



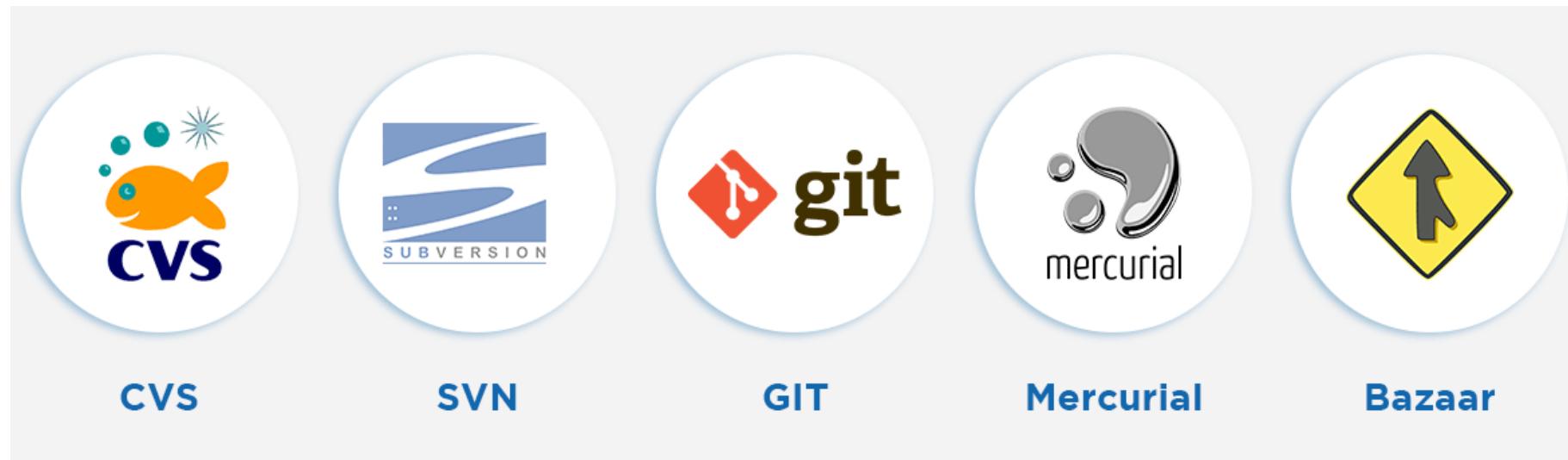
Tema 3. Git y Github

Comandos básicos para
control de versiones.

3.1 Definición de Sistema de Control de Versiones

Un sistema de control de versiones (o sistema de control de revisiones) es una combinación de tecnologías y prácticas para seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación, entre otros.

Los sistemas de control de versiones son un tipo de software que ayuda a hacer un seguimiento de los cambios realizados en el código a lo largo del tiempo. A medida que un desarrollador edita el código, el sistema de control de versiones toma una instantánea de los archivos. Después, guarda esa instantánea de forma permanente para que se pueda recuperar más adelante si es necesario.



Terminología

Repositorio ("repository")

El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios.

Revisión ("revision")

Una revisión es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador (Ej. subversion). Hay otros sistemas que identifican las revisiones mediante un código de detección de modificaciones (Ej. git usa SHA1).

Terminología

Etiqueta ("tag")

Los tags permiten identificar de forma fácil revisiones importantes en el proyecto. Por ejemplo, se suelen usar tags para identificar el contenido de las versiones publicadas del proyecto.

Rama ("branch")

Un conjunto de archivos puede ser ramificado o bifurcado en un punto en el tiempo de manera que, a partir de ese momento, dos copias de esos archivos se pueden desarrollar a velocidades diferentes o en formas diferentes de forma independiente el uno del otro.

Terminología

Cambio ("change")

Un cambio (o diff, o delta) representa una modificación específica de un documento bajo el control de versiones. La granularidad de la modificación que es considerada como un cambio varía entre los sistemas de control de versiones.

Desplegar ("checkout")

Es crear una copia de trabajo local desde el repositorio. Un usuario puede especificar una revisión en concreto u obtener la última. El término 'checkout' también se puede utilizar como un sustantivo para describir la copia de trabajo.

Terminología

Confirmar ("commit")

Confirmar es escribir o mezclar los cambios realizados en la copia de trabajo del repositorio. Los términos 'commit' y 'checkin' también se pueden utilizar como sustantivos para describir la nueva revisión que se crea como resultado de confirmar.

Conflicto ("conflict")

Un conflicto se produce cuando diferentes partes realizan cambios en el mismo documento (misma línea del código), y el sistema es incapaz de conciliar los cambios. Un usuario debe resolver el conflicto mediante la integración de los cambios, o mediante la selección de un cambio en favor del otro.

Terminología

Cabeza ("head")

También a veces se llama tip (punta) y se refiere a la última confirmación, ya sea en el tronco ('trunk') o en una rama ('branch'). El tronco y cada rama tienen su propia cabeza, aunque HEAD se utiliza a veces libremente para referirse al tronco.

Tronco ("trunk")

La única línea de desarrollo que no es una rama (a veces también llamada línea base, línea principal o máster).

Terminología

Fusionar, integrar, mezclar ("merge")

Una fusión o integración es una operación en la que se aplican dos tipos de cambios en un archivo o conjunto de archivos. Algunos escenarios de ejemplo son los siguientes:

- Un usuario, trabajando en un conjunto de archivos, actualiza o sincroniza su copia de trabajo con los cambios realizados y confirmados, por otros usuarios, en el repositorio.
- Un usuario intenta confirmar archivos que han sido actualizados por otros usuarios desde el último despliegue ('checkout'), y el software de control de versiones integra automáticamente los archivos (por lo general, después de preguntarle al usuario si se debe proceder con la integración automática, y en algunos casos sólo se hace si la fusión puede ser clara y razonablemente resuelta)

Terminología

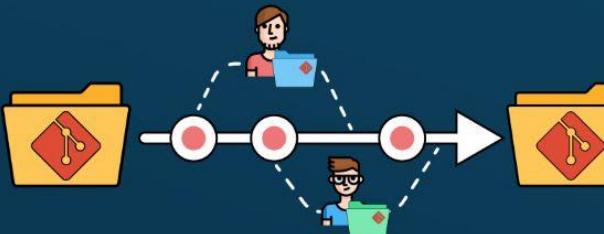
- Un conjunto de archivos (repositorio) se bifurca, un problema que existía antes de la ramificación se trabaja en una nueva rama, y la solución se combina luego en la otra rama.
- Se crea una rama, el código de los archivos es editado independientemente, y la rama actualizada se incorpora más tarde en un único tronco unificado.

Introducción a git

Git es un sistema de control de versiones distribuido que se diferencia del resto en el modo en que modela sus datos.

¿SABES QUÉ ES GIT?

Es un sistema de **control de versiones**. Lleva un registro de todos los **cambios y avances** de tu proyecto.



GIT TRABAJA CON RAMAS

Ayuda a que **varias personas trabajen** en un mismo proyecto y pueden realizar **modificaciones sin afectar a los demás** archivos. Una vez que estén listos los cambios se **fusionan con la rama principal**.



v1 v2 v3

Funciona como una máquina de tiempo, puedes ir al pasado de tu código o volver al presente.



Github es un servicio que te ayuda a **almacenar tu proyecto en la nube**, además existen otros servicios como **Gitlab** o **Bitbucket**.



Los tres estados.

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en tu base de datos local.

Modificado significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos.

Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

Los tres estados.

Directorio de trabajo (**Working directory**)

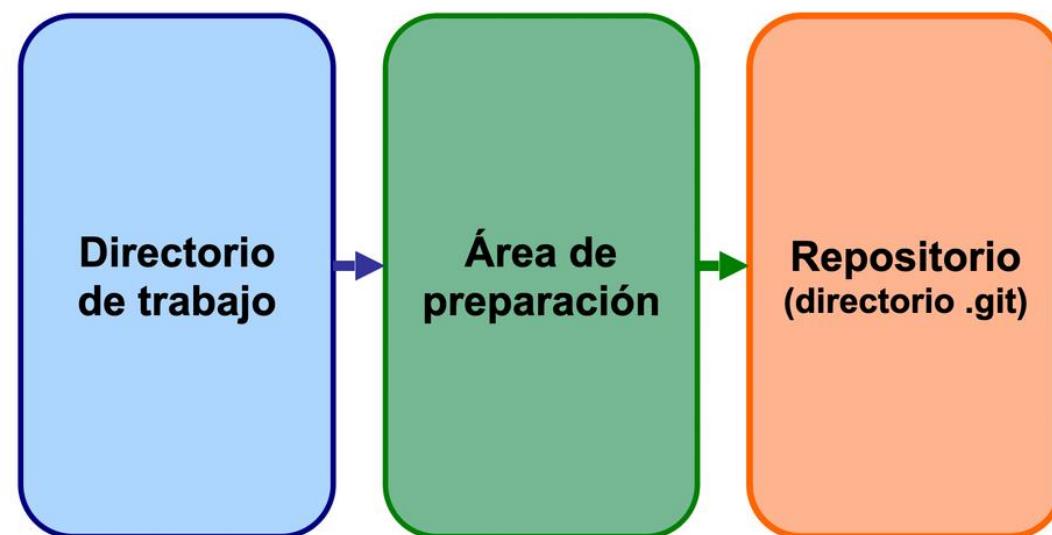
Este es el directorio del proyecto en el sistema de archivos, donde se guardan los archivos. Esta es, por ejemplo, la carpeta que abres en tu editor de código o IDE para trabajar con tus archivos.

Área de preparación (**Staging area**)

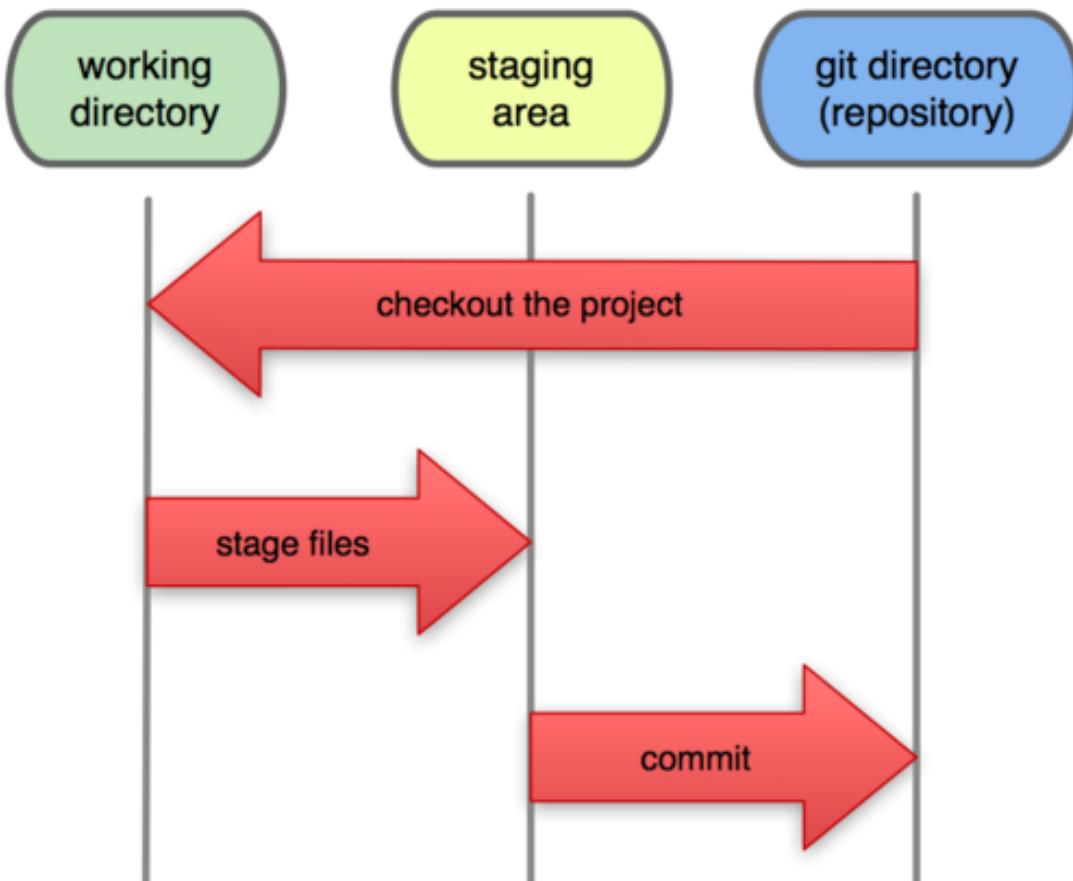
Este es el conjunto de archivos y cambios que serán incluidos en el siguiente commit. Podemos agregar y remover archivos de esta área si es necesario.

Repositorio (**Repository**)

Un repositorio es donde Git almacena los archivos de tu proyecto y las distintas versiones de tus archivos. Un repositorio puede ser local o remoto. Un repositorio local se guarda de forma local en tu computadora. Un repositorio remoto se guarda en los servidores del servicio de hosting que escojas (por ejemplo, GitHub).



Local Operations



COMANDOS BÁSICOS DE



GIT

GIT es un sistema
de control de
versiones

GIT CLONE



Clona un repositorio en un nuevo directorio.

GIT ADD



Agrega archivos al staged (staging area - index).

GIT COMMIT



Registra los cambios en el repositorio.

GIT STATUS



Muestra el estado actual del repositorio.

GIT PUSH



Actualiza las referencias remotas junto con los objetos asociados

GIT PULL

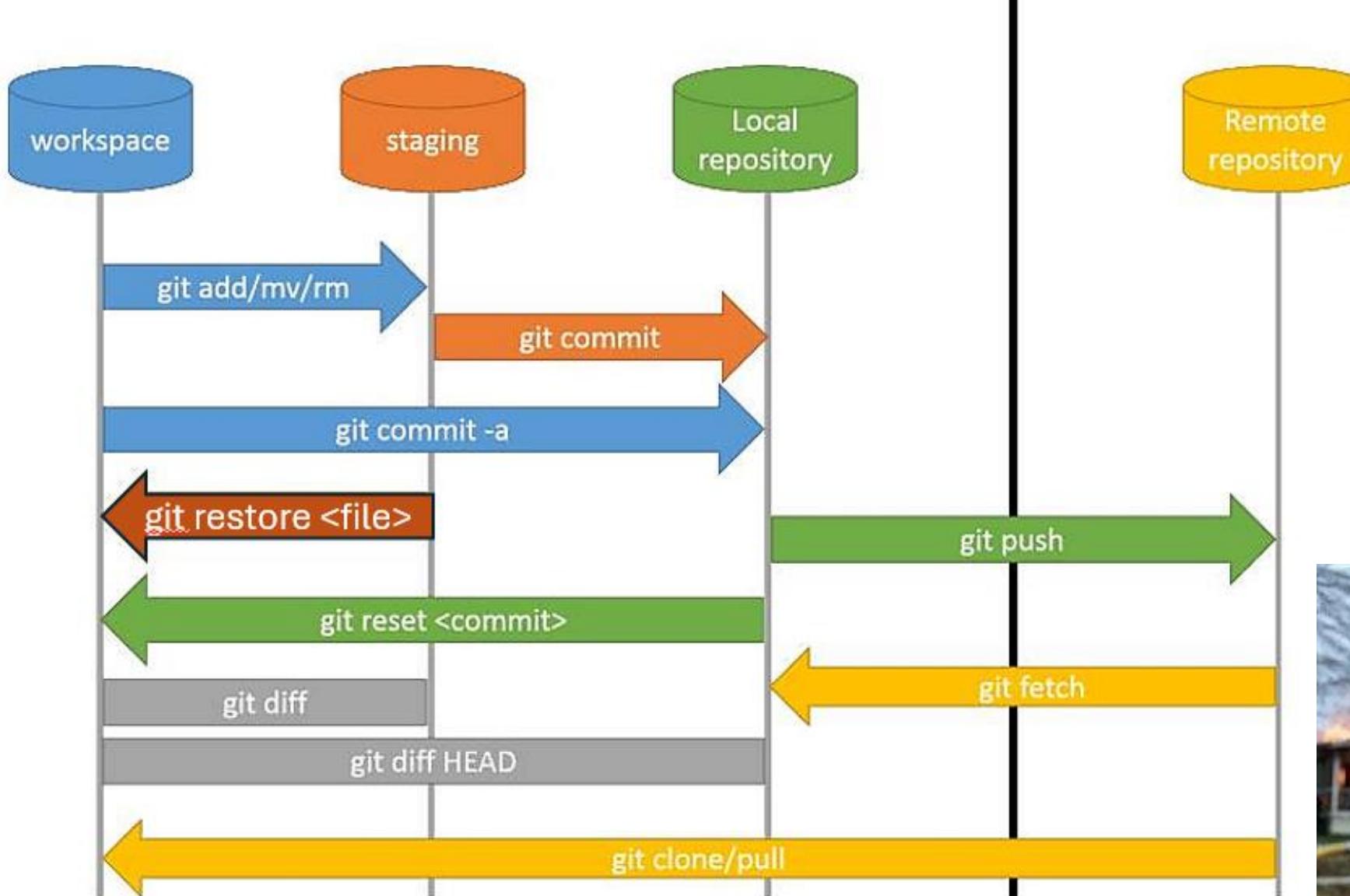


Actualiza las referencias locales junto con los objetos asociados

GIT CHECKOUT



Cambia de rama o restaura archivos del espacio de trabajo.

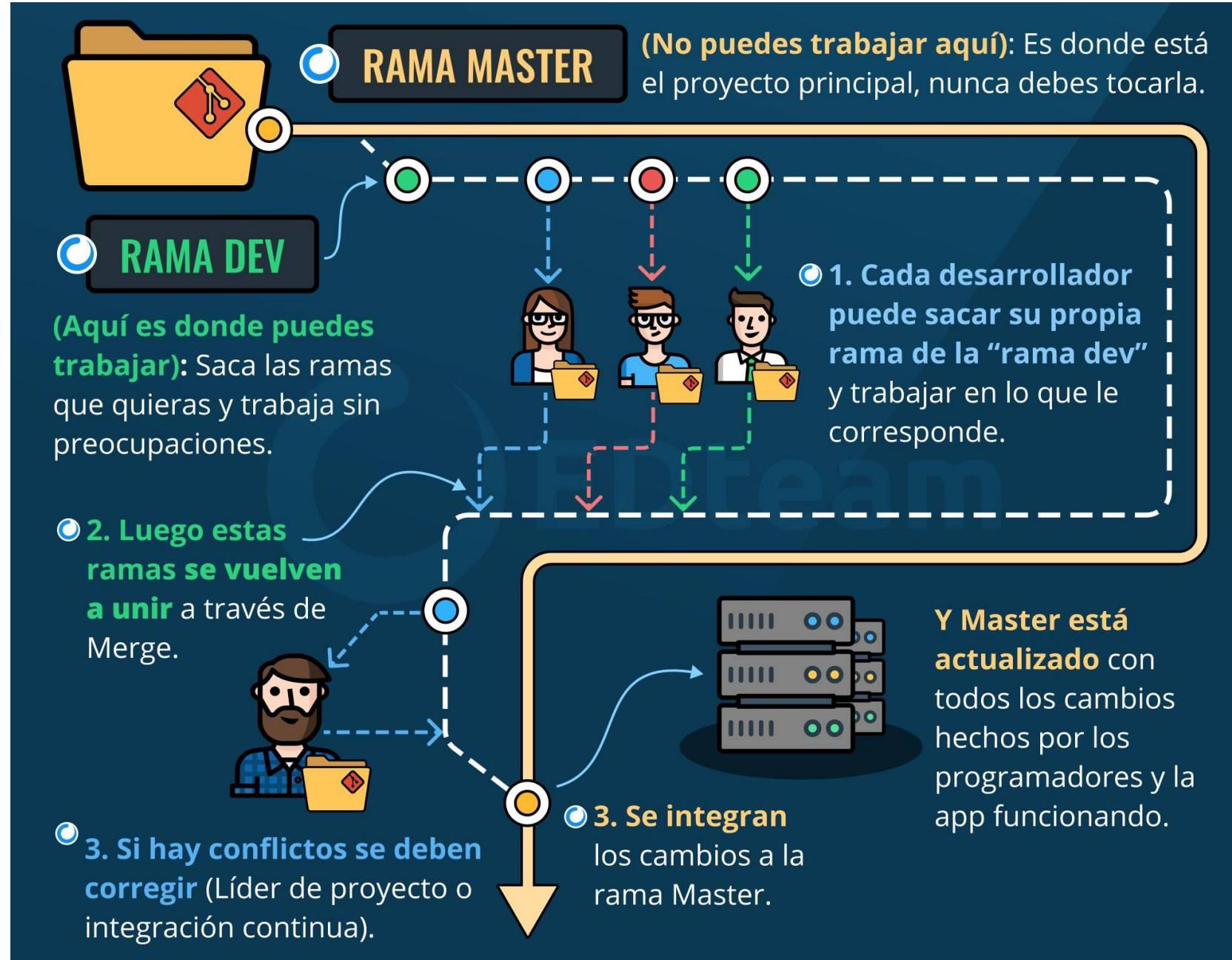


PUSH REJECTED, REBASE OR
MERGE



GIT PUSH --FORCE

¿Cómo trabajar en equipo con Git?





git

3.2 Aspectos básicos de git

3.2.1 Instalación

Desde la web oficial de git (<https://git-scm.com/>) puedes obtener las instrucciones para instalarlo en las diferentes distribuciones de Linux, así mismo podrás encontrar el instalador para Windows. En macOs ya viene instalado de serie, aunque desde la web oficial podrás bajar la última versión.



Página principal principal (home) de git



git --local-branching-on-the-cheap

Search entire site...

Git is a [free and open source](#) distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is [easy to learn](#) and has a [tiny footprint with lightning fast performance](#). It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like [cheap local branching](#), convenient staging areas, and [multiple workflows](#).



 **About**
The advantages of Git compared to other source control systems.

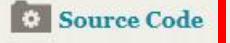
 **Documentation**
Command reference pages, Pro Git book content, videos and other material.

 **Downloads**
GUI clients and binary releases for all major platforms.

 **Community**
Get involved! Bug reporting, mailing list, chat, development and more.

 **Latest source Release**
2.44.0
[Release Notes \(2024-02-23\)](#)
[Download for Windows](#)

 [Windows GUIs](#)  [Tarballs](#)

 [Mac Build](#)  [Source Code](#)

 [Pro Git](#) by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Companies & Projects Using Git

Google Microsoft  LinkedIn  NETFLIX

 PostgreSQL  

Libro en Español de git

git --distributed-even-if-your-workflow-isnt

Search entire site...

About

Documentation

[Reference](#)

[Book](#)

[Videos](#)

[External Links](#)

Downloads

Community

This book is available in [English](#).

Full translation available in

[azərbaycan dili](#),

[български език](#),

[Deutsch](#),

[Español](#),

[Français](#),

[Ελληνικά](#),

[日本語](#),

[한국어](#),

[Nederlands](#),

[Русский](#),

[Slovenščina](#),

[Tagalog](#),

[Українська](#)

[简体中文](#),

Partial translations available in

[Čeština](#),

[Македонски](#),

[Polski](#),

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).

2nd Edition (2014)

The version found here has been updated with corrections and additions from [hundreds of contributors](#). If you see an error or have a suggestion, patches and issues are welcome in its [GitHub repository](#).

1. Inicio - Sobre el Control de Versiones

- [1.1 Acerca del Control de Versiones](#)
- [1.2 Una breve historia de Git](#)
- [1.3 Fundamentos de Git](#)
- [1.4 La Línea de Comandos](#)
- [1.5 Instalación de Git](#)
- [1.6 Configurando Git por primera vez](#)
- [1.7 ¿Cómo obtener ayuda?](#)
- [1.8 Resumen](#)

2. Fundamentos de Git

- [2.1 Obteniendo un repositorio Git](#)
- [2.2 Guardando cambios en el Repositorio](#)
- [2.3 Ver el Historial de Confirmaciones](#)
- [2.4 Deshacer Cosas](#)
- [2.5 Trabajar con Remotos](#)
- [2.6 Etiquetado](#)
- [2.7 Alias de Git](#)
- [2.8 Resumen](#)

Download Ebook

pdf

epub

Descargas de git



[About](#)

[Documentation](#)

[Downloads](#)

GUI Clients

Logos

[Community](#)

The entire [Pro Git book](#) written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Downloads



macOS



Windows



Linux/Unix

[Older releases](#) are available and the [Git source repository](#) is on GitHub.

GUI Clients

Git comes with built-in GUI tools ([git-gui](#), [gitk](#)), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

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

You can also always browse the current contents of the git repository using the [web interface](#).

Latest source Release

2.44.0

[Release Notes \(2024-02-23\)](#)

[Download for Windows](#)

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

Vamos a descargar git para Windows

git --distributed-is-the-new-centralized

[About](#) [Documentation](#) [Downloads](#) [Community](#)

The entire [Pro Git book](#) written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Latest source Release
2.44.0
Release Notes (2023-02-23)

Download for Windows

macOS Windows Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.

GUI Clients

Git comes with built-in GUI tools ([git-gui](#), [gitk](#)), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

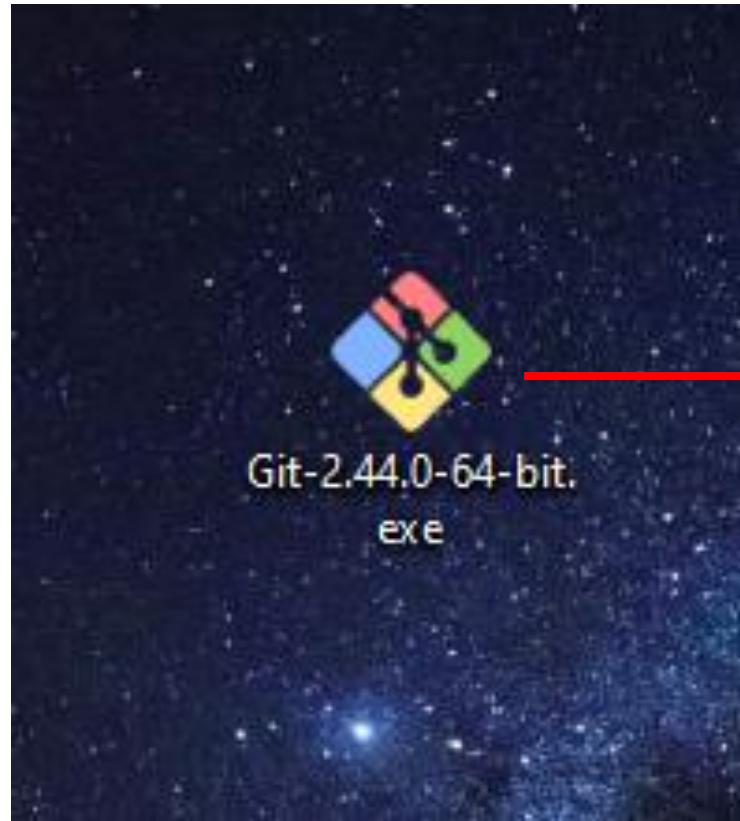
Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

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

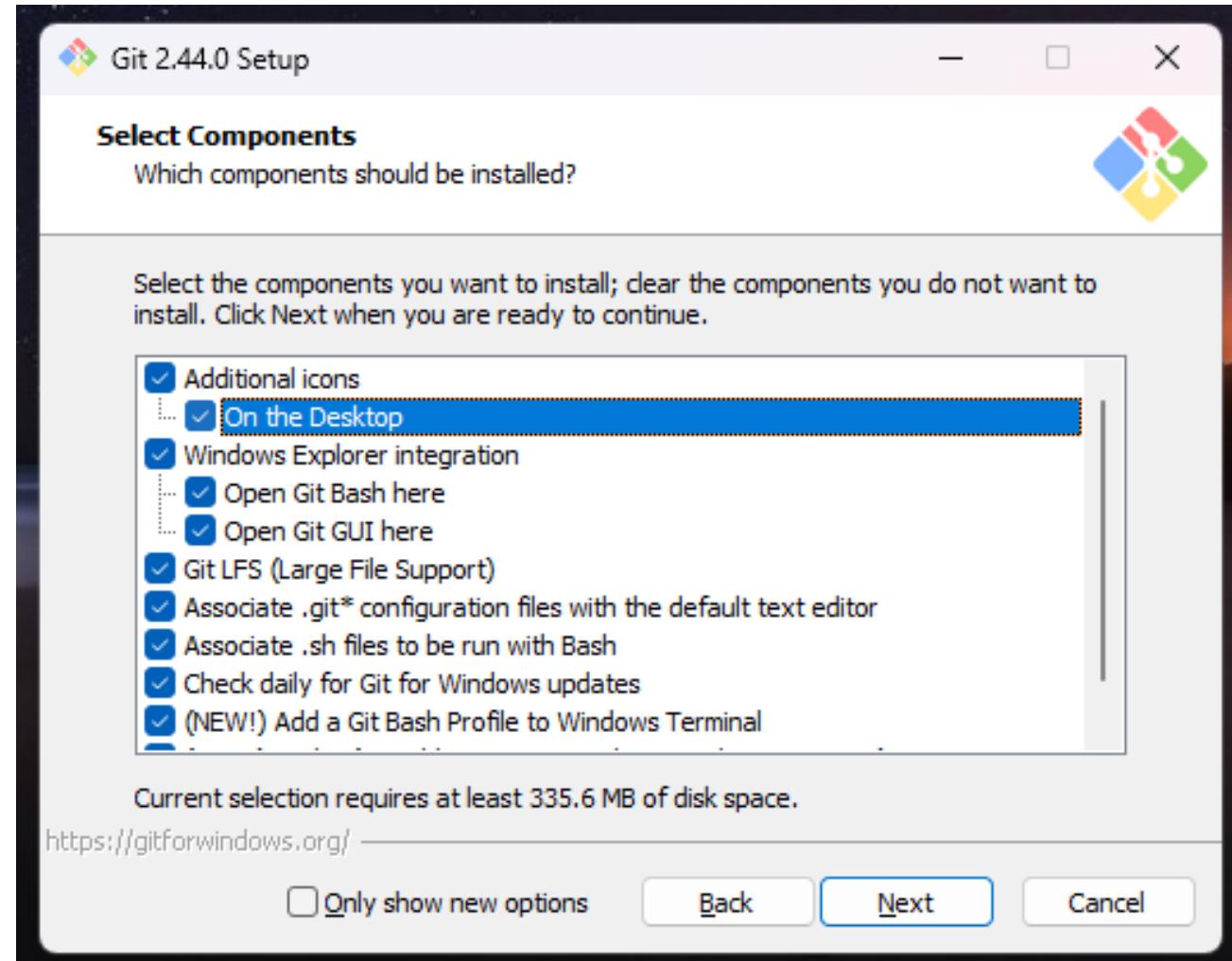
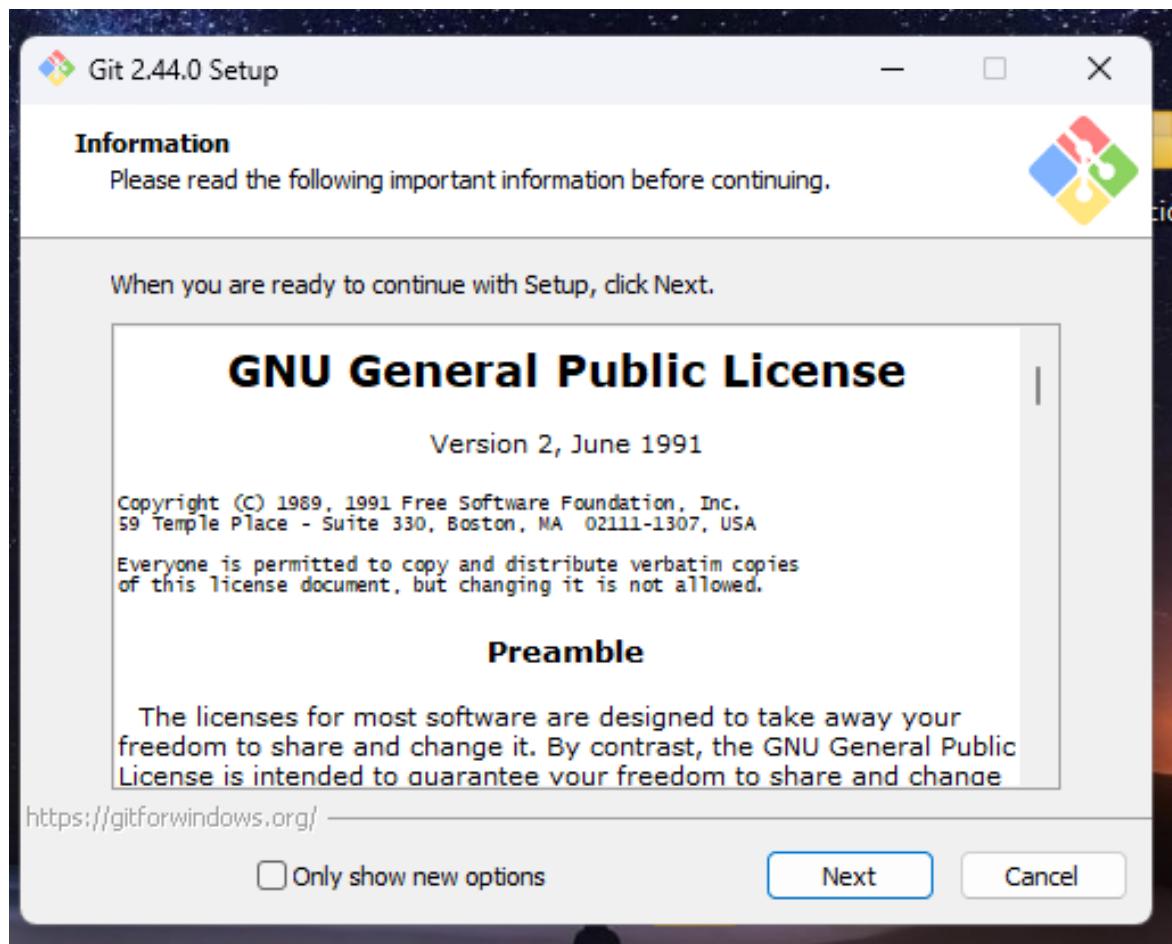
You can also always browse the current contents of the git repository using the [web interface](#).

Una vez descargado el ejecutable , lo instalamos como administrador:

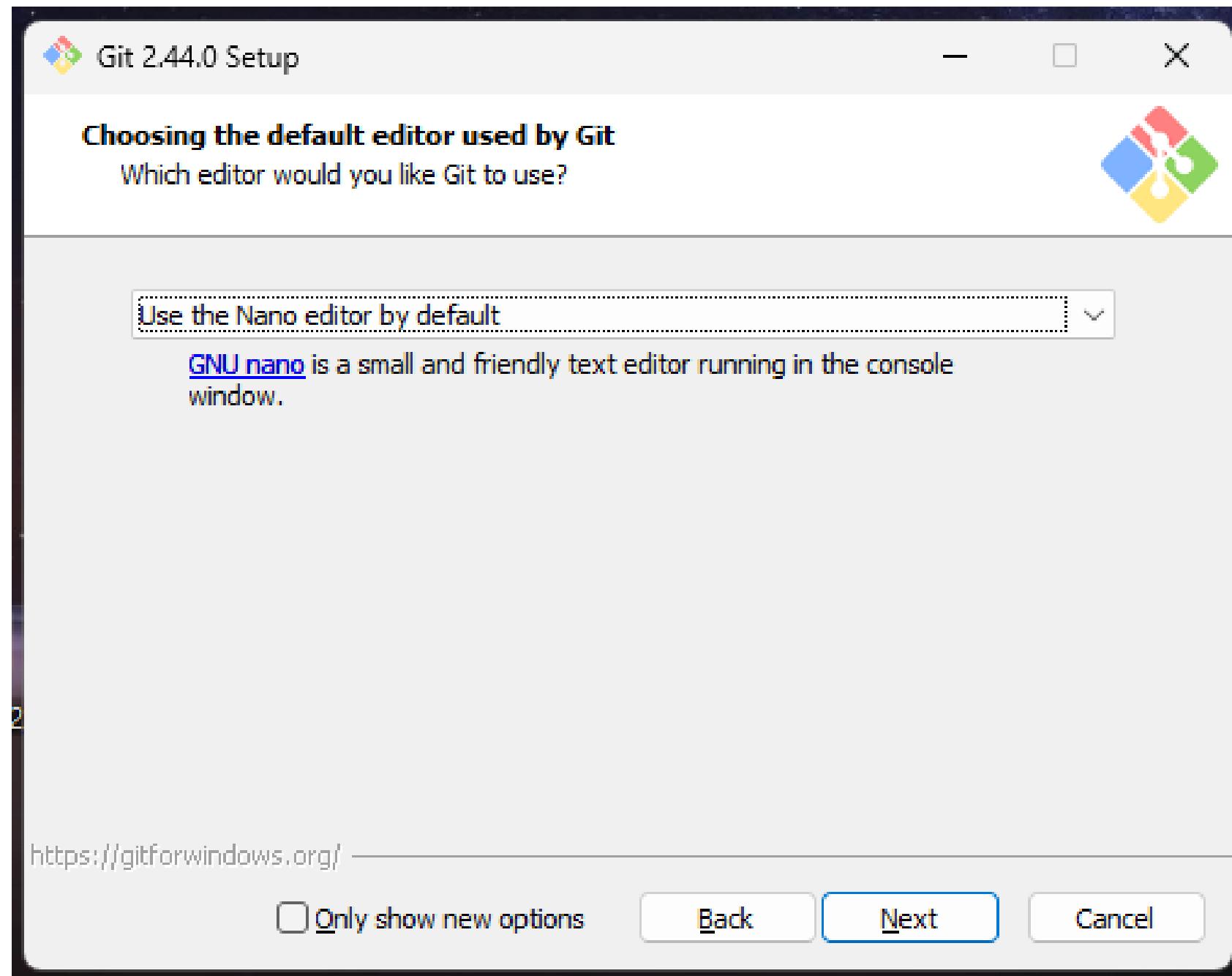


Click con el botón derecho del ratón sobre el ícono, luego:
Ejecutar como administrador

Next, luego activar todas las casillas después Next



**Seleccionar:
Use the Nano
editor by
default, luego
click en Next**



Seleccionar:
Let Git
decide, luego
click en Next

Git 2.44.0 Setup

Adjusting the name of the initial branch in new repositories

What would you like Git to name the initial branch after "git init"?

Let Git decide

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project [intends](#) to change this default to a more inclusive name in the near future.

Override the default branch name for new repositories

NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

main

This setting does not affect existing repositories.

<https://gitforwindows.org/>

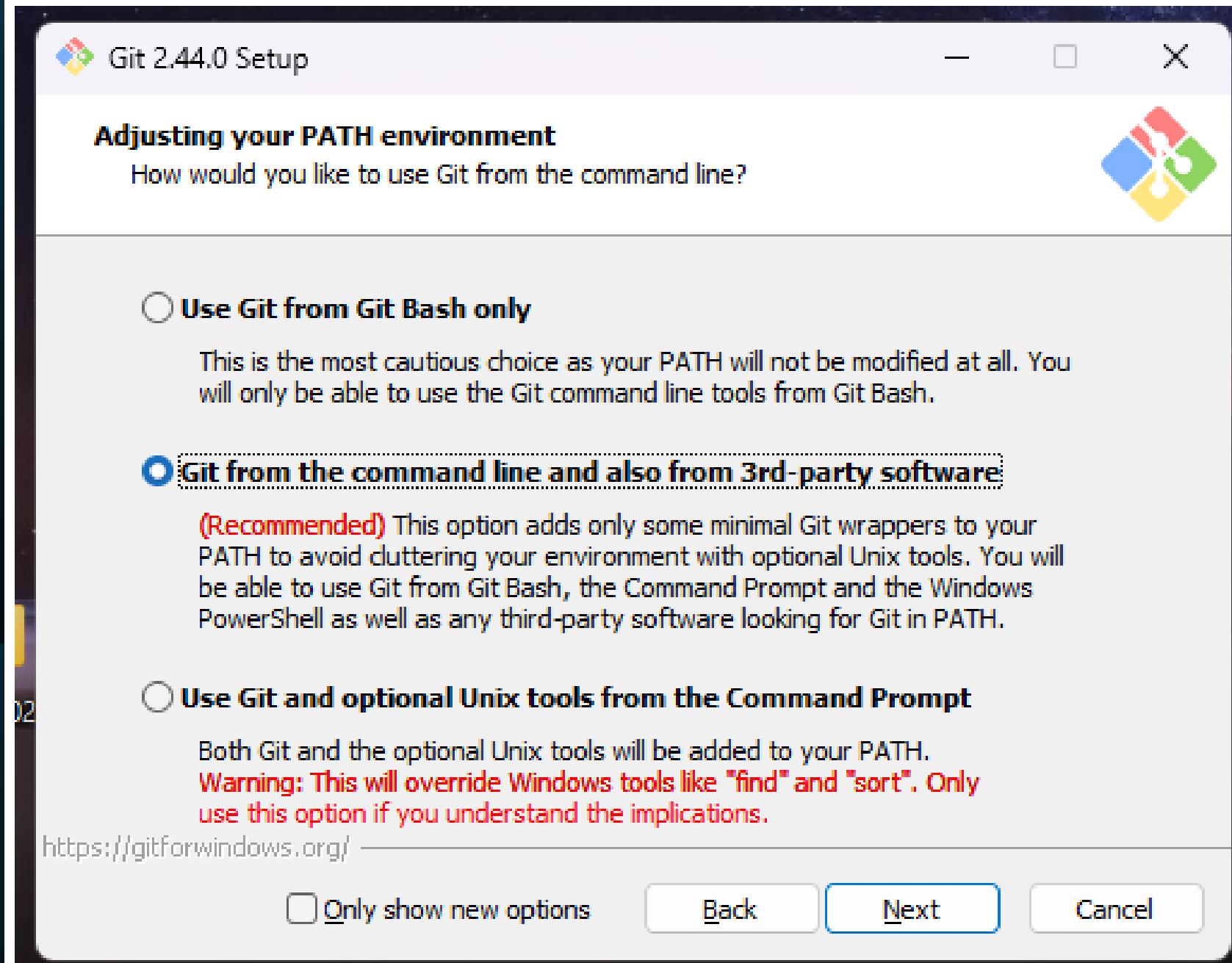
Only show new options

Back

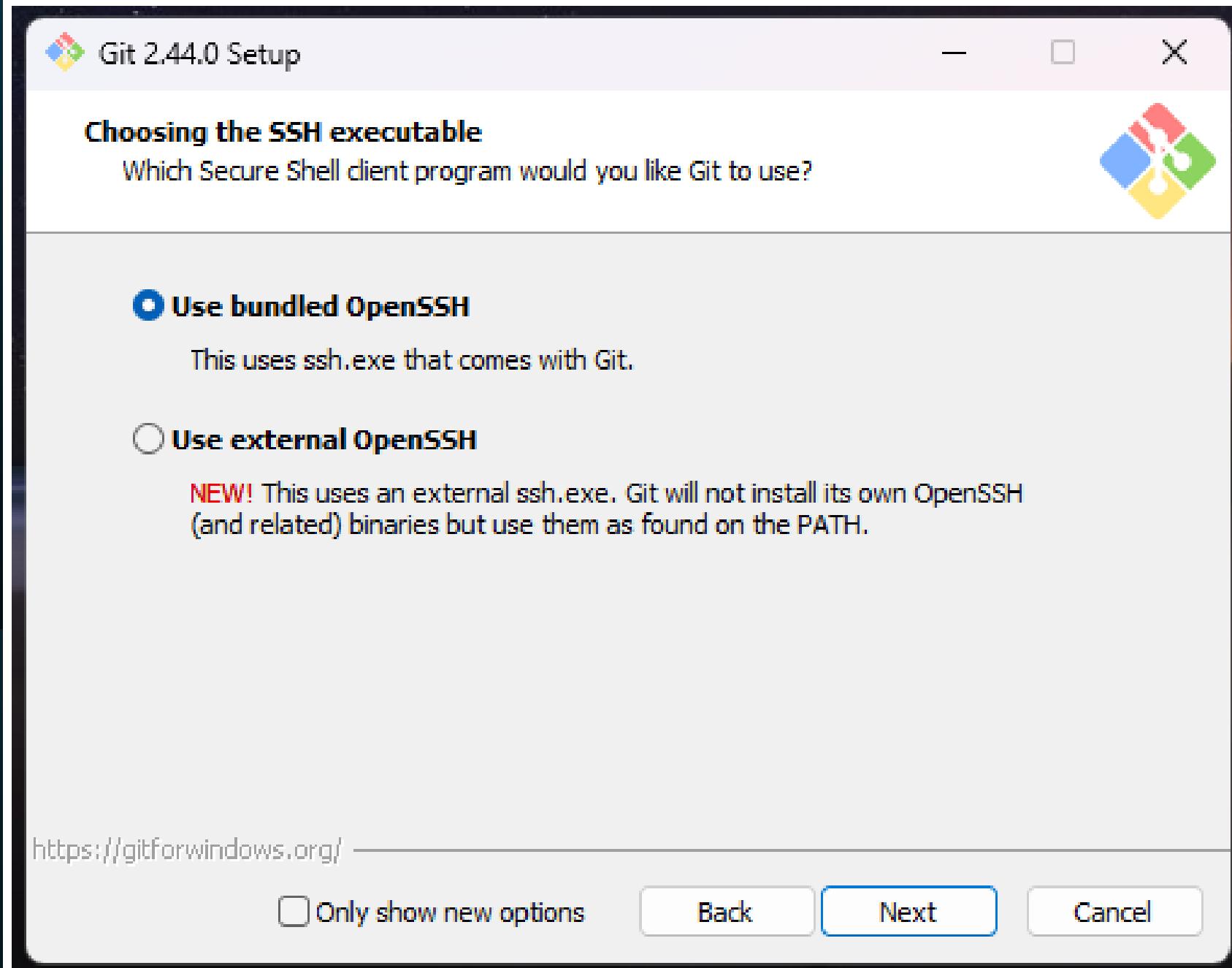
Next

Cancel

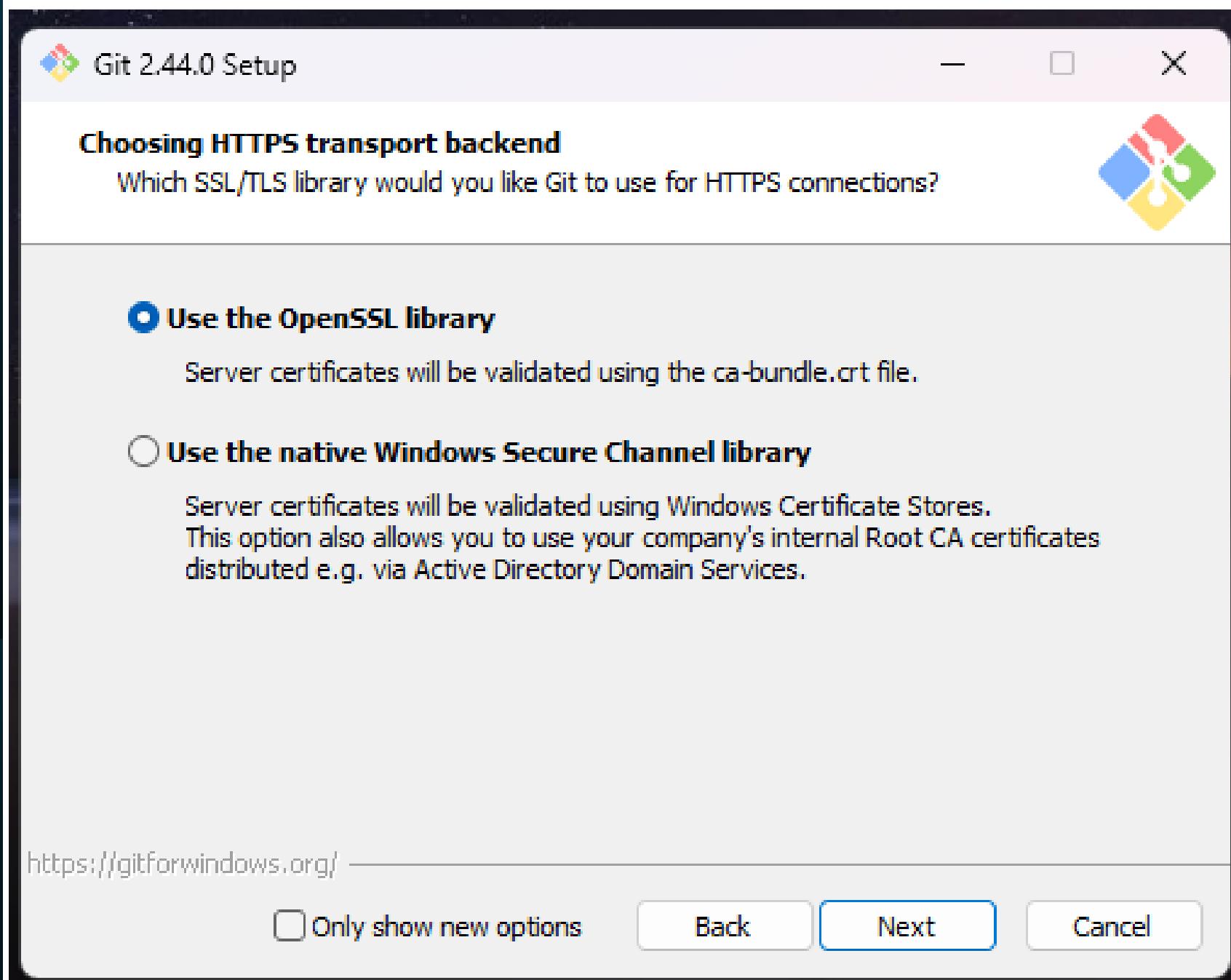
Seleccionar: **Git from the command line and also from 3rd-party software**, luego click en Next



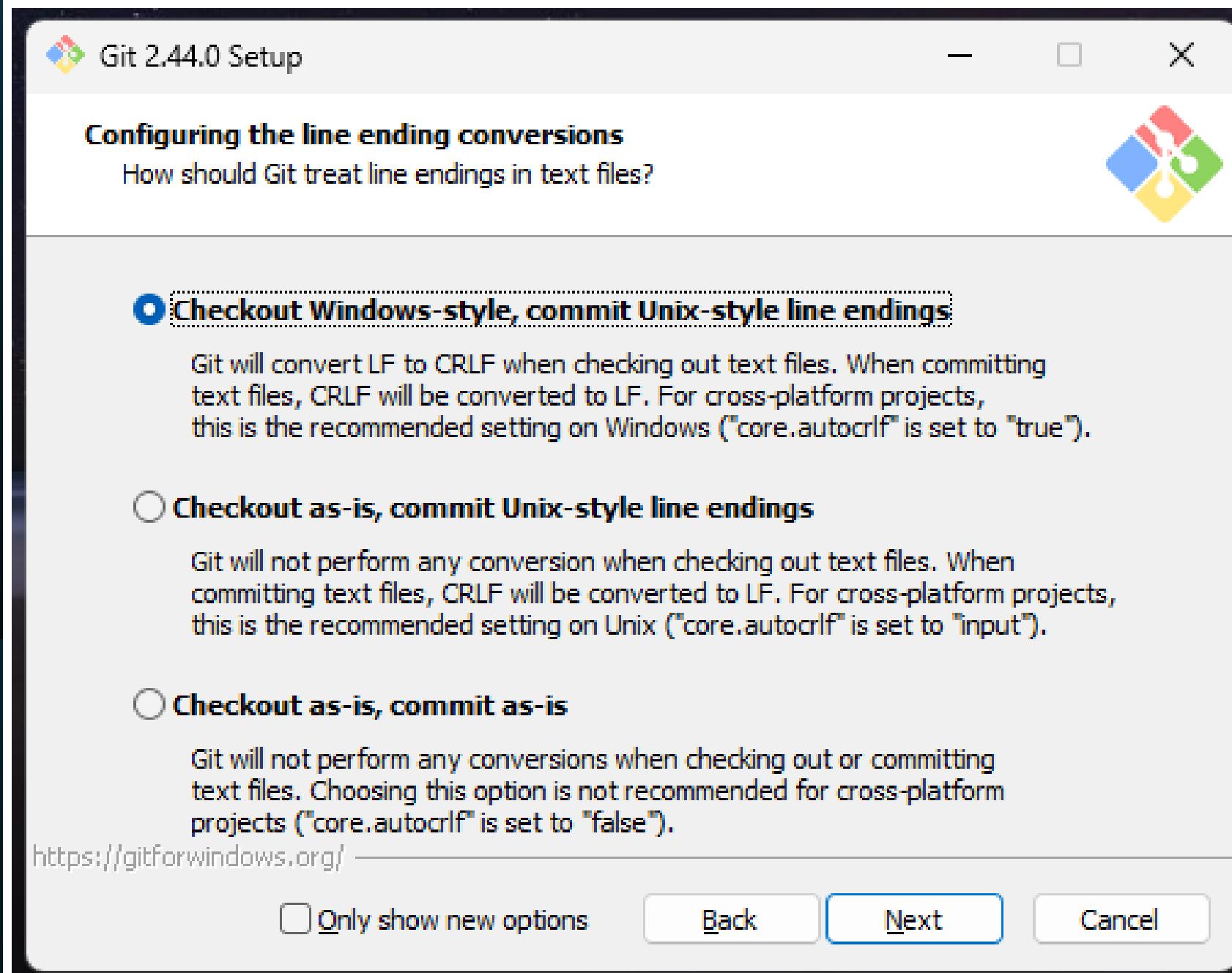
Seleccionar:
Use bundled
OpenSSH,
luego click en
Next



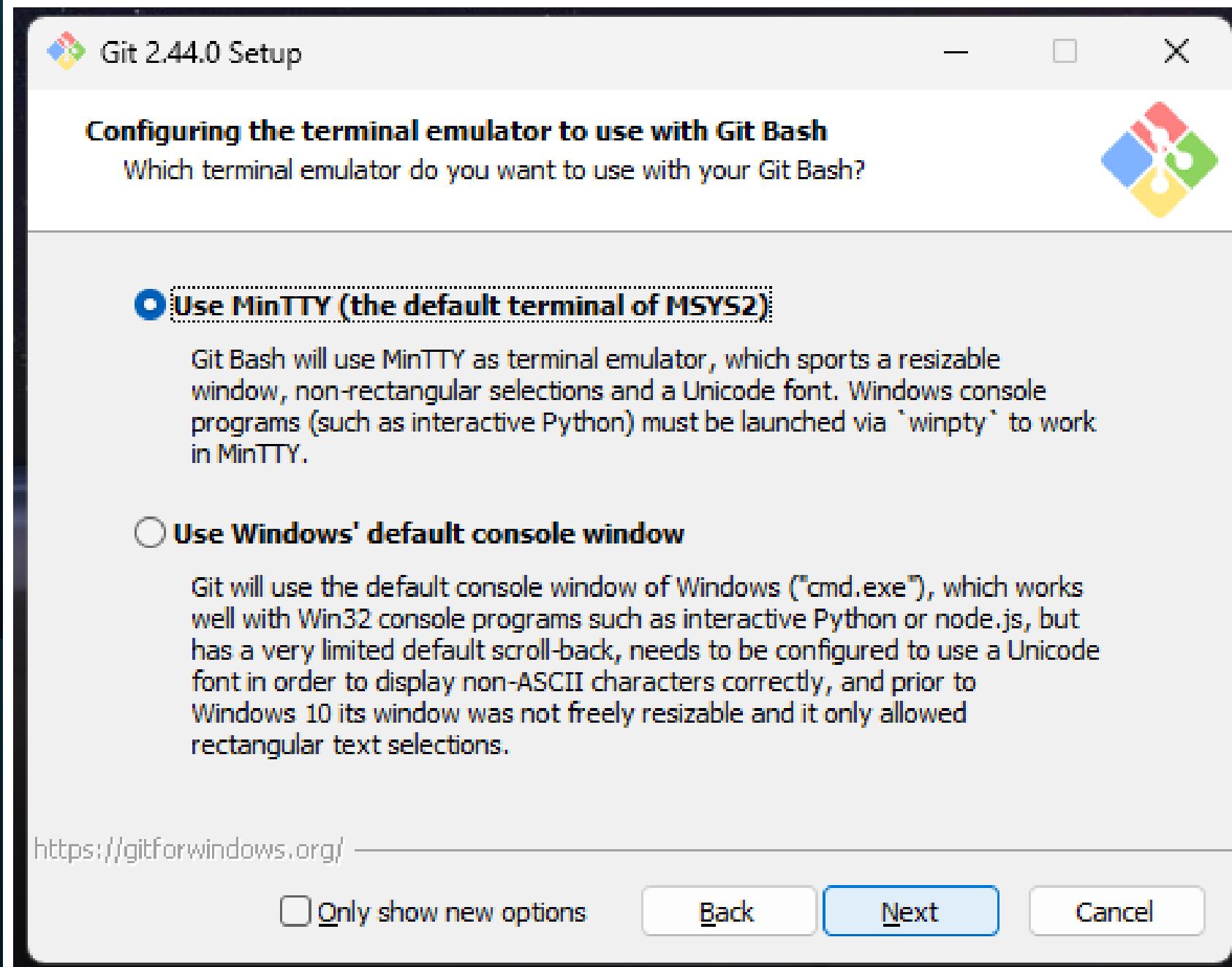
Seleccionar:
Use the
OpenSSL
library, luego
click en Next



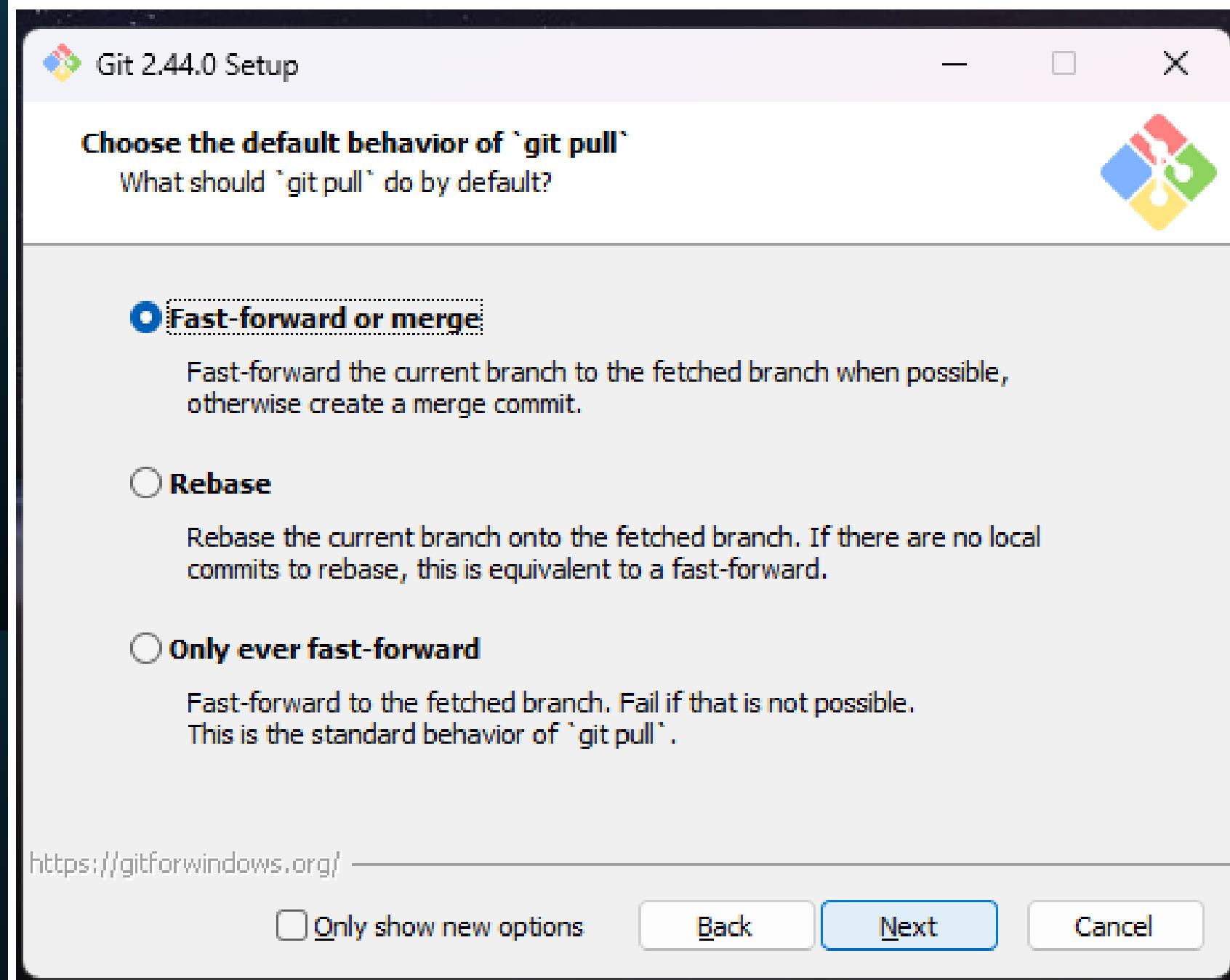
Seleccionar:
Checkout
Windows-
style, commit
Unix-style line
endings, luego
click en **Next**



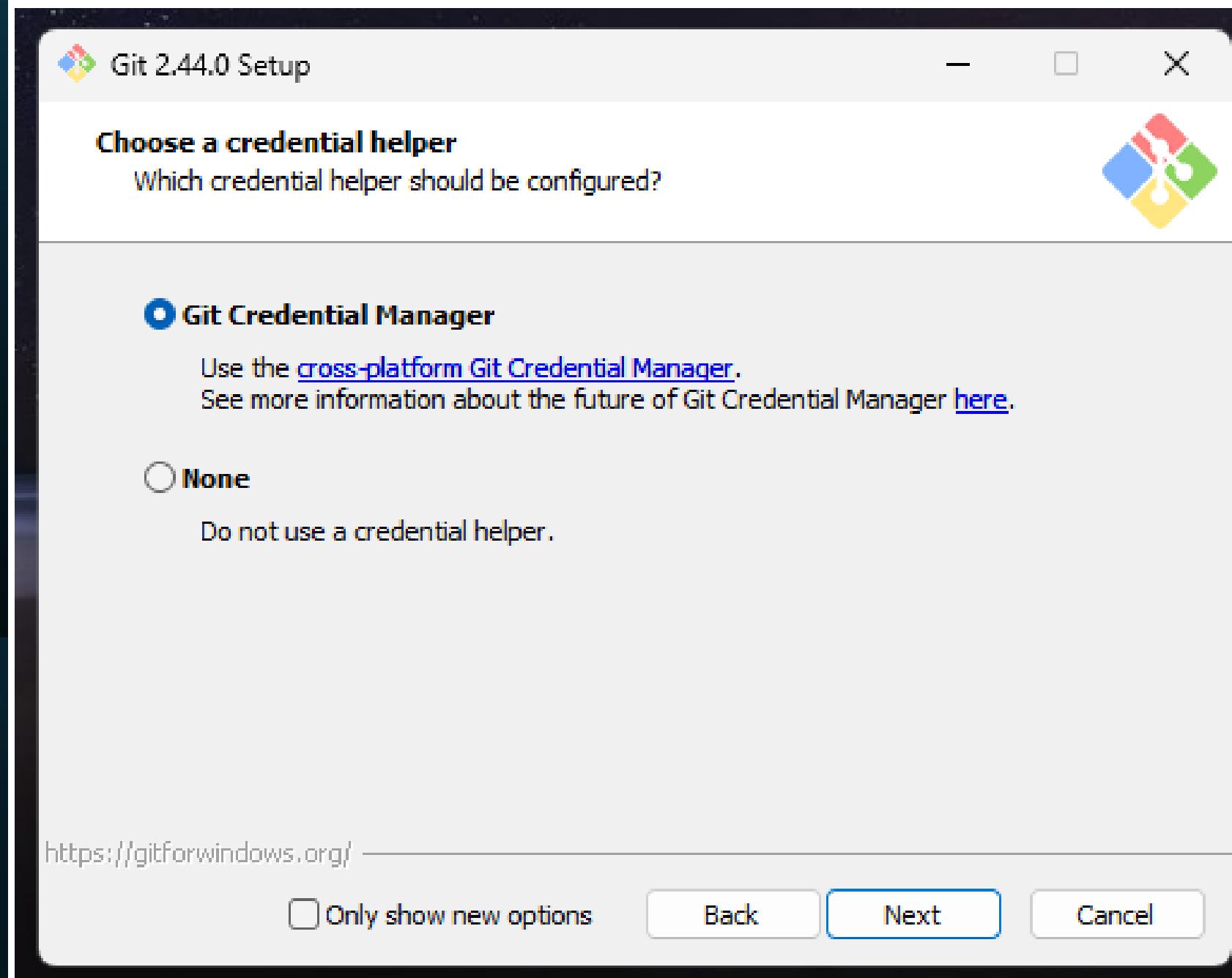
Seleccionar:
Use MinTTY
(the default
terminal of
MSYS2), luego
click en **Next**



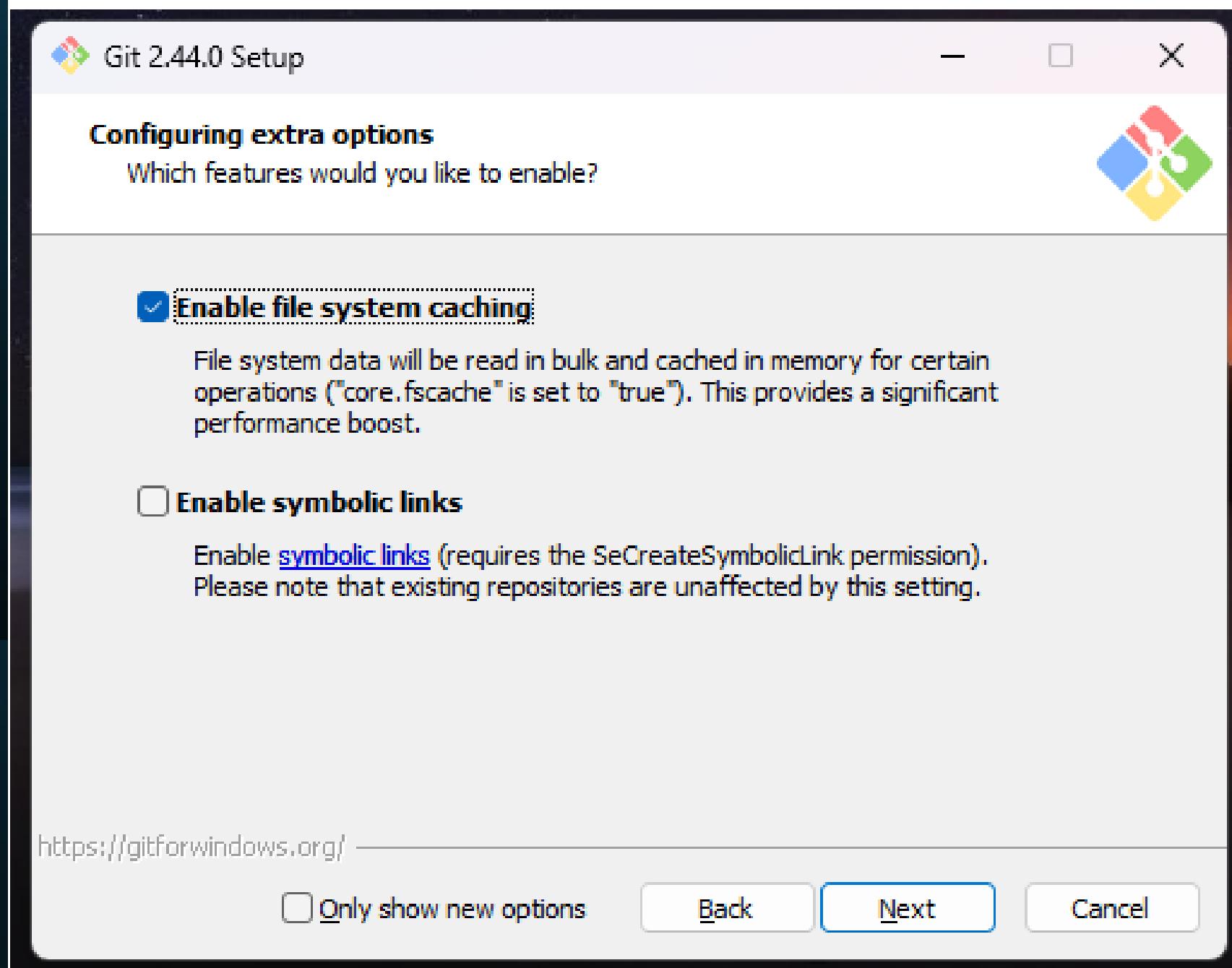
Seleccionar:
Fast-forward
or merge, luego
click en **Next**



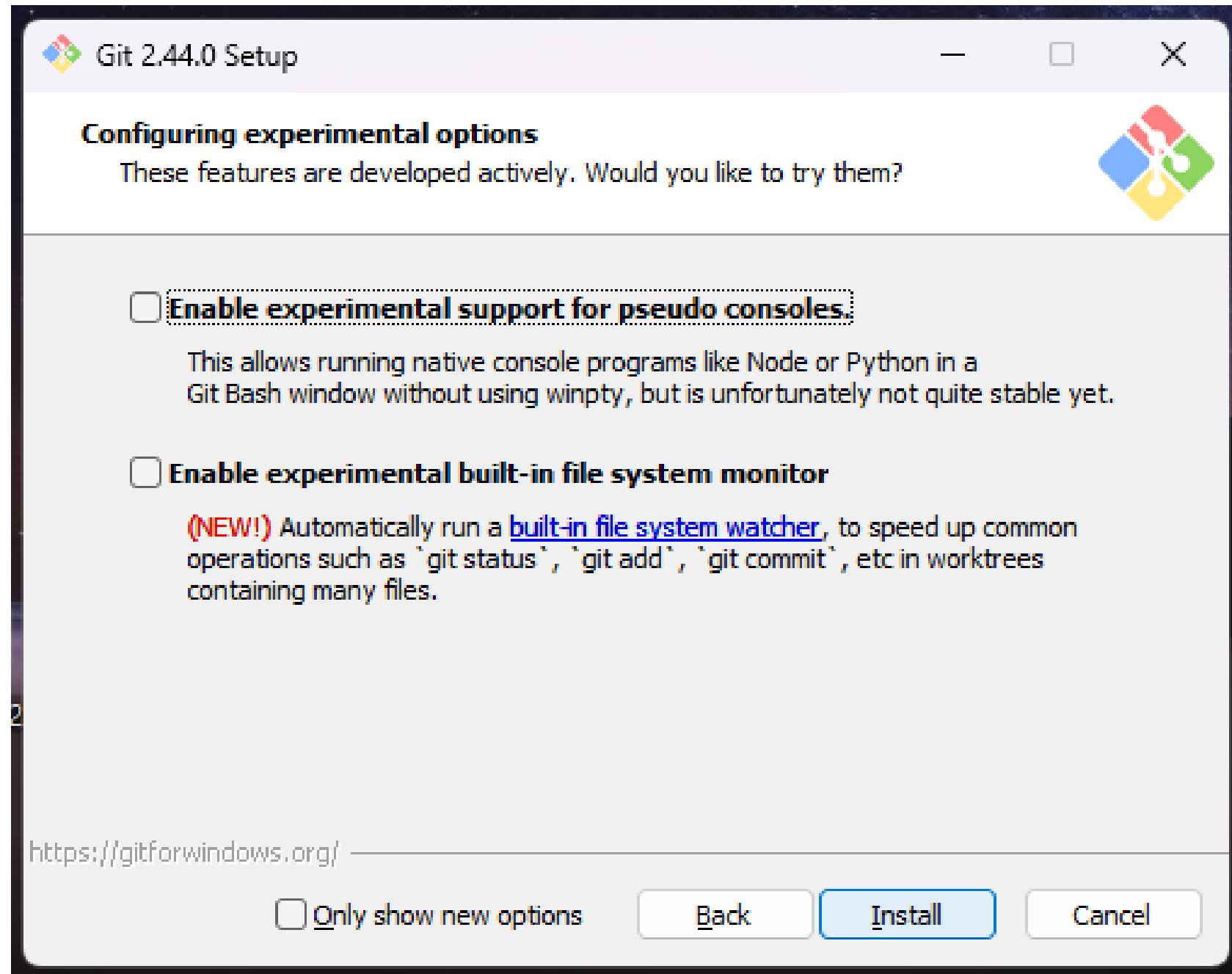
Seleccionar:
Git Credential Manager, luego
click en **Next**



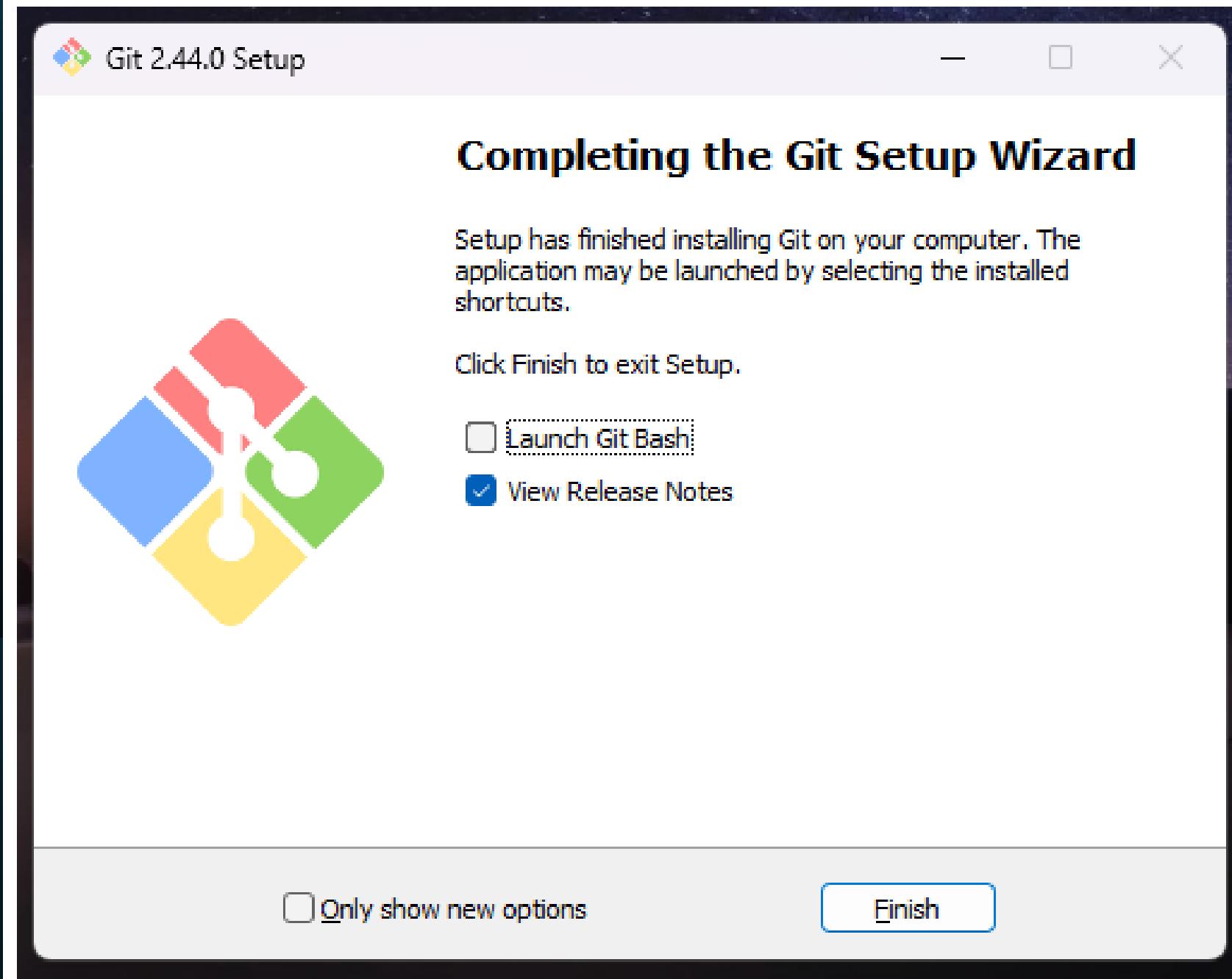
Seleccionar:
**Enable file
system
caching**, luego
click en **Next**



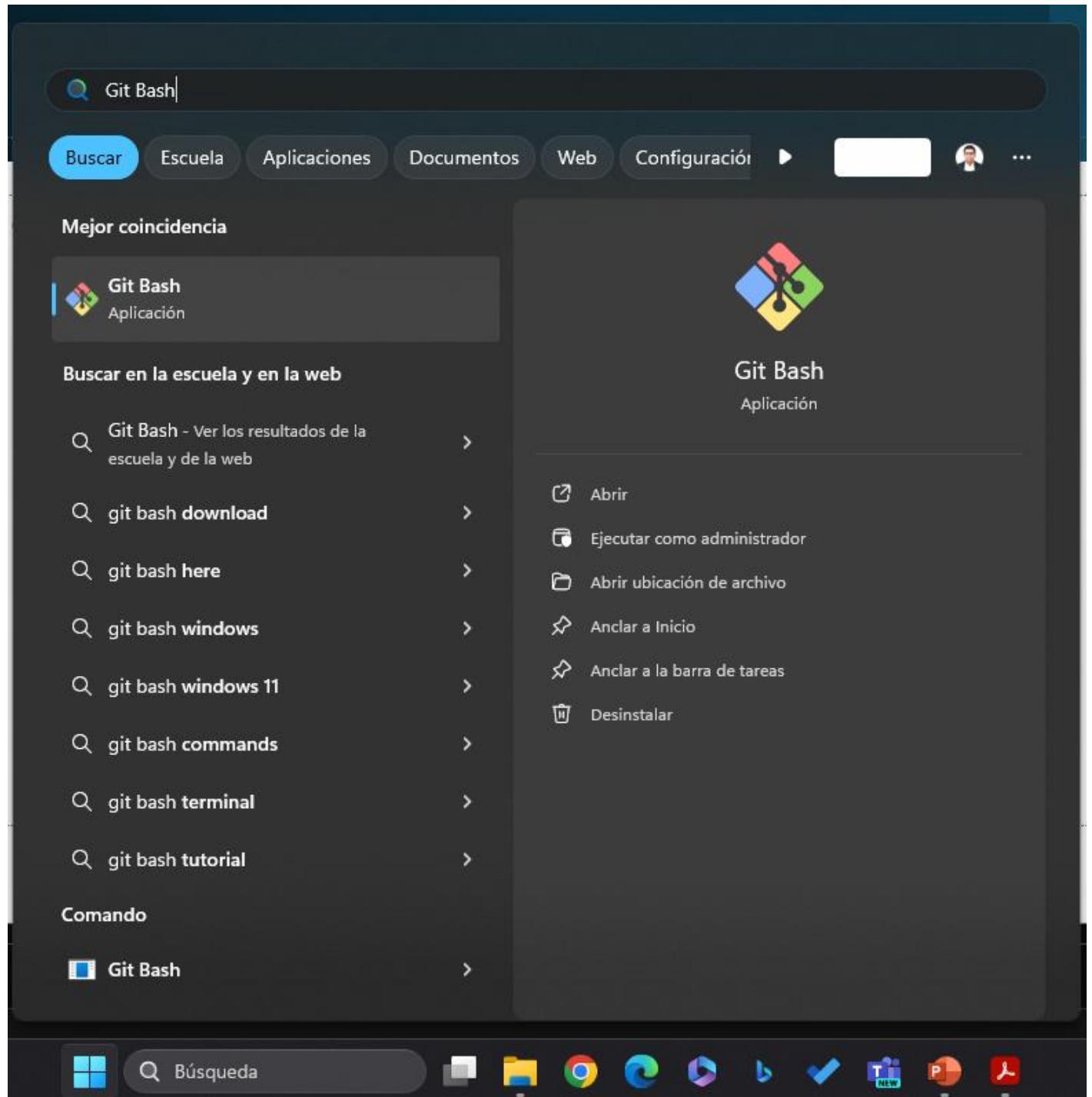
Click en Install
sin seleccionar
ninguna opción



Click en **Finish**
y reiniciar el
equipo

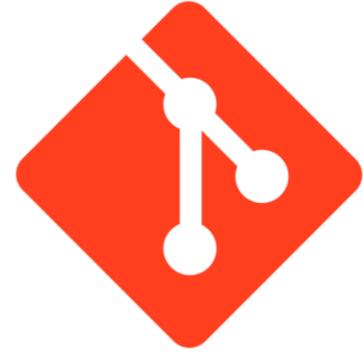


Vamos a ejecutar git:



Crear una
carpeta con
nombre:
modulo3-git

```
MINGW64:/c/Users/juanj/OneDrive/Documentos/modulo3-git
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$
```



git

3.2.2 Configuración inicial

Configuración del usuario:

Lo primero que deberás hacer cuando instalas Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ git config --global user.name "Aqui va tu nombre pseudonimo"
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ git config --global user.email tu-email@tajamar365.com
```

Comprobando configuración inicial:

Si quieres comprobar tu configuración, puedes usar el comando: **git config --list** para mostrar todas las propiedades que Git ha configurado:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor=nano.exe
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=balrod
user.email=juanjose.rodriguez@tajamar365.com
```

Establecer el editor de texto determinado:

Git usa un editor de texto para escribir mensajes de commit. Puedes especificar el editor que prefieras, por ejemplo, nano, vim o code (Visual Studio Code):

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/test_git
$ git config --global core.editor "code --wait"
```

Así se verifica el editor predeterminado:

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/test_git
$ git config --global core.editor
code --wait
```

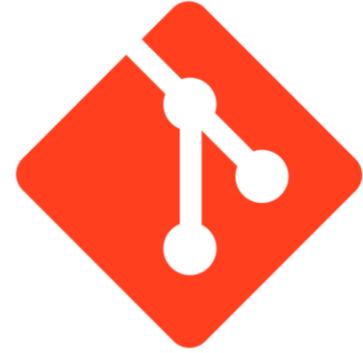
Configurar el manejo de fin de línea:

Si trabajas en un entorno mixto (Windows/Linux/Mac), es posible que desees configurar cómo Git maneja los caracteres de fin de línea para evitar problemas:

```
git config --global core.autocrlf true # Para Windows  
git config --global core.autocrlf input # Para Linux/Mac
```

Y para verificar:

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/test_git  
$ git config --global core.autocrlf  
true
```



git

3. Fundamentos de Git

Obteniendo un repositorio Git

Puedes obtener un proyecto Git de dos maneras. La primera es crear un proyecto o tomar un proyecto o directorio existente e importarlo en Git. La segunda es **clonar un repositorio existente en Git desde otro servidor (Github)**

Comenzamos por crear un nuevo proyecto

- Crear una carpeta que se llame proyecto1. Entramos en ella.
- Luego crear un fichero .txt que se llame hola-git.txt con nano o vs code y escribir:

Hola Mundo

Hola Git

- Guardar y salir.

Salidas

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ mkdir proyecto1
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ cd proyecto1
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1
$
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1
$ nano hola-git.txt
```

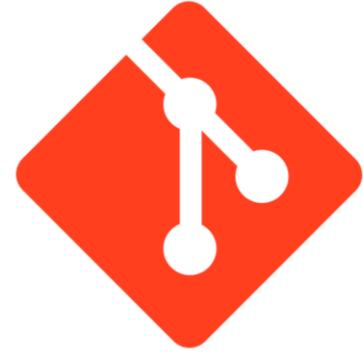
Escribir Hola Mundo Hola Git, guardar y salir

```
MINGW64:/c/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1
GNU nano 7.2
Hola Mundo
Hola Git|
```

```
MINGW64:/c/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1
GNU nano 7.2
Hola Mundo
Hola Git
hola-git.txt
Modified
```

[New File]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify
AC Location ^/ Go To Line M-U Undo
M-E Redo M-A Set Mark
M-6 Copy



git

3.1 Creando nuestro primer repositorio bajo control de versiones

Para pasar nuestro proyecto a nuestro working directory de git escribimos el comando: **git init**.

Debemos estar posicionados ‘dentro’ de la carpeta que queremos llevar al repositorio local **antes** de escribir **git init**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1
$ git init
Initialized empty Git repository in c:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1/.git/
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ ls -a
./  ../  .git/  hola-git.txt
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ |
```

Para ver el status de nuestro(s) fichero(s) escribir : **git status**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
on branch master
No commits yet
```

```
Untracked files:
(use "git add <file>..." to include in what will be committed)
  hola-git.txt
```

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

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ |
```

En dispositivos Mac aparece otro directorio llamado **.DS_Store**.



git

3.2 Enviando nuestros cambios al 'Staging area'

Vamos a indicarle a git que empiece a ‘trackear’ nuestros ficheros, esto es, enviar nuestro fichero al ‘staging área’

Escribimos: **git add hola-git.txt**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git add hola-git.txt
warning: in the working copy of 'hola-git.txt', LF will be replaced by CRLF the next time Git touches it
```

Vamos a pedir otro ‘status’ de nuestro proyecto:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   hola-git.txt
```



git

3.3 Confirmar cambios (commit) para
enviar/actualizar repositorio local

Supongamos que nuestro fichero ya está listo para ser enviado al repositorio local. Escribimos: git commit

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git commit|
```

Se abrirá el editor **nano** para que escribas el mensaje de los cambios que has realizado sobre el fichero.

Por ahora solo escribir: **Mi primer commit**. Luego guardar y salir

```
GNU nano 7.2                                     c:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1
Mi primer commit
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# on branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   hola-git.txt
#
```

Git ha tomado ‘foto’ de los cambios que has realizado sobre el archivo y le asigna una Hash (identificador único)

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git commit
[master (root-commit) 059c55b] Mi primer commit
 1 file changed, 2 insertions(+)
 create mode 100644 hola-git.txt
```

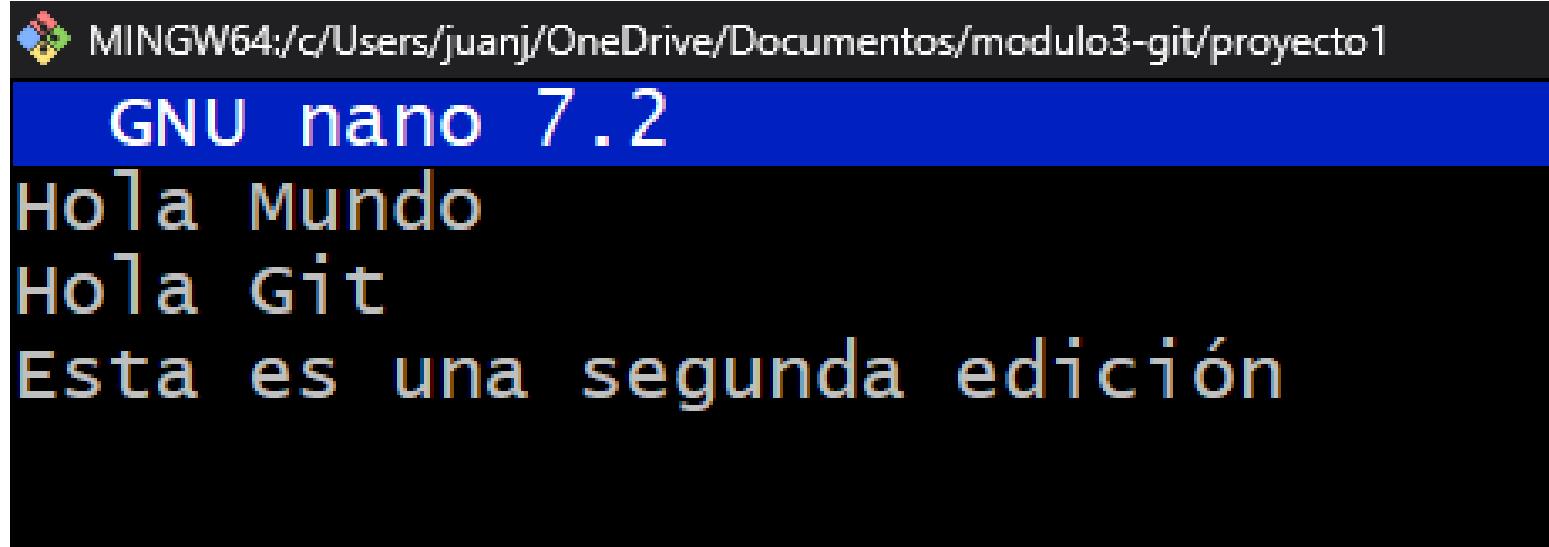
Vamos a pedir otro ‘status’

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Hay otra manera de hacer un commit más rápido, antes de ello vamos a realizar cambios sobre nuestro fichero **hola-git.txt**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ nano hola-git.txt
```

Escribimos en el editor: **Esta es una segunda edición**



```
MINGW64;/c/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1
GNU nano 7.2
Hola Mundo
Hola Git
Esta es una segunda edición
```

Guardar y salir.

Vamos a pedir un ‘status’

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
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:   hola-git.txt

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

Vamos a enviarlo a la zona de preparación (staging area): **git add hola-git.txt**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git add hola-git.txt
warning: in the working copy of 'hola-git.txt', LF will be replaced by CRLF the next time Git touches it
```

Vamos a pedir un ‘status’

```
juanj@balroddj: MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hola-git.txt
```

Ahora vamos a confirmar (**commit**) los cambios (enviar a *Repository*), esta vez escribimos:

git commit -m “he añadido una tercera línea”



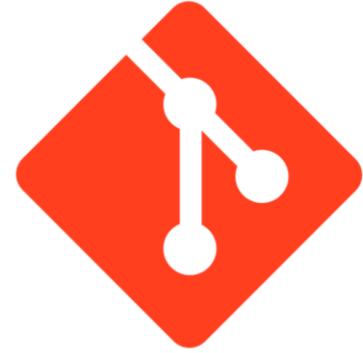
Estamos añadiendo el mensaje que va en el commit sin necesidad de que se abra el editor nano

Debería verse la siguiente salida en git bash:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git commit -m "he añadido una tercera línea"
[master 13b9b67] he añadido una tercera línea
 1 file changed, 1 insertion(+)
```

Vamos a pedir otro ‘status’

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```



git

3.4 Trabajando con el historial

Con la orden git log podemos ver todos los cambios que hemos hecho:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log
commit 13b9b67ca458e27d7596b9dea4299d134b4cbdff (HEAD -> master)
Author: balrod <juanjose.rodriguez@tajamar365.com>
Date:   Sat Apr 13 13:01:54 2024 +0200
```

he añadido una tercera línea

```
commit 059c55b12ae8d8693ca23388d64db1334afe995a
Author: balrod <juanjose.rodriguez@tajamar365.com>
Date:   Sat Apr 13 12:28:47 2024 +0200
```

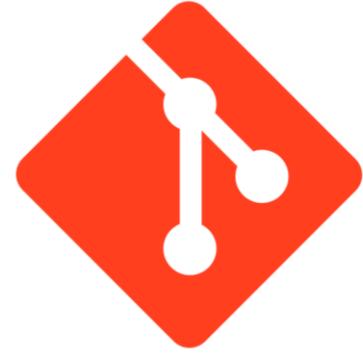
Mi primer commit

También es posible ver versiones abreviadas o limitadas, dependiendo de los parámetros: **git log --oneline**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
13b9b67 (HEAD -> master) he añadido una tercera línea
059c55b Mi primer commit
```

Una versión muy útil de git log es la siguiente, pues nos permite ver en que lugares está master y HEAD, y otros detalles mas: **git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
* 13b9b67 2024-04-13 | he añadido una tercera línea (HEAD -> master) [balrod]
* 059c55b 2024-04-13 | Mi primer commit [balrod]
```



git

3.5 Recuperando versiones anteriores

Cada cambio es etiquetado por un hash, para poder regresar a ese momento del estado del proyecto se usa la orden git checkout

Para ilustrar esto hagamos lo siguiente:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
13b9b67 (HEAD -> master) he añadido una tercera línea
059c55b Mi primer commit
```

Tomamos nota o copiamos el hash del commit al cual queremos regresar. Antes vamos a ver el contenido del fichero actualizado por última vez, escribimos: **cat hola-git.txt**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ cat hola-git.txt
Hola Mundo
Hola Git
Esta es una segunda edición
```

Supongamos que se quiere ver la versión anterior a ese fichero, para ellos escribimos: git checkout 059c55b

```
juanj@balroddj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git checkout 059c55b
Note: switching to '059c55b'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 059c55b Mi primer commit

juanj@balroddj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 ((059c55b...))
$
```

Veamos el contenido del fichero “en ese punto en el tiempo”:
cat hola-git.txt

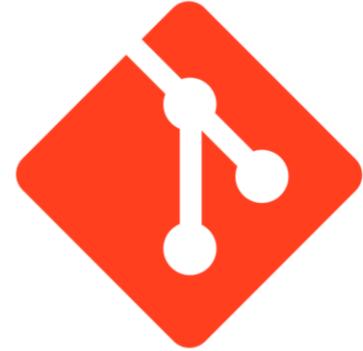
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 ((059c55b...))
$ cat hola-git.txt
Hola Mundo
Hola Git
```

En ningún caso se ha editado el archivo, sino que estamos viendo una instantánea en un “punto en el tiempo” de ese fichero. En este commit se pueden hacer varias cosas:

- Editar el archivo y hacer un nuevo commit (peligroso)
- Crear una rama experimental a partir de ese commit y hacer todos los cambios que necesito probar
- Volver al Head de la rama master (o de la rama de desarrollo), es decir la versión más actual (regresar al presente):
git checkout master (o también: **git switch -**)

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 ((059c55b...))
$ git checkout master
Previous HEAD position was 059c55b Mi primer commit
Switched to branch 'master'
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ |
```

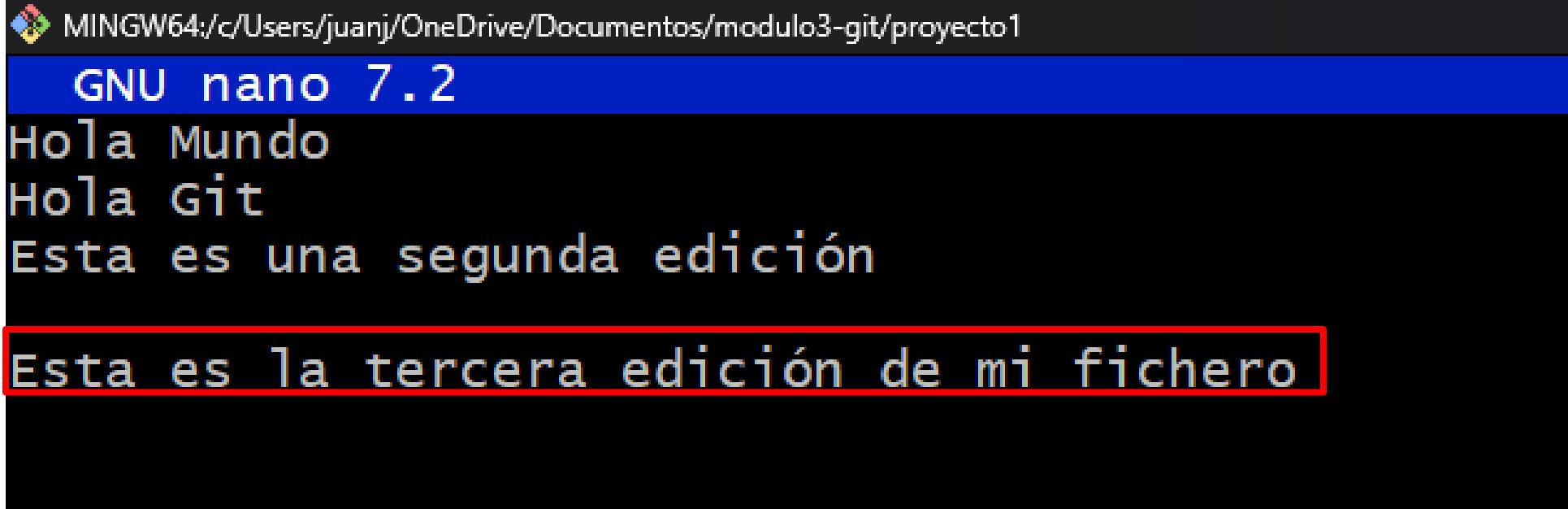


git

3.6 Etiquetando versiones

Antes de hablar del tag vamos a realizar otra modificación de nuestro fichero hola-git.txt :

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ nano hola-git.txt|
```



```
MINGW64:/c/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto1
GNU nano 7.2
Hola Mundo
Hola Git
Esta es una segunda edición
Esta es la tercera edición de mi fichero
```

Guardar y salir del editor nano.

Vamos a pedir un status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
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:   hola-git.txt

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

Enviamos nuestros cambios al “Staging area” : **git add hola-git.txt** y pedimos otro status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git add hola-git.txt

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hola-git.txt
```

Confirmamos cambios:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git commit -m "Linea 4 vacia y contenido nuevo en linea 5"
[master 1bc15bd] Linea 4 vacia y contenido nuevo en linea 5
 1 file changed, 2 insertions(+)
```

Pedimos otro status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Verificamos el histórico de nuestros cambios:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
1bc15bd (HEAD -> master) Linea 4 vacia y contenido nuevo en linea 5
13b9b67 he añadido una tercera linea
059c55b Mi primer commit
```

Una forma de usar ‘tag’ es escribiendo: `git tag ‘nombre de la etiqueta’`

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag v0.3
```

Por defecto, ese tag es asignado a la versión más reciente. Para asignar un tag a una versión específica añadimos el hash respectivo:

git tag <nombre-etiqueta> <commit-hash>

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag v0.2 13b9b67
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag v0.1 059c55b
```

Para ver todos los tags creados escribimos: **git tag**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag
v0.1
v0.2
v0.3
```

Si queremos ver los tags asociados a cada hash usamos **git log --oneline**

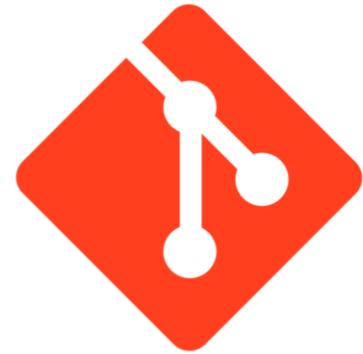
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
1bc15bd (HEAD -> master, tag: v0.3) Linea 4 vacia y contenido nuevo en linea 5
13b9b67 (tag: v0.2) he añadido una tercera linea
059c55b (tag: v0.1) Mi primer commit
```

Para borrar un tag escribimos: **git tag -d ‘nombre tag’**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag -d v0.3 v0.2 v0.1
Deleted tag 'v0.3' (was 1bc15bd)
Deleted tag 'v0.2' (was 13b9b67)
Deleted tag 'v0.1' (was 059c55b)
```

Verificamos con: **git tag** o bien escribiendo: **git log --oneline**

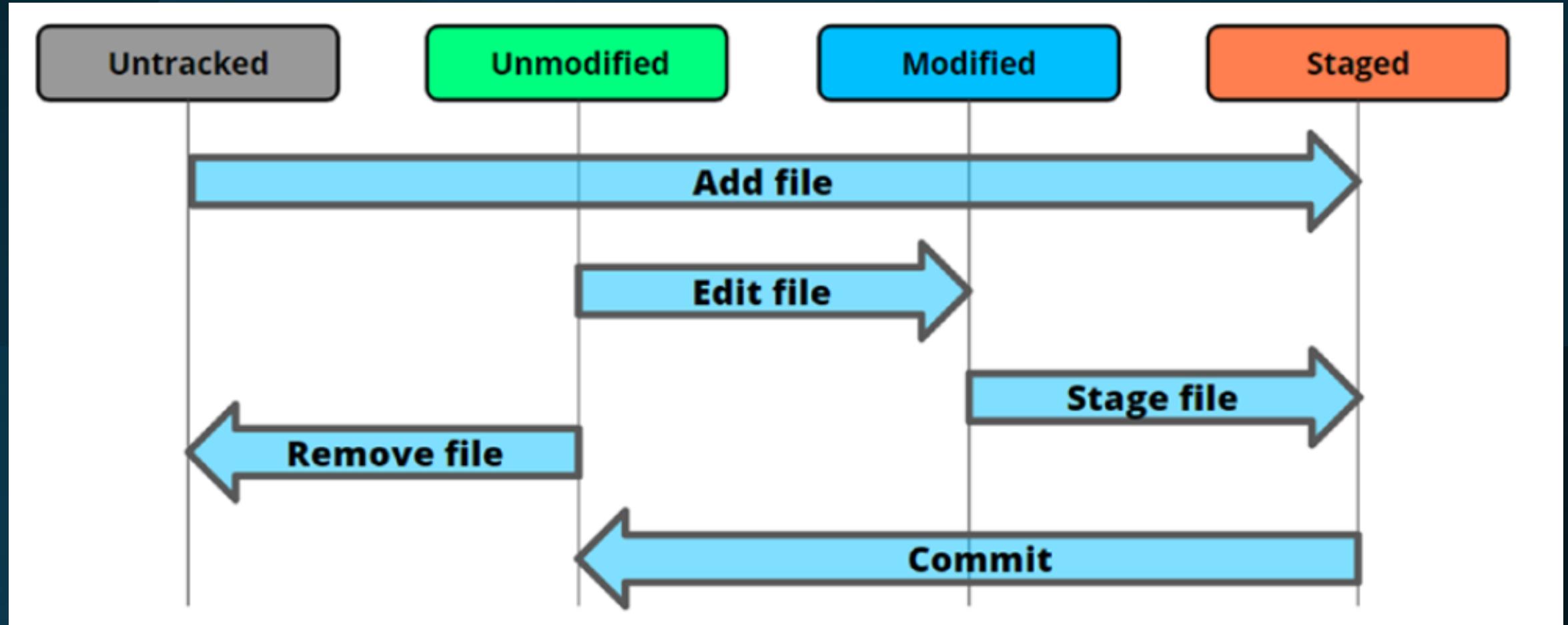
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
1bc15bd (HEAD -> master) Linea 4 vacia y contenido nuevo en linea 5
13b9b67 he añadido una tercera linea
059c55b Mi primer commit
```



git

3.7 Estados de un fichero

Los diferentes estados en los que puede estar un archivo y las acciones que permiten la transición entre ellos



1. Untracked (No rastreado)

- Un archivo en este estado es nuevo en el proyecto y Git no lo está rastreando. Es un archivo que existe en el sistema de archivos, pero Git no sabe nada de él.
- Para ver los archivos no rastreados, usa: **git status**

1. Untracked (No rastreado)

- Para comenzar a rastrear el archivo, necesitas agregarlo al área de preparación (staging area): **git add archivo.txt**

2. Unmodified (No modificado)

- Una vez que el archivo ha sido **agregado** al repositorio y se ha realizado un **commit**, el archivo se considera "unmodified" o no modificado. Esto significa que no ha habido cambios en el archivo desde el último commit.
- Si el archivo no se ha modificado, no es necesario hacer nada adicional hasta que se hagan cambios.

3. Modified (Modificado)

- Cuando se hacen cambios a un archivo que Git ya está rastreando, el archivo pasa al estado "modificado".
- Para ver qué archivos han sido modificados: `git status`. Este comando te mostrará los archivos que han sido modificados pero no preparados (**staged**) para el próximo commit.

4. Staged (Preparado)

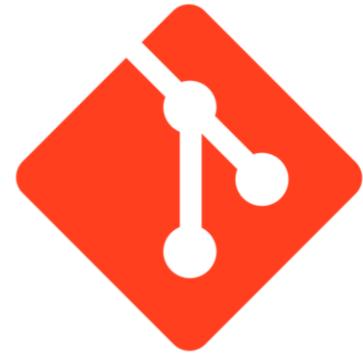
- Después de modificar un archivo, si deseas preparar esos cambios para un commit, debes usar nuevamente el comando git add para moverlo al estado "staged": `git add archivo.txt`.
- En este estado, el archivo está listo para ser confirmado en el repositorio con un commit.

5. Commit (Confirmación de cambios)

- Una vez que los archivos están en el estado "staged", puedes confirmarlos en el repositorio usando el siguiente comando: **git commit -m "Mensaje del commit"**
- Con esto, los archivos vuelven al estado "unmodified", ya que los cambios han sido guardados en el historial del repositorio.

6. Remove File (Eliminar un archivo)

- Si decides eliminar un archivo del repositorio, puedes usar: **git rm archivo.txt**
- Este comando eliminará el archivo tanto del sistema de archivos como del repositorio de Git. Después, puedes confirmar el cambio: **git commit -m "Eliminado archivo.txt"**



git

3.8 Las cuatro áreas de git

Cuando ejecutamos el comando "git init" se crean las siguientes áreas

The Stash

The Working Directory

The Index

The Repository

1. The Stash (La Alacena o Reserva Temporal)

Función: El stash es una característica de Git que te permite guardar temporalmente los cambios que aún no deseas confirmar. Es útil cuando estás trabajando en algo, pero necesitas cambiar de contexto sin perder tu progreso.

Comando: `git stash`. Este comando guarda los cambios actuales (modificaciones no confirmadas) en un área temporal, dejando tu directorio de trabajo limpio. Luego, puedes recuperarlos cuando lo necesites con:

`git stash apply`

2. The Working Directory (El Directorio de Trabajo)

Función: Es el área donde trabajas en los archivos del proyecto. Aquí es donde haces cambios y modificaciones en los archivos.

Los archivos en el directorio de trabajo pueden estar en dos estados: rastreados (tracked) o no rastreados (untracked). Si son rastreados, pueden estar modificados o no modificados, dependiendo de si han cambiado desde el último commit.

Para añadir cambios desde el directorio de trabajo al área de preparación (staging): `git add archivo.txt`

3. The Index (El Índice o Área de Preparación)

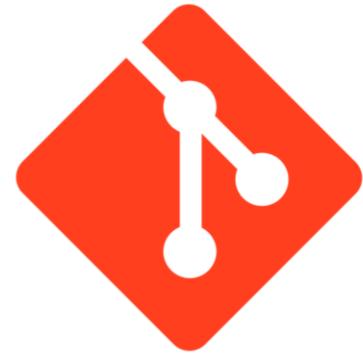
Función: El index es el área donde preparas los archivos para el commit. También se conoce como el "staging area" o área de preparación. Aquí defines exactamente qué cambios quieres confirmar en el próximo commit.

Después de modificar archivos en el directorio de trabajo, los agregas al índice antes de confirmarlos en el repositorio. **git add archivo.txt**

4. The Repository (El Repositorio)

Función: El repositorio es donde se guardan los cambios confirmados (commits). Es el historial oficial de tu proyecto, donde cada commit queda registrado con un mensaje descriptivo y un ID único.

Es en esta área donde Git almacena de manera permanente las versiones del proyecto. Para confirmar los cambios preparados en el índice en el repositorio: **git commit -m "Mensaje descriptivo"**



git

3.9 Visualizar y comparar los cambios sobre un fichero

git diff

Para ver los cambios que se han realizado en el código usamos la orden **git diff**. La orden sin especificar nada más, mostrará los cambios que no han sido añadidos aún, es decir, todos los cambios que se han hecho antes de usar la orden **git add**. Después se puede indicar un parámetro y dará los cambios entre la versión indicada y el estado actual. O para comparar dos versiones entre sí, se indica la más antigua y la más nueva.

git diff

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git diff
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$
```

No muestra nada mientras no se estén haciendo cambios en el **working directory**. Supongamos que realizamos cambios sobre el fichero en el directorio de trabajo:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ nano hola-git.txt
```

```
GNU nano 7.2
Guardando cambios en el fichero
Hasta la tercera edición de mi fichero
test version 4
```

Guardar y salir de nano

Escribamos: git diff

```
juanj@balroddj: MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git diff
diff --git a/hola-git.txt b/hola-git.txt
index c9a4d1f..1d6b172 100644
--- a/hola-git.txt
+++ b/hola-git.txt
@@ -3,3 +3,4 @@ Hola Git
 Esta es una segunda edición

 Esta es la tercera edición de mi fichero
+test version 4
```

A continuación, una breve explicación del significado de cada línea de la salida anterior:

Primera línea:

```
diff --git a/hola-git.txt b/hola-git.txt
```

Indica que se está comparando el archivo hola-git.txt de la carpeta a (la versión anterior) con el archivo hola-git.txt de la carpeta b (la versión más reciente).

Segunda línea:

```
index c9a4d1f..1d6b172 100644
```

c9a4d1f y **1d6b172** son los identificadores de commit (hashes) de la versión anterior y la más reciente del archivo, respectivamente.

100644 indica el modo de permiso del archivo, en este caso 100644 para lectura y escritura para el propietario, lectura para el grupo y otros.

Tercera línea:

```
--- a/hola-git.txt
```

Representa el inicio de la sección que contiene el contenido del archivo hola-git.txt de la versión anterior.

Cuarta línea

```
+++ b/hola-git.txt
```

Representa el inicio de la sección que contiene el contenido del archivo hola-git.txt de la versión más reciente

Quinta línea:

```
@@ -3,3 +3,4 @@ Hola Git
```

```
Esta es una segunda edición
```

```
Esta es la tercera edición de mi fichero  
+test version 4
```

Esta línea contiene información sobre las líneas modificadas. El -3,3 indica que se han eliminado 3 líneas desde la línea 3, y se ha agregado 1 línea en su lugar. En este caso, se ha agregado la línea “test version 4”.

+test version 4: Esta línea no está presente en la versión anterior y ha sido agregada en la versión más reciente. El signo + al inicio indica que esta línea es nueva.

Se deben contar las líneas después de Hola Git

```
Hola Mundo
```

```
Hola Git
```

```
Esta es una segunda edición
```

```
Esta es la tercera edición de mi fichero  
test version 4
```

Vamos a pedir status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git status
On branch master
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:   hola-git.txt

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

Podemos confirmar directamente los cambios al repositorio local sin pasar por el staging area (ver página 25) escribiendo: **git commit -a**

Se abrirá el editor nano. Escribir mensaje , guardar y salir.

```
juanj@balrodjj MINGW64 ~/oneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git commit -a
[master 724fe08] He añadido la linea 7
 1 file changed, 1 insertion(+)
```

Luego escribimos: **git log --oneline**

```
juanj@balrodjj MINGW64 ~/oneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
724fe08 (HEAD -> master) He añadido la linea 7
1bc15bd Linea 4 vacia y contenido nuevo en linea 5
13b9b67 he añadido una tercera linea
059c55b Mi primer commit
```

Podemos usar git diff usando el hash anterior a la versión más actual:

```
$ git diff 1bc15bd  
diff --git a/hola-git.txt b/hola-git.txt  
index c9a4d1f..1d6b172 100644  
--- a/hola-git.txt  
+++ b/hola-git.txt  
@@ -3,3 +3,4 @@ Hola Git  
 Esta es una segunda edición  
  
 Esta es la tercera edición de mi fichero  
+test version 4
```

Compara la versión actual con el commit inmediatamente anterior

Comparemos el commit actual con el primer commit

```
$ git diff 059c55b  
diff --git a/hola-git.txt b/hola-git.txt  
index 112ab47..1d6b172 100644  
--- a/hola-git.txt  
+++ b/hola-git.txt  
@@ -1,2 +1,6 @@  
 Hola Mundo  
 Hola Git  
+Esta es una segunda edición  
+  
+Esta es la tercera edición de mi fichero  
+test version 4
```

git diff admite comparar cambios entre dos commits específicos:

git diff 1bc15bd 13b9b67

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git diff 1bc15bd 13b9b67
diff --git a/hola-git.txt b/hola-git.txt
index c9a4d1f..a9293a3 100644
--- a/hola-git.txt
+++ b/hola-git.txt
@@ -1,5 +1,3 @@
 Hola Mundo
 Hola Git
 Esta es una segunda edición
-
-Esta es la tercera edición de mi fichero
```

Cuidado con el orden, se escribe primero la versión más antigua

Se pueden usar git diff utilizando los tags. Vamos a crear algunos tags

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag version4 724fe08

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag version3 1bc15bd

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag version2 13b9b67

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag version1 059c55b
```

Veamos todas las versiones escribiendo: **git tag**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git tag
version1
version2
version3
version4
```

Veamos las versiones con **git log --oneline**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git log --oneline
724fe08 (HEAD -> master, tag: version4) He añadido la linea 7
1bc15bd (tag: version3) Linea 4 vacia y contenido nuevo en linea 5
13b9b67 (tag: version2) he añadido una tercera linea
059c55b (tag: version1) Mi primer commit
```

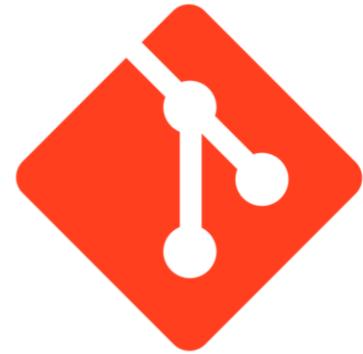
Comparemos cambios entre versiones usando git diff añadiendo tags

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git diff version4 version1
diff --git a/hola-git.txt b/hola-git.txt
index 1d6b172..112ab47 100644
--- a/hola-git.txt
+++ b/hola-git.txt
@@ -1,6 +1,2 @@
 Hola Mundo
 Hola Git
-Esta es una segunda edición
-
-Esta es la tercera edición de mi fichero
-test version 4
```

Cuidado con el orden, se escribe primero la versión más antigua

git diff version1 version4

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git diff version1 version4
diff --git a/hola-git.txt b/hola-git.txt
index 112ab47..1d6b172 100644
--- a/hola-git.txt
+++ b/hola-git.txt
@@ -1,2 +1,6 @@
 Hola Mundo
 Hola Git
+Esta es una segunda edición
+
+Esta es la tercera edición de mi fichero
+test version 4
```



git

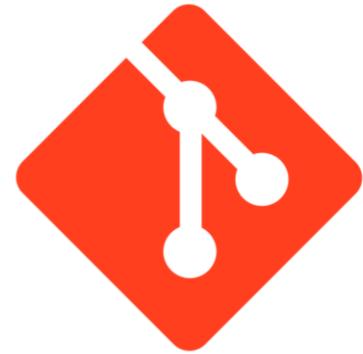
3.10 Cambiando el nombre de la
rama master por main

Comando: `git branch -m master main`

Git por defecto a la rama principal la llama master, en cambio GitHub la llama main. Para evitar conflictos a la hora de hacer envíos al repositorio remoto, vamos a cambiar en git el nombre de la rama principal master por main.

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (master)
$ git branch -m master main
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto1 (main)
$
```



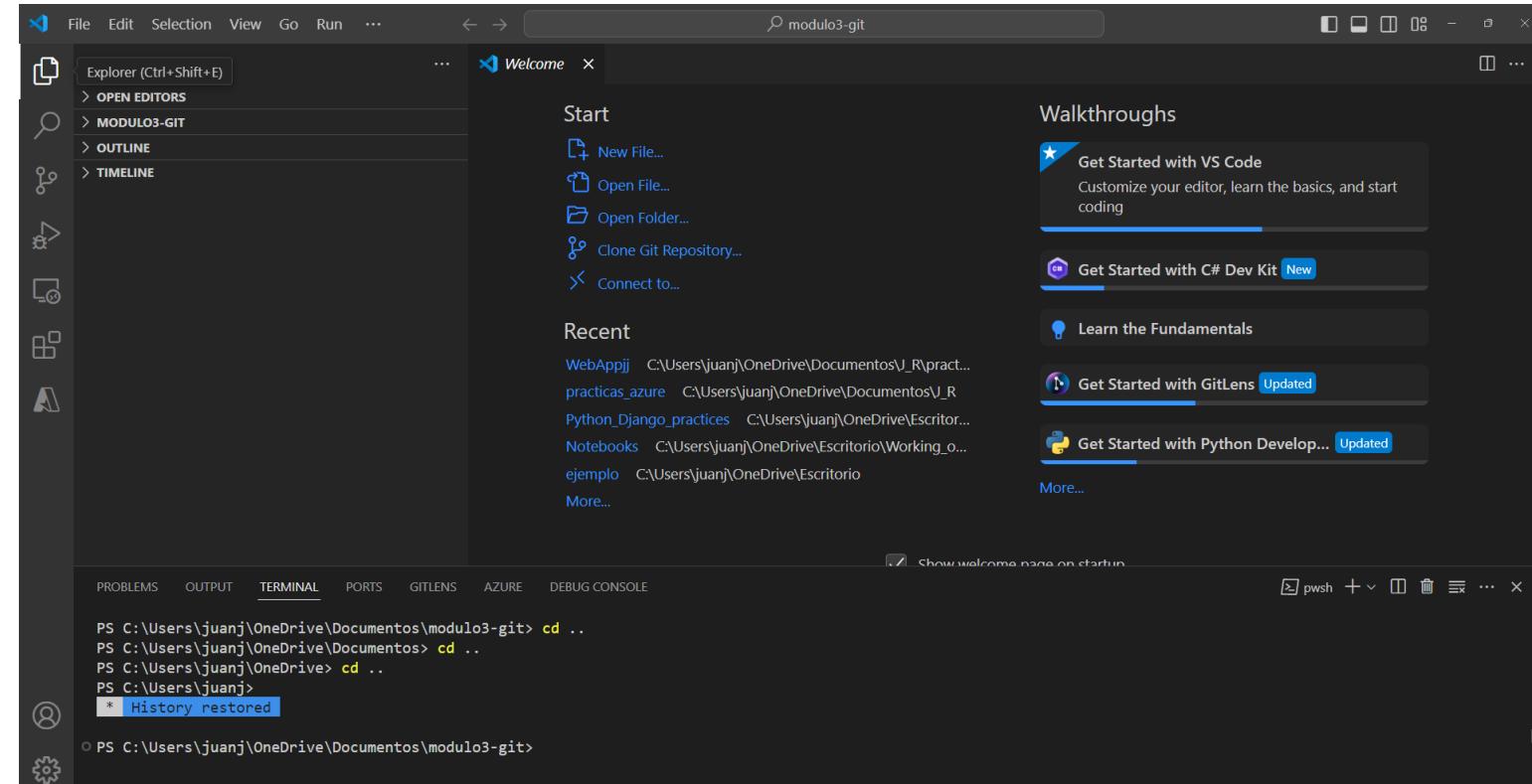
git

3.11 Visual Studio Code y Git

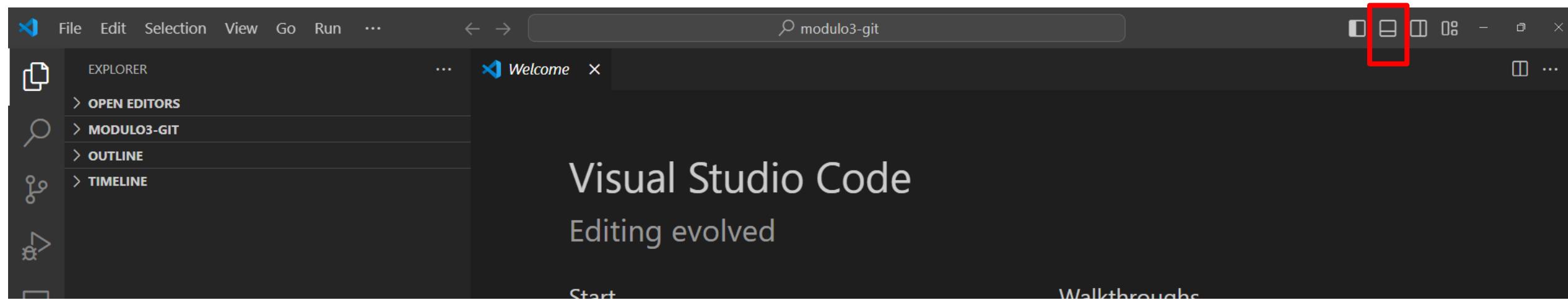
Desde PowerShell o desde git bash escribe: code

```
juanj@balrodjj MINGW64 ~/oneDrive/Documentos/modulo3-git
$ code
```

```
juanj@balrodjj MINGW64 ~/oneDrive/Documentos/modulo3-git
$
```



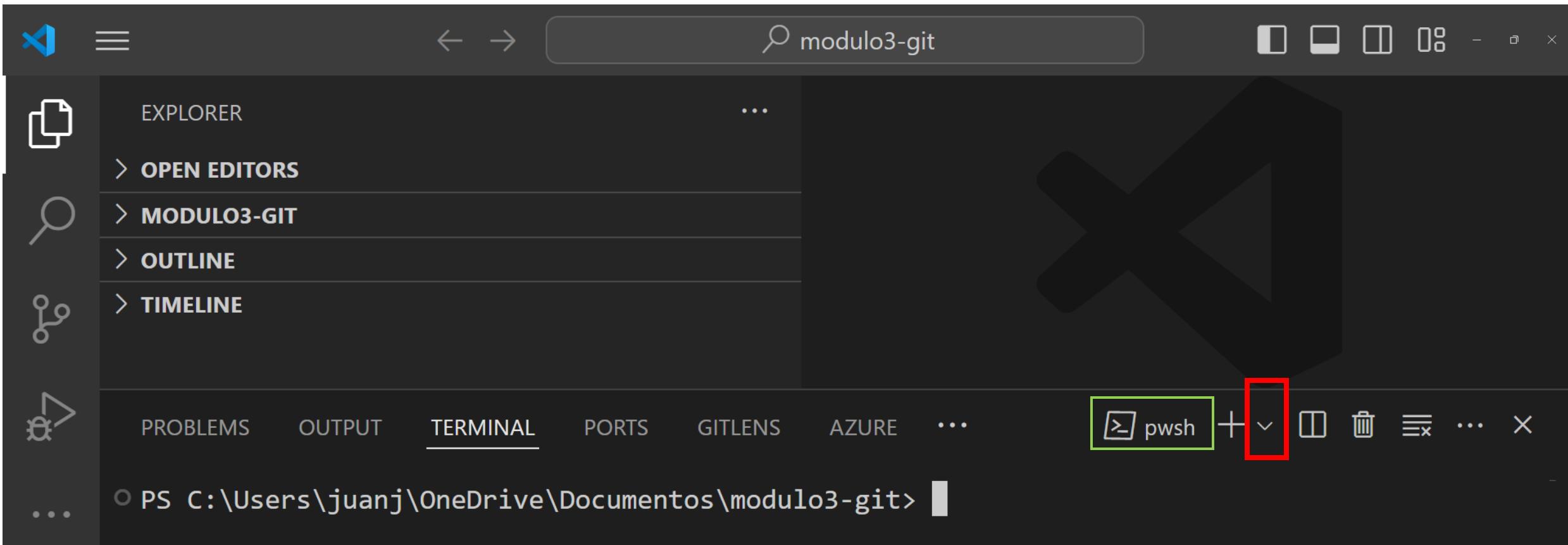
Visual Studio Code tiene una terminal integrada, desde el cual vamos a crear otro proyecto llamado proyecto2



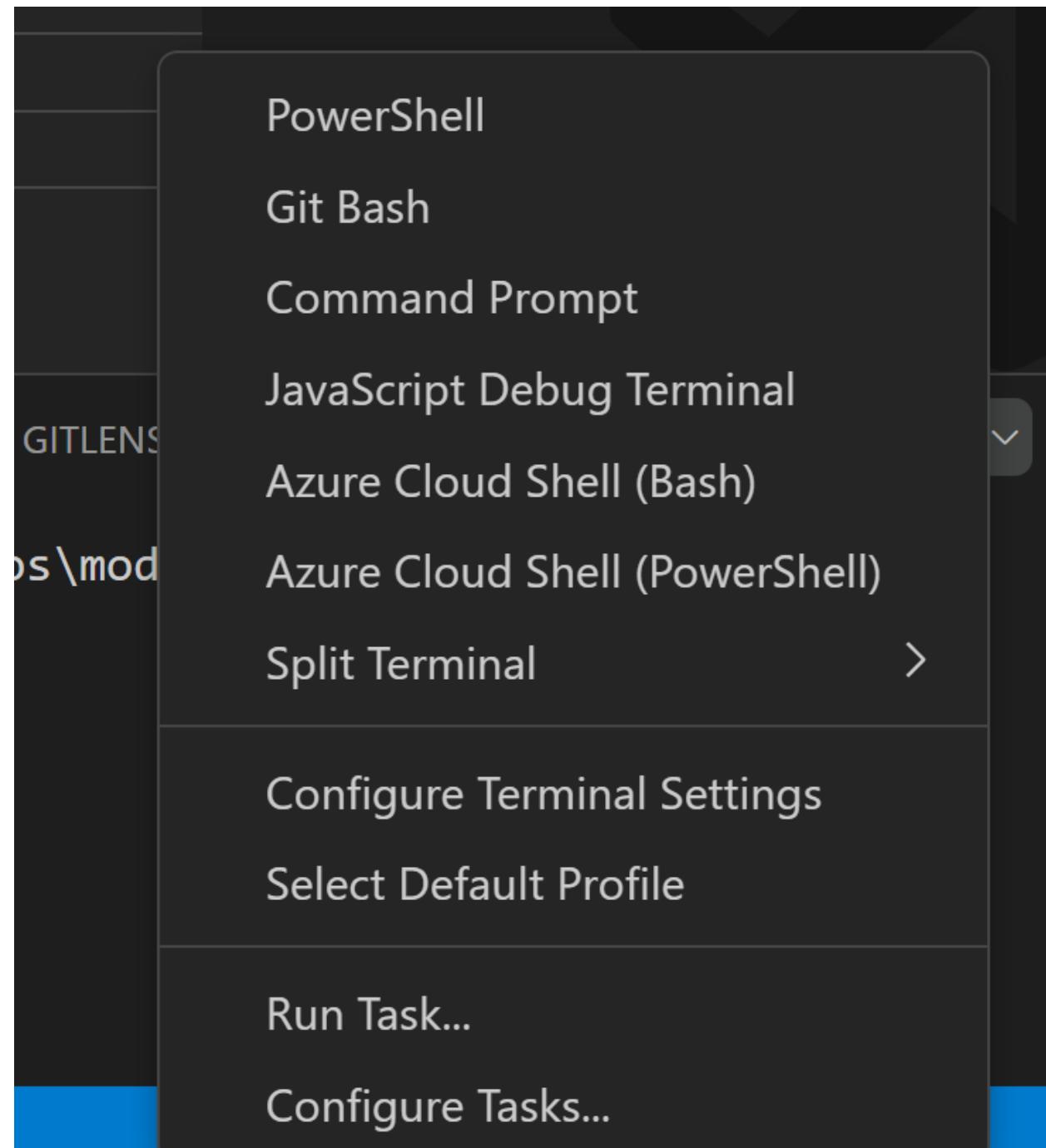
Al pinchar el cuadro rojo se abrirá en la parte inferior una terminal. Hay varias terminales para escoger:

- PowerShell
- Git Bash
- Command Prompt
- Azure CLI...

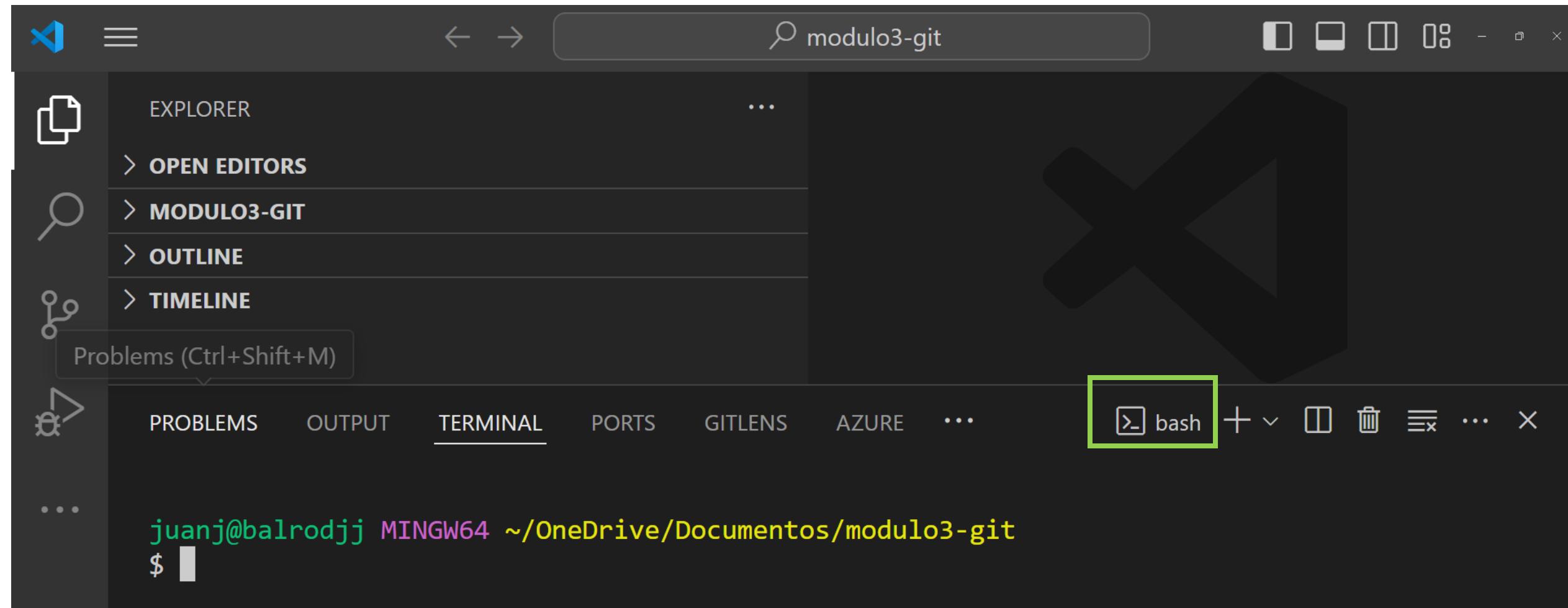
Hacemos click en el símbolo  para ver las opciones:



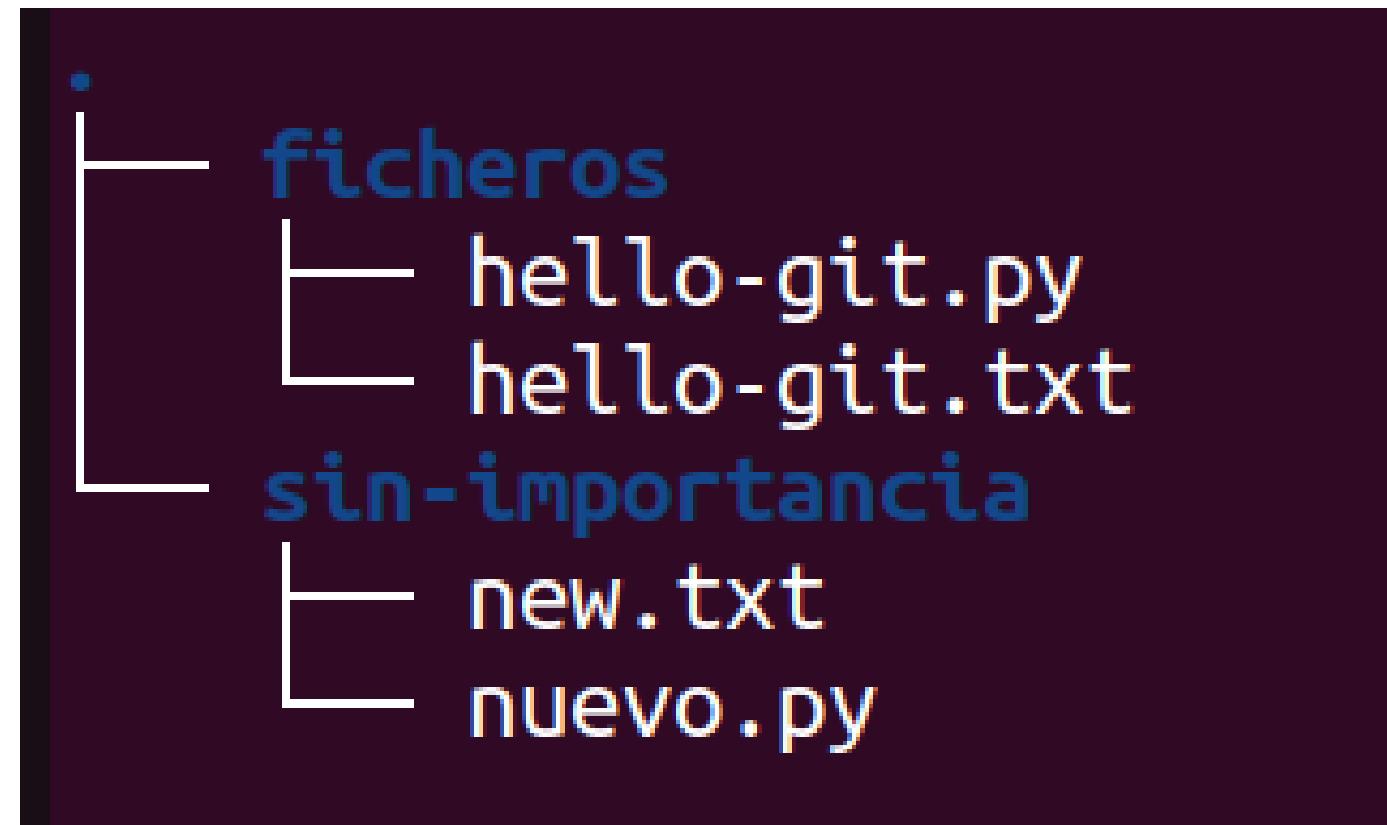
A continuación,
escoger Git Bash



Así debería verse Visual Studio Code (vs code). Vamos a movernos hasta el directorio donde tienes guardado tu proyecto de git.



Vamos a crear
desde la terminal
de vs code
(PowerShell o Git
Bash) una carpeta
de nombre:
 proyecto2 y luego
construir este
directorio:



Una vez creado el directorio, vamos a inicializar el control de versiones de nuestro proyecto2

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2
$ git init
Initialized empty Git repository in C:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto2/.git/
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (master)
$ █
```

Cambiamos el nombre de la rama master por main:

PROBLEMS OUTPUT TERMINAL PORTS GITLENS AZURE DEBUG CONSOLE

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (master)
$ git branch -m master main

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ █
```

Pedimos un status.

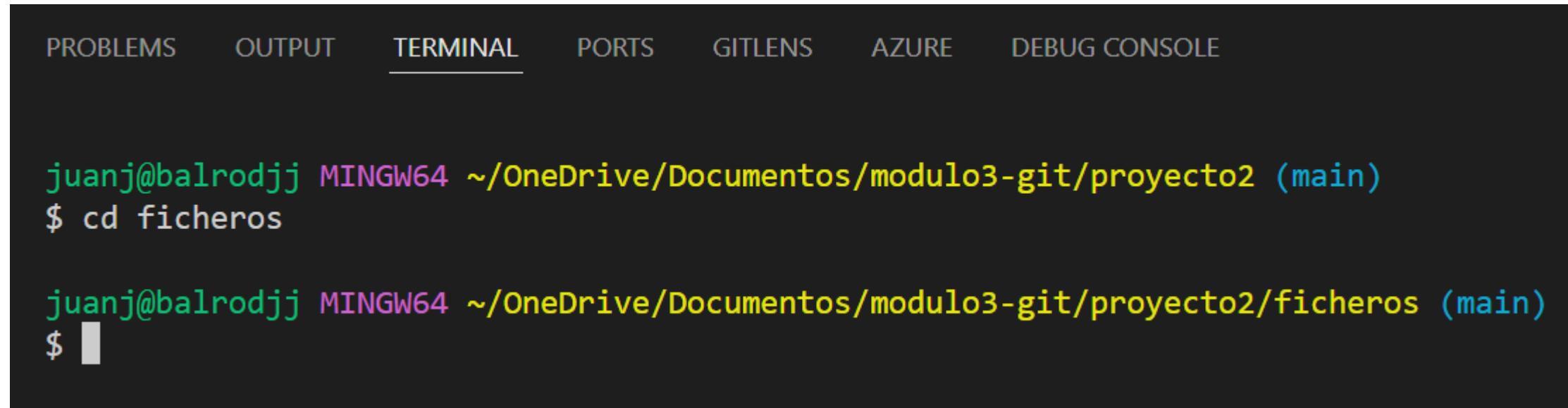
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ficheros/
    sin-importancia/

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

Vamos a entrar desde la terminal al directorio ficheros

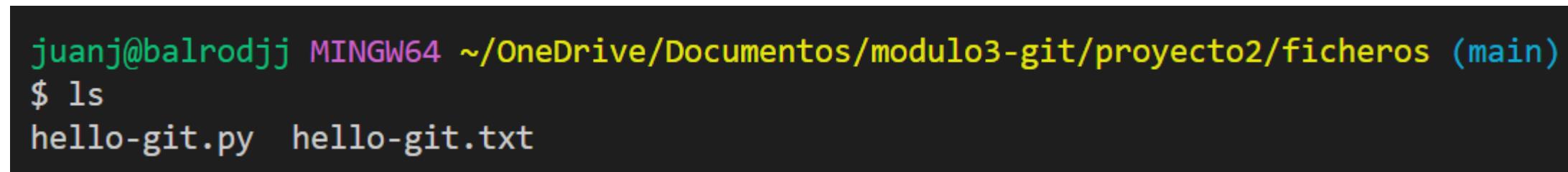


The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal window displays the following command-line session:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ cd ficheros

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$
```

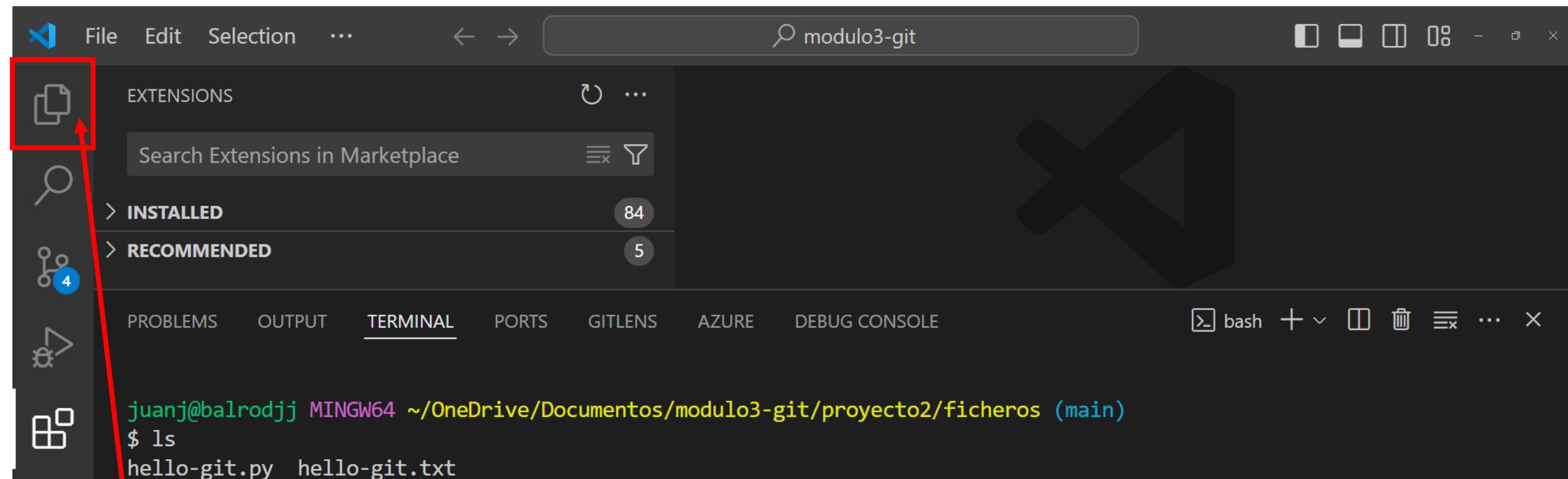
Listamos el contenido:



The screenshot shows the terminal window displaying the contents of the 'ficheros' directory:

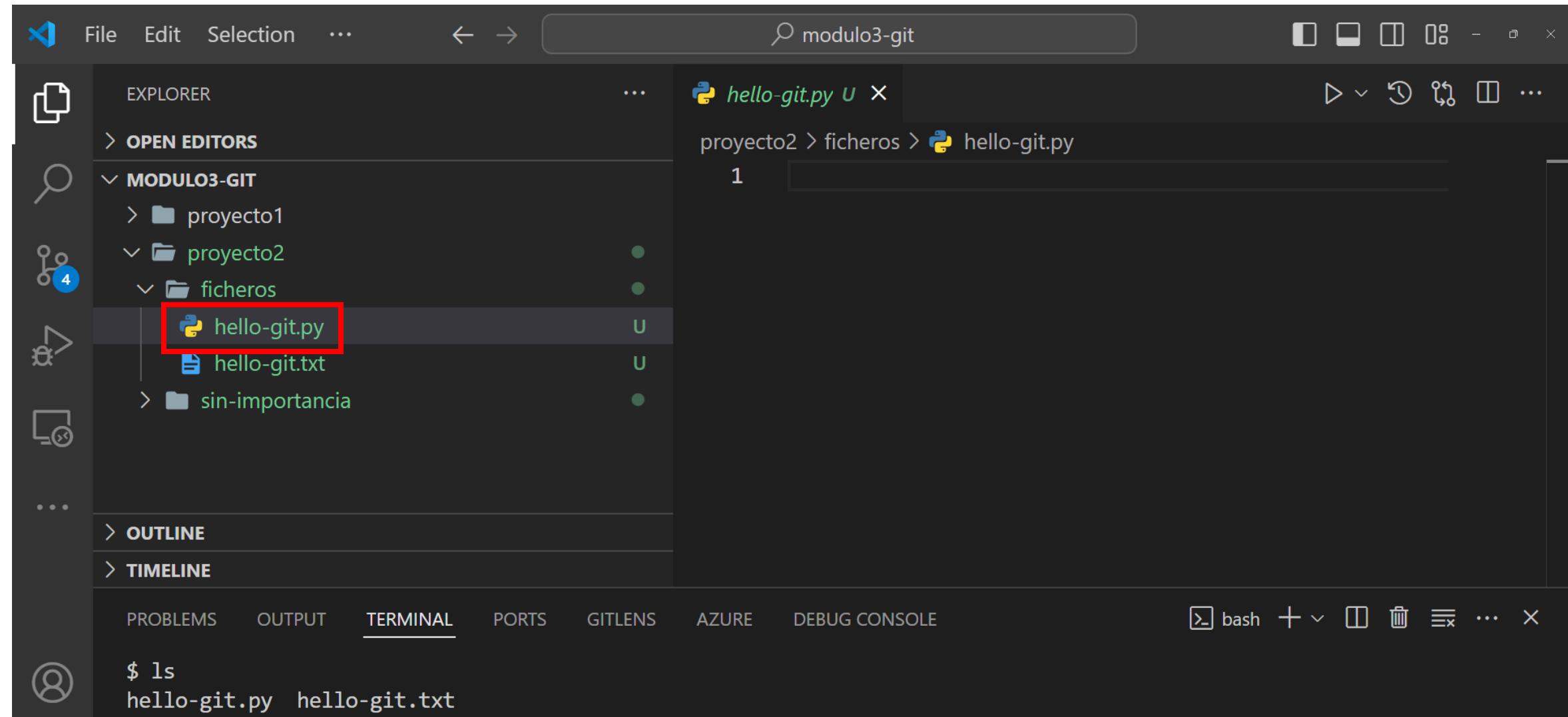
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ ls
hello-git.py  hello-git.txt
```

Vamos a editar el fichero: `hello-git.py` pero esta vez, no desde la terminal con nano sino desde el editor de vscode:

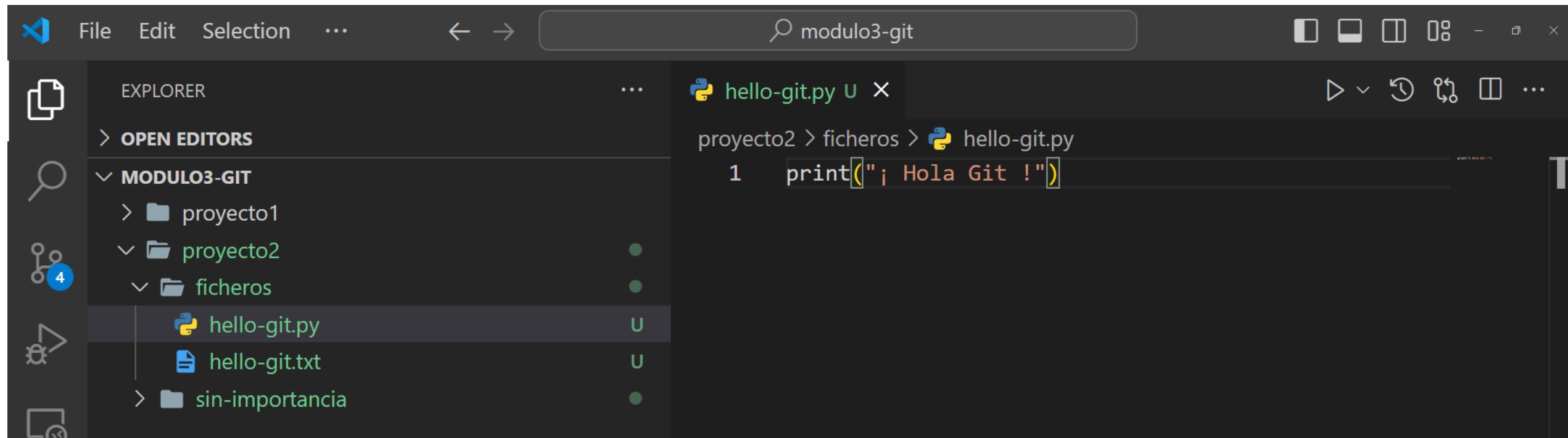


Hacer click

Nos desplazamos y hacemos click en el fichero: hello-git.py



Escribimos: print("¡ Hola Git !") y guardamos.



The screenshot shows the Visual Studio Code (VS Code) interface. The top bar includes the VS Code logo, File, Edit, Selection, a three-dot menu, navigation arrows, a search bar containing "modulo3-git", and window control icons. The left sidebar (Explorer) has icons for File Explorer, Search, Issues (with a '4' notification), Problems, and History. The 'MODULO3-GIT' section shows a folder structure: 'proyecto1', 'proyecto2' (expanded) with a 'ficheros' folder containing 'hello-git.py' (selected) and 'hello-git.txt', and a 'sin-importancia' folder. The right pane (Editor) displays the file 'hello-git.py' with the following content:

```
1  print("¡ Hola Git !")
```

Ahora nos vamos a la terminal de vscode y enviamos los cambios al staging área posicionados en el directorio ficheros:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add hello-git.py
```

git status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main

No commits yet

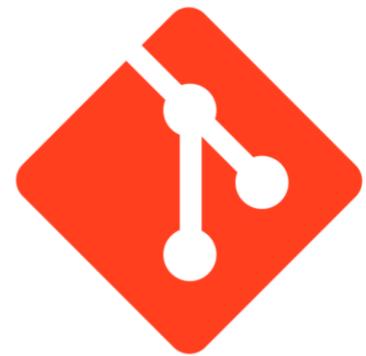
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   hello-git.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello-git.txt
    ../sin-importancia/
```

Confirmar (commit) el fichero hello-git.py

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "Primer commit: he añadido un print"
[main (root-commit) 5a238aa] Primer commit: he añadido un print
 1 file changed, 1 insertion(+)
 create mode 100644 ficheros/hello-git.py
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello-git.txt
    ../sin-importancia/
nothing added to commit but untracked files present (use "git add" to track)
```

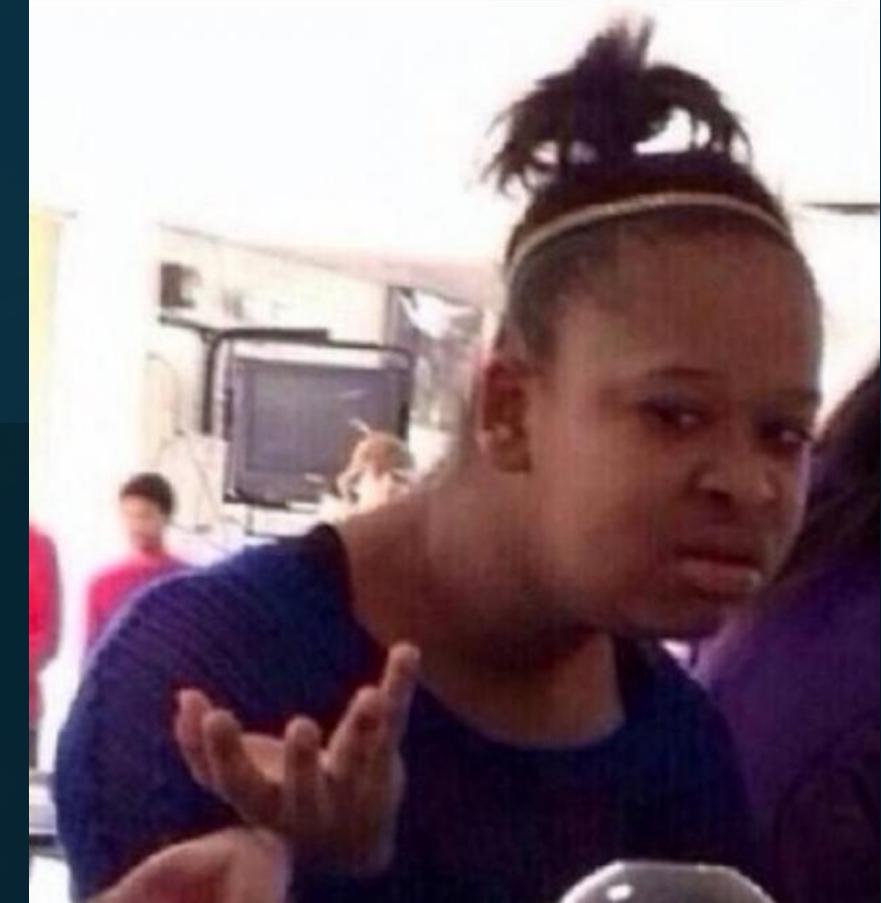


git

3.12 Uso de gitignore

When you add ` `.gitignore` in the gitignore file

Le Git:



Volvemos a proyecto2: cd ..

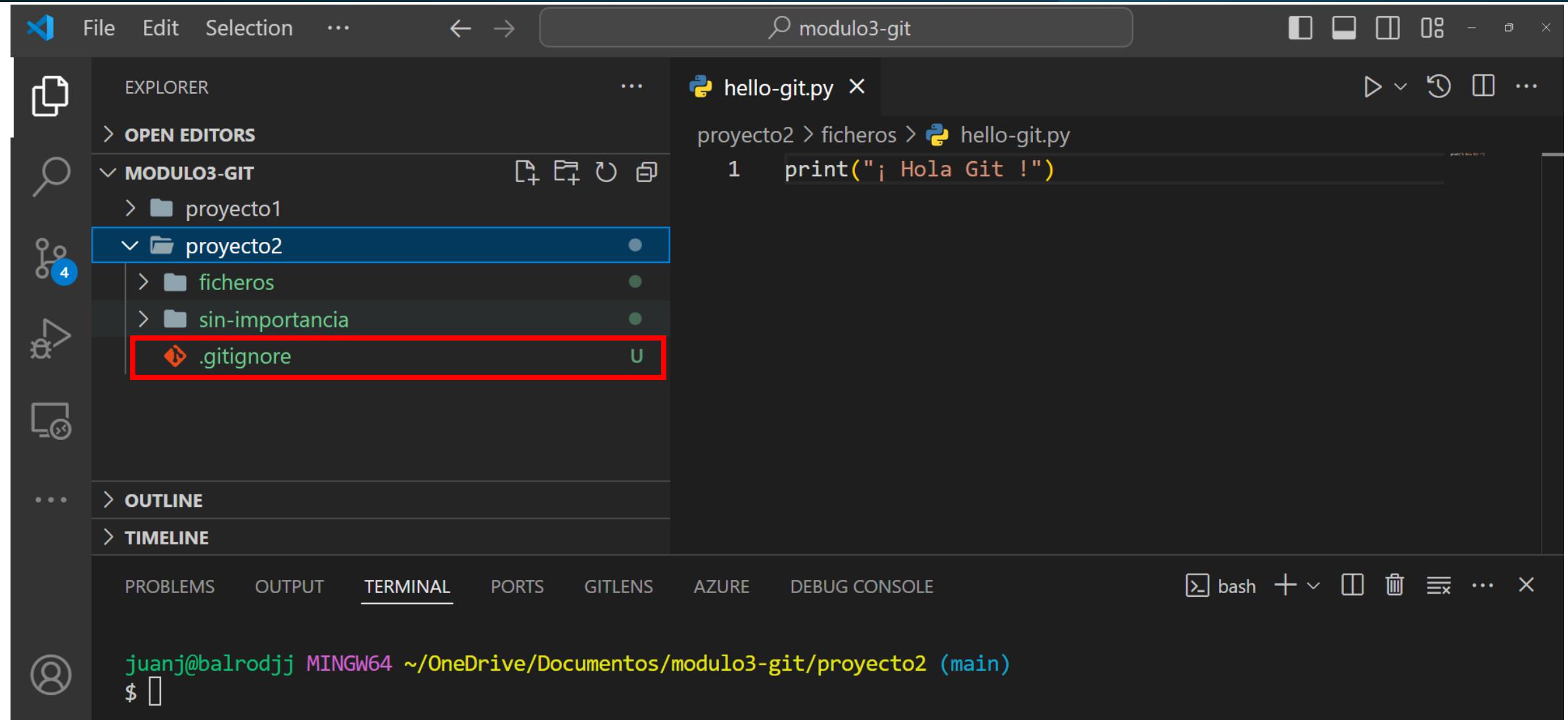
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ cd ..
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ █
```

Creamos un ‘fichero’ con nombre: **.gitignore**

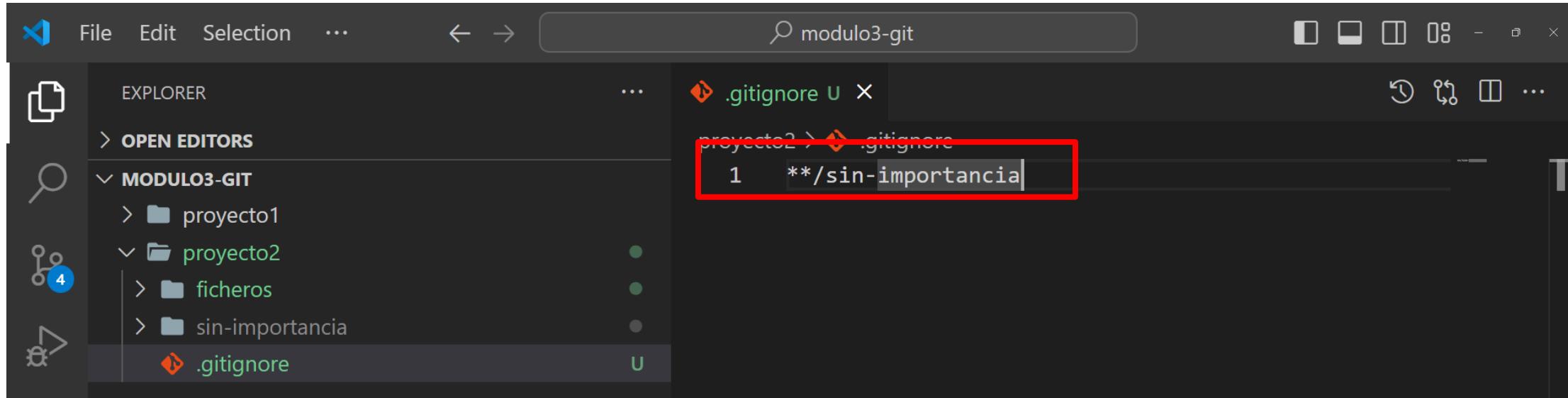
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ touch .gitignore
```

En vscode debería verse así:



Escribimos en el archivo .gitignore: **/sin-importancia

Guardar cambios



Git status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    ficheros/hello-git.txt

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

Añadimos al staging area a .gitignore y luego pedimos status

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git add .gitignore
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ficheros/hello-git.txt
```

Confirmamos (commit) los cambios.

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git commit -m "Se ha creado .gitignore y añadido el directorio /sin-importancia"
[main 1f9fc7c] Se ha creado .gitignore y añadido el directorio /sin-importancia
 1 file changed, 1 insertion(+)
create mode 100644 .gitignore
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ficheros/hello-git.txt

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

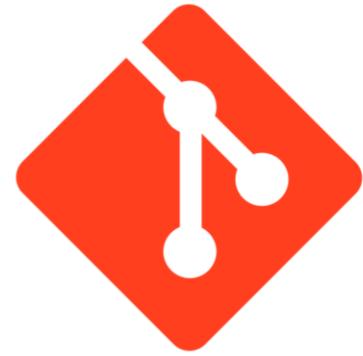
Vamos a añadir al staging área el archivo: hello-git.txt y luego lo confirmamos (commit)

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ cd ficheros

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add hello-git.txt

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "Se ha confirmado el archivo vacio hello-git.txt"
[main d355ba7] Se ha confirmado el archivo vacio hello-git.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ficheros/hello-git.txt
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
nothing to commit, working tree clean
```



git

3.13 Git reset

El comando git reset te permite restablecer tu estado actual a un estado específico. Puedes restablecer el estado de archivos específicos, así como el de toda una rama. Esto es útil si aún no has subido tu commit a GitHub o a otro repositorio remoto.

Antes de aplicar git reset, vamos a crear un par de ficheros más desde la terminal de git y hacemos cambios sobre el fichero hello-git.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ touch hola2.txt hola3.txt
```

Status

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hola2.txt
    hola3.txt

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

Añadimos cambios al staging area:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add .
```

Status

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola2.txt
    new file:   hola3.txt
```

Confirmamos cambios:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "se han creado dos archivos vacíos hola2.txt y hola3.txt"
[main 12ebbcb] se han creado dos archivos vacíos hola2.txt y hola3.txt
 2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ficheros/hola2.txt
create mode 100644 ficheros/hola3.txt
```

Status

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
nothing to commit, working tree clean
```

Hacer cambios sobre el archivo **hello-git.txt**

Guardar el fichero.

The screenshot shows a dark-themed code editor interface. At the top, there is a navigation bar with icons for File, Edit, Selection, and a search bar containing the text "modulo3-git". Below the navigation bar is the main workspace. On the left side is the Explorer sidebar, which displays a project structure under "MODULOS-GIT". The "ficheros" folder contains several files: "hello-git.py", "hello-git.txt", "hola2.txt", "hola3.txt", and ".gitignore". The "hello-git.txt" file is currently selected and open in the editor. The editor's status bar at the bottom right shows the path "modulo3-git/hello-git.txt" and the text "Primera línea: hola git".

Añadir cambios y confirmar: git commit -a

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -a
```

The screenshot shows a terminal window with the following details:

- Terminal tab is selected.
- Command: `git commit -a` is run.
- Output: A nano editor session is shown for the file `hello-git.txt`. The content is:

```
GNU nano 7.2      C:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto2/.git/COMMIT_EDITMSG      Modified
cambios en la primera linea del fichero hello-git.txt
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Accounts
# On branch main

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File     ^\ Replace        ^U Paste         ^J Justify       ^/ Go To Line   M-E Redo
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -a
[main fda2f62] cambios en la primera linea del fichero hello-git.txt
 1 file changed, 1 insertion(+)
```

git log --oneline:

```
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Resulta que se cometió un error y los archivos que se crearon y la edición de hello-git.txt no era la correcta

Vamos a ver el histórico de los commits:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 se ha confirmado el archivo vacio hello-git.txt
1f9fc7c se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Supongamos que queremos volver ‘en un punto en el tiempo’ en **d355ba7**, pero esta vez deshaciendo los commits. Podemos escribir: **git reset d355ba7**

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reset d355ba7
Unstaged changes after reset:
M      ficheros/hello-git.txt
```

git status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
on branch main
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:   ficheros/hello-git.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ficheros/hola2.txt
    ficheros/hola3.txt

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

Nos ha devuelto los cambios al working directory

Comparación de logs:

Antes del reset:

```
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcbe se han creado dos archivos vacíos hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Después del reset:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
d355ba7 (HEAD -> main) se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

De vuelta al status. Vamos a enviar todo al ‘staging area’.

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
on branch main
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:   ficheros/hello-git.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ficheros/hola2.txt
    ficheros/hola3.txt

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

Escribe: **git add .**

Status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   ficheros/hello-git.txt
    new file:   ficheros/hola2.txt
    new file:   ficheros/hola3.txt
```

Si quiero volver al estado donde tenía todos los commits se puede usar git reflog para buscar el SHA-1 del commit anterior al reset.

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reflog
d355ba7 (HEAD -> main) HEAD@{0}: reset: moving to d355ba7
fda2f62 HEAD@{1}: reset: moving to HEAD
fda2f62 HEAD@{2}: commit: cambios en la primera linea del fichero hello-git.txt
12ebbcb HEAD@{3}: commit: se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 (HEAD -> main) HEAD@{4}: commit: Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c HEAD@{5}: commit: Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa HEAD@{6}: commit (initial): Primer commit: he añadido un print
```

Escribimos: **git checkout fda2f62**

Escribimos: git checkout fda2f62

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git checkout fda2f62
Note: switching to 'fda2f62'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at fda2f62 cambios en la primera linea del fichero hello-git.txt

git log --oneline

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 ((fda2f62...))
$ git log --oneline
fda2f62 (HEAD) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 (main) Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Hemos logrado que aparezcan los commits perdidos, pero el puntero Head no está en la rama main.

Para devolver los commits a la rama main tenemos que crear una rama experimental y luego fusionar la rama experimental con la rama main y finalmente borrar la rama experimental.

Creamos una rama, la llamaremos dev: git checkout -b dev

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 ((fda2f62...))
$ git checkout -b dev
Switched to a new branch 'dev'
```

git log --oneline

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (dev)
$ git log --oneline
fda2f62 (HEAD -> dev) cambios en la primera linea del fichero hello-git.txt
12ebcb se han creado dos archivos vacíos hola2.txt y hola3.txt
d355ba7 (main) Se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Nos movemos la rama main:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (dev)
$ git checkout main
Switched to branch 'main'
```

git log --oneline

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
d355ba7 (HEAD -> main) Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Desde la rama main ‘fusionamos’ con la rama dev: git merge dev

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git merge dev
Updating d355ba7..fda2f62
Fast-forward
  ficheros/hello-git.txt | 1 +
  ficheros/hola2.txt     | 0
  ficheros/hola3.txt     | 0
  3 files changed, 1 insertion(+)
  create mode 100644 ficheros/hola2.txt
  create mode 100644 ficheros/hola3.txt
```

git log --oneline

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main, dev) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

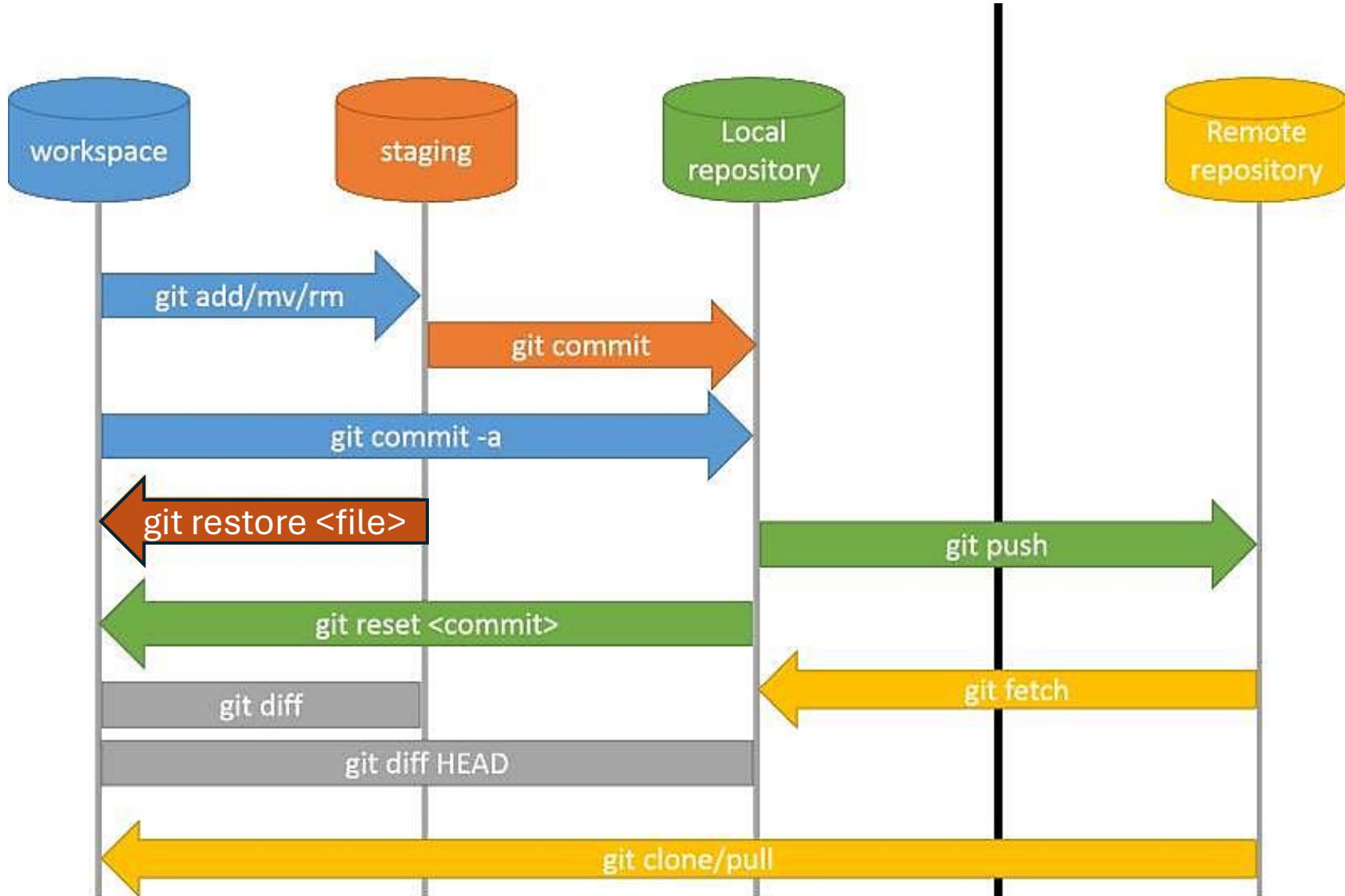
Ahora se puede borrar la rama dev:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git branch -d dev
Deleted branch dev (was fda2f62).
```

git log --oneline

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

¿Qué sabemos hasta ahora?



Git reset --hard:

El comando git reset --hard restaura a versiones anteriores de tus archivos afectando las tres zonas de trabajo de git: Working directory , staging área y local repository (o árbol de confirmaciones):

Al ejecutar git reset --hard, ocurre lo siguiente:

- El árbol de confirmaciones (local repository): Se mueve al estado especificado, descartando todos los commits posteriores al commit objetivo.
- El ‘staging área’: Se actualiza para reflejar el estado del commit objetivo. Los cambios en el índice se descartan.
- El directorio de trabajo: Se sincroniza con el estado del commit objetivo. Cualquier cambio no confirmado se descarta.

Precaución: Este modo elimina permanentemente los cambios no confirmados. Úsalo con cuidado.

Por ejemplo, si deseas volver al estado de un commit anterior, puedes usar:

```
git reset --hard <hash_del_commit_objetivo>
```

Reemplaza <hash_del_commit_objetivo> con el hash del commit al que deseas regresar. Ten en cuenta que esto descartará todos los cambios no confirmados desde ese commit.

Cuando se usa git reset --hard y no se añade ningún hash git te devuelve al estado del proyecto del commit inmediatamente anterior . En otras palabras, al ejecutar git reset --hard sin un hash de commit específico, eliminás permanentemente todos los cambios no confirmados y los commits posteriores al commit actual

¿Cuál es la diferencia entre `git reset <hash>` y `git reset --hard <hash>` ?

- `git reset <hash>` mantiene los cambios en el **directorio de trabajo**.
- `git reset --hard <hash>` descarta los cambios en el **directorio de trabajo**.

Ejemplo. Hasta ahora en nuestro log tenemos esto:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Queremos volver a este momento: d355ba7, por tanto, escribimos:
git reset --hard d355ba7

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reset --hard d355ba7
HEAD is now at d355ba7 Se ha confirmado el archivo vacio hello-git.txt
```

Pedimos status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
nothing to commit, working tree clean
```

git log --oneline

```
d355ba7 (HEAD -> main) Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

¿Cómo recuperamos los commits ‘borrados’?
Igual que en el ejemplo anterior, con ayuda de: git reflog

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reflog
d355ba7 (HEAD -> main) HEAD@{0}: reset: moving to d355ba7
fda2f62 HEAD@{1}: merge dev: Fast-forward
d355ba7 (HEAD -> main) HEAD@{2}: checkout: moving from dev to main
```

Por lo tanto, escribimos: **git reset --hard fda2f62**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reset --hard fda2f62
HEAD is now at fda2f62 cambios en la primera linea del fichero hello-git.txt
```

Verificar con git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacíos hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

git reset --soft <hash>

Este modo de reset no afecta el ‘staging area’ ni al ‘working directory’ en absoluto (pero reposiciona el HEAD al commit especificado, al igual que todos los modos).

- Deja todos tus archivos modificados como “Cambios a confirmar”, como lo mostraría git status.
- Es útil cuando deseas desconfirmar un commit pero mantener los cambios listos para volver a confirmar.
- Por ejemplo: git reset --soft HEAD~1 (para desconfirmar el último commit).

Ejemplo. Vamos a volver a 1 commit anterior al actual. Antes, veamos nuestro log:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcbe se han creado dos archivos vacíos hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Vamos a volver a ‘un punto en el tiempo’ en 12ebbcbe. Por lo tanto, escribimos: **git reset --soft 12ebbcbe**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reset --soft 12ebbcbe
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ █
```

Vamos a ver como quedó nuestro log:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
12ebbcb (HEAD -> main) se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   ficheros/hello-git.txt
```

Supongamos que queremos arreglar algunas cosas de ese fichero, una vez realizadas las modificaciones confirmamos los cambios.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git commit -m "correcciones"
[main 242962a] correcciones
 1 file changed, 1 insertion(+)
```

git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
242962a (HEAD -> main) correcciones
12ebccb se han creado dos archivos vacíos hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacío hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Si por casualidad queremos volver al commit previo al reset soft podemos usar git reset --hard <hash>

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git reset --hard fda2f62
HEAD is now at fda2f62 cambios en la primera linea del fichero hello-git.txt
```

git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
fda2f62 (HEAD -> main) cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

¿Cuál es la diferencia entre git reset soft <commit> y git reset <commit>?

1. git reset <commit> (por defecto usa --mixed)

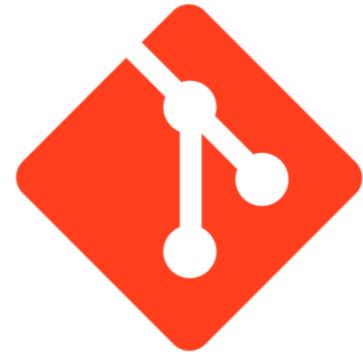
- HEAD: Se mueve a <commit>, es decir, cambia la referencia de la rama actual al commit especificado.
- Staging area (Índice): Se reinicia al estado del commit especificado. Los archivos que habían sido añadidos o modificados en el área de staging después de ese commit serán removidos del área de staging.
- Directorio de trabajo: No se modifica. Cualquier cambio en el directorio de trabajo permanece sin tocar, por lo que los archivos modificados siguen existiendo, pero no estarán en el staging area.

¿Cuál es la diferencia entre git reset soft <commit> y git reset <commit>?

2. git reset --soft <commit>

- HEAD: Se mueve a <commit>, igual que en el caso anterior.
- Staging area (Índice): No cambia. Todos los archivos que estaban en staging (es decir, listos para ser confirmados, commit) antes del reset siguen estando en staging.
- Directorio de trabajo: No se modifica. Todos los cambios en el directorio de trabajo permanecen intactos.

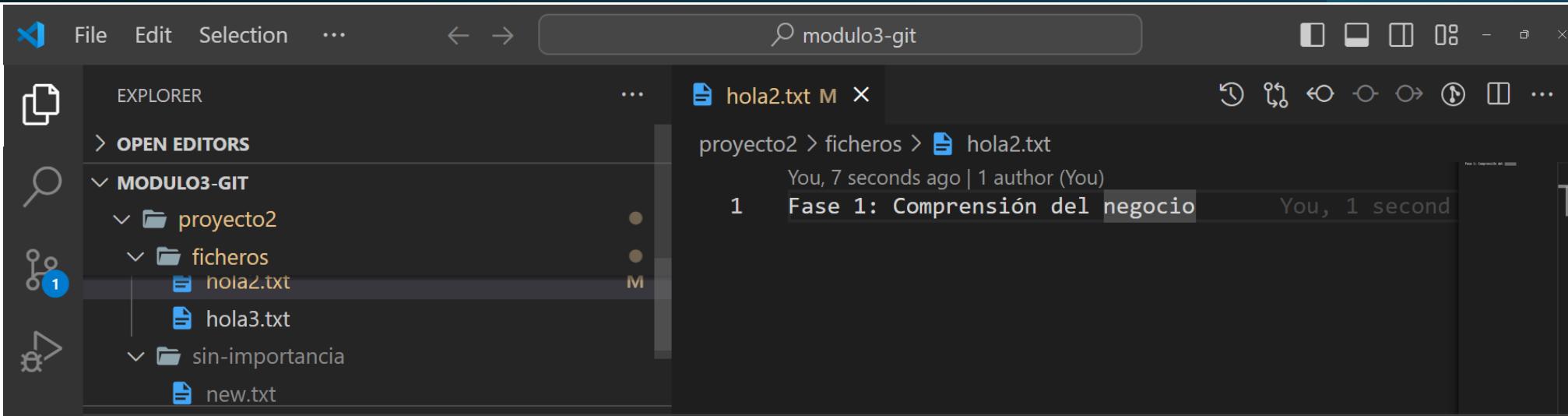
Desarrolla tus propios ejemplos para visualizar las diferencias entre git reset <commit>, git reset --soft <commit> y git reset --hard <commit>



git

3.14 Deshaciendo cambios antes del ‘staging area’

Hagamos una edición del archivo hola2.txt



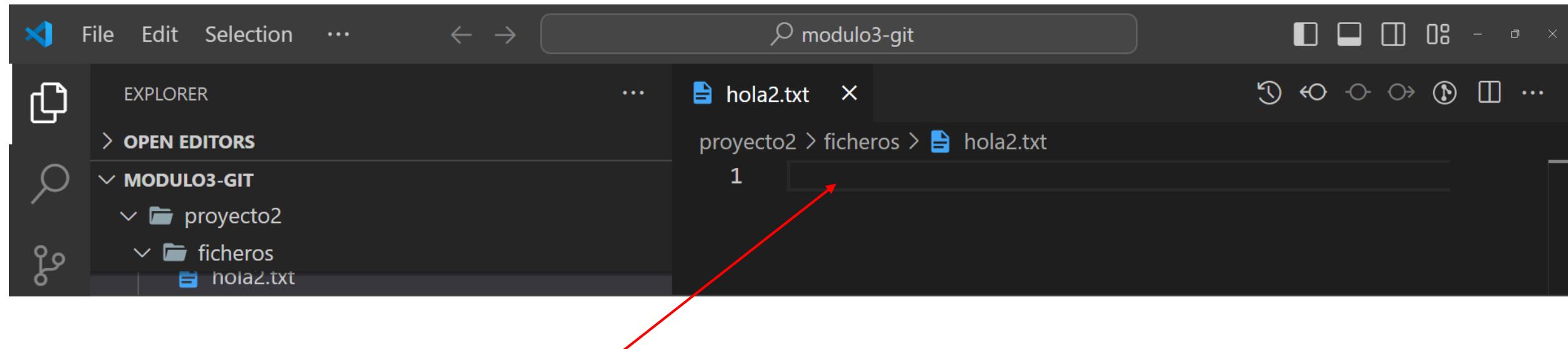
git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
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:   hola2.txt

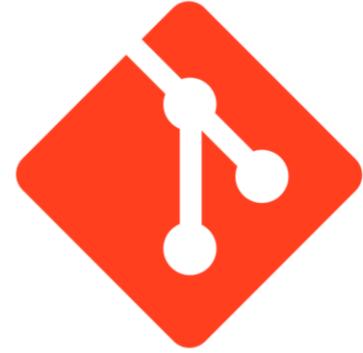
no changes added to commit (use "git add" and/or "git commit -a")
```

Supongamos que quiero deshacer ese cambio.
Escribimos: **git restore hola2.txt**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git restore hola2.txt
```



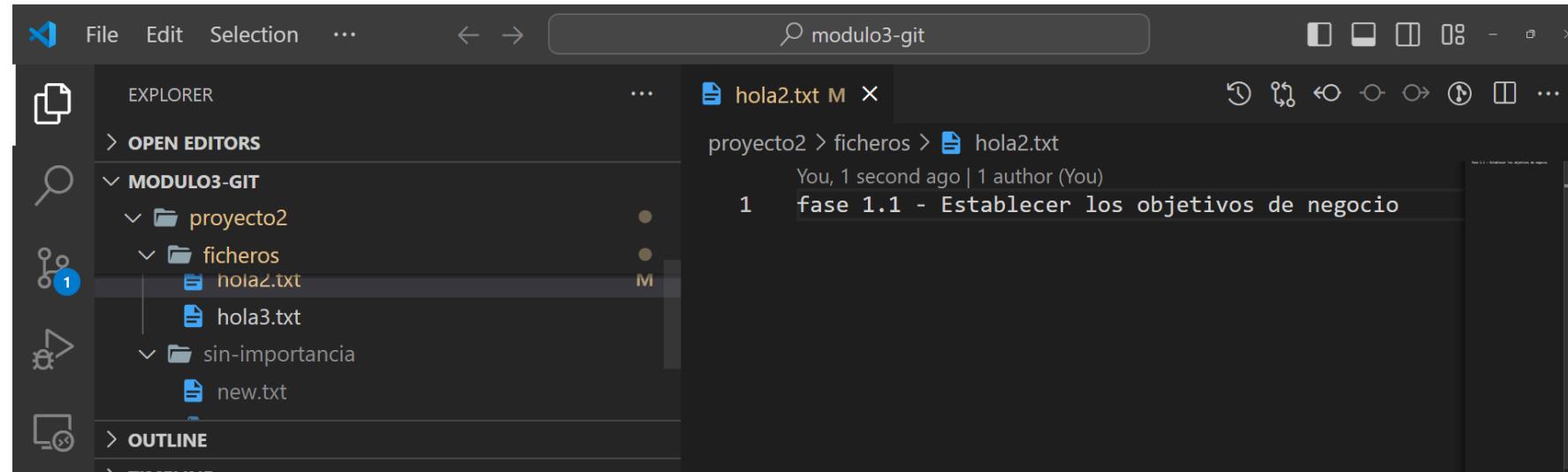
Después del restore no hay nada en el fichero hola2.txt



git

3.15 Deshaciendo cambios antes del commit

Hagamos una edición del archivo **hola2.txt** , luego guardar



git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
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:   hola2.txt

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

Añadir archivo al ‘staging area’: git add hola2.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add hola2.txt

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hola2.txt
```

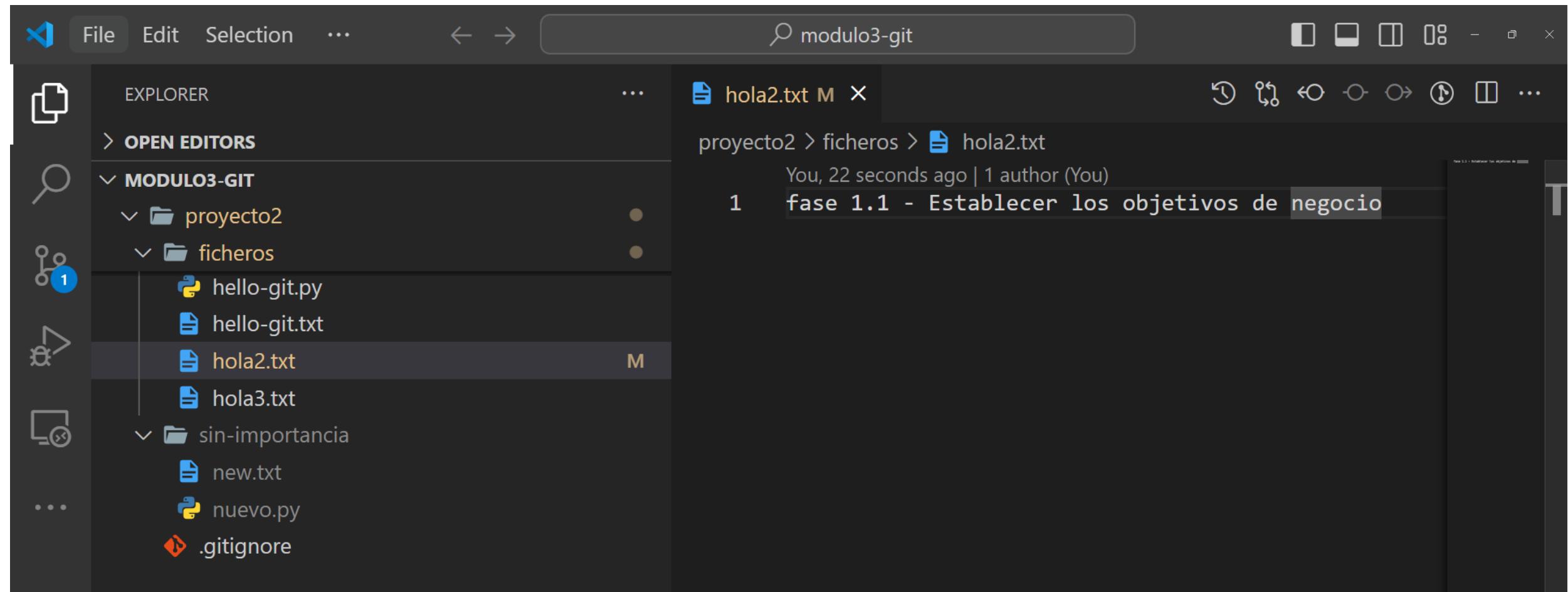
Supongamos que no me interesa confirmar (commit) el último cambio que hice sobre el archivo **hola2.txt** así que, quiero sacarlo del ‘staging area’: **git restore --staged hola2.txt**

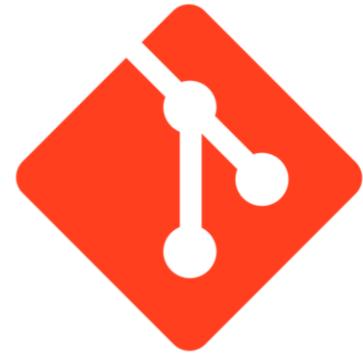
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git restore --staged hola2.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
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:   hola2.txt

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

`git restore --staged <file>` nos devuelve el ‘add’ utilizado y nos devuelve al directorio de trabajo sin borrar los cambios en el archivo:





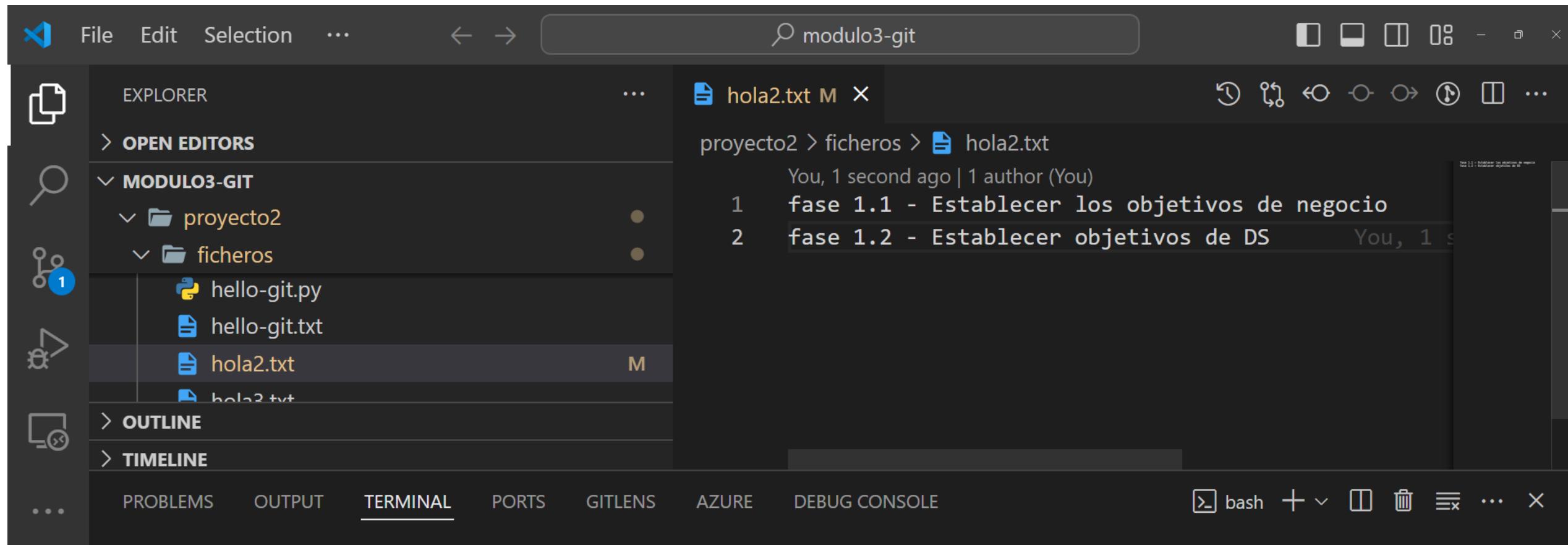
git

3.16 Deshaciendo commits no deseados

Si a pesar de todo hemos hecho un commit y nos hemos equivocado, podemos deshacerlo con la orden **git revert**.

- **git revert** sirve para deshacer los cambios introducidos por un commit específico en el historial de tu repositorio de Git.
- En vez de eliminar directamente el commit, crea un nuevo commit que básicamente revierte los efectos del commit original.
- Esto mantiene un historial de Git limpio y rastreable, dejando claro qué cambios se hicieron y por qué.

Modificamos otra vez el archivo hola2.txt como antes pero ahora sí hacemos commit:



git add hola2.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add hola2.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hola2.txt
```

git commit -m “Añadida fases 1.1 y 1.2 en líneas 1 y 2”

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "Añadida fases 1.1 y 1.2 en lineas 1 y 2"
[main 10f48c6] Añadida fases 1.1 y 1.2 en lineas 1 y 2
 1 file changed, 2 insertions(+)
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
nothing to commit, working tree clean
```

¿ Cómo usar git revert ?

1. Identifica el commit a revertir: Usa **git log** para ver tu historial de commits y busca el hash del commit (identificador único) del commit que deseas revertir.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git log --oneline
10f48c6 (HEAD -> main) Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

2. Ejecuta el comando git revert <hash>. En este caso escribimos: **git revert 10f48c6**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git revert 10f48c6
```

Corregir el mensaje del commit después de Revert cambiar por : “Ups...he cometido un error en este commit”

The screenshot shows a terminal window with the nano 7.2 editor open. The title bar indicates the file path: C:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto2/.git/COMMIT_EDITMSG. The main content of the editor is:

```
Revert "Añadida fases 1.1 y 1.2 en lineas 1 y 2"

This reverts commit 10f48c690317d7cf786f23205b9be5945a395431.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Changes to be committed:
#       modified:   ficheros/hola2.txt
#
```

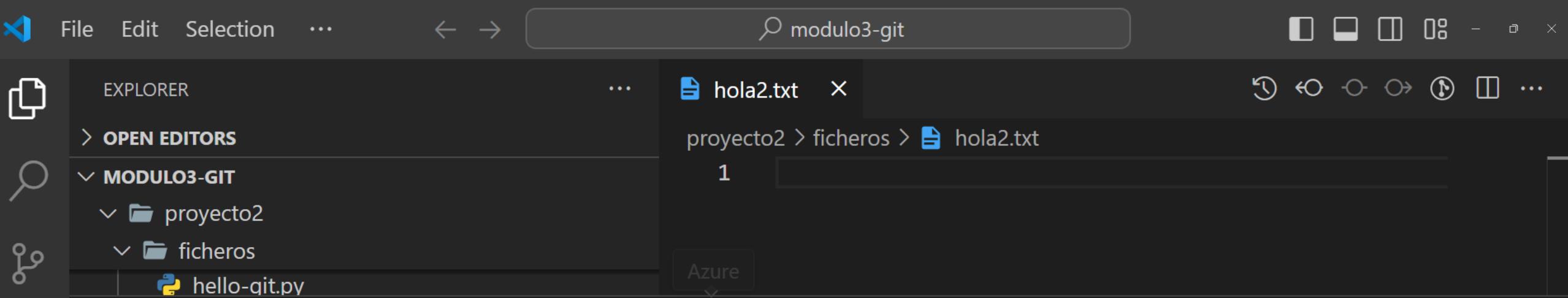
At the bottom of the screen, there is a status bar with various keyboard shortcuts. A green box highlights the text "[Read 11 lines]".

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location	M-U Undo
^X Exit	^R Read File	^V Replace	^U Paste	^J Justify	^/ Go To Line	M-E Redo

Salida esperada:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git revert 10f48c6
[main d9931b3] Revert "Ups...he cometido un error en este commit"
 1 file changed, 2 deletions(-)
```

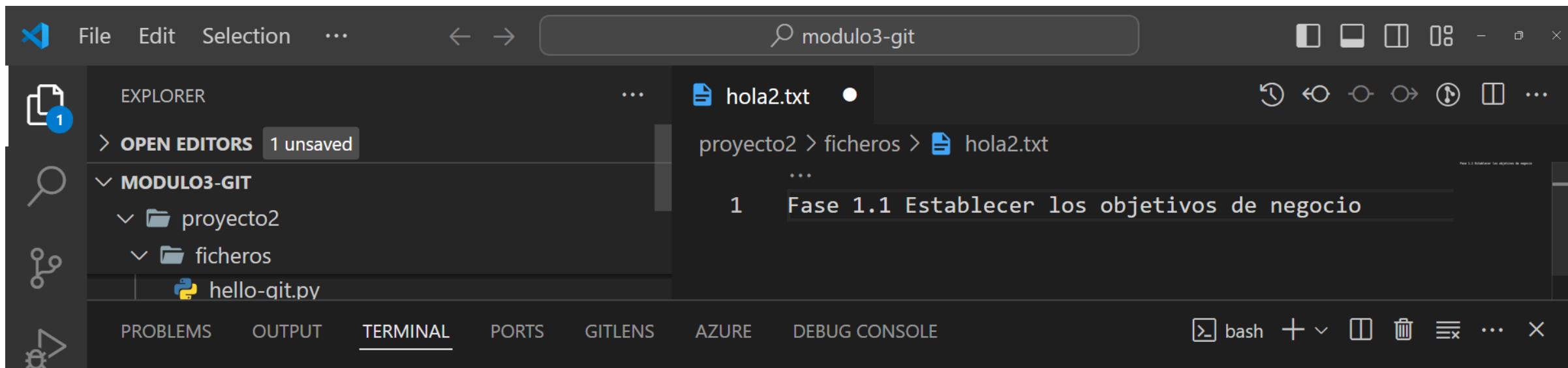
Verás que, después del revert, las líneas editadas durante ese commit se han borrado:



Así se ve en el log: git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git log --oneline
d9931b3 (HEAD -> main) Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

3. Corrige los cambios de tu archivo y vuelve a confirmar (commit).



```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git add hola2.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "Añadida Fase 1.1 en la primera linea del archivo hola3.txt"
[main cfe5fb1] Añadida Fase 1.1 en la primera linea del archivo hola3.txt
 1 file changed, 1 insertion(+)
```



git

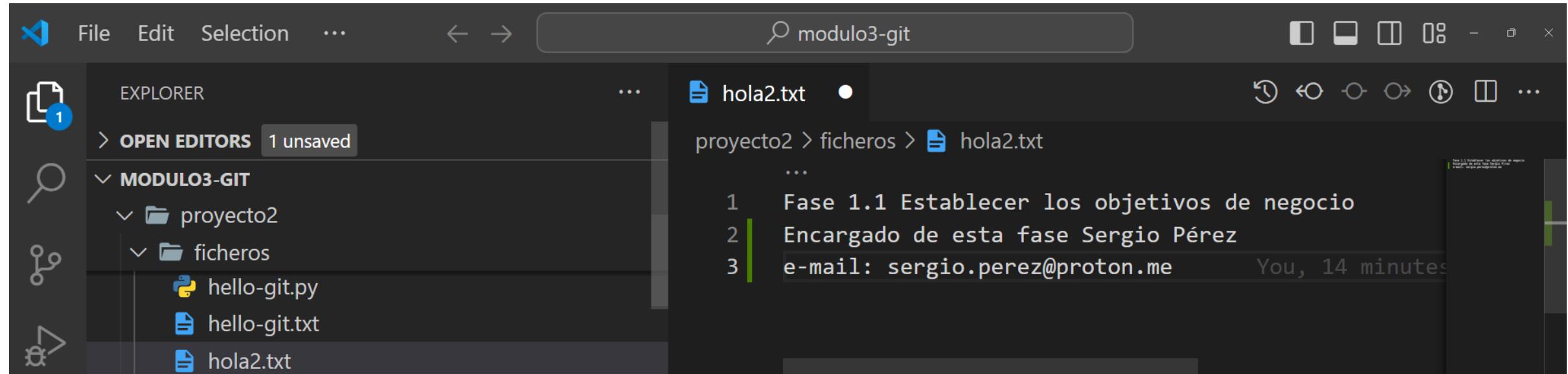
3.17 Modificar un commit

Se puede usar para corregir el mensaje dejado en un commit: `git commit --amend -m "mensaje a corregir"`

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit --amend -m "Añadida Fase 1.1 en la primera linea del archivo hola2.txt"
[main 6cb9316] Añadida Fase 1.1 en la primera linea del archivo hola2.txt
Date: Wed Apr 17 12:48:32 2024 +0200
1 file changed, 1 insertion(+)
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git log --oneline
6cb9316 (HEAD -> main) Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

Vamos a editar de nuevo el archivo hola2.txt



Un comando para añadir directamente al repositorio local y el mensaje sin llamar al editor es:

git commit -a -m “Añadido el encargado de la fase en la línea 2”

Hace en un solo comando: `git add + git commit -m “mensaje”`

git commit -a -m “Añadido el encargado de la fase en la línea 2”

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -a -m "Añadido el encargado de la fase en la linea 2"
[main 603b91d] Añadido el encargado de la fase en la linea 2
 1 file changed, 3 insertions(+), 1 deletion(-)
```

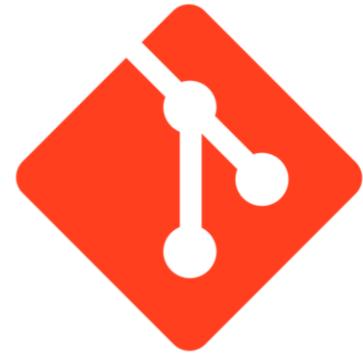
Resulta que el mensaje del commit está incompleto o mal escrito, entonces podemos usar **git commit --amend -m “mensaje”** para corregir el mensaje:

git commit --amend -m “Añadido el nombre y el e-mail del encargado de la fase 1.1 en las líneas 2 y 3 del archivo hola2.txt”

Salida esperada:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit --amend -m "Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt"
[main ac10391] Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
Date: Wed Apr 17 13:18:48 2024 +0200
1 file changed, 3 insertions(+), 1 deletion(-)
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git log --oneline
ac10391 (HEAD -> main) Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```



git

3.18 Mover un archivo a otro directorio con git

Para mover archivos usaremos la orden git mv:

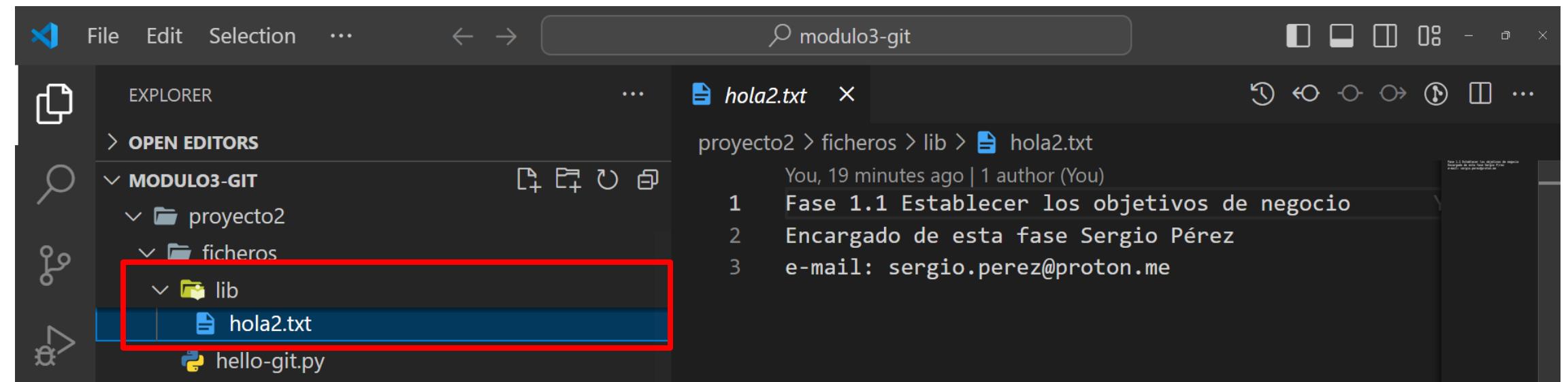
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ mkdir lib
```

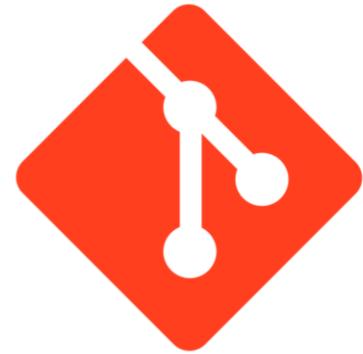
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git mv hola2.txt lib
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    hola2.txt -> lib/hola2.txt
```

Confirmar cambios, notar que en el editor que el archivo se ha movido al nuevo directorio lib

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -m "creación de carpeta lib y se ha movido hola2.txt a /lib"
[main 45255f2] creaci|n de carpeta lib y se ha movido hola2.txt a /lib
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename ficheros/{ => lib}/hola2.txt (100%)
```





git

3.19 Borrar un archivo

Se puede borrar un archivo usando `git rm`

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ mkdir lib2
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ cd lib2
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ touch hola4.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ./
nothing added to commit but untracked files present (use "git add" to track)
```

git add hola4.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git add hola4.txt

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola4.txt
```

git commit

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git commit -m "Se ha creado lib2/hola4.txt"
[main c4fd413] Se ha creado lib2/hola4.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ficheros/lib2/hola4.txt
```

git rm hola4.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git rm hola4.txt
rm 'ficheros/lib2/hola4.txt'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    hola4.txt
```

Informar que has borrado el archivo haciendo un commit:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git commit -m "Se ha borrado el archivo hola4.txt"
[main 656a399] Se ha borrado el archivo hola4.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 ficheros/lib2/hola4.txt
```

Practica 01.pdf.



git

Ramas (branches)
3.20 Crear una nueva rama

Cuando vamos a trabajar en una nueva funcionalidad, es conveniente hacerlo en una nueva rama, para no modificar la rama principal (main) y dejarla inestable.

Hay tres maneras de crear una rama:

- `git branch nombre-rama`
- `git checkout -b nombre-rama`
- `git switch -c nombre-rama`

Ejemplo: Crear una rama con nombre *feature*

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git branch feature
```

Para verificar las ramas (en local) que se tienen creadas escribimos: **git branch**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git branch
  feature
* main
```

Vamos a crear otra rama, esta vez usando:
git checkout -b nombre-rama

Escribimos: **git checkout -b guitar-errores**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git checkout -b guitarerrores
Switched to a new branch 'guitarerrores'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (guitarerrores)
$
```

Además de crear la rama git checkout te mueve hacia ella. Vamos a verificar las ramas que se tienen hasta ahora.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (guitarerrores)
$ git branch
  feature
  main
* guitarerrores
```

Vamos a borrar las ramas usando: **git branch -d nombre-rama**

Vamos a borrar la rama escribiendo: **git branch -d feature**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (quitarerrores)
$ git branch -d feature
Deleted branch feature (was 656a399).
```

Vamos a borrar la otra rama escribiendo: **git branch -d quitarerrores**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (quitarerrores)
$ git branch -d quitarerrores
error: cannot delete branch 'quitarerrores' used by worktree at 'C:/Users/juanj/OneDrive/Documentos/modulo3-
git/proyecto2'
```

La razón del error es
que no se puede
eliminar una rama si
estamos trabajando
en ella.



Antes de eliminar la rama **quitarerrores** debemos movernos a otra rama, en este caso nos vamos a pasar a la rama **main** escribiendo:

git checkout main

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (quitarerrores)
$ git checkout main
Switched to branch 'main'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$
```

Una vez que nos hemos pasado a la rama **main**, ya podemos eliminar la rama: **quitarerrores**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git branch -d quitarerrores
Deleted branch quitarerrores (was 656a399).
```

Verificamos las ramas que nos quedan:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git branch
* main
```

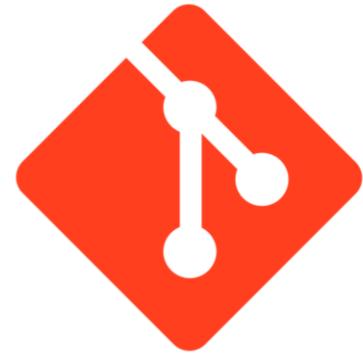
Aclaración:

El comando que se ha usado para borrar ramas (**git branch -d nombre-rama**) solo se utiliza para borrar ramas en repositorio **local**. Si se quiere borrar una rama que se halla en un repositorio **remoto** (por ejemplo, de Github) se tiene que escribir:

git push origin --delete nombre-rama

Esto será explicado cuando veamos GitHub

Práctica_02.ipynb.



git

3.21 Modificaciones en la rama secundaria

Vamos a movernos hasta el directorio lib2:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ cd ficheros/lib2
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$
```

Vamos a crear una rama que se llame: **desarrollo** y luego verificamos las ramas creadas con git branch

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git branch desarollo
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git branch
  desarollo
* main
```

Nos movemos hacia la rama **desarrollo** escribiendo:
git checkout desarrollo

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git checkout desarrollo
Switched to branch 'desarrollo'
```

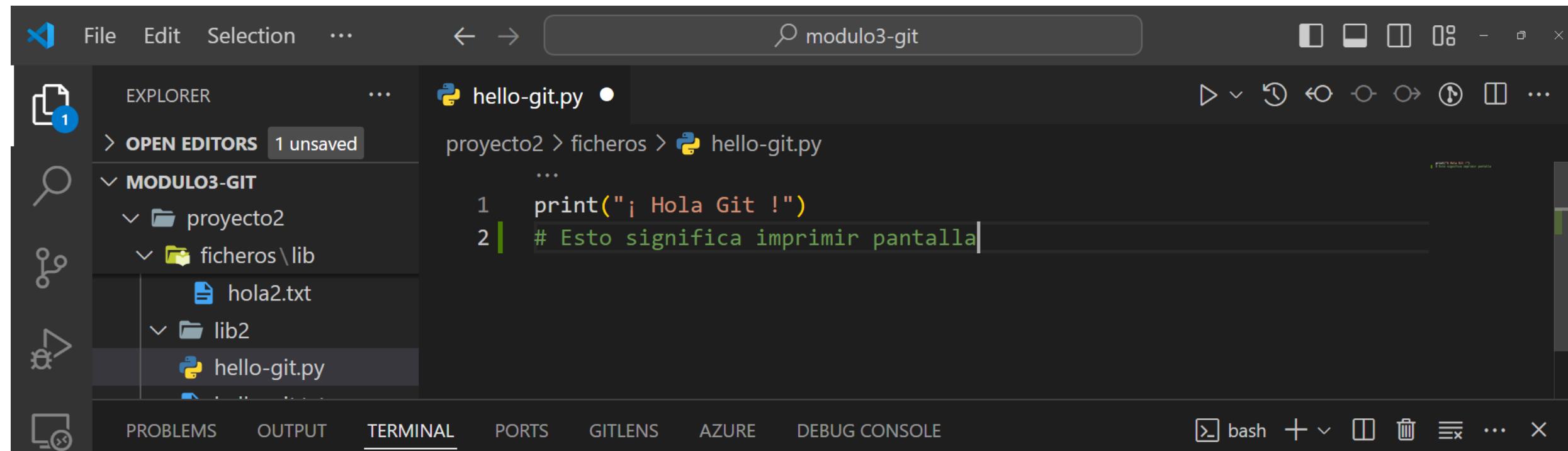
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$
```

Status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

Vamos a editar el archivo hello-git.py

Añadimos en la segunda línea: **# Esto significa imprimir pantalla**, luego guardar



Añadir (add) y confirmar (commit) : **git commit -a -m "comentario en línea 2, archivo hello-git.py"**

Salida esperada:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git commit -a -m "comentario en línea 2, archivo hello-git.py"
[desarrollo 042c504] comentario en linea 2, archivo hello-git.py
 1 file changed, 2 insertions(+), 1 deletion(-)
```

git status

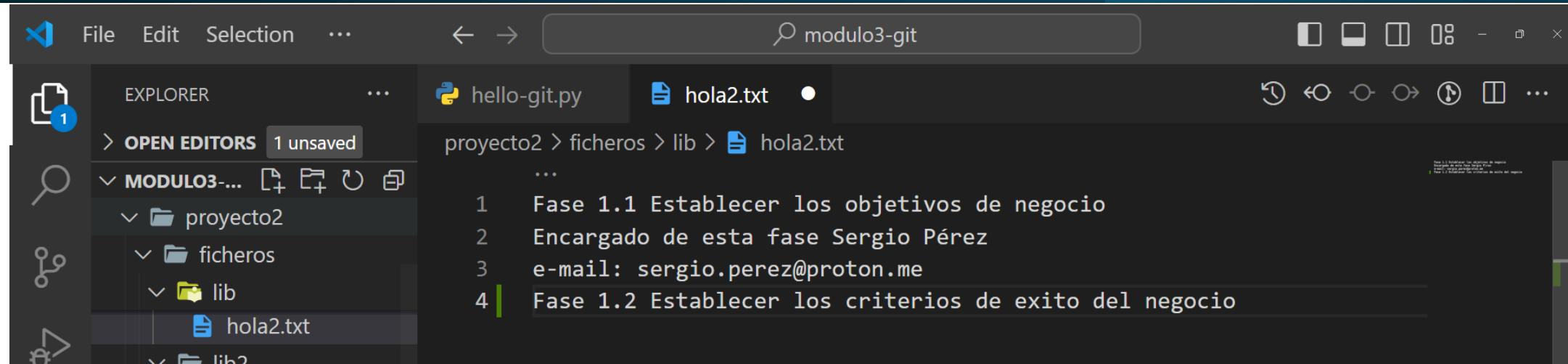
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git log --oneline
042c504 (HEAD -> desarrollo) comentario en línea 2, archivo hello-git.py
656a399 (main) Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
```

Vamos a editar el archivo: /lib/hola2.txt

No olvidar guardar después de editar



git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git status
On branch desarrollo
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:   ../lib/hola2.txt

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

git add hola2.txt

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ git add hola2.txt
fatal: pathspec 'hola2.txt' did not match any files
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (desarrollo)
$ cd ..
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ cd lib
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git add hola2.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ █
```

git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git status
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hola2.txt
```

git commit -m “añadida línea 4: Fase 1.2 en archivo /lib/hola2.txt”

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git commit -m "Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt"
[desarrollo 8edfe27] Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Vamos a ver los commits usando `git log--oneline`

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git log --oneline
8edfe27 (HEAD -> desarrollo) Añadida linea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en linea 2, archivo hello-git.py
656a399 (main) Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
d355ba7 Se ha confirmado el archivo vacio hello-git.txt
1f9fc7c Se ha creado .gitignore y añadido el directorio /sin-importancia
5a238aa Primer commit: he añadido un print
```

¿En qué momento nos pasamos a la rama desarrollo? git reflog

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git reflog
8edfe27 (HEAD -> desarrollo) HEAD@{0}: commit: Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 HEAD@{1}: commit: comentario en línea 2, archivo hello-git.py
656a399 (main) HEAD@{2}: checkout: moving from main to desarrollo
656a399 (main) HEAD@{3}: checkout: moving from quitarerrores to main
656a399 (main) HEAD@{4}: checkout: moving from main to quitarerrores
656a399 (main) HEAD@{5}: commit: Se ha borrado el archivo hola4.txt
c4fd413 HEAD@{6}: commit: Se ha creado lib2/hola4.txt
45255f2 HEAD@{7}: commit: creación de carpeta lib y se ha movido hola2.txt a /lib
```

Con la orden: **git checkout nombre-rama** podemos movernos entre ramas:

`git checkout main`

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
```

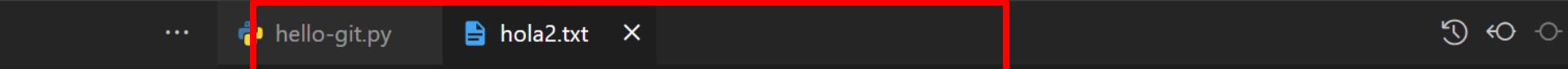
```
$ git checkout main  
Switched to branch 'main'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (main)
```

```
$
```

Vamos a permanecer un momento en la rama main

Vamos al fichero **hola2.txt**.
¿Has visto que la línea 4 no está?



```
projecto2 > ficheros > lib > hola2.txt
You, 23 hours ago | 1 author (You)
1 Fase 1.1 Establecer los objetivos de negocio
2 Encargado de esta fase Sergio Pérez
3 e-mail: sergio.perez@proton.me
```

git checkout desarrollo

Rama main

```
1 Fase 1.1 Establecer los objetivos de negocio
2 Encargado de esta fase Sergio Pérez
3 e-mail: sergio.perez@proton.me
4 Fase 1.2 Establecer los criterios de éxito del negocio
```

Rama desarrollo

Vamos de vuelta a la rama main y vamos a modificar el fichero

git checkout main

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (desarrollo)
$ git checkout main
Switched to branch 'main'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (main)
$ 
```

Nos movemos hasta lib2

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib (main)
$ cd ..

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ cd lib2

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ 
```

Vamos a editar el fichero hola3.txt

Añadir en línea 1: A continuación los equipos involucrados en el proyecto

```
projecto2 > ficheros > hola3.txt
You, 1 second ago | 1 author (You)
1 A continuación los equipos involucrados en el proyecto
```

No olvidar guardar...

git status

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git status
On branch main
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:   ../hola3.txt

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

Añadir y confirmar:

```
git commit -a -m "Añadida línea 1: a continuación... en el archivo hola3.txt"
```

Salida esperada

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git commit -a -m "Añadida línea 1: A continuación...en el archivo hola3.txt"
[main fbff1ea] Añadida línea 1: A continuación...en el archivo hola3.txt
 1 file changed, 1 insertion(+)
```

git status

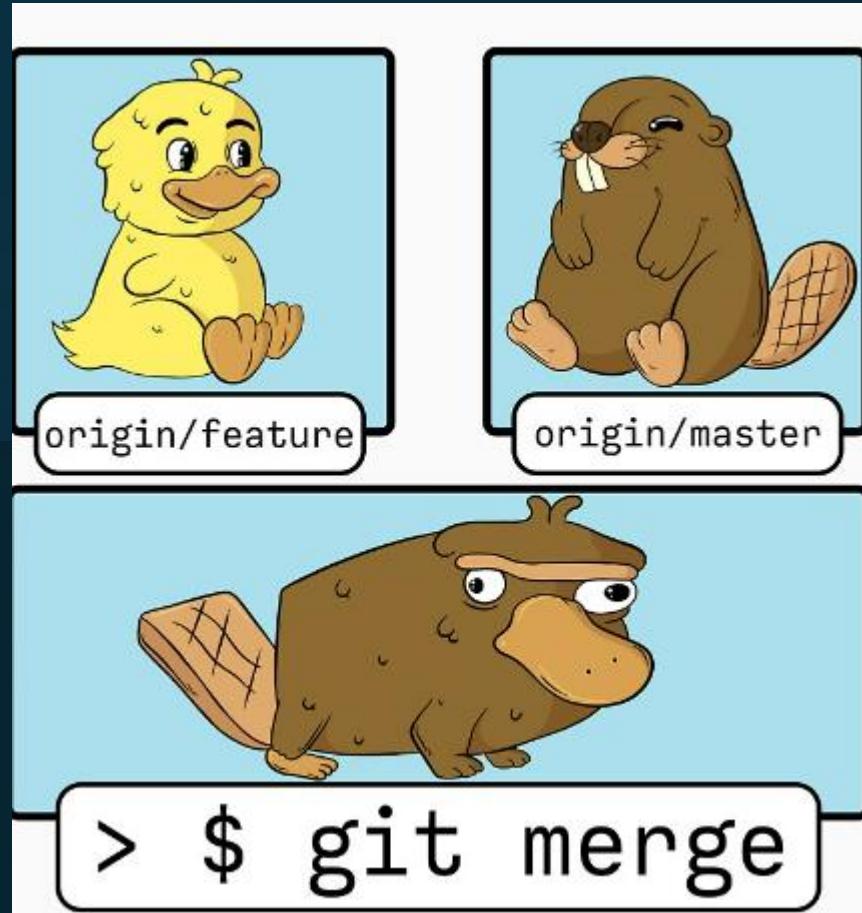
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros/lib2 (main)
$ git status
On branch main
nothing to commit, working tree clean
```

Práctica.



git

Fusión de ramas.
3.22 Unir ramas (git merge)



Podemos incorporar los cambios de una rama a otra con la orden:
git merge

Asegúrate de estar en la rama main antes de hacer el merge.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ |
```

Luego escribimos: **git merge desarrollo**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git merge desarrollo
```

Se abrirá el editor por defecto que tengas y añades el mensaje que va para el commit. Otra forma es escribiendo:

```
git merge nombre-rama -m "Mensaje personalizado del commit"
```

Se abrirá el editor, a continuación, añadimos el mensaje y después guardar y salir:

```
GNU nano 7.2          C:/Users/juanj/OneDrive/Documentos/modulo3-git/proyecto2/.git/MERGE_MSG
Merge branch 'desarrollo' para mejorar algunas características del proyecto
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

Algunos prefieren dejar el mensaje que trae por defecto: Merge Branch ‘desarrollo’

Una vez que guardamos y salimos, la salida esperada debería ser:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git merge desarrollo
Merge made by the 'ort' strategy.
  ficheros/hello-git.py | 3 +-+
  ficheros/lib/hola2.txt | 3 +-+
  2 files changed, 4 insertions(+), 2 deletions(-)
```

git log --oneline

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
62ce302 (HEAD -> main) Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida línea 1: A continuación...en el archivo hola3.txt
8edfe27 (desarrollo) Añadida linea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en línea 2, archivo hello-git.py
656a399 Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
```

git checkout desarrollo

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git checkout desarrollo
Switched to branch 'desarrollo'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (desarrollo)
$
```

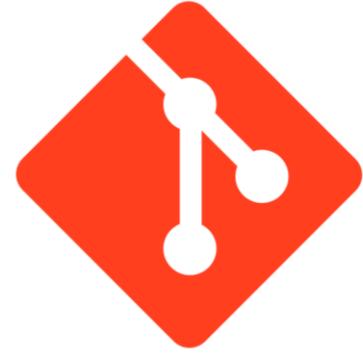
Comparación de logs:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git log --oneline
62ce302 (HEAD -> main) Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida línea 1: A continuación...en el archivo hola3.txt
8edfe27 (desarrollo) Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en línea 2, archivo hello-git.py
656a399 Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
```

Rama main

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2 (desarrollo)
$ git log --oneline
8edfe27 (HEAD -> desarrollo) Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en línea 2, archivo hello-git.py
656a399 Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
```

Rama desarrollo



git

3.23 Resolver conflictos entre ramas

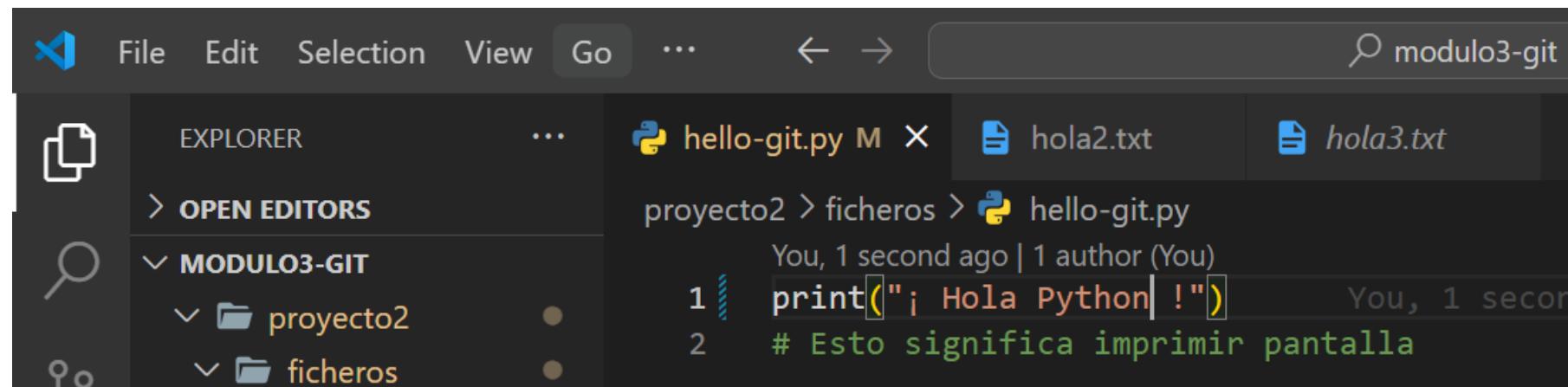
Un conflicto es cuando se produce una fusión (merge) que Git no es capaz de resolver.

Vamos a modificar contenido en la rama desarrollo para crear un conflicto con la rama main. Para empezar, debemos estar en la rama desarrollo, escribir: **git switch desarrollo**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git switch desarrollo
Switched to branch 'desarrollo'

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$
```

Una vez dentro de la rama desarrollo vamos a modificar el fichero hello-git.py



Guardar, luego añadir y confirmar:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git commit -a -m "Cambio de Git por Python"
[desarrollo 1fd186a] Cambio de Git por Python
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Vamos ahora a la rama main: **git switch main**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git switch main
Switched to branch 'main'

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ █
```

Vamos a comparar el contenido de hello-git.py desde ambas ramas:

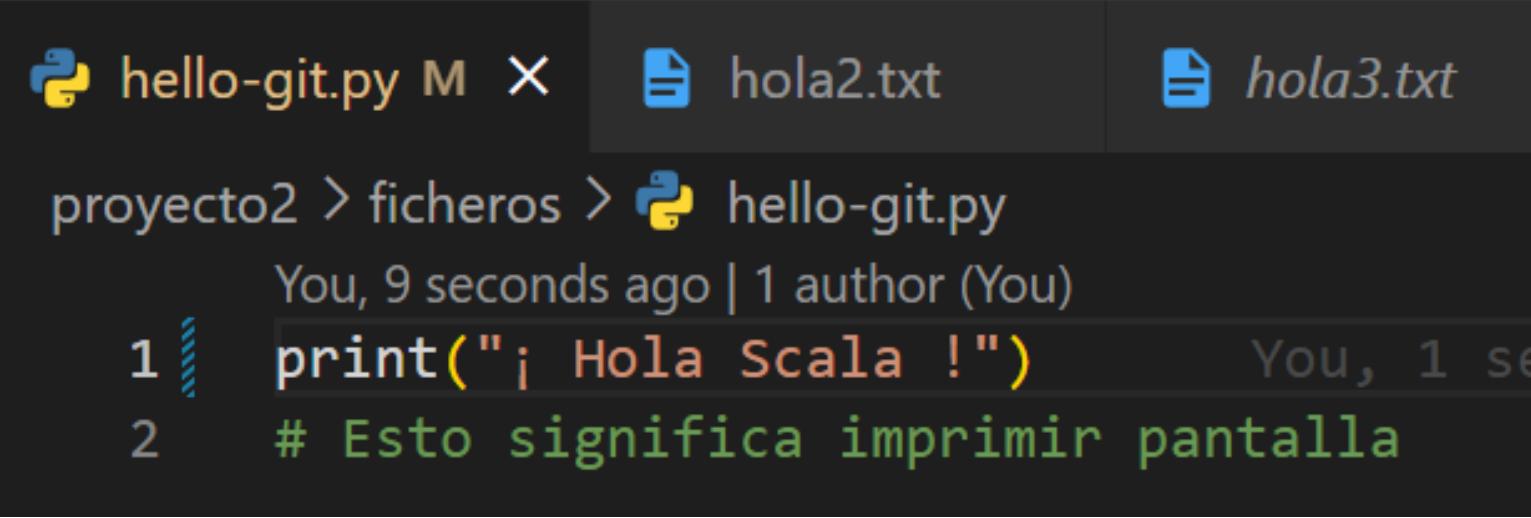
```
python hello-git.py X  hola2.txt  hola3.txt  
proyecto2 > ficheros > python hello-git.py  
You, 23 hours ago | 1 author (You)  
1 print("¡ Hola Git !")  
2 # Esto significa imprimir pantalla
```

Contenido del fichero **hello-git.py** de la rama **main**

Contenido del fichero hello-git.py de la rama **desarrollo**

```
python hello-git.py X  hola2.txt  hola3.txt  
proyecto2 > ficheros > python hello-git.py  
You, 7 minutes ago | 1 author (You)  
1 print("¡ Hola Python !")  
2 # Esto significa imprimir pantalla
```

Desde la rama main vamos a realizar cambios sobre el fichero **hello-git.py**



```
python hello-git.py M X  hola2.txt  hola3.txt
proyecto2 > ficheros > python hello-git.py
You, 9 seconds ago | 1 author (You)
1 print("¡ Hola Scala !")      You, 1 se
2 # Esto significa imprimir pantalla
```

Añadir y confirmar:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git commit -a -m "Cambio de Git por Scala"
[main 2a875dd] Cambio de Git por Scala
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Vamos a la rama desarrollo y hacemos un merge con la rama main

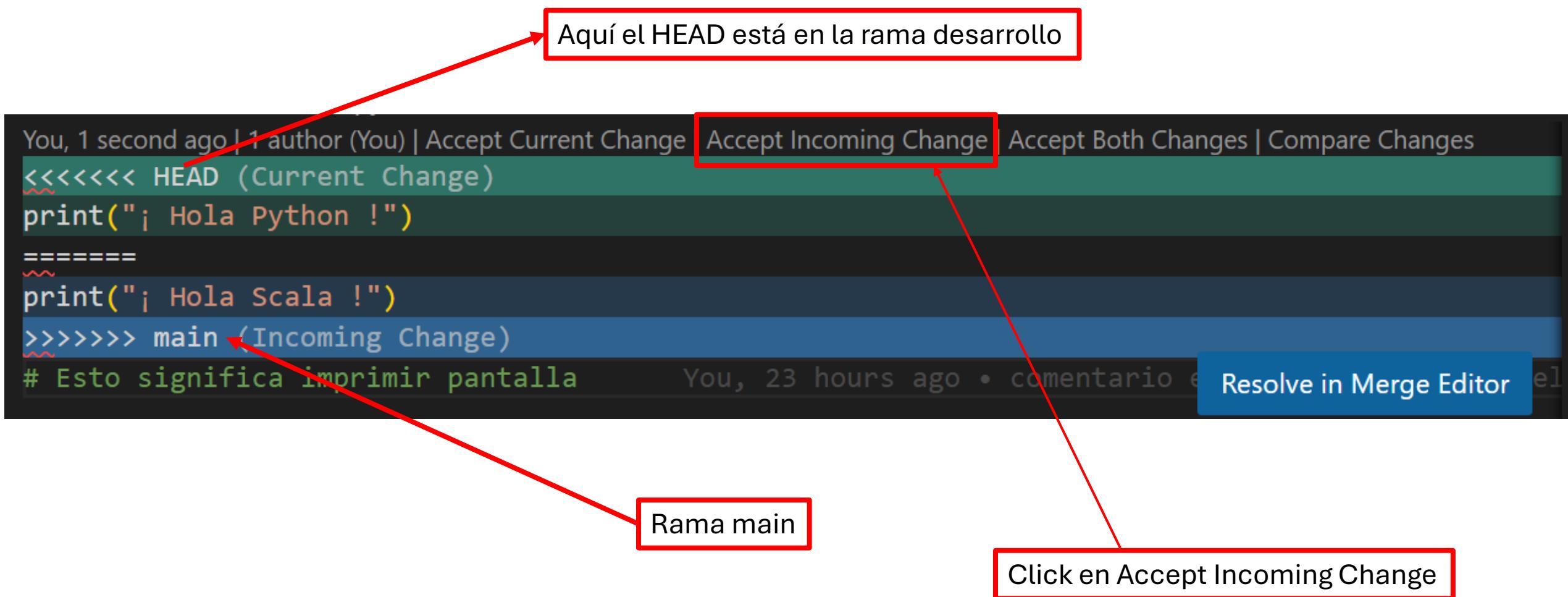
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git switch desarrollo
Switched to branch 'desarrollo'

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$
```

git merge main

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git merge main
Auto-merging ficheros/hello-git.py
CONFLICT (content): Merge conflict in ficheros/hello-git.py
Automatic merge failed; fix conflicts and then commit the result.
```

En vs code se ve lo siguiente:



Añadir y confirmar el merging:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo|MERGING)
$ git commit -a -m "Conflicto resuelto"
[desarrollo 683fe82] Conflicto resuelto

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$
```

```
683fe82 (HEAD -> desarrollo) Conflicto resuelto
2a875dd (main) Cambio de Git por Scala
1fd186a Cambio de Git por Python
62ce302 Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida linea 1: A continuación...en el archivo hola3.txt
8edfe27 Añadida linea 4:Fase 1.2... en archivo /lib/hola2.txt
```

Log de la rama desarrollo

Log de la rama main

```
2a875dd (HEAD -> main) Cambio de Git por Scala
62ce302 Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida linea 1: A continuación...en el archivo hola3.txt
8edfe27 Añadida linea 4:Fase 1.2... en archivo /lib/hola2.txt
```



git

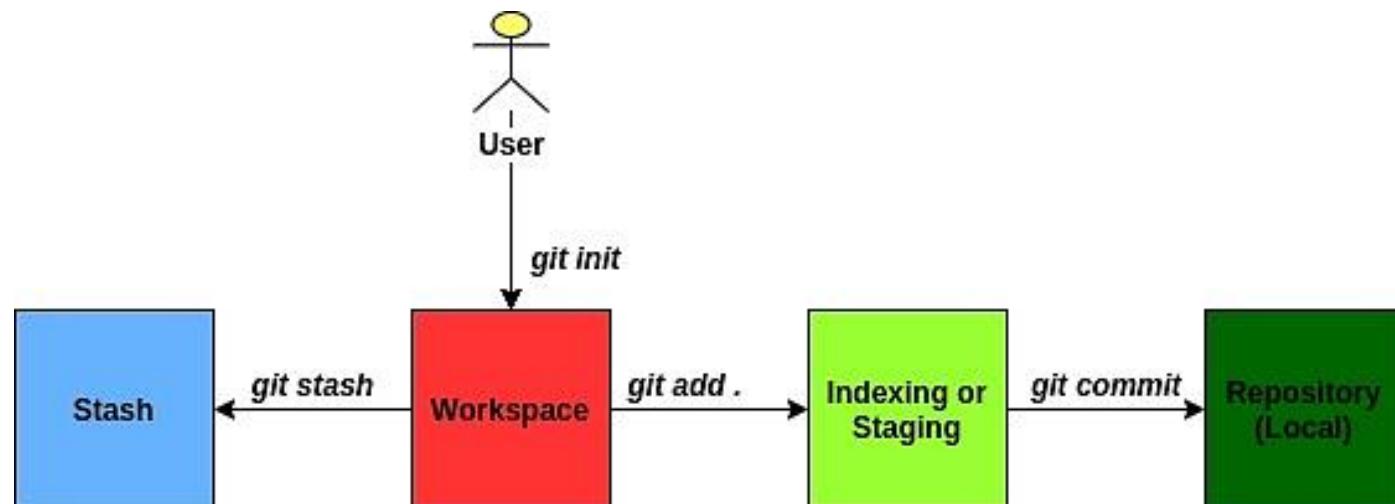
3.24 Git Stash.

git stash almacena temporalmente los cambios que has realizado en tu código para que puedas trabajar en otro proyecto y luego volver a aplicar esos cambios más tarde.

- Guarda temporalmente los cambios sin comprometerlos.
- Útil para cambiar de tarea o resolver problemas urgentes.
- Evita conflictos de fusión.
- Mantiene el historial de commits limpio.

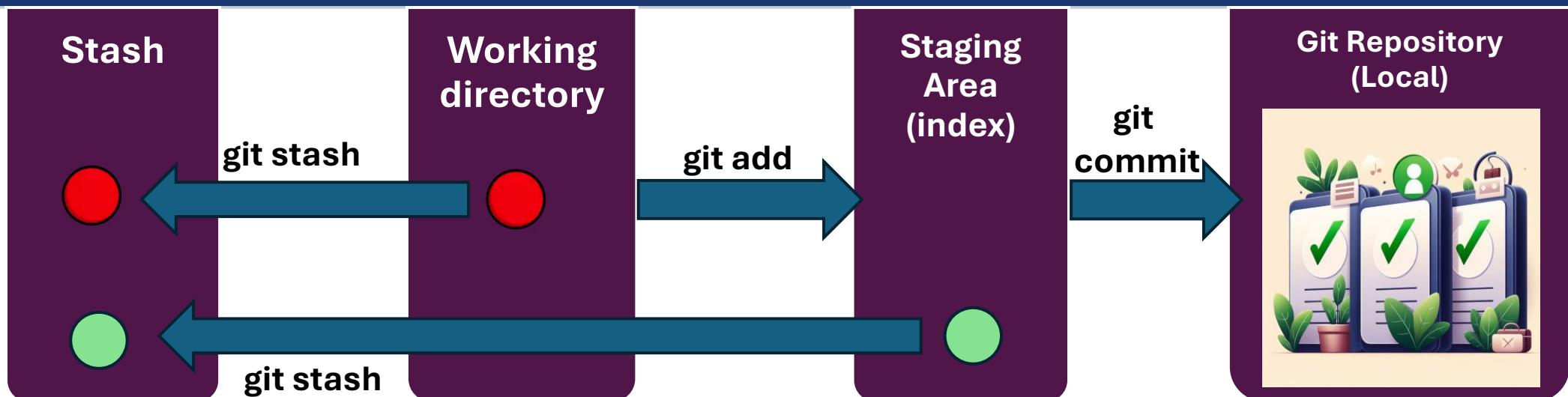
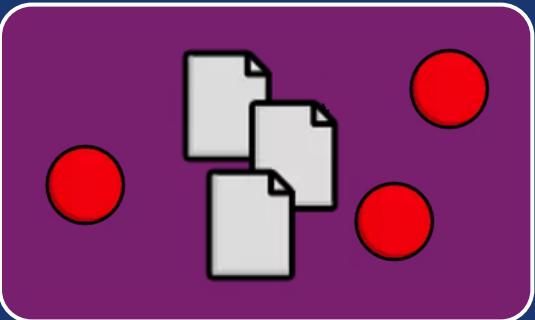
git stash

La ventaja de git stash es que guarda los cambios sin confirmar, tanto los preparados (staged-staging area) como los no preparados (unstaged-working directory), y los deshará en el código en el que estás trabajando.

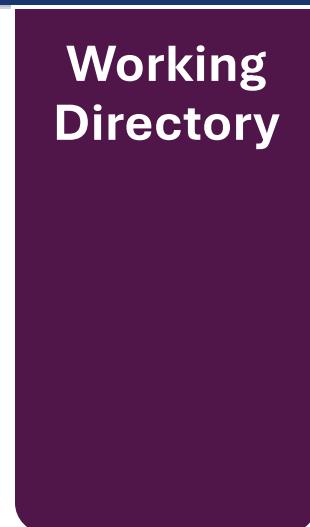
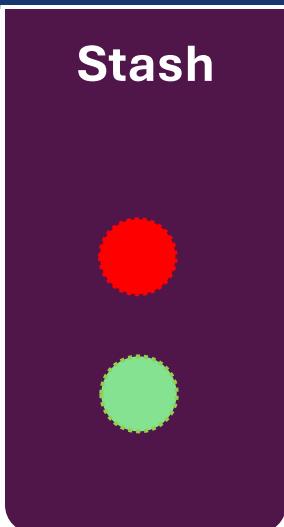
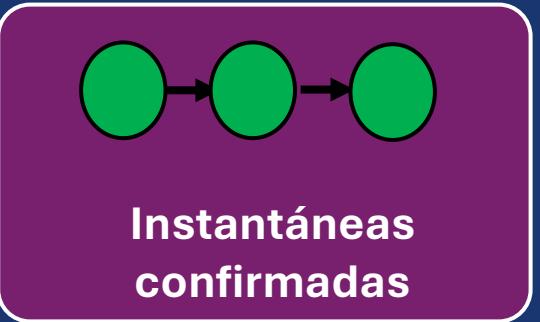
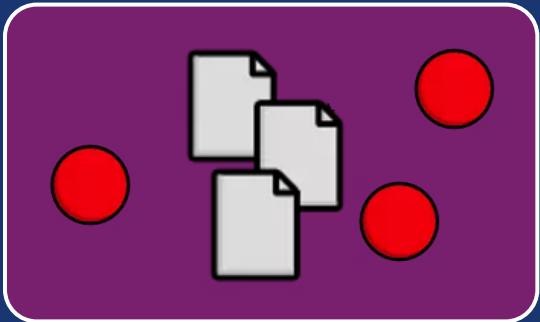


Stash es como un portapapeles para un proyecto. Cada copia del stash tiene un identificador de serie

Zonas de git antes y durante el stash

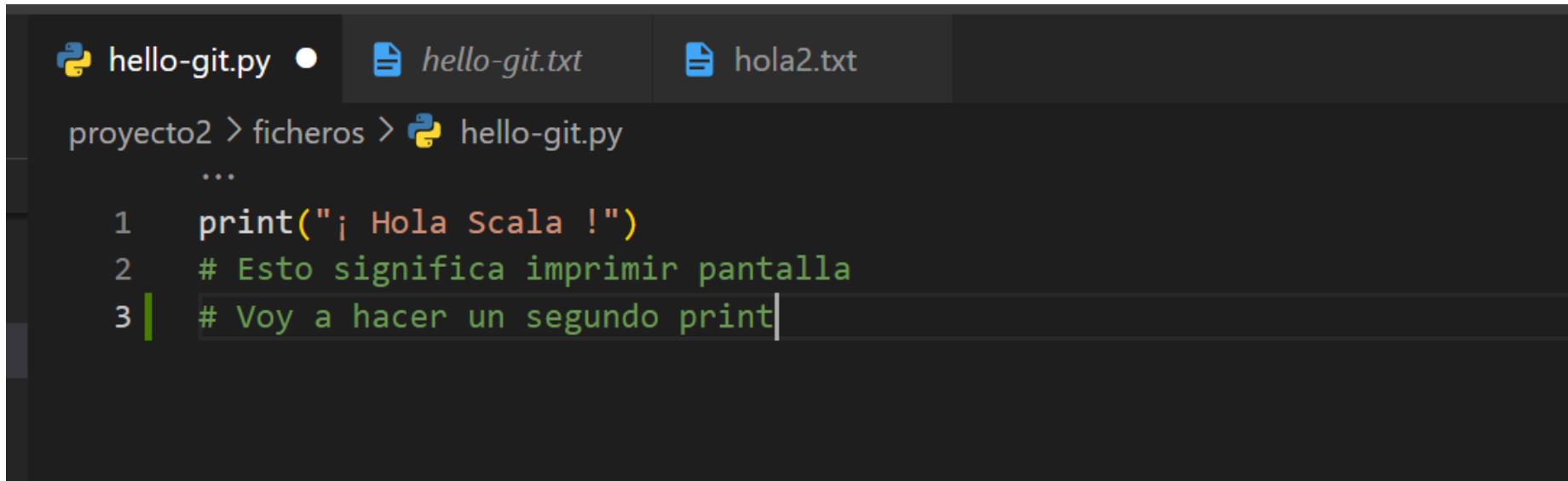


Zonas de git después del stash



Por ejemplo, supongamos que estamos editando el fichero hello-git.py desde la rama desarrollo:

Añadimos una tercera línea que diga: **# Voy a hacer un segundo print**



```
hello-git.py • hello-git.txt hola2.txt
proyecto2 > ficheros > hello-git.py
...
1 print("¡ Hola Scala !")
2 # Esto significa imprimir pantalla
3 # Voy a hacer un segundo print
```

Guardamos y hacemos un add.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git add hello-git.py
```

Ahora editamos el fichero hello-git.txt sin hacer un add.

Añadimos una segunda línea escribiendo: **Esto es un test de documentación**

```
hello-git.py M          hello-git.txt ●          hola2.txt
proyecto2 > ficheros > hello-git.txt
You, 3 days ago | 1 author (You)
1 Primera linea: hola git
2 Esto es un test de documentación You, 3
```

Guardamos los cambios sin hacer add. Supongamos que nuestro jefe nos pide hacer un cambio urgente sobre la rama main , y tenemos que salirnos de la rama desarrollo:

git te informa que has modificado dos archivos, y se puede correr el riesgo de perder los cambios.

Pedimos status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py

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:   hello-git.txt
```

Necesitamos tener nuestro directorio de trabajo y el área de preparación limpios para poder trabajar en la rama main. Sin embargo, aún no estoy seguro de que los cambios realizados en la rama desarrollo vayan a ser confirmados, de allí la ventaja de utilizar git stash.

Escribimos **git stash** para tener “una copia ” de los cambios que yo estaba haciendo en un “almacén temporal” un stash para así poder liberar el working directory y el staging área.

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash
Saved working directory and index state WIP on desarrollo: 683fe82 Conflicto resuelto
```

Una vez que tengo en un stash los cambios que estaba haciendo en la rama de desarrollo ya puedo moverme a la rama main sin problema, escribimos **git switch main**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git switch main
Switched to branch 'main'
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$
```

Puedo guardar todos los stash que necesite. Para poder ver todos los stash que están guardados se escribe: git stash list

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$ git stash list
stash@{0}: WIP on desarrollo: 683fe82 Conflicto resuelto
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (main)
$
```

Supongamos que hemos resuelto los problemas que tocaba resolver en la rama main. Ahora nos podemos mover a la rama de desarrollo y pedimos status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ switch main
bash: switch: command not found
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

Ahora podemos continuar con lo que se estaba haciendo en la rama de desarrollo: escribimos **git stash pop**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash pop
On branch desarrollo
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:   hello-git.py
    modified:   hello-git.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (b30cc49c62aff54e8de38b708cc811cf6c0c4bad)
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
```

git stash pop

Este comando eliminará los cambios del stash y los aplicará nuevamente en el código en el que estás trabajando.

Mientras que **git stash apply** aplica el último stash (por defecto, el stash más reciente) a tu rama actual y a su vez mantiene el stash en la lista después de aplicarlo.

También está **git stash apply <ID>** que aplica el stash especificando su ID.

Por ejemplo, vamos a enviar lo que tenemos en el working directory al staging area: ‘git add’ :

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git add .

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py
    modified:   hello-git.txt
```

Ahora creamos un stash: **git stash**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash
Saved working directory and index state WIP on desarrollo: 683fe82 Conflicto resuelto
```

git status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

Escribimos ahora: **git stash list**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
stash@{0}: WIP on desarrollo: 683fe82 Conflicto resuelto
```

Escribimos ahora git stash apply

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash apply
On branch desarrollo
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:   hello-git.py
    modified:   hello-git.txt

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

git stash list

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
stash@{0}: WIP on desarrollo: 683fe82 Conflicto resuelto
```

Vamos a borrar el stash que tenemos guardado en la lista:
git stash drop

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash drop
Dropped refs/stash@{0} (5775c8f7fc10f41cf5867ff2e98603a42561794d)

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
```

Vamos a probar una variante de git stash apply; para empezar pedimos status:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
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:   hello-git.py
    modified:   hello-git.txt

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

git add .

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git add .
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py
    modified:   hello-git.txt
```

Nuevamente vamos a crear un stash:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash
Saved working directory and index state WIP on desarrollo: 683fe82 Conflicto resuelto
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

Antes de aplicar git stash nuestros cambios estaban en el ‘Staging Area’, si aplicamos git stash apply nos devuelve los cambios al working directory ; pero ¿y si yo quiero que devuelva los cambios tal y como se dejaron? es decir, que git me devuelva los cambios que tenia en el ‘staging área (index)’

git stash apply --index

Devuelve los cambios que estaban en el working directory al working directory y los cambios que estaban en el staging area al staging area, además deja el stash en la lista sin borrar.

Vamos a probarlo:

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash apply --index
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py
    modified:   hello-git.txt
```

Ha devuelto los cambios a su zona original: ‘staging area’

Vamos a probar otro ejemplo. Vamos a sacar del ‘staging area’ el archivo: **hello-git.txt**

Escribimos: **git restore --staged hello-git.txt**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git restore --staged hello-git.txt
```

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
Changes to be committed:
```

```
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py
```

```
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:   hello-git.txt
```

Verificamos si tenemos nuestro stash vacío: **git stash list**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
stash@{0}: WIP on desarrollo: 683fe82 Conflicto resuelto
```

Borramos lista: **git stash drop**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash drop
Dropped refs/stash@{0} (9f1550faf7c7d49a69024b585e6c96597fb3a516)
```

Ahora sí guardamos en el stash lo que tenemos en el ‘working directory’ y en el ‘staging area’: **git stash**

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash
Saved working directory and index state WIP on desarrollo: 683fe82 Conflicto resuelto
```

Verificamos en lista y pedimos status:

git stash list

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list
stash@{0}: WIP on desarrollo: 683fe82 Conflicto resuelto
```

git status

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarrollo
nothing to commit, working tree clean
```

Ahora escribimos: `git stash apply --index`

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash apply --index
On branch desarrollo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello-git.py

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:   hello-git.txt
```

Nota: `git stash pop` no tiene esta opción, es decir, te devuelve los cambios al ‘working directory’

Añadimos y confirmamos cambios (add y commit)

```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git add .

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git commit -m "Cambios en ficheros: hello-git.txt y .py"
[desarrollo 28446d9] Cambios en ficheros: hello-git.txt y .py
 2 files changed, 4 insertions(+), 2 deletions(-)
```

git status,
git stash drop y
git stash list

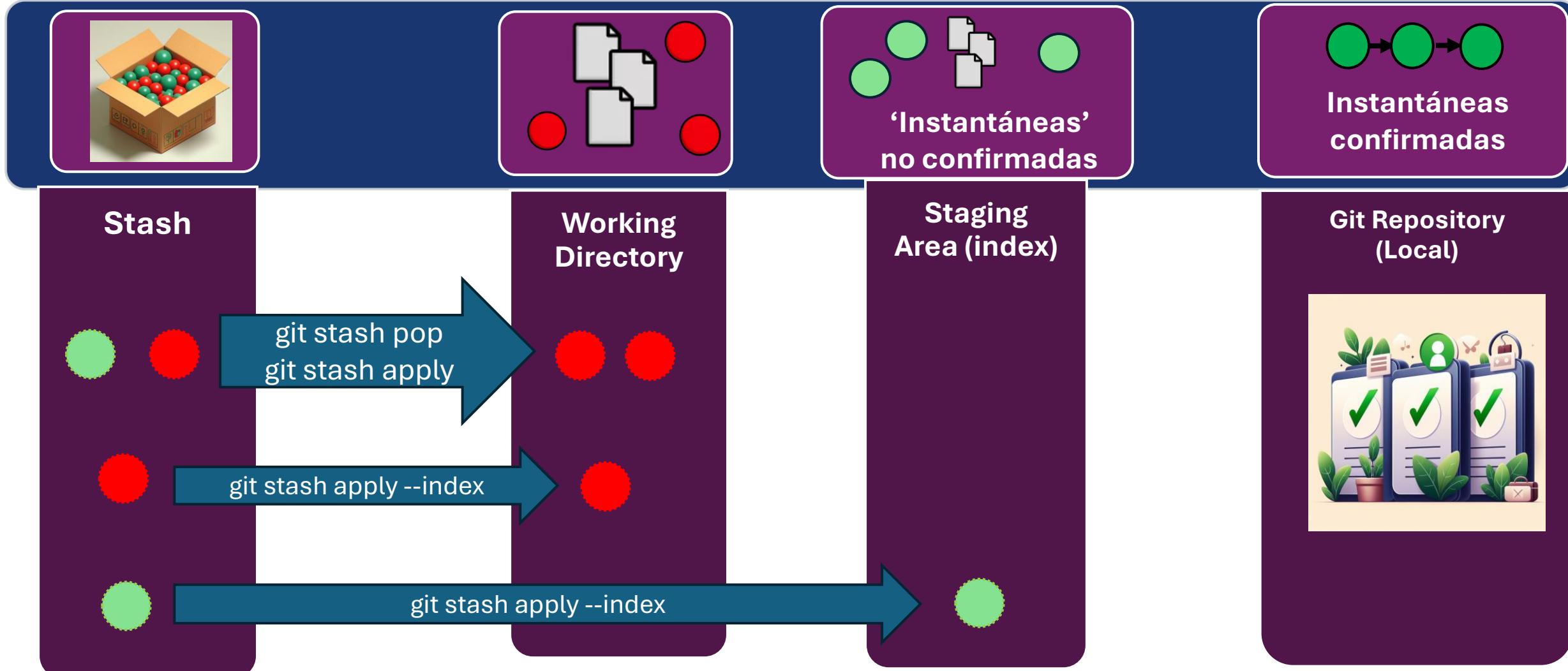
```
juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git status
On branch desarollo
nothing to commit, working tree clean

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash drop
Dropped refs/stash@{0} (72d9a6f2f90ac4c0558a2b5f46d0a583a5e57ab5)

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$ git stash list

juanj@balrodjj MSYS ~/OneDrive/Documentos/modulo3-git/proyecto2/ficheros (desarrollo)
$
```

Comparación de git stash pop y git stash apply



Practica_3



3.25 GitHub.

¿Qué es GitHub?

GitHub es el mayor proveedor de alojamiento de repositorios Git, y es el punto de encuentro para que millones de desarrolladores colaboren en el desarrollo de sus proyectos.

Un gran porcentaje de los repositorios Git se almacenan en GitHub, y muchos proyectos de código abierto lo utilizan para hospedar su Git, realizar su seguimiento de fallos, hacer revisiones de código y otras cosas.

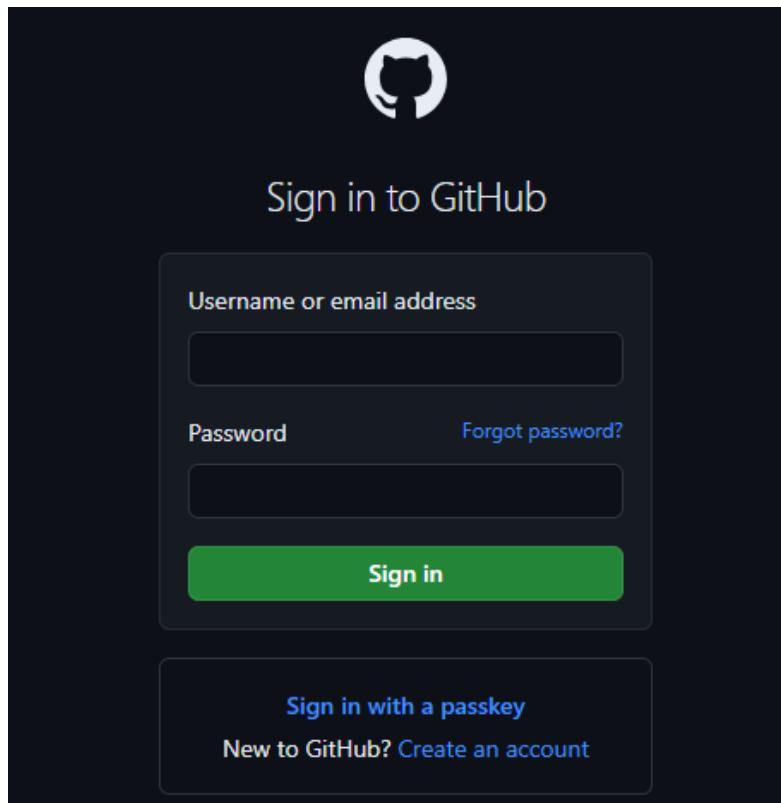


3.25.1. Creación y configuración basica de la cuenta.

Creación de la cuenta

Lo primero que se necesita es una cuenta de usuario gratuita. Simplemente visita <https://github.com>, elige un nombre de usuario que no esté ya en uso, proporciona un correo y una contraseña, y pulsa el botón verde grande “**Sign up for GitHub**”.

En nuestro caso ya hemos creado la cuenta al comienzo del curso, por lo tanto, vamos a [iniciar sesión](#).





Dashboard

Top Repositories

New

Find a repository...

robalza/robalza.github.io

robalza/proyecto-ds-x

robalza/mslearn-demo

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

Home

Send feedback

Filter 8

Updates to your homepage feed

We've combined the power of the Following feed with the For you feed so there's one place to discover content on GitHub. There's improved filtering so you can customize your feed exactly how you like it, and a shiny new visual design. 🎉

[Learn more](#)

Start writing code

Start a new repository for robalza

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

name your new repository...

Public

Anyone on the internet can see this repository

Private

You choose who can see and commit to this repository

[Create a new repository](#)

Introduce yourself with a profile README

Share information about yourself by creating a profile README, which appears at the top of your profile page.

robalza / README.md

Create

- 1 - 🌟 Hi, I'm @robalza
- 2 - 🌐 I'm interested in ...
- 3 - 🌱 I'm currently learning ...
- 4 - ❤️ I'm looking to collaborate on ...
- 5 - 📩 How to reach me ...
- 6 - 😊 Pronouns: ...
- 7 - ⚡ Fun fact: ...
- 8

UNIVERSE24

The world's fair of software is here.

Get 35% off your tickets to GitHub Universe, only until July 8.

[Get tickets](#)

Learning Pathways

AI-powered development with GitHub Copilot

Gain expertise and insights from top organizations in these GitHub-guided learning pathways.

- AI-powered development with GitHub Copilot
- CI/CD with GitHub Actions
- Application Security with GitHub Advanced Security
- Administration and Governance with GitHub Enterprise

[GitHub Learning Pathways](#)

Latest changes

- 4 days ago Push rules public beta
- 4 days ago CodeQL for Visual Studio Code documentation is now on docs.github.com
- 5 days ago Updates to GitHub Importer and the deprecation of the Source Import REST API...
- 5 days ago Configure organization-level CodeQL model packs for GitHub code scanning

Use tools of the trade

Write code in your web browser

Use the [github.dev](#) web-based editor from your repository or pull request to create and commit changes.

Get AI-based coding suggestions



Try GitHub Copilot free for 30 days, which suggests entire functions in real time, right from your editor.

Repositorios y creación

The screenshot shows the GitHub Dashboard. At the top left is a menu icon (three horizontal lines) and the GitHub logo. To the right, the word "Dashboard" is displayed. Below this, a section titled "Top Repositories" lists three repositories: "rodbalza/rodbalza.github.io", "rodbalza/proyecto-ds-x", and "rodbalza/mslearn-demo". Each repository entry includes a small circular profile picture and the repository name. To the right of the repository list is a green button with a white icon of a clipboard and the word "New". Below the repository list is a search bar with the placeholder text "Find a repository...". At the bottom of the dashboard, there is a section titled "Recent activity".

The screenshot shows the GitHub interface for creating a new repository. At the top, there is a button labeled "Start writing code". Below it, a section titled "Start a new repository for rodbalza" is shown. A descriptive text explains that a repository contains all of your project's files, revision history, and collaborator discussion. A form field is provided for entering the "Repository name *". Below the field is a placeholder text "name your new repository...". There are two radio button options: "Public" and "Private". The "Private" option is selected, indicated by a blue outline around the radio button and the text "You choose who can see and commit to this repository". At the bottom of the form is a green button labeled "Create a new repository".

Para crear una breve presentación sobre ti en un archivo en formato markdown (.md)

Introduce yourself with a profile README

Share information about yourself by creating a profile README, which appears at the top of your profile page.

rodbalza / README.md

Create

- 1 - 🙋 Hi, I'm @rodbalza
- 2 - 💬 I'm interested in ...
- 3 - 🌱 I'm currently learning ...
- 4 - ❤️ I'm looking to collaborate on ...
- 5 - 📩 How to reach me ...
- 6 - 😊 Pronouns: ...
- 7 - ⚡ Fun fact: ...
- 8

Algunos botones del home de GitHub

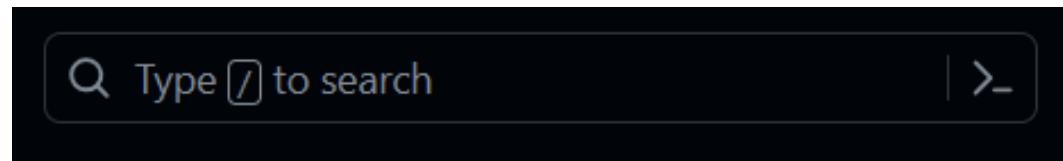
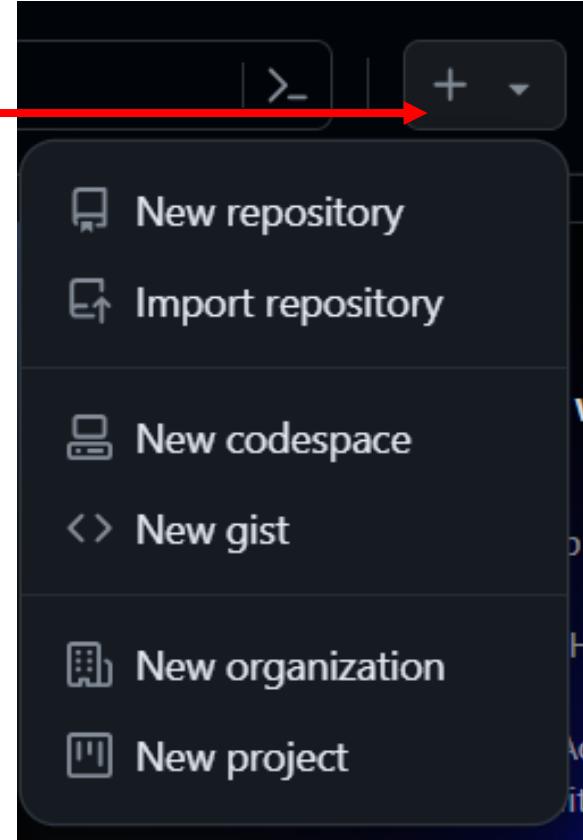


Botón
desplegable
Menú

Captura de pantalla del menú desplegable de GitHub. Se muestra una lista de enlaces: Home, Issues, Pull requests, Projects, Discussions, Codespaces, Explore y Marketplace. Abajo se muestran los repositorios: rodbalza/rodbalza.github.io, rodbalza/proyecto-ds-x y rodbalza/mslearn-demo. En la parte inferior hay un pie de página con links a About, Blog, Terms, Privacy, Security, Status, una opción para no compartir información personal y Manage Cookies.



Botón de
Create
new...

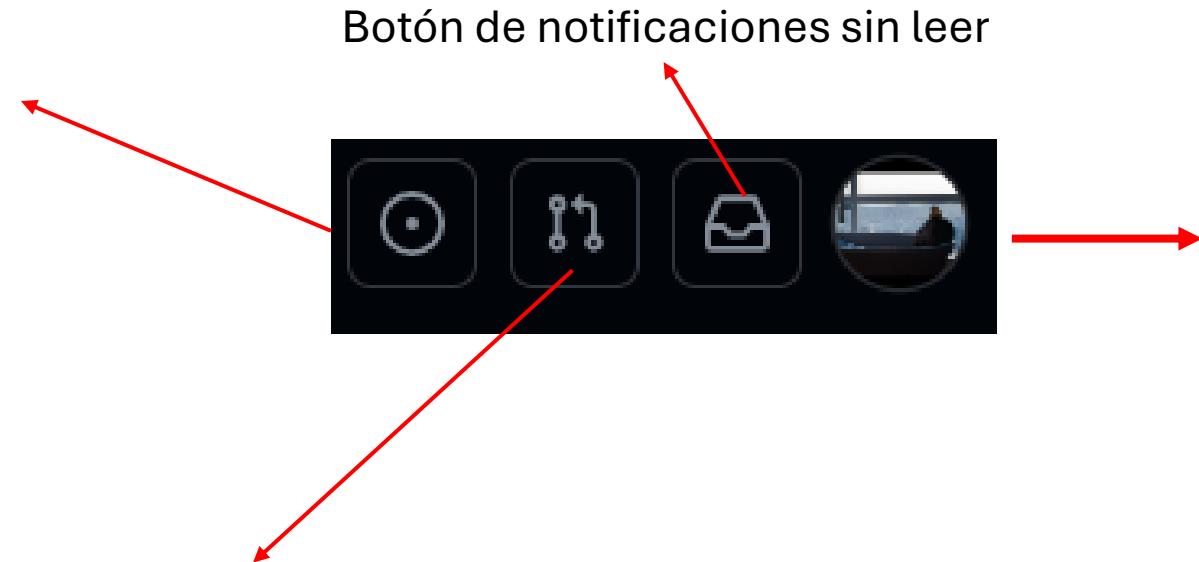


Barra de
búsqueda

Algunos botones del home de GitHub

Boton de Issues

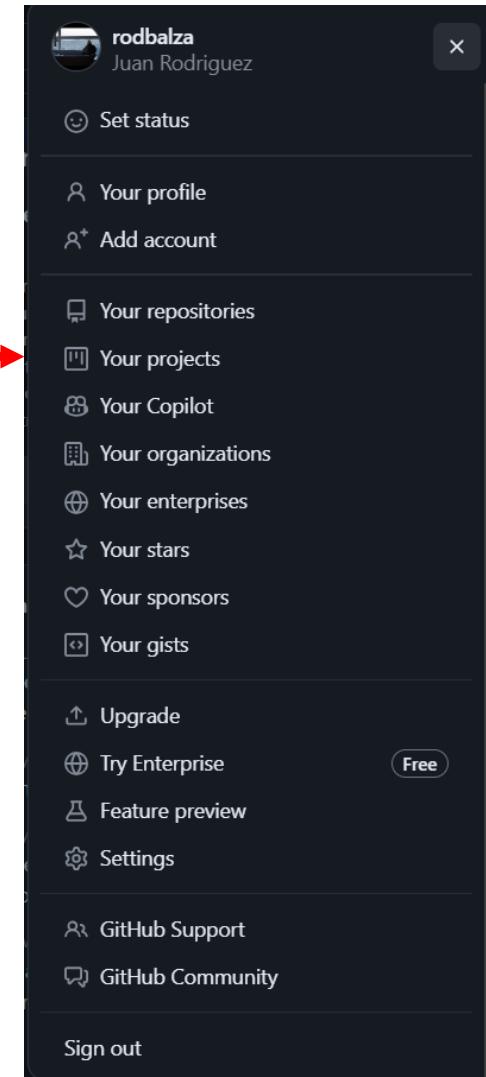
El botón “Issue” en GitHub se utiliza para rastrear tareas, problemas, mejoras y discusiones relacionadas con un repositorio.



Botón de ‘Pull requests’.

Un pull request es una petición que el propietario de un fork de un repositorio hace al propietario del repositorio original para que este último incorpore los commits que están en el fork.

Botón de notificaciones sin leer



Acceso SSH

Opción 1: Una vez que has creado tu cuenta de GitHub ya puedes acceder a los repositorios públicos mediante <https://>, identificándote con el usuario y la contraseña que acabas de elegir.

Opción 2: Mediante SSH si quieres utilizar una llave pública. A continuación, vamos a generar una clave pública (SSH key)

Pasos para generar la clave pública escribimos en Git Bash:

Para generar la clave en Windows debes estar posicionado en el directorio **.ssh/**

Para buscar la carpeta en git bash escribir:

```
juanj@balrodjj MSYS ~
$ echo $HOME/.ssh
/c/Users/juanj/.ssh
```

Luego, moverse al directorio que se muestra en la salida usando **cd** y luego confirmar con **pwd**.

```
juanj@balrodjj MSYS ~
$ pwd
/c/Users/juanj
```

Pasos para generar la clave pública escribimos en Git Bash:

Escribimos **ls -a** para verificar si aparece el directorio oculto **.ssh** si la carpeta no aparece debe crearla: **mkdir .ssh**

Nos movemos a la carpeta **.ssh**, escribimos: **cd .ssh**

```
juanj@balrodjj MSYS ~  
$ mkdir .ssh
```

```
juanj@balrodjj MSYS ~  
$ cd .ssh/
```

```
juanj@balrodjj MSYS ~/ssh  
$ ls
```

```
juanj@balrodjj MSYS ~/ssh  
$ █
```

Pasos para generar la clave pública escribimos en Git Bash:

Escribimos: ssh-keygen -t ed25519 -C "tu_email@ejemplo.com"

```
juanj@balrodji MSYS ~/.ssh
$ ssh-keygen -t ed25519 -C juanjose.rodriguez@tajamar365.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/juanj/.ssh/id_ed25519): clavepublica
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in clavepublica
Your public key has been saved in clavepublica.pub
The key fingerprint is:
SHA256: bv9n4fG3MxCrw/i7eqfbkHHMEBOnnJylhUd8o5XPEY8 juanjose.rodriguez@tajamar365.com
```

Si no quieres crear fichero cuando pregunta: **Enter file in which to save the key**

Entonces presiona ENTER 3 veces para que salte las preguntas de nombre archivo y contraseña

Escribe **ls** para verificar que los archivos han sido creados

```
juanj@balrodjj MSYS ~/.ssh
$ ls
clavepublica clavepublica.pub
```

```
juanj@balrodjj MSYS ~/.ssh
$ cat clavepublica.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILFgdzGOXAFqZ6uvfQ5qAGdEn0cYBkR09kI+vtuU0RQD juanjose.rodriguez@tajamar365.com
```

Agregar tu clave SSH al ssh-agent

- Abre una terminal de PowerShell y activa permisos de administrador escribiendo:

Start-Process powershell -Verb RunAs

- Escribe en la terminal de PowerShell :

Get-Service -Name ssh-agent | Set-Service -StartupType Manual

Start-Service ssh-agent

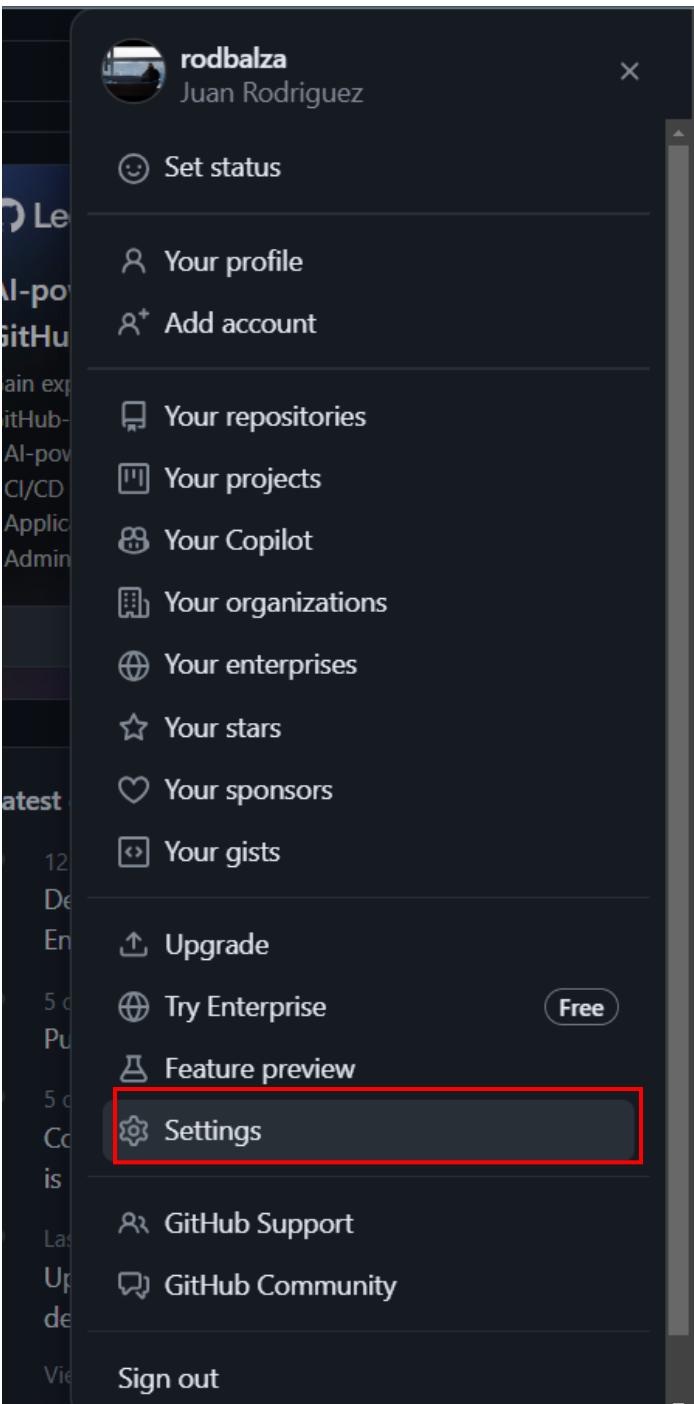
Cierra la terminal de PowerShell con permisos de administrador.

En una ventana de terminal sin permisos elevados, agregue la clave privada SSH al agente ssh. Si has creado tu clave con otro nombre o si vas a agregar una clave existente que tiene otro nombre, reemplaza id_ed25519 en el comando por el nombre de tu archivo de clave privada:

ssh-add c:/Users/YOU/.ssh/id_ed25519

```
● PS C:\Users\juanj\.ssh> ssh-add C:\Users\juanj\.ssh\clavepublica
Enter passphrase for C:\Users\juanj\.ssh\clavepublica:
Identity added: C:\Users\juanj\.ssh\clavepublica (juanjose.rodriguez@tajamar365.com)
```

Agrega la clave pública de SSH a tu cuenta en GitHub



Ir hasta SSH and GPG keys

Juan Rodriguez (rodbalza)
Your personal account [Go to your personal profile](#)

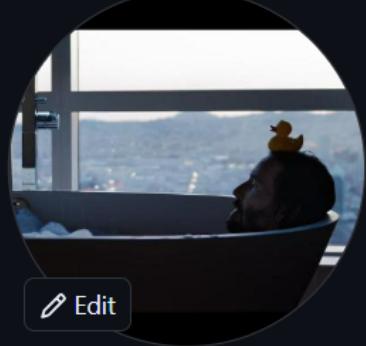
[Public profile](#) [Account](#) [Appearance](#) [Accessibility](#) [Notifications](#)

[Billing and plans](#) [Emails](#) [Password and authentication](#) [Sessions](#) [SSH and GPG keys](#) [Organizations](#) [Enterprises](#) [Moderation](#)

[Code, planning, and automation](#)

Public profile

Name Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Profile picture  [Edit](#)

Public email [Select a verified email to display](#) You have set your email address to private. To toggle email privacy, go to [email settings](#) and uncheck "Keep my email address private."

Bio You can @mention other users and organizations to link to them.

Pronouns [Don't specify](#)

URL

Click en New SSH key

SSH keys

New SSH key

There are no SSH keys associated with your account.

Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

New GPG key

There are no GPG keys associated with your account.

Learn how to [generate a GPG key](#) and add it to your account.

Vigilant mode

Flag unsigned commits as unverified

This will include any commit attributed to your account but not signed with your GPG or S/MIME key.

Note that this will include your existing unsigned commits.

[Learn about vigilant mode.](#)

En title, añade un título de referencia, por ejemplo: **curso-git**

Add new SSH Key

Title

Key type

Authentication Key

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

En **Key type**, debe estar seleccionado: **Authentication Key**

Ve al terminal de **git bash** , te mueves al directorio **.ssh/** y luego abres con un **cat** el archivo **clavepublica.pub** ; copias el texto que allí aparece.

```
juanj@balrodjj MINGW64 ~/ssh
$ cat clavepublica.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILFgdzGOXAFqZ6uvfQ5qAGdEnOcYBkR09kI+vtuU0RQD juanjose.rodriguez@tajamar365.com
```

Una vez copiado el texto, te vas a GitHub y pegas el texto en el cuadro que dice **key**:

Una vez pegada la clave pública hacer click en Add SSH key

Add new SSH Key

Title

Key type

Key

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIFgdzGOXAFqZ6uvfQ5qAGdEnOcYBkR09kl+vtuU0RQD juanjose.rodriguez@tajamar365.com
```

Add SSH key

La salida esperada sería:

SSH keys

[New SSH key](#)

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

Authentication keys

**curso-git**

SHA256: bv9n4fG3MxCrw/i7eqfbkHHMEB0nnJylhUd8o5XPEY8

SSH

Added on Apr 23, 2024

Never used — Read/write

[Delete](#)

Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).

Probar tu conexión SSH (documentación de GitHub)

1 Abra Git Bash.

2 Escriba lo siguiente:

```
$ ssh -T git@github.com  
# Attempts to ssh to GitHub
```

Puedes ver una advertencia como la siguiente:

```
> The authenticity of host 'github.com (IP ADDRESS)' can't be established.  
> ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvC0qU.  
> Are you sure you want to continue connecting (yes/no)?
```

3 Compruebe que la huella digital del mensaje que ve coincide con [la huella digital de clave pública de GitHub](#). En caso afirmativo, escriba `yes` :

```
> Hi USERNAME! You've successfully authenticated, but GitHub does not  
> provide shell access.
```

Nota: El comando remoto debe salir con el código 1.

4 Comprueba que el mensaje resultante contenga tu nombre de usuario. Si recibes un mensaje de "permiso denegado", consulta "[Error: Permiso denegado \(publickey\)](#)".

Probar tu conexión SSH (documentación de GitHub)

1 Abra Git Bash.

2 Escriba lo siguiente:

```
$ ssh -T git@github.com  
# Attempts to ssh to GitHub
```

Puedes ver una advertencia como la siguiente:

```
> The authenticity of host 'github.com (IP ADDRESS)' can't be established.  
> ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvC0qU.  
> Are you sure you want to continue connecting (yes/no)?
```

3 Compruebe que la huella digital del mensaje que ve coincide con [la huella digital de clave pública de GitHub](#). En caso afirmativo, escriba `yes`:

```
> Hi USERNAME! You've successfully authenticated, but GitHub does not  
> provide shell access.
```

Nota: El comando remoto debe salir con el código 1.

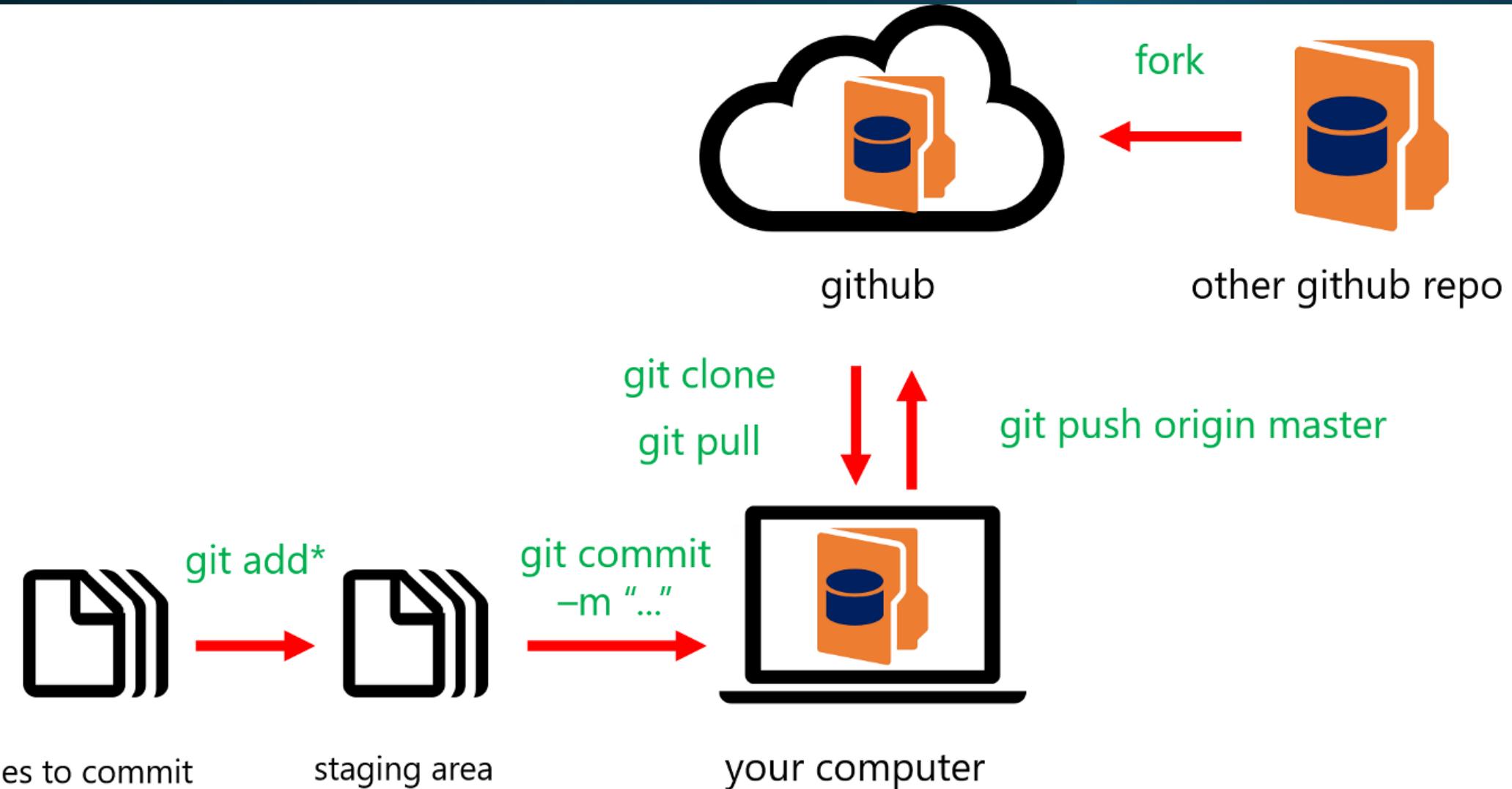
4 Comprueba que el mensaje resultante contenga tu nombre de usuario. Si recibes un mensaje de "permiso denegado", consulta "[Error: Permiso denegado \(publickey\)](#)".

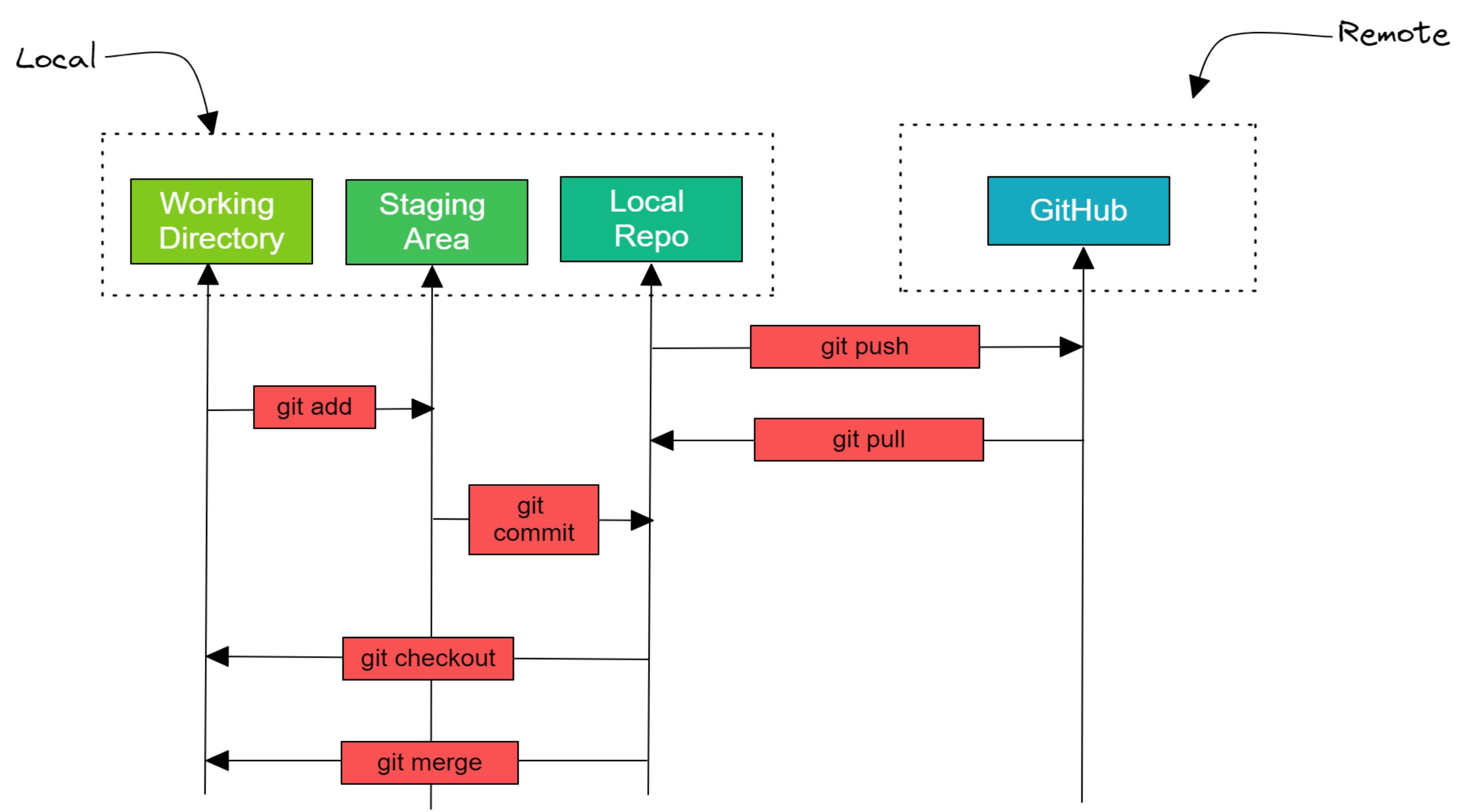
Salida esperada. Si da errores hay que consultar documentación

```
juanj@balrodjj MINGW64 ~/.ssh
$ ssh -T git@github.com
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
```

```
juanj@balrodjj MINGW64 ~/.ssh
$ ssh -T git@github.com
Hi rodbalza! You've successfully authenticated, but GitHub does not provide shell access.
```

Desde el directorio de trabajo hasta el repositorio remoto







3.25.2 Creando un repositorio en GitHub para enviar mi trabajo en local

Vamos a crear un nuevo repositorio en GitHub:

<> Start writing code

Start a new repository for rodbalza

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

Curso-Git-Github is available.

Public
Anyone on the internet can see this repository

Private
You choose who can see and commit to this repository

Create a new repository

Una vez creado el repositorio nos lleva a un ‘Quick Setup’:

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/rodbalza/Curso-Git-Github.git> [Copy](#)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Curso-Git-Github" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/rodbalza/Curso-Git-Github.git  
git push -u origin main
```

Copiar este comando 

...or push an existing repository from the command line

```
git remote add origin https://github.com/rodbalza/Curso-Git-Github.git  
git branch -M main  
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#) [Copy](#)



3.25.4 Git Remote

Vamos a Git Bash y escribimos (o pegamos) el comando:
`git remote add origin https://github.com/rodbalza/Curso-Git-Github.git`

Debe ser el comando que aparece en tu GitHub

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git remote add origin https://github.com/rodbalza/Curso-Git-Github.git
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ |
```

El comando: `git remote add origin https://github.com/rodbalza/Curso-Git-Github.git` se utiliza para agregar un repositorio remoto a tu repositorio local de Git. Esto te permite conectar tu proyecto local a un repositorio remoto en GitHub, lo que te facilita subir y bajar cambios entre tu repositorio local y el remoto.

Explicación desglosada

- **git remote**: Este comando le indica a Git que deseas administrar repositorios remotos asociados con tu proyecto local.
- **add**: Este subcomando indica que deseas agregar un nuevo repositorio remoto.
- **origin**: Este es un nombre común que se le da al repositorio remoto principal al vincularlo con tu repositorio local. Puedes elegir un nombre diferente si lo prefieres.
- **<https://github.com/rodbalza/Curso-Git-Github.git>**: Esta es la URL del repositorio remoto que deseas agregar. Especifica la ubicación del repositorio remoto en GitHub.

Una vez añadido nuestro repositorio remoto a nuestra zona de git , el siguiente paso es enviar todos nuestros cambios en local a remoto.



3.25.5 Git Push

Para enviar todo nuestro proyecto a remoto escribimos desde la terminal de Git Bash :

git push -u origin main

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git push -u origin main
Enumerating objects: 60, done.
Counting objects: 100% (60/60), done.
Delta compression using up to 8 threads
Compressing objects: 100% (50/50), done.
Writing objects: 100% (60/60), 5.81 KiB | 396.00 KiB/s, done.
Total 60 (delta 13), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (13/13), done.
To https://github.com/rodbalza/Curso-Git-Github.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

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

El comando **git push -u origin main** se utiliza para subir tu rama local main a un repositorio remoto llamado origin y configurarla como la rama (upstream) para rastrear cambios futuros. Esto significa que Git recordará automáticamente este repositorio remoto y rama como el destino predeterminado para subir tus cambios y facilitará las futuras actualizaciones.

Explicación desglosada:

- **git push**: Este comando le indica a Git que deseas subir tus commits locales a un repositorio remoto.
- **-u**: Esta bandera significa "set-upstream" y tiene dos efectos principales:
 - Establece la rama remota especificada (origin/main en este caso) como la rama “upstream” para tu rama local main. Esto significa que Git usará esta rama como referencia para rastrear cambios y facilitará las futuras actualizaciones.
 - Asocia la rama **local main** con la rama **remota origin/main**. Esto significa que cuando ejecutes **git push** sin especificar el nombre de la rama, Git automáticamente subirá tu rama local main a la rama remota origin/main.
- **origin**: Este es el nombre del repositorio remoto al que deseas subir tus cambios. Es el nombre predeterminado para el repositorio remoto que se creó cuando clonaste el repositorio o cuando lo creaste desde una ubicación remota.
- **main**: Este es el nombre de la rama local que deseas subir al repositorio remoto. Es el nombre de rama predeterminado para muchos repositorios de Git.

Vamos a GitHub y actualizamos la página:

The screenshot shows a GitHub repository page for 'Curso-Git-Github'. Key elements include:

- Branch Selection:** A dropdown menu for the 'main' branch is highlighted with a red box.
- Recent Activity:** A commit by 'rodbalza' titled 'Cambio de Git por Scala' is highlighted with a red box. It was made 5 days ago and includes 17 commits.
- Code Download:** A green 'Code' button with a dropdown arrow is highlighted with a red box. A red arrow points from this button to a modal window on the right.
- Modal Window (Local Clone):** Shows cloning options via HTTPS, SSH, or GitHub CLI. The URL <https://github.com/rodbalza/Curso-Git-Github.git> is displayed.
- README Section:** Includes a 'README' link, a 'Readme' icon, and a 'Add a README' button.
- Help Text:** Text encouraging users to add a README to help people understand their project.

Supongamos que otro usuario que no eres tú el cual tiene permisos para editar el repositorio decide crear el archivo README.md

The screenshot shows a GitHub repository interface. At the top, there are navigation links for 'main' (with a dropdown arrow), '1 Branch', '0 Tags', a search bar labeled 'Go to file', and buttons for 'Add file' and 'Code'. Below this, a list of files is shown:

File	Description	Last Commit
rodbalza Cambio de Git por Scala	2a875dd · 5 days ago	17 Commits
ficheros	Cambio de Git por Scala	5 days ago
.gitignore	Se ha creado .gitignore y añadido el directorio /sin-importan...	last week
README		

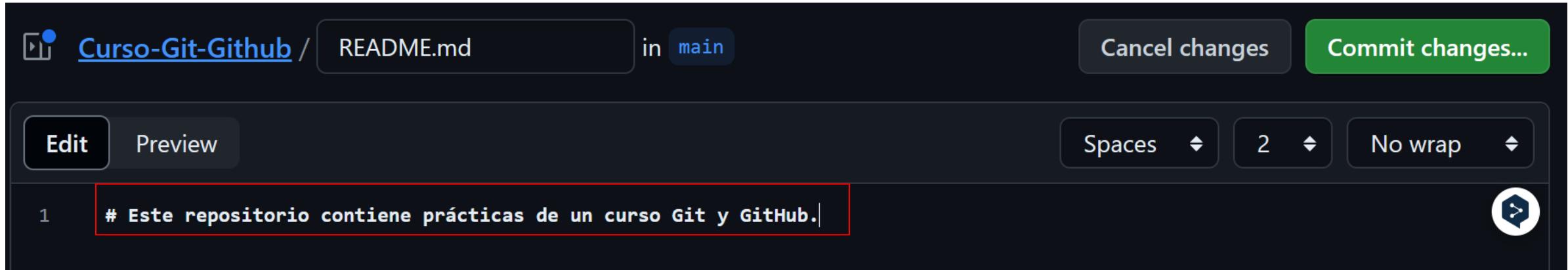
Below the file list, there is a large green button with the text 'Add a README' in white. This button is highlighted with a red rectangular border. In the center of the page, there is a small icon of an open book.

Add a README

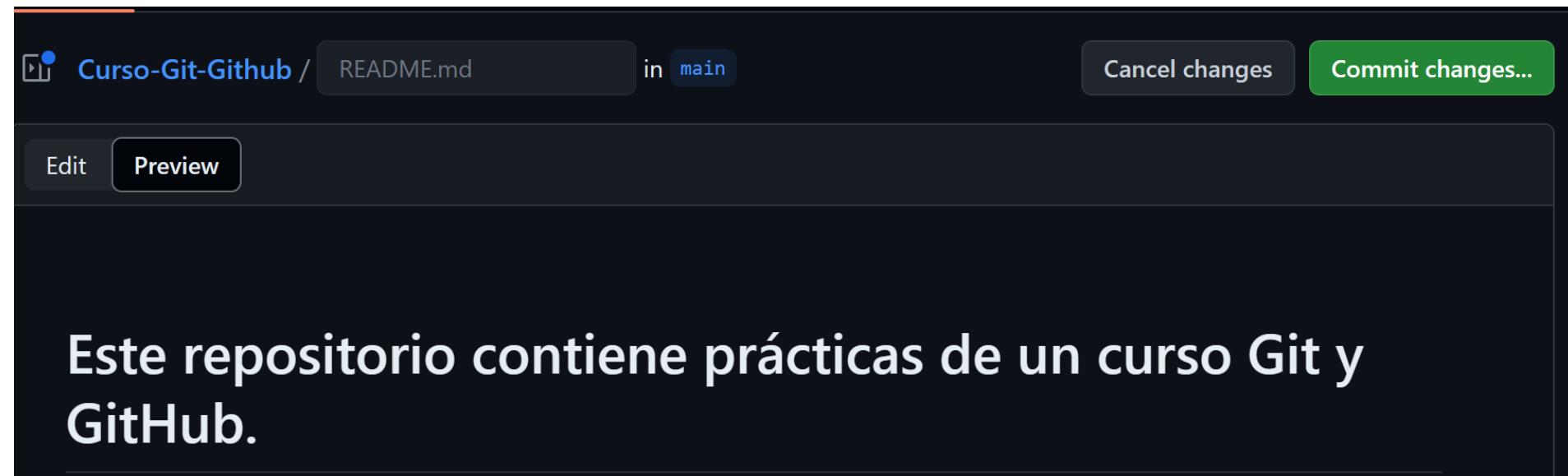
Help people interested in this repository understand your project by adding a README.

Add a README

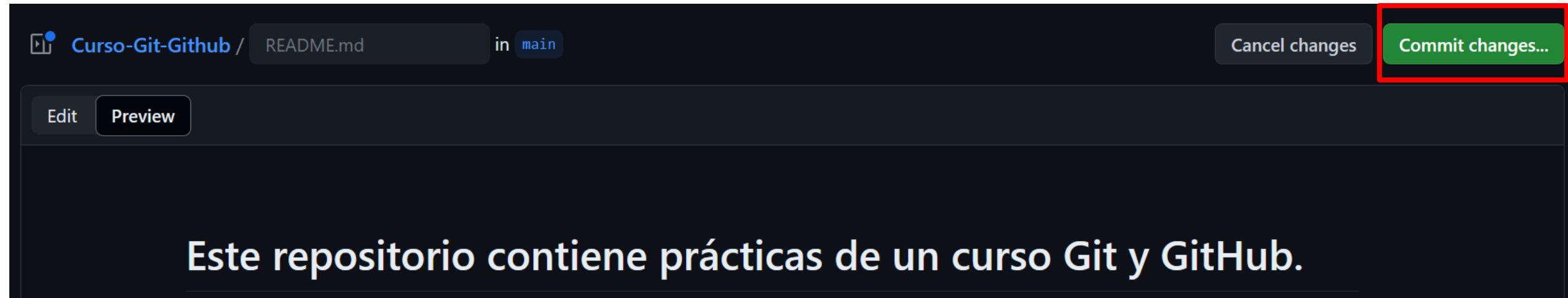
A continuación, se abre un editor de texto que admite los comandos de **markdown**.



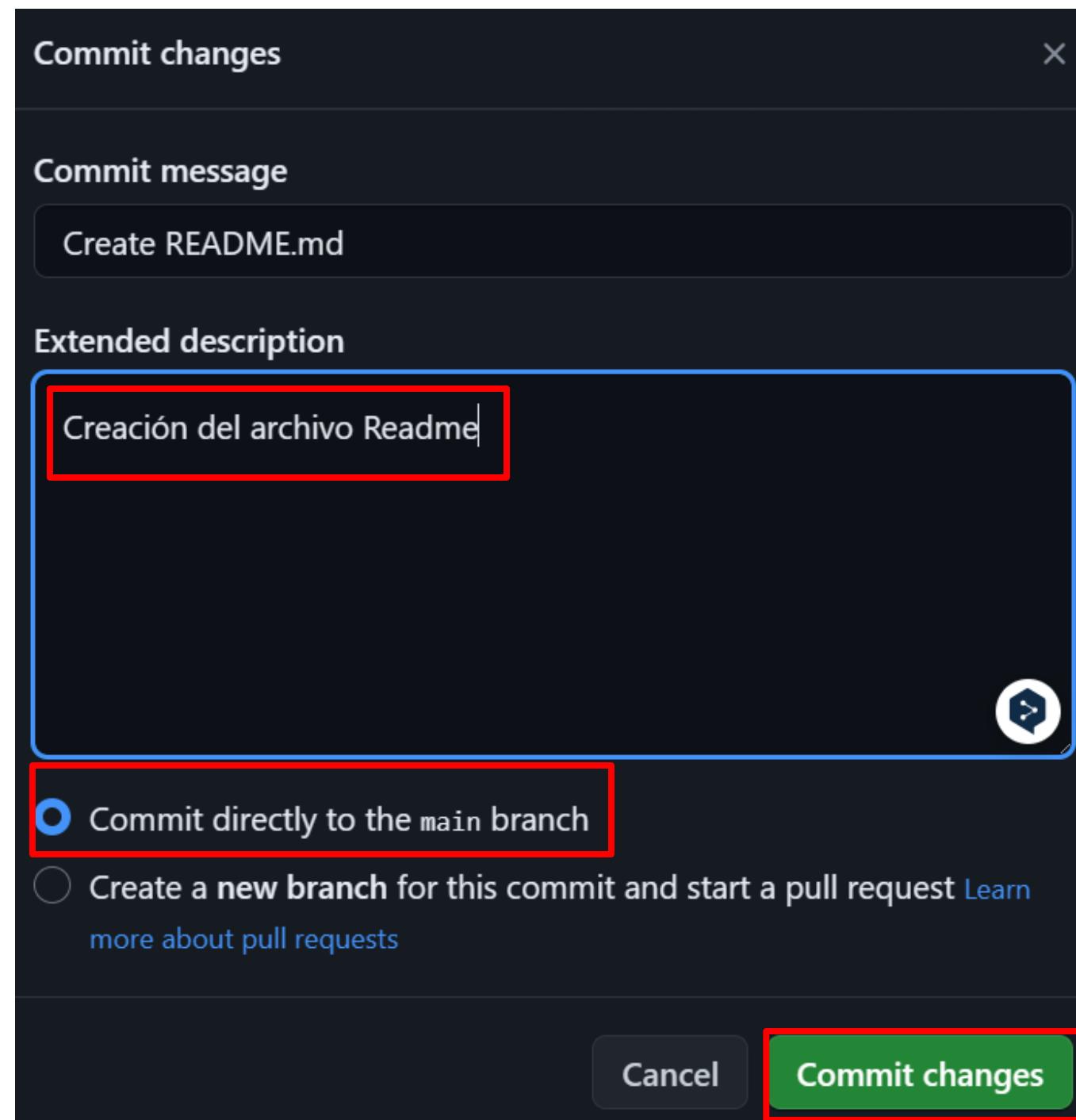
Luego, click en
Preview para
visualizar los cambios



Vamos a confirmar los cambios dando click en Commit Changes



Escribimos el
mensaje en la
descripción y
luego de
nuevo:
Commit
changes



Debería verse así:

The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with icons for repository, branch (main), repository name (Curso-Git-Github /), search (Go to file), a 't' icon, 'Add file', and more options. Below the navigation is a commit history card for a user named rodbalza. The card shows a commit to 'Create README.md'. The main content area displays a table of files with their last commit details. The table has columns for Name, Last commit message, and Last commit date. The files listed are 'ficheros', '.gitignore', and 'README.md'. The 'README.md' file is currently selected, indicated by a pencil icon in the top right corner. A large text overlay at the bottom of the page reads: 'Este repositorio contiene prácticas de un curso Git y GitHub.'

Name	Last commit message	Last commit date
ficheros	Cambio de Git por Scala	5 days ago
.gitignore	Se ha creado .gitignore y añadido el directorio /sin-importancia	last week
README.md	Create README.md	now

Este repositorio contiene prácticas de un curso Git y GitHub.

Hemos dicho que otro usuario ha realizado estos cambios en GitHub directamente. Ahora, tú no tienes esos cambios en el repositorio local

Supongamos que estás haciendo cambios en tu repositorio local y antes de empezar a editar pides un status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
nothing to commit, working tree clean
```

¿Cómo puedo ver los cambios que hay en el repositorio remoto sin modificar mi local?





5.5 Git Fetch

git fetch origin

El comando git fetch se utiliza para descargar información actualizada sobre las ramas y commits de un repositorio remoto sin fusionar esos cambios en tu rama local. En otras palabras, te permite obtener la última información del repositorio remoto, pero no realiza cambios en tu copia de trabajo local.

¿Cómo funciona git fetch origin?

Cuando ejecutas **git fetch origin**, Git se conecta al repositorio remoto origin (que generalmente es el repositorio remoto principal de tu proyecto) y descarga lo siguiente:

- Información de metadatos actualizada: Esto incluye detalles sobre las ramas, commits y autorías en el repositorio remoto origin.
- Nuevos objetos: Si hay nuevos commits o archivos en el repositorio remoto origin, Git los descarga en tu repositorio local.

Escribimos en nuestro Git Bash: git fetch origin

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git fetch origin
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ █
```

```
commit 046a93a40c0b6b375ae18aef46d8f5215bd7a6ee (origin/main)
Author: Juan Rodriguez <141120134+rodbalza@users.noreply.github.com>
Date:   Wed Apr 24 10:15:22 2024 +0200
```

Create README.md

Creación del archivo Readme

```
commit 2a875dd8c404001aa4650a9700f91a2cbecaf9e (HEAD -> main)
Author: balrod <juanjose.rodriguez@tajamar365.com>
Date:   Fri Apr 19 10:51:42 2024 +0200
```

Cambio de Git por Scala

Luego escribimos:
git log origin/main

También podemos escribir: git log origin/main --oneline

```
046a93a (origin/main) Create README.md
2a875dd (HEAD -> main) Cambio de Git por Scala
62ce302 Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida línea 1: A continuación...en el archivo hola3.txt
8edfe27 Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en línea 2, archivo hello-git.py
656a399 Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcb se han creado dos archivos vacios hola2.txt y hola3.txt
:
```

Otro comando utilizado también es: **git diff origin/main**



5.6 Git Pull

Una vez claro y entendido lo que tenemos en nuestro repositorio remoto, el siguiente paso lógico sería actualizar nuestro repositorio local antes de hacer cualquier cambio en mi directorio de trabajo:

Git pull:

Se utiliza para obtener las últimas actualizaciones de un repositorio remoto y fusionarlas con tu rama local. En otras palabras, combina la funcionalidad de **git fetch** y **git merge** en un solo comando. Esto lo convierte en una forma cómoda de mantenerte actualizado con los cambios en el repositorio remoto e integrarlos a tu trabajo local.

git pull paso a paso:

- Obtiene las últimas actualizaciones: Primero ejecuta git fetch para descargar la información más reciente sobre ramas, commits y otros cambios del repositorio remoto especificado.
- Fusiona los cambios: Una vez descargada la información más reciente, intenta fusionar los cambios de la rama remota (generalmente origin/main) con tu rama local (generalmente main). Si la fusión se realiza correctamente, tu rama local se actualizará para reflejar los últimos cambios del repositorio remoto.
- Resuelve conflictos (si los hubiera): Si hay algún conflicto durante la fusión, git pull se detendrá y te lo notificará. Entonces deberás resolver los conflictos manualmente y preparar los cambios antes de continuar.

¿Cuándo usar git pull?

Regularmente: Es una buena práctica usar git pull con frecuencia para mantenerte actualizado con los últimos cambios en el repositorio remoto, especialmente si trabajas en un proyecto colaborativo.

Antes de confirmar: Antes de crear una nueva confirmación (commit), se recomienda ejecutar git pull para asegurarte de que tu rama local esté sincronizada con el repositorio remoto. Esto ayudará a prevenir conflictos de fusión más adelante.

Después de obtener los cambios: Si ya has usado git fetch para descargar la información más reciente, puedes usar git pull para fusionar directamente esos cambios en tu rama local.

Vamos a la terminal de Git Bash y escribimos:
git pull origin main

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git pull origin main
From https://github.com/rodbalza/Curso-Git-Github
 * branch           main      -> FETCH_HEAD
Updating 2a875dd..046a93a
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

git status

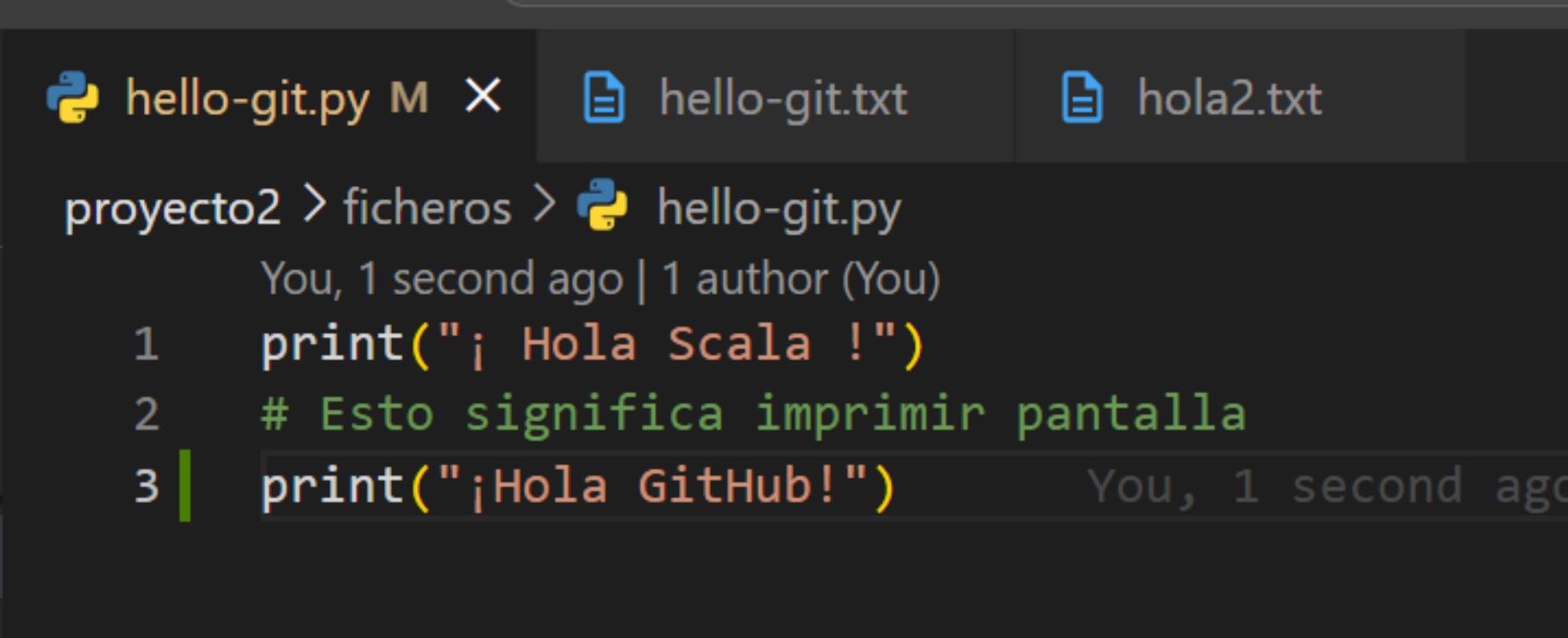
```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

git log --oneline

```
046a93a (HEAD -> main, origin/main) Create README.md
2a875dd Cambio de Git por Scala
62ce302 Merge branch 'desarrollo' para mejorar algunas características del proyecto
fbff1ea Añadida línea 1: A continuación...en el archivo hola3.txt
8edfe27 Añadida línea 4:Fase 1.2... en archivo /lib/hola2.txt
042c504 comentario en línea 2, archivo hello-git.py
656a399 Se ha borrado el archivo hola4.txt
c4fd413 Se ha creado lib2/hola4.txt
45255f2 creación de carpeta lib y se ha movido hola2.txt a /lib
ac10391 Añadido el nombre y el email del encargado de la fase 1.1 en las lineas 2 y 3 del archivo hola2.txt
6cb9316 Añadida Fase 1.1 en la primera linea del archivo hola2.txt
d9931b3 Revert "Ups...he cometido un error en este commit"
10f48c6 Añadida fases 1.1 y 1.2 en lineas 1 y 2
fda2f62 cambios en la primera linea del fichero hello-git.txt
12ebbcf se han creado dos archivos vacios hola2.txt y hola3.txt
```

Ya tengo mi repositorio local actualizado, por tanto, ya podemos hacer cambios. Vamos a editar el archivo hello-git.py desde vscode



```
hello-git.py M ×  hello-git.txt  hola2.txt  
proyecto2 > ficheros > hello-git.py  
You, 1 second ago | 1 author (You)  
1 print("¡ Hola Scala !")  
2 # Esto significa imprimir pantalla  
3 print("¡Hola GitHub!") You, 1 second ago
```

Añadimos la línea 3: **print("¡Hola GitHub!")**. Guardar, añadir y confirmar cambios.

Una vez realizado el commit pedimos status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git commit -a -m "cambios en hello-git.py: añadí la linea 3"
[main 8ebfa2d] cambios en hello-git.py: añadí la linea 3
 1 file changed, 2 insertions(+), 1 deletion(-)
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Enviamos los cambios al repositorio remoto escribiendo:
git push origin main

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 420 bytes | 420.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/rodbalza/Curso-Git-Github.git
 046a93a..8ebfa2d  main -> main
```

Si vamos a GitHub podemos ver los cambios que se hicieron desde local

The screenshot shows a GitHub repository named "Curso-Git-Github" which is public. The repository has one branch ("main") and no tags. The interface includes a search bar ("Go to file"), a code switcher ("Code"), and a commit history section.

The commit history is as follows:

- rodbalza** cambios en hello-git.py: añadí la linea 3 · 8ebfa2d · 9 minutes ago · 19 Commits
- ficheros** cambios en hello-git.py: añadí la linea 3 · 9 minutes ago
- .gitignore Se ha creado .gitignore y añadido el directorio /sin-imp... · last week
- README.md Create README.md · 1 hour ago
- README

Two specific commits are highlighted with red boxes: the first commit by "rodbalza" and the second commit by "ficheros".

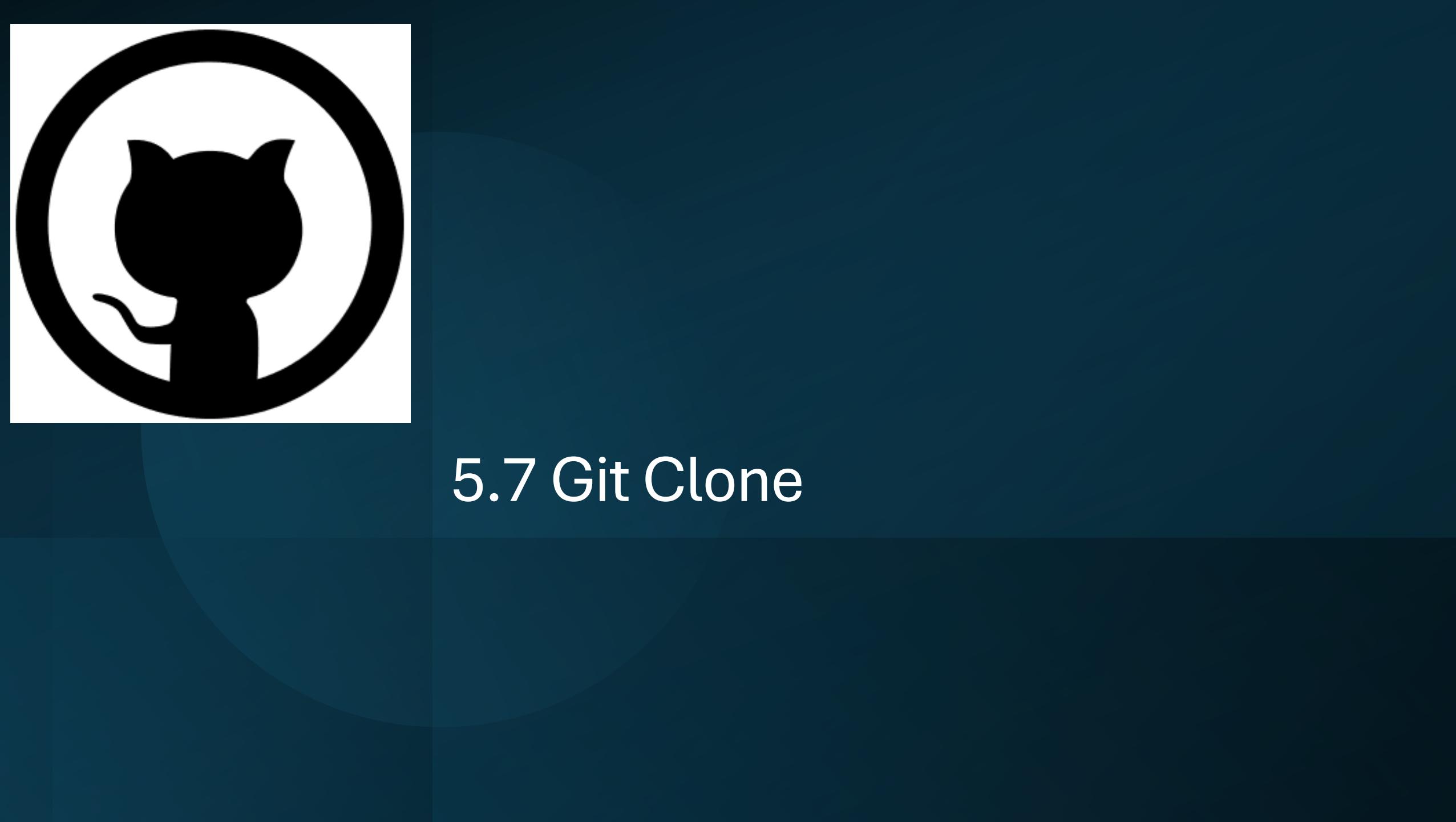
Si vemos el archivo `hello-git.py`, se podrá ver el cambio realizado:

A screenshot of a GitHub commit history page. The commits are listed in descending order of age. The first commit, highlighted with a red box, is for the 'ficheros' folder and includes a file named 'ficheros'. The commit message is 'cambios en hello-git.py: añadí la linea 3', it was made 15 minutes ago, and it has 19 commits. The second commit is for '.gitignore' and creates a file named '.gitignore'. The third commit is for 'README.md' and creates a file named 'README.md'. The commit history shows a total of 19 commits.

Commit	Message	Time
ficheros	cambios en hello-git.py: añadí la linea 3	15 minutes ago
.gitignore	Se ha creado .gitignore y añadido el directorio /sin-imp...	last week
README.md	Create README.md	1 hour ago

A screenshot of a GitHub file viewer for 'hello-git.py'. The file is located in the 'ficheros' folder. The code is displayed in a code editor interface. The code consists of three lines of Python code: 'print("¡ Hola Scala !")', '# Esto significa imprimir pantalla', and 'print("¡Hola GitHub!")'. The code area is highlighted with a yellow box. The file viewer also shows the commit history for this specific file, which is identical to the commit history shown in the previous screenshot.

```
1 print("¡ Hola Scala !")
2 # Esto significa imprimir pantalla
3 print("¡Hola GitHub!")
```



5.7 Git Clone

Contexto:

Supongamos que te has incorporado hoy al equipo de big data, y para versionar los cambios que vas a hacer en tu trabajo te piden clonar el repositorio remoto a tu equipo local (lo contrario a lo que se hizo en las diapositivas anteriores). En otras palabras, necesitamos clonar lo que hay en el repositorio remoto en una carpeta en mi directorio local para poder empezar a trabajar.

git clone

El comando **git clone** se utiliza para crear una copia local de un repositorio Git existente. En otras palabras, te permite descargar todo el código, el historial de commits y la estructura de carpetas de un proyecto alojado en un servidor remoto a tu ordenador.

¿Qué necesitas para usar git clone?

La URL del repositorio remoto: Esta URL te indica la ubicación del repositorio que deseas clonar. La encontrarás en la plataforma donde esté alojado el proyecto, como GitHub, GitLab o Bitbucket. Suele ser similar a https://github.com/usuario/nombre_del_repositorio.git.

Un cliente de Git instalado: Necesitas tener instalado un cliente de Git en tu computadora para ejecutar el comando git clone.

Pasos para usar git clone

1. Desde nuestro ordenador local abrimos una terminal (en nuestro caso Git Bash) y creamos la carpeta donde se va a guardar mi repositorio a clonar. Después nos movemos a ese directorio con el comando cd

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/proyecto2 (main)
$ cd ..
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ mkdir pruebaclon
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git
$ cd pruebaclon
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon
$ []
```

2. Nos vamos a GitHub , buscamos el enlace ssh de nuestro repositorio:

The image shows a screenshot of a GitHub repository page for 'Curso-Git-Github'. The repository is public and has one branch ('main') and no tags. The repository was created by 'rodbalza' and has 19 commits. A red box highlights the 'Code' button in the top right corner of the main content area. A red arrow points from this button to a detailed view of the cloning options on the right. In this view, there are three tabs: 'Local', 'Codespaces', and 'Clone'. The 'Clone' tab is selected, showing three options: 'HTTPS' (disabled), 'SSH' (selected and highlighted with a red box), and 'GitHub CLI'. The SSH URL is displayed as 'git@github.com:rodbalza/Curso-Git-Github.git'. A red box highlights this URL, and another red arrow points from it to the text 'Copiamos la URL (ssh)' located at the bottom left.

Copiamos la URL (ssh)

Curso-Git-Github Public

main 1 Branch 0 Tags

rodbalza cambios en hello-git.py: añadí la linea 3 8ebfa2d · 42 minutes ago 19 Commits

ficheros cambios en hello-git.py: añadí la linea 3 42 minutes ago

.gitignore Se ha creado .gitignore y añadido el directorio /sin-imp... last week

README.md Create README.md 2 hours ago

README

Pin Unwatch 1

Go to file + <> Code ▾

Local Codespaces

Clone

HTTPS SSH GitHub CLI

git@github.com:rodbalza/Curso-Git-Github.git

Use a password-protected SSH key.

Open with GitHub Desktop

Download ZIP

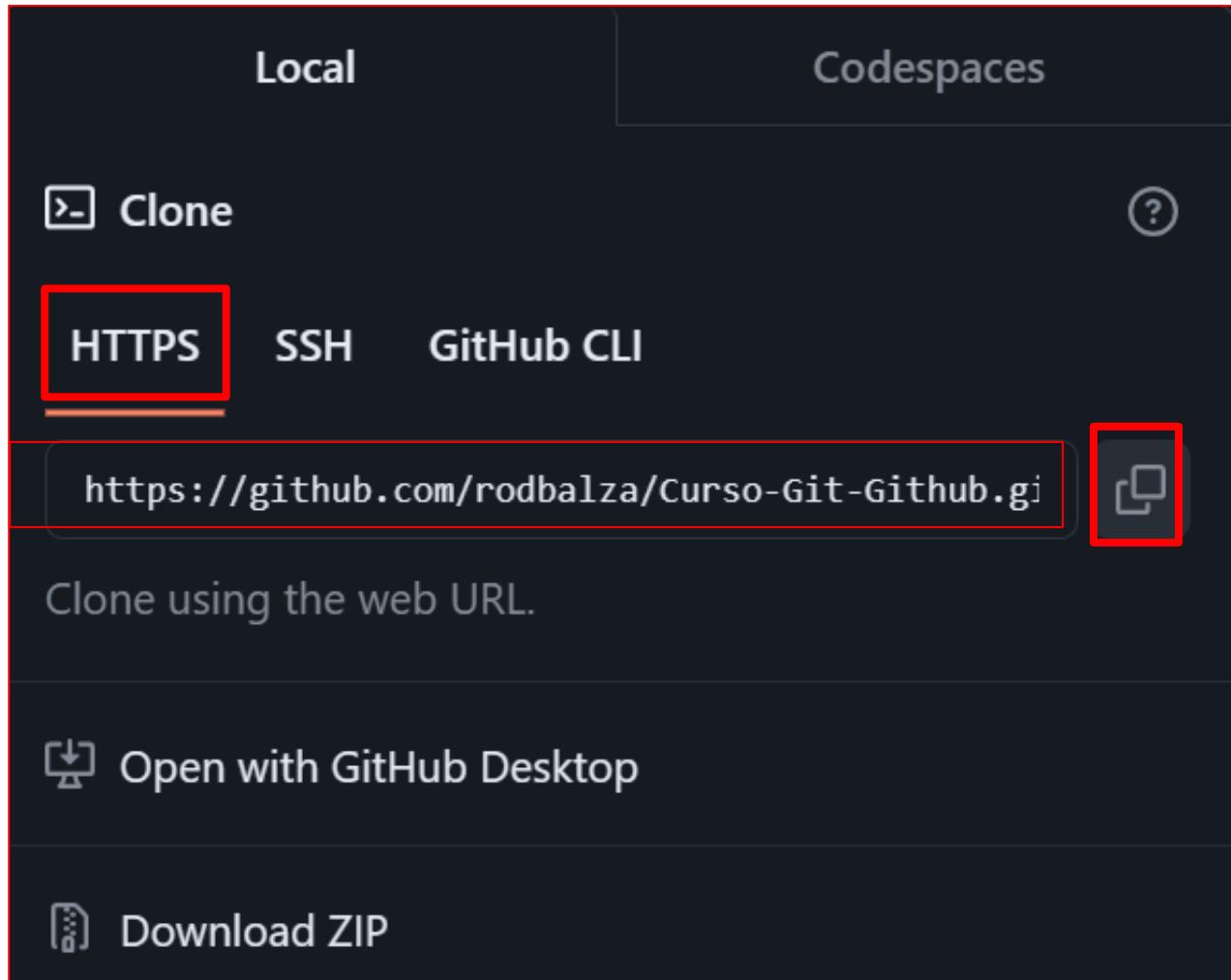
3. Ejecuta el comando git clone seguido de la URL del repositorio remoto: Una vez estés en el directorio deseado, escribe el siguiente comando:

git clone git@github.com:rodbalza/Curso-Git-Github.git

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon
$ git clone git@github.com:rodbalza/Curso-Git-Github.git
Cloning into 'Curso-Git-Github'...
remote: Enumerating objects: 67, done.
remote: Counting objects: 100% (67/67), done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 67 (delta 16), reused 64 (delta 15), pack-reused 0
Receiving objects: 100% (67/67), 7.13 KiB | 608.00 KiB/s, done.
Resolving deltas: 100% (16/16), done.
```

Si falla el clonado, usar la URL de https en de ssh

```
git clone https://github.com/rodbalza/Curso-Git-Github.git
```



Si nos vamos a nuestro directorio encontraremos el repositorio clonado

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon
$ ls
Curso-Git-Github/
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon
$ cd Curso-Git-Github
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github (main)
$ ls
ficheros/ README.md
```

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

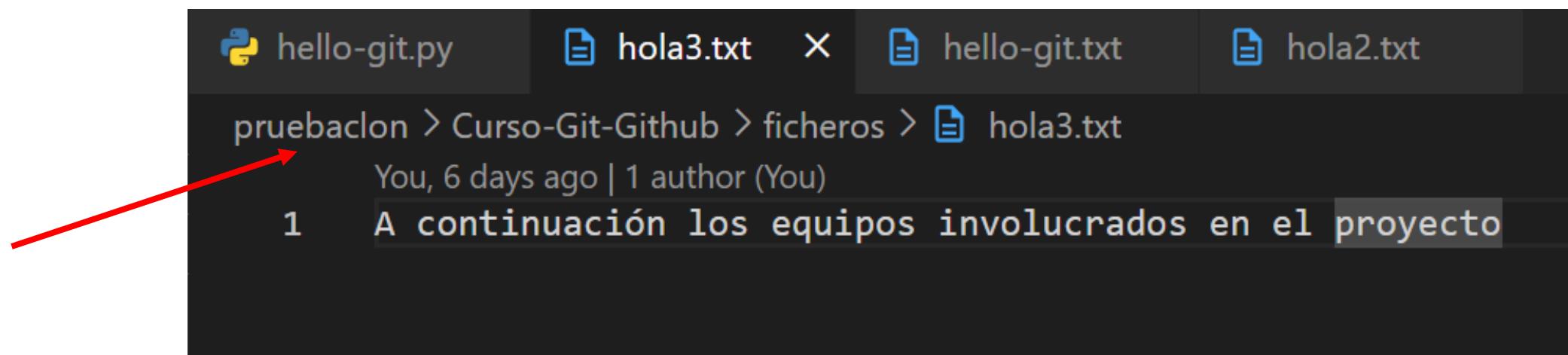
Vamos a efectuar otra edición desde local para hacer el push final

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github (main)
$ cd ficheros

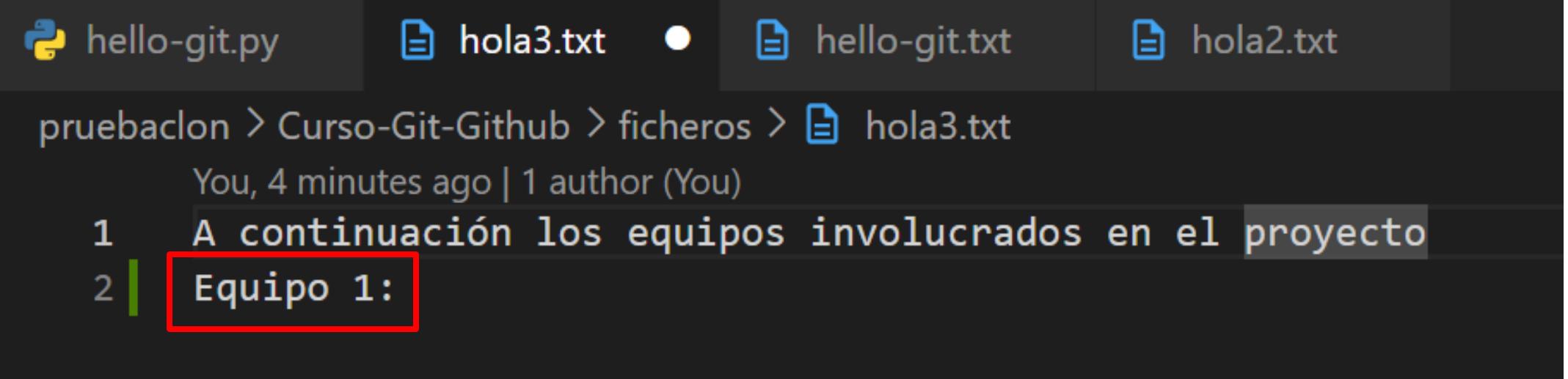
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ ls
hello-git.py  hello-git.txt  hola3.txt  lib/

juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ code hola3.txt
```

Al escribir **code hola3.txt** se actualiza el sistema de ficheros en visual studio code:



Añadimos contenido al fichero hola3.txt:



```
hello-git.py  hola3.txt  ●  hello-git.txt  hola2.txt
pruebaclon > Curso-Git-Github > ficheros > hola3.txt
You, 4 minutes ago | 1 author (You)
1 A continuación los equipos involucrados en el proyecto
2 Equipo 1:
```

Guardar, añadir y confirmar cambios:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ git commit -a -m "añadí una segunda línea en el fichero hola3.txt"
[main 82a894a] añadí una segunda línea en el fichero hola3.txt
1 file changed, 2 insertions(+), 1 deletion(-)
```

Pedimos status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Luego escribimos git push

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 412 bytes | 412.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/rodbalza/Curso-Git-Github.git
  8ebfa2d..82a894a  main -> main
```

Otro status:

```
juanj@balrodjj MINGW64 ~/OneDrive/Documentos/modulo3-git/pruebaclon/Curso-Git-Github/ficheros (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

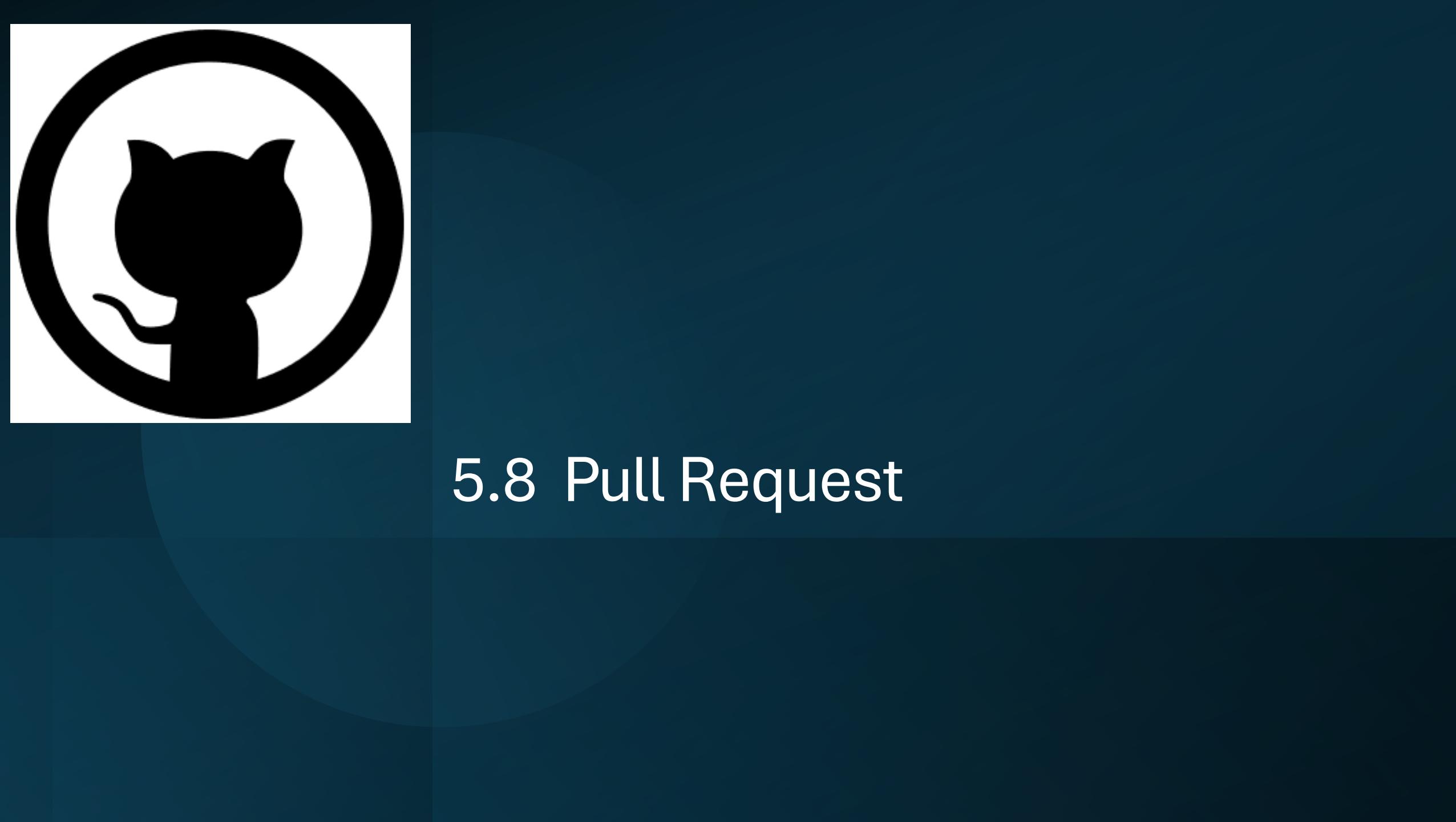
Los cambios se pueden ver desde GitHub:

Curso-Git-Github / ficheros / hola3.txt

 **rodbalza** añadí una segunda línea en el fichero hola3.txt

Code **Blame** 2 lines (2 loc) · 65 Bytes

1	A continuación los equipos involucrados en el proyecto
2	Equipo 1:



5.8 Pull Request

Pull Requests

Un pull request (solicitud de incorporación de cambios), también conocido como PR, es una forma en la que un desarrollador propone y colabora en cambios a un repositorio de código. Sirve como notificación para otros desarrolladores de que has realizado cambios en una rama y te gustaría que los revisen y, potencialmente, los fusionen con la rama principal.

¿Qué hace un pull requests?

- Propone cambios: Cuando creas un pull request, básicamente estás proponiendo tus cambios a los mantenedores del repositorio u otros colaboradores. Ellos pueden revisar tu código, darte feedback y sugerir modificaciones.
- Facilita la colaboración: Los pull requests fomentan la colaboración al permitir que varios desarrolladores revisen y discutan los cambios antes de fusionarlos con la base de código principal. Esto ayuda a garantizar la calidad y coherencia del código.
- Rastrea cambios: Cada pull request contiene una descripción detallada de los cambios que has realizado, incluidos los archivos modificados, las líneas agregadas o eliminadas, y cualquier mensaje de confirmación. Esto facilita a los revisores comprender el alcance de tus cambios.
- Habilita revisiones de código: Los pull requests permiten revisiones de código, donde los revisores pueden comentar líneas específicas de código, sugerir mejoras e identificar posibles problemas. Este ciclo de feedback ayuda a mejorar la calidad general del código.
- Fusiona cambios: Una vez que se aprueba un pull request y se realizan los cambios necesarios, el revisor puede fusionar los cambios en la rama principal. Esto integra tus contribuciones a la base de código principal.

¿Cuándo usar pull requests?:

- Regularmente: Es una buena práctica usar pull requests con frecuencia, especialmente cuando trabajas en un proyecto colaborativo. Esto permite una retroalimentación e integración continua de los cambios.
- Antes de fusionar: Antes de fusionar tus cambios en la rama principal, siempre crea un pull request para obtener feedback y asegurarte de que tus cambios estén listos para la integración.
- Después de cambios significativos: Si has realizado cambios significativos en el código, crear un pull request permite una revisión exhaustiva y una discusión antes de la fusión.

Para crear un pull request en GitHub, sigue estos pasos:

1. Crea una rama: Crea una rama separada para tus cambios. Esto asegura que tus cambios estén aislados de la rama principal hasta que estén listos para fusionarse.
2. Haz cambios: Realiza los cambios deseados en el código de tu rama.
3. Confirma los cambios: Confirma tus cambios con mensajes de confirmación descriptivos que expliquen el propósito de cada cambio.
4. Sube los cambios: Sube tu rama al repositorio remoto.
5. Crea un pull request: En el sitio web del repositorio, crea un pull request desde tu rama a la rama principal.
6. Describe los cambios: Proporciona una descripción clara de los cambios que has realizado y la lógica detrás de ellos.

Para crear un pull request en GitHub, sigue estos pasos:

7. Sigue los cambios: Haz los cambios necesarios en tu código y guarda las modificaciones.
8. Aborda el feedback: Aborda cualquier comentario o sugerencia proporcionada por los revisores.
9. Fusiona los cambios: Una vez que se aprueba el pull request, el revisor puede fusionar tus cambios en la rama principal.

Ejemplo:

1. Entra a tu GitHub e inicia sesión.
2. Entra al [siguiente repositorio](#) y haz un ‘fork’ para que tengas una copia del repositorio en tu GitHub.
3. Edita el fichero **colaborativo.txt** añadiendo una línea cualquiera, haz un commit de tus cambios desde el mismo GitHub.

A screenshot of a GitHub code editor interface. At the top, it shows a commit by user 'juanjorod' with the message 'Update colaborativo.txt'. The commit hash is 'adb2fce' and it was made '12 minutes ago'. To the right of the commit details is a 'History' button. Below the commit, there are tabs for 'Code' (which is selected), 'Blame', and 'Raw'. There are also icons for 'Copy', 'Download', and 'Edit'. The 'Edit' icon is highlighted with a red box. In the center, there's a 'Copilot' button with the text 'Code 55% faster with GitHub Copilot'. The code editor displays three lines of text:

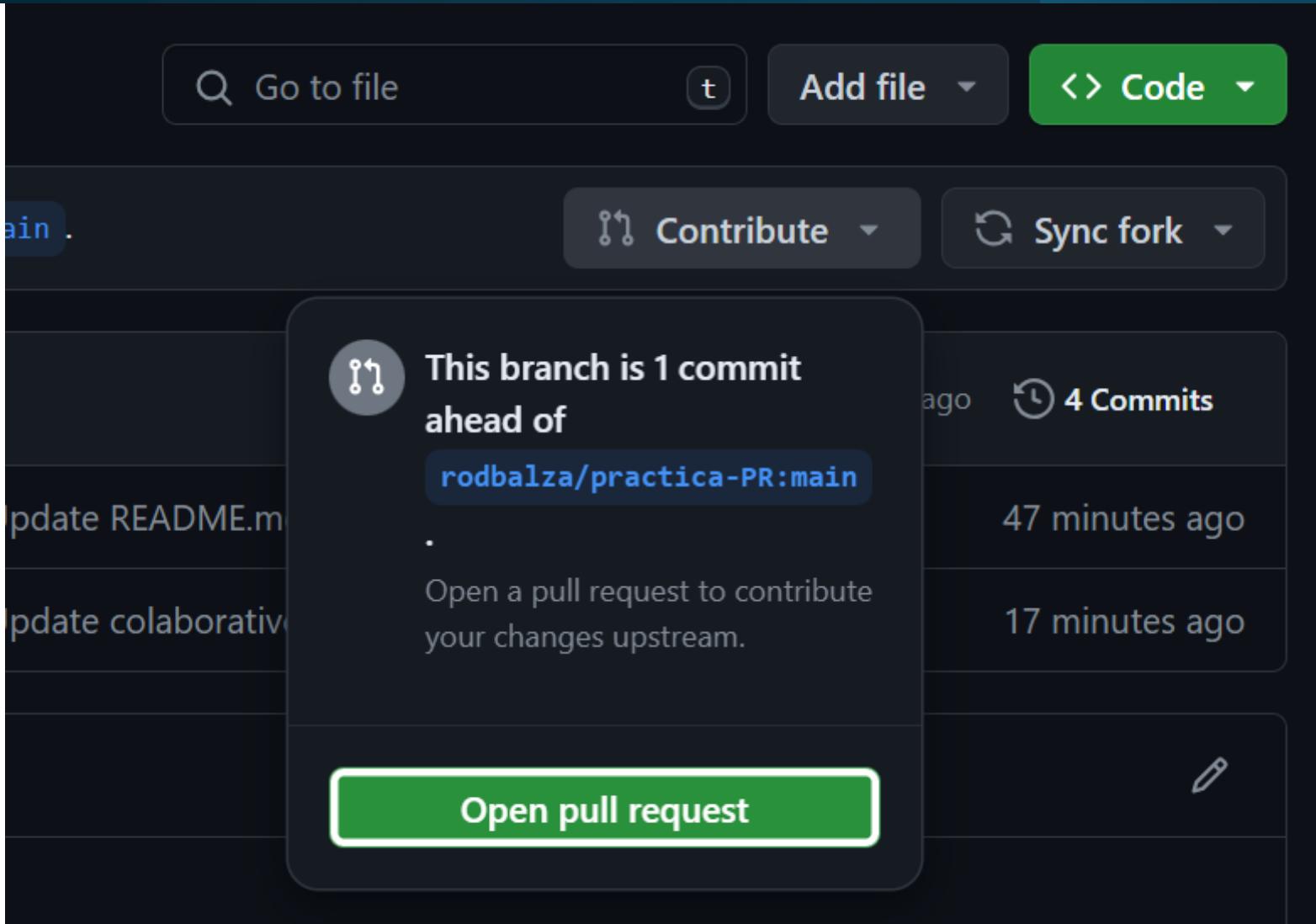
```
1  ¡Hola! colabora conmigo...
2
3  Hola! Mi nombre es juanjorod!
```

4. Una vez hecho el commit haz click sobre el botón **Contribute**:

This screenshot shows a GitHub repository interface. At the top, there are navigation links: 'main' (with a dropdown arrow), '1 Branch' (with a dropdown arrow), '0 Tags' (with a dropdown arrow), a search bar ('Go to file'), a 't' icon, an 'Add file' button (with a dropdown arrow), and a green 'Code' button (with a dropdown arrow). Below this, a message states 'This branch is 1 commit ahead of rodbalza/practica-PR:main'. On the right side, there are two buttons: 'Contribute' (with a dropdown arrow) and 'Sync fork' (with a dropdown arrow). The 'Contribute' button is highlighted with a red box. The main content area displays a list of commits:

- juanjorod** Update colaborativo.txt · 16 minutes ago · 4 Commits
- README.md** Update README.md · 46 minutes ago
- colaborativo.txt** Update colaborativo.txt · 16 minutes ago
- README** ·

5. Haz click sobre open pull requests:



6. Haz click en Create pull request

Discuss and review the changes in this comparison with others. [Learn about pull requests](#)

[Create pull request](#)

-o 1 commit 1 file changed 1 contributor

Commits on Apr 24, 2024

Update colaborativo.txt ...

 **juanjorod** committed 18 minutes ago

Verified  adb2fce 

 Showing 1 changed file with 2 additions and 0 deletions.

Split **Unified**

2 colabroativo.txt 

... ... @@ -1 +1,3 @@

1 1 ¡Hola! colabora conmigo...

2 +

3 + Hola! Mi nombre es juanjorod!

7. Añadir un título y escribir detalles de tu request en la descripción. Después click en **Create pull request**

The screenshot shows the GitHub interface for creating a pull request. At the top left, there is a user icon and the text "Add a title". Below it, a red box highlights the input field containing "Update colaborativo.txt (Sugerencias)". To the right, there are "Helpful resources" and a link to "GitHub Community Guidelines".

Below the title section is a "Add a description" area. It includes a "Write" tab and a "Preview" tab. The "Write" tab is active, showing a rich text editor toolbar with various icons for bold, italic, and code blocks. The preview area contains the text "Mi presentación en línea 3" and "He añadido algunos cambios que podrían ser interesantes", both of which are highlighted with a red box.

At the bottom, there is a note that "Markdown is supported" and a file upload area with the placeholder "Paste, drop, or click to add files". There is also a checkbox for "Allow edits by maintainers" and a large green "Create pull request" button with a dropdown arrow, which is also highlighted with a red box.

Luego queda esperar por la revisión del jefe

Update colaborativo.txt (Sugerencias) #1

[Open](#) juanjorod wants to merge 1 commit into `robalza:main` from `juanjorod:main`

Conversation 0 Commits 1 Checks 0 Files changed 1 +2 -0

juanjorod commented 3 minutes ago

Mi presentación en línea 3
He añadido algunos cambios que podrían ser interesantes

Update colaborativo.txt adb2fce

Add more commits by pushing to the `main` branch on [juanjorod/practica-PR](#).

Checking for ability to merge automatically...
Hang in there while we check the branch's status.

Reviewers
No reviews
Still in progress? [Convert to draft](#)

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone

El jefe ve la solicitud, revisa y responde a los cambios.

[Update colaborativo.txt \(Sugerencias\) #1](#)

[Open](#) juanjorod wants to merge 1 commit into [rodbalza:main](#) from [juanjorod:main](#) ⚙

Conversation 0 Commits 1 Checks 0 Files changed 1

 juanjorod commented 5 minutes ago First-time contributor ...

Mi presentación en línea 3

He añadido algunos cambios que podrían ser interesantes

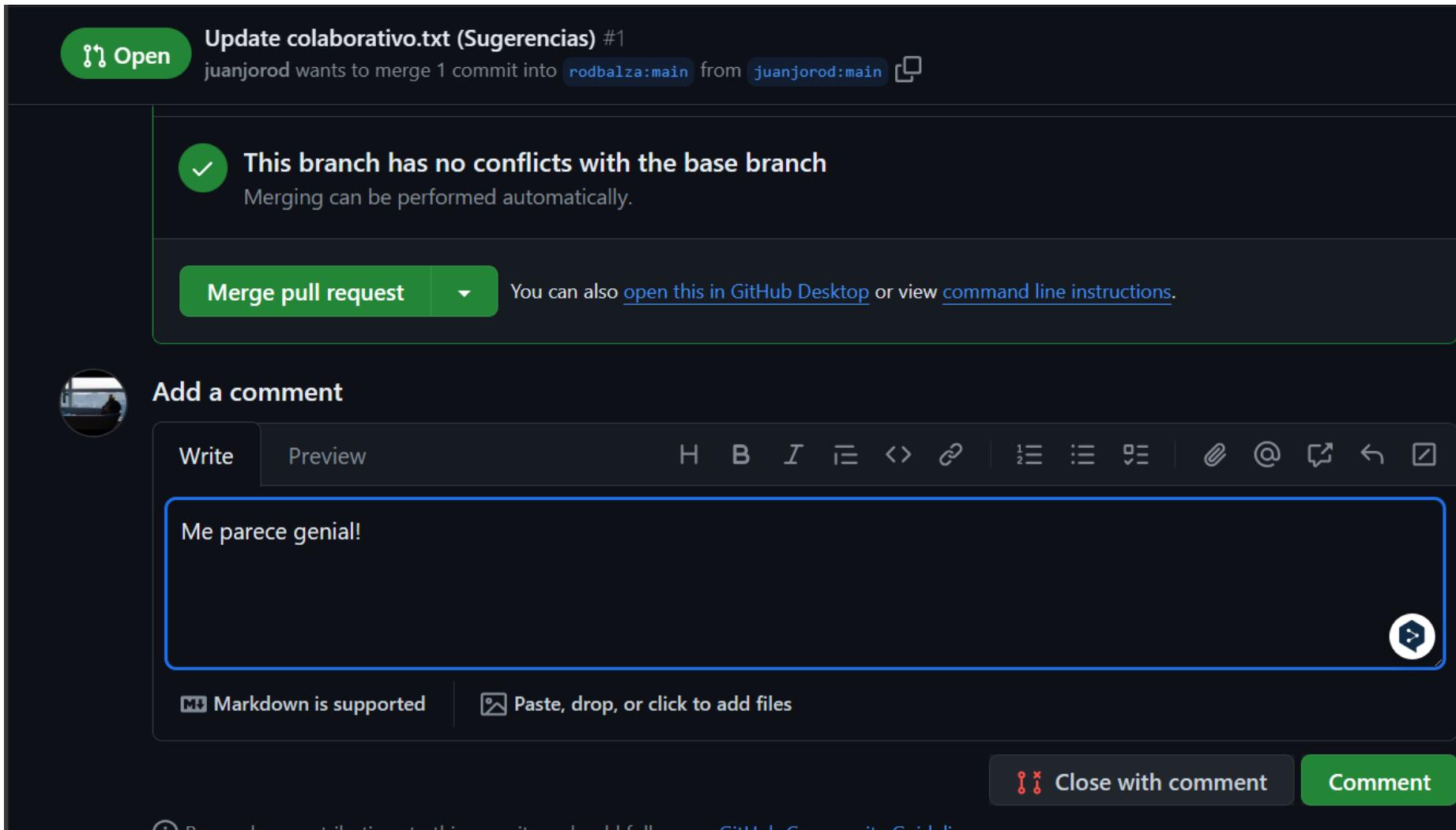


-o-  Update colaborativo.txt ...

Looks like you got your first contribution! Visit your community profile to learn more about recommended open source practices.

[Go to community profile](#)

Supongamos que el jefe aprueba los cambios y te deja el mensaje:



Una vez el jefe aprueba tus cambios, en tu GitHub aparece la aprobación y un comentario

Update colaborativo.txt (Sugerencias) #1

 Merged rodbalza merged 1 commit into rodbalza:main from juanjorod:main now

Conversation 0 Commits 1 Checks 0 Files changed 1

 juanjorod commented 9 minutes ago

Mi presentación en línea 3
He añadido algunos cambios que podrían ser interesantes



Update colaborativo.txt ... Verified adb2fce

 rodbalza merged commit cfb0341 into rodbalza:main now

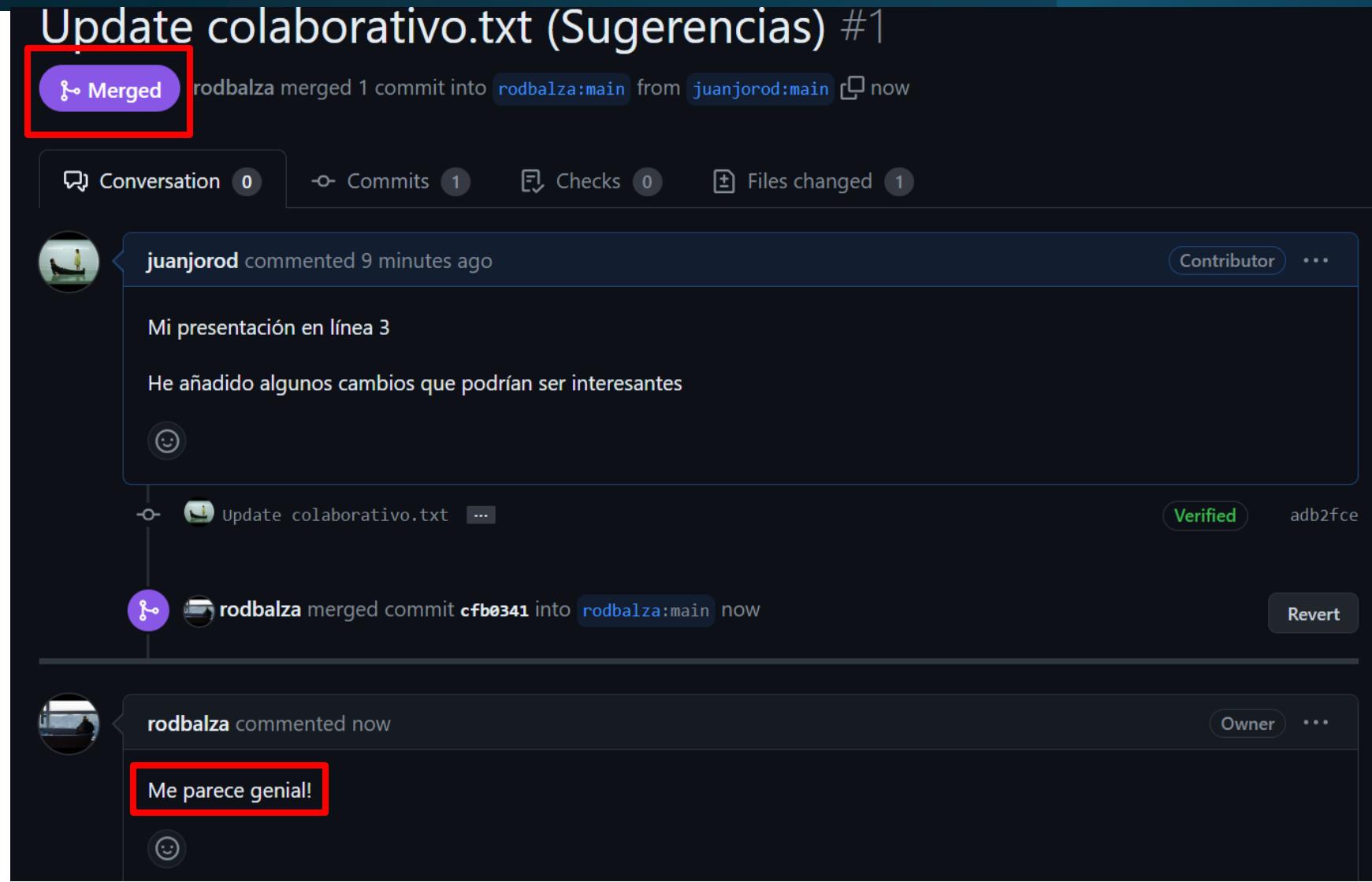


 rodbalza commented now

Owner ...

 Me parece genial!





Practica 05(no)

1. Pull Requests. Esta práctica es grupal, los miembros de cada grupo serán tus compañeros de tu misma fila. Elegir un jefe y los demás hacen el rol de empleados. El jefe crea el repositorio y lo comparte con los empleados. Simular lo practicado en las páginas anteriores.
2. Cambiar de jefe y repetir los pasos anteriores.
3. Todos los miembros deben practicar ambos roles: Jefe y empleado.

Realizar la entrega en un archivo comprimido en .zip