

CSCI 553: Networking III

Unix Network Programming

Spring 2008



Introduction to Software Carpentry
and
Unix software development Philosophy



Description & Objectives

- Develop better software development skills
- Learn the (basics) of the Unix development environment
- Apply tools, techniques & skills to development of simple networking applications.



Description & Objectives

- In the first half (appx.) of course we will introduce to standard programming and software development tools of UNIX environments.
 - The UNIX shell, shell scripts
 - C compiler, gdb, linker, object files, dynamic/static library linking
 - Python scripting language
 - Automated Builds w/ Make, revision control systems w/ Subversion
 - Some basic UNIX administration concepts



Description & Objectives

- In the second half of the course we will apply these tools to developing simple client/server applications using standard UNIX network programming tools and protocols.
 - signals, polling & I/O multiplexing
 - basic UNIX inter-process communication
 - TCP & UDP sockets
 - Network administration & information tools



Introduction

- The first half of his course will teach you how to design, build, maintain, and share programs efficiently
- Focus on the equivalent of good laboratory technique
 - The 20% of ideas that account for 80% of real world use
 - Software *carpentry*, rather than software *engineering*
 - About putting an extension on the house, rather than building the Channel Tunnel
- Everything that will make you more productive will also improve the quality of what you build
- Help computational science deserve the second half of its name



Self Assessment

1 for “yes”, 0 for “no” or “not relevant”, and -1 if you don't understand the question

- Do you use version control?
- Can you rebuild everything with one command?
- Do you build the software from scratch daily?
- Do you have an automated test suite?
- Do you run the suite before checking in changes?
- Do you know how much code your tests cover?
- Do you have a bug database?
- Do you use a symbolic debugger?
- Do you use assertions and other defensive programming techniques?
- Do you document as you program?
- Do you keep your documentation in your source files?
- Do you use a style checker to ensure that your software is written in a uniform, readable way?



Self Assessment

Your Score

- Negative: you will find this course challenging, but rewarding. You will need to put in some extra time up front to get a good start.
- 0-3: this course will fill in the gaps in what you already know
- 4-8: you will be able to apply the ideas in this course to your own projects immediately
- 9 and up: you will find at least the first half of this course very easy. Come to me I have GA positions available for you. You might want to look at the primary source material of the SWC site found in the Bibliography.



The State of Play

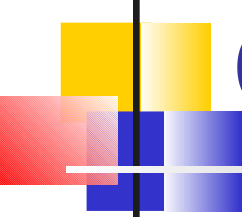
- Computers are as important to scientists as telescopes and test tubes
 - Simulate things that are too big, too small, too fast, too slow, or too dangerous to study in the lab
 - Analyze volumes of data that would overwhelm human “computers”
- Many scientists now spend much of their time writing and maintaining software
- But most have never been taught how to do this efficiently
 - It's a long way from the loops and arrays of first year to simulating bone development in foetal marsupials
 - Like being shown how to differentiate polynomials, then expected to invent the rest of calculus
- As a result, scientists spend far more time programming than they ought to
 - And they still don't know how trustworthy their programs are



Meeting Standards

- Experimental results are only publishable if they are believed to be *correct and reproducible*
 - Equipment calibrated, samples uncontaminated, relevant steps recorded, etc.
 - Same goes for business software development
- In practice, rely on expectations and cultural norms
 - Drilled into people starting with their first high school chemistry class
 - Only actually check work that is already under suspicion
- How well do computational scientists meet these standards?
 - Correctness of code rarely questioned
 - We all know programs are buggy...
 - ...but when was the last time you saw a paper rejected because of concerns over software quality?
 - Reproducibility often nonexistent
 - How many people can reproduce, much less trace, each computational result in their thesis?

The Grass Isn't That Much Greener

- 
-
- The bad news is that things aren't that much better in industry
 - Commercial projects of all sizes routinely go over time and over budget
 - What they deliver is often incomplete, riddled with bugs, and not what the customer actually wanted
 - How is this possible?
 - Low expectations
 - Like American cars in the 1970s
 - Lack of accountability
 - Hard to sue software developers
 - Most shrink-wrap licenses effectively say, "This CD could be blank, and we wouldn't have to give you back your money."



Hidden in Plain Sight

- The good news is that we've had solutions for these problems for years
 - They just aren't evenly distributed
 - This is one of the reasons good programmers are up to 28 times better than bad ones [[Glass 2002](#)]
- *Improving quality improves productivity*
 - The more effort you put into making sure it's right the first time, the less total time it'll take to get it built
- *The tools and techniques that help you write better code also help you write more code, faster*
 - Version control
 - Test-driven development
 - Task automation
 - Symbolic debuggers
 - And more that we'll meet later

The Times They Are A- Changing'

- The current situation is clearly unstable
 - The only direction standards and expectations can go is up
 - One high-profile lawsuit could set a precedent for the whole industry
 - Change can happen almost overnight
 - Like the American car market when German and Japanese imports appeared in the 1970s
- Offshoring is currently the biggest pressure
 - Jobs that *can* move overseas *will*
 - As in manufacturing, only work that adds significant value will remain
 - Which means that programmers in affluent societies either move up, or move out
- This course is just as relevant if you're in Mexico, India, China, or Hungary
 - "Ship, then fix" doesn't work if you're eleven time zones away from the customer
 - Your chances of getting a second project are much better if you deliver the first one on spec and on time



This Course

- Introduce some basic tools
 - Instant gratification
 - Serve as a guide to good practice
 - Every software tool implicitly embodies someone's ideas about how a task should be done
- Show how to build tools like these
 - What goes into the software
 - How to create it
 - Don't expect you to write your own version control system or bug tracker
 - But *do* expect you to automate common tasks, make your products extensible, etc.
- Show how to apply these ideas to other tasks
 - Most modern well-engineered applications are implemented as specialized sets of tools
 - (Where “well-engineered” means flexible, reliable, and maintainable)
- Keep in mind that it's impossible to cram an undergraduate degree and several years of industry experience into one course
 - If you really want to improve, you have to treat this course as a starter's block, not a finishing line



Setting Up

- Some previous programming experience
 - for loops, if/then/else, ...
 - Function calls, arrays, file I/O, ...
 - If you haven't done at least this much, you probably won't understand the problems this course is trying to solve
- Some tools
 - [Python](#) (version 2.5 or higher)
 - We'll look at why later
 - A Unix shell
 - Could Use [Cygwin](#) if you're on Windows, but we will use a Fedora Core dist
 - An editor (preferably one with syntax highlighting and macros)
 - We'll look at integrated development environments later in the course
 - [Subversion](#)
- Some time
 - You can only learn by doing
 - So expect to spend 2-3 hours outside class for each lecture

A Note on Linux and Unix



Philosophy

- Most of the tools used in this course are freely available under various open source licenses
- Makes Unix/Linux development attractive for this reason along
- However, Unix philosophy of software development goes even beyond this, and is embodied in many of the tools we will look at.
 - We will learn more about this as the course progresses and you get more familiar with Unix/Linux environment.



Summary

- 36 hours of instruction and 120 hours of practice can change your life for the better.
- Let's get started...



Assignment

- Assignment for Thursday
 - You MUST figure out how to get shell access remotely from your preferred work environment to your NISL account.
 - You MUST change your password from the default password.
 - Please get started on the readings. You MUST read the UPU 2 & 3 at a minimum, as you will need some basic familiarity with shell commands (we will be going over them fast).