CSE 303 Concepts and Tools for Software Development

Richard C. Davis
UW CSE – 10/4/2006

Lecture 4 – Shell Scripting

Homework 1 Due Soon!

- 48 hours left
- Office hours announced over cse303@cs
  - Mailing list problems resolved (mostly)
  - Check archives if you missed or joined late
- Got someone to talk to?

Newbie? You're Not Alone

Subject	None	Beg.
Unix Commands	49%	31%
Shell Scripting	64%	29%
C	55%	22%
C++	47%	25%
Dev Tools (make, gdb, gprof)	69%	25%

And lots of interest in C++

Lecture 3 Errata

- C-s and C-g in shells
  - Works on Linux, but not on Windows?
- Combining Commands
  - Opposite of Java: 0 == true
  - `cmd1 || cmd2` (OR: stop if any succeed)
  - `cmd1 && cmd2` (AND: stop if any fail)
  - Fixed in slides
  - We'll touch on later

Tip of the Day

- Emacs Undo
  - Undo: C-_
  - Redo: <any char> and C-_ again
- Emacs shell-script-mode
  - M-x shell-script-mode (space completes)
  - Open a file with extension ".sh"

Last Time

- Command-line editing
- Input/Output
  - Rdirection
  - Pipes
  - Logic Expressions
- Variables
- Quoting and escaping

Today

- Shell scripting details
  - Variables and Arrays
  - Arithmetic Expressions
  - Logic Expressions
  - Control Structures
- Java vs. Bash programming

Review: Sripting Part I

- Put all commands in file "script"
- `source script`
- All commands executed!

- Problem
  - State of shell is changed

Review: A better approach

- Method
  - Start a new shell
  - Share stdin, stdout, stderr
  - exit
- Advantages
  - Shell script is like any other program
  - Shell commands are programming language
  - But it's a bad language for anything else!

Review: Scripting Part II

- Process
  - First line of script = `#!/bin/bash`
  - Get "execute" permission: `chmod u+x script`
  - Run it: `script` (or `./script` if `.` not in `$PATH`)
- Why this works
  - The shell consults the first line
  - If shell program is there, launch and run script
  - Else, it's a "real" executable, so run it

All variables global unless…

- You don't `export` and start a new shell
  - `printenv` : show vars visible to new shells
  - `set` : show all variables
- You declare a function AND declare local
  - We won't cover this

Command Line Variables

- Special Variables
  - `$0`: program name
  - `$1`: first argument (`$2` second argument …)
  - `$#`: number of arguments
  - `$@`: same as `$1 $2 …`
- What if you want to handle many args?
  - `shift`: shifts argument numbers down

Array Variables

- One dimensional, no fixed size
- Make an array: `foo=(x y x)`
- Set element: `foo[2]=hi`
- Get element: `${foo[2]}`
- Get number of elements: `${#foo[*]}`
- All elts. separated by spaces: `${foo[*]}`
- Examples: *arrays.sh*

Arithmetic

- All variable values held in *strings*
- If told, shell converts to numbers (or 0)
- Three ways to do arithmetic (integer)
  - Method 1: `` `expr $i + 1` ``
  - Method 2: `((i = i + 1))` or `$(($i+1))`
    - Spaces and `$` optional inside `(( ))`
  - Method 3: `let "i = i + 1"`
    - Quotes permit use of spaces
- Examples: *arithmetic.sh*

Logic Expressions

- Through return code of `test` *i.e.* `[ ]`
  - 0 means *true*
  - Non-0 means *false*
- Lots of things to test
  - File tests (exists, non-empty): `[ -s file ]`
  - String tests (=): `[ s1 = s2 ]`
  - Numeric tests (<): `[ n1 -lt n2 ]`
  - Combining tests (&): `[ test1 -a test2 ]`

Logic Grouping

- Two ways to group : (v1>v2 AND v1<v3)
  - Separate into different tests
    - `[ $1 -gt $2 ] && [ $1 -lt $3 ]`
  - Use `\(` and `\)`
    - `[ \( $1 -gt $2 \) -a \( $1 -lt $3 \) ]`
- Examples: *logic.sh*

Control Structures: if

```
if command
then
  body
elif command2
then
  body2
else
  body3
fi
```


Control Structures: for

```
for variable in list
do
  body
done
```

- *list* can be
  - A given set: `a b c 1 2 3`
  - Content of an array: `${foo[*]}`
  - File pattern: `*`
  - Result of a command: `` `cat numbers.txt` ``
- Examples: *loops.sh*

More Control Structures

- `case` statement
- `while` loop
- `until` loop
- `break` and `continue`

- See Linux Pocket Guide p 171-175

Shell Gotchas

Shell Gotchas

- Typo in variable set makes new variable
- Typo in variable use gives empty string
- Non-number used as number gives zero
- Omit array superscript, get first element
- Array out-of-bounds
  - Increases size or returns empty string
- Spaces, substitution order, braces?
  - Start a shell and experiment (or try manual)

Shell vs. Java Programming

- Shell
  - (+) shorter, convenient file-access, file-tests, program execution, pipes
  - (-) crazy quoting rules and ugly syntax
- Java
  - (+) cleaner, array checking, type checking, real data structures
  - (-) hard to do manipulate files, users, processes

Bottom Line

- Never do something manually if writing a script would save you time
- Never write a script if you need a large, robust piece of software
- Some languages straddle the middle
  - Perl, Python, Ruby

Summary

- Shell scripting details
  - Variables and Arrays
  - Arithmetic Expressions
  - Logic Expressions
  - Control Structures
- Java vs. Bash programming

Reading

- Linux Pocket Guide
  - P170-176 (Control Structures)
- Bash Reference Manual
  - 6.5 Arithmetic
  - 6.7 Arrays

Next Time

- Regular expressions
- Grep