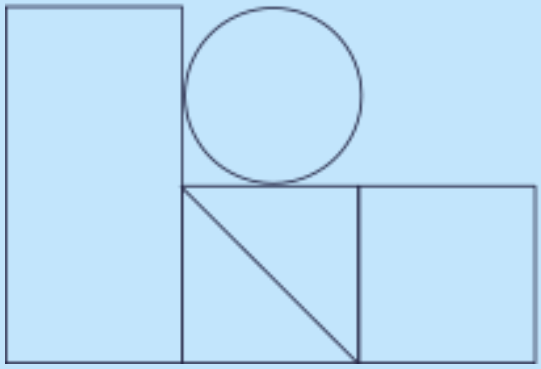
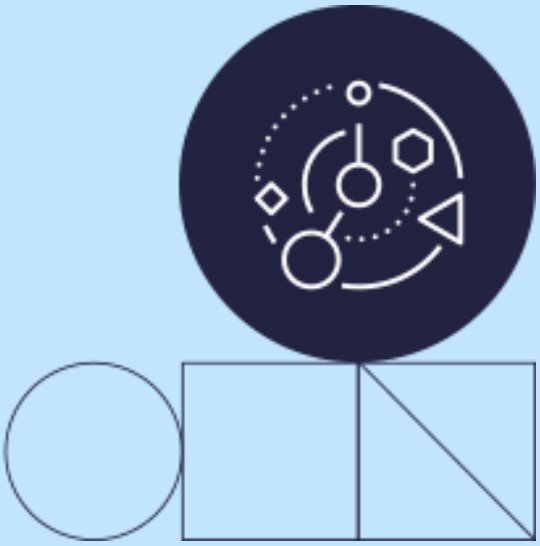
IBM SkillsBuild

IBM

Git e Github

Rebase, stash, clean





Índice

Fusão de ramas

Em Git há duas maneiras de fundir ramos, *git merge* e *git rebase*. A maneira mais conhecida é a *git merge*, que vimos anteriormente, que realiza uma fusão de três vias entre os dois últimos instantâneos de cada ramo e o ancestral comum a ambos, criando um novo *commit* com as mudanças fundidas.

Git rebase basicamente faz é coletar uma a uma as mudanças comprometidas em um ramo, e replicá-las em outro ramo. O uso do rebase pode nos ajudar a evitar conflitos desde que seja aplicado em *commits* que sejam locais e não tenham sido carregados em nenhum repositório remoto. Se não tivermos cuidado com estas últimas e um colega usar as mudanças afetadas, estamos fadados a ter problemas, já que tais conflitos são geralmente difíceis de reparar.

Como regra geral *git rebase* é usado para:

- Editar compromissos anteriores.

- Combinar vários compromissos em um só.

- Remover ou reverter os compromissos que não são mais necessários.

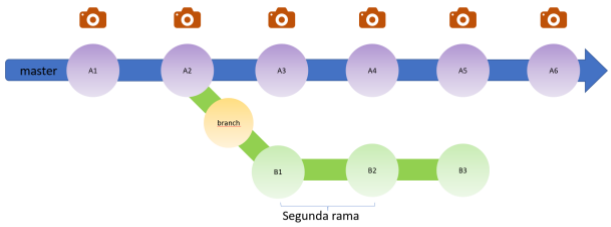
A ramificação significa se separar da ramificação principal para que possa trabalhar separadamente sem afetar o código principal e outros desenvolvedores. Quando criamos um repositório *Git*, por padrão, nos é atribuído o ramo *master*. À medida que começamos a perceber *commits*, este ramo mestre continua se atualizando e aponta para o último *commit* feito com o repositório.

A ramificação em Git tem este aspecto:

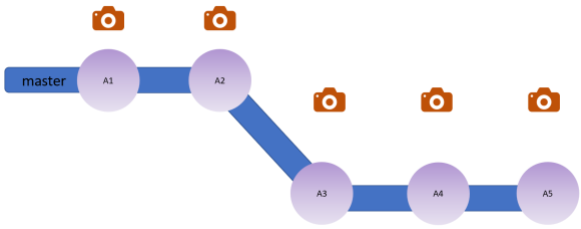
Rebase

Rebase in *Git* é um processo de integração de um conjunto de compromissos sobre outra sugestão base. Ele pega todos os compromissos de um ramo e os acrescenta aos compromissos de um novo ramo. *Git rebase* tem este aspecto:

Antes da *Rebase*:



Após o *Rebase*:



A sintaxe técnica do comando de *rebase* é:

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

Uso

O principal objetivo do *rebase* é manter um histórico de projetos progressivamente mais limpo e reto. O rebase resulta em um histórico de projeto perfeitamente linear que pode acompanhar o compromisso final da função até o início do projeto sem sequer ramificar-se. Isto facilita a navegação em seu projeto. Usar o *rebase* depois de enviar os *commits* para o repositório é considerado uma má prática, porque mudar seu histórico de *commits* pode tornar as coisas difíceis para todos os outros que usam o repositório.

A rebase se compromete contra um ramo

Para reorganizar todos os *commits* entre outra filial e o estado da filial atual, podemos entrar com o seguinte comando em nosso console:

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

$ git rebase --interactive other_branch_name
```

Rebase do últimos commits

Para reorganizar os últimos *commits* em nosso ramo atual, podemos entrar com o seguinte comando no console:

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

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

Comandos disponíveis

Há seis comandos disponíveis para a declaração de *rebase*:

Pick (abreviado ‘p’)

Pick simplesmente significa que a descrição do *commit* está incluída. A reordenação da ordem dos comandos de retirada muda a ordem dos *commits* quando o rearranjo está em andamento. Se você optar por não incluir um *commit*, você deve remover toda a linha.

Reword (abreviado ‘r’)

O comando *reword* é semelhante a *pick*, mas depois de usá-lo, o processo de rebase pára e lhe dá a oportunidade de modificar a descrição do *commit*. As mudanças feitas pelo *commit* não são afetadas.

Edit (abreviado ‘e’)

Se utilizarmos *edit commit*, teremos a oportunidade de modificar o *commit*, o que significa que podemos acrescentar ou alterar todo o *commit*. Também podemos assumir mais *commits* antes de continuar com a reorganização. Isto nos permite dividir um grande *commit* em menores ou remover alterações errôneas feitas a um *commit*.

Squash (abreviado ‘s’)

Este comando permite combinar dois ou mais *commits* em um só. Um *commit* é combinado com o *commit* acima dele. *Git* nos dá a oportunidade de escrever uma nova mensagem de compromisso descrevendo ambas as mudanças.

Fixup (abreviado ‘f’)

Isto é semelhante ao *squash*, mas o *commit* a ser fundido não terá descrição. O *commit* é simplesmente fundido com o *commit* anterior e a mensagem de compromisso anterior é usada para descrever ambas as mudanças.

Exec (abreviado ‘x’)

Permite-nos executar comandos de *shell* arbitrários contra um *commit*.

Merge Vs. Rebase

Tanto *merge* como o rebase são usados para fundir ramos, mas a diferença é o histórico de comprometimento após a integração de um ramo em outro. Se fizermos um *git merge*, os compromissos de todos os desenvolvedores estarão lá no registro de *git*. Considerando que, se utilizarmos o *git rebase*, um único compromisso do desenvolvedor será marcado no registro de git. Os desenvolvedores avançados preferem isto porque torna o histórico de compromisso linear e limpo.

Exemplo de uso de rebase

Iniciaremos nosso processo de rebase entrando no console:

git rebase --interactive HEAD~7

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

git rebase --interactive HEAD~7
```

Nosso editor de texto exibirá as seguintes linhas:

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

Neste exemplo, faremos o seguinte:

Combine o quinto *commit* (fa39187) com o *commit* "Patch A" (1fc6c95), usando *squash* (combinando).

Mova o último *commit* (7b36971) para cima antes do *commit* "Patch B" (6b2481b) e mantenha-o como um *pick*.

Fundir o *commit* "A change in Patch B" (c619268) com o *commit* "Patch B" (6b2481b) e pular a mensagem de confirmação usando o *fixup*.

Separe o terceiro *commit* (dd1475d) em dois *commits* menores usando a *edit* (editar).

Corrigir a mensagem de confirmação mal escrita do *commit* (4ca2acc), usando *reword* (outro texto).

Para começar, teremos que modificar os comandos no arquivo para que apareçam da seguinte forma:

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

Mudamos cada comando na linha, de *pick* para o comando em que estamos interessados. Salvamos e fechamos o editor; isto iniciará a *rebase* interativa.

Git pula o primeiro comando de *rebase*, *pick* 1fc6c95, já que ele não precisa fazer nada. Vai para o próximo comando, *squash* fa39187. Como esta operação requer entrada de dados, *Git* reabre nosso editor de texto padrão ou IDE. O arquivo que ele abre se parecerá com o seguinte:

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 arquivo é a forma de *Git* nos avisar, "eis o que vou fazer com esta *squash* (combinação)".

Ele detalha a primeira mensagem de *commit* ("Patch A") e a segunda mensagem de *commit* ("Algo a acrescentar ao patch A"). Se estivermos satisfeitos com estas mensagens de *commit*, podemos salvar o arquivo e fechar o editor. Caso contrário, temos a opção de alterar a mensagem de *commit* simplesmente alterando o texto.

Quando o editor é fechado, o *rebase* continua:

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

Também processa o comando *fixup* (fixup c619268), uma vez que este não precisa de nenhuma interação. *fixup* funde as mudanças de c619268 para o compromisso antes dele, 6b2481b. Ambas as mudanças terão a mesma mensagem de *commit*: "Patch B".

Podemos modificar o *commit* agora com:

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

$ git commit --amend
```

Uma vez que tenhamos feito *commit* as mudanças, executamos:

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

$ git rebase --continue
```

Neste momento, podemos editar qualquer um dos arquivos de nosso projeto para fazer outras mudanças. Para cada mudança que fizermos, teremos que fazer um novo *commit*. Podemos fazer isso entrando no comando:

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

$ git commit --amend.
```

Quando tivermos terminado de fazer todas as nossas mudanças, podemos correr:

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

$ git rebase --continue
```

Então *Git* chega ao comando *reword* 4ca2acc, que abre nosso editor de texto mais uma vez e apresenta as seguintes informações:

```
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* exibe a mensagem de confirmação para que possamos editá-la. Podemos alterar o texto ("Uma descrição para reword"), salvar o arquivo e fechar o editor. Git terminará a reformulação da palavra e nos retornará ao terminal.

Carregando o código de rebase para GitHub

Como modificamos a história de *Git*, a origem comum do *`git push origin`* **não** vai funcionar.

Teremos que modificar o comando executando um "push forçado" de nossas últimas mudanças:

```
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 tem uma área chamada "*stash*" onde podemos armazenar temporariamente um instantâneo de nossas mudanças sem comprometê-las com o repositório. É separado do diretório de trabalho (*working directory*), da área de preparação (*staging área*), ou do repositório.

Esta característica é útil quando fizemos mudanças em um ramo, não estamos prontos para realizar um *commit*, mas precisamos mudar para outro ramo.

Salvar as mudanças no stash

Para salvar nossas mudanças no *stash*, nós executamos o comando:

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

$ git stash save "Descripción opcional"
```

Isto poupa as mudanças e reverte o diretório de trabalho para o que parecia em nosso último *commit*. As mudanças salvas estão disponíveis em qualquer ramo daquele repositório.

Observe que as mudanças que você deseja salvar devem estar nos arquivos rastreados. Se tivermos criado um novo arquivo e tentarmos salvar nossas mudanças, poderemos ter o erro:

No local changes to save (Nenhuma mudança local para economizar).

Ver mudanças guardadas no stash

Para ver o que está em nosso *stash*, nós comandaremos o comando:

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

$ git stash list
```

Isto retorna uma lista de nossos estandes armazenados no formato:

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

A parte do *stash*@{0} é o nome do *stash*, e o número em chaves ({ }) é o índice (*index*) do *stash*. Se tivermos vários conjuntos de mudanças armazenados no *stash*, cada um terá um índice diferente.

Se esquecermos as mudanças que fizemos no *stash*, podemos ver um resumo delas com o comando:

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

Se quisermos ver o típico layout "dif" (com os + e - nas linhas com as mudanças), podemos incluir a opção -p (para adesivo o para patch). Aqui está um exemplo:

```
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
```

Recuperando mudanças no Stash

Para recuperar as mudanças do *stash* e aplicá-las ao ramo atual em que nos encontramos, temos duas opções:

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

Pode haver conflitos ao aplicar as mudanças. Podemos resolver conflitos de forma semelhante a um *merge*.

Apagar mudanças salvas no Stash

Se quisermos remover as mudanças guardadas no *stash* sem aplicá-las, executaremos o comando:

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

Para limpar tudo do *stash*, execute o comando:

```
git stash clear
```

Clean

O comando *git clean* limpa nosso projeto de arquivos indesejados.

Enquanto trabalhamos em um repositório, podemos adicionar a ele arquivos que não fazem parte de nosso diretório de trabalho, arquivos que não devem ser adicionados ao repositório remoto.

O comando *clean* age sobre arquivos não rastreados, que são arquivos que estão no diretório de trabalho, mas ainda não foram adicionados ao índice de rastreamento do repositório com o comando *add*.

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

$ git clean
```

A execução do comando padrão pode resultar em um erro. A configuração global de *Git* requer a opção de *force* a ser utilizada com o comando para que seja eficaz. Este é um importante mecanismo de segurança, pois este comando não pode ser desfeito.

Verifique quais arquivos estão desmarcados.

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

$ git clean --dry-run
```

Remover os arquivos listados como não rastreados.

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

$ git clean -f
```