Computer Science 35L: Week 5 Solutions

"Git Gud!"

Exploring Git

1. **Using GitHub.**
    a. Make a private GitHub repository titled "cs35l-assignments".
    b. Clone your newly created repository to your own computer.
    c. Create an "assignment3/" directory.
    d. Put your related assignment 3 work in this directory.
    e. Commit it and push it to GitHub.
    f. Clone your "cs35l-assignments" directory on SEASnet.

2. **Configuration.** Run these commands, replacing Gene Block's details with your own.
    a. `git config --global user.name "Gene Block"`
    b. `git config --global user.email "geneblock@ucla.edu"`

3. **Git GUIs.** Many students find it easier to work with Git, especially at first, by using a Git GUI program. Here are several: GitKraken, Tower, GitHub desktop (all of which are free for student use via the GitHub Education Pack). Editors like Visual Studio Code and JetBrains IDEs also have built-in integration with Git. Try one of these out and visually explore a Git repository.

4. **Git commands.** What do the following commands do?
    a. `git clone`
    Clone a repository.
    b. `git init`
    Initialize a git repository in the current directory.
    c. `git help`
    Provides information about various git commands.
    d. `git status`
    Gives the status of your current git state.
    e. `git add`
    Stages changes for commits.
    f. `git commit`
    Commits staged changes.
    g. `git log`
    List commits on the current branch.
    h. `git diff`
    Shows changes between commits, branches, etc.
    i. `git checkout`

Move to a different git branch, or create a new git branch.

j. `git push`
Send local changes to GitHub or some other remote Git host.

k. `git pull`
Retrieve changes from GitHub or some other remote Git host.

l. `git blame`
See who edited the lines of a file last.

m. `git show`
Shows diffs for a specific commit.

n. Now that you've explored these on your own, here's a great Git cheatsheet. :^)

5. Let's practice!
   a. Create a new local Git repository called "animals."
   You can do this by creating a new folder called "animals" and calling `git init` within that folder.

   b. Create a file called "dog.txt" that contains the text "Git me a dog!"
   ```
   echo "Git me a dog!" > dog.txt
   ```

   c. Create a file called "cat.txt" that contains the text "Git me a cat!"
   ```
   echo "Git me a cat!" > cat.txt
   ```

   d. What do you see when you run `git status`? Add "dog.txt" to the staging area. Did the output of `git status` change?
   Running `git status` will show "dog.txt" and "cat.txt" as "untracked file." To add "dog.txt" to the staging area, run
   ```
   git add dog.txt
   ```
   Running `git status` afterward will show "dog.txt" as a "change to be committed" and "cat.txt" as an "untracked file."

   e. Commit "dog.txt" with a helpful commit message. Verify your changes by running `git log`.
   ```
   git commit -m "Added a dog"
   ```

   f. Change the text of "dog.txt" to "Git me two dogs!" Commit these changes.
   ```
   echo "Git me two dogs!" > dog.txt
   git add dog.txt
   git commit -m "Added a second dog"
   ```

   g. Create and switch to a new branch called "cats."
   ```
   git checkout -b cats
   ```

Or...

```
git branch cats
git checkout cats
```

h. Add and commit "cat.txt" with a helpful commit message. Verify your changes by running `git log`.

```
git add cat.txt
git commit -m "Added a superior animal"
```

i. Return to the master branch. Does the output of `git log` change?

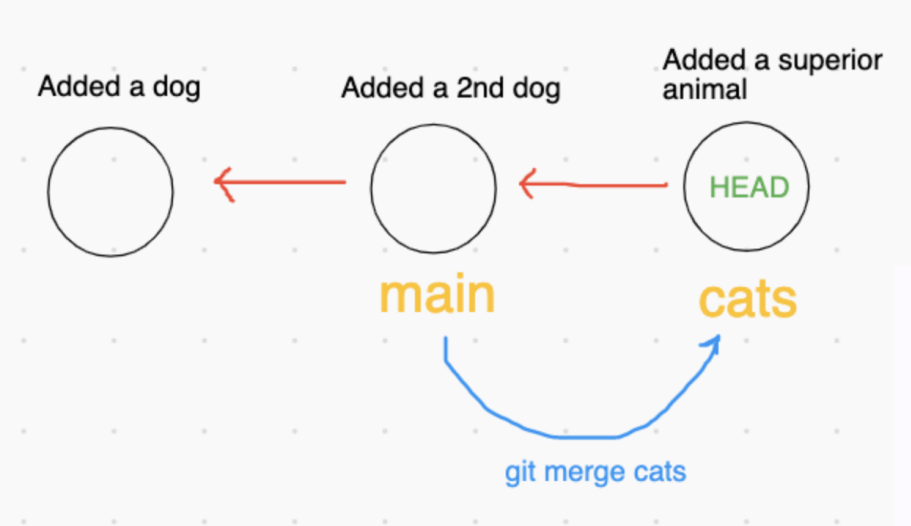
```
git checkout master
```

The output of `git log` should no longer have the "Added a superior animal" commit. We made that commit while on the "cats" branch, so it is not visible on master.

j. Use `git merge` to merge "cats" into "master."

```
git merge cats
```

Here's a diagram that describes the merge we performed:



6. **Comparing versions.**
   a. In the command `git diff HEAD~1 HEAD`, what do `HEAD` and `HEAD~1` refer to?
   `HEAD` **points to the most recent commit in your current branch.** `HEAD~1`**, which you can think of as "**`HEAD` **minus 1" points to the parent of** `HEAD`**, or the second-most recent commit in your current branch.**

   b. Run the command on your "animals" repository. What do you see?
   The command should show the "difference" between the `HEAD` **and** `HEAD~1` **commits, that is, everything that happened between them. In this case, you should see that the line "Git me a cat!" was added.**

c. Can you use `git diff` to output the same difference, but by using commit hashes instead of `HEAD` and `HEAD~1`?
Using `git log`, you can get the value of the commit hashes (a.k.a. commit IDs) of the `HEAD` and `HEAD~1` commits. Then, simply replace `HEAD` and `HEAD~1` in the command with their respective commit hashes.

d. *Homework hint:* You can also use `git diff` to compare tags, which you will have to do in Assignment 4!

Discussion Questions

7. Why do you think Git has the concept of a staging area? When would you use it?
Git's staging area allows you to be selective about what code you submit, which allows for you to ensure your commits only change one thing at a time. This Stack Overflow answer goes into more depth.

8. Many different people and companies have differing opinions on how to use Git.
    a. How to write a commit message
    b. Originate Git Guide
    c. Git Flow
    d. GitFlow considered harmful
    e. GitHub Flow
    f. As you work on your project, what Git practices do you think it makes sense to follow as a team?
    This answer really depends on the team. As we'll be using GitHub, a general workflow that makes sense to divide work into small features that can be completed by one person, and have a single branch correspond to each feature. Branches should be merged with pull requests. Commits should have descriptive messages. Ideally, you should be reviewing code from your teammates before features are merged in!

9. If you would like practice for exams, try #9 on the practice midterm.

Recommended Reading

- *Astrolabe v. Olson*, in which Dr. Eggert was named a defendant for possible copyright infringement for tzdb, an open-source project he maintains. This is an inspiration for a part of the upcoming Assignment 4.
- *Pro Git*, a comprehensive book of all things Git-related.