Cheatsheet

Git Cheatsheet

Everything You Need to Know in One Handy Cheatsheet

Git is a version control system that allows you to track changes to files and folders. It's a powerful tool that can be used for everything from small personal projects to large-scale enterprise applications.

This guide is a quick reference to the most common Git commands. It's not meant to be a comprehensive guide to Git, but rather a quick reference to the most common commands.

Here's the revised git cheatsheet with improved section titles and reorganized:

Setup and Configuration

```
# Initialize a new Git repository
git init
# Clone and create a local copy of a remote repository
git clone <url>
# Configure global Git settings
git config --global <setting_name> <value>
# Configure local Git settings for a specific repo
git config --local <setting_name> <value>

# --------------- Advanced ------------------

# Show a summary of your Git configuration settings
git config --list
# Set a custom text editor for Git messages
git config --global core.editor "<editor_command>"
# Create a Git command alias
git config --global alias.<shortcut> <command>
# Enable automatic colorization of Git output
git config --global color.ui auto
# Cache Git credentials for a certain amount of time
git config --global credential.helper 'cache --timeout=<seconds>'
# Configure git to detect specific types of whitespace errors
git config --global core.whitespace <options>
# Automatically prune remote-tracking branches when fetching updates
git config --global fetch.prune true
# Set a custom diff tool for Git
```

```
git config --global diff.tool <tool>

# Set a custom merge tool for Git

git config --global merge.tool <tool>

# Compare changes using a custom diff tool

git difftool

# Resolve merge conflicts with a custom merge tool

git mergetool
```

File Operations

```
# Show working tree status

git status

# Add files to the staging area

git add <file(s)>

# Remove files from working tree and staging area

git rm <file(s)>

# Move or rename a file

git mv <old_file> <new_file>

# Commit changes with a message

git commit -m "commit message"

# Show differences between working tree and last commit

git diff


# --------------- Advanced ------------------
```

```
# Assume a tracked file is unchanged
git update-index --assume-unchanged <file>
# Restore normal behavior of tracking changes
git update-index --no-assume-unchanged <file>
# Show differences between two commits
git diff <commit_id1>..<commit_id2>
# Unstage a file, but keep in the working directory
git rm --cached <file_name>
```

Branching and Merging

```
# List all branches
git branch
# Create a new branch
git branch <branch_name>
# Switch to a specific branch
git checkout <branch_name>
# Merge a branch into the current branch
git merge <branch_name>
# Delete a specific branch
git branch -d <branch_name>
# List all remote branches
git branch -r
```

```
# --------------- Advanced ------------------

# List branches with additional information
git branch -vv
# Create a new branch based on a remote branch
git checkout -b <branch_name> <remote_name>/<remote_branch>
# Cancel merge in case of conflicts
git merge --abort
# Rebase the current branch onto another branch
git rebase <branch_name>
# Cancel an ongoing rebase operation
git rebase --abort
# Interactive rebase for edit, squash, re-order or drop commits
git rebase -i
# Rebase commits in the current branch onto a remote branch interactively
git rebase -i <remote_name>/<remote_branch>
```

Remote Repositories

```
# List remote repositories
git remote
# Add a remote repository
```

```
git remote add <name> <url>

# Fetch from a remote repository

git fetch <remote_name>

# Pull changes from a remote branch

git pull <remote_name> <remote_branch>

# Push changes to a remote repository

git push <remote_name> <local_branch>

# Remove a remote repository

git remote rm <remote_name>

# Display information about a specific remote repository

git remote show <remote_name>

# Show the tracking branches for remote repositories

git remote show <remote_name> --verbose


# --------------- Advanced -------------------


# Fetch updates from all remote repositories

git remote update

# Force-push changes to a remote repository, overwriting remote history

git push --force <remote_name> <local_branch>

# Push all tags to a remote repository

git push --tags <remote_name>

# Rename a remote repository

git remote rename <old_name> <new_name>

# Change the URL of a remote repository

git remote set-url <name> <new_url>

# Remove stale remote-tracking branches
```

```
git remote prune <remote_name>
# List all remote branches that have been merged into the current branch
git branch -r --merged
# List all remote branches not yet merged into the current branch
git branch -r --no-merged
# Fetch updates from a remote repository and prune obsolete remote-trackin
git fetch -p

# Track a remote branch and set up the local branch to automatically sync
git branch --track <branch_name> <remote_name>/<remote_branch>
# Set an existing local branch to track a remote branch
git branch -u <remote_name>/<remote_branch>
# Push a branch to a remote repository and set it to track the remote bran
git push -u <remote_name> <local_branch>
# Remove the tracking association between a local and a remote branch
git branch --unset-upstream <branch_name>
```

Commit History

```
# Show commit history
git log
# Display a condensed commit history
git log --oneline
```

```
# Show branching commit history
git log --graph
# Filter commit history by author
git log --author=<author_name>
# Show commit history since specific date
git log --since=<date>
# Show commit history until specific date
git log --until=<date>
```

Tags

```
# List all tags
git tag
# Create a new tag at a specific commit
git tag <tag_name> <commit_id>
# Create an annotated tag with a message
git tag -a <tag_name> -m "tag message"
# Delete a specific tag
git tag -d <tag_name>
# Delete a specific remote tag
git push <remote_name> --delete <tag_name>
# Show information about a specific tag
git show <tag_name>
```

Stashes

```
# Temporarily save changes in the working tree
git stash save "stash message"
# List all stashes
git stash list
# Apply changes from a specific stash
git stash apply <stash>
# Remove a specific stash
git stash drop <stash>
# Remove all stashes
git stash clear
```

Cherry-Picking

```
# Apply a specific commit from one branch to another
git cherry-pick <commit_id>
```

Commit Management

```
# Modify the latest commit
git commit --amend
# Create a new commit that undoes changes from a previous commit
git revert <commit_id>
# Discard changes and move HEAD to a specific commit
git reset --hard <commit_id>
# Move HEAD to a specific commit, but preserve staged changes
git reset --soft <commit_id>
# Show a record of all changes made to the local repository head
git reflog
```

Submodules, Subtrees, and Advanced Submodules

```
# Add a submodule to the current repository
git submodule add <repository_url> <path>
# Initialize and update all submodules recursively
git submodule update --init --recursive
# Add a subtree to the current repository
git subtree add --prefix=<path> <repository_url>
# Initialize the submodules in the repository
git submodule init
# Update the submodules to their latest commits
```

```
git submodule update

# Execute a specific command in each submodule

git submodule foreach <command>

# Unregister a submodule

git submodule deinit <path>
```

Hooks and Automation, and Diff and Merge Tools

```
# Locate hooks directory in the Git repository (usually in .git/hooks/)

git hooks

# Script names for specific hooks that can be added to the hooks directory

pre-commit, post-commit, pre-push, post-merge, etc.

# Make a hook script executable to ensure it's triggered when necessary

chmod +x <hook_script>
```

Work with Patches

```
# Generate a patch file for a specific commit

git format-patch <commit_id>
```

```
# Apply a patch to the current branch
git apply <patch_file>
# Apply a patch using the "git am" (apply mailbox) command
git am <patch_file>
```

Collaboration

```
# Generate a request-pull summary with the changes between two commits
git request-pull <start_commit> <end_commit> <url>
# Summarize the commit history, listing authors and their contributions
git shortlog
# List all files tracked by Git
git ls-files
# Search for a specified pattern in files tracked by Git
git grep <pattern>
```

Bisecting, Debugging, and Performance Issues

```
# Begin a bisect session to find the commit that introduced a bug
git bisect start
# Mark a commit as "bad," indicating it contains the bug
git bisect bad <commit_id>
# Mark a commit as "good," indicating it does not contain the bug
git bisect good <commit_id>
# End the bisect session and return to the original branch/commit
git bisect reset
# Verify the integrity of the Git repository
git fsck
# Run garbage collection to optimize the repository's performance
git gc
# Remove untracked files and directories (use with caution)
git clean -df
```

Tips and Tricks

```
# Interactively choose parts (hunks) of files to stage
git add -p

# Show the commit history and associated patches for a specific file
git log -p <file_name>
# Customize the format of the git log output
```

```
git log --pretty=format:"%h - %an, %ar : %s"
# Find text in commit messages (useful for locating specific changes)
git log --grep="<text>"
# Quickly view the changes in the working directory since the last commit
git diff --stat
# Display the branch history with decoration to see where branches have sp
git log --oneline --decorate --graph

# Stash changes in the working tree, including untracked files
git stash save -u
# Create an empty commit, useful while testing branch protection rules
git commit --allow-empty -m "Empty commit message"

# Set the git output pager to quit when the output is less than one screen
git config --global core.pager 'less -RFX'
# Use Git's auto-correct feature to fix mistyped commands
git config --global help.autocorrect 1
# List aliases for Git commands
git config --get-regexp alias

# Perform a dry run of merging without actually merging branches
git merge --no-commit --no-ff <branch_name>
# Show a tree-like representation of the repo's structure
git ls-tree --name-only -r -t HEAD
```