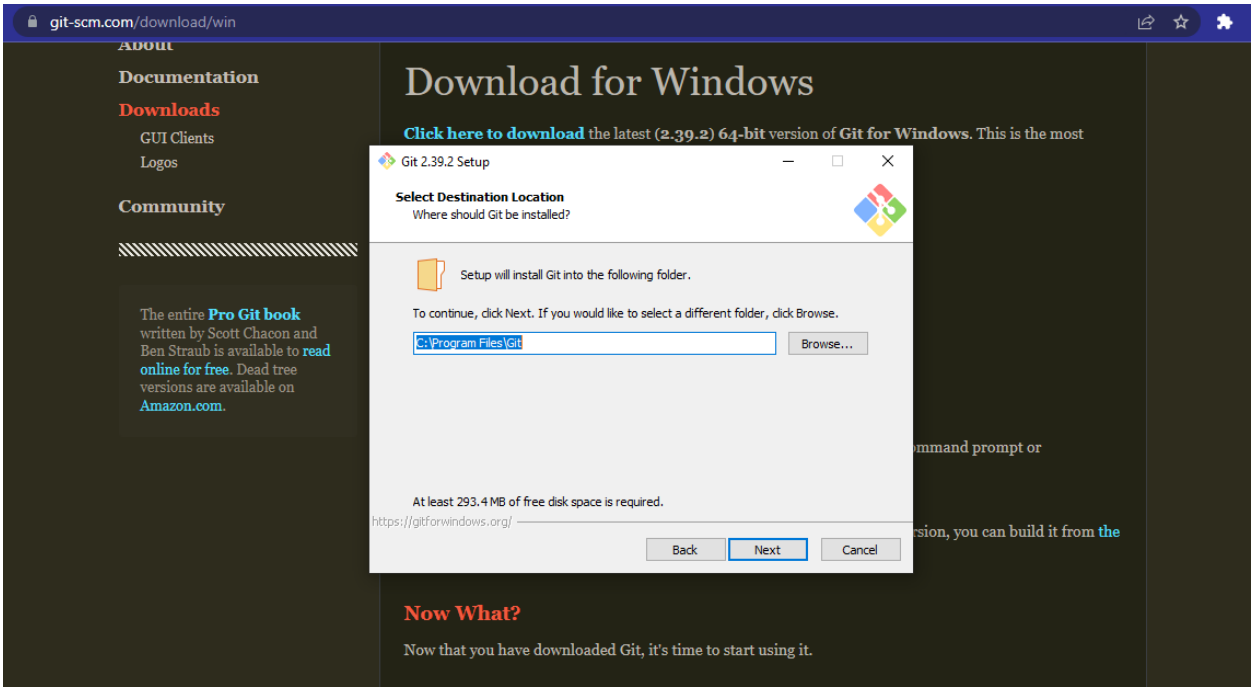
Nombre: Leonel Yucute Tuch

1. **Git**

Es una herramienta de control de versiona miento, con esta se pueden gestionar proyectos principalmente de software, ya que permite el control de versiones y cambios a nivel de código. Además permite la interacción con diversos sistemas para la implementación sobre cualquier ambiente. La principal característica es el trabajo colaborativo entre los equipos de desarrollo que se puede realizar para agilizar los procesos de entrega. Es un proyecto de código abierto y que actualmente permite la descarga e instalación de manera gratuita sobre los sistemas operativos principales.

Esta herramienta permite la descarga de proyectos, desde muchos lugares para que los desarrolladores de un equipo puedan realizar cambios, estos cambios son independientes de la versión original en GIT, luego pueden cargar archivos sin alterar la versión, esto hasta que se realiza una unión con la rama que contiene el proyecto principal.

Ejemplo de instalación y descarga:



2. **Control de versiones con GIT**

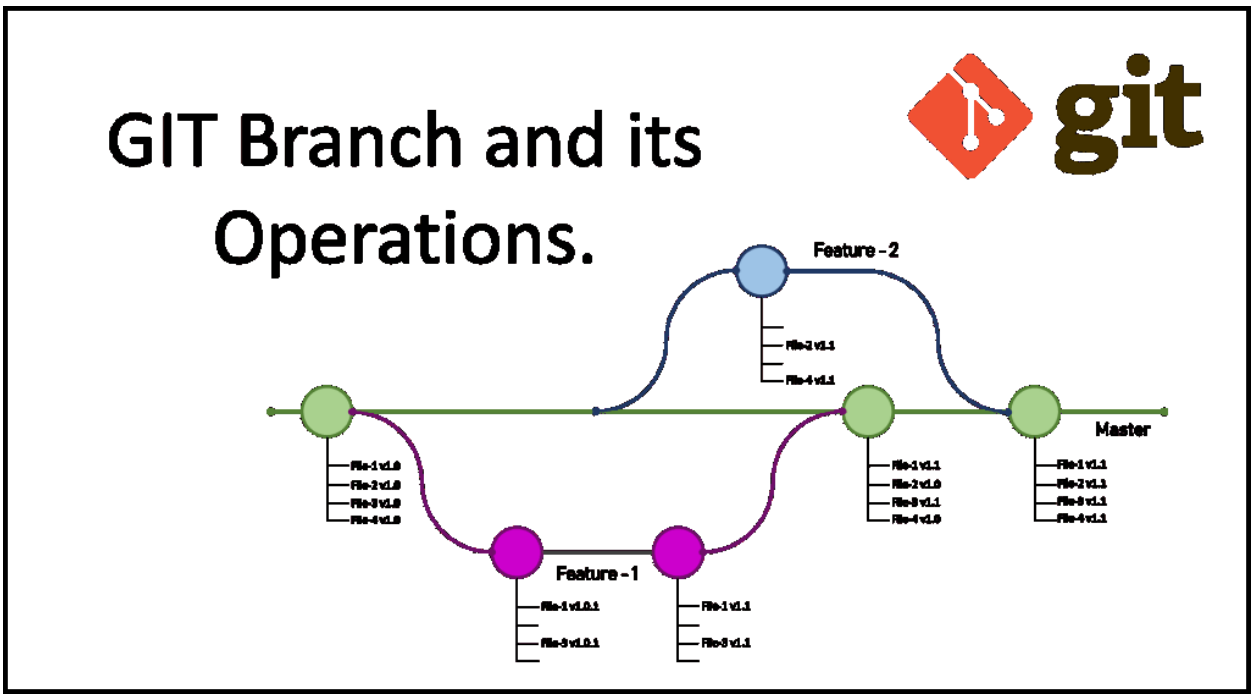
El control de versiones se refiere al control de todos los cambios, es decir Pre y Post de alguna mejora o cambio que se debe realizar sobre un proyecto de Software o sobre algún tipo de documento del cual se quiera tener control sobre una versión anterior del mismo.

Para el caso más común de aplicación de control de versiona miento es el que se utiliza para realizar cambios sobre alguna aplicación que está en producción y del cual no es recomendable bajo ningún escenario aplicar cambios, para esto se utiliza una rama, por ejemplo de Desarrollo donde se genera el cambio, para luego aplicar ese cambio a otra rama de pruebas, donde se certifica que el cambio no genere

errores , para posteriormente aplicar esos cambios en la rama de producción, esto para disminuir los errores sobre transacciones y en producción.

El otro caso común es el diseño y control de cambios que se realizan conforme se diseña un nuevo producto, agregando o modificando funcionalidades.

Ejemplo del control de versiona miento de GIT



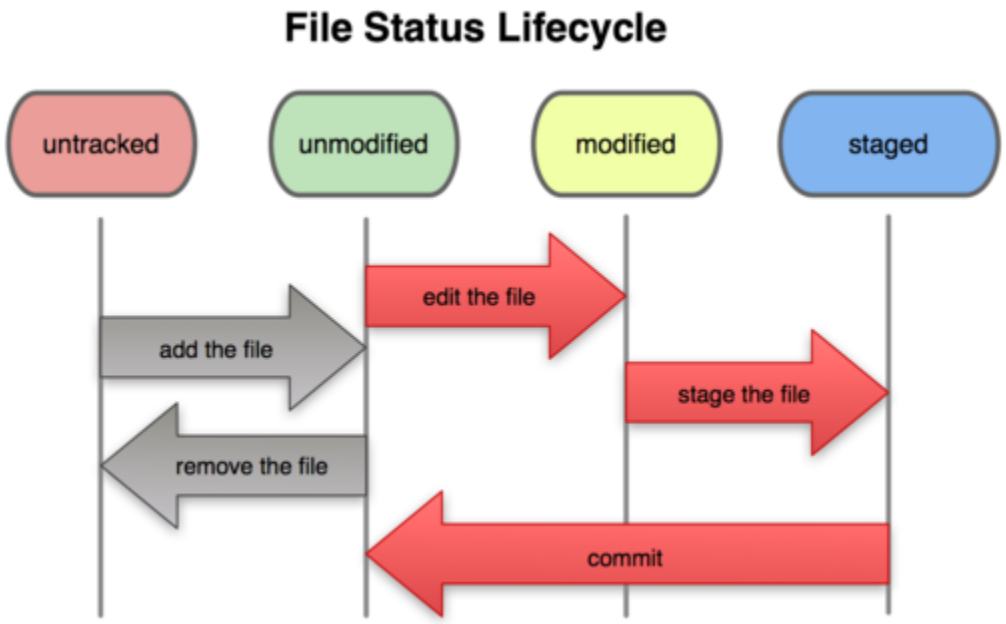
3. Estado de un archivo en GIT

Existen 4 estados de un archivo en GIT:

1. Archivos untracked:
    a. Estos archivos no son partes de GIT, solo están disponibles en el disco duro local donde se generó la instalación de la aplicación GIT.
    b. No son afectados en ningún momento por el comando git add
    c. Para estos archivos GIT no tiene ningún registro de que formen parte de algún proyecto que este ligado a GIT
2. Archivos untracked:
    a. Estos archivos son parte integral de GIT y que forman parte de su sistema de control
    b. En este caso es importante notar que estos archivos, no han sido afectados por el comando git add, este comando en escencia marca como elegible a los archivos que sufrirán modificaciones.

c. Las últimas versiones de estos archivos, solo existen en el disco local de la computadora donde se generó la instalación.

3. Archivos staged:
    a. El termino común para estos archivos es : staging
    b. Estos archivos existen directamente en GIT, su característica principal es que GIT tiene registro de los cambios que ha sufrido por el comando git add.
    c. Cabe resaltar que estos archivos tienen registro ya de los últimos cambios que han sufrido, pero es importante notar que aun no están almacenados en el repositorio, por lo que esto sucede cuando se ejecuta el comando git commit.
4. Archivos Tracked
    a. Estos archivos viven detro del reposito GIT del proyecto y dentro de GIT aplicación.
    b. Estos archivos no tienen cambios pendientes, por lo que el registro del estado es actualizado siempre.
    c. Sobre estos archivos se ejecutaron previamente los comandos git add y git comment



4. Configurar un repositorio en Git

La configuración de un repositorio sucede una vez durante un proyecto, existe un comando con el cual se realiza esta configuración:

Git init

Este comando creara un nuevo subdirectorio .git , esto también creara una rama principal.

Como guardar cambios en el repositorio

Luego de la configuración se procede a ejecutar los comandos para la actualización del repositorio del proyecto:

Pasos:

a. Crear un archivo nuevo, ej: CommitTest.txt
b. Ejecutar el comando: git add para añadir CommitTest.txt al siguiente estado del archivo.
c. Ejecutar el comando: git commit –m “Aquí se puede agregar un mensaje o tag”
d. Con esto se publica el archivo con su versión actualizada en el repositorio.

Comandos GIT

a. Git init

Este comando inicializa un nuevo proyecto para control de versionamiento se realiza al inicio de un proyecto.

b. Git add

Este comando actualiza los archivos que se agregaran al repositorio, esto previo a aplicar commit.

c. Git commit

Este comando actualiza el repositorio con los archivos que se agregaron previamente.

d. Git clone

Este comando genera una copia de un repositorio que puede ser remoto, hacia un destino local.

e. Git push

Este comando se usa para subir o publicar los cambios realizados hacia el repositorio remoto.

f. Git pull

Este comando se utiliza para descargar las actualizaciones de un repositorio remoto, al repositorio local.

g. Git checkout –b [nombre de rama]

Este comando sirve para dirigirse a una rama y si no existe crea una nueva rama, por ejemplo para desarrolladores.

h. Git merge

Este comando se utiliza para actualizar la rama principal, a partir de las versiones de una rama por ejemplo de pruebas.

Ejercicio:

```{r}
a <- runif(25, min= 160 , max = 200)
pesos<-matrix(a, ncol = 5 ,nrow = 5)
colnames(pesos) <- c("ENERO","FEBRERO","MARZO","ABRIL","MAYO")
rownames(pesos) <- c("Rodri" , "Elias" , "Leo" , "Diego","Ochoa")
View(pesos)
meanRow <- rowMeans(pesos,  na.rm = TRUE)
meanCol <- colMeans(pesos, na.rm = TRUE)
pesos <- cbind(pesos, promedioRow = meanRow)
pesos <- rbind(pesos, promedioCol = meanCol)
View(pesos)
```