CSE 391, Spring 2020
Homework 5: Version Control (Git)
Due Tuesday, May 5, 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. 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/

Task 1: 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:joshue/cse391-20sp-faang.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 `cse391-20sp-faang/images/products`. Once that is done, add, commit, and push your file to the remote repository. **Write the name of your file in Gradescope.**
2. **Add a product to Products.java:** From the `cse391-20sp-faang` 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 `joshsProduct` 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/products` 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. This is the same issue during lecture when Hunter and Josh tried to push to the same repository on master. 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 not 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. Copy the url of this commit into gradescope.**

Task 2: Create a new feature branch

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.

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
   ```

   If any changes have been made since you pulled, 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:joshue/cse391-20sp-faang
    * branch            master     -> FETCH_HEAD
      6745c4b..6c8d66e  master     -> origin/master
   Updating 6745c4b..6c8d66e
   Fast-forward
   ```

2. **Create a new branch for your additions to the products 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 `cse391-20sp-faang` 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/joshue/cse391-20sp-faang/tree/<yourNetID>_staff_page

   Alternatively, you can navigate there by going to the repository main page https://gitlab.cs.washington.edu/joshue/cse391-20sp-faang 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/joshue/cse391-20sp-faang/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.

3. (Optional): For fun! Compile all java files in the repository's root directory and run the `GenerateSite` program. This will produce an `index.html` file containing a list of all of the products our company sells and the staff who work there, as well as an `html` file for each product and employee. To view the website you have a few options.

   a. If you're working locally you can navigate to the `cse391-20sp-faang` directory using Finder (macOS) or File Explorer (Windows) and double click on the `index.html` file. This should open the website in your default browser.

   b. If you're working on `attu` you will need to copy your files over to your local machine to view them. You can do this with the following command:

   ```
   scp -r CSENetID@attu.cs.washington.edu:/path/to/cse391-20sp-faang .
   ```

   Now that this is copied over, you can view them in a web browser using part **a**).

   c. If you're working on the VM you can use the File Manager application to navigate to the `cse391-20sp-faang` directory and double click on the `index.html` file, which should open the site in Firefox.

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.