Git Commands

This document provides a comprehensive list of Git commands, categorized by their use case and complexity, with clear explanations for each command.

Basic Commands (Beginner)

Initializing and Configuring Git:

- `git init`: Initializes a new Git repository in the current directory.

Example:

```

git init

```

This creates a `.git` folder to track your project's version history.

- `git config --global user.name "Your Name"`: Sets the username for Git globally.

- `git config --global user.email "you@example.com"`: Sets the email for Git globally.

Cloning a Repository:

- `git clone <repository\_url>`: Clones a remote repository to your local machine.

Example:

```

git clone https://github.com/username/repo.git

```

Checking the Status:

- `git status`: Displays the state of the working directory and staging area.

Example:

```

git status

```

Output includes untracked files, changes staged for commit, and branch information.

Adding Files to Staging Area:

- `git add <file>`: Adds a specific file to the staging area.

- `git add .`: Adds all files in the current directory to the staging area.

Committing Changes:

- `git commit -m "Commit message"`: Commits the changes in the staging area with a message.

Example:

```

git commit -m "Added login functionality"

```

Viewing Commit History:

- `git log`: Shows the commit history with details like author, date, and commit hash.

- `git log --oneline`: Displays a simplified, one-line log of commits.

Branching and Merging (Intermediate)

Managing Branches:

- `git branch`: Lists all branches in the repository.

- `git branch <branch\_name>`: Creates a new branch.

Example:

```

git branch feature/new-feature

```

- `git checkout <branch\_name>`: Switches to the specified branch.

- `git checkout -b <branch\_name>`: Creates a new branch and switches to it.

Merging Branches:

- `git merge <branch\_name>`: Merges the specified branch into the current branch.

Deleting Branches:

- `git branch -d <branch\_name>`: Deletes a branch that has already been merged.

- `git branch -D <branch\_name>`: Force-deletes a branch, even if it hasn’t been merged.

Working with Remote Repositories (Intermediate)

Adding and Fetching Remote Repositories:

- `git remote add <name> <url>`: Adds a remote repository.

Example:

```

git remote add origin https://github.com/username/repo.git

```

- `git fetch <remote>`: Downloads changes from the remote repository without applying them.

Pulling and Pushing Changes:

- `git pull <remote> <branch>`: Fetches and merges changes from the remote branch into your current branch.

- `git push <remote> <branch>`: Pushes local commits to the remote branch.

Example:

```

git push origin master

```

Conflict Resolution (Advanced)

Identifying Conflicts:

- `git status`: Highlights files with merge conflicts.

- `git diff`: Shows the differences between files.

Resolving Conflicts:

- Edit the file manually to resolve conflicts.

- `git add <file>`: Marks the conflict as resolved.

- `git commit`: Commits the resolution.

Aborting Operations:

- `git merge --abort`: Aborts a merge in progress.

- `git cherry-pick --abort`: Aborts a cherry-pick operation.

Rewriting History (Advanced)

Undoing Changes:

- `git reset <file>`: Unstages a file but keeps changes in the working directory.

- `git reset --hard`: Resets the working directory to the last commit.

Amending Commits:

- `git commit --amend`: Edits the last commit message or adds staged changes to the last commit.

Removing Files:

- `git rm <file>`: Removes a file from the working directory and stages the deletion.

Collaboration (Intermediate)

Stashing Changes:

- `git stash`: Temporarily stores uncommitted changes.

- `git stash pop`: Applies the stashed changes back to the working directory.

Pull Requests:

- Create a branch, push changes, and open a pull request in GitHub/GitLab.

Debugging and Diagnostics (Advanced)

Inspecting Changes:

- `git diff`: Shows changes between commits, branches, or the working directory and staging area.

- `git blame <file>`: Shows who modified each line of a file.

Viewing Remotes:

- `git remote -v`: Lists all configured remote repositories.

Advanced Commands (Expert)

Rewriting History:

- `git rebase`: Re-applies commits on top of another base branch.

Example:

```

git rebase main

```

- `git filter-branch`: Used for rewriting commit history (e.g., removing sensitive data).

Cherry-Picking:

- `git cherry-pick <commit\_hash>`: Applies a specific commit to the current branch.

Tags:

- `git tag <tag\_name>`: Creates a lightweight tag at the current commit.

- `git push origin <tag\_name>`: Pushes tags to the remote repository.

Cleaning Up (Intermediate)

- `git clean -f`: Removes untracked files from the working directory.

- `git gc`: Cleans up unnecessary files and optimizes the local repository.

FAQs

What is the difference between `git pull` and `git fetch`?

- `git fetch` downloads changes from the remote repository without merging them, whereas `git pull` fetches and merges the changes.

How do I undo a commit without losing changes?

- Use `git reset --soft HEAD~1`.

How do I resolve merge conflicts?

- Manually edit the conflicting files, stage them with `git add`, and commit the resolution.

---