25 Bash Script Examples

By **Habib Ahmed** September 2, 2020

An operating system has two major core components Kernel and Shell. A kernel is the brain of the operating system that controls everything in the system. To protect the kernel from direct user interaction, there is an outer wrap called Shell.

What is Shell?

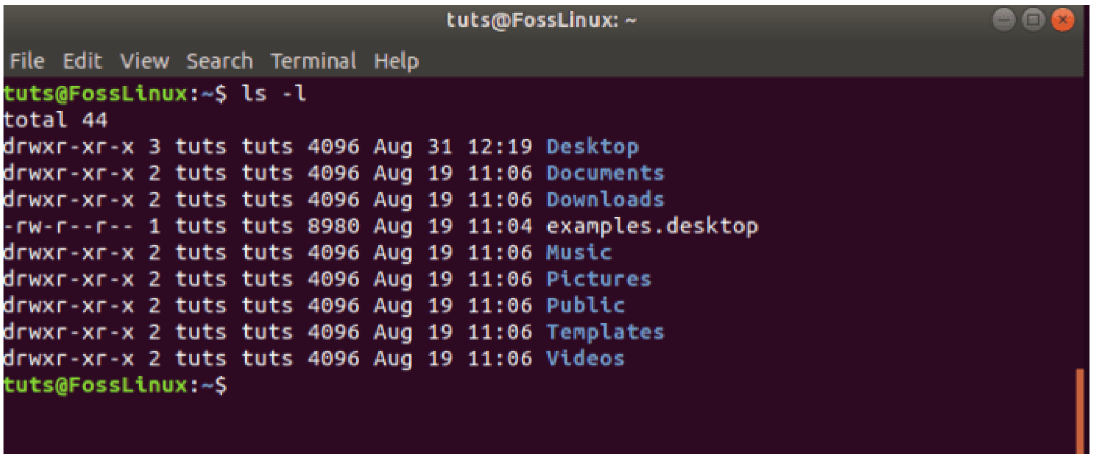
Shell is a unique program that provides the user an interface to interact with kernel accepting human-readable commands and then converts it to kernel understandable language. Shell, in a Linux operating system, can take input from the user in the form of commands, processes it, and then displays an output. You can access Shell using Terminal in Linux.

Shell has two categories:

1. Command-Line Shell
2. Graphical Shell

Command-Line Shell

A shell can be accessed by a user using command-line interfaces. We have programs like the terminal in (Linux or Mac) and Command Prompt in Windows to get input in the form of human-readable commands and then display output in the same command-line interface.



```
tuts@FossLinux:~$ ls -l
total 44
drwxr-xr-x 3 tuts tuts 4096 Aug 31 12:19 Desktop
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Documents
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Downloads
-rw-r--r-- 1 tuts tuts 8980 Aug 19 11:04 examples.desktop
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Music
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Pictures
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Public
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Templates
drwxr-xr-x 2 tuts tuts 4096 Aug 19 11:06 Videos
tuts@FossLinux:~$
```

Ubuntu Terminal Shell

Graphical Shell

Graphical shell provides users a Graphical User Interface (GUI) to interact, perform operations like opening, closing, saving files. Windows OS and Ubuntu are great examples of GUI Shell (Desktop), where a user does not have to type commands for every operation. Still, behind every action, there is a shell command that executes to perform these actions.

What is BASH Shell?

BASH (Bourne Again Shell) is the default command-line interpreter for most of the Linux Distros these days. It is an updated version of the earlier Bourne shell. If you are a Linux system administrator or a power user, you must have excellent knowledge of BASH shell commands to perform day to day tasks.

What is BASH Scripting?

Mostly we use shell commands one by one in the terminal for our everyday tasks. Still, sometimes you have to perform complex tasks or repetitive tasks, which involves a series of commands being executed in a proper sequence. A shell can also take commands as input from a file, so to make our job easy, we can write these commands in a file and can execute them in the shell to avoid manual work. These files are called shell scripts.

Let's understand the BASH Shell Scripting

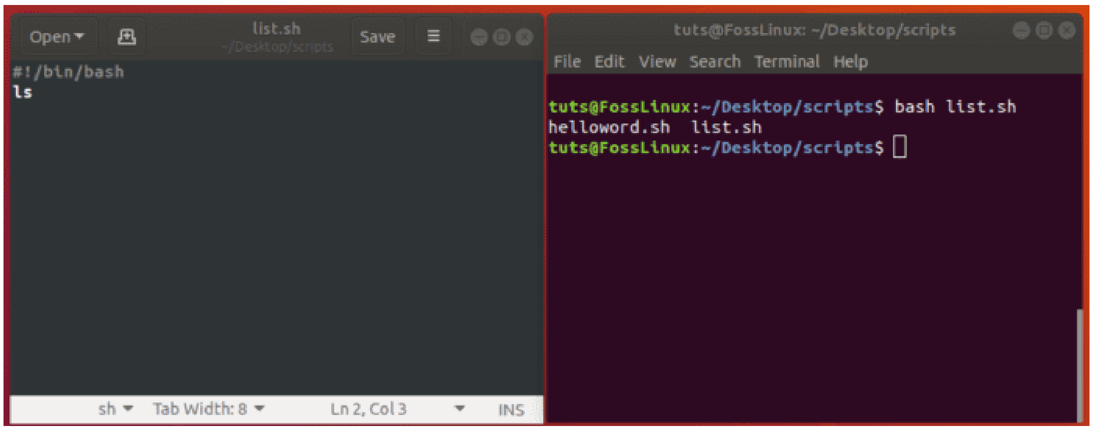
1. Create a file using a text editor with .sh extension
2. Start the script with #!/bin/bash
3. Write some code/commands
4. Save the script file as filename.sh

So here is a sample Shell Script file:

```
#!/bin/sh
ls
```

We will name it 'list.sh' and to run it in the terminal we will use the below command:

```
$ bash list.sh
```



```
#!/bin/bash
ls
```

```
tuts@FossLinux:~/Desktop/scripts$ bash list.sh
helloword.sh  list.sh
tuts@FossLinux:~/Desktop/scripts$
```

Open
list.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 2, Col 3
INS

Show files list Shell Script

Advantages

The bash script has many advantages:

- It automates repetitive work that saves a lot of effort and time.
- You can create your power tool or utility.
- Bash scripts are portable; you may use a script on other Linux systems without any modification.
- It has the same set of the syntax that we use in standard terminal, so do not involves additional learning.
- You can quickly write a bash script with little help.
- It can provide interactive debugging while running tasks that help in case of error or issue.

Disadvantages

The bash script can have disadvantages:

- Prone to errors, a single mistake can change the program's flow and can be harmful.
- Slow execution speed.
- Have very minimal data structures, unlike other programming languages.
- Not well suited for large and complex tasks.

In this article, we will help you to get the basic idea of bash scripting. Most of the widely used operations of bash scripting will be explained with simple scripting examples.

1. Hello World

"Hello world" program is the very first program that a programmer writes while learning any new language. It's a program that prints the "*Hello World*" as an output. So you can create a file helloword.sh using editor (vim or nano) as follows:

```
$ nano helloword.sh
```

Now copy below lines into 'helloworld.sh' and save it.

```
#!/bin/bash
echo "Hello World"
```

Now you can run the command:

```
$ bash helloworld.sh
```

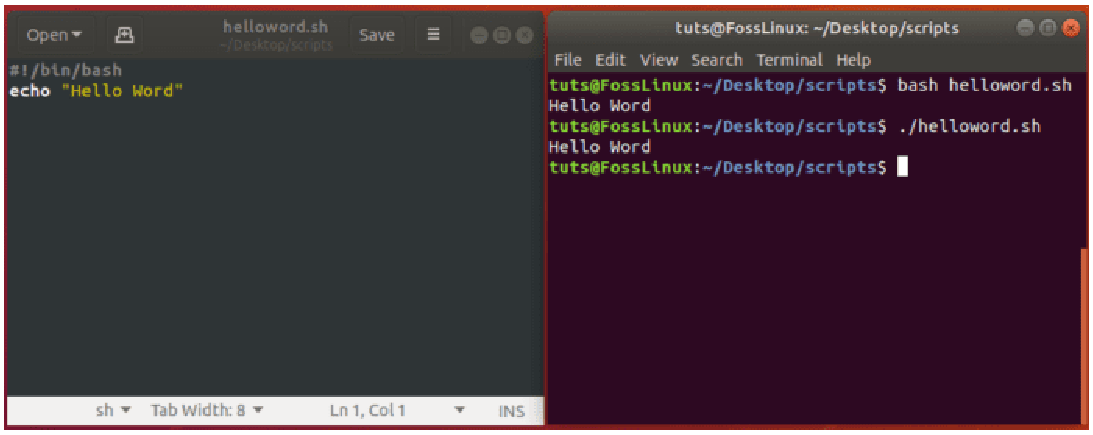
Another method is first to make the file executable:

```
$ chmod a+x helloworld.sh
```

And now, run the file using the below command.

```
$ ./helloworld.sh
```

Output:


Open
helloword.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 1, Col 1
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash
echo "Hello Word"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash helloword.sh
Hello Word
tuts@FossLinux:~/Desktop/scripts$ ./helloword.sh
Hello Word
tuts@FossLinux:~/Desktop/scripts$
```

hello world bash script

2. Using Echo Command

Echo command is the most common and frequently used command in Linux. It is used to print text or output in the Bash. It has many options that perform different operations.

Syntax:

```
echo [options] [ARGUMENTS]
```

Here options are:

`-n` is used to suppress trailing new line

`-e` is used to interpret backslash-escaped characters

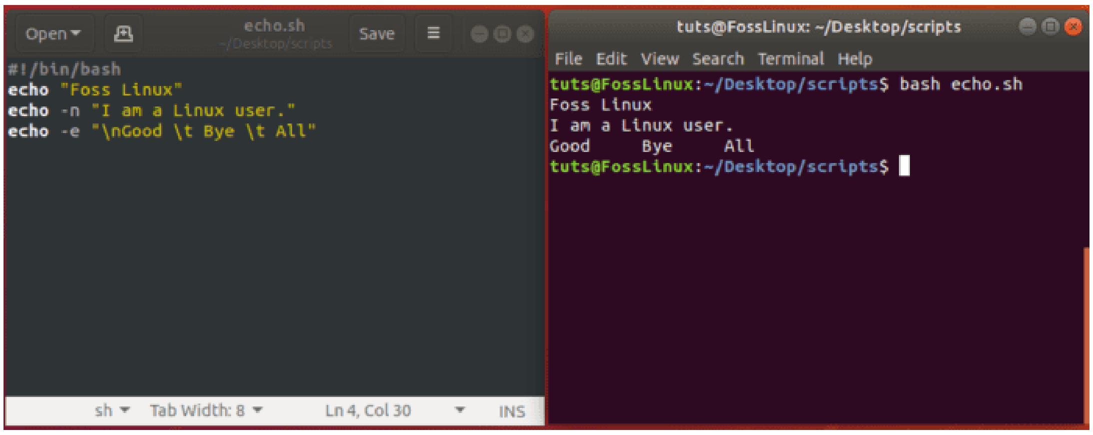
`-E` is used to disables the interpretation of the escape characters, and it is the default option for the echo command.

Create a new file echo.sh and add the below lines in it.

```
#!/bin/bash
echo "Foss Linux"
echo -n "I am a Linux User"
echo -e "\nGood \t Bye \t All"
```

Here \n is an escape character for a new line, and \t is an escape character for the horizontal tab.

Output:



```
#!/bin/bash
echo "Foss Linux"
echo -n "I am a Linux user."
echo -e "\nGood \t Bye \t All"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash echo.sh
Foss Linux
I am a Linux user.
Good     Bye     All
tuts@FossLinux:~/Desktop/scripts$
```

Open
echo.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 4, Col 30
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

echo command example

3. Using Comments

Comments are a programmer's remarks about the purpose of the code or logic. It's a widespread practice to add comments so that in the future, anyone can understand code by just reading comments. Comments are part of code but ignored by the compiler. In the bash script, any line that starts with # is considered a comment. For example:

```
#!/bin/bash

# this is a comment
echo "Comment Example"
```

Here' # this is a comment' is a comment, and when we run this script compiler will ignore the line.

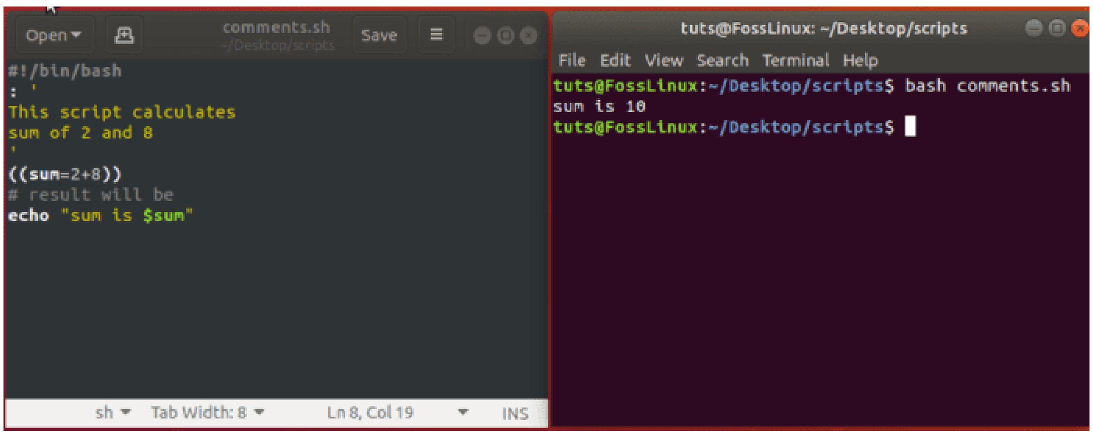
Comments can be:

1. Single Line Comment
2. Multiple Line Comment

We use '#' for single line comment and: 'content' for multiple line comments. Check the below command for both single and numerous comments in a bash script.

```
#!/bin/bash
: '
This script calculates
sum of 2 and 8.
'
((sum=2+8))
# result will be
echo "sum is $sum"
```

Output:


Open
comments.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 8, Col 19
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash
: '
This script calculates
sum of 2 and 8
'
((sum=2+8))
# result will be
echo "sum is $sum"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash comments.sh
sum is 10
tuts@FossLinux:~/Desktop/scripts$
```

bash comments example

4. Using Variables

Variables are named symbols used to store values temporarily. It can be a string or numeric value that we may use at any place within the script. You can make variables and assign them values. Variable names should be descriptive so that you can understand the purpose you created that variable.

We have three kinds of variables in bash scripts:

1. Special Variables:

The following are the other unique preset variables:

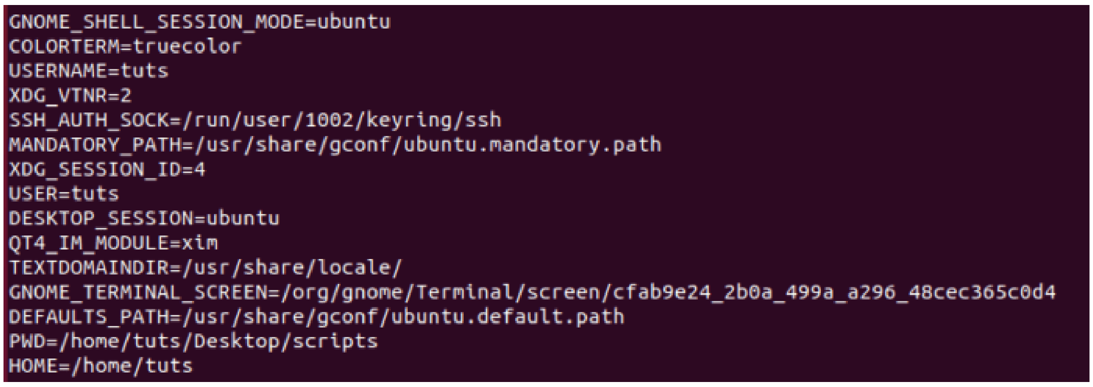
- $#: number of command line parameters that were passed to the script.
- $@: All the parameters sent to the script.
- $?: The end status of the last process to execute.
- $$: The Process ID of the current script.
- $USER: The user executing the script.
- $HOSTNAME: The hostname of the machine executing the script.
- $SECONDS: The number of seconds the script has been running for.
- $RANDOM: Returns a random number.
- $LINENO: Returns the current line number of the script.

2. Environment Variables:

To see the active environment variables in your Bash session, use the command:

```
env | less
```

Output:



```
GNOME_SHELL_SESSION_MODE=ubuntu
COLORTERM=truecolor
USERNAME=tuts
XDG_VTNR=2
SSH_AUTH_SOCK=/run/user/1002/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=4
USER=tuts
DESKTOP_SESSION=ubuntu
QT4_IM_MODULE=xim
TEXTDOMAINDIR=/usr/share/locale/
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/cfab9e24_2b0a_499a_a296_48cec365c0d4
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PWD=/home/tuts/Desktop/scripts
HOME=/home/tuts
```

environment variables

3. User-Defined Variables:

User-defined variables are those which are set by us in our script. For example, we have variable 'year' to store the current year like below.

```
year=2020
```

And we can later use

```
echo $year
```

you can see that we used `$` to reference its value.

So now create a file variables.sh and add the below lines in it.

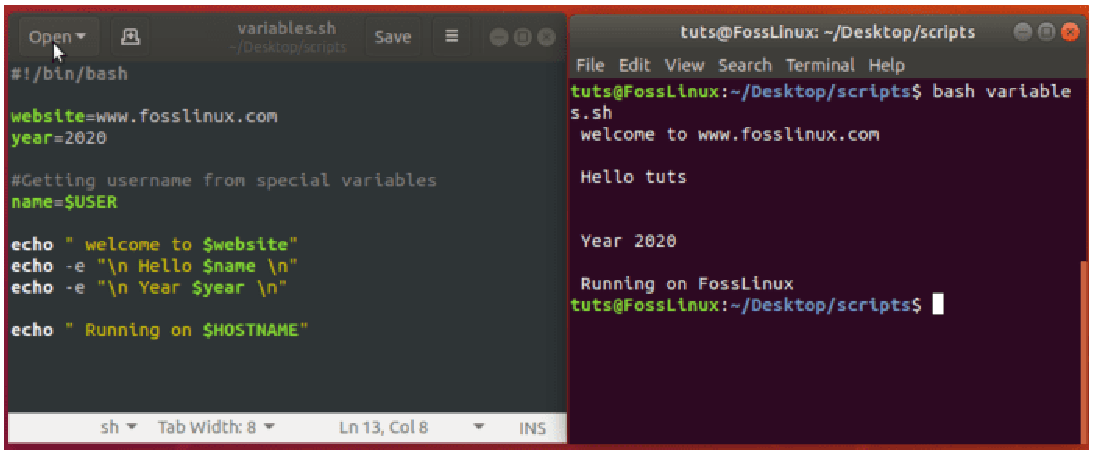
```
#!/bin/bash

website=www.fosslinux.com
year=2020

# Getting user name from special variables
name=$USER

echo "Welcome to $website"
echo -e "Hello $name \n"
echo -e "Year = $year \n"
echo "Running on $HOSTNAME"
```

Output:


Open
variables.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 13, Col 8
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash

website=www.fosslinux.com
year=2020

#Getting username from special variables
name=$USER

echo " welcome to $website"
echo -e "\n Hello $name \n"
echo -e "\n Year $year \n"

echo " Running on $HOSTNAME"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash variable
s.sh
 welcome to www.fosslinux.com

 Hello tuts


 Year 2020

 Running on FossLinux
tuts@FossLinux:~/Desktop/scripts$
```

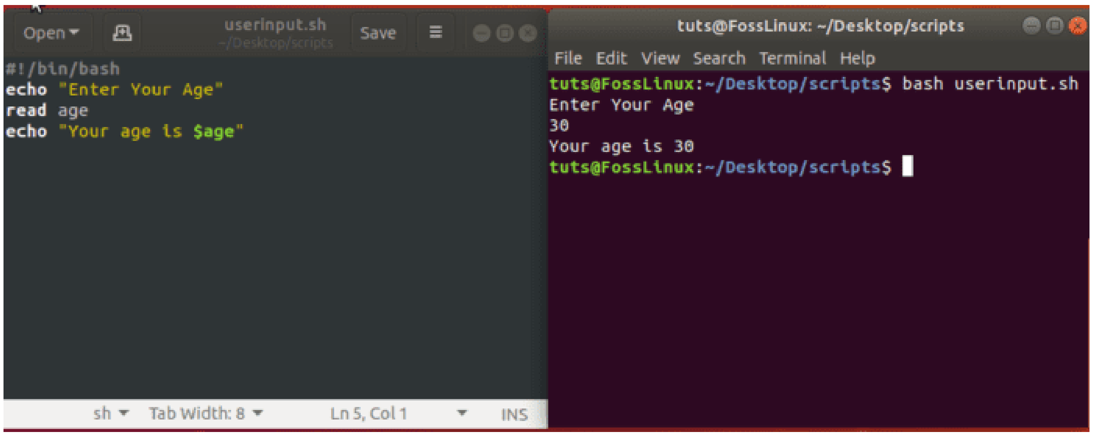
variables example

5. Getting User Input

Getting user input is very crucial for making a script interactive, so for this purpose in bash script, we use ‘read’ command.

```
#!/bin/bash
echo "Enter Your Age"
read age
echo "Your age is $age"
```

Output:


Open
userinput.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 5, Col 1
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash
echo "Enter Your Age"
read age
echo "Your age is $age"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash userinput.sh
Enter Your Age
30
Your age is 30
tuts@FossLinux:~/Desktop/scripts$
```

getting user input example

6. Using Command Arguments

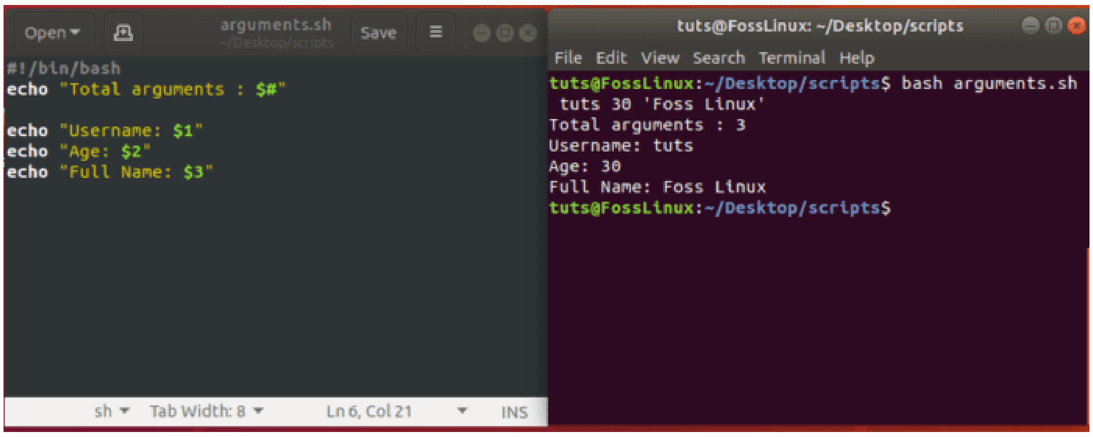
We can also read user input from command arguments, just like any other programming language. We can then use these arguments in our scripts as $1, $2, and so on, depending on the number of arguments we have provided. Create a file’ arguments.sh’ and copy the below lines in it.

```
#!/bin/bash
echo "Total arguments : $#"
echo "Username: $1"
echo "Age: $2"
echo "Full Name: $3"
```

Now run ‘arguments.sh’ script file with three additional parameters after its name.

```
$ bash arguments.sh tuts 30 'Foss Linux'
```

Output:


Open
arguments.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 6, Col 21
INS

```
#!/bin/bash
echo "Total arguments : $#"

echo "Username: $1"
echo "Age: $2"
echo "Full Name: $3"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash arguments.sh
 tuts 30 'Foss Linux'
Total arguments : 3
Username: tuts
Age: 30
Full Name: Foss Linux
tuts@FossLinux:~/Desktop/scripts$
```

command arguments example

7. Using Loops

Loops are used in every programming language where you need to execute the same code repetitively. There are two types of loops in bash script while and for loops. We will see each one by one.

While Loop

While it is used when you need to repeat the line of code an unknown number of times until it satisfies certain conditions. Here is how it is formed:

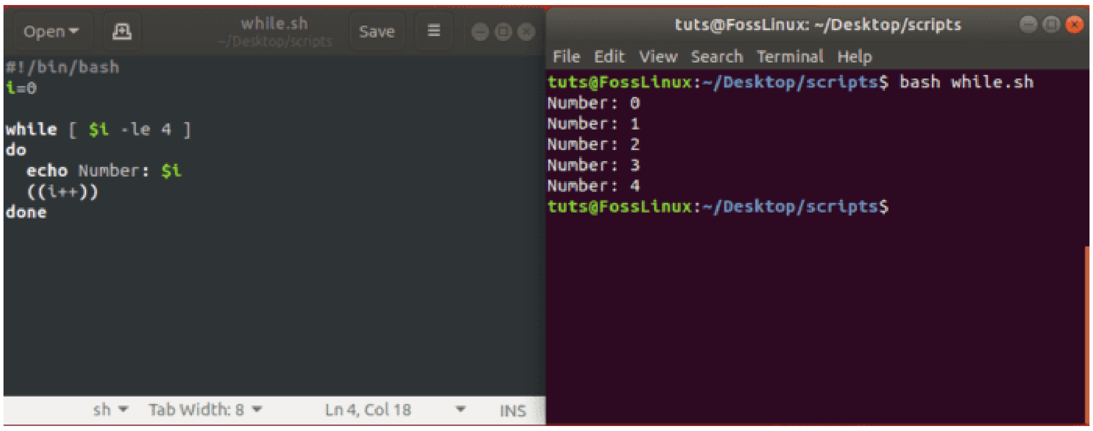
```
 #!/bin/bash
while [CONDITION]
do
  [COMMANDS]
done
```

The condition is evaluated before executing the commands at every iteration, and it will keep executing until the condition evaluates to false, and the loop will be terminated.

```
 #!/bin/bash
i=0

while [ $i -le 4 ]
do
  echo Number: $i
  ((i++))
done
```

Output:



```
#!/bin/bash
i=0

while [ $i -le 4 ]
do
  echo Number: $i
  ((i++))
done
```

```
tuts@FossLinux:~/Desktop/scripts$ bash while.sh
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
tuts@FossLinux:~/Desktop/scripts$
```

while loop example

For Loop

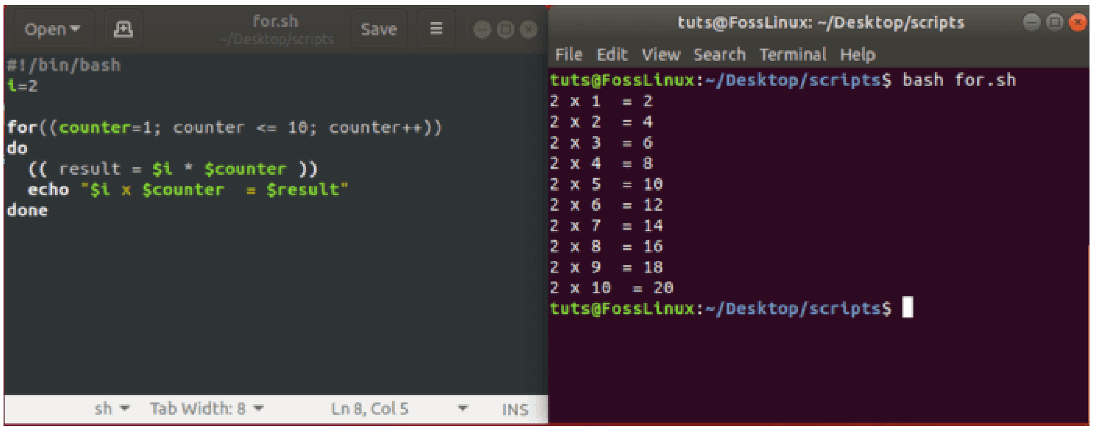
The `for` loop iterates over a list of items and performs the given set of commands. The Bash `for` loop takes the following form:

```
 #!/bin/bash
for item in [LIST]
do
  [COMMANDS]
done
```

In the example below, the loop will iterate over each item and will generate a table of variable i.

```
#!/bin/bash
i=2
for (( counter=1; counter<=10; counter++ ))
do
    ((result= $i * $counter))
    echo "$i x $counter = $result"
done
```

Output:



```
#!/bin/bash
i=2

for((counter=1; counter <= 10; counter++))
do
  (( result = $i * $counter ))
  echo "$i x $counter  = $result"
done
```

```
tuts@FossLinux:~/Desktop/scripts$ bash for.sh
2 x 1  = 2
2 x 2  = 4
2 x 3  = 6
2 x 4  = 8
2 x 5  = 10
2 x 6  = 12
2 x 7  = 14
2 x 8  = 16
2 x 9  = 18
2 x 10  = 20
tuts@FossLinux:~/Desktop/scripts$
```

for loop example

8. Using Conditional Statements

Conditional statements are one of the fundamental concepts of any programming language. You make decisions based on certain conditions fulfilled. In the bash script, we have conditional blocks.

if statement

In a bash script, if the condition has several forms but let's look at the basic condition.

```
 if Condition
then
  STATEMENTS
fi
```

You can see if statements start with evaluating the condition and Run statements between 'then' and 'fi', provided the "If" condition evaluates to True otherwise statement will be ignored.

```
 #!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
fi
```

In the above example, the user will be asked to input a number, and if the number is more than 10, you will see output 'The variable is greater than 10.', otherwise you will not see anything.

if else statement

Now we are going to add "if else" block as well, which will execute if the condition will be false.

```
 if Condition
then
  STATEMENTS1
else
  STATEMENTS2
fi
```

So we will modify the above example.

```
 #!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
else
  echo "The variable is equal or less than 10."
fi
```

If you execute the code and enter a number, the script will print a string based on whether the number is greater or less/equal to 10.

if elif statement

Bash has an equivalent syntax for 'else if' as well.

```
 if Condition1
then
  STATEMENTS1
elif Condition2
then
  STATEMENTS2
else
  STATEMENTS3
fi
```

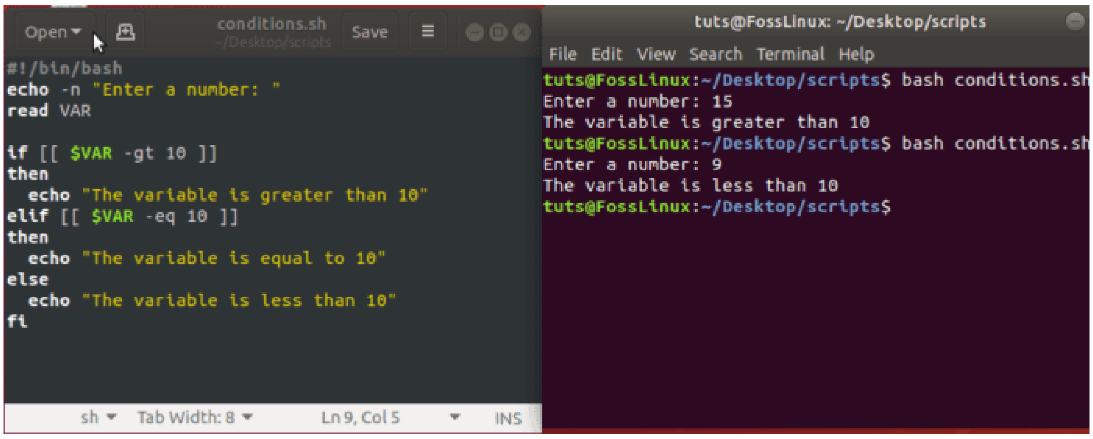
So after modifying the above example:

```
 #!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
elif [[ $VAR -eq 10 ]]
then
  echo "The variable is equal to 10."
else
  echo "The variable is less than 10."
fi
```

Output:



```
#!/bin/bash
echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10"
elif [[ $VAR -eq 10 ]]
then
  echo "The variable is equal to 10"
else
  echo "The variable is less than 10"
fi
```

```
tuts@FossLinux:~/Desktop/scripts$ bash conditions.sh
Enter a number: 15
The variable is greater than 10
tuts@FossLinux:~/Desktop/scripts$ bash conditions.sh
Enter a number: 9
The variable is less than 10
tuts@FossLinux:~/Desktop/scripts$
```

Open
conditions.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 9, Col 5
INS

conditional statements example

9. Using Functions

Just like other programming languages, the bash script also has the concept of functions. It allows the user to write a custom code block that will be required to be reused again and again.

Syntax:

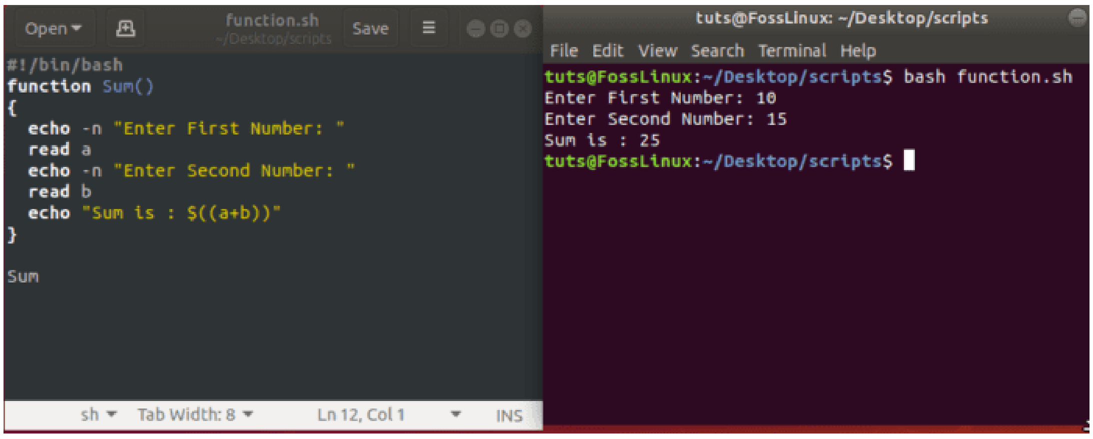
```
function FunctionName()
{
  statements
}
```

Now we shall create a function 'sum' that will take input numbers from the user and will show the sum of these numbers as output.

```
#!/bin/bash
function Sum()
{
  echo -n "Enter First Number: "
  read a
  echo -n "Enter Second Number: "
  read b
  echo "Sum is: $(( a+b ))"
}

Sum
```

Output:



```
#!/bin/bash
function Sum()
{
  echo -n "Enter First Number: "
  read a
  echo -n "Enter Second Number: "
  read b
  echo "Sum is : $((a+b))"
}

Sum
```

```
tuts@FossLinux:~/Desktop/scripts$ bash function.sh
Enter First Number: 10
Enter Second Number: 15
Sum is : 25
tuts@FossLinux:~/Desktop/scripts$
```

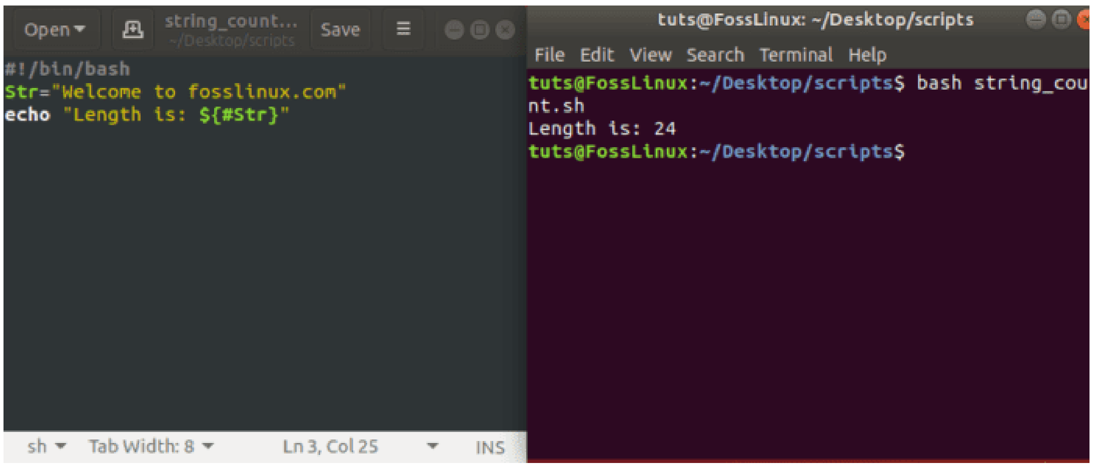
function example

10. Display String Length

Processing strings is an essential part of bash scripting. Bash script has a straightforward way of getting a string variable's length. In the example, we will show you how to get the length of a string in bash script.

```
#!/bin/bash
Str="Welcome to fosslinux.com"
echo "Length is: ${#Str}"
```

Output:



```
#!/bin/bash
Str="Welcome to fosslinux.com"
echo "Length is: ${#Str}"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash string_cou
nt.sh
Length is: 24
tuts@FossLinux:~/Desktop/scripts$
```

string length example

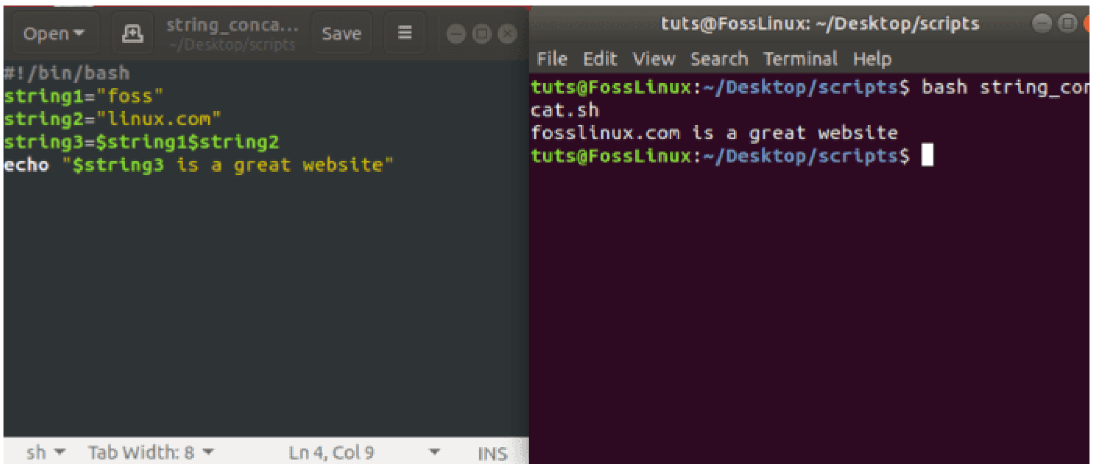
11. Concatenating Strings

Bash script provides an effortless way to handle string operations like the concatenation of multiple strings into a single string. In the example, we will show you how to do that.

```
#!/bin/bash

string1="foss"
string2="linux.com"
string=$string1$string2
echo "$string is a great website."
```

Output:



Open
string_conca...
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 4, Col 9
INS

```
#!/bin/bash
string1="foss"
string2="linux.com"
string3=$string1$string2
echo "$string3 is a great website"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash string_concat.sh
fosslinux.com is a great website
tuts@FossLinux:~/Desktop/scripts$
```

string concatenation example

12. Extracting String

Bash gives a way to extract a substring from a string. The below example explains how to parse n characters starting from a particular position.

```
${string:position}
```

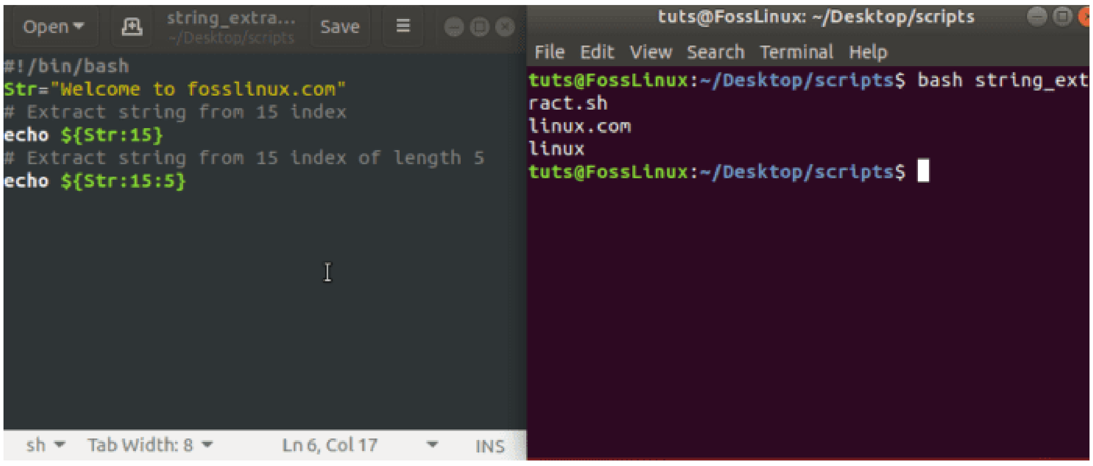
Extract substring from $string at $position

```
${string:position:length}
```

Extract $length of characters sub-string from $string starting from $position. In the below example, we will show you how it works.

```
#!/bin/bash
Str="Welcome to the fosslinux.com"
# Extracting string from index 15
echo ${Str:15}
# Extracting string from index 15 of length 5
echo ${Str:15:5}
```

Output:



```
#!/bin/bash
Str="Welcome to fosslinux.com"
# Extract string from 15 index
echo ${Str:15}
# Extract string from 15 index of length 5
echo ${Str:15:5}
```

```
tuts@FossLinux:~/Desktop/scripts$ bash string_ext
ract.sh
linux.com
linux
tuts@FossLinux:~/Desktop/scripts$
```

string extract example

13. Find and Replace String

Bash script has a handy and easy way to find and replace the text within a string. It can be used in two ways:

```
${string/pattern/replacement}
```

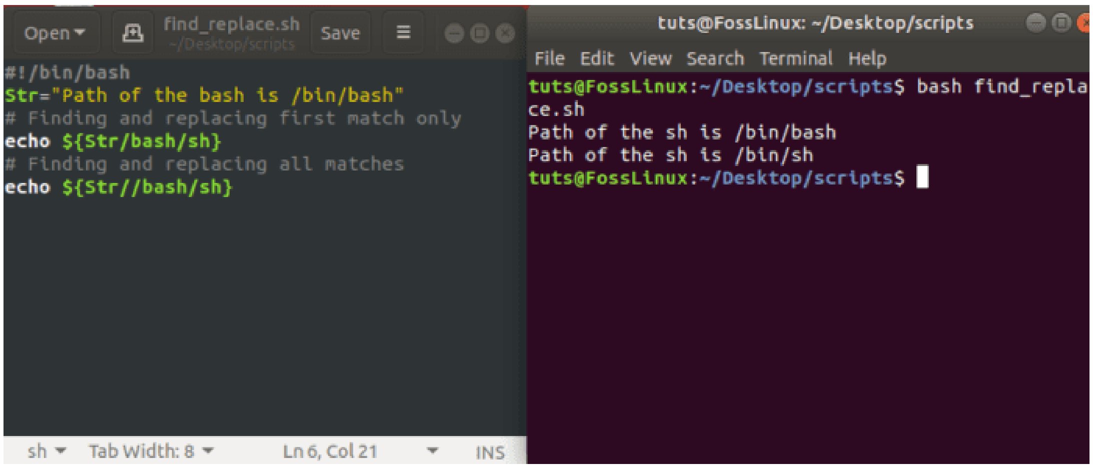
This will replace only the first match within the given string. To replace all matches, we will use it as shown below:

```
${string//pattern/replacement}
```

In another example, we will use both options to show you the difference in the output:

```
#! /bin/bash
Str="Path of the bash is /bin/bash"
# Finding and Replacing First match only
echo ${filename/bash/sh}
# Finding and Replacing all matches
echo ${filename//bash/sh}
```

Output:



```
#!/bin/bash
Str="Path of the bash is /bin/bash"
# Finding and replacing first match only
echo ${Str/bash/sh}
# Finding and replacing all matches
echo ${Str//bash/sh}
```

```
tuts@FossLinux:~/Desktop/scripts$ bash find_repla
ce.sh
Path of the sh is /bin/bash
Path of the sh is /bin/sh
tuts@FossLinux:~/Desktop/scripts$
```

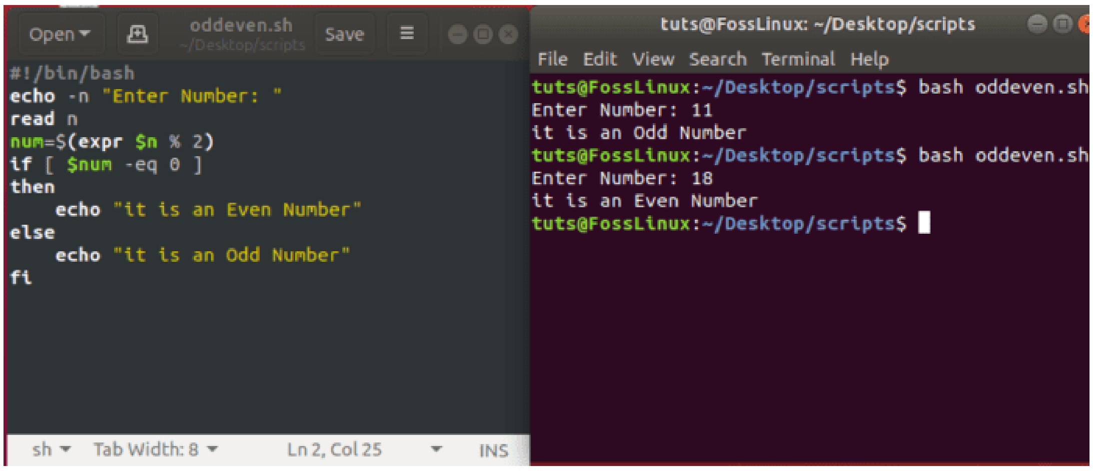
find and replace the example

14. Check Even/Odd Number

In our next example, we will write a bash script that will accept an input number from the user and will display if a given number is an even number or odd number.

```
#!/bin/bash
echo -n "Enter The Number: "
read n
num=$(expr $n % 2)
if [ $num -eq 0 ]; then
        echo "It is a Even Number"
else
        echo "It is a Odd Number"
fi
```

Output:



```
#!/bin/bash
echo -n "Enter Number: "
read n
num=$(expr $n % 2)
if [ $num -eq 0 ]
then
    echo "it is an Even Number"
else
    echo "it is an Odd Number"
fi
```

```
tuts@FossLinux:~/Desktop/scripts$ bash oddeven.sh
Enter Number: 11
it is an Odd Number
tuts@FossLinux:~/Desktop/scripts$ bash oddeven.sh
Enter Number: 18
it is an Even Number
tuts@FossLinux:~/Desktop/scripts$
```

Open
oddeven.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 2, Col 25
INS

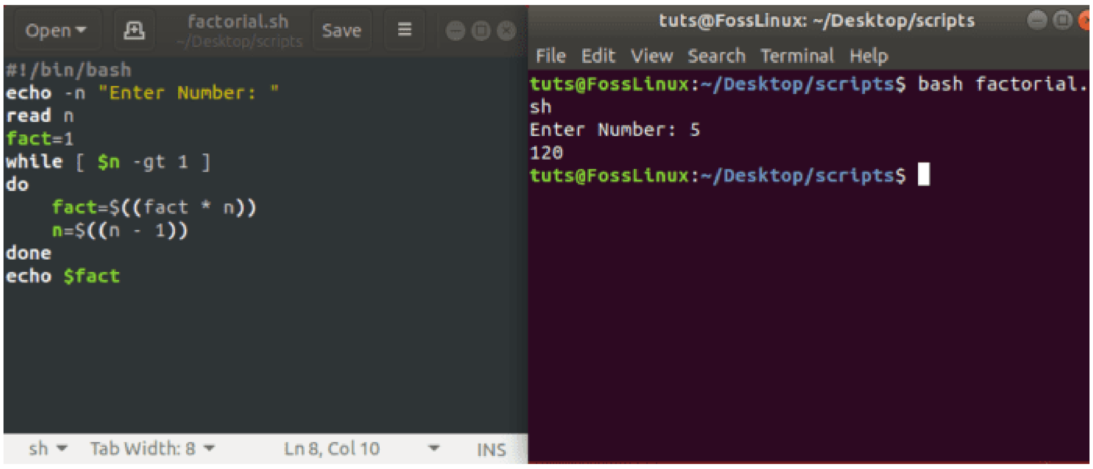
even odd number example

15. Generate Factorial of Number

The following example will show you how to generate a factorial of a given number using a shell script.

```
#!/bin/bash
echo -n "Enter Number: "
read n
fact=1
while [ $n -gt 1 ]
do
        fact=$((fact *  n))
        n=$((n - 1))
done
echo $fact
```

Output:


Open
factorial.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 8, Col 10
INS

```
#!/bin/bash
echo -n "Enter Number: "
read n
fact=1
while [ $n -gt 1 ]
do
    fact=$((fact * n))
    n=$((n - 1))
done
echo $fact
```

```
tuts@FossLinux:~/Desktop/scripts$ bash factorial.
sh
Enter Number: 5
120
tuts@FossLinux:~/Desktop/scripts$ 
```

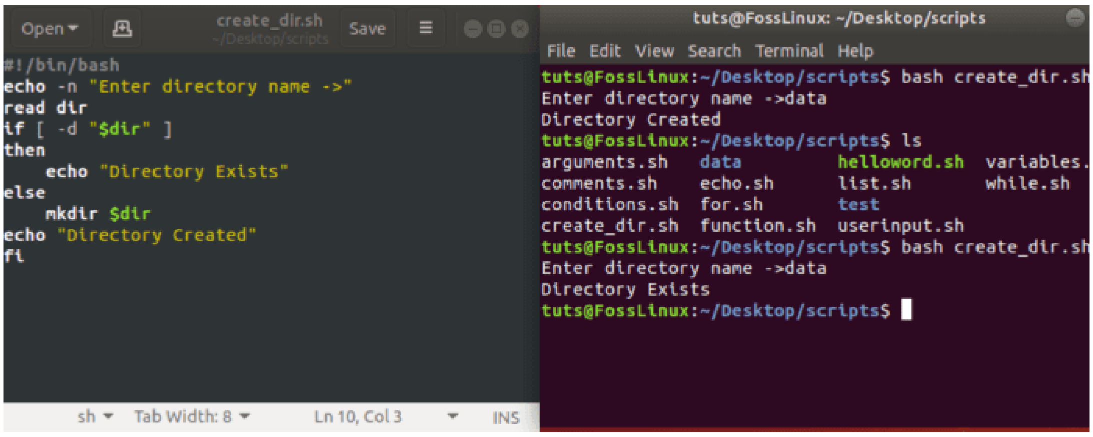
factorial example

16. Creating Directories

The following example will show you how to create a directory from within a shell script. The script will get the directory name from the user and will check if it already exists or not. In case it exists, you should see a message "Directory Exists"; otherwise, it will create a directory.

```
#!/bin/bash
echo -n "Enter directory name ->"
read dir
if [ -d "$dir" ]
then
echo "Directory exists"
else
`mkdir $dir`
echo "Directory created"
fi
```

Output:


Open
create_dir.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 10, Col 3
INS

```
#!/bin/bash
echo -n "Enter directory name ->"
read dir
if [ -d "$dir" ]
then
    echo "Directory Exists"
else
    mkdir $dir
echo "Directory Created"
fi
```

```
tuts@FossLinux:~/Desktop/scripts$ bash create_dir.sh
Enter directory name ->data
Directory Created
tuts@FossLinux:~/Desktop/scripts$ ls
arguments.sh   data          helloword.sh   variables.
comments.sh    echo.sh       list.sh        while.sh
conditions.sh  for.sh        test
create_dir.sh  function.sh   userinput.sh
tuts@FossLinux:~/Desktop/scripts$ bash create_dir.sh
Enter directory name ->data
Directory Exists
tuts@FossLinux:~/Desktop/scripts$ 
```

creating directory example

17. Reading Files

Using Bash you can read files very effectively. The below example will showcase how to read a file using shell scripts. Create a file called 'companies.txt' with the following contents.

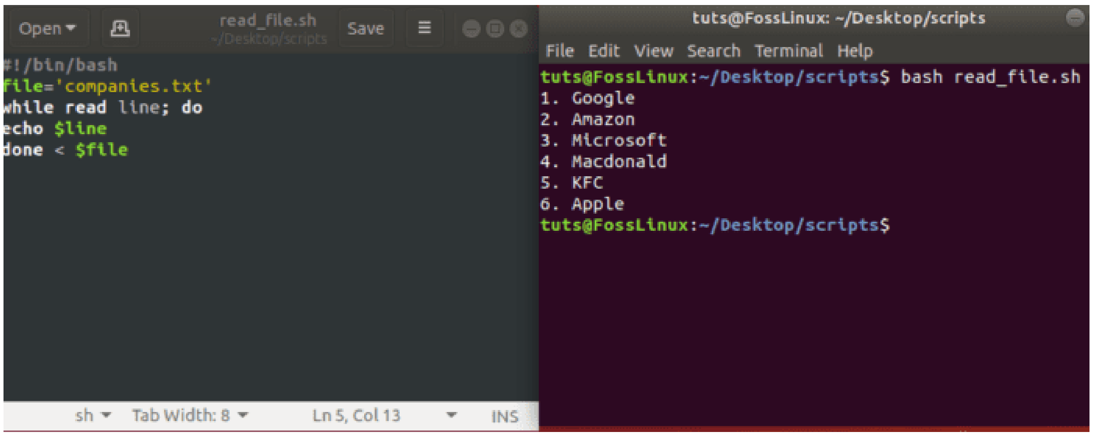
```
Google
Amazon
Microsoft
Macdonald
```

```
KFC
Apple
```

This script will read the above file and will display output.

```
#!/bin/bash
file='companies.txt'
while read line; do
echo $line
done < $file
```

Output:


Open
read_file.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 5, Col 13
INS

```
#!/bin/bash
file='companies.txt'
while read line; do
echo $line
done < $file
```

```
tuts@FossLinux:~/Desktop/scripts$ bash read_file.sh
1. Google
2. Amazon
3. Microsoft
4. Macdonald
5. KFC
6. Apple
tuts@FossLinux:~/Desktop/scripts$
```

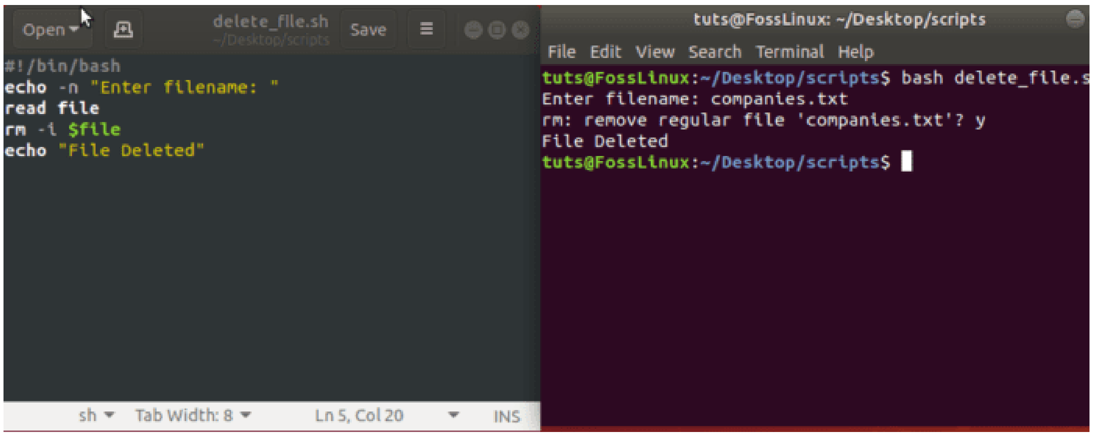
read file example

18. Deleting Files

Using a bash script, you can delete a file as well. In the example, the user will be asked to provide the filename as input and will delete it if it exists. It uses the Linux rm command for the deletion here.

```
#!/bin/bash
echo -n "Enter filename ->"
read name
rm -i $name
echo "File Deleted"
```

Output:


Open
delete_file.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 5, Col 20
INS

```
#!/bin/bash
echo -n "Enter filename: "
read file
rm -i $file
echo "File Deleted"
```

```
tuts@FossLinux:~/Desktop/scripts$ bash delete_file.s
Enter filename: companies.txt
rm: remove regular file 'companies.txt'? y
File Deleted
tuts@FossLinux:~/Desktop/scripts$
```

delete file example

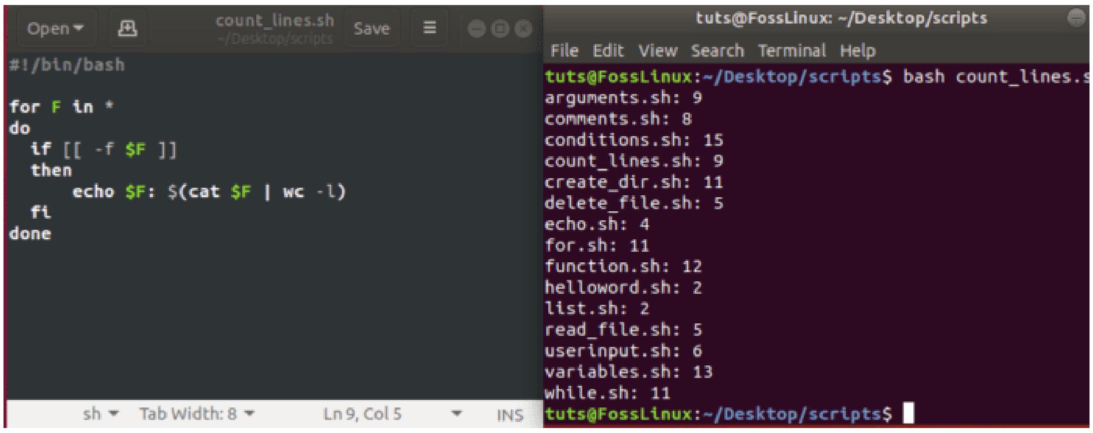
19. Print Files With Line Count

In our example, we shall write a bash script that will print all files with there line count in the current directory.

```
#!/usr/bin/env bash
for F in *
do
if [[ -f $F ]]
then
echo $F: $(cat $F | wc -l)
fi
done
```

You can see that we used a for loop to get the file and then used the cat command to count lines.

Output:


Open
count_lines.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 9, Col 5
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash

for F in *
do
  if [[ -f $F ]]
  then
      echo $F: $(cat $F | wc -l)
  fi
done
```

```
tuts@FossLinux:~/Desktop/scripts$ bash count_lines.s
arguments.sh: 9
comments.sh: 8
conditions.sh: 15
count_lines.sh: 9
create_dir.sh: 11
delete_file.sh: 5
echo.sh: 4
for.sh: 11
function.sh: 12
helloword.sh: 2
list.sh: 2
read_file.sh: 5
userinput.sh: 6
variables.sh: 13
while.sh: 11
tuts@FossLinux:~/Desktop/scripts$
```

File list with line count example

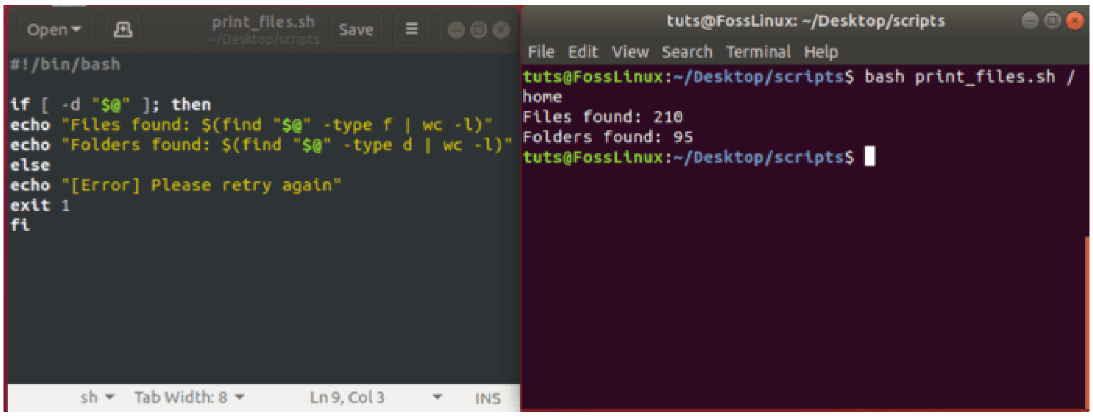
20. Print Number of Files and Folders

In our next example, the Linux bash script finds the number of files or folders present inside a given directory. It uses the Linux 'find' command. Users will be asked to input the directory name where you want to search for files from the command-line.

```
#!/bin/bash

if [ -d "$@" ]; then
echo "Files found: $(find "$@" -type f | wc -l)"
echo "Folders found: $(find "$@" -type d | wc -l)"
else
echo "[ERROR] Please try again."
exit 1
fi
```

Output:



```
#!/bin/bash

if [ -d "$@" ]; then
echo "Files found: $(find "$@" -type f | wc -l)"
echo "Folders found: $(find "$@" -type d | wc -l)"
else
echo "[Error] Please retry again"
exit 1
fi
```

```
tuts@FossLinux:~/Desktop/scripts$ bash print_files.sh /
home
Files found: 210
Folders found: 95
tuts@FossLinux:~/Desktop/scripts$
```

print files and folders count example

21. Check if User is Root

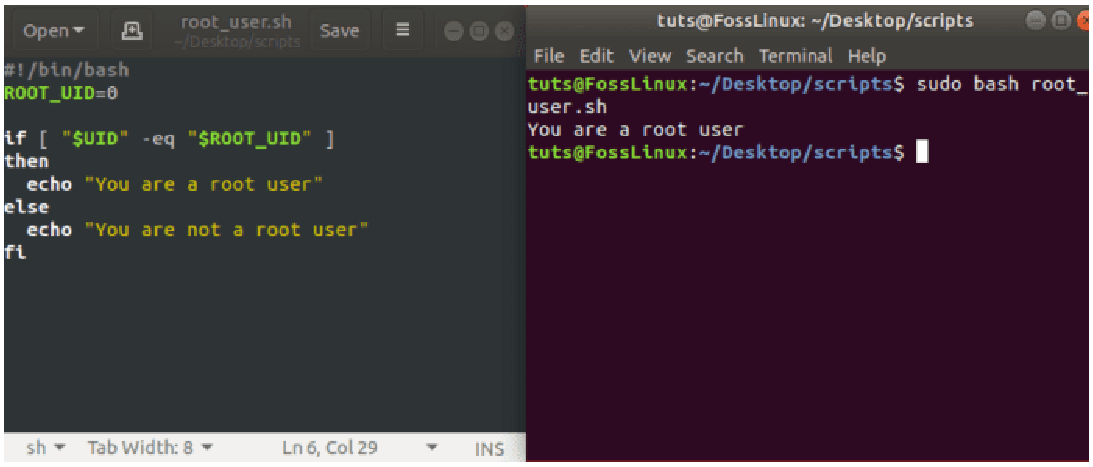
This example demonstrates a quick way to find out whether a user is a root or not from Linux bash scripts.

```
#!/bin/bash
ROOT_UID=0

if [ "$UID" -eq "$ROOT_UID" ]
then
  echo "You are a root user."
else
  echo "You are not a root user"
fi
```

You have to run the bash script with sudo command.

Output:



```
#!/bin/bash
ROOT_UID=0

if [ "$UID" -eq "$ROOT_UID" ]
then
  echo "You are a root user"
else
  echo "You are not a root user"
fi
```

```
tuts@FossLinux:~/Desktop/scripts$ sudo bash root_
user.sh
You are a root user
tuts@FossLinux:~/Desktop/scripts$
```

checking if root user example

22. Send Mail using Bash

You can also send emails from bash scripts. The following simple example will demonstrate one way of doing it from bash applications.

```
#!/bin/bash
recipient=”admin@fosslinux.com”
subject=”Greetings”
```

```
message=”Welcome to Fosslinux”
`mail -s $subject $recipient <<< $message`
```

It will send an email to the recipient containing the given subject and message.

23. Cleaning Log Files

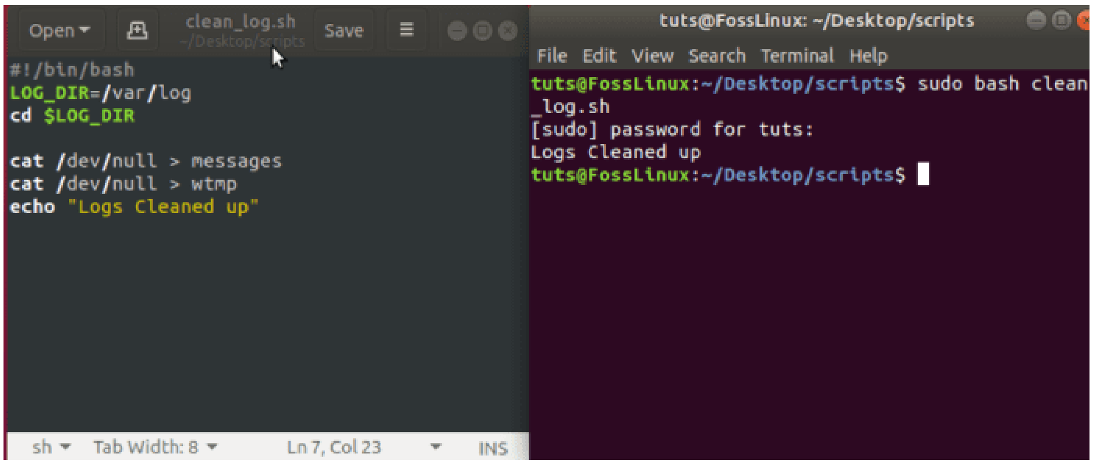
The program will delete all log files present inside your /var/log directory. You can add more variables to holds other log directories and clean them as well.

```
#!/bin/bash
LOG_DIR=/var/log
cd $LOG_DIR

cat /dev/null > messages
cat /dev/null > wtmp
echo "Logs cleaned up."
```

Please remember you need root privileges to run this bash script.

Output:


Open
clean_log.sh
~/Desktop/scripts
Save
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help
sh
Tab Width: 8
Ln 7, Col 23
INS

```
#!/bin/bash
LOG_DIR=/var/log
cd $LOG_DIR

cat /dev/null > messages
cat /dev/null > wtmp
echo "Logs Cleaned up"
```

```
tuts@FossLinux:~/Desktop/scripts$ sudo bash clean_log.sh
[sudo] password for tuts:
Logs Cleaned up
tuts@FossLinux:~/Desktop/scripts$
```

log cleaning example

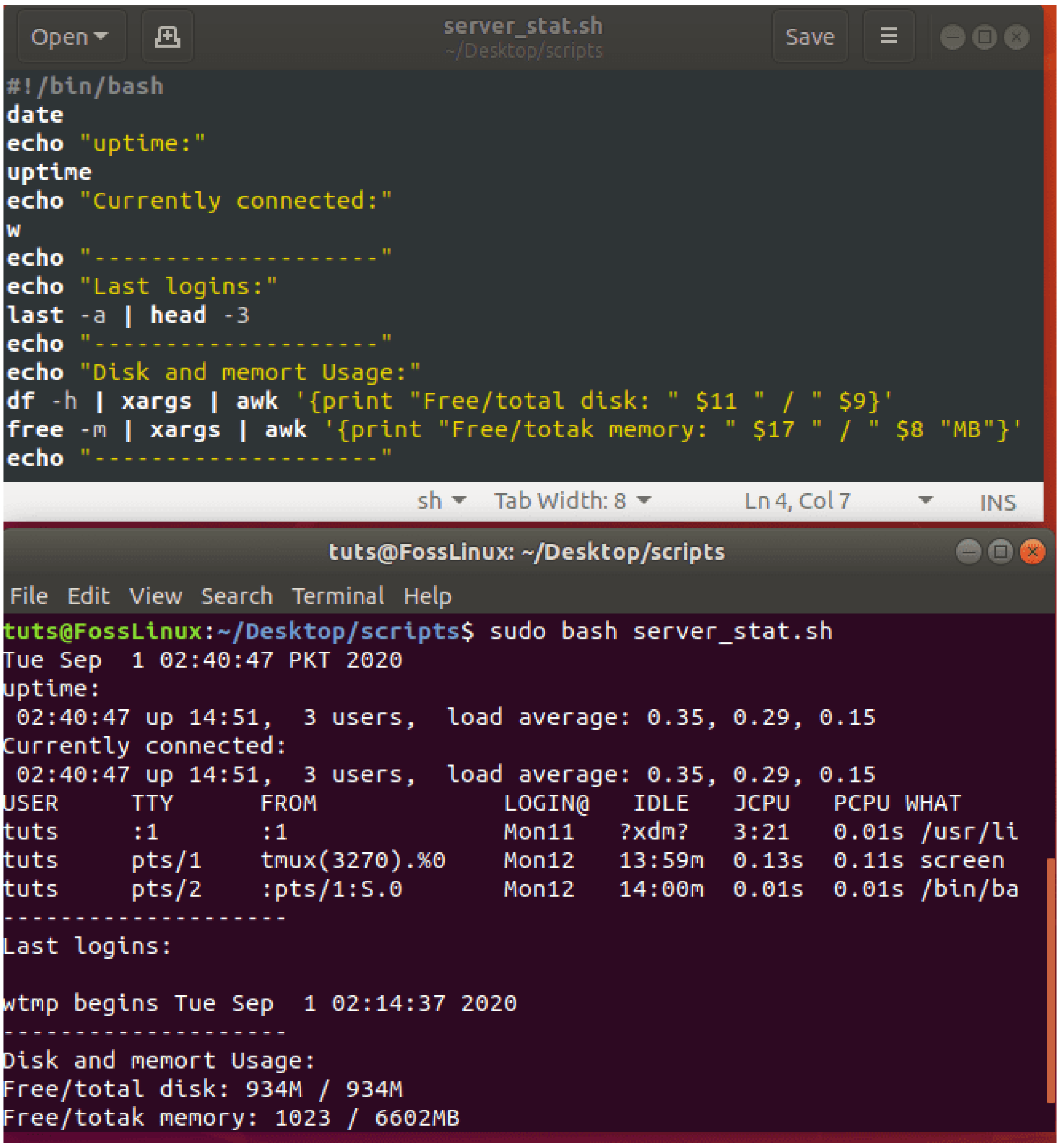
24. Display Server Stats

This example will show you a quick server stats. As a system administrator, this bash script will help you get important details like uptime, last logins, disk, and memory usage for a Linux machine.

```
#!/bin/bash
date
echo "uptime:"
uptime
echo "Currently connected:"
w
echo "--------------------"
echo "Last logins:"
last -a | head -3
echo "--------------------"
echo "Disk and memory usage:"
df -h | xargs | awk '{print "Free/total disk: " $11 " / " $9}'
free -m | xargs | awk '{print "Free/total memory: " $17 " / " $8 " MB"}'
echo "--------------------"
```

You need to run the script as a root user.

Output:


Open
server_stat.sh
~/Desktop/scripts
Save
sh
Tab Width: 8
Ln 4, Col 7
INS
tuts@FossLinux: ~/Desktop/scripts
File Edit View Search Terminal Help

```
#!/bin/bash
date
echo "uptime:"
uptime
echo "Currently connected:"
w
echo "--------------------"
echo "Last logins:"
last -a | head -3
echo "--------------------"
echo "Disk and memort Usage:"
df -h | xargs | awk '{print "Free/total disk: " $11 " / " $9}'
free -m | xargs | awk '{print "Free/totak memory: " $17 " / " $8 "MB"}'
echo "--------------------"
```

```
tuts@FossLinux:~/Desktop/scripts$ sudo bash server_stat.sh
Tue Sep  1 02:40:47 PKT 2020
uptime:
 02:40:47 up 14:51,  3 users,  load average: 0.35, 0.29, 0.15
Currently connected:
 02:40:47 up 14:51,  3 users,  load average: 0.35, 0.29, 0.15
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
tuts     :1       :1               Mon11   ?xdm?   3:21   0.01s /usr/li
tuts     pts/1    tmux(3270).%0    Mon12   13:59m  0.13s  0.11s screen
tuts     pts/2    :pts/1:S.0       Mon12   14:00m  0.01s  0.01s /bin/ba
--------------------
Last logins:

wtmp begins Tue Sep  1 02:14:37 2020
--------------------
Disk and memort Usage:
Free/total disk: 934M / 934M
Free/totak memory: 1023 / 6602MB
```

server stats example

25. System Maintenance

This little Linux shell script upgrades and cleans the system automatically instead of doing it manually.

```
#!/bin/bash

echo -e "\n$(date "+%d-%m-%Y --- %T") --- Starting work\n"

apt-get update
apt-get -y upgrade

apt-get -y autoremove
apt-get autoclean

echo -e "\n$(date "+%T") \t Script Terminated"
```

You need to run this script as a root user.

Conclusion

Linux shell scripts can be handy. Complex tasks appropriately executed can increase your productivity to a great extent and also help you troubleshoot issues in no time. Further, there is no limit on its scalability. If you're a new Linux enthusiast, we highly recommend you to master these bash script examples.