dfdjsfhdj

**Introducción**

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software para gestionar y rastrear los cambios en el código fuente durante el ciclo de vida de un proyecto. Fue creado por **Linus Torvalds** en 2005 para gestionar el código fuente del proyecto **Linux**, debido a las limitaciones de los sistemas de control de versiones existentes en ese momento.

A lo largo de los años, Git ha ganado una popularidad inmensa, especialmente con la aparición de plataformas como **GitHub**, **GitLab** y **Bitbucket**, que permiten a los desarrolladores colaborar en proyectos de manera eficiente y descentralizada.

Este informe explorará la historia de Git, sus fundamentos, y una lista detallada de los comandos esenciales que los desarrolladores utilizan a diario.

**Historia de Git**

**Orígenes y Creación**

Git fue creado por **Linus Torvalds** en abril de 2005, tras un desacuerdo con el sistema de control de versiones utilizado anteriormente en el desarrollo del kernel de Linux, **BitKeeper**. BitKeeper era un sistema propietario, y sus desarrolladores decidieron restringir su uso gratuito, lo que provocó que Torvalds decidiera crear un nuevo sistema de control de versiones que fuera **rápido, flexible y completamente libre**.

El nombre "Git" fue elegido por Torvalds como una broma. En inglés británico, "git" es un término que significa "tonto" o "idiota", pero también se puede ver como un acrónimo de "Global Information Tracker". Torvalds, conocido por su sentido del humor sarcástico, lo eligió para reflejar una sensación de rebeldía y simplicidad.

**Características Iniciales y Evolución**

Desde su creación, Git se ha centrado en la velocidad, la flexibilidad y el soporte para trabajar de manera distribuida, lo que lo diferencia de sistemas centralizados como **Subversion (SVN)** y **CVS**. En lugar de almacenar solo la última versión de los archivos, Git almacena todo el historial del proyecto, lo que permite a los usuarios acceder a versiones anteriores del código de manera eficiente y realizar operaciones locales sin necesidad de conectarse a un servidor central.

A medida que Git fue ganando popularidad, herramientas gráficas como **GitHub** (lanzado en 2008) facilitaron aún más su adopción por la comunidad de desarrolladores. GitHub se convirtió en un punto de referencia para proyectos de código abierto, permitiendo a los desarrolladores colaborar, compartir y gestionar código fuente de manera más eficiente.

**Situación Actual**

Hoy en día, Git es el sistema de control de versiones más popular en el mundo del desarrollo de software. Es usado tanto en proyectos de código abierto como en empresas de tecnología, y se ha convertido en una herramienta indispensable para los equipos de desarrollo modernos.

**Fundamentos de Git**

Git es un sistema de control de versiones **distribuido**, lo que significa que cada desarrollador tiene una copia completa del repositorio, incluyendo el historial completo de los cambios. Esto permite trabajar de forma autónoma sin depender de una conexión constante con un servidor central.

A continuación se destacan algunos conceptos clave de Git:

* **Repositorio (Repository)**: Es el espacio donde se almacenan todos los archivos y el historial de cambios de un proyecto. Puede ser local (en tu máquina) o remoto (en un servidor como GitHub, GitLab, etc.).
* **Commit**: Es un punto de control en el historial de cambios de un proyecto. Un commit guarda una instantánea de todos los archivos en el repositorio en un momento dado, y cada commit está vinculado a un identificador único (hash).
* **Rama (Branch)**: Es una línea de desarrollo independiente dentro de un repositorio. Las ramas permiten trabajar en diferentes funcionalidades de forma aislada sin afectar a la rama principal (como master o main).
* **Merge**: Es el proceso de combinar los cambios de una rama con otra. Puede implicar la resolución de conflictos si ambas ramas modificaron las mismas líneas de un archivo.

**Comandos Principales de Git**

Git tiene una amplia gama de comandos que permiten gestionar el flujo de trabajo en un repositorio. A continuación, se presenta una lista de los comandos más utilizados, junto con una breve descripción de cada uno.

**Comandos Básicos**

1. **git init**:
   * **Descripción**: Inicializa un nuevo repositorio Git vacío.
   * **Uso**: Se usa cuando se empieza un nuevo proyecto o se convierte un directorio en un repositorio Git.

bash

Copiar

git init

1. **git clone**:
   * **Descripción**: Crea una copia local de un repositorio remoto.
   * **Uso**: Usado para copiar un repositorio existente desde un servidor remoto (como GitHub) a tu máquina local.

bash

Copiar

git clone https://github.com/usuario/repositorio.git

1. **git add**:
   * **Descripción**: Añade cambios al área de preparación (staging area) para ser incluidos en el siguiente commit.
   * **Uso**: Se utiliza para seleccionar qué archivos serán incluidos en el próximo commit.

bash

Copiar

git add archivo.txt

1. **git commit**:
   * **Descripción**: Guarda los cambios añadidos al área de preparación en el repositorio.
   * **Uso**: Crea un nuevo commit que incluye los archivos añadidos con git add.

bash

Copiar

git commit -m "Mensaje de commit"

1. **git status**:
   * **Descripción**: Muestra el estado actual del repositorio, indicando archivos modificados, añadidos y si hay cambios que no han sido confirmados.
   * **Uso**: Usado para ver los archivos modificados y el estado del repositorio.

bash

Copiar

git status

1. **git push**:
   * **Descripción**: Sube los commits locales a un repositorio remoto.
   * **Uso**: Se utiliza para compartir los cambios con otros desarrolladores en un repositorio remoto.

bash

Copiar

git push origin rama

1. **git pull**:
   * **Descripción**: Descarga y fusiona los cambios desde un repositorio remoto a tu rama local.
   * **Uso**: Se utiliza para sincronizar tu copia local con los cambios realizados por otros desarrolladores en el repositorio remoto.

bash

Copiar

git pull origin rama

1. **git branch**:
   * **Descripción**: Muestra las ramas existentes en el repositorio o crea una nueva rama.
   * **Uso**: Se utiliza para gestionar las ramas del repositorio.

bash

Copiar

git branch

git branch nueva-rama

1. **git checkout**:
   * **Descripción**: Cambia de una rama a otra o restaura archivos a su último estado confirmado.
   * **Uso**: Se utiliza para cambiar entre ramas o restaurar archivos.

bash

Copiar

git checkout rama

1. **git merge**:
   * **Descripción**: Combina los cambios de una rama en otra.
   * **Uso**: Se usa para fusionar las modificaciones de una rama con la rama actual.

bash

Copiar

git merge rama

1. **git log**:
   * **Descripción**: Muestra el historial de commits del repositorio.
   * **Uso**: Se utiliza para ver los commits anteriores y su información asociada.

bash

Copiar

git log

**Conclusión**

Git ha revolucionado la forma en que los desarrolladores gestionan el código fuente y colaboran en proyectos de software. Su diseño distribuido y sus potentes herramientas de control de versiones lo han convertido en el sistema más popular para proyectos de código abierto y desarrollo profesional. Git ha evolucionado con el tiempo para ofrecer un conjunto robusto de comandos que permiten gestionar eficientemente el ciclo de vida del código, desde la creación de ramas hasta la fusión de cambios.

Git y sus plataformas asociadas como GitHub han cambiado la dinámica de la colaboración en el desarrollo de software, permitiendo a equipos globales trabajar de manera eficiente en proyectos complejos. Con su flexibilidad y amplio soporte, Git se ha establecido como una herramienta esencial en el panorama del desarrollo de software moderno.

2. Explicación de la actividad en clase

1.crer una carpeta en el escritorio

2. se abre la carpeta

Click derecho en la carpeta y se selecciona “open git bash here”

Se abrirá una consola

Se inicia con el comando “git init”

Se crea la carpeta “.git” una vez iniciamos, en caso de que no aparezca, se selecciona la opción de vista y se selecciona de elementos ocultos

Se crea o se sube dentro de la capeta el archivo que se desee subir a github

Se verifica el estado de la carpeta que se creó en el escritorio con el comando “git status”, aparecerá la dirección de la capeta n color y al final de la línea mostrará el nombre de la rama la cual será “master”.

si aparece el nombre del archivo en rojo es porque no se ha subido el archivo en el repositorio

se utilizara el comando “git add .” para subir el archivo en el repositorio

cada vez que se cree o se modifique un archivo se debe crear un mensaje o un comentario con “git commit -m “mensaje que desee”

cuando se suba el archivo se verifica con “git status” debe aparecer el nombre del archivo en verde

hay que configurar las variables globales cuando se instala git por primera vez con estos comandos: “git config – global user.email “se escribe el correo de la cuenta gitHub” y para el usuario “git config – global user.name “nombre del usuario ””

una vez hecho ese paso, se ingresa a GitHub.com y se inicia sesión, se crea el repostorio, dando click en “new”

se agrega un nombre para el repositorio y se crea

nos dirigimos a la consola del git y pegamos los siguientes comandos: “git remote add origin <https://github.com/Katherin-22/Taller-Git.git>”” y el comando

“git push -u origin main” al segundo comando se le tiene que cambiar el “main” por el “master”

Si dice que el usuario no esta autorizado, se debe buscar las credenciales de Windows y borrarlas, para ello se debe dirigir al buscador de Windows y escribir ”administrador de credenciales ” y darle click en credenciales de Windows y donde dice credenciales genéricas, buscar la credencial modificada y darle en eliminar

Una vez hecho eso de debe crear una rama, se digita el comando “git log” para mostrar el autor, la fecha y el comentario realizado, aparecerá el encabezado de HEAD-> master como origin/ master

Se creara la rama con el comando “git branch” y el nombre que se le quiera dar a la rama

Para cambiar de rama se debe poner el comando “git checkout”, seguido del nombre de la rama

Para que aparezca la rama en GitHub se debe poner el comando “git push origin” seguido del nombre de la rama.

Para añadir colaboradores:

En el GitHub nos debemos dirigir al repositorio que creamos, darse en “settings” después a “collaborators” y oprimir el botón “add people” e ingresar el correo electrónico que se desee ingresar y oprimir en “add to repository”

Para clonar el GitHub lo haremos con el siguiente comando “git clone” mas el link que nos envié el colaborar al correo del GitHub del repositorio

¿Qué es un merge?

Combinación de cambios:

Un merge toma dos ramas (o commits) diferentes y crea un nuevo commit que contiene todos los cambios de ambas ramas.

Se utiliza para integrar cambios de diferentes líneas de desarrollo (ramas) en una sola.

Creación de un nuevo commit:

Un merge generalmente resulta en la creación de un nuevo commit que representa la combinación de los cambios.

Este commit tiene dos commits padres, lo que indica su origen.

¿Cuándo se usa un merge?

Integración de ramas de características:

Cuando se trabaja en una nueva característica en una rama separada, se utiliza un merge para integrar los cambios en la rama principal (como "main" o "develop").

Resolución de conflictos:

Si hay cambios conflictivos entre las ramas que se están combinando, se requiere la intervención manual para resolver los conflictos antes de completar el merge.

Actualización de ramas:

Se puede usar un merge para actualizar una rama con los cambios de otra rama.

Tipos de merge:

Merge rápido (fast-forward):

Si la rama que se está fusionando está directamente por delante de la rama principal, se realiza un merge rápido.

En este caso, no se crea un nuevo commit; la rama principal simplemente se actualiza para apuntar al último commit de la rama fusionada.

Merge de tres vías (three-way merge):

Si las ramas han divergido, se realiza un merge de tres vías.

Git utiliza un commit base común para determinar los cambios en cada rama y luego los combina.

Este tipo de merge puede generar conflictos si hay cambios conflictivos.

En resumen:

Un merge es una operación fundamental en el control de versiones que permite combinar cambios de diferentes ramas.

Se utiliza para integrar características, resolver conflictos y actualizar ramas.

Git utiliza diferentes tipos de merge dependiendo de la situación.

como se realiza:

La forma de realizar un "merge" depende de la herramienta de control de versiones que estés utilizando. Aquí te explico cómo se realiza en Git, que es la herramienta más común:

1. Preparación:

Asegúrate de tener tu repositorio actualizado:

Antes de realizar un "merge", es importante asegurarte de que tanto la rama en la que estás trabajando como la rama que vas a fusionar estén actualizadas con los últimos cambios remotos. Puedes usar los comandos git fetch y git pull para esto.

Cambia a la rama de destino:

Debes cambiar a la rama en la que quieres integrar los cambios. Esta suele ser la rama principal ("main" o "develop"). Puedes usar el comando git checkout nombre\_de\_la\_rama.

2. Realización del "merge":

Ejecuta el comando "merge":

Una vez que estés en la rama de destino, ejecuta el comando git merge nombre\_de\_la\_rama\_a\_fusionar. Esto le indica a Git que combine los cambios de la rama especificada en la rama actual.

Resolución de conflictos (si es necesario):

Si hay cambios conflictivos entre las ramas, Git te informará y marcará los archivos con conflictos. Debes editar manualmente estos archivos para resolver los conflictos y luego usar el comando git add para marcar los archivos como resueltos.

Finalización del "merge":

Una vez que hayas resuelto todos los conflictos (si los hay), puedes finalizar el "merge" con el comando git commit. Git creará un nuevo commit que representa la combinación de los cambios.

3. "Merge" rápido ("fast-forward"):

Si la rama que estás fusionando está directamente por delante de la rama principal:

Git realizará un "merge" rápido. En este caso, no se crea un nuevo commit; la rama principal simplemente se actualiza para apuntar al último commit de la rama fusionada.

Ejemplo práctico:

Supongamos que tienes una rama llamada "feature-x" y quieres integrarla en la rama "main".

Actualiza tu repositorio:

git checkout main

git pull origin main

git checkout feature-x

git pull origin feature-x

Cambia a la rama "main":

git checkout main

Realiza el "merge":

git merge feature-x

Resuelve los conflictos (si es necesario).

Finaliza el "merge":

git commit

Sube los cambios al repositorio remoto:

git push origin main

Consideraciones adicionales:

Es importante comunicarse con tu equipo antes de realizar un "merge", especialmente si estás trabajando en un proyecto colaborativo.

Siempre revisa los cambios antes de finalizar el "merge" para asegurarte de que todo esté correcto.