### Justificación de las plataformas y herramientas de versionamiento.

**Git** es el sistema de control de versiones que decidimos implementar en el proyecto por varias razones, todo esto alineado con las necesidades de nuestro equipo:

* **Colaboración distribuida**: Nuestro equipo de desarrollo no siempre trabaja en el mismo lugar ni en el mismo horario. Con Git, cada integrante puede tener una copia completa del repositorio y realizar cambios localmente. Esto permite que todos trabajemos por cuenta propia, incluso sin conexión, lo cual es de gran ayuda para no depender de la conectividad constante con un servidor centralizado.
* **Eficiencia**: Git es muy rápido para realizar operaciones como commits, comparaciones y ramas. Nos permite hacer cambios en el código y retroceder si es necesario sin esperar tiempos de respuesta largos. Además, como el proyecto tiene varias etapas necesitamos una herramienta que permita realizar estas operaciones de manera ágil para mantener el ritmo de trabajo que tenemos.
* **Manejo de ramas (branches)**: Dado que el equipo trabaja en diferentes cosas al mismo tiempo, Git facilita la creación de ramas separadas para cada etapa. Cada miembro del equipo puede desarrollar en su propia rama sin afectar el código principal. Al final de cada sprint, podemos fusionar las ramas al repositorio principal (master) sin grandes conflictos. Esto es fundamental para nuestro proceso de desarrollo iterativo y colaborativo.
* **Historial completo de cambios**: El historial detallado de Git permite revisar todo el progreso del proyecto, quién hizo cada cambio y cuándo. Esto es muy útil para la revisión del código, detectar errores, lo cual es importante para mantener la calidad del proyecto.

### GitHub

Elegimos **GitHub** para alojar nuestros repositorios Git por las siguientes razones:

* **Colaboración y visibilidad del proyecto**: Dado que varios miembros del equipo están involucrados y algunos pueden estar en diferentes ubicaciones, GitHub nos permite trabajar en un entorno común donde todos pueden acceder al código en cualquier momento. GitHub ofrece un excelente control de acceso a los repositorios para garantizar que solo las personas autorizadas puedan modificar el código.
* **Gestión de Pull Requests**: GitHub facilita la colaboración a través de los **Pull Requests**, que nos permiten realizar revisiones de código antes de fusionar cambios en la rama principal. Esto asegura que el código revisado cumpla con los estándares del proyecto y evita que se introduzcan errores en el desarrollo.
* **Integración con herramientas de automatización**: Usamos **GitHub Actions** para integrar automatización en nuestro flujo de trabajo. Esto significa que cada vez que subimos código nuevo, se pueden correr automáticamente pruebas (tests) y verificar que todo funcione correctamente antes de fusionar los cambios. Esto nos ayuda a mantener la calidad del código y evitar problemas de última hora.

### Comparación y razones por no elegir otras herramientas:

* **SVN** no se aplicó en nuestro caso ya que, al ser un sistema centralizado, limitaría la capacidad de trabajar sin conexión y ralentizaría el desarrollo al requerir constantes accesos al servidor. Además, la gestión de ramas en SVN es más complicada, lo cual sería contraproducente en nuestro entorno de trabajo.
* **GitLab** fue considerado, pero elegimos **GitHub** debido a la facilidad de uso y la amplia integración con otras herramientas que ya utilizamos. GitHub también es más conocido entre los integrantes de nuestro equipo, lo que facilita su uso. Aunque GitLab ofrece más funcionalidades integradas como CI/CD, no necesitábamos esas capacidades adicionales en este momento, ya que GitHub Actions cumple con nuestras necesidades de automatización.

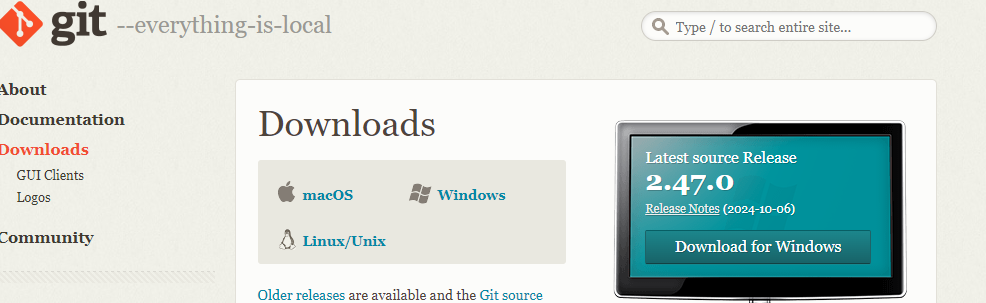
### Conclusión

La implementación de **Git** y **GitHub** se ajusta perfectamente a nuestro flujo de trabajo, permitiéndonos colaborar de manera eficiente, mantener un control estricto sobre los cambios, y automatizar pruebas. Estas herramientas nos ofrecen la flexibilidad y agilidad que necesitamos para mantenernos trabajando y manteniéndonos organizados, asegurando que el desarrollo del proyecto se de cómo lo planeamos y sin interrupciones.

### Parámetros de configuración de las plataformas y herramientas de versionamiento.

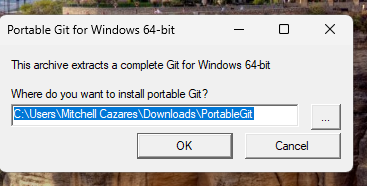
Para configurar Git de forma adecuada en cada equipo de desarrollo, seguimos los siguientes pasos clave:

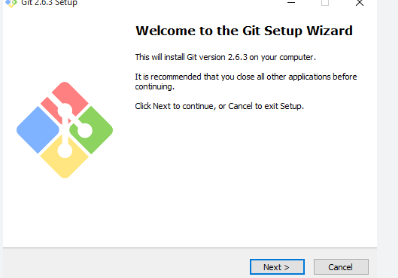
1. **Instalación de Git**:
   * Descargamos Git desde [git-scm.com](https://git-scm.com/) y lo instalamos en nuestro equipo de trabajo.

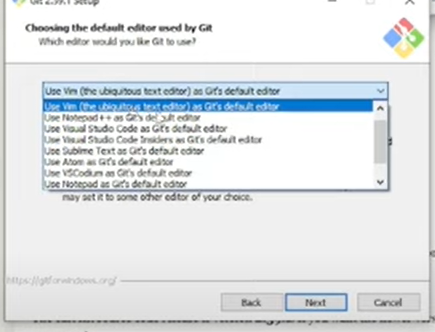


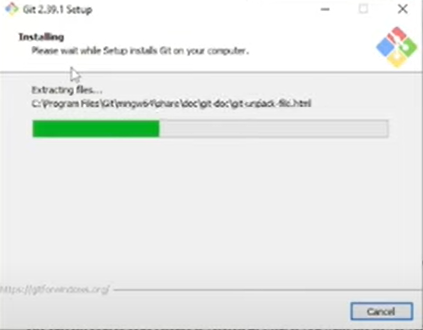


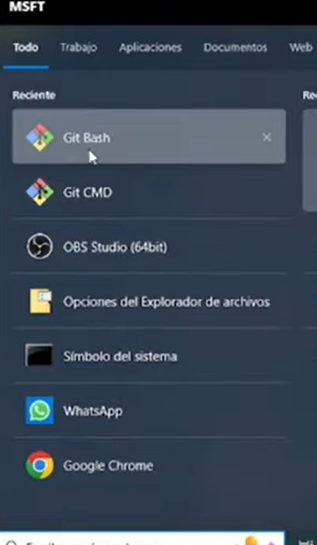
Aquí está instalándose, al finalizarse abrimos el archivo





Damos en siguiente y seguimos las instrucciones

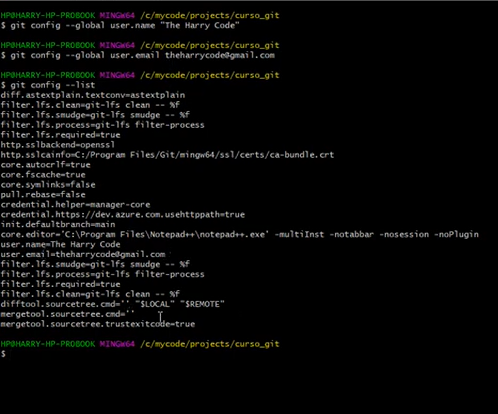


Después abrimos Git bash

1. **Configuración global de Git**:
   * Definimos los parámetros globales para cada usuario, de modo que cada commit estuviera asociado con la persona correcta. Esto es importante para el seguimiento del trabajo de cada miembro del equipo.
   * Los comandos que usamos son:

git config --global user.name "Nombre de Usuario"

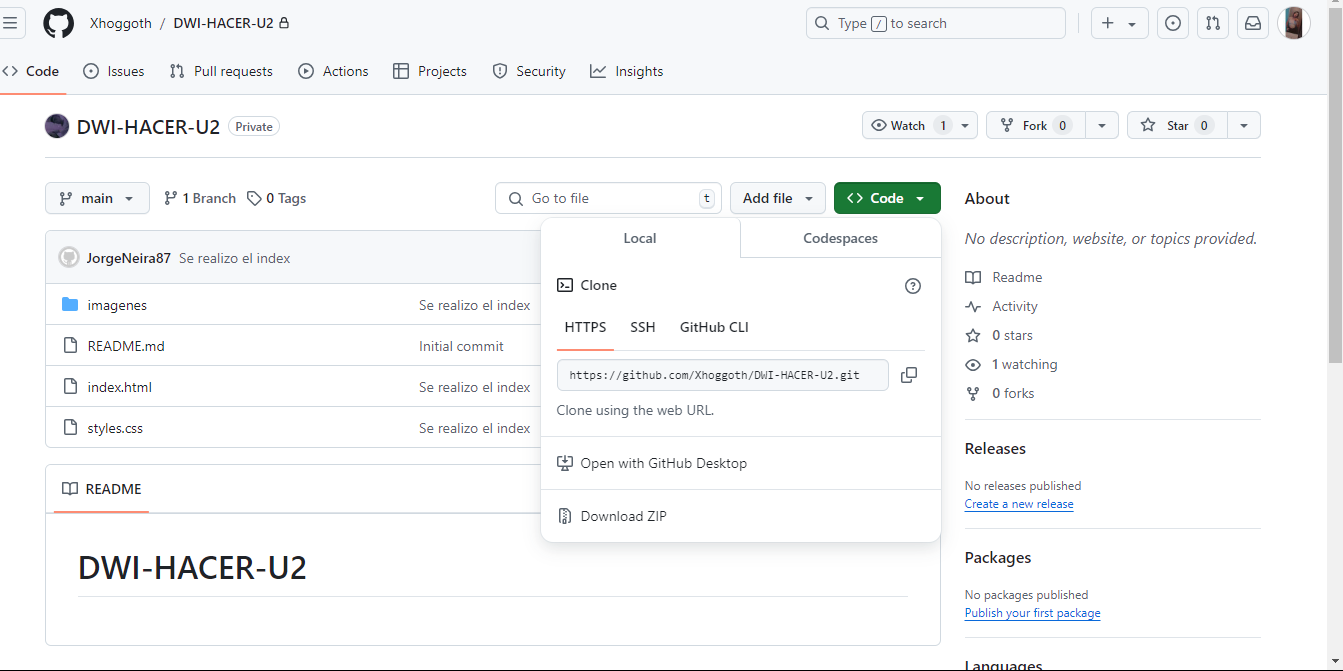
git config --global user.email [correo@ejemplo.com](mailto:correo@ejemplo.com)



1. **Creación del repositorio local**:
   * Cada miembro clona el repositorio central desde GitHub con el comando:

git clone https://github.com/usuario/proyecto.git

* + Esto asegura que todos trabajemos sobre la misma base de código y que tengamos acceso al historial de cambios completo.

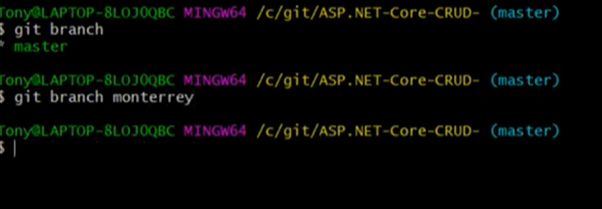


1. **Parámetros para trabajar con ramas**:
   * Para crear y cambiar de ramas, usamos los siguientes comandos:

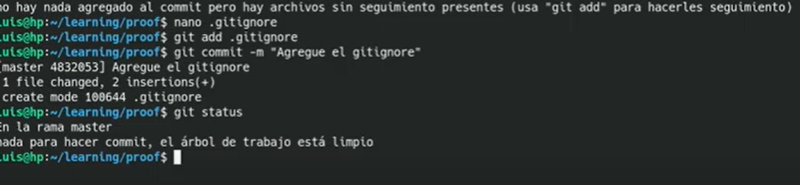
git branch

git checkout -b feature/nueva-funcionalidad

git checkout main



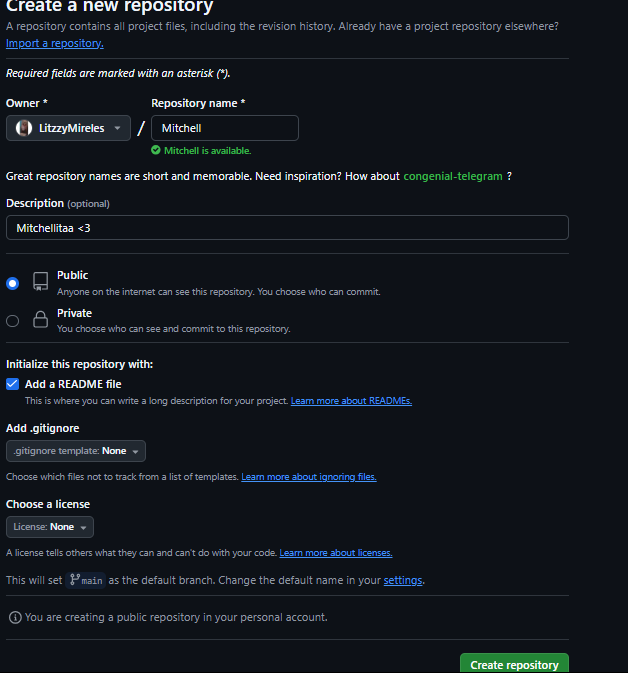
1. **Configuración de archivos ignorados**:
   * Usamos un archivo. gitignore para evitar que archivos innecesarios o sensibles se suban al repositorio.



### Parámetros de configuración de GitHub

Para que GitHub funcione correctamente como plataforma central para nuestro repositorio, realizamos las siguientes configuraciones:

1. **Creación y configuración del repositorio en GitHub**:
   * Creamos un nuevo repositorio en GitHub a través de la interfaz gráfica. Al crear el repositorio, seleccionamos la opción de inicializar con un archivo README y agregamos un archivo .gitignore específico para nuestro tipo de proyecto



1. **Acceso y permisos**:
   * Definimos roles y permisos para cada miembro del equipo en el proyecto. En GitHub, configuramos los niveles de acceso:
     + **Owner** (Propietario): Tiene control total sobre el repositorio.
     + **Colaborador** (Colaborador): Puede hacer commits, revisar y aceptar Pull Requests.
2. **Configuración de Pull Requests**:
   * Para garantizar que todas las contribuciones sean revisadas antes de integrarlas a la rama principal, configuramos reglas de protección para la rama main. Esto significa que no se pueden hacer commits directamente a main, sino que todas las contribuciones deben realizarse a través de Pull Requests.
   * En GitHub, activamos la opción de requerir revisiones para cada Pull Request, lo cual garantiza que al menos una persona revise el código antes de que se pueda fusionar.

### Conclusión

Estos parámetros de configuración permiten que el equipo trabaje de manera coordinada y eficiente, asegurando la integridad del código y facilitando la colaboración. Git y GitHub han sido configurados para adaptarse a las necesidades de nuestro proyecto, con control de acceso, automatización de pruebas, manejo de ramas estructurado y protección del código en la rama principal, lo que garantiza que cada cambio sea revisado y aprobado antes de integrarse.