Implementing Linux Automation Using Shell Scripts

Andrew Mallett
LINUX AUTHOR AND TRAINER

@theurbanpenguin   www.theurbanpenguin.com

Objectives

Understanding shell script in BASH

- Loops
- Conditionals
- Variables
- Built-in commands
- Redirection
- Common utilities

Shell Scripts

Shell scripts are commands grouped together in a text file that can be executed. These commands can be supported with additional operators. The operators are defined within the shell itself and can vary from one shell to another. We concentrate on BASH

Git Repo

To maintain version control of any script we create will we add them to our existing git repository. Making the time we spent learning git worthwhile. Sample files can also be obtained from the comptia-automation repos downloaded previously

```
$ cd ~/myproject
$ git checkout -b shellscripting
$ mkdir shellscripting
$ vim info.sh
echo "=================="
hostname
hostname -I
uname -r
echo "=================="
$ bash info.sh
$ git add . && git commit -m "First sample of info script"
```

Your First Shell Script

Making sure that we have fired up our working vagrant ubuntu system. We move into your git repository, previously created and create a new directory named shellscripting

```
$ file info.sh
$ sed -i '1i #!/bin/bash' info.sh
$ file info.sh
$ chmod a+x info.sh
$ ./info.sh
$ git commit -a -m "Added execute and shebang to info.sh"
```

Make Standalone

Adding a shebang line to the script tells the system which interpreter to use. Adding the execute permission allows independent execution, without neediing to run bash to open the file

Demo

Working in the git repository:

- Create directory for scripts
- Add script
- Use sed to insert shebang
- Add permissions
- Combine git add and git commit

Rock Star Script

- Adding variables can help the script become more readable
- The header and footer are OK but we must carefully count the correct number of = signs. We can use a loop to to the heavy lifting for us

info.sh

```
#!/bin/bash
INFO_HOSTNAME=$(hostname)
INFO_IP=$(hostname -I | cut -f1 -d" ")
INFO_KERNEL=$(uname -r)
for i in {1..25} ; do
  echo -n =
done
echo
echo "Host: $INFO_HOSTNAME"
echo "IP: $INFO_IP"
echo "Kernel: $INFO_KERNEL"
for i in {1..25} ; do
  echo -n =
done
echo
```

Demo

Improving the info.sh

- We add loops to generate headers
- Use variables to collect data

```
$ cat /etc/os-release
$ source /etc/os-release
$ echo $PRETTY_NAME
```

Using Imported Variables

The /etc/os-release file has data within variables that we can use in our own scripts. To read the variables into the current shell or script we use the source command. Adding these sourced variables to our script can add additional functionality

Demo

Using the Built-in Source Command:

- We add imported variables to our script

```
$ for u in bill bob jayne; do sudo useradd -m $u ; done
```

Working with Loops

We have used the FOR loop to iterate through a range, we can also use loops from the command line as well as in scripts. This can be a quick and easy way to carry out repetitive tasks

```
#!/bin/bash
PWD_OK="false"
while [ "$PWD_OK" != "true" ] ; do
  read -sp "Enter password: "
  echo ""
  PWD_LEN=$(echo -n "$REPLY" | wc -m)
  PASSWORD=$(openssl passwd -6 "$REPLY")
  if [ "$PWD_LEN" -gt 6 ]; then
    PWD_OK="true"
    echo "$PASSWORD"
  fi
done
```

WHILE Loops

We can use either WHILE or UNTIL loops to check conditions and keep looping until they are either true or false. We can also use conditionals within the loop to trigger it to end. In this example we use while, read and if statements and we will discover more about these statements in the demo

Demo

Creating a Password Generation Script

- In this script we use:
- Variables
- Use WHILE/UNTIL loops
- Read user input
- Test length of entered data
- Use IF statements to end loop

```
$ sudo -i
$ grep -q 'vagrant' /etc/passwd
$ echo $?
$ grep 'fred' -q /etc/passwd || useradd -m fred
$ grep 'fred' -q /etc/passwd && passwd fred
```

Testing Simple Logic

The variable $? will contain the success or failure of the previously executed command. We only want to create the user if the search for the user account fails, ||. We only want to create the user's password if the search succeeds, &&

```
if [[ $# -ne 1  ]] ; then
  read -sp "Enter password: "
  echo ""
else
  REPLY=$1
fi
```

Using Script Arguments

Scripts can take arguments; $0 is always the script name. The number or arguments can be read using $#. $1 is the first argument. Using [[ ]] as your test brackets automatically protects reserved characters in variables.

Demo

Improving Password Generator:

- Exit the script if password too short
- Only prompt for password if not supplied from the CLI

Demo

Creating Users:

- In this script we can investigate &&, || and $?
- Building a script to create users only if they don't exist
- User passwords can be set from the existing script

```
$ awk -F: '/vagrant/ { print $0 }' /etc/passwd

$ awk -F: '/vagrant/ { print "User: " $1 ; print "\tUID: " $3 }' /etc/passwd
```

Using AWK

We have seen the use of both sed and grep. The big brother to these commands is awk which is its own scripting language

Demo

Introduction to AWK

Summary

Shell Scripting Using BASH

- file command
- shebang
- echo (-n)
- read (-s -p REPLY)
- for loops
- brace expansion for i in {1..25}
- command expansion $(uname -r)
- variables
- source command
- while and until loops
- exit codes and $?

Implementing Configuration Management and IaC