Contribuindo para o Bitcoin—Dicas e Truques

Por Daniela Brozzoni

Tenho trabalhado em OSS relacionado ao Bitcoin há um bom tempo - tempo suficiente para não temer mais o `git reflog`.

Como você pode imaginar, houve um tempo em que eu ficava apavorada só de pensar em publicar meu código para que estranhos o vissem - felizmente, eu tinha amigos me ajudando, incentivando-me a sair da minha zona de conforto, lendo meu código antes de eu publicá-lo e, por que não, ajudando-me com conflitos de mesclagem (merge conflicts).

Há muitas pessoas inteligentes e humildes por aí que poderiam contribuir significativamente para o Bitcoin, mas não têm a mesma sorte que eu e se sentem bastante perdidas - esta é minha maneira de tentar ajudar. Neste post, cobrirei os fundamentos para começar a contribuir com projetos de código aberto:

Como escolher o projeto certo
Como começar a mergulhar no código
Como revisar código
Como escrever código

Estou ciente de que um post no blog não é tão útil quanto um mentor IRL (In Real Life), mas, ei, estou fazendo o que posso :)

Não espere que isso seja muito estruturado - é apenas uma coleção de anotações, pensamentos aleatórios e dicas de código, mais parecido com um fluxo de consciência do que com um livro - há grandes lacunas aqui e ali, provavelmente esqueci de explicar alguns passos importantes, provavelmente me perdi em detalhes de vez em quando.

Não se preocupe se algumas partes parecerem obscuras para você - você pode voltar a elas mais tarde. Alguns conceitos se referenciam circularmente, e você não pode entender A sem primeiro entender B, e não pode entender B sem primeiro entender A, e você não pode entender A sem primeiro entender B, e não pode entender B sem primeiro entender A, e ooooooopsie, StackOverflowError.

Nem preciso dizer que 99% das dicas listadas aqui vêm das experiências que tive nos projetos em que trabalhei - mas podem não se aplicar ao projeto para o qual você quer contribuir (Alguém aí mencionou o Cisne Negro?). Seja humilde, esteja pronto para se desculpar se fizer algo errado, e tenha em mente que não sou responsável se você fizer algo estúpido :P

Tenho certeza de que você encontrará algo útil neste post do blog, mesmo que não seja mais um completo iniciante. Aproveite!

Escolha um projeto
Encontre um projeto que pareça interessante para você. Você gosta de eletrônica? Considere contribuir para uma implementação de carteira de hardware. Você gosta de trabalhar em aplicativos móveis? Escolha sua carteira móvel favorita e comece a programar. Está pronto para um desafio? A Lightning Network é a escolha certa.

Alguns projetos regularmente incorporam muitos novos colaboradores, enquanto outros são desenvolvidos por uma pequena equipe e raramente recebem ajuda externa. Se você é realmente novo em programação, fique com projetos grandes e populares, pois terá mais pessoas ajudando você ao longo do caminho.

O Bitcoin Core é provavelmente a escolha mais popular, pois o projeto cobre muitos aspectos do Bitcoin, está cheio de mantenedores dispostos a ajudar os novatos, a documentação é impecável e todos adoram se gabar de contribuir para o Core, por algum motivo.

Mergulhando no código
Esta é uma fase muito importante que é frequentemente ignorada - antes de se apressar em escrever o código, você precisa dedicar um tempo para explorar o projeto, seus objetivos e suas limitações.

A fase de exploração é, como você adivinhou, sobre explorar. Deve ser ilimitada - pode durar minutos, horas ou dias. Brinque sem necessariamente ter um objetivo ou esperar que algo aconteça: experimente todos os recursos diferentes, clique em todos os botões, use todos os comandos possíveis e veja o que acontece. Algumas inspirações para você começar:

Qual é a natureza do projeto? Que problema ele está resolvendo? Para quem ele é?
Você está olhando para uma biblioteca ou um executável? Está pronto para produção?
Existe documentação do projeto ou algum tutorial oficial?
Onde os desenvolvedores se comunicam? Existe algum chat público (Discord, Slack, IRC, etc.) que você possa entrar?
Clone, compile (se necessário) e execute o código (se possível) - esse passo já pode ser difícil.
Você geralmente pode encontrar alguma ajuda no `README.md` do projeto - por exemplo, o `README.md` do HWI (Hardware Wallet interface) declara claramente como instalar e usar o `hwi`. O do Bitcoin Core, por outro lado, simplesmente diz para você verificar o diretório doc, que contém uma explicação detalhada para cada plataforma.

Alguns projetos não têm nenhuma documentação detalhada sobre como construir ou executar - isso geralmente significa que é trivial fazê-lo. Por exemplo, o BDK é construído da mesma forma que qualquer outro projeto Rust: usando `cargo build`. Sim, poderíamos escrever isso no `README.md`, mas presumimos que, se você quer contribuir para uma biblioteca Rust, já usou Rust pelo menos uma vez e sabe como compilar código Rust.

Se você estiver explorando uma biblioteca, considere construir um pequeno programa usando-a, para se familiarizar com a API. Não precisa ser algo útil ou excepcionalmente

complexo, já que é apenas para fins educacionais. No caso do BDK, você pode escrever 25 linhas para verificar o saldo de um descritor de testnet.

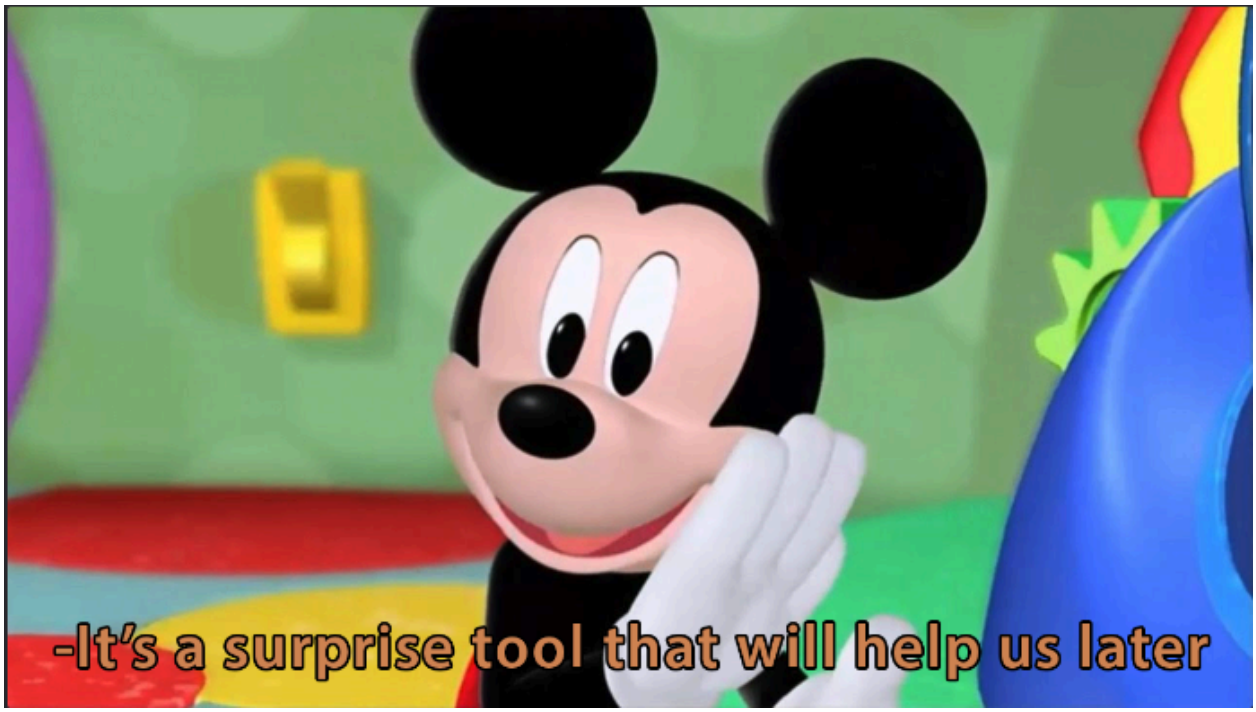
Então, vamos olhar para o código em si. Como ele está estruturado? Identifique os diferentes módulos disponíveis e qual função eles desempenham. Por exemplo, se olharmos para a estrutura do diretório do BDK:

```
bdk git:master
❯ ls -la src
total 76
blockchain
database
descriptor
doctest.rs
error.rs
keys
lib.rs
psbt
testutils
types.rs
wallet
```

Podemos ver que há muitos módulos diferentes - vamos tentar adivinhar o que eles fazem:

`blockchain` - interage com a blockchain, provavelmente?
`database` - mh, isso parece que está interagindo com um banco de dados. Provavelmente é usado para salvar dados?
`descriptor` - ah, descritores de Bitcoin! Eu li sobre eles em algum lugar, mas não me lembro do que fazem... procede para o google Bitcoin descriptor
`keys` - algum tipo de código para interagir com chaves de bitcoin? xpubs, WIFs, etc.
`psbt` - mh, transações parcialmente assinadas... Provavelmente contém algumas utilidades para lidar com PSBTs.
`testutils` - oh, testes! Eu os encontrei!
`wallet` - ah, este deve ser o código para a carteira em si. Provavelmente usa todos os outros módulos para construir uma carteira.

Como você pode ver, não estamos tentando entender perfeitamente o que cada parte do código faz: queremos apenas ter uma intuição sobre o que os diferentes módulos fazem. Isso será útil mais tarde.



Último passo da fase de exploração: os testes!

Normalmente, é declarado em algum lugar como executar os testes - se não, tente executar as ferramentas usuais para uma linguagem: `pytest`, `cargo test`, etc.

Se você não entender algo, sinta-se à vontade para pedir ajuda, mas não faça isso imediatamente - passe um tempo investigando o problema, sabendo que mesmo que não consiga chegar à solução por conta própria, ainda aprenderá algo ao longo do caminho. Tente lembrar que os mantenedores do projeto provavelmente estão lidando com muitos outros novatos como você, e tente não ser um fardo para eles.

Exemplo de mensagem em servidor do Summer of Bitcoin

espero não parecer muito severo aqui porque não estou tentando atacar ninguém especificamente, só quero dar minha opinião sobre todos os novatos que estão se juntando ao SoB: é incrível que você esteja empolgado em trabalhar em nossos projetos, mas, sendo honesto, sinto que vocês todos tendem a pedir ajuda um pouco demais. e acho que isso pode ser um pouco contraproducente para seu crescimento, na minha opinião é muito importante que você desenvolva a habilidade de depurar/cavar através do código se quiser trabalhar como desenvolvedor no futuro

isso não é sobre "você está desperdiçando meu tempo" ou "meu tempo é mais valioso que o seu": eu poderia simplesmente lhe dar a resposta e isso levaria alguns minutos, mas em vez disso estou disposto a gastar mais tempo explicando coisas se você tiver perguntas técnicas ou se quiser aprender como algumas coisas de baixo nível funcionam

mas meu ponto é que você deve tentar depurar problemas por conta própria primeiro, e só vir aqui quando você tiver tentado algumas coisas que todas falharam

Aviso: Muitos projetos de Bitcoin usam IRC para comunicações, que é, e isso vindo de alguém nascido não há muito tempo, sem julgamento aqui, só estou tentando ser honesto, por favor, não se ofenda, uma plataforma de mensagens estranha. Se você tentar se conectar a https://libera.chat, verá exatamente o que quero dizer.

Eu removi a estranheza usando o Matrix conectado ao IRC, o que significa que posso falar com as pessoas na plataforma estranha, enquanto fico em uma plataforma normal (eu sei, mágico). Você pode se registrar em https://matrix.org (a maneira fácil) ou hospedar seu próprio servidor (a maneira não tão fácil). E funciona (até quebrar).

Revisando código
Revisar código é a arte de ler o código de outra pessoa para verificar sua correção antes de mesclá-lo no branch principal. A etapa de revisão de código é crucial, mas pode ser entediante às vezes, ou pelo menos menos divertida do que escrever código! No entanto, você deve ajudar a revisar PRs abertos - quanto mais pessoas lerem o código com cuidado, menor a probabilidade de bugs críticos passarem. Mesmo se você for novo na base de código, sua revisão pode ser muito útil: novatos são mais propensos a desafiar suposições, são melhores em perceber quando a documentação está faltando e podem fazer perguntas difíceis de responder, mas extremamente perspicazes (vale mencionar: nem toda pergunta que você faz é perspicaz).

As diretrizes do Bitcoin Core pedem que os colaboradores revisem entre 5 e 15 PRs para cada PR que abrirem - esse número obviamente depende do tamanho do projeto e do número de PRs abertos todos os dias. Não há como descobrir o número exato para cada projeto, mas, como regra geral: tente dar mais do que receber. **Revise mais PRs do que abrir.**

Mas… como você realmente revisa o código?

Encontre algo para revisar
Abra a página de PRs e encontre algo para revisar. Evite grandes refatorações de código e fique com PRs de pequeno a médio porte ou mudanças/adicionamentos de testes. Encontre alguns PRs que soem interessantes para você - se você realmente não gosta de rede, talvez não comece revisando o código P2P do Bitcoin Core.

A fase de revisão
Primeiro, um aviso: revisar código leva tempo. PRs menores podem levar 15 minutos para revisar, enquanto PRs maiores geralmente levam horas. Obviamente, você não deve passar 6

horas olhando para o mesmo pedaço de código: faça uma pausa de vez em quando (não com muita frequência!), e não se sinta pressionado a terminar a revisão no mesmo dia em que começou, ou mesmo a terminar a revisão antes de publicá-la: desde que você diga que apenas deu uma olhada rápida no código, está tudo bem postar algumas perguntas e voltar ao PR mais tarde.

Tente entender qual é o objetivo do PR - está corrigindo um bug? Está adicionando uma funcionalidade? Se um problema estiver vinculado, leia-o.

Busque o código

```
git fetch origin pull/NUMERO_DO_PR/head && git checkout FETCH_HEAD
```

Execute os testes e, se possível, faça alguns testes manuais também, tentando reproduzir o bug antigo (que deve ter desaparecido agora!) ou brincando com o novo recurso. Se você estiver revisando uma correção de bug, também deve tentar reproduzir o bug na branch master, para ter certeza de que está entendendo claramente qual era o problema.

Hora de ler o código!

Adoro revisar commit por commit, se o autor souber como fazer commits corretamente. E desprezo revisar commit por commit se o autor não souber! (Não se preocupe, vou explicar como fazer commits).

Se for um projeto grande o suficiente, há uma grande chance de que o autor do PR saiba o que está fazendo, e revisar commit por commit deve ser viável.

Leia o código e tente entender o que cada mudança faz - primeiro, em um nível alto: como a estrutura do código está mudando? O autor está adicionando novos métodos? Novas classes? Novos módulos? Modificando os antigos? Essas não são perguntas fáceis: leve o tempo necessário para entender como o código era e como está mudando.

Em segundo lugar, revise o PR em um nível mais baixo: como cada método é implementado? Qual é o algoritmo usado? Pode ser melhorado?

Você deve ativamente tentar quebrar o código. Tanto no nível alto, identificando estruturas que podem ser simplificadas ou que podem ser acessadas incorretamente de fora; quanto no nível baixo, procurando bugs na implementação e coisas que podem dar errado. O último é mais fácil para um iniciante, então talvez concentre-se nisso.

Por fim, verifique se foram adicionados testes ao PR (é raro que um PR não precise de novos!), e quando houver testes, certifique-se de que o máximo de código possível esteja coberto.

:~$ Scalar School

Enquanto lê o código, escreva comentários sempre que encontrar algo que possa ser melhorado. Você pode colocar "nit:" na frente do seu comentário se não for realmente relevante e não bloquear a mesclagem. Por exemplo:

nit: Eu mudaria o nome da variável de `to_address` para `address`, pois é mais consistente com o resto do código

nit: Erro de digitação: deveria ser "platypus", não "platipus"

Você deve finalizar a revisão publicando seus comentários e, opcionalmente, colocando um veredicto, como **`tACK`**, `NACK`, `utACK`**,** etc. - a gíria é explicada aqui:

https://github.com/bitcoin/bitcoin/issues/6100

Evite colocar um mero `ACK` se você não estiver confiante de que o código do PR está correto - se tiver alguma dúvida, ainda pode colocar um `ACK` parcial, declarando explicitamente quais partes do código não fazem sentido para você. Por exemplo:

https://github.com/bitcoindevkit/bdk/pull/614#pullrequestreview-1003451363

Lembre-se: revisar PRs é uma habilidade que leva tempo para se desenvolver. Continue praticando!

Se você quiser aprender mais (deve):

https://github.com/glozow/bitcoin-notes/blob/master/review-checklist.md
https://jonatack.github.io/articles/how-to-review-pull-requests-in-bitcoin-core
https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests (todo este documento mostra como usar o Github para revisão de PRs)

Escrevendo código
Os passos para escrever código em um projeto são os seguintes:

Encontre algo para fazer
Codifique
Faça um fork do repositório
Empurre o código para o seu fork
Abra um PR
Receba revisões e atualize o código conforme necessário
Quando os mantenedores estiverem satisfeitos, eles mesclarão seu código
Sucesso!
(opcional) Ostente no Twitter

:~$ Scalar School

Antes de começar a trabalhar, leia o arquivo `CONTRIBUTING.md` do projeto. Ele geralmente contém muitas coisas copiadas e coladas dizendo para você não ser um idiota e algumas dicas específicas do projeto - onde pedir ajuda, o processo de revisão, o processo de lançamento, etc.

Encontre algo para trabalhar
Procure pelo rótulo `good first issue` - esses problemas geralmente são realmente fáceis e uma boa maneira de conhecer a base de código.

Esta lista compila alguns good first issues abertos em projetos open-source no ecossistema Bitcoin. https://bitcoindevs.xyz/good-first-issues

Good first issues geralmente são relativamente escassos em um projeto - seja consciente com os outros e tente evitar resolver todos os good first issues sozinho! Depois de completar um ou dois deles, provavelmente é hora de começar a trabalhar em outra coisa. Pegue um problema de baixa prioridade e comece a codificar :)

Se você realmente não tem ideia do que fazer, aumentar a cobertura de testes é sempre útil. Aqui coloquei algumas instruções para os colaboradores do BDK, mas os passos são os mesmos para qualquer projeto: encontre uma maneira de gerar um relatório de cobertura de código, encontre trechos de código não cobertos pelos testes atuais, escreva novos testes :)

Git
https://rogerdudler.github.io/git-guide/

Fazendo um fork do repositório
https://docs.github.com/en/get-started/quickstart/fork-a-repo

Escrevendo mensagens de commit
https://cbea.ms/git-commit/

Higiene de commits
Um grande problema que todos enfrentam ao usar git é: com que frequência devo fazer commits? Devo colocar minhas 1000 mudanças em um enorme commit? Devo criar muitos commits incrementais e pequenos?

Pesquisei demais em busca do artigo perfeito sobre higiene de commits e não encontrei nada. Por um momento, pensei que ninguém compartilhava minhas ideias e me senti bastante sozinha no mundo. Então, encontrei isto: https://github.com/bitcoin/bitcoin/blob/master/CONTRIBUTING.md#committing-patches, e agora sou um humano feliz novamente.

Mais posts em blogs sobre o assunto (não são tão perfeitos quanto o link acima):

https://blog.lanesawyer.dev/17501/git-hygiene
https://medium.com/@martindeto/commit-hygiene-git-blame-and-squashing-commits-58d1f511ea83
https://medium.com/transmute-techtalk/improve-your-commit-hygiene-with-git-add-patch-3b7dd9c117c4

Assinando commits
https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits

Antes de empurrar (push)
Execute um `git log` e verifique se o histórico está limpo. Todos os seus commits devem ter uma mensagem de commit clara e, opcionalmente, uma descrição. Nenhum dos seus commits deve ter "lol tudo está quebrado" como mensagem.

Dica: se você não é um falante nativo de inglês, copie e cole a mensagem/título do commit em um verificador ortográfico - isso vai te poupar de algum constrangimento mais tarde (uma vez consegui escrever "whether" errado duas vezes na mesma mensagem de commit...)

Para cada commit, execute um `git show`. Verifique se você não deu commit por engano nas fotos da sua avó. E verifique se não deixou espaços em branco feios por aí.
Espaços em branco são os piores.

Se você estiver introduzindo novos recursos, certifique-se de que estão bem testados. Pense em todos os casos extremos, não teste apenas as condições normais! Ferramentas de cobertura de código podem ajudar.

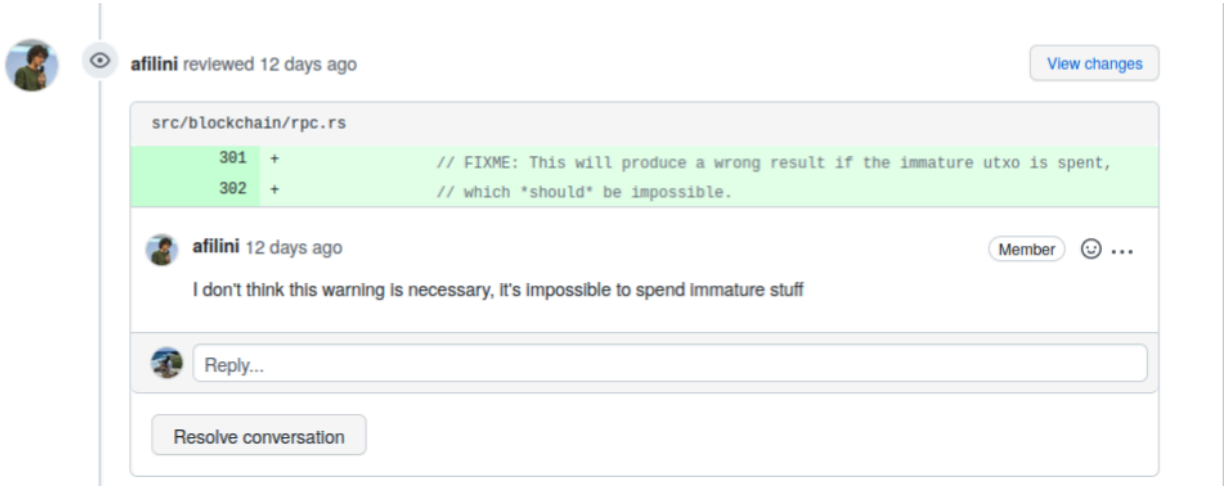
Abrindo PR
https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request

Lidando com revisões
Você abriu seu primeiro PR! Yay! É apenas uma questão de tempo até alguém educadamente lhe dizer por que seu código é ruim.

Quando isso acontecer, não fique triste - até os desenvolvedores experientes escrevem código ruim! Corrija seu código e empurre novamente. Lembre-se: higiene de commits! Não empurre um commit "corrigir coisas" no topo do seu PR - em vez disso, reescreva o histórico. Mais sobre isso depois.

Certifique-se de clicar em "resolver conversa" depois de, bem, resolver a conversa.

Uma conversa não resolvida parece assim:



Enquanto uma resolvida parece assim:

Como você pode ver, resolver ajuda a manter seu PR limpo :)

Quando tudo estiver corrigido, é hora de outra rodada de revisões.

Neste ponto, não sei se clicar no botão "solicitar revisão" é educado ou não, então simplesmente evito. Eu geralmente comento algo como:

"Obrigado pelas revisões, agora deve estar ok."

Ou eu apenas empurro, resolvo conversas e pacientemente espero que alguém volte ao meu PR.

Reescrevendo o histórico
Conseguir que seu código seja revisado é TUDO sobre reescrever o histórico. Isso acontece o tempo todo:

Alguém diz que uma função é feia
Alguém diz que sua mensagem de commit é feia
Alguém diz que, embora seu código seja muito legal, está fragmentado em 15 commits

diferentes que deveriam ser compactados
Alguém diz que, embora seus commits sejam muito legais, você deve reordená-los para tornar o histórico bisectável

Este artigo é incrível e explica tudo o que você precisa para sobreviver à sua primeira revisão de código: https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History

Se você nunca fez um rebase interativo antes, https://git-rebase.io/ pode ser útil.

Rebasing e corrigindo conflitos
Então, seu PR ficou aberto por um bom tempo e alguém comentou "por favor, rebase isso" - provavelmente porque suas mudanças conflitam com o master. Isso é facilmente resolvível:

```
# Este primeiro comando precisa ser feito apenas na primeira vez que você rebasear
git remote add upstream *link para o repositório upstream*
git fetch upstream
git rebase upstream/master
```

E então corrija os conflitos.

Um tutorial sobre como corrigir os conflitos: https://docs.openstack.org/doc-contrib-guide/additional-git-workflow/rebase.html - este usa comandos ligeiramente diferentes para atualizar a branch, mas o resultado é o mesmo.

Para evitar confusão, sugiro evitar usar `git merge` - fetch & rebase funciona perfeitamente e você não corre o risco de abrir um PR com commits de mesclagem feios, que os mantenedores certamente pedirão para você remover:

```
commit 207b4a481b171f86377dc656141af8c8b9111b39 (HEAD -> update_ci)
Merge: ee22627 c9b1b6d
Author: Daniela Brozzoni <danielabrozzoni@protonmail.com>
Date:   Thu Aug 18 10:45:11 2022 +0100

    Merge remote-tracking branch 'origin/master' into update_ci
```

Mais sobre a diferença entre rebase e merge: https://stackoverflow.com/a/804156

Por favor, não force push
Você **NUNCA** deve `git push --force`, nem mesmo quando acha que é realmente necessário - em vez disso, `git push --force-with-lease`, o que evita que você sobrescreva o trabalho de outras pessoas.

Mais informações: https://stackoverflow.com/a/52823955

Não se esqueça de se divertir
Vá a conferências e conheça pessoas. Participe de seminários online. Participe de BitDevs presenciais (se não houver um na sua cidade, organize um). Participe de clubes de revisão. Encontre uma maneira de fazer amigos no espaço e mantenha-se conectado com eles.

E apenas divirta-se.

Na época em que este texto foi escrito, os good first issues do BDK não eram nada escassos, e estou orgulhosa disso :) ↩