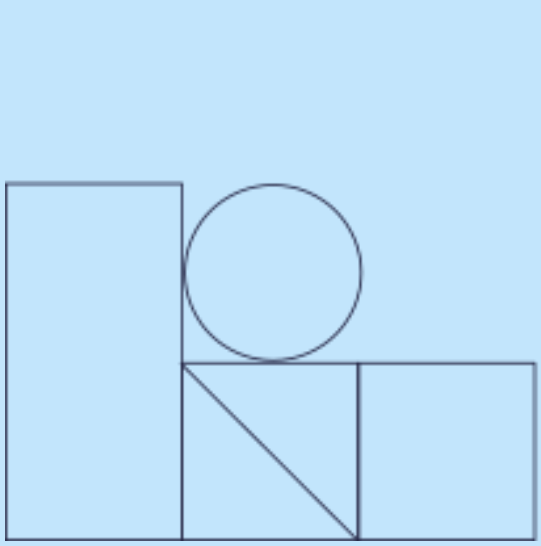
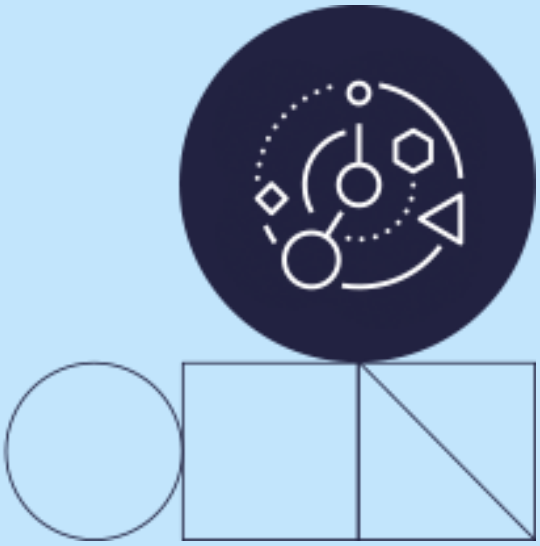
IBM SkillsBuild

IBM

Git y Github

Rebase, stash, clean





Índice

Fusión de ramas

En Git existen dos formas que nos permiten unir ramas, *git merge* y *git rebase*. La forma más conocida es *git merge* que ya vimos anteriormente, la cual realiza una fusión a tres bandas entre las dos últimas instantáneas de cada rama y el ancestro común a ambas, creando un nuevo *commit* con los cambios mezclados.

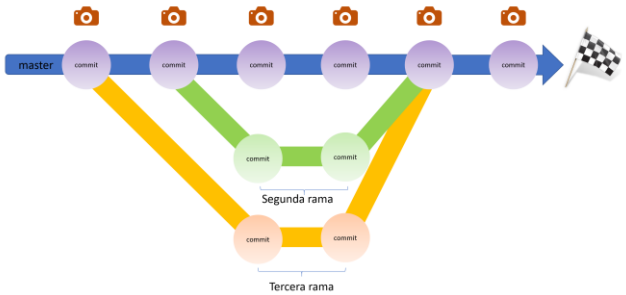
Git rebase básicamente lo que hace es recopilar uno a uno los cambios confirmados en una rama, y replicarlos sobre otra. Utilizar rebase nos puede ayudar a evitar conflictos siempre que se aplique sobre *commits* que están en local y no han sido subidos a ningún repositorio remoto. Si no tenemos cuidado con esto último y algún compañero utiliza cambios afectados, seguro que tendremos problemas ya que este tipo de conflictos normalmente son difíciles de reparar.

Como norma general *git rebase* se usa para:

- Editar commits anteriores
- Combinar varios commits en uno
- Eliminar o revertir commits que ya no son necesarios

Ramificar significa separarse de la rama maestra para que pueda trabajar por separado sin afectar el código principal y otros desarrolladores. Al crear un repositorio *Git*, por defecto, se nos asigna la rama *master*. A medida que comenzamos a realizar *commits*, esta rama *master* se sigue actualizando y apunta al último *commit* realizado en el repositorio.

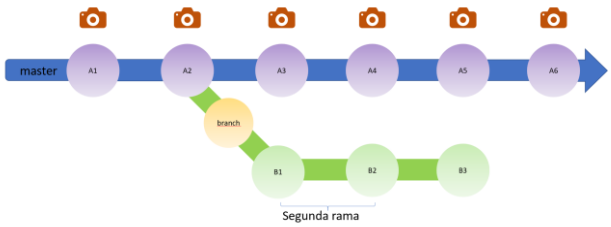
La ramificación en Git se ve así:



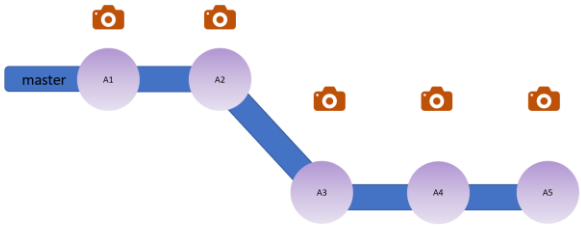
Rebase

Rebase en *Git* es un proceso de integración de una serie de confirmaciones sobre otra sugerencia base. Toma todas las confirmaciones de una rama y las agrega a las confirmaciones de una nueva rama. *Git rebase* tiene el siguiente aspecto:

Antes del *Rebase*:



Después del Rebase:



La sintaxis técnica del comando *rebase* es:

git rebase [-i | –Interactive] [opciones] [–exec cmd] [–onto newbase | –Keep-base] [corriente arriba [rama]]

Uso

El objetivo principal de *rebase* es mantener un historial de proyectos progresivamente más limpio y recto. El cambio de base da lugar a un historial de proyecto perfectamente lineal que puede seguir el compromiso final de la función hasta el comienzo del proyecto sin siquiera bifurcar. Esto facilita la navegación por su proyecto.

Usar *rebase* después de mandar los *commits* al repositorio se considera una mala práctica, debido a que cambiar su historial de *commits* puede dificultar las cosas para todos los demás que usan el repositorio.

Rebase confirmaciones contra una rama

Para reorganizar todos los *commits* entre otra rama y el estado de la rama actual, podemos ingresar el siguiente comando en nuestra consola:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git rebase --interactive other_branch_name
```

Rebase de los últimos commits

Para reorganizar los últimos *commits* en nuestra rama actual, podemos ingresar el siguiente comando en la consola:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git rebase --interactive HEAD~7
```

Comandos disponibles

Hay seis comandos disponibles para la sentencia *rebase*:

Pick (abreviado ‘p’)

Pick simplemente significa que la descripción del *commit* está incluido. Reorganizar el orden de los comandos *pick* cambia el orden de los *commits* cuando la reorganización está en marcha. Si elige no incluir un *commit*, debe eliminar toda la línea.

Reword (abreviado ‘r’)

El comando *reword* es similar a *pick*, pero después de usarlo, el proceso de *rebase* se detendrá y nos dará la oportunidad de modificar la descripción del *commit*. Los cambios realizados por el *commit* no se ven afectados.

Edit (abreviado ‘e’)

Si usamos *edit commit*, se nos dará la oportunidad de modificar el *commit*, lo que significa que podemos agregar o cambiar el *commit* por completo. También podemos realizar más commits antes de continuar con la reorganización. Esto nos permite dividir un *commit* grande en otros más pequeñas o eliminar los cambios erróneos realizados en un commit.

Squash (abreviado ‘s’)

Este comando nos permite combinar dos o más *commits* en uno solo. Un *commit* se combina con el *commit* que esté por encima de él. *Git* nos da la oportunidad de escribir un nuevo mensaje de confirmación que describa ambos cambios.

Fixup (abreviado ‘f’)

Es similar a *squash*, pero el *commit* que se fusionará no tendrá descripción. El *commit* simplemente se fusiona con el *commit* anterior y el mensaje de la confirmación anterior se usa para describir ambos cambios.

Exec (abreviado ‘x’)

Nos permite ejecutar comandos de *shell* arbitrarios contra un *commit*.

Merge Vs. Rebase

Tanto *merge* como *rebase* se utilizan para fusionar ramas, pero la diferencia radica en el historial de confirmación después de integrar una rama en otra. Si hacemos un *git merge*, las confirmaciones de todos los desarrolladores estarán allí en el registro de *git*. Mientras que, si usamos *git rebase*, la confirmación de un solo desarrollador se marcará en el registro de *git*. Los desarrolladores avanzados prefieren esta porque hace que el historial de confirmaciones sea lineal y limpio.

Ejemplo de uso de rebase

Comenzaremos nuestra rebase ingresando en la consola:

git rebase --interactive HEAD~7

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

git rebase --interactive HEAD~7
```

Nuestro editor de texto mostrará las siguientes líneas:

pick 1fc6c95 Patch A

pick 6b2481b Patch B

pick dd1475d Algo que quiero dividir

pick c619268 Un cambio en Patch B

pick fa39187 Algo que añadir a patch A

pick 4ca2acc Una descripción para reword

pick 7b36971 Algo para mover antes de patch B

En este ejemplo, haremos lo siguiente:

Combinar el quinto *commit* (fa39187) con el *commit* "Patch A" (1fc6c95), utilizando *squash* (combinar).

Mover el último *commit* (7b36971) hacia arriba antes del *commit* "Patch B" (6b2481b) y la conservarla como *pick*.

Fusionar el *commit* "Un cambio en Patch B" (c619268) con el *commit* "Patch B" (6b2481b) y omitir el mensaje de confirmación utilizando *fixup*.

Separar el tercer *commit* (dd1475d) en dos *commits* más pequeños utilizando *edit* (editar).

Corregir el mensaje de confirmación mal escrita del *commit* (4ca2acc), utilizando *reword* (otro texto).

Para comenzar, tendremos que modificar los comandos en el archivo para que aparezcan como sigue:

pick 1fc6c95 Patch A

squash fa39187 Algho para añadir en patch A

pick 7b36971 Algo para mover antes de patch B

pick 6b2481b Patch B

fixup c619268 Un cambio en Patch B

edit dd1475d Algo que queremos dividir

reword 4ca2acc Una descripción para reword

Hemos cambiado cada comando de la línea desde *pick* al comando que nos interesa. Guardamos y cerramos el editor; esto comenzará el *rebase* interactivo.

Git salta el primer comando de *rebase*, *pick 1fc6c95*, ya que no necesita hacer nada. Va al siguiente comando, *squash fa39187*. Como esta operación requiere una entrada, *Git* vuelve a abrir nuestro editor de texto o IDE predeterminado. El archivo que abre será parecido a lo siguiente:

Es una combinación de dos confirmaciones.

El mensaje de la primera confirmación es:

Patch A

Este es el mensaje de la 2.a confirmación:

something to add to patch A

Ingresa el mensaje de confirmación para tus cambios. Las líneas que comienzan con

con '#' se ignoran, y un mensaje vacío anula la confirmación.

Actualmente no se encuentra en una rama.

Cambios por confirmar:

(usa "git reset HEAD <file>..." para deshacer)

#

modificado: a

#

Este archivo es la manera de *Git* de advertirnos, "esto es lo que estoy a punto de hacer con esta *squash* (combinación)".

Detalla el primer mensaje de *commit* ("Patch A") y el segundo mensaje de *commit* ("Algo para añadir a patch A"). Si estamos satisfecho con estos mensajes de *commit*, podemos guardar el archivo y cerrar el editor. De lo contrario, tenemos la opción de cambiar el mensaje de *commit*, simplemente, cambiando el texto.

Cuando el editor esté cerrado, la *rebase* continúa:

pick 1fc6c95 Patch A

squash fa39187 se agrega algo a patch A

pick 7b36971 algo para mover antes de patch B

pick 6b2481b Patch B

fixup c619268 Un cambio en Patch B

edit dd1475d Algo que quiero dividir

reword 4ca2acc Descripción del reword

Git procesa los dos comandos *pick* (para *pick 7b36971* y *pick 6b2481b*).

También procesa el comando *fixup* (fixup c619268), ya que este no necesita ninguna interacción. *fixup* fusiona los cambios de *c619268* en el *commit* que tiene ante sí, *6b2481b*. Ambos cambios tendrán el mismo mensaje de *commit*: "Patch B".

Podemos modificar el *commit* ahora con:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git commit --amend
```

Una vez que hayamos hecho *commit* con los cambios, ejecutamos:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git rebase --continue
```

En este punto, podemos editar cualquiera de los archivos de nuestro proyecto para hacer más cambios. Para cada cambio que hagamos, tendremos que realizar un nuevo *commit*. Lo podemos hacer ingresando el comando:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git commit --amend.
```

Cuando terminemos de hacer todos nuestros cambios, podemos ejecutar

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git rebase --continue
```

Luego *Git* llega al comando *reword 4ca2acc*. Este abre nuestro editor de texto una vez más y presenta la siguiente información:

```
Una descripción para reword

# Ingresa el mensaje de confirmación para tus
cambios. Las líneas que comienzan con

# con '#' se ignoran, y un mensaje vacío anula la
confirmación.

# Actualmente no se encuentra en una rama.

# Cambios por confirmar:

#   (use "git reset HEAD^1 <file>..." desmontar)

#

# modificado:   a

#
```

Como antes, *Git* muestra el mensaje de confirmación para que lo editemos. Podemos cambiar el texto ("Una descripción para reword"), guardar el archivo y cerrar el editor. *Git* terminará el *rebase* y nos devolverá al terminal.

Subir código de rebase a GitHub

Como hemos modificado el historial de *Git*, el *git push origin* común **no** funcionará.

Tendremos que modificar el comando realizando un "push forzado" de nuestros últimos cambios:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

# No sobreescribir cambios
$ git push origin main --force-with-
lease

# Sobreescribir cambios
$ git push origin main --force
```

Stash

Git tiene un área llamada "*stash*" donde podemos almacenar temporalmente una captura de nuestros cambios sin enviarlos al repositorio. Está separada del directorio de trabajo (*working directory*), del área de preparación (*staging area*), o del repositorio.

Esta funcionalidad es útil cuando hemos hecho cambios en una rama, no estamos listo para realizar un *commit*, pero necesitamos cambiar a otra rama.

Guardar cambios en el stash

Para guardar nuestros cambios en el *stash*, ejecutamos el comando:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git stash save "Descripción opcional"
```

Esto guarda los cambios y revierte el directorio de trabajo a como se veía en nuestro último *commit*. Los cambios guardados están disponibles en cualquier rama de ese repositorio.

Hay que tener en cuenta que los cambios que queramos guardar deben estar en los archivos rastreados. Si hemos creado un nuevo archivo e intentamos guardar nuestros cambios, puede que obtengamos el error:

No local changes to save (No hay cambios locales que guardar).

Ver los cambios guardados en el stash

Para ver lo que hay en nuestro *stash*, ejecutaremos el comando:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git stash list
```

Esto devuelve una lista de nuestras capturas guardadas en el formato:

stash@{0}: RAMA-STASHED-CAMBIOS-SON-PARA: MESSAGE.

La parte de *stash@{0}* es el nombre del *stash*, y el número en las llaves ({ }) es el índice (*index*) del *stash*. Si tenemos múltiples conjuntos de cambios guardados en *stash*, cada uno tendrá un índice diferente.

Si olvidamos los cambios que hicimos en el *stash*, podemos ver un resumen de ellos con el comando:

```
git stash show NOMBRE-DEL-STASH.
```

Si queremos ver el típico diseño "diff" (con los + y - en las líneas con los cambios), podemos incluir la opción -p (para parche o patch). Un ejemplo:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git stash show -p stash@{0}
```

```
# Ejemplo de un resultado:

diff --git a/PathToFile/fileA b/PathToFile/fileA

index 2417dd9..b2c9092 100644

--- a/PathToFile/fileA

+++ b/PathToFile/fileA

@@ -1,4 +1,4 @@

-What this line looks like on branch

+What this line looks like with stashed changes
```

Recuperar Cambios en Stash

Para recuperar los cambios del *stash* y aplicarlos a la rama actual en la que estamos, tenemos dos opciones:

git stash apply NOMBRE-DEL-STASH aplica los cambios y deja una copia en el stash

git stash pop NOMBRE-DEL-STASH aplica los cambio y elimina los archivos del stash

Puede haber conflictos cuando se aplican los cambios. Podemos resolver los conflictos de forma similar a un *merge*.

Borrar los Cambios Guardados en Stash

Si queremos remover los cambios guardados en *stash* sin aplicarlos, ejecutaremos el comando:

```
git stash drop NOMBRE-DEL-STASH
```

Para limpiar todo del *stash*, ejecutaremos el comando:

```
git stash clear
```

Clean

El comando *git clean* limpia nuestro proyecto de archivos no deseados.

Mientras estamos trabajando en un repositorio podemos añadir archivos a él, que realmente no forma parte de nuestro directorio de trabajo, archivos que no se deberían de agregar al repositorio remoto.

El comando *clean* actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando *add*.

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git clean
```

La ejecución del comando predeterminado puede producir un error. La configuración global de *Git* obliga a usar la opción *force* con el comando para que sea efectivo. Se trata de un importante mecanismo de seguridad ya que este comando no se puede deshacer.

Revisar qué archivos no tienen seguimiento.

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git clean --dry-run
```

Eliminar los archivos listados de no seguimiento.

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git clean -f
```