

Curso

Introdutorio de GIT 2023

Ingeniería Civil en Informática
Departamento de Ciencias de la Ingeniería

Docente: Victor Saldivia Vera
Email: victor.saldivia@ulagos.cl



EMPEZAREMOS CONOCIENDO UN POCO DE GIT



git

¿Qué es GIT?

Git actualmente se ha estado convirtiendo en un estándar mundial para el control de versiones. Podemos definirlo como ***“un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo”***. Estos repositorios locales permiten trabajar sin conexión o de forma remota con facilidad. Git es open source creado por Linus Torvalds en el 2005.

Los programadores confirman su trabajo localmente y, a continuación, sincronizan su copia del repositorio con la copia en el servidor. Git también facilita el registro y comparación de diferentes versiones de un archivo. Esto significa que los detalles sobre qué cambió, quién cambió qué, o quién ha iniciado una propuesta, se pueden revisar en cualquier momento.

La popularidad de Git lo convierte en una excelente opción para cualquier desarrollador. Casi todos los entornos de desarrollo tienen compatibilidad con Git y las herramientas de línea de comandos de Git implementadas en cada sistema operativo principal.



Comenzando el uso de GIT Client

Para usar Git en la línea de comandos, primero hay que verificar si está instalado o no en nuestro equipo.

1. Se abre CMD o PowerShell y escribimos **git**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\Victo> git
git : El término 'git' no se reconoce como nombre de un cmdlet, función, archivo de script o programa ejecutable.
Compruebe si escribió correctamente el nombre o, si incluyó una ruta de acceso, compruebe que dicha ruta es correcta e
inténtelo de nuevo.
En línea: 1 Carácter: 1
```



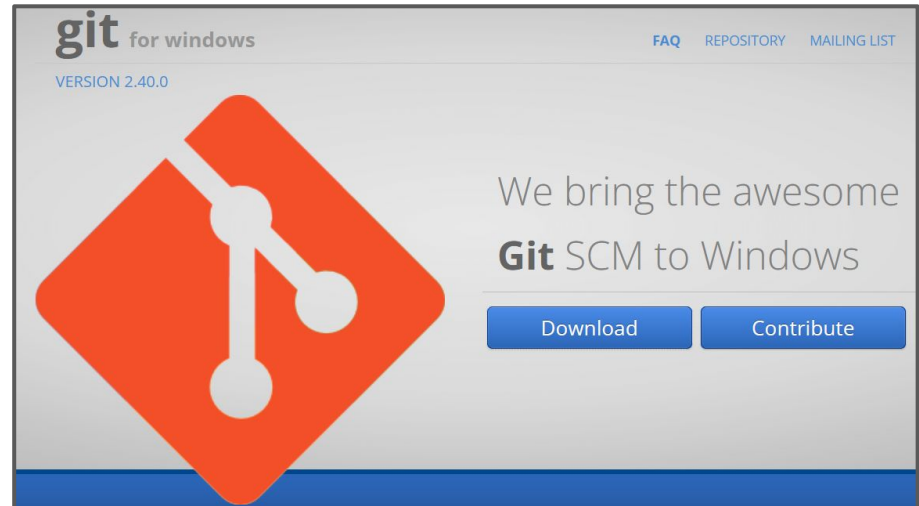
Instalación Cliente GIT Windows

Si Git no está instalado en nuestro equipo nos vamos directamente al sitio oficial de Git:

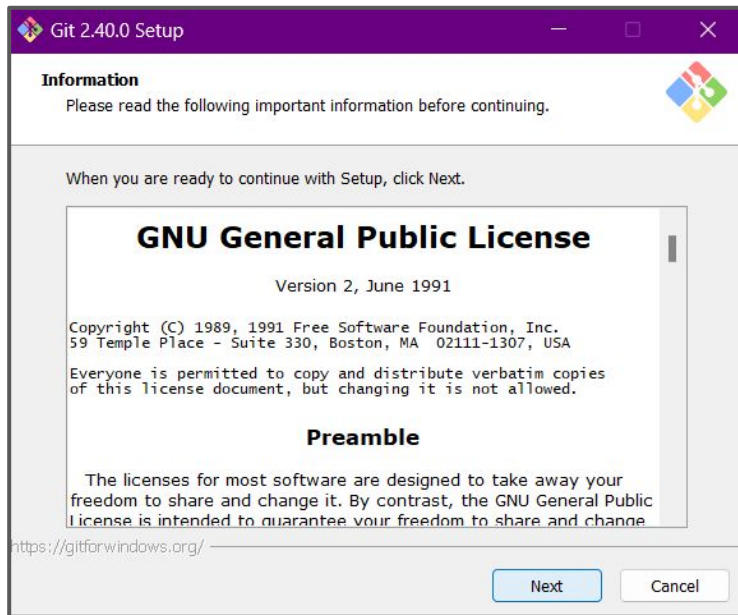
2. **Git para Windows**, donde directamente al entrar en ella desde Windows comenzará la instalación, pues detecta la versión que se tiene instalada en el computador.

Por otro lado también se puede descargar una versión portable. El inconveniente de esta versión, es que no tiene una interfaz gráfica, y se debe recurrir a un emulador de BASH.

Aparte de la instalación de Git el paquete instalador trae **Git BASH**, **Git GUI** y una **integración Shell**.



Instalación Cliente GIT Windows

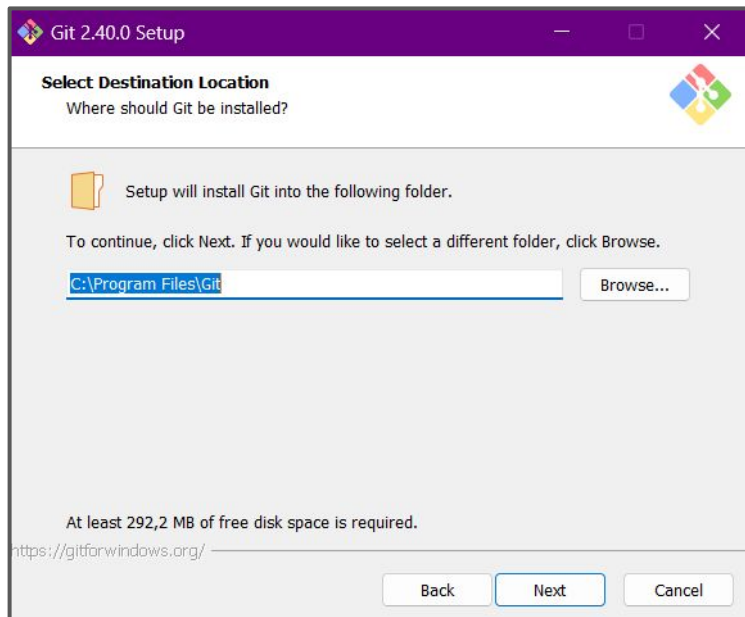


Luego de abrir el ejecutable, aparece información respecto a la licencia GNU que tiene Git

3. Se le da clic en Next



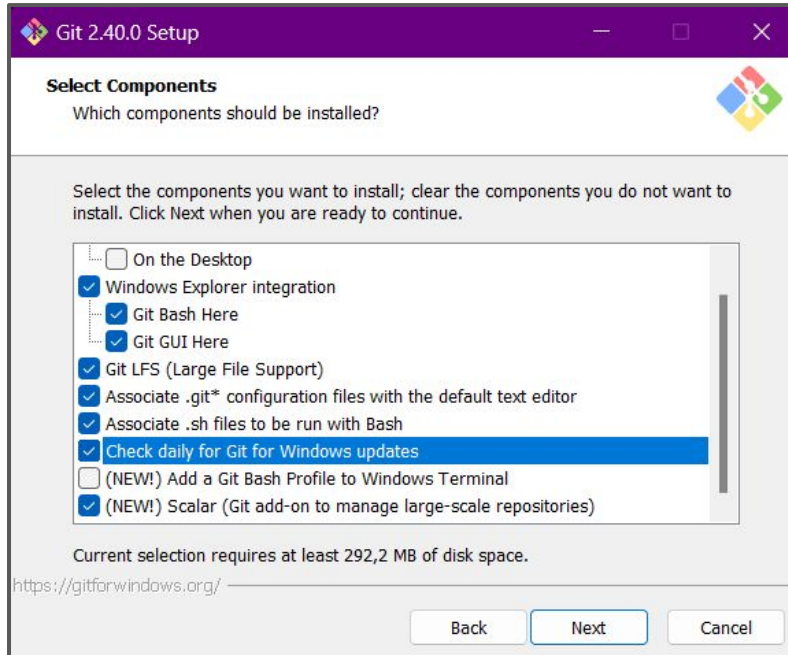
Instalación Cliente GIT Windows



4. Se selecciona la carpeta de instalación del programa



Instalación Cliente GIT Windows



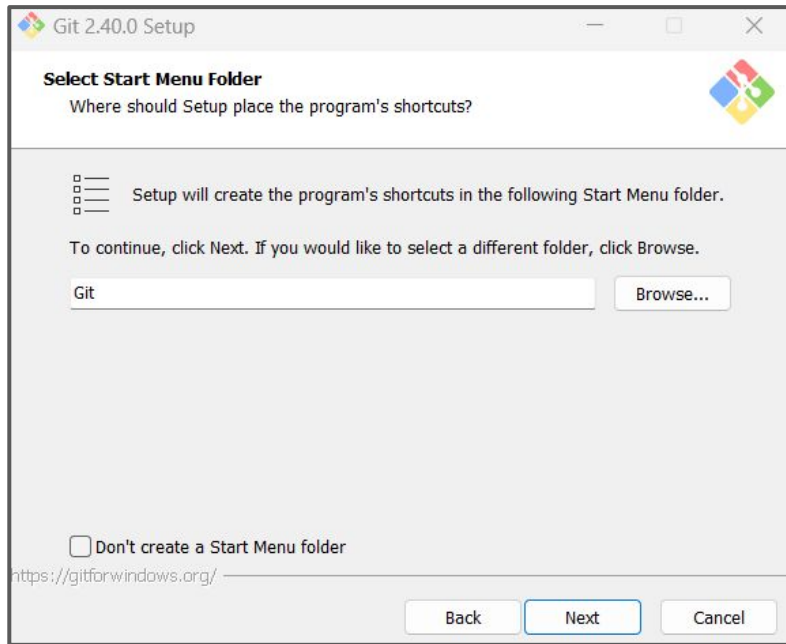
Aquí se selecciona que deseamos instalar, como recomendación seleccionar las siguientes opciones:

5. Marcar las casillas

- **Instalación de GIT Bash y GIT GUI**
- **Soporte de archivos grandes con GIT**
- **Configuración de archivos .git con editores de texto**
- **Configuración con archivos .sh**
- **Verificar las actualizaciones de GIT**
- **Agregar un perfil de BASH a la terminal de Windows**



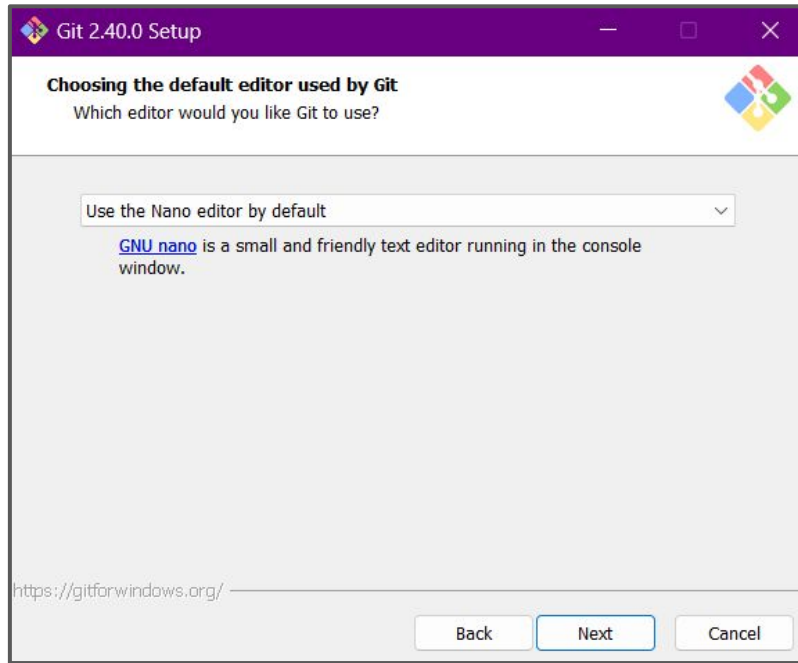
Instalación Cliente GIT Windows



6. Seleccionar el nombre del archivo donde se va instalar, por defecto es GIT.



Instalación Cliente GIT Windows

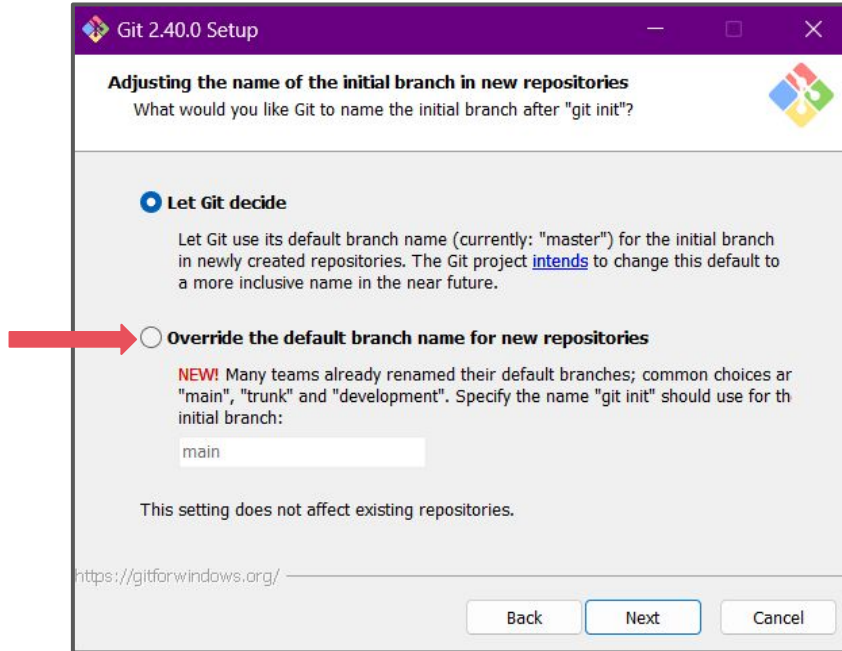


7. Acá se debe elegir qué editor de texto utilizar cuando realicemos comandos con GIT. Un editor de texto fácil y sencillo de usar es **Nano** (Pueden elegir a libre elección el editor de texto)

**** NOTA: Si ya usas VS Code, elige esa opción ****



Instalación Cliente GIT Windows

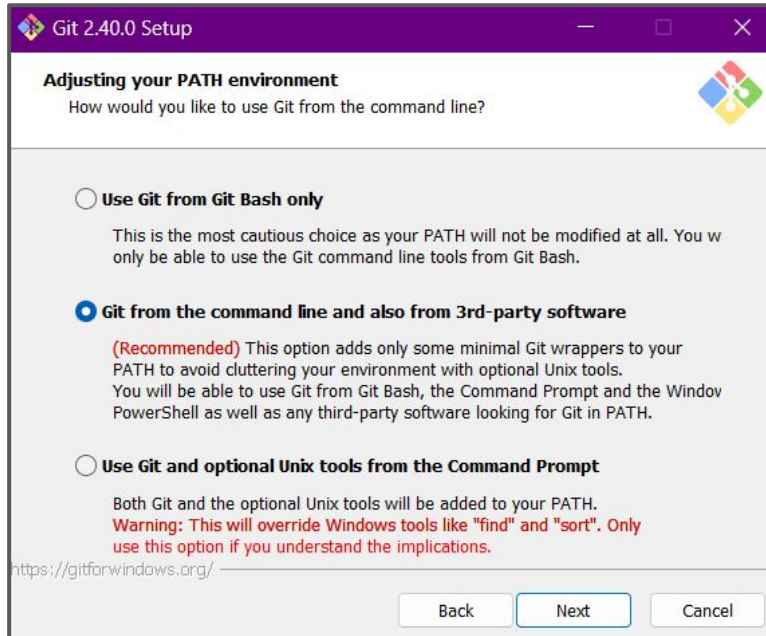


8. En esta parte debemos elegir el nombre de la rama principal por defecto. Cuando se crea un proyecto en GIT, se crea una “rama principal”. Anteriormente era la rama “Master”

Pero actualmente, el nombre por defecto que se utiliza en la rama principal es “Main”. Elegir la segunda opción.



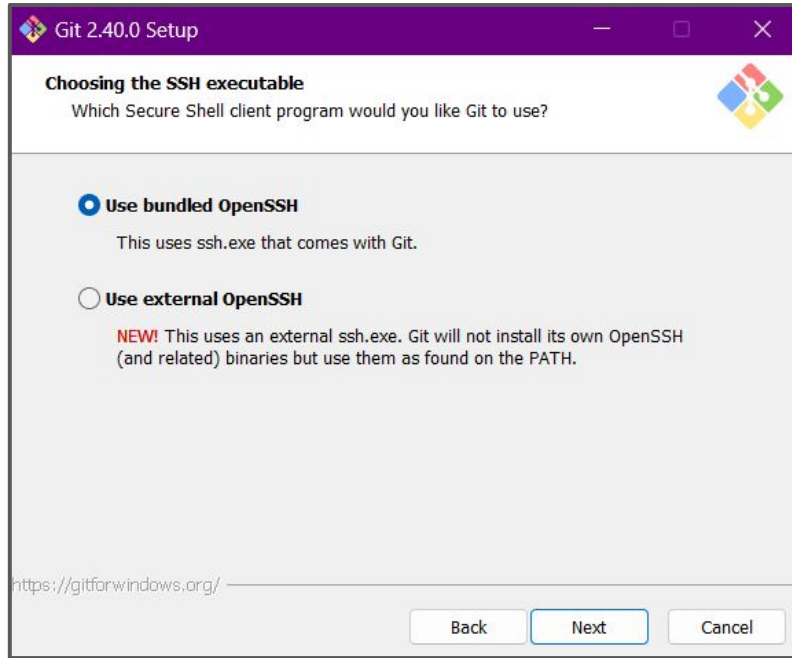
Instalación Cliente GIT Windows



9. En esta sección seleccionamos qué programas podrán utilizar GIT. Hay que elegir la opción más recomendada que es la segunda: **La línea de comandos y otros programas de terceros.**



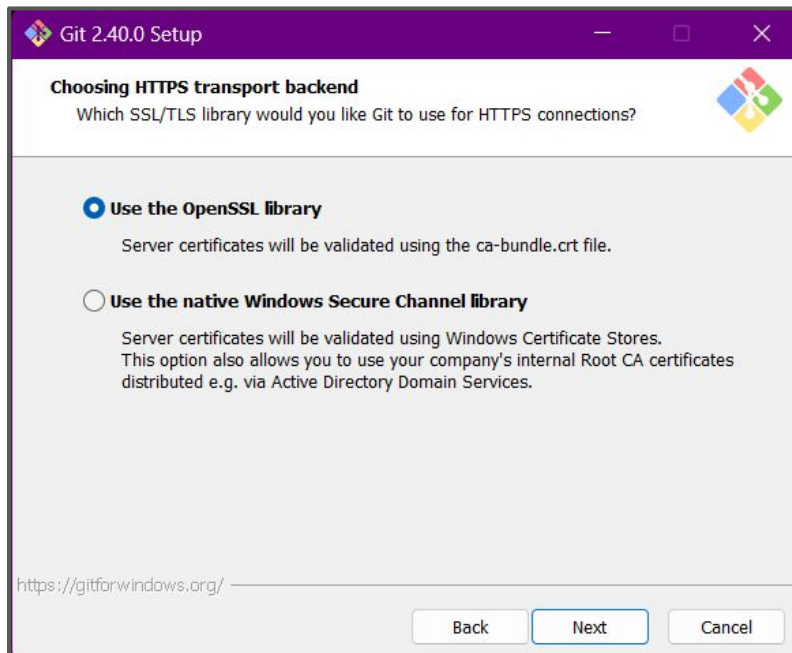
Instalación Cliente GIT Windows



10. En esta parte se selecciona el programa de cliente de shell para usar con GIT. Se recomienda la primera opción.



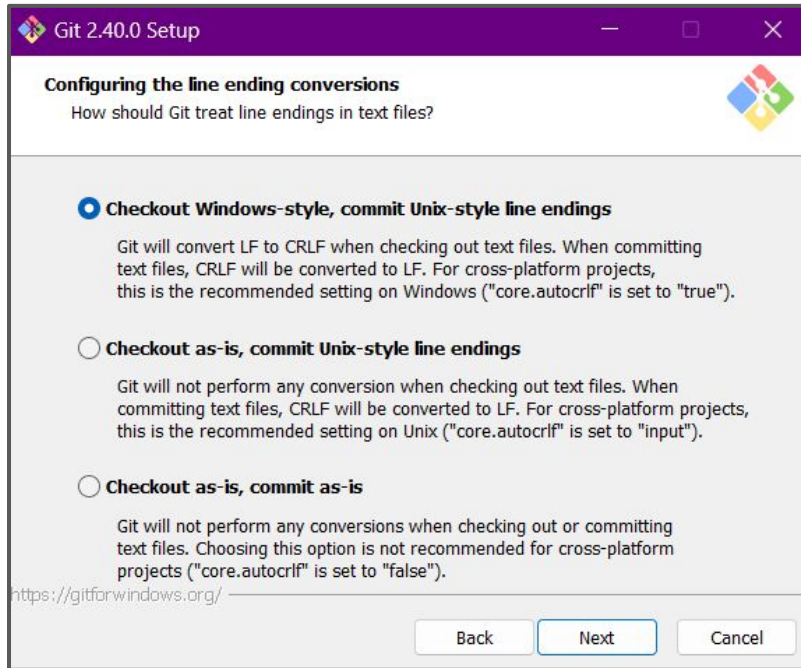
Instalación Cliente GIT Windows



11. Aquí se elige la librería que se quiere utilizar para realizar las comunicaciones con nuestra máquina remota y repositorios externos.



Instalación Cliente GIT Windows

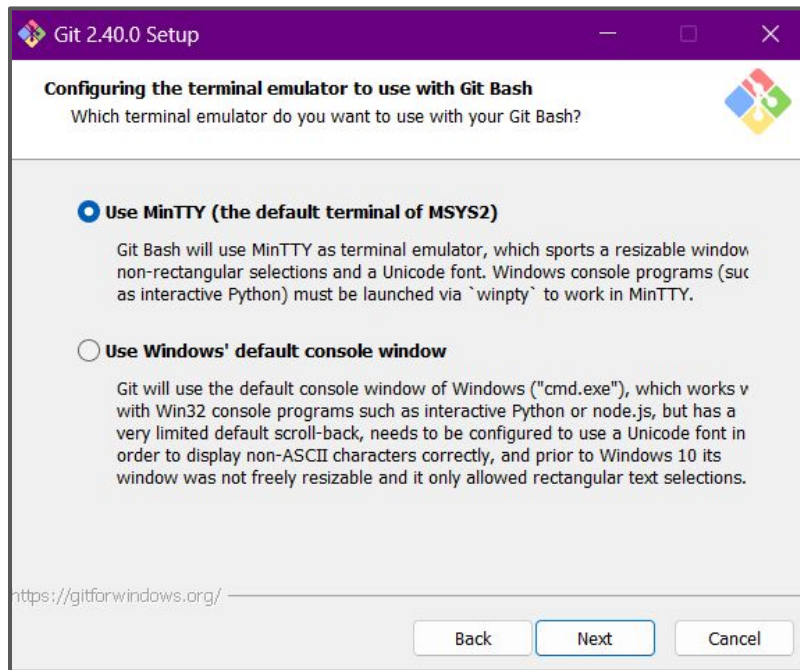


12. Acá nos pregunta cómo queremos manejar los saltos de línea. En este paso es importante analizar que trabajaremos con programadores que utilizarán diferentes sistemas operativos como: **Linux**, **macOS**, y **Windows**.

La mejor opción es: dejar que git realice una conversión desde cada sistema operativo a Unix (estándar).



Instalación Cliente GIT Windows

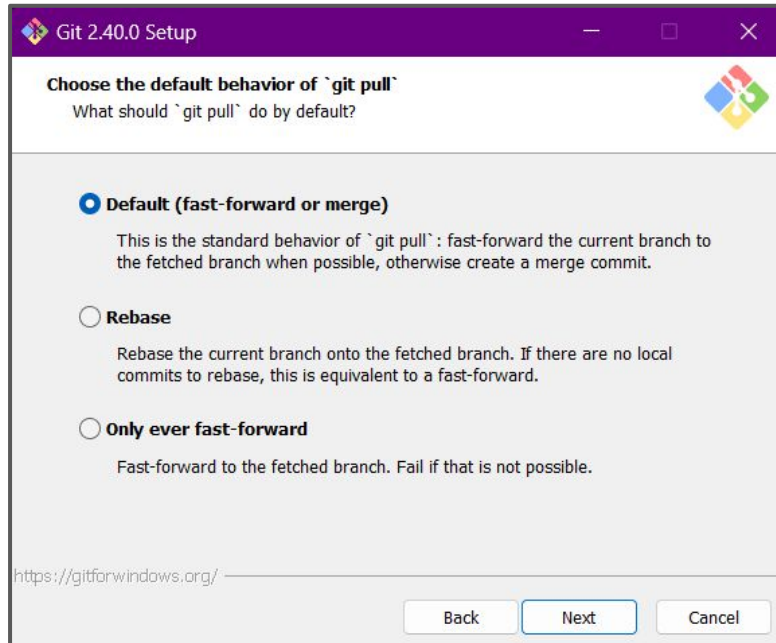


13. En esta sección debemos elegir con cual terminal trabajaremos al usar GIT BASH.

Elegimos la primera opción porque es más estilizada gráficamente que la CMD.



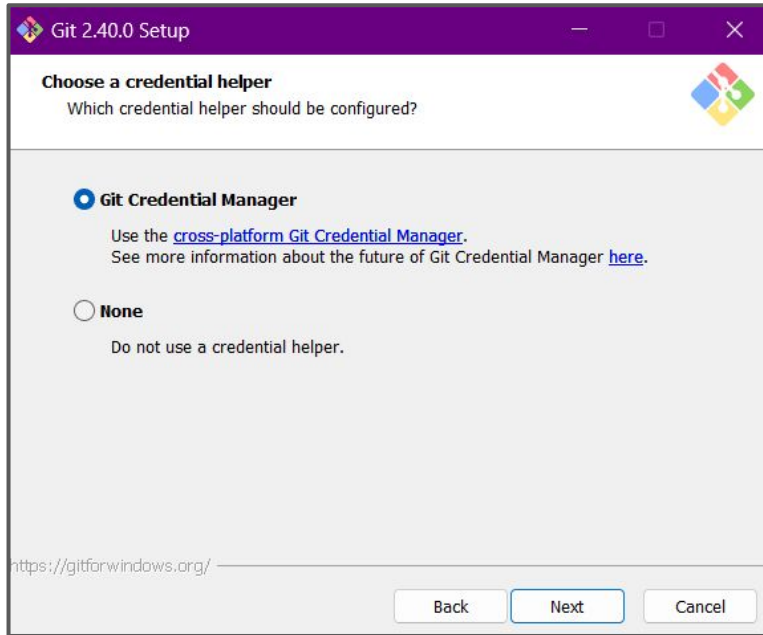
Instalación Cliente GIT Windows



14. En este paso nos pregunta el comportamiento por defecto cuando realicemos un **git pull**. Por defecto vamos a elegir la opción (fast-forward) ya que es lo más común.



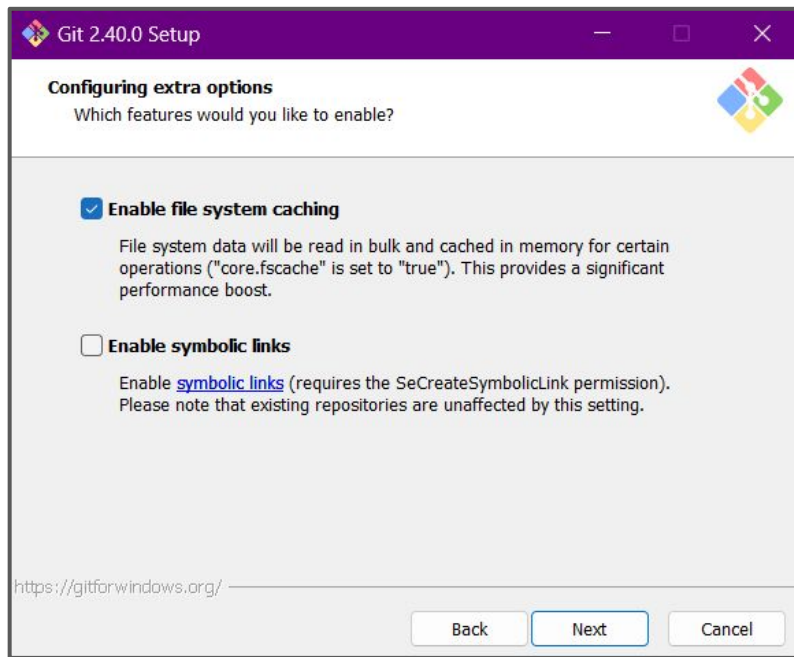
Instalación Cliente GIT Windows



15. Aquí nos pregunta sobre el manejador de credenciales. Esto es para nuestra cuenta de Github. Seleccionamos la primera opción.



Instalación Cliente GIT Windows

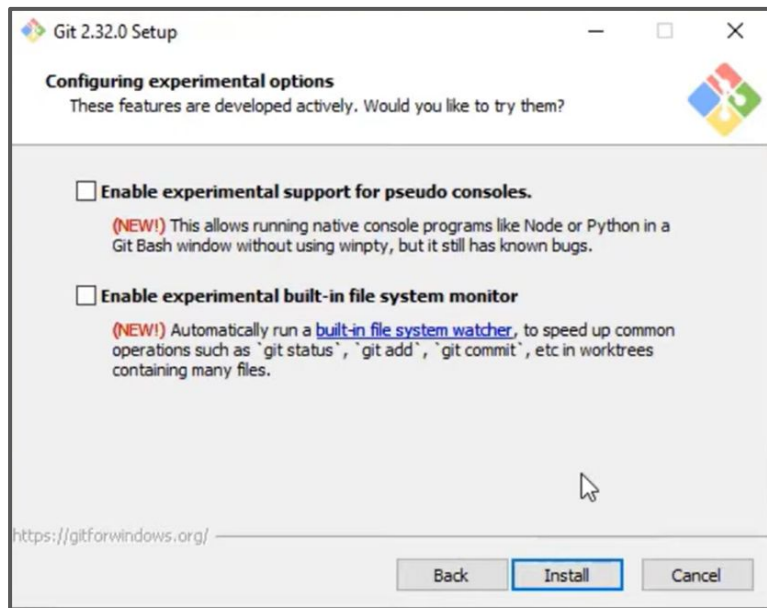


16. Por último hay unas configuraciones extras. En este caso solo vamos a **habilitar el caché**, pero no los links simbólicos (accesos directos en BASH)

La última ventana aparece opciones de herramientas experimentales se recomienda no seleccionar ninguna.



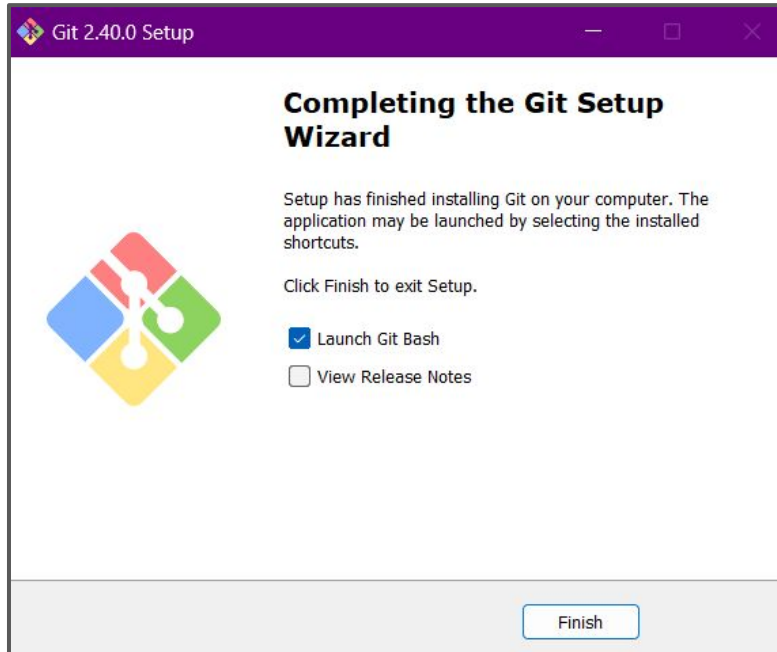
Instalación Cliente GIT Windows



17. La última ventana aparece opciones de herramientas experimentales se recomienda no seleccionar ninguna.



Instalación Cliente GIT Windows



18. Finalizar instalación y ejecutar GIT BASH.



Configurando Usuario y Correo en Git

Algo sumamente importante es la configuración del usuario y correo al usar Git/GitHub. Por ejemplo cada vez que se necesite realizar un **commit** en algún proyecto, además de realizar cambios en la rama, se registra también la información personal. Si se trabaja con un equipo de desarrolladores, con esa información personal se puede saber quien realizó los cambios en el repositorio, y cuál era su correo asociado a ese desarrollador.

Hay dos opciones:

1. Una configuración global para todos los repositorios del computador
2. Una configuración solamente para un repositorio en particular

En este caso se recomienda la configuración global si un usuario ocupará el dispositivo.

Se configurará todo desde la terminal de GIT BASH.



Configurando Usuario y Correo en Git

Primero que nada se empieza configurando el **nombre de usuario**. Este nombre es el que se muestra en los commit cuando se realizan cambios en los repositorios. Se escribe el comando que se muestra abajo en la imagen. Además acotar que:

- El comando **git config** hace referencia que se está configurando parámetros
- **--global**: se está configurando en el plano global

Luego para configurar el email se realiza lo mismo pero ocupando **user.email**

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@example.com
```

Es recomendable no usar un email personal por temas de seguridad, puedes utilizar el correo que te asigna GitHub

Podemos corroborar que tanto el nombre como el email fueron correctamente creados escribiendo las siguientes sentencias en el terminal de Git BASH:

- **git config user.name**
- **git config user.email**



git
bash

Instalación Cliente GIT Linux

Instalar Git en Linux es realmente sencillo, pero la forma de instalarlo dependerá de la distribución. A continuación se muestra cómo instalar en las distribuciones más habituales:

Debian/Ubuntu/Linux Mint y derivados:

```
sudo apt install git
```

Por otro lado, si además quieres una interfaz gráfica para trabajar con Git, existen alternativas como:

- **GitG:** Interfaz gráfica por defecto de GNOME para gestionar repositorios GIT.
- **GitEye:** Herramienta gratuita multiplataforma.

Para mayor detalle de la instalación en este enlace: [How to install Git on Ubuntu](#)



**AHORA
TRABAJAREMOS CON
LA PLATAFORMA GITHUB**



GitHub

Asociando un editor de Texto con GIT

Desde la consola de **GIT BASH** se puede configurar el editor de texto preferido con GIT. De esta manera se puede trabajar los archivos de git con el editor de texto de tu elección.

Si por ejemplo deseas configurar **VS Code** con **GIT**, solo deberás ejecutar el siguiente comando:

```
$ git config --global core.editor "code --wait"
```

Para mayor información de configuración con otros editores, visitar el siguiente enlace:

- [Asociar editores de texto con GIT](#)



¿Qué es GitHub?

GitHub es un sitio **"social coding"**. Esta plataforma permite subir repositorios de código para almacenarlo en el sistema de control de versiones Git. La gran ventaja de GitHub es que se puede colaborar en proyectos de código con otros desarrolladores. Esta plataforma es propiedad de Microsoft, y ofrece a los desarrolladores la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura.

Lo mejor es que facilita la organización de proyectos y permite la colaboración de varios desarrolladores en tiempo real. Es decir, permite centralizar el contenido del repositorio para poder colaborar con los otros miembros del equipo de desarrollo.



Ventajas de usar GitHub

GitHub es una gran opción para el control y gestión de proyectos de código. Algunas de sus grandes ventajas son:

- **GitHub permite que alojar proyectos en repositorios de forma gratuita**
- **Brinda la posibilidad de personalizar el perfil en la plataforma**
- **Los repositorios pueden ser públicos o privados (depende de la elección del desarrollador)**
- **Facilita el compartir proyectos de una forma mucho más fácil y crear un portafolio**
- **Permite colaborar para mejorar los proyectos de otros y viceversa**



GitHub vs GitLab

Característica	GitLab	GitHub
Git	✓	✓
Versión autoalojada	✓	✓ (Con el plan Enterprise)
Integración y entrega continuas	✓	✓ (Con aplicación de terceros)
Documentación basada en Wiki	✓	✓
Vista previa de los cambios de código	✓	✓
Rastreador de problemas	✓	✓



GitHub vs GitLab

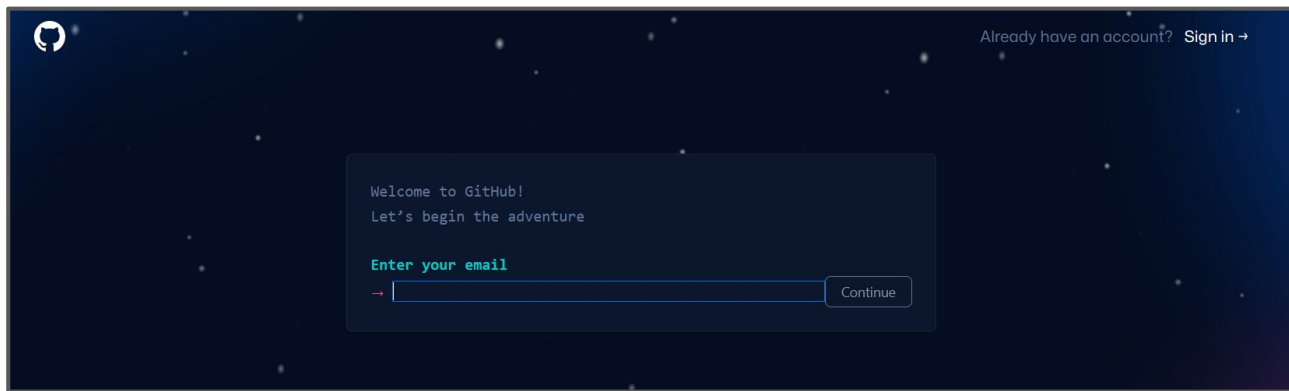
Característica	GitLab	GitHub
Asignación de múltiples propuestas/incidencias (issues)	✓ (Plan de pago)	✓ (Solo con el repositorio público en el plan gratuito)
Tableros de gestión de proyectos	✓	✓
Debates en equipo	✓	✓
Seguimiento del tiempo	✓	✓ (Con App)
Herramientas de seguridad y conformidad	✓	✓
Pruebas de rendimiento de la carga	✓ (Plan de pago)	✓ (Con App)
Pruebas de rendimiento del navegador	✓ (Plan de pago)	✓ (Con App)
Iteraciones y planificación de sprints (incluido el diagrama de descomposición)	✓ (Plan de pago)	✓ (Con App)
Dependencias de propuestas/incidencias (issues)	✓ (Plan de pago)	✓



Creando cuenta en GitHub

Ahora que está configurado el usuario y correo de GIT en nuestro computador tenemos que crear la cuenta de GitHub. Nos dirigimos al sitio oficial de GitHub: <https://github.com/>.

Se crea la cuenta y seguimos las instrucciones paso a paso:



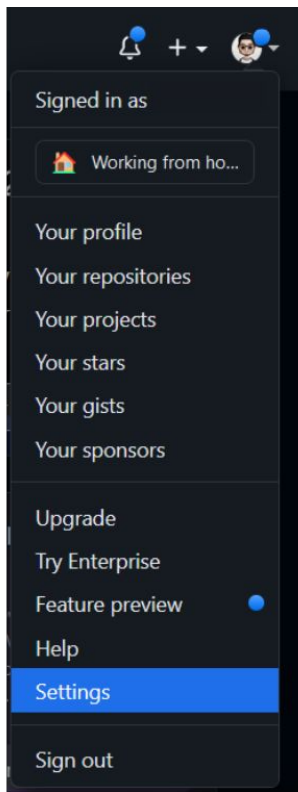
Recuerdas que configuramos el email de GIT?

Bueno, vamos a intentar modificar el email por el que nos proporciona Github

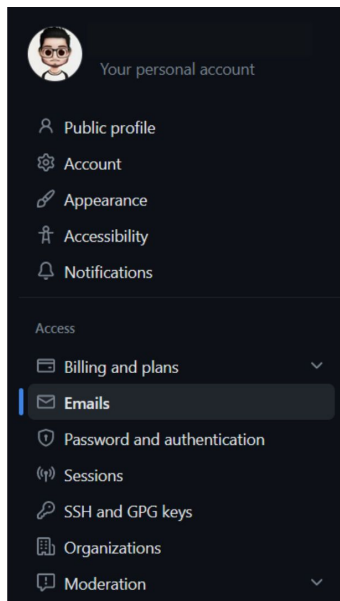
Para esta re-configuración ya tenemos que tener nuestra cuenta GitHub creada



Re-configurando el email de GIT



1. Nos dirigimos a la parte de Settings en el perfil de Usuario

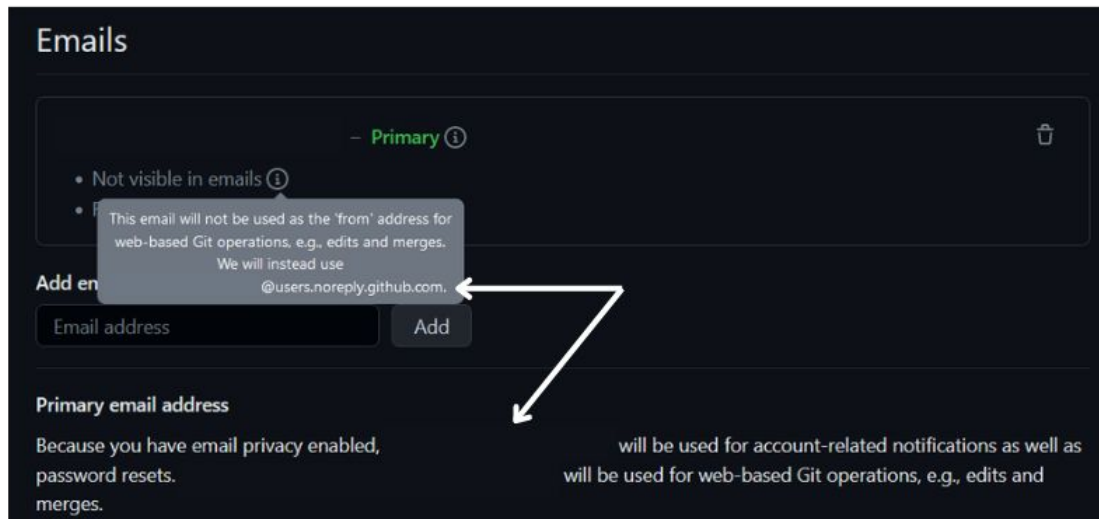


2. Luego en la barra lateral izquierda, en la sección de Correo Electrónico / Email



Re-configurando el email de GIT

3. Se va a desplegar la siguiente interfaz. Al pasar el puntero en el icono de información, se desplegará un tooltip con el correo que entrega GitHub. Este correo igual se puede encontrar en la parte de abajo, como lo indica la flecha de la figura.



Si quiere mantener la dirección de correo electrónico como privada, puede utilizar esta dirección de correo electrónico **noreply** proporcionada por GitHub. Este email lo ocuparemos en el **GIT BASH** (Todo lo mencionado es opcional)



Re-configurando el email de GIT

Ahora que tenemos nuestro correo **noreply** de **GitHub**, abrimos la terminal GIT BASH, y realizamos el mismo comando que la primera vez al configurar nuestro email:

```
git config --global user.email johndoe@example.com
```

Podemos corroborar que el cambio de email se realizó correctamente utilizando la siguiente sentencia:

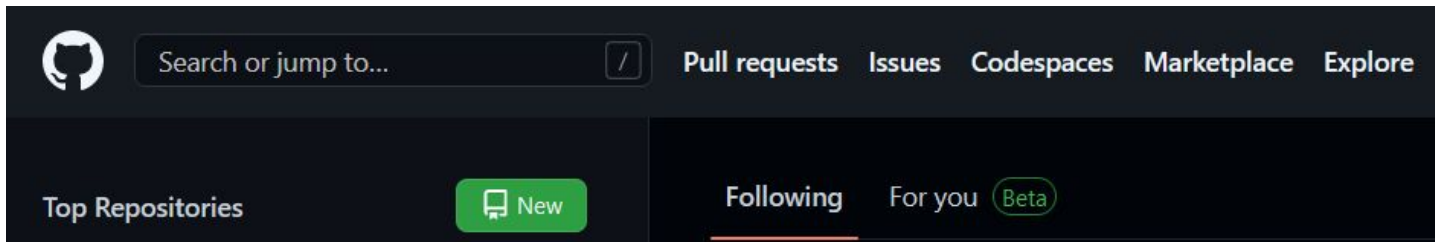
- **git config user.email**



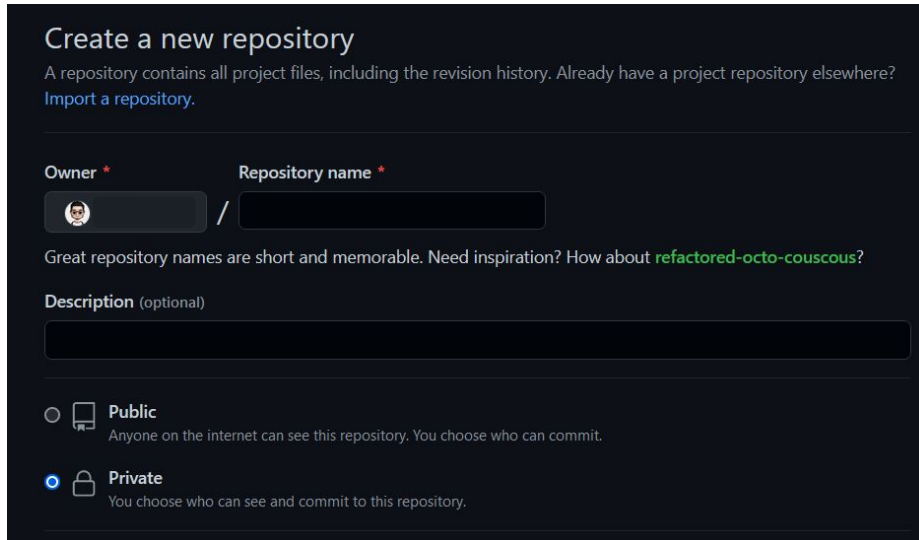
Creando un repositorio en GitHub

Vamos a crear el primer repositorio desde la plataforma de GitHub. Nos ubicamos en la interfaz principal, barra lateral izquierda.

1. Clic en el botón de color verde que dice New:



Creando un repositorio en GitHub



The screenshot shows the GitHub interface for creating a new repository. At the top, it says 'Create a new repository' with a subtitle 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.' Below this, there are two input fields: 'Owner' (with a dropdown menu showing a profile picture) and 'Repository name' (with a text input field). A hint text says 'Great repository names are short and memorable. Need inspiration? How about `refactored-octo-couscous`?'. Below the name fields is a 'Description (optional)' text area. At the bottom, there are two radio button options: 'Public' (with a globe icon) and 'Private' (with a lock icon). The 'Private' option is selected.

Se abre una interfaz en la cual debemos colocar la siguiente información

2. **A) Nombre del repositorio** (un nombre corto y sencillo, normalmente en minúscula y conectadas con guiones)

B) Descripción: **es opcional** (se puede cambiar después de crear el repositorio)

C) Luego se debe escoger si el repositorio será público o privado. Si es público, cualquier persona en internet puede ver el repositorio y hacer una copia de él. Si es privado, puedes elegir quién puede ver el repositorio y quienes hacen cambios en él.



Creando un repositorio en GitHub

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

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

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)
.gitignore template: None ▼

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)
License: None ▼

ⓘ You are creating a private repository in your personal account.

Create repository

Para inicializar el repositorio se puede agregar:

3. A) Archivo README (es un archivo que describe el repositorio para que las personas se informen del contenido del repositorio, siempre es bueno incluirlo)

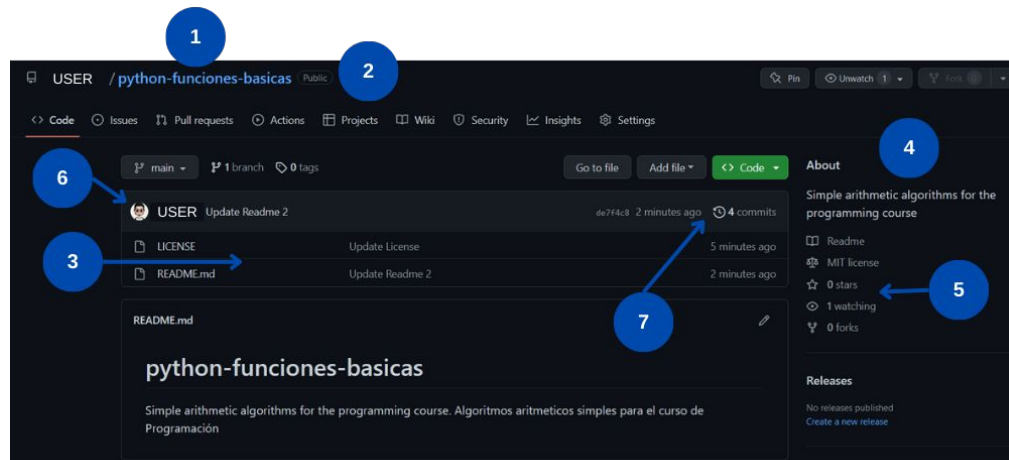
B) Archivo gitignore (es un archivo oculto, permite escoger qué archivos no van a ser rastreados, es decir que no estarán en el repositorio) - opcional

C) Licencia (indica lo que se puede hacer y lo que no se puede hacer con el código del repositorio. Hay muchos tipos de licencias) - Investigar para mayor información

Por defecto la rama principal de **GitHub** ahora es **main**, eso se puede cambiar desde settings, y personalizar la rama. O bien desde el editor de texto elegir el nombre de la rama principal.



Estructura del repositorio en GitHub



Al estar creado el repositorio su estructura es la que se muestra en la imagen de arriba

1. En la parte superior sale el nombre de usuario y el nombre del repositorio
2. Al lado del nombre del repositorio se indica si este es **público** o **privado**
3. Parte principal, salen los archivos (el código) del repositorio creado
4. Descripción del repositorio
5. Información como, personas que están pendiente del repositorio en el momento, favoritos, forks del repositorio.
6. En la parte superior aparece nombre del usuario que hizo el último commit,
7. En la misma parte superior pero en la derecha aparece la cantidad de commits realizados en el repositorio y cuando se realizó la última modificación.

AHORA HAY QUE ENVIAR EL REPOSITORIO LOCAL A GITHUB

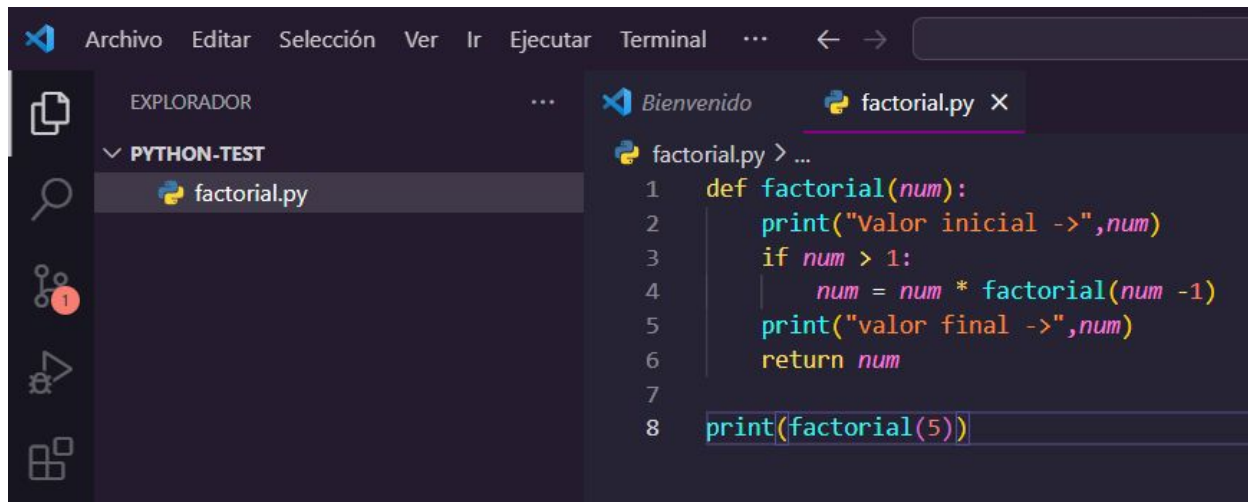


PERO PRIMERO DEBEMOS INICIALIZAR EL REPOSITORIO DE FORMA LOCAL CON GIT



Inicializando el repositorio en GIT (Local)

1. Primero que nada crearemos una carpeta de trabajo en VS Code, (nombre a elección, si hay espacios entre las palabras colocar un guión) y dentro de la carpeta creamos un archivo **.py** que contendrá un algoritmo de recursividad: *el Factorial de un número*, como se muestra en la imagen de abajo



The screenshot shows the Visual Studio Code interface. On the left, the Explorer panel shows a folder named 'PYTHON-TEST' containing a file 'factorial.py'. The main editor area shows the code for 'factorial.py' with the following content:

```
1 def factorial(num):
2     print("Valor inicial ->", num)
3     if num > 1:
4         num = num * factorial(num - 1)
5     print("valor final ->", num)
6     return num
7
8 print(factorial(5))
```



Visual Studio Code

Inicializando el repositorio en GIT (Local)

2. Luego escribiremos el primer comando en la terminal integrada de VS Code que será **git init**, para inicializar el repositorio en nuestro equipo

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git init
Initialized empty Git repository in C:/Users/Victo/OneDrive/Documentos/Programming/PythonPrograms/Python-Test/.git/
```

3. Con el comando **git status**, podemos observar que no se han realizado commits y tampoco se han hecho seguimiento de los archivos:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    factorial.py

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



Agregando Archivos para el Seguimiento

4. Después de inicializado el repositorio hay que añadir el proyecto al **staging área** para que Git haga un seguimiento del proyecto con los cambios que se han realizado. Aquí se escribe el comando **git add**

Para añadir todos los cambios y archivos específicamente **git add .** Con este comando tus archivos han sido añadidos al **staging área** y están siendo seguidos por Git, esperando a que sean confirmados más adelante con un **commit**. Se puede utilizar **git status** para corroborar que se hayan realizado los cambios.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

● PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git add .
● PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git status
○ On branch master

No commits yet

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



Visual Studio Code

Agregando Archivos para el Seguimiento

5. Para guardar una etapa en particular y cambios en tu proyecto de forma permanente en el repositorio de Git debes utilizar el comando **git commit**. Aunque en el futuro realices más cambios, esta confirmación en particular que realizaste ahora se guardará en el repositorio de Git y tendrás acceso a ella en cualquier momento. Así que después de haber añadido tu proyecto al **staging area**, lo siguiente es confirmarlo usando el comando **git commit -m "Mi primer commit"**.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git commit -m "Mi primer commit"
○ [master (root-commit) 3cf372e] Mi primer commit
  1 file changed, 8 insertions(+)
  create mode 100644 factorial.py
```



Cambiando el nombre de la Rama Principal

6. Actualmente en **GitHub** se está ocupando como estándar el nombre **main** para la Rama Principal. Para que exista una sola rama principal tanto en GIT Local como en GitHub tiene que tener el mismo nombre de dicha rama. Por ende, si por ejemplo en el caso de GitHub este por defecto la rama principal como **main** y en la configuración inicial de GIT Local quedó como rama principal **master**, esto se puede cambiar con el comando **git branch -m <nombre de la rama>**

```
PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git branch -m main
PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git branch --list
* main
```

(Otro método que no es por líneas de comando, es ir a la parte de Settings y cambiar el nombre de la rama en GitHub)



**AHORA EL PASO ES
CONECTAR EL REPOSITORIO
LOCAL CON EL REPOSITORIO
REMOTO DE GITHUB**

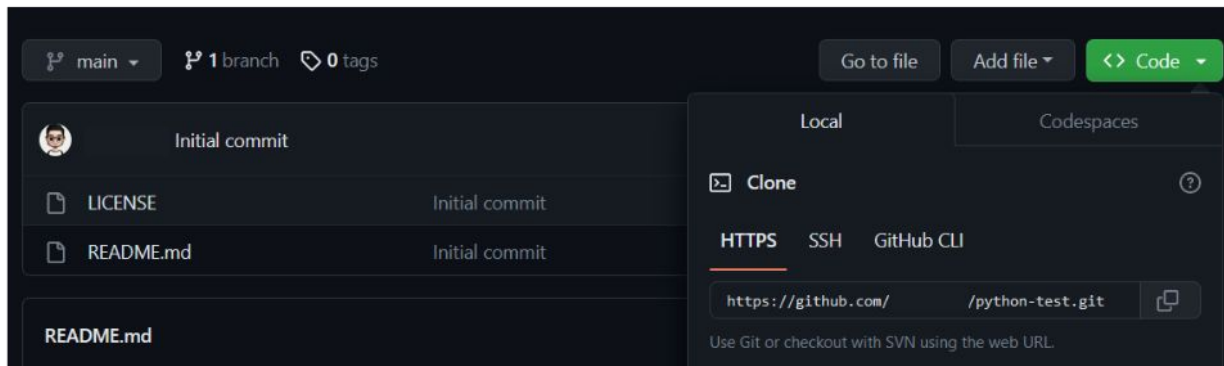


Conexión repositorio local con repositorio remoto

7. Ahora para sincronizar el repositorio local con el repositorio remoto creado en GitHub hay que utilizar el comando **git remote add origin <<URL HTTPS DEL REPOSITORIO>>**

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  powershell + v
PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git remote add origin https://github.com/[redacted]/python-test.git
```

LA URL DEL REPOSITORIO SE OBTIENE DESDE LA INTERFAZ DEL REPOSITORIO EN GITHUB, HACIENDO CLIC EN EL BOTON CODE Y ABAJO APARECERÁ URL HTTPS



Visual Studio Code

Corroborando repositorio remoto

8. Podemos verificar la información del repositorio remoto con **git remote -v**

```
PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git remote -v
origin https://github.com/          /python-test.git (fetch)
origin https://github.com/          /python-test.git (push)
```

Hay un **origin** para obtener los cambios del repositorio (fetch) y enviar los cambios(push)



Visual Studio Code





Enviando cambios al repositorio Remoto

9. Por último se envían los cambios al repositorio remoto con el comando **git push -u origin main**

```
PS C:\Users\Victo\OneDrive\Documentos\Programming\PythonPrograms\Python-Test> git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 668 bytes | 668.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/          /python-test.git
    main -> main
branch 'main' set up to track 'origin/main'.
```

Luego de ejecutar el comando **git push** podemos ir a nuestro repositorio en GitHub y corroborar que se encuentra el archivo que hemos creado en el repositorio local, lo que significa que se realizó la sincronización



	d057158 2 hours ago	🕒 3 commits
 LICENSE	Initial commit	4 hours ago
 README.md	Initial commit	4 hours ago
 factorial.py	Mi primer commit	2 days ago

(Es buena práctica de todos modos antes de hacer **git push** realizar un **git pull origin main** por si los archivos en el repositorio remoto han sido editados, con el fin de traernos a local la última versión para que estén sincronizados)



**DE ESTA MANERA SE LOGRA
SINCRONIZAR EL REPOSITORIO
LOCAL Y EL REMOTO**



Sitios Útiles

Errores comunes y buenas prácticas en Git

Preguntas relacionadas con GIT

[Stackoverflow](#)



Seidor

[Git soluciones errores típicos](#)



Midudev

[Buenas prácticas al utilizar comandos GIT](#)



Bibliografía



[1] «Git - La Guía Sencilla». <https://rogerdudler.github.io/git-guide/index.es.html>



[2] «Creación de un repositorio - Documentación de GitHub», GitHub Docs. <https://docs.github.com/es/get-started/quickstart/create-a-repo?tool=webui>



[3] «Glosario de GitHub - Documentación de GitHub», GitHub Docs. <https://docs.github.com/es/get-started/quickstart/github-glossary>



[4] «Git: tutorial básico del sistema de control de versiones», IONOS Digital Guide, 22 de julio de 2020. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/tutorial-de-git/>