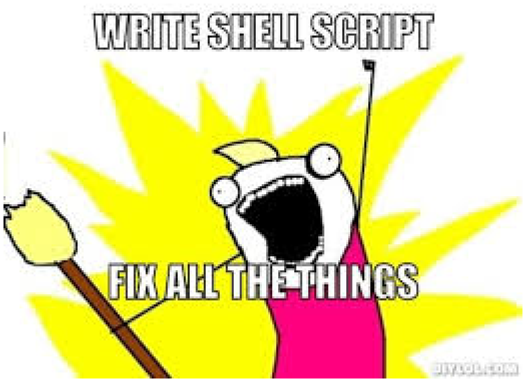
Shell Scripting

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.

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

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

An alternative:

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

Download the code

- Example 4

```bash
#!/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
```