Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Estudios de Postgrado
Maestría en Ingenieria para la Industria
Con Especialización en Ciencias de la Computación
Fundamentos de Programación y Scripting

Percy Matthew Jacobs Orellana

GIT

¿Qué es GIT?

Git es un sistema de control de versiones que nace como un proyecto de código abierto y que está bajo mantenimiento activo, desarrollado por Linus Torvalds, quien tuvo gran responsabilidad en la creación del kernel de Linux. Git cuenta con una arquitectura distribuida, es un ejemplo de DVCS (Sistema de Control de Versiones Distribuido), convirtiendo las copias de trabajo del código de cada desarrollador en un repositorio que puede albergar el historial completo de cambios. Caso contrario a los sistemas de control de versiones antiguos que tenían un único espacio para todo el historial de versiones del software (CVS o Subversion). En este nuevo modelo distribuido, los desarrolladores confirman su trabajo localmente y luego pueden sincronizar su copia con la copia del servidor.

Control de versiones con GIT

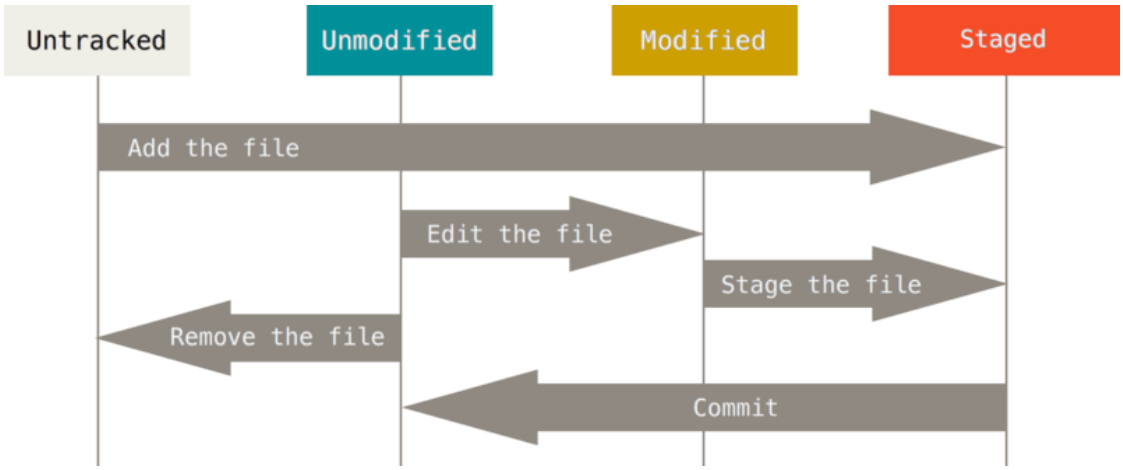
En la actualidad, los proyectos no se desarrollan en línea recta y el control de cambios, bugfixes, features y demás es una necesidad más que un servicio optativo, es por esto que el control de versiones con Git ha tomado tanta popularidad. El manejo de versiones permite separar fases de un mismo proyecto para su desarrollo por separado, teniendo las llamadas *Ramas*, se pueden hacer las modificaciones necesarias para una implementación que aún no saldrá a producción y cuando llegue el momento se podrá hacer un *merge* en el que se unirán las ramas, Git buscará la manera de hacerlo sin conflictos, en el caso que Git encuentre algún conflicto en los archivos dará un aviso. Para este control de versiones existen algunos flujos de trabajo y uno de los más utilizados es el Git-Flow que cuenta de:

- **Master** – rama principal que contiene el repositorio publicado en producción (Debe ser la versión estable).
- **Development** – rama secundaria que sale de Master, funciona como rama de integración en donde se suman todas las nuevas funcionalidades y/o correcciones de errores.

- **Features** – cada nueva funcionalidad se realiza en una rama feature y salen de Development a donde regresan con un merge al terminar el desarrollo.
- **Hotfix** – es donde se arreglan errores que surgen en producción y se publican de manera urgente, salen de Master y regresan a Master.
- **Release** – en esta rama se preparan las nuevas versiones de producción, salen de Development y regresan a Development luego de terminar la preparación para hacer un merge con Master.

Estados de un archivo en GIT

1. **Untracked**: archivos que están únicamente almacenados localmente y no están agregados dentro de Git, sucede cuando agregamos un archivo nuevo en la carpeta del repositorio.
2. **Modified**: archivos que están dentro de Git pero no han sido agregados para su posterior commit, sucede cuando se actualiza un archivo ya existente en el repositorio.
3. **Staged**: en esta etapa ya están agregados y listos para hacer commit y ser subidos al repositorio, pero aún no han sido guardados en el mismo.
4. **Unmodified**: archivos que están dentro de Git y no tienen cambios pendientes.

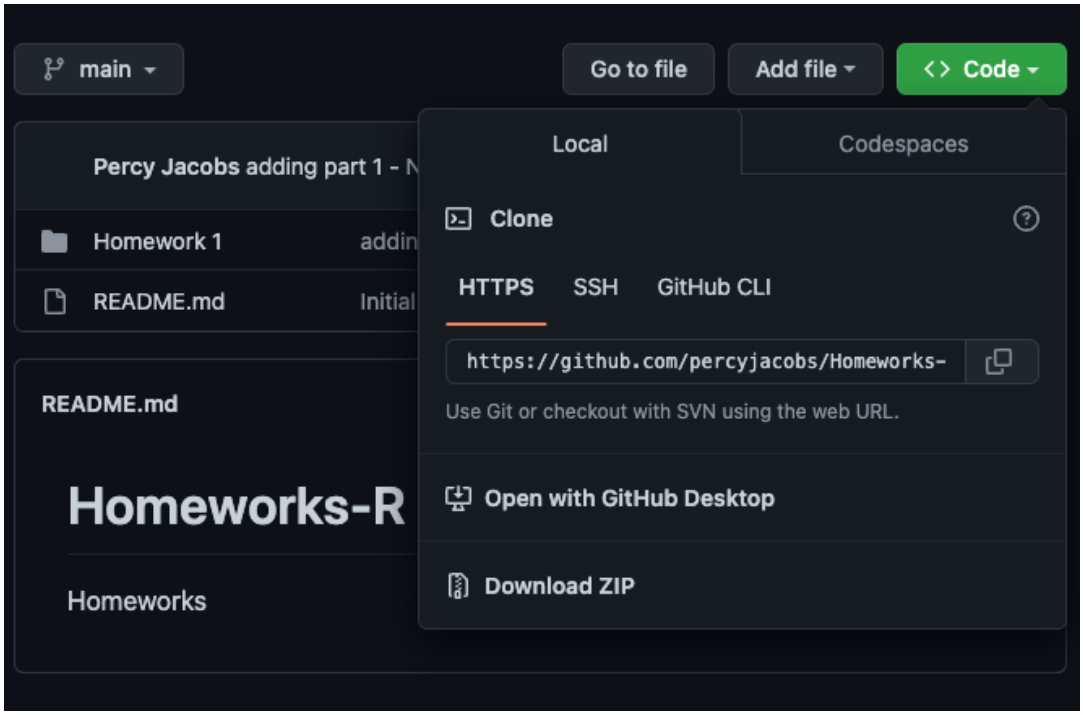


Como se configura un repositorio

Para la creación y configuración de un repositorio se puede realizar de dos maneras, desde la terminal o bien desde la página donde se aloja el repositorio. Al crearlo desde la página podemos tener algunas opciones visuales un tanto más amigable, como agregar el archivo *README*, crearlo de manera pública o privada, darle un nombre y descripción específica.

Una vez creado el repositorio se pueden seguir los siguientes pasos para configurar nuestro entorno local:

1. Copiar el link que provee en la opción de *Code -> Local -> Clone*



2. Colocar la terminal en la carpeta local donde queremos almacenar el repositorio y utilizar el comando *git clone* seguido del link copiado anteriormente.
3. Una vez clonado se podrá tener acceso a la copia local y modificar lo que se desee. Al momento de querer subir los cambios la plataforma (GitHub, GitLab, BitBucket) pedirá las credenciales necesarias para hacer la autenticación.
4. Para que git no solicite las credenciales cada vez que se desee subir cambios se puede configurar tanto el usuario como el email en el equipo.
    a. git config user.name “User Name”
    b. git config user.email email@email.com
5. Dependiendo de la Plataforma que se utilice, se pedirá una contraseña, que por lo general no es la contraseña de la cuenta en esa plataforma. La constraseña que pide generalmente es una *Llave de Autenticación* que se puede crear en la misma plataforma y se puede dar permisos específicos y así controlar que puede hacer cada miembro del equipo.

Comandos en GIT

Algunos de los comandos más comunes y utilizados:

- `git init`

Genera un subdirectorio nuevo llamado `.git` en el que se encuentran todos los archivos necesarios de un repositorio.

- `git fetch`

Descarga los cambios realizados en repositorio remoto.

- `git merge <rama>`

Se una la rama en la que se está actualmente y la que se indica en *<rama>*.

- `git pull`

Unifica los comandos fetch y merge en un único comando.

- `git commit -m “<mensaje>”`

Confirma cambios realizados y se adjunta un mensaje que se asocia al commit para tener claro los cambios realizados.

- `git push origin <rama>`

Sube la rama al servidor remoto.

- `git status`

Muestra el estado actual de la rama.

- `git add <archivo>`

Comienza a seguir el archivo.

- `git checkout -b <rama_nueva>`

Crea una rama a partir de la que se encuentra y salta a la nueva rama.

- `git checkout <rama>`

Salta a la rama que se indica.

- `git branch`

Muestra la rama actual entre todas las ramas locales.

- `git branch -a`

Lista todas las ramas locales y remotas.

- `git branch -d <rama>`

Elimina la rama local indicada.

- `git remote prune origin`

Actualiza repositorio remote 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.

- `git restore <archivo>`

Restaura los cambios realizados en el archivo mencionado.