Lecture 3

Organizing Code

- Introductions
- Lab 0 – Preview
- Workflows in CSCI 136
- Code Organization
  - `ssh` for secure remote connections
  - `nano` for console text editing
  - `git` for code management
Lab 0 — Preview
Lab 0 — Java/Git Intro

Labs are released online at 5pm on Tuesdays. They are due the following Tuesday before 5pm.

- Try to look at the lab handout before your lab, and (ideally) start working on it.

You may want to practice visiting TAs and labs.

- TA rooms and hours are on Google Calendar.

The main goal of Lab 0 is to familiarize yourself with the computing environment.

- You will move between the Mac and Unix labs.

It also is chance for you to test your basic Java programming (e.g., writing *Hello, World!*).

- Try to do this without looking at your notes.

Lab 0 handout has 16 steps across 6 pages.
Note: This weekly schedule is subject to change. Check Google Calendar for any updates.
Workflows in CSCI 136
**Supported Workflows**

1. Use a computer in the Mac lab (TCL 217A). Labs start here.
   - Login with your Unix credentials (Lida Doret or Mary Bailey)
   - Each Mac computer has its own local environment.
     - You will need to configure `git` on each one you use.
     - A file in your home folder only appears on one computer.
   - Text editors with windows are available.
   - Room is shared with CSCI 134.

2. Use a computer in the Unix lab (TCL 312).
   - Login with your Unix credentials (Lida Doret or Mary Bailey)
   - Unix computers share their local environment.
     - You configure `git` once.
     - Files in your home folder are accessible on all.
   - Only terminal-based editors are available.
   - Room is **not** shared with CSCI 134.

3. Use a computer in the Mac lab and `ssh` to a Unix machine.

4. Using your own computer to `ssh` to a Unix machine.
   - Terminal on Mac; WSL2 on Windows; Linux on Chrome OS.
Use a computer in the Unix lab to \texttt{ssh} to another Unix machine!

- You could do this multiple times.
- Please be reasonable!

Notes:
- The Mac machines cannot be logged into in this way.

Using your own computer (without \texttt{ssh} to a Unix machine).

- Use any text editor that you like.
- Not supported for many reasons:
  - We can't control which version of \texttt{java} you have.
  - Your computer could break down.
  - Security issues with certificates.

Note: You could use this approach for writing your code, but the teaching team won't answer any questions that you have if you are running it on your own computer, and your code will be tested and graded using our computing environment.
Code Organization
Code Organization

Java code allows for code to be organized in several ways.

- Instructions are organized into *methods* (a.k.a., functions). Examples from Lecture 1.
- *Classes* have methods and *attributes* (a.k.a., fields) and are stored in files. Examples from Lecture 2.
- *Packages* have classes and are stored in folders. Lecture 4 introduces Duane’s *structure* package.

More broadly, code organization can refer to a number of higher-level concepts.

- *Version control*. The history of a file can help us find regressions (i.e., where bugs originated).
- *Collaboration*. Large projects involve many developers working on the same code base.
- *Branches*. New features need to be isolated during development.

The industry’s standard tool for the above points is *git*. We’ll only use a fraction of its features.

Code organization can also refer to a number of concerns.

- Working on other computers using *ssh*.
- Configuring tools on a given computer.
  - Setting up *git*.
  - Configuring a text editor.
ssh
Live Coding: `ssh`

Let’s login to one of the machines in the Unix lab.

- `ssh user@machine.cs.williams.edu`
  - `user` is your Unix credentials
  - `machine` is the (cow-based) name of one of the machines in the Unix lab
    - see Lab 0 handout for a list of cows (excluding those in the “Knuth lab” at the back of the Unix lab)
  - e.g., `ssh aaron@lohani.cs.williams.edu`
  - `exit` terminates the connection
  - the connection may terminate for other reasons (e.g., network goes down or the laptop lid is closed, etc.)
    - make sure to save your work periodically

- Take a look at the local manual page (`man`) and tldr page online.

Note: Summaries of Live Coding demonstrations will be added to the slides when they are posted to the course website.
ssh from Terminal on Aaron’s Macbook (left) to the Unix environment on lohani. Note that the list of files (using ls) are different.
nano
nano is one of the simplest terminal-based text editors.

- Learn more: `man nano` or `tldr nano` (online version: `tldr.sh`) or `Ctrl+g` in the program.
- For configuration refer to the `~/.nano` folder and the `~/.nanorc` file.
- Our Unix machines have Version 4+ but the Mac machines may only have Version 2.
- Other terminal options: `emacs` or `vi(m)`. Atom is an excellent non-terminal text editor.
Live Coding: Configuring nano

Let’s make the following changes to the default behavior of nano.

- Change the length of the tabs.
  - By default a tab is 8 spaces in nano (yikes!).

- Tell nano to support mouse clicks.
  - By default it doesn’t listen to the mouse (yikes!).

- Take a look at the local manual page (man) and tldr page online.

Note: Most programs in Unix / Linux can be configured in similar ways. For example, git uses .git folders.
Move to your home folder with `cd ~`
Then edit a file called `.nanorc` by running `nano .nanorc`
Add the `set mouse` and `set tabsize 4` lines and save it with Ctrl+O (^O).
Exit `nano` with Ctrl+X then check the file contents using `cat .nanorc`
After the change, the tab length has changed from 8 spaces to 4 spaces.
After the change, `nano` displays tabs as 4 spaces instead of 8 spaces.
Search for the text `loop` and `yikes` in `nano` using `Ctrl+w`
Then the file `~/.nano/search_history` will be updated accordingly.
git
Live Coding: git

The basics of git that are used in this course.

- Configuring git on your machine.
  - `git config` use option `--list` to see current settings

- Creating a local copy of an existing git repository.
  - `git clone`
  - In this course, we'll use username / passwords for security; GitHub now only allows ssh keys
  - Side note: creating a new repository can be done with `git init` but you will not need to do this in CSCI 136

- Download changes to the repository.
  - `git pull`

- Making changes to repository.
  - `git add` on each file or file(s) that you have changed
  - `git commit -m` including a commit message
  - `git add` on each file or file(s) that you have changed
  - `git push` including `origin main` or `origin main` as needed

- Checking on the repository.
  - `evolene.cs.williams.edu` for the repositories in this course
  - `git status` for the status of a repository

Note: Summaries of Live Coding demonstrations will be added to the slides when they are posted to the course website.