#### **Basic Setup**

# Git-GitHub Cheatsheet

- git config --global user.name "Your Name"
- # Set your Git username.
- git config --global user.email "your.email@example.com" # Set your Git email.
- git config --list
- # List all Git configurations.





# **Initializing and Cloning**

- git init
- # Initialize a new Git repository in your project.
- git clone <repo-url>
- # Clone an existing repository.

### **Working with Changes**

- git add <file>
- # Stage a specific file for commit.
- git add.
- # Stage all changes in the current directory.
- git commit -m "Commit message"
- # Commit changes with a message.
- git commit -am "Message"
- # Add and commit tracked files in one step.
- git commit --amend
- # Edit the last commit message or add changes to it.

## **Handling Merge Conflicts**

- git diff
- # Compare working directory changes.
- git diff <branch1> <branch2>
- # Compare two branches.
- # Resolve conflicts: Open the files, fix conflicts, then add and commit.

#### **Status & Logs**

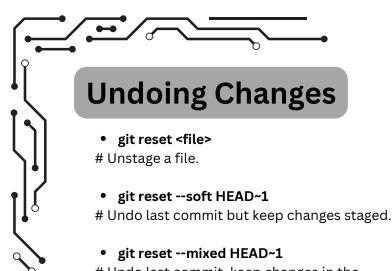
- git status
- # Show the current status of changes in the working directory.
- git log
- # View commit history.
- git log --oneline
- # Show concise commit history.

# **Branching & Merging**

- git branch <br/> branch-name>
- # Create a new branch.
- git checkout <br/>branch-name>
- # Switch to a specific branch.
- git checkout -b <br/>branch-name>
- # Create and switch to a new branch.
- git merge <br/>
  <br/>
  branch-name>
- # Merge specified branch into the current branch.
- git rebase <br/>
  <br/>
  branch-name>
- # Reapply commits on top of another base.
- git rebase -i HEAD~<n>
- # Interactive rebase to edit commit history, rearrange commits, modify commit messages, or squash the last n commits
- git branch -d <branch-name>
- # Delete a local branch (use -D to force delete).

Spoorti Shetty

0 የ



# Undo last commit, keep changes in the working directory (unstaged).

git reset --hard HEAD~1

# Completely remove the last commit.

• git revert < commit-id>

# Create a new commit that undoes the specified commit.

### **Stashing Changes**

git stash

# Temporarily save changes.

• git stash list

# View stashed changes.

• git stash pop

# Reapply stashed changes and remove them from the stash list.

git stash apply

# Reapply stashed changes without removing them.

• git stash clear

# Remove all stashed entries.

# **Collaborating & Pull Requests**

• git branch -a

# List all branches, including remote.

• git push origin :<br/>branch-name>

# Delete a remote branch.

# Creating a Pull Request: Go to your GitHub repository, select your branch, and click "New Pull Request."

#### **Remote Repositories**

• git remote add origin <url>

# Link your local repository to a remote one.

• git remote -v

# List the remote repository URLs.

• git remote set-url origin <new-url>

# Update the remote URL for the repository.

• git remote rename <old-name> <new-name>

# Rename a remote.

• git push -u origin <br/> sranch-name>

# Push changes to the remote repository.

• git pull origin <branch-name>

# Pull changes from the remote branch.

git fetch

# Download updates from the remote without merging.

• git fetch <remote>

# Fetch updates from a specific remote.

#### **Advanced Operations**

• git cherry-pick <commit-id>

# Apply a specific commit from another branch.

git cherry-pick <start-commit-id>^..<end-commit-id>

# Cherry-pick a range of commits.

git tag <tag-name>

# Add a tag to a commit.

git tag -d <tag-name>

# Remove a local tag.

git reflog

# View history of all changes (even uncommitted).

git reflog show <branch-name>

# Show reflog for a specific branch.

git show <commit-id>

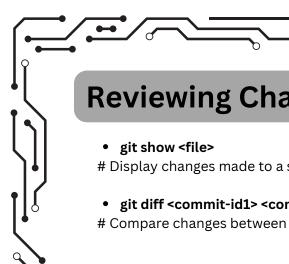
# Show detailed info for a specific commit.

git bisect start

# Start bisecting to locate a bug.

Spoorti Shetty





# **Reviewing Changes**

- # Display changes made to a specific file.
- git diff <commit-id1> <commit-id2>
- # Compare changes between two commits.

#### **Help Command**

- git help <command>
- # Get detailed help for a specific command.

# **GitHub Commands** (Optional with GitHub CLI)

- gh repo create
- # Create a new GitHub repo from the command line.
- gh repo clone <repo-url>
- # Clone a GitHub repository.
- · gh pr create
- # Create a pull request from the command line.
- gh pr list
- # List open pull requests in the repository.
- gh issue create
- # Create a GitHub issue from the command line.

# GitHub API (using curl)

• curl -H "Authorization: token YOUR\_TOKEN" https://api.github.com/repos/USERNAME/REPO\_NAME/issues # List issues in a repository.

# Submodules & Worktrees

- git submodule add <repo-url> <path>
- # Add a submodule.
- git submodule init
- # Initialize submodules.
- git submodule update
- # Update submodules.
- git worktree add <path> <branch>
- # Create a new working tree for a branch.

# Cleaning Up

- git clean -f
- # Remove untracked files.
- git clean -fd
- # Remove untracked files and directories.
- git gc --prune=now
- # Clean up unnecessary files and optimize the local repository.

# **Repository Management** and Information

- git shortlog -s -n
- # Summarize commits by author.
- git describe --tags
- # Get a readable name for a commit.
- git blame <file>
- # Show who last modified each line of a file.
- git grep "search-term"
- # Search for a term in the repository.
- git revert <commit-id1>..<commit-id2>
- # Revert a range of commits.
- git archive --format=zip HEAD -o latest.zip
- # Archive the latest commit as a ZIP file.
- git fsck
- # Check the object database for integrity.

Spoorti Shett

#### **Best Practices and Common Workflows**

- **Commit Often:** Make frequent commits with descriptive messages to maintain a clear project history.
- Branch for Features: Create a new branch for each feature or bug fix to keep changes organized and separate from the main codebase.
- **Use Meaningful Commit Messages:** Write clear and concise commit messages that explain the purpose of the changes.
- Pull Regularly: Regularly pull changes from the remote repository to stay updated with the latest changes and minimize merge conflicts.
- Resolve Conflicts Promptly: Address merge conflicts as soon as they arise to avoid complicating the integration process.
- **Review Pull Requests Thoroughly:** Ensure thorough review of pull requests to maintain code quality and facilitate knowledge sharing.
- Tag Releases: Use tags to mark important milestones or releases in the project for easy reference in the future.
- **Keep Your Branches Clean:** Delete branches that are no longer needed after merging them into the main branch to keep the repository organized.
- Use Git Hooks for Automation: Utilize Git hooks to automate tasks, like running tests before committing (pre-commit) or checking commit message formats. Hooks can help ensure code quality and consistency.
- **Squash Commits Before Merging:** Squash commits to combine related work into a single commit before merging, especially for feature branches. This keeps the project history clean and manageable.
- **Avoid Large Commits:** Try to keep commits small and focused on a single change or fix. This makes it easier to understand the history and isolate issues if something goes wrong.
- Create Descriptive Branch Names: Use branch naming conventions that describe the purpose, such as feature/login-form or fix/user-authentication-bug. This improves readability and collaboration.
- Keep the Main Branch Deployable: Always ensure that the main or production branch is stable and deployable. This allows the project to be released or updated at any time.