Exercise class 4

(week 11)

Introduction to Programming and Numerical Analysis

Class 4 and 8

Rosa Haslund Meyer

Spring 2024

KØBENHAVNS UNIVERSITET

Git

Solving equilibrium models

Inaugural Project

Poll

Git and GitHub

Git is a **version control system** for tracking changes made to your files typically during some sort of programming task/software development.

You can use Git to keep track of changes made to any type of document from Microsoft Word to code bases.

GitHub is a web-based Git repository hosting service/cloud-based platform for hosting code – basically like **Dropbox for code**.

Git repositories

A repository is just like a folder that Git keeps track of. You can create Git repositories in different ways:'

1. Initializing a repository in an existing folder on your computer
   - In VScode: Ctrl+Shift+P / cmd+shift+p (windows / mac) and type *Git: Initialize Repository*
   - Choose the folder you want

2. Clone an existing repository from GitHub
   - … Like when you cloned the lectures repo etc.

cloning – local copy vs. forking – own copy

Committing changes

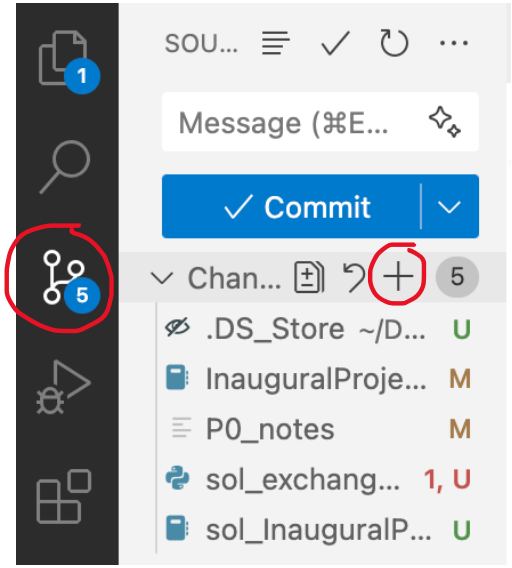
Git has multiple stages of saving your files:

- **Working directory**: the current version of the files on your computer
- **Staging area**: changes that are ready to be committed
- **Committed changes**: checkpoint or “snapshot” of the state of the repository

Committing changes

When you've made changes to a file, that you'll like to keep:

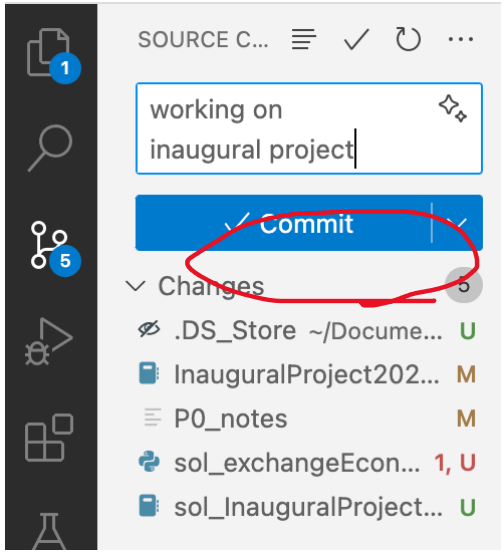
- Save the file (locally) on your computer: Ctrl+S / cmd+S (Windows / Mac)
- Stage the file – in VScode:



Committing changes

When you've made changes to a file, that you'll like to keep:

- Save the file (locally) on your computer: Ctrl+S / cmd+S (Windows / Mac)
- Stage the file – in VScode:
- Commit the file – remember to always enter a commit message!



- You can also commit new changes from GitHub by Ctrl/cmd+Shift+P and type *Git: commit*.

Uploading to GitHub

You can easily publish your local git repositories directly from VScode: Ctrl/cmd+Shift+P and type Git: *Publish Branch.*

If you cloned your repository from some GitHub repo, it's already published, and you are ready to upload your changes (provided you have writing access!).

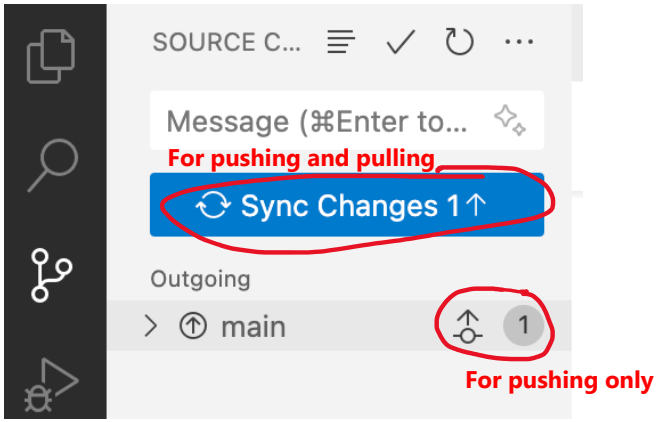
GitHub works in the same way as eg. OneDrive or Dropbox: you store your files in the cloud and then you can **synchronize** them to you own computer (locally).

NOTE! GitHub doesn't sync automatically – you must manually upload, download and sync to make changes to files.

Uploading to GitHub

You can upload changes to GitHub by Ctrl/cmd+Shift+P and type *Git: Push*. This will push all **committed** changes to the online GitHub repository.

You can also use the blue bottom (kind of like before) in VScode, which generally will **push and pull** online changes. You can also choose to push without pulling:



Downloading from GitHub

Just like pushing changes you can pull new changes from GitHub by Ctrl/cmd+Shift+P and type *Git: pull*.

NOTE! You can't do this if you have uncommitted changes in your local repo. Before "pull" you should always either **commit or discard your local changes**.

Bonus info: the "pull"-command is a combination of two other git-commands, namely "fetch" and "merge". The "fetch"-command loads changes from the cloud and "merge" merges them into the working directory. Advanced users sometimes prefer to do these steps manually, but for now it's enough for you to just use the "pull"-command to download changes.

Merge conflicts...

Merge conflicts occur when competing changes are made to the same line of a file, or when one person edits a file, and another person deletes the same file.

Thus, if your version of a file looks different than the corresponding file in the cloud, a **merge conflict** can occur when trying to sync!

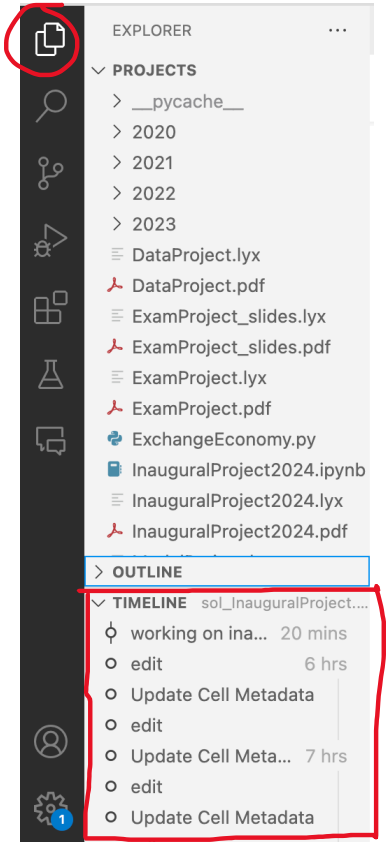
This can happen for example when two people are simultaneously working in the same file.

If you get a merge conflict, you need to resolve it before you can complete the sync. You must choose which changes to incorporate from the different branches in a new commit:

1. Choose the changes you want to merge into your working directory
2. Commit the merged changes

Version control (advanced)

VScode allows you to view the history of a file and in that way the history of the changes:



Version control (advanced)

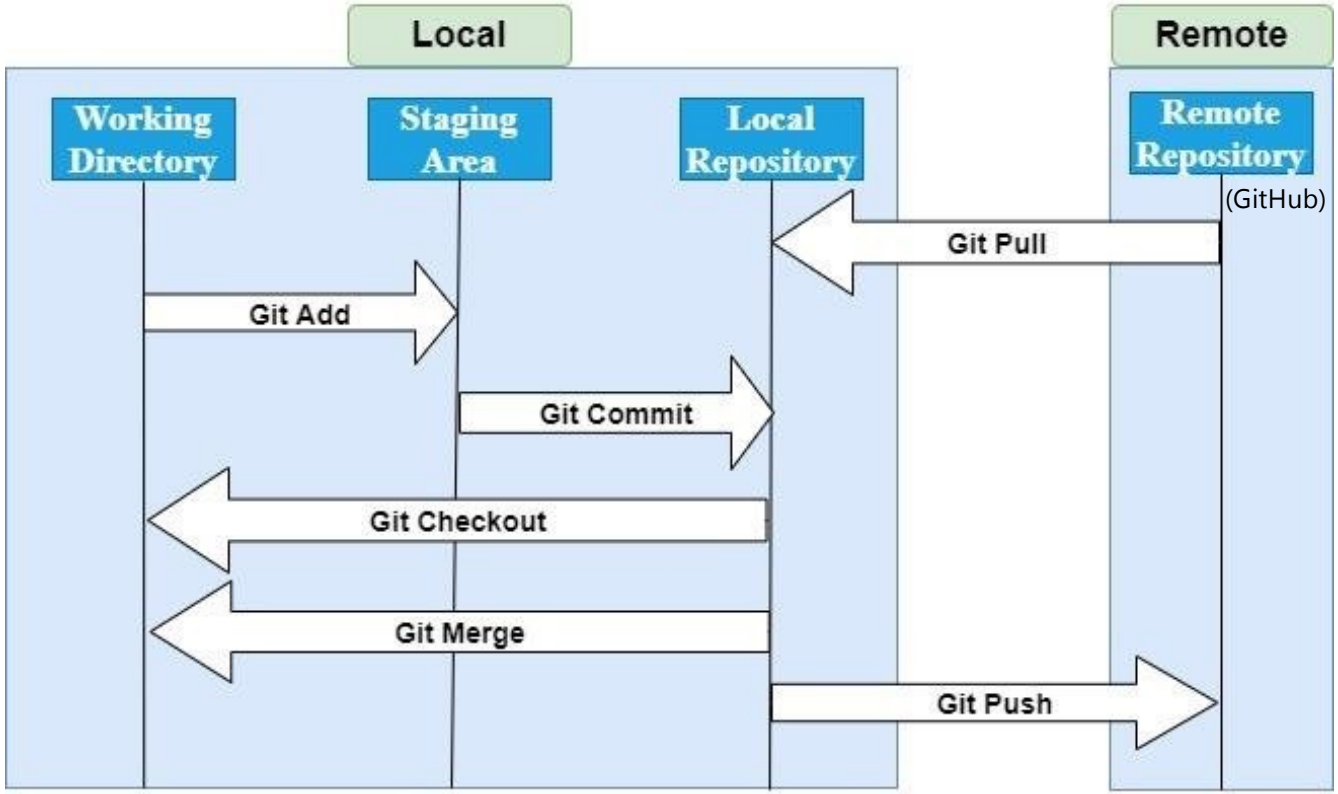
Git has a lot of different functionalities – for example:

- Create different branches to work on specific tasks and the **merge to the main branch**
- **Checkout** different branches while preserving the state of your own repository
- **Revert** to previous commits, if you made mistakes along the way or for some other reason want to go back to a previous version
- … and much more

There exist quite a few apps to make it easier to interact with Git. Personally, I use GitHub Desktop and I would recommend you do the same.

Check out this video for how to do version control in VScode (if interested).

Git workflow



Important tips for Git

If you're working in a group, **always sync** before making any changes.

Try not to work in the same file at the same time – likely that merge conflicts will occur!

Commit whenever you've added something to your code base – it's way easier navigating the commit history if there are few changes to each commit.

Remember to comment your commits well - might come in handy later!

In short…

Skills you'll need to use Git and GitHub:

- Staging and committing changes
- Pushing to GitHub
- Fetching and merging (aka. pulling) from GitHub
- Handling merge conflicts – you can read a bit more on how to solve here.

Some other nice knowledge you can check out if you want:

- Reviewing commit history (git log)
- Checking out previous commits (git checkout)
- … branches, merges stashes and many other version control tools

Solving the consumer problem

The consumer problem generally has the form

$$\max_{\boldsymbol{c} \in C} U(\boldsymbol{c})$$
$$s.t. \quad \text{constraints}$$

You'll need to know **constrained optimization**.

Revisit the lectures on **Optimization**, **The Consumer Problem** and/or **problem set 1**. You can also see the notebook on **optimization of the consumer problem** in our class repository, in the folder *Exercise_2*.

Solving equilibrium models

Models with both a **supply** side and **demand** side.

The solution is a **set of prices** that makes the supply **equal** to the demand **in all markets**.

Demand for consumption goods usually stems from the solution to some **consumer problem**.

Supply of consumption goods can for example be a production firm or consumers exchanging/trading their endowments.

Solving equilibrium models – standard approach

Solving models like these can typically be done by following these steps:

1. Find demand as a function of prices
2. Find supply as a function of prices
3. Find excess demand as a function of prices (demand-supply)
4. Find the point where excess demand equals zero (or close)
5. Finally, find the solution if the price vector ensures point 4.

Walras' law: if you have N markets you only need to do this for N-1 goods – the final will clear by Walras' law.

Revisit lectures on **Random Numbers – example** and **Production Economy** and/or **problem set 2** (take a closer look at the extra problem whit N>2 markets).

The Inaugural Project

The Inaugural Project evolves around an Edgeworth economy (exchange economy) with two agents and two goods.

Useful skills:

- Functions and classes (remember a class can always be written as a bunch of functions)
- Optimization using grid search and/or optimization using solvers, (un-)constrained
- Printing and plotting
- General practical coding knowledge and good code practice

The Inaugural Project

The deadline for hand-in is **March 24th**, and the deadline for peer feedback is **March 31st**.

Hand-in on GitHub by uploading it to the GitHub repo:
github.com/NumEconCopenhagen/projects-YEAR-YOURGROUPNAME/inauguralproject

Your hand-in must consist of:

- A README.md file with an introduction to your project (and optionally usage instructions)
- A notebook (.ipynb) presenting the analysis, i.e. presenting and discussing your results
- A fully documented Python file (.py) – do this based on the provided ExchangeEconomy.py-file

The Inaugural Project – general tips

This project requires you to:

1. Find the market equilibrium
2. Optimize utility under different conditions/restrictions

Think about **constrained optimization**, the **consumer problem** and **equilibrium models**.

Note: the questions are (mostly) written in a way where **you can move on to the next question if you get stuck** – don't hesitate to do this if needed!

The Inaugural Project – general tips

Don't forget your **economic intuition** when programming!

Print and illustrate your results and relate them to theory – this way you'll know if the economic interpretations makes sense.

Try to implement answers to all questions! If your code doesn't run or work at all, write down **your way of thinking**, **what you tried** and **what might have gone wrong**. This shows your problem-solving skills and can also give some points! :)

You can leave your code to show your process but please **comment it out by using #**.

Make sure that your notebook runs from top to bottom **without errors** before the hand-in.

Lastly, remember good coding practice! I've uploaded a .ipynb-file about this.

Time to get started!

The Inaugural project:

- Q1: Edgeworth box
- Q2: Excess demand, error in market clearing conditions, Walras
- Q3: Market clearing price
- Q4a-b: Finding allocations under certain conditions for prices
- Q5a-b: Finding allocations under certain restrictions for choice set, etc.
- Q6a: Finding yet another allocation and
- Q6b: Compare and illustrate allocations (Edgeworth) and discuss pros and cons
- Q7: Plot a random set with 50 elements
- Q8: Market equilibrium and Edgeworth box

Poll!

I want to know your opinion on the exercise classes so far using this poll!

Next time...

Video lectures:

- Data basics
- Data: Loading, cleaning and saving data

Exercises – Problem set 3. Working with data from Denmark Statistics:

- Pandas DataFrame
- Creating/adding new variables
- Summary statistics
- Indexing
- Dropping
- Renaming, cleaning, saving, etc...