

## **Issues: Excelencia en la colaboración en GitHub**

GitHub Issues se presenta como una herramienta multifacética dentro del ecosistema de GitHub, esencial para orquestar esfuerzos colaborativos. Funciona no solo como un lugar para reportar problemas, sino como un sistema integral para rastrear varios tipos de tareas y actividades relacionadas con tus proyectos. Esto incluye gestionar errores, proponer mejoras y monitorear otras tareas esenciales dentro de tus repositorios de GitHub.

En el entorno DevOps, GitHub Issues juega un papel crucial al facilitar la retroalimentación continua y la colaboración sin interrupciones. Actúa como una plataforma transparente y eficiente donde los desarrolladores pueden señalar problemas, los miembros del equipo pueden sugerir nuevas características y los interesados pueden participar en discusiones significativas sobre posibles mejoras. Esta funcionalidad está en perfecta armonía con los principios fundamentales de DevOps, que enfatizan la eliminación de barreras organizacionales, la promoción de la comunicación abierta y el fomento de una cultura de mejora continua y adaptación.

Al aprovechar GitHub Issues, los equipos pueden crear un espacio compartido y accesible que fomente la colaboración y asegure que todos los involucrados en un proyecto estén en la misma página. No se trata solo de rastrear problemas; se trata de construir un entorno dinámico y receptivo donde las ideas puedan florecer y gestionarse de manera eficiente.

### **Lo que hace único a GitHub Issues**

Veamos ahora el papel único de GitHub Issues en fomentar la transparencia y mejorar la experiencia del desarrollador.

GitHub Issues se destaca como una herramienta única en el panorama del desarrollo de software, particularmente en su enfoque hacia la transparencia y la colaboración. Esta herramienta, integral para GitHub, ha redefinido cómo los equipos de desarrollo, y de hecho, las comunidades de código abierto más amplias, se comunican y colaboran en proyectos. La importancia de GitHub Issues no radica solo en su funcionalidad como herramienta de seguimiento de errores o solicitudes de características, sino en su papel en cultivar un enfoque abierto, transparente y comunitario para el desarrollo de software, haciendo eco del espíritu del movimiento de código abierto.

### **La importancia de la transparencia en el desarrollo**

La transparencia en el desarrollo de software trata de hacer visible y comprensible para todos los interesados involucrados, desde desarrolladores hasta usuarios finales, todo el proceso de creación, modificación y mantenimiento del software. Esta transparencia es crucial por varias razones:

- **Mejora de la colaboración:** Cuando todos los aspectos de un proyecto son visibles, los miembros del equipo pueden colaborar de manera más efectiva. Todos tienen acceso a la misma información, lo que lleva a una mejor toma de decisiones y una comprensión compartida de los objetivos y desafíos.
- **Mayor responsabilidad:** La transparencia lleva a una asignación más clara de responsabilidades. Los miembros del equipo son más responsables de su trabajo cuando es visible para otros, fomentando un sentido de propiedad y compromiso.

- Mejora de la calidad y la innovación: El acceso abierto a los datos del proyecto permite que más ojos estén sobre el proyecto, lo que resulta en más retroalimentación, ideas y críticas. Este escrutinio colectivo no solo mejora la calidad, sino que también estimula la innovación.
- Construcción de confianza: La transparencia construye confianza entre los miembros del equipo y con los interesados externos, incluidos los usuarios y clientes. La confianza es crucial para el éxito a largo plazo del proyecto y para establecer un software confiable y centrado en el usuario.

### **GitHub Issues: Un catalizador para la colaboración transparente**

GitHub Issues ejemplifica este enfoque transparente. A diferencia de herramientas que permiten permisos detallados y jerárquicos, GitHub Issues típicamente opera en un modelo de acceso más abierto. Cada issue, su hilo de discusión y las decisiones tomadas son visibles para todos los miembros del equipo, y a menudo para el público en proyectos de código abierto. Esta apertura previene los silos de información y fomenta una cultura de abajo hacia arriba donde las ideas y la retroalimentación pueden provenir de cualquier nivel dentro de la organización o la comunidad.

Este enfoque se alinea perfectamente con el espíritu del desarrollo de código abierto, que valora la contribución comunitaria, la responsabilidad compartida y el diálogo abierto. Al adoptar un modelo similar internamente, las empresas pueden cosechar los beneficios de este enfoque de código abierto, eliminando los silos organizacionales y fomentando una atmósfera comunitaria dentro de los equipos. Alienta a los desarrolladores a tomar la iniciativa, contribuir con ideas y participar en debates saludables y constructivos.

### **El código abierto como modelo para la colaboración interna**

El enfoque de trabajo de código abierto, facilitado por herramientas como GitHub Issues, es una excelente estrategia para mejorar la experiencia del desarrollador. Trae la naturaleza colaborativa y transparente de las comunidades de código abierto al funcionamiento interno de una organización. Los desarrolladores se sienten más comprometidos y valorados cuando pueden ver el impacto de su trabajo y contribuir a discusiones más allá de sus tareas inmediatas. Este entorno abierto nutre un sentido de comunidad, mejora la moral y puede impulsar significativamente la innovación y la productividad.

Además, la transparencia y la apertura fomentadas por GitHub Issues y el modelo de código abierto proporcionan oportunidades de aprendizaje invaluable. Los desarrolladores pueden aprender unos de otros, obtener perspectivas diferentes y crecer al exponerse a una variedad de desafíos y soluciones. Este entorno es propicio para el desarrollo personal y profesional, crucial para retener talento y mantener a los equipos motivados y productivos.

### **GitHub Issues: Un catalizador de colaboración**

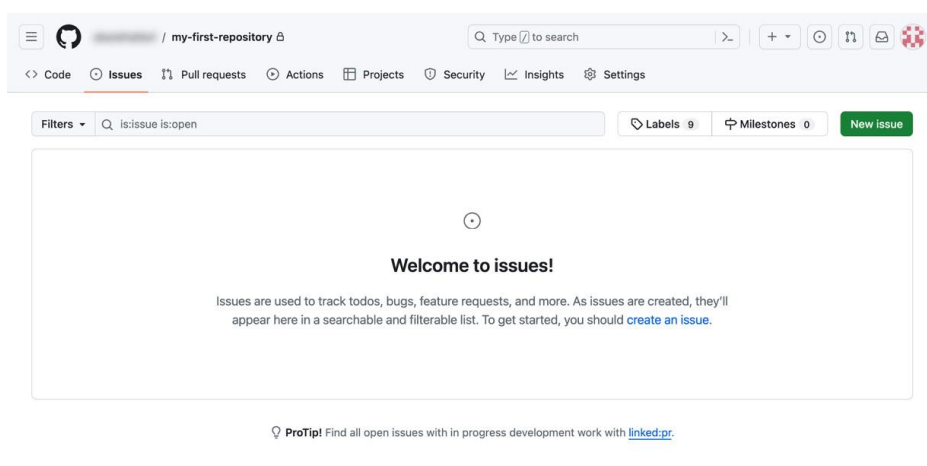
En resumen, GitHub Issues juega un papel fundamental en la promoción de la transparencia y un enfoque comunitario para el desarrollo de software. Al inspirarse en las prácticas de código abierto, ayuda a eliminar las barreras organizacionales, fomenta una cultura de trabajo colaborativa y transparente y mejora significativamente la experiencia del desarrollador. **En una era donde el desarrollo de software se trata cada vez más de comunidad y colaboración, GitHub Issues se erige como un faro, guiando a los equipos hacia una forma de trabajo más abierta, inclusiva y efectiva.**

Echemos un vistazo a un issue desde esa perspectiva.

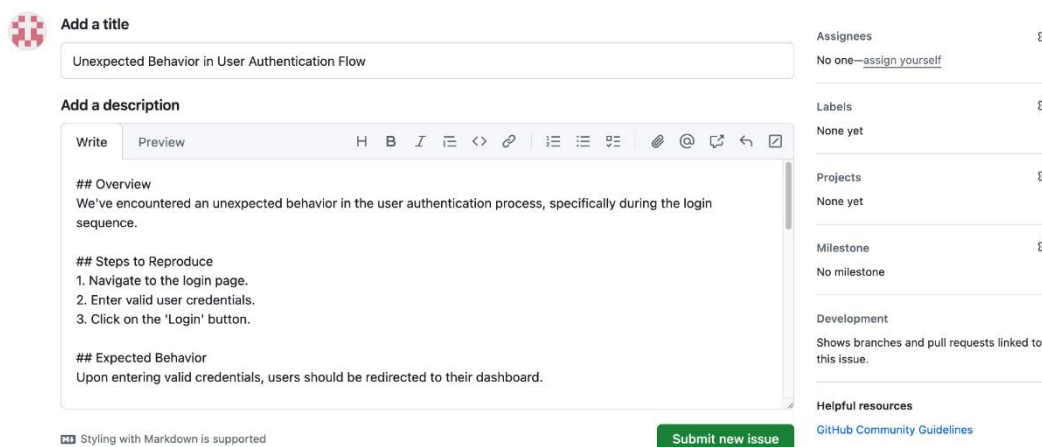
## Crear un issue: Elementos esenciales para un issue bien estructurado

Crear tu primer issue en GitHub puede parecer inicialmente un desafío debido a su simplicidad, pero dominar esta habilidad es crucial para una colaboración efectiva. Un issue bien estructurado es clave: debe ser claro, conciso y accionable. El objetivo es proporcionar suficiente contexto para que tu punto sea comprensible sin bombardear a tus colaboradores con información excesiva. Comienza identificando claramente el problema, explicando su importancia y delineando el resultado deseado.

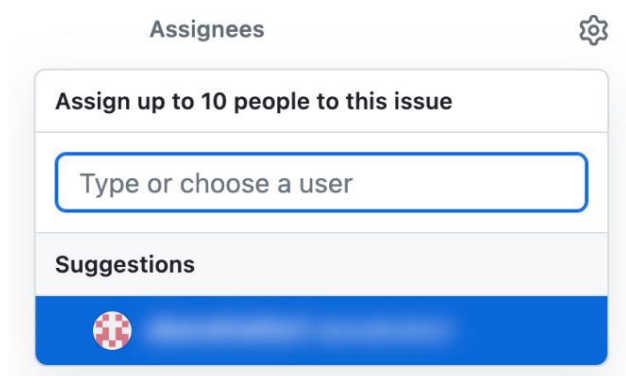
El proceso comienza en tu repositorio de GitHub. Si tienes un repositorio existente o acabas de crear uno, encontrarás la pestaña "Issues" en el menú del repositorio. Aquí, puedes iniciar un nuevo issue:



Crear un issue es sencillo. La interfaz presenta campos para el título y la descripción principal, junto con opciones de metadatos como asignatarios y etiquetas. El enfoque debe estar en el contenido del issue. GitHub admite Markdown para formateo de documentación, por lo que es beneficioso familiarizarte con la sintaxis de Markdown. Sin embargo, recuerda que la simplicidad es clave: Markdown no es tan rico en funciones como Microsoft Word, pero es perfecto para crear documentación clara y directa:

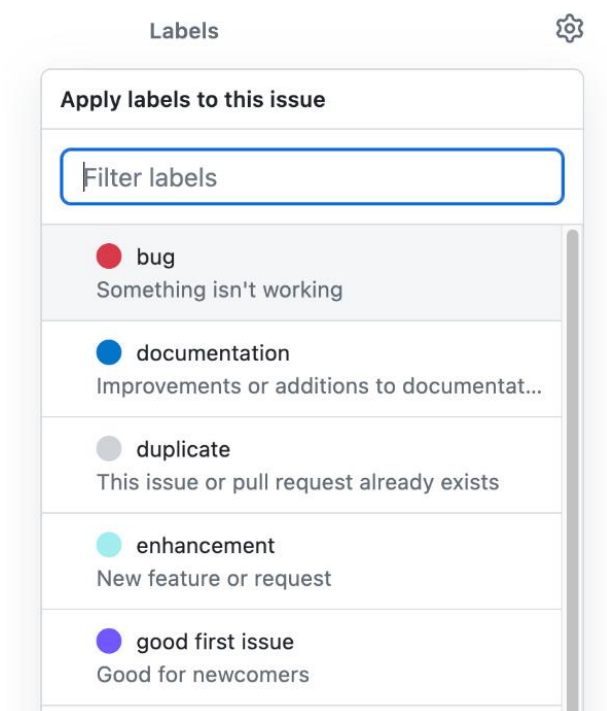


Dentro de un issue, puedes asignarlo a miembros del equipo y también mencionar directamente a individuos o equipos en el contenido para notificaciones:

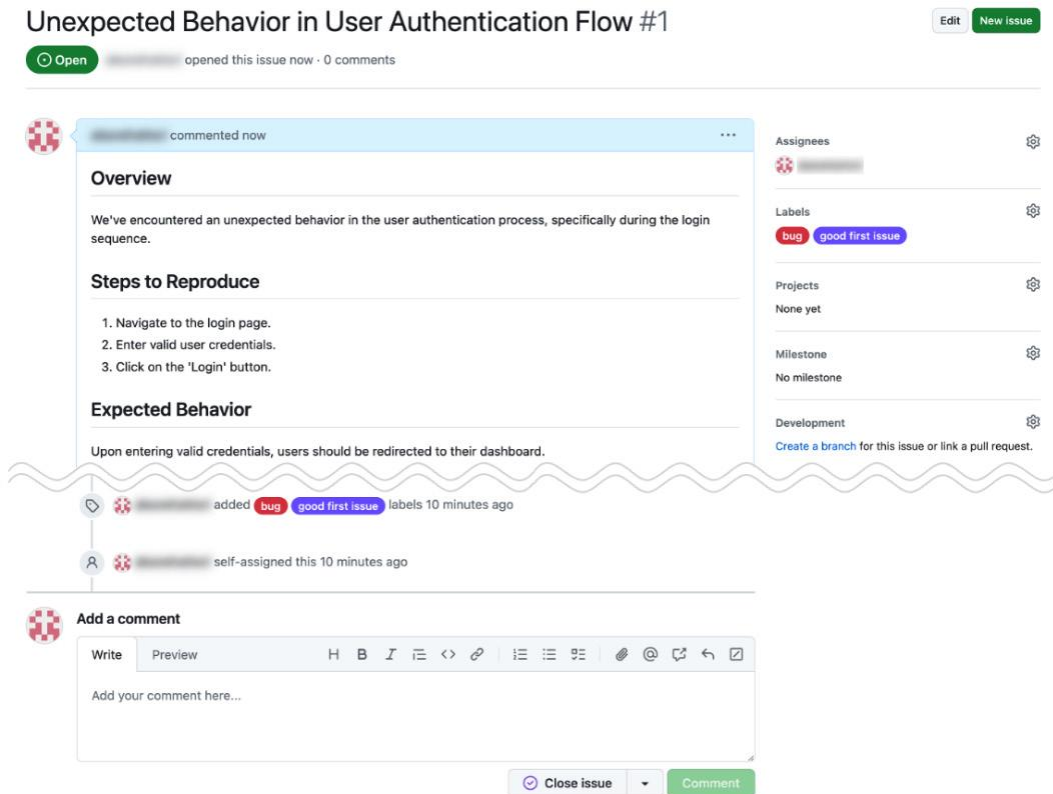


Etiquetar tareas (por ejemplo, como un bug, documentación o mejora) también es posible. Si bien se pueden crear etiquetas personalizadas, es recomendable comenzar con las etiquetas predeterminadas y mejorarlas gradualmente. El uso excesivo de etiquetas puede llevar a confusión y desafíos de categorización. Si tu equipo u organización tiene estándares específicos de etiquetado, es mejor adherirse a ellos.

El enfoque de usar issues en GitHub no es de arriba hacia abajo, sino que fomenta un estilo comunitario y de abajo hacia arriba. Imponer reglas estrictas desde el principio puede limitar la libertad necesaria para fomentar una cultura abierta, ágil y colaborativa. El equilibrio es crucial; a medida que tú y tu equipo se acostumbren al flujo de trabajo de GitHub, puedes ajustar tu enfoque en consecuencia:



Finalmente, verás el issue enviado:



Entonces, como puedes ver, gestionar issues en GitHub es sencillo. La clave radica en fomentar la colaboración y la comunicación.

Una buena comunicación al crear issues en un proyecto es fundamental para evitar malentendidos y facilitar la colaboración. Esto implica escribir títulos claros, descripciones detalladas y respuestas ordenadas.

Por ejemplo el título debe ser breve pero descriptivo, evitando vaguedades como "Problema". Por ejemplo, "Arreglar enlace roto en README". Usar términos como "Arreglar", "Mejorar" u "Optimizar" para indicar la naturaleza del issue. Se debe captar el problema principal sin detalles innecesarios. Centrarse en el issue, no en quién lo reportó. Incluir el componente afectado si es relevante. Usar etiquetas para indicar prioridad en lugar del título.

### Descripciones efectivas

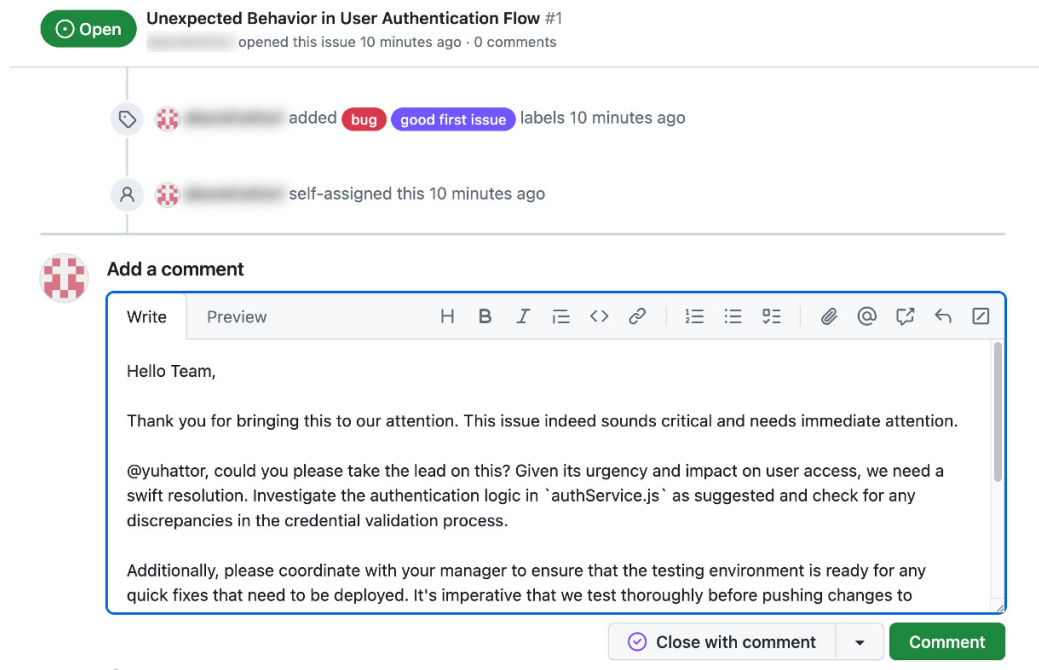
- **Contexto y claridad:** Iniciar con el contexto y definir el problema o mejora específica.
- **Resultado y reproducción:** Incluir el resultado deseado y pasos para reproducir el problema.
- **Markdown y visuales:** Usar Markdown para estructurar la descripción y mejorar la legibilidad. Incluir capturas de pantalla o enlaces si es necesario.
- **Foco y colaboración:** Mantenerse en el tema y fomentar la retroalimentación.

### Respuestas a issues para una cultura colaborativa

Los issues en GitHub son más que una herramienta para rastrear errores; sirven como un centro para la colaboración y la documentación. Cuando respondes a un issue, no solo estás abordando un

problema o consulta específica, sino que también estás contribuyendo a la documentación pasiva del proyecto.

Responder a un issue es fácil. Simplemente escribes tu respuesta en Markdown y respondes. Pero incluso aquí, necesitas tener una filosofía:



En GitHub, comentar en un issue es realmente importante, no solo una respuesta de correo. Veamos las siguientes perspectivas:

- Documentación pasiva: La documentación pasiva es un concepto que surge del contexto de InnerSource, donde la documentación no se crea activamente, sino que evoluciona naturalmente a través de interacciones dentro del repositorio. En GitHub, cada issue, pull request e hilo de discusión se convierte en parte de esta documentación pasiva. Este proceso es orgánico y de abajo hacia arriba; a medida que los miembros del equipo se involucran en conversaciones, solicitan características, resuelven problemas o implementan soluciones, sus interacciones se registran. Esto crea un registro completo y en evolución de decisiones, discusiones y cambios, contribuyendo a un documento vivo que captura la historia y el razonamiento del proyecto.
- Centralización de la comunicación para la transparencia: En un entorno colaborativo, la comunicación a menudo ocurre a través de varios canales como Slack, Teams, Jira y GitHub. Sin embargo, centralizar las discusiones relacionadas con el proyecto en GitHub tiene ventajas significativas. No solo crea una valiosa documentación, sino que también fomenta una cultura transparente. Todos los procesos de toma de decisiones y colaboraciones son visibles y accesibles para todos los miembros del equipo, fomentando la inclusividad y la comprensión. Cuando las conversaciones ocurren en otras plataformas como Slack, guiar esas discusiones de regreso a GitHub asegura que la información y las decisiones cruciales se documenten y compartan dentro de todo el equipo. Este enfoque previene los silos y asegura que todos, independientemente de cuándo se unan al proyecto, tengan acceso a la misma información.

- Abrazando una comunicación positiva e inclusiva: La colaboración efectiva se basa en una comunicación positiva e inclusiva. Al responder a los issues, es importante reconocer y elogiar el buen comportamiento, agradecer a los miembros del equipo por sus contribuciones y mantener un tono respetuoso y de apoyo. Esto no solo fomenta una dinámica de equipo saludable, sino que también mejora la moral y promueve un sentido de pertenencia y apreciación. La comunicación inclusiva significa asegurarse de que la voz de todos sea escuchada y valorada y que sus contribuciones sean reconocidas. Este enfoque no solo fortalece la cohesión del equipo, sino que también impulsa una mejor resolución de problemas e innovación.

En GitHub, responder a issues es más que solo una respuesta; es una parte integral de la construcción de documentación pasiva, el fomento de la transparencia del equipo y la promoción de una comunicación positiva e inclusiva. Al tratar las respuestas a issues como contribuciones a un documento vivo, creamos una rica historia de la evolución del proyecto.

Centralizar la comunicación en GitHub asegura que todos los miembros del equipo tengan acceso equitativo a la información, lo que ayuda en la toma de decisiones transparentes. Por último, abrazar un estilo de comunicación positivo fortalece los lazos del equipo y asegura que todos se sientan valorados y escuchados. Este enfoque holístico para responder a issues no solo se trata de resolver problemas, sino de construir una cultura de equipo fuerte, inclusiva y transparente.

## **Excelencia en Pull Requests**

La función de pull requests de GitHub se erige como una innovación fundamental en el panorama del desarrollo de software, una que puede ser acreditada con moldear significativamente el movimiento de software de código abierto (OSS). Su introducción marcó un momento transformador, redefiniendo cómo se realizan la colaboración, integración de código y aseguramiento de calidad (QA) en proyectos de software, particularmente en OSS.

Los pull requests en GitHub son más que solo una función; son un mecanismo fundamental para la colaboración en el mundo del desarrollo de software. **Un pull request es esencialmente una solicitud para fusionar un conjunto de cambios de una rama de un repositorio a otra, típicamente de una rama de características o temática a la rama principal o master. Pero el significado de los pull requests va mucho más allá de la mera fusión de código; son un nexo para la discusión, revisión y refinamiento de código en un entorno de proyecto colaborativo.**

### **¿Qué hace único a los pull requests?**

El modelo de pull request transformó la codificación colaborativa. Desplazó el enfoque de parches de código individuales enviados en aislamiento hacia un enfoque más interactivo y orientado a la comunidad. Los desarrolladores ahora podían no solo enviar cambios, sino también participar en discusiones sobre esos cambios directamente dentro de la plataforma de GitHub.

Esto fomentó una cultura de revisión colaborativa y retroalimentación continua, esencial para mantener la alta calidad del código y alinear las contribuciones con los objetivos del proyecto. Los pull requests introdujeron un método estructurado y transparente para revisiones de código. Los cambios de código ahora son fácilmente visibles y discutibles dentro del contexto de un pull request, lo que permite una retroalimentación más detallada y productiva. Este proceso asegura que los

cambios sean minuciosamente revisados, lo que lleva a una mayor calidad del código y proyectos de software más robustos.

Los pull requests no solo han revolucionado la forma en que se revisa y fusiona el código, sino que también han agilizado significativamente los flujos de trabajo de desarrollo en GitHub. Al integrarse perfectamente con características nativas de GitHub y una gran cantidad de herramientas de terceros, apoyan y mejoran las prácticas de Integración Continua (CI) y Despliegue Continuo (CD). Esta integración transforma el proceso de desarrollo, haciéndolo más eficiente, confiable y automatizado.

Un desarrollo fascinante en el uso de los pull requests es la integración de pruebas previas al lanzamiento dentro del propio hilo del pull request. Este avance ha llevado a un enfoque más completo y cohesivo para garantizar la calidad del código y la gestión de proyectos. Cada vez que un desarrollador inicia un pull request, se desencadena una serie de verificaciones automáticas y pruebas. Estas pueden ir desde evaluaciones de calidad del código y escaneos de vulnerabilidades de seguridad hasta pruebas de rendimiento. Los resultados de estas pruebas se muestran directamente en el pull request, proporcionando retroalimentación inmediata y accionable.

### **Más allá de la línea de comandos**

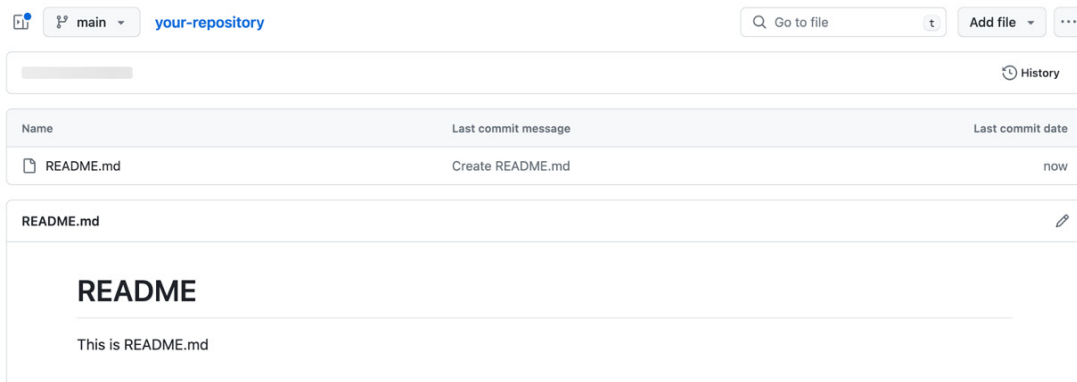
Cuando se discute el contexto histórico y la evolución de los pull requests de GitHub, es importante tener en cuenta que el término "pull request" en sí tiene raíces en el comando `git request-pull` en Git. Esta conexión podría llevar a algunos a ver la contribución de GitHub como principalmente proporcionar una interfaz de usuario para un concepto existente en lugar de inventar algo completamente nuevo. Sin embargo, una exploración más profunda de la historia y el desarrollo de los pull requests revela un impacto más significativo.

GitHub tomó una función básica de la línea de comandos y la transformó en una característica rica, interactiva y colaborativa dentro de su interfaz web. Esta transformación no se trató solo de agregar una capa de interfaz de usuario; se trató de reimaginar cómo se podría realizar la colaboración en torno a los cambios de código.

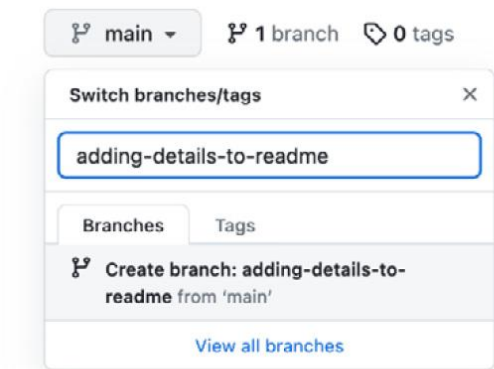
### **Crear un pull request**

Comienza seleccionando o creando un repositorio en GitHub. Para esta demostración, nos centraremos en la interfaz de usuario de GitHub en lugar de la interfaz de línea de comandos. Considera un repositorio con solo un archivo `README.md` como punto de partida. El objetivo es agregar contenido detallado a este archivo y enviar un pull request para fusionarlo:

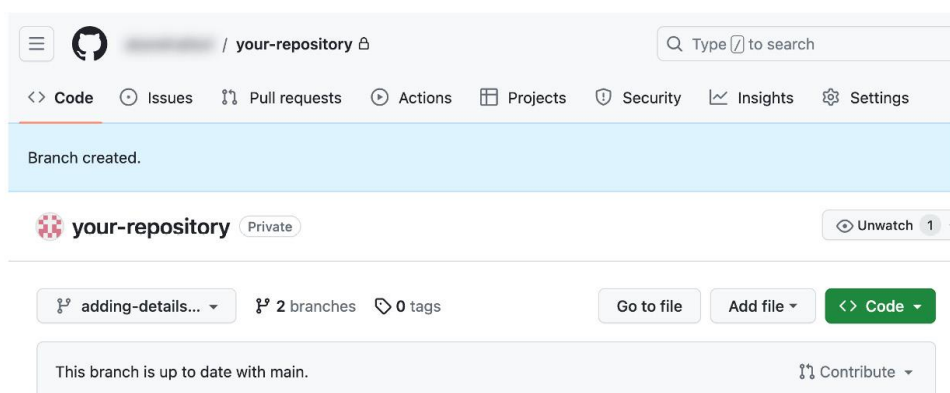




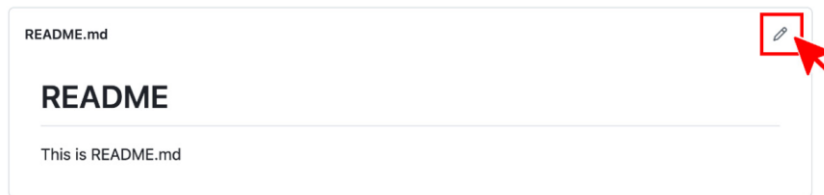
El primer paso implica crear una rama llamada `adding-details-to-readme`. Esta rama es donde harás tus ediciones antes de fusionarlas en la rama principal. Si bien puedes crear y subir nuevas ramas utilizando comandos de Git, GitHub también ofrece una interfaz intuitiva para este propósito. Utiliza el botón desplegable de ramas en la esquina superior izquierda para crear una rama derivada de la rama principal:



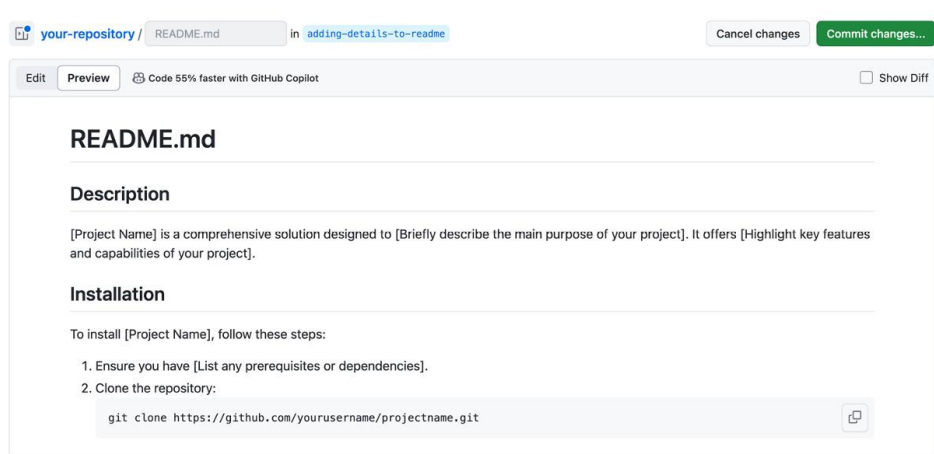
Una vez que se crea una rama, puedes hacer cambios en cualquier rama, no solo en las ramas principales o master:



Luego, editemos README.md. Asegúrate de estar en la rama correcta verificando el menú desplegable en la esquina superior izquierda antes de comenzar tus ediciones. Luego, usa el botón de edición en la esquina superior derecha para comenzar a editar el archivo README:



El archivo README.md es la cara y la puerta de entrada de tu repositorio, dando la bienvenida a los miembros del equipo y nuevos colaboradores. Aquí, deja volar tu imaginación y crea un README atractivo:



Bueno, ahora estamos listos. Hagamos que tu commit se refleje. Como ya estás en la rama para editar, puedes confirmar tus ediciones tal como están. Sin embargo, si abres la ventana de edición y confirmas mientras aún estás en la rama principal, aún tienes la opción de crear una nueva rama. Pero es más seguro crear conscientemente una rama inicialmente y confirmar. Por otro lado, hay una configuración de política de protección de ramas en GitHub que evita que confirmes directamente en el entorno compartido del equipo, como main, production y release más adelante.

El título se convierte en el mensaje del commit de Git, por lo que piensa en esto cuidadosamente:

Commit changes

×

Commit message

Update README.md example

Extended description

Add an optional extended description..

☒ Commit directly to the adding-details-to-readme branch


☐ Create a **new branch** for this commit and start a pull request

[Learn more about pull requests](#)

Cancel


Commit changes

Una vez que se confirmen tus cambios, probablemente verás una alerta en la página de inicio del repositorio para crear un pull request. También puedes crear un pull request desde el botón "New pull request" si no aparece una alerta o deseas seleccionar una rama diferente:


 your-repository Private


Unwatch


1

 adding-details-to-readme had recent pushes less than a minute ago

Compare & pull request


 adding-details... ▾

 2 branches


 0 tags



Go to file

Add file ▾


 Code ▾


This branch is 1 commit ahead of main.

 Contribute ▾

  Update README.md example

0a2b10d now

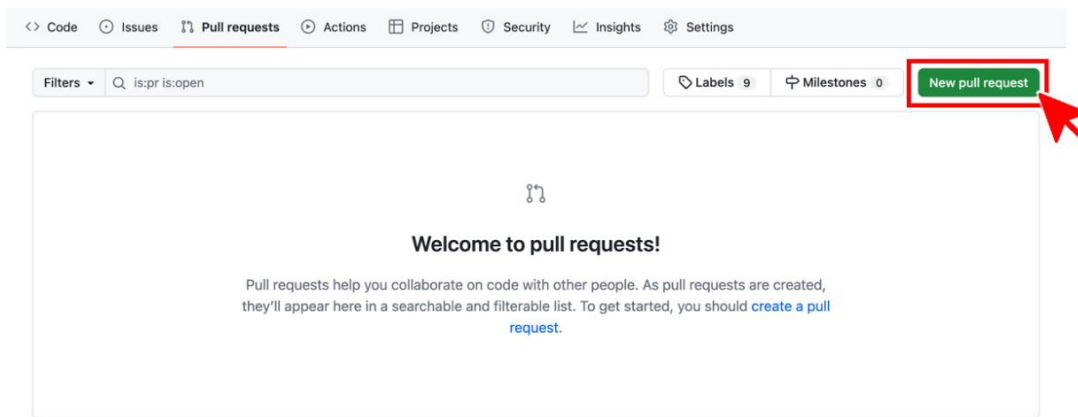
 2 commits

 README.md

Update README.md example

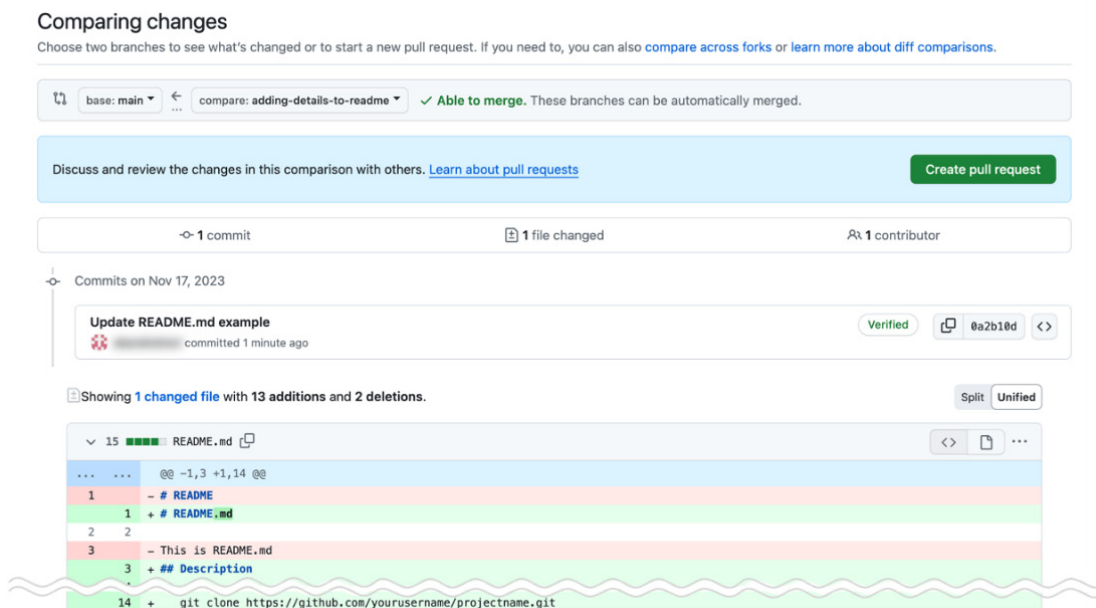
now

Si no ves esta alerta o deseas seleccionar una rama diferente, puedes iniciar el mismo proceso haciendo clic en el botón "New pull request" en la pestaña "Pull requests":



Al navegar por la interfaz, puedes encontrarte en la página de pull request ya sea siguiendo una alerta o haciendo clic en el botón "New pull request". Independientemente de cómo llegues, este es el lugar donde revisarás el historial de tus cambios.

La forma en que se presentan los cambios en GitHub puede variar. Como se ilustra en la captura de pantalla, las diferencias (o diffs) pueden mostrarse en línea, intercaladas dentro de cada línea del código:



Alternativamente, GitHub ofrece una vista dividida, mostrando las diferencias lado a lado. Esta vista bifurcada permite una comparación más clara entre el código original y el modificado:



Cuando estés satisfecho, escribamos tu pull request. Un buen pull request incluye un título claro, una descripción detallada y revisores designados. En cuanto a los revisores, puedes especificar personas o equipos. El contenido aquí es algo similar a lo que escribirías en un issue:

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: main compare: adding-details-to-readme ✓ Able to merge. These branches can be automatically merged.

**Add a title**

Add README.md detail for Project Documentation

**Add a description**

Write Preview H B I ≡ < > ↺ ↻

Hi @Reviewer\_1

**\*\*Description:\*\***

This pull request introduces the `README.md` file, which serves as the primary documentation for [Project Name]. The README includes a concise description of the project, highlighting its purpose, key features, and

This documentation aims to offer an initial overview for potential users and contributors, making it easier for them to understand and get started with [Project Name].

**Create pull request**

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

**Reviewers**

No reviews

**Assignees**

**Labels**

documentation

**Projects**

milestone

No milestone

**Development**

Use [Closing keywords](#) in the description to automatically close issues

**Helpful resources**

[GitHub Community Guidelines](#)

→ 1 commit 1 file changed 1 contributor

Commits on Nov 17, 2023

**Update README.md example** Verified 0a2b10d < >

committed 9 minutes ago

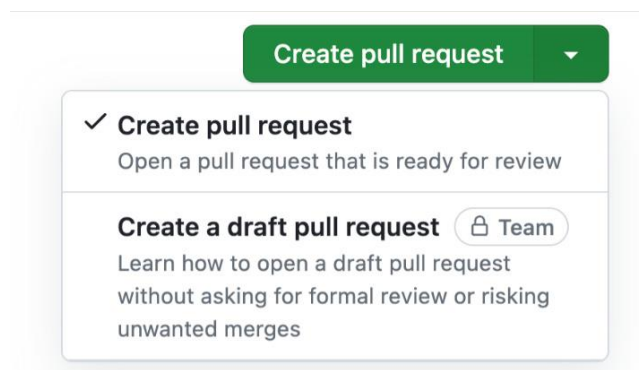
Showing 1 changed file with 13 additions and 2 deletions.

Line	Base	Target
...	@@ -1,3 +1,14 @@	
1	- # README	
		14 + git clone https://github.com/yourusername/projectname.git

Ahora, tus habilidades de escritura son puestas a prueba una vez más. El contenido a considerar al escribir aquí es el mismo que lo que nos preocupaba con respecto a los issues en la sección anterior. Escribe en un formato Markdown fácil de leer. Sin embargo, en este contexto, es crucial entender los objetivos distintos de estas herramientas. El objetivo principal de un pull request es ser fusionado, típicamente dirigido a personas específicas autorizadas para aprobarlo. En contraste, un issue puede estar dirigido a una audiencia más amplia e indefinida. Por lo tanto, al escribir una descripción para este pull request, es crucial considerar quién lo revisará y cómo escribirlo.

Si la revisión la llevará a cabo una persona específica, podría haber cierta comprensión compartida o contexto, lo que podría reducir la necesidad de comentarios extensos en la revisión. La técnica básica es la misma que para un issue, pero es importante adaptar tu enfoque a tu audiencia objetivo.

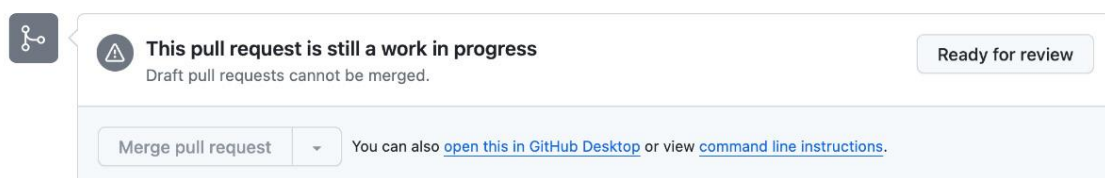
Puede que notes que hay dos opciones disponibles al crear un pull request, las cuales se pueden acceder a través del botón desplegable a la derecha. Puedes crear un pull request estándar u optar por un pull request en borrador (draft). Los pull requests en borrador pueden no ser aplicables a repositorios privados dependiendo de tu plan, pero cualquiera puede intentarlo para repositorios públicos.



Tanto los pull requests en borrador como los regulares no difieren mucho en cuanto a la interfaz de usuario. Sin embargo, un pull request en borrador se utiliza para indicar que el pull request aún no está finalizado o que el trabajo general aún está en progreso, incompleto o en una etapa específica de desarrollo.

Permite revisiones intermedias, lo que significa que no está destinado a una revisión formal aún. Los revisores generalmente aprecian recibir resultados o informes intermedios. Las revisiones tempranas pueden proporcionar información valiosa y compartir el progreso con el equipo, lo cual es beneficioso.

En el caso de un pull request en borrador, la fusión no se puede hacer de inmediato por error, como se muestra a continuación:

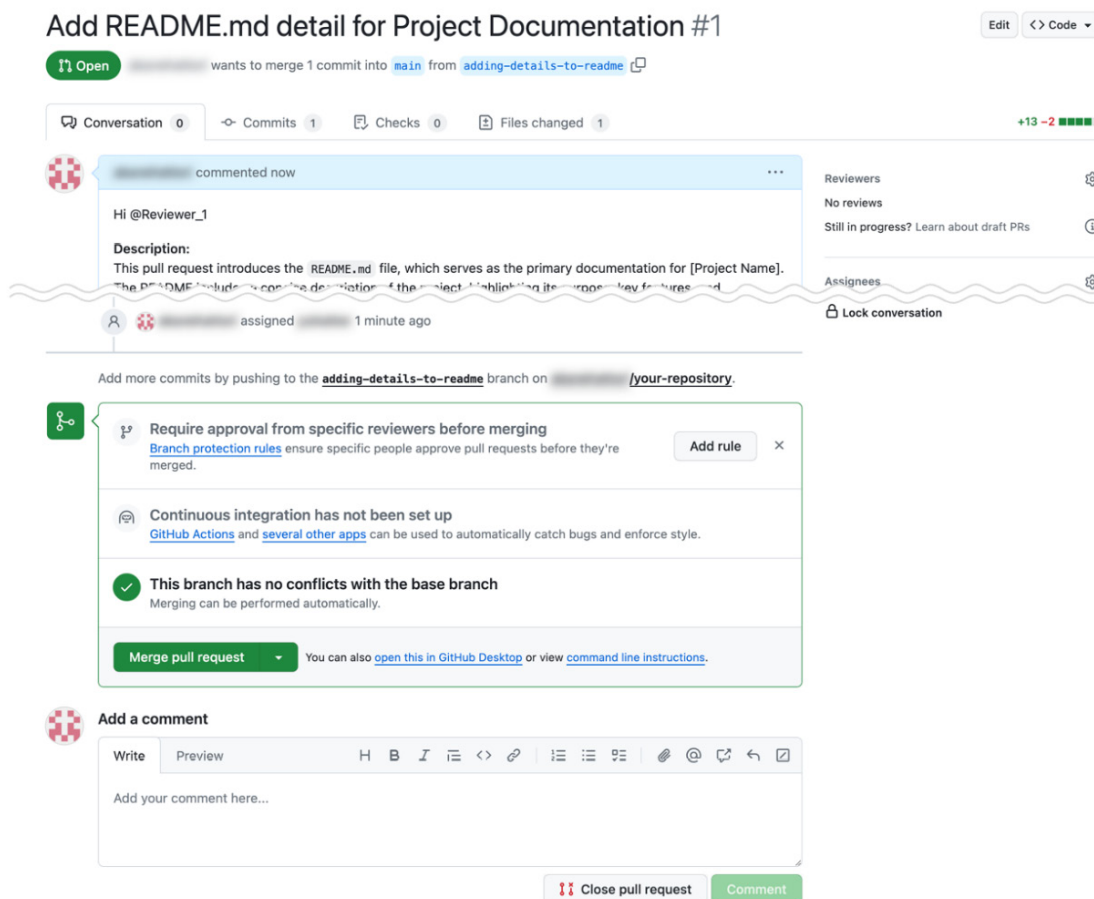


En un escenario extremo, puedes crear un pull request incluso sin escribir una sola línea de código con el comando `git empty commit`. Esto puede ser increíblemente útil desde las primeras etapas del desarrollo, permitiendo a los miembros del equipo o a un compañero de programación en pares

revisar el código, ayudando a identificar errores temprano. Por lo tanto, se recomienda encarecidamente aprovechar también los pull requests en borrador.

Un commit vacío en Git es un commit que no contiene ningún cambio; es como enviar un mensaje sin alterar ningún archivo. Esto puede ser particularmente útil para iniciar discusiones en las primeras etapas del desarrollo sin la necesidad de cambios reales en el código. Para crear un commit vacío, puedes usar el comando `git commit --allow-empty -m "Your message"`, que te permite hacer un commit con un mensaje pero sin cambios de contenido. Esta función es útil para crear pull requests que sirvan como marcadores de posición para futuras revisiones de código o para marcar hitos específicos en el proceso de desarrollo. Se recomienda tener en cuenta este commit vacío junto con el pull request en borrador.

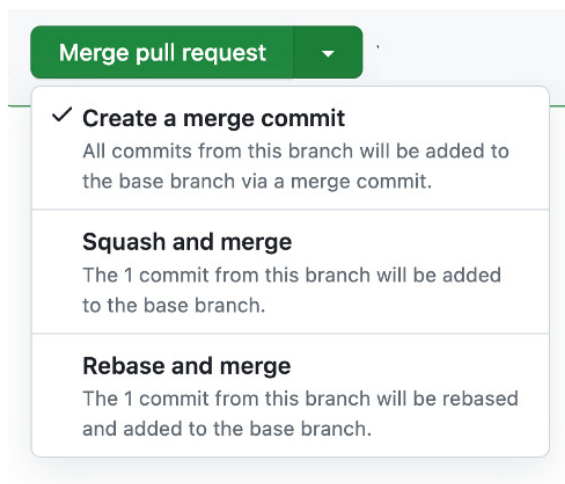
Ahora, una vez que hayas realizado una comunicación adecuada y completado la revisión, es momento de fusionar. Lo grandioso de GitHub aquí no es la interfaz de comandos, sino la capacidad de realizar fusiones en la interfaz de usuario. Puedes seguir adelante y fusionar tal como está, pero echemos un vistazo a las opciones antes de hacerlo:



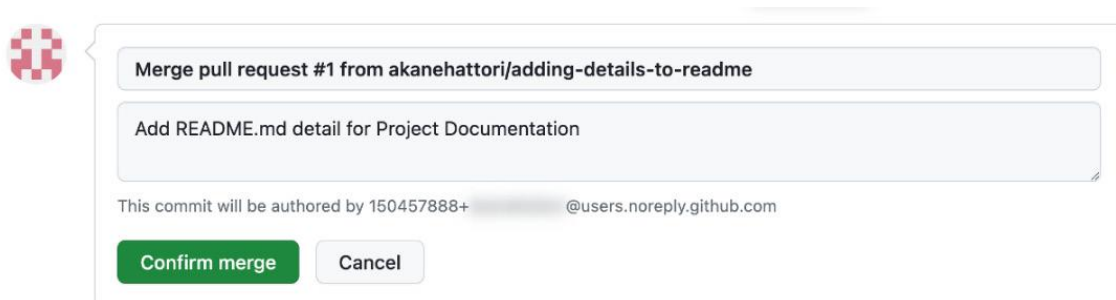
Al inspeccionar el menú desplegable "Merge", encontrarás tres opciones. Estas son crear un merge commit, luego hacer un squash para combinar commits en uno y fusionar, y finalmente, realizar una rebase merge. Si bien estas técnicas de fusión se cubrieron antes, la belleza de GitHub radica en la flexibilidad que ofrece para elegir cada una de estas opciones para gestionar tu historial de Git. Si deseas preservar el historial de contribuciones, primero incorporarás los commits, asegurando que se conserve la mayor parte del historial de cambios posible. Por otro lado, si prefieres un historial

más limpio, opciones como "Squash and merge" y "Rebase and merge" también están disponibles para considerar.

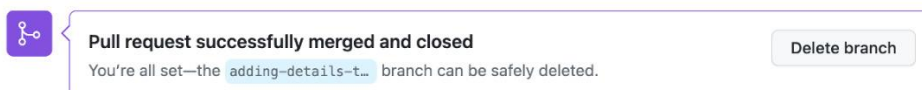
Es importante tener en cuenta, sin embargo, que la opción "Rebase and merge" de GitHub no es idéntica al comando tradicional git rebase. "Rebase and merge" en GitHub efectivamente reaplica cada commit de la rama de características en la rama base individualmente, sin crear un commit de fusión. Esto permite un historial lineal mientras se preserva el orden cronológico de los commits. Ya sea que busques mantener un registro detallado de las contribuciones o prefieras un historial simplificado, GitHub proporciona las herramientas para adaptarse a las necesidades de tu proyecto:



Esta vez, elegimos crear un merge commit. Crear un merge commit es fácil; simplemente escribe un título y una descripción. Luego, haz clic en el botón "Confirm merge" para fusionar:



Luego, confirmarás que la fusión se realizó con éxito en la plataforma GitHub:



Y eso para empezar.