Buenas practicas en git: evitando errores

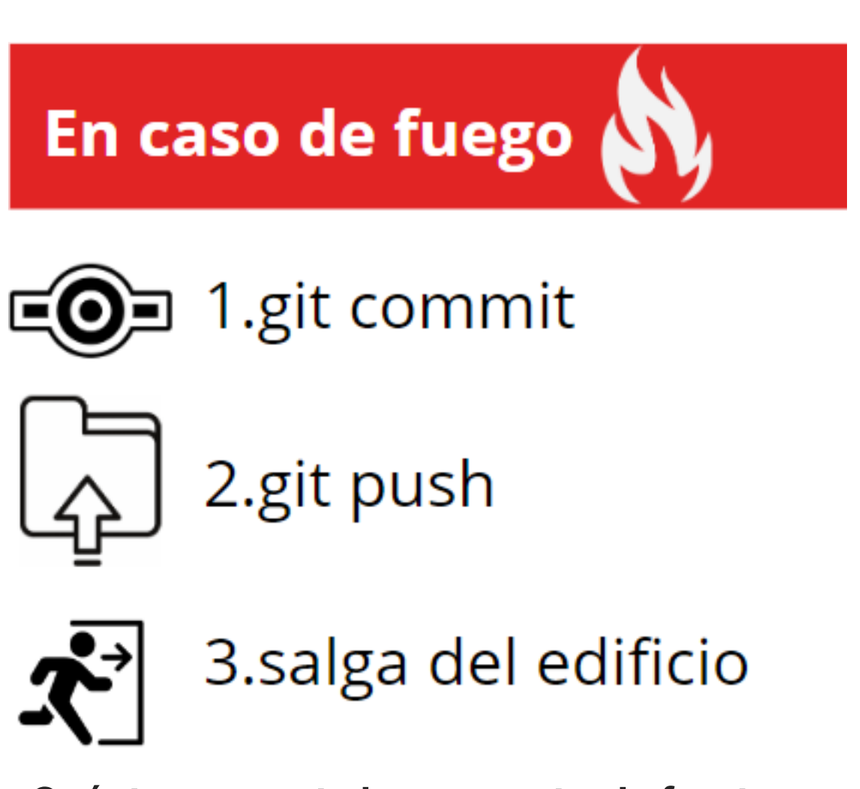
Ingrid Silva
26/07/2023

Este artículo forma parte de una serie de cinco artículos que abordan cómo evitar y corregir errores en Git. Si te interesa profundizar en este tema y aprender a evitar errores comunes en Git, te recomiendo leer los siguientes artículos de la serie:

1. Buenas practicas en git: evitando errores
2. Git: errores de comandos y repositorios
3. Git: errores de commit
4. Git: Errores de fusión
5. Git: Errores con el remoto

Cada artículo cubre diferentes aspectos de Git y ofrece consejos prácticos para evitar errores y optimizar tu flujo de trabajo. Espero que encuentres útil esta secuencia de artículos y que te ayude a mejorar tu experiencia con Git.

¡Disfruta de la lectura!



¿Cuántas veces te has encontrado frente a un mensaje de error en Git que te deja desconcertado y frustrado?

Si eres como la mayoría de los desarrolladores, probablemente más de una vez. Git es una herramienta increíblemente poderosa para el control de versiones, pero también puede ser complicada y propensa a errores si no se utiliza correctamente.

En este artículo, te guiaré a través de los pasos fundamentales para evitar errores en Git y te daré consejos y trucos para que puedas aprovechar al máximo esta herramienta esencial. Ya seas un principiante que acaba de empezar a utilizar Git o un desarrollador experimentado que busca mejorar su fluidez, ¡has llegado al lugar correcto!

1. Lea la documentación de Git

La documentación de Git es una valiosa fuente de información que te ayudará a utilizar Git de manera efectiva, resolver problemas y mantener un flujo de trabajo eficiente. Leerla te permitirá aprovechar al máximo esta poderosa herramienta de control de versiones.

- **La documentación oficial de Git**: El sitio web oficial de Git proporciona documentación completa y detallada sobre todos los aspectos de Git. Aquí encontrarás información sobre comandos, conceptos, configuraciones, flujos de trabajo y mucho más. Se recomienda empezar por la sección "**Git Basics**" para obtener una base sólida sobre los conceptos y comandos fundamentales.
- **Referencia de comandos**: La documentación de Git incluye una referencia completa de todos los comandos disponibles en Git. Puedes consultar la descripción de cada comando, sus opciones y cómo utilizarlo correctamente. Esto es especialmente útil cuando encuentres un comando desconocido o necesites entender en detalle cómo funciona.
- **Manuales de Git**: La documentación de Git también incluye varios manuales específicos sobre temas particulares. Algunos ejemplos son: **Pro Git**, **Git Glossary**,**Git Hooks**,**Git Attributes**,etc.
- **Ejemplos y tutoriales**: Además de la documentación técnica, el sitio web de Git ofrece una sección de "Git Tutorials" con ejemplos prácticos y tutoriales paso a paso. Estos tutoriales cubren una amplia gama de temas, desde los conceptos básicos hasta flujos de trabajo avanzados, colaboración y más.
- **Explorando Git en la línea de comandos**: La documentación de Git te enseñará a utilizar los comandos, pero también es útil para explorar Git directamente desde la línea de comandos. Puedes utilizar la opción `--help` con cualquier comando para obtener información breve sobre su uso y opciones. Por ejemplo, puedes ejecutar `git commit --help` para obtener información sobre el comando `commit`.

Recuerda que la documentación de Git se mantiene actualizada y brinda información precisa y confiable. A medida que adquieras experiencia con Git, es una buena práctica consultar la documentación regularmente para ampliar tus conocimientos, descubrir nuevas funciones y resolver cualquier problema que pueda encontrar.

2. Verificar el estado del repositorio

Antes de realizar cualquier operación en Git, verifica el estado actual del repositorio utilizando el comando `git status`. Esto te dará información sobre los cambios realizados, los archivos modificados y las ramas en las que te encuentras.

Cuando ejecutas `git status`, obtendrás información sobre varios aspectos del repositorio:

- **Rama actual**: `git status` te mostrará en qué rama te encuentras actualmente.
- **Cambios sin confirmar**: Git te mostrará una lista de los archivos modificados que aún no se han confirmado mediante un commit.
- **Archivos preparados para el commit** :Si has utilizado el comando `git add` para preparar archivos para el siguiente commit, `git status` te mostrará una lista de los archivos que están en el área de preparación (***staging area***).
- **Archivos no rastreados**: Si hay archivos en tu directorio de trabajo que no están siendo rastreados por Git, `git status` te mostrará una lista de estos archivos.

Además de proporcionar información sobre el estado actual del repositorio, `git status` también te dará recomendaciones sobre los comandos que puedes ejecutar para manejar esos cambios. Por ejemplo, te puede sugerir usar `git add` para agregar archivos modificados al área de preparación, o `git checkout` para descartar cambios en archivos específicos.

3. Actualizar tu repositorio antes de hacer cambios

Antes de comenzar a trabajar en tu repositorio local, se recomienda ejecutar el comando `git pull` para obtener las últimas actualizaciones del repositorio remoto. Esto evitará conflictos y garantizará que trabajes con la versión más reciente del código.

Cuando ejecutas `git pull`, ocurren dos operaciones principales:

- **Descarga (fetch) de cambios**: El comando `git pull` realiza una operación de descarga (`git fetch`) para obtener las últimas actualizaciones del repositorio remoto. Esta operación descarga los cambios en tu repositorio local sin fusionarlos automáticamente con tu rama actual.
- **Fusión (merge) de cambios**: Después de descargar los cambios, `git pull` realiza una operación de fusión (`git merge`) para combinar los cambios descargados con tu rama local. Durante esta fusión, Git intentará combinar los cambios automáticamente. En caso de conflictos, donde los cambios descargados y los cambios locales se superponen en el mismo archivo o línea, Git marcará los conflictos y requerirá tu intervención para resolverlos manualmente.

Recuerda que `git pull` realiza una descarga y una fusión automática, lo que implica cambios en tu repositorio local. Siempre es importante revisar los cambios realizados después de ejecutar `git pull` para asegurarte de que todo esté en orden antes de continuar con tu trabajo.

4. Hacer commits frecuentes y significativos

Realiza commits con frecuencia y asegúrate de que cada commit represente una unidad lógica de cambios significativos. Esto facilitará la comprensión de los cambios realizados y ayudará a deshacerlos si es necesario.

- **Claridad y comprensión de los cambios**: Hacer commits frecuentes y significativos ayuda a mantener un historial claro y coherente de los cambios realizados en tu repositorio.
- **Reversión y deshacer cambios**: Si algo sale mal o necesitas deshacer los cambios realizados, los commits frecuentes y significativos facilitan la reversión.
- **Colaboración y trabajo en equipo**: Hacer commits frecuentes y significativos mejora la colaboración entre los miembros del equipo.
- **Claridad en los mensajes de commit**: Al hacer commits frecuentes, también es importante proporcionar mensajes de commit claros y descriptivos. Un buen mensaje de commit debe ser conciso pero informativo, explicando de manera sucinta qué cambios se realizarán y por qué.
- **Mantenimiento y gestión del código**: Los commits frecuentes y significativos facilitan el mantenimiento y la gestión del código a largo plazo.

Los cambios en un commit deben estar relacionados entre sí y tener un propósito común. Si tienes cambios que pertenecen a diferentes aspectos o funcionalidades, considera dividirlos en commits separados para mantener la claridad y la coherencia.

5. Utilizar ramas separadas y evitar cambios directos en la rama principal *(master/main)*

Esta estrategia te ayudará a mantener un flujo de trabajo organizado, facilitando la colaboración con otros desarrolladores y minimizando los riesgos de conflictos y errores.

- **Evitar cambios directos a la rama principal**: La rama principal, generalmente llamada `master` o `main`, es la rama principal de tu repositorio y se considera estable. Evitar cambios directos en esta rama ayuda a garantizar que el código en producción sea confiable y funcional. En su lugar, utiliza ramas separadas para realizar cambios y desarrollar nuevas características.
- **Utilizar ramas separadas para el desarrollo**: Al utilizar ramas separadas, puedes aislar los cambios y trabajar en ellos de manera independiente. Esto te permite desarrollar nuevas funciones, solucionar problemas o realizar experimentos sin afectar directamente a la rama principal. Además, trabajar en ramas separadas facilita el seguimiento de los cambios y simplifica la colaboración con otros miembros del equipo.
- **Ramas con características separadas**: Si estás trabajando en una función o característica específica, es recomendable crear una rama separada para su desarrollo. Esto te permite mantener el código relacionado con esa función en una rama dedicada y evitar interferencias con otros cambios en desarrollo. Al finalizar la implementación y probar la nueva función, puedes fusionar la rama de características con la rama principal utilizando una operación de fusión (`merge`) o `rebase`.
- **Fusiones (merges) y rebase**: Una vez que hayas desarrollado y probado tus cambios en una rama separada, es importante incorporarlos a la rama principal de manera adecuada. Puedes utilizar la operación de fusión (`merge`) o `rebase` para integrar los cambios en la rama principal.

Al utilizar ramas separadas y realizar fusiones o rebase para incorporar los cambios en la rama principal, puedes mantener un flujo de trabajo organizado y minimizar los riesgos de conflictos y errores. Además, esta práctica facilita la revisión y prueba de los cambios antes de que se integren con la rama principal, lo que contribuye a la calidad y estabilidad del código.

6. No combine archivos binarios o generados

Evita incluir archivos binarios o generados automáticamente (por ejemplo, archivos compilados, archivos de dependencias) en tu repositorio. Estos archivos pueden causar conflictos y dificultar la colaboración en el proyecto.

- **Conflictos y dificultades en la colaboración**: Los archivos binarios o generados automáticamente, como archivos compilados, archivos de dependencias o archivos de construcción específicos de plataformas, tienen la característica de cambiar cada vez que se generan.
- **Tamaño y rendimiento del repositorio**: Los archivos binarios tienden a tener un tamaño mayor en comparación con los archivos de texto plano.
- **Reproducibilidad y mantenibilidad**: El propósito de utilizar un sistema de control de versiones como Git es rastrear y administrar el código fuente y los archivos relacionados. Los archivos binarios o generados automáticamente no forman parte del código fuente principal y pueden generarse nuevamente a partir del código fuente y las instrucciones de construcción adecuadas.

Para manejar adecuadamente estos archivos, siga estas practicas:

- **Utilizar archivos `.gitignore`**: El archivo `.gitignore` te permite especificar patrones de nombres de archivos o directorios que Git debe ignorar al realizar operaciones como `add` o `commit`.
- **Separar archivos generados de archivos fuente**: Almacena los archivos generados en un directorio separado y agrega ese directorio al archivo `.gitignore`.
- **Proporcionar instrucciones de construcción**: En lugar de incluir los archivos binarios en el repositorio, proporciona instrucciones claras sobre cómo generar esos archivos a partir del código fuente y las dependencias necesarias.

Recuerda que cada proyecto puede tener requisitos específicos, por lo que es importante evaluar el contexto y las necesidades de tu proyecto al decidir qué archivos binarios o generados automáticamente deben excluirse del repositorio.

Si deseas profundizar tus conocimientos sobre Git, te invito a realizar nuestro curso "Git y GitHub: repositorio, commit y versiones". Este curso, te brindará una sólida comprensión de los fundamentos de Git y GitHub. Aprenderás habilidades clave para administrar repositorios, realizar commits y manejar versiones de manera efectiva.

Además, te recomiendo que explores más sobre los errores comunes en Git. Este artículo es parte de una secuencia de cinco artículos centrados en los errores en Git:

- Buenas practicas en git: evitando errores
- Git: errores de comandos y repositorios
- Git: errores de commit
- Git: Errores de fusión
- Git: Errores con el remoto

Cada artículo aborda diferentes aspectos de Git y proporciona consejos prácticos para evitar errores. Te animo a profundizar en su lectura para mejorar tu dominio de Git. ¡Disfruta aprendiendo!

Ingrid Silva

Estudiante de Ciencias de la Computación en el Instituto Federal de Brasilia - IFB y se desempeña como desarrolladora Full-Stack. Forma parte del Scuba Team en Alura Latam, donde se especializa en programación. Su pasión por el conocimiento tecnológico es insaciable, ya que encuentra fascinante todas las áreas que lo componen.