

Beginner's Guide to Git and GitHub

Your Name

Introduction

This guide introduces Git and GitHub for users with no prior experience. It walks you through key concepts, workflows, and best practices using real terminal commands, practical examples, and helpful diagrams where relevant.

1. Git: The Mental Model

Git works across **three main layers**:

- **Working Directory**: Your actual files.
- **Index (Staging Area)**: What will be included in the next commit.
- **HEAD (Repository)**: The last committed snapshot.

Key transitions:

“`bash git add # Moves changes → index (staging) git commit # Moves index → HEAD (repository) git reset # Moves HEAD pointer (undo commits) Detached HEAD` A detached HEAD occurs when you check out a specific commit instead of a branch.

`bash Copy code git log --oneline git checkout` You are now in a detached HEAD state.

To save your changes:

`bash Copy code git switch -c recovered-work` This creates a new branch from that commit.

2. Commit Management Amending a Commit Use `git commit --amend` when you:

Made a typo in the last commit message.

Forgot to include a file.

Want to slightly modify the last commit.

bash Copy code git add forgotten_file.txt git commit -amend Avoid using -amend on commits that have already been pushed to shared branches.

Viewing Commit History bash Copy code git log # Full history git log --oneline # Condensed view git reflog Shows every movement of HEAD — even forgotten by git log.

bash Copy code git reflog Use it to recover lost commits after reset or checkout.

3. File Management Deleting Files bash Copy code git rm filename.txt git commit -m "Remove filename.txt" Why not just delete manually? If you delete manually (in RStudio or file browser):

bash Copy code git status # Shows 'deleted: filename.txt' git add filename.txt git commit -m "Remove file" git rm does both steps for you — safer and cleaner.

Recovering Deleted Files If not committed:

bash Copy code git restore filename.txt If already committed:

bash Copy code git checkout HEAD~1 -- filename.txt Or use git reflog to find the commit where the file still existed.

Removing Files from Tracking (But Not Deleting) bash Copy code git rm --cached data.csv git commit -m "Stop tracking data.csv" Keeps the file on disk but removes it from Git.

Common Use Case You accidentally committed a large or private file.

Steps:

bash Copy code echo "data.csv" >> .gitignore git rm --cached data.csv git commit -m "Remove data.csv from tracking" 4. Cleaning and Resetting git clean Deletes untracked files (not in Git yet):

bash Copy code git clean -n # Dry run git clean -f # Permanent delete When to Use What Situation Use This Undo safely on shared branches git revert Clean up your last commit git commit -amend Rewind local commit history git reset Accidentally reset/lost work git reflog Delete untracked junk git clean

5. Squashing Commits Squashing combines multiple commits into a single one. Useful for:

Cleaning messy history

Grouping small fixes before pushing

bash Copy code git rebase -i HEAD~3 Then change:

bash Copy code pick abc123 Add feature pick def456 Fix typo pick ghi789 Update docs To:

bash Copy code pick abc123 Add feature squash def456 squash ghi789 One clean commit. Easier to read, review, and share.

When to Squash Before pushing a feature branch. Avoid after pushing (rewrites history!).

6. Git LFS (Large File Support) GitHub doesn't support files over 50MB.

Setup for Git LFS bash Copy code `git lfs install` `git lfs track "*.csv"` `git add .gitattributes` `git add data.csv` `git commit -m "Track large file"` Run `git lfs install` before committing.

If You Already Committed a Large File: bash Copy code `git reset --soft HEAD~1` `git lfs install` `git add .gitattributes` `file.csv` `git commit -m "Adding data files through Git LFS"` `git push origin main`

7. Git Tags

Tags are labels that point to a specific commit.

Creating Tags bash Copy code `git tag -a v1.0` `git tag -a v1.0` Listing Tags bash Copy code `git tag` `git tag -n` `git show v1.0` Tagging Old Commits bash Copy code `git log --oneline` `git tag -a v1.2` Sharing and Deleting Tags bash Copy code `git push origin v1.2` `git tag -d v1.2` `git push origin --delete v1.2` Why Tags? Branches represent variations.

Tags are point-in-time records (ideal for versions, milestones).

8. Adding Git to Existing Projects Option 1: Start from GitHub Create repo on GitHub

Clone locally

Move your files in

`git add .`, `git commit`, `git push`

Option 2 (Preferred): Start Locally bash Copy code `cd my_project_folder` `git init` `git add .` `git commit -m "Initial commit"` `git remote add origin git@github.com:username/repo.git` `git push -u origin main` Use `git remote -v` to verify.

9. Git Blame Shows who last modified each line of a file:

bash Copy code `git blame my_script.R` Includes:

Commit hash

Author

Timestamp

Line content

Investigating Further bash Copy code `git show` `git log -L :my_function:my_script.R` Useful for debugging or understanding code evolution.

10. Git Bisect Binary search through commit history to find bugs.

Step-by-Step: `bash Copy code git bisect start git bisect bad # current commit git bisect good # known good` Then mark each tested commit:

`bash Copy code git bisect good # or git bisect bad` Once found:

`bash Copy code git bisect reset` Automate with Script `bash Copy code git bisect run Rscript test.R` `test.R` should return:

0 → Good

1 → Bad

Ideal for large projects or subtle bugs.

11. Reproducible Presentations with Quarto So far, we've looked at guides and documents. But Quarto can also create slides.

Slide Structure Use horizontal rules to define slides:

yaml Copy code — title: “My Slides” format: revealjs —

Slide 1

Slide 2

Combine your Git knowledge and Quarto skills to make reproducible presentations.

Final Tips Use `git status` constantly

Keep your `.gitignore` up-to-date

Write clear commit messages

Never fear `git reflog`