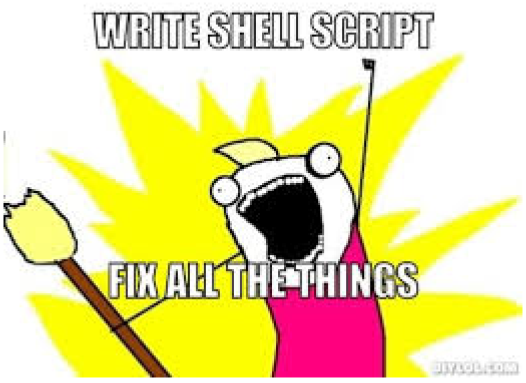
Shell Scripting

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter. Use this guide to get a great hold on shell scripting!

Index Of Contents

Scripts

You might have came across the word 'script' a lot of times, but what is the meaninig of a script?
So basically, a script is a command line program that contains a series of commands to be executed. These commands are execued by an interpreter.
Anything you can put into a command line, you can put in a script. And, scripts are great for automating tasks. If you find yourself repeating some commands frequently, you can, rather you should, create a script for doing it!



Our first script

```
#!/bin/bash
echo "My First Script!"
```

To run it:

```
$ chmod 755 script.sh
$ ./script.sh
```

Find the code here

Shebang

A script starts with #! **Path To Bash**

`#` is often called sharp and `!` is called Bang, hence the name sharp bang, but generally people say it **shebang** instead of sharp bang.

Comments

Comments are started by a `#` sign, anything after pound sign on that line is ignored.

Example

```
#!/bin/bash
echo "Hello World!"
# This line won be executed!
```

Basic Examples Of Shell Scripts

- Using csh as interpreter

```
#!/bin/csh
echo "This script uses csh as the interpreter!"
```

- Using ksh as interpreter

```
#!/bin/ksh
echo "This script uses ksh as the interpreter!"
```

- Using zsh as interpreter

```
#!/bin/zsh
echo "This script uses zsh as the interpreter!"
```

Use it or not?

If a script does not contain the shebang, the commands are executed using your shell, so there are chances that the code might run properly, but still, that isn't the correct way of doing it!
Different shells have slightly varying syntax.

More than just shell scripts!

You dont have to use shell as the interpreter for your scripts. For example, you can run a python script too by supplying the path in shebang.

```
#!/usr/bin/python
print "This is a python script!"
```

To run it:

```
$ chmod 755 name.py
$ ./name.py
```

Find the code here

Variables

Variables are basically storage location that have a name and can store some data which can be changed in future.

Syntax For Variables

```
VARIABLE_NAME = "Value"
```

Important

- Variables are case sensitive
- By convention, variables are uppercase
- To use a variable, just write the variable name followed by the `$` sign

Examples Of Variables

- Example 1

```
#!/bin/bash
MY_NAME="Madhav Bahl"
echo "Hello, I am $MY_NAME"
```

Download the code

- Example 2

```
#!/bin/bash
MY_NAME="Madhav Bahl"
echo "Hello, I am ${MY_NAME}"
```

Download the code

- Example 3: Assign command output to a variable

```
#!/bin/bash
CONTENTS=$(ls)
echo "The contents of this directory are: "
echo "$CONTENTS"
```

An alternative:

```
#!/bin/bash
CONTENTS=`ls`
echo "The contents of this directory are: "
echo "$CONTENTS"
```

Download the code

- Example 4

```
#!/bin/bash
SERVER_NAME=$(hostname)
echo "This script is being run on ${SERVER_NAME}"
```

Download the code

Variable Names

Alphanumeric characters, starting with an alphabet or underscore and can contain digits or underscores in between.

Valid Variable Names

- THIS3VARIABLE="ABC"
- THIS_IS_VARIABLE="ABC"
- thisIsVariable="ABC"

Invalid Variable Names

- 4Number="NUM"
- This-Is-Var="VAR"

No special character apart from underscore is allowed!

Tests

Scripts are basically needed to remove the need of again and again typing the commands which you use frequently or basically automating tasks. But, what if the script you wrote needs to execute differently under different circumstances? You can make the decisions using tests.

Syntax for tests

```
[ condition-to-test-for ]
```

Example

```
[ -e /etc/passwd ]
```

This test checks whether /etc/passwd exists, if it does, it returns true (or, it exits with a status of 0). If the file doesnt exists, it returns false (status 1).

File Test Operations

```
  -d FILE_NAM  # True if FILE_NAM is a directory
  -e FILE_NAM  # True if FILE_NAM exists
  -f FILE_NAM  # True if FILE_NAM exists and is a regular file
  -r FILE_NAM  # True if FILE_NAM is readable
  -s FILE_NAM  # True if FILE_NAM exists and is not empty
  -w FILE_NAM  # True if FILE_NAM has write permission
  -x FILE_NAM  # True if FILE_NAM is executable
```

String Test Operations

```
  -z STRING  # True if STRING is empty
  -n STRING  # True if STRING is not empty
  STRING1 = STRIN2 # True if strings are equal
  STRING1 != STRIN2 # True if strings are not equal
```

Arithmetic Operators

```
  var1 -eq var2  # True if var1 is equal to var2
  var1 -ne var2  # True if var1 not equal to var2
  var1 -lt var2  # True if var1 is less than var2
  var1 -le var2  # True if var1 is less than or equal to var2
  var1 -gt var2  # True if var1 is greater than var2
  var1 -ge var2  # True if var1 is greater than or equal to var2
```

Making Decisions

Just like any script, shell scripts can make decisions based on conditions.

The IF Statement

Syntax:

```
if [ condition-is-true ]
then
  command 1
  command 2
    ...
    ...
  command N
fi
```

Example:

```
#!/bin/bash
SHELL_NAME="bash"

if [ "$SHELL_NAME" = "bash" ]
then
  echo "You are using bash shell"
fi
```

IF-ELSE Tree

Syntax:

```
if [ condition-is-true ]
then
  command 1
  command 2
    ...
    ...
  command N
else
  command N+1
  command N+2
    ...
    ...
  command M
fi
```

Example:

```
#!/bin/bash
SHELL_NAME="bash"

if [ "$SHELL_NAME" = "bash" ]
then
  echo "You are using bash shell"
else
  echo "You are not using the bash shell"
fi
```

IF-ELIF Ladder

Syntax:

```
if [ condition-is-true ]
then
  command 1
elif [ condition-is-true ]
then
  command 2
elif [ condition-is-true ]
then
  command 3
else
  command 4
fi
```

Example:

```
#!/bin/bash
SHELL_NAME="bash"

if [ "$SHELL_NAME" = "bash" ]
then
  echo "You are using bash shell"
elif [ "$SHELL_NAME" = "csh" ]
then
  echo "You are using csh shell"
else
  echo "You are not using the bash or csh shell"
fi
```

For Loop

Loops can execute a block of code a number of times and are basically used for performing iterations. Just like any other programming language, shell scripts also have for loops.

Syntax

```
for VARIABLE_NAME in ITEM_1 ITEM_N
do
  command 1
  command 2
    ...
    ...
  command N
done
```

```
for (( VAR=1;VAR<N;VAR++ ))
do
  command 1
  command 2
    ...
    ...
  command N
done
```

Example

```
#!/bin/bash
for COLOR in red green blue
do
  echo "The Color is: ${COLOR}"
done
```

Alternatively,

```
#!/bin/bash
COLORS="red green blue"
for COLOR in $COLORS
do
  echo "The Color is: ${COLOR}"
done
```

Example 2

In this simple example we will see how to rename each file with .txt format

```
#!/bin/bash
FILES=$(ls *txt)
NEW="new"
for FILE in $FILES
do
  echo "Renaming $FILE to new-$FILE"
  mv $FILE $NEW-$FILE
done
```

Positional Parameters

Some arguements or parameters can be passed when we call the script.

For Example:

`$script.sh param1 param2 param3 param4`

All the parameters will be stored in:

```
  $0 -- "script.sh"
  $1 -- "param1"
  $2 -- "param2"
  $3 -- "param3"
  $4 -- "param4"
```

Example

```
#!/bin/bash
echo "Running script: $0"
echo "Parameter 1: $1"
echo "Parameter 2: $2"
echo "Parameter 3: $3"
echo "Parameter 4: $4"
```

To access all the parameters, use `$@` sign

Example

```
#!/bin/bash
echo "Running script: $0"
for PARAM in $@
do
  echo "Parameter: $PARAM"
done
```

User Input-STDIN

`read` command accepts STDIN (Standard Input)

Syntax

```
read -p "PROMPT MESSAGE" VARIABLE
```

Example

```
#!/bin/bash
read -p "Please Enter You Name: " NAME
echo "Your Name Is: $NAME"
```

Exit Status

Every command returns an exit status, also called the return code which ranges from 0 to 255. Exit status are used for error checking.

- 0 means success
- Any code other than 0 means an error condition.

To find out what an exit status for a command means, one can look for the documentations or manual using `man` or `info` command.

`$?` contains the return code of previously executed command.

[Example:](./exit-status/error.sh)

```
#!/bin/bash
ls /randomDirectory   # Any Directory which does not exist
echo "$?"    # This command will return 2
```

Another Example:

```
#!/bin/bash
HOST="google.com"
ping -c 1 $HOST     # -c is used for count, it will send the request,
number of times mentioned
RETURN_CODE=$?
if [ "$RETURN_CODE" -eq "0" ]
then
  echo "$HOST reachable"
else
  echo "$HOST unreachable"
fi
```

Logic Operations

Shell scripts supports **logical AND** and **logical OR**.

AND

The AND Operator results true if all the conditions are satisfied.

`&& = AND`

Example

```
#!/bin/bash
MY_VAR=10
if [ "$MY_VAR" -ge 5 ] && [ "$MY_VAR" -le 15 ]
then
  echo "Given variable is within the range"
fi
```

OR

The OR Operator results true if any one the conditions are satisfied.

`|| = OR`

Example

```bash
#!/bin/bash
MY_VAR=10
if [ -e ls /etc ] || [ -e ls /lib ]
then
  echo "Given variable is within the range"
fi
```

Some Sample Programs

Any field whether it may be computer science or any other, requires practise. Please consider this as a practise assignment, and try to do all the questions yourself and then see the code.

Click here to see the practise programs #1