

**Shell Script Order**

1. **Shebang** ***#!/PATH\_to\_INTERPRETER***  //all scripts should start with shebang

*#!/usr/bin/env bash* //you can make the bashscript portable with any unix system

(which env)

2. **Comments** // to help others understand the summary of the script

3. **Global variables** //these are the variables that you can use in entire script

4. **Functions** // write all you functions as soon as you declare the global variables,

//remember that shell executes lines of the script in order

//& hence you cannot properly make use of a function

//that is written later or at the end of the script

a.**use local variables** // these are local to the function

5. **Main Script**  // main logic of your script

6. **Exit with an exit status** //(0-255)without this the exit status of the previously executed

//cmd will be used as an exit status for the current cmd

a. **exit <STATUS> at various exit points**

Example Shell Script Template / Boilerplate:

#!/bin/bash

#

# <Replace with the description and/or purpose of this shell script.>

GLOBAL\_VAR1="one"

GLOBAL\_VAR2="two"

function function\_one() {

local LOCAL\_VAR1="one"

# <Replace with function code.>

}

# Main body of the shell script starts here.

#

# <Replace with the main commands of your shell script.>

# Exit with an explicit exit status.

exit 0

**//shebang or sharp bang**

**read**  *to accept input*

**;** *Semicolon can be used to chain multiple commands in a single line*

**##Variables**

Syntax :

VARIABLE\_NAME="value" ***//assigning value to a variable***

VARIABLE\_NAME=$(COMMAND) ***//assigning output of a command to variable***

${VARIABLE\_NAME} ***//accessing a variable***

Example:

#!/bin/bash

SERVER\_NAME=$(uname -n)

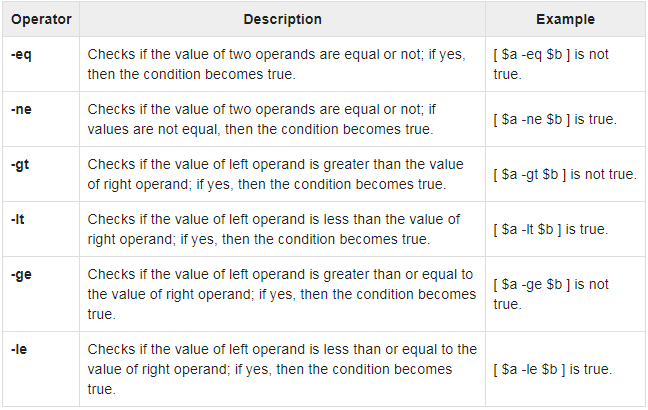
echo "servername is ${SERVER\_NAME}"

**## Relational operators**

It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them.

For example, [ $a <= $b ] is correct whereas, [$a <= $b] is incorrect.

-eq, -ne, -gt, -lt, -ge, -le



**## File test operators**

-e FILE True if file exists

-f FILE True if file exists and is a regular file

-d FILE True if file is a directory

-s FILE True if file exists and is not empty.

-r FILE True if file is readable by you.

-w FILE True if file is writable by you.

-x FILE True if file is executable by you.

**Syntax:**

**[ condition-to-test-for ]**

**[ -e /etc/passwd ]**

**## String test Operators**

-zSTRING True if string is empty

-nSTRING True if string is not empty

STRING1=STRING2 True is strings are equal

STRING1!=STRING2 True is strings are not equal

**##Logical Operators :**

They are also known as boolean operators. These are used to perform logical operations.

**Logical AND (&&)** : This is a binary operator, which returns true if both the operands are true otherwise returns false.

**Logical OR (||) :** This is a binary operator, which returns true is either of the operand is true or both the operands are true and returns false if none of then is false.

**Not Equal to (!) :** This is a uninary operator which returns true if the operand is false and returns false if the operand is true.

**## Arthmetic test Operators**

**## if/elif/else Statements**

if [ condition-is-true]

then

command 1

elif [ condition2-is-true]

then

command 2

else

command N

fi

**## for loop**

for VARIABLE\_NAME in ITEM\_1 ITEM\_2 ITEM\_n

do

command1

command2

command N

done

**## Positional parameters //to access items on a commandline**

$ script.sh parameter1 parameter2 parameter3

**$0** *The filename of the current script.*

**$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).*

**$#** *The number of arguments supplied to a script.*

**$\*** *All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2.*

**$@** *All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.*

**$?** *The exit status of the last command executed.*

**$$** *The process number of the current shell. For shell scripts, this is the process ID under which they are executing.*

**$!** *The process number of the last background command.*

**## return codes and Exit status**

Each shell command returns an exit code when it terminates, either successfully or unsuccessfully.  
By convention, an exit code of zero indicates that the command completed successfully, and non-zero means that an error was encountered.

The special variable **$?** returns the exit status of the last executed command

*vagrant@yaswanth-hashi:~$* ***echo $?***

*0*

The **exit** command exits the shell with a status of **N**. It has the following syntax:

**Exit N**

If N is not given, the exit status code is that of the last executed command

**##Functions**

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed.

***function\_name () {***

***list of commands***

***}***

The name of your function is **function\_name**, and that's what you will use to call it from elsewhere in your scripts. The function name must be followed by parentheses, followed by a list of commands enclosed within braces.

Executing **exit** inside a function will terminate the shell program that called the function. In this case we can use the **return** to return any value from the function.

***return code***

**##Wildcards**

* **\*** **//matches zero or more characters**
* **?** **//matches exactly one character**
* **[] //A character class matches any of the characters included between the**

**// brackets. Matches exactly one character.**

* **[aeiou]**
* **ca[nt]\***
* **can**
* **cat**
* **candy**
* **catch**
* **[!] //A character class matches any of the characters not included between the brackets.**

**//Matches exactly one character.**

* **Range //Use two characters separated by a hyphen to create a range in a character class**
* **[a-e]\* //matches all files that start with a,b,c,d or e**
* **[2-5]\* //matches all files that start with 2, 3, 4 or 5**
* **\ //escape character, used to match a wild character, ie if the name contains’?’**
* **Named character classes**
* **[[:alpha:]]**
* **[[:alnum:]]**
* **[[:digit:]]**
* **[[:lower:]]**
* **[[:space:]]**
* **[[:upper:]]**

**##Case statements**

The case statement is used to execute statements based on specific values. Often used in place of an **if** statement, if there are a large number of conditions. The variable is compared with the values using the shell wildcards ( ? \* […] ) , NOT regular expressions. All the statements are executed for the first matching value until the ending ;;. If no value matches, then the default \*) case is executed, if present.

**Syntax:**

The basic syntax of the **case...esac** statement is to give an expression to evaluate and to execute several different statements based on the value of the expression.

The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

**case <word> in**

**pattern1)**

**Statement(s) to be executed if pattern1 matches**

**;;**

**pattern2)**

**Statement(s) to be executed if pattern2 matches**

**;;**

**pattern3)**

**Statement(s) to be executed if pattern3 matches**

**;;**

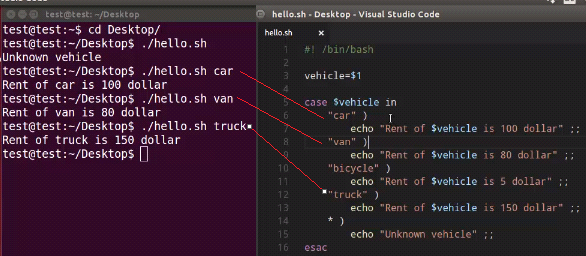
**\*)**

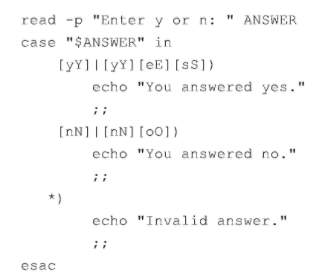
**Default condition to be executed**

**;;**

**esac**

**EXAMPLE:**





**##Logging**



**Logger** makes entries in the system log. It provides a shell command interface to the **syslog**(3) system log module.

**##while loop**

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

**while [condition-is-true]**

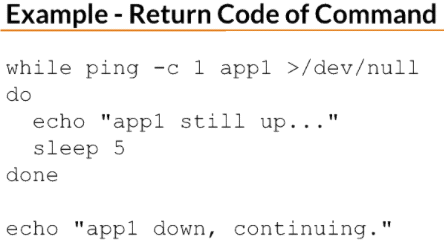
**do**

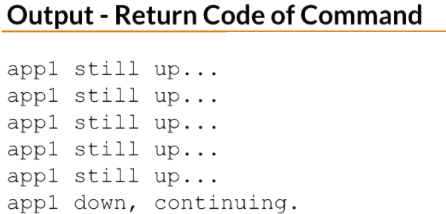
**command1**

**command2**

**commandN**

**done**





**## Break statement**

**##continue statement**

**##Debugging shell script**

**########################################################################**

* echo
* read
* redirection >, >>, < . |
* grep
* cut
* awk
* sed
* tr
* tee
* heredoc, herestring
* debugging
* case
* if
* for
* exit statuses
* break, continue
* Xargs
* source
* !$
* $?
* /dev/null