TCS332 Fundamental of Information Security and Blockchain

B. Tech CSE III Semester

Instructor:

Dr Mohammad Wazid

Professor, Department of CSE

Graphic Era (Deemed to be University), Dehradun, India

Email: wazidkec2005@gmail.com

Homepage: https://sites.google.com/site/mwazidiiith/home

Unit-II
Basics of Bash or Shell Scripting

Basics of Bash or Shell Scripting

- An Operating is made of many components, but its two prime components are -

Kernel and Shell

- A Kernel makes the communication between the hardware and software possible.
- While the Kernel is the innermost part of an operating system, a shell is the outermost one.

Basics of Bash or Shell Scripting

- A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output.
- It is the interface through which a user works on the programs, commands, and scripts.
- A shell is accessed by a terminal which runs it.
- When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when hit the Enter key.
- Output is also displayed on the terminal.

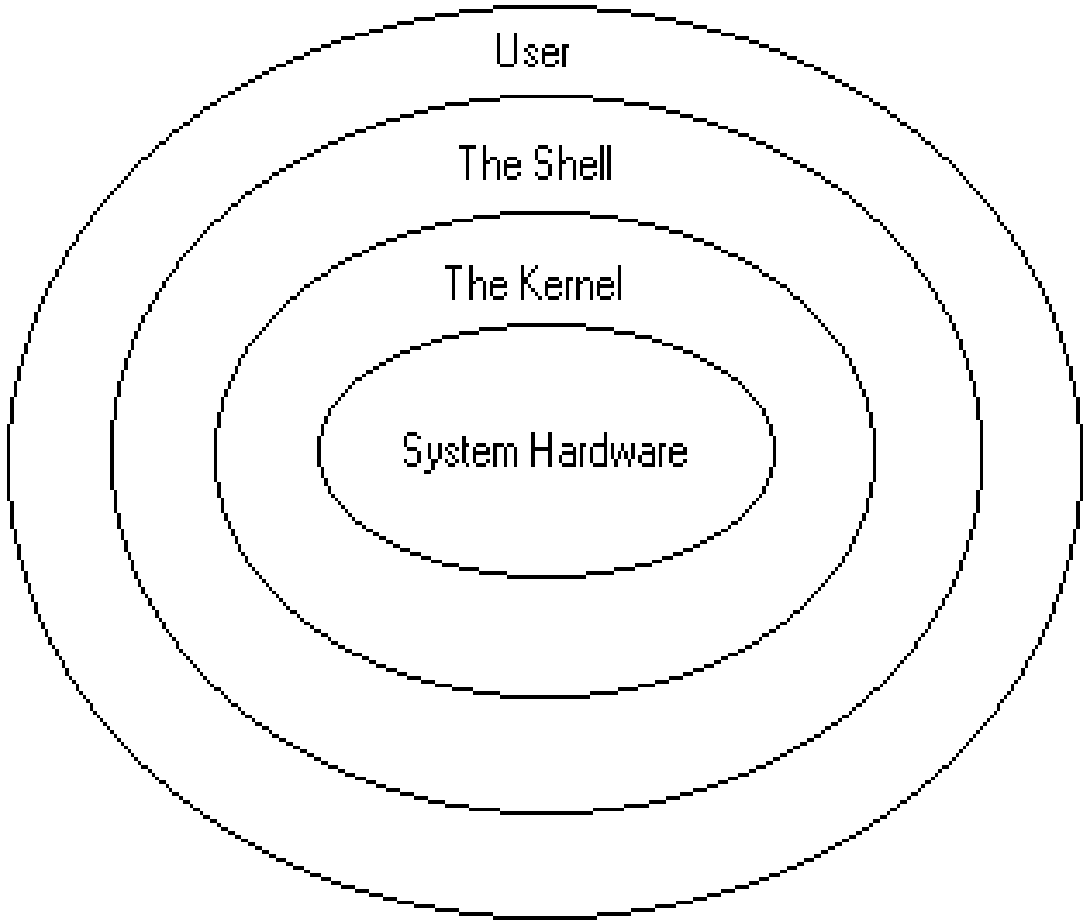

Fig. Operating system environment

Types of Shell

- There are two main shells in Linux:

1. Bourne Shell: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also known as sh
- Korn Shell also known as ksh
- **B**ourne **A**gain **SH**ell also known as bash (most popular).

Types of Shell

2. C shell: The prompt for this shell is %, and its subcategories are:

- C shell also known as csh
- Tops C shell also known as tcsh

Shell Scripting

- Shell scripting is writing a series of command for the shell to execute.
- It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime.
- This reduces the effort required by the end user.

Shell Scripting

- Let us understand the steps in creating a Shell Script
- **Create a file using** a gedit editor (or any other editor). Name script file with **extension (.sh)**
- **Start** the script with **#! /bin/sh**
- Write some code.
- Save the script file as filename.sh
- For **executing** the script type **bash filename.sh**

Shell Scripting

- "#!" is an operator called shebang which directs the script to the interpreter location.
- So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.
- Let's create a small script –

==========

#!/bin/sh

ls

==========

- **How to make it executable**
- mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ chmod 755 fshell.sh
- Running the executable
- mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ ./fshell.sh

Use of loops

For Loops

- The for loop is a looping statement that uses the keyword to declare a repetitive statement.
- The bash supports different syntaxes for the **for loop statement.**
- Syntax 1: For loop

```
for <varName> in <list>
do
    #### your statement here
done
```

Use of loops

- **Example:**

```
#!/bin/sh
for i in 1 2 3 4 5
do
    echo "Looping ... number $i"
done
```

Output:

```
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash forloop.sh
Looping ... number 1
Looping ... number 2
Looping ... number 3
Looping ... number 4
Looping ... number 5
```

While Loop

- The while statement is a type of repetitive structure in bash that utilizes the keyword while.
- Unlike the C-type syntax of the for looping structure, the while repetitive control structure separates the initialization, Boolean test and the increment/decrement statement.
- **Syntax:**

<initialization>

while(condition)

do

    ###your code goes here

<increment/decrement>

done

While Loop

Example:

```
#!/bin/sh
a=0
# -lt is less than operator
#Iterate the loop until a less than 10
while [ $a -lt 10 ]
do
# Print the values
    echo $a
# increment the value
    a=$(( $a + 1 ))
done
```

While Loop

Output:

```
 mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash whileloop.sh
0
1
2
3
4
5
6
7
8
9
```

Use of mathematical operators

-ge

- greater than equal to

-gt

- greater than

-lt

- less than

-le

- less than equal to

-eq

- equal to

until Loop

- The until loop is used to execute a given set of commands as long as the given condition evaluates to false.
- The Bash until loop takes the following form:

```
until [CONDITION]
do
   ###your code goes here
done
```

until Loop

Example:

```
#!/bin/bash
counter=0
until [ $counter -gt 5 ]
do
   echo Counter: $counter
((counter++))
done
```

until Loop

Output:

```
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash untillope.sh
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
```

if statement

- if statement is used to check certain conditions.

Syntax:

if <condition>; then

###your code goes here

fi

if statement

Example:

```
#!/bin/sh
#Initializing two variables
a=10
b=20
#Check whether they are equal
if [ $a == $b ]
then
   echo "a is equal to b"
fi
```

```
#Check whether they are not equal
if [ $a != $b ]
   then
    echo "a is not equal to b"
 fi
```

if statement

Output:

```
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$   bash if.sh
a is not equal to b
```

If-else statement

- If specified condition is not true in if part then else part will be executed.

Syntax

```
if [ expression ]
then
   statement1
else
   statement2
fi
```

If-else statement

- **Example:**

```
 #!/bin/sh
#Initializing two variables
a=20
b=20
if [ $a == $b ]
then
  # If they are equal then print
    echo "a is equal to b"
else
     #else print this
   echo "a is not equal to b"
fi
```

If-else statement

Output:

mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash ifelse.sh

a is equal to b

if..elif..else..fi statement (Else If ladder)

- To use multiple conditions in one if-else block.
- If expression1 is true then it executes statement 1 and 2, and this process continues.
- If none of the condition is true then it processes else part.
- Same thing that we do C language.

if..elif..else..fi statement (Else If ladder)

Syntax

```
if [ expression1 ]
then
        statement1
        statement2
elif [ expression2 ]
then
        statement3
        statement4
else
        statement5
fi
```

if..elif..else..fi statement (Else If ladder): Example

```
#!/bin/bash
read -p "Enter value of i :" i
if [ $i -eq 10 ]
then
   echo "Value of i is 10"
elif [ $i -eq 20 ]
  then
    echo "Value of i is 20"
else
   echo "Value of i is not equal to 10 or 20"
fi
```

if..elif..else..fi statement (Else If ladder): Example

Output:

```
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash if-elseif-else.sh
Enter value of i :10
Value of i is 10
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash if-elseif-else.sh
Enter value of i :20
Value of i is 20
mwazid@mwazid:~/geu-ddn/fundamenals-cyber-sec$ bash if-elseif-else.sh
Enter value of i :30
Value of i is not equal to 10 or 20
```

Home work:

- Q1: Write down a shell script to copy files from one location to another location.
- Q2: Write down a shell script to compute the factorial of a number.
- Q3: Write down a shell script to calculate a table of a given number.
- Q4: Write down a shell script to print following pattern.

References

- 1. M. Ebrahim et al., “Mastering linux shell scripting”, Packt Publishing Limited
- 2. Linux for Beginners Book by Jason Cannon