CO D/ NG

# CHEAT SHEET

Web & App Development

## LEARN GIT

- CLICK TO JUMP TO THE PAGE -


1. Install git....3
   - Windows....3
   - Mac....3
   - Linux....3
2. Global configurations....4
   - Username....4
   - Email....4
   - Pull rebase true....4
   - Pull rebase false....4
   - Core editor....4
3. Basic commands....5
   - Git init....5
   - Remove .git....5
   - Git add <file name>....5
   - Git add ....5
   - Git status....5
   - Git commit....6
   - Git commit -m....6
   - Git log....6
   - Git log --oneline....6
   - Git reflog....6
4. Git branches....7
   - Git branch....7
   - Git branch -r....7
   - Git branch -a....7
   - Git branch <branch name>....7
   - Git checkout <branch name>....7
   - Git checkout -b <branch name>....7
   - Git branch -d <branch name>....8
   - Git branch -D <branch name>....8
5. Git and Github or other remotes....9
   - Git add <remote alias> <remote url>....9
   - Git clone <remote url>....9
   - Git remote....9
   - Git remote -v....9
   - Git push -u <remote alias> <branch name>....9

Git push <remote> <branch name>..... 10
Git push..... 10
Git pull..... 10
Git pull <remote> <branch name>..... 10
**6. Git merge and rebase..... 11**
Git merge <branch name>..... 11
Git rebase <branch name>..... 11
**7. Git reset and revert..... 12**
Git reset <commit id>..... 12
Git reset --hard <commit id>..... 12
Git revert <commit id>..... 12

## 1. Install git

Go to https://git-scm.com/downloads and choose your operating system to see how you can download or/and install git on your computer or follow the instructions below.

### Windows

In the windows option:

1. Download the windows software.
2. Install it on your computer.
3. Open the git bash terminal you have just installed to start using git.

Back to index


### Mac

Install homebrew by pasting the command on the command line and pressing enter.
https://brew.sh/

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install the latest version of git when homebrew is done installing by pasting the command on the command line.

```
brew install git
```

Back to index

### Linux

Install git with this command by typing it into the terminal and pressing enter..

```
apt-get install git
```

Back to index

## 2. Global configurations

### Username

Sets the user's name for git projects. Replace the placeholder with your name inside the double quotes.

Back to index

```
git config --global user.name "Your Name"
```

### Email

Sets the user's email for the git projects. Replace the placeholder with your email.

Back to index

```
git config --global user.email your.email@host.com
```

### Pull rebase true

Sets the behaviour of the pull command to always rebase.

Back to index

```
git config --global pull.rebase true
```

### Pull rebase false

Sets the behaviour of the pull command to always merge. This is the default configuration.

Back to index

```
git config --global pull.rebase false
```

### Core editor

Sets the terminal in VSCode as the default editor for the commit messages.

Back to index

```
git config --global core.editor "code --wait"
```

## 3. Basic commands

### Git init

Initiates a local repository in the current location of your terminal by creating a hidden .git folder at the chosen directory.

Back to index

```
git init
```

### Remove .git

Removes the **.git** file to delete the local repository.

Back to index

```
rm -rf .git
```

### Git add <file name>

Adds all untracked and modified changes of a specific file to the staging area.

Back to index

```
git add index.html
```

### Git add .

Adds all untracked and modified changes of all files and folders of the directory you're in.

Back to index

```
git add .
```

### Git status

Shows the current status of the files in your repository.
Unstaged files are shown in red, while files that were staged with a previous git add command are shown in green.

Back to index

```
git status
```

### Git commit

Creates a commit to store the changes that had been staged in the git history log. This commit will open a text field in the terminal for you to write a message describing the commit.

Back to index

```
git commit
```

### Git commit -m

Creates a commit to store the changes that had been staged in the git history log and the -m flag allows to automatically write the commit message in the command itself.

Back to index

```
git commit -m "My first commit"
```

### Git log

Shows you the history of commits in detail in reverse chronological order.

Back to index

```
git log
```

### Git log --oneline

The flag -oneline displays a shortened, one line view for each commit in the commit history.

Back to index

```
git log --oneline
```

### Git reflog

Shows the reference logs saved in history. The reference logs are private and never pushed to the remote. They record commits, branch creation, branch deletion and other changes.

Back to index

```
git reflog
```

## 4. Git branches

### Git branch

Shows you a list of all the local branches.

Back to index

```
git branch
```

### Git branch -r

Shows you a list of all the remote branches.

Back to index

```
git branch -r
```

### Git branch -a

Shows you a list of all the local and remote branches.

Back to index

```
git branch -a
```

### Git branch <branch name>

Creates a new branch with the name that is passed in the command.

Back to index

```
git branch dev
```

### Git checkout <branch name>

Changes to the branch that the name is passed in the command.

Back to index

```
git checkout dev
```

### Git checkout -b <branch name>

Creates a new branch with the name that is passed in the command and automatically switches to that branch.

Back to index

```
git checkout -b dev
```

### Git branch -d <branch name>

Deletes the branch with the name that is passed in the command.

Back to index

```
git branch -d dev
```

### Git branch -D <branch name>

Forces the deletion of the branch with the name that is passed in the command when this is not yet merged with main or pushed to the remote.

Back to index

```
git  branch -d dev
```

## 5. Git and Github or other remotes

### Git add <remote alias> <remote url>

Creates a connection between a local repository and a remote one.

Back to index

```
git remote add origin git@github
```

### Git clone <remote url>

Creates a new local repository with the content of a remote one and creates a connection between them automatically.

Back to index

```
git clone git@github
```

### Git remote

Lists the remote connections that your local repo is connected to by alias.

Back to index

```
git remote
```

### Git remote -v

Enumerates the remotes that your local repo is connected to and shows the url that is used to connect to each one of them.

Back to index

```
git remote -v
```

### Git push -u <remote alias> <branch name>

Pushes the declared branch to the remote, and binds this local branch with the remote branch for future push and pull commands.

Back to index

```
git push -u origin main
```

### Git push <remote> <branch name>

Pushes the code from the local branch you are currently on to the remote one that has the same name. If there is no such branch, it will create it, but it will not be setting the local one to follow the remote one.

Back to index

```
git push origin main
```

### Git push

Pushes the code from the local branch you are currently on to the remote one that the local branch is set to follow.
It requires a previous **git push -u** command.

Back to index

```
git push
```

### Git pull

Pulls the code from the remote branch that the current local branch is set to follow.
It requires you to do the **git push -u** command first.

Back to index

```
git pull
```

### Git pull <remote> <branch name>

Pulls the code of the remote branch that is passed in the command from the remote repo that is specified, into the local branch you are currently on.

Back to index

```
git pull origin main
```

## 6. Git merge and rebase

**Git merge <branch name>**

Merges the code of the branch you pass the name in the command into the current branch.
This should be used for all public branches.

Back to index

```
git merge main
```

**Git rebase <branch name>**

Rebases the code of the branch you pass the name in the command into the current branch.
This **should only be used in private branches** that have not yet been pushed to the remote to avoid issues, as this command rewrites history, creating a new id for each commit.

Back to index

```
git rebase main
```

## 7. Git reset and revert

### Git reset <commit id>

It brings your history back to the commit which id you passed in the command. All the other commits will be deleted from history as they never existed. However, the code recorded by those commits will not be deleted, so you can manually add or remove the parts that you want.
It should only be used for branches that are not yet pushed to the remote to avoid conflicts when pushing the code.

Back to index

```
git reset 4965db1
```

### Git reset --hard <commit id>

Works as **git reset**, but the --hard flag will force the deletion of any code that was not stored in the commit you are resetting to.

Back to index

```
git reset --hard 4965db1
```

### Git revert <commit id>

It reverts the code to the one stored in the commit whose id has been passed in the command, removing all the code that later commits have stored. However, instead of deleting those later commits from history, it creates a new commit storing the code revert. This makes it possible to revert the code on public branches, where it is impossible to reset or push a branch that has reset, as the commit history would diverge and cause conflicts.

Back to index

```
git revert 4965db1
```