Bash Scripting Part 2

OPS102 Week 8 Class 2

Chris Tyler/John Sellens

July 2, 2024

Seneca Polytechnic

Outline

Recap From Last Class

Recap From Last Class

- Scripts are handy, and easy to create – yay text files!
- The “shebang” line is a “magic number” that guides the kernel on how to execute a script.
- Variables are handy – local and environment.
- Quoting and backslash escaping hide special characters and whitespace.

The Read Command

Reading a Variable Value from Stdin: read

- You can read from standard input into a variable with the read command:
  `read` *`variable`*
- For example:

```
$ read course
Seneca OPS102
$ echo $course
Seneca OPS102
```

- If you give `read` multiple variables, it will tokenize the input.

```
$ read first last restofline
Chris Tyler likes using Linux
$ echo $restofline
likes using Linux
```

Using read with a Prompt String

- You can display a message to the user when reading from stdin by using the `-p` (prompt) option to read:

```
$ read -p "Please enter a course code: " ccode
Please enter a course code: OPS102
$ echo "The selected course is $ccode"
The selected course is OPS102
```

- Of course, you can also use a separate echo command instead!
  - Which has a `-n` option to suppress the trailing newline.

Demo: Variables in a Script

```
#!/bin/bash
read -p "Please enter your name: " name
echo "Pleased to meet you, $name"
read -p "Please enter a filename: " file
echo "Saving your name into the file..."
echo "yourname=$name" >>$file
echo "Done."
```

Command Capture (Substitution)

Command Capture (Substitution)

You can capture the standard output (stdout) of a command as a string using the notation `$(command)`

```
$ echo "The current date and time is: $(date)"
The current date and time is: Mon 19 Jun 2034 12:02:11 AM EDT
$ files="$(ls|wc -l)"
$ echo "There are $files files in the current directory $(pwd)"
There are 2938 files in the current directory /bin
```

It's also called "command substitution" since the output of the command is substituted for what was on the command line.

Command Capture: Avoid Backticks

You may see old scripts that use backticks (reverse single quotes) for command capture:

```
$ files=`ls`
```

Don't do this! This is an archaic syntax which is deprecated. Some fonts make it hard to distinguish between backticks and single quotes, and nesting backticks is difficult.

Unless you're writing code that needs to be portable to non-bash systems.

Shell Arithmetic

Arithmetic!

- Bash can do *integer* arithmetic
- To evaluate a arithmetic expression and return a value, use `$(( ))`
- To evaluate a arithmetic expression without returning a value, use `(( ))`
- Dollar-sign prefixes for variables are not required inside `$(( ))` or `(( ))`

```
$ a=100
$ b=12
```

```
$ echo $((a*b))
1200
$ echo $((b++))
12
$ echo $b
13
```

```
$ ((a++))
$ echo $a
101
$ ((c=a*b*2))
$ echo "The answer is $c"
The answer is 2626
```

Old Style Arithmetic

- The `expr` command evalutes expressions.
- Can be used in command substitution (or output redirection).
- Less convenient than bash arithmetic, but more portable.

Exit Status and Conditionals

Exit Status

- When a program runs, it exits with a numeric value. This goes by any of several names:
  - exit status, exit code, status code, error code
- Usually, an exit status of zero means that no errors were encountered, and a non-zero status means that something went wrong.
- Alternately, program authors can use this value as they see fit, so the exit status may indicate something else, like the number of data items processed.
- In C programs the exit status is from exit(): `exit( 3 );`
- A shell script exits with the exit command: `exit 0`
  - Integer argument optional, defaults to 0

Exit Status: $?

The special variable $? can be used to find out the exit status of the last command executed:

```
$ ls /foo/bar/baz
ls: cannot access '/foo/bar/baz': No such file or directory
$ echo $?
2

$ ls /usr/bin/bash
/usr/bin/bash
$ echo $?
0
```

Exit Status: Why do we care?

- In a script you may want to notice if a command fails.
- And the exit statuses of commands are the key to conditional logic (if statements) and looping (for/while/until) in bash scripts.
- A C program exits by calling `exit( 0 );`
- A shell script exits by running `exit 0`
  - Or falling off the end of the script

Conditional logic: if / then / elif / else / fi

The if command takes two or more lists of commands, and uses the result of one list to control the execution of the other.

```
if cmdlist1   # if the exit status is 0
then
  cmdlist2    # then run these commands
fi
```

Conditional logic: if / then / elif / else / fi

```
if grep –q "OPS102" testfile
then
    echo "The course is mentioned in the file"
fi
```

The shell runs the `grep` command, and if the string is found in the file, `grep` exits 0, which indicates “true”, and so the command(s) in the “then” of the “if” are run.

Conditional logic: if / then / elif / else / fi

There are else and elif (else-if) keywords too:

```
if cmdlist1     # If the exit status is success
then
  cmdlist2      # then run this
elif cmdlist3   # else if this exits with success
then
  cmdlist4      # then do this
else
  cmdlist5      # otherwise do this.
fi
```

Conditional logic: if / then / elif / else / fi

```
if grep –q "OPS102" testfile
then
    echo "The course is mentioned in the file"
else
    echo "The file does not mention OPS102"
fi
```

Conditional logic: if / then / elif / else / fi

```
if grep –q "OPS102" testfile
then
    echo "The course is mentioned in the file"
elif grep –q "ULI101" testfile
then
    echo "The old ULI101 course is in the file"
else
    echo "The file does not mention OPS102 or ULI101"
fi
```

More on if Statements

- The command lists can be one of more commands, separated by newlines or semi-colons.
- Common formatting style:

```
if grep –q "OPS102" testfile ; then
    echo "The course is mentioned in the file"
elif grep –q "ULI101" testfile ; then
    echo "The old ULI101 course is in the file"
else
    echo "The file does not mention OPS102 or ULI101"
fi
```

Summary

Summary

- Reading input
- Command capture / command substitution
- Integer arithmetic
- Exit status
- if statements
- Next class?
  - test, parameters, while, until, for