

COMANDOS BÁSICOS DE GIT

Git es una parte importante de quien programa a diario, especialmente si trabaja con un equipo, es una herramienta extensamente usada en la industria de software.

Existe una gran variedad de comandos que podemos utilizar, pero algunos comandos se utilizan más frecuentemente, por lo tanto, vamos a explicar los comandos básicos de Git que todos debemos conocer:

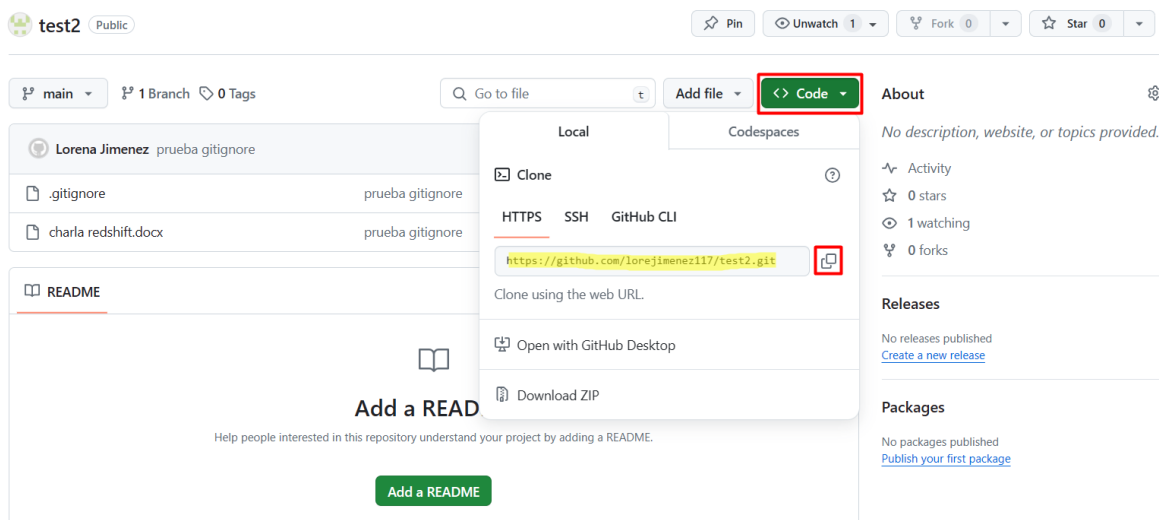
1. Git clone

Git clone es un comando para descargar el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en nuestro computador.

Una de las formas de descargar el código fuente es con https, así:

`git clone https://link-con-nombre-del-repositorio`

Por ejemplo, si queremos descargar un proyecto desde Github, todo lo que necesitamos es hacer clic sobre el botón verde <Code>, copiar la URL de la caja y pegarla después del comando git:



Ejemplo: `git clone https://github.com/lorejimenez117/test2.git`

2. Git branch

Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

Crear una nueva rama:

```
git branch <nombre-de-la-rama>
```

Este comando creará una rama en local. Para enviar (push) la nueva rama al repositorio remoto, necesitamos usar el siguiente comando:

```
git push <nombre-remoto> <nombre-rama>
```

Visualización de ramas:

```
git branch  
git branch --list
```

Borrar una rama:

```
git branch -d <nombre-de-la-rama>
```

3. Git checkout

Este es también uno de los comandos más utilizados en Git. Para trabajar en una rama, primero tenemos que cambiarnos a ella. Usamos git checkout principalmente para cambiarnos de una rama a otra. También lo podemos usar para chequear archivos y commits.

```
git checkout <nombre-de-la-rama>
```

Hay algunos pasos que debemos seguir para cambiar exitosamente entre ramas:

- Los cambios en la rama actual tienen que ser confirmados o almacenados en el guardado rápido (stash) antes de que cambiemos de rama.
- La rama a la que nos queremos cambiar debe existir en local.

También hay un comando de acceso directo que nos permite crear y cambiarnos a esa rama al mismo tiempo:

```
git checkout -b <nombre-de-tu-rama>
```

Este comando crea una nueva rama en local (-b viene de rama (branch)) y nos cambia a la rama que acabamos de crear.

4. Git status

El comando de git status nos da toda la información necesaria sobre la rama actual. Podemos encontrar información como:

- Si la rama actual está actualizada.
- Si hay algo para confirmar, enviar o recibir (pull).
- Si hay archivos en preparación (staged), sin preparación(unstaged) o que no están recibiendo seguimiento (untracked)
- Si hay archivos creados, modificados o eliminados

```
D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git add file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
```

git status nos da información acerca del archivo y las ramas

5. Git add

Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

Necesitamos usar el comando git add para incluir los cambios del o de los archivos en el siguiente commit.

Añadir un único archivo:

```
git add <archivo>
```

Añadir todo de una vez:

```
git add .
```

Si revisas la imagen del comando 4, verás que hay nombres de archivos en rojo, esto significa que los archivos están sin seguimiento, es decir, no serán incluidos en los commits hasta que no los añadamos.

Para añadirlos, necesitamos usar el git add:

```
D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git add file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
```

Los archivos en verde han sido añadidos a la preparación gracias al git add

Importante: El comando git add no cambia el repositorio y los cambios que no han sido guardados, hasta que no utilicemos el comando de confirmación git commit.

6. Git commit

Este es quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios, quizás después de una tarea o asunto específico.

Git commit es como establecer un punto de control en el proceso de desarrollo al cual podemos volver más tarde si es necesario.

También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente.

git commit -m "mensaje de confirmación"

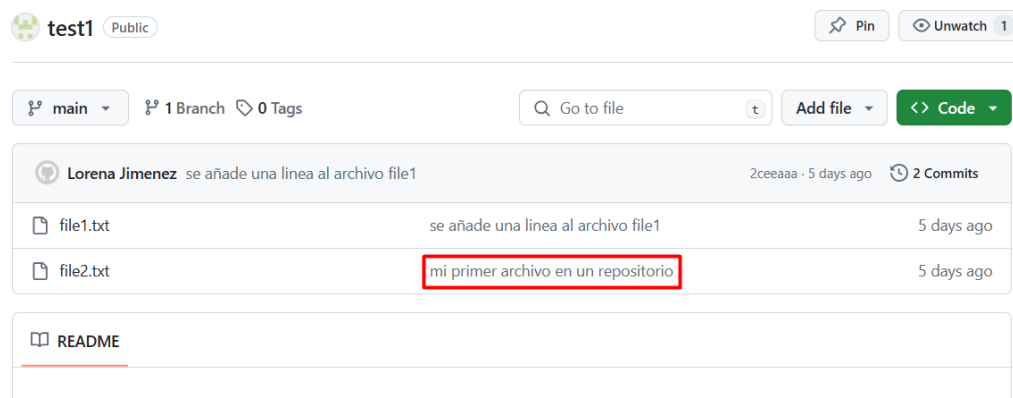
```
D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git commit -m "mi primer archivo en un repositorio"
[main (root-commit) 0b197f2] mi primer archivo en un repositorio
 2 files changed, 2 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt

D:\Bootcamp cloud - Betek\Git Repositories\test1>git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)
```



The screenshot shows a GitHub repository interface for 'test1' (Public). It displays the commit history with two commits. The first commit by Lorena Jimenez added 'file1.txt'. The second commit, also by Lorena Jimenez, added 'file2.txt' and has the message 'mi primer archivo en un repositorio' highlighted with a red box. The repository has 1 branch (main) and 0 tags.

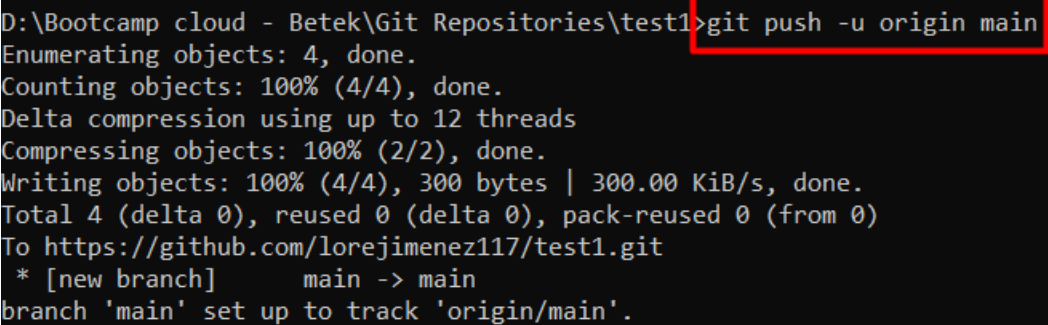
Commit	Author	Message	Files Changed	Time
2ceaaaa	Lorena Jimenez	se añade una linea al archivo file1	file1.txt	5 days ago
0b197f2	Lorena Jimenez	mi primer archivo en un repositorio	file1.txt, file2.txt	5 days ago

Importante: Git commit guarda los cambios únicamente en local.

7. Git push

Después de haber confirmado los cambios, el siguiente paso que queremos dar es enviarlos al servidor remoto. Git push envía los commits al repositorio remoto.

`git push -u origin <nombre-de-tu-rama>`



```
D:\Bootcamp cloud - Betek\Git Repositories\test1>git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 300 bytes | 300.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/lorejimenez117/test1.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Importante: Git push solamente carga los cambios que han sido confirmados.

8. Git pull

El comando git pull se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge, lo cual significa que cuando usamos el git pull recogemos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

`git pull <nombre-remoto>`

Esta operación puede generar conflictos que tengamos que resolver manualmente.

9. Git revert

A veces, necesitaremos deshacer los cambios que hemos hecho. Hay varias maneras para deshacer nuestros cambios en local y/o en remoto (dependiendo de lo que necesitemos), pero necesitaremos utilizar cuidadosamente estos comandos para evitar borrados no deseados.

Una manera segura para deshacer nuestras commits es utilizar git revert.

Para ver nuestro historial de commits, primero necesitamos utilizar el git log:

```
D:\Bootcamp cloud - Betek\Git Repositories\test2>git log
commit cffd21a518d76eb806484c6afb4933eb4e725e0c (HEAD -> main, origin/main, origin/HEAD)
Author: Lorena Jimenez <lorejimenez11@gmail.com>
Date:   Wed May 8 18:07:40 2024 -0500

    prueba gitignore

commit 8d1f7a5140a5fc684716dfd6d1ac1e85a2aa57f9
Author: lorejimenez117 <lorejimenez117@gmail.com>
Date:   Wed May 8 18:00:27 2024 -0500

    Initial commit

D:\Bootcamp cloud - Betek\Git Repositories\test2>
```

Entonces, solo necesitamos especificar el código de comprobación que se encuentra junto al commit que queremos deshacer:

```
git revert cffd21a518d76eb806484c6afb4933eb4e725e0c
```

Después de esto, vemos una pantalla como la de abajo, digitamos :qa! y presionamos Enter para salir:

```
Administrator: Símbolo del sistema - git revert cffd21a518d76eb806484c6afb4933eb4e725e0c
Revert "prueba gitignore"

This reverts commit cffd21a518d76eb806484c6afb4933eb4e725e0c.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is up to date with 'origin/main'.
#
# Changes to be committed:
#   modified:   .gitignore
#   deleted:    charla redshift.docx
#
# Untracked files:
#   file1.txt
#   file2.txt
#
~
```

El comando git revert deshazá el commit que le hemos indicado, pero creará un nuevo commit deshaciendo el anterior:

```
D:\Bootcamp cloud - Betek\Git Repositories\test2>git log
commit c3f4235c309ca66d3e3fc3f86a790ad9faa43ce6 (HEAD -> main)
Author: Lorena Jimenez <lorejimenez11@gmail.com>
Date: Mon May 13 20:17:33 2024 -0500

    Revert "prueba gitignore"

This reverts commit cffd21a518d76eb806484c6afb4933eb4e725e0c.

commit cffd21a518d76eb806484c6afb4933eb4e725e0c (origin/main, origin/HEAD)
Author: Lorena Jimenez <lorejimenez11@gmail.com>
Date: Wed May 8 18:07:40 2024 -0500

    prueba gitignore

commit 8d1f7a5140a5fc684716dfd6d1ac1e85a2aa57f9
Author: lorejimenez117 <lorejimenez117@gmail.com>
Date: Wed May 8 18:00:27 2024 -0500

    Initial commit

D:\Bootcamp cloud - Betek\Git Repositories\test2>
```

La ventaja de utilizar git revert es que no afecta al commit histórico. Esto significa que podemos seguir viendo todos los commits en el histórico, incluso los revertidos.

Otra medida de seguridad es que todo sucede en local a no ser que los enviemos al repositorio remoto. Por esto es que git revert es más seguro de usar y es la manera preferida para deshacer los commits.

10. Git merge

Cuando ya hayamos completado el desarrollo del proyecto en nuestra rama y todo funcione correctamente, el último paso es fusionar la rama con su rama padre (dev o master). Esto se hace con el comando git merge.

Git merge básicamente integra las características de nuestra rama con todos los commits realizados a las ramas dev o master. Es importante recordar que tenemos que estar en esa rama específica que queremos fusionar con nuestra rama de características.

Por ejemplo, cuando queremos fusionar nuestra rama de características en la rama dev:

Primero, debemos cambiarnos a la rama dev:

```
git checkout dev
```

Antes de fusionar, debemos actualizar nuestra rama dev local:

```
git fetch
```

Por último, podemos fusionar nuestra rama de características en la rama dev:

```
git merge <nombre-de-la-rama>
```