**Git y Github**

**Sistemas de Control de Versiones**

En el proceso de desarrollo de software es un requisito casi indispensable mantener unregistro de los cambios que se realizan sobre el código fuente a lo largo del tiempo. Es debido a esto que cobran importancia los sistemas de control de versiones. Estos sistemas son herramientas que permiten realizar un seguimiento de los cambios y también permiten proteger el código de errores humanos accidentales. Además, un sistema de control de versiones facilita el trabajo en equipo a la hora de desarrollar software, ya que mientras un integrante trabaja en alguna funcionalidad específica, otro podría estar trabajando en alguna corrección de errores o bien en otra funcionalidad, para luego integrar las soluciones y realizar una sincronización del trabajo de cada uno.

El uso de un sistema de control de versiones tiene tres ventajas principales:

1. Gracias al historial de cambios se puede saber el autor, la fecha y notas escritas sobre los cambios realizados. También permite volver a versiones anteriores para ayudar a analizar causas raíces de errores y es crucial cuando hay que solucionar problemas de versiones anteriores.

2. Creación y fusión de ramas. Al tener varios integrantes del equipo trabajando al

mismo tiempo, cada uno en una tarea diferente, pueden beneficiarse de tener flujos

de trabajo independientes. Posteriormente se pueden fusionar estos flujos de

trabajos o ramas a una principal. Los sistemas de control de versiones tienen

mecanismos para identificar que los cambios entre ramas no entren en conflicto para

asegurar la funcionalidad y la integración.

3. Trazabilidad de los cambios que se hacen en el software. Poder conectar el sistema

de control de versiones con un software de gestión de proyectos y seguimiento de

errores ayuda con el análisis de la causa raíz de los problemas y con la recopilación

de información.

El concepto de versión (también llamado revisión o edición) de un proyecto (código fuente) hace referencia al estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Los sistemas de control de versiones utilizan repositorios para almacenar el proyecto actualizado junto a sus cambios históricos. Los sistemas de control de versiones centralizados almacenan todo el código en un único repositorio, es decir que un único servidor contiene todos los archivos versionados. Esto representa un único punto de falla dado que si el servidor no está disponible por un tiempo nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.

Los sistemas de control de versiones distribuidos permiten en cambio continuar el trabajo aún cuando el repositorio de referencia no está disponible. En estos sistemas los clientes no solo descargan la última copia del código, sino que se replica completamente el repositorio con los cambios históricos (versiones). De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo.

Git

Git es un proyecto de código abierto maduro y con un activo mantenimiento desarrollado originalmente por Linus Torvalds. Este sistema de control de versiones distribuido que funciona bajo cualquier plataforma (Windows, MacOS, Linux, etc) y está integrado en una amplia variedad de entornos de desarrollo (IDEs). Este sistema presenta una arquitectura distribuida, es decir que, cada desarrollador posee una copia del trabajo en un repositorio local donde puede albergar el historial completo de todos los cambios y, mediante determinados comandos, realiza sincronizaciones al repositorio remoto.

Git fue diseñado teniendo en cuenta las siguientes características

● Rendimiento: Los algoritmos implementados en Git aprovechan el profundo

conocimiento sobre los atributos comunes de los auténticos árboles de archivos de

código fuente, cómo suelen modificarse con el paso del tiempo y cuáles son los

patrones de acceso. El formato de objeto de los archivos del repositorio de Git

emplea una combinación de codificación delta (que almacena las diferencias de

contenido) y compresión, y guarda explícitamente el contenido de los directorios y

los objetos de metadatos de las versiones.

● Seguridad: la principal prioridad es conservar la integridad del código fuente

gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los

directorios, las versiones, las etiquetas y las confirmaciones, están protegidos con un

algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se

salvaguarda el código y el historial de cambios frente a las modificaciones

accidentales y maliciosas, y se garantiza que el historial sea totalmente trazable.

● Flexibilidad: es flexible en varios aspectos, en la capacidad para varios tipos de flujos

de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como

pequeños y en su compatibilidad con numerosos sistemas y protocolos. Se ha ideado

para posibilitar la ramificación y el etiquetado como procesos de primera

importancia y las operaciones que afectan a las ramas y las etiquetas (como la fusión

o la reversión) también se almacenan en el historial de cambios.

Áreas y Estados

Para trabajar con git es fundamental entender los estados por los que pueden pasar los archivos durante todo el flujo de desarrollo.

En un proyecto de Git hay 4 secciones fundamentales:

● Directorio de trabajo (Working Directory): es una copia de una versión del proyecto,

Son archivos sacados de la base de datos comprimida y se colocan en el disco para

poder ser usados o modificados.

● Área de preparación (Staging Area): Es un archivo que se encuentra dentro del

directorio de Git y que contiene información acerca de lo que va a ir en la próxima

confirmación.● Directorio de Git (Local Repository): Es el lugar donde se almacenan los metadatos y

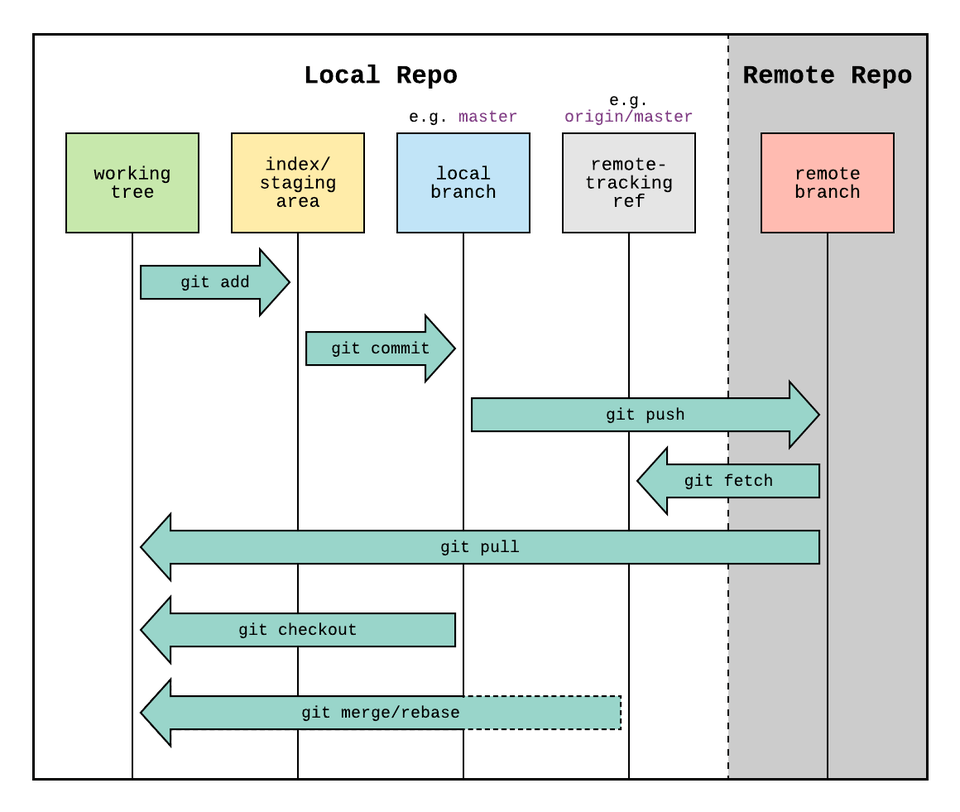
la base de datos de objetos del proyecto. Es lo que se copia cuando se clona un

repositorio desde otra fuente.

● Repositorio Remoto (Remote Repository): Es el repositorio que se encuentra en un

servidor remoto y con el que eventualmente se sincronizan los trabajos entre los

diferentes integrantes del equipo.



se puede ver como un set de herramientas muy completo, pero para un manejo básico

de repositorios en Git es necesario conocer, por lo menos, los siguientes comandos:

● git init es el comando para inicializar un directorio como repositorio Git, se ejecuta

dentro del directorio del proyecto y, como resultado, crea un subdirectorio .git que

contiene todos los archivos para poder realizar el seguimiento de los cambios,

etiquetas, etc.

● git add <file> luego de la creación, modificación o eliminación de un archivo, los

cambios quedan únicamente en el área de trabajo, por lo tanto es necesario pasarlos

al área de preparación mediante el uso del comando git add, para que sea incluido

dentro de la siguiente confirmación (commit).

● git status es un comando que permite conocer en qué estado se encuentran los

archivos

● git commit, con este comando se confirman todos los cambios registrados en el área

de preparación, o lo que es lo mismo, se pasan los cambios al repositorio local.

● git push es el comando que se utiliza para enviar todas las confirmaciones

registradas en el repositorio local a un repositorio remoto.

● git pull funciona al inverso de git push, trayendo todos los cambios al repositorio

local, pero también dejándolos disponibles directamente para su modificación o

revisión en el área de trabajo. Es importante mencionar que se utiliza cuando ya se

tiene un repositorio local vinculado a uno remoto, al igual que con el comando git

push.

● git clone, en el caso de necesitar “bajar” un repositorio remoto de algún proyecto ya

existente se puede ejecutar este comando. Genera un directorio (con el nombre del

repositorio o uno especificado explícitamente) que contiene todo lo propio al

proyecto, además del subdirectorio .git necesario para poder gestionar los cambios y

todo lo pertinente al repositorio Git.

Github

GitHub es una plataforma de colaboración formal e informal de desarrollo de software

(conocida también como plataforma de social coding). En esta plataforma se pueden

publicar repositorios remotos que funcionan bajo el sistema de control de versiones Git. La plataforma configura los proyectos nuevos como de código abierto, por lo que cualquier persona puede verlos, pero esto es configurable.

¿Cómo empezar?

Lo primero para comenzar en GitHub es crear una cuenta, para ello hay que:

● Acceder al siguiente enlace. https://github.com/

● Hacer click en el botón Sign up.

● Completar el formulario.

● Validar la cuenta mediante correo electrónico.

Con la cuenta creada y verificada, se puede crear un repositorio nuevo, para esto hay que seguir los siguientes pasos:

● Hacer click en el botón + ubicado en la esquina superior derecha de la pantalla y, del

menú contextual que se despliega, hacer click en el botón New repository.

● El paso anterior redireccionará a una pantalla con un formulario donde hay que

completar el nombre del repositorio, indicar si será un repositorio privado o público,

y finalmente elegir entre algunos archivos iniciales opcionales como un README,

.gitignore o un archivo de licencias.

● Una vez completo el formulario, hacer click en el botón Create repository, esta

acción redirigirá a la pantalla principal del sitio del repositorio, donde se pueden

observar diferentes instrucciones según la situación en la que se encuentre.

Interfaz de usuario gráfica

Descripción generada automáticamente

EJEMPLO PARA TRABAJAR CON GIT DESDE LA TERMINAL LUEGO DE HABER CREADO EL REPOSITORO EN GITHUB

* PARA ACTUALIZAR EL REPOSITORIO REMOTO

**git init** \*para iniciar en el directorio de trabajo local

**git add** segundo.txt \* para seleccionar el archivo a subir o . para subir todo

**git commit -m** "segundo" \* para hacer un commit y preparar todo para subir al repositorio remoto

**git brach -M main** \* para seleccionar la rama en donde se van a guardar los cambios

**git push -u origin main** \* para subir todo al repositorio remoto de github.

* PARA ACTUALIZAR NUESTRO REPOSITORIO LOCAL

**git fetch** \* descarga todo el historial del repositorio remoto

**git merge** \* realiza la combinación de la rama del marcador con la rama local

**git pull**  \* descarga el historial del marcador