CSE 303 Concepts and Tools for Software Development

Richard C. Davis
UW CSE – 10/2/2006

Lecture 3 – I/O and Scripting

Administravia

- Office hours after lecture
- Post Homework questions on Wiki
- Homework questions
  - Restarting in Problem 1
    - `man script` – find option to read from file!
  - How do I tell which commands I've used?
    - Pay attention in lecture today!

Emacs command of the day

- M-q = word wrap
  - Useful if you're writing text
  - Also see M-x auto-fill-mode

Last Time

- Shell as an interpreter
- Shell model (Files, Users, Processes)
- Shell Expansions/Substitutions
  - File names
  - Braces
  - History
  - Aliases

Today

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

Command Line Editing

- Shortcuts for navigating/editing commands
  - Extremely helpful when typing
    - Especially in terminals (e.g. when no arrow keys)
  - No help in scripts
- Bash similar to emacs
- Interesting cases
  - Up/Down arrows: access command history
  - Tab: Filename completion (space in emacs)
  - C-s: Stops displaying output until C-q

Input/Output

- Simple View
  - Input: command line arguments
  - Output: Return code (a.k.a. "exit status")
    - Convention: 0 = "success"
- More complete view
  - Input: stdin stream (0)
  - Output: stdout stream (1)
  - Error message output: stderr stream (2)

Streams in the Shell

- Default
  - stdin: read from keyboard
  - stdout, stderr: printed on command line
- Examples
  - `cat` (no args): sends stdin to stdout (`^d`)
  - `mail` : message body from stdin (`^d`)
  - `ls` : files to stdout and "No match" to stderr

File Redirection

- Tell shell to use files, not keyboard/screen
  - Redirect stdin: `cat < file`
  - Redirect stdout (overwrite): `history > file`
  - Redirect stdout (append): `cat >> file`
  - Redirect stderr: `man 2> file`
  - Redirect stdout and stderr: `ls &> file`
    - Short for `ls > file 2>&1`
- Yes, syntax is arcane; we're stuck with it

Pipes: Where the Fun Begins

- "cmd1 | cmd2"
  - cmd1 stdout goes to new stream
  - cmd2 stdin comes from this new stream
- Very powerful idea!
  - Create larger commands from smaller ones
    - Each command does one thing well

Pipe/Redirection Examples

- Silly Equivalences (like 1 * y = y)
  - `cat y = cat < y`
  - `x < y = cat y | x`
  - `x | cat = x`
- More useful examples
  - `ls –help | less`
  - `djpeg tur.jpg | pnmscale -xysize 100 150 | cjpeg >thumb.jpg`
  - `grep -e '^bash-3.1\$' problem1.txt | sort`

Using stdout on Command Line

- "Backquoting" can sent output as argument
  - Useless example: `cd `pwd``
  - Useful example: `mkdir `whoami``

Using the Return Code

- Commands can be combined
  - `cmd1 ; cmd2` (sequence)
  - `cmd1 || cmd2` (OR: stop if any succeed)
  - `cmd1 && cmd2` (AND: stop if any fail)
- Particularly useful in scripts
- Previous line's return code is in `$?`

Shell Variables

- Used to control shell state
  - `printenv`
- Assignment
  - `i=42` (no spaces!)
  - `Q="What is the answer?"`
- Retrieval uses `$`
  - `echo $PATH`, `echo ${PATH}`
  - `A="The answer is $i"`

Variables Details

- By default, other shells can't see variables
  - `bash` : lose variables (see scripts)
  - `export var`: to keep them
- `unset i`: note, no `$`!
- Gotcha: Trying to used undefined variable
  - No error!
  - Empty String
- Variables aren't aliases

Quoting and Escaping

- Lots of special characters
  - `` ` ' " ! % & * ~ ? [ ] { } \ < > | $ ``
- Getting these chacters into input
  - `"` : Does *some* substitutions (`"12 * ${i}"`)
  - `'` : Does *no* substitutions (`'$100'`)
  - `\` : Don't substitute next char (`"$N has \$5"`)

Moving Toward Scripts

- Shell has state
  - Working directory
  - User
  - Collection of aliases
  - History
  - Variables
- We can write programs!

Sripting Part I

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

- Problem
  - State of shell is changed

A better approach to scripting

- 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!

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

Command Line Arguments

- Special Variables
  - `$0` program name
  - `$1` first argument …
- What if you want to handle many args?
  - `shift`: shifts argument numbers down

Summary

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

Reading

- Linux Pocket Guide
  - p166-170, 176-179 (Programming with Shell Scripts)
- Bash Reference Manual
  - 4.1 (shift command)

Next Time

- Advanced Scripting
  - Control structures
  - Arrays