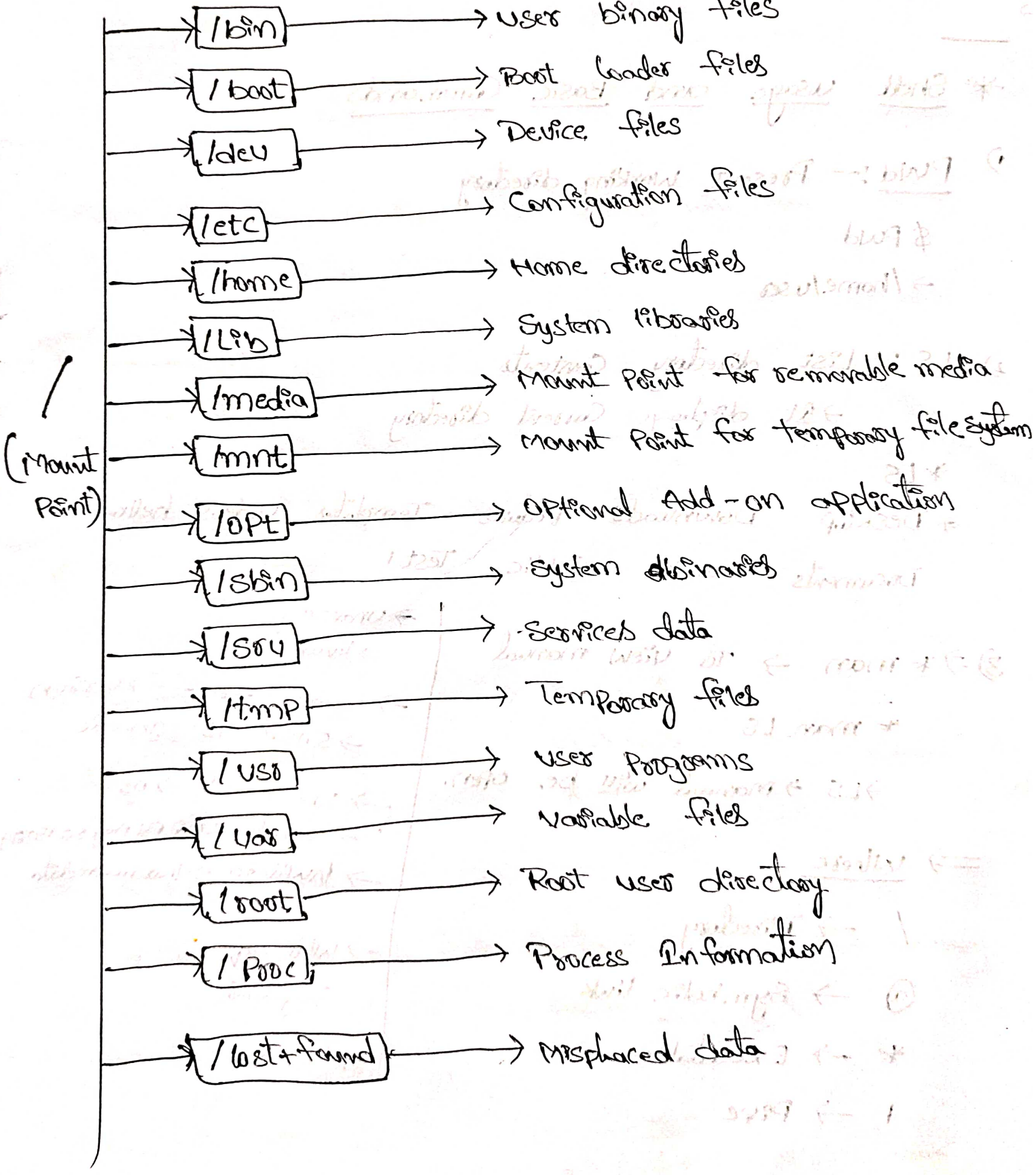
* Linux directory Structures

→ Linux keeps all the devices and data in form of files in different locations.

→ It Contains a root directory → Sub-directories → files.



* Work on Terminal tool in ubuntu Software.

* Who am i
  → Jagadeesh

* Uname
  → Linux

---

* Shell usage and Basic Commands

1) Pwd :- Present working directory

   $ Pwd
   → /home/user

2) LS :- List directory Contents
   → It displays Current directory

   * LS
   ⇒ Desktop Downloads Pictures Templates videos hello.c
   Documents music Public Test1

3) ⇒ * man → To view manual
   * man LS
   → LS → Manual will be open.

⇒ Where

/ → Directory

@ → Symbolic link

* → Executable

| → Pipe

→ uname
  -linux

→ uname -r --→ version
  → 5.15.0-48-generic

→ uname -Y → os
  → ubuntu SMP Fri Aug 26 13:26:29

→ hwinfo ⇒ hardwaredata

→ who -a
  Jagadeesh

→

* LS - Commands

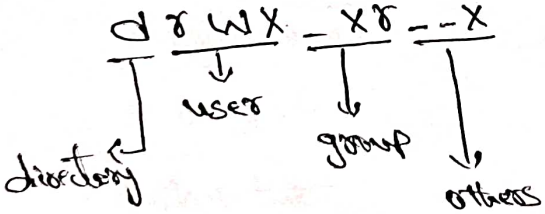
4) LS -L → long listing
   → all directory files with full details

5) LS -a → all files in the directory
   → It shows folder and files

6) LS -lh
   → all directory files memory will shows
   → 4KB
   → 11KB

7) ~~LS -lh/etc~~
   LS -lh /etc
   → It shows specific directory
   → In human readable form.

8) LS -ld /etc
   →



⇒ r → read
w → write
x → Excute

9) LS -R /etc
   → To list out all sub directories in a list in a directory

10) LS -i → unic Id for every directory
    Eg:- 868508 desktop
         788586 details

11) LS -n → It shows user Id
    Group Id
    size memory
    created date

Eg:-
d rwx r-x 2 1000
84 Sep 11 20:10
desktop

* Cd - Commands

Cd → To change directory

→ Cd to change directory

* Cd _ Documents/

⇒ Jagadeesh@documents

→ Cd .. → To come back

→ cd ~ → home directory

---

* Files :-

→ To create files.

* touch _ file1

* touch _ file1 file2 file3 ....

(or) * cat > file1

* Folders : → To create folders.

* mkdir _ Linux

* mkdir _ L1 L2 L3 ....

* Delete : → To delete folder / files

* rm _ file1

* rmdir → To empty directory

* rm _ Linux

* Edit

* nano _ filename

* Vi _ file name

* vim _ file name

* Copy
  * cp
  ⇒ cp_ folder1_ raju

* move
  * mv
  * mv_ f4/home/Jagadeesh/downloads/n2

* Properties
  * stat_ f3 (or)
  * ls_ -l (long list)

* To check file/folder → Information
  cat_ f1.txt

---

* How to get help from terminal - Commands
  * man ls
  * man man
  * ls_ -a
  ⇒ ch + tab

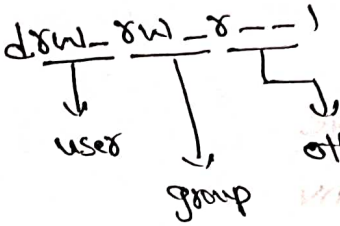
⇒ apropos_ copy → (To get any information from Command)

* Files and directory Permissions

⇒ In linux every think is a file (directory ls -l (long listing)

- → File
d → directory
L → link

r → read
w → write
x → execute



→ To give Permission

* Command → chmod +x

→ To remove Permission

* Command → chmod -x

0 = 0 = nothing
1 = 1 = execute
2 = 2 = write
3 = 2+1 = w + x
4 = ~~3+1~~ = read
5 = 4+1 = read + execute
6 = 4+2 = read + write
7 = 4+2+1 = r+w+x.

---

* System logs :-

→ DNS → (Domain name System)

** Package management :-

⇒ Sudo apt update

⇒ Sudo apt list upgradable

⇒ Sudo apt install terminator

⇒ Sudo apt update

---

* How to add new user to linux.

→ add user Mohan

* Sudo su

To remove user

→ userdel -r Mohan.

* Visual Editor → Vi

⇒ Vi improved version is called vim

⇒ Vi basic Commands

ESC → Escape → Search file edit

i/I → Edit → Insert remove
a/R

v/u → visual → selection

: → Comman line → Commands

→ vi_file.txt

→ ESC + :wq! → Save

→ ESC + :q! → quit

→ ESC + i+u → undo

→ ESC + V → select the text

→ Cat_file.txt

⇒ data will display in terminal → which is Stored in file.txt

* Append - a

Eg:- Hey all ! let do some Vimming

Press a

Hey The text will insert here all ! let do some Vimming

---

* Vi → Copy and Past :

ESC + YY and P

---

* Vi → Open

ESC + O

---

* Vi — Go

gg - Go to nth line → Say n = 10

→ gg takes you to 10 line

gg - take you first line of the file

G - take you the last line of the file

* Vi - Delete

→ D → the Current from the Cursor Position

dd → Delete n line.(s) ⇒ (Say n = 10)

10 dd delete 10 lines.

---

* Vi — Decrement

⇒ You may increment and decrement n lines.

CTRL x → Increments the first integer match from the Cursor Position

Say n=10 and number is 13,

10 CTRL x decrements the number by 10, resulting to number3

→ The increments works the same way.

---

* Vi — Navigation - Forward

→ [1] Line hey all ! let do some → Press W

Cursor

→ 1 line [h]ey all! let do some → Press 3W

→ 1 line hey all ![l]et do some.

⇒ move forward n words → Say

n=10W - move 10 words ahead.

* Same way for → Navigation - Backward.

* Vi → Change word

⇒ cw

Eg:- line Hey all !, let do some vimming → cw

line ... all !, let do some vimming

* ⇒ cw → change n words → Say n = 10cw -

change 10 words from the Current ~~cursor~~ Cursor Position.

---

* Vi → Delete word

⇒ dw

---

* Vi → Settings

ESC + :

:set hls → Enable highlight search

:set nohls → Disable highlight search

:set nu → Enable line numbers

:set nonu → Disable line numbers.

* Vi — Substitute

→ change small to big

1 Eg:- let do some vimming ⇒ ESC :
let do some vimming %s/vimming/VIMMING/g

⇒ let do some VIMMING
" " " VIMMING

2 Eg:- :2s/do/be/g

let do some VIMMING
let be some VIMING.
↵
only second line.

---

* Vi — Edit

:e filename → open another file without closing the current file.

---

* Vi — read

:r filename - reads file named filename at the current cursor position.

* Shell Scripting :-

→ There are various types of Programming languages, Compared on various Parameters.

→ Assembly language (ex: 8051)

→ Middle level language (ex: c)

→ High level / scripting language (ex: shell)

* What is a script?

→ Any Collection of shell Commands can be stored in a file, which is then called as shell script.

→ Programming the script is called shell scripting

* Where script to use?

* System Administration
  - → Automate tasks
  - → Repeated tasks
* Development
  - → Allows testing a limited sub-set
  - → Testing tools
* Daily usage
  - → Simple Scripts
  - → Reminders, e-mails etc...

→ #!/bin/bash
↓ ↳ directory
Comment

→ echo → Print.

---

* echo. \n

→ echo Hello\nWorld → \n → newline

Hello
World

* \t

→ echo Hello\t World → \t → tab Space

Hello _ World

* \r

→ echo Hello\r world → \r → overwrite

world

* Special characters

$ → used to access a variable.

→ echo $0 # expand to name of the shell (or) shell script

/bin/bash

→ echo $$ # expands to this shell Process id

2668

→ echo $? # expands to this status of the Previous

0 Command.

Eg:-

→ x=10

echo $x

10

→ Name="EMERTXE"

echo $Name

EMERTXE

→ Test1=10; Test2=20

echo $Test1 $Test2

10 20

**:-

→ $@ → value of all arguments Passed

→ $# → No. of arguments Passed to shell script

→ Eg:-

→ mkdir TBD

→ cd TBD

→ touch file_{1..5}.txt

→ touch file_{6..10}.c

→ touch file_{1..5}.sh

→ ls

→ It displays all
1-5 .txt files
6-10 .c files
1-5 .sh files

→ *.txt

→ It displays all
1-5(.txt files).

→ *1*

→ It displays which file contains 1

→ file_?.c

→ It displays .c files only
6 7 8 9 (only 1?)

→ file_??.c

→ 10 (2?)

* The Shell ENV Variables.

1) echo $HOME → The Current user's home directory
/home/Jagadeesh

2) echo $ SHELL → Shell that will be interpreting user Commands
/bin/bash

3) echo $ USER → The Current logged user.
user

4) echo $ PWD → The Previous working directory.
/home/user/ECEP/LS/classwork

5) echo $OLDPWD → The Previous working directory
/home/user.

6) echo $ PATH → System will check when looking for Commands here
/user/local/sbin

7) echo $ HOSTNAME → The host name of the Computer
Jagadeesh

8) echo $ TERM → Type of terminal to emulate when running the shell

* Shell Scripting - Expressions

→ expr 10 + 20 ⇒ Addition

30

→ expr 10 * 20 ⇒ Multiplication

200

→ expr 10 - 20 ⇒ Subtraction

-10

→ expr 20 / 10 ⇒ division

2

---

* expr → Evaluates simple math on the Command line Caculator.

* bc → An arbitary Precision Calculator language.

*Eg of Expression - script

```
#!/bin/bash

Num1=5

Num2=3

ADD=$((${Num1} + ${Num2}))
SUB=$((${Num1} - ${Num2}))
MUL=$((${Num1} * ${Num2}))
DIV=$((${Num1} / ${Num2}))
MOD=$((${Num1} % ${Num2}))

echo -e "Addition of two numbers is \t\t: ${ADD}"
echo -e "Subtraction of two numbers is \t\t: ${SUB}"
echo -e "Multiplication of two numbers is \t: ${MUL}"
echo -e "Division of two numbers is \t\t: ${DIV}"
echo -e "modulum of two numbers is \t\t: ${MOD}"
```

→ chmod +x filename.sh

→ ./expression.sh

⇒ Addition of two number is : 8
Substration of two number is : 2
Multiplication of two number is : 15
Division of two number is : 1
Modulum of two number is : 2

* Logical Operations

→ And && → both Conditions must be true

→ Or || → any one should satisfy

---

* Head & Tail

Eg:- file.txt

abc
123
786
495
112
186
220

Head -5 file.txt

abc
123
786
495
112

Tail -3 file.txt

112
186
220

* Grep & Pipe

⇒ Pipe : | → (enter up button)

→ Two (or) more Commands to Pass in a Single time.

→ The output of one Command is Input to another Command.

Eg:- file.txt

```
123
abc
x42
456
lbb
LIC
out
one
```

⇒ cat file.txt | head -2 file.txt

```
123
abc
```

⇒ grep → Global Search for regular expression & Printout.

1. Eg:- file.txt

Tata
Mahindra
Soda
Bmw
cycle

$ grep "Tata" file.txt

⇒ Tata

2. Eg:- $ grep "soda" file.txt -n → number.

⇒ 3 Soda

---

* du → To estimate file space usage
* df → helps us to find System disk space usage
* stat → To display file or file system status

* Redirection :-

→ Redirection is nothing but diverting the output from original location to some other location.

→ From a file to a Command - Input Redirection → <

→ From a file to a Command - output Redirection → >

Eg:- ls

string.sh while.sh array.sh.

→ echo Hello > test.txt

string.sh while.sh array.sh test.txt

⇒ cat text.txt

Hello

→ echo world > test.txt

cat text.txt

world

→ echo Hello >> test.txt

- cat text.txt

world

Hellowo

* ln

→ It helps us to make links between file.

⇒ Two types of links are Possible.

→ Hard link → only on files

→ Soft link → both on files and directions

⇒ Hard link:-

*** → Numeric Test operators

Operator		Function
-eq	→	Eaual
-ge	→	Greather than or Eaual to
-le	→	Lessthan or Eaual to
-ne	→	not Eaual
-lt	→	less than
-gt	→	Greather than
=	→	Equal
!=	→	Not Eaual
-n	→	greather than zero
-z	→	~~Equal~~ Eaual to zero
-a	→	AND logic expression
-o	→	OR logic expression.

* Shell - Scripting :-

=#=! /bin/bash → Command will exist in bash cell.
↓
Comment

=#= → Comments are unexpected lines, will helps Programmer to understand code easily.

echo → Command used to Print any lines in output just like Print.

⟹ Variables "Shell Scripting":

→ We can store any type of data :

Eg: =#=! /bin/bash

a = "Hello"
Variable ↵ ↳ String

echo "$a"

⇒ Hello

=#! bin/bash.

echo "What is your name?" → What is your name?

read name → Jagdeesh.

echo "How do you do $name?" → How do you do, Jagdeesh?

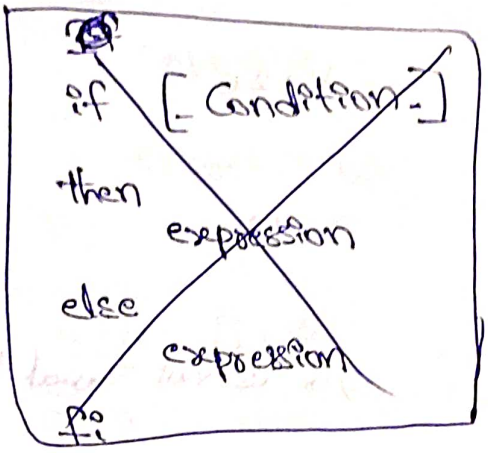
read remark → good.

echo "I am also $remark too" → I am also good too.

* Shell scripting – Conditions – if else

⟹ if - Statement

→ Syntax



→ Syntax

```
if [ Condition ]
then
    expression
fi
```

Eg:-

→

```
#!/bin/bash
a=10
b=20
if [ $a == $b ]
then
    echo "a is equal to b"
fi
if [ $a != $b ]
then
    echo "a is not equal to b"
fi
```

⟹ a ≠ b

3 Eg:-

```
#!/bin/bash
echo "Enter the number a"
read a
echo "Enter the number b",
read b
If [$a == $b]
then
    echo "a is Eaual to b"
fi
if [$a != $b]
then
    echo "a is not Eaual to b"
fi
```

→ a= 10

→ b=20

a ≠ b.

⇒ a is not Eaual to b.

* if else

⇒ Syntax

```
if [ Condition ]
then
    expression
else
    expression
fi
```

Eg:-1

```
#!/bin/bash
a=10
b=20
if [ $a == $b ]
then
    echo "a is eaual to b"
else
    echo " a is not eaual to b"
fi
```

→ a is not eaual to b.

Eg2:-

```
#!/bin/bash.
Num1=5
Num2=3
if [ ${Num1} -gt ${Num2} ]
then
    echo "Num1 is greater than Num2"
else
    echo " Num2 is greater than Num1"
fi
```

⇒ Num1 is greater than Num2

* if else if

Syntax

```
if [ Condition_a ]
 then
      Condition_a is true
 elif [ Condition_b ]
 then
      Condition_b is true
 else
      both false
 fi
```

Eg:-

```
#!/bin/bash
Num1=5
Num2=3
if [ ${Num1} -eq ${Num2} ]
then
    echo "Num1 is equal to Num2"
elif [ ${Num1} -gt ${Num2} ]
then
    echo "Num1 is greather than Num2"
 else
    echo "Num1 is less than Num2"
 fi
```

⇒ Num1 is less than Num2

* Case

→ The case statement compares the value of the variable ($var in the case) to one or more values.

Syntax

```
case ${VAR} in
    value-1)
        Commands;
        ;;
    value-2)
        Commands;
        ;;
    *)
        Commands;
        ;;
esac
```

Eg:-

```
#!/bin/bash
echo "Enter a car name BMW Lambo fortuner"
read car
case "$car" in
    "Lambo") echo "Head avatos in USA";;
    "BMW") echo "Head avatos in India";;
    "fortuner") echo "Head avatos in germeny
esac
```

| → BMW
→ head avatos in India.

* String Tests

```
#!/bin/bash
echo "Enter the first string"
read STR1
echo "Enter the second string"
read STR2
if [ -z ${STR1} ] ; then
    echo "First string is empty"
else
    echo "Second string is not empty"
fi
if [ -n ${STR2} ] ; then
else
    echo "second string is empty"
fi
if [ ${STR1} = ${STR2} ] ; then
    echo "Both string are eaual"
else
    echo "Both strings are not eaual"
fi
```

① → Hello
② → World
First string is not empty
second string is not empty
Both strings are not empty

① Hello
② Hello
First string is not empty
Second string is not empty
Both strings are not empty

* Logical Operators :-

```
#!/bin/bash
echo "Enter the first number A"; read A
echo "Enter the second number B"; read B
echo "Enter the third number c"; read C

if [ ${A} -gt ${B} -a ${A} -gt ${C} ]; then
    echo "A is the greatest of all"
elif [ ${B} -gt ${A} -a ${B} -gt ${C} ]; then
    echo "B is the greatest of all"
elif [ ${C} -g ${A} -a ${C} -gt ${B} ; then
    echo "c is greatest of all"
else
    echo "Invalid Input"
fi
```

A → 10
B → 2
C → 3
A is greatest of all

A → 3
B → 2
C → 3
B is the greatest of all

* Shell - scripting - loops - for.

→ The structure is a looping structure. used to execute a set of commands while the provided list is empty.

⇒ Syntax

```
for i in list
do
    Code block
done
```

Eg:-

```
#!/bin/bash.
for i in 1 2 3 4 5
do
    echo "Loop Counter is ${i}"
done
```

⇒ bash file.sh.

→ Loop Counter 1
→ Loop Counter 2
→ Loop Counter 3
→ Loop Counter 4
→ Loop Counter 5

Eg:-

```
for a in 1 2 3 4 5 6 7 8
do
    if [ $a == 5 ]
    then
        break
    fi
    echo "iteration number $a"
done.
```

⇒ bash file.sh.

iteration number 1
" " 2
" " 3
" " 4. → break at (5)

```
for a in 1 2 3 4 5 6 7 8
do if [ $a == 5 ]
    then
        Continue
    fi
    echo "iteration number $a"
done
```

* bash file.sh

⇒ iteration number 1
" " 2
" " 3
" " 4
" " 6
" " 7
" " 8

(5)

* While-loop

→ The structure is a looping structure.. used to execute a set of commands while a specified condition is true.

⇒ Syntax

```
while [ condition ]
do
    code block
done
```

Eg:-
```
#!/bin/bash
loop=1
while [${loop} -le 5]
do
    echo "looping : ${loop}"
    Loop=$((${loop}+1))
done
```

⇒ looping : 1
looping : 2
looping : 3
looping : 4
looping : 5

```
#!/bin/bash
a=1
while [ $a -lt 10]      → until the condition satisfy
do
    echo $a
    a=`expr $a+1`
done.
```

⇒ 1
2
3
4
5
6
7
8
9

* Command line Arguments

→ Shell script can accept Command-line arguments & options just like other linux Commands.

→ Can refer to these arguments as
$1, $2, $3 ---- & so on.

→
```
#! /bin/bash
if [ $# !=2 ]
then
    echo "usage: Pass 3 arguments"
    exit 0
fi
echo "The arguments of the script you passed are :"
echo "Total number of arguments you Passed are : $#"
echo "The name of the script is : $0"
echo "The first argument is : $1"
echo "The Second argument is : $2"
```

=
```
$ bash file.sh Hello 1234

2
bash file.sh
Hello
1234
```

* Shell Scripting — Functions

→ Writing functions can greatly simplify a Program

→ Arguments are accessed as $1, $2, $3 ....

⇒ Syntax

```
function name ()
{
    < command >
    < statements >
    < expression >
}
```

```
#!/bin/bash.
function sum ()
{
    x=`expr $1 + $2`
    echo $x
}
y=`sum 5 3`
echo "The sum is 5 and 3 is $y"
echo "The sum is 6 and 2 is `sum 6 2`"
```

→ The Sum is 5 and 3 is 8

→ The Sum is 6 and 2 is 8

* Sort

→ It filters data into right order.

Eg:- file.sh

d
c
e
a
b

$ sort file.sh

a
b
c
d
e

file.sh

5
3
4
1
2

$ sort file.sh

1
2
3
4
5

* User: → Management [add user]

#useradd Mohan

#user add Raghu

→ To check for users

#cat. /etc/Passwd

→ su Mohan → shift to Mohan.

#Passwd Mohan.! ...

New Password : xxxxx

Retype Password : xxxxx

* AWK Command

→ The AWK Command is a linux tool and Programming language that allows users to Process and manipulate data and Produce formatted repots.

→ vi details → file

```
ab  cd  ef  1000
eg  hi  gi  2000
rt  vi  ji  4000
```

→ cat details

```
ab  cd  ef  1000
eg  hi  gi  2000
rt  vi  ji  4000
```

→ awk '{Print}' details

```
ab  cd  ef  1000
eg  hi  gi  2000
rt  vi  ji  4000
```

⇒ awk '{Print NR, $0}' details

```
1 ab  cd  ef  1000
2 eg  hi  gi  2000
3 rt  vi  ji  4000
```

→ awk '{Print $NF}' details

```
1000
2000
4000
```

→ awk '{Print $1, $3}' details

```
ab  ef
eg  gi
rt  ji
```

* Zip & unzip Files :-

→ zip

$ touch file1 file22 file3 file 4

zip -r file1 file2 file3 file4

→ unzip

→ unzip file (or) gunzip file.

---

* who, whoami, man and where

→ logname
→ Jagadeesh

→ whoami
→ Jagadeesh

→ who
→ Jagadeesh tset 2.20

→ finger
→ more details

→ which ls
↓
directory location

→ man ls
→ complete details

→ man cal

* Cut and Past

⇒ Cut

⇒ cat raju

```
1   3   13
4   5   12
6   7   11
8   9   10
```

⇒ cat babu

```
100   101   102
103   104   105
106   107   108
```

Cut -f 1 raju

```
1
4
6
8
```

Cut -f 2 raju

```
3
5
7
9
```

Cut -f 1-3 raju

```
1   3   13
4   5   12
6   7   11
8   9   10
```

⇒ Past

Past raju ganesh.

```
1  3  13 ; 100  101 102
4  5  12 ; 103  104 105
6  7  11 ; 106  107 108
8  9  10 ;
```

Paste -s raju

```
1 3 13  4 5 12  6 7 11  8 9 10
```

Paste -d "," -s ganesh.

```
100 101 102, 103 104 105, 106 107 108.
```

* To increase CPU -Speed

top

⇒ CPU . 2.660 6.0

⇒ To increase Speed

Yes>/dev/null &

⇒ CPU -78.

---

* Date and Calender Commands:-

* Date:

⇒ date

⇒ Sun 18 Sep 07:10:20 AM EDT

⇒ date +%D

⇒ 04/15/20

⇒ date +%d

15

⇒ date +%Y

2020

⇒ date +% H

07.

⇒ date +%.S

20

⇒ date +%.M

⇒ 10

⇒ date +%.m

⇒ 09

* Calender :-

* Cal

→ Present month Calender

⇒ Cal 2020

all months of 2020 calender

⇒ Cal January 2020

2020 January Calender will display.

* All Commands in LS

→ LS
⇒ all directory files will display
Eg:- Desktop downloads documents

→ LS -L ⇒
⇒ All directory files will full details

→ LS -lh
⇒ All directory files memory will shows
→ 4kb
→ 11kb.

→ LS -lh /etc
↳ Specific directory
⇒ In human readable form

→ LS -Ld /etc

drwx-xw-x ----

→ LS -a
⇒ All hidden files.

→ LS -R/etc
To list all sub directories in a list in a directory

→ ls -i
→ unic id
868508 desktop
788586 details

→ ls -n
⇒ User Id
⇒ Group Id

* Regular expressions ⇒

→ #, *, ?, [ ]

⇒ ls
ganesh babu babu basha → files

⇒ ls *b
babu basha

⇒ ls l*
Lalu

⇒ ls *h
ganesh

⇒ ls g*h
ganesh.

?

→ ls
ganesh babu basha

→ ls g?
error

→ ls ganes?
ganesh

→ ls ???u
babu

[ ]

→ ls [abcdefgh] anesh
⇒ ganesh

→ ls gan [abcdefh] s
⇒ ganesh.

* Sed ⇒ Command. → Stream editor

⇒ for filtering and transforming text

→ cat seshu-file

Tenali is a beatiful village

He is a he noble Personality

⇒ sed "s/He/hoo/i" seshu

Tenali is a beautiful village

hoo is a hoo noble Personality

⇒ sed "s/He/hoo/gi" seshu (g → globally)

Tenali is a beautiful village

hoo is a hoo noble Personality

⇒ sed "s/^$/ Maheshbabu/g" seshu.

Tenali is a beautiful village

maheshbabu

He is a he noble Personality

⇒ sed -n '$p' seshu

He is a he noble Personality

⇒ sed -n '1p' seshu

Tenali is a beautiful place

⇒ sed -n '2p' seshu

------ empty ⇒ 2nd line.

⇒ sed '2d' seshu

Tenali

He is ---→ line deleted

Sudo → Super user → Execute a Command as other user.

1) GUI → Geographical user Interface

2) CLI → Command line Interface.

3) VI → Visual editor

4) VIM → Advance Version of VI

5) Cat → Concatenate files and Print on the Standard output.

6) env → run a Program in a modified environment

7) expr → evaluate expressions, Simple math on the Command line Calculator.

8) bc → An arbitrary Precision Calculator language

9) head → output the first Part of files

10) tail → output the last Part of files.

10) du → disk usage

11) df → file data

12) Stat → display file or file system status

13) Sed → Stream editor → for filtering text

14) Sort → It fitless data into right order

15) Pipe → Two (or) more Commands to Pass in a single time

16) Grep → Global Search for regular Expression & Printout

17) Ssh → Secure Shell

18) Scp → Secure Copy

19) Find → filter Command

20) tr → translate (or) delete characters.

21) Chown → change file owner and group.

22) Shuf → Generate random Permutations

1) Database:

→ A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

→ A database is usually controlled by a database management system (DBMS)

2) Variable:

→ A variable is a value that can change, depending on conditions passed to the program.

3) String:

→ A sequence of characters as some kind of variable.

4) Script.:

→ A program (or) sequence of instructions that is interpreted by the computer processor.

5) Condition:

→ A condition is something that a computer can decide is either true (or) flase.

6) Statement:

→ A single of code that performs a specific task.

7) Case:

→ The case statement compares the value of the variable to one (or) more values.

8) Function:

→ A function is block of code that Performs a task.

* Why linux?

→ open source operating system

→ access to source code.

→ highly secure

→ Runs fast

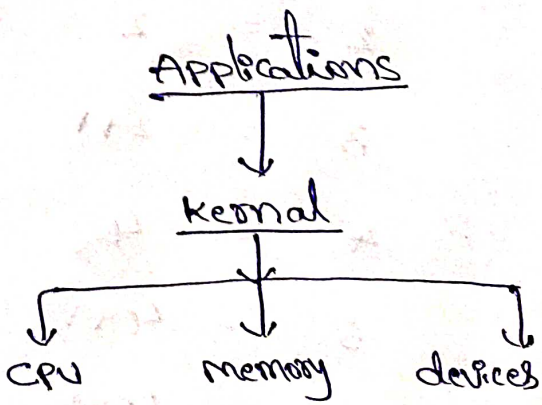
* What is linux?

→ linux is a unix-like, open source and community-developed operating system for which is capable of handling activities from multiple users at the same time.

* What is a kernel?

→ The Computer Programs that allocate the system resources and co-ordinate all the details of the Computer's internals is called the operating system (or) kernal.

→ users communicate with the OS through a Program called Shell



* Command Line Essentials

* Terminal

- → cd
- → Pwd
- → ls
- → cp
- → mv
- → rm
- → echo
- → cat
- → less
- → grep
- → mkdir
- → touch
- → chmod
- → man + help

* change mode for file Permissions.

drwx-rwx-rwx chmod ... filename.

d — directory; rwx — user; rwx — group; rwx — others.

0	=	0	= nothing
1	=	1	= execute
2	=	2	= write
3	=	2+1	= write + x
4	=	4	= read
5	=	4+1	= r + x
6	=	4+2	= r + w
7	=	4+2+1	= r + w + x

* Command line interface

→ Consde representation

→ Difficult for beginners

→ Faster os

→ Granular Control

* Graphical user Interface

→ Graphical representation

→ Easy of use

→ os is slower

→ lesser Control

* What is CLI?

→ CLI → Command line interface used to interact with software and operating system by typing commands into the interface and recive a response in the same way.

* What is Shell?

→ The shell is a Command line Interpreter.

→ It translates Commands entered by the user and converts them into a language that is understood by the kernel.

It takes Input from user → Translate Commands → language understand by the kernel.

* What is a shell script?

⇒ The basic concept of a shell script is a list of Commands, which are listed in the order of execution. A good shell script will have Comments. Preceded by #sign. for describing the steps

* Basic shell scripting:

```
#!/bin/bash
echo "What is your name?"
read Person.
echo "Hello, $ Person"
```

| ⇒ Jagadeesh

Hello, Jagadeesh.

* What is a Variable?

→ A Variable is a character string to which we assign a value.

→ The value assigned could be a number, text, filename, device, or any other type of data

* Types of Variables.

→ local variable

→ Environment Variable.

→ Shell Variable

⇒ Local variable

→ A local variable is a variable that is present within the current instance of the shell.

→ It is not available to programs that are started by the shell.

→ They are set at the command prompt.

⇒ Environment Variable

→ An environment variable is available to any child process of the shell.

→ Some programs need environment variables in order to function correctly.

* Shell Variable?

→ A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly.

→ Some of these variable are environment variables where as others are local variables

* Using Variables

→ Defining Variables
→ Special Variables
→ Command line Arguments
→ Special Parameters
→ Exit status.

Eg:- #!/bin/bash

Name="Jagadeesh"
echo $Name

./variable.sh ⇒ Defining variable.
→ Jagadeesh.

---

* Special variables

$0 → file name of the script
$# → No. of arguments (total value)
$* → All the arguments are double cotted
$@ → Value of all arguments
$? → last Command status
$$ → Process number of Present shell script.

Eg:-

#!/bin/bash.

⇒ nano spvarr.sh filename

```
#!/bin/bash
echo "File name : $0"
echo "First Parameter : $1"
echo "second Parameter : $2"
echo "Quoted Value : $@"
echo "Quoted Value : $*"
echo "No of Parameters : $#"
```

output

./spvarr.sh

Jagadeesh

Chinna

Jagadeesh Chinna

Jagadeesh Chinna

2

Eg:-

nano forloop.sh

```
#!/bin/bash

for TOKEN in $*
do
    echo $TOKEN
done
```

output

./forloop.sh hai hello you

→ hai
hello
you

* Basic operators

→ Arthmetic operators
→ Relational operators
→ Boolean operators
→ String operators
→ File test operators.

⟹ Arthmetic operators

→ + (Addition) ⟶ 'expr $a + $b'
→ - (Substraction) ⟶ 'expr $a - $b'
→ * (multiplication) ⟶ 'expr $a * $b'
→ / (Division) ⟶ 'expr $b * $a'
→ % (modules) ⟶ 'expr $b % $a'

→ = (Assignment) ⟶ a=$b (( ))
→ == (Equality) ⟶ $a == $b
→ != (Not Equality) ⟶ $a != $b

⇒ Relational operators : [ ]

→ -eq [equal to] ⟶ [$a -eq $b] → not true

→ -ne [not equal to] ⟶ [$a -ne $b] → is true

→ -gt [Greater than] ⟶ [$a -gt $b] → is not true

→ -lt [less than] ⟶ [$a -lt $b] → is true

→ -ge [greatthan equalto] ⟶ [$a -ge $b] → not true

→ -le [lessthan equal to] ⟶ [$a -le $b] → is true

⇒ Boolen Operators

→ ! → logical negation ⟶ [! false] is true

→ -o → logical OR ⟶ [$a -lt 20 -o $b -gt 100] is true

→ -a → logical AND ⟶ [$a -lt 20 -a $b -gt 100] is false

⇒ String Operators [ ]

==	→ Equal	→ [$a = $b]	is not true
!=	→ Not Equal	→ [$a != $b]	is true
-z	→ Operator size is zero	→ [-z $a]	is not true.
-n	→ Size is non zero	→ [-n $a]	is not false.
str	→ String is Empty or not	→ [$a]	is not false

⇒ File Test Operators:-

1) -b file → [-b $file] false
2) -c file → [-c $file] false ----> character.
3) -d file → [-d $file] not true ----> directory
4) -f file → [-f $file] true ----> file
5) -g file → [-g $file] false ----> group Id
6) -k file → [-k $file] false
7) -p file → [-p $file] false ----> Pipe
8) -t file → [-t $file] false ----> Terminal
9) -u file → [-u $file] false ----> user ID
10) -r file → [-r $file] true ----> readable

11) -w file ⟶ [-w $file] ------> writable

12) -x file ⟶ [-x $file] ------> executable

13) -s file ⟶ [-s $file] ------> size

14) -e file ⟶ [-e $file] ------> exists

---

* Shell Loops

⟶ The while loop

⟶ The For loop

⟶ The until loop

⟶ Nested loops

⟶ loop Control

* For loop :-

The for loop operates on list of items. It repeats set of commands for every item in the list

→ Syntax

```
for [ condition ]
then
    expression
done
```

Eg:-

```
#!/bin/bash
for i in 1 2 3 4 5 6
do
    echo $i
done
```

⇒ output

```
1
2
3
4
5
6
```

⇒ while loop :

It enables to execute a set of commands repeatedly until some condition occurs.

→ It is usually used when you need to manipulate the value of a variable repeatedly

→ Syntax:

```
while [ condition ]
do
    code block
done
```

Eg:-

```
#!/bin/bash
a=0
while [ $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

output

```
0
1
2
3
4
5
6
7
8
9
```

⇒ Until Command

until Command → same as while loop.
do
    statement
done

```
#!/bin/bash

a=0
until [! $a -lt 10]

do
    echo $a
    a=`expr $a +1`

done.
```

output

```
0
1
2
3
4
5
6
7
8
9
```

⇒ Nested loop:-

→ Can do another loop in same loop

→ nested loop can use unlimited times.

eg:-

```
#!/bin/bash

a=0
while ["$a" -lt 10]
                0-9
do
    b="$a"
    while ["$b" -ge 0]
    do
        echo -n "$b"
        b=`expr $b - 1`
    done
    a=`expr $a +1`
done
```

=>a=0
b=0
b=1
2
3

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

* Loop Control:

break

nano break.sh.

```
#!/bin/bash
a=0
while [$a -lt 10]
do
    echo $a
    if [$a -eq 5]
    then
        break.
    fi
    a='expr $a +1'
done
```

⇒ 0
1
2
3
4
5.

Continue

nano Continue.sh.

```
#!/bin/bash
nums="1 2 3 4 5 6"
for i in $nums
do
    Q='expr $i % 2'
    if [ Q -eq 0]
    then
        echo "Number is even number"
        Continue
    fi
    echo "Found odd number"
done
```

⇒ odd
even
odd
even
odd
even

* Shell functions:

→ Creating functions
→ Passing Parameters to function
→ Returning values from function
→ Nested functions
→ Function Call from Prompt

Functions:

→ Writing functions can greatly simplify a Program
→ Arguments are accessed as $1, $2, $3 ------

→ Syntax

```
function name( )
{
    <Command>
    <Statement>
    <expression>
}
```

eg:-

```
#!/bin/bash
Hello( )
{
    echo "Hello world"
}
```

→ output
→ Hello world.

* Shell functions:

→ Creating functions

→ Passing Parameters to function

→ Returning values from function

→ Nested functions

→ Function Call from Prompt

Function:

→ Writting functions can greatly simplify a Program

→ Arguments are accessed as $1, $2, $3 ------

→ Syntax

```
function name( )
{
    <Command>
    <Statement>
    <expression>
}
```

Eg:-

```
#!/bin/bash
Hello ( )
{
    echo "Hello world"
}
```

→ output

→ hello world.