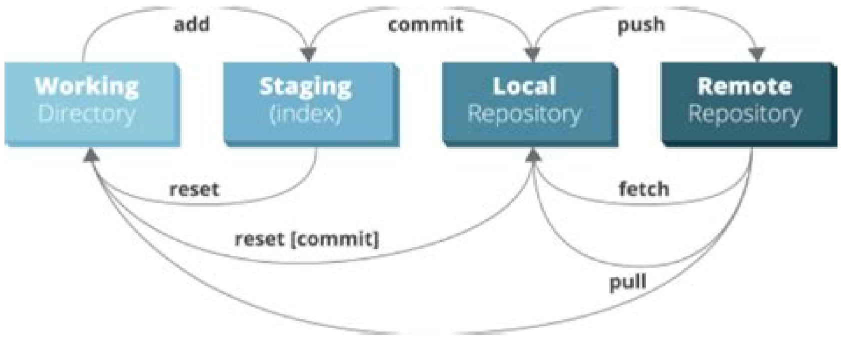
GIT | My first repository

ABOUT VCS

Git is the world's **most popular version control system (VCS)**, and, consequently, knowing how to use Git has become a mandatory skill in the world of coding.The version control is a system that records changes to a file or set of files over time so that you can recall specific versions later and is an essential part of the every-day of the modern software team's professional practices. Git allows multiple developers to work together on the same project with ease.On the other hand, GitHub is a company that offers a cloud-based repository allowing developers to store and manage their code and to track and control code changes. You can use Git without GitHub, but you can't use GitHub without Git.



BASIC COMMANDS

Below is a list of the basic commands to use this tool

Git Setup

Create a new Git repository from an existing directory:

```
git init [directory]
```

Clone a repository (local or remote via HTTP/SSH):

```
git clone [repo / URL]
```

Git configuration

Attach an author name to all commits that will appear in the version history:

```
git config --global user.name "[your_name]"
```

Attach an email address to all commits by the current user:

```
git config --global user.email "[email_address]"
```

Git Managing Files

Attach an author name to all commits that will appear in the version history:

```
git status
```

List the commit history of the current branch:

```
git log
```

Examine the difference between the working directory and the index:

```
git diff
```

Display the content and metadata of an object (blob, tree, tag or commit):

```
git show [object]
```

Git Branches

List all branches in the repository:

```
git branch
```

Switch to a branch under a specified name (if it doesn’t exist, a new one will be created):

```
git checkout [branch]
```

Making changes

Stage changes for the next commit:

```
git add [file/directory]
```

Stage everything in the directory for an initial commit:

```
git add .
```

Commit staged snapshots in the version history with a descriptive message included in the command:

```
git commit -m "[descriptive_message]"
```

To tag our last commit we make:

```
git tag -a entrega1 -m "Entrega final de la tarea 1"
```

Stash the changes in a dirty working directory away

```
git-stash
```

Undoing changes

Undo changes in a file or directory and create a new commit with the git revert command:

```
git revert [file/directory]
```

Unstage a file without overwriting changes:

```
git reset [file]
```

Undo any changes introduced after the specific commit:

```
git reset [commit]
```

Restore changes in a directory

```
git restore . [file/directory]
```

Rewriting history

Replace the last commit with a combination of the staged changes and the last commit combined:

```
git commit --amend
```

Rebase the current branch with the specified base (it can be a branch name, tag, reference to a HEAD, or a commit ID):

```
git rebase [base]
```

git-merge - Join two or more development histories together

```
git merge
```

List changes made to the HEAD of the local repository:

```
git reflog
```

Remote repository

Create a new connection to a remote repository (give it a name to serve as a shortcut to the URL):

```
git remote add [name] [URL]
```

Fetch a branch from a remote repository:

```
git fetch [remote_repo] [branch]
```

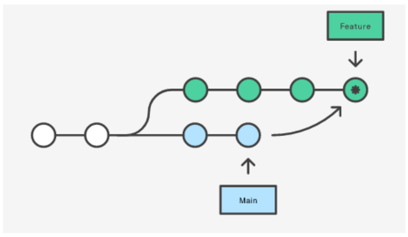
Push a branch to a remote repository with all its commits and objects:

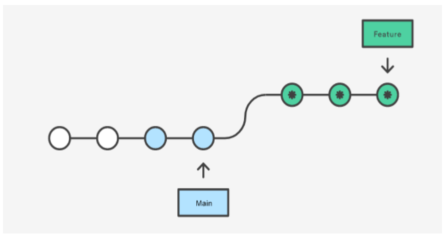
```
git push [remote_repo] [branch]
```

Fetch a repository and merge it with the local copy:

```
git pull [remote_repo]
```

Clarifications

`git merge`	The easiest option to merge the branches is using the git merge command. Git merge safeguards the histories of both the repositories. **Advantage** The Merge command is a non-destructive command. in this command the existing branches are not changed in any way. **Disadvantage** In addition to being easy, there are many few demerits that also merge. 

`git rebase`	Reapply commits on top of another base tip,The golden rule of git rebase is to never use it on public branches. 

Differences between Merge and Rebase

There are two ways that allow us to join branches, git merge and git rebase. The best known form is git merge, which performs a three-way fusion between the last two snapshots of each branch and the common ancestor of both, creating a new commit with the changes mixed. Git basically overdose what it does is collect one by one the confirmed changes in one branch, and reapply them over another. Using overshoot can help us avoid conflicts as long as it is applied on commits that are local and have not been uploaded to any remote repository. If they are not careful with the latter and a colleague uses affected changes, you will surely have problems since these types of conflicts are usually difficult to repair.

Squashing Commit During Merge

You can use **git merge --squash** to squash changes introduced by a branch into a single commit.
No actual commit will be created.

```
git merge --squash <branch>
git commit
```

---

https://git-scm.com/book/en/v2
https://goalkicker.com/GitBook/
https://devdocs.io/git/git-rebase
https://devdocs.io/git/git-merge-base
https://www.atlassian.com/git/tutorials/merging-vs-rebasing
https://phoenixnap.com/kb/how-to-use-git
https://medium.com/@nasreddine.skandrani/absolute-estimation-vs-relative-estimation-be31c14c58aa
https://www.geeksforgeeks.org/git-difference-between-merging-and-rebasing/
https://miro.medium.com/max/855/1*pzT4KMiZDOFsMOKH-cJjfQ.png
https://jeffkreeftmeijer.com/git-rebase/git-rebase.png
https://www.perforce.com/blog/vcs/git-rebase-vs-git-merge-which-better#:~:text=Git%20rebase%20and%20merge%20both%20integrate%20changes%20from.merge%20adds%20a%20new%20commit%2C%20preserving%20the%20history
https://phoenixnap.com/kb/wp-content/uploads/2021/11/git-commands-cheat-sheet-by-pnap-v2.pdf
https://www.gitkraken.com/learn/git/git-flow
