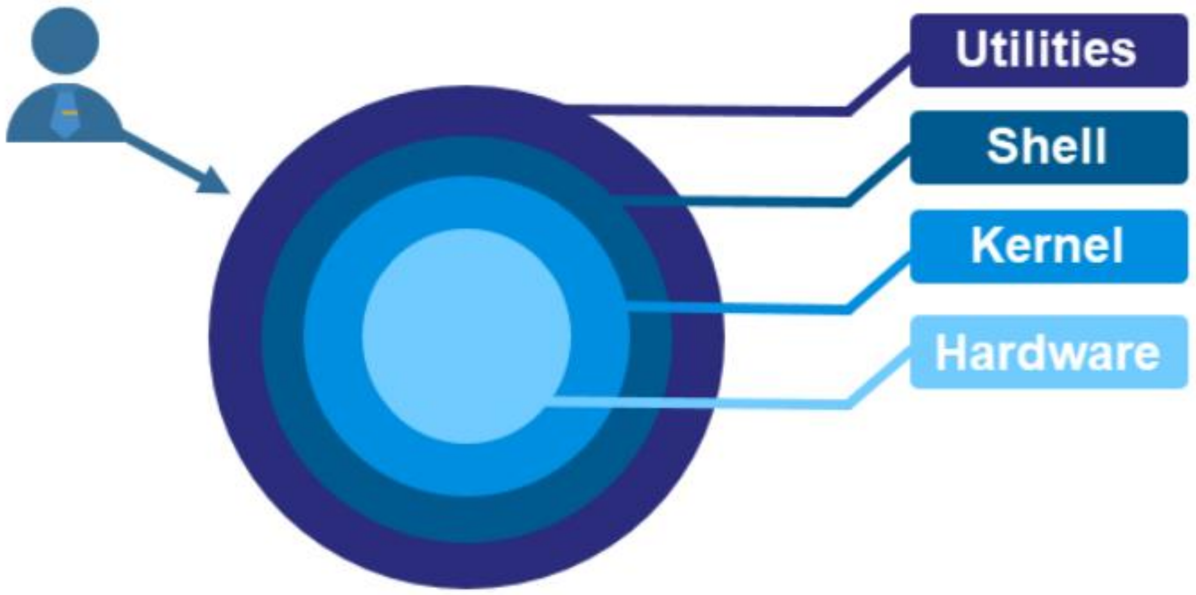
LINUX COMMAND LINE SHELL

- Shell is command-line interpreter program that parses and sends commands to the operating system.
- This program represents and operating system's interactive interface and the kernel's (computer program that manages everything in the system like process, file management) outermost layer (or shell).
- It allows users and programs to send signals and expose an operating system's low-levels utilities (commands like ls, cd).
- The terminal enables in interaction with the system programs like we can create or delete files and folder, run programs or navigate through files and folders and access data and write into files using a typing command.
- When we run any command in the terminal, such as ls or cat, the shell parses, evaluated, searches for, and executes the corresponding program, if found.



- We have n number of Linux CLI Shells including: sh (Bourne Shell), C Shell (csh), bash (Bourne Again Shell), Z Shell (zsh) etc.

BASH:

- Bash is CLI (Command Line Interpreter) for interacting with the computer from the command line.
- Can be used to write shell scripts.
- Location is /bin/bash.
- Bash prompt:
  - $ for regular user.
  - # for root user.
- which $SHELL (output:: /bin/bash)
  Here, which command tells the path of the shell.

BASH SCRIPTING:

- ➔ Write programs or commands that can be run on the system from the command line.
- ➔ Extension: .sh
- ➔ Bash script first line is:
  #! /bin/bash (shebang)

  Shebang: instruct the OS to use the bash as a command interpreter and instructs that shell script is written.
  - ➔ #! /bin/sh → Bourne Shell
  - ➔ #! /bin/csh -> C Shell

- ➔ To execute the script: ./scriptname
- ➔ ls -l (-rwxrwxrwx → owner, member, user) –(filename) l(link) d(dir)
- ➔ chmod 755 scriptname or chmod +x scriptname (give execute permission to the script)
- ➔ commands are case-sensitive.

VARIABLES

- ➔ No data type given.
- ➔ Syntax: variablename = value
- ➔ To access the variable, $a.
- ➔ By default, variables are global.
- ➔ To make local variable, use: local variablename = value.
- ➔ ./scriptname arg1 arg2 arg3 (Positional arguments)
  Here, ./scriptname ($0), arg1 ($1), arg2 ($2), arg3 ($3)
- ➔ '' single quotes: consider $a as literal.
- ➔ "" double quotes: consider $a as variable.

ECHO AND READ:
- ➔ echo command display data on screen.
  echo "Hello World!"
- ➔ read command used to read user input.
  read variablename
  read -p "PROMPT MSG" variablename

ARITHMETIC OPERATIONS:

((expression)) → with or without $
Operations: [+, -, *, /, **, %, +=, -=, *=, /=, %=]

1) var=$((10 + 3))   Wrong:: var = $((10 + 3))
2) ((var = 10 + 3)) or ((var=10 + 3))
3) a=10
   b=20
   ((var = a + b)) or ((var=a + b))
4) var=$((a + b)) Wrong:: var = $((a + b))

echo $var

STATEMENTS:

1) IF STATEMENT:

```
if [[ expression ]]; then
        statements
fi

if [[ expression ]]; then statements; fi
```

Example:

➔
```
if [[ 10 -gt 3 ]]; then
        echo “Greater than”
fi
```

➔
```
if [[ $a == “abc” ]]; then
        echo “Equal to”
fi
```

2) IF-ELSE STATEMENT:

```
if [[ expression ]]]; then
        statements
else
        statements
fi

if [[ expression ]]; then statements; else statements; fi
```

Example:

➔
```
if [[ 10 -gt 3 ]]; then
        echo “Greater than”
else
        echo “Less than”
fi
```

➔
```
if [[ $a == “abc” ]]; then
        echo “Equal to”
else
        echo “Not Equal to”
fi
```

3) IF-ELIF-ELSE STATEMENT:

```
if [[ expression ]]; then
        statements
elif [[ expression ]]; then
        statements
else
        statements
fi
if [[ expression ]]; then statements; elif [[ expression ]]; then statements; else statements; fi
```

Example:

➔
```
if [[ 10 -gt 3 ]]; then
        echo “Greater than”
elif [[ 10 -lt 3 ]]; then
        echo “Less than”
else
        echo “Conditions False”
fi
```

➔
```
if [[ $a == “abc” ]]; then
        echo “Equal to”
elif [[ $a != “abc” ]]; then
        echo “Not Equal to”
else
        echo “Conditions False”
fi
```

Expressions:

- ➔ [[expression1]] && [[expression2]]
- ➔ [[expression1]] | | [[expression2]]
- ➔ -gt: greater than
- ➔ -lt: less than
- ➔ -eq: equal to

Store function value in variable: hello() {} ➔ variablename=”$(hello)”

<u>LOOPS AND CONDITIONS</u>:

1) WHILE LOOP:
   - ➔ Execute commands if condition is true.
   - ➔
```
while [[ expression ]];
do
    statements
    increment / decrement
done

while [[ expression ]]; do statements; increment/decrement; done
```

   - ➔ **Example:**

```
n1=10
n2=15
while [[ n1 -lt n2 ]];
do
    echo “$n1”
    (( n1++ ))
done

while [[ n1 -lt n2 ]]; do echo "$n1"; (( n1++ )); done
```

```
i=10
while [[ i -le 10 ]];
do
    echo “$i”
    let i++
done

while [[ i -le 15 ]]; do echo "$i"; let i++; done
```

2) FOR LOOP:

➔ Perform repetitive tasks / iterate over set of statements.

➔ for variable in list:
```
do
    statements
done
```
➔ for ((exp1; exp2; exp3))
```
do
    statements
done
```
➔ **Example:**

```
list=”Hello World, This is for loop”
for i in $list
do
    echo $i
done
#Output: In different lines.

for i in “$list”
do
    echo $i
done
#Output: In same line.
```

RANGE:

{start..end..increment/decrement}

```
for i in {1..10}
do
    echo $i
done

for (( i=1; i <= 10; i++ ))
do
    echo $i
done
```

FUNCTIONS:

- ➔ Reusable code.
- ➔ Set of commands that can be called several times.
- ➔ Functions can return values.
- ➔ functionname() {

```
	statements
}
```

- ➔ function functionname() {

```
	statements
}
```

- ➔ Function call:
  functionname [arguments positional]
- ➔ **Example:**

```
first_function () {
	echo “Hello”
}
first_function
```

wc:

- ➔ wc is word count. Also, tells the characters and lines.
- ➔ By default, gives word count, characters and lines.
- ➔ wc -options filename [output: lines, words, characters]
- ➔ **Example:**
  wc -l filename -> Gives number of lines.
  wc -m filename -> Gives number of characters.
  wc -w filename -> Gives number of words.

sed:

- ➔ Stream editor.
- ➔ Used to perform operations like find and replace, insertion or deletion, searching.
- ➔ sed options filename/scriptname
- ➔ To permanently update the changes, use -i option.
- ➔ **Example:**

```
sed ‘s/word_to_search/word_to_replace_with’ filename   -> replace first occurrence in each line.
sed ‘s/word_to_search/word_to_replace_with/g’ filename   -> replace all occurrence in each line.
sed ‘$d’ filename   -> delete last line.
sed ‘2d’ filename  ->  delete second line.
```

awk:

- ➔ Manipulating data.
- ➔ Works with data organized into records with fields.
- ➔ Filter the data and display it.
- ➔ awk options filename
- ➔ **Example:**

```
awk ‘{print}’ filename    ->  prints all column of the data.
awk ‘{print $1}’ filename    ->  prints first column of the data.
awk ‘{print $1, $3}’ filename
```

<u>Piping</u>:

- ➔ Output of one command is input to another command.
- ➔ Symbol: |
- ➔ **Example:**

```
cat abc.txt | wc -l
```

<u>Redirection to the file</u>:

- ➔ Used to save the output to a file instead to printing it on the screen.
- ➔ '>' operator: overwrite the existing data.
- ➔ '>>' operator: append the data.
- ➔ **Example:**

```
ls > output
ls >> output
```