**Git & GitHub**

[INTRODUCCION A GIT & GITHUB 2](#_Toc206826680)

[Git: 2](#_Toc206826681)

[GitHub: 2](#_Toc206826682)

[Resumen 2](#_Toc206826683)

[Acceso a Git y GitHub 2](#_Toc206826684)

[GIT 3](#_Toc206826685)

[Consola vs. GitBash 3](#_Toc206826686)

[Iniciar repositorio Git desde Terminal “Bash” 3](#_Toc206826687)

[Iniciar repositorio Git desde Símbolo de sistema 4](#_Toc206826688)

[Configuración de identidad 4](#_Toc206826689)

[Ciclo de vida de los archivos (Etapas) 5](#_Toc206826690)

[GITHUB 5](#_Toc206826691)

[Crear repositorio en GitHub 5](#_Toc206826692)

[Actualizar repositorio Git a repositorio GitHub vacío 6](#_Toc206826693)

[Almacenamiento de datos en GITHUB 6](#_Toc206826694)

[GIT & GITHUB: 8](#_Toc206826695)

[Clonar un proyecto desde consola 8](#_Toc206826696)

[Flujo de trabajo típico en GIT & GITHUB 9](#_Toc206826697)

[Desarrollo Colaborativo 9](#_Toc206826698)

[Compartir el entorno virtual con Google Drive 9](#_Toc206826699)

[Sincronizar con Google Drive vs. Exportar entornos virtuales 9](#_Toc206826700)

[Hay opciones diferentes a sincronizar Google Drive con los archivos de programa o exportar los entornos virtuales? 10](#_Toc206826701)

[Instrucciones si se hace el desarrollo inicial del proyecto computador (nivel local) 11](#_Toc206826702)

[Usando GitHub puedo ingresar desde cualquier computador a un proyecto, ejecutando previamente el archivo "requirements.txt"? 12](#_Toc206826703)

[Si GitHub permite clonar el repositorio en otra máquina y luego configurar y ejecutar el entorno virtual localmente, significa que después de trabajar el código actualizo mi repositorio en GitHub para que allí se guarden los avances del proyecto? 13](#_Toc206826704)

[Usualmente en el código se guarda todo lo relacionado con las conexiones a bases de datos. Qué debo considerar al respecto? 14](#_Toc206826705)

[GitHub es la mejor forma de replicar entornos y códigos de manera colaborativa. 14](#_Toc206826706)

# INTRODUCCION A GIT & GITHUB

Git y GitHub son conceptos diferentes, aunque están relacionados y a menudo se utilizan juntos.

Git se instala localmente en tu PC y GitHub es una plataforma en línea que puedes acceder mediante un navegador web para gestionar tus repositorios de Git de manera remota.

## Git:

* Git es un sistema de control de versiones distribuido que te permite realizar un seguimiento de los cambios en tu código a lo largo del tiempo.
* Te permite realizar acciones como realizar "commits" para guardar cambios, crear ramas para trabajar en nuevas características o correcciones, fusionar ramas, y más.
* Git se utiliza principalmente en la línea de comandos, aunque también hay interfaces gráficas disponibles.

## GitHub:

* GitHub es una plataforma en línea que utiliza Git para alojar repositorios de código y proporciona herramientas adicionales para la colaboración en equipo.
* GitHub facilita la colaboración entre desarrolladores al proporcionar funcionalidades como solicitudes de extracción, problemas (issues), seguimiento de proyectos, y más.
* Aunque GitHub es una de las plataformas más populares que utiliza Git, hay otras plataformas similares como GitLab y Bitbucket.

## Resumen

Git es el sistema de control de versiones que maneja el seguimiento de cambios en tu código, mientras que GitHub es una plataforma en línea que utiliza Git para alojar repositorios y proporciona herramientas colaborativas adicionales:

* Git se instala y se usa a nivel local
* GitHub se accede/usa por un navegador

Se puede usar Git de forma independiente en tu máquina local sin necesidad de una plataforma en línea como GitHub. Sin embargo, GitHub (u otras plataformas similares) puede ser útil para colaborar con otros desarrolladores, compartir tu código de manera pública o privada, realizar un seguimiento de problemas y solicitudes de extracción, y aprovechar otras características de colaboración que ofrece la plataforma.

## Acceso a Git y GitHub

Git es un sistema de control de versiones distribuido que se debe tener instalado en el PC para rastrear cambios en proyectos de software. Se descarga desde el sitio web oficial (<https://git-scm.com/>) y se instala en la máquina.

GitHub, por otro lado, es una plataforma de alojamiento de repositorios de Git en la nube. Una vez que has instalado Git en tu PC, puedes utilizarlo para interactuar con repositorios tanto en tu máquina local como en GitHub. Se accede a GitHub a través de un navegador web, donde se pueden crear repositorios, colaborar con otros desarrolladores, gestionar problemas (issues), y realizar otras acciones relacionadas con el desarrollo de software.

En resumen:

* Git se instala y se usa localmente en tu PC y
* GitHub es una plataforma en línea que puedes acceder mediante un navegador web para gestionar tus repositorios de Git de manera remota.

# GIT

## Consola vs. GitBash

En términos generales, es esencialmente lo mismo trabajar con Git a través de la consola de comandos estándar y trabajar con Git Bash.

Git Bash es simplemente una emulación de terminal que proporciona un entorno de línea de comandos similar al que encontrarías en sistemas Unix, pero diseñado para funcionar en entornos Windows.

La mayoría de los comandos de Git que se usan en la línea de comandos estándar funcionarán de la misma manera en Git Bash. La elección entre utilizar la consola de comandos estándar o Git Bash depende principalmente de tu preferencia y del entorno en el que te sientas más cómodo. Ambas opciones proporcionan la funcionalidad completa de Git, así que puedes elegir la que te resulte más conveniente.

Importante: **Algunos comandos específicos de Unix pueden no funcionar en la consola de comandos estándar de Windows, pero funcionarán en Git Bash.**

En general, Git Bash es una opción popular entre los desarrolladores que trabajan en entornos Windows y desean una experiencia de línea de comandos más familiar.

Principio del formulario

## Iniciar repositorio Git desde Terminal “Bash”

Para iniciar un repositorio de Git en un proyecto existente a nivel local:

1. **Crear una carpeta para el Proyecto**: navega a él en la terminal:

mkdir mi-proyecto

cd mi-proyecto

1. **Navega a tu proyecto en la terminal**: por ejemplo, abriendo Git Bash:
   * Busca la carpeta de tu proyecto en el Explorador de archivos.
   * Haz clic derecho dentro de la carpeta.
   * Selecciona "Git Bash Here" para abrir la terminal de Git Bash en ese directorio.
2. **Inicializa un nuevo repositorio Git:**

En la terminal de Git Bash, ejecutar el comando **git init** para iniciar un nuevo repositorio Git en tu proyecto. Este comando crea un nuevo subdirectorio llamado **.git** que contiene todos los archivos necesarios para el repositorio Git.

1. **Desarrollar tu Proyecto**:

Agrega y edita archivos en tu proyecto localmente.

1. **Añade archivos al área de preparación (staging) del repositorio:**

Después de inicializar el repositorio, puedes agregar archivos al área de preparación (**staging**) usando el comando:

Opción a: git add . (para todos los archivos).

Opción b: git add nombre\_del\_archivo (para archivos individuales).

1. **Realiza un commit inicial:**

Después de agregar los archivos, realiza un commit con el comando: git commit -m "Primer commit"

Este comando guarda los cambios en el repositorio con un mensaje descriptivo.

De esta forma se ha configurado un repositorio Git local en el proyecto.

Se puede seguir trabajando en el proyecto y realizar más commits a medida que se realicen cambios.

Si en el futuro se colabora con otros desarrolladores o se necesita respaldar el proyecto en línea, se puede conectar el repositorio local a un repositorio remoto en GitHub u otra plataforma de alojamiento.

## Iniciar repositorio Git desde Símbolo de sistema

Una vez que has ingresado a la consola y te encuentras en la carpeta de tu proyecto, aquí están los pasos básicos que puedes seguir para iniciar un repositorio Git y realizar tu primer commit:

1. Crear un Proyecto: Crea un nuevo directorio para tu proyecto y navega a él en la terminal:

mkdir mi-proyecto

cd mi-proyecto

1. **Inicializa un nuevo repositorio Git:** Utiliza el comando para iniciar un nuevo repositorio Git en tu proyecto:

git init → Este comando creará un nuevo subdirectorio llamado **.git** en tu carpeta, que contendrá la información del repositorio.

1. **Desarrollar tu Proyecto:**

Agrega y edita archivos en tu proyecto localmente.

1. **Añade archivos al área de preparación (staging):** Puedes agregar todos los archivos de tu proyecto al área de preparación utilizando:

git add .

O, si solo quieres agregar archivos específicos, puedes hacerlo individualmente:

git add nombre\_del\_archivo

1. **Realiza tu primer commit:** Después de agregar los archivos al área de preparación, realiza tu primer commit con el siguiente comando:

git commit -m "Primer commit"

Asegúrate de incluir un mensaje descriptivo que resuma los cambios realizados en este commit.

A partir de aquí, se ha creado un repositorio local de Git y se ha realizado el primer commit.

Se puede continuar trabajando en el proyecto y realizar commits adicionales a medida que se realicen cambios.

Al colaborar con otros desarrolladores o respaldar el proyecto en línea, se puede vincular el repositorio local a un repositorio remoto en GitHub u otra plataforma similar.

## Reingreso a Git Bash

* Activar Git Bash
* Ir a la carpeta del proyecto: **cd /c/webempleados**
* Probar que no arroje error el comando: **git status**

## Archivo “.gitignore”

No todos los archivos deben guardarse en el repositorio (por ejemplo, el entorno virtual) por cuestiones de:

* Peso/volumen
* Contiene binarios y dependencias que no necesitas subir a GitHub.
* Puede reconstruirse fácilmente con requirements.txt o pyproject.toml

Para ignorar archivos de configuración en Git, hay que asegurarse de agregar archivos de configuración sensibles al archivo “.gitignore” para evitar que se suban accidentalmente a tu repositorio en GitHub.

Se debe crear el archivo “.gitignore” en la carpeta del proyecto con la ayuda del editor de texto u otro medio. Pero debemos asegurarnos de que el archivo no tenga extensión. Dado el caso hay que ordenarle a Windows que muestre las extensiones de los archivos para verificar y eliminar la extensión si es el caso.

El contenido mínimo recomendado para el caso (Python con entorno virtual):

# Entornos virtuales

venv01/

.venv01/

# Cachés de Python

\_\_pycache\_\_/

\*.py[cod]

\*.pyo

\*.pyd

# Archivos de base de datos locales

\*.sqlite3

\*.db

# Archivos temporales

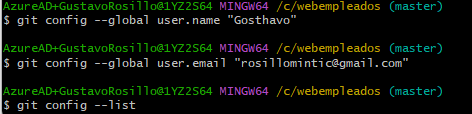
\*.log

\*.tmp

En caso de haber hecho un commit con anterioridad a la creación del archivo gitignore se debe correr “git add .gitignore“ y hacer un nuevo commit

## Configuración de identidad

Para configurar identidad globalmente y que todos los repositorios usen un correo personal (o específico) de GitHub, ejecuta en Git:



Para configurar identidad solo afecte en un repositorio/proyecto (y dejar otros repos con otro correo), entonces:

git config user.name "Gustavo Rosillo"

git config user.email "gustavo.rosillo@gmail.com"

*Nota: aquí no va --global*

## Ciclo de vida de los archivos (Etapas)

En Git, los archivos pueden pasar por tres etapas durante su ciclo de vida en un repositorio:

1. **Directorio de trabajo (Working Directory)**: Esta es la etapa donde se encuentran los archivos tal como están en el sistema de archivos local. Se pueden modificar, eliminar o agregar archivos en esta etapa sin que Git esté al tanto de esos cambios.
2. **Área de preparación (Staging Area o Index)**: También conocida como "staging", es una etapa intermedia donde los cambios realizados en los archivos se preparan para ser confirmados en el repositorio. Los archivos en esta etapa se han agregado con el comando **git add .** pero aún no se han confirmado con git commit.
3. **Repositorio (Repository)**: Esta es la etapa final donde se almacenan permanentemente las versiones confirmadas de los archivos. Los archivos en esta etapa se confirman utilizando el comando **git commit**. Después de la confirmación, los cambios se convierten en parte del historial del repositorio.

Para resumir:

* **Working Directory**: Los archivos tal como están en tu sistema local.
* **Staging Area**: Los archivos preparados para ser confirmados.
* **Repository**: Los archivos confirmados y parte del historial del repositorio.

# GITHUB

## Crear repositorio en GitHub

1. **Crea un nuevo repositorio en GitHub:**
   * Ir a la página principal de GitHub (<https://github.com/>).
   * Iniciar sesión en tu cuenta (si no lo has hecho).
   * En la esquina superior derecha, clic en el signo más **+** y selecciona "Nuevo repositorio".
   * Completar la información del repositorio, como el nombre, la descripción y otras opciones.
   * Hacer clic en "Crear repositorio".
2. **Conecta tu repositorio local con el repositorio remoto en GitHub:**

En tu terminal de Git Bash, navega al directorio de tu proyecto. Ejecuta los siguientes comandos:

git remote add origin <URL\_del\_repositorio\_en\_GitHub> (obtenida al crear repositorio GitHub)

git branch -M main # usar "main" si ese es el nombre de tu rama principal

git push -u origin main

Estos comandos añaden un origen remoto llamado "origin" y empujan tu código al repositorio remoto.

1. **Verifica en GitHub:**

Actualiza la página de tu repositorio en GitHub y deberías ver tus archivos allí.

De esta forma se ha llevado el proyecto a un nuevo repositorio en GitHub. Se puede seguir trabajando en el proyecto y realizar **commits**, y cuando haya que actualizar el repositorio en GitHub con los cambios locales, ejecuta el comando **git push origin main** (o el nombre de tu rama principal) en la terminal.

## Fasfk;asfkd;

**🔗 1. Conectar tu repositorio local con GitHub**

Obtener la URL del Repositorio de GitHub: En la página del repositorio en GitHub, buscar y copiar la URL del repositorio. Puede ser algo como [**https://github.com/tu-usuario/tu-repositorio.git**](https://github.com/tu-usuario/tu-repositorio.git) o para el caso https://github.com/rosillogithub/Web-Empleados.git

En la terminal, parado en la carpeta de tu proyecto:

git remote add origin https://github.com/rosillogithub/Web-Empleados.git

Verificar que quedó bien:

git remote -v

Debe mostrar origin con la URL de tu repo en GitHub.

**🚀 2. Subir el proyecto por primera vez**

Si todavía no hiciste un commit inicial:

git add .

git commit -m "Primer commit - versión inicial"

Luego subir al repositorio remoto (GitHub):

git push -u origin master

⚠️ Vigilar el nombre de la rama (master o main) para hacer el cambio en el comando que convenga.

**🔄 3. Flujo normal de trabajo diario**

Cada vez que trabajes en tu proyecto:

1. **Ver qué cambió**

git status

1. **Preparar cambios**

git add . (o podés elegir archivo por archivo: git add archivo.py)

1. **Guardar cambios en la historia local**

git commit -m "Mensaje corto y claro de lo que hiciste"

1. **Enviar a GitHub**

git push

**📥 4. Traer cambios de GitHub (si trabajas en otro PC o alguien más colabora)**

git pull

📌 **Tip:** el -u origin master/main solo se usa la primera vez; después con git push basta.

Validar la ejecución de: git push -u origin master

Con el comando se envió el contenido del repositorio local al repositorio remoto en **GitHub**.

Ahora, para **verificar que la información migró correctamente**, haz lo siguiente:

**🔎 Paso 1: Revisar en GitHub**

1. ir al repositorio en GitHub (ejemplo: https://github.com/tu\_usuario/tu\_repositorio).
2. Refrescar la página.
3. Allí se debe ver:
   * Los **archivos de tu proyecto** (excepto los ignorados en .gitignore).
   * Un **mensaje de commit** con el texto que usaste en git commit -m "...".

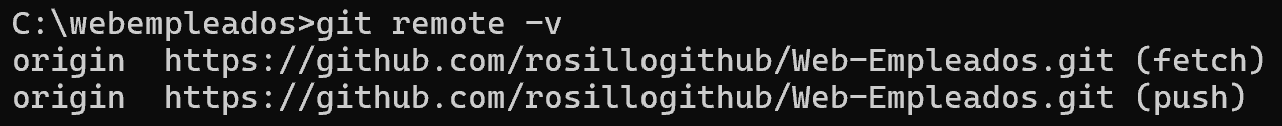
A screenshot of a computer

AI-generated content may be incorrect.

**🔎 Paso 2: Revisar desde Git en tu PC**

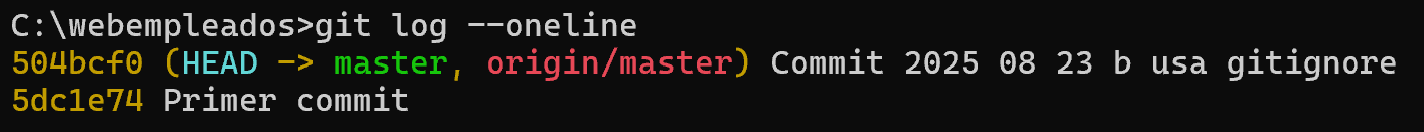
Para confirmar que el repositorio remoto está vinculado a GitHub.

git remote -v



Para ver el historial de commits locales (el último commit debe coincidir con el que ves en GitHub).

git log –oneline



## Actualizar repositorio Git a repositorio GitHub vacío

Hechos los commits en el repositorio local y creado un repositorio vacío en GitHub:

1. **Obtener la URL del Repositorio de GitHub:**

En la página del repositorio en GitHub, buscar el botón verde que dice "Code" y copiar la URL proporcionada. Puede ser algo como [**https://github.com/tu-usuario/tu-repositorio.git**](https://github.com/tu-usuario/tu-repositorio.git).

<https://github.com/rosillogithub/DA_ProyectoFinal.git>

1. **Conecta tu Repositorio Local con GitHub:**

Abre la terminal (Git Bash) en la carpeta de tu proyecto local. Ejecuta los comandos:

* git remote add origin <URL\_del\_repositorio\_en\_GitHub> (obtenida del botón “code” en GitHub)
* git branch -M main # usar "main" si ese es el nombre de tu rama principal
* git push -u origin main

1. **Verifica en GitHub:**

Refresca la página de tu repositorio en GitHub. Deberías ver tus archivos y commits reflejados en el repositorio remoto.

De esa manera se ha llevado el proyecto de Git a GitHub.

Se puede continuar trabajando en el proyecto y realizar más commits, y cuando haya que actualizar el repositorio en GitHub con los cambios locales, ejecuta el comando **git push origin main** (o el nombre de tu rama principal) en la terminal.

Principio del formulario

## Almacenamiento de datos en GITHUB

GitHub está diseñado principalmente para almacenar y gestionar código fuente y archivos relacionados con el desarrollo de software.

Si bien puedes almacenar diversos tipos de archivos en un repositorio de GitHub, como documentos de texto, imágenes y archivos de configuración, no es la mejor opción para almacenar grandes conjuntos de datos o archivos binarios. GitHub impone ciertos límites en el tamaño total del repositorio y en el tamaño individual de los archivos. Además, no es una solución eficiente para la gestión de grandes conjuntos de datos debido a la forma en que se manejan las actualizaciones y las versiones.

Para almacenar grandes conjuntos de datos o archivos binarios, es mejor considerar otras soluciones específicas para gestión de datos, almacenamiento en la nube o sistemas de almacenamiento distribuido.

**Git LFS (Large File Storage)**: Git LFS es una extensión de Git diseñada para manejar grandes archivos binarios.

Permite almacenar estos archivos de manera eficiente y mantener un enlace simbólico en el repositorio de Git.

**Servicios de Almacenamiento en la Nube**: Puedes utilizar servicios de almacenamiento en la nube como Amazon S3, Google Cloud Storage o Azure Blob Storage para almacenar grandes conjuntos de datos y archivos binarios. Puedes incluir en tu repositorio solo referencias o metadatos que apunten a estos archivos en la nube.

**Plataformas de Datos Específicas**: Para conjuntos de datos grandes, es posible que desees considerar plataformas específicas para la gestión de datos, como DataHub, DVC (Data Version Control) o plataformas de almacenamiento y gestión de datos distribuidos.

En resumen, mientras que GitHub es excelente para el código fuente y archivos pequeños relacionados con el desarrollo, para grandes conjuntos de datos o archivos binarios, es preferible utilizar soluciones específicas diseñadas para ese propósito.

## Clonar un proyecto desde consola

La **terminal o línea de comandos** permite interactuar con Git utilizando los mismos comandos usados en **Git Bash** u otra interfaz gráfica. Aquí, algunos comandos básicos de Git que se pueden ejecutar desde la consola:

1. Clonar un repositorio remoto: git clone <URL\_del\_repositorio\_remoto>
2. Configurar el Entorno Virtual y Dependencias:

Configura un entorno virtual y activa el entorno.

Luego, instala las dependencias desde el archivo requirements.txt:

python -m venv mi\_entorno\_virtual

source mi\_entorno\_virtual/bin/activate

# En macOS/Linux; usa "mi\_entorno\_virtual\Scripts\activate"

# en Windows pip install -r requirements.txt

1. Ejecuta el Proyecto:

Ahora se puede ejecutar el proyecto localmente en el nuevo computador.

1. Agregar cambios al área de preparación: git add <nombre\_del\_archivo>

Para agregar todos los archivos modificados: git add .

1. Realizar un commit: git commit -m "Mensaje del commit"
2. Ver el estado del repositorio: git status
3. Ver el historial de commits: git log
4. Sincronizar el repositorio local con cambios hechos en el repositorio remoto

git pull origin <nombre\_de\_la\_rama>

Se utiliza para obtener y combinar (hacer merge) los cambios de la rama <nombre\_de\_la\_rama> del repositorio remoto con la rama actual en el repositorio local.

Combina dos comandos: git fetch (descarga los datos) y git merge (los integra en tu rama actual).

1. Sincronizar o enviar los cambios del repositorio local al repositorio remoto:

git push origin <nombre\_de\_la\_rama>

Se utiliza para subir los commits de tu rama local <nombre\_de\_la\_rama> al repositorio remoto, permitiendo que otros colaboradores puedan ver y trabajar con los cambios que has realizado.

Este flujo de trabajo te permite desarrollar localmente, versionar tu código con Git, y luego subirlo a GitHub para colaborar y compartir el código con otros o simplemente mantener una copia de seguridad en línea.

Estos son solo algunos ejemplos básicos.

Se pueden realizar la mayoría de las operaciones de Git desde la línea de comandos.

Es útil tener un conocimiento básico de la interfaz de línea de comandos de Git, ya que brinda flexibilidad y control sobre el flujo de trabajo.

Principio del formulario

## Flujo de trabajo típico en GIT & GITHUB

Después de crear los repositorios, el flujo de trabajo típico en Git & Github implica:

* modificar los archivos en el directorio de trabajo,
* luego agregar los cambios relevantes al área de preparación con **git add .**
* confirmar esos cambios en el repositorio con **git commit**
* Enviar los cambios del repositorio local al repositorio remoto con: git push origin <nombre\_de\_la\_rama>

Opcional:

* Traer los cambios hechos en el repositorio remoto al repositorio local: git pull origin <nombre\_de\_la\_rama>

# Desarrollo Colaborativo

## Compartir el entorno virtual con Google Drive

Se puede compartir el archivo requirements.txt (o el equivalente en herramientas como pipenv o conda) a través de Google Drive y luego descargarlo en otro computador para recrear el entorno virtual. A continuación un ejemplo de cómo hacerlo:

1. Generar el archivo requirements.txt:

Desde el entorno virtual activado, ejecuta el siguiente comando para generar un archivo requirements.txt:

pip freeze > requirements.txt → Guarda todas las dependencias y versiones en el archivo requirements.txt.

1. Subir el archivo a Google Drive → Sube el archivo requirements.txt a una cuenta Google Drive.
2. Descargar el archivo en otro computador →

En otro computador, descarga el archivo requirements.txt desde Google Drive.

1. Crear y activar un nuevo entorno virtual:

Crea un nuevo entorno virtual en el nuevo computador y actívalo

python -m venv mi\_nuevo\_entorno\_virtual

# Activar el entorno virtual (depende del sistema operativo)

1. Instalar dependencias desde requirements.txt en el nuevo entorno virtual:

pip install -r path/to/requirements.txt

Asegúrate de proporcionar la ruta correcta al archivo requirements.txt descargado desde Google Drive.

Esta es una forma conveniente de compartir entornos virtuales a través de Google Drive. Solo se comparte la lista de dependencias y versiones, no los archivos específicos o configuraciones del entorno virtual.

## Opciones gestionar proyectos y repositorios?

Existen varias opciones adicionales para manejar entornos virtuales y archivos relacionados con proyectos de desarrollo. Algunas opciones incluyen:

1. **Control de versiones con Git:**
   * Utilizar un sistema de control de versiones como Git puede ser una excelente opción para gestionar proyectos de desarrollo.
   * Puedes almacenar tu código, entornos virtuales y otros archivos relacionados en repositorios Git, lo que facilita el seguimiento de cambios y colaboración.
2. **Plataformas de gestión de proyectos:**
   * Plataformas como GitHub, GitLab o Bitbucket no solo ofrecen control de versiones, sino también herramientas de gestión de proyectos y colaboración.
   * Puedes almacenar y gestionar tus proyectos de manera centralizada, incluyendo el código, entornos virtuales y documentación.
3. **Contenedores Docker:**
   * Docker es una tecnología de contenedores que permite empaquetar una aplicación y sus dependencias en un contenedor aislado.
   * Puedes definir el entorno de tu aplicación en un archivo llamado Dockerfile, lo que facilita la replicación del entorno en diferentes sistemas.
4. **Entornos virtuales con herramientas de gestión avanzada:**
   * Herramientas como **pipenv**, **poetry** o **conda** ofrecen características avanzadas para la gestión de entornos virtuales y dependencias.
   * Pueden simplificar la creación, activación y gestión de entornos virtuales, así como la instalación de dependencias específicas.
5. **Almacenamiento en la nube con otras plataformas:**
   * Además de Google Drive, otras plataformas de almacenamiento en la nube como Dropbox, OneDrive, o Box pueden ser opciones para almacenar y sincronizar archivos relacionados con el desarrollo.

La elección entre estas opciones dependerá de tus preferencias personales, requisitos específicos del proyecto y del equipo, así como del flujo de trabajo que mejor se adapte a tus necesidades. Puedes combinar varias de estas opciones según tu contexto y los requisitos de tus proyectos.

## Instrucciones: desarrollo inicial del proyecto en computador personal (nivel local)

Si estás iniciando el desarrollo de tu proyecto localmente en tu computadora y luego deseas subirlo a GitHub, aquí hay una guía básica:

**1. Iniciar un Proyecto Localmente:**

1. **Crear un Proyecto:**
   * Crea un nuevo directorio para tu proyecto y navega a él en la terminal:

mkdir mi-proyecto cd mi-proyecto

1. **Inicializar un Repositorio Git Local:**

Inicializa un repositorio Git local en tu proyecto:

git init

Esto establecerá tu directorio local como un repositorio Git.

1. **Desarrollar tu Proyecto:**

Agrega y edita archivos en tu proyecto localmente.

Realiza commits a medida que avanzas en el desarrollo:

git add .

git commit -m "Primer commit"

**2. Crear un Repositorio en GitHub:**

1. **Ir a GitHub:**

Ve a la página principal de GitHub e inicia sesión en tu cuenta.

1. **Crear un Nuevo Repositorio:**

Crea un nuevo repositorio haciendo clic en el botón "New" en la parte superior derecha de tu página de inicio de GitHub.

1. **Sigue las Instrucciones de GitHub:**
   * GitHub te proporcionará instrucciones sobre cómo agregar un repositorio existente desde la línea de comandos. Sigue esas instrucciones.
   * En resumen, serán algo así como:

git remote add origin <https://github.com/tu-usuario/tu-proyecto.git>

git branch -M main

git push -u origin main

**3. Clonar el Repositorio en Otro Computador:**

**Clonar el Repositorio:**

En otro computador, puedes clonar el repositorio desde GitHub utilizando el comando **git clone**:

git clone <https://github.com/tu-usuario/tu-proyecto.git>

**Configurar el Entorno Virtual y Dependencias:**

Como mencionamos anteriormente, configura un entorno virtual y activa el entorno. Luego, instala las dependencias desde el archivo **requirements.txt**:

python -m venv mi\_entorno\_virtual

source mi\_entorno\_virtual/bin/activate

# En macOS/Linux; usa "mi\_entorno\_virtual\Scripts\activate"

# en Windows pip install -r requirements.txt

**Ejecutar el Proyecto:**

Ahora puedes ejecutar tu proyecto localmente en este nuevo computador.

Este flujo de trabajo te permite desarrollar localmente, versionar tu código con Git, y luego subirlo a GitHub para colaborar y compartir el código con otros o simplemente mantener una copia de seguridad en línea.

Principio del formulario

## Usando GitHub puedo ingresar desde cualquier computador a un proyecto, ejecutando previamente el archivo "requirements.txt"?

GitHub es una plataforma en línea que alberga repositorios de código, y no te proporciona directamente la capacidad de ejecutar archivos, como el archivo **requirements.txt**, en un entorno virtual en tu máquina local simplemente accediendo al proyecto en GitHub. Sin embargo, te permite clonar el repositorio en tu máquina y luego configurar y ejecutar el entorno virtual localmente.

Aquí hay una guía básica paso a paso:

1. **Clonar el Repositorio:**
   * En tu máquina local, abre la terminal y utiliza el comando **git clone** para clonar el repositorio de GitHub a tu máquina:

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

* + Reemplaza "tu-usuario" y "tu-proyecto" con tu nombre de usuario de GitHub y el nombre de tu proyecto, respectivamente.

1. **Configurar el Entorno Virtual:**
   * Navega al directorio del proyecto clonado:

cd tu-proyecto

* + Crea un entorno virtual (puedes usar **venv**, **virtualenv**, **conda**, etc.):

python -m venv mi\_entorno\_virtual

* + Activa el entorno virtual:

En Windows mi\_entorno\_virtual\Scripts\activate

En macOS/Linux source mi\_entorno\_virtual/bin/activate

1. **Instalar Dependencias desde requirements.txt:**
   * Instala las dependencias del archivo **requirements.txt**:

pip install -r requirements.txt

1. **Ejecutar el Proyecto:**
   * Dependiendo del tipo de proyecto (web, scripts, etc.), puedes ejecutarlo según la configuración específica del proyecto.
   * Por ejemplo, si es un proyecto de Django, podrías correr:

python manage.py runserver

Asegúrate de que tu entorno virtual esté activado al ejecutar el proyecto.

Con estos pasos, habrás configurado el entorno virtual y estarás listo para ejecutar el proyecto localmente en tu máquina, después de haber clonado el código desde GitHub.

## Si GitHub permite clonar el repositorio en otra máquina y luego configurar y ejecutar el entorno virtual localmente, significa que después de trabajar el código actualizo mi repositorio en GitHub para que allí se guarden los avances del proyecto?

¡Exacto! Después de realizar cambios en tu código localmente y trabajar en tu proyecto, puedes actualizar tu repositorio en GitHub para reflejar esos avances.

Aquí hay un conjunto de pasos comunes que puedes seguir:

1. **Añadir y Confirmar Cambios:**
   * Antes de actualizar tu repositorio en GitHub, debes confirmar los cambios locales. Utiliza los siguientes comandos en tu terminal:

# Asegurarte de estar en la rama correcta

git checkout tu-rama

# Añadir todos los cambios

git add .

# Realizar un commit con un mensaje descriptivo

git commit -m "Descripción de los cambios realizados"

1. **Subir Cambios al Repositorio de GitHub:**
   * Después de realizar los commits locales, puedes subir los cambios al repositorio de GitHub utilizando el comando **git push**:

git push origin tu-rama o git push origin main

* + Reemplaza "tu-rama" con el nombre de la rama en la que estás trabajando.

1. **Verificar los Cambios en GitHub:**
   * Ve a la página de tu repositorio en GitHub. Deberías ver los últimos cambios reflejados en la interfaz web.
   * Si estás trabajando en una rama diferente, es posible que desees crear una solicitud de extracción (pull request) para fusionar tus cambios en la rama principal del proyecto.

Estos pasos permitirán mantener actualizado el repositorio en GitHub con los avances realizados en el código local. La capacidad de clonar, trabajar localmente y luego subir los cambios es una parte fundamental del flujo de trabajo en Git y GitHub, permitiendo la colaboración y el seguimiento eficiente de los cambios en tu proyecto.

Principio del formulario

## Usualmente en el código se guarda todo lo relacionado con las conexiones a bases de datos. Qué debo considerar al respecto?

Guardar información de conexión a bases de datos en el código puede ser común, pero es importante manejar la información de manera segura y seguir buenas prácticas de seguridad. Aquí hay algunas recomendaciones:

1. **Evitar Credenciales Duras en el Código:**

Evita almacenar credenciales de bases de datos directamente en el código fuente. Esto es riesgoso ya que cualquier persona que tenga acceso al código podría ver las credenciales.

1. **Usar Variables de Entorno o Archivos de Configuración:**

Guarda información sensible, como credenciales de bases de datos, en variables de entorno o archivos de configuración externos. De esta manera, puedes mantener la configuración separada del código fuente.

1. **Ignorar Archivos de Configuración en Git:**

Asegúrate de agregar archivos de configuración sensibles a tu archivo “.gitignore” para evitar que se suban accidentalmente a tu repositorio en GitHub.

1. **Utilizar Herramientas de Gestión de Secretos:**

Considera el uso de herramientas de gestión de secretos como “[HashiCorp” “Vault](https://www.vaultproject.io/" \t "_new)” o servicios de gestión de secretos en la nube para almacenar y recuperar credenciales de forma segura.

1. **Configuración Diferenciada por Entorno:**

Organiza tu código y configuración para admitir diferentes entornos (desarrollo, prueba, producción) de manera que puedas configurar las conexiones de bases de datos de manera diferente en cada entorno.

1. **Acceso Controlado:**

Limita el acceso a la información de configuración sensible solo a aquellos que necesitan conocerla. No todos los miembros del equipo necesitan acceso a las credenciales de bases de datos en todos los entornos.

Recuerda que la seguridad es una consideración crítica en el desarrollo de software. Al seguir buenas prácticas de seguridad, puedes proteger la información sensible y minimizar los riesgos asociados con la gestión de credenciales en tu código fuente.

## GitHub es la mejor forma de replicar entornos y códigos de manera colaborativa.

GitHub es una plataforma muy popular y efectiva para la colaboración y gestión de proyectos de software. Aquí hay algunas razones por las cuales GitHub es ampliamente utilizado para replicar entornos y códigos de manera colaborativa:

1. **Control de Versiones:** GitHub utiliza Git como su sistema de control de versiones, lo que permite un seguimiento preciso de los cambios en el código a lo largo del tiempo. Cada contribución se registra como un commit, lo que facilita la comprensión de la evolución del proyecto.
2. **Colaboración:** GitHub proporciona herramientas integradas para la colaboración en equipo, como solicitudes de extracción (pull requests), problemas (issues), y comentarios en línea. Esto facilita la revisión del código, la discusión de cambios propuestos y la gestión de tareas.
3. **Repositorios Remotos:** Puedes almacenar tus repositorios en la nube, lo que facilita la colaboración entre equipos distribuidos geográficamente. Los miembros del equipo pueden clonar, realizar cambios y enviar esos cambios de nuevo al repositorio central.
4. **Gestión de Dependencias:** Puedes utilizar archivos como **requirements.txt**, **Pipfile** o **environment.yml** para gestionar las dependencias y entornos virtuales. Esto facilita la replicación del entorno de desarrollo en diferentes máquinas.
5. **Automatización:** GitHub Actions permite la automatización de tareas, como la ejecución de pruebas, la implementación continua y otras actividades del flujo de trabajo del desarrollo.
6. **Integración con Herramientas de Desarrollo:** GitHub se integra con una amplia variedad de herramientas y servicios, lo que facilita la incorporación de nuevas funciones y flujos de trabajo personalizados.

Aunque GitHub es muy popular, hay otras plataformas de desarrollo colaborativo, como GitLab y Bitbucket, que también ofrecen funcionalidades similares.

**Crear Índice O Tabla De Contenido Automática En Word [2024] Guía paso a paso para principiantes**

<https://www.youtube.com/watch?v=att2DYOXG0w>

**Tu FOTO estilo PIXAR, FÁCIL y GRATIS**

<https://www.youtube.com/watch?v=HqRGduar9eM>

Bing: Chatea con IA