git: the tool to manage them all

Suite 2026 - Queen's University

6 May 2026

R. Martin
Queen's University

Outline

1. What is git
2. The basics, working alone
3. The intention, working with others
4. Argh! merge conflict
5. What the fork
6. Random tidbits
7. The word is
8. Demo

`git`: A tool for managing software repositories

`git` is a tool for managing a large collection of files that multiple users can edit. It handles the following

- Manages a large codebase (directories and files) in a way that multiple users can edit at the same time.
- It keeps track of every single change.
- It allows one to undo specific changes.
- It does this efficiently without storing a copy of each user's version of a file.
- It maintains a backup of your files on the server, accessible from anywhere.

This is called "version control", and `git` is one tool among others. It's very useful for managing code in general, not just if you're working with others. You can use it for things other than code, I use it for open source textbooks! I have many projects that I work on by myself and keep them on github. It allows me to work on them from any computer, work on parallel updates, etc. Github is also often used in people's resumes to showcase their work.

Git uses a client-server model

Many clients connect to a central server, such as github.com.

git: a brief history

- Git was created by Linus Torvalds in 2005 for Linux kernel development after the project lost access to BitKeeper.
- Design goals: speed, data integrity, distributed work, and strong support for branching and merging.
- Junio Hamano became maintainer in 2005 and remains central to Git's development.
- GitHub, GitLab, Bitbucket, and similar platforms later built social collaboration around Git repositories.

Linus Torvalds created Git in 2005 while coordinating Linux kernel development.

git: a brief history

- Git was created by Linus Torvalds in 2005 for Linux kernel development after the project lost access to BitKeeper.
- Design goals: speed, data integrity, distributed work, and strong support for branching and merging.
- Junio Hamano became maintainer in 2005 and remains central to Git's development.
- GitHub, GitLab, Bitbucket, and similar platforms later built social collaboration around Git repositories.

Linus Torvalds cartoon version generated by AI running on my home computer!

Some example workflows

We'll look at two cases:

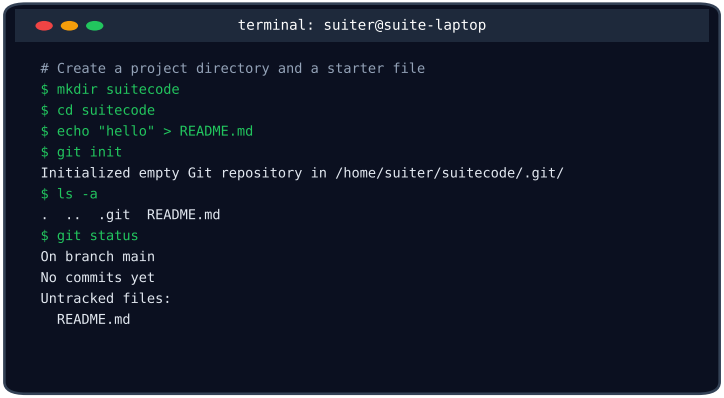
1. You set up a small repository for your own work
2. You work on shared codebase with other collaborators

A few notes:

- Most of you are new to Linux, terminal work, coding, etc. Don't worry about understanding how to do everything.
- Use this talk to know what is possible with git. Refer back to these slides when/if you actually need to use git.
- If you do any coding, you should use git! Even for your assignments in PHYS/ENPH 213!

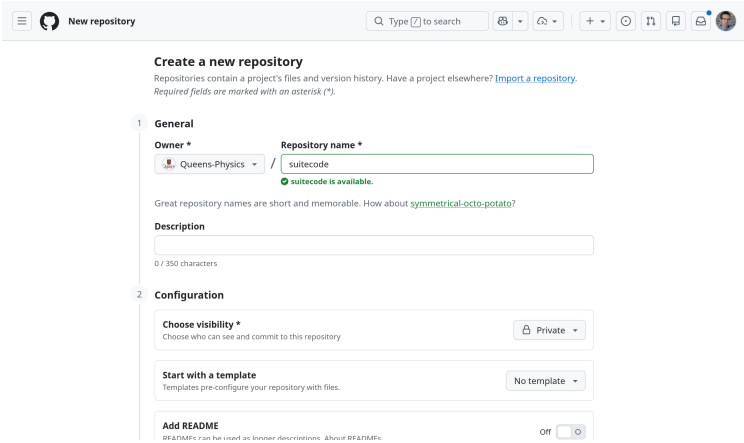
Case 1: Make a repository out of your code

- Create a directory on your computer with the code.
- (install git)
- Use `git` to initialize that directory as a “repository”.



Case 1: Make a repository out of your code

- Create a repository on a server, such as github.com in the cloud.



Case 1: Make a repository out of your code

- Link the local to the “remote”.
- Check the status of your files.
- “Commit” modified files to stage them.
- “Push” the commit into the repository.

Key take concepts

- “Repositories”: Local and remote copies of code.
- Main and feature “branches”.
- ”Commit” and “push”.

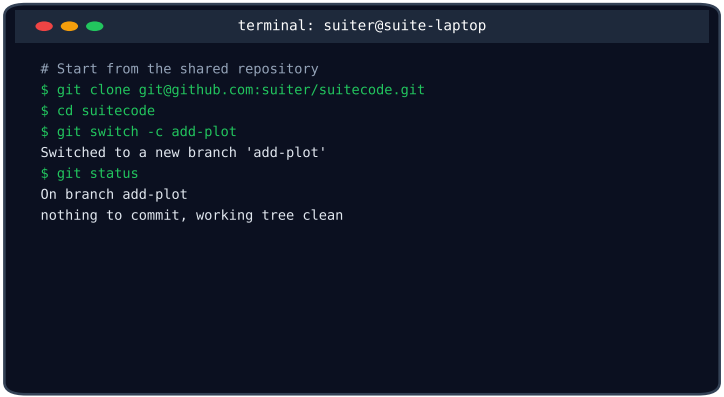
Case 2: Collaborating through branches and pull requests

A more complicated workflow:

- Your user is given access to a private repository.
- You “clone” the code from the repository onto your local machine.
- You create a “branch” to add a feature, and push that branch to the repository.
- You make local changes, commit them and then push them to the remote branch.
- Once feature is ready, you might have to run your code through specific tests (e.g. linter).
- You make a “pull request”. A “PR”.
- Admin reviews your work, gives you feedback.
- In the meantime, the main branch has changed and your code is no longer compatible.
- You fetch and merge the main branch into your feature branch.
- You manually resolve any conflicts.
- You make a pull request, it finally gets merged!
- Your first accepted PR, you get a pat on the back.

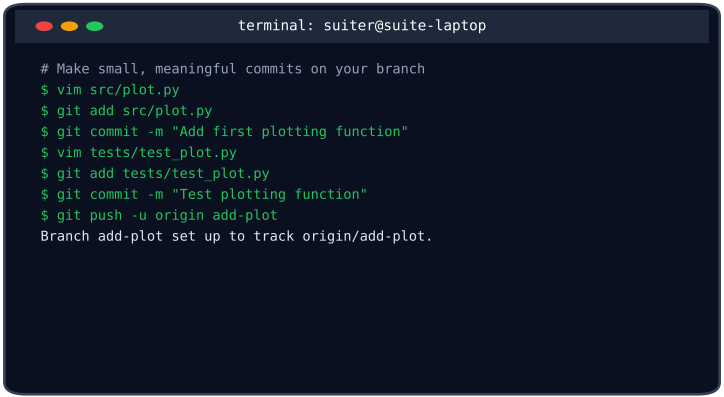
Step 1: Clone and branch

- `clone` downloads the repository and its history.
- Make a new branch for one coherent task.
- Keep branch names short and descriptive.



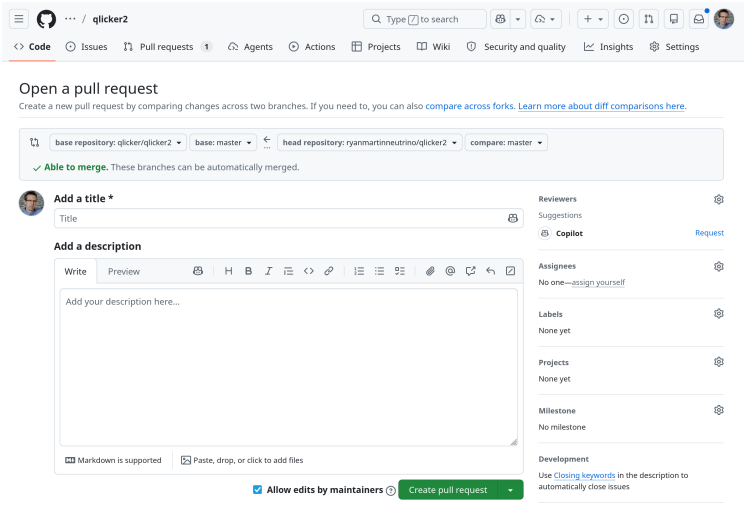
Step 2: Commit small pieces of work

- Stage the files that belong to this change with `git add`.
- Make one commit per logical idea.
- Push the branch to the remote so GitHub can see it.



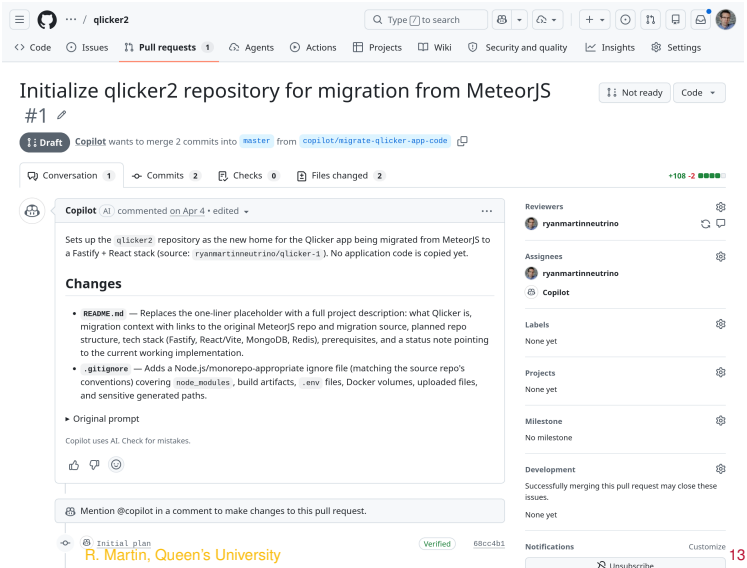
Step 3: Open a pull request

- Compare your branch against `main`.
- Explain what changed and why.
- Reviewers can comment before anything is merged.



Step 4: Respond to review

- Treat comments as part of the workflow, not as a failure.
- Make the requested edits on the same branch.
- Push a new commit; the pull request updates automatically.



Step 5: Implement feedback locally

- Switch back to your feature branch.
- Edit, test, commit, and push again.
- Update your open pull request.

Step 6: Bring your branch up to date

- While you were working, `main` may have changed.
- Fetch the newest remote data.
- Merge `origin/main` into your branch, resolve any conflicts, then push.

Step 7: Merge and clean up

- Once the pull request is approved and checks pass, merge it.
- Switch back to `main` and pull the merged result.
- Delete the old branch locally; prune old remote references.

terminal: suiter@suite-laptop

```
# After the PR is merged, clean up locally
$ git switch main
$ git pull origin main
$ git branch -d add-plot
Deleted branch add-plot (was d4e5f6a).
$ git fetch --prune
$ git status
On branch main
Your branch is up to date with origin/main.
```

A note on authentication

- You must prove to GitHub that you are allowed to push, and for private repositories, pull.
- For most terminal workflows, “SSH keys” are the cleanest option.
- Like most encryption schemes, two keys are generated, public and private.
- The private key stays on your machine, never share it; GitHub stores only the public key.
- Personal access tokens are another option for HTTPS workflows.

What is a merge conflict?

- A conflict happens when git cannot automatically combine two edits.
- git will not let you merge code if there's a conflict, you must resolve it first.
- In this case, you can view a file that merges both edits in a way that you can choose which to keep.
- Look up how to do it if it comes up...

Forks: a copy you own on the server

- **Branch**: best when you have write access to the shared repository.
- **Fork**: best when you do not have write access, or want an independent public copy.
- You still clone, branch, commit, push, and open a pull request.
- The pull request goes from your fork back to the upstream project.
- Or your fork can slowly drift from the main project and become it's own project.

But wait, there's more

- The README.md file acts an index for the repository, it supports markdown for styling.
- Add a .gitignore file - this allows file to exist in your local repository without getting synched (e.g. build files, data)
- On Windows (maybe Mac?), there is Github Desktop that is a graphical interface to all of this, quite easy to use.
- You can add automatic "actions" to repositories that trigger on a pull request:
  - Run a linter test, PR review to update code
  - Run a series of pre-determined tests on the code to see if you broke anything
  - Run a AI agent for a code review.
  - and many more!

Some git terminology

Term	Meaning
Repository	A project plus its hidden `.git` history.
Working tree	The files you see and edit in your directory.
Stage / index	The set of changes prepared for the next commit via `git add`.
Commit	A snapshot of tracked files plus author, time, and message.
Branch	A movable name pointing to a commit; used for isolated work.
Remote	Another copy of the repo, often on GitHub/GitLab.
Clone	A local copy of a remote repository.
Pull / fetch	Downloading new commits from a remote; `pull` also merges them.
Push	Sending local commits to a remote.
Fork	Your server-side copy of someone else's repository.
Pull request	A proposed change with discussion, review, and checks.
Merge conflict	A place where Git needs human help to combine changes.

Demo

Browsing a repository, using agents, looking at PRs.

Questions?