INTRODUCCIÓN A GIT

Tabla de contenido

INTRODUCCIÓN A GIT 1

PREPARACIÓN PARA LA INSTALACIÓN DE GIT 3

INSTALACIÓN DE GIT 4

CONFIGURACIÓN DE GIT 5

¿Qué pasa si no usamos --global en la configuración de git? 6

INICIALIZAR UN REPOSITORIO CON GIT 7

git init 7

AÑADIR ARCHIVOS AL ÁREA DE PREPARACIÓN 9

CONFIRMAR CAMBIOS 10

ESTADO DEL REPOSITORIO GIT 11

HISTORIAL DE COMMITS EN GIT 13

VOLVIENDO A VERSIONES ANTERIORES CON GIT 14

VOLVER A LA ÚLTIMA VERSIÓN (ÚLTIMO COMMIT) CON GIT 15

ELIMINAR COMMIT 15

Git: Git es un sistema de control de versiones distribuido, lo que significa que permite gestionar el historial de cambios en proyectos de desarrollo de software, este fue diseñado por Linus Torvalds creador del Kernel (núcleo) del sistema operativo Linux.

PREPARACIÓN PARA LA INSTALACIÓN DE GIT

Pasos

1. **Actualizar la lista de paquetes**: Nuestro primer paso, será ejecutar el comando **sudo apt-get update**, este nos permitirá actualizar la lista de paquetes disponibles en los repositorios configurados en el sistema. No instala ni actualiza los paquetes, solo descarga información actualizada sobre las versiones más recientes de los paquetes disponibles para que el sistema pueda instalar o actualizar programas de manera eficiente cuando se le indique.

2. **Instalar las últimas actualizaciones**: Como segundo paso, procederemos a ejecutar el comando **sudo apt-get upgrade,** con este comando actualizamos todos los paquetes instalados en el sistema que tienen versiones más nuevas disponibles, según la información obtenida con el comando **apt-get update**. Este comando instala las actualizaciones sin eliminar ni instalar nuevos paquetes; simplemente actualiza los existentes a sus versiones más recientes.

INSTALACIÓN DE GIT

Pasos

1. Ejecutaremos el comando **sudo apt install git**. Este comando está compuesto por varias instrucciones que se pueden segmentar de la siguiente forma:

    a. **sudo**: Le da permisos de administrador al comando, permitiéndole modificar el sistema (en este caso, instalar software).

    b. **apt:** Es la herramienta de gestión de paquetes para distribuciones basadas en Debian, como Ubuntu, que facilita la instalación, actualización y eliminación de programas.

    c. **install:** Especifica que se desea instalar un paquete en lugar de realizar otras acciones como actualizar o eliminar.

    d. **git:** Es el nombre del paquete que se quiere instalar. En este caso, estás solicitando la instalación de Git, un sistema de control de versiones.

    En resumen, sudo **sudo apt install git** descarga e instala el paquete de Git en nuestro sistema. Si Git ya está instalado, actualizará a la versión más reciente disponible.

CONFIGURACIÓN DE GIT

Para usar Git correctamente, es importante configurarlo adecuadamente, especialmente en términos de la identidad del usuario que realiza las modificaciones.

Para realizar esta configuración de forma adecuada usaremos el comando **git config**.

Pasos:

1. Ejecutar el comando **git config --global user.name "David Martinez"**

   Este comando se usa para configurar el nombre de usuario global en Git. La opción **--global** establece esta configuración para todos los repositorios en el sistema. Esto significa que cada vez que realices un commit (registro de cambios), Git asociará tu nombre con ese commit.

   --global: Aplica esta configuración a todos los repositorios en nuestra máquina, es decir, cualquier repositorio que crees o clones usará este nombre de usuario.

   user.name "David Martinez": Es el nombre que se asignará a nuestros commits. Git lo usará para identificarte como autor de los cambios.

2. Ejecutar el comando **git config --global user.email David.martinez@riwi.io**

   De manera similar, este comando configura el correo electrónico de forma global que se asocia con nuestros commits. El correo electrónico es otra pieza crucial de la identidad de un autor en Git.

   --global: De nuevo, se aplica a todos los repositorios en la máquina.

   user.email "David.martinez@riwi.io": Es el correo electrónico que aparecerá junto con tu nombre en cada commit. Es útil para identificar de manera única al autor, especialmente en proyectos colaborativos.

¿Qué pasa si no usamos --global en la configuración de git?

Si omitimos el parámetro **--global** en el comando y solo usas **git config user.name "David Martinez"** o **git config user.email "David.martinez@riwi.io"**, estas configuraciones solo se aplicarán al repositorio actual en el que estés trabajando, no a todos los repositorios de tu máquina.

Esto puede ser útil si teneos varios proyectos con diferentes identidades. Por ejemplo, si trabajamos en un proyecto personal y otro en una empresa, puedes usar diferentes nombres y correos electrónicos para cada uno.

Ejemplo sin --global:

git config user.name "David Martinez"

git config user.email "David.martinez@riwi.io"

Esto solo afectará al repositorio (Directorio) donde ejecutemos estos comandos, y no a otros repositorios en nuestro sistema.

Resumen

Con **--global**: Estableces la configuración a nivel global para todos los repositorios en tu máquina. Útil cuando usas la misma identidad en todos los proyectos.

Sin **--global**: Configuras los parámetros solo para un repositorio específico, lo que te permite tener diferentes identidades en distintos proyectos.

Este tipo de configuración es clave para asegurar que los commits realizados estén correctamente asociados con nuestro nombre y correo electrónico. Es una práctica esencial cuando trabajamos con Git, especialmente en proyectos colaborativos.

INICIALIZAR UN REPOSITORIO CON GIT

Cuando trabajamos con Git, antes de poder realizar cualquier seguimiento de cambios o historial de versiones en un proyecto, necesitaremos inicializar un repositorio. Este proceso convierte un directorio normal de nuestro sistema en un repositorio Git, es decir, un lugar donde Git puede empezar a gestionar y almacenar información sobre el historial de cambios de tus archivos.

git init

El comando git init se utiliza para inicializar un repositorio Git vacío en el directorio actual. Esto crea una estructura interna de carpetas y archivos necesarios para que Git pueda gestionar el control de versiones. Al ejecutar este comando, le indicas a Git que deseas que el directorio sea un repositorio, comenzando a registrar los cambios de los archivos a partir de ese momento.

Ejemplo del Comando:

$ **git init**

Al ejecutar git init en la terminal, observaremos un mensaje similar al siguiente en la terminal:

Initialized empty Git repository in /ruta/a/tu/proyecto/.git/

Durante la ejecución del comando anterior, Git ha realizado una serie de pasos en segundo plano que nos arroja como resultado, la creación de un sub directorio llamado **.git**. Este directorio que se encuentra oculto se crea dentro del directorio donde ejecutamos el comando **git init**.

Este contiene toda la información de configuración, historial y datos necesarios para que Git pueda gestionar el repositorio y dentro de él se incluyen:

- **config**: Contiene la configuración del repositorio (como los datos de autor y las opciones de seguimiento).
- **HEAD**: Archivo que señala la rama actual en la que estás trabajando (generalmente master o main).

- **objects**: Guarda los objetos Git que representan los archivos y sus versiones.
- **refs**: Contiene las referencias a las ramas y etiquetas.

En este punto, nuestro directorio no contiene aún archivos rastreados. Aunque el repositorio Git se ha inicializado, no hay archivos bajo control de versiones hasta que los agreguemos, sin embargo, Git ya tendrá la capacidad de empezar a rastrear los cambios que realicemos en los archivos dentro del directorio

AÑADIR ARCHIVOS AL ÁREA DE PREPARACIÓN

En Git, el proceso de añadir archivos a un repositorio se realiza en dos etapas principales:

- Añadir los archivos a la "zona de preparación" (staging area)
- Confirmar esos cambios con un commit.

Vamos a ver cómo funcionan esos pasos.

1. Crear un nuevo archivo

   echo "Esto es una prueba" > ejemplo.txt

   Este comando crea un archivo llamado ejemplo.txt y agrega el texto "Esto es una prueba" en su contenido.

   ¿Qué hace echo?: echo es un comando que simplemente imprime el texto que le pases como argumento en la salida estándar.

   > ejemplo.txt: Redirige esa salida del comando echo al archivo ejemplo.txt, creando el archivo si no existe o sobrescribiéndolo si ya existe.

   Este paso es simplemente la creación de un nuevo archivo en el directorio de tu proyecto. A partir de ahora, ese archivo está en tu sistema de archivos, pero Git aún no lo está rastreando.

2. Añadir el archivo al área de preparación

   Utilizaremos el comando git add para añadir nuestro archivo ejemplo.txt al área de preparación o staging area de Git. El área de preparación es una especie de "zona intermedia" donde Git guarda los archivos antes de confirmarlos con un commit.

¿Qué hace git add?

git add no realiza un commit ni almacena el archivo definitivamente en el repositorio. Simplemente le indica a Git que deseas incluir ese archivo en el siguiente commit.

Este comando se usa para preparar los cambios antes de hacer el commit, permitiéndote decidir qué archivos específicos deseas agregar al historial de versiones.

Al usar **git add ejemplo.txt**, Git empieza a rastrear el archivo ejemplo.txt y lo prepara para ser incluido en el próximo commit.

CONFIRMAR CAMBIOS

¿Qué sigue después de usar git add?

Una vez que añades un archivo con git add, el archivo está listo para ser confirmado en el repositorio.

Para guardar definitivamente el archivo en el historial de versiones de Git, debes usar el comando **git commit**. Este comando confirmaría los cambios y guardaría el archivo ejemplo.txt en el repositorio Git.

Ejemplo del comando:

git commit -m "Añadido archivo ejemplo.txt con un mensaje de prueba"

Este comando estará acompañado de un parámetro representado por un guion medio (-) y la letra m los cuales permitirán agregar un comentario descriptivo que apoye la confirmación del cambio que realizaremos.

¿Por qué se necesita el área de preparación para la confirmación?

El área de preparación (staging area) es importante porque te permite controlar qué cambios se van a registrar en el historial de Git. Puedes agregar solo ciertos archivos a esta área, dejando otros sin tocar hasta que estés listo para incluirlos en un commit.

Este enfoque es útil, ya que permite realizar cambios en múltiples archivos y confirmar solo aquellos que deseas, sin tener que comprometer todos los cambios al mismo tiempo.

En el caso que modifiquemos un archivo, podemos confirmar el cambio, pero tenemos que usar la opción a:

Ejemplo:

1. **nano ejemplo.txt**
2. **git commit-am "He modificado el fichero ejemplo.txt"**

ESTADO DEL REPOSITORIO GIT

Git nos permite utilizar una herramienta muy útil para ver el estado actual del repositorio, es decir, qué archivos han sido modificados, cuáles están listos para ser confirmados (staged), cuáles están sin seguimiento (untracked), y si hay algún cambio pendiente por confirmar.

Esta herramienta está disponible bajo el comando **git status**

¿Qué muestra git status?

- **Archivos modificados (modified):** Indica qué archivos han sido modificados, pero aún no han sido añadidos al área de preparación con **git add**.

- **Archivos no rastreados (untracked):** Muestra los archivos que Git no está rastreando, es decir, aquellos que aún no se han añadido al repositorio con **git add**.

- **Archivos preparados para el commit (staged):** Muestra los archivos que han sido añadidos al área de preparación (con **git add**) y están listos para ser confirmados en el próximo commit.

- **Rama activa (branch):** Te muestra en qué rama estás trabajando actualmente, por ejemplo, main o master.

- **Commits pendientes:** Si hay cambios en el repositorio que aún no han sido comprometidos, **git status** te lo indica.

Ejemplo de salida de **git status**:

- nothing to commit, working tree clean
  Este mensaje indica que no hay cambios pendientes y que tu repositorio está limpio.

HISTORIAL DE COMMITS EN GIT

Utilizando el comando **git log** te permitirá ver el historial de commits realizados en un repositorio, mostrándote un listado de los cambios anteriores con detalles como el autor, la fecha y el mensaje de cada commit y un identificador único para cada commit dentro de nuestro repositorio.

¿Qué muestra git log?

- **ID de commit (SHA-1 hash):** Un identificador único de cada commit, normalmente largo, que sirve para referirse a un commit específico.
- **Autor:** Muestra el nombre del autor del commit.
- **Fecha:** Indica cuándo se realizó el commit.
- **Mensaje del commit**: El mensaje escrito por el autor, que describe qué cambios se hicieron en ese commit.

Ejemplo de salida de git log:

commit 9fceb02a0f3a8b4bc41b26f4d4212f3b5a463ec1

Author: David Martinez <David.martinez@riwi.io>

Date: Tue Mar 24 14:30:35 2025 -0400

Añadido archivo ejemplo.txt con mensaje de prueba

Este mensaje indica que hay un commit con un identificador único (9fceb02a0f3a8b4bc41b26f4d4212f3b5a463ec1), realizado por David Martinez, el 24 de marzo de 2025, con el mensaje "Añadido archivo ejemplo.txt con mensaje de prueba".

VOLVIENDO A VERSIONES ANTERIORES CON GIT

Git como herramienta de control de versiones no solo permite realizar el registro y seguimiento de cambios, sino que también nos permitirá regresar a un punto determinado o versión anterior de un archivo sin perder las modificaciones posteriores.

Para volver a un commit o versión anterior tenemos disponible el comando **git checkout <commit_id>**

Ejemplo:

git checkout <commit_id>

- **<commit_id>**: Es el identificador único del commit al que deseas volver. Puedes obtener el commit_id usando **git log**, que te muestra el historial de commits.

Al usar git checkout <commit_id>, el repositorio se moverá a ese commit específico, pero **sin perder las modificaciones posteriores**. Nuestros cambios no comprometidos aún estarán disponibles en el área de trabajo, y podremos seguir editándolos.

Nota: Este comando nos coloca en un **estado de "detached HEAD"** (cabeza desanclada), lo que significa que no estás en una rama activa. Para continuar trabajando, es recomendable crear una nueva rama si deseas hacer más cambios a partir de ese punto.

Ejemplo de uso práctico:

Supongamos que tenemos el siguiente commit id obtenido de **git log**: 9fceb02a0f3a8b4bc41b26f4d4212f3b5a463ec1

Podemos volver a ese commit con: **git checkout 9fceb02a0f3a8b4bc41b26f4d4212f3b5a463ec1**

VOLVER A LA ÚLTIMA VERSIÓN (ÚLTIMO COMMIT) CON GIT

Si deseamos volver a la última versión del repositorio (es decir, al último commit de nuestra rama activa), puedes usar el comando **git switch**. Esto te permitirá salir del estado de "detached HEAD" y volver a trabajar normalmente en la rama que estabas utilizando.

Ejemplo:

git switch main

- **main**: Es el nombre de la rama a la que deseas volver. Este comando te llevará a la última versión de esa rama.

ELIMINAR COMMIT

El comando git reset se utiliza para **mover la cabeza del repositorio a un commit anterior**. Esto puede implicar la eliminación de los commits posteriores, y con el modificador --hard, también eliminarás los cambios en los archivos del área de trabajo.

Ejemplo:

git reset --hard HEAD~3

- **HEAD~3**: Esto le dice a Git que vuelva a **tres commits atrás** desde el commit actual. HEAD hace referencia al último commit, y ~3 indica que nos movemos tres commits hacia atrás.
- **--hard**: Esta opción asegura que los cambios no solo se deshagan en el historial de commits, sino también en tu **área de trabajo y el índice** (lo que significa que cualquier cambio local no confirmado también será eliminado).

Este comando es muy útil si deseas **eliminar por completo** los últimos commits del historial, pero debemos tener cuidado, ya que **elimina permanentemente esos commits y los cambios asociados**.