GETTING STARTED
WITH
FILES & DIRECTORIES

➢ MANAGING FILES & DIRECTORIES:

- We will explore various aspects of file management in Linux, including listing files, creating new files, displaying file contents, and performing essential file operations such as copying, moving, renaming, and deleting.

LS COMMAND:

- It lists directory contents of files and directories. It provides valuable information about files, directories, and their attributes.

SYNTAX: **$ls [options] <file-name>**

→ To List Files:

$ls

→ To List reverse order:

$ls -r

→ Long format that displays detailed information about files and directories:

$ls -l

→ To list specific file details:

$ls -l file-name

→ Represent all files Include hidden files and directories:

$ls -a [File start with **"."**]

→ List files and directories recursively, including subdirectories:

$ls -R

→ List inode which displays the index number of each file and directory:

$ls -i

→ List directories themselves, rather than their contents:

$ls -d

→ Sort files and directories by their sizes, listing the largest ones first:

$ls -s **or** **$ls -S**

→ Print file sizes in human-readable format (e.g., 1K, 234M, 2G).

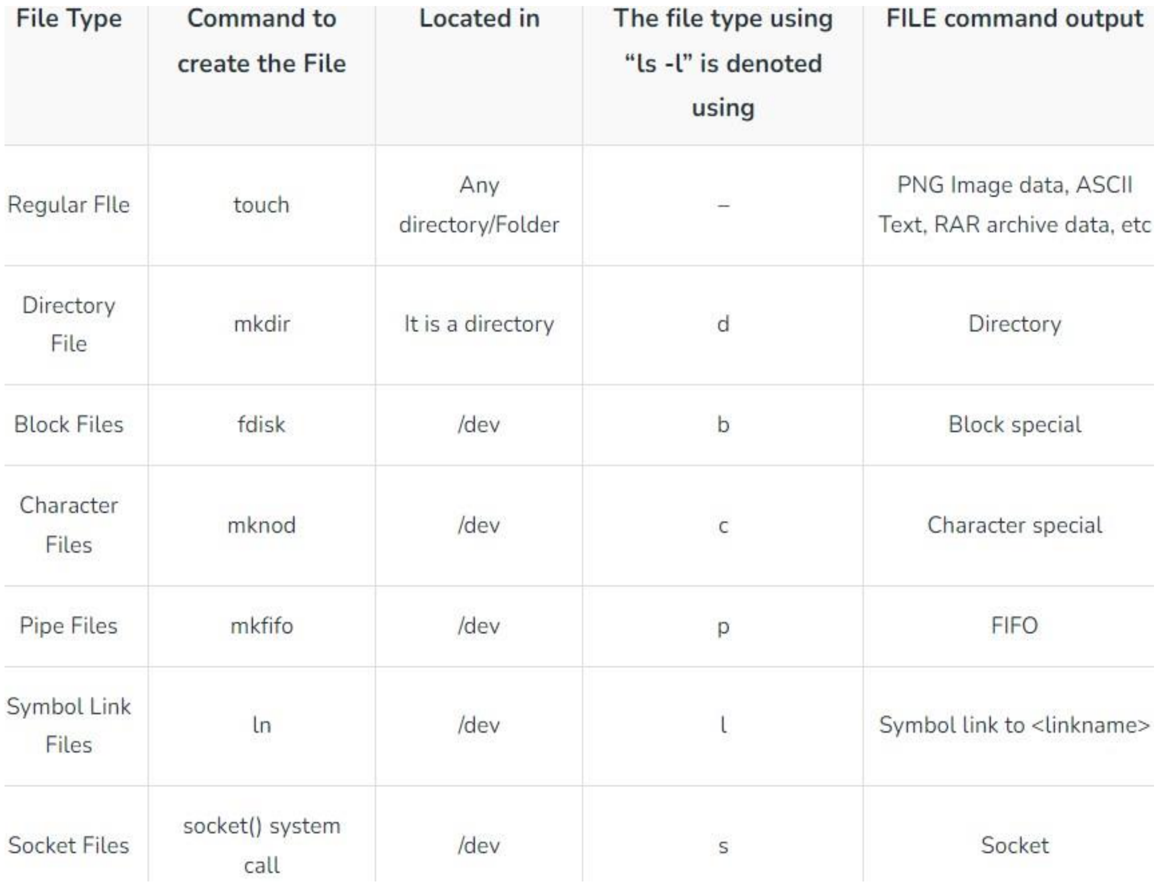
$ls -lh

→ Sort files and directories by their last modification time, displaying the most recently modified ones first:

$ls -t **$ls -lhtr**

IDENTIFYING LINUX FILE TYPES:

- In Linux everything considers as a file. The following are the file identifiers.



File Type	Command to create the File	Located in	The file type using “ls -l” is denoted using	FILE command output
Regular FIle	touch	Any directory/Folder	–	PNG Image data, ASCII Text, RAR archive data, etc
Directory File	mkdir	It is a directory	d	Directory
Block Files	fdisk	/dev	b	Block special
Character Files	mknod	/dev	c	Character special
Pipe Files	mkfifo	/dev	p	FIFO
Symbol Link Files	ln	/dev	l	Symbol link to <linkname>
Socket Files	socket() system call	/dev	s	Socket

SYNTAX: $file [options] <file-name>

→ To check the file version:

```
$file -v
```

→ We can test a file type by typing the following command:

```
$file -f file.txt
```

→ We can read the block or character special file.

```
$file -s /dev/sda
$file -s /dev/nvme0n1
```

➢ PATH:

- A path describes a location to a file in a file system of an OS.
- A path to a file is a combination of / and alpha numeric characters.
- There are two types.

ABSOLUTE PATH:

- Path is defined as the specifying the location of a file or directory from the **root directory (/).**
  Example: **$ls /usr/share/doc**
  $ls /home/raju

RELATIVE PATH:

- Absolute paths are useful, but they're not always efficient.
- It is defined as path related to the **present working directory (pwd).**
  Example: **$cd /var/log**

  $ls

PWD COMMAND:

- Prints the current working directory.
- The pwd command is your Linux system's compass, in that it tells you where you are.
  → To print working directory
  $pwd

STAT COMMAND:

- It displays file or file system status.
  → To check file statistics:

  $stat myfile

  $stat -f myfile

CAT COMMAND (concatenates files):

- It is used to concatenate files and print on the standard output.

SYNTAX: **$cat [options] <file>**

→ Create a new file:

```
$cat>filename
  Hi….
  This is a first text file.
Ctrl+D (To save)
```

→ To Print a output:

```
$cat  <filename>
        (or)
$cat < filename
```

→ To appending a content in a last line:

```
$cat >> filename
   Hello…
Ctrl + D (To save)
```

→ To setting Line Numbers:

```
$cat -n filename
```

→ Concatenating (merging) files (as the name suggests):

```
$cat filename1 filename2  >filename12
```

→ To View multiple files:

```
$cat filename1 filename2
```

→ Cat command can display content in reverse order using tac command:

```
$tac filename
```

→ Cat command can highlight the end of line:

```
$cat -E filename
```

TOUCH COMMAND:

- The touch command is another one that serves a **dual purpose**.
- It is used to create a new **empty (zero byte)** file.
- Touch can create, change, and modify file timestamps.

SYNTAX: **$touch [options] <filename>**

→ Create a new empty file:

```
$touch myfile
$ls -l myfile
$stat myfile
```

→ Create multiple empty files:

```
$touch myfile1 myfile2 myfile3
```

→ To change file access and modification time:

```
$touch -a myfile
$stat myfile
```

→ To change file modification time:

```
$touch -m myfile
```

→ Explicitly Set the **Access** and **Modification** times:

```
$touch -c -t YYMMDDHHMM filename
$touch -c -t 202307152359 myfile
```

→ Create a File using a specified time:

```
$touch -t YYMMDDHHMM.SS filename
$touch -t 202307151159.59 myfile
```

→ Use the time stamp of another File:

```
$touch -r myfile myfile1
$ls -l myfile
$ls -l myfile1
$stat myfile
```

MKDIR COMMAND:

- It is used to create a new directory.

SYNTAX: **$mkdir [options] <Dir name>**

$mkdir cloud

→ To make multiple directories:

$mkdir cloud1 cloud2 cloud3

(or)

$mkdir cloud{1..3}

→ Create a nested directory:

$mkdir -p world/asia/india/ap/vskp

→ Verifying directory structure:

$ls -R world

$tree world

DIRECTORY NAVIGATION:

CD COMMAND:

- It is a change directory.
- Changing directories is easy as long as you know where you are (your current directory) and how that relates to where you want to go.

SYNTAX: **$cd <Dir name>**

$cd cloud

$touch aws

$ls

→ To change directory to the one above your current directory, use the double period (dot) argument:

```
$cd ..

$pwd
```

→ To get Users home directory:

```
$cd ~    (or)    $cd
```

NOTE: SHORTCUTS:

▪ Single dot, or **.**	**:** Represents current location.
▪ Double dot, or **..**	**:** To change one above directory
▪ Double dot, or **../..**	**:** To change two above directory
▪ Hyphen or -	**:** Previous Directory
▪ Tilde, or ~	**:** Users home directory

RM COMMAND:

- The rm command **removes (deletes)** files and directories.

SYNTAX: **$rm [oprions] <file/dir name>**

→ To remove a file with interactive:

```
$rm -i filename
```

→ To remove file forcefully:

```
$rm -f filename
```

→ Remove a directory:

```
$rm -ri cloud
```

RMDIR COMMAND:

- It will remove only empty directories.

```
$rmdir dirname

$rmdir cloud1
```

CP COMMAND:

- cp command is used to **copies files** and **directories.**
- There's no great secret to its usage and you simply issue the copy (cp) command, the file or directory source, and the destination.

SYNTAX: **$cp [options] <source-file> <target-file>**

→ Copying file to file:

```
$cp -i file1 file2
$cat file2
```

→ Copying file to directory:

```
$cp -i file1 cloud
$ls cloud
```

→ Copy an entire directory and all its contents, including subdirectories:

```
$cp -ri cloud world
$ls world
```

MV COMMAND:

- The mv command **moves files** and **directories** from one directory to another or renames a file or directory.
- When you use the mv command to rename a file or directory, the Target Directory parameter can specify either a new file name or a new directory path name.

→ To rename a file or directory:

```
$mv existingfile newfile
$mv file1 file123
```

→ To move a file into a directory:

```
$mv filename Dirname
$mv file1 cloud
$ls cloud
```

➢ META / WILD CARD CHARACTERS:

- Metacharacters are **special characters** that are used to represent something other than themselves.

ASTERISK (*): It matches zero or more characters.

→ List all files and directories:

$ls *

→ List files and directories start with a:

$ls a*

→ List files and directories ending with .cfg:

$ls *.cfg

→ List all files and directories that have numbers after file:

$ls file[0-9]

→ Copying all files ending with .py into cloud directory:

$cp -i *.py cloud

→ Copying all files and directories starting with name aws into cloud directory:

$cp -ri aws* cloud

→ Removing all files .png:

$rm -i *.png

QUESTION MARK (?): It matches any single character.

→ List all files and directories that have one character:

$ls ?

→ List all files that have two characters after ‘file’.

$ls file??

SEMICOLON (;):

- A command line can consist of multiple commands. Each command is separated by a semicolon.
- The exit status is that of the last command in the chain of commands.
- It is a command independent; meaning is not depending on first command.

SYNTAX: **$command1 ; command2 ; …..commamdn**

→ Run commands who, date, and whoami command in a line:

```
$who ; date ; whoami
$who ; data; whoami
```

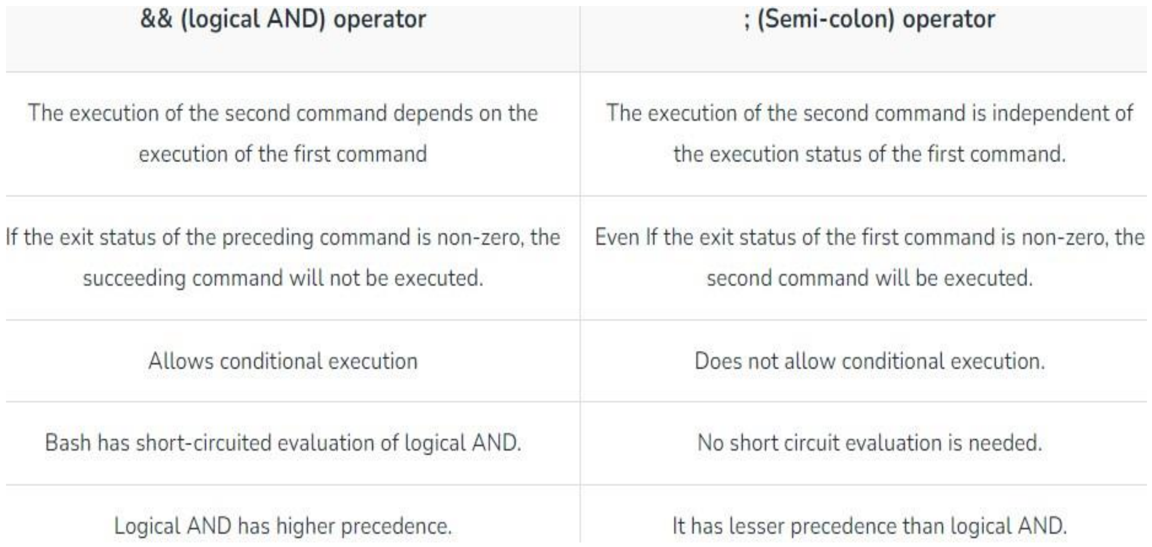
LOGICAL AND (&&):

- Run multiple commands in a line separated by &&.
- It is a command dependent; meaning is the second command will only execute if the first command has executed successfully.

SYNTAX: **$command1 && command2 && …. Commandn**

→ Run commands who, date, and whoami command in a line:

```
$who && date && whoami
$who && data && whoami
```



&& (logical AND) operator	; (Semi-colon) operator
The execution of the second command depends on the execution of the first command	The execution of the second command is independent of the execution status of the first command.
If the exit status of the preceding command is non-zero, the succeeding command will not be executed.	Even If the exit status of the first command is non-zero, the second command will be executed.
Allows conditional execution	Does not allow conditional execution.
Bash has short-circuited evaluation of logical AND.	No short circuit evaluation is needed.
Logical AND has higher precedence.	It has lesser precedence than logical AND.