---

This book is available in English.

Full translation available in

Español,
Français,
日本語,
한국어,
Nederlands,
Русский,
Українська
简体中文,

Partial translations available in

български език,
Čeština,
Indonesian,
Polski,
Српски,
Tagalog,
繁體中文,

Translations started for

Беларуская,
Deutsch,
فارسی,
Ελληνικά,
Italiano,
Bahasa Melayu,
Polski,
Português (Brasil),
Türkçe,
Ўзбекча.

---

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

Chapters ▾

1st Edition

# .2 Ramificação (Branching) no Git - Básico de Branch e Merge

## Básico de Branch e Merge

Vamos ver um exemplo simples de uso de branch e merge com um fluxo de trabalho que você pode usar no mundo real. Você seguirá esses passos:

1. Trabalhar em um web site.
2. Criar um branch para uma nova história em que está trabalhando.
3. Trabalhar nesse branch.

Nesse etapa, você receberá um telefonema informando que outro problema crítico existe e precisa de correção. Você fará o seguinte:

1. Voltar ao seu branch de produção.
2. Criar um branch para adicionar a correção.
3. Depois de testado, fazer o merge do branch da correção, e enviar para produção.
4. Retornar à sua história anterior e continuar trabalhando.

### Branch Básico

Primeiro, digamos que você esteja trabalhando no seu projeto e já tem alguns commits (veja Figura 3-10).

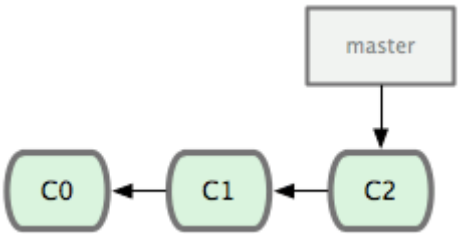


Figura 3-10. Um histórico de commits pequeno e simples.

Você decidiu que irá trabalhar na tarefa (issue) #53 do gerenciador de bugs ou tarefas que sua empresa usa. Para deixar claro, Git não é amarrado a nenhum gerenciador de tarefas em particular; mas já que a tarefa #53 tem um foco diferente, você criará um branch novo para trabalhar nele. Para criar um branch e mudar para ele ao mesmo tempo, você pode executar o comando `git checkout` com a opção `-b`:

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

Isso é um atalho para:

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

Figura 3-11 ilustra o resultado.

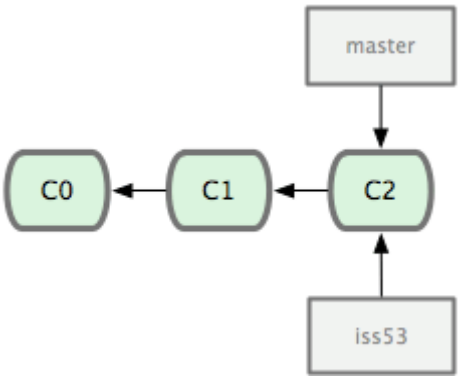


Figura 3-11. Criando um branch novo

Você trabalha no seu web site e faz alguns commits. Ao fazer isso o branch `iss53` avançará, pois você fez o checkout dele (isto é, seu HEAD está apontando para ele; veja a Figura 3-12):

```
$ vim index.html
$ git commit -a -m 'adicionei um novo rodapé [issue 53]'
```

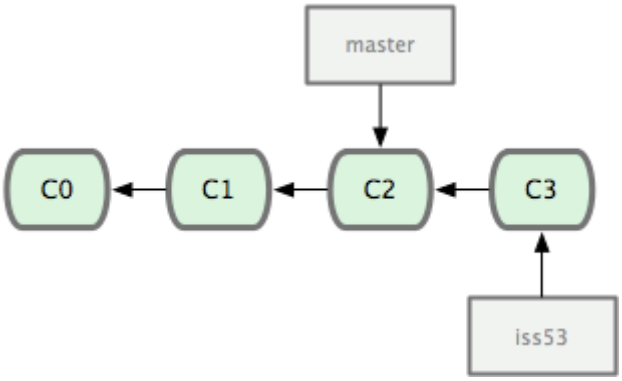


Figura 3-12. O branch iss53 avançou com suas modificações.

Nesse momento você recebe uma ligação dizendo que existe um problema com o web site e você deve resolvê-lo imediatamente. Com Git, você não precisa fazer o deploy de sua correção junto com as modificações que você fez no `iss53`, e você não precisa se esforçar muito para reverter essas modificações antes que você possa aplicar sua correção em produção. Tudo que você tem a fazer é voltar ao seu branch master.

No entanto, antes de fazer isso, note que seu diretório de trabalho ou área de seleção tem modificações que não entraram em commits e que estão gerando conflitos com o branch que você está fazendo o checkout, Git não deixará você mudar de branch. É melhor ter uma área de trabalho limpa quando mudar de branch. Existem maneiras de contornar esta situação (isto é, incluir e fazer o commit) que vamos falar depois. Por enquanto, você fez o commit de todas as suas modificações, então você pode mudar para o seu branch master:

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

Nesse ponto, o diretório do seu projeto está exatamente do jeito que estava antes de você começar a trabalhar na tarefa #53, e você se concentra na correção do erro. É importante lembrar desse ponto: Git restabelece seu diretório de trabalho para ficar igual ao snapshot do commit que o branch que você criou aponta. Ele adiciona, remove, e modifica arquivos automaticamente para garantir que sua cópia é o que o branch parecia no seu último commit nele.

Em seguida, você tem uma correção para fazer. Vamos criar um branch para a correção (hotfix) para trabalhar até a conclusão (veja Figura 3-13):

```
$ git checkout -b 'hotfix'
Switched to a new branch "hotfix"
$ vim index.html
$ git commit -a -m 'consertei o endereço de email'
[hotfix]: created 3a0874c: "consertei o endereço de email"
 1 files changed, 0 insertions(+), 1 deletions(-)
```

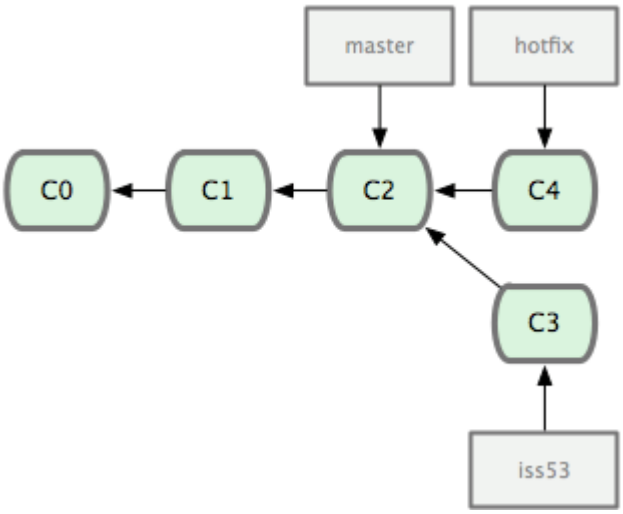


Figura 3-13. branch de correção (hotfix) baseado num ponto de seu branch master.

Você pode rodar seus testes, tenha certeza que a correção é o que você quer, e faça o merge no seu branch master para fazer o deploy em produção. Você faz isso com o comando `git merge`:

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast forward
 README |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)
```

Você irá notar a frase "Fast forward" no merge. Em razão do branch que você fez o merge apontar para o commit que está diretamente acima do commit que você se encontra, Git avança o ponteiro adiante. Em outras palavras, quando você tenta fazer o merge de um commit com outro que pode ser alcançado seguindo o histórico do primeiro, Git simplifica as coisas movendo o ponteiro adiante porque não existe modificações divergente para fazer o merge — isso é chamado de "fast forward".

Sua modificação está agora no snapshot do commit apontado pelo branch `master`, e você pode fazer o deploy (veja Figura 3-14).

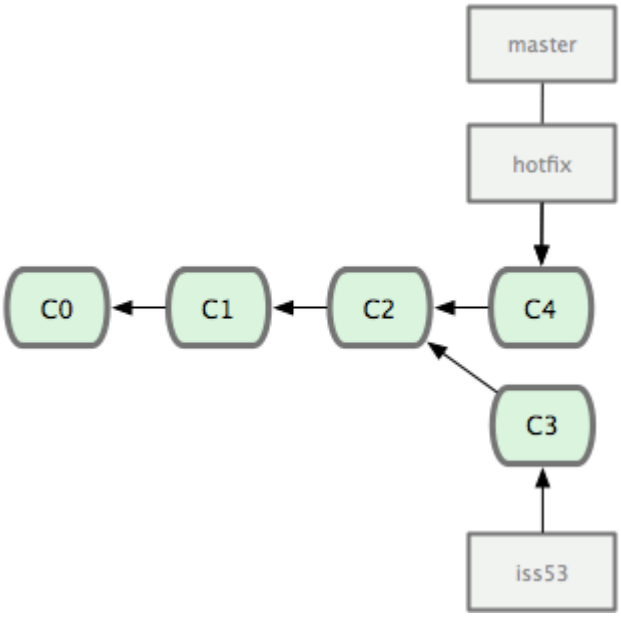


Figura 3-14. Depois do merge seu branch master aponta para o mesmo local que o branch hotfix.

Depois que a sua correção super importante foi enviada, você está pronto para voltar ao trabalho que estava fazendo antes de ser interrompido. No entanto, primeiro você apagará o branch `hotfix`, pois você não precisa mais dele — o branch `master` aponta para o mesmo local. Você pode excluí-lo com a opção `-d` em `git branch`:

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

Agora você pode voltar para o trabalho incompleto no branch da tafera #53 e continuar a trabalhar nele (veja Figura 3-15):

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'novo rodapé terminado [issue 53]'
[iss53]: created ad82d7a: "novo rodapé terminado [issue 53]"
 1 files changed, 1 insertions(+), 0 deletions(-)
```

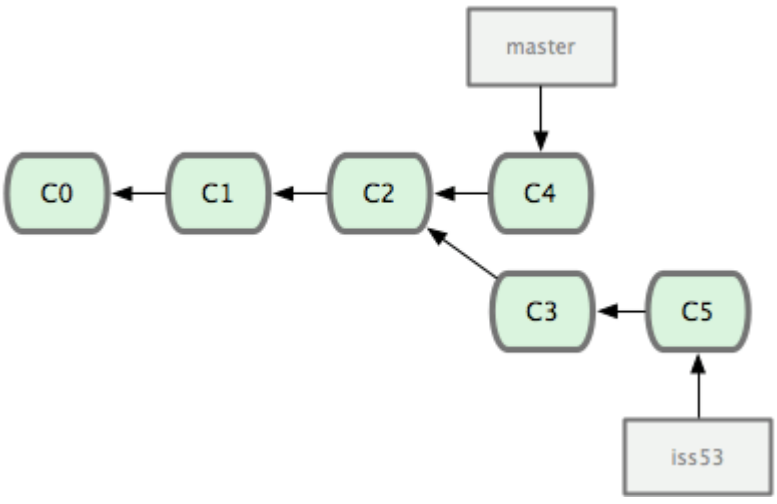


Figura 3-15. Seu branch iss53 pode avançar de forma independente.

Vale a pena lembrar aqui que o trabalho feito no seu branch `hotfix` não existe nos arquivos do seu branch `iss53`. Se você precisa incluí-lo, você pode fazer o merge do seu branch `master` no seu branch `iss53`

executando o comando `git merge master`, ou você pode esperar para integrar essas mudanças até você decidir fazer o pull do branch `iss53` no `master` mais tarde.

## Merge Básico

Suponha que você decidiu que o trabalho na tarefa #53 está completo e pronto para ser feito o merge no branch `master`. Para fazer isso, você fará o merge do seu branch `iss53`, bem como o merge do branch `hotfix` de antes. Tudo que você tem a fazer é executar o checkout do branch para onde deseja fazer o merge e então rodar o comando `git merge`:

```
$ git checkout master
$ git merge iss53
Merge made by recursive.
 README | 1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

Isso parece um pouco diferente do merge de `hotfix` que você fez antes. Neste caso, o histórico do seu desenvolvimento divergiu em algum ponto anterior. Pelo fato do commit no branch em que você está não ser um ancestral direto do branch que você está fazendo o merge, Git tem um trabalho adicional. Neste caso, Git faz um merge simples de três vias, usando os dois snapshots apontados pelas pontas dos branches e o ancestral comum dos dois. A Figura 3-16 destaca os três snapshots que Git usa para fazer o merge nesse caso.

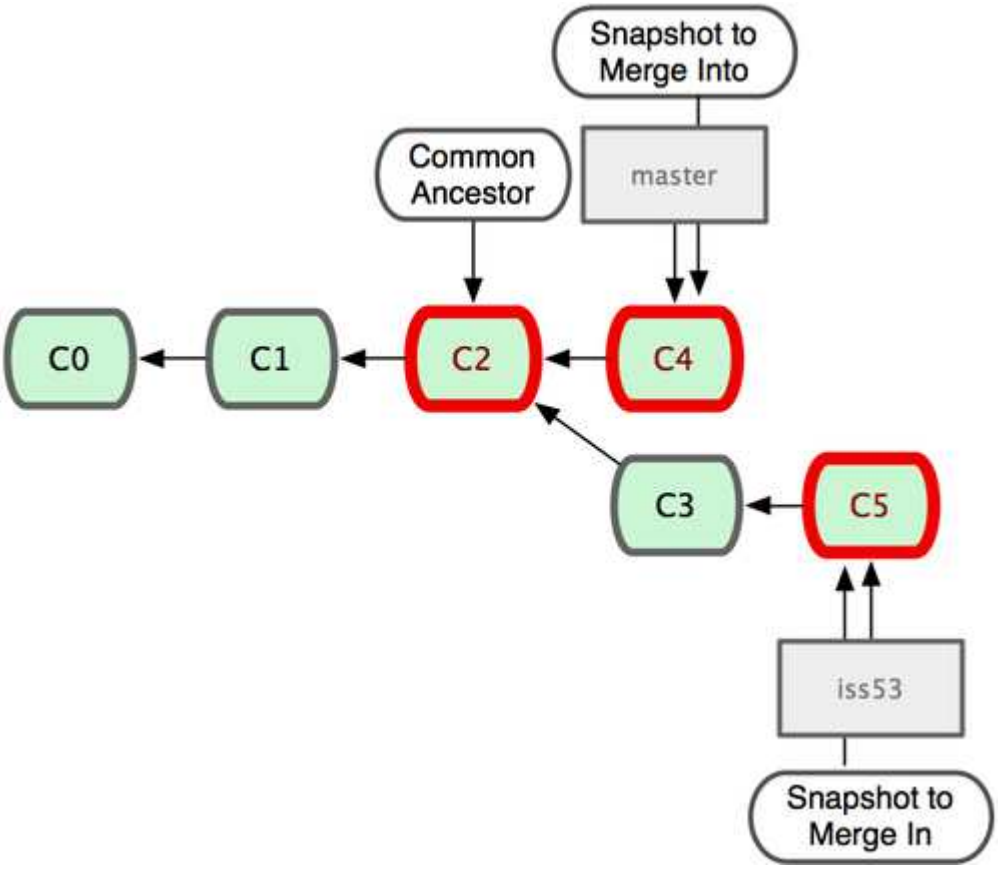


Figura 3-16. Git identifica automaticamente a melhor base ancestral comum para o merge do branch.

Em vez de simplesmente avançar o ponteiro do branch adiante, Git cria um novo snapshot que resulta do merge de três vias e automaticamente cria um novo commit que aponta para ele (veja Figura 3-17). Isso é conhecido como um merge de commits e é especial pois tem mais de um pai.

Vale a pena destacar que o Git determina o melhor ancestral comum para usar como base para o merge;

isso é diferente no CVS ou Subversion (antes da versão 1.5), onde o desenvolvedor que está fazendo o merge tem que descobrir a melhor base para o merge por si próprio. Isso faz o merge muito mais fácil no Git do que nesses outros sistemas.

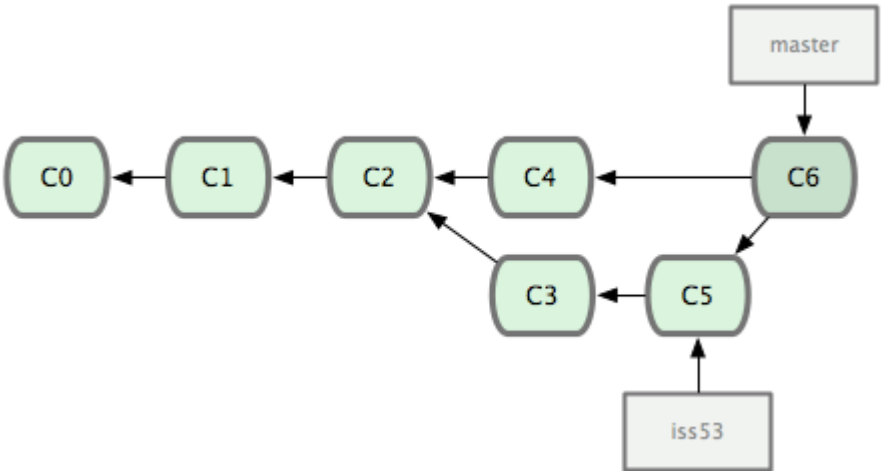


Figura 3-17. Git cria automaticamente um novo objeto commit que contém as modificações do merge.

Agora que foi feito o merge no seu trabalho, você não precisa mais do branch `iss53`. Você pode apagá-lo e fechar manualmente o chamado no seu gerenciador de chamados:

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

## Conflitos de Merge Básico

Às vezes, esse processo não funciona sem problemas. Se você alterou a mesma parte do mesmo arquivo de forma diferente nos dois branches que está fazendo o merge, Git não será capaz de executar o merge de forma clara. Se sua correção para o erro #53 alterou a mesma parte de um arquivo que `hotfix`, você terá um conflito de merge parecido 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.
```

Git não criou automaticamente um novo commit para o merge. Ele fez uma pausa no processo enquanto você resolve o conflito. Se você quer ver em quais arquivos não foi feito o merge, em qualquer momento depois do conflito, você pode executar `git status`:

```
[master*]$ git status
index.html: needs merge
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    unmerged:   index.html
#
```

Qualquer coisa que tem conflito no merge e não foi resolvido é listado como "unmerged". Git adiciona marcadores padrão de resolução de conflitos nos arquivos que têm conflitos, para que você possa abri-los manualmente e resolver esses conflitos. Seu arquivo terá uma seção parecida com isso:

```
<<<<<<< HEAD:index.html
<div id="footer">contato : email.support@github.com</div>
```

```
=======
<div id="footer">
  por favor nos contate em support@github.com
</div>
>>>>>>> iss53:index.html
```

Isso significa que a versão em HEAD (seu branch master, pois era isso que você tinha quando executou o comando merge) é a parte superior desse bloco (tudo acima de =======), enquanto a versão no seu branch `iss53` é toda a parte inferior. Para resolver esse conflito, você tem que optar entre um lado ou outro, ou fazer o merge do conteúdo você mesmo. Por exemplo, você pode resolver esse conflito através da substituição do bloco inteiro por isso:

```
<div id="footer">
por favor nos contate em email.support@github.com
</div>
```

Esta solução tem um pouco de cada seção, e eu removi completamente as linhas <<<<<<<, =======, e >>>>>>>. Depois que você resolveu cada uma dessas seções em cada arquivo com conflito, rode `git add` em cada arquivo para marcá-lo como resolvido. Colocar o arquivo na área de seleção o marcar como resolvido no Git. Se você quer usar uma ferramenta gráfica para resolver esses problemas, você pode executar `git mergetool`, que abre uma ferramenta visual de merge adequada e percorre os conflitos:

```
$ git mergetool
merge tool candidates: kdiff3 tkdiff xxdiff meld gvimdiff opendiff emerge vimdiff
Merging the files: index.html

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

Se você quer usar uma ferramenta de merge diferente da padrão (Git escolheu `opendiff` para mim, neste caso, porque eu rodei o comando em um Mac), você pode ver todas as ferramentas disponíveis listadas no topo depois de “merge tool candidates”. Digite o nome da ferramenta que você prefere usar. No *capítulo 7*, discutiremos como você pode alterar esse valor para o seu ambiente.

Depois de sair da ferramenta de merge, Git pergunta se o merge foi concluído com sucesso. Se você disser ao script que foi, ele coloca o arquivo na área de seleção para marcá-lo como resolvido pra você.

Se você rodar `git status` novamente para verificar que todos os conflitos foram resolvidos:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   index.html
#
```

Se você está satisfeito com isso, e verificou que tudo que havia conflito foi colocado na área de seleção, você pode digitar `git commit` para concluir o commit do merge. A mensagem de commit padrão é algo semelhante a isso:

```
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.
#
```

Você pode modificar essa mensagem com detalhes sobre como você resolveu o merge se você acha que isso seria útil para outros quando olharem esse merge no futuro — por que você fez o que fez, se não é óbvio.

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy