CSE 391, Winter 2020
Homework 5: Version Control (Git)

Due Tuesday, February 18, 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. Similar to last week, **there is no autograder for this assignment.** You will still submit your answers to Gradescope, however they will be manually graded.

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. On the same machine you used for Homework 4, navigate to your `cse391-faang-20wi` directory that contains the shared repository. You will be doing all of your work for this assignment from this directory.

Task 1: Create a new feature branch

It's your third week at FAANG. You're starting to hit your stride. The company is proud of the products it has put together on the website and FAANG is getting ready to re-launch the site. The frontend team wants to build a staff page that features all of the incredible staff at the company. You will be adding yourself to that page! The purpose of this task is to get practice creating and merging feature branches. You should have been access to the shared repository `cse391-faang-20wi`, which we used for the last homework assignment.

1. **Update your local repository to contain the latest master:** Many other engineers have been hard at work updating the shared company repository and it has likely changed since you last worked on it. First, make sure that your current branch is on master by running

   ```
   git branch
   ```

   You should see the branch name master with an asterisk (*) next to it, indicating that master is your current branch.

   ```
   * master
   ```

   Then, to update your local copy of master to match master on the remote repository, type

   ```
   git pull origin master
   ```

   You should see output that looks something like this, followed by a list of updates made to the repository that you're pulling down.

   ```
   From gitlab.cs.washington.edu:zorahf/cse391-faang-20wi
    * branch            master     -> FETCH_HEAD
      6745c4b..6c8d66e  master     -> origin/master
   Updating 6745c4b..6c8d66e
   Fast-forward
   ```

2. **Create a new branch for your additions to the staff page**: Your work on the staff page should be on a feature branch, separate from the main master branch. Run

   ```
   git checkout -b <yourNetID>_staff_page
   ```

   Copy and paste the results of both `git branch` **and** `git log -1` **into Gradescope.**

3. **Add your bio to the staff page:** From the `cse391-faang-20wi` directory open up `Staff.java` in your favorite text editor. `Staff.java` is a simple program that contains one method for each employee at FAANG. Create a method, which you can name whatever you want (as long as it's unique), that returns an `Employee` object that represents you. An `Employee` has a bio, picture and job title. In addition, you should add a photo that represents you to the `images/staff` directory. Simply include the name of the file (not including the `images/staff` prefix) as part of your Employee object. Be sure that `Staff.java` still compiles after editing!
4. **Push your changes to the remote repo:** Use the appropriate commands to commit your changes to your branch and push to the remote repo. You should be able to see your commits in gitlab using the following link

https://gitlab.cs.washington.edu/zorahf/cse391-faang-20wi/tree/<yourNetID>_staff_page

Alternatively, you can navigate there by going to the repository main page https://gitlab.cs.washington.edu/zorahf/cse391-faang-20wi and clicking on "Branch" at the top of the page. Here you will be able to see all of the branches for this repository, including yours.

5. **Make a merge request in Gitlab:** When you've gotten your work in a completed state, you will typically have another teammate review your code before merging your changes in the master branch. In Gitlab, you can create a "merge request", which allows a teammate to review, comment on and approve your work. Create a pull request for your feature branch in Gitlab. You can do so by selecting "Merge Requests" on the left side-pane (the icon with the arrow), clicking on "New Merge Request" and then selecting your branch in the dropdown under "Select source branch". Your target branch should be master. Click "Compare branches and continue". Give your merge request a title without the WIP: prefix and write a description of your changes. Leave the other dropdowns and checkboxes blank and submit your merge request.

   Copy the url of your merge request into Gradescope. It should look something like https://gitlab.cs.washington.edu/zorahf/cse391-faang-20wi/merge_requests/12345

6. **Leave a comment on your merge request:** After creating a merge request and requesting a review from your teammate(s), they can leave comments on your changes. You can leave comments on your own merge requests, as well. Leave at least one comment on a change in your merge request. It can say whatever you'd like.

7. **Merge your branch from gitlab**: Let's incorporate your changes into master so it can make it on the main website! The merge request that you created in Gitlab should have a Merge button at the top of the page. Click it. Your changes should now be in master! If the merge button is disabled because your branch has merge conflicts with master, use the resolve conflicts button to resolve the merge conflicts from Gitlab then proceed to merging to master.
   Copy the url to your merge commit in master to Gradescope.

Task 2: Create a new feature branch, again

You realized you want to make some changes to your bio, but you've already merged your changes! Let's create another feature branch for these changes.

8. **Update your local repository to contain the latest master:** You've merged your changes into master on the remote repository, but your local copy of master doesn't have these. Update your local repository to contain the latest master branch.
   Going back to your terminal, switch back to the master branch

   ```
   git checkout master
   ```

   You can verify that you are on master by running `git branch`. You should see the branch name master with an asterisk (*) next to it, indicating that master is your current branch, as well as your old feature branch, which you are no longer actively working on.

   ```
   * master
     zorahf_staff_page
   ```

   Now, to update your local copy of master to match master on the remote repository run

   ```
   git pull
   ```

   Similar to before, the changes that you are pulling from remote should be incorporated into your local copy of master with a Fast-forward merge.

9. **Create another new branch for your changes to your bio**: Use the appropriate command to create a new feature branch, based off of master, called `<yourNetID>_staff_page_again`

10. **Make some edits to your entry on the staff page**: You can either update the `Employee` object in `Staff.java` or change your staff photo. Add, commit and push these changes to the remote repository. **Copy and paste the results of** `git log -1` **into Gradescope.**

11. Typically, while you are working on a feature branch, your teammates will have committed changes to master. In order to simulate changes to master on the remote repository, from the `cse391-faang-20wi` directory run

```
./simulate_change_to_master.sh
```

`simulate_change_to_master.sh` is a script that will create a commit on the remote master on your behalf. This may take a couple of minutes. If you set up your ssh keys for Gitlab with a password, you will have to enter your password.

12. After you've run the script above, checkout and pull the latest master.
13. **Rebase your feature branch on top of the update master:** Checkout your feature branch `<yourNetID>_staff_page_again` and run `git rebase master`.

    - **If Git is able to automatically rebase your changes**: Nothing to do here! Git will display the commits that it replayed along with a success message

    - **If Git is not able to automatically rebase your changes with the new changes**: you will see that the call to `git rebase` reports there was a `CONFLICT` with at least one file. You fix rebase conflicts similarly to merge conflicts. Open up the conflicting file(s) 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:

      ```
      <<<<<<< HEAD
      REPOSITORY'S VERSION
      =======
      YOUR REPOSITORY
      >>>>>>> message of the commit being rebased
      ```

      For each conflicting file, edit it to remove the <<<<<<<, =======, and >>>>>>> lines. When repairing a conflict, the state you leave the file is what will be the final result of your commit edit the file 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.

      After you are done resolving any conflicts, `add` the files you fixed and run `git rebase --continue`. If you are rebasing multiple commits and more than one commit contains conflict, you will have to fix conflicts and `git rebase --continue` for each commit containing conflicts.

Copy and paste the results of `git log -1` into Gradescope.

Push your updated branch to remote: Run `git push` to push your updated feature branch to remote. When you do this you will notice that your changes will be rejected with a message like the following

DO NOT DO A `git pull`. EVEN THOUGH THIS IS RECOMMENDED IN THE REJECTION MESSAGE. Because rebasing commits gives them brand new SHAs, git thinks that your remote branch contains changes you don't have locally and vice versa. Instead, push your changes with git push -f. Verify that your changes successfully made it to your remote branch. **Copy the url of your branch `<yourNetID>_staff_page_again` from Gitlab into Gradescope.**

(Optional): For fun! Compile all java files in the repository's root directory and run the `GenerateSite` java program. This will produce an `index.html` file contains both links to the products the company sells, as well the new staff page. You can view any `html` files locally in a browser. If you want to only generate the staff page, without generating the products, you can run the `GenerateSite` program.