BASH Programming

30 bash script Interview questions and answers

3 years ago • by Fahmida Yesmin

Bash scripting is a very useful and powerful programming language that is mainly used to make any manual task automated. A jobseeker who wants to be an automation tester or bash programmer has to face some common questions in the interview. Different types of shell scripts exist in Linux. One of the popular and mostly used shell scripts is Bourne Again Shell or Bash. 30 important interview questions and answers on bash scripting language are described in this article to take preparation for the job.

#01. What is bash script?

The bash script is a shell programming language. Generally, we run many types of shell commands from the terminal by typing each command separately that require time and efforts. If we need to run the same commands again then we have to execute all the commands from the terminal again. But using a bash script, we can store many shell command statements in a single bash file and execute the file any time by a single command. Many system administration related tasks, program installation, disk backup, evaluating logs, etc. can be done by using proper bash script.

#02. What are the advantages of using bash scripts?

Bash script has many advantages which are described below:

04:32

- It is easy to use and learn.
- Many manual tasks that need to run frequently can be done automatically by writing a bash script.
- The sequence of multiple shell commands can be executed by a single command.
- Bash script written in one Linux operating system can easily execute in other Linux operating system. So, it is portable.
- Debugging in bash is easier than other programming languages.
- Command-line syntax and commands that are used in the terminal are similar to the commands and syntax used in bash script.
- Bash script can be used to link with other script files.

#03. Mention the disadvantages of bash scripts

Some disadvantages of bash script are mentioned below:

- It works slower than other languages.
- The improper script can damage the entire process and generate a complicated error.
- It is not suitable for developing a large and complex application.
- It contains less data structure compare to other standard programming languages.

04:32

d in bash script. These are:

ed and maintained by the Linux operating system are called
ariables are always used by an uppercase letter. The default
hanged based on requirements.

ands can be used to print the list of system variables.

Example:

```
#!/bin/bash
# Printing System Variables

#Print Bash shell name
echo $BASH

# Print Bash shell Version
echo $BASH_VERSION

# Print Home directory name
echo $HOME
```

User-defined variable

The variables which are created and maintained by users are called user-defined variables. They are also called local variables. These types of variables can be declared by using lowercase or uppercase or both uppercase and lowercase letters. But it is better to avoid using all uppercase letter to differentiate the variables from system variables.

Example:

```
#!/bin/bash

num=100
echo $num
```

#05. How to declare and delete variables in bash?

04:32

command to define the data type of the variable at the time

–r, **-i, -a, -A, -l, -u, -t** and **–x** options can be used with **declare** command to declare a variable with different data types.

Example:

```
#!/bin/bash

#Declare variable without any type
num=10

#Values will be combined but not added
result=$num+20
echo $result

#Declare variable with integer type
declare -i num=10

#Values will be added
declare -i result=num+20
echo $result
```

unset command is used to remove any bash variable. The variable will be inaccessible or undefined after using **unset** command.

Example:

```
#!/bin/bash

str="Linux Hint"
echo $str
unset $str
echo $str
```

#06. How to add comments in a bash script?

Single line and multi-line comments can be used in bash script. '**#**' symbol is used for single-line comment. **'<<'** symbol with a delimiter and **':'** with single (') are used for adding multi-line comment.

Example:

```
#!/bin/bash
#Print the text [Single line comment]
echo "Bash Programming"
<<addcomment
Calculate the sum
Of two numbers [multiline comment]
addcomment
num=25+35
echo $num
: '
Combine two
String data [multiline comment]
'
String="Hello"
echo $string" World"
```

#07. How can you combine strings in a bash script?

String values can be combined in bash in different ways. Normally, the string values are combined by placing together but there are other ways in bash to combine string data.

Example:

```
#!/bin/bash
#Initialize the variables
str1="PHP"
str2="Bash"
str3="Perl"

# Print string together with space
echo $str1 $str2 $str3

#Combine all variables and store in another variable
str="$str1, $str2 and $str3"

#Combine other string data with the existing value of the string
str+=" are scripting languages"

#Print the string
echo $str
```

04:32

ɐ used to print output in bash?

n be used to print output in bash. `**echo**` command is used
ntf` command is used to print the formatted output.

```
#!/bin/bash

#Print the text
echo "Welcome to LinuxHint"
site="linuxhint.com"
#Print the formatted text
printf "%s is a popular blog site\n" $site
```

#09. How to take input from the terminal in bash?

`**read**` command is used in a bash script to take input from the terminal.

Example:

```
#!/bin/bash
#Print message
echo "Enter your name"
#Take input from the user
read name
# Print the value of $name with other string
echo "Your name is $name"
```

#10. How to use command-line arguments in bash?

Command-line arguments are read by **$1, $2, $3…$n** variables. Command-line argument values are provided in the terminal when executing the bash script. **$1** is used to read the firs argument, **$2** is used to read the second argument and so on.

Example:

04:32

```
fi

                                    string
                                    \n" $color
```

#11. Is bash a weakly typed language? Why?

Yes, bash is considered a weakly or loosely typed language because it does not require to declare the type of the data at the time of variable declaration. All bash variables are treated as a string by default and the type of the variable will be set based on the current value. Bash variables with data types can be defined by using **declare** command with a particular option. But the options to define data types are limited and don't support all types of data. For example, **float** data type can't be declared by using **declare** command.

Example:

```
#!/bin/bash

#The data type of $myVar is string by default
myVar=29

# Print the variable
echo $myVar

# Declare integer variable $number with the value 67
declare -i number=67

#Print the variable
echo $number

# Assign string data into the numeric variable. The following line will generate
# syntax error and the value of $number will not change
number="I like bash"
echo $number
```

#12. How to read the second word or column from each line of a file?

The second word or column of a file can be read in a bash script by using different bash commands easily, such as `awk`, `sed` etc. Here, the use of `**awk**` is shown in the following example.
Example: Suppose, course.txt file contains the following content and we have printed only the second word of each line of this file.

04:32

```
# of each line.
                                     ıt $2}'`
```

#13. How to declare and access an array variable in bash?

Both numeric and associative arrays are supported by a bash script. An array variable can be declared with and without declare command. **–a** option is used with declare command to define a numeric array and **–A** option is used with declare statement to define an associative array in bash. Without declare command, the numeric array can be defined only in bash.

Example:

```
#!/bin/bash

# Declare a simple numeric array
arr1=( CodeIgniter Laravel ReactJS )

# Print the first element value of $arr1
echo ${arr1[0]}

# Declare a numeric array using declare command
declare -a arr2=( HTML CSS JavaScript )

# Print the second element value of $arr2
echo ${arr2[1]}

# Declare an associative array using declare statement
declare -A arr3=( [framework]=Laravel [CMS]=Wordpress [Library]=JQuery )

# Print the third element value of $arr3
echo ${arr3[Library]}
```

All elements of an array can be accessed by using any loop or '*' symbol as an array index.

#14. How can conditional statements be used in bash?

The most common conditional statement in most programming languages is **the if-elseif-else** statement. The syntax of **if-elseif-else** statement in bash is a little bit different from other programming languages. **'If'** statement can be declared in two ways in a bash script and every type of **'if'** block must be closed with **'fi'**. **'if'** statement can be defined by third bracket or first brackets like other programming languages.

Syntax:

04:32

```
statements
```

```
if [ condition ]; then
statements 1
else
statement 2
fi
```

C.

```
if [ condition ]; then
statement 1
elif [ condition ]; then
statement 2
….
else
statement n
fi
```

Example:

```
#!/bin/bash

# Assign a value to $n
n=30
# Check $n is greater than 100 or not
if [ $n -gt 100 ]; then
    echo "$n is less than 100"
# Check $n id greater than 50 or not
elif [ $n -gt 50 ]; then
    echo "$n is less than 50"
else
    echo "$n is less than 50"
fi
```

		Description
		It is used to check equality
		It is used to check inequality
<	-lt	It is used check the first value is less than the second value or not
>	-gt	It is used check the first value is greater than the second value or not
<=	-le	It is used check the first value is less than or equal to the second value or not
>=	-ge	It is used check the first value is greater than or equal to the second value or not

Example:

```
#!/bin/bash
# Initialize $n
n=130
o="even"
# Check $n is greater than or equal to 100 or not using ‘–ge’.
if [ $n -ge 100 ]; then
     echo "$n is greater than or equal to 100"
else
     echo "$n is less than 100"
fi
# Check $n is even or odd using ‘==’ operator
if (( $o == "even" )); then
     echo "The number is even"
else
     echo "The number is odd"
fi
```

04:32

#16. Which conditional statement can be used as an alternative to if-
ash?

an alternative tp **if-elseif-if** statement. The syntax for **'case'** nt from the **switch-case** statement of other programming by **'esac'** statement in bash. No '**break**' statement is used m the block.

```
case  in
Match pattern 1) commands;;
Match pattern 2) commands;;
……
Match pattern n) commands;;
esac
```

Example:

```
#!/bin/bash
#Initialize the variable $ticket
ticket=101
# Compare the value of $ticket with 23, 101 and 503
case $ticket in
23)
# Print message if the value is 23
echo "You got the first prize";;
101)
# Print message if the value is 101
echo  "You got the second prize";;
503)
# Print message if the value is 503
echo  "You got the third prize";;
*)
# Print message if the value does not match with 23, 101 and 503
echo "Sorry, try for the next time"
exit 0;;
esac
```

#17. What different types of loops can be used in bash?

Three types of loops are supported by a bash script. These are **while, for** and **until** loops. Loops in bash check the condition at the start of the loop. **While** loop works until the conditio remains true and **until** loop works until the condition remains false. There are two ways to us **for** loop. One is general **for** loop that contains three parts and another is **for-in** loop. The uses of these three loops are shown in the following example.

04:32

```
n=5
                                 ng while loop

                                ."

                                 ng for loop

                                ."

# Calculate the square of 5-1 using until loop
until [ $x -le 0 ]
do
   sqr=$((x*x))
   echo "The square of $x is $sqr"
   ((x--))
done
```

#18. How can subroutines be declared and called in bash?

In bash a function or procedure is called a subroutine. The declaration and calling of a subroutine in bash is different from other languages. No argument can be declared in subroutines unlike other standard programming languages. But local variables can be defined within the subroutine by using the **'local'** keyword.

Example:

```
#!/bin/bash
# Initialize the variable $x and $y which are global
x=10
y=35

# Declare the function
myFunc () {
   # Declare the local variable $x
   local x=15

   # Re-assign the global variable $y
   y=25

   # Calculate the sum of $x and $y
   z=$((x+y))

   # Print the sum of a local variable, $x, and global variable, $y
   echo "The sum of $x and $y equal to $z"
}

# Call the function
myFunc

# Print the sum of global variables, $x, and $y
echo "The sum of $x and $y equal to $z"
```

04:32

ome part of a string data in bash?

ther languages to cut some portion of the string data. But ring value can be cut in bash. Three parts can be defined in g with a colon to cut any part of the string data. Here, the the last part is optional. The first part contains the main string variable that will be used to cut, the second part is the starting position from where the string will be cut and the third part is the length of the cutting string. The starting position must be counted from 0 and the length must be counted from 1 of the main string to retrieve the cutting value.

Example:

```
#!/bin/bash
# Initialize a string value into $string
string="Python Scripting Language"
# Cut the string value from the position 7 to the end of the string
echo ${string:7}
# Cut the string value of 9 characters from the position 7
echo ${string:7:9}
# Cut the string value from 17 to 20
echo ${string:17:-4}
```

#20. Mention some ways to perform arithmetic operations in bash?

Arithmetic operations can be done in multiple ways in bash. **'let', 'expr', 'bc'** and **double brackets** are the most common ways to perform arithmetic operations in bash. The uses of these commands are shown in the following example.

Example:

```
#!/bin/bash
# Calculating the subtraction by using expr and parameter expansion
var1=$( expr 120 - 100 )
# print the result
echo $var1
```

04:32

```
echo "scale=2; 44/7" | bc
```

cation using double brackets

#21. How to check a directory exists or not using bash?

Bash has many test commands to check if a file or directory exists or not and the type of the file. **'-d'** option is used with a directory path as a conditional statement to check if the directory exists or not in bash. If the directory exists, then it will return true otherwise it will return false.

Example:

```
#!/bin/bash
# Assign the directory with path in the variable, $path
path="/home/ubuntu/temp"
# Check the directory exists or not
if [ -d "$path" ]; then
   # Print message if the directory exists
   echo "Directory exists"
else
   # Print message if the directory doesn't exist
   echo "Directory not exists"
fi
```

#22. How can a bash script be terminated without executing all statements?

Using **'exit'** command, a bash script can be terminated without executing all statements. The following script will check if a particular file exists or not. If the file exists, then it will print the total characters of the file and if the file does not exist then it will terminate the script by showing a message.

Example:

```
#!/bin/bash

# Initialize the filename to the variable, $filename
filename="course.txt"

# Check the file exists or not by using -f option
if [ -f "$filename" ]; then
    # Print message if the file exists
    echo "$filename exists"
```

04:32

```
the file exists
```

reak and continue statements in bash?

break statement is used to terminate from a loop without completing the full iteration based on a condition and **continue** statement is used in a loop to omit some statements based on a condition. The uses of **break** and **continue** statements are explained in the following example.

Example:

```
#!/bin/bash
# Initialize the variable $i to 0 to start the loop
i=0
# the loop will iterate fot 10 times
while [ $i -le 10 ]
do
    # Increment the value $i by 1
    (( i++ ))
    # If the value of $i equal to 8 then terminate the loop by using 'break' statement
    if [ $i -eq 8 ]; then
         break;
    fi
    # If the value of $i is greater than 6 then omit the last statement of the loop
    #  by using continue statement
    if [ $i -ge 6 ]; then
         continue;
    fi
    echo "the current value of i = $i"
done

# Print the value of $i after terminating from the loop
echo "Now the value of i = $i"
```

#24. How to make a bash file executable?

Executable bash files can be made by using **'chmod'** command. Executable permission can be set by using **'+x'** in **chmod** command with the script filename. Bash files can be executed without the explicit **'bash'** command after setting the execution bit for that file.

Example:

```
# Set the execution bit
$ chmod +x filename.sh

# Run the executable file
$ ./filename.sh
```

#25. Mention some options that are used to test files

Many options are available in bash to test file. Some options are mentioned below.

Option	Description
-f	It is used to test the file exists and it is a regular file.

04:32

	›st the file exists and it has to write permission.
	›st the file exists and it has execution permission.
	›st the directory exists.
	›st the file exists and It is a symbolic link.
	›st the file exists and It is a socket.
-b	It is used to test the file is a block device.
-s	It is used to check the file is not zero sizes.
-nt	It used to check the content of the first file is newer than the second file. For example, file1 -nt file2 indicates that file1 is newer than file2.
-ot	It used to check the content of the first file is older than the second file. For example, file1 -ot file2 indicates that file1 is older than file2.
-ef	It is used to check that two hard links refer to the same file. For example, flink1 -ef flink2 indicates that flink1 and flink2 are hard links and both refer to the same file.

#26. What is meant by ‘bc’ and how can this command can be used in bash?

The full form of ‘bc’ is **Bash Calculator** to perform arithmetic operations more accurately in bash. The fractional part is omitted if any arithmetic operation is done in bash by using **‘expr’** command. The fractional part can be rounded also by using **scale** value with **‘bc’** command.

Example:

```
#!/bin/bash
# Calculate the division without the fractional value
echo "39/7" | bc

# Calculate the division with the full fractional value
echo "39/7" | bc -l

# Calculate the division with three digits after the decimal point
echo "scale=3; 39/7" | bc
```

#27. How can you print a particular line of a file in bash?

There are many ways to print a particular line in bash. How the **‘awk’, ‘sed’** and **‘tail’** commands can be used to print a particular line of a file in bash is shown in the following example.

Example:

```
#!/bin/bash

# Read and store the first line from the file by using `awk` command with NR variable
line1=`awk '{if(NR==1) print $0}' course.txt`
# Print the line
echo $line1

# Read the second line from the file by using `sed` command with -n option
line2=`sed -n 2p course.txt`
```

04:32

```
# Print the file
```

full form of **IFS** is Internal Field Separator,
word from the line of text. It is mainly used for splitting a
ing text etc.

Example:

```
#!/bin/bash
# Declare ':' as delimiter for splitting the text
IFS=":"
# Assign text data with ':' to $text
text="Red:Green:Blue"
# for loop will read each word after splitting the text based on IFS
for val in $text; do
    # Print the word
    echo $val
done
```

#29. How to find out the length of a string data?

'expr', **'wc'** and **'awk'** commands can be used to find out the length of a string data in bash. **'expr'** and **'awk'** commands use **length** option, **'wc'** command uses **'–c'** option to count the length of the string.

Example:

The uses of the above commands are shown in the following script.

```
#!/bin/bash
# Count length using `expr` length option
echo `expr length "I like PHP"`
# Count length using `wc` command
echo "I like Bash" | wc -c
# Count length using `awk` command
echo "I like Python" | awk '{print length}'
```

#30. How to run multiple bash script in parallel?

Multiple bash scripts can be executed in parallel by using **nohup** command. How multiple bash files can be executed in parallel from a folder is shown in the following example.

Example:

```
# Assign a folder name with the path in the variable $dir that contains
# multiple bash files
dir="home/Ubuntu/temp"

# for loop will read each file from the directory and execute in parallel
for script in dir/*.sh
do
    nohup bash "$script" &
done
```

Conclusion:

Most basic bash script interview questions are described in this article for the readers who want to start a career as bash programmer.

04:32

ABOUT THE AUTHOR

Fahmida Yesmin

I am a trainer of web programming courses. I like to write article or tutorial on various IT topics. I have a YouTube channel where many types of tutorials based on Ubuntu, Windows, Word, Excel, WordPress, Magento, Laravel etc. are published: Tutorials4u Help.

View all posts

RELATED LINUX HINT POSTS

- Bash Pad String with Spaces
- How to Use cURL Command in Linux
- Can You Run Bash On Windows 11, And How?
- Resolve Issue: Bash Bad Substitution
- Resolve Issue: Bash Unary Operator Expected
- Bash Check If String Is Empty
- Resolve Issue: Bin/Bash^M: Bad Interpreter: No Such File Or Directory

04:32

04:32