

Bash shell and Scripts.

- Script starts with shebang `#!/`
 - Followed by path to bash or env
`#!/usr/bin/env bash`
`#!/bin/bash`
 - This is executed via the kernel system called `execve()`
 - Kernel checks for `#!/` and passes path to the original program as a command line argument
- So, `./myscript.sh` will have the kernel execute
`/bin/bash ./myscript.sh`

`chmod u+x` → making it executable.

With only read permission, not `x`, you can run it but not directly.

eg `bash myscript.sh` → uses bash to run

If directory containing the script is in your path
You can skip the `./` and run directly eg: `myscript.sh`

Time commands and set variables


time command

eg `time find / -name core`

o/p -
real 3m54.10s → how long in real time
user 0m2.6s → program time
sys 0m5.12s → time by os

Variables

= no space before or after

`myvar="this is some chars; * $"` 

unset → remove a variable

Referencing: `$myvar`

If no value is assigned then it's null.

Your shell stores var in 2 places

env = exported var are copy to new processes u run

So u need to export a var to use it in ur script.

export myvar // declare -x myvar.

export myvar="var2 value"

export -f myfunc → to inherit functions.

export → prints exported variables

```
a=1
{
  a=2
}
echo $a
prints 1
```

```
a=1
{
  a=2
}
echo $a
prints 2
```

when you change a var in fn it changes var in shell.
They're shared

enable → lists bash builtin

compgen -k → lists keywords

Bash startup

• bash-profile → read when bash is invoked as a login shell.

• bashrc → executed when a new shell is started.

PATH=\$PATH:/usr/local/bin

Alias & fn should be normally defined in bashrc.

vi → edit file in terminal

insert → i

Save & quit → esc + w + q

Source & Alias

Source myscript.sh → shell executes as it is its own
OR . ./myscript.sh process and does not create new one

Alias

alias ll="ls -l"

alias copy=cp

alias → lists all aliases

unalias → remove alias

Echo

- n → don't print trailing newline
- e → enable backslash escaped char \n, \t
- E → disable ———— in case they are enabled

ls * → list contents of directories

echo * → shows file & directory names

Redirect

echo "Error" > &2

- 0 - stdin
- 1 - stdout
- 2 - stderr

the typeset and declare commands for variables.

Typeset - sets var as local var

typeset -i x
x must be an integer

1) allows fast operations

2) $x++$, $y = x + 2$, $x = x + 3$,
 x^2 .

declare -l → converts into lowercase

-r - readonly

-u - uppercase

-i integer

-a to make array indexed

-A to make associative array(hash)/dictionary

Read command

- Read a line into var or multiple var

read a b → reads first word into a and rest into b.

Loops

while loop

while

~~while~~ while command list |
 do
 command list
 done

while
 ((x < 10))
 do
 echo loop \$x; date > date.\$x
 ((x++))
 done

(; if u want to do more than one command on same line)

while
 read a b
 do
 echo \$a \$b — reads from file
 done < data-file

ls -l | while
 read -
 ; —→ piping of ls -l in while

For loop

for <var> in <list>
 do
 command list
 done

seq 1 5
 ⇒ 1 2 3 4 5

for num in `seq 1 5` // for d in \$(cat data-file)
 loops over 1, 2, 3, 4, 5

* {A..2}, {1..10} etc.

for j in *.c → any file in current dir ending with .c

Functions

function NAME {
 function body...
 return
 }

call → NAME eg. printhello.

var = \$(printhello) → capture return value.

Exit command -

exit <value> eg exit 0

→ terminates shell process.

→ terminates whole program not just the function

→ default: 0 ⇒ success

Redirection and pipes

0 - stdin

1 - out

2 - error

> - to stdout

< - from stdin

2> → error message redirection

command > stdout-here 2> stderr-here < stdin-from-here

★ command &> file

file gets stdout & stderr from command,
file is created or overwritten

★ command | command2
output → input to cmd2, error goes to screen

★ command 2>&1 | command2
gets stdout & err from command.

★ command >> file ⇒ append to end of file

★ command &>> file
both stdout & stderr to file

sort <<END.

cherry input till reaches END.
apple
END.

File Descriptors -

exec N < myfile.

N - file descriptor to read from myfile

exec N > myfile

to write into myfile

exec A <> myfile → Read, write

exec N >& - or exec N <& -
closes file descriptor N.

ls of → lists open files

ls of -p shell'oid(78121) → files opened by shell with PIDs.

Control flow statements

case statement

case expression in

pattern 1)

command list;;

pattern 2)

command list;;

*); _____ default

esac.

if

command list . #last result is used

then

command list

[else

command list]

fi

Test -

if

test -f afile

or

if [[-f bfile]]

if

test \$x -gt 5

gt, le, eq, ge, lt, ne → numeric comparisons

n → exclusive OR.