**TUTORIAL GITHUB**

Soft =>

"elimina" los commits posteriores al commit al que estas haciendo el reset,

conserva los cambios en el stage area,

conserva los cambios que tengas en tus archivos (working directory)

Mixed =>

“elimina los commits posteriores al commit al que estas haciendo el reset”

“Deshace los cambios en el stage area”

“conserva los cambios que tengas en tus archivos (working directory)”

Hard=>

"elimina los commits posteriores al commit al que estas haciendo el reset”

“Deshace los cambios en el stage area”

“Deshace los cambios que tengas en tus archivos (working directory)”

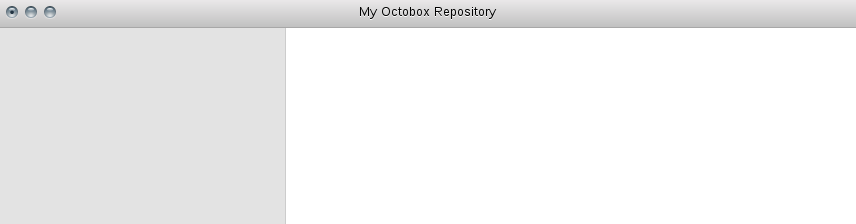
**1. Inicializar un repositorio**

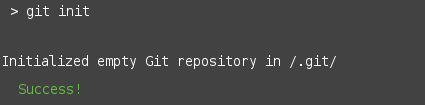
*Directorio***:**

Una carpeta utilizada para almacenar múltiples archivos.

*Repositorio*:

Un directorio donde Git se ha inicializado para iniciar la versión que controla tus archivos.





**2. Comprobación del estado (git status)**

¡Buen trabajo! Como Git nos acaba de decir, nuestro directorio "octobox" ahora tiene un repositorio vacío en /.git/. El repositorio es un directorio oculto donde opera Git.

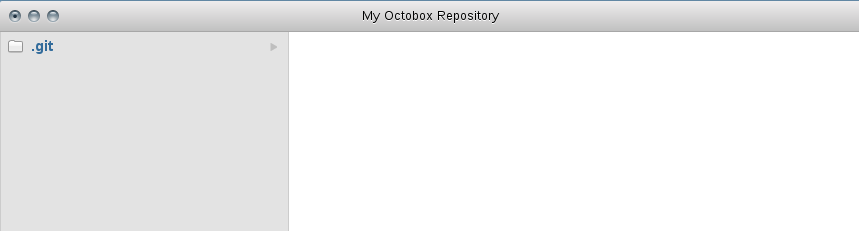
Para guardar su progreso a medida que avanza en este tutorial, y gane una insignia cuando lo complete con éxito, diríjase a crear una cuenta gratuita de Code School. Te esperaremos aquí.

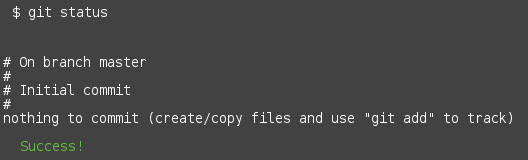
A continuación, escriba el comando git status para ver cuál es el estado actual de nuestro proyecto:

*El directorio .git*

A la izquierda, verás un directorio .git. Por lo general, está oculto, pero te lo mostramos por conveniencia.

Si hace clic en él, verá que tiene todo tipo de directorios y archivos dentro. Casi nunca necesitarás hacer nada aquí, pero son las entrañas de Git, donde sucede toda la magia.





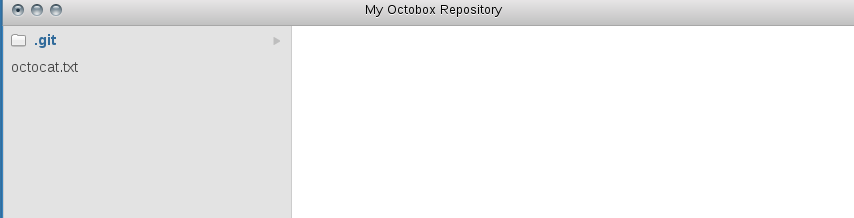
**3. Agregar y comprometer**

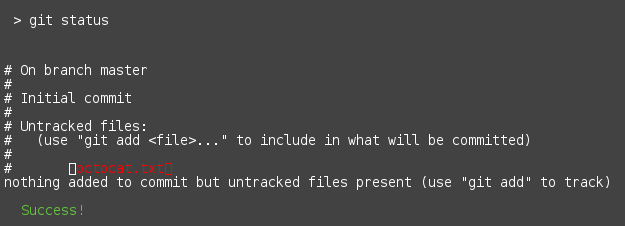
Creé un archivo llamado octocat.txt en el repositorio de octobox para usted (como puede ver en el navegador a continuación).

Debería ejecutar nuevamente el comando git status para ver cómo ha cambiado el estado del repositorio:

Tip:

Es recomendable ejecutar **git status** a menudo. A veces las cosas cambian y no te das cuenta.





**4. Agregar cambios (git add octocat.txt)**

Bueno, parece que nuestro repositorio Git está funcionando correctamente. ¿Nota cómo Git dice que *octocat.txt* está "sin seguimiento"? Eso significa que Git ve que octocat.txt es un archivo nuevo.

Para decirle a Git que comience a rastrear los cambios realizados en octocat.txt, primero debemos agregarlo al área de ensayo usando «**git add»**.

*Staged*:

Los archivos están listos para ser comprometidos.

*Unstaged*:

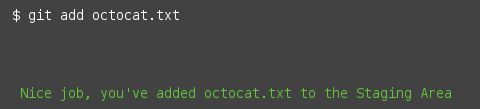
Archivos con cambios que no han sido preparados para ser comprometidos.

*Untracked* (sin seguimiento):

Los archivos aún no son rastreados por Git. Esto generalmente indica un archivo recién creado.

*Deleted* (eliminado):

El archivo ha sido eliminado y está esperando ser eliminado de Git.



**5 Comprobación de cambios (git status)**

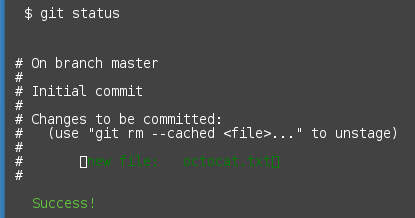
¡Buen trabajo! Git ahora está rastreando nuestro archivo octocat.txt. Vamos a ejecutar el estado de git nuevamente para ver dónde nos encontramos:

*Añadir todo:*

También puede escribir «**git add -A** .» donde el punto representa el directorio actual, por lo que se agrega todo dentro y debajo de él. El **«-A»** asegura que incluso se incluyen las eliminaciones de archivos.

*Restablecimiento de git:*

Puede usar **«git reset <nombre de archivo>»** para eliminar un archivo o archivos del área de ensayo.



**6 Compromiso**

¿Te das cuenta de cómo Git dice que los cambios deben ser cometidos? Los archivos enumerados aquí están en el *Área de ensayo* y aún no están en nuestro repositorio. Podríamos agregar o eliminar archivos del escenario antes de almacenarlos en el repositorio.

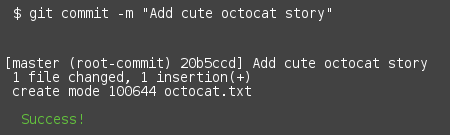
Para almacenar nuestros cambios por etapas, ejecutamos el comando de confirmación (commit) con un mensaje que describe lo que hemos cambiado. Hagámoslo ahora escribiendo:

*Área de ensayo:*

Un lugar donde podemos agrupar archivos juntos antes de "comprometerlos" con Git.

*Cometer*

Un "commit" es una instantánea de nuestro repositorio. De esta forma, si alguna vez tenemos que mirar hacia atrás en los cambios que hemos realizado (o si alguien más lo hace), veremos una buena línea de tiempo de todos los cambios.



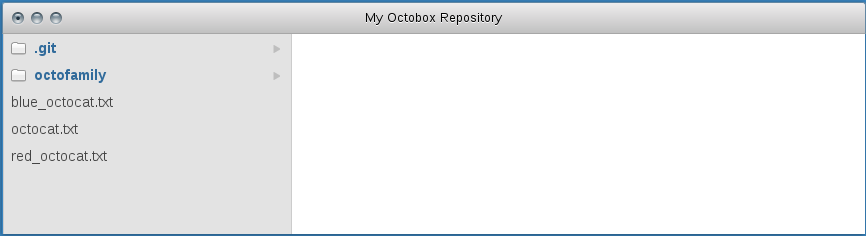
**7. Agregar todos los cambios**

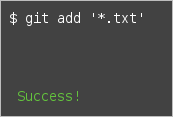
¡Estupendo! También puede usar comodines si desea agregar muchos archivos del mismo tipo. Tenga en cuenta que he agregado un grupo de archivos «***.txt*** »en su directorio a continuación.

Puse algunos en un directorio llamado "*octofamily*" y algunos otros terminaron en la raíz de nuestro directorio "*octobox*". Afortunadamente, podemos agregar todos los archivos nuevos usando un comodín con ***git add***. No te olvides de las **comillas**!

*Comodines*:

Necesitamos «**comillas»** para que git recibirá el comodín antes de que nuestro Shell pueda interferir con él. Sin comillas, nuestra shell sólo ejecutará la búsqueda de comodín en el directorio actual. Git recibirá la lista de archivos que el shell encontró en lugar del comodín y no podrá agregar los archivos dentro del directorio octofamily





**8. Cometer todos los cambios**

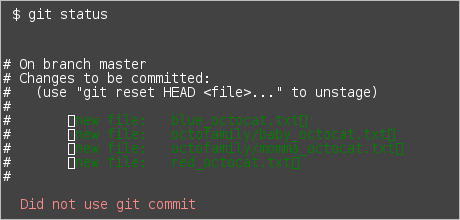
Bien, has agregado todos los archivos de texto al área de staging. Siéntase libre de ejecutar git status para ver lo que está a punto de cometer.

Si se ve bien, adelante y corre:

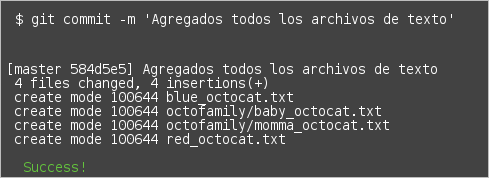
¡Comprueba todas las cosas!

Cuando se utilizan comodines, se debe tener mucho cuidado al hacer commits. Asegúrese de comprobar qué archivos y carpetas se escenifican mediante git status antes de hacer el commit real. De esta manera usted puede estar seguro de que está cometiendo sólo las cosas que desea.

Git status:



Ejecución:



**9. Historia**

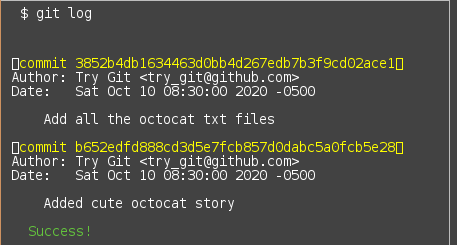
Así que hemos hecho algunos commits. Ahora vamos a navegar por ellos para ver lo que hemos cambiado.

Afortunadamente para nosotros, hay ***git log***. Piense en el registro de Git como un diario que recuerda todos los cambios que hemos cometido hasta ahora, en el orden en que los cometimos. Intente ejecutarlo ahora:

*Registros más útiles:*

Use ***git log --summary*** para ver más información para cada commit. Puede ver dónde se agregaron nuevos archivos por primera vez o dónde se eliminaron los archivos. Es una buena visión general de lo que está sucediendo en el proyecto.

Ejecución:



**10. Repositorios remotos**

¡ gran trabajo! Hemos hecho un nuevo repositorio vacío de github para que lo uses con try git en <https://github.com/try-git/try_git.git>. Para empujar nuestro repositorio local al servidor de github necesitaremos agregar un repositorio remoto.

Este comando toma un nombre remoto y una URL del repositorio, que en su caso es <https://github.com/try-git/try_git.git>..

Siga adelante y ejecute «***git remote add»*** con las siguientes opciones:

*remoto git*:

A Git no le importa qué nombre le den a sus controles remotos, pero es típico nombrar su ‘origin’ principal.

También es una buena idea que su repositorio principal esté en un servidor remoto como GitHub en caso de que su máquina se pierda en el mar durante un crucero transatlántico o sea aplastada por tres estatuas de mono durante un terremoto.



**11. Empujar remotamente**

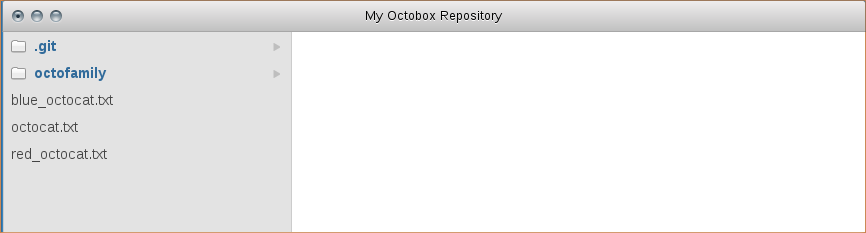
El comando Push le dice a GIT donde poner nuestros commits cuando estemos listos, y ahora estamos listos. Así que vamos a empujar nuestros cambios locales en nuestro repositorio de origen (en GitHub).

El nombre de nuestro control remoto es “origin” y el nombre de rama local por defecto es “master”. El-u le dice a GIT que recuerde los parámetros, por lo que la próxima vez que podamos simplemente ejecutar «**git push»** y git sabrá qué hacer. ¡adelante, empuja!

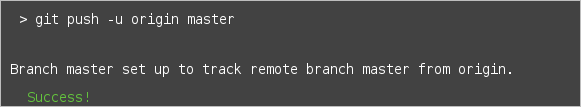
*Cosas interesantes*:

Cuando empiezas a dominar git, puedes hacer cosas realmente geniales con "**hooks**" cuando empujas (push).

Por ejemplo, puede cargar directamente a un servidor web cada vez que presione hacia su control remoto maestro en lugar de tener que cargar su sitio con un cliente ftp. Consulte "Personalizar Git - Git Hooks" para obtener más información.



Ejecución:



**12. Tirando de forma remota**

Supongamos que ha pasado un tiempo. Hemos invitado a otras personas a nuestro proyecto de GitHub que han sacado sus cambios, hecho sus propios *commits* y los han empujado (*push*).

Podemos verificar los cambios en nuestro repositorio GitHub y desplegar (bajar) cualquier cambio nuevo ejecutando:

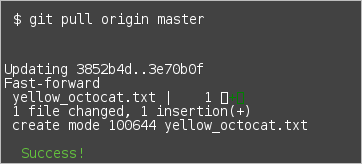
Consejo

git stash:

Algunas veces, cuando vas a tirar, puedes tener cambios que no quieres cometer todavía. Una opción que tiene, además de comprometerse, es esconder los cambios.

Usa el comando 'git stash' para esconder tus cambios, y 'git stash apply' para volver a aplicar tus cambios después de tu extracción.

Ejecución:



**13. Diferencias**

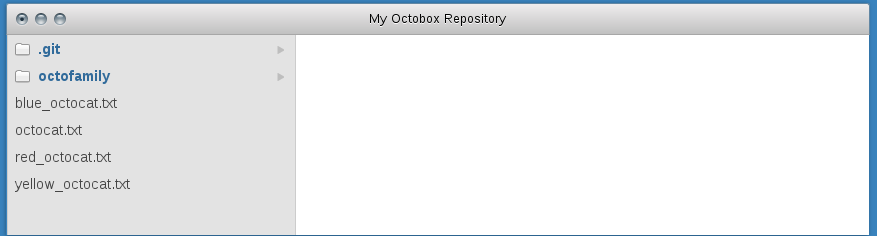
Uh oh, parece que ha habido algunas adiciones y cambios en la familia octocat. Echemos un vistazo a lo que es diferente de nuestro último compromiso utilizando el comando git diff.

En este caso, queremos la diferencia de nuestra confirmación más reciente, a la que podemos hacer referencia utilizando el puntero HEAD.

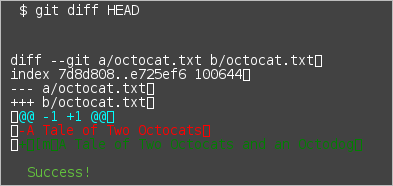
Consejo

*HEAD*

HEAD es un puntero que mantiene tu posición dentro de tus diferentes commits. Por defecto, HEAD apunta a su commit más reciente, por lo que puede usarse como una forma rápida de referenciar ese commit sin tener que buscar el SHA.



Ejecución:



1.14 Diferencias por etapas

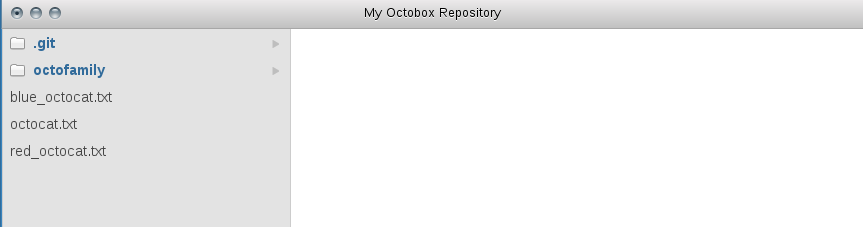
Otro gran uso de diff es observar los cambios dentro de los archivos que ya han sido organizados. Recuerde, los archivos escalonados son archivos que le hemos dicho a git que están listos para ser confirmados.

Usemos git add para la etapa octofamily / octodog.txt, que acabo de agregar a la familia para usted.

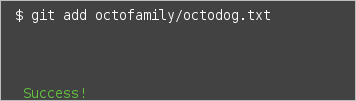
Consejo

Commit Etiquette:

Desea intentar mantener los cambios relacionados juntos en commits separados. El uso de 'git diff' le brinda una buena visión general de los cambios que ha realizado y le permite agregar archivos o directorios de uno en uno y enviarlos por separado.

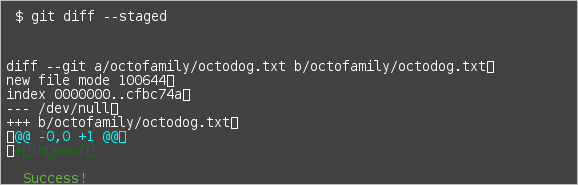


Ejecución:



1.15 Diferencias por etapas (cont.)

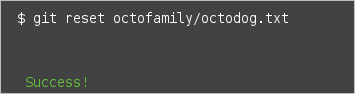
Bien, ahora adelante y ejecute **git diff** con la opción **--staged** para ver los cambios que acaba de realizar. Debería ver que se creó octodog.txt.



1.16 Restableciendo el escenario

Entonces, ahora que el octodog es parte de la familia, octocat está deprimido. Ya que amamos a octocat más que a octodog, volveremos a fruncir el ceño quitando octodog.txt.

Puede desempatar archivos usando el comando git reset. Continúa y elimina octofamily / octodog.txt.



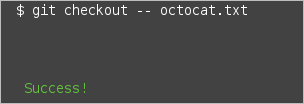
1.17 Deshacer

El reinicio de git hizo un excelente trabajo al desglosar octodog.txt, pero notará que todavía está allí. Él ya no está escenificado más. Sería genial si pudiéramos volver a cómo estaban las cosas antes de que llegara el octodog y arruinara la fiesta.

Los archivos se pueden cambiar a cómo estaban en la última confirmación mediante el comando: **git checkout -- <target>**. Continúa y deshazte de todos los cambios desde el último commit para octocat.txt

Los '--'

Entonces, usted se estará preguntando, ¿por qué tengo que usar esto **'--'**? **git checkout** parece funcionar bien sin eso. Simplemente está prometiendo a la línea de comandos que no hay más opciones después del **'--'**. De esta manera, si tienes una rama llamada octocat.txt, aún revertirá el archivo, en lugar de cambiar a la rama del mismo nombre.



**18 ramificación hacia fuera**

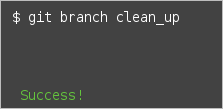
Cuando los desarrolladores están trabajando en una característica o error, a menudo crean una copia (también conocida como **rama**) de su código para poder hacer commits por separado. Luego, cuando terminen, pueden fusionar esta rama nuevamente en su rama **master** principal.

Queremos eliminar todos estos **octocats** molestos, así que vamos a crear una rama llamada **clean\_up**, donde haremos todo el trabajo:

Ramificación

Las ramas son lo que ocurre naturalmente cuando desea trabajar en múltiples funciones al mismo tiempo. No querrá terminar con una rama principal que tenga la característica A medio hecha y la Función B medio hecha.

Más bien, separarías la base del código en dos "snapshots" (ramas) y trabajarías y te comprometerías con ellas por separado. Tan pronto como uno esté listo, puede fusionar esta rama nuevamente en la rama principal y enviarla al servidor remoto.



**19. Ramas de conmutación**

¡Estupendo! Ahora, si escribe **git branch**, verá dos ramas locales: una rama principal llamada **master** y su nueva rama llamada **clean\_up**.

Puedes cambiar de ramas usando el comando git checkout <branch>. Pruébelo ahora para cambiar a la rama **clean\_up**:

De repente

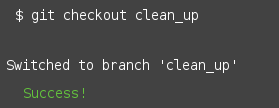
Puedes usar:

git checkout -b new\_branch

para cambiar y crear una rama al mismo tiempo. Esto es lo mismo que hacer:

git branch new\_branch

git checkout new\_branch



**20. Eliminando todas las cosas**

Ok, entonces estás en la rama **clean\_up**. Finalmente puede eliminar todos esos molestos octocats utilizando el comando **git rm**, que no solo eliminará los archivos reales del disco, sino que también eliminará los archivos para nosotros.

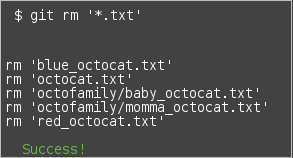
Vas a querer usar un comodín de nuevo para obtener todos los octocats en un barrido, ve y ejecuta:

Eliminar todas las cosas!

La eliminación de un archivo es genial y todo, pero ¿qué sucede si desea eliminar una carpeta completa? Puede usar la opción recursiva en git rm:

**git rm -r folder\_of\_cats**

Esto removerá recursivamente todas las carpetas y archivos del directorio dado.



**21. Cambios de rama comprometidos**

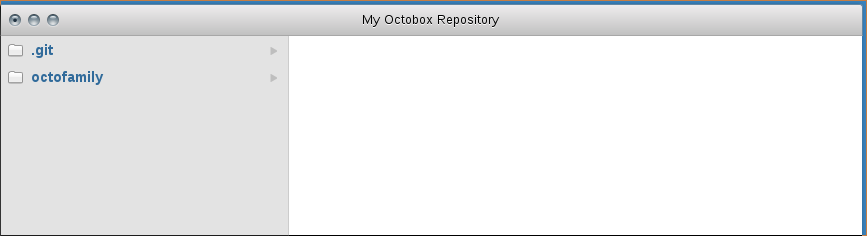
Ahora que eliminaste a todos los cats necesitarás confirmar tus cambios.

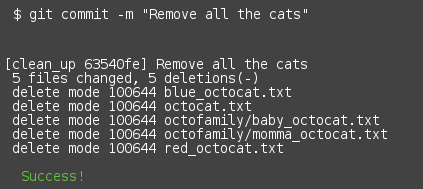
No dude en ejecutar el **git status** para verificar los cambios que está por cometer.

La opción '-a'

Si elimina un archivo sin usar **'git rm'**, encontrará que todavía tiene que **'git rm'** eliminar los archivos eliminados del árbol de trabajo. Puede guardar este paso utilizando la opción **'-a'** en **'git commit'**, que elimina automáticamente los archivos eliminados con la confirmación.

**git commit -am "Eliminar cosas"**

****

****

**1.22 Cambiar de nuevo a master**

Genial, ya casi terminaste con el cat ... er la corrección de errores, solo necesitas volver a la rama master para que puedas copiar (o unir, merge) tus cambios de la rama **clean\_up** a la rama **master**.

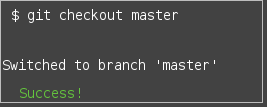
Continúa y revisa la rama **master**:

*Solicitudes de extracción*

Si está organizando su repositorio en GitHub, puede hacer algo llamado solicitud de extracción.

Una solicitud de extracción permite que el jefe del proyecto revise sus cambios y haga comentarios antes de decidir fusionarse en el cambio. Es una gran característica que se usa todo el tiempo para trabajadores remotos y proyectos de código abierto.

Consulte la página de ayuda de solicitud de extracción para obtener más información.



**23. Preparación para fusionarse**

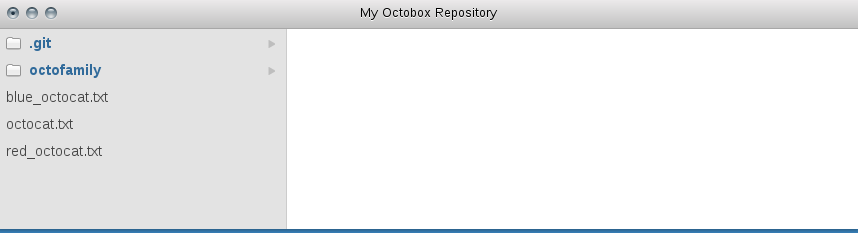
De acuerdo, ha llegado el momento en que debes fusionar tus cambios de la rama **clean\_up** a la rama **master**. Toma una respiración profunda, no es tan aterrador.

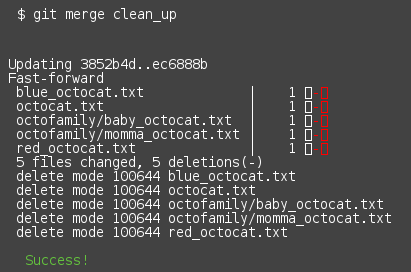
Ya estamos en la rama **master**, así que solo necesitamos decirle a Git que fusione la rama **clean\_up** en ella:

Fusionar Conflictos

Conflictos de combinación pueden ocurrir cuando los cambios se realizan en un archivo al mismo tiempo. Mucha gente se asusta cuando ocurre un conflicto, ¡pero no temas! No son tan aterradores, solo necesitas decidir qué código guardar.

Los conflictos de combinación están más allá del alcance de este curso, pero si le interesa leer más, eche un vistazo a la sección del libro de Pro Git sobre cómo se presentan los conflictos.





**24. Mantener las cosas limpias**

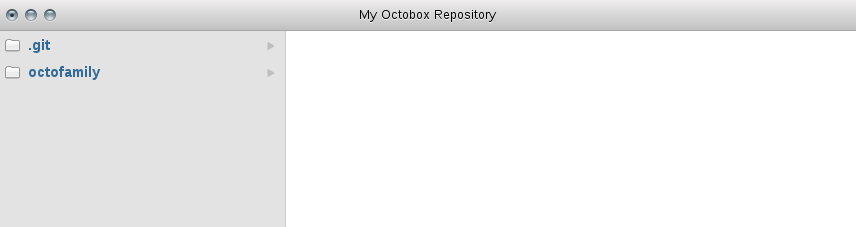
¡Felicitaciones! Acabas de lograr tu primera corrección de errores exitosa y fusión. Todo lo que queda por hacer es limpiarlo después de ti. Ya que terminaste con la rama **clean\_up** ya no la necesitas.

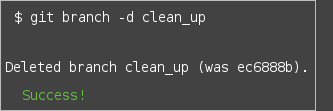
Puede usar **git branch -d <branch name>** para eliminar una rama. Continúa y elimina la rama **clean\_up** ahora:

Eliminar fuerza

¿Qué sucede si ha estado trabajando en una rama de funciones y decide que ya no desea esta función? Puede decidir eliminar la rama ya que está descartando la idea. Notarás que **git branch -d bad\_feature** no funciona. Esto se debe a que -d no te permitirá eliminar algo que no se ha fusionado.

Puede agregar la opción **--force (-f)** o usar -D que combine -d -f en un solo comando.





1.25 El empuje final

Aquí estamos, en el último paso. Estoy orgulloso de que hayas llegado hasta aquí, y ha sido genial aprender a Git contigo. Todo lo que tienes que hacer ahora es llevar todo lo que has estado trabajando a tu repositorio remoto, ¡y listo!

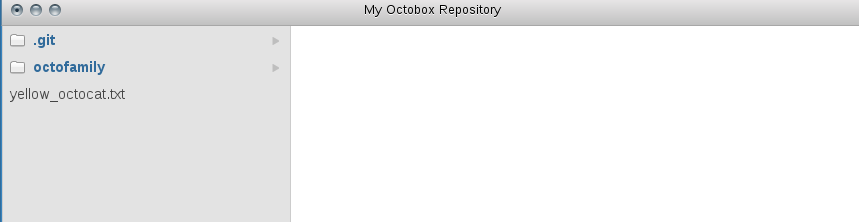
Aprendiendo más sobre Git

Solo arañamos la superficie de Git en este curso. Hay mucho más que puedes hacer con eso. Consulte la documentación de Git para obtener una lista completa de funciones.

El libro Pro Git, de Scott Chacon, es un recurso excelente para enseñarte el funcionamiento interno de Git.

help.github y GitHub Training también son excelentes para cualquier cosa relacionada con Git en general y el uso de Git con GitHub.





Este ha sido un tutorial de github