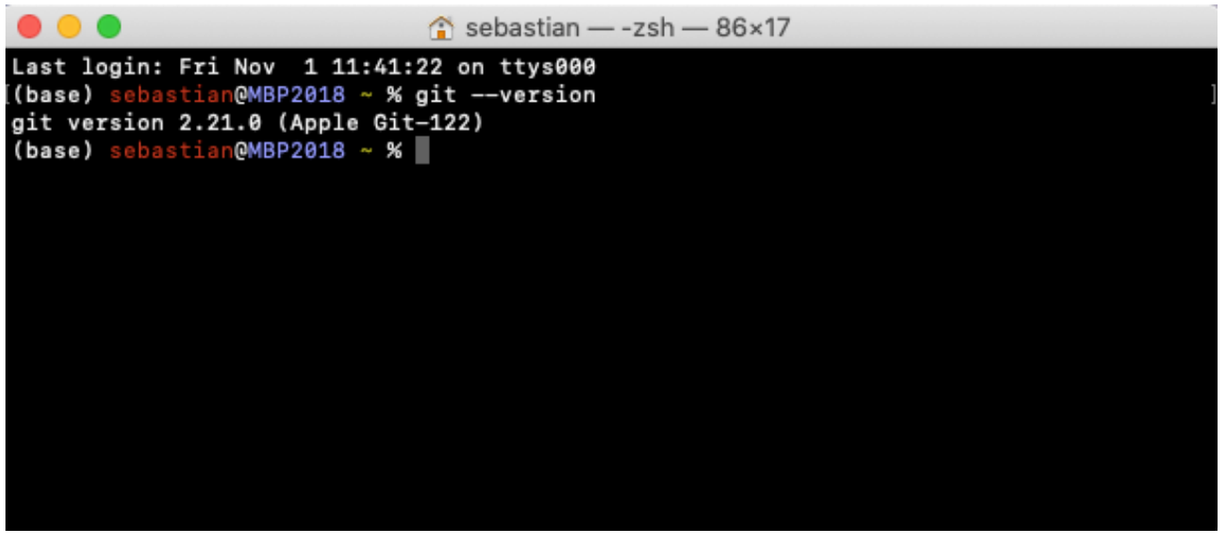
Contributing to Open Source Projects

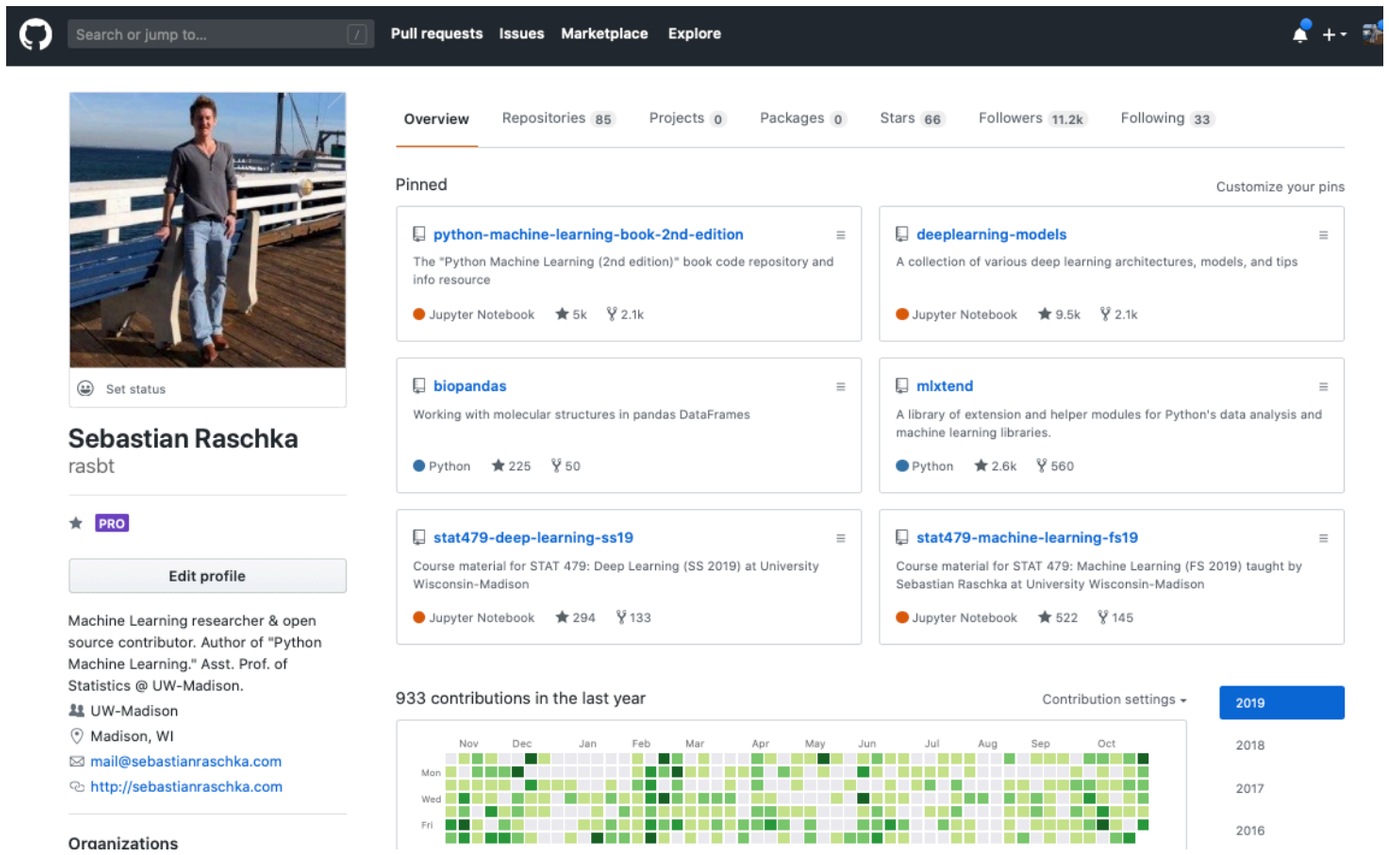
(like scikit-learn, mlxtend, etc.)

First Step: Familiarize yourself with Git & GitHub

Git vs. GitHub

Git is a software tool

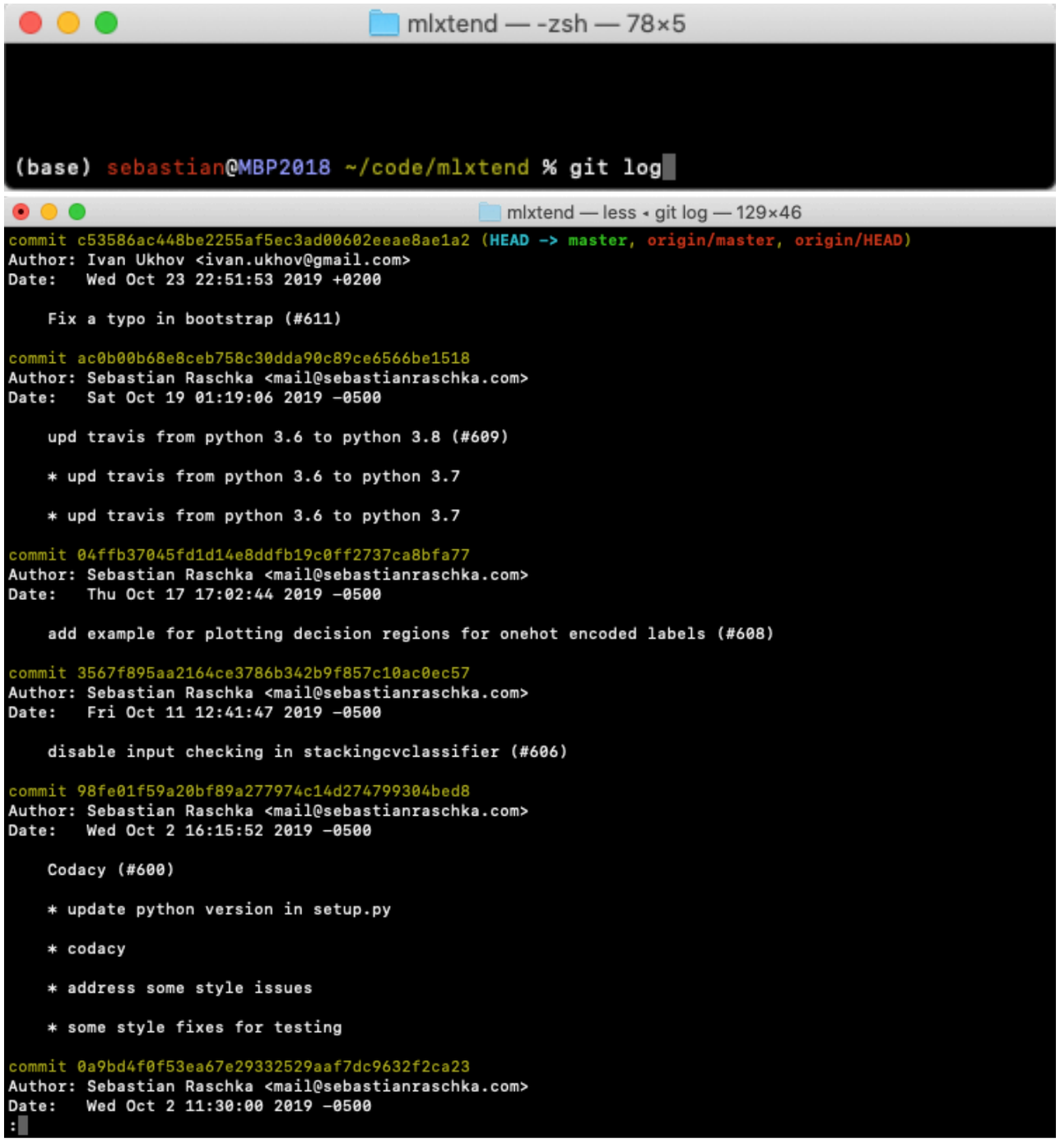




GitHub is a (web) platform for hosting "Git" projects

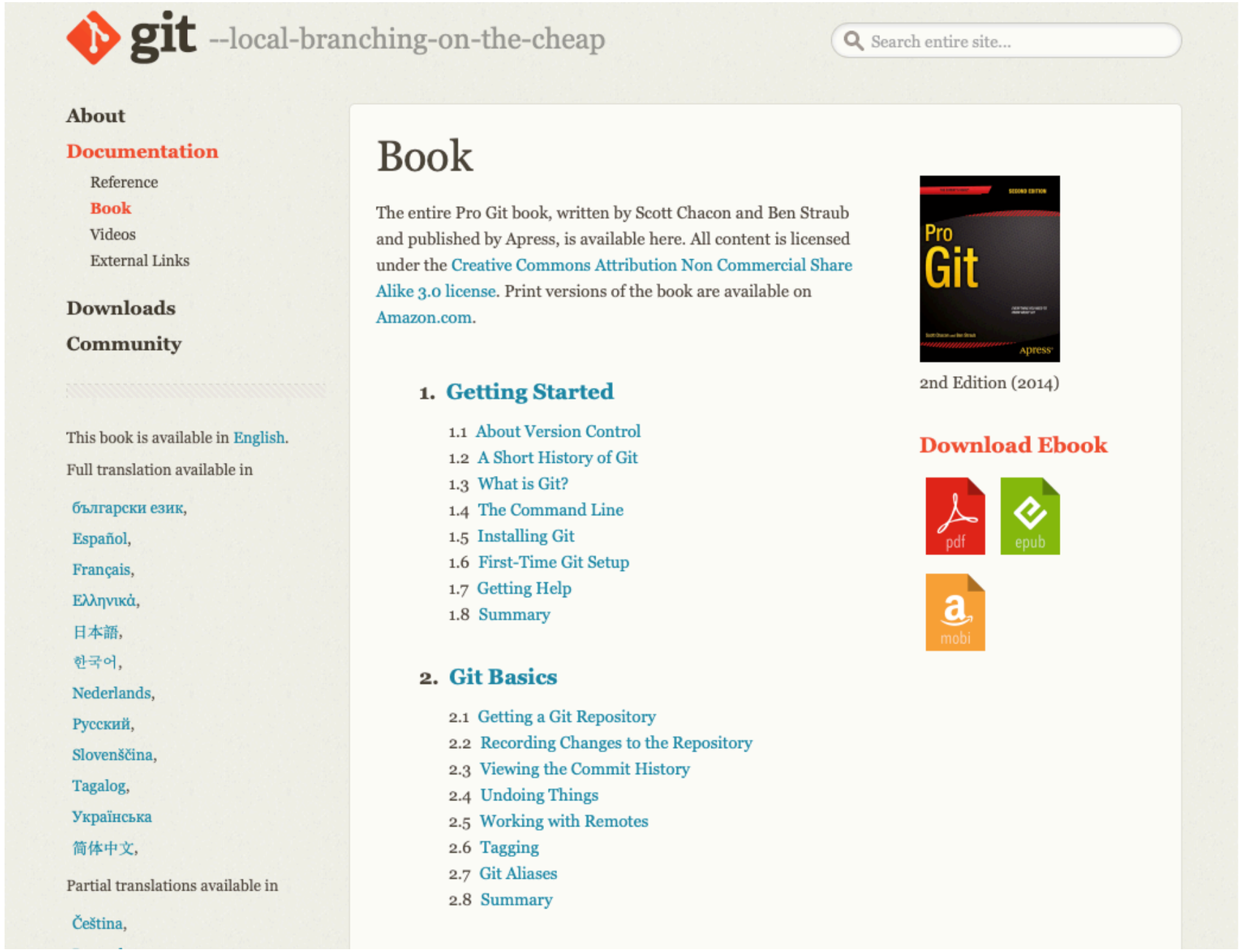
Git

- A version control system for software development
- Let's you keep track of changes in your code base



Git

- Moreover, it's great for collaborative code development
- I.e., it's great for managing software development that involves multiple people
- A great learning resource is the free "Git" book at https://git-scm.com/book/en/v2



Additional Git Resources

1) What is Git | What is GitHub | Git Tutorial | GitHub Tutorial | Devops Tutorial | Edureka
https://www.youtube.com/watch?v=xuB1Id2Wxak

2) Understanding branches in Git
https://blog.thoughtram.io/git/rebase-book/2015/02/10/understanding-branches-in-git.html

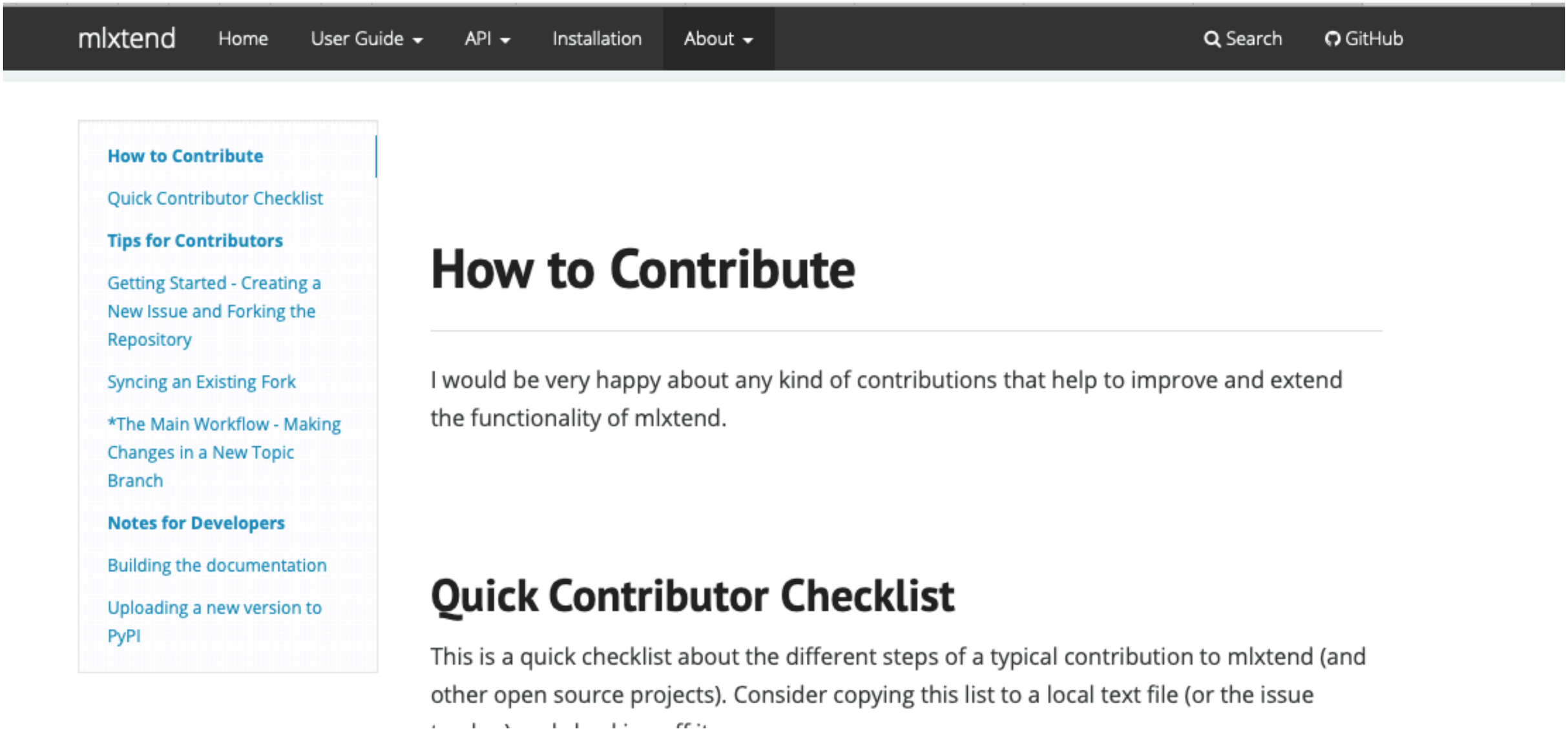
3) .1 Git Branching - What a Branch Is
https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is

4) Forking Workflow
https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

Basic Workflow for (most) Open Source Projects

Step 0: Read the Contributor Guidelines

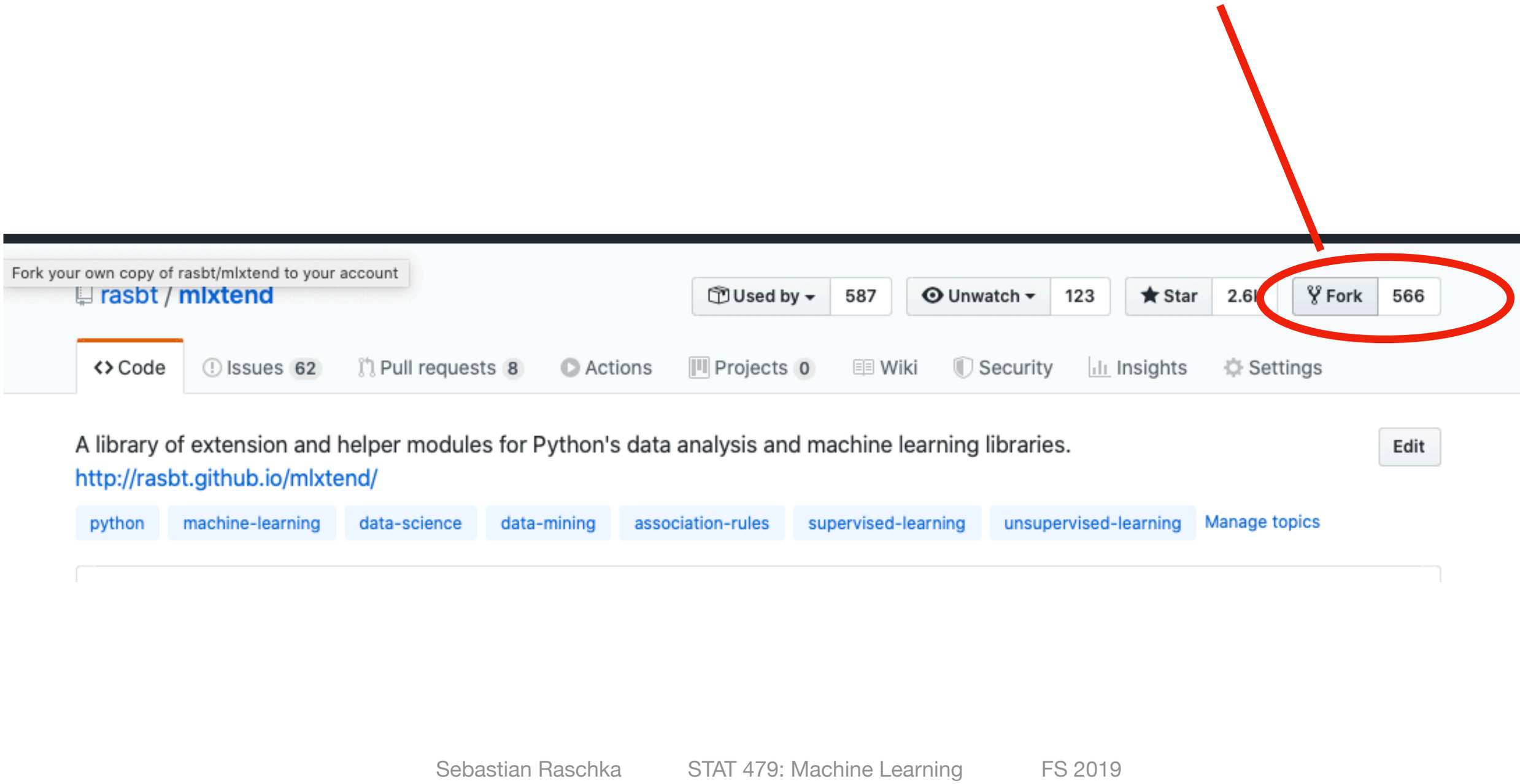
http://rasbt.github.io/mlxtend/CONTRIBUTING/



Basic Workflow for (most) Open Source Projects

Step 1: Fork & Clone the repository

First, you have to fork the repository to your user account on GitHub. For this, go to the main repository website and click on "Fork"

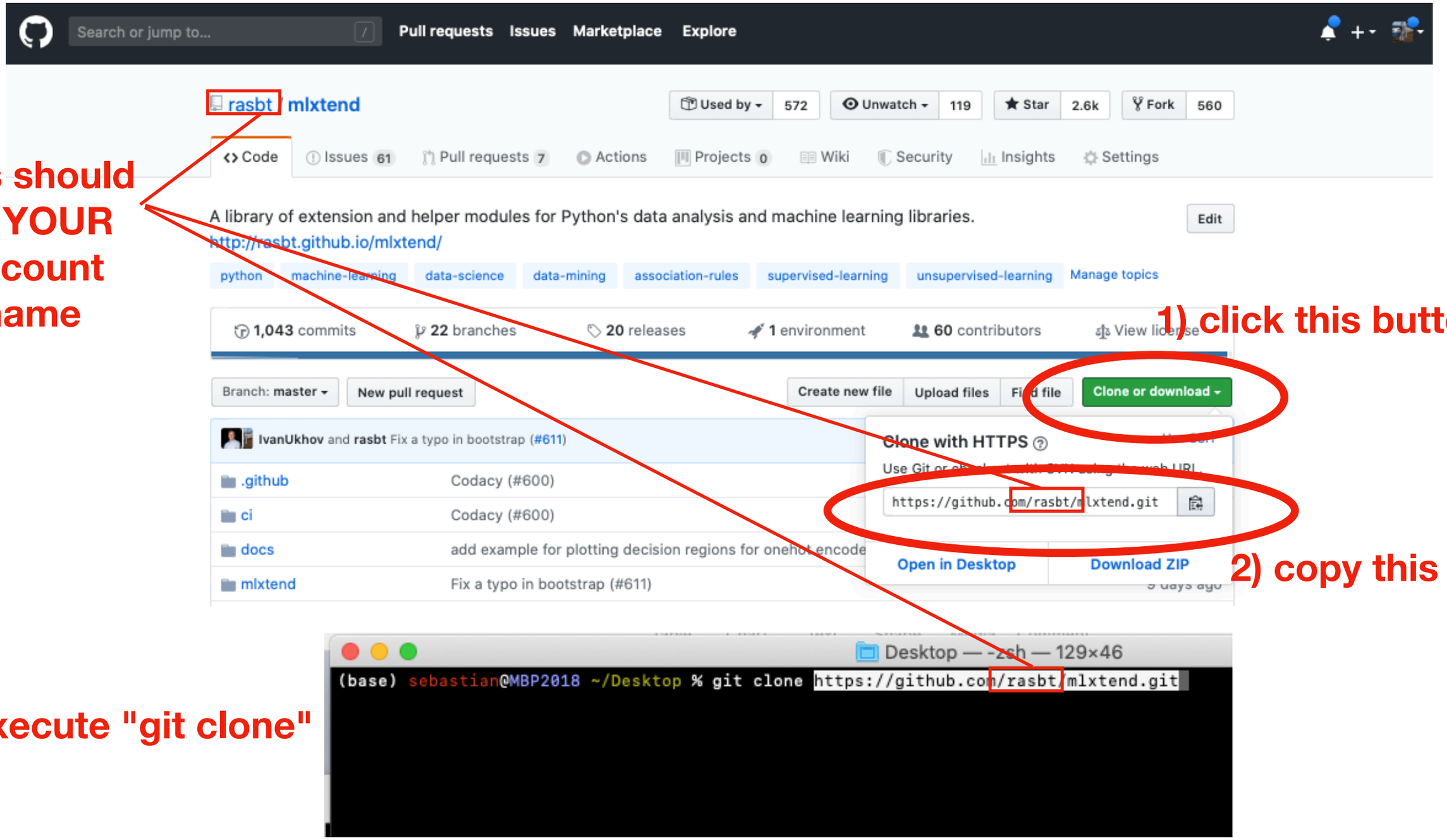


Basic Workflow for (most) Open Source Projects

Step 1: Fork & Clone the repository

Next, go to your GitHub account and open the forked repository there, then ...

This should be YOUR account name

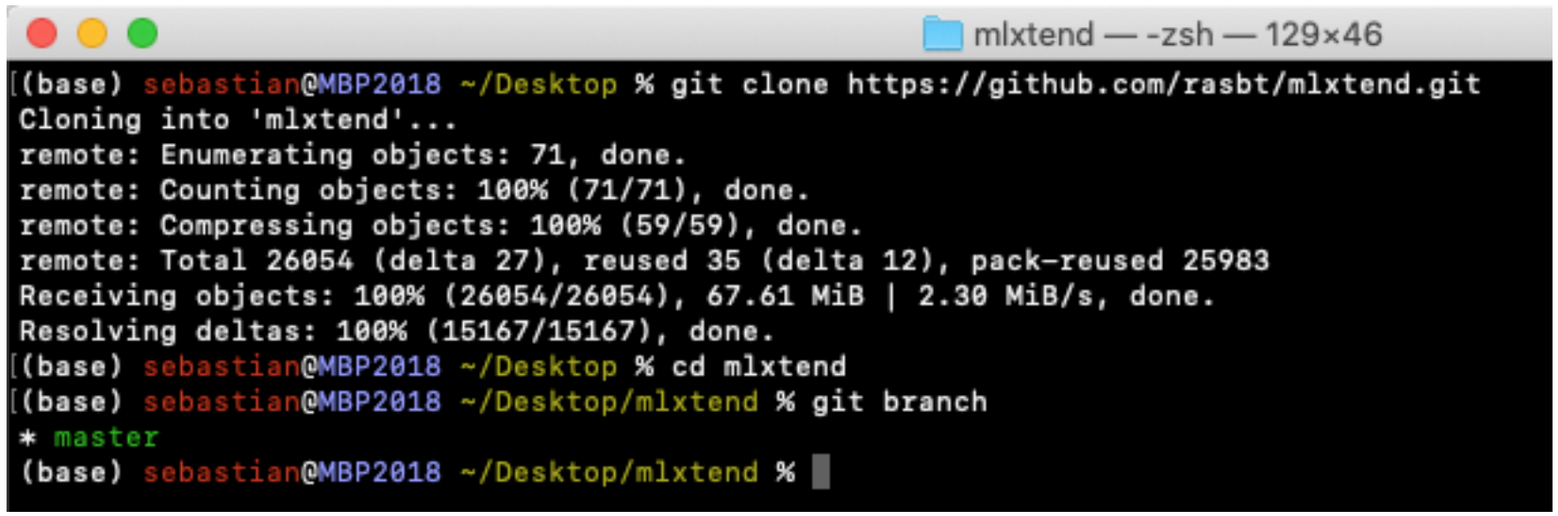


3) execute "git clone"

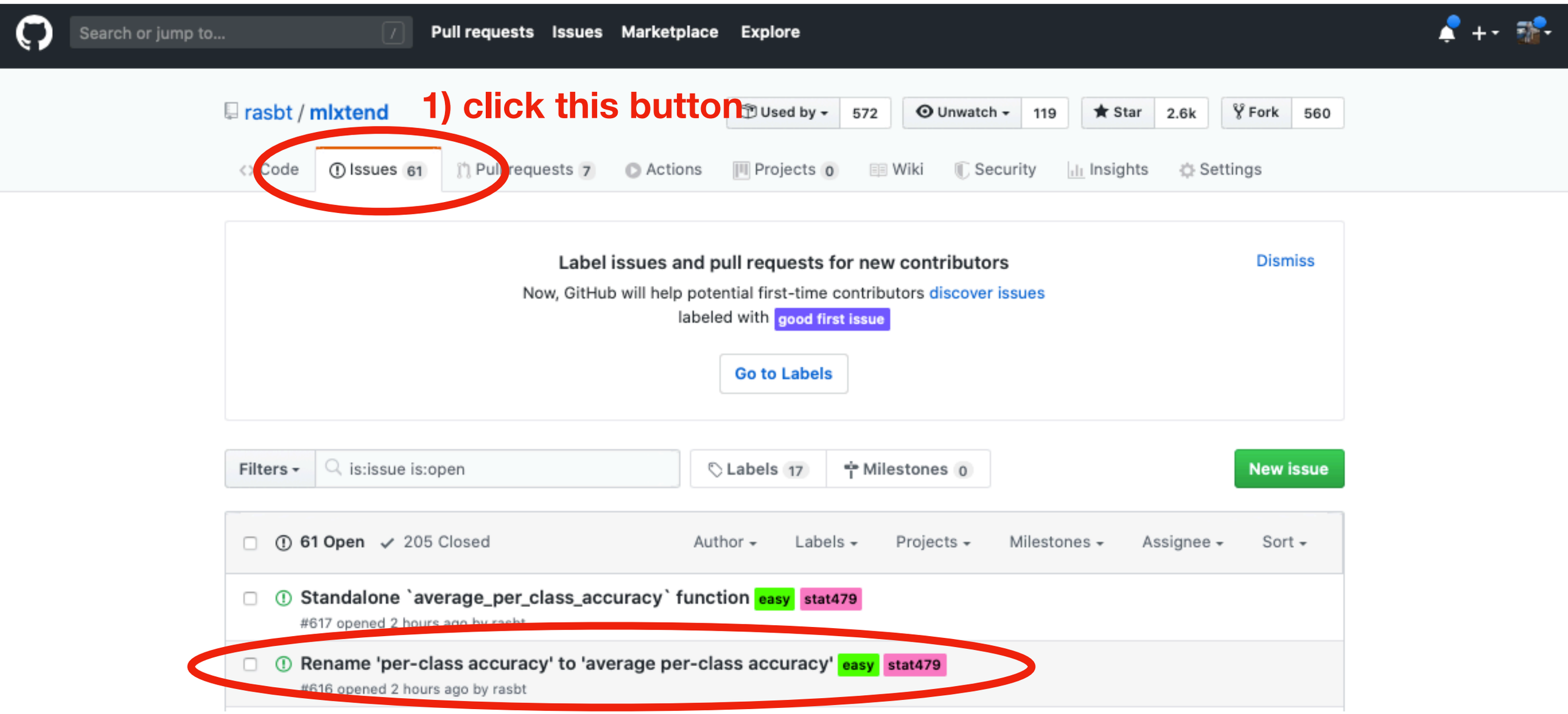
Basic Workflow for (most) Open Source Projects

Step 1: Fork & Clone the repository

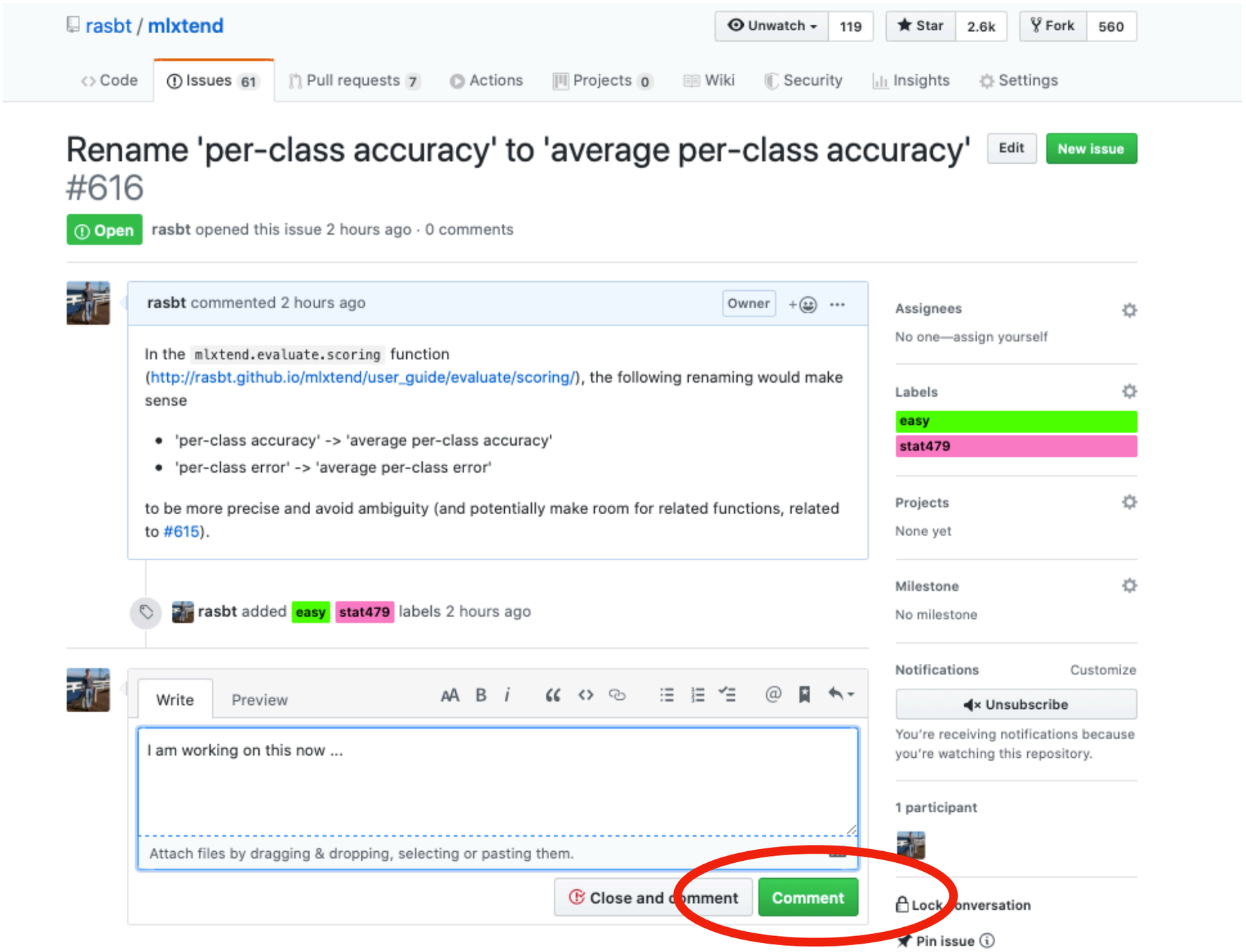
4) Navigate to the cloned directory



Step 2: Find an Issue to Work On



Step 2: Find an Issue to Work On

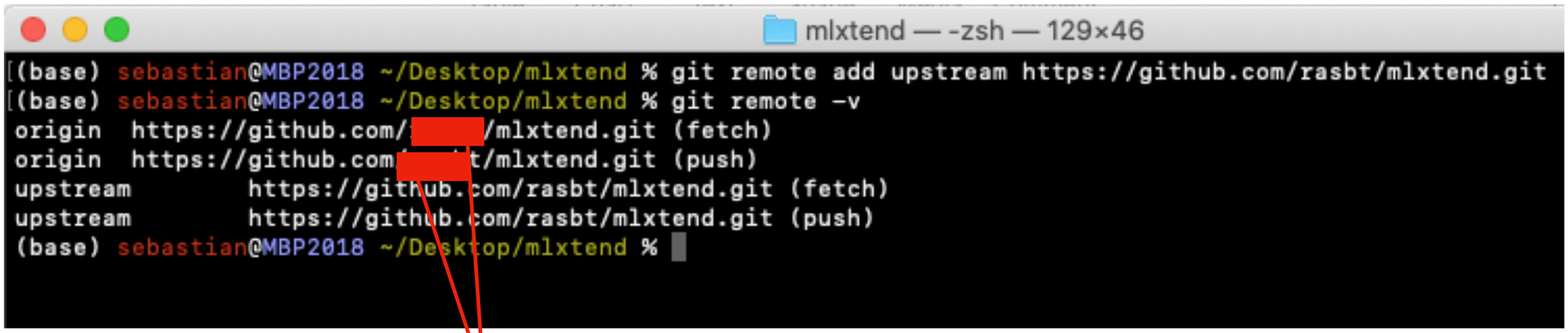


3) add a comment to discuss the plan with the maintainer before working on it

Step 3: Keep Everything in Sync

- **It's a good idea to periodically check that everything in your cloned directory is in sync with the original project on GitHub.**

- **For this, you have to setup an "upstream" link to the original repository (you only have to do it once after you cloned the repository!)**



Step 3: Keep Everything in Sync

1) Before starting working on a new feature, fetch changes from the original repo

```
mlxtend — -zsh — 129×46
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git remote add upstream https://github.com/rasbt/mlxtend.git
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git remote -v
origin  https://github.com/rasbt/mlxtend.git (fetch)
origin  https://github.com/rasbt/mlxtend.git (push)
upstream        https://github.com/rasbt/mlxtend.git (fetch)
upstream        https://github.com/rasbt/mlxtend.git (push)
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git fetch upstream
```

2) Then merge the changes with the master branch

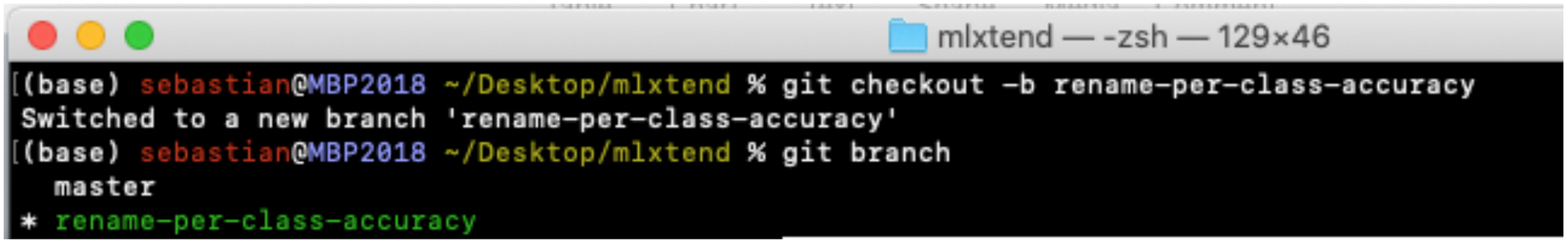
```
mlxtend — -zsh — 129×46
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git remote add upstream https://github.com
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git remote -v
origin  https://github.com/rasbt/mlxtend.git (fetch)
origin  https://github.com/rasbt/mlxtend.git (push)
upstream        https://github.com/rasbt/mlxtend.git (fetch)
upstream        https://github.com/rasbt/mlxtend.git (push)
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git fetch upstream
From https://github.com/rasbt/mlxtend
 * [new branch]        adaline-docs           -> upstream/adaline-docs
 * [new branch]        categorical-decision   -> upstream/categorical-decision
 * [new branch]        codacy                 -> upstream/codacy
 * [new branch]        docfix                 -> upstream/docfix
 * [new branch]        drop-proba             -> upstream/drop-proba
 * [new branch]        fix-feat-imp           -> upstream/fix-feat-imp
 * [new branch]        fixed-features         -> upstream/fixed-features
 * [new branch]        gh-pages               -> upstream/gh-pages
 * [new branch]        grid-support-feat-sele -> upstream/grid-support-feat-sele
 * [new branch]        heatmap                -> upstream/heatmap
 * [new branch]        ldaloadings            -> upstream/ldaloadings
 * [new branch]        master                 -> upstream/master
 * [new branch]        mnist                  -> upstream/mnist
 * [new branch]        pca-ddof               -> upstream/pca-ddof
 * [new branch]        pca-svd                -> upstream/pca-svd
 * [new branch]        py37                   -> upstream/py37
 * [new branch]        readme-links           -> upstream/readme-links
 * [new branch]        sclf-warn              -> upstream/sclf-warn
 * [new branch]        setup                  -> upstream/setup
 * [new branch]        stacking-dtype         -> upstream/stacking-dtype
 * [new branch]        v0.17.0                -> upstream/v0.17.0
 * [new branch]        verbose-apriori        -> upstream/verbose-apriori
(base) sebastian@MBP2018 ~/Desktop/mlxtend % git merge upstream/master
Already up to date.
(base) sebastian@MBP2018 ~/Desktop/mlxtend %
```

if there are any changes listed, i.e., if it does not say "Already up to date," you have then have to also execute

`git merge upstream/master`

Step 4: Make a New Feature Branch

- Now, create a new feature branch, which will be the branch where you will work on the new feature you are planning to implement or issue you are going to fix
- Then, execute "git branch" to ensure that you are on the correct branch now



Step 5: Start Coding

- **You can start working on the feature/issue now**

```
mlxtend — -zsh — 129×46
(base) sebastian@MBP2018 ~/Desktop/mlxtend % ls
LICENSE-BSD3.txt  MANIFEST.in       ci                files.txt         paper.bib         requirements.txt  setup.py
LICENSE-CC-BY.txt README.md         docs              mlxtend           paper.md          setup.cfg
(base) sebastian@MBP2018 ~/Desktop/mlxtend % code mlxtend/evaluate/scoring.py
```

scoring.py

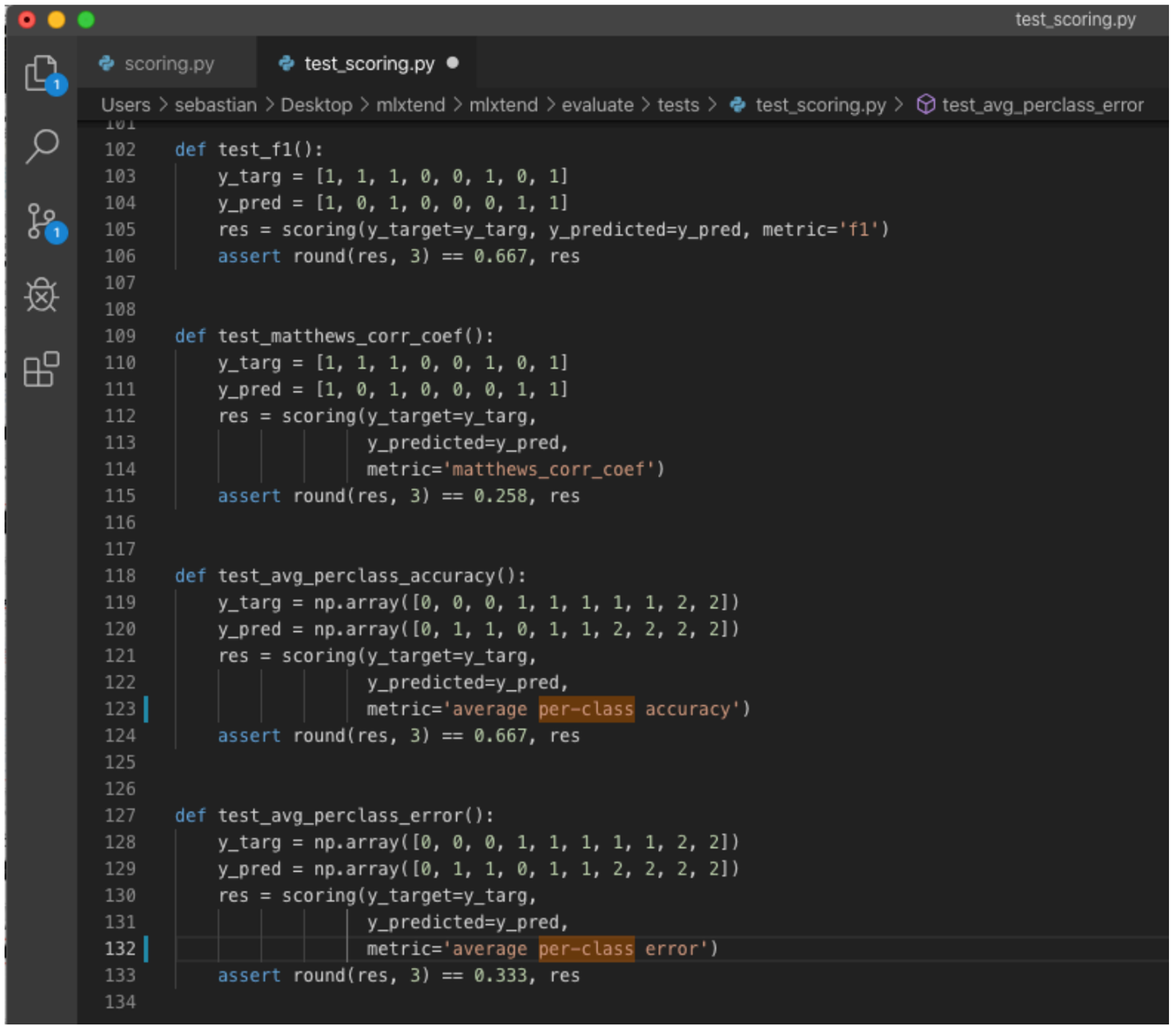
scoring.py

Users > sebastian > Desktop > mlxtend > mlxtend > evaluate > scoring.py > scoring

```
    scores = []
    for l in unique_labels:
        scores.append(func(np.where(true != l, 1, 0),
                           np.where(pred != l, 1, 0)))
    return float(sum(scores)) / len(scores)


def scoring(y_target, y_predicted, metric='error',
            positive_label=1, unique_labels='auto'):
    """Compute a scoring metric for supervised learning.

    Parameters
    ------------
    y_target : array-like, shape=[n_values]
        True class labels or target values.
    y_predicted : array-like, shape=[n_values]
        Predicted class labels or target values.
    metric : str (default: 'error')
        Performance metric:
        'accuracy': (TP + TN)/(FP + FN + TP + TN) = 1-ERR\n
        'average per-class accuracy': Average per-class accuracy\n
        'average per-class error':  Average per-class error\n
```

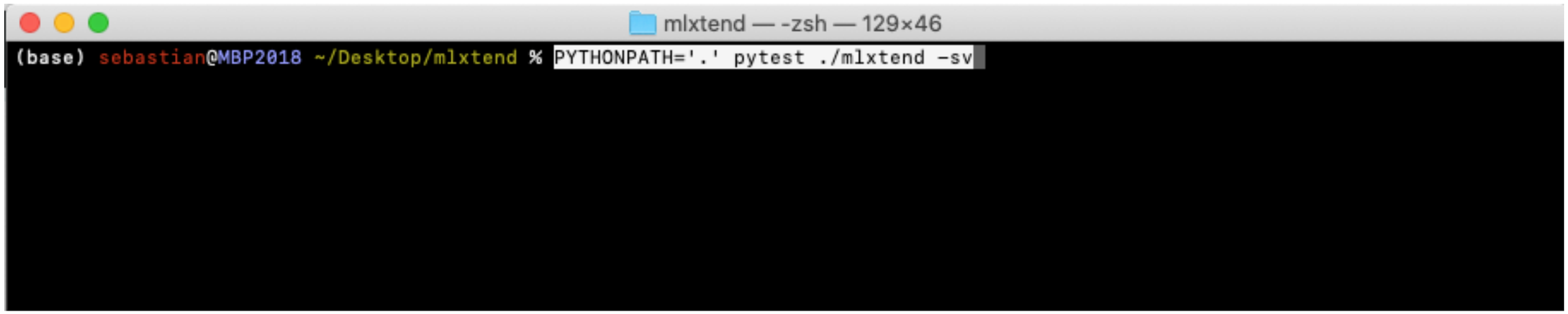
Step 6: Code Testing

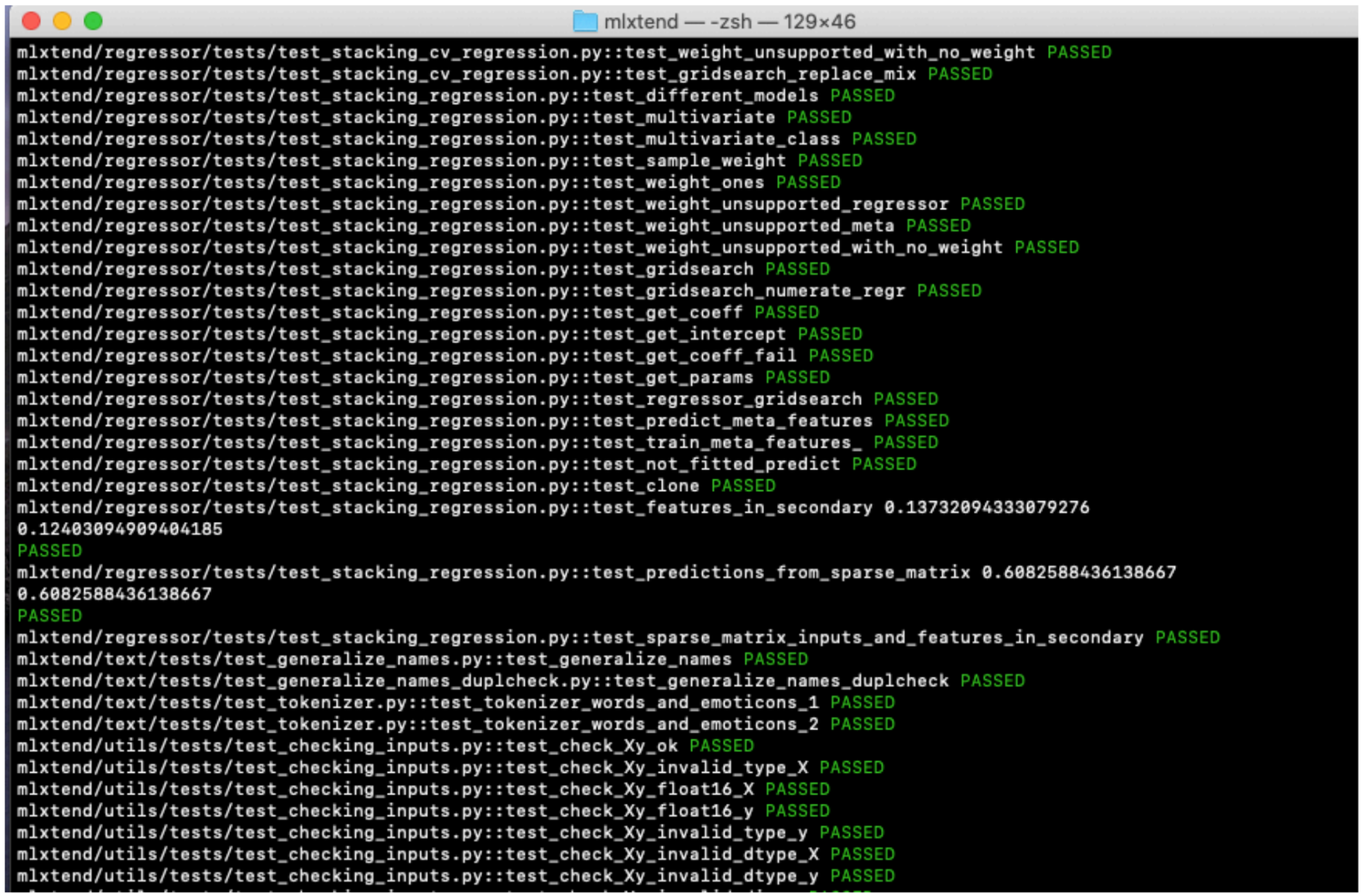
- **After developing the new code, also add or modify the test functions**



Step 6: Code Testing

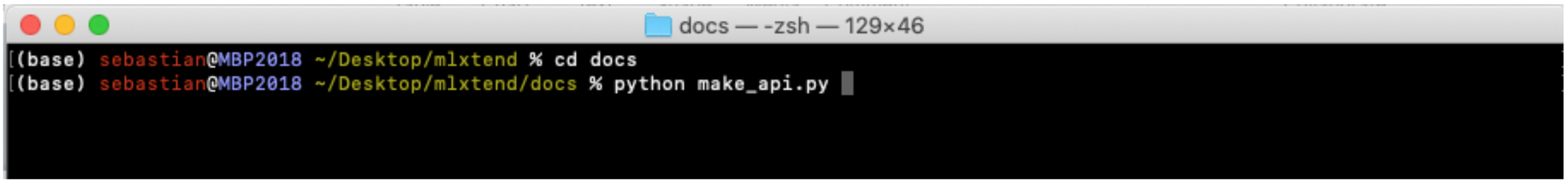
- **Then, test your code locally to make sure that your changes didn't break the existing code base**





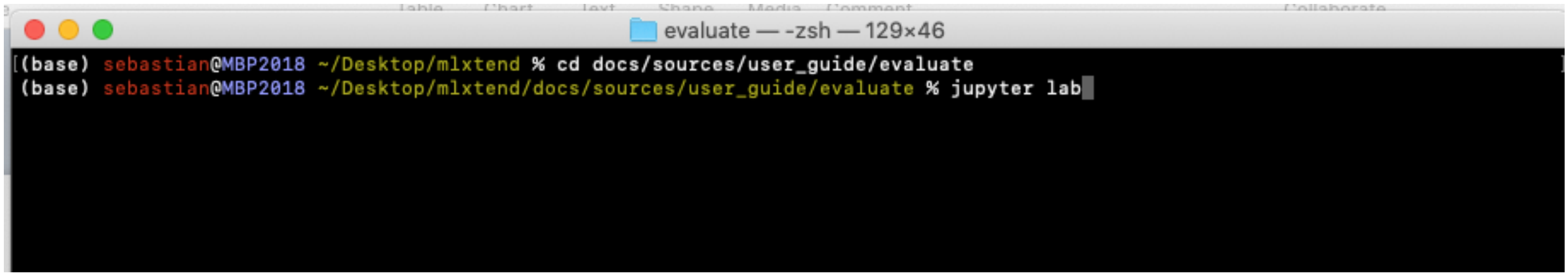
Step 7: Update the Documentation

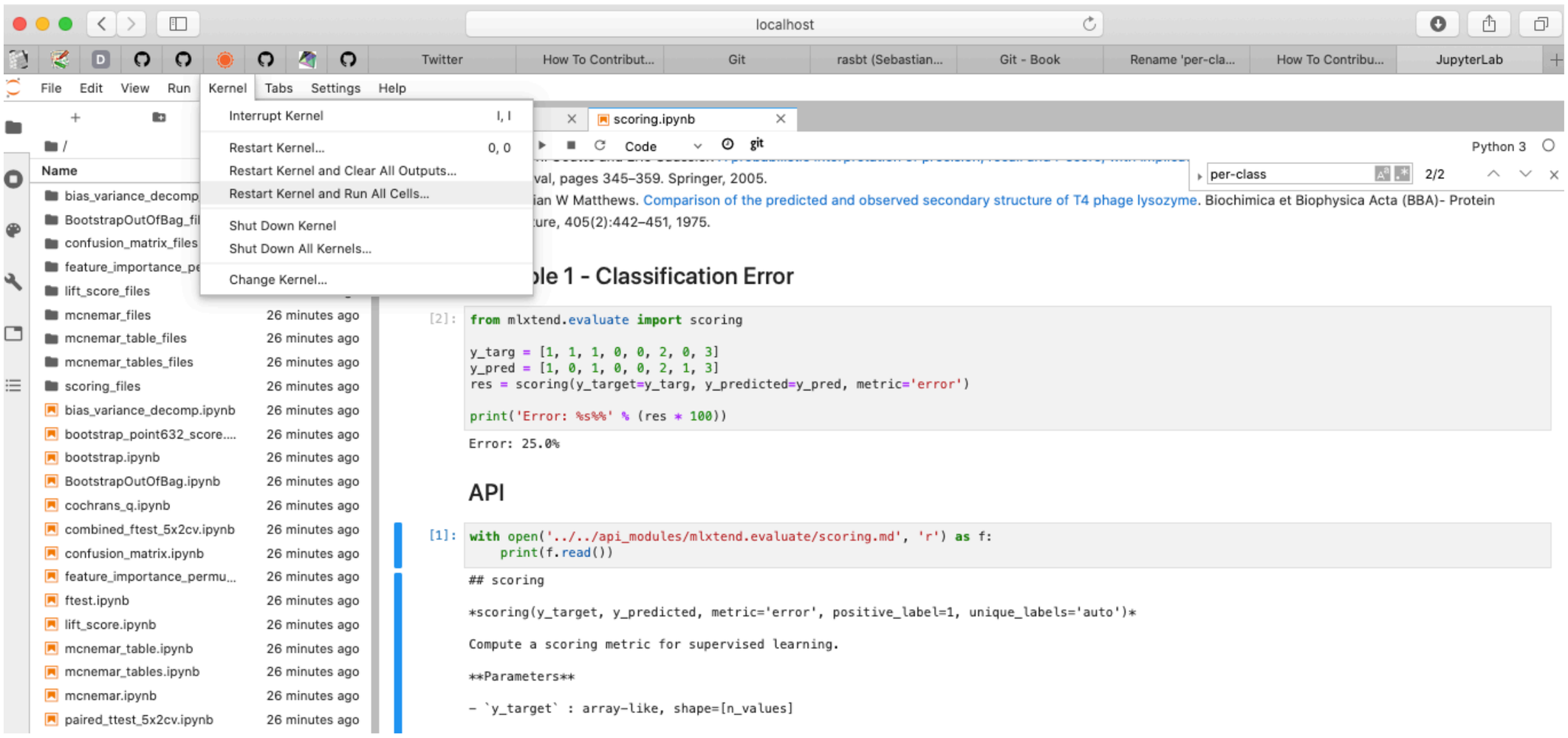
- **Run the python make_api.py file to update the general code documentation (this is mlxtend-specific)**



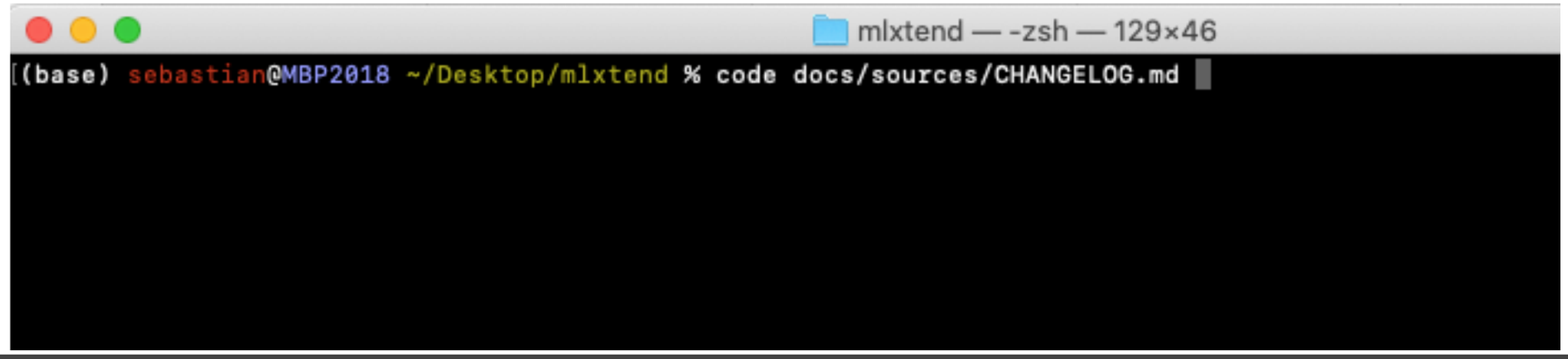
Step 7: Update the Documentation

- **Next, update the documentation on the Jupyter notebook (this is mlxtend-specific)**





Step 8: Make a Changelog Entry



CHANGELOG.md

scoring.py | test_scoring.py | CHANGELOG.md

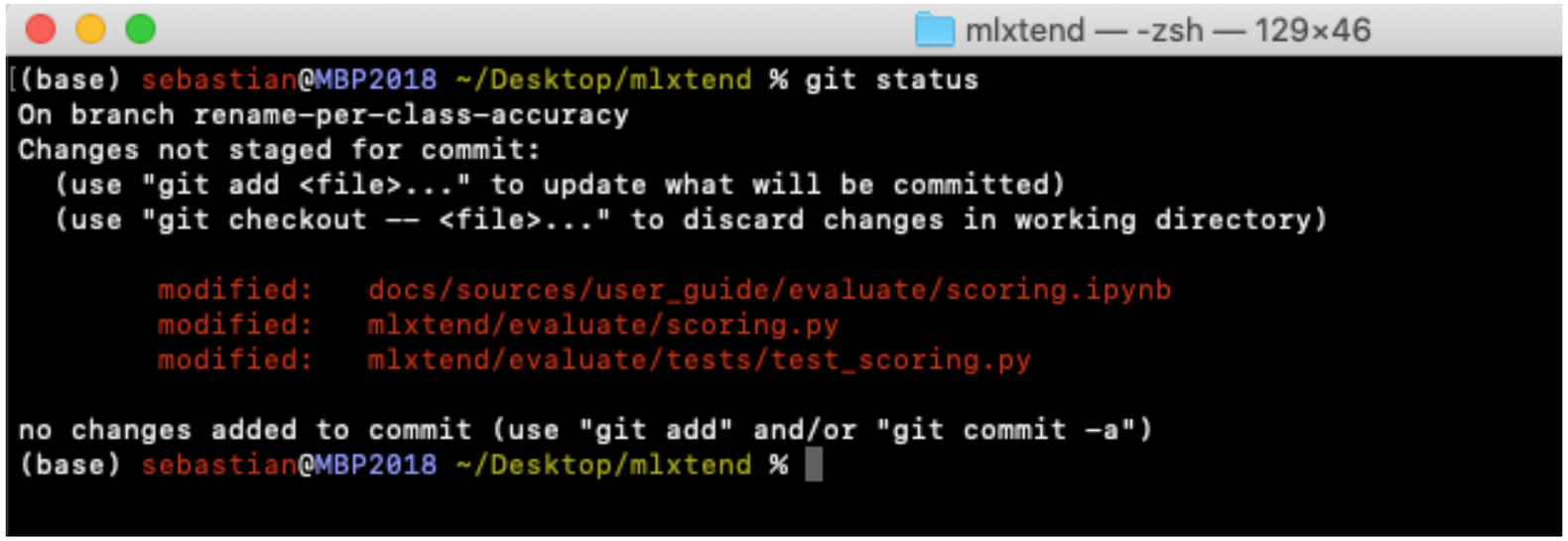
Users > sebastian > Desktop > mlxtend > docs > sources > CHANGELOG.md > # Release Notes > ### Version 0.18.0 (TBD) > ##### Changes

per-class 1 of 4

```
1   # Release Notes
2
3   ---
4
5   The CHANGELOG for the current development version is available at
6   [https://github.com/rasbt/mlxtend/blob/master/docs/sources/CHANGELOG.md](https://github.com/rasbt/mlxtend/blob/master/docs/sources/CHANGELOG.md).
7
8   ---
9
10  ### Version 0.18.0 (TBD)
11
12  ##### Downloads
13
14  - [Source code (zip)](https://github.com/rasbt/mlxtend/archive/v0.18.0.zip)
15
16  - [Source code (tar.gz)](https://github.com/rasbt/mlxtend/archive/v0.18.0.tar.gz)
17
18  ##### New Features
19
20  - The `SequentialFeatureSelector` now supports using pre-specified feature sets via the `fixed_features` parameter. ([#578](https://github.com/rasbt/mlxtend/pull/578))
21
22  ##### Changes
23
24  - Changed `'per-class accuracy'` and `'per-class error'` to `average 'per-class accuracy'` and `average 'per-class error'`, respectively, to improve clarity.
25
```

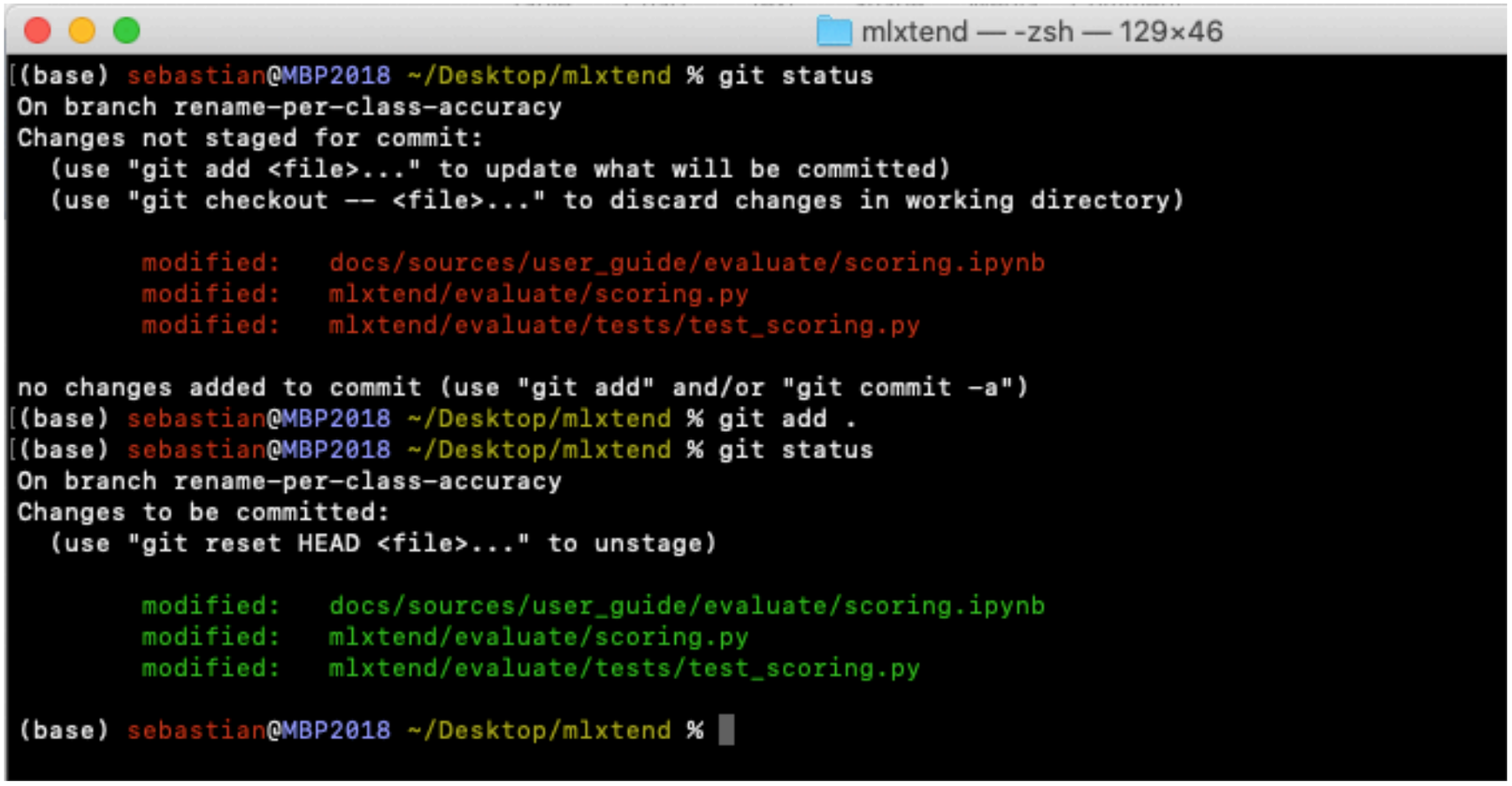
Step 9: Upload your Changes

- Use "git status" to see what files you have changes



Step 9: Upload your Changes

- Then add the files you want to upload via "git add filename"
- If the "git status" list does not include any files that you don't want to upload, you can also execute "git add ." to stage all files for uploading all files



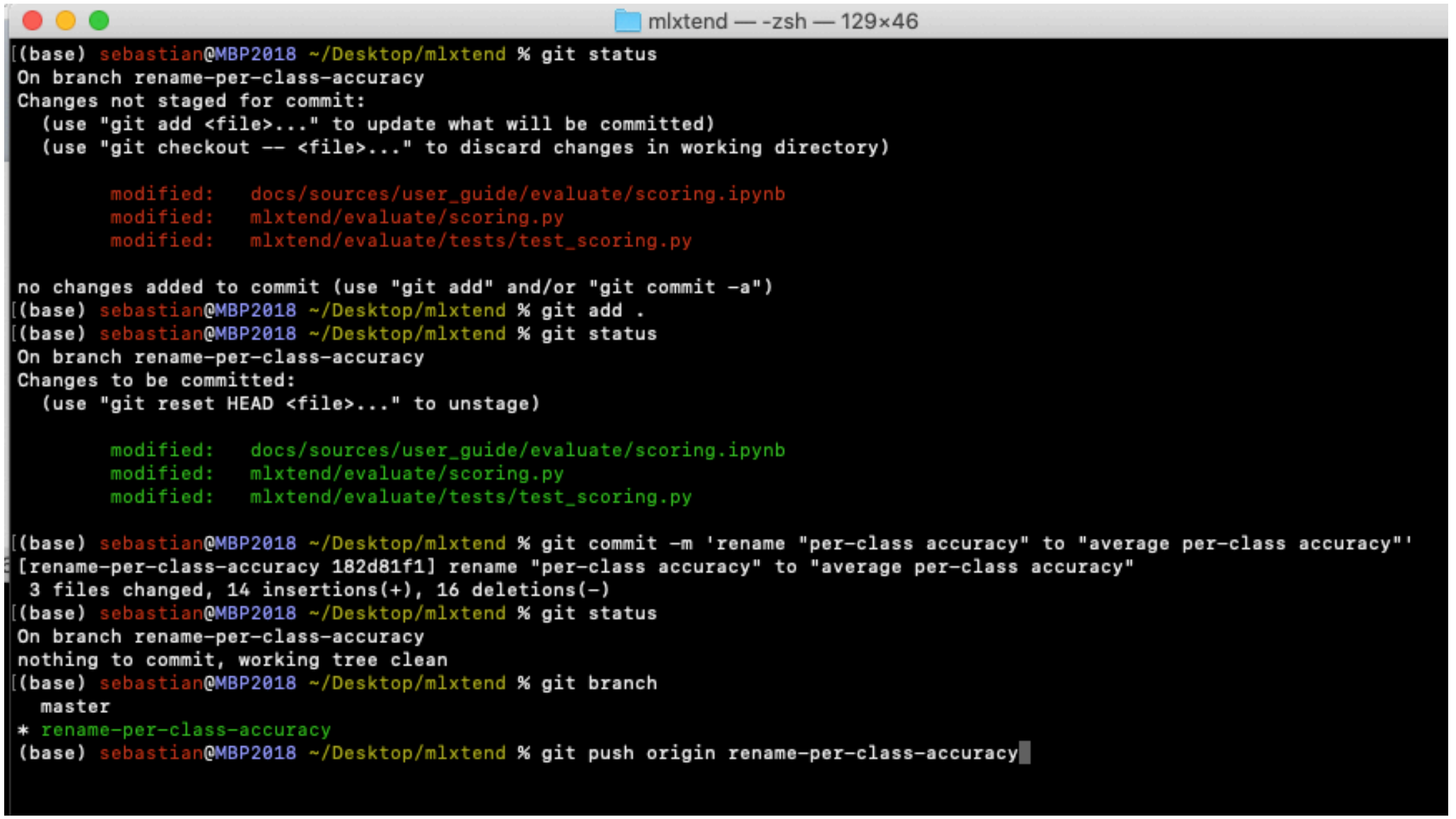
Step 9: Upload your Changes

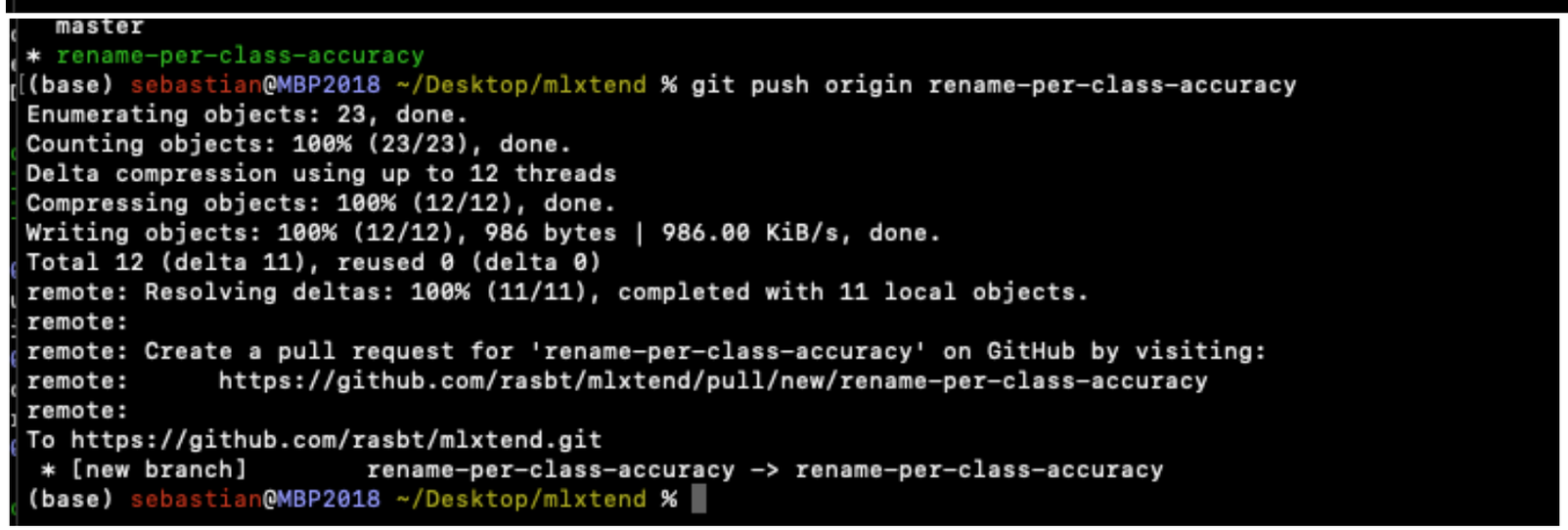
- After marking the files you want to upload via "git add", you can now use "git commit -m 'your message'" to make a entry in the GitHub history

```
mlxtend — -zsh — 129×46
[(base) sebastian@MBP2018 ~/Desktop/mlxtend % git status
On branch rename-per-class-accuracy
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   docs/sources/user_guide/evaluate/scoring.ipynb
        modified:   mlxtend/evaluate/scoring.py
        modified:   mlxtend/evaluate/tests/test_scoring.py

no changes added to commit (use "git add" and/or "git commit -a")
[(base) sebastian@MBP2018 ~/Desktop/mlxtend % git add .
[(base) sebastian@MBP2018 ~/Desktop/mlxtend % git status
On branch rename-per-class-accuracy
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   docs/sources/user_guide/evaluate/scoring.ipynb
        modified:   mlxtend/evaluate/scoring.py
        modified:   mlxtend/evaluate/tests/test_scoring.py

(base) sebastian@MBP2018 ~/Desktop/mlxtend % git commit -m 'rename "per-class accuracy" to "average per-class accuracy"'
```

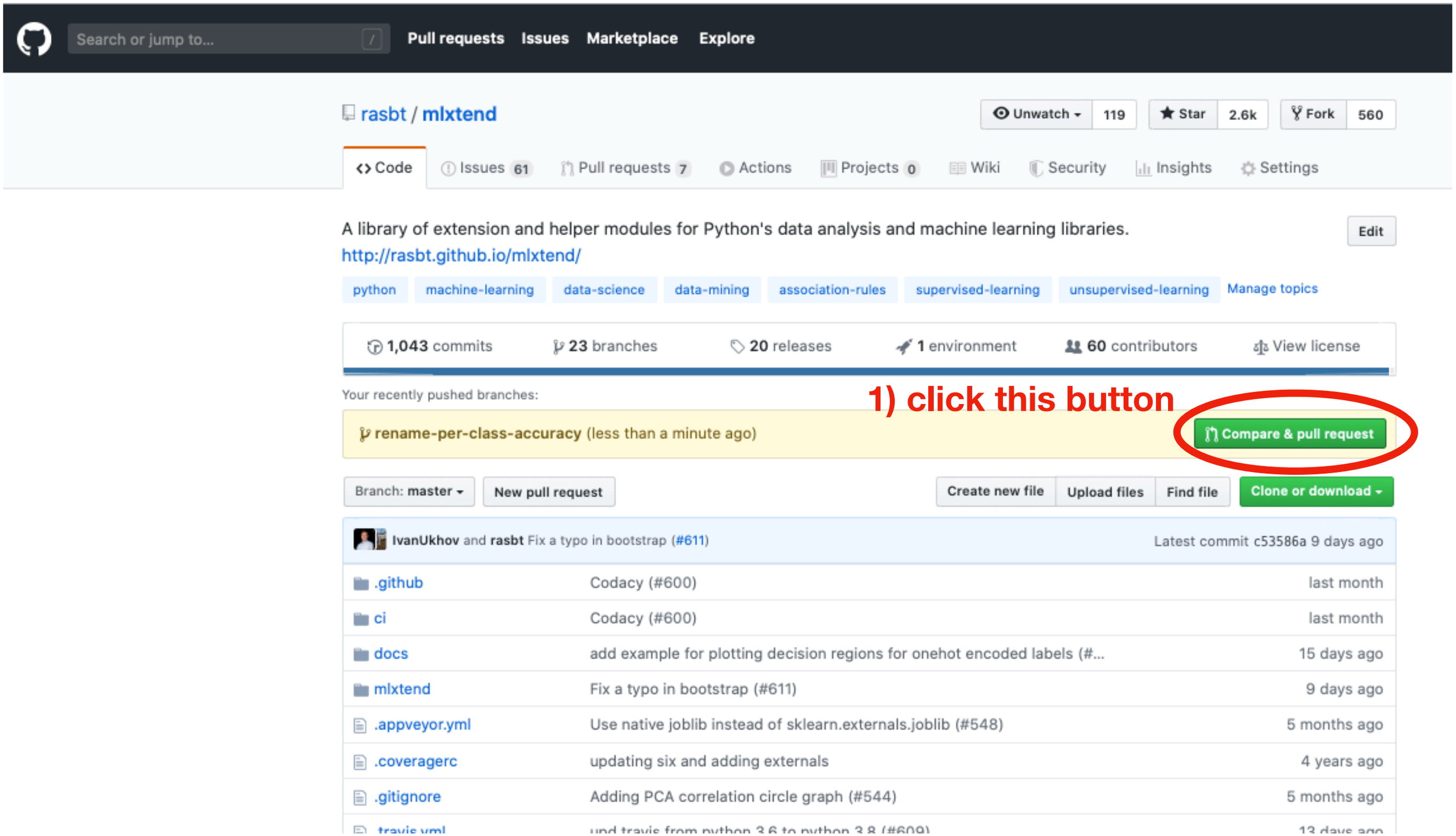
Step 9: Upload your Changes

- **Push your changes to the server**

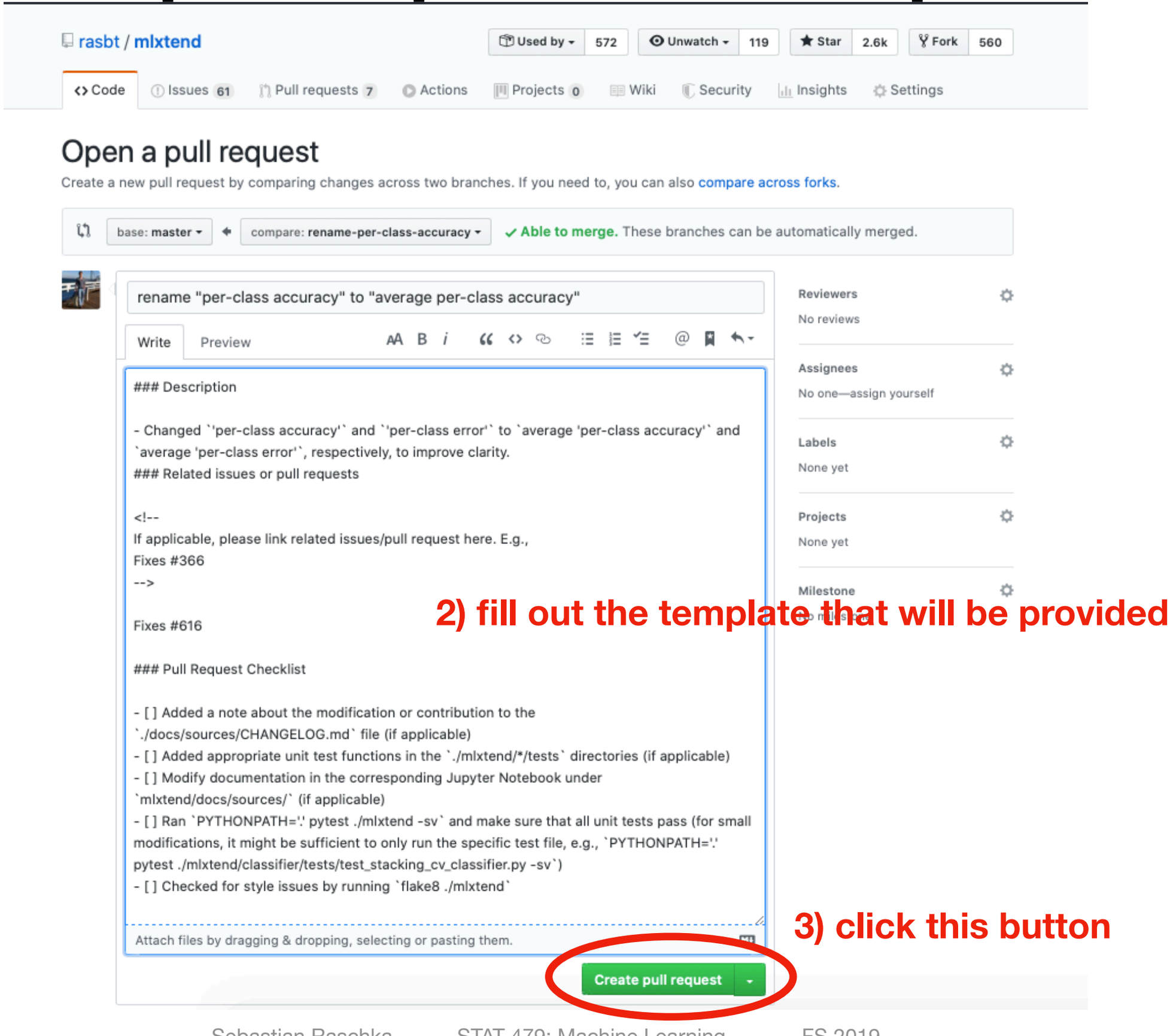




Step 10: Open a Pull Request



Step 10: Open a Pull Request



2) fill out the template that will be provided

3) click this button

Step 10: Open a Pull Request

4) check off these items

5) check the automated unit test results later