

# Git and GitHub Cheat Sheet

by Codex

## 1. ⚙ Initial Setup & Configuration

### Configure Your Identity

```
# Set your name for commits  
git config --global user.name "Your Name"  
  
# Set your email for commits  
git config --global user.email  
"you@example.com"
```

### View Your Configuration

```
# See all configuration settings  
git config --list  
  
# See specific setting  
git config user.name  
git config user.email
```

### Additional Setup

```
# Set default text editor  
git config --global core.editor "vim"  
  
# Enable colored output  
git config --global color.ui auto  
  
# Set default branch name  
git config --global init.defaultBranch main
```

## 2. ✚ Creating Repositories

### Initialize New Repository

```
# Create a new Git repository in current  
folder  
git init  
  
# Create repo in a new directory  
git init project-name  
  
# This creates a .git folder with repository  
data
```

### Clone Existing Repository

```
# Copy a remote repository to your computer  
git clone https://github.com/user/repo.git  
  
# Clone into a specific folder name  
git clone https://github.com/user/repo.git  
my-folder  
  
# Clone a specific branch  
git clone -b branch-name  
https://github.com/user/repo.git
```

## 3. ✓ Basic Workflow

### Check Status

```
# See which files are modified, staged, or  
untracked  
git status  
  
# Short format status  
git status -s
```

### Add Files to Staging Area

```
# Stage a specific file for commit  
git add filename.txt  
  
# Stage all changes in current directory  
git add .  
  
# Stage all changes in entire repository  
git add -A  
  
# Stage all files with specific extension  
git add *.js  
  
# Interactive staging (choose what to stage)  
git add -p
```

### Commit Changes

```
# Save staged changes with a message  
git commit -m "Add new feature"  
  
# Stage all tracked files and commit  
git commit -am "Update documentation"  
  
# Modify the last commit (add forgotten  
files)  
git commit --amend  
  
# Amend without changing the commit message  
git commit --amend --no-edit
```

## 4. ⏪ Viewing History & Changes

### View Commit Log

```
# Show full commit history  
git log  
  
# Compact one-line view  
git log --oneline  
  
# Show visual branch graph  
git log --graph --oneline --all  
  
# Show last 5 commits  
git log -5  
  
# Show commits by specific author  
git log --author="John"  
  
# Show commits with specific word in message  
git log --grep="bug fix"  
  
# Show commits for specific file  
git log filename.txt
```

### View Differences

```
# Show unstaged changes  
git diff  
  
# Show staged changes (ready to commit)  
git diff --staged  
  
# Compare two commits  
git diff commit1 commit2  
  
# Compare two branches  
git diff branch1 branch2
```

### Show Commit Details

```
# Show details of a specific commit  
git show commit-hash  
  
# Show details of latest commit  
git show HEAD
```

## 5. ⚡ Branching

### Why Branches?

```
# Branches let you work on features  
independently  
# Main branch = production code  
# Feature branches = new development
```

### Working with Branches

```
# List all local branches (* = current  
branch)  
git branch  
  
# List all branches (local and remote)  
git branch -a  
  
# Create a new branch  
git branch feature-login  
  
# Switch to a branch  
git checkout feature-login  
  
# Create and switch to new branch (shortcut)  
git checkout -b feature-payment  
  
# Modern way to switch branches  
git switch feature-login  
  
# Create and switch (modern syntax)  
git switch -c feature-payment  
  
# Switch back to previous branch  
git switch -
```

### Delete Branches

```
# Delete a branch (safe - prevents if  
unmerged)  
git branch -d feature-login  
  
# Force delete a branch  
git branch -D feature-login  
  
# Rename current branch  
git branch -m new-name
```

## 6. Merging Branches

### Basic Merge

```
# First, switch to branch you want to merge  
INTO  
git checkout main  
  
# Then merge the feature branch into main  
git merge feature-login  
  
# This combines the changes from  
feature-login into main
```

### Merge Options

```
# Create a merge commit even if fast-forward  
possible  
git merge --no-ff feature-branch  
  
# Combine all commits into one  
git merge --squash feature-branch
```

### Handling Merge Conflicts

```
# If merge conflicts occur, Git will tell  
you  
# 1. Open conflicted files  
# 2. Look for <<<<< ===== >>>>>  
markers  
# 3. Edit to keep the code you want  
# 4. Remove the conflict markers  
# 5. Stage the resolved files  
git add resolved-file.js  
  
# 6. Complete the merge with a commit  
git commit  
  
# Or abort the merge if needed  
git merge --abort
```

## 7. ⌂ Undoing & Restoring

### Unstage Files

```
# Remove file from staging area (keep changes)
git restore --staged filename.txt
# Old way to unstage
git reset HEAD filename.txt
```

### Discard Changes

```
# Discard changes in working directory
git restore filename.txt
# Discard all changes (dangerous!)
git restore .
# Old way to discard changes
git checkout -- filename.txt
```

### Undo Commits

```
# Undo last commit, keep changes staged
git reset --soft HEAD~1
# Undo last commit, unstage changes
git reset --mixed HEAD~1
# Undo last commit, discard all changes (dangerous!)
git reset --hard HEAD~1
# HEAD~1 = one commit back, HEAD~2 = two commits back
```

### Revert Commits

```
# Create new commit that undoes a previous commit
git revert commit-hash
# This is safer than reset for public branches
```

## 8. ☁ Remote Repositories

### View Remotes

```
# List remote repositories
git remote
# List with URLs
git remote -v
# Show detailed info about a remote
git remote show origin
```

### Add/Remove Remotes

```
# Add a new remote repository
git remote add origin
https://github.com/user/repo.git
# Remove a remote
git remote remove origin
# Rename a remote
git remote rename old-name new-name
```

### Authentication

```
# For HTTPS: Use personal access token as password
# For SSH: Set up SSH keys in GitHub settings
# Check if SSH is working:
ssh -T git@github.com
```

## 9. 🔍 Pushing to Remote

### Push Changes

```
# Push commits to remote repository
git push
# Push to specific remote and branch
git push origin main
# Push and set upstream (first time)
git push -u origin feature-branch
# After -u, you can just use: git push
```

### Push All

```
# Push all branches
git push --all
# Push tags to remote
git push --tags
```

### Delete Remote Branch

```
# Delete a branch from remote repository
git push origin --delete branch-name
```

### Force Push (Dangerous!)

```
# Overwrite remote with local (use carefully!)
git push --force
# Safer force push (fails if remote has changes)
git push --force-with-lease
```

## 10. Fetching & Pulling

### Fetch vs Pull

```
# Fetch = download changes, don't merge  
# Pull = fetch + merge (fetch and apply  
changes)
```

### Fetch Changes

```
# Download changes from remote (safe)  
git fetch  
  
# Fetch from specific remote  
git fetch origin  
  
# Fetch from all remotes  
git fetch --all  
  
# Remove references to deleted remote  
branches  
git fetch --prune  
  
# After fetch, you can review changes before  
merging
```

### Pull Changes

```
# Download and merge changes in one step  
git pull  
  
# Pull from specific remote and branch  
git pull origin main  
  
# Pull with rebase instead of merge  
git pull --rebase  
  
# Always pull before pushing to avoid  
conflicts!
```

## 11. Stashing (Temporary Storage)

### Why Stash?

```
# Save work temporarily without committing  
# Useful when switching branches with  
uncommitted work
```

### Basic Stashing

```
# Save current changes temporarily  
git stash  
  
# Stash with a descriptive message  
git stash save "work in progress on login"  
  
# Include untracked files in stash  
git stash -u
```

### View Stashes

```
# List all stashed changes  
git stash list  
  
# Show contents of latest stash  
git stash show  
  
# Show detailed diff of a stash  
git stash show -p stash@{0}
```

### Apply Stashes

```
# Apply latest stash and keep it in list  
git stash apply  
  
# Apply specific stash  
git stash apply stash@{2}  
  
# Apply latest stash and remove from list  
git stash pop
```

### Delete Stashes

```
# Delete specific stash  
git stash drop stash@{0}  
  
# Delete all stashes  
git stash clear
```

## 12. Collaboration Workflow

### Typical Team Workflow:

```
# 1. Clone the repository  
git clone https://github.com/team/project.git
```

```
# 2. Create a feature branch  
git checkout -b feature-new-button
```

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

```
# 4. Push your branch to remote  
git push -u origin feature-new-button
```

```
# 5. Create Pull Request on GitHub  
# - Go to repository on GitHub  
# - Click "Pull requests" > "New pull  
request"  
# - Select your branch and submit for review
```

```
# 6. After approval, update your main  
branch  
git checkout main  
git pull origin main
```

```
# 7. Delete the feature branch (cleanup)  
git branch -d feature-new-button
```

### Syncing a Fork

```
# Add original repo as upstream  
git remote add upstream  
https://github.com/original/repo.git  
  
# Fetch changes from original  
git fetch upstream  
  
# Switch to main branch  
git checkout main  
  
# Merge upstream changes  
git merge upstream/main  
  
# Push updates to your fork  
git push origin main
```

## 13. ❖ Useful Extras

### Tagging Releases

```
# Create a tag (like v1.0, v2.0)
git tag v1.0.0

# Create annotated tag with message
git tag -a v1.0.0 -m "Release version 1.0"

# Push tag to remote
git push origin v1.0.0

# List all tags
git tag
```

### .gitignore File

```
# Create .gitignore to exclude files from
Git
# Common patterns:
*.log      # Ignore all log files
node_modules/ # Ignore folder
.env        # Ignore environment variables
*.tmp       # Ignore temporary files
# Apply gitignore to already tracked files:
git rm --cached filename
git rm -r --cached .
```

### Useful Commands

```
# See who changed each line of a file
git blame filename.txt

# Search for text in repository
git grep "search term"

# Clean untracked files (dry run first!)
git clean -n
git clean -f
```

## 14. 📖 Quick Reference

### Common Terms

**Repository (repo):** Project folder tracked by Git  
**Commit:** Snapshot of changes  
**Branch:** Independent line of development  
**HEAD:** Pointer to current commit  
**Origin:** Default name for remote repository  
**Main/Master:** Default primary branch  
**Stage:** Prepare files for commit  
**Clone:** Copy repository to local machine  
**Fork:** Personal copy of someone's repository  
**Pull Request (PR):** Request to merge changes

### Essential Shortcuts

```
# Create alias for common commands
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
# Now you can use: git st, git co, git br,
git ci
```

## 15. ❓ Best Practices

### DO:

- Commit often with clear messages
- Pull before you push
- Use branches for new features
- Write meaningful commit messages
- Review changes before committing
- Keep commits focused (one feature/fix)

### DON'T:

- Commit sensitive data (passwords, API keys)
- Use `git push --force` on shared branches
- Make huge commits with many unrelated changes
- Commit directly to main branch
- Leave merge conflict markers in code

### Good Commit Messages:

Add user authentication feature  
Fix bug in payment calculation  
Update README with setup instructions  
Refactor database connection logic

### Bad Commit Messages:

fix  
updated stuff  
changes  
asdfasdf

❤️ **Remember:** Git is about collaboration and version control. Commit often, write clear messages, and always pull before you push!

❓ **Need Help?** Use `git help <command>` or visit <https://git-scm.com/doc>