Linux Activity

Learning Outcomes:

Understand some key concepts of Linux operating system.

1. Users, Groups.
2. File permissions and authorizations.
3. Linux File System.
4. Interactive and Non-interactive Shell.
5. Path Variables.
6. Soft-Links.
7. Linux basic commands.

Linux Introduction: Linux is a multi-user operating system that is based on the UNIX concepts of file ownership and permissions to provide security, at the file system level.

To verify the linux operating system details:
$lsb_release –a

```
LSB Version:    :core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:printing-4.1-amd64:printing-4.1-noarch
Distributor ID: CentOS
Description:    CentOS Linux release 7.2.1511 (Core)
Release:        7.2.1511
Codename:       Core
```

Linux OS has the following components.

Kernel
kernel is the core of the operating system. It establishes communication between hardware devices and software, it also manages system resources.

Basic responsibilities of Kernel are given below.

Device management: A system has many devices connected to it like CPU, memory device, graphic cards, etc. A kernel stores all the data related to all the devices in device driver (without this kernel won't be able to control the devices). All the time Kernel is aware of how to communicate between different devices.

- **Memory management**: Another function that kernel manage is the memory management. Kernel keeps a track of used and unused memory and make sure that processes shouldn't manipulate data of each other using virtual memory address.

- **Process management**: In process management kernel assign enough time and gives priorities to processes before handling CPU to other process. It also deals with security and ownership information.
- **Handling system calls**: Handling system calls means a programmer can write a query or ask the kernel to perform a task.

System Libraries

System libraries are special programs that helps in accessing the kernel's features. A kernel must be invoked to perform a task and this triggering is done by the applications. But applications must know how to place a system call because each kernel has a different set of system calls.

System Tools

Linux OS has a set of utility tools which are usually simple commands. With the help of commands, you can access your files, edit and manipulate data in your directories or files, change location of files or anything.

Development Tools

With the above three components, your OS is running and working. But to update your system you have additional tools and libraries. These additional tools and libraries are written by the programmers and are called tool chain. A tool chain is a vital development tool used by the developers to produce a working application.

We have given example of some tools below.

Bluefish - is one of the most popular IDEs for Web development available. It can handle programming and markup languages, but it focuses on creating dynamic and interactive Web sites

Anjuta - is a free, open source IDE for the C and C++ languages.

Eclipse - is a multi-language IDE, written in Java, with an extensive plug-in system to allow you to extend functionality. Downloaded over 1 million times each month, Eclipse is one of the strongest forces in software development today

End User Tools

These end tools make a system unique for a user. End tools are not required for the operating system but are necessary for a user.

Every operating system has their own features which make the user friendly below are the some of the features of the Linux operating System.

- Multiuser capability: Multiple users can access the same resource
- Multitasking: More than one operation can be performed simultaneously
- Portability: Support different types of hardware
- Security: It provides security in three ways namely authenticating, authorization and encryption
- Graphical User Interface (X Window system):
- Application support: It has its own software repository from where users can download and install many applications.
- File System: Provides hierarchical file system in which files and directories are arranged.
- Open Source: Linux code is freely available to all and is a community based development project.

If we ask the question like why we use Linux operating system, when other popular operating systems are available. We can see some of its features given below:

It is an open source OS which gives a great advantage to the programmers as they can design their own custom operating systems.

Above all you don't have to pay for software and server licensing to install Linux, it’s free and you can install it on as many computers as you want.

It’s completely trouble free operating system and don't have an issue with viruses, malware and slowing down your computer.

Difference between Linux and UNIX:

There is a small confusion like Linux and UNIX are the same but the truth is that they are different and below we can see why.

UNIX	LINUX
It is an operating system which can be used by the copy-right holders.	It is an open-source operating system and freely available.
HP-UX, Sun Solaris etc.	Different distributors like Ubuntu, Red hat etc.
Code developed by AT & T lab.	Linux is a Unix clone, behave like Unix but does not contain its code.
	Supports more file systems than Unix.

User and Group Permissions:
Linux/Unix operating systems have the ability to multitask in a manner similar to other operating systems. However, Linux’s major difference from other operating systems is its ability to have multiple users. Linux was designed to allow more than one user to have access to the system at the same time. In order for this multiuser design to work properly, there needs to be a method to protect users from each other. This is where permissions come in to play.

An easy way to view all of the users on a system is to look at the contents of the /etc/passwd file. Each line in this file contains information about a single user, starting with its *user name* (the name before the first :). Print the passwd file with this command:

```
$ cat /etc/passwd
```

Understanding Sudo

Root is the super user and has the ability to do anything on a system. Therefore, in order to have protection against potential damage sudo is used in place of root. Sudo allows users and groups access to commands they normally would not be able to use. Sudo will allow a user to have administration privileges without logging in as root. A sample of the sudo command is as follows:

```
$sudo <command>
```

Groups:
Groups are collections of zero or more users. A user belongs to a default group, and can also be a member of any of the other groups on a server.

An easy way to view all the groups and their members is to look in the /etc/group file on a server. We won't cover group management in this class, but you can run this command if you are curious about your groups:
$ cat /etc/group

Ownership and Permissions:
An easy way to view all the groups and their members is to look in the /etc/group file on a server. We won't cover group management in this article, but you can run this command if you are curious about your groups:

The most common way to view the permissions of a file is to use ls with the long listing option, e.g. ls -l myfile. If you want to view the permissions of all of the files in your current directory, run the command without an argument, like this:
$ls –l

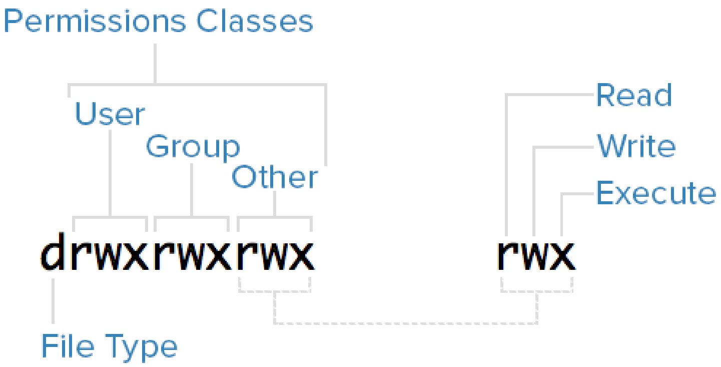
Hint: If you are in an empty home directory, and you haven't created any files to view yet, you can follow along by listing the contents of the /etc directory by running this command: ls -l /etc

Here is an example screenshot of what the output might look like, with labels of each column of output:

```
$ ls -l /etc/
total 1644
drwxr-xr-x.  3 root root       97 Jul 13 17:50 abrt
drwxr-xr-x   4 root root       53 Jul 17 18:10 accumulo
-rw-r--r--.  1 root root       16 Jul 13 18:01 adjtime
-rw-r--r--.  1 root root     1518 Jun  7  2013 aliases
-rw-r--r--.  1 root root    12288 Jul 13 18:14 aliases.db
drwxr-xr-x.  2 root root       49 Jul 13 17:51 alsa
drwxr-xr-x.  2 root root     4096 Sep 29 13:22 alternatives
drwxr-xr-x   3 root root       17 Jul 17 17:27 ambari-agent
drwxr-xr-x   3 root root       17 Jul 17 18:20 ambari-logsearch-logfeeder
drwxr-xr-x   3 root root       17 Jul 17 18:21 ambari-metrics-monitor
-rw-------.  1 root root      541 Jul 27  2015 anacrontab
-rw-r--r--.  1 root root       55 Mar  6  2015 asound.conf
-rw-r--r--.  1 root root        1 Nov 20  2015 at.deny
drwxr-xr-x   4 root root       53 Jul 17 18:12 atlas
drwxr-xr-x.  2 root root       31 Jul 13 17:53 at-spi2
drwxr-x---.  3 root root       41 Jul 13 17:50 audisp
drwxr-x---.  3 root root       79 Jul 13 18:14 audit
-rw-r--r--   1 root root       32 Jul 14 23:47 auto.data
-rw-r--r--   1 root root    12706 Nov 20  2015 autofs.conf
-rw-------   1 root root      232 Nov 20  2015 autofs_ldap_auth.conf
-rw-r--r--   1 root root      844 Jul 14 23:47 auto.master
drwxr-xr-x   2 root root        6 Nov 20  2015 auto.master.d

-- mode --    -Owner-        -File    -Last    -- File Name --
                   -Group-    Size-   Modified-
```

Note that each file's mode (which contains permissions), owner, group, and name are listed. Aside from the *Mode* column, this listing is fairly easy to understand. To help explain what all of those letters and hyphens mean, let's break down the *Mode* column into its components.



File Type

In Linux, there are two basic types of files: normal and special. The file type is indicated by the first character of the mode of a file--in this guide, we refer to this as the file type field.

Normal files can be identified by files with a hyphen (-) in their file type fields. Normal files are just plain files that can contain data. They are called normal, or regular, files to distinguish them from special files.

Special files can be identified by files that have a non-hyphen character, such as a letter, in their file type fields, and are handled by the OS differently than normal files. The character that appears in the file type field indicates the kind of special file a particular file is. For example, a directory, which is the most common kind of special file, is identified by the d character that appears in its file type field (like in the previous screenshot). There are several other kinds of special files but they are not essential what we are learning here.

File types in a long list

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

Permissions Classes

From the diagram, we know that Mode column indicates the file type, followed by three triads, or classes, of permissions: user (owner), group, and other. The order of the classes is consistent across all Linux distributions.

Let's look at which users belong to each permissions class:

User: The owner of a file belongs to this class

Group: The members of the file's group belong to this class

Other: Any users that are not part of the user or group classes belong to this class.

Reading Symbolic Permissions

The next thing to pay attention to are the sets of three characters, or triads, as they denote the permissions, in symbolic form, that each class has for a given file.

In each triad, read, write, and execute permissions are represented in the following way:

Read: Indicated by an r in the first position

Write: Indicated by a w in the second position

Execute: Indicated by an x in the third position. In some special cases, there may be a different character here

A hyphen (-) in the place of one of these characters indicates that the respective permission is not available for the respective class.

For example, if the group triad for a file is r--, the file is "read-only" to the group that is associated with the file.

Understanding Read, Write, Execute

Now that you know how to read which permissions of a file, you probably want to know what each of the permissions actually allow users to do. We will explain each permission individually, but keep in mind that they are often used in combination with each other to allow for meaningful access to files and directories.

Here is a quick breakdown of the access that the three basic permission types grant a user.

Read

For a normal file, read permission allows a user to view the contents of the file.

For a directory, read permission allows a user to view the names of the file in the directory.

Write

For a normal file, write permission allows a user to modify and delete the file.

For a directory, write permission allows a user to delete the directory, modify its contents (create, delete, and rename files in it), and modify the contents of files that the user can read.

Execute

For a normal file, execute permission allows a user to execute a file (the user must also have read permission). As such, execute permissions must be set for executable programs and shell scripts before a user can run them.

For a directory, execute permission allows a user to access, or traverse, into (i.e. cd) and access metadata about files in the directory (the information that is listed in an ls -l).

Examples of Modes (and Permissions)

Now that know how to read the mode of a file, and understand the meaning of each permission, we will present a few examples of common modes, with brief explanations, to bring the concepts together.

-rw-------: A file that is only accessible by its owner

-rwxr-xr-x: A file that is executable by every user on the system. A "world-executable" file

-rw-rw-rw-: A file that is open to modification by every user on the system. A "world-writable" file

drwxr-xr-x: A directory that every user on the system can read and access

drwxrwx---: A directory that is modifiable (including its contents) by its owner and group

drwxr-x---: A directory that is accessible by its group

As you may have noticed, the owner of a file usually enjoys the most permissions, when compared to the other two classes. Typically, you will see that the group and other classes only have a subset of the owner's permissions (equivalent or less). This makes sense because files should only be accessible to users who need access to them for a particular reason.

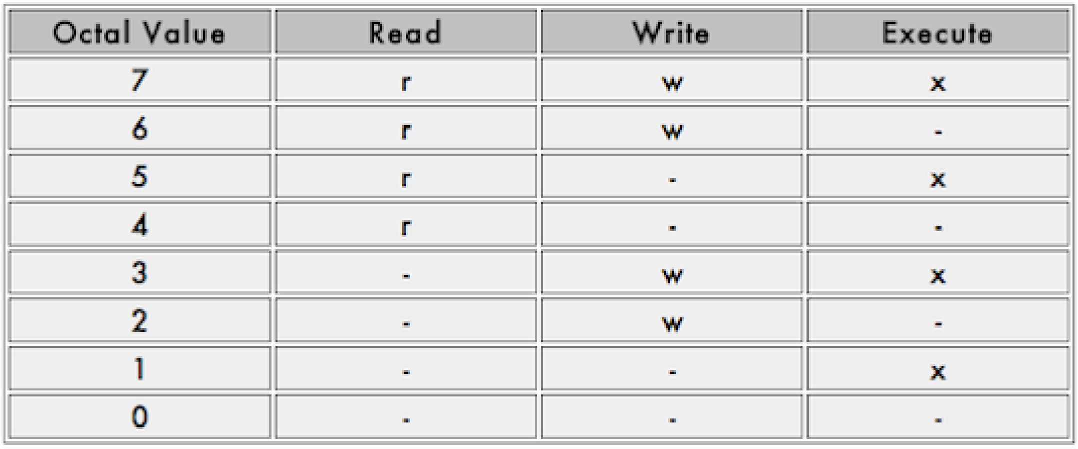
Another thing to note is that even though many permissions combinations are possible, only certain ones make sense in most situations. For example, write or execute access is almost always accompanied by read access, since it's hard to modify, and impossible to execute, something you can't read.

Chmod Command

The command chmod is short for change mode. Chmod is used to change permissions on files and directories. The command chmod may be used with either letters or numbers (also known as octal) to set the permissions.

Chmod Octal Format

To use the octal format, you have to calculate the permissions for each portion of the file or directory. The first ten characters mentioned above will correspond to a four digit numbers in octal. The execute permission is equal to the number one (1), the write permission is equal to the number two (2), and the read permission is equal to the number four (4). Therefore, when you use the octal format, you will need to calculate a number between 0 and 7 for each portion of the permission. A table has been provided below for clarification.



Octal Value	Read	Write	Execute
7	r	w	x
6	r	w	-
5	r	-	x
4	r	-	-
3	-	w	x
2	-	w	-
1	-	-	x
0	-	-	-

```
$chmod 777 <file-name>
```

Linux File System, Kernel and Shell:

A Linux system is basically divided in three major components: File System (LFS), Shell and Kernel. Kernel is the core program which manage system hardware devices. Shell provides user interface to run the commands. File system organizes the data in systematic way. Collectively LFS, Shell and kernel provides a way to interact with system and an environment to run commands and manage data.

The Linux file system is usually thought of as a tree in structure. A general structure is given below.

Let’s understand these components in more details one by one.

Linux File System (LFS)

Linux accesses every object as file. Files are systematically organized in directories. Linux starts file system with root directory(/). All files and directories are created and managed under the root directory. Since root directory stands on the top in file system, it has no parent directory. Besides root directory, every directory in Linux has a parent directory. Linux allows us to create as many files and directories as we want. We can create files under the existing directories or may create new directories.

System Directories
System directories contain files, software, applications and scripts which are required to run and maintain the Linux. System directories are automatically created during the installation.

Some common system directories are:
$ls /

Directory	**Description**
/	First directory in Linux File System. It is also known as root Directory or main directory. All files and directories are created And managed under this directory.
/home	Default directory for user data. Whenever we add a new user, Linux automatically creates a home directory matching with his username in This directory. Whenever user login, Linux starts his login session From home directory.
/root	This is the home directory for root user. Root user is the super user in Linux. For security reason Linux creates a separate home directory for root user. Root user account is also being created during the installation automatically.
/bin	This directory contains standard commands files. Commands stored in this directory are available for all users and usually do not require any special permission to run.

/sbin	This directory contains system administration commands files. Commands st directory are available only for root user and usually requires special privilege to run.
/usr	This directory contains user application software files, third party software and scripts, document files and libraries for programming languages.
/var	This directory stores variable data files such as printing jobs, mail box etc.
/etc	This directory contains system configuration files.
/boot	This directory contains Linux boot loader file.
/mnt	This directory is used to mount remote file system and temporary devices such as CD, DVD and USB.
/dev	This directory contains device files. Usually files in this directory are dynamically generated and should be never edited.
/tmp	This directory provides temporary location for applications.

Shell is a command interpreter. It take commands from user, execute them and display the results. Shell supports I/O (Input / Output) redirection which means it can read commands from non-standard sources such as script files. As well as it can also redirect output to any supportive device (such as printer) or data server.

Several Shells are available in Linux such as Kom, TCSH, Z shell, Bash etc. Although several shells are available, only one shell is set to default in CentOS Linux. **Bash (*Bourne Again shell*)** shell is the default shell in CentOS Linux.

To list all the available shells - $cat /etc/shells
To list the current shell - $echo $0

Kernel
Kernel is the core application in Linux operating system. It communicates directly with system devices such as memory, CPU, CDROM, Hard disk etc.

User □ Shell -> kernel □ hardware □ kernel □ Shell □ User

When a user wants to access any device, he types appropriate command at command prompt. Shell interpreters the command and hands over the instruction to kernel. Kernel communicates with device and process the user requests.

Interactive and Non-interactive Shell:
An interactive login shell is started after a successful login, using /bin/login, by reading the /etc/passwd file. This shell invocation normally reads /etc/profile and its private equivalent ~/.bash_profile upon startup.

A non-interactive shell is usually present when a shell script is running. It is non-interactive because it is processing a script and not waiting for user input between commands. For these shell invocations, only the environment inherited from the parent shell is used.

What is /etc/profile used for?
The .profile or .bash_profile files in your home directory. These files are used to set environmental items for a users shell. Items such as umask, and variables such as PS1 or PATH.

The /etc/profile file is not very different however it is used to set system wide environmental variables on users shells. The variables are sometimes the same ones that are in the .bash_profile, however this file is used to set an initial PATH or PS1 for all shell users of the system.

/etc/profile.d

In addition to the setting environmental items the /etc/profile will execute the scripts within /etc/profile.d/*.sh. If you plan on setting your own system wide environmental variables it is recommended to place your configuration in a shell script within /etc/profile.d.

/etc/bashrc
Like .bash_profile you will also commonly see a .bashrc file in your home directory. This file is meant for setting command aliases and functions used by bash shell users.

Just like the /etc/profile is the system wide version of .bash_profile.
The /etc/bashrc and /etc/bash.bashrc (in Ubuntu) is the system wide version of .bashrc.
Interestingly enough in the Red Hat implementation the /etc/bashrc also executes the shell scripts within /etc/profile.d but only if the users shell is a Interactive Shell (aka Login Shell)

When are these files used?

The difference between when these two files are executed are dependent on the type of login being performed. In Linux you can have two types of login shells, Interactive Shells and Non-Interactive Shells. An Interactive shell is used where a user can interact with the shell, i.e. your typical bash prompt. Whereas a non-Interactive shell is used when a user cannot interact with the shell, i.e. a bash scripts execution.

The difference is simple, the /etc/profile is executed only for interactive shells and the /etc/bashrc is executed for both interactive and non-interactive shells.

Path Variables:
PATH is an *environmental variable* in Linux and other Unix-like operating systems that tells the *shell* which directories to search for *executable files* (i.e., ready-to-run programs) in response to commands issued by a user. It increases both the convenience and the safety of such operating systems and is widely considered to be the single most important environmental variable.

```
$echo $PATH
$export PATH=$PATH:/usr/local/bin
$PATH=$PATH:/usr/local/bin; export PATH
```

Soft Links:
In your Linux file system, a link is a connection between a file name and the actual data on the disk. There are two main types of links that can be created: "hard" links, and "soft" or symbolic links. Hard links are low-level links which the system uses to create elements of the file system itself, such as files and directories.
Users typically do not want to create or modify hard links themselves, but symbolic links are a useful tool for any Linux user. A symbolic link is a special file that points to another file or directory, which is called the target. Once created, a symbolic link can be treated as if it is the target file, even if it is has a different name and is located in another directory. Multiple symbolic links can even be created to the same target file, allowing the target to be accessed by multiple names.

The symbolic link is a file in its own right, but it does not contain a copy of the target file's data. It is similar to a shortcut in Microsoft Windows: if you delete a symbolic link, the target is unaffected. Also, if the target of a symbolic link is deleted, moved, or renamed, the symbolic link is not updated. When this happens, the symbolic link is called "broken" or "orphaned", and will no longer function as a link.

To create symbolic link:

```
$ln –s <target> <link-name>
$ln –s myfile.txt mylink
```

Linux Commands:

1. **pwd**: [print working directory / present working directory]

    Usage: pwd

2. To change the directory to the current user desktop: **cd**

    Usage: cd <location>
    cd /home/rameshmelapu9416/datasets

    cd ~ : Brings to home directory

    cd - : Brings you to the previous directory of the current directory.

    cd .. : brings to parent directory of the current directory.

    cd / : Takes to the entire system's root directory.

It is very important to know the difference between absolute and relative path. Because correct path will only lead you to your destination directory.

When you define, a path starting with a slash ('/') sign, then root of the file is assumed. If you don't put a '/' then the current directory is assumed to be the starting point.

Try cd /home/<your-id>

Try cd home

Try cd /home

3. To create a directory / make a directory: **mkdir**

Usage: mkdir <location of the directory>
Ex: mkdir test

mkdir -m=rwx myfile : It will create a directory and give us read, write and execute permission.

4. To list all the files available in a given directory: **ls**

Usage: ls <location>
Ex: ls . [dot "." refers to the current location. It is optional and only ls will also list the files]

ls -a: In Linux hidden files start with a “ . “ , the normal ls command will not show them ls -a will show the hidden files.

ls – l: It will show the list in long list format.

ls -lh: Command will show the file size in readable format. Normally file size is in terms of bytes, this command help in displaying them of mb, gb or tb.

ls -lhS: Display the files in descending order according to the size.

ls -d */ : Only display sub-directories.

ls -li : This command prints the index number if file in the first column.

ls -p : It is used to identify the directory easily by marking the directories with a slash (/) line sign.

ls -r : It is used to print the list in reverse order.

ls -R : It will display the content of the sub-directories also.

ls -lX : It will group the files with same extensions together in the list.

ls -lt : It will sort the list by displaying recently modified filed at top.

ls ~ : It gives the contents of home directory.

ls ../ : It give the contents of parent directory.

ls –version : It checks the version of ls command.

5. To list all files and view the metadata of files in the folder
Usage: ls -<Options> <location>
Ex: ls -ltr or ls –latr

6. To create a file from terminal using "vi" editor: **vi**
Usage: vi <filename> <Enter>
Ex: vi testTxt.txt
Esc i <This brings you to insert mode>

Type some text to be persistent on the file. Save and quit from vi prompt.
Esc :wq!

This brings you back to terminal
Type "ls" to see if the file is created.

7. To view the content of the file or write the content of file to console. Cat [concatenate] is also used to join one or more files into a single file: **cat**
Usage: cat <filename>
Ex: cat testTxt.txt

8. To get the name of the machine: **uname**
Usage: uname -a

9. Command to print the "history" of events / commands used in a terminal session: **history**
Usage: history

10. Command to create a file, only if it doesn’t exist. If the file already exists it will update the timestamp and not the contents of the file: **touch**
Usage: touch <filename>
Ex: touch testTxt.txt
touch newFile.txt

11. Display the calendar of the present month: **cal**
Usage: cal <MM YYYY>
Ex: cal 02 1996

12. Print the current date and timestamp on console: **date**

    Usage: date

13. Copy a file from one location to another. Retains the source file, creates a new copy at destination: **cp**

    Usage: cp testTxt.txt copyFile.txt
    ls <Displays both textTxt.txt and copyFile.txt

14. Move a file from one location to another. Does not retain the file at source, moves the file to destination. The command can be used to rename a file: **mv**

    Usage: mv <source> <destination>
    Ex: mv copyFile.txt newName.txt

15. To remove a file: **rm**

    Usage: rm <filename>

16. To remove a directory and files in it recursively: **rmdir**

    Usage: rmdir -r directory name [**DO NOT perform this activity at root or in any folder whose content we are not aware of]**
    Ex: create a directory that can be deleted

    ```
    mkdir testDel
    cd testDel
    touch file1.txt
    cd ..
    rmdir -r testDel
    ```

17. Allows file contents or piped output to be sent to the screen one page at a time: **more**

    Usage: more <filename>

18. View contents of a file one page at a time. [Similar to more command]: **less**

    Usage: less <filename>

19. To clear the contents of terminal or flush the screen: **clear** [Alternatively Ctrl + L can be used]

    Usage: clear

20. Locate a file on the filesystem or find the file on the filesystem: **find** or **locate**

    Usage: find <location> -name "name of entity"
    Usage: locate <filename>

21. Print the number of newlines, words, and bytes in files: **wc**

Usage: wc <options> <filename>
Ex: wc -c testTxt.txt //print the byte counts
wc -l testTxt.txt //prints the number of lines
wc -w //print the number of words

22. To arrange the content items in a specific order: **sort**
Usage: sort <options> <filename>
Ex: sort -k2 testTxt.txt //The k2 refers to the second column. Have a CSV file and run the command against it.

23. To report or filter out repeated lines in a file: **uniq**
Usage: uniq [option] [input [output]]
Ex: Create a file, myFile.txt with the following content
This is a line.
This is a line.
This is a line.
This is also a line.
This is also a line.
This is also also a line
uniq myFile.txt
uniq -c myFile.txt //Count
uniq -d myFile.txt //Only print duplicated
uniq -u myFile.txt //Print only unique lines

24. To execute commands as a super user: **su** or **sudo**
Usage: su - password: cloudera
changes cloudera@localhost to root@localhost> prompt

25. Display used and available disk space: **df**
Usage: df <options>

26. Show much space each file takes up: **du**
Usage: du <options>

27. Display the name of the user who has logged in: **who**
Usage: who
Note how does "who" defer in its output "who am i" or "whoami"

28. Print the last 10 lines of each FILE to standard output: **tail**
Usage: tail <Options> filename
Ex: tail myFile.txt

29. The first 10 lines of each FILE are printed to standard output: **head**
Usage: head <option> <file>
Ex: head -15 myFile.txt //Displays first 15 lines of output

30. Command to determine the file type of a give file: **file**

    Usage: file <name of file>
    Ex: file myFile.txt

31. Search in text or search the given file for lines containing a match to the given strings or words: **grep**

    Usage: grep <pattern> <filename>
    Ex: grep "test" myFile.txt

32. Look at all the processes running by the OS: **ps**

    Usage: ps <options>
    Ex: ps -ef

33. Get a manual of all commands or a specific command from the unix / linux systems: **man /whatis**

    Usage: man <command name>
    Ex: man ls

34. File permissions: Understanding the user permissions on files and changing the owners of the files.

    chmod
    chown
    Usage:
    There are three types of file permissions – read, write, execute
    File-Type Owner Group Others
    rwx rwx rwx

        Read permission – 4
        Write permission – 2
        Execute Permission – 1
        rwxrw-r-- = 764
        673 = rw-rwx-wx //Octal representation of each triplet

    $chmod 744 dd.txt

        Results in a permission list of "rwxr--r--"

    chown changes the ownership of the file

        Usage: chown "valid user" "File name"

    Following are the symbolic representation of three different roles:

        u is for user,
        g is for group,
        and o is for others.

Following are the symbolic representation of three different permissions:

r is for read permission,
w is for write permission,
x is for execute permission.

Give full access to user and group (i.e read, write and execute) on a specific file.

```
$ chmod ug+rwx file.txt
```

Revoke all access for the group (i.e read, write and execute) on a specific file.

```
$ chmod g-rwx file.txt
```

Summary:

In this session, we learnt about:

Basic commands in Linux Operating System, and Key Concepts such as

Users and Groups

File Permissions and Authorizations

Kernel

Shell

Linux File System

Interactive vs Non-interactive Shell

Path variables.

Symbolic or Soft Links.