**Section A – Open Acounts & Installations**

install GIT - https://git-scm.com/downloads

install PYTHON 3.7 - https://www.python.org/downloads/  
if you run windows operating system make sure python is defined under path variables:

https://stackoverflow.com/questions/35328991/setting-up-python-on-windows-10-pro

install PYCHARM COMMUNITY EDITION:

<https://www.jetbrains.com/pycharm/download/>

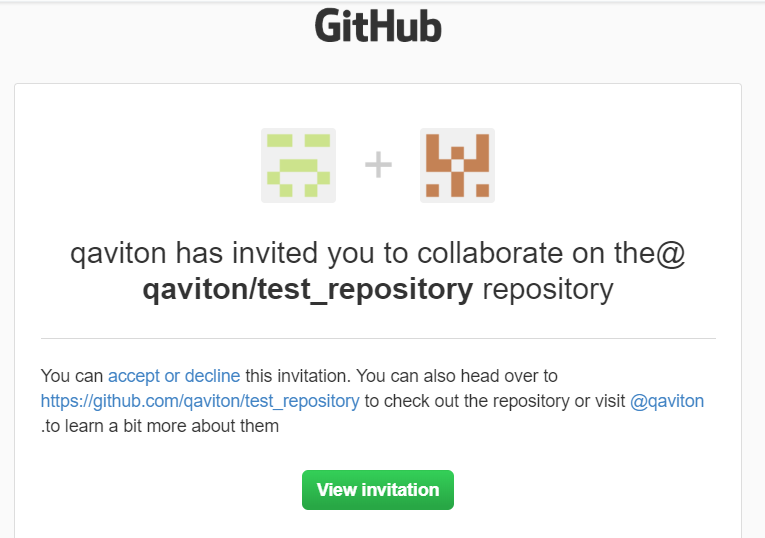
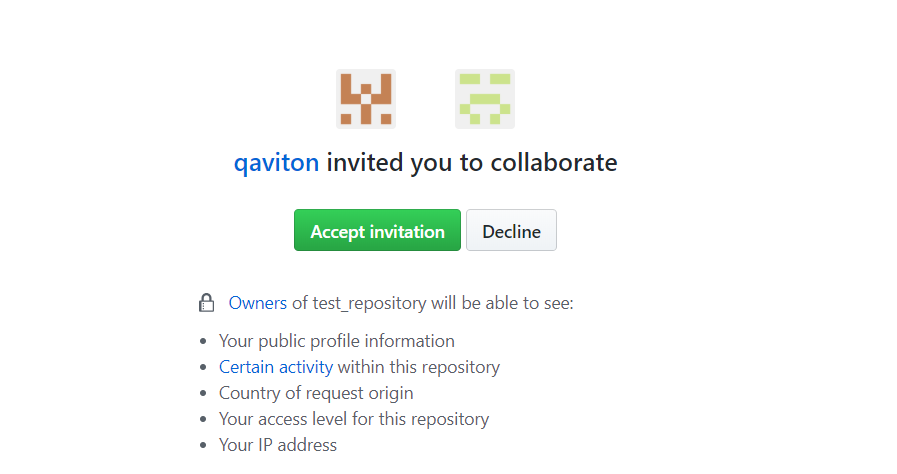
open a github account:

https://github.com/join?source=experiment-header-dropdowns-home

verify your account via email

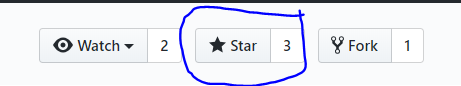
send us your github username so we could add you as a contributor.

Check your email for our invitation:

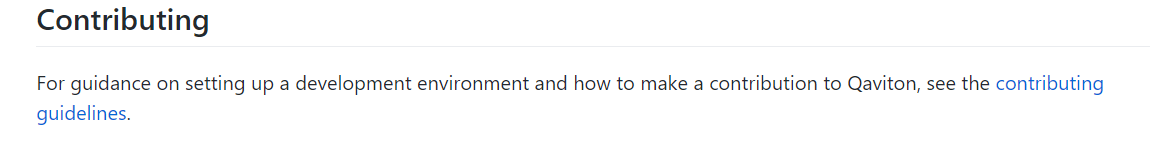
  
  


**Section B - FORK & STAR**

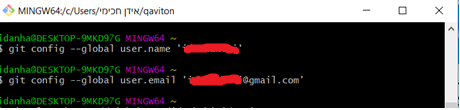
1) Go to:  
<https://github.com/qaviton/qaviton>

2) Please give us a star:  
  


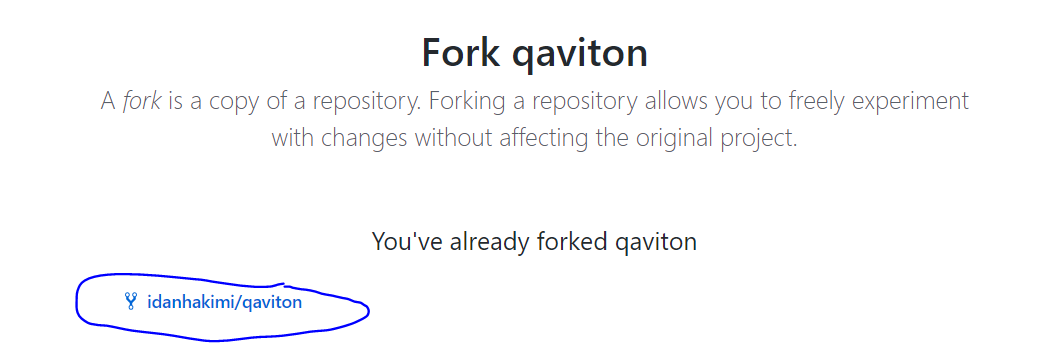
3) scroll down to the Contributing category and click on Contributing Guidelines

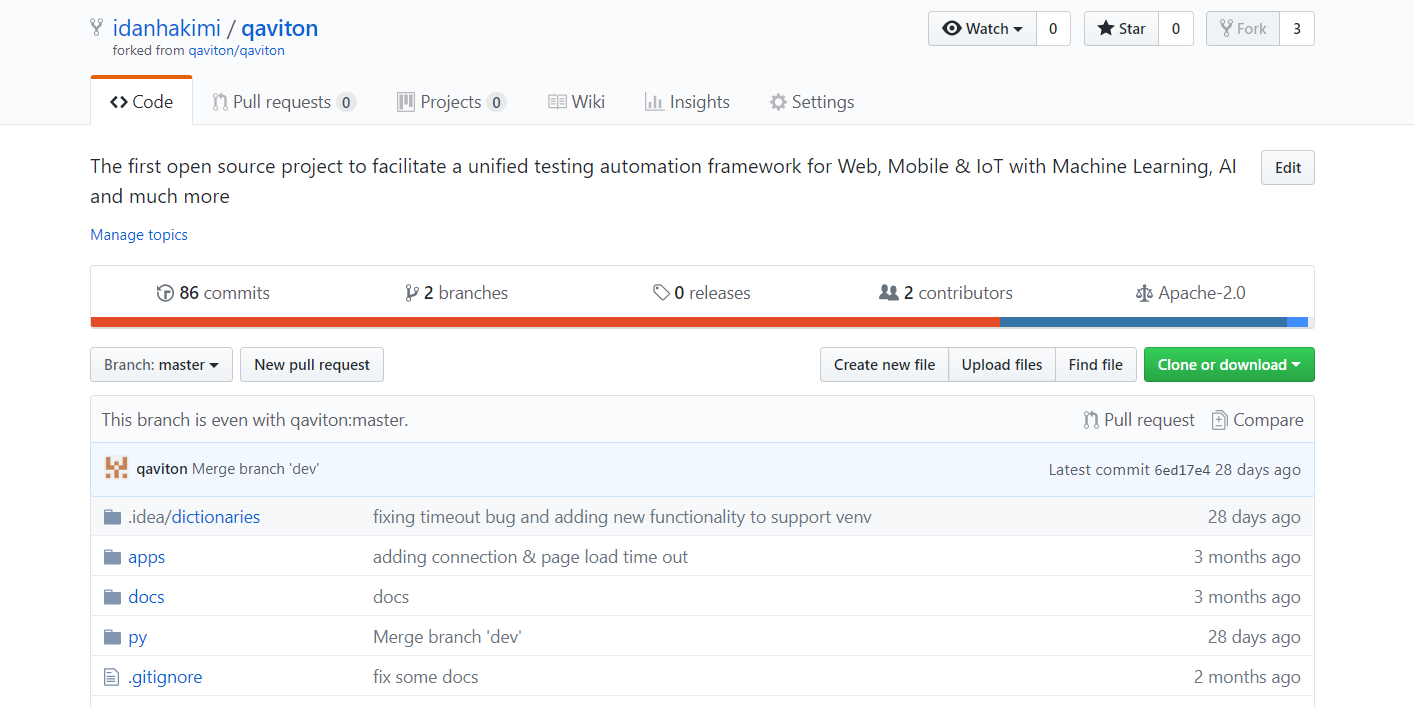


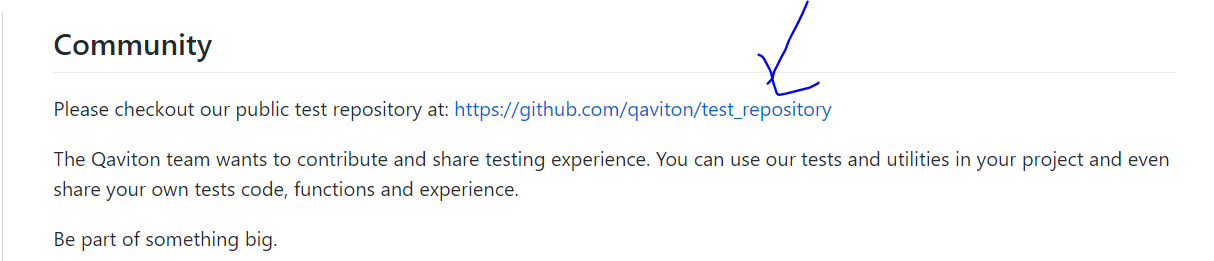
4) scroll down to **First time setup**  
  
  
  
  
  
5) now lets open git bash and enter these commands(you only need to do this once):

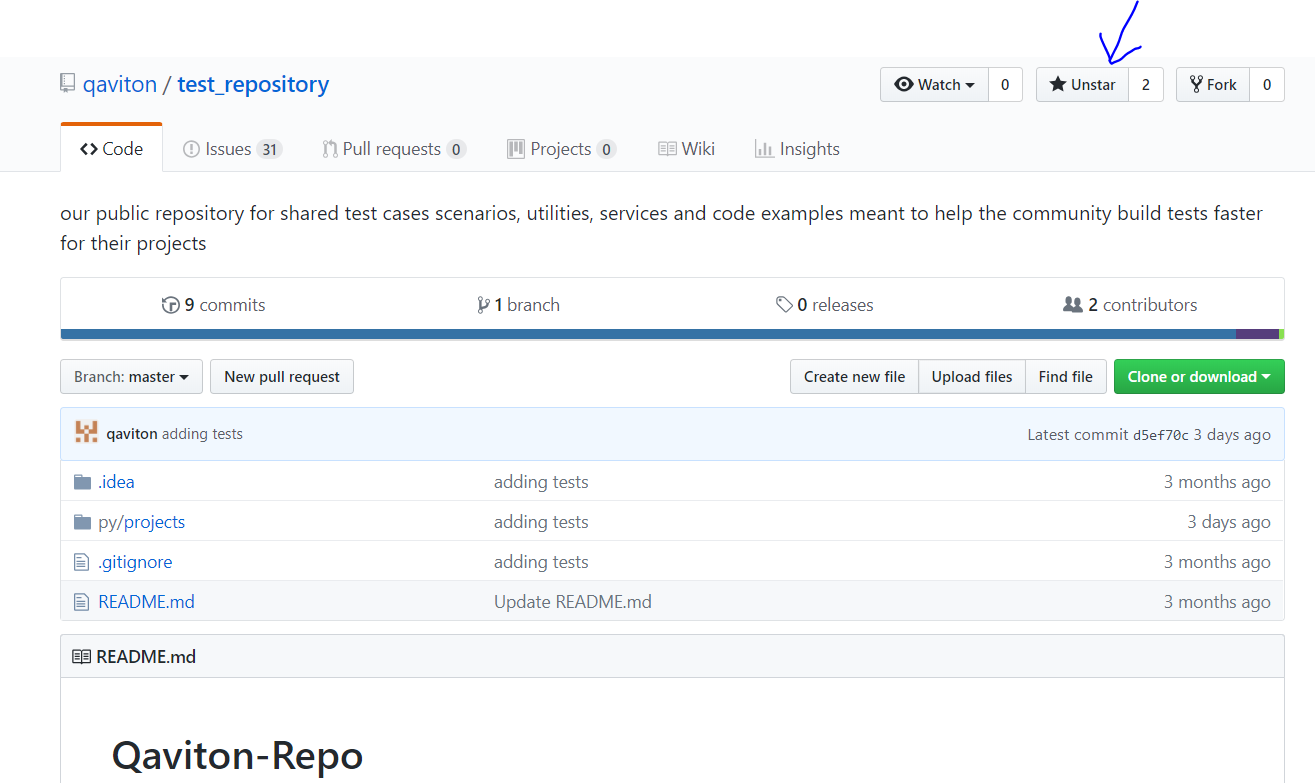
  
  
6) click on FORK

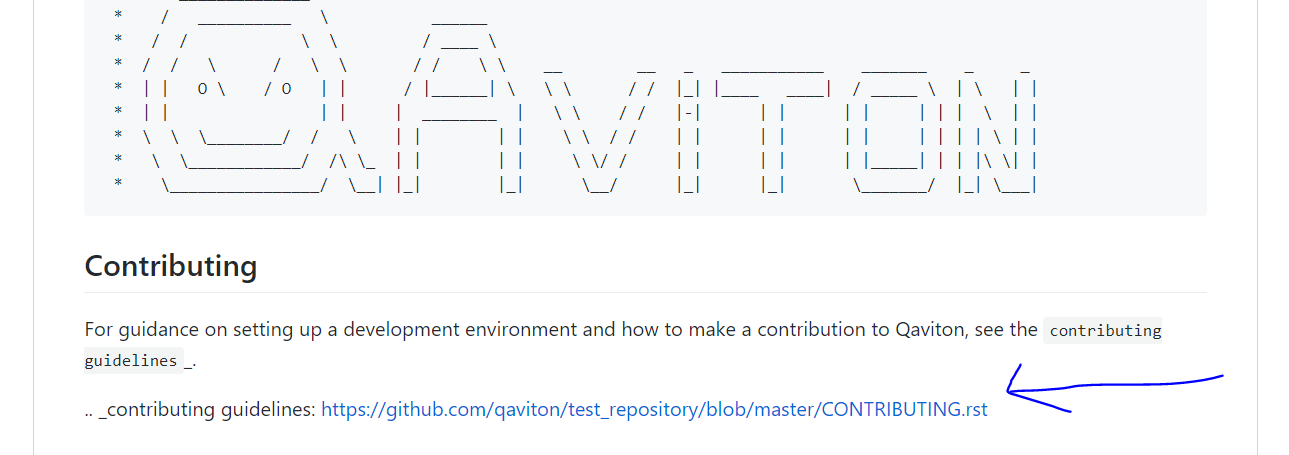
https://help.github.com/articles/fork-a-repo/

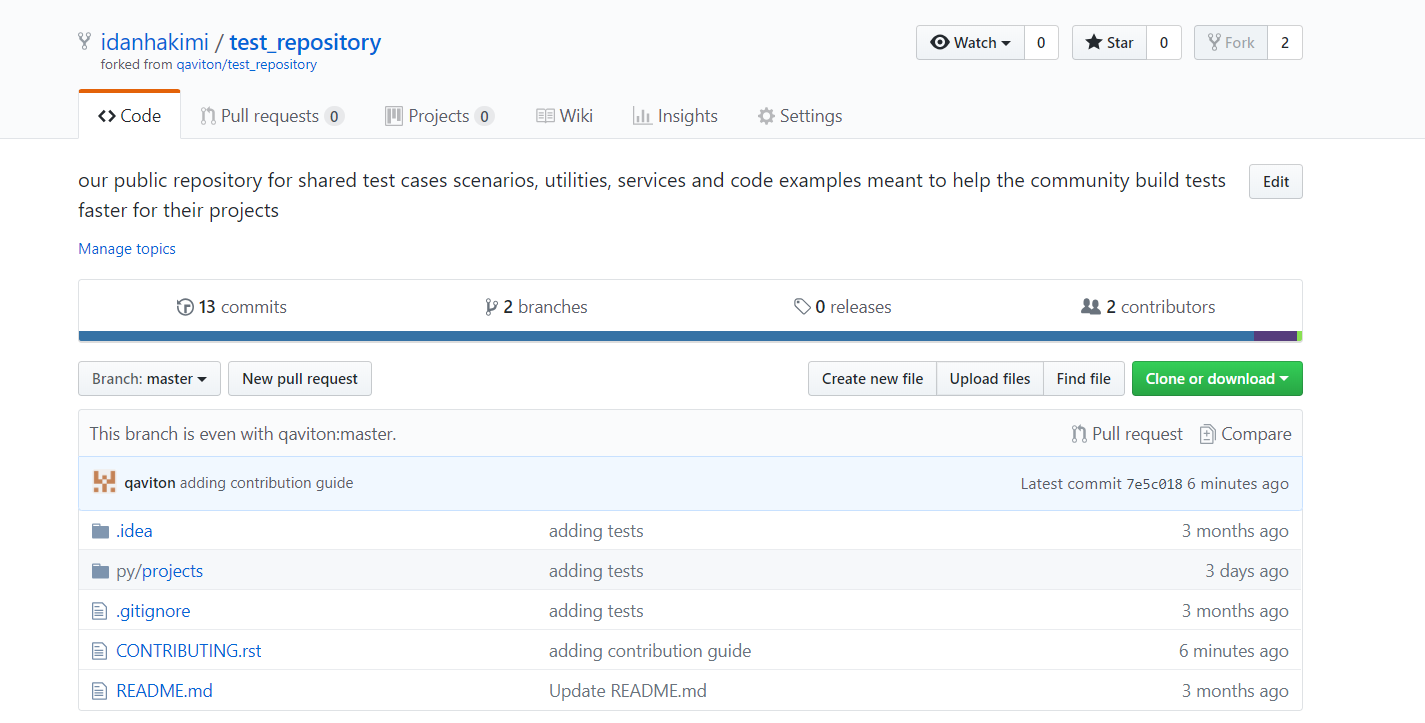


Now we expect to see the repo under our user:  
  
  
  
7) now let's repeat this process for: <https://github.com/qaviton/test_repository>





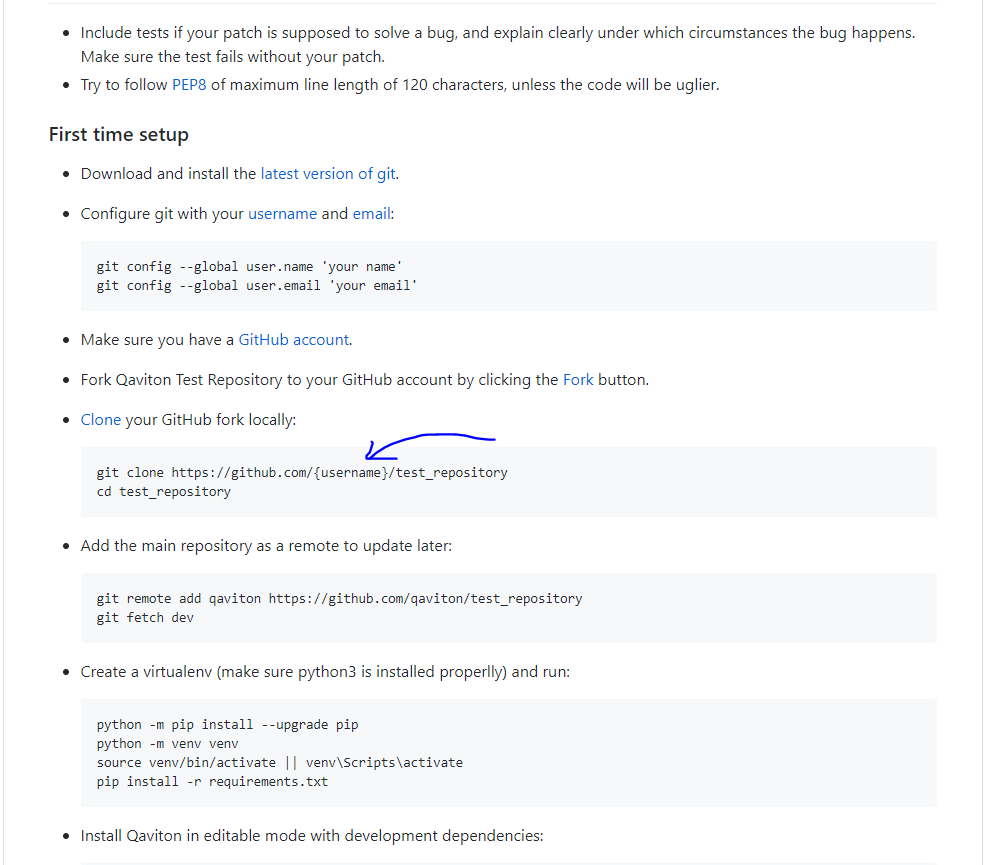


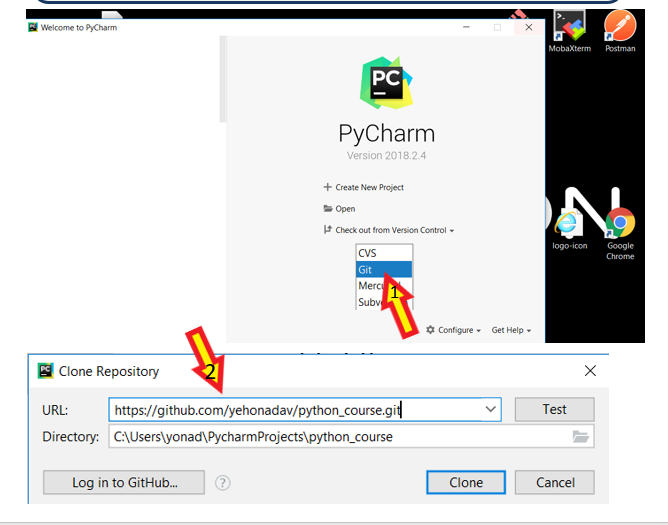


**Section C - CLONES**

Let's proceed and clone our test\_repository to our local host using gitbash/pycharm

Write down the clone command in your cli terminal and put in your username where you see {username}, then proceed with the cd test\_repository command.



Clone using pycharm: URL: [https://github.com/{username}/test\_repository.git](https://github.com/%7busername%7d/test_repository.git)  
  
  
  
let's check wer'e in the correct branch:

git checkout -b dev

git branch -vv

we need to make sure we are in /origin/dev and not in /origin/master:

git branch --set-upstream-to origin/dev

git branch -vv

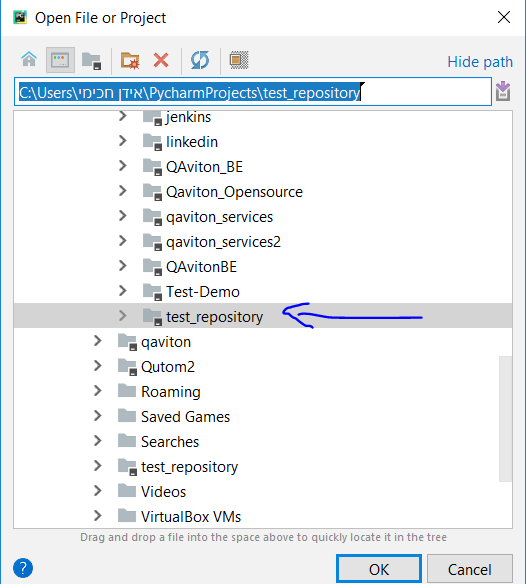
we also want to make sure our remote url is correct and under our username:

git remote -v

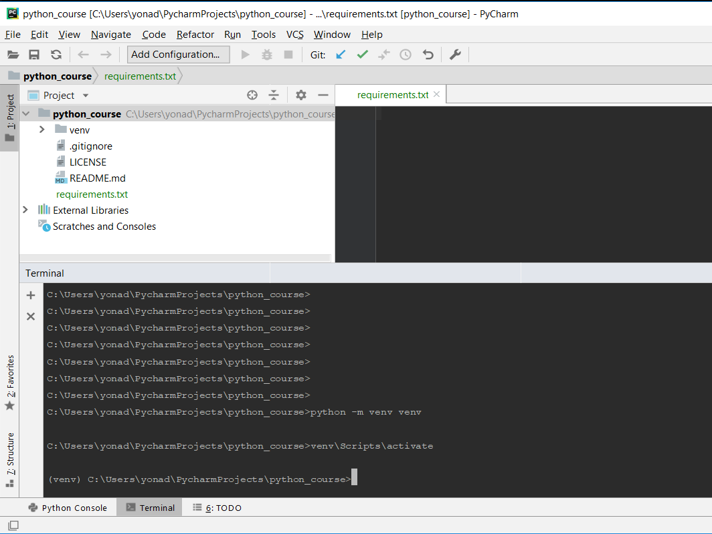
if we see the original owner in the url we can fix that with the following command:

git remote set-url origin [https://github.com/{username}/test\_repository.git](https://github.com/%7busername%7d/test_repository.git)

**Section D – IDE configuration**

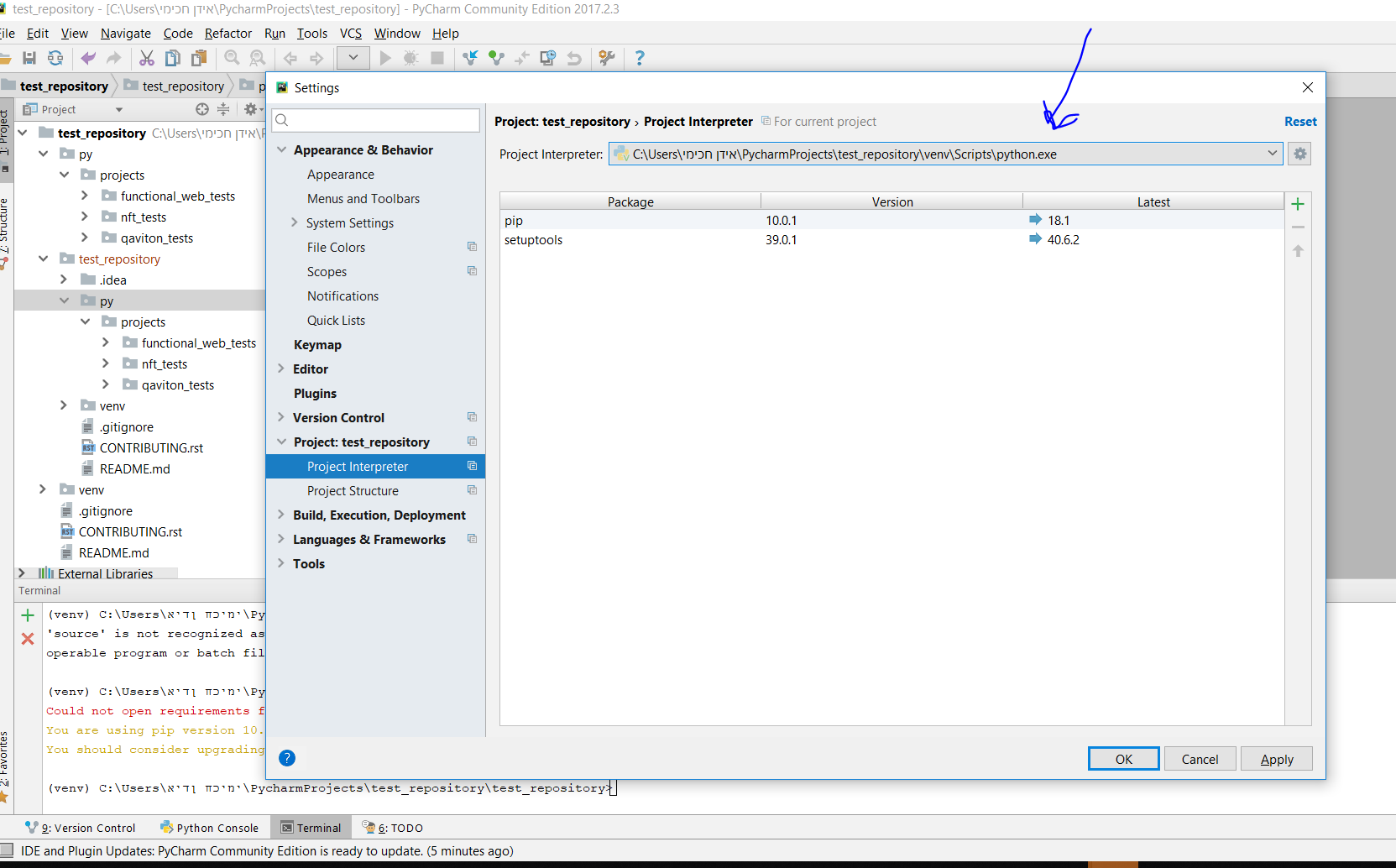
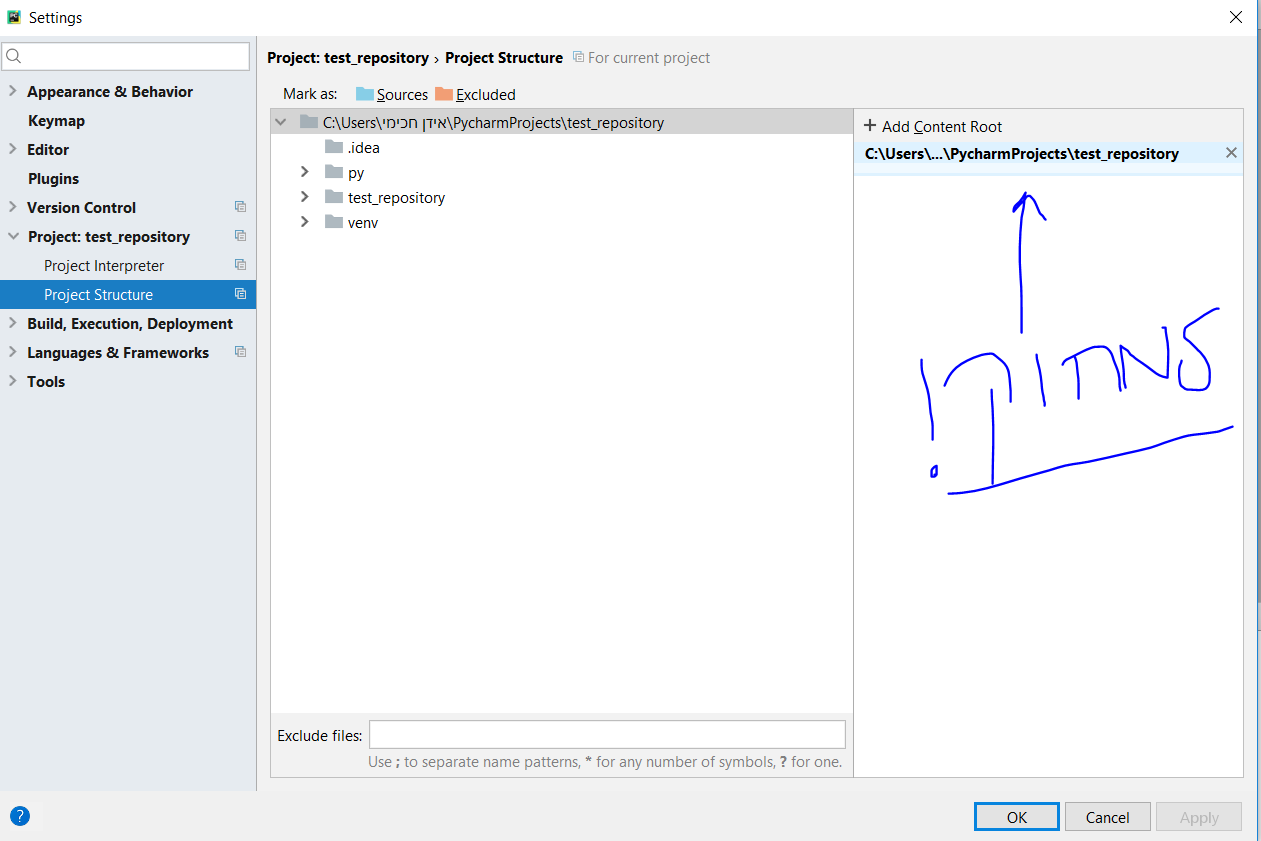
Let's open the project with pycharm  
  


Now we install a virtual environment:

https://pythontips.com/2013/07/30/what-is-virtualenv/  
  
after running  
python -m venv venv  
  
we can start using our virtual env with running:

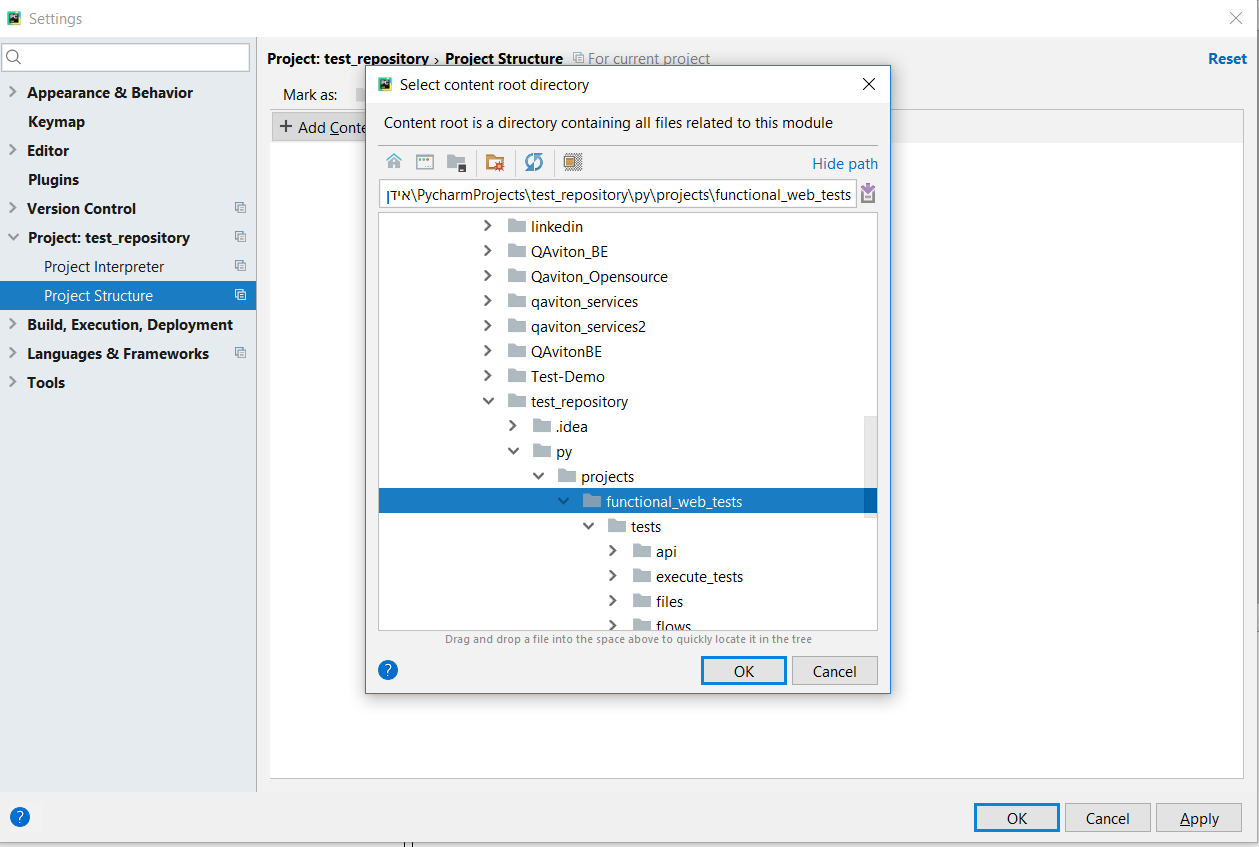
venv\Scripts\activate

now let's open pycharm's file -> settings and define a project interpreter:

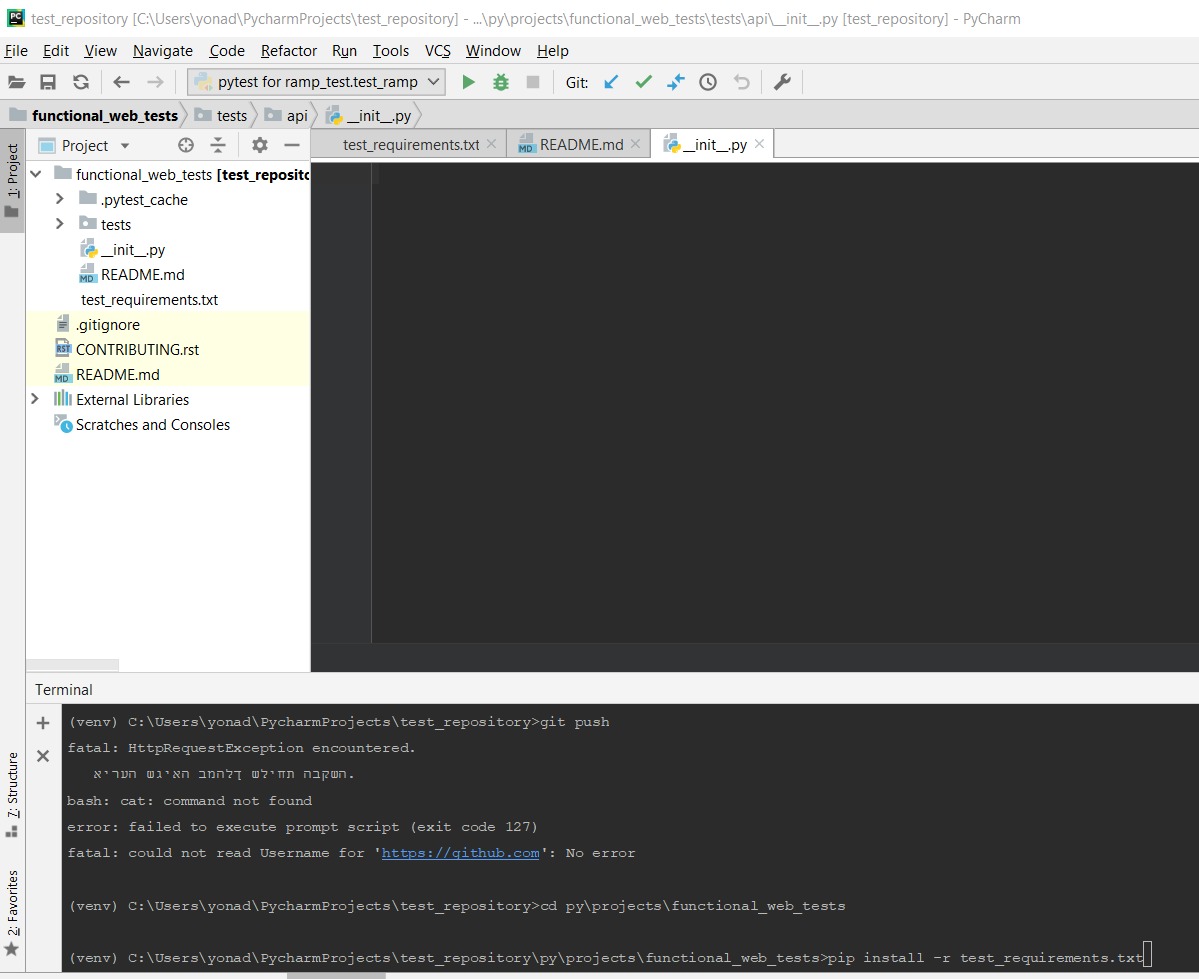
1.   
     
   the next step is to setup the project's root path. All we have to do is go to project structure and delete the current path:   
   

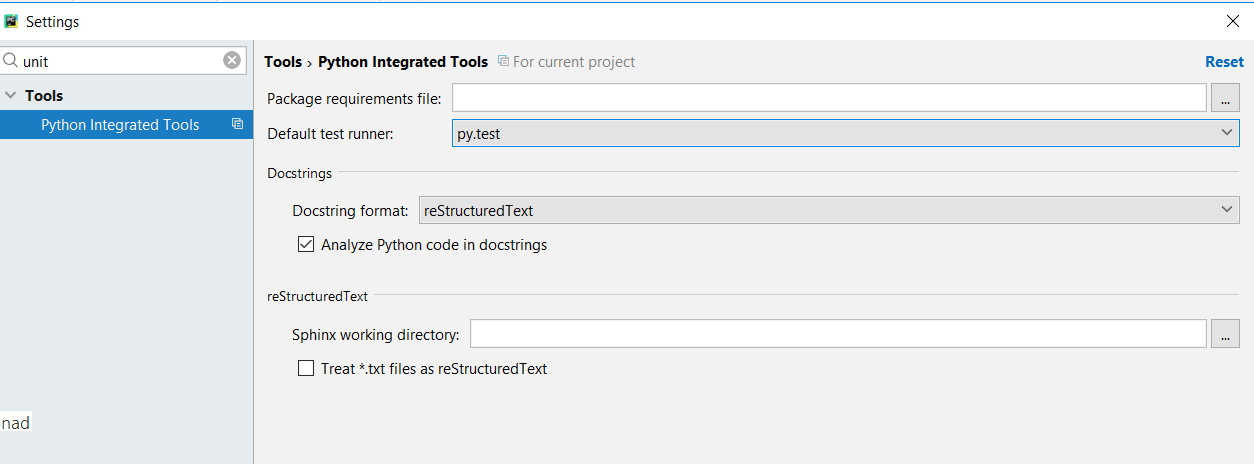
'add content root' and add the project we want to work on. then click on

We currently work on functional\_web\_tests:



Now in the pycharm terminal we run: cd py\projects\functional\_web\_tests

And run the following command: pip install -r test\_requirments.txt  
  
  
  
and lastly we need to change our test runner from the default unittest to py.test/pytest

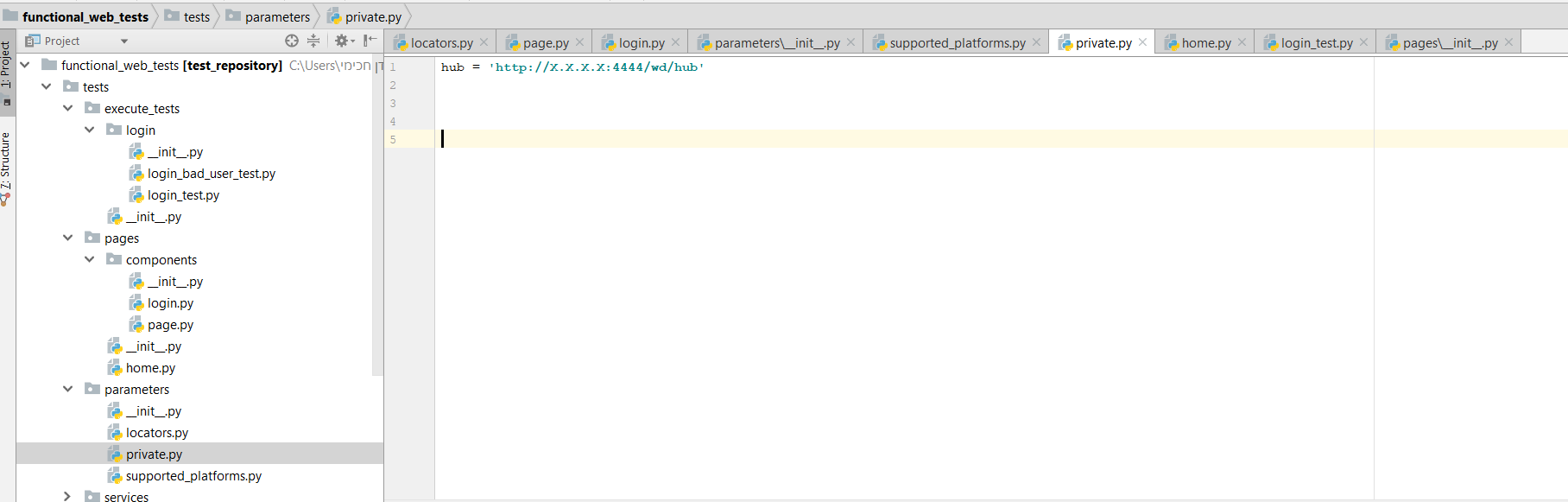
Apply, save and re-open pycharm afterwards.

now after being approved as a contributor by qaviton, we can start working and create new files under the tests dir.

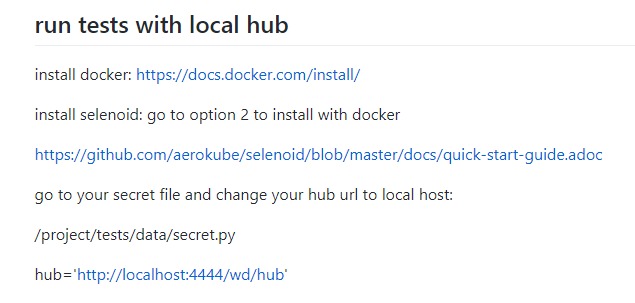
Create a new file: py\projects\functional\_web\_tests\tests\parameters\private.py and enter this code line:

hub = **'http://X.X.X.X:4444/wd/hub'**

this hub url will provide us with remote drivers for testing in chrome, android, firefox etc.

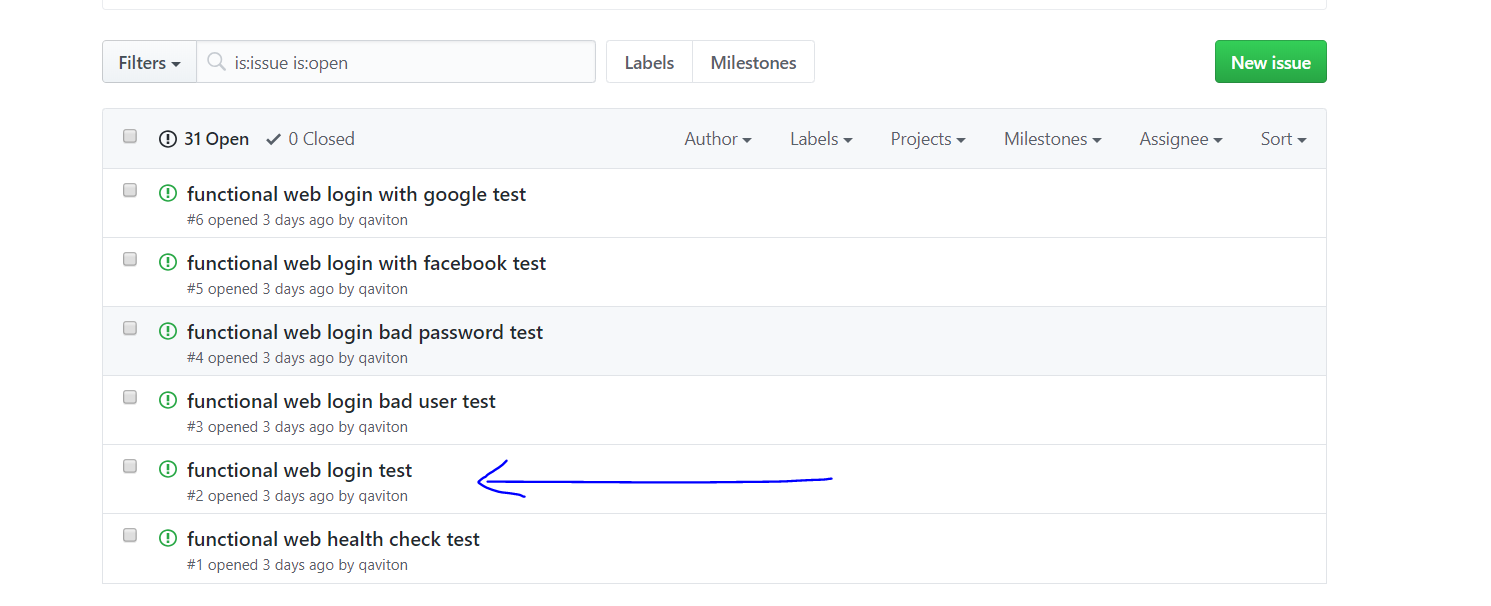


we can install a local server for this:

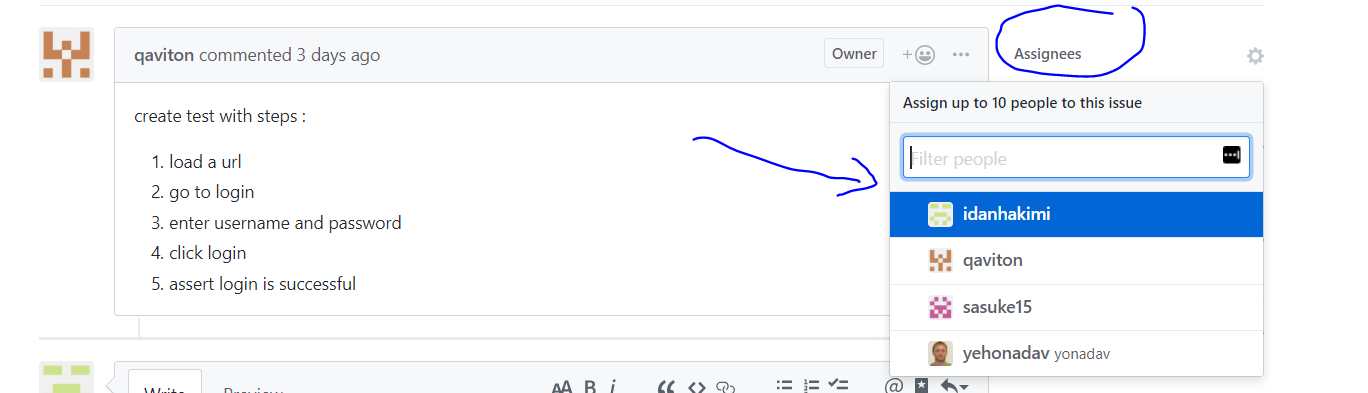
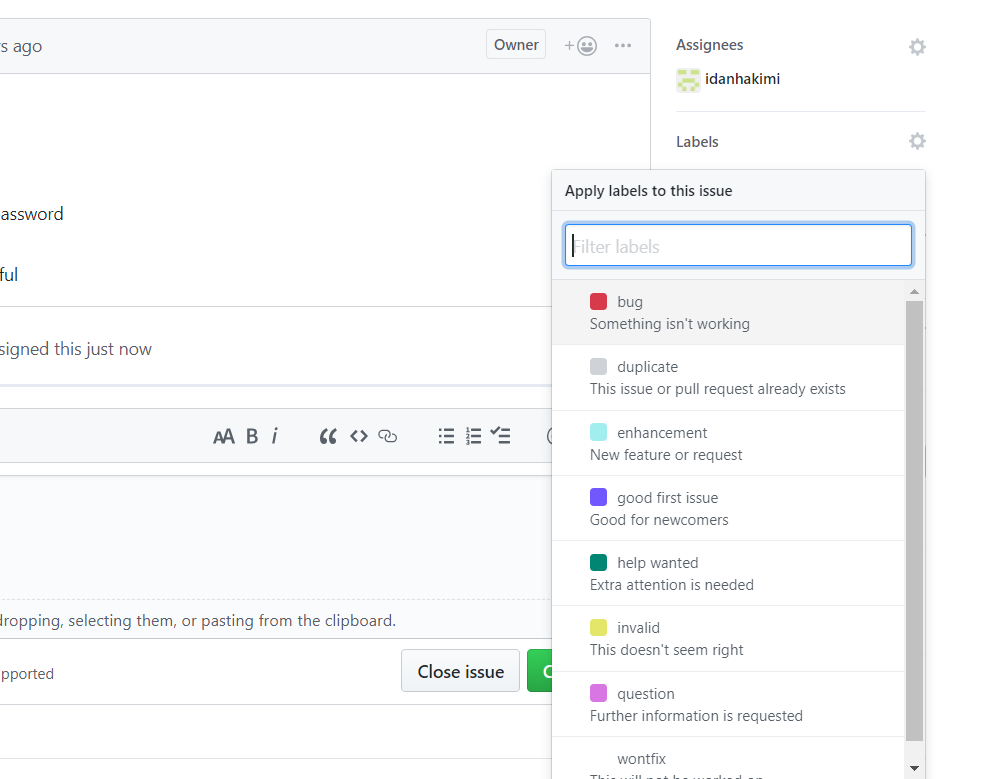
https://github.com/qaviton/qaviton  
  


**Section E - Issues Assignment**

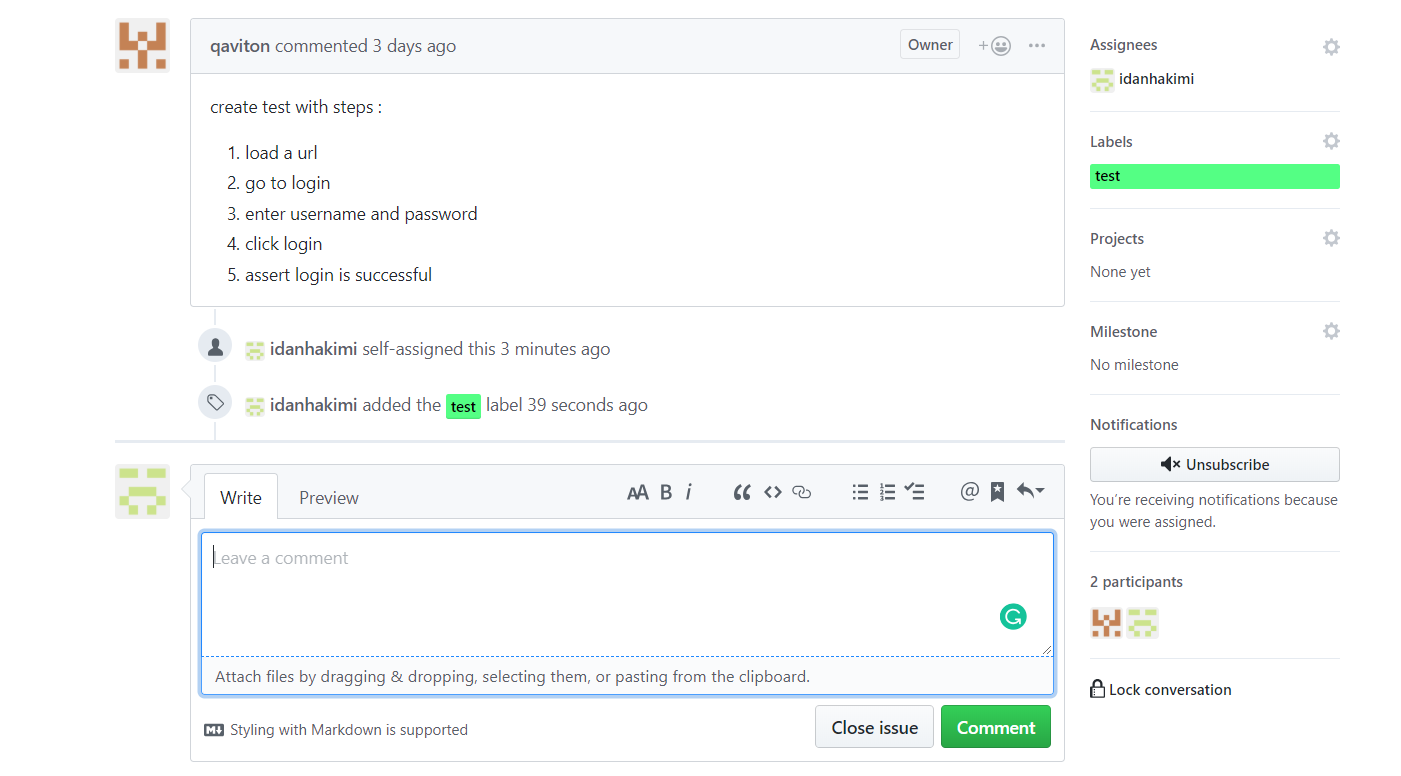
Take and create issues and assignments from here:

<https://github.com/qaviton/test_repository/issues>  
  


"functional web login test"example of how to take an issue:

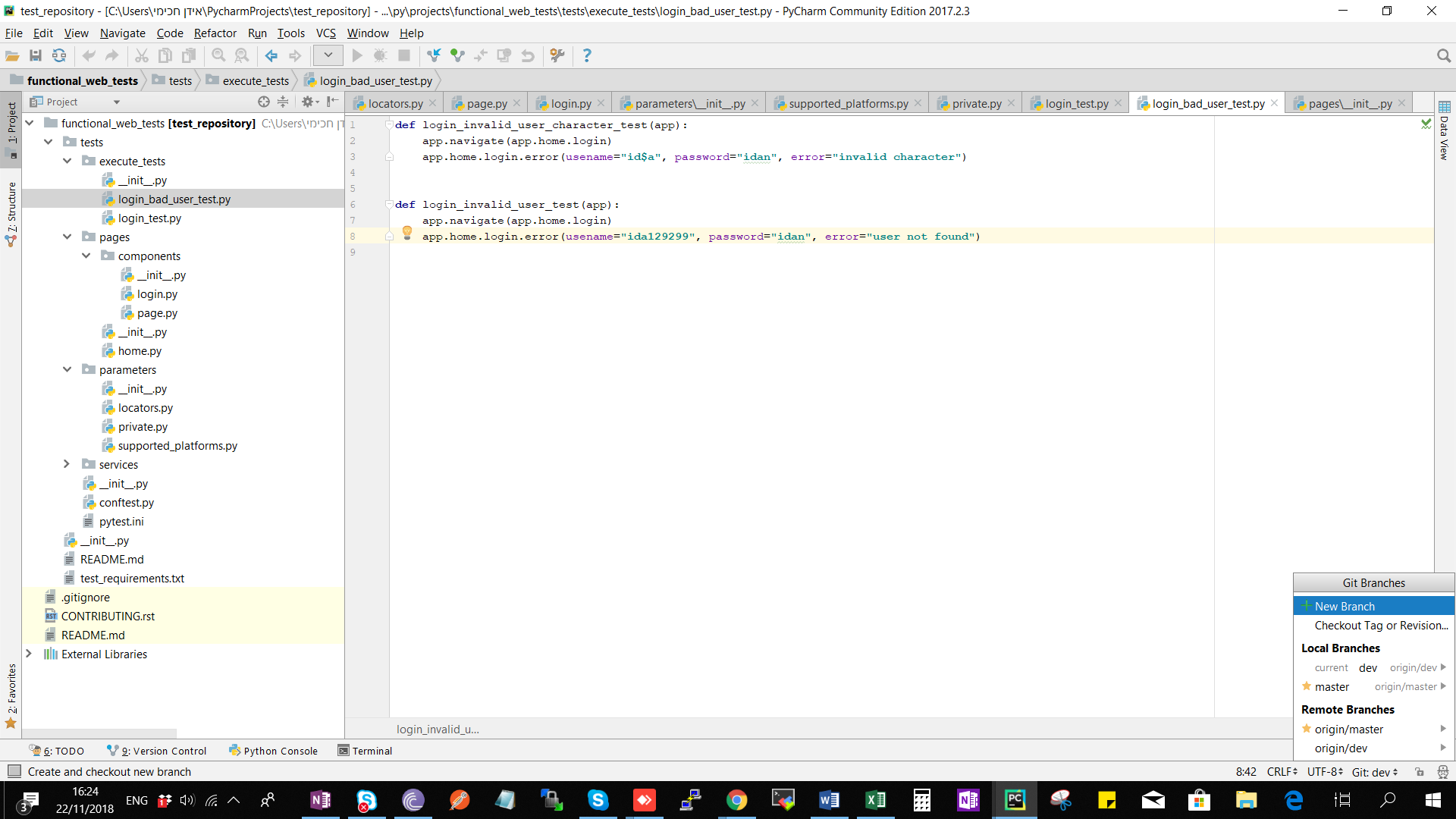
Now we assign a contributor to the test:  
  
  
we can label this issue as test:  


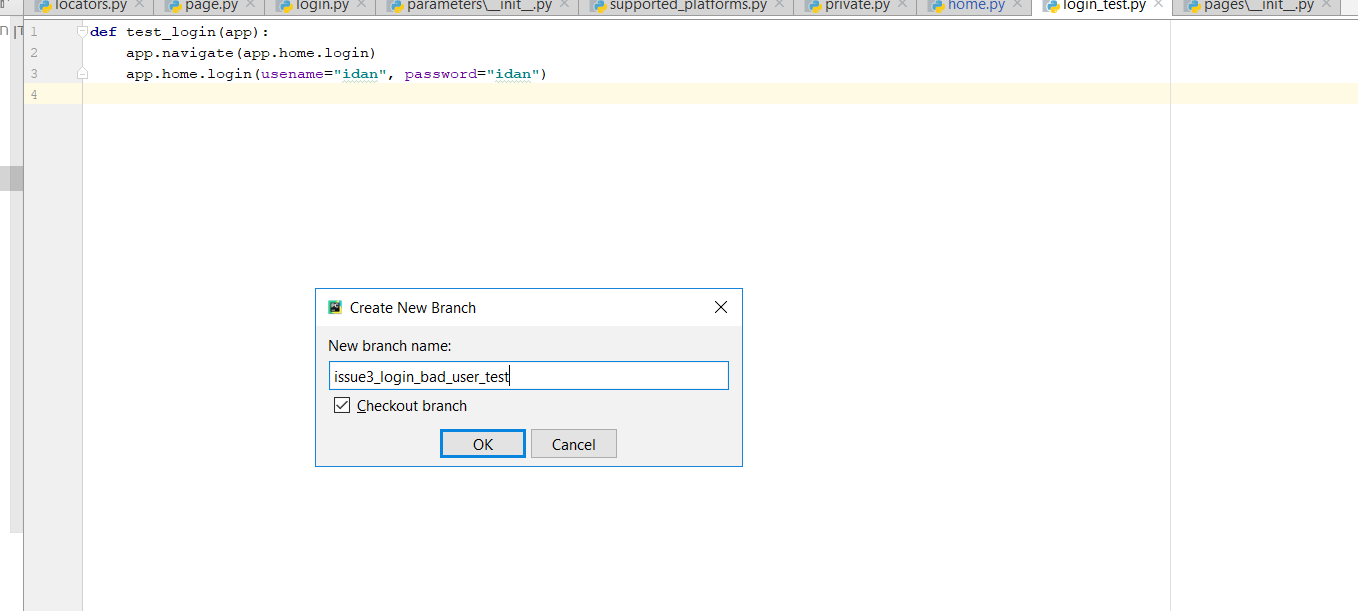
And add comments as we work on solving the issue

Finally when wer'e done we can close the issue   
  
  
after we take issues and have our IDE setup with our cloned repo we can start building tests.

**Section F - Open new branch**

For every issue we take, we need to create a local temporary branch with a name convention of the issue name & number with underscores replacing spaces:



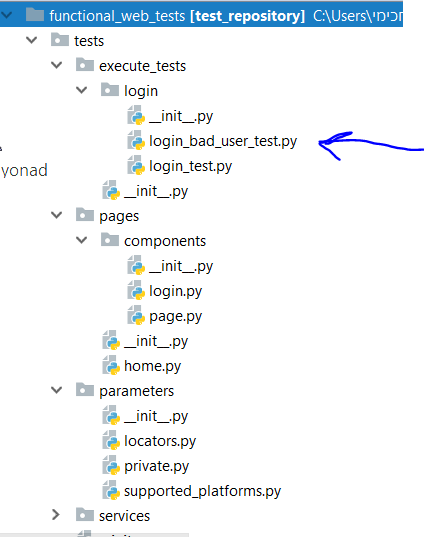


**Section G – Build Tests**

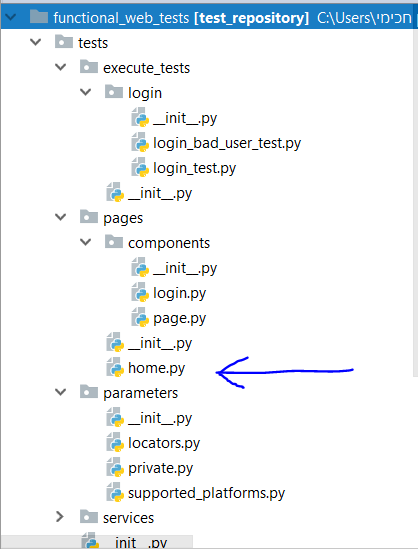
This is the file & test structure we work on, inside the execute\_tests dir.

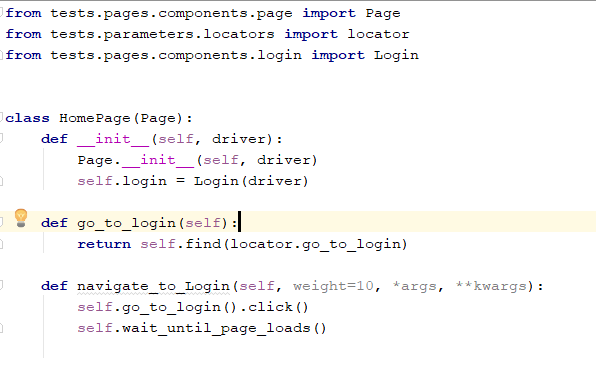
For every issue we create a .py file with the issue name with underscores replacing spaces.

If the issue contains many tests, so will the test file.

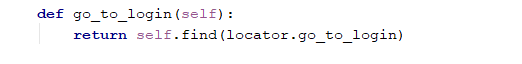


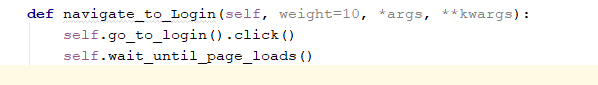
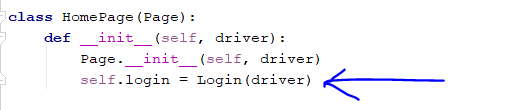
Under pages we define our pages & components, home.py as an example.

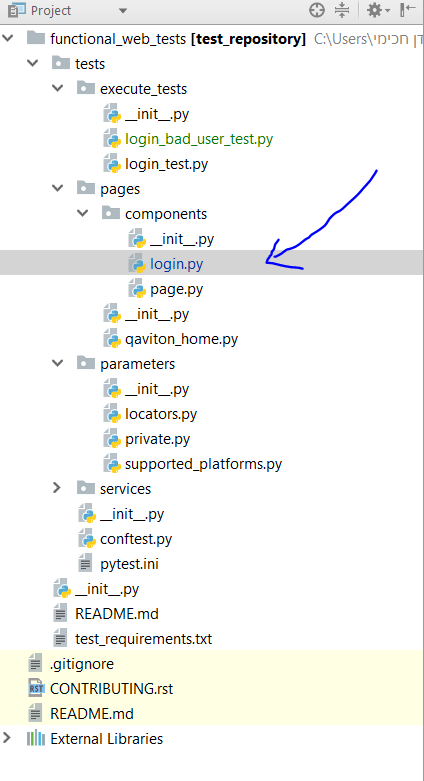




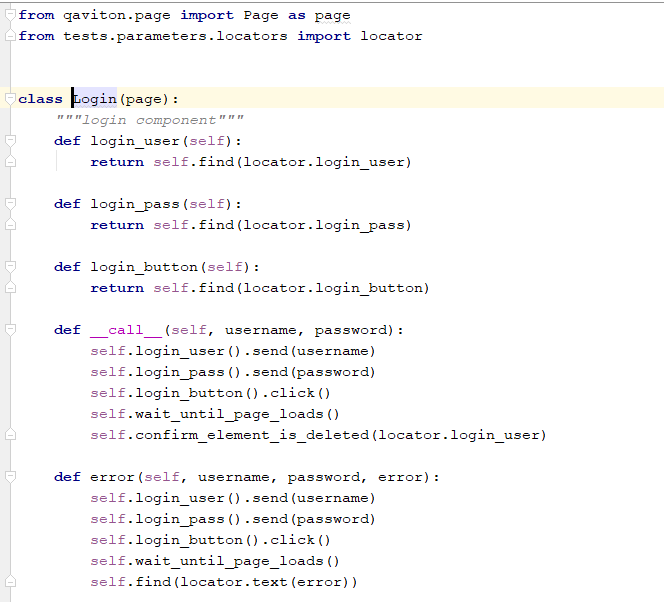
Pages contain Navigation function, Elements & Components

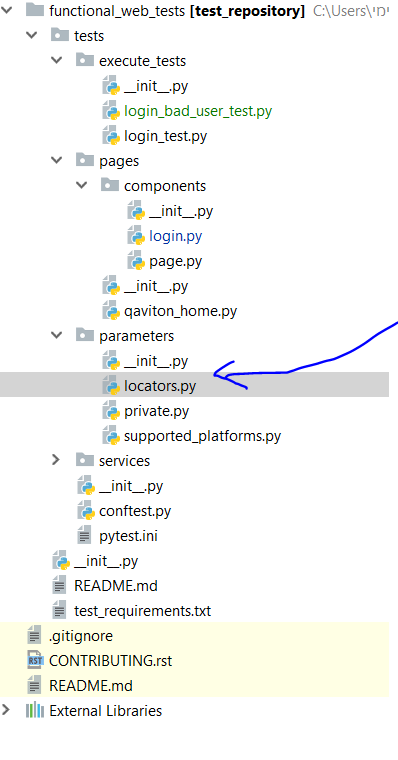
Elements are reactive buttons in the page and are configured using the locator object which holds a list of all element identifiers in our app. We use the find method to locate the element and return it with the page function with identical name to the locator name.  


Navigations are defined when moving from one page/component to another.  
the navigation functions have name convention is to start with 'navigate\_to\_'+'{Page/Component name}'. usually after clicking we will wait for the page to load the changes. The navigation function takes a weight=10 argument and \*args and \*\*kwargs. More on this will be explained in the docs.   
  
when we have a page with reusable/repeating parts we need to split that page into components.  


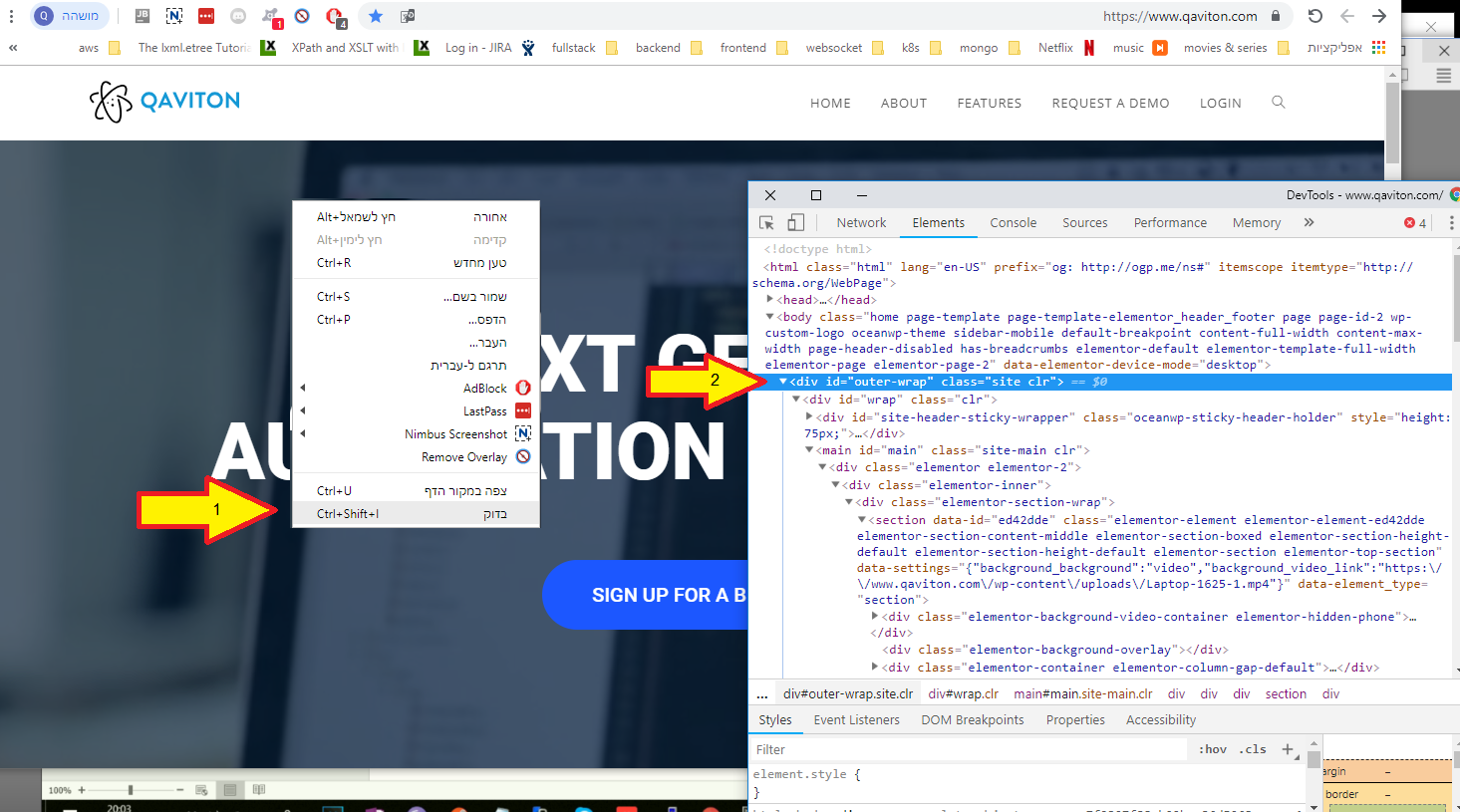
Example for the login component:  


Components behave just like pages. They have elements, buttons, functions and navigations.

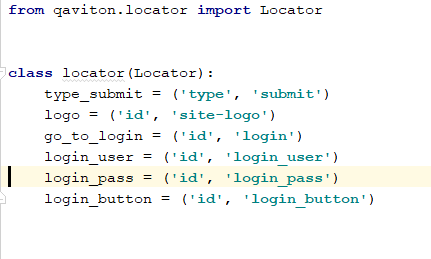


Where we define locators:  
  


Using the browser inspect we can find the app's locator values:



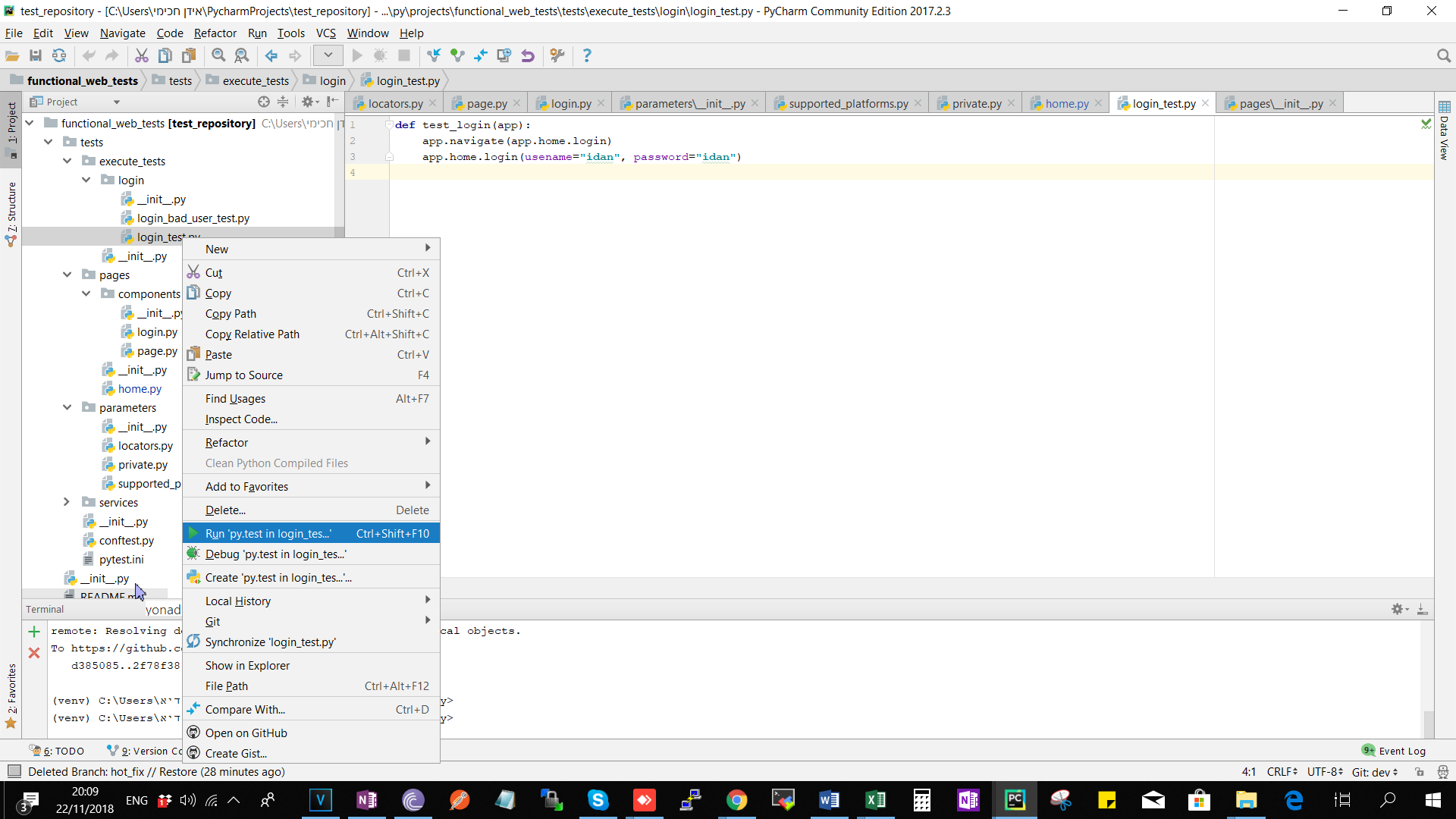
After finding a testable element we can locate him with let's say his id attribute, we need to copy that attribute to our locator object:



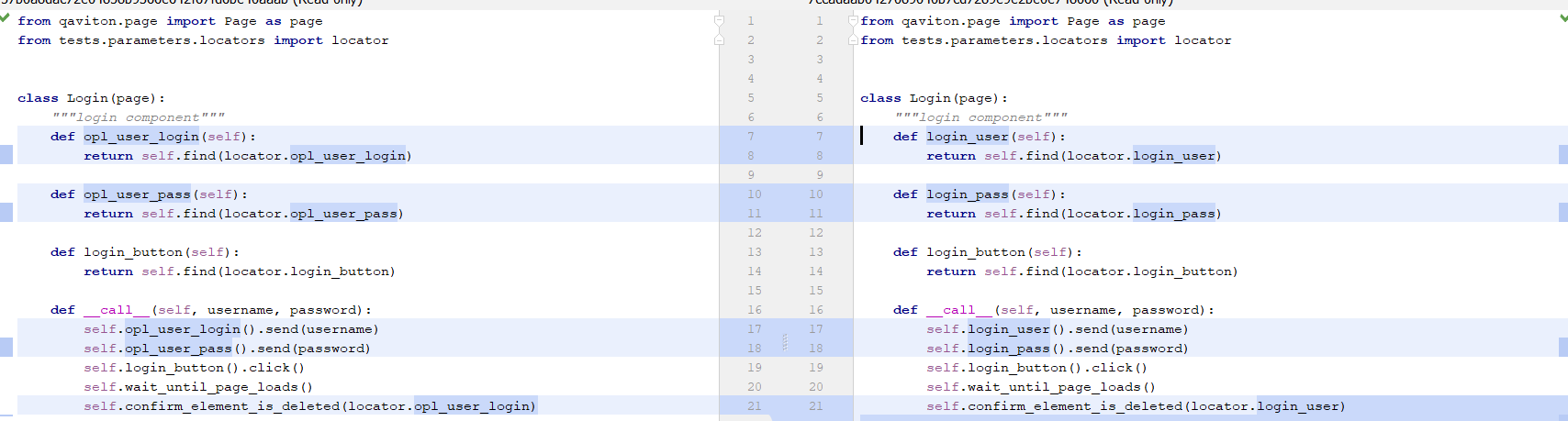
After defining and modeling our locators, pages, components we can start creating tests:



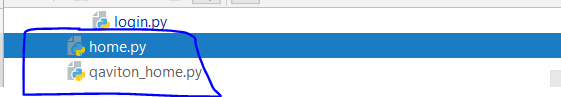
We run the test locally with a simple right click and choosing Run with pytest



<https://www.jetbrains.com/help/pycharm/pytest.html>  
after the test finished running successfully we would change our names and values to something generic so that other users can simply copy our tests and make relevant changes to run them on their applications.

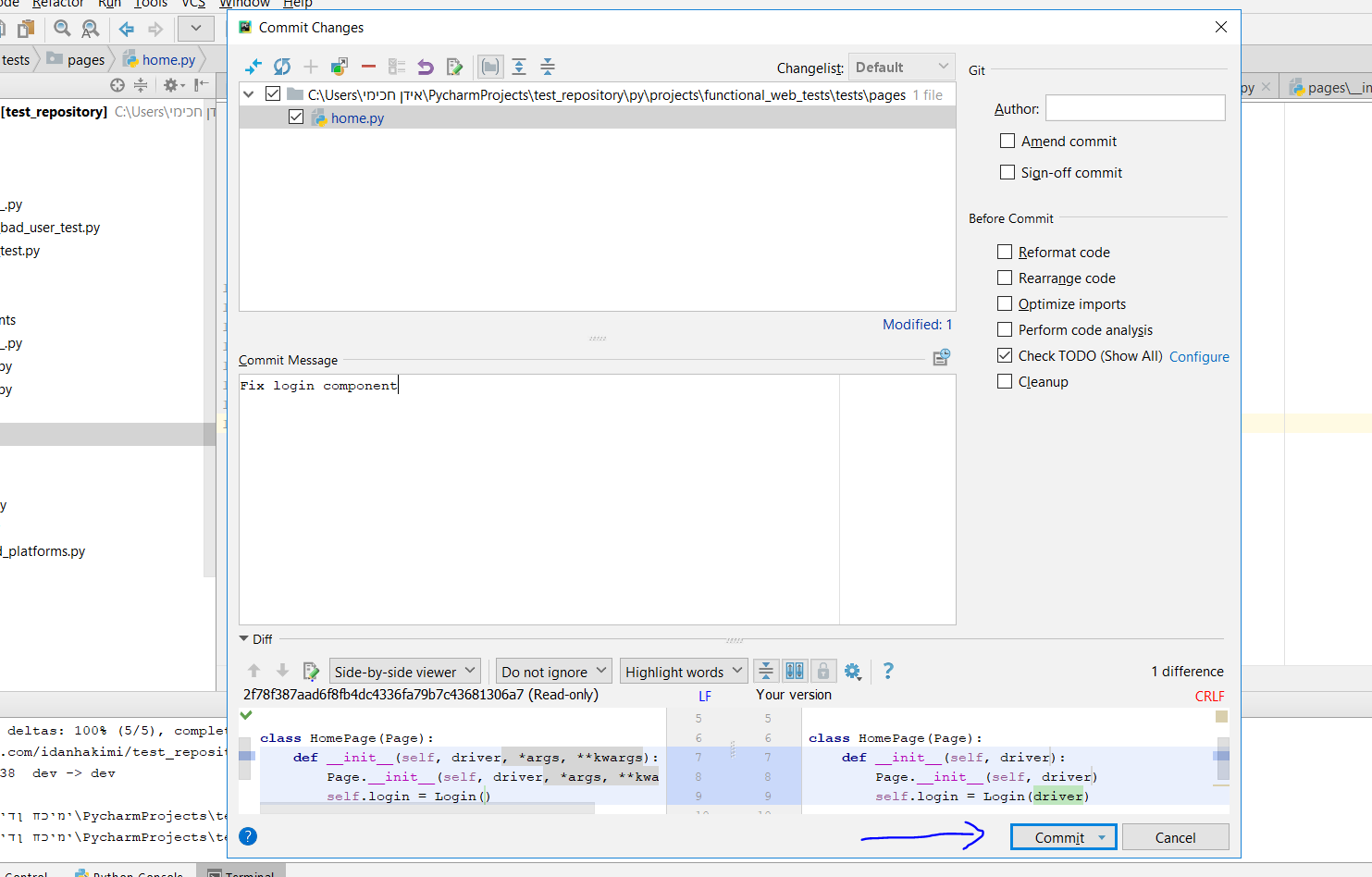






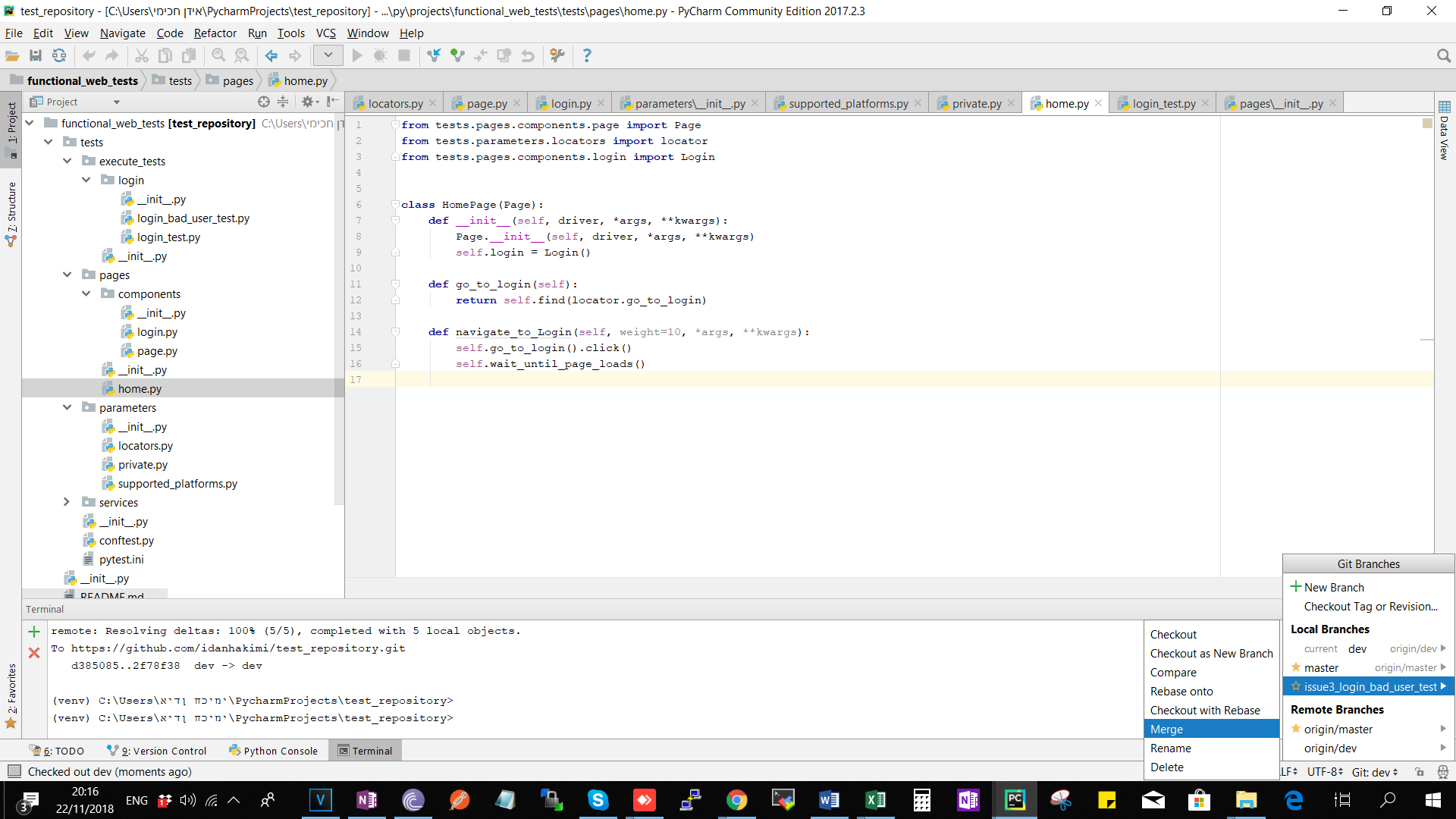
when generalization is finished we will commit the project with a relevant message to our test.

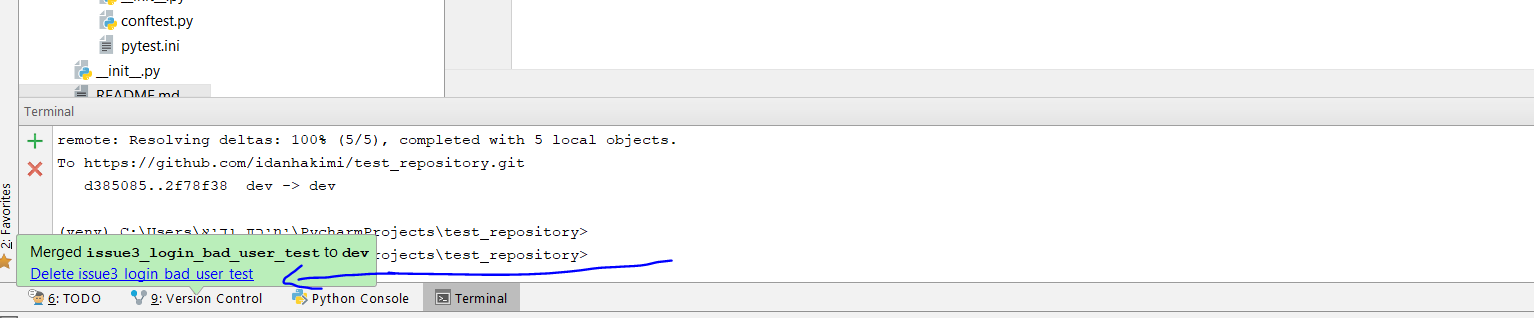


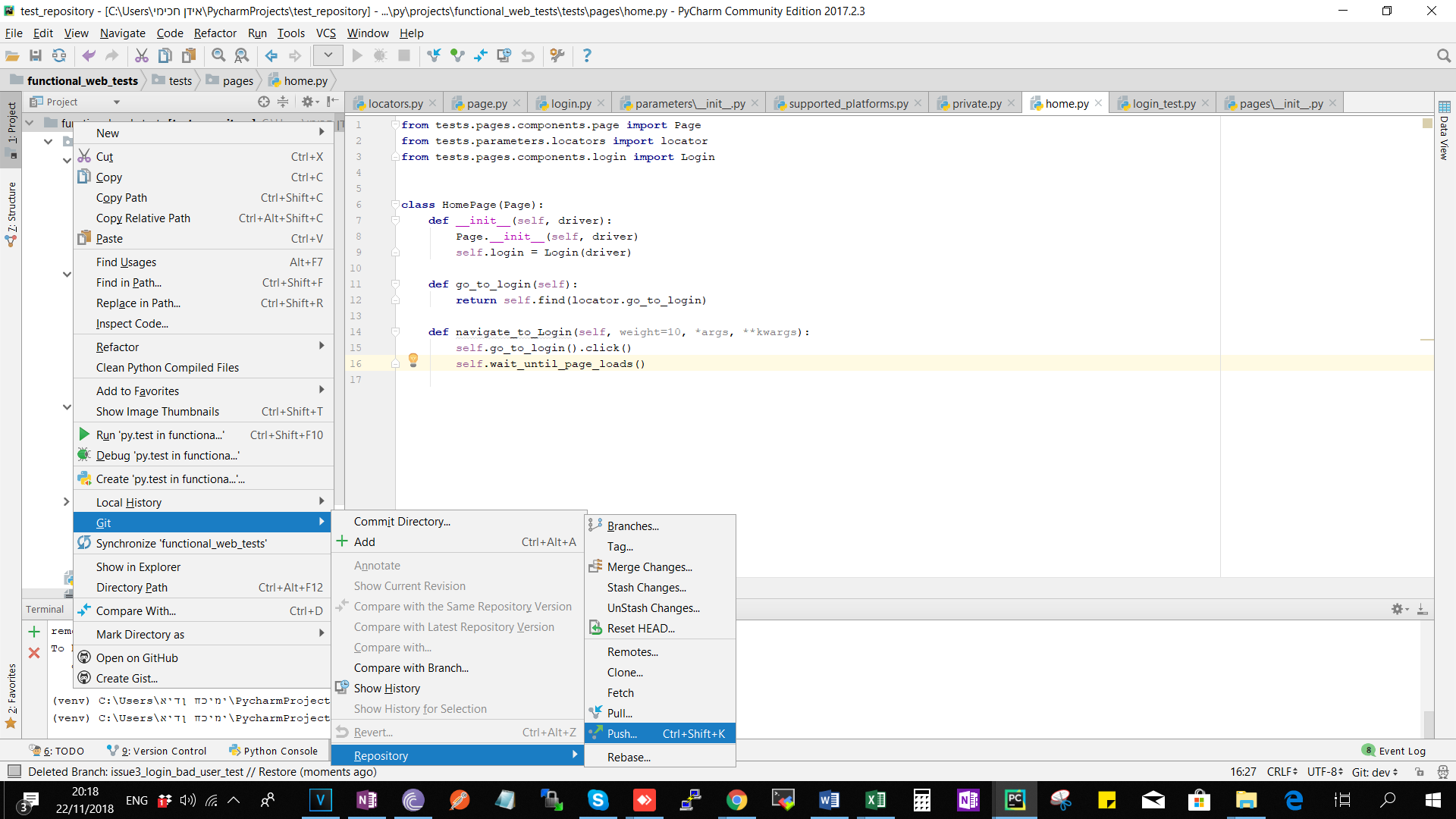


After all tests of the issue are done and committed we need to merge into the dev branch and delete the temporary issue branch.







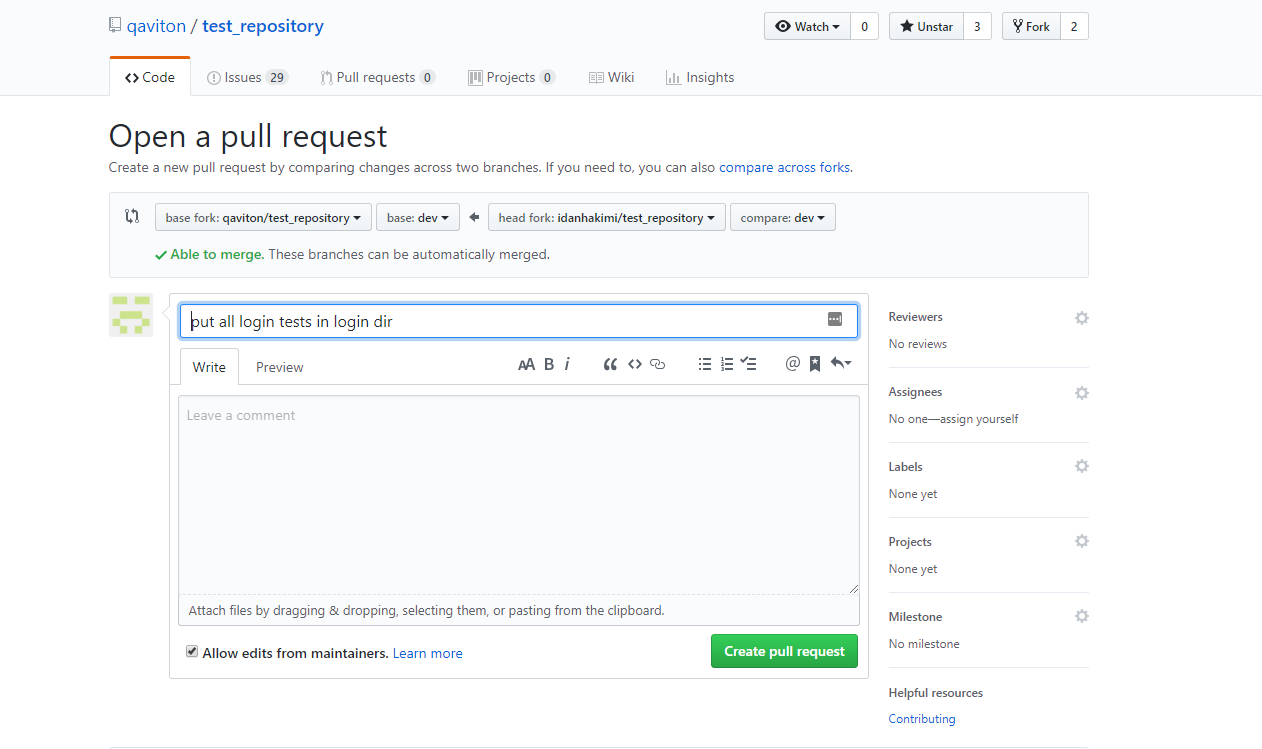


Finally we want to push the changes merged in dev to our remote url.

**Section H - Pull Request**

After we push code to our forked remote url we would ask for code review and make a pull request from our github account:





After clicking on create pull request the final steps are for the owner to approve or reject the request.

**DO NOT PROCEED WITH 'Merge pull request' & 'Confirm merge'**

