

Git & GitHub Training Module

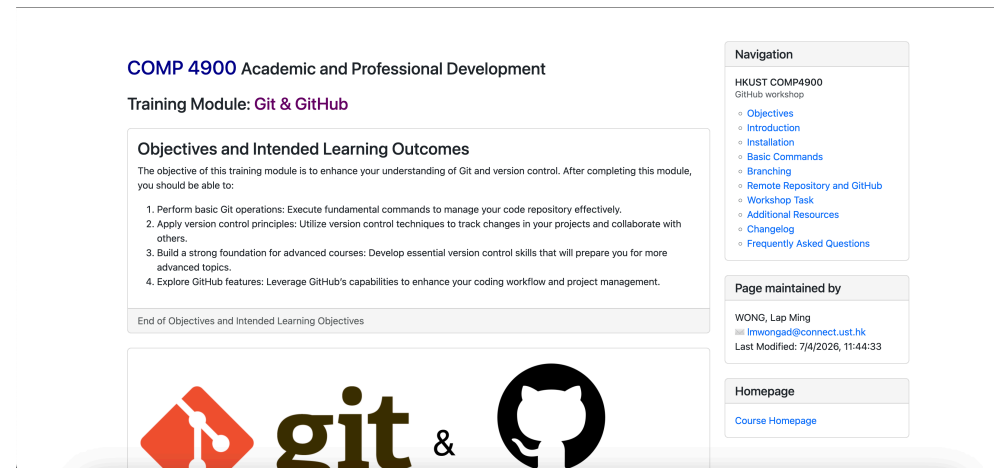
COMP 4900 Academic and Professional Development

WONG, Lap Ming

Hong Kong University of Science and Technology

15th April, 2026

- You can find the workshop module website here:
 - https://minglol7794.github.io/github_tutorial/COMP4900-Website/
- It include the topic we will cover today, installation step for git and github, extra self-learning material, and task you need to do in this training module



The objective of this training module is to enhance your understanding of Git and version control. After completing this module, you should be able to:

1. Perform basic Git operations: Execute fundamental commands to manage your code repository effectively.
2. Apply version control principles: Utilize version control techniques to track changes in your projects and collaborate with others.
3. Build a strong foundation for advanced courses: Develop essential version control skills that will prepare you for more advanced topics.
4. Explore GitHub features: Leverage GitHub's capabilities to enhance your coding workflow and project management.

Contents

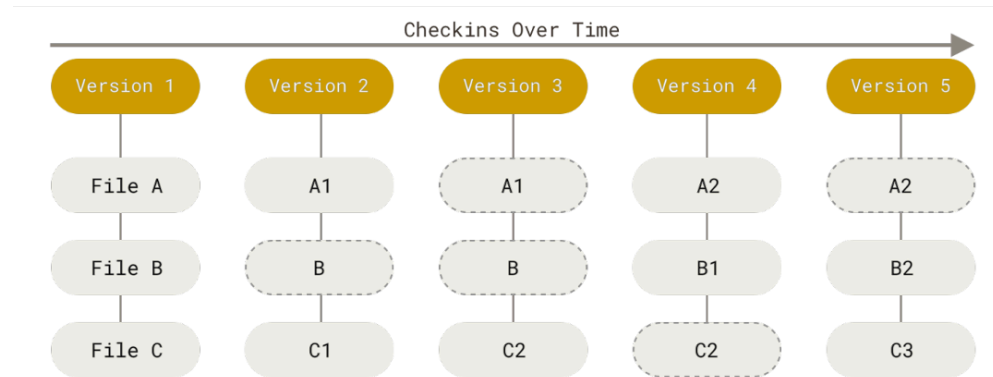
1. Part I: Version Control Fundamentals
2. Part II: Installation and Setup
3. Part III: Git Fundamentals
4. Part IV: Branching and Merging
5. Part V: Remote Repositories and GitHub
6. Part VI: Workshop Task
7. Part VII: Additional Resources (Self-study)

Part I: Version Control Fundamentals

Version control is a system that records changes to files over time so you can return to a specific version later.

A **version control** system helps you:

- Revert a selected file back to a previous state.
- Revert an entire project back to a previous state.
- Compare changes over time.
- See who last modified the files.



- **Git** is a distributed version control system that tracks changes in your code over time.
- **Git** stores work as a stream of snapshots.
- A **Git repository** contains a hidden `.git` folder inside the project directory.
- **Git** is the tool that manages local history, branches, and commits.



- **GitHub** is a website and cloud service for hosting code and projects.
- It is commonly used to save **repositories** online and collaborate with other people.
- **GitHub** supports sharing, reviewing, and publishing code.
- It can also serve as part of a developer portfolio.



Part II: Installation and Setup

This workshop uses **Git**, **GitHub**, and **VS Code** together. Follow these setup steps before starting the exercises.

1. Create a **GitHub** account

- Go to <https://github.com/>
- Sign up for a free account

2. Install **Git**

- **Windows**: Download **Git** from <https://git-scm.com>, open the .exe installer, click Next until the editor page, then choose Visual Studio Code as the default editor
- **macOS**: Install **Homebrew** first if needed, then run:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/  
install/HEAD/install.sh)"  
brew install git
```


3. Configure **Git** identity

- Open the **VS Code** terminal and run:

```
git config --global user.name "Your Name"  
git config --global user.email "your.email@example.com"
```

4. Set up **SSH** access

- Generate an **SSH** key with:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

- Press Enter to accept the default save location
- Copy the public key to **GitHub** later for secure authentication

For step-by-step installation guidance, refer back to the workshop website:

https://minglol7794.github.io/github_tutorial/COMP4900-Website/#installation

Part III: Git Fundamentals

To start tracking your project files, you need to initialize a Git repository. This creates a hidden `.git` folder that stores all version history and metadata.

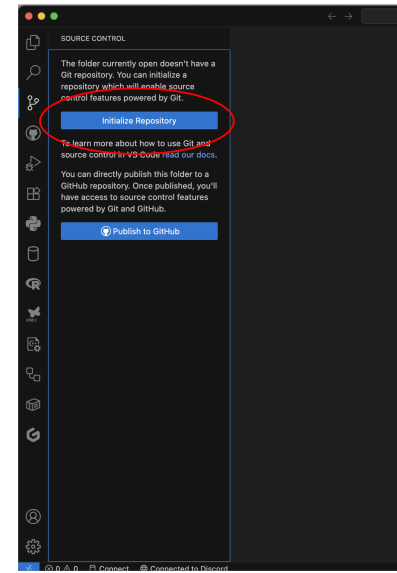
VS Code method:

Initialize a repository: Sets up Git to track your project

- In VS Code: Click Initialize Repository
- In terminal: Run `git init`

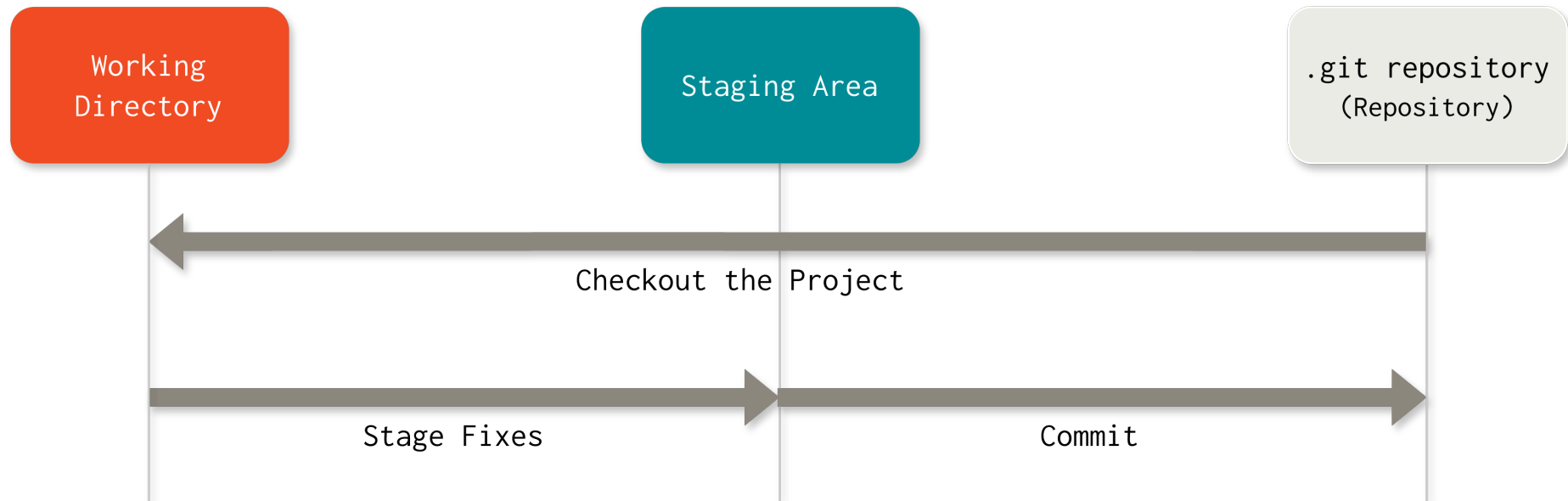
```
git init
```

This creates the hidden `.git` folder in your project root.



Three states of a local Git Workflow:

- Modified: files were changed in the working directory.
- Staged: files were marked for the next commit.
- Committed: changes were saved to repository history.



Before saving changes to history, you select which files to include. Staging marks files as “ready to be saved” and lets you group related changes together.

Stage files: Marks them ready for the next commit

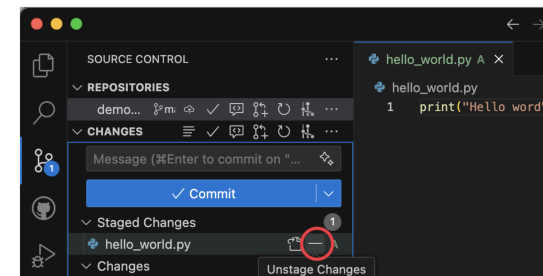
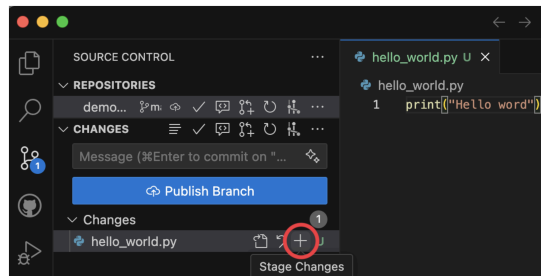
- Click the + icon in VS Code
- Or in terminal: `git add filename`

Unstage files: Removes them from staging if you changed your mind

- Or in terminal: `git reset filename`

`git add hello_world.py`

`git reset hello_world.py`



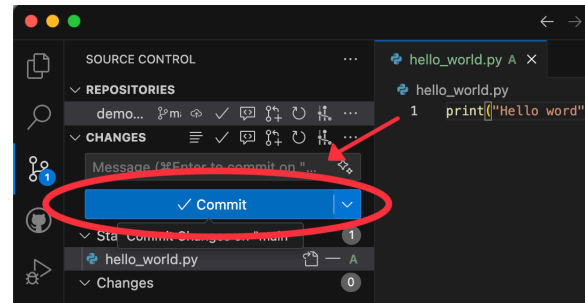
Committing saves a permanent snapshot of your staged changes. Each commit is a saved point in history that you can return to later.

What committing does:

- Saves all staged changes with a timestamp
- Creates a permanent record in your repository history
- Allows you to later revert to this exact state
- Makes collaboration easier with clear change descriptions

VS Code method:

```
git status  
git commit -m "commit message"
```

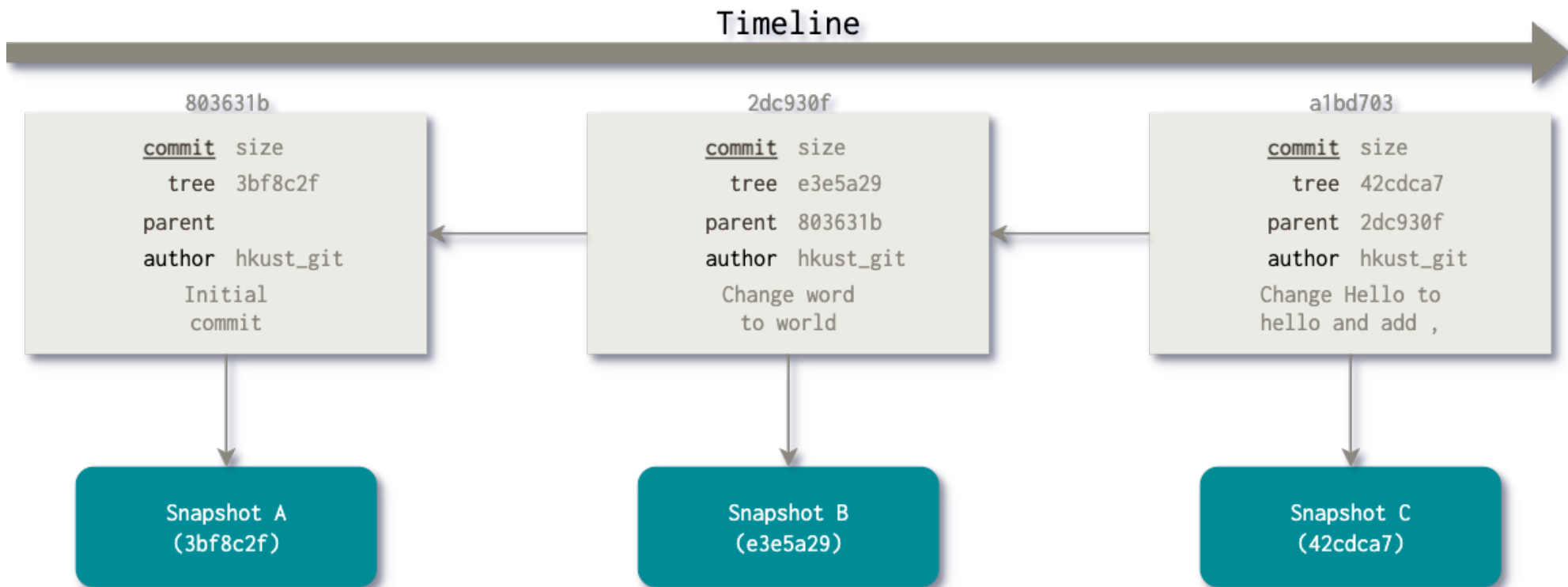


A commit object is Git's saved record of a snapshot. It stores the content tree, the parent commit, the author, the timestamp, and the commit message.

What a commit object does:

- Points to the exact state of the project at that moment
- Links to the previous commit so Git can track history
- Uses the commit pointer to show which snapshot is currently checked out
- Keeps the commit message that explains why the change was made
- Gives each commit a unique hash ID

Commit Object visualization:



How to view commit messages:

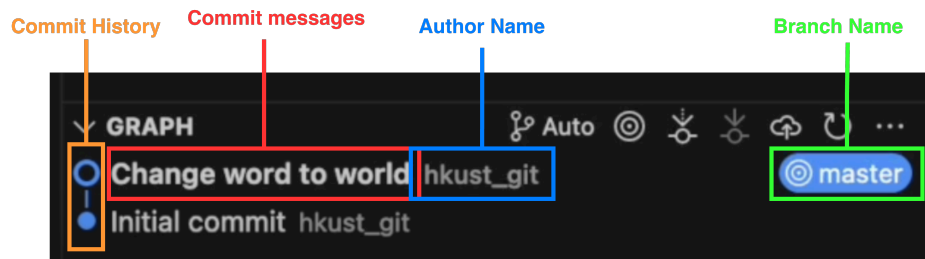
- `git log` shows the full commit history with messages
- `git log --oneline` shows a short one-line summary per commit
- `git show <commit-hash>` shows one commit and its message

In terminal:

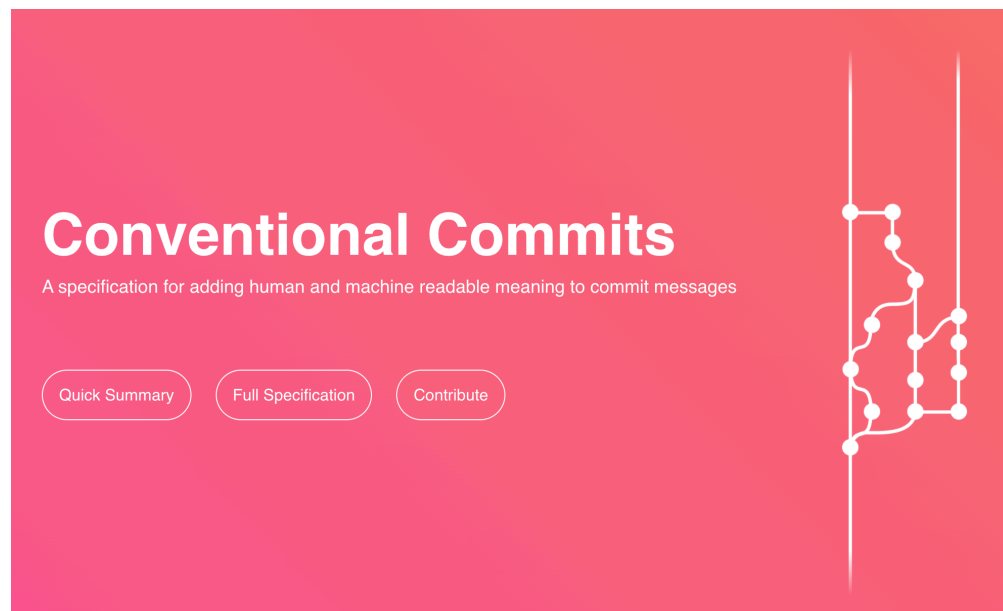
```
git log --oneline
```

```
git show <commit-hash>
```

Commit history in VS Code:

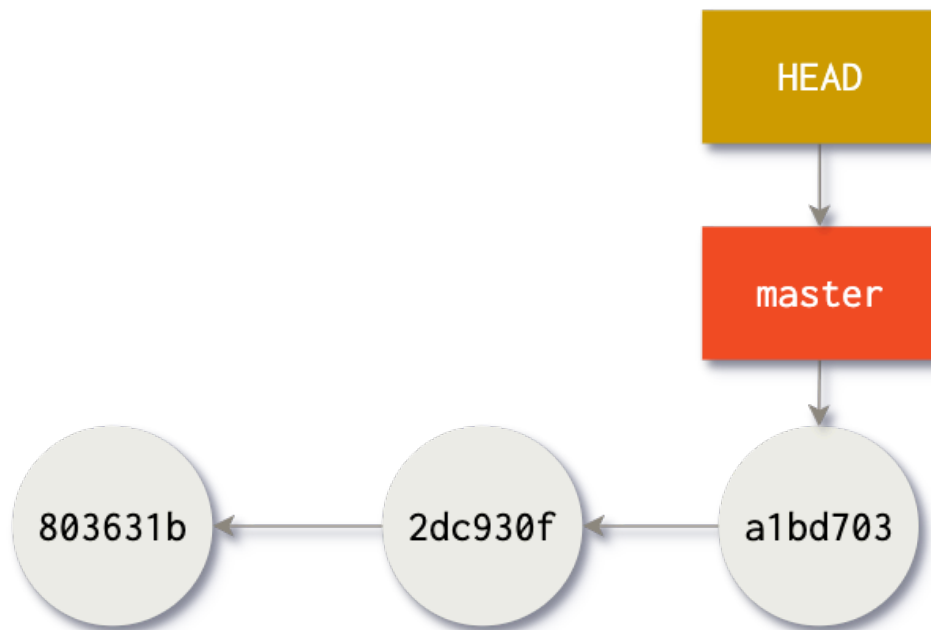


When working with other, you always want to write meaningful commit message. One of the standard format of writing commit message: **Conventional Commits**. You can read more details through this link: <https://www.conventionalcommits.org/en/v1.0.0/>



Part IV: Branching and Merging

- A branch is a pointer to a commit.
- HEAD shows the branch you are currently on.
- Branches let you work on features without changing the main line directly.
- This makes development safer and easier to organize.



You need to create a new branch to work on a feature separately from the main code. This allows you to make changes without affecting the stable version.

Create a branch: Creates a new isolated workspace for your feature

- `git branch feature` — creates the branch

Switch to the branch: Moves you to work on that branch

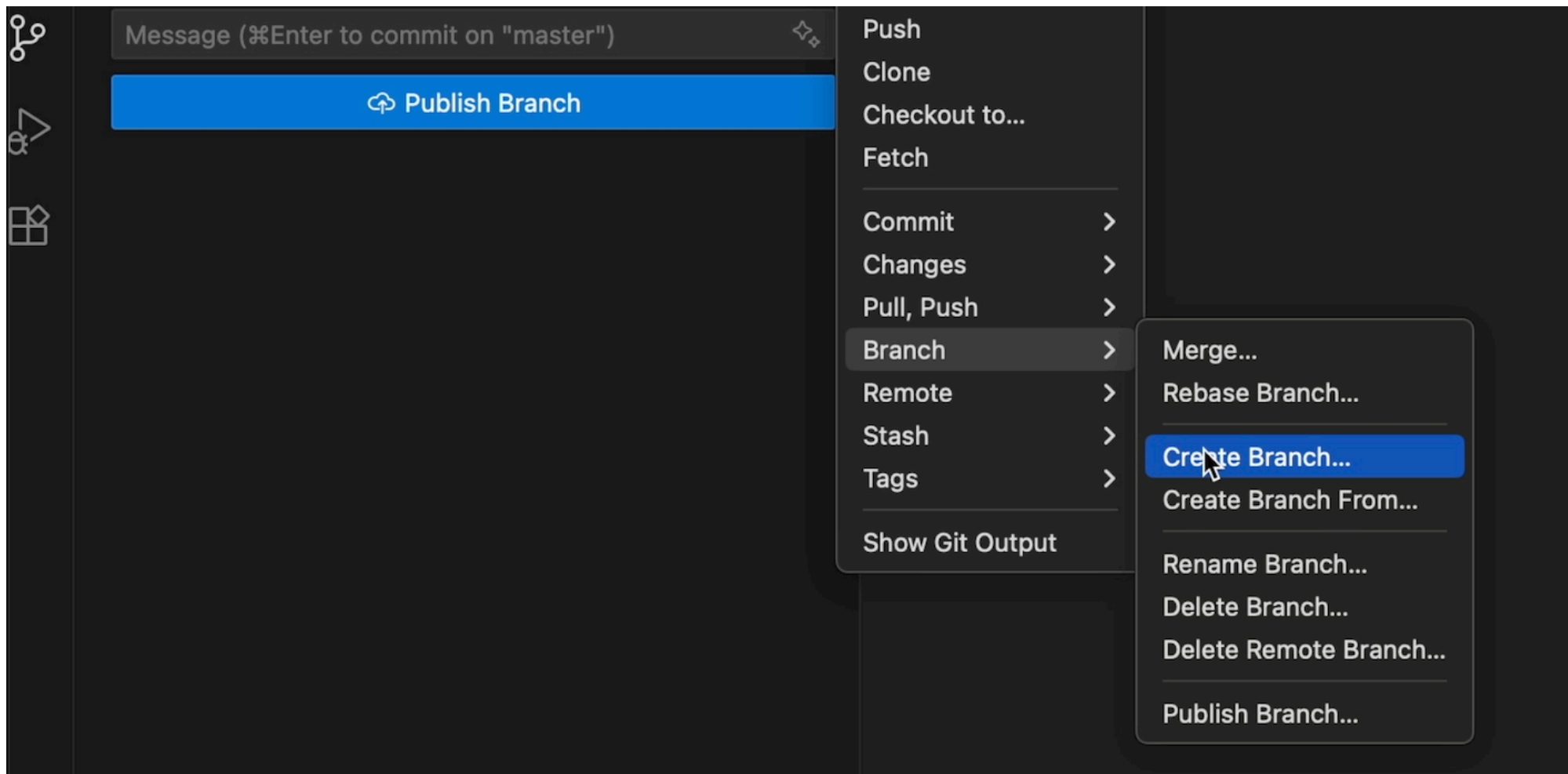
- `git checkout feature` — switches to it
- `git checkout -b feature` — creates and switches in one step

```
git branch feature
```

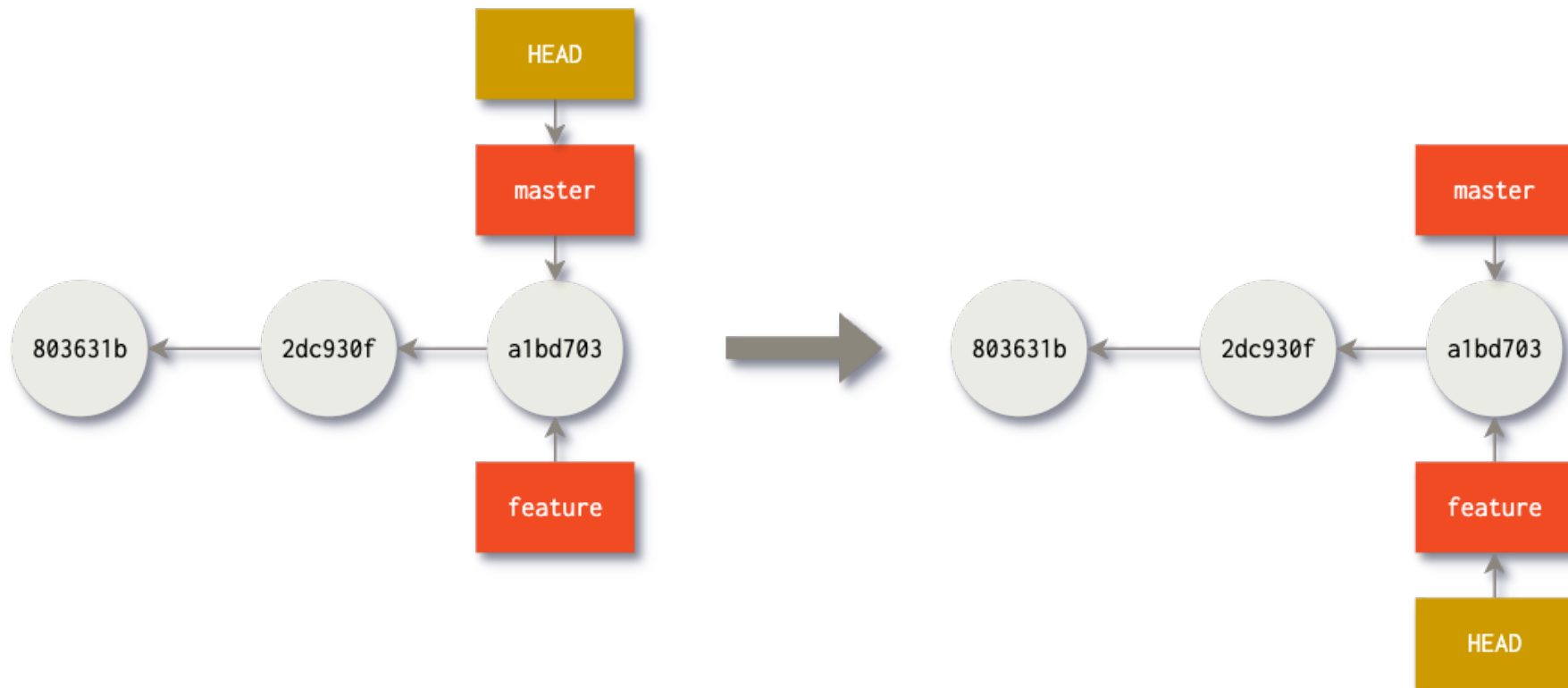
```
git checkout feature
```

```
git checkout -b feature
```

Or use VS Code from the Branch menu:



Result: Your pointer moves to the new branch



Once your feature is complete and tested on its own branch, you merge it back into the main branch. This integrates your changes into the stable codebase.

Steps to merge:

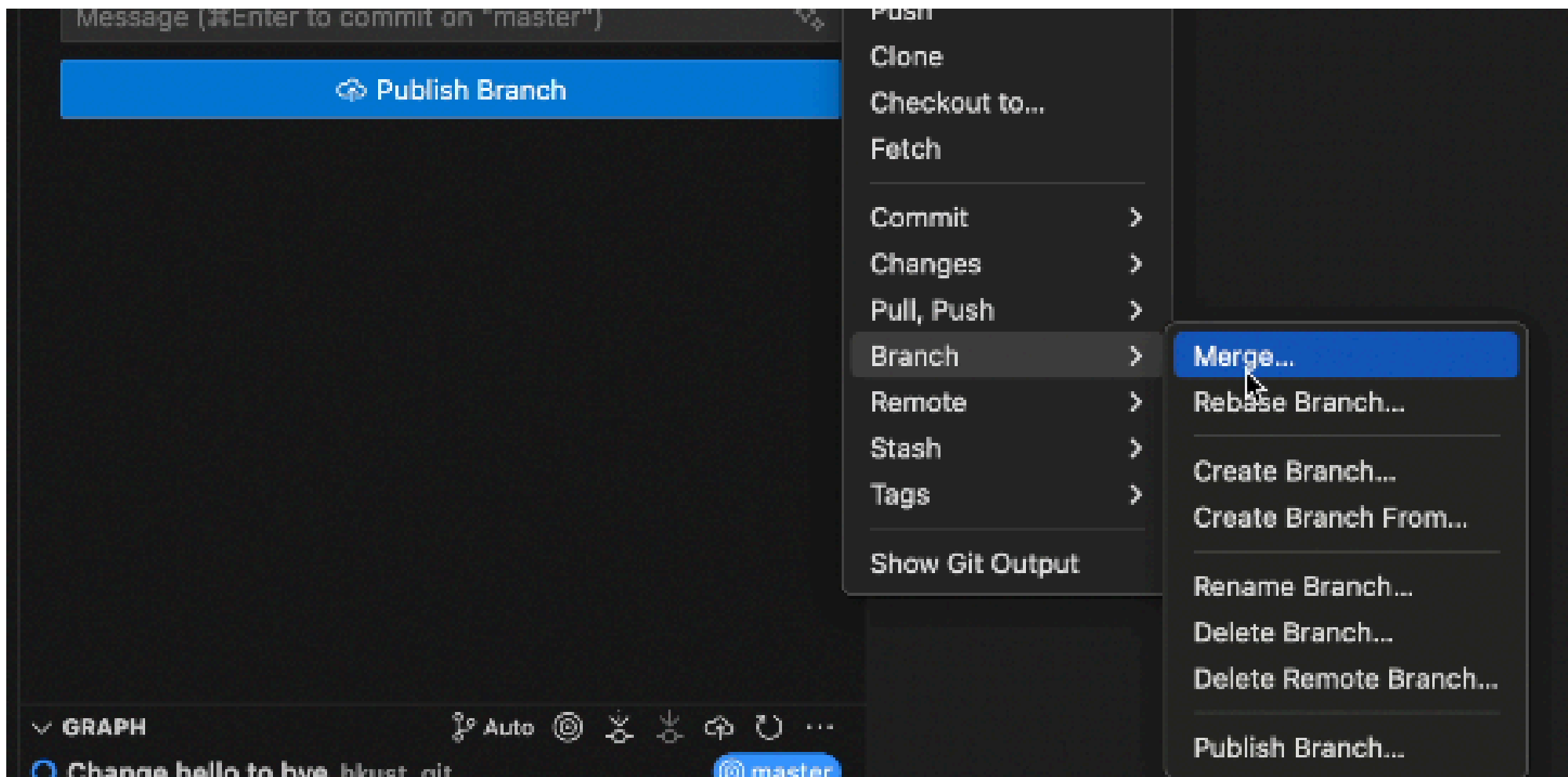
- Switch to the target branch (the one receiving the changes)
- Use Git Merge to integrate the feature branch
- Git creates a merge commit combining both branches

In terminal:

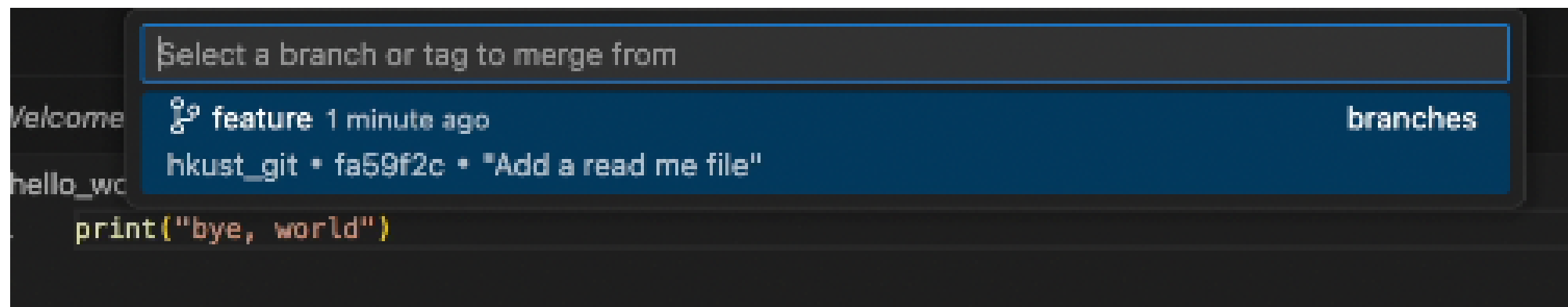
```
git checkout master  
git merge feature
```

Or in VS Code:

- Go to Branch menu → Choose Merge

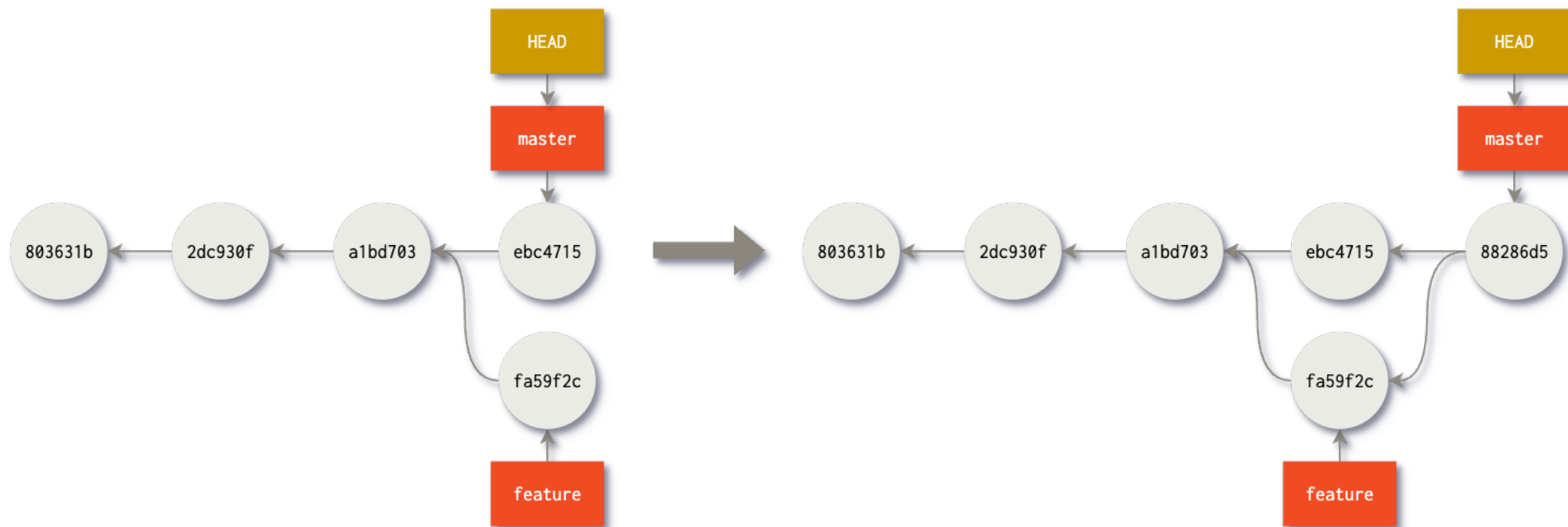


- Select the source branch to merge from



Merge branch (Picture illustration)

29 / 74



Part V: Remote Repositories and GitHub

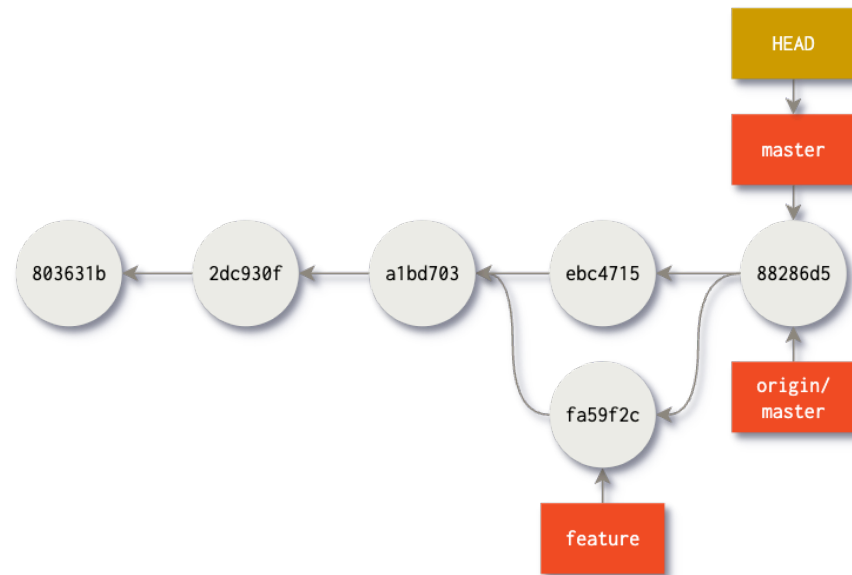
A remote repository is the online backup of your project (hosted on GitHub, GitLab, etc.). It enables collaboration and protects your code.

What a remote does:

- Stores your code safely online
- Lets multiple people collaborate on the same project
- Enables backup of your local changes
- Uses the format: remote-name/branch-name
- origin is the default remote name

You can have multiple remotes pointing to different services.

A remote branch origin/master pointing at the same branch on master



Creating a GitHub repository creates the online home for your project. You'll link this to your local repository afterward.

Steps to create:

- Go to GitHub and create a new repository
- **Important:** If your project already exists locally, do NOT initialize with README, .gitignore, or license (avoid conflicts)
- Copy the repository URL for linking to your local project
- Keep assignment repositories private for security

Create repo page:

Create a new repository

[Try the new experience](#)

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner *

Repository name *

hkustGIT

/ demo

Checking availability...

Great repository names are short and memorable. Need inspiration? How about [effective-octo-rotary-phone](#) ?

Description (optional)

☒ Public

Anyone on the internet can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

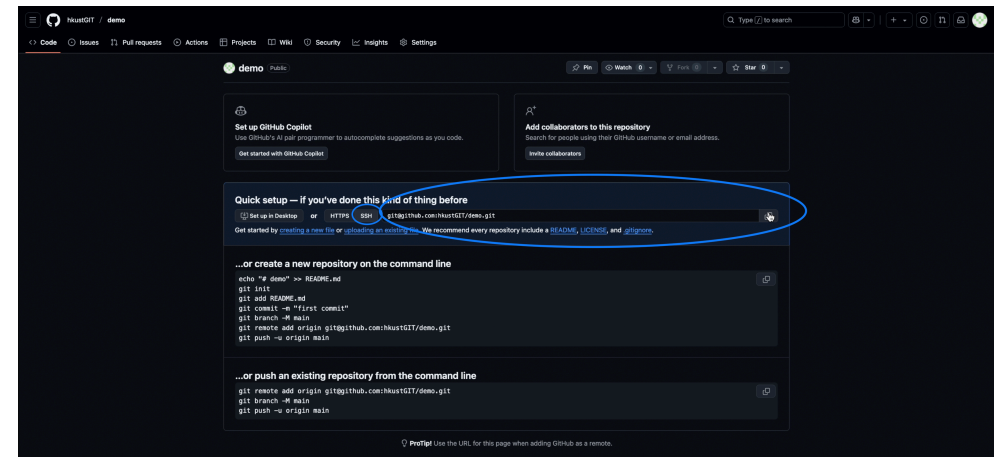
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

☐ You are creating a public repository in your personal account.

Create repository

Copy the URL:



Linking your local repository to GitHub tells Git where to upload your code. This connection is called a “remote.”

What linking does:

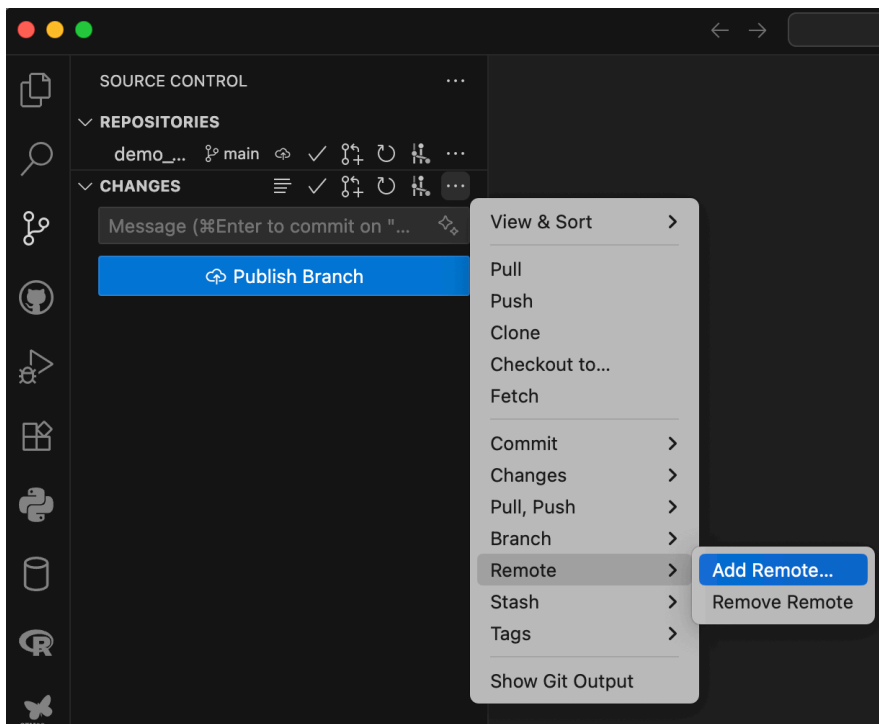
- Tells Git where your online repository lives
- Enables pushing your local commits to GitHub
- Enables pulling other people's changes from GitHub
- Creates the `origin` remote as your default upload destination

On terminal:

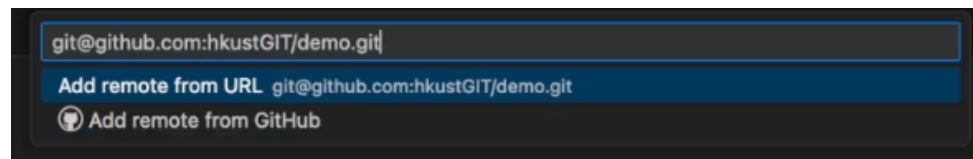
```
git remote add origin <url>  
git remote -v
```


VS Code method:

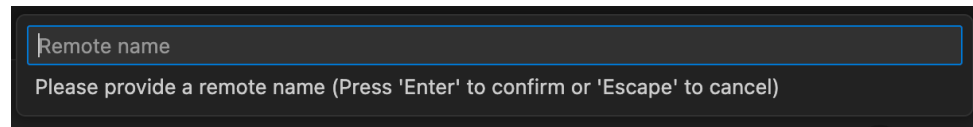
1. Use Remote → Add Remote



2. Paste the GitHub URL



3. Use origin as the remote name



If you need to disconnect from a remote (e.g., the URL changed, project moved), you can remove that connection.

When to remove a remote:

- The repository URL changed
- You no longer need to push to this location
- You moved the online repository to a different service

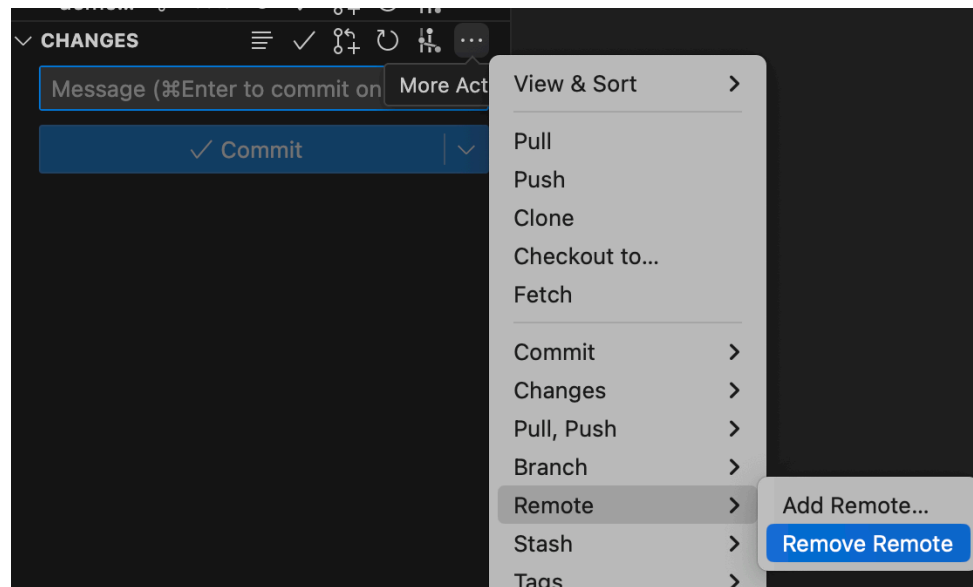
In terminal:

```
git remote remove <remote-name>
```

VS Code method:

- Use Remote → Remove Remote
- Select the remote to disconnect

VS Code interface:



Pushing uploads your local commits to GitHub. This backs up your work and makes it available to collaborators.

What pushing does:

- Uploads your local commits to the remote repository
- Backs up your code online
- Shares your changes with teammates
- The first push sets up “upstream tracking” (tells Git where to push by default)
- Later pushes only need `git push` without extra parameters

In terminal First push (sets upstream):

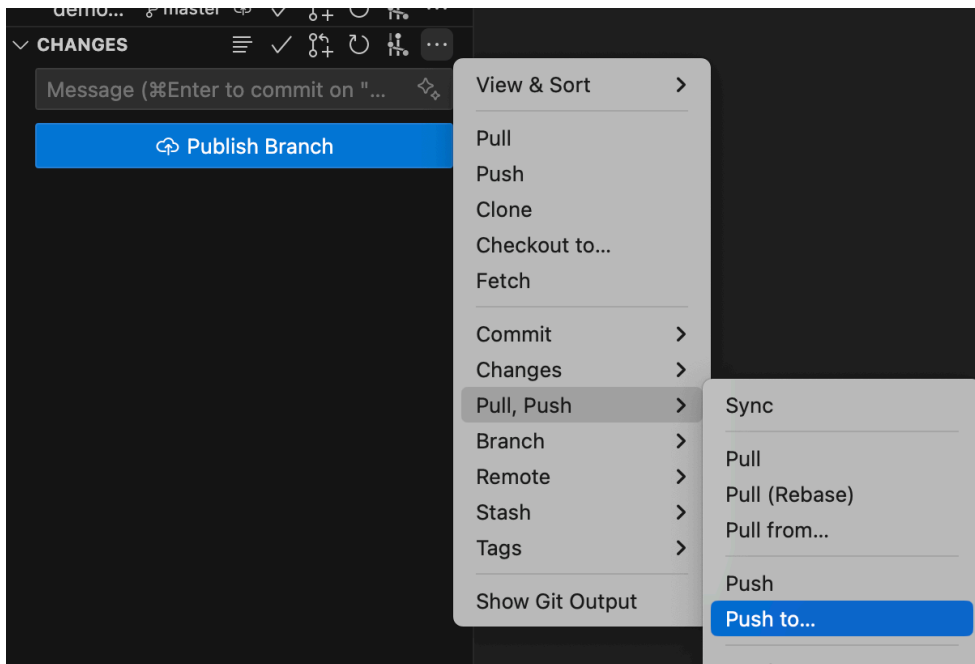
```
git push --set-upstream <remote> <branch>
```

Later pushes:

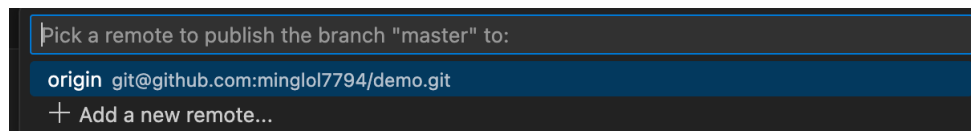
```
git push
```

In VS Code:

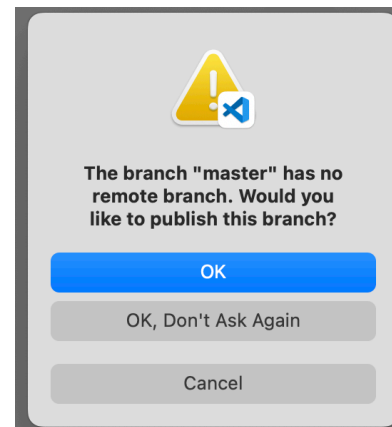
1. Use Pull Push → Push to...



2. Select the remote repository



3. Confirm the push



Keeping your local copy in sync with GitHub prevents conflicts and ensures you have the latest changes. Fetch gets the updates, then merge integrates them.

What syncing does:

- **Fetch:** Downloads remote changes without modifying your files
- **Merge:** Integrates those remote changes into your local branch
- Together they keep your local work aligned with the team's shared work
- Sync frequently to avoid your branch drifting too far

In terminal:

```
git fetch <remote>
```

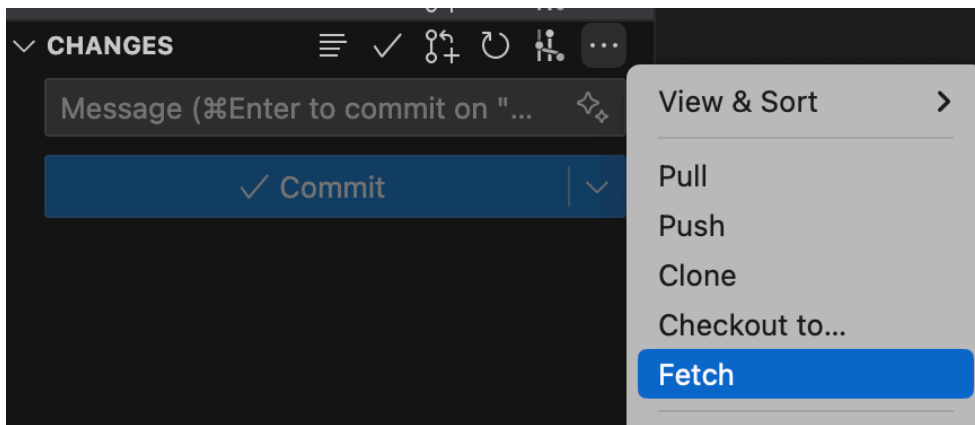
```
git merge <remote>/<branch> <branch>
```

Or doing both together using Pull

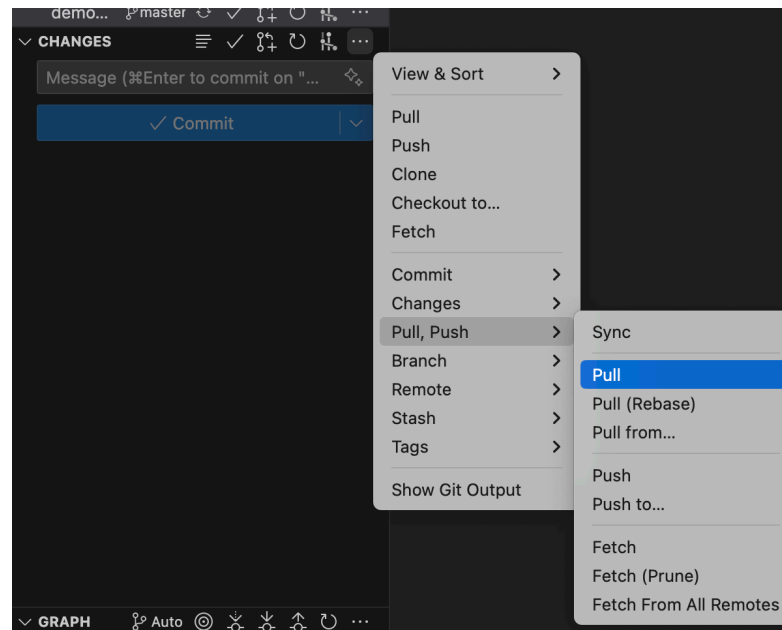
```
git pull <remote> <branch>
```

Using VS Code Fetch/Pull command.

- Use Fetch



- Use Pull

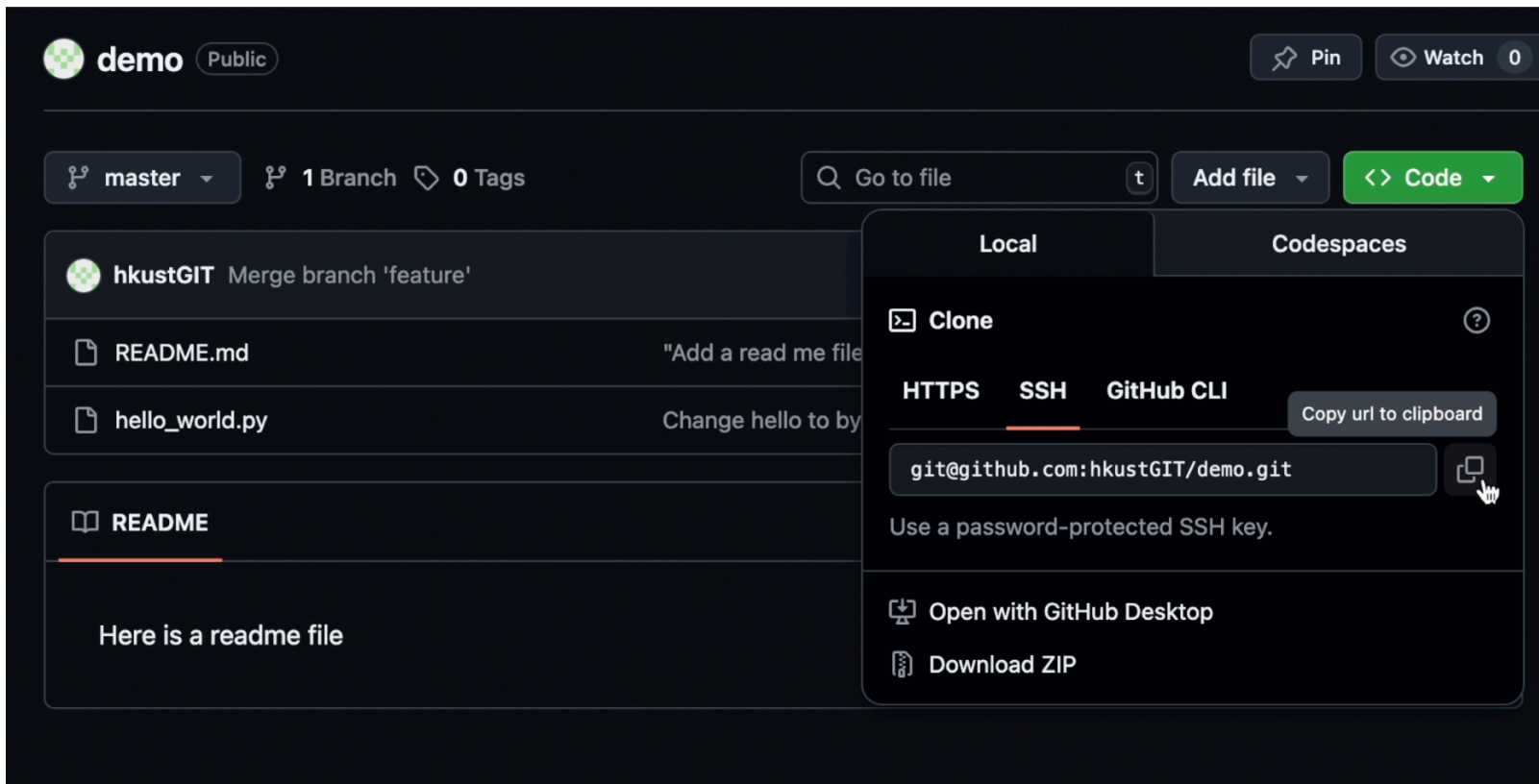


Cloning creates a complete local copy of an existing remote repository. Use this to download a project and its full history.

What cloning does:

- Downloads the entire remote repository to your computer
- Includes all branches and commit history
- Automatically sets up `origin` as the default remote
- Creates a folder for the project
- Ready to work immediately without any setup

Get the URL from GitHub:

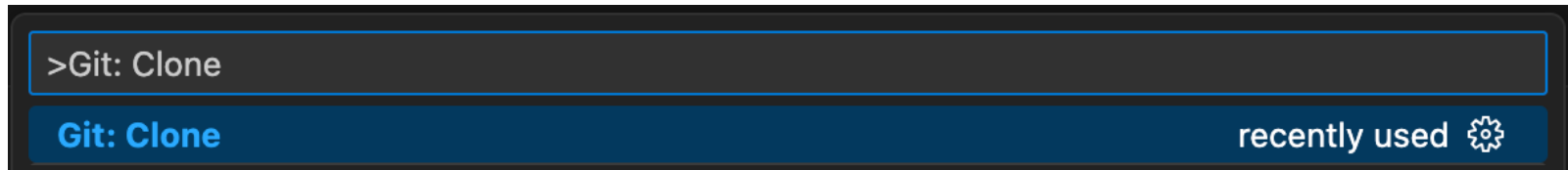


Clone in terminal:

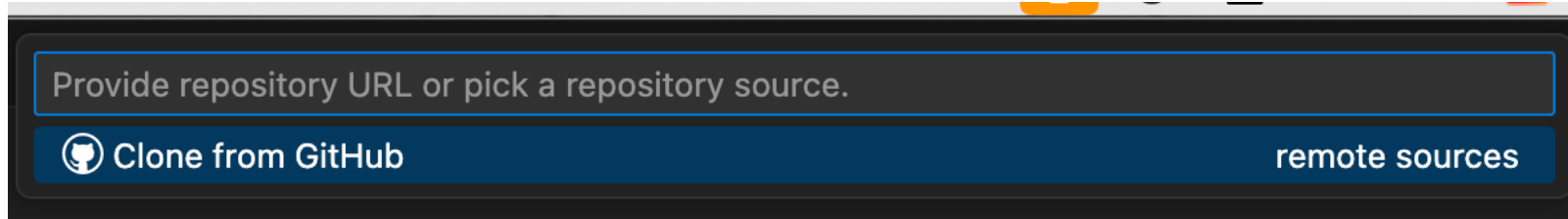
```
git clone <remote-url>
```

Clone in VS Code:

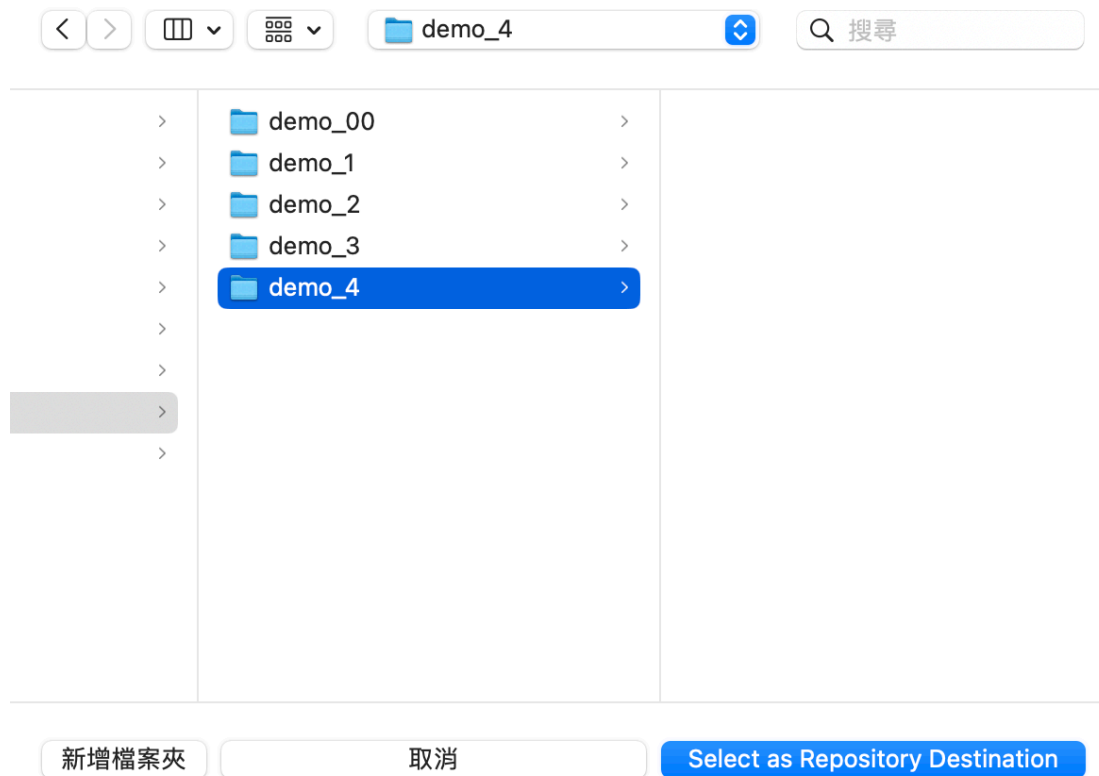
1. Ctrl/Command + Shift + P and type Git: Clone



2. Paste URL:



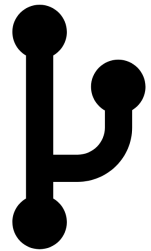
3. Choose local folder:



Forking creates your own independent copy of someone else's repository on GitHub. Use fork when you don't have write access to the original.

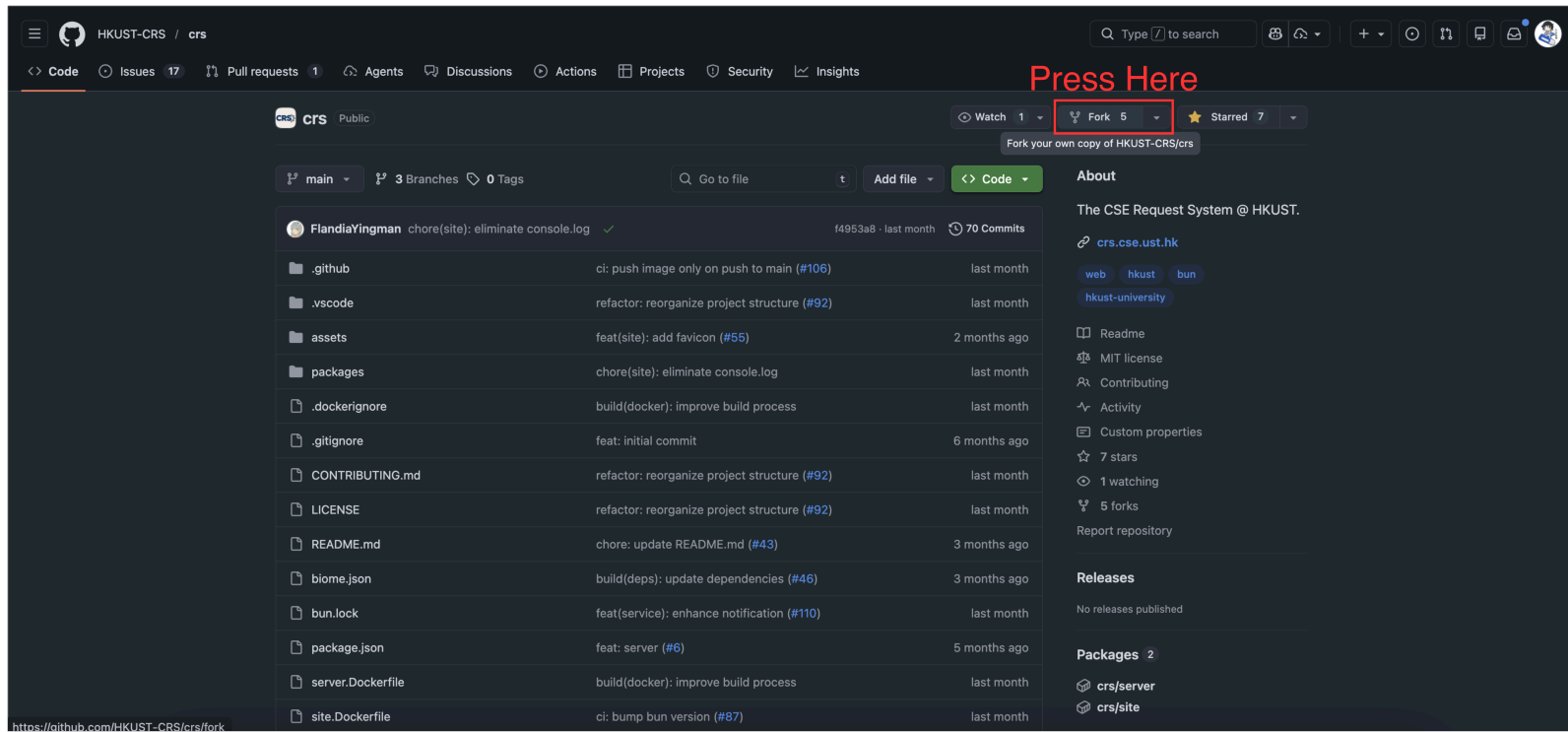
Key differences:

- **Clone:** Use when you have write access to the repository
- **Fork:** Use when you don't have write access; creates your own copy
- After forking, you can clone your fork locally and work on it
- You can submit changes back via pull request
- Commonly used for open-source contributions



Steps to fork:

1. At the original repository, click Fork button:



2. Confirm fork:

HKUST-CRS / crs

Code Issues 17 Pull requests 1 Agents Discussions Actions Projects Security Insights

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (*)

Owner * Repository name *

Choose an owner / crs

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description

The CSE Request System @ HKUST.

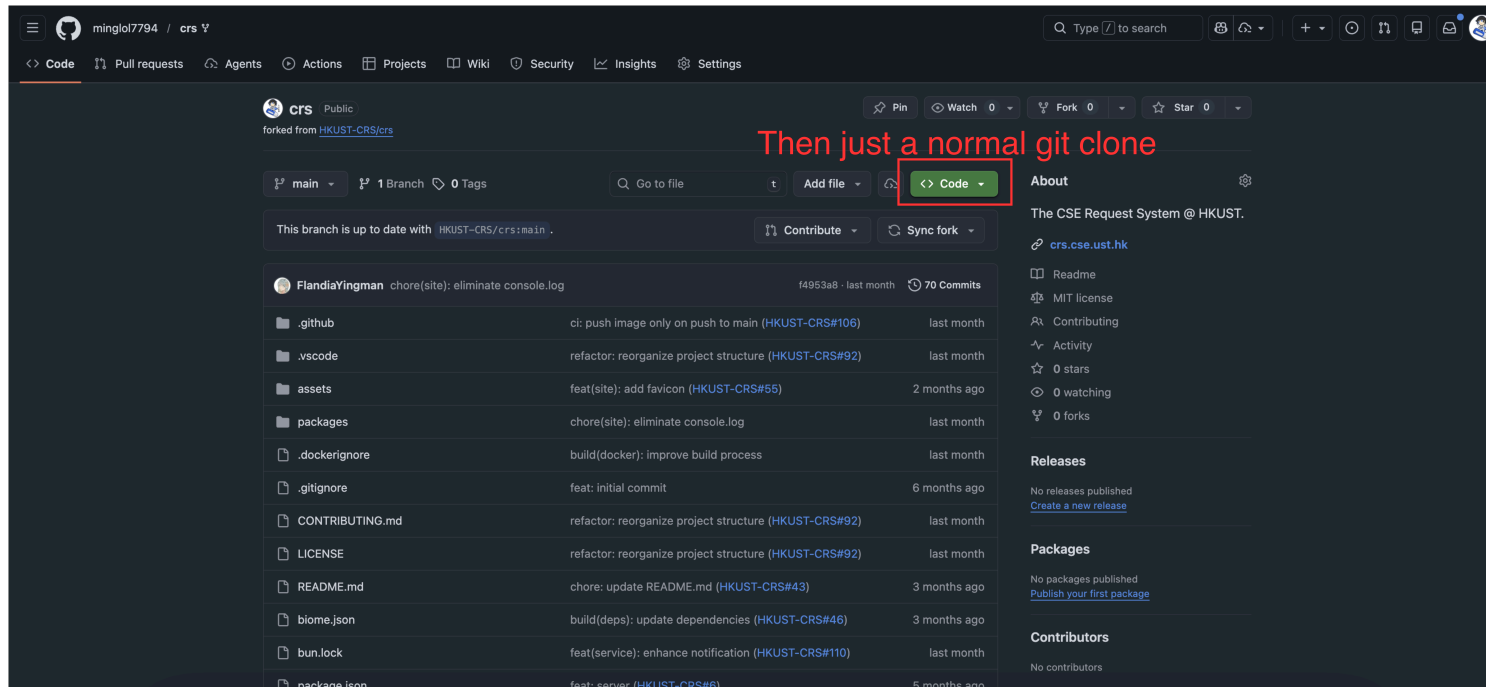
32 / 350 characters

☒ Copy the `main` branch only

Contribute back to HKUST-CRS/crs by adding your own branch. [Learn more.](#)

Create fork

3. Forked to your account:

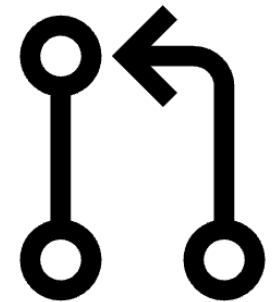


Then you can use `git clone` to clone the repositories you have made

A pull request proposes changes from your fork back to the original repository. It allows maintainers to review and approve your contributions.

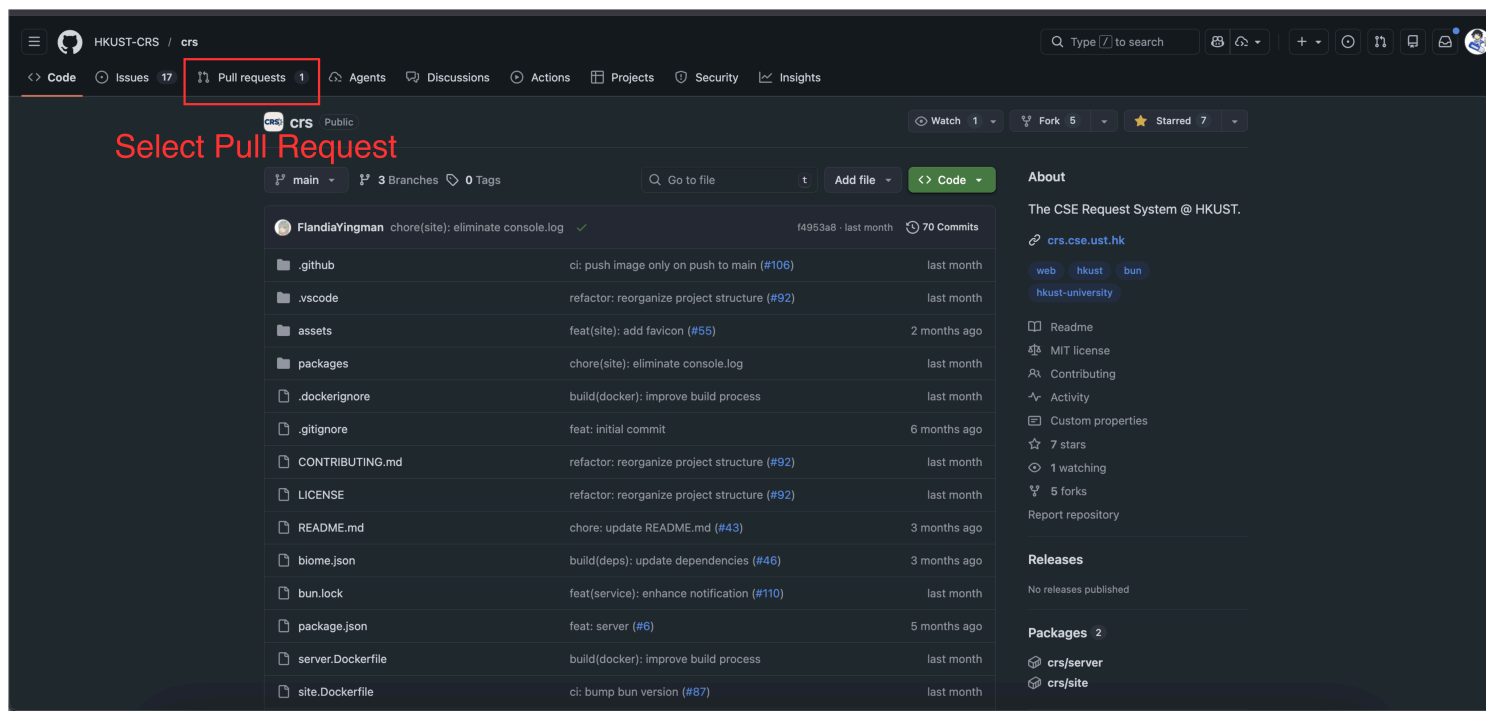
What a pull request does:

- Shows the differences (diffs) between your code and the main branch
- Lets maintainers review your changes before accepting
- Creates a conversation thread for discussion and feedback
- Can be merged once approved or closed if not accepted
- Standard way to contribute to open-source projects

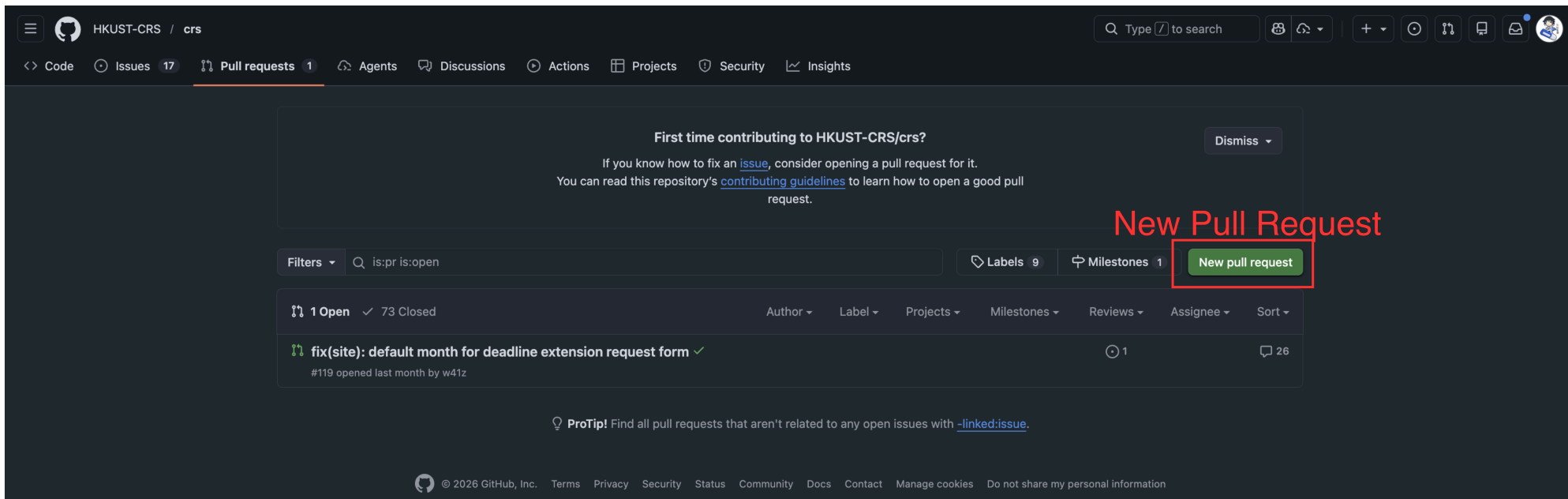


Steps to open a pull request:

1. Open original repo, click Pull requests:



2. Select branches to compare:



The screenshot shows the GitHub interface for the repository **HKUST-CRS / crs**. The navigation bar at the top includes links for Code, Issues (17), Pull requests (1), Agents, Discussions, Actions, Projects, Security, and Insights. A search bar is located on the right. A notification box titled "First time contributing to HKUST-CRS/crs?" is displayed, with a "Dismiss" button. Below the notification, there is a filter bar with a search input "is:pr is:open", a "Labels" section with 9 items, and a "Milestones" section with 1 item. A green button labeled "New pull request" is highlighted with a red box and the text "New Pull Request" in red. Below the filter bar, there is a list of pull requests. The first pull request is titled "fix(site): default month for deadline extension request form" with a green checkmark, indicating it is merged. It was opened last month by user w41z. The interface also shows a "ProTip!" at the bottom: "Find all pull requests that aren't related to any open issues with `-linked:issue`."

HKUST-CRS / crs

Dismiss

First time contributing to HKUST-CRS/crs?

If you know how to fix an [issue](#), consider opening a pull request for it. You can read this repository's [contributing guidelines](#) to learn how to open a good pull request.

Filters Labels 9 Milestones 1

New Pull Request

New pull request

1 Open ✓ 73 Closed

Author Label Projects Milestones Reviews Assignee Sort

fix(site): default month for deadline extension request form ✓

#119 opened last month by w41z

1 26

ProTip! Find all pull requests that aren't related to any open issues with `-linked:issue`.

© 2026 GitHub, Inc. Terms Privacy Security Status Community Docs Contact Manage cookies Do not share my personal information

3. Add title & description:

The screenshot shows the GitHub 'Comparing changes' page for creating a pull request. The interface is dark-themed. At the top, there's a navigation bar with links: Code, Pull requests, Agents, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the 'Comparing changes' section is active. It contains a message: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.' Below this message are four dropdown menus: 'base repository: HKUST-CRS/crs', 'base: main', 'head repository: minglo17794/crs', and 'compare: feat-sort-order'. A red box highlights the 'head repository' and 'compare' dropdowns, with a red text annotation 'Select the repository (if have) and branch' pointing to it. Below the dropdowns, a green checkmark indicates 'Able to merge. These branches can be automatically merged.' The next section is 'Add a title *', with a text input field containing 'Demonstrate on pull request'. Below that is the 'Add a description' section, which has a 'Write' tab and a 'Preview' tab. The 'Write' tab is active, showing a rich text editor with a toolbar and a large text area containing the placeholder 'Add your description here...'. At the bottom of the form, there's a checkbox labeled 'Allow edits by maintainers' which is checked, and a green button labeled 'Create pull request' with a dropdown arrow. A red box highlights this button, with a red text annotation 'And write the comment and submit pull request' pointing to it.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base repository: HKUST-CRS/crs base: main head repository: minglo17794/crs compare: feat-sort-order

✓ Able to merge. These branches can be automatically merged.

Add a title *

Demonstrate on pull request

Add a description

Write Preview

Add your description here...

Markdown is supported Paste, drop, or click to add files

Allow edits by maintainers Create pull request

Part VI: Workshop Task

The workshop task is a hands-on assignment that combines all the Git and GitHub skills you've learned. You'll practice forking, branching, generating a hash for your identity data, editing the student record, and submitting a pull request.

What you'll do:

- Fork the assignment repository to your GitHub account
- Clone your fork to your local machine
- Create a feature branch for your changes
- Create and activate a `.venv` virtual environment with Python 3.13.7
- Run `python3 Data/generate_hash.py` and copy the hash-value output
- Edit your student record in `Data/student.json` with your GitHub username and hash-value
- Commit and push your changes
- Submit a pull request to the original repository

- Receive automated validation feedback

Repository: <https://github.com/minglol7794/HKUST-COMP4900-2026S-Task>

The screenshot shows a GitHub repository page for 'HKUST-COMP4900-2026S-Task' by user 'minglol7794'. The repository is public and has 1 branch (master) and 0 tags. The file list shows several files: .github, Assets, Data, .gitignore, and README.md, all with their commit history and timestamps. The README file is selected, showing the title 'HKUST 2026 Spring COMP 4900 GitHub Training Module Task' and a welcome message. The right sidebar contains sections for 'About' (no description), 'Releases' (no releases published), 'Packages' (no packages published), and 'Contributors' (1 contributor: minglol7794).

minglol7794 / HKUST-COMP4900-2026S-Task

Code Issues Pull requests 1 Agents Actions Projects Wiki Security Insights Settings

HKUST-COMP4900-2026S-Task Public

master 1 Branch 0 Tags

Go to file Add file Code

minglol7794 fix: workflow edit again 8a312db · 11 minutes ago 3 Commits

.github	fix: workflow edit again	11 minutes ago
Assets	init: initial commit	41 minutes ago
Data	init: initial commit	41 minutes ago
.gitignore	init: initial commit	41 minutes ago
README.md	init: initial commit	41 minutes ago

README

HKUST 2026 Spring COMP 4900 GitHub Training Module Task

Welcome to the COMP 4900 GitHub Training Module task. You can review the training module material website [here](#)

Task

In this exercise, you will practice a standard open-source contribution workflow:

1. Fork a repository

About

No description, website, or topics provided.

Readme Activity 0 stars 0 watching 1 fork

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)

Contributors 1

minglol7794 minglol7794

Follow these steps in order. Each step builds on the previous one—don't skip any.

Workflow:

1. **Fork** the original task repository to your GitHub account
2. **Clone** your fork to your local machine to work offline
3. **Create a branch** named something like `add-my-student-info` to isolate your work
4. **Create and activate** a `.venv` virtual environment with Python 3.13.7

```
=== COMP4900 Hash Generator ===  
Enter your 8-digit student ID: 20000000  
Enter your ITSC email (xxxx@connect.ust.hk): demo@connect.ust.hk  
  
Copy this value into Data/student.json:  
hash-value: 281c27373e940d6a60f7fb11fd314c66eb0a58326d2acde9ba30a14db37c3477
```


5. **Run** `python3 Data/generate_hash.py` and copy the hash-value output
6. **Edit** `Data/student.json` with your GitHub username and hash-value; keep the JSON valid
7. **Commit** your changes with a clear message
8. **Push** your branch back to your fork on GitHub
9. **Open a pull request** to propose your changes to the original repository
10. **Wait for validation**—GitHub Actions will automatically verify your submission
11. **Submit** the pull request link on Canvas once validation passes

Once your pull request passes validation, you're ready to submit.

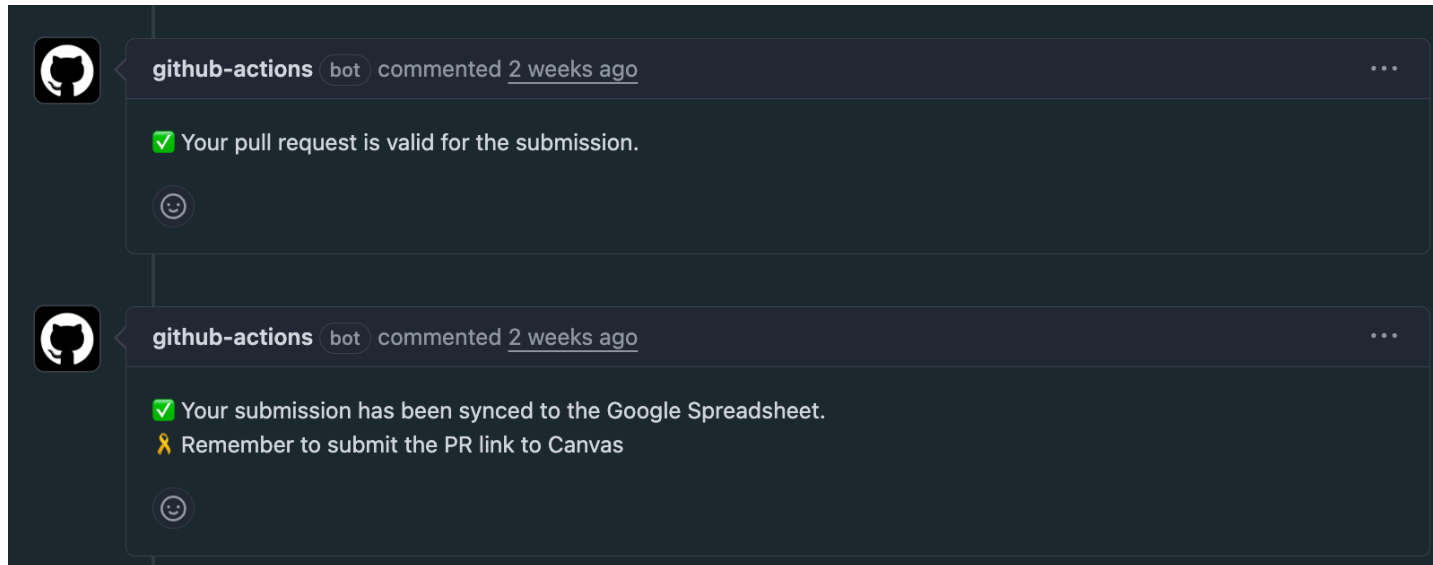
Final steps:

1. **Check the PR** for green checkmarks—this means GitHub Actions validated your JSON format and changes
2. **Copy the PR link** from the browser address bar (the full URL)
3. **Submit on Canvas** by pasting that link in the assignment submission
4. **Keep the PR open** until validation is complete (do not close it)

Canvas submission URL:

https://canvas.ust.hk/courses/67791/assignments/424723?module_item_id=1753451

Screenshot showing successful validation:



Before submitting on Canvas, verify that you've completed all steps correctly:

Checklist:

- ✓ You forked the original repository (shows your username in the fork)
- ✓ You created and worked on a separate branch (not directly on main)
- ✓ You created and activated a `.venv` virtual environment
- ✓ You ran `python3 Data/generate_hash.py` locally and copied the hash output
- ✓ You updated only `Data/student.json` with your GitHub username and hash-value
- ✓ Your JSON syntax is valid (green checkmark from GitHub Actions)
- ✓ Your pull request is open and visible on GitHub
- ✓ Your pull request link is submitted through Canvas

If all items are checked, the assignment is complete!

If you encounter problems, try these solutions:

Common issues:

- **Push failed:** Check `git remote -v` to verify your remote is set correctly
- **Can't find your branch:** Confirm the branch name with `git branch -a`
- **Authentication error:** Verify your SSH keys or GitHub credentials in VS Code
- **Validation failed:** Read the GitHub Actions comment in your PR—it explains what's wrong
- **JSON format error:** Open `Data/student.json` and check for missing commas or quotes
- **Can't see your changes:** Make sure you committed AND pushed (don't forget push!)
- **Pull request from wrong branch:** You can change the source branch in the PR settings

Important: Mark your calendar!

- **Deadline: 9 May 2026, 23:59:00 HKT**
- Follow the Canvas deadline as the final source of truth
- Late submissions may not be accepted—submit early to avoid issues
- If you have technical problems, ask for help before the deadline

Part VII: Additional Resources (Self-study)

Here are the most important Git commands you'll use repeatedly. Bookmark this for quick reference!

Repository setup:

```
git init                # Create a new repo
git clone <url>         # Copy an existing repo
git remote add origin <url> # Link to GitHub
```

Daily workflow:

```
git status              # Check what changed
git add <file>          # Stage changes
git commit -m "msg"     # Save snapshot
git push               # Upload to GitHub
git pull               # Download from GitHub
```

Branching:


```
git branch <name>      # Create branch  
git checkout <name>     # Switch branch  
git merge <name>        # Merge branch
```

As you continue learning Git and GitHub, these resources will help you:

Official sources (always accurate):

- **GitHub Docs:** <https://docs.github.com>
 - Official guides for GitHub features
- **Git documentation:** <https://git-scm.com/doc>
 - Deep dives into Git command details
- **Pro Git book:** <https://git-scm.com/book>
 - Free, comprehensive book on Git concepts

Tutorial materials:

- **This workshop source:** https://github.com/minglol7794/minglol7794.github.io/tree/github-tutorial-update/github_tutorial
 - See how this training was built

Keep these cheat sheets handy for quick command lookup:

Official cheat sheets:

- **GitHub Education Cheat Sheet:** <https://education.github.com/git-cheat-sheet-education.pdf>
 - Concise reference of common Git commands
- **GitHub Training Kit:** <https://github.github.com/training-kit/downloads/github-git-cheat-sheet/>
 - Visual guide to GitHub workflows

How to use them:

- Bookmark these links
- Print them out or save as PDF
- Reference them when you forget command syntax
- Don't try to memorize everything—referencing is normal!

Merge conflicts occur when two branches change the same lines of code differently. They're normal and solvable!

What causes conflicts:

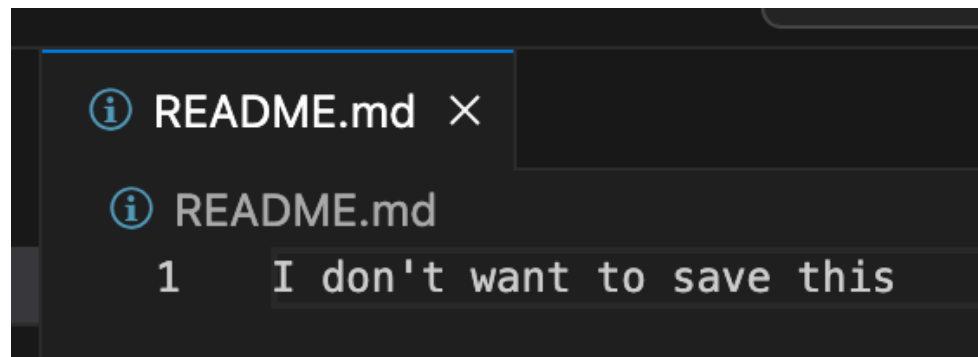
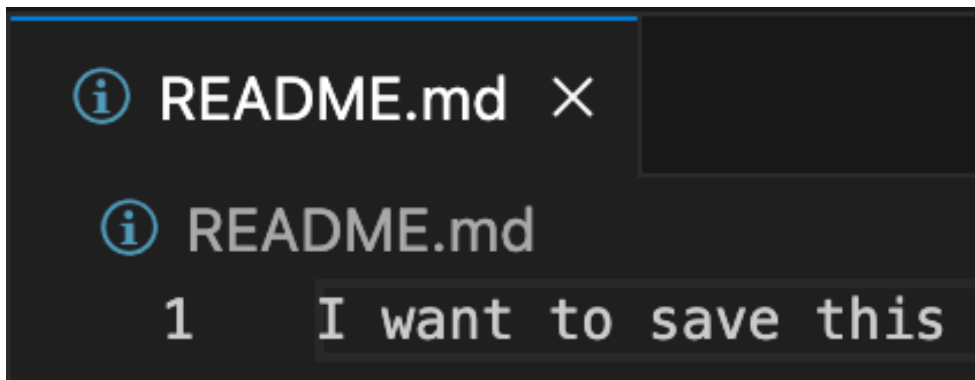
- You and a teammate both edit the same file
- The same line changed differently on each branch
- Git can't automatically decide which version to keep

How to resolve:

- Open the conflicted file in VS Code's Merge Editor
- Review the differences side-by-side
- Choose which changes to keep (yours, theirs, or both)
- Delete the conflict markers
- Stage and commit the resolved file
- Continue with your merge

Pro tip: Communicate with teammates about who's editing what to minimize conflicts!

```
git status  
git add <file>  
git commit  
git merge --abort
```



- Use `git revert` to safely undo changes from an earlier commit.
- Unlike `git reset`, it does not rewrite history.
- It creates a new commit that reverses the target changes.

```
git log --oneline
```

```
git revert <commit-hash>
```

```
git revert HEAD
```

```
git revert -m 1 <merge-commit-hash>
```

