Introduction to Shell Scripting

or how to be a shell Ninja

Dennis Kibbe
dennis.kibbe@mesacc.edu

Mesa Community College
Network Academy

Fall 2016

Table of Contents

Why Script?

A bit more information about this

Scripting lets system administrators:

- Combine commands to accomplish a task.
- Automate repetitive processes.
- Branch after testing for a condition.
- Improve efficiency.

System Files

Bash shell scripts start many system processes.

Starts the rsyslog Daemon

```
NAME=rsyslog

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME
```

Which Shell?

Finding your default shell

The SHELL Variable

```
$ env | grep SHELL
SHELL=/bin/bash
```

Which Shell?

Finding your default shell

/etc/passwd

```
tux:x:1000:1000:Tux Penguin,,,:/home/tux:/bin/bash
```

Programming At The Command Line

You can enter programming commands directly at the command line.

At the Command Line

```
$ for i in {1..5}
> do touch file\$i
> done
```

Editing Scripts

Shell scripts must be edited in a text editor that saves the script as plain, unformatted ASCII text. Common editors are:

- Vi/Vim
- Emacs
- Nano
- Gedit
- notepad++

Editing Scripts

Editors not to use are any word processor such as:

- Libre Office
- OpenOffice.org
- M$ Word

Line Endings

Windows uses both a carriage return and a new line character. Linux and Unix use new line only. Editing a script in Windows notepad will break the script.

Line endings

```
#!/bin/bash^M
    # This script was edited in Windows Notepad^M
    ^M
    echo "Hello World"^M
    ^M
```

Anatomy Of A Shell Script

Common parts of a shell script are:

- Shebang
- Comments
- Variables
- Functions
- Exacutable code

Shebang

The Shebang line assures that the script is run as a Bash script and must be the first line in the script.

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

Shebang (Cont.)

The Shebang line assures that the script is run as a Bash script and must be the first line in the script.

```
#!/usr/bin/env bash
```

Comments

Any text after the hash tag character # is a comment. Here is an example from /.bashrc.

Comments

```
# uncomment for a colored prompt, if the terminal has the capability; tu
# off by default to not distract the user: the focus in a terminal windo
# should be on the output of commands, not on the prompt
force_color_prompt=yes # This is also a comment.
```

Variables

Defining variables in your script will make it easier to modify.

```
#!/usr/bin/env bash

# Set message to print.
message="Hello world"

# Print the value of message to STDOUT.

echo "$message"

# END
```

Variables (Cont.)

The dollar sign in from of the variable name tells echo to print the value of the variable and not just the word message.

```
$ echo $message
```

Functions

Functions are mini-scripts you can call inside your script. First define the function...

```
#!/usr/bin/env bash

# define usage function

usage(){
echo "Usage: $0 greeting"
exit 1
}
```

Functions (Cont.)

then call it when needed.

```
if [[ ! $1 ]]; then
usage
else
echo "$1"
fi

# END
```

Conditionals

Depending on the outcome of a test something is done. Example from /.bashrc.

```
f# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

Code

Finally the code that does the work.

```
$ echo "Hello World"
```

Positional Parameter

Positional Parameters let you feed arguments to the script. Use quotes around multiple words.

```
$ myscript.sh Hello World
```

Positional Parameter (Cont.)

Each positional parameters is identified by a number 1 to 9.

```
#!/usr/bin/env bash

# Prints the value of the first two positional parameters.

echo "$1" "$2"

# END
```

For Loop

Do iterations over a list.

```
$ for i in apple banana cherry "red grapes" ; do echo "$i"; done
```

If/Then/Else

```
if [[ ! $1 ]]
then echo "I have nothing to print."
else
echo "$1"
fi
```

Conditionals

```
if [[ ! $1 ]]; then
echo "I have nothing to print."
else
echo "$1"
fi
```

Variables

Variable are easy to set.

```
message="Hello World"
echo $message
```

Variables (Cont.)

You can used pre-definded environmental variables, too.

```
echo "Today is $(date +%A) and you are logged into $HOSTNAME as $USER.
```

While Loops

```
    #!/bin/bash

# Set inital value for variable
counter=0

while [ $counter -lt 10 ]; do
echo "The counter is $counter"
sleep 1
let counter=$counter+1
done

# END
```

Exit Code

Every program returns an exit code when it finishes. An exit code of zero means that the command succeeded.

```
$ ls /tmp
$ echo $?
0
```

Exit Code (Cont.)

Every program returns an exit code when it finishes. An exit code other than zero means that the command failed.

```
$ ls /temp
ls: cannot access /temp: No such file or directory
$ echo $?
1
```

Getting Help

```
help test
test: test [expr]
Evaluate conditional expression.

Exits with a status of 0 (true) or 1 (false) depending on the evaluation

The behavior of test depends on the number of arguments.  Read the bash
```

Resources

- Shell Check
- Greg's Bash Guide
- Shell Explained
- Google Style Guide
- How do I convert between Unix and Windows text files?