

Mis apuntes de las clases de Git y Github

Jhamil Arnez Hidalgo

Última actualización: May 8, 2025

1 ¿Qué es Git?

Git es un sistema de control de versiones distribuido que permite a los desarrolladores llevar un registro de los cambios en sus archivos y coordinar el trabajo en proyectos de software de manera eficiente.

Distribuido quiere decir que no depende de un único sitio en específico para almacenar su código. Existen muchos controladores de versiones, pero no todos son distribuidos, eso significa que por ejemplo si el servidor donde tienen almacenado el código llega a reventar, entonces perderían todo el código.

En cambio, un sistema distribuido permite tener a todos los colaboradores una copia del proyecto en sus propios equipos, y en caso de que el servidor principal reventase, el proyecto seguiría existiendo en las máquinas personales de los desarrolladores y se podría recuperar.

2 Comandos de Git

2.1 Git Init

Para empezar a usar git, primero se necesita un repositorio local, es decir una carpeta, sobre la cual git va a ir controlando y guardando sus versiones y revisando sus cambios.

Una vez creada la carpeta, desde la terminal hay que posicionarse sobre esa carpeta (en bash con `cd /directorio`).

Una vez hecho esto en la terminal se va a escribir, **git init** y esto automáticamente creará una carpeta oculta llamada `".git"` en la carpeta.

```
~/Desktop/Repo (0.053s)
git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:     git config --global init.defaultBranch <name>
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:     git branch -m <name>
Initialized empty Git repository in /home/jhamux/Desktop/Repo/.git/

Help me start a new project Ctrl Shift ⌘
~/Desktop/Repo git:(master) Pair Ctrl ⌘ Dispatch Beta Ctrl Shift ⌘
git status
```

Figure 1: Git Init ejecutado en la terminal

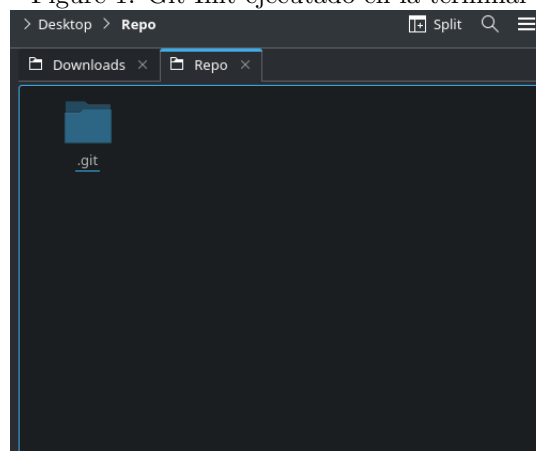


Figure 2: Carpeta Oculta .git en el repositorio

2.2 Git Add y Git Commit

Ahora que git está monitoreando el repositorio podemos crear un fichero de lo que gustemos y git lo detectará, pero no lo guardará automáticamente, es importante entender los 3 estados en los que puede estar un fichero en un repositorio con git.

1. **modified:** El archivo tiene cambios pero aún no fueron marcados para ser confirmados, se encuentra en el directorio de trabajo.
2. **staged:** Son los archivos que fueron modificados y confirmados para guardarse en el repositorio local, se encuentran en un área temporal transitoria.
3. **committed:** El archivo se encuentra grabado en el repositorio local.

Cuando creamos un archivo nuevo, git no lo reconocerá automáticamente, aparecerá como "untracked" para que git empiece a hacerle seguimiento hay que ejecutar el comando: **git add nombre-del-archivo**, y así pasará al estado de **staged**.

Una vez que todos los archivos se encuentren en el estado staged, estarán listos para que se les pueda realizar un commit, básicamente les estamos sacando una foto en el momento actual a todos los archivos, guardando el momento. Esto se hace con el comando **git commit**, este comando nos abrirá un editor de código para colocar un nombre al commit, una vez colocado el nombre el commit ya queda realizado.

2.3 Git Log y Git Status

Estos comandos son para observar el estado de tus commits y de tu repositorio respectivamente.

El comando **git log**, nos muestra todos los commits existentes en una lista, donde nos muestra el nombre de los commits, la hora y fecha de creación, el código hash de los commits, y el autor de este mismo.

También existen diferentes formas de visualizarlo, una bastante práctica es escribir el comando: **git log --oneline** donde te muestra todos los commits cada uno en una línea, haciendo más práctico su visualización.

```
commit 006394b6dd3e595a07cbad17189c3505bb4779df (HEAD -> master, origin/master)
Author: Jhamil <arnezhidalgojj@gmail.com>
Date: Thu May 8 14:45:47 2025 -0400

    fix: change name of directory

commit 4e7722586230ee0e792b0198d0c14964be6990f9
Author: Jhamil <arnezhidalgojj@gmail.com>
Date: Thu May 8 14:44:09 2025 -0400

    feat: add git add and git commit to notes

commit 200d783e0756ed82059829dd3a86b1bb124e0bc2
Merge: e4abcc7 48ed5cc
Author: Jhamil <arnezhidalgojj@gmail.com>
Date: Thu May 8 13:30:36 2025 -0400

    Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository

commit e4abcc7852e9311b9bd23a0c4216974f3906fec1
Author: Jhamil <arnezhidalgojj@gmail.com>
Date: Thu May 8 13:20:29 2025 -0400

    feat: remove hola.py add images to notes
```

Figure 3: Git Log en la consola

```

jhamux@debian:~/Desktop/My Cool Repo$ git log --oneline
006394b (HEAD -> master, origin/master) fix: change name of directory
4e77225 feat: add git add and git commit to notes
200d783 Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository
e4abcc7 feat: remove hola.py add images to notes
48ed5cc remove hola.py
6d3b0bb feat: add course notebook in latex
47393a2 feat: add task list
aecc816 Cambio de texto en readme, y agregación de dos objetos al .py
af08a8d Un ejercicio de Programación Orientada a Objetos
1d247eb Creación del Archivo Readme.md
jhamux@debian:~/Desktop/My Cool Repo$

```

Figure 4: Git Log in one line en la consola

El comando **git status**, nos muestra la lista de todos los archivos que estén en estado staged, modified, y si es nuevo, untracked. Así como también nos muestra si no hay novedades en el repositorio.

```

jhamux@debian:~/Desktop/My Cool Repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
jhamux@debian:~/Desktop/My Cool Repo$

```

Figure 5: Git Status en la consola

3 Alias

Los alias son formas de nombrar una serie de comandos en una sola palabra, como si se tratase de un atajo de teclado, nosotros podemos crear un alias para una sucesión de códigos de git larga y para ejecutarla solo redactando una palabra.

```

jhamux@debian:~/Desktop/My Cool Repo$ git log --graph --decorate --all --oneline
* 3fb187f (HEAD -> master, origin/master) Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository
|
| * bc7bc96 fix: remove directory notes
| * b52844f fix: fix right latex format
|/
* c59b737 feat: add git log and git status in notes
* 006394b fix: change name of directory
* 4e77225 feat: add git add and git commit to notes
* 200d783 Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository
|
| * 48ed5cc remove hola.py
| * e4abcc7 feat: remove hola.py add images to notes
|/
* 6d3b0bb feat: add course notebook in latex
* 47393a2 feat: add task list
* aecc816 Cambio de texto en readme, y agregación de dos objetos al .py
* af08a8d Un ejercicio de Programación Orientada a Objetos
* 1d247eb Creación del Archivo Readme.md
jhamux@debian:~/Desktop/My Cool Repo$ git config --global alias.log "log --graph --decorate --all --oneline"
jhamux@debian:~/Desktop/My Cool Repo$

```

Figure 6: Creamos un alias para git log graph decorate all y oneline

```

jhamux@debian:~/Desktop/My Cool Repo$ git tree
* 3fb187f (HEAD -> master, origin/master) Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository
|
| * bc7bc96 fix: remove directory notes
| * b52844f fix: fix right latex format
|/
* c59b737 feat: add git log and git status in notes
* 006394b fix: change name of directory
* 4e77225 feat: add git add and git commit to notes
* 200d783 Merge branch 'master' of https://github.com/Jhamil75029/My-Cool-Repository
|
| * 48ed5cc remove hola.py
| * e4abcc7 feat: remove hola.py add images to notes
|/
* 6d3b0bb feat: add course notebook in latex
* 47393a2 feat: add task list
* aecc816 Cambio de texto en readme, y agregación de dos objetos al .py
* af08a8d Un ejercicio de Programación Orientada a Objetos
* 1d247eb Creación del Archivo Readme.md
jhamux@debian:~/Desktop/My Cool Repo$

```

Figure 7: Usamos el alias Tree

4 Git Checkout y Git Reset

git checkout es un comando que tiene 2 usos principales: Restaurar archivos a la versión de su último commit (**git checkout nombre-de-archivo**), y también sirve para cambiar de ramas con (**git checkout nombre-de-tu-rama**), pero desde la versión 2.23 de git se recomienda utilizar en su lugar el comando **git switch** para cambiar de rama.

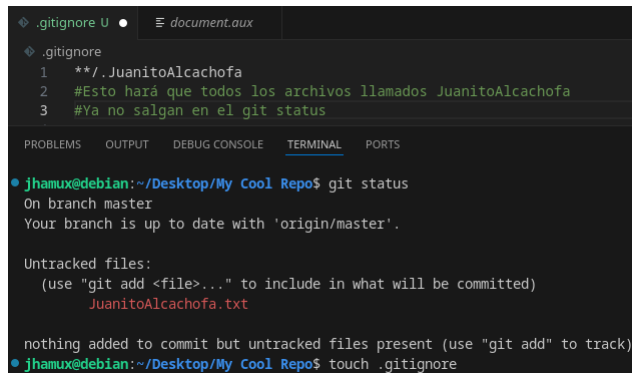
git reset es un comando que tiene unas 3 formas de usarse: soft, mixed y hard:

1. (**git reset --soft HEAD~1**) Mueve el puntero de la rama (HEAD) a un commit anterior, pero conserva tus archivos y el stage.
2. (**git reset HEAD~1**) Quita el commit y saca los archivos del stage, pero mantiene los cambios en el directorio.
3. (**git reset --hard HEAD~1**) Elimina el commit y borra todos los cambios del working directory y el stage.

5 Git Ignore

Se utiliza para ignorar archivos que sabemos que nunca vamos a querer que entren en ningún commit, pueden ser archivos temporales, o archivos de apoyo indirecto al proyecto, pero sin vital relevancia.

Para ejecutarlo se usa el comando **touch .gitignore**, donde se crea un archivo vacío, en el cual los ficheros que añadamos dentro no serán tomados en cuenta para los futuros commit.



The image shows a VS Code editor interface. At the top, there's a file explorer showing a file named `.gitignore`. The file content is as follows:

```
1  **/.JuanitoAlcachofa
2  #Esto hará que todos los archivos llamados JuanitoAlcachofa
3  #Ya no salgan en el git status
```

Below the file explorer, there's a terminal window with the following output:

```
jhamux@debian:~/Desktop/My Cool Repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  JuanitoAlcachofa.txt

nothing added to commit but untracked files present (use "git add" to track)
jhamux@debian:~/Desktop/My Cool Repo$ touch .gitignore
```

Figure 8: Usando el Git Ignore

En este caso creé un archivo de texto llamado `JuanitoAlcachofa` en el repositorio, coloqué su nombre en el `gitignore`, luego de eso quedan un par de pasos extra.

Ahora al abrir el `git status` no aparecerá `JuanitoAlcachofa` sino que saldrá el archivo `.gitignore`, a este hay que hacerle un commit y una vez hecho esto, `JuanitoAlcachofa` no volverá a salir.

6 Git Merge

Este es uno de los comandos más importantes después del commit, ya que esto ayuda a fusionar ramas, las ramas las detallaré más adelante. El (**git merge tu-rama**), fusiona la rama que estás llamando a la rama actual en la que estás, y esto es muy útil al trabajar con repositorios remotos, ya que primero programas en tu repositorio local, luego con `git push`(ya lo veremos) se sube al repositorio remoto y ahí realizamos un merge con el commit que acabamos de subir, y el que ya estaba allí en el repositorio remoto.

7 Git Push y Git Pull

El comando `git push` como estaba mencionando anteriormente se utiliza para empujar un commit al repositorio remoto, se usa de esta manera: **git push origin nombre-de-tu-rama**, donde `origin` es una palabra clave que representa el link de enlace al repositorio remoto, y se debe configurar antes de usarse.

El comando `git pull`, hace lo opuesto, este comando jala los commit del repositorio remoto al repositorio local y se usa así: **git pull origin nombre-de-tu-rama**.