INTRODUCCIÓN A GIT, GITHUB Y GITLAB

CONTENIDO

¿QUÉ ES GIT?

Git es un sistema de control de versiones distribuido que rastrea los cambios en cualquier conjunto de archivos informáticos, generalmente utilizado para coordinar el trabajo entre programadores que desarrollan código fuente de manera colaborativa durante el desarrollo de software. Sus objetivos incluyen la velocidad, la integridad de los datos y la compatibilidad con flujos de trabajo distribuidos y no lineales.

El control de versiones se refiere al proceso de guardar diferentes archivos o «versiones» a lo largo de las diferentes etapas de un proyecto. Esto permite a los desarrolladores hacer un seguimiento de lo que se ha hecho y volver a una fase anterior si deciden que quieren revertir algunos de los cambios que han hecho.
Esto es útil por varias razones, por ejemplo, facilita la resolución de errores y la corrección de problemas que puedan ocurrir durante el desarrollo. También sirve para documentar los cambios en cada versión, para ayudar a cualquier miembro del equipo a mantenerse al día sobre lo que se ha completado y lo que aún queda por hacer.

En otras palabras, Git es un sistema de control de versiones distribuido de código abierto que permite a los desarrolladores guardar «instantáneas» de un proyecto de software para rastrear los cambios en sus proyectos. En esencia, mantiene un registro de todos los cambios que se realizan en cualquier programa.

Lo que lo diferencia del control de versiones tradicional es que se puede trabajar en diferentes versiones de rama del software y gestionarlas todas al mismo tiempo.
Git brinda esta libertad para trabajar en una gran variedad de versiones a la vez. Es imprescindible para cualquier equipo que quiera hacer un desarrollo basado en características.

Con Git, todo esto ocurre localmente en nuestro ordenador, pero también podemos solicitar y dar acceso a otros desarrolladores de forma manual, por ejemplo, a través de una LAN.
Cuando se trabaja con equipos remotos o distribuidos más grandes, la mayoría de las empresas recurren a soluciones basadas en la nube, como GitLab y GitHub.

En resumen, Git es una herramienta poderosa que permite a los desarrolladores gestionar y controlar las versiones de su código fuente de manera eficiente y que facilita el desarrollo colaborativo.

HISTORIA DE GIT

Linus Torvalds es un programador finlandés que es considerado uno de los padres fundadores del software libre. En 1991, Torvalds comenzó a desarrollar el kernel de Linux, el núcleo del sistema operativo GNU/Linux y para poder gestionar las versiones del código fuente de Linux, Torvalds utilizó el sistema de control de versiones BitKeeper. Sin embargo, en 2005, los desarrolladores de BitKeeper decidieron restringir el uso de su software para proyectos comerciales.
Esto llevó a Torvalds a crear su propio sistema de control de versiones, al que llamó Git y se basó en los conceptos de control de versiones distribuidos, que permitían a los desarrolladores trabajar en el mismo proyecto desde diferentes ubicaciones.

Git se lanzó por primera vez en 2005, en los primeros años fue utilizado principalmente por desarrolladores de Linux, sin embargo, a medida que se hizo más conocido, comenzó a ser adoptado por otros proyectos de software libre y comercial.

El auge de Git se produjo en los años 2010, en esta década el desarrollo de software se volvió cada vez más colaborativo y distribuido. Git se adaptó perfectamente a este nuevo paradigma, convirtiéndose en el sistema de control de versiones más popular del mundo.

CARACTERÍSTICAS DE GIT

Algunas de sus características clave son:

Distribución y descentralización: A diferencia de los sistemas de control de versiones centralizados, donde se almacena una única copia del repositorio, Git es distribuido. Cada copia local del código de un desarrollador es también un repositorio completo que puede contener todo el historial de cambios.

Rendimiento sólido: Las características básicas de rendimiento de Git son muy sólidas en comparación con otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado para un mejor rendimiento. Los algoritmos en Git aprovechan el conocimiento sobre los atributos comunes de los árboles de archivos de código fuente, cómo suelen modificarse con el tiempo y cuáles son los patrones de acceso.

Control de versiones: Git proporciona un listado de los archivos llamados *commits*, con la fecha en que se modificó cada archivo. Permite tener un control de versiones de los cambios realizados en los archivos de un proyecto y la posibilidad de restablecer cambios o retroceder en el tiempo.

Creación de ramas (branches): Git permite crear ramas o branches, lo que facilita el desarrollo flexible y simultáneo. Cada rama puede contener su propio conjunto de cambios sin afectar a otras ramas.

Compatibilidad: Git es compatible con una amplia gama de sistemas operativos y lenguajes de programación.

En resumen, Git es un sistema maduro de control de versiones de código abierto que se mantiene activo y es utilizado por miles de desarrolladores en todo el mundo. Su arquitectura distribuida, rendimiento sólido y flexibilidad lo convierten en una herramienta esencial para el desarrollo de software.

CONCEPTOS CLAVE

REPOSITORIO: Un repositorio en Git es un lugar donde se almacenan los archivos de un proyecto, junto con el historial de todos los cambios realizados en los archivos. Puede contar con múltiples colaboradores y ser público o privado.
Los repositorios en Git son esenciales para el desarrollo colaborativo y el seguimiento de versiones de software.

En otras palabras, un repositorio es todo proyecto que está siendo seguido por Git, es decir, que ya tiene un historial de Git en el que se están registrando sus cambios.

COMMIT: Un commit en Git es una acción que permite a un usuario guardar los cambios realizados en un archivo o conjunto de archivos en un repositorio Git. Cada commit tiene un identificador único que lo diferencia de otros commits y permite a los usuarios ver los cambios realizados en el archivo o archivos. En resumen, un commit es una instantánea o un hito dentro del proyecto de Git.

Dos características importantes de los commits son:

- Podemos recordar los cambios a los que se les hizo commits en una fecha posterior, o revertir el proyecto a esa versión.

- Si varios commits editan diferentes partes del proyecto, no se sobrescribirán entre sí, aunque los autores de los commits no se conozcan entre sí. Este es uno de los beneficios de usar Git sobre una herramienta como Dropbox o Google Drive.

Existen dos opciones comunes que se pueden incluir con el comando git commit:

1. La opción -m: La más utilizada, que significa mensaje. Cuando se llama a git commit, es necesario incluir un mensaje. El mensaje debe ser una breve descripción de los cambios a los que se les está realizando commit. Por ejemplo:

   git commit -m "Mi mensaje"

   Si no se incluye la opción -m, nos pedirá agregar un mensaje en el editor de texto predeterminado.

2. La opción -a: Representa todo (all). Esta opción prepara automáticamente todos los archivos para realizarles commit. Sin embargo, solo se confirmarán los archivos que el repositorio de Git tenga conocimiento. Por ejemplo:

   git commit -a "Mis nuevos cambios"

   La opción -a no pondrá en el staging área (área de preparación) los archivos que no existen actualmente en el repositorio. Para preparar nuevos archivos, primero debemos invocar el comando git add.

En síntesis, un commit es cada uno de los cambios registrados en el historial de Git.
Cada uno de los desarrolladores manda los commits de cambios que ha hecho, esto no es automático, el desarrollador tiene decirle a Git "yo he hecho esto y lo he hecho por tal motivo"

RAMAS: Las ramas en Git son una bifurcación del estado del código que crea un nuevo camino de cara a la evolución del código, en paralelo a otras ramas que se puedan generar. Las ramas figuran una línea independiente de desarrollo y sirven como una abstracción en los procesos de edición, preparación y confirmación. Las ramas son pequeñas nuevas versiones de nuestro proyecto que permiten desarrollar características, corregir errores, o experimentar con seguridad las ideas nuevas en un área contenida de nuestro repositorio.

En otras palabras, las ramas son ramificaciones, bifurcaciones, nuevos caminos que toma el proyecto.
En Git todo se trabaja por ramas, hay una rama que es la principal y suele llamarse master, donde está el proyecto en producción. Cada vez que se quiere trabajar en alguna característica nueva o corregir algo, se saca una rama para trabajar en un ambiente aislado, el cual es una copia exacta del proyecto, pero separada.
Trabajamos en ese ambiente aislado y si algo se rompe no comprometemos el proyecto, si todo va bien luego esa rama la unificamos con el proyecto principal y si va mal, podemos eliminar la rama sin ningún problema.

CLON: Un clon de un repositorio Git ya existente es lo mismo que ese repositorio. Significa que es una copia completa, incluidos los metadatos del repositorio.

Un clon es una copia exacta del repositorio. Cuando un programador se integra a un equipo de trabajo, lo primero que debe hacer es clonar el repositorio en su equipo local, entonces, cada uno de los desarrolladores tiene un clon del repositorio en su equipo local.

FORK: Un fork en Git es una copia exacta de un proyecto que se hace para partir de él y hacerle modificaciones. Cuando se trabaja con repositorios Git, hacer un fork supone generar dos URL distintas, y los cambios que se hacen en el repositorio original no se transmiten automáticamente a la copia (fork). Con fork se puede crear una copia en remoto de un repositorio Git en nuestra cuenta de GitHub para clonar de forma local y hacer pull y push de los cambios realizados.

Un fork, a diferencia de un clon y a diferencia de una rama, es un proyecto completamente diferente que se crea a partir de otro. Por ejemplo, las distribuciones de Linux, todas se basan en el mismo kernel, pero a partir de ahí cada una toma su diferente camino.
Cuando un proyecto es un fork de otro, significa que está basado en lo que había trabajado el otro proyecto.

DIFERENCIEMOS RAMA, CLON Y FORK: Una rama es un nuevo camino dentro del mismo repositorio o un camino alterno dentro del mismo proyecto (piensa en una rama como un ambiente aislado dentro del mismo proyecto).
Un clon es una copia completa del repositorio en tu máquina, es el proyecto, pero en otro lugar (local), piensa en un clon como traerte el proyecto entero a tu máquina.
Un proyecto nuevo basado en otro, es partir de un proyecto existente para crear otro.

Ejemplo simple: Imagina un libro:

- **Clon:** Te descargas el libro completo a tu casa.
- **Rama:** Escribes notas y cambios en hojas separadas dentro del mismo libro.
- **Fork:** Tomas el libro original y escribes tu propia versión, con otro título y otro rumbo.

FLUJO BÁSICO DE GIT

1. Crear el repositorio: Hay dos formas:
   git init si es un repositorio nuevo desde cero
   git clone si es un repositorio que ya existe

2. Trabajar en el repositorio local y enviar los cambios al repositorio a través de commits. Cada vez que el programador hace un cambio no se guarda en el repositorio, manualmente tiene que enviar los commits porque cada commit debería representar una funcionalidad específica.
   Pero los cambios del programador no van directamente al repositorio, hay una etapa intermedia llamada staging area o área de preparación. Cuando se hacen cambios y se tocan varios archivos, cada cambio se envía al staging área con: **git add**.

3. Cuando ya estamos seguros de que hicimos todas las modificaciones en todos los archivos necesarios, enviamos los cambios que estaban en espera en el staging área al repositorio, con el comando: **git commit**.

   El comando git commit se utiliza para guardar los cambios en el repositorio local y están listos para ser enviados al repositorio remoto. El mensaje de commit debería explicar la característica destacada de la captura que estamos guardando.

   Por ejemplo, podemos usar el comando git add index.html y git commit -m 'el botón de función de formulario creado'. También podemos dejar una nota de los cambios para que nuestros colegas puedan comprender lo que sucedió.

4. Cuando se trabaja en equipo, lo normal es tener dos ramas, la master que es el proyecto principal y sacamos otra rama, generalmente llamada dev, desde la rama dev cada desarrollador saca su propia rama para trabajar en la funcionalidad que le corresponde.
   Las ramas deben volver a integrarse cuando el trabajo se complete, con **git merge**.

5. Integrar la rama dev a la rama master, luego de corregir todos los conflictos y de integrar el trabajo en dev.
   Una vez la rama master esté actualizada y tenga todos los cambios que han hecho los programadores, es el momento de enviarlos a producción, con **git push**.

¿QUÉ ES GITHUB?

GitHub fue fundada en 2008 por Chris Wanstrath, Tom Preston-Werner y Scott Chacon. Es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador.
Fue adquirido por Microsoft en junio de 2018. Esta plataforma permite a los desarrolladores subir y gestionar el código de sus aplicaciones y herramientas utilizando el sistema de control de versiones Git diseñado por Linus Torvalds.
GitHub ha contribuido a la popularización de Git, ya que ofrece una interfaz fácil de usar y una comunidad activa de desarrolladores.

Algunas características de GitHub son:

Repositorios: Los desarrolladores pueden crear repositorios para alojar sus proyectos de forma gratuita. Estos repositorios contienen el código fuente y permiten que otros usuarios lo descarguen, revisen y colaboren en su desarrollo.

Control de versiones: GitHub utiliza Git para administrar las diferentes versiones de un proyecto. Esto permite comparar el código entre versiones, restaurar versiones antiguas y fusionar cambios de distintas ramas.

Colaboración: Además de mirar el código y descargar diferentes versiones de una aplicación, GitHub también funciona como una red social para conectar desarrolladores y usuarios. Esto facilita la colaboración y mejora de las aplicaciones.

GitHub también incluye funciones de organización y gestión de proyectos. Permite asignar tareas a individuos o grupos, establecer permisos y roles para los colaboradores y usar la moderación de comentarios para mantener a todos en la tarea.

GitHub facilita la colaboración con Git. Es una plataforma que puede mantener repositorios de código en almacenamiento basado en la nube para que varios desarrolladores puedan trabajar en un solo proyecto y ver las ediciones de cada uno en tiempo real.
Es un sistema como Facebook o Twitter, que guarda un proyecto, sus cambios y cada una de sus versiones. Es tan popular que es la red social del código, una hoja de vida que demuestra lo que sabe cada programador.
Además, los repositorios de GitHub están disponibles públicamente. Los desarrolladores de todo el mundo pueden interactuar y contribuir al código de los demás para modificarlo o mejorarlo, lo que se conoce como «codificación social». En cierto modo, esto hace que GitHub sea un sitio de redes para profesionales de la web.

En resumen, GitHub es una plataforma esencial para la comunidad de desarrolladores, donde se comparte, colabora y mejora el código de proyectos de código abierto.

DIFERENCIA ENTRE GIT Y GITHUB

Git es un software de VCS (sistema de control de versiones) local que permite a los desarrolladores guardar instantáneas de sus proyectos a lo largo del tiempo. Generalmente es mejor para uso individual.

GitHub es una plataforma basada en la web que incorpora las características de control de versiones de Git para que puedan ser utilizadas de forma colaborativa. También incluye características de gestión de proyectos y equipos, así como oportunidades para la creación de redes y la codificación social.

CAMBIOS EN GITHUB: DE MASTER A MAIN

El movimiento de #BlackLivesMatter ha ayudado a que GitHub sustituya algunas palabras usadas en su plataforma con relación al racismo.

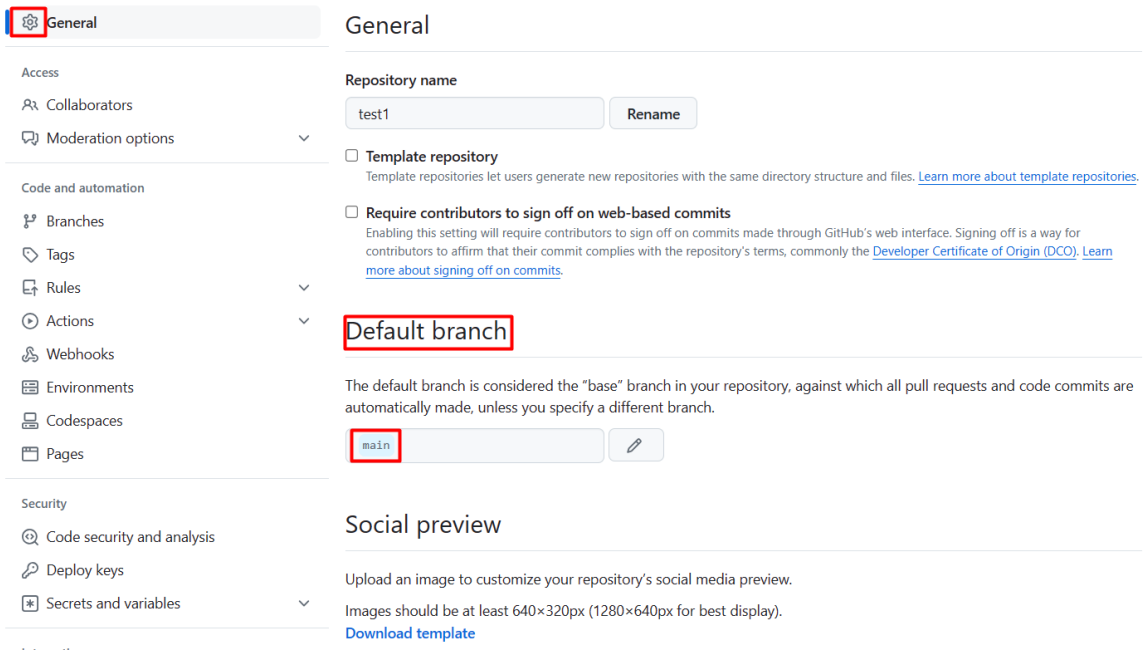
Palabras como master, whitelist, blacklist y slave se encuentran en este proceso de cambio. Pero el más importante en este momento y que ya ha empezado a tener efecto es que la rama master ahora se llama main.

Ahora, todos los repositorios nuevos que se creen empezarán a mostrar main como rama principal y tendrán que hacer sus comandos respectivos allí.

La preferencia por usar “main” en lugar de “master” se debe a una iniciativa para eliminar términos que puedan considerarse ofensivos o insensibles. En el contexto de la programación, “main” se utiliza como la rama principal de un repositorio de código fuente.

Algunas personas consideran que “master” tiene connotaciones negativas relacionadas con la esclavitud, por lo que han optado por cambiarlo. Sin embargo, esta elección puede variar según la comunidad y la organización.

Sin embargo, si quieres llamar a tu rama principal master, puedes cambiar el nombre por defecto:



¿QUÉ ES GITLAB?

GitHub no es el único repositorio Git que puede considerar si está buscando colaborar en un proyecto de desarrollo. GitLab es otra plataforma muy similar que también vale la pena ver.

Al igual que GitHub, GitLab permite almacenar código y usar las capacidades de control de versiones de Git. Sin embargo, también proporciona permisos de usuario más personalizados e incluye una integración continua (CI) incorporada. Esto elimina la necesidad de las solicitudes de extracción utilizadas en GitHub.

GitLab ofrece una amplia gama de características DevOps, como la integración continua, la seguridad e incluso herramientas de despliegue de aplicaciones. Es una plataforma Git y DevOps basada en la nube que ayuda a los desarrolladores a supervisar, probar y desplegar su código.

GitLab comenzó como una alternativa de código abierto auto alojada a GitHub. Ahora también ofrece planes SaaS gratuitos y de pago basados en la nube.
También ofrece herramientas esenciales de gestión de proyectos para supervisar y controlar a los miembros de un equipo. No es solo un sistema de control de versiones para el código fuente de software.

CONCLUSIONES

- En la máquina local usamos git y funciona en la terminal o línea de comandos, ahora bien, si queremos colaborar con otros, usar una interfaz web o publicar nuestros proyectos en la web, usamos GitHub.

- Git y GitHub son dos conceptos que a menudo se confunden. Git es el sistema de control de versiones, mientras que GitHub es una plataforma de alojamiento de repositorios de código fuente que utiliza Git para el control de versiones.

- Para equipos que están distribuidos alrededor del mundo usamos repositorios en la nube. Existen empresas que nos dan toda una plataforma para usar Git en la nube, estos repositorios son ramas, llamadas ramas remotas.
  Tenemos la rama local, que es la master y la rama remota que suele llamarse origin y ese origin puede apuntar a GitHub, GitLab o BitBucket.

- Ambas plataformas, tanto GitHub como Git son útiles, pero para situaciones ligeramente diferentes. Si no está interesado en trabajar con desarrolladores fuera de su equipo, GitLab puede ayudarlo a acelerar un poco su flujo de trabajo. Sin embargo, GitHub puede ser la mejor ruta para aquellos que buscan crecer en sus carreras.