Computer Science 134 git Workflows
For use in Spring 2020

This draft document describes the basic git-based workflow we will use in Computer Science 134. It is meant to be a reference describing typical tasks we would expect in this course.

Basic Utilities

This course makes use of three basic systems:

- Python, version 3.
- `git` and `gitlab`
- Atom, or an equivalent editor.

You must have each of these in order to support the workflows you should expect in this course.

Accounts and the CS git Server

Everyone has an account on our CS servers. Typically, the account name is your OIT 'unix' account name, prepended with your class year. You have been given a random password. Please make sure you change this to something you can remember. In this document, we'll represent this account with 22gmh0, the fictional account of Grace Hopper.

We have several servers that support different types of services. For our use, `evolene` is a server that holds the files we'll be using for this class. The full name of this server is `https://evolene.cs.williams.edu`. It is visible anywhere inside the college.

If you are off campus and wish to do CS134 work, you will need a Virtual Private Network (VPN) connection to campus to do so. OIT has instructions on how to do this, here:
`https://oit.williams.edu/help-guides/wifi-and-wired-connections/vpn/`

Setting Up a New Environment

The following should be done the first time you use any new environment. Over the semester, you may use machines in CS labs, OIT machines, or, perhaps, your own computer. You can tell if you're setup for use in this environment by looking for a `cs134` folder (referred to as a `directory` in your Home folder on the Mac (`Shift-Command-H`). If it doesn't exist, you need to set up the environment as follows:

1. Click on the `Terminal` application (identified by a `>_` symbol) in the dock. If the application is not in the dock, try searching it on the Mac (`Command-Option-Space`); if unsuccessful you must install it. We will use the `Terminal` frequently as part of our workflow.

2. Create a directory that will hold all of your CS134 work in your home directory. Throughout the course we will assume this directory is called `cs134`.[1] The `mkdir` command implies `make directory`. The `~` tilde indicates your home directory (i.e., where `Documents`, `Downloads`, `Music`, etc. are all located). The `cs134` is the name of the new directory.

   ```
   mkdir ~/cs134
   ```

3. Now, descend into the `cs134` directory (folder).

[1] You can name it anything you like, but in the future you'll have to remember to change instructions so they work with your particular setup.

```
cd ~/cs134
```

Retrieving a `git` Repository to Work On

1. Log-on to `https://evolene.cs.williams.edu` using your CS account credentials on a web browser. When asked for a password, use the password associated with your CS account.

2. On the homepage after you log in, you should see a link to a repository named `cs134-s20/lab01/22xyz3` (where the last part is your username), click on it.

3. On the right side near the top is a blue button that is a drop-down menu that says `Clone`. Click on it and select the `Copy URL to clipboard` button under the `Clone with HTTPS` text. This copies the URL of the git repository for this lab.

4. Return to the `Terminal` application and type `git clone` followed by the URL of the lab project you just copied (you can paste on Mac by pressing `Command-V`) followed by the name of the lab `lab01`. Your line should look something like this:

   ```
   git clone https://evolene.cs.williams.edu/cs134-s20/lab01/22xyz3.git ~/cs134/lab01
   ```

   Here, `evolene` is the gitlab server dedicated to holding all of our collective work. The `cs.williams.edu` is the Computer Science IP domain at Williams. The `cs134-s20` reference is the course home directory, `lab01` is a directory we use for organizing student files by lab number, and `22xyz3.git` is the name of your repository on the server. `~` is your home directory, `cs134` is the directory you made for this class, and `lab01` is the name of the directory you want for this new copy of the lab starter files. `git` is pre-installed on all Lab machines; if using a personal machine you must install it from `https://git-scm.com/`.

Modifying Files in `Atom`

Finally, we need an editor in which we will write our scripts. We will use `Atom`—you should see the application in the dock (identified by a green atom symbol). Again, if it is not there, you can search for it on the Mac using the shortcut `Command-Option-Space`. `Atom` is pre-installed on all Lab machines; if using a personal machine you must install it from `https://atom.io/`.

1. Open the editor `Atom` by clicking on the application. You will see some welcome tabs, which you can ignore or close (there's a checkbox on the `Welcome` tab to hide these tabs in the future). On the top left, click on `File` and select Add Project Folder from the dropdown menu. Click the scroll-down menu at the top of the File Chooser, and find your user directory. Navigate to the `Lab01` folder within your `cs134` directory and enter (single-clicks only!). Click the `Open` button. You should see the folder added to your Project tab on the left, with the starter files.

2. Click on the `python` file you'd like to edit in the left pane. Let's say for this example, it's `hello.py`. It will open up in the editor pane on the right. You can now edit this file in the editor pane.

Testing Your Python Code

Expert programmers write tiny chunks of code and test them frequently! If you write a lot of code and then test, it can be difficult to find out where any issues are. To test our code, we return to `Terminal` and run the following command:

```
python3 hello.py
```

The python interpreter will print out any warnings, errors, or program output to the `Terminal`. Now we can go back-and-forth between the errors and our code in `Atom` to debug any issues. Make sure you have saved your work before attempting to run it with Python, Atom does not automatically save edits!

This command will only work if you are in the same location as `hello.py`. To retrieve a list of items in your current Terminal location, type `ls`. If you do not see `hello.py` listed there, you are not in the correct location. Try changing directories with this command: `~/cs134/lab01`, and then check to see if the correct files are there. You can see where you're currently located in the file system by `pwd` (provide working directory).

Submitting Your Work

At any point, you can push your work to the server as follows. First, make sure you have saved all your edits to `hello.py` in `Atom`. `Atom` does not automatically save any changes, so it is important to frequently save your work.

1. From the top menu in `Atom`, go to `Packages` and select `Github` and click on `Toggle Git Tab`. A new pane will appear on the right with the title `Git`.
2. You should see `hello.py` under `Unstaged Changes` in the `Git` pane. Right click on the file name and select `Stage`, which will move the file to the `Staged Changes` section below.
3. Once all your edited files are staged (in this case `hello.py` and `honorcode.txt`), you are ready to commit your work. Write a brief but descriptive commit message in the text box at the bottom of the `Git` pane and click on `Commit to master`. This commits your work locally and you do not need to be connected to the internet to commit.
4. After you commit a `Push` button will appear at the bottom of the pane. Clicking on it will push your work to the CS server. You need to be connected to the internet for this step. After you push your finished work, you are done!

Certifying the Honor Code

Every week you will need to certify that your work is your own. Open up the `honorcode.txt` in `Atom` and type out the Honor Code statement, along with your full name. Be sure to `Stage` your changes to `honorcode.txt`, `Commit to master`, and `Push` just as you did with your python work.

Grabbing File Updates Without an Entirely New Copy of Your Files.

Sometimes, we want an updated version of our files, and not a brand new copy. This would happen in the following situations (among others):

1. (1) You `git clone` the lab repo on a CS Lab machine, make edits, and then `Push` your edits back to the server. (2) Later, you `git clone` an updated copy to a second machine, perhaps your personal laptop or another Lab computer, make edits and `Push` the edits back to the server. (3) Before submitting your final version, you return to the original machine from (1), hoping to make some more changes. You don't want a brand new copy, as you already have the files on that machine. You just want the updates!

2. You and a partner are working on a project together. You both `git clone` a copy of the starter files from the same repository. Your partner makes edits to the files and `Push`es them to the server. You want the updates your partner made, but you already have a copy of the files. You just want the updates!

In this case, what you'll want to do is called `Fetch` the updated files. In `Atom`, where the `Push` button usually appears, it should say `Fetch`. Press the `Fetch` button, and your files should update.

Mental Model

`git` provides us with a powerful version-tracking ability with files stored on a central server. As you make changes on your local machine, your repository holds work not stored in the global repository (i.e., central server). *Whenever* you are done working, you should perform a `stage` (as above), and then `Push` your new work up to the server's version of the repository. This causes all the commits (changes) in your local repository to be applied to the global repository. It is important to remember that all the different versions of your data that you've ever committed are remembered. If you think you made a mistake, you can always *checkout* an older version (not described here). The only way, however, that you can keep track of these older versions is if you

1. `Fetch` at the beginning of a work session,
2. `Stage` and `Commit` often, and
3. `Push` at the end of the work session.

When you `Push` changes up to the server those changes will be transferred to other environments you may use when you perform a `Fetch` in those locations.

Private CS134 Work Repositories.

Whenever we start working on a new lab in a new environment, say `lab03`, we clone our private repository from the server. (We suggest this repository be stored in your local `cs134` directory you constructed at the beginning of the semester.) This gives us access to starter files and handouts for that lab:

```
git clone https://evolene.cs.williams.edu/cs134-s20/lab03/22gmh0.git ~/cs134/lab03
```

If you perform work in more than one environment (say two or more OIT machines or your laptop), you'll need to `clone` (exactly once!) in each location.

Command Line.

If you prefer to manage `git` entirely in `Terminal`, your workflow might look like the following (after cloning once already):

1. Pull at the beginning of a work session:

   ```
   git pull
   ```

2. `add` and `commit` one or more times:

   ```
   git add <changed files>
   git commit -m '<*useful* comments about changes>'
   ```

3. `push` at the end of the work session:

```
git push
```

When your assignments are graded, we will update your repository with a grade file and add helpful feedback/comments to your submitted files.

What To Do When You Use a New Machine

Occasionally, you'll have to start fresh on a new OIT machine. Here are the commands you'll likely find useful as you start up:

```
mkdir ~/cs134
cd ~/cs134
git clone https://evolene.cs.williams.edu/cs134-s20/lab03/22gmh0.git ~/cs134/lab03
```

Terminal Commands

If you are interested in how to navigate the File Directories via `Terminal`, there is a resource guide on the CS134 course website under `Resources`, titled `Duane's Incredibly Brief Intro to Unix and Emacs`.

⋆