Using Git for your Project

Using git or another type of version control is highly recommended for collaboratively coding your project.

It is also recommended that you write your code mostly in functions put in modules (.py files). You can call these functions from a simple notebook. The reason for this design is that .py files are pretty straightforward to merge whereas .ipynb files are messy to merge.

Set-up

1. The code-keeper sets up a "central" repo on github.
2. Decision Point: direct commits by team or pull requests.
   (a) Direct Commits: each member clones the central repo
   (b) Pull Requests: each member forks the central repo and clones their fork

Working Session

Below is how I work: its a suggestion. As I sit down to do some work on my piece, say the plotting subroutine

1. `git status` First thing I check that I left everything nicely committed. As I do the status command I am looking in particular for any modified files. These need to be committed. You may also see untracked files: should some of those be committed? Also note if you have commits that have not been pushed. If so, you have made changes that your collaborators don't have yet. Run your code to check where you were. Does it work or do you have remaining bugs? (Just note these).
2. Second thing: pull in my collaborators work.
   (a) Direct Commits: `git pull`
   (b) Pull Requests: On github, fetch from upstream, then on your computer `git pull`
3. `git log` Read through the changes your teammates have made.
4. If you had no outstanding pushes, your code will now be identical to the "central" repo. If you have outstanding pushes, it will be the "central" repo + your changes. Run the code to check it works or works as well as it did before you pulled in the changes.
5. Now do your work.
6. `git add`
   `git commit` Commit often with clear commit messages. You future self and your team will appreciate!
7. When you have working code you can push your code to the "central" repo. Do not push code that does not work! For example, if someone else is working on the stability problem, they will not appreciate no plots! Better plots they would like but no plots will make it hard for them to work.

8. Check for/pull in any new work by your collaborators
    (a) Direct Commits: `git pull`
    (b) Pull Requests: On github fetch from upstream `git pull` and then on github, send pull request
9. Push your work
    (a) Direct Commits: `git push`
    (b) Pull Requests: `git push` and then on github, send pull request
10. Do some more work
11. `git add`
    `git commit` Try very very hard not to leave uncommitted code. Commit before you finish your work session.

If you are doing direct commits, working as the code keeper is just like above. If you are doing pull requests, the code keeper will be like direct commits. The code keeper will also need to regularly review and pull in the pull requests from team mates.

My favorite git cheat sheet:
`https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet`