Git Cheat Sheet

Daniel Park

March 26, 2022

Contents

1 Preliminaries 2

2 Terminal commands 2

3 Git commands 3

4 Github commands 5

1 Preliminaries

Step	Description
Installing VS Code	Open up VSCode. Go to 'Shell Command: Install 'code command ...' to set up the environment variable. This enables the command `code .`
`command -flag` ⟨`dir`⟩	Every terminal and git command follows the given format. All file and folder names ⟨dir⟩ must be in quotations.
Configure VS Code	Run `git config --global core.editor "code --wait` in git bash. This will configure VS Code as the default git editor over VIM.

2 Terminal commands

Command	Description

`ls`
`ls` ⟨`folder`⟩
`ls -a`
	Lists contents of current directory
	Lists contents of ⟨folder⟩
	Lists all directories and files, including hidden ones

`start .` `open .`	Opens up the file explorer in the current directory The `start` command on Mac
`pwd`	Shows the current directory location

`cd` ⟨`folder`⟩
`cd`
`cd ..`
	Changes your current directory to ⟨folder⟩
	Go back to home directory. Note that denotes home directory
	Move backwards to the parent folder of current directory

`clear`	Clears the terminal
`code .`	Opens the current directory in VSCode
`touch` ⟨`file`⟩ `touch` ⟨`file1`⟩ ⟨`file2`⟩ `mkdir` ⟨`folder`⟩	Creates a file called ⟨file⟩ in the current directory Creats files called ⟨file1⟩ and ⟨file2⟩ in the current directory Creates a folder called ⟨folder⟩
`rm` ⟨`file`⟩ `rm -rf` ⟨`folder`⟩	Deletes the file called ⟨file⟩ Deletes the folder ⟨folder⟩

3 Git commands

Git structure:

$$\text{Working directory} \xrightarrow{\texttt{git add}} \text{Staging area} \xrightarrow{\texttt{git commit}} \text{Local repo} \xrightarrow{\texttt{git push}} \text{Remote repo}$$

Git GUI: Git Kraken. Use the GUI to copy commit hashes.

Resources:

- Full documentation: git-scm.com/docs
- Book on git: git-scm.com/book

Git ignore template: git ignore io

- Use `touch '.gitignore'` to create the file.
- Set up the .gitignore file first before committing files. Otherwise you must remove the files to be ignored from the cache.

Command	Description
`git init`	Initializes the repository in the current directory. Creates a hidden folder called '.git'.
`git status`	Shows the status of the current git repository.
`git add ⟨file⟩` `git add .`	Adds ⟨file⟩ to the staging area. Adds all changed files to the staging area.

`git commit`
`git commit -m ⟨msg⟩`
`git commit --amend`
	Uploads all staged files to the git repository. Creates a commit.
	Creates a commit with the message ⟨msg⟩.
	Updates the previous commit with all currently staged changes.

`git log` `git log --oneline`	Shows the history of all commits made in the repo. Condenses the log into abbreviated hashes and the header line of each commit message.
`git branch` `git branch ⟨branch⟩`	Lists all branches. Active branch has an asterisk (*). Creates a new branch ⟨branch⟩ with the parent commit being where HEAD is on. Multiple branches share the same parent commit.

`git switch ⟨branch⟩`
 	
`git checkout ⟨branch⟩`
`git switch -c ⟨branch⟩`
 	
`git checkout ⟨branch⟩`
	Switches the active branch to ⟨branch⟩. All unstaged changes are lost.
	A generalized command of `git switch` and `git restore`.
	
	Creates a new branch and switches you over to it.
	
	Does the same thing as `git switch -c ⟨branch⟩`.

`git branch -d ⟨branch⟩`	Deletes the branch ⟨branch⟩. Must be full merged before doing so.

`git branch -D` ⟨branch⟩	Deletes the branch ⟨branch⟩ unconditionally.
`git branch -m` ⟨branch⟩	Renames the current branch name to ⟨branch⟩.
`git merge` ⟨branch⟩	Merges ⟨branch⟩ to the active branch. If there is a merge conflict, it must be resolved manually. After, you must stage and commit these resolved changes.
`git diff`	Shows all unstaged changes since the last commit.
`git diff HEAD`	Lists all changes in the working tree since your last commit. Includes both staged and unstaged changes.
`git diff` ⟨branch1⟩ ⟨branch2⟩	Lists all changes between the tips of ⟨branch1⟩ and ⟨branch2⟩.
`git diff` ⟨commit1⟩ ⟨commit2⟩	Lists all changes between the commit hashes ⟨commit1⟩ and ⟨commit2⟩.
`git stash (save)`	Saves all uncommitted changes to the stash.
`git stash pop`	Removes the most recently stashed items and reapplies them to your working directory.
`git stash clear`	Clears all stashes.
`git checkout HEAD` ⟨file⟩	Discard any changes made in that file, reverting it back to HEAD.
`git checkout HEAD` ⟨k⟩	Detach HEAD k commits back. To re-attach HEAD, use the `git switch` command.
`git restore` ⟨file⟩	Alternative command to `git checkout`.
`git restore --source` ⟨commit⟩ ⟨file⟩	Reverts ⟨file⟩ to its previous state at the commit ⟨commit⟩.
`git restore --staged` ⟨file⟩	Unstages the file ⟨file⟩.
`git reset` ⟨commit⟩	Resets the repo back to a previous commit. All changes persist as unstaged changes.
`git reset --hard` ⟨commit⟩	Implements git reset, and all changes are removed from your working directory.
`git revert`	Undo the changes as a new commit. Does not delete prior commits made.

4 Github commands

Command	Description
`git clone ⟨url⟩`	Retrieves all files associated with the github repository in ⟨url⟩ and will copy them to your local machine. Creates a new folder with the cloned repo in it, and assigns the default remote name 'origin'.
`git remote -v`	Lists all remote names.
`git remote add ⟨name⟩ ⟨url⟩`	Creates a new remote called ⟨name⟩ linked to the github repo ⟨url⟩.
`git remote rename ⟨old-name⟩ ⟨new-name⟩`	Renames the remote ⟨old-name⟩ to ⟨new-name⟩.
`git remote remove ⟨name⟩`	Removes the remote ⟨name⟩.
`git push ⟨remote⟩ ⟨branch⟩`	Pushes work in branch up to the remote branch.
`git push -u ⟨remote⟩ ⟨branch⟩`	Sets the upstream of the local branch so that it tracks the corresponding remote branch with the same name. This allows us to set up the shortcut command `git push`.
`git push ⟨remote⟩ ⟨local-b⟩:⟨remote-b⟩`	Pushes the local branch ⟨local-b⟩ up to the remote branch ⟨remote-b⟩.
`git branch -r`	View all remote branches tracked by the local repository.
`git switch ⟨remote-branch⟩`	Creates a local branch called ⟨remote-branch⟩ and sets it up to track the remote branch ⟨remote⟩ / ⟨remote-branch⟩.
`git fetch ⟨remote⟩ ⟨remote-b⟩`	Takes remote changes from the remote branch ⟨remote-b⟩ and creates a new branch called ⟨remote⟩/⟨remote-b⟩ on your local repo.
`git pull ⟨remote⟩ ⟨branch⟩`	Updates our HEAD branch and whatever changes are retrieved from the remote branch ⟨branch⟩. Essentially, `git fetch` + `git merge`.