



INSTITUTO TECNOLÓGICO DE LAS AMÉRICAS (ITLA)

Participante

Jeury Alexander Quezada Báez (2022-0621)

Maestro/a

Kelyn Tejeda Belliard

Materia

Programacion III

Fecha de entrega

29/07/2023

1- ¿Qué es Git?

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.



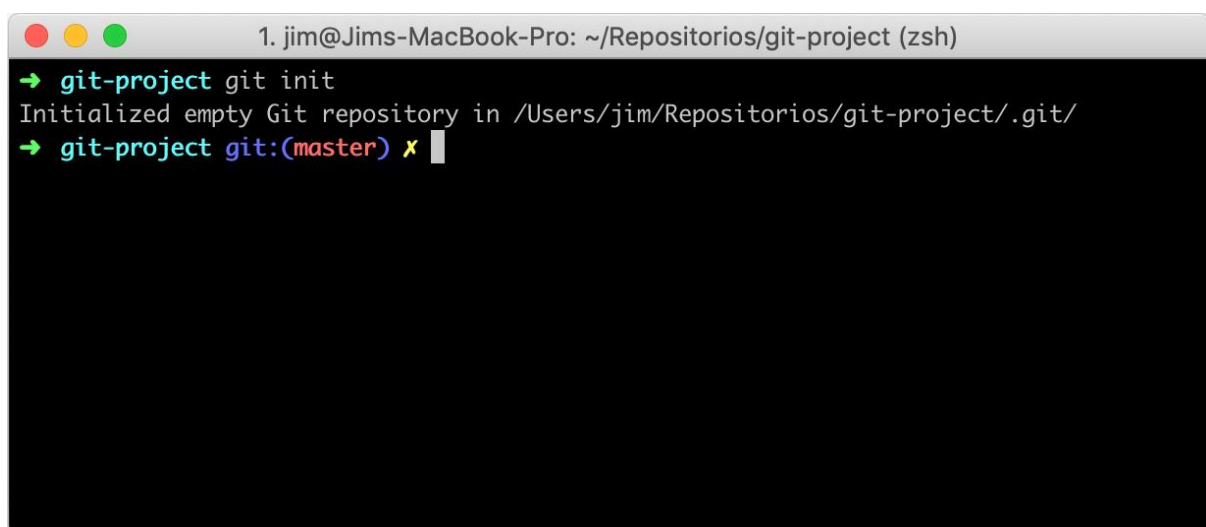
Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

2- ¿Para que funciona el comando Git init?

El comando `git init` crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

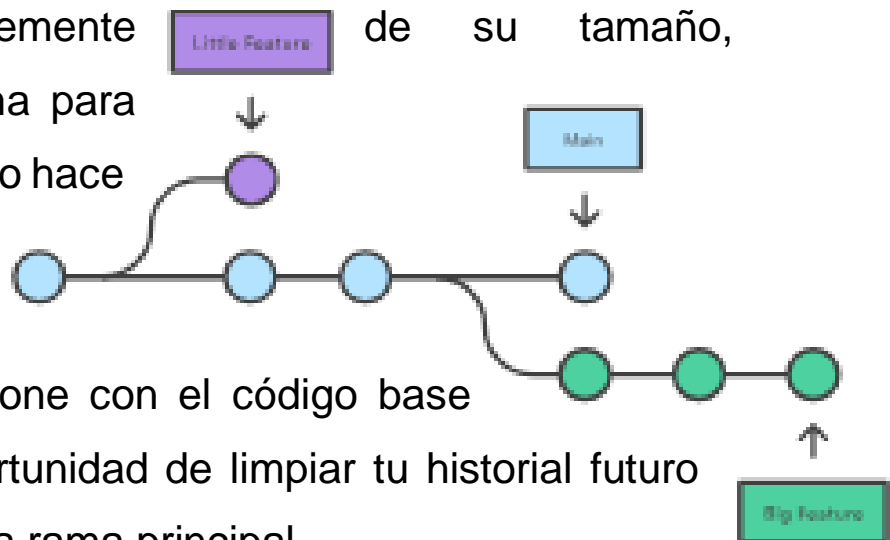
A screenshot of a terminal window on a Mac. The title bar shows the window name '1. jim@Jims-MacBook-Pro: ~/Repositorios/git-project (zsh)'. The terminal content shows the command 'git init' being executed, followed by the output 'Initialized empty Git repository in /Users/jim/Repositorios/git-project/.git/'. The prompt then shows 'git:(master)' with a cursor. The terminal has a dark background with light-colored text.

```
1. jim@Jims-MacBook-Pro: ~/Repositorios/git-project (zsh)
→ git-project git init
Initialized empty Git repository in /Users/jim/Repositorios/git-project/.git/
→ git-project git:(master) x
```

Al ejecutar `git init`, se crea un subdirectorio de `.git` en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo `HEAD` que apunta a la confirmación actualmente extraída.

3- ¿Que es una rama?

En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para las instantáneas de tus cambios. Cuando quieres añadir una nueva función o solucionar un error, independientemente de su tamaño, generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código inestable se fusione con el código base principal, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal.



El diagrama anterior representa un repositorio con dos líneas de desarrollo aisladas, una para una función pequeña y otra para una función más extensa. Al desarrollarlas en ramas, no solo es posible trabajar con las dos de forma paralela, sino que también se evita que el código dudoso se fusione con la rama main.

4- ¿Como saber es que rama estoy?

mediante un apuntador especial denominado HEAD. Aunque es preciso comentar que este HEAD es totalmente distinto al concepto de HEAD en otros sistemas de control de cambios como Subversion o CVS. En Git, es simplemente el apuntador a la rama local en la que tú estés en ese momento, en este caso la rama master; pues el comando `git branch` solamente crea una nueva rama, pero no salta a dicha rama.

5- ¿Quién creo git?

Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005.

El kernel de Linux es un proyecto de software de código abierto con un alcance bastante amplio. Durante la mayor parte del mantenimiento del kernel de Linux (1991-2002), los cambios en el software se realizaban a través de parches y archivos. En el 2002, el proyecto del kernel de Linux empezó a usar un DVCS propietario llamado BitKeeper.

En el 2005, la relación entre la comunidad que desarrollaba el kernel de Linux y la compañía que desarrollaba BitKeeper se vino abajo y la herramienta dejó de ser ofrecida de manera gratuita. Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron mientras usaban BitKeeper. Algunos de los



objetivos del nuevo sistema fueron los siguientes:

- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el kernel de Linux) eficientemente (velocidad y tamaño de los datos)

Desde su nacimiento en el 2005, Git ha evolucionado y madurado para ser fácil de usar y conservar sus características iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal

6- Cuales son los comandos más esenciales de Git

-Git clone

Git clone es un comando para descargar el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente hace una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en su computadora.

-Git branch

Las ramas son muy importantes en el mundo de git. Mediante el uso de ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crear, enumerar y eliminar ramas.

-Git checkout

Este es también uno de los comandos Git más utilizados. Para trabajar en una rama, primero debe cambiarse a ella. Usamos git checkout principalmente para cambiar de una rama a otra. También podemos usarlo para verificar archivos y confirmaciones.

-Git status

El comando de estado de Git nos brinda toda la información necesaria sobre la rama actual.

-Git add

Cuando creamos, modificamos o eliminamos un archivo, estos cambios ocurrirán en nuestro local y no se incluirán en la próxima confirmación (a menos que cambiemos las configuraciones).

Entre otros...

7- ¿Que es git Flow?

Gitflow es un modelo alternativo de creación de ramas en Git en el que se utilizan ramas de función y varias ramas principales. Fue Vincent Driessen en nvie quien lo publicó por primera vez y quien lo popularizó. En comparación con el desarrollo basado en troncos, Gitflow tiene diversas ramas de más duración y mayores confirmaciones. Según este modelo, los desarrolladores crean una rama de función y retrasan su fusión con la rama principal del tronco hasta que la función está completa.

Estas ramas de función de larga duración requieren más colaboración para la fusión y tienen mayor riesgo de desviarse de la rama troncal. También pueden introducir actualizaciones conflictivas.

Gitflow puede utilizarse en proyectos que tienen un ciclo de publicación programado, así como para la práctica recomendada de DevOps de entrega continua. Este flujo de trabajo no añade ningún concepto o comando nuevo, aparte de los que se necesitan para el flujo de trabajo de ramas de función. Lo que hace es asignar funciones muy específicas a las distintas ramas y definir cómo y cuándo deben estas interactuar. Además de las ramas de función, utiliza ramas individuales para preparar, mantener y registrar publicaciones. Por supuesto, también puedes aprovechar todas las ventajas que aporta el flujo de trabajo de ramas de función: solicitudes de incorporación de cambios, experimentos aislados y una colaboración más eficaz.

8- Que es trunk based development?

Qué es Trunk Based Development

El desarrollo basado en tronco es una práctica de gestión de control de versiones en la que los desarrolladores fusionan pequeñas actualizaciones de forma frecuente en un “tronco” o rama principal (main).

Dado que esta práctica simplifica las fases de fusión e integración, ayuda a lograr la CI/CD (Continuous Integration / Continuous Deployment, siendo estos la Integración Continua y Despliegue Continuo) y, al mismo tiempo, aumenta la entrega de software y el rendimiento de la organización.

A medida que los sistemas de control de versiones se desarrollaron, surgieron varios estilos de desarrollo que permitieron a los programadores encontrar errores con más facilidad, crear código en paralelo con sus compañeros y acelerar el ritmo de publicación. Hoy en día, la mayoría de los programadores aprovechan uno de estos dos modelos de desarrollo para ofrecer software de calidad: Git Flow y desarrollo basado en troncos (Trunk Based Development).

Git Flow, que se popularizó primero, es un modelo de desarrollo más estricto en el que solo determinadas personas pueden aprobar los cambios en el código principal. Así se mantiene la calidad del código y se minimiza el número de errores. El desarrollo basado en troncos es un modelo más abierto, ya que todos los desarrolladores tienen acceso al código principal, lo que permite a los equipos iterar con rapidez y pone en práctica la CI y la CD. Sobre este último flujo, ya veremos más adelante cómo funciona a nivel teórico-práctico.

Trunk Based Development

El desarrollo basado en troncos es otro modelo de trabajo, en este caso, más eficiente para DevOps, ya que permite agilizar el ciclo

