Metis Git Cheat Sheet

First time setup

Note that the steps in first time setup are only meant to be performed once.

For these steps, you will need to keep two repos separate. The first is the **Metis repo** which has a url that looks like `https://github.com/thisismetis/chi20_ds13`. The other is your fork, which has a url that looks like `https://github.com/gh_username/chi20_ds13`, but with your username in place of `gh_username`.

1. **Make a fork of your class repo**. Go to the web page for your class repo (example: https://github.com/thisismetis/chi20_ds13) and select the "fork" button in the top right.
2. **Clone your fork**. Using the url from your new fork, clone by running

```
git clone https://github.com/gh_username/chi20_ds13
```

3. **Add the metis repo as a remote**. First change directory into your cohort repo by running `cd ch20_ds13`. Then add the remote with

```
git remote add upstream
↪  https://github.com/thisismetis/chi20_ds13
```

You can now verify that your repo is set up by running `git remote -v`. This should confirm that you have two remotes, `upstream` which points to the Metis repo and `origin` which points to your fork.

Update your repo

1. Change to the master branch: `git checkout master`
2. Fetch and then merge changes from the Metis repo: `git pull upstream master`.

Save your work

1. (Optional) Create a new branch: `git checkout -b mybranch`
2. Add modified files to the workspace: `git add path/to/file`.
   - There are shortcuts that you may see, such as `git add .`, that allow you to "just add all changed files". This is not good practice, though, because it will track any unintended changes to files and eventually cause merge conflicts.
3. Describe your changes and commit them: `git commit -m "commit message"`

Submitting your work

First, create a submission branch based on the metis repo

- Fetch the current state of the upstream branch `git fetch upstream master` (this will not affect any current files).
- Create a submission branch based on the upstream master branch `git branch your_branch_name upstream/master --no-track`.
- Switch to that branch with `git checkout your_branch_name`.

Then, submit your work through a Pull Request

- Add the quiz file you just completed with `git add filepath/filename.extension`.
- Write a commit message `git commit -m "your commit message"`.
- Push the commit to your fork `git push origin your_branch_name`.
- Create a pull request with `hub pull-request`

Things to remember

- Git is a command line application. The commands in monospace font, like `git status` are meant to be run in a terminal program.
- You must be inside the repo in order to run git commands. See where you are with `pwd` and change directories with `cd`.
- `git status` is your friend. Use this to check which branch you're on, the current state of your files, and other info.
- Git can be intimidating and that's okay. *Don't let that stop you from reading the responses.* Often, the answer to git confusion is something the git tool has already told you. If I try to `git pull` and it fails, then I need to read what git says about it failing. If git says:

```
error: Your local changes to the following files would be
↪  overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
```

Then I probably need to commit my changes to `README.md` before trying to pull.

Common git commands

- `git fetch <branch>` download commits from remote. Replace `<branch>` with `--all` to download commits for all branches.
- `git merge <branch>` merge `<branch>` into current branch.
- `git pull <branch>` shortcut for fetch then merge.
- `git add <path>` add files or folder to staging before a commit.
- `git commit -m <message>` save staged changes in a commit.
- `git push` upload commits to remote.
- `git log` print recent commits on current branch.
- `git checkout <branch>` switch to `<branch>`.

Dealing with a merge conflict

Merge conflicts happen when you have changed a file locally that has also been changed on the remote repository. Below is an example of a merge conflict:

```
git pull
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

After this, `git status` will show you that you have "unmerged paths".
There are two ways to fix merge conflicts.

1. **Manually**: To manually fix a merge conflict, open the file in a text editor and find the conflict markers. This shows the two versions of the code in conflict

```
<<<<<<< HEAD
# Test!
=======
# Test!! Test!!
>>>>>>> c9690a940ddb4b86914527741c467c3855f45e46
```

Choose which version of the code you want to keep and then delete the lines that contain markers (`<<<<<<<` , `=======` , or `>>>>>>>`)

2. **Using a Merge Tool** Many modern text editors have merge tools built in to make this process easier. There's no "best" option but we recommend you start with VS Code.

Additional resources

- Hands-down the best reading material is the Pro Git book and Git SCM reference. Both are free.
- Learn by doing with Learn git branching.