Git

General

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

More Git related links:

- Website
- Download Cheat Sheet as PDF | PNG

Create a Repository

`git init` - Create a new repository

`git clone <repository>` - Clone an existing repository

`git remote add <remote> <url>` - Add a new remote (usually the default value for `<remote>` is `origin`)

Branches

`git checkout -b <branch>` - Create and switch to a new branch

`git checkout <branch>` - Switch to an existing branch

`git branch` - List all branches

`git branch -m <old-branch> <new-branch>` - Rename a branch

`git branch -d <branch>` - Delete a branch

`git branch -D <branch>` - Force delete a branch

`git push <remote> --delete <branch>` - Delete a remote branch

Commits

`git add <file>` - Stage file for commit

`git add <file> -p` - Interactively stage hunks for commit

`git rm <file>` - Remove file from working directory and stage for commit

`git rm --cached <file>` - Unstage file but keep it in working directory

`git mv <old-file> <new-file>` - Move or rename a file and stage for commit

`git restore <file>` - Discard changes in working directory and restore file to last committed state

`git restore --staged <file>` - Unstage file but keep changes in working directory

`git commit -m <message>` - Commit staged changes with a message

`git commit --amend --no-edit` - Amend the last commit with staged changes but keep the existing commit message

Synchronize

`git push` - Push local changes to the origin

`git push -u <remote> <branch>` - Push local changes to the origin that were never pushed before

`git fetch` - Fetch changes from the origin without merging

`git pull` - Fetch and merge changes from the origin

`git pull --rebase` - Fetch and rebase changes from the origin

Observe

`git status` - List new or modified files to be committed

`git diff` - Show unstaged changes

`git diff --cached` - Show staged changes

`git diff HEAD` - Show staged and unstaged changes

`git show <commit>` - Show diff between a commit and its parent

`git blame <file>` - Show who last changed each line of a file

`git log <file>` - Show commit history for a file

Stashing

`git stash -u` - Stash changes, including untracked files

`git stash -p` - Stash changes partially by interactively selecting hunks

`git stash list` - List all stashes

`git stash apply "stash@{n}"` - Apply the stash at index `n`

`git stash pop "stash@{n}"` - Apply the stash at index `n` and remove it from the stash list

`git stash show "stash@{n}"` - Show the diff summary of the stash at index `n`

`git stash drop "stash@{n}"` - Drop the stash at index `n`

Revert

`git revert <commit>` - Revert a commit by creating a new commit

`git reset <commit>` - Move HEAD to `<commit>` and discard all changes after it

`git reset --soft <commit>` - Move HEAD to `<commit>` but keep changes staged

`git reset --hard` - Delete all staged and unstaged changes

Tags

`git tag <tag>` - Add tag to current commit

`git push origin <tag>` - Push tag to remote repository

`git tag --delete <tag>` - Delete local tag

`git push --delete origin <tag>` - Delete remote tag

Combine Branches

`git rebase <branch>` - Rebase current branch onto `<branch>`

`git merge <branch>` - Merge `<branch>` into current branch

`git merge --squash <branch>` - Merge `<branch>` into current branch but combine all commits into a single commit

`git cherry-pick <commit>` - Apply the changes introduced by `<commit>` onto the current branch

Conventional Commits

The Conventional Commits specification is a lightweight convention on top of commit messages. It provides an easy set of rules for creating an explicit commit history; which makes it easier to write automated tools on top of. The commit message should be structured as follows:

```
<type>(<scope>): <description>

[optional body]

[optional footer(s)]
```

The following `<types>` are commonly used in the Conventional Commits specification:

`feat` - New feature

`fix` - Bug fix

`docs` - Documentation only

`style` - Code style (formatting, no logic change)

`refactor` - Code change that neither fixes a bug nor adds a feature

`perf` - Performance improvement

`test` - Adding or correcting tests

`build` - Build system or external dependencies

`ci` - CI configuration

`chore` - Maintenance tasks

`revert` - Reverts a previous commit

Tip: Add a `!` after the `<type>` / `<scope>` to indicate a breaking change.