Lecture 9: Intro to Bash Scripting

CS2042 - UNIX Tools

October 20, 2008

Lecture Outline

Scripting 101

Definition:

A *script* is very similar to a program, although it can only be run within a certain context. In other words, it can't run on its own like a program can. *Shell scripts* are scripts designed to run within a command shell like **bash**.

Scripts are written in a scripting language, like perl or ruby or python. They are then run using an interpreter. In our case, the scripting language and the interpreter are both **bash**.

The Shebang

All of the shell scripts we'll see in this course begin the same way: with a *shebang* (**#!**). This is followed by the full path of the shell we'd like to use as an interpreter: */bin/bash*.

Example:

```
#! /bin/bash
# This is the beginning of a shell script.
```

- Any line that begins with a # (except the shebang) is a comment.
- Comments are ignored during execution - they serve only to make your code more readable.

Setting Variables

Creating and setting a variable is this easy:

Example:

MYVAR="A new variable!"

Our new variable *MYVAR* can now be accessed as *$MYVAR*:

Example:

echo $MYVAR

```
A new variable!
```

- Note that there are no spaces around the equals sign when you set a variable - this is important!

Lecture Outline

Hello World

Type this into a new text file (let's call it *hello.sh*):

Example:

```
#! /bin/bash
echo "Hello World!"
```

Now set your file permissions to allow execution:

Example:

```
chmod +x hello.sh
```

And you can run your first shell script!

./hello.sh

- `Hello World!`

Hello World - String Version

Let's add a twist and use a variable in *hello2.sh*:

Example:

```
#! /bin/bash
STRING="Hello again, world!"
echo $STRING
```

Set your permissions and run the script:

chmod +x hello2.sh && ./hello2.sh

- `Hello again, world!`

A Backup Script

Here is something a little more practical - a simple script to back up all the files in your documents directory:

Example:

```
#! /bin/bash
tar -czf ~/backups/backup.tar.gz ~/documents/
```

This script makes use of the **tar** archiving command:

Making Tarballs:

tar -c(z/j)f <dest_archive> <source>
tar -x(z/j)f <archive>

- **-c** version creates a new archive from a source file/dir
- **-x** extracts an existing archive to the current dir
- pick either -z or -j options: -z for a .tar.gz, -j for a .tar.bz2

Lecture Outline

Using Command Output

In order to use the output of a command within our script, we must set it apart using backticks
`command`
or in this fashion:
$(command)

Example:

```
#! /bin/bash
echo date
echo `date`
```

- The backtick method requires escaping of '\', '`', and '$'

Backup Script: Revisited

Let's try to fix up our backup script a little. Maybe we want to save our documents in an archive without overwriting our old backup files.

Example:

```
#! /bin/bash
tar -czf ~/backups/docs_$(date +%d%b%y).tar.gz ~/documents/
```

- Today, will write to a file named docs_20Oct2008.tar.gz

- The manpage for **date** is very helpful if that formatting is confusing.

Lecture Outline

Is This a Test?

Occasionally we need to perform one action or another depending on whether a condition is true or false. We call the checks for these conditions "*tests*".

Testing a Condition

There are many condition expressions that can be checked in your shell. To test an expression, use one of these:

- **test EXPRESSION**
- **[EXPRESSION]**

These commands return with an "exit status" of 0 if the condition is true, or 1 if it is false.

- Hint: you can check the exit status of the last program using the **$?** variable.

Primary Expressions

The **test** command has many built-in test expressions known as "primaries" which we'll use for the majority of our conditionals. While a full list can be found in the (thankfully brief) **test** manpage, here are a few handy ones:

Useful Primaries

Expression	Returns true if:
[! EXPR]	EXPR is false
[EXPR1 -a EXPR2]	EXPR1 and EXPR2 are true
[EXPR1 -o EXPR2]	EXPR1 or EXPR2 is true
[INT1 -(eq/ne) INT2]	ints are equal/not equal
[INT1 -(gt/lt) INT2]	INT1 greater than/less than INT2
[-o optionname]	shell option optionname is enabled
[STR1 == STR2]	the strings are equal

File Primaries

Scripts are frequently used to operate on files, so we have access to a long list of file-related primary test expressions:

File Tests

Expression	Returns true if:
[-e FILE]	File exists
[-r FILE]	Exists and is readable
[-s FILE]	Exists and has size > 0
[-w FILE]	Exists and is writable
[-O FILE]	Exists and is owned by you
[-G FILE]	Exists and is owned by your group

There are many additional expressions listed in the **test** manpage - recommend you skim it!

Lecture Outline

Standard Syntax

The most compact syntax of the **if** command is:

- **if TEST-EXPR; then CONDITIONAL-COMMANDS; fi**

TEST-EXPR list is run first - it can be either a primary or any command with an exit status. If the return status is true, CONDITIONAL-COMMANDS are executed. Otherwise, nothing happens.

Example:

```
#! /bin/bash
if [ $? -eq 0 ]
then echo "Last command exited cleanly!"
fi
```

Adding an Else

If we wish to choose between two actions, rather than choosing whether or not to perform one, we can add an **else** expression:

Example:

```
#! /bin/bash
if [ $? -eq 0 ]
then echo "Last command exited cleanly!"
else echo "Uh-oh - non-zero exit status!"
fi
```

Note:

You need to use semicolons after each statement if A) you are typing these commands into a shell, or B) you are putting more than one command on the same line.

If's Full Form

If's full syntax is as follows:

if TEST-COMMANDS; then CONSEQUENT-COMMANDS;

elif MORE-TEST-COMMANDS;
then MORE-CONSEQUENT-COMMANDS;

else ALTERNATE-COMMANDS;

fi

Lecture Outline

Using Arguments

We have used a lot of commands that accept parameters to change the way they operate. This makes them much more flexible than if they had hard-coded values for all of those parameters.

- How do we add that kind of flexibility?

Command-line Arguments

This is really easy - each argument passed to our script is assigned a variable **$1, $2, $3**, etc. **$0** stores the name of the script, and **$#** gives us the number of arguments.

Example:

```
#! /bin/bash
echo "\$1 = $1"
```

- Try running that in a script with an extra argument!

Correcting Command Usage

Often our scripts will require a certain number of arguments. If it is not given 3 arguments, the short script below will alert the user and then exit.

Example:

```
#! /bin/bash
if [ ! $# -eq 3 ]
then echo "Correct usage: $0 arg1 arg2 arg3"
exit
fi
```

./example.sh -something anotherthing

- `Correct usage:  ./example.sh arg1 arg2 arg3`