- 2 ways to run:
  - bash script.sh
    - file doesnt need execute permission
    - shebang not required cuz system knows to use bash
  - ./script.sh
    - ensure execute permission if not : chmod +x script.sh
    - shebang required (to tell the shell what program to use)
    - #!/usr/bin/env bash or #!/bin/bash (env to find bash in the system)
      - python instead of bash to use python
- echo
  - echo -e “line1\nline2\tTabbed” (e to enable escape sequences, cuz it usually ignores special characters)
  - echo “something u wanna append to a file” >> file
  - echo -e “\e[32mSuceess: yayyy\e[0m” - Success: yayyy
    - /e escape character
    - [/start ANSI seq
    - 32m green, 34m blue, 31m red
    - \e[0m resets formatting
  - echo word1 word2 - word1 word2
  - echo “word1 word2” - word1 word2
- variables
  - no type casting or declaration, usually string but depending on context permits int operations (like py)
  - name=”nira”
    - no space before or after =
    - var name : alphanum_ , cant start with num
    - to use, prefix it with $ (echo “hello, $name”)
  - echo ‘$name’ - $name
    echo “$name” - nira
    echo $name - nira
  - var=”abc xyz”
    num=”123”
    echo varnum - abc xyz123
    echo “$var” - abc xyz
    echo “$varXX$num” - 123
    echo “${var}XX$num” - abc xyzXX123
    Inside “” tab is tab, but without “” tab is just taken as 1 space.
    Special characters like \,`,$ and “ need escaping by \\, \`. \$ and \” rsptly or they should be in single quotes.

Everything in single quotes is literal, no variable expansion, no command substitution, no special characters.

```
aradhana@NIRA:~$ echo "\\ \` \" \$ \'" '\' '`' '$' '"'
\ ` " $ \' \ ` $ "
```

- command substitution
  lsResult=$(ls)
  dir=`pwd`
  echo “The files are: $lsResult in $dir”
- “” preserves literal value of all except $,`,\
  echo $HOME - /home/aradhana
  echo “$HOME” - /home/aradhana
  ls *.sh - lists all the .sh files
  ls “*.sh” - since its in quotes, * is treated as character, so it looks for a file named “*.sh”.
- env (environment) variables
  (env command shows everything)

```
#!/bin/bash
echo $HOME # Home Directory
echo $PWD # current working directory
echo $BASH # Bash shell name
echo $BASH_VERSION # Bash shell Version
echo $LOGNAME # Name of the Login User
echo $OSTYPE # Type of OS
echo "User $LOGNAME is working in folder $PWD using OS
$OSTYPE"
```

- arithmetic
  - let - built-in command, use when doing inline arithmetic operations that modify variables.

```
let num=5*2+1
echo $num  #11
let "a = 5"
let "a++"
echo $a    #6
```

  - $[ ] - deprecated
    $(()) - use when u need arithmetic expansion and want to directly assign the result.

```
echo $[5*2+1] #avoid -- deprecated 11
echo $((5*2+1))   #11
a=10
```

```
b=2
result=$((a ** b))
echo $result        #100
echo $((a+ b))      #7
echo $((4+5))       #9 [cuz $() is for command substitution]
echo "float: $(echo "scale=2; $a/$b" | bc -l)"  #3.33
```

- Bash does not support floating-point arithmetic natively
  Use bc (basic calculator) command - default does integer arithmetic, -l option for float (-l loads the math library and sets scale=20 by default)

```
echo $((10 % 3))   #1
echo $((3/4))    #0
echo -n "10 "  #no trailing newline
echo "3/4" | bc -l #.75000000000000000000
# or "10 $(echo "3/4" | bc -l)"
echo "scale=5; sqrt(49)" | bc -l    # Output: 7.00000
#Variables persist within a single bc session but are not
retained outside
echo "a=10; b=3; a/b" | bc -l    #3.33333333333333333333
echo "5 > 2" | bc     #1
```

- -i to make a variable integer only, non-numeric value will be ignored.

```
declare -i num
num=$((5/2+1))
echo $num      #3
num="hello"
echo $num      #0
```

- operators:
  - Arithmetic Operators → +, -, *, /, %, **
  - Comparison Operators → -eq, -ne, -gt, -lt, -ge, -le

```
[[ $a -eq $b ]] && echo "a equal b" || echo "a not equal b"
```

  - Logical Operators → &&, ||, !

```
[[ $a -gt 5 && $b -lt 30 ]] && echo "AND"
[[ $a -gt 15 || $b -lt 30 ]] && echo "OR"
[[ ! $a -eq 10 ]] && echo "Negation:a isnt 10"||echo "a is 10"
```

  - String Operators → =, !=, -z, -n

```
str1="hello"
str2="world"
[[ $str1 = $str2 ]] && echo "equal" || echo "not equal"
[[ -z $str3 ]] && echo "String is empty"
[[ -n $str1 ]] && echo "String is not empty"
```

- File Operators → -e, -f, -d, -r, -w, -x

```
[[ -e tempfile.txt ]] && echo "File exists"
[[ -f tempfile.txt ]] && echo "It is a regular file"
[[ -s tempfile.txt ]] && echo "not empty" || echo "empty"
[[ -r tempfile.txt ]] && echo "File is readable"
[[ -w tempfile.txt ]] && echo "File is writable"
```

- Bitwise Operators → &, |, ^, ~, <<, >>

```
a=5  # 0101
echo "NOT a: $((~a))"  # -6 (1010)
echo "Right Shift: $((a >> 1))" # 2 (0010)
echo "Left Shift: $((a << 1))"  # 10 (1010)
```

- Assignment Operators → =, +=, -=, *=, /=, %=

```
x=5
((x+=2)) # No $ when u modify var directly
echo "x += 2: $x"
y=$((x+2))  # $ to return computed value to assign
echo $y    #9
```

- conditionals:

Feature	(( ))	[[ ]]
Purpose	Arithmetic evaluation	String and conditional testing
Usage	Math operations and comparisons	String comparisons, regex
Variables	No `$` needed (e.g., `((x++))`)	`$` needed for variables (e.g., `[[ $x -gt 5 ]]`)
String Comparison	Not supported	Supports string comparison and regex (`==`, `!=`, `=~`)
Logical Operators	Supports only basic arithmetic comparison (e.g., `==`, `!=`, `>` and `<`)	Supports logical operators like `&&` (AND), `
Return value	Numeric result (0 or non-zero)	Boolean (true or false)

```
if (( x > 5 )); then
 echo "x is greater than 5"
fi
-----------------------------------------------------------------
if ls nonexistent_directory &>/dev/null; then
echo "Directory exists!"
else
 echo "Error: Directory does not exist!"
fi
-----------------------------------------------------------------
```

```
if [[ "$name" == "Sunita" ]]; then
 echo "Hello, Sunita!"
elif [[ "$name" == "Arun" ]]; then
 echo "Hello, Arun!"
else
 echo "Hello, Stranger!"
fi
```

- loops:

```
for i in {1..5}; do
 if (( i == 3 )); then
   break #or continue
 fi
 echo "Number: $i"
done
```

for i in apple banana mango, for i in $(ls)

```
#${variable:starting position:no.of letters} → slicing
word="HELLO"
while [[ $word != "" ]]; do
   echo "$word"
   word=${word:1}  # Remove first character
done
-----------------------------------------------------------------
count=1
until (( count > 5 )); do
 echo "Count: $count"
 (( count++ ))
done
```

- shell variables:
  $0: Script name
  $1, $2, … :Arguments passed to the script
  $# Number of arguments
  $@ All arguments as separate words (i.e. a set of strings)
  $* All arguments as a single string (i.e. one string)
  $?: Exit status of the last command (0 = success, nonzero = failure)
  $$: Process ID (PID) of the current shell
  $_: Last argument of the last command (Input & output variables)

```
#Running a command to check exit status
ls "hello.c" 2>/dev/null
echo "ls hello.c exited with $?"
```

```
ls "non-existent-file" 2>/dev/null
echo "ls non-existent-file exited with $?"
echo "Last argument of the last command: $_"
```

```
ls hello.c exited with 0
ls non-existent-file exited with 2
Last argument of the last command: ls non-existent-file exited with 2
```

-