



--everything-is-local



- [About](#)
 - [Branching and Merging](#)
 - [Small and Fast](#)
 - [Distributed](#)
 - [Data Assurance](#)
 - [Staging Area](#)
 - [Free and Open Source](#)
 - [Trademark](#)
- [Documentation](#)
 - [Reference](#)
 - [Book](#)
 - [Videos](#)
 - [External Links](#)
- [Downloads](#)
 - [GUI Clients](#)
 - [Logos](#)
- [Community](#)

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.

1. [1. Começando](#)

1. 1.1 [Sobre Controle de Versão](#)
2. 1.2 [Uma Breve História do Git](#)
3. 1.3 [O Básico do Git](#)
4. 1.4 [A Linha de Comando](#)
5. 1.5 [Instalando o Git](#)
6. 1.6 [Configuração Inicial do Git](#)
7. 1.7 [Pedindo Ajuda](#)
8. 1.8 [Sumário](#)

2. [2. Fundamentos de Git](#)

1. 2.1 [Obtendo um Repositório Git](#)
2. 2.2 [Gravando Alterações em Seu Repositório](#)
3. 2.3 [Vendo o histórico de Commits](#)
4. 2.4 [Desfazendo coisas](#)
5. 2.5 [Trabalhando de Forma Remota](#)
6. 2.6 [Criando Tags](#)
7. 2.7 [Apelidos Git](#)
8. 2.8 [Sumário](#)

3. [3. Branches no Git](#)

1. 3.1 [Branches em poucas palavras](#)
2. 3.2 [O básico de Ramificação \(Branch\) e Mesclagem \(Merge\)](#)
3. 3.3 [Gestão de Branches](#)
4. 3.4 [Fluxo de Branches](#)
5. 3.5 [Branches remotos](#)
6. 3.6 [Rebase](#)
7. 3.7 [Sumário](#)

4. [4. Git on the Server](#)

1. 4.1 [The Protocols](#)
2. 4.2 [Getting Git on a Server](#)
3. 4.3 [Generating Your SSH Public Key](#)
4. 4.4 [Setting Up the Server](#)
5. 4.5 [Git Daemon](#)
6. 4.6 [Smart HTTP](#)
7. 4.7 [GitWeb](#)
8. 4.8 [GitLab](#)
9. 4.9 [Third Party Hosted Options](#)
10. 4.10 [Summary](#)

5. [5. Distributed Git](#)

1. 5.1 [Fluxos de Trabalho Distribuídos](#)
2. 5.2 [Contribuindo com um Projeto](#)
3. 5.3 [Maintaining a Project](#)
4. 5.4 [Summary](#)

1. [6. GitHub](#)

1. 6.1 [Configurando uma conta](#)
2. 6.2 [Contribuindo em um projeto](#)
3. 6.3 [Maintaining a Project](#)
4. 6.4 [Managing an organization](#)
5. 6.5 [Scripting GitHub](#)
6. 6.6 [Summary](#)

2. [7. Git Tools](#)

1. 7.1 [Revision Selection](#)
2. 7.2 [Interactive Staging](#)
3. 7.3 [Stashing and Cleaning](#)
4. 7.4 [Signing Your Work](#)
5. 7.5 [Searching](#)

- 6. 7.6 [Rewriting History](#)
- 7. 7.7 [Reset Demystified](#)
- 8. 7.8 [Advanced Merging](#)
- 9. 7.9 [Rerere](#)
- 10. 7.10 [Debugging with Git](#)
- 11. 7.11 [Submodules](#)
- 12. 7.12 [Bundling](#)
- 13. 7.13 [Replace](#)
- 14. 7.14 [Credential Storage](#)
- 15. 7.15 [Summary](#)

3. **8. Customizing Git**

- 1. 8.1 [Git Configuration](#)
- 2. 8.2 [Git Attributes](#)
- 3. 8.3 [Git Hooks](#)
- 4. 8.4 [An Example Git-Enforced Policy](#)
- 5. 8.5 [Summary](#)

4. **9. Git and Other Systems**

- 1. 9.1 [Git as a Client](#)
- 2. 9.2 [Migrating to Git](#)
- 3. 9.3 [Summary](#)

5. **10. Funcionamento Interno do Git**

- 1. 10.1 [Encanamento e Porcelana](#)
- 2. 10.2 [Objetos do Git](#)
- 3. 10.3 [Referências do Git](#)
- 4. 10.4 [Packfiles](#)
- 5. 10.5 [The Refspec](#)
- 6. 10.6 [Transfer Protocols](#)
- 7. 10.7 [Maintenance and Data Recovery](#)
- 8. 10.8 [Variáveis de ambiente](#)
- 9. 10.9 [Sumário](#)

1. **A1. Appendix A: Git em Outros Ambientes**

- 1. A1.1 [Graphical Interfaces](#)
- 2. A1.2 [Git in Visual Studio](#)
- 3. A1.3 [Git in Eclipse](#)
- 4. A1.4 [Git in Bash](#)
- 5. A1.5 [Git in Zsh](#)
- 6. A1.6 [Git in Powershell](#)
- 7. A1.7 [Resumo](#)

2. **A2. Appendix B: Embedding Git in your Applications**

- 1. A2.1 [Command-line Git](#)
- 2. A2.2 [Libgit2](#)
- 3. A2.3 [JGit](#)

3. **A3. Appendix C: Git Commands**

- 1. A3.1 [Setup and Config](#)
- 2. A3.2 [Getting and Creating Projects](#)
- 3. A3.3 [Basic Snapshotting](#)
- 4. A3.4 [Branching and Merging](#)
- 5. A3.5 [Sharing and Updating Projects](#)
- 6. A3.6 [Inspection and Comparison](#)
- 7. A3.7 [Debugging](#)
- 8. A3.8 [Patching](#)
- 9. A3.9 [Email](#)
- 10. A3.10 [External Systems](#)
- 11. A3.11 [Administration](#)
- 12. A3.12 [Plumbing Commands](#)

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 uma 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`.

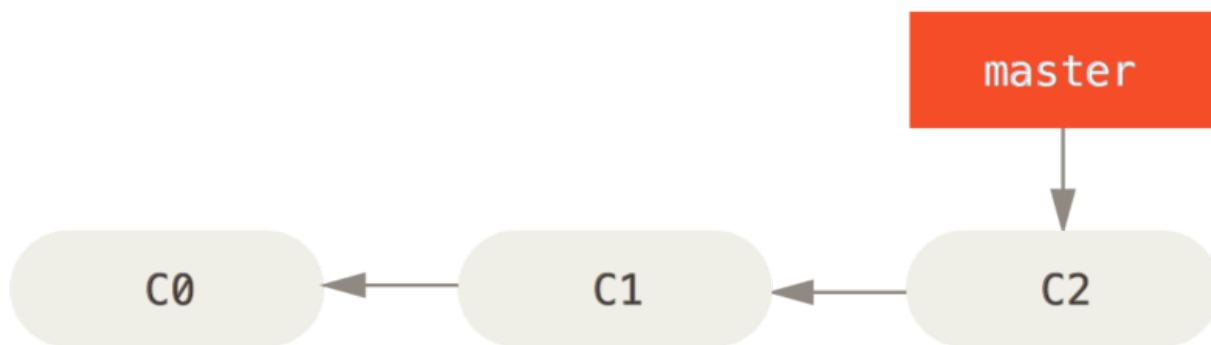


Figure 18. Um histórico de commits simples

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
```

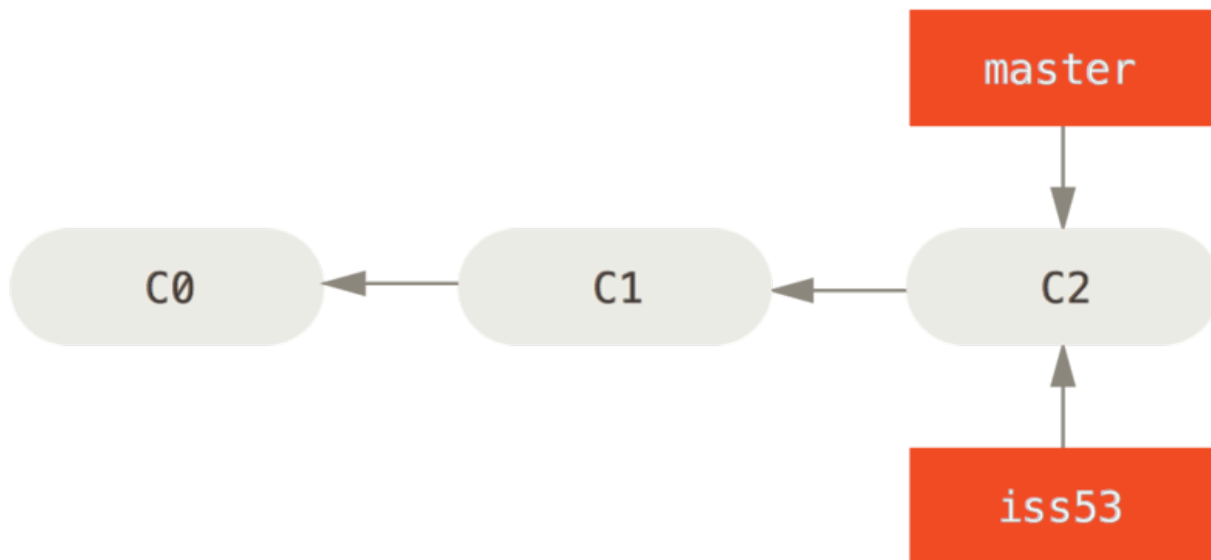


Figure 19. Criando um novo ponteiro de branch

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]'
```

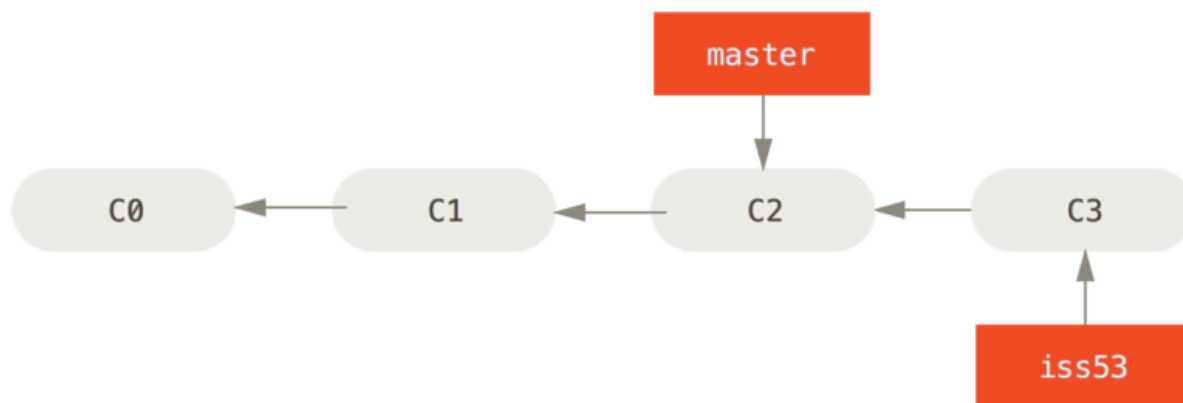


Figure 20. O branch `iss53` moveu para a frente graças ao seu trabalho

Agora você recebe a ligação dizendo que há um problema com o site, e que você precisa corrigi-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 `--amend`) 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(+)
```

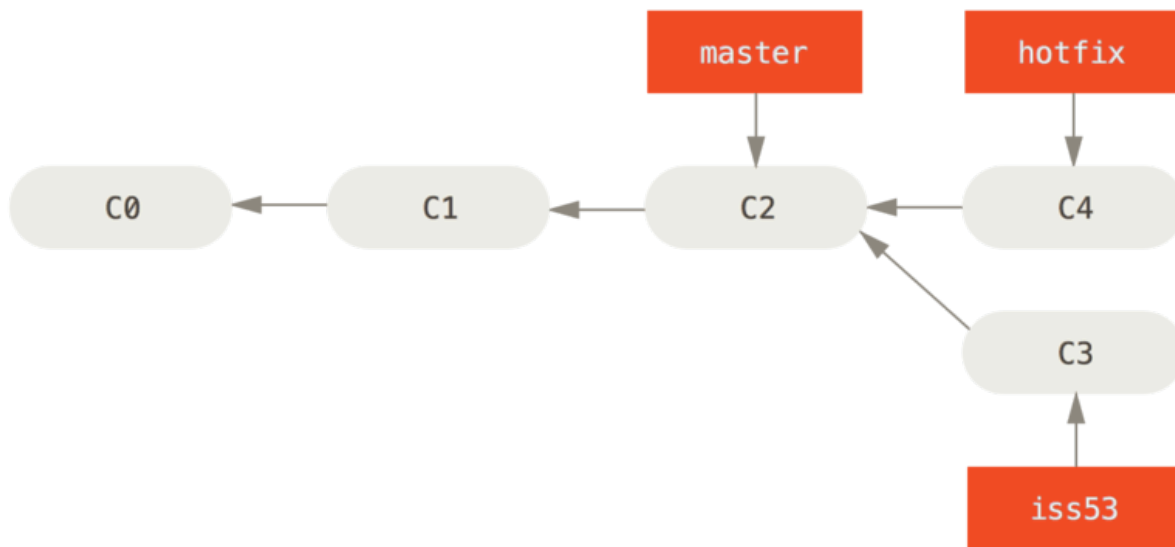


Figure 21. Branch de correção (hotfix) baseado em master

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 nenhuma alteração divergente para mesclar — isso é conhecido como um merge “fast-forward.”

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

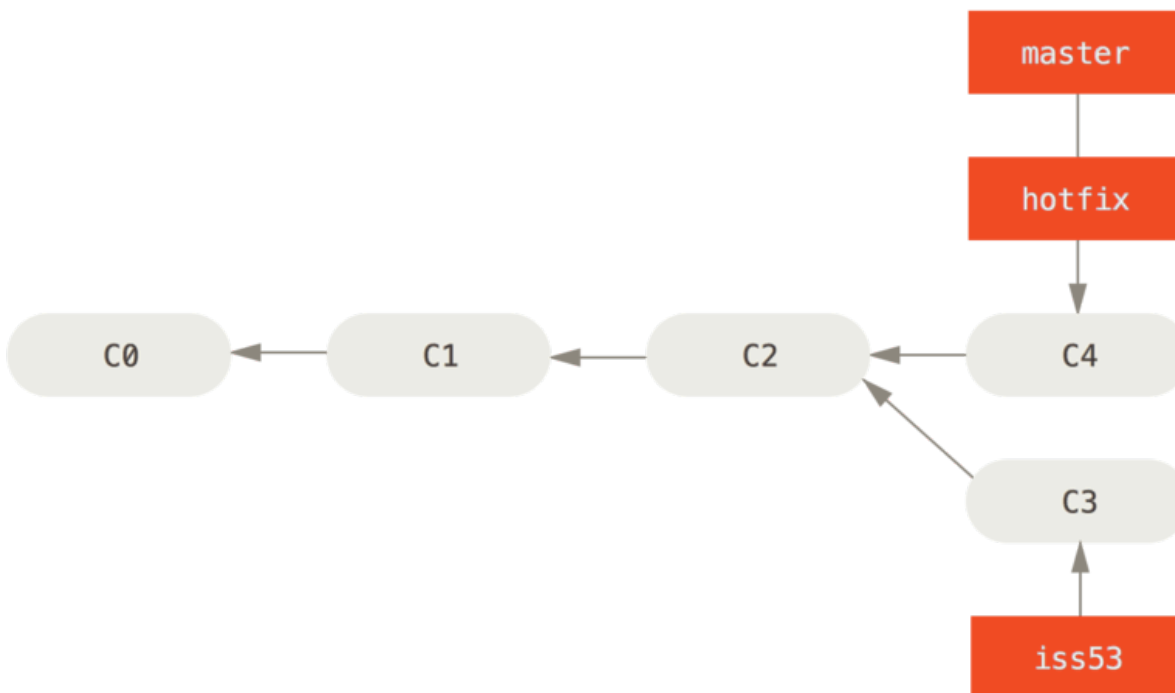


Figure 22. o branch master sofre um "fast-forward" até hotfix

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(+)
```

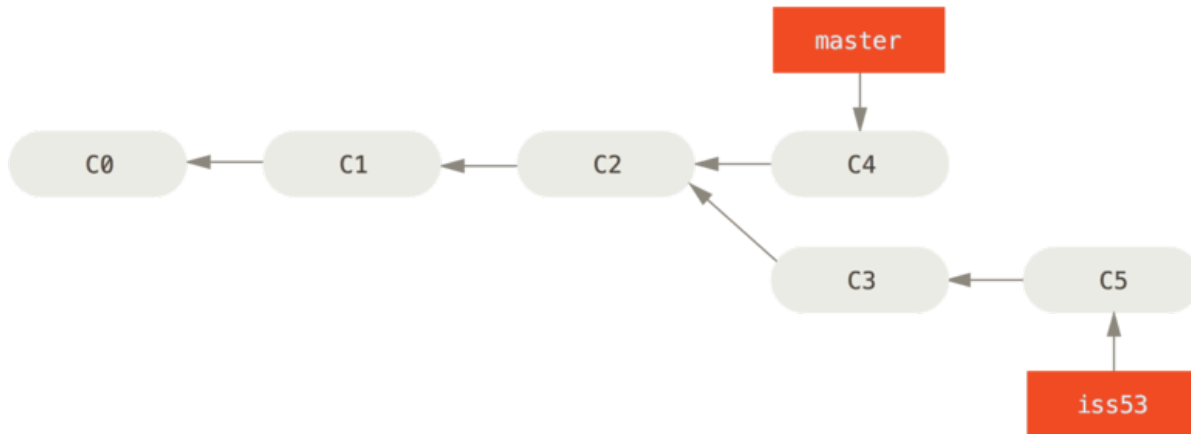


Figure 23. Continuando o trabalho no branch *iss53*

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.

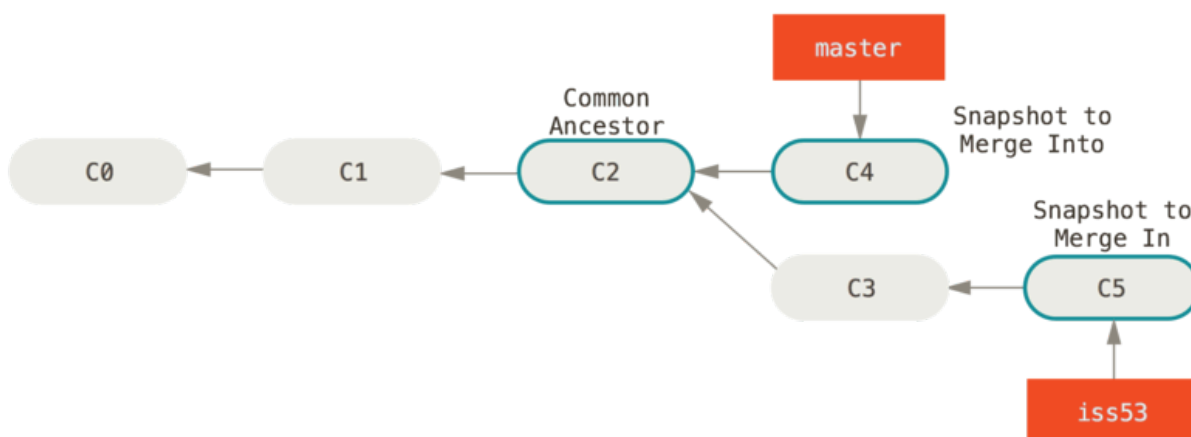


Figure 24. Três snapshots usados em um merge típico

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.

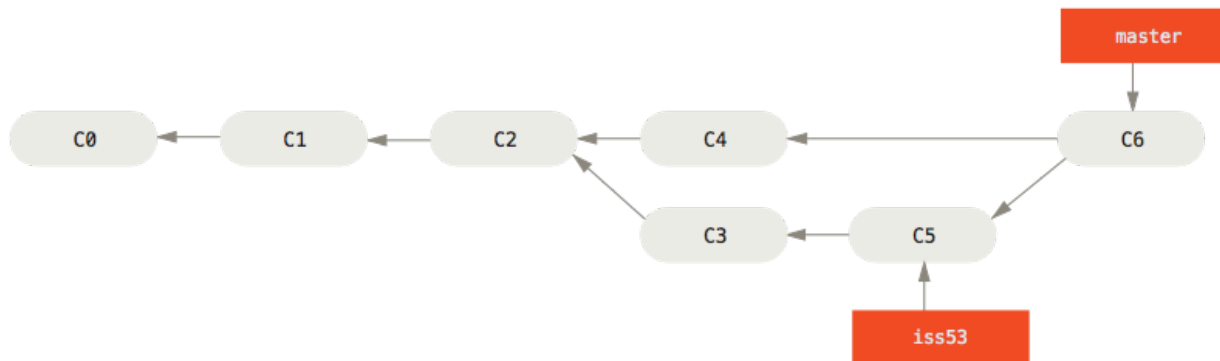


Figure 25. Um commit de merge

Agora que seu trabalho foi integrado, você não precisa mais do branch `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 abri-los manualmente e solucionar os conflitos. O seu arquivo contém uma seção que se parece com isso:

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

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 ferramenta 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 diffuze 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 ferramenta 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](#)

Patches, suggestions, and comments are welcome.
Git is a member of [Software Freedom Conservancy](#)

