Alice, Bob, and Git

In this assignment, you and a partner will learn to use many common Git commands for creating a repository, commiting changes, branching, and merging.

1 Read

In this lab, you will pair with another student of your choosing, working at adjacent computers in the classroom. You may use either:

1. a lab machine, or
2. your laptop *if* you already have Git installed on it.

Consistent with class discussions, and to simplify this write-up, I will refer to you and your partner as Alice and Bob. You may choose who will play the part of Alice and who of Bob. If there are an odd number of students in the class, there may be *one* group of three students. In this case, there should *one* Alice and two Bobs.

You should work through *all* the lab instructions in Section 5 *together*. Alice should observe Bob's work and vice versa.

This write-up assumes that you are working with Git on the command line. For Mac and Linux, simply open a terminal window and type commands at your standard shell (e.g., `bash`). For Windows, use the Git bash shell.

2 Set Up

You and your partner will both need your own account on GitHub. If you already have an account, you may use it for this lab. If you do not have an account, create one as follows:

1. Navigate to the GitHub home page at `https://github.com/`.
2. Enter a user name, email address, and password.
3. Click Sign up for GitHub.

3 Learn

Although not emphasized in class, these commands are very helpful whenever you're using Git.

1. The `git status` command reports on the current state of your working directory, staging area, and repository. Throughout this lab, I recommend that you run `git status` before and after each command and take time to analyze and understand its output. What is `git status` telling you about your working directory? Staging area? Local repository?

2. The `git log` command returns a list of all the commits stored in your repo in reverse chronological order. To better understand how Git manages commits, run `git log` whenever you use related commands (e.g., `git commit`, `git fetch`, `git pull`).

   Git's `log` command has a large number of options (try `git help log` to see them all). One combination that I use often is:

   ```
   git log --oneline --decorate --graph --color
   ```

   These *flags* (options) to the `log` command ask for:

   - A one-line version of the commit message (`oneline`)
   - Additional decorations related to tags and heads (`decorate`)
   - A simple ASCII graph showing branches in the repo (`graph`)
   - Pretty colors (`color`)

4 Remember

Here's a quick reminder of the syntax of the Git commands you'll use in this lab. Following Unix conventions for documentation:

- Angle brackets (`<file>`) indicate a placeholder for an actual value
- An ellipsis (...) shows repetition
- Square brackets indicate optional arguments

`git status` prints status information

`git log [--oneline] [--decorate] [--graph] [--color]` prints commit history. The optional arguments can be used individually or together; this set is helpful when working with branches.

`git add <file> [<file> ...]` adds one or more files to the staging area

`git commit [-a] -m'Commit message'` commits files added to the staging area to the local repository; with the `-a` flag, stages any changed files *already* under version control. The `-a` flag allows you to skip `git add` for files already under version control. New files must still be added with `git add`.

`git push` replicates changes from the *local* repo to a *remote* repo (e.g., a shared repo on GitHub).

`git fetch` replicates changes from a *remote* repo (e.g., GitHub) to the *local* repo

`git merge` merges changes from the local repo into the working directory; may result in merge conflicts

`git pull` is a convenience command that is essentially `git fetch` followed by `git merge`

5 Collaborate

Don your Alice and Bob personas and follow these instructions.

5.1 Create Shared Repo

You will collaborate with one another using a shared repo on GitHub.

1. Alice: Log on to GitHub and create a new repo.
   (a) At the top of your list of repositories, click New. GitHub will display a setup page for your new repo.
   (b) Give your new repo a name. In this write-up, I'll refer to it as `alice-bob-repo`, but you are welcome to give it any valid name.
   (c) Be sure the repo is marked as *public* so that Bob can share it.
   (d) Click Create repository. GitHub will display instructions about how to start working with your shiny new repo. The top section, entitled "Quick Setup. . . ," shows the URL for your new repo. Note this URL for use in Section 5.2.
2. Alice: You've set up a public repo, but by default, other GitHub users can only *read* such a repo. Bob, however, will also have to *write* to the repo in order to share work. Grant Bob write permission as follows:
   (a) From your repo's GitHub page, choose the Settings tab.
   (b) In the sidebar on the left, choose Collaborators.
   (c) Using the "Search by. . . " box, search for Bob's GitHub user name.
   (d) Click Add collaborator
3. Bob: When Alice added you as a collaborator, GitHub sent you an email invitation to collaborate. To respond to the invitation:
   (a) Go to the email account associated with your GitHub user name
   (b) Follow the instructions in the email from GitHub to gain access to the shared repo that Alice created.

   You should now have full access to the shared GitHub repo.

5.2 Clone Repo

1. Alice and Bob: On your *local* workstation or laptop, change to a directory of your choice and clone the repo from Section 5.1; for example:

```
git clone https://github.com/<your github id>/alice-bob-repo.git
```

5.3 Add a `README`

A repo typically has a `README` file that describes its purpose and content.

1. Alice: In the top-level of your working directory, create a new file called `README.md`. The `.md` stands for *Markdown*, a simple textual format for marking up a file; see the main Markdown web site as well as the details of GitHub-flavored Markdown. Add a few relevant words to the `README.md` file.

   Share your blockbuster `README` file with Bob:

   (a) Add `README.md` to your staging area (`git add`)
   (b) Commit your changes to your local repo (`git commit`)
   (c) Push your updated repo to GitHub (`git push`)

2. Bob: *After* Alice completes the previous step, fetch Alice's update from GitHub to your local repo (`git pull`). Verify that the `README.md` file you downloaded contains the content that Alice created.

5.4 Add Files

Although we normally use Git for managing revisions of software projects, we can use it to manage pretty much any file. We'll use two poems from Lewis Carroll (1832–1898; Oxford mathematician and author of *Alice's Adventures in Wonderland* and *Through the Looking-Glass*).

1. Alice: Download `jabberwocky.txt` from the course web site into the top-level directory of your cloned repo.
2. Bob: Download the text for *The Walrus and the Carpenter*, found on the lab web page as `walrus-carpenter.txt`, into the top-level directory of your cloned repo.
3. Alice and Bob: Add your poem file to your staging area (`git add`) and then commit it to your local repo (`git commit`).
4. Alice: Push your changes to your shared repo on GitHub (`git push`).
5. Bob: *After* Alice completes the previous step, try to push your changes to GitHub (`git push`). Your `push` should *fail* because the shared repo has changed since your last pull (Alice just added a file to the GitHub repo).
6. Bob: Update your local repo from GitHub (`git pull`). This operation brings your local repo up to date with the shared one. Check the files in your working directory, which should now include the `jabberwocky.txt` file that Alice recently pushed.
7. Bob: Try again to push your changes to the GitHub shared repo (`git push`). Your `push` should succeed this time because the shared repo on GitHub has not changed since you pulled from it in step 6.
8. Alice: Refresh your local repo and working directory from GitHub (`git pull`). Verify that the file Bob added (`walrus-carpenter.txt`) is now in your working directory.
9. Alice and Bob: Run the `git log` command with the flags mentioned in Section 3. You should have identical copies of the shared GitHub repo in your local repos.

5.5 Branch

Recall that a *branch* allows for a divergent line of development within a project. Working on a branch, you and your group can operate independently of the rest of your team. You won't "step on their toes" with your own work.

Eventually, you'll be ready to make available to the rest of your team the feature, defect repair, documentation, or other deliverable on which you're working. At their discretion, other members of your team can *merge* your branch into their repos.

(Don't forget about the `git status` and `git log` commands. It will be helpful to run these periodically while you're working on branches to help you understand what Git is doing. The `git log` command with the flags mentioned in Section 3 is particularly helpful when working with branches.)

1. ALICE: In your repo, create a branch called `alice-work` for your own work (`git branch`).

2. ALICE: So that the changes you're about to make end up on your new branch, check out the branch you just created (`git checkout`). Use the `git status` command to check that you're now on the new branch.

3. ALICE: In your new branch, modify your `jabberwocky.txt` file (reflow the text, change capitalization, add a new stanza, whatever). Commit your changes to your local repo (`git add`, `git commit`).

4. ALICE: Push your changes to the shared GITHUB repo. Because the GITHUB repo doesn't yet know about your new branch, you'll receive a warning message if you just run `git push`. Instead, ask Git to create a corresponding branch on GITHUB by running:

   ```
   git push --set-upstream origin alice-work
   ```

   In the message returned by git, you should see an indication that Git created a new branch called `alice-work`.

5.6 Merge

In Section 5.5, ALICE created a new branch (`alice-work`), modified a file on that branch (`jabberwocky.txt`), and pushed the change to GITHUB. ALICE could be working on a new feature and might have made *many* commits on the `alice-work` branch. Because BOB is still on the `master` branch, ALICE's changes have not impacted BOB. Now BOB will *merge* ALICE's changes from on the `alice-work` branch into the `master` branch.

1. BOB: Retrieve the latest changes from the GITHUB repo to your local repo using `git pull`. You should see a message that you have a new branch called `alice-work`.

2. BOB: You can inspect ALICE's work by checking out the `alice-work` branch:

   ```
   git checkout alice-work
   ```

   You should now have a copy of the update that ALICE made to `jabberwocky.txt`.

3. BOB: Check out the `master` branch (`git checkout`)
4. BOB: Merge ALICE's changes

   ```
   git merge alice-work
   ```

All the work that ALICE has stored on `alice-branch` is available to BOB and the rest of the team on `master`.

6 Submit

Both ALICE and BOB should do the following:

1. If you have made any changes in your local repository that you have not committed and pushed to GITHUB, make sure you do so (`git add`, `git commit`, `git push`).
2. On the submission page for the assignment on the course web site, enter the URL of your GITHUB repository.