
## Presentación

Nombre;

Jeremy Polanco Lara

## Matricula:

2021-2198

## Tema

Tarea 3

## Sección/Grupo

5

## Facilitador/a

Kelyn Tejada Belliard.

## Fecha:

01/04/2023

## Santo Domingo Este

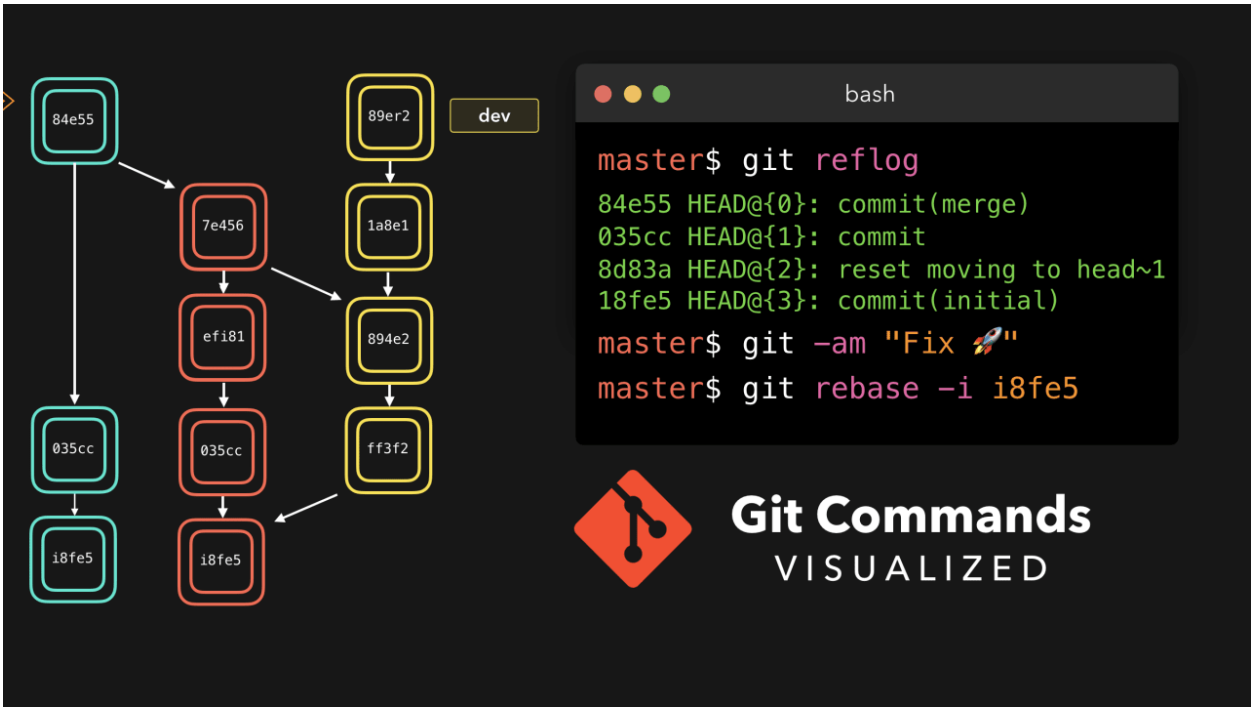
# Introducción

En la actualidad, el trabajo colaborativo y el control de versiones son esenciales en cualquier proyecto de desarrollo de software. Git es una herramienta de control de versiones distribuida que permite a los equipos de trabajo mantener un registro de los cambios en el código fuente, coordinar y compartir el trabajo de manera efectiva. En esta tarea, exploraremos los fundamentos de Git y cómo se utiliza en la práctica. También analizaremos los principales conceptos y comandos de Git para entender su funcionamiento y aplicaciones, así como algunos casos de uso en la industria. Al finalizar esta tarea, los estudiantes estarán en capacidad de manejar proyectos de software utilizando Git, lo que les permitirá colaborar en equipo de manera efectiva y mantener un registro de los cambios en su código fuente.

# 1- Desarrolla el siguiente Cuestionario

## 1- ¿Que es Git?

Git es el sistema de control de versiones más utilizado en la actualidad. Este software realiza un seguimiento de todos los cambios que se realizan en los archivos, lo que le permite tener un registro completo de todo lo que se ha hecho. En caso de ser necesario, Git puede volver a versiones específicas de los archivos. Además, Git es una herramienta muy útil para la colaboración ya que permite que varias personas realicen cambios en un archivo y luego se fusionen en una sola fuente, lo que facilita enormemente el trabajo en equipo.




Git es un sistema de control de versiones distribuido que se ejecuta en su computadora local y almacena los archivos y su historial en su disco duro. Además, también puede utilizar servicios de alojamiento en línea como GitHub o Gitlab para almacenar una copia de los archivos y su historial de revisiones. Esta opción proporciona un lugar centralizado donde puede cargar sus cambios y descargar los cambios de otros colaboradores, lo que facilita la colaboración en proyectos de desarrollo de software.

Una de las principales ventajas de Git es que puede fusionar automáticamente los cambios realizados por diferentes colaboradores. Por lo tanto, dos o más personas pueden trabajar en diferentes partes del mismo archivo y luego fusionar esos cambios sin perder el trabajo de los demás. Esto reduce los conflictos y simplifica el proceso de colaboración.

## 2- ¿Para que funciona el comando Git init?

El comando git init es utilizado para crear un nuevo repositorio Git en el cual se puede trabajar. Este comando puede ser utilizado para convertir un proyecto no versionado existente en un repositorio de Git o para inicializar un nuevo repositorio vacío. Es el primer comando que se debe ejecutar en un nuevo proyecto, ya que la mayoría de los otros comandos de Git no estarán disponibles fuera de un repositorio inicializado.

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

Es importante mencionar que, aparte del directorio .git, el proyecto existente permanece inalterado en el directorio raíz del proyecto. A diferencia de SVN, Git no requiere un subdirectorio .git en cada subdirectorio. En resumen, el comando git init es esencial para iniciar un nuevo proyecto.

### git init --bare

cuando se desea crear un repositorio compartido, se recomienda utilizar la opción `--bare` para omitir el directorio de trabajo.

Un repositorio compartido se crea típicamente utilizando la opción `–bare` de la siguiente manera:

### ```git init –bare example.git```

La convención es que los repositorios iniciados con la opción --bare deben tener el sufijo ".git" en su nombre. Por ejemplo, el nombre de un repositorio llamado "my-project" creado con la opción --bare debe ser " example.git"

La opción --bare establece el repositorio como un repositorio Git sin un directorio de trabajo. En cambio, todos los archivos y los metadatos del repositorio se almacenan en el directorio ".git". Este tipo de repositorio es ideal para la colaboración en equipo, ya que no se utiliza un directorio de trabajo local para trabajar directamente en los archivos del repositorio, sino que se clona el repositorio en un directorio de trabajo local para trabajar con los archivos.

### git init templates

Una forma de inicializar un nuevo repositorio de Git es mediante la copia de archivos desde otro repositorio y la configuración de una plantilla para el subdirectorio .git. Esta plantilla puede incluir directorios y archivos predeterminados que se copiarán en el nuevo repositorio.

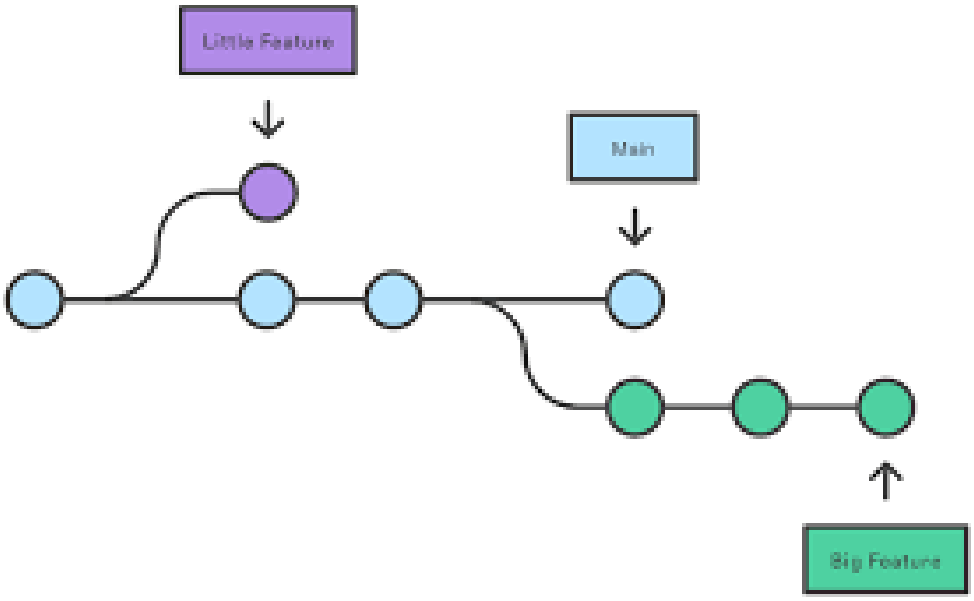
Por defecto, Git tiene plantillas predeterminadas que se encuentran en el directorio /usr/share/git-core/templates, pero también se pueden establecer rutas diferentes en cada máquina. Estas plantillas predeterminadas son una buena referencia y ejemplo para utilizar las funciones de la plantilla.

Una característica poderosa de las plantillas es la configuración de Git Hooks. Puede crear una plantilla con ganchos de Git predefinidos e inicializar sus nuevos repositorios de Git

## 3- ¿Que es una rama?

Una rama en Git representa una línea independiente de desarrollo. Las ramas son una abstracción del proceso de edición, preparación y confirmación de cambios en el código. Podría decirse que son una forma de solicitar un directorio de trabajo, un área de preparación y un historial de proyectos completamente nuevos. Cada nueva confirmación de cambios se registra en el historial de la rama actual, lo que resulta en una bifurcación en la historia del proyecto.

El comando git branch permite crear, listar, renombrar y eliminar ramas en Git. Sin embargo, este comando no permite cambiar entre ramas o volver a unir historiales bifurcados. Para realizar estas tareas, Git tiene integrados los comandos git checkout y git merge.




Las opciones más utilizadas de git branch son:

**git branch**: Te da una lista de todas las ramas en tu repositorio.

**git branch <branch>:** Crea una nueva rama con nombre pasado

**git branch -d <branch>:** Elimina la rama especificada, pero no te deja hacerlo si tiene cambios sin hacer unir

**git branch -D <branch>:** Elimina la rama especificada sin importar que.

**git branch -m <branch>:** Renombra la rama actual en el valor pasado como <branch>

## 4- ¿Como saber es que rama estoy?

En git existen varias maneras en las que podemos saber en que rama nos encontramos actualmente, algunas de ellas son:

**git branch --show-current** en la version Git 2.22 en adelante

**git symbolic-ref --short HEAD**

**git rev-parse --abbrev-ref HEAD**

**git name-rev --name-only HEAD**

Todos estos comandos nos devuelven el nombre de la rama del repositorio en el que nos encontramos.

## 5- ¿Quién creo git?

El desarrollo de Git comenzó en abril de 2005 después de que los desarrolladores del kernel de Linux perdieran el acceso al sistema de gestión de control de fuente patentado que estaban usando, llamado BitKeeper.

Linus Torvalds, el creador de Linux, quería un sistema distribuido como BitKeeper, pero ninguno de los sistemas gratuitos disponibles satisfacía sus necesidades. Se dispuso a escribir su propio sistema de control de versiones, especificando criterios que eliminaron todos los sistemas en uso en ese momento. Torvalds anunció el proyecto el 6 de abril de 2005 y se convirtió en autohospedador al día siguiente.

La primera fusión de múltiples ramas tuvo lugar el 18 de abril y, para el 29 de abril, Git fue evaluado comparando parches de grabación en el árbol del kernel de Linux a una velocidad de 6,7 parches por segundo.

Torvalds logró sus objetivos de rendimiento y Git administró el lanzamiento del kernel 2.6.12 el 16 de junio de 2005. Torvalds entregó el mantenimiento a Junio Hamano el 26 de julio de 2005 y Hamano fue responsable del lanzamiento 1.0 el 21 de diciembre de 2005. Git es distribuido bajo la licencia GPL-2.0 y es un software gratuito y de código abierto.

El nombre de Git fue elegido por el mismo Linus Torvalds, quien tiene reputación por su sarcástico sentido del humor. Git es la jerga del inglés británico para una persona desagradable o despreciable, y Torvalds bromeó diciendo que nombró a todos sus proyectos con su nombre, como Linux. En la página del manual, Git se describe como "el estúpido rastreador de contenido", y el archivo Léame del código fuente explica que "git" puede significar cualquier cosa, según el estado de ánimo de cada uno.

## 6- ¿Cuáles son los comandos más esenciales de Git?


Estos son de los comandos más comunes y esenciales en Git:

**git init**: este comando inicializa un repositorio Git vacío en el directorio de trabajo actual.

**git add**: este comando agrega un archivo o directorio al área de preparación, listo para ser confirmado.

**git commit**: este comando crea una nueva confirmación con los cambios en el área de ensayo.

**git status**: este comando muestra el estado del repositorio, incluidos los archivos que se han modificado, agregado o eliminado.

**git log**: este comando muestra el historial de confirmaciones del repositorio.

**git branch**: Este comando muestra las ramas en el repositorio.

**git checkout:** este comando cambia entre ramas o restaura archivos de una confirmación específica.

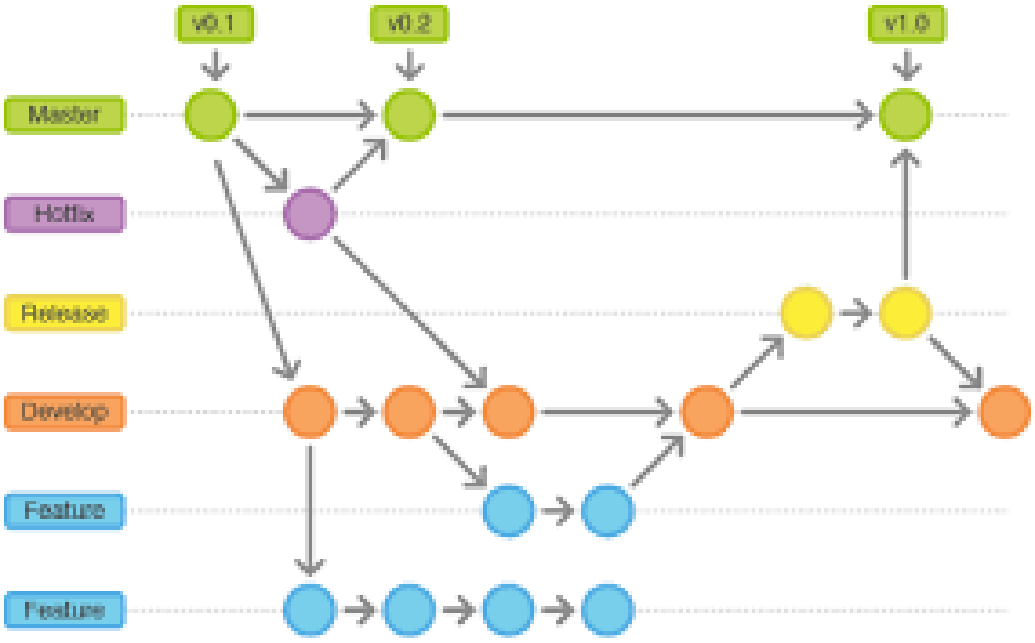
**git merge**: este comando fusiona una rama con otra.

**git pull**: este comando obtiene y fusiona los cambios de un repositorio remoto.

**git push:** este comando envía los cambios a un repositorio remoto.

Estos comandos proporcionan la funcionalidad básica necesaria para administrar un repositorio de Git. También hay muchos más comandos y opciones avanzados disponibles.

## 7- ¿Qué es git Flow?




El modelo de Git Flow es una metodología de bifurcación muy utilizada en Git, que fue presentada por el desarrollador de software Vincent Driessen en 2010 para simplificar la gestión de versiones. En esencia, el flujo de Git implica la segmentación del trabajo en diferentes tipos de ramas de Git. Este artículo detallará las distintas ramas utilizadas en el modelo de Git Flow, cómo se puede aplicar en el cliente GitKraken, y además explorará de manera concisa otras dos metodologías de bifurcación de Git: GitHub Flow y GitLab Flow.

En git Flow, hay 5 tipos de ramas distintas:

**Main**: Esta rama principal, también conocida como master, es donde se encuentra el código de producción estable y listo para usar.

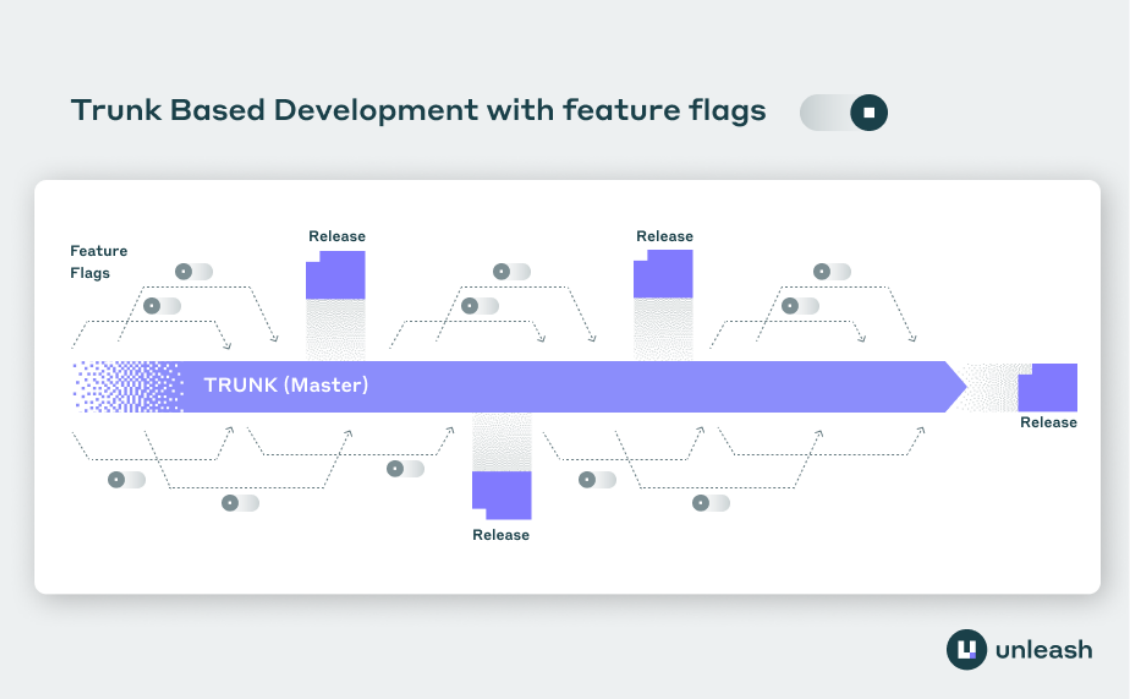
**Develop**: Esta rama se utiliza para integrar el trabajo de varios desarrolladores y para realizar pruebas de integración. Es donde se fusionan todas las ramas de características.

**Feature**: Las ramas de características se crean a partir de la rama develop y se utilizan para desarrollar nuevas características o funcionalidades. Una vez que se completa una característica, se fusiona de nuevo en la rama develop.

**Release**: Las ramas de lanzamiento se crean a partir de la rama develop y se utilizan para preparar el código para un lanzamiento. Se pueden realizar pruebas finales y correcciones de errores antes de fusionar la rama de lanzamiento en la rama main.

**Hotfix**: Las ramas de hotfix se crean a partir de la rama main y se utilizan para corregir errores críticos en el código de producción. Una vez que se corrige el error, la rama de hotfix se fusiona tanto en la rama main como en la rama develop.

## 8- ¿Qué es trunk based development?




El desarrollo basado en troncos (TBD) es un enfoque de ramificación utilizado en el desarrollo de software, donde los desarrolladores fusionan cada nueva característica, corrección de errores o cambio de código en una rama central en el sistema de control de versiones. Esta rama se conoce como "troncal", "línea principal" o, en Git, como la "rama maestra".

TBD permite la integración continua, lo que a su vez conduce a la entrega continua, al crear un entorno en el que los desarrolladores realizan compromisos con la rama principal varias veces al día de forma natural. Esto facilita el cumplimiento del requisito de integración continua que establece que "todos los miembros del equipo de desarrollo deben comprometerse con la rama principal al menos una vez cada 24 horas". Además, sienta las bases para que el código base se pueda liberar en cualquier momento, lo que es esencial para la entrega y la implementación continua.

Dependiendo del tamaño del equipo de desarrollo, pueden surgir dos estilos diferentes de desarrollo basado en troncales. En equipos pequeños, es común fusionar cada nuevo cambio directamente en la rama principal. Sin embargo, en equipos más grandes, se pueden utilizar ramas de corta duración propiedad de una o varias personas o pequeños equipos, las cuales se fusionarán con la rama principal en unos pocos días después de su creación. Para evitar el "infierno de combinación" que puede surgir con ramas de características de larga duración, cualquier cambio que requiera más de unos pocos días para completarse se debe hacer mediante la abstracción de la función en una rama de características.

Existen dos características fundamentales del desarrollo basado en troncales que deben ser consideradas al decidir si implementarlo o no. En primer lugar, este enfoque permite una gran velocidad de desarrollo, lo que significa que los cambios se pueden implementar y fusionar en la rama principal muy rápidamente. En segundo lugar, se deposita una gran confianza en los desarrolladores, ya que se espera que mantengan la estabilidad de la compilación en todo momento. Si bien estos son beneficios reconocidos del desarrollo basado en troncales, es importante tener en cuenta que ningún sistema funciona de manera perfecta para todos los casos.