CSE 374
Programming Concepts & Tools

Hal Perkins

Winter 2017

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

News

- HW1 out now, due Thursday night, 11 pm
  - Linux commands, files, editing

- Please use discussion board, or mail to cse374-staff when email is needed; not mail to individual TAs/instructor
  - Remember to post a followup message on the discussion board if you haven't already

- If you're still trying to add the course and haven't filled out the online form, please do that by 1 pm today
  - Details on a separate slide at the end of the hour…
  - We'll try to have decisions later today or tomorrow
    - (have more people trying to add than we can take, so we'll do the best we can use the seats for people who need the course for graduation this year, etc....)

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)

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 (originally) password – actual encrypted password is elsewhere now for security
  - 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, 1st 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)

What is emacs?

- A programmable, extensible text editor, with lots of goodies for programmers
- Not a full-blown IDE but much “heavier weight” than vi
- Top-6 commands:

  C-g

  C-x C-f

  C-x C-s, C-x C-w

  C-x C-c

  C-x b

  C-k, C-w, C-y, . . .
- Take the emacs tutorial to get the hang of the basics
- Everyone should know this at least a little – emacs editing shortcuts are common in other Linux programs
- Customizable with elisp (starting with your .emacs)

Command-line editing

- Lots of control-characters for moving around and editing the command-line. (Pocket Guide page 28, emacs-help, and Bash reference manual Sec. 8.4.)
- They make no sense in scripts
- Gotcha: C-s is a strange one (stops displaying output until C-q, but input does get executed)
- Good news: many of the control characters have the same meaning in emacs (and bash has a vi “mode” too)

vi(m) vs emacs

- You need to learn one of these
  - Yes, there are many other editors out there, but you need to learn one of these – they are standard and available on all Unix/Linux systems (& emacs, at least, has versions for Windows, OS X)

- Emacs keyboard shortcuts work in many other programs and situations like bash
  - Including vi(m) if you set the right mode!
  - Learn them – it will make you more efficient

- We won't try to dictate or settle the vi vs emacs wars
  - You're on your own for that!

Summary

As promised, we are flying through this stuff!

- Your computing environment has files, processes, users, a shell, and programs (including emacs)
- 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