GIT & GITHUB

Do Básico ao Avançado

Domine Git e GitHub com este material completo. Com ele você aprenderá a gerenciar fluxos de trabalho de maneira simples e dinâmica.

Git Fork

Lembra dos famosos **"universos alternativos"** que conhecemos lá no material que fala sobre Branchs?

Lá, aprendemos a como criar outras realidades diferentes que fazem parte de uma determinada realidade, que no caso é a pasta do projeto principal.

Vamos atualizar um pouco esse conceito agora, ok?

Considere que a partir de agora, a pasta do seu projeto representa um universo, e as branchs a partir de agora, representam sub-universos que fazem parte do universo maior que é a pasta do projeto.

Atualizou a informação? Então podemos prosseguir para o famoso **Fork** ;)

O que é Fork?

Quando falamos em **Fork**, logo lembramos de um garfo, um utensílio de cozinha que contém uma haste que no final se separa em três hastes, certo?

De maneira geral, um Fork representa uma cópia de um determinado repositório para outro, permitindo que os usuários experimentem alterações em um determinado projeto.

Mas daí você se pergunta: *"Porque alguém faria uma cópia de um determinado projeto no GitHub?"*.

São diversos os motivos:

+ **Usar o projeto de outra pessoa, como ponto de partida para sua própria ideia.**

+ **Quando você deseja propor alterações no repositório principal ao mesmo tempo que você não tem acesso ao mesmo.**

Mas daí você pode pensar: *"Se eu quiser usar um determinado repositório como ponto de partida para minha própria ideia, basta criar meu próprio repositório, e usar somente os arquivos que eu preciso daqueles repositórios que não são meus, não é verdade?"*.

Sim, isso é uma ideia totalmente plausível de acontecer, pois pra que fazer uma cópia de um repositório existente, dizendo a todos que meu projeto é uma **"cópia"**, se eu não posso simplesmente criar meu próprio repositório que contenha alguns arquivos de outros? Não é verdade?

Ou quem sabe pensar: *"Se eu quiser propor alterações em um repositório existente, basta eu conversar com algum dos contribuidores e tentar pedir acesso ao repositório. E mesmo se mesmo assim não me derem acesso, posso simplesmente criar uma nova Issue..."*.

Sim, isso também é verdade!

Podemos dizer também que podemos criar um fork quando não temos permissões de push para o repositório do GitHub de alguém.

Bem, se tudo isso aí é uma grande verdade, podemos resumir (em outras palavras) que fazemos um Fork quando:

1) Encontramos um repositório interessante, mas com pontos que 'achamos' que podem ser melhorados, ao mesmo tempo que não temos acesso ao push, e que a nossa mão está coçando para realizar essas melhorias de tal modo que já queremos fazer push para ontem. Em casos como esses executamos um Fork.

2) Encontramos um repositório interessante, com pontos que 'achamos' que podem ser melhorados, ao mesmo tempo que não te deixam fazer parte do projeto, ou não consideram suas alterações relevantes. Em casos como esses executamos um Fork.

3) Encontramos um repositório interessante, mas tão antigo, que parece que o criador o abandonou de vez, e como queremos ressuscitá-lo, podemos fazer um Fork.

Se tratando do item 3, podemos dizer então que um FORK, nada mais é do que uma prática do estilo **"Reinventar a Roda"**? Sim, podemos dizer que sim.

Existem aqueles que criam um Fork de seus antigos projetos porque perderam o acesso a sua antiga conta do GitHub, ao mesmo tempo que existem aqueles que também criam um Fork dos seus próprios projetos para representar e dizer aos outros que a ideia tomou novos rumos (famosa **"pivotagem"** do mundo do empreendedorismo).

É importante ressaltar que toda vez que fazemos uma cópia de um determinado repositório (Fork), tanto o repositório original quanto a cópia são assíncronos, ou seja, as atualizações acontecem de forma independente (e não automática).

Nesse caso, se o repositório original sofrer alguma atualização (receber novos commits), tal operação não vai refletir no repositório de cópia.

Upstream & Downstream

Na plataforma do GitHub existem duas nomenclaturas que representam categorias diferentes de um repositório:

Upstream: Significa que é o repositório original de uma determinada aplicação.

DownStream: Significa que o repositório atual, é uma cópia de um determinado repositório do tipo upstream.

Ou seja, vamos imaginar que criamos um novo repositório no GitHub, que armazena um grande projeto chamado de **Fastly 98**, que se trata de um navegador feito na linguagem C# com um layout parecido com o Internet Explorer do Windows 98.

Como a ideia foi exclusivamente minha, e fui eu que postei no meu GitHub, podemos considerar que o repositório é do tipo **Upstream**.

Supondo, que um outro usuário do github gostou tanto do projeto, que ele resolveu fazer um clone da aplicação para a máquina local.

Podemos dizer que esse usuário fez **Downstream**?

Nesse caso, o **Downstream** acontece quando a informação flui para baixo.

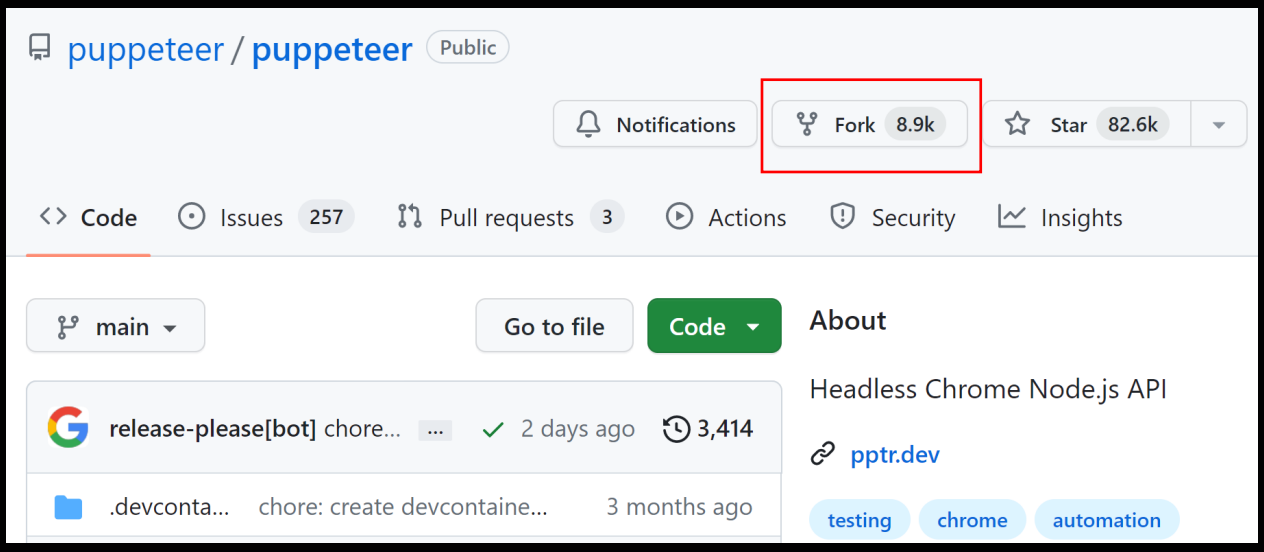
E o **Upstream**, quando a informação flui para cima.

Um exemplo de **Upstream**, é quando você faz alterações no seu repositório, de modo a enviá-las de volta ao repositório remoto, para que todos que extraem da mesma fonte trabalhem com as mesmas alterações.

Lembre-se que upstream e downstream são nomenclaturas criadas pelo Git ou GitHub, mas sim pela própria comunidade, ok?

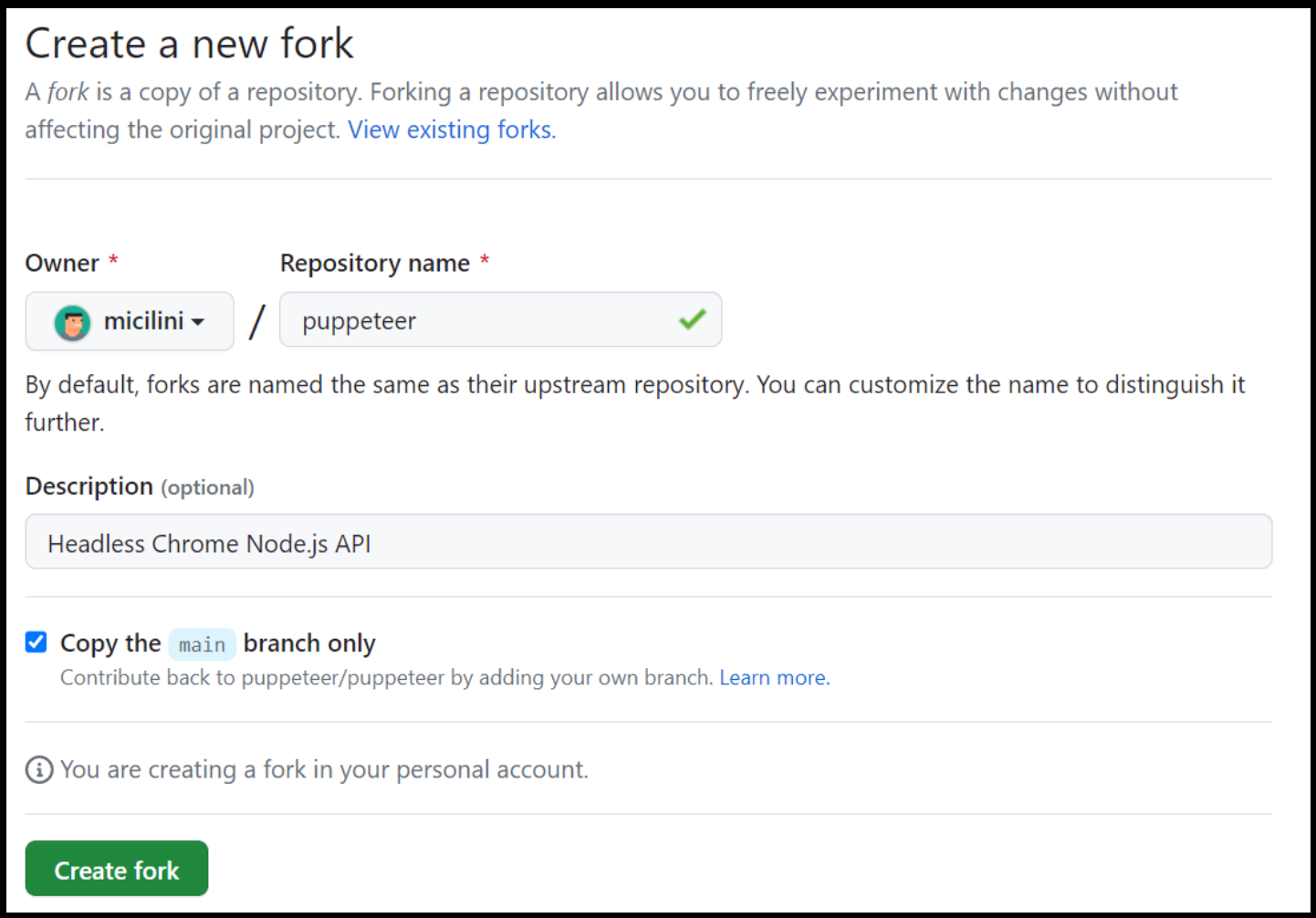
Trabalhando Fork

Tudo começa quando a gente encontra um repositório legal no GitHub, e decide clicar no botão escrito Fork:



No caso do repositório acima, ele conta com mais 8.9 mil Forks, incrível não?

Automaticamente após clicar no botão de Fork, o GitHub vai te redirecionar para uma tela parecida com aquela que criamos nossos repositórios.



Automaticamente após clicar no botão de Fork, o GitHub vai te redirecionar para uma tela parecida com aquela que criamos nossos repositórios.

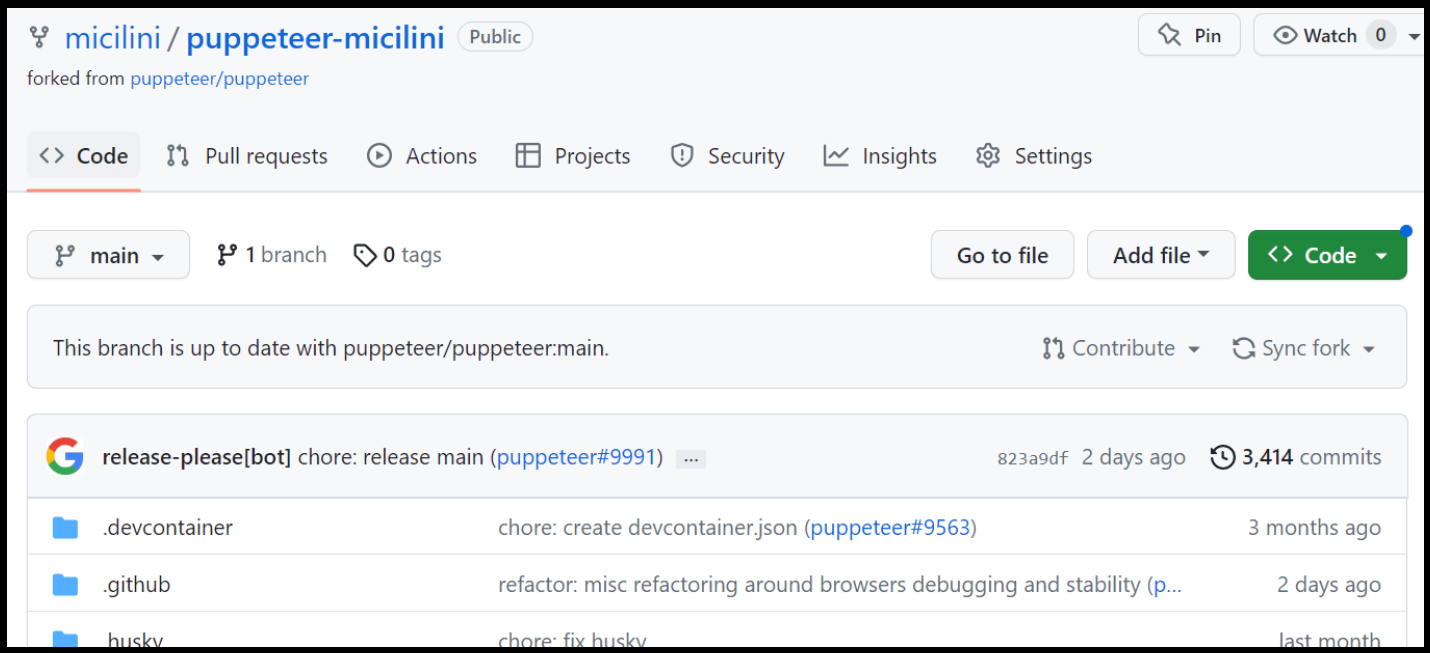
A diferença é que agora, nós vamos criar um Fork, onde podemos alterar o nome dele, a descrição e copiar a branch principal daquele repositório.

Para criar o Fork basta clicar no botão **"Create Fork"**.

E pronto, um novo repositório cópia será adicionado dentro da sua conta do GitHub, e entrará na lista dos seus repositórios.

Note que o repositório contém a mensagem **"forked from..."** indicando que o repositório atual é uma cópia de outro.

Observe também que a branch principal desse repositório é a **MAIN**, que foi herdada do repositório original.

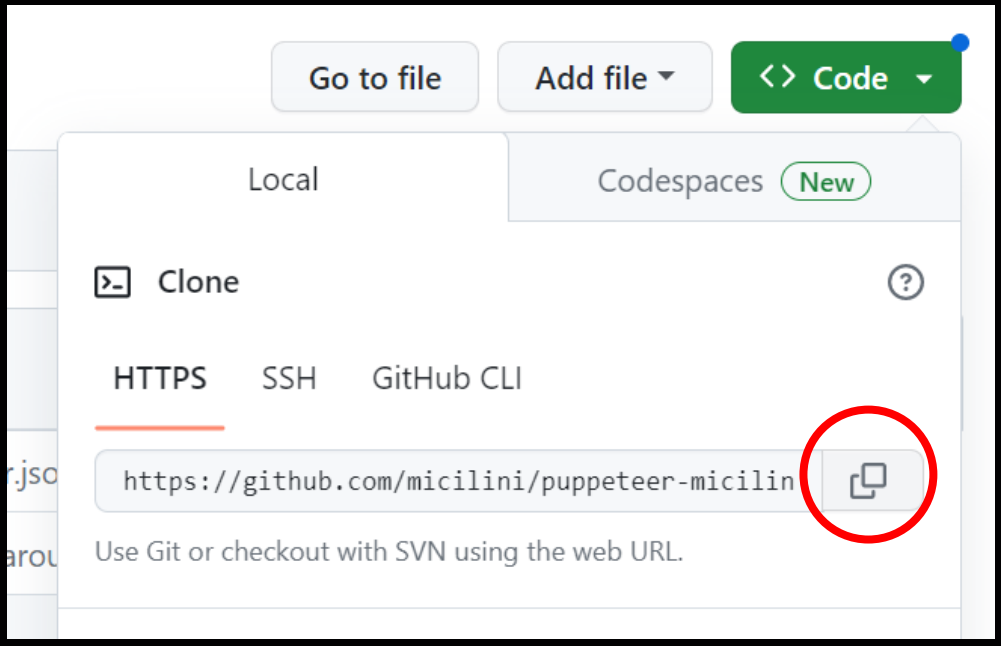


Clonando um Fork

Após criar o seu repositório cópia, chegou a hora de fazermos um **clone** para a sua máquina local.

Com o repositório cópia aberto, basta seguir aquela receita de bolo que já estamos super acostumados a fazer, ainda se lembra, né?

Primeiro copiar o link para fazer o clone:



Em seguida executar o comando do **git clone**:

```
git clone https://github.com/micilini/puppeteer-micilini.git
```

A partir desse momento, você pode fazer as alterações que desejar, como, por exemplo:

+ Criar novas branchs.
+ Adicionar novos commits.
+ Realizar rebases.
+ Deletar arquivos.
+ Fazer pull/push com seu repositório cópia presente no GitHub.
+ E entre outras coisas.

Lembre-se que tudo o que fizemos até agora, são operações que realizamos dentro do nosso repositório cópia, e não no repositório original.

Atualizando o Fork com Upstream

Supondo que houve atualizações (commits) no repositório original, como eu faço para que meu repositório cópia (Fork), receba tais atualizações?

O procedimento, é bem simples, com o terminal do git aberto dentro da pasta do seu projeto, execute o seguinte comando:

```
git remote add upstream https://github.com/puppeteer/puppeteer.git
```

Com esse comando nós estamos adicionando um novo link para outro repositório, onde nomeamos ele como “upstream” (por padrão usamos essa nomenclatura, mas você pode adicionar o nome que desejar ali).

Perceba que ali colocamos a URL do repositório original que é **https://github.com/puppeteer/puppeteer.git.**

Executando o comando:

```
git remote -v
```

Podemos visualizar uma lista de todos os repositórios que seu projeto aponta:

```
MINGW64:/c/Puppeteer

SeuUser@DESKTOP-UHDBDBV MINGW64 C:/Projeto (main)
$ git remote -v

origin https://github.com/micilini/puppeteer-micilini.git (fetch)
origin https://github.com/micilini/puppeteer-micilini.git (push)
upstream https://github.com/puppeteer/puppeteer.git (fetch)
upstream https://github.com/puppeteer/puppeteer.git (push)
```

Ali podemos ver o origin, que nada mais é do que o nosso repositório cópia, e também o upstream, que usaremos para puxar as atualizações do repositório original.

Agora para atualizar o seu projeto local com as atualizações do repositório original, basta executar o seguinte código:

```
git pull upstream main
```

No lugar de main, você pode colocar master ou alguma branch desejada. No meu caso usei main, pois o repositório do puppeter não usa a branch master como principal, mas sim a **branch main**.

Para finalizar podemos realizar o **git push**, que automaticamente as atualizações irão para o repositório cópia do GitHub.

Fork Workflow (Pull Requests)

Um dos ciclos mais comuns de colaboração no git e github, é o famoso **'Fork Workflow'**.

O processo se baseia no uso do Fork, com o uso dos **Pull Requests**.

As solicitações do tipo **Pull Requests**, é um mecanismo que permite que as alterações de um repositório cópia (Fork) sejam notificadas ao repositório original (**Upstream**).

Ou seja, vamos imaginar que você gostou tanto do projeto Puppeteer que sentiu a necessidade de criar uma funcionalidade, daí você fez a cópia daquele repositório (**Fork**), criou a funcionalidade (**git add e git commit**) e agora gostaria que ela fosse integrada ao repositório original.

Essa integração é possível graças ao **Pull Request**, com ele você consegue atrelar seus commits com alguma Issue presente no repositório original, dizendo o seguinte:

"Olha donos do repositório do Puppeter, adicionei novas funcionalidades por meio do Fork, vocês agora poderiam validar e quem sabe fazer um merge desses meus commits para dentro do repositório original?"

Sendo assim, você faz um **Pull Request** toda vez que:

+ Você gostaria de enviar suas alterações do seu repositório cópia para o repositório original.

+ Realizar discussões sobre o código.

+ Ou quando você deseja integrar algum código revisado ao projeto original.

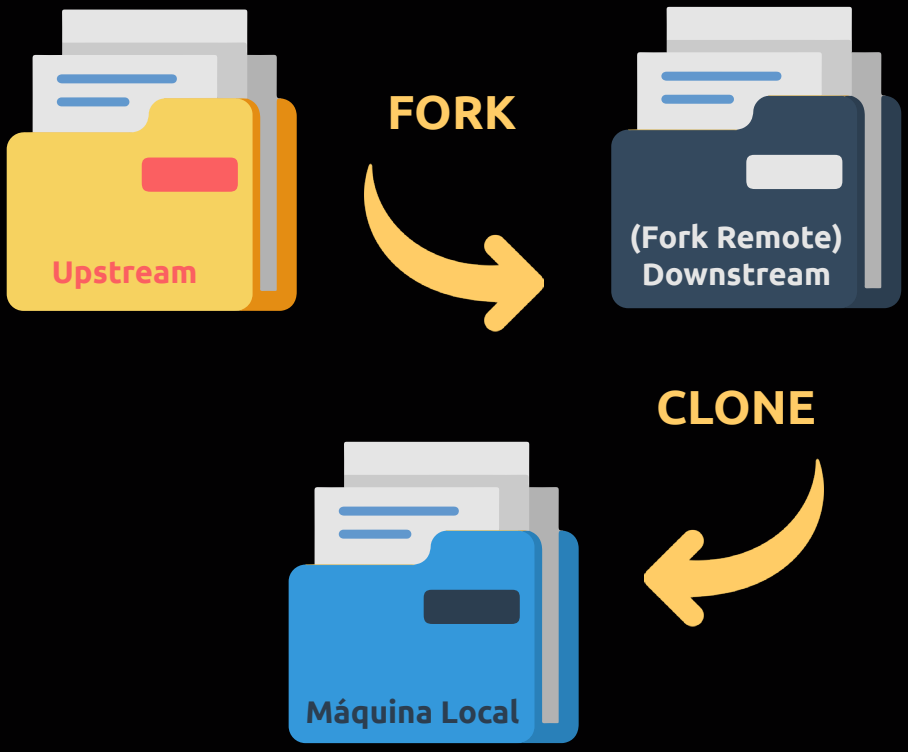
Ou seja, é quando você quer que um repositório de um terceiro, receba as alterações que você realizou dentro do seu Fork.

Obviamente que esse tipo de procedimento depende da aprovação do criador do repositório.

Fork Workflow (Ciclo)

Para entendermos melhor a ideia do **Fork Workflow**, vamos as ilustrações!

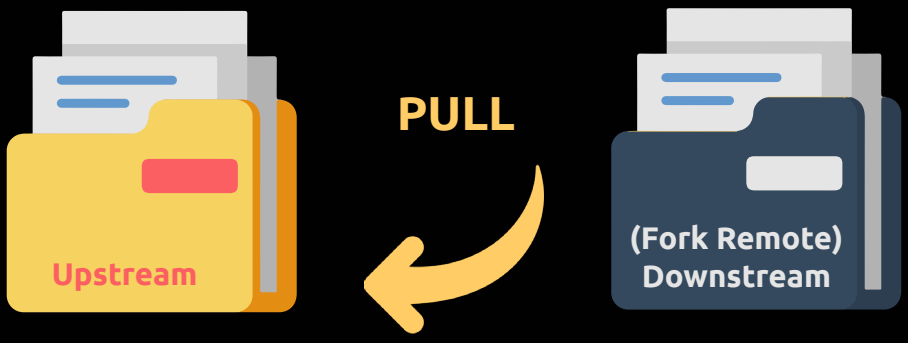
Tudo começa quando realizamos um **Fork do Upstream**, e a partir daí realizamos o clone para nosso repositório local:



Nesse momento nós podemos alterar o nosso repositório local, seja adicionando **novas funcionalidades** ou **corrigindo bugs**.

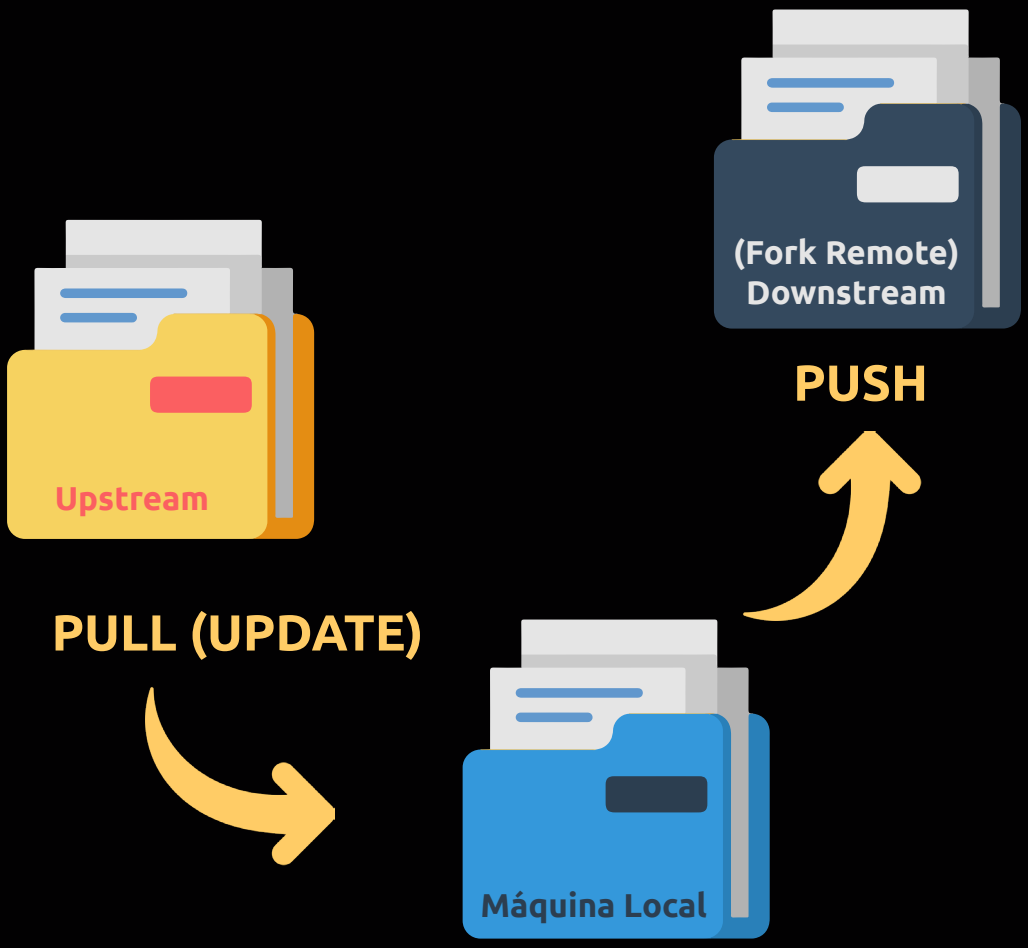
Após isso você pode fazer um push da sua aplicação local para o seu Fork normalmente:

E a partir do seu repositório Fork, você poderá fazer uma solicitação de Pull para o Upstream:



Quando você realiza o **Pull Request**, o dono do repositório upstream receberá uma solicitação junto da branch que você trabalhou com as alterações que você fez.

Se o dono do upstream decidir aceitar suas solicitações, elas começarão a fazer parte do upstream, bastando apenas que você faça uma breve sincroniza do repositório original com seu Fork:



Fazendo um resumo do fluxo do **Fork Workflow**, podemos separa-lo em 6 passos:

1) Realizar um Fork do repositório original pelo GitHub.

2) Clonar o seu repositório Fork para a máquina local. **[git clone]**

3) Realizar as modificações na pasta do projeto por meio de outra branch. **[git add., git commit]**

4) Enviar as alterações para o repositório Fork Remoto. **[git push]**

5) Enviar as solicitações de Pull Request para o repositório original. **[Fazemos isso pelo GitHub]**

6) Caso o dono do repositório upstream aceitar suas alterações, mais tardar você deve realizar um **git pull upstream** para atualizar sua máquina local, seguido do git push para atualizar o fork remoto.

Uma dica importante é não utilizar a branch master para fazer seus pedidos de **Pull Request**, em vez disso, faça com que essa branch seja aquela que esteja sincronizada com as últimas alterações realizadas no upstream.

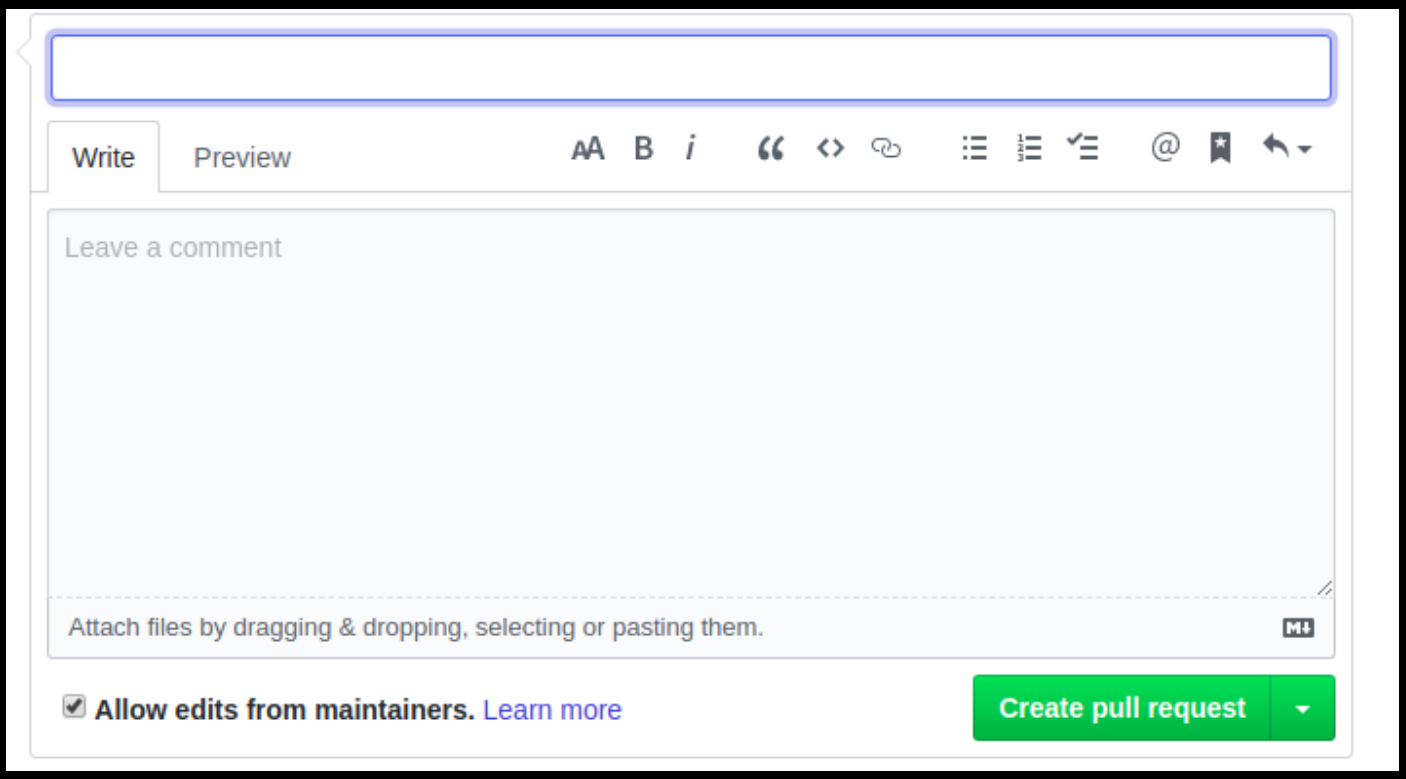
Sendo assim, sempre que você for trabalhar no **repositório cópia (Fork)**, sempre crie uma nova branch para realizar as modificações no projeto, e em seguida siga os passos 3, 4 e 5 vistos acima.

Portanto, todas as sugestões de alterações devem ser feitas em branchs separadas, isto é, para evitar conflitos quando temos mais de uma pessoa trabalhando no mesmo projeto.

Realizando um Pull Request

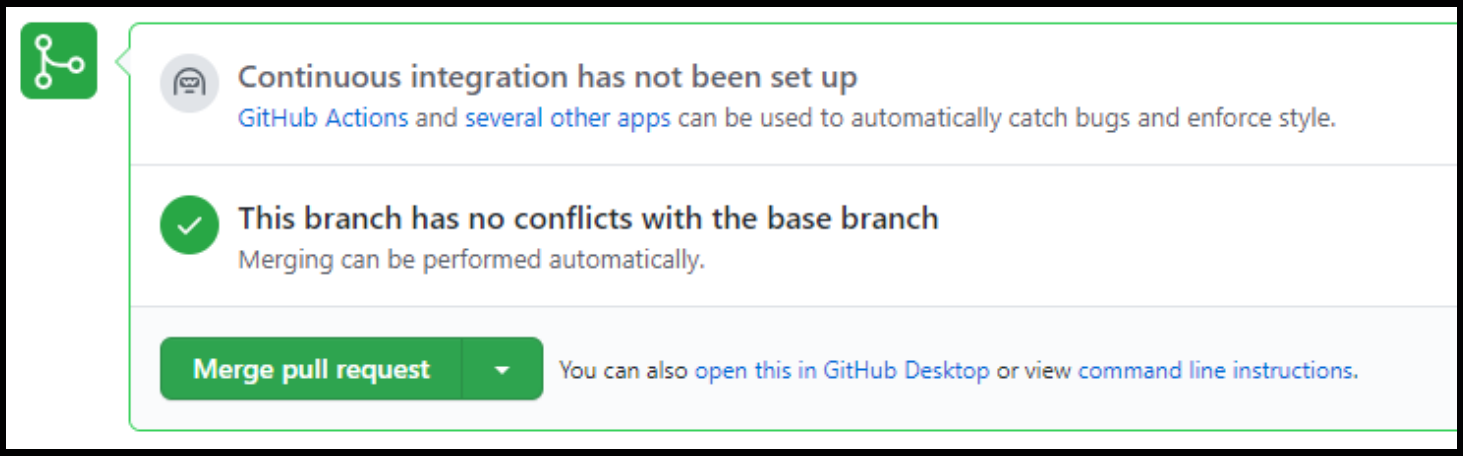
Quando você cria um Fork, faz um clone para a máquina local, realiza alterações pela sua branch, e por fim faz um push para o repositório cópia, o GitHub nos dá um alerta de que um **Pull Request** é possível de ser feito:

Ao clicar no botão **"Compare & Pull Request"** seremos levados a uma nova tela aonde poderemos fazer comentários e enviar as modificações para o repositório original.



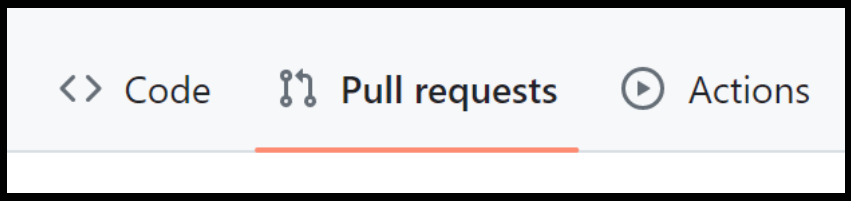
Quando criamos um **Pull Request** o Git pega nossa branch e associa dentro do repositório original (upstream), de modo que o dono do repositório tenha acesso à sua branch com seus commits.

Se nosso commit estiver associado com uma determinada Issue do repositório original, após o **Pull Request** seremos redirecionados a tela daquela Issue aonde encontraremos uma mensagem alegando que houve um pull request de modo a solucionar aquela Issue.

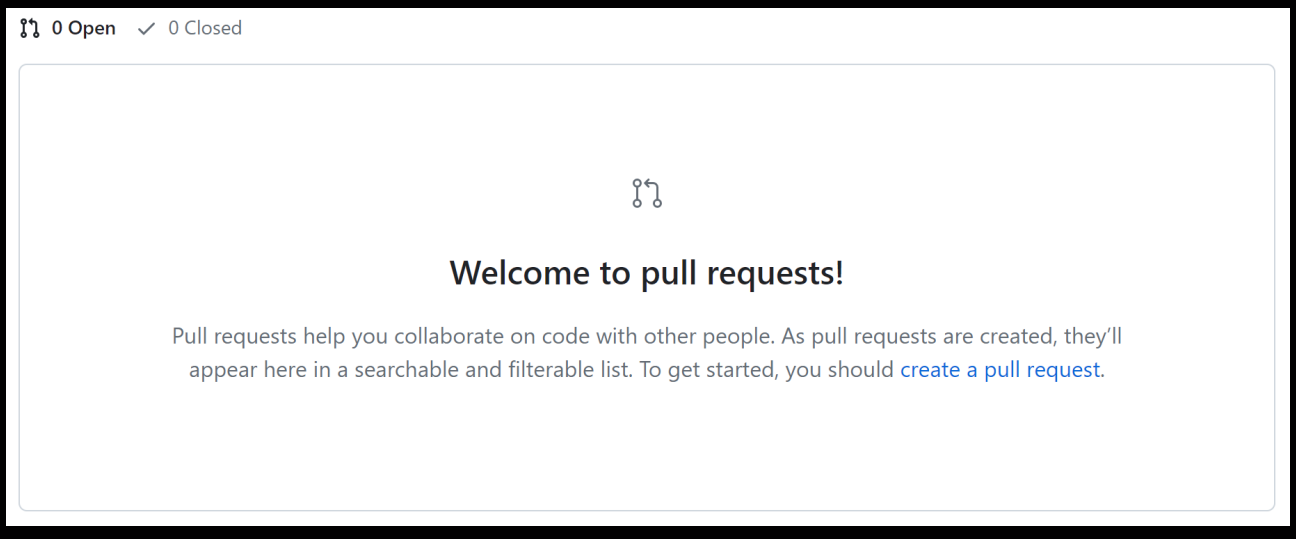


Aceitando um Pull Request

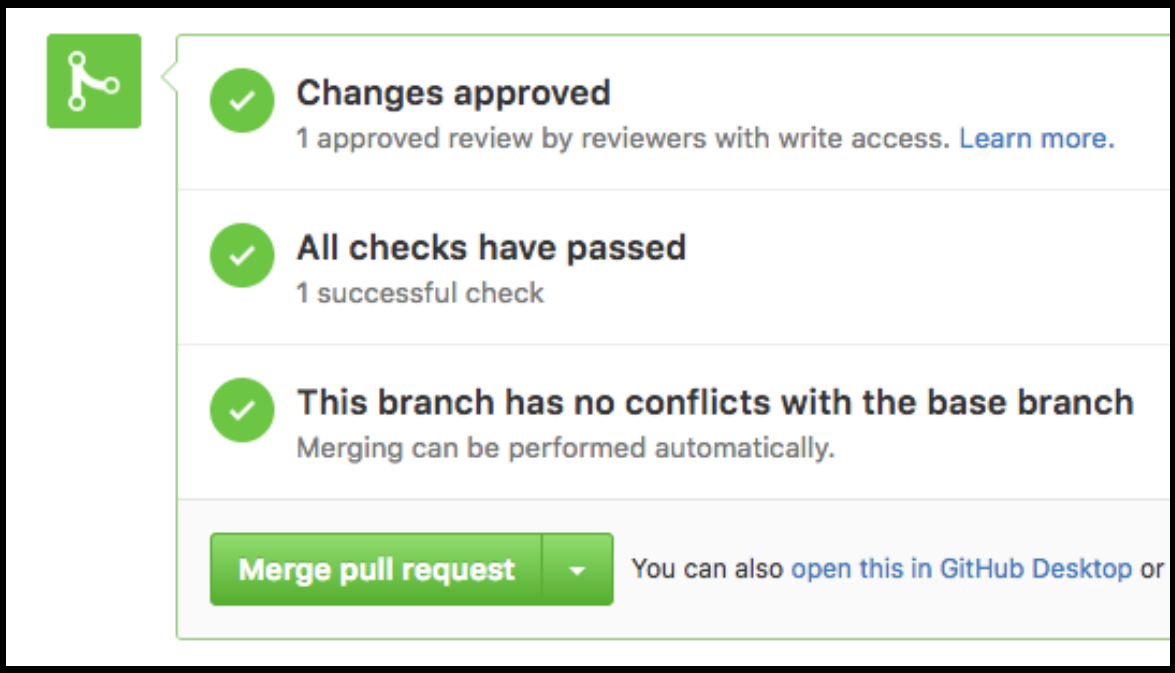
Quando alguém faz um Fork do seu repositório e realiza um **Pull Request**, você como dono daquele repositório terá acesso a uma nova aba:



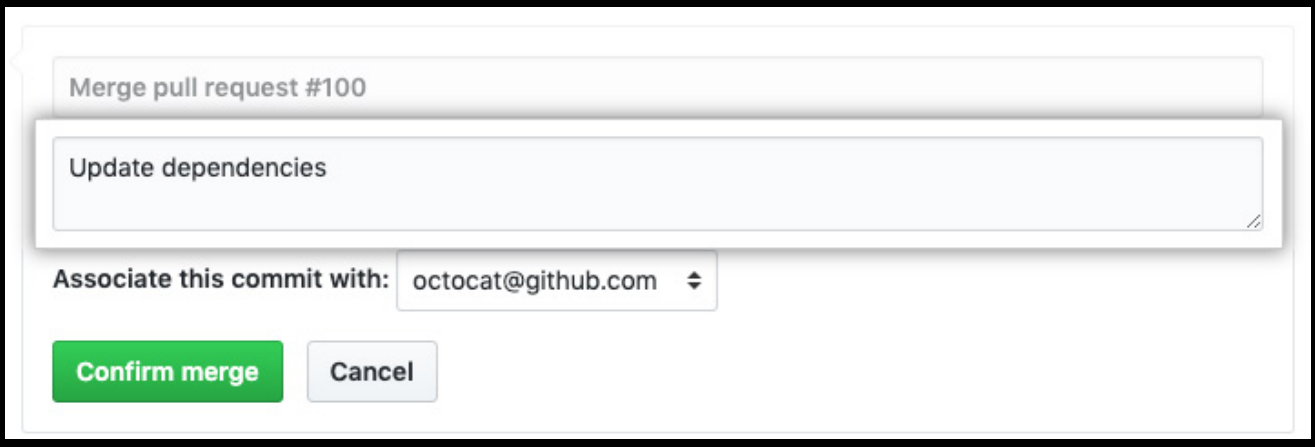
Ao clicar nesta aba, você será direcionado a uma tela aonde poderá visualizar todas requisições feitas:



Caso houver pull requests disponíveis, ao clicar em uma deles, um novo botão chamado **“Merge Pull Request”** estará disponível para uso:



Ao clicar nele, o Github vai solicitar que você digite uma mensagem, como se você estivesse fazendo um **merge** normalmente de uma branch para outra:

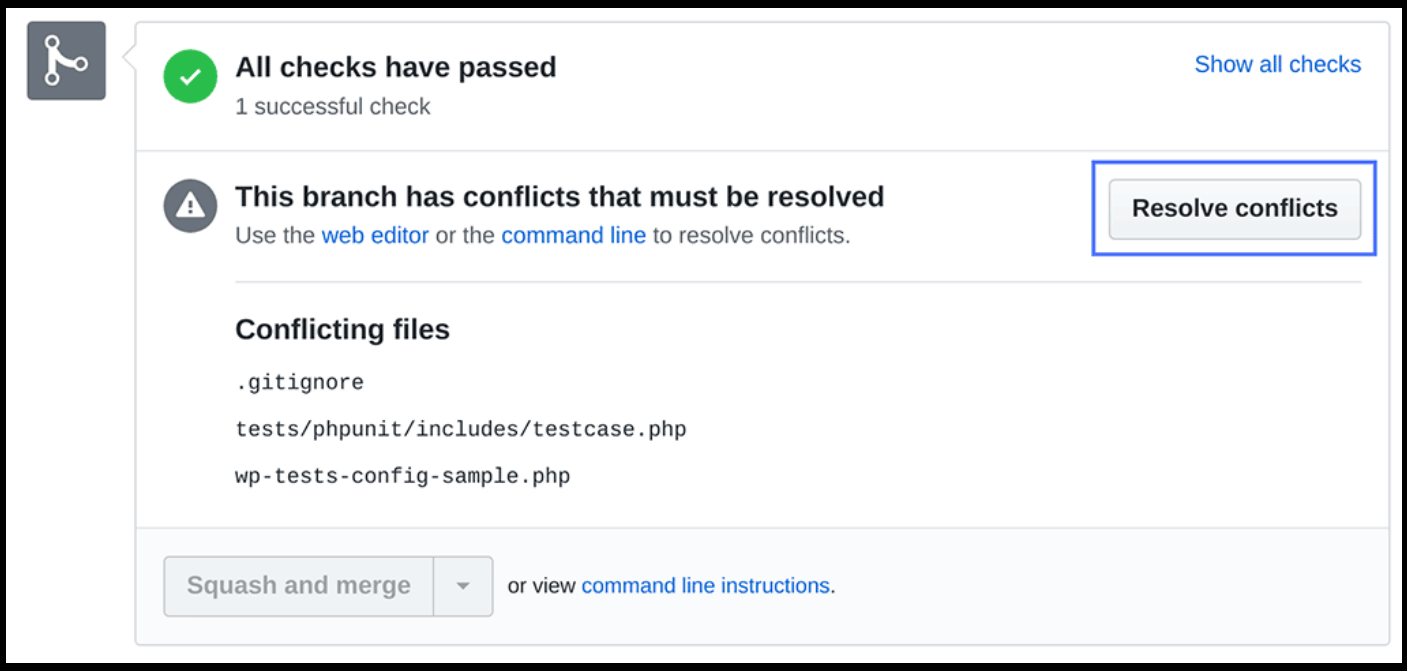


Por fim, você deve clicar em **“Confirm Merge”** para realizar o merge.

E pronto, neste momento o seu repositório original estará sincronizado com as atualizações feitas por um outro usuário dentro de um repositório cópia (Fork).

Pull Request com Conflito

Caso houver conflitos durante o Pull Request, o criador do repositório original vai se deparar com uma mensagem diferente, onde o botão **"Merge Pull Request"** será substituído por **"Resolve Conflict"**:



Em casos como esses, você pode resolver os conflitos (caso desejar), ou atrelar uma mensagem para o usuário (na Issue) dizendo que ele precisa resolver tais conflitos para que você (dono do repositório) possa realizar um merge sem precisar resolver conflitos.

Caso você esteja do outro lado da moeda, ou seja, se você não for o dono do repositório, e acabar recebendo uma mensagem do dono daquele repositório, de que você precisa resolver o conflito por si mesmo.

Basta voltar para o terminal do Git, atualizar a pasta do seu projeto local com as novas alterações do repositório upstream, em seguida, resolva os possíveis conflitos, seguindo o mesmo procedimento para atualizar seu repositório fork.

A única diferença, é que agora você não precisa enviar manualmente o Pull Request do seu repositório Fork, pois ele já estará conectado com o upstream.

Pull Request com Rebase

Você sabia que alguns desenvolvedores não gostam muito da ideia do Fork com Merge, pois segundo eles: *"Acaba poluindo muito o histórico do Projeto"*.

Talvez você já tenha escutado isso antes, mas a verdade é que quanto mais desenvolvedores e mais repositórios cópia nós temos, mais bagunçado pode se tornar o histórico do repositório original.

Com isso em mente, nós podemos fazer o uso do nosso velho amigo **Rebase**.

Considerando que já temos:

+ Um repositório "Forkado".

+ Fizemos o clone para a pasta local.

+ Criamos uma branch para resolver tal issue ou criar nova funcionalidade.

+ Já fizemos as alterações que queríamos.

+ Realizamos o commit, o push e o pull request.

Vamos imaginar que após o **Pull Request**, acabou ocorrendo um conflito em um dos arquivos que trabalhamos.

Sendo assim, uma alternativa é executar o rebase dentro da branch que criamos.

O único problema é que se fizermos isso na branch atual nós temos que usar a flag -f (force), pois como você já sabe, ela já está sincronizada com o processo de merge que está ocorrendo no repositório upstream.

Como consequência, essa flag irá reescrever todo o histórico de commits, e acredito que o dono do repositório upstream não quer isso, não é verdade?

Segundo a própria documentação do Git, ele não recomenda que usemos a flag -f (force) para branchs que já foram publicadas.

E no exemplo acima, a branch já foi publicada, certo?

Outra alternativa bem comum que a maioria dos desenvolvedores fazem quando se deparam com esses conflitos é:

1) Criar uma nova branch a partir da branch que você já publicou:

```
git checkout -b issue-23-versao-2
```

(Considerando que a branch publicada se chama “issue-23”, foi criada uma nova branch chamada de “issue-23-versao-2” para indicar que é uma branch de correção de outra)

2) Realizar o Rebase nela, de modo a resolver conflitos:

```
git pull upstream master --rebase
```

```
#Resolva seus conflitos no Projeto
```

```
git add .
```

```
git rebase --continue
```

3) Realizar o push e fazer o Pull Request:

```
git push origin issue-23-versao-2
```

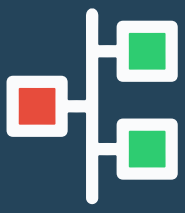
```
#Realize um Pull Request no GitHub
```

4) Entrar na Issue do repositório upstream (que tem a mensagem de conflito) e encerrar o Pull Request.

5) Entrar no repositório cópia (Fork), e submeter a nova branch (Pull Request).

Lembrando que pela própria plataforma do GitHub, nós podemos usar a hastag correspondente a Issue que estamos trabalhando, de forma a atrelar aquela branch com aquela issue, como, por exemplo:

Desse modo, o Merge poderá ser feito sem nenhum conflito e de forma totalmente limpa!



Removendo Branchs após o Pull

Uma das funcionalidades da própria plataforma do GitHub é excluir uma determinada branch após o processo de merge do Pull Request.

Você já deve ter percebido isso, mas após realizar o Merge de um repositório Fork, ainda na tela da Issue o GitHub nos mostrará um botão chamado **"Delete branch"**.

Basta clicar nele para remover a branch automaticamente sem a necessidade de usar o terminal de comando do Git para isso :)

END (y/n)

Gostou desse material?

Então não deixe de acessar a nossa seção de Git & GitHub do básico ao avançado.