

git y GitHub

Taller práctico 28-29 de mayo 2025

Qué es el control de versiones?

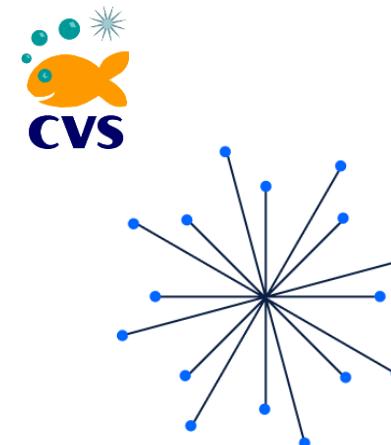
Los programas para control de versiones son un grupo de aplicaciones originalmente ideadas para gestionar ágilmente los **cambios en el código** fuente de los programas y poder **revertirlos**.

Su ámbito se ha ampliado para englobar a todas las actividades que pueden realizarse por un **equipo** sobre un **gran proyecto de software** u otra actividad que genere **ficheros digitales**.

Modelo centralizado:

repositorio central al que acceden usuarios.

Ejemplo código abierto: **cvs**



Centralized

Modelo distribuido:

cada desarrollador trabaja en su repositorio local.

Ejemplo código abierto: **git**



Distributed

Aspectos básicos de git: áreas de trabajo

git tiene tres estados principales en los que se pueden encontrar tus archivos:
modificado (**modified**), preparado (**staged**) y confirmado (**committed**).



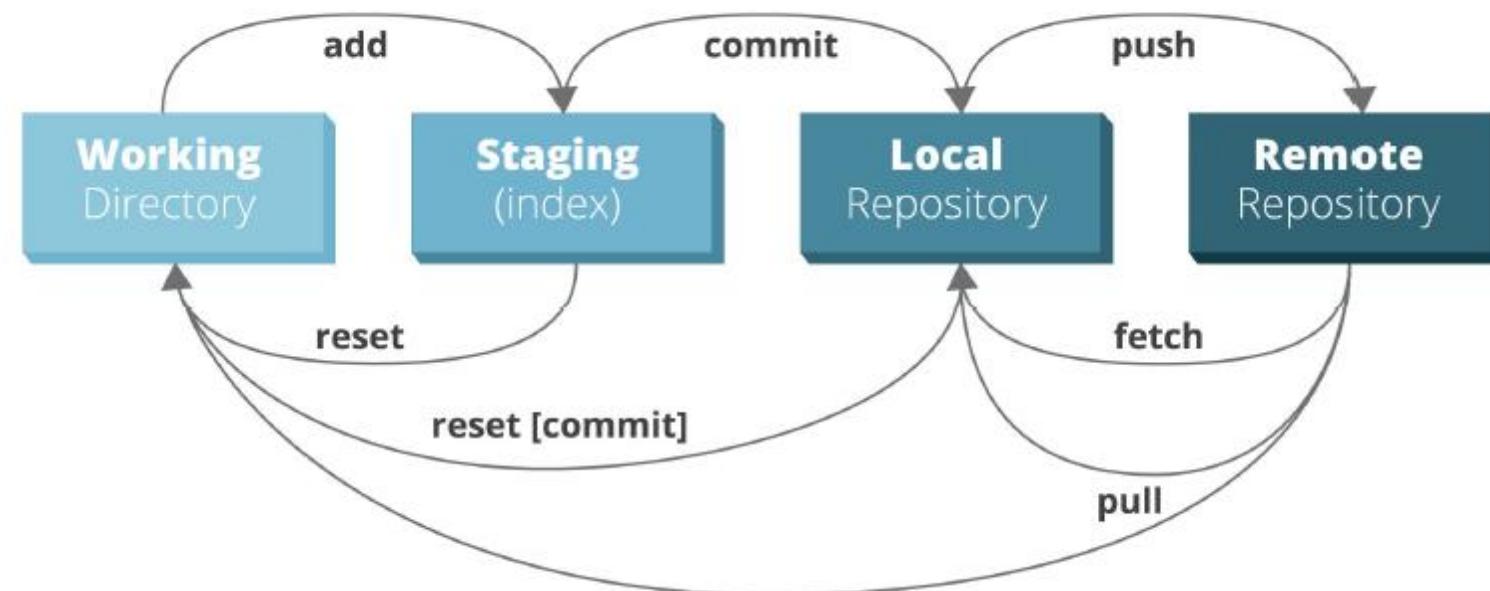
- **Working Directory:** donde trabajas en los archivos.

- **Staging Area (Index):** donde añades los archivos para el seguimiento.

- **Repository:** donde se almacenan permanentemente los cambios.

Aspectos básicos de git: áreas de trabajo

git tiene tres estados principales en los que se pueden encontrar tus archivos:
modificado (**modified**), preparado (**staged**) y confirmado (**committed**).



- **Working Directory:** donde trabajas en los archivos.

- **Staging Area (Index):** donde añades los archivos para el seguimiento.

- **Repository:** donde se almacenan permanentemente los cambios.

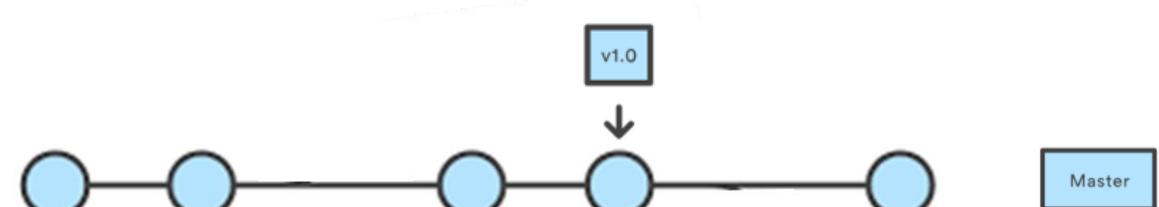
Aspectos básicos de git: flujo de trabajo

Flujo básico de trabajo

El flujo de trabajo básico en git implica tres etapas principales:

- 1.Modificas archivos en tu **Working Directory**.
- 2.Añades esos cambios al área de **Staging**.
- 3.Confirmas esos cambios en el **repositorio**.

Modificación → git add → git commit

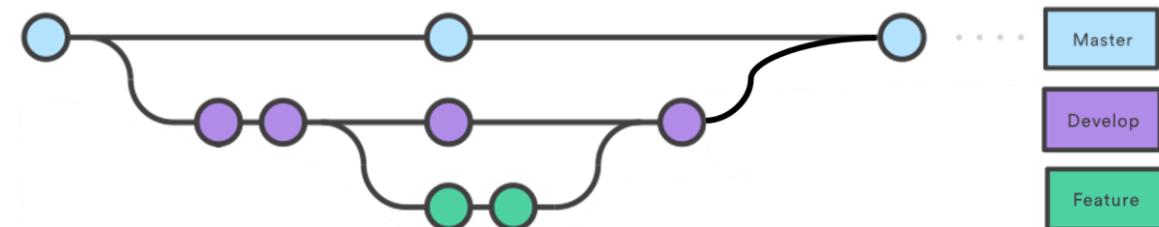


Flujo de trabajo con ramas

Las **ramas** (*branches*) son simplemente líneas independiente de desarrollo.

- 1.Creas una nueva rama
- 2.Trabajas en tu rama
- 3.Fusionas las ramas (o no)

git checkout → flujo básico → git merge



Aspectos básicos de git: forja remota

forja – servidores o alojamientos online que ofrecen espacio de almacenamiento para crear repositorios.

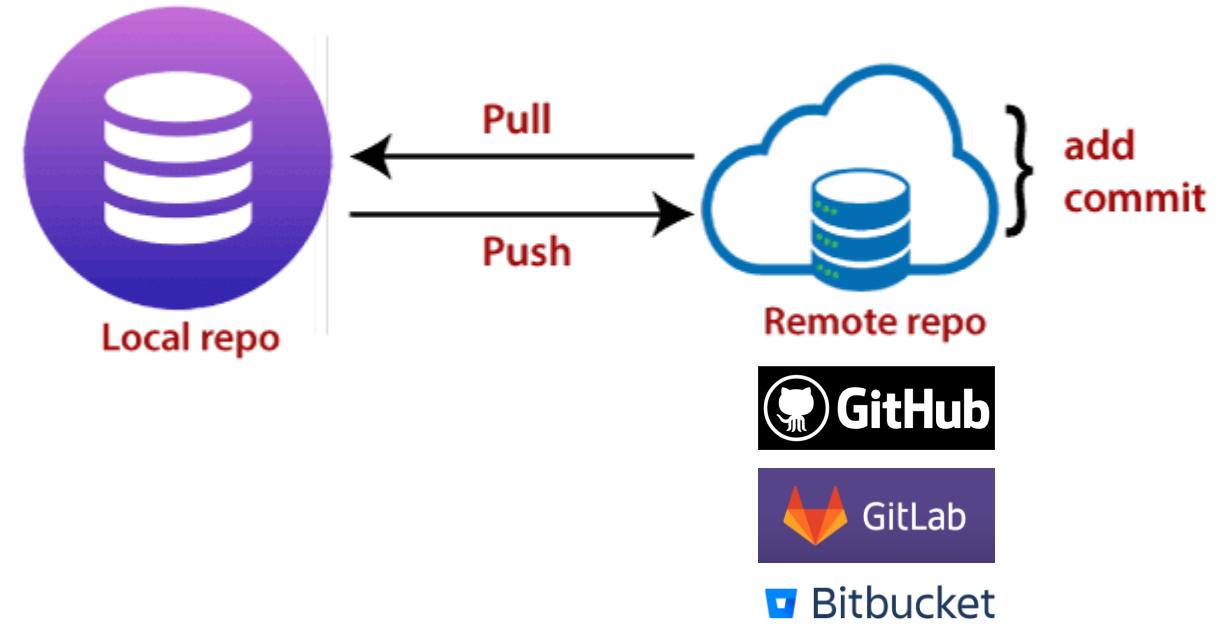
Flujo Típico de Sincronización

La sincronización de los cambios con el repositorio remoto requiere pasos adicionales.

1. Clonas/sincronizas un repositorio remoto.
2. Trabajas de forma local.
3. Subes tus cambios al repositorio remoto.

4. Obtienes cambios desde el repo remoto.

`git clone` → *trabajo local* → `git push`

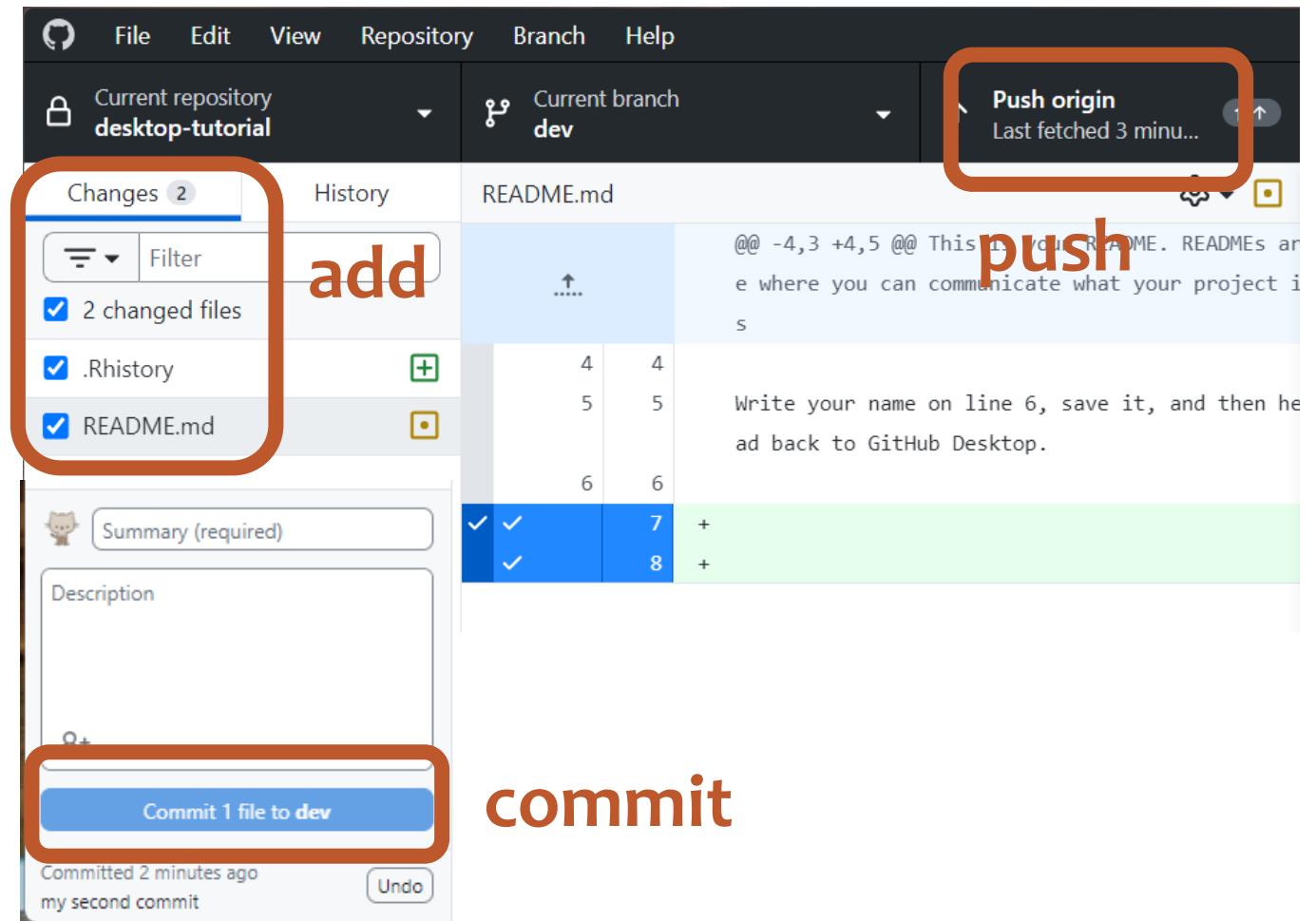


Contenido del taller y requisitos

Vamos a usar **git + GitHub**

→ desde la línea de comandos (CLI).

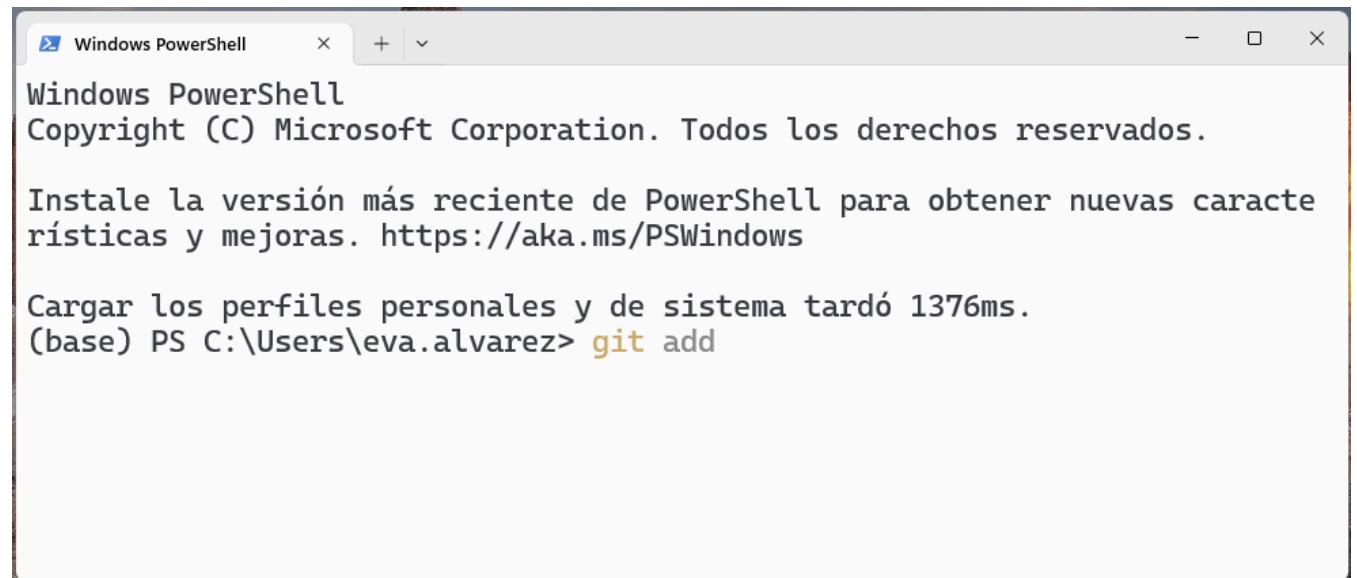
→ existen alternativas gráficas (GUI).



Contenido del taller y requisitos

Vamos a usar **git + GitHub**

- desde la línea de comandos (CLI).
- existen alternativas gráficas (GUI).

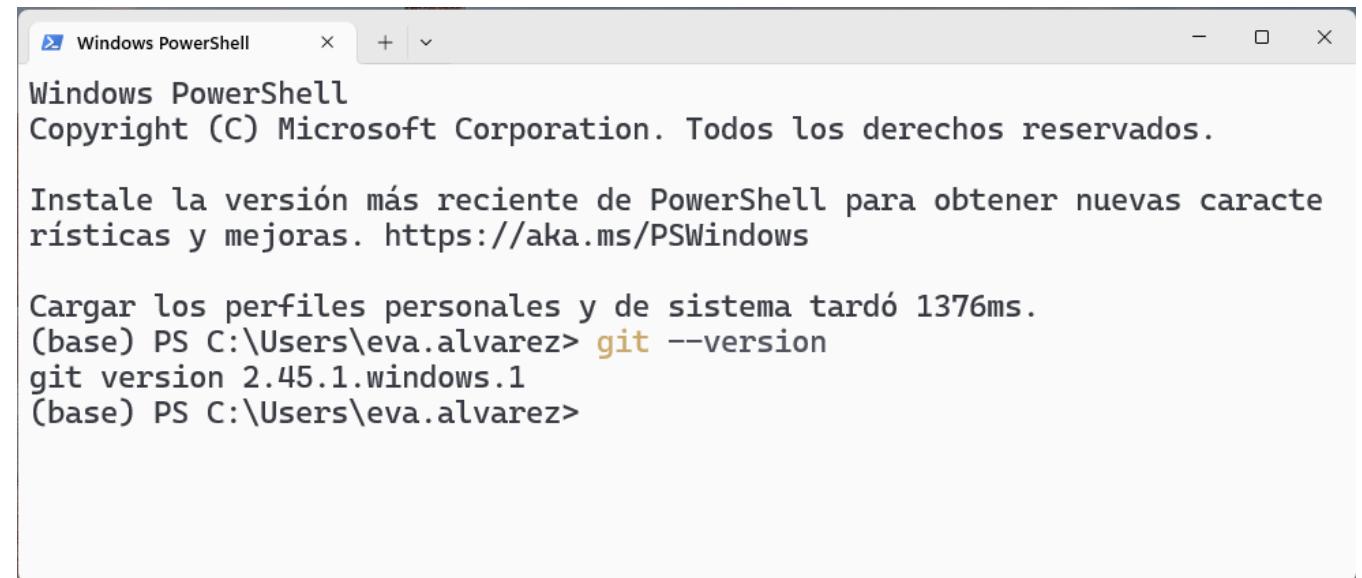


A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following text:
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. <https://aka.ms/PSWindows>
Cargar los perfiles personales y de sistema tardó 1376ms.
(base) PS C:\Users\eva.alvarez> **git add**

Contenido del taller y requisitos

Vamos a usar git + GitHub

- desde la línea de comandos (CLI).
- existen alternativas gráficas (GUI).



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following text:
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. <https://aka.ms/PSWindows>
Cargar los perfiles personales y de sistema tardó 1376ms.
(base) PS C:\Users\eva.alvarez> git --version
git version 2.45.1.windows.1
(base) PS C:\Users\eva.alvarez>

Requisitos previos para cada parte

Parte 1: Control de versiones en repositorios locales (git)

- acceso a terminal
- haber instalado git

Parte 2: Control de versiones en repositorios remotos (GitHub)

- cuenta de usuario en GitHub
- haber generado un par de claves ssh

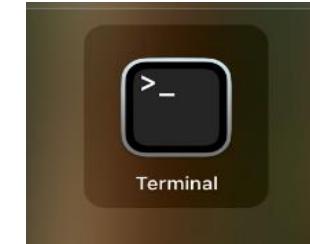
Checkpoint Parte 1

Tenemos todos acceso a un terminal?

Windows 11 →



Linux/macOS →



Tenemos todos git?

Windows: `winget install --id Git.Git -e --source winget`

Linux: `apt-get install git`

MacOS: `git -h`

Tenemos todos *username* en GitHub?

<https://docs.github.com/es/get-started/start-your-journey/creating-an-account-on-github>

Parte 1: repositorios locales



PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Directorio de trabajo

Muévete al directorio de trabajo:

```
cd 'Your directory'
```

Comprueba la ruta:

```
pwd
```

y el contenido:

```
ls
```

Uso del comando git

```
git <command> [-s | --subcommand]
```

Accede a la lista de comandos comunes:

```
git -h
```

```
git help git
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_1. Configuración básica e iniciar un repositorio

Comprueba la configuración con:

```
git config --list
```

Sustituye los datos entrecomillados por los tuyos y ejecuta los siguientes comandos:

```
git config --global user.name 'Tu Nombre'
```

```
git config --global user.email 'tucorreo@ejemplo.com'
```

Esto le dice a git quién eres. Repite:

```
git config --list
```

y verás como aparece la información adicional.

Crea una carpeta para tu repositorio:

```
mkdir mi-primer-repo
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_1. Configuración básica e iniciar un repositorio

Muévete a la carpeta de tu repositorio:

```
cd mi-primer-repo
```

Inícialo como repositorio Git:

```
git init
```

Crea un archivo dentro del repositorio

```
echo "# Mi primer repositorio" > README.md
```

Verifica el estado del repositorio:

```
git status
```

Ahora vamos a cambiar algunos aspectos de configuración, solo para este repositorio:

```
git config user.name 'Tu Nombre (local)'
```

```
git config user.email tunuevocorreo@ejemplo.com
```

Comprueba tu nueva configuración local con:

```
git config --local --list
```

y comprueba la global que configuramos antes con:

```
git config --global --list
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_2. Seguimiento de archivos y primer commit

Dentro del repositorio que has creado, comprueba su estado:

```
git status
```

Crea algunos archivos adicionales, y vuelve a comprobar el estado del repositorio:

```
Out-File -FilePath notas.txt -Encoding utf8
```

```
touch script.py notas.txt datos.csv
```

```
git status
```

Los archivos *untracked* no están bajo control de versiones. Añádelos con:

```
git add README.md (solo un archivo)
```

```
git add README.md script.py (varios archivos)
```

```
git add . (todo lo que está en el directorio actual y subdirectorios)
```

Y vuelve a comprobar el estado con:

```
git status
```

Haz tu primer commit:

```
git commit -m 'Mi primer commit'
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_2. Seguimiento de archivos y primer commit

extra: Cómo deshacer cosas en Git

Para quitar archivos de la zona de preparación (staging):

Opción 1: quitar un archivo concreto:

```
git rm --cached archivo.txt
```

Opción 2: quitar todos los archivos del staging:

```
git rm --cached .
```

Modificar el último commit

Corregir solo el mensaje del commit más reciente:

```
git commit --amend -m 'Primer commit con mensaje más claro'
```

Revisa el historial de commits actualizado:

```
git log
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_3. Modificar, verificar y actualizar repositorio

Modifica uno o más archivos existentes. Por ejemplo:

```
echo "Este es mi primer proyecto con git." >> README.md  
echo "print('Hola Git!') " >> script.py
```

Comprueba el estado del repositorio:

```
git status
```

Añade los archivos modificados al staging:

```
git add README.md script.py
```

Haz un nuevo commit:

```
git commit -m 'Actualizar archivos con contenido nuevo'
```

Revisa el historial de commits:

```
git log
```

Revisa el historial de commits (con hash):

```
git log --oneline
```

Flujo básico de trabajo

modificar -> git add -> git commit

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_3. Modificar, verificar y actualizar repositorio

extra: Cómo deshacer cosas en Git

Eliminar el último commit (pero conservar los cambios en tu directorio):

```
git reset --soft HEAD~1
```

Esto borra el commit, pero los cambios siguen listos para hacer un nuevo commit.

Compruébalo con:

```
git log
```

```
git status
```

Eliminar el último commit y quitar también del staging:

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

Compruébalo con:

```
git log
```

```
git status
```

PRÁCTICA 1: Iniciar, configurar y controlar un repositorio local

Ejercicio 1_3. Modificar, verificar y actualizar repositorio

extra: Comandos útiles para revisar cambios

¿Qué cambios se han hecho en el repositorio?

```
git log  
git log --oneline  
git log --graph --oneline --all  
git log archivo  
git log -p archivo
```

¿Qué contiene un cambio concreto?

```
git show  
git show hash  
git diff  
git diff archivo  
git diff hash1 hash2
```

Fin PRÁCTICA 1

Buenas prácticas al hacer commits

⌚ ¿Qué ventajas ves en hacer commits frecuentes frente a guardar todo en un solo commit grande al final?

Commits pequeños, con sentido y mensajes claros :

- **Historial más claro:** cada commit cuenta una parte del progreso.
- **Es más fácil volver atrás** si algo se estropea.
- **Permite trabajar modularmente:** puedes aislar errores fácilmente.
- **Ayuda en el trabajo en equipo:** es más fácil revisar y entender los cambios de otros.
- Cómo escribir buenos mensajes de commit.

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_1. Introducción al trabajo con ramas

¿Alguna vez habéis hecho una copia de un archivo antes de probar algo por miedo a estropearlo?
Pues eso es una rama. Pero git lo hace mejor.

Cómo ver las ramas existentes. La actual aparecerá con un asterisco (*).

```
git branch -av
```

Crea una nueva rama:

```
git branch mejora-readme
```

Y cámbiate a ella:

```
git checkout mejora-readme
```

O haz las dos cosas a la vez:

```
git checkout -b mejora-readme2
```

Comprueba de nuevo las ramas (fíjate que son todas iguales):

```
git branch
```

```
git branch -av
```

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_1. Introducción al trabajo con ramas

Haz cambios en esta rama:

```
echo "He hecho este cambio en la rama mejora-readme2." >> README.md  
git add README.md  
git commit -m 'Anadir contexto al README en rama mejora-readme2'
```

Comprueba en tu explorador de archivo que README.md ha sido modificado.

Y vuelve a master:

```
git checkout master  
git branch
```

Comprueba ahora en el explorador de archivos que README.md es el original.

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_1. Introducción al trabajo con ramas

Para fusionar ramas, debes estar en la rama que va a recibir los cambios, así que asegúrate de estás en la principal:
git branch

Quieres traer los cambios de mejora-readme2, así que el flujo sería:
git merge mejora-readme2

Comprueba en tu explorador de archivos que los cambios de mejora-readme2 han sido incorporados en master.

Git no permite especificar ambas ramas en el comando, siempre operas desde una rama concreta. Pero si te interesa comparar ramas sin fusionarlas (por ejemplo, para revisar diferencias), puedes usar:

git diff master mejora-readme
git diff master...mejora-readme

Elimina la rama si ya no la necesitas:
git branch -d mejora-readme2

git branch

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_1. Introducción al trabajo con ramas

Tips para el día a día con ramas

git branch -av

te muestra:

Las ramas locales

Las ramas remotas

El último commit de cada rama (su hash abreviado y el mensaje)

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_2. Merge con conflicto (resolución manual)

Crear un entorno con conflicto, dentro de tu repositorio:

```
git checkout master  
"# Línea original" | Out-File -FilePath conflicto.txt -Encoding utf8  
echo "Línea original" > conflicto.txt  
git add conflicto.txt  
git commit -m 'Crear archivo base para conflicto'
```

Crear y cambiar a una nueva rama

```
git checkout -b rama-A
```

Edita el archivo conflicto.txt y cambia su contenido a:

Cambio desde la rama A

Haz commit:

```
git add conflicto.txt  
git commit -m 'Cambio desde rama A'
```

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_2. Merge con conflicto (resolución manual)

Vuelve a main y crea una nueva rama:

```
git checkout master  
git checkout -b rama-B
```

Edita conflicto.txt y cambia su contenido a (conflicto de contenido):

Cambio desde la rama B

Además, elimina un archivo antiguo (cambio en estructura):

```
rm notas.txt
```

Haz commit:

```
git add conflicto.txt notas.txt  
git commit -m 'Cambio desde rama B'
```

Vuelve a master y fusiona una de las ramas (por ejemplo, rama-A):

```
git checkout master  
git merge rama-A
```

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_2. Merge con conflicto (resolución manual)

Ahora intenta fusionar rama-B:

```
git merge rama-B
```

Comprueba cuáles son los conflictos:

```
git status
```

Verás como cambio el archivo borrado. Que no es conflictivo, pero puedes decidir que quieres conservarlo con:

```
git restore --staged notas.txt
```

Y verás un conflicto marcado como “both modified”.

Si quieres abortar y revisarlo con calma:

```
git merge --abort
```

solo funciona si no has finalizado el proceso aún.

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_2. Merge con conflicto (resolución manual)

Vete al explorador de archivos y abre conflicto.txt. Verás algo como esto:

```
<<<<< HEAD  
Cambio desde la rama A  
=====  
Cambio desde la rama B  
>>>>> rama-B
```

Para Resolver el conflicto de contenido, decide qué línea quieras conservar (o combina ambas), y elimina los marcadores de conflicto.

Guarda los cambios y confirma la resolución con:

```
git add conflicto.txt  
git commit -m 'Resolver conflicto de contenido entre A y B'
```

Repasa el historial de commits:

```
git log --oneline --graph
```

PRÁCTICA 2: Trabajo en un repositorio local

Ejercicio 2_2. Merge con conflicto (resolución manual)

extra: Resolver conflictos automáticamente

Extra: formas de Resolver Conflictos Automáticamente en Git

1. Cuando ya estás en medio de un conflicto, para aceptar la versión de la rama que estás fusionando:

```
git checkout --theirs archivo.txt
```

Para aceptar tu versión local (rama actual):

```
git checkout --ours archivo.txt
```

2. Antes del conflicto, usar estrategias de merge

```
git merge -X theirs rama      ## Aplica siempre los cambios de la rama que estás fusionando.
```

```
git merge -X ours rama       ## Aplica los cambios de la rama actual (descarta los de la otra rama).
```

Las resoluciones automáticas son útiles para conflictos simples o repetitivos, pero en conflictos más complejos, es mejor revisarlos manualmente.

Extras útiles

Añadir etiquetas (tags) para marcar versiones importantes en el historial del proyecto:

Puedes añadir una etiqueta a un commit (normalmente al último), con:

```
git tag v1.0
```

Ver todas las etiquetas del repositorio:

```
git tag
```

Para eliminarla localmente:

```
git tag -d v1.0
```

Los tags en git funcionan como marcadores permanentes en la historia del repositorio

Puedes usarlos para ver cómo estaba el proyecto en ese punto,

```
git show v1.0
```

volver temporalmente a una versión anterior (modo lectura),

```
git checkout v1.0
```

crear una nueva rama desde una versión antigua,

```
git checkout -b nueva-rama-desde-v1.0 v1.0
```

o volver atrás definitivamente.

```
git reset --hard v1.0
```

Fin PRÁCTICA 2

Buenas prácticas al trabajar con ramas

Todo lo que está en la **rama master** está listo para ser puesto en producción.

Para trabajar en algo nuevo, inestable o no testeado, se debe crear una nueva rama con un nombre descriptivo. Solo cuando creamos que la rama está lista debe integrarse en la rama master, y ponerse en producción.



: ¿Qué ventajas tiene usar ramas aunque estés trabajando solo?

- **Aíslas cambios experimentales o delicados:** puedes probar nuevas ideas sin miedo.
- **Organizas mejor tu trabajo:** y el historial del repositorio no es un caos.
- **Te ayuda a separar desarrollos:** evita mezclar cambios que no están relacionados.
- **Facilita volver atrás si algo falla:** si algo no funciona, simplemente no haces el merge.
- **Te prepara para colaborar con otros.**

Checkpoint Parte 2

Tenemos una cuenta en GitHub?

<https://docs.github.com/es/get-started/start-your-journey/creating-an-account-on-github>

Hemos generado y añadido todos un par de claves ssh?

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

Todos: `ssh-add -l`

Parte 2: repositorios remotos

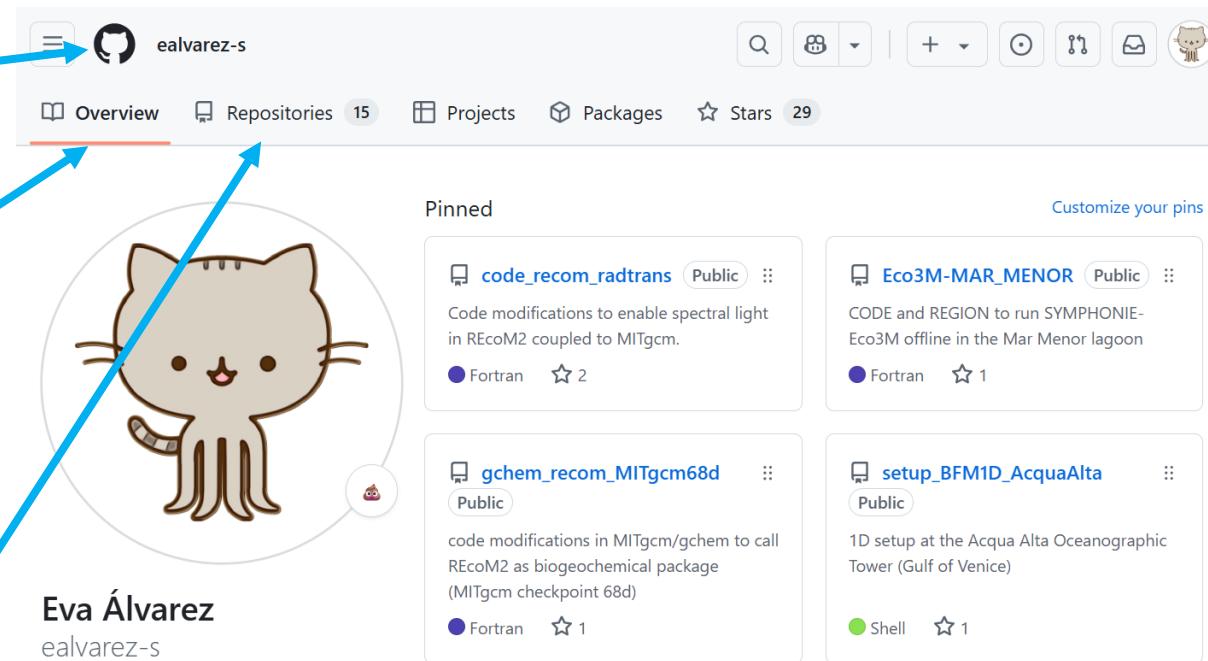


GitHub.com

Cuenta de usuario

1. **Crear una cuenta de usuario en GitHub:** el *username* será tu identificador para todo.
2. **Configurar *username* y *password* con autenticación en dos pasos (2FA) o *passkey*:** un segundo método de autenticación es obligatorio desde marzo de 2023.

Dashboard



Settings globales

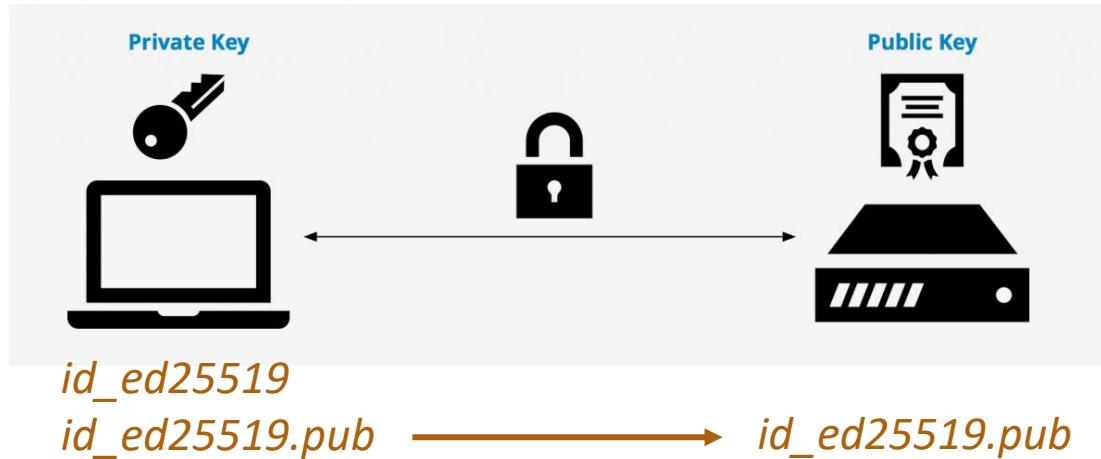
Tu perfil

Tus repositorios

GitHub.com

Clave ssh

Las claves ssh permiten a los usuarios acceder a un servidor remoto sin la necesidad de introducir la contraseña cada vez. Desde 2021, GitHub **no permite** autentificación solo con **usuario/contraseña**.



1. **Crear un par de claves (pública/privada) ssh y añadirla al agente** (procedimiento específico de SO y versión):

Buscad ***id_ed25519.pub*** en : /Users/tu_username/.ssh

GitHub.com

Clave ssh

2. Agrega la clave ssh pública a tu cuenta de GitHub (común para todos):

Copiad el contenido de la clave pública y añadidla a GitHub.

The screenshot shows the GitHub Settings interface. An orange arrow labeled '1' points to the 'Settings' icon in the top navigation bar. Another orange arrow labeled '2' points to the 'SSH and GPG keys' link in the left sidebar. A third orange arrow labeled '3' points to the 'New SSH key' button in the 'SSH keys' section of the main content area.

Eva Álvarez (ealvarez-s)
Your personal account

SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication keys

Key	SHA256	Added on	Last used	Action
g100_macAir	[REDACTED]	Added on Feb 28, 2024	Last used within the last 2 months — Read/write	Delete
clusterEO_macAir	[REDACTED]	Added on Apr 24, 2024	Last used within the last week — Read/write	Delete

New SSH key

PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

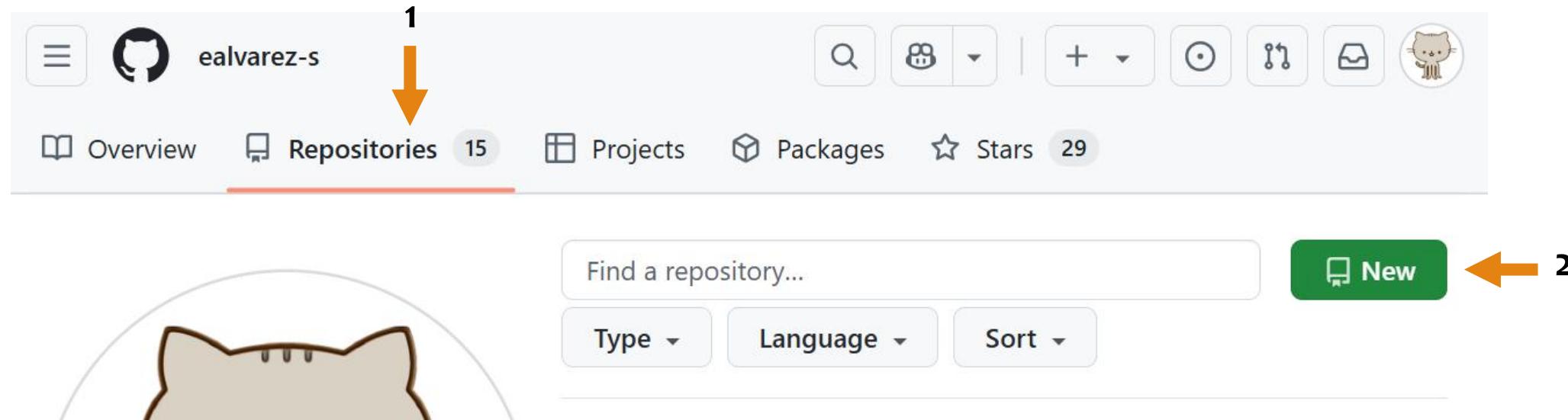
Ejercicio 3_1. Crear repositorio remoto y sincronizarlo con el local

Vete a GitHub.com, a **Repositories** y haz click en **New**.

Asigna un nombre a tu repositorio y una descripción corta.

Elige si quieres que sea público o privado (para este taller, cualquiera vale).

El resto déjalo como esta y dale a **Create repository**.



PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

Ejercicio 3_1. Crear repositorio remoto y sincronizarlo con el local

Muévete al directorio de trabajo:

```
cd 'Your directory'
```

y al repositorio que queremos sincronizar:

```
cd mi-primer-repo
```

y desde ahí sigue las instrucciones que te da GitHub:

```
git remote add origin git@github.com:username/reponame.git
```

```
git branch -M main
```

Comprueba la información de tu repositorio local con:

```
git config --list
```

Ya tienes los repos sincronizados.

PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

Ejercicio 3_2. Sincronizar cambios

Ahora ya podemos enviar nuestros archivos al repo remoto:

```
git push -u origin main
```

Modifica alguno de los ficheros en tu repo local y haz un commit siguiendo un flujo de trabajo básico.

Recuerda:

```
git add archivo modificado  
git commit -m 'mensaje'
```

Sube los cambios a GitHub:

```
git push
```

Flujo de sincronización

```
git add -> git commit -> git push
```

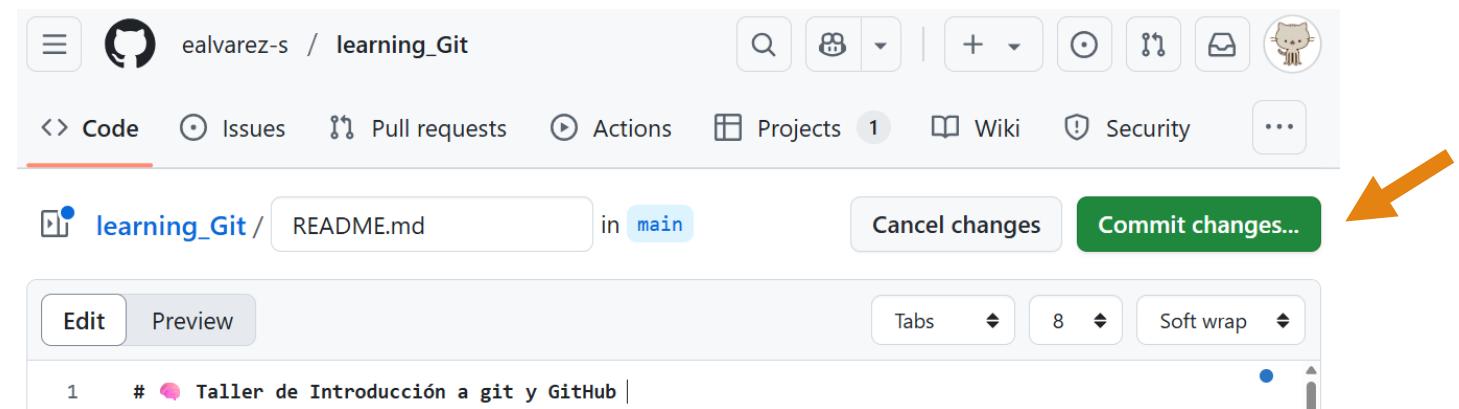
Ahora, ve a GitHub y verifica que tus cambios están allí.

PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

Ejercicio 3_2. Sincronizar cambios

Ahora vamos a ver cómo recuperar los cambios del repo remoto al repositorio local.

**## Ve al repositorio en GitHub. Abre el archivo README.md y haz clic en el lápiz  para editarlo directamente en la web.
Haz clic en Commit changes.**



**## Ahora vuelve a tu terminal y sincroniza los cambios remotos con tu copia local:
git pull**

Si abres el archivo modificado, verás los cambios que hiciste en GitHub reflejados localmente.

PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

Ejercicio 3_3. Trabajar con ramas remotas

Crea una nueva rama local para trabajar en una funcionalidad nueva. Recuerda:

```
git checkout -b nueva rama
```

Haz algún cambio simple en esta rama (por ejemplo, crea o edita un archivo).

Añade y haz commit como siempre:

```
git add archivo modificado  
git commit -m 'mensaje'
```

Sube tu nueva rama al remoto:

```
git push -u origin nueva rama
```

El -u configura la rama local para rastrear la rama remota con el mismo nombre.

Recordáis como comprobar las ramas existentes y en cual estamos?

```
git branch -av
```

Para eliminar una rama remota, puedes hacer:

```
git push origin --delete nueva rama
```

PRÁCTICA 3: Trabajo con tus repositorios propios (propietario)

Ejercicio 3_3. Trabajar con ramas remotas

Extra: añadir etiquetas (tags)

Puedes añadir una etiqueta a un commit (normalmente al último), con:

```
git tag v1.0
```

Una vez creada, súbelo a GitHub con:

```
git push origin v1.0
```

o si tienes varias etiquetas y quieres subirlas todas:

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

Ver todas las etiquetas del repositorio:

```
git tag
```

Para eliminarla localmente:

```
git tag -d v1.0
```

y en el remoto:

```
git push origin --delete tag v1.0
```

Fin PRÁCTICA 3

Buenas prácticas al trabajar con repositorios remotos

Archivos no necesarios pero muy útiles:

README.md

.gitignore

LICENSE

- Crear ramas con **prefijos descriptivos**: dev/, feature/, fix/, test/, etc.
- Commits claros**: usa mensajes breves pero descriptivos.
- Sincroniza a menudo: usa **git pull** con regularidad para evitar conflictos grandes.
- Asegúrate de que tus modificaciones **funcionan** antes de hacer push.
- Acuérdate de hacer **push después de commit**: no dejes commits colgados sin subir si estás colaborando.

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_1. Clonar un repositorio remoto (colaborativo)

Desde tu terminal, muévete al directorio de trabajo:

```
cd 'Your directory'
```

o si estás en mi-primer-repo, simplemente haz:

```
cd ../
```

Ejecuta:

```
git clone git@github.com:ealvarez-s/learning_Git.git
```

Si no has conseguido configurar la clave ssh en GitHub, usa HTTPS:

```
git clone https://github.com/ealvarez-s/learning_Git.git
```

Muévete a la carpeta recién creada:

```
cd learning_Git
```

Y comprueba la configuración del repositorio clonado. Recuerda:

```
git remote -v
```

```
git branch -av
```

```
git status
```

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

💡 ¿Por qué es mejor clonar un repositorio con git en lugar de simplemente descargarlo como .zip?

Porque **clonar** te conecta directamente con el historial y el flujo de trabajo del proyecto.
Al clonar:

- Te traes **todo el historial** del repositorio.
- Pueder recibir las **actualizaciones**.
- Puedes **contribuir** de vuelta al repositorio remoto si eres colaborador.
- Tienes acceso a las **funcionalidades** git.
- Es la forma estándar para **trabajar en equipo** y mantener la sincronización.

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_2. Trabajo colaborativo en un repositorio común

Extra: controlar la visibilidad y añadir colaboradores (Settings locales)

Para cambiar la visibilidad (público/privado):

⚙️ Settings → General → Danger Zone → "Change repository visibility"

Para añadir colaboradores:

⚙️ Settings → Collaborators and teams

The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with icons for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The Settings icon is highlighted with a red arrow labeled '1'. Below the navigation bar, the repository name 'ealvarez-s / learning_Git' is displayed. The main content area is titled 'Collaborators and teams'. A red arrow labeled '2' points to the 'Collaborators' tab, which is currently selected. Other tabs visible include 'General' and 'Access'. Under the 'Collaborators' tab, there's a section for 'Public repository' stating 'This repository is public and visible to anyone'. A 'Manage visibility' button is also present.

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_2. Trabajo colaborativo en un repositorio común

Voy a crear una carpeta llamada participantes/, **y dentro crearé el archivo** eva.txt:

```
mkdir participantes
";Hola desde Git!" | Out-File -FilePath participantes/eva.txt -Encoding utf8
echo ";Hola desde Git!" > participantes/eva.txt
git add participantes/eva.txt
git commit -m 'Añadir archivo de Eva en participantes'
git push
```

Haced:

```
git pull
```

y veréis como la nueva carpeta y el nuevo archivo aparece en vuestro repo local.

Copiad mi archivo con vuestro nombre, añadidlo y haced commit, si podéis con un mensaje que os identifique.

Haced:

```
git pull
git push
```

Si ahora echamos un ojo al repo remoto en GitHub, deberíamos ver vuestros archivos y vuestros commits.

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_2. Trabajo colaborativo en un repositorio común

Abrid el archivo compartido lista.txt y añadid vuestro nombre o un mensaje propio.

Añadidlo, haced commit y push como antes:

```
git add lista.txt  
git commit -m 'Fulanito añade su nombre a lista.txt'  
git pull  
git push
```

Hemos editado el mismo archivo (lista.txt) sin que se generen conflictos (aún).

En general, es buena idea coordinarse si varios vais a tocar lo mismo.

Si algo falla en el push, lee el mensaje del error: normalmente solo necesitas hacer git pull.

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_3. Resolver conflictos de manera manual y hacer Pull Requests

Editad todos la primera línea del archivo lista.txt e intentad hacer add, commit y push.

Todos los que no seáis el primero, recibiréis señales de conflicto.

Tenéis que hacer:

```
git pull
```

Y editar el archivo para dejar solo la versión correcta o combinarlas.

Guarda y confirma la resolución:

```
git add lista.txt
```

```
git commit -m 'Resolver conflicto en lista.txt'
```

```
git push
```

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_3. Resolver conflictos de manera manual y hacer Pull Requests

Primero, cada uno vamos a crear nuestra propia rama a partir de main:

```
git checkout -b nombre-usuario-cambio1
```

Haz algun cambio:

```
git add
```

```
git commit
```

Y sube tu rama al remoto:

```
git push origin nombre-usuario-cambio1
```

Veamos ahora las ramas disponibles en el repositorio, incluidas remotas y locales:

```
git branch -av
```

Para moverte a alguna de las ramas, recuerda:

```
git checkout remote_branch_name
```

PRÁCTICA 4: Trabajo con repositorios ajenos (colaborador)

Ejercicio 4_3. Resolver conflictos de manera manual y hacer Pull Requests

Si queremos que nuestros cambios se integren formalmente en la rama remota principal (main, develop, etc.), debemos abrir un Pull Request en GitHub.

- “Pull requests”

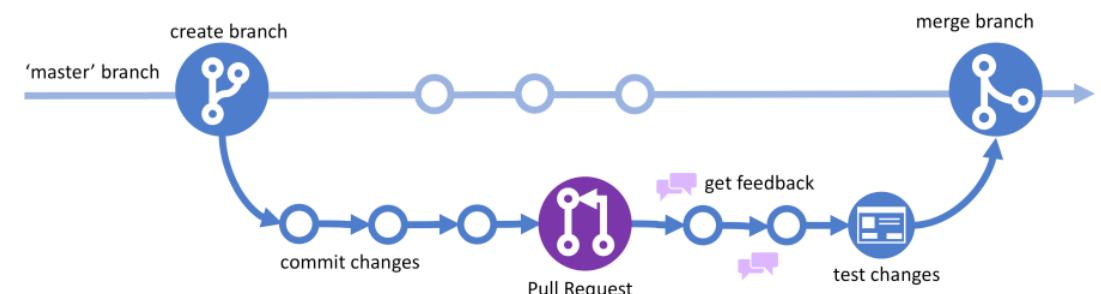
Escribe un título y descripción para tu PR.

Dale a **Create Pull Request**.

Yo o un/a compañer@ con permisos, lo revisara y hará:

- Merge pull request

The screenshot shows a GitHub repository interface for 'ealvarez-s / learning_Git'. The 'Pull requests' tab is highlighted with an orange arrow. At the bottom of the page, there is a yellow button labeled 'Create pull request' with an orange arrow pointing to it.



Fin PRÁCTICA 4

Extras útiles: proyectos, wiki, releases ...

Projects
Planificación y seguimiento del trabajo.

Wiki
Hoja de ruta, estado actual, documentación.

The screenshot shows a GitHub repository page for 'ealvarez-s / learning_Git'. The top navigation bar has tabs for Code, Issues, Pull requests, Actions, Projects (1), Wiki, and Security. Below the navigation is a card for the repository 'learning_Git' (Public). The main content area shows a table of recent commits:

Author	Commit Message	Date
ealvarez-s	test to check key	6748cdc · last week
	docs	exercices Part 4
	.gitignore	update .gitignore w...
	README.md	test to check key
	environment.yml	examples scripts
	list_my_files.R	examples scripts
	list_my_files.jl	examples scripts

To the right of the table is an 'About' section with the following text:
Hands-on training at the COG on the use of git and GitHub
Readme, Activity, 0 stars, 2 watching, 0 forks.
A blue arrow points from the 'Projects' text in the sidebar to the 'Projects' tab in the navigation bar. Another blue arrow points from the 'Wiki' text in the sidebar to the 'Wiki' tab in the navigation bar. A third blue arrow points from the 'Releases' text in the sidebar to the 'Releases' section in the repository details.

Releases
No releases published
[Create a new release](#)

Releases
Almacenar y citar repositorios (DOI con Zenodo).

fin

In case of fire



1. git commit



2. git push



3. leave building

Taller práctico 28-29 de mayo 2025