Git Commands Cheat Sheet: Without Code Screens, Plus Tips and Tricks

Understanding Git is crucial for version control and efficient collaboration in software projects. Here is a cheat sheet that serves as a quick reference guide for commonly used Git commands, along with some tips and tricks to make your Git experience smoother.

Basic Commands

Git Init: Use this to initialize a new Git repository in your project directory. This is often the first step in setting up version control for a new project.

Tip: It's best to initialize the Git repository at the root level of your project to capture all the files and sub-directories.

Git Clone: This command allows you to clone or download a repository from a remote source, like GitHub.

Tip: When cloning a repository, pay attention to the branch you are cloning. By default, Git will clone the default branch (often 'main' or 'master').

Working with Branches

Git Branch: Use this to list all the branches in your local repository.

Tip: Regularly run this command to ensure you know which branch you are on before making changes.

Git Branch with Branch Name: Create a new branch with this command and the specified name.

Tip: Keep your branch names descriptive so that you can easily switch between tasks.

Git Checkout: This command allows you to switch to a specified branch.

Tip: Use 'git checkout -b' followed by the branch name to create and switch to a new branch in a single command.

Staging and Committing
Git Status: Shows the status of changes in the working directory, staged for your next commit.

Tip: Use this often to track your changes and avoid accidental commits.

Git Add with File Name: Use this to add a specific file to the staging area before committing.

Tip: This command is perfect for chunking your changes into smaller, manageable commits.

Git Add Dot: Adds all new and changed files to the staging area.

Tip: Be cautious when using this command; it's easy to include files you might not want to commit.

Git Commit: Use this command to commit the staged files and add a message describing the changes.

Tip: Craft your commit messages to be descriptive and straightforward, so they serve as useful documentation.

Merging and Pulling

Git Merge: Use this command to merge the specified branch into the current branch you're working on.

Tip: Before merging, make sure you're on the branch you want to merge into.

Git Pull: This fetches changes from a remote repository and merges them into the branch you're currently on.

Tip: Running a 'git pull' regularly keeps your local repository updated with the latest changes.

Remote Repositories

Git Remote Add: Use this to add a remote repository, providing it a name and URL for easy reference.

Tip: 'origin' is a conventional name used for the primary repository, but you can name your remote anything that makes sense to you.

Git Push: This command pushes your commits to a remote repository, sharing your updates with others.

Tip: Always do a 'git pull' before pushing to avoid conflicts.

Git Remote -V: Lists all the remote repositories you have configured.

Tip: Use this to double-check the remotes if you encounter any issues with pushing or pulling.

Miscellaneous

Git Log: This shows the commit history for the currently active branch.

Tip: Use 'git log --oneline' for a more compact view of the commit history.

Git Revert: Use this to revert changes from a specific commit.

Tip: Reverting is a safe way to undo changes, as it creates a new commit that undoes the changes rather than altering history.

Git Reset: This command resets your HEAD pointer to a previous commit and discards changes that came after it.

Tip: Be cautious when using 'git reset', especially with the '--hard' option, as it will discard changes irreversibly.

This cheat sheet aims to cover the most commonly used Git commands and should significantly improve your proficiency in Git-based version control.