Git & GitHub: A Beginner's Guide for Research Collaboration

Chiatse Wang

Institute of Statistical Science, Academia Sinica

Jul 18, 2025

Statistical School 2025

Main Takeaways

- Learn how to manage projects and collaborate effectively using GitHub.
- Explore a real-world scenario through a hands-on demo.
- All materials are available at: `https://github.com/chiatsewang/github-collaboration-tutorial`

Outline

Introduction

Introduction

- Git is a distributed version control system.
- Cloud-based Git services (e.g., GitHub, GitLab, Bitbucket) are used for collaboration and backup.

Version Control

Version control helps maintain a history of changes in code and data.

- No more confusing filenames like `code_final_v2.py` or `proj_20250425`.

Why is version control important?

1. Track issues and bugs; audit or revert changes when needed.
2. Manage multiple versions of a project efficiently.
3. Reproduce results consistently and reliably.
4. Collaborate smoothly with others.

Git Basics

Git Overview

1. The concepts of staging and branching are fundamental in Git.
2. A commit captures a snapshot of your project at a specific point in time.
3. The core unit of a Git project is the repository — it stores all commits and branches.
4. Git tracks changes in files and directories over time.
5. Merging branches combines their changes into one unified version.

Git Commit: Saving Your Work

1. Make changes: Edit or create files in your project.
2. `git add`: Marks selected changes to be included in the next version.
3. `git commit`: Saves a version of your work by recording a snapshot of the project.
4. Each commit acts like a checkpoint — similar to saving progress in a game.
5. You can always go back to a previous version if needed.

Branching in Git

- A **branch** represents an independent line of development starting from a specific commit.
- The default branch is usually named `main` or `master`.
- `git checkout branch-name`: Switch to another branch.
- You can create new branches to work on features or ideas without affecting the `main` branch.

Why use branches?

- Isolate development work.
- Enable parallel feature development.
- Safely experiment with new ideas.

Git Pull: Compare and Merge Branches

1. Git allows you to combine branches that contain different work.
2. `git pull` is a shortcut that:
   - 2.1 Downloads the latest changes from a target branch.
   - 2.2 Merges those changes into your current branch.
3. If both branches changed the same parts of the project, Git may ask you to resolve a conflict manually.

Forking a Git Repository

- A **repository** stores your project's files, commit history, and branches — like a versioned project folder.
- **Fork**:
  - Creates a copy of someone else's repository under your own GitHub account.
  - Includes the full history — all commits and branches — starting a new stream from the same point.
  - Ideal when you don't have direct write access — common in open source contributions.
  - You can make changes in your fork and open a pull request to propose them to the original repository.

GitHub Collaboration Best Practices

A Good Repository

Best practices for collaboration:

1. Document your project with a `README.md`.
2. Define usage terms with a `LICENSE`.
3. Use a `.gitignore` file to exclude unnecessary files.
4. Use clear and consistent branch naming conventions.
5. Write meaningful commit messages.
6. Keep commits small and focused.
7. Use pull requests for code review and discussion.
8. Track bugs and feature requests via issues.
9. Tag releases clearly.

README, LICENSE, and .gitignore

- `README.md`: Describes the project — includes an overview, installation steps, and usage examples.
- `LICENSE`: Defines how the project can be used, modified, and shared (e.g., MIT, Apache 2.0, GPL).
- `.gitignore`: Lists files and folders Git should ignore (e.g., `*.pyc` for Python cache files, `*.log` for logs).

Branching Strategy

- `main` — the stable, production-ready branch. Avoid making direct changes here.
- `dev` — the primary development branch.
- `test` — the main testing branch used for integration and QA.
- Branch naming convention:
  `account_name/feature/feature-name`,
  `account_name/bugfix/bug-name`.
- Feature branches should be short-lived and focused on specific tasks.

Writing Good Commits

- Start with a clear subject line using tags like `[feat]`, `[fix]`, `[docs]`, etc.
- Optionally add a body for more context or explanation.
- Use present tense and imperative mood.
- Example:
  - `[feat] Add data loader for experiment A`
  - `[fix] Correct typo in README`
  - `[docs] Update installation instructions`

GitHub Issues

- GitHub Issues are used to track bugs, feature requests, and tasks.
- Each issue can have a title, description, labels, and assignees.
- Use issues to discuss ideas and gather feedback.
- Link issues to commits and pull requests for better tracking.

Pull Request

- A **Pull Request (PR)** is a request to merge one branch into another on GitHub.
- Typically used to propose merging a contributor's feature branch into the main project branch.
- PRs enable code review, discussion, and collaboration before merging.

Best Practices

- Keep PRs small and focused.
- Provide context in the PR description.
- Respond to feedback promptly.

Pull Request Review Process

- PRs can be reviewed by team members or maintainers.
- Reviewers can comment, request changes, or approve the PR.
- Use comments to discuss code changes and suggest improvements.
- Once approved, the PR can be squashed and merged into the main branch.
- After merging, the feature branch can be deleted.

Hands-on Demo

Collaboration Simulation

1. Open an issue to describe the proposed change or feature.
2. Fork the repository to your own GitHub account.
3. Create a feature branch for your work.
4. Make changes and commit them with clear messages.
5. Submit a Pull Request (PR) to the original repository.
6. The maintainer reviews the PR and may add comments or request changes.
7. Revise your code based on the feedback.
8. The maintainer will merge the PR once it is approved.
9. Sync your fork with the latest updates from the original repository.

Part 1: Self-Introduction

Goal: Practice the complete GitHub Pull Request workflow

1. Open a GitHub Issue: `Add profile: yourname`
2. Fork the tutorial repository to your own GitHub account
3. Create a new branch: `yourname/feat/add-profile`
4. Add a self-introduction file `yourname.md` in the `profiles/` folder
5. Commit and push your changes; submit a Pull Request (PR)
6. The instructor reviews your PR and may request changes. Revise if needed and push again
7. Once approved, the PR is merged
8. Sync your fork with the original repository

Part 2: Statistics School 2025 Knowledge Cards

Goal: Collaboratively contribute a knowledge card

1. The class is divided into 8 groups, each assigned one course topic
2. Each group opens an Issue: `Add card: topic-name`
3. The group leader creates a branch: `groupX/feat/topic-name`
4. Add group members as collaborators in the forked repository
5. Co-edit the assigned file in `statistical-school-2025-cards/topic-name.md`
6. The group leader submits a PR and @mentions team members
7. Group members review, revise, and finalize the card
8. Once approved, the PR is merged and the fork is synced

Resources

- choosealicense.com
- conventionalcommits.org
- git-scm.com/doc
- docs.github.com
- opensource.guide

Git & GitHub: Recap

- Git helps you version, track, and manage changes in your projects.
- GitHub adds powerful collaboration features for teams and open source.
- Branches and pull requests support safe and structured teamwork.
- Clear commits, good documentation, and proper workflows lead to better collaboration.

Thank you!

GitHub: `@chiatsewang`
Email: `chiatsewang@stat.sinica.edu.tw`