CS35L: Software Construction Laboratory
Lecture 1 Notes

Neel Redkar

April 2, 2024

Contents

1 Course Overview

1.1 What CS35L Is About

CS35L is designed to bridge the gap between knowing how to program and being able to create fully functional software systems. It focuses on the "other stuff" beyond just writing code: making programs work, understanding software ecosystems, and becoming proficient in various tools and technologies. The course assumes familiarity with programming and aims to prepare students for more advanced topics in operating systems, networking, and security by ensuring they have a solid foundation in software construction.

1.2 Key Learning Areas

- **File Systems**: Understanding how data is organized on secondary storage, including POSIX and Linux file system standards.
- **Scripting**: Learning to automate tasks and manipulate data with scripting languages like Shell, Python, and Emacs Lisp.
- **Building and Distributing Software**: Covering the processes of compiling, linking, packaging, and distributing software.
- **Version Control**: Using Git for managing changes and collaborating on software projects.
- **Debugging**: Techniques and tools for identifying and fixing errors in software.
- **Client-Server Models**: Developing applications that operate over networks, including basics of web development.
- **Security Basics**: Introduction to securing software against common vulnerabilities.

1.3 What CS35L Isn't

The course does not cover how to become a software entrepreneur or manage software development teams. It doesn't focus on making you famous in the software world or delve deeply into any single programming language like C++.

1.4 Technologies You'll Learn

- **Shell**: Bash scripting and command-line utilities.
- **Emacs**: As an Integrated Development Environment (IDE) for coding, debugging, and managing projects.
- **Git**: For version control, including both basic usage and understanding its internals.
- **Python**: For scripting and automation tasks.
- **Client-Server Technologies**: Basics of web development, possibly touching on Node.js and React for project work.

2 Introduction to Emacs

2.1 What Is Emacs?

Emacs is more than just a text editor; it's a highly customizable, programmable environment where you can code, compile, debug, and manage your entire workflow. It operates on the concept of buffers (in-memory text) and files (text stored on disk), allowing for efficient editing and manipulation of text data.

2.2 Getting Started with Emacs

- **Opening Emacs**: Use `emacs -nw` in the terminal for a non-windowed (console-based) session, ideal for remote development or when working in a terminal-centric environment.
- **Basic Commands**:
  - `Ctrl-x Ctrl-f`: Open or create a file.
  - `Ctrl-x Ctrl-s`: Save the file.
  - `Ctrl-x Ctrl-c`: Exit Emacs.
  - `Ctrl-x b`: Switch between buffers.
  - `Ctrl-x Ctrl-b`: List all buffers.

2.3 Buffers vs. Files

Buffers are sequences of characters stored in RAM, fast to access and manipulate but volatile. Files are sequences of characters stored on disk, persistent but slower to access compared to buffers. Emacs seamlessly bridges the gap between buffers and files, allowing you to edit files efficiently by loading them into buffers.

2.4 Emacs for Software Construction

Emacs is particularly suited for software construction due to its extensibility and integration with various programming tools and languages. Through Emacs Lisp, users can customize and extend the editor to fit their workflow, automate repetitive tasks, and integrate with version control systems like Git.

2.5 Practical Tips for Using Emacs

- **Learn the Keyboard Shortcuts**: Emacs efficiency comes from its extensive use of keyboard shortcuts for nearly all its functions.
- **Customize Your Environment**: Spend time learning Emacs Lisp to customize your editing environment to your liking.
- **Use Emacs for More Than Coding**: Explore its capabilities for project management, debugging, and even email and calendar integration.