

# Basic Of Shell Scripting

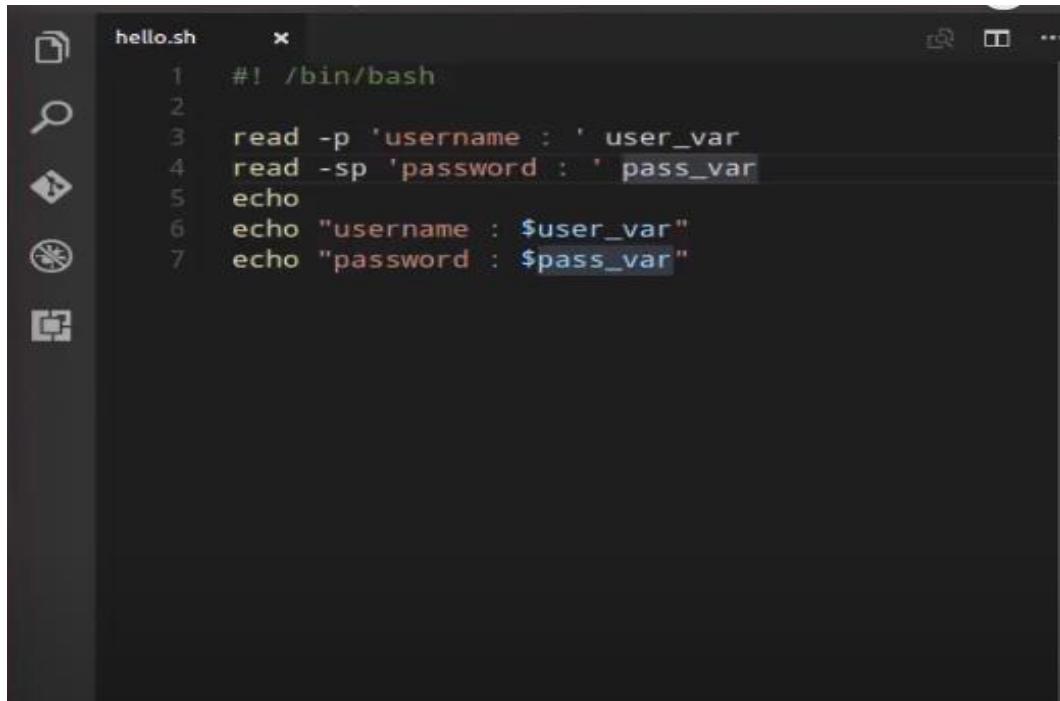
||

# read Command

- Syntax for taking multiple user input using read keyword.

```
#!/bin/bash
echo "Enter three course name"
read name1 name2 name3
echo "names are: $name1, $name2, $name3"
```

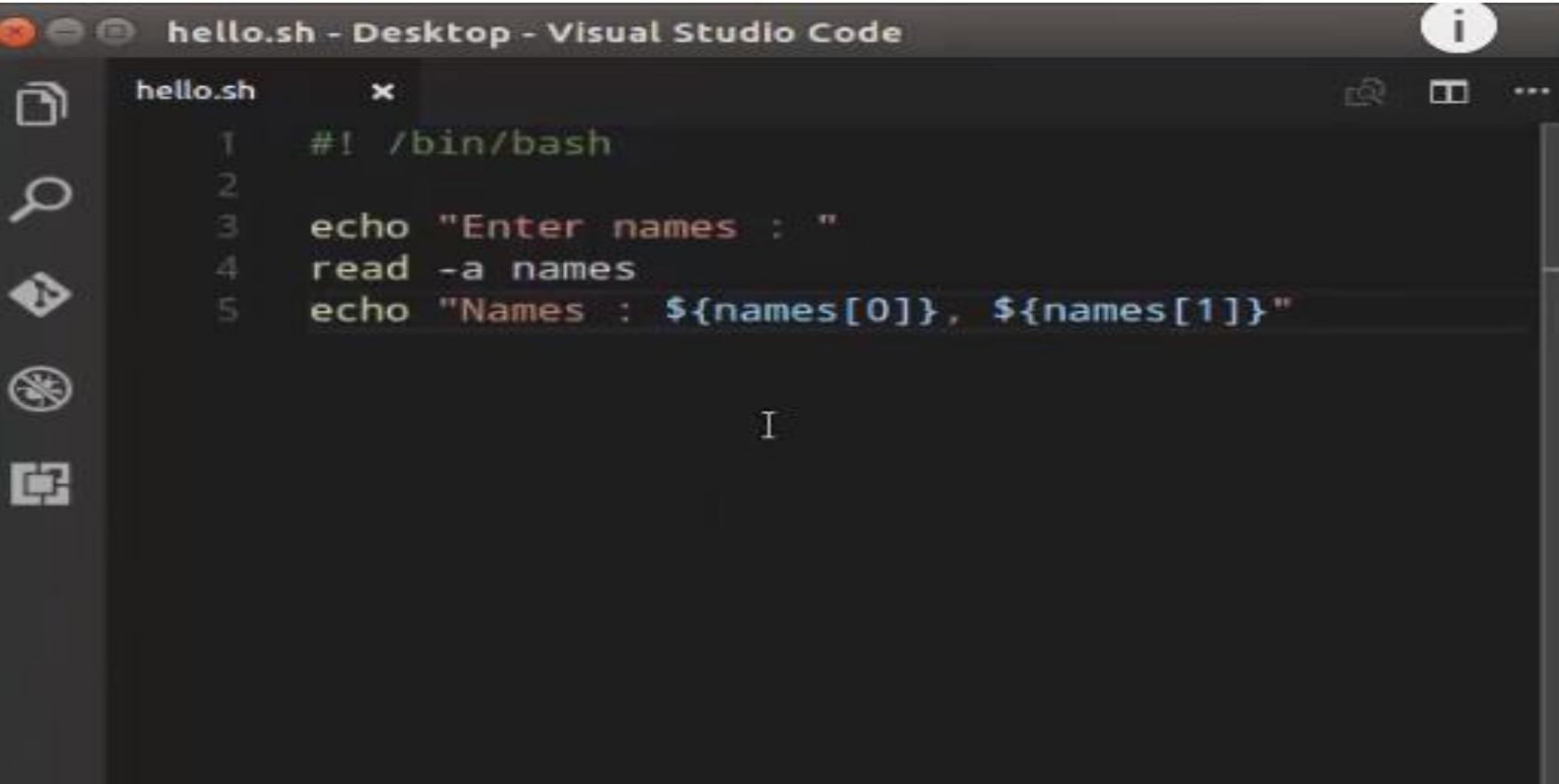
# Example



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

```
test@test:~/Desktop$ ./hello.sh
username : myuser
password :
username : myuser
password : 12345
```

# Example



A screenshot of the Visual Studio Code interface showing a file named "hello.sh". The code editor displays the following script:

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

The code is written in Bash. It prompts the user to enter names, reads them into an array called "names", and then prints the first two names separated by a comma and a space.

# Example

The screenshot shows a Linux desktop environment with a terminal window and a code editor window.

**Terminal Window:**

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

**Code Editor (Visual Studio Code):**

```
hello.sh  x
1  #!/bin/bash
2
3  echo "Enter name : "
4  read
5  echo "Name : $REPLY"
```

# Adding Basic Options

- Let's take a look at some of the most basic options we can use:
- *-a array*: stores the results of the word split operation in an array rather than separate variables
- *-s*: does not echo the input line to the standard output stream
- *-p prompt*: print the prompt text before requesting the input from the standard input stream without a *<newline>* character
- *-t timeout*: attempt to read the input for a given period of seconds
- *-N*: read exactly N characters from the input unless a timeout occurs or *EOF* is reached

# Conditional Statement

## If-else Statement #

- Bash if conditionals can have different forms. The most basic if statement takes the following form:

if TEST-COMMAND

then

STATEMENTS

Else

STATEMENTS

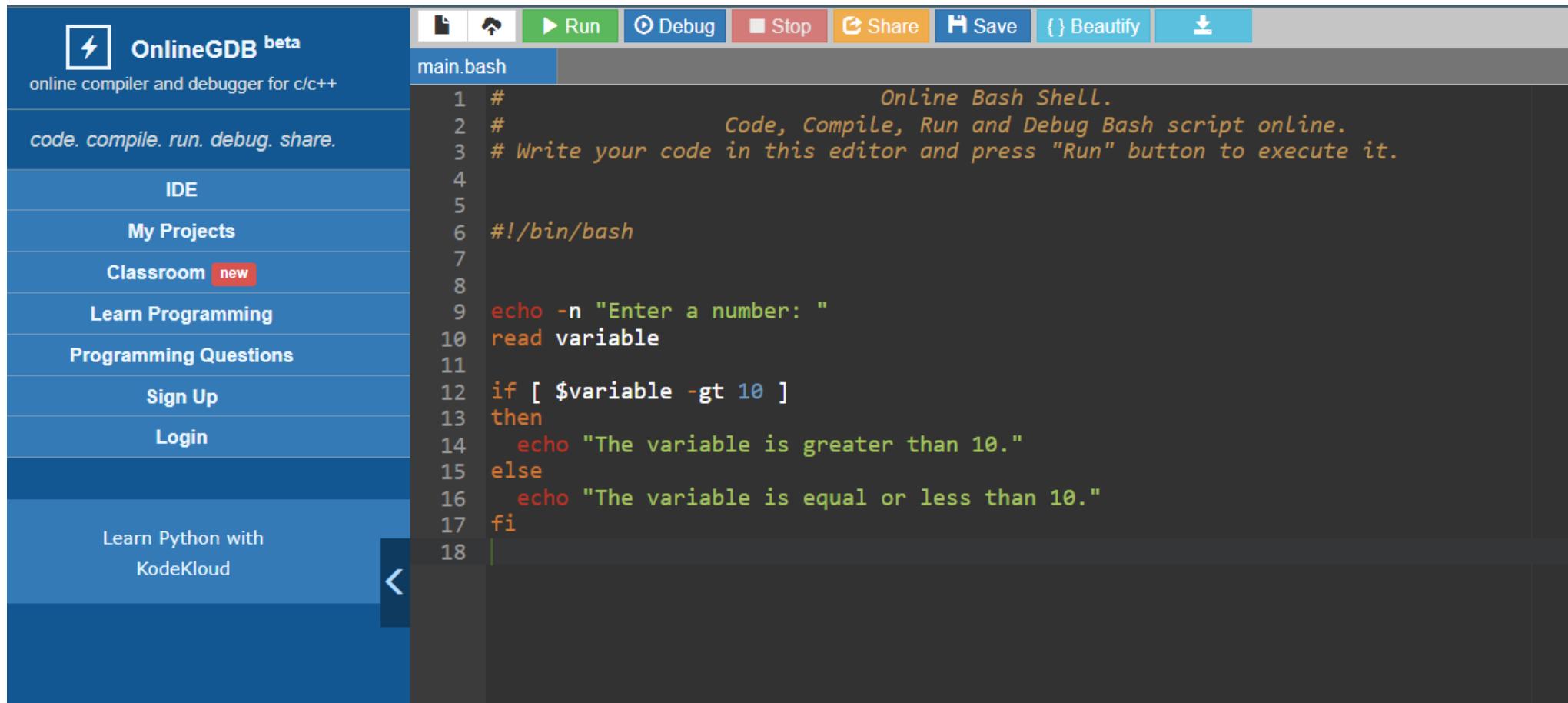
fi

- The if statement starts with the if keyword followed by the conditional expression and the then keyword. The statement ends with the fi keyword.

# Syntax

```
untitled • Telemetry Consent Welcome
1 integer comparison
2
3 -eq - is equal to - if [ "$a" -eq "$b" ]
4 -ne - is not equal to - if [ "$a" -ne "$b" ]
5 -gt - is greater than - if [ "$a" -gt "$b" ]
6 -ge - is greater than or equal to - if [ "$a" -ge "$b" ]
7 -lt - is less than - if [ "$a" -lt "$b" ]
8 -le - is less than or equal to - if [ "$a" -le "$b" ]
9 < - is less than - (( "$a" < "$b" ))
10 <= - is less than or equal to - (( "$a" <= "$b" ))
11 > - is greater than - (( "$a" > "$b" ))
12 >= - is greater than or equal to - (( "$a" >= "$b" ))
13
14 string comparison
15 = - is equal to - if [ "$a" = "$b" ]
16 == - is equal to - if [ "$a" == "$b" ]
17 != - is not equal to - if [ "$a" != "$b" ]
18 < - is less than, in ASCII alphabetical order - if [[ $a < $b ]]
19 > - is greater than, in ASCII alphabetical order - if [[ $a > $b ]]
20 -z - string is null, that is, has zero length
```

# Example Code



The screenshot shows the OnlineGDB beta interface. The left sidebar includes links for IDE, My Projects, Classroom (new), Learn Programming, Programming Questions, Sign Up, and Login. A banner at the bottom left says "Learn Python with KodeKloud". The main area has a toolbar with Run, Debug, Stop, Share, Save, and Beautify buttons. The code editor window is titled "main.bash" and contains the following Bash script:

```
1 #                                     Online Bash Shell.
2 #                                     Code, Compile, Run and Debug Bash script online.
3 # Write your code in this editor and press "Run" button to execute it.
4
5
6 #!/bin/bash
7
8
9 echo -n "Enter a number: "
10 read variable
11
12 if [ $variable -gt 10 ]
13 then
14     echo "The variable is greater than 10."
15 else
16     echo "The variable is equal or less than 10."
17 fi
18 |
```

# if..elif..else Statement

```
if TEST-COMMAND1  
then  
    STATEMENTS1  
elif TEST-COMMAND2  
then  
    STATEMENTS2  
else  
    STATEMENTS3  
fi
```

If the TEST-COMMAND1 evaluates to True, the STATEMENTS1 will be executed. If the TEST-COMMAND2 evaluates to True, the STATEMENTS2 will be executed. If none of the test commands evaluate to True, the STATEMENTS2 is executed.

# Example Code



The screenshot shows the OnlineGDB beta interface. The left sidebar has links for IDE, My Projects, Classroom (new), Learn Programming, Programming Questions, Sign Up, and Login. A banner at the bottom left says "Learn Python with KodeKloud". The main area has tabs for "main.bash" and "Online Bash Shell". The code editor contains the following Bash script:

```
1 #                                         Online Bash Shell.
2 #                                         Code, Compile, Run and Debug Bash script online.
3 # Write your code in this editor and press "Run" button to execute it.
4
5
6#!/bin/bash
7
8
9echo -n "Enter a number: "
10read Variable
11
12if [ $Variable -gt 10 ]
13then
14    echo "The variable is greater than 10."
15elif [ $Variable -eq 10 ]
16then
17    echo "The variable is equal to 10."
18else
19    echo "The variable is less than 10."
20fi
```

# Arithmetic Operation

```
main.bash
3 # Write your code in this editor and press "Run" button to execute it.
4#!/bin/bash
5x=10
6y=20
7echo "x=10, y=5"
8echo "Addition of x and y"
9echo $(( $x + $y ))
10echo "Subtraction of x and y"
11echo $(( $x - $y ))
12echo "Multiplication of x and y"
13echo $(( $x * $y ))
14echo "Division of x by y"
15echo $(( $x / $y ))
16echo "Exponentiation of x,y"
17echo $(( $x ** $y ))
18echo "Modular Division of x,y"
19echo $(( $x % $y ))
20echo "Incrementing x by 10, then x= "
21(( x += 10 ))
22echo $x
23echo "Decrementing x by 15, then x= "
24(( x -= 15 ))
25echo $x
26echo "Multiply of x by 2, then x="
27(( x *= 2 ))
28echo $x
29echo "Dividing x by 5, x= "
30(( x /= 5 ))
31echo $x
32echo "Remainder of Dividing x by 5, x="
33(( x %= 5 ))
34echo $x
35
```

# Arithmetic Operation

```
x=10, y=5                                     input
Addition of x and y
30
Subtraction of x and y
-10
Multiplication of x and y
200
Division of x by y
0
Exponentiation of x,y
7766279631452241920
Modular Division of x,y
10
Incrementing x by 10, then x=
20
Decrementing x by 15, then x=
5
Multiply of x by 2, then x=
10
Dividing x by 5, x=
2
Remainder of Dividing x by 5, x=
2

...Program finished with exit code 0
Press ENTER to exit console.[]
```

# Floating Point Operation

```
#!/bin/bash

# Define floating-point numbers
num1=3.14
num2=2.5

# Use bc with -l option for floating-point arithmetic and math functions
result=$(echo "scale=2; $num1 + $num2" | bc -l)

# Print the result
echo "The sum is: $result"
```

# Explanation

- `scale=2;` sets the number of decimal places to 2 in the result. You can adjust the scale value based on your precision requirements.
- `echo "scale=2; $num1 + $num2" | bc -l` performs the floating-point arithmetic with the specified precision.
- Again, this is just a simple example. You can modify the expression within echo to perform other mathematical operations or use additional functions provided by the bc math library when using the `-l` option.
- This output represents the result of the addition operation  $3.14 + 2.5$  with a scale of 2 decimal places, as specified in the `scale=2` setting. The `bc -l` option allows for floating-point arithmetic and makes the math library available for use in the expression.