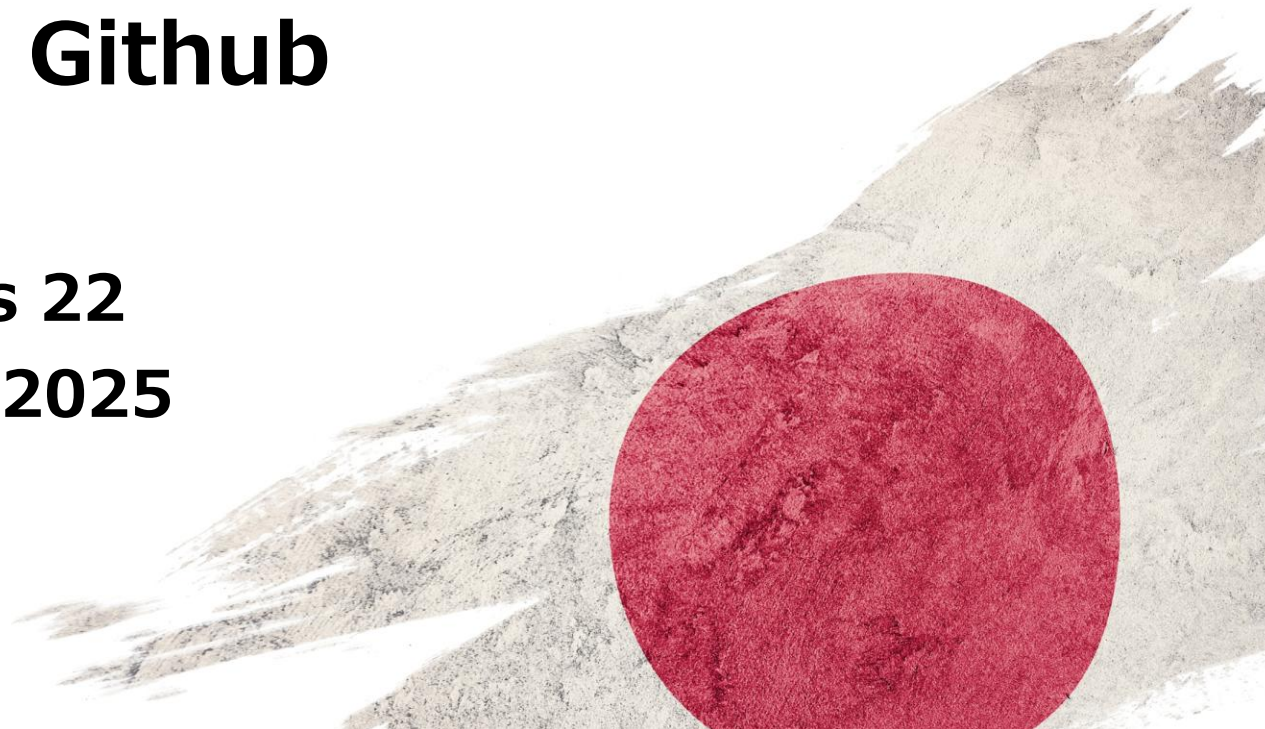
Track your code modification with Git and Github

Class 22

26/7/2025



Acknowledgement

The series of the IT & Japanese language course is Supported by AOTS and OEC.

Ministry of Economy, Trade and Industry

Overseas Employment Corporation

What you have Learnt Last Week

We were focused on following points.

- Usage of control and loop flow statement
- Performing Linear Algebra in Numpy
- Why Requirement Analysis is so important in the process?
- Software development Life cycle
- Importance of Security compliance
- Introduction of Bash Scripting
- Introduction of Ansible, docker and docker compose
- API testing with Postman

What you will Learn Today

We will focus on following points.

1. Introduction of Git and Github
2. Setting up account for Github
3. Explanation of essential Git commands (add, commit, push, pull, clone) to manage code versions and collaborate
4. How to create branches, work on new features, and merge changes without disrupting the main codebase
5. Quiz
6. Q&A Session

What is Version Control?

Track, manage, and collaborate on code changes

Definition: Version control is a system that records changes to a file or set of files over time.

Purpose:

- Keep track of every modification
- Restore earlier versions
- Collaborate efficiently

Types:

- Local
- Centralized
- Distributed (like Git)

Introduction to Git

A powerful distributed version control system

What is Git?

Git is a **distributed version control system** that tracks changes in source code.

Key Features:

- Offline capabilities
- Branching and merging
- Fast performance

Created By: Linus Torvalds (2005)

What is GitHub?

Introduction

What is GitHub?
A cloud-based **hosting service for Git repositories**.

Features:
- Remote collaboration
- Pull requests
- Issues, Wikis, Actions

Git vs. GitHub:
- Git: Tool
- GitHub: Hosting platform for Git projects

Difference Between Git and GitHub

Tool vs Platform — how they work together

Feature	Git	GitHub
Type	Tool (local)	Platform (online)
Use Case	Track code changes	Share and collaborate on code
Internet Needed	No	Yes (for pushing/pulling)
Installed?	On your computer	Web-based

Why Git is Important for Developers and Teams

Enabling individual productivity and team collaboration

- **For Developers:**
- Track changes easily
- Try out new features with branching
- Roll back if something breaks

- **For Teams:**
- Collaborate on same project
- Handle merge conflicts
- Track who did what, and when

Real-World Use Cases of Git & GitHub

From startups to open source — universal tools

- **Software Development** – used by all major tech companies
- **Open Source Projects** – anyone can contribute
- **Versioned Documentation** – track changes in documents
- **Education** – share assignments, projects, feedback
- **Visual:** GitHub project page of a real repo (e.g., VS Code)

Overview of Repositories

our project’s home — locally and remotely

What is a Repository?
A folder where your project and its history is stored.

Two Types:

- **Local Repo**: On your computer
- **Remote Repo**: On GitHub

Commands:

git clone, git push, git pull

Creating a GitHub Account

First step to becoming a contributor

Steps to Sign Up:

1. Visit github.com
2. Click **Sign Up**
3. Enter email, username, password
4. Verify email

Tip: Use a professional username!

Setting Up Your GitHub Profile

Build your professional presence on GitHub

Elements to Fill:

- Name
- Bio
- Profile Picture
- Location
- Website (optional)

Why it matters:

- Builds trust for collaborators
- Looks professional

Generating & Adding SSH Keys

Secure authentication without typing your password

Why SSH?
Securely push/pull without entering credentials each time.

Steps:

1.ssh-keygen -t ed25519

2.Copy key using cat ~/.ssh/id_ed25519.pub

3.Go to GitHub → **Settings** → **SSH Keys**

4.Paste and save

Configuring Git on Your Local Machine

Tag your work with your identity

Initial Setup Commands:

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

Verify Settings:

```
git config --list
```

Purpose:
These settings are used to tag commits with your identity.

Introduction to Essential Git Commands

Manage code efficiently with Git basics

- Git helps you track changes, collaborate, and revert code safely.
- Core commands form the daily workflow of developers.
- You’ll learn how to create, update, and sync your project.

Creating & Cloning Repositories

Start your Git journey

git init – Initialize a new local repository

→ Creates a .git directory for tracking

git clone <url> – Clone an existing remote repository

→ Copies the entire project history

Example:

git init
git clone https://github.com/username/repo.git

Tracking and Staging Changes

Prepare changes before committing

git add stages files for commit

→ Adds modified/created files to Git’s staging area

Staging = “What you want to include in the next save”

Example:

```
git add index.html

git add .   # Adds all changes
```

Saving Changes (Commits)

Store your progress permanently

- **git commit -m "message"** – Saves staged changes to history
- Commit = Snapshot + Message
- Good messages help collaborators understand changes

Example:

git commit -m "Added login form"

Synchronizing with Remote

Push and pull changes from remote repositories

- **git push** – Upload local commits to remote
- **git pull** – Fetch and merge remote changes to local
- Keeps your work in sync with team or backup

Example:

git push origin main

git pull origin main

Viewing Status and History

Inspect your codebase and logs

- **git status** – Shows changes (staged/unstaged/untracked)
- **git log** – View commit history
- **git diff** – See line-by-line differences between versions

Example:

git status

git log

git diff index.html

What is Branching?

Work on new features without breaking your code

- Branch = Independent line of development
- Default branch is usually main
- Helps isolate features, bug fixes, experiments

Real-life analogy: Branch = Copy of code to experiment without risk

Creating and Switching Branches

Handle multiple tasks in parallel

- **Create:**

git branch feature-xyz
git checkout -b feature-xyz *(create & switch)*

- **Switch:**

git checkout main – Change to another branch

Tip: Use meaningful names like bugfix-login or feature-payment

Merging and Resolving Conflicts

Combine changes safely

- **git merge branch-name** – Merge feature into current branch
- Git auto-merges if no conflicts
- Conflicts arise when same line is edited in both branches

Conflict resolution steps:

1. Edit conflicting files
2. Mark what to keep
3. Commit the changes

Deleting Branches After Merging

Keep your repo clean

Delete merged branches to avoid clutter

Command:

git branch -d feature-xyz

Use -D to force delete if unmerged

Pro Tip: Use pull requests on GitHub to handle merge, review & delete

Introduction to Pull Requests

Collaborate before merging code

- A **Pull Request (PR)** proposes changes from one branch to another
- Used in collaborative platforms like GitHub
- Enables **discussion, review, and approval** before merging
- Encourages code quality and team visibility

Think of a PR like: “Here’s what I changed—can you review before I merge?”

PR vs Direct Merge

Why PRs are better than skipping reviews

Pull Request	Direct Merge
Reviewed by team	No review
Discussion & feedback	One-way change
Can be declined	Always accepted
Promotes collaboration	Risk of breaking code

Use PRs to avoid bugs, improve quality, and encourage shared ownership

Creating a Pull Request (GitHub UI)

Submit your code for review

- Push your branch to GitHub
- Click **"Compare & pull request"**
- Add title and description
- Assign reviewers (optional)
- Click **"Create pull request"**

Code Review and Comments

Improve code through feedback

- Team members review the PR
- Leave inline comments on specific lines
- Request changes or approve
- Helps spot bugs, improve readability, and share knowledge

Merging Pull Requests

Finalize and apply reviewed changes

Options on GitHub:

- **Merge commit:** Keeps full history
- **Squash and merge:** Combines commits into one
- **Rebase and merge:** Linear history, cleaner log

Choose based on your team’s workflow

Handling Conflicts in PRs

Fix overlapping changes before merging

- Happens when two branches change the same lines
- GitHub highlights conflicts

Steps to resolve:

1. Fetch the latest main branch
2. Merge into your branch locally
3. Resolve conflicts in code
4. Commit and push again

Best Practices for Pull Requests

Make PRs easy to review and merge

- ✅ Keep PRs small and focused
- ✅ Write clear titles and descriptions
- ✅ Use meaningful commit messages
- ✅ Link related issues
- ✅ Be open to feedback and quick to respond

Assignment

Assignment

1. Create a new repository
2. Add some files
3. Create a branching strategy like dev , feature, and stg
4. Create PR for merging into dev branch
5. Practice the git fetch and git pull commands

Quiz Section

Quiz

Everyone student should click on submit button before time ends otherwise MCQs will not be submitted

[Guidelines of MCQs]

1. There are 20 MCQs
2. Time duration will be 10 minutes
3. This link will be share on 12:25pm (Pakistan time)
4. MCQs will start from 12:30pm (Pakistan time)
5. This is exact time and this will not change
6. Everyone student should click on submit button otherwise MCQs will not be submitted after time will finish
7. Every student should submit Github profile and LinkedIn post link for every class. It include in your performance

Assignment

Assignment should be submit before the next class

[Assignments Requirements]

1. Create a post of today's lecture and post on LinkedIn.
2. Make sure to tag @Plus W @Pak-Japan Centre and instructors LinkedIn profile
3. Upload your code of assignment and lecture on GitHub and share your GitHub profile in respective your region group WhatsApp group
4. If you have any query regarding assignment, please share on your region WhatsApp group.
5. Students who already done assignment, please support other students

Q&A Session

ありがとうございます。
Thank you.
شکریا

For the World with Diverse Individualities