Git Command Cheat Sheet

Quick reference for all essential Git commands

Git Tutorial by Koorosh Komeili Zadeh and De Leidsche Flesch

CREATE

$ git init

```
git init
```

Initialize a new Git repository in the current directory

Use case: Use this when starting a new project to track changes with Git. Creates a .git folder that stores all version control information.

$ git clone

```
git clone
```

Clone a repository into a new directory

Use case: Download a copy of a remote repository to your local machine. Use this to start working on an existing project or to make a local backup.

LOCAL CHANGES

$ git status

```
git status
```

Show the working tree status - see which files are tracked, modified, added, or deleted

Use case: Check which files have been modified, which are staged for commit, and which are untracked. Use this frequently to understand your repository's current state.

$ git add

```
git add OR git add -A (add all)
```

Add file contents to the staging area (index) to prepare for commit

Use case: Stage changes before committing. Use 'git add .' to stage all changes, or specify individual files. This is like preparing a package before shipping it.

$ git commit

```
git commit -m ""
```

Record changes to the repository with a descriptive message

Use case: Save your staged changes as a snapshot in the repository history. Think of it as a checkpoint in your project that you can return to later.

$ git diff

```
git diff OR git diff --staged
```

Show changes between commits, commit and working tree, etc.

Use case: View what has changed in your files before staging or committing. Use --staged to see changes that are already staged for commit.

$ git stash

```
git stash OR git stash pop
```

Temporarily save changes without committing them

Use case: Save your work temporarily when you need to switch branches but aren't ready to commit. Later retrieve it with 'git stash pop'.

COMMIT HISTORY

$ git log

```
git log OR git log --oneline (compact view)
```

Show commit logs - view the history of commits in the repository

Use case: Review project history, see who made what changes and when. Useful for understanding how the project evolved and finding specific commits.

BRANCHES & TAGS

$ git branch

```
git branch OR git branch -d
```

List, create, or delete branches in your repository

Use case: Create separate lines of development to work on features without affecting the main code. Essential for team collaboration and testing new ideas.

$ git checkout

```
git checkout OR git checkout -b
```

Switch branches or restore files to a specific state

Use case: Move between different branches to work on separate features. Use -b to create and switch to a new branch in one command.

UPDATE & PUBLISH

$ git fetch

```
git fetch origin
```

Download objects and refs from remote repository without merging

Use case: Download changes from remote repository but don't merge them. Allows you to review changes before integrating. Safer than git pull.

$ git pull

```
git pull origin
```

Fetch from and integrate with another repository or local branch

Use case: Download and merge changes from a remote repository. Use this to update your local copy with the latest changes from the team.

$ git push

```
git push origin
```

Update remote repository with local commits

Use case: Upload your local commits to a remote repository. Essential for sharing your work with the team and backing up your code.

$ git remote

```
git remote add OR git remote -v
```

Manage set of tracked repositories (add, remove, rename remotes)

Use case: Connect your local repository to remote servers like GitHub. Use to add, remove, or view remote repositories that you sync with.

MERGE & REBASE

$ git merge

```
git merge
```

Join two or more development histories together

Use case: Combine changes from one branch into another, typically merging feature branches back into main. Integrates completed work into the main codebase.

$ git rebase

```
git rebase
```

Reapply commits on top of another base branch

Use case: Alternative to merge that creates a linear history. Moves your branch commits to start from the tip of another branch. Useful for keeping a clean project history.

UNDO

$ git revert

```
git revert
```

Create a new commit that undoes changes from a previous commit

Use case: Safely undo a commit without rewriting history. Creates a new commit that reverses the changes, preserving the complete project history.

$ git reset

```
git reset --hard OR git reset --soft HEAD~1
```

Reset current HEAD to specified state - move or remove commits

Use case: Move the branch pointer to a different commit. --hard discards changes, --soft keeps them staged. WARNING: Can rewrite history if used on pushed commits.