USO DE GIT

Algoritmos y estructuras de datos I

¿QUÉ ES GIT?

.

Es un software de control de versiones distribuido. Está diseñado pensando en la compatibilidad del mantenimiento de versiones de aplicaciones. Se creó como una solución para gestionar los cambios que se realizan cuando se trabaja sobre el código sin correr el riesgo de sobrescribir archivos, perder documentos en un directorio, tener problemas al incorporar colaboradores al equipo, o depender de un servidor centralizado.

Es uno de los versionadores más utilizados en la actualidad.

Su propósito es llevar registro de los cambios en archivos incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

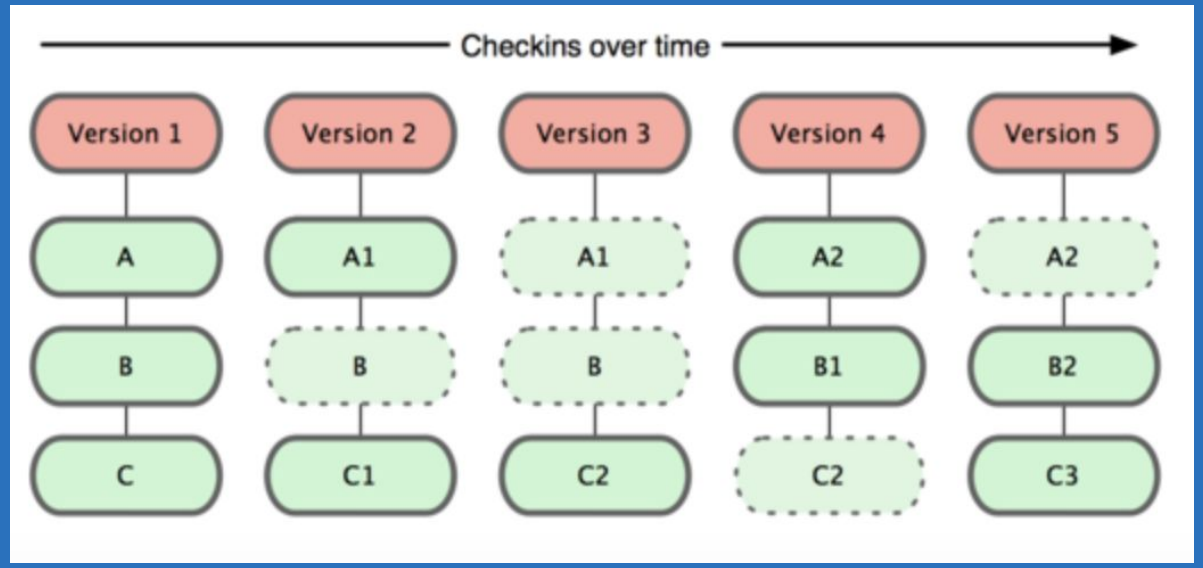
ALMACENAMIENTO DE CAMBIOS

Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que se confirma un cambio, o se guarda el estado del proyecto en Git, lo que realiza es tomar como una foto del aspecto de todos los archivos en ese momento y guarda una referencia a esa copia instantánea.

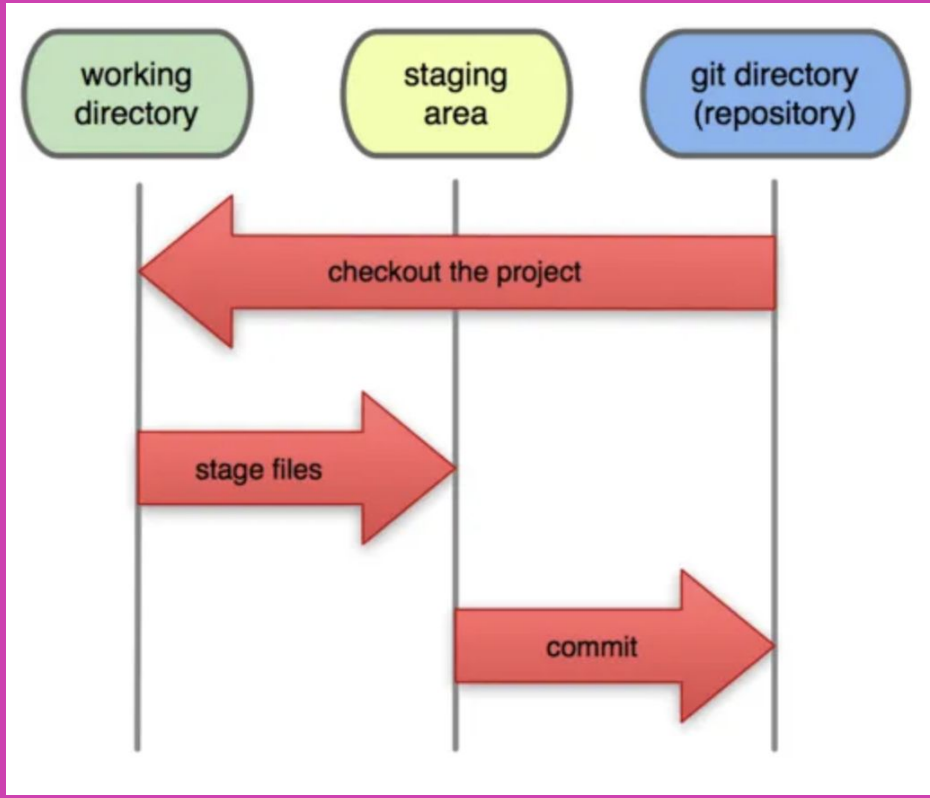
Para ser eficiente, si los archivos no se han modificado Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado.

Es decir, que Git maneja sus datos como una secuencia de copias instantáneas.

Esta es una característica que diferencia a GIT de otros versionadores



ESTADOS DE GIT



Git tiene tres fases principales por donde va a pasar el código a medida que uno vaya trabajando en él.

El flujo de trabajo básico en Git es:

1. Modificar una serie de archivos en el directorio de trabajo (*Working directory*). Aquí es donde se puede hacer cualquier cambio sin modificar el repositorio.

2. Preparar los archivos, añadiéndolos al área de preparación (*Staging área*). En este momento es donde se le puede dar nombre a la nueva versión.

3. Confirmar los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git donde se encuentra el proyecto.

LINEA DE COMANDOS EN GIT

Git nació como una herramienta para utilizar desde la línea de comandos (o terminal). Desde la línea de comandos GIT permite ejecutar varios comandos disponibles para realizar diferentes acciones que se dividen en las siguientes áreas:

1. Iniciar un área de trabajo.
2. Trabajar en los cambios actuales
3. Revisar el historial de cambios y el estado.
4. Editar el historial
5. Colaborar.

Con el tiempo, se han creado varias aplicaciones gráficas dedicadas que funcionan como una interfaz visual para Git. Estas aplicaciones permiten emitir casi todos los diversos comandos de una forma más visual.

Muchos IDE y editores diseñados para desarrolladores ofrecen acceso rápido a los comandos principales de git.

COMANDOS BÁSICOS

Instalación y Configuración

- Para instalar Git en la computadora se puede hacer visitando el sitio web oficial de Git y descargando la versión adecuada para el sistema operativo.

  https://git-scm.com/downloads

- Una vez instalado se debe proceder a la configuración del nombre de usuario y dirección de correo electrónico. Para ello se utiliza el comando "git config". Estos detalles se asociarán cada vez que dicho usuario suba un cambio en el repositorio.

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@example.com"
```

COMANDOS BÁSICOS

Crear un Repositorio:

```
cd /ruta/a/tu/proyecto
git init
```

Clonar un Repositorio:

```
git clone URL_DEL_REPOSITORIO
```

Creación de Repositorios:

Para crear un nuevo repositorio hay que navegar hasta el directorio donde está alojado el proyecto y luego aplicar el comando *"git init"*.

Si se necesita trabajar en un proyecto existente que está alojado en una plataforma como GitHub, puedes clonar el repositorio en tu máquina local usando el comando *"git clone"*. Este comando requiere la URL del repositorio.

COMANDOS BÁSICOS

Manejo de Cambios:

Una vez que se haya realizado algún cambio en el proyecto, habrá que preparar esos cambios para que queden en la staging área, para luego enviarlos al repositorio remoto del proyecto. El comando “git add” permite preparar los cambios.

Luego de preparar los cambios, se pueden confirmar los mismos en el repositorio con el comando “git commit". Este comando requiere un mensaje que describa los cambios realizados.

Como último paso, se deben enviar estos cambios al repositorio remoto utilizando el comando “git push”

```
git add
```

```
git commit -m “Un mensaje de commit descriptivo”
```

```
git push origin master
```

COMANDOS BÁSICOS

```
git log
```

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   001-working-locally/02-save-changes.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   001-working-locally/02-save-changes.md
        modified:   001-working-locally/03-query-status.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    preview.html
```

Historial de cambios:

Si se está trabajando en equipo y otra persona subió cambios al proyecto remoto, se puede recibir los cambios realizados en el proyecto local mediante el uso del comando “git pull origin master”

En el caso de que se quiera ver el historial de cambios realizados se puede utilizar el comando “git log”. Por esto es importante escribir mensajes claros y concisos que describan con precisión el cambio realizado, cuando uno hace un commit.

Otro de los comandos útiles es el comando git status, el cual permite enumerar los archivos que se han preparado, los que están sin preparar y los archivos sin seguimiento.

Recordar la secuencia de los cambios: git add para prepararlos, git commit para confirmarlos y git push para subirlos al repositorio remoto.

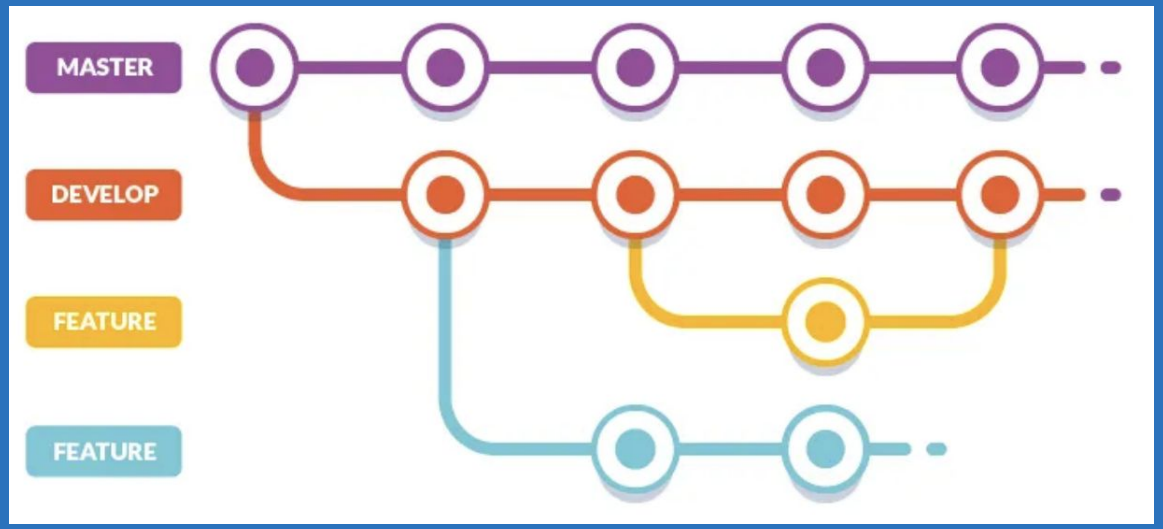
BRANCHES EN GIT

En general, una rama de desarrollo (“Git Branch”) es una bifurcación del estado del código que crea un nuevo camino para la evolución del mismo. Puede ir en paralelo a otras Git Branch que se pueden generar.Esto hace posible incorporar nuevas funcionalidades al código de manera ordenada y precisa.

Ventajas:

- Hace posible desarrollar nuevas funciones para una aplicación sin obstaculizar el desarrollo de la rama principal.

- Es posible crear diferentes ramas de desarrollo que pueden converger en el mismo repositorio.Por ejemplo tener ramas como producción, test y desarrollo.



Así es como se ve una ramificación. Es como un árbol del que se desglosan varias ramas en diferentes direcciones y en algún punto se unen para formar una sola. Los puntos que se ven en la imagen son los commit que se hacen en cada rama.

BRANCHES EN GIT

Con el comando *"git branch"* se puede visualizar todas las ramas que se encuentran activas.

Con el comando *"git branch mi_primera_rama"* git crear una rama en tu repositorio local (es decir en tu computadora). Luego si quieres trabajar en ella deberás posicionarte allí con el uso del comando "git checkout mi_primera_rama"

Con el comando *"git push origin mi_primera_rama"* se puede subir la rama que creamos en el repositorio local al repositorio remoto.

```
killmer@hp MINGW64 ~/Desktop/pruebagit (master)
$ git branch
* master
```

```
killmer@hp MINGW64 ~/Desktop/pruebagit (master)
$ git checkout mi_primera_rama
Switched to branch 'mi_primera_rama'
M       test.txt

killmer@hp MINGW64 ~/Desktop/pruebagit (mi_primera_rama)
$ git branch
  master
* mi_primera_rama
```

Según se observa en los ejemplo. La rama marcada en otro coló junto con el *, es la rama en la cual estamos posicionados trabajando.

BRANCHES EN GIT

Fusión (o merge) de ramas:

Fusionar es combinar los cambios de una rama en otra. Esto es útil cuando se ha desarrollado una funcionalidad en una rama separada y se desea integrar esos cambios en la rama principal (como main o master).

Imaginemos que tenemos dos ramas: main y feature. La rama feature contiene cambios que dejamos fusionar en main.

Los pasos para fusionar serían los siguientes

1. Posicionarse en la rama de destino
   Primero, cambia a la rama en la que deseas aplicar los cambios (por lo general, main).

```sh
git checkout main
```

2. Ejecutar el comando de fusión
   Luego, ejecuta el comando git merge seguido del nombre de la rama que contiene los cambios (en este caso, feature).

```sh
git merge feature
```

Resultado de la Fusión

1. Si no hay conflictos, Git combinará automáticamente los cambios y creará un nuevo commit en la rama main que integra los cambios de feature.
2. Si hay conflictos, Git te avisará y te pedirá que resuelvas los conflictos manualmente.

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.