Git & GitHub

Autor: José Guillermo Sandoval Huerta

Fecha: 26 octubre 2023

Índice

Introducción a GIT

Git es un sistema de control de versiones distribuido (DVCS, por sus siglas en inglés) ampliamente utilizado en el desarrollo de software y en la gestión de proyectos que involucran archivos y cambios en el tiempo.

Historia

A principios de la década de 2000, el desarrollo del kernel de Linux estaba en pleno auge. Linus Torvalds y otros desarrolladores trabajaban en colaboración en el núcleo del sistema operativo, pero estaban utilizando un sistema de control de versiones centralizado llamado BitKeeper, este sistema era una herramienta de control de versiones propietaria y gratuita para proyectos de código abierto, pero en 2005, su desarrollador decidió revocar la licencia gratuita, lo que causó un problema para el desarrollo de Linux. Ante la necesidad de una nueva solución, Linus Torvalds decidió crear su propio sistema de control de versiones distribuido que fuera rápido, eficiente y fácil de usar. Comenzó a trabajar en Git en abril de 2005 y lo anunció públicamente en una lista de correo de desarrollo de Linux el 6 de abril.
En diciembre de 2005, Git 1.0 se lanzó públicamente como software de código abierto bajo la Licencia Pública General de GNU (GPL). A partir de ese momento, Git se volvió ampliamente accesible para otros proyectos de código abierto y empresas.
La comunidad de código abierto contribuyó significativamente a la difusión y el éxito de Git. La facilidad de bifurcación (fork) y colaboración en proyectos en GitHub y otros servicios similares impulsó su adopción masiva, debido a su flexibilidad, velocidad y confiabilidad. Grandes plataformas en línea, como GitHub y GitLab, se crearon para alojar repositorios Git, lo que facilitó aún más la colaboración en línea y la gestión de proyectos [1].

Características y funciones

Entre las principales características y funciones de Git se incluyen:

- **Control de versiones:** Git registra y gestiona cambios en los archivos con el tiempo, lo que permite a los usuarios realizar un seguimiento de las modificaciones realizadas en un proyecto.

- **Distribuido:** Git es un sistema de control de versiones distribuido, lo que significa que cada colaborador en un proyecto tiene una copia completa del historial y de los archivos, lo que permite trabajar de forma independiente sin necesidad de una conexión constante a un servidor central.

- **Ramas (Branches):** Git permite la creación de ramas (branches), lo que facilita el trabajo en nuevas características o correcciones de errores sin afectar la rama principal del proyecto (normalmente llamada "master" o "main"). Luego, puedes fusionar (merge) las ramas cuando estén listas.

- **Fusiones (Merges):** Puedes combinar las modificaciones de diferentes ramas en una sola, lo que facilita la colaboración en proyectos de equipo.

- **Histórico detallado:** Git almacena un historial detallado de cada cambio, lo que facilita la identificación de quién hizo qué, cuándo y por qué.

- **Seguridad y confiabilidad:** Los datos en Git están protegidos mediante cifrado y verificación de integridad. Esto asegura que los cambios en los archivos no se corrompan ni se modifiquen inadvertidamente.

- **Colaboración:** Git facilita la colaboración en proyectos, ya que varias personas pueden trabajar en un mismo proyecto sin problemas de conflicto. Los cambios se fusionan de manera eficiente y se resuelven conflictos si es necesario.

- **Gestión de repositorios remotos:** Git permite la gestión de repositorios remotos en servicios como GitHub, GitLab y Bitbucket, lo que facilita la colaboración en línea y el almacenamiento seguro de proyectos.

- **Velocidad y eficiencia:** Git es conocido por su rapidez y eficiencia en operaciones como la clonación, la actualización y la confirmación de cambios.

En resumen, Git es una herramienta esencial en el desarrollo de software y en la gestión de proyectos que involucran archivos en evolución. Facilita el seguimiento, la colaboración y el control de versiones, lo que lo convierte en una elección común para equipos de desarrollo y proyectos individuales [2].

Comandos básicos en GIT

A continuación, se presentan algunos de los comandos más comunes que se necesitan para trabajar con Git.

- **git init:** Inicia un nuevo repositorio Git en un directorio local.

- **git clone [URL]:** Clona un repositorio Git existente desde una URL remota en tu directorio local.

- **git add [archivo(s)]:** Agrega archivos al área de preparación (staging) para que estén listos para ser confirmados. Donde [archivo(s)] = '.' agregara todos los archivos y carpetas actualizados en ese momento.

- **git commit -m "Mensaje del commit":** Confirma los cambios en el área de preparación con un mensaje descriptivo que indica qué cambios se realizaron.

- **git status:** Muestra el estado de los archivos en el directorio de trabajo, incluyendo los cambios no confirmados y los archivos en el área de preparación.

- **git log:** Muestra un registro de los commits realizados en el repositorio, incluyendo información como el autor, fecha y mensaje del commit.

- **git branch:** Muestra una lista de ramas en el repositorio y destaca la rama actual con un asterisco (*).

- **git branch [nombre de la rama]:** Crea una nueva rama con el nombre especificado.

- **git checkout [nombre de la rama]:** Cambia a una rama específica.

- **git merge [nombre de la rama]:** Fusiona una rama con la rama actual.
- **git pull:** Obtiene los cambios más recientes de un repositorio remoto y los fusiona en tu rama local.
- **git push:** Sube los commits locales al repositorio remoto.
- **git remote -v:** Muestra la lista de repositorios remotos configurados en tu proyecto.
- **git diff:** Muestra las diferencias entre los cambios en el directorio de trabajo y el área de preparación.
- **git reset [archivo]:** Quita un archivo del área de preparación, pero mantiene los cambios en el directorio de trabajo.
- **git rm [archivo]:** Elimina un archivo del directorio de trabajo y lo marca como eliminado en el área de preparación.
- **git stash:** Guarda temporalmente cambios no confirmados en una pila (stash) para que puedas trabajar en otra cosa y luego recuperarlos más tarde.

Para obtener información más detallada sobre cualquiera de estos comandos o para explorar comandos adicionales, se puede consultar la documentación de Git o utilizar **git --help** seguido del nombre del comando para obtener información específica sobre ese comando [2].

Trabajar con repositorios en GitHub

Trabajar con repositorios en GitHub implica una serie de acciones y conceptos clave como la gestión de ramas (branches), la fusión (merge) de ramas y la resolución de conflictos cuando varios colaboradores trabajan en el mismo proyecto. A continuación, se explica estos conceptos:

Crear un Repositorio en GitHub

El primer paso para tener un repositorio en GitHub es crearlo, esto es mediante la siguiente sucesión de pasos:

(1) Iniciar sesión en GitHub.

(2) Hacer clic en el botón "+", ubicado en la esquina superior derecha de la pantalla, y seleccionar "New repository".

(3) Ingresar un nombre para el repositorio, una descripción (opcional), elegir la visibilidad (público o privado), y configurar otras opciones según se requiera.

(4) Por último, hacer clic en "Create repository" para crear este nuevo espacio de trabajo.

Clonar un Repositorio

Esta acción en otras palabras significa copiar en nuestra computadora local el contenido de cada uno de los distintos repositorios que se tengan en GitHub de forma que se pueda trabajar en él localmente. Para ello se emplea el siguiente comando en la terminal para clonar un repositorio:

- **git clone URL_del_repositorio**

Ramas (Branches)

Crear una rama permite al colaborador trabajar en una característica o corrección sin afectar la rama principal (generalmente "main" o "master"). Para ello se sigue la siguiente sucesión de pasos:

(En repositorio GitHub)

(1) En GitHub, seleccionar el repositorio el cual se quiera trabajar.

(2) Hacer clic en la pestaña "Code".

(3) Se desplegará un menú que muestra la rama actual (generalmente "main" o "master"). Consecutivamente hacer clic en la opción "Create a new branch".

(4) Proporcionar un nombre a la nueva rama.

(5) Hacer clic en el botón de crear rama.

(6) Y finalmente seleccionar la rama creada para cambiar a la nueva rama creada.

(En terminal Git)

Posicionándose en una rama origen colocar uno de los siguientes comandos y dar enter.

- **git branch [nombre de la rama]** - Crear nueva rama.
- **git checkout -b [nombre de la rama]** - Crear nueva rama y hacer uso de ella.

Realizar Confirmaciones (Commits)

Después de haber hecho cambios en los archivos, se debe de agregar estos cambios al stage con el comando **git add** y confirmarlos con el comando **git commit**. Para ir registrando estos cambios de manera local.

Subir Cambios a GitHub

Para enviar los cambios de manera local al repositorio en línea se emplea el comando **git push**.

Crear una Solicitud de Extracción (Pull Request)

Al estar trabajando en un proyecto colaborativo, puedes crear una solicitud de extracción para proponer cambios a la rama principal. Esto es por medio del repositorio de GitHub, al seleccionar rama trabajada y hacer clic en "New pull request".

Fusionar Cambios (Merge)

El merge es un proceso que combina los cambios de una rama (branch) en otra rama. Esta operación se utiliza comúnmente para incorporar cambios de una rama secundaria (o feature branch) en la rama principal (generalmente llamada "main" o "master"). El proceso de fusión es esencial para la colaboración y el desarrollo de software, ya que permite unir diferentes líneas de desarrollo en un solo flujo de trabajo. A continuación, se expone este proceso:

(1) **Seleccionar la Rama Destino:** Primero, se asegura de estar en la rama destino a la que se desee fusionar los cambios. Por lo general, es la rama principal del proyecto.

(2) **Inicia la Fusión:** Utilizar el comando **git merge** seguido del nombre de la rama que se dese fusionar en la rama destino.

(3) **Resolución de Conflictos (si es necesario):** Si los cambios en la rama que se está fusionando entran en conflicto con los cambios en la rama destino, Git generará conflictos. Estos se deberán de resolver manualmente, editando los archivos en conflicto para elegir qué cambios mantener y cuáles descartar. Una vez resueltos, se debe confirmar los cambios.

(4) **Confirmar la Fusión**: Después de resolver cualquier conflicto, se debe de realizar un **commit** para confirmar la fusión.

(5) **Finalizar la Fusión:** Una vez confirmado la fusión, la rama destino contendrá todos los cambios de la rama a la que se estaba fusionando.

Colaborar con Otros

Otra de las acciones que permite un repositorio GitHub es agregar colaboradores al repositorio y colaborar en proyectos de otros. La colaboración en Git y GitHub es un proceso colaborativo que permite a equipos de desarrollo trabajar juntos de manera eficiente en proyectos de software. Es importante comunicarse, revisar el código y seguir las mejores prácticas de colaboración para garantizar un flujo de trabajo ágil y productivo [3].

Referencias

[1] «1.2. Una breve historia de Git,» Uniwebsidad, [En línea]. Available: https://uniwebsidad.com/libros/pro-git/capitulo-1/una-breve-historia-de-git. [Último acceso: 26 Octubre 2023].

[2] «Git,» Wikipedia, 4 Octubre 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Git. [Último acceso: 26 Octubre 2023].

[3] «Documentación de GitHub,» GitHub, [En línea]. Available: https://docs.github.com/es/get-started/quickstart/create-a-repo. [Último acceso: 26 Octubre 2023].