Some Important Directories

- Home Directories: `/root`,`/home/username`
- User Executable: `/bin`, `/usr/bin`, `/usr/local/bin`
- System Executables: `/sbin`, `/usr/sbin`, `/usr/local/sbin`
- Other Mountpoints: `/media`, `/mnt`
- Configuration: `/etc`
- Temporary Files: `/tmp`
- Kernels and Bootloader: `/boot`
- Server Data: `/var`, `/srv`
- System Information: `/proc`, `/sys`
- Shared Libraries: `/lib`, `/usr/lib`, `/usr/local/lib`

-
- **whoami**: user name will be displayed
- **pwd**: present working directory
- **sudo -i:** switch to root user
- **#**: root user shell, **$**: normal user shell
- **/root:** home directory of root user, **/:** root directory (both are different)
  - **sudo -i**(from vagrant user's shell)**:** takes to **/root** directory
  - **cd /:** takes to the root directory
- ---
- **touch:** create empty file
  - **touch <file_name.extension_name>**: create one file of the file name type
  - **touch myfile{1..5}.txt** : create 5 files I.e. myfile1.txt, .. myfile5.txt
- **mkdir:** create empty folder
  - **mkdir f1/f2/f3/f4/f5/f6**
    - If f1 or f2 is not exist, then it'll not create all the folder
    - use **-p** flag in this case

```
[root@vbox ~]# mkdir /opt/dev/ops/devops/test
mkdir: cannot create directory '/opt/dev/ops/devops/test': No such file or directory
[root@vbox ~]# mkdir -p /opt/dev/ops/devops/test
[root@vbox ~]#
```

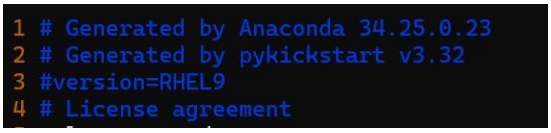
- **cp <source> <destination>** : copy
  - **cp mydevfile.txt dev/**
  - **cp mydevfiles{1..5}.txt dev/** : all 5 files will be copied
  - If you want to copy directories then use the flag -r
    - **cp -r <dir1> <destination>**

- **mv <source> <destination>** : move
  - No need to specify **-r** here to move directories. You can directly move the directories here.
  - Ex: **mv /home/vagrant/devtemp2 /home/vagrant/dev**
  - **mv *.txt testdir/** :move all txt files to testdir directory
- **rm <file_path>** :delete a file
  - If you want to delete a directory: **rm -r <directory>**

- **VIM EDITOR:**
- **vim <file_name>** :create and open a file
  - 3 modes:
    - command mode: default, (after pressing escape key)
    - insert mode(edit mode): pressing **i** key
    - extended command mode: pressing **Esc** key then **:(colon)**key
      - w: write/save, q: quit

- If you change something but don't want to save it and quit,
  - **:q** will give error,
  - **:q!** will do the work.
  - **!** means **forcefully** do the things

➢ **:se nu**: set line numbers



- To go to last line: **shift+g** or **G** (capslock on)
- To go to first line: **gg** (small g+small g)
- To copy the line: **yy**
  - If you want to copy multiple lines: **<number>yy**
    - Ex: **4yy**: 4 lines will be copied starting from the line where cursor is present
- To paste the copied line:
  - **p** (small p): paste below
  - **P** (capital P): paste above the cursor
- To cut:
  - Same as copy
    - **dd**: cut
    - **4dd**: cut4 lines
- To undo: **u**
- To search any word: **/<keyword>:**Case-sensitive
  - To go to next keyword, presss "**n**"
- **yy, p/P: copy paste, dd, p/P: cut paste**

gg	To go to the beginning of the page
G	To go to end of the page
w	To move the cursor forward, word by word
b	To move the cursor backward, word by word
nw	To move the cursor forward to n words **(SW)**
nb	To move the cursor backward to n words **{SB)**
u	To undo last change (word)

➢

➢

U	To undo the previous changes (entire line)
Ctrl+R	To redo the changes
VY	To copy a line
nyy	To copy n lines (Syy or 4yy)
p	To paste line below the cursor position
P	To paste line above the cursor position
dw	To delete the word letter by letter {like Backspace}
x	To delete the world letter by letter (like DEL Key)
dd	To delete entire line
ndd	To delete n no. of lines from cursor position{Sdd)
I	To search a word in the file

➢

- **Use this command to link a path. (just like shortcut app in windows)**

```
[root@vbox ~]# vim /opt/dev/ops/devops/test/commands.txt
[root@vbox ~]# ln -s /opt/dev/ops/devops/test/commands.txt cmds
```

- To unlink the link:
  - **unlink <link>**

```
[root@vbox ~]# ln -s /opt/dev/ops/devops/test/commands.txt cmds
[root@vbox ~]# ls -l
total 8
-rw-------. 1 root root 2027 Dec 18  2023 anaconda-ks.cfg
lrwxrwxrwx. 1 root root   37 Jan 26 17:56 cmds -> /opt/dev/ops/devops/test/commands.txt
-rw-------. 1 root root 1388 Dec 18  2023 original-ks.cfg
[root@vbox ~]# unlink cmds
[root@vbox ~]# ls
anaconda-ks.cfg  original-ks.cfg
[root@vbox ~]# ls -l
total 8
-rw-------. 1 root root 2027 Dec 18  2023 anaconda-ks.cfg
-rw-------. 1 root root 1388 Dec 18  2023 original-ks.cfg
[root@vbox ~]#
```

- To search any keyword:
  - **grep <keyword> <filepath>**

```
[root@vbox ~]# ls
anaconda-ks.cfg  grep-test.txt  inputred.txt
[root@vbox ~]# grep 'unix' grep-test.txt
unix ahasfs lafjfa funix
sfd lfk;jaf unix ajf faafuaf alok unix
jf sal;faslok alon aslok alok unix unix
unix linux alok ranjan joshi alok unix
```

    - It is case sensitive, if you don't want case sensitivity then mention **-i**.
    - To check inside a directory, use the flag **-R**
      - **(image in next page ---------->)**

```
[root@vbox ~]# grep SELINUX -R /etc/*
/etc/selinux/config:# SELINUX= can take one of these three values:
/etc/selinux/config:# NOTE: In earlier Fedora kernel builds, SELINUX=disabled would also
/etc/selinux/config:SELINUX=permissive
/etc/selinux/config:# SELINUXTYPE= can take one of these three values:
/etc/selinux/config:SELINUXTYPE=targeted
/etc/selinux/targeted/contexts/x_contexts:property _SELINUX_*                    system_u:object_r:seclabel_xproperty_t:s0
/etc/selinux/.config_backup:# BACKUP_SELINUX= can take one of these three values:
/etc/selinux/.config_backup:# NOTE: In earlier Fedora kernel builds, BACKUP_SELINUX=disabled would also
/etc/selinux/.config_backup:BACKUP_SELINUX=permissive
/etc/selinux/.config_backup:# BACKUP_SELINUXTYPE= can take one of these three values:
/etc/selinux/.config_backup:BACKUP_SELINUXTYPE=targeted
/etc/sysconfig/selinux:# SELINUX= can take one of these three values:
/etc/sysconfig/selinux:# NOTE: In earlier Fedora kernel builds, SELINUX=disabled would also
/etc/sysconfig/selinux:SELINUX=permissive
/etc/sysconfig/selinux:# SELINUXTYPE= can take one of these three values:
/etc/sysconfig/selinux:SELINUXTYPE=targeted
[root@vbox ~]# grep SELINUX /etc/*
grep: /etc/alternatives: Is a directory
grep: /etc/audit: Is a directory
grep: /etc/authselect: Is a directory
grep: /etc/bash_completion.d: Is a directory
grep: /etc/binfmt.d: Is a directory
grep: /etc/cifs-utils: Is a directory
grep: /etc/cockpit: Is a directory
grep: /etc/cron.d: Is a directory
grep: /etc/cron.daily: Is a directory
grep: /etc/cron.hourly: Is a directory
grep: /etc/cron.monthly: Is a directory
grep: /etc/cron.weekly: Is a directory
grep: /etc/crypto-policies: Is a directory
```

    - **-v** keyword is used to see the files where the keyword is not present.
- **head** and **tail:**
  - **head:** first 10 lines of the file**, tail:** Last 10 lines of the file
    - **-f** is used to see if the logs are getting changed
    - **head -20 <file_path>** : first 20 line you'll be able to see.

- **Cut** command:

```
[root@vbox ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
unbound:x:999:999:Unbound DNS resolver:/etc/unbound:/sbin/nologin
systemd-coredump:x:998:996:systemd Core Dumper:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:997:995:User for polkitd:/:/sbin/nologin
libstoragemgmt:x:996:992:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
cockpit-ws:x:995:991:User for cockpit web service:/nonexisting:/sbin/nologin
cockpit-wsinstance:x:994:990:User for cockpit-ws instances:/nonexisting:/sbin/nologin
tss:x:59:59:Account used for TPM access:/dev/null:/sbin/nologin
sssd:x:993:989:User for sssd:/:/sbin/nologin
setroubleshoot:x:992:988::/var/lib/setroubleshoot:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/usr/share/empty.sshd:/sbin/nologin
chrony:x:991:987::/var/lib/chrony:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
systemd-oom:x:985:985:systemd Userspace OOM Killer:/:/usr/sbin/nologin
systemd-resolve:x:984:984:systemd Resolver:/:/usr/sbin/nologin
vagrant:x:1000:1000::/home/vagrant:/bin/bash
vboxadd:x:983:1::/var/run/vboxadd:/bin/false
```

  - See this file, here in each line **:** is the separator present.
  - It's just like the split function in javascript, that split each line with the separator.
  - **cut -d: -f4 /etc/passwd**
    - **-d:** =>separator is colon
    - **f4** => which field (in js index) you want to see after split (here **4**)

```
[root@vbox ~]# cut -d: -f1 /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
```

```
[root@vbox ~]# cut -d: -f4 /etc/passwd
0
1
2
4
7
0
0
0
12
0
```

    - Here delimeter is required (-d)

- **awk** command:

```
[root@vbox ~]# awk -F':' '{print $4}' /etc/passwd
0
1
2
4
7
0
0
0
```

  - **-F'<separator>**

- **'<command>'**

```
[root@vbox ~]# awk -F ':' '{print NR" "$1" "$4}' separatorfile.txt
1 value_1_1 value_1_4
2 value_2_1 value_2_4
3 value_3_1 value_3_4
4 value_4_1 value_4_4
5 value_5_1 value_5_4
6 value_6_1 value_6_4
7 value_7_1 value_7_4
8 value_8_1 value_8_4
```

```
value_49_1:value_49_2:value_49_3:value_49_4:value_49_5:value_49_6:value_49_7:value_49_8:value_49_9:value_4
value_50_1:value_50_2:value_50_3:value_50_4:value_50_5:value_50_6:value_50_7:value_50_8:value_50_9:value_5
[root@vbox ~]# awk -F '_' 'BEGIN {print "Addition starts"} {sum += $2} END {print sum}' separatorfile.txt
Addition starts
1275
```

- `w` (write): Saves the changes to the file.
- `q` (quit): Exits the editor.
- `h` (help): Displays a brief help message.
- `j` (join): Joins two lines into one.
- `d` (delete): Deletes lines.
- `s` (substitute): Makes substitutions on a line.
- `g` (global): Applies commands globally.
- `a` (append): Appends text after the current line.
- `i` (insert): Inserts text before the current line.
- `/string` : Search for a string
- `?string?` : Reverse search

➢

➢

➢ **<u>To replace any keyword in any file:</u>**

- Method-1:
  - Open file using **vim**, in extended command mode write: **:%s/<current keyword>/<new keyword>** (note: this will replace only one word per line)

`:%s/foo/bar/g` to replace all 'foo' with 'bar'

- `:` Enters command mode
- `%` Means across all lines
- `s` Means substitute
- `/foo` is regex to find things to replace
- `/bar/` is regex to replace things with
- `/g` means global, otherwise it would only execute once per line

  -

```
[root@vbox ~]# cat sample-file.txt
bcdef flasfkjf; fflaf sfaj flka afjaf alsfa; fa coronavirus
jlfsa alkfjd fccf coronavirus lksdfj falfjla fdf l klfjasfd fc coronavirus ljfdsa ;ffljf fsac coronavirus
laksjff fjlkaf; jlkf coronavirus flkjsaf lkjfja sfa;fkjlaf aaf;jas coronavirus
fsfjkl coronavirus fdajf fljf coronavirus skfdjslf f;fj ffafj coronavirus fslkjfsf coronavirus
```

  - Here, coronavirus is the keyword to be replaced.
  - After executing that command, only first keyword of each line that is "coronavirus" will be replaced. If any more keyword is there in the same line, they won't be replaced.
  - If you want to replace the words of first 3 lines only then:
    - **:1,3s/<old_word>/<new_word>/g**
  - **:%s/<current keyword>/<new keyword>/g** after adding **/g** all the keyword will be replaced.
  - **:%s/<current keyword>//g** if you want to replace it with nothing
- **Method 2: (important)**
  - If there is multiple file that has to be changed, then method-1 will not be useful.
  - **sed** command will be used for this.
  - **sed '/s/<keyword>/<new_keyword>/g' <filepath>**
    - Here in case of **filepath** you can give **/*.txt** kind of thing, every txt file will be change

```
[root@vbox ~]# sed 's/covid19/coronavirus/g' sample-file.txt
bcdef flasfkjf; fflaf sfaj flka afjaf alsfa; fa coronavirus
jlfsa alkfjd fccf coronavirus lksdfj falfjla fdf l klfjasfd fc coronavirus ljfdsa ;ffljf fsac coronavirus
laksjff fjlkaf; jlkf coronavirus flkjsaf lkjfja sfa;fkjlaf aaf;jas coronavirus
fsfjkl coronavirus fdajf fljf coronavirus skfdjslf f;fj ffafj coronavirus fslkjfsf coronavirus
```

- Here it doesn't change the orignial file, it will only display how the file will look like after replacing the keyword.

- Without **-i** flag, it won't replace. Correct command is:
  - **sed -i '/s/<keyword>/<new_keyword>/g' <filepath>**

➢ **Redirection:** **>**, **>>** (output redirection) and **<**, **<<** (input redirection)

- Output redirection: transfer the output of a command to a file
- Input redirection: get the file content of a file as input of a command.
- If you don't want to see the output of any command in the screen and you want to transfter the output to another file, then it is used.
- **ls > <filepath>** => it will transfer the output that ls would have generated to the file which path is mentioned.
  - If the file doesn't exist, it'll create one and transfer the content.
  - If the file exists, it'll override the file.
- **ls >> <filepath>** => It'll not override, rather it'll append the content

```
[root@vbox ~]# ls
anaconda-ks.cfg  original-ks.cfg  redirection-file.txt  sample-file.txt
[root@vbox ~]# ls >> redirection-file.txt
[root@vbox ~]# cat redirection-file.txt
It is a redirection file...

hahahahahahahahahaha
anaconda-ks.cfg
original-ks.cfg
redirection-file.txt
sample-file.txt
```

- >> means **1>>** (default behaviour) means **output redirection**
- If you want **error redirection** then **2>>**

```
[root@vbox ~]# lssssssss >> redirection-file.txt
-bash: lssssssss: command not found
[root@vbox ~]# lssssssss 2>> redirection-file.txt
[root@vbox ~]# cat redirection-file.txt
It is a redirection file...

hahahahahahahahahaha
anaconda-ks.cfg
original-ks.cfg
redirection-file.txt
sample-file.txt
-bash: lssssssss: command not found
```

- ***&>> is for both Output and Error redirection.***
- There is a file **/dev/null**, which is empty. If you redirect anything to this file-path, the output will neither be generated in the terminal nor in this file. If you redirect the content of this file to another file then the new will won't have any content.

```
[vagrant@vbox ~]$ wc /etc/passwd
  30   62 1561 /etc/passwd
[vagrant@vbox ~]$ wc < /etc/passwd
  30   62 1561
```

- (line words characters file-path) output of **wc** command

- In first command: **wc** (word count) command gets the file as input, so it has the knowledge about the file. So it prints the filename
- In second command: **wc** command gets the content of the file as input, so it doesn't know about the filename.
- This is the use of input redirection.

➢ **Pipe: |**

- **wc** => used to count something in a file. (**-l** to count number of lines)
-
```
[root@vbox ~]# wc -l separatorfile.txt
50 separatorfile.txt
```
-
```
[vagrant@vbox ~]$ wc /etc/passwd
  30   62 1561 /etc/passwd
```
  - Lines, words, characters, filepath
-
```
[vagrant@vbox ~]$ wc -l < /etc/passwd
30
[vagrant@vbox ~]$ wc < /etc/passwd
  30   62 1561
```
- Symbol of Pipe is: |
-
```
[vagrant@vbox etc]$ ls | wc -l
187
```
  - Here **ls** generate the list of the file names inside the etc directory.
  - This will be used as the input of **wc -l**
  - **wc -l** takes the file content (in case of input redirection) and gives the number of lines, so think output of **ls** as the file content.
-
```
[root@vbox ~]# ls /etc/ > temp.txt
[root@vbox ~]# wc -l < temp.txt
187
```
  - This is the same as the previous pipe example.
-
```
[root@vbox etc]# ls | grep host
host.conf
hostname
hosts
```
-
```
[root@vbox ~]# wc < /etc/passwd
  33   65 1688
[root@vbox ~]# cat /etc/passwd | wc
     33      65    1688
```
(both same)
-
  - Search all the files that contains **host** keyword in it.

➢ **Find:**

- It is used to find any file
-
```
[root@vbox ~]# find /etc -name host*
/etc/host.conf
/etc/hosts
/etc/hostname
```

input redirection, xargs, pipe

- **Input Redirection:**
  - You can feed some content of file (or) some multiple lines (or) one single line as the **stdin** to a **command** using input redirection.
    - For content of file, <
    - For multiple lines, <<placeholder (generally we write **EOF** as placeholder)

      Line1
      Line2
      placeholder
    - For single string literal, <<< (it is called **here-string**)
  - **Taking content of file as stdin**
    - It converts the contents of a file to **stdin** (standard input).
    - Commands can take 2 type of inputs: **argument** or **stdin**.
    - If the command supports **stdin**, then only **input redirection** is helpful. Otherwise **xargs** will be used.
    - **cat** command takes both **stdin** as well as **arguments**.
    - ```
      [root@ip-172-31-20-103 test]# cat f1.txt
      abcd
      efgh
      ijkl
      mnop
      ```
      - Here **cat** command used **argument** type of input (f1.txt).
    - ```
      [root@ip-172-31-20-103 test]# cat < f1.txt
      abcd
      efgh
      ijkl
      mnop
      ```
      - Here cat command used **stdin** type of input (< f1.txt)
    - When you just enter the command and you'll be asked to enter something, then you can get to know that the command takes **stdin** as input.
    - ```
      [root@ip-172-31-20-103 test]# cat
      abcdef
      abcdef
      4347834
      4347834
      ```
      - Here you can see, I only wrote **cat**, and whatever I wrote in the terminal, **cat** command took that as **stdin** and printed that again.
      - Its just like, a loop of **scanf** and **printf** (in context of **C** programming)
    - ```
      [root@ip-172-31-20-103 test]# ls
      f1.txt
      [root@ip-172-31-20-103 test]# cat f2.txt
      cat: f2.txt: No such file or directory
      ```
      - You can see, there is no file named as *f2.txt*, and when I executed **cat f2.txt**, one error was displayed.
      - And see, who threw the error. Its **cat** command it self.
    - ```
      [root@ip-172-31-20-103 test]# cat < f2.txt
      -bash: f2.txt: No such file or directory
      ```
      - In here, I used **input redirection**, and here the error was thrown by **bash**.
    - So, when you write **command filename**, the **command** will try to open that file and do the task. If the file doesn't exist then the **command will throw error**.
      ```

- If you write **command < filename**, here **bash** will try to open the file and feed the **content of the file** to the **command**. If the file doesn't exist, then **bash will throw the error**.
- The above examples are of taking **file contents as stdin**, here we used single angular bracket i.e. **<**

- **Taking multiple lines as stdin**
  -
```
[root@ip-172-31-20-103 test]# cat <<eof
> abcd
> efgh
> ijkl
> eof
abcd
efgh
ijkl
```
    - That **eof** is just one convention. Anything you can write in that place.
    - When you write that again after writing the multiple lines, the command executes.
  - **wc** command displays the **row counts, word counts, character counts, filename**
    - Filename will be visible if **wc** command is executed by giving filename as argument.
    - If you have used **stdin** type of input then filename will not be displayed (because of course it has no idea about the file *:)* …..)
    -
```
[root@ip-172-31-20-103 test]# cat f1.txt
abcd
efgh ujkl
dfadflj kljf dfkldf lkfjd
[root@ip-172-31-20-103 test]# wc f1.txt
 3  7 41 f1.txt
[root@ip-172-31-20-103 test]# wc < f1.txt
 3  7 41
```
    -
```
[root@ip-172-31-20-103 test]# wc <<hello
> f flkd fljf
> fdljk f
> jfkd dfdlfjaf oruf  lsfjs jlf
> hello
       3       10       50
```
      - Here **input redirection for multiple lines** is used.
      - I used **hello** as the keyword that denotes end of lines.
    -
```
[root@ip-172-31-20-103 test]# wc
fdl
jl ore hfd
jfdl
fd rp lf lhf ljf
       4       10       39
```
      - Here I didn't use any **input redirection**.
      - I just used standard input (**stdin**) that I entered by my own.
      - Note: use **ctrl+d** (not *ctrl + c*)
      - **ctrl+c** will kill the process, where **ctrl+d** tells the command like *"it is the end of giving input, now you execute yourself"*

```
[root@ip-172-31-20-103 test]# cat <<end > ftemp.txt
> kldfjd
> kljkfl lkfjf f;lk fjs
> klf dkljfslf s
> ljkfs fjls;a
> ljl fjdf
> end
```

- If you want to use both **input redirection** and **output redirection** at once then you can use like this.

```
[root@ip-172-31-20-103 test]# cat <<end
> jlk;sjf
>  jhf fldfj f ljsjf
> flksf f lkjsdlf
> jfldfs
> end > ftemp.txt
```

- But its wrong

- One more use case:

```
[root@ip-172-31-20-103 test]# ls | grep 'f1'
f1.txt
[root@ip-172-31-20-103 test]# grep 'some' f1.txt
```

  - grep supports both type of inputs i.e. **stdin** and **arguments.**
  - First case it used **stdin** (because **pipe sends the output of current command as stdin to another**) and second case it used **argument**.

- Find the below use-case **(IMPORTANT)**

```
[root@ip-172-31-20-103 test]# grep '.txt' <<end
> $(ls)
> end
f1.txt
f2.txt
f3.txt
f4.txt
```

  - Here I wrote **$(ls)** because **$()** is used to execute some command.
  - If I only write **ls** in there then it'll take **ls** as a normal keyword.

```
[root@ip-172-31-20-103 test]# grep '.txt' <<< $(ls)
f1.txt
f2.txt
f3.txt
f4.txt
```

  - It might be confusing because **<<<** is used for single string input redirection and **ls** gives multiple lines.
  - But if you try to print **echo $(ls)** it'll print in a single string.
  - So result of **$(ls)** will be single line only.

```
[root@ip-172-31-20-103 test]# cat <<< $'hello\nhieeee\nbyeeeee'
hello
hieeee
byeeeee
```

  - Here you can see I used **single string** to print **multiple lines.**
  - **$** with **single quote** is used where you are using **excape characters** like \n, \t etc (**double quote** doesn't work here)

- **Double quote** can be used for **variable expansion;** but for **escape characters** we need to use **single quote** with **$**

- **here-string**
  - To get single line as **stdin**, you need to use **<<<**
  - **(IMPORTANT)** You need to give " or ' (double quote or single quote) otherwise only first keyword will be taken as **stdin input** and remaining will be taken as **filename.**

```
[root@ip-172-31-20-103 test]# cat <<< alok ranjan joshi
cat: ranjan: No such file or directory
cat: joshi: No such file or directory
```

    - Here it'll take first word i.e. *alok* as the stdin and other 2 i.e. *ranjan* and *joshi* as filenames.

```
[root@ip-172-31-20-103 test]# cat <<< "hi hello bye"
hi hello bye
```

    - Here I used cat command to print that.

```
[root@ip-172-31-20-103 test]# touch <<< "f1.txt f2.txt f3.txt"
touch: missing file operand
Try 'touch --help' for more information.
```

    - It'll give error as expected because **touch** command doesn't support **stdin**.

```
[root@ip-172-31-20-103 test]# xargs touch <<< "f1.txt f2.txt"
[root@ip-172-31-20-103 test]# ls
f1.txt  f2.txt
```

    - Now we used **xargs**, it converted those **stdin** values to **arguments** and it worked.

- **xargs**
  - It converts the **stdin to arguments.**
  - by default **xargs** put the **stdin** values at the end of the command as **arguments.**
  - Consider the following scenario:
    - I have 1 file ***f1.txt*** and one file ***test.txt*** which contains name of the file *f1.txt*.

```
[root@ip-172-31-20-103 test]# ls
f1.txt  test.txt
[root@ip-172-31-20-103 test]# cat test.txt
f1.txt
```

```
[root@ip-172-31-20-103 test]# cat test.txt | xargs cp /tmp
cp: -r not specified; omitting directory '/tmp'
```

      - This failed because the command was executed like the below:
      - **cp /tmp f1.txt**
    - Here I need to place the filename i.e. **f1.txt** between **cp** and **/tmp**

```
# cat test.txt | xargs -I{} cp {} /tmp
```

      - It'll pass now. In place of that {} you can write anything, its nothing but one placeholder.
      - Generally {} or **%** are used as placeholder.
  - If a command expects **arguments** as its input and you have to use **input redirection** or **pipe** (these both sends the content as **stdin**, not **arguments**), then you can use **xargs.**
  - Lets say you have **n** lines in a file *f1.txt* and each line will be like

```
x21.txt
x22.txt
```

    (Means each line contains one file name)

- **touch file1.txt file2.txt file3.txt** (you can give any number of arguments here and the files will be created) ----- just for your knowledge
- **xargs touch < f1.txt**
  - it'll be equivalent to the following:
  - **touch x21.txt x22.txt**
- **xargs -I% touch % < f1.txt**
  - It'll be equivalent to the following:
  - **touch x21.txt**
    touch x22.txt

NOTE

- When you use the **-I** flag, **xargs** executes the command **N times** (if the file has N lines), using each line as one argument in **each** execution.
- When you don't use the **-I** flag, **xargs** executes the command **once**, passing all N lines as N arguments in a **single** execution.
-

- **Users and Groups:**

TYPE	EXAMPLE	USER ID (ID)	GROUP ID (GID)	HOME DIR	SHELL
ROOT	root	0	0	/root	/bin/bash
REGULAR	imran, vagrant	1000 to 60000	1000 to 60000	/home/username	/bin/bash
SERVICE	ftp, ssh, apache	1 to 999	1 to 999	/var/ftp etc	/sbin/nologin

1. **/etc/passwd**

```
[root@ktlinux ~]# head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

The above fields are

- **root** =name
- **x**= link to password file i.e. /etc/shadow
- **0** or **1**= UID (user id)
- **0** or **1**=GID (group id)
- **root** or **bin** = comment (brief information about the user)
- **/root** or **/bin** = home directory of the user
- **/bin/bash** or **/sbin/nologin** = shell

- **username:password:UID:GID:GECOS:home_directory:shell**
  - **GECOS:** extra info about the user like full name or something like that.
- **john:x:1001:1001:John Doe:/home/john:/bin/bash**
  - **x:** link to the password file
- **/etc/passwd :passwd file, /etc/group : group file**

```
[root@vbox ~]# id vagrant
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant)
```

  - **id** command is used to see the user info.
- **useradd <username>** => create new user and add it to /etc/passwd file

```
[root@vbox ~]# useradd ansible
user [root@vbox ~]# useradd jenkins
us[root@vbox ~]# useradd aws
[root@vbox ~]# tail -4 /etc/passwd
vboxadd:x:983:1::/var/run/vboxadd:/bin/false
ansible:x:1001:1001::/home/ansible:/bin/bash
jenkins:x:1002:1002::/home/jenkins:/bin/bash
aws:x:1003:1003::/home/aws:/bin/bash
[root@vbox ~]# id ansible
uid=1001(ansible) gid=1001(ansible) groups=1001(ansible)
```

- **useradd --home-dir /opt/tomcat --shell /sbin/nologin tomcat**
  - It means /opt/tomcat will be the home directory.

Shell	Full path	Description
`bash`	`/bin/bash`	Default shell on most Linux systems.
`sh`	`/bin/sh`	Basic POSIX shell (usually a symlink to `bash` or `dash`).
`zsh`	`/bin/zsh`	Popular advanced shell, especially for developers.
`ksh`	`/bin/ksh`	Korn shell, used in some enterprise setups.
`csh`	`/bin/csh`	C shell, legacy syntax.
`tcsh`	`/bin/tcsh`	Enhanced version of `csh`.
`nologin`	`/sbin/nologin`	Prevents user from logging in (often used for service accounts).
`false`	`/bin/false`	Also denies login — the shell immediately exits.

- **groupadd** =>

```
[root@vbox ~]# groupadd devops
[root@vbox ~]# usermod -aG devops ansible
[root@vbox ~]# grep devops /etc/group
devops:x:1004:ansible
```

  - **devops** group was created and ansible was added to this group.
  - G flag is for secondary group, g flag is for primary group.
  - ansible is now inside the devops group (last line of image)
  - **usermod** is used to add the user in a group.
  - **Otherwise open the *<u>/etc/group</u>* file and write the users name (comma separated)**

```
ansible:x:1001:
jenkins:x:1002:
aws:x:1003:
devops:x:1004:ansible,aws,jenkins
-- INSERT --
```

```
[root@vbox ~]# id jenkins
uid=1002(jenkins) gid=1002(jenkins) groups=1002(jenkins),1004(devops)
```

- **-rw-r--r-- 1 john developers 1234 Mar 16 14:00 file.txt**
  - This is the output of a file after doing: **ls -l <file_path>**
  - **rw:** (1st one)owner's permission
  - **r:** (2nd one)group's permission
  - **r:** (3rd one)**other's permission**
- **File permissions:**

```
[root@vbox ~]# ls -l
total 64
-rw-------. 1 root root 2027 Dec 18  2023 anaconda-ks.cfg
-rw-r--r--. 1 root root  143 Feb  2 12:09 grep-test.txt
-rw-r--r--. 1 root root   12 Feb  2 08:20 inputred.txt
-rw-r--r--. 1 root root   81 Feb  2 07:04 longtextfile.txt
```

  - **drwxr-xrw-** total 10 characters
    - First char: file type (here d: directory)

- Then 3 chars 3 times (rwx) format
  - **Rwx :** root user's permissions
  - **r-x** : read, execute (not write)
  - **rw-** : read, write (not execute)

- **Change Ownership of a File:**
  - **chown** command can be used for this
    - **chown -R <user name>:<group name> <directory path>** (If u give -R, it'll change the ownership of all the sub-directories as well)
    - Or: **chown -R <user name>.<group name> <directory path>**
  - **chmod** is used to change the access like read, write,
    - **chmod o-x <directory path>** (it'll revoke permission for "others" to execute) ("o" for others, "-" means revoke, "+" means give permission, "g" for group, "u" for user)
    - Numeric values for
      - R = 4
      - Write = 2
      - Execute = 1
    - If u want to give full access to user, group and no permission to others then
      - chmod -R 770 <directory> (7 = 4 + 2 + 1)
- **Sudo:**
  - You can create any new file inside the **/etc/sudoers.d** directory to give sudo access to any user.
  - It is not recommended to use **/etc/sudoers** file.
- **Packages:**
  - To see the repositories of yum: **/etc/yum.repos.d/**
  - To download: **wget**, **curl** (curl has also some different usability)
  - **curl <url> -o <path name of the would be file>** (**-o** means output)
  - After downloading, it can be installed using the following command:
    - **rpm -ivh <file path>** (i: install, v: verbose i.e. output, h: human readable format)
  - **yum search <package name>** (to search any package, it'll install all the dependent package)
  - Now a modern package manager is there in **rpm** based Linux: **dnf**
  - In place of yum, we should use **dnf**
  - **dnf repolist**: it'll give all the repos that have in there in the system.
- **Services:**
  - **systemctl**: to operate the service (for ex: **httpd** is a service that we installed before)…
    - systemctl start <service>
    - systemctl stop <service> …etc etc
- **Processes:**
  - **ps -ef**: show all the processes
  - **kill <process id>** (to kill any process, if any children processes are there then it'll stop them and then it'll stop itself)
  - **kill -9 <process id>** (to stop the process forcefully)
- **xargs** converts piped input into valid command-line arguments.
  - **echo "file1.txt file2.txt" | rm**
    - It'll not work as rm doesn't expect piped input, it needs arguments in proper format like **rm file1.txt file2.txt**
    - **echo "file1.txt file2.txt" | *xargs* rm**
      - It'll work properly

```
[root@vbox ~]# ps -ef | grep "httpd"
root       83195       1  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83196   83195  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83197   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83198   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83277   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
root       83411    3810  0 17:33 pts/0    00:00:00 grep --color=auto httpd
[root@vbox ~]# ps -ef | grep "httpd" | grep -v "grep"
root       83195       1  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83196   83195  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83197   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83198   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83277   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
[root@vbox ~]# ps -ef | grep "httpd" | grep -v "grep" | awk "{print $2}"
root       83195       1  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83196   83195  0 17:12 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache     83197   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83198   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
apache     83277   83195  0 17:12 ?        00:00:02 /usr/sbin/httpd -DFOREGROUND
[root@vbox ~]# ps -ef | grep "httpd" | grep -v "grep" | awk '{print $2}'
83195
83196
83197
83198
83277
[root@vbox ~]# ps -ef | grep "httpd" | grep -v "grep" | awk '{print $2}' | xargs kill -9
[root@vbox ~]# ps -ef | grep "httpd"
root       83434    3810  0 17:36 pts/0    00:00:00 grep --color=auto httpd
[root@vbox ~]# ps -ef | grep "httpd" | grep -v 'grep'
[root@vbox ~]#
```

- name="hello"
- awk "{ print **$name** }" file.txt
- It is same as awk "{print hello}" file.txt
- Here, awk '{ print $name }' doesn't interprete the value of name variable.
  - '……' => Prevents shell from interpreting $, \, etc. Best for awk code.
- "……." => Shell does interpret variables inside. Avoid for awk code unless needed.

- **Archiving**

```
[root@vbox log]# ls
anaconda  btmp    cron            dnf.log      hawkey.log  kdump.log  maillog   private  samba   s
audit     chrony  dnf.librepo.log  dnf.rpm.log  httpd       lastlog    messages  README   secure  s
[root@vbox log]#
[root@vbox log]#
[root@vbox log]# tar -czvf httpd_alok.tar.gz httpd
httpd/
httpd/error_log
httpd/access_log
[root@vbox log]# ls -ltr
total 2232
-rw-------. 1 root   root         0 Dec 18  2023 tallylog
drwx------. 2 root   root         6 Dec 18  2023 private
-rw-------. 1 root   root         0 Dec 18  2023 maillog
-rw-------. 1 root   root         0 Dec 18  2023 spooler
drwxr-xr-x. 2 root   root      4096 Dec 18  2023 anaconda
lrwxrwxrwx. 1 root   root        39 Dec 18  2023 README -> ../../usr/share/doc/systemd/README.logs
-rw-------. 1 root   root       874 Dec 18  2023 vboxadd-install.log
drwxr-x---. 2 chrony chrony       6 Nov 11  2024 chrony
-rw-r--r--. 1 root   root       107 Jan 26 17:46 vboxadd-setup.log.4
drwx------. 2 root   root        41 Jan 29 17:49 httpd
-rw-r--r--. 1 root   root       107 Feb  1 10:40 vboxadd-setup.log.3
-rw-r--r--. 1 root   root       107 Feb  2 06:58 vboxadd-setup.log.2
drwxr-x---. 2 sssd   sssd         6 Feb 12 15:29 sssd
drwx------. 3 root   root        17 Feb 17 13:06 samba
-rw-r--r--. 1 root   root       107 Mar 16 06:13 vboxadd-setup.log.1
-rw-------. 1 root   root      7160 Mar 29 11:25 kdump.log
-rw-r--r--. 1 root   root       107 Mar 29 11:25 vboxadd-setup.log
-rw-rw----. 1 root   utmp      3072 Mar 29 17:36 btmp
drwx------. 2 root   root        23 Apr 11 14:37 audit
-rw-rw-r--. 1 root   utmp     32256 May 16 17:45 wtmp
-rw-rw-r--. 1 root   utmp    294044 May 16 17:45 lastlog
-rw-------. 1 root   root     18464 May 16 17:51 cron
-rw-------. 1 root   root   1121260 May 16 17:56 messages
-rw-r--r--. 1 root   root    174246 May 16 17:56 dnf.librepo.log
-rw-r--r--. 1 root   root    127043 May 16 17:56 dnf.rpm.log
-rw-r--r--. 1 root   root      3600 May 16 17:56 hawkey.log
-rw-r--r--. 1 root   root    621302 May 16 17:56 dnf.log
-rw-------. 1 root   root     65475 May 16 17:56 secure
-rw-r--r--. 1 root   root       861 May 16 17:58 httpd_alok.tar.gz
[root@vbox log]# file httpd_alok.tar.gz
httpd_alok.tar.gz: gzip compressed data, from Unix, original size modulo 2^32 10240
[root@vbox log]#
```

- Zip

```
[root@vbox tmp]# tar -xzvf httpd_alok.tar.gz
httpd/
httpd/error_log
httpd/access_log
[root@vbox tmp]# ls
hsperfdata_root    systemd-private-2ce2a7bce4164ea1a158cdca44519537-
httpd              systemd-private-2ce2a7bce4164ea1a158cdca44519537-
httpd_alok.tar.gz  systemd-private-2ce2a7bce4164ea1a158cdca44519537-
```

- Unzip
- Another command that is used is: **zip**
  - It might not be there so install it using: **yum install zip**

```
 596  zip -r httpd_alokk.zip httpd
 597  ls
 598  mv httpd_alokk.zip /opt/
 599  ls
 600  cd /opt/
 601  ls
 602  unzip httpd_alokk.zip
```

- **Ubuntu Commands**
  -
```
root@ubuntu-jammy:~# useradd devops
root@ubuntu-jammy:~# su - devops
su: warning: cannot change directory to /home/devops: No such file or directory
$ exit
root@ubuntu-jammy:~# userdel -r devops
userdel: devops mail spool (/var/mail/devops) not found
userdel: devops home directory (/home/devops) not found
```
    - Unlike **centos**, in **ubuntu**, when we create one user it doesn't create the home directory for that user.
  -
```
root@ubuntu-jammy:~# adduser devops
Adding user `devops' ...
Adding new group `devops' (1002) ...
Adding new user `devops' (1002) with group `devops' ...
Creating home directory `/home/devops' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for devops
Enter the new value, or press ENTER for the default
        Full Name []: devops
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
root@ubuntu-jammy:~#
```
    - Use **adduser** command in place of **useradd**.
  - **visudo**
    - When you give this command, it'll open the **/etc/sudoers** file in **nano** editor.
    - To open it in vim editor:
      -
```
root@ubuntu-jammy:~# export EDITOR=vim
root@ubuntu-jammy:~# visudo
visudo: /etc/sudoers.tmp unchanged
root@ubuntu-jammy:~#
```
  -
```
root@ubuntu-jammy:~# wget http://archive.ubuntu.com/ubuntu/pool/universe/t/tree/tree_1.8.0-1_amd64.deb
--2025-05-17 14:27:30--  http://archive.ubuntu.com/ubuntu/pool/universe/t/tree/tree_1.8.0-1_amd64.deb
Resolving archive.ubuntu.com (archive.ubuntu.com)... 185.125.190.83, 185.125.190.81, 91.189.91.81, ...
Connecting to archive.ubuntu.com (archive.ubuntu.com)|185.125.190.83|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 43044 (42K) [application/vnd.debian.binary-package]
Saving to: 'tree_1.8.0-1_amd64.deb'

tree_1.8.0-1_amd64.deb                                  100%[=========================================

2025-05-17 14:27:31 (80.3 KB/s) - 'tree_1.8.0-1_amd64.deb' saved [43044/43044]

root@ubuntu-jammy:~# ls
snap  tree_1.8.0-1_amd64.deb
root@ubuntu-jammy:~# dpkg -i tree_1.8.0-1_amd64.deb
Selecting previously unselected package tree.
(Reading database ... 64003 files and directories currently installed.)
Preparing to unpack tree_1.8.0-1_amd64.deb ...
Unpacking tree (1.8.0-1) ...
Setting up tree (1.8.0-1) ...
Processing triggers for man-db (2.10.2-1) ...
```
    - Downloaded one package named "tree".

```
root@ubuntu-jammy:~# tree
.
├── snap
│   └── lxd
│       ├── 24322
│       ├── 31333
│       ├── common
│       └── current -> 31333
└── tree_1.8.0-1_amd64.deb
```

- **dpkg -l**
  - To list all debian packages in the machine
- *Unlike **yum** in centos, in Ubuntu **apt** is there.*

```
root@ubuntu-jammy:~# cd /etc/apt
root@ubuntu-jammy:/etc/apt# ls
apt.conf.d  auth.conf.d  keyrings  preferences.d  sources.list  sources.list.d  trusted.gpg.
root@ubuntu-jammy:/etc/apt# cat sources.list
## Note, this file is written by cloud-init on first boot of an instance
## modifications made here will not survive a re-bundle.
## if you wish to make changes you can:
## a.) add 'apt_preserve_sources_list: true' to /etc/cloud/cloud.cfg
##     or do the same in user-data
## b.) add sources in /etc/apt/sources.list.d
## c.) make changes to template file /etc/cloud/templates/sources.list.tmpl

# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://archive.ubuntu.com/ubuntu jammy main restricted
# deb-src http://archive.ubuntu.com/ubuntu jammy main restricted

## Major bug fix updates produced after the final release of the
## distribution.
deb http://archive.ubuntu.com/ubuntu jammy-updates main restricted
# deb-src http://archive.ubuntu.com/ubuntu jammy-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team. Also, please note that software in universe WILL NOT receive any
## review or updates from the Ubuntu security team.
deb http://archive.ubuntu.com/ubuntu jammy universe
# deb-src http://archive.ubuntu.com/ubuntu jammy universe
deb http://archive.ubuntu.com/ubuntu jammy-updates universe
# deb-src http://archive.ubuntu.com/ubuntu jammy-updates universe
```

- **apt install <package name>**
  - It'll install the package.
  - In ubuntu, when a package it installed, it automatically get started and enabled.
- **apt remove <package name>**
  - Removes the package but will *not properly remove* all the configs and all.
- **apt purge <package name>**
  - Remove the packages as well as the configurations.
- **sudo**:
  - Runs a command as another user (default: root) but stay as your original user.
  - Needs **own password** (not root's)
  - Requires you to be in the **sudoers** group.
- **su**
  - Switches to another user account (default: root) and gives you an interactive shell
  - Needs the **target user's password**
  - No logging or restriction like **sudo**.
  - Ex: **su john** (requires john's password)
- **-u <username>**
  - Used with **sudo** to specify a user to run the command as.
  - Ex: **sudo -u xyz <some command>** : It'll run the command as user **xyz**.

- **sudo -i :**
  - It checks whether the user is allowed to execute commands as root in **/etc/sudoers.d** directory. If he is allowed then only he can execute.
  - You might be seeing, when you give command ***sudo -i*** in vagrant default user, it doesn't ask for the password. It is because it is mentioned in the ***/etc/sudoers.d/vagrant***
  - ```
    root@ubuntu-focal:~# cat /etc/sudoers.d/vagrant
    vagrant ALL=(ALL) NOPASSWD:ALL
    ```
  - ```
    alok@ubuntu-focal:/root$ sudo -i
    [sudo] password for alok:
    alok is not in the sudoers file.  This incident will be reported.
    ```
    - If it is not mentioned, you'll see this kind of one error.
- **su**
  - When you give **su**, it switches to the root user (default: root), so to authorize you to be the root user, it needs the ***root user password***.
  - ```
    alok@ubuntu-focal:/root$ sudo -i
    [sudo] password for alok:
    alok is not in the sudoers file.  This incident will be reported.
    alok@ubuntu-focal:/root$ su
    Password:
    root@ubuntu-focal:~#
    ```
    - See, here to switch as the root user, you need the root user's password. If you are giving correct password, you can switch to the root user.
- In short: **su** changes the current user to a specific user (su john: means the user is john.. if no user is mentioned then root user) which requires perticular user's password. Whereas **sudo -i** allows you to execute the commands as root user (if you are allowed means if you are being allowed in the /etc/sudoers.d/ directory.
- If you want to open the shell of any perticular user
  - ***sudo -i -u <username>*** or ***sudo -u <username> -i*** (2nd one is better understandable)
    - It's kind of running **-i** command as the user **<username>**
  - **sudo -i** is shorthand for **sudo -u root -i**.
  - ```
    vagrant@ubuntu-focal:~$ sudo -i -u alok
    alok@ubuntu-focal:~$ whoami
    alok
    ```

➢ **Writing into a file directly from the command line:**

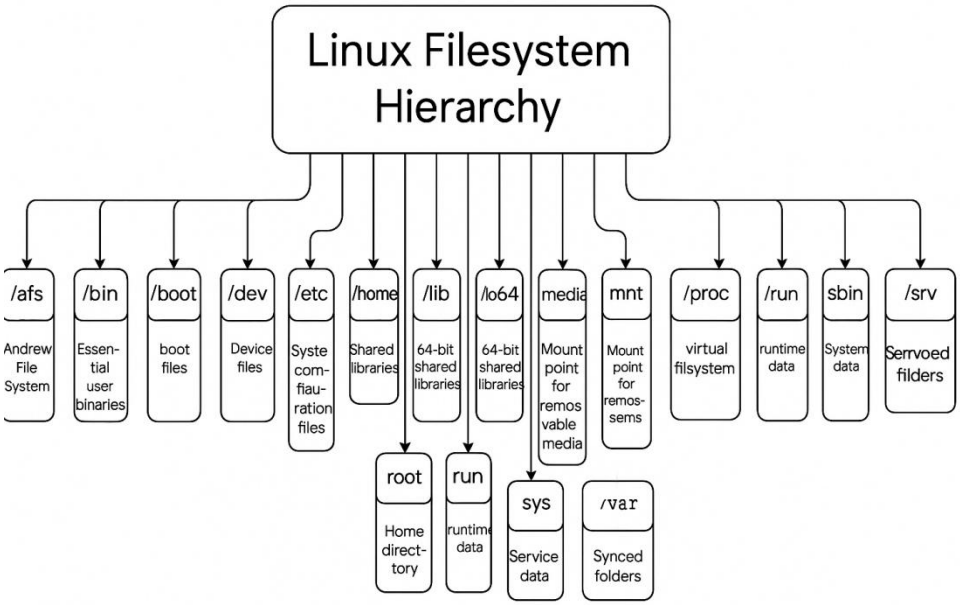
- **cat > hello.txt** (if it is not there, then it'll create one)
  - Write whatever you want… then press **Ctrl+D**
  - Now, the texts you entered will be overwritten in side the hello.txt
  - If you want to append then
    - **cat >> hello.txt**
- If you want to add the content along with the command itself then:
  - *printf "alok \nranjan \njoshi" > hello.txt* *(I used \n to add multilines)*
  - *Or echo "something something" > hello.txt*

```
[root@vbox tmp]# cat <<EOF > hello.txt
> Line 1
> Line 2
> Line 3
> Anything here
> EOF
[root@vbox tmp]# cat hello.txt
Line 1
Line 2
Line 3
Anything here
```

- Anything that comes under **<<EOF** and **EOF** will be saved in side that file. (Note: here <<EOF is not any input redirection)

```
[root@vbox tmp]# cat > hello.txt <<EOF
> Line 1
> Line 2
> Line alok
> Line ranjan
> Line joshi
> EOF
[root@vbox tmp]# cat hello.txt
Line 1
Line 2
Line alok
Line ranjan
Line joshi
```

- This one is looking better.



- When you are downloading any repo (extra repo using ***dnf*** or something) it might be disabled by default. I mean when you execute ***dnf install <some package>*** it'll check in the repo that are enable. So, if your package is not in the enabled repo list, then it won't get downloaded. So, to download your package, your repo must be enabled.
  - You can either permanently enable the repo by editing the ***.repo*** file: ***enabled=1***
  - Else, you can temporarily enable the repo:
    - ***dnf --enablerepo=<repo-name> install <package>***

- ***NOTE***
- ***sudo echo "something" > /root/testfile.txt***
  - It'll give error as *echo* has been executed as super-user (sudo) but output redirection is executed by current user only.
  - So use ***sh*** to run the whole command string inside a new shell having root user privileges.
  - ***sudo sh -c "something" > /root/testfile.txt***

- **Some Examples:**
  - **head and tail:**

```
[root@vbox ~]# head longtextfile.txt
1
2
3
4
5
6
7
8
9
10
[root@vbox ~]# head -5 longtextfile.txt
1
2
3
4
5
[root@vbox ~]# tail longtextfile.txt
21
22
23
24
25
26
27
28
29
30
[root@vbox ~]# tail -4 longtextfile.txt
27
28
29
30
```

```
[root@vbox ~]# head -n 2 separatorfile.txt
value_1_1:value_1_2:value_1_3:value_1_4:value_1_5:value_1_6:value_1_7:value_1_8:value_1_9:value_1_10
value_2_1:value_2_2:value_2_3:value_2_4:value_2_5:value_2_6:value_2_7:value_2_8:value_2_9:value_2_10
```

  - **cut**
    - Splitting the file with delimeter.

```
[root@vbox ~]# cut -d : -f 4 separatorfile.txt
value_1_4
value_2_4
value_3_4
value_4_4
value_5_4
value_6_4
value_7_4
value_8_4
value_9_4
value_10_4
value_11_4
value_12_4
```

    - **Or**

```
[root@vbox ~]# cut -d: -f6 separatorfile.txt
value_1_6
value_2_6
value_3_6
value_4_6
value_5_6
value_6_6
value_7_6
value_8_6
value_9_6
value_10_6
value_11_6
```

    - Getting the values at any positions:

```
[root@vbox ~]# head -n 2 separatorfile.txt
value_1_1:value_1_2:value_1_3:value_1_4:value_1_5:value_1_6:value_1_7:value_1_8:value_1_9:value_1_10
value_2_1:value_2_2:value_2_3:value_2_4:value_2_5:value_2_6:value_2_7:value_2_8:value_2_9:value_2_10
[root@vbox ~]# cut -c 1,2,4,10 separatorfile.txt
vau:
vau:
vau:
```

- **awk**

```
[root@vbox ~]# awk -F: '{print $1}' separatorfile.txt
value_1_1
value_2_1
value_3_1
value_4_1
```

- You can give **-F':'** as well.

```
[root@vbox ~]# awk -F_ '{print $2}' separatorfile.txt
1
2
3
4
5
```

```
[root@vbox ~]# awk -F_ '{sum += $2} END {print sum}' separatorfile.txt
1275
```

- ***awk*** command is very useful to query a file, here I extracted all 2nd indexed values after splitting with _ (which are 1, 2, ..50) and printing out their sum.

- **Sed**
  - **To see the file replacing the keyword (not replacing in the original file)**

```
[root@vbox ~]# sed s/covid19/coronavirus/g sample-file.txt
bcdef flasfkjf; fflaf sfaj flka afjaf alsfa; fa coronavirus
jlfsa alkfjd fccf coronavirus lksdfj falfjla fdf l klfjasfd fc coronavirus ljfdsa ;ffljf fsac coronavirus
laksjff fjlkaf; jlkf coronavirus flkjsaf lkjfja sfa;fkjlaf aaf;jas coronavirus
fsfjkl coronavirus fdajf fljf coronavirus skfdjslf f;fj ffafj coronavirus fslkjfsf coronavirus
```

  - Use **-i** flag to replace in the original file.

```
[root@vbox ~]# sed -i s/covid19/coronavirus/g samplefile.txt
[root@vbox ~]# cat samplefile.txt
bcdef flasfkjf; fflaf sfaj flka afjaf alsfa; fa coronavirus
jlfsa alkfjd fccf coronavirus lksdfj falfjla fdf l klfjasfd fc coronavirus ljfdsa ;ffljf fsac coronavirus
laksjff fjlkaf; jlkf coronavirus flkjsaf lkjfja sfa;fkjlaf aaf;jas coronavirus
fsfjkl coronavirus fdajf fljf coronavirus skfdjslf f;fj ffafj coronavirus fslkjfsf coronavirus
```