
This book is available in English.

Full translation available in

azərbaycan dili,
български език,
Deutsch,
Español,
Français,
Ελληνικά,
日本語,
한국어,
Nederlands,
Русский,
Slovenščina,
Tagalog,
Українська
简体中文,

Partial translations available in

Čeština,
Македонски,
Polski,
Српски,
Ўзбекча,
繁體中文,

Translations started for

Беларуская,
فارسی,
Indonesian,
Italiano,
Bahasa Melayu,
Português (Brasil),
Português (Portugal),
Svenska,
Türkçe.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

Chapters ▾

2nd Edition

3.2 Branches no Git - O básico de Ramificação (Branch) e Mesclagem (Merge)

O básico de Ramificação (Branch) e Mesclagem (Merge)

Vamos ver um exemplo simples de ramificação (*branching*) e mesclagem (*merging*) com um fluxo de trabalho que você pode vir a usar no mundo real. Você seguirá os seguintes passos:

1. Trabalhar um pouco em um website.
2. Criar um *branch* para um nova história de usuário na qual você está trabalhando.
3. Trabalhar um pouco neste novo *branch*.

Nesse ponto, você vai receber uma mensagem dizendo que outro problema é crítico e você precisa fazer a correção. Você fará o seguinte:

1. Mudar para o seu branch de produção.
2. Criar um novo branch para fazer a correção.

3. Após testar, fazer o merge do branch de correção, e fazer push para produção.

4. Voltar para sua história de usuário original e continuar trabalhando.

Ramificação Básica

Primeiramente, digamos que você esteja trabalhando em seu projeto e já tenha alguns commits no branch `master`.

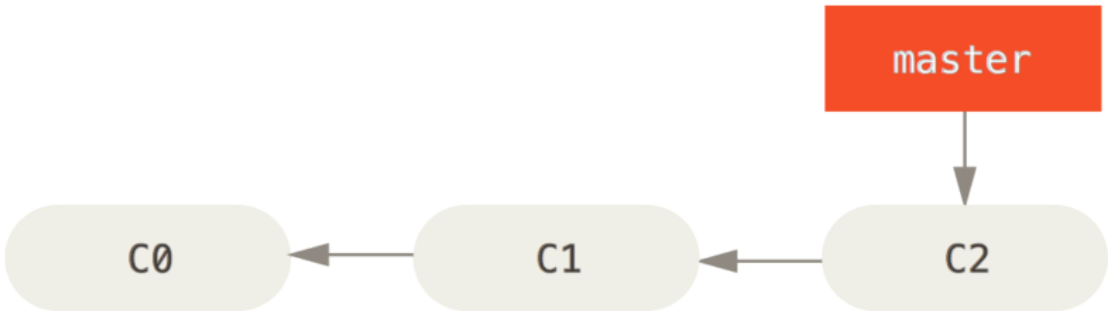

Figure 18. Um histórico de commits simples

Você decidiu que você vai trabalhar no chamado #53 em qualquer que seja o sistema de gerenciamento de chamados que a sua empresa usa.

Para criar um novo branch e mudar para ele ao mesmo tempo, você pode executar o comando `git checkout` com o parâmetro `-b`:

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

Esta é a abreviação de:

```
$ git branch iss53
$ git checkout iss53
```

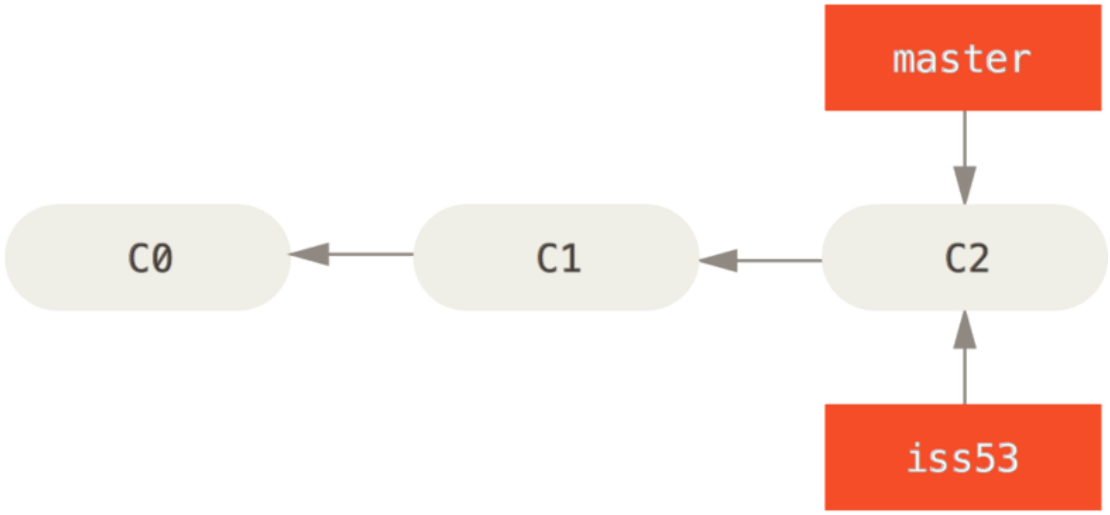

Figure 19. Criando um novo ponteiro de branch

Você trabalha no seu website e adiciona alguns commits.

Ao fazer isso, você move o branch `iss53` para a frente, pois este é o branch que está selecionado, ou *checked out*(isto é, seu `HEAD` está apontando para ele):

```
$ vim index.html
$ git commit -a -m 'Create new footer [issue 53]'
```

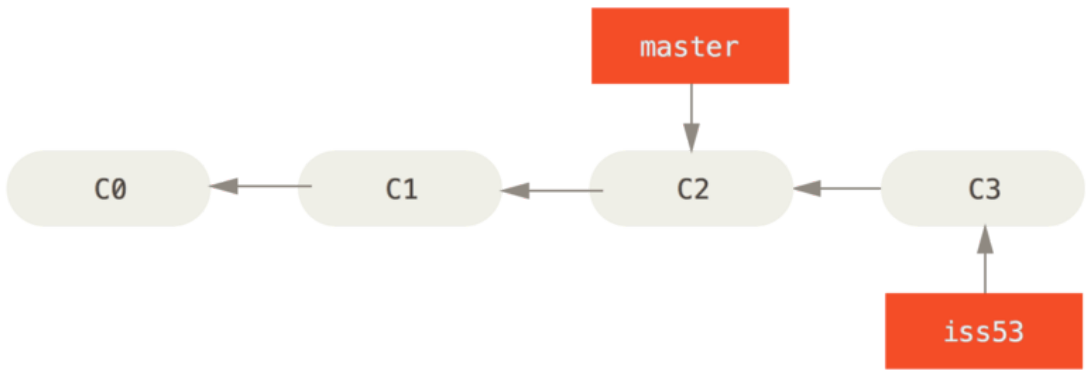

Figure 20. O branch `iss53` moveu para a frente graças ao seu trabalho

Agora você recebe a ligação dizendo que há um problema com o site, e que você precisa corrigí-lo imediatamente. Com o Git, você não precisa enviar sua correção junto com as alterações do branch `iss53` que já fez. Você também não precisa se esforçar muito para desfazer essas alterações antes de poder trabalhar na correção do erro em produção. Tudo o que você precisa fazer é voltar para o seu branch `master`.

Entretanto, antes de fazer isso, note que se seu diretório de trabalho ou stage possui alterações ainda não commitadas que conflitam com o branch que você quer usar, o Git não deixará que você troque de branch. O melhor é que seu estado de trabalho atual esteja limpo antes de trocar de branches. Há maneiras de contornar isso (a saber, o comando stash e commit com a opção --ammend) que iremos cobrir mais tarde, em Stashing and Cleaning. Por agora, vamos considerar que você fez commit de todas as suas alterações, de forma que você pode voltar para o branch `master`:

```
$ git checkout master
Switched to branch 'master'
```

Neste ponto, o diretório de trabalho de seu projeto está exatamente da forma como estava antes de você começar a trabalhar no chamado #53, e você pode se concentrar na correção. Isso é importante de se ter em mente: quando você troca de branches, o Git reseta seu diretório de trabalho para a forma que era na última vez que você commitou naquele branch. O Git adiciona, remove e modifica arquivos automaticamente para se assegurar que a sua cópia de trabalho seja igual ao estado do branch após você adicionar o último commit a ele.

Seu próximo passo é fazer a correção necessária; Vamos criar um branch chamado `hotfix` no qual trabalharemos até a correção estar pronta:

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'Fix broken email address'
[hotfix 1fb7853] Fix broken email address
 1 file changed, 2 insertions(+)
```

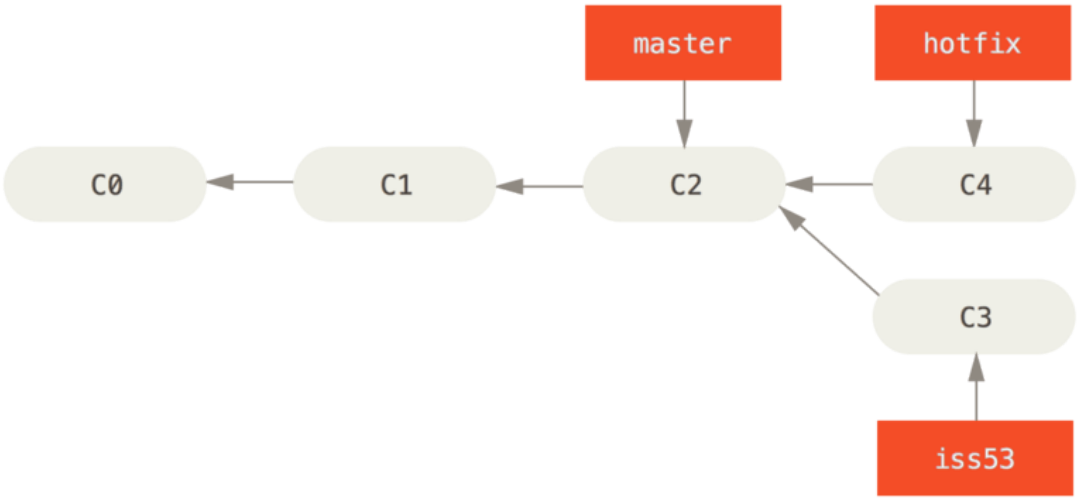

Figure 21. Branch de correção (hotfix) baseado em `master`

Você pode executar seus testes, se assegurar que a correção está do jeito que você quer, e finalmente mesclar o branch `hotfix` de volta para o branch `master` para poder enviar para produção. Para isso, você usa o comando `git merge` command:

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

Você vai notar a expressão “fast-forward” nesse merge. Já que o branch `hotfix` que você mesclou aponta para o commit `C4` que está diretamente à frente do commit `C2` no qual você está agora, o Git simplesmente move o ponteiro para a frente. Em outras palavras, quando você tenta mesclar um commit com outro commit que pode ser alcançado por meio do histórico do primeiro commit, o Git simplifica as coisas e apenas move o ponteiro para a frente porque não há nenhum alteração divergente para mesclar — isso é conhecido como um merge “fast-forward.”

Agora, a sua alteração está no snapshot do commmit para o qual o branch `master` aponta, e você você enviar a correção.

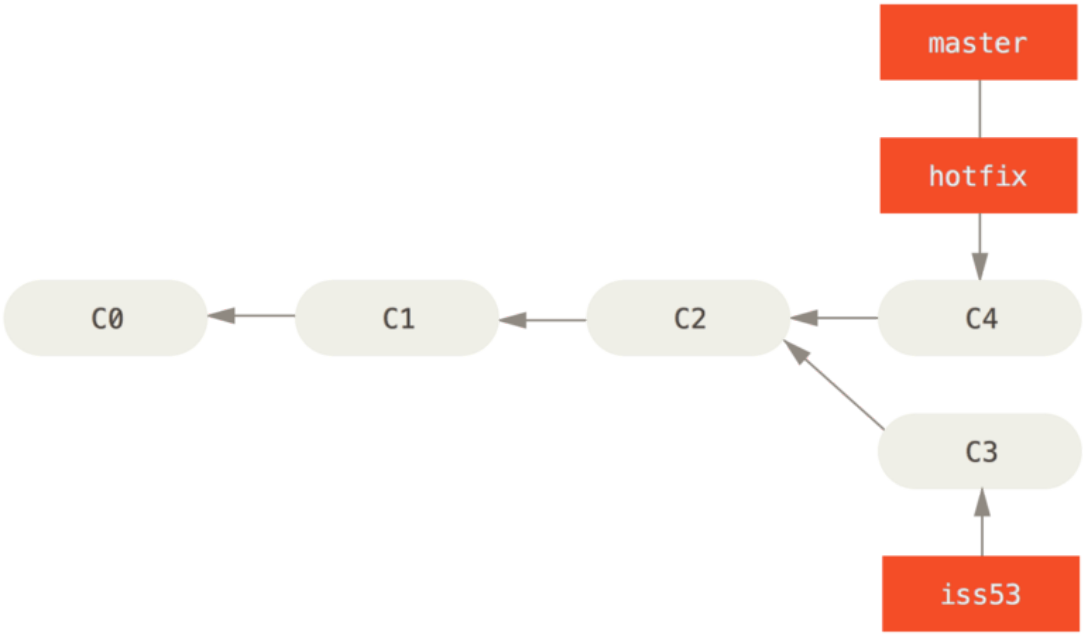

Figure 22. o branch `master` sofre um "fast-forward" até `hotfix`

Assim que a sua correção importantíssima é entregue, você já pode voltar para o trabalho que estava fazendo antes da interrupção. Porém, você irá antes excluir o branch `hotfix`, pois ele já não é mais necessário — o branch `master` aponta para o mesmo lugar. Você pode remover o branch usando a opção `-d` com o comando `git branch`:

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

Agora você pode retornar ao branch com seu trabalho em progresso na *issue* #53 e continuar trabalhando.

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'Finish the new footer [issue 53]'
[iss53 ad82d7a] Finish the new footer [issue 53]
1 file changed, 1 insertion(+)
```

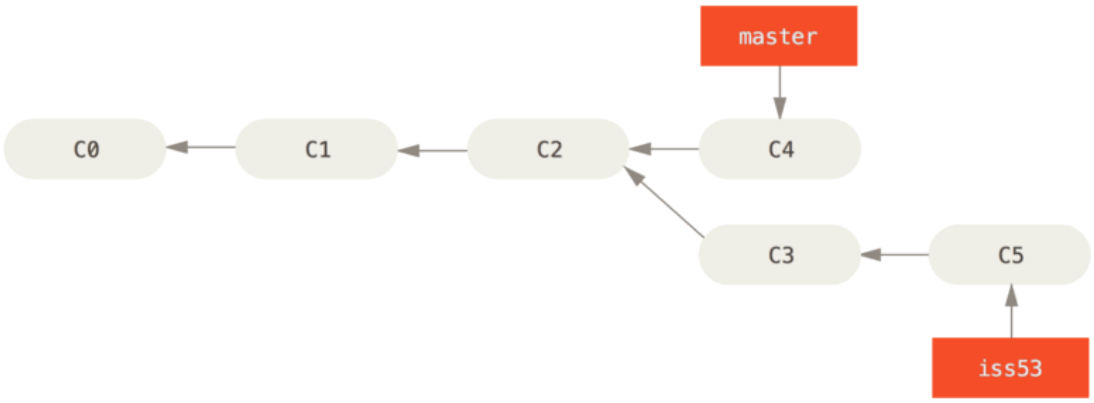

Figure 23. Continuando o trabalho no branch `iss53`

É importante frisar que o trabalho que você fez no seu branch `hotfix` não está contido nos arquivos do seu branch `iss53`. Caso você precise dessas alterações, você pode fazer o merge do branch `master` no branch `iss53` executando `git merge master`, ou você pode esperar para integrar essas alterações até que você decida mesclar o branch `iss53` de volta para `master` mais tarde.

Mesclagem Básica

Digamos que você decidiu que o seu trabalho no chamado #53 está completo e pronto para ser mesclado de volta para o branch `master`. Para fazer isso, você precisa fazer o merge do branch `iss53`, da mesma forma com que você mesclou o branch `hotfix` anteriormente. Tudo o que você precisa fazer é mudar para o branch que receberá as alterações e executar o comando `git merge`:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

Isso é um pouco diferente do merge anterior que você fez com o branch `hotfix`. Neste caso, o histórico de desenvolvimento divergiu de um ponto mais antigo. O Git precisa trabalhar um pouco mais, devido ao fato de que o commit no seu branch atual não é um ancestral direto do branch cujas alterações você quer integrar. Neste caso, o Git faz uma simples mesclagem de três vias (*three-way merge*), usando os dois snapshots referenciados pela ponta de cada branch e o ancestral em comum dos dois.

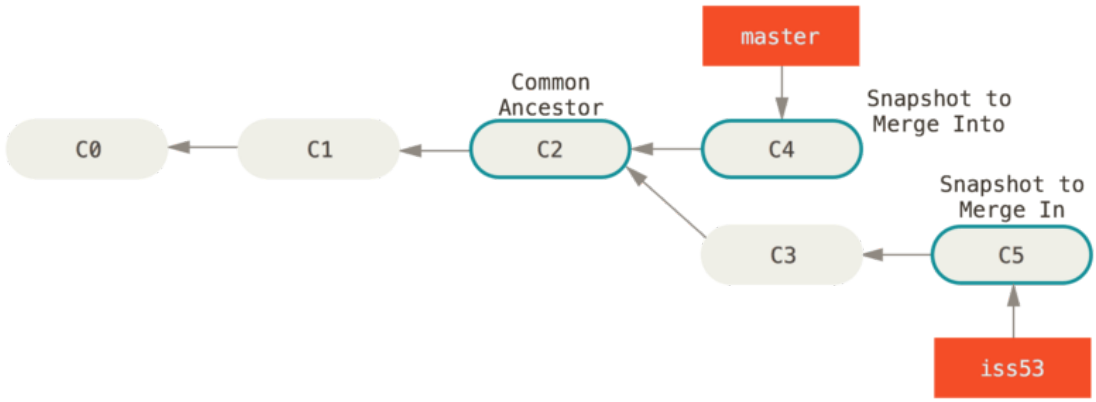

Figure 24. Três snapshots usados em um merge típico

Ao invés de apenas mover o ponteiro do branch para a frente, o Git cria um novo snapshot que resulta desse merge em três vias e automaticamente cria um novo commit que aponta para este snapshot. Esse tipo de commit é chamado de commit de merge, e é especial porque tem mais de um pai.

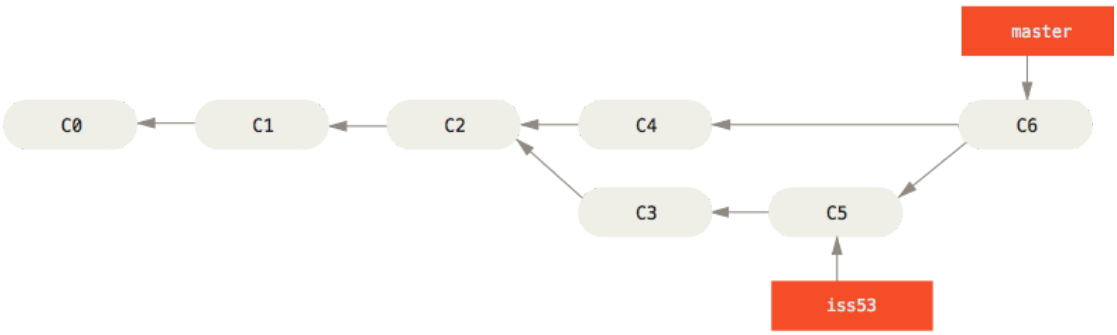

Figure 25. Um commit de merge

Agora que seu trabalho foi integrado, você não precisa mais do brnach `iss53`. Você pode encerrar o chamado no seu sistema e excluir o branch:

```
$ git branch -d iss53
```

Conflitos Básicos de Merge

De vez em quando, esse processo não acontece de maneira tão tranquila. Se você mudou a mesma parte do mesmo arquivo de maneiras diferentes nos dois branches que você está tentando mesclar, o Git não vai conseguir integrá-los de maneira limpa. Se a sua correção para o problema #53 modificou a mesma parte de um arquivo que também foi modificado em `hotfix`, você vai ter um conflito de merge que se parece com isso:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

O Git não criou automaticamente um novo commit de merge. Ele pausou o processo enquanto você soluciona o conflito. Para ver quais arquivos não foram mesclados a qualquer momento durante um conflito de merge, você pode executar `git status`:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Qualquer arquivo que tenha conflitos que não foram solucionados é listado como *unmerged*("não mesclado"). O Git adiciona símbolos padrão de resolução de conflitos nos arquivos que têm conflitos, para que você possa abrí-los manualmente e solucionar os conflitos. O seu arquivo contém uma seção que se parece com isso:

```
<div id="footer">
 please contact us at support@github.com
</div>
```

Isso significa que a versão em `HEAD` (o seu branch `master`, porque era o que estava selecionado quando você executou o comando merge) é a parte superior daquele bloco (tudo após `=======`), enquanto que a versão no branch `iss53` contém a versão na parte de baixo. Para solucionar o conflito, você precisa escolher um dos lados ou mesclar os conteúdos diretamente. Por exemplo, você pode resolver o conflito substituindo o bloco completo por isso:

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

Essa solução tem um pouco de cada versão, e as linhas com os símbolos `<<<<<<<`, `=======`, e `>>>>>>>` foram completamente removidas. Após solucionar cada uma dessas seções em cada arquivo com conflito, execute `git add` em cada arquivo para marcá-lo como solucionado. Adicionar o arquivo ao stage o marca como resolvido para o Git.

Se você quiser usar uma ferramenta gráfica para resolver os conflitos, você pode executar `git mergetool`, que inicia uma ferramente de mesclagem visual apropriada e guia você através dos conflitos:

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge
Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
```

Caso você queira usar uma ferramente de merge diferente da padrão (o Git escolheu `opendiff` neste caso porque o comando foi executado em um Mac), você pode ver todas as ferramentas suportadas listadas acima após “one of the following tools.” Você só tem que digitar o nome da ferramenta que você prefere usar.

Note Se você precisa de ferramentas mais avançadas para resolver conflitos mais complicados, nós abordamos mais sobre merge em Advanced Merging.

Após você sair da ferramenta, o Git pergunta se a operação foi bem sucedida. Se você responder que sim, o Git adiciona o arquivo ao stage para marcá-lo como resolvido. Você pode executar `git status` novamente para verificar que todos os conflitos foram resolvidos:

```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   index.html
```

Se você estiver satisfeito e verificar que tudo o que havia conflitos foi testado, você pode digitar `git commit` para finalizar o commit. A mensagem de confirmação por padrão é semelhante a esta:

```
Merge branch 'iss53'

Conflicts:
    index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   index.html
#
```

Se você acha que seria útil para outras pessoas olhar para este merge no futuro, você pode modificar esta mensagem de confirmação com detalhes sobre como você resolveu o conflito e explicar por que você fez as mudanças que você fez se elas não forem óbvias.

prev | next
About this site