NYU Computation in Economics Traineeship
Session 1 Notes

September 24, 2019

1 Day 1

1.1 Getting started with Git: Workflow

1. Create a GitHub account
2. Choose a repository you want to work on by searching the website. This will be called the "main repository" for the project
3. Fork the main repository from GitHub.com. This creates a "master branch" of your fork in your GitHub.com account
4. Choose a folder in your computer where you want to create a copy of the main repository. A few alternative methods to organise your git repositories :
   - Create one folder where you keep all your git repositories
   - If you have different folders for each project you are working on, within each of these folders, create a "git_github" folder in which you will make a copy of the git repository
5. Clone the fork to your local machine (don't clone the main repository) by copying the url from the Clone/Download button on the GitHub.com page for your fork by typing the following command in your terminal. This will create a copy of the folder on your computer (local master).
   `git clone ‘‘url’’`
6. Add the main repository as a remote repository named "upstream" so that you can update your local master whenever there are changes in the main repository, and you can push changes from your local master to the main repository. Take the url from the Clone/Download button on the GitHub.com page of your main repository.
   `git remote add upstream ‘‘url’’`
7. You can see a list of your remote repositories using:
   `git remote -v`
8. Before making changes to files on your computer or adding new files, create a new branch. This is a local project branch, separate from the local master branch. In general, we want to avoid making changes directly to the local master branch so that the local master branch remains consistent with the main repository which is online
9. Create a new branch:
   `git checkout -b ‘‘new branch name’’`
10. If you want to switch between branches which already exist:
    `git checkout ‘‘existing branch name’’`
11. Make changes or add new files

12. Stage the new and modified files so they can be committed
    `git status` shows you if there are any new or modified files
    `git add ‘‘filename’’` OR `git add -A` (be careful because this uploads all files and you may end up adding files that you don't intend to)

13. Commit the staged files (this saves the changes to your local project branch)
    `git commit -m ‘‘Message’’`
    With every commit, include a 1-2 line message explaining the changes which were made in that commit. Keep it short but descriptive

14. Now create a remote project branch (this is a branch of your fork of the main repo and exists on your GitHub.com account)
    `git push origin ‘‘branch name’’`

15. In order to incorporate the changes you made to the project folder with the main repository so that all collaborators can see these changes, you need to submit a pull request on GitHub.com. Once this pull request is accepted by whoever is managing the main repository, your changes will become part of the main repository

16. Now update your local master to reflect the changes in the main repo
    `git fetch upstream`
    `git merge upstream/master`

17. Update your remote master to reflect changes in the main repo
    `git push origin master`

18. Delete the local and remote project branches that your created because the task is now complete and has been incorporated in the main repo
    `git branch -D ‘‘new branch name’’`
    `git push origin |delete ‘‘new branch name’’`

19. The order in which these changes are made is important.

    - First make changes in your local project branch (on your computer)
    - Push these to a remote project branch (on GitHub.com)
    - Submit a pull request to the main repository (on GitHub.com)
    - After your pull request is accepted and the main repository has been updated, update the local master
    - Update the remote master

2 Day 2

2.1 Updating your local repo to reflect changes in the main repo on GitHub

1. Grab changes in the main repo: `git fetch upstream`
2. Update your local master branch: `git merge upstream/master`
3. Update your remote master branch: `git push origin master`

2.2 Merge conflict examples

1. Suppose you check out a local project branch and make changes to a file, then fetch upstream changes to your master where someone has made changes to the same file, there will be a merge conflict between your local master branch and local project branch
2. Conflicts can arise between any local branches (master vs project branch)
3. When merging from Remote Upstream or Remote Origin branch to your local branch

2.3 Dealing with a merge conflict

1. git reset –hard HEAD: moves your repo back to the last commit you made before the issue arised
2. In what direction should you merge? You want to be in your local project branch and then merge in changes from your local master branch. Because you want your local master to be consistent with the main repo (upstream), so you don't want to mix in changes you made on your project branch until these are approved via pull request to be incorporated in the main repo.
   - `git checkout newbranch`
   - `git merge master`
3. Once there is a merge conflict, open the file on your local machine, git has added a bunch of new things to the file
   - <<<<< HEAD: refers to the branch you are on
   - >>>>> master: refers to the master branch
   - ==========: Separates the stuff that is conflicted between the two versions of the file in the current branch and the master branch.
   - Choose what you want to keep and save the file
   - Add to staging area, commit, and then merge the new branch with the master again
4. Merging conflict with binary files (PDF, Jupyter notebooks): Just have to pick one file or the other (and delete the rejected file), because you can't edit changes within these files

2.4 Work flow

If you have made changes to your local project branch, but meanwhile changes have been made on the main repo that are merged with your local master:

1. Merge your local project branch with your local master
2. Resolve any merge conflicts
3. Submit a pull request to upload your changes to the remote main repo

2.5 Sensitive files

To make sure any of your sensitive data doesn't get uploaded to github in a commit, add the file name of the file to the gitignore file

2.6 Best Practices

1. Stanford Data Management Best Practices Guide
2. File Naming Best Practices
3. Include a readme for your file naming convention
4. Project builder: Keeps a record of which data file and code each figure and table came from, so that when you make a change to the data or code you know what output you need to update

3 Day 3

3.1 Code Lay-out Tips

1. Set up text editor so that tab = 4 spaces
2. Limit each line to 72 characters, makes it easier to view multiple code files side by side without having to scroll.
3. Can set up boundaries in the text editor to mark the 72 character mark
4. Can also set the text editor to automatically wrap to the next line
5. Make sure to indent properly for better readability
6. Blank lines: Include 2 blank lines before and after function definitions. Avoid unnecessary blank lines. Blank lines should have no white spaces
7. Python PEP8 guidelines
8. Set up a PEP8 linter in your text editor which flags parts of your code which are inconsistent with PEP8
9. When making in-line comments, include a space after the symbol which starts the comment
10. Python's Black package, automatically formats your code to be consistent with PEP8

3.2 Documentation Tips

1. In-line comments explaining what a line or section does
2. Documentation for functions that you write will be larger blocks of comments, bracketed by 3 double quotes """..."""
3. Set up a Sphinx website which outlines the documentation for your code
4. When writing up the paper, it's important to include details about the estimation methods, algorithms etc. especially if results are sensitive to the estimation method. Include these details in a technical appendix.
5. Computational economics is a means to an end, the focus is not the methods but the questions we can answer with them

3.3 ParamTools

1. Python package which defines your model's parametersFeed in the type, default value, description, max and min value of each parameter in your model
2. If you need to make changes to your parameters, you can do this easily in one place

3.4 Tax Simulation Model

1. Tax policy simulation model, Evans' tax preferences project: PSLmodels/TaxBrain
2. Hosted on compute studio
3. Change tax and benefit rates, outputs the effect on federal revenue over the next 10 years, and also the effect on the income of people in different income deciles

3.5 Visualization

Open Source Economics Visualizations Gallery. Code for all visualizations available. Uses D3 D3 and python's bokeh library

3.6 Resources for Computational Economics

1. Jupyter notebooks with notes on computational economics and finance: QuantEcon
2. Also see QuantEcon repository on GitHub
3. Consider applying to Open Source Economics Laboratory Bootcamp (applications open in January, will be held over 6 weeks from July to first week of August at Yale)
4. Python resources: Foundation of Applied Mathematics