CSE 374: Programming Concepts and Tools

Eric Mullen

Spring 2017

Lecture 2: Globbing and Processes

Administrivia

- Overloading the class: Details on a slide at end. Many more requests than slots, don't have high hopes
- HW0 is out and due Friday at midnight
  - I opened the discussion board to anyone with a UW ID
- Office hours posted on course website
  - 4pm MTWF, 1:15pm on Thur with TAs (218)
  - 11am on Tues with me (218)
- Friday lecture is guest lecture. Will be awesome!

Laptops

- Only screens flat on desk are allowed.

- Put away your laptops.

Where We Are

- We're learning from scratch to use a computer
  - All we have are little tiny programs (so far)
- Learning a model (files, processes, users) and how to control (shell)
- Once you understand the model, it's powerful
- Today:
  - Processes and Users
  - Globbing
  - Text Editing

Users

- You, and others. Linux is built for multiple users
- Use `whoami` to show your username
- Each user has username and password, originally stored in `/etc/passwd`
- Home directory, default shell. On login shell runs startup scripts which you can edit (.bash_profile, .bashrc).
- There is one super user, **root**. Has permission to do everything.

Hidden Files

- I just told you that .bash_profile and .bashrc are run every time you log in.
- Turns out `ls` doesn't display filenames that begin with a .
- If you want to see them, use `ls -a`
- If you want more details about the files, use `ls -l`
- `ls` has lots more options, read about them

Programs

- A program is a file that can be executed
- Almost all system commands are programs
- The shell itself is a program
    1. Reads lines as you type them
    2. Finds whatever program you want, runs it
    3. Upon exit of that program, go back to 1

Processes

- A process is what's created when a program is run
- It is the running "thing"
- The shell runs a program by launching a process, waiting for it to finish, and then gives you your prompt
- Each process has own memory and I/O streams
- A running shell is just a process that kills itself when you type `exit`

Processes

- One application can be many processes
- You can interact with running processes on your machine
  - `<command> &` to run in the background
  - Ctl-z to suspend current process
  - `fg` to resume in foreground, `bg` to resume in background
  - `ps` to list processes, `top` more like a task manager
  - `kill` to kill a process, Ctl-c to kill current process

Standard I/O Streams

- Each process has 3 standard streams: stdin (input), stdout (output), and stderr (error messages)
- The *default* behavior in the shell is the keyboard hooked to stdin, and both stdout and stderr hooked to print to the screen

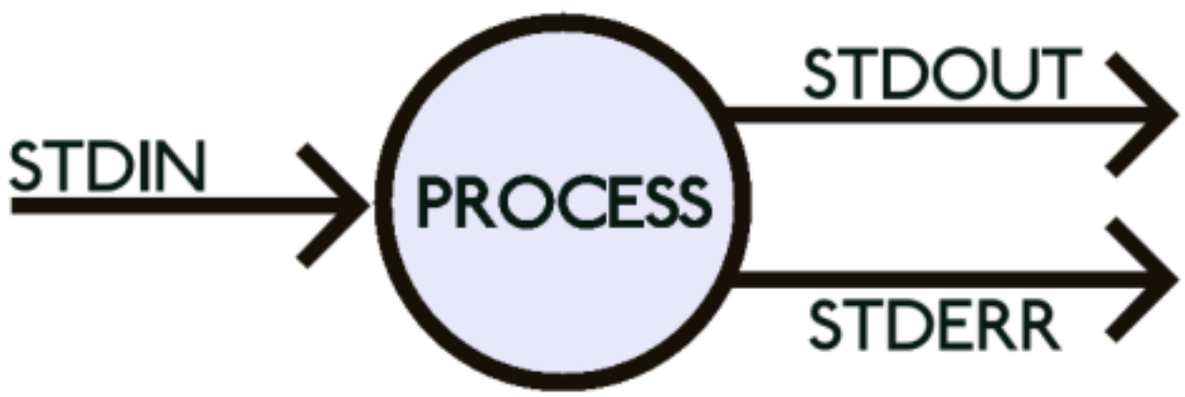

image from http://www.randomnoun.com/wp/2013/10/08/its-all-made-out-of-pipes/

Entire System Recap

- The operating system manages everything
- We have a file system, users, processes
- Processes can perform I/O, change files, launch other processes

How Does Bash Know?

- When you type `ls`, bash is finding and running the `ls` program

- Uses the `$PATH` environment variable to know where to look

- More on environment variables later…

Shell Scripts

- A shell script is just a file that contains shell commands

- Sometimes we give them the file extension .sh, but that's only for human benefit (computer doesn't care)

- `$1` means first argument, `$2` for second, etc…

Running Shell Scripts

- `./script.sh` makes new process
    - If script is not in current directory, use the path to the script instead. (i.e. `/usr/bin/script.sh`)
- `source script.sh` runs in same process

Globbing

- Bash is even more magical. It transforms arguments before it gives them to programs.
- `~foo` means the home directory for user `foo`
- `~` is your home directory
- `*` is all the files in the directory
- `*.txt` is all the files that end in `.txt`
- There's lots more: `?`, `[abc]`, `[a-E]`, `[^a]`, etc..
- Sounds great now, works badly with `grep` (we'll see in a few weeks)

Globbing

- What if I want to pass a * as an argument?
- Put it in either single or double quotes, or escape it (with a backslash)
  - “*”, ‘*’, or *

History

- `history` prints out the previous commands entered
- `!!`, `!abc` expand to previous commands
- Good for manual use, not so much in scripts

Alias

- Define one command to be another
  - `alias best editor=emacs`
  - `alias list_all='ls -a'`
  - `alias` lists existing aliases
  - Careful: you can't put spaces around the =

Bash Startup Files

- ~/.bash_profile on login shell
- ~/.bashrc on non-login shell
- My ~/.bash_profile includes (yours probably does too):

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

Editing Files

- You have options
- `pico` is easy, displays commands on screen
- `emacs` is what I know (and will teach you)
- `vi/vim` is also good

Emacs

- How to read an emacs command
- C-s means “Hold down `ctrl`, then press s”
- C-x C-c means “Hold down `ctrl`, press x, then press c while still holding `ctrl`”
- C-x o means “Hold down `ctrl`, press x, then release `ctrl`, and after press o”
- M-x means “Hold the `meta` key, then press x”. Usually `meta` is the `alt` key

Emacs

- C-x C-c : quit
- C-x C-f : find (open) a file
- C-x C-s : save currently open file
- C-n : next line
- C-p : previous line
- C-f : forwards
- C-b : backwards

Emacs

- C-a : beginning of line
- C-e : end of line
- C-s : find string
- This is just scratching the surface, emacs is huge and ridiculously complicated
- You should be able to complete this course with just the commands on the previous page

Wrap Up

- OS, Filesystem, Users, Processes, Shell all make up our linux system
- There are a million little tips and tricks, focus on the core
- Text editors are useful, learning one will help you

Homework 0

- Don’t forget to do it by Friday at midnight!
- I opened the discussion board for anyone with a UW login
- We’re enjoying the introduction emails