**Git and GitHub**

**GitHub🡪** Just a web site, a “nice add-on”.

**Git 🡪** Handles all the version control stuff.

* Local and Remote Repository:

\*\* **Ej:** 4 people team making a videogame. They are in charge of different things and making changes to the code, sending the changes to each other. They do not use zip files, they use some automated program like GIT. \*\*

**Repository** 🡪 It is a Store, where we have a bunch of code. Can be local or remote.

**Local** 🡪 It sits in your machine, other people do not see it and it is your version of the code.

\*\* Each of the four developers have their own local repository on their machine. The first thing they will do is too look at the **Remote** repository; it is the source of truth. It has all the “up to date” code. They can make changes, push them to the repository and ask for new changes.\*\*

\*\* Person 1 is making **changes**, and **push** them to the **Remote**. The next day tells it to Person 2. He **pulls** the changes and works around them (maybe some files have changed, or there are new files, etc). This way they Push and Pulls. GIT can also merge same files together, this way 1 and 2 can work on the same file in different places.\*\*

* History and Branches:

Tracks a history of **Commits**, a change we made. Save all the changes, kind of a Checkpoint.

I can always come back to that Commit. (At this point of time this was working). It also stores some information.

🡪 **Commits** all my changes into my **local** repository, once I have finished I can take this commits and **push** them up to the remote repository **Branch**.

🡪 I’m on my Local repository, it’s been a few days since I update my repo… Remote has changed… I need to **Pull** all changes to my local, since I last pulled all the changes.

\*\* GITHUB: It tracks all of the commits. We have a Branch (Master Branch). Imagine a timeline with dots in it. Those are different commits, all changes that happened. **This Master Branch is what we wanna make sure is always working.** So all other developers can **Pull** from it without bugs.

Instead of always making changes to the Master Branch we use a different **Branch** (Copy of a repository at a certain point in time). Ej: We create a **new Branch** out of the Master at a point in time to work on a **new Feature**. Once we are **OK** with the new Feature (**it works without bugs**), we can **merge** it to the Master Branch. Here is where GIT does its magic.

**We can have a branch out of a new Branch to work on multiple features at the same time**.

**Local:**

* **Mkdir** namefolder 🡪 Make Directory (new folder)
* **Cd** test\_folder 🡪 CD Change Directory
* **Git init 🡪** Makes the folder a GitHub Repository
* Verificar estar en un Repo antes de comenzar (git commit) Podrá solicitar Email, Usuario o decir que NO es un repo.
* **Git config –global user.name “ “ 🡪** Tomas Padilla
* **Git config –global user.email “ “**

**🡪** [Padillatomasagustin@gmail.com](mailto:Padillatomasagustin@gmail.com)

* **Git config user.name / user.email 🡪** See who is “logged in”

\*\* Ej: Agregamos un archivo cualquiera: readme.md a la folder.

Existe una etapa previa al Commit (**Staging area**: adding or removing files that we want to commit) \*\*

* **Git add** readme.md 🡪 Add to staging area
* **Git status 🡪** Vemos que tenemos

\*\* No commits // Changes to be commited \*\*

* Usually we commit when we have made some changes, some progress, achieved a milestone.
* **Git commit –m “**commit message**”**

\*\* master // root-commit ID // message \*\*

* **Git status 🡪** Para ver si quedó algo en la Staging Area.

\*\* Hacemos algunos cambios solo para probar \*\*

* **Git Status 🡪** Muestra cambios NO staged.
* **Git add . 🡪** El PUNTO = ALL (todos los cambios dentro del repository al Staging Area)

**Branches**

Queremos hacer los cambios en otra branch para no “mess up” la master branch y joder todo. Luego la Branch podrá ser merged a la master.

* **Git checkout –b “**name” 🡪 **-b** : Create a new Branch

\*\* Modificamos el archivo y queremos guardar en la nueva branch

* **Git add . 🡪** Agregamos valores a la staging área
* **Git commit –m “new files on branch”**

\*\* Volvemos a la Master Branch para verificar si hay algun cambio en la Master \*\*

* **Git Checkout master 🡪** Nos cambia a la Master y por supuesto eliminaría los cambios realizados en newbranch”

\*\* Update Master branch con los cambios de la Newbranch \*\*

\*\* Agregamos algún .py a la master branch para probar de merge it con la newbranch\*\*

* **Git merge master 🡪** Merges everything from the Master to the Newbranch

\*\* Vamos a la Newbranch y git merge master \*\*

**ALL THIS WAS LOCAL, NOW LETS TRY A REMOTE ONE (CLOUD)**

**Remote:**

* Vamos a GitHub y log in.
* New repository, name and public.
* Tipeamos la linea que nos recomienda Git Hub.
* **Git remote add origin https:// 🡪** Crea el remote

**🡪 ORIGIN = Https://** “Como si fuera una variable”

**\*\*** Vemos si estamos en Master y la PUSHEAMOS:

* **Git checkout master**
* **Git push –u origin master** 🡪 Save: **-u**

Where: origin

What: Master

\*\* Refresh la website de Git Hub y vemos mis archivos subidos!!!!! \*\*

\*\* Haremos como si alguien modifica mi repository remotamente 🡪 Vamos al github y abrimos el archivo, edit y luego commit. Usaremos luego una **PULL** \*\*

* **Git pull origin master 🡪** where and what.

\*\* Vemos los cambios realizados desde remote, ahora probaremos provocar un “error” de git: Modificar remote e intentar pushear sin haber pulleado dichos cambios:

\*\* Modificamos GitHub desde el mismo website \*\*

* Modificamos y **commit** de Git e intentamos **PUSH** a remote. 🡪 ERROR ¡!! REJECTED

\*\*Intentamos PULL de Origin a Master \*\*

* **Git pull origin master🡪** Error cannot merge (porque hemos modificado la misma línea en Remote y Local, entonces git no sabe cuál guardar…. Debemos indirale manualmente como proceder)

\*\*Me muestra que **VIENE** y que **VA**, debemos decidir que hacer: Borramos todo y decidimos dejarlo como queremos, grabamos, **add . , commit, push. \*\***

\*\* Esto es un error común con varios archivos a la vez \*\*

**Pusheamos otra branch:**

* **Git checkout newbranch**
* **Git push origin newbranch 🡪** Especificamos la newbranch

\*\* Nos aparece un cartel de “**compare & pull request**” 🡪 Porque comúnmente usamos las branches for merging them with the master branch once they are ready.

Tocamos el boton y cambiamos la “Pull request” por “Draft Pull Request” y vemos debajo el “historial”

🡪 Azul: Agregado // Rojo: Eliminado

**ENTONCES: USAMOS GIT PARA CREAR BRANCHES, COMPARAMOS CON LA MASTER. PODEMOS HACER PULLS Y PUSHS PARA TENER TODO AL DIA.**