```
test      [ ]
        -f file,
        -d (directory)
        -e (file / directory)
        -s (file exists and not empty),
        -r readable,
        -w writable,
        -x executable
This is used for evaluating conditions, like checking strings, number and
files

- spaces inside are mandatory
- [ ] = test
- sh (Shell)
ex. if test -e $filename
        if [ -e $filename ]
====================================================================
[[ ]] - Advanced Test (Bash, Zsh, Ksh)

More powerful
Key points
- Safer for strings
- Supports Regex (Regular Expressions)
- Supports logical Operator ( &&, || )
- It won't work with Sh (Shell)

Compare the String
if [[ $choice == 'y' ]]
then
        echo "YES YES YES"
else
        echo "NO"
fi

==================================================================
( ) -> subshell Execution

Runs a command in subshell (a separate Process), so environment changes don't
affect the current shell.

Key points
 - Useful when you want to isolate environment changes
 - Changes don't persist

ex.
(cd / && ls)
=====================================================================
(( )) -> Arithmetic Evaluations

integer arithmetic and comparisons

Keypoints
- No need for $ on variable names inside (( ));
- Supports c style operators ( +, -, *, / , %, ++, --) etc
- Returns an exit status (0 if success, 1 is failed), so you can use with if
```

```
ex.
increment of x variable by 1 ---->>     ((x++))

=========================================================
{ } - Command Grouping in the same Shell


Group multiple commands in the same shell.

Unlike ( ) which runs a subshell, {} doesn't create a subshell.

changes in variable, directories ,etc persists

- ; is required or newlines between commands.
- space after opening { and before closing }
```