Shell Scripting introduction

1. Shell Scripting Introduction

How to execute the script?

1. use ./script-name.sh
2. else can do sh script-name.sh for Bourne shell
3. else bash script-name.sh for Bourne again shell execution.

This does not matter which shell you use sh or bash as for normal scripts it will have same output, this is needed when you want specific shell to run your script.

Variables-

Variable is nothing but a character string to which we assign a value, or else we can say it's a pointer to actual data.

The value can be a number, string, filename, device or any other type of data.

Types of variables

1. Local Variable
2. Environmental variable
3. Shell Variable

Local variables –

A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.

Ex.

Func () {

NAME=Variable name *# here the name variable is just limited to this particular function block but not in the program*

}

Environmental variables –

An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.

Ex.

1. $PATH
2. $HOME

We can set env variables using export command.

Shell Variables –

A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Ex.

A shell variable is created with the following syntax:

"variable_name=variable_value"

The following table shows a number of special variables that you can use in your shell scripts –

Sr.No.	Variable & Description
1	**$0** The filename of the current script.
2	**$n** These variables correspond to the arguments with which a script was invoked. Here **n** is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
3	**$#** The number of arguments supplied to a script.
4	**$*** All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.

5	**$@** All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
6	**$?** The exit status of the last command executed.
7	**$$** The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
8	**$!** The process number of the last background command.

Exit Status($?) variable in Linux –

"$?" is a variable that holds the return value of the last executed command. **"echo $?"** displays **0** if the last command has been successfully executed and displays a **non-zero** value if some error has occurred.

```
root@ip-172-31-42-137:~# ls
exm.txt  information.txt  msg.txt  scripts  snap  ygminds.txt
root@ip-172-31-42-137:~# cat msg.txt
this is jtp
welcome to jtp
learn linux
linux is very easy
its interesting
root@ip-172-31-42-137:~# echo $?
0
root@ip-172-31-42-137:~#
```

```
root@ip-172-31-42-137:~# eecho hello
Command 'eecho' not found, did you mean:
  command 'echo' from deb coreutils (8.32-4.1ubuntu1)
Try: apt install <deb name>
root@ip-172-31-42-137:~# echo $?
127
root@ip-172-31-42-137:~#
```

Basic operators in shell-

- **Arithmetic Operators**

Assume variable **a** holds **10** and variable **b** holds **20** then

Operator	**Description**	**Example**
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
– (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a – $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a

== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

==========

Script-

```
#!/bin/sh

val=`expr 2 + 2`
echo "Total value : $val"
```

- **Relational Operators-**

Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

For example, following operators will work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable **a** holds **10** and variable **b** holds **20** then

Operator	**Description**	**Example**
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.

-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

- **Boolean Operators –**

The following Boolean operators are supported by the Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

- **String Operators –**
  - The following string operators are supported by Bourne Shell.
  - Assume variable a holds **"abc"** and variable b holds **"efg"** then −

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.

!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
Str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

- **File Test Operators**

We have a few operators that can be used to test various properties associated with a Unix file.

Assume a variable **file** holds an existing file name "test" the size of which is 100 bytes and has **read**, **write** and **execute** permission on –

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.

-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set; if yes, then the condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe; if yes, then the condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.	[-t $file] is false.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.

-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.
-e file	Checks if file exists; is true even if file is a directory but exists.	[-e $file] is true.

Types of shell-

The Bourne Shell (sh) :-

It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.

The C shell (csh):-

It includes helpful programming features like built-in arithmetic and C-like expression syntax.

The Korn Shell (ksh):-

Korn Shell or KSH was developed by a person named David Korn, which attempts to integrate the features of other shells like C shell. Is a superset of the Bourne shell. So it supports everything in the Bourne shell. It has

interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities. It is faster than C shell. It is compatible with script written for C shell.

Bourne Again Shell (bash) :-

Bash shell was developed by the Freeware Software and it is written and licensed under the GNU organization. It is compatible to the Bourne shell. It includes features from Korn and Bourne shell. Bash shell is free and open-source to use for computer users.