CSE 374
Programming Concepts & Tools

Hal Perkins

Spring 2010

Lecture 1 – Course Introduction

Welcome!

- We have 10 weeks to move to a level well above novice programmer:
  - Command-line tools/scripts to automate tasks
  - C programming (lower level than Java; higher than assembly)
  - Tools for programming
  - Basic software-engineering concepts
  - Basics of concurrency
- That's a lot!
- Get used to exposure, not exhaustive investigation
  - This is not intro programming anymore

Today

- In class today
  - Course mechanics
  - Overview and plan
  - Dive into the *command shell*
- By next time
  - Get going on homework 1
    - Due Thursday!!
    - Check right away to be sure you can log on to a Linux machine and have your shell set right

Who and What

- 3 classes/week (slides, code, demos, questions)
  - Material online (often after class), but take notes!
  - Advice: jot down keywords, ideas; look up details later
  - Advice: use class for concepts (you can do *this*); use documentation for details (*how*)
  - Advice: experiment, try things later that day
  - Warning: the slides are **not** *nearly enough* to learn everything you need. They are an outline, tour guide, orientation only.

Requirements

- 7 homeworks (+ / - 1) (50%)
  - 3 shell commands and scripting (first very short)
  - 3 C programming
    - Later two of these involve tools extensively; one is a team project (work in pairs)
  - 1 C++ programming
- 1 midterm (20%), 1 final (25%)
- Last 5% is citizenship, participation, etc.
- Collaboration: individual work unless announced otherwise; *never* look at or show your code to others
- Extra credit: when available, small effect on your grade if you do it – no effect if you don't

Deadlines

- Turn things in on time!
- But things happen, so …
  - You have 4 late days to use this quarter
  - No more than 2 late days per assignment
  - Counted in 24 hour chunks (5 min = 23 hours late)
  - On group projects, can only use if both partners have late days and both partners are charged
- That's it. No other extensions (but contact instructor if you are hospitalized)
- Advice: Save late days for the end of quarter when you (might) really need them

Academic Integrity

- Policy on the course web. **Read it!**
- Do your own work – always explain any unconventional action on your part
- I trust you completely
- I have no sympathy for trust violations – nor should you
- Honest work is the most important feature of a university. It shows respect for your colleagues *and yourself.*

What to Expect

- Assignments may be less structured than you're used to
  - "Write a program that does this"
  - Learning how to deal with this is part of the plan
- Learning how to learn things is part of the plan
  - Learn your way around man pages, books, online documents (Google *is* your friend)
  - But *don't* just cut-n-paste code to "get it to work"
    - You *must* understand why your code does what it does, and be able to explain it!
- Tinker – try things. Write toy programs
  - The course will be *much* harder if you only do the assigned work
  - *Don't* avoid learning new tools

Linux Cycles

- We’re pretty agnostic about what computer you use
  - Needs to run a relatively recent version of the GNU toolchain (bash, utilities, gcc, other tools)
  - Default is dante.u.washington.edu, which is a Redhat Linux machine; all student have accounts there
  - EE department Linux machines are great (if you are in EE)
    - Either local or remote access
  - Cygwin runs GNU tools on Windows
  - Mac OS X includes what you need in developer tools
  - Can always install single- or dual-boot Linux on your own machines (Ubuntu, Fedora, etc., all ok)
  - It is also possible to run Linux in a virtual machine (vmware or Sun virtualbox)
- More details on course web; contribute ideas or questions if you don’t see what you need there

So What is CSE 374?

- New course last year
  - Descendent of CSE 303
  - Most material taken from there (particularly slides by Dan Grossman, but whatever else I can steal)
- Something of a "laundry list of everything else", but…

  There is an amorphous set of things computer scientists know about and novice programmers don't. Knowing them empowers you in computing, lessens the "friction" of learning in other classes, and makes you a mature programmer.
- The goal is to give you a sense of what's out there and what you can expect – and how you can learn more later when you need to
- "For engineers" – but pretty generic actually

5 General Areas

1. The command line
   - Text-based manipulation of the computing environment
   - Automating (scripting) this manipulation
   - Using powerful *utility* programs

- Let the computer do what it's good at so you don't have to!
- We'll use Linux (an operating system) and bash (a *shell*) – but the concepts are not tied to these
- Idea: Knowing the name of what "ought to exist"
- Idea: Programming in a language designed for interaction

5 General Areas

2. C (and a little C++)
   - "The" programming language for operating systems, networking, embedded devices, …
   - Manual resource management
   - Trust the programmer: a "correct" C implementation will run a program with an array-bounds error and whatever happens, happens
   - A "lower level" view of programming: all code and data sits together in a "big array of bits"

- Idea: Parts look like Java – don't let that deceive you!
- Idea: Learn to think before you write, and test often

5 General Areas

3. Programming tools – so far you have written programs and run them. There are programs for programming you should know about:
   - Compilers (vs interpreters)
   - Debuggers
   - Linkers
   - Recompilation managers
   - Version-control systems
   - Profilers
   - …

5 General Areas

4. Software development concepts – what do you need to know to write a million lines of code*?
   - Testing strategies
   - Team-programming concepts
   - Software specifications and their limits
   - …

* No, you will not write a million lines of code for CSE 374 this quarter, although it may seem like it at times…

5 General Areas

5. Basics of concurrency – what happens when more than one thing can happen at once in a program?
   - Brand-new kinds of bugs (e.g., races)
   - Approaches to synchronization
   - And it matters – most computers you can buy have (at least) 2 processors; that will be 8+ in a year or two
     - How do we run enough stuff concurrently to keep all the processors busy?

Perspective

"There is more to programming than Java methods"

"There is more to software development than programming"

"There is more to computer science than software development"

So let's get started. . .

The O/S, the Filesystem, the Shell

- Some things you might have a sense of but never were told precisely (may as well start at the beginning). . .
- The file-system is a tree
  - (Actually it's a dag)
  - The top is /
  - Interior nodes are directories (displayed as folders in GUIs)
- Users log-in, which for Linux means getting a shell
  - They have permissions to access certain files/directories
  - They have a "home directory" somewhere in the file-system
  - They can run programs. A running program is a process. (Actually could be more than one.)

File Manipulation

- You may be used to manipulating files via a GUI using WIMP
- You can do all the same things by running programs in the shell
- Just like an “explorer window”, the shell has a *current working directory*
- It really helps to remember the names of key commands: ls, cp, mv, rm, cat, cd, pwd.
  - (Most are really just programs.)
- Current directory: .
- Parent directory: ..
- Relative vs. absolute pathnames

What?

- Why would anyone want to interact like this?
  - Old people who remember life before GUIs :-)
  - Power users who can go faster
  - Users who want easy logging
  - Users who want easy instructions
  - Users who want programmability
- The last one will be the core of the first assignments
- Most computer scientists use GUIs and shells, depending what they're doing.
- Linux has GUIs and Windows has shells

Options, man (and info)

- Bad news for new users:
  - Program names and options are short, arcane, and numerous
- Good news
  - Most programs print a *usage* argument if given bad options (or often implement -help or --help)
  - The command man *what* prints a file describing program *what*
    - Similar: info *what* for complex programs (bash, gcc, some others
  - Tons of other resources (e.g., the web)
  - Things are somewhat standardized (dashes for options followed by argument as needed)

A Few More Programs and Options

- less (is more)
  - used by man
  - spacebar, b, /search-exp, q
- chmod
- mail

And some that aren't technically programs (more on this later)

- exit
- echo
- (cd)

Work to do!

- Get started on homework 1 ***now!***
  - Due Thursday
  - This means figure out what machine you plan to use and check that your account is working *today* (well, ok, maybe tomorrow – but **not Thursday!**)