

# Índice

## [GIT - Primer Round: Comandos básicos](#)

[Clone](#)

[Status](#)

[Add - Commit](#)

[Push](#)

[Pull](#)

[Diff](#)

[Log](#)

## [Git - Round 2: Trabajando con otros!](#)

## GIT - Primer Round: Comandos básicos

Primero, creamos un repo en GitHub:

<https://guides.github.com/activities/hello-world/#repository>

### Clone

Después, lo clonamos! Es decir, generamos nuestra copia local:

`git clone https://github.com/ClaraAllende/practica-git-ada.git`

```
gondor:Desktop clara.allende$ git clone https://github.com/ClaraAllende/practica-git-ada.git
Cloning into 'practica-git-ada'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Estos dos pasos los hacen UNA SOLA VEZ, de hecho si trabajan sobre un repositorio ya creado, solo van a tener que hacer el clone.

Para poder ejecutar comandos de git, necesitamos estar en la carpeta que nos acaba de generar el clone. En la consola, para movernos de carpetas adentro de un directorio, usamos el comando `cd`

`cd <nombre de la carpeta>` → moverse a la carpeta  
ej: `cd practica-git-ada`

Si queremos ver el contenido de la carpeta en la que estamos, usamos el comando `ls`  
`ls` → listar los archivos adentro de una carpeta

```
gondor:Desktop clara.allende$ cd practica-git-ada/  
gondor:practica-git-ada clara.allende$ ls  
README.md
```

Nota: `cd` y `ls` son comandos de la consola, no de git (o sea que lo pueden usar en cualquier lado, no solo en carpetas repositorios de git)

## Status

Creamos un archivo, “adas-backend”, escribimos nuestro nombre, y lo guardamos. Luego queremos ver el estado de nuestra copia:

`git status` → muestra los cambios que tenemos en nuestra copia local

```
gondor:practica-git-ada clara.allende$ nano adas-backend  
gondor:practica-git-ada clara.allende$ ls  
README.md      adas-backend  
gondor:practica-git-ada clara.allende$ 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)  
  
      adas-backend  
  
[nothing added to commit but untracked files present (use "git add" to track)]
```

## Add - Commit

Los archivos sin versionar están en rojo ;)

Y ahora queremos agregar el archivo y armar la “foto” de los cambios, es decir, la nueva versión:

`git add <nombre del archivo>` → agregar un archivo en específico para versionar

`git add .` → agregar todos los archivos cambiados a la vez

```
gondor:practica-git-ada clara.allende$ git add adas-backend
gondor:practica-git-ada clara.allende$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   adas-backend
```

Ahora los archivos listos para versionar están en verde.

`git commit` → armar el commit (la foto de los cambios) para subir al repo.

Esto abre un diálogo para escribir el mensaje. El editor por defecto es [vim](#), que para guardar los cambios después de escribir el mensaje de commit, necesitan apretar el botón Esc y luego tipear `:wq!` (si, con el signo de exclamación).

Acá les dejo un [cheat-sheet](#) de comandos de vim (no es obligatorio que lo sepan, el editor se puede cambiar).

La alternativa para no tener que pelearse con vim, es:

```
git commit -m "aca va el mensaje"
```

Git nos obliga a poner un mensaje de commit, porque como está pensado para compartir código, está bueno que la versión que generamos esté acompañada de un mensaje que describa mínimamente qué cambios se introdujeron.

```
Aborting commit due to empty commit message.
gondor:practica-git-ada clara.allende$ git commit
[master efb20f3] Primer commit de la listas de ada-backend
 1 file changed, 1 insertion(+)
 create mode 100644 adas-backend
```

Si no ponen mensaje de commit, Git no les va a subir nada :O

```
gondor:practica-git-ada clara.allende$ git commit
Aborting commit due to empty commit message.
```

Acá va la versión con `git add .` y `git commit -m`:

```
gondor:practica-git-ada clara.allende$ git add .
gondor:practica-git-ada clara.allende$ git commit -m "Primer commit de la lista de Adas-Backend"
[master a8d300f] Primer commit de la lista de Adas-Backend
 1 file changed, 1 insertion(+)
 create mode 100644 adas-backend
```

**La primera vez** que intenten hacer commit, Git pide (literal) que le digamos quienes somos. Para eso:

```
git config --global user.email "el mail con el que se registraron"
git config --global user.name "su nombre"
```

Esto solo lo tienen que hacer una vez, después ya queda configurado. :)

## Push

Ahora si, queremos subir nuestro cambio al repositorio remoto!

```
git push <rama remota a donde subo el commit> <rama equivalente
dentro del remote, que se va a actualizar>
ej: git push origin master
```

Quiere decir que git va a buscar una rama `master` en el repositorio local y actualizar esa misma rama en `origin`.

Si la copia local está actualizada respecto al remoto, se ve así:

```
gondor:practica-git-ada clara.allende$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:ClaraAllende/practica-git-ada.git
 6400da6..a8d300f master -> master
```

Si no... Git nos avisa que tenemos fotos distintas, y nos recomienda actualizarnos antes de tratar de subir nuestros cambios:

```
gondor:practica-git-ada clara.allende$ git push origin master
To github.com:ClaraAllende/practica-git-ada.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:ClaraAllende/practica-git-ada.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

## Pull

Para actualizar nuestra copia local con los cambios del remoto:

```
git pull origin master
```

Si no modificamos la misma línea en el mismo archivo, git se encarga solito de “apilar” correctamente los commits, en función de la fecha y hora en la que se crearon. A eso le dice “fast-forward”

```

gondor:practica-git-ada clara.allende$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:ClaraAllende/practica-git-ada
 * branch                master      -> FETCH_HEAD
   a8d300f..d3f53a2 master      -> origin/master
Updating a8d300f..d3f53a2
Fast-forward
 adas-backend | 1 +
 1 file changed, 1 insertion(+)

```

Si no... nos dice "CONFLICT" y nos va a pedir que resolvamos nosotros como quedan los cambios:

```

gondor:practica-git-ada clara.allende$ git pull origin master
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 15 (delta 0), reused 15 (delta 0), pack-reused 0
Unpacking objects: 100% (15/15), done.
From github.com:ClaraAllende/practica-git-ada
 * branch                master      -> FETCH_HEAD
   66dc394..2d68a45 master      -> origin/master
Auto-merging adas-backend
CONFLICT (content): Merge conflict in adas-backend
Automatic merge failed; fix conflicts and then commit the result.

```

Para eso, el orden que yo recomiendo, es:

git status → para ver qué archivos tienen conflictos

```

gondor:practica-git-ada clara.allende$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 5 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   adas-backend

no changes added to commit (use "git add" and/or "git commit -a")

```



## Diff

`git diff` → para ver exactamente que diferencias hay entre el remoto y mi copia local. Este comando muestra qué es lo que cambió en un archivo de un commit a otro.

```
gondor:practica-git-ada clara.allende$ git diff
diff --cc adas-backend
index 737c18c,55fd87a..0000000
--- a/adas-backend
+++ b/adas-backend
@@@ -1,5 -1,10 +1,14 @@@
  Clari
  Pilar
  Viviana
+ =====
+ Dani
+ Abril
+ =====
+ Viviana
  Ana
- FlorG
++<<<<<< HEAD
+Clari otra vez
+=====
++FlorG
++>>>>>> 2d68a45f07fef2d857f57459d117d2dfc23f0eb2
```

En verde ven lo que se agrega, en rojo lo que se borra. Donde dice HEAD, es donde apunta el último commit de nuestra copia local, lo otro es lo que viene del remoto.

Una vez que editaron el archivo y resolvieron el conflicto, hay que hacer otra vez:

```
git add <nombre archivo>
git commit -m "mensaje de commit descriptivo"
git push origin master
```

```
gondor:practica-git-ada clara.allende$ git add .
gondor:practica-git-ada clara.allende$ git commit -m "arregle los conflictos de nuevo"
[master 9985522] arregle los conflictos de nuevo
gondor:practica-git-ada clara.allende$ git push origin master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 1000 bytes | 1000.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To github.com:ClaraAllende/practica-git-ada.git
 52b4958..9985522 master -> master
```

## Log

Para ver el listado de los commits, quien los hizo, la fecha, y el identificador del commit:

`git log` → listar los commits

```
gondor:practica-git-ada clara.allende$ git log
commit 998552215aca959cddd2709c613cbfad2cb4d33 (HEAD -> master, origin/master, origin/HEAD)
Merge: 09f49f7 52b4958
Author: Clara Allende <callende@despegar.com>
Date: Wed Mar 27 22:00:49 2019 -0300

    arregle los conflictos de nuevo

commit 09f49f7e11b721927b32b86042949a622cf91fc4
Merge: 05aa296 2d68a45
Author: Clara Allende <callende@despegar.com>
Date: Wed Mar 27 21:59:21 2019 -0300

    arreglé los conflictos

commit 52b49584bcebe85aa0a857d25e5083af009583f5
Merge: 4e64ef2 e476a68
Author: Romina <rominaglzej@gmail.com>
Date: Wed Mar 27 21:55:07 2019 -0300

    nuevo commit romina

commit e476a68102dc4508ccfd7b8dc24371c4e3fdc9e4
Merge: 0cf5f54 2d68a45
Author: dianaagustinaf <dianaagustinaf@gmail.com>
Date: Wed Mar 27 21:56:26 2019 -0300

    diana add

commit 05aa29600e35d7c61905e3de13b953120979a2dd
Author: Clara Allende <callende@despegar.com>
Date: Wed Mar 27 21:50:39 2019 -0300

    comiteo de nuevo

commit 2d68a45f07fef2d857f57459d117d2dfc23f0eb2
Merge: 0cb5076 5466b15
Author: Flor <florxgomez@gmail.com>
Date: Wed Mar 27 21:50:26 2019 -0300

    mi nombre
```

Recursos útiles sobre git/github (la mayoría están en inglés, si necesitan una mano con el idioma, avisen que los podemos traducir)

<https://guides.github.com/>

<https://github.com/AlexZeitler/gitcheatsheet/blob/master/gitcheatsheet.jpg>

<https://guides.github.com/introduction/git-handbook/>

<https://es.atlassian.com/git>

<https://www.youtube.com/githubguides> → tutoriales en video!

y pueden usar `git help <nombre del comando>` donde está la explicación de que hace, como se usa, e incluso ejemplos :)

## Git - Round 2: Trabajando con otros!

Con la práctica anterior comprobamos que cuando trabajan muchas personas sobre los mismos archivos al mismo tiempo es receta para el caos.

Entonces, aprendimos a usar branches:

```
----- master → el tronco del árbol (de commits)
      \----- mi branch → una rama
```

Las ramas o branches nos permiten poder hacer cambios, y subirlos al remote, sin interferir en la historia común (que es lo que está en la rama de la cuál se crea el branch).

Las usamos MUCHO para poder trabajar cada quien en una funcionalidad específica, sin ser afectados por los cambios de otros ni afectar a otros con nuestros cambios.

Comandos útiles para trabajar con branches:

- Listar los branches que tengo en la copia local: `git branch` (a secas)

```
gondor:practica-git-ada clara.allende$ git branch
clari
* master
```

En verde aparece el branch en el que estamos actualmente.

- Moverse a otro branch: `git checkout <branch al que me quiero mover>`

```
gondor:practica-git-ada clara.allende$ git checkout clari
Switched to branch 'clari'
```

- Crear un branch: vimos dos opciones:

- `git branch "nombreDelNuevoBranch"`

```
gondor:practica-git-ada clara.allende$ git branch clari
```

Esta opción solo lo crea...

- `git checkout -b "nombre del branch"`

```
gondor:practica-git-ada clara.allende$ git checkout -b otraRama
Switched to a new branch 'otraRama'
```

Esta opción crea el branch, y nos mueve a ese nuevo branch.

Cosas que nos pasan cuando trabajamos con branches:

- Normalmente queremos subir nuestros cambios al remote, pero en el branch sobre que estamos trabajando: acá hablamos que era importante respetar la sintaxis del comando `push` que habíamos usado antes:

`git push origin master` → actualiza la rama master del remote origin

`git push origin clari` → actualiza la rama clari del remote origin

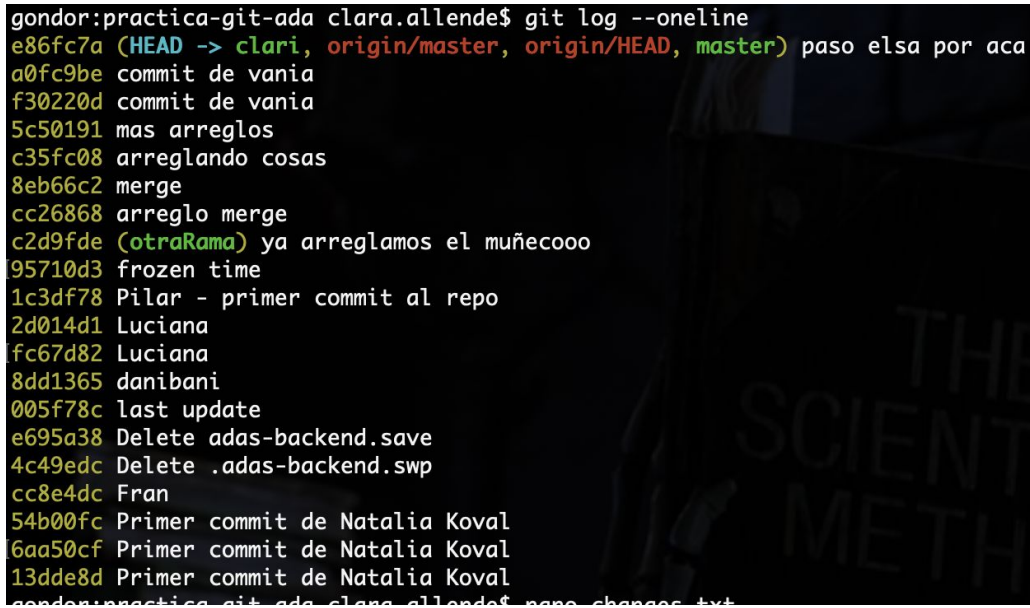
Si la rama no existe en el remote, git la crea.

Así que hay que prestar atención a dónde queremos subir los cambios, para no afectar la historia de los otros branches.



- Normalmente queremos también actualizar nuestro branch con los cambios del branch “madre”, del tronco digamos. Acá también hablamos de respetar la sintaxis de `pull` de la que hablamos antes, similar a lo dicho para `push`:  
`git pull origin master` → actualizar el branch EN EL QUE ESTOY con los cambios de `origin/master`  
`git pull origin clari` → actualizar el branch EN EL QUE ESTOY con los cambios de `origin/clari`

Cuando usamos branches, la historia del árbol de commits se bifurca:



```
gondor:practica-git-ada clara.allende$ git log --oneline
e86fc7a (HEAD -> clari, origin/master, origin/HEAD, master) paso elsa por aca
a0fc9be commit de vania
f30220d commit de vania
5c50191 mas arreglos
c35fc08 arreglando cosas
8eb66c2 merge
cc26868 arreglo merge
c2d9fde (otraRama) ya arreglamos el muñecooo
95710d3 frozen time
1c3df78 Pilar - primer commit al repo
2d014d1 Luciana
fc67d82 Luciana
8dd1365 danibani
005f78c last update
e695a38 Delete adas-backend.save
4c49edc Delete .adas-backend.swp
cc8e4dc Fran
54b00fc Primer commit de Natalia Koval
6aa50cf Primer commit de Natalia Koval
13dde8d Primer commit de Natalia Koval
gondor:practica-git-ada clara.allende$ nano changes.txt
```

En este caso `clari` y `master` están en el último commit, y `otraRama` quedó bastante atrás.  
Gráficamente:

```
master      -----
clari       \-----/
otraRama    \-----
```

Entonces, cuando actualizamos `clari` con lo que había en `master`, hicimos que las ramas que se habían separado vuelvan a unirse, es lo que llamamos **merge**. La manera más fácil de hacer un merge es usando `pull` como hasta ahora, aunque no es la única forma.\*

\* Pueden investigar `git fetch` y `git merge`, si están con mucha curiosidad :)