

**UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN INGENIERIA PARA LA INDUSTRIA
CON ESPECIALIZACIÓN EN CIENCIAS DE LA COMPUTACIÓN
CURSO: FUNDAMENTOS DE PROGRAMACIÓN Y SCRIPTING**



HOMEWORK 1

**JHONATAN E. SIMON ROMANO
999010164**



Git

GIT es considerado el sistema de versiones mas utilizado y moderno a nivel global. Este software de control de versiones, es utilizado para la eficiencia, la confiabilidad y compatibilidad en el mantenimiento de versiones de aplicaciones. Su finalidad es llevar registros de los cambios en archivos de computadora.

Este sistema fue desarrollado por Linus Torvalds en el 2005, desde esa fecha hasta el día de hoy un gran número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Se puede utilizar desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia.

En la pagina oficial de Git, nos indica que es fácil de aprender, que ocupa poco espacio con un rendimiento ultrarrápido. Supera otras herramientas de control de versiones, como sucursales locales de bajo costo, áreas de preparación convenientes y múltiples flujos de trabajo.

Una de las grandes prioridades de Git es la seguridad al conservar la integridad del código fuente gestionado. Este esta protegido con un algoritmo llamado SHA1. De esta forma se guardan los cambios y garantiza que el historial sea completamente trazable.

Algunas de las ventajas de Git son las siguientes:

- Desarrollo simultaneo
- Versiones más rápidas
- Integración integrada
- Soporte técnico solido de la comunidad
- Funcionalidad con cualquier equipo
- Solicitudes de incorporación de cambios
- Directivas de rama

Algunas de las características más representativas son:

- Apoyo al desarrollo no lineal
- Gestión distribuida
- emulación de servidores CVS
- Autenticación criptográfica de historial
- Gestión eficiente de proyectos grandes
- Realmacenamiento periódico en paquetes
- Compatibilidad con GitHub y Visual Studio Code
- Alta libertad al trabajar en torno a un proyecto.
- Acuerdos de flujo de trabajo



Control de versiones

Una pregunta importante que debemos aprender es: ¿Qué es un control de versiones?, podemos definirlo como: un sistema que registra los cambios hechos en un archivo o un grupo de archivos durante el proceso. Esta finalidad ayuda que en el momento necesario se pueda utilizar una versión determinada.

Dependiendo de la aplicación a la que sea destinada o rama de aplicación de la tecnología un sistema de control de versiones es algo mas que recomendado. Ya que este tipo de sistema, no solo te permite regresar a versiones anteriores de algún archivo en específico sino también al proyecto completo. Teniendo esta ventaja de la trazabilidad en el tiempo se pueda regresar en cualquier momento, si algún error durante el desarrollo fue cometido.

Entre los distintos sistemas de control de versiones que están a disposición son:

Sistemas de control de versiones locales:

En este método se copian los archivos a otro directorio con la finalidad de mantener un historial a medida que se va trabajando en cada versión. Esto logra tener un respaldo de cada archivo en su totalidad, pero es vulnerable debido a que se puede cometer errores al sobre escribir un archivo ya existente.

Sistemas de control de versiones centralizados:

Este sistema se basa en la colaboración de desarrolladores en otros sistemas. La forma principal de este, es que poseen un único servidor que contiene todos los archivos con su correspondiente versión y se descarga los archivos de este lugar central. Se tiene un control detallado de lo que cada colaborador está trabajando.

Este sistema también tiene desventajas, debido a que si falla el servidor centralizado, los colaboradores perderán la información en la que están trabajando. O algún otro error que pueda afectar a cualquier recurso del servidor, se corre un gran riesgo de pérdida de la información.

Sistemas de control de versiones distribuidos:

Estos sistemas fueron hechos con la finalidad de resolver los problemas con los sistemas anteriores, una de sus características es que los clientes pueden descargar la última copia instantánea de los archivos y también se replica completamente el repositorio. Estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, a fin de colaborar al mismo tiempo con distintos grupos de desarrolladores en otras formas dentro del mismo proyecto.



Estados de un archivo

Git cuenta con tres estados principales en los que un archivo podría estar, que son:

Confirmado (committed): significa que los datos están almacenados de manera segura en tu base de datos local.

Modificado (Modified): significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos.

Preparado (Staged): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto lleva a tres secciones principales de un proyecto en Git que son:

El directorio de Git (Git Directory)

Aquí es donde se almacenan los metadatos y la base de datos para el proyecto. Aquí es donde se clona el repositorio desde otra computadora.

Directorio de trabajo (Working Directory)

Es una copia de la versión del proyecto. Los archivos de WD se sacan de la base de datos comprimida en el directorio de Git y se colocan en disco para el uso o modificación.

Área de preparación (Staging Area)

Este es un archivo, que esta generalmente en el directorio de Git, que guarda información sobre lo que vendrá en la siguiente confirmación. En ocasiones se le denomina índice.

Flujo de trabajo en Git

1. Modificar archivos en el directorio de trabajo
2. Preparar archivos, añadirlos en el área de preparación
3. Confirmar los cambios, esto toma los archivos que están en el área de preparación guardando de forma instantánea y permanente en el directorio de Git

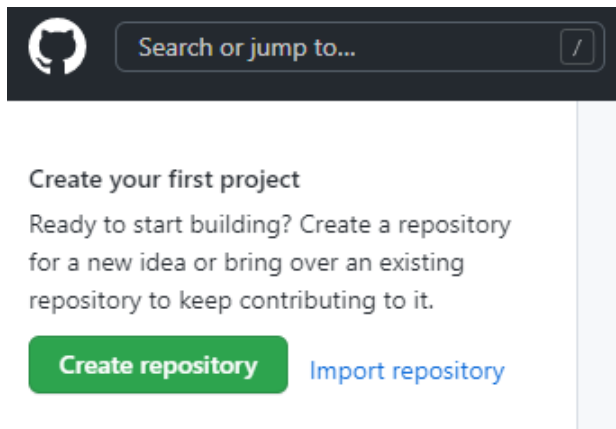


Configuración de un repositorio

Un repositorio de Git es un almacenamiento virtual de un proyecto. Permite guardar versiones del código a las que se puede acceder cuando sea necesario.

Para la creación de un repositorio en GitHub

1. Damos click en Create repository




2. Una nueva ventana se abrirá para setear las configuraciones del nuevo repositorio, agregamos el nombre del repositorio

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *	Repository name *
 jhntnsr ▾	/ Homeworks-R ✓

Great repository names are short and memorable. Need inspiration? How about [fluffy-sniffle](#)?

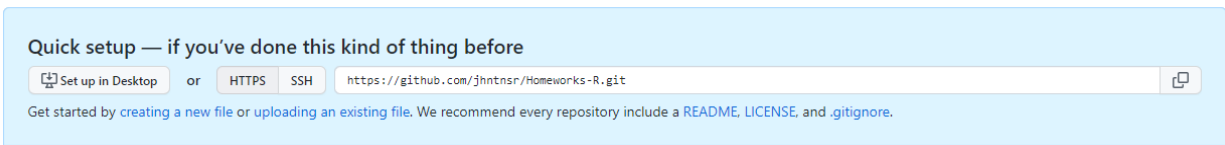
3. Damos click en create repository

 You are creating a public repository in your personal account.

Create repository



4. Ahora ya tenemos el URL con el repositorio



5. También se puede hacer desde la línea de comandos

```
echo "# Homeworks-R" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/jhntnsr/Homeworks-R.git
git push -u origin main
```

6. Se puede dar push a un repositorio ya existente desde la línea de comandos

```
git remote add origin https://github.com/jhntnsr/Homeworks-R.git
git branch -M main
git push -u origin main
```

7. O se puede importar el código desde otro repositorio

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)



Comandos

git help:

Se utiliza para presentarte toda la documentación contenida con Git sobre cualquier comando.

git clone:

Éste crea un nuevo directorio, entra en él y ejecuta git init para que sea un repositorio vacío de Git, añade uno remoto (git remote add) hacia la dirección URL que se le pasa (por defecto llamado origin), ejecuta un git fetch de ese repositorio remoto y después activa el último commit en el directorio de trabajo con git checkout.

git add:

Añade contenido del directorio de trabajo al área de ensayo (staging area o 'index') para la próxima confirmación

git diff:

Se utiliza cuando deseas ver las diferencias entre dos árboles.

git rm:

Se utiliza para eliminar archivos del área de ensayo y el directorio de trabajo para Git

git init:

Esto crea un subdirectorio nuevo llamado .git, el cual contiene todos los archivos necesarios del repositorio.

git clean:

Se utiliza para eliminar archivos no deseados de tu directorio de trabajo.

git fetch:

Descarga los cambios realizados en el repositorio remoto.

git merge <nombre_rama>:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama "nombre_rama".

git pull:

Unifica los comandos fetch y merge en un único comando.

git commit -m "<mensaje>":

Confirma los cambios realizados. El "mensaje" generalmente se usa para asociar al commit una breve descripción de los cambios realizados.

git push origin <nombre_rama>:

Sube la rama "nombre_rama" al servidor remoto.

git status:

Muestra el estado actual de la rama, como los cambios que hay sin commitear.



`git checkout -b <nombre_rama_nueva>:`

Crea una rama a partir de la que te encuentres parado con el nombre “nombre_rama_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

`git checkout -t origin/<nombre_rama>:`

Si existe una rama remota de nombre “nombre_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

`git branch:`

Lista todas las ramas locales.

`git branch -a:`

Lista todas las ramas locales y remotas.

`git branch -d <nombre_rama>:`

Elimina la rama local con el nombre “nombre_rama”.

`git push origin <nombre_rama>:`

Commitea los cambios desde el branch local origin al branch “nombre_rama”.

`git remote prune origin:`

Actualiza tu repositorio remoto en caso de que algún otro desarrollador haya eliminado alguna rama remota.

`git reset --hard HEAD:`

Elimina los cambios realizados que aún no se hayan hecho commit.

`git revert <hash_commit>:`

Revierte el commit realizado, identificado por el “hash_commit”.

`git merge:`

Se utiliza para fusionar uno o más ramas dentro de la rama que tienes activa.

`git log:`

Se utiliza para mostrar la historia registrada alcanzable de un proyecto desde la más reciente instantánea confirmada hacia atrás.

`git tag:`

Se utiliza para dar un marcador permanente a un punto específico en el historial del código fuente.

`git archive:`

Se utiliza para crear un archivo empaquetado de una instantánea específica del proyecto.

`git submodule:`

Se utiliza para gestionar repositorios externos dentro de repositorios normales.