

# Introducción a GitHub para la docencia y la investigación

## Curso de Formación del Profesorado

14 de marzo de 2023

**Rafael Cabañas de Paz**

Departamento de Matemáticas  
Área de Estadística e Investigación Operativa  
Universidad de Almería  
[rcabanas@ual.es](mailto:rcabanas@ual.es)





GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

# 1. Introducción



# Control de versiones con GitHub y git

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

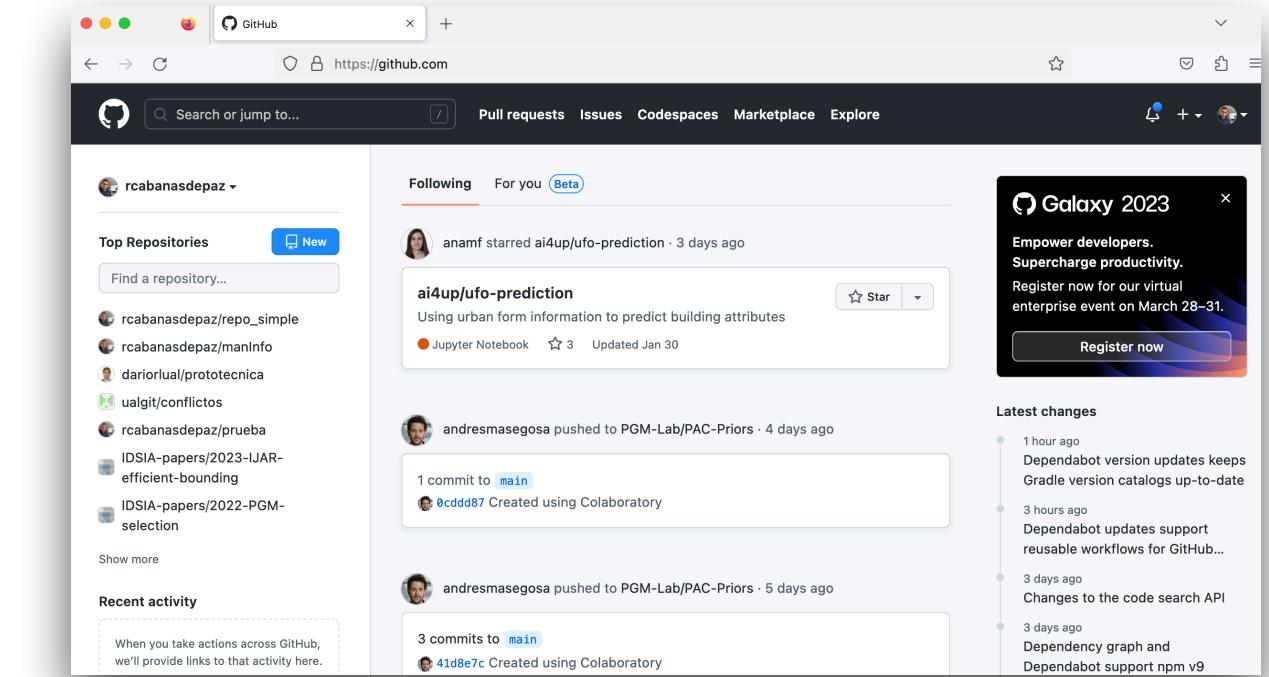
4. Correcciones

5. Ramas

6. Sincronización remota

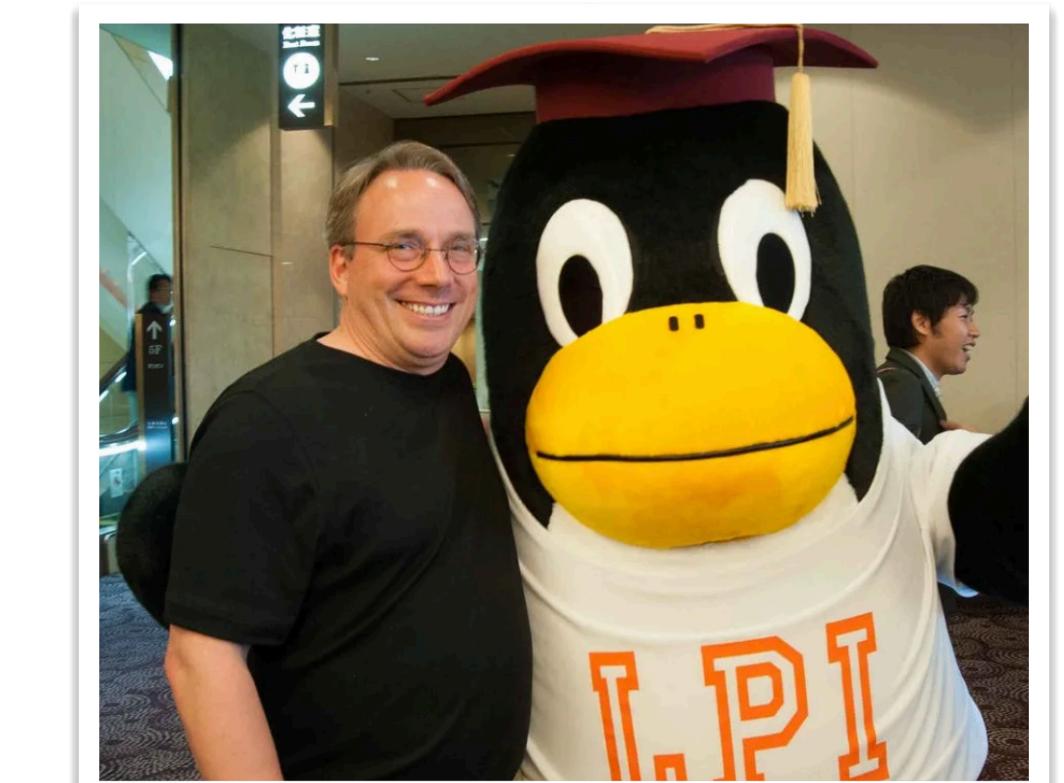
## ¿Qué es GitHub?

- Sitio web y servicio en nube para la gestión de código.
- Utiliza el sistema git
- Actualmente pertenece a Microsoft



## ¿Qué es Git?

- Sistema de control de versiones
- Desarrollado por Linus Torvalds en 2005
- Distribuido
- Open source



git



Bitbucket



GitLab

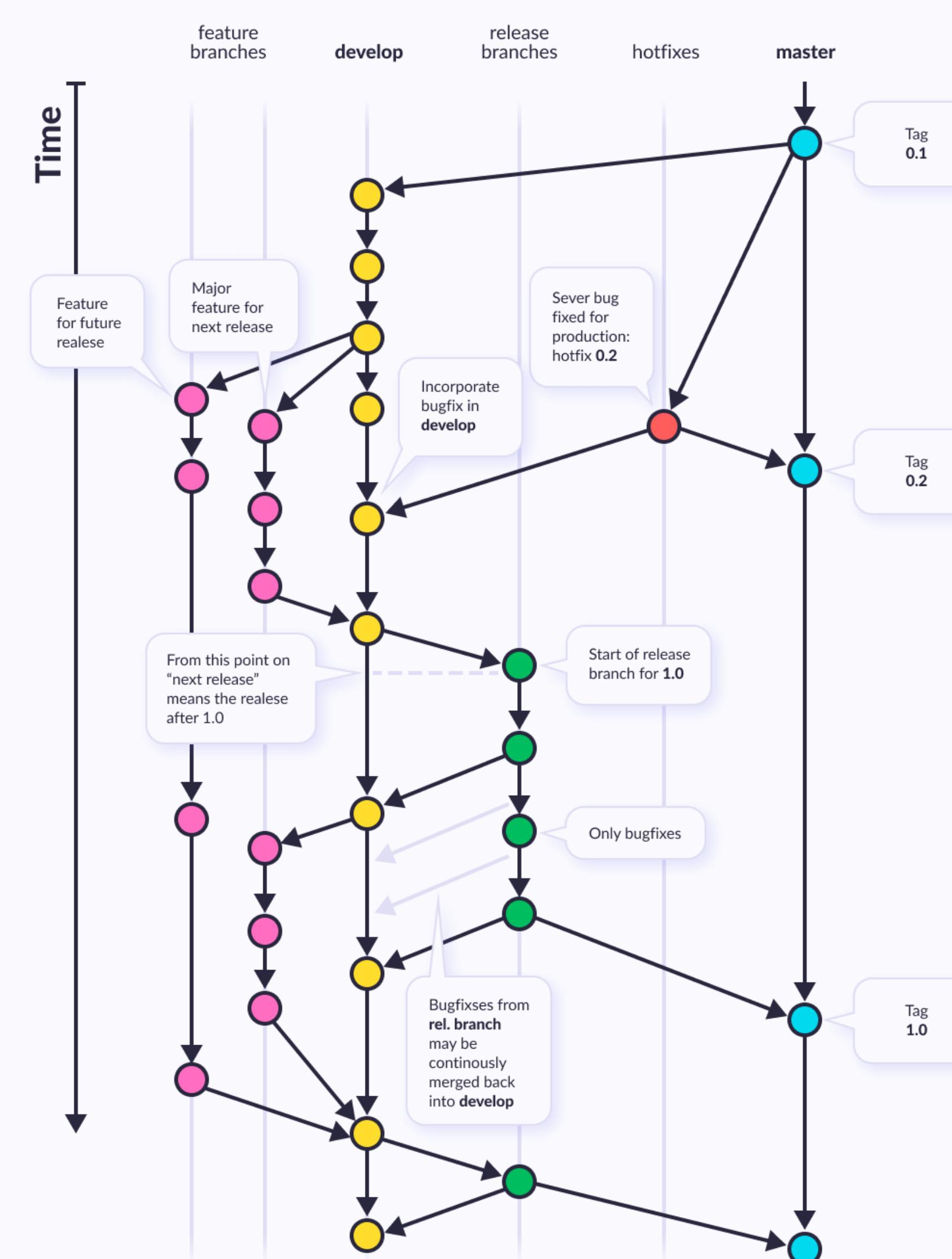


# Control de versiones con GitHub y git

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota





# Usuarios en GitHub

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

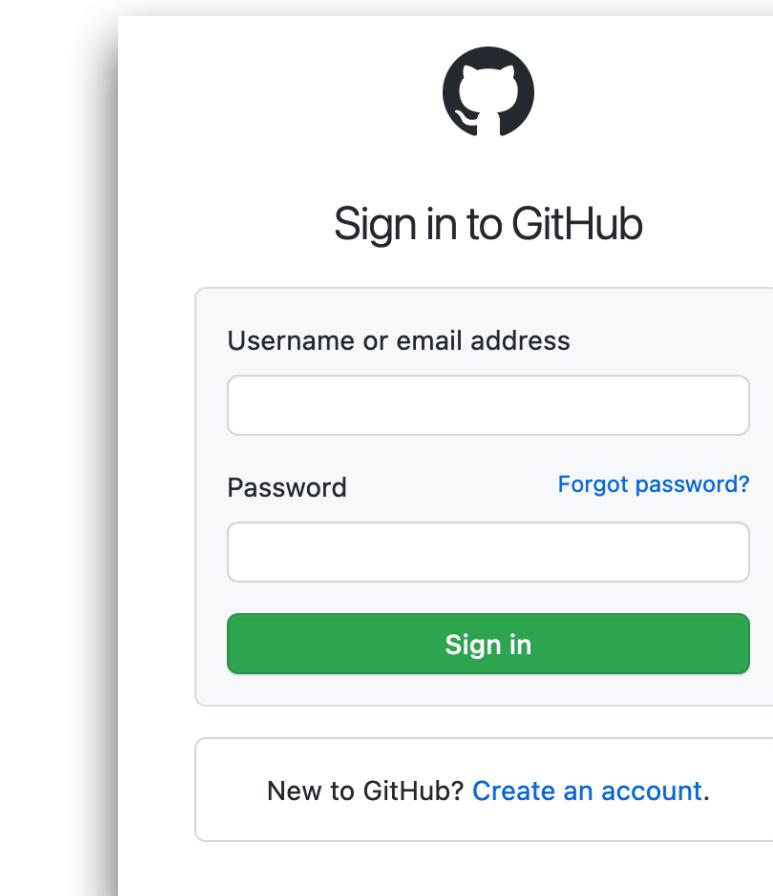
4. Correcciones

5. Ramas

6. Sincronización  
remota

Para poder seguir este curso, es necesario disponer de un usuario en **GitHub**

1. Abre en el navegador <https://github.com/>
2. Si no tienes cuenta, pulsa en **Sign up**
3. Una vez creada la cuenta, pulsa en **Sign in** e identifícate:





# Interfaz web de GitHub

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Muchas operaciones con GitHub se pueden realizar via web: <https://github.com/>

The screenshot shows the GitHub homepage in a web browser. The top navigation bar includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. The main area displays the user's profile (rcabanasdepaz), followed by a list of repositories (Top Repositories) such as rcabanasdepaz/repo\_simple, rcabanasdepaz/manInfo, darioalual/prototecnica, ualgit/conflictos, rcabanasdepaz/prueba, IDSIA-papers/2023-IJAR-efficient-bounding, and IDSIA-papers/2022-PGM-selection. Below the repositories is a "Recent activity" section showing pushes from andresmasegosa to PGM-Lab/PAC-Priors. A sidebar on the right shows "Latest changes" with updates from Dependabot. A promotional banner for "Galaxy 2023" is visible on the right side.



# Uso del Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- La mayor parte de este curso se hará mediante terminal.

bash / zsh

```
jen -- bash -- 80x24
Last login: Mon May 10 13:15:38 on ttys000
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Jens-MacBook-Pro:~ jen$ sudo powermetrics --samplers smc |grep -i "CPU die temperature"
>Password:
CPU die temperature: 51.47 C
CPU die temperature: 51.16 C
CPU die temperature: 50.88 C
CPU die temperature: 50.56 C
CPU die temperature: 50.34 C
[Restored 4 Jun 2021 at 09:37:39]
Last login: Mon May 17 09:58:50 on console
Restored session: Mon 10 May 2021 13:16:54 BST
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Jens-MacBook-Pro:~ jen$ sudo powermetrics
```

bash

```
mark@linux-desktop:/tmp/tutorial
File Edit View Search Terminal Help
mark@linux-desktop:/tmp/tutorials$ mv "folder 1" folder_1
mark@linux-desktop:/tmp/tutorial$ mv "folder 2" folder_2
mark@linux-desktop:/tmp/tutorials$ mv "folder 3" folder_3
mark@linux-desktop:/tmp/tutorial$ mv "folder 4" folder_4
mark@linux-desktop:/tmp/tutorials$ mv "folder 5" folder_5
mark@linux-desktop:/tmp/tutorial$ mv "folder 6" folder_6
mark@linux-desktop:/tmp/tutorial$ ls
another      dir1 folder  folder_3 folder_6
combined_backup.txt  dir2 folder_1 folder_4 output.txt
combined.txt   dir4 folder_2 folder_5
mark@linux-desktop:/tmp/tutorial$ rm dir4/dir5/dir6/combined.txt combined_backup
.txt
mark@linux-desktop:/tmp/tutorial$ rm folder_*
rm: cannot remove 'folder_1': Is a directory
rm: cannot remove 'folder_2': Is a directory
rm: cannot remove 'folder_3': Is a directory
rm: cannot remove 'folder_4': Is a directory
rm: cannot remove 'folder_5': Is a directory
rm: cannot remove 'folder_6': Is a directory
mark@linux-desktop:/tmp/tutorial$
```

cmd / PowerShell

```
Command Prompt
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Users\Brennan>
```





# Interfaz gráfica

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- También es posible interactuar con GitHub utilizando un programa con interfaz gráfica. Aquí veremos principalmente **GitHub Desktop**

The screenshot shows the GitHub Desktop application window. At the top, there's a menu bar with File, Edit, View, Repository, Branch, and Help. Below the menu is a toolbar with icons for repository status, current branch (progress-reporting), and publishing. The main area displays a code diff for a file named app\src\ui\app.tsx. The diff shows several lines of code being modified, with blue highlights for new lines and red highlights for deleted lines. The commit dialog at the bottom has fields for 'Show progress in toolbar' (checkbox), 'Description' (text area), and a 'Commit to progress-reporting' button.

- Otras alternativas son: SourceTree, GitKraken



# Simulador

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

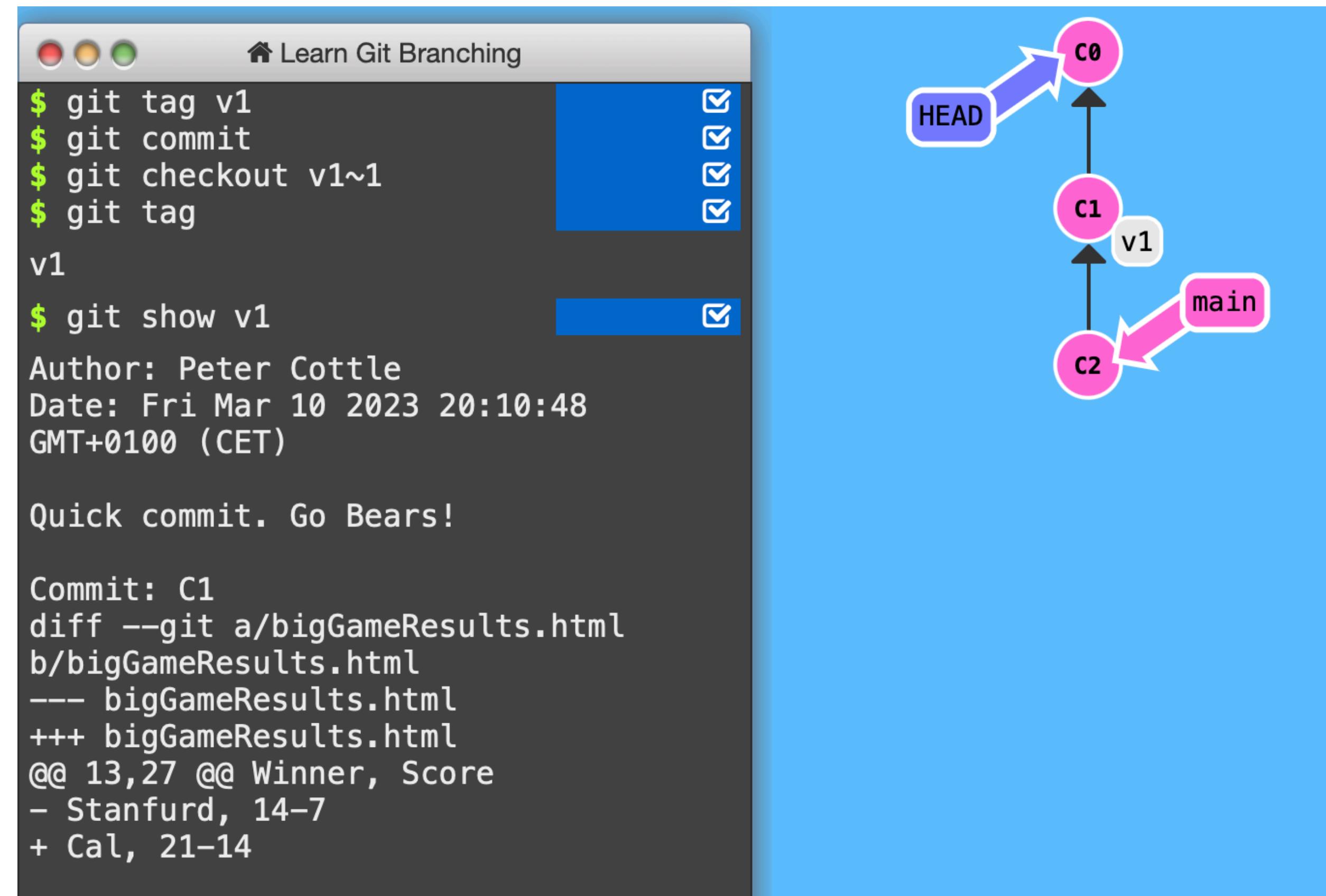
6. Sincronización  
remota

```
$ git tag v1
$ git commit
$ git checkout v1~1
$ git tag
v1
$ git show v1
Author: Peter Cottle
Date: Fri Mar 10 2023 20:10:48
GMT+0100 (CET)

Quick commit. Go Bears!

Commit: C1
diff --git a/bigGameResults.html
b/bigGameResults.html
--- bigGameResults.html
+++ bigGameResults.html
@@ 13,27 @@ Winner, Score
- Stanfurd, 14-7
+ Cal, 21-14
```

Learn Git Branching



The diagram shows a commit graph with three commits: C0 (top), C1 (middle), and C2 (bottom). Commit C0 is the root. Commit C1 is a child of C0. Commit C2 is a child of C1. A tag 'v1' points to commit C1. The 'HEAD' pointer points to commit C0. A branch 'main' points to commit C2.

URL: <https://learngitbranching.js.org/?NODEMO>



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## 2. Configuración y creación de repositorios



# Creación de repositorios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

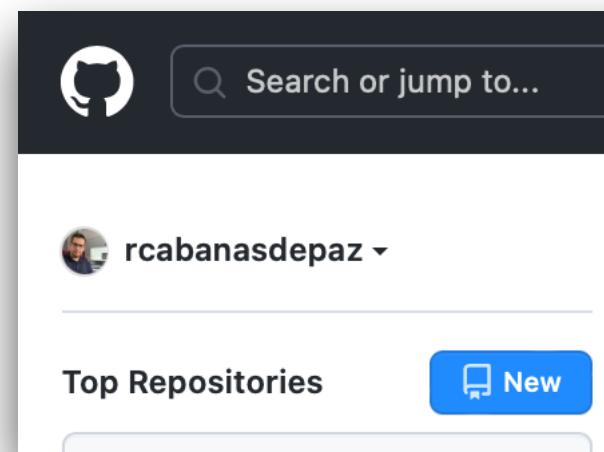
4. Correcciones

5. Ramas

6. Sincronización  
remota

- Una manera de crear un repositorio es hacerlo a través de la web.

1



2

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner \* Repository name \*

/

Great repository names are short and memorable. Need inspiration? How about [sturdy-octo-spoon](#)?

Description (optional)

Ejemplo de cómo crear un repositorio desde la web

Public  
Anyone on the internet can see this repository. You choose who can commit.

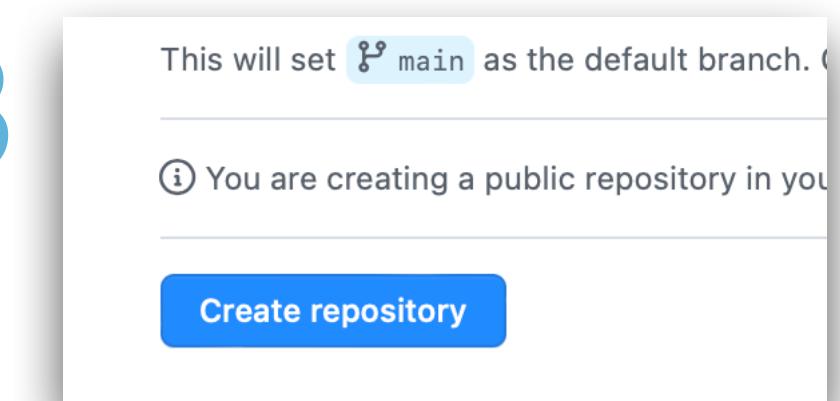
Private  
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file  
This is where you can write a long description for your project. [Learn more](#).

3





# Clonar repositorios

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

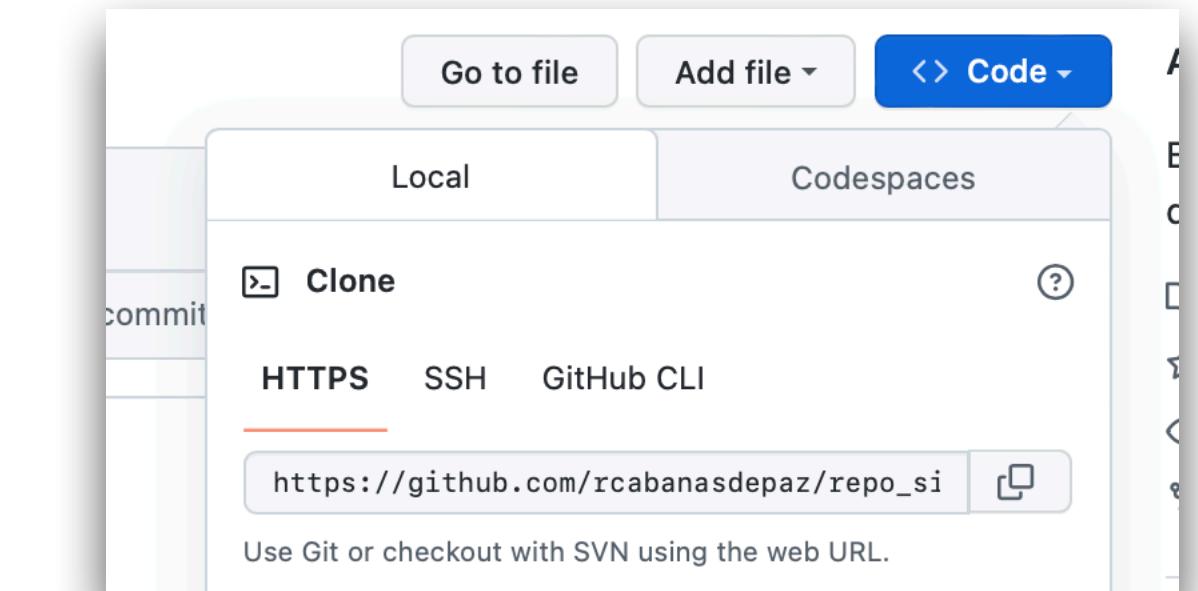
4. Correcciones

5. Ramas

6. Sincronización  
remota

- Clonar repositorios consiste en descargar una copia local.

- Para ello, primero hay que obtener la url del fichero .git

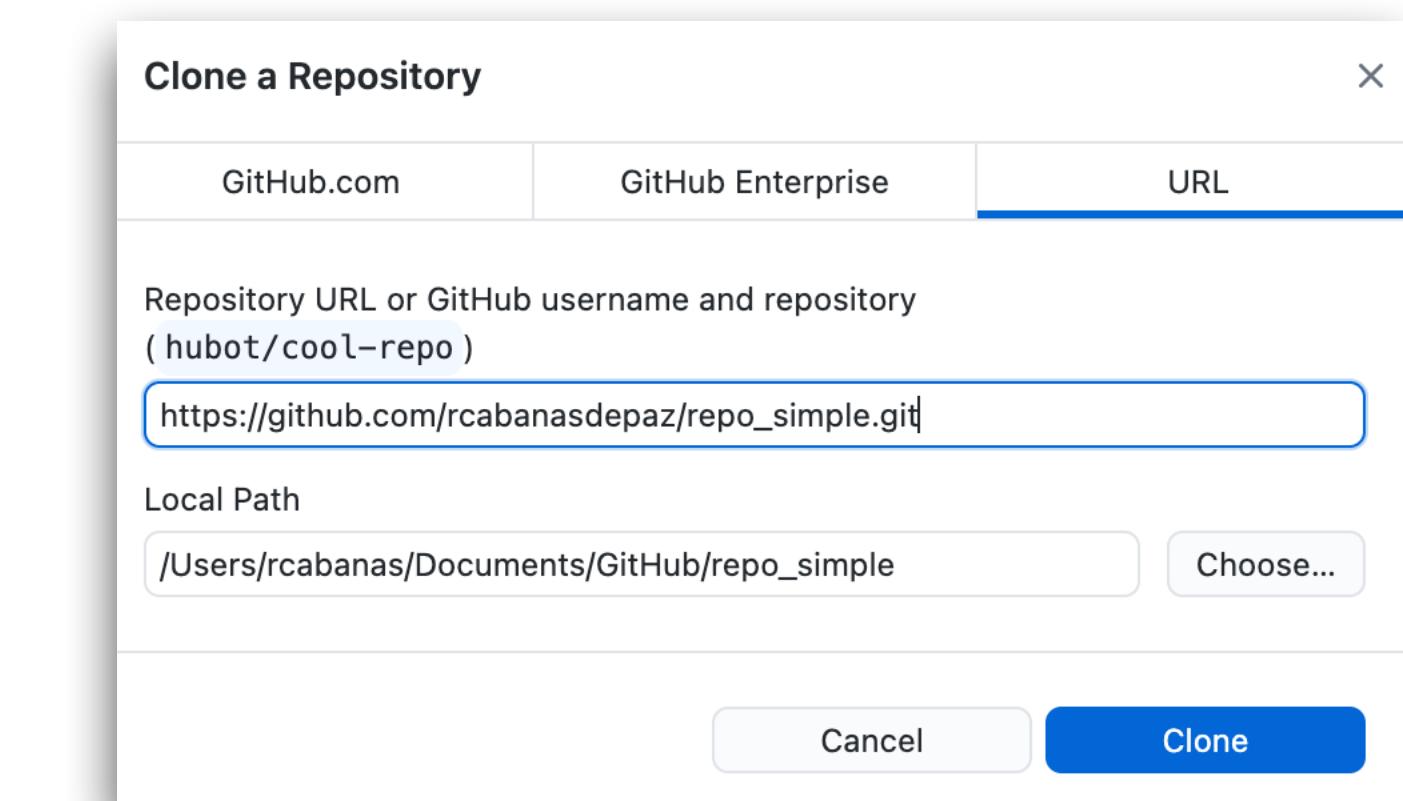
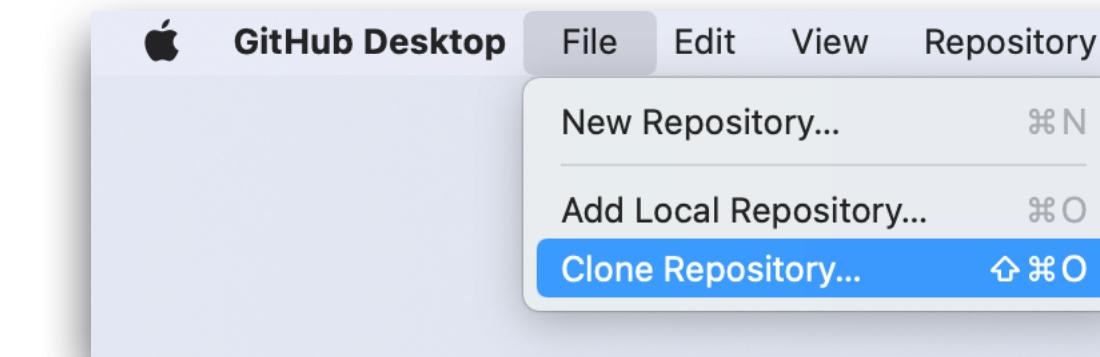


- Para realizar la descarga, se utiliza el comando **git clone**:

\$ \_

```
git clone https://github.com/rafacabanasdepaz/repo_simple.git
```

- O mediante GitHub Desktop:





# Identificación mediante tokens

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

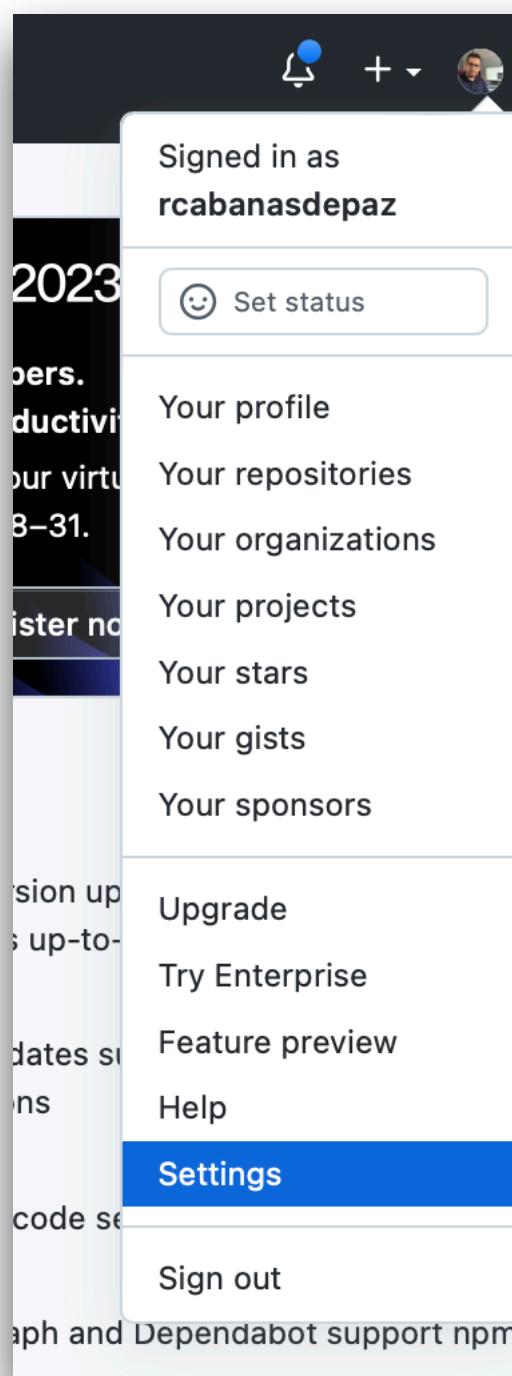
4. Correcciones

5. Ramas

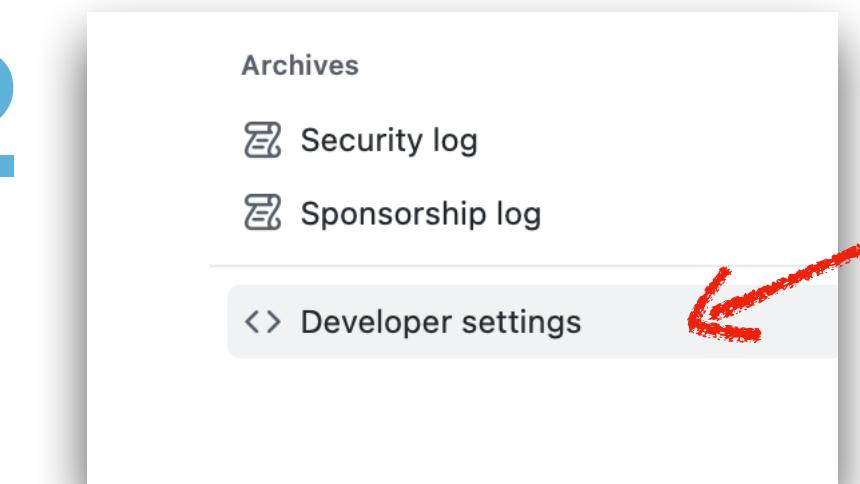
6. Sincronización remota

- Los tokens son contraseñas generadas de forma aleatoria que nos permitirán conectar con nuestra cuenta de GitHub

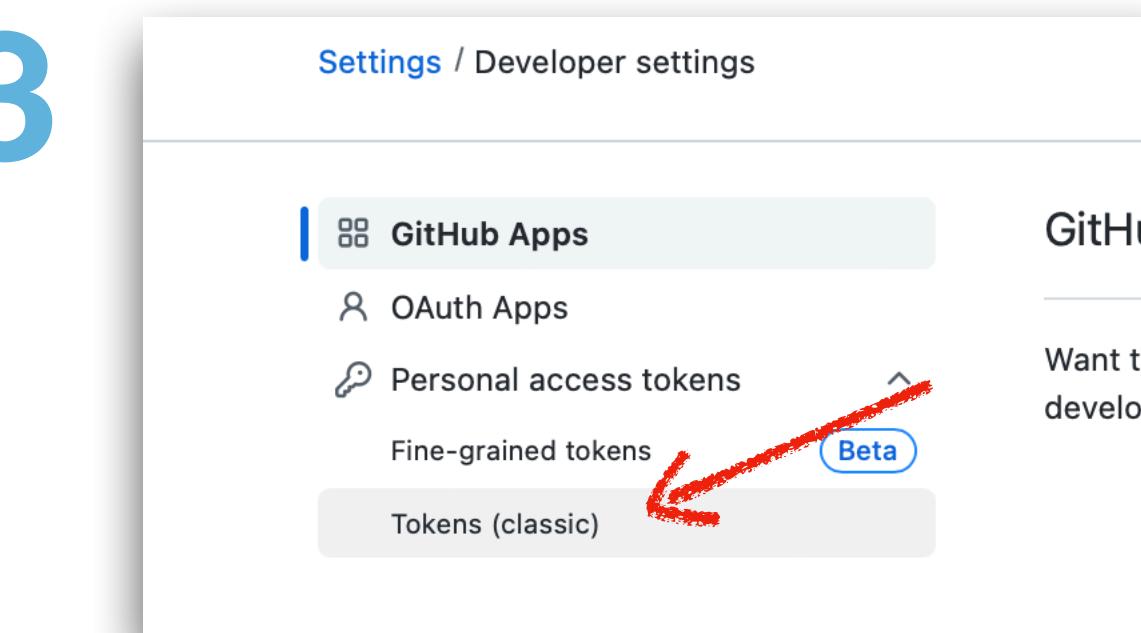
1



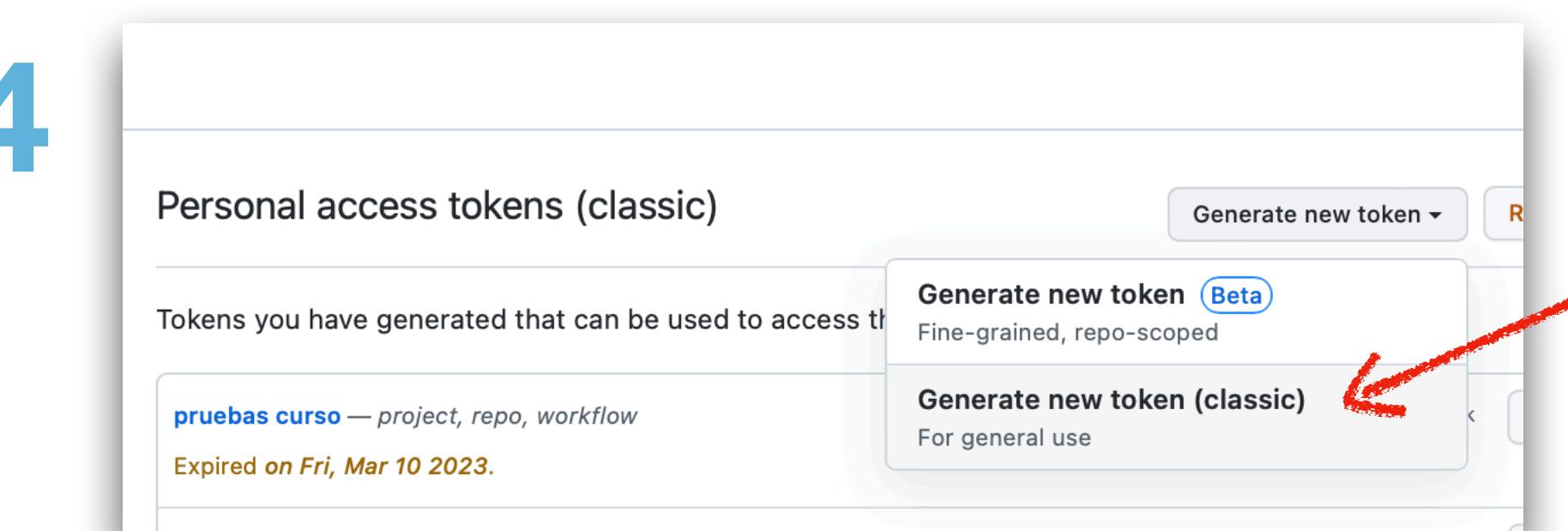
2



3



4



- Importante:** Guárdala hasta el final de la clase. Si no estás usando tu ordenador, usa mañana como fecha de caducidad.



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

### 3. Control de cambios en local



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

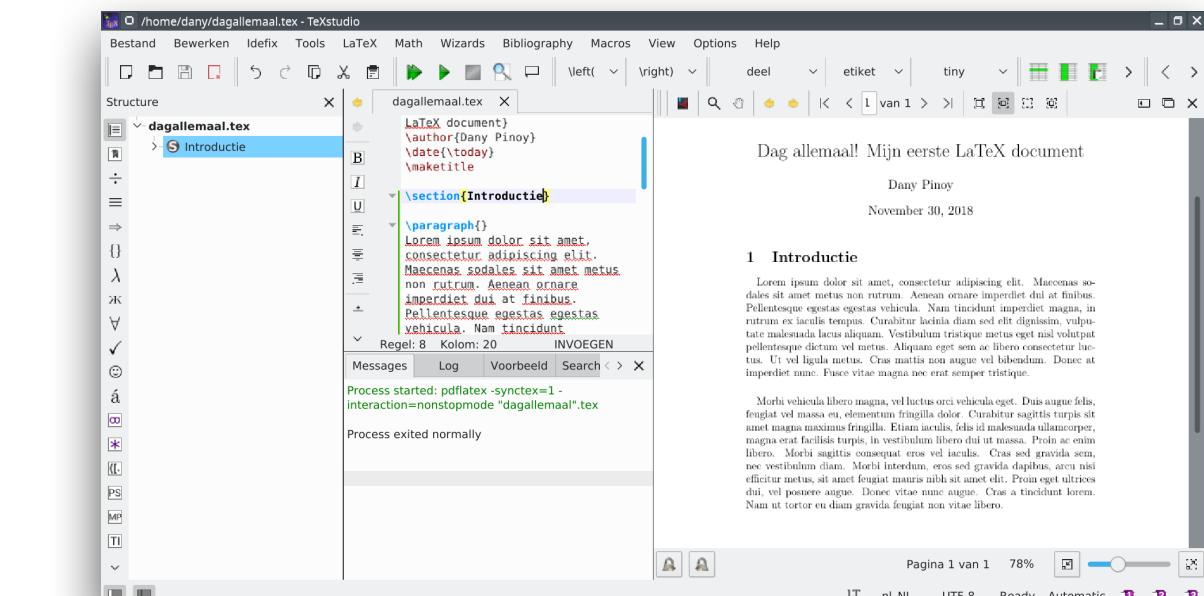
4. Correcciones

5. Ramas

6. Sincronización remota

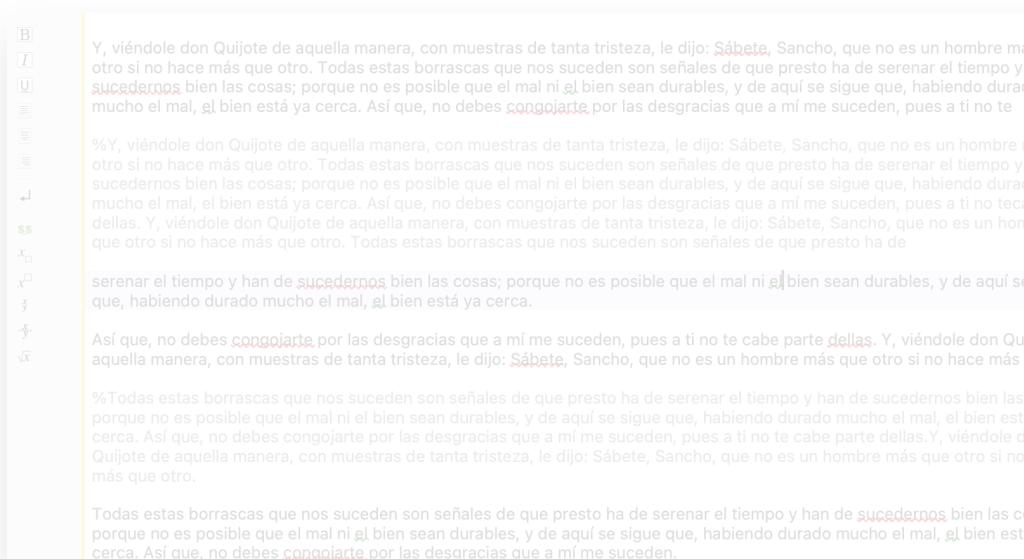
## Contexto

- Estamos preparando un artículo en Latex
- Somos el autor único (o el único que escribe)



## Possibles escenarios

- Eliminamos un párrafo que luego queremos volver a añadir
- Reescribimos un texto pero queremos volver a la versión anterior
- Queremos registrar la versión exacta enviada (y las revisiones)



## Comentarios

## Copias de seguridad manuales



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

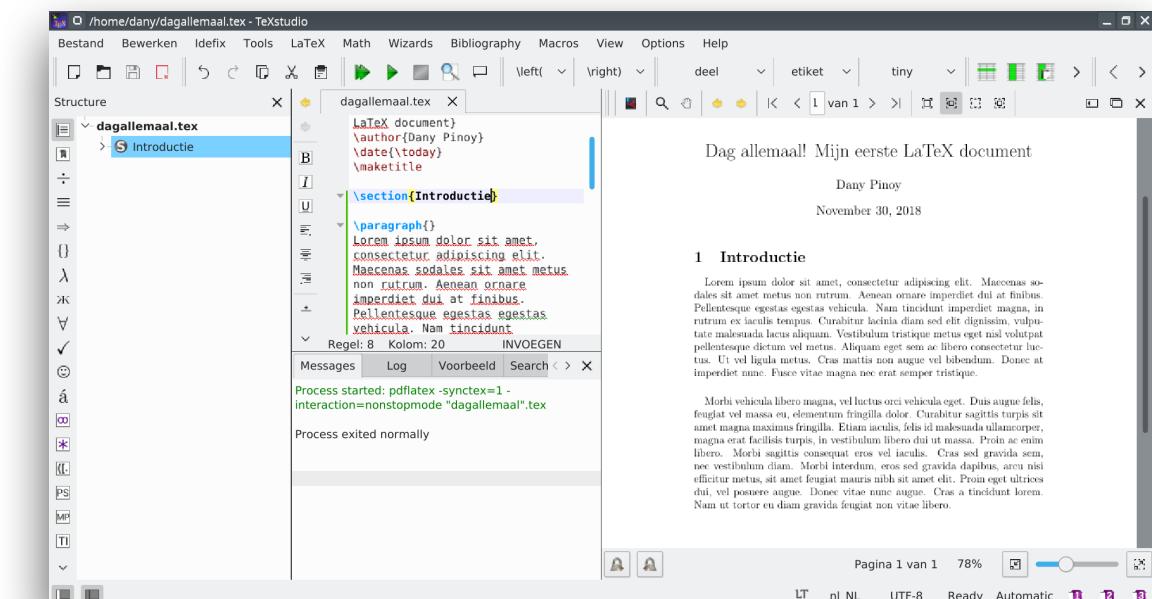
4. Correcciones

5. Ramas

6. Sincronización remota

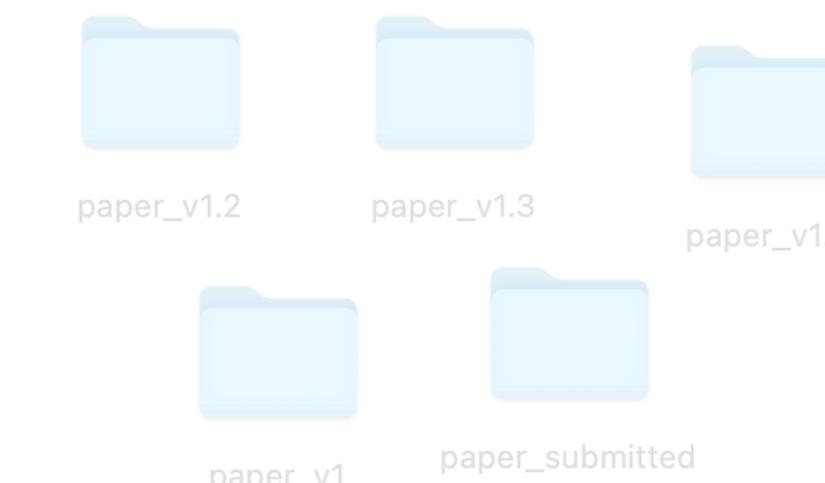
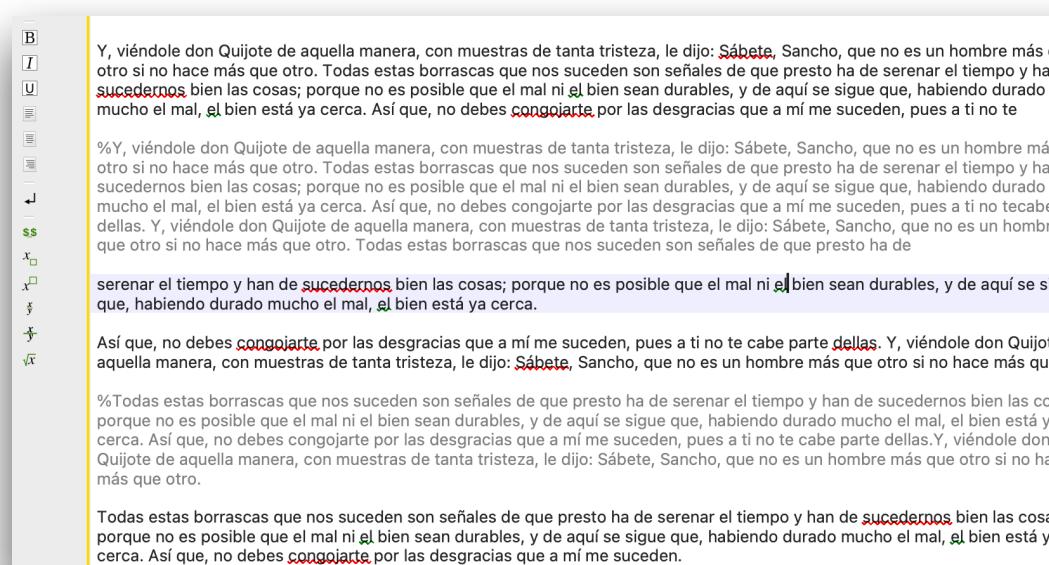
## Contexto

- Estamos preparando un artículo en Latex
- Somos el autor único (o el único que escribe)



## Posibles escenarios

- Eliminamos un párrafo que luego queremos volver a añadir
- Reescribimos un texto pero queremos volver a la versión anterior
- Queremos registrar la versión exacta enviada (y las revisiones)



## Comentarios

## Copias de seguridad manuales



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

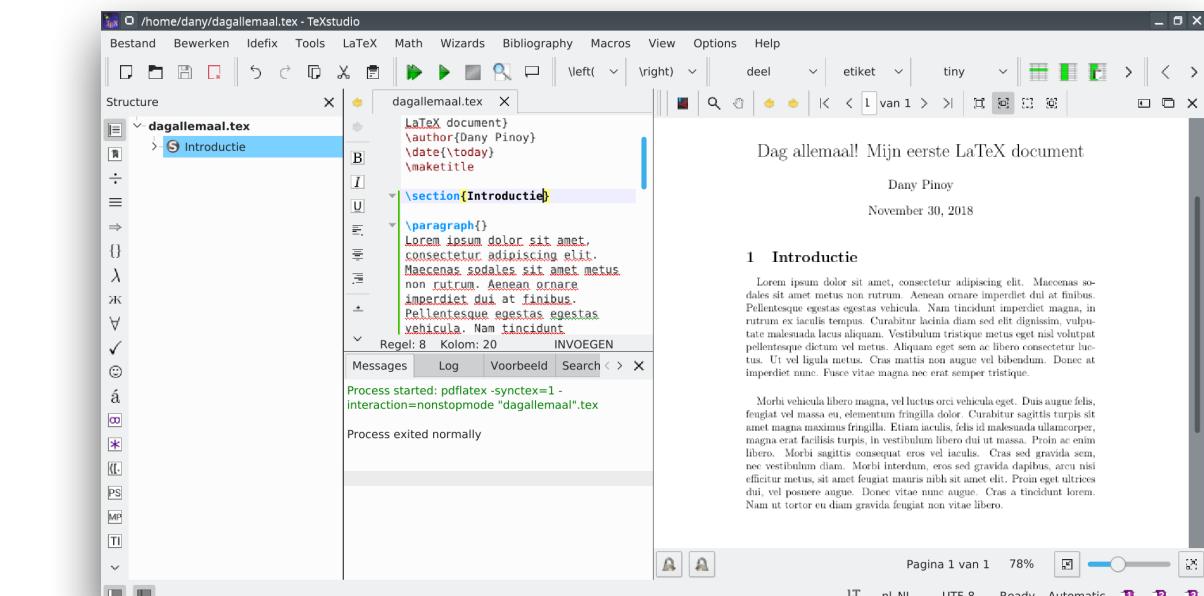
4. Correcciones

5. Ramas

6. Sincronización remota

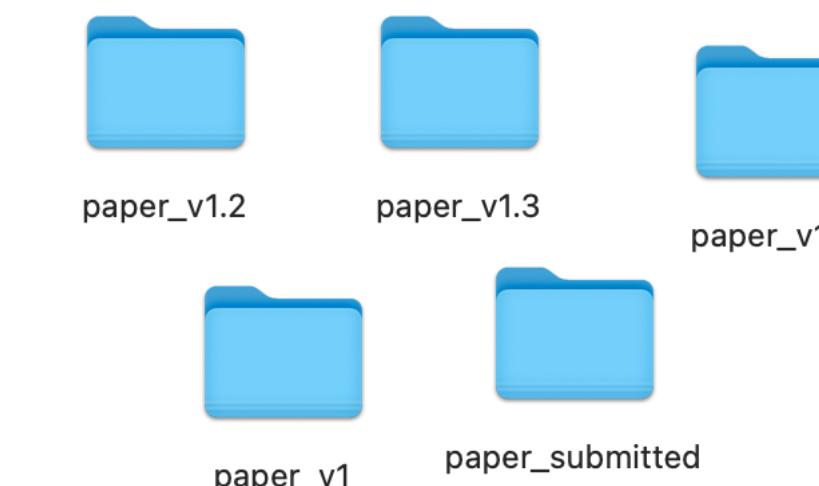
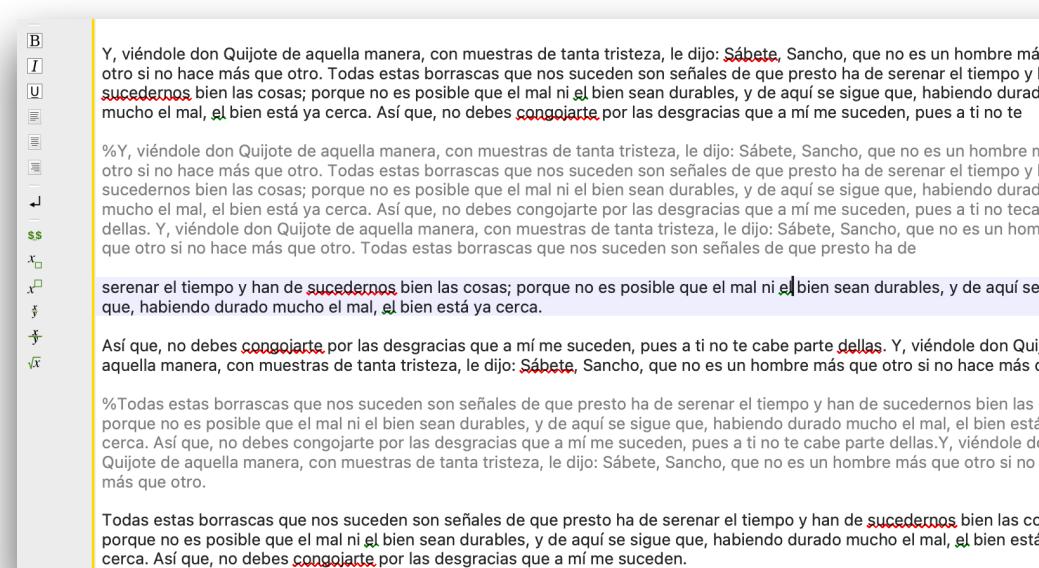
## Contexto

- Estamos preparando un artículo en Latex
- Somos el autor único (o el único que escribe)



## Posibles escenarios

- Eliminamos un párrafo que luego queremos volver a añadir
- Reescribimos un texto pero queremos volver a la versión anterior
- Queremos registrar la versión exacta enviada (y las revisiones)



## Comentarios

## Copias de seguridad manuales



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

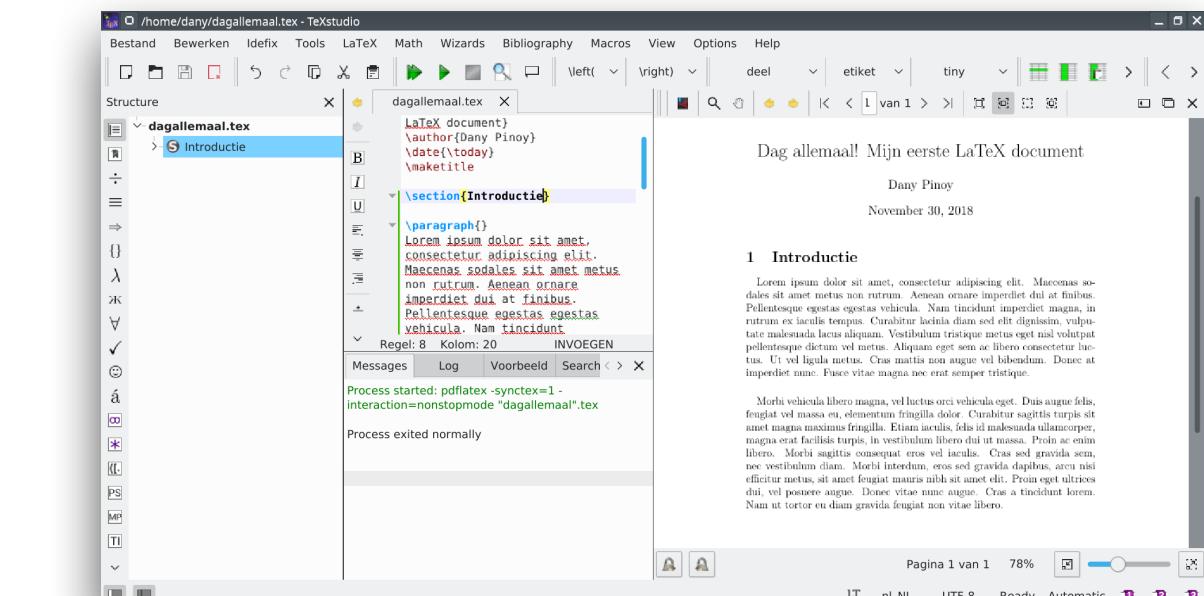
4. Correcciones

5. Ramas

6. Sincronización remota

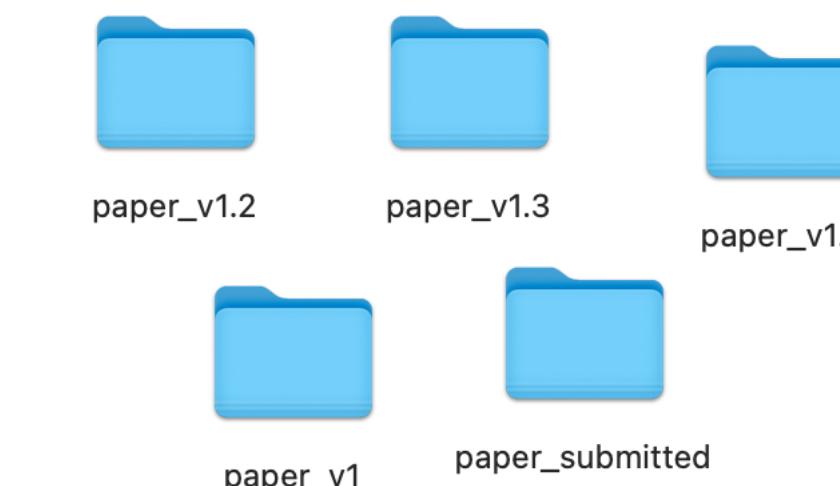
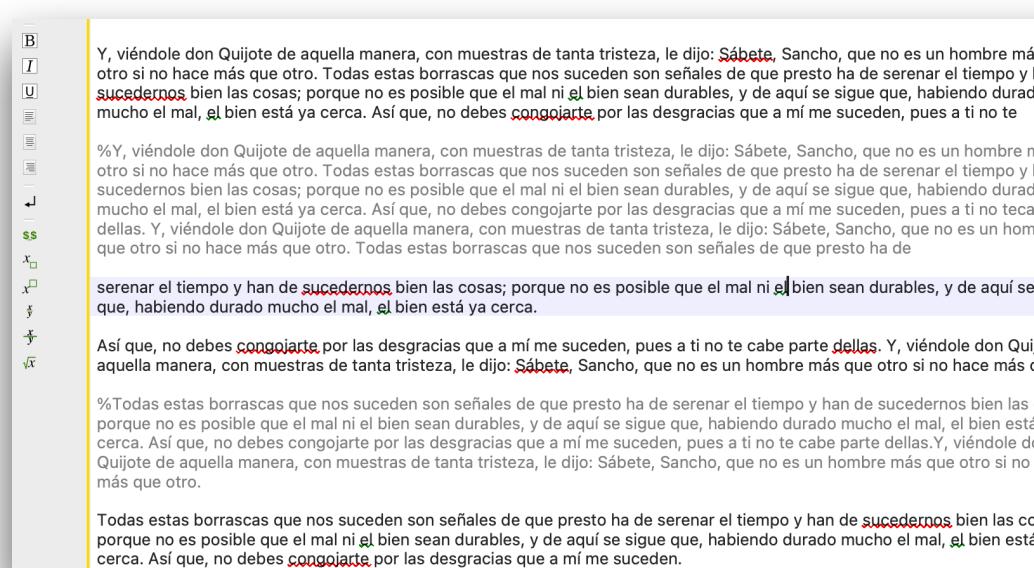
## Contexto

- Estamos preparando un artículo en Latex
- Somos el autor único (o el único que escribe)



## Posibles escenarios

- Eliminamos un párrafo que luego queremos volver a añadir
- Reescribimos un texto pero queremos volver a la versión anterior
- Queremos registrar la versión exacta enviada (y las revisiones)



## Comentarios

## Copias de seguridad manuales



# Commits

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

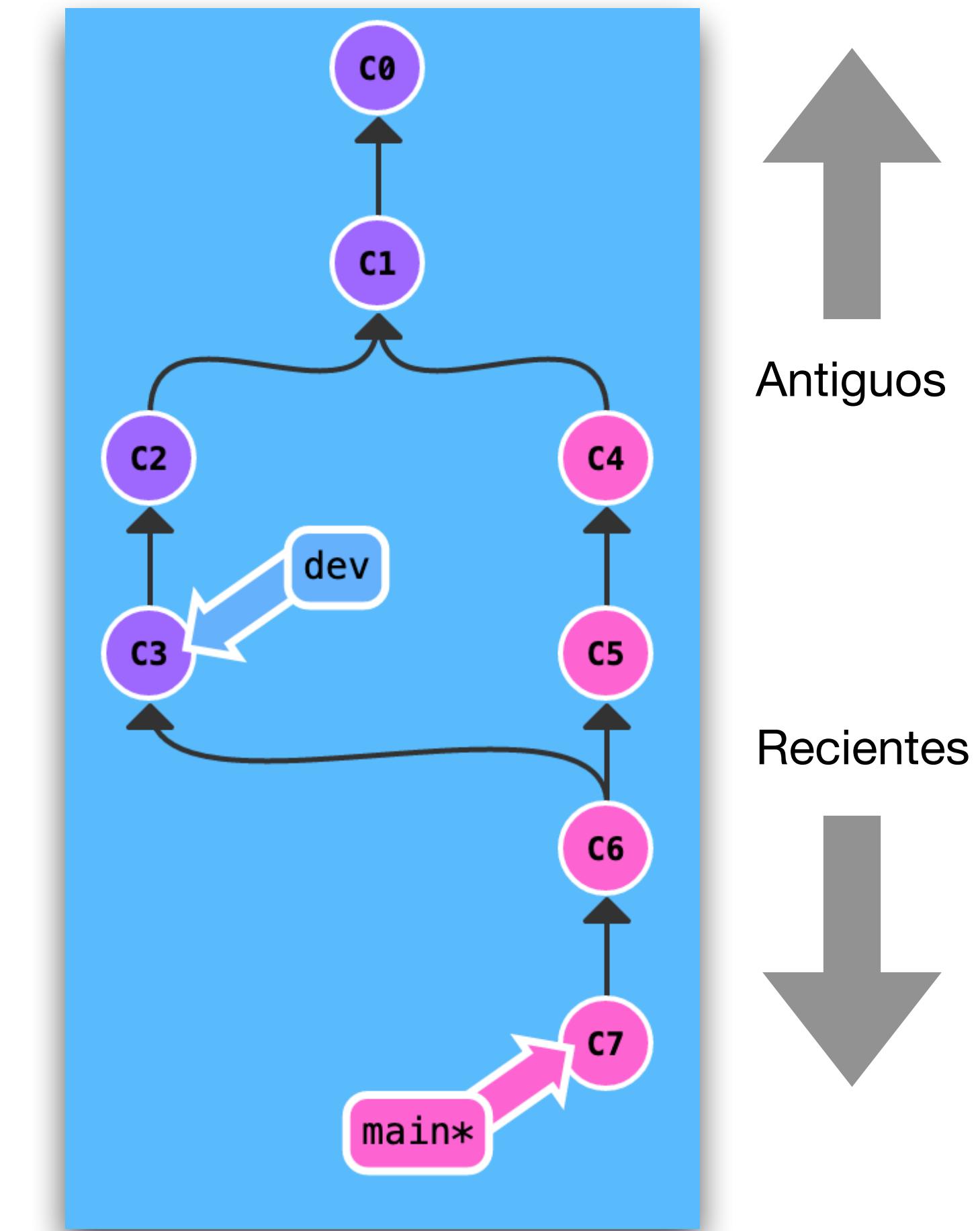
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Un **commit** es un conjunto de cambios en nuestro repositorio.
- Representa un estado concreto en la evolución del repositorio (sólo se almacena la variación)
- Cada uno tiene asociado un **identificador SHA1** único.
- Se estructuran en forma de **grafo dirigido**.
- Es posible volver a cualquier commit del pasado.





# Commits - GitHub Desktop

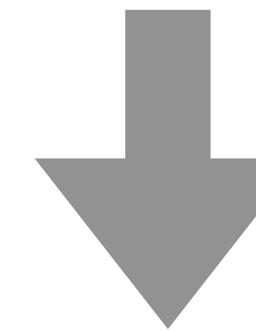
## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota

Los pasos para realizar un commit son:

1 Seleccionar los ficheros involucrados



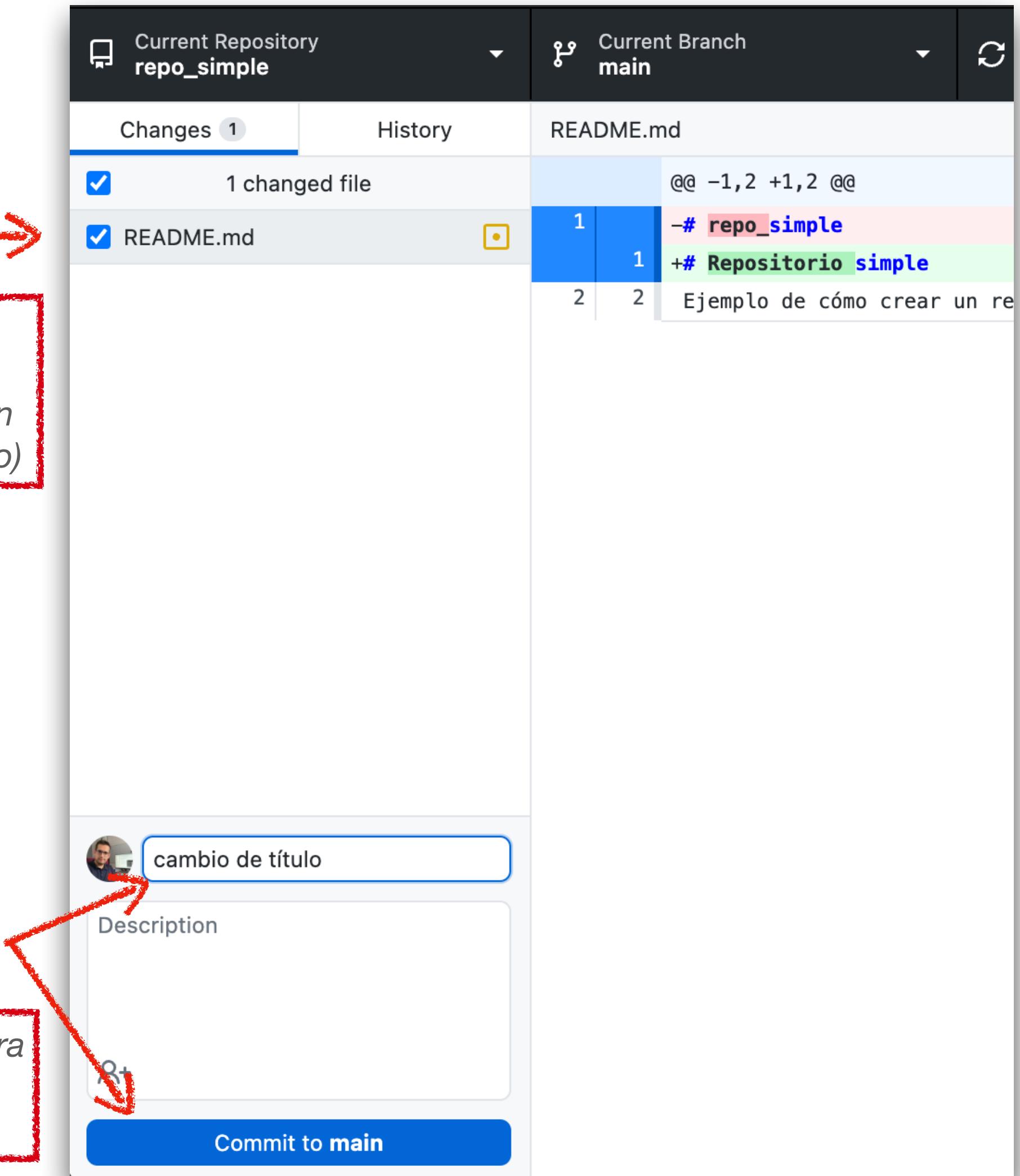
2 Confirmar los cambios

1 →

Marca los ficheros a incluir (todos los modificados aparecen marcados por defecto)

2

Indica un mensaje para el commit y pulsa el botón azul





# Commits - Ejercicio 3.1

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

En el fichero README.md del mismo repositorio. Añade las siguientes modificaciones y guárdalas en **2 commits diferentes**.

```
# Repositorio simple
Ejemplo de cómo crear un repositorio desde la web
Segundo cambio
Tercer cambio
```

Current Repository: repo\_simple

Current Branch: main

Changes 1 History

1 changed file

README.md

1	1	@@ -1,2 +1,2 @@
2	2	-# repo_simple
		+# Repositorio simple
Ejemplo de cómo crear un re		

cambio de título

Description

Commit to main



# Commits - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

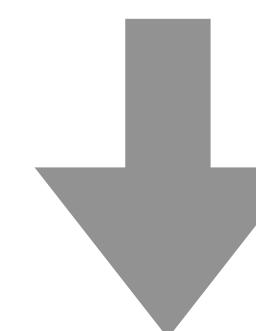
4. Correcciones

5. Ramas

6. Sincronización  
remota

Los pasos para realizar un commit son:

1 Seleccionar los  
ficheros involucrados



2 Confirmar los  
cambios

- El comando básico (sólo el PASO 2) es el siguiente:

\$\_ git commit -m "texto descriptivo"

- Si queremos que además haga el PASO 1 hay que añadir la opción -a

\$\_ git commit -a -m "texto descriptivo"

- Para obtener ayuda:

\$\_ git commit --help



# Commits

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

```
● ● ● README.md — Edited  
# repo_simple  
Ejemplo de cómo crear un repositorio desde la web
```

```
● ● ● README.md — Edited  
# Repositorio simple  
Ejemplo de cómo crear un repositorio desde la web
```

```
● ● ● README.md — Edited  
# Repositorio simple  
Ejemplo de cómo crear un repositorio desde la web  
segundo cambio
```

```
● ● ● README.md — Edited  
# Repositorio simple  
Ejemplo de cómo crear un repositorio desde la web  
segundo cambio  
tercer cambio
```

\$ \_

```
git commit -a -m "cambio de título"
```

\$ \_

```
git commit -a -m "segundo"
```

\$ \_

```
git commit -a -m "tercer"
```



# Commits - Ejercicio 3.2

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

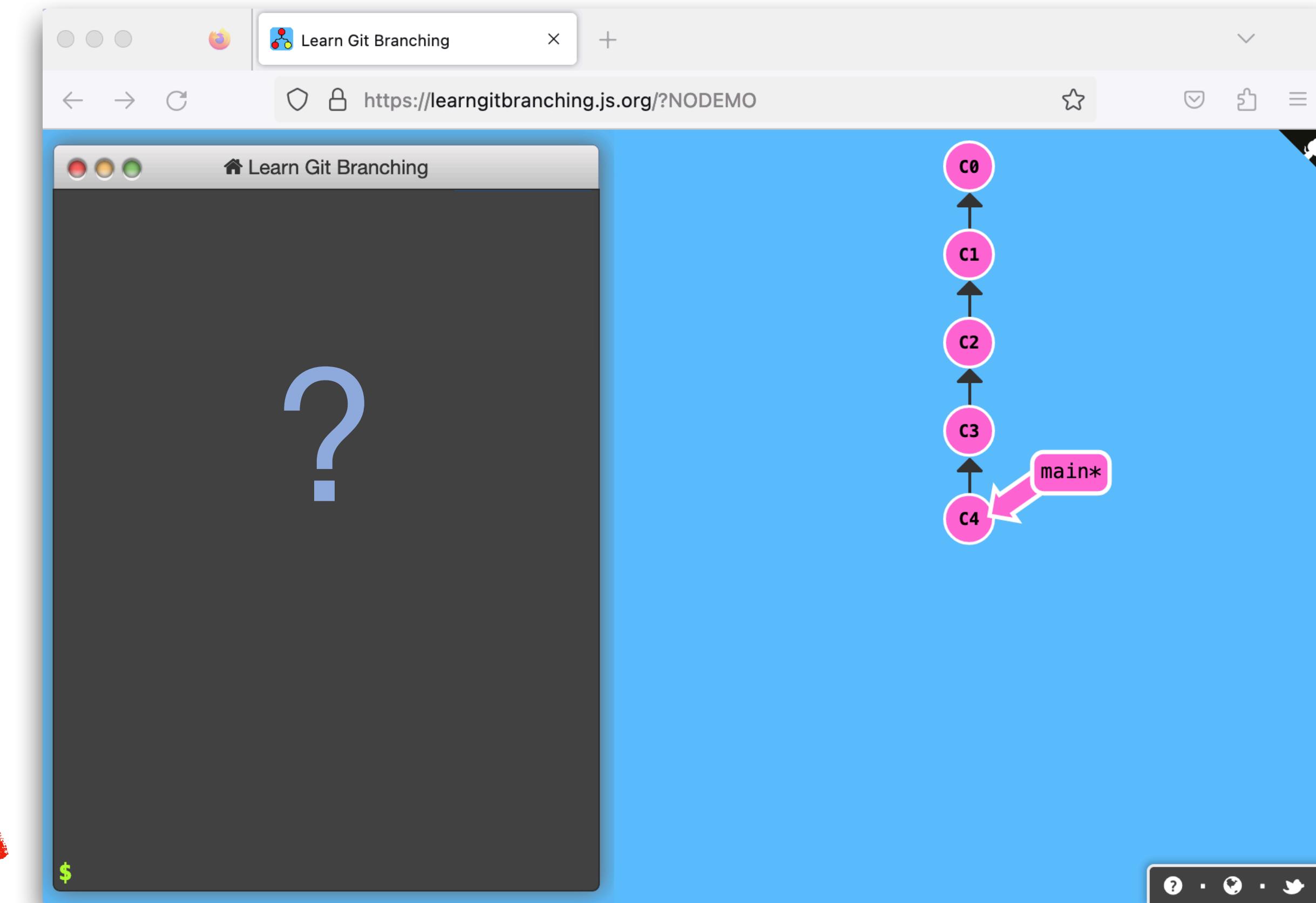
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

En el simulador online <https://learngitbranching.js.org/?NODEMO> teclea los comandos necesarios para generar el grafo de commits que se muestra en la imagen.





# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

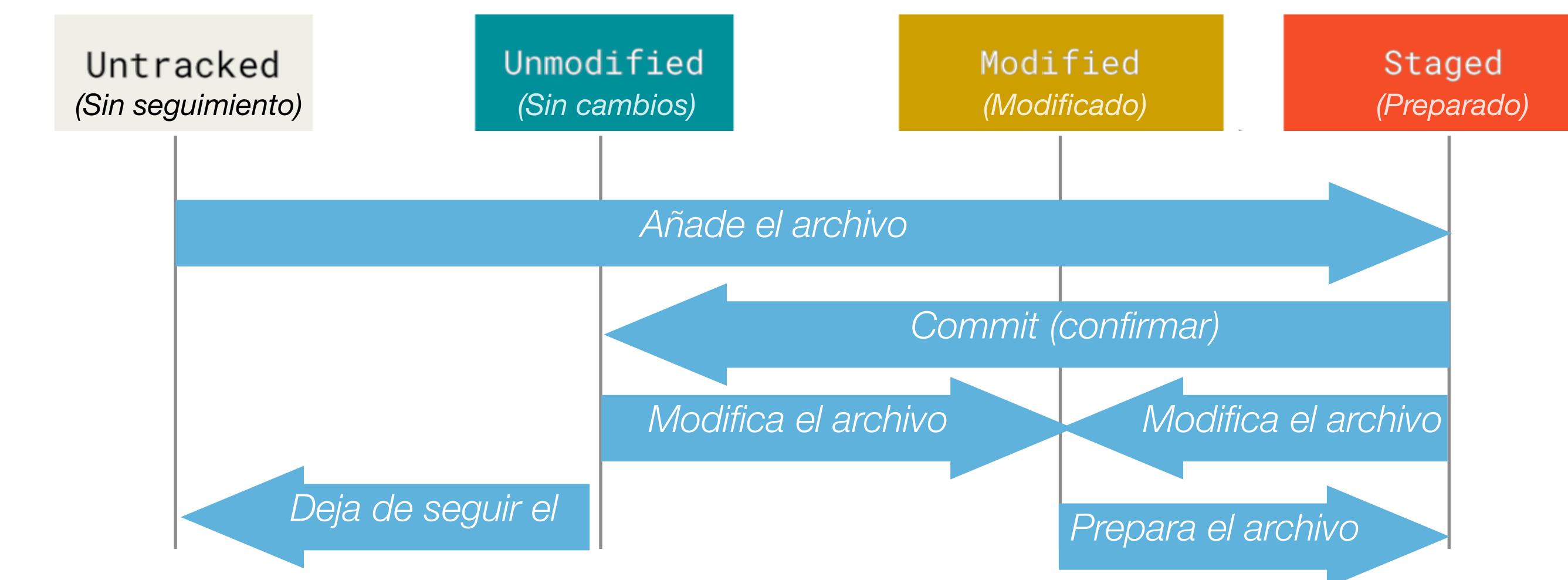
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- **Untracked (*sin seguimiento*):** ficheros que están en el directorio del repositorio (o en un subdirectorio) pero que nunca han sido controlados por git (e.g, nuevos ficheros)
- **Unmodified (*sin cambios*):** ficheros controlados por git pero sin cambios nuevos.
- **Modified (*modificado*):** ficheros controlados por git con cambios no registrados.
- **Staged (*preparado*):** ficheros controlados por git con cambios registrados, es decir, preparados para meterlos en un commit.





# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

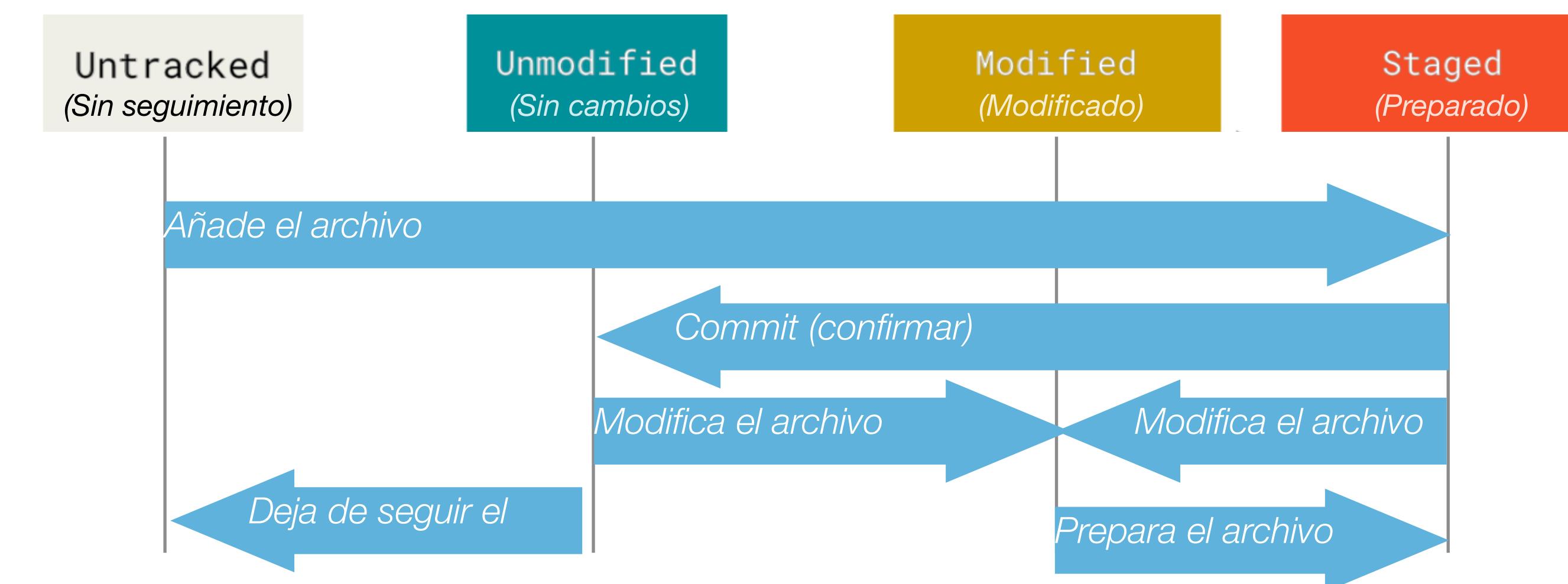
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

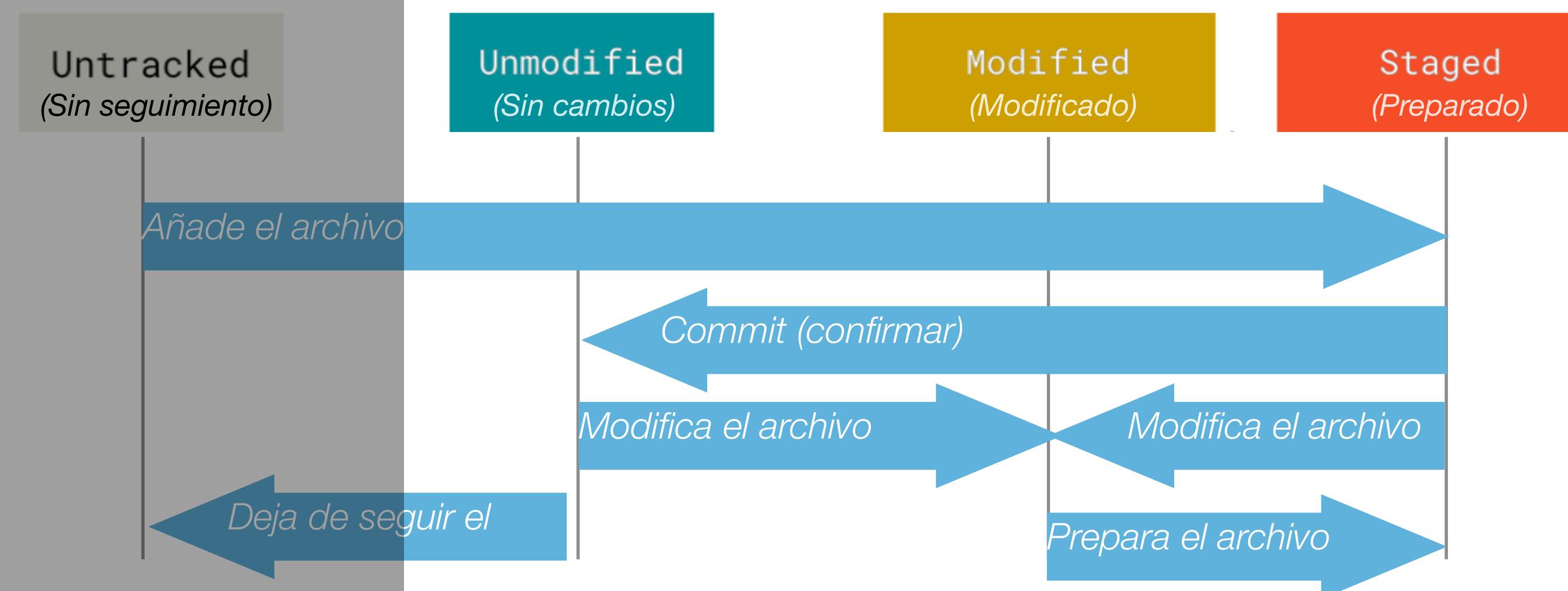
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## Zona desconocida





# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

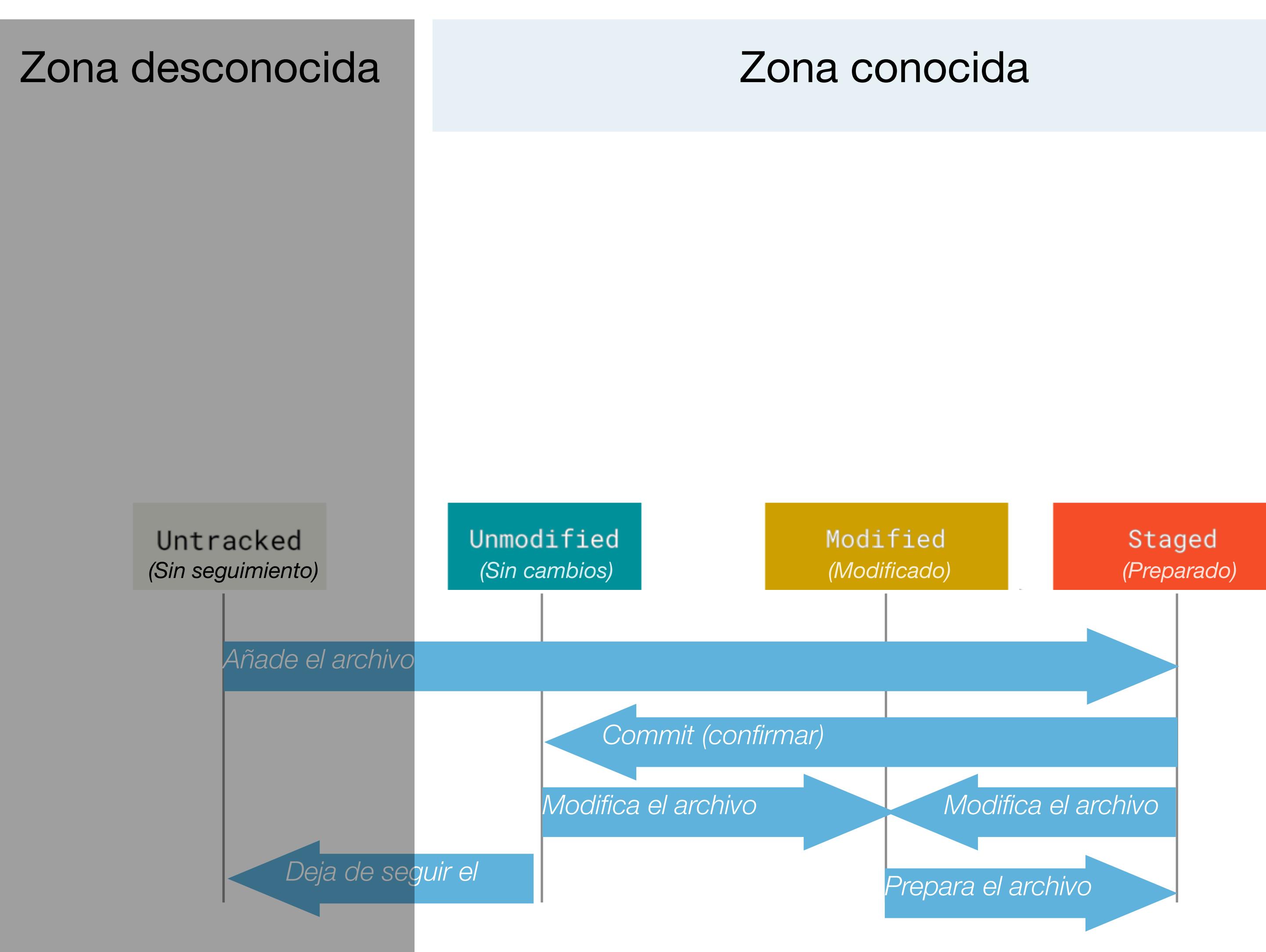
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

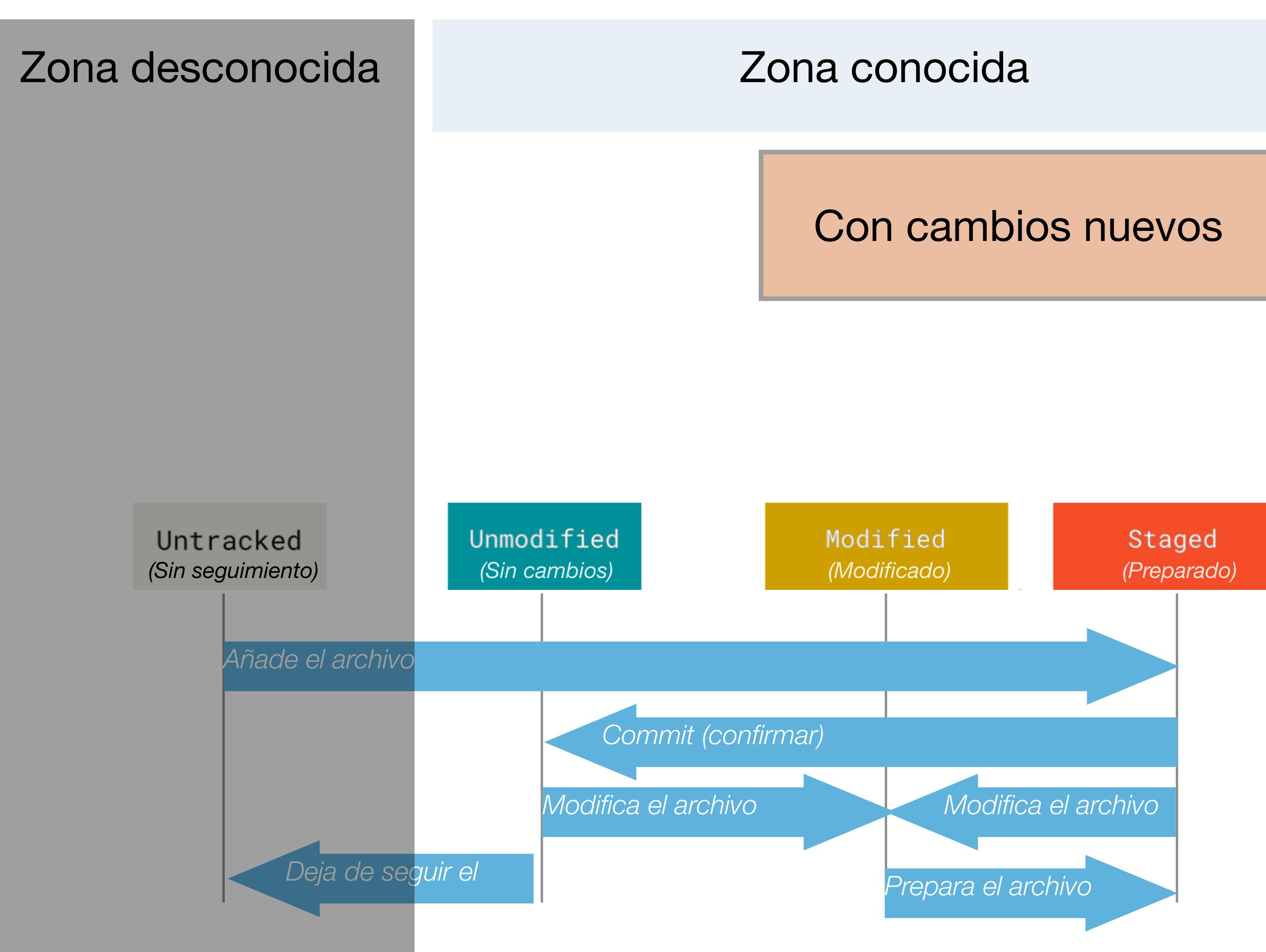
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



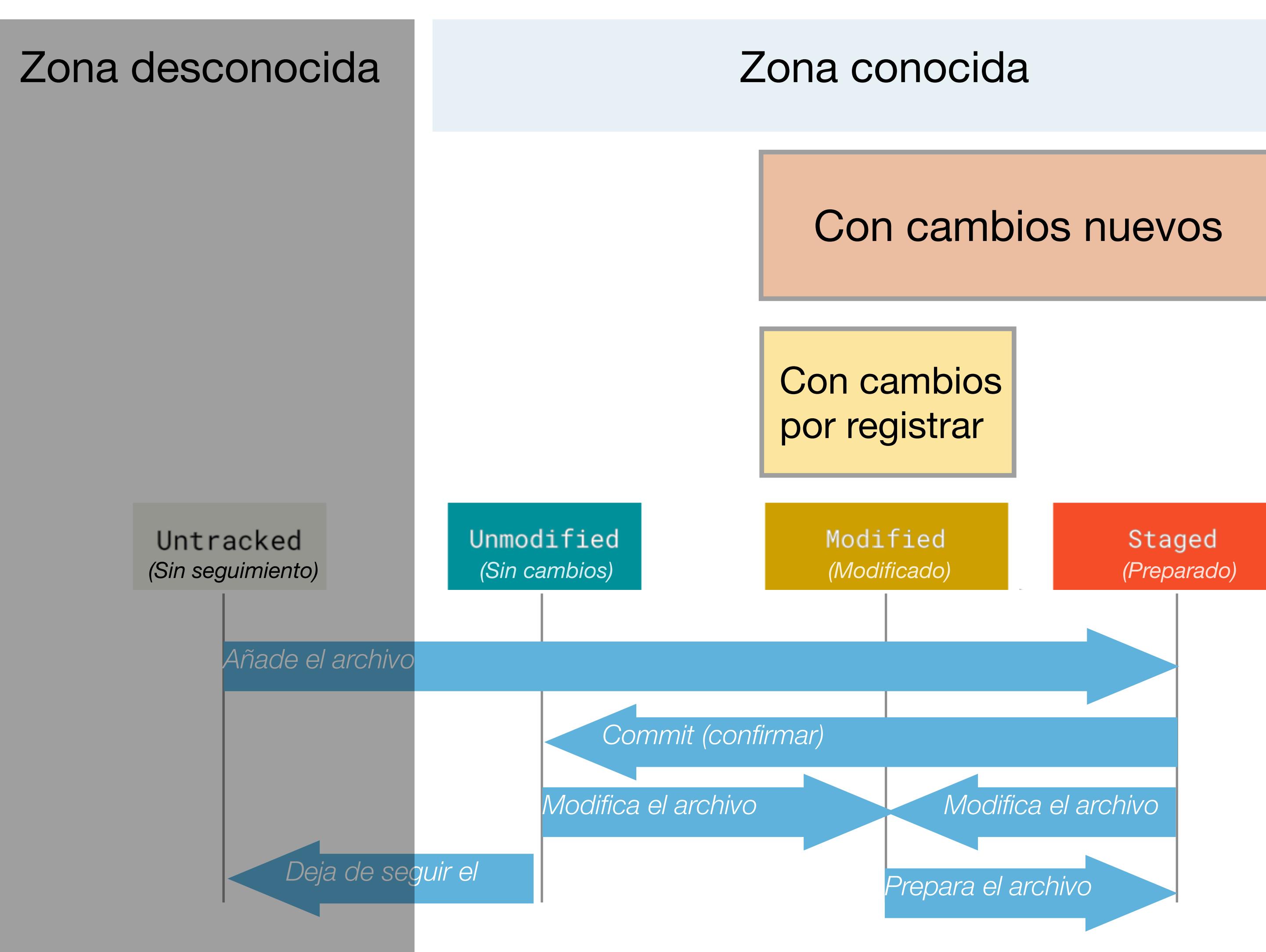


# Ciclo de vida

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota





# Ciclo de vida - Fichero controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

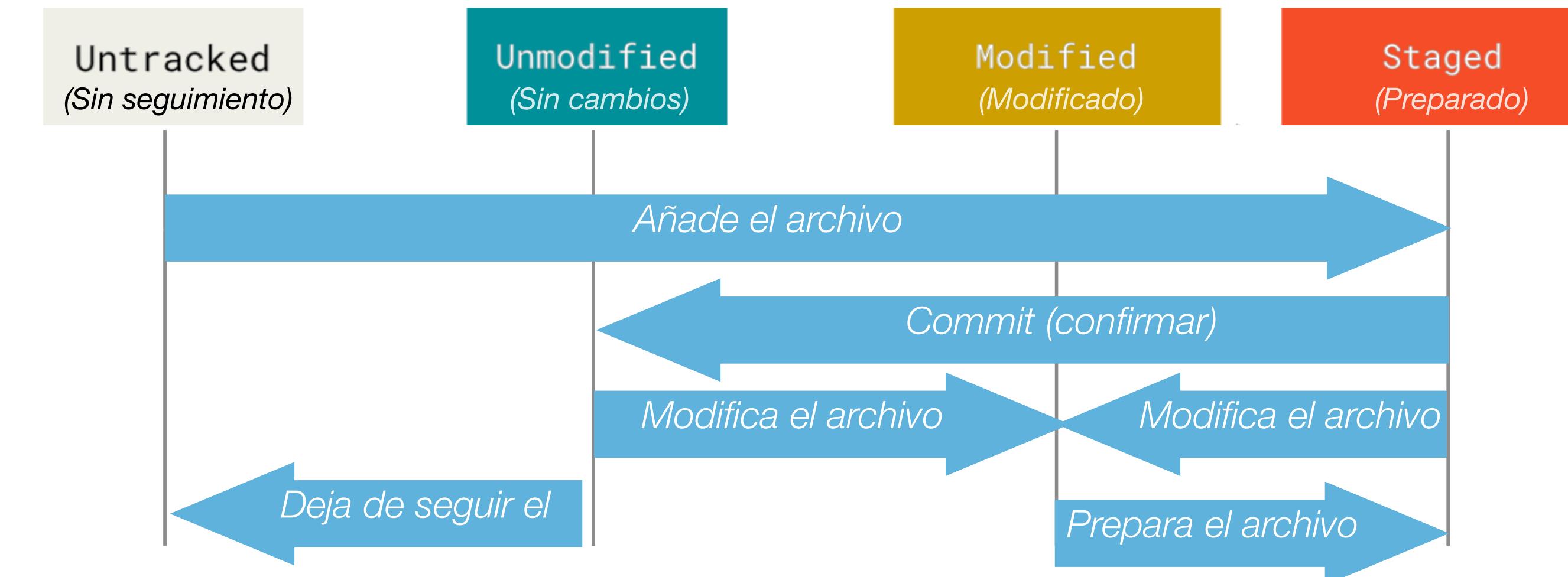
4. Correcciones

5. Ramas

6. Sincronización  
remota



README.md





# Ciclo de vida - Fichero controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

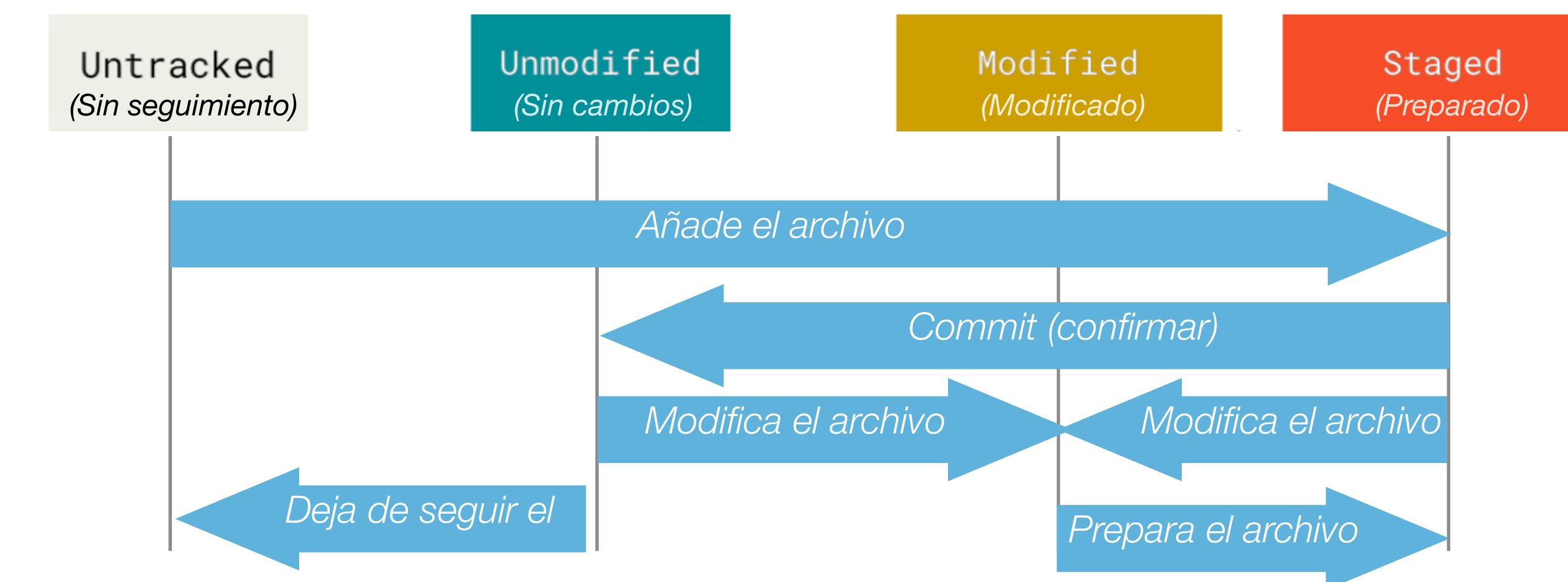
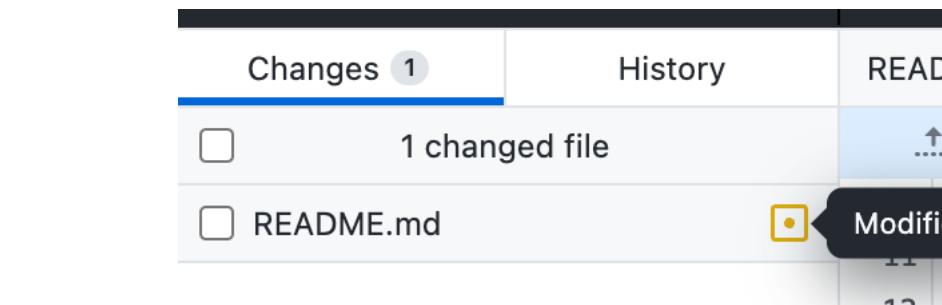
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida - Fichero controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

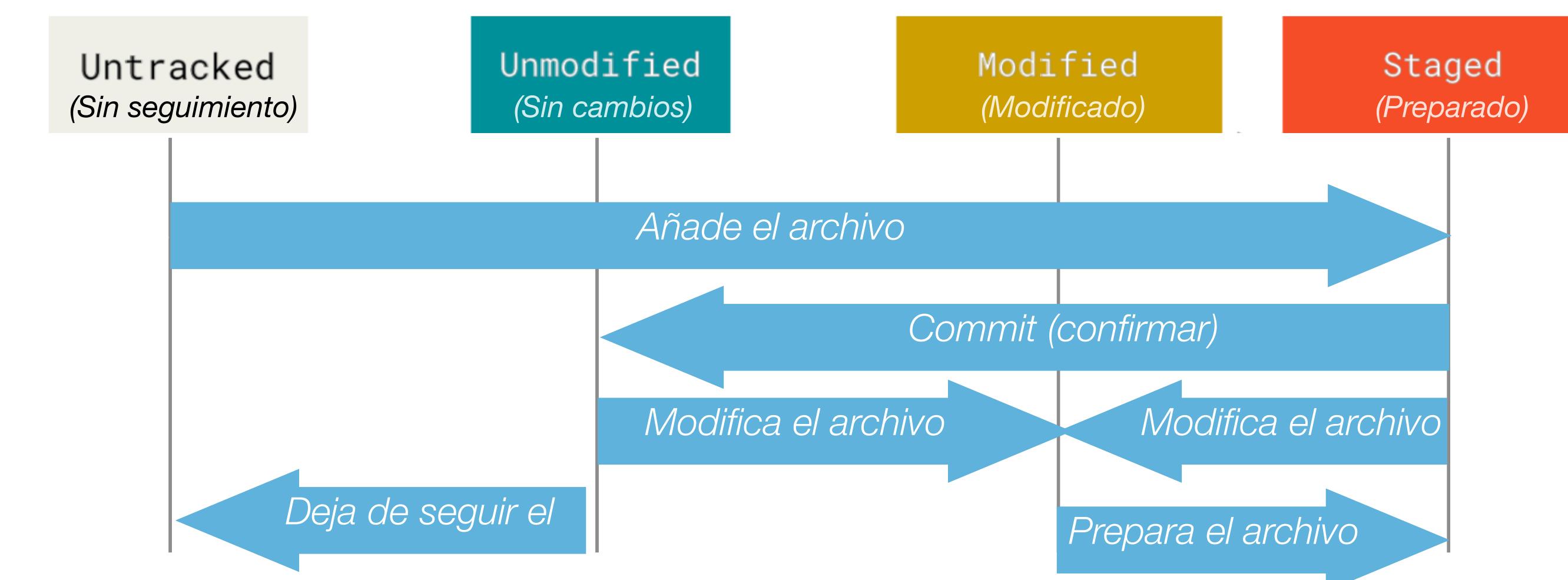
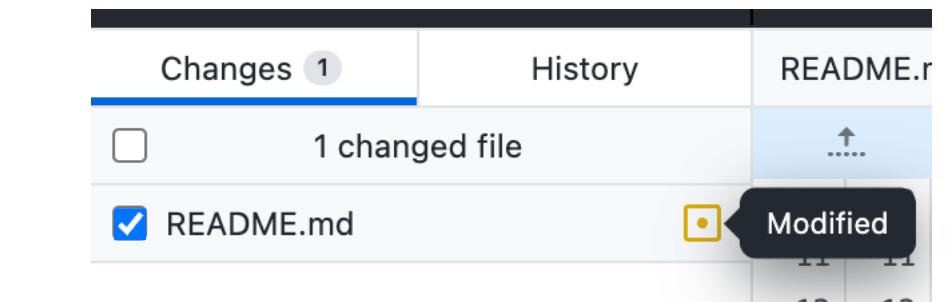
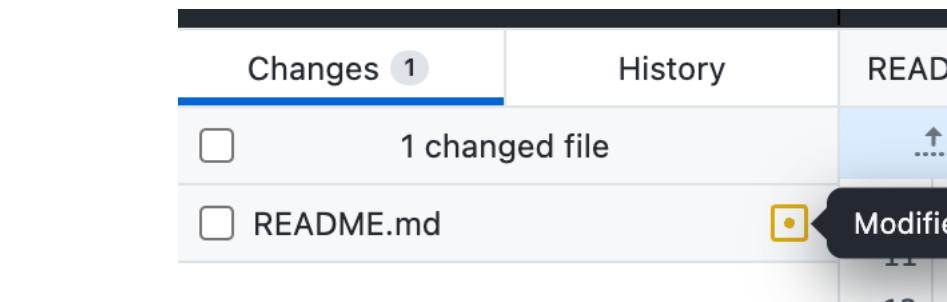
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida - Fichero controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

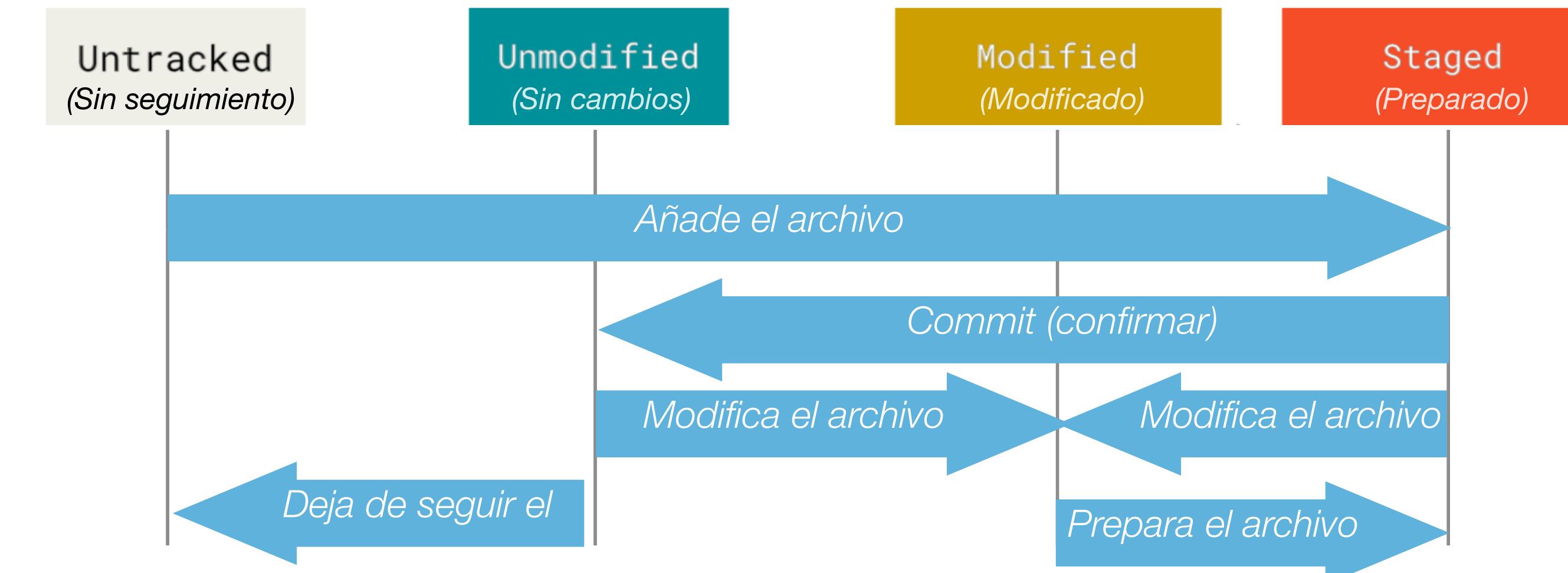
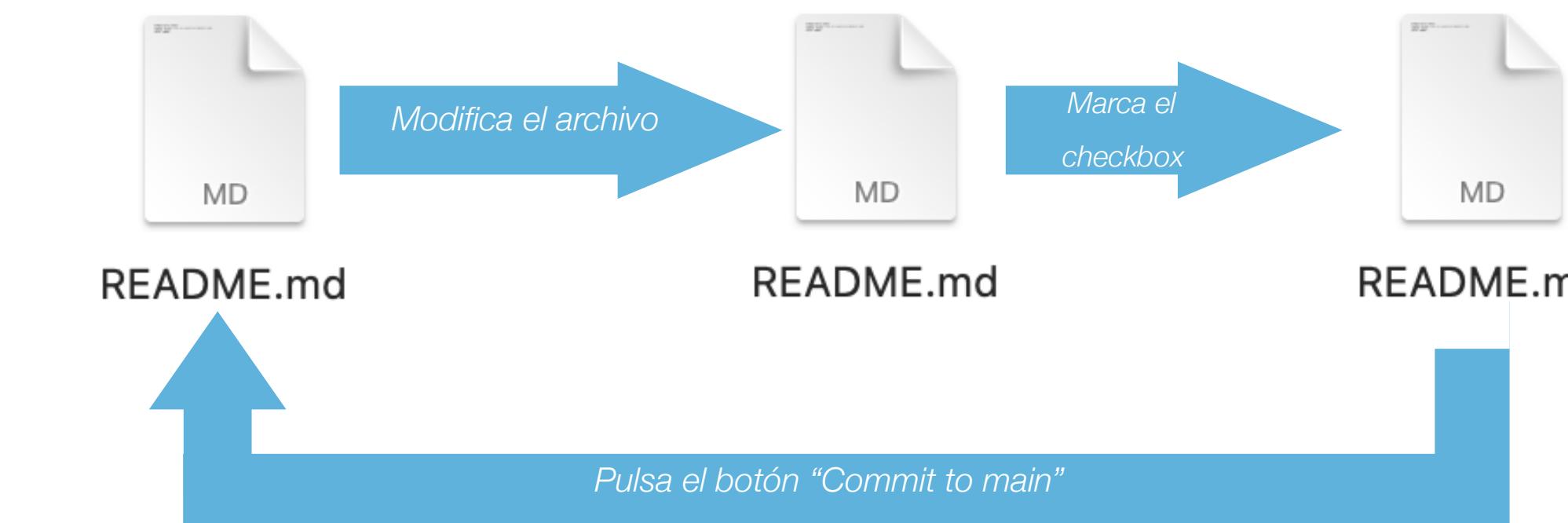
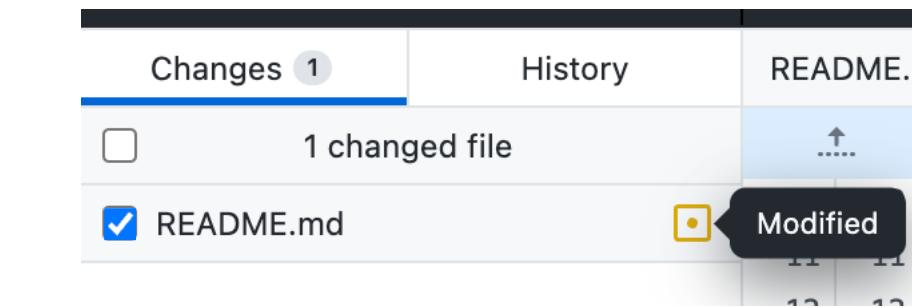
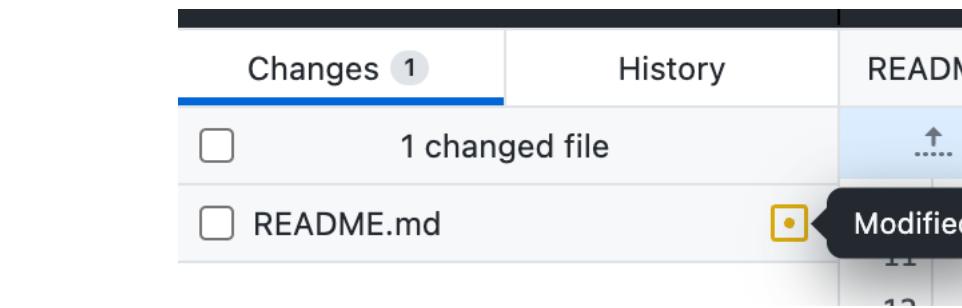
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida - Fichero no controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

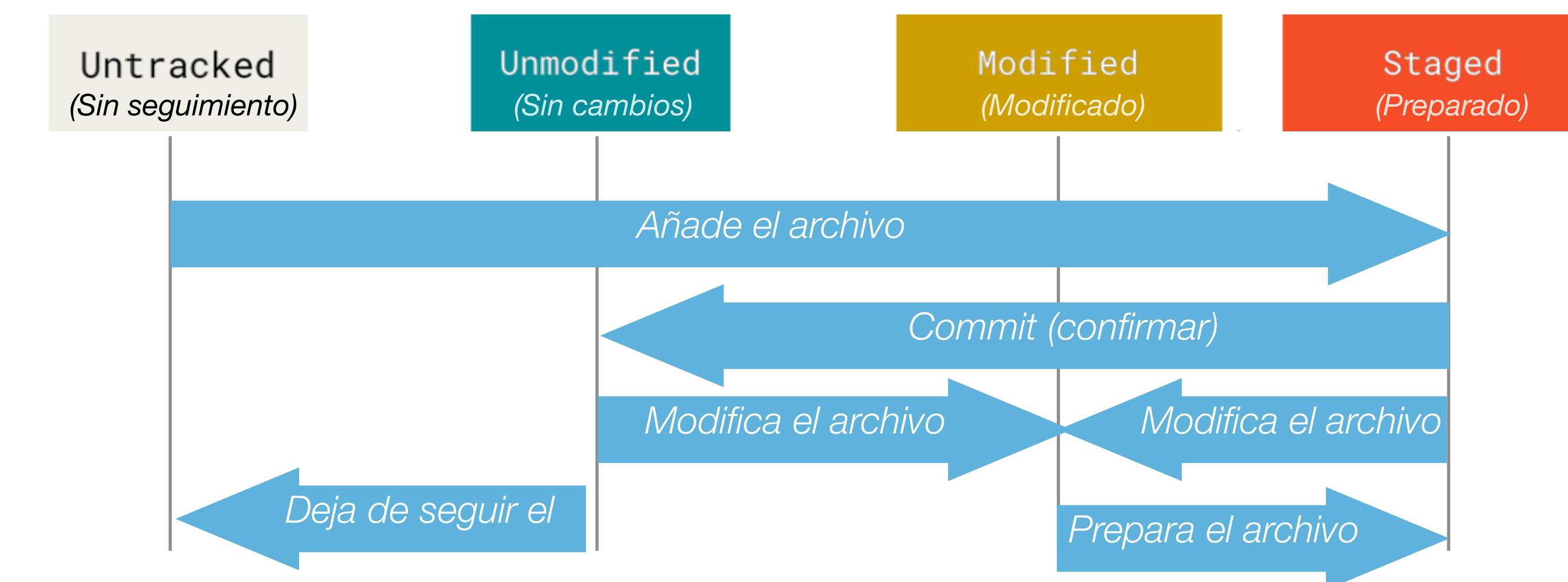
4. Correcciones

5. Ramas

6. Sincronización  
remota



nuevo.txt





# Ciclo de vida - Fichero no controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

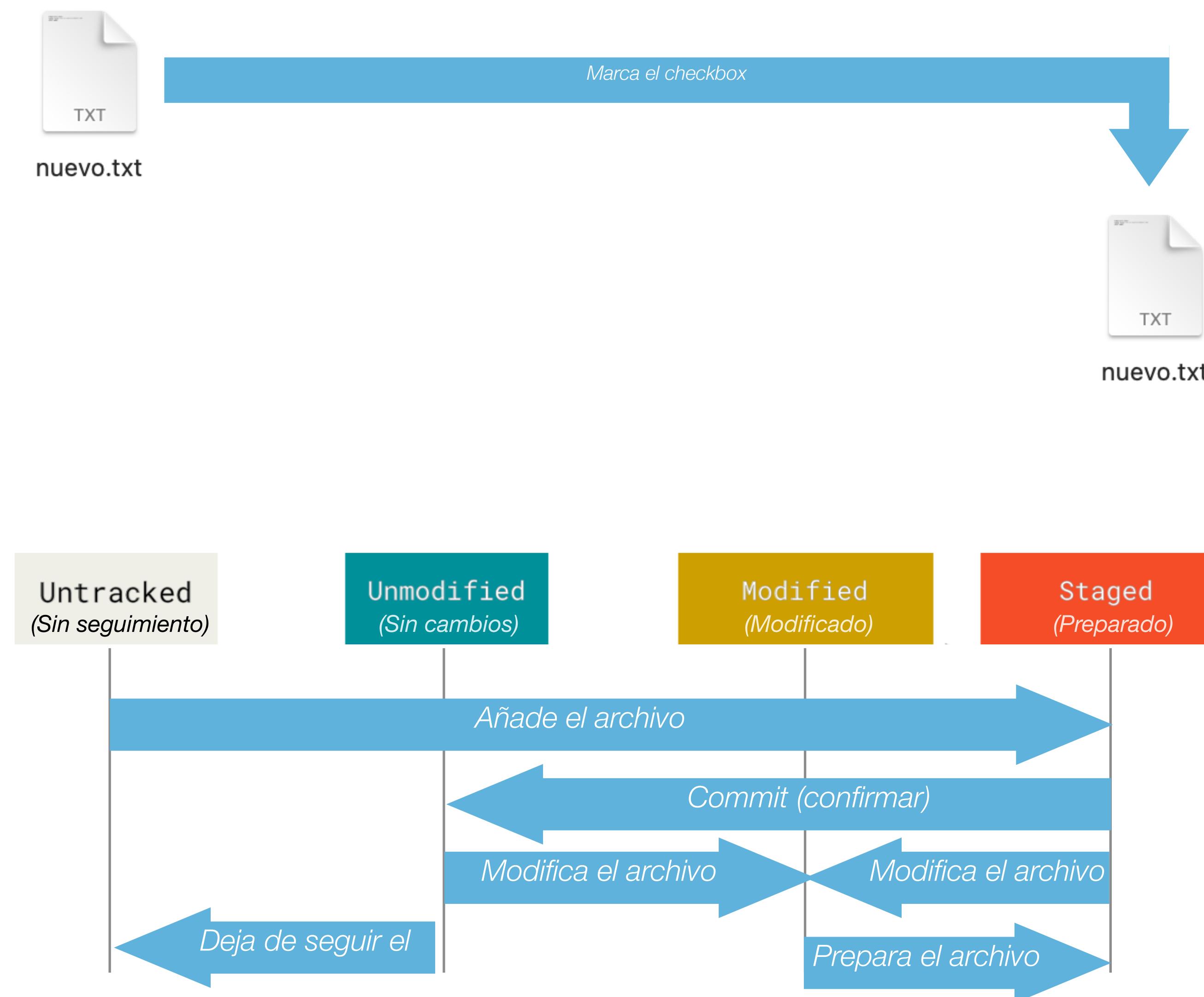
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida - Fichero no controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

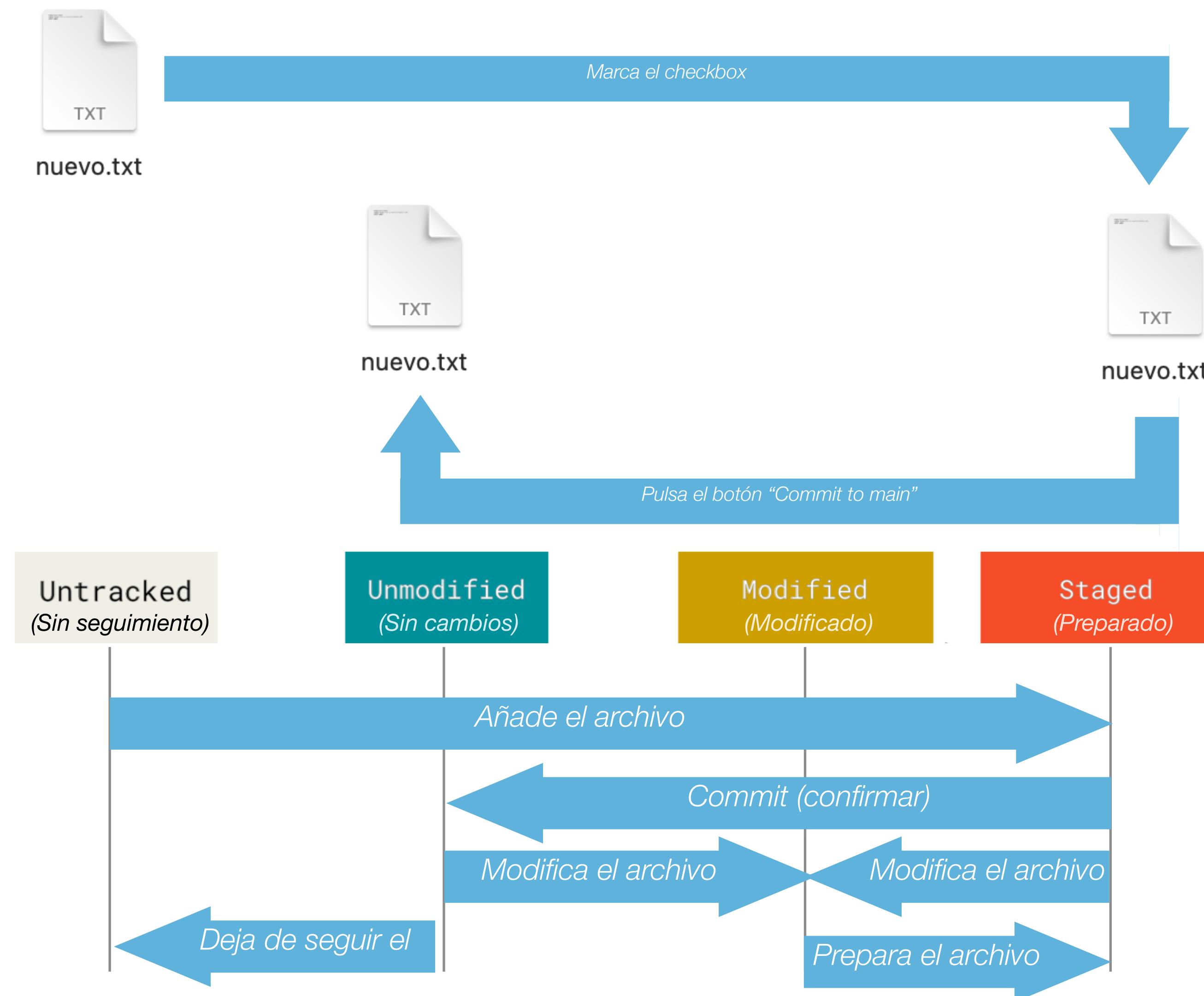
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Ciclo de vida - Fichero no controlado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

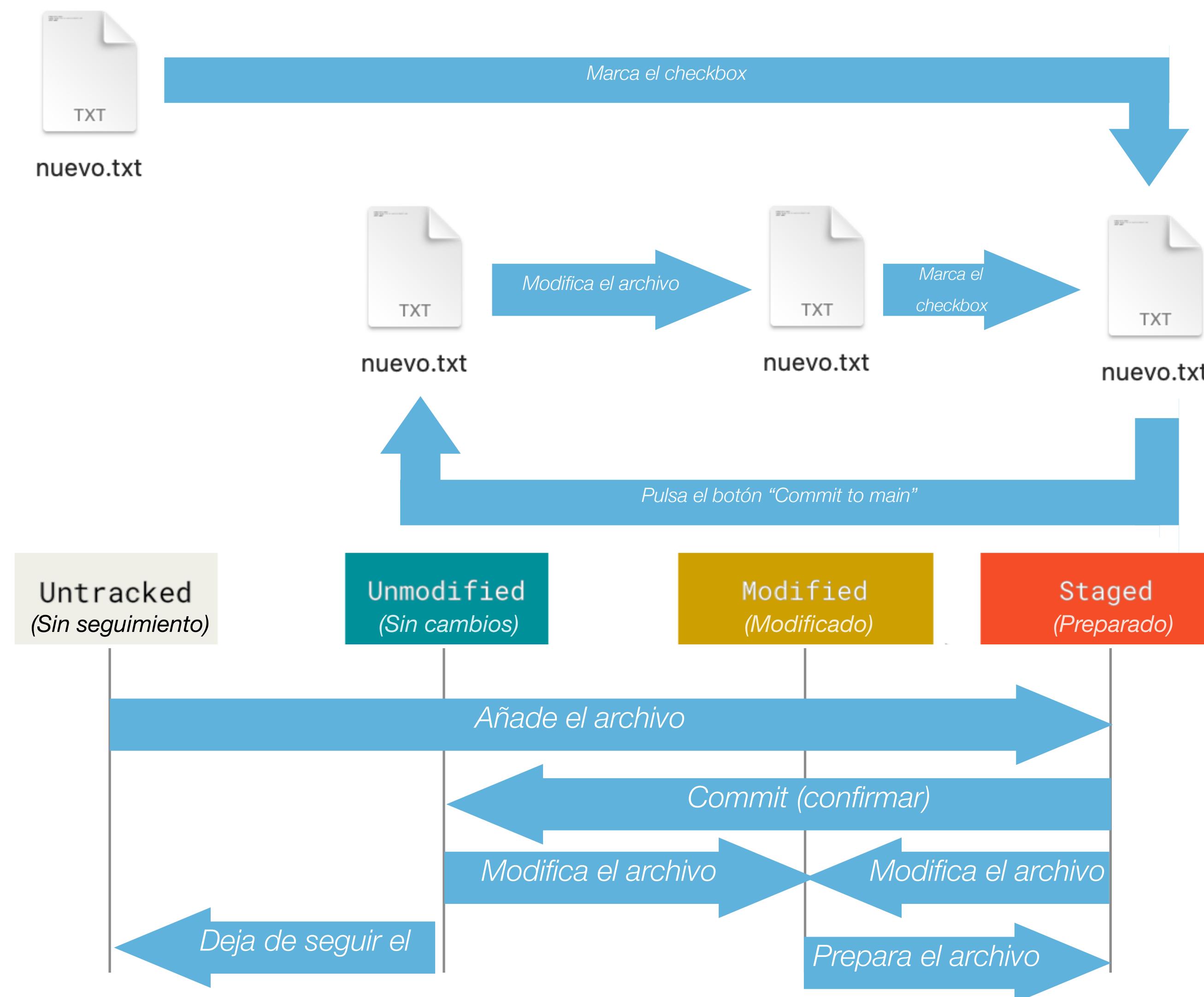
2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota





# Ciclo de vida - Ejercicio 3.3

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



Crea un fichero de texto plano **nuevo.txt** y utilizando GitHub Desktop realiza todas las acciones necesarias para que pase por todas las etapas del ciclo de vida.



nuevo.txt

Untracked  
*(Sin seguimiento)*

Unmodified  
*(Sin cambios)*

Modified  
*(Modificado)*

Staged  
*(Preparado)*



# Ciclo de vida - Ejercicio 3.3

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

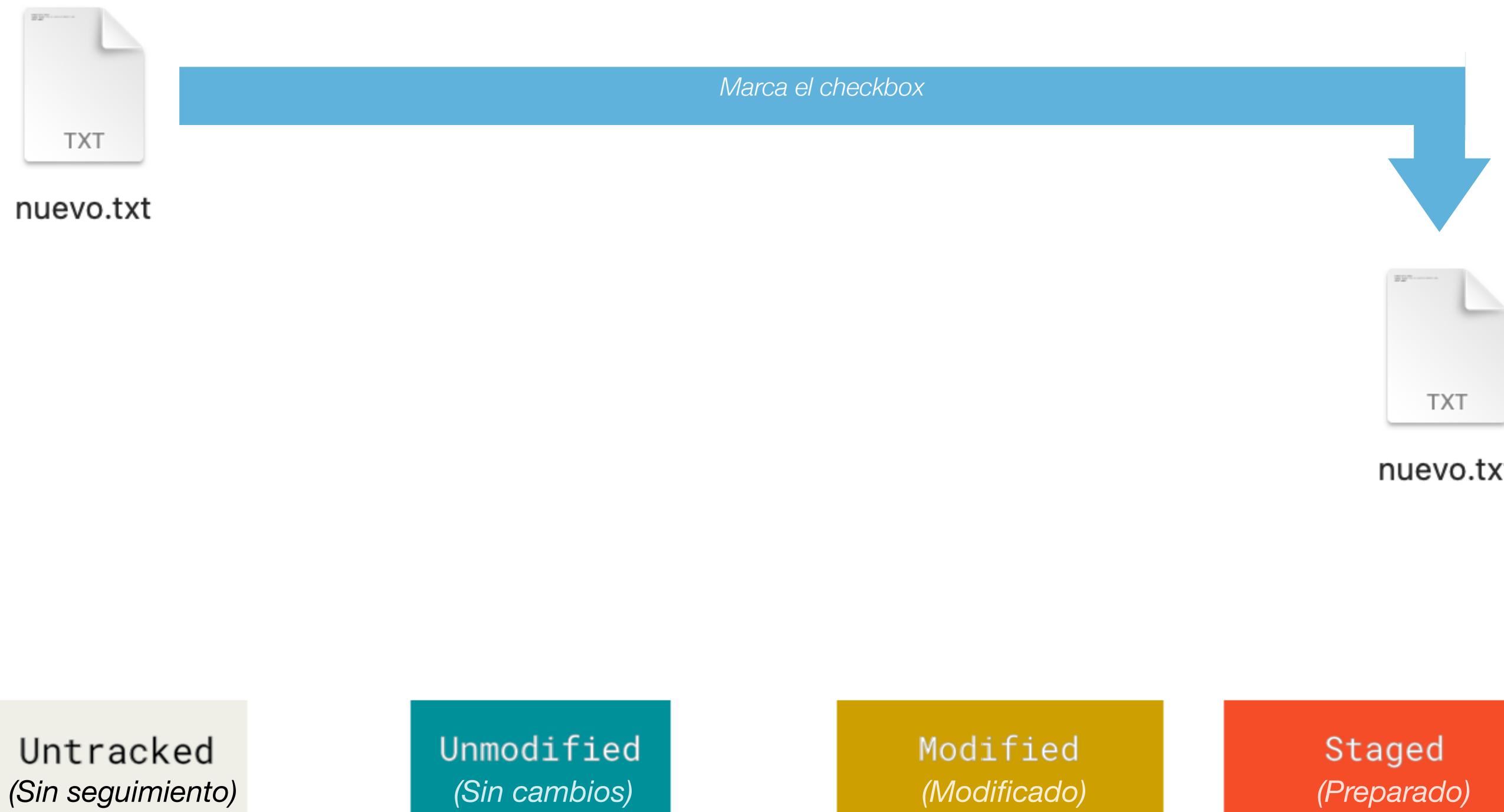
4. Correcciones

5. Ramas

6. Sincronización remota



Crea un fichero de texto plano **nuevo.txt** y utilizando GitHub Desktop realiza todas las acciones necesarias para que pase por todas las etapas del ciclo de vida.





# Ciclo de vida - Ejercicio 3.3

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

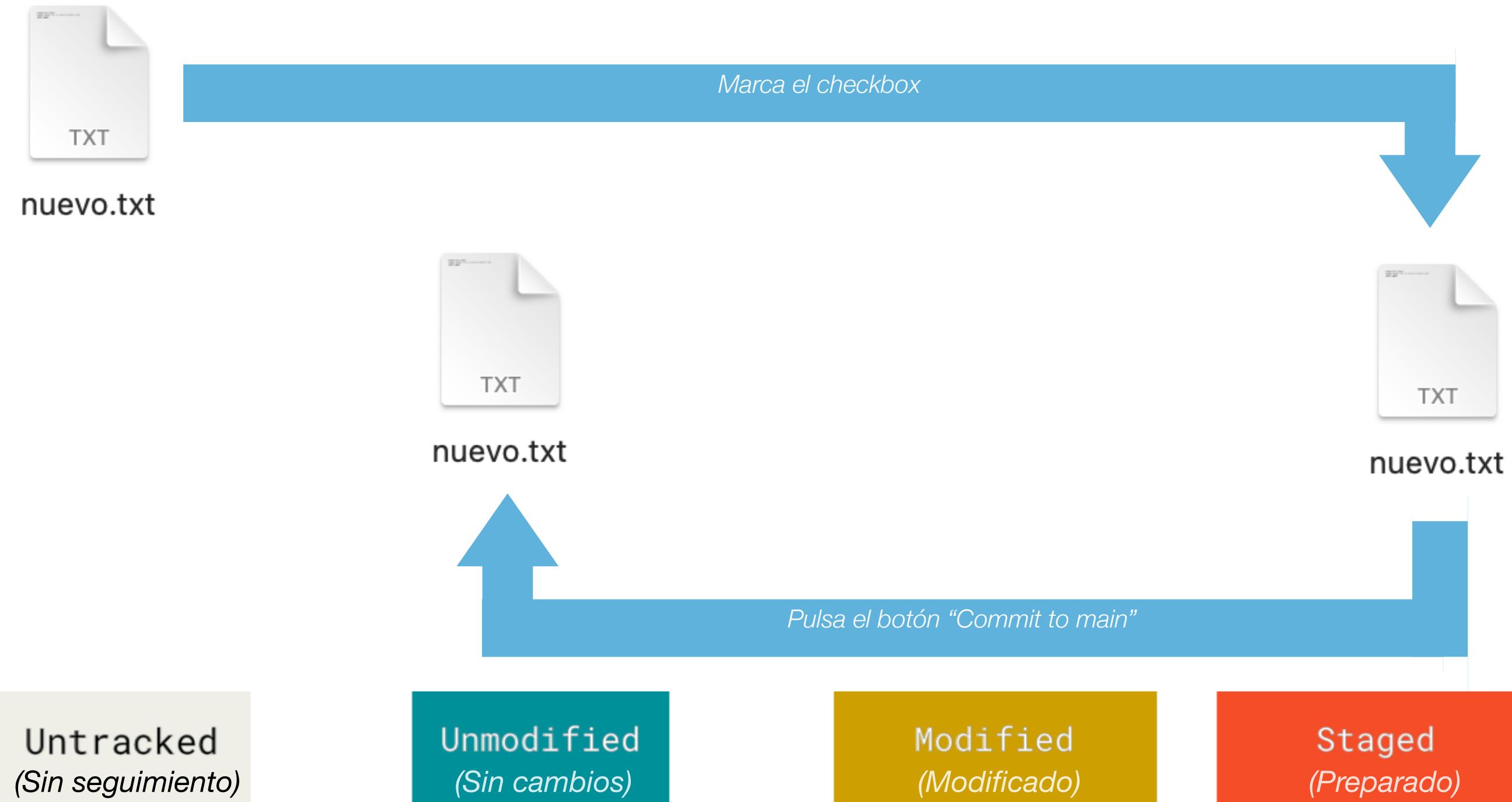
4. Correcciones

5. Ramas

6. Sincronización  
remota



Crea un fichero de texto plano **nuevo.txt** y utilizando GitHub Desktop realiza todas las acciones necesarias para que pase por todas las etapas del ciclo de vida.





# Ciclo de vida - Ejercicio 3.3

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

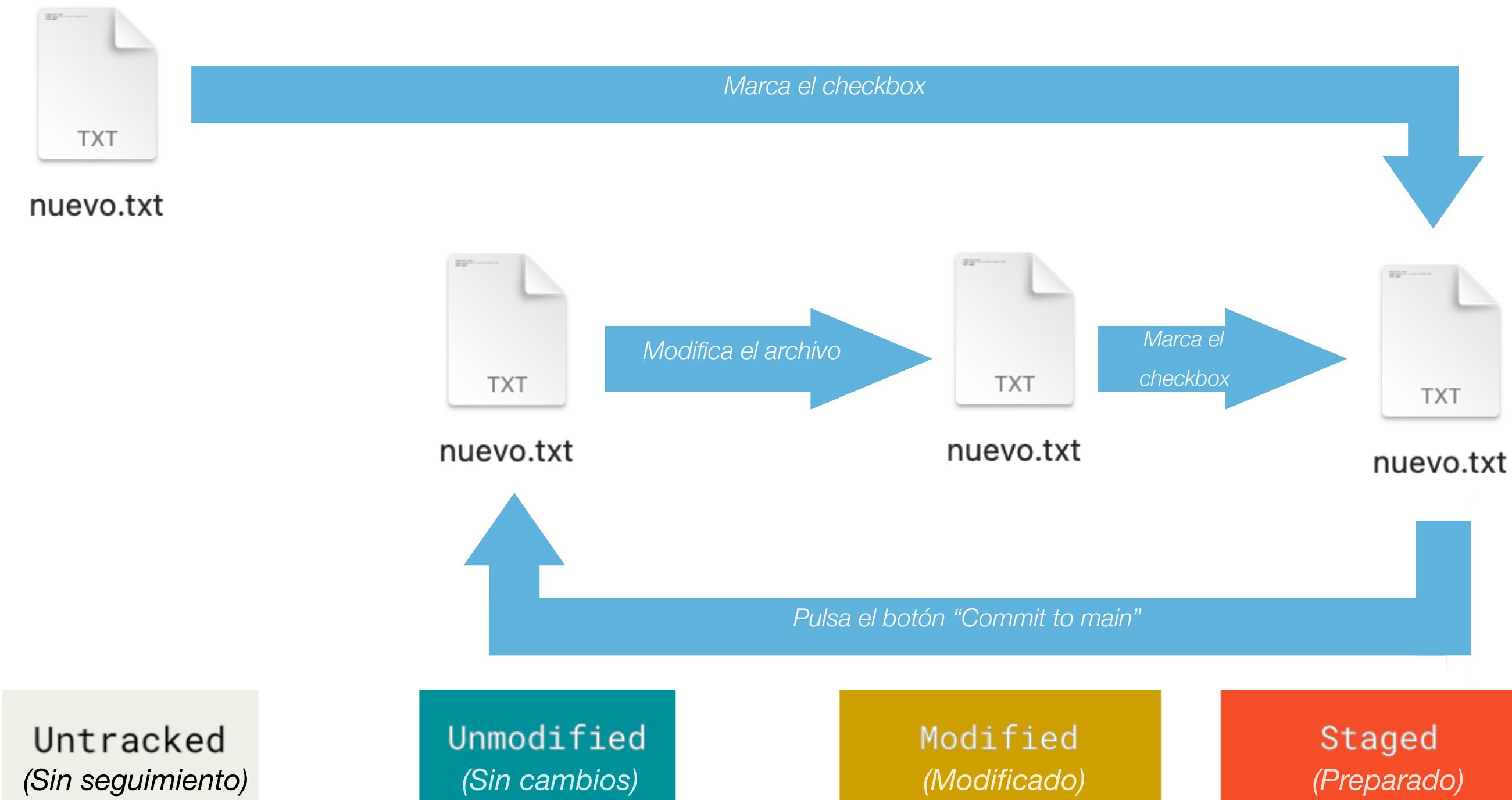
4. Correcciones

5. Ramas

6. Sincronización remota



Crea un fichero de texto plano **nuevo.txt** y utilizando GitHub Desktop realiza todas las acciones necesarias para que pase por todas las etapas del ciclo de vida.





# Ciclo de vida - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Para enviar a la zona de **preparado**, ya sea desde **sin seguimiento** o desde **modificado**,  
se usa:

\$ \_

```
git add archivo.txt
```

\$ \_

```
git add directorio
```

\$ \_

```
git add .
```

\$ \_

```
git add ..
```

\$ \_

```
git add ../../..
```



# Ciclo de vida - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

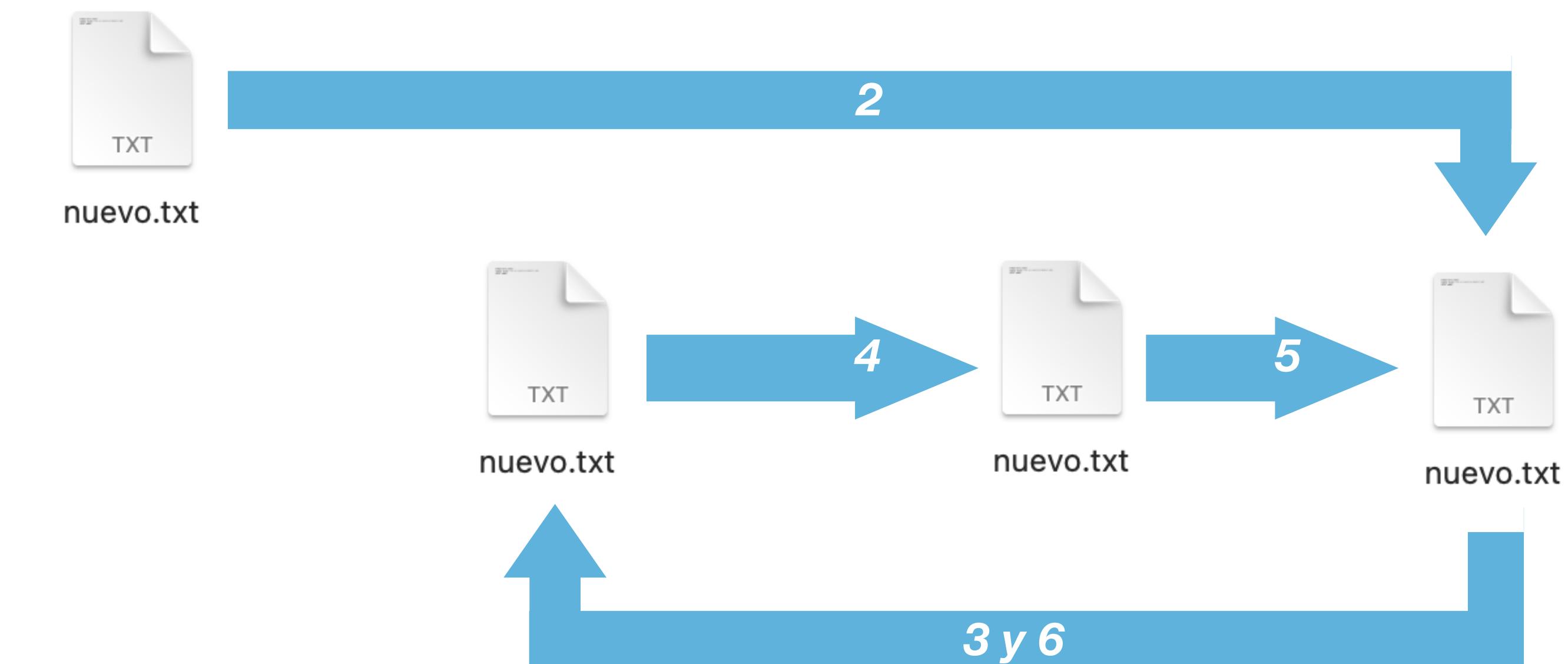
4. Correcciones

5. Ramas

6. Sincronización remota

\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add nuevo2.txt
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git add nuevo2.txt
6 git commit -m "cambios"
```



Untracked  
(Sin seguimiento)

Unmodified  
(Sin cambios)

Modified  
(Modificado)

Staged  
(Preparado)



# Ciclo de vida - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

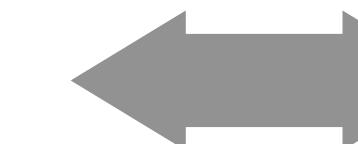
Las siguientes secuencias de comandos son equivalentes (producen los mismos resultados). Encuentra las diferencias.

\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add nuevo2.txt
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git add nuevo2.txt
6 git commit -m "cambios"
```

\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add .
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git commit -a -m "cambios"
```





# Ciclo de vida - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

Las siguientes secuencias de comandos son equivalentes (producen los mismos resultados). Encuentra las diferencias.

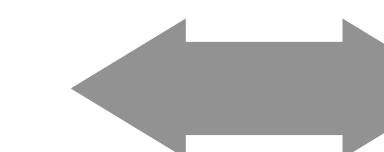
\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add nuevo2.txt
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git add nuevo2.txt
6 git commit -m "cambios"
```

\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add . ←
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git commit -a -m "cambios"
```

Con el punto  
se añaden  
todos los  
ficheros  
nuevos o  
modificados





# Ciclo de vida - Terminal

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

Las siguientes secuencias de comandos son equivalentes (producen los mismos resultados). Encuentra las diferencias.

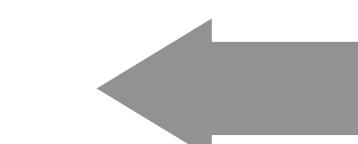
\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add nuevo2.txt
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git add nuevo2.txt
6 git commit -m "cambios"
```

\$ \_

```
1 echo "cont inicial" > nuevo2.txt
2 git add . ←
3 git commit -m "nuevo archivo 2"
4 echo "cambios" >> nuevo2
5 git commit -a -m "cambios"
```

Con el punto  
se añaden  
todos los  
ficheros  
nuevos o  
modificados



La opción **-a** permite realizar el comando **add** junto con el **commit**.  
Nota: sólo sirve para ficheros ya controlados



## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

# Ciclo de vida - Comprobar estado

## El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
1 echo "cont inicial" > nuevo3.txt
2 git status
3 [...]
4 Untracked files:
5   (use "git add <file>..." to include in what will be committed)
6     .
7       nuevo3.txt
8
9   git add nuevo3.txt
10
11  git status
12  [...]
13 Changes to be committed:
14   (use "git restore --staged <file>..." to unstage)
15     .
16       new file:   nuevo3.txt
17
18  git commit -m "nuevo archivo 3"
19
20  [main 6b95f2e] nuevo archivo 3
21    1 file changed, 1 insertion(+)
22    create mode 100644 nuevo3.txt
23
24  git status
25  [...]
26
27  nothing to commit, working tree clean
28
29  echo "cambios" >> nuevo3.txt
```



# Ciclo de vida - Comprobar estado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
1 echo "cont inicial" > nuevo3.txt
2 git status
3 [...]
4 Untracked files:
5     (use "git add <file>..." to include in what will be committed)
6         .
7             nuevo3.txt
8 git add nuevo3.txt
9 git status
10 [...]
11 Changes to be committed:
12     (use "git restore --staged <file>..." to unstage)
13         .
14             new file:   nuevo3.txt
15 git commit -m "nuevo archivo 3"
16 [main 6b95f2e] nuevo archivo 3
17     1 file changed, 1 insertion(+)
18     create mode 100644 nuevo3.txt
19 git status
20 [...]
21 nothing to commit, working tree clean
22 echo "cambios" >> nuevo3.txt
```



Sin seguimiento



# Ciclo de vida - Comprobar estado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

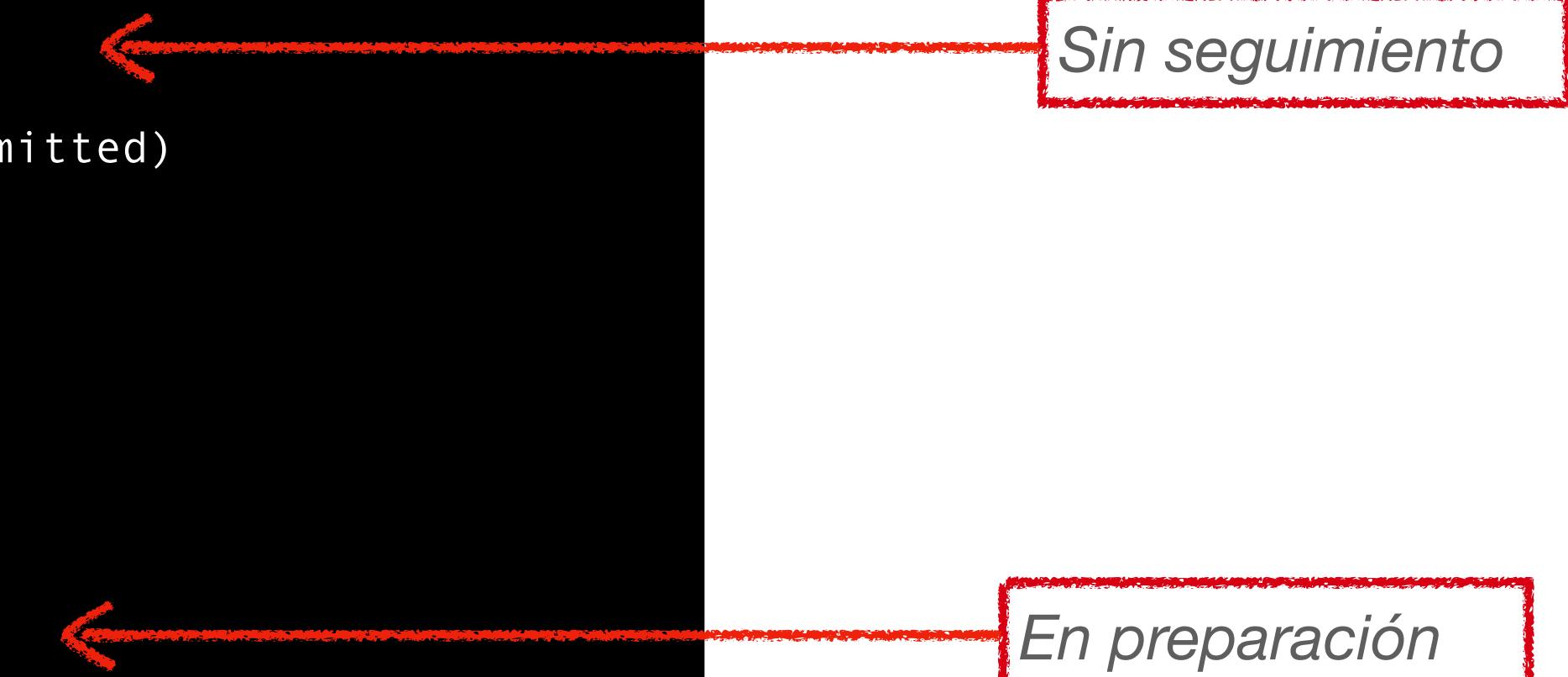
5. Ramas

6. Sincronización  
remota

El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
1 echo "cont inicial" > nuevo3.txt
2 git status
3 [...]
4 Untracked files:
5     (use "git add <file>..." to include in what will be committed)
6         .
7             nuevo3.txt
8 git add nuevo3.txt
9 git status
10 [...]
11 Changes to be committed:
12     (use "git restore --staged <file>..." to unstage)
13         .
14             new file:   nuevo3.txt
15 git commit -m "nuevo archivo 3"
16 [main 6b95f2e] nuevo archivo 3
17     1 file changed, 1 insertion(+)
18     create mode 100644 nuevo3.txt
19 git status
20 [...]
21 nothing to commit, working tree clean
22 echo "cambios" >> nuevo3.txt
```



Sin seguimiento

En preparación



# Ciclo de vida - Comprobar estado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota

\$ \_

```
1 echo "cont inicial" > nuevo3.txt
2 git status
3 [...]
4 Untracked files:
5   (use "git add <file>..." to include in what will be committed)
6     nuevo3.txt
7 git add nuevo3.txt
8 git status
9 [...]
10 Changes to be committed:
11   (use "git restore --staged <file>..." to unstage)
12     new file:   nuevo3.txt
13 git commit -m "nuevo archivo 3"
14 [main 6b95f2e] nuevo archivo 3
15   1 file changed, 1 insertion(+)
16   create mode 100644 nuevo3.txt
17 git status
18 [...]
19 nothing to commit, working tree clean
20 echo "cambios" >> nuevo3.txt
```

Sin seguimiento

En preparación

Sin cambios



## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
1 echo "cont inicial" > nuevo3.txt
2 git status
3 .
4 .
5 Untracked files:
6 .
7   (use "git add <file>..." to include in what will be committed)
8     .
9       nuevo3.txt
10  git add nuevo3.txt
11  git status
12  .
13  .
14  Changes to be committed:
15  .
16  (use "git restore --staged <file>..." to unstage)
17  .
18    new file:   nuevo3.txt
19  git commit -m "nuevo archivo 3"
20  [main 6b95f2e] nuevo archivo 3
21  .
22  1 file changed, 1 insertion(+)
23  .
24  create mode 100644 nuevo3.txt
25  git status
26  .
27  .
28  nothing to commit, working tree clean
29  echo "cambios" >> nuevo3.txt
```



## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
4 git status
.
.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    new file:   nuevo3.txt
5 git commit -m "nuevo archivo 3"
[main 6b95f2e] nuevo archivo 3
  1 file changed, 1 insertion(+)
  create mode 100644 nuevo3.txt
6 git status
...
nothing to commit, working tree clean
echo "cambios" >> nuevo3.txt
7 git status
...
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:   nuevo3.txt
.
```



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
. [main 6b95f2e] nuevo archivo 3
.   1 file changed, 1 insertion(+)
.   create mode 100644 nuevo3.txt
6   git status
.   [...]
.   nothing to commit, working tree clean
.   echo "cambios" >> nuevo3.txt
7   git status
.   [...]
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:   nuevo3.txt
.
8   git add nuevo3.txt
9   git status
.   [...]
Changes to be committed:
.   (use "git restore --staged <file>..." to unstage)
.       modified:   nuevo3.txt
.
```



# Ciclo de vida - Comprobar estado

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

El estado en el que se encuentran los ficheros de nuestro repositorio se comprueba con **git status**

\$ \_

```
. echo "cambios" >> nuevo3.txt
7 git status
.
[...]
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:   nuevo3.txt

8 git add nuevo3.txt
9 git status
[...]
Changes to be committed:
.
(use "git restore --staged <file>..." to unstage)
.
modified:   nuevo3.txt
10 git commit -m "cambios"
.
[main 21266c0] cambios
.
1 file changed, 1 insertion(+)
11 git status
[...]
nothing to commit, working tree clean
```



# Ciclo de vida - Ejercicio 3.4

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



Crea una copia local del repositorio <https://github.com/ualgit/latex.git> y realiza las siguientes tareas:

- A. Sin hacer nada con git, descarga del repositorio de material el fichero latex1.zip y descomprímelo en el repositorio del ejercicio.
- B. Crea 3 commits distintos con los cambios que se especifican a continuación.
  - Commit 1: añade la carpeta img y descomenta la línea que incluye la imagen en document.tex.
  - Commit 2: descomenta la referencia sec.tex en document.tex.
  - Commit 3: cambia alguna palabra del texto en document.tex. Con GitHub Desktop analiza los cambios y realiza el commit.

★ Si en cualquier momento quieres volver al estado inicial usa `git reset --hard 16eb8`



# Historial de commits - log

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

Para consultar el historial de commits, se utilizan los comandos:

\$ \_

git log

\$ \_

git log [SHA o etiqueta]

```
[rcabanas@rc-mac-ual repo_simple % git log
commit 21266c0eab84b2ebd9f1c2d840ca439616de856c (HEAD -> main)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:47:14 2023 +0100

    cambios

commit 6b95f2efef91d308da4f53b67131401d24af8ff7
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:41:11 2023 +0100

    nuevo archivo 3

commit 9123a69aa9db2412c420660d17de116fc56c2881 (origin/main, origin/HEAD)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:35:08 2023 +0100

    readme

commit bbe4460acb2280ccb48729f616dab70aa1ce6c4e
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 18:36:15 2023 +0100

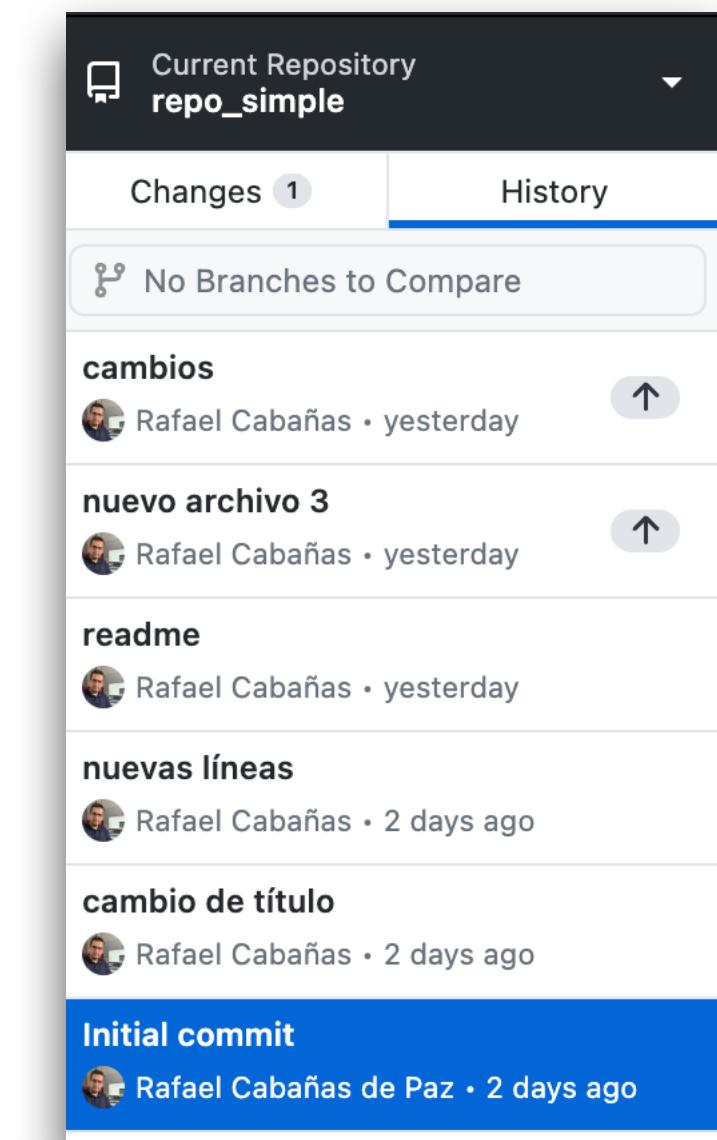
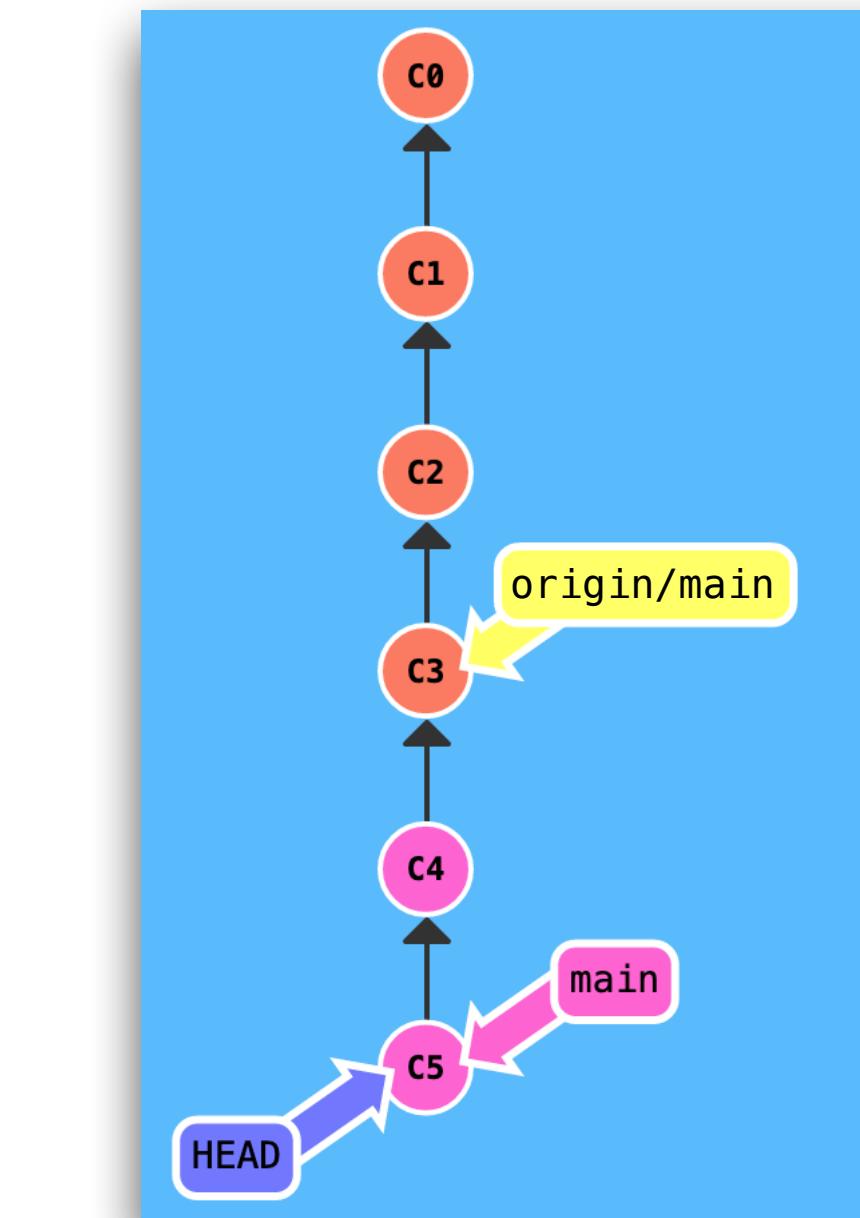
    nuevas líneas

commit 635cd8fc283d684f15b358d1249fcc18278ebf8f
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 10:01:36 2023 +0100

    cambio de título

commit 457456110fae65104c1cdb0f56041b5c2d5149e9
Author: Rafael Cabañas de Paz <rafacabanas@gmail.com>
Date:   Tue Mar 7 09:28:07 2023 +0100

    Initial commit
rcabanas@rc-mac-ual repo_simple % ]
```



Muestra el historial a partir de un commit específico.



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En un repositorio git es posible volver al pasado



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En un repositorio git es posible volver al pasado





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En un repositorio git es posible volver al pasado
- El siguiente comando permite "moverse" a cualquier commit

\$ \_

git checkout [SHA o etiqueta]



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En un repositorio git es posible volver al pasado
- El siguiente comando permite "moverse" a cualquier commit

\$ \_

git checkout [SHA o etiqueta]

- GitHub Desktop no permite esta acción
- Cambian los ficheros que vemos en nuestra copia local: **no se pierden datos**
- Al hacer checkout es recomendable que no haya cambios sin meter en un commit.

No obstante, hay que tener en cuenta:

- Los ficheros **sin seguimiento** no se ven afectados por el cambio.
- No se puede hacer checkout si hay ficheros en la zona de **preparado** o **modificado**



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

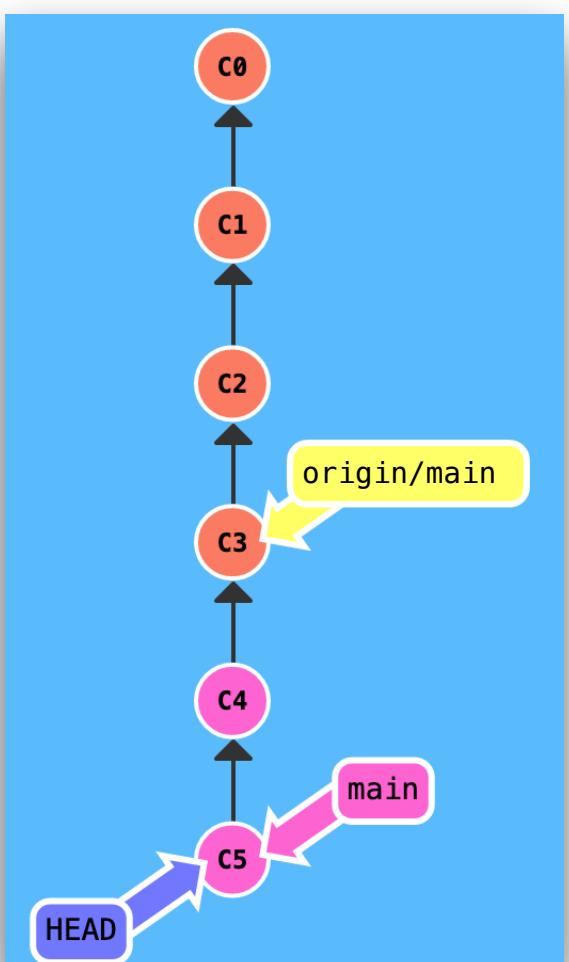
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## Ejemplo de uso:





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

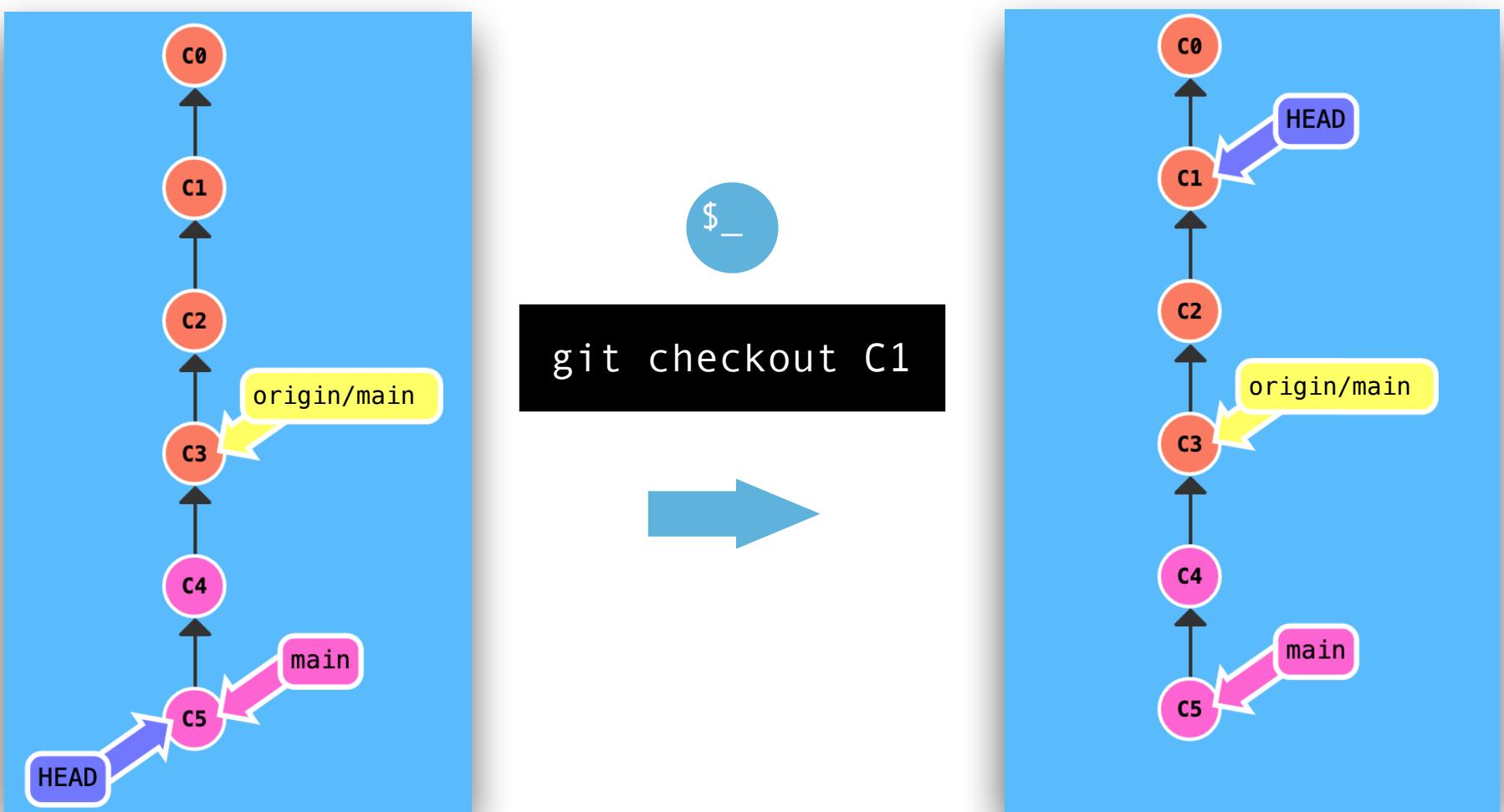
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## Ejemplo de uso:





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

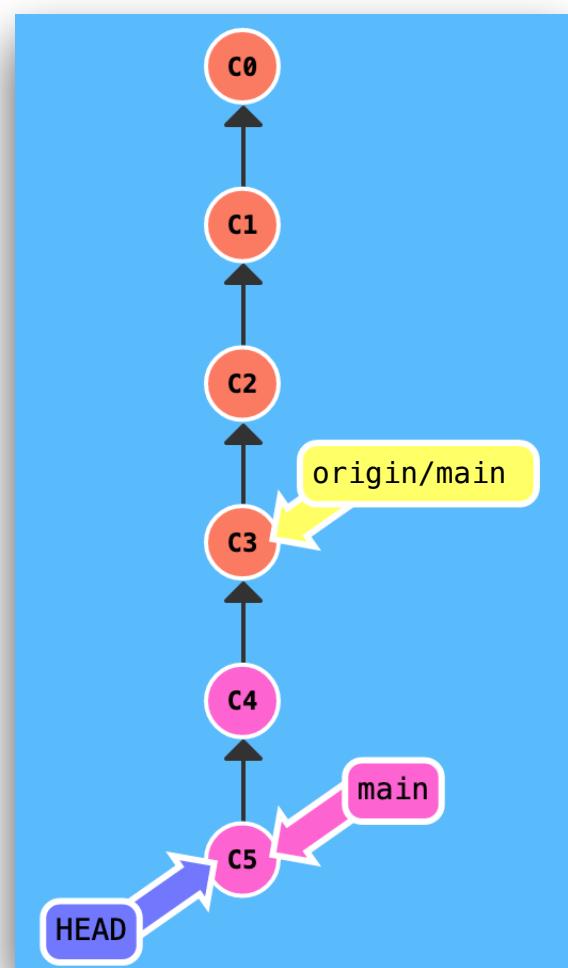
3. Control de  
cambios en local

4. Correcciones

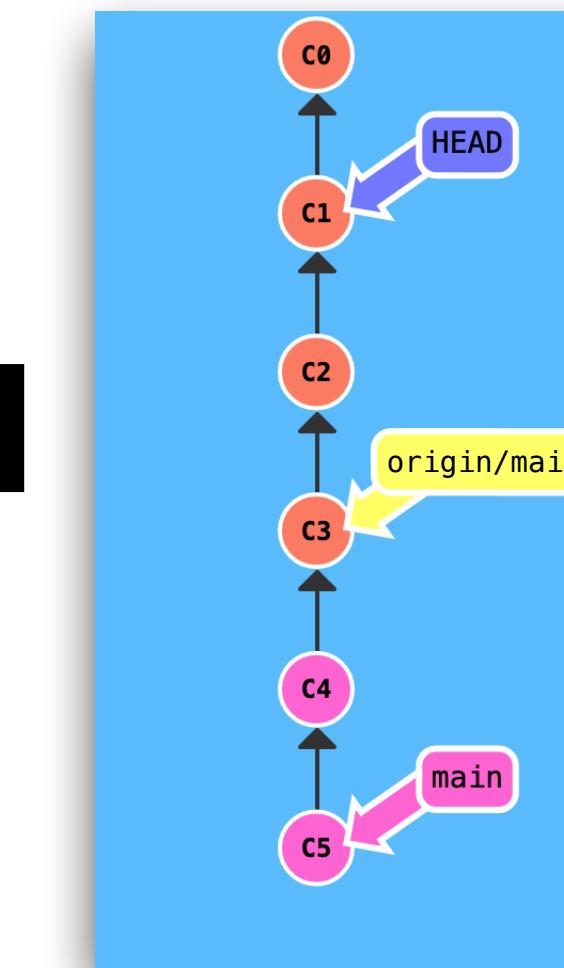
5. Ramas

6. Sincronización  
remota

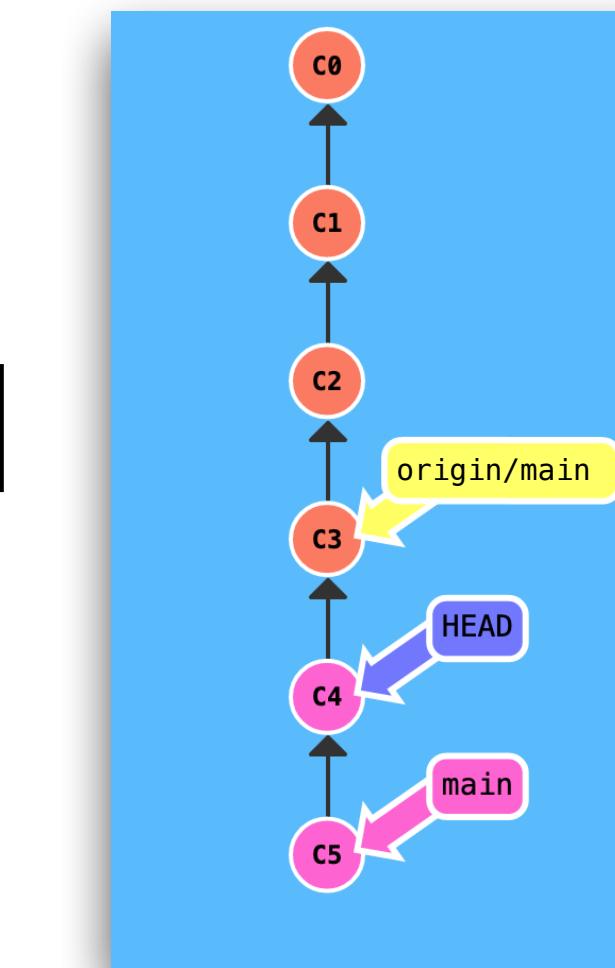
## Ejemplo de uso:



git checkout C1



git checkout C4





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

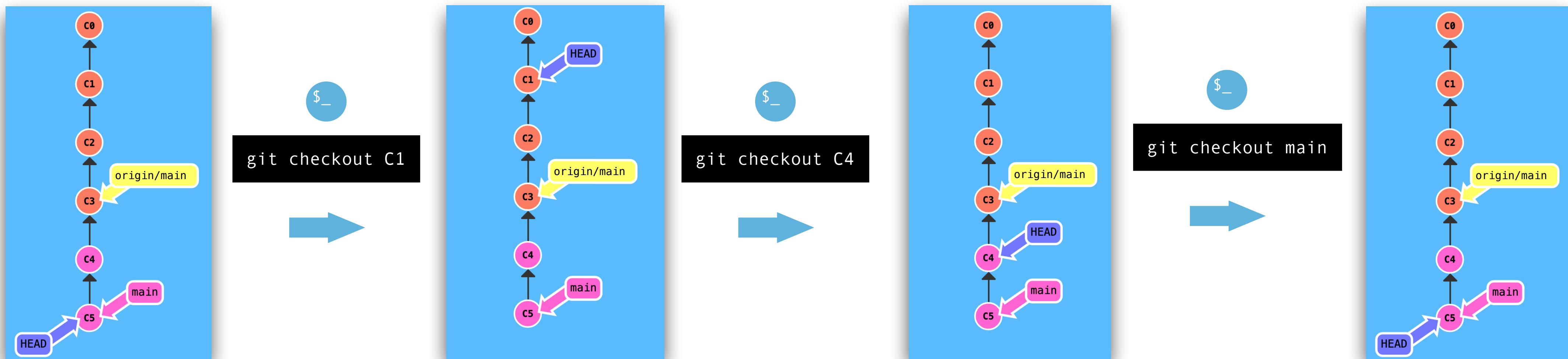
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

## Ejemplo de uso:





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

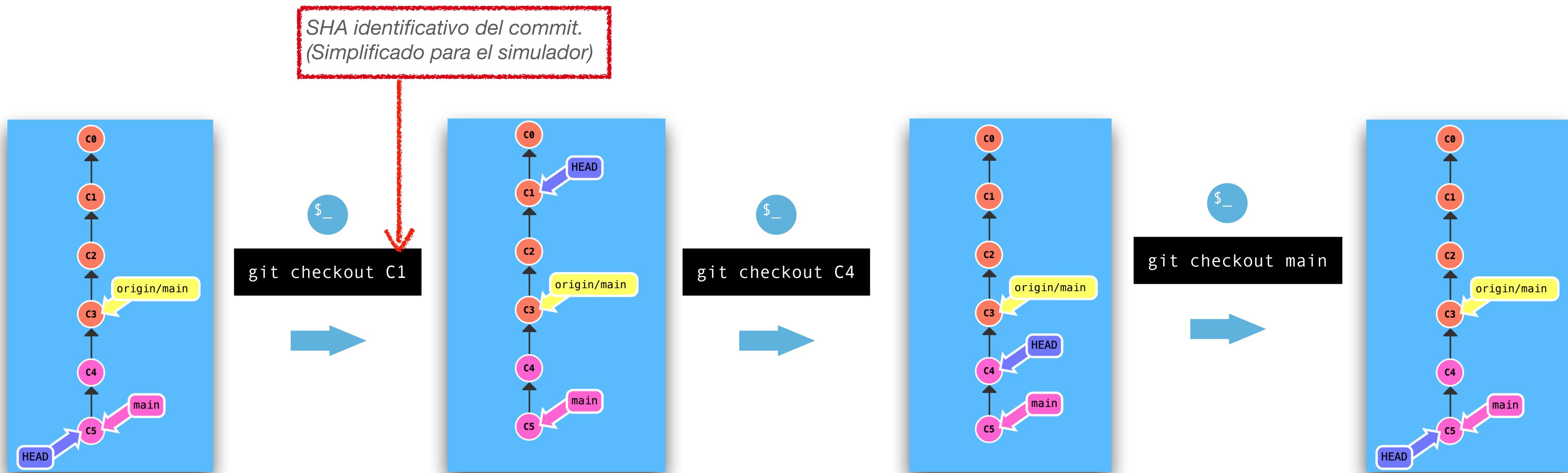
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

## Ejemplo de uso:





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

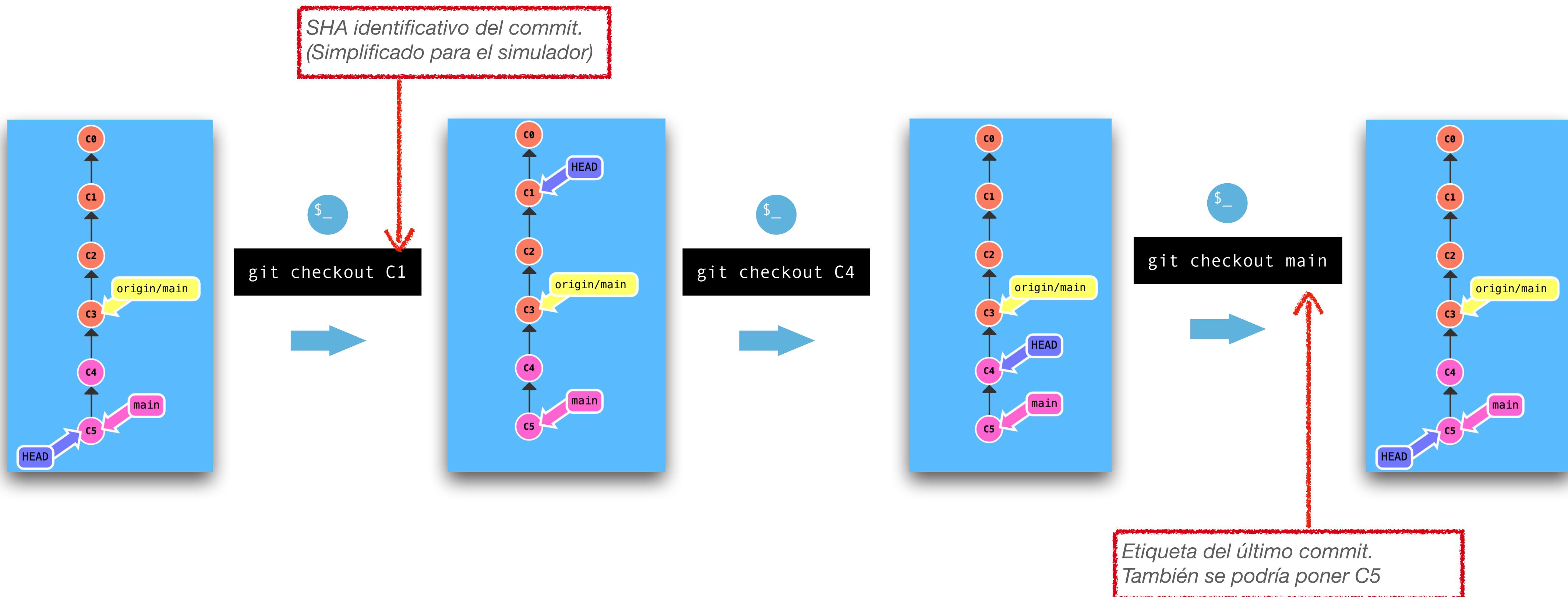
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

## Ejemplo de uso:





# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

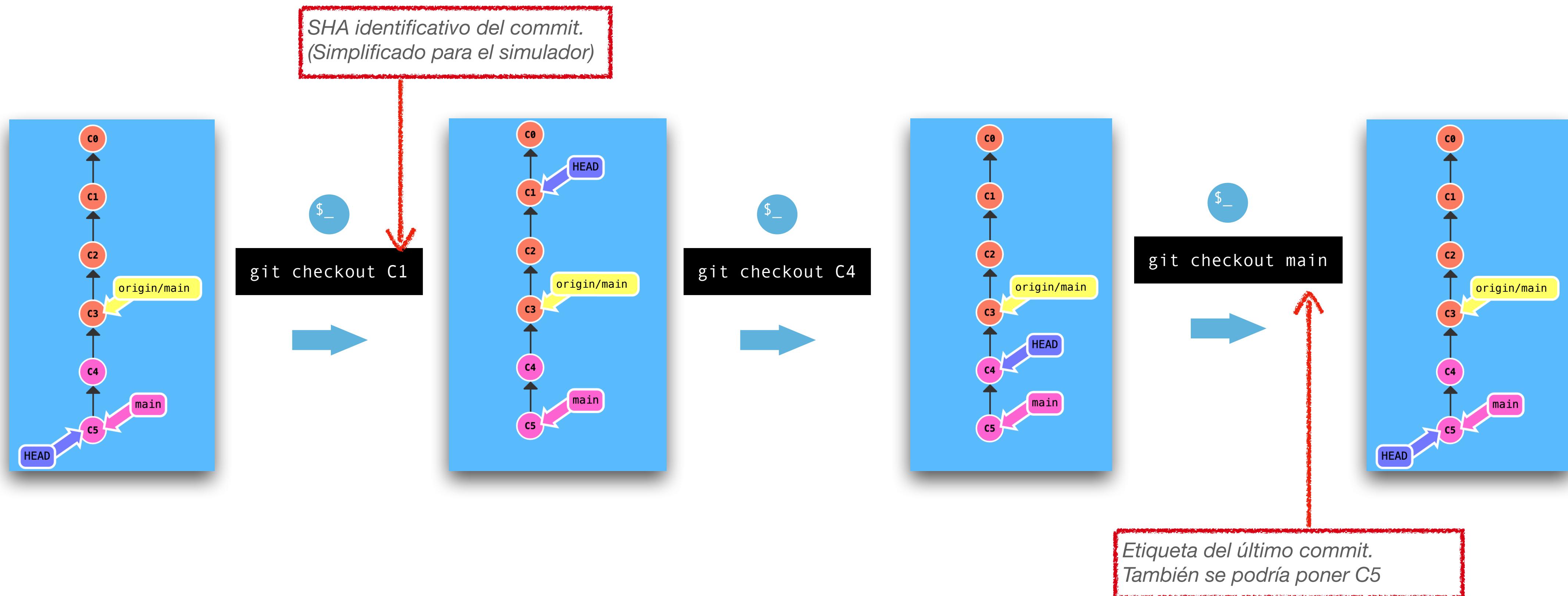
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

## Ejemplo de uso:



Prueba los comandos anteriores en el [simulador](#)



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- En un repositorio real (fuera del simulador) los commits se identifican mediante un número hexadecimal (SHA).
- Se puede indicar el commit de destino de distintas formas

```
[rcabanas@rc-mac-ual repo_simple % git log
commit 21266c0eab84b2ebd9f1c2d840ca439616de856c (HEAD -> main)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:47:14 2023 +0100

    cambios

commit 6b95f2efef91d308da4f53b67131401d24af8ff7
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:41:11 2023 +0100

    nuevo archivo 3

commit 9123a69aa9db2412c420660d17de116fc56c2881 (origin/main, origin/HEAD)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:35:08 2023 +0100

    readme

commit bbe4460acb2280cbb48729f616dab70a1ce6c4e
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 18:36:15 2023 +0100

    nuevas líneas

commit 635cd8fc283d684f15b358d1249fcc18278ebf8f
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 10:01:36 2023 +0100

    cambio de título

commit 457456110fae65104c1cdb0f56041b5c2d5149e9
Author: Rafael Cabañas de Paz <rafacabanas@gmail.com>
Date:   Tue Mar 7 09:28:07 2023 +0100

    Initial commit
rcabanas@rc-mac-ual repo_simple % ]
```

\$ \_

git checkout 6b95f2efef91d308da4f53b67131401d24af8ff7

\$ \_

git checkout 6b95f2efef91d308da4f5

\$ \_

git checkout 6b95

\$ \_

git checkout main



# Historial de commits - checkout

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- En un repositorio real (fuera del simulador) los commits se identifican mediante un número hexadecimal (SHA).
- Se puede indicar el commit de destino de distintas formas

```
[rcabanas@rc-mac-ual repo_simple % git log
commit 21266c0eab84b2ebd9f1c2d840ca439616de856c (HEAD -> main)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:47:14 2023 +0100

    cambios

commit 6b95f2efef91d308da4f53b67131401d24af8ff7
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:41:11 2023 +0100

    nuevo archivo 3

commit 9123a69aa9db2412c420660d17de116fc56c2881 (origin/main, origin/HEAD)
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Wed Mar 8 09:35:08 2023 +0100

    readme

commit bbe4460acb2280cbb48729f616dab70a1ce6c4e
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 18:36:15 2023 +0100

    nuevas líneas

commit 635cd8fc283d684f15b358d1249fcc18278ebf8f
Author: Rafael Cabañas <rcabanas@ual.es>
Date:   Tue Mar 7 10:01:36 2023 +0100

    cambio de título

commit 457456110fae65104c1cdb0f56041b5c2d5149e9
Author: Rafael Cabañas de Paz <rafacabanas@gmail.com>
Date:   Tue Mar 7 09:28:07 2023 +0100

    Initial commit
rcabanas@rc-mac-ual repo_simple % ]
```

\$ \_ git checkout 6b95f2efef91d308da4f53b67131401d24af8ff7

\$ \_ git checkout 6b95f2efef91d308da4f5

\$ \_ git checkout 6b95 ← mínimo 4 dígitos

\$ \_ git checkout main



# Historial de commits - etiquetas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En un repositorio git nos podemos situar en cualquier commit y no siempre estaremos el más reciente.
- Para saber en cual estamos localmente, git asigna la etiqueta **HEAD**
- Los siguientes **git show** y **git rev-parse** permiten obtener información de cualquier commit, y también del HEAD:

\$ \_

git show HEAD

\$ \_

git rev-parse HEAD

- La gran utilidad de esta etiqueta está en poder referenciar cualquier commit de forma relativa a HEAD, indicando la distancia:

\$ \_

git checkout HEAD^

\$ \_

git checkout HEAD~1

\$ \_

git checkout HEAD^^

\$ \_

git checkout HEAD~2

\$ \_

git checkout HEAD^^^

\$ \_

git checkout HEAD~3



# Grafo de commits - etiquetas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

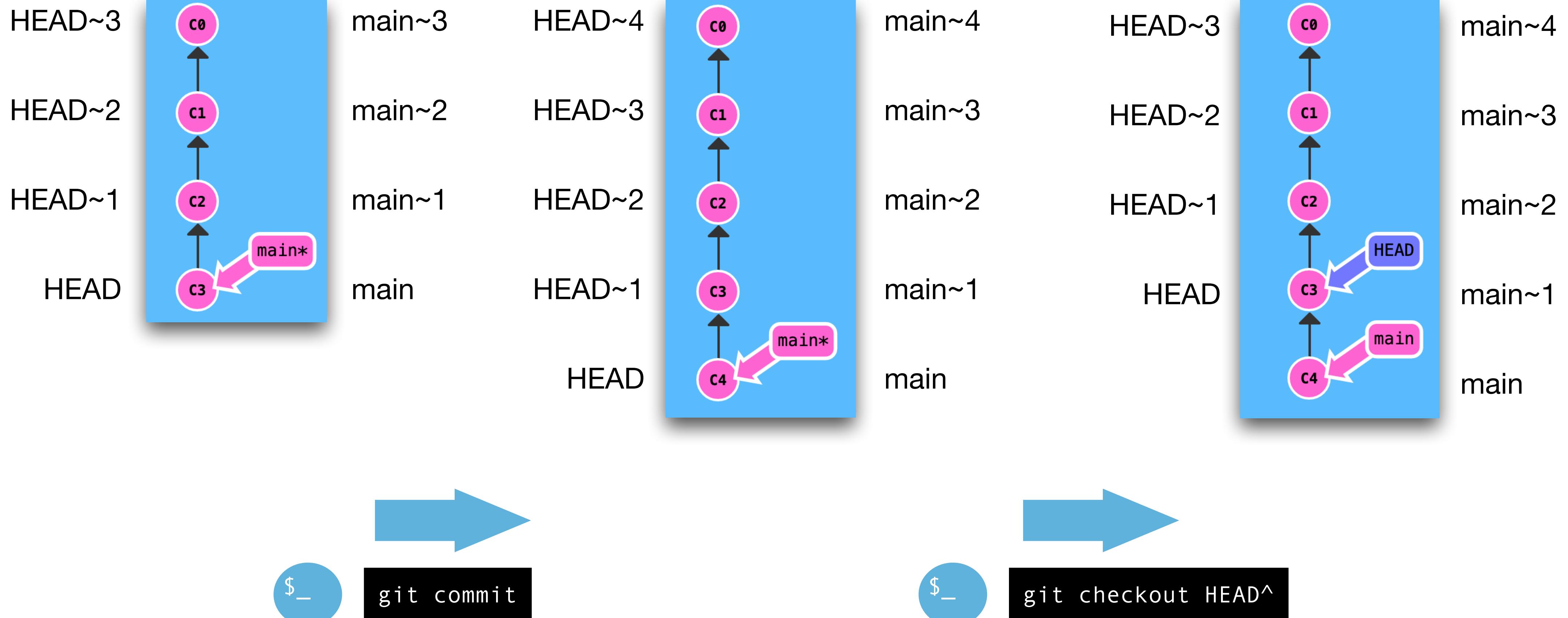
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Las referencias relativas se pueden utilizar con cualquier etiqueta.





# Grafo de commits - etiquetas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Se pueden definir etiquetas "custom" a cualquier commit

\$ \_

```
git tag version1
```

\$ \_

```
git tag v1
```

\$ \_

```
git tag enviado
```

- El siguiente comando muestra el listado de todas las etiquetas definidas

\$ \_

```
git tag
```

- Es posible utilizar los mismos comandos que con las etiquetas predefinidas

\$ \_

```
git checkout v1
```

\$ \_

```
git checkout v1~2
```

\$ \_

```
git show v1
```



# Grafo de commits - etiquetas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

```
Learn Git Branching
```

\$ git tag v1  
\$ git commit  
\$ git checkout v1~1  
\$ git tag

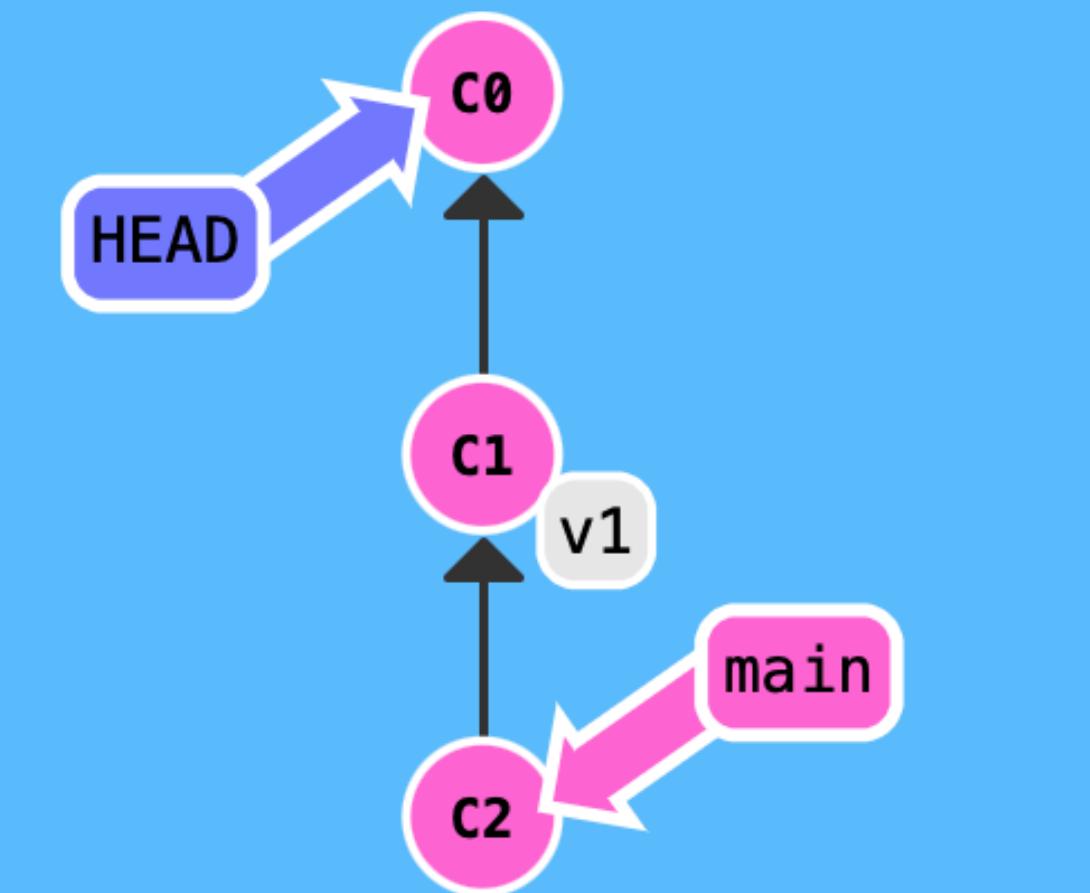
v1

\$ git show v1

Author: Peter Cottle  
Date: Fri Mar 10 2023 20:10:48  
GMT+0100 (CET)

Quick commit. Go Bears!

Commit: C1  
diff --git a/bigGameResults.html  
b/bigGameResults.html  
--- bigGameResults.html  
+++ bigGameResults.html  
@@ 13,27 @@ Winner, Score  
- Stanfurd, 14-7  
+ Cal, 21-14





# Grafo de commits - etiquetas

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

Learn Git Branching

```
$ git tag v1
$ git commit
$ git checkout v1~1
$ git tag
```

v1

```
$ git show v1
```

Author: Peter Cottle  
Date: Fri Mar 10 2023 20:10:48  
GMT+0100 (CET)

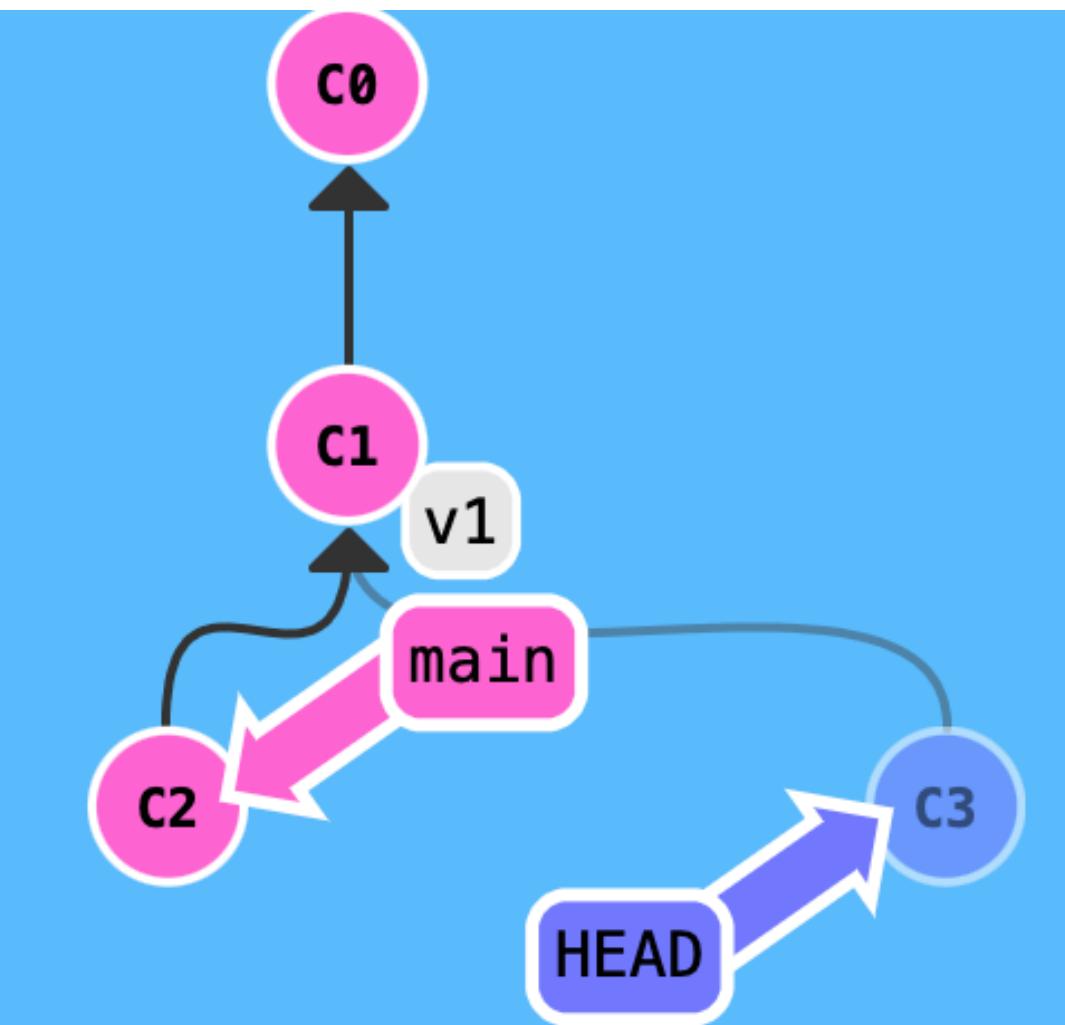
Quick commit. Go Bears!

Commit: C1

```
diff --git a/bigGameResults.html
b/bigGameResults.html
--- bigGameResults.html
+++ bigGameResults.html
@@ 13,27 @@ Winner, Score
- Stanfurdf, 14-7
+ Cal, 21-14
```

```
$ git checkout v1
$ git commit
```

! Warning!! Detached HEAD state





# Ciclo de vida - Ejercicio 3.5

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

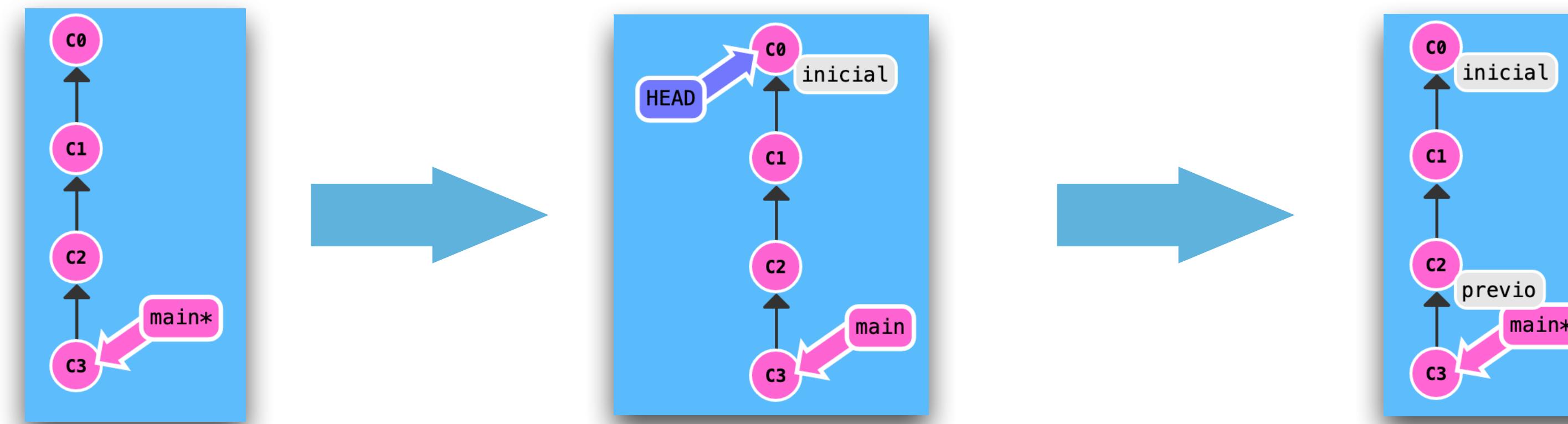
4. Correcciones

5. Ramas

6. Sincronización  
remota



Reproduce el siguiente árbol de commits en el simulador utilizando referencias relativas a las etiquetas, es decir, no se pueden utilizar comandos del tipo *git checkout C0*.





# Ciclo de vida - Ejercicio 3.6

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



En el mismo repositorio con los fuentes de latex, realiza las siguientes tareas:

- A. Crea un archivo nuevo (da igual el contenido) sin añadirlo
- B. Modifica el README.md si añadir los cambios a git.
- C. Prueba a hacer checkout (no será posible)
- D. Añade sólo **los cambios del readme** a un commit nuevo
- E. Haz checkout a los commits anteriores, observa el contenido y el log
- F. Añade etiquetas a los commits en los que se añadió la figura y la sección.
- G. Vuelve al HEAD



# Sin seguimiento - eliminar

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Para eliminar un fichero de un repositorio se utiliza el comando **git rm**.

\$ \_

```
git rm [fichero]
```

\$ \_

```
git rm --cached [fichero]
```

- La opción **--cached** evita que se elimine también del disco duro (simplemente pasa a untracked)

\$ \_

```
1 git rm README.md
2 git status
· On branch main
· [...]
Changes to be committed:
(use "git re..." to unstage)
· deleted: README.md
```

\$ \_

```
1 git rm --cached README.md
2 git status
· On branch main
· [...]
Untracked files:
(use "git add ... committed")
· README.md
```



En el repositorio que has creado, crea 2 ficheros de texto plano nuevo, añadirlos a un commit y probar el comando git rm con y sin la opción **--cached**.



# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El archivo **.gitignore** (con punto al inicio) define una serie de reglas de ficheros a ignorar
- Este archivo se debe guardar en el raíz del repositorio
- Las reglas consisten en expresiones regulares, de tal manera que si un fichero cumple alguna de estas expresiones, no se podrá añadir al repositorio (sí podrá estar en el disco duro local).



# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El archivo **.gitignore** (con punto al inicio) define una serie de reglas de ficheros a ignorar
- Este archivo se debe guardar en el raíz del repositorio
- Las reglas consisten en expresiones regulares, de tal manera que si un fichero cumple alguna de estas expresiones, no se podrá añadir al repositorio (sí podrá estar en el disco duro local).
- Las expresiones regulares se definen de la siguiente manera.



# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El archivo **.gitignore** (con punto al inicio) define una serie de reglas de ficheros a ignorar
- Este archivo se debe guardar en el raíz del repositorio
- Las reglas consisten en expresiones regulares, de tal manera que si un fichero cumple alguna de estas expresiones, no se podrá añadir al repositorio (sí podrá estar en el disco duro local).
- Las expresiones regulares se definen de la siguiente manera.
  - Las líneas que comienzan por # se ignoran
  - Si el patrón comienza por con una barra "/", el patrón se comprueba sólo en el raíz. En caso contrario se comprueba de forma recursiva
  - Un asterisco \* representa cualquier secuencia de caracteres excepto "/".



# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El archivo **.gitignore** (con punto al inicio) define una serie de reglas de ficheros a ignorar
- Este archivo se debe guardar en el raíz del repositorio
- Las reglas consisten en expresiones regulares, de tal manera que si un fichero cumple alguna de estas expresiones, no se podrá añadir al repositorio (sí podrá estar en el disco duro local).
- Las expresiones regulares se definen de la siguiente manera.
  - Las líneas que comienzan por # se ignoran
  - Si el patrón comienza por con una barra "/", el patrón se comprueba sólo en el raíz. En caso contrario se comprueba de forma recursiva
  - Un asterisco \* representa cualquier secuencia de caracteres excepto "/".
  - Dos asteriscos \*\* pueden representar varias cosas:
    - \*\*/b representa cualquier fichero "b" dentro de cualquier subdirectorio.
    - a/\*\* representa cualquier secuencia de ficheros dentro de un directorio "a"
    - a/\*\*/b representa cualquier fichero "b" dentro de "a" y con otros subdirectorios



# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

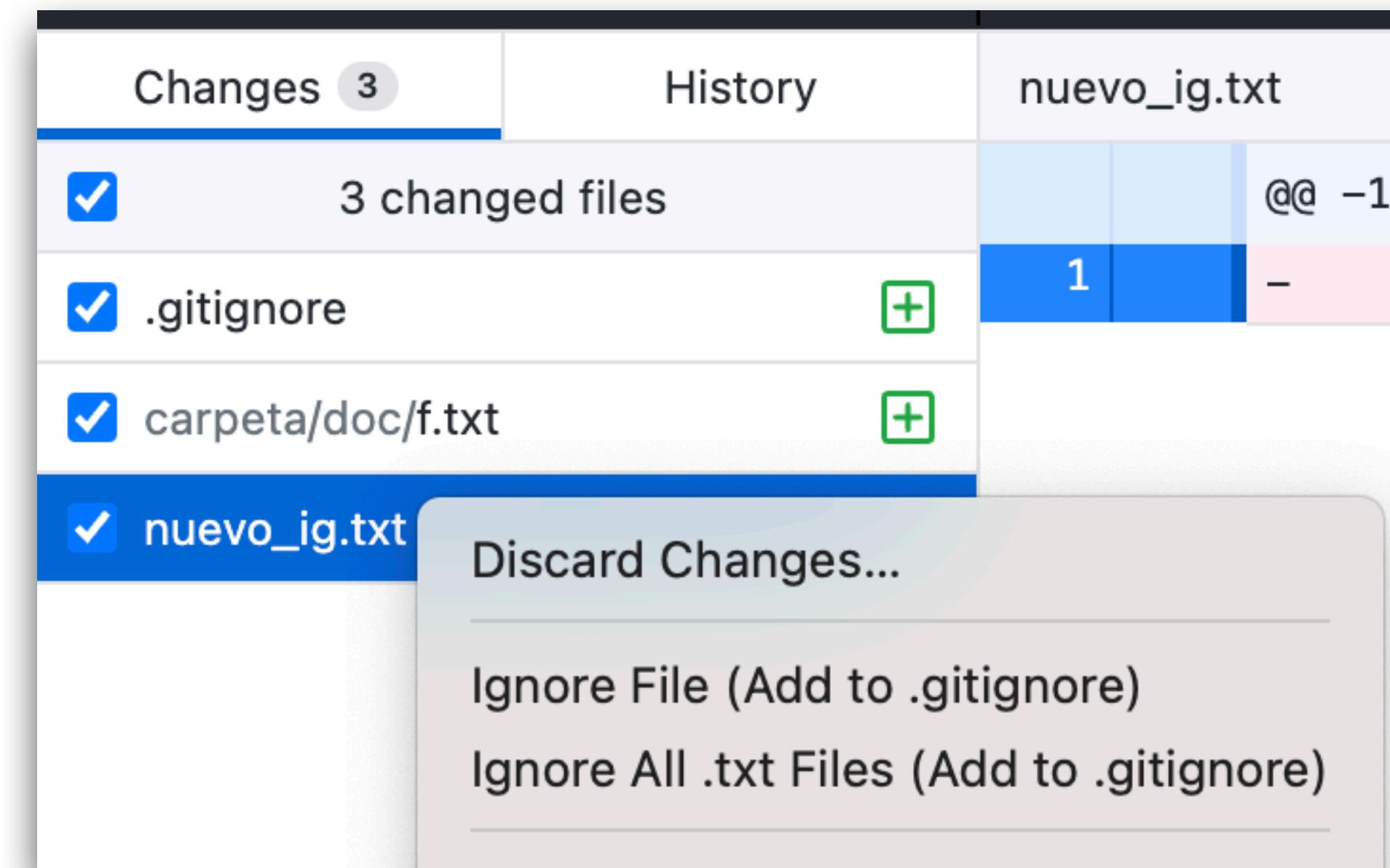
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Se pueden añadir reglas desde GitHub Desktop





# Sin seguimiento - gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## ○ Ejemplo de fichero .gitignore

```
# ignora cualquier archivo con extensión .a
*.a
# ignora el fichero TODO en el directorio actual (no afecta a los subdirectorios)
/TODO
# ignora cualquier directorio (recursivamente) llamado build
build/
# ignora de forma recursiva cualquier archivo .txt dentro de directorio doc
doc/*.txt
#ignora todos los .pdf que estén en un directorio doc/ o en cualquiera de los subdirectorios
doc/**/*.pdf
```

## ○ Es posible comprobar qué ficheros están siendo ignorados (y debido a qué regla)

\$ \_

```
git check-ignore -v **/*
```

## ○ También puede ser útil listar los ficheros que están en seguimiento:

\$ \_

```
git ls-files
```

\$ \_

```
git ls-files | grep fichero.txt
```



# Sin seguimiento - .gitignore

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

## ○ Ejemplo de fichero .gitignore

```
# ignora cualquier archivo con extensión .a
*.a
# ignora el fichero TODO en el directorio actual (no afecta a los subdirectorios)
/TODO
# ignora cualquier directorio (recursivamente) llamado build
build/
# ignora de forma recursiva cualquier archivo .txt dentro de directorio doc
doc/*.txt
#ignora todos los .pdf que estén en un directorio doc/ o en cualquiera de los subdirectorios
doc/**/*.pdf
```

## ○ Es posible comprobar qué ficheros están siendo ignorados (y debido a qué regla)

\$ \_

```
git check-ignore -v **/*
```

## ○ También puede ser útil listar los ficheros que están en seguimiento:

\$ \_

```
git ls-files
```

\$ \_

```
git ls-files | grep fichero.txt
```



En el repositorio que has creado, usa el .gitignore anterior y añade distintos ficheros que vayan a ser ignorados. Comprueba que esto ocurre.



# Sin seguimiento - Problemas frecuentes

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- PROBLEMA: las reglas en `.gitignore` **no afectan a los ficheros que ya están en seguimiento.**
- Prueba los siguientes comandos y analiza qué ocurre:

\$ \_

```
1 echo "" > nuevo_ig.txt
2 git add nuevo_ig.txt
3 git commit -m "nuevo_ig"
4 echo "*_ig*" >> .gitignore
5 git check-ignore -v **/*
```

- SOLUCIÓN: dejar de seguir todos los ficheros y volver a añadirlos:

\$ \_

```
1 git rm -r --cached .
2 git add .
```

\$ \_

```
1 git rm --cached nuevo_ig.txt
```



# Sin seguimiento - Ejercicio 3.7

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



En el mismo repositorio con los fuentes de latex, realiza las siguientes tareas:

- A. Busca un fichero .gitignore para latex <https://github.com/github/gitignore> y verifica qué expresiones regulares eliminarían los ficheros auxiliares de latex (aux, log, gz)
- B. Añádelo al repositorio y comprueba que funciona



# Problemas frecuentes - commits incompletos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

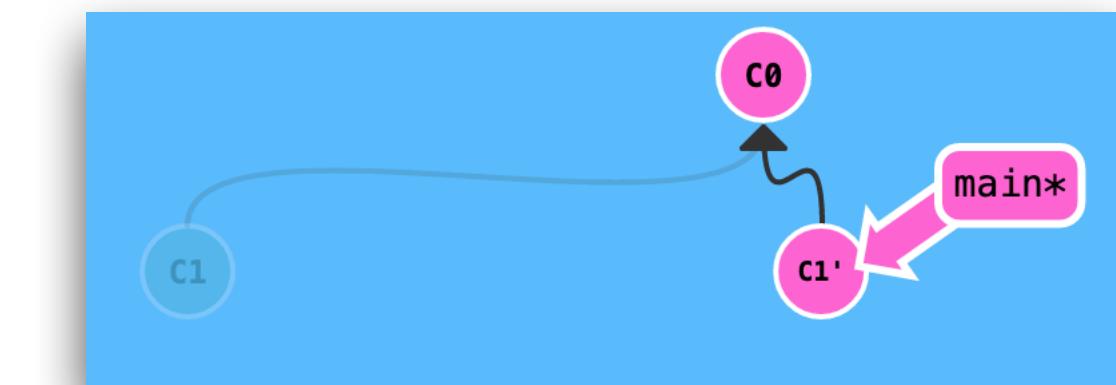
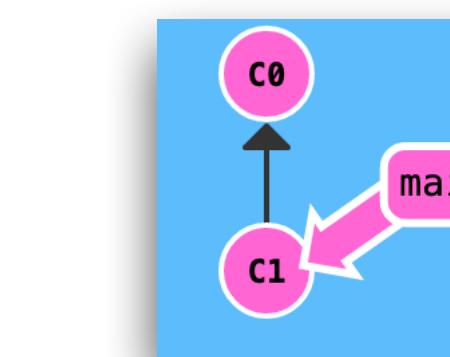
5. Ramas

6. Sincronización remota

- **PROBLEMA:** ya hemos hecho un commit y detectamos que nos faltaban archivos por añadir
- Es posible rehacer o reemplazar el último commit con la opción --amend

\$ \_

```
git commit --amend
```



- Esta opción elimina el último commit y crea uno nuevo con contenido adicional.

\$ \_

```
1 echo "cambios" >> README.md
2 git add .
3 git commit -m "camvios README"
4 git commit -m "cambios README"
```

\$ \_

```
1 echo "cambios" >> README.md
2 mkdir carpeta
3 cd carpeta
4 echo "cambios" >> nuevo.txt
5 git add .
6 git commit -m "readme y carpeta"
7 git ..
8 git commit --ammend -m "readme y carpeta"
```



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## 4. Correcciones



# Deshacer cambios en ficheros

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Para deshacer los cambios en un fichero se utiliza el comando **git restore**

\$ \_

```
git restore [fichero]
```

\$ \_

```
git restore --staged [fichero]
```

- Permiten quitar un fichero de la zona de modificado o de preparado



- No se pueden deshacer



# Revertir commits

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

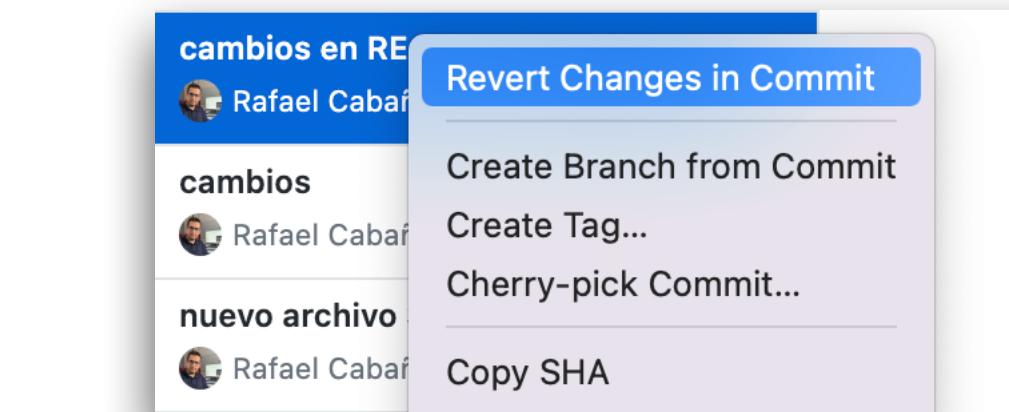
5. Ramas

6. Sincronización  
remota

- Los cambios confirmados se pueden deshacer con git revert:



```
git revert [SHA o etiqueta]
```



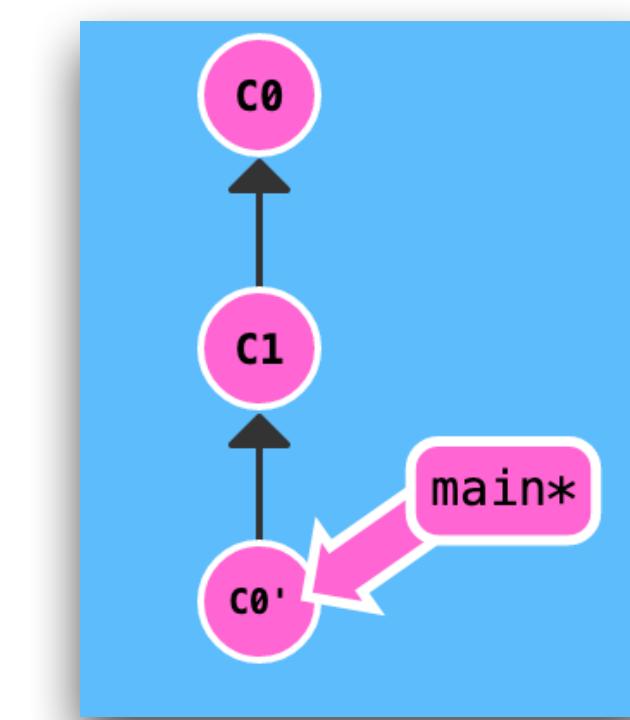
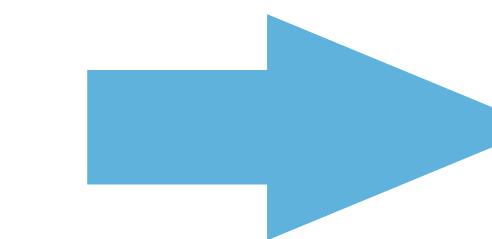
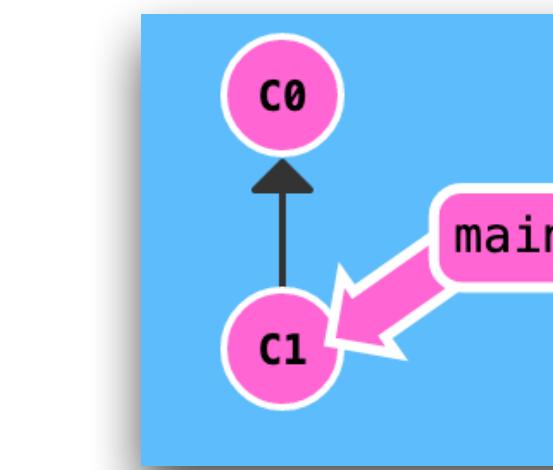
- En la práctica hay que indicar a qué commit se quiere volver



```
git revert HEAD^
```



```
git revert HEAD~1
```



- No se eliminan commits del grafo



En el repositorio que has creado, añade 3 commits con cambios y luego reviertelos al primer commit.



# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

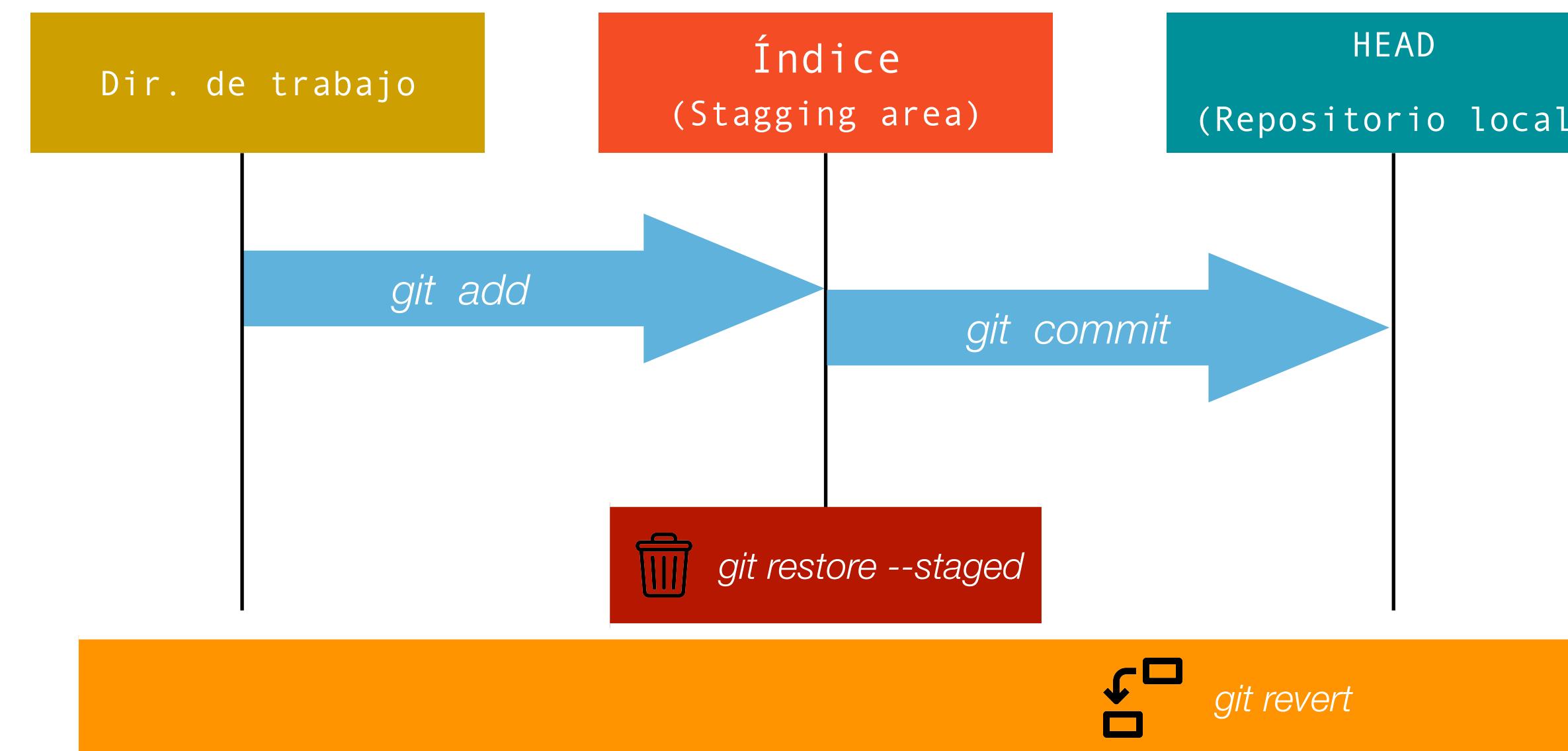
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Los estados de un fichero en git son: *sin seguimiento*, *sin cambios*, *con cambios* y *preparado*.
- Git trabaja internamente con 3 copias locales del repositorio:
  - **Directorio de trabajo:** con los cambios tal y como están en nuestro disco duro
  - **Índice:** con los cambios en preparado (staged) tras usar `git add`.
  - **HEAD:** con los cambios confirmados en el último commit.





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

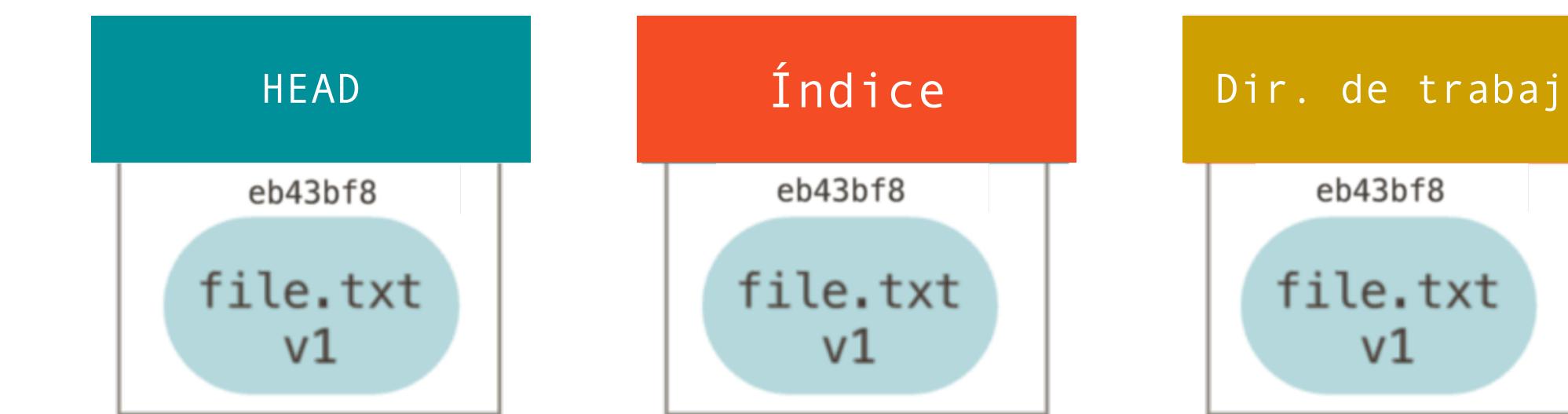
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Supongamos el siguiente escenario





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

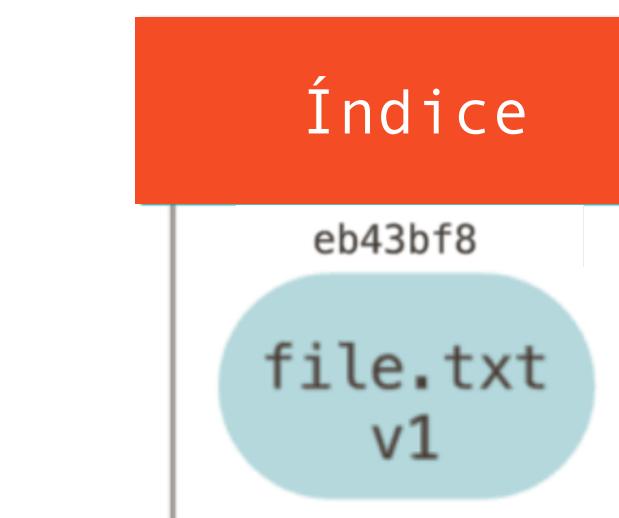
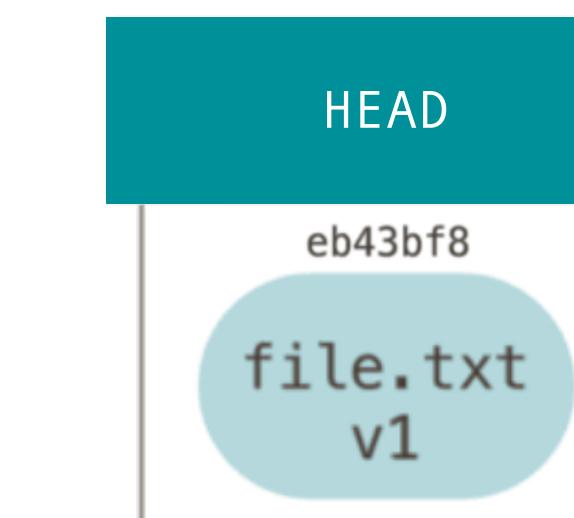
5. Ramas

6. Sincronización remota

## Realizamos cambios en el fichero

\$ \_

```
echo "cambios" >> file.txt
```





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

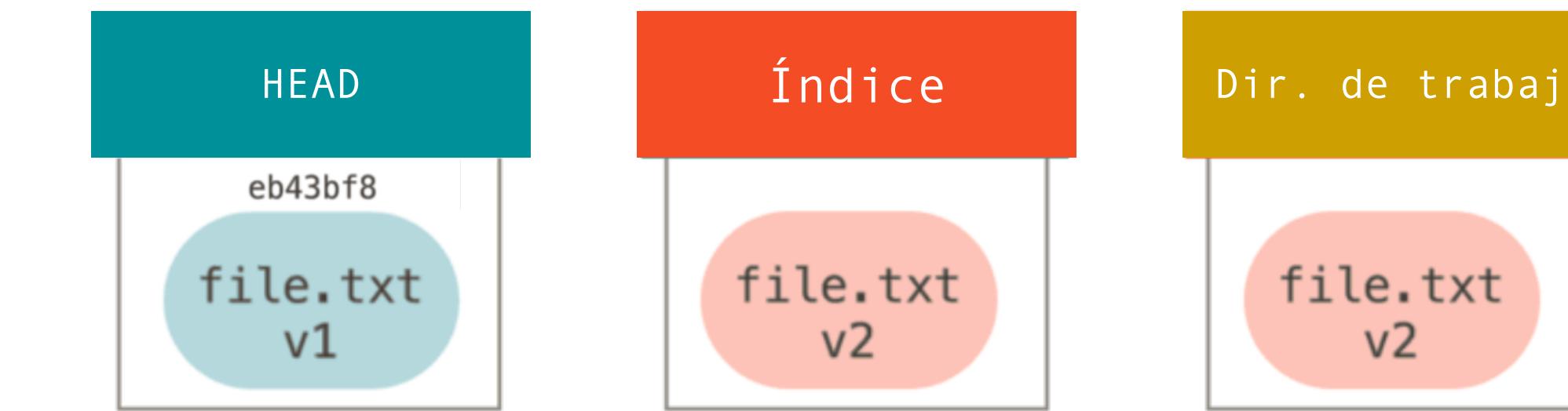
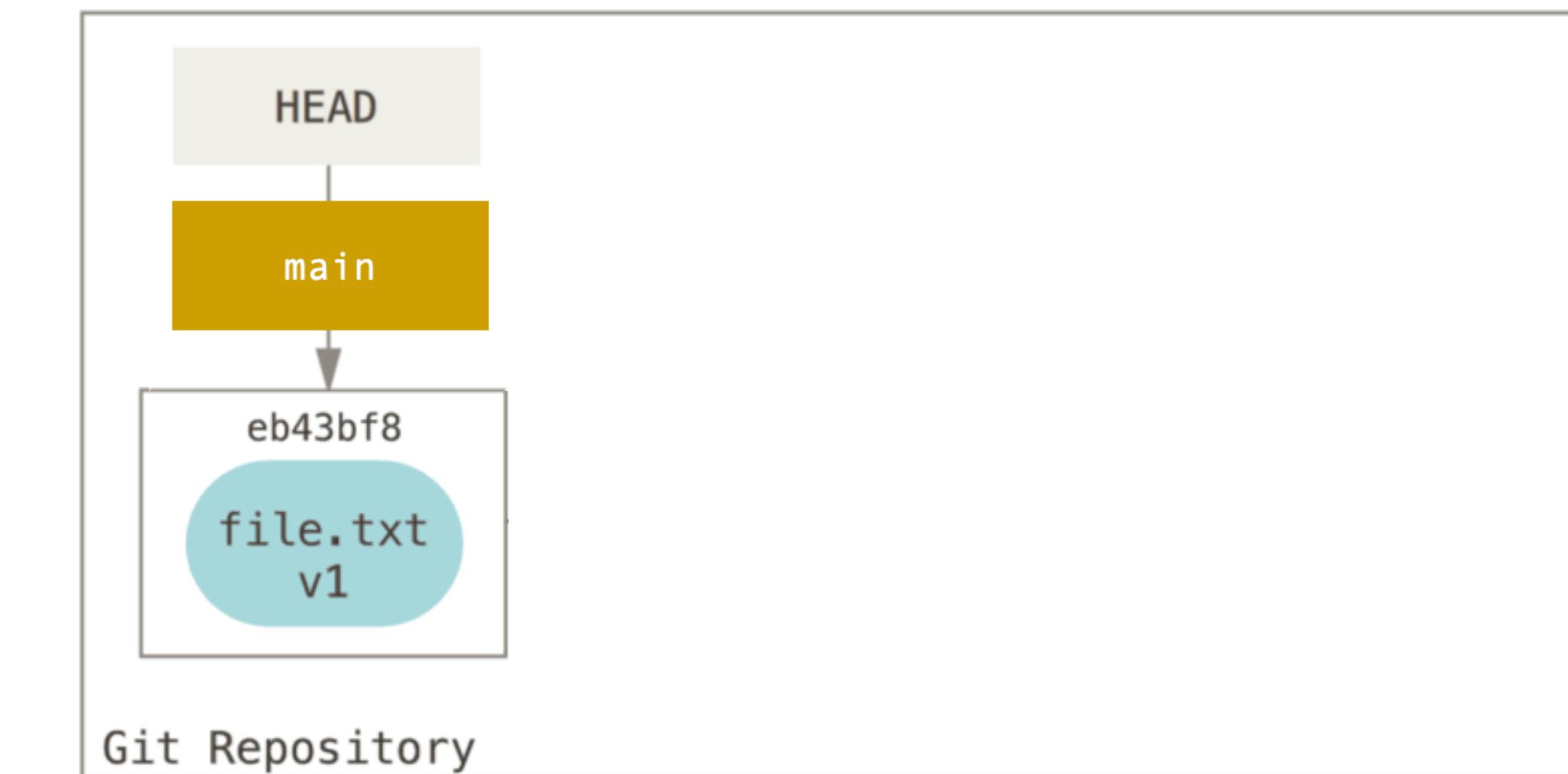
5. Ramas

6. Sincronización  
remota

- Añadimos el fichero a la zona de preparado (staged)

\$ \_

```
git add file.txt
```





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

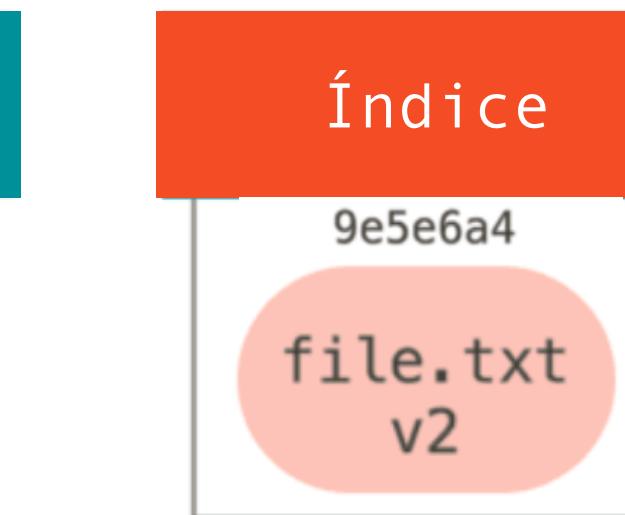
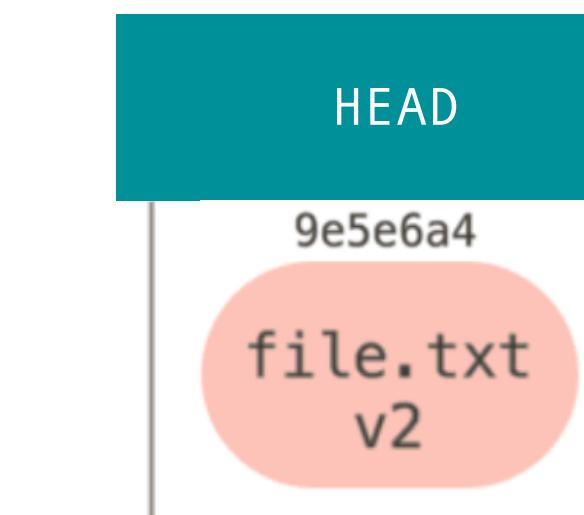
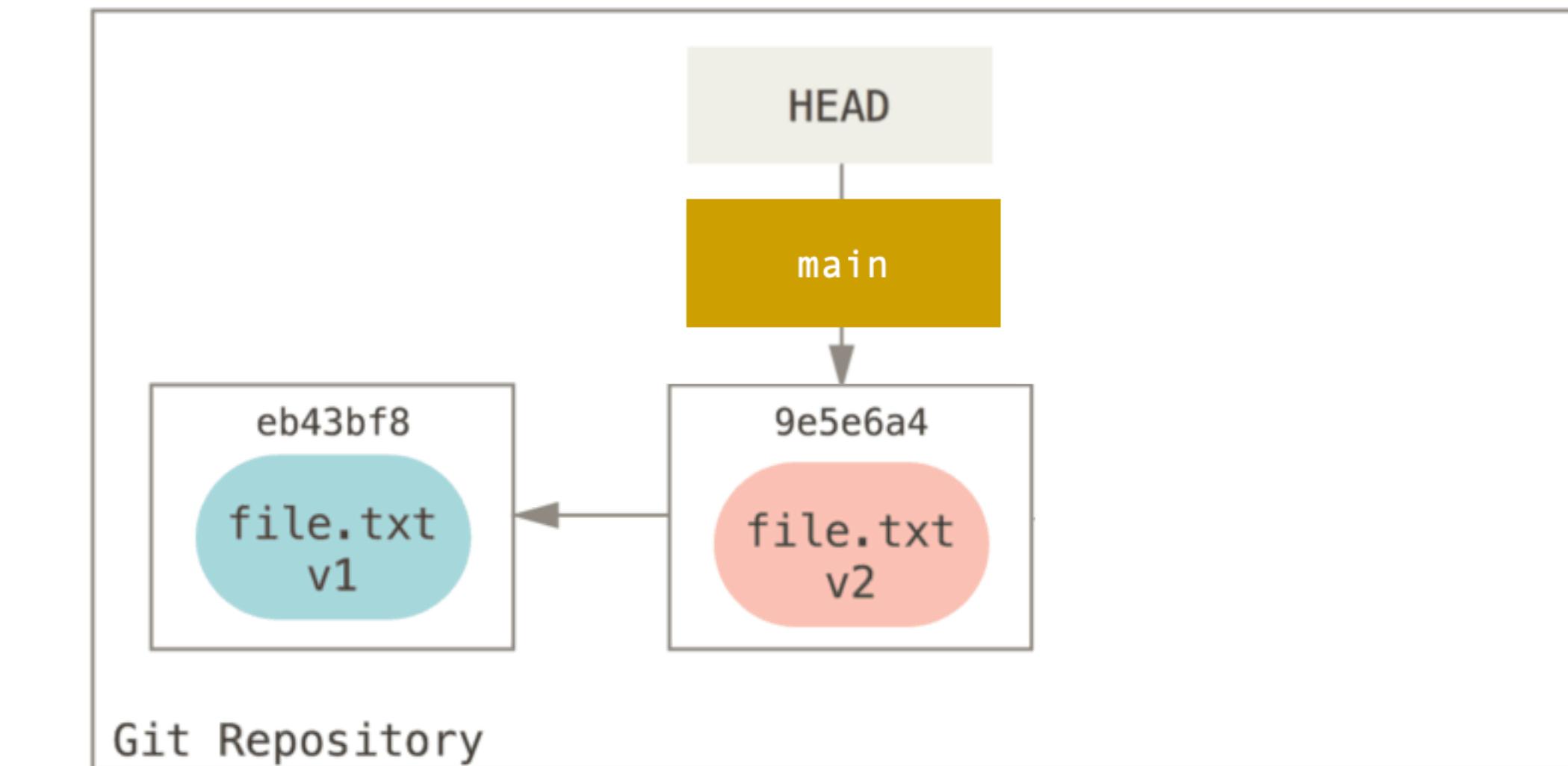
5. Ramas

6. Sincronización remota

- Confirmamos los cambios haciendo un commit

\$ \_

```
git commit -m "v2 file.txt"
```





# Flujo trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

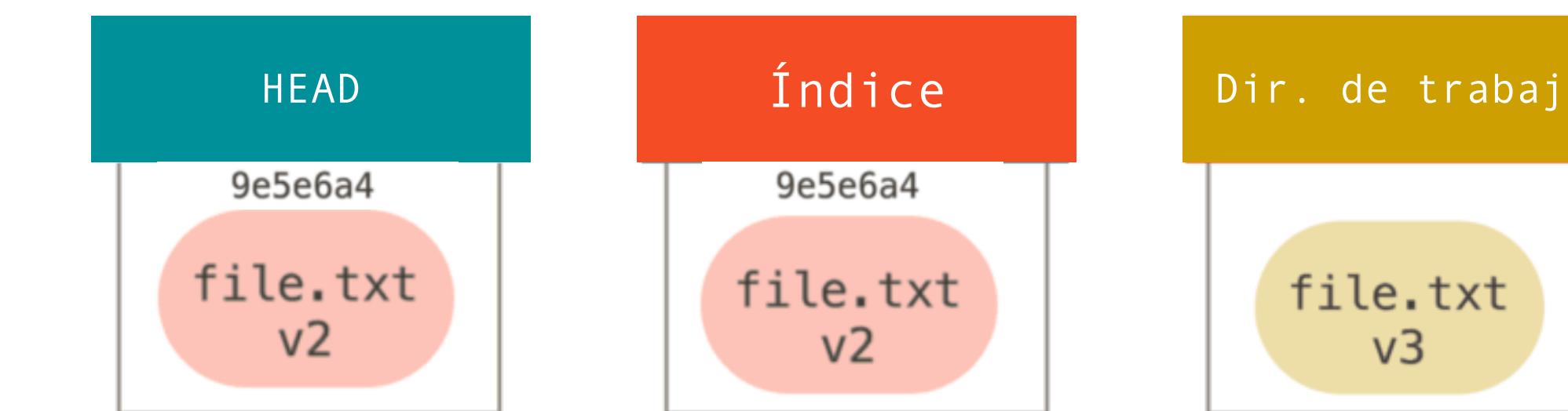
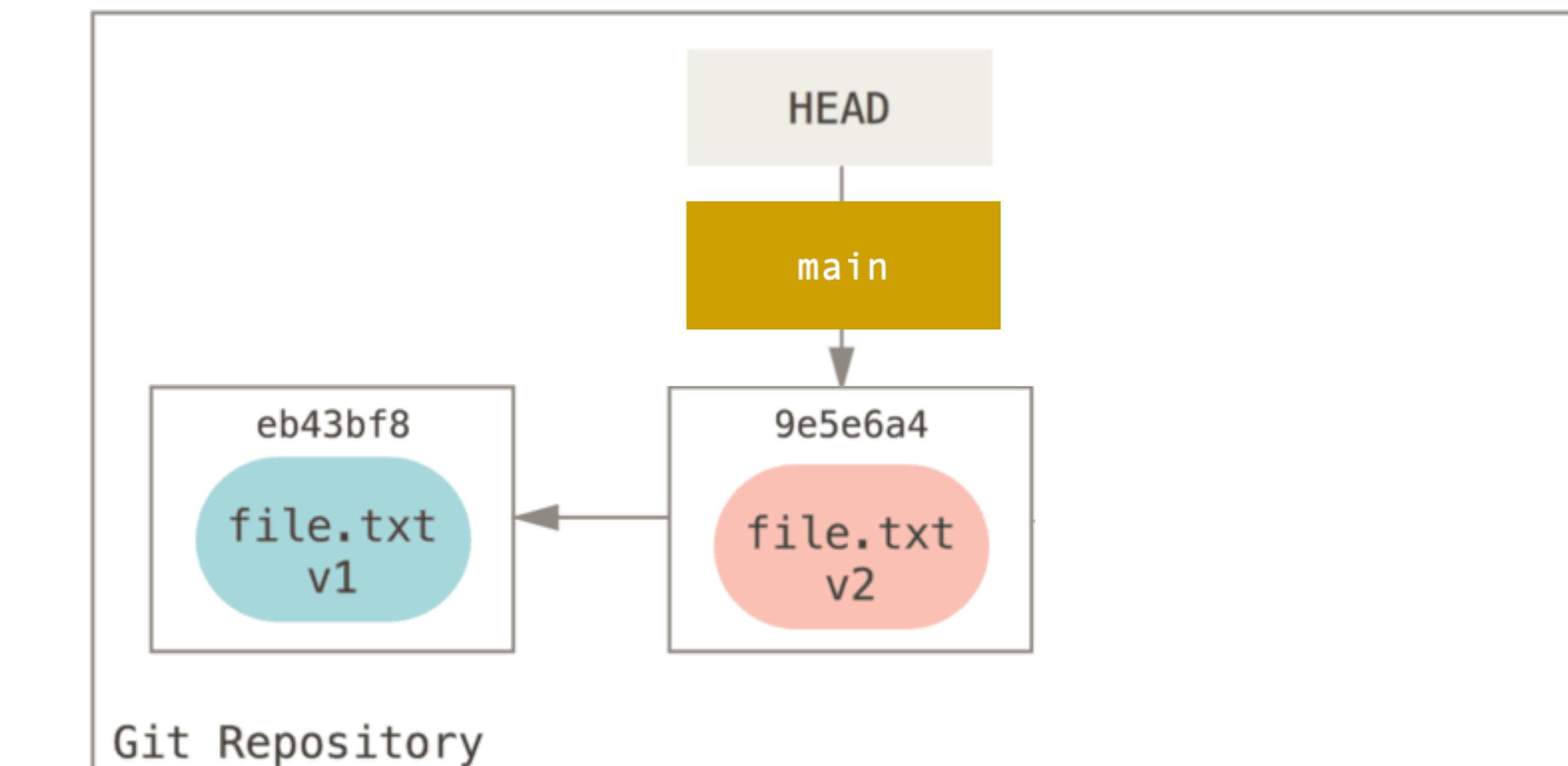
5. Ramas

6. Sincronización remota

- Realizamos nuevos cambios en el fichero

\$ \_

```
echo "nuevos cambios" >> file.txt
```





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

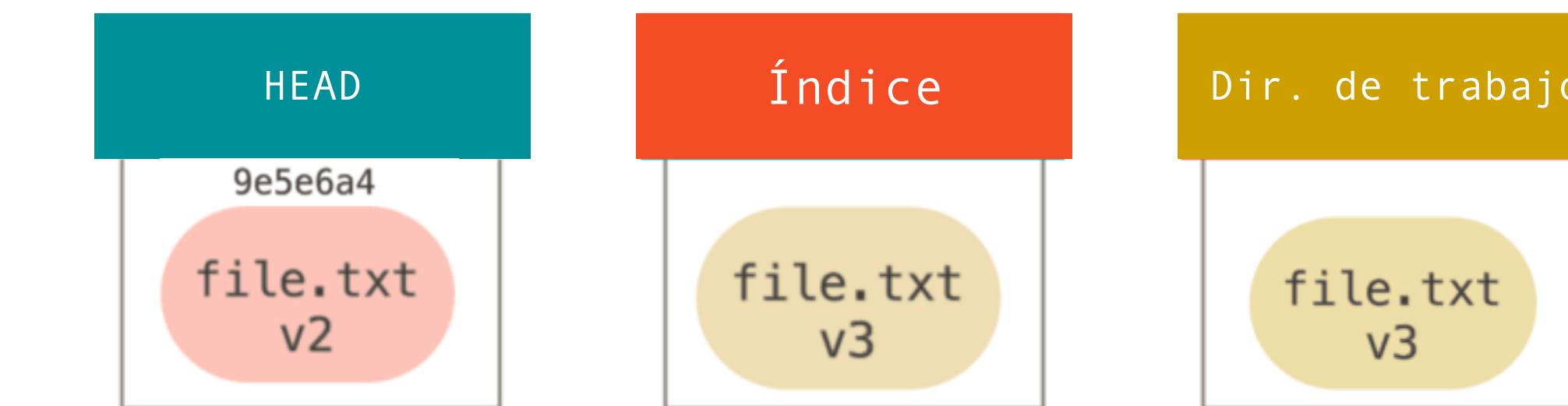
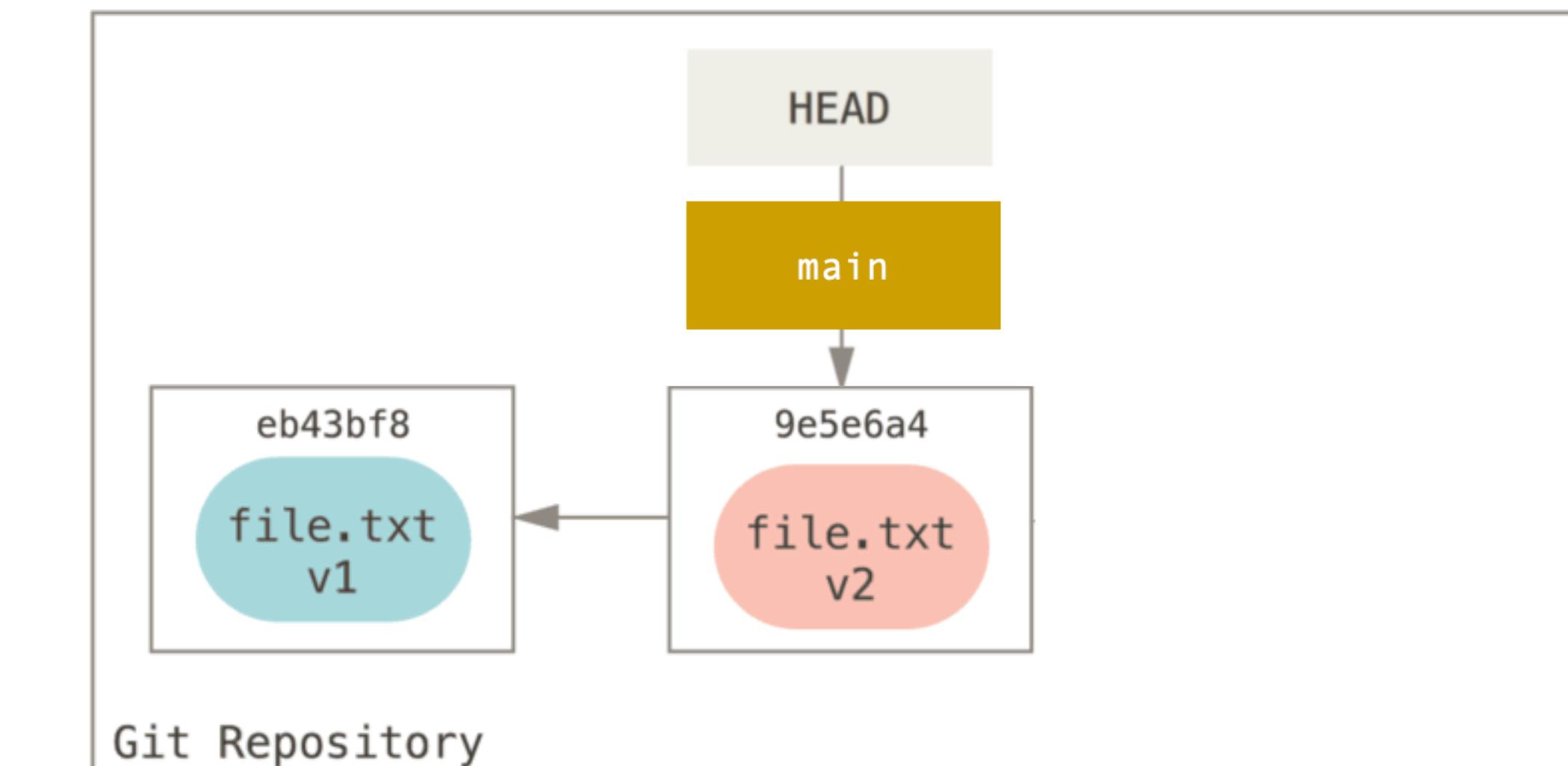
5. Ramas

6. Sincronización remota

- Añadimos el fichero a la zona de preparado (staged)

\$ \_

```
git add file.txt
```





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

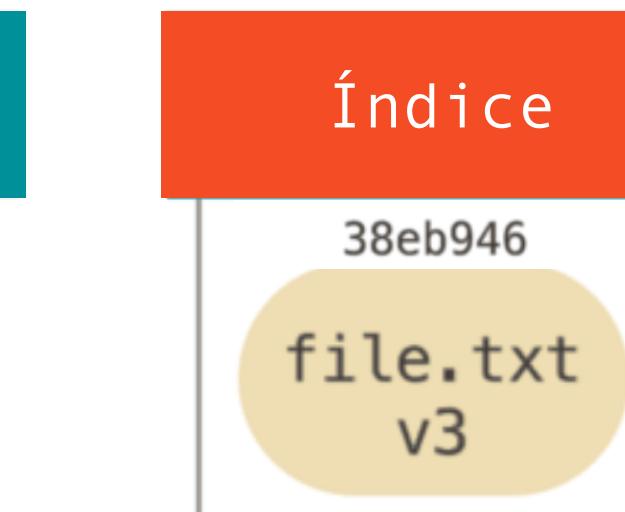
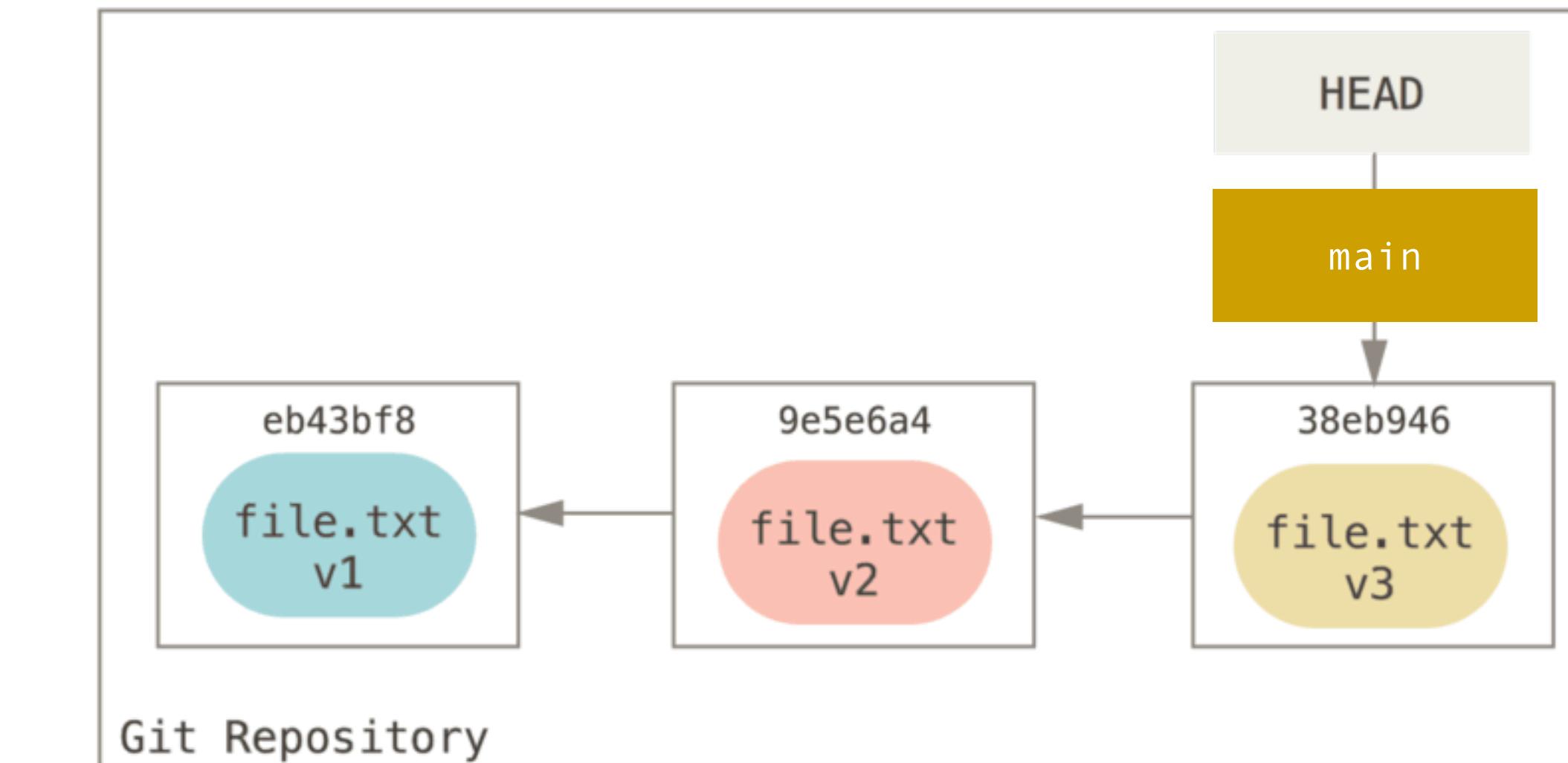
5. Ramas

6. Sincronización remota

## ○ Confirmamos los cambios

\$ \_

```
git commit -m "v3 file.txt"
```





# Resetear

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Otra manera de volver a un commit antiguo es utilizando el comando **git reset**

\$ \_

```
git revert [modo] [SHA o etiqueta]
```

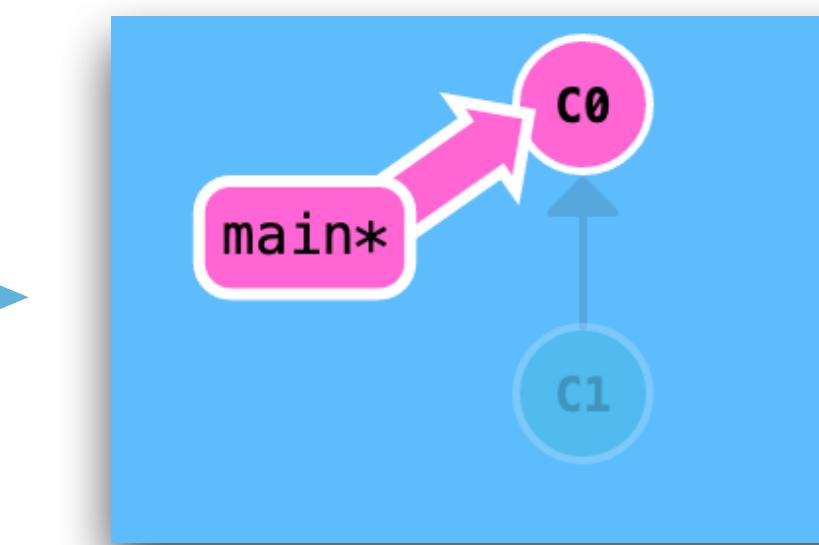
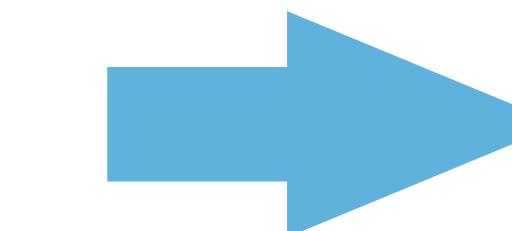
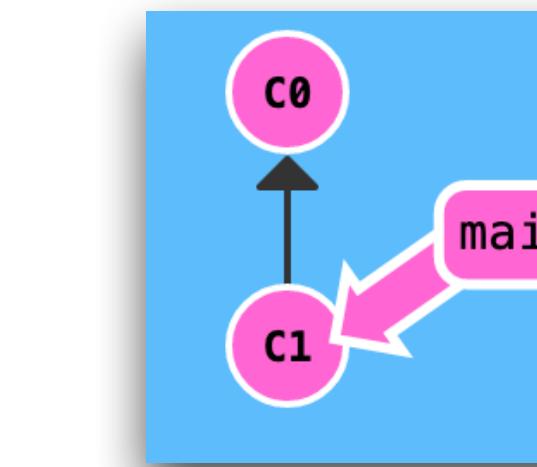
- En la práctica hay que indicar a qué commit se quiere volver

\$ \_

```
git reset HEAD^
```

\$ \_

```
git reset HEAD~1
```



- El modo de reseteo indica a qué repositorio del flujo de trabajo afecta:
  - soft: *HEAD*
  - mixed: *HEAD* e índice. Opción por defecto
  - hard: *HEAD*, índice y directorio de trabajo



# Resetear - Modos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

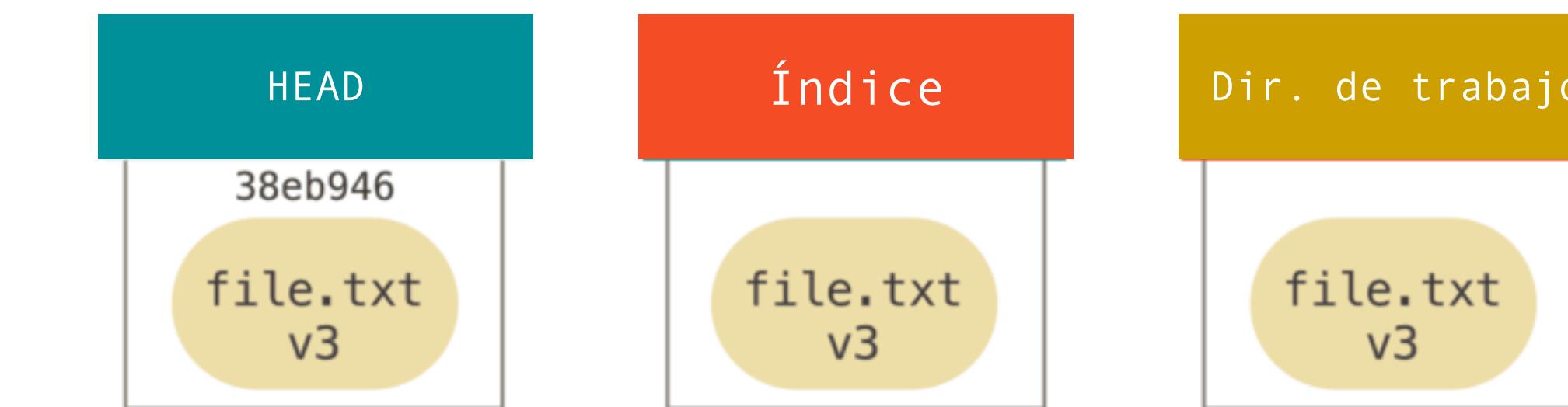
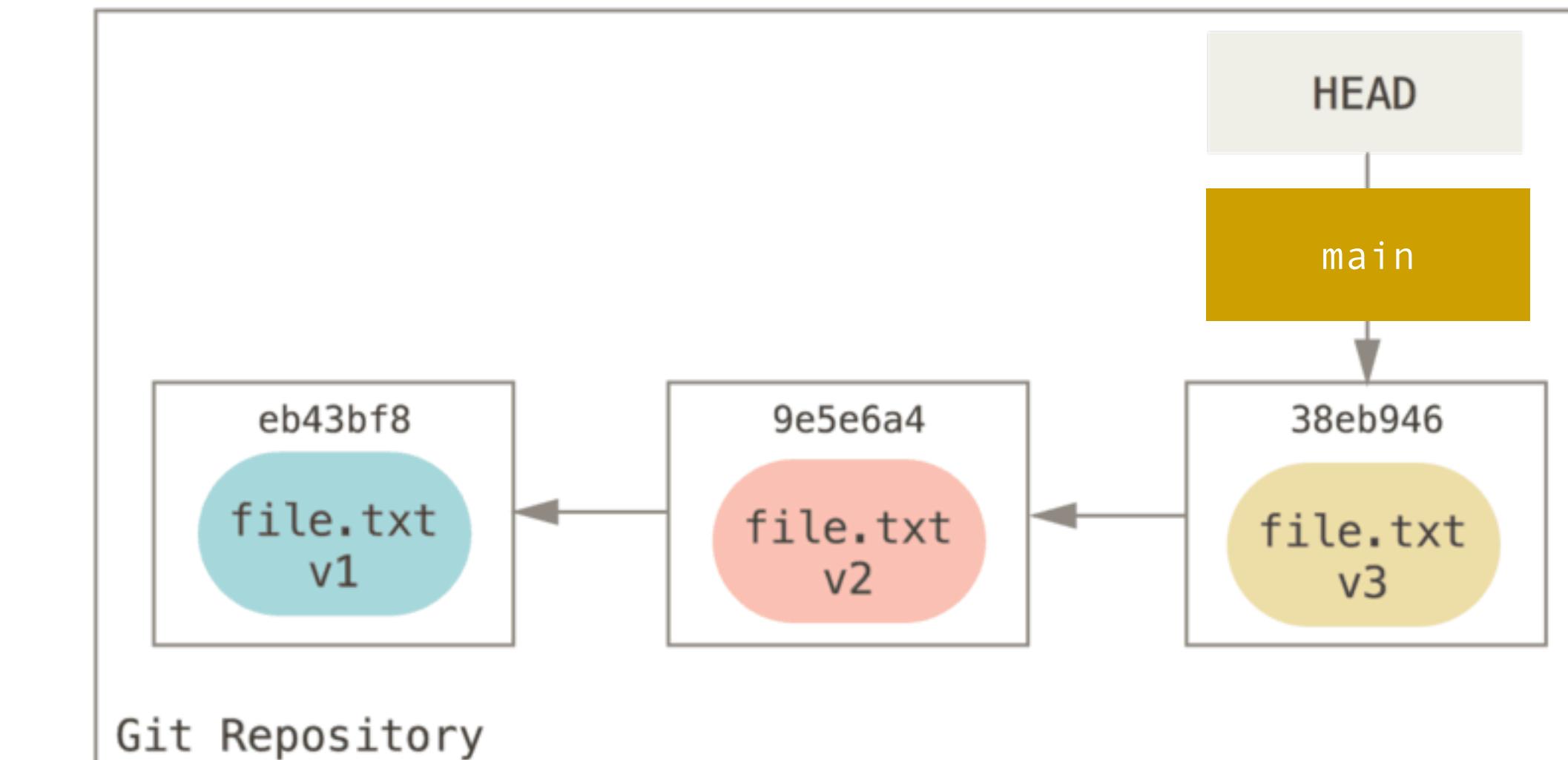
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Supongamos el siguiente escenario





# Resetear - Modos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

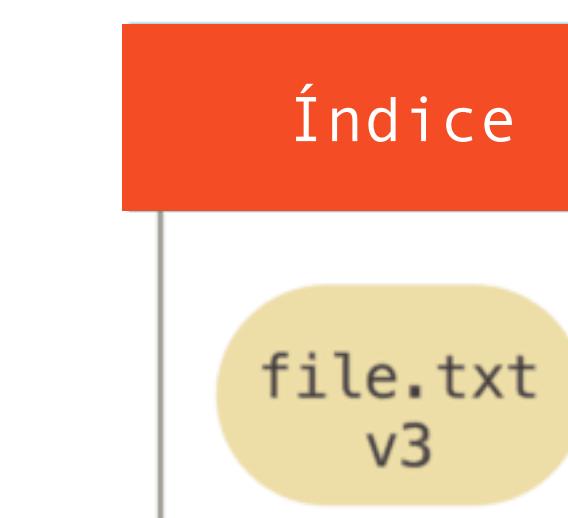
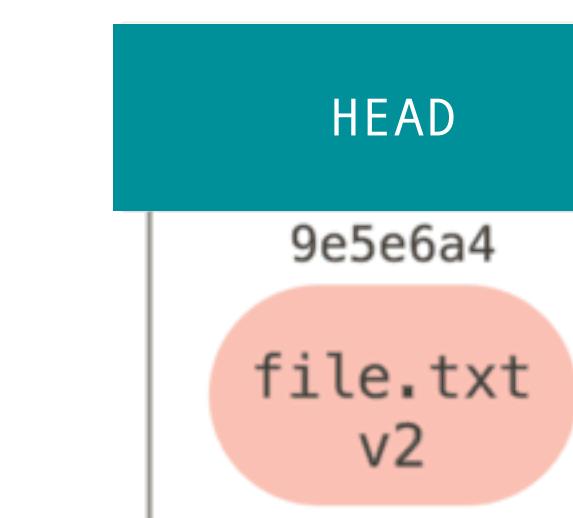
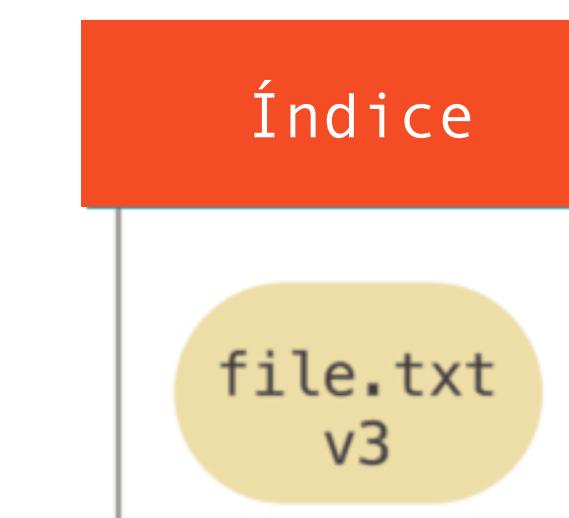
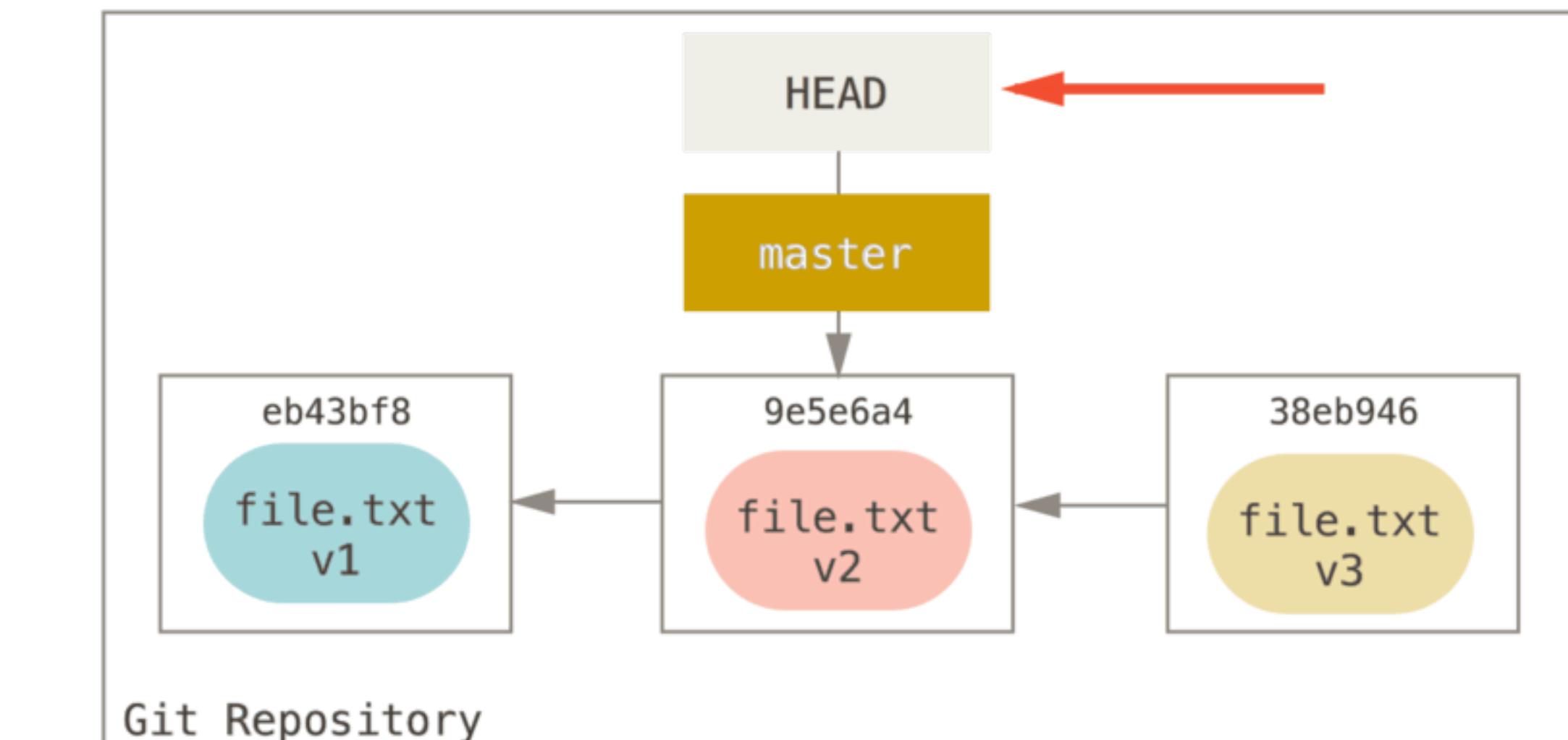
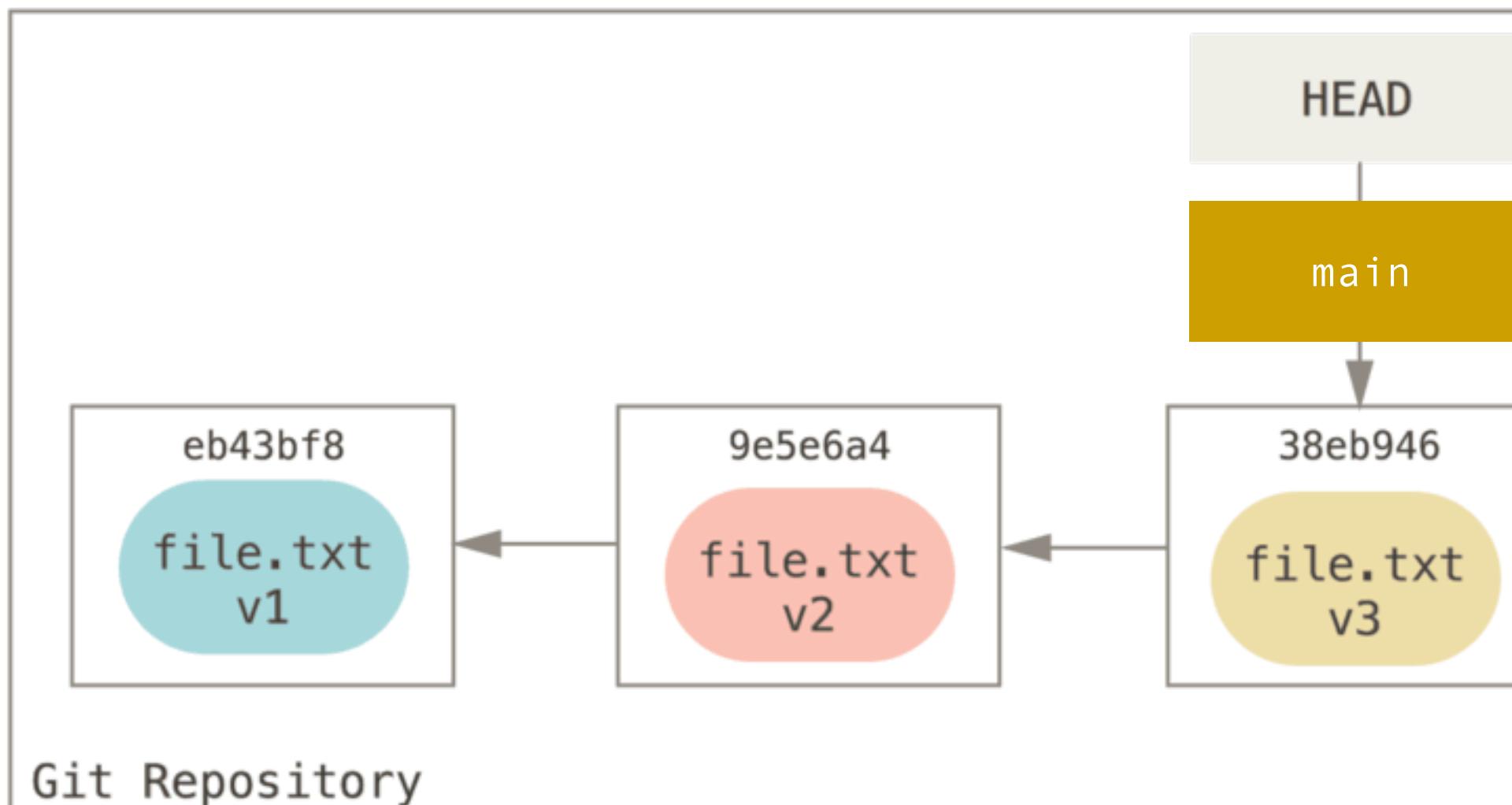
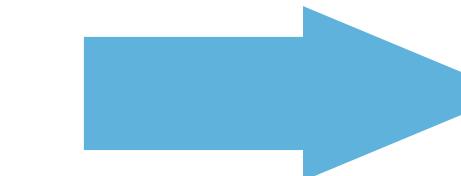
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

\$ \_ git reset --soft HEAD~1





# Resetear - Modos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

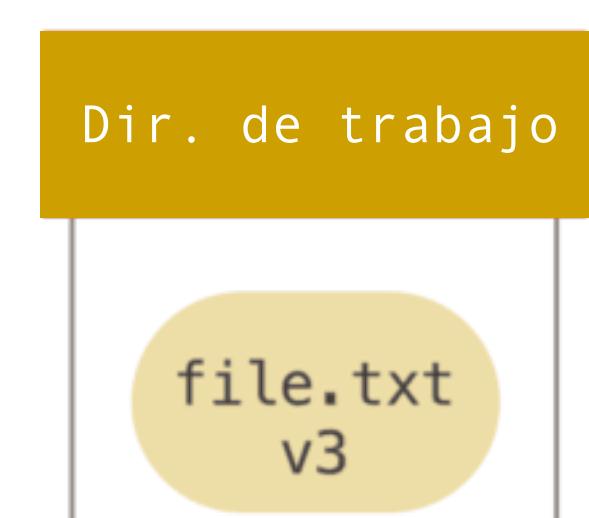
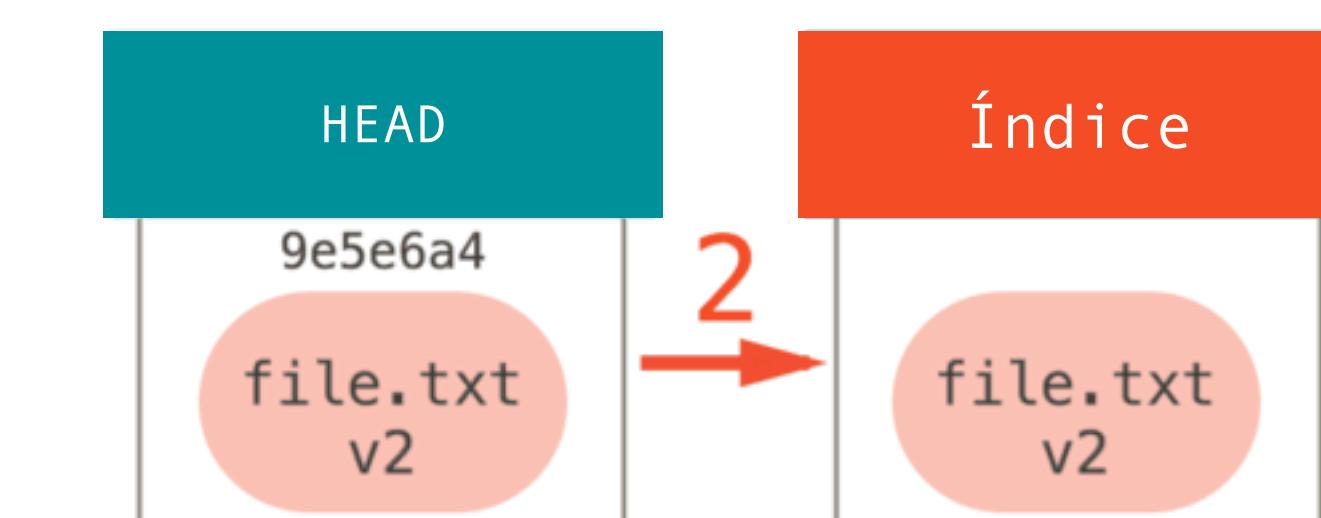
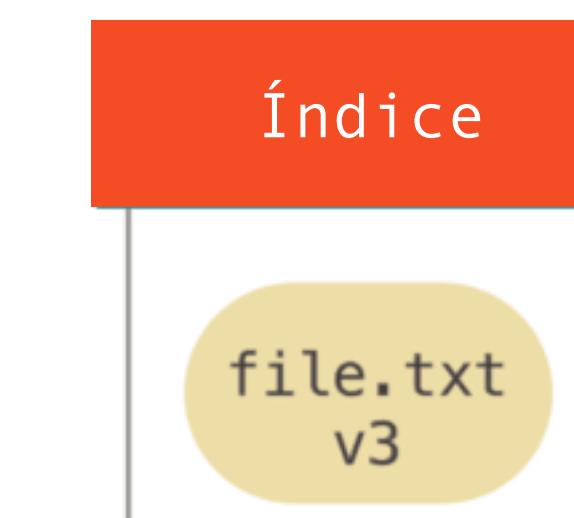
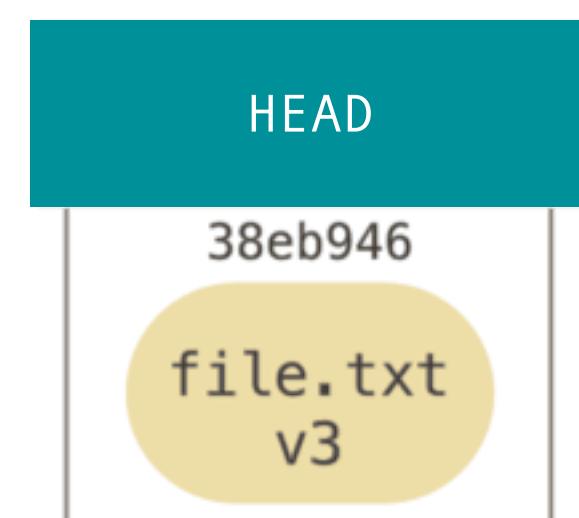
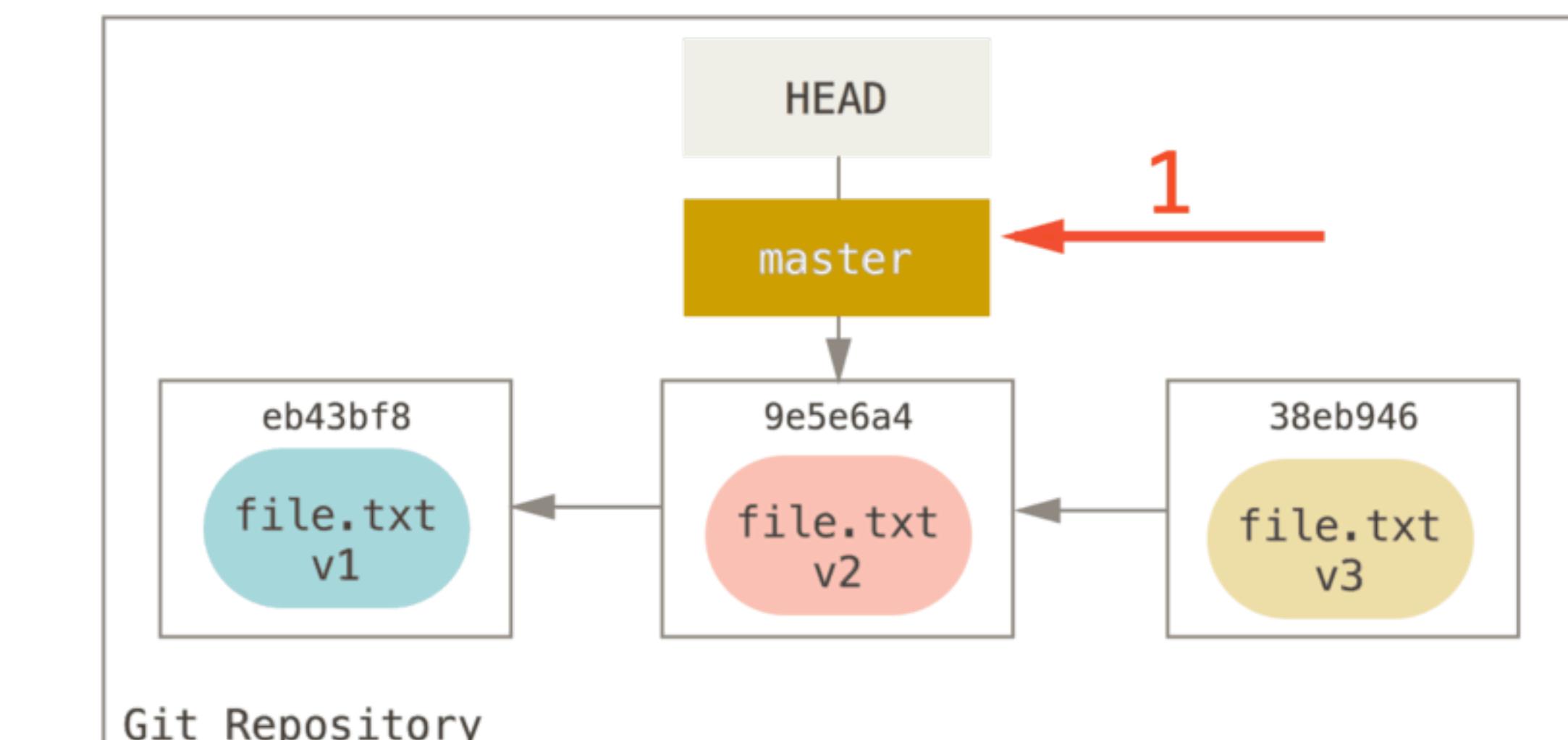
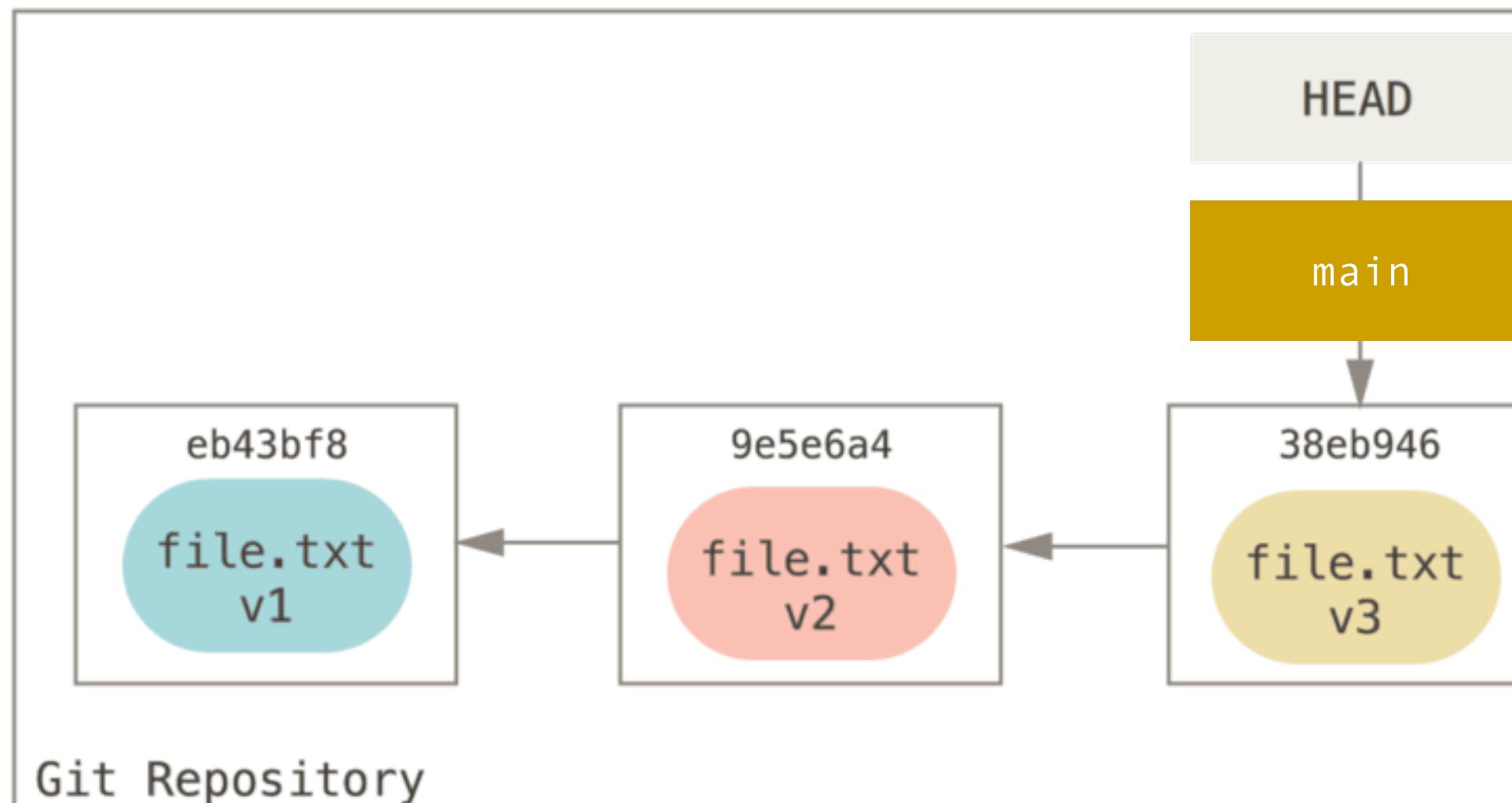
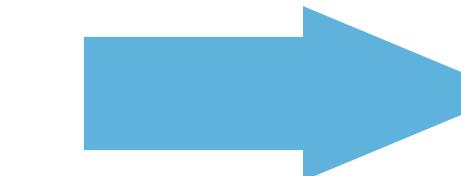
4. Correcciones

5. Ramas

6. Sincronización  
remota

\$ \_

```
git reset --mixed HEAD~1
```





# Resetear - Modos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

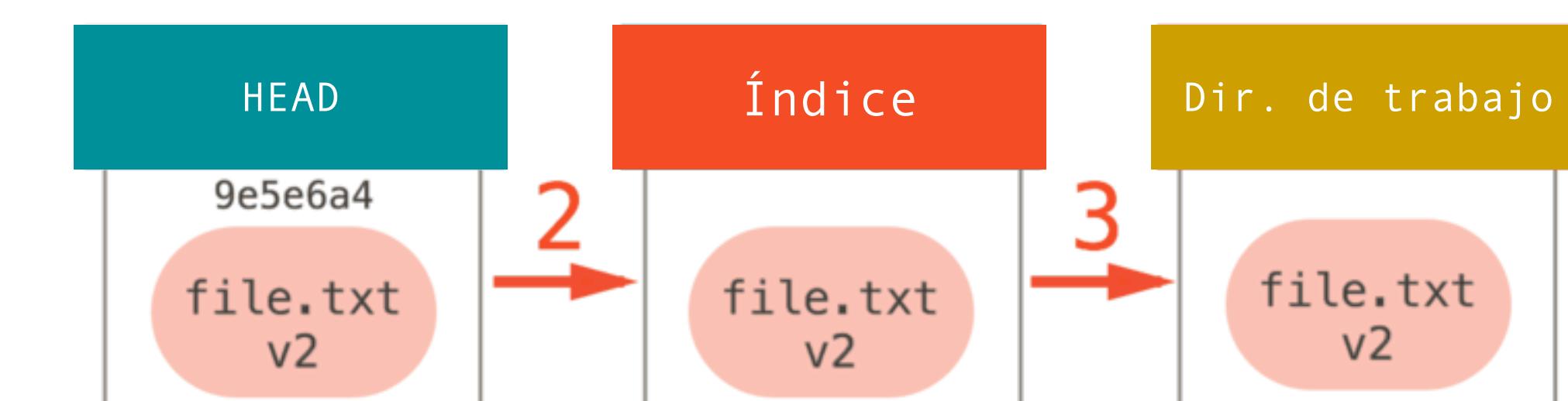
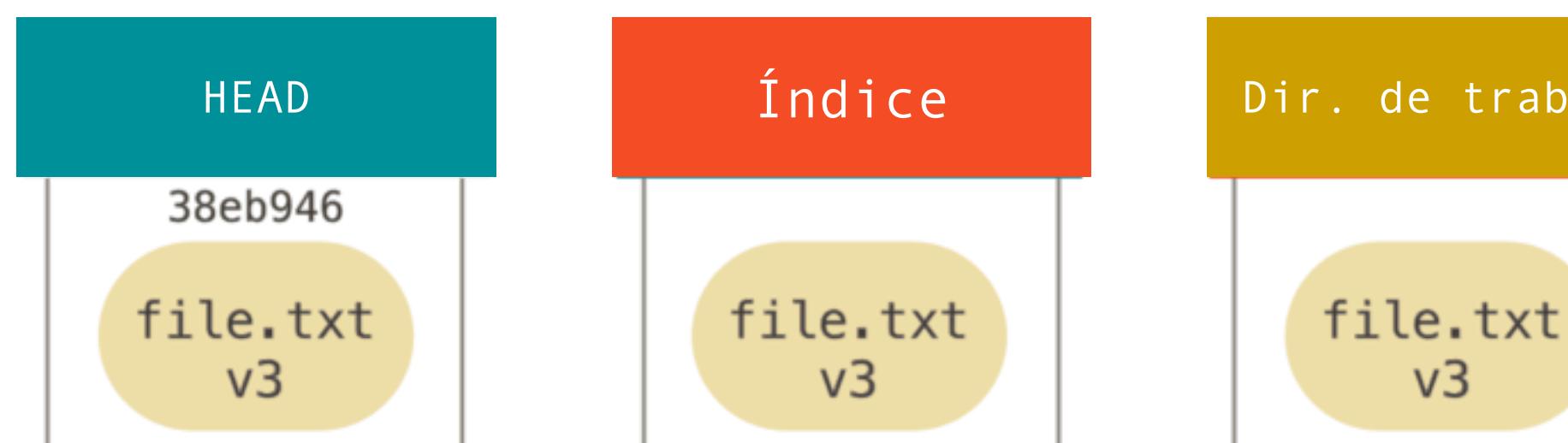
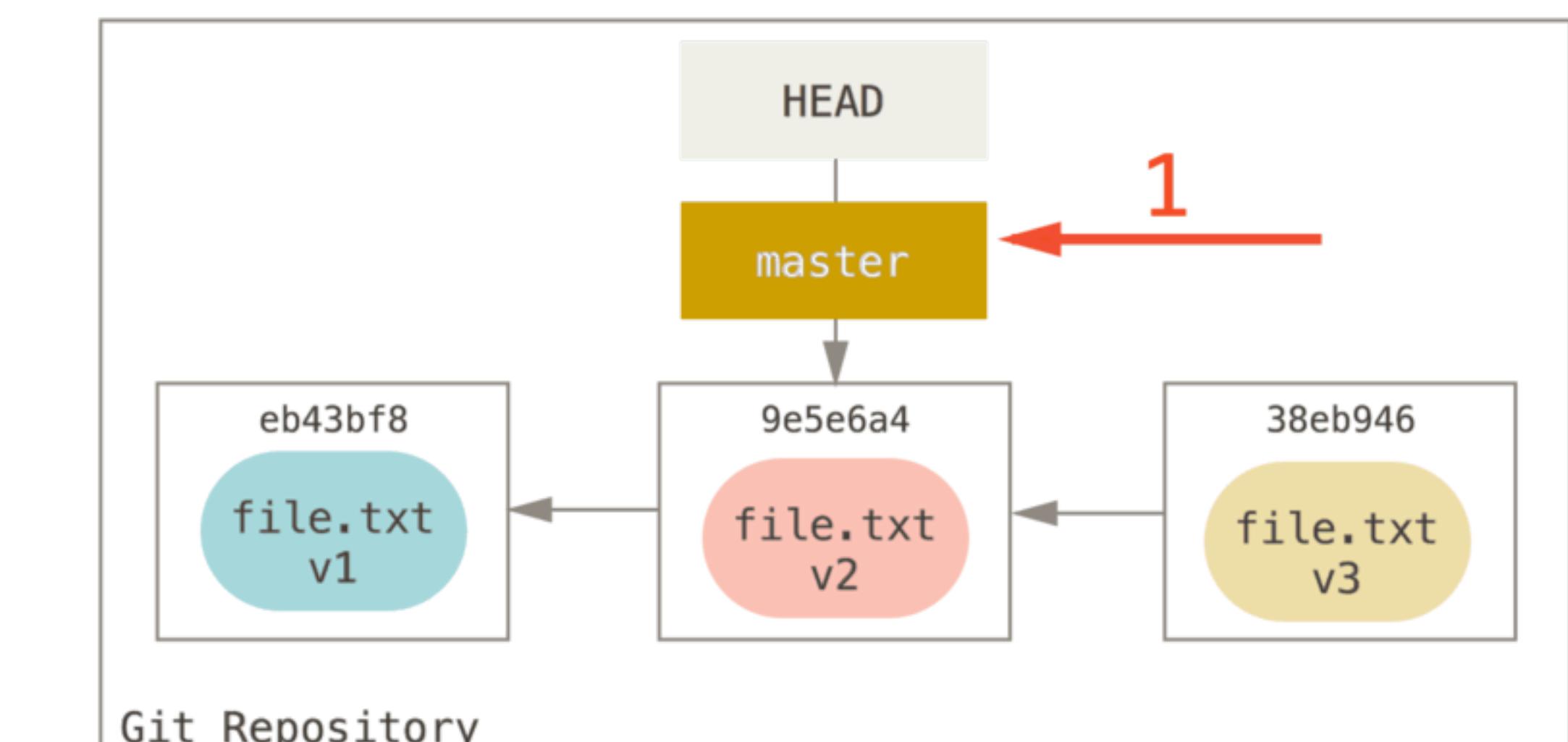
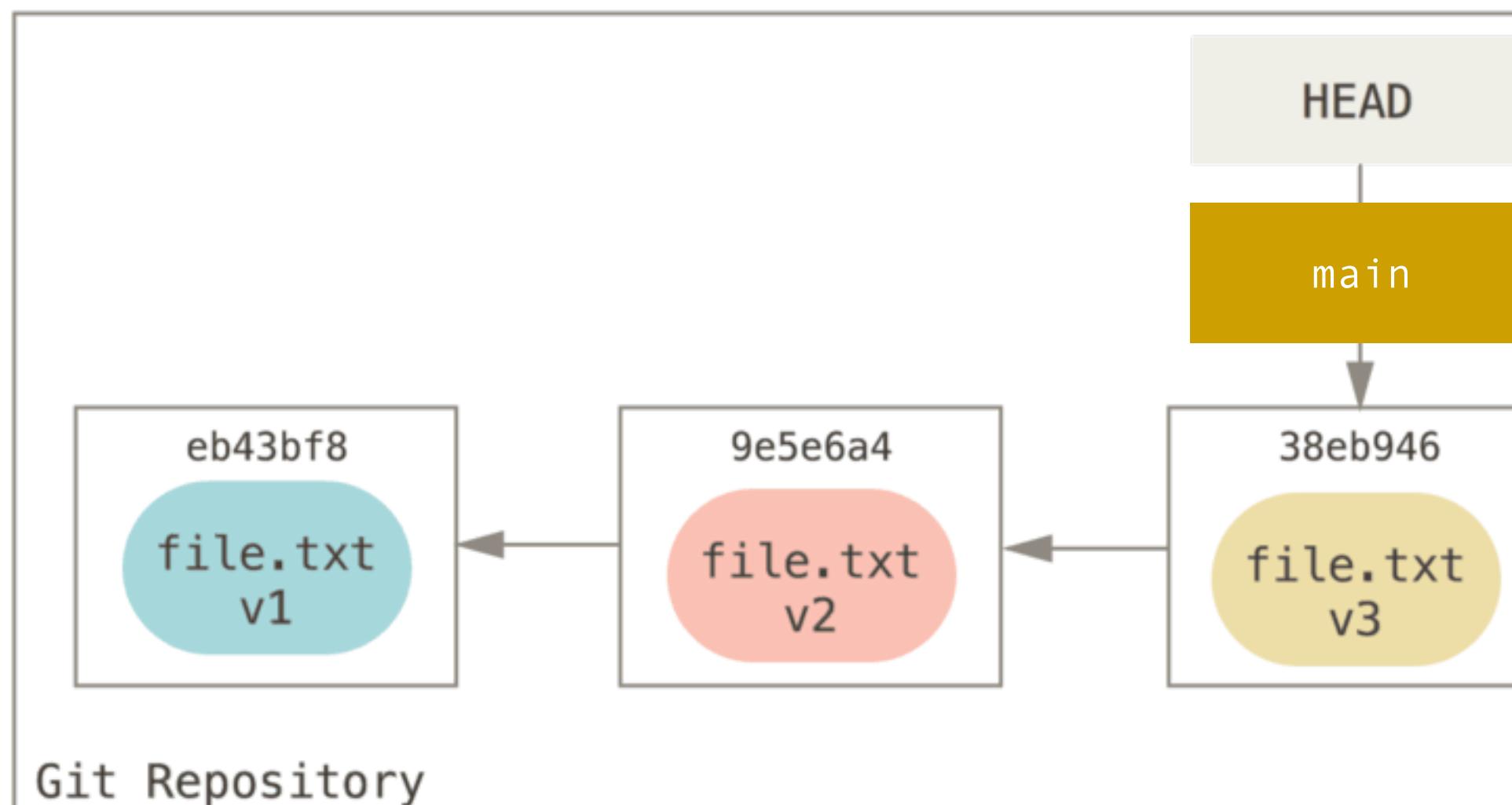
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

\$ \_ git reset --hard HEAD~1



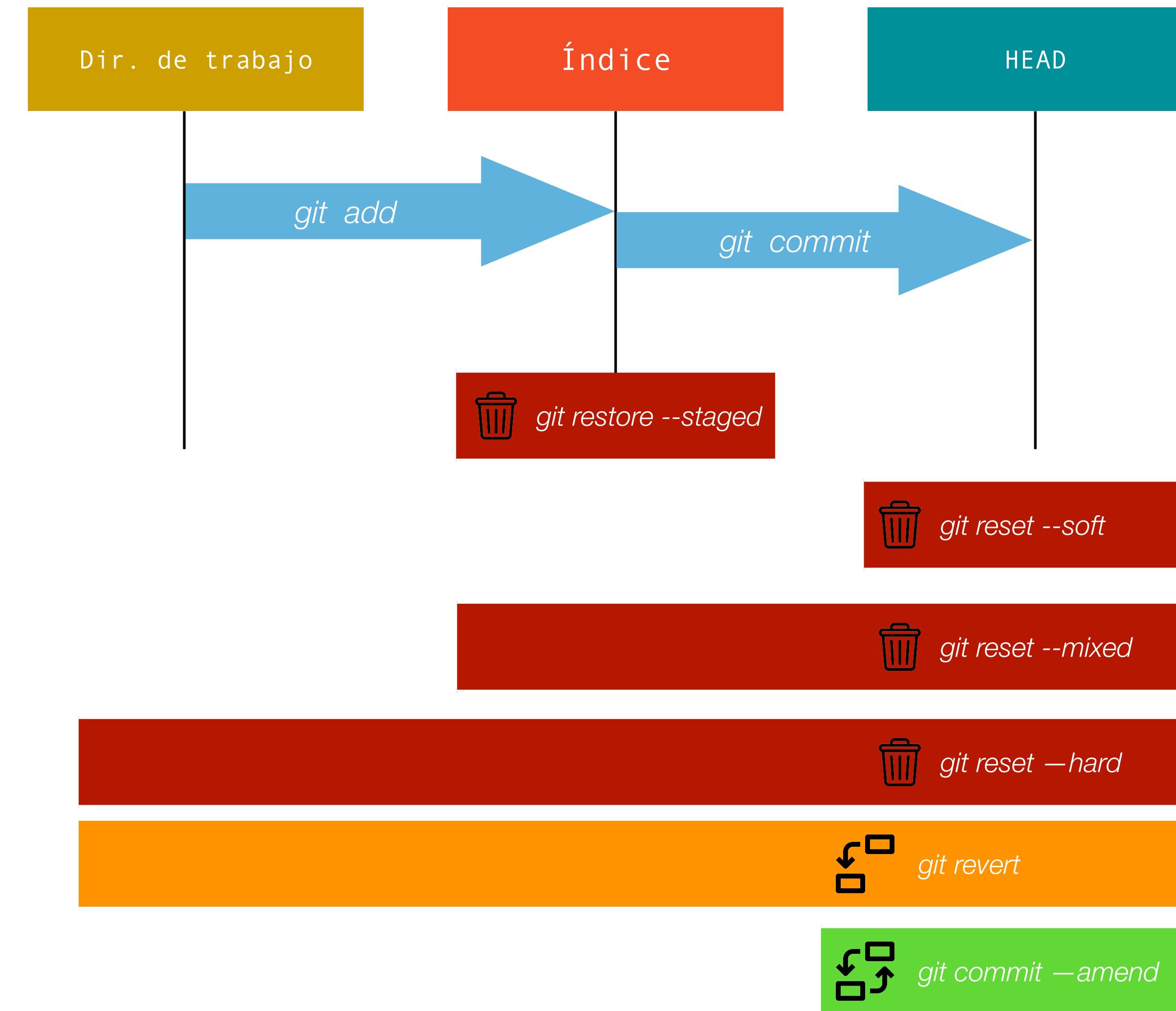


# Resumen

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota





# Resumen

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

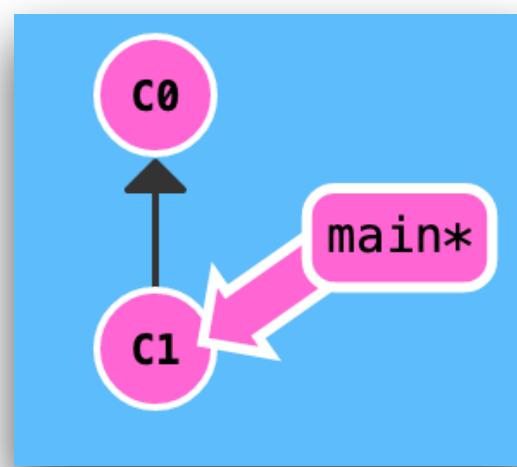
2. Configuración

3. Control de  
cambios en local

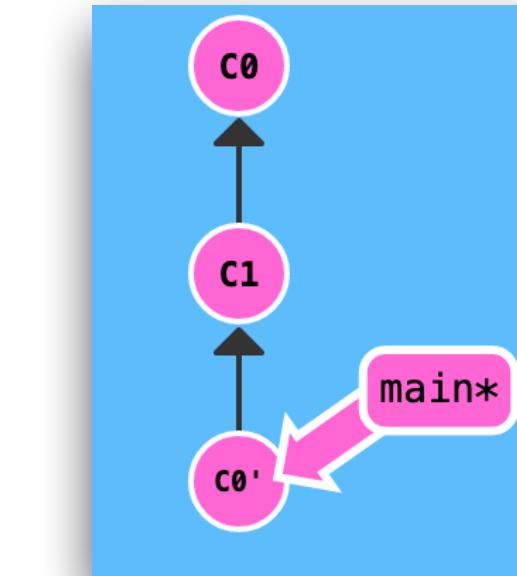
4. Correcciones

5. Ramas

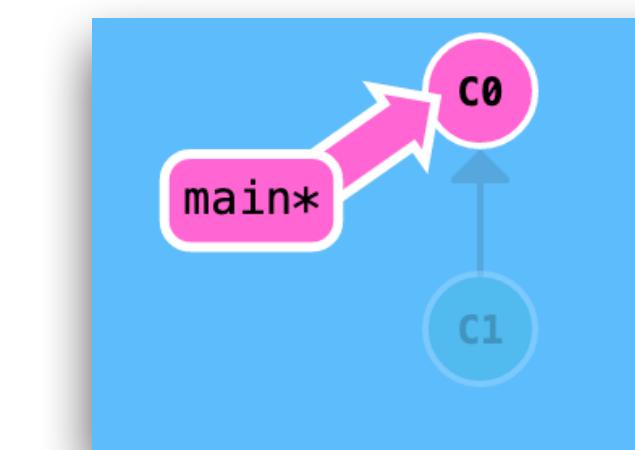
6. Sincronización  
remota



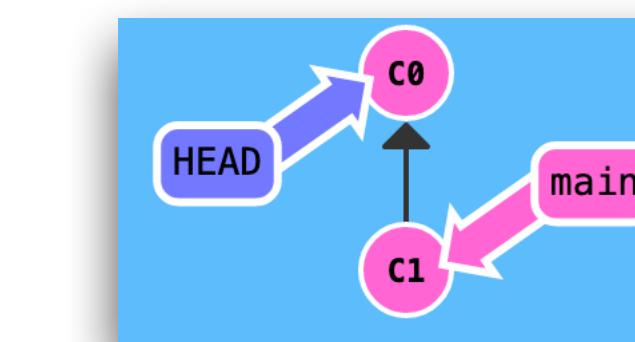
*git revert*



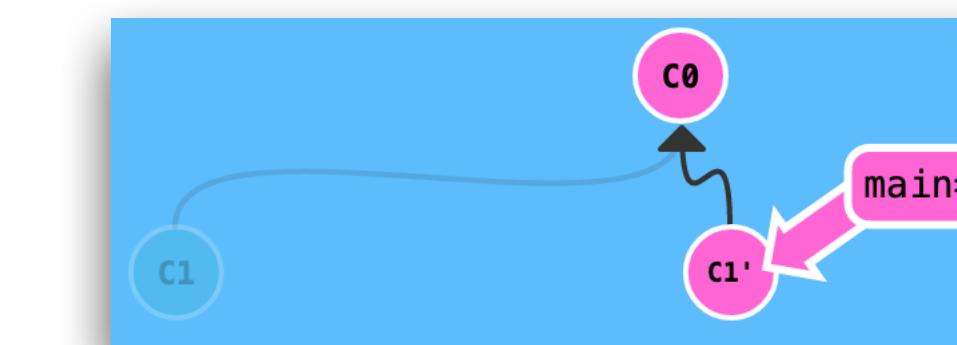
*git reset*



*git checkout*



*git commit --amend*





GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



# Deshacer cambios - Ejercicio 4.1

- En el mismo repositorio con los fuentes de latex, realiza las siguientes tareas sobre el fichero document.tex:
- Elimina la referencia a la imagen, añade este cambio a un commit. Deshaz este cambio **sin eliminar ningún commit**.
  - Elimina la referencia a la imagen pero no la añadas el cambio a la zona de preparación (no hagas git add). Deshaz este cambio.
  - Elimina la referencia a la imagen y añade el cambio a la zona de preparación (git add). Deshaz este cambio.
  - Elimina la referencia a la imagen, añade el cambio a un commit (git add + git commit). Deshaz este cambio.



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## 5. Ramas



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

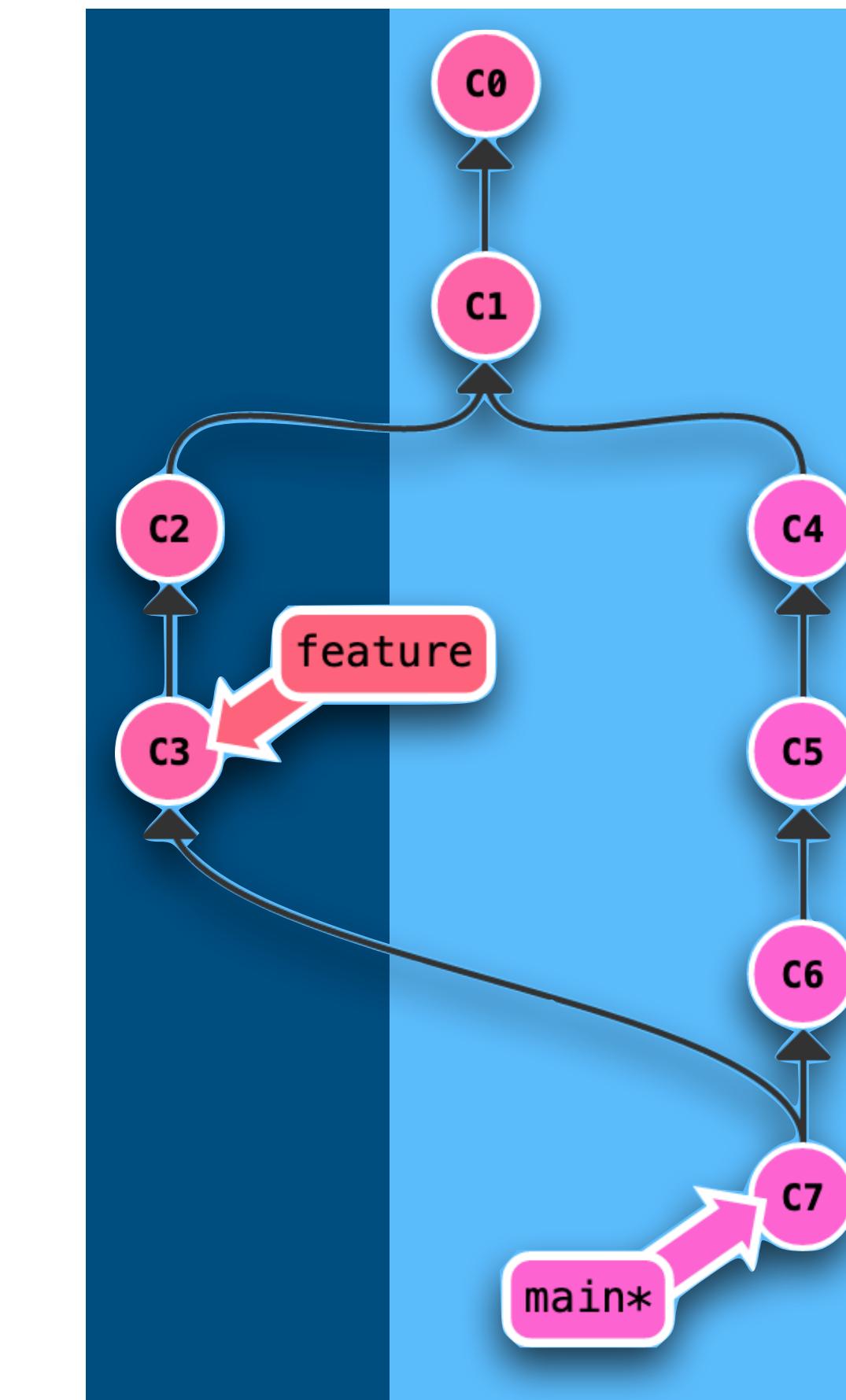
6. Sincronización remota

## Idea

- Las ramas permiten crear **líneas de trabajo en paralelo** que eventualmente se pueden juntar.
- Separar las versiones estables de los desarrollos.

## Posibles escenarios

- Desarrollo de software científico
  - Nueva funcionalidad
  - Corrección de errores
- Repositorio con material docente
  - Queremos empezar a desarrollar el material del año siguiente sin modificar el del curso actual



Rama  
feature      Rama  
principal

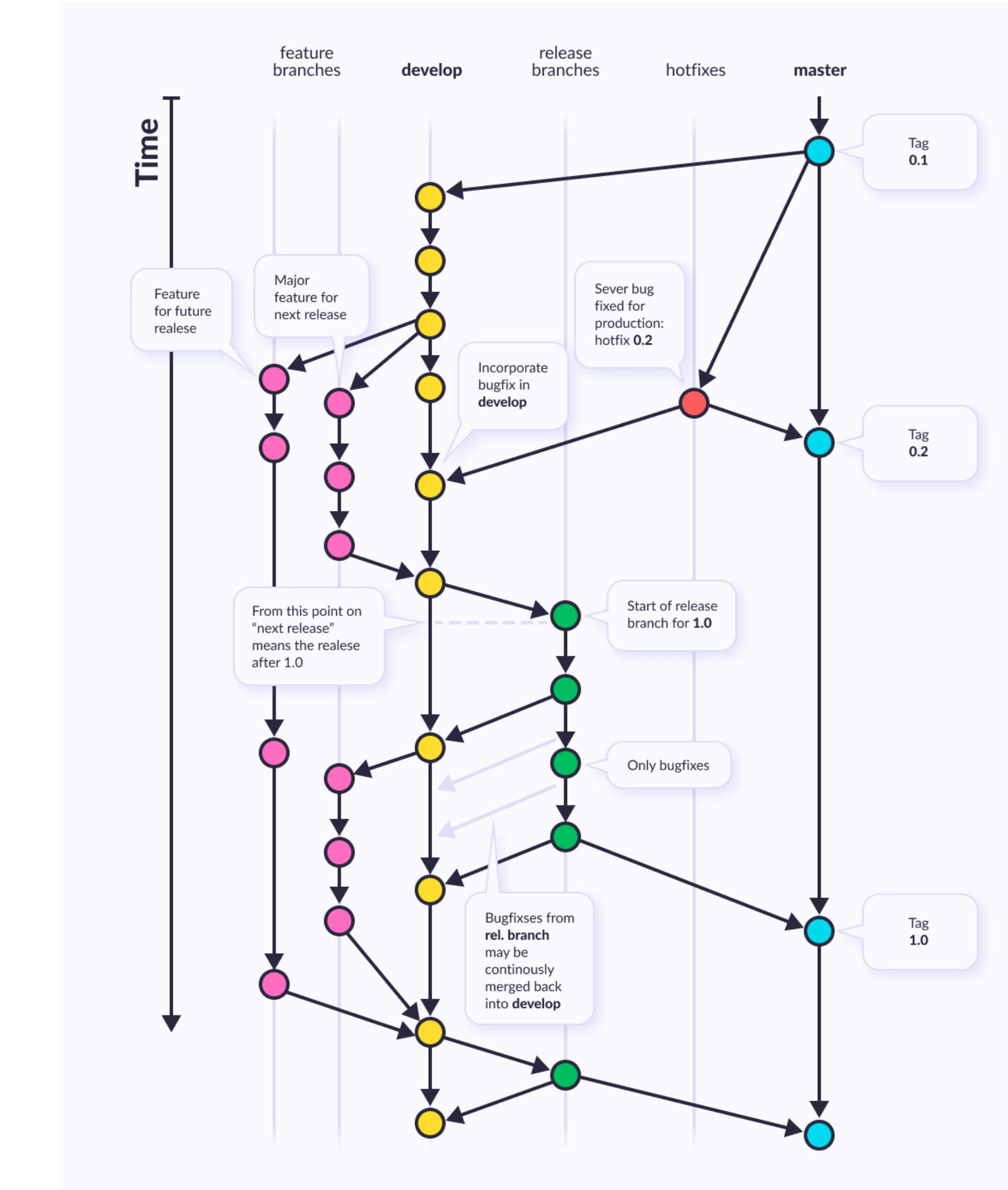


# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Una rama se crea a partir de otra con el comando **git branch**:

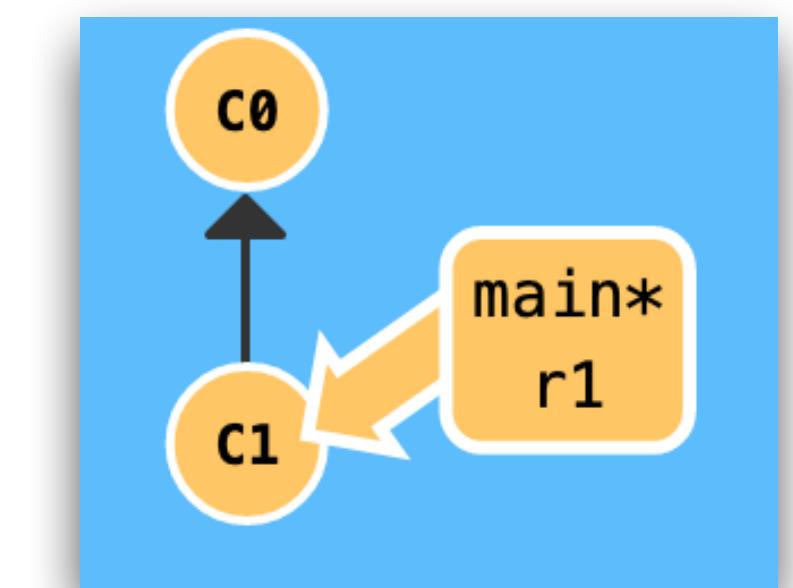
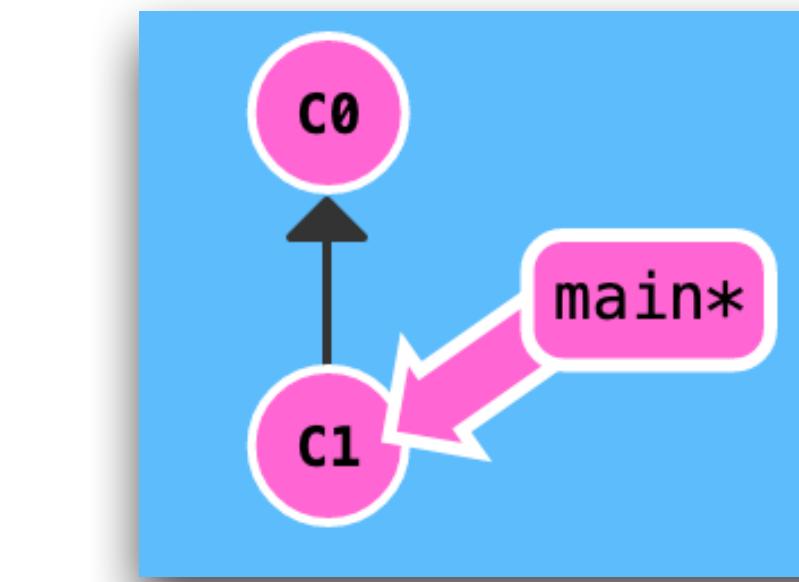
\$ \_

```
git branch [nombre_rama]
```

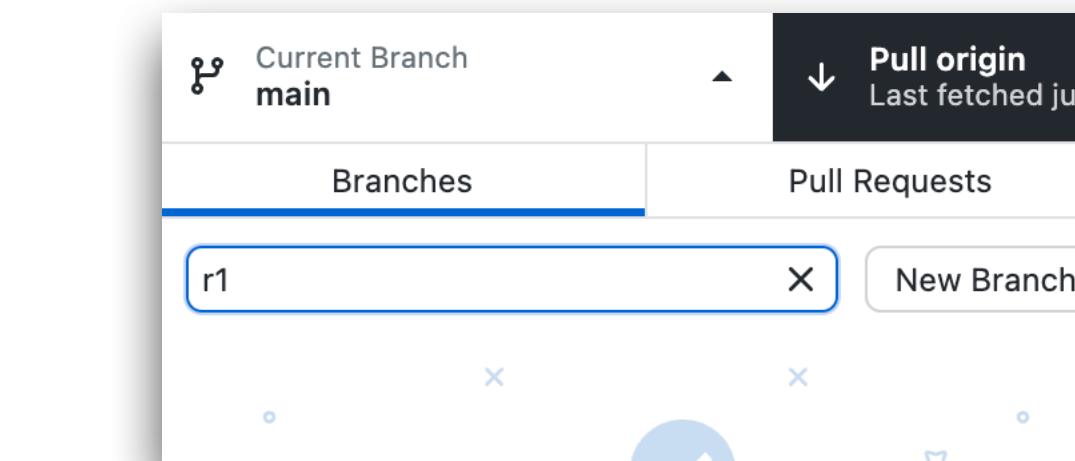
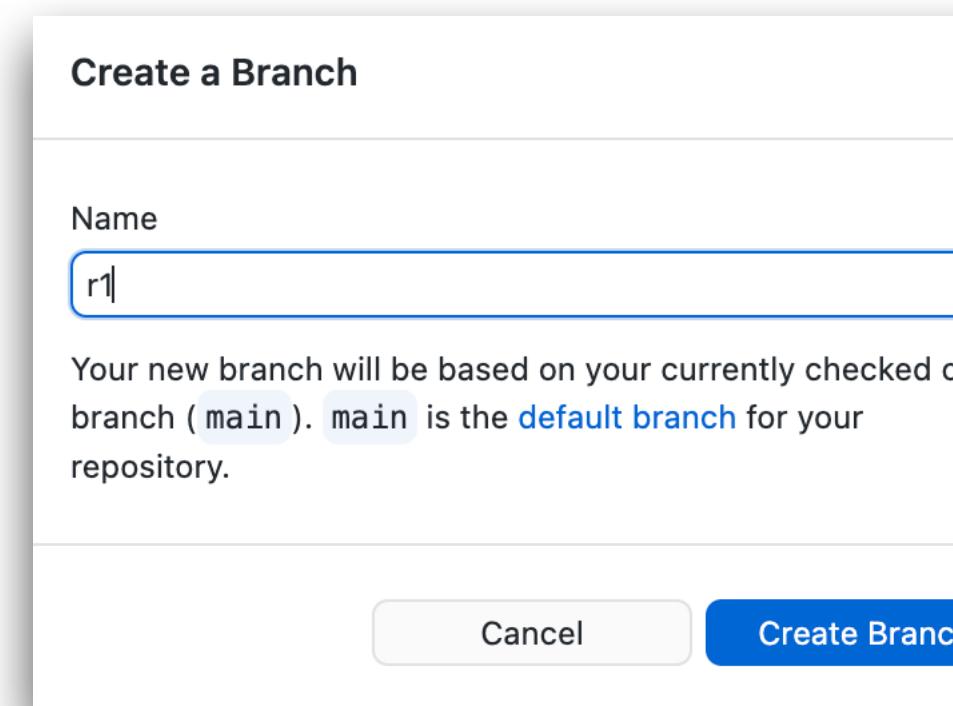
- Por ejemplo, para crear una rama `r1` a partir de la `main` (rama activa):

\$ \_

```
git branch r1
```



- En GitHub Desktop:





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Una rama se crea a partir de otra con el comando **git branch**:

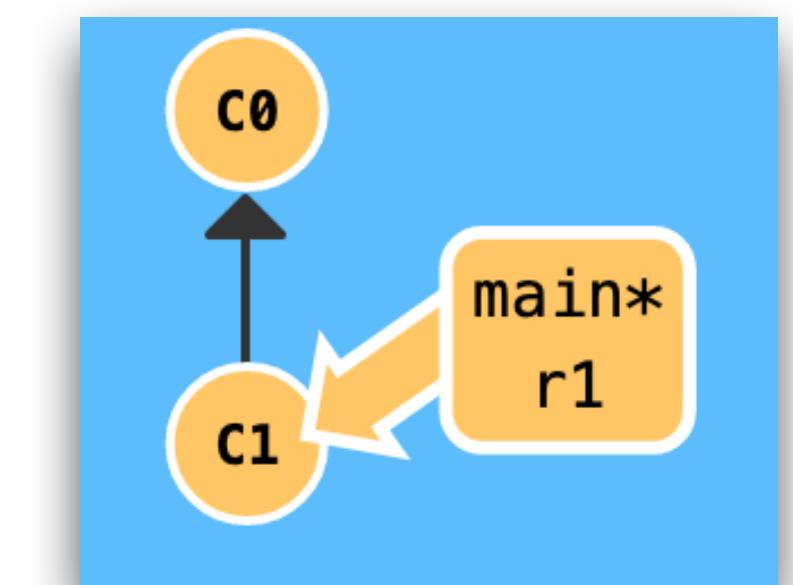
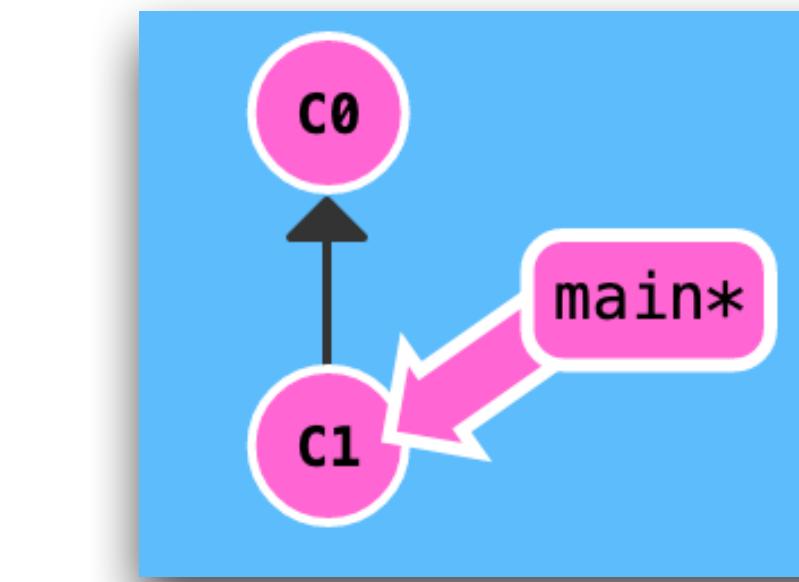
\$ \_

```
git branch [nombre_rama]
```

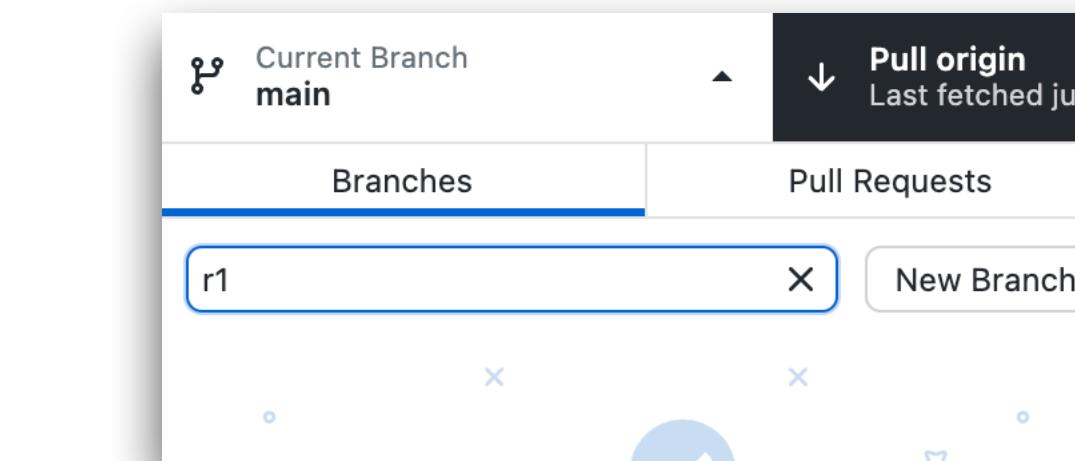
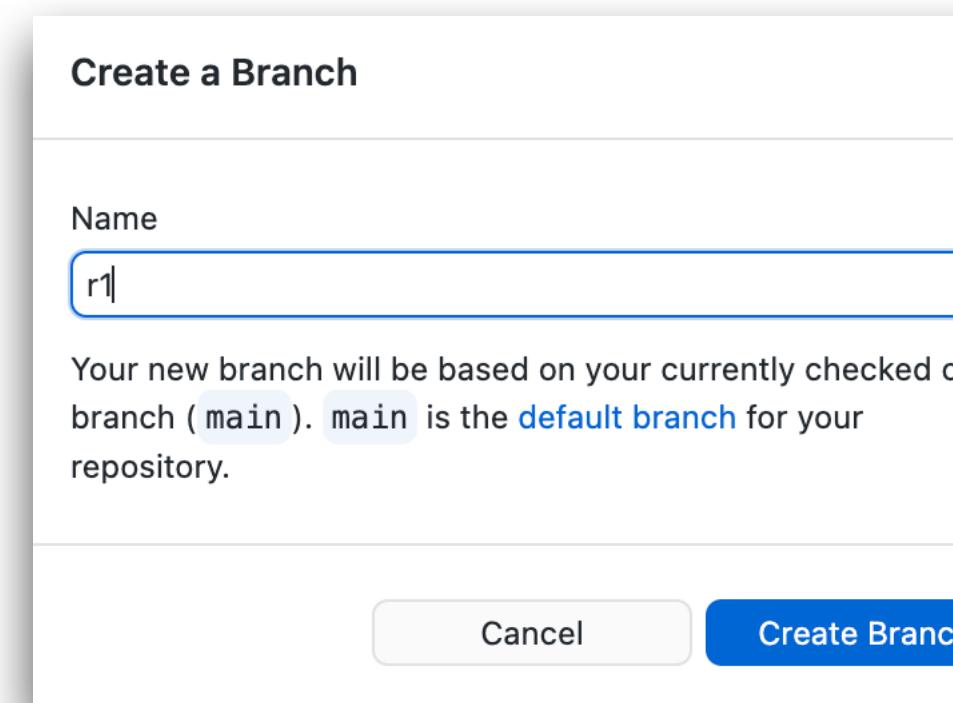
- Por ejemplo, para crear una rama `r1` a partir de la `main` (rama activa):

\$ \_

```
git branch r1
```



- En GitHub Desktop:





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

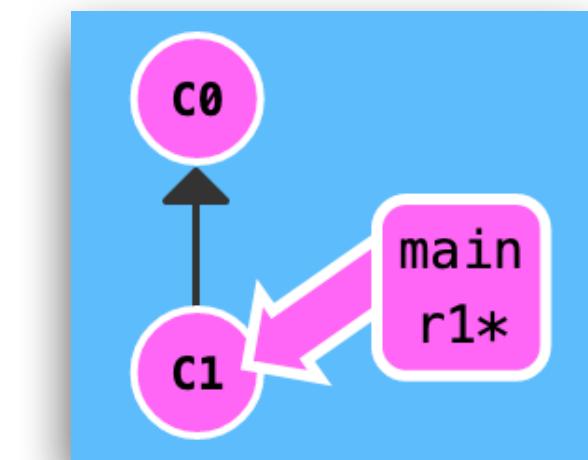
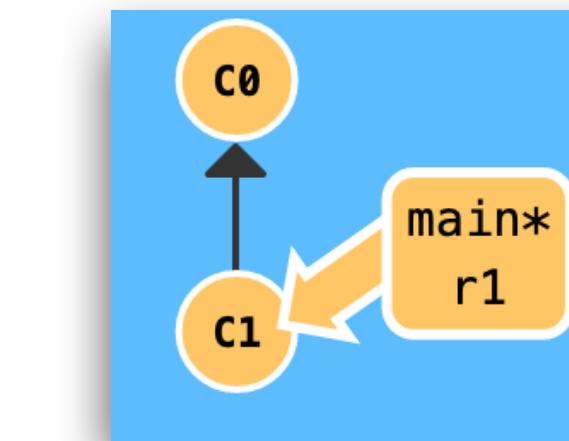
6. Sincronización  
remota

- Hay que tener en cuenta:

- La rama activa se cambia con el comando checkout:

\$ \_

```
git checkout r1
```



- Si se quiere crear una rama y cambiarse a ella, hay que añadir la opción -b:

\$ \_

```
git checkout -b [nombre_rama]
```

=

\$ \_

```
git branch [nombre_rama]
```

+

\$ \_

```
git checkout [nombre_rama]
```



# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

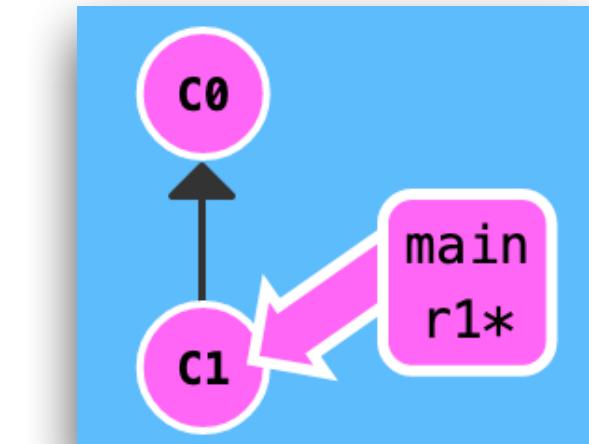
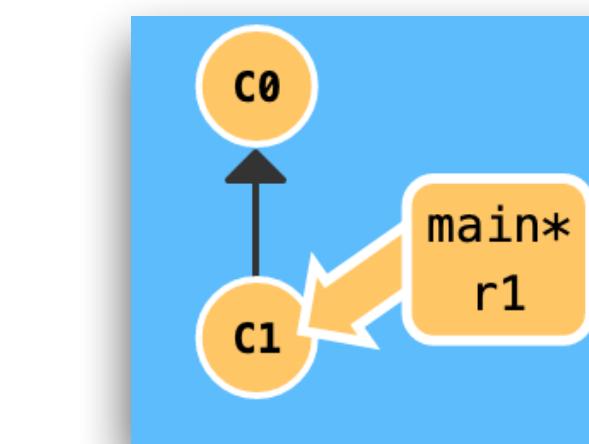
- Hay que tener en cuenta:

- Las ramas básicamente son "etiquetas inteligentes".
- Siempre hay una activa (asociada a HEAD)
- Al hacer un commit, la rama activa se desplaza junto con el HEAD

- La rama activa se cambia con el comando checkout:

\$ \_

```
git checkout r1
```



- Si se quiere crear una rama y cambiarse a ella, hay que añadir la opción -b:

\$ \_

```
git checkout -b [nombre_rama]
```

=

\$ \_

```
git branch [nombre_rama]
```

+

\$ \_

```
git checkout [nombre_rama]
```



# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

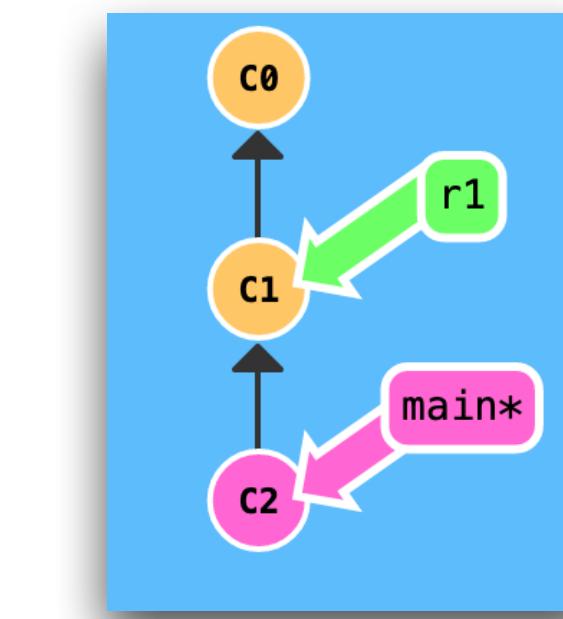
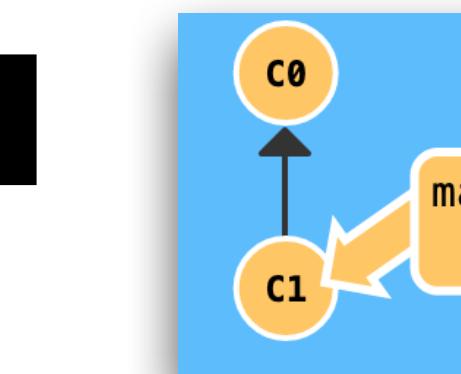
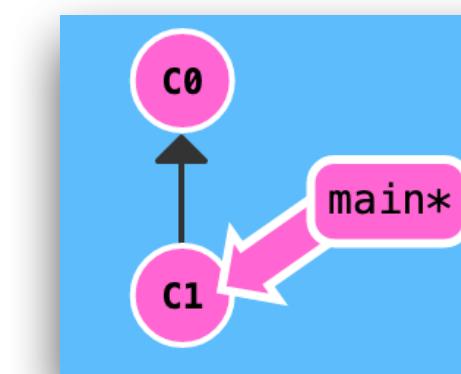
4. Correcciones

5. Ramas

6. Sincronización  
remota

\$ \_ git branch r1

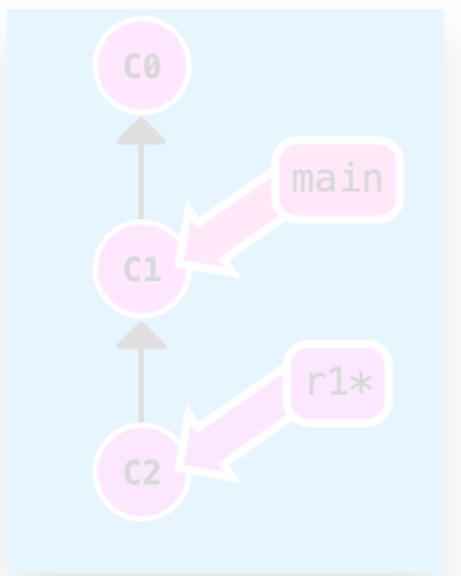
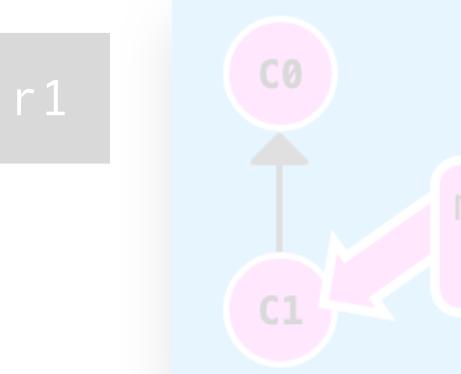
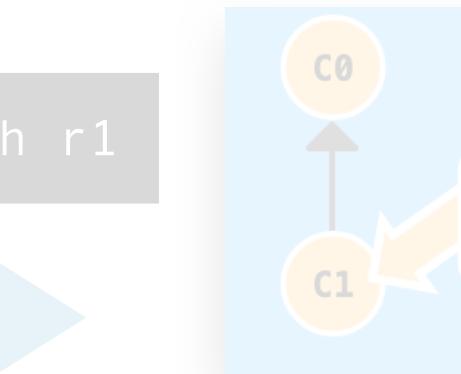
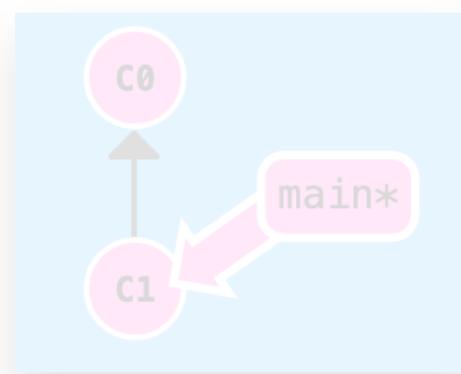
\$ \_ git commit



\$ \_ git branch r1

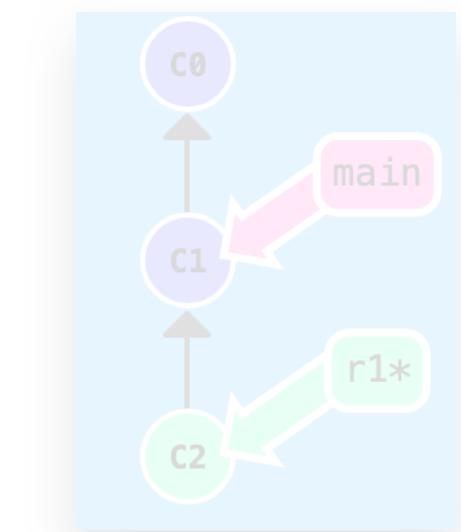
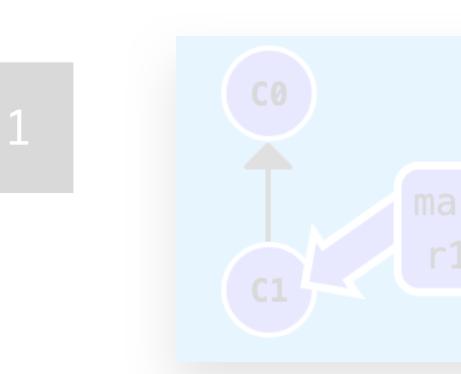
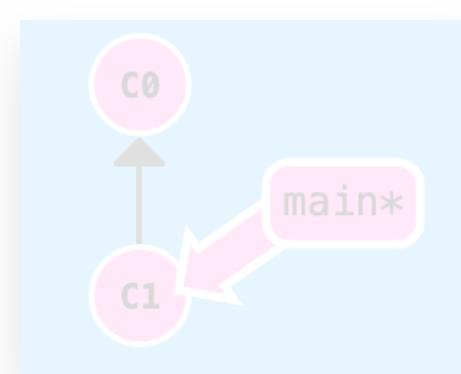
\$ \_ git checkout r1

\$ \_ git commit



\$ \_ git checkout -b r1

\$ \_ git commit





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

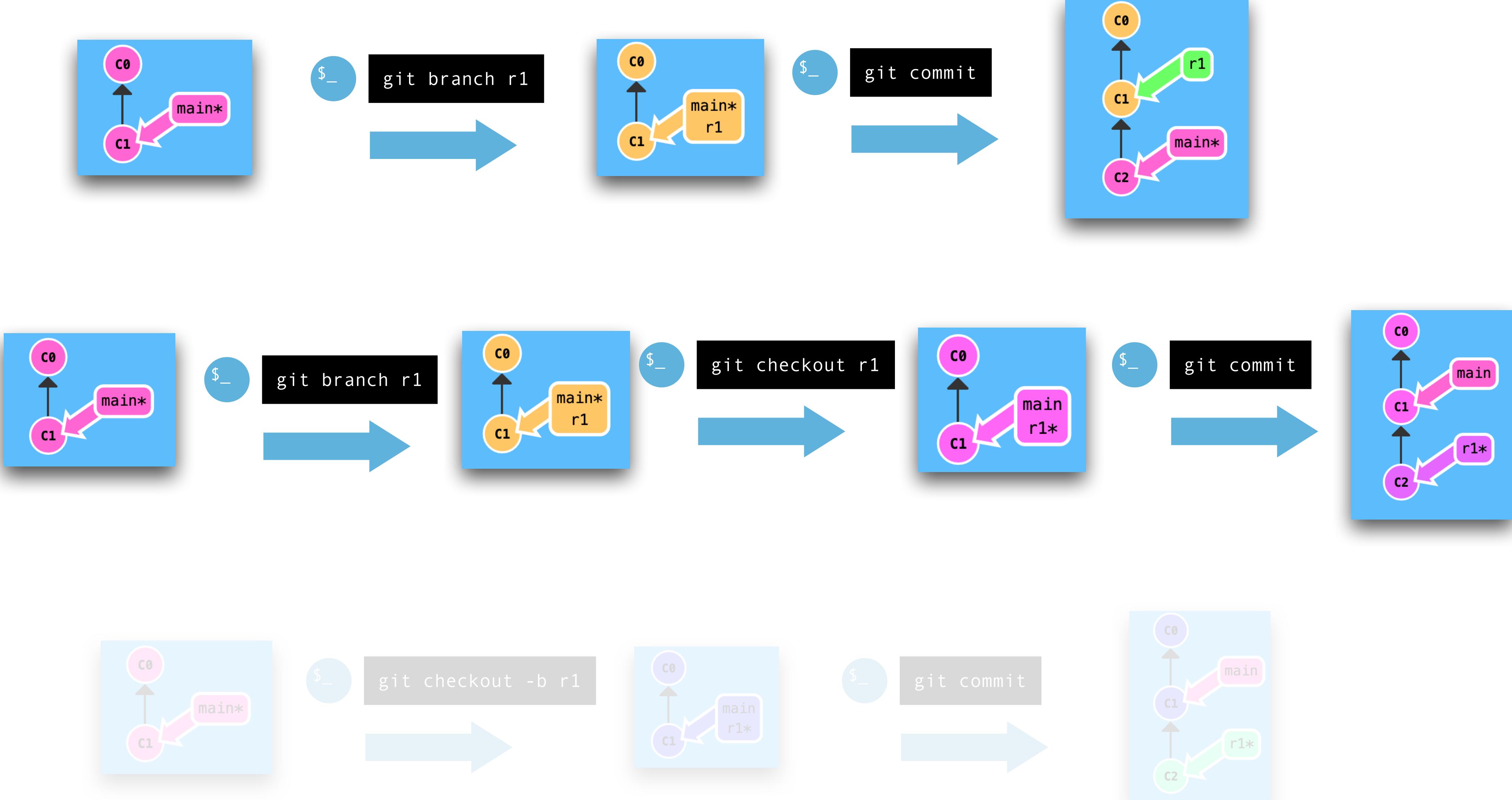
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

\$ \_ git branch r1

\$ \_ git commit

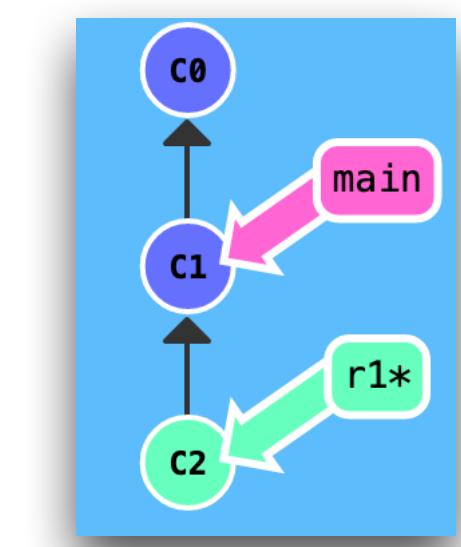
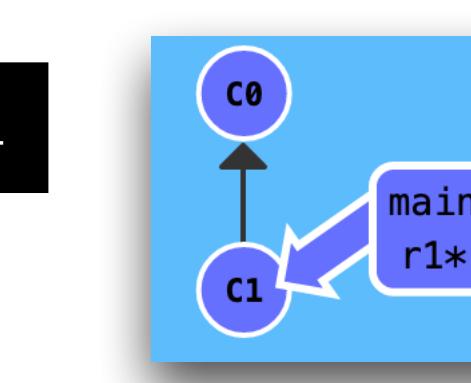
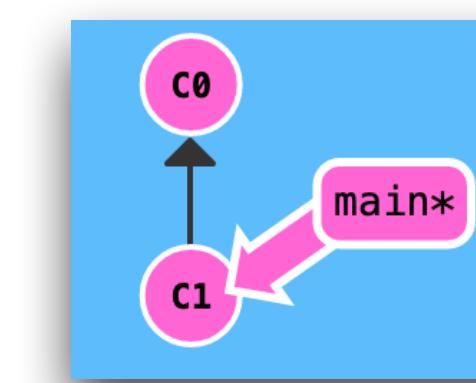
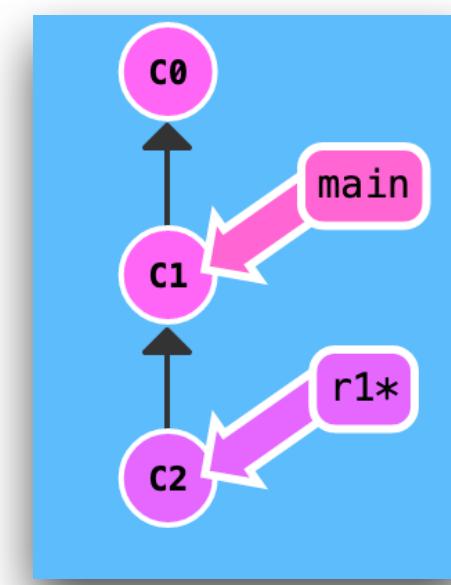
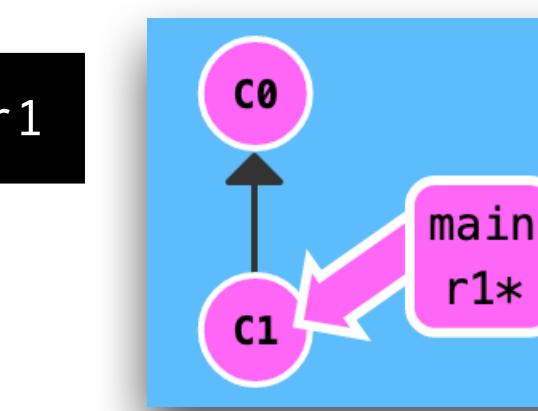
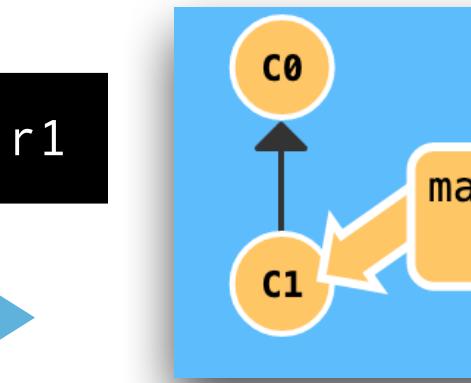
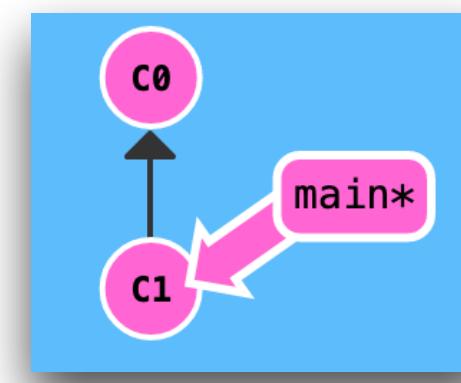
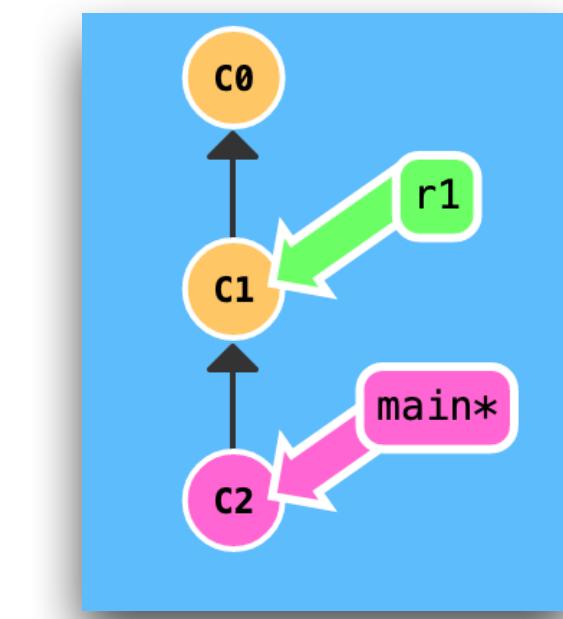
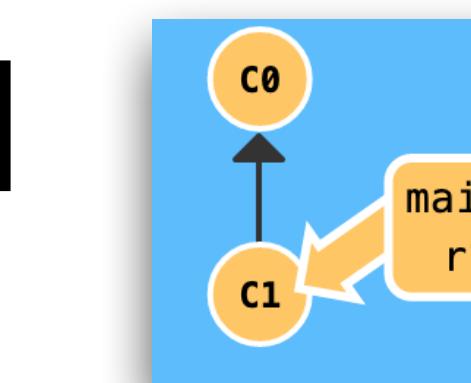
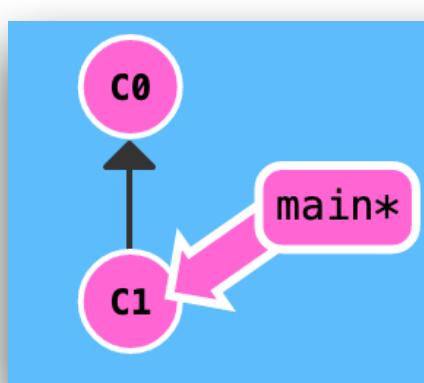
\$ \_ git branch r1

\$ \_ git checkout r1

\$ \_ git commit

\$ \_ git checkout -b r1

\$ \_ git commit





# Creación y cambio de ramas - Ejercicio 5.1

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

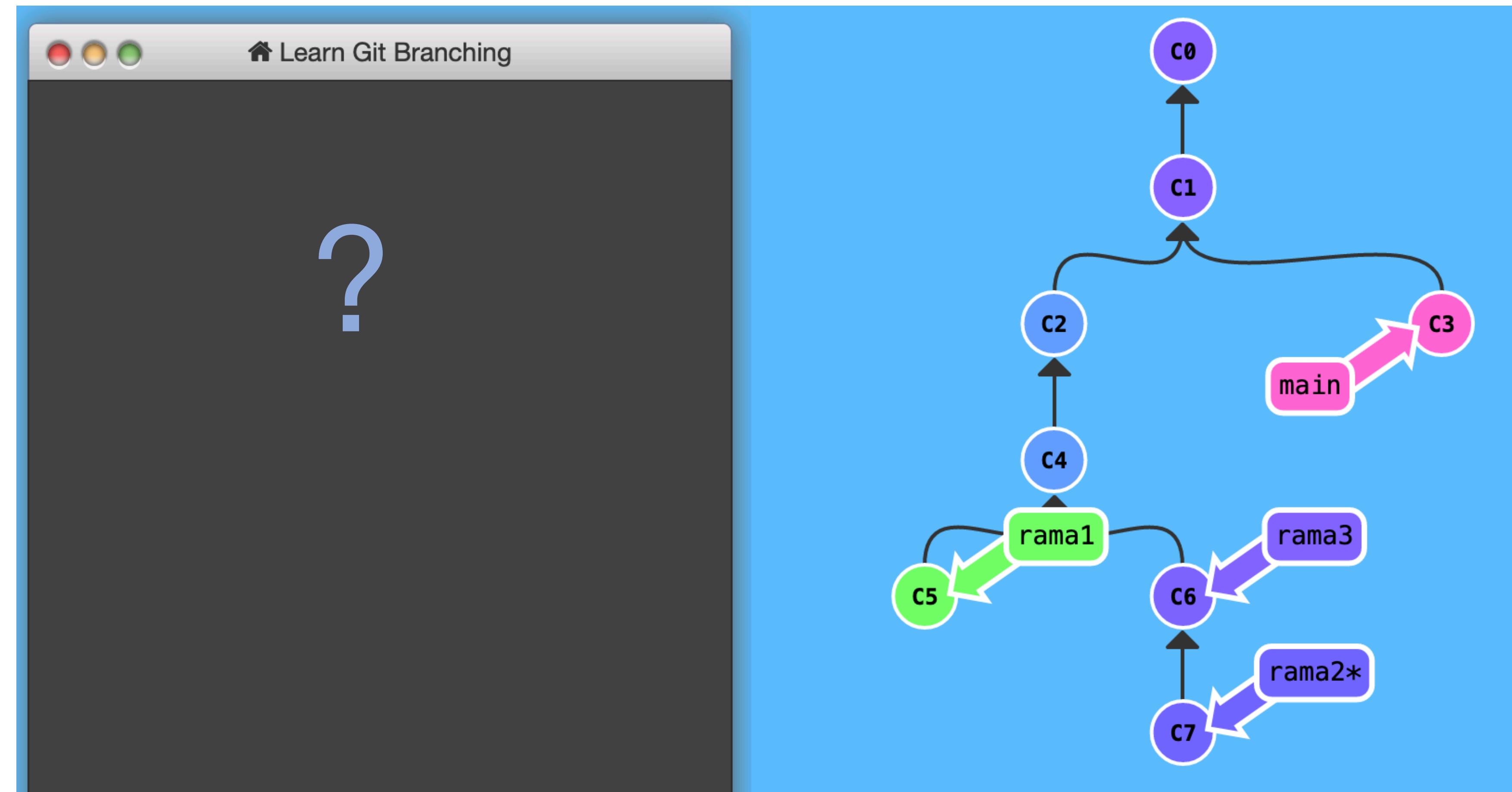
4. Correcciones

5. Ramas

6. Sincronización  
remota



En el simulador, añade los comandos necesarios para crear el grafo que se muestra a continuación.





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

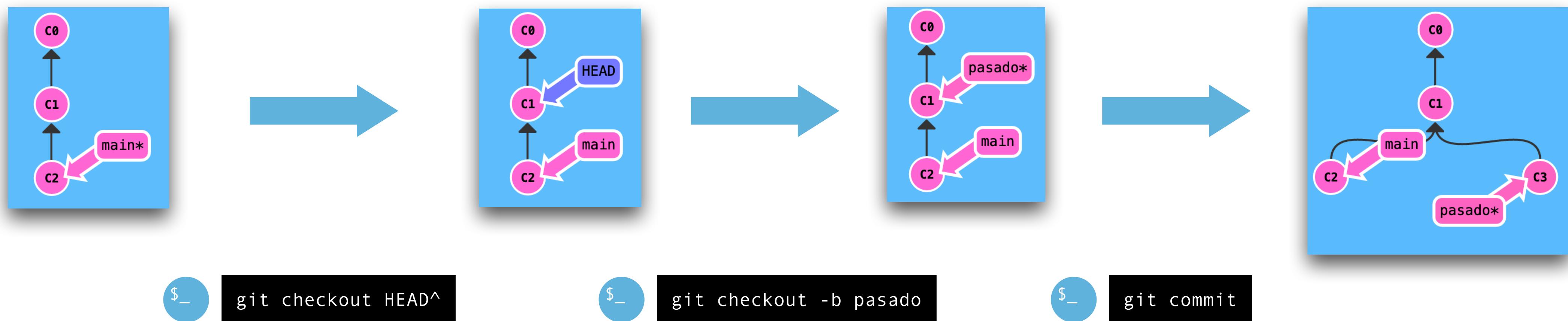
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Una rama se puede crear a partir de cualquier commit (aunque no tenga asociada una rama)
- Por ejemplo, se puede volver a un commit antiguo y hacer la nueva rama a partir este.
- Para eso, hay que mover el HEAD utilizando **git checkout** y después hacer la rama





# Creación y cambio de ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

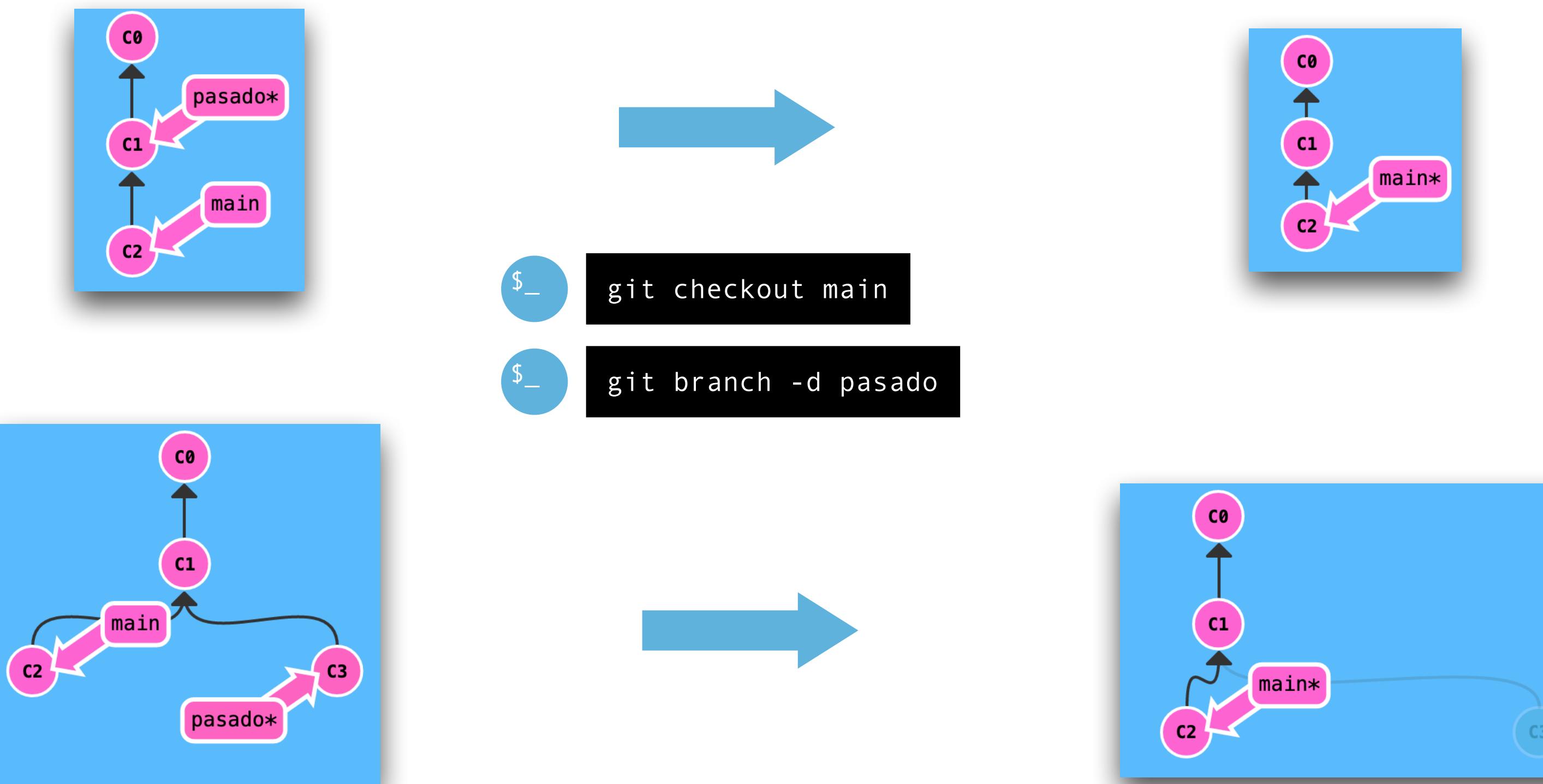
3. Control de cambios en local

4. Correcciones

5. Ramas

6. Sincronización remota

- Se puede borrar una rama añadiendo la opción **-d** al comando **git branch**.
- Cuando se borra una rama, sólo se borra la "etiqueta"
- No se puede borrar la rama activa





# Mezclar ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

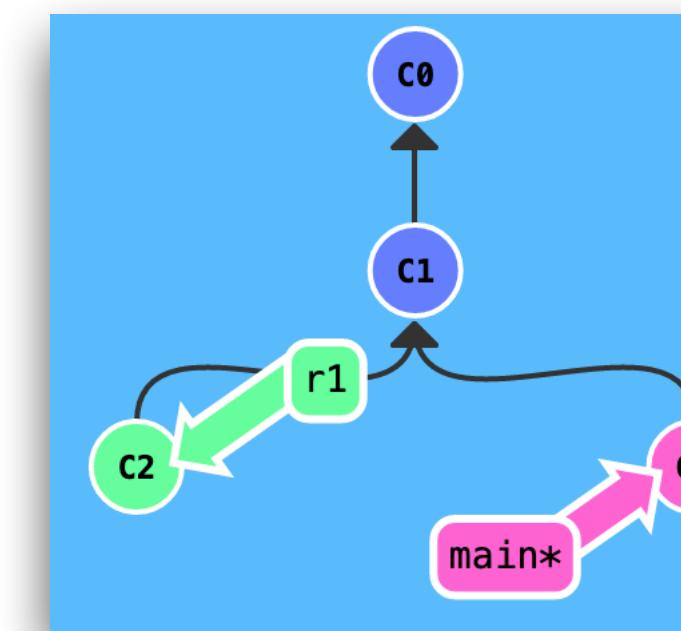
6. Sincronización  
remota

- Cuando se ha realizado el desarrollo de una rama, es necesario juntar las ramas.
- El siguiente comando permite "**traer**" el contenido de otra rama a la activa.

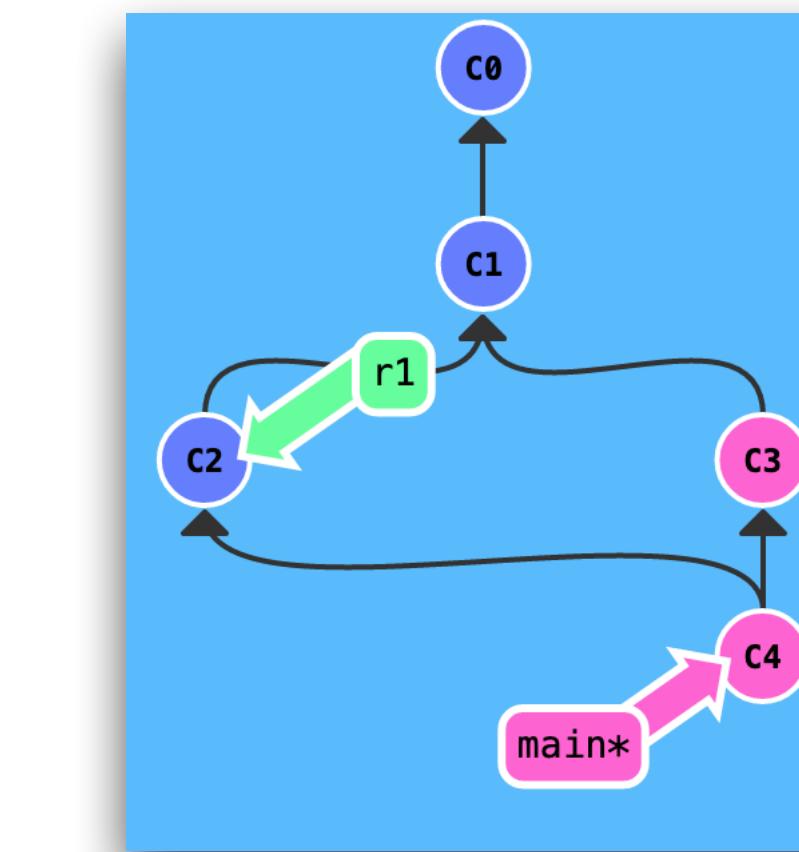


```
git merge [otra_rama]
```

- El commit resultante es la unión de ambas ramas.



→  
\$ \_  
git merge r1





# Mezclar ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

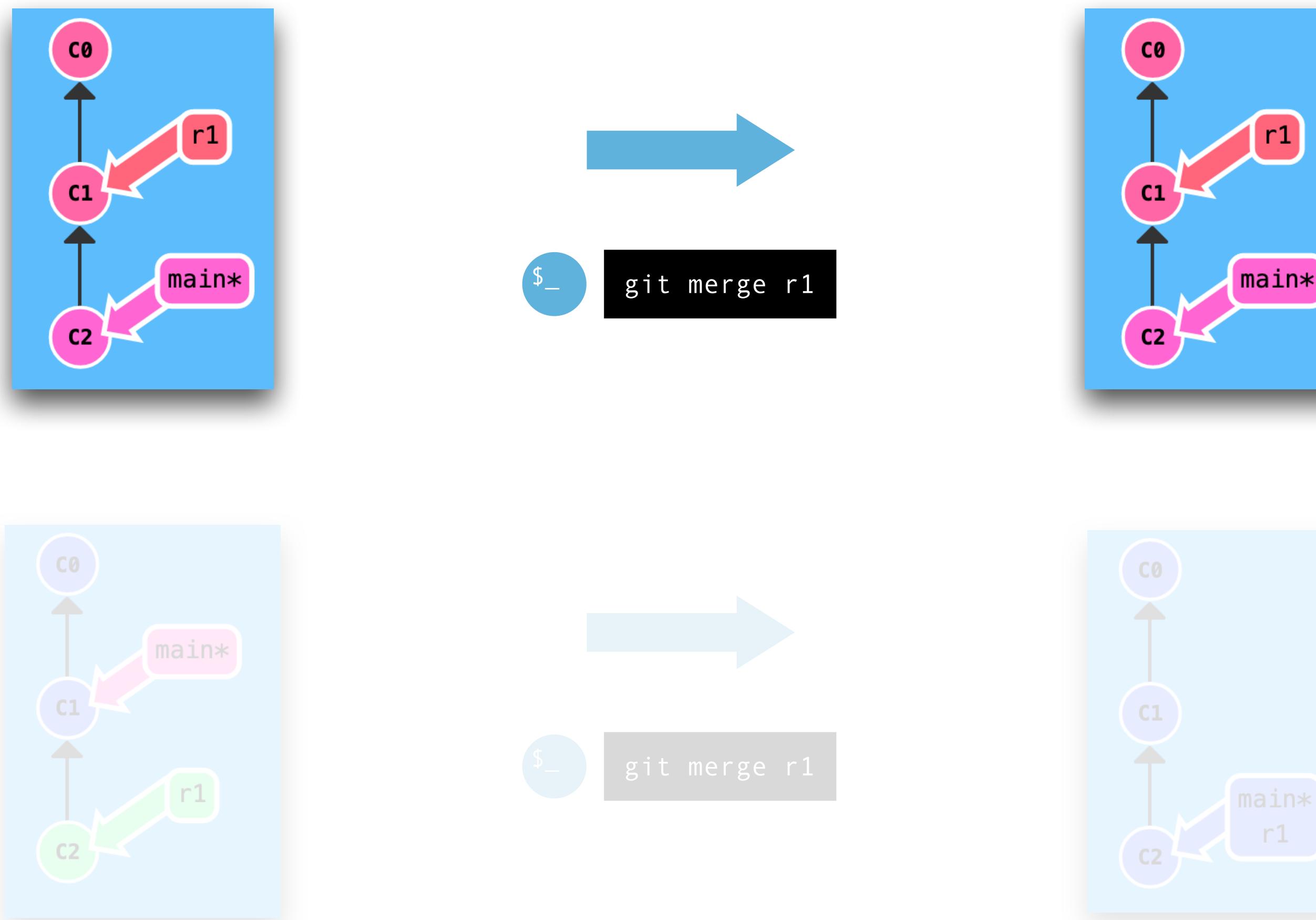
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Cuando las ramas apuntan a commits que en la misma línea de ancestros, no se crean commits nuevos.





# Mezclar ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

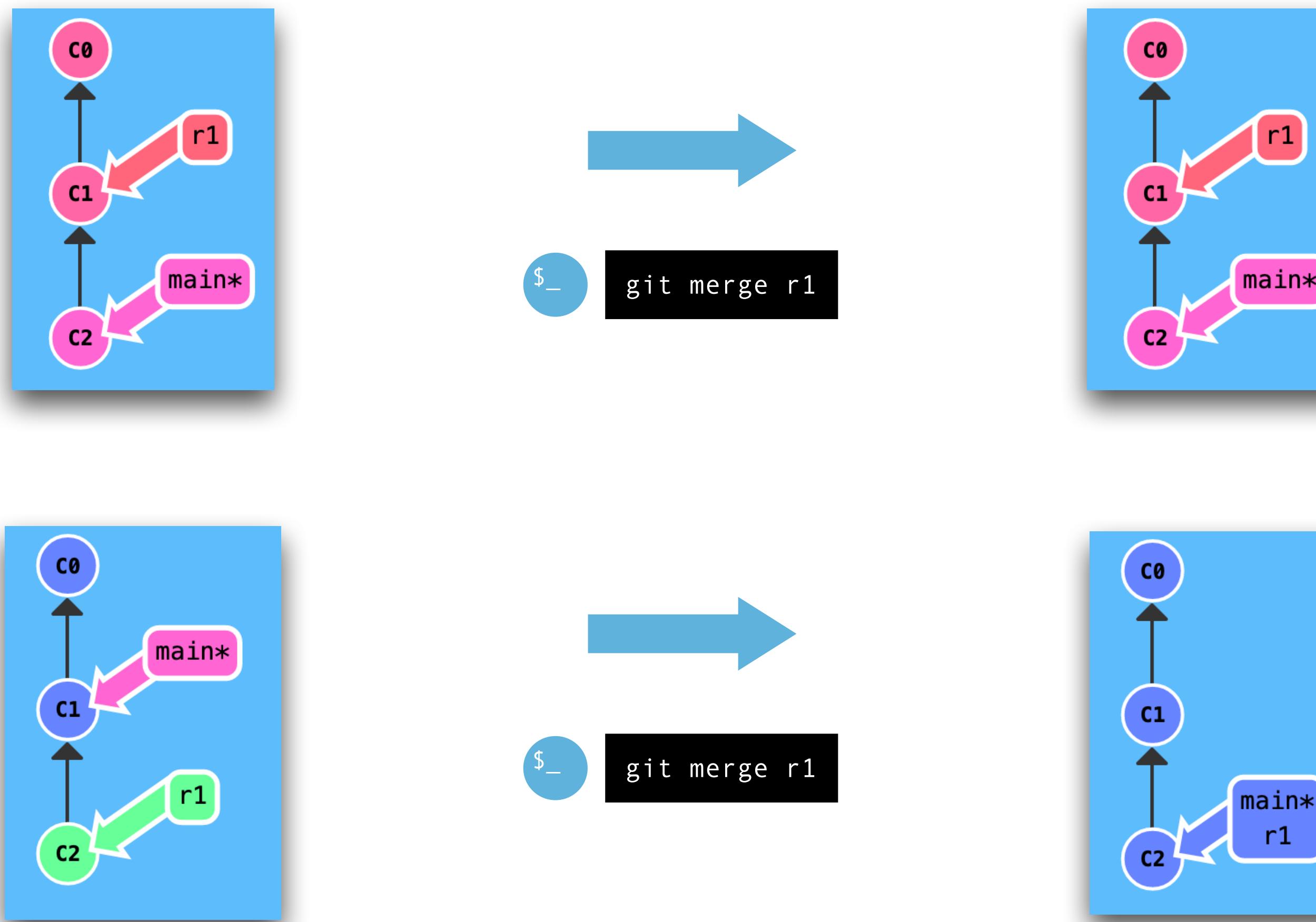
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Cuando las ramas apuntan a commits que en la misma línea de ancestros, no se crean commits nuevos.





# Mezclar ramas - Ejercicio 5.2

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

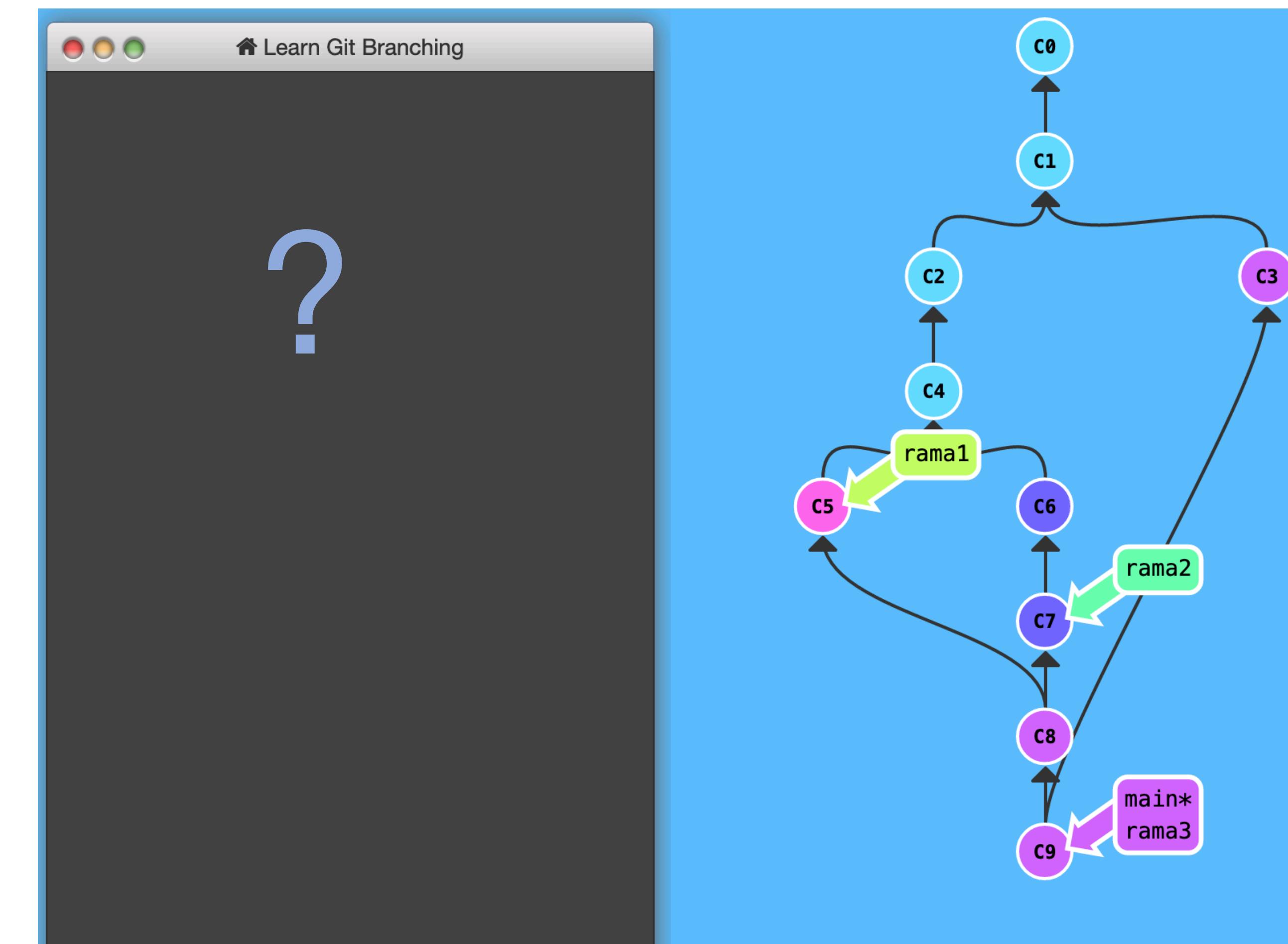
4. Correcciones

5. Ramas

6. Sincronización remota



En el simulador, añade los comandos necesarios para crear el grafo que se muestra a continuación.





# Mezclar ramas

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

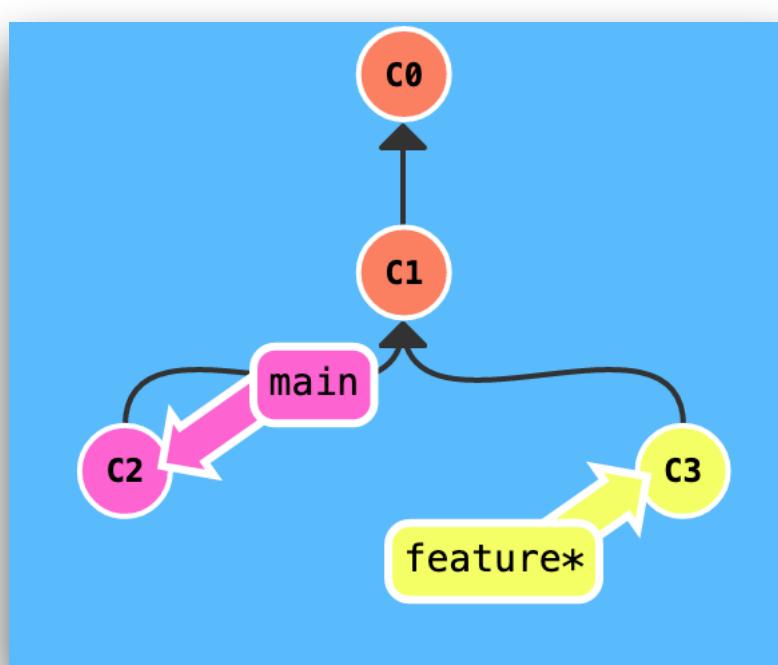
3. Control de  
cambios en local

4. Correcciones

5. Ramas

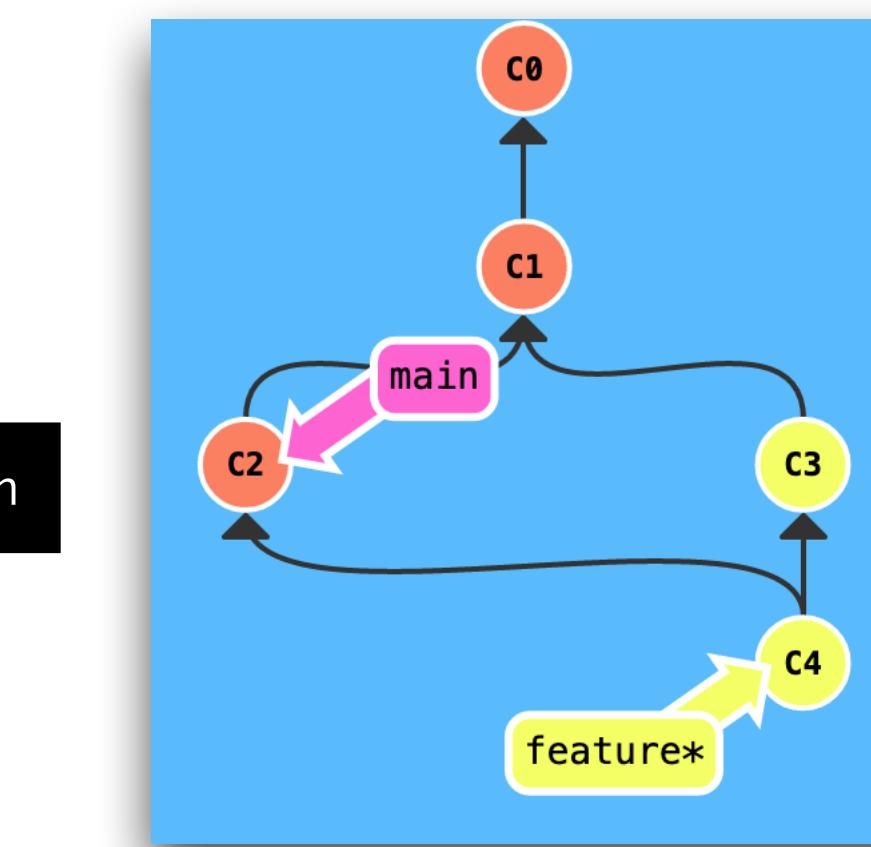
6. Sincronización  
remota

- Al mezclar ramas es recomendable hacerlo primero en la rama "secundaria" y resolver los posibles conflictos en esta:



\$ \_

git merge main

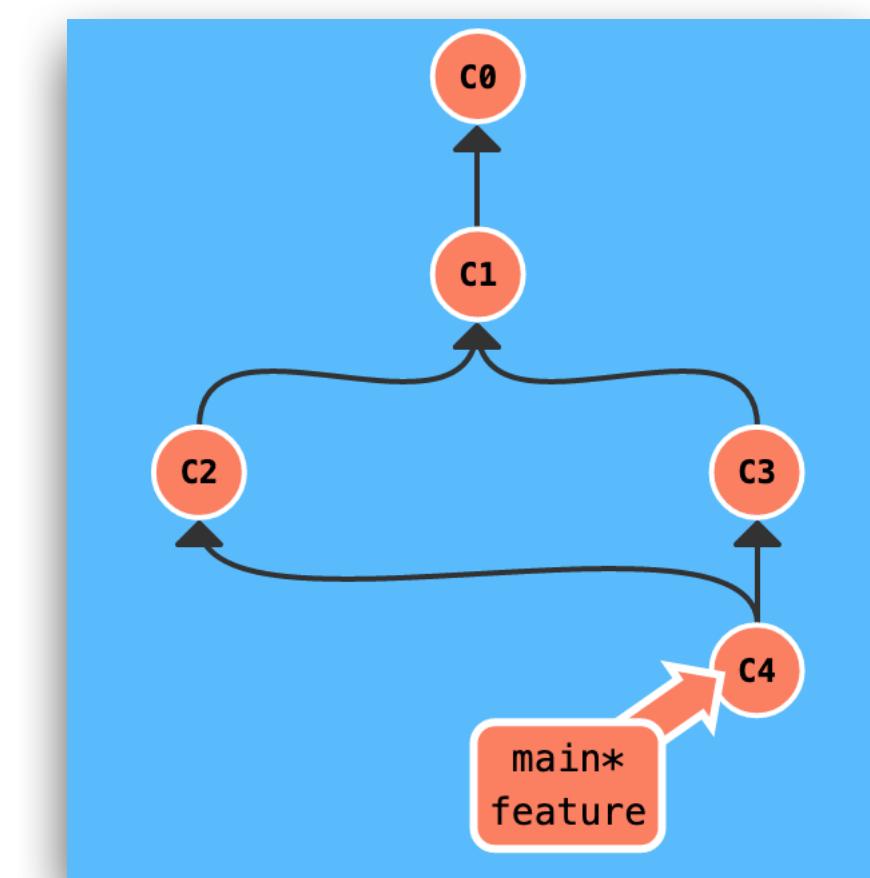


\$ \_

git checkout main

\$ \_

git merge feature





# Información sobre ramas

GitHub

Rafael Cabañas  
rcabanasa@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

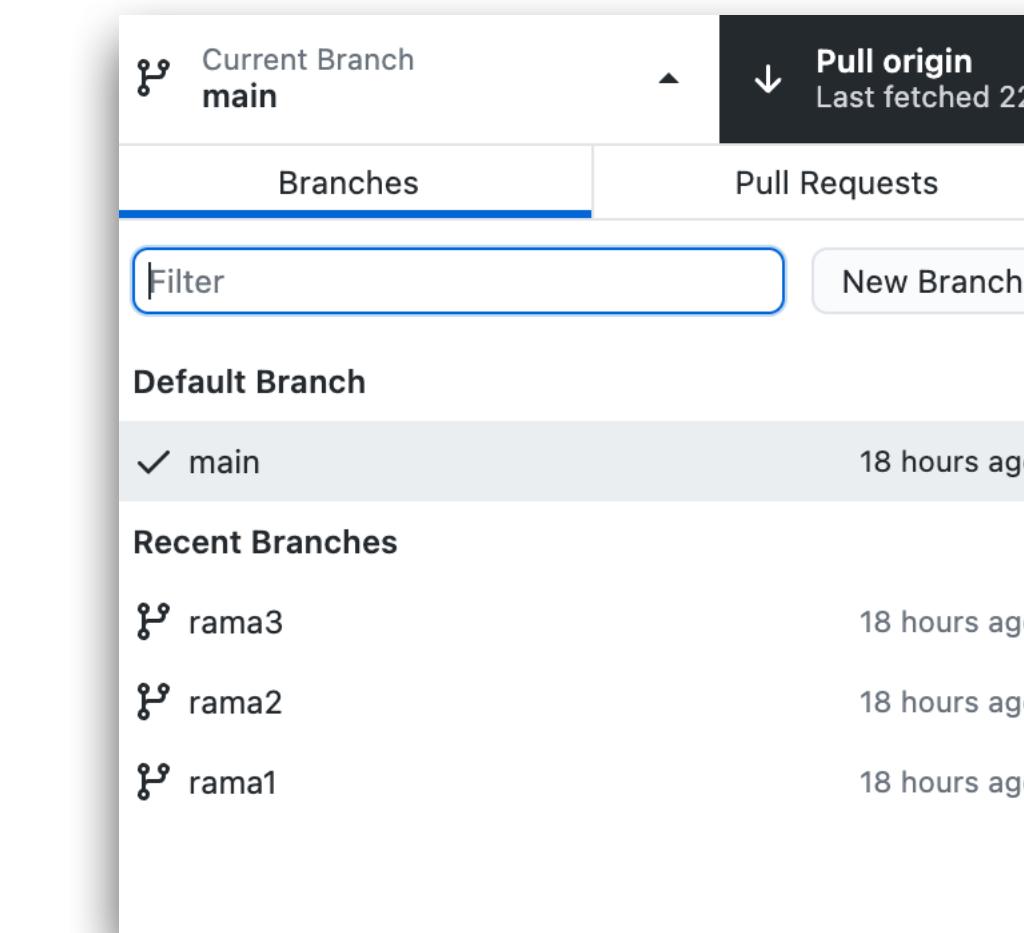
4. Correcciones

5. Ramas

6. Sincronización  
remota

- Podemos listar las ramas y saber en cual está activa

```
$_
1
git branch
* main
  rama1
  rama2
  rama3
```





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

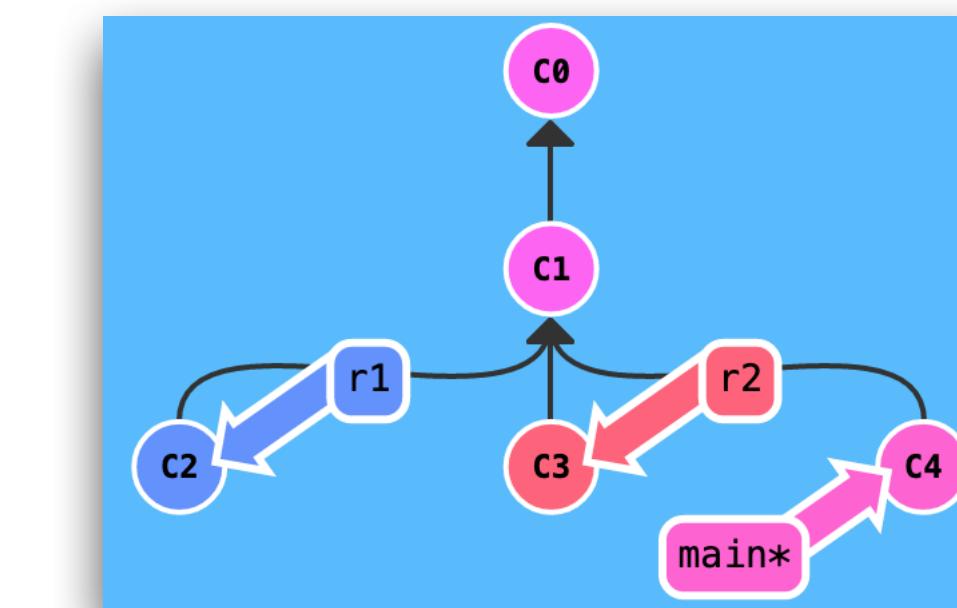
5. Ramas

6. Sincronización  
remota

- Habitualmente, git se encargará de mezclar automáticamente 2 ramas al hacer merge. En otras no sabrá cómo mezclarlas y producirá el siguiente mensaje:

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

- Los conflictos pueden surgir (o no) cuando se mezclan ramas cuyos cambios se han producido en paralelo.
- Git detecta los cambios a nivel de línea, por lo tanto si 2 commits paralelos modifican la misma línea (aunque sean palabras distintas) no sabrá cómo mezclarlos.





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Para ilustrar la resolución de conflictos, vamos a descargar el siguiente repositorio:

\$ \_

```
git clone https://github.com/ualgit/conflictos.git
```

- Contiene sólo un fichero README.md y 3 ramas (main, r1 y r2)

Este es un repositorio creado para ilustrar la operación de `merge` y los conflictos. Para ello se han creado 2 ramas (`r1` y `r2`) a las que se ha añadido un commit. También se ha añadido un commit a la rama `main`. Las únicas diferencias están en los párrafos de la sección siguiente. Más concretamente:

- En `r1` se ha añadido un cambio al primer párrafo.
- En `r2` se ha añadido un cambio al segundo párrafo.
- En `main` se ha añadido un cambio al segundo párrafo.

Por lo tanto, al mezclar `r1` y `r2` no habrá conflictos, pero sí al mezclar `r2` con `main` (o con una que contenga sus cambios).

Nota: aunque los párrafos se muestren formateados en varias líneas, en realidad cada párrafo es una sola línea



# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

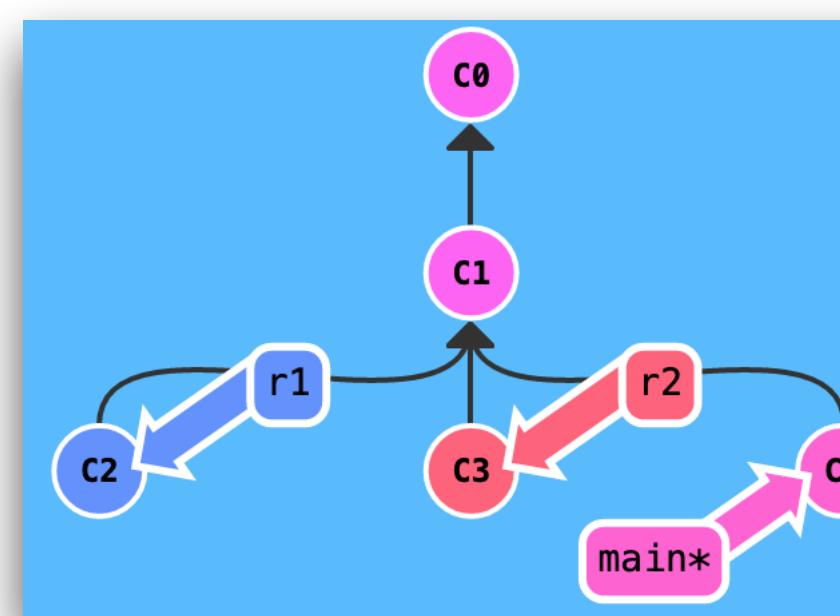
6. Sincronización  
remota

- Para ilustrar la resolución de conflictos, vamos a descargar el siguiente repositorio:

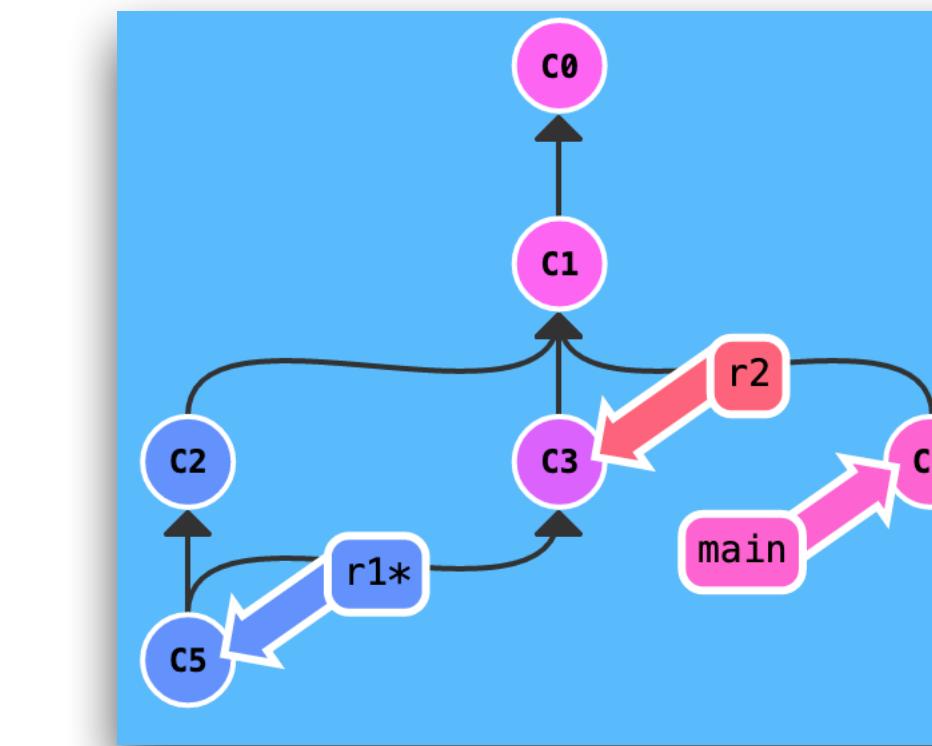
\$ \_

```
git clone https://github.com/ualgit/conflictos.git
```

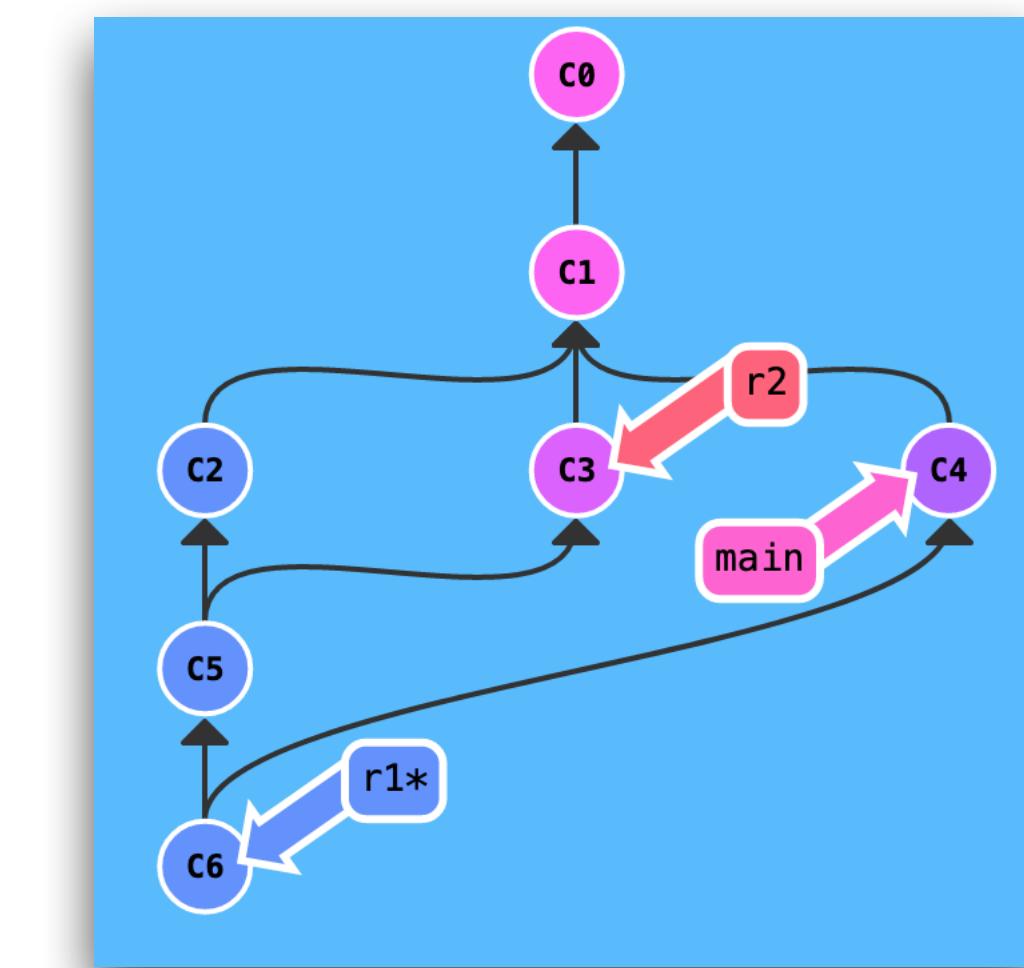
- Contiene sólo un fichero README.md y 3 ramas (main, r1 y r2)

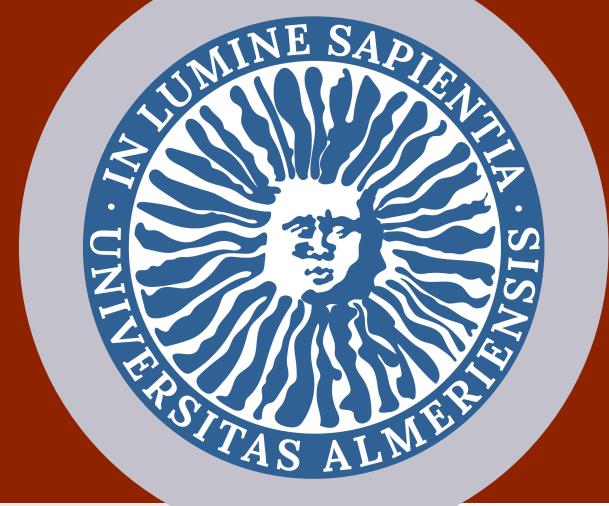


merge  
sin conflictos



merge  
con conflictos





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## Texto específico de cada rama

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Sollicitudin aliquam ultrices sagittis orci a scelerisque purus semper eget. Aliquet bibendum enim facilisis gravida neque convallis. Viverra aliquet eget sit amet tellus cras adipiscing enim.

Metus aliquam eleifend mi in nulla posuere. Tortor id aliquet lectus proin nibh nisl. Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod nisi quis eleifend quam adipiscing.

## Texto específico de cada rama

Lorem ipsum **CAMBIOS EN R1** dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Sollicitudin aliquam ultrices sagittis orci a scelerisque purus semper eget. Aliquet bibendum enim facilisis gravida neque convallis. Viverra aliquet eget sit amet tellus cras adipiscing enim.

Metus aliquam eleifend mi in nulla posuere. Tortor id aliquet lectus proin nibh nisl. Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod nisi quis eleifend quam adipiscing.

## Texto específico de cada rama

Metus aliquam **CAMBIOS EN R2** eleifend mi in nulla posuere. Tortor id aliquet lectus proin nibh nisl. Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod nisi quis eleifend quam adipiscing.

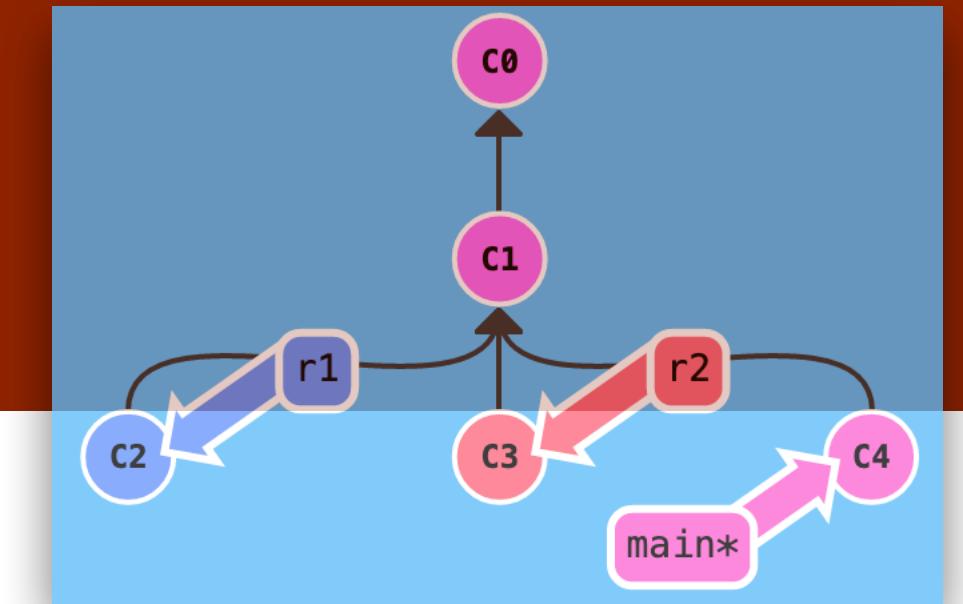
## Texto específico de cada rama

Metus aliquam eleifend mi in nulla posuere. Tortor id aliquet **CAMBIOS EN MAIN** lectus proin nibh nisl. Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod nisi quis eleifend quam adipiscing.

Metus aliquam eleifend mi in nulla posuere. Tortor id aliquet **CAMBIOS EN MAIN** lectus proin nibh nisl. Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod nisi quis eleifend quam adipiscing.

merge  
sin conflictos

merge  
con conflictos





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

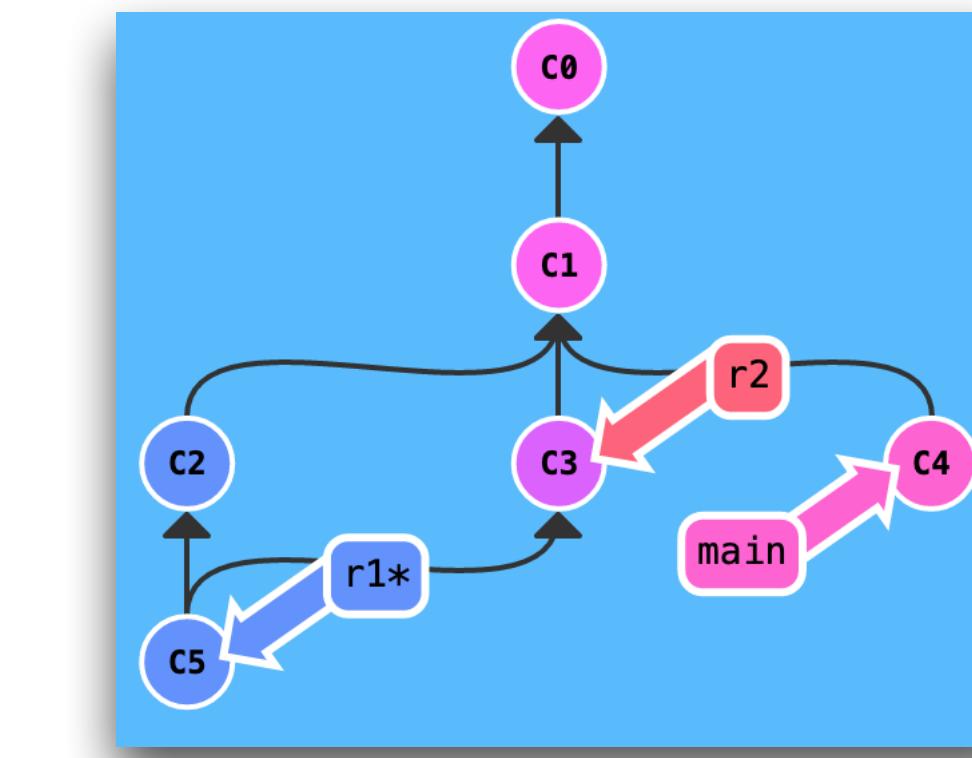
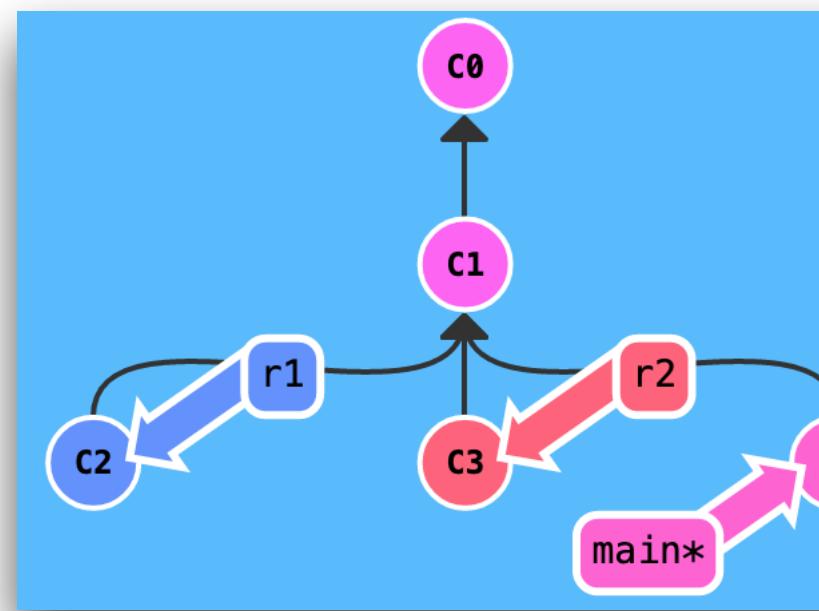
5. Ramas

6. Sincronización  
remota

\$ \_

```
1 git checkout r1
2 git merge r2 -m "sin conflictos"
.
.
.
Auto-merging README.md
Merge made by the 'recursive' strategy.

 README.md | 2 ++
1 file changed, 1 insertion(+), 1 deletion(-)
```





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

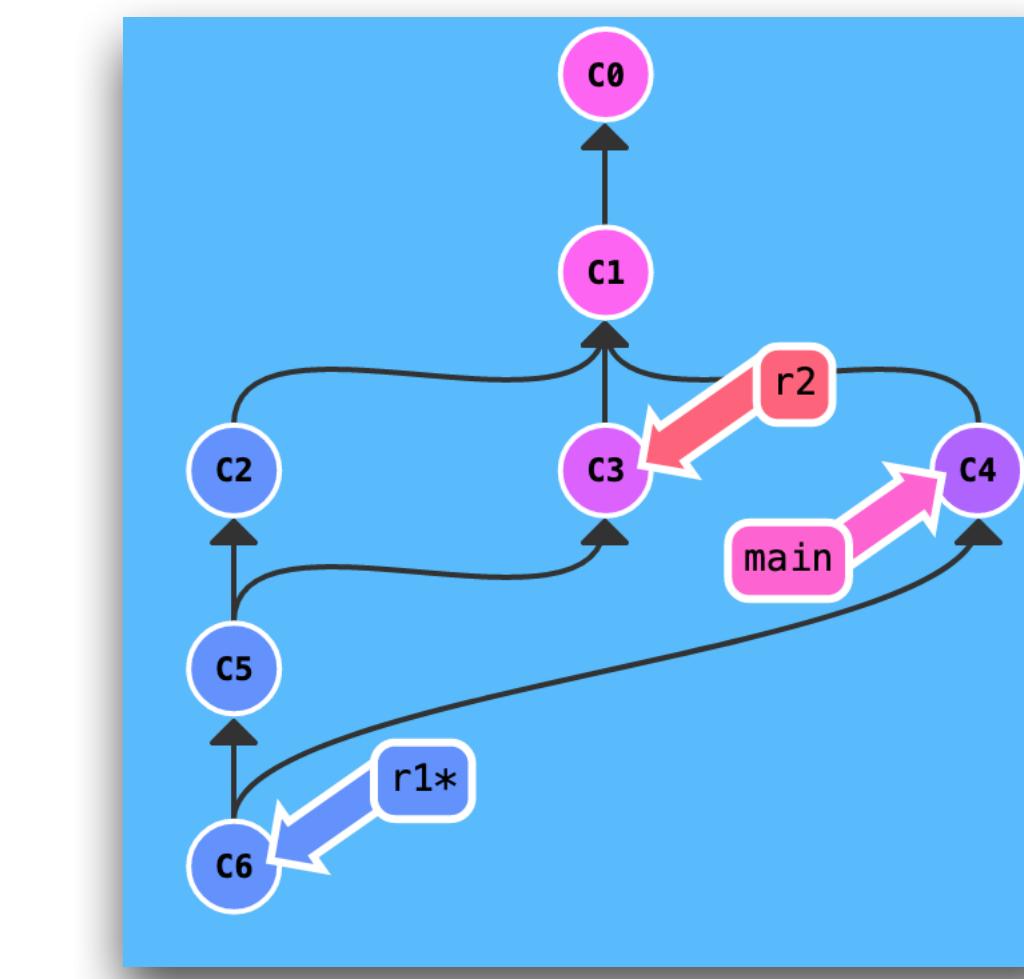
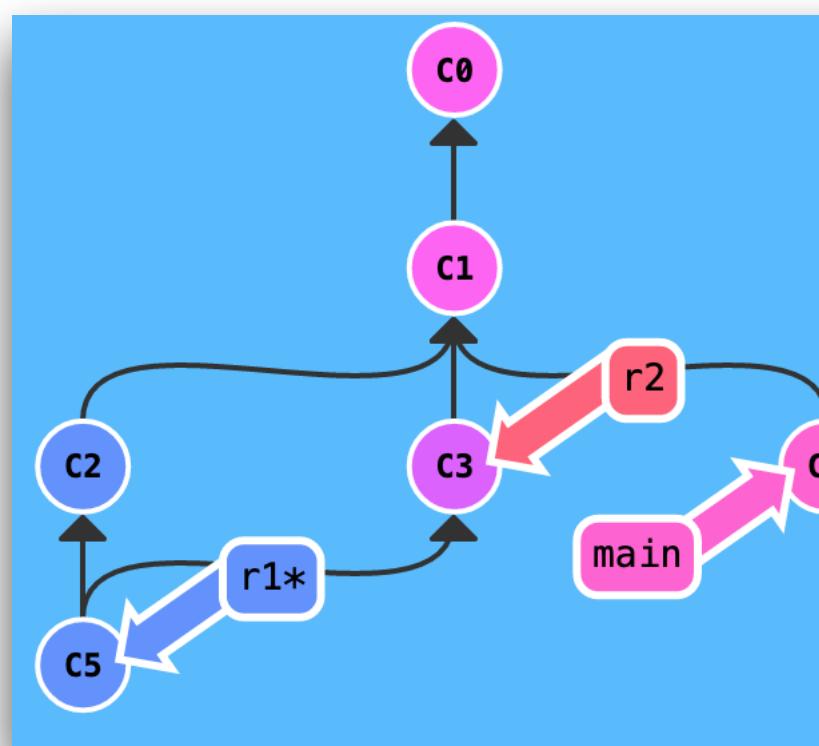
4. Correcciones

5. Ramas

6. Sincronización  
remota

3  
. . .

```
git merge main -m "conflictivo"  
Auto-merging README.md  
CONFLICT (content): Merge conflict in README.md  
Automatic merge failed; fix conflicts and then commit the result.
```





# Conflictos

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

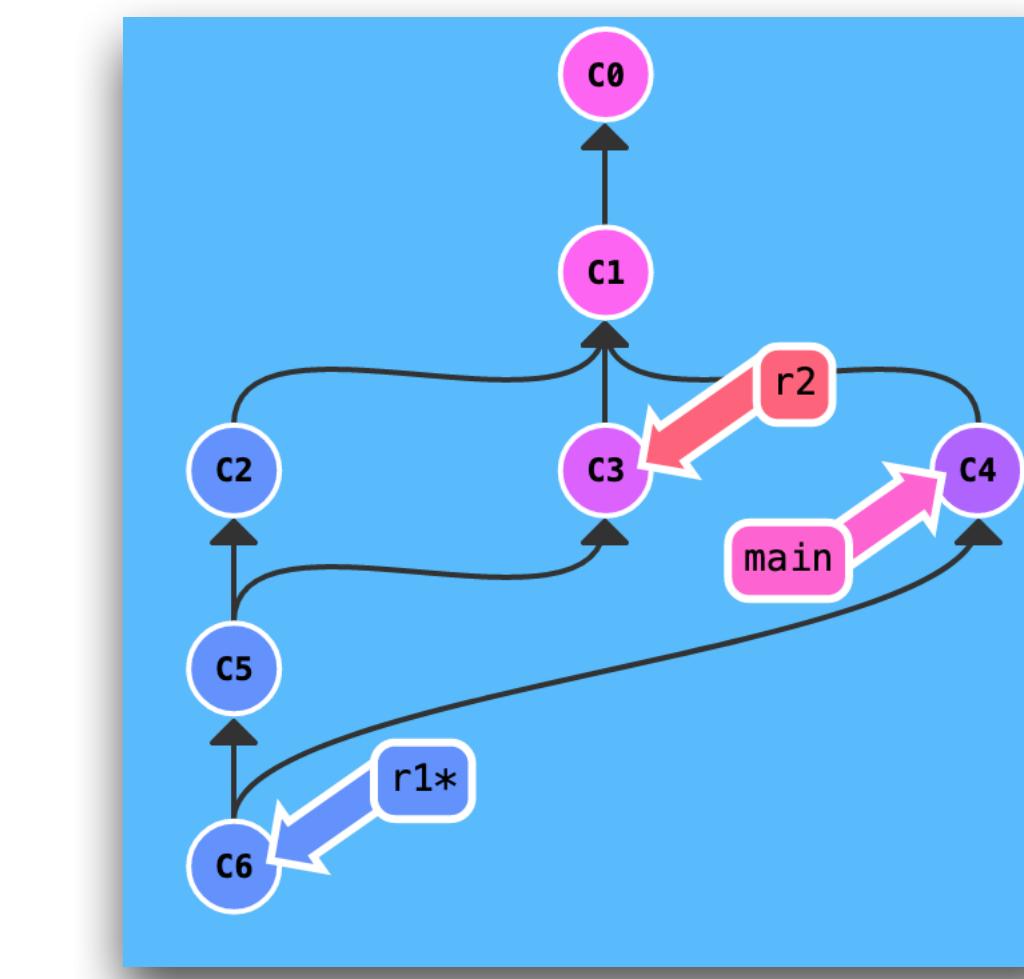
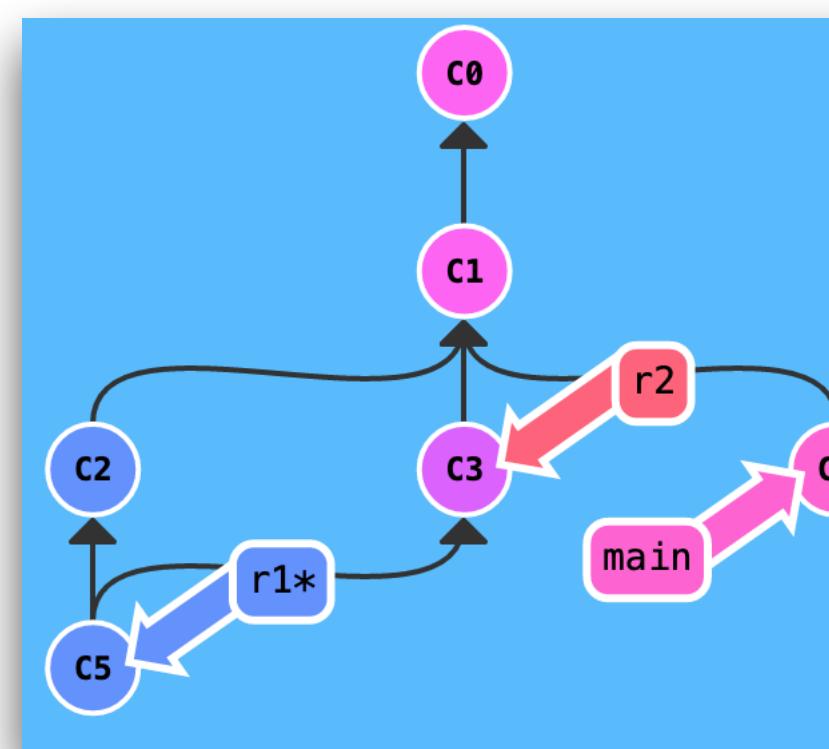
4. Correcciones

5. Ramas

6. Sincronización  
remota

3  
. . .

```
git merge main -m "conflictivo"  
Auto-merging README.md  
CONFLICT (content): Merge conflict in README.md  
Automatic merge failed; fix conflicts and then commit the result.
```





# Conflictos

# GitHub

**Rafael Cabañas**  
rcabanas@ual.es

## 1. Introducción

## 2. Configuración

## 3. Control de cambios en local

## 4. Correcciones

5. Ramas

## 6. Sincronización remota

- Finalmente podemos ver en qué ficheros hay conflictos y editarlos manualmente.

```
4     git diff --check  
.  
.  
.  
 README.md:19: leftover conflict mark  
 README.md:21: leftover conflict mark  
 README.md:23: leftover conflict mark
```

1

● ● ● README.md

Markdown | ⓘ

^ ▼ Metus aliquam \*\*CAMBIOS EN R2\*\* eleifend mi in...mod elementum nisi quis eleifend quam adipiscing. ☊

- línea ya que sólo contienen un salto de línea al final (se ha pulsado la tecla intro).

13

14

15 ## Texto específico de cada rama

16

17 Lorem ipsum \*\*CAMBIOS EN R1\*\* dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
- incididunt ut labore et dolore magna aliqua. Sollicitudin aliquam ultrices sagittis orci a scelerisque purus  
- semper eget. Aliquet bibendum enim facilisis gravida neque convallis. Viverra aliquet eget sit amet tellus cras  
- adipiscing enim.

18

19 <<<<< HEAD

20 Metus aliquam \*\*CAMBIOS EN R2\*\* eleifend mi in nulla posuere. Tortor id aliquet lectus proin nibh nisl. Hac  
- habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla pellentesque  
- dignissim. Ultrices vitae auctor eu augue. Aenean euismod elementum nisi quis eleifend quam adipiscing.

21 =====

22 Metus aliquam eleifend mi in nulla posuere. Tortor id aliquet \*\*CAMBIOS EN MAIN\*\* lectus proin nibh nisl.  
- Hac habitasse platea dictumst quisque sagittis purus sit. Gravida dictum fusce ut placerat orci nulla  
- pellentesque dignissim. Ultrices vitae auctor eu augue. Aenean euismod elementum nisi quis eleifend quam  
- adipiscing.

23 >>>>> main

24

25

Lines: 25 Characters: 1,864 Location: 1,518 Line: 21 1.89 KB | Unicode (UTF-8) ☊ | LF ☊

1

```
git merge -s ours feature
```



# Conflictos - Ejercicio 5.3

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



Realiza un fork a tu cuenta del repositorio <https://github.com/ualgit/experimentos> , crea una copia local y realiza las siguientes tareas.

- A. Crea una rama *feature* para implementar la función media.
- B. Tanto en la rama *feature* como en la *main*, modifica el parámetro sigma y confirma los cambios (haz commit).
- C. Mezcla todo el contenido en la rama *main*.



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## 6. Sincronización remota



# Motivación

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de cambios en local

4. Correcciones

5. Ramas

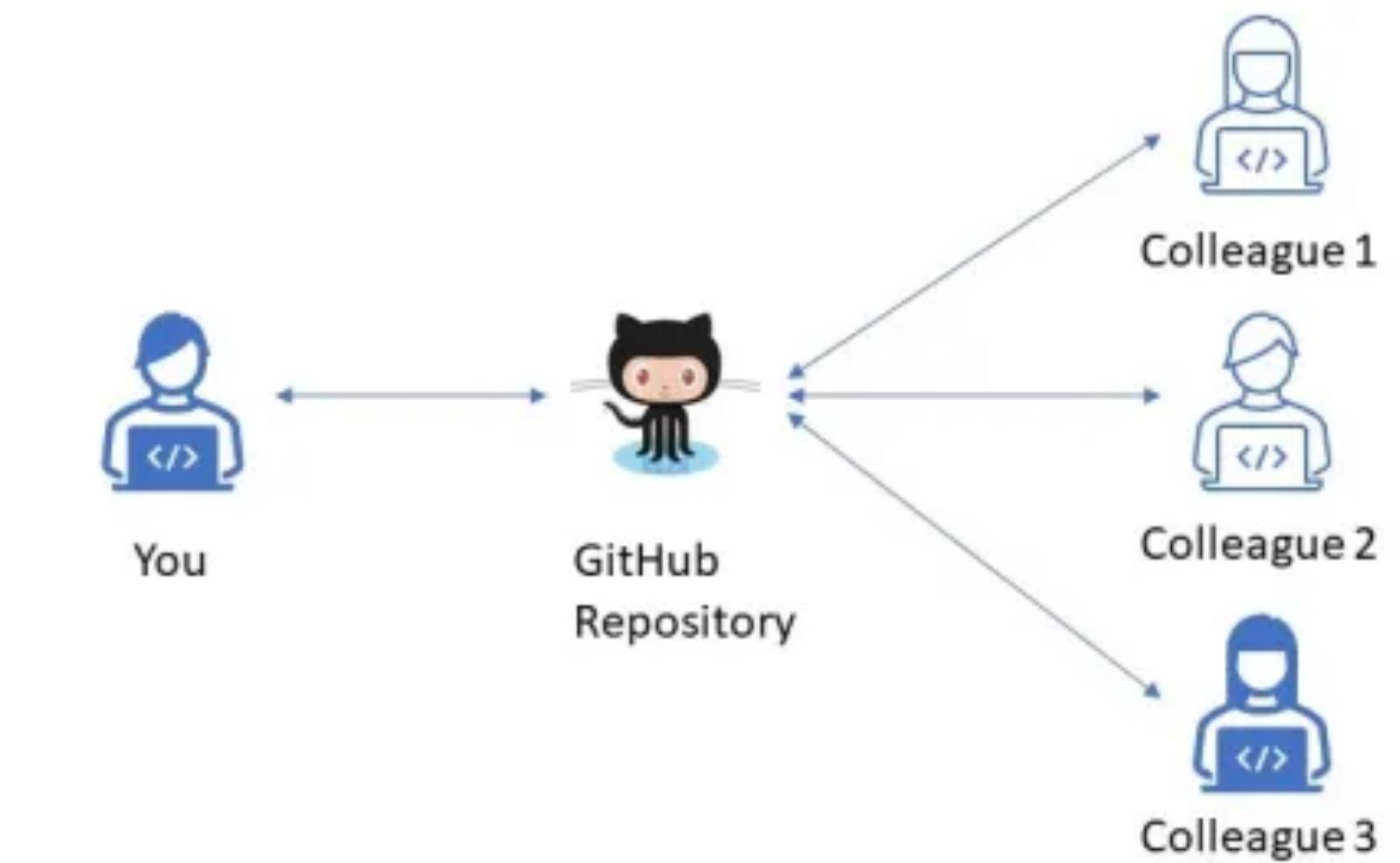
6. Sincronización remota

## Idea

- El repositorio se sincroniza con otra copia remota en un servidor.
- De esta manera se pueden gestionar varias copias locales

## Posibles escenarios

- Desarrollo de software científico
  - Varias personas participan
  - Ejecutar experimentos en un servidor remoto
  - Reproducir resultados de publicaciones
- Repositorio con material docente
  - Para que puedan acceder los alumnos
  - Sistema de copia de seguridad





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

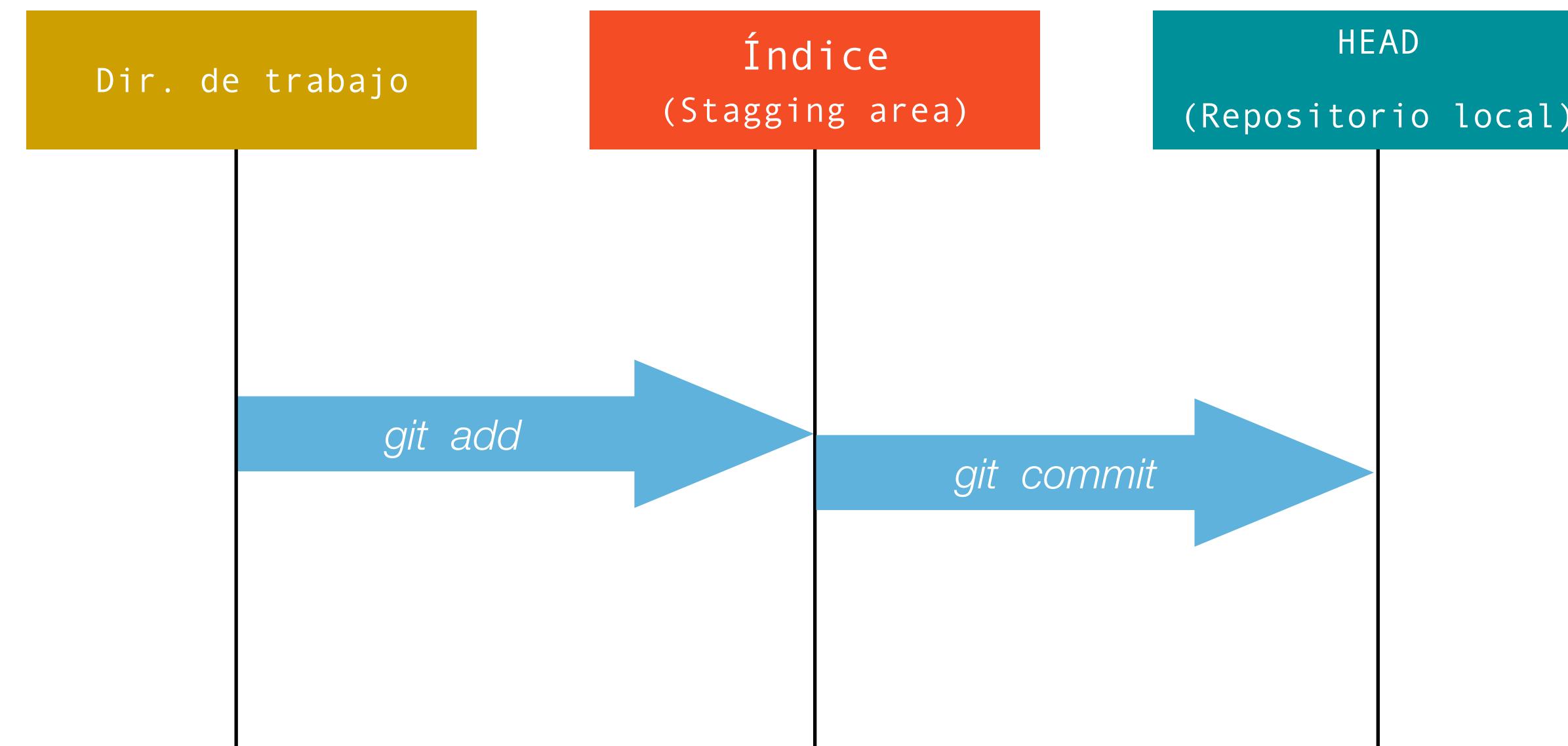
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

Dir. de trabajo

Índice  
(Staging area)

HEAD  
(Repositorio local)

Remoto

`git add`

`git commit`



# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota

Dir. de trabajo

Índice  
(Staging area)

HEAD  
(Repositorio local)

Remoto

`git add`

`git commit`



Ordenador local



GitHub

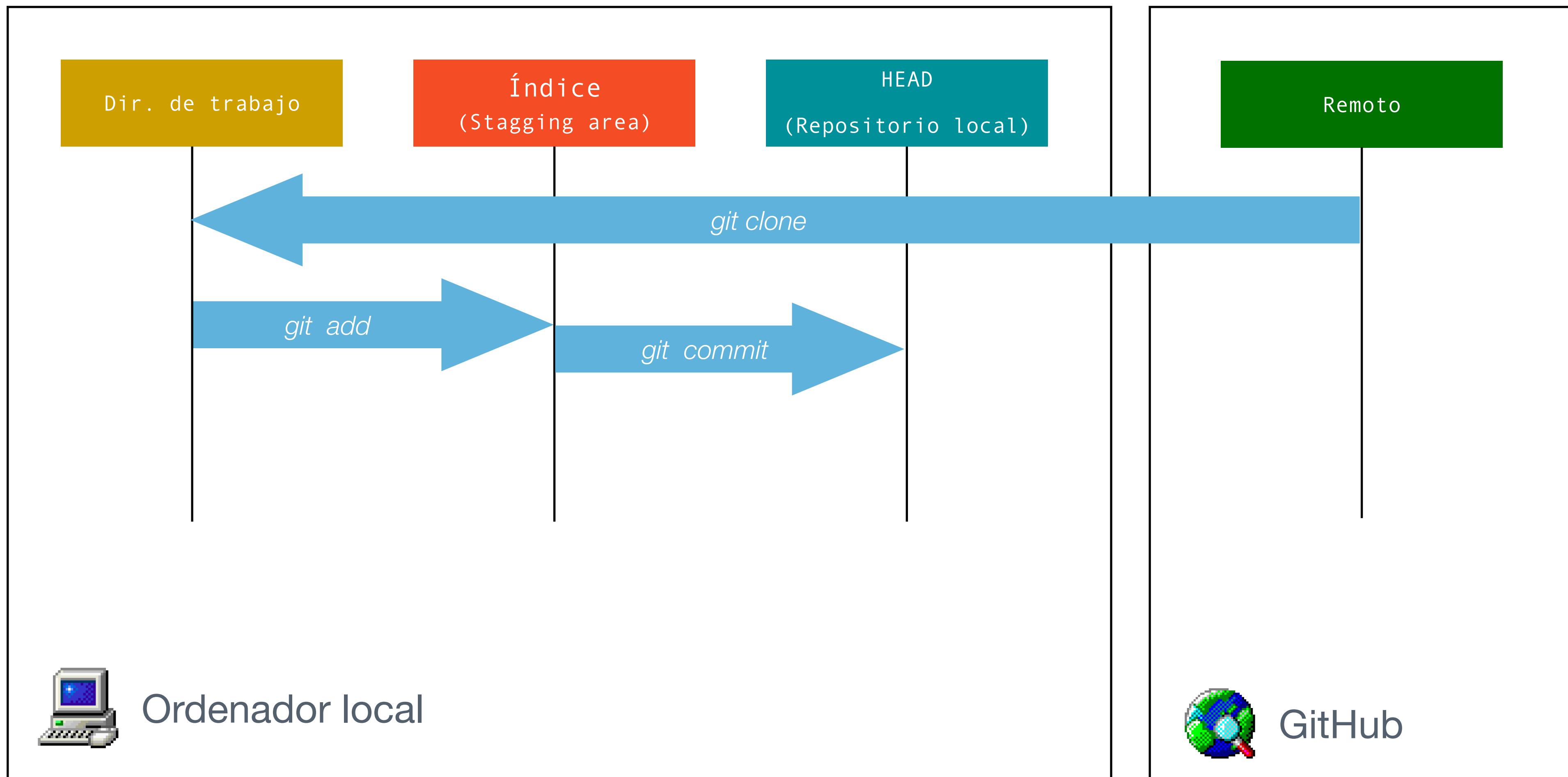


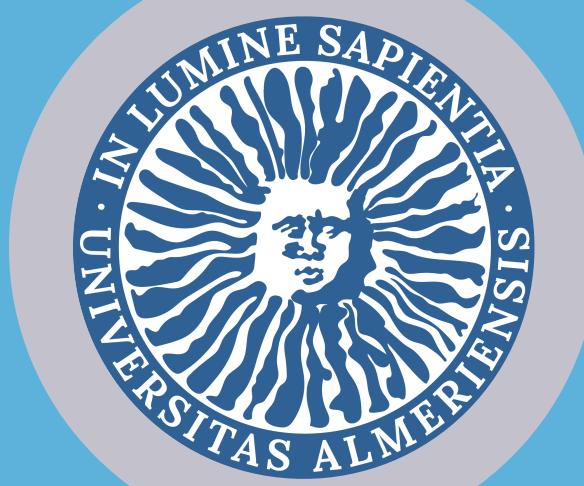
# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción
2. Configuración
3. Control de cambios en local
4. Correcciones
5. Ramas
6. Sincronización remota





# Flujo de trabajo

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

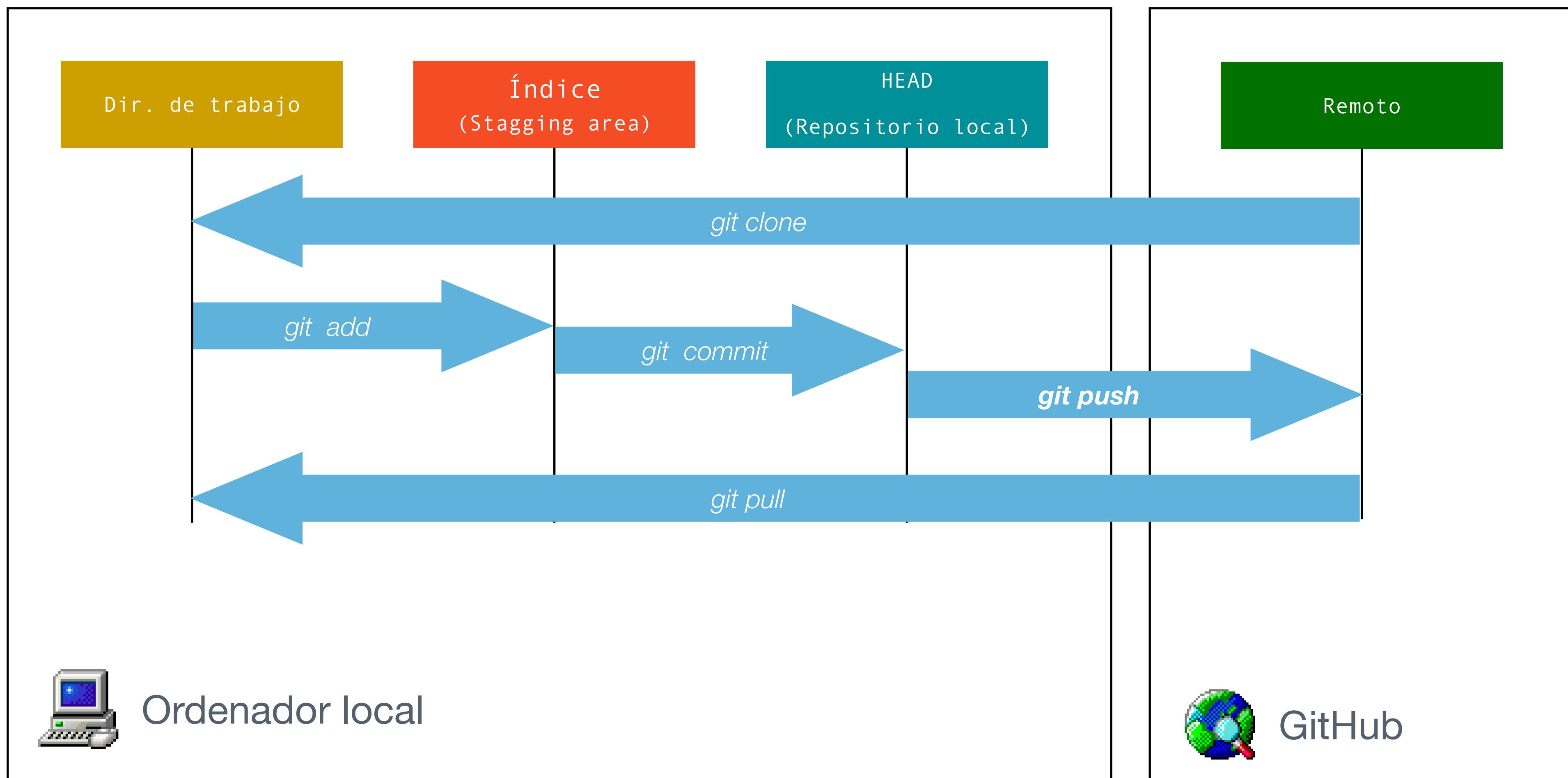
2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El comando **git push** permite enviar los commits locales al servidor remoto.

\$ \_

git push

\$ \_

git push origin main

- Es aconsejable que no haya cambios por confirmar.

- Por defecto sólo se suben los cambios de la rama activa. Para subir todas las ramas

\$ \_

git push --all origin

- La primera vez que se sube una rama hay que especificar

\$ \_

git push --set-upstream origin [nombre\_rama]

- También es necesario subir las etiquetas (por defecto push no las sube):

\$ \_

git push origin v1.1

\$ \_

git push origin --tags



# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El comando **git push** permite enviar los commits locales al servidor remoto.

\$ \_

git push

\$ \_

git push origin main

- Es aconsejable que no haya cambios por confirmar.
- Por defecto sólo se suben los cambios de la rama activa. Para subir todas las ramas

\$ \_

git push --all origin

- La primera vez que se sube una rama hay que especificar

\$ \_

git push --set-upstream origin [nombre\_rama]

- También es necesario subir las etiquetas (por defecto push no las sube):

\$ \_

git push origin v1.1

\$ \_

git push origin --tags



# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- El comando **git push** permite enviar los commits locales al servidor remoto.

\$ \_

git push

\$ \_

git push origin main

- Es aconsejable que no haya cambios por confirmar.

- Por defecto sólo se suben los cambios de la rama activa. Para subir todas las ramas

\$ \_

git push --all origin

- La primera vez que se sube una rama hay que especificar

\$ \_

git push --set-upstream origin [nombre\_rama]

- También es necesario subir las etiquetas (por defecto push no las sube):

\$ \_

git push origin v1.1

\$ \_

git push origin --tags



# Subir cambios

## GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

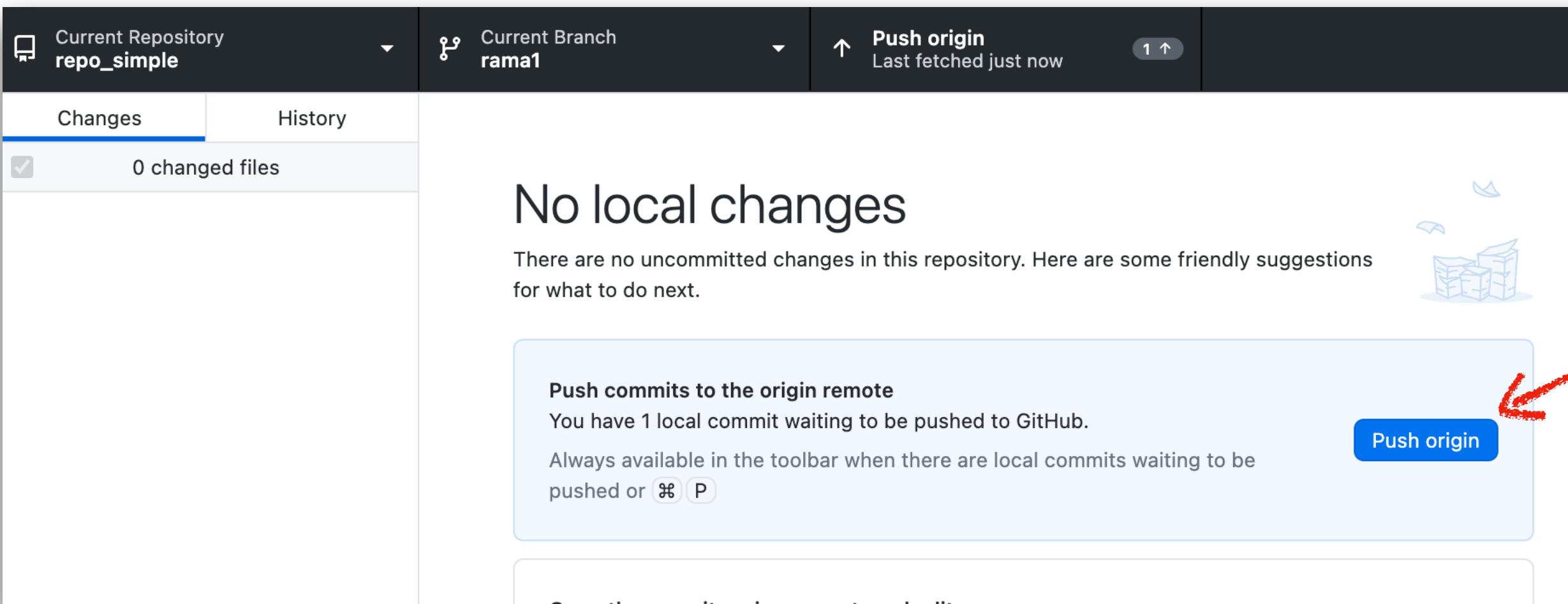
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- En GitHub Desktop sólo hay que pulsar el botón **push origin**





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

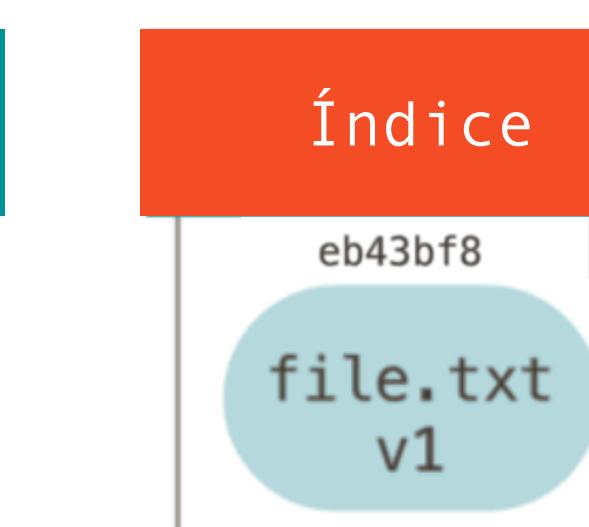
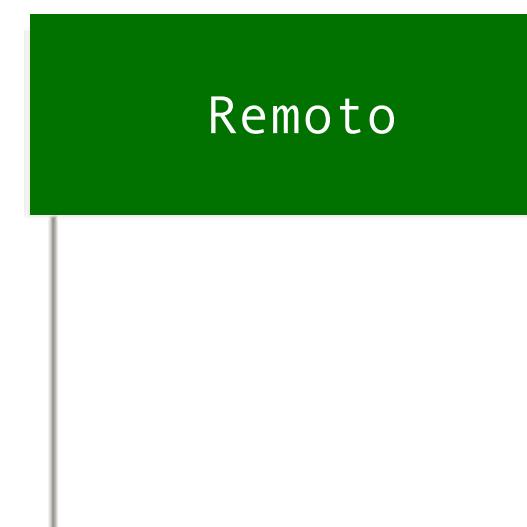
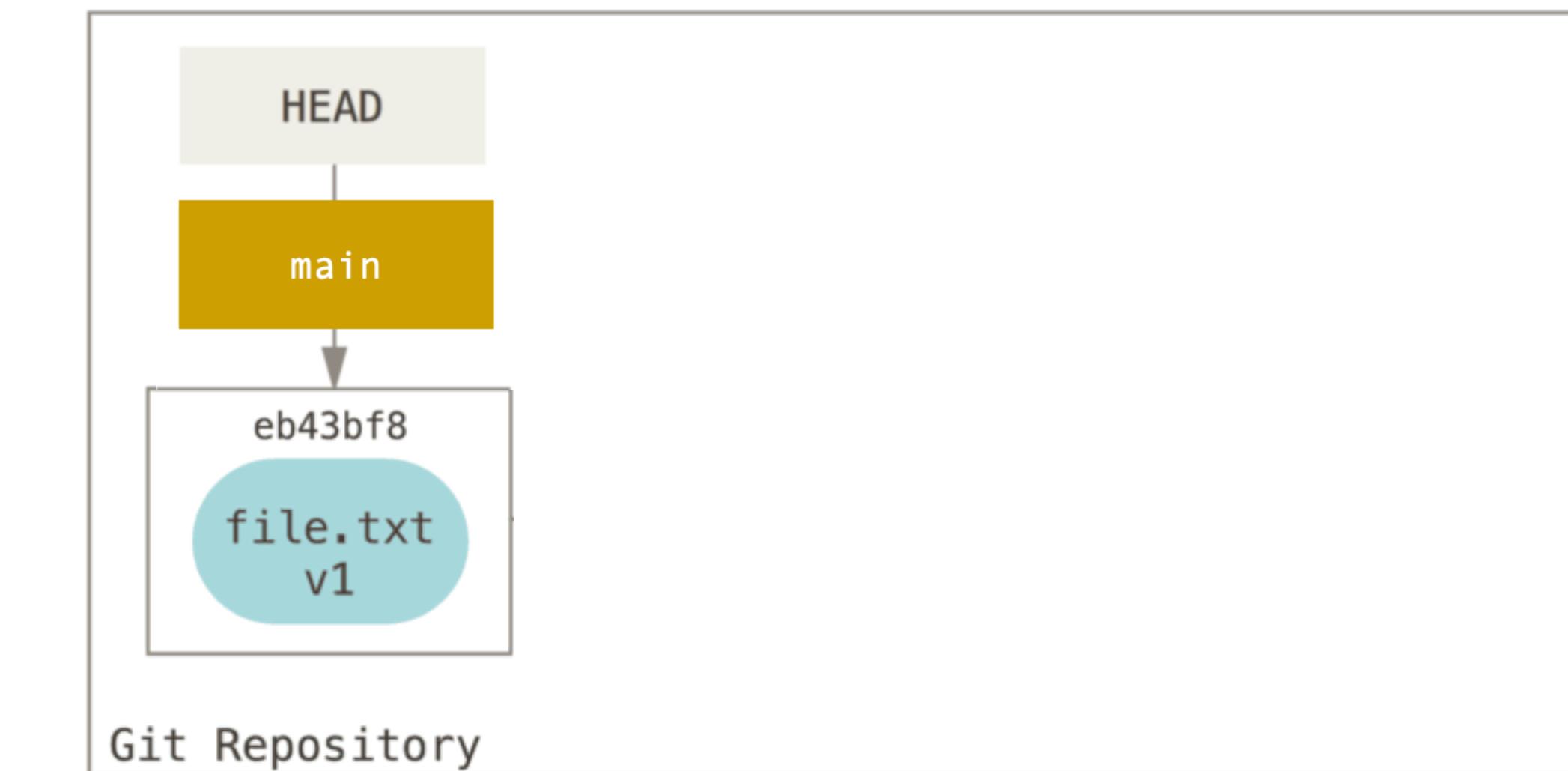
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Supongamos el siguiente escenario





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

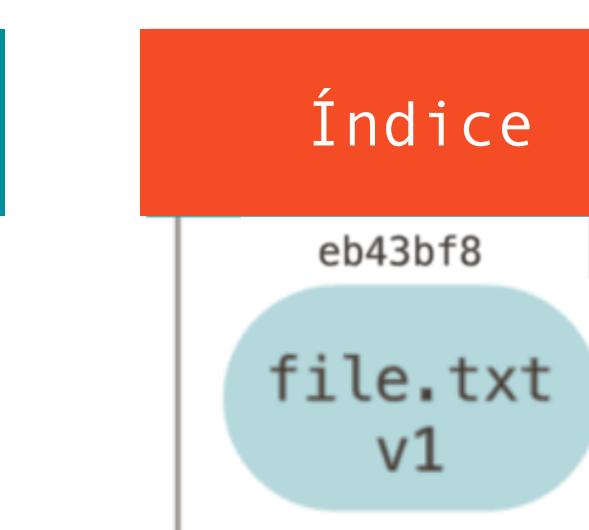
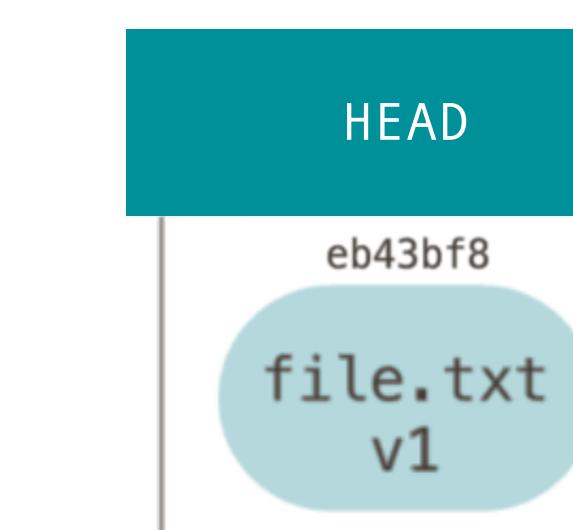
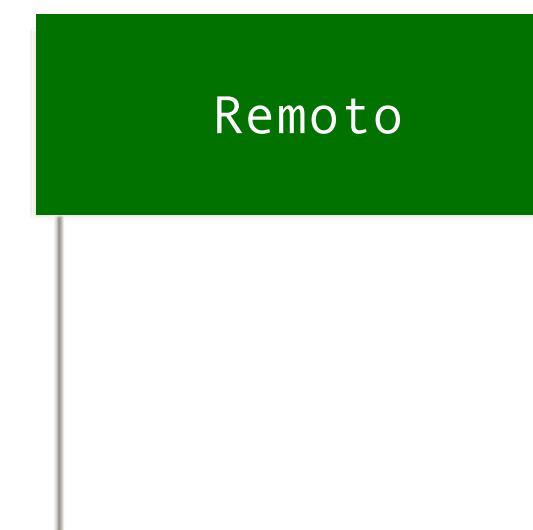
5. Ramas

6. Sincronización  
remota

- Realizamos cambios en el fichero

\$ \_

```
echo "cambios" >> file.txt
```





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

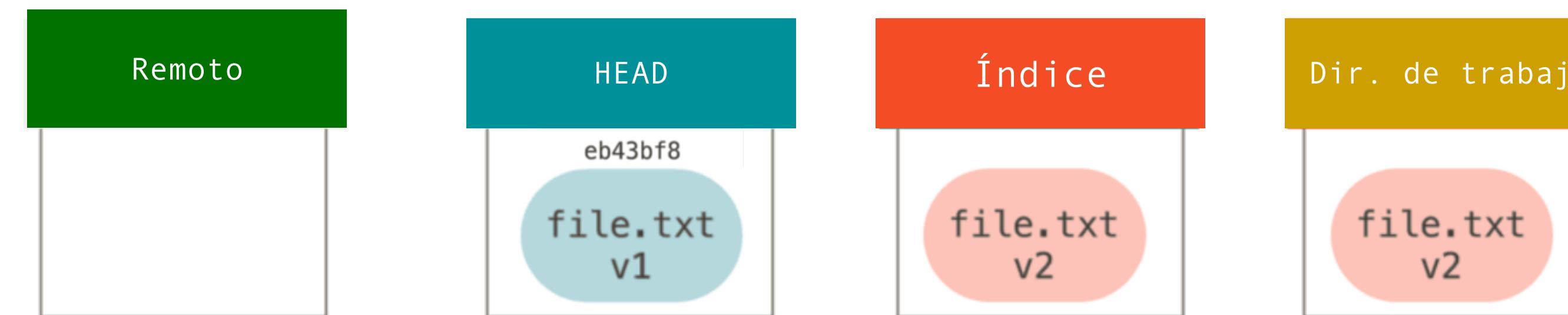
5. Ramas

6. Sincronización  
remota

- Añadimos el fichero a la zona de preparado (staged)

\$ \_

```
git add file.txt
```





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

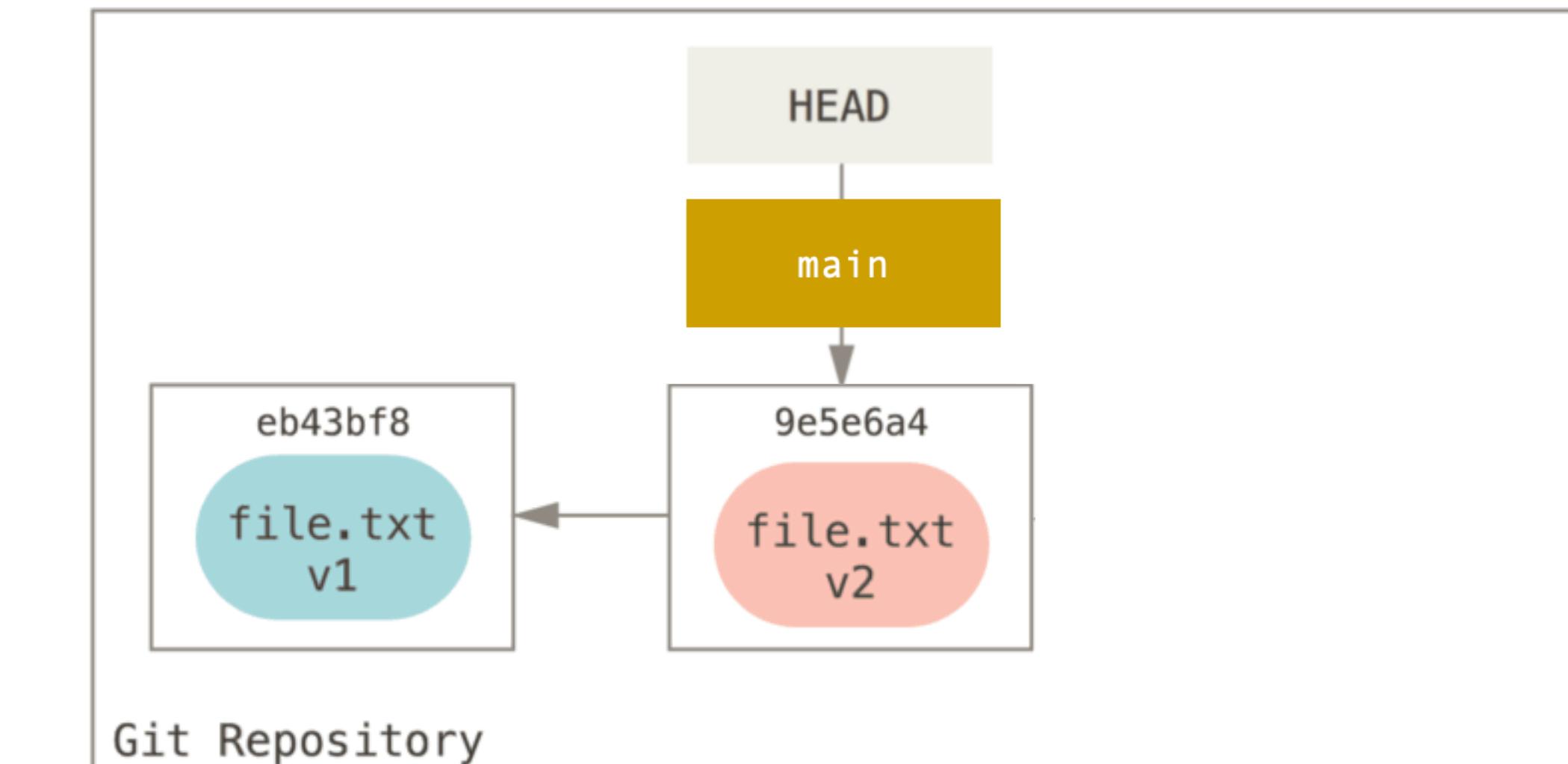
5. Ramas

6. Sincronización  
remota

- Confirmamos los cambios haciendo un commit

\$ \_

```
git commit -m "v2 file.txt"
```



Remoto

HEAD

Índice

Dir. de trabajo



# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

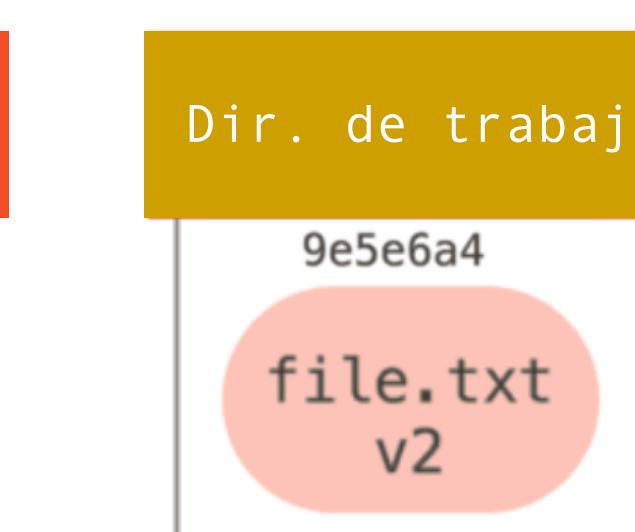
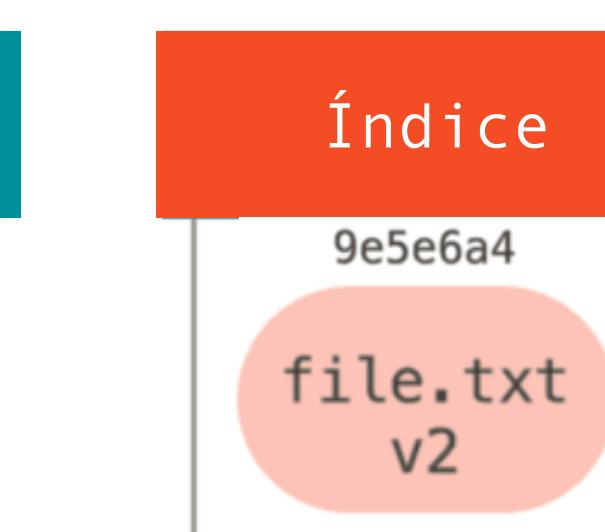
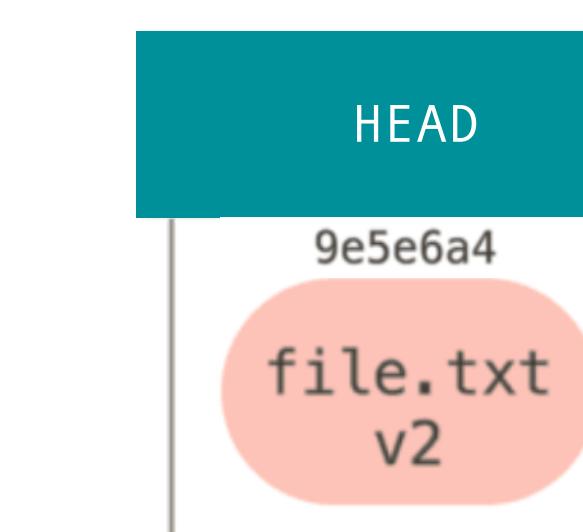
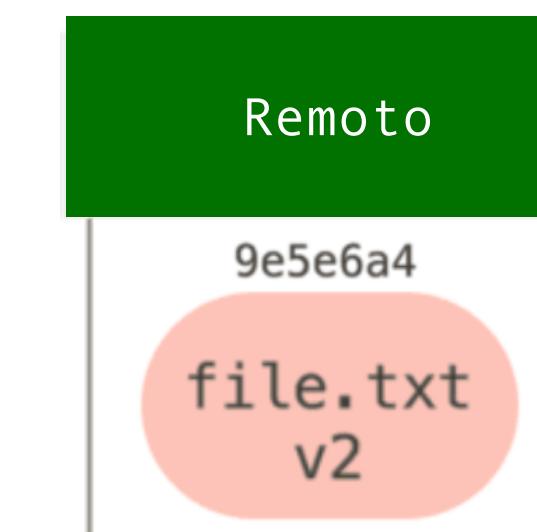
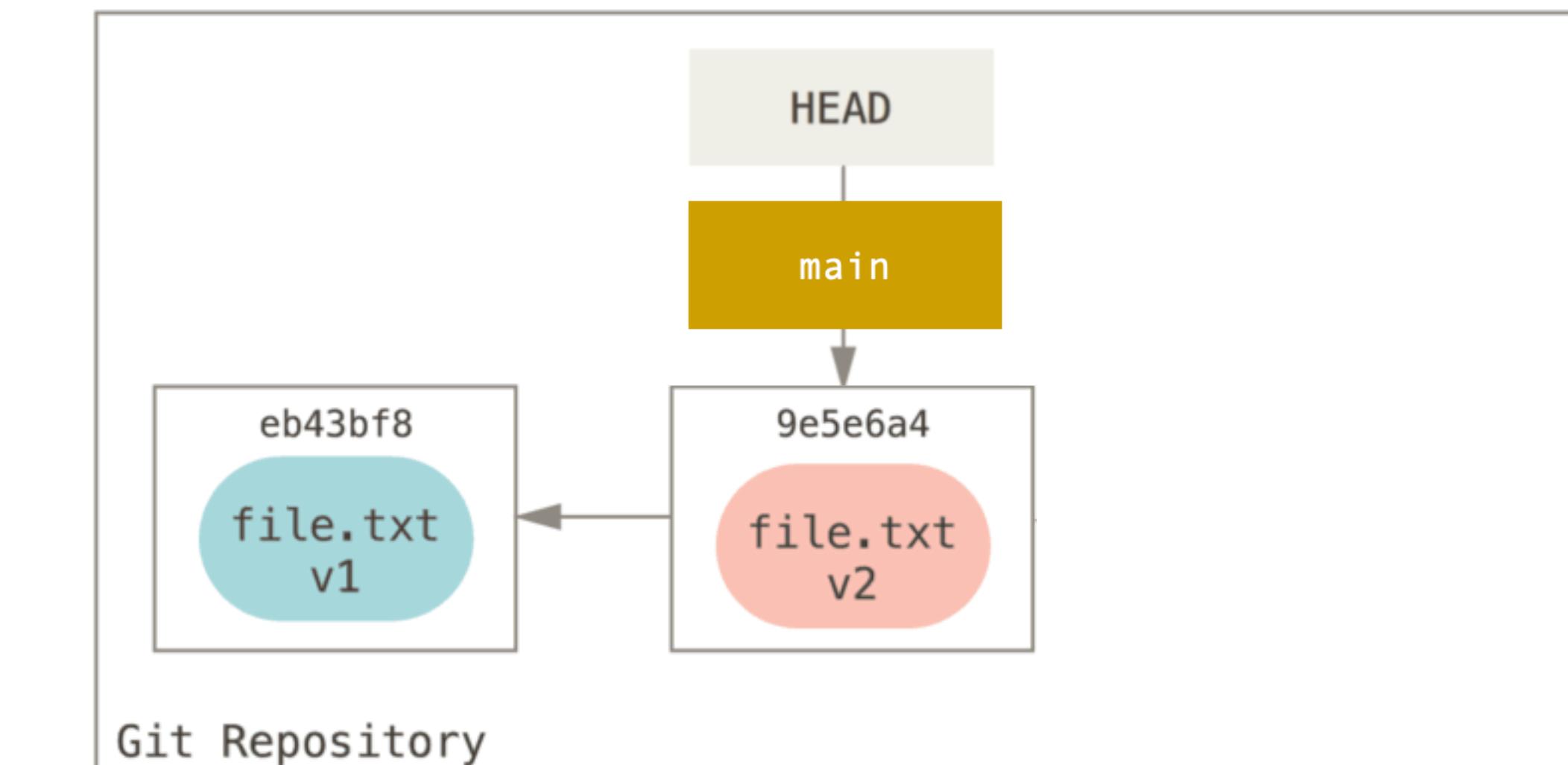
5. Ramas

6. Sincronización  
remota

## ○ Subimos los cambios al servidor

\$ \_

git push





# Subir cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

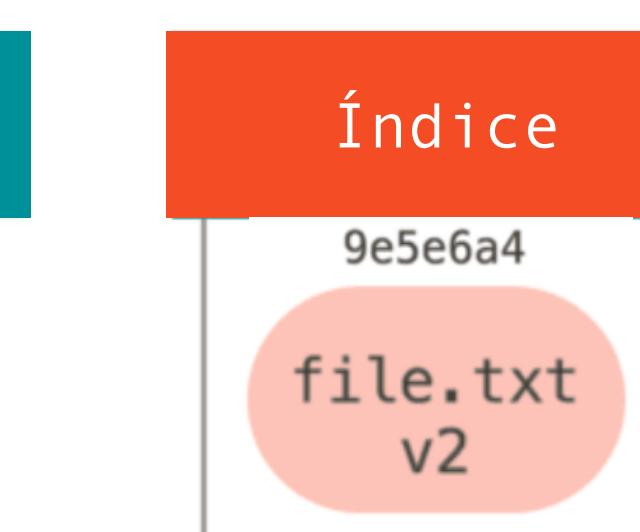
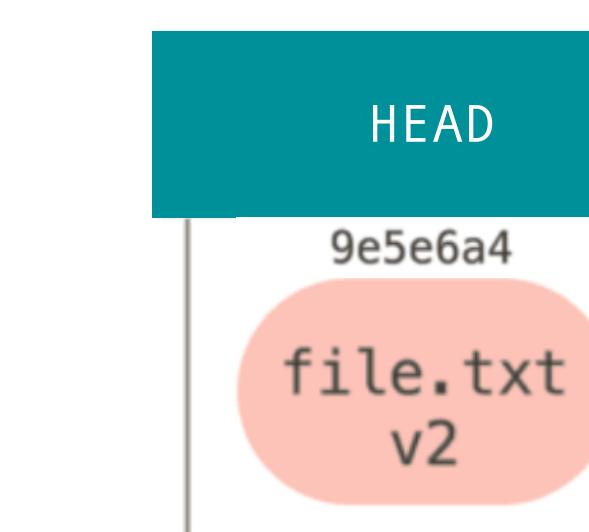
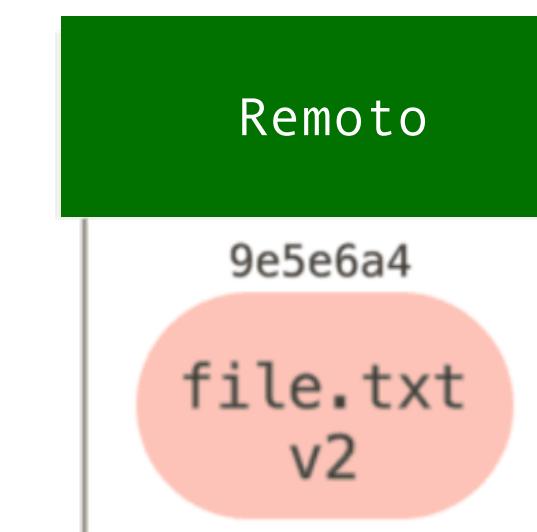
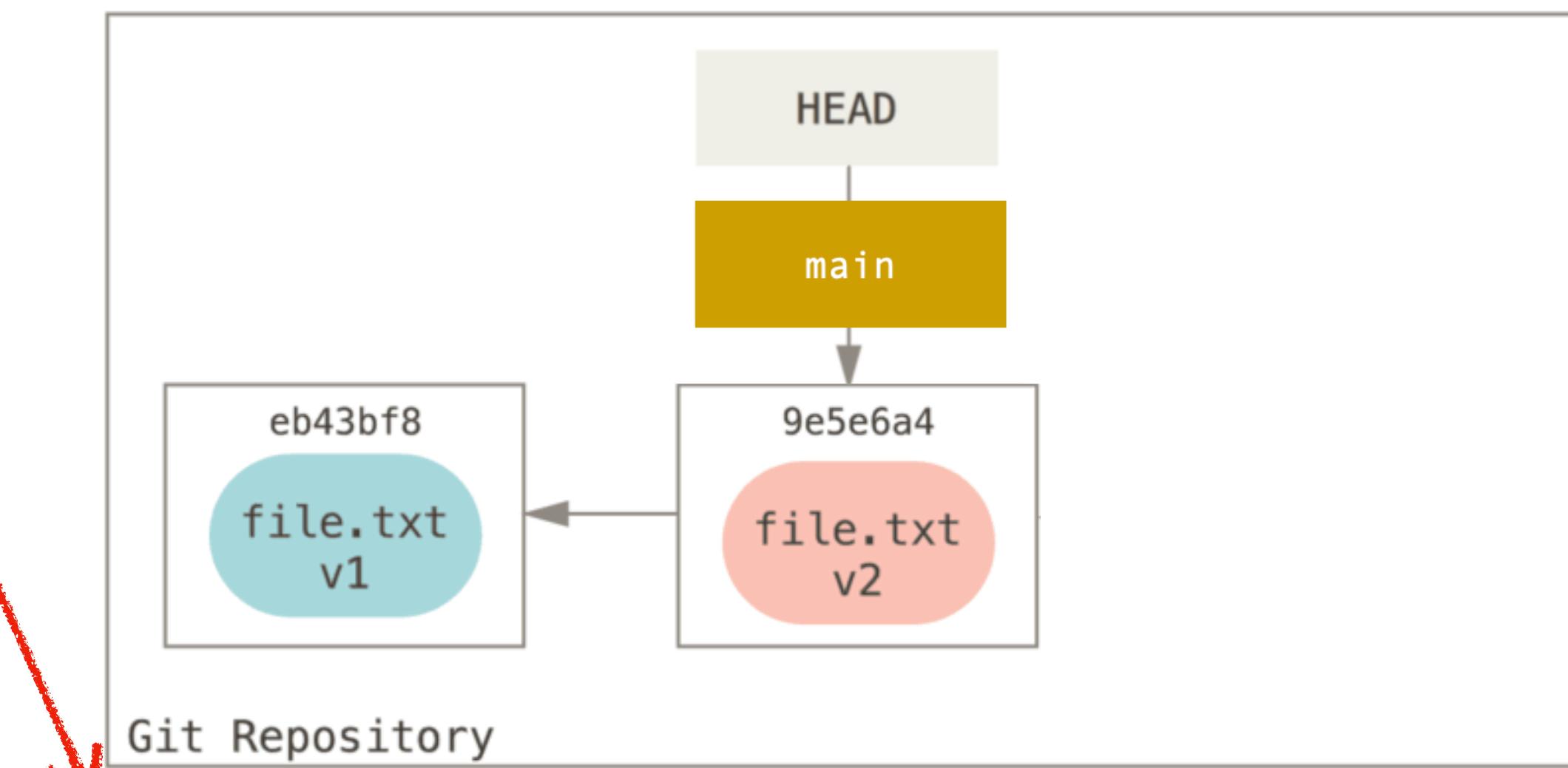
6. Sincronización  
remota

## Subimos los cambios al servidor

\$ \_

git push

*Es  
recomendable  
que todos los  
cambios estén  
en un commit*





# Descargar cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

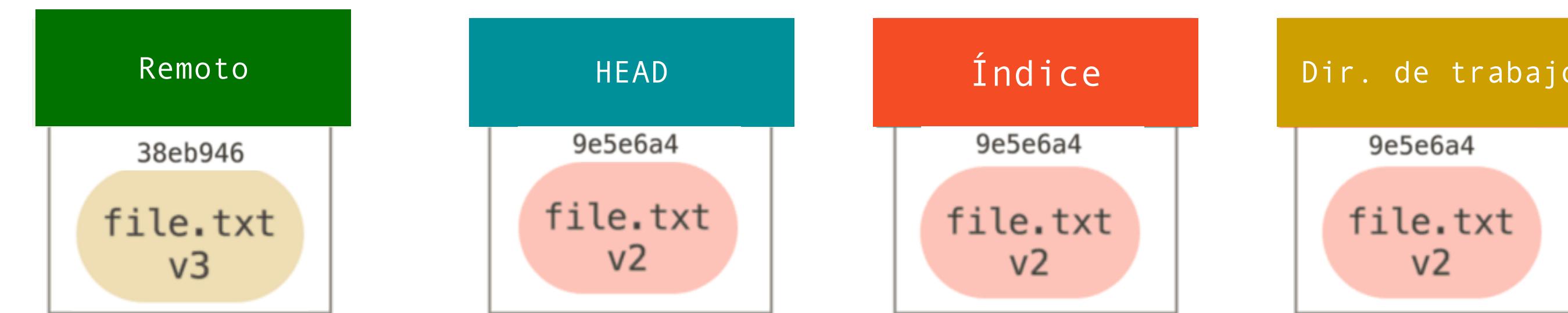
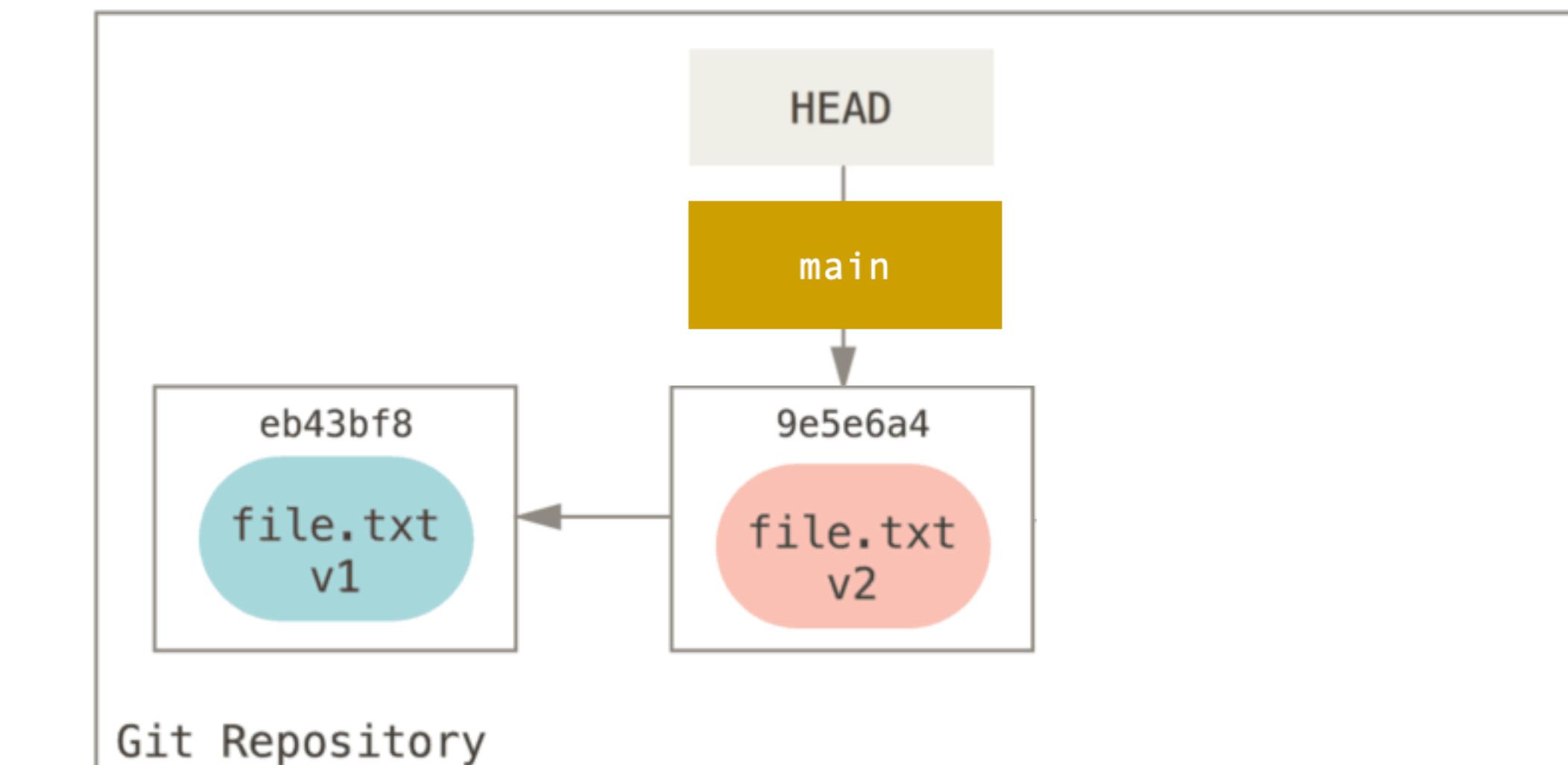
3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

- Desde otra copia local (alguien) modifica el fichero y sube los cambios.





# Descargar cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

## 1. Introducción

## 2. Configuración

## 3. Control de cambios en local

## 4. Correcciones

5. Ramas

## 6. Sincronización remota

- Para descargar los cambios remotos se utiliza el comando pul

\$

git pull

- Es recomendable que estén todos los cambios locales añadidos a algún commit
  - Es posible que se produzcan conflictos, en ese caso hay que resolverlos igual que mezclar ramas locales

—



# Descargar cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

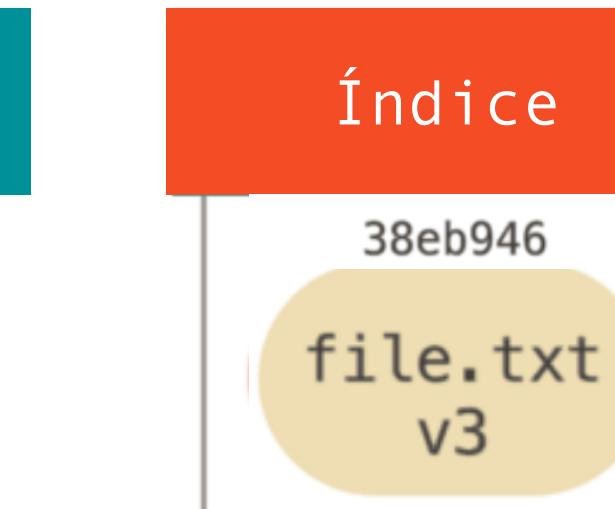
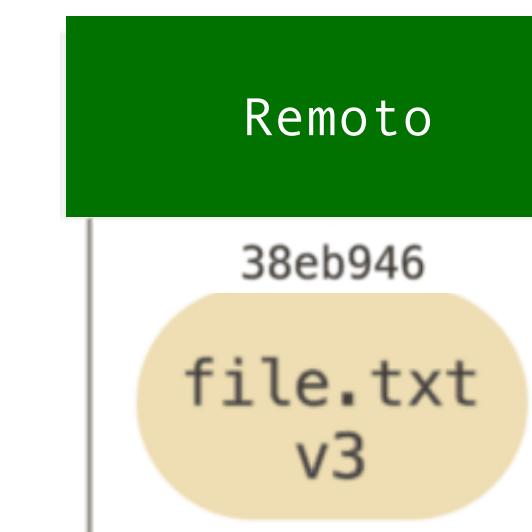
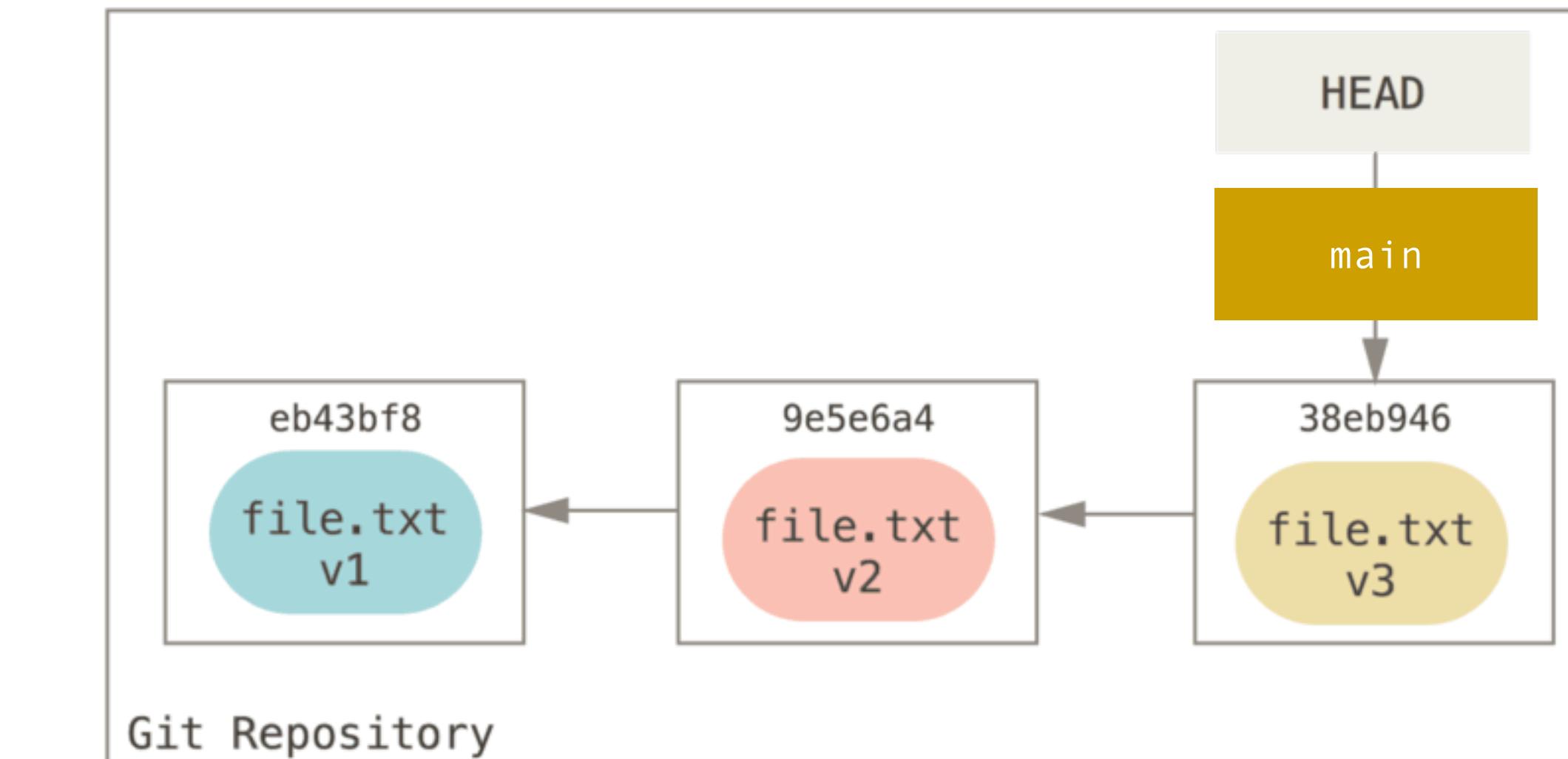
5. Ramas

6. Sincronización  
remota

## ○ Descargamos los cambios

\$ \_

git pull





# Descargar cambios

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

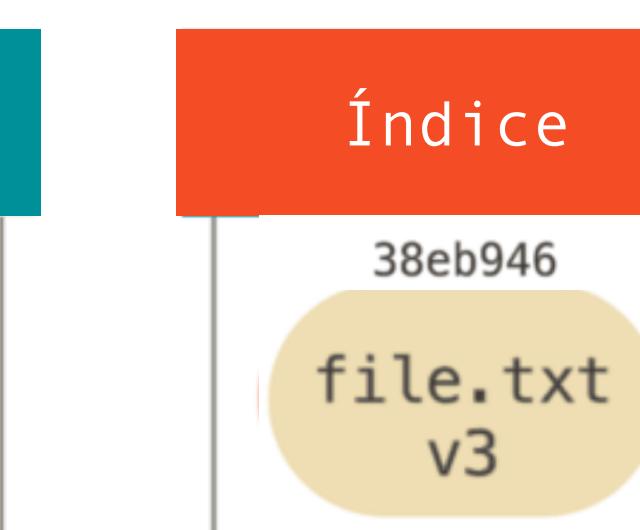
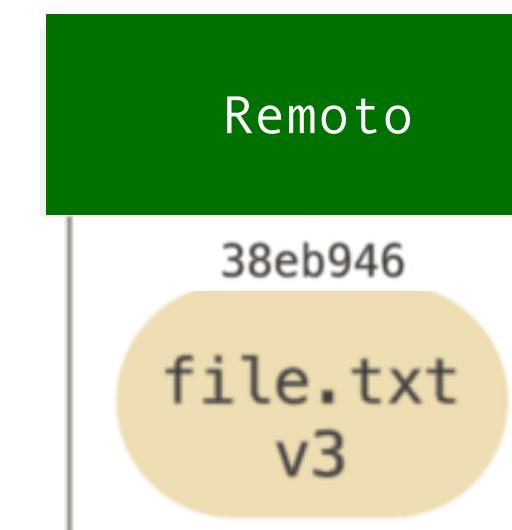
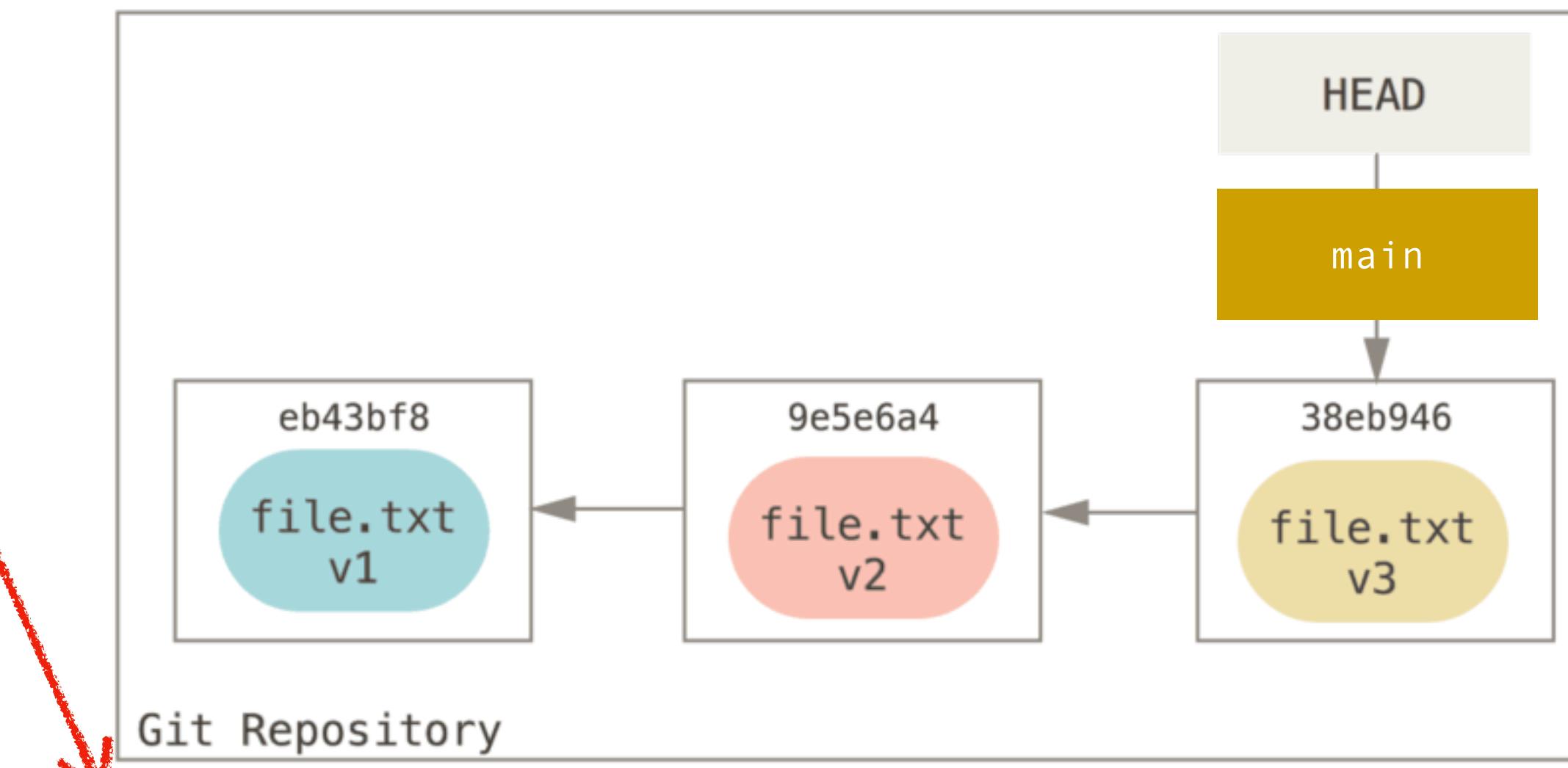
6. Sincronización  
remota

## ○ Descargamos los cambios

\$ \_

git pull

*Todos los  
cambios tienen  
que estar en  
un commit*





# Sincronización remota - Ejercicio 6.1

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



En el repositorio **experimentos**, realiza las siguientes tareas:

- A. Realiza cambios en local y en la web que no generen conflictos. Sincroniza los repositorios locales y el remoto.
- B. Realiza cambios en local y en la web que sí generen conflictos. Sincroniza los repositorios locales y el remoto.
- C. Clona el repositorio en google colab y lanza el script:

```
# Nota: cambia la dirección por la del fork en tu cuenta
!git clone https://github.com/ualgit/experimentos.git

Cloning into 'experimentos'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), 2.87 KiB | 735.00 KiB/s, done.

[5] %cd /content/experimentos
!python run.py

/content/experimentos
[ 1.15150869 -7.3139719  5.5130961 -2.05827155  0.03144393  0.52418475
  1.95609746 -2.65474509 -0.18466379 -3.73718081]
```



GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota

## Tarea final



# Tarea para entregar (opcional)

GitHub

Rafael Cabañas  
rcabanas@ual.es

1. Introducción

2. Configuración

3. Control de  
cambios en local

4. Correcciones

5. Ramas

6. Sincronización  
remota



Crea algún repositorio que pueda ser de utilidad para tu docencia o investigación que incluya las características que se muestran a continuación. Entrega también un documento donde se muestre la lista de comandos generados.

- A. Fichero gitignore
- B. Varias ramas que finalmente se han juntado
- C. Cambios/commit corregidos