Git Commands Cheat Sheet | Learn Git

gitkraken.com/learn/git/commands

Git Branch Commands

- `git branch` – Display a list of the local branches in your Git repository.
- `git branch -a` – Display a list of both local branches and remote branches in your Git repository.
- `git branch -c` – Copy a Git branch.
- `git branch -d <branch-name>` – Delete a local Git branch. This command will not work if the branch you are attempting to delete has unmerged changes.
- `git branch -D <branch-name>` – Delete a local Git branch with unmerged changes.
- `git branch -m <branch-name> <new-branch-name>` – Rename a Git branch.
- `git branch -r` – Display a list of the remote branches in your Git repository.
- `git push <remote> --delete <remote-branch-name>` – Delete a remote Git branch.
- `git push --set-upstream <remote> <branch>` – Set an upstream branch. Running this command will push your local branch to the new remote branch.

Git Checkout Commands

- `git checkout <branch-name>` – Switch to a different Git branch.
- `git checkout -b <branch-name>` – Create a new branch and switch to it.
- `git checkout -b <branch-name><remote-name>/<branch-name>` – Create a local branch from the remote Git branch and checkout that branch.
- `git checkout <commit hash>` – Checkout a previous Git commit.
- `git checkout <tag name>` – Checkout a Git tag in a detached HEAD state.
- `git checkout -b <branch-name><tag-name>` – Checkout a Git tag as a branch.

Git Cherry Pick Commands

`git cherry-pick [insert commit reference]` – Apply a commit's changes onto a different branch.

Git Clone Commands

- `git clone <repository-url>` – Clone a specified remote repository. See Git-SCM's best practices for remote URL format.
- `git clone <repository-url> <directory-name>` – Clone a repository and name the local directory.

- `git clone <repository-url> --origin <name>` – Clone a repository and name the remote (`<name>`). If you do not wish to name the remote, Git will provide the default name `origin`.
- `git clone <repository-url> --branch <branch-name>` – Clone a repository and checkout the specific branch.
- `git clone <repository-url> --depth <depth>` – Clone a repository with a specified number of commits (`<depth>`).
- `git clone <repository-url> --no-tags` – Clone a repository without copying the repo's tags.

Git Commit Commands

- `git status` – Display a list of files in your staging directory with accompanying file status.
- `git add` – Stage file changes. Running this command with an associated file name will stage the file changes to your staging directory.
- `git commit` – Save changes to your Git repository. Running this command with an associated file name will save the file changes to your repo.
- `git commit -a` – Add all modified and deleted files in your working directory to the current commit.
- `git commit --amend` – Amend a Git commit. Edit a Git commit message by adding a message in quotation marks after the command.
- `git commit -m` – Add a Git commit message. Add your message in quotation marks following the command.

Git Config Commands

- `git config --global` – Customize configurations stored in your home directory, and can overwrite Git config system settings.
- `git config --email` – Sets the email associated with your Git commits and other Git actions.
- `git config --system` – Customize configuration settings for your operating system.
- `git --config user.name` – Sets the username associated with your Git commits and other Git actions.
- `git config --list` – View all of your Git config settings including the local, global, and system levels.
- `git config --local` – Customize settings that are Git repository specific and overwrite Git configurations on the global and system level.

Git Merge Commands

- `git merge` – Combine two or more development histories together. Used in combination with fetch, this will combine the fetched history from a remote branch into the currently checked out local branch.
- `git merge <branch-name>` – Merge changes from one branch into the branch you currently have checked out.
- `git merge --abort` – Aborts the merge process and restores the project's state to before the merge was attempted. This works as a failsafe when a conflict occurs.
- `git merge --continue` – Attempt to complete a merge that was stopped due to file conflicts after resolving the merge conflict.
- `git merge --squash` – Combine all changes from the branch being merged into a single commit rather than preserving them as individual commits.
- `git merge --no-commit` – Combine branch into the current branch, but do not make a new commit.
- `git merge --no-ff` – Creates a merge commit instead of attempting a fast-forward.

Git Pull Commands

- `git pull` – This will perform a `git fetch` followed by a `git merge FETCH_HEAD`, and will allow you to fetch from and integrate with another repository or a local branch.
- `git pull --quiet` – Suppress the output text after both `git fetch` and `git merge`.
- `git pull --verbose` – Expand the output text after both `git fetch` and `git merge`.

Git Pull Commands Related to Merge

- `git pull --squash` – Combine all changes from the branch being merged into a single commit, rather than preserving the individual commits.
- `git pull --no-commit` – Combine the currently checked out branch with the remote upstream branch.
- `git pull --no-ff` – Create a merge commit in all cases, even when the merge could instead be resolved as a fast-forward.

Git Pull Commands Related to Fetch

- `git pull --all` – Fetch all remotes.
- `git pull --depth=<depth>` – Fetch a limited number of commits.
- `git pull --dry-run` – Show the action that would be completed without actually making changes to your repo.
- `git pull --prune` – Remove all remote references that no longer exist on the remote.
- `git pull --no-tags` – Do not fetch tags.

Git Push Commands

- `git push` – Push the current checked out branch to the default remote `origin`.
- `git push <remote><branch>` – Push the specified branch along with all of its necessary commits to your destination remote repository.
- `git push <remote> --force` – Force a Git push in a non-fast-forward merge. This option forces the update of a remote ref even when that is not the ancestor of the local ref. This can cause the remote repository to loose commits, so use with care.
- `git push <remote> --all` – Push all local branches to a specified remote.
- `git push <remote> --tags` – Push all local tags to a specified remote. Tags are not automatically sent when using `--all`.

Git Rebase Commands

- `git rebase <target branch name>` – Rebase your currently checked out branch onto a target branch. This rewrites a commit(s) from the source branch and applies it on the top of the target branch.
- `git rebase --continue` – Proceed with a Git rebase after you have resolved a conflict between files.
- `git rebase --skip` – Skip an action that results in a conflict to proceed with a Git rebase.
- `git rebase --abort` – Cancel a Git rebase. Your branch will be back in the state it was before you started the rebase.
- `git rebase <target branch name> -i` – Initiate interactive rebase from your currently checked out branch onto a target branch.

Git Stash Commands

- `git stash` – Create a stash with local modifications and revert back to the head commit.
- `git stash list` – Display a list of all stashes in your repository.
- `git stash show` – View the content of your most recent stash. This will show your stashed changes as a diff between the stashed content and the commit from back when the stash was created.
- `git stash drop <stash>` – Remove a stash from the list of stashes in your repository.
- `git stash pop <stash>` – Apply a stash to the top of the current working tree and remove it from your list of stashes.
- `git stash apply <stash>` – Apply a stash on top of the current working tree. The stash will not be removed from your list of stashes.
- `git stash clear` – Remove all stashes from your repository.
