

---

CSE 374

# Programming Concepts & Tools

Brandon Myers

Winter 2015

Lecture 2 – Processes, Programs, the Shell (& emacs / vim)

(thanks to Hal Perkins)

---

# Where we are

---

- It's like we started over using the computer from scratch
- All we can do is run dinky programs at the command-line
- But we are learning a model (a system is files, processes, and users) and a powerful way to control it (the shell)
- If we get the model right, hopefully we can learn lots of details quickly
- Today:
  - The rest of the model briefly: Processes and Users
  - More programs (ps, chmod, kill, . . . )
  - Special shell characters (\*, ~, . . . )
  - Text editing (particularly emacs / vi)

# Users

---

- There is one file-system, one operating system, one or more CPUs, and multiple users
- whoami
- ls -l and chmod (permissions), quota (limits)
  - Make your homework unreadable by others!
- /etc/passwd (or equivalent) guides the login program:
  - Correct username and password
  - Home directory
  - Which shell to open (pass it the home directory)
  - The shell then takes over, with startup scripts (e.g., .bash\_profile, .bashrc). (ls -a)
- One “superuser” a.k.a. **root**. (Change passwords, halt machine, change system directories, add/remove user accounts, . . . )

# Programs & the Shell

---

- A program is a file that can be executed
- Almost all system commands are programs
- The shell is itself a program
  - Reads lines you type in & carries them out
  - Normally finds the named program and runs it
    - A few commands are shell “built-ins” that the shell executes itself because they change the state of the shell. Obvious example: `cd`
  - After the named program runs it exits and the shell reads the next command
  - More to this story to come...

# Processes

---

- A running program is called a process. An application (e.g., emacs), may be running as 0, 1, or 57 processes at any time
- The shell runs a program by “launching a process” waiting for it to finish, and giving you your prompt back.
  - What you want for ls, but not for emacs.
  - &, jobs, fg, bg, kill — job control
  - ps, top
- Each process has private memory and I/O streams
- A running shell is just a process that kills itself when interpreting the exit command
- (Apologies for aggressive vocabulary, but we’re stuck with it for now.)

# Standard I/O streams

---

- Every process has 3 standard streams: stdin (input), stdout (output), stderr (error messages)
- Default is keyboard (stdin), terminal window (stdout, stderr)
- Default behavior is to read from stdin, write normal output to stdout, write diagnostic output to stderr
  - Many programs accept command-line arguments naming files to read
  - If not supplied, just read stdin
  - Also ways to redirect stdin, stdout, stderr. Later...

# That's most of a running system

---

- File-system, users, processes
- The operating system manages these
- Processes can do I/O, change files, launch other processes.
- Other things: Input/Output devices (monitor, keyboard, network)
- GUIs don't change any of this, but they do hide it a bit
- Now: Back to the shell. . .

# The shell so far

---

- So far, our view of the shell is the barest minimum:
  - builtins affect subsequent interpretations
  - New builtin: source
  - Otherwise, the first “word” is a program run with the other “words” passed as arguments
    - Programs interpret arguments arbitrarily, but conventions exist



# Complicating the shell

---

- But you want (and bash has) so much more:
  - Filename metacharacters
  - Pipes and Redirections (redirecting I/O from and to files)
  - Command-line editing and history access
  - Shell and environment variables
  - Programming constructs (ifs, loops, arrays, expressions, ...)
- All together, a very powerful feature set, but awfully inelegant

# Filename metacharacters - globbing

---

- Much happens to a command-line to turn it into a “call program with arguments” (or “invoke builtin”)
- Certain characters can expand into (potentially) multiple filenames:
  - ~foo – home directory of user foo
  - ~ – current user’s home directory (same as ~\$user or ‘whoami’).
  - \* (by itself) – all files in current directory
  - \* – match 0 or more filename characters
  - ? – match 1 filename character
  - [abc], [a-E], [^a], . . .more matching
- Remember, this is done by the shell before the program sees the resulting arguments

# Filename metacharacters: why

---

- Manually, you use them all the time to save typing.
- In scripts, you use them for flexibility. Example: You do not know what files will be in a directory, but you can still do: `cat *` (though a better script would skip directories)
- But what if it's not what you want? Use quoting ("`*`" or '`*`') or escaping (`\*`)
- The rules on what needs escaping where are very arcane
- **A way to experiment: echo**
  - `echo args. . .` copies its arguments to standard output after expanding metacharacters

# History

---

- The history builtin
- The ! special character
  - !!, !n, !abc, . . .
  - Can add, substitute, etc.
- This is really for fast manual use; not so useful in scripts

# Aliases

---

- Idea: Define a new command that expands to something else (not a full script)
- Shell builtin command:

```
alias repeat=echo
alias dir=ls
alias hello="echo hello"
alias rm="rm -i"           % for cautious users
alias                    % list existing aliases
```
- Often put in a file read by source or in a startup file read automatically
- Example: your `.bashrc` – feel free to change

# Bash startup files

---

- Bash reads (sources) specific files when it starts up. Put commands here that you want to execute every time you run bash
- Which file gets read depends on whether bash is starting as a “login shell” or not
  - Login shell: `~/.bash_profile` (or others – see bash documentation)
  - Non-login shell: `~/.bashrc` (or others if not found)
- Suggestion: Include the following in your `.bash_profile` file so the commands in `.bashrc` will execute regardless of how the shell starts up

```
if [ -f ~/.bashrc ]; then source ~/.bashrc; fi
```

# Where we are

---

Features of the bash “language”:

1. builtins
2. program execution
3. filename expansion (Pocket Guide 23-25, 1<sup>st</sup> ed 22-23)
4. history & aliases
5. command-line editing
6. shell and environment variables
7. programming constructs



But file editing is too useful to put off. . . so a detour to emacs (which shares some editing commands with bash) / vim

# What is emacs / vim?

---

- Programmable, extensible text editors, with lots of goodies for programmers
- Not full-blown IDEs but very powerful!
- Top commands:

Description	Emacs	vim
Stop current command	C-g	Esc / C-c
Open file in buffer	C-x C-f	:e NAME
Save buffer	C-x C-s	:w
Switch buffer	C-x b	:b NUM
Delete line	C-S-Bksp	dd
Selection/deletion	C-y ... C-w	v ... d
Cursor down/up	C-n / C-p	j / k
exit	C-x C-c	:q



# Getting started with emacs / vim

---

- Take the emacs / vim tutorial to get the hang of the basics. Both have a learning curve but will make your text editing fast!
- Everyone should know this at least a little – emacs and vim editing shortcuts are common in other Linux programs
- Customize, starting with your `.emacs` / `.vimrc` file
- Lots of plugins out there
- The best IDEs have good emacs and vim mode plugins
- Which should I pick? It's a war, just try both and pick one to become proficient in

# Command-line editing

---

- Lots of control-characters for moving around and editing the command-line. (Pocket Guide page 30 (2<sup>nd</sup> ed) or 28, emacs-help, and Bash reference manual Sec. 8.4.)
- They make no sense in scripts
- Good news: many of the control characters have the same meaning in emacs (and bash has a vim “mode” too)

# Summary

---

As promised, we are flying through this stuff!

- Your computing environment has files, processes, users, a shell, and programs (including emacs / vim)
- Lots of small programs for files, permissions, manuals, etc.
- The shell has strange rules for interpreting command-lines. So far:
  - Filename expansion
  - History expansion
- The shell has lots of ways to customize/automate. So far:
  - alias and source
  - run (i.e., automatically source) `.bash_profile` or `.bashrc` when shell starts

Next: I/O Redirection & stream details, Shell Programming