■ Git Commands Cheat Sheet ■

Basic Git Operations

`git init`	Initialize a new Git repository
`git clone <url>`	Clone remote repository to local machine
`git clone <url> <directory>`	Clone into specific directory
`git status`	Show working tree status
`git add <file>`	Add file to staging area
`git add .`	Add all files to staging area
`git add -A`	Add all files (including deleted)
`git commit -m '<message>'`	Commit staged changes with message
`git commit -am '<message>'`	Add all tracked files and commit
`git commit --amend`	Modify last commit
`git log`	Show commit history
`git log --oneline`	Show condensed commit history
`git log --graph`	Show commit history as graph

Branching & Merging

`git branch`	List local branches
`git branch -a`	List all branches (local + remote)
`git branch <branch-name>`	Create new branch
`git branch -d <branch-name>`	Delete merged branch
`git branch -D <branch-name>`	Force delete branch
`git checkout <branch-name>`	Switch to branch
`git checkout -b <branch-name>`	Create and switch to new branch
`git switch <branch-name>`	Switch to branch (Git 2.23+)
`git switch -c <branch-name>`	Create and switch to new branch
`git merge <branch-name>`	Merge branch into current branch
`git merge --no-ff <branch-name>`	Merge with merge commit

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

Remote Repository Operations

`git remote`	List remote repositories
`git remote -v`	List remotes with URLs
`git remote add <name> <url>`	Add remote repository
`git remote remove <name>`	Remove remote repository
`git fetch`	Download changes from remote
`git fetch <remote>`	Fetch from specific remote
`git pull`	Fetch and merge from remote
`git pull --rebase`	Fetch and rebase instead of merge
`git push`	Push changes to remote
`git push <remote> <branch-name>`	Push branch to specific remote
`git push -u origin <branch-name>`	Push and set upstream branch
`git push --force`	■■ Force push (dangerous)
`git push --force-with-lease`	Safer force push

Inspection & Comparison

`git diff`	Show unstaged changes
`git diff --staged`	Show staged changes
`git diff <branch-name>`	Compare with another branch
`git diff HEAD~1`	Compare with previous commit
`git show <commit-id>`	Show specific commit details
`git blame <file>`	Show who changed each line
`git log --follow <file>`	Show file history across renames
`git log --grep='<pattern>'`	Search commits by message
`git log --author='<name>'`	Filter commits by author
`git reflog`	Show reference log (recovery tool)

Undoing Changes

`git checkout -- <file>`	Discard changes in working directory
`git restore <file>`	Discard changes (Git 2.23+)
`git reset <file>`	Unstage file (keep changes in working directory)
`git reset --soft HEAD~1`	Resets to one commit before HEAD (the immediate previous commit). HEAD~1 means 'parent of HEAD'
`git reset --soft <commit-id>`	Resets to a specific commit hash you provide. You can reset to any commit in history
`git reset --mixed HEAD~1`	Resets to one commit before HEAD, unstaging changes but keeping them in working directory
`git reset --mixed <commit-id>`	Resets to a specific commit hash, unstaging changes but keeping them in working directory
`git reset --hard HEAD~1`	■■ Resets to one commit before HEAD and permanently deletes all uncommitted changes
`git reset --hard <commit-id>`	■■ Resets to a specific commit hash and permanently deletes all uncommitted changes
`git revert <commit-id>`	Create commit that undoes specified commit
`git clean -n`	Preview untracked files to delete
`git clean -f`	Delete untracked files
`git clean -fd`	■■ Delete untracked files and directories

Stashing

`git stash`	Stash current changes
`git stash save '<message>'`	Stash with message
`git stash list`	List all stashes
`git stash show`	Show latest stash changes
`git stash show -p`	Show latest stash as patch
`git stash apply`	Apply latest stash
`git stash apply stash@{<index>}`	Apply specific stash
`git stash pop`	Apply and remove latest stash
`git stash drop`	Delete latest stash
`git stash clear`	Delete all stashes

Advanced Commands

`git tag`	List tags
`git tag <tag-name>`	Create lightweight tag
`git tag -a <tag-name> -m '<message>'`	Create annotated tag
`git tag -d <tag-name>`	Delete tag
`git cherry-pick <commit-id>`	Apply specific commit to current branch
`git bisect start`	Start binary search for bug
`git submodule add <url>`	Add Git submodule
`git submodule update --init`	Initialize and update submodules
`git archive --format=zip HEAD`	Create archive of current HEAD
`git gc`	Cleanup unnecessary files
`git fsck`	Check repository integrity

Configuration

`git config --global user.name '<name>'`	Set global username
`git config --global user.email '<email>'`	Set global email
`git config --list`	Show all configuration
`git config user.name`	Show username
`git config --global init.defaultBranch main`	Set default branch name
`git config --global core.editor <editor>`	Set default editor
`git config --global alias.<alias> <command>`	Create alias for command
`git config --global core.autocrlf true`	Auto convert line endings (Windows)
`git config --global core.autocrlf input`	Auto convert line endings (Mac/Linux)
`git config --global pull.rebase false`	Default merge behavior for pull

■ Common Git Workflows:

Feature Branch Workflow:
1. git checkout -b feature/<feature-name>
2. Make changes and commits
3. git push -u origin feature/<feature-name>
4. Create pull request
5. git checkout main && git pull
6. git branch -d feature/<feature-name>

Hotfix Workflow:
1. git checkout -b hotfix/<fix-name>
2. Make fix and commit
3. git checkout main && git merge hotfix/<fix-name>
4. git checkout develop && git merge hotfix/<fix-name>
5. git branch -d hotfix/<fix-name>

Release Workflow:
1. git checkout -b release/<version>
2. Bump version numbers, final testing
3. git checkout main && git merge release/<version>
4. git tag -a <version> -m "Version <version>"
5. git checkout develop && git merge release/<version>

■ Git Tips & Best Practices:

• Write clear, descriptive commit messages
• Commit early and often with logical chunks
• Always review changes before committing (git diff --staged)
• Use .gitignore to exclude unnecessary files
• Never commit sensitive information (passwords, keys)
• Use branches for features, experiments, and fixes
• Rebase feature branches before merging (when safe)
• Use 'git stash' when switching contexts quickly
• Regularly fetch updates from remote repositories
• Learn to read and understand git log --graph
• Use git reflog as a safety net for recovery
• Set up GPG signing for verified commits

■■ Dangerous Commands (use with caution):
• git reset --hard: Permanently loses uncommitted changes
• git push --force: Can overwrite others' work
• git rebase: Changes commit history (don't rebase shared branches)
• git clean -fd: Permanently deletes untracked files