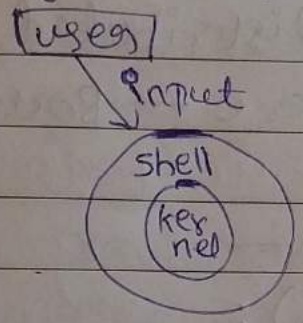
Module - V

SHELL PROGRAMMING.

* A shell is a prgm that provides an interface b/w a user & an os kernel.
* OS starts a shell for each user when the user logs in / opns a terminal / consde.



* A shell is both a cmnd interter (allowing users to interact with the os through cmnds) & a prgming lang.
* Typically opr performed by shell scripts include file manipul", prgm execution & printing txt.
* The cmnd line shell is a txt-based user interface for ~~y8~~ os.
* Shell scripts are interpreted, not compiled. The shell reads cmnds from the scripts per line by line & searches for those cmnds on the system.
* Shells may be used interactively / non-interactively. when executing a non-interactively, shells execute cmnds read

frm a file → Shell Pgm.
* provide built-in cmnds → cd, break, continue, exec, etc.

=> UNIX Shells =
Most widely distributed & influencial of UNIX Shells were — Bourne Shell & C Shells.
Types of Shells —

1) Bourne Shell (sh):
* written by Steve Bourne, is the original UNIX Shell.
* It is the preferred Shell for Shell prgming bcz of its compactness and speed.
* Drawback is that it lacks features for interactive use, such as the ability to recall previous cmnds (history)
* Also lacks built-in arithmtic & logical expression handling.
* Cmnd full-path name is /bin/sh & /sbin/sh
* non-root user default prompt is $
* Root user default promp is #.

2) C Shell (csh) =
* written by Bill Joy at Berkeley.
* Introduced many features such as aliases, cmnd history & C-like expression s/x, path hashing, etc.
* Cmnd full path name → /bin/csh.
Non root user default prompt → hostname %
" " " is hostname #

3) Korn Shell (ksh):
* written by David Korn
* It is a Suprset of Bourne Shell
* Sprt everything in the Bourne Shell & Cmnd line editing.
* provide built in cmnds & is fastr than the C-Shell
* run Scripts written for the Bourne sh.

4) Bash Shell : (Bourne Again shell). (bash)
* It is a default shell ovr several distribu" of linux tdy
* It is a cmnd line intrface shell pgm used in linux & mac os.
* It could be installed ovr windows op.
* create hacking tools & run multiple Cmnds as only one cmnd.
* customizing administrative tasks.

5) Tc Shell (tcsh):

* It is an upgraded c shell. & can be used as a shell script cmnd processor and interactive login shell.
* Has C like Shx, filename completion cmnd-line editor & spelling correction.
* It is a cmnd lang intrpreter usable both as an interactive login shell & a shell script cmnd processor.

✱ check for the shell in use:

* $cat /etc/shells → /bin/sh
  ↓ list available shells in yr systm
  " /ash
  " /bash
  " /dash
  " /mksh
  " /pdksh
  " /posh
  " /tcsh
  " /zsh
* $echo "$SHELL" → to see which shell you are using.
* $echo "$SHELL($0)" → to get crnt shell.
  (or) $echo $0. —dp→ /bin/bash

* $sh → to change from bash shell to sh shell
  $echo $0 —dp→ /bin/bash (sh).

II Shell meta chars

I Navigation cmnts →

1) pwd → (print working directory)
   → shows crnt directory (foldr).
   → /home/user/foo/bar.
2) # → (cmnts)
   → eg → # this is a cmnt
3) cd → (change directory)
   → eg → $pwd → /home/user/foo/bar.
   $ cd ..
   Then, $pwd → /home/user/foo.
4) ls → (list) → list down all the cntnt inside the directory.
   → eg → $pwd → /home/user/foo
   $ ls → apple.txt bar foo some.txt
   2 folders
   → 2 files
5) File execution → $./filename
   [pgm — ctrl+c]
6) mv → to move files
   Shx → MV [option] source destina".

cp → syn → cp [option] source destination.
file → syn → file [option] filename
↳ -b, -p.

→ eg → $ ls → apple.txt bar/foo/some.txt
$ mv apple.txt foo
$ ls → bar/foo/some.txt.
$ ls foo → apple.txt.

7) cp (copy)
→ eg → $ ls → file1 file2.txt file3
$ cp file2.txt file2copy.txt.
all data will copied
$ ls → file1 file2.txt file2copy.txt files

8) cat / more / less / head / tail →
to display the cntnt of a file, use any of the cmnds above.
syn → cat [option] file...
→ eg → $ cat hello.txt
→ hello, I am ansar.

9) mkdir (make directory) → to create a directory
eg → $ ls → apple.txt foo/some.txt.
$ mkdir bar
$ ls → apple.txt bar/foo/some.txt.

10) rmdir (remove dir) → remove each & every dir specified in the cmnd.
[mkdir అంటే create అయిన dir పోతుంది
rmdir అంటే remove అయ్యేది ~ Ab]
syn → rmdir [options] directory...

11) whereis → locates the binary, source & manual pg files for a cmnd
syn → whereis [options] filename...

II Info cmnds =

1) ps → displays info in the form snapshot about cntly active PIDs
2) w → displays info about the users cntly on the machine & their PIDs
3) id → to print the genuine & effective user ID & grp ID
4) free → displays amount of free & used mly in the system
5) clear → to clear the terminal screen
6) Echo → displays whtevr input txt is given to it.

II Meta Chars =

1) # → cmnts.
2) ; → permits putting 2/more cmnds on same line.
eg → echo hello ; echo Hi
3) $ → var substitution
eg → var1 = 10
echo $var1 → 10
4) ' ' → eg → echo 'hi' → hi
5) " " → eg → a=5
echo "value : $a" → value : 5
echo "Hi" → Hi

VI Getting Hlp = (5 ways)

1) **hlp cmnd**: displays info about built-in cmnds.
   Stx → help [-dms] [PATTERN ...]
   eg → help echo.

2) **man cmnd**: display the user manual of any cmnd that we can run on the terminal. used to display manual pgs, scroll up & down, search for occurrence of specific txt, etc.
   Stx → $man [option]... [COMMAND NAME]...
   - eg → $man printf
   - Has nos of options —

a) -h → --help → print a hlp msg & exit
b) -f → --whatis
c) -k → --apropos
d) -w → --where
e) -i → --ignore-case , -I → --match-case

3) **info cmnd**: reads docum" in the info format. Give detailed info for a cmnd when compared with the man pg. The pgs are made using textinfo tools.
   - Stx → info [option]... [Menu-item...]
   - -a → --all → use all matching manuals.
   - -d → --directory=DIR
     eg → info -d CVS

manual → file

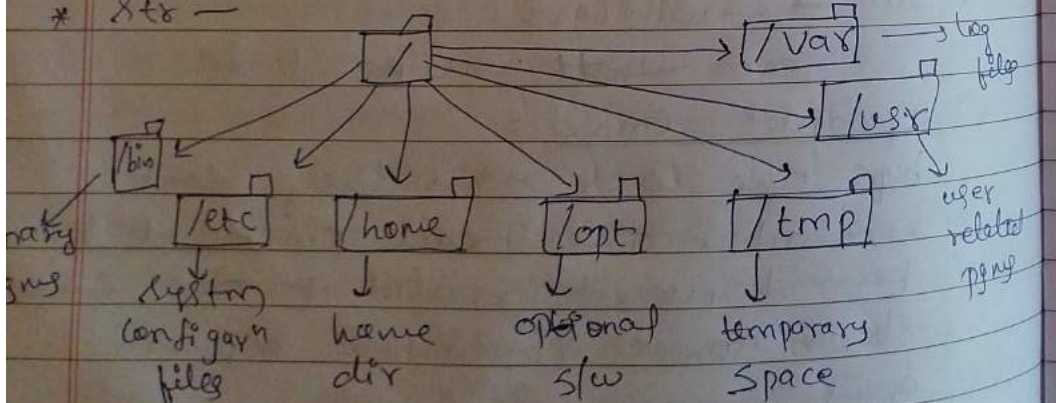
- -f → --file=MANUAL.
- -h → --help
- -n → --node=NODENAME
- -w → --where, --location → print physical loc of info file.

4) **whatis cmnd**:
   used to get a 1-line manual pg descrip".
   In linux, each manual pg has some sort of descrip" within it.
   Stx → whatis [option] [cmnd-name]
   - -d → --debug. → print debugging info
   - -l → --long
   - -h → help
   - -V → --version
   - -w → --wildcard
     eg → ~~whatis~~ whatis -d ls.

5) **apropos cmnd**:
   hlps the user when they don't remember the exact cmnd bt knows a few keywords related to the cmnd
   Stx → apropos [option...] keyword ...
   eg → apropos bind
   - -h → help , -a → --and
   - -V → --version , -w → --wildcard
   - -e → --exact , -d → debug.

V LINUX/UNIX Directory layout :

* A file system is a logical collection of files on a partition.
  A partition is a container for info
* FHS (file system hierarchy standard) defines the directory str & directory contents in LINUX distri~
* (pro) —
  - Has root directory (/) that contains other files & directories.
  - Each file is uniquely identified by its name.
  - It is self contained. There are 2 dependencies b/w one file system & anthr.
* Str —



a) Navigation cmnds.

VI Info cmnds :

1) ps → Str → ps [options]
2) w → Str → w [options] [username1, uname 2 ..]
3) id → Str → id [options] .. [username]
   ↳ -z, -g, -u, -n.
4) free → Str → free [options] → -b, -k, -m, -g
5) clear → Str → clear.
6) echo → Str → echo [options] [Strings]
   ↳ -n, -e, -E
7) more → displays txt, 1 screenat a time.
   It is basically a filter for pging through txt 1 screen at a time.
   Does not provide any options
   Str → more [-options] [-num] [+/pattern] [filename]

VI Piping & Redirection =

* A pipe is a form of redis (transfr of standrd o/p to some othr destn) that is used in linux & othr unix-like os to snd the o/p of 1 cmnd/prgm to anthr cmnd/prgm for further processing.
* You can make it do so by using the pipe char '|'.

* pipe is used to combine 2/more cmnds & the o/p of 1 cmnd acts as i/p to anthr cmnd, & this cmnd's o/p may act as i/p of nxt cmnd & so on.
* Data flow frm L - R through the pipeline
* Syx → cmnd1 | cmnd2 | cmnd3 | ... | cmndN.
* use sort & uniq cmnd to sort a file & print unique values —
  $ sort record.txt | uniq
  (ഒരു file-നെ sort ചെയ്തിട്ട് then print only unique chars/values/names).
* use head & tail to print lines in a particular range in a file —
  $ cat sample.txt | head -7 | tail -5
  This cmnd select first 7 lines through (head -7) cmnd & that will be the i/p to (tail -5) cmnd which will finally print last 5 lines from that 7 lines.
* Redirecting into files —
  (Screen-ൽ display ചെയ്യുന്നതിന് പകരം file-ൽ store ചെയ്യാൻ, for this we use '>.')
  eg → ls > files
* The > operator indicates to the cmnd line that we wish the prgm o/p to be saved in a file instead of printed to the screen.

* Redirecting from files —
  (for this we use '<')
  < operator to snd data the other way.
  eg → wc -l < myoutput.

VII File permissions =

* Every file & dir in yr unix/linux system has 3 permissions —

1) Read :
   This permission give you the authority to opn & read a file.
   Read permission on a dir gives you the ability to lists its contnts.
2) write :
   Gives you the authority to modify the contnts of a file.
   write permission on a dir gives you the authority to add, remove & rename files stored in the dir.
3) Execute: In windows, an executable prg usually has an extension '.exe' & which you can easily run.
   In Linux/unix, you cannot run a pgm unless the execute permission

(per) → permission

is set.
If the execute permission is not set, you might still be able to see/modify the prgm code, bt not run it.

* permissions on unix systm are managed in 3 distinct scopes / clss —
a) user → files & dirs are owned by user. The owner determines the file's usr cls.
b) Grp → files & dirs are assigned a grp which defines the file's grp cls.
c) othr → users who are not the owner, nor a membr of the grp, comprise a file's othr cls.

* To view file (per) —
for all files in a dir — use the 'ls' cmnd with the -la or -l options
eg → ls -l.

* To change file (per) —
use the cmnd chmod (change mode). The owner of a file can change the per for user (u), grp (g) or othrs (o) by adding (+) or subtracting (-) the read w & Execute (per).

There are 2 ways of using chmod to change the (per) —
(a) Symbolic method:
A chmod cmnd using this method consists of at least 3 parts —

Access cls	operator	Access type
u	+ (add access)	r (read)
g	- (remove access)	w (write)
o	= (set exact access)	x (execute)
a (all: u,g & o)		

eg → To add (per) for everyone to read a file in the crnt dir named myfile at unix prompt —
chmod a+r myfile

(b) Absolute method:
(Using octal nos for r, w & x)

(per)	No —
r	4
w	2
x	1

Add the nos of the (per) you want to give
eg → for myfile to grant r, w & x (per) (4+2+1=7), r & x (per) to users in yr grp (4+0+1=5) &

only execute (per) to others (0+0+1=1)
how to use —
chmod 751 myfile
Make the script executable with cmnd
chmod +x <filename>.

VII shell scripting =

1) Cmnts :
* A word / line beginning with "#" causes that word & all remaining chars on that line to be ignored.
* It is ng bt explanatory txt about script & makes source code easier to understand.
* It hlps othr sys admins to understand the code, logic & hlp them to modify the script you wrote.
* eg → # Simple line.
* multiline cmnts — using heredoc.
eg → << multiline cmnt
everything inside the
heredoc is a multiline cmnt.

2) Variables : Char string to which we assign a value. The value be a num, txt, filename / device, etc.
* Shell enables you to create, assign & dlt var.
* Contain letters (A to z or a-z), num (0 to 9) & _ char.
* eg → var_1 = "ansar"
* Syx → var_name = var_value
* to access the value stored in a var, prefix its name with the '$' sign
eg → name = "ansar"
echo $name. → ansar.
* 3 types of var —
a) Local var : var that is prnt within the crnt instance of the shell. They are set at the cmnd prompt.
eg → getNum () {
num=100 # local var
echo "$num - inside ()"
}
echo "$num - outside ()"
getNum.
→ 100 - outside ()
100 - inside ().

2) Global var : Accessible throughout the prgm. Are declared outside any block

of code / ().

eg → num = 200

getNum {

— — —

— — —

}

echo — —

getNum ⟶ 200 o/p ()

100 I/p ().

3) Shell var : var that are set by shell itself & hlp shell to work with O/Ps crctly.

eg → '$PWD' = Stores working dir.

'$HOME' = Stores user's hom dir.

4) Envrmnt var:

Are only created once, after which they can be used by any user.

eg → BASH → Shell name, BASH_VERSION.

3) operators:

1) Arithmetic (o) — used to perform normal arithmetical opr.

a) + → Binary opr used to add 2 operands

b) −

c) *

d) /

e) %

f) ++ (increment) → to inc value of operand by 1

g) −− "

eg →

```
read -p 'Entr a:' a
read -p 'Enter b:' b
add = $((a+b))
echo " + " $add"
sub = $((a-b))
echo " - " $sub"
```

-p → used to display prompt msg.

2) Relational (Comparison) (o):

used to compare 2 var / quantities.

a) -eq ⟶ == (e) -ge → >=

b) -nq ⟶ != (f) -le → <=

c) -gt → >

d) -lt → <

eg →

```
a=10
b=20.
if [$a -eq $b]
then
  echo "equal"
else
  echo "not equal"
fi
```

```
if [$a -lt $b]
then
    echo "lessthan"
else
    echo "not lessthan"
fi
```

3) Logical (Boolean) (o):

a) ! → logical NOT
b) -o or || → logical OR
c) -a or && → logical AND.

eg →

```
echo -n Enter 1st no:
read a                          # a→20
echo -n Enter 2nd no:
read b.                         # b→4
if [ $a -lt 100 -a $b -gt 15 ]
     20 < 100 → T    4 > 15 → F   → F
then
    echo "false return"
else                            → false
    echo "true return"
fi
```

* [! $a -lt 100 -a $b -gt 15]
  ! T -a F
  F -a F → f.

4) Quoting :

a) Single quotes : holds onto the literal value of each char within the quotes. No char in single quote has spl meaning Not intrpret var, backticks, \escapes, $,

b) Double quotes :
Allow the shell to intrpret $, backtick, \, ! .
Have spl meaning when used with double quotes & bfre display they are evaluated.

eg → ① test = 10.
echo "$test" → 10
echo '$test' → $test.

② printf "k\llnk" → k
printf 'k\llnk' → k\llnk.

③ a=10
echo "'$a'" → '10'
echo "'$a" → "$a"

5) read : to get i/p frm keyboard, we use the read cmnd.
takes i/p frm keyboard & assigns it to a var.
eg → echo -n "Entr txt:"
read txt

-n → to keep the cursor on same line.
-p → to display prompt msg
-t, -s, -e

echo "You entered: $txt"

Has several cmnd line options —

a) -p → Saves the extra step of using an echo to prompt the user.
eg → read -p "Enter txt:" txt.
echo "You entered: $txt"

b) -t → followed by a num of sec provides an automatic timeout for read cmnt.
eg → read -t 3 response
(3 sec കഴിഞ്ഞാൽ ഈ read cmnd work ചെയ്യുന്നു, for that we use this)

c) -s → used to make i/p invisible.
The user's i/p will not be displayed on the screen as they type.
eg: → read -s -p "Enter password: pass
echo -e "\n your pass : $pass"
→ enter password: (ഒന്നും type ചെയ്താലും കാണില്ല.

your pass : (ഒന്നും type ചെയ്താൽ കാണും)

-e → used to enable interpret" of backslash escapes
(\n ഉപയോഗിക്കാൻ ആണ് -e. -e ഉപയോഗിച്ചാൽ o/p-ൽ \n വരില്ല.)

2) test : Checks file types & Compare values. used as part of the condi- execution of shell cmnds.
Syx → test EXPRESSION
[EXPRESSION].

④ echo "$?" → 0 means exp evaluated as T
→ 1 " exp " as f.

eg → ① test 100 -gt 99 && echo "yes, it is true" ||
"No, it is false"
→ yes, it is true.
② test 100 -lt 99 && echo "yes" ||
echo "No"
→ No.

5) Condi- Statemnts : (5).

I if (5). → This block will ⓟ if Specified condi- is true.
Syx → if [exp]
then
(8)
fi

II If_else (S): If specified condin is not T in if part then else part will be executed.

Syn → if [exp]
then
S1
else
S2
fi

III if...elif...else...fi (S) (else if ladder):
To use multiple condin in 1 if else block, then elif keyword is used in shell.

Syn → if [exp1] then
S1
elif [exp2] then
S2 ..
S3 ..
else
S4 ..
fi

IV Nested if (S):
can be used when, 1 condin is satisfied then it again checks another condin.

Syn → if [exp1] then
S1 ..
S2 ..
else
if [exp2] then
S3 ..
fi
fi

V Switch (S) / case (S):
works as a switch (S) if specified value match with the pattern then it will execute a block of that particular pattern.
A case will be terminated when the last cmnd is executed.
If there is no match, the exit status of the case is 0.

Syn → case varname in
pattern 1) S1 ;; → act as break (S).
pattern 2) Sn ;;
*)
esac.

*Eg →

```
a=20
b=20
if [$a==$b]
then
    echo "a is =b"
else
    echo "a is !=b"
fi
```

if_else (S)

① cars = ~~bmw~~ "bmw"
pass the variable in string
case "$cars" in
"mercedes") echo "Germany car";;
"audi") echo "Germany cars";;
"bmw") echo "Chennai";;
esac

o/p → Chennai.

② read -p "Enter a number:" n
case $n in
1) echo zero;;
2) echo one;;
⋮
*) echo invalid;;

③ time = $(date +"%H")
↳ hr in 24 hr format.
case $time in
9) echo gd morg;;
12) echo gd noon;;
21) echo gd ngt;;

6) Iterative (8): (3)

I while (5):

cmnd is evaluated & based on the rslt loop will executed, if cmnd raise to F then loop will be terminated.

syn → while cmnd
do
statmnt
done

eg → a=0.
while [$a -lt 10]
do
echo $a
a=$(expr $a+1) (or) ← a='expr $a+1'
done → 0 ⋮ 9

II For (5):

operate on list of items. It repeat a set of cmnds for every item in a list.
Each time the for loop executes, the value of the var is set to the nxt word in the list of words.

syn → for var in word1, word2 ... n
do
stmnt to be executed
done

III) until (5) :

It is executed as many as times the condition evaluates to F. The loop terminates when the condition becomes T.

S/x → until cmnd
do
stmnt.~ until cmnd & T
done.

* eg for loop –

```
for a in 1 2 3 4 5 6
do
    if [ $a==5 ]
    then
        continue
    fi
    echo " iteration no~ $a "
done
```

→ iteration no~ 1
" 2
" 3
" 4
" 6

($a==5 command it comes to continue & it directly goes to for & not goes to echo)

* eg for until loop –

```
a=0
until [ $a -gt 5 ]
do
    echo $a
    a='expr $a+1'
done
```

→ 0 : 5

6 > 5 (loop terminate bcz condi. is T)

* eg for while loop –

```
num=1
while ((num<=5))
do
    echo $num
    ((num++))
done.
```

7) break & continue (5) :

I. break (5) : used to terminate the execution of the entire loop, after completing the execution of all of the lines of code to the break (5).

S/x → break.

use to exit frm a nested loop using –

break n , n → nth closing loop to the exit from.

eg → a=0
while [$a -lt 10]
do
echo $a
if [$a -eq 5]
then
break
fi
a='expr $a+1'
done

5 → first 5 print then goes to if & break.

→ 0 1 2 3 4 5

II Continue (s):

Similar to break (s) except that it calls the cnt iteration of the loop to exit rather than the entire loop.

s/x → continue. (or) continue n → for nested loop.

eg →

Nums="1 2 3"
for num in nums
do
r='expr $num %2'
if [$r -eq 0]
then
echo "evn nom"
continue
fi
echo "odd no~"
done

→ odd no~
evn no~
odd no~

* expr cmnd evaluates a gvn exp & displays its corresponding o/p.
- used for opr like +, -, *, / & % on int & evaluating regular exp, string opr like substring, length of str~
- s/x → $expr expression
- eg → ① for addition → $ expr 3+5
② for subtrac → $ expr 3-5
③ for multipli" → $ expr 5 * 3 = 15
④ for ÷ → " 5/3
⑤ increment → $4=10
$4='expr $4+1'
$ echo $4.
⑥ for find length of str → a=hello
b='expr length $a'
echo $b → 5
⑦ to find indx/position of char in str~

b=$(expr index "hello" "l")
echo $b.
→ 3

a=hello
b='expr index $a l'
echo $b → 3.

(8) to find substr of str —

a=hello

b=`expr substr $a 2 3`

echo $b

→ ell

(↳ 2 position 3 letters.)

8) bc cmnd :

* used for cmnd line calculator.
* Similar to basic calculator by using which we can do basic mathematical cal.
* L/unix provides the bc cmnd & expr cmnd for doing arithmetic cal.
* S/x → bc [-hlwsqv] [long-options] [file ...]

(-hlp, -interactive, -mathlib, -standard, -quiet, -version)

* Sprt features → Ari-(6), inc/dcremnt(6), Assignmnt(6), comparison, logical, math(), Condi-(5), iterative (5)

* eg →

ari — (1) echo "12+5" | bc → 17

(2) echo "10^2" | bc → 100

assign → (3) echo "var=10; var" | bc → 10

inr → (4) echo "var=10; ++var" | bc → 11

dnr → (5) echo "var=10; --var" | bc → 9

compr → (6) echo "10>5" | bc → 1 (T)

9) grep cmnd: [globally search for regular exp & print output]

* searches a file for a particular pattern of chars & displays all lines that contain that pattern.

The ptrn that is searched in the file → regular exp.

* S/x → grep [options] pattern [files]

* options —

1) -c → print only a count of the lines that match a pattern.
2) -h → display matched lines, bt not display filenames
3) -l → display list of filenames only.
4) -v → prints all the lines that do not matches the ptrn.

eg → $ cat > abcfile.txt

→ unix is great os

UNix you choose.

(1) * $grep -i "UNix" abcfile.txt

↳ case insentive.

→ unix is great os

UNix you choose.

* $grep "unix" abcfile.txt

→ unix is great os.

② $grep -c "unix" declfile.txt
→ 2
[2 line-ல 2 unix வரும், So the line no...

10) Arrays :
In shell script,
* It is a var which contains multiple values may be same type / diffrnt type since by default everything is treated as a string in shell script.
* declare array in diffrnt ways—
a) Indirect declaration :
we assigned a value in a particular indx of array var. no need to 1st declare
Syn → Arr_name[indxno] = value.

b) Explicit (d) :
1st we declare array then assigned the value.
Syn → declare -a Arrayname

c) Compound Assignmnt:
we declare array with a bunch of values. we can add othr values later too
Syn → Arr_name = (val1 val2 ... valN).

[Indexing]

* eg → to print array value.
a) to print all elemnts → [@] or [*].
Syn → echo ${Arr_name[*]}.
eg → arr=(abc def ghi rsd)
echo ${arr[@]}
→ abc def ghi rsd.
b) to print 1st elemnt → indx no...
eg → echo ${arr[0]}. (or)
echo $ {arr}
c) to print selected indx elemnt —
Syn → echo ${Arr_name[indxno]}.
eg → echo ${arr[5]}
d) print elemnts in range.
eg → echo ${arr[@]:1:4}.
1st pos to 4th pos
e) to count the length of a particular elemnt
— use # to print length.
eg → echo ${#arr[0]}
(0th pos-ல 2nd elemnt-ன length).
f) to count length of array →
eg → echo ${#arr[@]}.
g) to search in array — arr[@] /* search element */

eg → echo ${arr[@] /*[aA]*/}.
(aA arr-ல் இருந்தால் returns
otherwise returns 0)

b) to dlt arr var — unset.
eg → unset arr_name[i]
(indx 1-ஐ மட்டும் dlt ஆகும்)
unset arr_name
(to dlt array full)

c) to print & tabl array
${#arr[@]} #to find array size
arr=(1 2 3 4 5)
i=0
while [$i -lt ${arr[@]}]
do
echo ${arr[$i]}
i=`expr $i +1`
done → 1 2 3 4 5

d) to read array elemnts at run time
& then print array.
echo -n "Entr total no:"
read n
echo "Enter no:"
declare -a a #declaring array
i=0

while [$i -lt $n]
do
read a[$i]
i=$((i+1))
done
echo "output:"
for i in "${a[@]}"
do
echo $i
done

→ Entr total no: 3
Entr no:
5
10
9
output:
5
10
7