CSE 391, Spring 2020
Homework 4: Version Control (Git)
Due Tuesday, April 28, 2020, 1:00 PM

This assignment focuses on using Git for version control. You can do this lab from the CSE VM, from attu, or from another place you have linux and Git installed. For Task 1, you will copy and paste the output from certain git commands into Gradescope. **There is no autograder for this assignment.** You will still submit your answers to Gradescope, however they will be manually graded after the deadline.

Git is a fairly complex tool that can be used in many different ways. We will show you common ways you might use git in a course or for your own projects. But if you run into problems with Git, be aware that doing a web search for answers could lead you to a solution that refers to a different problem than the one you have. The homework page on the course website links to several resources that you may find helpful.

Task 0: Getting ready to use Git and the CSE GitLab service

1. **Log on to GitLab** - All students in CSE 391 have been given access to the CSE GitLab service for this quarter. If you are a CSE major, you should log on using your CSENetID, otherwise use your UWNetID. Log on to CSE GitLab by going here: https://gitlab.cs.washington.edu/

2. **Add one or more ssh keys to your account** – In order to talk to the GitLab service from your computer or attu you will want to create ssh keys on those computers and copy the public ssh for each computer into your GitLab account. We suggest you **do not add a password**, even though the documentation says it is best practice. Read more about this here: https://gitlab.cs.washington.edu/help/ssh/README.md . The process is also described briefly on slide 18 from lecture. We suggest you accept the default location for the key and that you **do not add a password**, meaning you can just hit return three times.

   Once you have created a key on your local machine, next, on GitLab, click on the down arrow next to the circle at the top right of the screen and select "Settings". Next, select "SSH Keys" from the left-hand side of the page. Or you may go directly to the URL: https://gitlab.cs.washington.edu/profile/keys. On your local machine, type: `cat ~/.ssh/id_rsa.pub` to see your key. Copy and paste the key into the provided text box on GitLab. You can give it a Title like "attu" or "CSE VM" to identify which computer the key goes with. Click the green button labeled "Add Key".

3. **Configure Git** – Whether you are working on attu or the CSE VM, you will want to configure a few things before you get started with Git. Slide 19 from lecture gives more info, but basically type these things at your linux prompt:

   ```
   git config --global user.name "Your Name"
   git config --global user.email yourEmail@uw.edu
   ```

   If you want to use an editor other than vim for commit messages, you may want to set your default editor, for example

   ```
   git config --global core.editor emacs
   ```

   (Tip: if you find yourself in vim by mistake, use `:q` (colon, the letter q, then enter) to quit. More tips here.)

   You can check what you have set using `git config --list`

   Note: When you push, you may encounter a message like this when pushing:

   ```
   warning: push.default is unset; its implicit value is changing in
   Git 2.0 from 'matching' to 'simple'.
   ```

   This new default value in Git 2.0 will be fine. You can make this warning go away by setting `push.default` to be the new default in Git 2.0 like this:

   ```
   git config --global push.default simple
   ```

Task 1: Create a Git repository, import and edit files

This task gives you experience creating a Git repo and modifying files from it in a way that you may wish to use to manage your own individual projects. We suggest you place this repository on CSE GitLab but it will also be acceptable if you create it elsewhere (we only provide instructions for CSE GitLab).

1. **Create a repo on GitLab:** Click on the "W Gitlab" icon at the top left of the screen to take you to the Dashboard. Once at the Dashboard click on the green "New project" button on the upper right hand side of the screen. **Make sure your username is selected in the dropdown in the project URL.** Type (do not cut and paste from this document): `cse391-hw04` as the "Project name". Creating this as a private project (the default) is fine for this task. Hit the green "Create project" button.
2. **Add initial files to the repo**: After you have created the repo, the project page will show you customized instructions for adding your first files to it. The first option ("Create a new repository") involves cloning the empty repo and adding, committing, and pushing a README file to the remote repo. The second option ("Existing folder") is useful if you already have an existing folder of files that you want to add to the repo you just created. Here we will use the first option.

   Type these commands at a linux prompt in the directory where you would like to copy the (currently empty) repo:

   ```
   git clone git@gitlab.cs.washington.edu:yourUserID/cse391-hw04.git
   cd cse391-HW04
   touch README.md
   git add README.md
   git commit -m "add README"
   git push -u origin master
   ```

   After executing these commands, the directory you are in (named `cse391-hw04`) is your "working directory" in Git terminology. Your working directory is where you will edit and create files to add and commit to your local repo. Later you will push them to the remote repo on GitLab. At this point if you do an `ls -a` in your working directory you should see the file `README.md` (`md` stands for markdown) and a `.git` directory that is where your local copy of the repo lives. You will not cd into the `.git` directory or modify files in this directory directly. Instead this directory serves as your local copy of the repository and you will interact with it via Git commands (eg. `git add`, `git commit`). If you go to the GitLab web page for your repo, and refresh, you should see that `README.md` is now there since you pushed it to the remote repo.
3. **Add a file to your repo:** Copy the `menu.txt` file from the homework website into your working directory. If you are still in your working directory you can do this using:

   ```
   wget https://courses.cs.washington.edu/courses/cse391/20sp/homework/4/menu.txt
   ```

   Or, if you're using macOS

   ```
   curl -o menu.txt https://courses.cs.washington.edu/courses/cse391/20sp/homework/4/menu.txt
   ```

   Type `git status` and notice that `menu.txt` is shown in red and listed as an "untracked file". Now we will change this by asking `menu.txt` to be tracked with:

   ```
   git add menu.txt
   ```

   Typing `git status` again will show that `menu.txt` is now shown in green as a new file to be committed. Let's commit it to your local repo by typing (**do NOT cut and paste!**) the following:

   ```
   git commit -m "Adding menu to repo"
   ```

   Finally, let's push the changes in your local copy of the repo to the remote repo on GitLab.

   ```
   git push
   ```

   On GitLab, if you refresh the files, you should see that `menu.txt` is now there. Run `git log -1` (one, not el) to see only the most recent commit. **Copy and paste the results of `git log -1` into Gradescope.**
4. **Edit a file from the repo:** In your working directory, edit the file `menu.txt` and add lines or modify lines already present in the file. Now issue the command `git status`. Issue the command `git diff` to see changes that appear in your working directory that have not been staged. **Copy and paste the output of `git diff` into Gradescope.**
5. **Stage your changes to the repo:** Use the appropriate command to add your changes to the staging area. Then type `git status` and `git diff --cached` to see any staged changes. **Copy and paste the output of `git status` into Gradescope.**

6. **Commit your changes to your local repo:** Use the appropriate command to commit your changes to your local repo. Don't forget to use –m to add a one-line commit message or else you will be thrown into an editor to write your commit message. **Copy and paste the output of** `git status` **into Gradescope.**
7. **Push your changes to the remote repo:** Use an appropriate command to push your changes to the remote repo. Run `git status` to see what it says now. Type `git blame menu.txt` to see who was responsible for changing each line of this file when. **Copy and paste the output of** `git blame menu.txt` **into Gradescope.**
8. Feel free to experiment with adding more files, making edits and other commands such as: `status, diff, log, blame`. Try `git help` *`command`* for more info on these.