Computer Science CS136 (Fall 2021)
Duane Bailey & Aaron Williams
Laboratory 0
Setting up a Git and Java Environment

Objective. To set up a workflow for using Git and Java on CS lab machines.

Overview. This semester, we will be making use of the computers in the various Computer Science labs:

"The Mac Labs" in Chemistry 216 and 217a. These labs are populated with Mac computers running macOS, a unix derivative. Your CS credentials will give you access to each Mac. The files are not shared between Macs: the work you perform on one computer is stored locally on that machine.

"The Unix Lab" in Chemistry 312. This lab is populated with Ubuntu, another unix derivative. Your CS credentials will give you access to each workstation. Files created on these machines are all shared: the work you perform on any of these computers can be accessed from any other. Because they are central to the department, each machine is named after a breed of cattle. Here are the current names of these machines:

```
amerifax bagual barzona brava charolais devon galloway guernsey kuri
lidianiata panda rathi reina sharabi sind siri sykia tundaca zebu
```

These machines are accessible from anywhere on campus as, for example, `amerifax.cs.williams.edu`. The department also has three "compute servers" that are accessible from off-campus. These machines are:

```
lohani limia deoni
```

Our purpose is show you how to make use of any of these labs. This will allow you to work in the environment of your choice, without having to use your personal computer.

This Week's Tasks. The following steps will guide you through this week's lab assignment. Make sure you read and follow the instructions carefully:

1. During our labs, we'll be sitting in front of Mac computers. Log into one of these computers and provide your Computer Science username and password, your *CS credentials*. Notice that your CS username has your graduation year in front of what would normally be your Williams username.

2. At the bottom of the screen is the application *Dock*. From the "Go" menu, select "Utilities". This will open a folder with common utilities. Find the application called "Terminal" and drag it into the Dock. The Terminal application allows us to type commands directly into a Unix command shell.

3. Start the Terminal application by clicking on its icon in the Dock. You will be presented with an interactive "shell". There are many Unix commands you can type at this point: try `date`, or `whoami`. The `exit` command will cause the command shell to exit. If you type

```
ls
```

unix will list of all the files that are in your home directory. You will notice this is listing all of the names of files you would normally see in a home folder in macOS. These are two views of the same thing.

4. Make a new sub-directory to store all of your CS136 files:

   ```
   mkdir cs136
   ```

5. The “cd” command will allow you to change the current directory. Change the current directory to be the `cs136` directory:

   ```
   cd cs136
   ```

   If you type

   ```
   pwd
   ```

   it will print the name of the current working directory.

6. Before we can ask `git` to download the lab, it’s useful to configure it. The following commands tell `git` who you are (make sure you replace Joe Cool’s username and email with your own):

   ```
   git config --global user.name 'Joseph Cool'
   git config --global user.email 'jcool@cs.williams.edu'
   git config --global push.default simple
   git config --global core.editor emacs
   ```

7. Now, we’ll clone the repository for this lab. In the following command, replace `22xyz3` with your CS username:

   ```
   git clone https://evolene.cs.williams.edu/cs136-labs/22xyz3/lab0.git
   ```

   This will download a copy of the lab for this week. It will be stored in a subdirectory called `lab0`.

8. Now, change into the `lab0` directory and get a directory listing. You should see something like the following:

   ```
   GradeSheet.txt
   LabHandout.pdf
   honorcode.txt
   ```

   The file `GradeSheet.txt` describes how we are going to assign grades to the lab. Typically, it will involve a checklist of items that must be completed. You’re currently reading `LabHandout.pdf`. The file `honorcode.txt` is a text file that we expect you edit and logically sign, indicating that the work you are submitting is your own.

9. Using the editor of your choice, create a new file, `First.java` in the `lab0` directory that prints out the string `"Hello, world."`:

```
public class First {
    public static void main(String[] args) {
        System.out.println("Hello, world.");
    }
}
```

To compile `First.java` type:

```
javac First.java
```

This process results in file `First.class` which can be run with

```
java First
```

Repeat the process of editing, compiling, and running this first java program until it prints `Hello, world.`

10. The command

    ```
    git status
    ```

    is useful in making sure that `git` is tracking the modifications you make to the work you want graded. Typing that now gives you

    ```
    On branch main
    Your branch is up to date with 'origin/main'.

    Untracked files:
    (use "git add <file>..." to include in what will be committed)
    First.class
    First.java

    nothing added to commit but untracked files present (use "git add" to track)
    ```

    We'd like to turn in `First.java` for credit, so it's important to track that file. Since `First.class` is the result of compiling the Java code, we don't need to track it. Let's tell `git` to track `First.java`. Type:

    ```
    git add First.java
    ```

    The `git status` command will now report:

    ```
    On branch main
    Your branch is up to date with 'origin/main'.

    Changes to be committed:
    (use "git restore --staged <file>..." to unstage)
    ```

```
new file:   First.java

Untracked files:
(use "git add <file>..." to include in what will be committed)
First.class
```

Now, it's time to commit whatever has been added:

```
git commit -m 'My first java program.'
```

This tells git that we're commiting to a new version of the repository. The `-m` indicates the string that follows is a message that describes the substantive changes associated with this new version. A `git status` at this point yields:

```
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

Untracked files:
(use "git add <file>..." to include in what will be committed)
First.class

nothing added to commit but untracked files present (use "git add" to track)
```

Notice how the commits on our local machine are *ahead* of `evolene`'s `origin/master` by one commit. If we would like that version to be stored on the server, we must push it:

```
git push
```

Now we see git's status is:

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
(use "git add <file>..." to include in what will be committed)
First.class

nothing added to commit but untracked files present (use "git add" to track)
```

This will contact `evolene` to update its view of your project. You will have to provide a password.

11. Edit the `honorcode.txt` file, writing your name in the appropriate place. By doing this, you are acknowledging this work is your own. Make sure you `add`, `commit`, and `push` this file.

12. We're pretty close to having finished the lab. Our next steps will repeat this process on the Ubuntu machines, upstairs. The reason that we're suggesting you do this is so that you can (1) feel comfortable using the Ubuntu machines, and (2) to demonstrate that the work you did in the Mac Lab is reflected wherever you decide to clone or pull the repository.

    Now, log into one of the Ubuntu machines in the unix lab. As mentioned, these machines have interesting names, different breeds of cattle. We'll use the Mac to log into one of these machines and clone a copy of the work you've done so far, on the machine upstairs. Suppose you decide to log into one of the compute servers, `lohani`. You would type:

    ```
    ssh lohani.cs.williams.edu
    ```

    This establishes a connection to a *secure shell* shell on `lohani`. All of your interactions with this remote machine will take place in this terminal window. Because many editors open new windows, you should limit editing to using an editor that will work within the window. Emacs is commonly used, but you may find other editors you like better.

    When you've logged in, you should create a `cs136` directory, change to that directory, configure `git`, and then `clone` the current version of your `lab0`:

    ```
    mkdir cs136
    cd cs136
    git config --global user.name 'Joseph Cool'
    git config --global user.email 'jcool@cs.williams.edu'
    git config --global push.default simple
    git config --global core.editor emacs
    git clone https://evolene.cs.williams.edu/cs136-labs/22xyz3/lab0.git
    cd lab0
    ```

    On the Ubuntu machines, this initial setup is only necessary once since your home directory is shared across the entire network.

    Now, when you type `ls`, you will see the following:

    ```
    First.java
    GradeSheet.txt
    LabHandout.pdf
    honorcode.txt
    ```

13. You should be able to compile and run your program, `First.java`.

14. When you type the command `w`, you get a list of the current users on the machine:

    ```
    12:43:41 up 8 days,  1:30,  7 users,  load average: 0.00, 0.00, 0.00
    USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
    expresso pts/0    137.165.124.204  11:32    1:11m  0.03s  0.03s -bash
    expresso pts/1    137.165.124.204  Wed12   47:48m  0.09s  0.09s -bash
    bailey   pts/4    137.165.120.176  12:22   19:45   0.03s  0.03s -bash
    bailey   pts/6    137.165.120.176  12:43    2.00s  0.03s  0.01s w
    ```

Let's capture this output to a file called `users.txt`:

```
w >users.txt
```

(You won't see any output because it was redirected to the file.) Let's `add`, `commit`, and `push` your copy of `users.txt` as a token of having been here:

```
git add users.txt
git commit -m 'Added a list of users logged into lohani.'
git push
```

15. At this point, you can type

    ```
    logout
    ```

    to close the secure shell connection and return to directly typing into the `Terminal` window on the Mac.

16. If you look to see what files are in the local directory, you'll see that `users.txt` is not there. That change to the repository is "up in the cloud" on `evolene`. To refresh the local version of the repository with changes there were committed elsewhere, we'll *pull* down those changes locally:

    ```
    git pull
    ```

    Any interaction with `evolene` will require a password, for security. After you've pulled down the most recent copy, you'll see those changes locally.

The first time you bring down an up-to-date copy of a repostiory, you use `git clone`. Any refreshing of that work is done with `git pull`.

Typically, when we enter the lab, we perform a `git pull`, to pull down any changes that might have been made while we were away from the machine. We work on our project and, when we leave, we `add`, `commit`, and `push` all your modifications work back up to `evolene`. If, for some reason, you decide to work in another lab (or on your own machine), the latest version of your work will have been stored in GitLab.

It's also important, of course, to make sure your work is pushed up to the servers so that we can grade it.

Finally, while we will work in lab in the Mac Lab, we encourage you to use the machines of the Ubuntu lab to perform work outside of lab hours. These machines are available for your use, and we expect that you will know how to use them in later courses. Furthermore, because the Mac Lab can be conjested with CS134 and CS136 students, you may find it more productive to physically work in the Unix lab.

Submitting Your Work. To get credit for this week's lab make sure that you've added, committed, and pushed the files `First.java`, `users.txt`, and `honorcode.txt` (containing your signature). The command:

```
git status
```

Should indicate that none of the above files have modifications that need to be committed. Finally, if you type

```
git push
```

The response `"Everything up to date"` is an indication that your work is reflected on `evolene`.

⋆