Proyectos de repositorios parcial 1

Nombre: Carlos Gustavo Valladares Corral

Profesor: Leonardo Juárez Zucco

Fecha: 6/abril/2025

Índice

Objetivo

El objetivo de este proyecto es explicar la manera de crear, subir, modificar y administrar proyectos tanto locales como remotos utilizando git con un enfoque en el control de versiones para la correcta administración de cambios en un programa controlando las versiones, y el trabajo colaborativo utilizando cualquier servicio derivado de git. Todo esto considerando la importancia de git como una de las maneras mas seguras de trabajar con proyectos de manera local o remota para evitar cualquier problema con los archivos y poder manejar de mejor manera todas las versiones.

También veremos un breve revisión de como funcionan las licencias mas comunes y su importancia, así mismo como lo que son los documentos readme y código de conducta.

Durante este proyecto se presentaran las herramientas básicas de git así como su implementación y el como trabajar en un repositorio remoto y sus ventajas para el trabajo de software colaborativo.

Repositorio local

Un repositorio local de software es un almacenamiento de archivos y datos de un proyecto de software ubicado en la computadora del usuario.

Características principales:

- Contiene el código fuente, archivos de configuración y el historial de versiones.
- Permite trabajar sin conexión a internet.
- Permite recuperar versiones previamente guardadas para regresar a una versión anterior a la actual
- Se gestiona mediante sistemas de control de versiones como Git, Mercurial o Subversion.

El repositorio local es normalmente denominado como “origin” y es la manera mas básica de comenzar a trabajar con repositorios aunque como principal contra, no permite trabajar de manera remota simultanea.

Pasos para crear un repositorio local con git init.

Como preparativos previos antes que nada debemos asegurarnos que tengamos instalado git para ello abriremos la consola de comandos y debemos de revisar si lo tenemos, para ello hay dos comandos el primero seria “git –version” o “where git” en ambos casos deberá de decirte la versión de git o la ruta de instalación

```
C:\Users\Acer Nitro>git --version
git version 2.48.1.windows.1
```

```
C:\Users\Acer Nitro>where git
C:\Program Files\Git\cmd\git.exe
```

Si en ninguno de los casos muestra información o dice que no se a encontrado el programa podemos utilizar el comando “winget install git” para poder instalarlo de manera rápida.

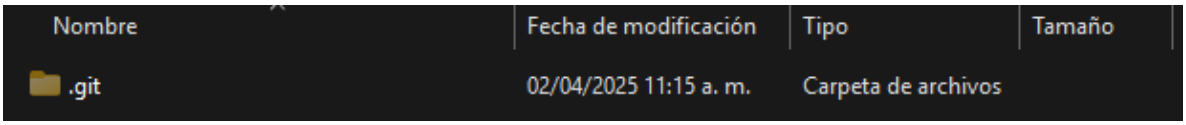
Una vez instalado ya podremos comenzar.

Como paso 1 debemos de colocarnos en la ruta del directorio que queremos convertir en el repositorio, una vez en el directorio debemos de ejecutar el comando git init.

```
D:\ExamenRepo>git init
Initialized empty Git repository in D:/ExamenRepo/.git/
```

Dentro del directorio se creara una carpeta oculta llamada “.git”

El directorio oculto .git/ es la parte más importante de un repositorio Git, ya que contiene toda la información necesaria para que Git funcione correctamente. Dentro de esta carpeta se encuentra el archivo HEAD, que indica en qué rama estás trabajando; el archivo config, con la configuración local del repositorio; y la carpeta objects/, donde Git guarda todos los objetos internos como commits, blobs y árboles que conforman el historial del proyecto. También está la carpeta refs/, que contiene referencias a las ramas y etiquetas, y el archivo index, que actúa como área de preparación antes de hacer un commit. Otros componentes importantes son logs/, que registra los cambios y movimientos dentro del repositorio; hooks/, que permite automatizar tareas con scripts; y archivos como FETCH_HEAD, ORIG_HEAD o COMMIT_EDITMSG, que Git usa durante operaciones como fetch, merges o commits. En conjunto, todo lo que se guarda en .git/ permite que Git pueda rastrear versiones, gestionar ramas y controlar el proyecto de forma eficiente.



Después debemos de agregar los archivos que vamos a hacer un commit para esto usamos el comando con la siguiente estructura “git add (Nombre de los archivos)” en este caso usaremos el . para indicar que añadiremos aboslutamente todo lo que contenga el directorio.

```
D:\ExamenRepo>git add .
```

Después usaremos el comando git commit, para poder crear un nuevo punto en el historial de modificaciones , en este momento solo funcionando en el repositorio local, también añadiremos el -m que indica que definiremos de una vez el mensaje del commit siendo este “Primer commit”

```
D:\ExamenRepo>git commit -m "Primer commit"
[master (root-commit) eebc68d] Primer commit
 1 file changed, 2 insertions(+)
 create mode 100644 Readme.md
```

¿Qué es un repositorio remoto?

Un repositorio remoto es una versión del repositorio de Git que está alojada en un servidor o plataforma en la nube, como GitHub, GitLab o Bitbucket, y que permite compartir el proyecto con otras personas. A diferencia del repositorio local (que está en tu computadora), el repositorio remoto sirve como punto central para colaborar, sincronizar cambios y respaldar el código.

Para crear un repositorio remoto hay muchas maneras, como antes comentaba las empresas privadas que te ofrecen estos son muchas y variadas en este caso vamos a utilizar github pero al final el proceso es muy similar con cada una de ellas.

Como primer paso debes de ya tener una cuenta y haber iniciado sesión en la pagina de github, una vez en la pagina debemos de buscar la pestaña que diga, repositorios y deberás de crear uno nuevo pulsando el botón verde que dice new y deberás darle un nombre, una descripción escoger si será publico o privado(esto iinfluira en que si la gente puede verlo o solo por invitación) si incluirá un archivo readme (archivo que contiene los datos que se deben de considerar por quien sea que use el repositorio a antes de su manipulación) y lo mas importante, las licencias. Una licencia en el contexto de software es un documento legal que establece los permisos y restricciones sobre cómo otras personas pueden usar, modificar, compartir o distribuir un proyecto.

Las mas populares son

- MIT: Es una licencia muy permisiva. Permite usar, copiar, modificar y distribuir el software, incluso en proyectos comerciales. Solo se exige que se mantenga el aviso de copyright y la licencia original. Ideal para compartir libremente.
- GNU (GPL): Es una licencia de tipo copyleft. Permite modificar y distribuir, pero cualquier software derivado también debe ser libre y abierto. Protege la libertad del código, pero obliga a compartir igual.
- Apache 2.0: Similar a la MIT, pero añade protección legal contra patentes. Permite usar el software con libertad, incluso comercialmente, siempre que se conserve la licencia y avisos de derechos.
- Creative Commons (CC): No es para código, sino para contenidos como textos, imágenes o música. Tiene varias versiones (por ejemplo, con o sin fines comerciales, con o sin permiso para modificar). Muy útil para documentación o material educativo.

Es importante saber que tipo de licencia usar al momento de crear tu repositorio para tener en claro como se usara el código del mismo.

En este caso usaremos la licencia del MIT y crearemos el repositorio.

Para conectar nuestro repositorio local al nuevo repositorio en github es tan fácil com o presionar el botón de CODE de ahí copiaremos el link y en la consola de comando utilizaremos el siguiente comando.

```
D:\ExamenRepo>git remote add origin https://github.com/BlackcreatorX/ExamenRepo1.git
```

Eso nos conectara el repositorio online con el repositorio local.

De ahí se debe de hacer un git pull para checar que estemos en la versio n correcta, y después podremos hacer nuestro commit y un git push así podremos subir todo lo que haya en el repositorio a la versión online

En caso de que alguien quisiera copiar nuestro repositorio es tan fácil como hacer un git clone “inserte la liga del repositorio” y así los archivos se añadirán de manera satisfactoria. Eso generara una copia de todos los archivos del repositorio en la ruta seleccionada.

Clonar un repositorio es un proceso sencillo que permite a cualquier persona obtener una copia exacta de todo su contenido, incluyendo su historial. Basta con ejecutar el comando git clone seguido de la URL del repositorio. Esto descargará todos los archivos del proyecto y también su historial completo de versiones, creando una copia local funcional en la carpeta seleccionada. Esta acción es muy útil para colaborar, estudiar el código o simplemente hacer pruebas sin afectar el repositorio original.

Una de las principales ventajas de Git es su historial de commits, ya que permite visualizar cada cambio realizado en el proyecto: quién lo hizo, cuándo y por qué. Esto facilita enormemente la colaboración, depuración de errores y entendimiento de la evolución del código.

Para explorar este historial, se puede utilizar el comando git log, que muestra todos los commits realizados. Además, se pueden añadir opciones para mejorar la visualización, como --oneline, que muestra cada commit en una sola línea de manera compacta, y --graph, que permite visualizar gráficamente las ramas y fusiones del proyecto. Ambas opciones pueden combinarse para obtener una vista más clara y organizada del historial del repositorio.

Git ignore

El archivo .gitignore es un archivo especial que se coloca en la raíz de un repositorio y le indica a Git qué archivos o carpetas debe ignorar, es decir, no rastrear ni incluir en los commits. Esto es especialmente útil para evitar subir archivos temporales, configuraciones locales, credenciales o archivos generados automáticamente que no deben formar parte del control de versiones.

Para crear un archivo .gitignore, basta con agregar un archivo de texto llamado exactamente así (con el punto al inicio) en el directorio raíz del proyecto.

Patrones básicos y avanzados para ignorar archivos

Git permite utilizar patrones para definir qué ignorar. Algunos ejemplos:

*.log → Ignora todos los archivos con extensión .log.

temp/ → Ignora toda la carpeta llamada temp.

*.env → Ignora archivos de entorno, como .env

!important.log → ¡Importante! El signo de exclamación al inicio excluye de ser ignorado, es decir, ese archivo sí será rastreado aunque el patrón general lo ignore.

**/cache/ → Ignora cualquier carpeta llamada cache, sin importar en qué nivel del proyecto esté.

Conclusión

En conclusión git es una tecnología que hoy en día se ha vuelto indispensable para el trabajo colaborativo en tema de desarrollo de software y es importante de conocer ya que al igual que muchas otras herramientas ayudan a hacer mas eficiente todo el trabajo que se realice en equipo y ayuda a prevenir fallos, errores y otro tipo de incidentes que podían afectarnos tanto a nosotros como ingenieros como a nuestros clientes y colaboradores. Un buen manejo de git es el primer paso para el lado oscuro de la fuerza y la conquista de esta galaxia llamada “desarrollo de software”.