Computer Science 136
Data Structures
Lecture #3 (September 14, 2021)

Organization and workflow.

1. Announcements:
   (a) The first lab ("lab 0") is available on the web.
   Labs are typically released Tuesday afternoon.
   Labs will be due on the following Tuesday at 5.
   (b) Make sure you have tested your username and password on one of our machines in TCL 217a or TCL 312 before you arrive for the lab meeting.
   The combination for all labs is: 3-9-2-7-8-1.
   Third floor locks: press hash mark (`#`) before combo.
   (c) Labs are 75 minutes, but we expect that some labs will take considerably longer, on your own.
   This week we hope to be done by the end of lab.
   (d) TAs are available (probably in TCL 312) for most days of the week. Check the calendar:

   `https://tinyurl.com/cs136-calendar`

   (e) Questions?
2. Organization of Java programs:
   (a) All code is part of some class definition.
   Every class is described in a file of the same name.
      i. Most classes describe how objects are constructed, accessed, and modified. The class `java.lang.String` is an example of this. Most classes in Python have this structure.
      ii. Some classes contain collections or "libraries" of `static` methods. These classes are not meant to be factories for creating objects; they're simply an organizational structure. The class `java.lang.Math` is an example of this.
   (b) Some class has a `main` method.
   That class is what we run with `java`.
   (c) When Java compiles code, it identifies types/classes that have not been defined and compiles those classes first. Java may automatically compile many classes!
   (d) Generally, Java searches for class definitions a `.class` in the current directory. If one is not found, it searches for a `.java` to compile. If neither are found, the search continues in the directories mentioned in the "class path." More about this later.
3. Coursework organization.
   (a) We will use various machines in the Computer Science environment. Your account gets you access to more than 100 CS machines in 5 labs.
   (b) On Mac computers, your home directory is stored *locally*: you have a different home directory on each computer. If you use a different Mac each week, you will have to set up a new environment each week.
   (c) On Ubuntu computers, your home directory is stored on a *server*: you have one home directory, visible from each of the Ubuntu machines. Only one setup is necessary.
   (d) Wherever you work, we suggest you create a dedicated `cs136` directory to hold all your work for that environment. Your labs (and other resources) will be stored as subdirectories under `cs136`.
   (e) Soon, we will also show you how to store instructor examples and *Java Structures* resources under the `cs136` directory.
4. We will be using the `git` version control system to keep track of your work:
   (a) Logically, we save our work on our `GitLab` server, `evolene`, in private *repositories*. These repositories are initially constructed on the Tuesday before lab.
   (b) The first time we work on a lab in a particular environment, we `clone` the repository from `evolene`. That command looks something like:

   ```
   cd ~/cs136
   git clone ...lab3-URL...
   ```

   where `...lab3-URL...` is the specification of your repository.
   (c) Alternatively, if you're returning to work on an existing repository, you don't `clone`, you `pull` down any new versions you might have tracked elsewhere:

   ```
   cd ~/cs136/lab3
   git pull
   ```

   (d) You work, editing files, changing content.
   (e) If you modify (or create) a file, `f.java` in the repository, you need to tell `git` that you want it to keep track of this version of that file:

   ```
   git add f.java
   ```

   You may `add` as many files as you want, either in a single command or in several.

(f) When you are finished `add`-ing files to be tracked, you `commit` your work, specifying a message that describes this version:

```
git commit -m 'Fixed a spelling error.'
```

(g) We suggest that you `add` and `commit` frequently. This will allow you to document the progress you are making on your code. It will allow you better control if you need to reverse any changes you make.

(h) Whenever you leave the lab, **always** `add`, `commit`, and then `push` your work to the server:

```
git push
```

Because you are making changes to the server, it will ask you for your password. Once it is finished with the `push`, you can logout and leave the lab.

This workflow—`pull`, edit, `add`, `commit`, and `push`—will ensure whatever machine you use, you are always working on latest version.

(i) The `git status` command will help you keep track of the status of your files:

i. `Untracked files`. These files are not being tracked. Changes to these files will not be recorded on the server. Either they're unimportant to you, or you should `add` them.

ii. `Changes not staged`. These files have changes that have not been included in the latest version. You should `add` them to some commit before your next `push`.

iii. `Changes to be committed`. Changes in these files have been added, but you must `commit` them for those changes to be remembered.

iv. `Your branch is ahead...`. You have successfully committed one or more versions of your project that have not been pushed up to the server. You must `push` the commits to the server to make sure they are visible to other computers.

(j) The `git log` command will help you see the progress you are making on the project.

i. To see the list of versions you've committed, type

```
git log
```

This will generate a list of entries that look like these:

```
commit be1fa50f215aa7e576e2...
Author: Duane A. Bailey <bailey@...
Date:   Wed Sep 8 10:23:24 2021

    Fixed spelling error.

commit f0d03b4ddbe5a979e1de...
Author: Duane A. Bailey <bailey@...
Date:   Tue Sep 7 13:04:09 2021

    Lab ready for testing.
...
```

Instead of version numbers, each commit is described by a string of digits in base 16 (a number system that uses digit values 0-9 and a-f). These strings are essentially random (a small edit generates a very big change in the string). It is typically safe to describe a version with the first 7 or more letters of the commit string.

ii. The command

```
git log --one-line
```

prints a very compact version of the log file with lines that look like:

```
be1fa50 Fixed spelling error.
f0d03b4 Lab ready for testing.
...
```

iii. The commit strings are useful if you need to recover work from an older version (we'll not discuss that, but the details are available on line).

Notes: