



# Git

## Configuración inicial

[Almacenar usuario](#)

[Almacenar credenciales](#)

## Fundamentos

[Crear un repositorio](#)

### Cambios

[Guardar una modificación](#)

[Estado de un repositorio](#)

[Confirmar una modificación](#)

[Deshacer un cambio](#)

[Recuperar un cambio](#)

## Paralelismo

[Crear una rama](#)

[Cambiarse de rama](#)

[Combinar cambios](#)

[Desplazar cambios](#)

[Eliminar una rama](#)

## Funcionalidades

[Alias](#)

[Etiquetas](#)

[Stash](#)

[Archivos especiales](#)

[.gitconfig](#)

[.gitignore](#)

[README.md](#)

## Sincronización de repositorios

[Clonar un repositorio remoto](#)

[Publicar cambios](#)

[Descargar cambios](#)

Los argumentos obligatorios se representan usando

`<>`, los opcionales con `[ ]`.

## Configuración inicial

### Almacenar usuario

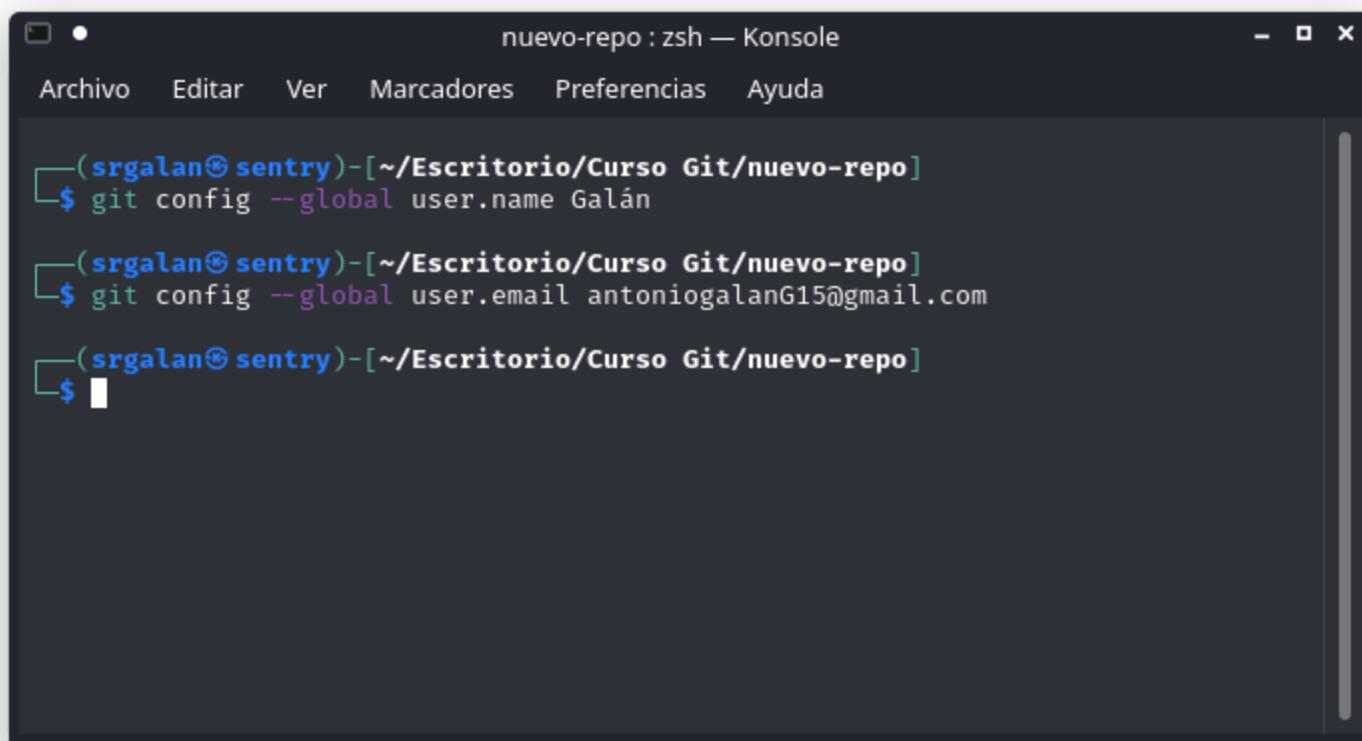
Una vez instalado Git, abre *Git Bash* y ejecuta los siguientes comandos:

```
git config --global user.name <nombre>
```

```
git config --global user.email <email>
```

**Establece tu alias en el sistema.**

**Establece tu correo en el sistema.**



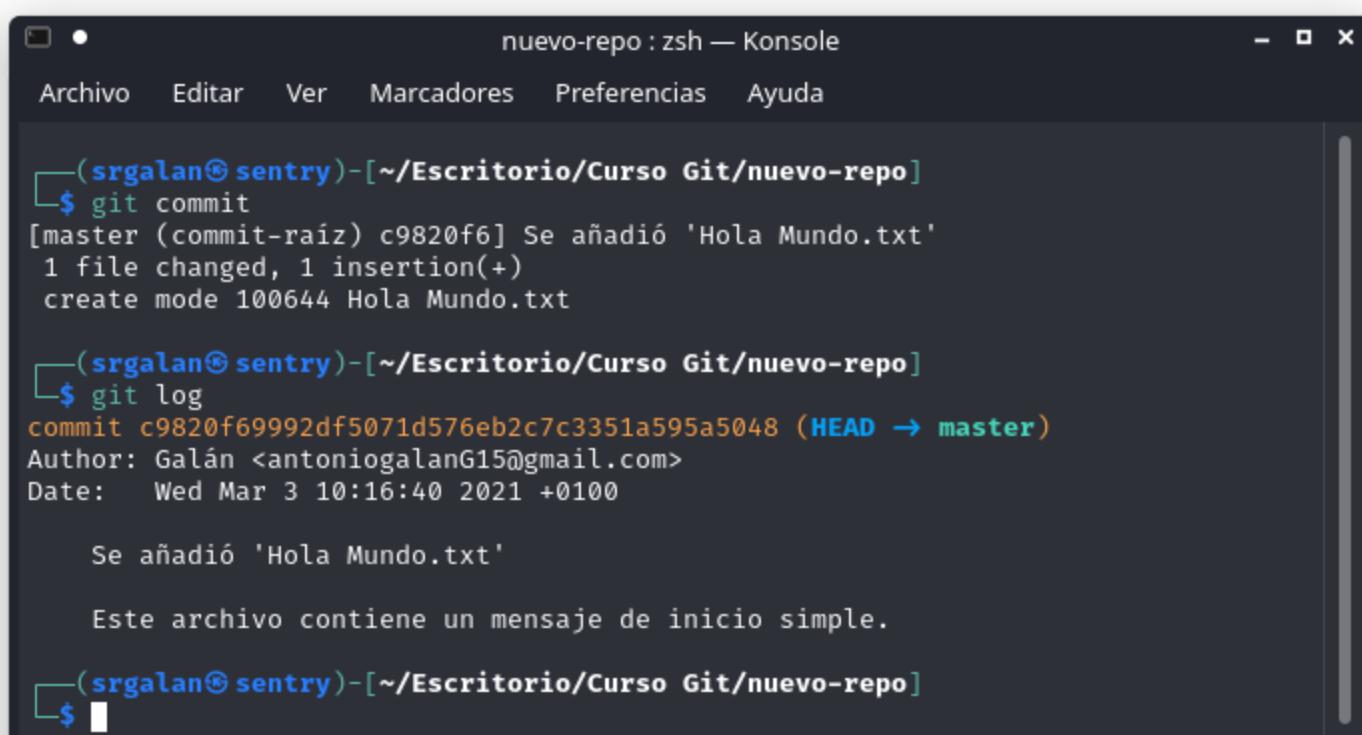
```
(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git config --global user.name Galán

(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git config --global user.email antoniogalanG15@gmail.com

(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
```



Git firmará los commits locales con el nombre de usuario y el correo especificados con los comandos anteriores; por tanto, si el correo coincide con el de una cuenta de GitHub, a la hora de publicar los cambios en un repositorio remoto, estos serán asociados al usuario de GitHub con dicho correo.



```
(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git commit
[master (commit-raíz) c9820f6] Se añadió 'Hola Mundo.txt'
1 file changed, 1 insertion(+)
create mode 100644 Hola Mundo.txt

(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git log
commit c9820f69992df5071d576eb2c7c3351a595a5048 (HEAD → master)
Author: Galán <antoniogalanG15@gmail.com>
Date:   Wed Mar 3 10:16:40 2021 +0100

    Se añadió 'Hola Mundo.txt'

    Este archivo contiene un mensaje de inicio simple.

(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
```



Todos los cambios en la configuración local vienen reflejados en un fichero de texto llamado `.gitconfig`, que se encuentra en la carpeta de usuario del sistema.

Tras haber ejecutado los comandos anteriores, debería aparecer algo similar a esto:

```
nuevo-repo : zsh — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ cat /home/srgalan/.gitconfig
[user]
  name = Galán
  email = antoniogalanG15@gmail.com

(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$
```



Por supuesto, si se modifica el fichero directamente, los cambios también se aplican.

## Almacenar credenciales

```
git config --global credential.helper store
```

### Activa el almacenamiento local de credenciales.

Tras ejecutar por primera vez un `git push` en el sistema, se solicitarán las credenciales de GitHub (usuario en la web y contraseña). Las credenciales serán almacenadas en el fichero `.git-credentials` y desde ese momento no será necesario volver a introducirlas de forma manual.

# Fundamentos

## Crear un repositorio



**Repositorio:** espacio centralizado donde se almacena, organiza, mantiene y difunde información digital, habitualmente archivos informáticos, que pueden contener trabajos científicos, conjuntos de datos o software.

Un repositorio local en Git no es más que una carpeta, pero debe indicarse que es un repositorio.

Para ello, se usa el siguiente comando:

```
git init
```

**Inicializa un repositorio local de Git.**

```
git init <nombre>
```

**Crea un repositorio local de Git con el nombre indicado.**

```
nuevo-repo : zsh — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
└─(srgalan@sentry)-[~/Escritorio/Curso Git]
$ git init nuevo-repo
ayuda: Using 'master' as the name for the initial branch. This default branch name
ayuda: is subject to change. To configure the initial branch name to use in all
ayuda: of your new repositories, which will suppress this warning, call:
ayuda:
ayuda: git config --global init.defaultBranch <name>
ayuda:
ayuda: Names commonly chosen instead of 'master' are 'main', 'trunk' and
ayuda: 'development'. The just-created branch can be renamed via this command:
ayuda:
ayuda: git branch -m <name>
Inicializado repositorio Git vacío en /home/srgalan/Escritorio/Curso Git/nuevo-repo/.git/
└─(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$
```

## Cambios



**Cambio:** cualquier modificación de un fichero en el repositorio.

### Guardar una modificación

Estas modificaciones deben ser tratadas adecuadamente, para eso se usa el *área de preparación*.

```
git add <fichero(s)>
```



Pueden añadirse varios archivos a la vez separándolos como argumentos.

**Añade un cambio en el área de preparación.**

```
nuevo-repo : zsh — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
└─(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git add Hola\ Mundo.txt
└─(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$ git status
En la rama master

No hay commits todavía

Cambios a ser confirmados:
  (usa "git rm --cached <archivo> ..." para sacar del área de stage)
    nuevo archivo: Hola Mundo.txt

└─(srgalan@sentry)-[~/Escritorio/Curso Git/nuevo-repo]
$
```



Cabe destacar que **guardar un cambio** y **confirmar un cambio** son cosas diferentes:

- Lo primero se usa para llevar un control de los ficheros.
- Lo segundo se usa para llevar un control sobre el estado del repositorio.

## Estado de un repositorio

Para comprobar el estado de un repositorio puede usarse el siguiente comando:

```
git status
```



Aparecerán en **rojo** los cambios no guardados, y en **verde** los que sí.

**Muestra los cambios pendientes y guardados del repositorio.**

## Confirmar una modificación



**Commit:** confirmación permanente de cambios en un repositorio.

Un commit consta de 2 partes:

1. **Título:** resumen genérico y breve de los cambios (una sola línea).
2. **Descripción:** información sobre los cambios con un poco más de detalle (campo opcional).

```
git commit
```

```
git commit -m "<Título>"
```

**Crea un commit solicitando sus 2 campos.**

**Crea un commit con el título indicado y sin descripción.**



No es posible crear un commit si el área de preparación está vacía.



**Árbol de Trabajo:** historial de commits de un repositorio.

## Deshacer un cambio

En ocasiones es necesario dar un paso atrás, por suerte Git tiene eso muy en cuenta y permite hacer *rollbacks* de forma segura. Sin embargo, hay que tener muy en cuenta la siguiente tabla:

|                        | Repositorio Local | Repositorio Remoto |
|------------------------|-------------------|--------------------|
| Repositorio propio     | Fácil             | Molesto            |
| Repositorio compartido | Fácil             | Cuidado            |



Este comando sirve tanto para **deshacer cambios**, como para **deshacer commits**.

```
git reset <fichero(s)>
```

```
git reset [opción] <commit>
```

Saca del área de preparación los ficheros indicados.

Elimina los commits posteriores al indicado.

Si el comando se está aplicando sobre un commit, deben tenerse en cuenta estas opciones:

- `--soft`: mantiene los cambios en el área de preparación.
- `--mixed`: mantiene los cambios en el área de trabajo (opción por defecto).
- `--hard`: elimina los cambios.

## Recuperar un cambio



Usando `git reset <commit-reflog>`, es posible revertir el repositorio a un estado concreto.

```
git reflog
```



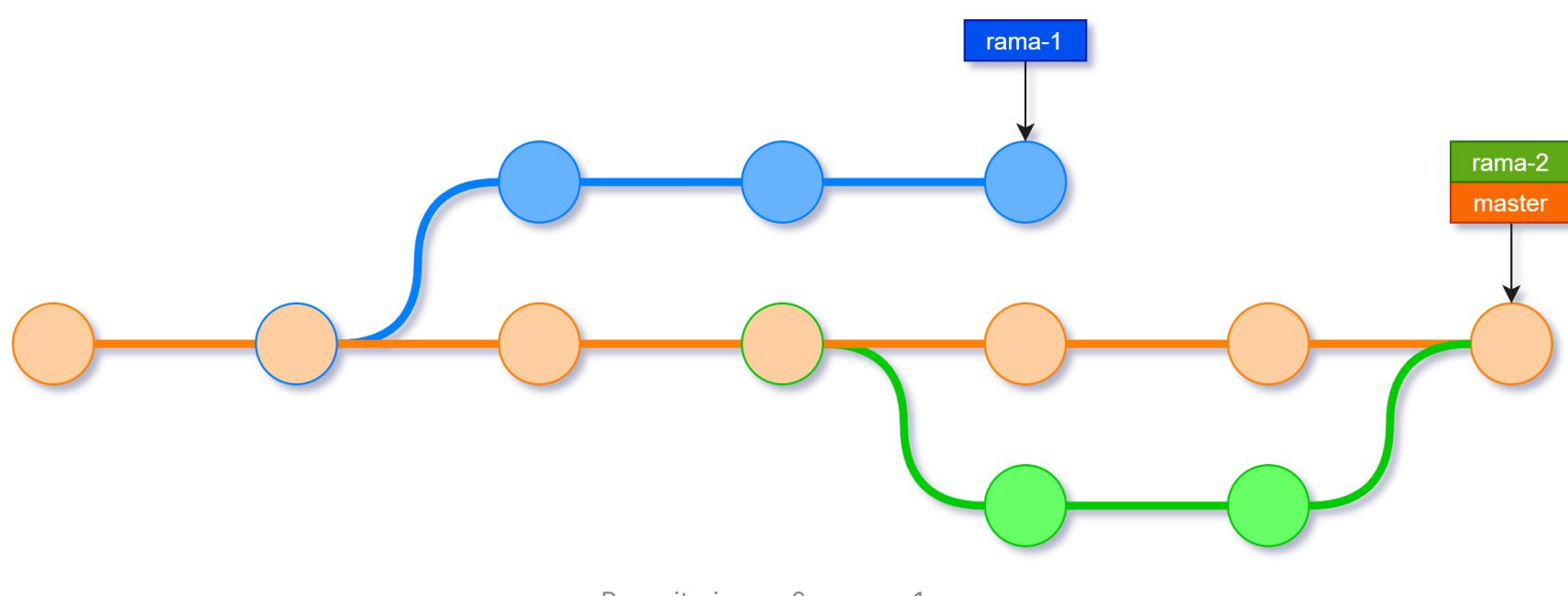
Útil para recuperar commits eliminados.

Muestra el historial de acciones ejecutadas sobre el repositorio.

## Paralelismo



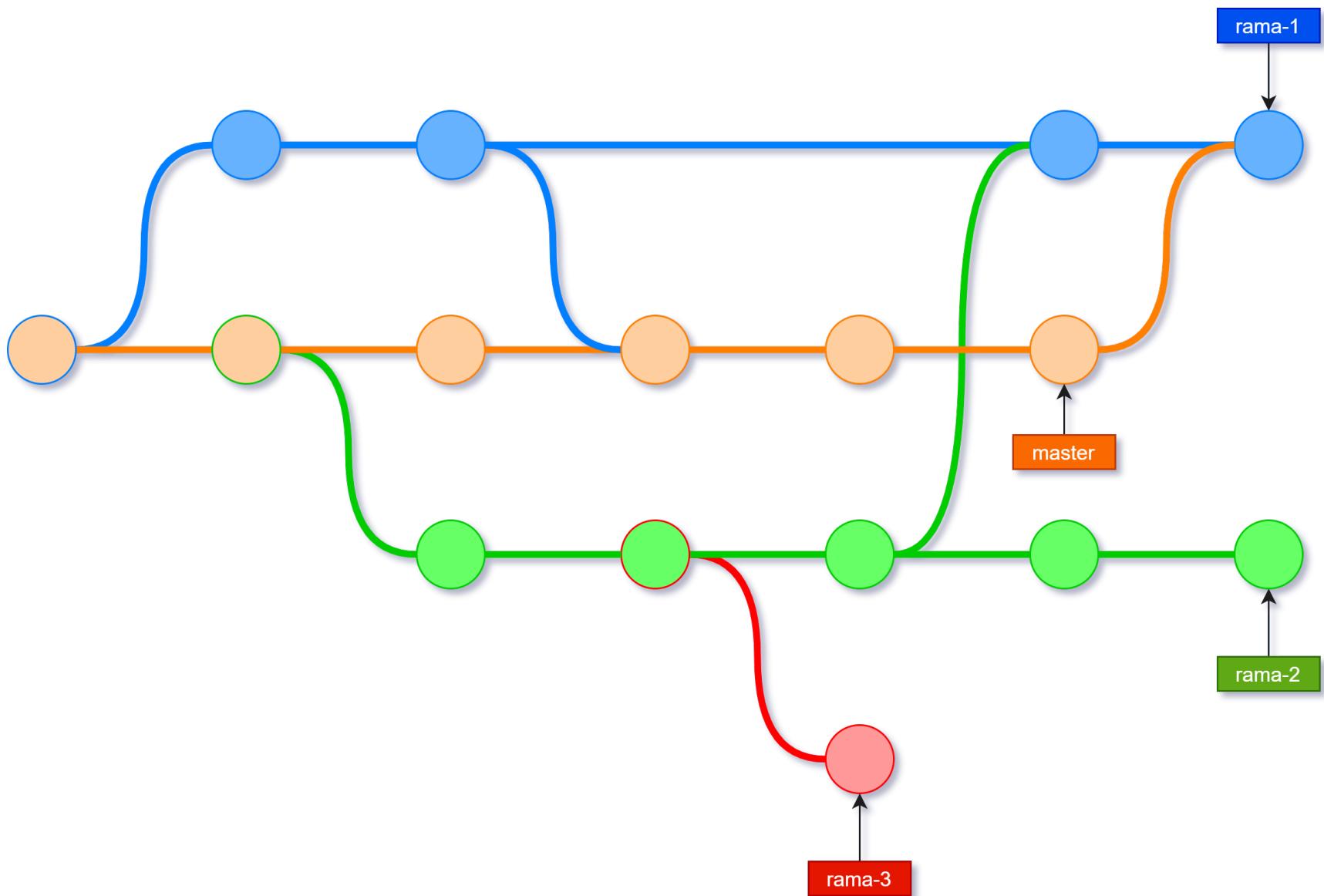
Rama: flujo de commits.



Repositorio con 2 ramas y 1 merge



Una rama se identifica con un puntero que referencia al último commit de esa rama.



Repositorio con 4 ramas y 3 merges

## Crear una rama



Una rama siempre parte desde un commit concreto.

`git branch`

**Muestra una lista de las ramas locales del repositorio.**

`git branch <rama> [commit]`

**Crea una rama con el nombre indicado, desde el commit indicado.**

## Cambiarse de rama

Los cambios realizados en una rama son independientes de las demás; por tanto, será necesario moverse de rama en rama según la situación lo requiera.

Para ello, pueden usarse los siguientes comandos:

`git checkout <rama>`

**Cambia la rama de trabajo actual a la indicada, si existe.**

`git checkout -b <rama>`

**Crea y cambia la rama de trabajo actual a la indicada.**

! No es posible cambiar de rama mientras haya cambios pendientes.

## Combinar cambios



**Fusión (merge):** unión de los cambios de una rama sobre los cambios de otra.

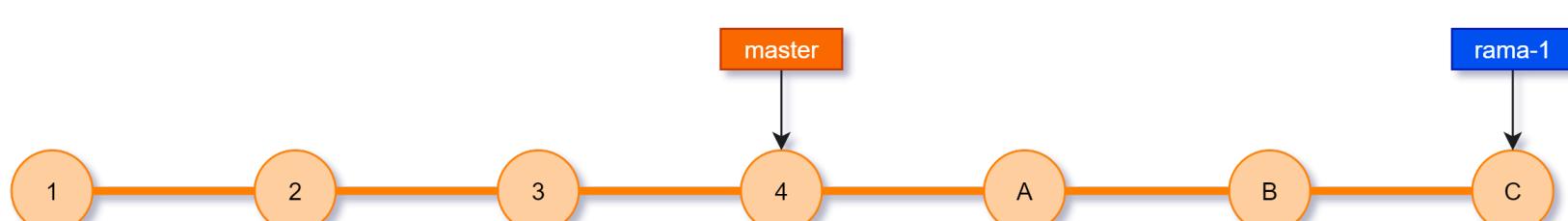
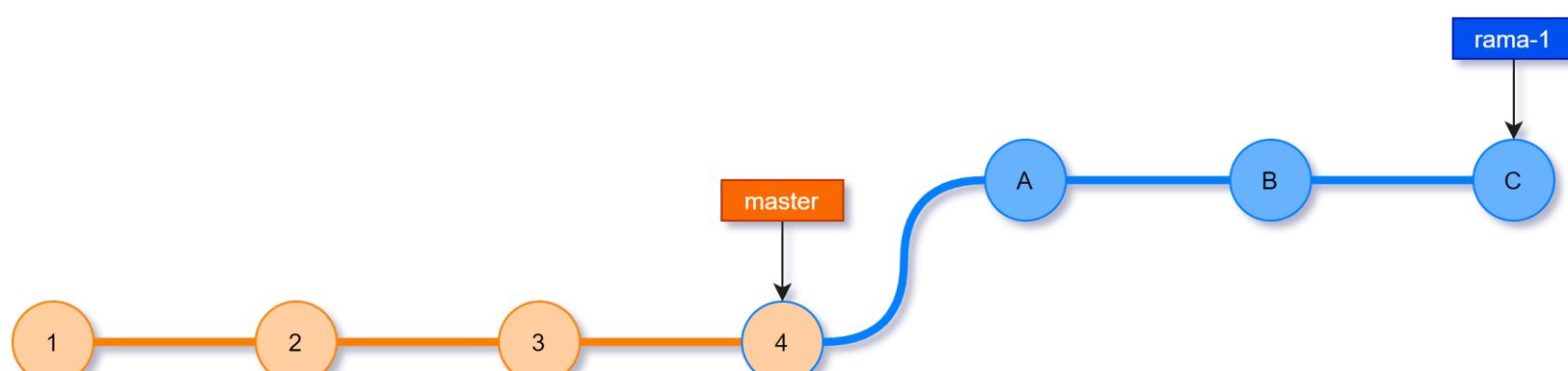
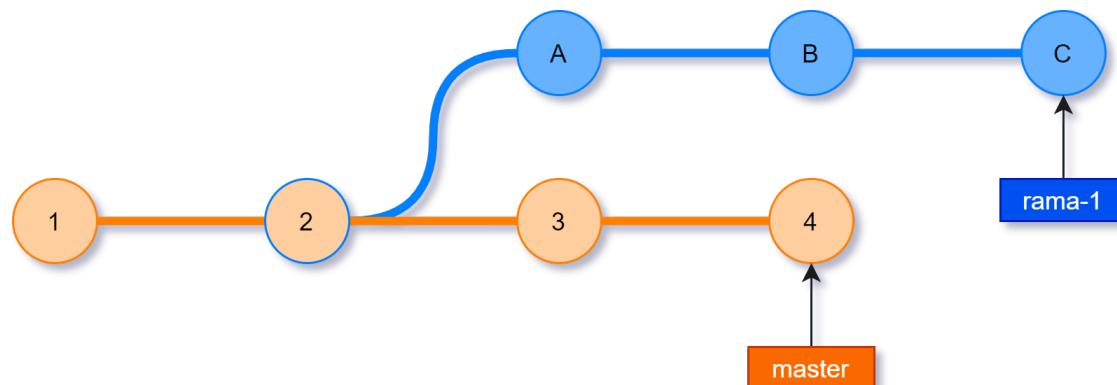
```
git merge <rama>
```

**Combina los cambios de la rama indicada con los de la rama de trabajo actual.**



Los cambios siempre **se traen**; por tanto, el usuario debe colocarse en la rama destino antes de hacer el `merge`.

## Desplazar cambios



Descripción del proceso tras ejecutar `git rebase master`, desde la rama `rama-1`

```
git rebase <rama>
```

**Coloca los commits de la rama de trabajo actual, detrás de la rama indicada.**



Para actualizar el puntero de la rama atrasada, basta con repetir el proceso con dicha rama.

## Eliminar una rama

```
git branch -d <rama>
```

**Elimina la referencia a la rama indicada del repositorio local, pero no sus commits.**

```
git branch -D <rama>
```

**Elimina la referencia a la rama indicada del repositorio local, y sus commits.**

```
git push origin --delete <rama>
```

**Elimina la rama indicada del repositorio remoto.**

## Funcionalidades

### Alias



Básicamente, un alias permite crear comandos de git personalizados, usando otros comandos.

```
git config --global alias.<nombre> "<acción>"
```

**Crea un alias con el nombre indicado y que realiza la acción descrita.**



Normalmente se crean alias conforme el usuario detecta que emplea varios comandos concretos de forma habitual, lo que facilita su uso.

## Etiquetas



Git gestiona las etiquetas internamente como si fueran ramas que constan solo de su puntero: por tanto, las ramas y las etiquetas presentan un comportamiento similar en cuanto a sintaxis de comandos de git y funcionalidades.

```
git tag
```

**Muestra una lista de las etiquetas existentes.**

```
git tag <etiqueta> [commit]
```

**Crea una etiqueta con el nombre indicado, y la asocia al commit indicado.**

```
git push origin <etiqueta>
```

**Sincroniza la etiqueta indicada con el repositorio remoto.**

```
git push origin --tags
```

**Sincroniza todas las etiquetas con el repositorio remoto.**

```
git tag -d <etiqueta>
```

**Elimina la etiqueta indicada del repositorio local.**

```
git push --delete <etiqueta>
```

**Elimina la etiqueta indicada del repositorio remoto.**

## Stash



Consiste en una zona de la memoria donde poder almacenar cambios pendientes, de forma que no interfieran con procesos sensibles como cambios de ramas, fusiones...

```
git stash push <fichero(s)>
```

**Almacena en el stash de la rama de trabajo actual los ficheros indicados.**

```
git stash pop <fichero(s)>
```

**Extrae del stash de la rama de trabajo actual los ficheros indicados.**



Cada rama tiene su propio stash y son independientes entre sí.

## Archivos especiales

Estos archivos son interpretados por Git para activar una serie de funcionalidades en específico, algunos se crean de forma automática, mientras que otros deben crearse manualmente.

### .gitconfig



Contiene los datos de la configuración local de Git.

- Se ubica en la carpeta de usuario del sistema.
- Modificar los datos cambia la configuración de forma directa.

### .gitignore



Contiene rutas relativas tipo LEX para identificar archivos a los que el repositorio no ofrece seguimiento.

- Debe crearse manualmente.
- Debe ubicarse en la raíz de un repositorio.

### README.md



Este archivo escrito en formato MARKDOWN (.md) tiene la función de mostrar su contenido como texto enriquecido en la web de GitHub.

- Debe crearse manualmente.
- El uso depende mucho del usuario, pero como su nombre indica (README), suele contener información acerca del contenido del repositorio, de su funcionamiento, de su instalación...

15Galan / mencion-330

Code Issues Pull requests Actions Projects 4 Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file Code

**15Galan Tema 4 y Práctica 4 añadidos** 4e76a35 18 days ago 54 commits

Códigos y ejercicios Tema 4 y Práctica 4 añadidos 18 days ago

Temario Tema 4 y Práctica 4 añadidos 18 days ago

.gitignore Archivo .gitignore añadido 5 months ago

Presentación.pdf Presentación de la asignatura añadida 5 months ago

Help people interested in this repository understand your project by adding a README. Add a README

About Desarrollo de Servicios Telemáticos

Releases No releases published Create a new release

Packages No packages published Publish your first package

Languages Java 88.3% HTML 9.6% CSS 1.5% JavaScript 0.6%

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

Repository sin archivo README.md

15Galan / asignatura-302

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 4 branches 0 tags Go to file Add file Code

**15Galan Fusión con la rama de Galán ...** a592218 on 23 Jan 85 commits

Práctica final Práctica Final corregida y terminada last month

Recursos Archivos reordenados 5 months ago

Temario Parcial 3 modificado last month

.gitignore Archivo .gitignore modificado last month

README.md README.md modificado 5 months ago

README.md

**Manual de Git casero**

Este documento es una guía básica y simple escrita por mí y para mis amigos, con información acerca de Git y su uso entre varias personas, en concreto para esta asignatura (pequeño proyecto personal).

Lo más interesante creo que sería la sección de [trabajo en equipo](#), así que dejo un enlace ahí para consultarla rápidamente.

Descargas:

- Última versión de [Git](#).
- Recomiendo la GUI que ofrece [GitKraken](#) para principiantes que quieran «ver» el estado del repositorio.

Notas:

- Todos los comandos que aparecen durante este documento están descritos en la sección de [comandos](#) al final del mismo.
- Los argumentos en los comandos vienen indicados con `<>`, no es que deban escribirse, literalmente, como `<argumento>`.

About Procesadores de Lenguajes

Releases No releases published Create a new release

Packages No packages published Publish your first package

Languages Java 67.2% Lex 27.1% Shell 4.1% Other 1.6%

Repository con archivo README.md

# Sincronización de repositorios

## Clonar un repositorio remoto

```
git clone <url-repositorio> [carpeta]
```

Descarga el repositorio remoto indicado sincronizado, y lo guarda en la carpeta indicada.

## Publicar cambios



Cada vez que se crea una rama local que no existe en el repositorio remoto es necesario vincularla inicialmente, esto hará que Git y GitHub se entiendan a la hora de publicar los commits.

El siguiente comando debe usarse solo la primera vez que se publiquen cambios desde una rama:

```
git push -u origin <rama>
```

**Publica los commits locales en la rama indicada, creando una vinculación entre la rama local y la remota.**

```
git push
```

**Publica los commits locales de la rama actual de trabajo que no estén sincronizados remotamente.**

```
git push <rama>
```

**Publica los commits locales de la rama indicada que no estén sincronizados remotamente.**

## Descargar cambios

```
git pull
```

**Descarga los commits remotos en la rama de trabajo actual y los mezcla con los locales.**

```
git fetch
```

**Descarga los commits remotos en la rama de trabajo actual.**