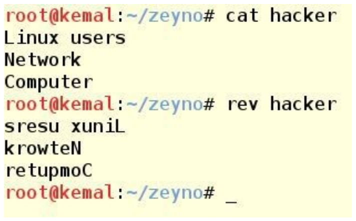
To view toward the beginning of the end of the contents of a text file, we can use the tac command.



```
root@kemal:~/zeyno# cat hacker
Linux users
Network
Computer
root@kemal:~/zeyno# tac hacker
Computer
Network
Linux users
root@kemal:~/zeyno# _
```

rev

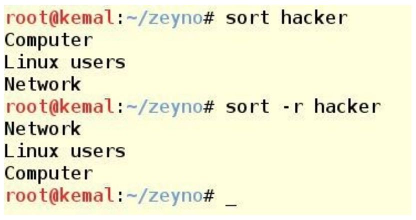
As an interesting command is rev command, you can display each line on the screen reversed.



```
root@kemal:~/zeyno# cat hacker
Linux users
Network
Computer
root@kemal:~/zeyno# rev hacker
sresu xuniL
krowteN
retupmoC
root@kemal:~/zeyno# _
```

sort

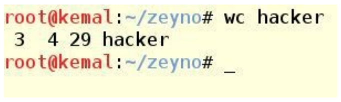
To reflect the Text file/document output to terminal in alphabetical order, the sort command is used. If it used as sort -r, reflects in reverse alphabetical order.



```
root@kemal:~/zeyno# sort hacker
Computer
Linux users
Network
root@kemal:~/zeyno# sort -r hacker
Network
Linux users
Computer
root@kemal:~/zeyno# _
```

wc

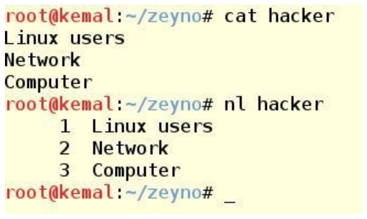
In a file a line, character or the number of words you're wondering, you can use wc (wordcount) command. With -*l* the line, with -*c* the character and with -*w* and parameter, the number of words can be learned. If you use without parameters as *wc* , it shows all.



```
root@kemal:~/zeyno# wc hacker
 3  4 29 hacker
root@kemal:~/zeyno# _
```

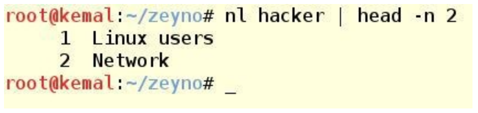
nl

Previously, numbering the printing process we did it with cat -n lines, we can also do it with *nl* command. For example, in the context of *hacker* file to enumerate rows, *nl hacker* command is used.



```
root@kemal:~/zeyno# cat hacker
Linux users
Network
Computer
root@kemal:~/zeyno# nl hacker
     1  Linux users
     2  Network
     3  Computer
root@kemal:~/zeyno# _
```

Content with a text file in a long line as enumerate as well as to see a few lines at the beginning, nl [TextFile] | head -n [number] command can be used

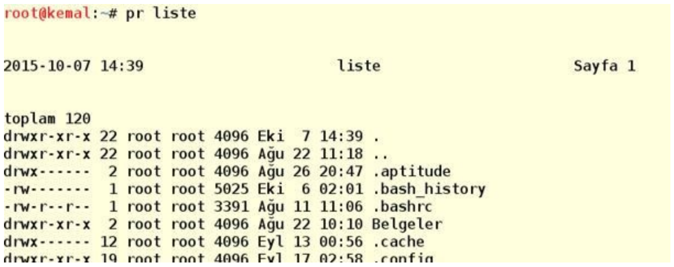


```
root@kemal:~/zeyno# nl hacker | head -n 2
     1  Linux users
     2  Network
root@kemal:~/zeyno# _
```

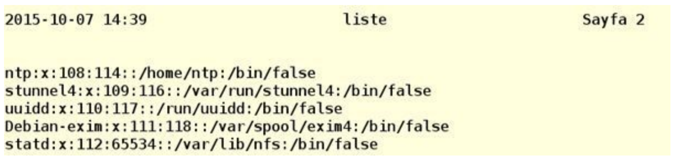
Similarly, to list a few lines from the end, tail command is used instead of head.

pr

To divide the contents of the page document, we use the pr command. For example, to view a file called *list*, divide and the content of our page, we can use *pr list* command. Therefore, if you print it, you will get a smoother output.



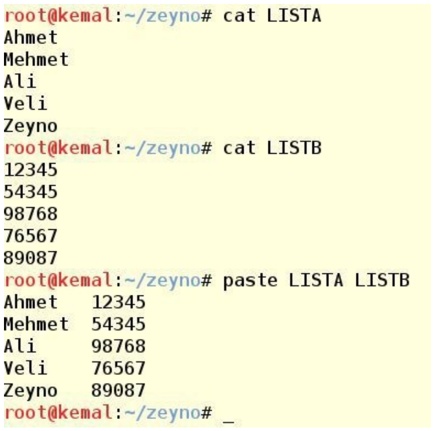
```
root@kemal:~# pr liste


2015-10-07 14:39                    liste                    Sayfa 1


toplam 120
drwxr-xr-x 22 root root 4096 Eki  7 14:39 .
drwxr-xr-x 22 root root 4096 Ağu 22 11:18 ..
drwx------  2 root root 4096 Ağu 26 20:47 .aptitude
-rw-------  1 root root 5025 Eki  6 02:01 .bash_history
-rw-r--r--  1 root root 3391 Ağu 11 11:06 .bashrc
drwxr-xr-x  2 root root 4096 Ağu 22 10:10 Belgeler
drwx------ 12 root root 4096 Eyl 13 00:56 .cache
drwxr-xr-x 19 root root 4096 Eyl 17 02:58 .config
```



```
2015-10-07 14:39                    liste                    Sayfa 2


ntp:x:108:114::/home/ntp:/bin/false
stunnel4:x:109:116::/var/run/stunnel4:/bin/false
uuidd:x:110:117::/run/uuidd:/bin/false
Debian-exim:x:111:118::/var/spool/exim4:/bin/false
statd:x:112:65534::/var/lib/nfs:/bin/false
```

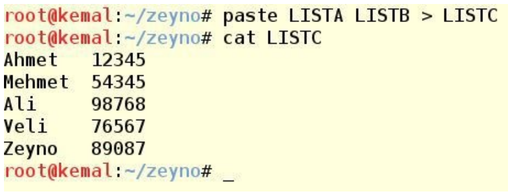
paste

To placing the base of the intermediate space between two or more file located opposite columns, paste command is use. For example, let’s have two files named as *LISTA* and *LISTB* . To list mutually the lines side by side contained in these files, *paste LISTA LISTB* command is used.



```
root@kemal:~/zeyno# cat LISTA
Ahmet
Mehmet
Ali
Veli
Zeyno
root@kemal:~/zeyno# cat LISTB
12345
54345
98768
76567
89087
root@kemal:~/zeyno# paste LISTA LISTB
Ahmet   12345
Mehmet  54345
Ali     98768
Veli    76567
Zeyno   89087
root@kemal:~/zeyno# _
```

If you want, you can combine the contents of these files and you can redirect to a new file. For example, when we use paste LIST LISTB > LISTC command, the next contents of two files side by side in mutual line and at the same time creating a new file named *LISTC* will be saved into the file.



```
root@kemal:~/zeyno# paste LISTA LISTB > LISTC
root@kemal:~/zeyno# cat LISTC
Ahmet   12345
Mehmet  54345
Ali     98768
Veli    76567
Zeyno   89087
root@kemal:~/zeyno# _
```

grep

To search for an expression (string) in files or output of a command, we can use *grep* command. Before the example of our command, let's use some of the important parameters.

• *-v:* Reverses the behavior of the command line that does not contain the sought word lists.

• *-i:* Upper / lower case is not taken during a search.

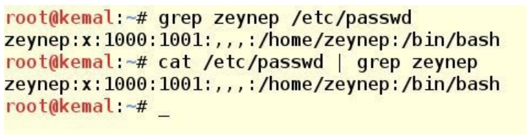
• *-r:* Searches in the subdirectories of a given directory.

• *-n:* It displays the line number where the search word now.

• *-c:* The specified index indicates the number of times the word you are looking for.

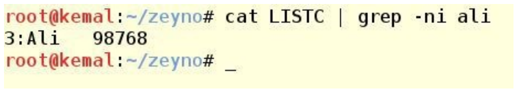
• *-l:* Where appropriate template column lists file names.

Let's make a simple example. For example, in /etc/passwd file, to list the lines includes *zeynep* , we can use either *grep /etc/passwd* or *cat /etc/passwd* | *grep zeynep* commands.



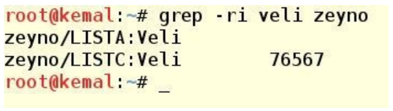
```
root@kemal:~# grep zeynep /etc/passwd
zeynep:x:1000:1001:,,,:/home/zeynep:/bin/bash
root@kemal:~# cat /etc/passwd | grep zeynep
zeynep:x:1000:1001:,,,:/home/zeynep:/bin/bash
root@kemal:~# _
```

Now Let's try cat LIST |grep -ni ali command. With this command: in *LISTC* file, upper / lower case without distinction, (-I) in *ali* expression lines, line number (*n*), we have also shown together.



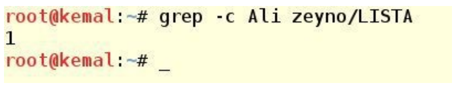
```
root@kemal:~/zeyno# cat LISTC | grep -ni ali
3:Ali   98768
root@kemal:~/zeyno# _
```

Another example let us examine grep -ri veli zeyno command. With this command, the *zeyno* directory as *recursive* (ie, including subdirectories), upper / lowercase distinction without parents' expression of the files, we ment list lines that have *veli* expression. (because *zeyno* directory in the same directory, I used *zeyno* instead of */zeyno*)



```
root@kemal:~# grep -ri veli zeyno
zeyno/LISTA:Veli
zeyno/LISTC:Veli        76567
root@kemal:~# _
```

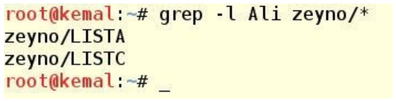
The following example, grep -c Ali zeyno /LIST command, the *LIST* file under the *zeyno* directory, we want to learn how many times *Ali* stated (with *-c* parameter)



```
root@kemal:~# grep -c Ali zeyno/LISTA
1
root@kemal:~# _
```

The last example that we have about this command, we listed the files containing

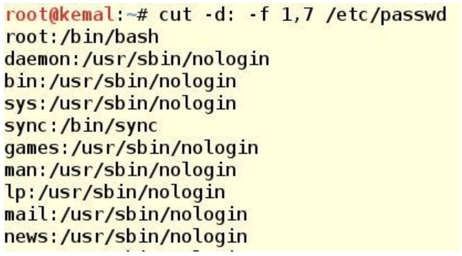
The last example that we have about this command, we listed the files containing the line through which we seek expression in the following example.



```
root@kemal:~# grep -l Ali zeyno/*
zeyno/LISTA
zeyno/LISTC
root@kemal:~# _
```

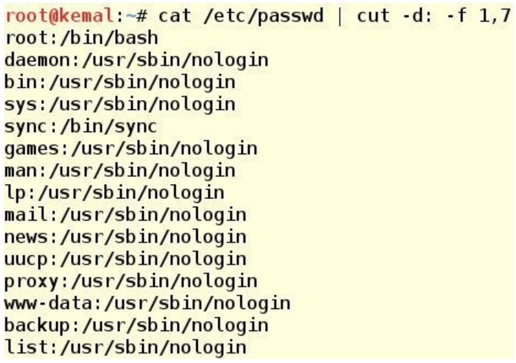
cut

In the file we want to examine, to see some lines on specific areas, we use cut command. For example, to list the 1st and 7th field with /etc/passwd line of the file, we can use cut -d: -f 1,7 /etc/passwd command.



```
root@kemal:~# cut -d: -f 1,7 /etc/passwd
root:/bin/bash
daemon:/usr/sbin/nologin
bin:/usr/sbin/nologin
sys:/usr/sbin/nologin
sync:/bin/sync
games:/usr/sbin/nologin
man:/usr/sbin/nologin
lp:/usr/sbin/nologin
mail:/usr/sbin/nologin
news:/usr/sbin/nologin
```

Here, we specified that we wanted to see the separation of areas with -d , the count of fields with *-f* . *pipe* operator we can use (|) command in a different way. For example, with *cat / etc / passwd | cut -d: -f 1,7* command, you can get the same result.

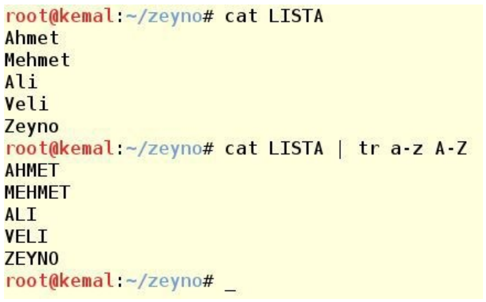


```
root@kemal:~# cat /etc/passwd | cut -d: -f 1,7
root:/bin/bash
daemon:/usr/sbin/nologin
bin:/usr/sbin/nologin
sys:/usr/sbin/nologin
sync:/bin/sync
games:/usr/sbin/nologin
man:/usr/sbin/nologin
lp:/usr/sbin/nologin
mail:/usr/sbin/nologin
news:/usr/sbin/nologin
uucp:/usr/sbin/nologin
proxy:/usr/sbin/nologin
www-data:/usr/sbin/nologin
backup:/usr/sbin/nologin
list:/usr/sbin/nologin
```

Because of the command output is long, I did not put all the visual image of the screen, in both visuals. If at the beginning or at the end, you want to list some of the specific number of rows, you can still use head and tail command. For example, like tail -n 5 /etc/passwd | cut -d: -f 1,7 or cut -d: -f 1,7 /etc/passwd | head -n 5

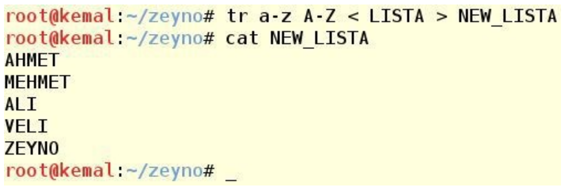
tr

Earlier, we mentioned briefly about the tr command, let's make a few examples about it. With this command, we can make some changes about the character of the files. These procuders are for change in lowercase letters or do the opposite and delete characters, etc. For example, the *LIST* text file, to translate lowercase letters to uppercase letters, *cat LISTA | tr a-z A-Z* can be used command.



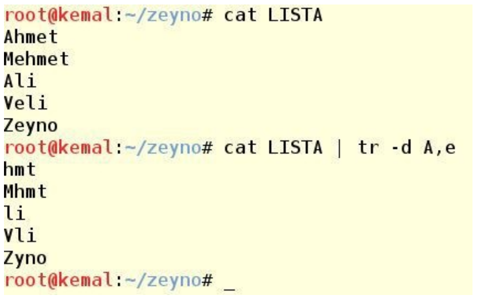
```
root@kemal:~/zeyno# cat LISTA
Ahmet
Mehmet
Ali
Veli
Zeyno
root@kemal:~/zeyno# cat LISTA | tr a-z A-Z
AHMET
MEHMET
ALI
VELI
ZEYNO
root@kemal:~/zeyno# _
```

We talked about routing the operator (> and >>). Other than that, we also have a < operator. These operators, unlike the others, take the file as *entry* . So, taking the contents of the file, enabling the inactivity associated with it. Now, let's use with *tr* command > and < operators together. For example, Let's use the command *tr a-z A-Z <LIST> NEW_LISTA* .



```
root@kemal:~/zeyno# tr a-z A-Z < LISTA > NEW_LISTA
root@kemal:~/zeyno# cat NEW_LISTA
AHMET
MEHMET
ALI
VELI
ZEYNO
root@kemal:~/zeyno# _
```

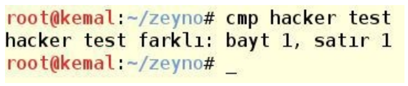
With our command, we got LIST file as input, created *NEW_LISTA* file by lowercase letters in uppercase in the text and we write in the file. If we use the *tr* command with *d* , the specified character will be ignored. For example, when we use *cat LISTA | tr -d A, e* command, *A* and *e* characters in *LISTA* file will be ignored.



```
root@kemal:~/zeyno# cat LISTA
Ahmet
Mehmet
Ali
Veli
Zeyno
root@kemal:~/zeyno# cat LISTA | tr -d A,e
hmt
Mhmt
li
Vli
Zyno
root@kemal:~/zeyno# _
```

cmp

One of the commands that we can use to make comparisons between files is cmp command. For example, I use *cmp hacker test* command to compare two files called *hacker* and *test* . In result, it will display the difference in files, the line of first difference occured, and as byte number.

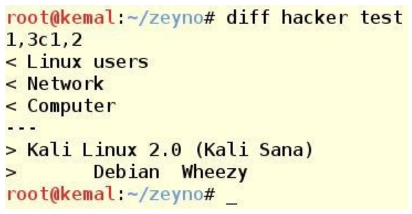


```
root@kemal:~/zeyno# cmp hacker test
hacker test farklı: bayt 1, satır 1
root@kemal:~/zeyno# _
```

There are a few parameters to use this command but you can experiment with them to examine man cm p . For example, when you use the *-l* parameter byte number for each of the differences, and different byte values to standard output (screen / terminal) will be sent.

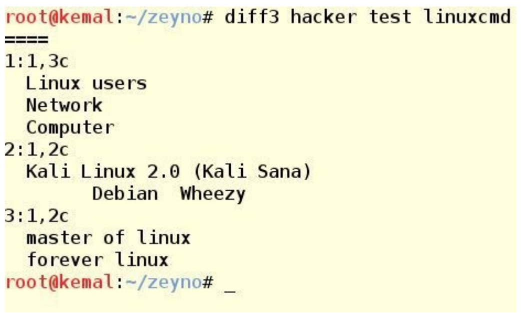
diff

Again, it is a command used to compare files. If a comparison between the two files diff , if the comparison between three files *diff3* command should be used. For example, to see again the difference between *hacker* and *test* files, *diff hacker test* command is used.



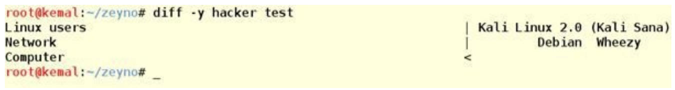
```
root@kemal:~/zeyno# diff hacker test
1,3c1,2
< Linux users
< Network
< Computer
---
> Kali Linux 2.0 (Kali Sana)
>       Debian  Wheezy
root@kemal:~/zeyno# _
```

Now, to compare two files diff3 hacker test linuxcmd command and see differences between *hacker* , *test* and *linuxcmd* .



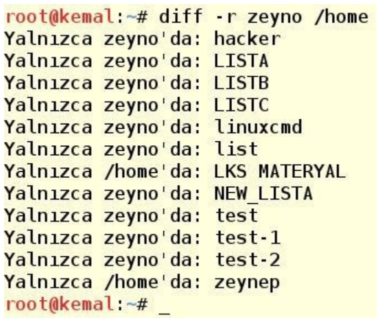
```
root@kemal:~/zeyno# diff3 hacker test linuxcmd
====
1:1,3c
  Linux users
  Network
  Computer
2:1,2c
  Kali Linux 2.0 (Kali Sana)
        Debian  Wheezy
3:1,2c
  master of linux
  forever linux
root@kemal:~/zeyno# _
```

If you use -y with *diff* command, the output will be projected on the screen side by side.



```
root@kemal:~/zeyno# diff -y hacker test
Linux users                                                   | Kali Linux 2.0 (Kali Sana)
Network                                                       |         Debian  Wheezy
Computer                                                      <
root@kemal:~/zeyno# _
```

Not only files, diff command can also be used to compare directories. For example, with *diff -r zeyno /home* command, *zeyno* and */home* directories *recursive* (*-r*) are compared and the differences will be listed.

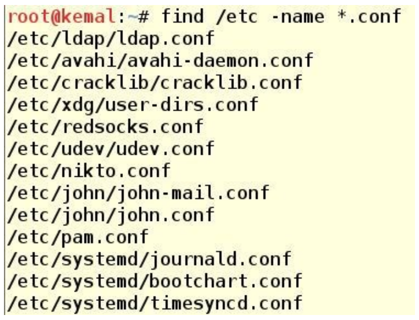


```
root@kemal:~# diff -r zeyno /home
Yalnızca zeyno'da: hacker
Yalnızca zeyno'da: LISTA
Yalnızca zeyno'da: LISTB
Yalnızca zeyno'da: LISTC
Yalnızca zeyno'da: linuxcmd
Yalnızca zeyno'da: list
Yalnızca /home'da: LKS MATERYAL
Yalnızca zeyno'da: NEW_LISTA
Yalnızca zeyno'da: test
Yalnızca zeyno'da: test-1
Yalnızca zeyno'da: test-2
Yalnızca /home'da: zeynep
root@kemal:~# _
```

find

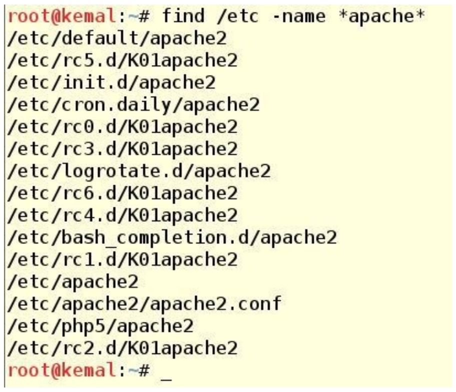
From time to time, you know only part of the name, but there will be files and directories that you do not remember which directory. To find these files and perform various actions related to the them, we use find command.

Using pattern is find [file_path] [search_phrase] . So, we type what we're gonna do it in the first directory search, then we define the search expression with *-name* . For example to list *conf* files under */etc* directory, we can use *find /etc -name *.conf* command.



```
root@kemal:~# find /etc -name *.conf
/etc/ldap/ldap.conf
/etc/avahi/avahi-daemon.conf
/etc/cracklib/cracklib.conf
/etc/xdg/user-dirs.conf
/etc/redsocks.conf
/etc/udev/udev.conf
/etc/nikto.conf
/etc/john/john-mail.conf
/etc/john/john.conf
/etc/pam.conf
/etc/systemd/journald.conf
/etc/systemd/bootchart.conf
/etc/systemd/timesyncd.conf
```

*Another example is with find / etc -name * apache ** again, we search files include apache word under */etc* directory.



```
root@kemal:~# find /etc -name *apache*
/etc/default/apache2
/etc/rc5.d/K01apache2
/etc/init.d/apache2
/etc/cron.daily/apache2
/etc/rc0.d/K01apache2
/etc/rc3.d/K01apache2
/etc/logrotate.d/apache2
/etc/rc6.d/K01apache2
/etc/rc4.d/K01apache2
/etc/bash_completion.d/apache2
/etc/rc1.d/K01apache2
/etc/apache2
/etc/apache2/apache2.conf
/etc/php5/apache2
/etc/rc2.d/K01apache2
root@kemal:~# _
```

We can use some parameters when making a search to facilitate our work and will help us get the results we want:

perm 755 (Access to the permission 755 file / directories)

-*type f* (Files)

-*type d* (Directories)

-*size +100k* (Files that are larger than 100 Kbytes)

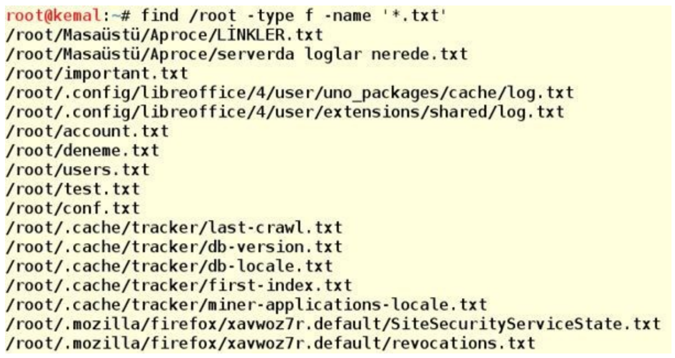
-*ctime 3* (Files / directories which amended exactly 3 days ago)

-*ctime -7* (The files/directories that have changed for a shorter time before 7 days)

-*ctime +7* (The files/directories that have changed for a longer time before 7 days)

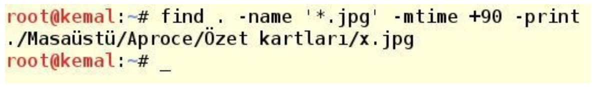
Let’s make some examples

For example, we want to list files that find /root -type f -name ' .txt'* command in the working directory and the extension *txt* . Working directory is the directory that already by that time we tried to / root instead of typing */root we can use* ". " (*point*) to use this command. Because point mark shows which directory we work on.



```
root@kemal:~# find /root -type f -name '*.txt'
/root/Masaüstü/Aproce/LİNKLER.txt
/root/Masaüstü/Aproce/serverda loglar nerede.txt
/root/important.txt
/root/.config/libreoffice/4/user/uno_packages/cache/log.txt
/root/.config/libreoffice/4/user/extensions/shared/log.txt
/root/account.txt
/root/deneme.txt
/root/users.txt
/root/test.txt
/root/conf.txt
/root/.cache/tracker/last-crawl.txt
/root/.cache/tracker/db-version.txt
/root/.cache/tracker/db-locale.txt
/root/.cache/tracker/first-index.txt
/root/.cache/tracker/miner-applications-locale.txt
/root/.mozilla/firefox/xavwoz7r.default/SiteSecurityServiceState.txt
/root/.mozilla/firefox/xavwoz7r.default/revocations.txt
```

For example with find . -name '.jpg' -mtime +90 -print* , we comand to list *jpg* files that modified 90 days.



```
root@kemal:~# find . -name '*.jpg' -mtime +90 -print
./Masaüstü/Aproce/Özet kartları/x.jpg
root@kemal:~# _
```

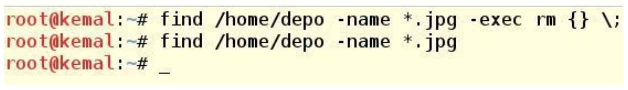
exec

In order to do various operations on files, use exec with find command.

*For example, with find /home/depo -name *.jpg* command find jpg files in *depo* folder. There are many things that fodler, therefore must be some photos ☺

Now, lets use find command together with the -*exec* and we want to delete *jpg* files it finds. To do this we must use *find /home/depo -name *.jpg -exec rm {}*

\; command. In command, files include {}are placed in these brackets as a parameter, and right after *-exec* the specified program is executed accordingly. Let's look at the results.



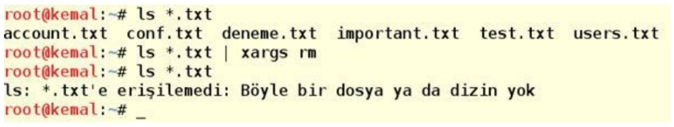
```
root@kemal:~# find /home/depo -name *.jpg -exec rm {} \;
root@kemal:~# find /home/depo -name *.jpg
root@kemal:~# _
```

As you can see because of jpg files are deleted, we cannot find them when we search with find.

xargs

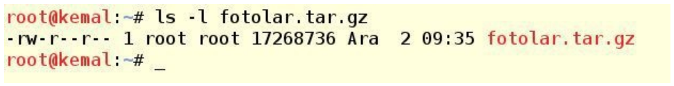
Another tool that we can use with find command is xargs tool. Xargs can be used either alone, or can be done similar job as -exec with find command. Xargs, gives the argument as input data given to it to the next program from the individual itself.

*A simple example: with ls *.txt | xargs rm* , we said "find files ending with .txt files in the working directory, and deleted them."



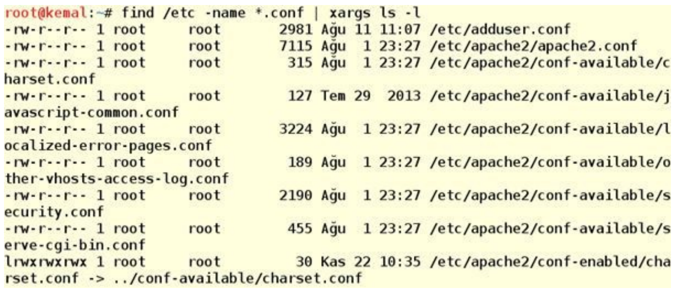
```
root@kemal:~# ls *.txt
account.txt  conf.txt  deneme.txt  important.txt  test.txt  users.txt
root@kemal:~# ls *.txt | xargs rm
root@kemal:~# ls *.txt
ls: *.txt'e erişilemedi: Böyle bir dosya ya da dizin yok
root@kemal:~# _
```

*Now let's combine find with xargs commands and use them. For example, with find / -name *.jpg -type f -print | xargs tar -cvzf fotolar.tar.gz* command, find all *jpg* files under the root directory, we want to make a compressed archive file with the name *fotolar.tar.gz* . Our command is working now and creating the archive file we want.



```
root@kemal:~# ls -l fotolar.tar.gz
-rw-r--r-- 1 root root 17268736 Ara  2 09:35 fotolar.tar.gz
root@kemal:~# _
```

Finally, lets give /etc -name "*.conf" | xargs ls -l *example and finish this section.*



```
root@kemal:~# find /etc -name *.conf | xargs ls -l
-rw-r--r-- 1 root     root       2981 Ağu 11 11:07 /etc/adduser.conf
-rw-r--r-- 1 root     root       7115 Ağu  1 23:27 /etc/apache2/apache2.conf
-rw-r--r-- 1 root     root        315 Ağu  1 23:27 /etc/apache2/conf-available/c
harset.conf
-rw-r--r-- 1 root     root        127 Tem 29  2013 /etc/apache2/conf-available/j
avascript-common.conf
-rw-r--r-- 1 root     root       3224 Ağu  1 23:27 /etc/apache2/conf-available/l
ocalized-error-pages.conf
-rw-r--r-- 1 root     root        189 Ağu  1 23:27 /etc/apache2/conf-available/o
ther-vhosts-access-log.conf
-rw-r--r-- 1 root     root       2190 Ağu  1 23:27 /etc/apache2/conf-available/s
ecurity.conf
-rw-r--r-- 1 root     root        455 Ağu  1 23:27 /etc/apache2/conf-available/s
erve-cgi-bin.conf
lrwxrwxrwx 1 root     root         30 Kas 22 10:35 /etc/apache2/conf-enabled/cha
rset.conf -> ../conf-available/charset.conf
```

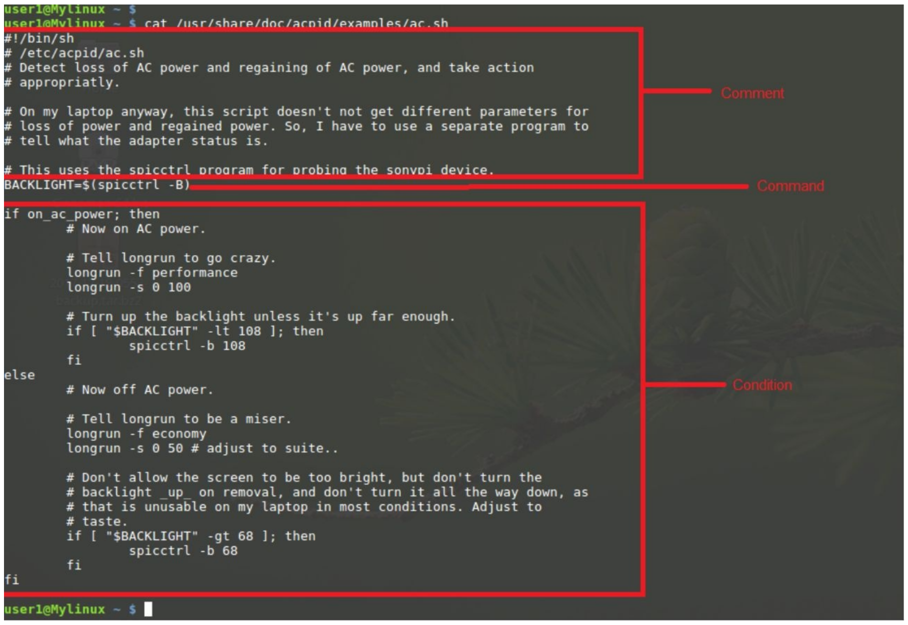
Chapter 13 Exploring Shell

Shell

A shell is a command-line interpreter. It is interface between user and Operating system. User gives command on Shell prompt and operating system executes that command. As shell is a command language interpreter, the basic purpose of shell is to translate human readable commands typed at a terminal into system actions. A shell is a special user program through which other programs are invoked. The shell gets started when the user logs in or start the terminal. Although there are several different shells like Bash, Korn, C shell etc, Bash is the default shell for Linux. Based on the shell you use there are minor differences in commands also.

Shell Script

Generally, we use shell in interactive mode, where user inputs command and gets output. However, some time you use shell scripts. Shell script is script you write to give series of command. Scripts that are more complex check conditions and run loops. Seasoned administrators use shell scripts to automate administrative tasks and to monitor system. Suppose, you want to check the disk space, memory availability and environment variables before invoking a program. There are two ways to do it, either you can give all commands manually every time before running that program or you can use shell script to do that. Shell scripts generally end with *sh* extension. Figure bellow shows an example of shell script

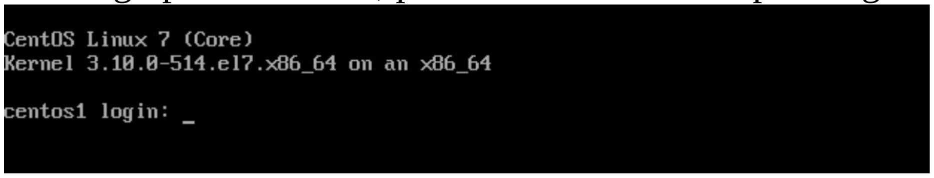


```
user1@Mylinux ~ $
user1@Mylinux ~ $ cat /usr/share/doc/acpid/examples/ac.sh
#!/bin/sh
# /etc/acpid/ac.sh
# Detect loss of AC power and regaining of AC power, and take action
# appropriatly.

# On my laptop anyway, this script doesn't not get different parameters for
# loss of power and regained power. So, I have to use a separate program to
# tell what the adapter status is.

# This uses the spicctrl program for probing the sonypi device.
BACKLIGHT=$(spicctrl -B)

if on_ac_power; then
        # Now on AC power.

        # Tell longrun to go crazy.
        longrun -f performance
        longrun -s 0 100

        # Turn up the backlight unless it's up far enough.
        if [ "$BACKLIGHT" -lt 108 ]; then
                spicctrl -b 108
        fi
else
        # Now off AC power.

        # Tell longrun to be a miser.
        longrun -f economy
        longrun -s 0 50 # adjust to suite..

        # Don't allow the screen to be too bright, but don't turn the
        # backlight _up_ on removal, and don't turn it all the way down, as
        # that is unusable on my laptop in most conditions. Adjust to
        # taste.
        if [ "$BACKLIGHT" -gt 68 ]; then
                spicctrl -b 68
        fi
fi
user1@Mylinux ~ $
```

Command line Interfac e

Command line interface (CLI) is way to access the shell. If you do CentOS installation without GUI or Minimal, you will get only command line interface through virtual terminals. When you install Linux with GUI you have two option to access CLI

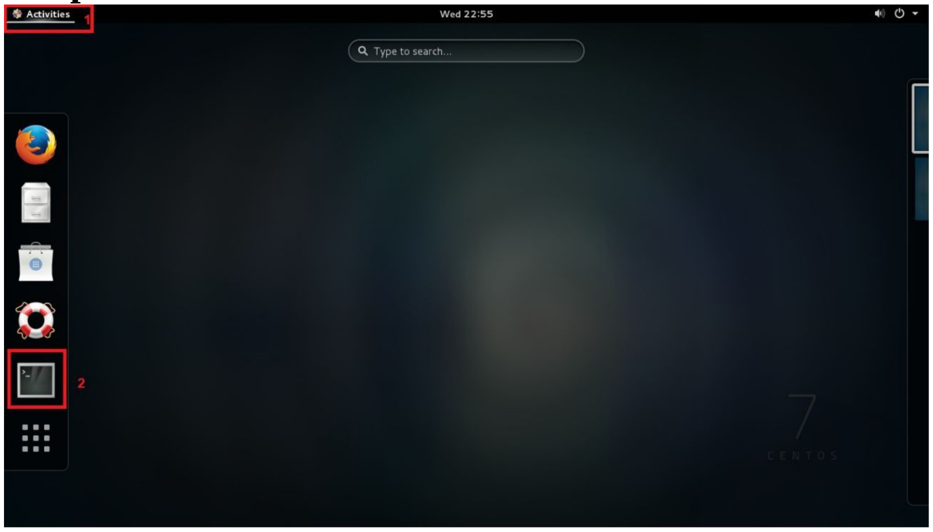
- Virtual Terminals
- Graphical Terminal

Virtual Terminals
Virtual Terminal is full screen command line interface. If you are working on CentOS server locally, you will get more than one Virtual terminal running on same server. Having more than one virtual terminal allows the administrator to switch to another terminal if necessary. Even if you have Graphical Desktop environment is installed you can still access virtual terminals. Virtual terminals can be accessed on CentOS System by pressing Ctrl+Alt+F2 till F6. To come back to the graphical session, press Ctrl+Alt+F1. On pressing Ctrl+Alt+F2 you will 

```
CentOS Linux 7 (Core)
Kernel 3.10.0-514.el7.x86_64 on an x86_64

centos1 login: _
```

get first Virtual terminal like that

Login and start issuing commands

Graphical Terminal Emulator


System with Graphical desktop environment can get CLI inside GUI with *Terminal Emulator*. Terminal allows you to open a window and lets you interact with the shell. Depending on Graphical Desktop environment installed on Linux distributions you have bunch of terminal emulator available like gnome-terminal, konsole, rxvt, xterm, LXterminal, guake, terminator, tilde, yakuake etc. Gnome-terminal is the default terminal emulator for CentOS Linux with GNOME desktop environment. To start Gnome terminal, press Activities and press terminal icon.

Command prompt

Although you can customize the prompt, but more often you will get either # or $ at end of prompt. Where # indicates the user is logged in as root (administrator) user and $ prompt as normal user. You can display additional information in the prompt like username, current directory and server name even you can show current date and time.

Structure of commands

Normally if you run command from the shell prompt will have the following format:

```
command –options <filename>
```

Example

```
tail –f myfil e
```

However, file name is not used in all commands. in some commands only command or command and options are used

```
ls –la
```

Tips for the bash shell

Whether you are on virtual terminal or using Terminal emulator, command format is same. Following are some of the useful features of bash shell you can use in day-to-day tasks

TAB completion

Tab completion is one of the most useful feature of Bash shell. It has ability to guess the command, directory or file name by pressing TAB after writing partial command, directory or file name. Shell will automatically complete the word if only one option is present otherwise it will offers all possible completions. For example, suppose you want to change directory to *abc* you write *cd ab* and press TAB, the system will complete the word automatically.

Bash Histor y

Shell saves the command you have given on shell in a file in your home directory. You can check the last commands by giving *history* command.

```
 $ history
```

If you want to give last command again, you can press UP arrow key and then press *ENTER* key. You can press *UP* arrow key repeatedly to scroll backward in the history of commands.

Wildcards

Wildcards or wild characters are symbols represents one or more characters in the shell command. Most common wild card characters are

***** Star

? **Question mark**

[ ] **Square baskets**

Where

Star (*)
Represents zero or more characters

Example

$ ls b*
List all files starting with b

Question mark (?)
Represents single character

Example

```
$ ls ?at
```

List all files whose first letter of file name can be any character like bat, cat or hat

Square brackets
Represent a range of characters

Example
ls b[1-5]

List all files starting with b and second character should be numeric character starting from 1 to 5. It can b1, b2, b3, b4 and b5

Check the current shell
# echo $SHELL

Chapter 14 Archiving and Compression

Archiving is the process of collecting and storing a group of files and directories into one file, Tar, cpio, zip and ar are example of utilities which performs this action. Whereas compression is processes of reducing the size of files using mathematical compression algorithm. It is quite useful in sending large files over the internet. Example of compression utilities are gzip, bzip2, xz etc.

GZIP

With gzip you can compress and decompress individual file. When you give gzip command to compress any file, the file will be replaced with extension .gz file keeping the same ownership modes, access and modification times. In sort, you will not see original file only compressed file will be there.

Syntax

```
gzip filename
```

Example

```
gzip sysctl.con f
```

Decompress

To decompress the compressed file use gunzip command. However this command will remove the compressed file you will see only uncompressed file only.

Syntax

```
gunzip filename
```

Example

```
gunzip sysctl.conf.gz
```

BZIP2

Bzip compresses file using compression algorithm known as Burrows Wheel block sorting. Like gzip it removes the original file and creates compressed file with same name plus extension bz2. Each compressed file has same modification date, permission and if possible with same ownership as corresponding original file.

Syntax

```
bzip2 filename
```

Example

bzip2 sysctl.conf
Decompress

To decompress the file compressed with bzip2 command use bunzip2. By default it will not overwrite if file of same name exist unless you give –f option

Syntax

bunzip2 filename

Example

bunzip2 sysctl.conf.bz2
XZ

xz is successor of the lzma utility, it will add .xz extension to compressed file automatically and remove original file. Like bzip and gzip it will keep same modification date, permission and if possible with same ownership as corresponding original file.

Syntax

xz filename

Example

xz sysctl.conf
Decompress

Syntax

unxz filename

Example

unxz sysctl.conf.xz
ZIP

Zip is archiving and compression utility. It creates files with .zip extension. It is cross platform compression utility so you can compress and decompress files from Linux to windows or vice versa. It will keep the original files intact

Syntax

zip zipfile.zip file/files_to_compress

Example

zip abc.zip sysctl.conf signond.conf
you can use zip also with wild card s

zip abc.zip s*
Zip files recursively (include sub directories also)

Syntax

zip –r zipfile.zip directoryname

Example

zip –r asp.zip monk

Listing content of compressed file

Syntax

unzip –l zipfile.zip

Example

unzip –l asp.zip

Decompress

To decompress compressed files with zip command unzip is used

Syntax

unzip zipfile.zip

Example

unzip asp.zip

a r

ar is not so known predecessor of rar which is in still in use in CentOS. It used for archiving only as such there is no compression in it.

Create archive

Syntax

ar cvsr archivefile.a files_to_archive

Example

ar cvsr abc.a s*

Extract archive

Syntax

ar xv archivefile.a

Example

ar xv abc.a

There are many third party utilities available which can be installed, you can try following

- peazip
- p7 zip
- pax
- kgb

Many more..

Chapter 15 BONUS

There are times that you will notice that something is not working right while you are in a GUI desktop environment – there are times wherein a program crashes and the entire system refuses to respond to mouse clicks. There are even situations wherein the GUI may not start at all. When you run into trouble, you can still tell your operating system what to do, but you will have to do it using a text screen or the shell, which serves as the command interpreter for Linux.

Since Linux is essentially a Unix program, you will be doing a lot of tasks using the text terminal. Although the desktop will be doing the job of providing you the convenience to access anything through a click of the mouse, there are several occasions wherein you need to access the terminal.

Learning how to enter commands will save you from a lot of trouble when you encounter an X Window system error, which is the program that controls all the menus and the windows that you see in your desktop GUI. To fix this error, or to prevent it from stopping you to access the program or file that you want, you can pull up a terminal and enter a command instead. In the future, you might want to keep a terminal open in your desktop since it can make you order your computer faster than having to point and click.

The Bash Shell

If you have used the MS-DOS OS in the past, then you are familiar with command.com, which serves as the command interpreter for DOS. In Linux, this is called the shell. The default shell in all the different distros, is called the bash.

Bourne-Again Shell, or bash, can run any program that you have stored in your computer as an executable file. It can also run shell scripts, or a text files that are made up of Linux commands. In short, this shell serves as a command interpreter, or a program that interprets anything that you type as a command and performs what this input is supposed to do.

Pulling up the terminal window can be as simple as clicking on a monitor-looking icon on the desktop – clicking on it will lead you to a prompt. If you can't find that icon, simply search through the Main menu and select the item with has the Terminal or Console label.

Tip: You have the choice to use other shells apart from the bash, just like you have a choice in choosing desktops. You can always change the shell that you are using for your distro by entering the chsh command on the terminal.

The Shell Command

Every shell command follows this format:

A command line, such as a command that follows the above format, is typically followed by parameters (also known as arguments). When entering a command line, you need to enter a space to separate the main command from the options and to separate one option from another. However, if you want to use an option that contains a space in its syntax, you will need to enclose that option in quotation marks. Look at this example :

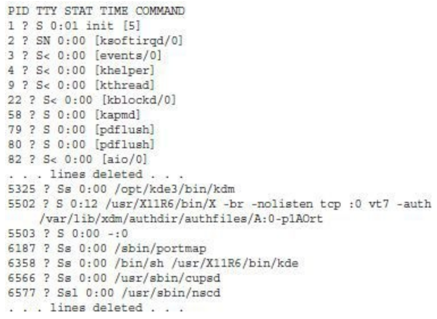
The grep command allowed you to find for a text in a file, which is Emmett Dulaney in this case. Once you press enter, you will get the following result:

If you want to read a file, you can use the “more” command. Try entering this command:

You will be getting a result that appears like this:

To see all programs are running on your computer, use the “ps” command. Try entering this command on the terminal:

The options ax (option a lists all running processes, while opion x shows the rest of the proceses) allows you to see all the processes that are available in your system, which looks like this:



```
PID TTY STAT TIME COMMAND
1 ? S 0:01 init [5]
2 ? SN 0:00 [ksoftirqd/0]
3 ? S< 0:00 [events/0]
4 ? S< 0:00 [khelper]
9 ? S< 0:00 [kthread]
22 ? S< 0:00 [kblockd/0]
58 ? S 0:00 [kapmd]
79 ? S 0:00 [pdflush]
80 ? S 0:00 [pdflush]
82 ? S< 0:00 [aio/0]
. . . lines deleted . . .
5325 ? Ss 0:00 /opt/kde3/bin/kdm
5502 ? S 0:12 /usr/X11R6/bin/X -br -nolisten tcp :0 vt7 -auth
    /var/lib/xdm/authdir/authfiles/A:0-p1AOrt
5503 ? S 0:00 -:0
6187 ? Ss 0:00 /sbin/portmap
6358 ? Ss 0:00 /bin/sh /usr/X11R6/bin/kde
6566 ? Ss 0:00 /usr/sbin/cupsd
6577 ? Ssl 0:00 /usr/sbin/nscd
. . . lines deleted . . .
```

The amount of the command-line options and their corresponding formats would depend on the actual command. These options appear like the –X, wherein X represents one character. For esampe, you can opt to use the option –l for the ls command, which will list a directory’s contents. Look at what happens when you enter the command ls –l in the home directory for a user:

If you enter a command that is too long to be contained on a single line, press the \ (backslash) key and then hit Enter. Afterwards, go on with the rest of the command on the following line. Try typing the following command and hit Enter when you type a line:

This will display all the contents inside the /ets/passwd file.

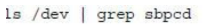
You can also string together (also known as concatenate) different short commands on one line by separating these commands with the ; (semicolon) symbol. Look at this command:

This command will make you jump to your user’s home directory, show the contents of the directory you shanged into, and then display the name of the current directory .

Putting Together Your Shell Commands

If you are aiming to make a more sophisticated command, such as finding out whether you have a file named sbpcd in the /dev directory because you need that file for your CD drive, you can opt to combine different commands to make the entire process shorter. What you can do is that you can enter the ls /dev command to show the contents of the /dev directory and see if it contains the file that you want.

However, you may also get too many entries in the /dev directory when the command returns with the results. However, you can combine the grep command, which you have learned earlier, with the ls command and search for the exact file that you are looking for. Now, type in the following command:



```
ls /dev | grep sbpcd
```

This will show you the directory listing (result of the ls command) while the grep command searches for the string “sbpcd”. The pipe (|) serves as the connection between the two separate commands that you use, wherein the first command’s output is used as the input for the second one.

Chapter 16 How To Create Basic Scripts

As we've seen when you begin learning the command line interface, you generally explore it interactively. That means that you enter one command at a time so that you see the results of each one.

As we begin this section, you can first take a look at *this gif* (*http://bit.ly/2linux3*) which explores a Shakespearean plays' directory using the command line; it counts the number of words and the frequency or how many times the term 'murder' appears in all the plays of Shakespeare.

It's totally fine to use the command-line interface in this interactive manner when you're trying out things but as you may likely notice, typing is one of those activities that are prone to errors. For tasks that are more complex i.e. tasks that you want to repeat, you don't want to retype the code right from the beginning, but make a self-contained shell script that's possible to run as a one-liner .

Your First Shell Script

We'll begin with something simple. Create a junk directory somewhere, like /tmp/my-playground. Your actual workspace doesn't have to be littered with test code.

A shell script is simply a text file that must make sense. To create one, we'll use the nano text editor.

Nano?

Nano is a text editor. It comes preinstalled in nearly all Linux distros. New users prefer it mainly because of its simplicity, which stands out when compared to other command line text editors like emacs and vi/vim. It basically contains many other useful features like line numbering, syntax coloring and easy search (among others).

Let's continue.

We'll create a shell script called hello.sh. Just follow the following steps:

Type 'nano hello.sh' and ru n

Nano will open and give you a blank file to work in. Now enter the following shell command.

Echo 'hello world'

On your keyboard, press ctrl +x to exit the editor. When asked whether you want to save the file, press yes (y).

Nano will then confirm whether you want to save the file. Press enter to confirm

Nano will then confirm whether you want to save the file. Press enter to confirm the action.

Now run the script ‘hello. sh’ with the following command:

```
Bash hello.sh
```

When you look at it as a gif, the steps look something *like this* .

http://bit.ly/2piclinux

Therefore, ‘hello.sh’ is not particularly exciting, but at least it catches the essence of what you want to do, which is to wrap up a series of commands into a file, that is, a script, so that you can re-run the script as much as you’d want. That helps you remove the chance of having typographic errors that come about when you’re retyping commands, and allows you to make the script reusable in various contexts.

Having Arguments In A Re-Usable Shell Script

Let’s now try making ‘hello. sh’ a bit complicated. Now instead of repeating hello world, you’ll create the script in such a way that it says ‘hello’ to a certain value- for instance, a person’s name. That will make the script seem a bit better. You can use it like so:

The gif is as *follows* .

http://bit.ly/2linux5

First off, you customize ‘hello world’ by adding a variable in the place of ‘world’. Try to do that from the command line interactively:

The output is:

```
Hello Dan
```

Therefore, the question here is, how do you get the script ‘hello.sh’ to read in our argument (which is a person’s name in this case) that you pass into it ?

You do that through a special bash variable. The first, second and third arguments you passed from the command line into the script are denoted by the variables which include $1, $2, $3. In the example above therefore, the name George will be kept in the variable $1 as ‘hello.sh’ starts running.

Just reopen ‘hello.sh’ and change your code to the following

After saving the changes, now run the following to see the output.

```
bash hello.sh Mary
```

If you desire to have the output returned in all caps, simply modify ‘hello.sh’ in the following manner, making sure to pipe the output through ‘tr’ in order to replace all the lowercase letters with those in the uppercase.

Now if you have a desire to be concise, you may find that ‘$yourname’ variable is not necessary. The code will be simplified like so :

Now slow down my friend. If you can create a script, you can execute like so:

…then congratulations, you’ve learned an important concept. You’ve just learned how programmers stuff complicated things into a ‘container’ that can be run into one line. Before we start making some decisions, let’s see how we can use a feature known as variables to refer to data, which includes commands’ output or results.

Conclusion

Thank you for making it through to the end of Hacking tools for computers, let's hope it was informative and able to provide you with all the tools you need to achieve your goals whatever they may be.

The next step is to practice hacking by following different examples available in the internet. The most important thing you need to remember being a hacker is to be ethical. Always try to get permission before attacking any targets.

If you want to master hacking further, we have another module that explains in detail other hacking tools and about scripting that is necessary for hackers. Hacking can be a good career to if you can concentrate well without any deviation. This book roughly started from the very beginning that is my installing Linux mint and all the way we have gone to wireless hacking methods.