CSE 374
Programming Concepts & Tools

Hal Perkins

Fall 2015

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 0
    - Due Friday night!!
    - Goal: Get a login shell on your Linux machine (probably CSE VM or klaatu) so you're ready to go!
  - Start reading Linux Pocket Guide (pp. 1-36 [1-33 1st ed] for sure; skim other parts for interesting things)

Who

- Staff
  - Hal Perkins, instructor
  - Irene Liu
  - Kuikui Liu
  - Josh Nazarian
  - Soumya Vasisht
  - Jack Zhang
- Office hours posted now. Use them to get "unstuck" so you can make progress on your own.
- You!
  - Over 70 people(!)
  - Who has used Linux before?
  - Who has written a C program before?
    - Beyond hello world?

If you're trying to add the course…

- Watch for positions to open up in the next couple of days as people adjust their schedules

- Fill out the online "I went to the first lecture and want to add the class" form
  - Instructions at the end of class

What

- 3 classes/week (slides, code, demos, questions)
  - Material online (often after class), but **TAKE NOTES!**
  - Advice: jot down keywords and 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) (55%)
  - 3 shell commands and scripting
  - 3 C programming
    - Later two of these use tools extensively
    - One is a team project (work in pairs)
  - 1 C++ programming
- 1 midterm (15%), 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

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.*

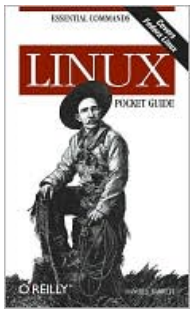
Deadlines

- 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 (10 min = 24 hours)
  - 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

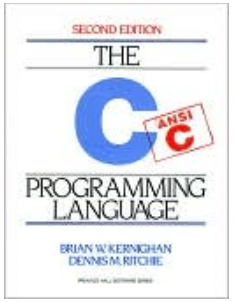
What to Expect

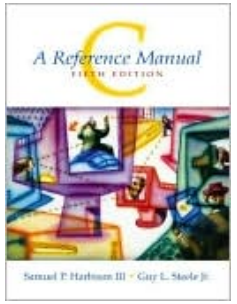
- Assignments may be less structured than you're used to
  - "Write a program that does this"
    - You need to figure out if you're getting the right output
    - Usually no "sample solution" to compare with
  - Learning how to deal with this is part of the plan
- Learning how to learn things is part of the plan
  - Learn your way around man pages, books, online documents (Google *is* your friend – but only one of them)
  - 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 is *much* harder if you only do the assigned work
  - *Don't* avoid learning new tools

Resources – Books



- Linux Pocket Guide: Enough Linux for CSE 374 and well beyond (you should have this). Either edition
- *C Programming Language* (K&R) – The classic Good for C & programming philosophy, examples + concise language & library reference (optional)
- *C: A Reference Manual* (Harbison & Steele) – Not listed as a text for CSE 374. More modern than K&R; best source for authoritative details about current C language/libraries.
- Others: O'Reilly publishes good books on many specific topics. Free online access via UW library (Safari Books Online; includes O'Reilly) Safari includes other good tech publishers/books.





Books? In the 21st Century??

- Why not just use Google, Stack Overflow, Reddit, Quora, …?
- Web-search good for:
  - Quick reference (What is the name of the function that does …? What are its parameters?)
  - Links to a good reference
- (can be) Bad for:
  - Why does it work this way?
  - What is the intended use?
  - How does my issue fit into the bigger picture?
- Beware:
  - Random code blobs cut-and-paste into your code (why does it work? what does it do?)
  - This inscrutable incantation solved my problem on an unstated version for no known reason

So What is CSE 374?

- 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

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
     - 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.)

Linux Cycles

- We're somewhat agnostic about what you use
  - We provide a standard CSE Fedora Linux – two flavors: remote login and virtual machine (next)
- Other environments are possible
  - Needs to be a fairly recent Linux distribution with standard tools, bash shell, gcc, utilities (Ubuntu, Fedora, others…)
  - Mac OS X developer tools has what you need
    - But not quite: Apple has their own C/C++
- We use CSE Fedora to test your code, so you should verify your code works there

Free CSE Linux (virtual) Machines!

- CSE Linux virtual machine
  - 64-bit Fedora 22 with CSE configuration
  - Runs on Mac, Windows, even other Linux(!)
  - Need VMware Player (free, Windows or Linux host) or VMware Fusion (free to registered students – you have gotten email this morning if registered)
    - Some people have used Virtualbox successfully
  - Download ~4-6 GB
  - Run ‘sudo dnf upgrade’ after initial setup to grab any recent updates/security patches
  - CSE 374 web has links to details

UW CSE Linux Virtual Machine

- Startup
- Configure user account name and user/superuser passwords
- Shell window

Demo

Alternative - klaatu

- Everyone in the class has an account on klaatu.cs.washington.edu
  - Userid is your UW netid
  - Password was mailed to your @uw email address earlier today if you're registered
- Same Fedora Linux as the virtual machine
- More details and suggestions on the CSE 374 web

demo

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

Start reading and trying things in the *Linux Pocket Guide*

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 (scripting!)
- 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 Linux 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*
    - Also: info *what* for complex programs (bash, gcc, some others
  - Tons of other resources (e.g., Linux Pocket Guide; the web – google is your friend)
  - 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 0 ***now!*** – due Friday
  - This means get your Linux setup working ***today*** (well, ok, maybe tomorrow – but **not later!**)
    - klaatu account info mailed this afternoon
  - Includes follow up on discussion board – join in!
- Start reading/trying the Pocket Guide
  - pp. 1-32, then skim through later sections
  - Don't memorize stuff – try it!
  - Get an idea of what's possible and where to look
- If you're trying to register, follow instructions here…
  - And you can turn in hw0 even before registering…