

Basic Of Shell Scripting

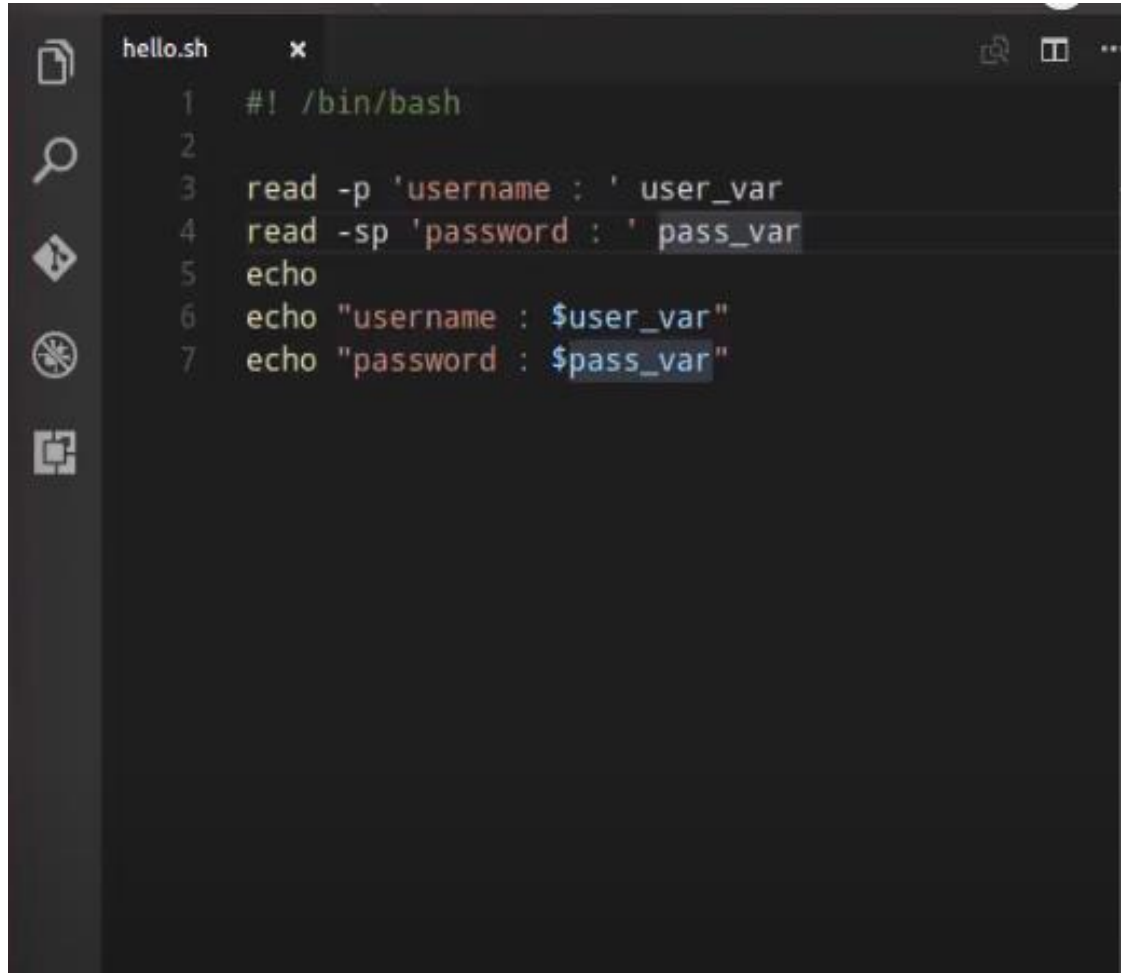
II

read Command

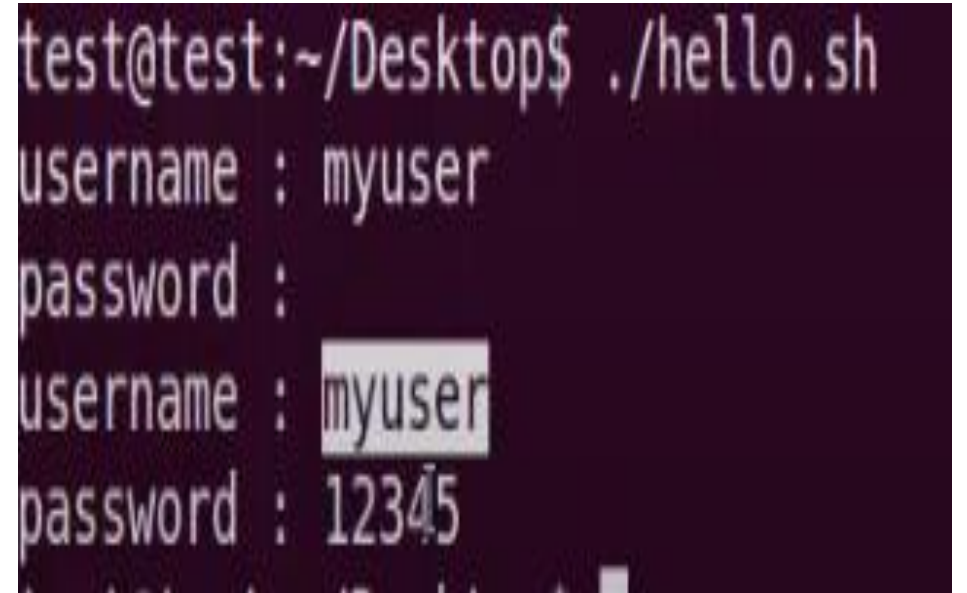
- Syntax for taking multiple user input using read keyword.

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

Example

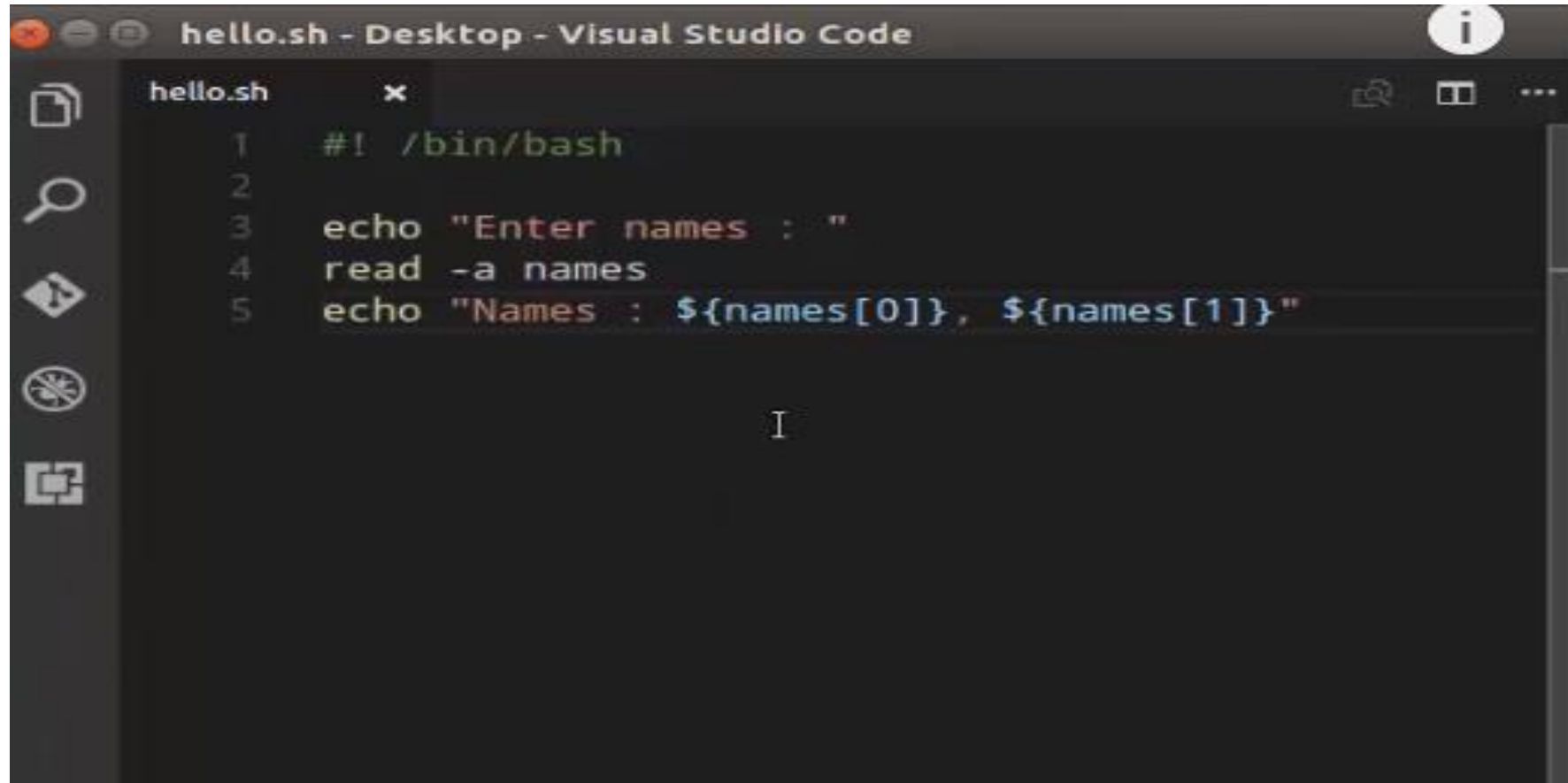
A screenshot of a code editor window with a dark theme. The window title is 'hello.sh'. The code is as follows:

```
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"
```

A screenshot of a terminal window with a dark background. The prompt is 'test@test:~/Desktop\$'. The user has entered './hello.sh'. The script prompts for a username and password. The user has entered 'myuser' for the username and '12345' for the password. The output is as follows:

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

Example

A screenshot of a Visual Studio Code editor window. The title bar at the top reads 'hello.sh - Desktop - Visual Studio Code'. The editor is open to a file named 'hello.sh'. The code in the file is a shell script with five lines: line 1 is '#! /bin/bash', line 2 is an empty line, line 3 is 'echo "Enter names : "', line 4 is 'read -a names', and line 5 is 'echo "Names : \${names[0]}, \${names[1]}"'. The script uses color syntax highlighting: '#' is green, '/' is blue, 'bin' is green, 'bash' is blue, 'echo' is red, 'Enter' is red, 'names' is blue, and 'Names' is red. The variable expansion '\${names[0]}' and '\${names[1]}' are in blue. A cursor is positioned at the end of line 5. The left sidebar shows the Explorer, Search, and Run and Debug views. The right sidebar shows the Output and Debug Console views.

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

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

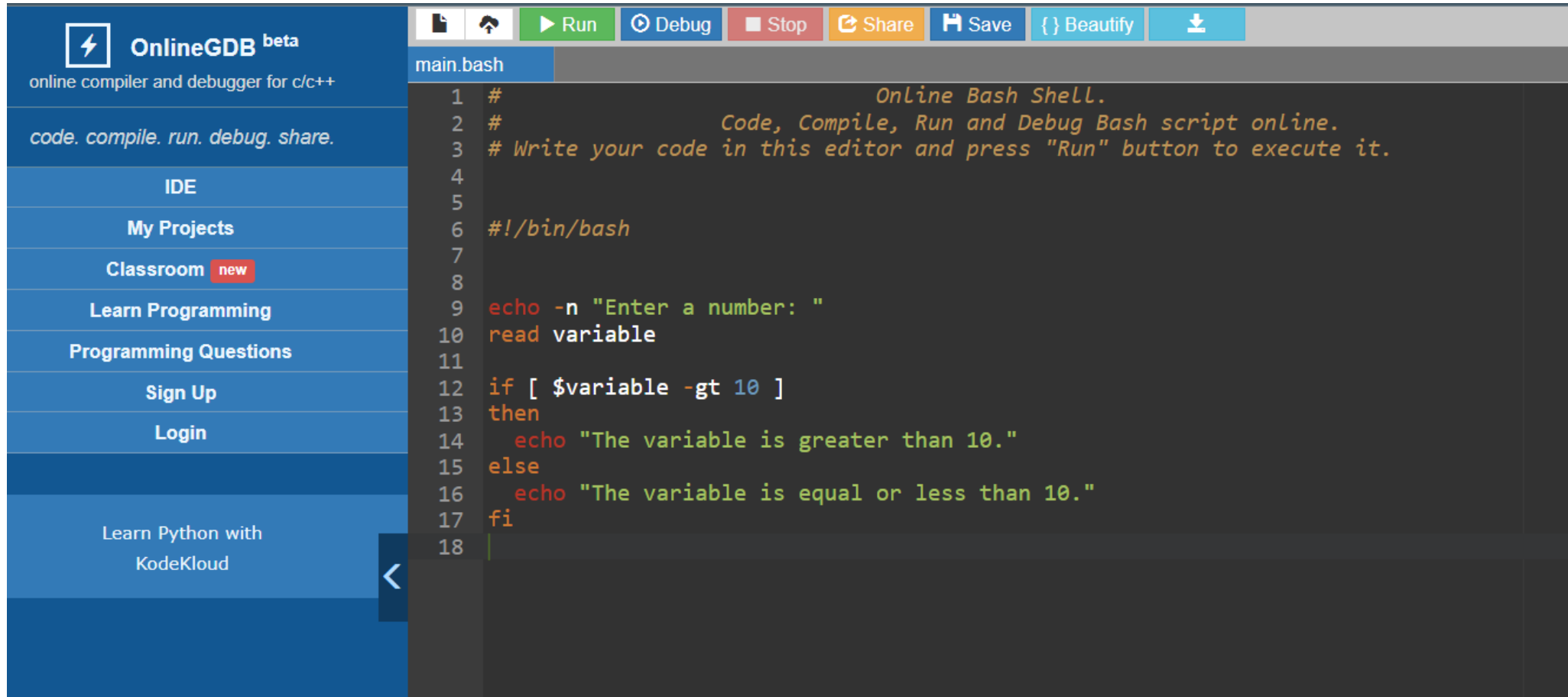
- The if statement starts with the if keyword followed by the conditional expression and the then keyword. The statement ends with the fi keyword.

Syntax

```
untitled — Atom
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 displays the OnlineGDB web interface. On the left is a blue sidebar with navigation links: 'OnlineGDB beta' (with a lightning bolt icon), 'online compiler and debugger for c/c++', 'code. compile. run. debug. share.', 'IDE', 'My Projects', 'Classroom' (with a red 'new' badge), 'Learn Programming', 'Programming Questions', 'Sign Up', 'Login', and 'Learn Python with KodeKloud'. The top of the editor features a toolbar with icons for file operations and buttons for 'Run' (green), 'Debug' (blue), 'Stop' (red), 'Share' (orange), 'Save' (blue), 'Beautify' (cyan), and a download icon. The main editor area shows a file named 'main.bash' containing a Bash script. The script includes comments about the Online Bash Shell and a logic to check if a user-entered number is greater than 10.

```
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 displays the OnlineGDB beta web interface. On the left is a dark blue sidebar with navigation links: 'IDE', 'My Projects', 'Classroom' (with a red 'new' badge), 'Learn Programming', 'Programming Questions', 'Sign Up', 'Login', and 'Learn Python with KodeKloud'. The main area features a toolbar with icons for file operations and buttons for 'Run' (green), 'Debug' (blue), 'Stop' (red), 'Share' (orange), 'Save' (blue), 'Beautify' (light blue), and a download icon. Below the toolbar, a tab labeled 'main.bash' is active, showing a Bash script with line numbers 1 through 20. The script includes comments about the Online Bash Shell and a logic to check if a variable is greater than, equal to, or less than 10.

```
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 elif [ $Variable -eq 10 ]
16 then
17     echo "The variable is equal to 10."
18 else
19     echo "The variable is less than 10."
20 fi
```