

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:~$
```

```
sktop
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
```

The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal window has a title bar that says "Terminal" and a prompt "test@test: ~/Desktop". It shows the execution of a script named "hello.sh". The output of the script is "Hello World". The terminal also shows the user running "/bin/bash" and checking the version "4.3.11(1)-release". The Visual Studio Code editor has a title bar that says "hello.sh - Desktop - Visual Studio Code". It shows the contents of the "hello.sh" script, which is a shell script that prints "Hello World" and some system information.

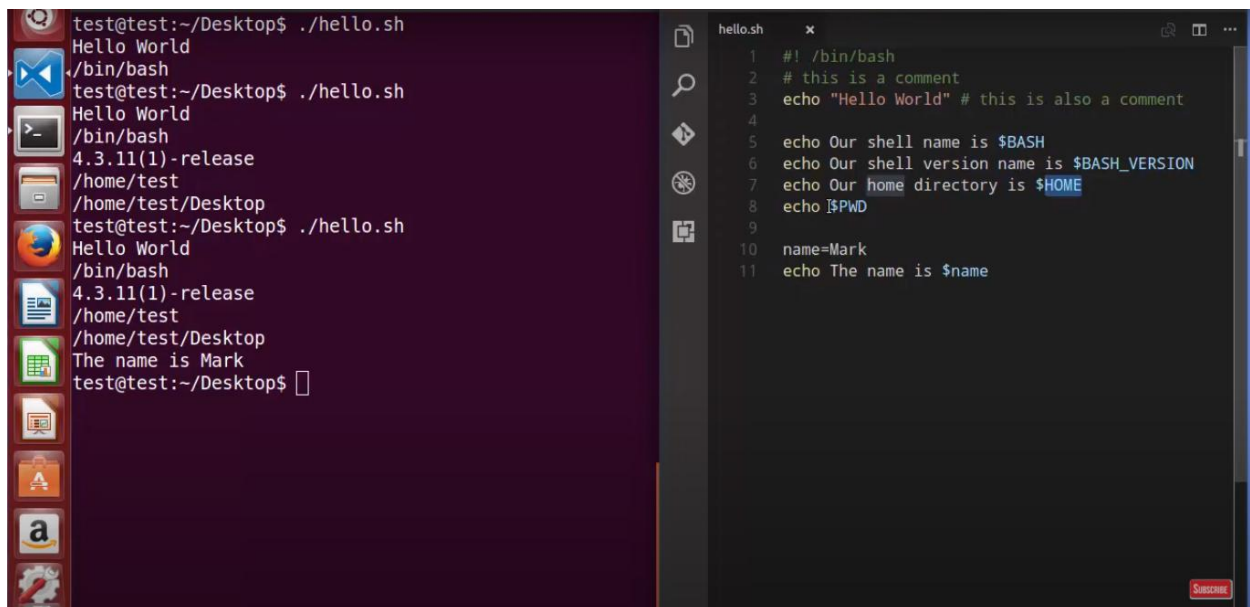
```
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
test@test:~/Desktop$ ./hello.sh
Hello World
/bin/bash
4.3.11(1)-release
/home/test
/home/test/Desktop
test@test:~/Desktop$
```

```
hello.sh
1  #!/bin/bash
2  # this is a comment
3  echo "Hello World" # this is also a comment
4
5  echo $BASH
6  echo $BASH_VERSION
7  echo $HOME
8  echo $PWD
```

The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal window has a title bar that says "Terminal" and a prompt "test@test: ~/Desktop". It shows the execution of a script named "hello.sh". The output of the script is "Hello World". The terminal also shows the user running "/bin/bash" and checking the version "4.3.11(1)-release". The Visual Studio Code editor has a title bar that says "hello.sh - Desktop - Visual Studio Code". It shows the contents of the "hello.sh" script, which is a shell script that prints "Hello World" and some system information, and also sets a variable "name" to "Mark" and prints "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$
```

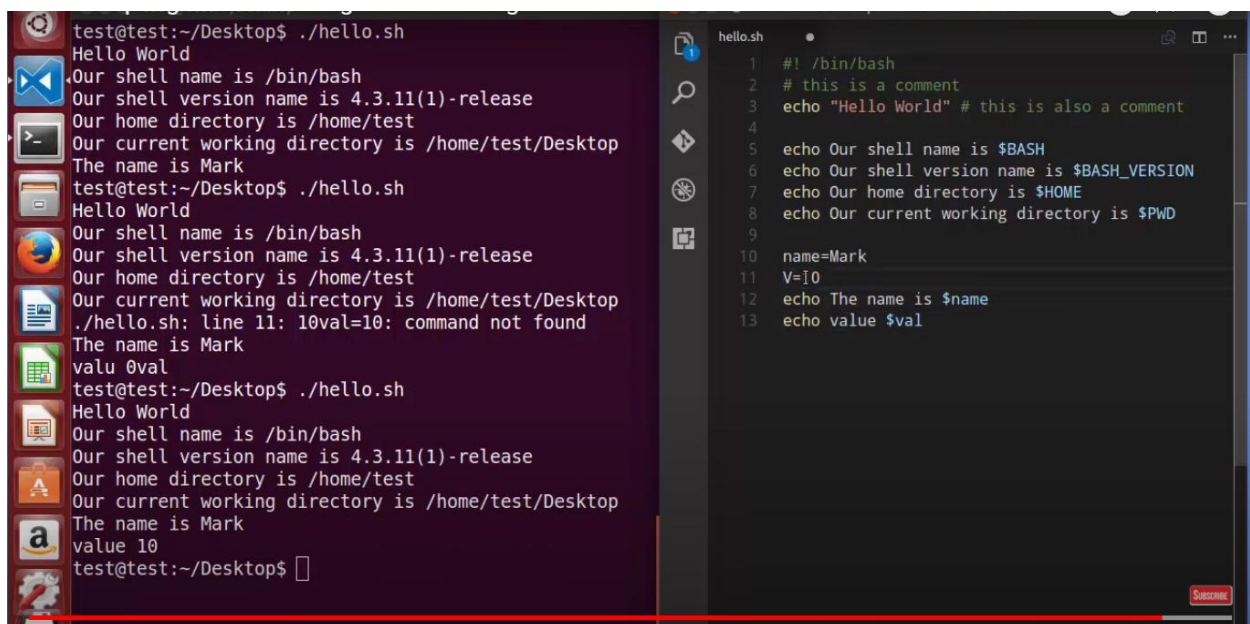
```
hello.sh
1  #!/bin/bash
2  # this is a comment
3  echo "Hello World" # this is also a comment
4
5  echo $BASH
6  echo $BASH_VERSION
7  echo $HOME
8  echo $PWD
9
10 name=Mark
11 echo The $name
```



The image shows a terminal window on the left and a script editor on the right. The terminal displays the output of running a script named `hello.sh`. The script prints "Hello World", the shell name (`/bin/bash`), the shell version (`4.3.11(1)-release`), the home directory (`/home/test`), and the current working directory (`/home/test/Desktop`). It also prints "The name is Mark". The script editor on the right shows the source code of `hello.sh`, which includes comments and uses environment variables like `$BASH`, `$BASH_VERSION`, `$HOME`, and `$PWD` to display system information. It also sets a variable `name=Mark` and prints it.

```
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$
```

```
hello.sh
1  #!/bin/bash
2  # this is a comment
3  echo "Hello World" # this is also a comment
4
5  echo Our shell name is $BASH
6  echo Our shell version name is $BASH_VERSION
7  echo Our home directory is $HOME
8  echo $PWD
9
10 name=Mark
11 echo The name is $name
```

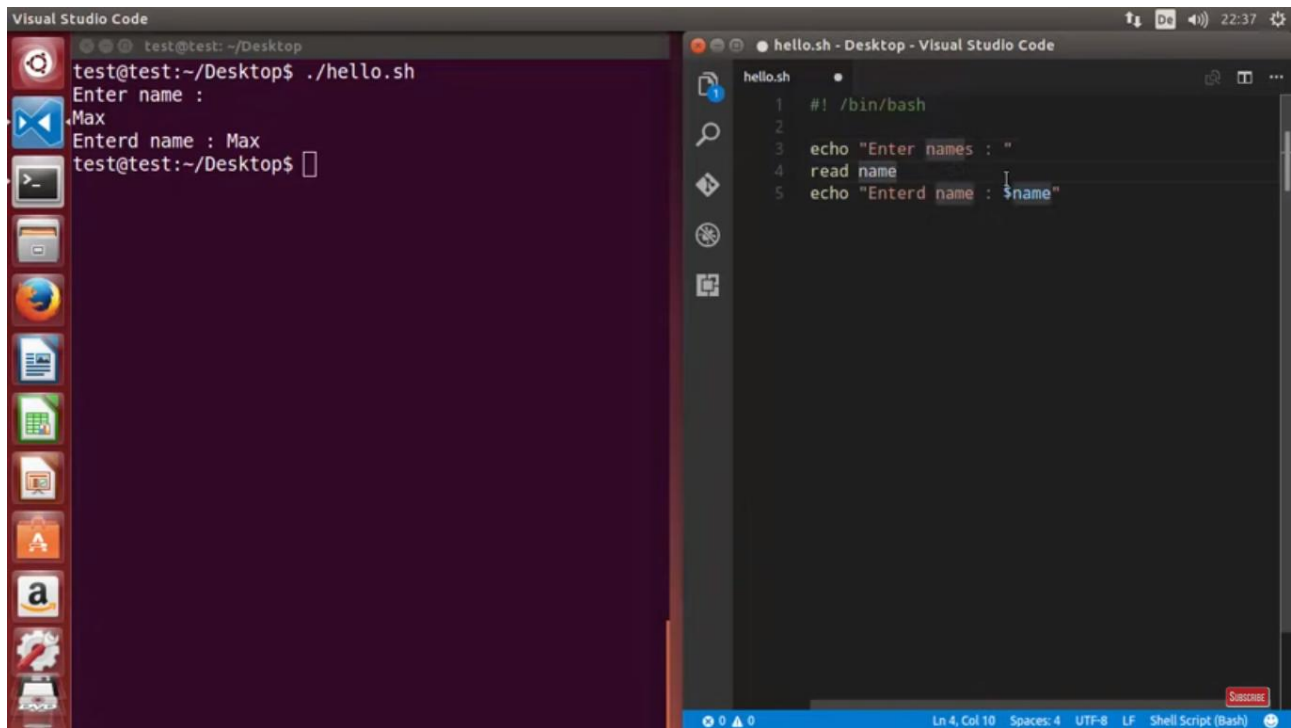


The image shows a terminal window on the left and a script editor on the right. The terminal displays the output of running the script `hello.sh` multiple times. The script prints system information and the value of the `name` variable. It also shows an error message: `./hello.sh: line 11: 10val=10: command not found`. The script editor on the right shows the source code of `hello.sh`, which includes comments and uses environment variables like `$BASH`, `$BASH_VERSION`, `$HOME`, and `$PWD` to display system information. It also sets a variable `name=Mark` and prints it. The script also includes a line `V=10` and a line `echo value $val`.

```
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$
```

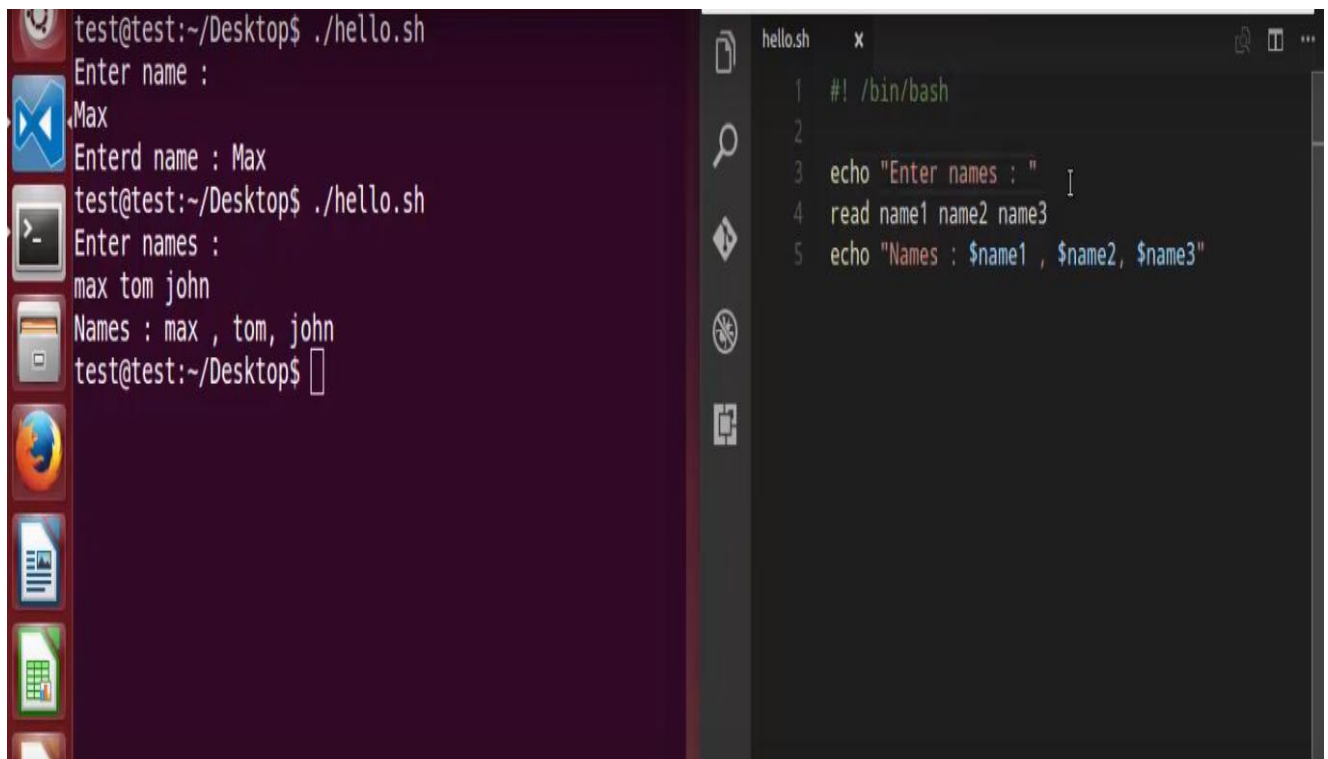
```
hello.sh
1  #!/bin/bash
2  # this is a comment
3  echo "Hello World" # this is also a comment
4
5  echo Our shell name is $BASH
6  echo Our shell version name is $BASH_VERSION
7  echo Our home directory is $HOME
8  echo Our current working directory is $PWD
9
10 name=Mark
11 V=10
12 echo The name is $name
13 echo value $val
```

Read User Input



The screenshot shows the Visual Studio Code interface. On the left, a terminal window displays the execution of a script named `hello.sh`. The prompt is `test@test:~/Desktop$./hello.sh`. The script prompts for a name: `Enter name :`. The user enters `Max`. The script then prompts for a name: `Enterd name : Max`. The prompt is misspelled as `Enterd`. The terminal shows the user's input `Max` and the prompt `test@test:~/Desktop$`. On the right, the script file `hello.sh` is open, showing the following code:

```
1 #!/bin/bash
2
3 echo "Enter names : "
4 read name
5 echo "Enterd name : $name"
```



The screenshot shows the Visual Studio Code interface. On the left, a terminal window displays the execution of a script named `hello.sh`. The prompt is `test@test:~/Desktop$./hello.sh`. The script prompts for names: `Enter names :`. The user enters `max tom john`. The script then prompts for names: `Names : max , tom, john`. The prompt is misspelled as `Names`. The terminal shows the user's input `max tom john` and the prompt `test@test:~/Desktop$`. On the right, the script file `hello.sh` is open, showing the following code:

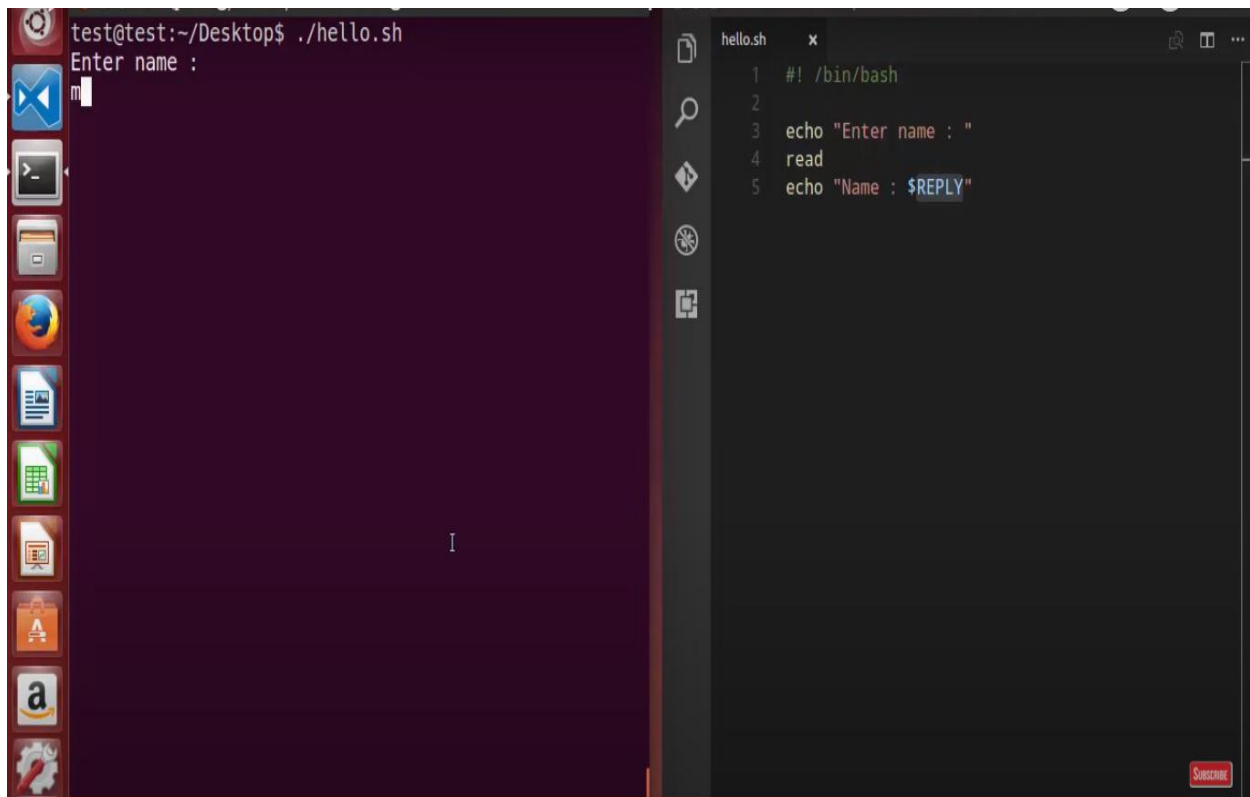
```
1 #!/bin/bash
2
3 echo "Enter names : "
4 read name1 name2 name3
5 echo "Names : $name1 , $name2, $name3"
```

The image shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a script named `hello.sh`. The user enters 'Max' for the name, 'max tom john' for names, and 'myuser' for the username. The password is entered as '12345'. The code editor shows the script content:

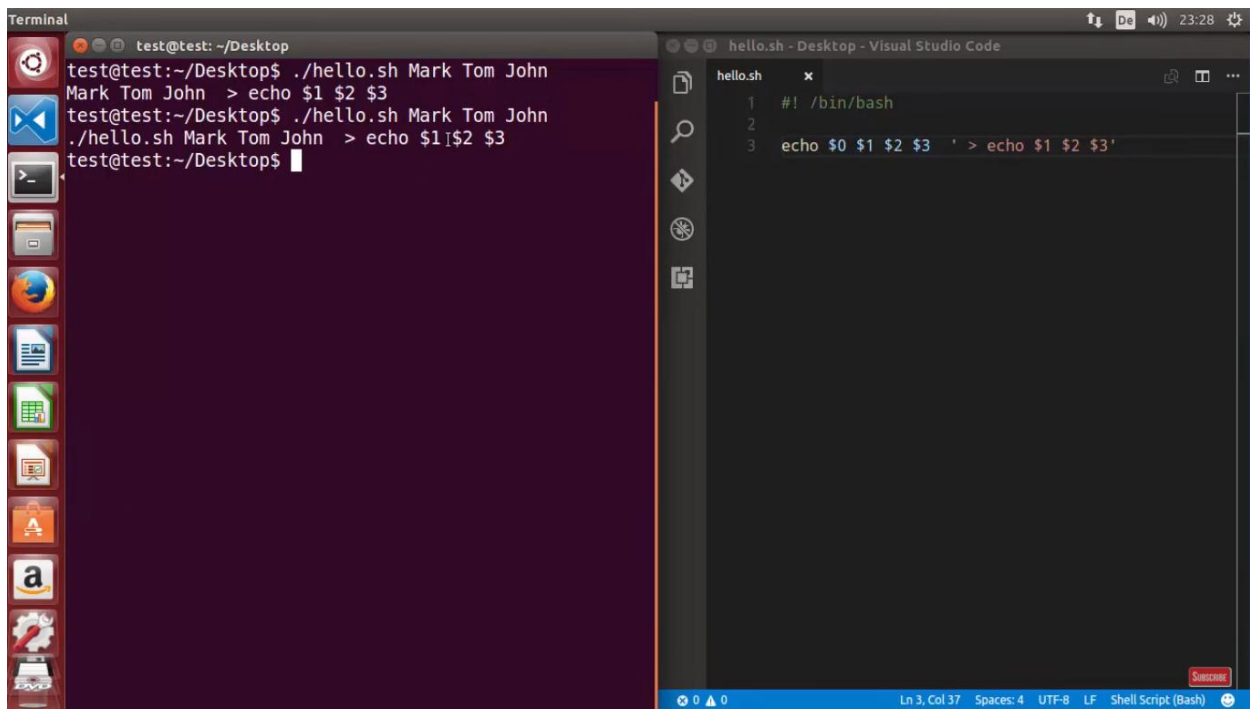
```
hello.sh x
1  #!/bin/bash
2
3  read -p 'username : ' user_var
4  read -sp 'password : ' pass_var
5  echo "username : $user_var"
6  echo "password : $pass_var"
```

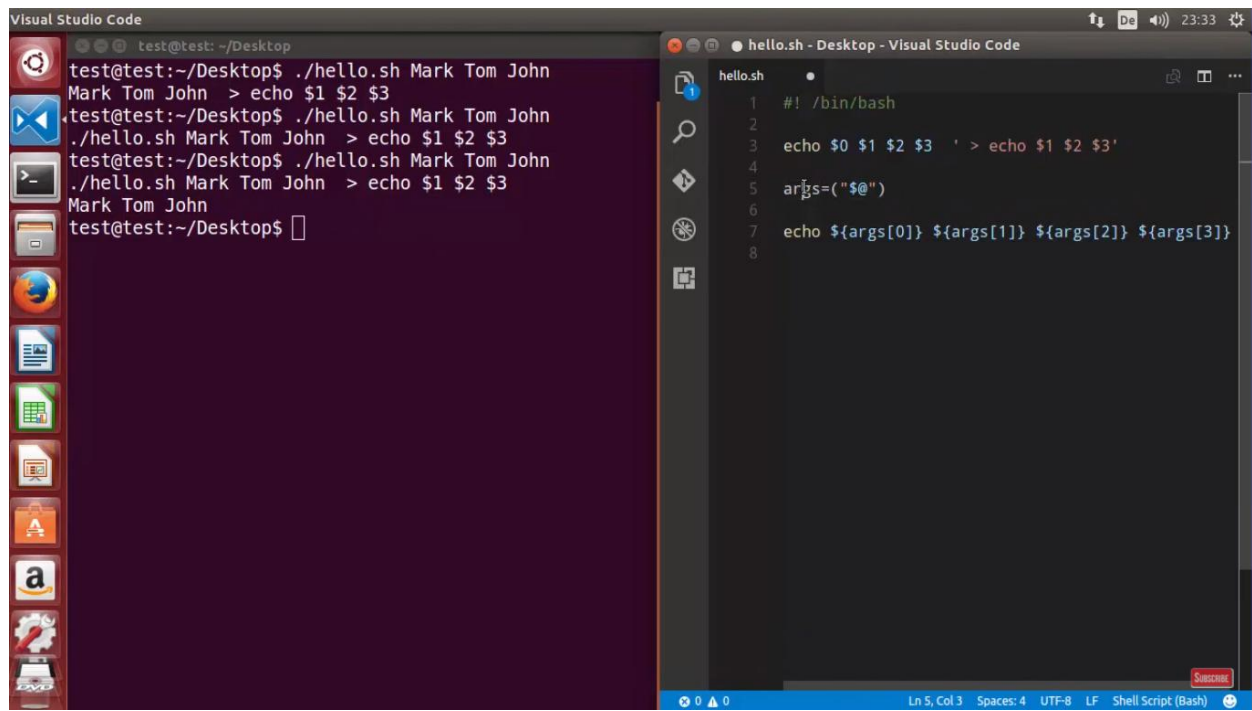
The image shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a script named `hello.sh`. The user enters 'Max' for the name, 'max tom john' for names, and 'myuser' for the username. The password is entered as '12345'. The code editor shows the script content:

```
hello.sh x
1  #!/bin/bash
2
3  echo "Enter names : "
4  read -a names
5  echo "Names : ${names[0]}, ${names[1]}"
```

Passing Arguments to Bash Script





Visual Studio Code

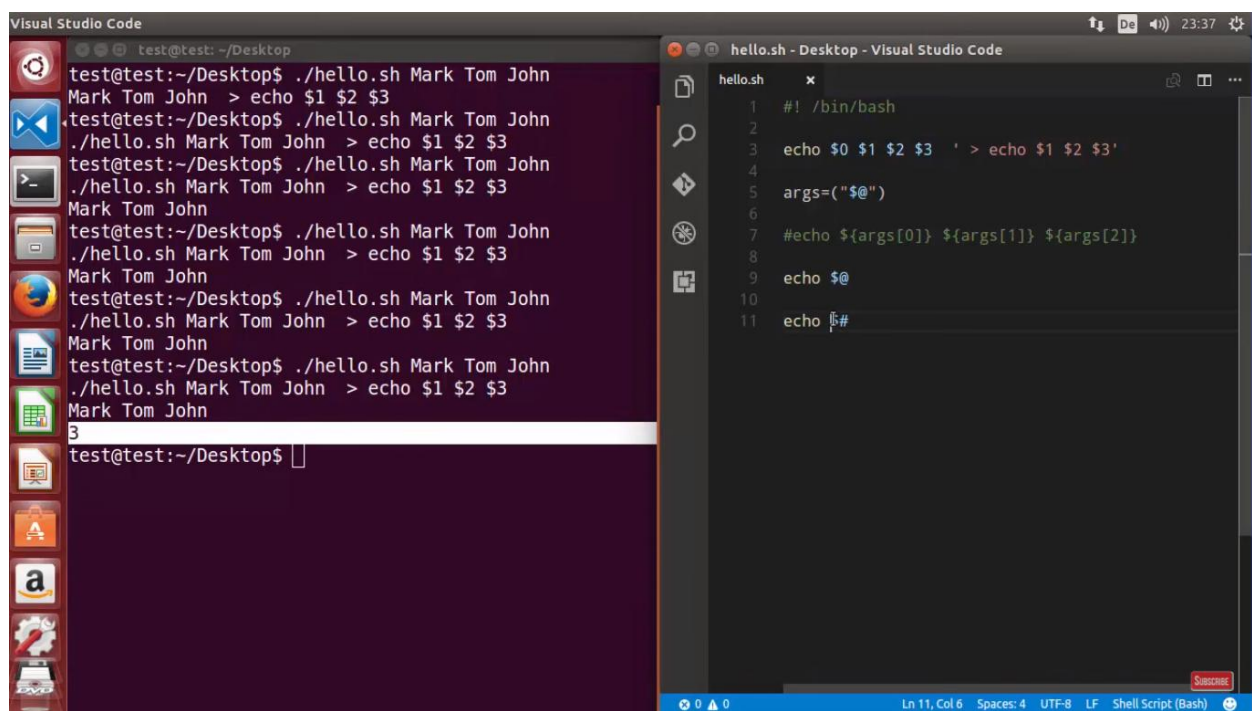
test@test: ~/Desktop

```
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 - Desktop - Visual Studio Code

```
hello.sh
1  #!/bin/bash
2
3  echo $0 $1 $2 $3 ' > echo $1 $2 $3'
4
5  args=("$@")
6
7  echo ${args[0]} ${args[1]} ${args[2]} ${args[3]}
8
```

Ln 5, Col 3 Spaces: 4 UTF-8 LF Shell Script (Bash)



Visual Studio Code

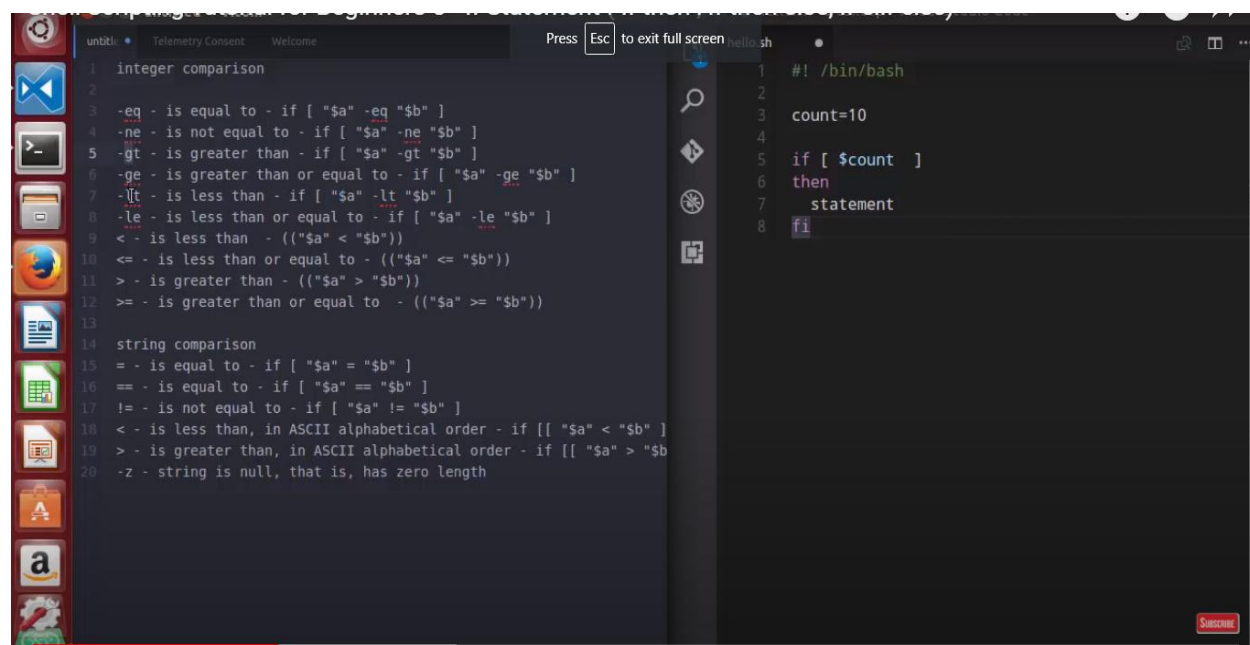
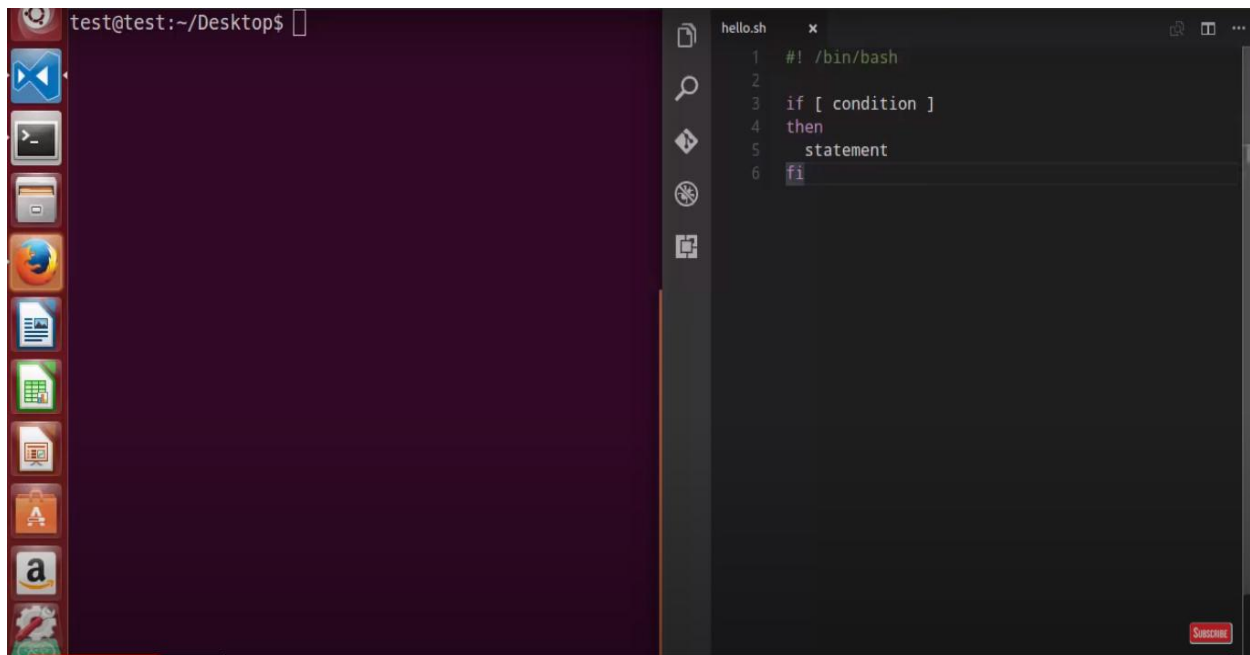
test@test: ~/Desktop

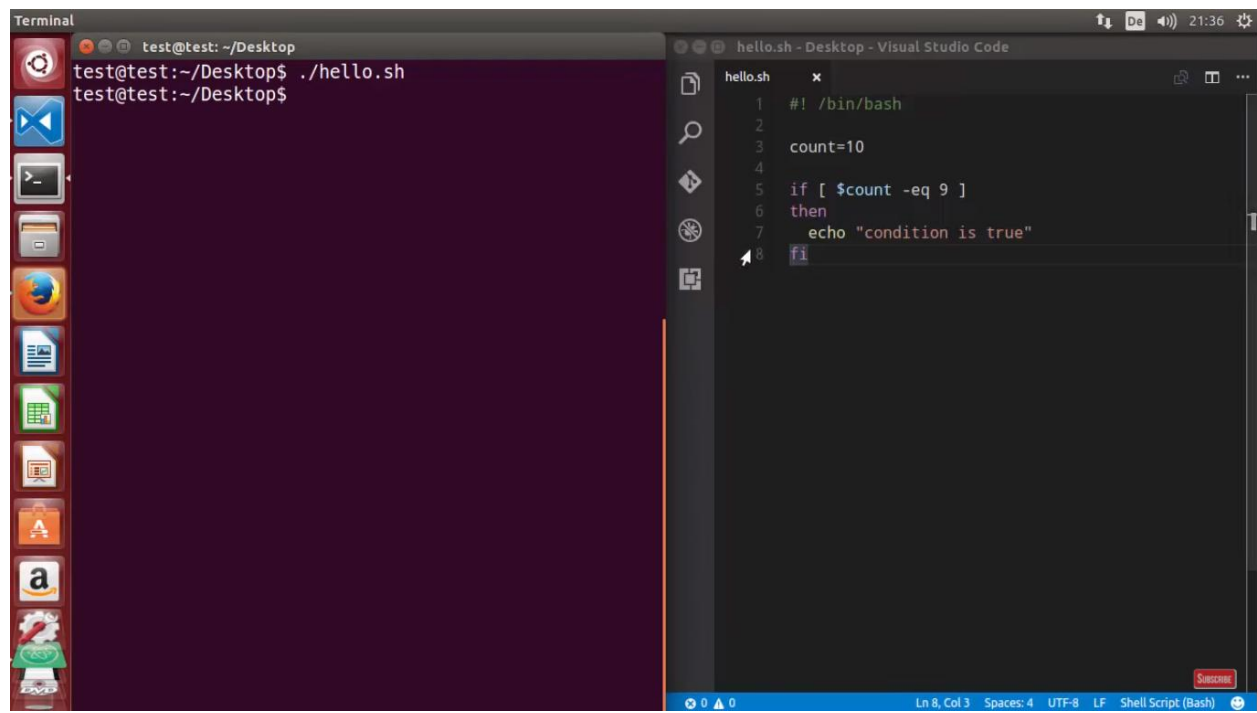
```
test@test:~/Desktop$ ./hello.sh Mark Tom John
Mark Tom John > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John > echo $1 $2 $3
test@test:~/Desktop$ ./hello.sh Mark Tom John
./hello.sh Mark Tom John > echo $1 $2 $3
Mark Tom John
test@test:~/Desktop$ ./hello.sh Mark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

hello.sh - Desktop - Visual Studio Code

```
hello.sh
1  #!/bin/bash
2
3  echo $0 $1 $2 $3 ' > echo $1 $2 $3'
4
5  args=("$@")
6
7  #echo ${args[0]} ${args[1]} ${args[2]}
8
9  echo $@
10
11 echo $#
```

Ln 11, Col 6 Spaces: 4 UTF-8 LF Shell Script (Bash)

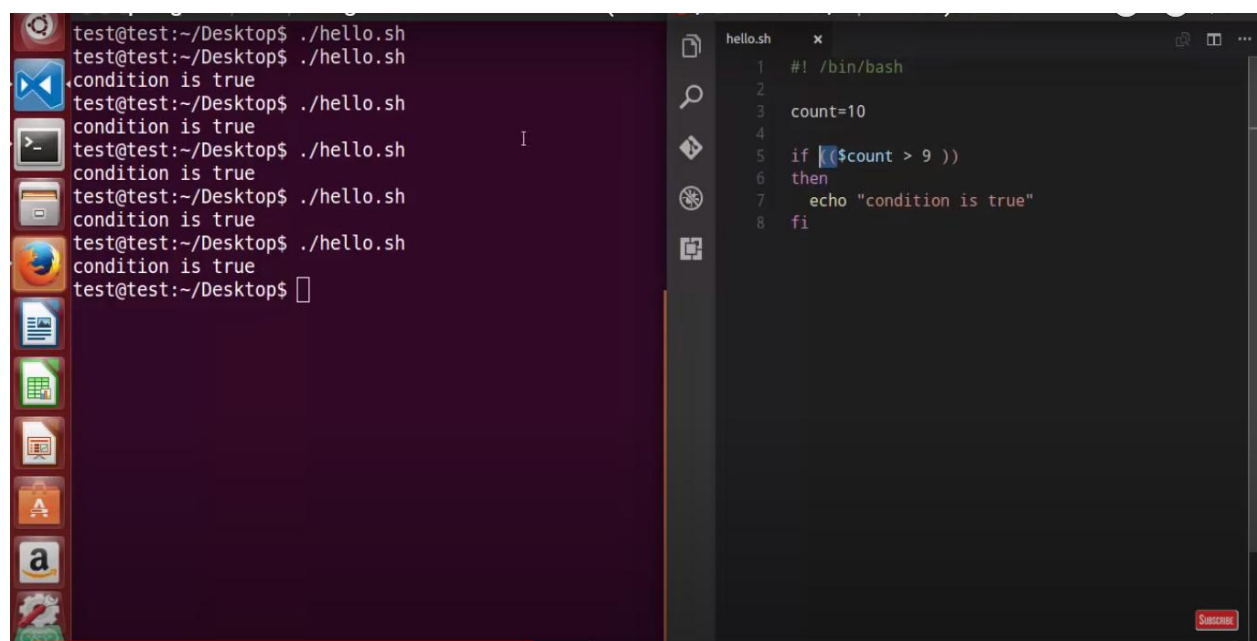




This screenshot shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal window has a title bar that says "Terminal" and a prompt "test@test: ~/Desktop". The user has entered the command `./hello.sh` twice, and the prompt is now `test@test:~/Desktop$`. The Visual Studio Code editor has a title bar that says "hello.sh - Desktop - Visual Studio Code". The editor shows a shell script named `hello.sh` with the following content:

```
1  #!/bin/bash
2
3  count=10
4
5  if [ $count -eq 9 ]
6  then
7      echo "condition is true"
8  fi
```

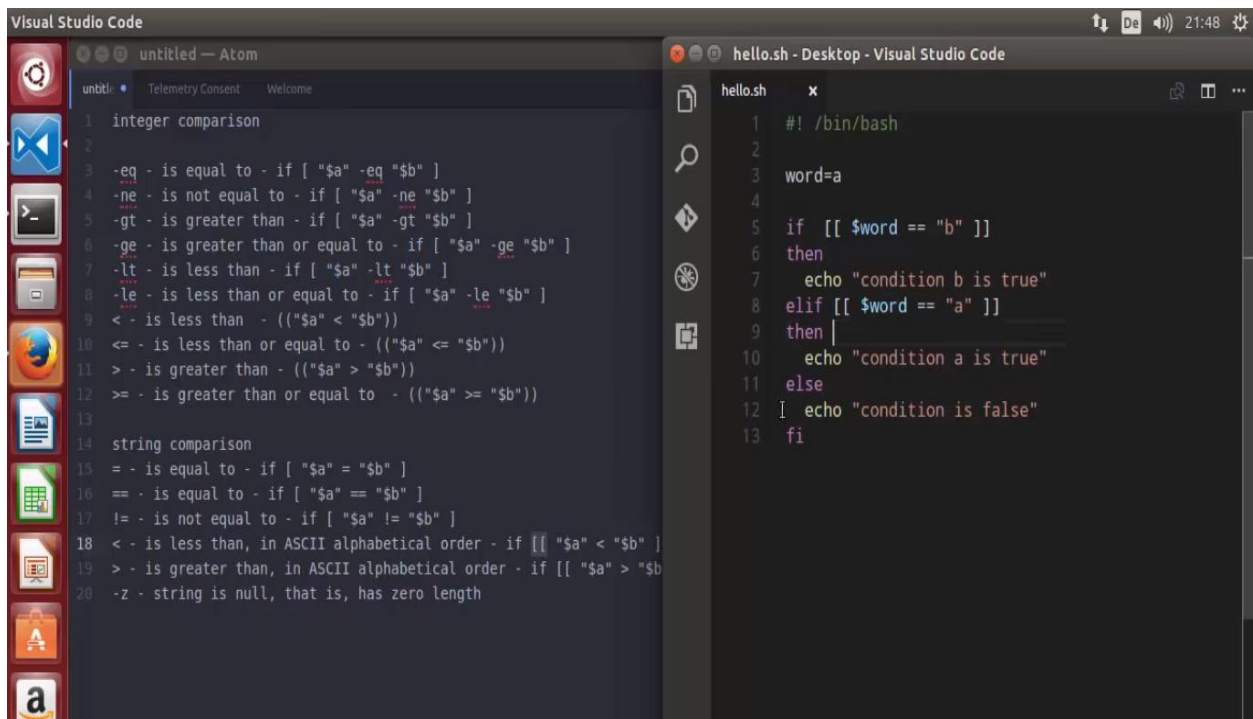
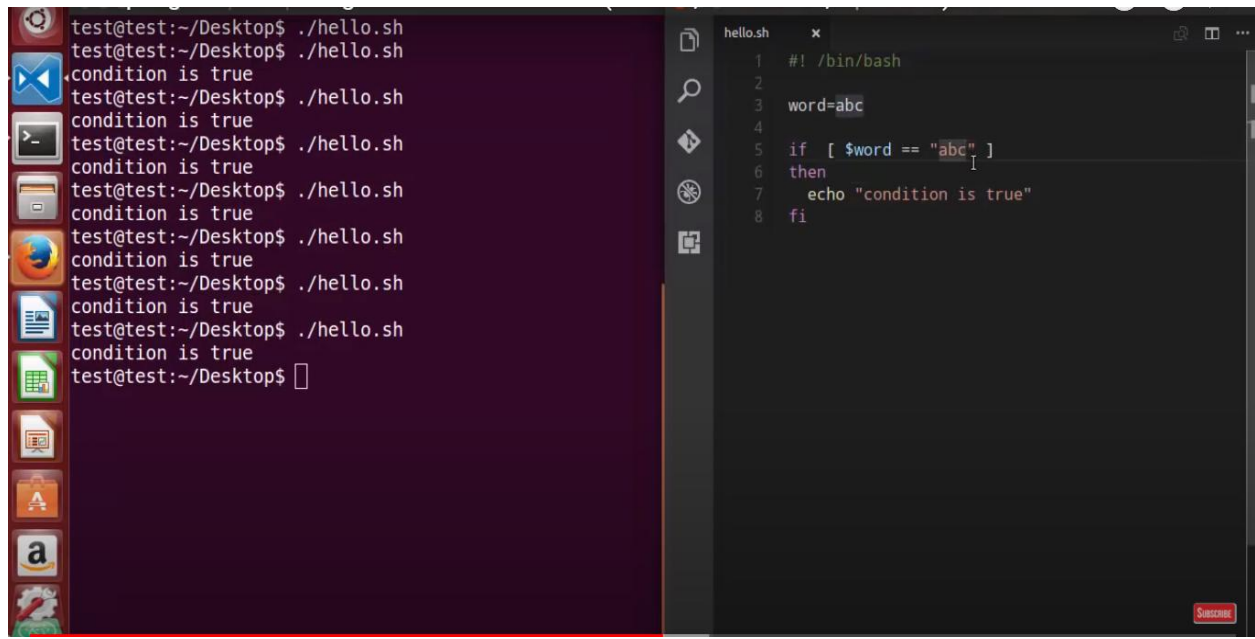
The status bar at the bottom of the editor shows "Ln 8, Col 3", "Spaces: 4", "UTF-8", "LF", and "Shell Script (Bash)".



This screenshot shows the same terminal and VS Code editor setup as the first screenshot. The terminal window now shows the output of the `./hello.sh` command being executed multiple times. The output is:

```
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$
```

The Visual Studio Code editor shows the same shell script as before, but with a modification on line 5: `if [($count > 9)]`. The status bar at the bottom of the editor shows "Ln 8, Col 3", "Spaces: 4", "UTF-8", "LF", and "Shell Script (Bash)".



Visual Studio Code

Shell Scripting Tutorial for Beginners 6 - File test operators

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\$

```
hello.sh
1  #!/bin/bash
2
3  echo -e "Enter the name of the file : \c"
4  read file_name
5
6  if [ -e $file_name ]
7  then
8      echo "$file_name found"
9  else
10     echo "$file_name not found"
11 fi
12
```

4:50 / 10:29

Visual Studio Code

Shell Scripting Tutorial for Beginners 7 - How to append output to the end of text file

test@test:~/Desktop\$

```
hello.sh
1  #!/bin/bash
2
3  echo -e "Enter the name of the file : \c"
4  read file_name
5
6  if [ -f $file_name ]
7  then
8      if [ -w $file_name ]
9      then
10         echo "Type some text data. To quit press ctrl+d."
11         cat >> $file_name
12     else
13         echo "The file do not have write permissions"
14     fi
15 else
16     echo "$file_name not exists"
17 fi
18
```

5:46 / 10:46

```
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$
```

```
#!/bin/bash

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

[ -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"

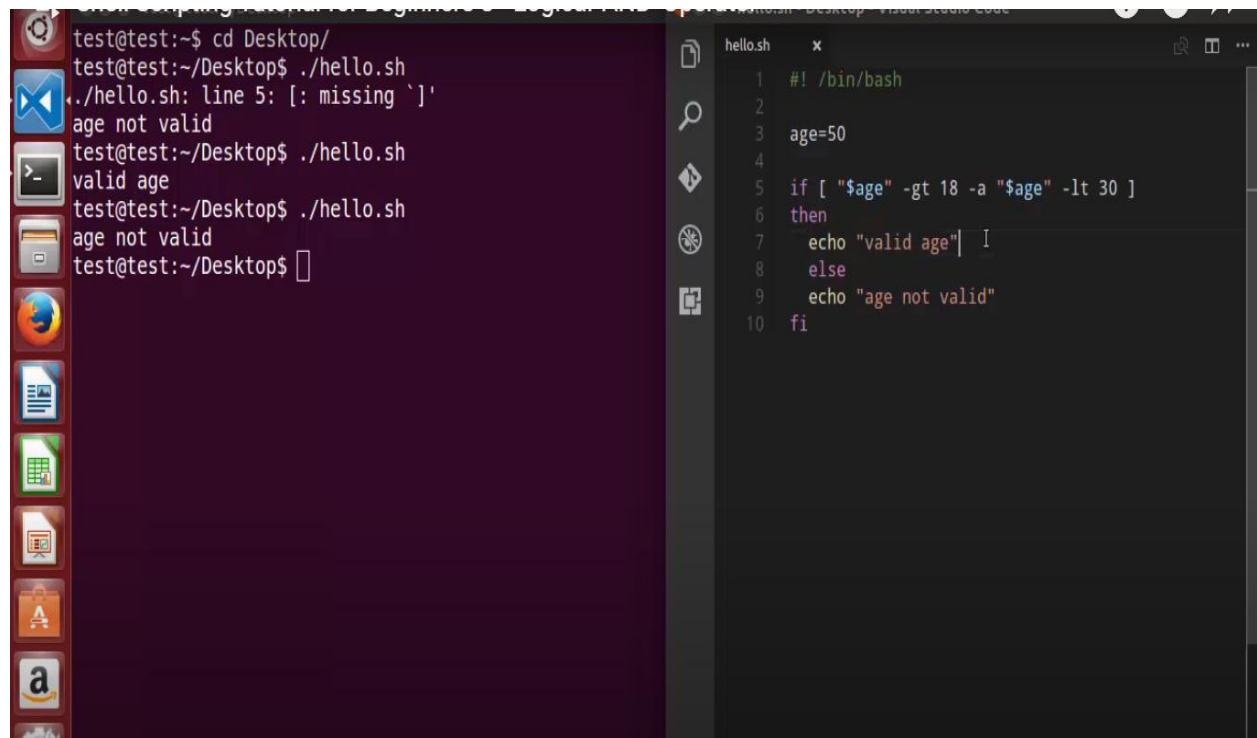
$file_name not exists"
```

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

```
#!/bin/bash

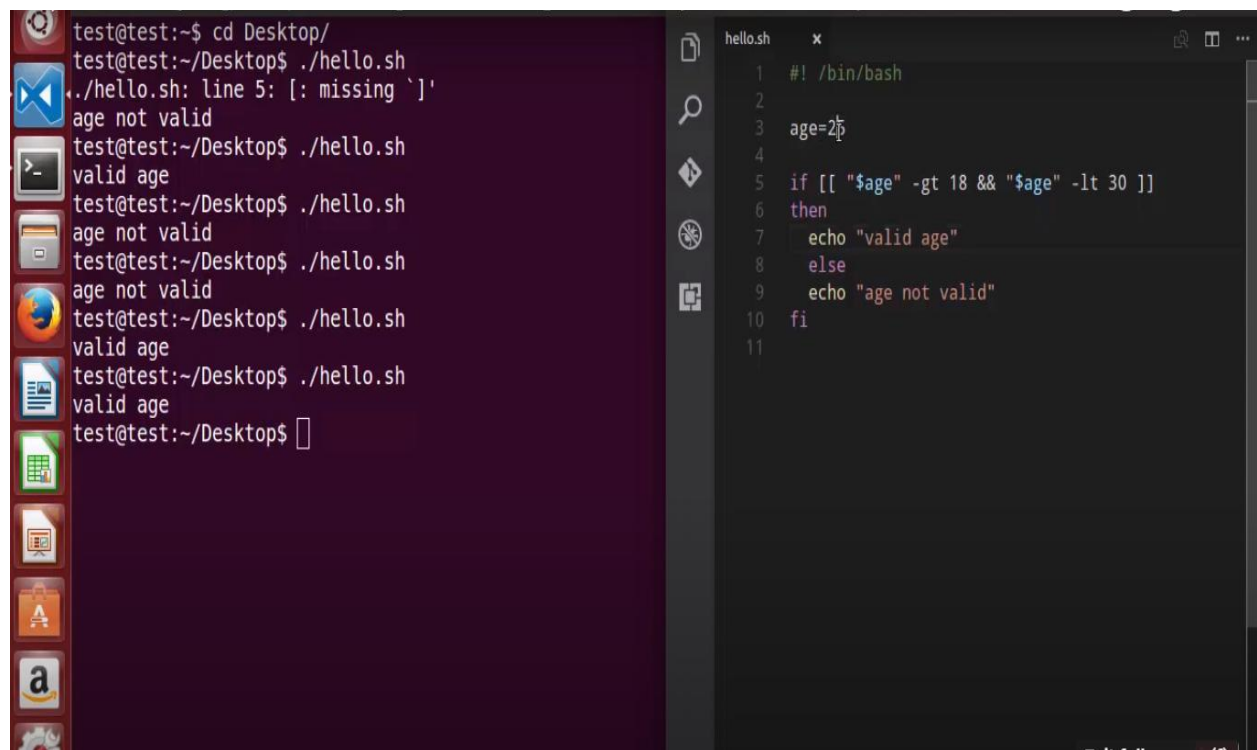
age=25

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



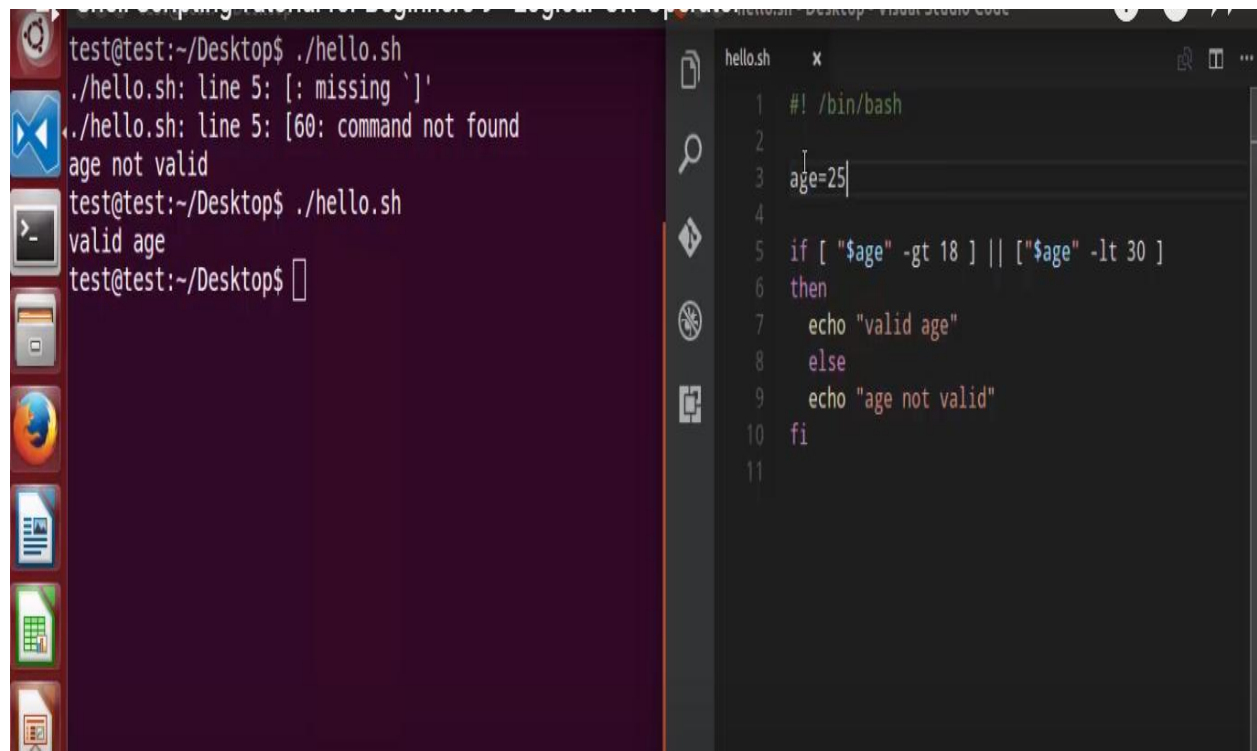
The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the execution of a script named `hello.sh`. The first run results in a syntax error: `./hello.sh: line 5: [: missing `']`, followed by the output `age not valid`. The second run, after a correction, outputs `valid age`. The third run, with another correction, outputs `age not valid`. The VS Code editor shows the script content:

```
hello.sh x
1  #!/bin/bash
2
3  age=50
4
5  if [ "$age" -gt 18 -a "$age" -lt 30 ]
6  then
7      echo "valid age" | I
8  else
9      echo "age not valid"
10 fi
```



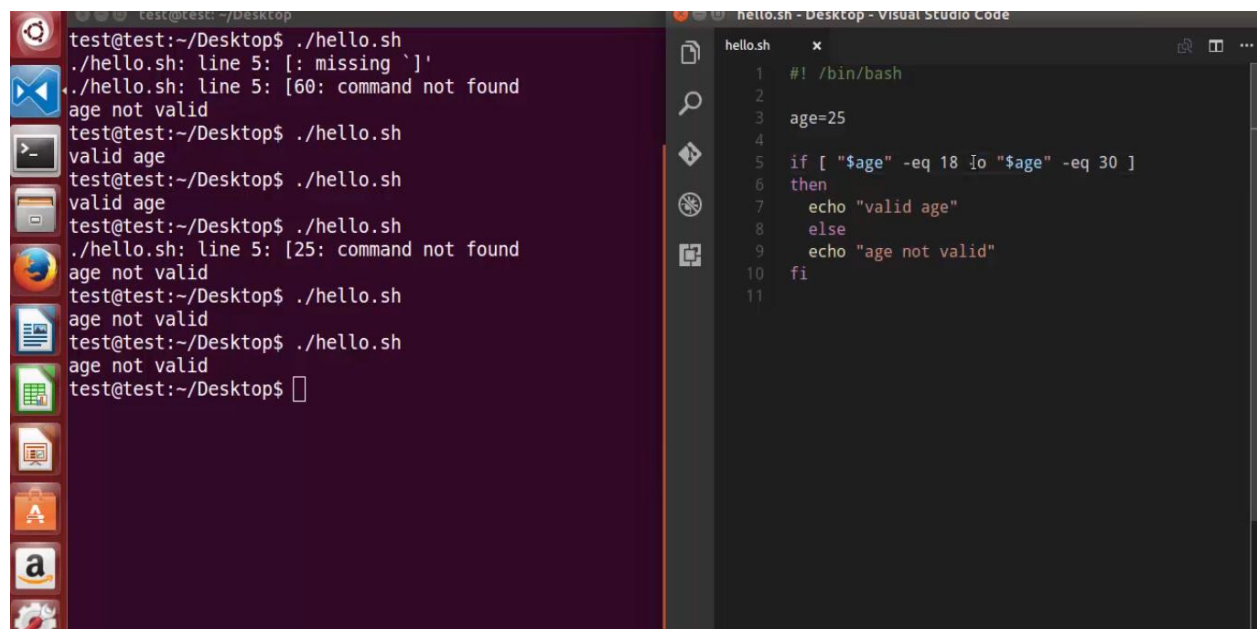
The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the execution of the script `hello.sh` with various age inputs. The first run (age=50) results in a syntax error and `age not valid`. The second run (age=20) results in `valid age`. The third run (age=15) results in `age not valid`. The fourth run (age=35) results in `age not valid`. The fifth run (age=25) results in `valid age`. The sixth run (age=20) results in `valid age`. The VS Code editor shows the script content with logical operator corrections:

```
hello.sh x
1  #!/bin/bash
2
3  age=20
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
```

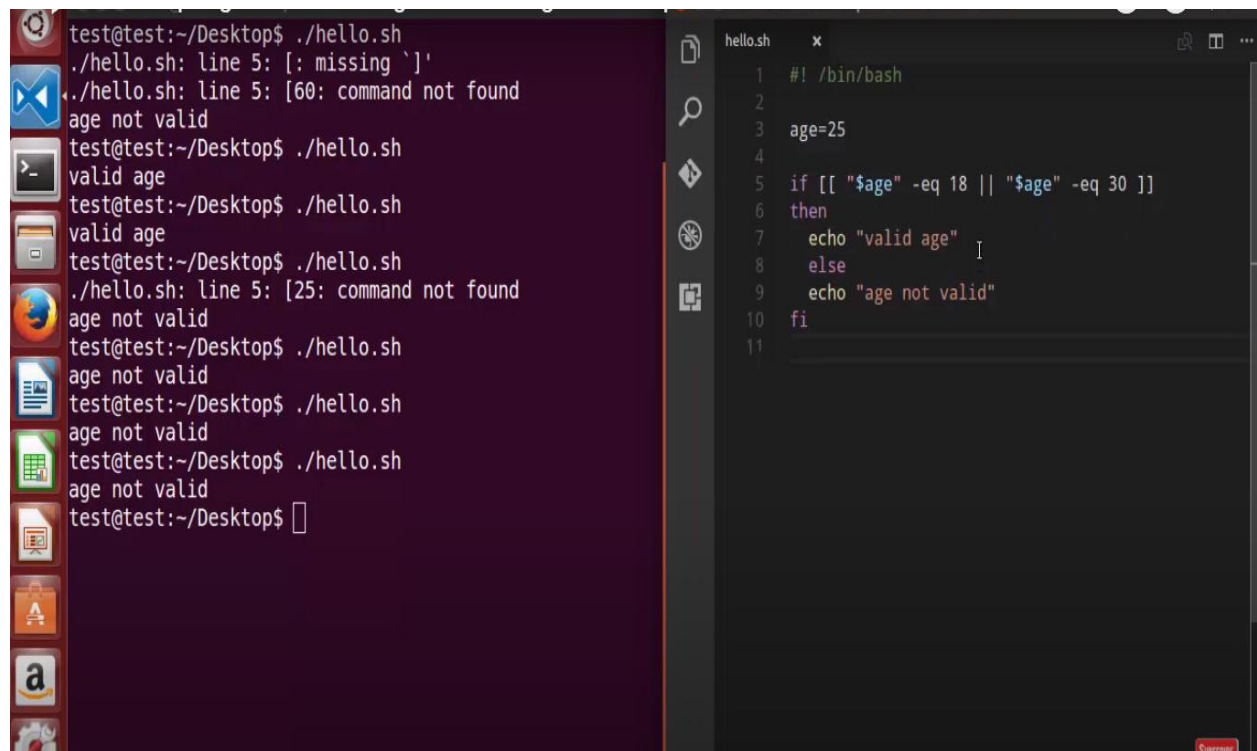
The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the execution of a script named `hello.sh`. The first run results in a syntax error: `./hello.sh: line 5: [: missing `']`. The second run results in a command not found error: `./hello.sh: line 5: [60: command not found`. The third run results in the output `age not valid`. The fourth run results in the output `valid age`. The Visual Studio Code editor shows the script `hello.sh` with the following content:

```
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
```



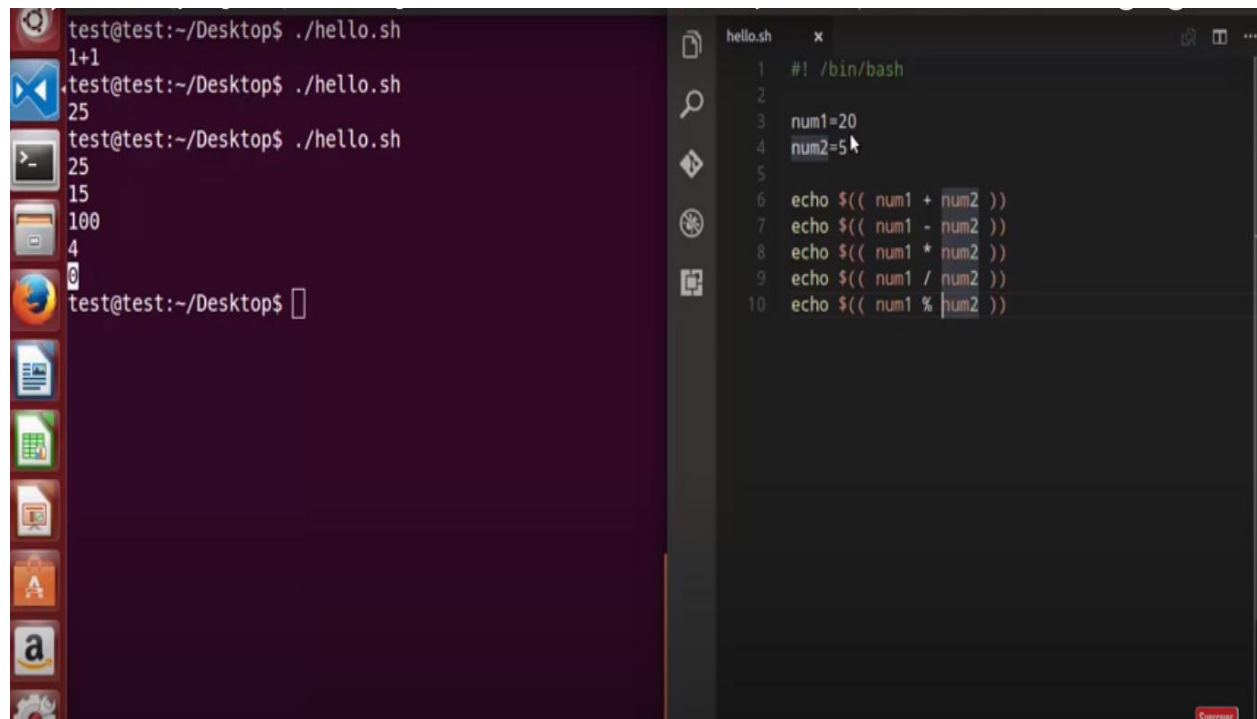
The image shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the execution of a script named `hello.sh`. The first run results in a syntax error: `./hello.sh: line 5: [: missing `']`. The second run results in a command not found error: `./hello.sh: line 5: [60: command not found`. The third run results in the output `age not valid`. The fourth run results in the output `valid age`. The fifth run results in the output `valid age`. The sixth run results in a command not found error: `./hello.sh: line 5: [25: command not found`. The seventh run results in the output `age not valid`. The eighth run results in the output `age not valid`. The ninth run results in the output `age not valid`. The tenth run results in the output `age not valid`. The Visual Studio Code editor shows the script `hello.sh` with the following content:

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



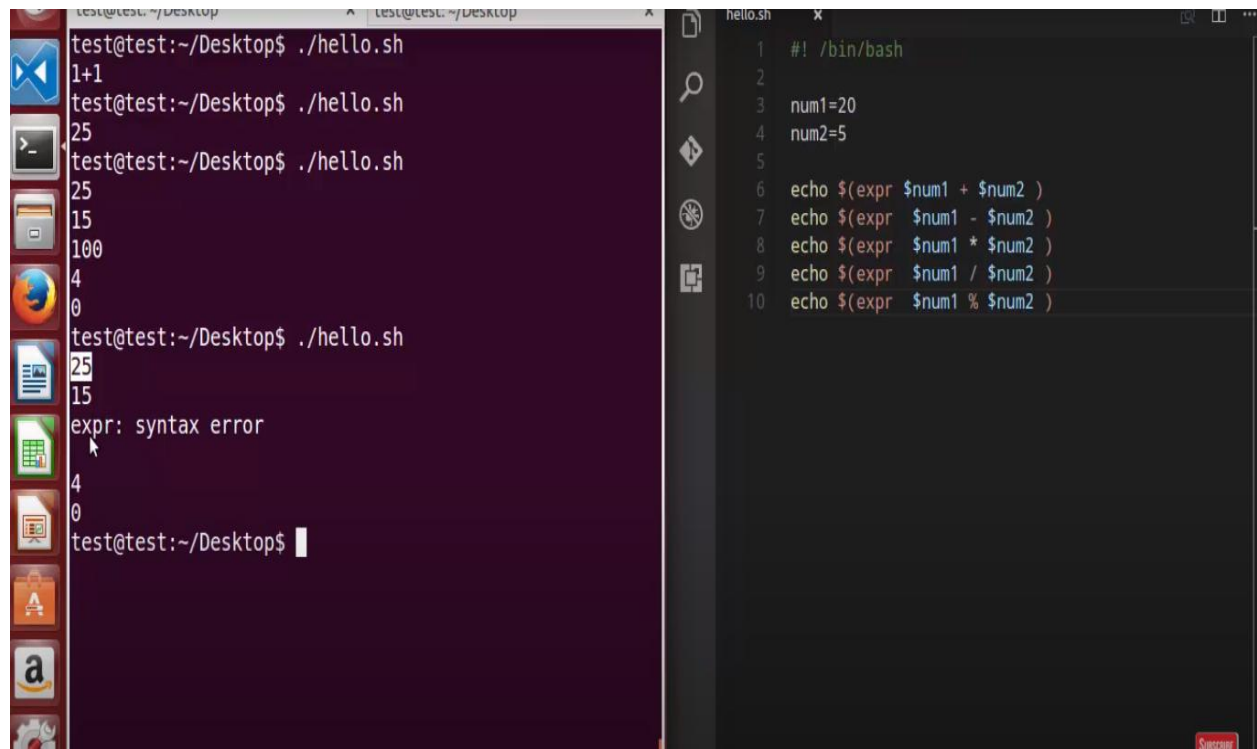
The image shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a script named `hello.sh`. The first run results in an error: `./hello.sh: line 5: [: missing `']`. The second run results in `age not valid`. The third run results in `valid age`. The fourth run results in `age not valid`. The fifth run results in `age not valid`. The sixth run results in `age not valid`. The seventh run results in `age not valid`. The eighth run results in `age not valid`. The ninth run results in `age not valid`. The tenth run results in `age not valid`. The eleventh run results in `age not valid`. The twelfth run results in `age not valid`. The thirteenth run results in `age not valid`. The fourteenth run results in `age not valid`. The fifteenth run results in `age not valid`. The sixteenth run results in `age not valid`. The seventeenth run results in `age not valid`. The eighteenth run results in `age not valid`. The nineteenth run results in `age not valid`. The twentieth run results in `age not valid`. The twenty-first run results in `age not valid`. The twenty-second run results in `age not valid`. The twenty-third run results in `age not valid`. The twenty-fourth run results in `age not valid`. The twenty-fifth run results in `age not valid`. The twenty-sixth run results in `age not valid`. The twenty-seventh run results in `age not valid`. The twenty-eighth run results in `age not valid`. The twenty-ninth run results in `age not valid`. The thirtieth run results in `age not valid`. The thirty-first run results in `age not valid`. The thirty-second run results in `age not valid`. The thirty-third run results in `age not valid`. The thirty-fourth run results in `age not valid`. The thirty-fifth run results in `age not valid`. The thirty-sixth run results in `age not valid`. The thirty-seventh run results in `age not valid`. The thirty-eighth run results in `age not valid`. The thirty-ninth run results in `age not valid`. The fortieth run results in `age not valid`. The forty-first run results in `age not valid`. The forty-second run results in `age not valid`. The forty-third run results in `age not valid`. The forty-fourth run results in `age not valid`. The forty-fifth run results in `age not valid`. The forty-sixth run results in `age not valid`. The forty-seventh run results in `age not valid`. The forty-eighth run results in `age not valid`. The forty-ninth run results in `age not valid`. The fiftieth run results in `age not valid`. The fifty-first run results in `age not valid`. The fifty-second run results in `age not valid`. The fifty-third run results in `age not valid`. The fifty-fourth run results in `age not valid`. The fifty-fifth run results in `age not valid`. The fifty-sixth run results in `age not valid`. The fifty-seventh run results in `age not valid`. The fifty-eighth run results in `age not valid`. The fifty-ninth run results in `age not valid`. The sixtieth run results in `age not valid`. The sixty-first run results in `age not valid`. The sixty-second run results in `age not valid`. The sixty-third run results in `age not valid`. The sixty-fourth run results in `age not valid`. The sixty-fifth run results in `age not valid`. The sixty-sixth run results in `age not valid`. The sixty-seventh run results in `age not valid`. The sixty-eighth run results in `age not valid`. The sixty-ninth run results in `age not valid`. The seventieth run results in `age not valid`. The seventy-first run results in `age not valid`. The seventy-second run results in `age not valid`. The seventy-third run results in `age not valid`. The seventy-fourth run results in `age not valid`. The seventy-fifth run results in `age not valid`. The seventy-sixth run results in `age not valid`. The seventy-seventh run results in `age not valid`. The seventy-eighth run results in `age not valid`. The seventy-ninth run results in `age not valid`. The eightieth run results in `age not valid`. The eighty-first run results in `age not valid`. The eighty-second run results in `age not valid`. The eighty-third run results in `age not valid`. The eighty-fourth run results in `age not valid`. The eighty-fifth run results in `age not valid`. The eighty-sixth run results in `age not valid`. The eighty-seventh run results in `age not valid`. The eighty-eighth run results in `age not valid`. The eighty-ninth run results in `age not valid`. The ninetieth run results in `age not valid`. The ninety-first run results in `age not valid`. The ninety-second run results in `age not valid`. The ninety-third run results in `age not valid`. The ninety-fourth run results in `age not valid`. The ninety-fifth run results in `age not valid`. The ninety-sixth run results in `age not valid`. The ninety-seventh run results in `age not valid`. The ninety-eighth run results in `age not valid`. The ninety-ninth run results in `age not valid`. The hundredth run results in `age not valid`. The code editor shows the following script:

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



The image shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a script named `hello.sh`. The first run results in `1+1`. The second run results in `25`. The third run results in `25`. The fourth run results in `15`. The fifth run results in `100`. The sixth run results in `4`. The seventh run results in `0`. The eighth run results in `0`. The ninth run results in `0`. The tenth run results in `0`. The eleventh run results in `0`. The twelfth run results in `0`. The thirteenth run results in `0`. The fourteenth run results in `0`. The fifteenth run results in `0`. The sixteenth run results in `0`. The seventeenth run results in `0`. The eighteenth run results in `0`. The nineteenth run results in `0`. The twentieth run results in `0`. The twenty-first run results in `0`. The twenty-second run results in `0`. The twenty-third run results in `0`. The twenty-fourth run results in `0`. The twenty-fifth run results in `0`. The twenty-sixth run results in `0`. The twenty-seventh run results in `0`. The twenty-eighth run results in `0`. The twenty-ninth run results in `0`. The thirtieth run results in `0`. The thirty-first run results in `0`. The thirty-second run results in `0`. The thirty-third run results in `0`. The thirty-fourth run results in `0`. The thirty-fifth run results in `0`. The thirty-sixth run results in `0`. The thirty-seventh run results in `0`. The thirty-eighth run results in `0`. The thirty-ninth run results in `0`. The fortieth run results in `0`. The forty-first run results in `0`. The forty-second run results in `0`. The forty-third run results in `0`. The forty-fourth run results in `0`. The forty-fifth run results in `0`. The forty-sixth run results in `0`. The forty-seventh run results in `0`. The forty-eighth run results in `0`. The forty-ninth run results in `0`. The fiftieth run results in `0`. The fifty-first run results in `0`. The fifty-second run results in `0`. The fifty-third run results in `0`. The fifty-fourth run results in `0`. The fifty-fifth run results in `0`. The fifty-sixth run results in `0`. The fifty-seventh run results in `0`. The fifty-eighth run results in `0`. The fifty-ninth run results in `0`. The sixtieth run results in `0`. The sixty-first run results in `0`. The sixty-second run results in `0`. The sixty-third run results in `0`. The sixty-fourth run results in `0`. The sixty-fifth run results in `0`. The sixty-sixth run results in `0`. The sixty-seventh run results in `0`. The sixty-eighth run results in `0`. The sixty-ninth run results in `0`. The seventieth run results in `0`. The seventy-first run results in `0`. The seventy-second run results in `0`. The seventy-third run results in `0`. The seventy-fourth run results in `0`. The seventy-fifth run results in `0`. The seventy-sixth run results in `0`. The seventy-seventh run results in `0`. The seventy-eighth run results in `0`. The seventy-ninth run results in `0`. The eightieth run results in `0`. The eighty-first run results in `0`. The eighty-second run results in `0`. The eighty-third run results in `0`. The eighty-fourth run results in `0`. The eighty-fifth run results in `0`. The eighty-sixth run results in `0`. The eighty-seventh run results in `0`. The eighty-eighth run results in `0`. The eighty-ninth run results in `0`. The ninetieth run results in `0`. The ninety-first run results in `0`. The ninety-second run results in `0`. The ninety-third run results in `0`. The ninety-fourth run results in `0`. The ninety-fifth run results in `0`. The ninety-sixth run results in `0`. The ninety-seventh run results in `0`. The ninety-eighth run results in `0`. The ninety-ninth run results in `0`. The hundredth run results in `0`. The code editor shows the following script:

```
1 #!/bin/bash
2
3 num1=20
4 num2=5
5
6 echo $(( num1 + num2 ))
7 echo $(( num1 - num2 ))
8 echo $(( num1 * num2 ))
9 echo $(( num1 / num2 ))
10 echo $(( num1 % num2 ))
```

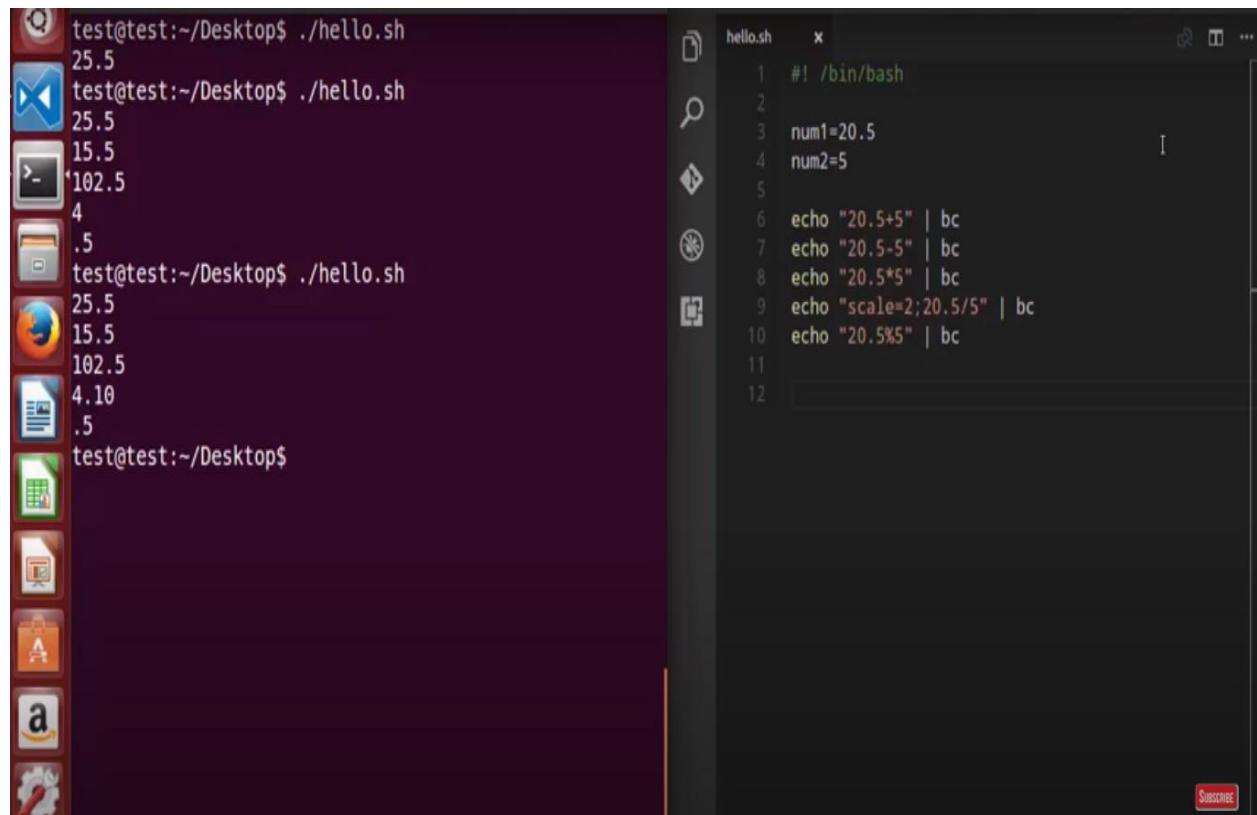


The image shows a terminal window on the left and a script editor on the right. The terminal displays the execution of a script named `hello.sh` multiple times. The first three runs show integer arithmetic: `1+1` results in `1`, `25` results in `25`, and `15` results in `15`. The fourth run shows a syntax error: `expr: syntax error`. The script editor on the right shows the content of `hello.sh`, which uses `expr` for integer arithmetic.

```
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$
```

```
hello.sh
1 #!/bin/bash
2
3 num1=20
4 num2=5
5
6 echo $(expr $num1 + $num2 )
7 echo $(expr $num1 - $num2 )
8 echo $(expr $num1 * $num2 )
9 echo $(expr $num1 / $num2 )
10 echo $(expr $num1 % $num2 )
```

Floating point arithmetic operation:

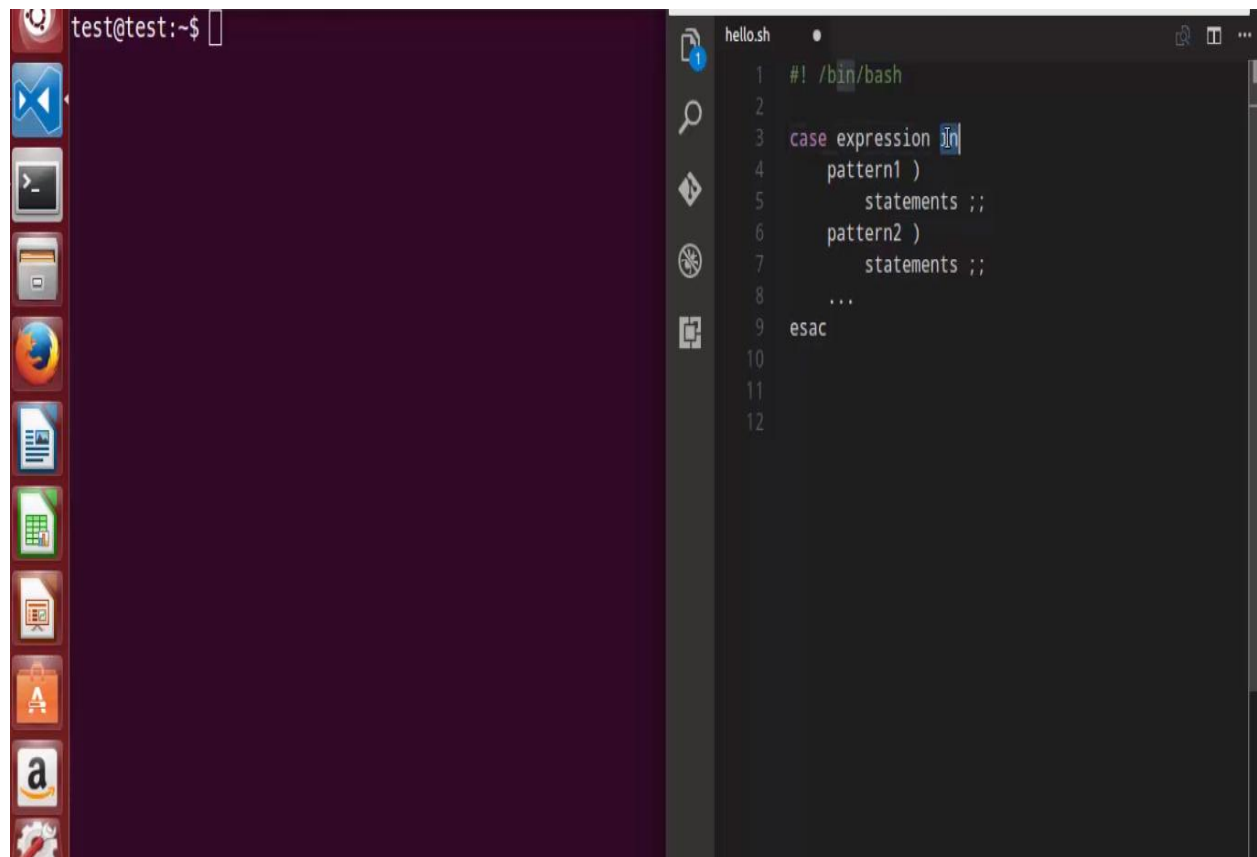


The image shows a terminal window on the left and a script editor on the right. The terminal displays the execution of a script named `hello.sh` multiple times. The first three runs show floating point arithmetic: `25.5` results in `25.5`, `15.5` results in `15.5`, and `102.5` results in `102.5`. The fourth run shows a syntax error: `expr: syntax error`. The script editor on the right shows the content of `hello.sh`, which uses `bc` for floating point arithmetic.

```
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$
```

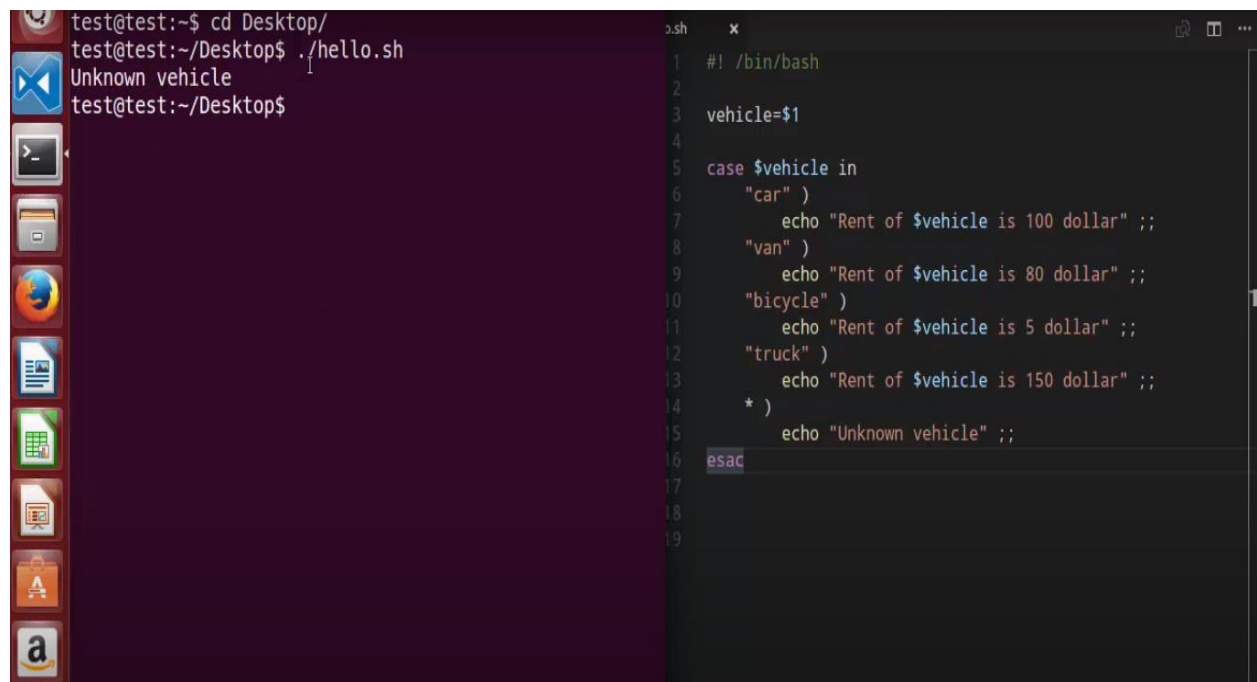
```
hello.sh
1 #!/bin/bash
2
3 num1=20.5
4 num2=5
5
6 echo "20.5+5" | bc
7 echo "20.5-5" | bc
8 echo "20.5*5" | bc
9 echo "scale=2;20.5/5" | bc
10 echo "20.5%5" | bc
11
12
```


Case Statement:



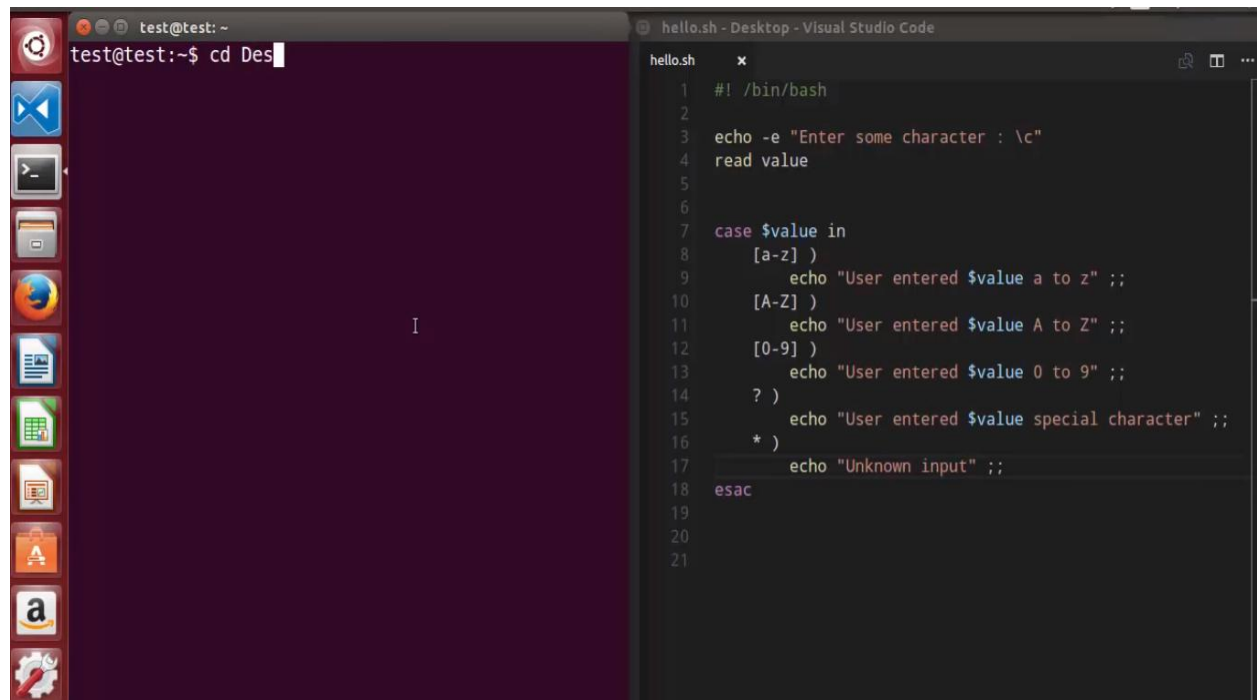
The image shows a terminal window on the left and a script editor on the right. The terminal window has a dark purple background and a vertical sidebar with application icons. The script editor, titled 'hello.sh', has a dark grey background and shows the basic syntax of a case statement.

```
test@test:~$  
  
hello.sh  
1  #!/bin/bash  
2  
3  case expression in  
4      pattern1 )  
5          statements ;;  
6      pattern2 )  
7          statements ;;  
8      ...  
9  esac  
10  
11  
12
```

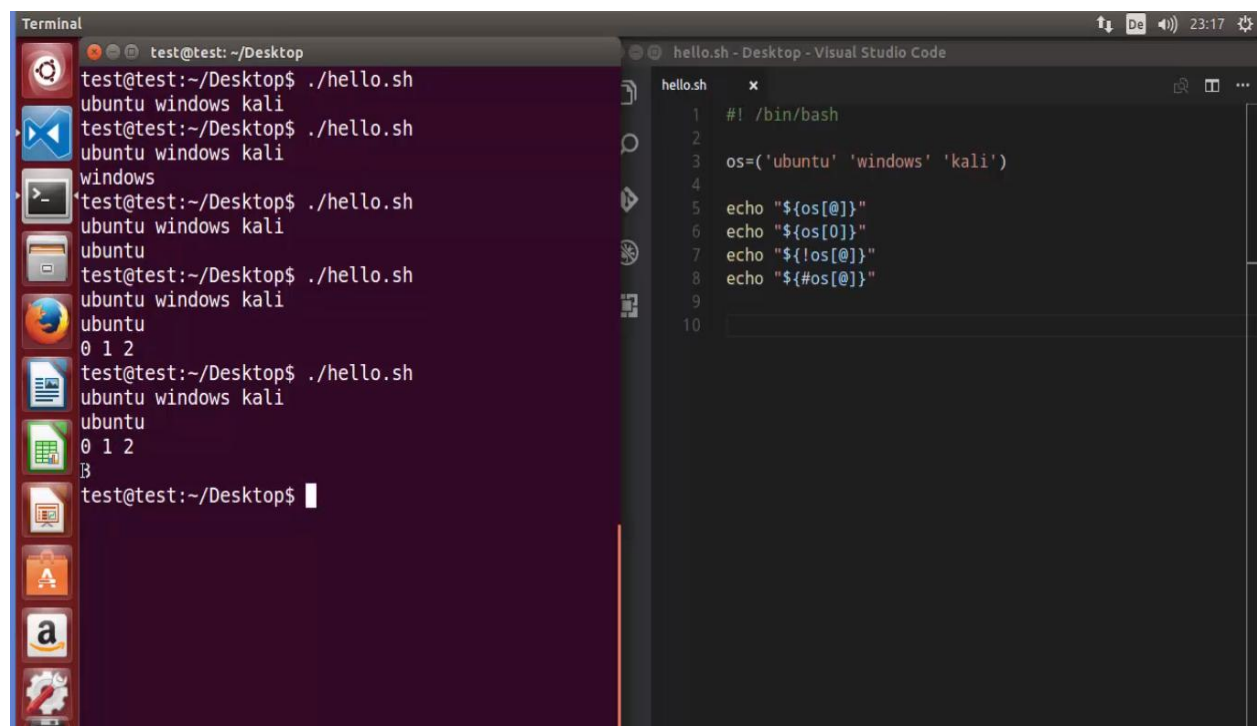


The image shows a terminal window on the left and a script editor on the right. The terminal window shows the execution of a script named 'hello.sh' in the Desktop directory, which outputs 'Unknown vehicle'. The script editor, titled 'hello.sh', shows the complete script code.

```
test@test:~$ cd Desktop/  
test@test:~/Desktop$ ./hello.sh  
Unknown vehicle  
test@test:~/Desktop$  
  
hello.sh  
1  #!/bin/bash  
2  
3  vehicle=$1  
4  
5  case $vehicle in  
6      "car" )  
7          echo "Rent of $vehicle is 100 dollar" ;;  
8      "van" )  
9          echo "Rent of $vehicle is 80 dollar" ;;  
10     "bicycle" )  
11         echo "Rent of $vehicle is 5 dollar" ;;  
12     "truck" )  
13         echo "Rent of $vehicle is 150 dollar" ;;  
14     * )  
15         echo "Unknown vehicle" ;;  
16  esac  
17  
18  
19
```



Array:



Terminal

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

hello.sh - Desktop - Visual Studio Code

```
hello.sh
1  #!/bin/bash
2
3  os=('ubuntu' 'windows' 'kali')
4  os[3]='mac'
5  echo "${os[@]}"
6  echo "${os[0]}"
7  echo "${!os[@]}"
8  echo "${#os[@]}"
9
10
```

Visual Studio Code

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

hello.sh - Desktop - Visual Studio Code

```
hello.sh
1  #!/bin/bash
2
3  os=('ubuntu' 'windows' 'kali')
4  os[6]='mac'
5
6  unset os[2]
7  echo "${os[@]}"
8  echo "${os[0]}"
9  echo "${!os[@]}"
10 echo "${#os[@]}"
11
12
```

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 "Addition 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"
```