Bash Scripting Part 1

Patrick TenHoopen
WMLUG

What is the Shell?

The shell is a command interpretor.

Bash is the most common shell in use on Linux. It's a wordplay of “Bourne-Again shell” because it is based on another shell called Bourne.

What is a Script?

A script is a shell program.

A script could consist of just a file listing commands that you could issue at the command line. For example, this is a simple script named simplescript.sh and consists of two lines:

Contents of the simplescript.sh file:

```
cd /var/log
ls *.log
```

Running Scripts

You can run scripts two ways:

1. If the script file is not flagged as executable:

```
sh simplescript.sh
```

2. If it is flagged as executable (and is in the path):

```
simplescript.sh
```

Note: You can flag the file as executable by using:

```
chmod +x simplescript.sh
```

Bash Scripts

If you use actual bash commands (discussed next), then the file needs to be setup with the first line containing:

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

This indicates that this file is a script and that it should be fed into (i.e., ran with) the bash shell. The `#!` is commonly referred to as “shebang” (or hashbang).

Comments

The # (pound or hash sign) is used to indicate comments and any text following it on that line is ignored.

Whole line comments:

```
# Pound signs indicate comment lines and are
# ignored by the script processor
```

End of line comments:

```
ls *.txt # List text files in current directory
```

Script Output

You can use the `echo` command to write text to the terminal window.

Tip: Enclose the text in double-quotes for best results.

To create a blank line, you can have an echo command with no text.

Some examples:

```
echo “Hello”    # this writes Hello
echo            # this writes a blank line
```

Variables

Variables are used to hold information. They can contain text or numbers.

```
echo “Hello, $USER”
```

This outputs the text “Hello, ” followed by the name of the current user, using a system variable named USER.

System Variables

Some useful system variables:

`HOME`	Current user's home directory
`HOSTNAME`	Hostname of the computer running the script
`USER`	Current user
`PWD`	Current directory

Escaping Special Characters

When outputting special characters using an echo command, you must prefix the characters with a \ (back slash).

Special characters include $, ", and \.

Escaping Special Characters, Cont.

For example:

```
echo "System variable, \$USER, is the current user."
```

This outputs:

```
System variable, $USER, is the current user.
```

Without the \, it would expand the variable and output:

```
System variable, pat, is the current user.
```

Creating Variables

To create a variable, specify the name of the variable on the left, follow it with an equals sign and then the value. **Note:** There can be no spaces before or after the equals sign.

```
# a text/string variable
MyVar=“Some text”

# a numeric variable
aNum=17
```

Using Variables

When you reference (use) a variable, you need to add a `$ in front of it`.

Using variables in a script:

```
echo $MyVar
# prints: Some text

echo “Variable aNum = $aNum”
# prints: Variable aNum = 17
```

Coloring Echo Output

You can colorize the output from the echo command by using ANSI escape sequences.

```
echo -e "This word is \e[0;32mgreen\
e[0m."
```

Output:

```
This word is green.
```

Coloring Echo Output

The `-e` option lets echo interpret the ANSI escape sequences which have a beginning and ending tag.

The `\e[0;32m` starts coloring text in green.

The `\e[0m` stops coloring and reverts back to normal output.

Coloring Echo Output

Beginning sequence detail:

`\e`	Escape character
`[`	Beginning tag
`0`	Foreground attribute
`;`	Attribute separator
`32`	Background attribute (32=green)
`m`	Ending tag

Coloring Echo Output

Foreground attributes:

0	Normal
1	Bold
4	Underlined
5	Blinking
7	Inverted

Note: There are no background attributes.

Coloring Echo Output

Foreground colors:

Color Code	**00**	**01**
30	Black	Gray
31	Red	Light Red
32	Green	Light Green
33	Brown	Yellow
34	Blue	Light Blue
35	Magenta	Pink
36	Cyan	Light Cyan
37	White	Light Gray

Coloring Echo Output

Ending sequence detail:

`\e`	Escape character
`[`	Beginning tag
`0`	Foreground attribute
`m`	Ending tag

Coloring Echo Output

To change the foreground and background color, just add the attribute for the background color after the foreground color.

```
echo -e "This is \e[0;31m\e[42mred on green\
e[0m."
```

Output:

```
This is red on green.
```

Coloring Echo Output

Variables can be used to make it easier:

```
SUCCESS="\e[0;32m"
ERROR="\e[0;31m"
WARNING="\e[1;33m"
NORMAL="\e[0m"

echo -e "${ERROR}An error occurred.$NORMAL"
```

Script Input

You can read input from the user using the `read` command.

```
echo “What is your favorite color?”
read favcolor
```

The above commands ask the user what their favorite color is and stores it in the `favcolor` variable.

The read command can also show the prompt:

```
read –p “What is your name? ” username
```

If Statement

The `IF` statement checks for a true condition and, if it has one, it performs the commands listed.

```
if condition-true
then command1
     command2
fi
```

Alternate syntax:

```
if condition-true; then
   command1
   command2
fi
```

If Statement, Cont.

The `else` keyword is used for the non-true condition:

```
if condition-true
then command1
else
   command2
fi

if condition-1-true
then command1
else if condition-2-true
   command2
else
   command3
fi
```

If Statement, Cont.

Some examples:

```
if grep -q Linux MyFile.txt
then echo "The file contains the word Linux"
fi

if grep -q Linux MyFile.txt
then echo "The file contains the word Linux"
else echo "The file does not contain the word
Linux"
fi
```

Test Command

The test command is a Bash command that tests file types and compares strings.

```
if test "5" = "7"
then echo "5 equals 7"
else
   echo "5 does not equal 7"
fi
```

Test Command

Using the `[ ]` brackets surrounding a condition in an `if` statement invokes the `test` command automatically. **Note:** Spaces are needed around the condition.

This is equivalent to the above code:

```
If [ "5" = "7" ]
then echo "5 equals 7"
else
   echo "5 does not equal 7"
fi
```

Test Command Uses

Testing if a file exists:

```
if [ -e file.txt ]
```

Testing if the current user has write privileges to a file:

```
if [ -w file.txt ]
```

Testing whether a string is null:

```
if [ -n "$var" ]
```

Exit Status

The `exit` command can be used to exit a script.

Commands return an exit status on completion. This is also referred to as the exit code or return status. An exit status of 0 indicates success while a non-zero value indicates failure.

The `$?` variable stores the exit code of the last command executed and can be used to check for errors.

Putting It All Together

Script Demo

References

For more information, see the Advanced Bash-Scripting Guide:

http://tldp.org/LDP/abs/html/index.html

Other sources:

https://en.wikipedia.org/wiki/Shebang_(Unix)