1. Introduction

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 " " is properly used or not
4. If everything is proper. then shell interprets (converts) that command into kernel understandable from and handover that converted command to kernel.

→ kernel is responsible to execute that command with the help of hardware.

→ shell acts as interface between user and kernel
→ shell + kernel is nothing but operating system.

Shells Types:-

1. Bourne shell
2. BASH shell → Born again Shell. it's GNU Shell.
3. Korn shell
4. Cshell
5. TShell
6. ZShell. and etc...

→ ./filename.sh — to execute the shell script file.

→ # this is a comment

2. VARIABLES

Variables are containers which stores some data inside them

Two types of variables:

1. System variable

These variables are created (or) maintained by the linux (or) unix operating system. These are the predefined variables. which are defined by the OS.

The std convention is, they are defined in capital CASES

2. User defined variable.

There are created and maintaind by user (us).

Defined in lowercase) capital cases.

Note — The variable should Not start with variable.

Define:- name=Hari

echo My name is $name.

3. READ THE INPUT

Here we used "read" to fetch the input

→ read hari ⟶ Store data into hari variable.

read name1 name2 name3 → for multiple input

read -p "username:" user_var

Flag - Allows to enter input in the same line.

read -sp "userpassword :" user_Pass.

↓

User input is silent and, while entering input it doesn't show on terminal.

storing data in a array :-

read -a names.

↓

storing input in "names" array.

to access. → echo $names[0], $names[1] ....

Note :- read ____

↓ empty

In case if user not define any variable by default it store in a "$REPLY" → which is system variable.

4. Pass Arguments to a Bash-script

2nd Argument

↑

Arguments default stored as $1 $2 .....

↓

1st Argument

arguments.

"$@" → the input is stored as array.

"$#" → the hash (#) gives the NOF of arguments in an array.

5 - If statement (if then, if then else, if elif else)

Start.

```
if . [ . $m . condition . $n . ]
then
   echo_statement1

elif [ condition2 ]
then
    statement 2

else
   statement 3

fi
```

end

⊛.

" . " is space while writing clean code.

Integer comparision :-

- -eq - is equal to = if ["$a" -eq "$b"]
- -ne - is not equal to " "
- -gt - is grater than "
- -ge - is grater than or equal to "
- -lt - is less than "
- -le - is less than or equal "

- <
- <=
- >
- >=

} Same as c language.

(("$a" < "$b")).

↓

Double braces when use. <, > ....

String comparision :-

= - is equal to - if ["$a" = "$b"]

== - is equal to - if ["$a" == "$b"]

!= - is not equal to - if ["$a" != "$b"]

< - is less than, in ASCII alphabatical order - if [["$a" < "$b"]]

> - is grata than, in ASCII alphabatical order - if [["$a" > "$b"]]

↑ Double square with <, >, etc.

-z - string is null, that is has zero length.

6. File test operations.

To checks wether file exists (or) not, wether a file is spl, chara spl file etc.

flags

if [-e filename]

this flag is used to check file exits or not. ↑ in "if"

-f → check file is regular or not.

-d → check for the directory.

-b → vid, music, image block spl files

-c → character spl file check.

-s → check wether a file is empty or not

-r, -w, -x → to check read, write, execution permission

7. How to append output to the end of a file

cat >⟶ file will be overwritten.

cat >>⟶ appending data

Etcps: check the file mode (r,w,x) and add the data next.

8. Logical AND operator

AND ⟶ "&&"

if [condition] && [condition2]

then

statement.

(or)

if [[condition1 && condition]]

(or)

if [condition -a condition]

-a flag stands for AND operator

9. Logical_OR_Operator

if [condition]] || [condition2]

if [condition1 -o condition2]

if [[condition1 || condition2]]

10 - perform Arthemetic operation

```
$(( num1 + num2))
$(( num  - num2 ) → *, /, %
```

(or)

```
$(expr $num1 + $num2) → /, -, %
```

Note:- for multiplication " * "

```
$ (expr $num \* $num2)
```

11. Floating math operation- bash | bc command.

we have to use the bc command.

```
echo "20.5 + 5" | bc  → -, *, %
```

giving entire left side of pipe data to bc.

Note:- by " / " for division we want use the. scale

~~echo "20.5/5"~~

```
echo "scale = 20; 20.5/5" | bc = 4.10.....0
```

↓

upto 20 decimals

Square root :-

```
echo "scale = 20; sqrt ($num)" | bc -l
```

12. The case statement

```
case expression in
    pattern1 )
        statements ;;
    pattern 2)
        statements;;
    ...
esac
```

13. Examples on Case Statement

→ For Spl case [A-Z]

→ we have to execute "LANG=C" in cmd.

The LANG environment variable indicates the language/local and encoding wher "C" is the Language setting.

14. Array variables

OS=('ubuntu' 'is' 'a' 'operating system') → Initialization array

OS[0] = 'mac' ——→ Replace the element

OS[6] = 'Hari' ——→ It print 6th element and ignore 4,5 index.

unset OS[2] ——→ Removing the element from index

echo "${os[@]}" ——→ print the all elements

echo "${os[0]}" ——→ Print the 0th index element.

echo "${!os[@]}" ——→ print the index's of elements.

echo "${#os[@]}" ——→ print the count of elements.

15. While-loops

```
while [condition]
do
   command1
   command2
   command3
done
```

Example:-

```
n=1
while (( $n -le 10 ))
do
   echo "$n"
   n=$(( n+1 ))  ( or )  (( n+1 ))
done.
```

Sleep 1

16. Using Sleep in while

Sleep 1 ⟶ pause 1 second

gnome-terminal & ⟶ to generate the new seperate terminal

17. Read a file content in Bash.

Method 1:-

```
while read P
do
  echo $P
done < filename
```

⇒ Data is re directed to P and data saved into P.

- Method 2:- using "|"

```
<filename> | while read P.
             do
               echo "$P"
             done.
```

} Pipe is used to access content in left and give to right side.

Method ③ :- If file in another location.

```
while read p
do
   echo "$ P"
done < file location with filename.
```

③④⑤ → Most used Method.

Method ④ :- using IFS = Internal field seperator and it used by shell to recognized the boundaries

-r → This flag is used to prevent back slash "\" escapes being interpreted.

```
while IFS= read -r P
do
   echo "$P"
done < filename@location with filename.
```

18. UNTIL LOOPS

until [condition] ⟶ It executed untill the condition false.

```
do
   command1
   command2
   ....
done.
```

Example :-

```
n=1
until [ $n -gt 10]
do
   echo "$n"
   n= $((n+1))
done.
```

Output :- 1, 2 . . . 10.

19. For_loop

Method (1):-

```
for variable in 1,2,3 4 5 ... N
do
   command1 on variable.
   command2
done.
```

Method (2):-

```
for output in $(linux-or-unix-cmd-here)
do
   command1 on $output
   command2 on $output
   ...
done.
```

Method (3):-

```
for VARIABLE in file1 file2 file3
do
   command1 on $VARIABLE
   ...
done.
```

Method (4) :- C-type.

```
for((EXP1; EXP2; EXP3))
do
   command 1
   command 2
   command ... N
done.
```

(EXP1 → int, EXP2 → condition, EXP3 → Increment.)

NOte:

{1..10} → Range 1,2,3 .. 10

{1..10..2} → 1,3,5,7

(2 → Difference)

20. For loop executing cmd

```
for cmd in ls. pwd.
do
   "$cmd" → No echo bcz echo print cmds not execute
done.
```

21. Select loop

- Used for menu's. selection.

Syntax:-

```
Select VarName in list

do

    cmd 1
    cmd 2
    . . . .
done.
```

Ex:-

```
select name in mango egg cura.
do
    echo "$name" selected"
done.
```

Output

1) Mango
2) egg
3) cura
#? 1 ← This input will asked

Mango selected.

22. Break and Continue

break — break the loop
continue — skip the loop
} same as "C- language"

23. Functions

Method ① :

```
function namec){
    cmd's
}
```

Method ②

```
namec){
    cmds
}
```

24. Local Variables

→ Every variable is a global variable.

Declearing local variable by using "local" keyword.

we have to use local keyword in functions.

```
Func(){
    local var=23 → It eligible to @ inside function
}
```

25. Function Example (practice urown)

Ternary operators

[Condition] && return 0 || return 1.
statement 1 stmt 2.

26. Readonly command

→ Using the readonly cmd we can only read to any variable

→ we can't overwritten the readonly variable.

→ we can also make readonly functions too.

readonly -f filename.

27. Signals & Traps

Interept Signals:

Ctrl+Z → stop script — suspend signal

ctrl+c

kill. -9 Name of PID and etc.

PID is the can executing script ID.

When some signals (interepts) of kill the code then the trap. is

Trap:- It provides to scripts to capture and Interept and clean it up within script.

→ TO know more about signal use RUN. → $ man 7 signal.

Syntax:

trap "print...." 0 15 ..... etc. signal type to trap(or) capture

Remove trap signals

$ trap - signalsname.

DEBUG

use:- $ bash -x ./filename.

It print all of execution cmds on windows.

Another way:

In a file use: #!/bin/bash -x

(or)

set -x → It activate the debug from that point of line.

set +x → It deactivate the debug from that point of line.