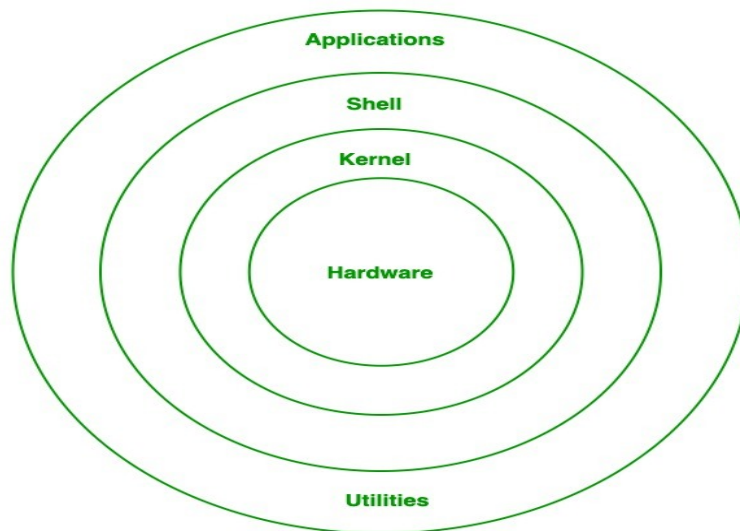# Architecture of Linux operating system

The architecture of Linux is composed of kernel, shell and application programs that is software.

HARDWARE: physical parts of a computer, such as central processing unit (CPU), monitor, mouse, keyboard, hard disk and other connected devices to CPU.

KERNEL: A kernel is a computer program and is the central, core part of an operating system. It manages the operations of the computer and the hardware, most notably memory and CPU time. It is an integral part of any operating system.

SHELL: Shell is an environment in which we can run our commands, programs, and shell scripts. It is a user interface for access to an operating system's services. (User interface program execution, file system manipulation, input/output operations, communication, resource allocation, error detection, security, and protection)

The diagram of kernel shell user relationship is as below:

THE KERNEL

1. Kernel is core (main) part of Linux operating system.
2. It is collection of routine communicate with hardware directly.
3. It loads into memory when Linux is booted.
4. Kernel provides support to user programs through system call.
5. Kernel manages Computer memory, schedules process, decides priorities of processes and performs other tasks.
6. Kernel does lot of work even if no application software is running.
7. Hence kernel open called as application software gateway to the computer resources.
8. Kernel is represented by /boot/vmlinuz.

Shell

1. Shell is interface between user and kernel.

2. It is outer part of operating system.

3. A shell is a user interface for access to an operating system's services Shell is an environment in which we can run our commands, programs, software and shell scripts.

4. Computers do not have any inherent capability of translating commands into actions, it is done by shell.

5. There can be many shells in action - one shell for each user who logged in.
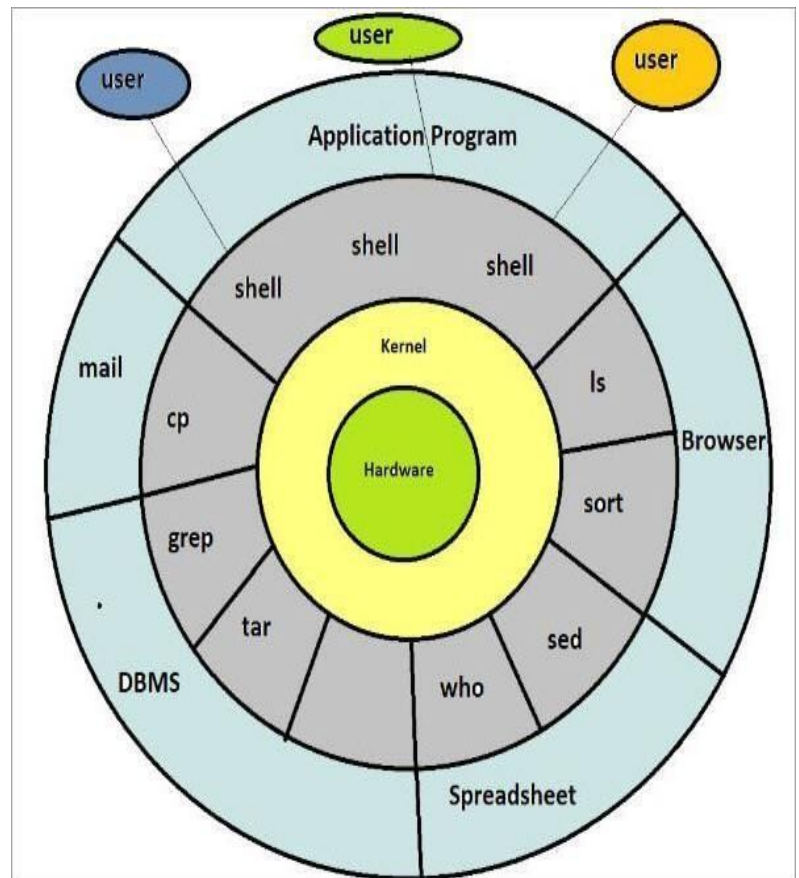
How Shell works?

When we enter commands through the keyboard, it gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output. **OR** the shell thoroughly examines the keyboard input for special characters. If it finds any, it rebuilds a simplified command-line, and finally communicate with the Kernel to see that the command is executed.

To know the running shell, use echo $SHELL command in terminal.

Application programs/software

An application, or application program, is a software program that runs on your computer. It is excited by user. Some inbuilt application programs in Linux are terminal, Firefox browser, Libre o ce.
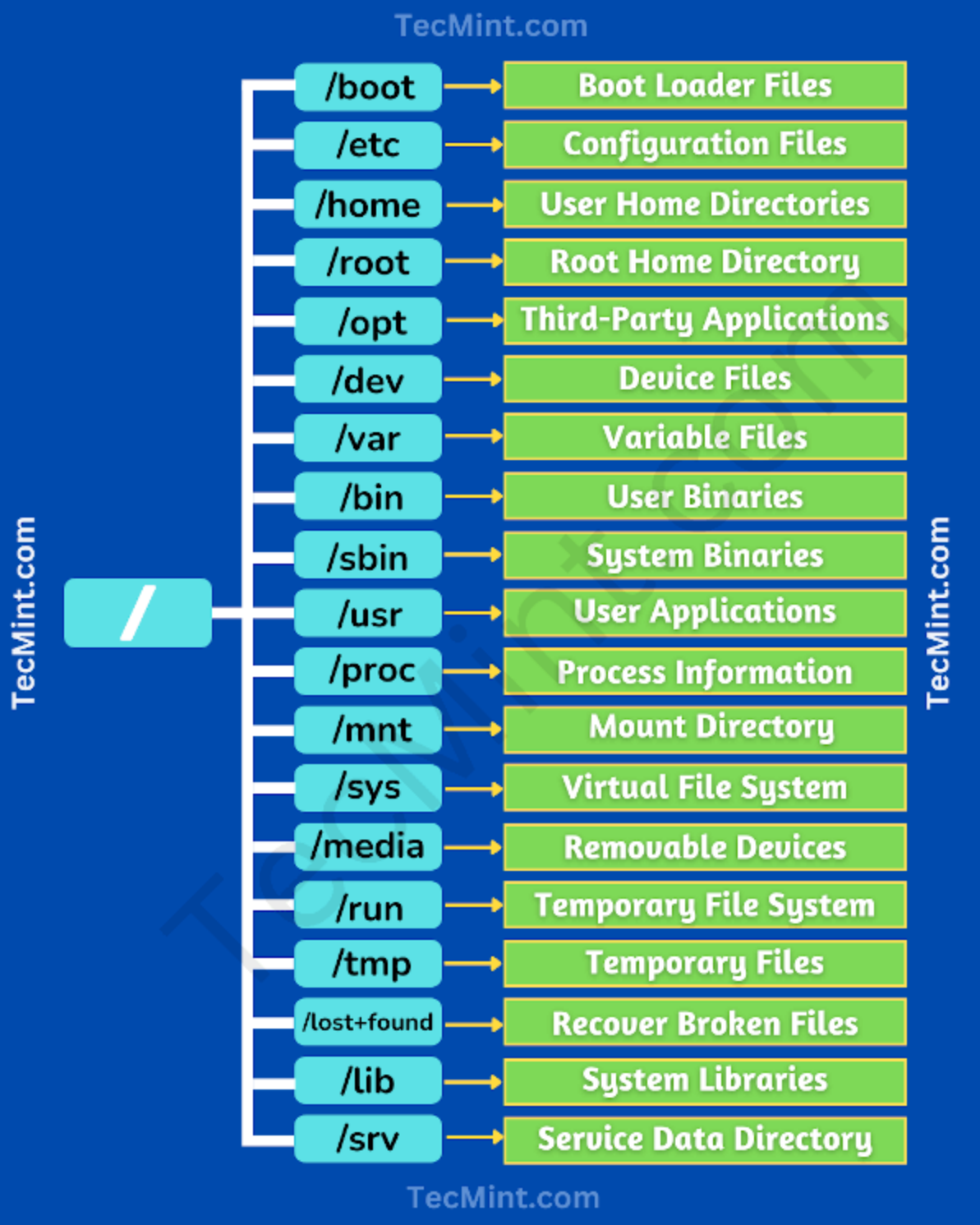
System calls

There are over a thousand commands available in Linux operating system. These all commands use a **function to communicate with kernel - and it is called as system call**.

System call is the interface between a process and an operating system or **System calls** are the only **entry points into the kernel system.**

Ex.   1) a typical Linux writes a file with write system call. Same system call can access both a file and device.

2) Open system call opens both file and device. These system calls are built into kernel. And interaction through the system calls represents and e cient means of communication with operating system.

TecMint.com

Directory	Description
/boot	Boot Loader Files
/etc	Configuration Files
/home	User Home Directories
/root	Root Home Directory
/opt	Third-Party Applications
/dev	Device Files
/var	Variable Files
/bin	User Binaries
/sbin	System Binaries
/usr	User Applications
/proc	Process Information
/mnt	Mount Directory
/sys	Virtual File System
/media	Removable Devices
/run	Temporary File System
/tmp	Temporary Files
/lost+found	Recover Broken Files
/lib	System Libraries
/srv	Service Data Directory

1. **ls** - The most frequently used command in Linux to list directories.
2. **pwd** - Print working directory command in Linux
3. **cd** - Linux command to navigate through directories.
4. **mkdir** - Command used to create directories in Linux.
5. **mv** - Move or rename files in Linux.
6. **cp** - Similar usage as mv but for copying files in Linux
7. **rm** - Delete files or directories
8. **touch** - Create blank/empty files
9. **cat** - Display file contents on the terminal
10. **clear** - Clear the terminal display
11. **echo** - Print any text that follows the command
12. **man** - Access manual pages for all Linux commands
13. **uname** - Linux command to get basic information about the OS
14. **whoami** - Get the active username
15. **tar** - Command to extract and compress files in Linux
16. **grep** - Search for a string within an output
17. **head** - Return the specified number of lines from the top
18. **tail** - Return the specified number of lines from the bottom
19. **diff** - Find the difference between two files
20. **cmp** - Allows you to check if two files are identical
21. **comm** - Combines the functionality of diff and cmp
22. **sort** - Linux command to sort the content of a file while outputting
23. **export** - Export environment variables in Linux
24. **zip** - Zip files in Linux
25. **unzip** - Unzip files in Linux
26. **tar** -xvzf filename --- to untar tar file
27. **ssh** - Secure Shell command in Linux
28. **service** - Linux command to start and stop services
29. **ps** - Display active processes
30. **df** - Display disk filesystem information
31. **mount** - Mount file systems in Linux
32. **chmod** - Command to change file permissions
33. **chown** - Command for granting ownership of files or folders
34. **ifconfig** or **ip addr** - Display network interfaces and IP addresses
35. **traceroute** - Trace all the network hops to reach the destination
36. **wget** - Direct download files from the internet
37. **ufw** - Firewall command
38. **iptables** - Base firewall for all other firewall utilities to interface with
39. **apt, pacman, yum, rpm** - Package managers depending on the distro
40. **sudo** - Command to escalate privileges in Linux
41. **cal** - View a command-line calendar
42. **alias -** Create custom shortcuts for your regularly used commands
    ex: alias rmd = 'rm -r'

43. **top** - View active processes live with their system usage
44. **useradd and usermod** - Add new user or change existing user data
45. **usermpd** -G user1 user2 ----so here user is going to add into user1 group
46. **passwd** - Create or update passwords for existing users
47. **cat /etc/passwd** ----to check list of users
48. **passwd <user>** --- to create password for user
49. **cat /etc/group** --- to list of groups
50. **vi /etc/ssh/sshd_config** ---to enable password base authentication.
51. **service sshd restart** –to restart service
52. **id root** ------to know root user and group status
53. **netstat -tulpn | grep LISTEN** ---to list port enable
54. **ping** – to hit any public domain like google
55. **yum list installed** ----to know the list of installations
56. **id root** ----- check user and group detail
57. **service sshd restart**

The ls command in Linux

The ls command is used to list files and directories in the current working directory. This is going to be one of the most frequently used Linux commands you must know of.

```
root@ubuntu:/# ls
bin   dev  go1.13.5.linux-amd64.tar.gz  initrd.img      lib    lost+found  mnt  proc  run   snap  sys  usr  vmlinuz
boot  etc  home                         initrd.img.old  lib64  media       opt  root  sbin  srv   tmp  var  vmlinuz.old
root@ubuntu:/#
```

As you can see in the above image, using the command by itself without any arguments will give us an output with all the files and directories in the directory. The command offers a lot of flexibility in terms of displaying the data in the output.

Learn more about the ls command (link to full article)
The pwd command in Linux

The pwd command allows you to print the current working directory on your terminal. It's a very basic command and solves its purpose very well.

```
root@ubuntu:/etc/network/if-pre-up.d# pwd
/etc/network/if-pre-up.d
```

Now, your terminal prompt should usually have the complete directory anyway. But in case it doesn't, this can be a quick command to see the directory that you're in. Another application of this command is when creating scripts where this command can allow us to find the directory where the script has been saved.

The cd command in Linux

While working within the terminal, moving around within directories is pretty much a necessity. The cd command is one of the important Linux commands you must know and it will help you to navigate through directories. Just type **cd** followed by directory as shown below.

```
root@ubuntu:~# cd <directory path>
```
Copy

```
root@ubuntu:~# pwd
/root
root@ubuntu:~# cd /etc/
root@ubuntu:/etc# pwd
/etc
root@ubuntu:/etc#
```

As you can see in the above command, I simply typed **cd /etc/** to get into the /etc directory. We used the **pwd command** to print the current working directory.

The mkdir command in Linux

The mkdir command allows you to create directories from within the terminal. The default syntax is **mkdir** followed by the directory name.

```
root@ubuntu:~# mkdir <folder name>
```
Copy

```
root@ubuntu:~# ls
root@ubuntu:~# mkdir JournalDev
root@ubuntu:~# ls
JournalDev
root@ubuntu:~#
```

As you can see in the above screenshot, we created the JournalDev directory with just this simple command.

Learn more about the mkdir command (Link to article)

The cp and mv commands

The cp and mv commands are equivalent to the copy-paste and cut-paste in Windows. But since Linux doesn't really have a command for renaming files, we also make use of the mv command to rename files and folders.

```
root@ubuntu:~# cp <source> <destination>
```
Copy

```
root@ubuntu:~# ls
Sample
root@ubuntu:~# cp Sample Sample-Copy
root@ubuntu:~# ls
Sample  Sample-Copy
root@ubuntu:~#
```

In the above command, we created a copy of the file named Sample. Let's see how what happens if we use the mv command in the same manner. For this demonstration, I'll delete the Sample-Copy file.

```
root@ubuntu:~# mv <source> <destination
```
Copy

```
root@ubuntu:~# ls
Sample
root@ubuntu:~# mv Sample Sample-Copy
root@ubuntu:~# ls
Sample-Copy
root@ubuntu:~#
```

In the above case, since we were moving the file within the same directory, it acted as rename. The file name is now changed.

Learn more about the cp command (Link to article) and mv command (Link to article).
The rm command in Linux

In the previous section, we deleted the Sample-Copy file. The rm command is used to delete files and folders and is one of the important Linux commands you must know.

```
root@ubuntu:~# rm <file name>
root@ubuntu:~# rm -r <folder/directory name>
```
Copy

```
root@ubuntu:~# ls
Sample-Copy
root@ubuntu:~# rm Sample-Copy
root@ubuntu:~# ls
root@ubuntu:~#
```

To delete a directory, you have to add the **-r** argument to it. Without the -r argument, rm command won't delete directories.

The touch command in Linux

To create a new file, the touch command will be used. The **touch** keyword followed by the file name will create a file in the current directory.

```
root@ubuntu:~# touch <file name>
```
Copy

```
root@ubuntu:~# ls
root@ubuntu:~# touch New-File
root@ubuntu:~# ls
New-File
root@ubuntu:~#
```

The cat, echo, and less commands

When you want to output the contents of a file, or print anything to the terminal output, we make use of the cat or echo commands. Let's see their basic usage. I've added some text to our New-File that we created earlier.

```
root@ubuntu:~# cat <file name>
root@ubuntu:~# echo <Text to print on terminal>
```
Copy

```
root@ubuntu:~# cat New-File
Hello, welcome to JournalDev. The one spot to learn everything related to programming.
root@ubuntu:~# echo New-File
New-File
root@ubuntu:~#
```

As you can see in the above example, the **cat command** when used on our "New-File", prints the contents of the file. At the same time, when we use **echo command**, it simply prints whatever follows after the command.

The **less command** is used when the output printed by any command is larger than the screen space and needs scrolling. The less command allows use to break down the output and scroll through it with the use of the enter or space keys.

The simple way to do this is with the use of the pipe operator (|).

```
root@ubuntu:~# cat /boot/grub/grub.cfg  | less
```
Copy

Learn more about the echo command(Link to article) and cat command(Link to article).

The man command in Linux

The man command is a very useful Linux command you must know. When working with Linux, the packages that we download can have a lot of functionality. Knowing it all is impossible.

The man pages offer a really efficient way to know the functionality of pretty much all the packages that you can download using the package managers in your Linux distro.

```
root@ubuntu:~# man <command>
```
Copy

The uname and whoami commands

The uname and whoami commands allow you to know some basic information which comes really handy when you work on multiple systems. In general, if you're working with a single computer, you won't really need it as often as someone who is a network administrator.

Let's see the output of both the commands and the way we can use these.

```
root@ubuntu:~# uname -a
```
Copy

```
root@ubuntu:~# uname -a
Linux ubuntu 4.15.0-74-generic #84-Ubuntu SMP Thu Dec 19 08:06:28 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
root@ubuntu:~# whoami
root
root@ubuntu:~#
```

The parameter **-a** which I've supplied to uname, stands for "all". This prints out the complete information. If the parameter is not added, all you will get as the output is "Linux".

The tar, zip, and unzip commands

The tar command in Linux is used to create and extract archived files in Linux. We can extract multiple different archive files using the tar command.

To create an archive, we use the -c parameter and to extract an archive, we use the -x parameter. Let's see it working.

```
#Compress
root@ubuntu:~# tar -cvf <archive name> <files seperated by space>
#Extract
root@ubuntu:~# tar -xvf <archive name>
```
Copy

```
root@ubuntu:~# tar -cvf Compress.tar New-File New-File-Link
New-File
New-File-Link
root@ubuntu:~# tar -xvf Compress.tar
New-File
New-File-Link
root@ubuntu:~# ls
```

In the first line, we created an archive named **Compress.tar** with the New-File and New-File-Link. In the next command, we have extracted those files from the archive.

Now coming to the zip and unzip commands. Both these commands are very straight forward. You can use them without any parameters and they'll work as intended. Let's see an example below.

```
root@ubuntu:~# zip <archive name> <file names separated by space>
root@ubuntu:~# unzip <archive name>
```
Copy

```
root@ubuntu:~# zip Sample.zip New-File New-File-Edited
updating: New-File (deflated 16%)
updating: New-File-Edited (deflated 19%)
root@ubuntu:~# unzip Sample.zip
Archive:  Sample.zip
replace New-File? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: New-File
  inflating: New-File-Edited
root@ubuntu:~#
```

Since we already have those files in the same directory, the unzip command prompts us before overwriting those files.

Learn more about the tar command (Link to article) and zip and unzip commands (Link to article)
The grep command in Linux
If you wish to search for a specific string within an output, the grep command comes into the picture. We can pipe (**|**) the output to the grep command and extract the required string.

```
root@ubuntu:~# <Any command with output> | grep "<string to find>"
```
Copy

```
root@ubuntu:~# cat New-File
Hello, welcome to JournalDev.
The one spot to learn everything related to programming.
Adding a few more lines
root@ubuntu:~# cat New-File | grep "learn"
The one spot to learn everything related to programming.
root@ubuntu:~#
```

This was a simple demonstration of the command. Learn more about the grep command (Link to article)
The head and tail commands

When outputting large files, the head and the tail commands come in handy. I've created a file named "Words" with a lot of words arranged alphabetically in it. The head command will output the first 10 lines from the file, while the tail command will output the last 10. This also includes any blank lines and not just lines with text.

```
root@ubuntu:~# head <file name>
root@ubuntu:~# tail <file name>
```
Copy

```
root@ubuntu:~# head Words
Carrot

Cave

Chair

Chess Board

Chief

root@ubuntu:~#
```

As you can see, the head command showed 10 lines from the top of the file.

The tail command outputted the bottom 10 lines from the file.

Learn more about the [tail command(Link to article)](#)
[The diff, comm, and cmp commands](#)

Linux offers multiple commands to compare files. The diff, comm, and cmp commands compare differences and are some of the most useful Linux commands you must know. Let's see the default outputs for all the three commands.

```
root@ubuntu:~# diff <file 1> <file 2>
```
Copy

As you can see above, I've added a small piece of text saying "This line is edited" to the New-File-Edited file.

```
root@ubuntu:~# cmp <file 1> <file 2>
```
Copy

```
root@ubuntu:~# cmp New-File New-File-Edited
New-File New-File-Edited differ: byte 112, line 3
root@ubuntu:~#
```

The cmp command only tells use the line number which is different. Not the actual text. Let's see what the comm command does.

```
root@ubuntu:~# comm <file 1> <file2>
```
Copy

```
root@ubuntu:~# comm New-File New-File-Edited
                Hello, welcome to JournalDev.
                The one spot to learn everything related to programming.
Adding a few more lines
comm: file 1 is not in sorted order
        Adding a few more lines - This line is edited
comm: file 2 is not in sorted order
root@ubuntu:~#
```

The text that's aligned to the left is the text that's only present in file 1. The center-aligned text is present only in file 2. And the right-aligned text is present in both the files.

By the looks of it, comm command makes the most sense when we're trying to compare larger files and would like to see everything arranged together.

The sort command in Linux

The sort command will provide a sorted output of the contents of a file. Let's use the sort command without any parameters and see the output.

The basic syntax of the sort command is:

```
root@ubuntu:~# sort <filename>
```
Copy

The ssh command allows us to connect to an external machine on the network with the use of the ssh protocol. The basic syntax of the ssh command is:

```
root@ubuntu:~ -->> ssh username@hostname
```
Copy

Learn more about ssh command(Link to article)

The service command in Linux is used for starting and stopping different services within the operating system. The basic syntax of the command is as below.

```
root@ubuntu:~ -->> service ssh status
root@ubuntu:~ -->> service ssh stop
root@ubuntu:~ -->> service ssh start
```
Copy

As you can see in the image, the ssh server is running on our system.

```
root@ubuntu:~ -->> mount /dev/cdrom /mnt


root@ubuntu:~ -->> df -h
```
Copy

In the above case, **/dev/cdrom** is the device that needs to be mounted. Usually, a mountable device is found inside the /dev folder. **/mnt** is the destination folder to mount the device to. You can change it to any folder you want but I've used /mnt as it's pretty much a system default folder for mounting devices.
To see the mounted devices and get more information about them, we make use of the df command. Just typing **df** will give us the data in bytes which is not readable. So we'll use the **-h** parameter to make the data human-readable.

```
root@ubuntu:~ -->> df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            210M     0  210M   0% /dev
tmpfs            49M  892K   48M   2% /run
/dev/vda1       9.8G  7.0G  2.4G  75% /
tmpfs           241M  8.0K  241M   1% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           241M     0  241M   0% /sys/fs/cgroup
tmpfs            49M   16K   49M   1% /run/user/120
tmpfs            49M     0   49M   0% /run/user/0
root@ubuntu:~ -->>
```

Learn more about the df command(Link to article)
The chmod and chown commands

The chmod and chown commands give us the functionality to change the file permissions and file ownership are the most important Linux commands you should know.

The main difference between the functions of the two commands is that the chmod command allows changing file permissions, while chown allows us to change the file owners.

The default syntax for both the commands is **chmod <parameter> filename** and **chown user:group filename**

```
root@ubuntu:~ -->> chmod +x loop.sh
root@ubuntu:~ -->> chmod root:root loop.sh
```
Copy

```
root@ubuntu:~ -->> ls -l loop.sh
-rw-r--r-- 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> chmod +x loop.sh
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->>
```

In the above example, we're adding executable permissions to the loop.sh file with the **chmod command**. Apart from that, with the **chown command**, we've made it accessible only by root user and users within the root group.

```
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> chown www-data:www-data loop.sh
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 www-data www-data 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->>
```

As you will notice, the **root root** part is now changed to **www-data** which is the new user who has full file ownership.
Learn more about the chmod command(Link to article) and chown command (Link to article)
The ifconfig and traceroute commands

Moving on to the networking section in Linux, we come across the ifconfig and traceroute commands which will be frequently used if you manage a network.

The ifconfig command will give you the list of all the network interfaces along with the IP addresses, MAC addresses and other information about the interface.

```
root@ubuntu:~ -->> ifconfig
```
Copy

There are multiple parameters that can be used but we'll work with the basic command here.

```
root@ubuntu:~ -->> ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:3b:09:02:00  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

When working with traceroute, you can simply specify the IP address, the hostname or the domain name of the endpoint.

```
root@ubuntu:~ -->> traceroute <destination address>
```
Copy

```
root@ubuntu:~# traceroute localhost
traceroute to localhost (127.0.0.1), 30 hops max, 60 byte packets
 1  localhost (127.0.0.1)  0.029 ms  0.007 ms  0.006 ms
root@ubuntu:~#
```

Now obviously, localhost is just one hop (which is the network interface itself). You can try this same command with any other domain name or IP address to see all the routers that your data packets pass through to reach the destination.

Learn more about the ifconfig command(Link to article)
The wget command in Linux

If you want to download a file from within the terminal, the wget command is one of the handiest command-line utilities available. This will be one of the important Linux commands you should know when working with source files.

When you specify the link for download, it has to directly be a link to the file. If the file cannot be accessed by the wget command, it will simply download the webpage in HTML format instead of the actual file that you wanted.

Let's try an example. The basic syntax of the wget command is :

```
root@ubuntu:~ -->> wget <link to file>
OR
root@ubuntu:~ -->> wget -c <link to file>
```
The sudo command in Linux
"With great power, comes great responsibility"

This is the quote that's displayed when a sudo enabled user(sudoer) first makes use of the sudo command to escalate privileges. This command is equivalent to having logged in as root (based on what permissions you have as a sudoer).

```
non-root-user@ubuntu:~# sudo <command you want to run>
Password:
```
Copy

Just add the word **sudo** before any command that you need to run with escalated privileges and that's it. It's very simple to use, but can also be an added security risk if a malicious user gains access to a sudoer.
Learn more about the sudo command (Link to article)
The cal command in Linux

Ever wanted to view the calendar in the terminal? Me neither! But there apparently are people who wanted it to happen and well here it is.

The **cal** command displays a well-presented calendar on the terminal. Just enter the word cal on your terminal prompt.

```
root@ubuntu:~# cal
root@ubuntu:~# cal May 2019
```
Copy

```
root@ubuntu:~# cal
      January 2020
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

root@ubuntu:~# cal May 2019
      May 2019
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

root@ubuntu:~#
```

Even though I don't need it, it's a really cool addition! I'm sure there are people who are terminal fans and this is a really amazing option for them.

The alias command
Do you have some commands that you run very frequently while using the terminal? It could be **rm -r** or **ls -l**, or it could be something longer like **tar -xvzf**. This is one of the productivity-boosting Linux commands you must know.

If you know a command that you run very often, it's time to create an alias. What's an alias? In simple terms, it's another name for a command that you've defined.

```
root@ubuntu:~# alias lsl="ls -l"
OR
root@ubuntu:~# alias rmd="rm -r"
```
Copy

Now every time you enter **lsl** or **rmd** in the terminal, you'll receive the output that you'd have received if you had used the full commands.

The examples here are for really small commands that you can still type by hand every time. But in some situations where a command has too many arguments that you need to type, it's best to create a shorthand version of the same.

Learn more about [alias command (LInk to article)](#)
[The dd command in Linux](#)

This command was created to convert and copy files from multiple file system formats. In the current day, the command is simply used to create bootable USB for Linux but there still are some things important you can do with the command.

For example, if I wanted to back up the entire hard drive as is to another drive, I'll make use of the dd command.

```
root@ubuntu:~# dd if = /dev/sdb of = /dev/sda
```
Copy

The **if** and **of** arguments stand for **input file** and **output file**.
[The whereis and whatis commands](#)

The names of the commands make it very clear as to their functionality. But let's demonstrate their functionality to make things more clear.

The **whereis** command will output the exact location of any command that you type in after the **whereis** command.

```
root@ubuntu:~# whereis sudo
sudo: /usr/bin/sudo /usr/lib/sudo /usr/share/man/man8/sudo.8.gz
```
Copy

The **whatis** command gives us an explanation of what a command actually is. Similar to the whereis command, you'll receive the information for any command that you type after the **whatis** command.

```
root@ubuntu:~# whatis sudo
sudo (8) - execute a command as another user
```
Copy

[The top command in Linux](#)

A few sections earlier, we talked about the ps command. You observed that the ps command will output the active processes and end itself.

The top command is like a CLI version of the task manager in Windows. You get a live view of the processes and all the information accompanying those processes like memory usage, CPU usage, etc.

To get the top command, all you need to do is type the word **top** in your terminal.

```
top - 20:41:20 up 6 days, 17:42,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 124 total,   1 running,  86 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  1.0 sy,  0.0 ni, 96.9 id,  1.3 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   492644 total,    11616 free,   387228 used,    93800 buff/cache
KiB Swap:        0 total,        0 free,        0 used.    77676 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0  225344   4428   2056 S   0.0  0.9   1:27.94 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.11 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0.0  0.0   0:12.93 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:50.10 rcu_sched
    9 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:01.81 watchdog/0
```

The useradd and usermod commands

The **useradd** or **adduser** commands are the exact same commands where adduser is just a symbolic link to the useradd command. This command allows us to create a new user in Linux.

```
root@ubuntu:~# useradd JournalDev -d /home/JD
```
Copy

The above command will create a new user named JournalDev with the home directory as **/home/JD**.
The **usermod** command, on the other hand, is used to modify existing users. You can modify any value of the user including the groups, the permissions, etc.

For example, if you want to add more groups to the user, you can type in:

```
root@ubuntu:~# usermod JournalDev -a -G sudo, audio, mysql
```
Copy

Learn more on how to create and manage users on Linux (Link to article)

The passwd command in Linux

Now that you know how to create new users, let's also set the password for them.
The **passwd** command lets you set the password for your own account, or if you have the permissions, set the password for other accounts.

The command usage is pretty simple:

```
root@ubuntu:~# passwd
New password:
```
Copy

```
root@ubuntu:~# passwd
New password:
Retype new password:
passwd: password updated successfully
root@ubuntu:~#
```

If you add the username after **passwd**, you can set passwords for other users. Enter the new password twice and you're done. That's it! You will have a new password set for the user!

Linux File Ownership

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user- group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a

group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your linux computer, to view your images. This is where **Permissions** set in, and they define **user behavior**.

Let us understand the **Permission system** on Linux.

Linux File Permissions

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
- **Write:** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the

execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

Owners assigned Permission On Every File and Directory

Read
Write
Execute
User

Read
Write
Execute
Group

Read
Write
Execute
All

File Permissions in Linux/Unix
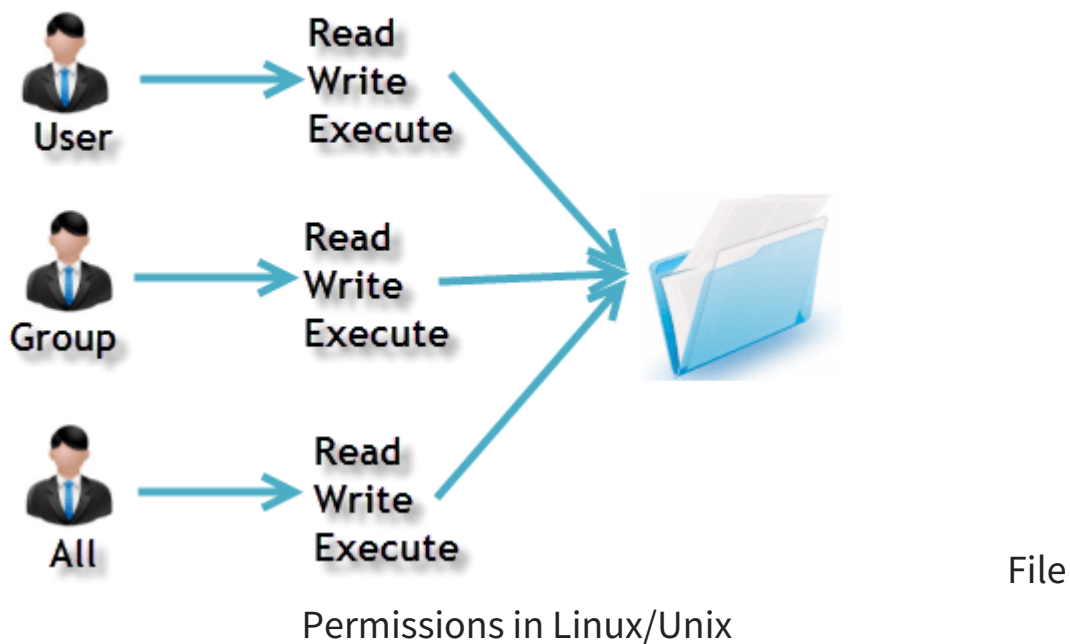
Let's see file permissions in Linux with examples:

ls – l on terminal gives

```
ls - l
```

File type and Access Permissions.

home@VirtualBox: ~

home@VirtualBox:~$ ls -l
-rw-rw-r-- 1 home home   0 2012-08-30 19:06 My File

Here, we have highlighted **'-rw-rw-r–'**and this weird looking code is the one that tells us about the Unix permissions given to the owner, user group and the world.

Here, the first '**–**' implies that we have selected a file.p>

- rw-rw-r--

**indicates
file**

Else, if it were a directory, **d** would have been shown.

d represents directory

drwxr-xr-x 2 ubuntu ubuntu 80 Sep 6 07:27 Desktop

The characters are pretty easy to remember.

  r = read permission
  w = write permission
  x = execute permission
  – = no permission

Let us look at it this way.

The first part of the code is **'rw-'**. This suggests that the owner 'Home' can:

- rw-rw-r--

**no execute
permission**

- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

By design, many Linux distributions like Fedora, CentOS, ubuntu etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.

The second part is **'rw-'.** It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says **'r–'.** This means the user can only:

- Read the file
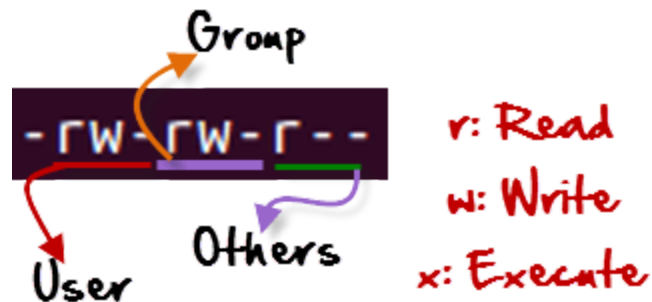
Changing file/directory permissions in Linux Using 'chmod' command

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

Syntax:

```
chmod permissions filename
```
There are 2 ways to use the command

1. **Absolute mode**
2. **Symbolic mode**

Absolute(Numeric) Mode in Linux

In this mode, file **permissions are not represented as characters but a three-digit octal number**.

The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	—
1	Execute	–x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r–
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Let's see the chmod permissions command in action.

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.
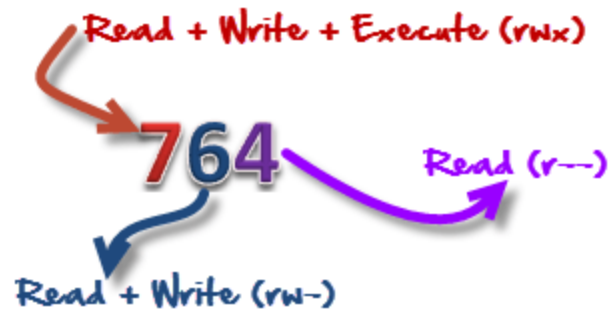
'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- World can only read

This is shown as '-rwxrw-r–

This is how you can change user permissions in Linux on file by assigning an absolute number.

Symbolic Mode in Linux

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the Unix file permissions.

Operator	Description
+	Adds a permission to a file or directory
–	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as –

User Denotations	
u	user/owner
g	group

o	other
a	all

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example

Changing Ownership and Group in Linux

For changing the ownership of a file/directory, you can use the following command:

```
chown user filename
```

In case you want to change the user as well as group for a file or directory use the command

```
chown user:group filename
```

Let's see this in action

Check the current file ownership using ls -l

```
-rw-rw-r--    1 root n10      18 2012-09-16 18:17 sample.txt
```

Change the file owner to n100 . You will need sudo

```
n10@N100:~$ sudo chown n100 sample.txt
```

Ownership changed to n100

```
-rw-rw-r--    1 n100 n10      18 2012-09-16 18:17 sample.txt
```

Changing user and group to root 'chown user:group file'

```
n10@N100:~$ sudo chown root:root sample.txt
```

User and Group ownership changed to root

```
-rw-rw-r--    1 root root      18 2012-09-16 18:17 sample.txt
```

In case you want to change group-owner only, use the command

```
chgrp group_name filename
```
'**chgrp**' stands for change group.

Check the current file ownership using ls -dl

```
guru99@VirtualBox:~$ ls -dl test1
-rwxrwxrwx 1 root cdrom 0 Oct  6 11:27 test1
```

Change the file owner to root . You will need sudo

```
guru99@VirtualBox:~$ sudo chgrp root test1
```

Group Ownership changed to root

```
guru99@VirtualBox:~$ ls -dl test1
-rwxrwxrwx 1 root root 0 Oct  6 11:27 test1
```

Tip

- The file /etc/group contains all the groups defined in the system

- You can use the command "groups" to find all the groups you are a member o

```
guru99@VirtualBox:~$ groups
cdrom guru99 adm sudo dip plugdev lpadmin sambashare
guru99@VirtualBox:~$ █
```

- You can use the command newgrp to work as a member a group other than your default group

```
guru99@VirtualBox:~$ newgrp cdrom
guru99@VirtualBox:~$ cat > test
this is a test to change group
^C
guru99@VirtualBox:~$ ls -dl test
-rw-rw-r-- 1 guru99 cdrom 31 Oct 11 16:39 test
guru99@VirtualBox:~$ █
```

- You cannot have 2 groups owning the same file.
- You do not have nested groups in Linux. One group cannot be sub-group of other
- x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-dirs