Complete Git & GitHub Command Reference

Table of Contents

Git Basics

What is Git?

Git is a distributed version control system that tracks changes in files and coordinates work between multiple people. GitHub is a web-based hosting service for Git repositories.

Key Concepts

- **Repository (Repo)**: A project folder tracked by Git
- **Commit**: A snapshot of your project at a specific point in time
- **Branch**: A parallel version of your repository
- **Remote**: A version of your repository hosted on a server (like GitHub)
- **Clone**: Creating a local copy of a remote repository
- **Fork**: Creating your own copy of someone else's repository on GitHub

---

Initial Setup

Configure Git (First Time Setup)

bash

```bash
# Set your name and email
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# Check your configuration
git config --list

# Set default branch name
git config --global init.defaultBranch main

# Set default editor
git config --global core.editor "code --wait"  # For VS Code
git config --global core.editor "nano"         # For Nano
```

SSH Key Setup for GitHub

bash

```bash
# Generate SSH key
ssh-keygen -t ed25519 -C "your.email@example.com"

# Start SSH agent
eval "$(ssh-agent -s)"

# Add SSH key to agent
ssh-add ~/.ssh/id_ed25519

# Copy public key to clipboard (Linux/Mac)
cat ~/.ssh/id_ed25519.pub

# Test SSH connection to GitHub
ssh -T git@github.com
```

Repository Operations

Initialize a Repository

bash

```
# Initialize a new Git repository
git init

# Initialize with specific branch name
git init --initial-branch=main
git init -b main
```

Clone a Repository

bash

```
# Clone a repository
git clone https://github.com/username/repository.git

# Clone with SSH
git clone git@github.com:username/repository.git

# Clone into specific directory
git clone https://github.com/username/repository.git my-project

# Clone specific branch
git clone -b branch-name https://github.com/username/repository.git
```

File Operations

Check Status

bash

```
# Check repository status
git status

# Check status in short format
git status -s
```

Adding Files

bash

```bash
# Add specific file
git add filename.txt

# Add all files in current directory
git add .

# Add all files in repository
git add -A

# Add files interactively
git add -i

# Add part of a file
git add -p filename.txt
```

Committing Changes

bash

```bash
# Commit with message
git commit -m "Your commit message"

# Commit all tracked files (skip git add)
git commit -am "Your commit message"

# Commit with detailed message
git commit -m "Short summary" -m "Detailed description"

# Amend the last commit
git commit --amend -m "New commit message"
```

Removing Files

bash

```bash
# Remove file from Git and filesystem
git rm filename.txt

# Remove file from Git but keep in filesystem
git rm --cached filename.txt

# Remove directory
git rm -r directory-name
```

Moving/Renaming Files

bash

```
# Rename/move file
git mv old-filename.txt new-filename.txt
```

Branching and Merging

Branch Operations

bash

```
# List all branches
git branch

# List all branches (including remote)
git branch -a

# Create new branch
git branch new-branch-name

# Create and switch to new branch
git checkout -b new-branch-name
git switch -c new-branch-name  # Newer syntax

# Switch to existing branch
git checkout branch-name
git switch branch-name  # Newer syntax

# Delete branch
git branch -d branch-name

# Force delete branch
git branch -D branch-name

# Rename current branch
git branch -m new-branch-name
```

Merging

bash

```
# Merge branch into current branch
git merge branch-name

# Merge with no fast-forward
git merge --no-ff branch-name

# Abort merge
git merge --abort
```

Rebasing

bash

```
# Rebase current branch onto another branch
git rebase branch-name

# Interactive rebase
git rebase -i HEAD~3  # Last 3 commits

# Continue rebase after resolving conflicts
git rebase --continue

# Abort rebase
git rebase --abort
```

Remote Operations

Managing Remotes

bash

```
# List remotes
git remote
git remote -v

# Add remote
git remote add origin https://github.com/username/repository.git

# Remove remote
git remote remove origin

# Rename remote
git remote rename origin upstream
```

Fetching and Pulling

```bash
# Fetch changes from remote
git fetch origin

# Fetch all remotes
git fetch --all

# Pull changes (fetch + merge)
git pull origin main

# Pull with rebase
git pull --rebase origin main
```

Pushing

```bash
# Push to remote
git push origin main

# Push new branch to remote
git push -u origin new-branch-name

# Push all branches
git push --all origin

# Push tags
git push --tags origin

# Force push (use with caution)
git push --force origin main
```

GitHub Specific Commands

GitHub CLI (gh)

bash

```bash
# Install GitHub CLI first, then authenticate
gh auth login

# Create repository on GitHub
gh repo create repository-name

# Clone your repository
gh repo clone username/repository-name

# Fork a repository
gh repo fork username/repository-name

# Create pull request
gh pr create --title "Title" --body "Description"

# List pull requests
gh pr list

# View pull request
gh pr view 123

# Merge pull request
gh pr merge 123

# Create issue
gh issue create --title "Issue title" --body "Issue description"

# List issues
gh issue list
```

Viewing History and Information

Log Commands

bash

```bash
# View commit history
git log

# One line per commit
git log --oneline

# Show last n commits
git log -n 5

# Show commits with file changes
git log --stat

# Show commits with actual changes
git log -p

# Show commits in graph format
git log --graph --oneline --all

# Show commits by author
git log --author="Author Name"

# Show commits in date range
git log --since="2023-01-01" --until="2023-12-31"
```

Diff Commands

bash

```bash
# Show unstaged changes
git diff

# Show staged changes
git diff --cached

# Show changes between commits
git diff commit1 commit2

# Show changes in specific file
git diff filename.txt

# Show changes between branches
git diff branch1 branch2
```

Show Commands

bash

```
# Show details of specific commit
git show commit-hash

# Show files in commit
git show --name-only commit-hash

# Show specific file from commit
git show commit-hash:filename.txt
```

Undoing Changes

Unstaging Changes

bash

```
# Unstage file
git reset HEAD filename.txt

# Unstage all files
git reset HEAD
```

Discarding Changes

bash

```
# Discard changes in working directory
git checkout -- filename.txt

# Discard all changes in working directory
git checkout -- .

# Clean untracked files
git clean -f

# Clean untracked files and directories
git clean -fd
```

Reset Commands

bash

```
# Soft reset (keep changes in staging)
git reset --soft HEAD~1

# Mixed reset (keep changes in working directory)
git reset --mixed HEAD~1
git reset HEAD~1  # Default is mixed

# Hard reset (discard all changes)
git reset --hard HEAD~1
```

Revert Commands

bash

```
# Revert a commit (creates new commit)
git revert commit-hash

# Revert without committing
git revert --no-commit commit-hash
```

Advanced Commands

Stashing

bash

```
# Stash current changes
git stash

# Stash with message
git stash save "Work in progress"

# List stashes
git stash list

# Apply most recent stash
git stash apply

# Apply specific stash
git stash apply stash@{2}

# Apply and remove stash
git stash pop

# Drop stash
git stash drop stash@{2}

# Clear all stashes
git stash clear
```

Tagging

bash

```bash
# Create lightweight tag
git tag v1.0.0

# Create annotated tag
git tag -a v1.0.0 -m "Version 1.0.0"

# List tags
git tag

# Push tags to remote
git push origin v1.0.0
git push origin --tags

# Delete tag
git tag -d v1.0.0

# Delete remote tag
git push origin --delete v1.0.0
```

Cherry-picking

bash

```bash
# Apply specific commit to current branch
git cherry-pick commit-hash

# Cherry-pick without committing
git cherry-pick --no-commit commit-hash
```

Bisect (Finding Bugs)

bash

```bash
# Start bisect
git bisect start

# Mark current commit as bad
git bisect bad

# Mark known good commit
git bisect good commit-hash

# Continue bisecting
git bisect good  # Current commit is good
git bisect bad   # Current commit is bad

# End bisect
git bisect reset
```

Common Workflows

Basic Workflow

bash

```bash
# 1. Clone repository
git clone https://github.com/username/repository.git
cd repository

# 2. Create feature branch
git checkout -b feature/new-feature

# 3. Make changes and commit
git add .
git commit -m "Add new feature"

# 4. Push branch
git push -u origin feature/new-feature

# 5. Create pull request on GitHub
# 6. After PR is merged, update main branch
git checkout main
git pull origin main
git branch -d feature/new-feature
```

Collaboration Workflow

bash

```bash
# 1. Fork repository on GitHub
# 2. Clone your fork
git clone https://github.com/yourusername/repository.git
cd repository

# 3. Add upstream remote
git remote add upstream https://github.com/original-owner/repository.git

# 4. Create feature branch
git checkout -b feature/new-feature

# 5. Make changes and commit
git add .
git commit -m "Add new feature"

# 6. Push to your fork
git push origin feature/new-feature

# 7. Create pull request
# 8. Keep fork updated
git fetch upstream
git checkout main
git merge upstream/main
git push origin main
```

Hotfix Workflow

bash

```bash
# 1. Create hotfix branch from main
git checkout main
git checkout -b hotfix/urgent-fix

# 2. Make fix and commit
git add .
git commit -m "Fix urgent issue"

# 3. Push and create PR
git push -u origin hotfix/urgent-fix

# 4. After merge, update main
git checkout main
git pull origin main
git branch -d hotfix/urgent-fix
```

Best Practices

Commit Messages

- Use present tense ("Add feature" not "Added feature")
- Keep first line under 50 characters
- Use imperative mood ("Fix bug" not "Fixed bug")
- Reference issues when applicable ("Fix #123")

Branching Strategy

- Use descriptive branch names (feature/user-authentication)
- Keep branches focused on single features
- Delete merged branches
- Use prefixes: feature/, bugfix/, hotfix/

Repository Management

- Use .gitignore to exclude unnecessary files
- Keep commits small and focused
- Review changes before committing
- Use pull requests for code review
- Write meaningful commit messages

Security

- Never commit sensitive information (passwords, API keys)
- Use environment variables for secrets
- Review public repositories for sensitive data
- Use SSH keys for authentication

Useful .gitignore Examples

General

```
# OS generated files
.DS_Store
Thumbs.db

# IDE files
.vscode/
.idea/
*.swp

# Dependencies
node_modules/
vendor/

# Build outputs
dist/
build/
*.exe
*.dll

# Environment files
.env
.env.local
```

Language Specific

```
# Python
__pycache__/
*.pyc
venv/

# JavaScript/Node.js
node_modules/
npm-debug.log
yarn-error.log

# Java
*.class
target/

# C++
*.o
*.exe
```

This reference covers the essential Git and GitHub commands you'll need for version control and collaboration. Keep this handy as you work with repositories!