جامعة القاهرة الجديدة التكنولوجية

Fall 2023

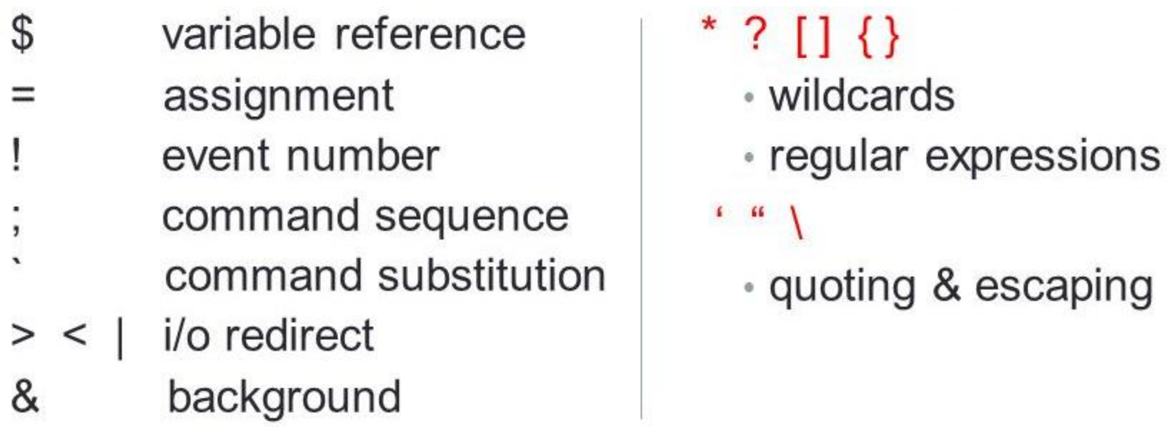
(6)

Linux Essentials

Dr. Hatem Yousry

Agenda

- **Shell Variables.**
- **Shell Scripts.**
- **Basics of writing shell script.**
- **Looping and conditions Statements.**
- **Shell Special Characters.**



$	variable reference
=	assignment
!	event number
;	command sequence
`	command substitution
> < |	i/o redirect
&	background

`*` ? [] {}
- wildcards
- regular expressions

' " \
- quoting & escaping

Shell Prompt Example

- Code segments and script output will be displayed as monospaced text. Command-line entries will be preceded by the **Dollar sign ($).** If your prompt is different, enter the command:
- **PS1="$ " ; export PS1**
- Then your interactions should match the examples given (such as ./my-script.sh below).
- Script output (such as **"Hello World"** below) is displayed at the start of the line.
- **$ echo '#!/bin/sh' > my-script.sh**
- **$ echo 'echo Hello World' >> my-script.sh**
- **$ chmod 755 my-script.sh**
- **$ ./my-script.sh**
- **Hello World**
- **$**

How to write shell script

- To write shell script you can use in of the Linux's **text editor such as vi , nano or mcedit** or even you can use **cat command.** Here we are using cat command you can use any of the above text editor.
- Now we write our first script that will print "Knowledge is Power" on screen.
- First type following cat command and rest of text as its
- **$ cat > first**
- **#**
- **# My first shell script**
- **#**
- **clear**
- **echo "Knowledge is Power“**
- **Press Ctrl + D to save. Now our script is ready. To execute it type command**
- **$ ./first**
- **This will give error since we have not set Execute permission for our script first; to do this type command**
- **$ chmod +x first**
- **$ ./first**

First screen will be clear, then Knowledge is Power is printed on screen.

How to Run Shell Scripts

- Because of security of files, in Linux, the creator of Shell Script does not get execution permission by default. So if we wish to run shell script we have to do two things as follows
- **(1) Use chmod command as follows to give execution permission to our script**
- **Syntax: chmod +x shell-script-name**
- **OR Syntax: chmod 777 shell-script-name**
- **(2) Run our script as**
- **Syntax: ./your-shell-program-name**
- **For e.g.**
- **$ ./first**
- Here **'.'(dot)** is command, and used in conjunction with shell script. The dot(.) indicates to current shell that the command following the dot(.) has to be executed in the same shell i.e. without the loading of another shell in memory.

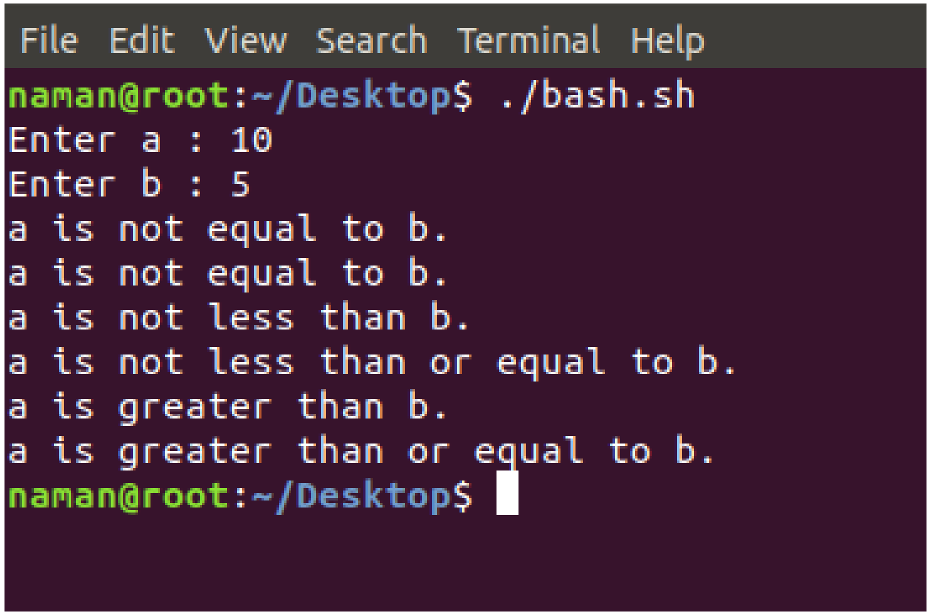
if-then-fi for decision making

- **Syntax:**
- ***if condition***
- ***then***
- ***command1 if condition is true or if exit status***
- ***of condition is 0 (zero)***
- **…**
- **…**
- ***Fi***
- Here condition is nothing but comparison between two values, for compression we can use **test or [expr] statements** or even **exist status** can be also used. An expression is nothing but combination of values, **relational operator (such as >,<, <> etc)** and **mathematical operators (such as +, -, / etc).**

```
#!/bin/bash

#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b

if(( $a==$b ))
then
        echo a is equal to b.
else
        echo a is not equal to b.
fi
if(( $a!=$b ))
then
        echo a is not equal to b.
else
        echo a is equal to b.
fi
if(( $a<$b ))
then
        echo a is less than b.
else
        echo a is not less than b.
fi
if(( $a<=$b ))
then
        echo a is less than or equal to b.
else
        echo a is not less than or equal to b.
fi
if(( $a>$b ))
then
        echo a is greater than b.
else
        echo a is not greater than b.
fi
if(( $a>=$b ))
then
        echo a is greater than or equal to b.
else
        echo a is not greater than or equal to b.
fi
```



```
naman@root:~/Desktop$ ./bash.sh
Enter a : 10
Enter b : 5
a is not equal to b.
a is not equal to b.
a is not less than b.
a is not less than or equal to b.
a is greater than b.
a is greater than or equal to b.
naman@root:~/Desktop$
```

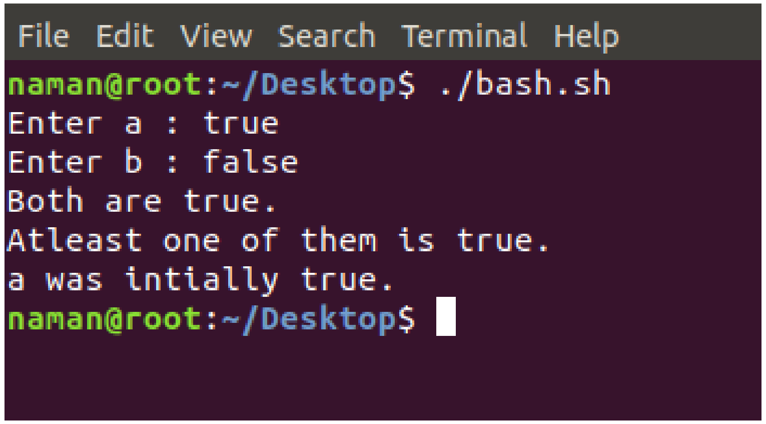
```
#!/bin/bash

#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b

if(($a == "true" & $b == "true" ))
then
        echo Both are true.
else
        echo Both are not true.
fi

if(($a == "true" || $b == "true" ))
then
        echo Atleast one of them is true.
else
        echo None of them is true.
fi

if(( ! $a == "true" ))
then
        echo "a" was initially false.
else
        echo "a" was initially true.
fi
```



```
naman@root:~/Desktop$ ./bash.sh
Enter a : true
Enter b : false
Both are true.
Atleast one of them is true.
a was intially true.
naman@root:~/Desktop$
```

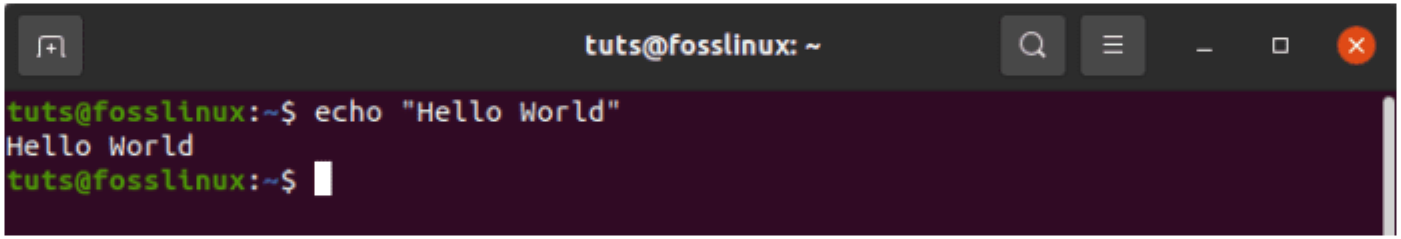
if-then-fi for decision making

- **Following are all examples of expression:**
- 5 > 2
- 3 + 6
- 3 * 65
- a < b
- c > 5
- c > 5 + 30 -1
- **Type following command (assumes you have file called foo)**
- **$ cat foo**
- **$ echo $?**
- The cat command return zero(0) on successful, this can be used in if condition as follows, Write shell script as
- **$ cat > showfile**
- **#!/bin/sh**
- **#**
- **#Script to print file**
- **#**
- **if cat $1**
- **then**
- **echo -e "\n\nFile $1, found and successfully echoed"**
- **fi**

File , found and successfully echoed

How to write shell script

- **First Bash Program**
- To get a bash file up and running, you need to execute it by running a terminal command. For instance, if we run "Hello World" in our terminal, the output we get will be "Hello World."

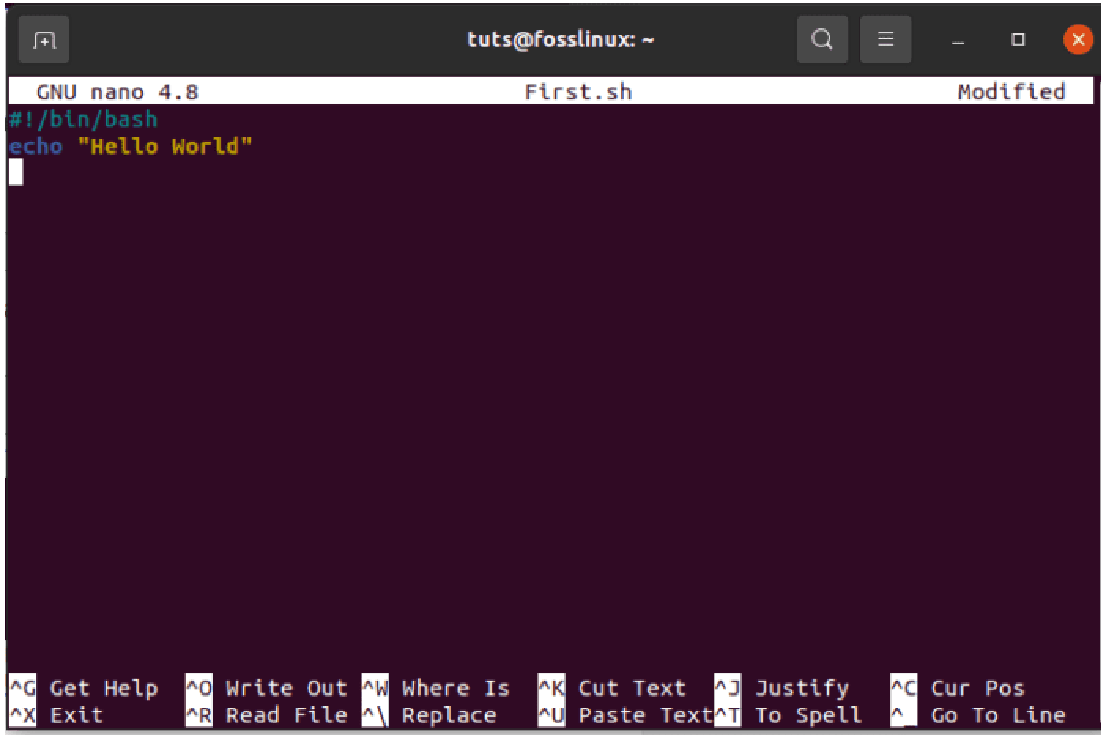


```
tuts@fosslinux:~$ echo "Hello World"
Hello World
tuts@fosslinux:~$
```

- To create a bash file, you can use any text editor installed in your operating system. In this case, we will use the nano editor for creating the file.

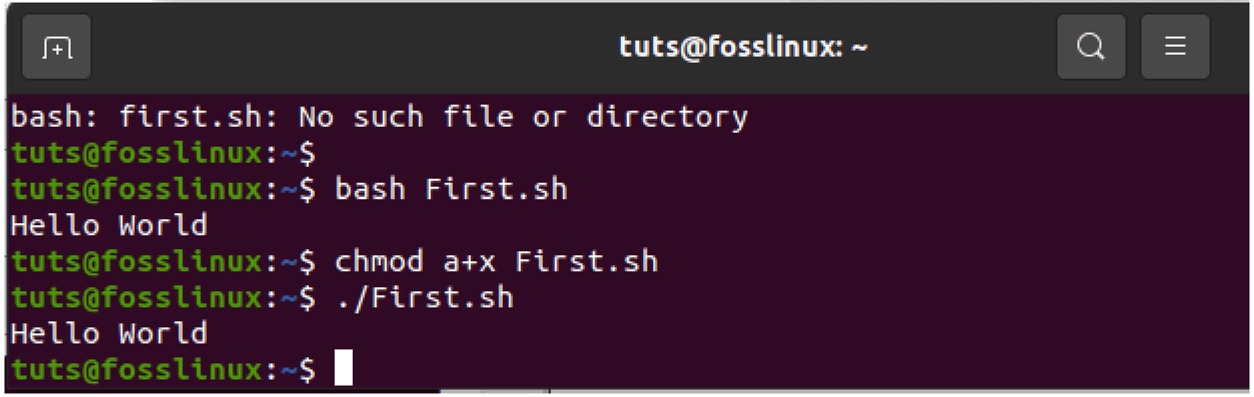
- Let us name our file **'First.sh'.** Execute the command using the following command:
- **nano First.sh**
- Add the following bash script to the file before saving the file.
- **#!/bin/bash**
- **echo "Hello World"**

There are different ways of running bash commands. For instance, below are two distinct examples of executing bash.
./First.sh
Alternatively, you can use execute the command below:
chmod a+x First.sh
./First.sh



```
  GNU nano 4.8                    First.sh                          Modified
#!/bin/bash
echo "Hello World"

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```



```
bash: first.sh: No such file or directory
tuts@fosslinux:~$
tuts@fosslinux:~$ bash First.sh
Hello World
tuts@fosslinux:~$ chmod a+x First.sh
tuts@fosslinux:~$ ./First.sh
Hello World
tuts@fosslinux:~$
```

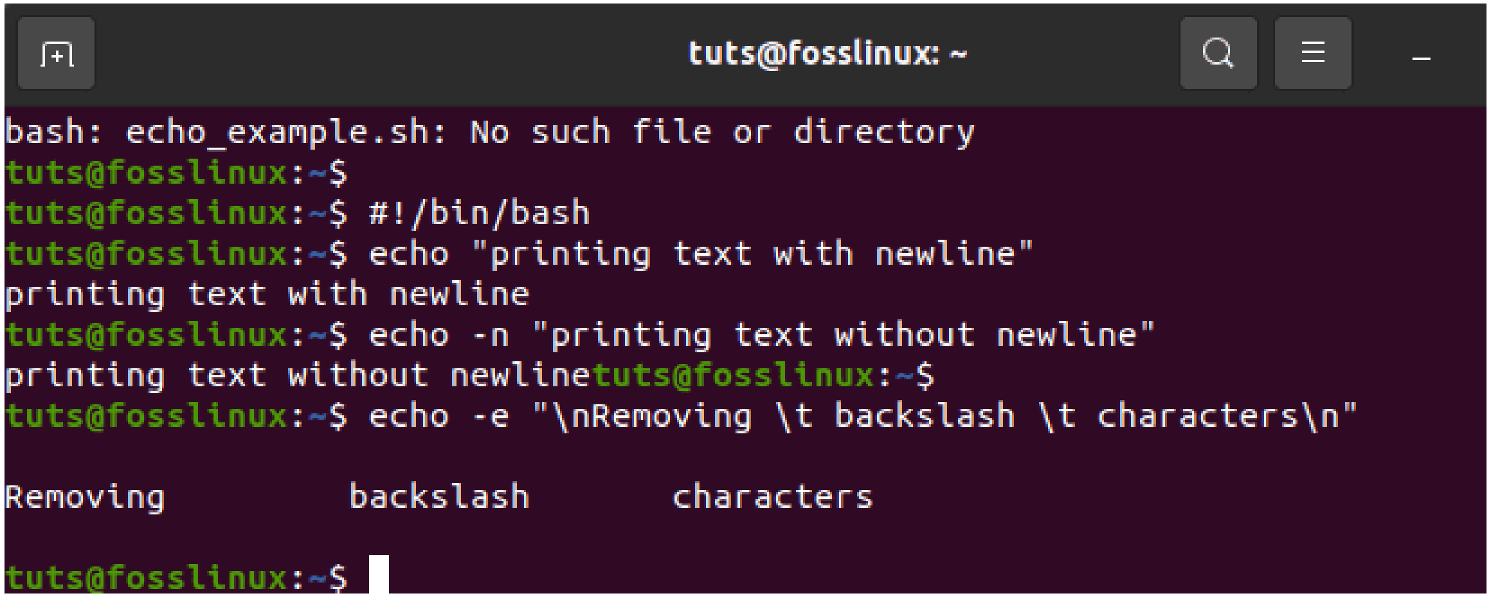
Echo commands

- echo commands have numerous options for selection.
- For instance, there is an addition of a new line by default if you use the ‘echo’ command without any other option.
- Alternatively, you can use **‘-n’** to print any text without a new line. Make use of the **‘-e’** command to remove backslash characters from the given output.
- To demonstrate this, create a bash file named ‘echo_example.sh’. After that, add the script below

- **#!/bin/bash**
- **Echo “printing text with newline”**
- **Echo -n “printing text without newline”**
- **Echo -e “\nRemoving \t backslash \t characters**

Echo commands

- After adding the script, execute the file with the command below:
- **bash echo_example.sh**



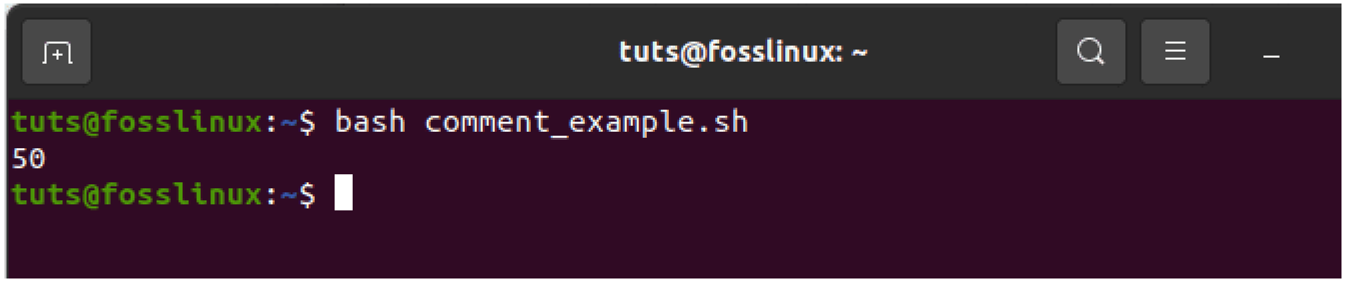
```
bash: echo_example.sh: No such file or directory
tuts@fosslinux:~$
tuts@fosslinux:~$ #!/bin/bash
tuts@fosslinux:~$ echo "printing text with newline"
printing text with newline
tuts@fosslinux:~$ echo -n "printing text without newline"
printing text without newlinetuts@fosslinux:~$
tuts@fosslinux:~$ echo -e "\nRemoving \t backslash \t characters\n"

Removing         backslash       characters

tuts@fosslinux:~$
```

Use of comment “#”

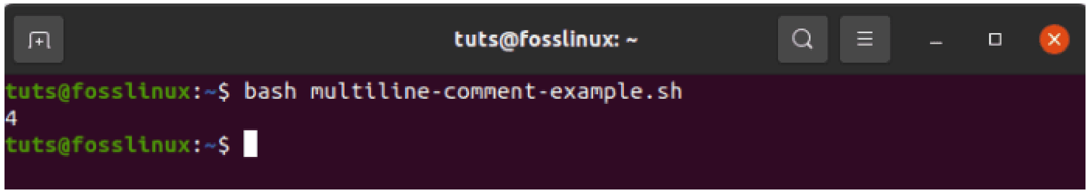
- While using comments, we use **the “#” symbol** to add a single line comment in bash scripting. Here, you will create a new file with a simple name such as ‘comment_example’. Include a script with a single comment like the example displayed below.
- **#!/bin /bash**
- **# Add two values**
- **((sum 30+20))**
- **#thereafter print the result**
- **echo $sum**
- execute the file with bash command-line



```
tuts@fosslinux:~$ bash comment_example.sh
50
tuts@fosslinux:~$ 
```

Multiline comment

- In bash, the multiline comment is applicable in different ways.
- To prove this, create a new bash named, ‘multiline-comment example.sh’, after that, **add ‘:’ and “ ’ ” scripts symbols** to add a multi-line comment in the script. The following example will execute the square of 2.
- **#!bin/bash**
- **: ‘**
- **The script written below is used to calculate the square of 2**
- **‘**
- **((area=2*2))**
- **echo$area**
- execute the file with bash command-line



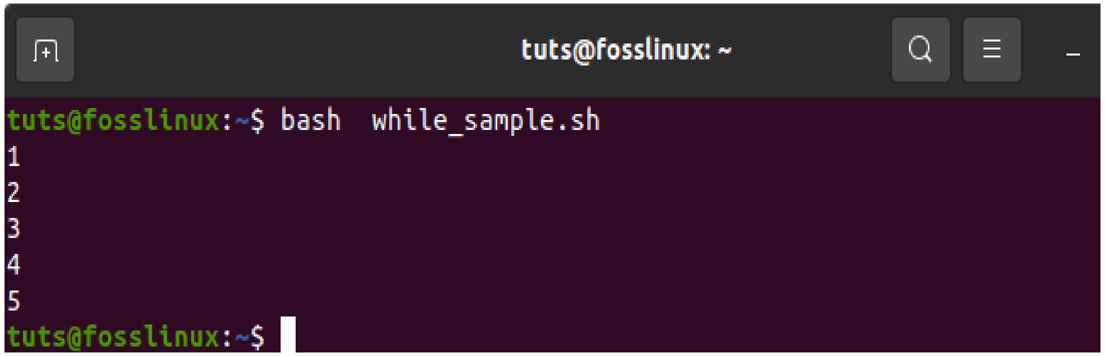
```
tuts@fosslinux: ~
tuts@fosslinux:~$ bash multiline-comment-example.sh
4
tuts@fosslinux:~$
```

While Loop

- For easy comprehension of this bash script, create a file named ‘while_sample.sh’. **The while loop will repeat five times before terminating the process.** While looping, the count variable increases the count by 1 in every step till the fifth time when the loop stops.

- **#!/bin/bash**
- **valid=True**
- **count=1**
- **while [$valid]**
- **do**
- **echo $count**
- **if [$count -eq 5];**
- **then break**
- **fi**
- **((count++))**
- **done**

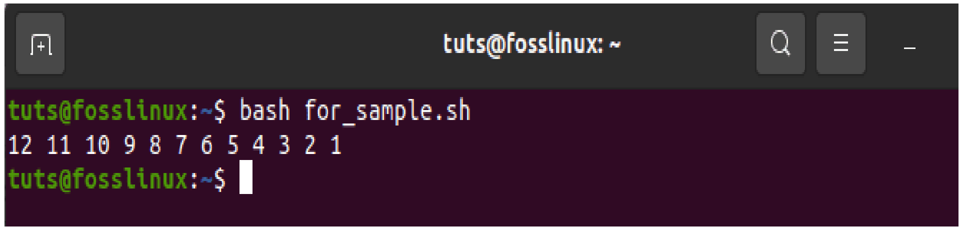
execute the file with bash command-line
bash while_example.sh



```
tuts@fosslinux:~$ bash  while_sample.sh
1
2
3
4
5
tuts@fosslinux:~$
```

For Loop

- Take a look at the following for loop example. After creating a file named ‘for_sample.sh’, add the script using ‘for loop’.
- This process will re-occur 12 times. After that, it will display the fields in a single line, as shown below;
- **#!/bin/bash**
- **for ((counter=10; counter>0; counter--))**
- **do**
- **echo -n "$counter "**
- **done**
- **printf "\n"**

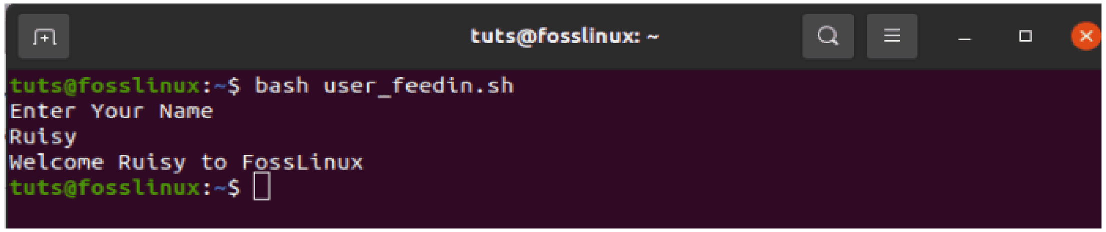


```
tuts@fosslinux:~$ bash for_sample.sh
12 11 10 9 8 7 6 5 4 3 2 1
tuts@fosslinux:~$
```

Execute the command by running the code below
bash for_sample.sh

Get User Input

- To get user input from bash, we will use the ‘read’ command. Follow the simple steps below to achieve the expected results.
- First, create a file named ‘user_feedin.sh’ and include the script below to get the user input. One value will be taken and displayed by combining other string values. As indicated below,



```
tuts@fosslinux:~$ bash user_feedin.sh
Enter Your Name
Ruisy
Welcome Ruisy to FossLinux
tuts@fosslinux:~$
```

- **#!/bin/bash**
- **echo "Enter Your Name"**
- **read name**
- **echo "Welcome $name to FossLinux"**

execute the file with bash command-line
bash user_feedin.sh

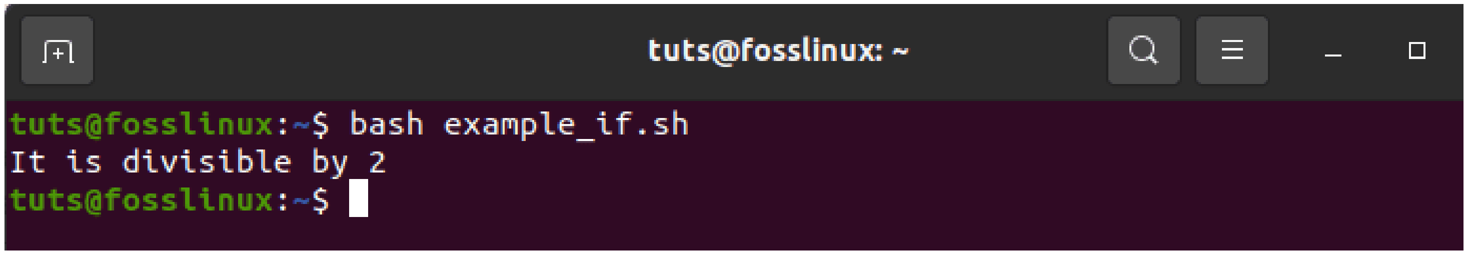
If statement

- The if statement is used by both multiple and single conditions. A definition of **‘if’ and ‘fi’** are used Before and after an if statement.
- To easily understand the if statement in bash, we shall use an example. Create a file named ‘example_if.sh’.
- For instance, the number 4 is assigned a variable ‘s.’ If the number is divisible by 2, then the output will be “it is divisible by 2”; otherwise, if the number is not divisible by 2, then the outcome will be “it is not divisible by 2”. The ‘-lt’, in this case, is used for comparison purposes. Another comparison feature is **‘-eq.’** **‘-ne**’, on the other hand, is used to show inequality while **‘-gt’** shows if a value is more significant in bash script.

If statement

- **#!/bin/bash**
- **s=4**
- **if [$s / 2];**
- **then**
- **echo "It is not divisible by 2"**
- **else**
- **echo "It is divisible by 2"**
- **fi**

To easily understand the if statement in bash, we shall use an example. Create a file named ‘example_if.sh’.



```
tuts@fosslinux:~$ bash example_if.sh
It is divisible by 2
tuts@fosslinux:~$ 
```

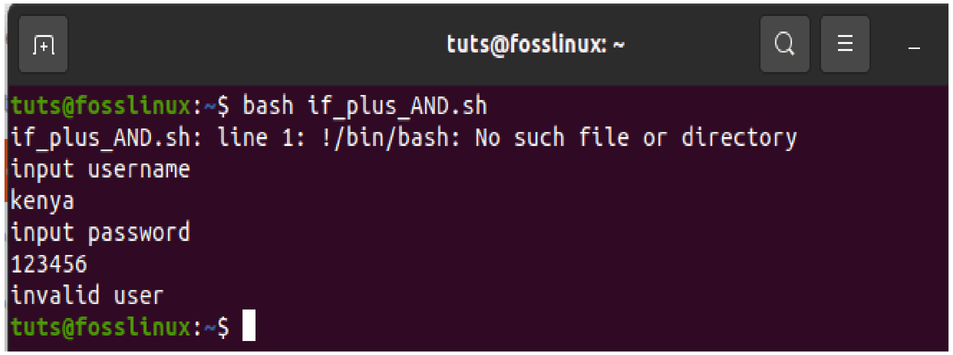
execute the file with bash command-line
bash example_if.sh

Use of if statement together with AND logic

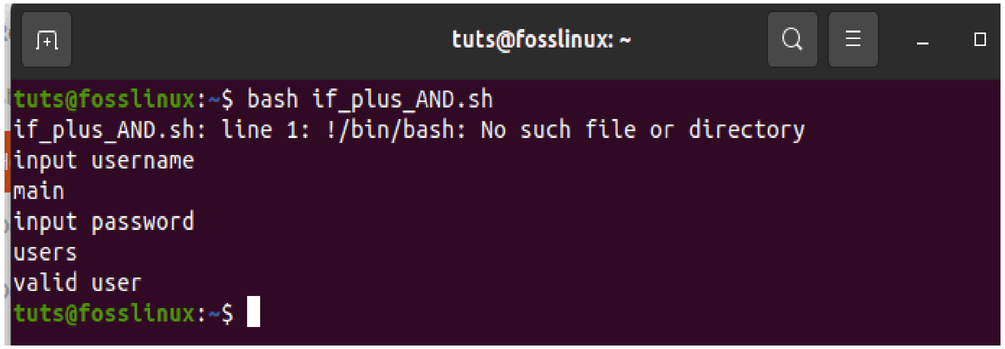
- Various logical conditions can be used with the if statement whenever there are two or more conditions.
- The next example shows how the logic **“AND”** is used in defining multiple conditions in an if statement. The **“&&”** symbols represent the “AND” logic in a bash script. Create a file named ‘if_plus_AND.sh’.

- In this example, **the username and password variables** entered by the user will be compared with the “main” and “users” directory to see if they match. If they do, the process will be successful, thus displaying “valid-user” as the output. Otherwise, if they do not match, the outcome will be “invalid user.”

- !/bin/bash
- echo "input username"
- read username
- echo "input password"
- read password
- if [[($username == "main" && $password == "users")]]; then
- echo "valid user"
- else
- echo "invalid user"
- fi



```
tuts@fosslinux:~$ bash if_plus_AND.sh
if_plus_AND.sh: line 1: !/bin/bash: No such file or directory
input username
kenya
input password
123456
invalid user
tuts@fosslinux:~$
```



```
tuts@fosslinux:~$ bash if_plus_AND.sh
if_plus_AND.sh: line 1: !/bin/bash: No such file or directory
input username
main
input password
users
valid user
tuts@fosslinux:~$
```

Execute the file using the bash command-line
bash if_plus_AND.sh

The first example shows authentication failure since the user-provided does not match with the main fields.
The second examples show successful authentication since the provided fields matched with the main fields.

Use if statement with OR logic

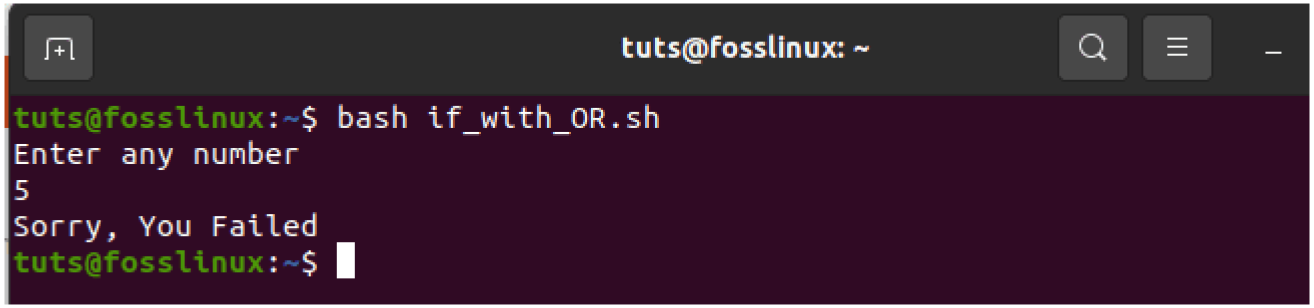
- When using OR with the if function, the **‘||’** symbol is used.
- To demonstrate this, we will create a file named ‘if_with_OR.sh’ to check the use of OR logic in an IF statement.
- Take an instance of value **‘s’** being assigned to two numbers (10 or 40). If a user inputs either of the given numbers, then the system’s output will be “Well Played”; otherwise, the result shown will be **“Sorry, You Failed.”**
- If you examine this example, you will notice that the value of s is generated from the user.

- **#!/bin/bash**
- **echo "Enter any number"**
- **read s**
- **n=s**
- **if [[($s -eq 10 || $n -eq 40)]]**
- **then**
- **echo "Well Played"**
- **else**
- **echo "Sorry, You Failed"**
- **fi**

As indicated in the example above, 5 is not equal to 10 or 40. Therefore, the output displays “Sorry, You Failed,”

In the figure, the user was prompted to enter any number, and he/she chose 10, and the output given is “Well Played” since 10==10

execute the file with bash command-line
bash if_with_OR.sh



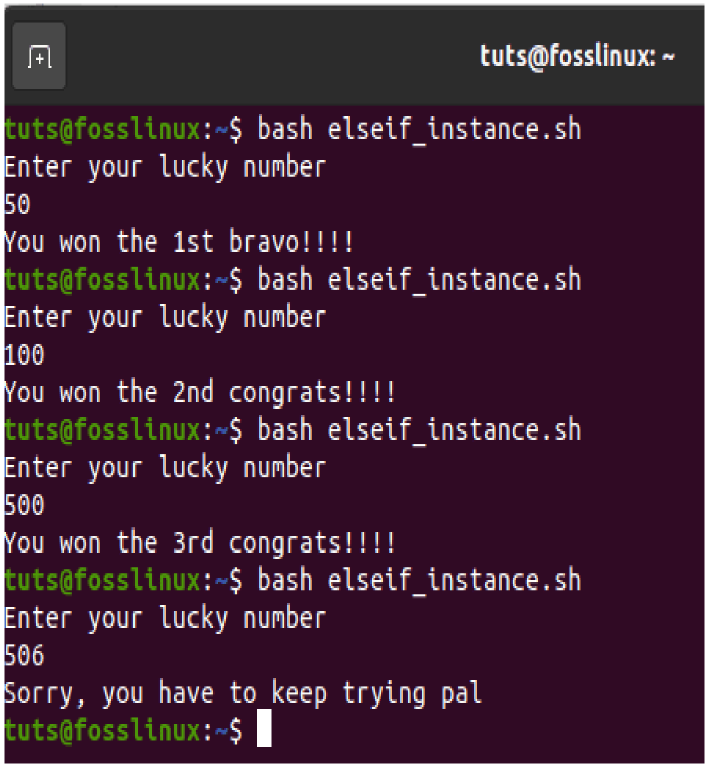
```
tuts@fosslinux:~$ bash if_with_OR.sh
Enter any number
5
Sorry, You Failed
tuts@fosslinux:~$
```

Use of else if statement

- Many conditional statements are nearly the same despite the programming languages you choose.
- However, in bash programming, the use of the **‘else if’** condition is kind of different. In bash, **Elif** is used in place of the else if condition.
- We will create a file named ‘elseif_instance.sh’ then add the bash script for demonstration purposes.

- echo "Enter your lucky number"
- read n

- if [$n -eq 50];
- then
- echo "You won the 1st bravo!!!!"
- elif [$n -eq 100];
- then
- echo "You won the 2nd congrats!!!!"
- elif [$n -eq 500];
- then
- echo "You won the 3rd congrats!!!!"
- else
- echo "Sorry, you have to keep trying pal"
- fi


tuts@fosslinux: ~

```
tuts@fosslinux:~$ bash elseif_instance.sh
Enter your lucky number
50
You won the 1st bravo!!!!
tuts@fosslinux:~$ bash elseif_instance.sh
Enter your lucky number
100
You won the 2nd congrats!!!!
tuts@fosslinux:~$ bash elseif_instance.sh
Enter your lucky number
500
You won the 3rd congrats!!!!
tuts@fosslinux:~$ bash elseif_instance.sh
Enter your lucky number
506
Sorry, you have to keep trying pal
tuts@fosslinux:~$
```

Execute the file with bash command-line
Bash elseif_instance.sh

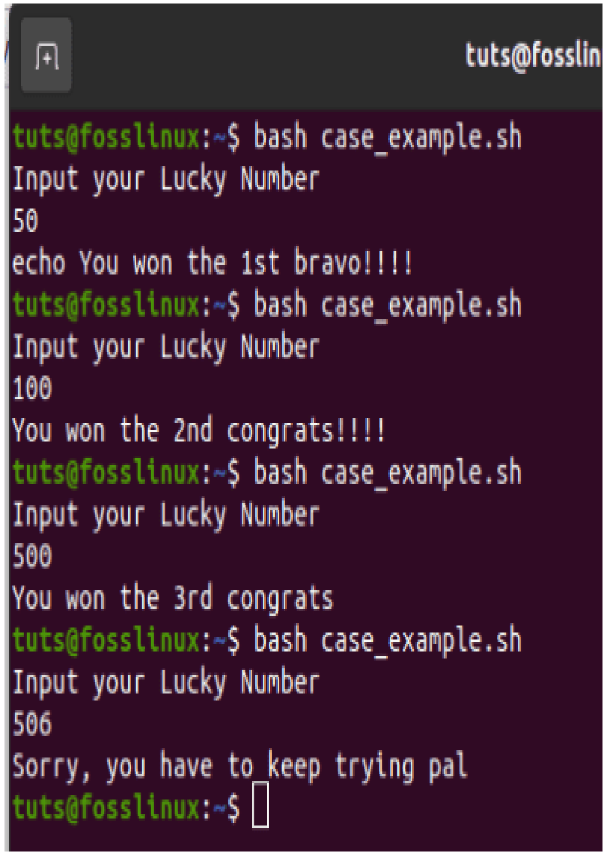
Case statement

- The Case statement is used as a substitute for the if-elseif-else statement in case of different cases scenario that can’t be the same type.
- **‘Case’ and ‘esac’** delineate the starting and ending block respectively while using this statement.
- Create a file named ‘case_example.sh’. After that, include the script provided below.
- Then, take a look at the output and compare it to the previous one.
- Compare it with if-elseif-else statements.

- **#!/bin/bash**

- **echo "Input your Lucky Number"**
- **read s**
- **case $s in**
- **50)**
- **echo echo "You won the 1st bravo!!!!" ;;**
- **100)**
- **echo "You won the 2nd congrats!!!!" ;;**
- **500)**
- **echo "You won the 3rd congrats" ;;**
- ***)**
- **echo "Sorry, you have to keep trying pal" ;;**
- **esac**

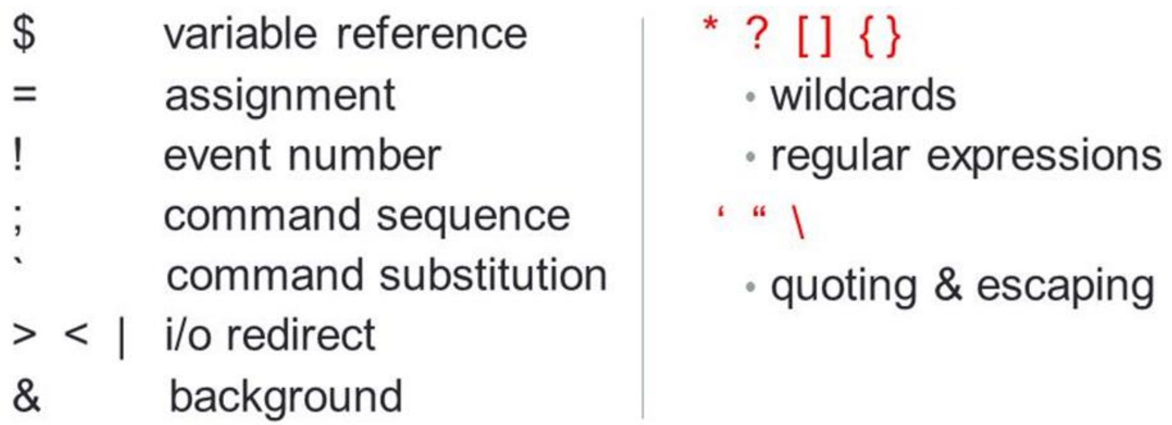
execute the file with bash command-line
bash case_example.sh



```
tuts@fosslinux:~$ bash case_example.sh
Input your Lucky Number
50
echo You won the 1st bravo!!!!
tuts@fosslinux:~$ bash case_example.sh
Input your Lucky Number
100
You won the 2nd congrats!!!!
tuts@fosslinux:~$ bash case_example.sh
Input your Lucky Number
500
You won the 3rd congrats
tuts@fosslinux:~$ bash case_example.sh
Input your Lucky Number
506
Sorry, you have to keep trying pal
tuts@fosslinux:~$
```

Special Characters You Need to Know for Bash

- If you want to master the Bash shell on Linux, macOS, or another UNIX-like system, special characters (like ~, *, |, and >) are critical. We’ll help you unravel these cryptic Linux command sequences and become a hero of hieroglyphics.
- **What Are Special Characters?**
- There are a set of characters the Bash shell treats in two different ways. When you type them at the shell, they act as instructions or commands and tell the shell to perform a certain function. Think of them as single-character commands.



$	variable reference
=	assignment
!	event number
;	command sequence
`	command substitution
> < |	i/o redirect
&	background

* ? [] {}
- wildcards
- regular expressions

‘ “ \
- quoting & escaping

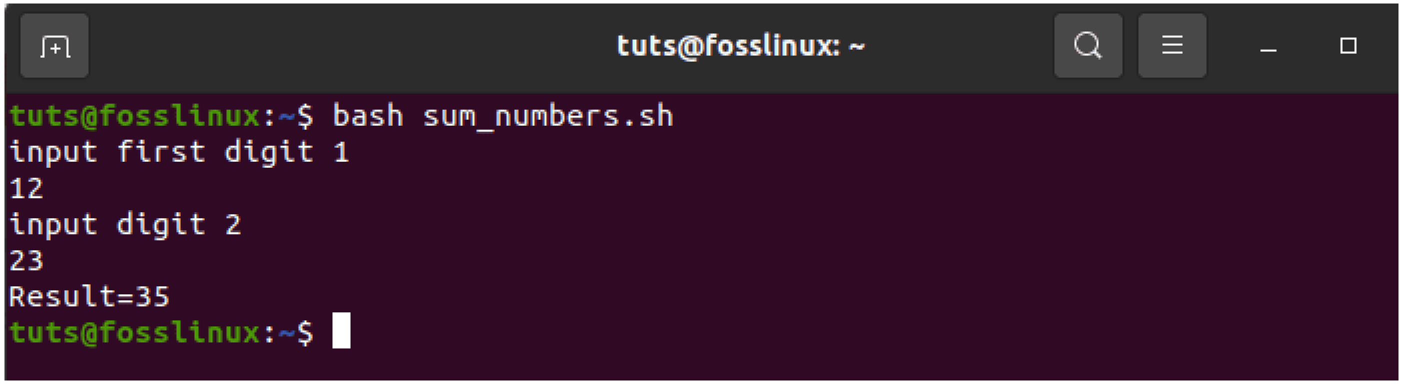
Special Characters

- ~ Home Directory
- . Current Directory , .. Parent Directory
- / Path Directory Separator
- # Comment or Trim Strings
- ? Single Character Wildcard
- * Character Sequence Wildcard
- [] Character Set Wildcard
- ; Shell Command Separator
- & Background Process
- $ Variable Expressions

Addition of two numbers

- #!/bin/bash
- echo "input first digit 1"
- read a
- echo "input digit 2"
- read b
- ((sum=a+b))
- echo "Result=$sum"

Execute the file with bash command-line
bash sum_numbers.sh


tuts@fosslinux: ~

```
tuts@fosslinux:~$ bash sum_numbers.sh
input first digit 1
12
input digit 2
23
Result=35
tuts@fosslinux:~$
```

Thank You

Linux

Dr. Hatem Yousry
E-mail: Hyousry@nctu.edu.eg