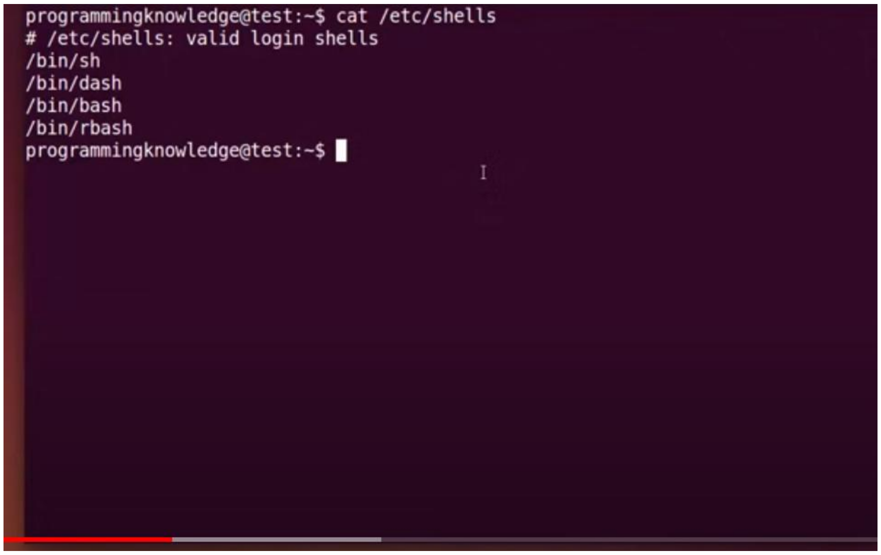
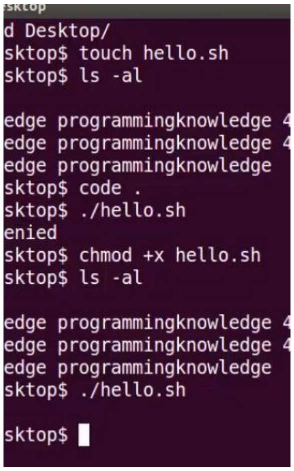
How to create and run a shell script



```
programmingknowledge@test:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
programmingknowledge@test:~$ 
```



```
esktop
d Desktop/
sktop$ touch hello.sh
sktop$ ls -al

edge programmingknowledge 4
edge programmingknowledge 4
edge programmingknowledge
sktop$ code .
sktop$ ./hello.sh
enied
sktop$ chmod +x hello.sh
sktop$ ls -al

edge programmingknowledge 4
edge programmingknowledge 4
edge programmingknowledge
sktop$ ./hello.sh

sktop$ 
```

- #!/bin/bash: This is the shebang line. It tells the system to use the Bash interpreter to execute the script.
- The echo command is used to display text or variables on the terminal. It’s commonly used for printing messages, variable values, and generating program output.
- You can add comments using the # symbol.
- Many people use multi-line comments to document their shell scripts. Check how this is done in the next script called comment.sh.

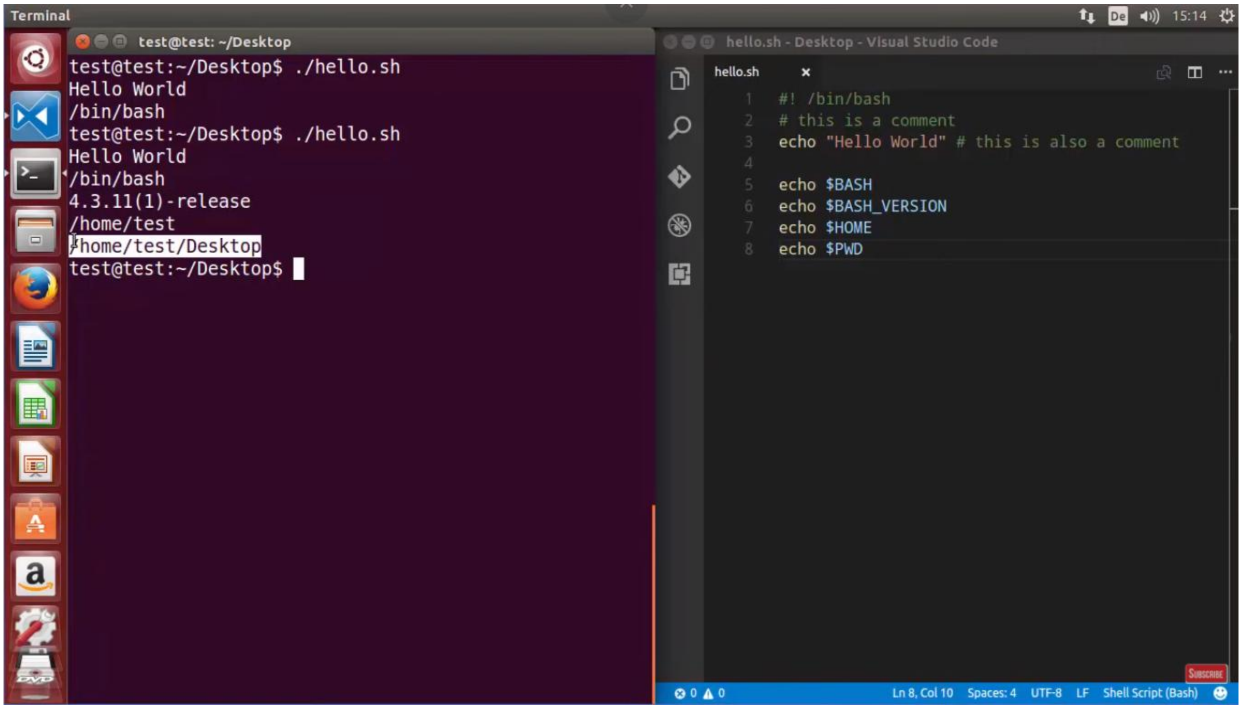
```
#!/bin/bash
: '
This script calculates
the square of 5.
'
((area=5*5))
echo $area
```

Single quotes (‘) and double quotes (“) are used to enclose strings in shell scripting, but they have different behaviors:

- Single quotes: Everything between single quotes is treated as a literal string. Variable names and most special characters are not expanded.
- Double quotes: Variables and certain special characters within double quotes are expanded. The contents are subject to variable substitution and command substitution.

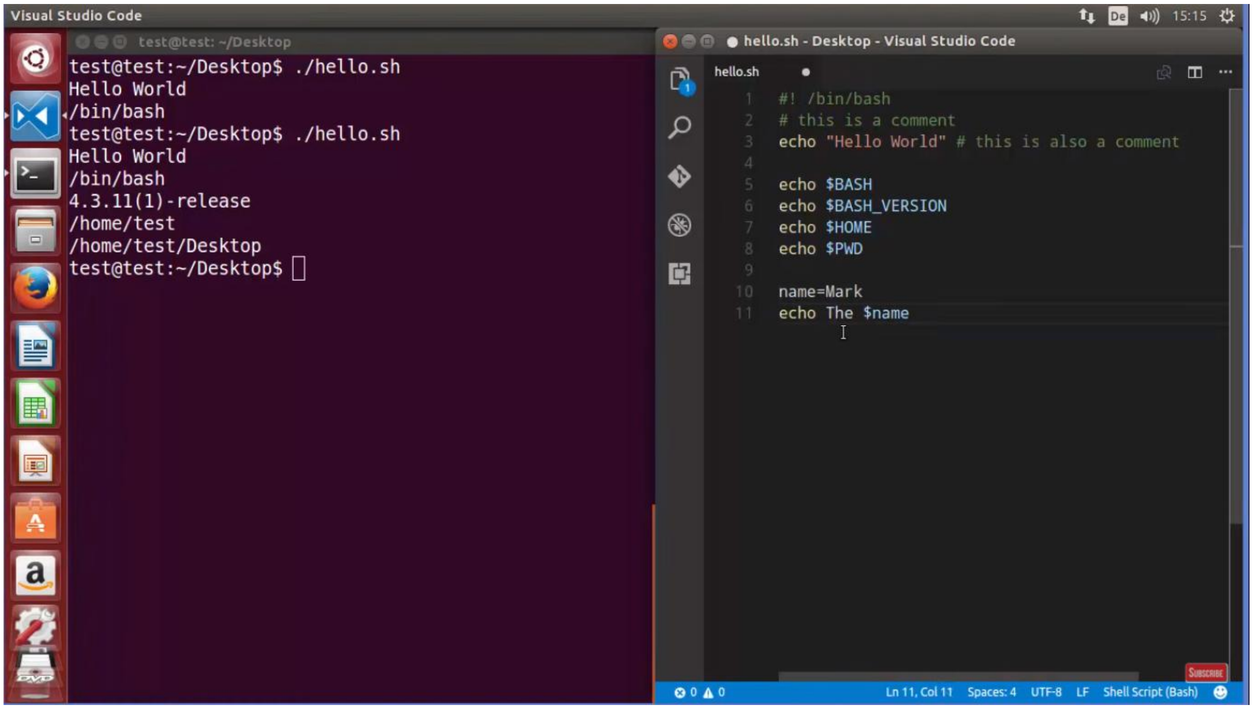
  Example:

  ```
  #!/bin/bash
  abcd=”Hello”
  echo ‘$abcd’ # Output: $abcd
  echo “$abcd” # Output: Hello
  ```



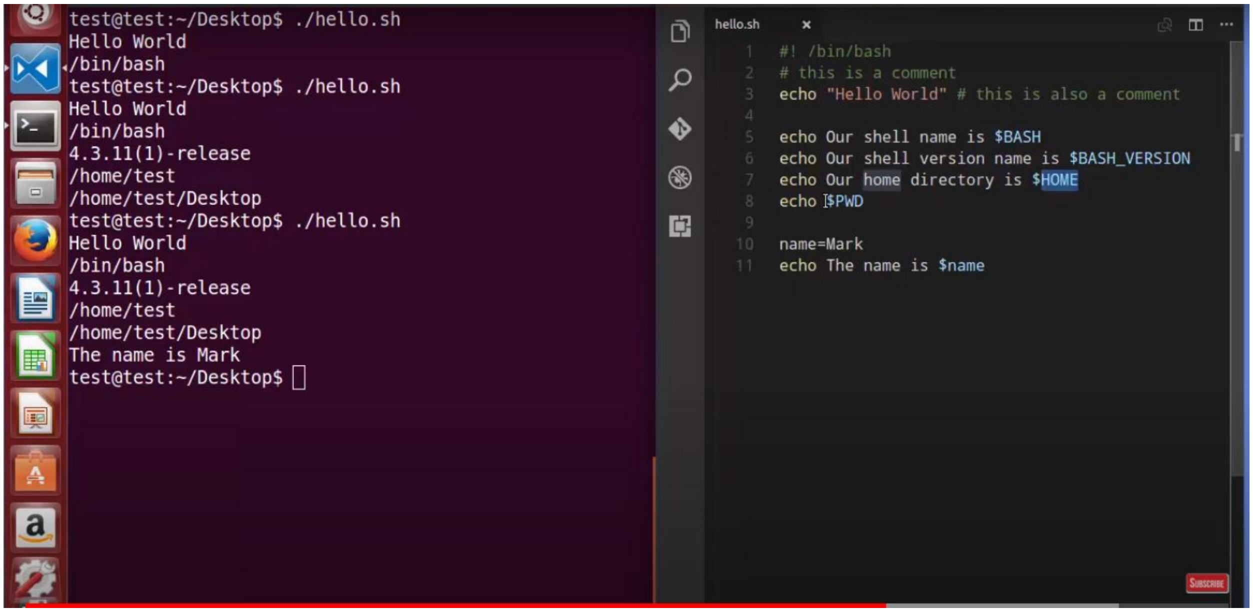
```
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
4.3.11(1)-release
/home/test
/home/test/Desktop
test@test:~/Desktop$ 
```

```
#! /bin/bash
# this is a comment
echo "Hello World" # this is also a comment

echo $BASH
echo $BASH_VERSION
echo $HOME
echo $PWD
```



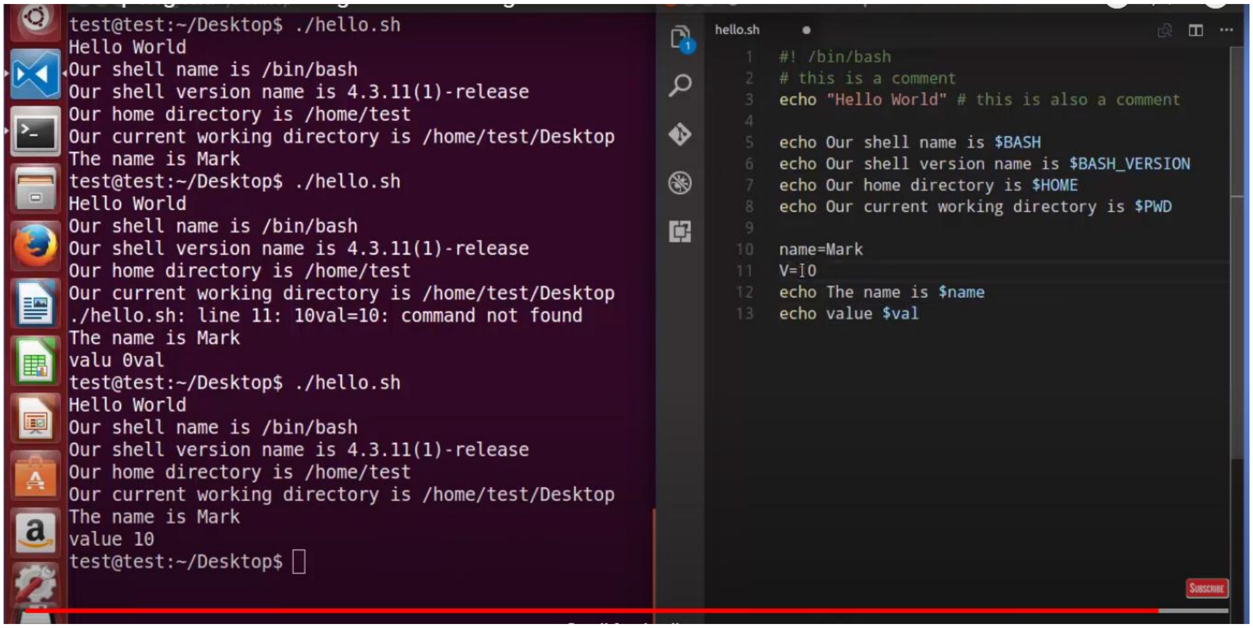
```
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
4.3.11(1)-release
/home/test
/home/test/Desktop
test@test:~/Desktop$ 
```

```
#! /bin/bash
# this is a comment
echo "Hello World" # this is also a comment

echo $BASH
echo $BASH_VERSION
echo $HOME
echo $PWD

name=Mark
echo The $name
```



```
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
4.3.11(1)-release
/home/test
/home/test/Desktop
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
4.3.11(1)-release
/home/test
/home/test/Desktop
The name is Mark
test@test:~/Desktop$ 
```

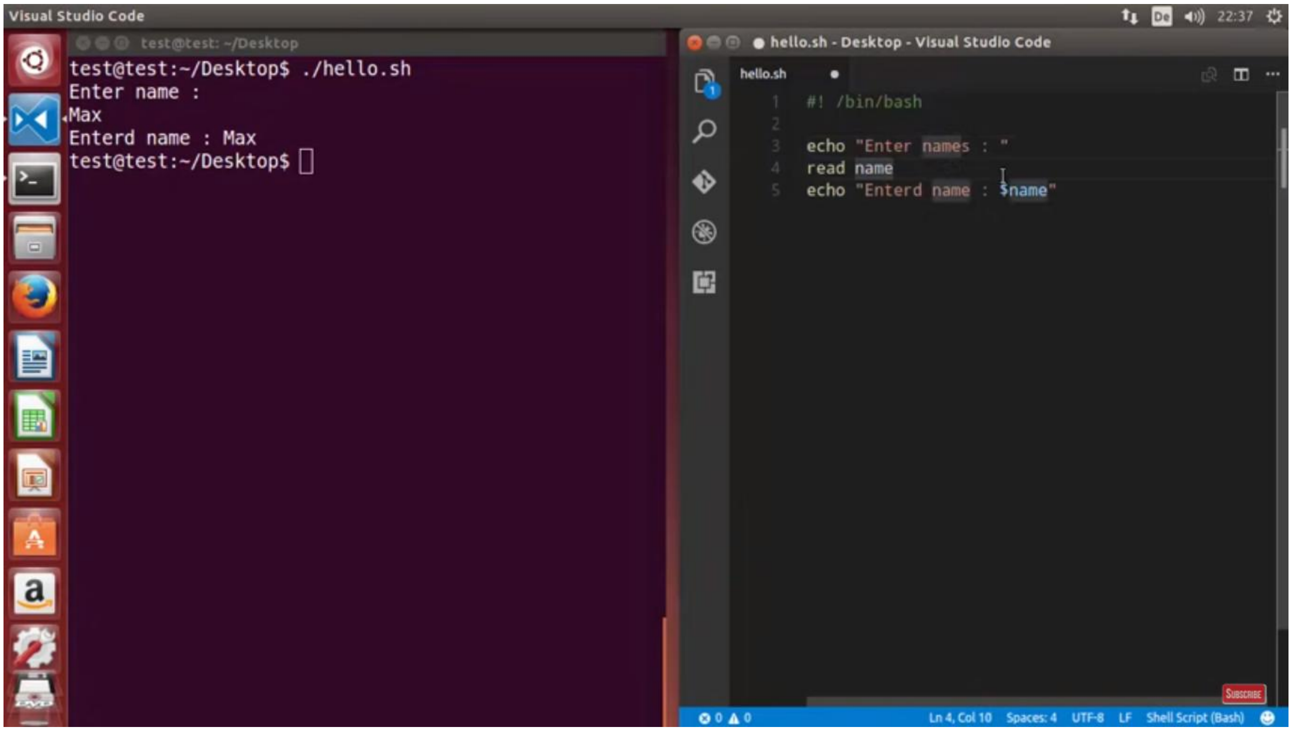
```
#! /bin/bash
# this is a comment
echo "Hello World" # this is also a comment

echo Our shell name is $BASH
echo Our shell version name is $BASH_VERSION
echo Our home directory is $HOME
echo $PWD

name=Mark
echo The name is $name
```



```
test@test:~/Desktop$ ./hello.sh
Hello World
Our shell name is /bin/bash
Our shell version name is 4.3.11(1)-release
Our home directory is /home/test
Our current working directory is /home/test/Desktop
The name is Mark
test@test:~/Desktop$ ./hello.sh
Hello World
Our shell name is /bin/bash
Our shell version name is 4.3.11(1)-release
Our home directory is /home/test
Our current working directory is /home/test/Desktop
./hello.sh: line 11: 10val=10: command not found
The name is Mark
valu 0val
test@test:~/Desktop$ ./hello.sh
Hello World
Our shell name is /bin/bash
Our shell version name is 4.3.11(1)-release
Our home directory is /home/test
Our current working directory is /home/test/Desktop
The name is Mark
value 10
test@test:~/Desktop$ 
```

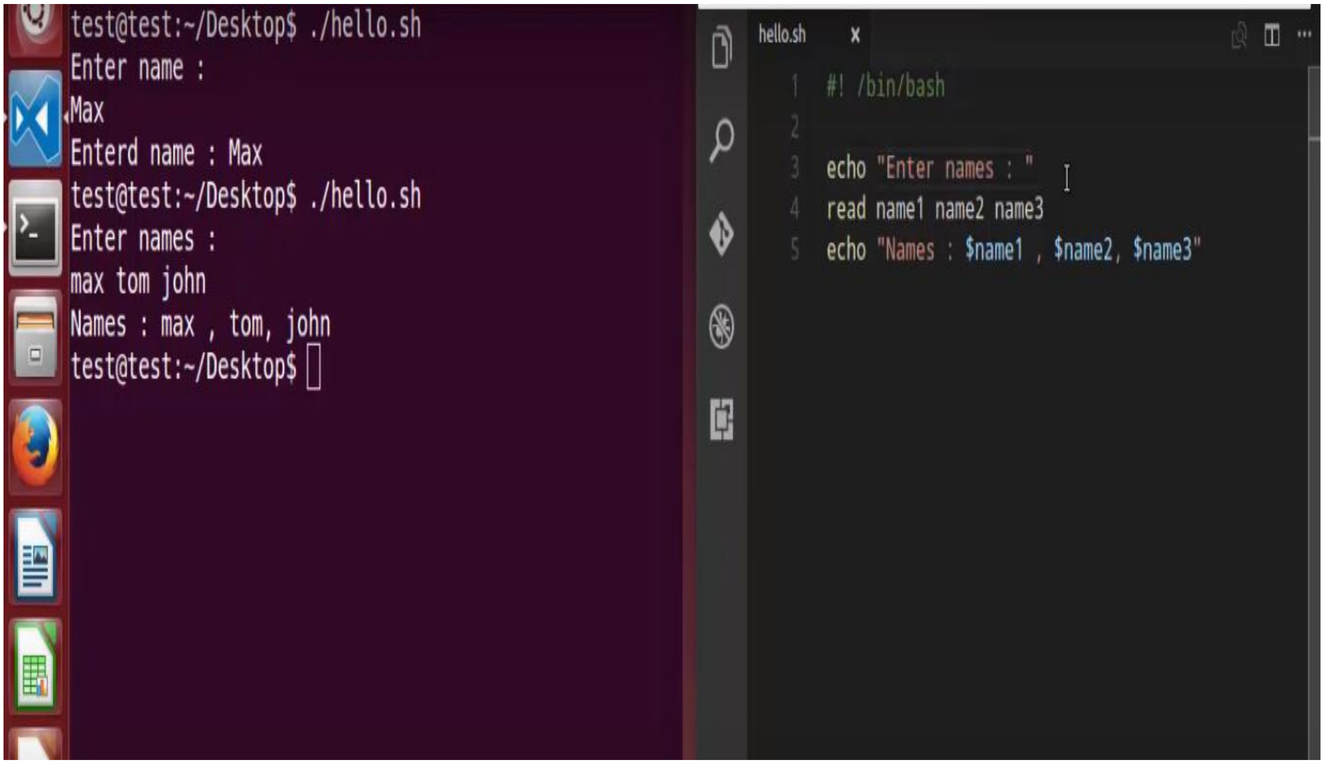
```
#! /bin/bash
# this is a comment
echo "Hello World" # this is also a comment

echo Our shell name is $BASH
echo Our shell version name is $BASH_VERSION
echo Our home directory is $HOME
echo Our current working directory is $PWD

name=Mark
V=10
echo The name is $name
echo value $val
```

Read User Input



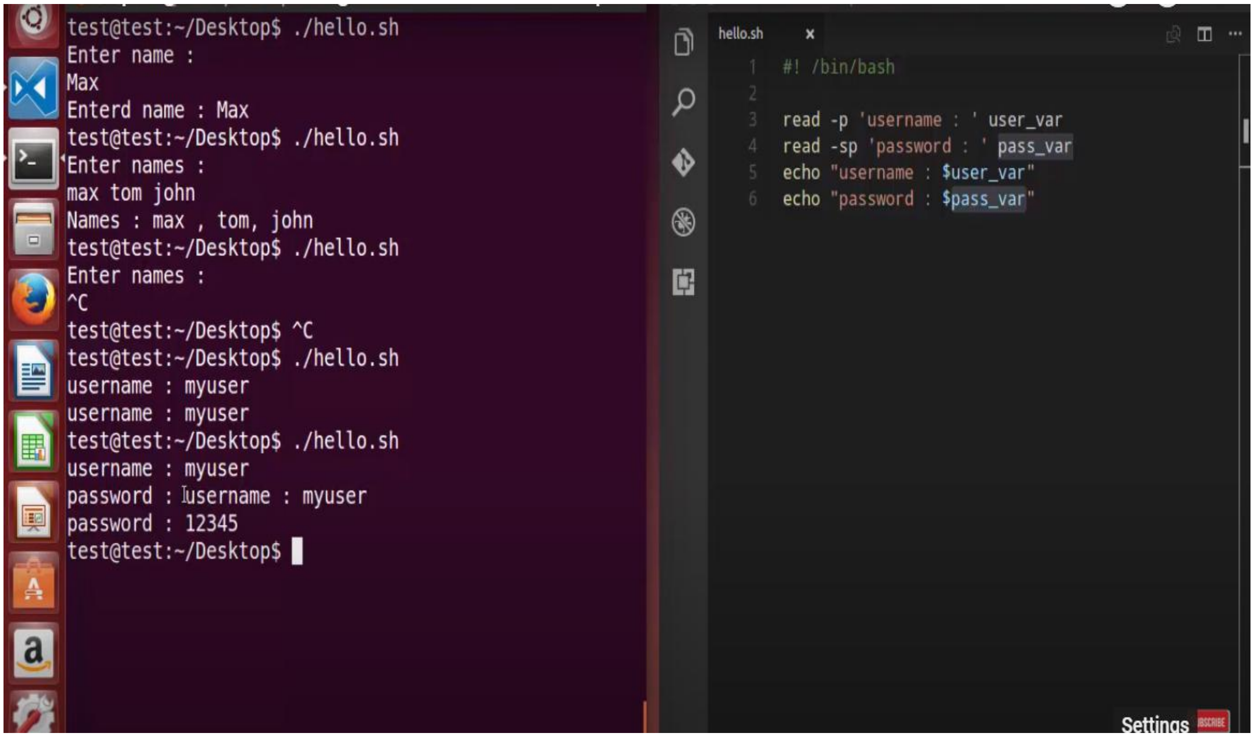
```
test@test:~/Desktop$ ./hello.sh
Enter name :
Max
Enterd name : Max
test@test:~/Desktop$ 
```

```bash
#! /bin/bash

echo "Enter names : "
read name
echo "Enterd name : $name"
```



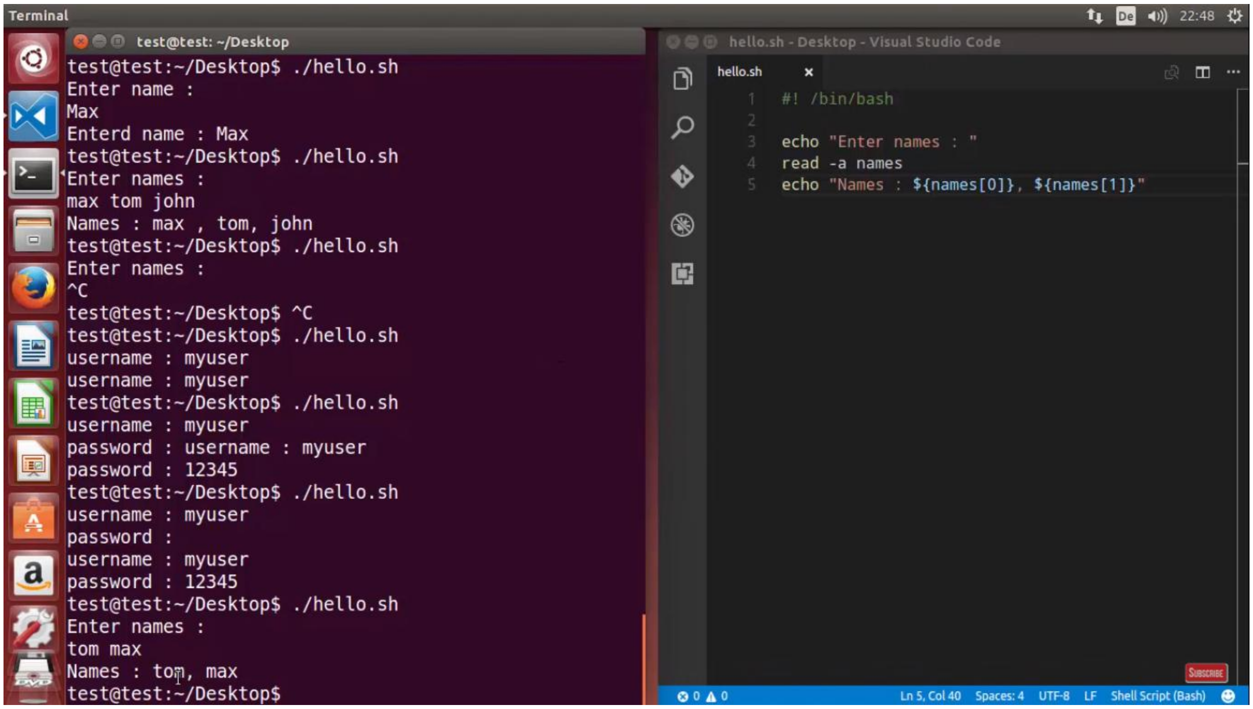
```
test@test:~/Desktop$ ./hello.sh
Enter name :
Max
Enterd name : Max
test@test:~/Desktop$ ./hello.sh
Enter names :
max tom john
Names : max , tom, john
test@test:~/Desktop$ 
```

```bash
#! /bin/bash

echo "Enter names : "
read name1 name2 name3
echo "Names : $name1 , $name2, $name3"
```



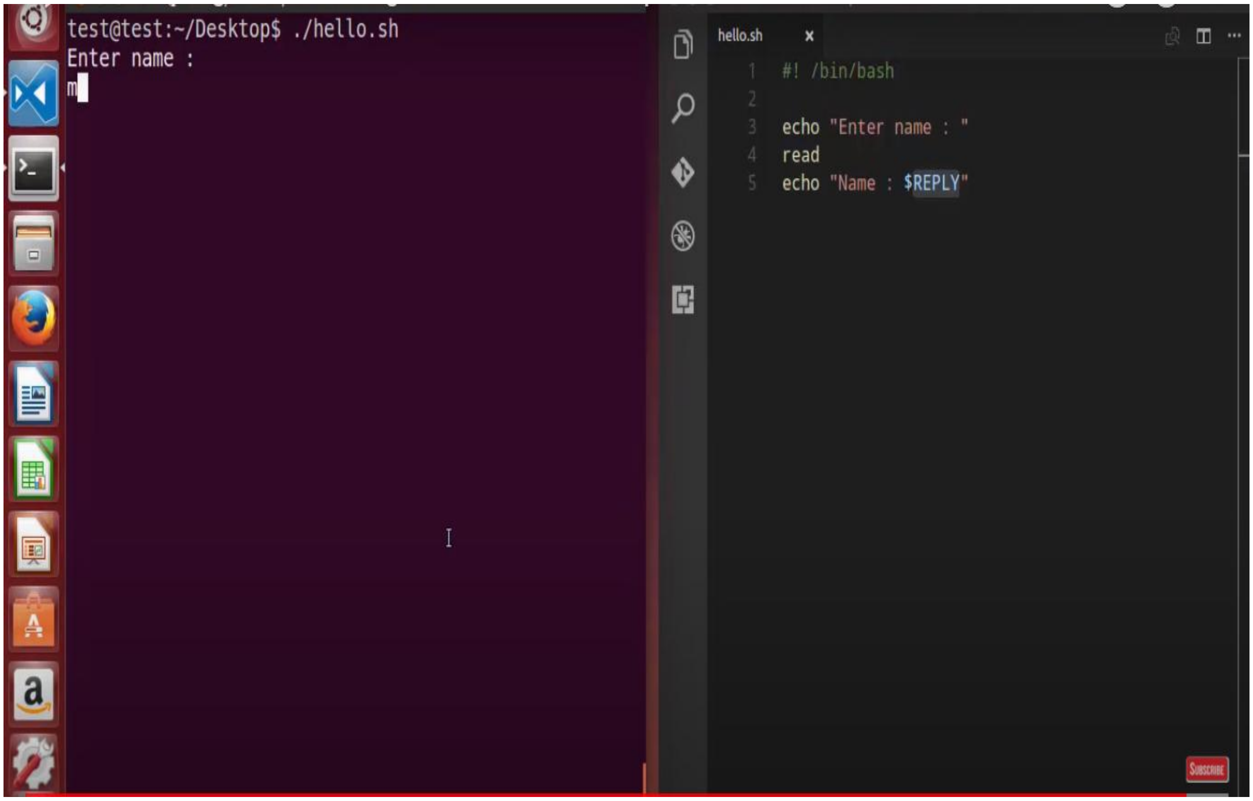
```
test@test:~/Desktop$ ./hello.sh
Enter name :
Max
Enterd name : Max
test@test:~/Desktop$ ./hello.sh
Enter names :
max tom john
Names : max , tom, john
test@test:~/Desktop$ ./hello.sh
Enter names :
^C
test@test:~/Desktop$ ^C
test@test:~/Desktop$ ./hello.sh
username : myuser
username : myuser
test@test:~/Desktop$ ./hello.sh
username : myuser
password : username : myuser
password : 12345
test@test:~/Desktop$
```

```bash
#! /bin/bash

read -p 'username : ' user_var
read -sp 'password : ' pass_var
echo "username : $user_var"
echo "password : $pass_var"
```



```
test@test:~/Desktop$ ./hello.sh
Enter name :
Max
Enterd name : Max
test@test:~/Desktop$ ./hello.sh
Enter names :
max tom john
Names : max , tom, john
test@test:~/Desktop$ ./hello.sh
Enter names :
^C
test@test:~/Desktop$ ^C
test@test:~/Desktop$ ./hello.sh
username : myuser
username : myuser
test@test:~/Desktop$ ./hello.sh
username : myuser
password : username : myuser
password : 12345
test@test:~/Desktop$ ./hello.sh
username : myuser
password :
username : myuser
password : 12345
test@test:~/Desktop$ ./hello.sh
Enter names :
tom max
Names : tom, max
test@test:~/Desktop$
```

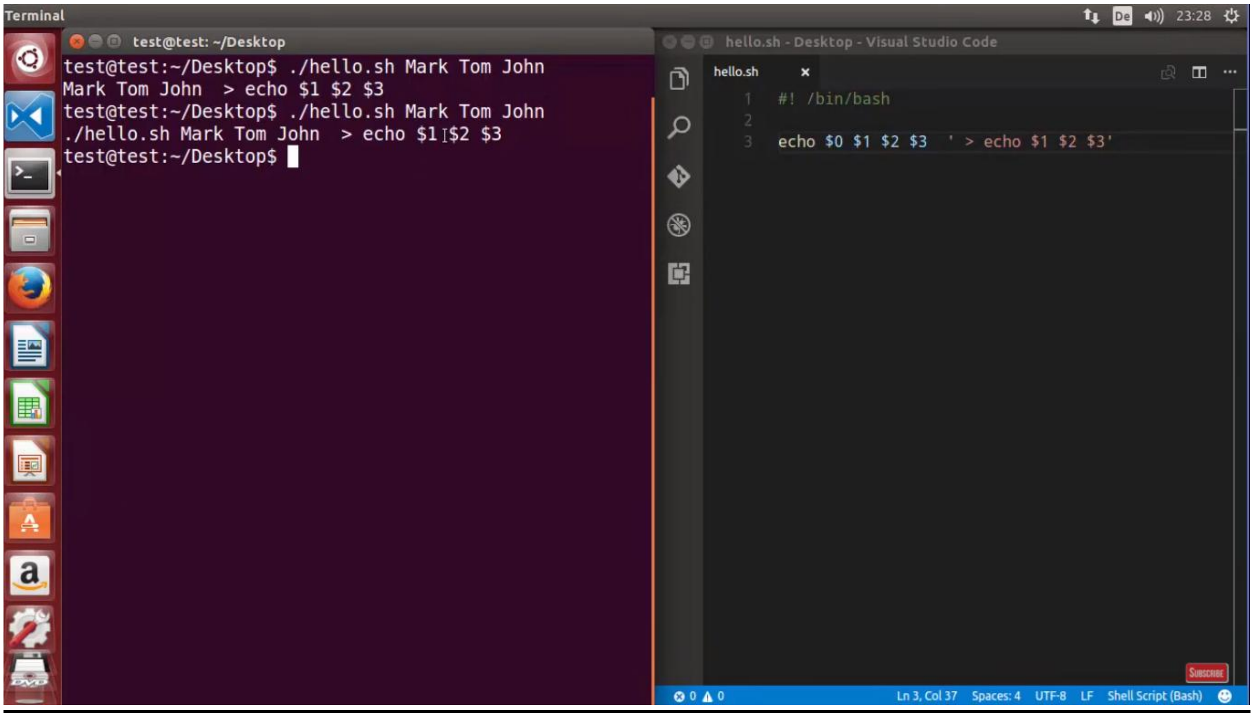
```bash
#! /bin/bash

echo "Enter names : "
read -a names
echo "Names : ${names[0]}, ${names[1]}"
```



```
test@test:~/Desktop$ ./hello.sh
Enter name :
m
```

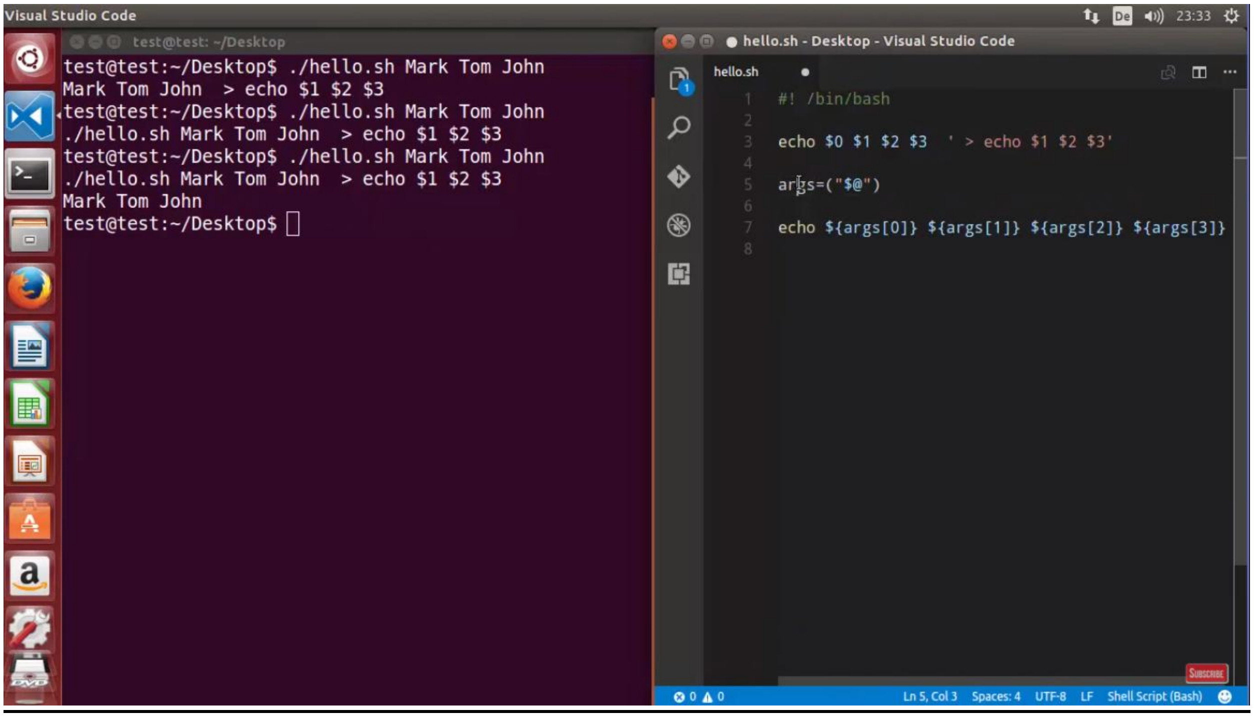
```
#! /bin/bash

echo "Enter name : "
read
echo "Name : $REPLY"
```

Passing Arguments to Bash Script



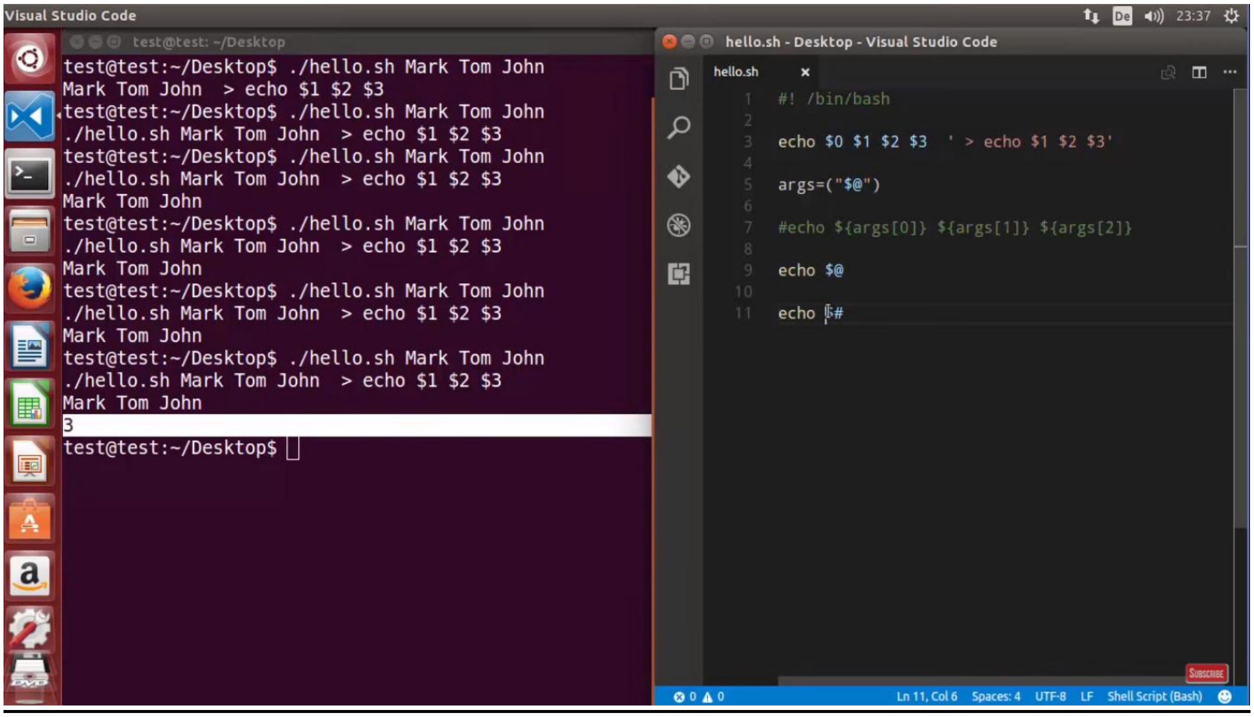
```
test@test:~/Desktop$ ./hello.sh Mark Tom John
Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$
```

```
#! /bin/bash

echo $0 $1 $2 $3  ' > echo $1 $2 $3'
```



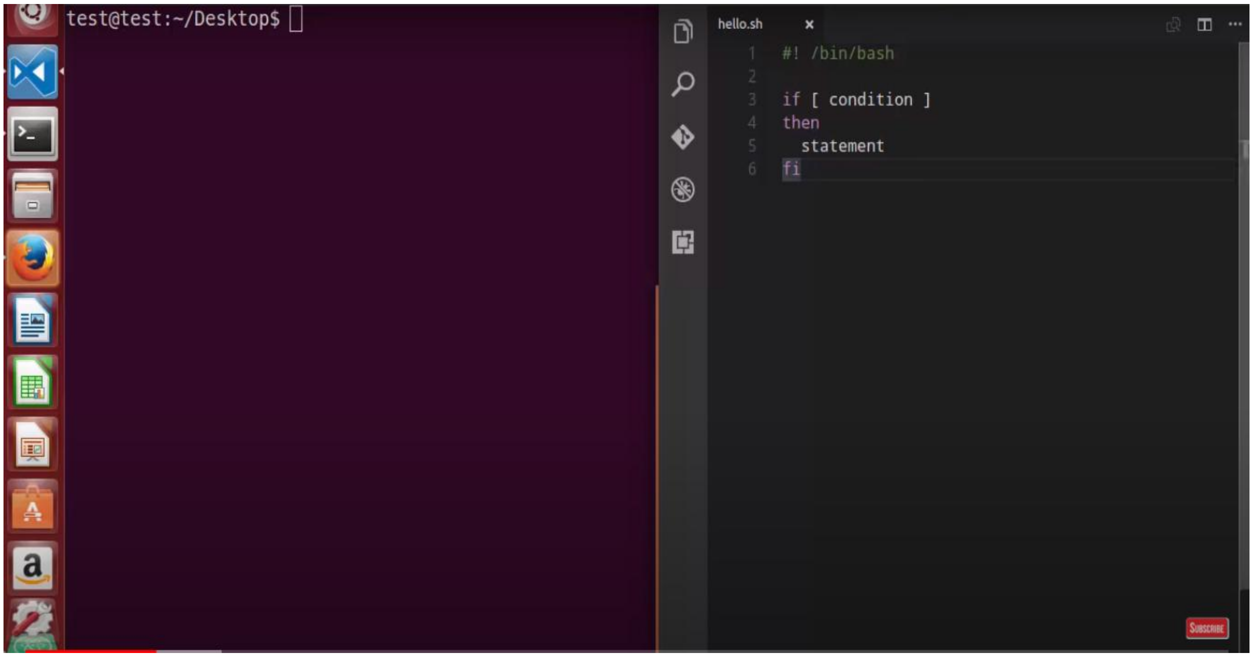
```
test@test:~/Desktop$ ./hello.sh Mark Tom John
Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
Mark Tom John
test@test:~/Desktop$ 
```

```
#! /bin/bash

echo $0 $1 $2 $3  ' > echo $1 $2 $3'

args=("$@")

echo ${args[0]} ${args[1]} ${args[2]} ${args[3]}

```



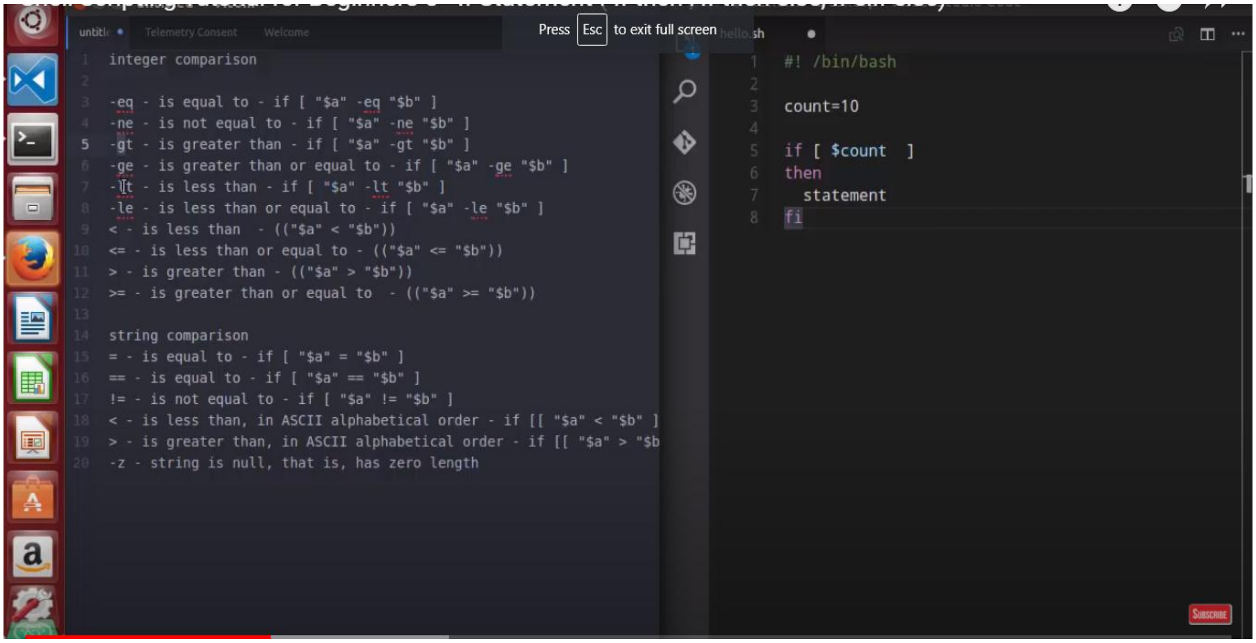
```
test@test:~/Desktop$ ./hello.sh Mark Tom John
Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
Mark Tom John
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
Mark Tom John
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
Mark Tom John
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John  > echo $1 $2 $3
Mark Tom John
3
test@test:~/Desktop$ 
```

```
#! /bin/bash

echo $0 $1 $2 $3  ' > echo $1 $2 $3'

args=("$@")

#echo ${args[0]} ${args[1]} ${args[2]}

echo $@

echo $#
```



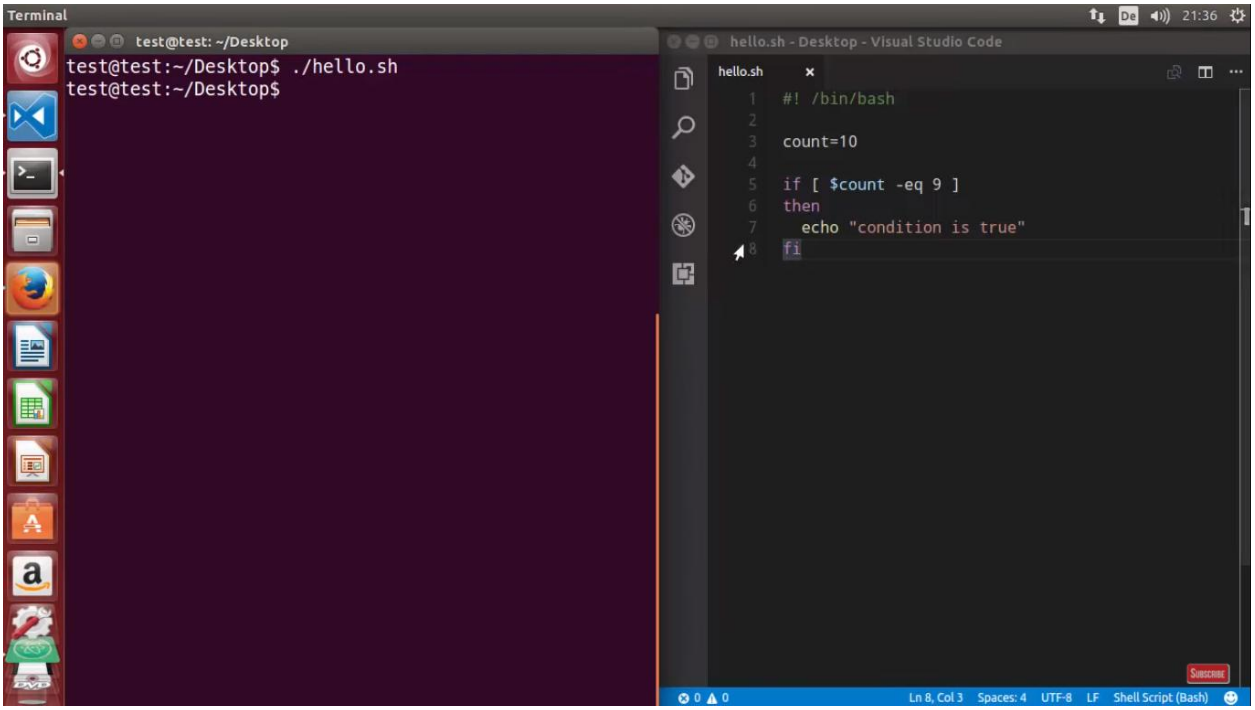
```
test@test:~/Desktop$
```

```
#! /bin/bash

if [ condition ]
then
  statement
fi
```



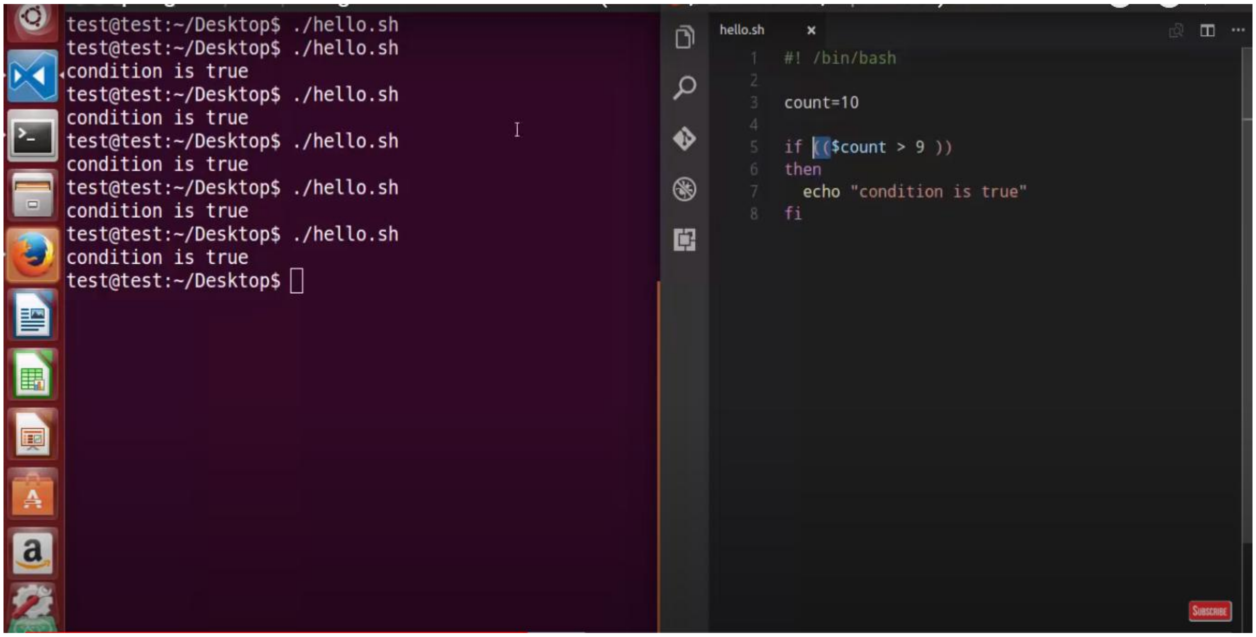
```
integer comparison

-eq - is equal to - if [ "$a" -eq "$b" ]
-ne - is not equal to - if [ "$a" -ne "$b" ]
-gt - is greater than - if [ "$a" -gt "$b" ]
-ge - is greater than or equal to - if [ "$a" -ge "$b" ]
-lt - is less than - if [ "$a" -lt "$b" ]
-le - is less than or equal to - if [ "$a" -le "$b" ]
< - is less than  - (("$a" < "$b"))
<= - is less than or equal to - (("$a" <= "$b"))
> - is greater than - (("$a" > "$b"))
>= - is greater than or equal to  - (("$a" >= "$b"))

string comparison
= - is equal to - if [ "$a" = "$b" ]
== - is equal to - if [ "$a" == "$b" ]
!= - is not equal to - if [ "$a" != "$b" ]
< - is less than, in ASCII alphabetical order - if [[ "$a" < "$b" ]
> - is greater than, in ASCII alphabetical order - if [[ "$a" > "$b
-z - string is null, that is, has zero length
```

```
#! /bin/bash

count=10

if [ $count  ]
then
  statement
fi
```



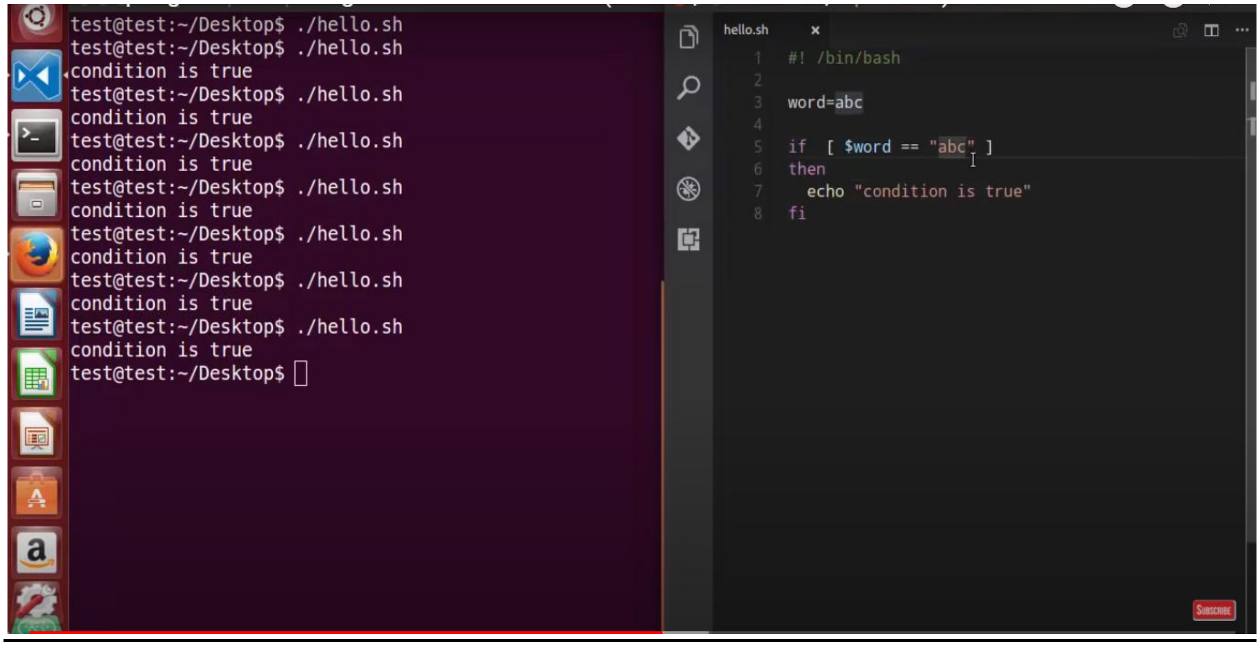
```
test@test:~/Desktop$ ./hello.sh
test@test:~/Desktop$
```

```
#! /bin/bash

count=10

if [ $count -eq 9 ]
then
  echo "condition is true"
fi
```



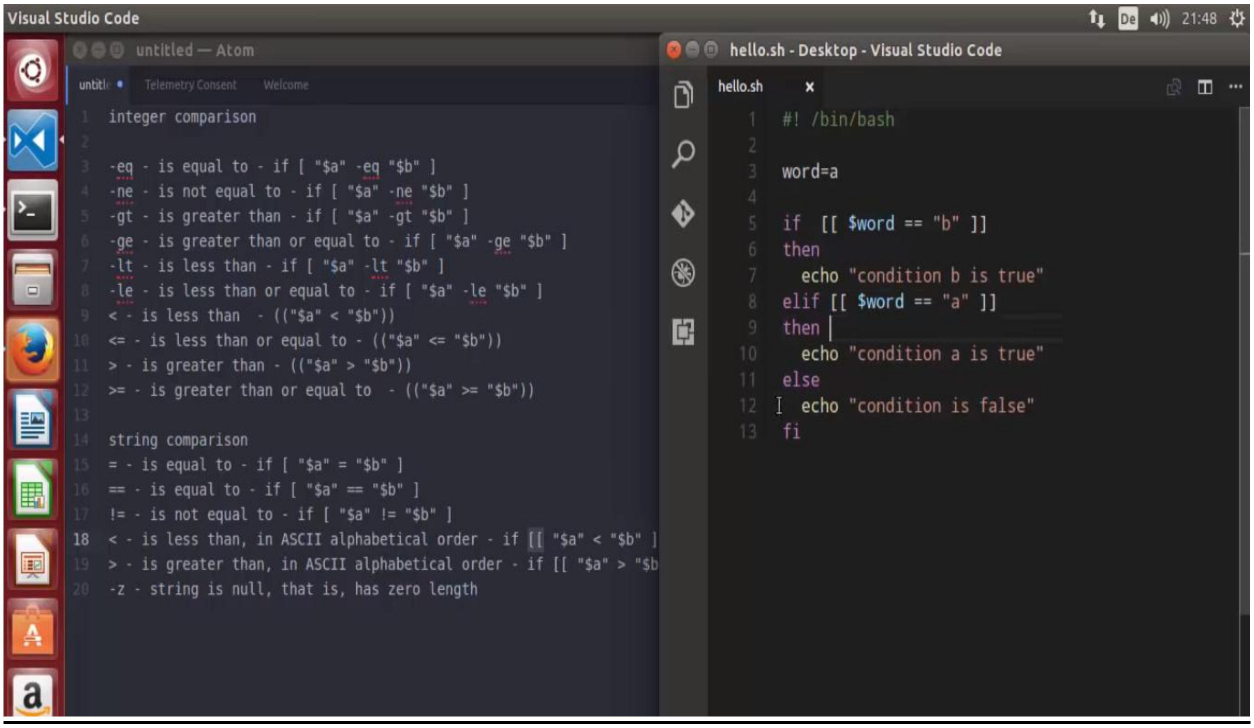
```
test@test:~/Desktop$ ./hello.sh
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ 
```

```
#! /bin/bash

count=10

if (($count > 9 ))
then
  echo "condition is true"
fi
```



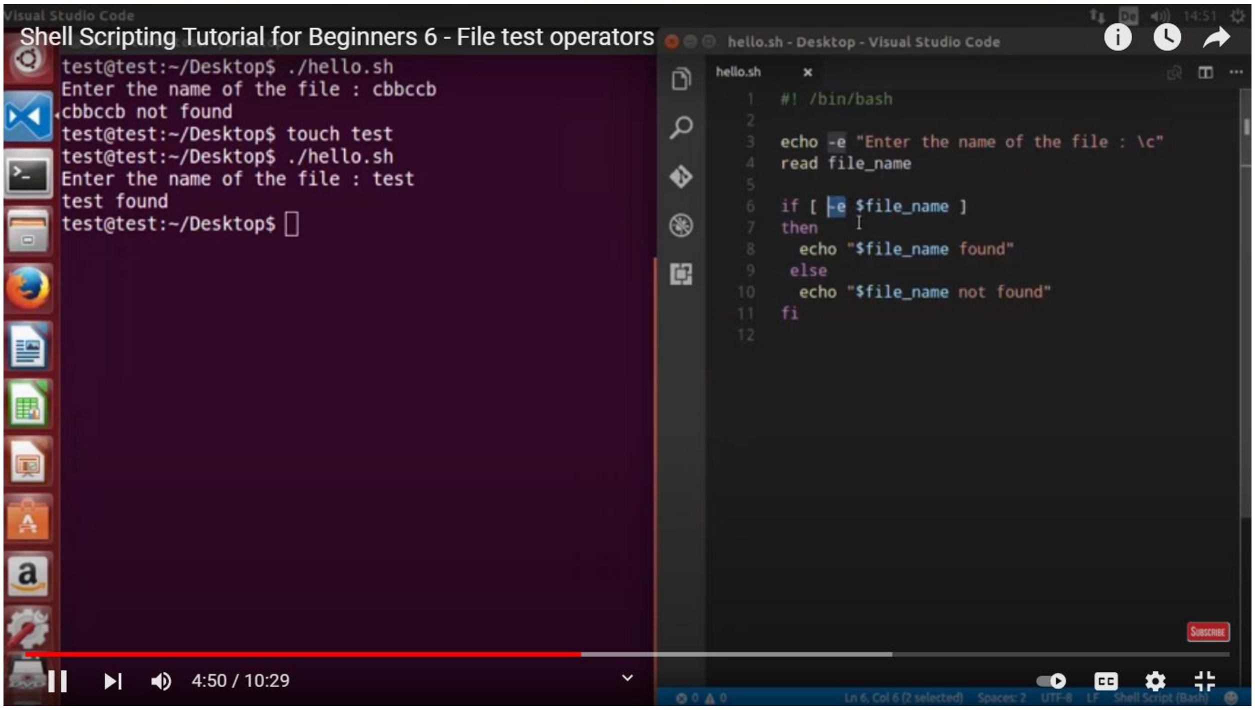
```
test@test:~/Desktop$ ./hello.sh
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ ./hello.sh
condition is true
test@test:~/Desktop$ 
```

```
#! /bin/bash

word=abc

if  [ $word == "abc" ]
then
  echo "condition is true"
fi
```



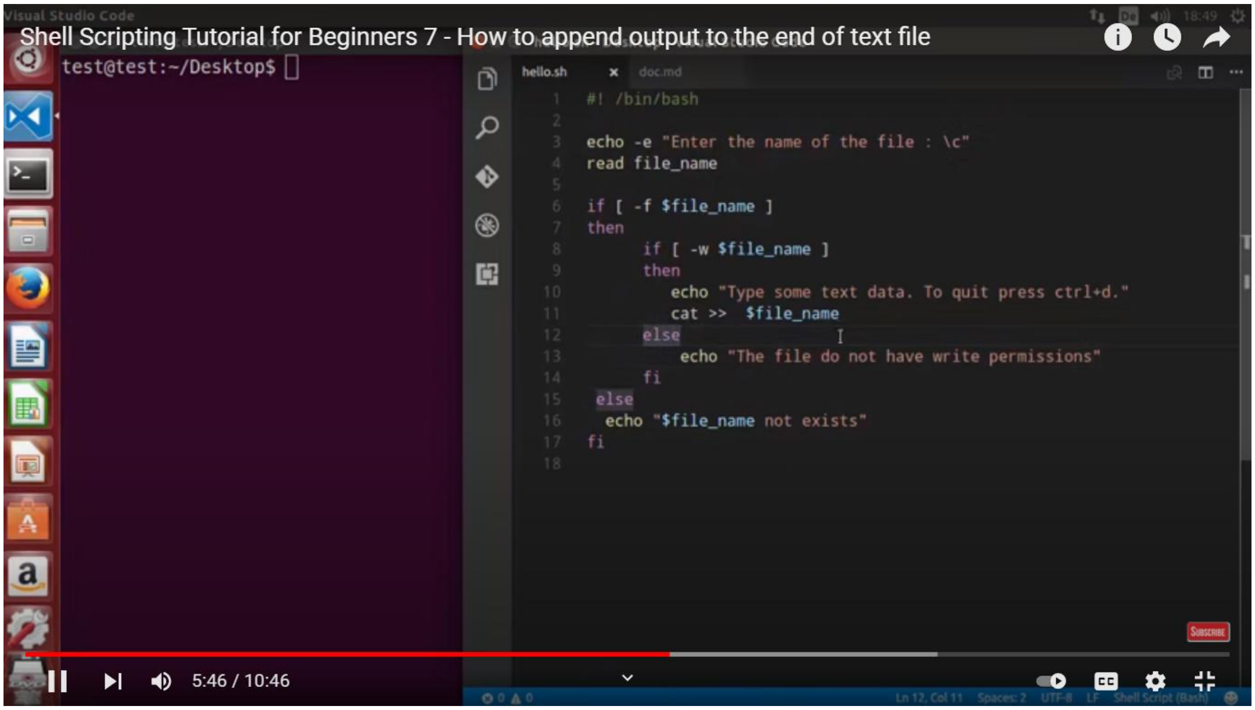
```
integer comparison

-eq - is equal to - if [ "$a" -eq "$b" ]
-ne - is not equal to - if [ "$a" -ne "$b" ]
-gt - is greater than - if [ "$a" -gt "$b" ]
-ge - is greater than or equal to - if [ "$a" -ge "$b" ]
-lt - is less than - if [ "$a" -lt "$b" ]
-le - is less than or equal to - if [ "$a" -le "$b" ]
< - is less than  - (("$a" < "$b"))
<= - is less than or equal to - (("$a" <= "$b"))
> - is greater than - (("$a" > "$b"))
>= - is greater than or equal to  - (("$a" >= "$b"))

string comparison
= - is equal to - if [ "$a" = "$b" ]
== - is equal to - if [ "$a" == "$b" ]
!= - is not equal to - if [ "$a" != "$b" ]
< - is less than, in ASCII alphabetical order - if [[ "$a" < "$b" ]
> - is greater than, in ASCII alphabetical order - if [[ "$a" > "$b
-z - string is null, that is, has zero length
```

```
#! /bin/bash

word=a

if  [[ $word == "b" ]]
then
  echo "condition b is true"
elif [[ $word == "a" ]]
then
  echo "condition a is true"
else
  echo "condition is false"
fi
```


Visual Studio Code
Shell Scripting Tutorial for Beginners 6 - File test operators
hello.sh - Desktop - Visual Studio Code
hello.sh
SUBSCRIBE
4:50 / 10:29
Ln 6, Col 6 (2 selected) Spaces: 2 UTF-8 LF Shell Script (Bash)

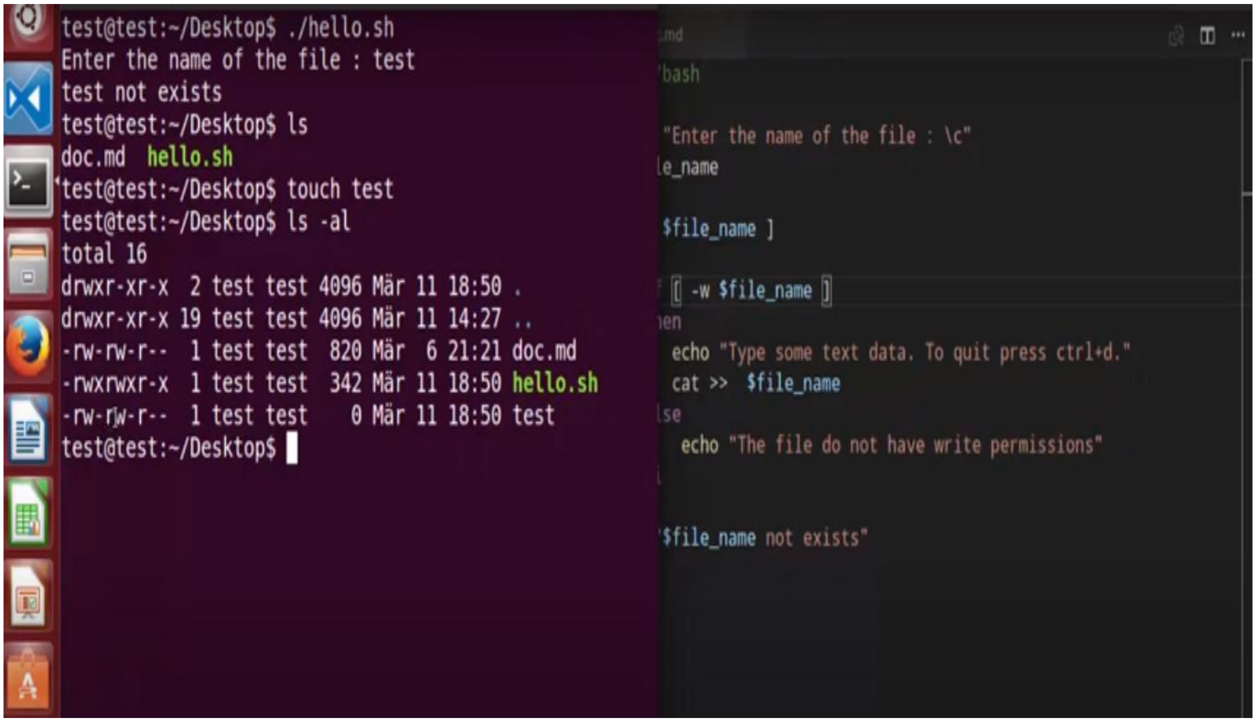
```
test@test:~/Desktop$ ./hello.sh
Enter the name of the file : cbbccb
cbbccb not found
test@test:~/Desktop$ touch test
test@test:~/Desktop$ ./hello.sh
Enter the name of the file : test
test found
test@test:~/Desktop$
```

```
#! /bin/bash

echo -e "Enter the name of the file : \c"
read file_name

if [ -e $file_name ]
then
  echo "$file_name found"
 else
  echo "$file_name not found"
fi
```


Visual Studio Code
Shell Scripting Tutorial for Beginners 7 - How to append output to the end of text file
hello.sh
doc.md
SUBSCRIBE
5:46 / 10:46
Ln 12, Col 11 Spaces: 2 UTF-8 LF Shell Script (Bash)

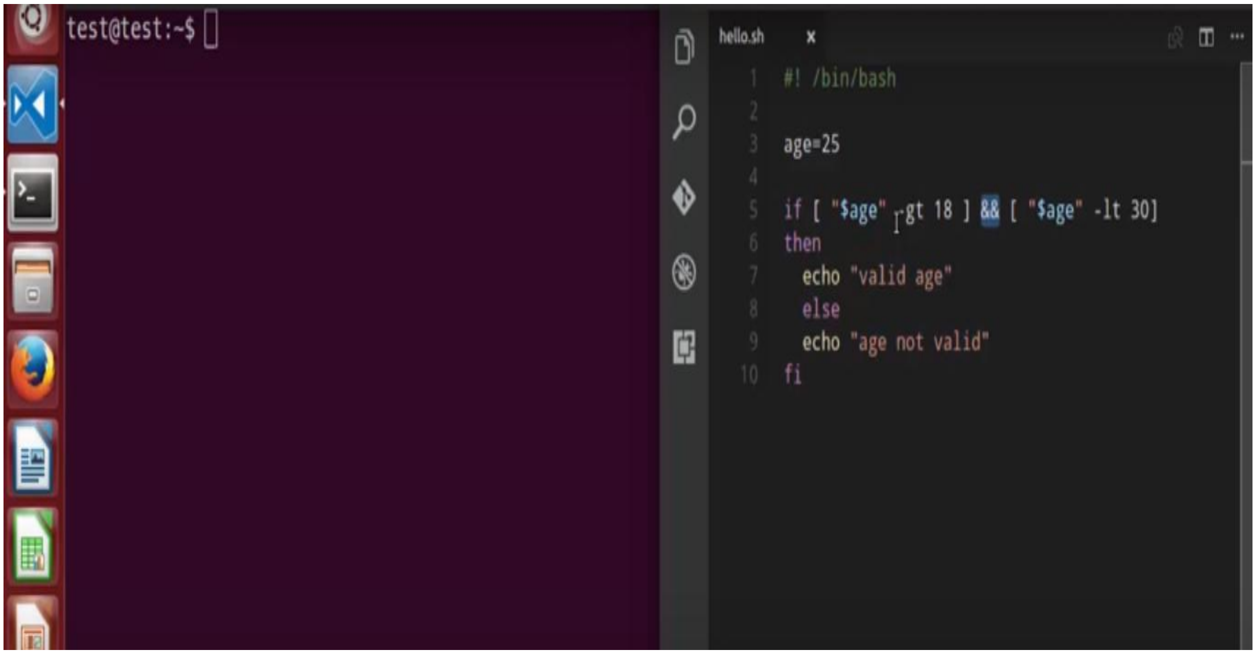
```
test@test:~/Desktop$
```

```
#! /bin/bash

echo -e "Enter the name of the file : \c"
read file_name

if [ -f $file_name ]
then
      if [ -w $file_name ]
      then
         echo "Type some text data. To quit press ctrl+d."
         cat >>  $file_name
      else
          echo "The file do not have write permissions"
      fi
 else
  echo "$file_name not exists"
fi
```



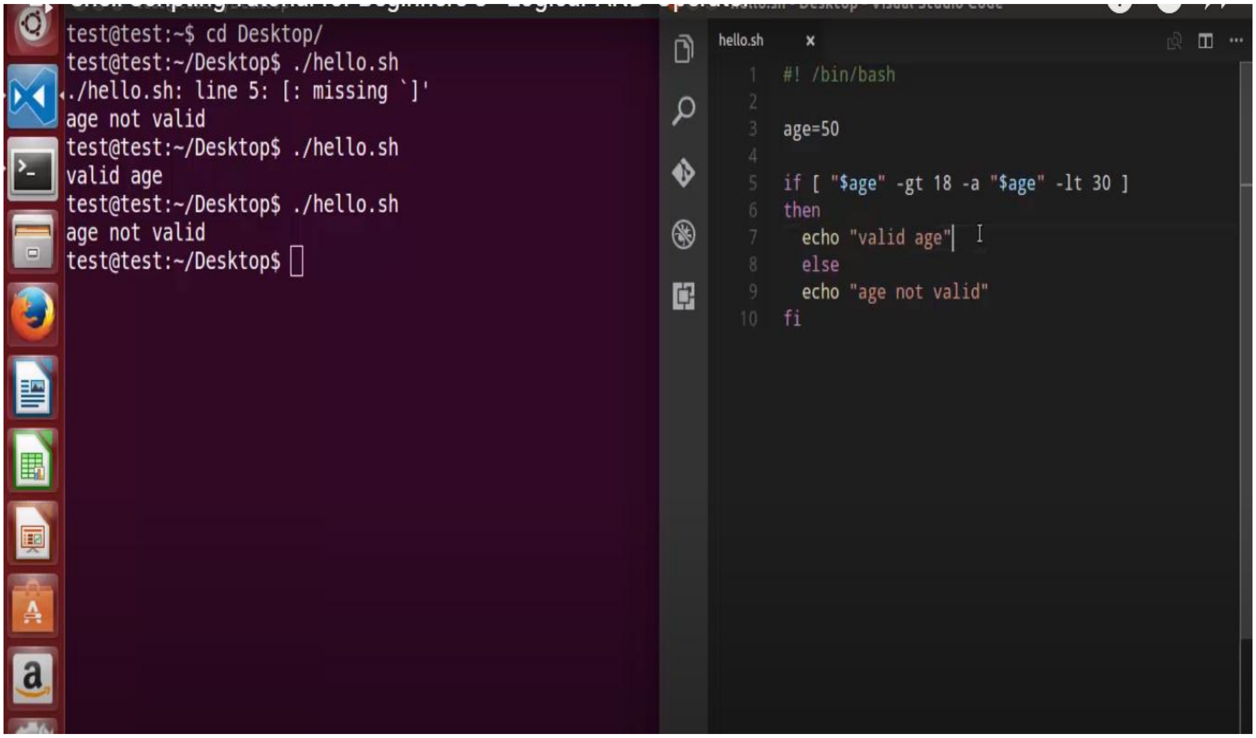
```
test@test:~/Desktop$ ./hello.sh
Enter the name of the file : test
test not exists
test@test:~/Desktop$ ls
doc.md  hello.sh
test@test:~/Desktop$ touch test
test@test:~/Desktop$ ls -al
total 16
drwxr-xr-x  2 test test 4096 Mär 11 18:50 .
drwxr-xr-x 19 test test 4096 Mär 11 14:27 ..
-rw-rw-r--  1 test test  820 Mär  6 21:21 doc.md
-rwxrwxr-x  1 test test  342 Mär 11 18:50 hello.sh
-rw-rw-r--  1 test test    0 Mär 11 18:50 test
test@test:~/Desktop$
```

```
/bash

"Enter the name of the file : \c"
le_name

$file_name ]

[ -w $file_name ]
nen
 echo "Type some text data. To quit press ctrl+d."
 cat >>  $file_name
lse
  echo "The file do not have write permissions"


'$file_name not exists"
```



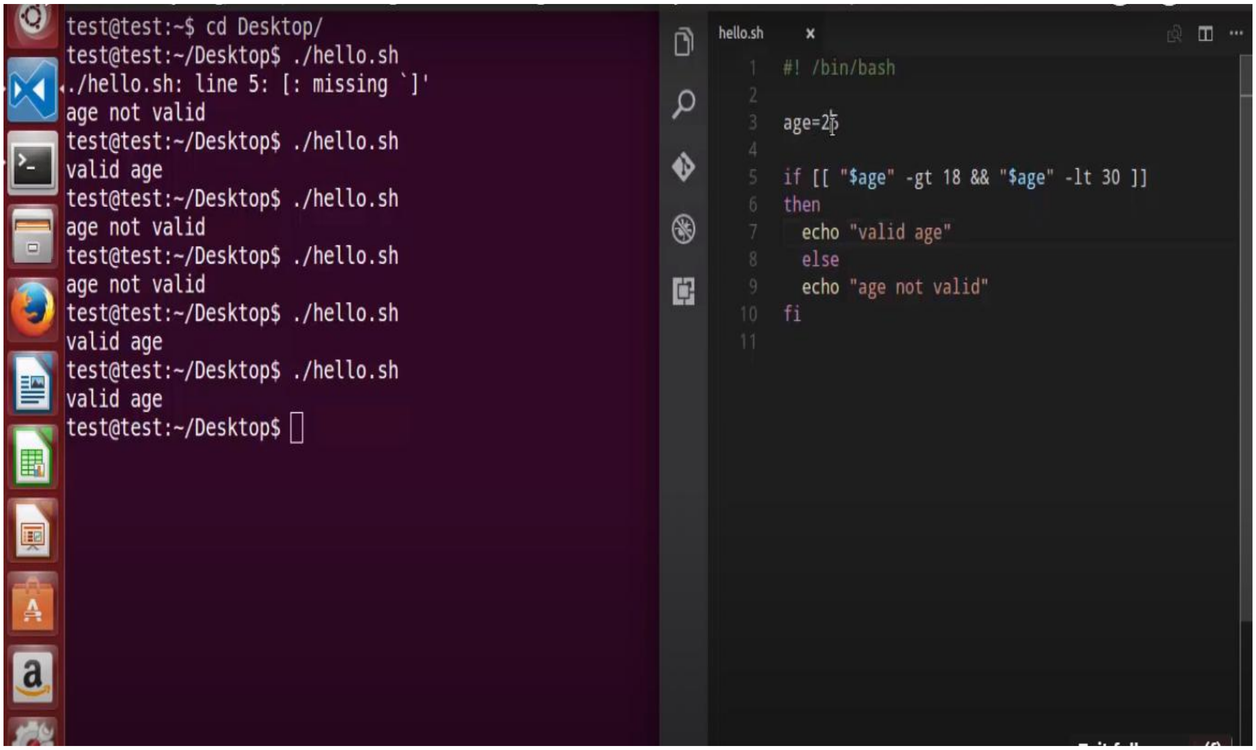
```
test@test:~$
```

```
#! /bin/bash

age=25

if [ "$age" -gt 18 ] && [ "$age" -lt 30]
then
  echo "valid age"
  else
  echo "age not valid"
fi
```



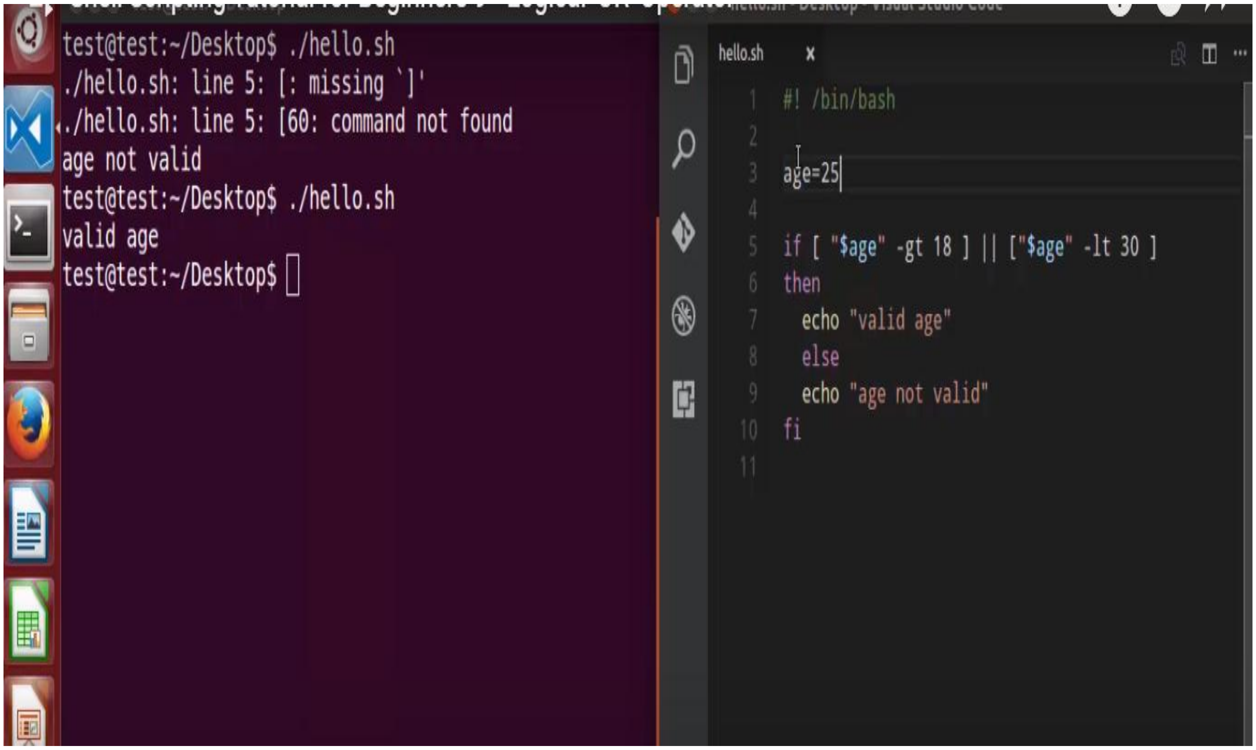
```
test@test:~$ cd Desktop/
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [: missing `]'
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$
```

```
#! /bin/bash

age=50

if [ "$age" -gt 18 -a "$age" -lt 30 ]
then
  echo "valid age"
  else
  echo "age not valid"
fi
```



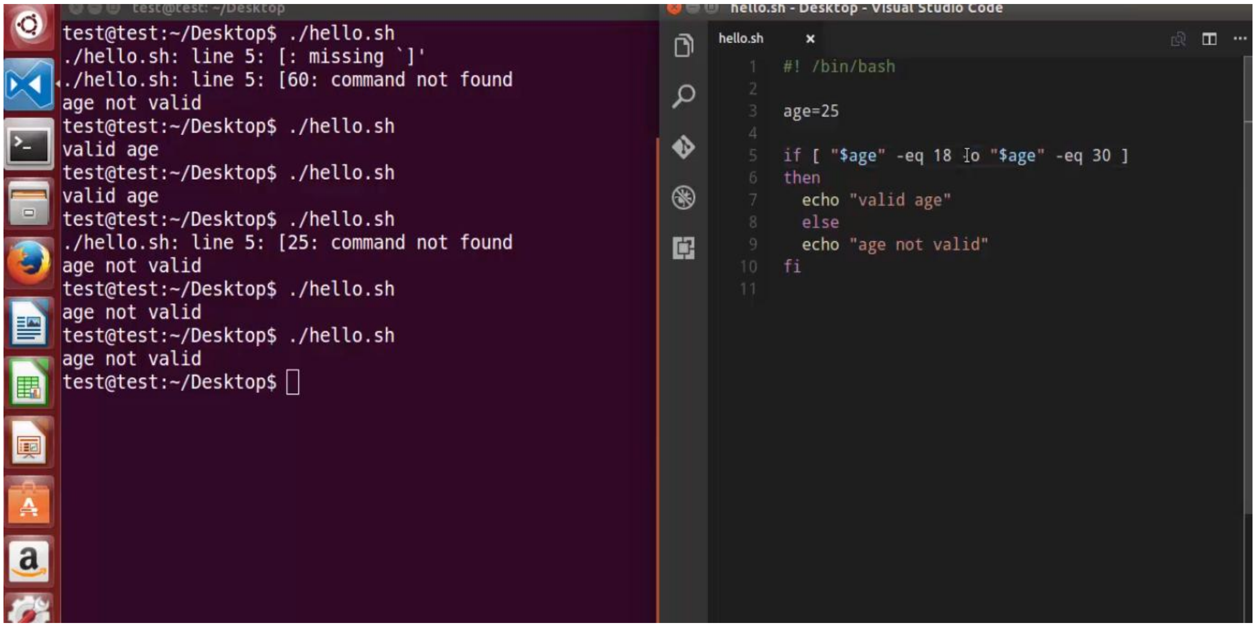
```
test@test:~$ cd Desktop/
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [: missing `]'
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$
```

```
#! /bin/bash

age=25

if [[ "$age" -gt 18 && "$age" -lt 30 ]]
then
  echo "valid age"
  else
  echo "age not valid"
fi

```



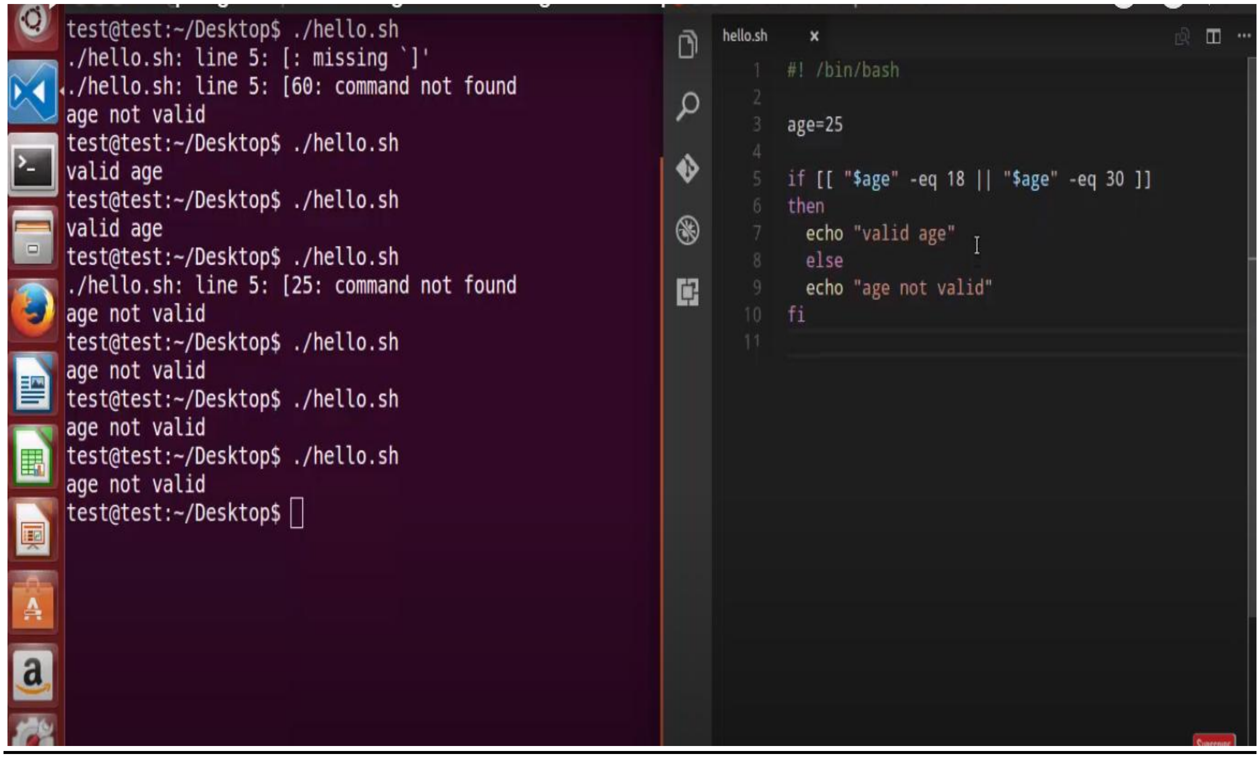
```
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [: missing `]'
./hello.sh: line 5: [60: command not found
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ 
```

```
1  #! /bin/bash
2  
3  age=25
4  
5  if [ "$age" -gt 18 ] || ["$age" -lt 30 ]
6  then
7    echo "valid age"
8    else
9    echo "age not valid"
10 fi
11 
```



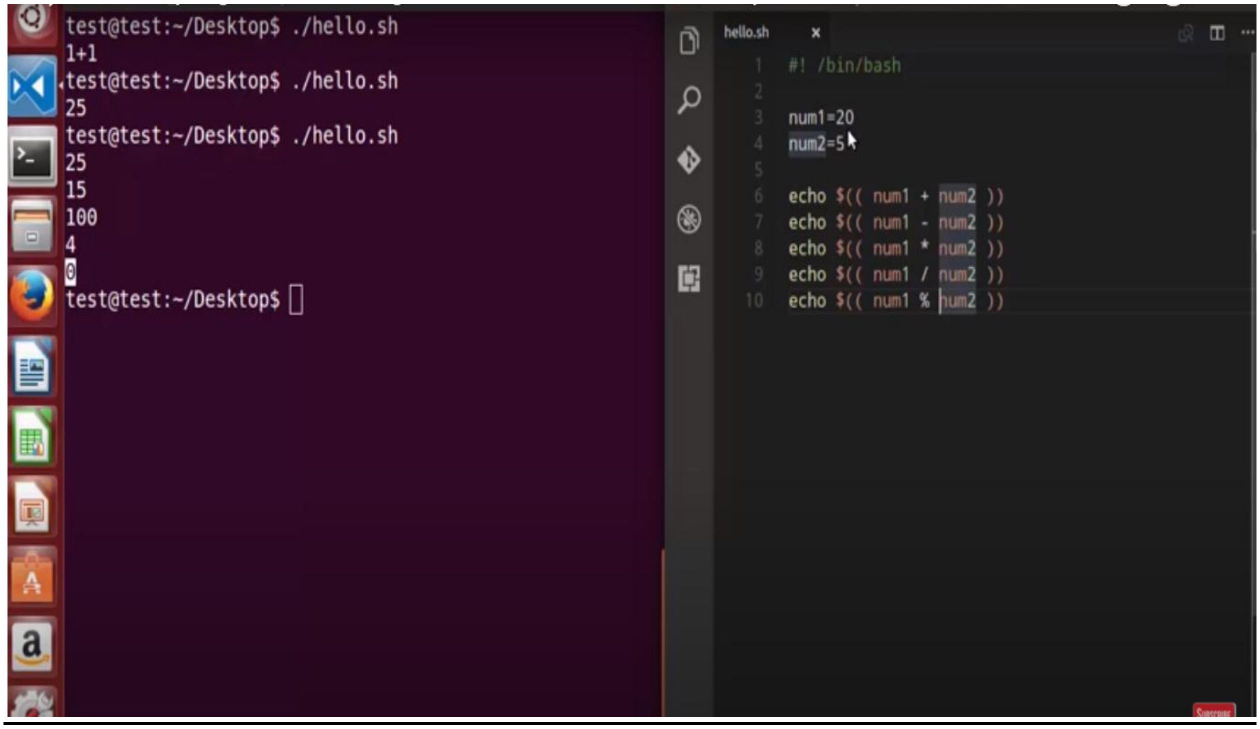
```
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [: missing `]'
./hello.sh: line 5: [60: command not found
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [25: command not found
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ 
```

```
1  #! /bin/bash
2  
3  age=25
4  
5  if [ "$age" -eq 18 -o "$age" -eq 30 ]
6  then
7    echo "valid age"
8    else
9    echo "age not valid"
10 fi
11 
```



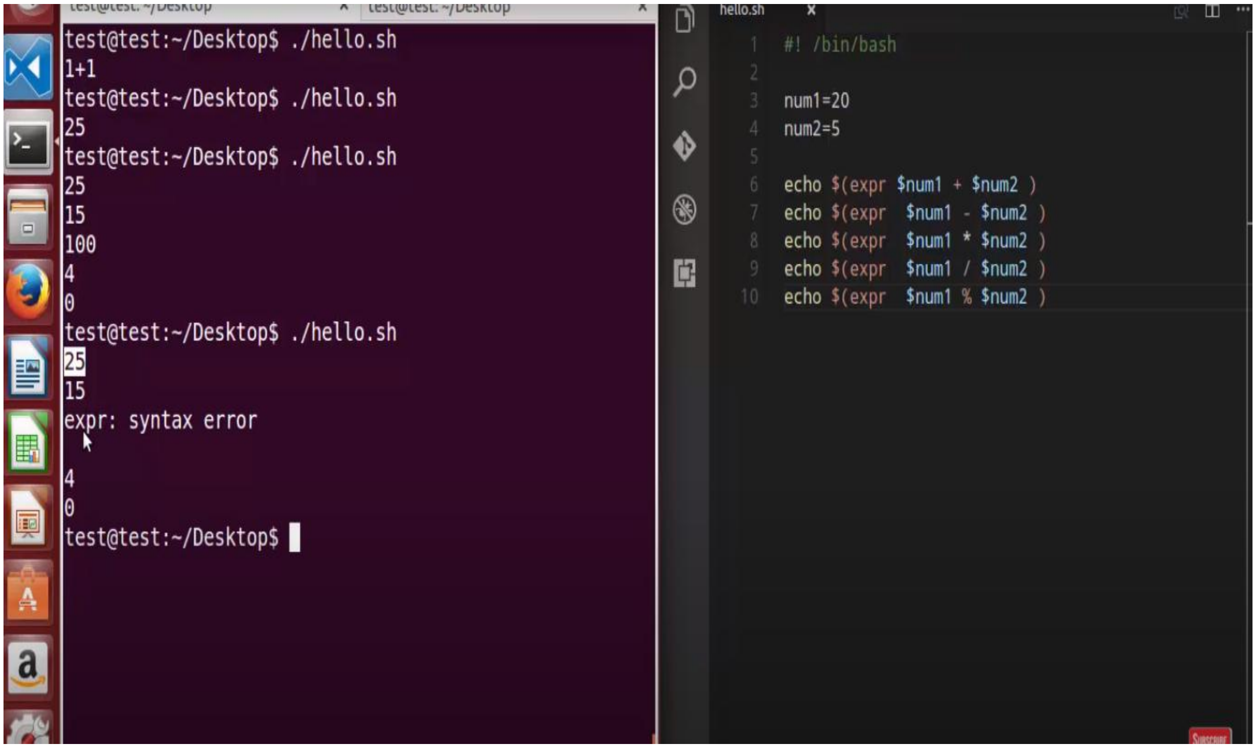
```
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [: missing `]'
./hello.sh: line 5: [60: command not found
age not valid
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
valid age
test@test:~/Desktop$ ./hello.sh
./hello.sh: line 5: [25: command not found
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$ ./hello.sh
age not valid
test@test:~/Desktop$
```

```
#! /bin/bash

age=25

if [[ "$age" -eq 18 || "$age" -eq 30 ]]
then
  echo "valid age"
  else
  echo "age not valid"
fi
```



```
test@test:~/Desktop$ ./hello.sh
1+1
test@test:~/Desktop$ ./hello.sh
25
test@test:~/Desktop$ ./hello.sh
25
15
100
4
0
test@test:~/Desktop$
```

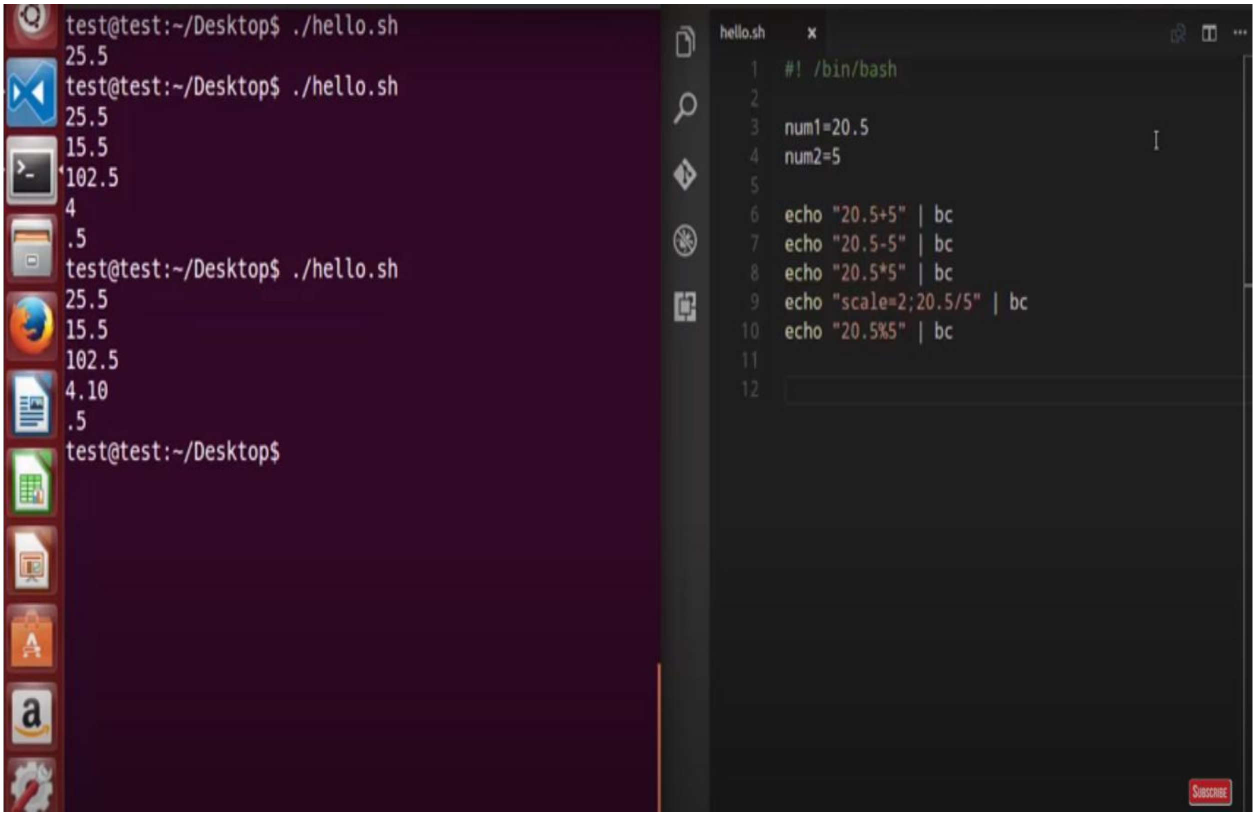
```
#! /bin/bash

num1=20
num2=5

echo $(( num1 + num2 ))
echo $(( num1 - num2 ))
echo $(( num1 * num2 ))
echo $(( num1 / num2 ))
echo $(( num1 % num2 ))
```



```
test@test:~/Desktop$ ./hello.sh
1+1
test@test:~/Desktop$ ./hello.sh
25
test@test:~/Desktop$ ./hello.sh
25
15
100
4
0
test@test:~/Desktop$ ./hello.sh
25
15
expr: syntax error

4
0
test@test:~/Desktop$
```

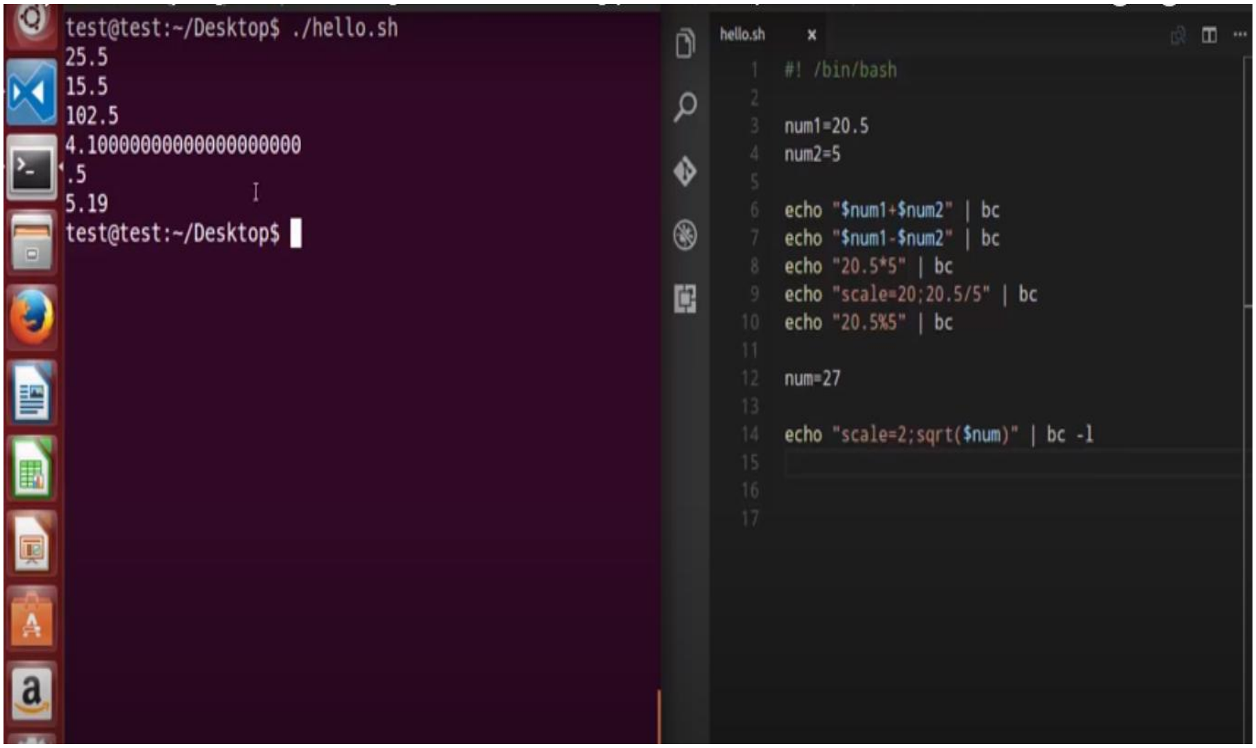
```
#! /bin/bash

num1=20
num2=5

echo $(expr $num1 + $num2 )
echo $(expr  $num1 - $num2 )
echo $(expr  $num1 * $num2 )
echo $(expr  $num1 / $num2 )
echo $(expr  $num1 % $num2 )
```

Floating point arithmetic operation:



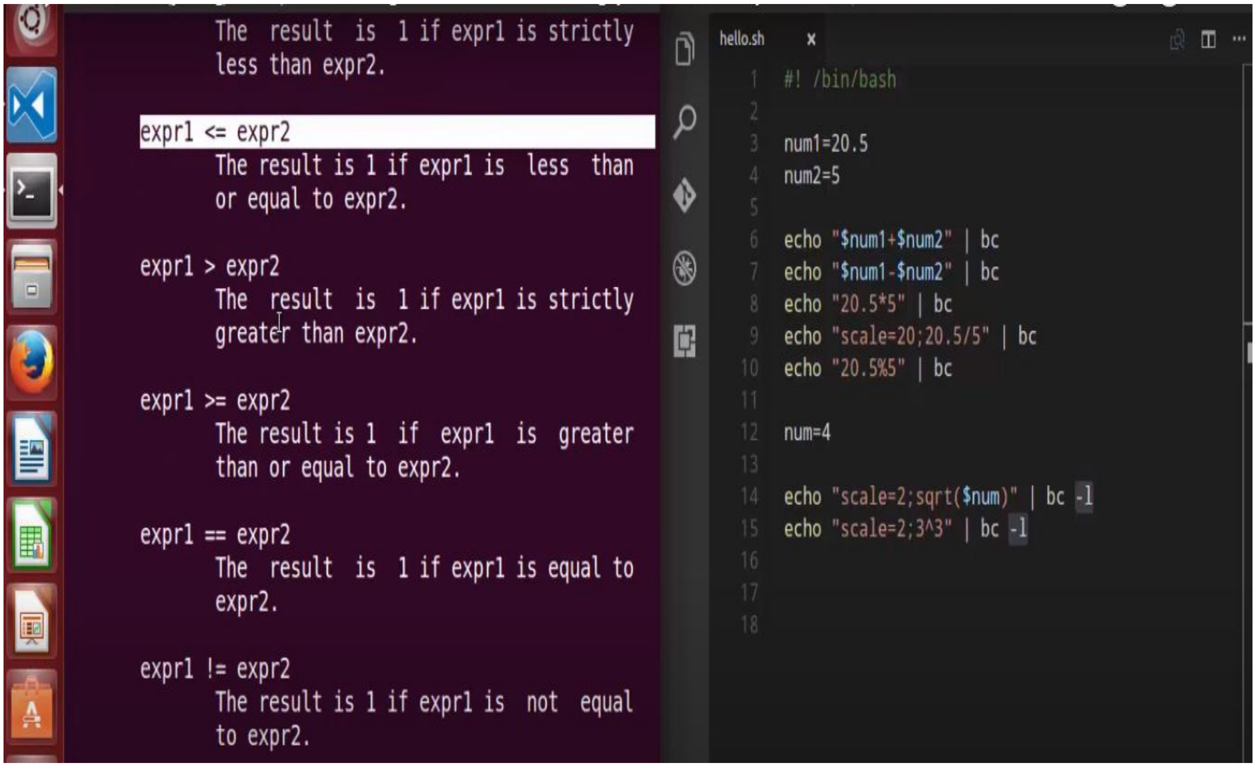
```
test@test:~/Desktop$ ./hello.sh
25.5
test@test:~/Desktop$ ./hello.sh
25.5
15.5
102.5
4
.5
test@test:~/Desktop$ ./hello.sh
25.5
15.5
102.5
4.10
.5
test@test:~/Desktop$
```

```
#! /bin/bash

num1=20.5
num2=5

echo "20.5+5" | bc
echo "20.5-5" | bc
echo "20.5*5" | bc
echo "scale=2;20.5/5" | bc
echo "20.5%5" | bc
```



```
test@test:~/Desktop$ ./hello.sh
25.5
15.5
102.5
4.10000000000000000000
.5
5.19
test@test:~/Desktop$
```

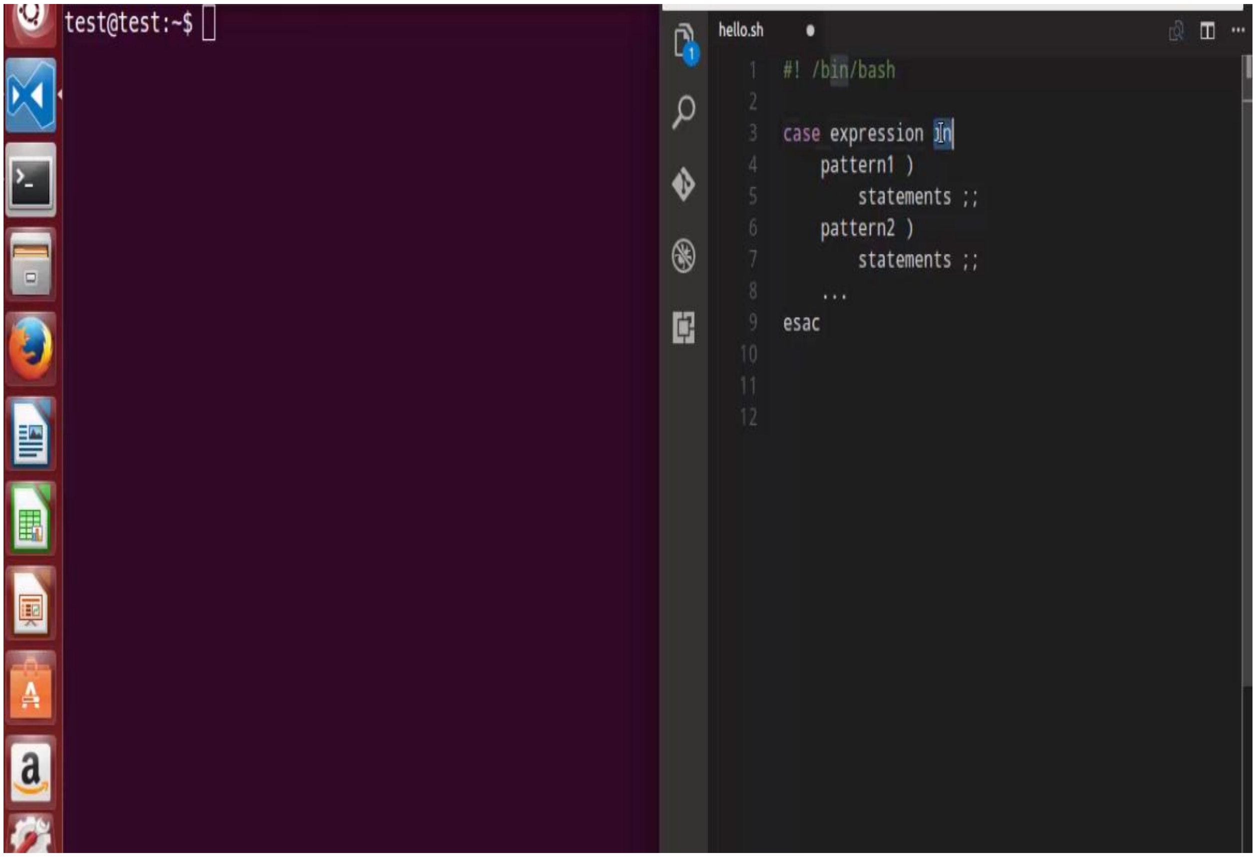
```
#! /bin/bash

num1=20.5
num2=5

echo "$num1+$num2" | bc
echo "$num1-$num2" | bc
echo "20.5*5" | bc
echo "scale=20;20.5/5" | bc
echo "20.5%5" | bc

num=27

echo "scale=2;sqrt($num)" | bc -l
```



```
              The  result  is  1 if expr1 is strictly
              less than expr2.

       expr1 <= expr2
              The result is 1 if expr1 is  less  than
              or equal to expr2.

       expr1 > expr2
              The  result  is  1 if expr1 is strictly
              greater than expr2.

       expr1 >= expr2
              The result is 1  if  expr1  is  greater
              than or equal to expr2.

       expr1 == expr2
              The  result  is  1 if expr1 is equal to
              expr2.

       expr1 != expr2
              The result is 1 if expr1 is  not  equal
              to expr2.
```

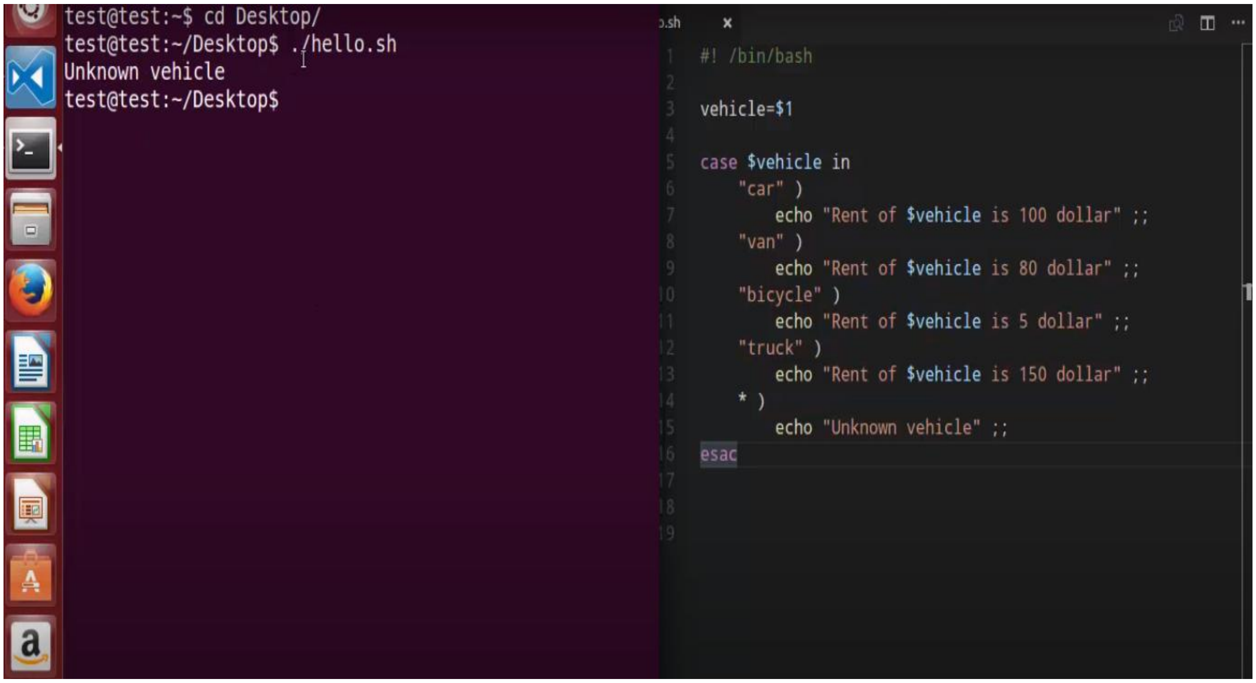
```
#! /bin/bash

num1=20.5
num2=5

echo "$num1+$num2" | bc
echo "$num1-$num2" | bc
echo "20.5*5" | bc
echo "scale=20;20.5/5" | bc
echo "20.5%5" | bc

num=4

echo "scale=2;sqrt($num)" | bc -l
echo "scale=2;3^3" | bc -l
```

Case Statement:



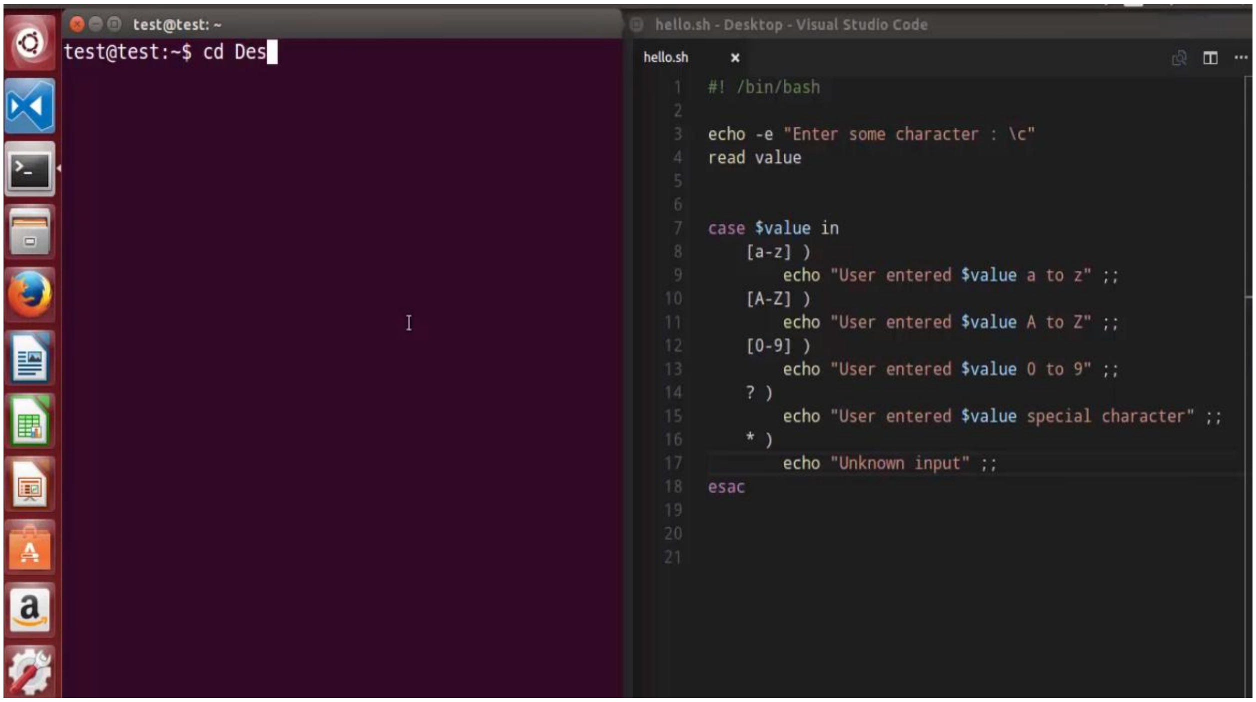
```
test@test:~$
```

```
#! /bin/bash

case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac
```



```
test@test:~$ cd Desktop/
test@test:~/Desktop$ ./hello.sh
Unknown vehicle
test@test:~/Desktop$
```

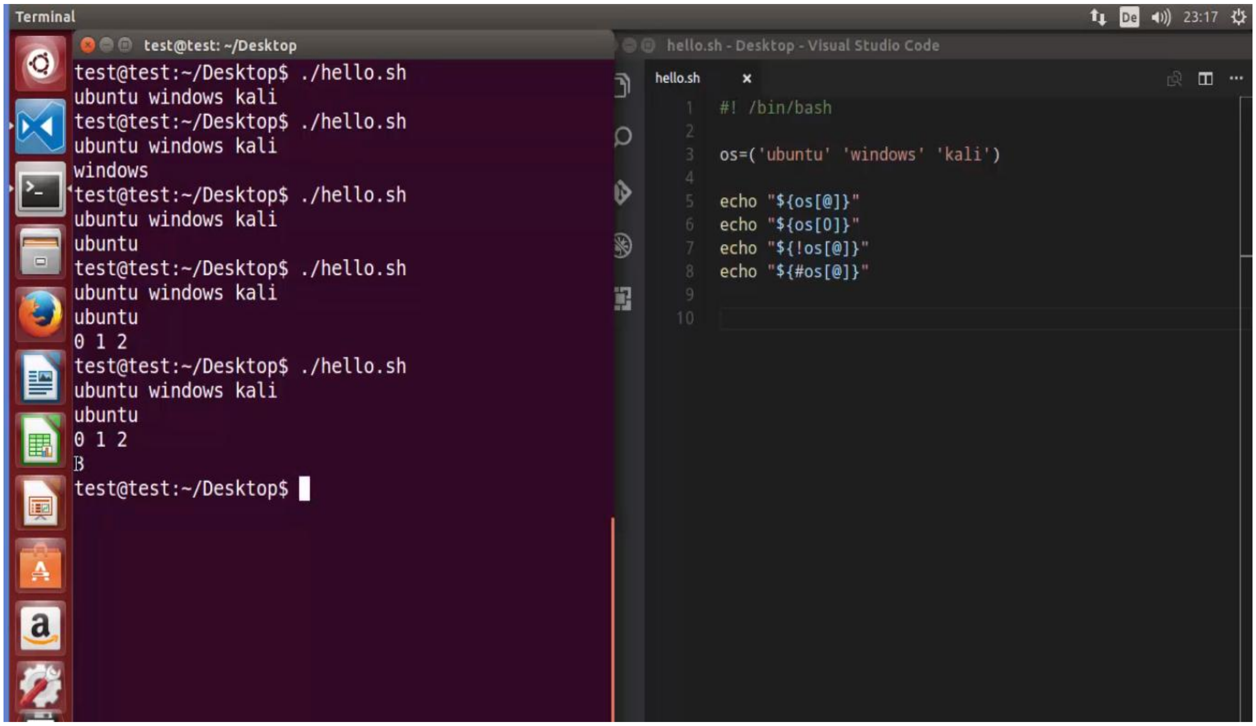
```
#! /bin/bash

vehicle=$1

case $vehicle in
    "car" )
        echo "Rent of $vehicle is 100 dollar" ;;
    "van" )
        echo "Rent of $vehicle is 80 dollar" ;;
    "bicycle" )
        echo "Rent of $vehicle is 5 dollar" ;;
    "truck" )
        echo "Rent of $vehicle is 150 dollar" ;;
    * )
        echo "Unknown vehicle" ;;
esac
```



```
test@test:~$ cd Des
```

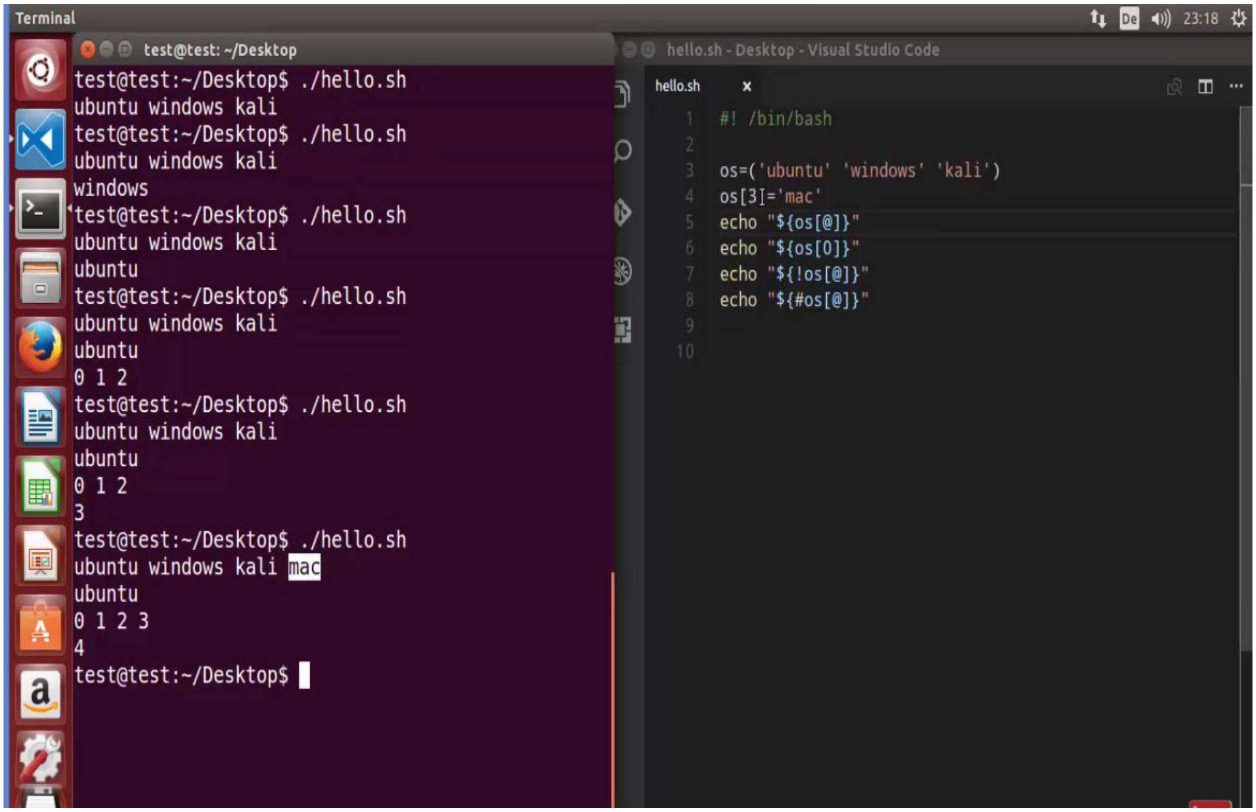
```
#! /bin/bash

echo -e "Enter some character : \c"
read value


case $value in
    [a-z] )
        echo "User entered $value a to z" ;;
    [A-Z] )
        echo "User entered $value A to Z" ;;
    [0-9] )
        echo "User entered $value 0 to 9" ;;
    ? )
        echo "User entered $value special character" ;;
    * )
        echo "Unknown input" ;;
esac
```

Array:



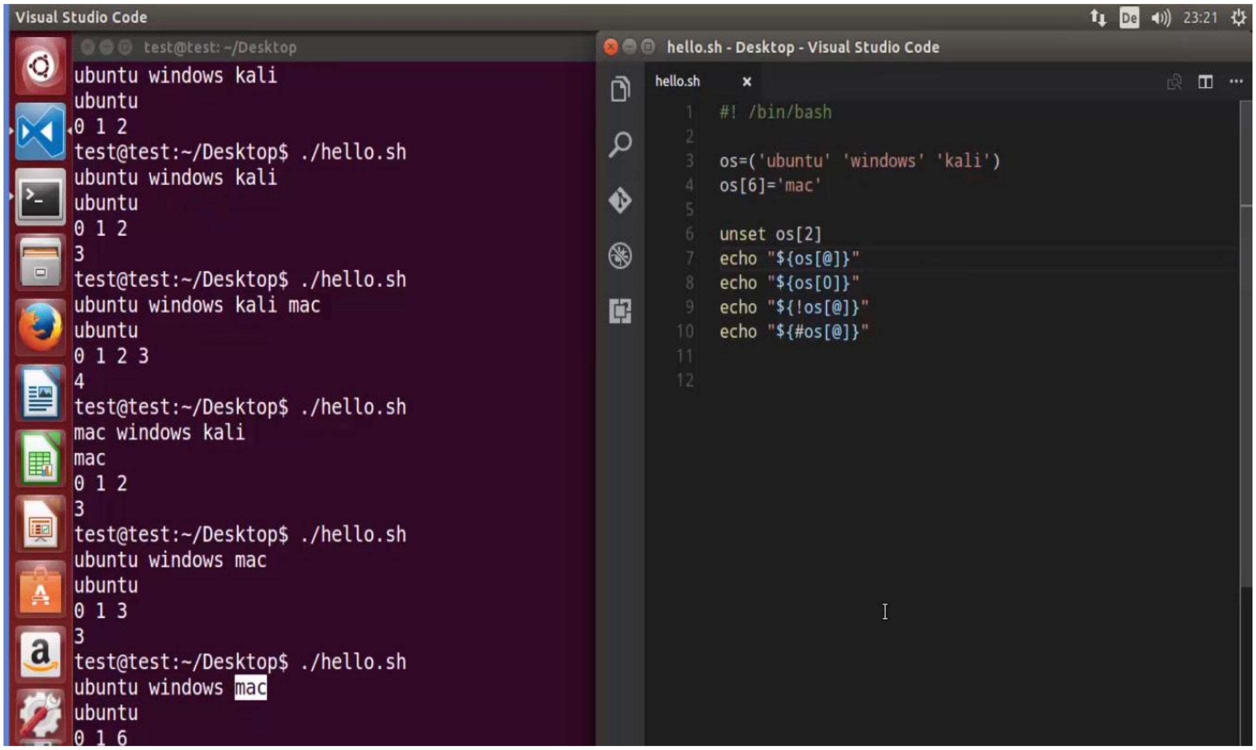
```
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
windows
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
0 1 2
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
0 1 2
3
test@test:~/Desktop$
```

```
#! /bin/bash

os=('ubuntu' 'windows' 'kali')

echo "${os[@]}"
echo "${os[0]}"
echo "${!os[@]}"
echo "${#os[@]}"
```



```
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
windows
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
0 1 2
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
0 1 2
3
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali mac
ubuntu
0 1 2 3
4
test@test:~/Desktop$
```

```
#! /bin/bash

os=('ubuntu' 'windows' 'kali')
os[3]='mac'
echo "${os[@]}"
echo "${os[0]}"
echo "${!os[@]}"
echo "${#os[@]}"


```



```
ubuntu windows kali
ubuntu
0 1 2
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali
ubuntu
0 1 2
3
test@test:~/Desktop$ ./hello.sh
ubuntu windows kali mac
ubuntu
0 1 2 3
4
test@test:~/Desktop$ ./hello.sh
mac windows kali
mac
0 1 2
3
test@test:~/Desktop$ ./hello.sh
ubuntu windows mac
ubuntu
0 1 3
3
test@test:~/Desktop$ ./hello.sh
ubuntu windows mac
ubuntu
0 1 6
```

```
#! /bin/bash

os=('ubuntu' 'windows' 'kali')
os[6]='mac'

unset os[2]
echo "${os[@]}"
echo "${os[0]}"
echo "${!os[@]}"
echo "${#os[@]}"


```

The For Loop

The for loop is another widely used bash shell construct that allows users to iterate over codes efficiently. A simple example is demonstrated below.

```
#!/bin/bash
for (( counter=1; counter<=10; counter++ ))
do
echo -n "$counter "
done
printf "\n"
```

Save this code in a file named for.sh and run it using ./for.sh. Don’t forget to make it executable. This program should print out the numbers 1 to 10.

```
#!/bin/bash
fruits=(“apple” “banana” “cherry” “date”)
for fruit in “${fruits[@]}”; do
echo “Current fruit: $fruit”
done
```

The While Loop

The while loop construct is used to run some instructions multiple times. Check out the following script called while.sh for a better understanding of this concept.

```
#!/bin/bash
i=0
while [ $i -le 2 ]
do
echo Number: $i
((i++))
done
```

So, the while loop takes the below form.

```
while [ condition ]
do
commands 1
commands n
done
```

The space surrounding the square brackets is mandatory.

Concatenating Strings

String processing is of extreme importance to a wide range of modern bash scripts. Thankfully, it is much more comfortable in bash and allows for a more precise, concise way to implement this. See the below example for a glance into bash string concatenation.

```
#!/bin/bash

string1="Ubuntu"
string2="Pit"
string=$string1$string2
echo "$string is a great resource for Linux beginners."
```

The following program outputs the string "UbuntuPit is a great resource for Linux beginners." to the screen.

Slicing Strings

Unlike many programming languages, bash doesn't provide any built-in function for cutting portions of a string. However, the below example demonstrates how this can be done using parameter expansion.

```
#!/bin/bash
Str="Learn Bash Commands from UbuntuPit"
subStr=${Str:0:20}
echo $subStr
```

This script should print out "*Learn Bash Commands*" as its output. The parameter expansion takes the form **${VAR_NAME:S:L}**. Here, S denotes the starting position, and L indicates the length.

Extracting Substrings Using Cut

The Linux cut command can be used inside your scripts to 'cut' a portion of a string, aka the substring. The next example shows how this can be done.

```
#!/bin/bash
Str="Learn Bash Commands from UbuntuPit"
#subStr=${Str:0:20}
```

```
subStr=$(echo $Str| cut -d ' ' -f 1-3)
echo $subStr
```

Adding Two Values

It’s quite easy to perform arithmetic operations inside Linux shell scripts. The example below demonstrates how to receive and add two numbers as input from the user and add them.

```
#!/bin/bash
echo -n "Enter first number:"
read x
echo -n "Enter second number:"
read y
(( sum=x+y ))
echo "The result of addition=$sum"
```

As you can see, adding numbers in bash is reasonably straightforward.

Adding Multiple Values

You can use loops to get multiple user inputs and add them to your script. The following examples show this in action.

```
#!/bin/bash
sum=0
for (( counter=1; counter<5; counter++ ))
do
echo -n "Enter Your Number:"
read n
(( sum+=n ))
#echo -n "$counter "
done
printf "\n"
echo "Result is: $sum"
```

However, omitting the **(( ))** will result in string concatenation rather than addition. So, check for things like this in your program.

Functions in Bash

As with any programming dialect, functions play an essential role in Linux shell scripts. They allow admins to create custom code blocks for frequent usage. The below demonstration will outline how functions work in Linux bash scripts.

```
#!/bin/bash
function Add()
{
echo -n "Enter a Number: "
read x
echo -n "Enter another Number: "
read y
echo "Adiition is: $(( x+y ))"
}

Add
```

Functions with Return Values

One of the most fantastic functions is allowing the passing of data from one function to another. It is useful in a wide variety of scenarios. Check out the next example.

```
#!/bin/bash
function Greet() {

str="Hello $name, what brings you to UbuntuPit.com?"
echo $str
}

echo "-> what's your name?"
read name

val=$(Greet)
echo -e "-> $val"
```

Parsing Date and Time

The next bash script example will show you how to handle dates and times using scripts. Again, the Linux date command is used to get the necessary information and our program parses.

```
#!/bin/bash
year=`date +%Y`
month=`date +%m`
day=`date +%d`
hour=`date +%H`
minute=`date +%M`
second=`date +%S`
echo `date`
echo "Current Date is: $day-$month-$year"
echo "Current Time is: $hour:$minute:$second"
```