Basic shell scripting with BASH

Shell scripting is much easier than you think. Essentially, a shell script is a bunch of normal shell commands (defined as commands which you can run at a command prompt) contained in a file. Instead of running the individual commands, you call the file containing them and the commands are run in the order listed.

The first note we'd like you to be aware of is that any command which can be entered in at the shell prompt, may be included in a shell script.

For example:

If you can enter in the following command at the shell prompt:

```
$ ls -ld /etc
```

You could include it in a shell script!

In order for you to execute a shell script you need to have the execute permission, failing which the following error would be returned.

```
# foo
-bash: /usr/local/bin/foo: Permission denied
```

Generally the first line in your shell script is called a shebang and specifies the path to the interpreter. The interpreter is the program which processes all the commands listed in your shell script and may look like this:

```
#!/bin/bash
```

!!! NOTE !!!

Even though the shebang begins with a # character, it does not mean that the line is commented. This exception takes place because of the ! which follows. Any other line which begins with a # is seen as a comment and will not be processed by your interpreter.

Storing your shell script

If the shell script is only to be used by a single user, then store that shell script in ~/bin as each user on your system automatically has a bin subdirectory in their home directories which is part of their PATH statement.

Should the shell script be used by all normal users, then store the script in /usr/local/bin as this directory is part of all users PATH statements.

If the shell script is to be used by all users but is somewhat administrative in nature, then store the script in /usr/local/sbin and again, this directory is part of all users PATH statements.

Output

These are examples of some of the commands which you can use to give feedback to the user of your script.

Command	Purpose
echo	Displays the text following the command
printf	Displays the text following the command (has more features than echo)
cat	Displays the contents of the file top to bottom
tac	Displays the contents of a file bottom to top
sort	Sorts text in a defined order
uniq	Removes duplicate lines
grep	Filters out text based on regular expressions
fmt	Formats text
head	Displays the specified first few lines of a file
tail	Displays the specified last few lines of a file
tr	Translates characters (like uppercase to lowercase)
sed	Streaming editor

Our first script:

The purpose of the script below is to print out the text “Hello world”

```
#!/bin/bash
echo “Hello world”
```

To have this script available for all users, store it in a file called foo in the directory /usr/local/bin and don’t forget to make it executable using the command:

```
# chmod +x /usr/local/bin/foo
```

Variables

A variable is a string of text (limited to a whole word) which has a value associated with it.

You may create a variable using the notation:

VARIABLE=VALUE

!!! NOTE !!!

Your variable's name doesn't have to be in uppercase, however, we recommend that you do this so that your variable's name stands out from the other text around it.

The value of your variable doesn't have to be in uppercase, it can be any case and it is entirely up to you.

To query the value of a variable use the echo command followed by the name of your variable preceded with a $ sign.

For example, to query the value of the variable called PATH:

```
# echo $PATH
/usr/lib64/qt-3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

Should your variable's value have special characters like spaces then you need to encapsulate it in quotation marks. 2 types of quotation marks exist.

1. Strong quotation marks : ' '
2. Weak quotation marks : " "

Strong quotation marks prevent any form of expansion in them whereas weak ones allow expansion.

For example, to create a variable called CURRENCY with the value US$:

```
# CURRENCY='$money'
# echo $CURRENCY
$money
```

If we used weak quotations instead of strong ones it would result in the following

```
# CURRENCY=”$money”
# echo $CURRENCY
<no output>
```

Variables may have underscores in their names as follows:

```
# FIRST_LAST="Fred Flintstone"
# echo $FIRST_LAST
Fred Flintstone
```

But what if our variables are called FIRST and LAST?

```
# FIRST=Fred
# LAST=Flintstone
```

To have them displayed at the same time with an underscore separating them we use:

```
# echo ${FIRST}_$LAST
Fred Flintstone
```

Failing which would result in:

```
# echo $FIRST_$LAST
Flintstone
```

Some special variables include:

$0	The name of the script itself
$1	The first argument passed to the command
$2	The second argument passed to the command
$3	The third argument passed to the command
$?	The exit status of the command
$#	The number of arguments passed to the command
$@	All the arguments passed to the command

!!! NOTE !!!

If the exit status code is 0 then it means that the command completed successfully, if it is anything other than 0 then there was some error.

Conditional branches

If you only wanted to execute a command should a condition be true you may use the if statement.

It follows the syntax:

```
if condition
then
        run these commands
else
       run these commands instead
fi
```

For example, in the file /usr/local/bin/fredcheck

```
if “$NAME” = “fred”
then
        echo “yaba daba doo”
else
        echo “You’re not who I’m expecting”
fi
```

Explanation:

If the value of the variable NAME has the value fred, the script will echo yaba daba doo at the screen otherwise it returns the string You’re not who I’m expecting.

Branching can sometimes be complicated to the point where you’d want to run different commands given different values of a variable. This is a perfect case for case!

It follows the syntax:

```
case $VARIABLE in
value1) run these commands;;
value2) run these commands;;
```

```
value3) run these commands;;
*) run these commands;;
esac
```

For example, in the file /usr/local/bin/flintstonecheck:

```
#!/bin/bash
NAME=fred
case $NAME in
fred) echo "It's the hero of the show";;
wilma) echo "She's Fred's lovely red headed wife";;
barney) echo "Fred's best friend, neighbor and co-worker";;
betty) echo "She's Barney's wife and Wilma's best friend and neighbor";;
*) echo "We don't know who this is"
esac
```

Explanation:

In case the value of the variable NAME is fred then the command associated with the variable will be executed. The same applies to wilma, barney and betty. If the value of the variable NAME is something other than the ones listed, the command associated with the * will be executed with is echo “We don’t know who this is”.

Getting input from your script

Let’s say that you wanted the user to be prompted for the variable NAME, we could use the read command. The read command creates a variable and follows the syntax:

```
read VARIABLE_NAME
```

For example, we could change the script above to:

```
#!/bin/bash
read NAME
case $NAME in
fred) echo "It's the hero of the show";;
wilma) echo "She's Fred's lovely red headed wife";;
barney) echo "Fred's best friend, neighbor and co-worker";;
betty) echo "She's Barney's wife and Wilma's best friend and neighbor";;
*) echo "We don't know who this is"
```

```
esac
```

Explanation:

Same as the explanation earlier, except this time the user is prompted for the value of the variable NAME.

Looping

Loops give us the ability to perform tasks over and over again given a true condition.

We have 3 types of loops:

1. for loop (works for a condition to be true)
2. while loop (works while a condition is true)
3. until loop (works until a condition becomes true)

Syntaxes:

```
for condition
do
        commands
done
```

or

```
while condition
do
        commands
done
```

or

```
until condition
do
        commands
done
```

Examples:

Here we will create 3 scripts with the 3 loop types which will count to 10.

In /usr/local/bin/counter1:

```
1. #!/bin/bash
2. for i in {1..10}
3. do echo $i
4. done
```

Results in:

```
# counter1
1
2
3
4
5
6
7
8
9
10
```

Explanation:

1. Our shebang
2. We begin our loop stating that the variable i has numerous values starting at 1 to 10
3. Our loop commands begin
4. The current value of the variable $i is displayed
5. The loop commands terminate

In /usr/local/bin/counter2:

```
1. #!/bin/bash
2. declare -i i=1
3. while [ "$i" -le "10" ]
4.     do
5.         echo $i
6.         i=$i+1
7.     done
```

Results in:

```
# counter2
1
2
3
4
5
6
7
8
9
10
```

Explanation:

1. Our shebang
2. We are declaring a variable called i as an integer which has the intial value of 1
3. We begin our until loop stating that the loop will be executed while the value of the variable $i is tested to be less or equal to 10
4. Our loop commands begin
5. The current value of the variable $i is displayed
6. The variable $i is incremented by 1
7. The loop commands terminate

In /usr/local/bin/counter3:

```
1. #!/bin/bash
2. declare -i i=1
3. until [ "$i" -gt "10" ]
4.     do
5.         echo $i
6.         i=$i+1
7.     done
```

Results in:

```
# counter2
1
2
3
4
5
6
7
8
9
10
```

Explanation:

Very similar to the while loop example before.

1. Our shebang
2. We are declaring a variable called i as an integer which has the intial value of 1
3. We begin our until loop stating that the loop will be executed until we test the value of the variable $i to be greater than 10
4. Our loop commands begin
5. The current value of the variable $i is displayed
6. The variable $i is incremented by 1
7. The loop commands terminate

Lab activity

Create a shell script called /usr/local/sbin/brainiac which, when passed the following arguments, executes the associated actions with it.

security	prints out the filewall rules and the output of lastlog to the file /root/security and emails it to fred with the subject “Security review on $HOSTNAME”
software	lists all the software packages with are currently insteaded in an email to fred with the subject “Software package list on $HOSTNAME”
reboot	reboots the server and adds the text “Reboot by $USERNAME” added to /var/log/messages