Git & GitHub Training Module

COMP 4900 Academic and Professional Development

WONG, Lap Ming

Hong Kong University of Science and Technology

15th April, 2026

WONG, Lap Ming

Local Year 3 Computer Science Student

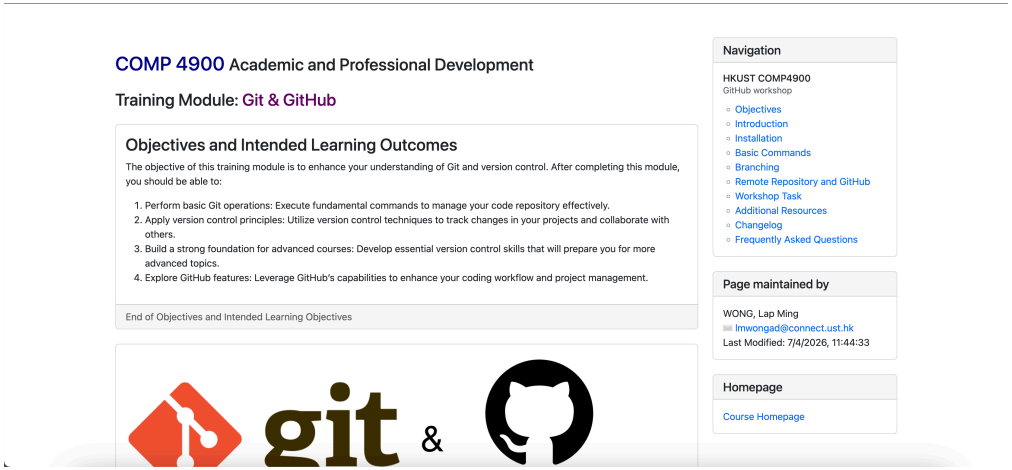
Email: lmwongad@connect.ust.hk

Experience in teaching:

- 2025 Spring: COMP 2012 UGTA
- 2025 Fall: UTOP 3201 (Prompt Engineering Workshop)
- 2025 Fall - 2026 Spring: COMP 1023 UGTA

Before Anything start

- 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



Objectives and Intended Learning Outcomes

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

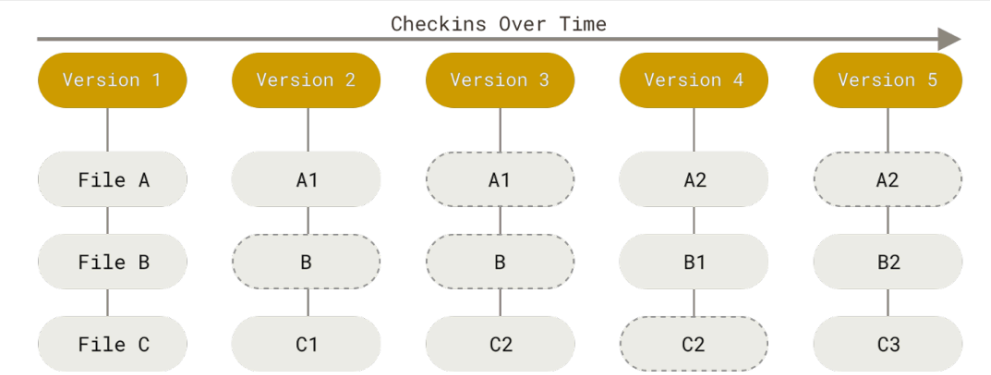
Part I: Version Control Fundamentals

What is Version Control

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.



What is Git

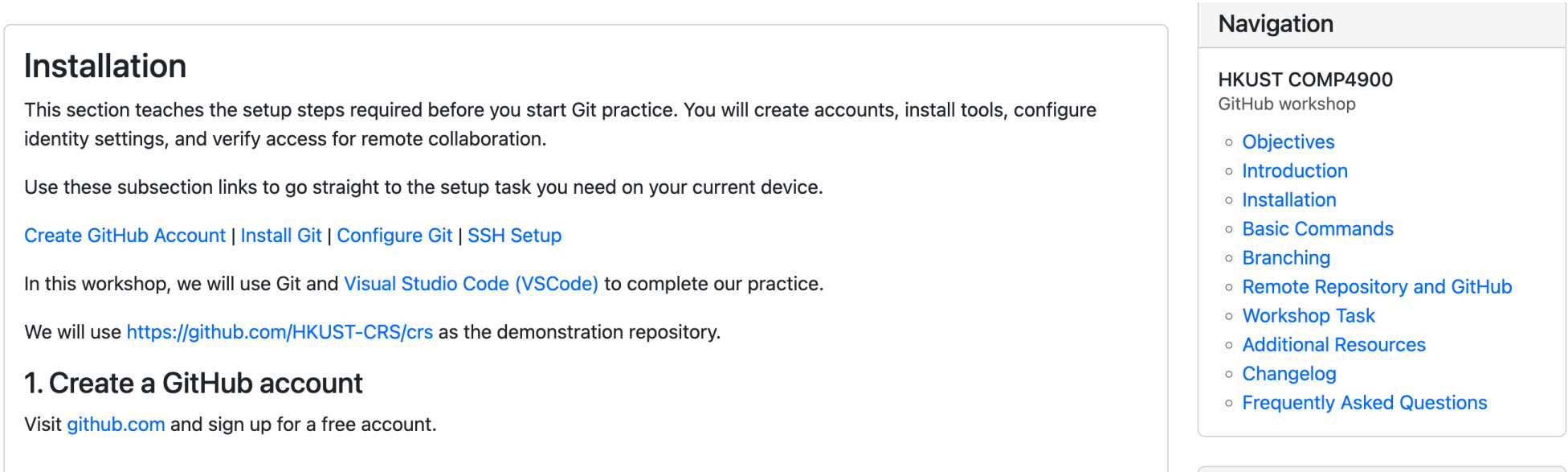
- **Git** is a distributed version control system that tracks changes in your code over time.
- **Git** stores work as a stream of snapshots.
- A **Git repository** contains a hidden `.git` folder inside the project directory.
- **Git** is the tool that manages local history, branches, and commits.

What is GitHub

- **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 (Self-study)

- For step-by-step installation guidance, refer back to the workshop website: https://minglol7794.github.io/github_tutorial/COMP4900-Website/#installation



Installation

This section teaches the setup steps required before you start Git practice. You will create accounts, install tools, configure identity settings, and verify access for remote collaboration.

Use these subsection links to go straight to the setup task you need on your current device.

Create GitHub Account | Install Git | Configure Git | SSH Setup

In this workshop, we will use Git and Visual Studio Code (VSCode) to complete our practice.

We will use https://github.com/HKUST-CRS/crs as the demonstration repository.

1. Create a GitHub account

Visit github.com and sign up for a free account.

Navigation

HKUST COMP4900
GitHub workshop

- Objectives
- Introduction
- Installation
- Basic Commands
- Branching
- Remote Repository and GitHub
- Workshop Task
- Additional Resources
- Changelog
- Frequently Asked Questions

Part III: Git Fundamentals

Getting a local Git repository

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.

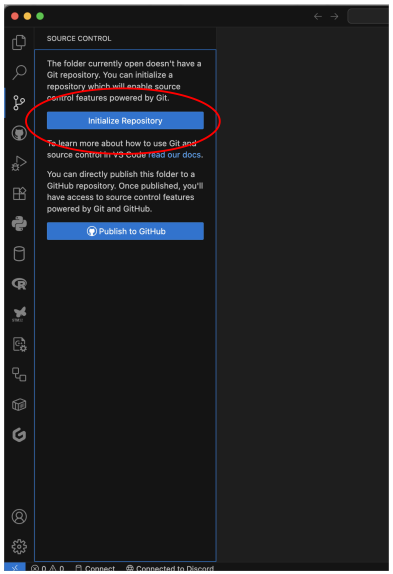
VS Code method:

Initialize a repository: Sets up Git to track your project

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

```
git init
```

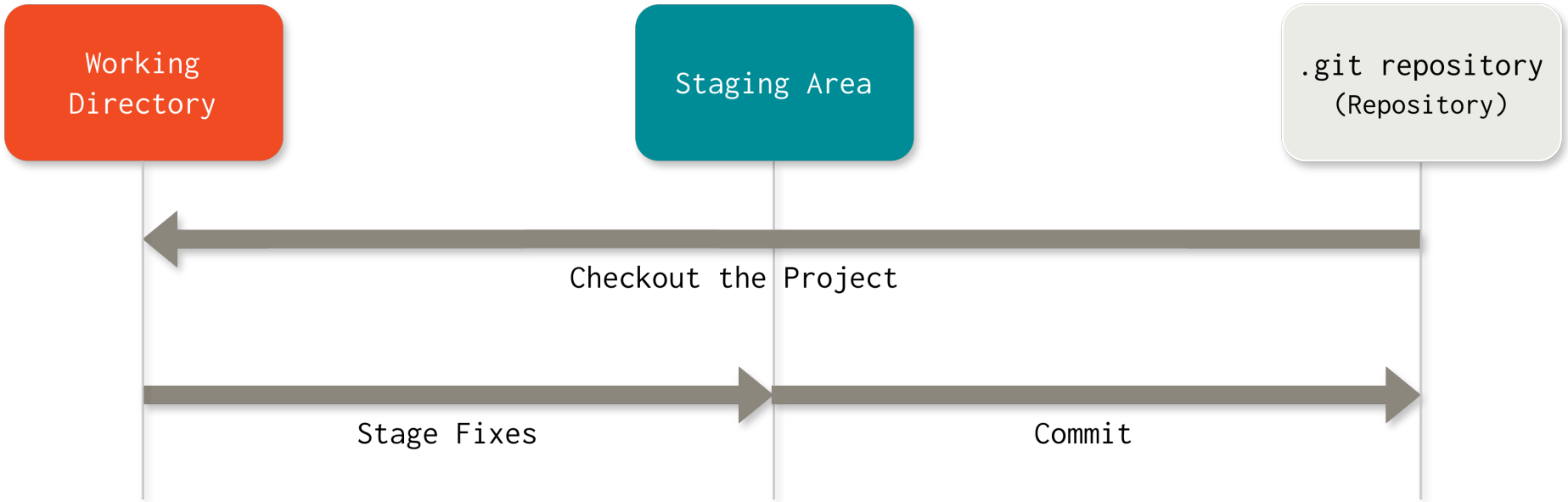
This creates the hidden `.git` folder in your project root.



Basic workflow of a Git project

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.



Add a file to the staging area

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.

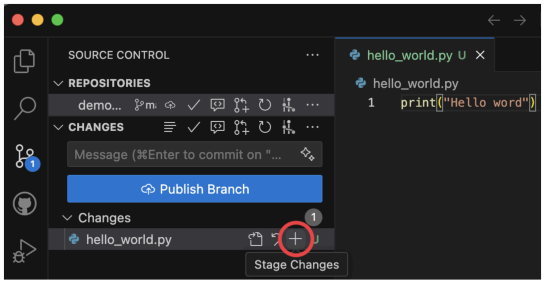
Stage files: Marks them ready for the next commit

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

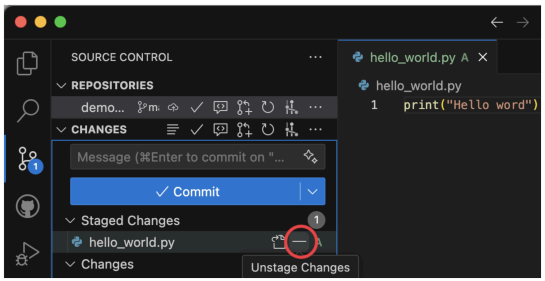
Unstage files: Removes them from staging if you changed your mind

- Or in terminal: `git reset filename`

```
git add hello_world.py
```




```
git reset hello_world.py
```



Commit a file to the repository

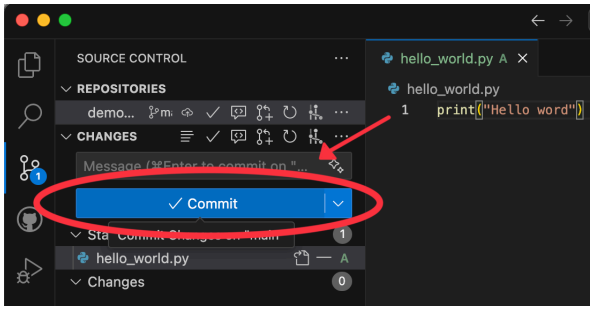
Committing saves a permanent snapshot of your staged changes. Each commit is a saved point in history that you can return to later.

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

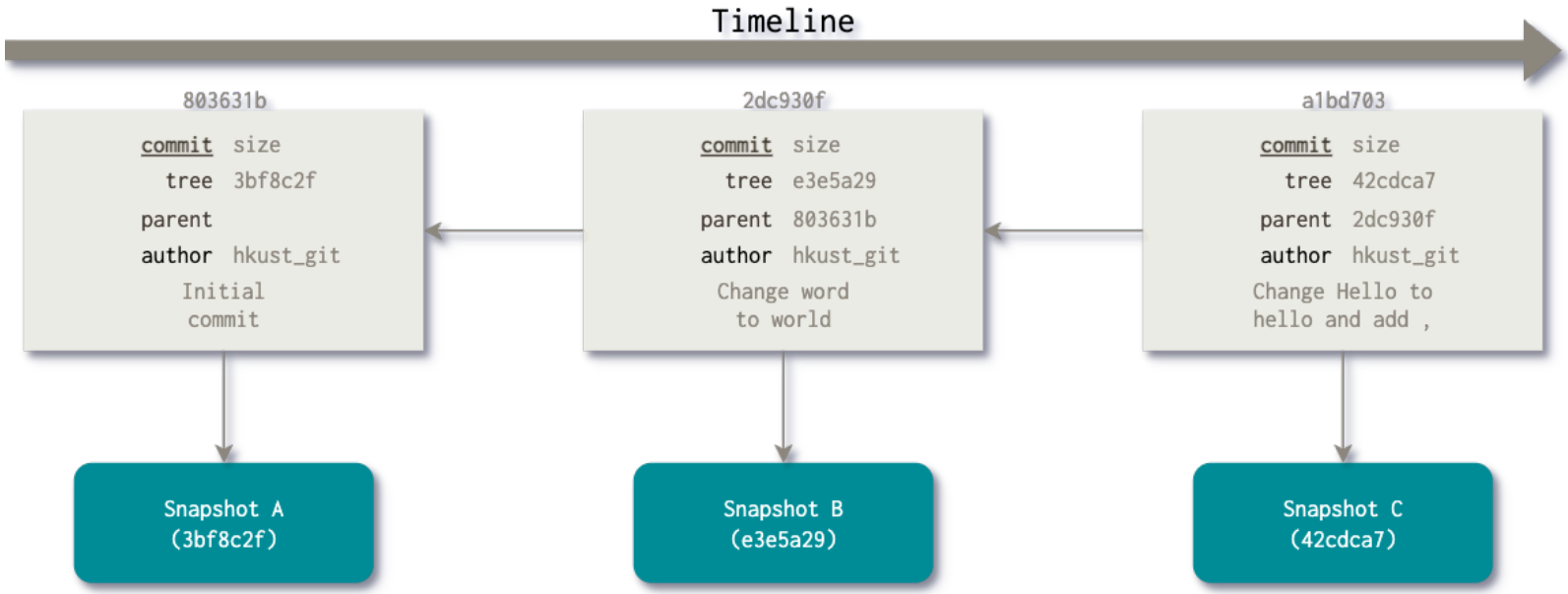
```
git status
git commit -m "commit message"
```

VS Code method:



Commit object and 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.



Commit object and commit message

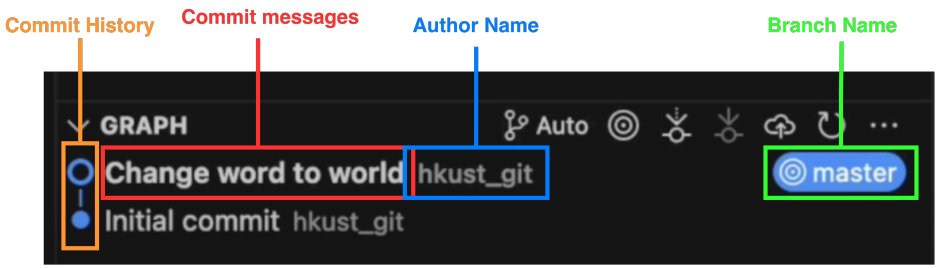
How to view commit messages:

- `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:

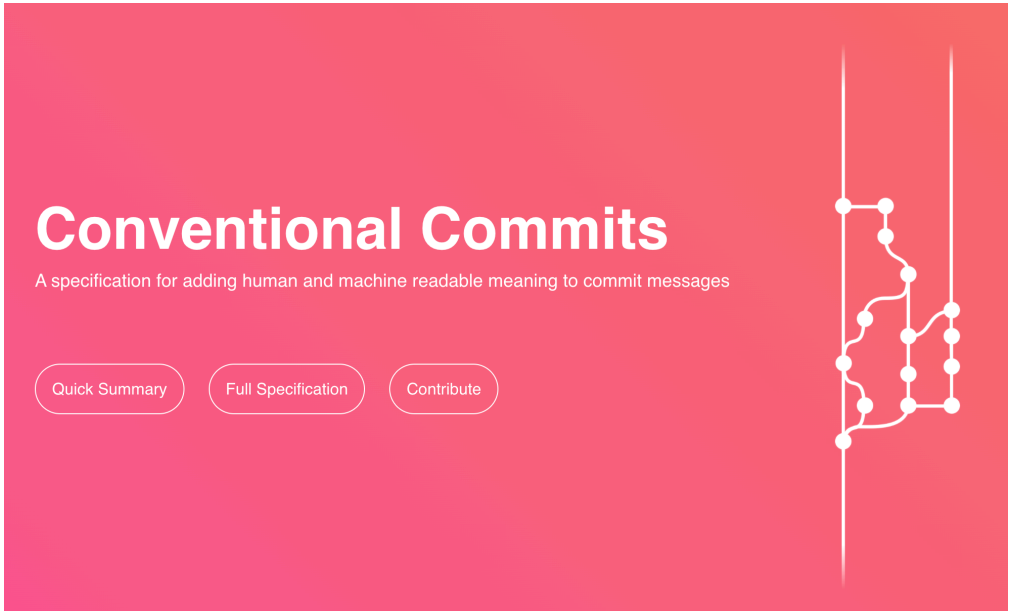
```
git log --oneline
git show <commit-hash>
```

Commit history in VS Code:



When writing commit message..

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 throught this link: https://www.conventionalcommits.org/en/v1.0.0/



When writing commit message..

You definitly don’t want to see something like this…

done redesign level 2
committed on Jan 26, 2025 f02c14b

12311
committed on Jan 26, 2025 37c1d81

123
committed on Jan 26, 2025 619adf5

making Fan
committed on Jan 26, 2025 9d75dca

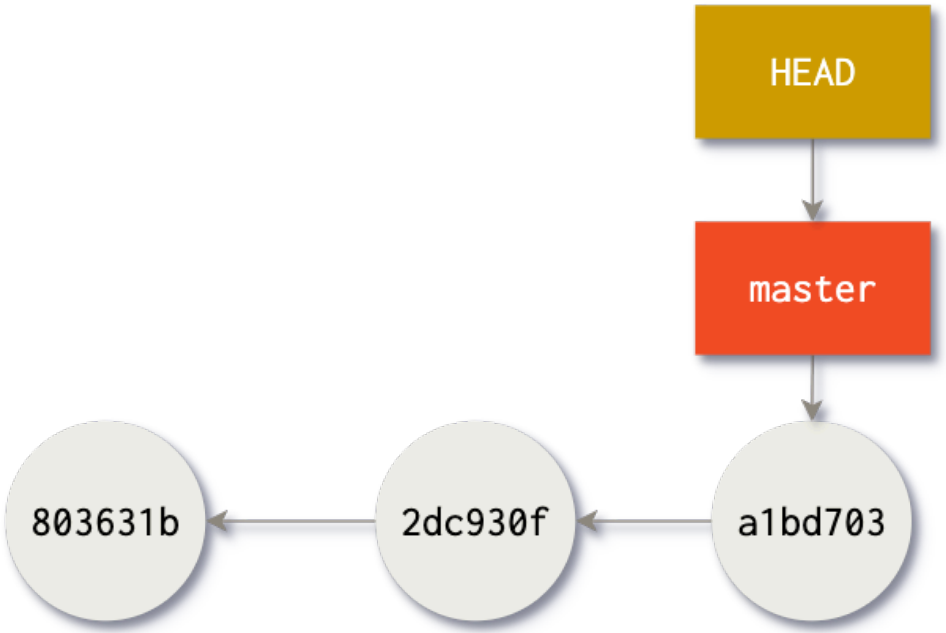
Added Andy's Map
committed on Jan 26, 2025 9b7eab3

...
committed on Jan 26, 2025 de9c381

Part IV: Branching and Merging

Git branching

- 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.



Creating and switching branches

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
```

Merge 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
```

Part V: Remote Repositories and GitHub

Remote repository

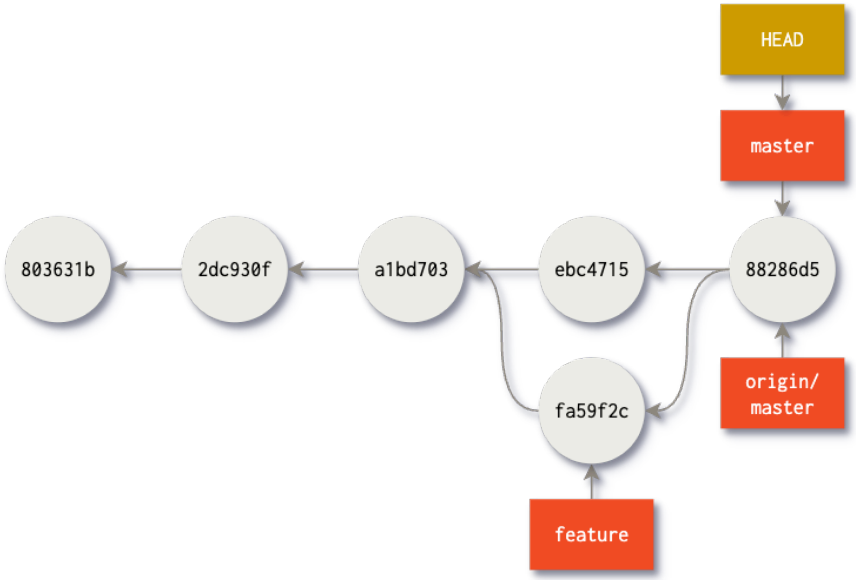
A remote repository is the online backup of your project (hosted on GitHub, GitLab, etc.). It enables collaboration and protects your code.

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



Set up a GitHub repository

We will use GitHub to setup our remote repository this time. I will demonstrate how to setup on main screen :3

Add a remote in your local repository

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
```

Remove a remote in your local repository

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>
```

Push your work to the remote repository

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
```

Sync from the remote repository

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>
```

Git clone

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

What cloning does:

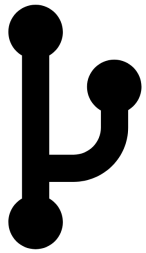
- Downloads the entire remote repository to your computer
- Includes all branches and commit history
- Automatically sets up `origin` as the default remote
- Creates a folder for the project
- Ready to work immediately without any setup

Fork on GitHub

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

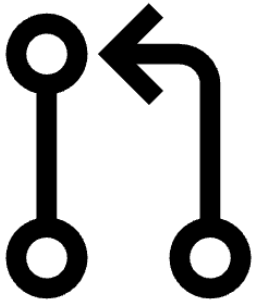


Pull request workflow

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



Part VI: Workshop Task

Workshop task overview

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.

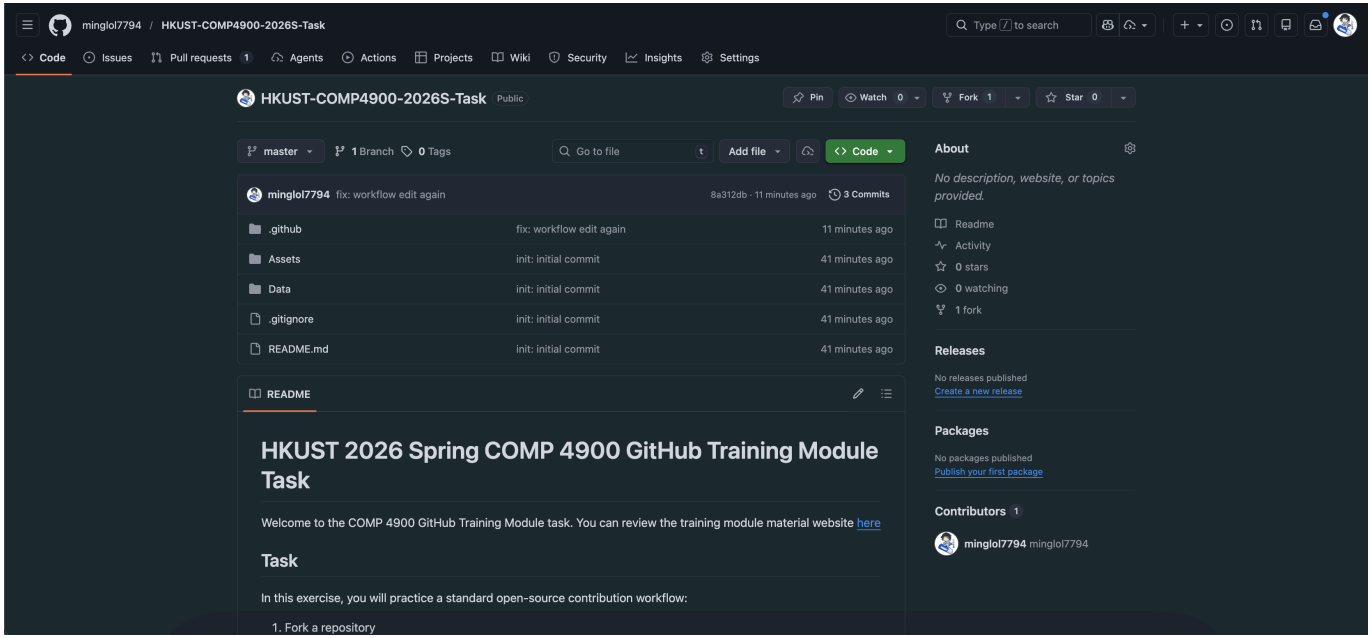
What you'll do:

- 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

Workshop task overview

- Receive automated validation feedback

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

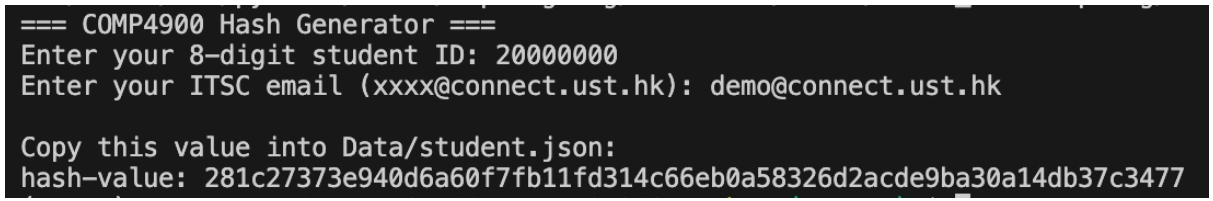


Workshop task instructions

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



Workshop task instructions

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

Success and submit

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

Final steps:

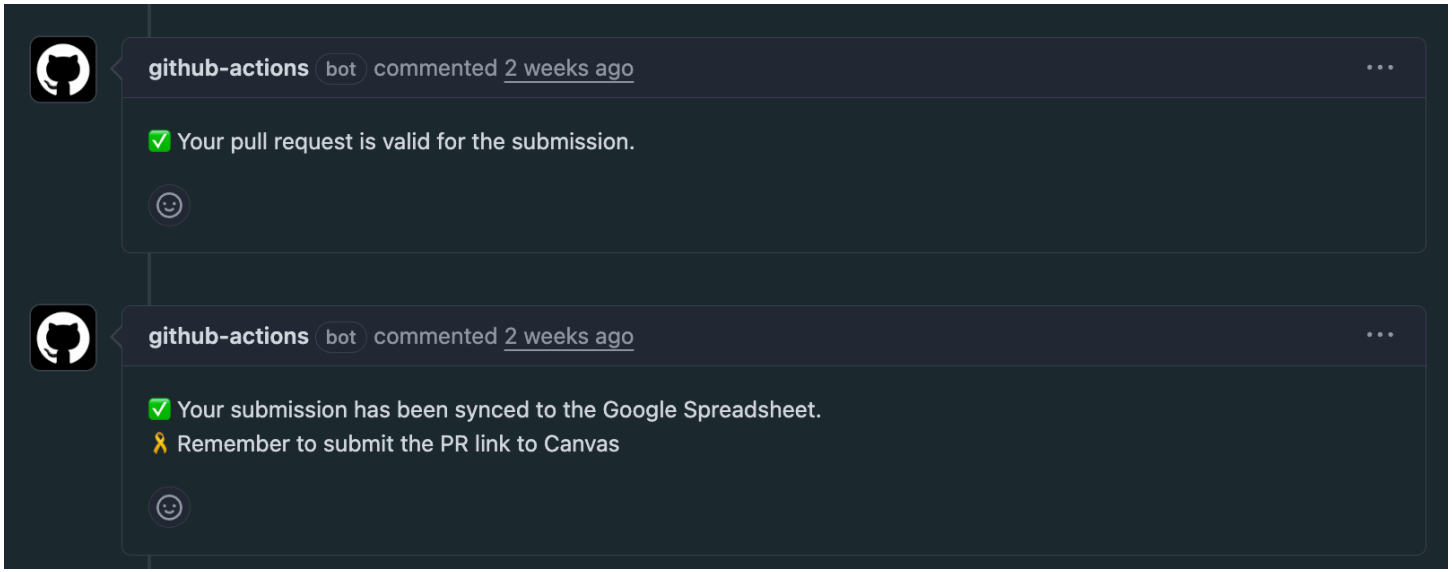
1. **Check the PR** for green checkmarks—this means GitHub Actions validated your JSON format and changes
2. **Copy the PR link** from the browser address bar (the full URL)
3. **Submit on Canvas** by pasting that link in the assignment submission
4. **Keep the PR open** until validation is complete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

Canvas submission URL:

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

Success and submit

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!

Troubleshooting

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

Deadline

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)

Additional Resources

Please refer back to the workshop website:

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

Any Question?