Bash: scripting

Glenn Bruns
CSUMB

Lecture Objectives

After this lecture, you should be able to:

- put bash code into a script
- write bash 'if' and 'test' statements

A simple Bash script

What if you have to do this a lot?

```
$ ps -eF | tail -n +2 | awk '{print $1}' | sort | uniq
```

Don’t type – create a bash script “users.sh”:

```
#!/usr/bin/env bash   # tells OS how to interpret file
ps -eF | tail -n +2 | awk '{print $1}' | sort | uniq
```

Give the script execute permission and run it:

```
$ chmod +x users.sh          # give ‘execute’ permission
$ ./users.sh                 # run your script
```

Another scripting example

```
$ cat copybk.bsh
#!/usr/bin/env bash
# make backup copies of .c files
for f in *.c; do cp $f $f.bak; done
$
$ chmod +x copybk.bsh
$
$ ls *.c
barrier-skeleton.c  richer-barrier.c  rwlock2.c  rwlock4.c
sbuf-skeleton.c
fsbuf-skeleton.c    rwlock1.c         rwlock3.c  rwlock-skeleton.c
simple-lock.c
$
$ ./copybk.bsh
$ ls *.bak
barrier-skeleton.c.bak rwlock1.c.bak rwlock4.c.bak  simple-lock.c.bak
fsbuf-skeleton.c.bak   rwlock2.c.bak rwlock-skeleton.c.bak
richer-barrier.c.bak   rwlock3.c.bak sbuf-skeleton.c.bak
$
```

Command line arguments

What if we want this script to work with any directory?

```
#!/usr/bin/env bash
ls -l /home/CLASSES/brunsglenn/data
```

Use $1 to refer to first command-line argument

```
#!/usr/bin/env bash
ls -l $1
```

Running the script:

```
$ ./lslong.sh /home/CLASSES/brunsglenn/data
```

Command-line arguments

```
$ cat > myscript.sh
#!/usr/bin/env bash
echo $0
$
$ chmod +x myscript.sh
$
$ ./myscript.sh
./myscript.sh
$
$ cat > myscript.sh
#!/usr/bin/env bash
echo $# $1
$
$ ./myscript.sh
0
$ ./myscript.sh foo bar
2 foo
$
$
$
```

$#	number of arguments
$i	ith argument

Command-line arguments

```
#!/usr/bin/env bash
if [ $# -eq 0 ]
then
  echo "missing parameter"
  exit 1
fi
touch $1
echo "touched "$1
$
$ ./myscript.sh
missing parameter
$
$ ./myscript.sh baz
touched baz
$
```

Exit statement and exit status

Recall: every command returns an exit status

0 for success; non-0 for error

```
$ ls foo
ls: cannot access 'foo': No such file or directory
$ echo $?
2                         # look at man page for ls for details
$ ls temp.txt
temp.txt
$ echo $?
0
```

? is a special bash variable; shows exit status of last cmd

- You can use the 'exit' statement to terminate a script and set the exit status.
- By default the exit status of a script is the exit status of last command in the script.

test statement

test expression

exit with status determined by the expression

[expression]   (an alternative way to write it)

```
$ x=foo
$ test $x = foo; echo $?
0
$ test $x = bar; echo $?
1
$ [ $x = baz ]; echo $?
1
$ i=10
$ [ $x = foo ] && [ $i > 5 ]; echo $?
0
```

expression failed, so exit status 1

If statement

```
$ i=2
$ echo $i
2
$ if [ $i = 2 ]; then echo $i; fi
2
$
$
$ x=5
$ if [ $x -lt 6 ]; then echo "less"; fi
less
$
```

string1 = string2
(string comparison)

also, !=, <, >

arg1 -lt arg2
(numeric comparison)

also, -eq, -ne, -le, -gt, -ge

File tests

```
$
$ if [ -f hw2.txt ]; then echo "okay"; else echo "missing"; fi
missing
$
$
$
$ if [ -d foo ]; then echo "foo is a dir“; fi
$
```

-f file
(true if file exists and is a regular file)

-d file
(true if file exists and is a directory)

Some other file-related conditional expressions:

`-e file`	`true if file exists`
`-r file`	`true if file exists and is readable`
`-x file`	`true if file exists and is writeable`

Combining tests

```
$ ls hw*.txt
hw1.txt  hw3.txt
$
$ if [ ! -f hw2.txt ]; then echo "hw2.txt does not exist"; fi
hw2.txt does not exist
$
$ if [ -f hw1.txt -a -f hw3.txt]; then echo "both exist"; fi
both exist
$
$ if [ -f hw1.txt -o -f hw2.txt ]; then echo "one or both exist"; fi
one or both exist
$
```

not

and

or

If statements: advanced

The brackets [and] are not really part of an if statement.

Structure of an if statement

```
        if cmds; then cmds; fi
```

Run the then-commands if the exit status of the if-commands is 0.

```
$ if ls hw1.txt; then wc -l hw1.txt; fi
hw1.txt
76 hw1.txt
$
$ if [ -f hw1.txt ]; then wc -l hw1.txt; fi      # alternative way
76 hw1.txt
```

More examples (optional)

```
# this works because -f hw2.txt is a conditional
# expression, and ! negates a conditional expression
$ if [ ! -f hw2.txt ]; then "hw2.txt does not exist"; fi
hw2.txt does not exist
$
# this works because [ -f bar.txt ] is a command, and
# ! negates the exit status of a command
$ if ! [ -f bar.txt ]; then "bar does not exist"; fi
$
# the test command can be used without 'if'
$ [ -f hw2.txt ] || echo "hw2.txt does not exist"
hw2.txt does not exist
$
# another fun example
$ [ -f hw3.txt ] && echo "hw3.txt does exist"
hw3.txt does exist
```

bash: sequence expansion

```
$ for i in {1..10}
> do
> echo $i
> done
1
2
…
8
9
10
$
$ for i in {4..8..2}; do echo $i; done
4
6
8
$
```

{x..y[..incr]}

Summary

We have learned how to:

- create and run simple bash scripts
- use bash 'for' and 'if' statements

Commands introduced in this lecture:

- `exit`

Bonus content: more on shebang

You can run a bash script like this:

```
  $ bash script.sh
```

The shebang line lets the script to be run as an executable:

```
  $ ./script.sh
```

This shebang line

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

is common, but

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

is more portable – it finds bash in your path.