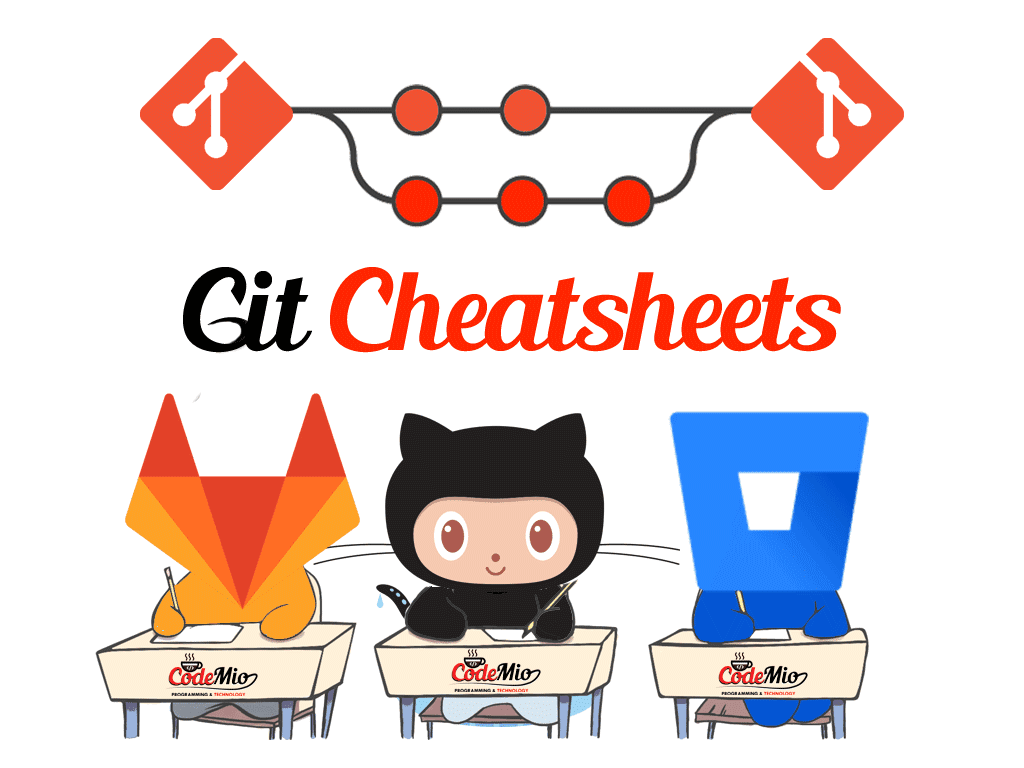
[](https://4.bp.blogspot.com/-b8GFwgTbYas/XLnQLrC4DGI/AAAAAAAAU3Q/cTh3wiMADEID6pkoeULHi4vMy-Aed0DqQCLcBGAs/s1600/codemio-git-cheatsheets.png)

**The Git Workflow**

There are **three** main components of a Git project:

**Repository**

The repository is the component that tracks the changes to the files in your project. It contains all the commits (a snapshot of the files at a specific point) you create.

**Working directory**

The working directory contains the files currently being worked on.

**Staging**

The staging, is where commits are prepared. It allows users to view changes made to the files in the project.  When there is a change to a file in staging, it is marked as modified and the user is able to see the diff (The modifications made to the files).

**Git States**

Files in Git can be in three different states:

* Modified
* Staged
* Commited

When a file is modified, the change can only be found in the working tree. Then, user adds the changes to the staging to be able to commit them later on. The final step is to commit these changes and this allow us to keep a snapshot of our work that we can revisit later.

|  |
| --- |
| <https://4.bp.blogspot.com/-2oRkFnCKNjs/XLvkR-un41I/AAAAAAAAU4c/Sl-D81Nm8tgKM61PQbn3_GKgUAUqx8i5wCLcBGAs/s1600/git-workflow-codemio.png> |
| The tree file states: modified, staged and committed |

**Configuring Git**

Before we can use Git on a working directory, we need to configure it properly. Here are the commands for Git configuration:

# Shows current values for all global configuration parameters

git config --list --global

# Allows git to automatically correct typos such as "commet" and "pusg."

git config --global help.autocorrect 1

# Sets a username globally

git config --global user.name "username"`

# Sets an email address globally

git config --global user.email "email@provider.com"

# Always --prune for git fetch and git pull

git config --global fetch.prune true

# Removes the previously set global username

git config --global --unset user.name

# Colors the git console

git config color.ui true

# Sets the tool used by git for diffing globally

git config --global diff.tool mytool

# Sets the tool used by git for merging globally

git config --global merge.tool mytool

# Removes the previously set configuration value globally

git config --global --unset myparam

# Allows populating the working directory sparsely:

# cloning only certain directories from a repository

git config core.sparseCheckout true

# Instructs Git to retrieve only some directory

# in addition to those listed in `.git/info/sparse-checkout

echo "some/dir/in/the/repository" >> .git/info/sparse-checkout

# Defines which whitespace problems git should

# recognize(any at the end of a line, mixed spaces or tabs)

git config --global core.whitespace trailing-space,space-before-tab

# Always shows a diffstat at the end of a merge

git config --global merge.stat true

# No CRLF to LF output conversion will be performed

git config --global core.autocrlf input

# When pushing, also push local tags

git config --global push.followTags true

# Shows also individual files in untracked directories in status queries

git config --global status.showUntrackedFiles all

# Always decorates the output of git log

git config --global log.decorate full

# Always sets the upstream branch of the current branch as the branch

# to be pushed to when no refspec is given

git config --global push.default tracking

**Initialization and Cloning a Remote Repository**

Initializing Git is the next step after configuring it with the user name, email etc. that we intend to use.

# Initializes a git repository in the current working directory

git init

# Clones a remote repository over https

git clone https://github.com/myrepo.git

# Clones a remote repository over ssh

git clone ssh://git@remote.com:/repo.git

# Recursively clones a repository over https

git clone --recursive https://github.com/myrepo.git

# Recursively clones a repository over ssh

git clone --recursive ssh://git@remote.com:/repo.git

**Add and Commit**

# Tells Git to start tracking a file or add its current state to the index.

git add filename

# Tells Git to add everything which is untracked

# or has been changed to the index. Commonly used.

git add .

# Commits to local history with a given message.

git commit -m "message"

# Adds all changes to already tracked files and commit

# with a given message, non-tracked files are excluded

git commit -am "message"

# Modifies the last commit including both new modifications and given message.

git commit --amend -m "message"

# Performs a commit with an empty message (Discouraged)

git commit --allow-empty-message -m

**Status**

# Shows the status of the local git repository

git status

# Shows a short version of the status

git status -s

**Checking Out**

# Checks out the branchname given.

git checkout branchname

# Switches the working tree with commit b4c21b.

git checkout b4c21b

# Replaces the current working tree with the

# head of the master branch.(Commonly used)

git checkout master

**Working with Remotes**

# Shows the remote branches and their associated urls.

git remote -v

# Adds an URL as a remote branch under the name origin.

git remote add -f origin https://remote.com/repo.git

# Adds an ssh url as remote branch under the name origin.

git remote add -f origin ssh://git@remote.com:/repo.git

# Removes the remote with ID origin.

git remote remove -f origin

# Sets an https url for the remote with ID origin.

git remote set-url origin https://remote.com/repo.git

# Cleans up remote's non-existent branches.

git remote prune origin

# Sets the upstream branch, to which changes will be pushed, to origin/master.

# This is also called switching the tracking target.

git branch --set-upstream-to=origin/master

# Sets name as the tracking branch for origin/master

git branch –track name origin/master

# Updates the local tracking branches with

# changes from their respective remotes.

git fetch

# Updates the local tracking branches and removes

# local references to non-existent remote branches.

git fetch -p

# Deletes remote tracking branch origin/branch

git branch -r -d origin/branch

# Updates the local tracking branches and

# merges changes with local working directory

git pull

# Updates the local tracking branches and attempts to

# move your changes on top of the remote changes pulled.

git pull --rebase

# Given one or more existing commits,

# applies the change each one introduces, recording a new commit for each.

# This requires your working tree to be clean.

git cherry-pick commitID

# Pushes current commits to the upstream URL

git push

# Pushes current commits to the remote named origin

git push origin

# Pushes HEAD to the branch master on the remote origin

git push origin master

# Pushes and sets the origin master as upstream

git push -u origin master

# Deletes previous commits and pushes your current one

# Important: This will break other contributors' local branches,

# never use this if there are many people working on the same branch.

git push --force all origin/master

# Turns the head of a branch into a commit in the currently checked out branch and merge it

git merge --squash mybranch

**Diffing**

# Shows the difference between two branches.

git diff branch1..branch2

git diff --name-status master..branchname

git diff --stat --color master..branchname

git diff > changes.patch

git apply -v changes.patch

**Resetting**

# Unstages file, keeping the file changes

git reset [file]

# Revert everything to the last commit

git reset --hard