Version Control, GitHub Collaboration, and Reproducible Team Workflow

Dylan Day, Shuo Li, Trevor Debutch, Shouhardyo Sarkar

The University of Iowa, Department of Statistics and Actuarial Science

October 28, 2025

Why Are We Talking About Version Control?

- We work in groups and need to edit the same files without overwriting each other.
- We want a record of **who changed what and when**.
- We want to be able to **go back in time** if something breaks.
- We want our work to be **reproducible** on any machine.

Version control is the system that tracks changes to files over time so you can recall or restore specific versions later.

What We'll Show in This Talk

- What Git and GitHub actually do (not just buzzwords).
- How teams collaborate using branches, commits, and pull requests.
- How to connect a local folder to a GitHub repo and push/pull.
- How this supports small coding projects (for class or research).

Git vs GitHub

Git

A distributed version control system: every person has a full copy of the repository, history and all. Fast, offline-friendly.

GitHub

A cloud platform **built on top of Git** that adds:

- remote hosting / backup
- pull requests & code review
- issues & project boards
- CI/CD automation

GitHub turns source code into a collaborative workspace.

GitHub Glossary

- **Repository (repo)**: the project folder on GitHub.
- **Commit**: a saved change with a message.
- **Branch**: a parallel version of the project used to develop features safely.
- **Pull Request (PR)**: a request to merge changes from one branch into another after review.

These concepts let multiple people experiment, review, and merge without breaking `main`.

Why Use GitHub in a Class Project?

- **Collaboration**: everyone can contribute code, documentation, or data without emailing files.
- **History**: you can see exactly how the project evolved.
- **Accountability**: each commit is tied to an author.
- **Safety**: if someone breaks something, you can roll back.
- **Publishability**: you can keep it private or make it public to show future advisors / employers.

Core Commands You Will Actually Use

- `git clone <repo>`
  download a copy of a GitHub repo to your computer
- `git add .`
  stage your edits (tell Git which changes you want to commit)
- `git commit -m "message"`
  record a snapshot of those changes
- `git pull` and `git push`
  sync with GitHub (pull down others' work, push up yours)
- `git branch`, `git merge`
  work on new ideas in isolation, then merge them back

These actions are the backbone of team development.

Branches and Pull Requests

- Each teammate can create a new branch (for example, `feature-ui`).
- You do your work there without touching the `main` branch.
- When ready, you open a **Pull Request** on GitHub:
  - show what changed
  - explain why
  - request review
- After review, the branch is merged into `main`.

This workflow gives you feedback, quality control, and a clean history instead of chaos.

Step 1: Clone the Repository

- One person creates a repo on GitHub.
- Everyone else runs:

```
git clone https://github.com/owner/RepoName.git
```

Now you have a full local copy of the project folder, including history and branches.

Step 2: Check the Remote Link

- Inside the project folder, verify GitHub is set as “origin”:

```
git remote -v
```

Expected:

```
origin  https://github.com/owner/RepoName.git (fetch)
origin  https://github.com/owner/RepoName.git (push)
```

If you see that, your local repo is correctly connected to the GitHub repo.

Step 3: Pull and Push

- **Pull** before you start editing:

```
git pull origin main
```

- After editing files:

```
git add .
git commit -m "explain what you changed"
git push origin main
```

If push is rejected, it means someone else updated `main`; run `git pull` to merge first.

Step 4: Authentication / Permissions

- GitHub requires you to prove who you are when pushing.
- If you see "Permission denied" or "Authentication failed":
  - You may need a Personal Access Token (PAT) instead of a password.
  - Or you may not have been added as a collaborator yet.
- Good news: pulling (reading) is often easier than pushing (writing).

How This Supports a Team Project

- Everyone can contribute in parallel:
  - Code / analysis scripts
  - README documentation
  - Example data or test cases
- Changes are tracked and credited to specific authors.
- The repo becomes a “single source of truth” for the project.
- You can show the final repo (or a link) as part of your assignment deliverable.

What We Practiced as a Group

- Setting up a shared GitHub repository.
- Cloning to each teammate's laptop.
- Making edits on branches and creating pull requests for review.
- Resolving merge conflicts.
- Documenting everything in `README.md` so it is reproducible later.

Key Takeaways

- Version control = a time machine for your project.
- GitHub = collaboration, accountability, and backup.
- Branches + pull requests = safe teamwork without stepping on each other's work.
- Pull before you push.
- Clear commit messages and good README files make your work reusable.

Questions?

Thank you!