What is UNIX ?

- It is operating system created in 1960s.

The soft reqr to communicate user with hardware of computer.

With lot of extensions and improvements to base version, several flavours introduced by organization / companies

RedHat Linux
Ubuntu
Suse

- Characteristics of UNIX :

→ It is Freeware

→ It is Open source software

→ It is FOSS (Freeware & Open Source soft)

→ It is multi user operating system

→ It is multi tasking OS

→ It is user friendly

→ Support both CUI based and GUI based

UNIX | LINUX

(terminal) CUI

command Line Interrupt

CUI => Character User Interface

GUI => Graphical User Interface

right click
↳ new folder
↳ Folder Name: JavaDoc1
create

CUI is more powerful than GUI

dir1, dir2, dir3, ... dir50 In GUI create directory one by one.

but in terminal (CUI)

```
Noor@noor-VirtualBox : ~$ mkdir dir{1..50}
Noor@noor-VirtualBox : ~$ pwd
/home/Noor
```

Make Directory

```
Noor@noor-VirtualBox : ~$ cd Desktop/
Noor@noor-VirtualBox : ~/Desktop$ mkdir dir{1..50}
Noor@noor-VirtualBox : ~/Desktop$ touch dir{1..50}/
                                  File{1..100}
```

create 100 Files in each directory.

→ It is more **secure** than windows OS.
  → Two Levels of security.
  → 3rd person access is restricted.

Flavours OF UNIX

As unix is Open source OS, multiple Flavours are available with lot of extensions and improvement

→ Ubuntu
→ RedHat Linux tin
→ CentOS
→ Fedora
→ Slackware
→ Open Solaries
→ Kali

→ Suse Linux Enterprise server (SLES)
→ Open Suse
→ Multiple ~~Commands~~... flavours

Distrowatch.com

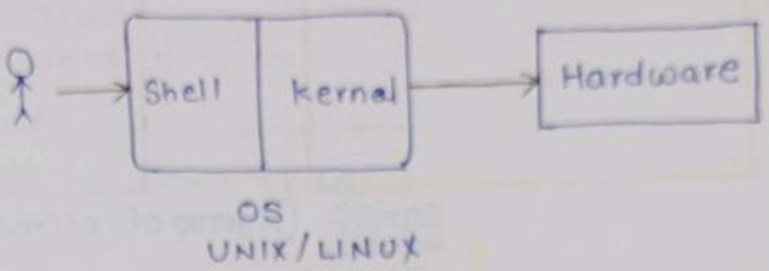
Note → Base version of all flavours is same
→ 80 to 90 % functionality is same

Compo
OF UN

Session - 2

Components of UNIX

→ 2 imp components :
→ Shell
→ Kernal



Shell

→ It is outer layer of UNIX OS.
→ Shell reads command provided by user.
→ Shell will check is it valid command or not.
→ Shell will check is it properly used or not.

```
noor@noor-VirtualBox:~/Desktop$ touch123 abc.txt
command not found error
noor@noor-VirtualBox:~/Desktop$ touch -def abc.txt
invalid date format 'ef'
```

→ If everything is proper, then shell interprets/converts that command into kernel understandable form and handover that to the kernel.

→ Shell acts as interface between user and Kernel.

Kernel

→ It is the core component of UNIX OS.
→ It is responsible to execute our command with the help of hardware components.
→ Memory allocation & processor allocation will take care by kerne

→ It acts as interface between shell and hardware comp.

To do a big task

noor@noor-VirtualBox : ~/Desktop
$ get

Multiple commands
_____
_____
_____

---→

```
cal
date
echo "welcome to unix"
```

test.sh

Script (a group of commands)

```
noor@noor-VirtualBox : ~/Desktop $ chmod u+x test.sh
                                  $ ./test.sh
```

o/p

March 2020

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	8	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				

Mon Mar 23 12:33:13 IST 2020

Welcome to unix

o/p

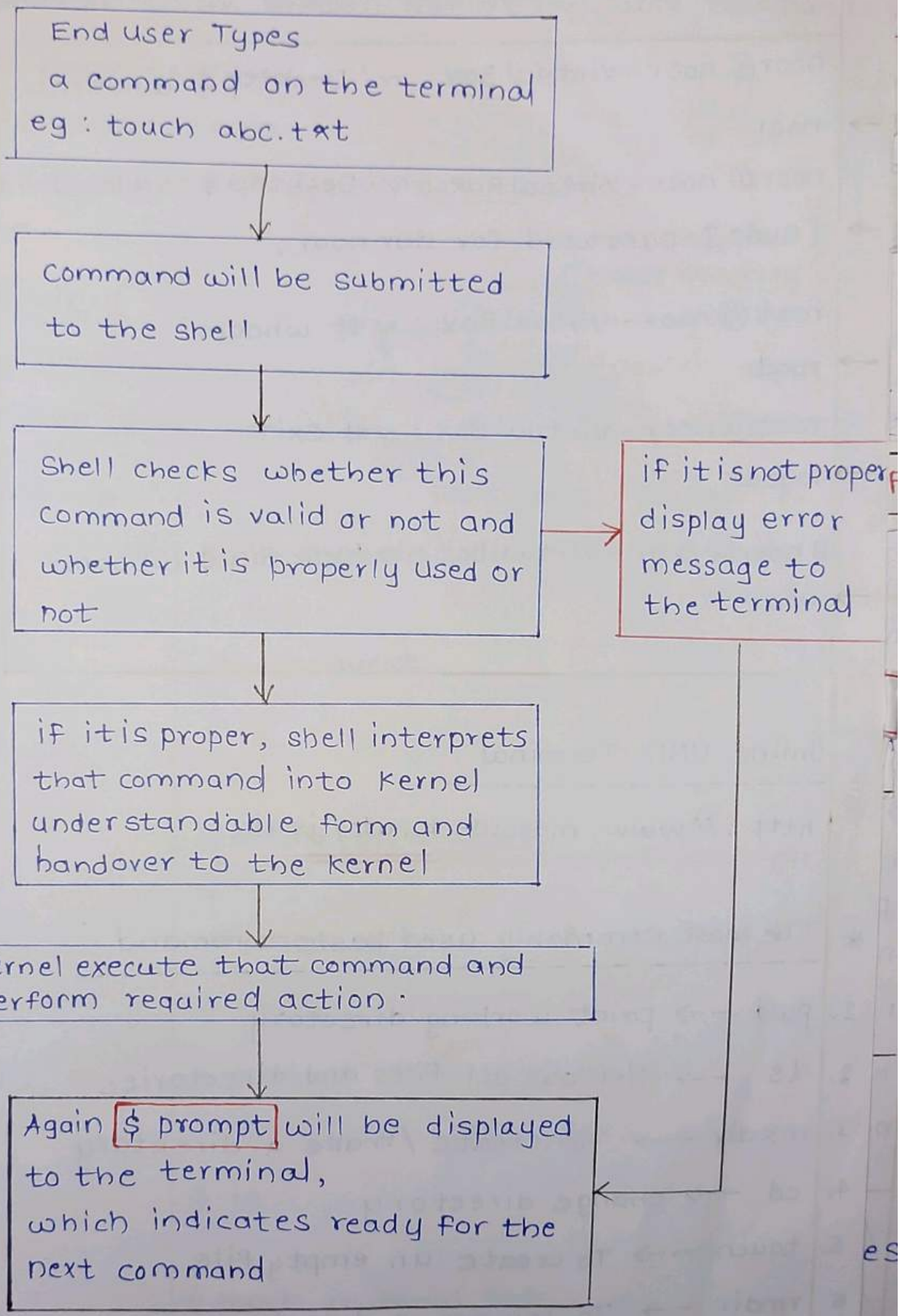
In windows : batch file Java

Interpreted Language - line by line will be executed

Interpreter → kernel

like
↳ Jvm in Java

Command Execution Flow



- normal user --> $ prompt
- super user / root user / admin user --> # prompt

```
noor@noor-VirtualBox:~/Desktop$ whoami
noor
noor@noor-VirtualBox:~/Desktop$ sudo -i
[sudo] password for ~~dar~~ noor:

root@noor-VirtualBox:~# whoami
root

root@noor-VirtualBox:~# exit
logout

noor@noor-VirtualBox:~/Desktop$ whoami
noor
```

→ To switch to super user

Online UNIX Terminal

http://www.masswerk.at/jsuix

* The most commanly used basic command

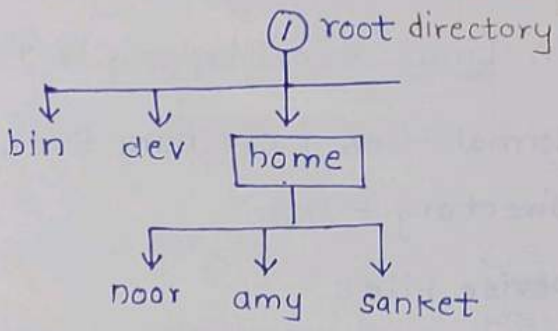
1. pwd --> print working directory
2. ls --> list out all files and directories
3. mkdir --> To create / make a directory
4. cd --> change directory
5. touch --> To create an empty file
6. rmdir --> To remove a directory
7. rm --> To remove a file
8. cal --> Display current month calender

3. date --> Display current date and time
4. help --> To clear ~~terminal~~ display list of available commands
1. clear --> To clear terminal
2. exit --> To logout session
3. hello --> To display brief system information.

⊙ user home directory

for noor : /home/noor

for amy : /home/amy



Online terminal →

```
$ pwd
/home/guest
$ cd..
$ pwd
/home
$ cd..
$ pwd
/
$ clear
```

```
$ cd /home/guest
$ pwd
/home/guest
$ ls

$ mkdir unixclasses
$ ls
unixclasses
$ cd unixclasses
$ pwd
/home/guest/unixclasses
$ ls

$ touch material.txt
$ ls
material.txt
$ rm material.txt
```

```
$ rmdir
uniclasses
```

UNDO files

There is no retrieve commands in linux

Topic 2 : Linux File System :

Session 3

* Types of files in Linux

→ In Python everything is treated as an Object

→ In Linux everythisng is treated as a file

1. Normal files / ordinary files
2. Directory files
3. Device files

1. Normal files / ordinary files

→ These files contain data.

→ It can be normal file or binary files (images, video files, audio files etc.)

→

abc.txt	img.jpg
test.sh	aud.mp4
test.java	

In Linux file extension is not important
Based on content the linux can identify file type

2. Directory files

→ These files represent directories

→ In windows, we can use folder terminalogy but in Linux, we can use directory terminalogy.

→ directory contains files and sub directories also

noor@noor-VirtualBox:~/Desktop$ ls

dir1 dir3
dir2 dir4 ...

noor@noor-VirtualBox:~/Desktop$ rm -r dir*
All directories removed ...

noor@noor-VirtualBox:~/Desktop$ touch img.jpg
noor@noor-VirtualBox:~/Desktop$ ls -l img.jpg
-rw-r--r-- 1 noor newgrp 0 NOV 06 8:45 img.jpg

↑ 0 bytes.

:~/Desktop$ file img.jpg

img.jpg : empty

file command → identify type of file

Hello, welcome to img file.

img.jpg

→ :~/Desktop$ gedit img.jpg

noor@noor-VirtualBox:~/Desktop$ cat img.jpg
Hello, welcome to img file.

noor@noor-VirtualBox:~/Desktop$ file. img.jpg
img.jpg : ASCII text

noor@noor-VirtualBox:~/Desktop$ mv img.jpg img.xyz

renamed ←----------

3. Device Files :

→ In Linux every device is represented as a file.

→ By using this file we can communication with that device.

→ Inside /dev directory
  → all device related files will be there.

terminal ---→ file

terminal related file : tty

terminal represent at ...

terminal - 1

```
noor@noor-VirtualBox:~/Desktop$ tty
/dev/pts/0.

noor@noor-VirtualBox:~/Desktop$ /dev/pts/0
```

terminal - 2

```
noor@noor-VirtualBox:~/Desktop$ tty
/dev/pts/1
```

terminal - 1

```
~$ echo "I Love Java"
I Love Java

~$ echo "I Love Java" > /dev/pts/1
~$
```

send output to terminal 2

comm° bet° 2 terminals

terminal-2

```
~$ I Love Java
```

Output redirection

terminal - 1

```
~$ clear > /dev/pts/1
~$
```

terminal-2

terminal-2 is cleared.

To stop communication between 2 terminals

```
^C
noor@noor-VirtualBox:~$
```

⊙ To open the terminal

① ⋮⋮⋮ Show Application → terminal → Terminal

② ctrl + alt + t ------> noor@noor-VirtualBox:~
File Edit
noor@noor-VB:~$

⊙ To close the terminal

ctrl + d

In Ubuntu :
blue colour files represents : directories
remaining : ordinary files

We have to use ls command with -l option. → find type of file

```
$ ls -l
total 93464
-rw-r--r--  1 noor newgrp      0 Mar  3 12:40 abc.txt
-rw-r--r--  1 noor newgrp     68 Jar 28 13:35 efg.txt
...
```

ls listing | -l long listing

- → normal file

```
drwxr-xr-x  2 noor newgrp   4096 Jun  7 11:22 unixclass
```

d → directory file

```
lrwxrwxrwx  2 noor newgrp      9 Jan 11 12:38
```

l rwx rwx rwx 1 noor newgrp 9 Jan 23 12:38 file2.txt -> file1.txt

↳ link file

- The first character represents the type of file

d ---> directory file
- ---> ordinary file
l ---> link file

c ---> character special file
b ---> block special file
s ---> socket file

} Most of the time used by super user.

Session 4

File System Navigation Commands

directory :- root directory

```
noor@noor-VirtualBox :~$ ls
bin         etc         lib
boot        home        lib64...
cdrom       initrd.img
dev
```

Desktop
↳ dir1
↳ dir2
Empty ← ls

hidden
↓
ls -a

a → all

```
noor@noor-VirtualBox :~/Desktop/dir1/dir2 $ ls -a
.  ..
```

↓

. ---> represents current directory
.. ---> represents parent directory.

current directory → dir2

parent directory → dir1

cd command

1. $ cd.

   changes to current directory (useless)

2. $ cd..

   change to parent directory

```
noor@noor-VirtualBox:~/Desktop/dir1 $ pwd
/home/dtunoor/desktop/dir1
noor@noor-VirtualBox:~/Desktop/dir1 $ cd ../../..
noor@noor-VirtualBox:/home $ pwd
/home
```

3. $ cd.../../..

```
noor@noor-VB:/ $ pwd
/
noor@noor-VB:/ $ cd ..
noor@noor-VB:/ $ pwd
/
```

To go to user home directory

/home/noor

$ cd ~

$ cd /home/noor

↓

$ cd ~ || $ cd

```
noor@noor-VB:~/Desktop/dir1/dir2 $ pwd
/home/noor/Desktop/dir1/dir2
noor@noor-VB:~/Desktop/dir1/dir2 $ cd ~
noor@noor-VB:~$
```

$ cd

cd command without any argument

To go to user home directory

/home/noor

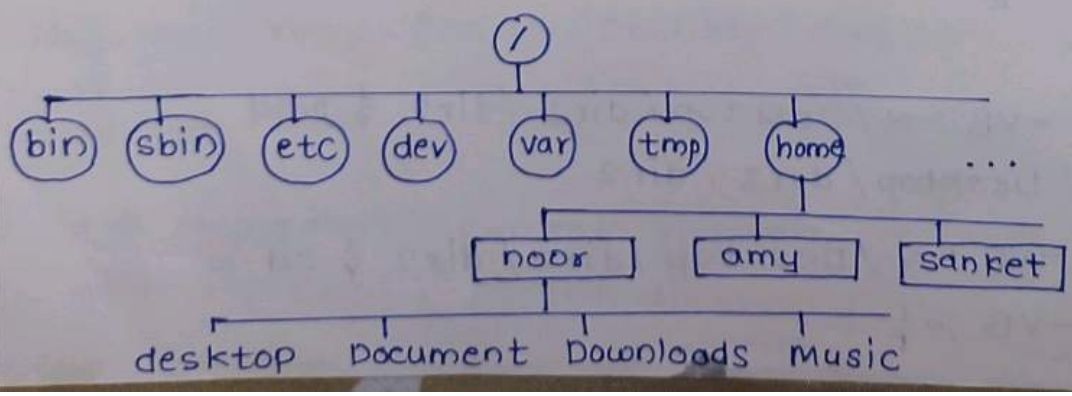
$ cd -

If we want to go previous working directory

```
noor@noor-VB:~/Desktop/dir1/dir2 $ pwd
/home/noor/Desktop/dir1/dir2
noor@noor-VB:~/Desktop/dir1/dir2 $ cd
noor@noor-VB:~ $ pwd
/home/noor
noor@noor-VB:~ $ cd -
/home/noor/Desktop/dir1/dir2
noor@noor-VB:~ $ /Desktop/dir1/dir2 $
```

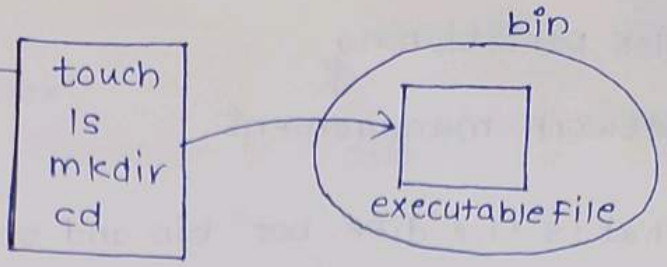
Linux File System Hierarchy

- Linux File System has TREE like structure.
- It starts with root (/)
  → / is the topmost directory
- Subdirectories :
  bin etc home lib dev user cdrom

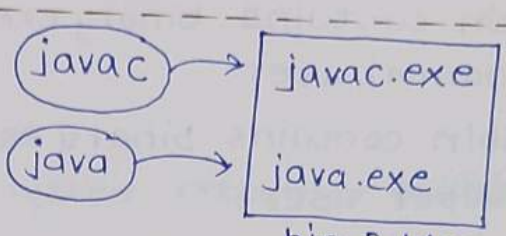


1. bin directory

bin → binary.



noor@noor-VB:~$ which touch
/usr/bin/touch
(in old version ↓
/bin/touch)



noor@noor-VB:~$ which ls mkdir
/bin/mkdir

noor@noor-VirtualBox:~$ cd bin
noor@noor-VirtualBox:~/bin$ ls

bash	mount	mkdir
dash	mountpoint	ls
date	cat	...
dd	cp	

→ It contains all binary executables related to our linux commands.

---

2. sbin directory

→ sbin → systembin.

normal user used commands related binary executable files available in bin directory.

super user used commands related binary executable files available in sbin directory

Disk partitioning

network management

Q. What is the diff. bet° bin and sbin ?

bin contains binary executable related to commands used by normal user.

sbin contains binary executable related to commands used by super user.

---

3. etc directory

→ This directory contains all system configuration files.

→ These configuration files can be used by OS itself.

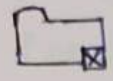
```
noor@noor-VirtualBox : /bin $ whoami
noor
noor@noor-VirtualBox : /bin $ su - noor1
password :                                  wrong password
su: Authentication failure
noor@noor-VirtualBox : /bin $ su - noor1
password :
noor1@noor-VirtualBox : /bin $ cd /etc
noor1@noor-VirtualBox : /etc $ cat passwd
noor : x : 1000 : 1009 : noor ,,, : /home/durga : /bin/bash
noor1 : x : 1001 : 1001 : ,,, : /home/noor1 : /bin/bash
```

/etc/passwd ---> All users information

/etc/group ---> All group information

/etc/hosts ---> All hosts information ipaddress [DNS names]

root



root users home folder
→ requires root permission to access it

run

→ tempfs file system
→ runs in RAM
→ Everything will gone when system reboot or shutdown
→ store runtime information.

snap

→ snap packages -- self contain applications

6. srv

→ ~~server~~ servies directory
→ servies data stored

Folder is Empty

Probably be empty But

if you run server —— web server
FTP server

files will get stored by external users here.

17. sys

→ created everytime the system boots up.

18. usr

→ user applications space -- appl° are installed

19. var

→ variable directory.
→ contains files and directories that expected to grow

crash
log

4. tmp directory

→ temparory directory.
→ contains copy..

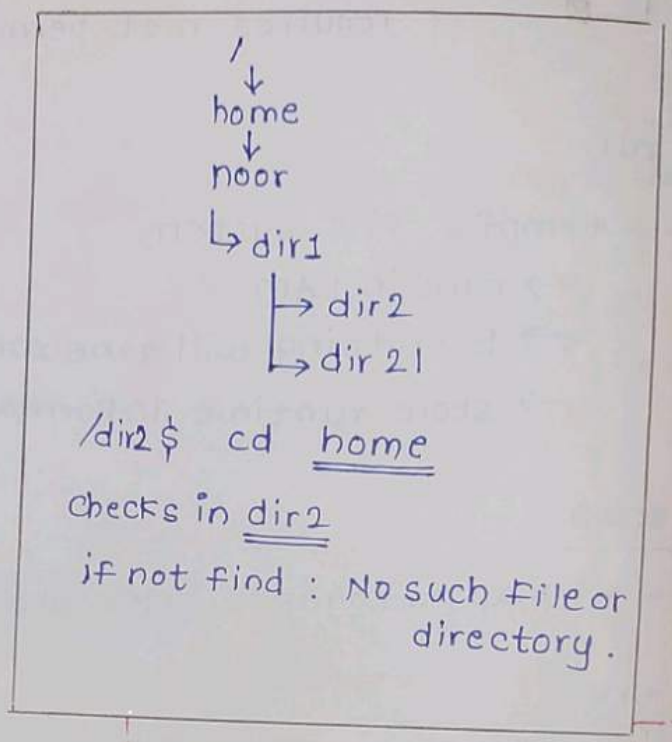
5. boot

→ boot loaders

6. cdrom

7. ~~char~~ dev

→ devices related files
→ sda
sda1
sda2



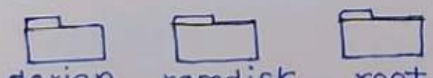
8. lib

→ libraries are stored
- lib32
- lib64

9. mnt (mounting)

→ storage devices.
→ manually manage | in newer os, mounting will be performed automatically.

10. media



→ automatically manage by OS (mounting.

11. opt

→ optional directory.

12. proc (uqnique id : PID)

→ sudo files -- contains information abt sys. resources / processes

$ cat /proc/cpuinfo
$ cat /proc/uptime

$ps -ef

ps → process status.

tty : terminal related file
fd : Floppy drive
hd : Hard disk
ram : RAM
stdin : Keyboard
stdout : terminal / monitor
stderr : std error

home directory

For every user a separate directory will be created to hold his specific data like videos, images, documents etc.
All these user directories stored inside home dire.

/home/noor : It is called noor user home directory.

what is diff. between / and root directories

/ acts as root for Linux file system
→ It is topmost directory of Linux file system.

root --> subdirectory of /
which acts as home directory for super user.

---

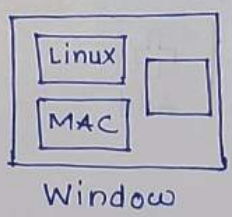
session 6
8 Nov 2020

Linux Installation

- Step1 : Oracle Virtual Box Installation
- Step 2 : Ubuntu Installation

⊙ Need of VirtualBox :

To run virtual computers in our system without effecting original computer



computer → window
VirtualBox →
virtual comp → Linux
virtual comp → mac
virtual comp → sun solaries

→ By using this VB we can run multiple Operating system simultaneously.

* Oracle Virtual Box Installation

→ Download VB software from

VirtualBox.org

Download
Virtul Box 6.1.4

In this virtual box we have to create virtual computer with ubuntu operating system.

* Ubuntu Installation

→ Dowload Ubuntu from

ubuntu.com

downloads

Ubuntu Desktop

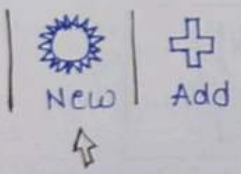
18.04 (18 = year, 04 = month)

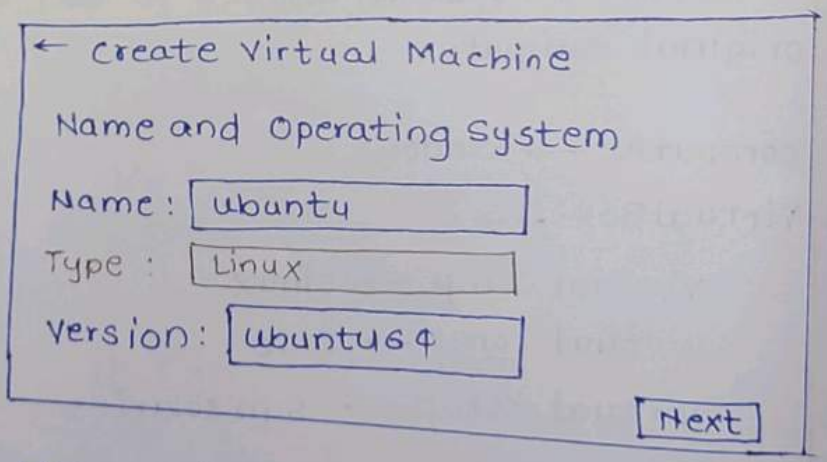
18.04 LTS — Long Term Support

19.10

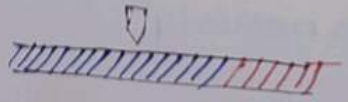
* Create virtual computer

In VB





Memory size →



Hard Disk

- create a hard disk now.
- virtual
- Next
- VDI
- Next
- Create

Settings

System ---> processor

Processor :- 2

Storage ---> Controller → Optical Drive:

select downloaded ubuntu file iso

Start

Welcome

Install Ubuntu

- English (UK)
- Minimal Installation

continue

NOTE

If in system setting : Problem to select Processor.

→ At the time of system start enter into bias mode (system confi. mode)

→ enable virtual technology

After login

ctrl + Alt + t

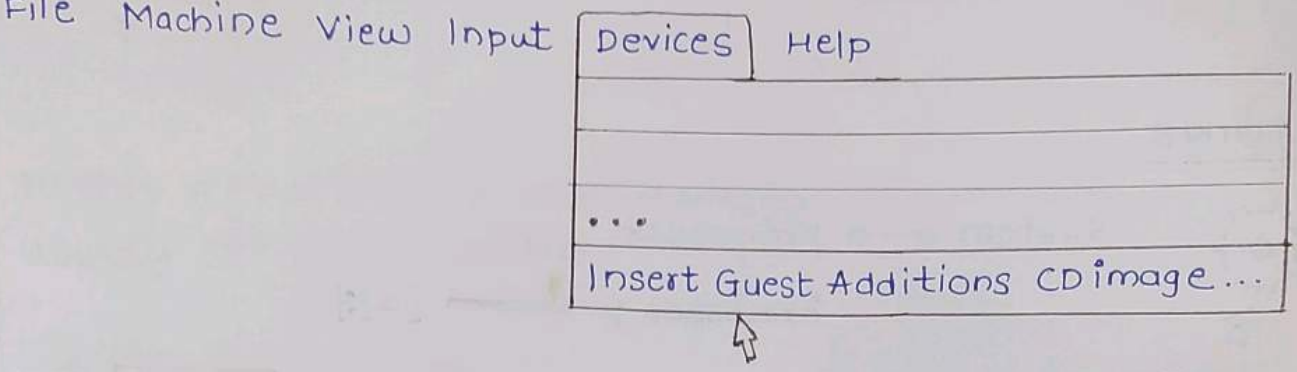
In terminal

```
noor@noor-VirtualBox :~$ echo "Hello"
Hello
noor@noor-VirtualBox :~$
```

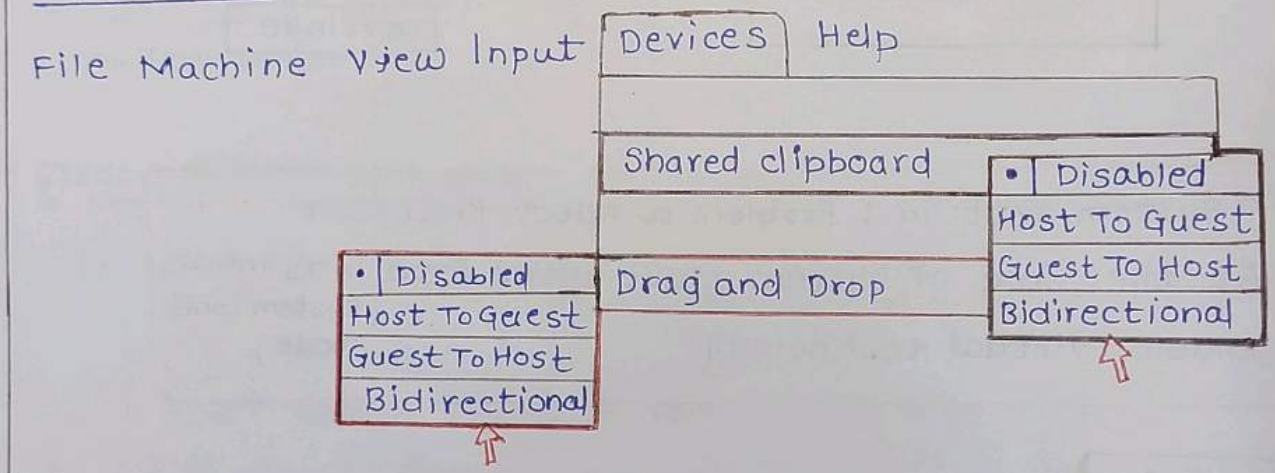
Windows ⟶ host computer
ubuntu ⟶ guest

https://bit.ly/36evx8y
download software

* FULL SCREEN



* Copy-Paste



ls command

- list out all files and directories present in the given directory
- If we are not providing any directory then it will list out in current working directory.

```
$ ls dir1
$ ls ~
```

Tip

Increase the font : ctrl + shift + '+'

For documentation

- manual pages

man pages

```
$ man command
$ man ls
```

```
noor@noor-VB:~$ man ls
```

```
LS(1)          User Commands          LS(1)
NAME
       ls - list directory contents
SYNOPSIS    ....
```

h - help
q - quit

```
noor@noor-VB:~$
```

```
noor@noor-VB:~$ man ls > ls_doc.txt
noor@noor-VB:~$ ls
Desktop      Downloads      Music
Documents    ls_doc.txt     Pictures ...
noor@noor-VB:~$ gedit ls_doc.txt
```

```
LS(1)          User Command          LS(1)
NAME
       ls - List directory contents
  ...
```

ls_doc.txt

Q. How many lines in file

```
noor@noor-VB:~$ wc -l ls_doc.txt
250 ls_doc.txt
```

Various Options for ls command:

1. `ls`

→ It will list out all files and directories present in current working directories in alphabetical order.

---

2. `ls -r`

→ -r means reverse

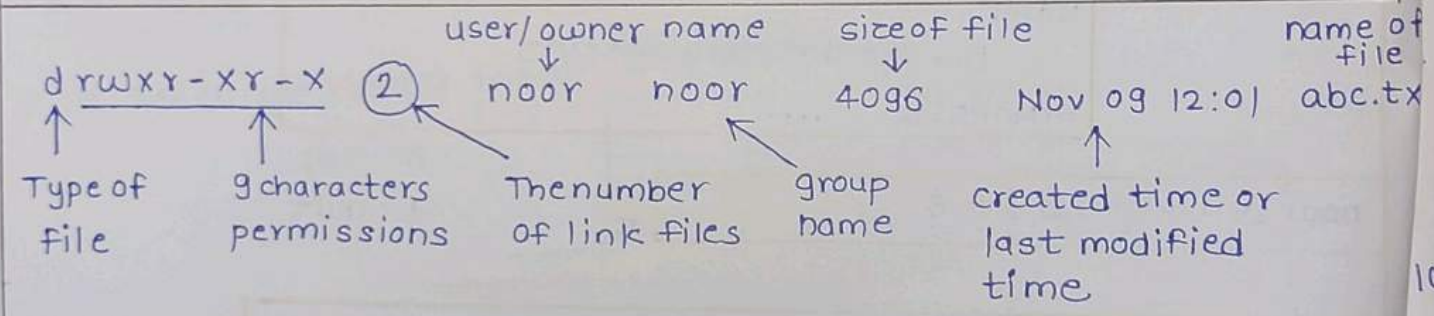
→ It will list out all files and directories in reverse alphabetical order

---

3. `ls -l`

→ To display in long listing form

```
noor@noor-VB:~$ ls -l
total 48
drwxr-xr-x 2 durga noor 4096 Nov 09 12:01 Desktop
drwxr-xr-x 2 noor  noor 4096 Nov 09 12:14 Documents
...
```



detail in files permissions

---

4. `ls -t`

-t means creation or last modified time

→ Most recent will be at the top and old are at bottom

```
noor@noor-VB:~$ ls -lt
```

---

5. `ls -ltr`

→ old files should be at top and most recent are at bottom

`ls -l -t -r` | `ls -ltr` | `ls -lrt` | `ls -rlt` (OK)

`ls -a`

-a means all (hidden files)

```
noor@noor-VB : ~$ ls -a
.                   .dbus
..                  Desktop
.bash_history       Documents
.bash_logout        Downloads
.bashrc             ....
.cache
```

. current working dir.
.. parent directory

```
noor@noor-VB : ~$ ls | wc -l
10
```
← default directories

```
noor@noor-VB : ~$ ls -a | wc -l
27
```
← all hidden and default directories.

---

3. `ls -A`

→ -A means almost (All)

→ except . and .. all remaining things will be displayed

---

10. `ls -F`

→ To display all files by type

```
noor@noor-VB : ~$ ls -F
Desktop/     Downloads/     Music/     public/     Templates/
Doeument/    ls_doc.txt     Pictures/  scripts/    Videos/
```

directory ---→ /

executable file ---→ *

link file ---→ @

```
noor@noor-VB : ~$ ls -F /bin
```

11. `ls -f`

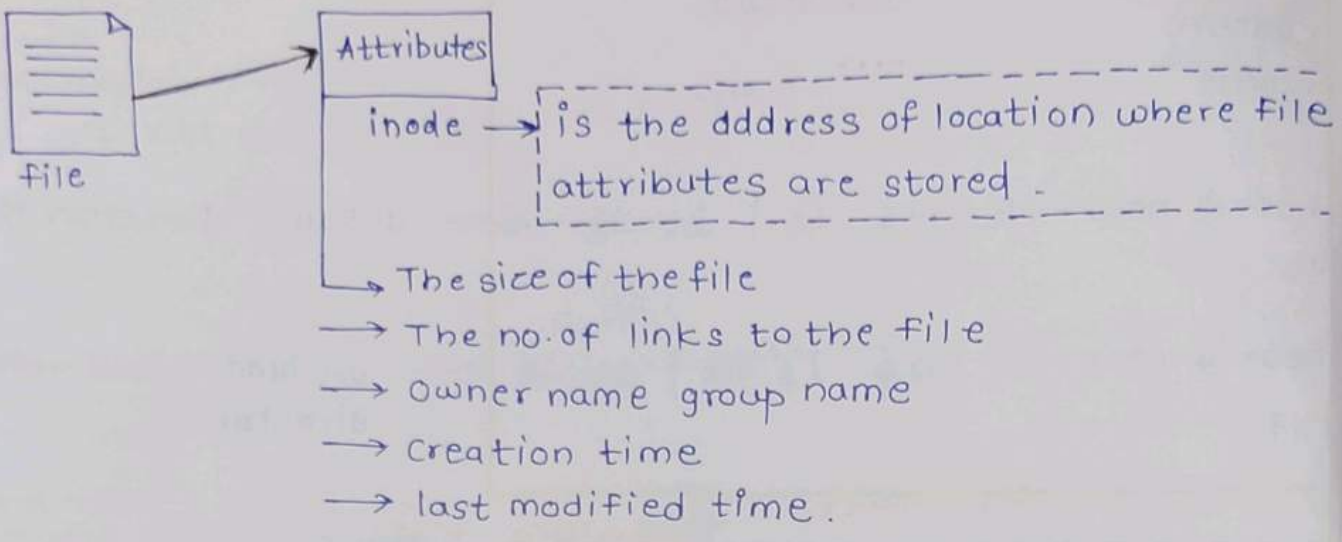
→ list out all files including hidden files but disable colors

---

12. `ls -i`

→ To display all files including incode number

incode → index node



---

13. `ls -R`

→ -R means recursive.

```
noor@ noor@-VB : ~ $ ls
Desktop          Downloads      Music
Documents        ls_doc.txt     ...

noor@ noor-VB : ~ $ ls -R
. :
Desktop          Downloads      Music
Documents        ls_doc.txt     ...

./Desktop :

./Documents :

./Downloads :

./Music :

./Scripts :
a.txt  b.txt  jj  test.sh
```

`ls -l -t -r` | `ls -lt` | ...

`ls -s`

→ -s means size

→ The number of blocks used

`1 block = 1KB` or 4KB

```
noor@noor-VirtualBox:~$ ls -ls
total 48
4 drwxr-xr-x 2 noor noor 4096 Nov 09 12:48 abc.txt
....
```

---

5. `ls -h`

→ -h means human readable form - for memory statistics.

```
noor@noor-VirtualBox:~$ ls -lh
total 48
drwxr-xr-x 2 noor noor 4.0K Nov 09 12:48 Desktop
drwxr-xr-x 2 noor noor 4.0K Nov 09 12:48 Documents
...
```

NOTE

`ls -l | head`

---> top 10 entries

ls -l /etc ---> 212 entries

ls -l | head -20

---> top 20 entries

```
@noor@noor-VB:~$ ls -l /etc | head
top 10 entries ↓

noor@noor-VB:~$ ls -l /etc | head -3
top 3 entries ↓
```

`ls -l | tail`

---> last 10 lines

```
noor@noor-VB:~$ ls -l /etc | tail -3
```

last 3 entries ↓

`ls -l /etc | more` ---> only in forward direction

`ls -l /etc | less` ---> Both in forward and backward direction (nextpage and previous page)

```
noor@noor:VB:~$ ls -l /etc | more
drwxr-xr-x  2 root root   73 Aug  6 2019 shells
-rw-r--r--  1 root root   59 Mar 26 12:10 ss|

--More--
```

↑

Enter space to move to next page

`q` - to come out

NOTE

order is not imp

```
ls -l -t -r
ls -ltr
ls -lrt
ls -trl
```

Q. All files including hidden files with their inode number ?

→ `ls -ai`

Q. long listing of all files including hidden files sorted by modification date (oldest first)

→ `ls -latr`

Q. hidden files :

→ .abc .efg .bashrc

Argument

Command Line Arguments

session-9

The arguments which are passing From command prompt

$./tesh.sh 10 20 30 40
command linearg. ⟶ 10, 20, 30, 40

$ ./test.sh learing linux
command line arg. ⟶ learning , linux

$# ---> Number of arguments (2)
$0 ---> Script Name (./test.sh)
$1 ---> First argument (learning)
$2 ---> second argument (linux)
$* ---> All arguments (learning linux)
$@ ---> All arguments (learning linux)
$? ---> Represents exit code of Previously executed command or script.

test.sh → /home/noor/scripts/test.sh

noor@noor-VirtualBox:~/Desktop$ cd ../scripts/
noor@noor-VirtualBox:~/Desktop/scripts $ gedit tesh.sh

```
#! /bin/bash
echo "The Number of arguments: $#"
echo "Scripts Name: $0"
echo "First argument: $1"
echo "second argument: $2"
echo "All argument with * : $*"
echo "All argument with @ : $@"
```

Script

tesh.sh

noor@noor-VirtualBox:~/scripts $ ./tesh.sh learning linux

The Number of arguments: 2
Script File Name : ./test.sh
First argument : learning
second argument : linux
All argument with * : learning linux
All argument with @ : learing linux

if we execute tesh.sh
/home/noor/scripts/testsh

Difference between $@ and $* :

$@ ---> All command line arguments with space separator

"$1" "$2" "$3" "$4" "$5"

learning linux is very easy

$* ---> All command line arguments as a single string

"$1c$2c$3c$4c$5"

learning linux is very easy

Where, c is the 1st character of the IFS [Internal Field seperator]

The default is space.

How to check default IFS

$ set | grep "IFS"

grep: searching command in linux

set : get all environment variables

env : ↑

```
noor@noor-virtualBox:~/scripts$ set | grep "IFS"
IFS=$' \t\n'
...
```

1st char. of IFS is space by default.

```
#! /bin/bash
IFS="_"
echo "All arguments with @ : $@"
echo "All arguments with * : $*"
```

```
noor@noor-VB :~/scripts$ tesh.sh learning linux is very easy
All arguments with @: learning linux is very easy
All arguments with *: learning-linux-is-very-easy
```

$@ ---> separate entity (recommanded)

$* ---> single string.

What is the main purpose of command line arguments :

→ We can customize behaviour of the script

tesh.sh :

```
#! /bin/bash

APPLE
echo "APPLE" | wc -c
```

wc -l No of lines
wc -c No of characters

```
noor@noor-VirtualBox:~/scripts $ echo "APPLE" | wc -c
6
```

```
#! /bin/bash
echo "APPLE"
echo "MANGO"
```

```
noor@noor-VirtualBox:~/scripts $ tesh.sh
APPLE
MANGO
```

After printing string /n (next line) is printed....
if new line # is not wanted in output then ↓

```
#! /bin/bash
echo -n "APPLE"
echo "MANGO"
```

```
noor@noor-VirtualBox:~/scripts $ tesh.sh
APPLEMANGO
```

APPLE /n → /n is also counted as 6th character

```
noor@noor-VirtualBox:~/scripts $ echo -n "APPLE" | wc -c
5
```

```
#! /bin/bash
len = $(echo -n "APPLE" | wc -c)
echo "The length of given string: $len"
```

```
noor@noor-VB:~/scripts $ tesh.sh
The length of given string : 5
```

```
#! /bin/bash
len=$(echo -n "$1" | wc -c)
echo "The length of given string : $len"
```

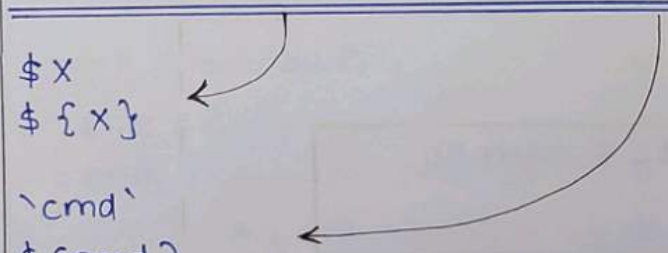
len=$ nospace.

```
noor@noor-VB:~/scripts $ tesh.sh MANGO
The length of given string : 5
noor@noor-VB:~/scripts : $ tesh.sh WELCOME TO LINUX
The length of given string : 7
```

→ ∴ Based on our arguments the script will work

purpose of command Line arg.

Variable substitution and command substitution



Write a script to create log file with timestamp:

test.sh

```
#! /bin/bash
date +%d_%m_%Y_%H_%M_%S

noor@noor-VirtualBox :~/scripts $ date +%d_%m_%Y_%H_%M_%S
27_03_2020_19_56_58

#! /bin/bash
timestamp = $(date +%d_%m_%Y_%H_%M_%S)
echo "This is data to
log file" >> ${timestamp}.log

echo "This is extra data
to log file" >>
${timestamp}.log

date >> ${timestamp}.log

echo >> ${timestamp}.log

echo "Data written to
log file successfully"
```

if file not present then new file created

```
$ echo "APPLE" > abc.txt
$ cat abc.txt
- APPLE
```

overwrite operation

```
$ echo "MANGO" > abc.txt
$ echo cat abc.txt
- MANGO
```

append to existing data

```
$ echo "Bananna" >> abc.txt
$ cat abc.txt
- MANGO
  Bananna
```

```
noor@noor-VirtualBox :~/scripts $ test.sh
Data written to log file successfully
noor@noor-VirtualBox :~/scripts $ ls
27_03_2020_19_58_59.log   demo.sh   test.sh
abc.txt                   env.sh    magic.sh
```

```
noor@noor-VirtualBox:~/scripts$ cat 27_03_2020_19_58_59.log
This is data to log file
This is extra data to log file
Fri Mar 27 19:58:59 IST 2020
                                        <---- blank line
```

```
noor@noor-VirtualBox:~/scripts$ test.sh
Data written to log file successfully
noor@noor-VirtualBox:~/scripts$ ls
27_03_2020_19_58_59.log   abc.txt   env.sh   test.sh
27_03_2020_20_10_42.log   demo.sh   magic.sh
```

for every sec new file will be created
if you want to create new file for every minute then just change

```
#! /bin/bash
timestamp = $(date +%.d_%.m_%Y_%H_%M)
```

```
noor@noor-VB:~/scripts$ tesht.sh   <- 1
Data written to log file successfully
noor@noor-VB:~/scripts$ test.sh    <- 2   (within one sec.)
Data written to log file successfully

noor@noor-VB:~/scripts$ ls
27_03_2020_19_58_59.log   acb.txt   magic.sh
27_03_2020_20_10_42.log   demo.sh   test.sh
27_03_2020_20_12.log  <-  env.sh

noor@noor-VB:~/scripts$ cat 27_03_2020_20_12.log
This is data to log file
This is extra data to log file
Fri Mar 27 20:12:10 IST 2020          - 1

This is data to log file
This is extra data to log file
Fir Mar 27 20:12:11 IST 2020          - 2
```

timestamp = $(date +%d-%m-%Y-%H-%M-%S)
log file will be created for every second

timestamp = $(date +%d-%m-%Y-%H-%M)
log file will be created for every minute

timestamp = $(date +%d-%m-%Y-%H)
log file will be created for every hour.

timestamp = $(date +%d_%m_%Y)
log file will be created for every day.

timestamp = $(date +%m_%Y)
log file will be created for every month.

timestamp = $(date +%Y)
log file will be created for every year.

View content of the file

Using following commands

cat
tac
rev
head
tail
more
less

✓ cat < file1.txt
↳ where < is optional [Input redirection]

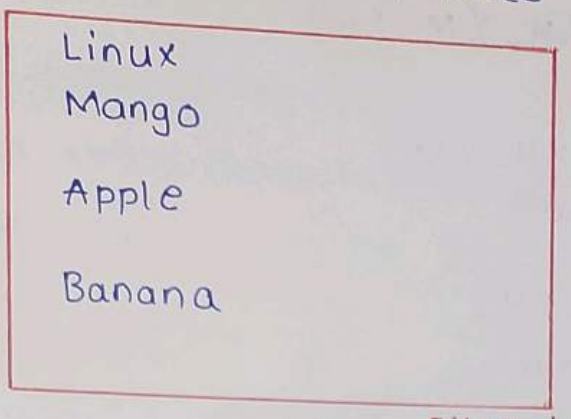
✓ cat file1.txt

$ cat abc.txt
APPLE
MANGO
Banana

While display data if we want line numbers, we have to use -n option

```
noor@noor-VB:~/Desktop$ cat -n file1.txt
     1  Mango
     2  Apple
     3  Orange
```

if there are some blank lines in file :



file1.txt

```
noor@noor-VirtualBox:~/Desktop$ cat file1.txt
Linux
Mango

Apple

Banana

noor@noor-VirtualBox:~/Desktop$ cat -n file1.txt
     1  Linux
     2  Mango
     3
     4  Apple
     5
     6  Banana
```

- To skip blank lines we have to use (-b) option

```
noor@noor-VirtualBox:~/Desktop$ cat -b afile1.txt
     1  Linux
     2  Mango

     3  Apple

     4  Banana
```

↑ Numbering skipped for blank lines

To view multiple files content simultaneously

```
noor@noor-VB:~/Desktop$ cat file1.txt file2.txt file3.txt
Mango
Apple
Orange
Banana
This is firstline
This is second line
Audi
BMW
Mercediz
```

Various utilities of cat command :

1. Create a new file with some content :

```
cat > file1.txt
required data
ctrl + d ---> to save and exit
```

2. To Append some extra data to an existing file :

```
cat >> file.txt
some more extra data
ctrl + d ---> to save and exit
```

To view content of the file :

```
Cat < file.txt   or   cat file.txt
```

To copy content of one file to another file :

```
cp file1.txt file2.txt
```

```
Cat file1.txt > file2.txt ---> Both files content same data
```

To append one file content to another

```
Cat file1.txt >> file2.txt
```

To copy content of Multiple files to a single file

```
cat a.txt b.txt c.txt d.txt > totat.txt . [overwrite]

cat a.txt btxt c.txt d.txt >> total.txt [append.]
```

Cat is the word derived form ~~com~~ concatenation

2. tac command

→ It is the reverse of cat

→ It will display file content in reverse order of file

```
noor@noor-VB:~/Desktop$ cat abc.txt
Sunny
Bunny
Chinny
Vinny
Pinny
noor@noor-VB:~/Desktop$ tac abc.txt
Pinny
Vinny
Chinny
Bunny
Sunny
```

3. rev command

→ rev means reversal

→ Here each line will be reversed [It is Horizontal reversal]

```
noor@noor-VB:~/Desktop$ rev abc.txt
ynnuS
ynnuB
ynnihC
ynniV
ynniP
```

cat command will display total file content at a time.

```
noor@noor-VirtualBox:~/Desktop$ man cat > abc.txt
noor@noor-VirtualBox:~/Desktop$ ls -l abc.txt
-rw-r--r-- 1 noor newgrp 2039 Mar 31 12:57 abc.txt
noor@noor-VirtualBox:~/Desktop$ cat abc.txt
CAT(1)                    User Commands                    CAT(1)

NAME
       cat - concatenate files and print on the standard op
...
```

It is best suitable for small files.

If the file contains thousands of lines it is not recommended to use cat command. For this requirement we should go for

head more
tail less

head command :

→ To view specified number of lines from top of the file.

abc.txt

```
Line-1
Line-2
Line-3
Line-4
Line-5
....
Line-21
Line-22
Line-23
Line-24
Line-25
```

$ head abc.txt

→ It will display only 10 lines from the top of the file

→ 10 is default value.

$ head -n 15 abc.txt ✓

↳ head -15 abc.txt ✓

$ head -n -5 abc.txt

↳ It will display all lines from top except last 5 lines.

$ head -n -5 abc.txt

Line-1
Line-2
...
Line-18
Line-19
Line-20

$ head -c 100 abc.txt

→ It will display the first 100 characters of the file.

→ In linux every character will take one byte.
Hence it will display first 100 bytes of file content.

---

② tail command

→ To view file content from bottom of the file
opposite to head command

$ tail abc.txt

→ Last 10 lines only displayed.

$ tail -n 5 abc.txt

→ Last 5 lines only display.

$ tail -5 abc.txt

$ tail -n -5 abc.txt

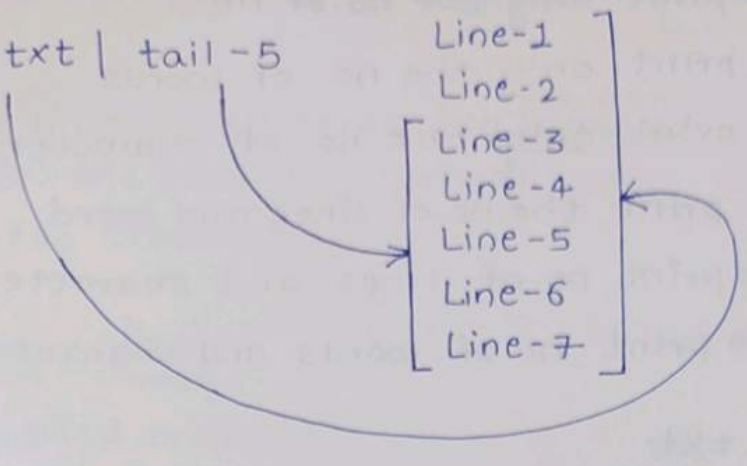
→ display all last 5 lines only

$ tail -c 100 abc.txt

→ It will display 100 bytes of content from the last.
100 characters.

Q. Assume file contains enough data.
Write command to display from 3rd line to 7th line?

→ head -7 abc.txt | tail -5



session - 21

Word Count Command

We can use wc command to count number of words, lines and characters present in the file.

$ wc filename

no_of_lines no_of_words no_of_characters filename

textfile

Linux is a freeware OS
And it is open source

file1.txt

$ wc file1.txt

2 10 45 file1.txt

space is also a character and \n (new line) is also a character

We can use the following options with wc command

-l ---> To print only the no. of lines
-w ---> To print only the no of words
-c ---> To print only the no. of characters
-lw ---> To print the no. of lines and word
-lc ---> To print no. of lines and characters.
-wc ---> To print no of words and character

```
$ wc -l test.txt
2  test.txt
```

-L ---> The length of longest line
To print no. of characters present in the longest line

We can use wc command for multiple file simultaneously

```
$ cp file1.txt  file2.txt
$ wc  file1.txt  file2.txt  file3.txt
```

4	24	135	file1.txt
4	24	135	file2.txt
3	9	47	file3.txt
11	57	317	total.

I/P O/P ERROR redirection

Session 24

Input Output and Error Redirection

Redirection :

Std I/p , std o/p and std error are **Data streams** and can flow from one place to another place. We can redirect these streams.

1. Redirecting Std. O/p

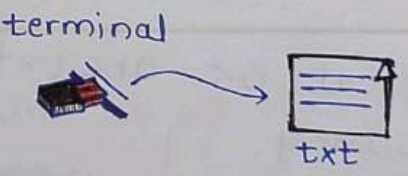
Std o/p associated with (1)
Std o/p connected with terminal, But we can redirect this standard o/p from terminal to some where else based on our requirement.
We can perform o/p redirection by using **> and >>** symbols.

> ---> will perform overwritting of existing data

>> ---> will perform appending to existing data.

- eg: To redirect std. o/p of cat command From terminal to o/p.txt

```
$ cat 1> output.txt
→ This is my first line
  This is my second line
  This is my third line   ctrl+d
$ cat output.txt
  This is my first line
  This is my second line
  This is my third line
$ cat 1>> output.txt
→ This is extra line
$ cat output.txt
→ This is my first line
  This is my second line
  This is my third line
  This is extra line
```



cat 1> op.txt ✓

cat 1 > op.txt ✗

by default > and >> are always meant for output redirection only.

$ cat 1> output.txt == $ cat > output.txt

$ cat 1>> output.txt == $ cat >> output.txt

2. Redirecting std Error

→ We can redirect error msg from terminal to our req. place

→ Using > and >>

→ std Error stream is associated with digit (2)

```
$ cal sjkfstxy 2>> error.txt
```

3. Redirecting Std Input

→ We can redirect from keyboard to our req. place

→ std I/p stream associated with digit 0

→ using < symbol

```
$ cat 0< abc.txt
sunny
Bunny
chinny
vinny
```
} displayed on terminal.

$ cat < abc.txt ✓

$ cat abc.txt ✓

$ cat xyz.txt < abc.txt

$ cat 0< a.txt 1> result.txt 2>> error.txt

↓

$ cat a.txt > result.txt 2>> error.txt

$ cat a.txt &> result.txt o/p and error redirection

Redirecting standard o/p of one terminal to another terminal

terminal-0

```
noor@noor-VB:~/Desktop$ tty
/dev/pts/0

noor@noor-VB:~/Desktop$ cal > /dev/pts/1
```

terminal-1

```
noor@noor-VB:~/Desktop$
                April 2020
Su Mo Tu We Th Fri Sat
 .  .  .  .  .  .   .
```

Session 25

Piping and Importance of tee & xargs

std I/p
std O/p
std Error

command line arg ---> static

→ sometimes we can use O/p of one command as i/p to other command. This concept is called piping.

→ using vertical bar (|)

```
noor@noor-VB:~/Desktop$ ls -l /etc | wc -l
222
```

std o/p of ls command will become std i/p to wc command.

noor@noor-vB : ~/Desktop $ ls -l /etc | more

-rw-.-xr--. 1 noor root 4096 Mar 12:06 abc.txt

. . . .

. . . .

- more -

The o/p of the ls command should be saved to the file op.txt and should be provided as i/p to wc command?

$ ll /etc > op.txt | wc -l

(No o/p.)

→ 0

ll /etc → op.txt

↓

wc -l.

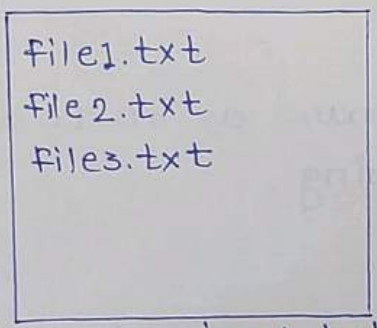
T- junction

T- pipe

tee command

$ ls -l | tee op.txt | wc -l

→ 3

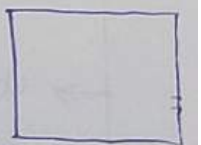


input.txt

Read filename from the input.txt and remove each file

$ cat input.txt | tee rm *

⊗ we should have to save to file....

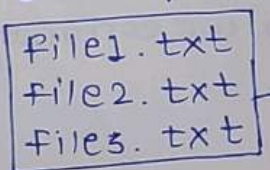


input.txt

xargs command

$ cat input.txt

→

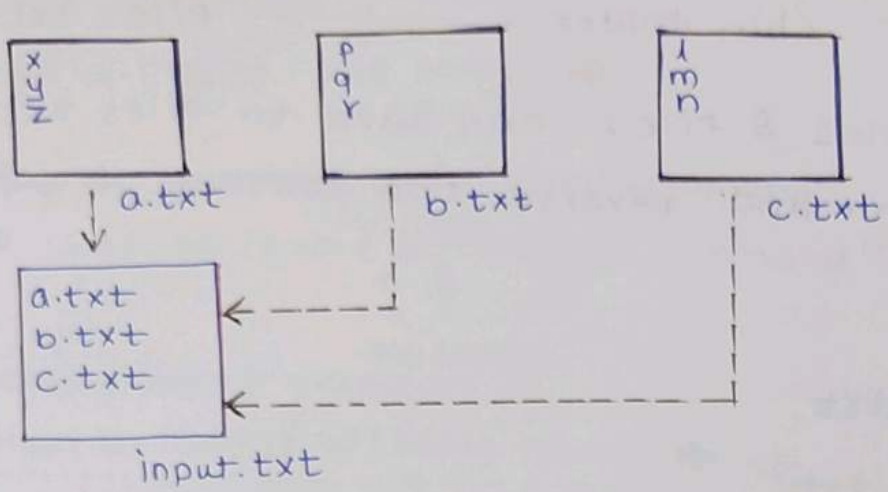


$ rm file1.txt file2.txt file3.txt

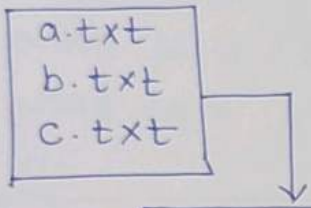
$ cat input.txt | xargs rm

Xargs command :-

to convert output stream into command line arguments



$ cat input.txt



$ cat a.txt b.txt c.txt

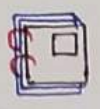
$ cat input.txt | xargs cat

x
y
z
p
q
r
l
m
n

$ date | xargs echo

Sun Apr 5 12:49:44 IST 2020

Assignment



List out all content of /dev folder and save to file1.txt
〃 /bin folder 〃 file2.txt

read content of file1 & file2 and save to file3.txt
By using sort command reverse the content & save to sorted.txt

① ls /dev > file1.txt

② ls /bin > file2.txt

③ cat file1.txt file2.txt > file3.txt

④ sort -r file3.txt > sorted.txt.

cat file1.txt file2.txt | tee file3.txt | sort -r > sorted.txt

Q. What is piping ?

→ A way of connecting commands together
→ A way of passing output of one command as input to another command

session-27

Command Aliasing

↓

Alternative Name

* How to create alias name ➡

$ alias nickname = 'original command'

$ alias nickname = "original command"

'' "" ↕ No space

quotes are not mandatory...

in command prompt

```
Microsoft windows
(C) 2017 Microsoft
c:\users\lenovo> cls
```

* How to listout all alias names ➡

```
$ alias
→ alias ls = 'ls -CF'
  alias ll = 'ls -alF'
  alias la = 'ls -A'
  ...
```

in terminal

```
noor@noor-VB:~/Desktop$ cls
→ command cls not found.
noor@noor-VB:~/Desktop$ alias cls='clear'
noor@noor-VB:~/Desktop$ alias
  alias cls='clear'   getadded
  alias l = 'ls -CF'
  ...
```

Doubt

after giving alias name, is original command will work?

YES

* How to remove alias name ➡

$ unalias command_name

$ unalias cls

$ unalias -a → All alias names (also by default) will gone.

Doubt

How to check whether the alias name is available already or not

use type command

```
$ type cls
→ cls is aliased to 'clear'
```

creating alias to alias is possible.

$ alias cls='clear'

$ alias c='cls'

$ c ☑

* Where we can use alias ➡

→ It any lengthy command repeatedly required.

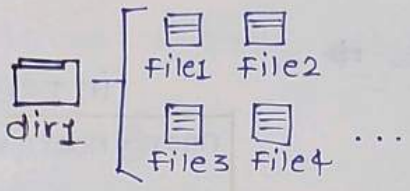
$ alia d20f='mkdir dir1 ; touch dir1/file{1..20}.txt'

$ ls

$ d20f

- $ ls

→ dir1



Q To listout all files present in curr. working dir, save this data to output.txt.
and display no. of lines to the terminal

→ ls -l | tee output.txt | wc -l

$ alias current='ls -l | teeoutput.txt | wc -l

---

noor@noor-VirtualBox:~/scripts$ man current

→ No manual entry for current
man pages are going to define only for the original commands

→ To use home opera. sys. command in linux
↓
Windows
Mac

→ To handle typing mistakes.

→ To handle language barriers

* How to persist aliases permanently

↳ by using 2 ways

1st way → .bashrc file inside user home directory.
↓
hidden file.

Open ∨ *.bashrc

```
...
# some more ls aliases
alias ll = 'ls -alF'
alias la = 'ls -A'

# my own aliases
alias cls = 'clear'
alias dd = 'date'
...
```

To reflect these aliases we have to restart terminal...

2nd way →

Instead of editing .bashrc file, we can create our own file to maintain our defined aliases.

→ The name should be .bash_aliases

→ should be present in user home directory

$ source $ gedit .bash_aliases →

*.bash_aliases

```
# my own aliases
alias cls = 'clear'
...
```

$ alias ← close and open new terminal

→ aliases come

Session-29

How to use Regular Expression

1. To list out all files present in current working directory

```
$ ls *
→ file1.txt  abc.txt
  dir1 :
   file1.txt  file2.txt  file3.txt.
```

2. List out all files with some extension

```
— $ ls *.*
→ a.java  d.java  file1.txt  file2.txt  x.py

  $ ls *.txt
→  a.txt  file1.txt  abc.txt.
```

3. list out all files start with 'a'

```
  $ ls a*
→  abc.txt  axy.py  abs.java.

  $ ls a*t
→  abc.txt  dir1
```

4. where filename contains only 2 characters & First char should be 'a'

```
→ $ ls a?
→ aa  ab  az
```

? - Represent only one character

5. at least 3 characters ...

```
$ ls ???*
aaa  bbb  ccc  test.txt  abc.txt  ax.py.
```

6. To list out all files where file name starts with a or b or c

```
$ ls [abc]*
→ aa  bb  abc.txt  cc  axy.py
```

7. should not start with a or b or c

```
$ ls [!abc]*
→ dd  xyz.txt  file.txt
```

8. filename starts with lower case alphabet symbol.

```
$ ls [a-z]*
abc.txt  bb  axy.py
```

] in ubuntu it will not work

```
$ ls [[:lower:]]*
abc.txt  bb  axy.py
```

] for ubuntu another way.

```
$ ls [[:upper:]]*
Abc.txt  XYZ  AA.py
```

Tiger Ele tiger Den

9. file name starts with digits

```
$ ls [0-9]*
```

(or)

```
$ ls [[:digit:]]*
→ 1abc.txt  1file.txt  2test.py
```

```
$ ls [A-Z][0-9][a-z]
↳ $ ls [[:upper:]][[:digit:]][[:lower:]]
```

10. file name starts with alphanumeric

```
— $ ls [![:alnum:]]*
→ _.abc.txt
```

11. all file with .java or .txt

$ ls {*.java, *.txt}

→ abc.java xy.txt file1.java file2.java.

↓

$ ls *.{java,txt} ✓

NOTE: we can use regular expressions & wild card chara. with other command also.

↓

⊙ To copy all files starts with digit to dir1 directory ?

$ ls

→ dir1 1abc.txt 2file.py

$ cd dir1

→ .

$ cp [0-9]* dir1

$ ls

1abc.txt 2.file.py

⊙ To move all files with alph. sym. & with .txt exten to dir2

$ ls

dir2 abc.txt xyz.txt 1ab.java xy.py

$ mv [[:alpha:]]*.txt dir2

→ $ cd dir2

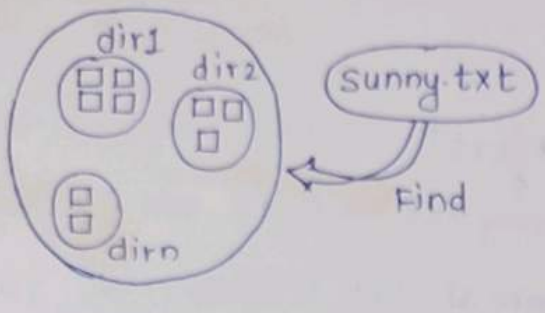
$ ls

→ abc.txt xyz.txt .

$ rm [abc]* [e]

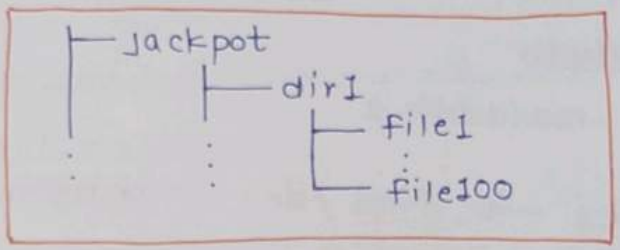
Jackpot Assignment

Section 32




```
noor@noor-VB:~/Desktop$ ls
noor@noor-VB:~/Desktop$ mkdir jackpot
noor@noor-VB:~/Desktop$ mkdir jackpot/dir{1..100}
noor@noor-VB:~/Desktop$ touch jackpot/dir{1..100}/file{1..100}.txt
```

$ tree




```
$ shuf -i 1-100 -n 1   ← some random number.
48

$ shuf -i 1-100 -n 4   ← 4 random nos.
48
51
2
67
```

$ touch jackpot/dir$(shuf -i 1-100 -n 1)/sunny.txt

in random directory → sunny.txt got created.

```
$ find jackpot -type f -name 'sunny.txt'
jackpot/dir9/sunny.txt.
```

$ find jackpot '-type f -name 'sunny.txt' -exec mv {} ~/Desktop \;

$ ls

jackpot sunny.txt

'*.txt'

$ find jackpot -type f -name 'sunny.txt' - exec rm {} \;

Remove all .txt files.

$ find jackpot -type d -exec rmdir {} \;

Remove all directories.

or → rm -r {} \;

→ To find files/direc. inside /dev folder & limit its search to only 2 levels of depth

find /dev -maxdepth 2

→ only directories → find /dev -maxdepth 2 -type d

→ Find from root (/) where filename ends with .txt?

find / -type f -name '*.txt'

find / type f -iname '*.txt'

→ Find all files/direc inside /dev upto max 3 level deep & size > 200k bytes

~~find~~ / -type

find /dev -maxdepth 3 -size +200k.

→ Find all files below home dire. where file size is > 3 MegaByte & remove all files

find ~ -type f -size +3M -exec rm {} \;

find	locate
① in the file system	① database
② More accurate results	② may not accurate
③ slowly	③ faster.
④ More options	④ less no. of options.
⑤ There is a way to use search result directly for some other commands by using -exec option	⑤ There is no direct way.
⑥ We can reduce the search depth	⑥ No such option.

Section 35

grep command and various options

Search multiple content in file

$ grep 'java' demo.txt

$ grep -e 'java' -e 'linux' demo.txt

⇨ Instead of using -e ⟶ we can use egrep command
(egrep → extended grep)

$ egrep -i '(java|linux)' demo.txt

learning linux is very easy
the knowledge of the linux is important.
java is programming language.

(java|linux) ===> either java or linux

$ grep -i '(java|linux)' demo.txt

consider this as single command string

$ grep -i '^the' demo.txt

The most important sub is Linux
the knowledge of linux required anywhere.
↑
starting with the

egrep command is powerful that grep command...

grep	egrep
'^linux' ✓	'^linux' ✓
'(java|linux)' ✗	'(java|linux)' ✓
'[0-9]{9}' ✗	

---

* grep command with F option :

-F means Fixed Strings

The strings should be separated by new line

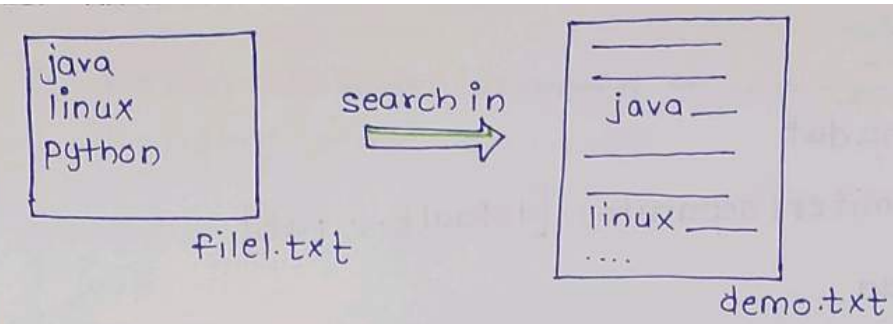
$ grep -F "java ↵ Enter
> linux ↵ Enter
> python" demo.txt

$ fgrep "java
> linux
> python" demo.txt

- Linux is OS used everywhere
We can use Python everytime.
Java developed by james.
'(Linux | java)'

$ fgrep "^the" demo.txt → can't understand...

grep	egrep	fgreps
→ understand patterns but not all patterns	→ und. all patterns	→ cannot understand any command.




```
$ fgrep -f file1.txt demo.txt
```

java

....

---

session 39

Cut command and various options

Cut command

↳ extract data from file.

eno	ename	esal	eaddr	dept
101	noor	1000	pune	admin
102	amy	2000	mumbai	sales
103	sankya	3000	KP	admin

```
$cut -c 9 emp.dat
```

e ← 9th character

n

n

y

y

```
$ cut -c 5-9 emp.dat
```

name|esal

noor|1000

amy|2000|

sankya|300

```
$ cut -c 5- emp.dat
```

→ 5th chara. to last

```
$ cut -c -3 emp.dat
```

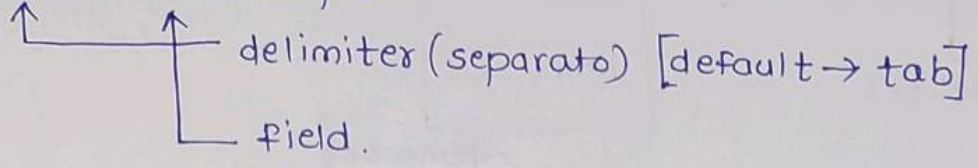
→ upto 3rd chara. from 1st

```
$ cut -c 5-13, 17-19 emp.data
```

5 to 13 & 17 to 19

Specific column data :

```
$ cut -d "|" -f 3 emp.dat
```



range of columns

```
$ cut -d "|" -f 2-4 emp.data
 ename | sal | addr
  noor |1000| pune
  amy | 2000| mumbai
    :
```

```
$ cut -d "|" -f 2- emp.dat
```

2 to the last of column.

```
$ cut -d "|" -f -4 emp.dat
```

1st to 4th column.

```
$ cut -d "|" -f 1,3,5 emp.dat
```

1st 3rd & 5th column display.

skip specific column :

```
$ cut -d "|" --complement -f 3,5 emp.dat
```

display all columns **except 3 and 5**

```
$ cut -d "|" --complement -f 3- emp.dat
```

display only first two columns

* paste command

We can use paste command to join two or more files horizontally by using some delimiter.

sub.txt
```
java
python
cpp
```

fee.txt
```
1000
2000
4000
```

```
$ paste sub.txt feetxt
java 1000
python 2000
cpp 4000
```

```
$ paste -d '-' sub.txt fee.txt
java-1000
python-2000
cpp-4000
```

```
$ paste -d '---->' sub.txt fee.txt
java->1000
python->2000
cpp->4000
```

* tr command

tr → translate.

```
$ tr 'aeiou' 'AEIOU' < demo.txt
```

lowercase ⟶ uppercase

```
$ tr '[a-z]' '[A-Z]' < demo.txt
$ tr '[a-z][A-Z]' '[A-Z][a-z]' < demo.txt
```

WHILE LEARNING JAVA.

```
$ tr 'aeiou' '7' < demo.txt
wh7l7 L77nr7ng j7v7

$ tr '|' '\t' < emp.dat
eno     ename   esal    eadd
100     noor    1000    pune
200     amy     2000    mumbai
```

TAB
\t

```
$ tr -d 'a' < demo.txt
```

delete

delete all occurances of 'a'.

```
tr -d 'aeiou' < demo.txt
```

```
$tr -S 'a' < demo.txt
```

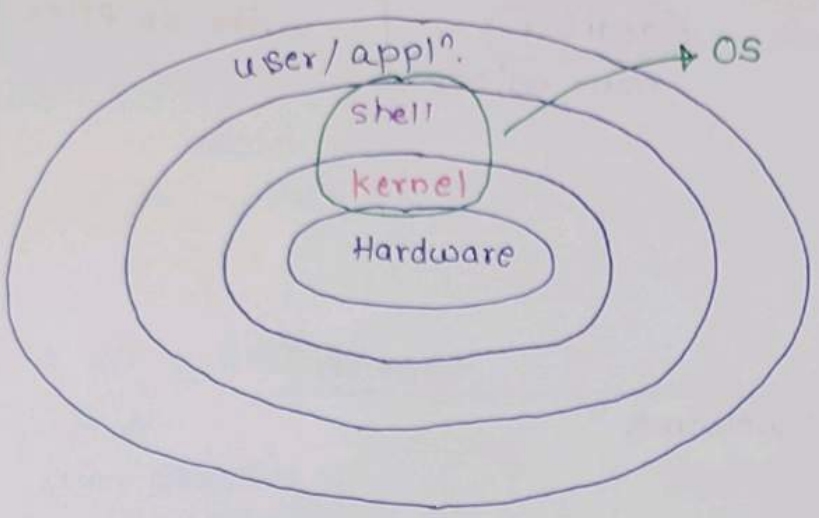
squeeze repeats.

```
aaaaa
aaa
a
```

demo.txt

```
a
a
a
```

What is Shell



1. Shell is responsible to read command provided by user.
2. Shell will check whether the command is valid or not.
3. shell will check whether the command is properly used or not
4. Shell interpretes (converts) that command into kernel understandable form. & handover to kernel.

* Kernel execute that command with the help of hardware.

Shell + kernel = Operating system.

Types of Shells

1. Bourne Shell
2. BASH Shell ⟹ imp.
3. Korn Shell
4. CShell
5. TShell
6. ZShell

- BASH Shell
  → used in realtime.

1. Boume Shell :

— Stephen Bourne.

→ 1st shell deve. for UNIX
→ By using sh command we can access this shell.

Shell Scripting

vi editor
geditor
nano editor
} editor req^r to write .sh files.

a grp of commands
⋮
file → test.sh

$ vtesh.sh
gedit →

test.sh

```
echo " welcome "
date
cal
```

ctrl + s → save
ctrl + q → close.

→ user can't execute this script
User should has execute permission on this script then only he can execute that script.

```
$ ls -l test.sh
-rw-r--r-- 1 noor newgrp 62 Apr 23 19:14 test.sh
```

rw- == user permission
r-- == grp permission
r-- == other permission

rwx → read write execute.

```
$ chmod u+x tesh.sh
```

u → for user, + → add, x → execute

```
$ ls -l test.sh
-rwxr--r-- 1 noor newgrp 62 Apr 23 19:16 test.sh
```

Variable substitution :

$ ./tesh2.sh

- Hello
  Mon Mar 23 19:45 ..

        March 2021
  Su Mo Tu We Th Fr Se
     1 2 3 4 ...

} BASH shell (by default)

$ sh ./tesh.sh

- Hello
  Mon Mar 23 ..

        March 2021

  Su Mo Tu We Th Fr Sa
     1 2 3 ..

} Bourn shell.

2. BASH Shell

BASH ---> Bourne Again SHell.

→ It is advanced version of Bourne shell.
→ This is default shell for most of the linux Flavours.
→ By using bash command we can access.

$ bash ./test.sh

- Hello.. ← bash shell.
  ...

3. Korn Shell

— David korn.

→ Used in IBM AIX OS.
→ By using ksh command, we can access.

4. cshell

— Bill joy.

→ C meant for → California University.
It is also bydefault available with UNIX.
By using csh command we can access.

5. TShell

→ T means Terminal
Advanced version of cshell
→ Used in HP UNIX system
→ By using tcsh command, we can access

6. Z shell

→ Developed by Paul.
→ By using zsh command, we can access.

The most commonly used shell in linux environment is Bash.

$ echo $0
- bash
$ echo $SHELL
- /bin/bash ← complete path.

We can check default file information

$ cat /etc/passwd
- :
noor:x:1000:1009:noor,,,:/home/dnoor:/bin/bash
:

$How to check all available shells in our system

$ cat /etc/shells
- # /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash.

Variable Substitution :

How to switch from shell

```
$ sh
$ echo $0
-sh
$ exit
$ echo $0
- bash
$ rbash
$ echo $0
- rbash
```

What is shell Script

Shell Programming / Shell Scripting

A sequence of commands saved to a file & file is nothing but shell script.

Inside script ↓

programming features also: control statement, loops, functions, arrays eetc.

python scripting → migrated

Shell scripting → still used for maitenance projects.

How to write & run shell scripts

① Write script

```
$ mkdir scripts
$ cd scripts
$ gedit test.sh
```

test.sh :

```
echo "welcome"
date
cal.
```

vi editor
nano editor
geditor → ctrl+s = save
ctrl+q = quit

② User has to execute this script. To execute this script compulsory user should has execute permission on this file

```
$ ls -l test.sh
-rw-r--r-- 1 noor newgrp 52 Mar 24 18:54 test.sh

$ chmod u+x test.sh
$ ls -l test.sh
-rwxr--r-- 1 noor newgrp 52 Mar 24 18:54 test.sh
```

? who is allowed to change perm

③ Run the script

```
$ /bin/bash /home/durga/scripts/test.sh

$ bash /home/noor/scripts/test.sh.

$ bash ./test.sh

$ ./test.sh

$ sh ./test.sh bourne shell.
```

Variable Substitution :

$ gedit demo.sh

demo.sh

```
$
echo "welcome"
echo " Happy to
      see you."
```

$ chmod u+x demo.sh

$ ./demo.sh
↑ current working directory.

Importance of Sha-Bang

---> SHARP
! ---> BANG
#! ---> Sharp Bang / Sha-Bang.

By using sha-bang, we can specify the interpreter (command) which is responsible to execute the script.

$ gedit demo.sh
$ ./demo.sh.
$ gedit. demo.sh ⟶

```
#! /bin/sh
echo " welcome"
echo " Happy to see you"
```

$ ./demo.sh
↑ execute

test.py

```
import random
name = input ("Enter Your Name :");
l = [ "Sunny" , "Malika" , "Veena", "Katrina", "Karina" ]
print ("Hello :", name)
print (" Your cc is : ", random.choice(l))
```

```
$ gedit test.py
$ ./test.py
bash: ./test.py: Permission denied
$ chmod u+x test.py
$ ./test.py
```

error

```
$/usr/bin/python3 ./test.py
Enter your name : noor
Hello : noor
Your cc is Sunny.
```

$ gedit test.py ⟶

```
#! /usr/bin/python3
import random
name = input ....
```

```
$ ./test.py
Enter yourname : noor
Your cc is veena.
```

magic.sh

```
#! /bin/rm
echo "The current Date"
date
```

(internally.)

```
/bin/rm ./magic.sb
```

↑

script/ file removed

remove
rm-command

Variable Substitution :

Accessing the value of a variable by using $ symbol is called variable substitution.

syntax :

```
$variablename   |   ${variablename}
echo $x
```

```
$ gedit test.sh ──►
$ chmod u+x test.sh
$ export PATH=$PATH:
    /home/noor/scripts
$ gedit .bashrc
$ cd /scripts
$ test.sh
```

```
#! /bin/bash
a=10
b=20
course="Linux"
action="sleep"
echo "The value of a&b" $a $b"
echo "My course $course"
echo "My Fav action : $Actioning"
                       ↑ X
                $ {action}ing " ✓
```

```
The value of a & b is 20
My course Linux
My Fav action : sleeping
```

```
NAME="noor"
echo NAME  ===> NAME
echo $NAME ===> VALID noor ✓
echo "$NAME" ===> VALID noor ✓
echo '$NAME' ===> $NAME
echo '${NAME}' ===> $NAME
```

} variable substitution won't be happend.

Command substitution

Execute command and substitute its result.

syntax

old style : `command` (backquotes.) 

new style : $(command)

* eg: Display today date

date ⟶ both date & time

date +%D ⟶ only date

date +%T ⟶ only time.

```
$ echo "Today date is : `date +%d%D` "
Today date is 03/26/20.

$ echo "Today date is : $(date +%D)"
Today date is 03/26/20.
```

* eg: To print enumber of files present in current working direc.

```
ls | wc -l
↑    ↑  ↑
listing count line

$ echo " Total number of files : `ls|wc -l` "
This Total number of files : 10

$ echo "Total number of files :$(ls|wc-l)"

$ echo "hello `mkdir JJ` "
hello
```

Comments in shell

```
# It is comment
# It is comment.
```

```
$ echo "$"
$
$ echo $
$
```

ctrl + l ⟶ clear.
ctrl + shift + plus ⟶ increase the font
ctrl + shift + minus ⟶ decreasing the font.

How to read dynamic data from the user :-

java : Scanner
python : input()

By using `read` keyword we can read dynamic data from the end user.

Without promt message :-

```
$ read a b
10 20 ← user i/p
$ echo "The values of a & b : $a $b"
The values of a & b are : 10 20
```

Write prompt message :

Approach (1)

```
$ gedit test.sh ⟶
$ ./test.sh
→ Enter A value :
  10
  Enter B value :
  20
  A=10
  B=20
```

```
#!/bin/bash
echo "Enter A value :" ← (/n)
read A
echo "Enter B value :"
read B
echo "A = $A"
echo "B = $B"
```

```
$ gedit test.sh ⟶
$ ./test.sh
→ Enter A value : 10
  Enter B value: 10
  A = 10  B = 20
```

```
#! /bin/bash
echo "A Value : "
echo (-n) "Enter A value :"
read A
  echo (-n) "Enter B value :"
read B
  echo "A = $A  B = $B" :
```

Approach (2)

```
read -p "Enter A value :" A
read -p "Enter B value :" B
echo "A = $A  B = $B"
```

-p ==> prompt message

-s ==> hide i/p on screen

Example :-

```
$ gedit tesh.sh ⟶
$ ./test.sh
→ Enter username noor
  Enter password Thanks
  for providing info
```

no line.

```
#! /bin/bash
read -p "Enter username" user
read -s -p "Enter password" pass
echo "Thanks for providing info"
```

```
$ test.sh
→ Enter username : noor
Enter password :
Thanks for registration
Your User Name : noor
Your Password : linux123
```

```
#! /bin/bash
read -p "Enter username : " user
read -s -p "Enter password : " pass
echo
echo "Thanks for registration"
echo "Your User Name : $user"
echo "Your Password : $pass"
```

```
read -p -s "Enter password" pass ← This will not going to work.
read -s -p "Enter password " pass ← ✓
```

Q. Write a script to read student data & display.

```
#! /bin/bash
read -p "Enter Student Name : " name
read -p "Enter rollno : " rollno
read -p "Enter marks : " mark
echo "Please confirm your details"
echo "Student Name : $name"
echo "Student Rollno : $rollno"
echo "Student Mark : $mark"
echo "$ename : $rollno : $mark" >> emp.txt
```

```
#! /bin/bash
read -p "Enter String:" str
len = $(echo -n $str | wc -c)
echo "The length of apple : $len"
```

```
#! /bin/bash
echo "Enter File Name : " fname
$cat $fname
```

```
$ test.sh
→ Enter File Name : abc.txt
  Hello world
  welcome to Linux OS
```

Q. read File name & remove blank spaces lines present in file.

→

$ grep "^Mango" abc.txt ===> display all lines starts with Mango.

$ grep "Mango$" abc.txt ===> display all lines ends with Mango.

$ grep "^$" abc.txt ===> It will display all blank lines.

$ grep -v "^$" abc.txt ===> It will display all lines except blank lines.

```
#! /bin/bash
read -p "Enter Any File Name : " fname
grep -v "^$" $fname > temp.txt   ← 1st save in temp file
mv temp.txt $fname               ← move temp file to file (fname)
```

Q read File name & remove duplicate lines present in file

```
$ sort abc.txt  ⟶
Bunny
Bunny
Bunny
sunny
sunny
sunny
Vinny
vinny
```

```
$ sort -u abc.txt
Bunny
Sunny
Vinny
```

u → unique

```
#! /bin/bash
read -p "Enter Any File Name : " fname
sort -u $fname > temp.txt
mv temp.txt $fname
echo "All duplicate lines removed"
```

1. if
2. if-else
3. nested if
4. ladder if

① if

```
if [condition]
then
    action
fi
```

```
#! /bin/bash
read -p "Enter your name" name
if [ $name = "sunny" ]
then
   echo "Hello $name good morning"
fi
echo "How are you"
```

test.sh

x=10 assignment op.
x = 10 comparision

Terminal

```
$ test.sh
Enter your name :
                sunny
Hello sunny good morn
How are you
```

```
if [ $name = "sunny" ]
   ↑ space reqⁿ ↑
```

```
if [ condition ] ; then
    action
fi
```

✓ [intendation not require]

② if-else

```
if [ condition ]
then
    action 1
else
    action 2
fi
```

③ Nested if

```
if [ condition ]
 then
    .....
    if [ condition ]
    then
        ....
    else
        ....
    fi
    ....
else
   ....
fi
```

④ ladder if

```
if [ condition 1 ]
then
    action-1
elif [ condition 2 ]
then
    action-2
elif [condition-3 ]
then
    action-3
else
    default action
fi
```

```
#! /bin/bash
echo -p "Enter no1" n1
read
read -p "Enter no2" n2
if [ $n1 -gt $n2 ]; then
    echo " n1 is greater"
else
    echo "n2 is greater"
```

`$n1 -eq $n2`	check equal
`$n1 -gt $n2`	check greater
`$n1 -lt $n2`	check less than

```
#! /bin/bash
read -p "Enter no1" (n1)
read -p "Enter no2" (n2)
read -p "Enter no3" (n3)
if [ $n1 -gt $n2 -a $n1 -gt $n3 ]; then
    echo "The biggest no is $n1"
elif [ $n2 -gt $en3 ]; then
    echo "The biggest no is $n2"
else
    echo "The biggest no is $n3"
fi
```

-a → and (&&)

-a ===> logical AND
-o ===> logical OR
! ===> logical NOT.

~~tt~~

* Switch-case

```
case $variable in
    option 1 )
        action-1
        ;;
    option 2 )
        action-2
        ;;
    option-n )
        action-n
        ;;
    *)
        default action
        ;;
esac
```

option1) ✓
option1) ✓

;; madatory for every action
but for default option it is opt°.

```
# /bin/bash
read -p "Enter weekDay no" n
case $n in
    0)
      echo "Sunday"
      ;;
    1)
      echo "Monday"
    2)
      echo "Tuesday"
    :
    *)
      echo "Wrong no"
      ;;
esac
```

"a") → only a
[abc]) → a, b or c
[^abc]) → except a b or c
[a-z]) → Any a-z
[A-z]) → Any A-Z
[a-z A-Z] → all a-z & A-Z
[0-9] → all digits. (0-9)

(optional) → *)

[^a-zA-Z0-9] → special character.

A → Display content
B → Append content
C → overwrite content
D → Delete content

cat /dev/null > abc.txt.

echo -n "" > abc.txt

```
#! /bin/bash
echo "A → Display Content"
echo "B → Append Content"
echo "C → Append Content"
echo "D → Append Content"
      Delete

read -p " choose option : " op

case $op in
    A)
      if [ -s "abc.txt" ] ; then        <- size of abc.txt
          echo "It is an
          echo " content of file "
          cat abc.txt
          echo " ..... "

      else
          echo " It is an empty file "

      ;;
    B)
      echo read -p " data to append " d
      echo $d >> abc.txt
      ;;
    C)  read -p " data to overwrite " d
      echo $d > abc.txt
      ;;
    D)
      echo -n "" > abc.txt.
      ;;
    *)
      echo " Enter Correction option"
```

Write a script to read employee data & insert into emp.txt file?

```
#! /bin/bash
while [ true ]
do
   read -p " Enter Emp no " eno
   read -p " Enter Emp name " ename
   read -p " Enter Emp salary " esal
   echo " $eno : $ename : $esal " >> emp.txt

   read -p " Do you want to insert onemore record [Y/N] " op
   case $op :
      [yY] | [yY][eE][sS] )
                  continue
                  ;;
      [nN] | [nN][oO] )
                  break
                  ;;
   .esac
done
echo " Insert Done"
```

How to read data from File by using while loop:

syntax-I

```
cat emp.txt |
while read line
do
   do something with that line
done.
```

syntax-II

```
while read line
do
   echo $line
done < emp.txt
```

```
#! /bin/bash
cat emp.txt |
while read line
do
   echo $line
done
```

If the file is empty display → it is an empty file.

```
#! /bin/bash
fname = $1
if [! -e $fname ] ; then        ← if it is exist
    echo " test: $fname: No such file or directory"
    exit 1
fi
if [ -d $fname] ; then          ← if it is directory
    echo " test: $fname: Is a directory"
    exit 2
fi
if [ ! -s $fname ] ; then       ← if it is empty
    echo " test: $fname: Is an empty file"
    exit 3
fi
count = 1
while read line
do
    echo "    $count $line"
    elet count++
done < $fname
```

Write a script to create an array with some elements and print all its elements by using while loop, for loop & adv for loop.

```
#! /bin/bash
declare -a fruits ← optional
fruits = ("Apple" "Banana" "Orange" "Mango")
```

```
echo "All elements by using while loop:"
i=0
while [ $i -lt ${#fruits[@]} ]
do
    echo ${fruits[$i]}
    let i++
done
```

While

```
echo
For fruits in ${fruits[@]}
do
  echo $fruit
done
```

For

```
for (( i=0; i< ${#fruits[@]}; i++))
do
    echo ${fruits[i]}
done
```

Add new element

```
fruits[4] = "Grapes"
```

```
Fruits[10] = "Apple"
Fruits[20] = "Banana"
Fruits[40] = "Mango"
```

```
For i in ${!fruits[@]}
do
    echo ${fruits[$i]}
done
```

Accessing values based on index.

unset fruits[40] ← delete.

Write a script to store given n no. in array.

```
#! /bin/bash
read -p "Enter Number of values : " n
for ((i=0,j=1; i<n; i++))
do
    read -p "Enter the no: " NUM[$i]
                              $[j++]
done.

echo "Array Elements: "${NUM[@]}"
```

```
$ test.sh
Enter Number of values : 5
Enter the no 1 : 10
Enter the no 2 : 20
Enter the no 3 : 30
Enter the no 4 : 40
Enter the no 5 : 50
Array Elements : 10 20 30 40 50
```

Q. Write a script to store all .txt file names present in curr working dire. into an array & print permission of every file.

abc.txt	READ(Y)	WRITE(Y)	EXECUTE(N)
hyd.txt	READ(Y)	WRITE(N)	EXECUTE(N)
res.txt	READ(Y)	WRITE(N)	EECUTE(Y)

```
#! /bin/bash
files = ($(ls *.txt))
for fname in ${files[@]}
do
  echo -ne "$fname:\t"
  if [ -r $fname]; then
     echo -ne "READ(Y)\t"
  else
  fi  echo -ne "READ(N)\t"
  if [-w $fname] ; then
     echo -ne "Write(Y)\t"
  else
     echo -ne "WRITE(N)\t"
  fi

  if [-x $fname] ;then
     echo "EXECUTE(Y)\t"
  else
     echo "EXECUTE(N)\t"
  fi
```

Q. Write a function to find maximum of two given numbers.

```
#! /bin/bash
max()
{
  if [$1 -gt $2]; then
     echo "Max no is $1"
  else
     echo "Max no is $2"
  fi
}
max 10 20
read -p "Enter no 1" n1   read -p "Enter no 2" n2   max $n1 $n2
```

```
$ test.sh
Max no is 20.
```

Factorial.

```
#! /bin/bash

factorial()
{
    original = $1
    n = $1
    fact = 1
    while [ $n -gt 1 ]
    do
        let fact = fact * n
        let n-1
    done
    echo " The factorial of $original : $fact"
}
factorial 5
read -p " Enter no : " n
factorial $n

for i in {1..10}
do
    factorial $i
done.
```

Prime Number

2.

```
#! /bin/bash
prime_check()
{
    n=$1
    if [ $n -le 1 ]; then
        echo "$n is not a PRIME no"
    else is_prime = "yes"
        for ((i=2; i<n; i++))
        do
            if [ $[n%i] -eq 0 ]; then
                echo "$n is not a PRIME no"
                break  is-prime="no"
        done
        if [ $is_prime = yes ]; then
            echo "$n is a PRIME no"
        fi
    fi
}
prime_check 17  ---->  17 is a not a PRIME no
```