TEMA 3: Git y GitHub

(parte I)

Ciclo de Desarrollo de Aplicaciones Web

Módulo: Entornos de Desarrollo

Contenido

1- Introducción

➢ ¿Qué es Git?

Git es una herramienta de control de versiones.

Permite controlar el proceso de creación de software llevando un **registro exhaustivo de todos los cambios realizados**. Con Git podemos:

- crear versiones del proyecto
- ramificar el desarrollo en diferentes flujos de trabajo (ramas o branches)
- volver atrás si algo sale mal (revertir cambios o commits).

Ejemplo: Imagina que estás desarrollando una web con varios compañeros. Uno de ellos cambia el archivo *index.html* y rompe el menú. Gracias a Git puedes volver exactamente al punto anterior en el que todo funcionaba.

Git es un programa **libre y gratuito** que se distribuye mediante la licencia GNU (GPL 2.0)

NOTA: *En realidad se puede extender el uso de Git a proyectos no solo de software sino también de documentos (hay muchos libros escritos con Git), presentaciones y, en general, a casi cualquier tipo de contenido*.

➢ ¿Qué es GitHub?

GitHub es una plataforma web que permite alojar proyectos controlados mediante Git.

Podemos pensar en GitHub como el "espejo" en la nube (en la red) en el que duplicar los proyectos que tenemos en nuestro propio ordenador (en local). Además, es una **red social para programadores**, donde los usuarios pueden:

- ver y descargar proyectos de otros
- dar "estrellas" ⭐ a los repositorios que les gustan
- seguir a otros desarrolladores
- colaborar en proyectos comunes mediante *pull requests* e *issues*.

Ejemplo: Un alumno sube su proyecto de clase a GitHub. Otro compañero puede hacerle una sugerencia o corregirle un error a través de una *pull request*. Esto simula perfectamente cómo se trabaja hoy en día en las empresas tecnológicas.

Hoy en día GitHub es tan influyente que muchas empresas revisan el perfil de GitHub de los candidatos durante los procesos de selección. Por eso se le suele llamar *"el escaparate del programador"*.

Su uso es gratuito para los **proyectos públicos**, sin límite en cuanto a la cantidad de repositorios.

➢ ¿Por qué utilizar Git junto con GitHub?

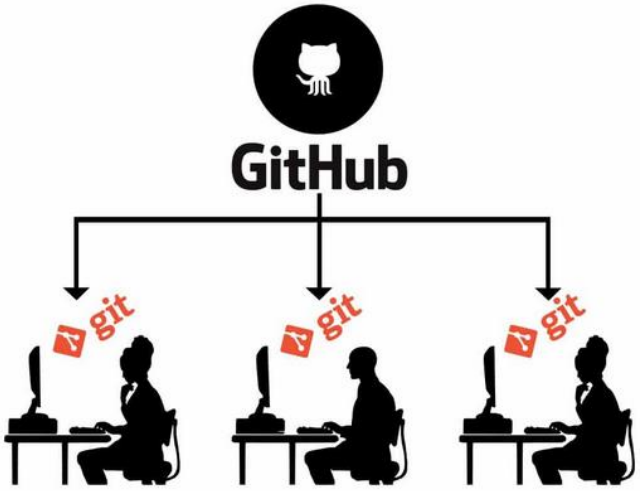
Usar Git por sí solo ya nos permite llevar un control de versiones muy preciso en nuestro ordenador.

Sin embargo, al combinarlo con GitHub, ganamos algo fundamental: **colaboración y trabajo en equipo**.

Ventajas principales

- **Copia de seguridad en la nube:** Todo el código queda almacenado online, de forma que si se rompe o se pierde el ordenador, el proyecto sigue a salvo en GitHub.
- **Trabajo en equipo sin caos:** Varias personas pueden modificar el mismo proyecto sin pisarse el trabajo gracias a las ramas (*branches*) y a los *merge* controlados.

  Ejemplo: mientras uno programa el sistema de *login*, otro puede trabajar en el diseño sin interferir.



- **Historial completo de cambios:** Git mantiene un registro de quién cambió qué y cuándo, lo que facilita detectar errores o retroceder a versiones anteriores.
- **Revisiones y sugerencias** (*pull requests*): GitHub permite que otros revisen tu código antes de incorporarlo al proyecto principal.

  Ejemplo: igual que un profesor revisa un trabajo antes de aprobarlo, los compañeros revisan el código antes de integrarlo.
- **Visibilidad y reputación profesional:** GitHub funciona también como una red social. Tener un perfil activo con proyectos propios es una carta de presentación para empresas o equipos de desarrollo.

En resumen:

- **Git** = herramienta para **controlar versiones** de un proyecto.
- **GitHub** = **plataforma web** basada en Git que facilita la **colaboración online** entre desarrolladores.

2- Instalación y Configuración de Git y GitHub

2.1. En cuanto a Git

Para instalar Git en sistema Linux (por ejemplo, Ubuntu), basta con teclear la siguiente línea en una ventana de terminal:

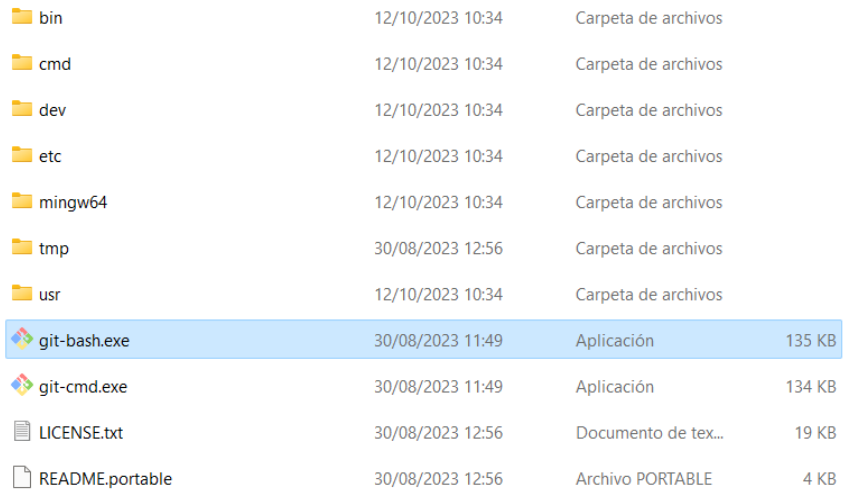
```
sudo apt install git
```

Para instalar Git en Windows, puedes consultar los binarios disponibles en https://git-scm.com/install/

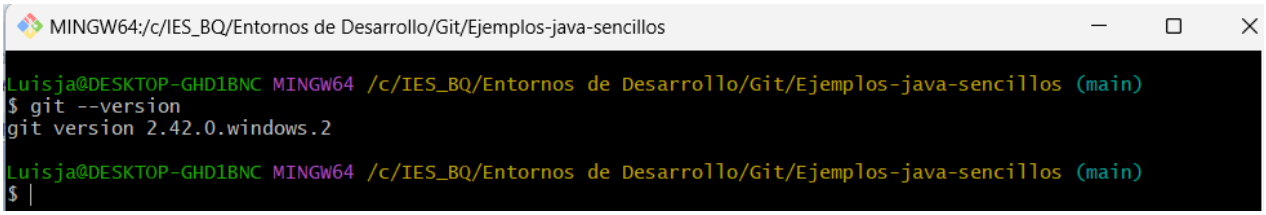
- En este curso, utilizaremos la versión portable para Windows que puedes encontrar en el curso del Aula Virtual). No requiere instalación: basta con descomprimir la carpeta y ejecutar **`git-bash.exe`** (para similar la interfaz Linux) o **`git-cmd.exe`** (para simular la interfaz Windows).

Una vez descargado (e instalado si procede), **Git se utiliza mediante comandos** en una ventana de terminal.

Vamos a trabajar con la opción en la cual podemos abrir la interfaz de línea de comandos (Shell) de Linux Bash:



Podemos comprobar que el programa se ha instalado correctamente comprobando el número de versión:



Configuración inicial de Git

Antes de usar Git es conveniente emplear unos minutos en configurar la herramienta.

Lo primero que hay que hacer es registrar nuestro nombre y correo electrónico ya que estos datos aparecerán en los cambios (*commits*) que realicemos:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Entornos de Desarrollo/Git/Ejemplos-java-sencillos (main)
$ git config --global user.name "Luis Javier Menendez"
```

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Entornos de Desarrollo/Git/Ejemplos-java-sencillos (main)
$ git config --global user.email "luisjmt@educastur.org"
```

- El parámetro `--global` hace que esta configuración se aplique a todos los proyectos en tu ordenador. Puedes omitirlo si quieres que los datos sean diferentes para cada proyecto.

Cuando trabajemos con GitHub, Git te pedirá usuario y contraseña cada vez que hagas una acción remota (por ejemplo, subir un proyecto). Para evitar repetirlas constantemente, puedes activar el **gestor de credenciales** de Windows:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Entornos de Desarrollo/Git/Ejemplos-java-sencillos (main)
$ git config --global credential.helper wincred
```

Esto guarda de forma segura tus datos de acceso para futuras sesiones.

Es posible comprobar estas, y otras configuraciones, con la opción *--list*:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.interactive=true
color.ui=auto
help.format=html
diff.astextplain.textconv=astextplain
rebase.autosquash=true
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
credential.helper=!"C:/IES_BQ/Entornos de Desarrollo/Recursos/PortableGit/mingw64/bin/git-credential-manager.exe"
user.name=Luis Javier Menendez
user.email=luisjmt@educastur.org
credential.helper=wincred

Luisja@DESKTOP-GHD1BNC MINGW64 /
$
```

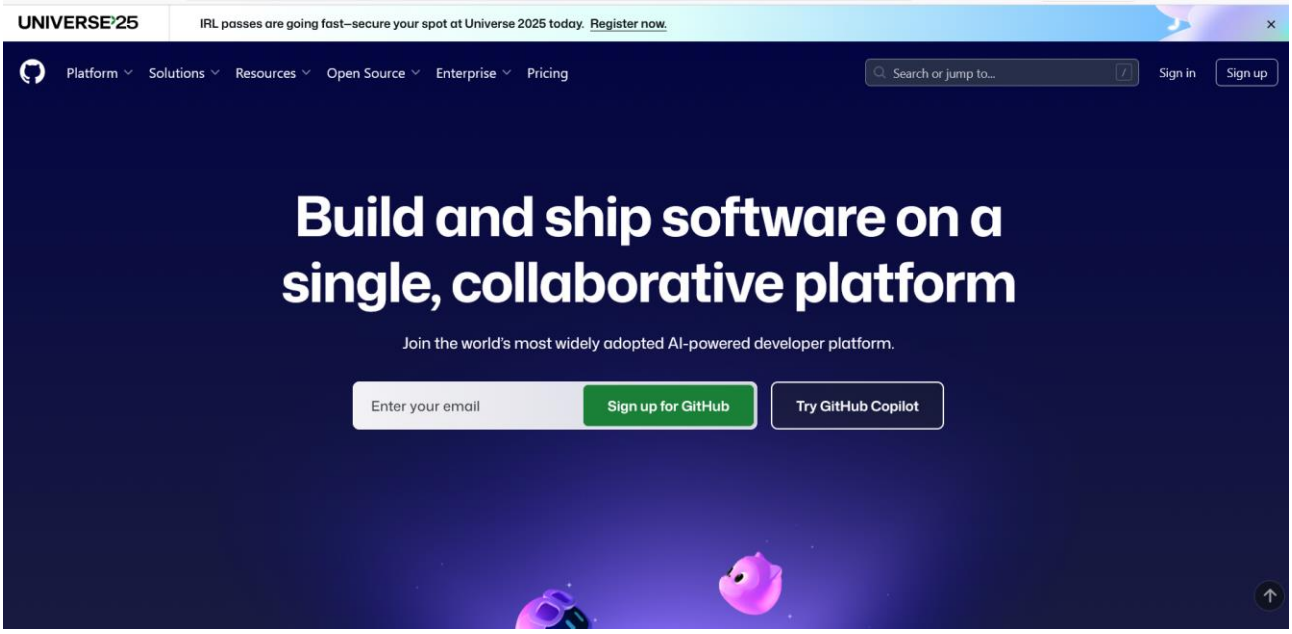
Aunque existen entornos visuales muy cómodos para Git (como GitHub Desktop, SourceTree o GitKraken), te recomiendo acostumbrarte a la terminal.

Aprender Git con comandos te permitirá entender mejor lo que ocurre "detrás del botón", resolver problemas con más autonomía y desenvolverte en cualquier entorno profesional. Los auténticos programadores no le temen a la línea de comandos.

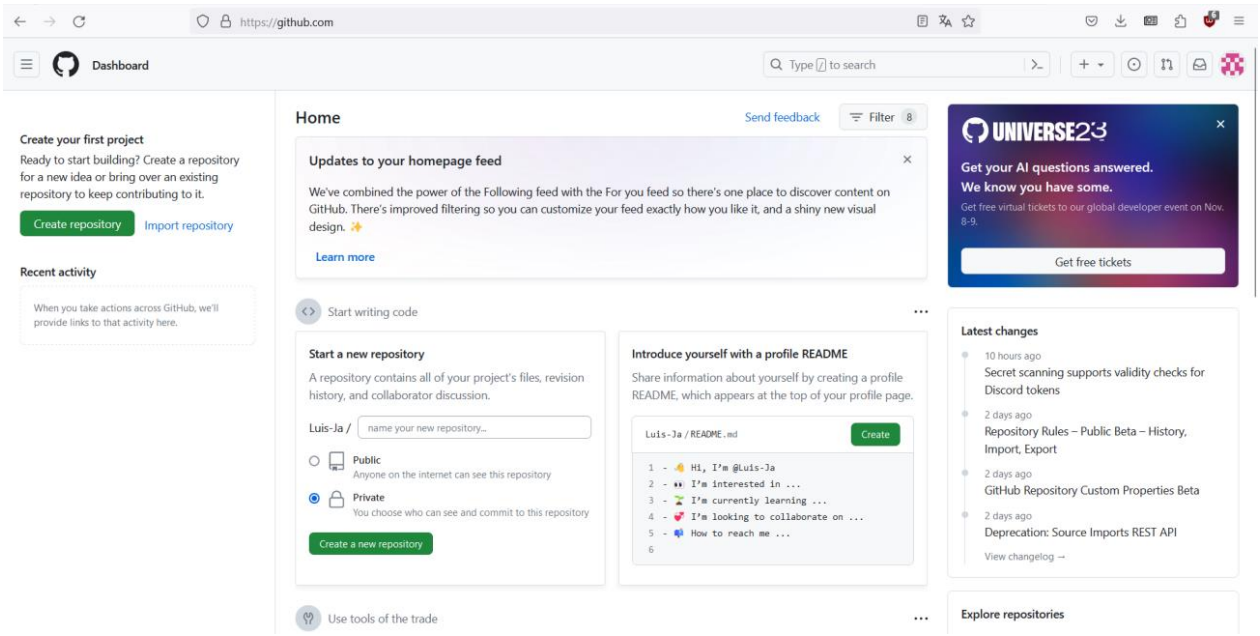
2.2. En cuanto a GitHub

Lo primero que tenemos que hacer es crearnos una cuenta en GitHub.

Entra en la web de GitHub (https://github.com) y haz clic en el botón *Sign up*:

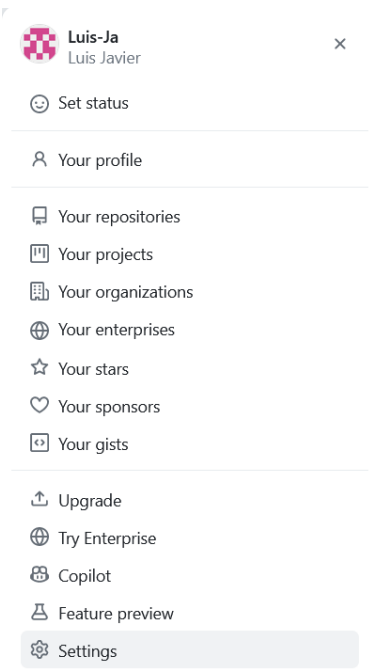


Una vez introducida la información necesaria (UserName, mail, password,…) ya tendremos creada nuestra cuenta y podremos entrar a nuestro **Dashboard inicial** donde veremos nuestro perfil y las primeras opciones para crear proyectos (tu Dashboard no tiene por qué ser exactamente igual este):

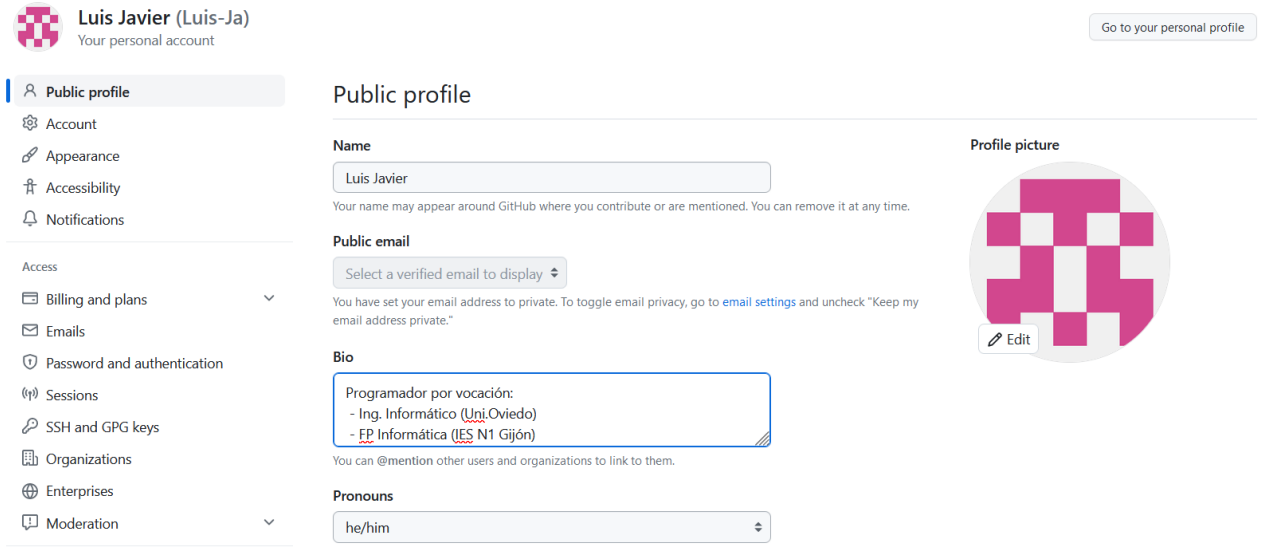


Configuración inicial de nuestro perfil en GitHub

Antes de realizar cualquier acción dentro de GitHub es conveniente realizar unos ajustes previos. En la esquina superior derecha, haz clic en el icono de tu avatar (inicialmente se asigna un avatar aleatorio por defecto). En el menú desplegable que aparece selecciona "*Settings*":



De esta manera accedes a tu página de perfil:



Podemos cambiar el Avatar por una foto más personal (siempre con ojo, hay muchos perfiles de buenos programadores en GitHub estropeados por una mala foto).

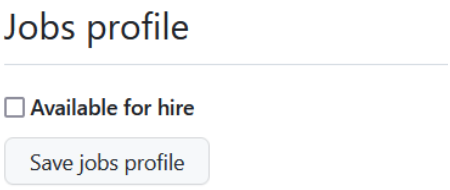
Escribe tu nombre completo (nombre y apellidos), con los correspondientes espacios, mayúsculas y tildes si procede. Anota en el campo *Bio* cualquier curso que hayas hecho o estés haciendo relacionado con la

programación o con la informática en general, en orden de importancia. Por ejemplo, si estás estudiando un ciclo formativo o un grado universitario, éste es el lugar idóneo para escribirlo: "*Estudiante de DAW en Mieres. Apasionado del desarrollo web y la automatización con Python.*"

Si tienes una web personal o un blog con contenidos relacionados con la programación o con algo que tenga que ver con la informática, escribe la dirección en el campo URL. Si no tienes nada de eso, escribe en este campo la dirección de tu perfil de LinkedIn. En caso de no tener cuenta en LinkedIn, te recomiendo encarecidamente que te des de alta (si no ahora, en un futuro próximo) → es la red social profesional por excelencia, muy práctica para hacer contactos a nivel corporativo y también es muy útil en la búsqueda de empleo.

El campo *Location* es muy importante. Escribe el nombre de tu ciudad y, entre paréntesis, el nombre de tu país en inglés. Imagina que una compañía importante está buscando programadores en tu ciudad o país. Lo primero que haría la empresa es mirar el ranking de un determinado lenguaje (por ejemplo Java) en tu ciudad/país. **Si no tienes relleno el campo *Location* simplemente no aparecerías en ese ranking (ni siquiera en los últimos puestos) y habrías perdido una buena oportunidad.**

Por último, si quieres decirle al mundo que estás disponible para que te contraten, marca la casilla "*Available for hire*". Hay aplicaciones que sondean los perfiles de GitHub en busca de candidatos para puestos de programación y miran si esta casilla está marcada. Recuerda que tu cuenta de GitHub es tu mejor carta de presentación como programador.



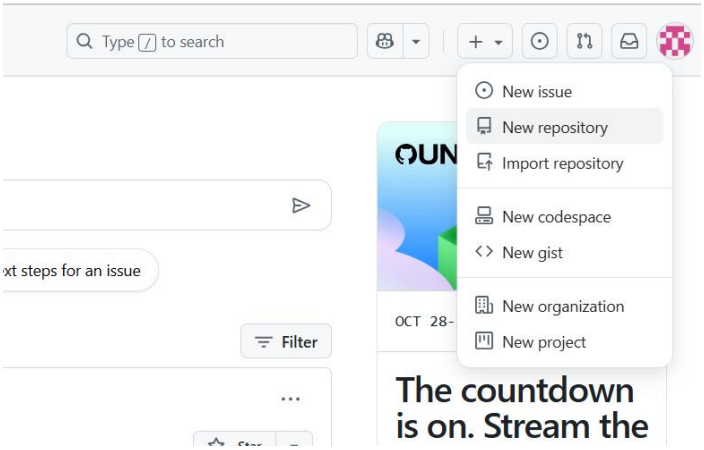
3- Creación de repositorios en GitHub

Un **repositorio** es el "contenedor" donde se guarda el código de un proyecto.

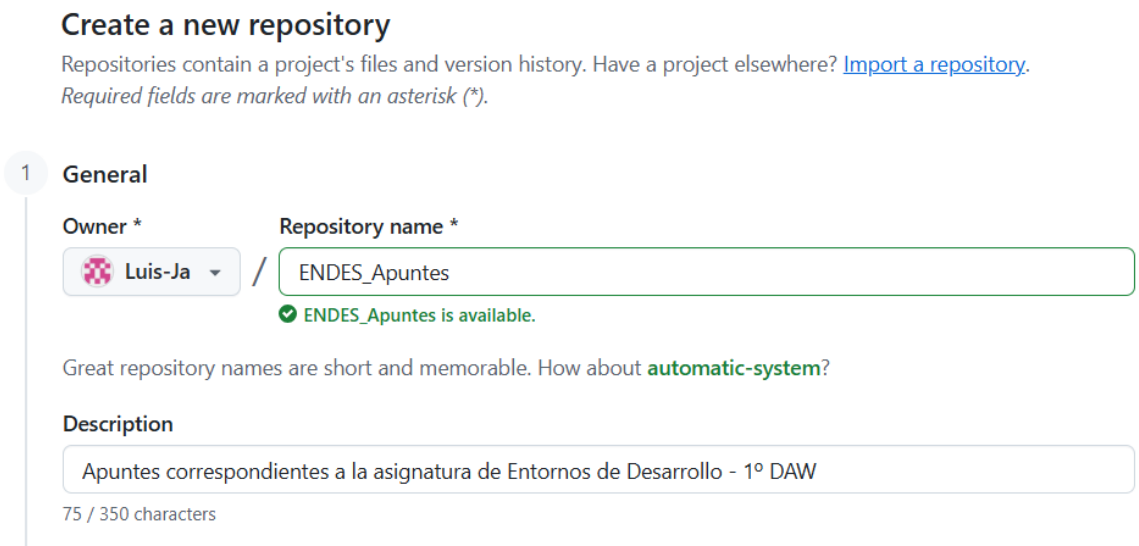
Incluye todos los archivos, carpetas y el historial de cambios (commits) que realices. En GitHub, el repositorio está **alojado en la nube**, lo que facilita compartirlo y colaborar con otros.

Para crear un nuevo repositorio:

- Haz click en el signo **+** que hay junto a tu avatar.
- Selecciona *New repository* en el menú desplegable:

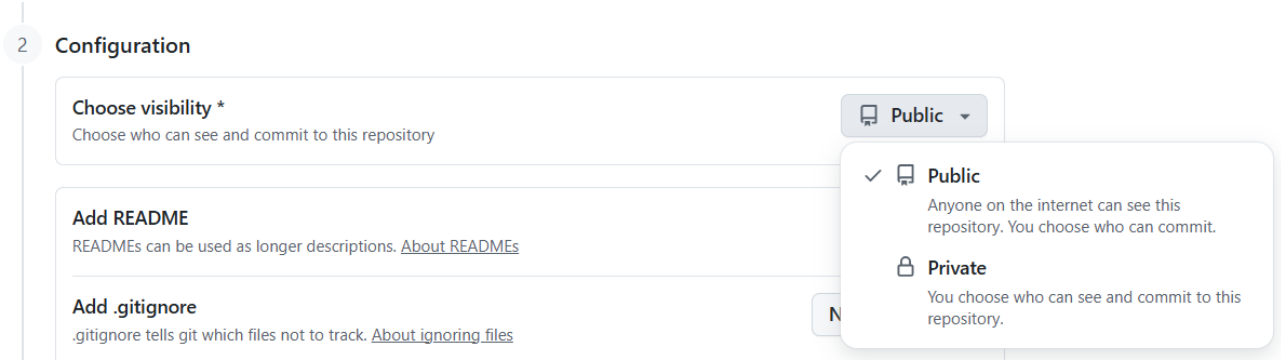


Escribe el nombre del repositorio en el campo *Repository name*. Debe ser un nombre lo más claro y conciso posible. **No se permiten espacios en blanco ni caracteres especiales en este campo. Las palabras pueden estar separadas por guiones**. Como ejemplo práctico, si el repositorio que estamos creando va a contener nuestros primeros programas en Java, un nombre perfecto puede ser "Ejemplos-java-sencillos":



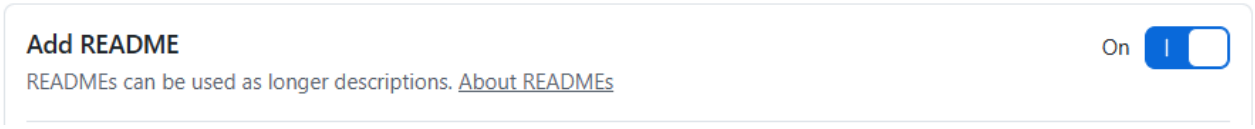
El campo *Description* es opcional pero es muy recomendable rellenarlo porque ayuda a entender el propósito del repositorio, incluso cuando uno vuelve meses después.

Deberemos especificar si nuestro repositorio será público y accesible para todo el mundo (opción más lógica), o si será privado.



(Esta opción puede ser modificada más adelante)

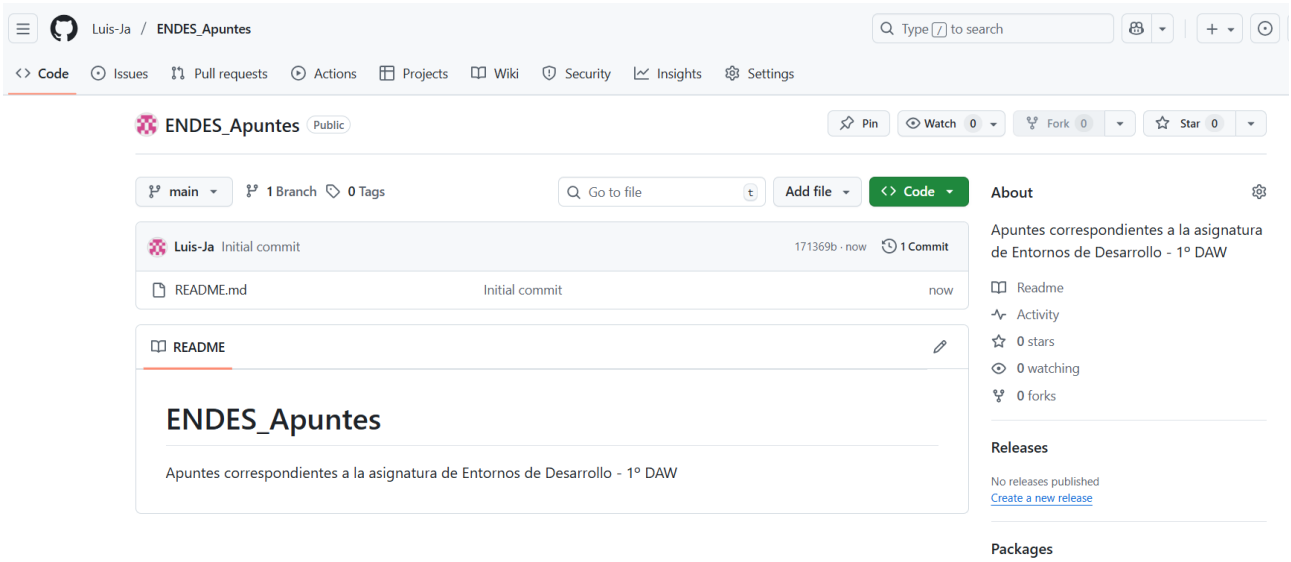
Este paso es muy importante: Marca la casilla "*Initialize this repository with README*":



Al marcar esta opción, se crea el fichero README.md que contiene por defecto el nombre y la descripción del repositorio; de esta forma ya hay algo dentro del proyecto, no está vacío y, por tanto, ya tenemos algo que clonar a nuestro ordenador (en el siguiente apartado veremos que significa esto).

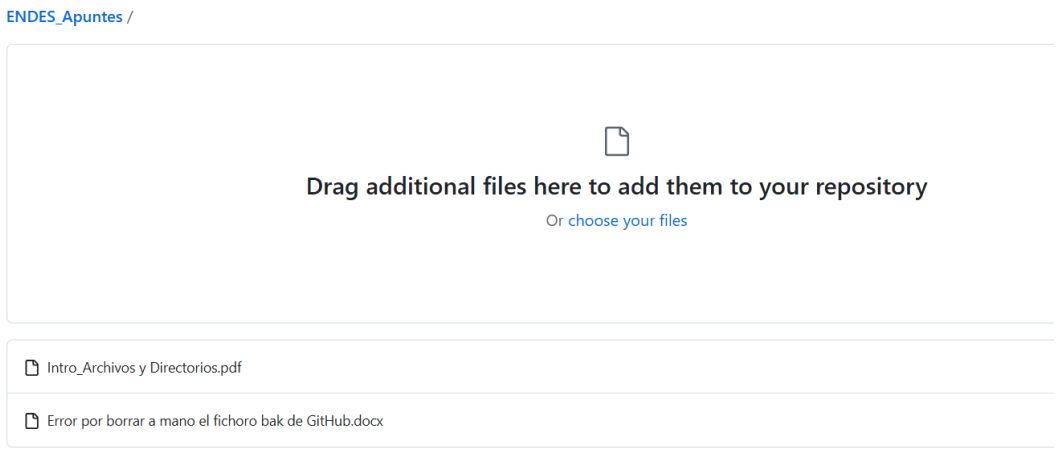
El resto de opciones (gitignore o license) podemos dejarlas por defecto.

Una vez creado el repositorio, aparece una vista general desde la que se puede ver el contenido del mismo y mucha otra información que iremos analizando más adelante:

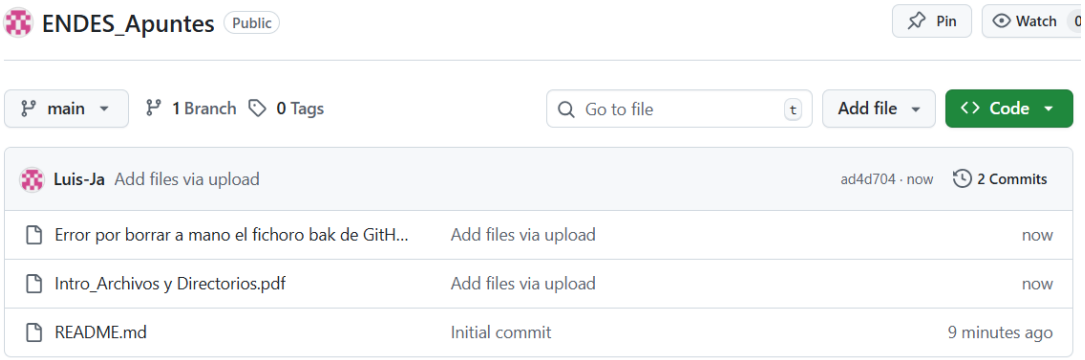


Ya tenemos nuestro primero repositorio recién creado en GitHub.

Puedes añadir archivos directamente desde el botón **"Add file → Upload files"**. Por ejemplo:



Pulsamos el botón de *Commit changes* y los ficheros quedarán subidos a nuestro repositorio:



Y vemos que se almacenan en nuestro repositorio.

Subir archivos directamente en GitHub es útil para pequeñas pruebas o ejercicios rápidos. Sin embargo, **en proyectos reales trabajaremos desde Git en nuestro ordenador (en LOCAL), y los archivos se subirán al repositorio de GitHub asociado haciendo commits y pushes, lo que nos dará un control total sobre el historial**.

En el siguiente apartado aprenderemos a clonar el repositorio y conectar nuestro Git local con el remoto.

4- Clonando Repositorios de GitHub a nuestra máquina

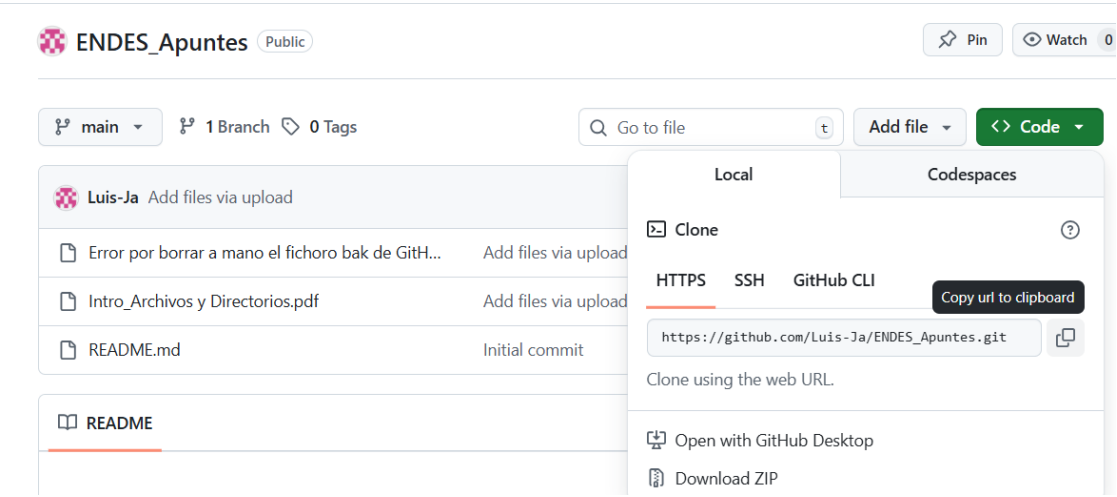
Clonar significa hacer una copia exacta; por tanto, clonar un repositorio es hacer una copia idéntica de un proyecto que existe en GitHub y llevártela a tu máquina.

Estarás pensando ¿qué diferencia hay entre clonar un repositorio y simplemente bajar los archivos? En principio no hay diferencia, pero si el repositorio de GitHub cambia porque se añaden, se borran o se modifican ficheros, **entonces tendrás en tu máquina una versión desactualizada**, ya que no será un clon de lo que hay en GitHub. Sin embargo, cuando se ha clonado un repositorio con Git, la actualización es algo trivial como veremos en el siguiente apartado.

Acción	¿Se guarda historial de cambios?	¿Puedes actualizar desde GitHub a Git facilmente?	¿Puedes actualizar de Git a GitHub?
Descargar Una copia "muerta" del proyecto en ese momento	No	No	No
Clonar Una copia completa + historial + configuración	Sí	Sí (con git pull)	Sí (si tienes permisos y haces git push)

Para clonar un repositorio tan solo nos hace falta saber su dirección exacta en GitHub. Unas veces se nos proporcionará ese dato y, en otras ocasiones, deberemos buscar la dirección desde el mismo GitHub.

- **Veamos un caso práctico:** Vamos a clonar el repositorio que acabamos de crear para nuestros apuntes del módulo. Para obtener su dirección, pulsamos el botón "*Code*": Cuando aparezca la ventana emergente, haz click en el icono del portapapeles:



De esta manera, la dirección del repositorio se copia a la memoria.

Para pegarla en una ventana de terminal se utiliza el botón derecho del ratón y la opción *Pegar* (o *Paste*).

Vete a tu terminal de línea de comandos. Nos situamos dentro de la carpeta donde queremos clonar el repositorio, por ejemplo en la carpeta *C:\IES_BQ\Entornos de Desarrollo\Git* (en tu caso, pon la carpeta que consideres oportuna):

```
$ pwd
/c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git
$
```

Para clonar el repositorio Github, tecleamos *git clone* y pegamos la dirección del repositorio que tienes en el portapapeles. Aparecerán los siguientes mensajes, lo que indica que el proceso ha terminado satisfactoriamente.

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git
$ git clone https://github.com/Luis-Ja/ENDES_Apuntes.git
Cloning into 'ENDES_Apuntes'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), 352.17 KiB | 1.64 MiB/s, done.
```

Si listamos ahora el contenido de nuestra carpeta (en nuestro ejemplo, el directorio "*C:\IES_BQ\Entornos de Desarrollo\Git*") vemos que aparece una nueva carpeta (subdirectorio ENDES_Apuntes) con el repositorio que acabamos de clonar:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git
$ ls
ComandoBasicosDirectorios.txt  ENDES_Apuntes/  Ejemplos-java-sencillos/  Ejercicios_Java/
```

Chequeamos su contenido y vemos que es correcto:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git
$ cd ENDES_Apuntes/

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$ ls
'Error por borrar a mano el fichoro bak de GitHub.docx'  'Intro_Archivos y Directorios.pdf'  README.md

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$
```

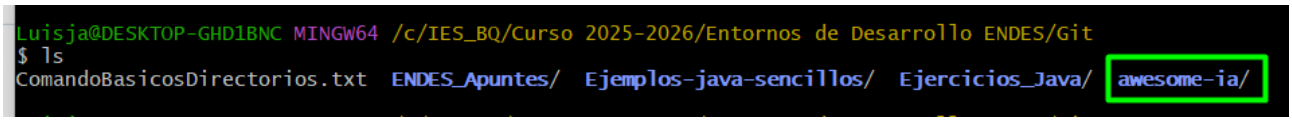
ACTIVIDAD 1

Busca el repositorio del proyecto donde se almacena el proyecto **awesome-ia (**usuario **IAARhub).** Se trata de una lista curada de recursos en Inteligencia Artificial en español (bibliografía, herramientas, tutoriales).

Verás que se trata de un proyecto muy ligero, pero muy útil ya que cuenta con muchísimos enlaces a recursos fundamentados en IA (la lista la encontrarás en el archivo README.MD).

Enlace en GitHub: https://github.com/IAARhub/awesome-ia

Deberás realizar el clonado a un directorio de tu PC (o disco duro externo): Comprueba que el clonado se ha realizado correctamente, es decir, que en el directorio en el que hayas realizado el clonado, se ha creado automáticamente una carpeta que contiene el clon del repositorio:



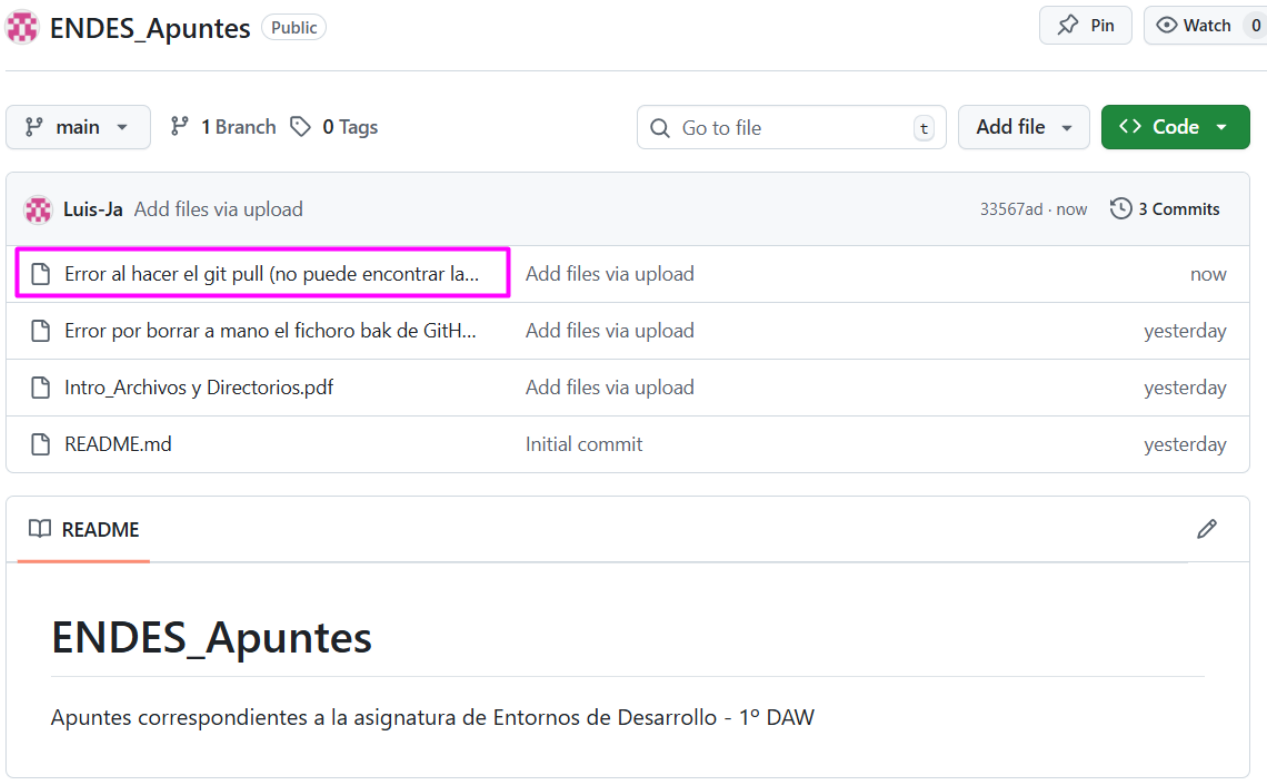
5- Actualización local de repositorios de GitHub

Vimos en el apartado anterior cómo clonar repositorios.

Como práctica, hicimos una copia exacta en nuestro ordenador del repositorio "ENDES_Apuntes" que estaba en Github y que habíamos creado nosotros mismos, y también del repositorio "**awesome-ia**" creado por el usuario **IAARhub**, a quien no conocemos pero que comparte su trabajo a través de GitHub.

Lo habitual es que el contenido de un repositorio vaya cambiando con el tiempo: se arreglan errores, se amplía la funcionalidad con nuevas características, se cambia el aspecto de la aplicación, etc. Por ejemplo, ¿No sería raro que se añadiese nueva documentación en el repositorio de ENDES_Apuntes o que se actualizase la lista de recursos IA del repositorio awesome-ia, verdad?

Por ejemplo: *Alguien* (el creador del repositorio o cualquier colaborador) ha añadido un nuevo recurso, el fichero "*Error al hacer el git pull (no puede encontrar la ruta especificada).docx*".



Sin embargo, en nuestro Git Local (el directorio donde hemos clonado el repositorio en local) este fichero todavía no ha sido descargado. Para ello, **vamos a actualizar directorio** local de una forma muy sencilla y segura, simplemente utilizando el comando ***git pull***.

Es muy importante situarse justo dentro del directorio del proyecto. Comprobamos que efectivamente el nuevo fichero todavía no existe en mi Git local.

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$ pwd
/c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$ ls
'Error por borrar a mano el fichoro bak de GitHub.docx'  'Intro_Archivos y Directorios.pdf'   README.md
```

Procedemos con la actualización -> ***git pull***

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 11.66 KiB | 568.00 KiB/s, done.
From https://github.com/Luis-Ja/ENDES_Apuntes
   ad4d704..33567ad  main       -> origin/main
Updating ad4d704..33567ad
Fast-forward
 ... pull (no puede encontrar la ruta especificada).docx | Bin 0 -> 13531 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Error al hacer el git pull (no puede encontrar la ruta especificada).docx

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/ENDES_Apuntes (main)
$ ls
'Error al hacer el git pull (no puede encontrar la ruta especificada).docx'  'Intro_Archivos y Directorios.pdf'
 'Error por borrar a mano el fichoro bak de GitHub.docx'                      README.md
```

Como podemos observar, se ha añadido un nuevo fichero llamado "*Error al hacer el git pull (no puede encontrar la ruta especificada).docx*"

Ahora los ficheros y carpetas que hay en el repositorio que está en nuestro ordenador son exactamente iguales que los que hay en el repositorio de GitHub. **Recuerda que la actualización hay que realizarla siempre con *git pull* de forma manual** (los archivos no se sincronizan automáticamente al estilo de otras aplicaciones como Dropbox o Google Drive).

Hacer ***git pull*** es como decirle a Git: "descárgame todas las novedades que otros han subido al repositorio remoto y actualiza mi copia local".

ACTIVIDAD 2

Tal y como acabas de ver en los apuntes, añade algún fichero a tu repositorio de Github.

A continuación, con ***git pull***, sincroniza tu copia local y comprueba que el/los nuevos ficheros aparecen.

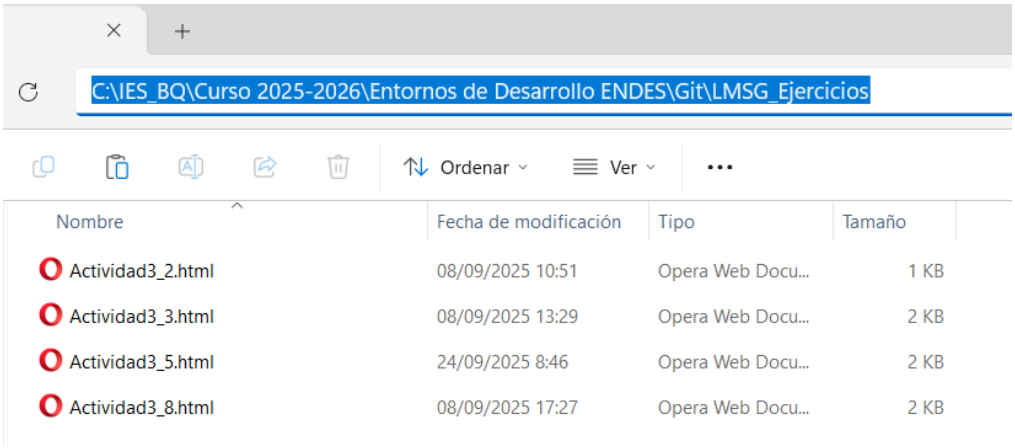
6- Flujo de Trabajo con Git

Si has realizado las actividades anteriores, deberías tener creado el repositorio GitHub "*ENDES_Apuntes*". Puedes mantener este repositorio sin ningún problema (o borrarlo).

No obstante, tengo que advertirte que, aunque en el apartado anterior hemos utilizado GitHub como punto de partida para clonar repositorios y explorar contenido a nuestro directorio local, **el flujo de trabajo real en proyectos consiste en trabajar localmente con Git y sincronizar los cambios con GitHub**. GitHub actúa como repositorio remoto para compartir, colaborar y mantener una copia en la nube.

Por lo tanto, a partir de ahora y como normal general, **no subas más archivos directamente al repositorio de GitHub. A partir de ahora, trabajaremos en nuestro equipo de manera local.** Es decir, el flujo de trabajo habitual, profesional y recomendado es: ***trabajo local → sincronización con GitHub***.

Estamos trabajando en local y el producto de nuestro trabajo está en un directorio. Por ejemplo:



Cuando consideramos oportuno subirlo a GitHub ("publicarlo"), seguiremos los siguientes pasos:

1.- Desde línea de comandos, nos posicionamos en la carpeta de nuestro proyecto:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git
$ cd LMSG_Ejercicios/

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios
$ pwd
/c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios
$ ls
Actividad3_2.html  Actividad3_3.html  Actividad3_5.html  Actividad3_8.html
```

2.- Inicializamos Git, comando *git init*

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
hint:
hint: Disable this message with "git config set advice.defaultBranchName false"
Initialized empty Git repository in C:/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios/.git/

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (master)
$
```

Por el mensaje, vemos que ha llamado a nuestra rama principal "master".

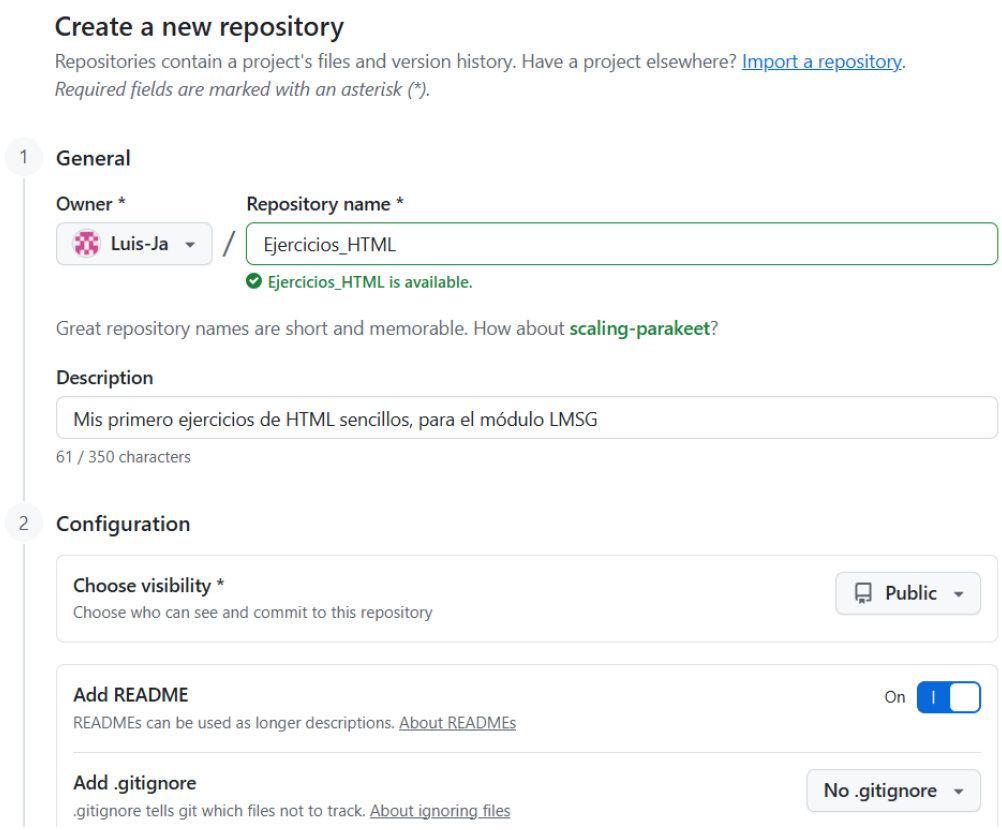
Una rama principal (se llame *master* o *main*) es simplemente la versión principal y estable del proyecto. En un flujo colaborativo con varias ramas, ramas secundarias, pruebas y desarrollos experimentales, hace falta una rama que represente la versión "estable" o "definitiva" del proyecto.

En la actualidad, "*master/slave*" (maestro/esclavo) se considera una terminología obsoleta o poco inclusiva y, por ello, muchas organizaciones tecnológicas (GitHub, Microsoft, Google…) están migrando a términos más neutros como "*main*". De hecho, en GitHub, por defecto, la rama principal del repositorio es "*main*". Por este motivo, y para evitar posibles conflictos, es aconsejable cambiar el nombre de la rama principal de nuestro repositorio de *master* → *main*, utilizando el comando *git branch -m main*

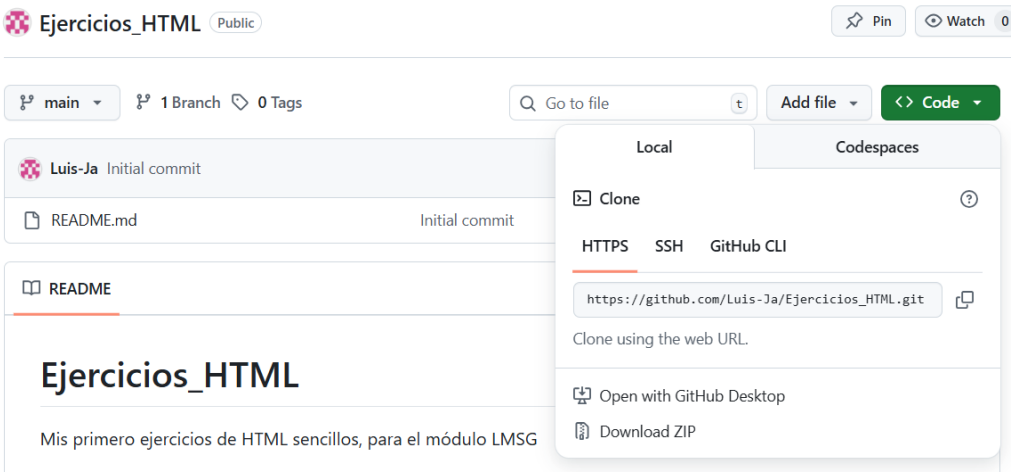
```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (master)
$ git branch -m main

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$
```

3.- Ya podemos clonar nuestro directorio local al repositorio GitHub, pero ¿Tenemos creado ya el repositorio en GitHub? Si todavía no lo hemos creado, como es mi caso, lo crearíamos tal y como lo hemos hecho en el apartado "*3.-Creación de repositorios en GitHub*":

Por ejemplo:



Una vez que lo tenemos creado, copiamos su URL:



4.- Ya podemos conectar nuestro directorio local con el repositorio remoto GitHub:

```
git remote add origin https://github.com/Luis-Ja/Ejercicios_HTML.git
```

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git remote add origin https://github.com/Luis-Ja/Ejercicios_HTML.git
```

Realizamos la comprobación y vemos que efectivamente nuestro directorio local está sincronizado con el repositorio remoto de GitHub:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git remote -v
origin  https://github.com/Luis-Ja/Ejercicios_HTML.git (fetch)
origin  https://github.com/Luis-Ja/Ejercicios_HTML.git (push)
```

Nos lo muestra dos veces: una para indicarnos que podemos actualizar desde el repositorio **remoto hacia local** (comando `fetch`) y la otra para indicarnos que podemos actualizar desde el **directorio local hacia el repositorio remoto** (comando `push`). Lo lógico es realizar la segunda opción: *local → remoto*.

5.- Ahora ya podemos subir nuestros ficheros desde el directorio local de Git al repositorio de GitHub. Vamos a consultar el estado de nuestro directorio Git, para comprobar si tenemos ficheros listos para ser subidos (comando `git status`):

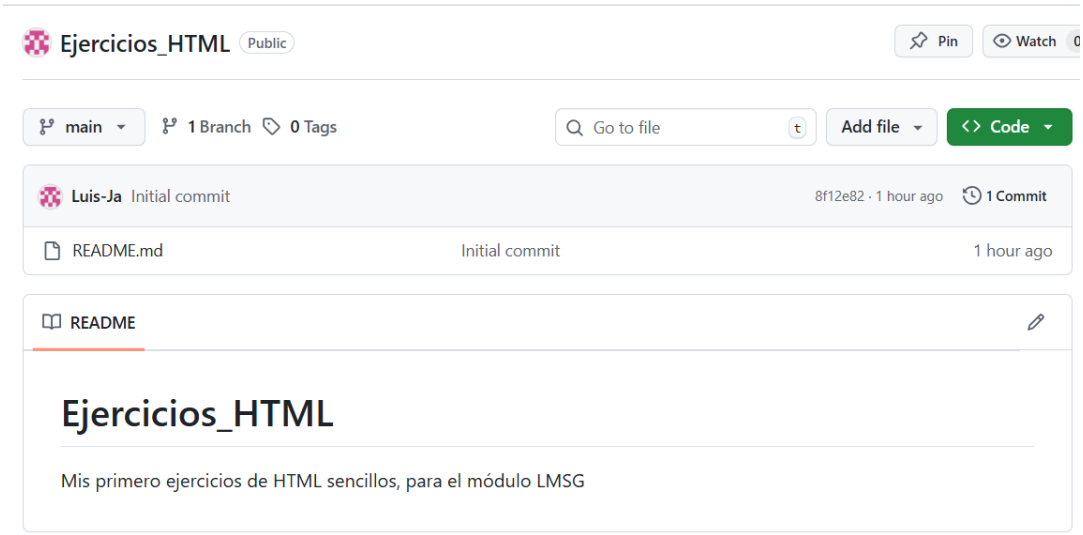
```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Actividad3_2.html
        Actividad3_3.html
        Actividad3_5.html
        Actividad3_8.html

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

Git me dice que no se han realizado "commits" todavía.

El mensaje *Untracked files* me informa de que el directorio contiene unos archivos que todavía no están listos para ser subidos (no se han añadido a Git con `git add`).

6.1. Ciclo completo de actualización de un repositorio (add, commit, push)

Como hemos visto en el paso anterior (comando `git status`), tenemos una serie de ficheros en nuestro directorio Git local que queremos subir ("*publicar*") al repositorio GitHub, en el cual, de momento, solo tenemos el fichero README.md:



Comando Add

Si nos fijamos, *git status* nos está dando una pista muy importante sobre el siguiente paso: utilizar el comando ***git add*** seguido del nombre del archivo. Por lo tanto, para que un fichero sea añadido al índice de archivos a tener en cuenta en la próxima subida al repositorio de GitHub (lo que se conoce como "añadir el fichero a Git"), utilizamos el comando *git add Actividad3_2.html*

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo E
$ git add Actividad3_2.html
```

Este comando, si todo va bien, no devuelve feedback. Para comprobar que el fichero se ha añadido correctamente a Git utilizamos de nuevo el comando `git status`:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Actividad3_2.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Actividad3_3.html
        Actividad3_5.html
        Actividad3_8.html
```

Vemos que el fichero añadido aparece ya como preparado para ser incluido en el próximo *commit* (en el próximo apartado veremos que significa esto).

Ahora bien, lo normal es añadir, modificar y borrar con frecuencia muchos ficheros de un repositorio (como en nuestro ejemplo); hacer *git add* para cada uno de ellos puede resultar tedioso y, lo peor, se nos puede olvidar alguno. En la práctica, lo más cómodo es utilizar ***git add . --all*** (fíjate que hay un punto detrás de *add*), de esta manera **Git chequea todos los archivos que se han añadido al directorio y todas las modificaciones que se han realizado**. En nuestro ejemplo:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git add . --all

Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Actividad3_2.html
        new file:   Actividad3_3.html
        new file:   Actividad3_5.html
        new file:   Actividad3_8.html
```

Comando Commit

Cuando usamos `git add` le decimos a Git: "Estos archivos están listos para guardarse", pero es necesario establecer un punto a partir del cual los archivos se subirán tal cual están en ese momento y, para ello, utilizamos el comando ***commit***.

Cuando hacemos un `git commit`, lo que hacemos es tomar una "fotografía" del estado actual de esos archivos, guardarla en la historia del repositorio (como una versión o punto de control) y añadirle un mensaje que explique qué se hizo en esa versión del proyecto.

Es decir, un *commit* guarda oficialmente una versión del proyecto en la línea de tiempo de Git.

Veamos de nuevo cuál es ahora el estado del directorio local Git:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Actividad3_2.html
        new file:   Actividad3_3.html
        new file:   Actividad3_5.html
        new file:   Actividad3_8.html


Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de
```

Nos está diciendo que estos archivos están listos para el próximo commit (ya están en el "staging area").

Llevemos a cabo nuestro primer *commit* (fíjate que debemos escribir tras la opción **-m** un mensaje explicativo indicando en qué consisten los cambios realizados):

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git commit -m "Primera versión del proyecto: subimos nuestras primeras paginas .html"
[main (root-commit) 5901c27] Primera versión del proyecto: subimos nuestras primeras paginas .html
 4 files changed, 149 insertions(+)
 create mode 100644 Actividad3_2.html
 create mode 100644 Actividad3_3.html
 create mode 100644 Actividad3_5.html
 create mode 100644 Actividad3_8.html
```

Este mensaje nos está informando de que el *commit* se ha ejecutado correctamente y, además, nos informa de:

[main (root-commit) 5901c27] → Estás en la rama *main*, y este es el commit raíz (*root-commit*, el primero del proyecto). El código 5901c27 es el identificador único (hash abreviado) de este commit.

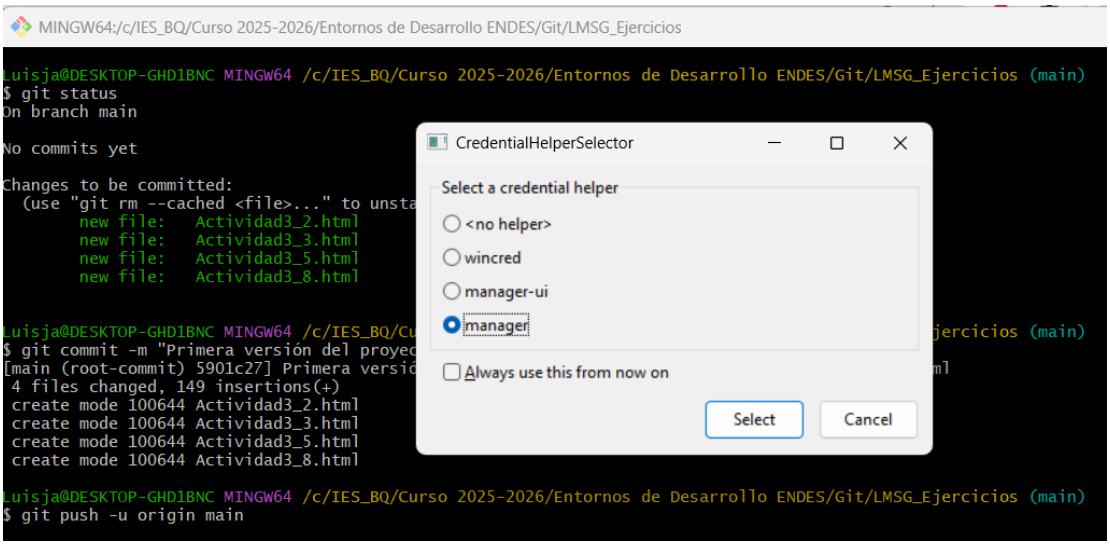
4 files changed, 149 insertions(+) → Se han registrado cambios en 4 archivos, y se han añadido 149 líneas de contenido.

create mode 100644 Actividad3_X.html → Cada archivo ha sido añadido y guardado oficialmente en el historial del proyecto. 100644 indica que son archivos normales de lectura/escritura.

Comando Push

Es importante tener claro que, de momento, los ficheros siguen estando solo en nuestro directorio Git local (en nuestro PC): GitHub aún no sabe nada de este *commit* (si vamos a nuestro repositorio GitHub y recargamos la página en el explorador, vemos que todavía no se ha subido ningún archivo nuevo).

El último paso consiste en "empujar" todos los cambios realizados en local incluidos en el *commit* y llevarlos al repositorio que está en GitHub. Para ello se utiliza el comando ***git push –u origin main***:



Lo normal es que nos aparezca una pantalla emergente cuando hacemos nuestro primer ***`git push`*** desde un repositorio local que aún no ha guardado tus credenciales de GitHub: Git te está pidiendo que elijas un "*Credential Helper*", es decir, un método para guardar de forma segura tus credenciales (usuario/token) para futuras conexiones. Esto se hace para que no tengas que iniciar sesión cada vez que hagas *git push* o *git pull* (que ya veremos para que sirve).

La mejor opción es ***`manager`***, que aparece ya marcada por defecto: Esto activa el **Git Credential Manager for Windows**, que es seguro, moderno y está recomendado oficialmente por GitHub y Git.

Pero puede que, después de pulsar el botón "*Select*", nos aparezca el siguiente error:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git push -u origin main
remote: Invalid username or token. Password authentication is not supported for Git operations.
fatal: Authentication failed for 'https://github.com/Luis-Ja/Ejercicios_HTML.git/'
```

Probablemente, el motivo es que en nuestra configuración de Git no tenemos el gestor de credenciales ("*Credential Helper*") con la opción "*manager*" (en mi caso tenía puesta "*wincred*", por lo que se produce una colisión):

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.interactive=true
color.ui=auto
help.format=html
diff.astextplain.textconv=astextplain
rebase.autosquash=true
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
credential.helper=!"C:/IES_BQ/Curso 2025-2026/Entornos de Desarro
user.name=Luis Javier Menendez
user.email=luisjmt@educastur.org
credential.helper=wincred
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
```

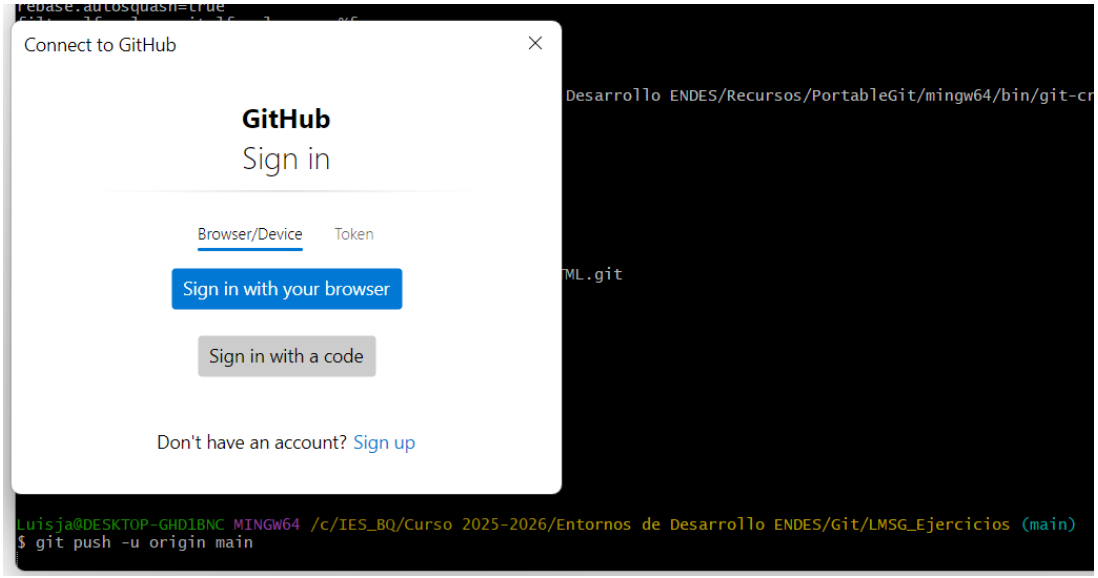
Para actualizar esta configuración, utilizamos el siguiente comando:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git config --global credential.helper manager
```

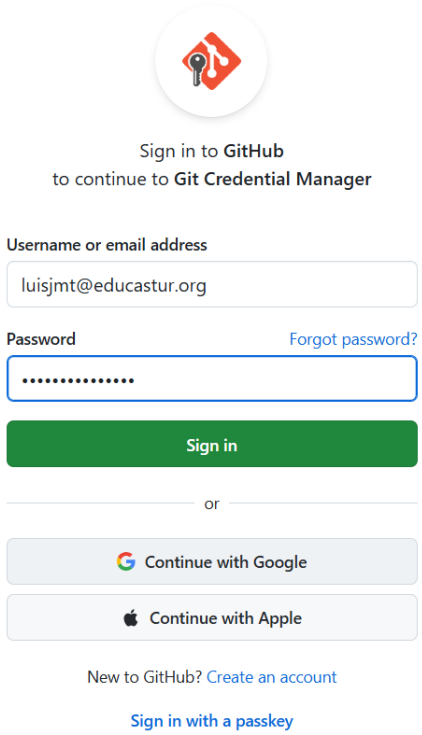
Y comprobamos que ahora si tenemos "manager":

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo E
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.interactive=true
color.ui=auto
help.format=html
diff.astextplain.textconv=astextplain
rebase.autosquash=true
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
credential.helper=!"C:/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Recurs
user.name=Luis Javier Menendez
user.email=luisjmt@educastur.org
credential.helper=manager
core.repositoryformatversion=0
core.filemode=false
```

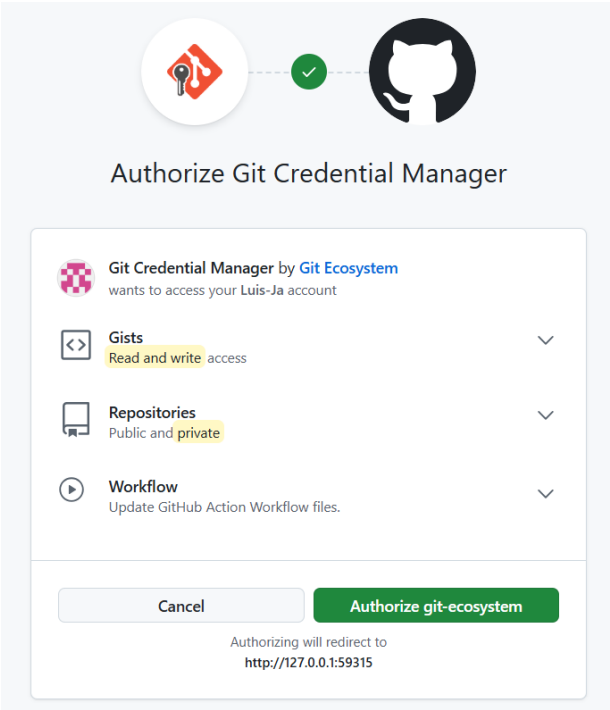
Volvemos a ejecutar el comando ***git push –u origin main***, y ahora ya nos sale una nueva pantalla emergente en la que escoger el método de autenticación en GitHub:



Lo recomendado es escoger la opción por defecto **"Sign in with your browser":** Nos abrirá el navegador para iniciar sesión con tu usuario de GitHub y Git, que quedará autorizado automáticamente.



Si nos autenticamos correctamente, lo normal es que nos aparezca una pantalla como esta:



Debemos pulsar en **"Authorize git-ecosystem"**, ya que sin esta autorización Git no podrá subir tus cambios a GitHub desde la terminal.

Si todo va bien, nuestro repositorio GitHub se habrá actualizado con los ficheros que hemos subido desde Git.

Sin embargo, es bastante común obtener un error como este:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git push -u origin main
To https://github.com/Luis-Ja/Ejercicios_HTML.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Luis-Ja/Ejercicios_HTML.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Este error es debido a que existen discrepancias entre nuestra rama local de Git y la remota GitHub, ya que tienen historiales distintos: en GitHub hay al menos un commit no llevado al Git local, en nuestro caso concreto, el README creado allí no existe en nuestro Git local. Por eso Git rechaza el `push` con el mensaje "fetch first / not fast-forward".

Esto es importante: Git no permite empujar (comando *push*) cambios locales al remoto si el remoto tiene *commits* que tú no tienes en local, porque se perdería ese trabajo remoto. Puedes tener ficheros en GitHub que aún no están en tu Git local, pero entonces debes traerlos primero (comando *pull*) antes de poder subir tus cambios (push). Esta situación es habitual en proyectos colaborativos donde son varios los usuarios que pueden actualizar el repositorio remoto de GitHub.

Por lo tanto, si nos ocurre este error (repito, muy común en proyectos colaborativos), primero sincronizamos de remoto a local `git pull --rebase origin main`

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git pull --rebase origin main
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 930 bytes | 34.00 KiB/s, done.
From https://github.com/Luis-Ja/Ejercicios_HTML
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main
Successfully rebased and updated refs/heads/main.
```

Comprobamos el contenido de nuestro directorio local y vemos que ahora aparece el fichero README.md que antes existía solo en remoto:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ ls
Actividad3_2.html  Actividad3_3.html  Actividad3_5.html  Actividad3_8.html  README.md
```

Ahora ya podemos subir nuestro *commit* de local a remoto:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ejercicios (main)
$ git push -u origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.61 KiB | 548.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Luis-Ja/Ejercicios_HTML.git
   8f12e82..00917f5  main -> main
branch 'main' set up to track 'origin/main'.
```

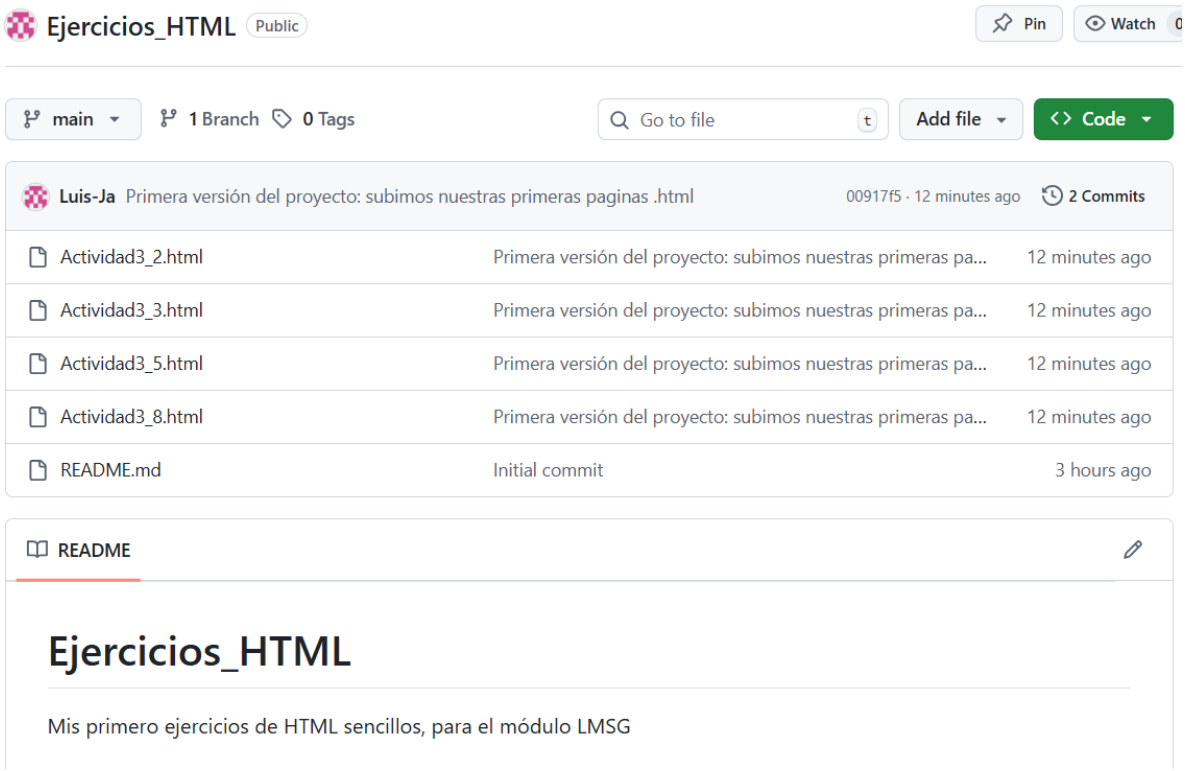
Estos mensajes nos están informando de que el *push* se ha ejecutado correctamente:

"main -> main" → tus commits locales se han enviado a la rama main en GitHub.

Resolving deltas 100% → Git ha empaquetado y optimizado los cambios.

branch 'main' set up to track 'origin/main' → por usar -u, tu rama local queda vinculada al remoto; **desde ahora bastará con git push y git pull a secas.**

Refrescamos el repositorio GitHub en nuestro navegador para comprobar que los ficheros se han subido correctamente desde nuestro directorio Git local:



Observar que el mensaje de status es el que nosotros indicamos cuando hicimos el *git commit*.

Si volvemos a ejecutar *git push*, nos dirá que todo está actualizado:

```
Luisja@DESKTOP-GHD1BNC MINGW64 /c/IES_BQ/Curso 2025-2026/Entornos de Desarrollo ENDES/Git/LMSG_Ej
$ git push
Everything up-to-date
```

ACTIVIDAD 3

Ahora, para consolidar lo que acabamos de ver, vamos a crear o mover nuevos ficheros (al menos 3 ficheros) en nuestro equipo local, en la carpeta correspondiente a nuestro proyecto en Git. Después seguiremos el flujo que acabamos de ver (*add, commit, push*) y comprobaremos que los nuevos ficheros se han subido correctamente al repositorio remoto de GitHub.