

## Operating Systems (Lab 3 )

### Contents and topics to be covered:

#### ➤ *Shell Script Programming*

1. Commands Redirection
2. Pipelines
3. Variables
4. Read Statement
5. Controls and Loops
  - If..Then..Else
  - Multilevel If..Then..Else
  - Case
  - Do..While
  - For

# Shell Scripting

## Commands Redirection

Mostly all commands give output on screen or take input from keyboard, but in Linux (and in other OSs also) it's possible to send output to file or to read input from file.

- There are three main redirection symbols >,>>,<

1) > Redirector Symbol

*Syntax:*

**Linux-command > filename**

To output Linux-commands result (output of command or shell script) to file.

To exit Ctrl+C or trl+D

**Note** that if file already exist, it will be overwritten else new file is created.

For e.g. To send output of ls command give

**\$ ls > myfiles**

**\$ cat > myfiles**

Now if 'myfiles' file exist in your current directory it will be overwritten without any type of warning.

2) >> Redirector Symbol

*Syntax:*

**Linux-command >> filename**

To output Linux-commands result (output of command or shell script) to END of file.

**Note** that if file exist, it will be opened and new information/data will be written to END of file, without losing previous information/data, and if file is not exist, then new file is created. For e.g. To send output of date command to already exist file give command

**\$ date >> myfiles**

3) < Redirector Symbol

*Syntax:*

**Linux-command < filename**

To take input to Linux-command from file instead of key-board. For e.g. to take input for cat command give

```
$ cat < myfile
```

## Pipelines

"A pipe is nothing but a temporary storage place where the output of one command is stored and then passed as the input for second command. Pipes are used to run more than two commands (Multiple commands) from same command line."

*Syntax:*

```
commandA | commandB    Pipe the output from commandA into commandB
```

Examples:

```
$ ls | more
```

```
sort names | head
```

## Pipelines

- You can use a pipeline (symbolized by "|") to make the output of one command serve as the input to another command. This idea can be used to create a combination of commands to accomplish something no single command can do.

Enter this command:

```
$ echo "cherry apple peach"
cherry apple peach
```

Okay, let's say we want to sort these words alphabetically. There is a command "sort", but it sorts entire lines, not words, so we need to break this single line into individual lines, one line per word.

Step one: pipe the output of "echo" into a translation (tr) command that will replace spaces with linefeeds (represented by "\n"):

```
$ echo "cherry apple peach" | tr " " "\n"
cherry
apple
peach
```

Success: each word appears on a separate line. Now we are ready to sort.

Step two: add the sort command:

```
$ echo "cherry apple peach" | tr " " "\n" | sort
apple
cherry
peach
```

- Let's try reversing the order of the sort:

```
$ echo "cherry apple peach" | tr " " "\n" | sort -r
peach
cherry
apple
```

## Variables

### - In Linux (Shell), there are two types of variable:

- 1) **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.

Example: HOME, USER, PATH, SHELL, PWD, OSTYPE...etc

- 2) **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

To define UDV use following syntax

*Syntax:*

**variable name=value**

'value' is assigned to given 'variable name' and Value must be on right side = sign.

*Example:*

```
$ no=10 # this is ok
```

```
$ 10=no # Error, NOT Ok, Value must be on right side of =sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

### - Rules for naming variables ( Both UDV and System Variables)

- 1) Variable name must begin with Alphanumeric character or underscore character (\_), followed by one or more Alphanumeric character.
- 2) Don't put spaces on either side of the equal sign when assigning value to variable.
- 3) Variables are case-sensitive, just like filename in Linux.
- 4) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

```
$ vech= OR $ vech=""
```

### - Print or access value of variables ( Both UDV and System Variables)

*Syntax:*

```
$variablename=ali
```

Use 'echo' command to print the content of variables as follows:

```
$ echo $variablename
```

## Read Statement

Use to get input (data from user) from keyboard and store (data) to variable.

*Syntax:*

```
read variablename
```

Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname.

```
#!/bin/bash  
echo "Your first name please:"  
read fname  
echo "Hello $fname"
```

## Controls and Loops

### *Mathematical Operators in Shell Script*

<b>-eq</b>	is equal to	<b>==</b>
<b>-ne</b>	is not equal to	<b>!=</b>
<b>-lt</b>	is less than	<b>&lt;</b>
<b>-le</b>	is less than or equal to	<b>&lt;=</b>
<b>-gt</b>	is greater than	<b>&gt;</b>
<b>-ge</b>	is greater than or equal to	<b>&gt;=</b>

[ expr ] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

*Syntax:*

```
[ expression ]
```

*Example:*

```
if [ 5 -eq 6 ]
```

## ***If..Then..Else***

If given condition is true then command1 is executed otherwise command2 is executed.

*Syntax:*

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to else statement
else
    if condition is not true then
    execute all commands up to fi
fi
```

*Example:*

```
#!/bin/bash
echo "Please enter a number:"
read n
if [ $n -ge 0 ]
then
    echo "Number is positive"
else
    echo "Number is negative"
fi
```

## ***Multilevel If..Then..Else***

*Syntax:*

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to elif statement
elif condition1
then
    condition1 is zero (true - 0)
    execute all commands up to elif statement
else
    None of the above condtion,condtion1,condtion2 are true (i.e.
    all of the above nonzero or false)
    execute all commands up to fi
```

**fi**

*Example:*

```
#!/bin/bash
echo "Please enter a number:"
read n
if [ $n -gt 0 ]
then
    echo "Number is positive"
elif [ $n -lt 0 ]
then
    echo "Number is negative"

else
    echo "Number is zero"
fi
```

## ***Case***

The case statement is good alternative to Multilevel if-then-else-fi statement. It enables you to match several values against one variable. It is easier to read and write.

*Syntax:*

```
case $variable-name in
pattern1) command
    ..
    command;;
pattern2) command
    ..
    command;;
patternN) command
    ..
    command;;
*) command
    ..
    command;;
esac
```

The *\$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is \*) and it is executed if no match is found.

*Example:*

```
#!/bin/bash
echo "Please enter a number:"
read n
case $n in
    1) echo "First Choice" ;;
    2) echo "Second Choice" ;;
    3) echo "Third Choice" ;;
    *) echo "Other Choice" ;;
esac
```

## ***Do..While***

*Syntax:*

```
while [ condition ]
do
    command1
    .....
done
```

*Example:*

```
#!/bin/bash
i=1
while [ $i -le 5 ]
do
    echo "Hello $i times"
    i=`expr $i + 1`          Or    i=$((expr $i + 1))
done
```

## ***For***

*Syntax:*

```
for { variable name } in { list }
do
    execute one for each item in the list until the list is not finished (And
    repeat all statement between do and done)

done
```

*OR:*

```
for (( expr1; expr2; expr3 ))
```



```
do
    repeat all statements between do and done until expr2 is TRUE
done
```

*Example:*

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

OR

```
#!/bin/bash
intlist="1 2 3 4 5"
for i in $intlist
do
    echo "Welcome $i times"
done
```

OR

```
#!/bin/bash
for (( i = 0 ; i <= 5 ; i++ ))
do
    echo "Welcome $i times"
done
```

**Exercises:**

- 1) Write a program that displays the squares of the numbers from 0 to 14**
- 2) Calculate the Product of two numbers WITHOUT using the \* operator**
- 3) Write a shell script that Check the number if odd or even.**

////////////////////////////////////

```

guest-oNIa7Q@ubuntu:~$ cat > dowhile
#!/bin/bash
i=1
while [ $i -le 5 ]
do
echo "hello $i times"
i=$((expr $i + 1))
done
guest-oNIa7Q@ubuntu:~$ chmod 755 dowhile
guest-oNIa7Q@ubuntu:~$ ./dowhile
hello 1 times
hello 2 times
hello 3 times
hello 4 times
hello 5 times
guest-oNIa7Q@ubuntu:~$ echo `expr 5 + 6`
11
guest-oNIa7Q@ubuntu:~$ cat > forf1
#!/bin/bash
for i in 1 2 3 4 5
do
echo "welcome $i times"
done
guest-oNIa7Q@ubuntu:~$ chmod 755 forf1
guest-oNIa7Q@ubuntu:~$ ./forf1
welcome 1 times
welcome 2 times
welcome 3 times
welcome 4 times
welcome 5 times

```

```

guest-oNIa7Q@ubuntu:~$ cat > forf2
#!/bin/bash
intlist="1 2 3 4 5"
for i in $intlist
do
echo "welcome $i times"
done
guest-oNIa7Q@ubuntu:~$ chmod 755 forf2
guest-oNIa7Q@ubuntu:~$ ./forf2
welcome 1 times
welcome 2 times
welcome 3 times
welcome 4 times
welcome 5 times
guest-oNIa7Q@ubuntu:~$ cat > forf3
#!/bin/bash
for (( i=0; i<=5; i++))
do
echo "welcome $i times"
done
guest-oNIa7Q@ubuntu:~$ chmod 755 forf3
guest-oNIa7Q@ubuntu:~$ ./forf3
welcome 0 times
welcome 1 times
welcome 2 times
welcome 3 times
welcome 4 times
welcome 5 times
guest-oNIa7Q@ubuntu:~$ █

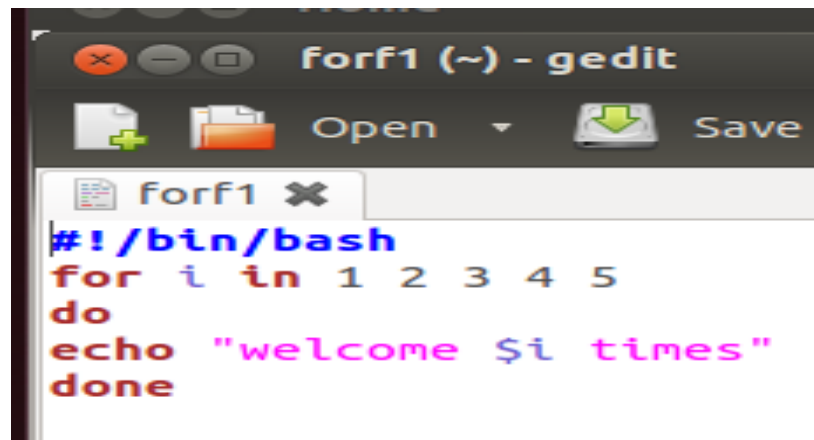
```



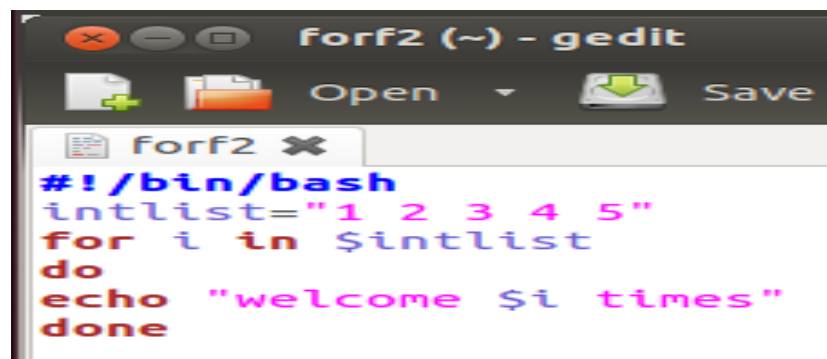
```

#!/bin/bash
i=1
while [ $i -le 5 ]
do
echo "hello $i times"
i=$(expr $i + 1)
done

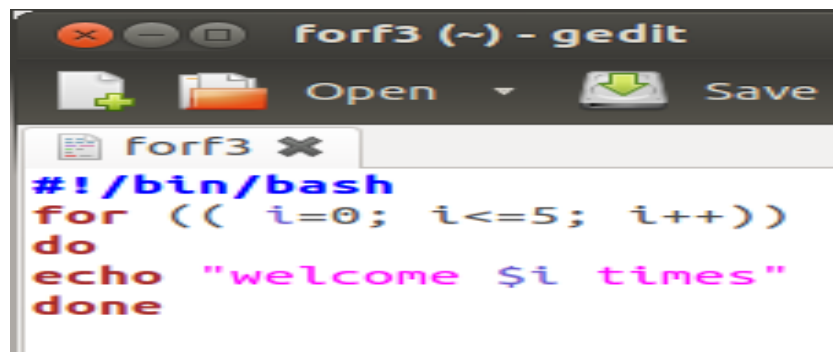
```



```
#!/bin/bash
for i in 1 2 3 4 5
do
echo "welcome $i times"
done
```



```
#!/bin/bash
intlist="1 2 3 4 5"
for i in $intlist
do
echo "welcome $i times"
done
```



```
#!/bin/bash
for (( i=0; i<=5; i++))
do
echo "welcome $i times"
done
```

```
oper ✕
#!/bin/bash
echo "enter number1:"
read num1
echo "enter number2:"
read num2
echo "enter the operation:"
read op
case $op in
'+') echo "$($expr $num1 + $num2)";;
-') echo "$($expr $num1 - $num2)";;
*) echo "$($expr $num1 \* $num2)";;
/') echo "$($expr $num1 / $num2)";;
*) echo "error";;
esac
```