Git

ندا سلطانی

۵ فروردین ۱۴۰۰

Version Control

- Teamwork is the basis of many modern development projects.
- Often teams are distributed all over the world and over various time zones.
- Synchronizing the development work is the main task for project managers.
- Software Version Control Systems (SVCS)
- In the past, Concurrent Versions System (CVS) and Subversion (SVN) have been popular for workflows with a centralized server infrastructure.
- Nowadays, more and more decentralized workflows have been established, making use of a new SVCS called Git.

Git

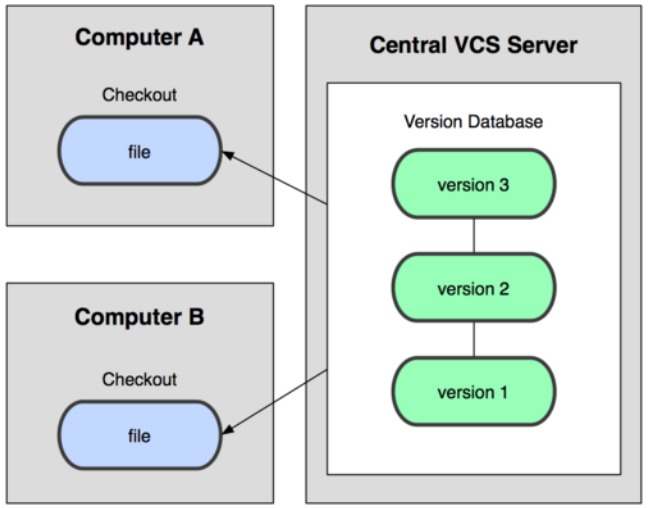
- Collaborate on code and research data
- Manage Git repositories with fine grained access controls and keep your code and your research data secure.
- Perform code and research data reviews and enhance collaboration with support even for large files (Git-LFS).
- Each project can track issues, utilize individual wiki and many more features out-of-the-box.

Git History

- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently

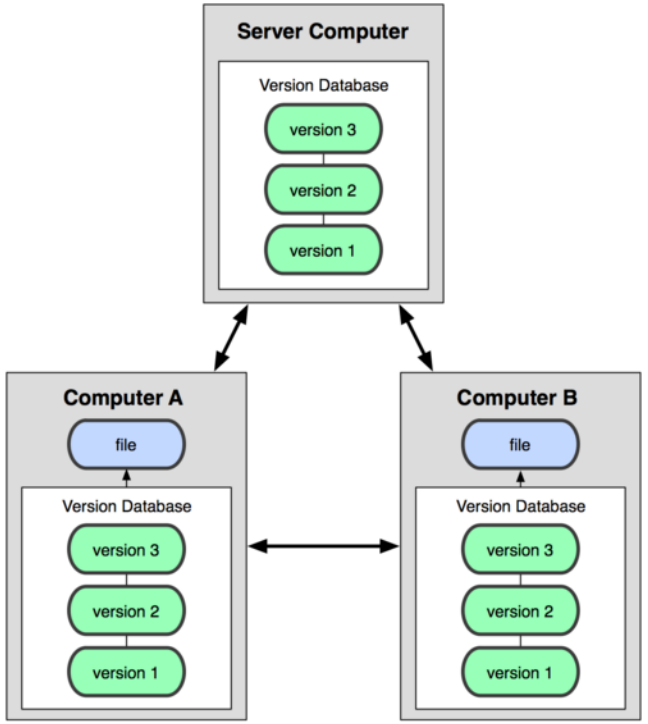
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

Distributed Model

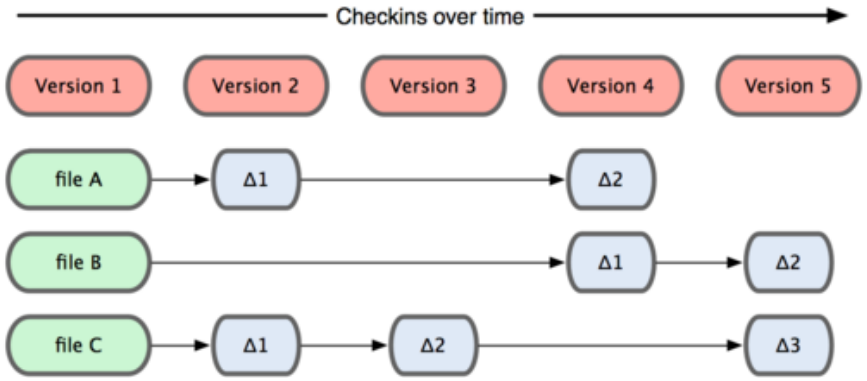


(Git, Mercurial)

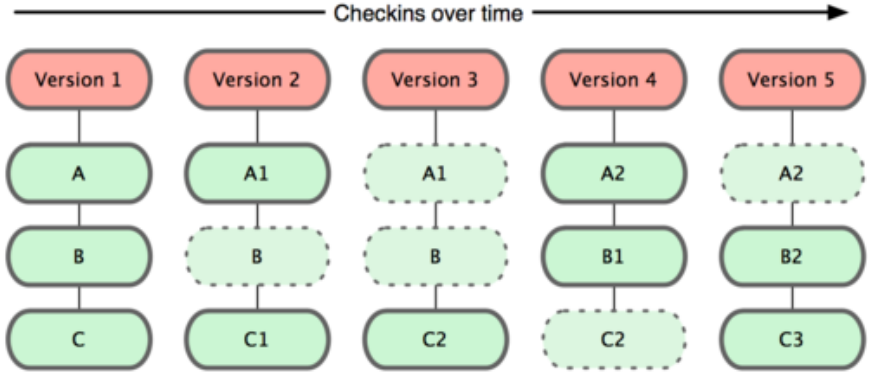
Result: Many operations are local

Git takes snapshots

Subversion



Git



Git uses checksums

- **Each user has their own copy of the repo**, and commits changes to their local copy of the repo before pushing to the central server.
- Git generates a unique SHA-1 hash - 40 character string of hex digits, for every commit.
- Refer to commits by this ID rather than a version number.
- Often we only see the first 7 characters:

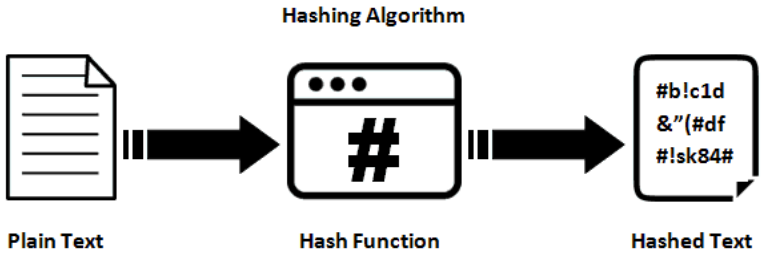
  1677b2d Edited first line of readme

  258efa7 Added line to readme

  0e52da7 Initial commit

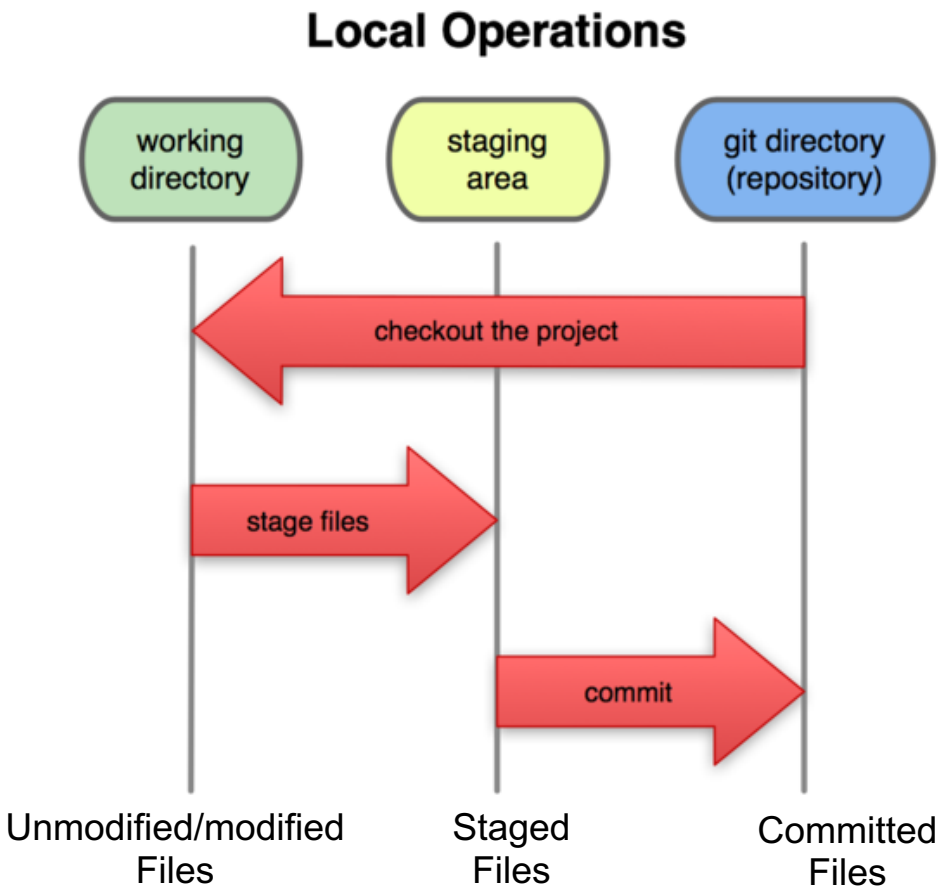
Quick review: Hash

- Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.



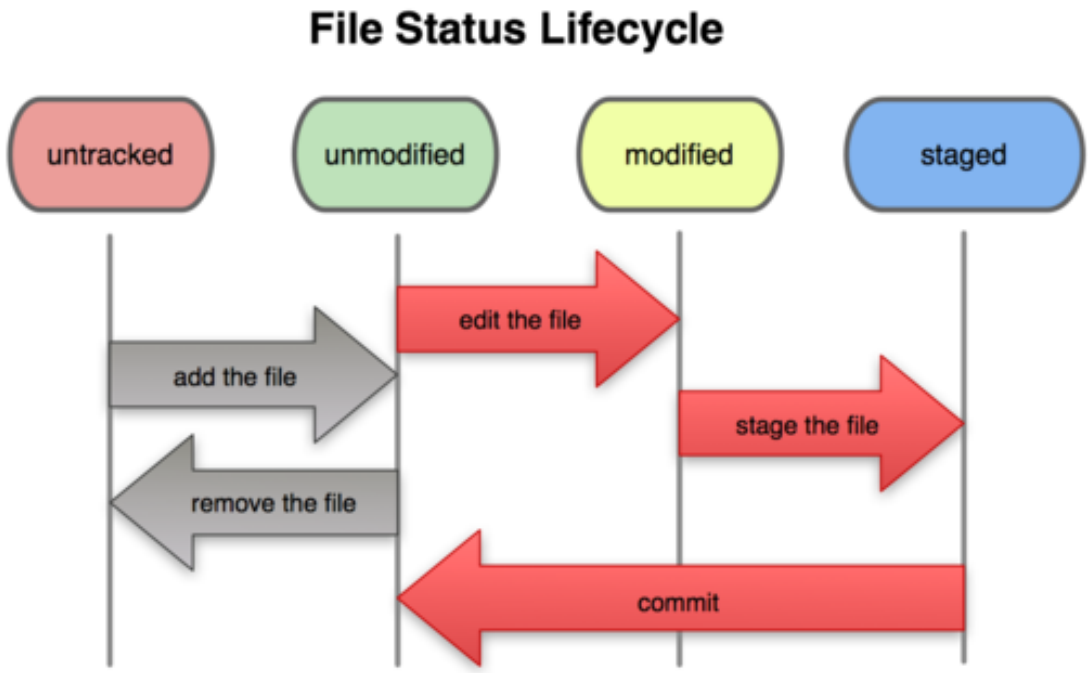
- In cryptography, SHA-1 (Secure Hash Algorithm 1)
  - a cryptographic hash function
  - takes an input and produces a 160-bit (20-byte) hash value known as a message digest
  - typically rendered as a hexadecimal number, 40 digits long.

A Local Git project has three areas



Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

Git file lifecycle

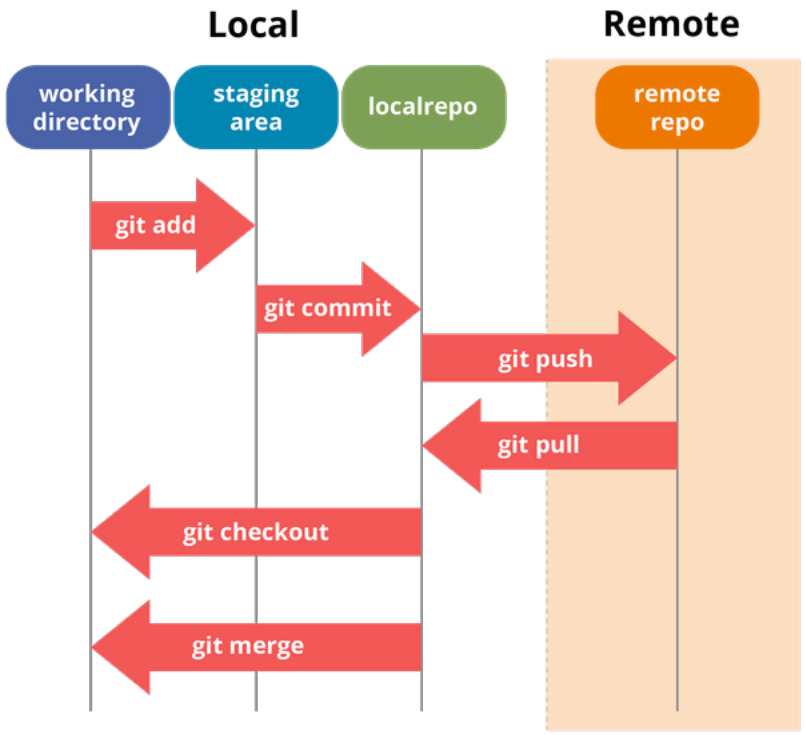


Workflow (local)

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

- Notes:
  - If a particular version of a file is in the **git directory**, it's considered **committed**.
  - If it's modified but has been added to the **staging area**, it is **staged**.
  - If it was **changed** since it was checked out but has not been staged, it is **modified**.

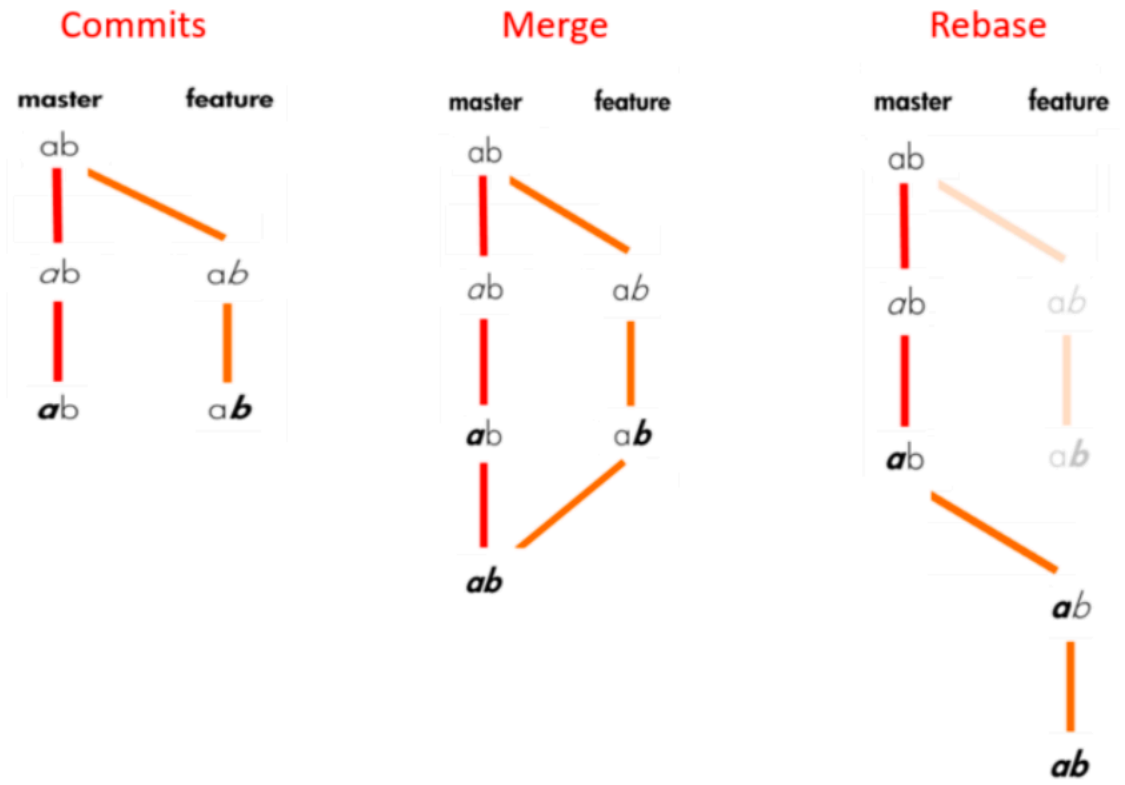
Git workflow



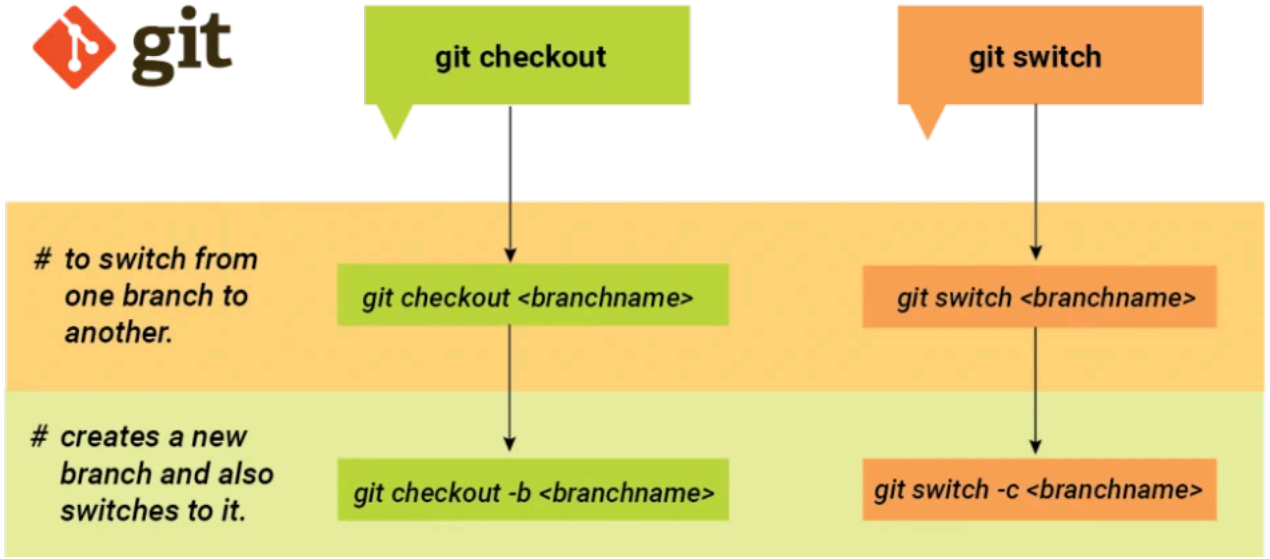
Basic Workflow (Remote)

1. **Clone** the remote repository to local directory.
2. **Modify** files in your working directory.
3. **Stage** files, adding snapshots of them to your staging area.
4. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.
5. **Pull** the changes from remote repository.
6. **Merge** the changes.
7. **Push** the committed files to the remote repository.

Merge vs. Rebase



Checkout vs Switch



Aside: So what is github?

- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects or you can pay for private projects.

Question: Do I have to use github to use Git?

Answer: No!

- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system.

Others

Installation

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

Get ready to use Git!

- Set the name and email for Git to use when you commit:

  $ git config --global user.name “Bugs Bunny”

  $ git config --global user.email bugs@gmail.com

- You can call git config --list to verify these are set.
- These will be set globally for all Git projects you work with.
- You can also set variables on a project-only basis by not using the --global flag.

Create a local copy of a repo

- Two common scenarios: (only do one of these)
  1. To clone an already existing repo to your current directory:

     $ git clone <url> [local dir name]

     - This will create a directory named local dir name, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual repo)
  2. To create a Git repo in your current directory:

     $ git init

     - This will create a .git directory in your current directory.
     - Then you can commit files in that directory into the repo:

     $ git add file1.py

     $ git commit –m “initial project version”

Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but un-staged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

Committing files

- The first time we ask a file to be tracked, and every time before we commit a file we must add it to the staging area:

$ git add README.txt hello.py

- This takes a snapshot of these files at this point in time and adds it to the staging area.

- To move staged changes into the repo we commit:

$ git commit –m “Fixing bug #22”

  - Note: To unstage a change on a file before you have committed it:

  $ git reset HEAD -- filename

  - Note: To unmodify a modified file:

  $ git checkout -- filename

- Note: These commands are just acting on your local version of repo.

Status and Diff

- To view the status of your files in the working directory and staging area:

$ git status

$ git status -s

(-s shows a short one line version similar to svn)

- To see what is modified but unstaged:

$ git diff

- To see staged changes:

$ git diff --cached

Viewing logs

- To see a log of all changes in your local repo:

$ git log

$ git log --oneline

(to show a shorter version)

  - 1677b2d Edited first line of readme
  - 258efa7 Added line to readme
  - 0e52da7 Initial commit

$ git log -5 (to show only the 5 most recent updates, etc.)

- Note: changes will be listed by commit ID #, (SHA-1 hash)
- Note: changes made to the remote repo before the last time you cloned/pulled from it will also be included here

Branching

- To create a branch called experimental:

$ git branch experimental

- To list all branches: (* shows which one you are currently on)

$ git branch

- To switch to the experimental branch:

$ git checkout experimental

- Later on, changes between the two branches differ, to merge changes from experimental into the master:

$ git checkout master

$ git merge experimental

- Note: git log --graph can be useful for showing branches.
- Note: These branches are in your local repo!

Example

- Clone a remote repo to your local repo:
  $ git clone https://github.com/nedasoltanih/maktab51 git_test
- Create a new branch in your name, then switch to it:
  $ git branch [your name]
  $ git switch [your name]
  or
  $ git switch -c [your name]
- Check branches:
  $ git branch
- Edit README.md file, then check the status and differences:
  $ git status
  $ git diff

Example...

- Add README.md to staging area:

  $ git add README.md

- Commit the changes:

  $ git commit -m “[your name] edited readme file”

- Pull recent changes from the remote repo:

  $ git pull https://github.com/nedasoltanih/maktab51

- Push your branch:

  $ git push https://github.com/nedasoltanih/maktab51

Example

- Switch to branch “neda”

  $ git switch neda

- Edit README.md file, add the file to the staging area, commit the changes, pull from remote again, push to remote.

  $ git add README.md

  $ git commit –m “some message here”

  $ git pull https://github.com/nedasoltanih/maktab51 neda

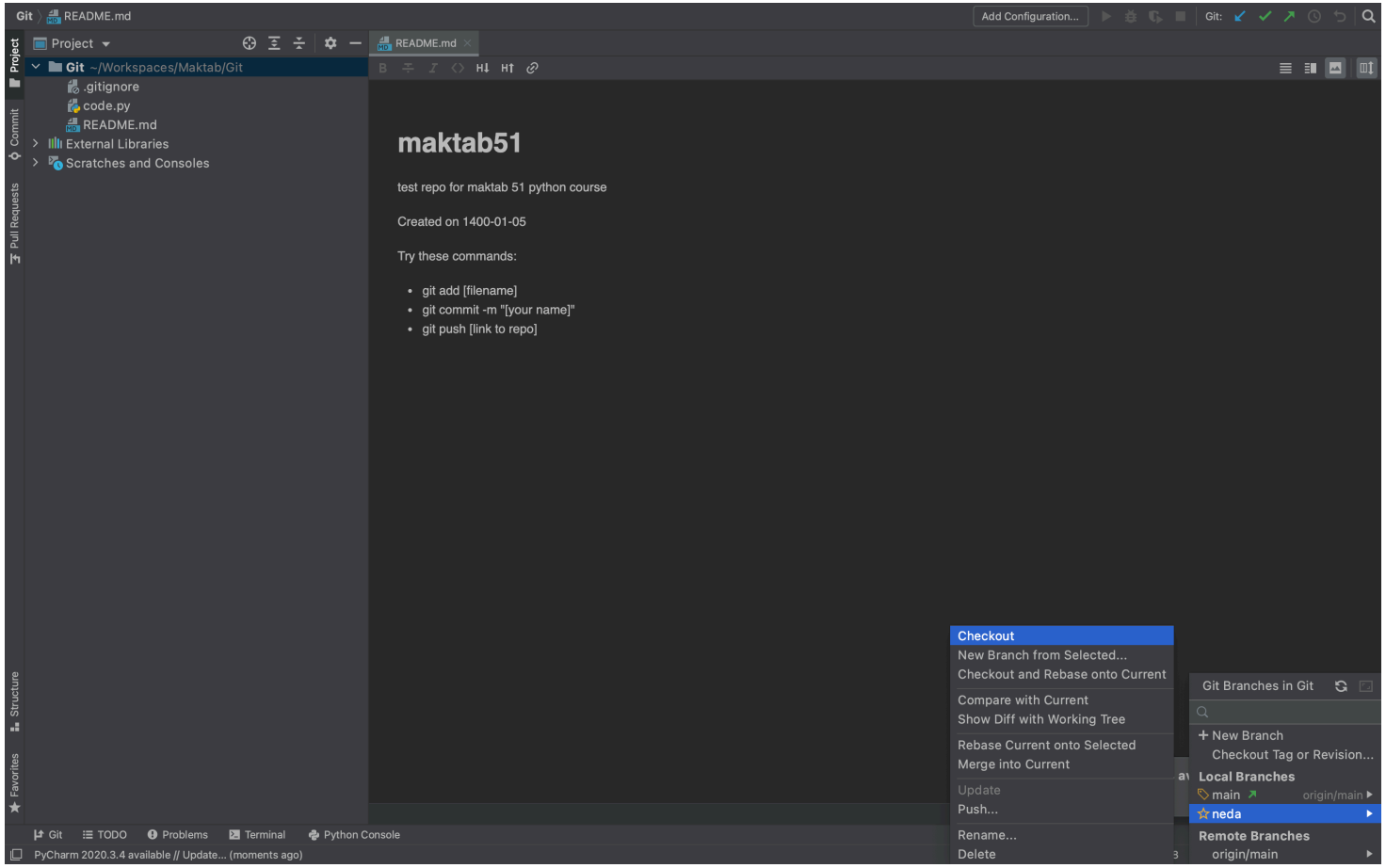
  - What happened? working on a common file? needs to be merged!

  $ git push https://github.com/nedasoltanih/maktab51

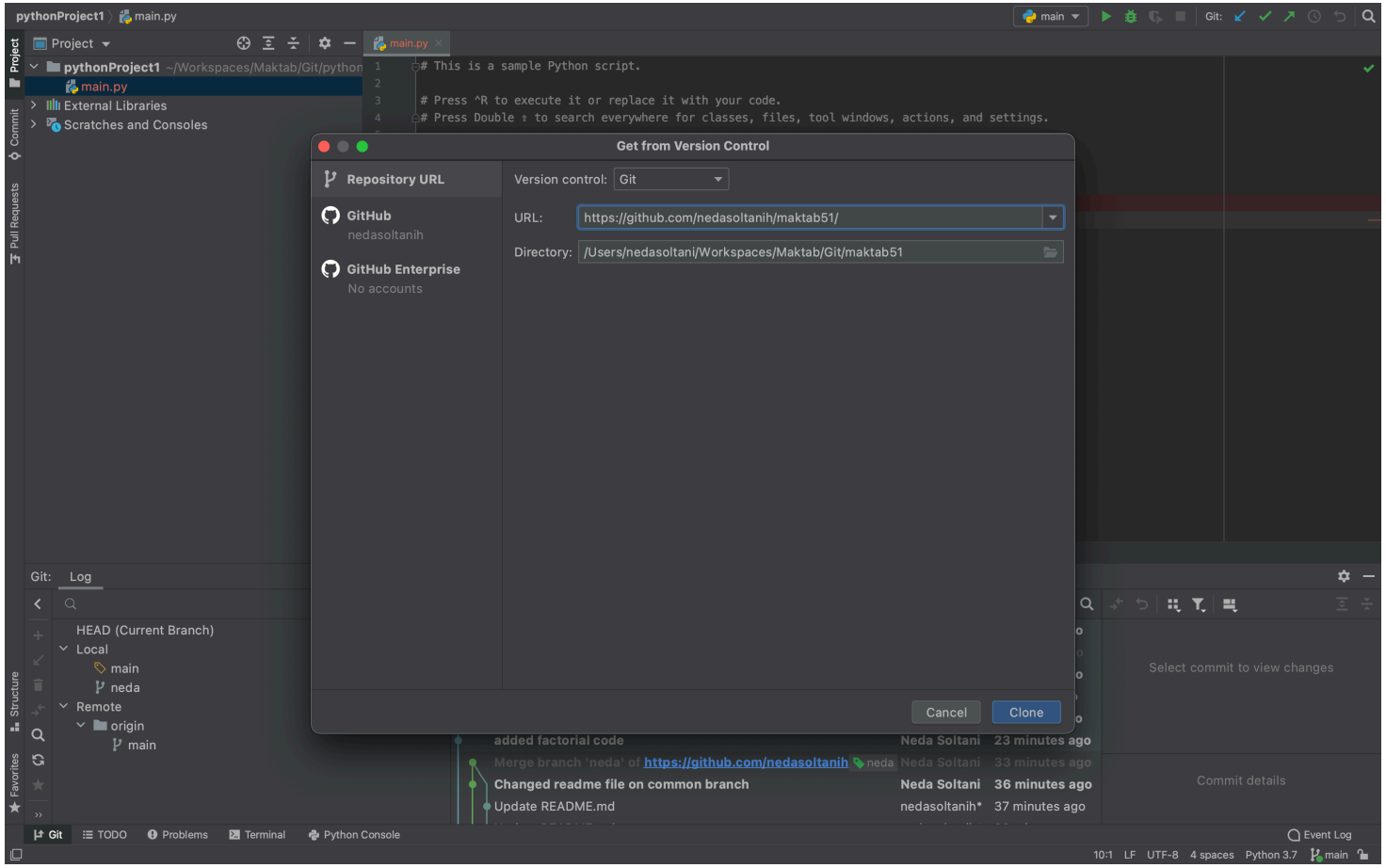
Example...

- check this link: https://github.com/nedasoltanih/maktab51/compare/neda?expand=1
- The project admin checks the commits and merges them to the main branch if approved.

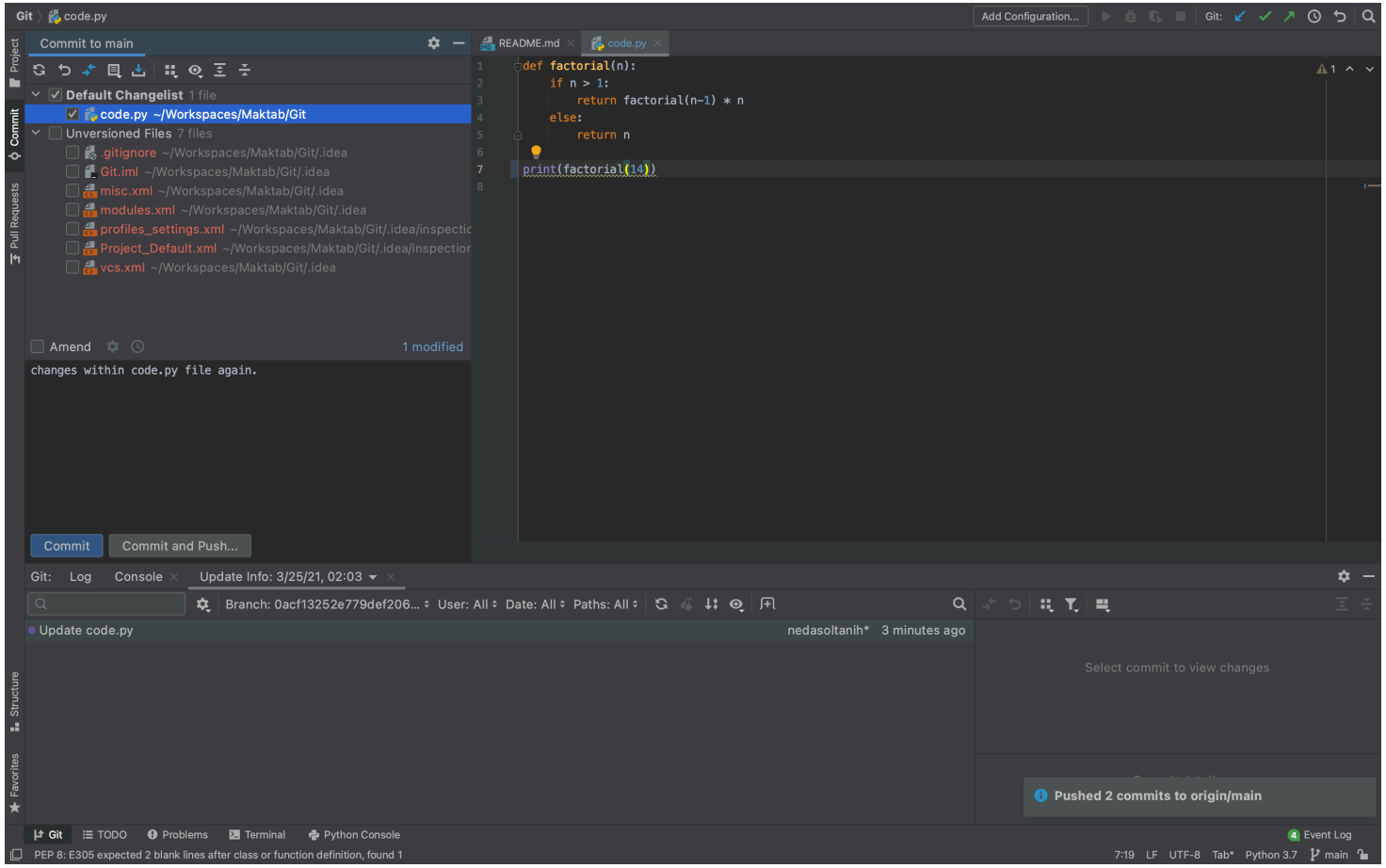
Git in IDEs: PyCharm



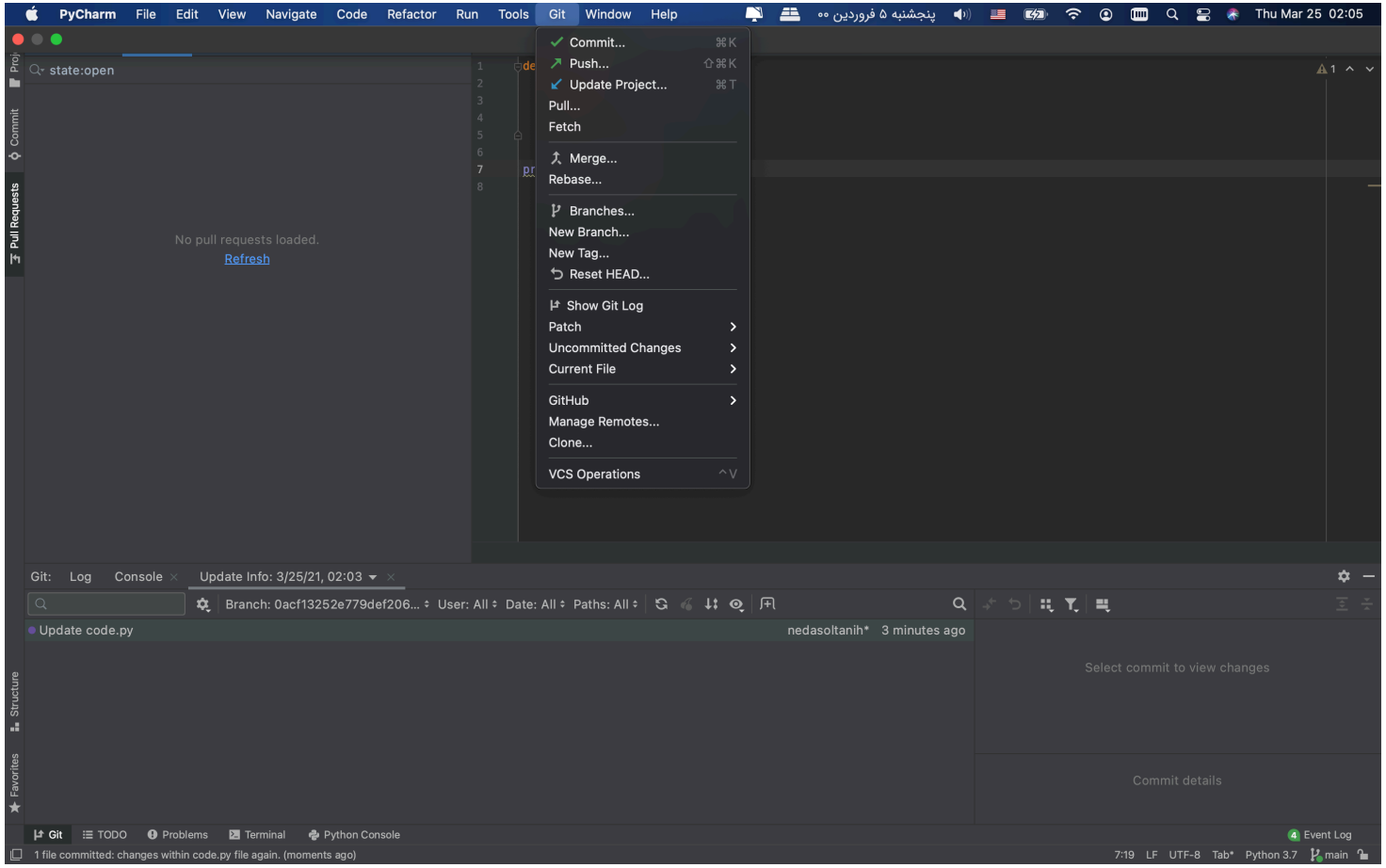
Git in IDEs: PyCharm - Clone



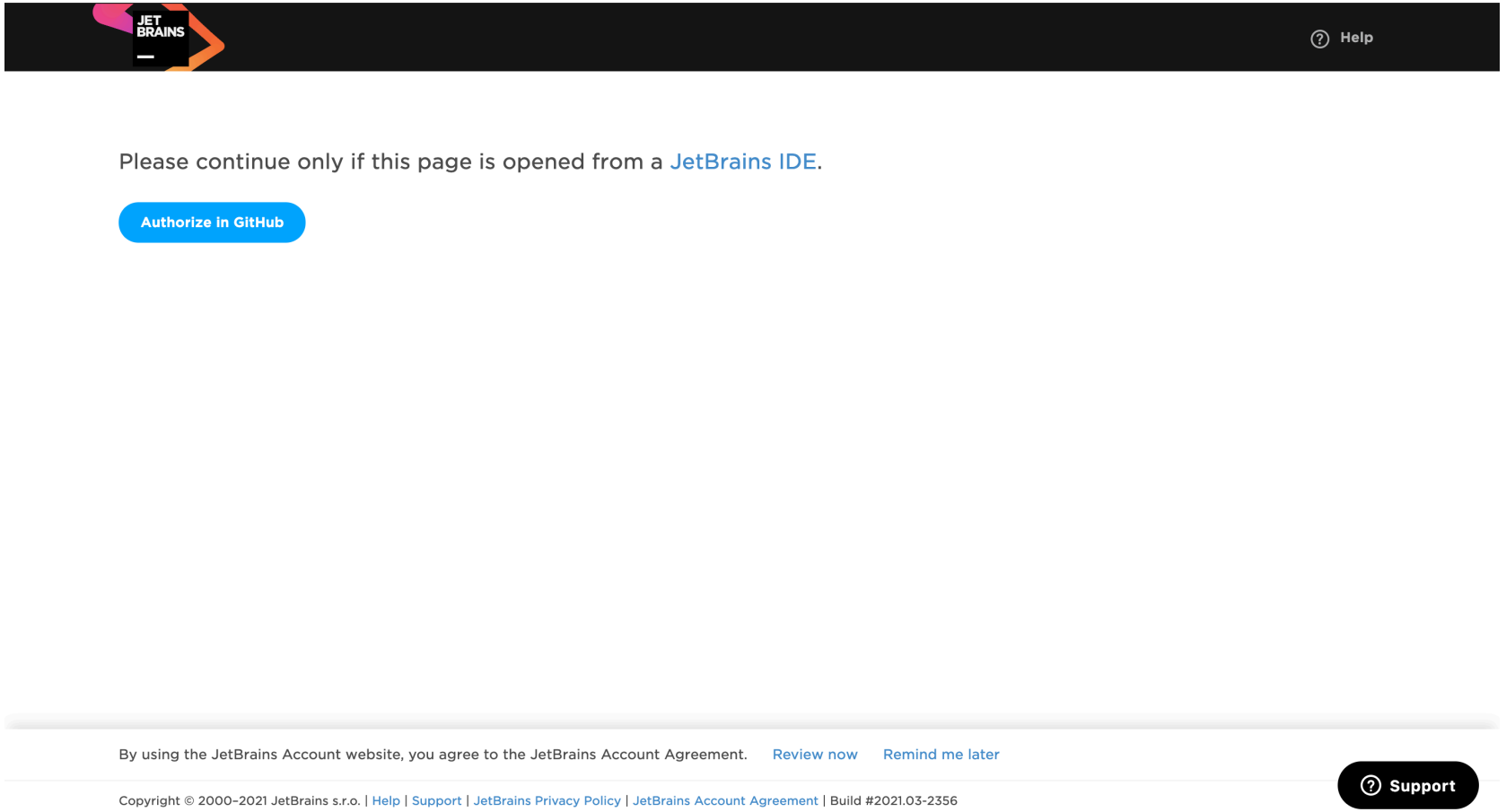
Git in IDEs: PyCharm - Commit



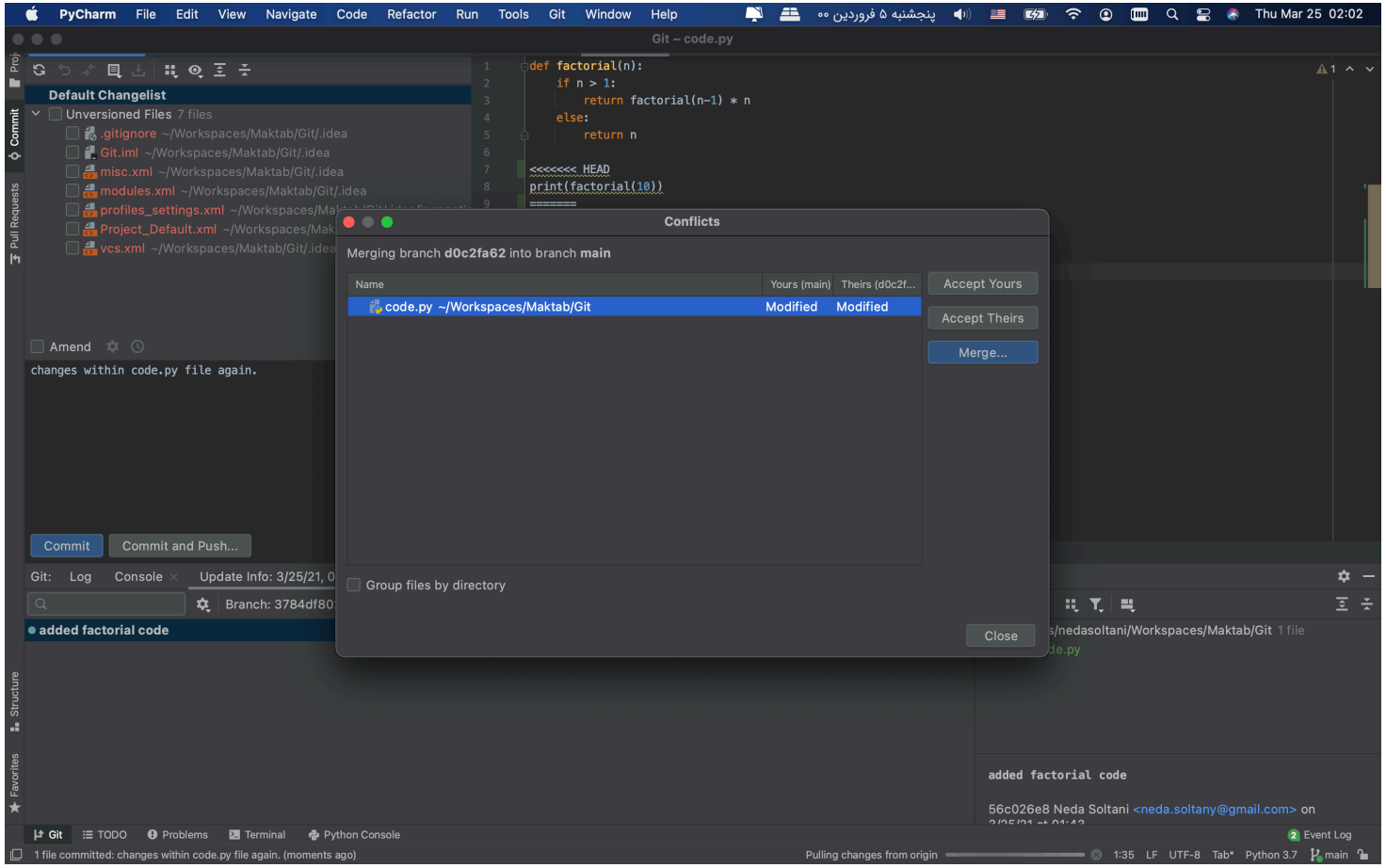
Git in IDEs: PyCharm - Pull, Push



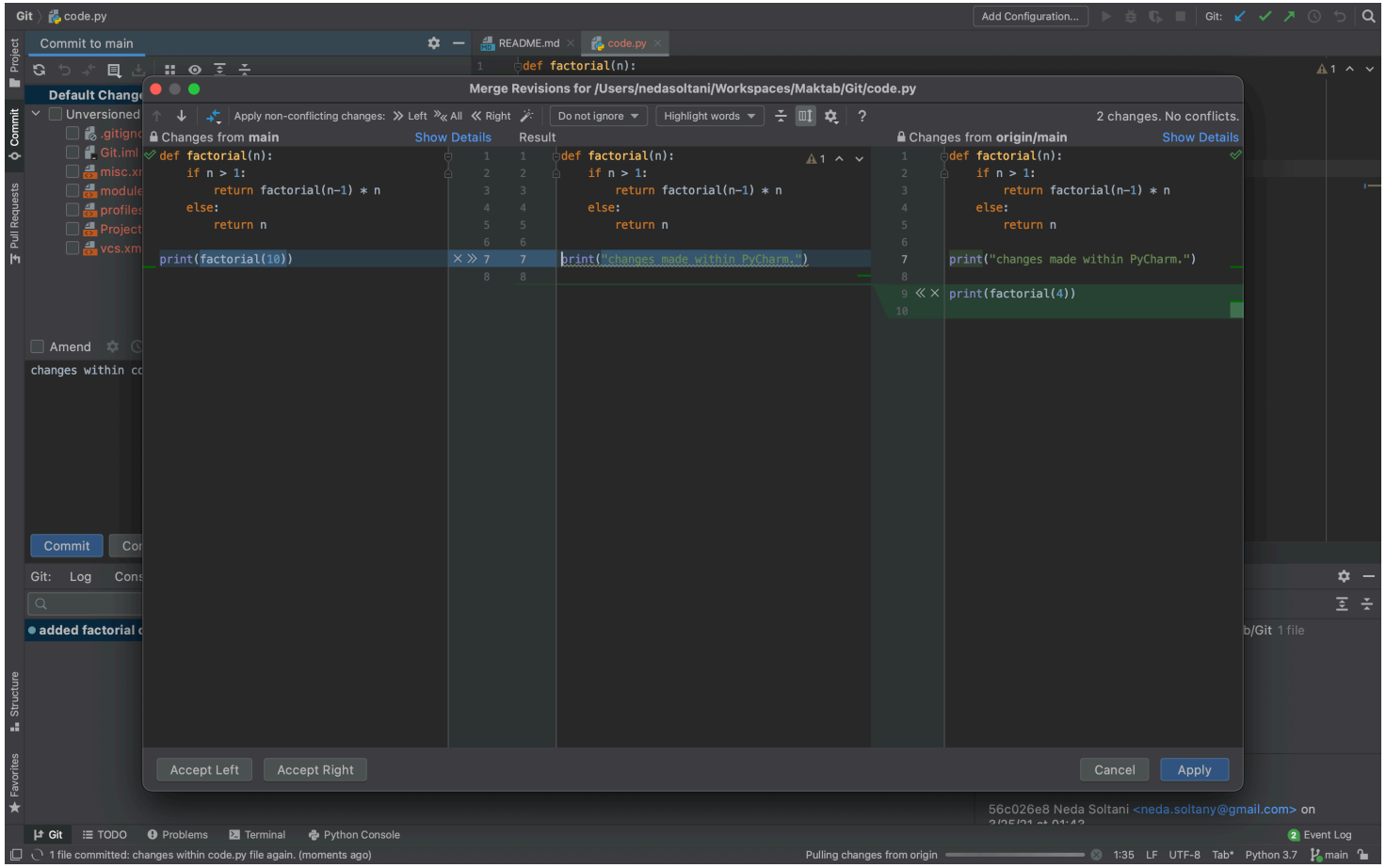
Git in IDEs: PyCharm - Login



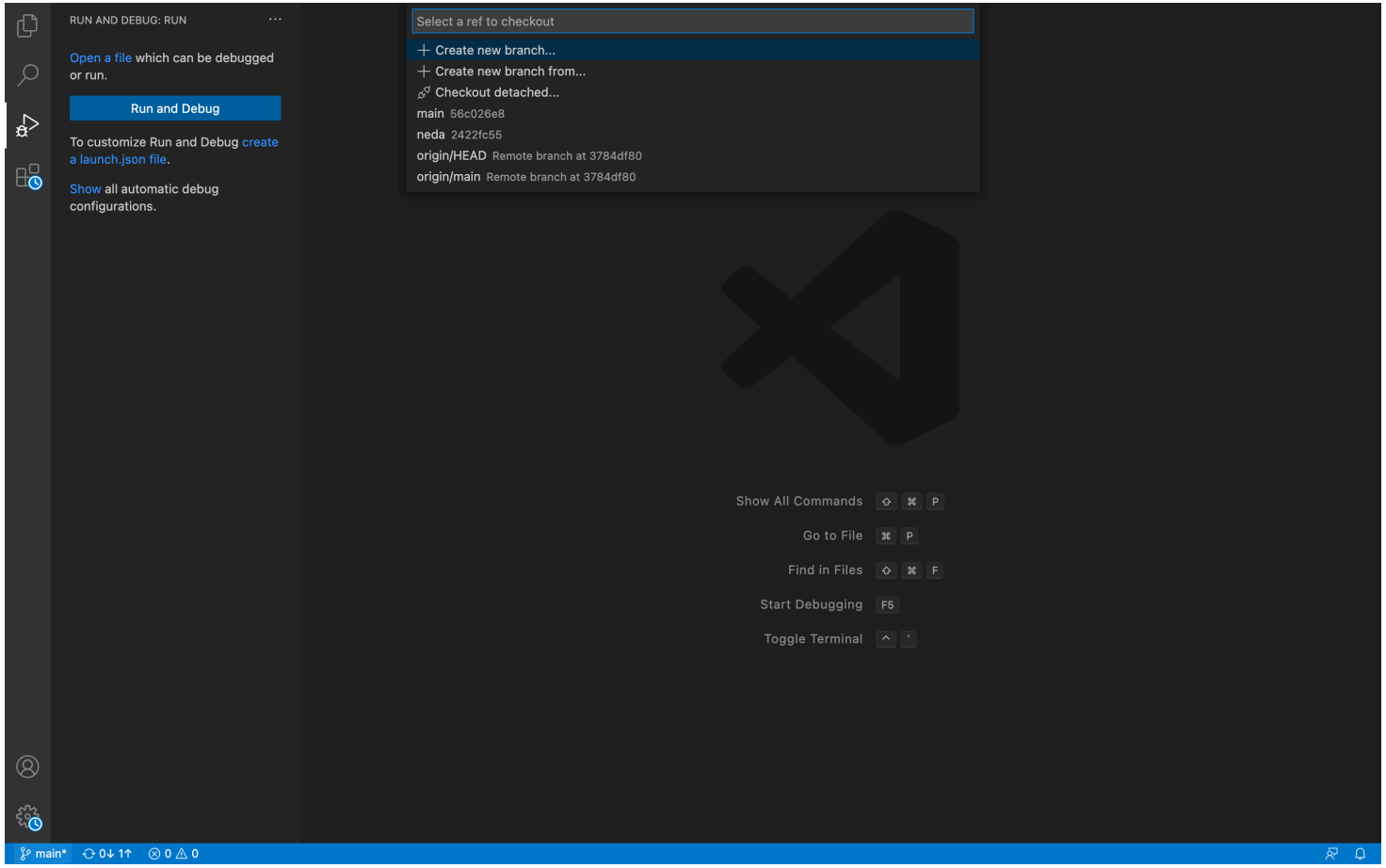
Git in IDEs: PyCharm - Conflict



Git in IDEs: PyCharm - Merge



Git in IDEs: Visual Studio Code



Practice

- Sign up in github.com
- Create a new repository for your python assignments.
- Create a new branch for each assignment.
- Push your assignment files into the repository.
- Add a README file explaining the assignment and a .gitignore file which includes unneeded files.
- Share the link with your groupmates.
- Try git within an IDE of your choice.