Enhancing Computational Power 1: Version Control with Git

CS

Contents

What problems do VCS solve?

- Code revision
- Code collaboration
- Last known good
- Refactoring code
- Accountability
- Auditing

Code Revisions

- The core of a version control system is to track changes to code.
  - What changed,
  - in what file,
  - by whom,
  - when was it changed.

Collaboration

- We need to save, share and merge codes with collaborators.
  - We use GitHub as a remote server to save, share and merge codes with your collaborators.
  - The underlying technology is Git (Global Information Tracker), which allows multiple user to work on a file at the same time.

Last known good

- Git allows us to tag code: i.e., tag specific points in a repository's history as important.
  - People use it to mark release points (v1.0, v.2.0, or so on).
- It is helpful to know what version of code is working.
  - If something breaks, we can revert to the last known good configuration.

Refactor Code

- Git has a feature branch that is helpful for refactoring code or working on code before it is ready for release.
  - We can make changes to the project without breaking what is running in production.

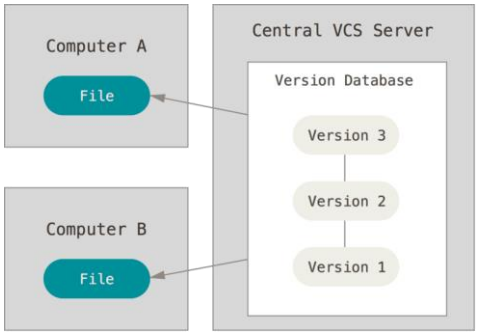
Accountability & auditing

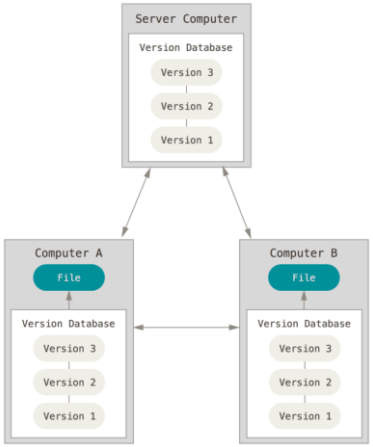
- Most companies have legal or institutional requirements around auditing and accountability. Git allows us to keep track of this.
  - Who changed this?
  - What has been changed?
  - Why this change?

Git: A Distributed Version Control System

- Git is a distributed version control system (VCS) that allows multiple developers to work on the code at the same time.
  - A centralized VCS can only allow one developer to work at the same time.

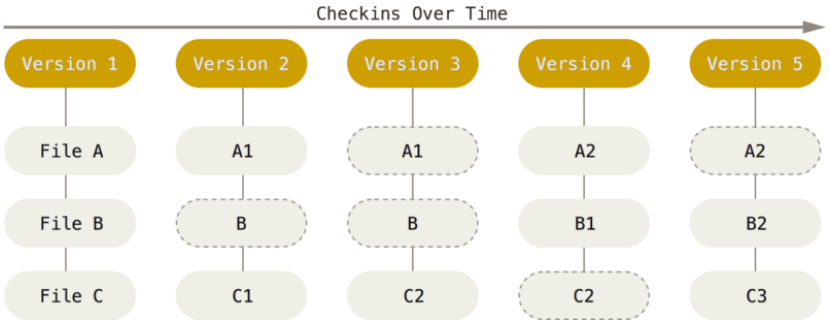
- Centralized vs. Distributed VCS





How Git Works

- Git thinks about data like a stream of snapshots of the file system.
  - Every time you commit (save) a file, Git takes a picture of it at that moment and store a reference to that snapshot (i.e., commit id) in the local repository.



  - Nearly every operation is local
    - Don’t rely on the network, no time lag due to network latency.
  - Integrity
    - Impossible to change contents without Git knowing about it.

Git Workflow – 3 States

- Git works with three stages in a local machine:
  - **Working Directory**: a folder where you work on a project.
  - **Staging Area:** a temp area that stores information about what will go into the local repository (also called as 'index').
  - **(Local) Repository**: an area where Git stores different versions of your project and historical changes.

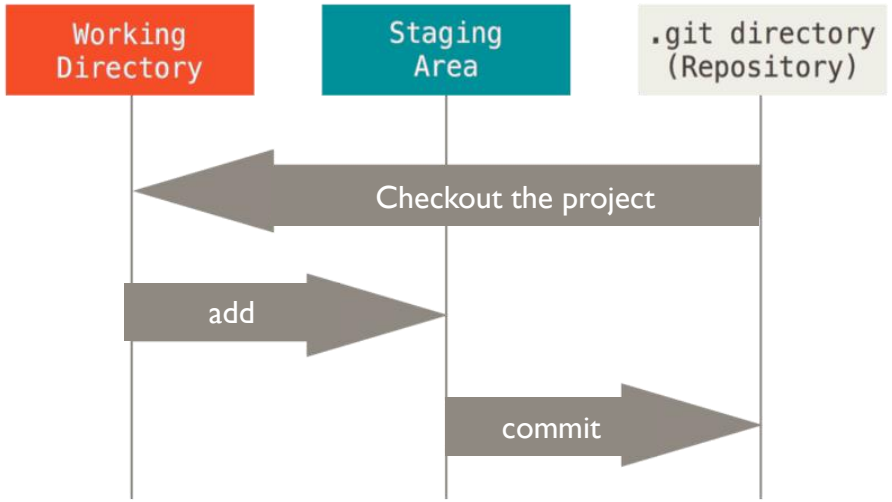
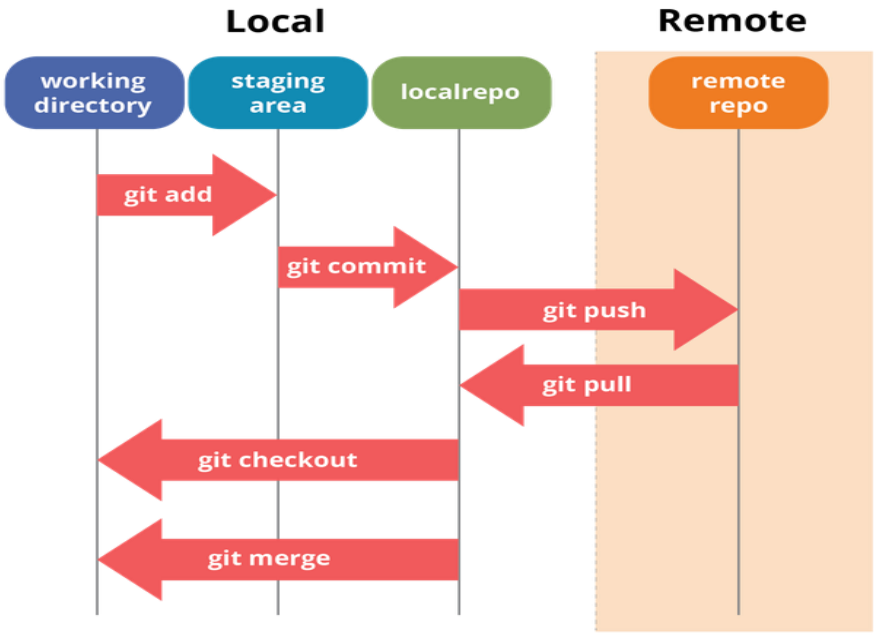


Figure 6. Working directory, staging area, and Repository

GitHub

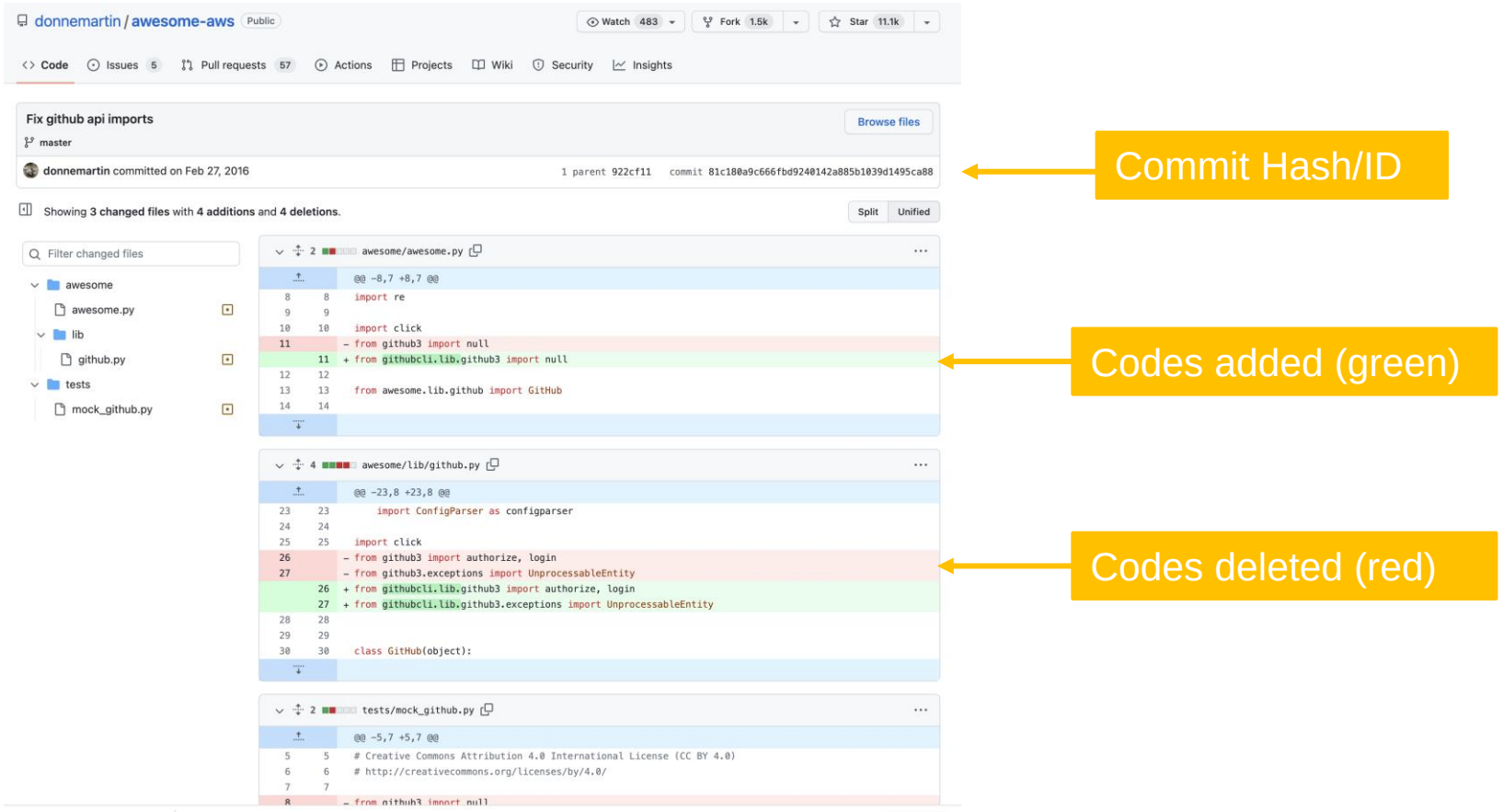
- GitHub is a website (i.e., a remote server) which hosts Git repositories online.
  - Millions of open-source projects use GitHub for repository hosting, sharing, issue tracking, code review, etc.



How Git works locally and remotely?

Check Commit Records on GitHub

- Example: a commit record from a remote repo awesome-aws.



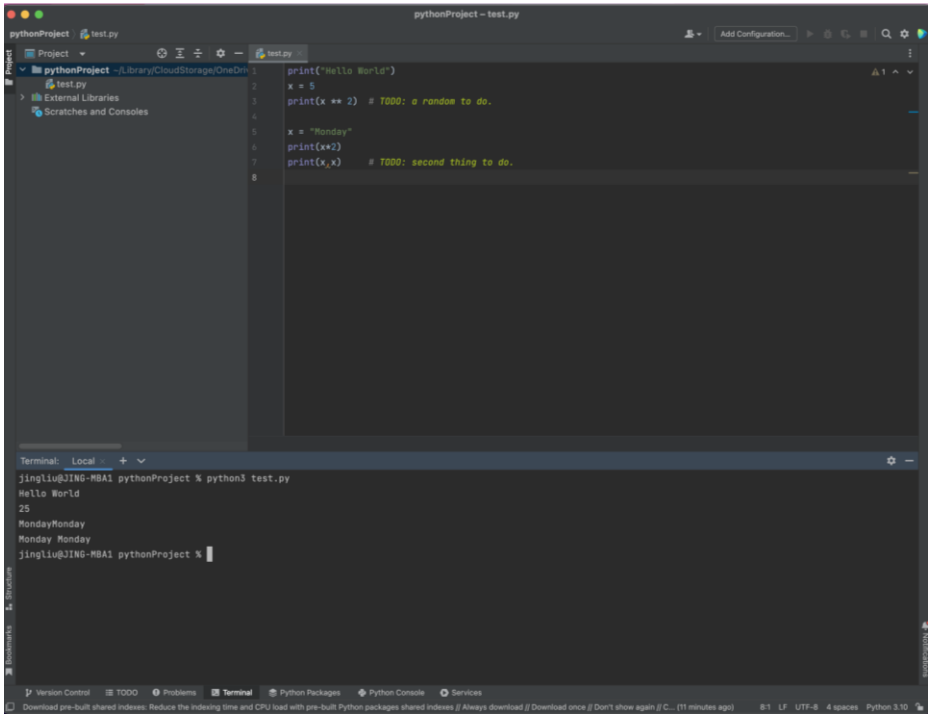
Install Git

- Download an EXE file for Windows
  - https://git-scm.com/download/win
  - Git-2.43.0-64-bit.exe
- Download for macOS (with homebrew)
  - https://git-scm.com/download/mac
  - brew install git (need to install homebrew first)
- Check the version of Git in terminal/Command Prompt
  - git --version

Git commands displayed in orange color

Work with IDE

- While Git commands are executed in Terminal/Command Prompt, we can use Python IDEs (e.g., **PyCharm** or Jupyter Notebook) for project developing and version control.
  - We will be using PyCharm for demonstration.



Demo: First-time Setup

- Tell Git who you are
  - git config --global user.email “<Your Email>”
  - git config --global user.name “<Your Name>"
- Check the configuration setting
  - git config --list
- Remove the configuration
  - git config --global --unset-all user.email
  - git config --global --unset-all user.name
- Rename the default branch as main
  - git config --global init.defaultBranch main Git’s default branch is master, while GitHub’s has changed to main since 2020.

Required input displayed with <>

Demo: Initialize a Repository

1. Navigate to your project folder
   - pwd (on mac) | cd (on win) check current working directory
   - cd <target working directory> navigate to your project folder
     - When open terminal in PyCharm, you will be located in the project folder by default.

2. Initialize an empty Git repository in working directory
   - git init create a hidden folder named **.git** in your project folder, which contains all your repo files.
   - git status display the current state of your Git repository.
     - Which branch are you on? Anything to commit?

Exercise 1: Quick Setup (15 Mins)

1. Install Git and check its version in Terminal/Command Prompt.
   - git --version

2. Tell Git who you are.
   - git config --global user.email “<Your Email>”
   - git config --global user.name “<Your Name>"

3. Check your configuration setting.
   - git config --list

4. Create a project folder econ7035, initialize a Git repository there.
   - git init

5. Check the current state of this local repository.
   - git status

Record Changes to the Repository

- Each file in your working directory can be either:
  - Tracked: files that Git knows about.
  - Untracked: files that Git don't know about.

- Each time a tracked file reaches a state you want to save; you commit a snapshot of the changes into your local repository.

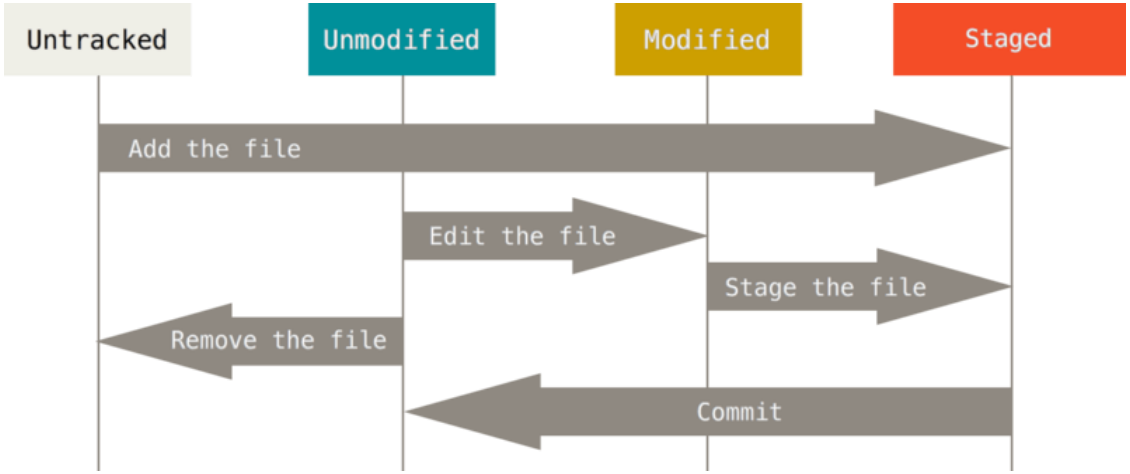


Figure 8. The lifecycle of the status of your files

Demo: Record Changes

1. Create a **README file** in project folder.
   - echo “My Project” > README.md alternatively, create a new file by right clicking on project folder.

2. Check the status of the local repository.
   - git status you will see ‘Untracked files:”

3. Ask Git to track the README file (i.e., stage the change).
   - git add <file name> you will see “Changes to the committed:”
   - git add README.md

4. Commit the change to the local repository.
   - git commit –m “<msg>”
   - git commit –m “Add README”

Required input displayed with <>.
Example of commands displayed in green.

Demo: Record Changes

5. Check the status again
   - git status you will see “nothing to commit, working tree clean’.

6. Make some changes in the tracked file, check status again
   - git status you will see “Changes not staged for commit’.

7. When the changes in a tracked file is ready to be saved, commit them again.
   - git commit –m “<msg>” <file name>
   - git commit –m “first change” README.md

Alternately, separate the command into two lines:
- git add README.md
- git commit –m ‘first change’

Stage & Commit Multiple Files

- With multiple files, you can add and commit them together.
  - If not tracked yet:
    - git add <file1> <file2> <file3>
    - Git commit –m “<msg>”
  - If the files are tracked already:
    - git commit -m “<msg>” <file1> <file2> <file3>

- If you’d like to start version-controlling all existing files.
  - git add .
  - git commit –m “initial commit”

Always check git status before staging & committing all.

Demo: Check Change History

8. View all committed snapshots in local repository.
   - git log

```
jingliu@JING-MBA1 Demo22 % git log
commit 31001ec593ad3ff6d9799885a5d216bb1ecca8cf (HEAD -> main)
Author: Jing Liu <jingliu@hkbu.edu.hk>
Date:   Fri Jan 19 15:51:28 2024 +0800

    changes in project 1&2

commit 9a8b33128f6b84f7f27c888410065706add812f7
Author: Jing Liu <jingliu@hkbu.edu.hk>
Date:   Fri Jan 19 15:50:02 2024 +0800

    two projects created
```

Branch Name

Commit Hash/ID

Commit Message

Demo: Undo Changes

1. Create and track a new file **project1.py** in project folder.
   - git add project1.py
   - git commit -m ‘add project1’

2. Write some codes, commit the changes.
   - git commit -m ‘initial analysis’ project1.py

3. Undo a commit (local data will be changed).
   - git revert <hash> --no-edit undo a commit and and commit the revert step automatically

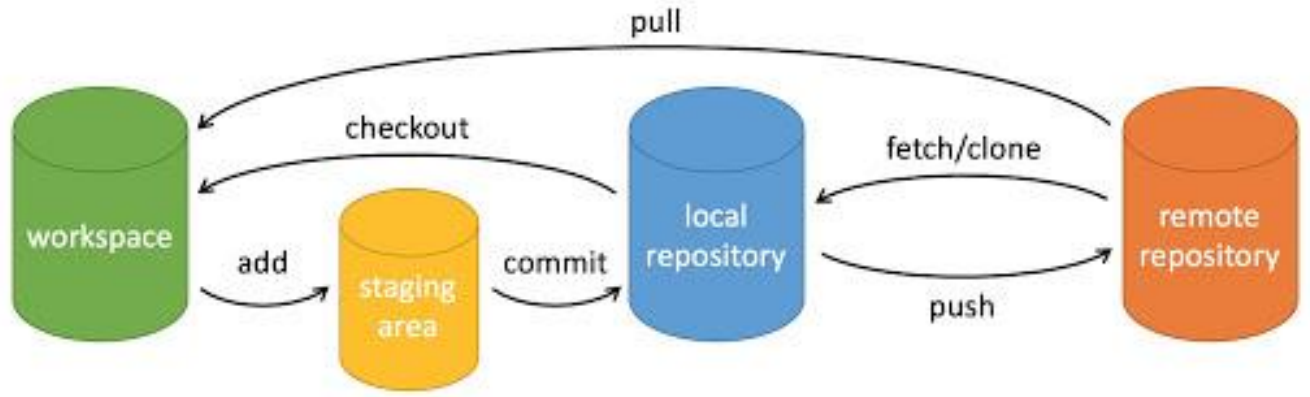
4. Go back to a specific commit.
   - git reset --hard <hash> remove commits (head moved), data changed
   - git reset <hash> remove commits (head moved), data NOT changed

Exercise 2 (15 mins)

1. Create two python files in project folder econ7035, commit them.
   - git add project_a.py project_a.py or git add .
   - git commit –m “initial commit”
2. Edit project_a.py by creating a variable *x*, commit the change.
   - git commit -m “variable x” project_a.py
3. Edit project_b.py by creating a variable *y*, commit the change.
   - git commit -m “variable y” project_b.py
4. Undo the last commit in project_b.py.
   - git revert <hash> --no-edit
5. Go back to ‘initial commit’, remove both data and commits.
   - git reset --hard <hash>

Work with Remote Repositories

- Remote repositories are versions of your project that are hosted on the internet.
  - **Add** or **remove** remote repos in local working directory.
  - **Push** (write to) or **pull** (read from) data between a local repo and a remote repo.



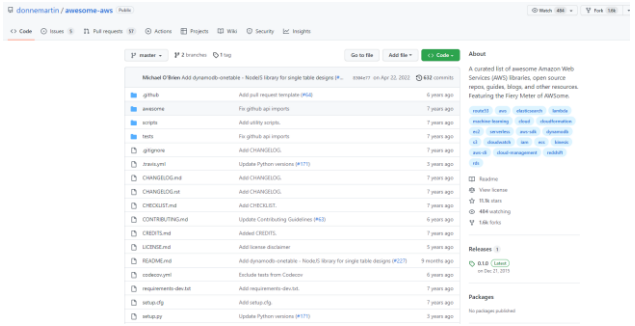
Demo: Git Clone

1. You may need to navigate to another project folder first.
   - cd.. move outside the current folder

2. Download a remote repo awesome-aws from GitHub.
   - git clone <repo url> clone the default branch master
   - git clone https://github.com/donnemartin/awesome-aws

   To clone a specific remote branch: **git clone –b <branch> <remote url>**

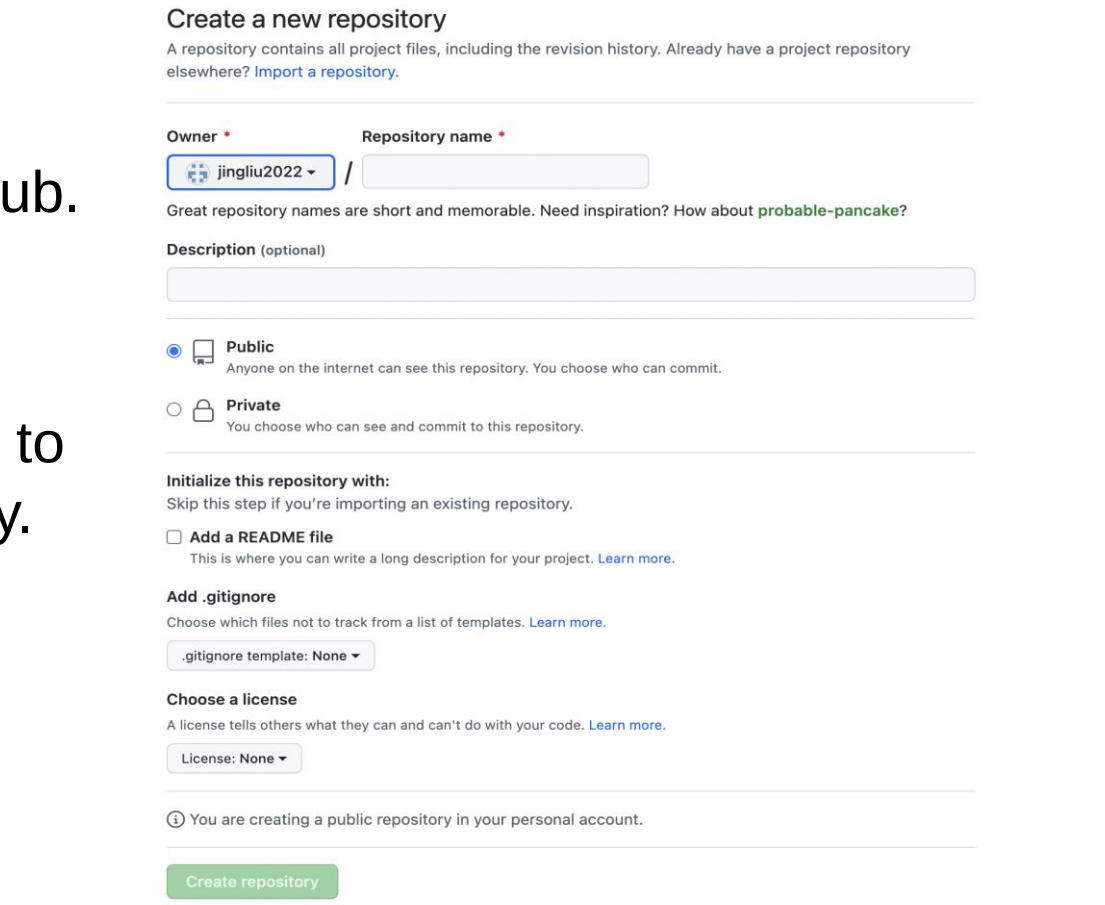
3. Check the current state of the local repo
   - git status

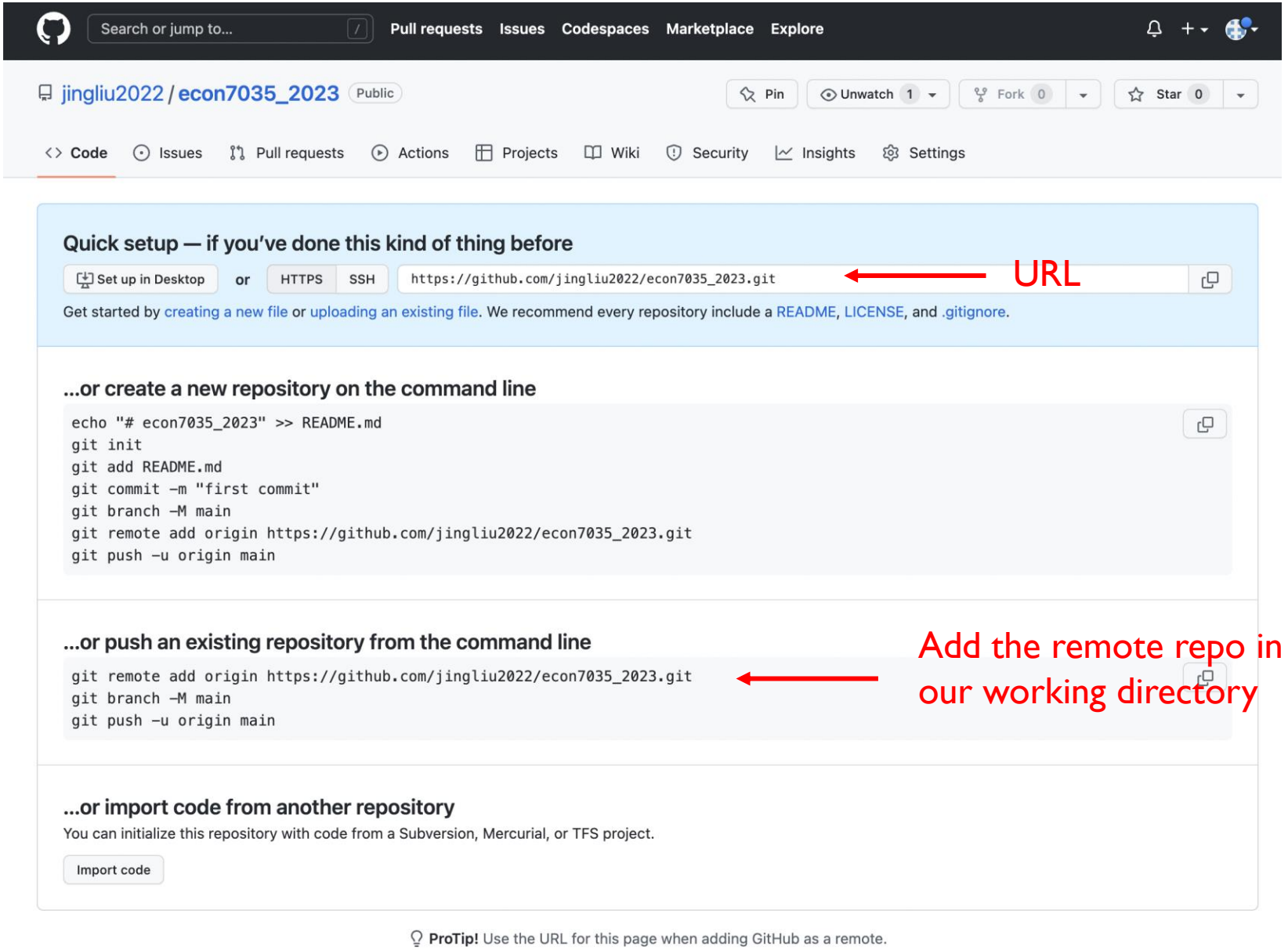


4. Check previous commit history
   - git log

Create a Remote Repo on GitHub

- Sign up and sign in GitHub.
  - Use your HKBU email.
- Move to your dashboard to create a public repository.




Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
jingliu2022 / econ7035_2023
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/jingliu2022/econ7035_2023.git
URL
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.
...or create a new repository on the command line
echo "# econ7035_2023" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/jingliu2022/econ7035_2023.git
git push -u origin main
...or push an existing repository from the command line
git remote add origin https://github.com/jingliu2022/econ7035_2023.git
git branch -M main
git push -u origin main
Add the remote repo in our working directory
...or import code from another repository
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.
Import code
ProTip! Use the URL for this page when adding GitHub as a remote.

Demo: Work with a Remote Repo

1. Add the remote repo named origin, defined by its URL.
   - git remote add <remote alias> <URL>

OK to use a different name

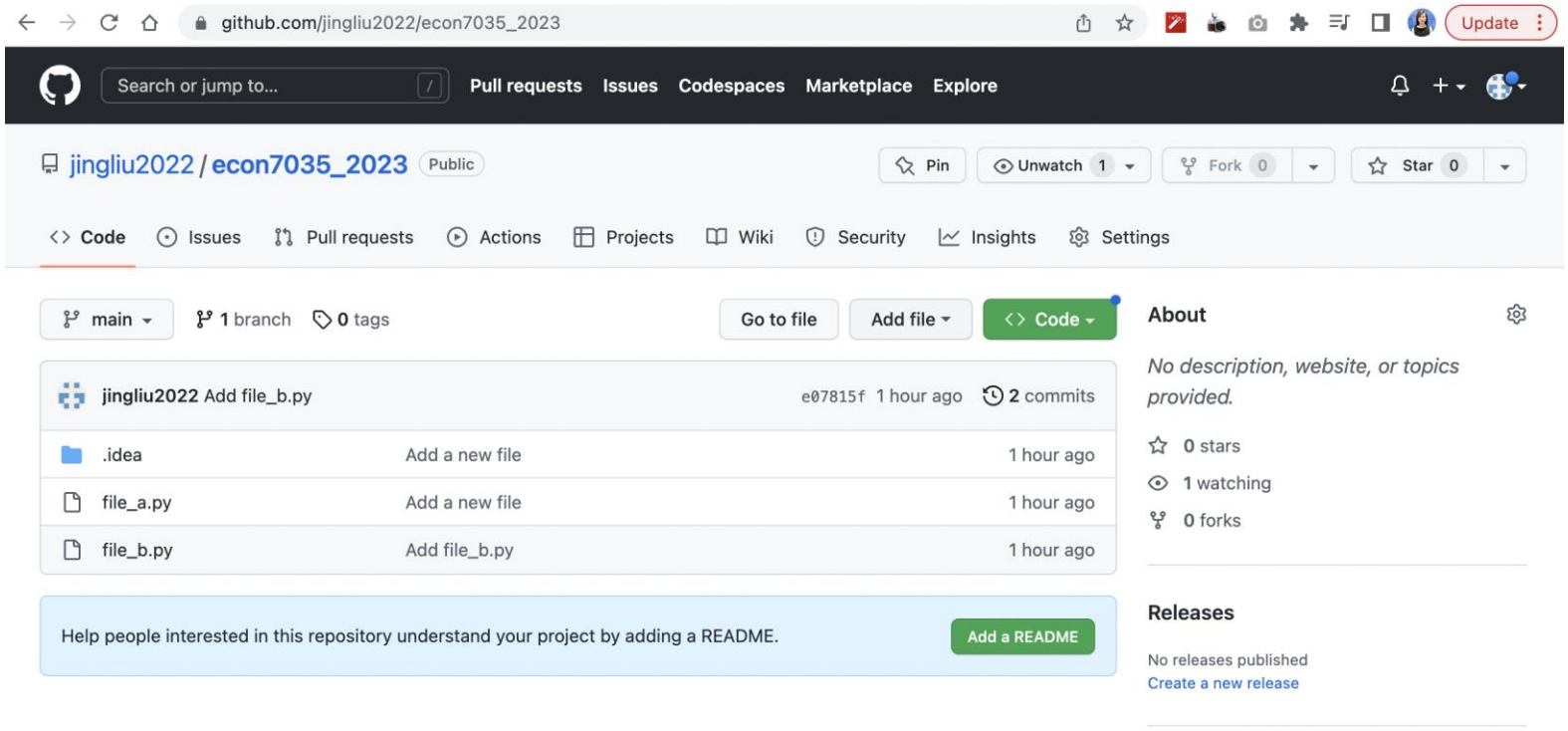
2. Check which remote repo you have connected with
   - git remote
   - git remote -v

3. Push all data from current branch main (in local repo) to the remote repo named origin.
   - git push <remote alias> <local branch>
   - git push origin main
     - you will be prompted to login with username & personal access token.

To push to a specific remote branch:
git push <remote alias> <local branch>:<remote branch]

Check Your Remote Repo on GitHub

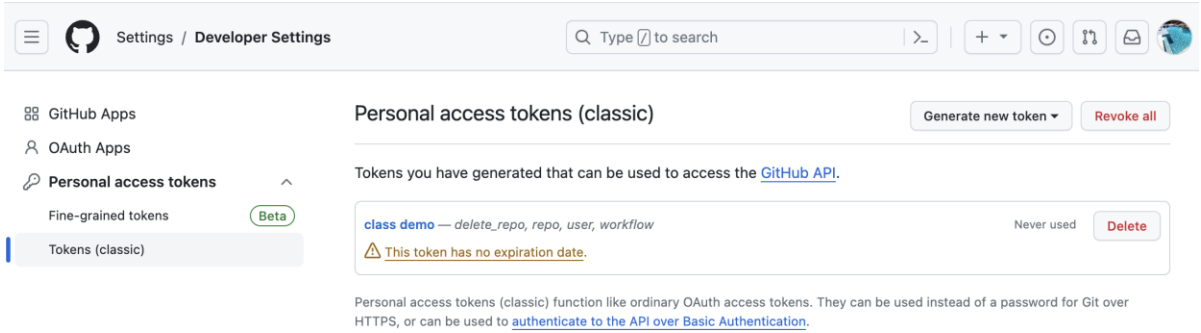
- After pushing a local repo to the remote server, check it out.
  - refresh the URL of your remote repo named origin, you see all files from local repo (main branch) have been pushed there.



Demo: Create a Personal Access Token

- Settings –> Developer settings –> Personal access tokens –> Tokens (classic) –> Generate new token (classic)
  - Create a note (i.e., token name)
  - Set expiration date (default: 30 days)
  - Select scopes (select *repo*, *workflow*, *user*, *delete_repo* for now)

- Make sure you save the token, because you won't see it again.



- Check detailed guideline: Managing your personal access tokens

Exercise 3 (15 Mins)

1. Sign up / in GitHub and create a public remote repository named econ7035.

2. Add this remote repo econ7035 to git, name it as origin.
   - git remote add origin <URL>

3. Push all the files you committed in local repository (main branch) to the remote repo origin on GitHub.
   - git push origin main
   - You may need to create a personal access token first.

4. Check out your remote repo econ7035 in a web browser.
   - Check all the files and change history.

Git Workflow Review

- You start a project worker, add some data and python files.
  - git add so Git will track these files and changes made to them.
  - git commit so Git will save all changes in the local repo.
- You work on a file, when ready to be saved:
  - git commit so that Git will save all changes in local repo.
  - You may need to git add so that Git know what will be committed.
- When all done
  - git push so local repo are pushed to a remote server.

More about Git Remote

- Inspect a remote
  - git remote show <remote alias>
  - git remote show origin
- Rename a remote
  - git remote rename <current alias> <new alias>
  - git remote rename origin origin2
- Remove a remote
  - git remote remove <remote alias>
  - git remote remove origin2