CSE 391, Winter 2020
Homework 4: Version Control (Git)
Due Tuesday, February 4, 2020, 2:00 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. For Task 2, you will add a file to a shared repository on the CSE GitLab server (you will be graded based on the presence of this file in the shared repository). **There is no autograder for this assignment.** You will still submit your answers to Gradescope, however they will be manually graded.

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/cse391/20wi/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 your 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 <u>add</u> 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 <u>commit</u> 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 <u>push</u> 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. Before moving on to the next task, 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.

Task 2: Contribute to a shared Git repository for FAANG on CSE GitLab

It's your second week at FAANG and you're starting to feel more comfortable. The frontend team needs more help with the website, and wants to get you integrated into their version-control workflow using git. The purpose of this task is to get practice committing to a shared git repository and resolving merge conflicts. You should have been access to the shared repository described below. **If you are having trouble cloning or pushing to the repository due to a permissions problem, you may need to double check your email for an invitation to the repository**.

All of the students in CSE 391 have been hired by FAANG to update the products on their products page. For this task, you will add a product that they think FAANG should sell. The product can be whatever you want, but keep it appropriate! Your mom may want to make a purchase on the FAANG website.

First, Clone a copy of the remote repo – On your machine (e.g. the CSE VM, attu), `cd` into the directory where you would like to create your local copy of the repo and execute this command:

```
git clone git@gitlab.cs.washington.edu:zorahf/cse391-faang-20wi.git
```

Since the repo is shared by so many people, most changes you make to the repo will require some sort of merge. Thus if you try to push a file and you get an error, then try pulling using `git pull`. If you are merely adding a file with your product photo and someone else has added another file, then on the pull you are likely to be thrown into your editor and asked to enter a commit message explaining why this merge is needed. You can just accept the message that is there by saving and exiting from the editor. See step 2 below for info on resolving more complex conflicts when they occur. **Don't forget to push again to really send your changes to the remote repo.**

1. **Find a picture that describes your product:** While we often think of git as a tool to manage text files, it really can be used for any type of file. Find a relatively small image file (e.g. a .jpg or .png roughly less than 2 MB) and add it to your local repository under `cse391-faang-20wi/images`. Once that is done, add, commit, and push your file to the remote repository. We will grade this section by looking at the remote repository and seeing that the file is there.
2. **Add a product to Products.java:** From the `cse391-faang-20wi` directory open up `Products.java` in your favorite text editor. `Products.java` is a simple program that contains one method for each of the products that our company sells. Create a method, which you can name whatever you want (as long as it's unique) that returns a new Product object that describes your product (see the sample `myProduct` method that is provided). In addition, one of the fields is for your picture. Simply include the name of the file (not including the `images/` prefix) as a String for this part of the question. Be sure that this class still compiles after editing!

   Since the `Products.java` file is going to be edited by everyone in the course, you should be sure to do a pull before you edit the appropriate file. To pull from the remote repo type:

   ```
   git pull
   ```

   Edit the `Products.java` file with your product. Then add and commit it to your local repo and push it to the remote repo using:

   ```
   git push
   ```

   When you do `git push` you may have complete success (a call to `git status` will show that `Your branch is up-to-date with 'origin/master'`). Hooray!!

However if you forgot to pull, or **if someone edited the same file and pushed it since you last pulled**, you may get a message when you try to push indicating that your changes were **rejected**, giving you an error that it failed to push some refs. If you read the "hints" Git prints out they should be useful. It suggests that the repo contains work that you do not have, likely because someone else has pushed their copy of the same file to the repo.

*** What to do when **`git push`** rejects your changes:

Do a `git pull`. This will bring changes in the remote repo to your local repo. Git will try to merge your changes with these new changes pulled from the remote repo. Git can merge many changes automatically, but sometimes it needs help. Two things are likely to happen when you do `git pull`.

- **If Git is able to automatically merge your changes with the new changes**: then you will be thrown into your default editor to either just save and exit, accepting the default merge message, or add something else to it before exiting. At this point you have committed the merged version of the file to your local repo, but you will need to try `push`ing it to the remote repo again. Hopefully this time you will succeed!

- **If Git is <u>not</u> able to automatically merge your changes with the new changes**: then you will see that the call to git pull reports there was a CONFLICT and that the "`Automatic merge failed; fix conflicts and then commit the result.`" Open up the conflicting file in an editor. When Git detects a file conflict, it changes the file to include both versions of any conflicting portions (yours and the one from the repository), in this format:

```
<<<<<<< filename
YOUR VERSION
=======
REPOSITORY'S VERSION
>>>>>>> 4e2b407... -- repository version's revision number
```

  For each conflicting file, edit it to remove the <<<<<<<, =======, and >>>>>>> lines. When repairing a conflict, do so in such a way that your changes, as well as changes made by all other students, are preserved. Please do not submit a change to a file that damages or removes changes made by another student. The repo will keep a record of all actions, so we will be able to tell who has submitted new versions of the file and what changes were made by each student. (More information about resolving conflicts, here.)

  After you are done resolving any conflicts, you will need to `add` and `commit` the edited file to your local repo. When committing, if you do not supply a message using `-m` at the command line, you will see a default merge message in your editor which you can accept as is or edit. **Finally, try `push`ing again to the remote repo.**

3. **(Optional): For fun!** Compile all java files in the repository's root directory and run the `GenerateProducts` program. This will produce an `index.html` file containing a list of all of the products out company sells, as well as an `html` file for each product. You can view these `html` files locally in a browser to see the products on the website. *Note:* Compiling `.java` files will generate `.class` files and `GenerateProducts.java` will generate several .`html` files. These should NOT be included in the repository. A git repository should only contain the source code needed to generate all necessary artifacts, but not the artifacts themselves. Every git repository has a `.gitignore` file in the top-level directory that contains the names of files should be ignored by git (for example, these files should not appear in `git status` or `git add`). If you inspect the `.gitignore` file for our repository you will see that `.class` and `.html` files are listed. **DO NOT MODIFY THE .gitignore FILE.**

- Tips:
  If you ever run the `git status` command and see the output "`Your branch is ahead of 'origin/master' by N commits`", then this means that you have local commits that have not been pushed to the remote repository yet.
- Git will not allow you to pull updates into your repository while you have **unstaged** changes. Therefore, you must stage (`git add`) and commit all of your changes before pulling.