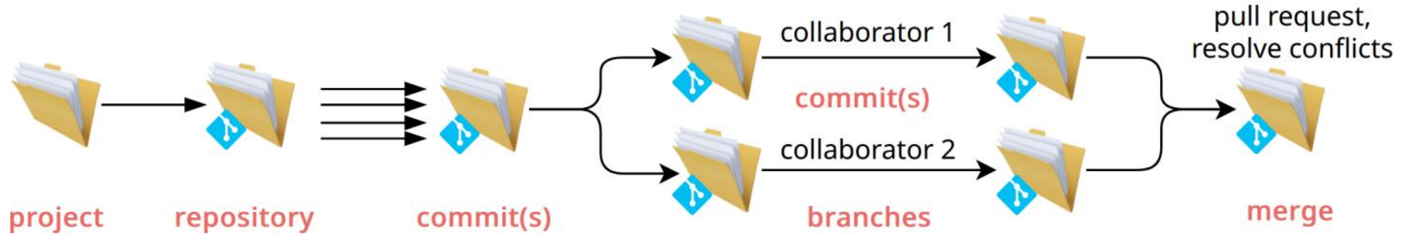
Git, GitHub, and why it matters

Git is a version-control system that tracks changes to files over time, so you can see what changed, when, and why. It lets you go back to earlier states, test new ideas safely, and collaborate with others. GitHub is a web-based platform that hosts Git repositories, enabling sharing, collaboration, and automation through features like pull requests and GitHub Actions. Version control ensures every analysis step is recorded, every change is reviewable, and anyone can rerun your workflow exactly as you did.

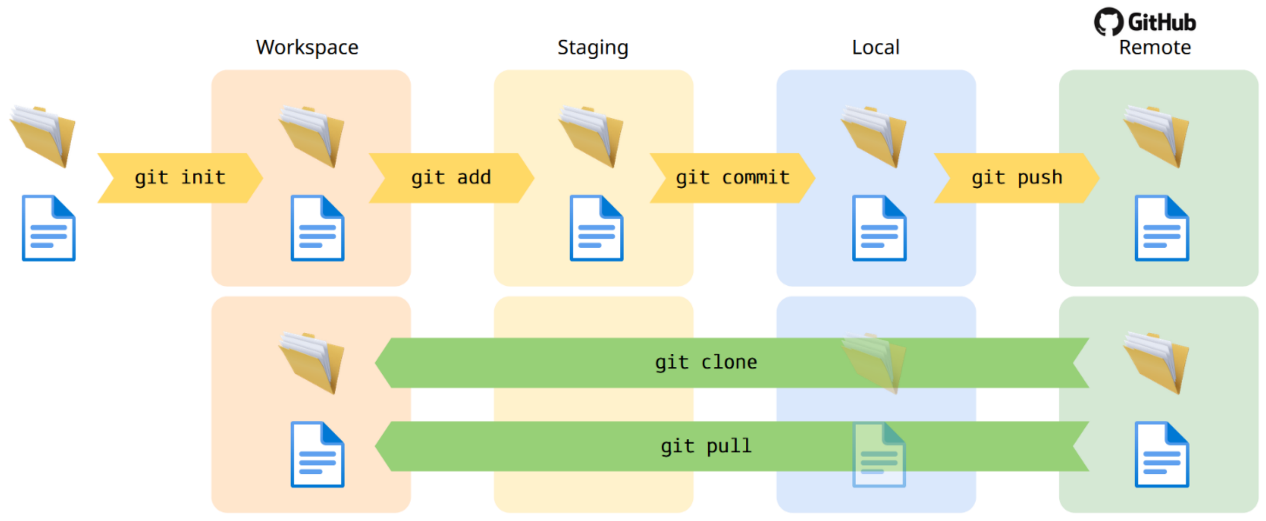
The mental model

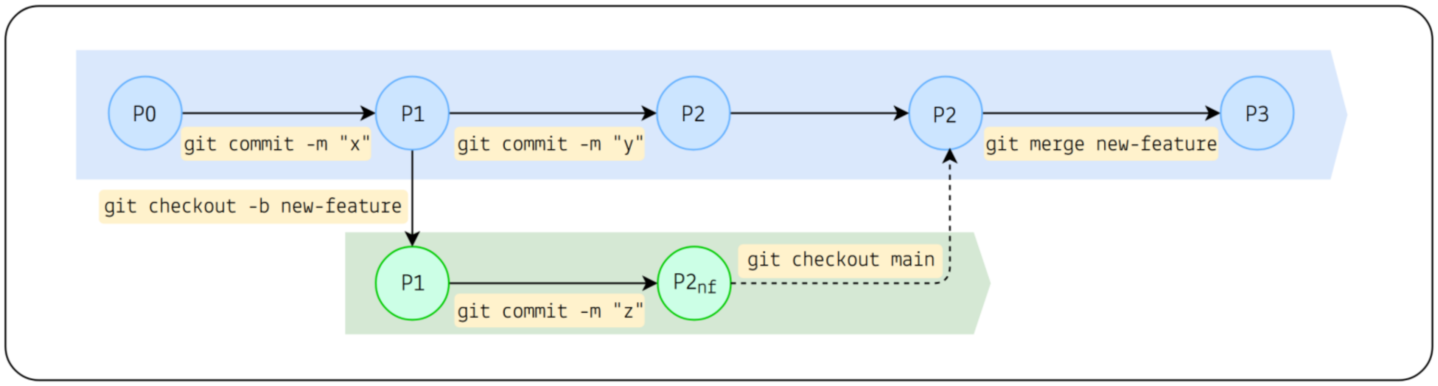
Your research project is managed as a Git repository, where its history is tracked as a series of commits. Branches allow for independent work on new ideas, and pull requests are used to review and merge those changes back into the main project.



Key terms and concepts

Repository (Repo): The project's main folder, containing all files and history.
Working Directory: The folder on your computer where you are actively editing your files.
Staging Area: A temporary holding area where the changes you want to include in your next commit.
Commit: A permanent snapshot of your staged changes, saved to the project's history.
Local Repository: The complete history of your project, stored on your own computer.
Remote Repository: The version of your project hosted on a server, like GitHub.com.
Branch: A parallel line of development within a repository.
Merge: The action of combining the changes from one branch into another.





Git Branching: This graphic illustrates the branching workflow. A new branch (new-feature) is created from the main branch to develop work in isolation. New commits are made on this branch without affecting the main project. Once complete, the feature branch is merged back into main, integrating the new changes into the primary project history.

Core Commands

Command	**Description**	**Syntax / Example**
Setting Up a Project		
`git config`	Sets your user name and email for commits.	`git config --global user.name "First Last"`
`git init`	Creates a new Git repository in a directory.	`git init`
`git clone`	Copies a remote repository to your computer.	`git clone <repository-url>`
Saving Changes Locally		
`git status`	Shows the current state of your files.	`git status`
`git add`	Adds file changes to the staging area.	`git add <file> or git add .`
`git commit`	Saves a snapshot of staged files to project history.	`git commit -m "Informative message"`
`git log`	Shows the project's commit history.	`git log`
Working with Branches		
`git branch`	Lists, creates, or deletes branches.	`git branch <new-branch-name>`
`git checkout`	Switches from one branch to another.	`git checkout <branch-name>`
`git merge`	Combines the history of a branch into your current one.	`git merge <branch-to-merge>`
Syncing with a Remote Repository		
`git clone`	Creates a local copy of a remote repository.	`git clone <repository-url>`
`git push`	Sends your committed changes to the remote repository.	`git push origin main`
`git pull`	Fetches changes from the remote repository and merges them.	`git pull origin main`

Tips

✓ **Git Do's**	✗ **Git Don'ts**
Add sensitive or untracked files to **.gitignore**.	commit large files (>100 MB); GitHub will reject them.
Write clear, meaningful commit messages.	commit sensitive info (e.g., API keys, passwords).
Commit often; small, logical commits make debugging easier.	commit binary files (.docx, .pdf, .xls) unless necessary.
Run **git pull** before **git push** to avoid merge conflicts.	use git push -f or git reset --hard unless you're absolutely sure; data loss is irreversible.
Use branches to organize features or experiments; merge or rebase thoughtfully.	bloat your repo; large repos slow down cloning and pushing.
Use **git stash** to temporarily save uncommitted work.	