

 Git intermedio

Mitsiu Alejandro Carreño Sarabia

*The popular approach to version control*



# Memorizing Six Git Commands

*A Practical Guide*

O RLY?

@ThePracticalDev

- 
- Intro
    - ¿Dónde está el mouse?
    - ¿Qué significa el comando?
  - Config
    - Configurar su editor de textos
    - PS1
    - user.name & user.email
    - Autenticación SSH
  - Basics
    - Git workflow (Read your outputs)
    - Git for dummies (Cheatsheet)
  - Malabares locales
    - git branch (Cheatsheet)
    - Stashing
  - Malabares remotos
    - Cherry-picking
  - F\*ck ups
    - F\*ck up sencillo; mensaje de commit (--amend)
    - F\*ck up sencillo; Deshacer último commit (reset HEAD~)
    - F\*ck up; difícil; Subir a rama incorrecta (cherry-pick sin delete)
    - F\*ck up; difícil; Multiples errores en multiples commits (rebase -i)
    - Mega f\*ck up; crítico; Subir un PR mal (revert -m 1 SHA1)
    - Mega f\*ck up; F\*ck up en el fix de otro F\*ck up (reflog)
    - Who f\*ck up?
  - Pt. 2?
    - git add -p
    - Origins
    - Submodules
    - git rebase vs git merge (Qué/Cómo/Cuándo)



## Intro

### Convenciones - Algunos slides dicen EJERCICIO 0

En la sección superior está marcado cuando hay un ejercicio relacionado al slide:

- Pueden clonar el repo pero no van a poder hacer push
- Pueden hacer fork y hacer push a su copia del repo

Bloques de código, con highlight:

```
$ git log --oneline # $ == comandos
3cf53b0 (HEAD -> UNA-10) feat: UNA-10 1 commit
70196b3 (main) Add filder struct and add titles
84e3445 (origin/main) Upload wip presentation
```

Hasta abajo de mi pantalla hay dos cuadros "Slides" y "Terminal" el activo se marca en color verde

Bloques de código ejecutables:

```
# 🤖 : Este emoji es porque voy a ejecutar el
# código dentro de la presentación
echo "Hola mundo"
```

```
# 🐱💻: Este otro emoji es porque voy a
# ejecutar código en la pestaña "Terminal"
ps aux
```

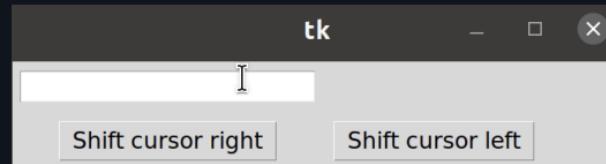
## Intro

### ¿Dónde está el mouse?

---

En mi experiencia teclear es **más rápido** que usar el mouse.

- Misclicks
- Perder el puntero
- Arrastrar hasta el botón



Así que uso el teclado!

Pero ustedes usen lo que les hace sentir más cómodos, pueden **aprender a usar el teclado** o pueden explorar y **aprender y configurar su interfáz gráfica** de preferencia.

## Intro

### ¿Qué significa el comando?

---

La fórmula general para cualquier comando de git es

git <command> <args>

cuando tengan dudas usen

git <command> --help o usen <https://git-scm.com/docs>

¿Cómo sé la fórmula general?

```
$ git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--config-env=<name>=<envvar>] <command> [<args>]
```

Donde:

- [] = Un argumento/comando opcional
- | = otra manera de llamar el mismo argumento/comando
- <> = algo variable

## Intro

### ¿Qué significa el comando?

```
$ git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--config-env=<name>=<envvar>] <command> [<args>]
```



```
# Ejecutando
git -v
# Es igual a
git --version
```

🐱💻 Veamos el resultado completo de `git --help`



## Config

### Configurar su editor de textos

---

Primero es importante **configurar un editor de textos** con el que se sientan cómodos, puede ser de interfáz gráfica o de terminal.

Para configurar el editor pueden usar el siguiente comando reemplazando <path>:

**Nota: Esta opción cambia la configuración global (afecta a todos los repos)**



```
git config --global core.editor "<path>"  
# Ejemplos:  
git config --global core.editor "vim"  
git config --global core.editor "nano"  
git config --global core.editor "gnome-text-editor"
```

## Config

### Configurar Visual Studio Code

```
code --help
```

if you do not see help, please follow these steps:

- \* Mac: Type Shell Command: Install 'Code' command in path from the Command Palette. Command Palette is what pops up when you press shift + ⌘ + P while inside VS Code. (shift + ctrl + P in Windows)
- \* Windows: Make sure you selected Add to PATH during the installation.
- \* Linux: Make sure you installed Code via our new .deb or .rpm packages.

<https://stackoverflow.com/a/36644561>

```
# Visual studio code
git config --global core.editor "code --wait"
# O ruta completa
git config --global core.editor "<path/to/code> --wait"
```

## Config

### Configurar su editor de textos

---

Verifiquemos nuestra configuración corriendo:



```
git config --global -e
```

## Config

### Configurar su editor de textos

---

Extra: Podemos listar los archivos de configuración con:

```
git config --list --show-origin
```

## Config

### PS1

#### The PS1 Shell Variable

The PS1 shell variable defines the text printed before the blinking cursor in our terminal. We may set a simple value like a single character, for example, \$ or #. On the other hand, we may set complex expressions that are evaluated when the prompt is printed.

<https://www.baeldung.com/linux/customize-bash-prompt>



<https://gist.github.com/mitsiu-carreno/b2ed36387ff4157f44002dda8bf81798#file-git-prompt-sh>

Credito: Shawn O. Pearce

Linux

```
# Fedora (/etc/bashrc)
source ~/.git-prompt.sh
[ "$PS1" = "\s-\v\$ " ] && PS1="\u@\W[\033[32m]\$(__git_ps1)\[\033[00m\] \$ "
```

## Config

## PS1

---

Mac

```
source ~/.git-prompt.sh
# Mac (~/.bashrc)
export PS1="\u@\W[\e[32m]\$(__git_ps1)\e[00m] $ "
```

Windows (WSL)

```
source ~/.git-prompt.sh
# WSL Linux (~/.bashrc)
PS1='${VIRTUAL_ENV:+($basename $VIRTUAL_ENV)} ${debian_chroot:+($debian_chroot)}
\[$(tput bold)\]\u@\W[\e[32m]\$(__git_ps1 " (%s)")\e[00m] $ ';
```

## Config

### user.name & user.email

Usualmente queremos tener nuestra cuenta personal separada de nuestra cuenta de trabajo

fix: RIMA-29 deshabilitar el boton de logeo con microsoft cuando la ventan

 JoseMiguelEscalera and  UP200667 committed 2 weeks ago · ✓ 3 / 3

fix: RIMA-28 evitar enviar un error a sentry cuando el usuario cancela el lo

 JoseMiguelEscalera and  UP200667 committed 2 weeks ago · ✓ 3 / 3

chore: RIMA-23 quitar las dependencias de types y moverlas a devdependenci

 JoseMiguelEscalera and  UP200667 committed 2 weeks ago · ✓ 1 / 3



UP200667

Follow

owser para resolver el erro...

ago · ✓ 1 / 3

par alumnos en masa (#91) !

ago · ✓ 1 / 1

## Config

### user.name & user.email

Personalmente divido mis cosas del trabajo en una subcarpeta `*/designa/*`

```
$ tree -L 2 Documents/
Documents/
├── designa
│   ├── acuerdo-286
│   ├── administracion-usuarios-causanatura
│   ├── aprendizajes-para-todos
│   ├── avicola.app
│   ├── causa-natura-media
│   ├── database_backup
│   ├── devops
│   ├── privasol.mx
│   ├── servicios-escolares
│   └── ssh
└── dev_hell
    ├── crunch-test
    ├── cmake_tutorial
    └── git-presentation
└── upa
    ├── oop
    └── plat_web
```

## Config

### user.name & user.email

Personalmente divido mis cosas del trabajo en una subcarpeta `*/designa/*`

`~/.gitconfig`

```
1 [user]
2   name = mitsiu-carreno
3   email = mitsiu.carreno@gmail.com
4 [includeIf "gitdir:*/designa/"]
5   path = .gitconfig.designa
6 [includeIf "gitdir:*/upa/"]
7   path = .gitconfig.upa
8 [core]
9   editor = vim
```

`~/.gitconfig.designa`

```
1 [user]
2   name = designa-mitsiu
3   email = mitsiu.carreno@designamx.com
```

DENTRO DE TI  
EXISTEN DOS LOBOS



ANEXATE, POR FAVOR  
TE QUIERO MUCHO

## Config

### Autenticación SSH

---

Finalmente es posible tener múltiples llaves ssh y asignar cada una a una cuenta de github, en este caso basado en una subcarpeta `*/designa/*` se referencia a la llave `id_ed25519_designa`

`~/.ssh/config`

```
1 Match host github.com exec "[ $(pwd | egrep -ic 'designa') = 1 ]"
2   IdentityFile ~/.ssh/id_ed25519_designa
3
4 Match host github.com exec "[ $(pwd | egrep -ic 'upa') = 1 ]"
5   IdentityFile ~/.ssh/id_ed25519_upa
6
7 Host github.com
8   IdentityFile ~/.ssh/id_ed25519
```

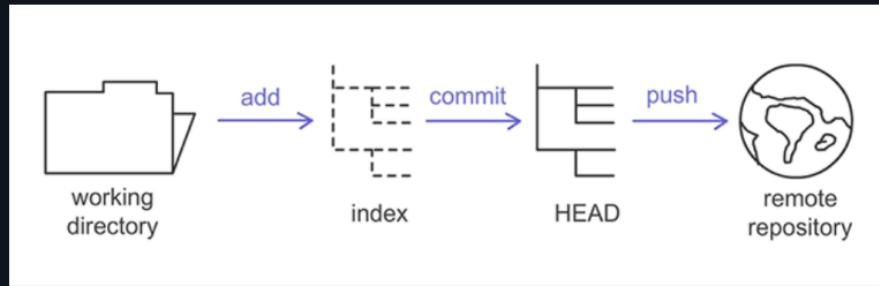
Para más información

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>



## Basics

### Git workflow (Read your outputs)



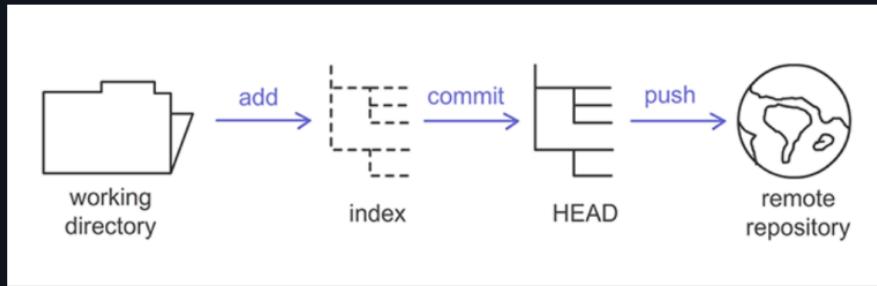
- **Working directory** => El área donde podemos **ver, crear, borrar, editar etc.** Git no da seguimiento de estos archivos **untracked files**

```
$ touch new_file #Crear archivo nuevo  
  
$ git status  
On branch feature-1  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    new_file
```

nothing added to commit but untracked files present (use "git add" to track)

## Basics

### Git workflow (Read your outputs)



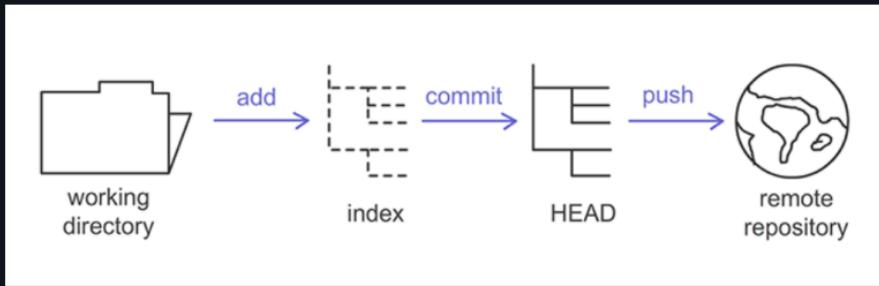
- **Working directory** => El área donde podemos **ver, crear, borrar, editar etc.** Git no da seguimiento de estos archivos **untracked files**

```
$ echo "Modificar archivo commiteado" > modif_file # Agregar texto a archivo existente  
$ git status  
On branch feature-1  
Changes not staged for commit:  
(use "git add <file>..." to update what will be committed)  
(use "git restore <file>..." to discard changes in working directory)  
modified:   modif_file
```

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

## Basics

### Git workflow (Read your outputs)



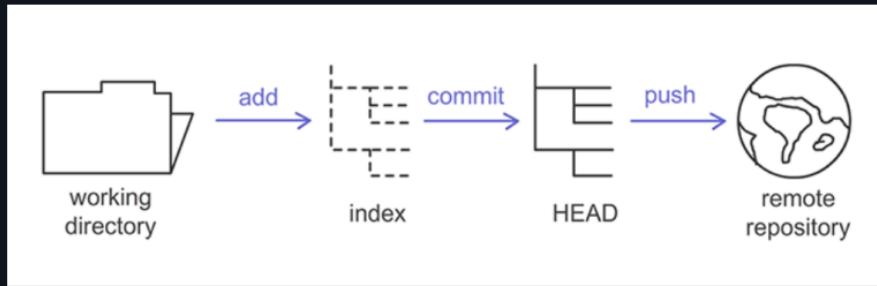
- **Index/Staging** => El área en que git comienza a dar seguimiento de los archivos y a guardar sus cambios

```
$ git add modif_file new_file  
  
$ git status  
On branch feature-1  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
  modified:   modif_file  
  new file:   new_file
```

## Basics

### Git workflow (Read your outputs)

---



- **HEAD** => Un apuntador al commit más reciente de la rama actual

Pero... ¿Qué es un commit?

## Basics

### Git workflow (Read your outputs)

- **HEAD** => Un apuntador al commit más reciente de la rama actual
- **Commit** => Comando para capturar el estado (index/stage) de punto en el tiempo

Imaginemos git como el juego adivina quién, con unas modificaciones...



\*En nuestra versión del juego primero movemos las fichas y luego preguntamos

- Cada pregunta es un mensaje de commit

```
git commit -m'¿Tu personaje tiene barba?'
```

- Cada commit tiene asociado un cambio de estado

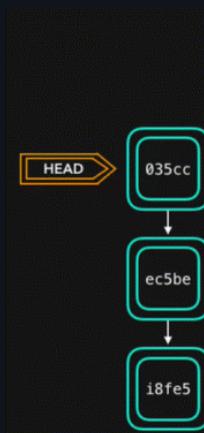
```
git add bill  
git rm sofie
```

## Basics

### Git workflow (Read your outputs)

---

- **HEAD** => Un apuntador al commit más reciente de la rama actual
- **Commit** => Comando para capturar el estado (index/stage) de punto en el tiempo



## Basics

### Git workflow (Read your outputs) EJERCICIO 1

- **HEAD** => Un apuntador al commit más reciente de la rama actual
- **Commit** => Comando para capturar el estado (index/stage) de punto en el tiempo

#### Agregar y commitear

```
$ git add new_file  
$ git commit -m'adding new file'  
[feature-1 70f6e27] adding new file  
 1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 new_file
```

```
$ git add modif_file  
$ git commit -m'modif exist file'  
[feature-1 dd574be] modif exist file  
 1 file changed, 1 insertion(+)
```

#### Historial de commits

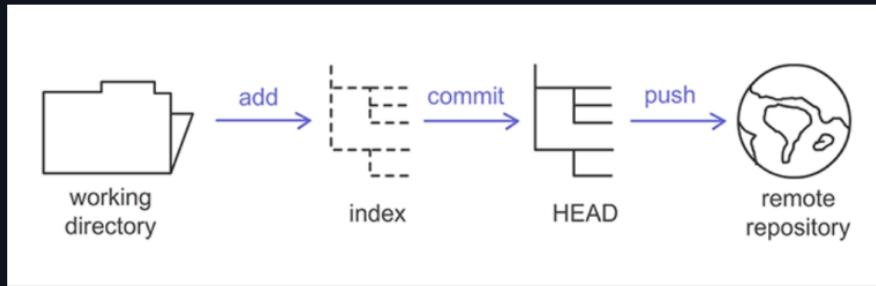
```
$ git log --graph --oneline  
* ecf4186 (HEAD -> feature-1) Setting up
```

```
$ git log --graph --oneline  
* 70f6e27 (HEAD -> feature-1) adding new file  
* ecf4186 (origin/feature-1) setting up prev commit
```

```
$ git log --graph --oneline  
* dd574be (HEAD -> feature-1) modif exist file  
* 70f6e27 adding new file  
* ecf4186 (origin/feature-1) setting up prev commit
```

## Basics

### Git workflow (Read your outputs)

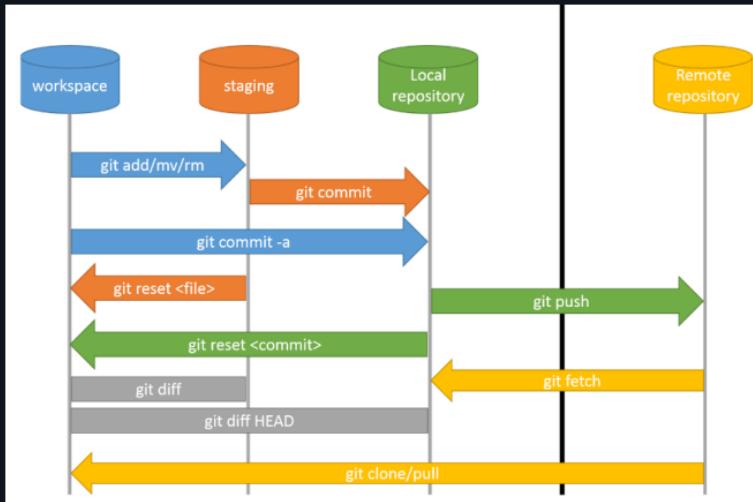


- Remote repo

```
$ git push origin feature-1
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 520 bytes | 520.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:mitsiu-carreno/git-interim-presentation.git
  ecf4186..dd574be feature-1 -> feature-1
```

## Basics

### Git for dummies (Cheatsheet)



## Basics

### Git for dummies (Cheatsheet)

```
1 # Crear repo local
2 git init
3
4 # Clonar repo remoto
5 git clone <URL>
6
7 # Obtener el estatus actual del repositorio
8 git status
9
10 # Seguimiento de los cambios de un archivo
11 git add <file1> <file2>
12
13 # Crear un nuevo commit y agregar un mensaje
14 git commit -m'<Descrip de cambios>'
15
16 # Obtener los cambios del repo remoto
17 git pull origin <branch>
18
19 # Subir cambios al repo remoto
20 git push origin <branch>
```

\*origin => remote name





```
*origin => remote name
```

## ■ Malabares locales

### ■ Stashing

Permite guardar los cambios del directorio de trabajo e index pero a la vez, permitiendo regresar a un directorio de trabajo limpio.

Casos de uso:

- Cambios necesarios en local que nunca deben subirse al repo remoto (.env, config individuales, código de depuración)
- Cambios locales momentáneamente irrelevantes para un fix urgente (queremos mantener los cambios sólo de manera local, pero queremos un directorio de trabajo limpio para resolver un bug)

```
$ git status
On branch UNA-25
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   nginx.conf
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    queries
```

## Malabares locales

### Stashing

---

Guardando en stash

```
$ # git stash
$ # git stash push
$ # git stash push -- nginx.conf test
$ git stash -m"My new stash" -- test nginx.conf
Saved working directory and index state On UNA-25: My new stash

$ git status
On branch UNA-25
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    queries

nothing added to commit but untracked files present (use "git add" to track)

$ git stash list
stash@{0}: On UNA-25: My new stash
stash@{1}: WIP on UNA-18: 7ff5ee3 fix: UNA-18 Allow single questions be omitted
```

**Nota:** El stash más reciente siempre tiene el índice 0

## Malabares locales

### Stashing EJERCICIO 2

---

Recuperado de stash

```
1 $ git stash show 0
2 nginx.conf | 1 +
3 test | 0
4 2 files changed, 1 insertion(+)
5
6 $ git stash pop 0
7 On branch UNA-25
8 Changes to be committed:
9   (use "git restore --staged <file>..." to unstage)
10    new file:   test
11
12 Changes not staged for commit:
13   (use "git add <file>..." to update what will be committed)
14   (use "git restore <file>..." to discard changes in working directory)
15     modified:   nginx.conf
16
17 Untracked files:
18   (use "git add <file>..." to include in what will be committed)
19     queries
20
21 Dropped refs/stash@{0} (a41ae5cabf5c0de873e2bd2b872984538e3c46aa)
```

## Malabares locales

### Stashing

```
git stash --help [pop]
Applying the state can fail with conflicts; in this case, it is not removed from the stash list.
You need to resolve the conflicts by hand and call git stash drop manually afterwards.
```

\*Cuando tengan duda si recuperaron su stash correctamente

```
1 git stash list
```



 **Malabares remotos** **Cherry-picking**

---

Permite aplicar commits existentes en el directorio de trabajo actual.

Casos de uso:

- Hicieron y publicaron (`pushearon`) commits en una rama equivocada.
- Necesitan de manera urgente `cierto commit` de otra rama (pero no toda la rama) [ej, dependencia de una rama en progreso]

**Importante: Por simplicidad, cuando decidan usar `cherry-pick` asegurense de que una de las dos siguientes condiciones se cumpla:**

- La rama de donde copiaron los commits se va a borrar
- Los archivos relacionados al commit que copiaron no se van a modificar hasta que se merge con `main` o la rama de donde se copió

## Malabares remotos

### Cherry-picking

Ejemplo: Se creó la rama incorrecta UNA-10 en lugar de UNA-11 **Nota:** Nadie está trabajando el issue UNA-10 por lo que se puede borrar, pero queremos rescatar el trabajo avanzado

#### Setup

```
$ git log --oneline  
531d90d (HEAD -> UNA-10) commit 4  
2f0b408 commit 3  
c8df2a7 commit 2  
28eacf7 commit 1  
70196b3 (main) Add filder struct and add titles
```



## ■ Malabares remotos

## ■ Cherry-picking

Ejemplo: Se creó la rama incorrecta UNA-10 en lugar de UNA-11 **Nota:** Nadie está trabajando el issue UNA-10 por lo que se puede borrar, pero queremos rescatar el trabajo avanzado

### Setup

```
# Regresamos a la rama principal (¿Por qué?)  
$ git checkout main  
# Actualizamos nuestra rama local (¿Por qué?)  
$ git pull origin main
```



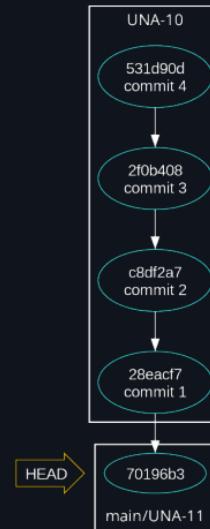
## Malabares remotos

### Cherry-picking

Ejemplo: Se creó la rama incorrecta UNA-10 en lugar de UNA-11 **Nota:** Nadie está trabajando el issue UNA-10 por lo que se puede borrar, pero queremos rescatar el trabajo avanzado

#### Setup

```
# Creamos la rama correcta
$ git checkout -b UNA-11
Switched to a new branch 'UNA-11'
$ git log --oneline
70196b3 (HEAD -> UNA-11, main) Add filder struct and add titles
```



## Malabares remotos

### Cherry-picking

Ejemplo: Se creó la rama incorrecta UNA-10 en lugar de UNA-11 **Nota:** Nadie está trabajando el issue UNA-10 por lo que se puede borrar, pero queremos rescatar el trabajo avanzado

#### Setup

```
# Elegimos los commits a copiar a la nueva rama
$ git cherry-pick 28eacf7
[UNA-11 6f6b8d7] commit 1
  Date: Sat Mar 30 15:43:48 2024 -0600
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 git-interim/a

$ git log --oneline
6f6b8d7 (HEAD -> UNA-11) commit 1
70196b3 (main) Add filder struct and add titles
```

**Importante:** Noten que generamos un nuevo commit



## Malabares remotos

### Cherry-picking

Ejemplo: Se creó la rama incorrecta UNA-10 en lugar de UNA-11 **Nota:** Nadie está trabajando el issue UNA-10 por lo que se puede borrar, pero queremos rescatar el trabajo avanzado

#### Setup

```
# Elegimos los commits a copiar a la nueva rama
$ git cherry-pick c8df2a7
[UNA-11 0292526] commit 2
Date: Sat Mar 30 15:44:04 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 git-interm/b

$ git cherry-pick 2f0b408
[UNA-11 e440696] commit 3
Date: Sat Mar 30 15:44:13 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 git-interm/c

$ git cherry-pick 531d90d
[UNA-11 5b033f9] commit 4
Date: Sat Mar 30 15:44:22 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 git-interm/d
```



**Importante:** Noten que generamos un nuevo commit

### ■ Malabares remotos

#### ■ Cherry-picking EJERCICIO 3

---

Permite aplicar commits existentes en el directorio de trabajo actual.

**Importante: Dado que se crean nuevos commits idénticos a commits anteriores se debe tener cuidado de no generar conflictos:**

- La rama de donde copiaron los commits se va a borrar
- Los archivos relacionados al commit que copiaron no se van a modificar hasta que se merge con main o la rama de donde se copió

```
# Borrrar rama local
$ git checkout -D UNA-10
Deleted branch UNA-10 (was 531d90d)

# Borrar rama remota
$ git push origin --delete UNA-10
```

## Malabares remotos

### git rebase vs git merge (Qué/Cómo/Cuándo)

---

git-merge - Join two or more development histories together

git-rebase - Reapply commits on top of another base tip

Ambos comandos nos permiten integrar cambios entre dos ramas pero lo hacen de manera diferente.

Comencemos por merge que estamos familiarizados:

```
git init
touch a
git add a
git commit -m'main 1'
[master (root-commit) fe2058d] main 1
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 a

touch b
$ git add b
$ git commit -m'main 2'
[master 5bccf6f] main 2
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 b

git checkout -b merge-b
```

```
Switched to a new branch 'merge-b'  
touch m-c  
git add m-c  
git commit -m'merge 1'  
[merge-b a49858a] merge 1
```

## ■ F\*ck ups

Los f\*ck ups sencillos son aquellos que son locales, una vez que se suben al repo remoto se debe tener sensibilidad sobre el riesgo de que alguien haya descargado los cambios a su repositorio remoto.

Parecido al comando cherry-pick, estos comandos literalmente están cambiando la historia (commits nuevos que sustituyen a commits viejos).

## F\*ck ups

### F\*ck up sencillo; mensaje de commit (--amend) EJERCICIO 4

Dada la integración de semantic release

<https://github.com/semantic-release/commit-analyzer?tab=readme-ov-file#default-rules-matching> que usamos en los proyectos, los mensajes de commit deben tener un formato específico para generar una nueva versión a deployar.

```
$ git commit -m'arreglar archivo'  
[main (root-commit) 397c847] arreglar archivo  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 a  
  
$ git log --oneline  
397c847 (HEAD -> main) arreglar archivo  
  
$ git commit --amend -m "fix: UNA-10 arreglar archivo"  
[main f35029d] fix: UNA-10 arreglar archivo  
Date: Sun Mar 31 21:25:32 2024 -0600  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 a  
  
$ git log --oneline  
f35029d (HEAD -> main) fix: UNA-10 arreglar archivo  
  
$ git push origin UNA-10
```

Aspectos a considerar:

- Estamos generando nuevos commits (**sustituyendo historia**) por lo que si el commit ya fue pusheado esta solución solo se debe emplear si estamos seguros que nadie ha descargado el primer commit (397c847).
- Amend solo funciona para el último commit (el más reciente)

## ■ F\*ck ups

### ■ F\*ck up sencillo; Deshacer último commit (reset)

Hay situaciones en las que acabamos de hacer un commit solo para darnos cuenta que faltó hacer algún otro cambio, o faltó agregar otro archivo.

#### Set up

```
git add a && git commit -m'1 +a'  
[main (root-commit) ab641d8] 1 +a  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 a  
$ git add b && git commit -m'2 +b'  
[main e940ab8] 2 +b  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 b  
$ git add c && git commit -m'3 +c'  
[main 757a605] 3 +c  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 c  
$ git add d && git commit -m'4 +d'  
[main 7087d5d] 4 +d  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 d
```

```
$ git add e && git commit -m'5 +e'  
[main 21ae9ea] 5 +e  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 e  
  
$ git log --oneline  
21ae9ea (HEAD -> main) 5 +e  
7087d5d 4 +d  
757a605 3 +c  
e940ab8 2 +b  
ab641d8 1 +a
```

## F\*ck ups

### F\*ck up sencillo; Deshacer último commit (reset)

Es posible deshacer los últimos n commits:

- Regresando los archivos a index `git reset HEAD~`

```
$ git reset HEAD~  
  
$ git status  
On branch main  
Untracked files:  
  (use "git add <file>..." [...])  
    e  
  
nothing added to commit but untracked files present
```

```
$ git log --oneline  
7087d5d (HEAD -> main) 4 +d  
757a605 3 +c  
e940ab8 2 +b  
ab641d8 1 +a
```

- Descartando completamente el último commit `git reset HEAD~ --hard` (el archivo d no está en el working dir)

```
# Descartar completamente el último commit  
$ git reset HEAD~ --hard  
HEAD is now at 757a605 3 +c  
$ git status  
On branch main  
Untracked files:  
  (use "git add <file>..." [...])  
    e  
  
nothing added to commit but untracked files present
```

```
$ git log --oneline  
757a605 (HEAD -> main) 3 +c  
e940ab8 2 +b  
ab641d8 1 +a
```

## F\*ck ups

### F\*ck up sencillo; Deshacer último commit (reset) EJERCICIO 5

---

Para descartar mutliples commits basta con especificar cuantos commits detrás de HEAD `git reset HEAD~2` (afecta los últimos dos commits)

```
$ git reset HEAD~2
$ git status
On branch main
Untracked files:
  (use "git add <file>..." [ . . . ])
    b
    c
    e

nothing added to commit but untracked files present
$ git log --oneline
ab641d8 (HEAD -> main) 1 +a
```

## F\*ck ups

### F\*ck up; difícil; Subir a rama incorrecta (cherry-pick sin delete)

Muy similar al ejemplo de cherry-pick, pero qué pasa si no podemos borrar la rama incorrecta (por ejemplo es otra rama que alguien activamente está trabajando)

```
git log --oneline
4edb9d2 (HEAD -> UNA-10) c11  # Rama incorrecta
a5bef5f c10                      # Rama correcta
ff33b1f b11                      # Rama incorrecta
e54f3f2 b10                      # Rama correcta
d53515f a11                      # Rama incorrecta
1bf9731 a10                      # Rama correcta
a6a7c9c (main) main 1 +a        # Main
```

```
$ git checkout main

$ git checkout -b UNA-11
Switched to a new branch 'UNA-11'

$ git cherry-pick d53515f ff33b1f 4edb9d2
[UNA-11 908ed17] a11
Date: Tue Apr 2 11:38:11 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a11
[UNA-11 ab582cd] b11
Date: Tue Apr 2 11:39:02 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 b11
[UNA-11 41df64b] c11
Date: Tue Apr 2 11:39:28 2024 -0600
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 c11
```

## F\*ck ups

### F\*ck up; difícil; Subir a rama incorrecta (cherry-pick sin delete)

Muy similar al ejemplo de cherry-pick, pero qué pasa si no podemos borrar la rama incorrecta (por ejemplo es otra rama que alguien activamente está trabajando)

```
$ git checkout UNA-10
Switched to branch 'UNA-10'

$ git revert d53515f ff33b1f 4edb9d2

This reverts commit
d53515ff4efcfbf2ba3c7a6c95c0492c60b5b616.

# Please enter the commit message for your changes
# Lines starting with '#' will be ignored,
# and an empty message aborts the commit.
#
# On branch UNA-10
# Revert currently in progress.
#
# Changes to be committed:
#   deleted:    a11
#
```

```
git revert d53515f ff33b1f 4edb9d2
[UNA-10 caaac7a] Revert "a11"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 a11
[UNA-10 288af4d] Revert "b11"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 b11
[UNA-10 c5ba190] Revert "c11"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 c11
```

## F\*ck ups

### F\*ck up; difícil; Subir a rama incorrecta (cherry-pick sin delete) EJERCICIO 6

---

Muy similar al ejemplo de cherry-pick, pero qué pasa si no podemos borrar la rama incorrecta (por ejemplo es otra rama que alguien activamente está trabajando)

```
$ git log --oneline
c5ba190 (HEAD -> UNA-10) Revert "c11"
288afd4 Revert "b11"
caaac7a Revert "a11"
4edb9d2 c11
a5bef5f c10
ff33b1f b11
e54f3f2 b10
d53515f a11
1bf9731 a10
a6a7c9c (main) main 1 +a

$ git checkout main
$ git merge UNA-11
$ git merge UNA-10
```

## F\*ck ups

### F\*ck up; dificil; Multiples errores en multiples commits (rebase -i)

---

Regresando al setup anterior, pero ahora imaginemos que el error es distinto, todos los commits pertenecen a la rama pero olvidamos poner "fix: UNA-10" a todos nuestros commits

```
git log --oneline
4edb9d2 (HEAD -> UNA-10) c11
a5bef5f c10
ff33b1f b11
e54f3f2 b10
d53515f a11
1bf9731 a10
a6a7c9c (main) main 1 +a
```

```
$ git rebase -i HEAD~6
```

## F\*ck ups

### F\*ck up; dificil; Multiples errores en multiples commits (rebase -i)

---

git rebase -i HEAD~<n> Abre el editor de texto y nos permite realizar cualquiera de las siguientes operaciones:

```
$ git rebase -i HEAD~6
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                               commit's log message, unless -C is used, in which case
#                               keep only this commit's message; -c is same as -C but
#                               opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#                   create a merge commit using the original merge commit's
#                   message (or the oneline, if no original merge commit was
#                   specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                           to this position in the new commits. The <ref> is
```

 F\*ck ups F\*ck up; difícil; Multiples errores en multiples commits (rebase -i)

---

En nuestra situación queremos cambiar el mensaje del commit (`reword`) primero debemos indicar la operación a realizar

Noten que los commits están ordenados comenzando por el más antiguo

```
$ git rebase -i HEAD~6
pick 1bf9731 a10
pick d53515f a11
pick e54f3f2 b10
pick ff33b1f b11
pick a5bef5f c10
pick 4edb9d2 c11
```

```
$ git rebase -i HEAD~6
reword 1bf9731 a10
reword d53515f a11
reword e54f3f2 b10
reword ff33b1f b11
reword a5bef5f c10
reword 4edb9d2 c11
```

Guardamos y cerramos el editor

## F\*ck ups

### F\*ck up; difícil; Multiples errores en multiples commits (rebase -i)

---

A continuación se abrirá el editor de textos para **escribir el nuevo mensaje del commit**, además de mostrar el contexto del commit.

```
fix: UNA-10 a10

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue Apr 2 11:37:33 2024 -0600
#
# interactive rebase in progress; onto a6a7c9c
# Last command done (1 command done):
#   reword 1bf9731 a10
# Next commands to do (5 remaining commands):
#   reword d53515f a11
#   reword e54f3f2 b10
# You are currently editing a commit while rebasing branch 'UNA-10' on 'a6a7c9c'.
#
# Changes to be committed:
#   new file:   a10
#
```

Esta operación se repetirá para cada commit seleccionado

## F\*ck ups

### F\*ck up; difícil; Multiples errores en multiples commits (rebase -i) EJERCICIO 7

---

Finalmente queda resaltar que esta operación **reescribe la historia al generar nuevos commit** así que usenla de manera apropiada

F\*ck ups

Mega f\*ck up; crítico; Subir un PR mal (revert -m 1 SHA1)

You know when you fuck something up and you wish you had the power to hit undo?

Like when you say the dumbest thing in front of your biggest crush

**Or when you push a commit that undoes your boss work, and it gets approved and merged into main?** We all have those days, but today is not going to be one of them.

## Anecdota time!

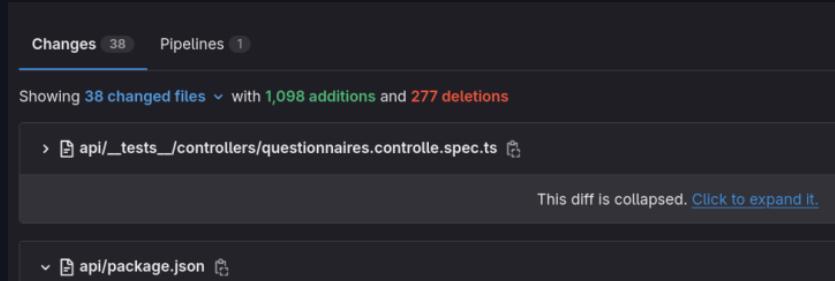
1 Por casualidad ví que mi cambio  
mío aprobado unos días antes se  
había borrado

## ■ F\*ck ups

### ■ Mega f\*ck up; crítico; Subir un PR mal (revert -m 1 SHA1)

---

2 Y este no era un PR pequeño...



3 Por una parte me urgía hacer deploy, pero por otra, ese PR era demasiado grande como para revisarlo apropiadamente.



## ████ F\*ck ups

### ████ Mega f\*ck up; crítico; Subir un PR mal (revert -m 1 SHA1)

Revert no cambia los commits históricos, aún se podía preservar los cambios para una mejor revisión posterior, pero ya no estaban en main

The screenshot shows a list of GitHub commits from March 20, 2024. It includes several merge commits and one revert commit:

- Revert "Merge branch 'UNA-22-R' into 'main'" by Mitsiu Carreño, authored 1 week ago.
- Merge branch 'UNA-28' into 'main' by Mitsiu Carreño, authored 1 week ago.
- fix: UNA-28 extend try block by Mitsiu Carreño, authored 1 week ago.
- Mar 20, 2024
- fix: UNA-27 Bug en crear cuestionario by AngelSilva85, authored 1 week ago.
- Merge branch 'UNA-26' into 'UNA-26-R' by Jose Angel Silva Salgado, authored 1 week ago.
- Merge branch 'UNA-9-R' into 'main' by José Miguel Escalera, authored 1 week ago.
- Merge branch 'UNA-13' into 'UNA-13-R' by Jose Angel Silva Salgado, authored 1 week ago.
- Merge branch 'UNA-22-R' into 'main' by Ramsés Meza, authored 1 week ago.

## F\*ck ups

### Mega f\*ck up; F\*ck up en el fix de otro F\*ck up (reflog) EJERCICIO 8

Con reflog podemos **regresar operaciones destructivas** como reset, rebase

```
$ git reflog
c0d6242 (HEAD -> UNA-10) HEAD@{0}: rebase (finish): returning to refs/heads/UNA-10
c0d6242 (HEAD -> UNA-10) HEAD@{1}: rebase (pick): c10
bbd3e90 HEAD@{2}: rebase (pick): b10
1bf9731 HEAD@{3}: rebase (start): checkout HEAD~6
4edb9d2 HEAD@{4}: rebase (finish): returning to refs/heads/UNA-10
4edb9d2 HEAD@{5}: rebase (start): checkout HEAD~6
4edb9d2 HEAD@{6}: commit: c11
a5bef5f HEAD@{7}: commit: c10
ff33b1f HEAD@{8}: commit: b11
e54f3f2 HEAD@{9}: commit: b10
d53515f HEAD@{10}: commit: a11
1bf9731 HEAD@{11}: commit: a10
a6a7c9c (main) HEAD@{12}: checkout: moving from main to UNA-10
a6a7c9c (main) HEAD@{13}: commit (initial): main 1 +a
$ git reset --hard HEAD@{4}
HEAD is now at 4edb9d2 c11
```

## F\*ck ups

## Who f\*ck up?

Finalmente...

A veces veo código tan feo que necesito ponerle una cara, otras veces me sirve entender el contexto de cierta línea de código para entender mejor qué está haciendo o porqué se hizo así. Si se encuentran en una situación similar pueden usar:

```
# git log -L <start_line>,<end_line>:path/to/file
git log -L 12,15:api/src/controllers/users.controller.ts
```

```
commit c76627696db109d766bbe19651c4d868dbb369f0
Author: mitsiu.carreno <mitsiu.carreno@gmail.com>
Date:   Wed Mar 20 20:24:20 2024 -0600

        Revert "Merge branch 'UNA-22-R' into 'main'"

    This reverts commit 1261564b40d623e909904f42ee55d2cd90fc60e8, reversing
    changes made to 65ea6582dbd26ab01e3ecb74873471d72294ffe.

diff --git a/api/src/models/questionnaire.ts b/api/src/models/questionnaire.ts
--- a/api/src/models/questionnaire.ts
+++ b/api/src/models/questionnaire.ts
@@ -7,12 +8,9 @@
-export type Questionnaire = {
-  id?: number;
-  title: string;
-  frequency: 'monthly' | 'weekly' | 'daily';
-  description: string;
-  product: 'huevo' | 'pollo';
-  enabled?: boolean;
-};
```

Copiamos el SHA1 (c76627696db109d766bbe19651c4d868dbb369f0)

████ F\*ck ups

████ Who f\*ck up?

En el repo remoto abrimos cualquier commit

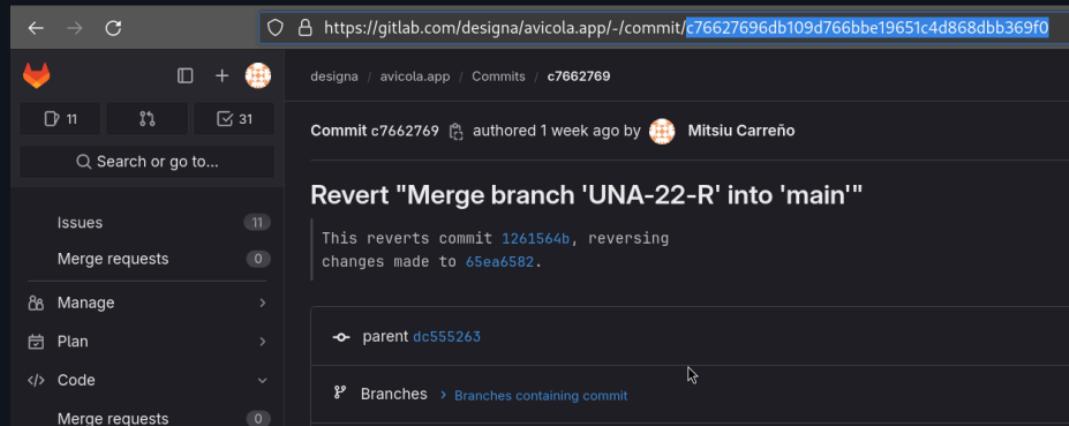
The screenshot shows a GitLab repository interface for the project 'designa / avicola.app'. The main navigation bar includes 'Issues' (11), 'Merge requests' (0), and 'Manage'. The repository name is 'avicola.app'. A search bar at the top right says 'Search or go to...'. Below it, there's a 'Pinned' section with 'Issues' (11) and 'Merge requests' (0). The main content area displays a merge request titled 'Merge branch 'UNA-38-R' into 'main''. It was authored by Mitsu Carreno 6 hours ago. The merge request has two commits: 'fix: UNA-38 remove hardcoded' and 'feat: cambio para filtrar variables'.

| Name   | Last commit                         |
|--------|-------------------------------------|
| api    | fix: UNA-38 remove hardcoded        |
| import | feat: cambio para filtrar variables |

F\*ck ups

Who f\*ck up?

Y reemplazamos el SHA1 en la url





■ Gracias

*git commit -m "changes"*



*Writing*

# Useless Git Commit Messages



