PEM - Git

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Henrique do Nascimento Cunha, MSc.

PEM - Git

- Para que serve um sistema de controle de versões?
- Quando estamos trabalhando em um projeto, é comum cometermos erros. Principalmente se o trabalho é em equipe.
- Independente da origem do erro, muitas vezes faz sentido voltar o trabalho a um estado anterior, em que as coisas “funcionavam”
- Além disso, imagine que à medida em que o projeto avança você possa ter uma medida de progresso do projeto no tempo:
  - Quantas linhas de código foram escritas
  - Quantos arquivos gerados
  - Quantas funcionalidades foram implementadas
  - Quais as diferenças entre uma **versão** e outra

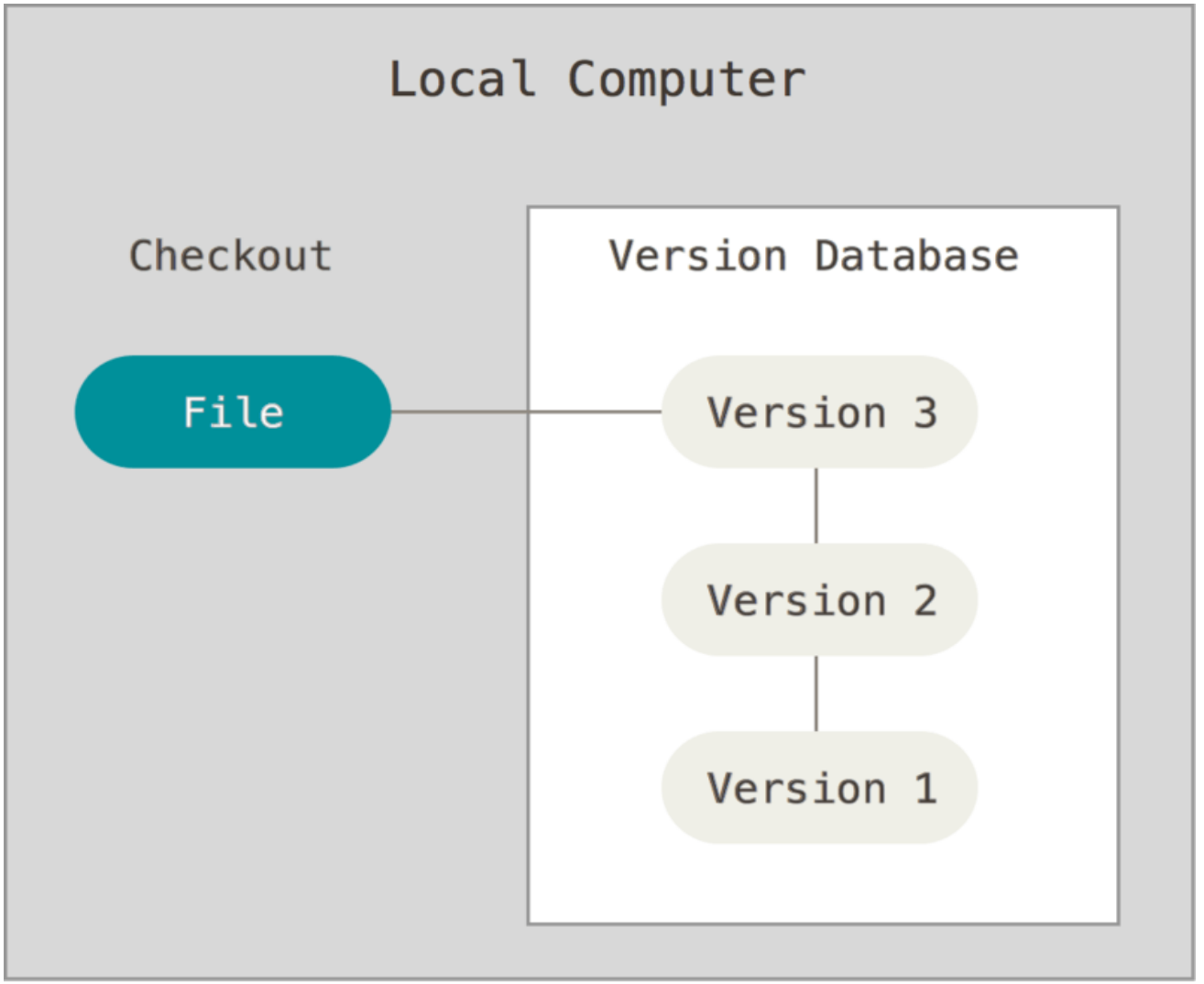
PEM - Git

- Sistemas de Controle de Versão (VCS – Version Control Systems) de uso local
  - RCS
- Como se faz controle de versões sem um VCS?
- Muita gente copia uma versão para outra pasta ou renomeia os arquivos em que estão trabalhando
  - ProjetoNovo
  - ProjetoNovoDeVerdade
  - OutroNovoDeOntem
  - EsseVai
  - FulanoUsaEsseAqui

PEM - Git

- Essa abordagem pode funcionar para sistemas minúsculos
- Mesmo assim é muito susceptível a erros
- Para lidar com isso, programadores desenvolveram VCSs que matinham uma base de dados em que todos os arquivos ficavam sob revisão

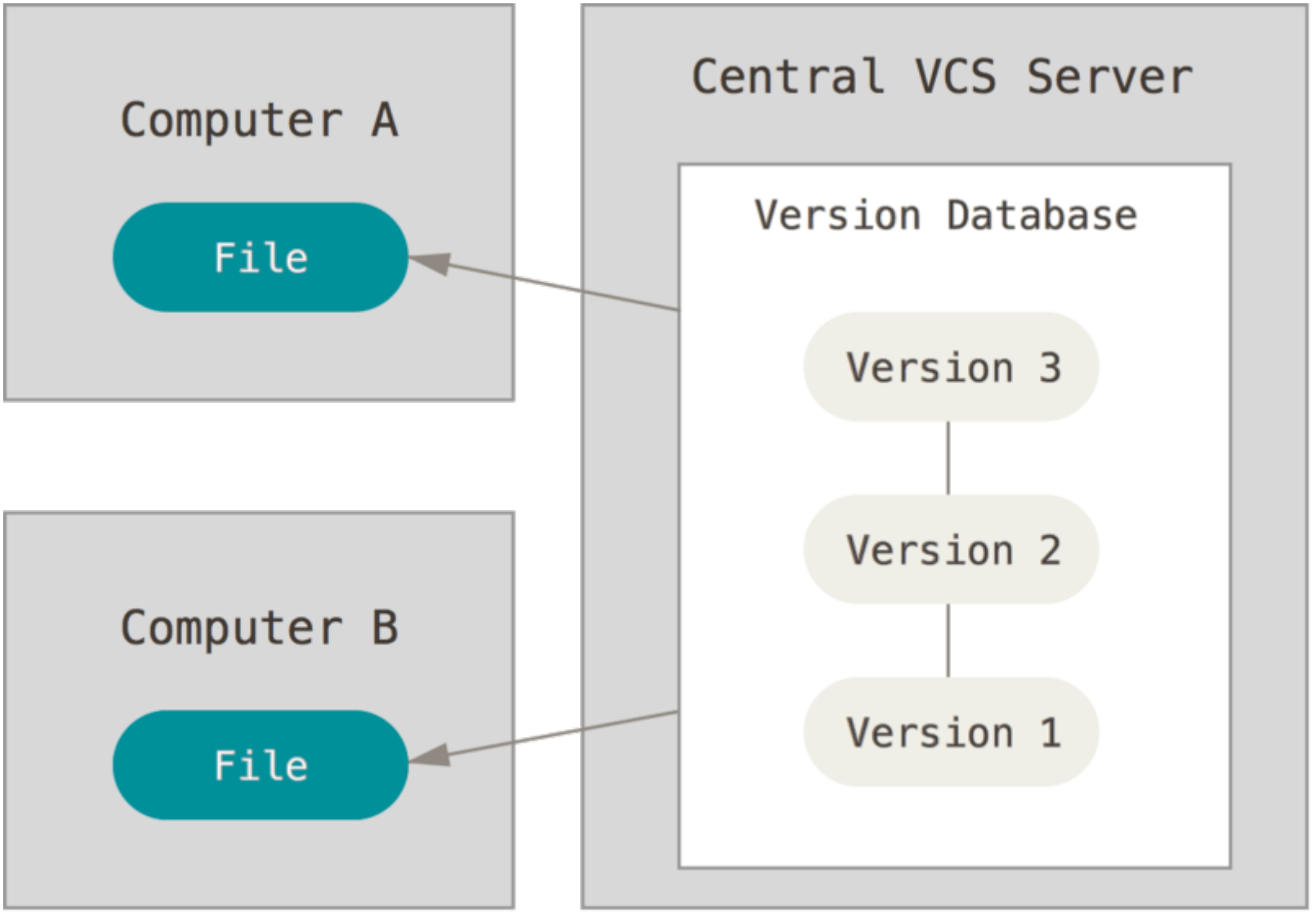
PEM - Git


Local Computer
Checkout
Version Database
File
Version 3
Version 2
Version 1

PEM - Git

- Os VCSs tinham um problema de não conseguir lidar com trabalho colaborativo
- Como resolver isso?
- Sistemas de Controle de Versão Centralizados (CVCS – Centralized Version Control Systems)
  - CVS, Subversion, Perforce

PEM - Git


Computer A
File
Computer B
File
Central VCS Server
Version Database
Version 3
Version 2
Version 1

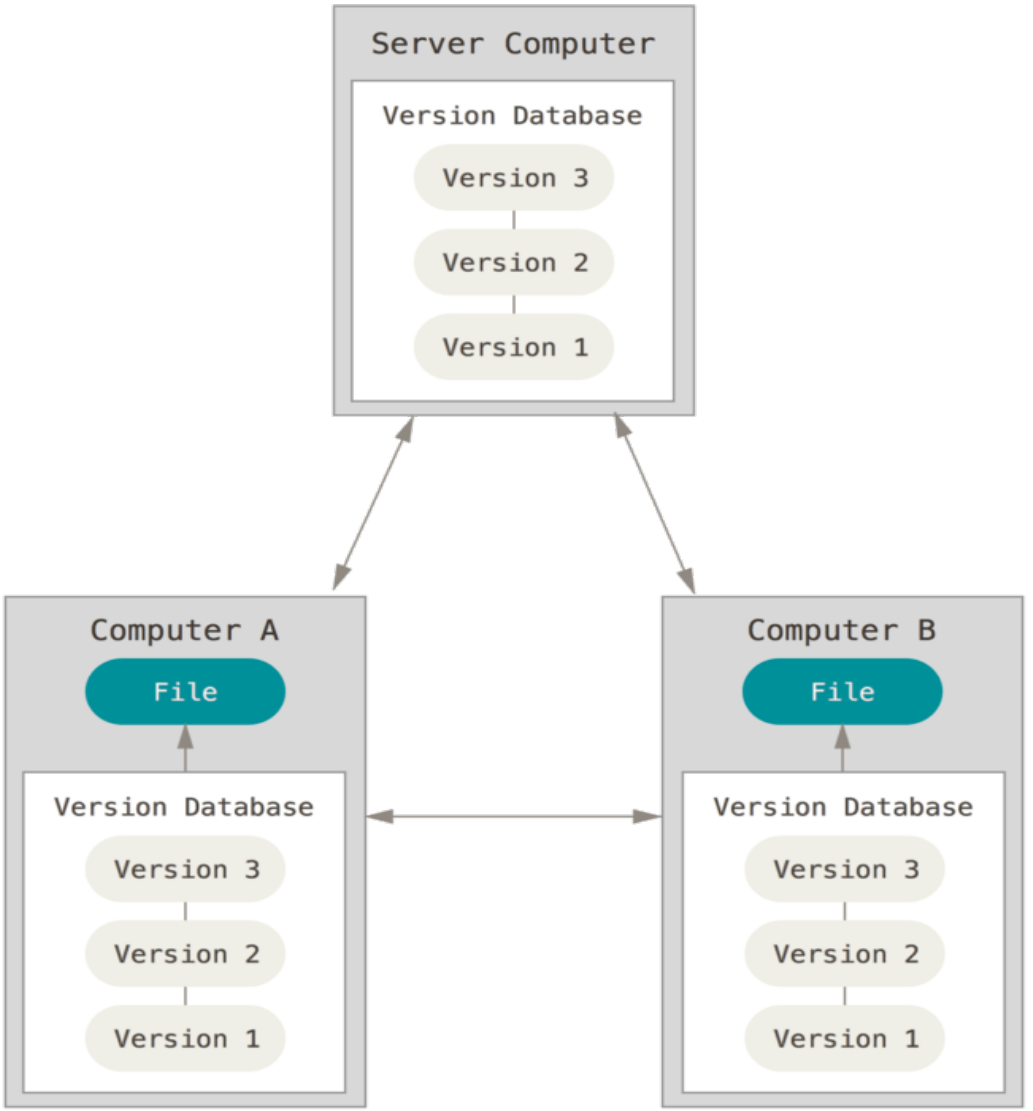
PEM - Git

- CVCSs também tem seus problemas:
  - A própria centralização da base de dados é um ponto de falha
    - O que acontece se o servidor cai?
      - Não é mais possível enviar versões do seu programa para o repositório
    - O que acontece se o servidor explodir?
      - Vai sobrar apenas as cópias nos computadores dos colaboradores
      - Nem sempre a versão mais nova
      - O histórico é todo perdido

PEM - Git

- Sistemas de Controle de Versão Distribuídos (DVCSs - *Distributed Version Control Systems*)
  - Git, Mercurial
- Cada cliente não pega apenas a última versão do repositório
- Cada cliente faz uma cópia completa de todo o repositório
- Se o servidor explode, basta pegar uma das cópias e colocar o servidor de volta no ar

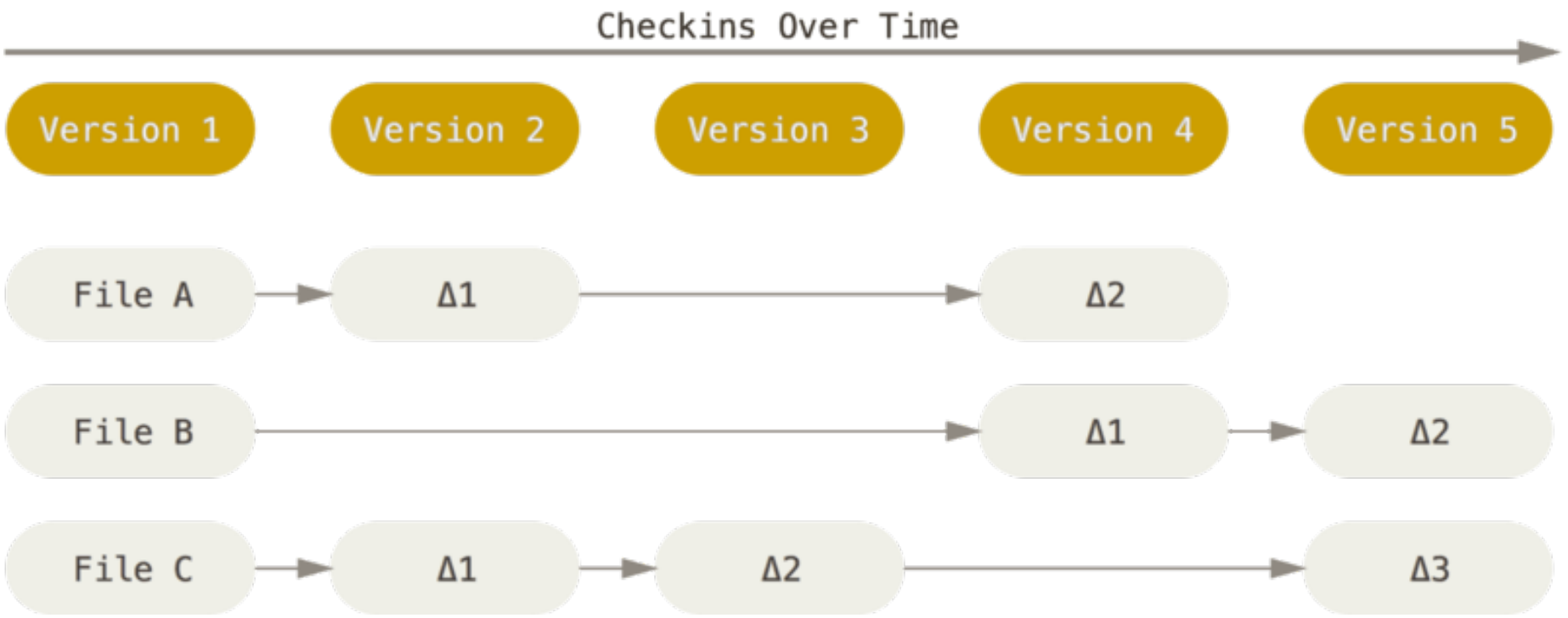
PEM - Git


Server Computer
Version Database
Version 3
Version 2
Version 1
Computer A
File
Version Database
Version 3
Version 2
Version 1
Computer B
File
Version Database
Version 3
Version 2
Version 1

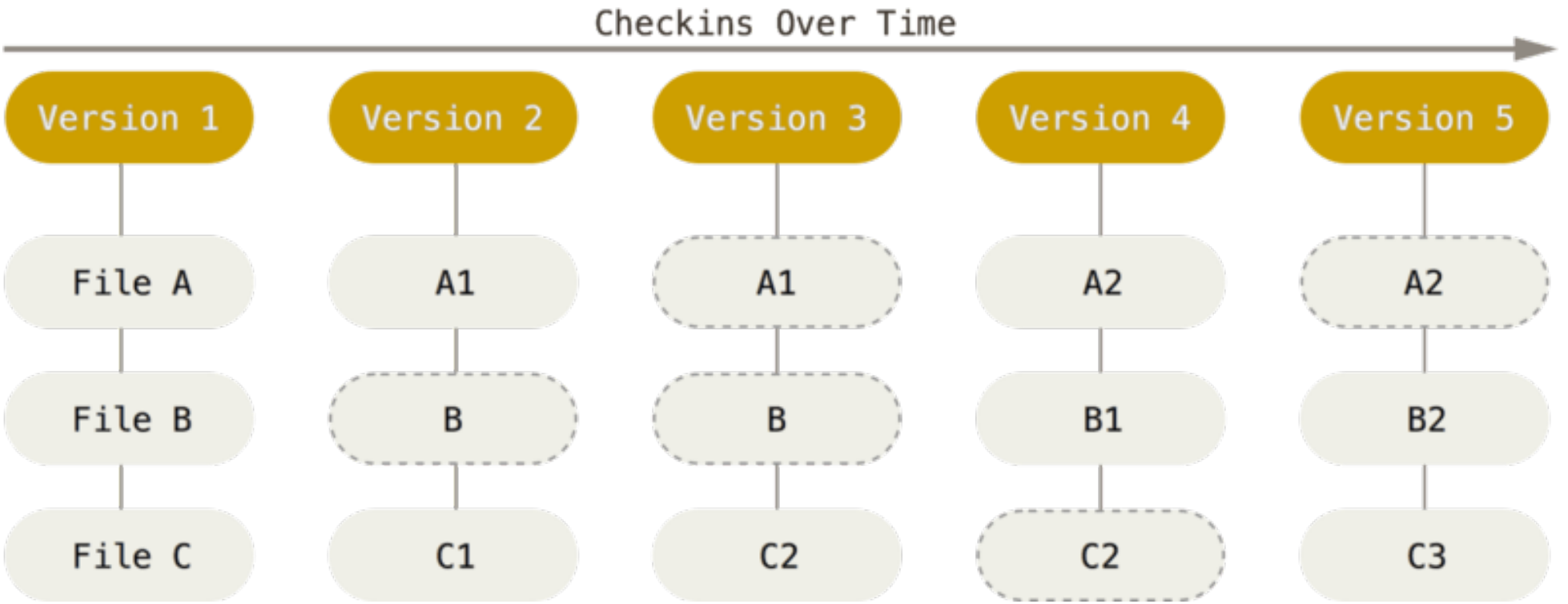
PEM - Git

- Git
  - Veloz
  - Simples
  - Grande suporte a desenvolvimento não linear
  - Distribuído
  - Adequado a qualquer tamanho de projeto
    - O Kernel do linux é desenvolvido usando Git desde 2005

PEM - Git


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
Δ1
Δ2
File B
Δ1
Δ2
File C
Δ1
Δ2
Δ3

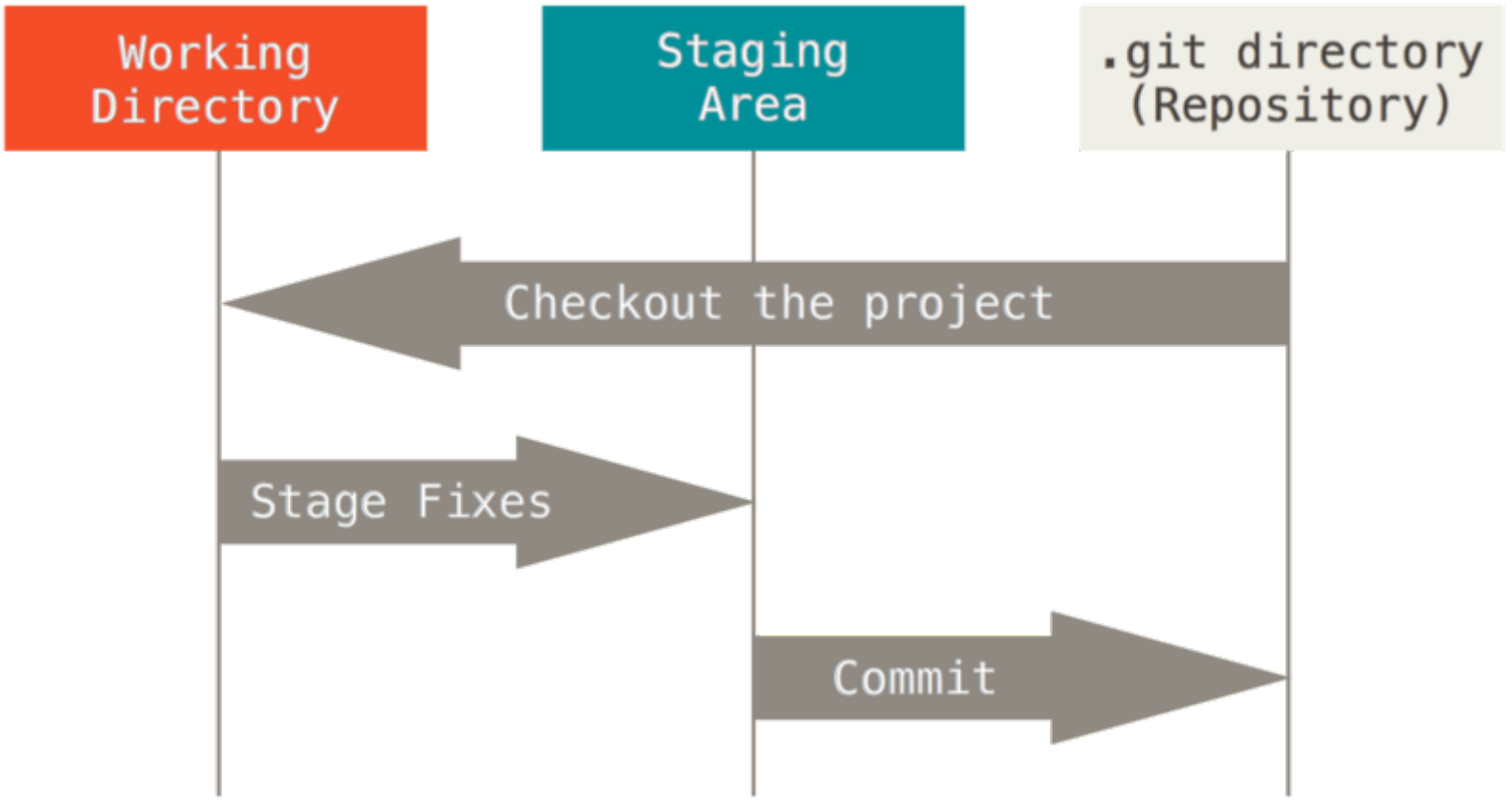
PEM - Git


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
A1
A1
A2
A2
File B
B
B
B1
B2
File C
C1
C2
C2
C3

PEM - Git

- Git
  - Quase toda operação é local
    - De vez em quando você deve fazer upload da sua cópia em um servidor central
    - Em outros sistemas (como Subversion) toda operação em um servidor centralizado necessita de acesso à rede o que tipicamente gera um atraso
  - Muito difícil perder dados
    - Principalmente se você os coloca em um servidor central de tempos em tempos

PEM - Git


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

PEM - Git

- Setando sua identidade

```
$ git config --global user.name
"Henrique Cunha"

$ git config --global user.email
henrique.cunha@gmail.com
```

- Isso vai ajustar sua identidade para todos os projetos. Se em algum projeto você quiser uma identidade diferente, basta rodar o comando novamente sem a opção --global

PEM - Git

- Setando seu editor de texto

```
$ git config --global core.editor
nano
```

- Esse é o editor de texto que será usado para escrever os comentários de cada revisão
- Para verificar suas configurações:

```
$ git config --list
```

PEM - Git

- Como pegar uma cópia de um repositório?

```
$ git clone [url]
```

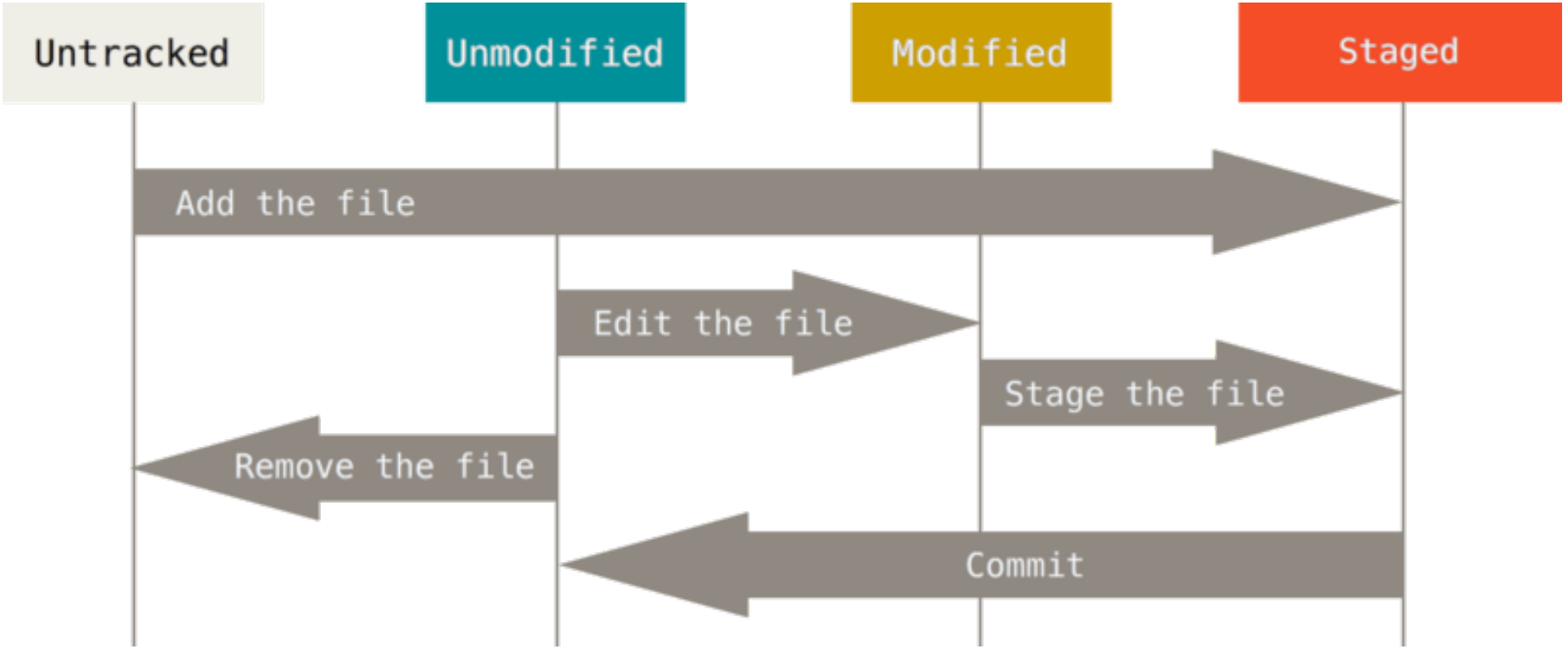
  - A `[url]` pode ser um caminho para um diretório local do seu computador ou para um computador remoto
  - Exemplo:

    git clone https://github.com/libgit2/libgit2

    git clone https://github.com/libgit2/libgit2 meulibgit

PEM - Git

- Fazendo modificações no repositório



PEM - Git

```
$ git status

On branch master

Your branch is up-to-date with
'origin/master'.

nothing to commit, working directory
clean

$
```

PEM - Git

```
$ git status
On branch master
Your branch is up-to-date with
'origin/master'.

nothing to commit, working directory
clean
$
```

Qual branch do projeto estamos

PEM - Git

```
$ git status
On branch master
Your branch is up-to-date with
'origin/master'.
nothing to commit, working directory
clean
$
```

Está no mesmo estado que o mesmo branch no servidor

PEM - Git

- Arquivos que estão no diretório de trabalho não são automaticamente rastreados (*tracked*)
- Você precisa dizer explicitamente ao Git que você quer que eles façam parte do repositório

```
$ git add [arquivo]
```

PEM - Git

- Indexando arquivos (*Staging*)
- Há momentos em que não queremos que seja feito *commit* em um arquivo já rastreado
- Pode ser que as modificações que foram feitas nele ainda não estejam indexadas
- Como Ver se um arquivo foi indexadas ou não?

```
$ git status
```

PEM - Git

- Short Status
- Apesar de ser razoavelmente fácil de entender, o `git status` produz muita coisa na saída padrão.
- Podemos diminuir isso com a opção -s
- `git status -s`

PEM - Git

- Como rastrear um arquivo?

```
$ git add [arquivo]
```

- É comum que todos os arquivos modificados sejam indexados. Há um comando para isso.
- Imagine você ter 50 arquivos e ter que consolidá-los 1 por 1! Não é nada produtivo.

PEM - Git

- Ignorando arquivos
- É comum termos arquivo que não queremos que sejam rastreados
- São comumente arquivos de log de simulações ou arquivos que são gerados automaticamente
  - Arquivos .o ou .a em programas em C
  - Arquivos terminados em ~ que são comumente usados por alguns aplicativos como arquivos temporários
- Basta adicionar esses arquivos no arquivo .gitignore na raiz do seu repositório
- Você também pode adicionar padrões de arquivos usando expressões regulares e metacaracteres

PEM - Git

- Às vezes precisamos saber não só quais arquivos mudaram, mas exatamente o que mudou em cada arquivo
- Para isso existe o comando `git diff`
- Ele mostra quais linhas foram adicionadas a um arquivo e quais foram removidas

PEM - Git

- `git commit`
  - Grava DE FATO alterações indexadas (staged)
  - Se for usado em sua forma mais simples, abre o editor de texto escolhido para que você coloque um comentário sobre esse snapshot
  - Podemos usá-lo com a opção -m para adicionar o comentário mais rapidamente. Ex.:

```
$ git commit -m "Versão inicial do projeto."
```

PEM - Git

- `git commit`
  - Você pode passar por cima da fase de indexação (*staging*) coma opção -a. Ex.:

    ```
    $ git commit -a -m “Adicionada funcionalidade de remover personagem.”
    ```

PEM - Git

- Removendo arquivos
- Ao remover um arquivo com o comando rm, isso não retira o rastreamento do arquivo
- Para retirar o arquivo de rastreamento, você deve usar git rm
- Ao realizar o próximo commit, o arquivo não será mais rastreado a partir daquele *snapshot*

PEM - Git

- Outra coisa que pode ser útil é retirar o rastreamento do arquivo sem removê-lo de sua cópia de trabalho
- `git rm –cached [arquivo]`

PEM - Git

- Movendo arquivos
- Renomear um arquivo rastreado faz com que o git pense que ele é um arquivo não rastreado
- Se quiser renomear (ou mover) um arquivo, use o comando `git mv`

PEM - Git

- Histórico de commits

```
$ git log
```

  - Esse comando tem muitas opções que permitem:
    - as diferenças entre vários *snapshots* do repositório
    - formatar a saída do comando de maneira adequada
    - Delimitar a saída por tempo

```
$ git help log
```

PEM - Git

- Desfazendo coisas
  - Se você quiser consertar um commit mal planejado e ainda não tiver feito nenhuma alteração depois desse commit, você pode consertá-lo

```
$ git commit - “commit inicial”
$ git add [arquivo esquecido]
$ git commit --amend
```

PEM - Git

- Desfazendo coisas
  - Como desfazer uma indexação indesejada

    ```
    $ git reset HEAD [arquivo]
    ```

  - Como “desmodificar” um arquivo:

    ```
    $ git checkout --[arquivo]
    ```

PEM - Git

- Tags
  - Em algum ponto, é importante marcar pontos específicos na história do seu projeto como sendo importantes
  - É comum darmos nomes ou números às releases: v1.0, v2.3, etc.
  - Para listar as tags que existem:

```
$ git tag
```

PEM - Git

- Tags
  - Como criar uma tag?

```
$ git tag -a v1.0 -m "minha versão 1.0"
```

  - Para mostrar informações da tag:

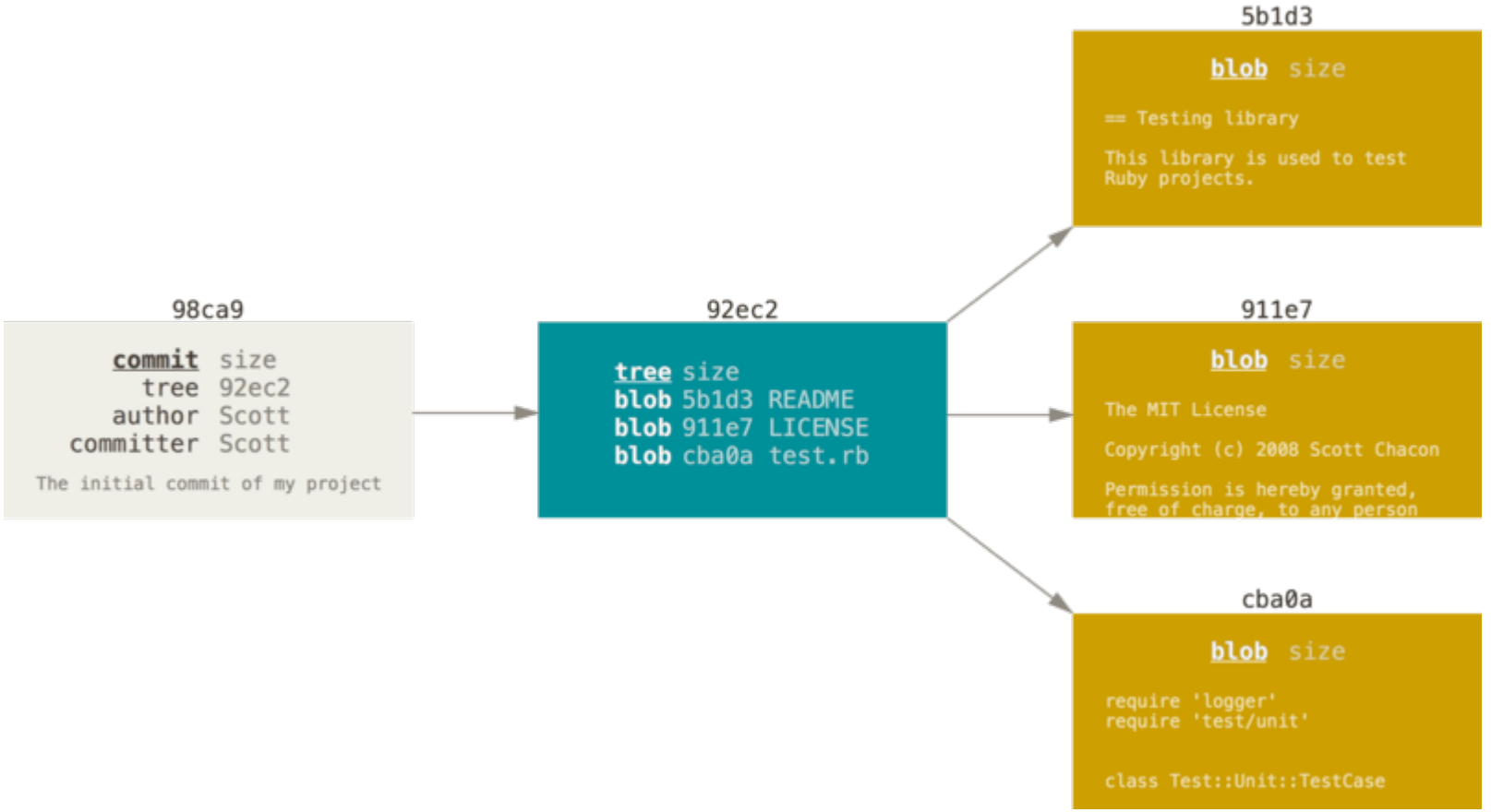
```
$ git show v1.0
```

PEM - Git

- Branches
  - Para colaborar, é comum que tenhamos que separar pedaços do repositório para as equipes poderem trabalhar em funções diferentes do projeto
  - Git faz isso usando *branches* (galhos)

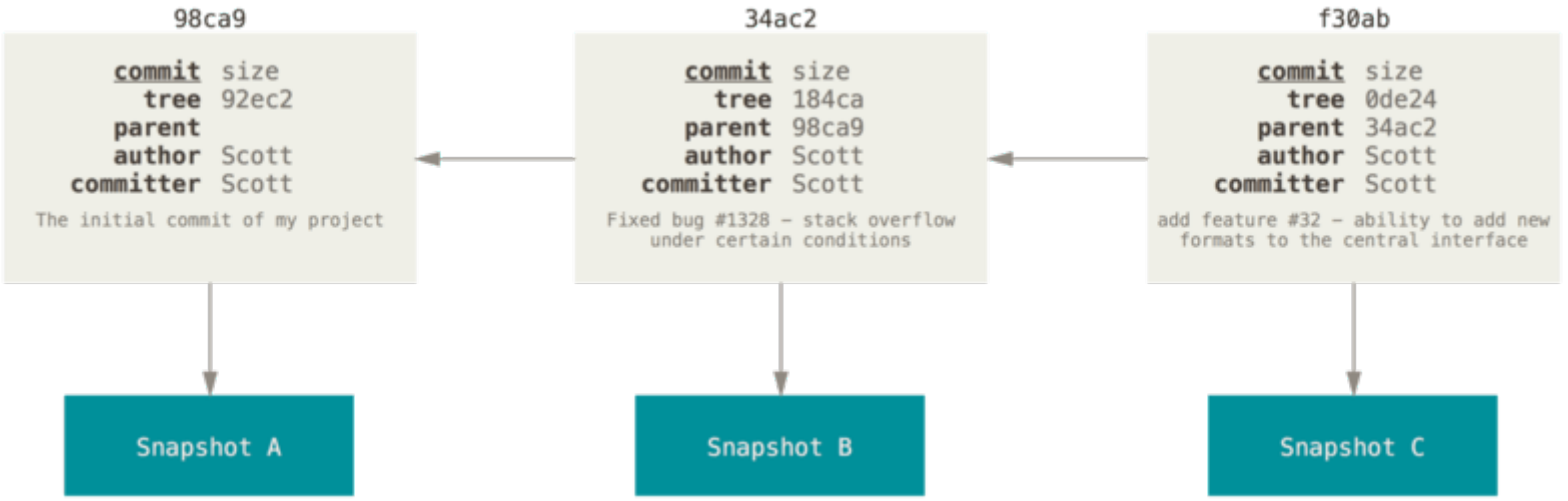
PEM - Git

- Primeiro commit



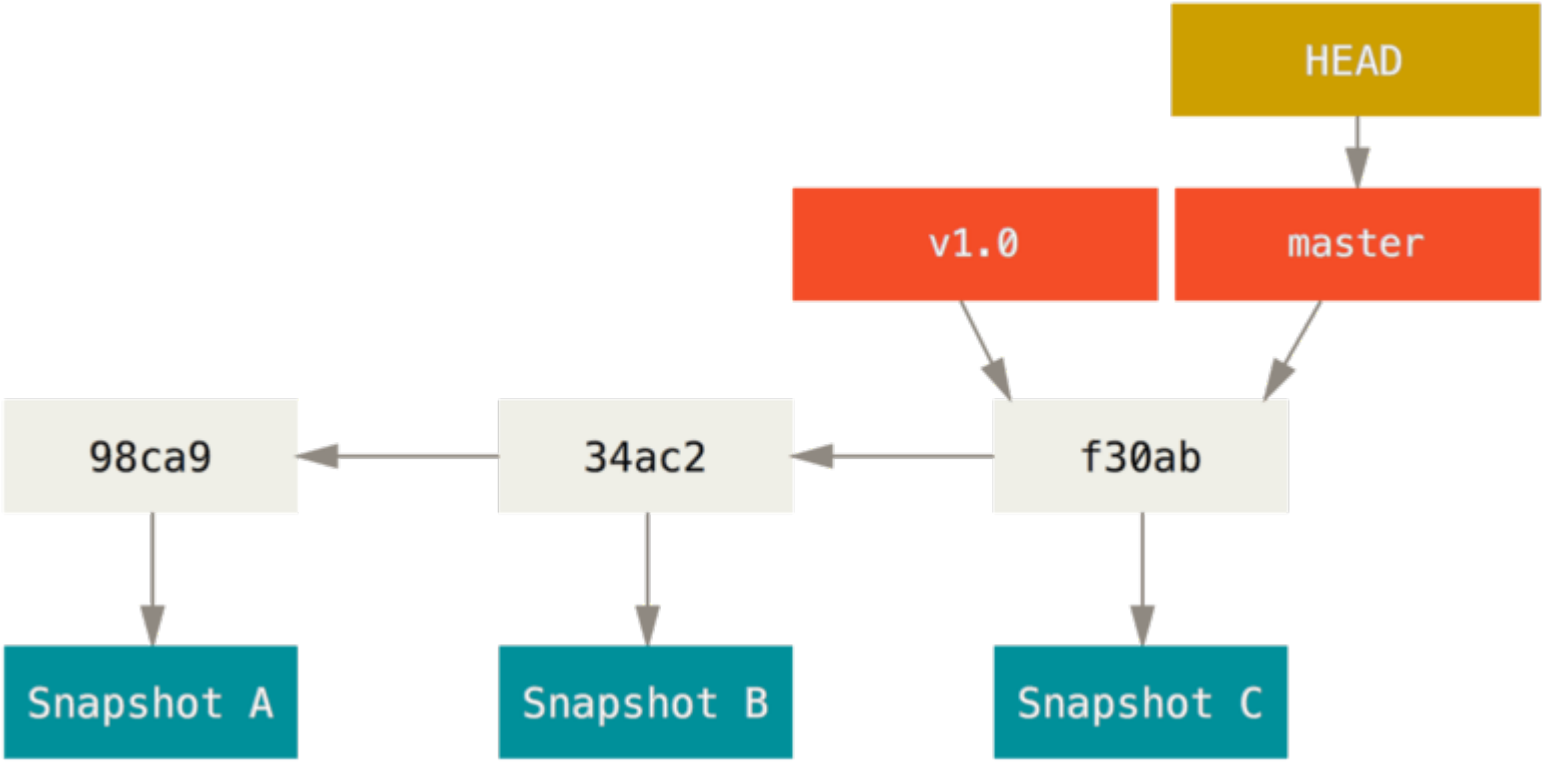
PEM - Git

- Vários commits



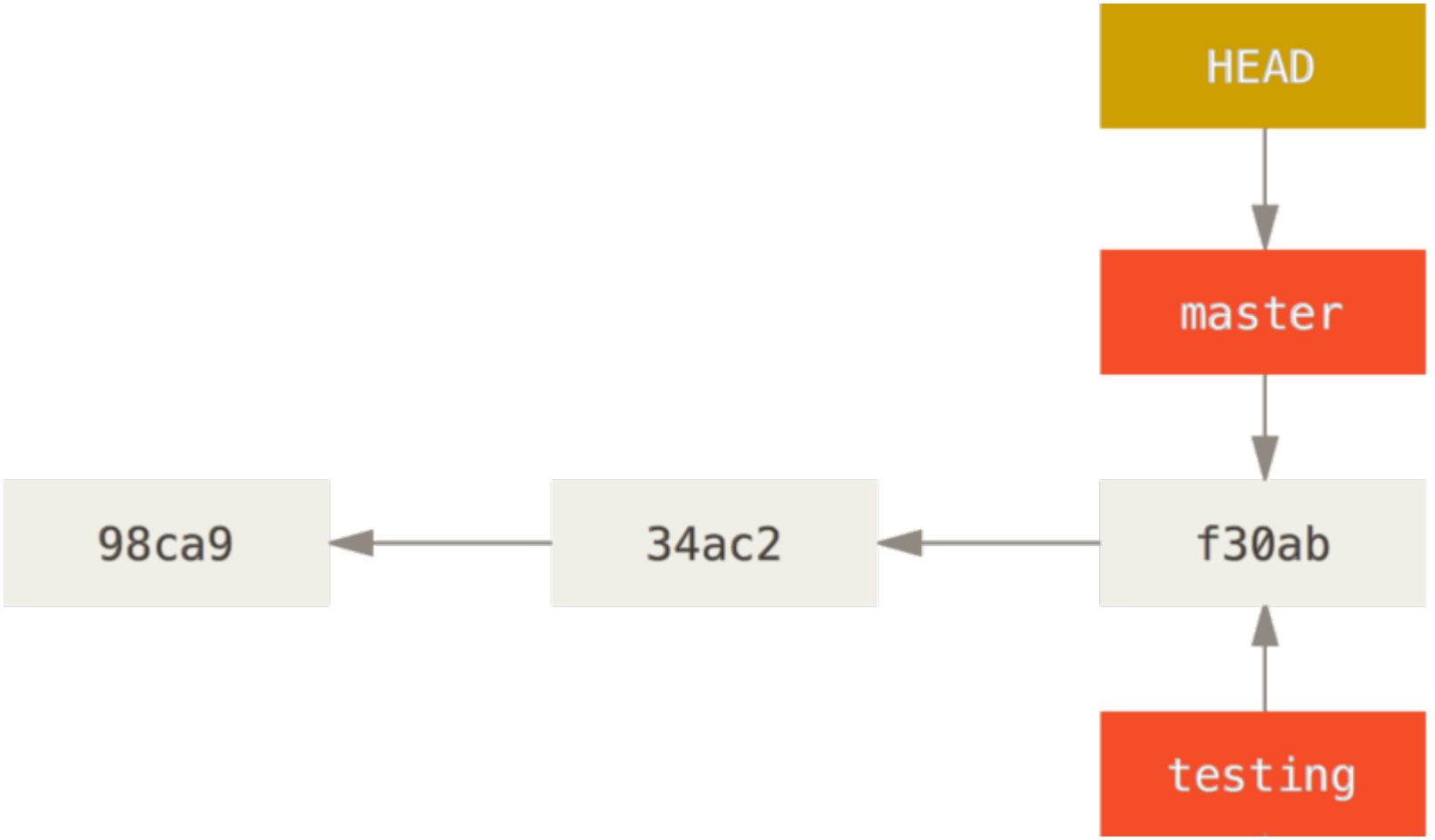
PEM - Git

- Vários commits



PEM - Git

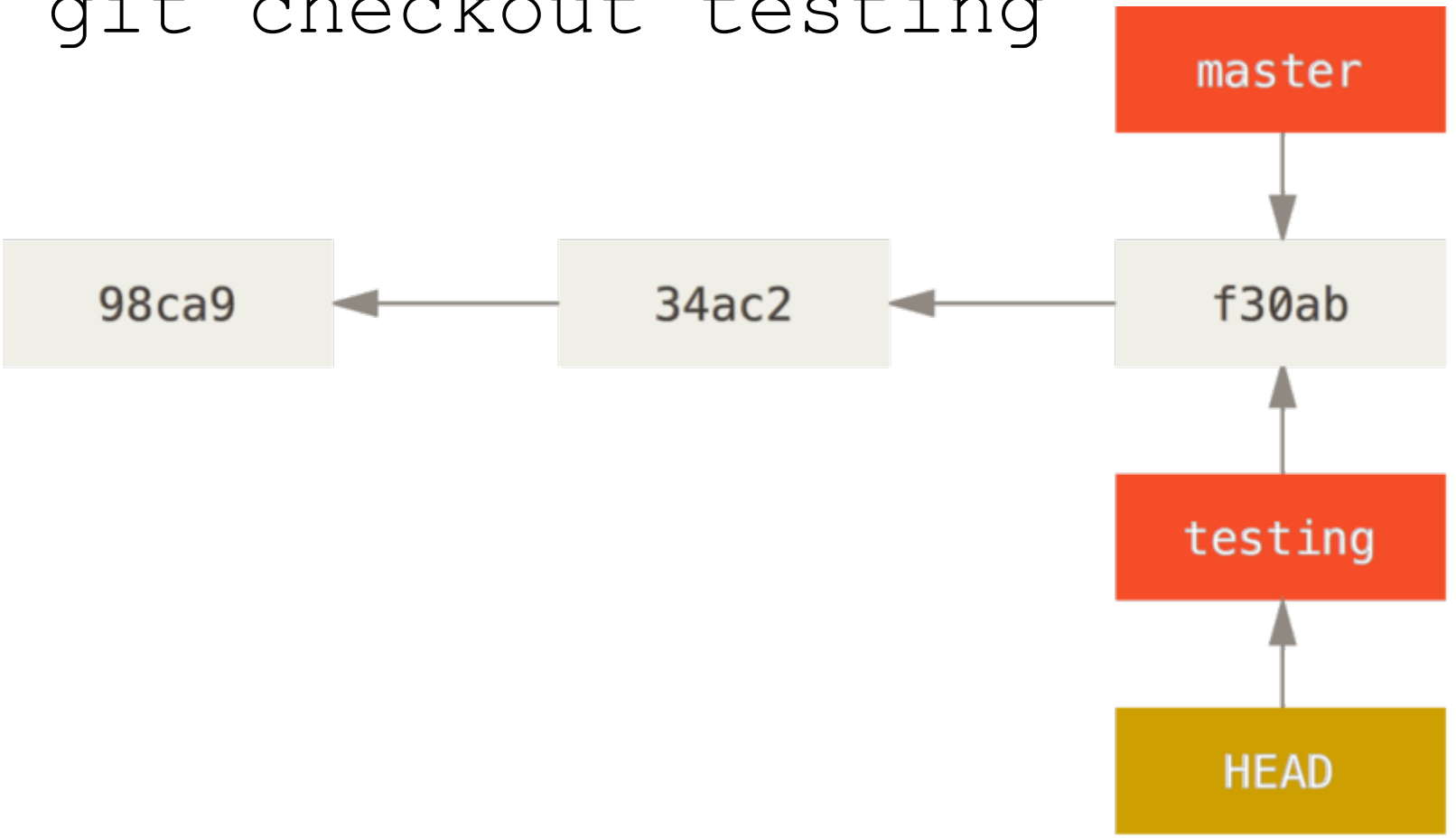
```
$ git branch testing
```



PEM - Git

Como mudar o HEAD para outro branch

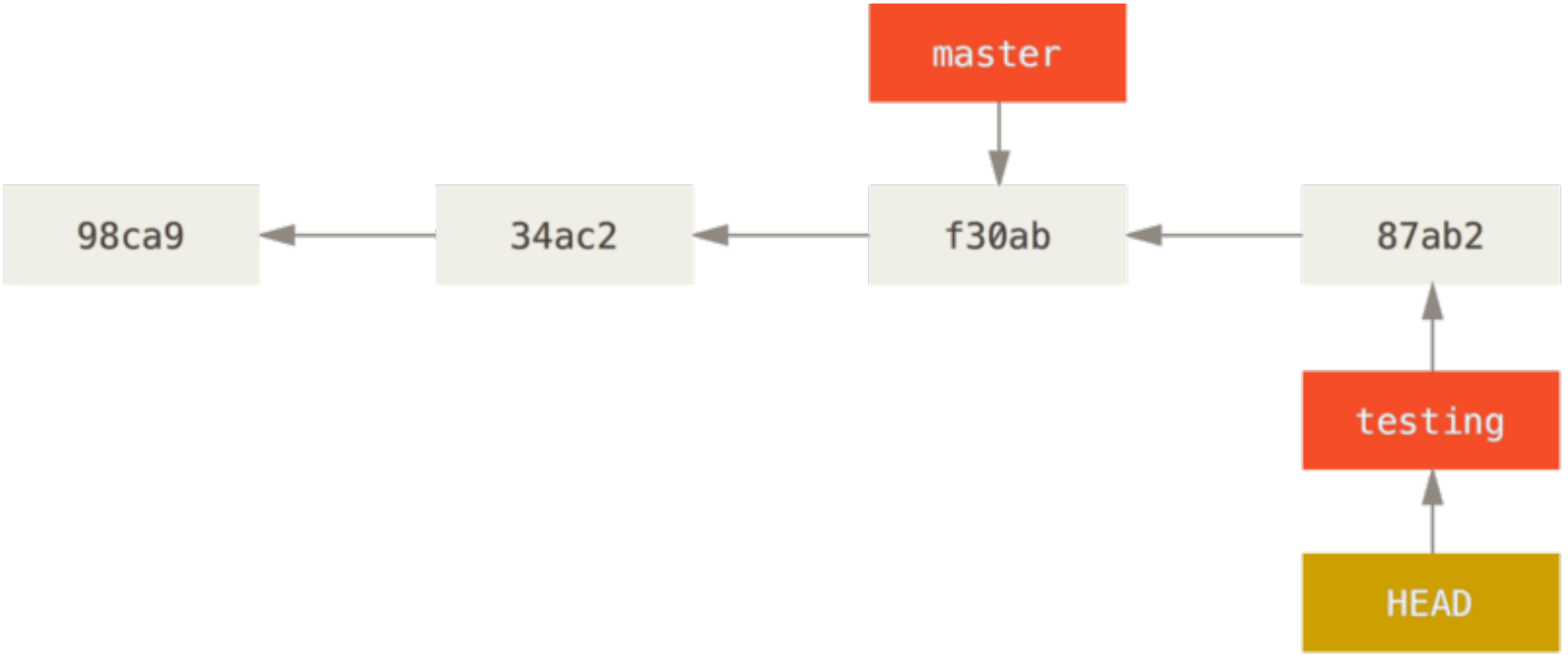
```
$ git checkout testing
```



PEM - Git

O que acontece ao fazer commit em outro branch?

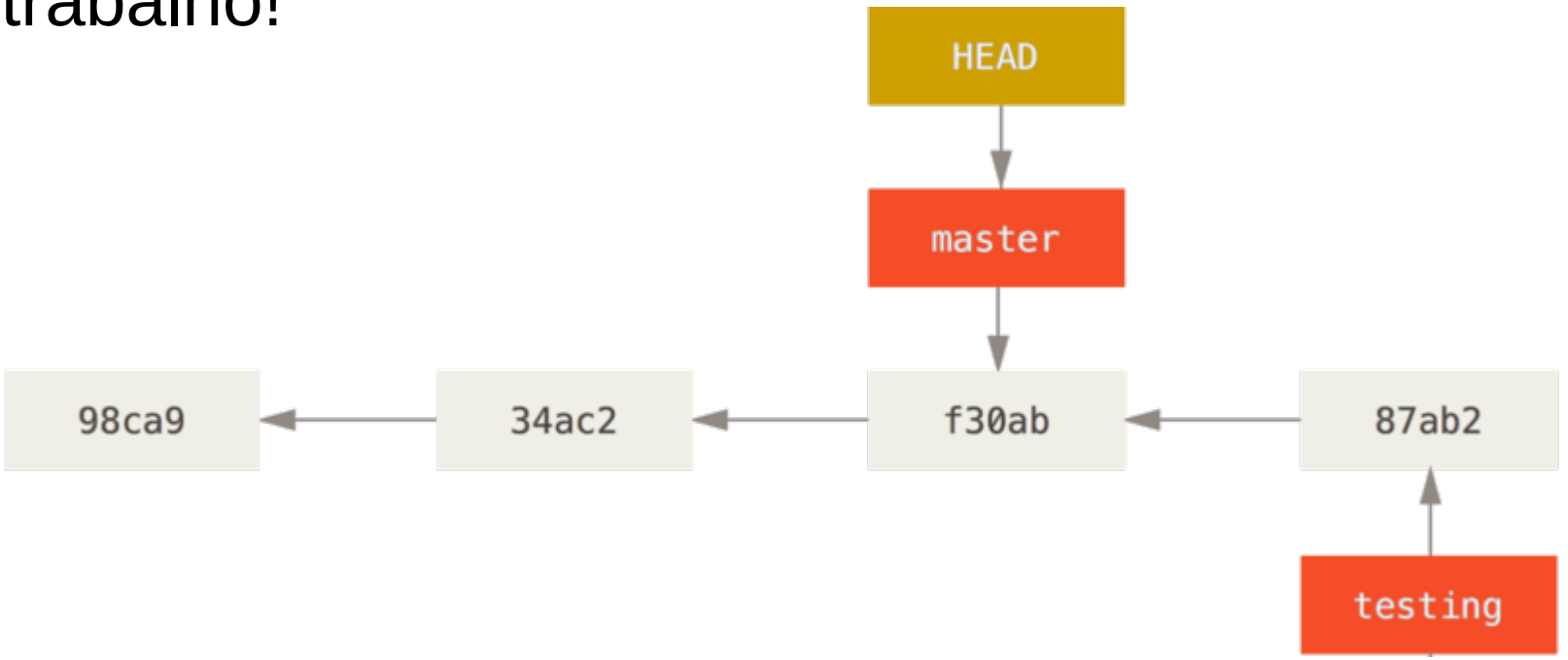
```
$ nano [arquivo]
$ git commit -a -m “mudança”
```



PEM - Git

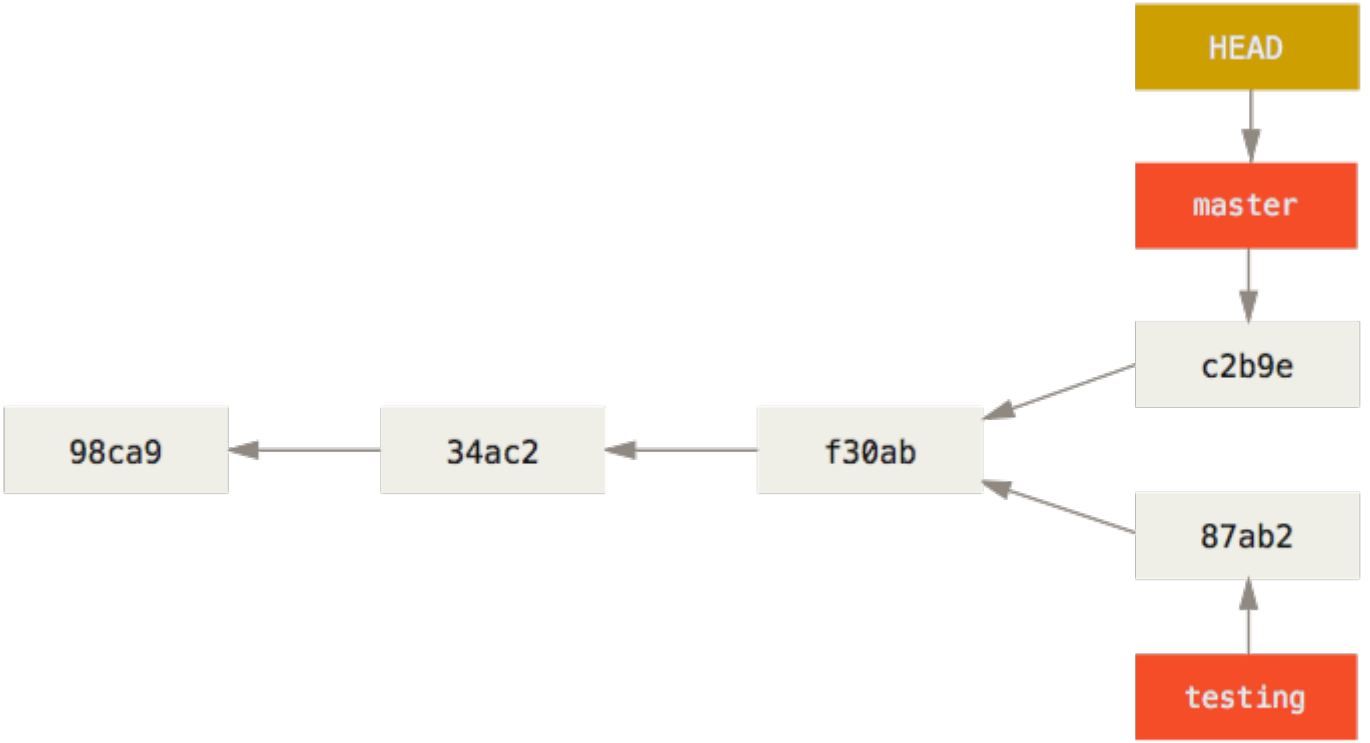
```
$ git checkout master
```

- Esse comando faz modificações no seu diretório de trabalho!



PEM - Git

```
$ nano [arquivo]
$ git commit -m “outras mudançcas”
```

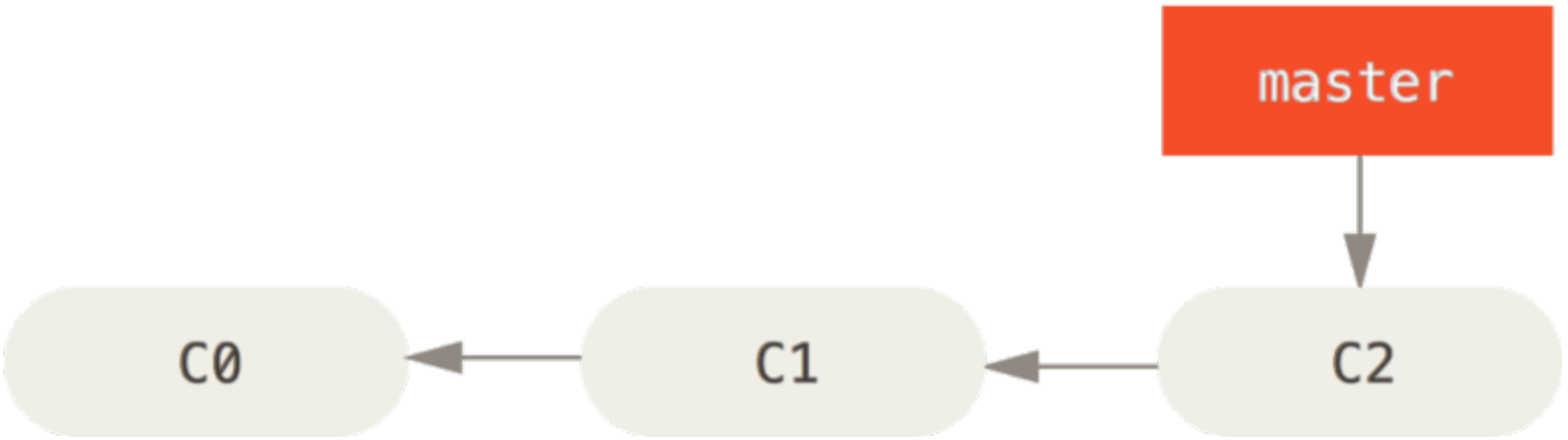


PEM - Git

- Merge
  - Criar *branches* é fácil. Mas pode ser difícil juntar isso tudo depois
  - Vamos mostrar como juntar *branches* com um exemplo:
    - Suponha que estamos criando um site
    - Criaremos um *branch* para resolver um problema específico
    - Em algum momento aparece um outro problema de maior prioridade. Vamos resolvê-lo e depois fazer merge de tudo

PEM - Git

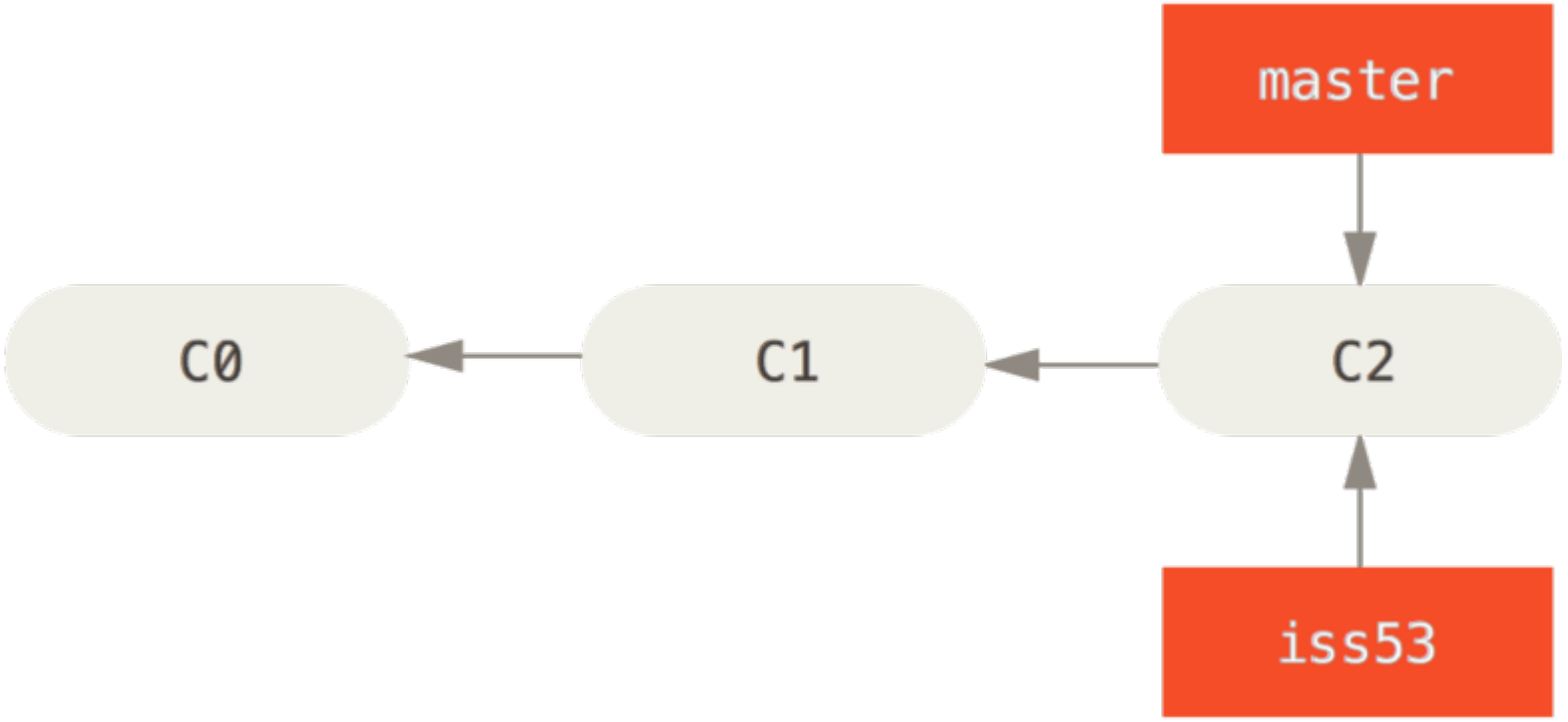
- Projeto com alguns poucos commits



PEM - Git

- Criando outro *branch* e movendo o HEAD para o *branch* recém-criado

  $ git checkout -b iss53



PEM - Git

- Agora você recebe um chamado e precisa fazer um conserto de urgência
- Você não precisa reverter o que foi feito no *branch* iss53. Nem precisa fazer o conserto nesse *branch*.
- Basta você voltar para o branch master:

```
$ git checkout master
```

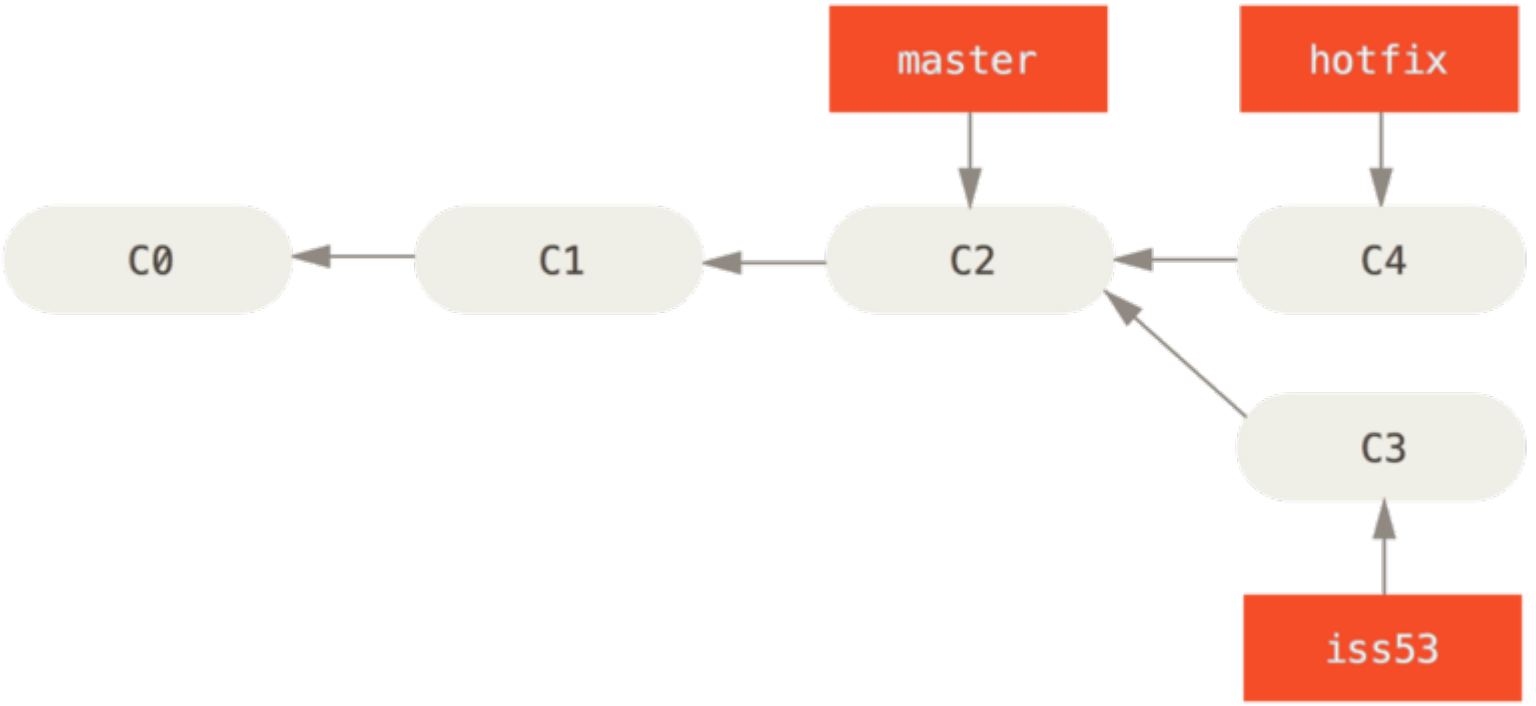
- E depois criar um outro branch para o conserto de urgência:

```
$ git checkout -b hotfix
```

PEM - Git

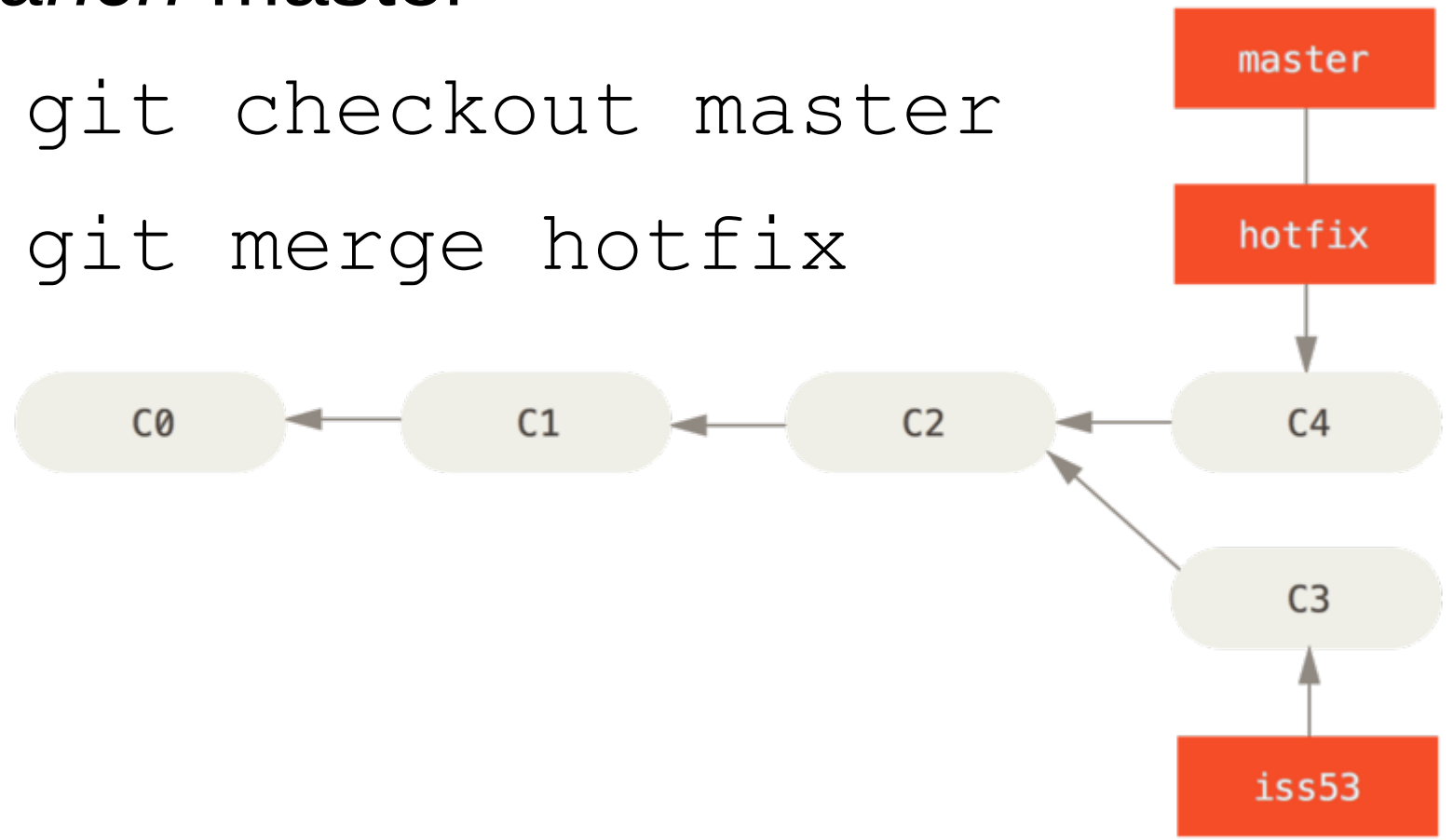
- nano index.html

```
$ git commit -a -m “endereço
para site externo consertado”
```



PEM - Git

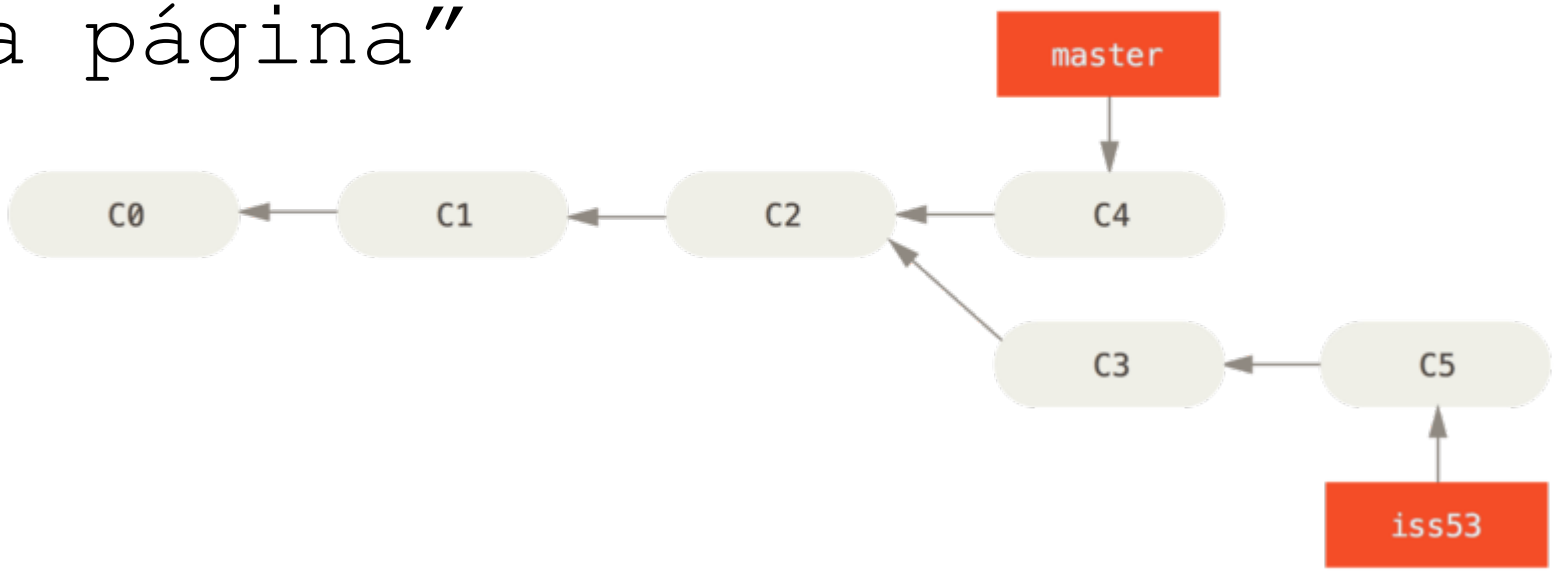
- Agora que o conserto urgente foi completado, podemos fazer o merge com o *branch* master
- `$ git checkout master`
- `$ git merge hotfix`



PEM - Git

- Voltando ao iss53

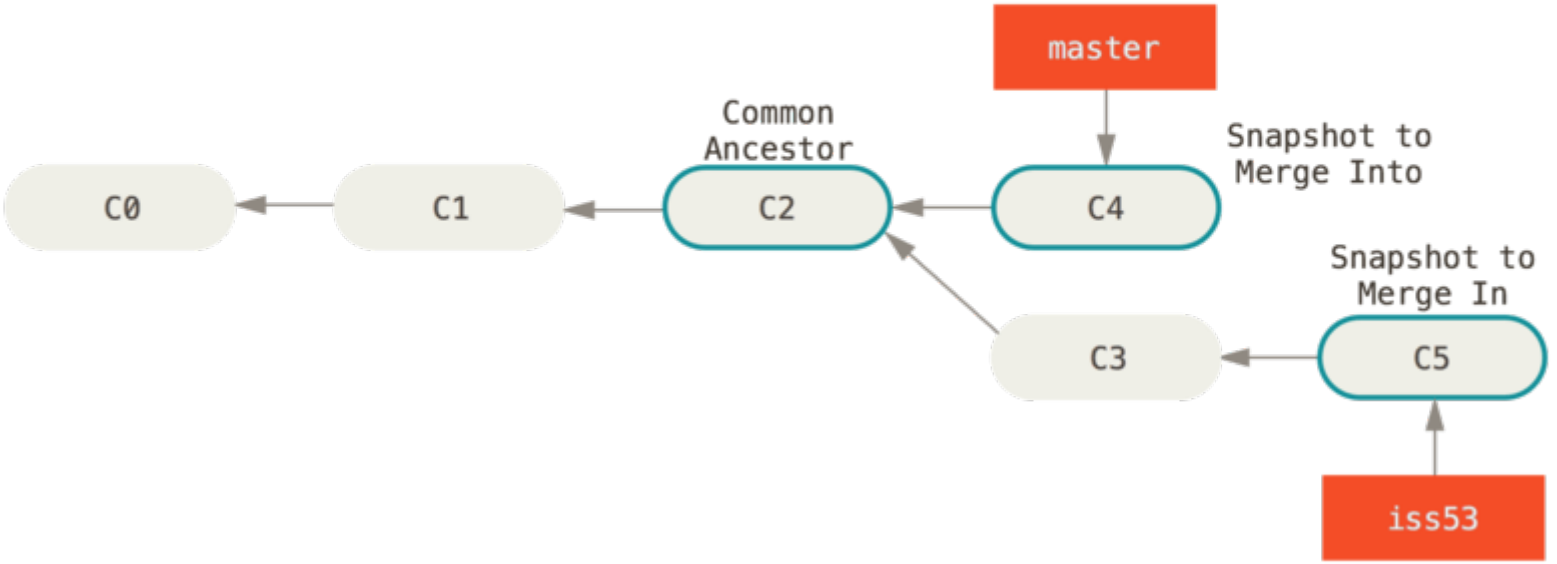
```
$ git checkout iss53

$ nano index.html

$ git commit -a -m “novo rodapé
da página”
```



PEM - Git

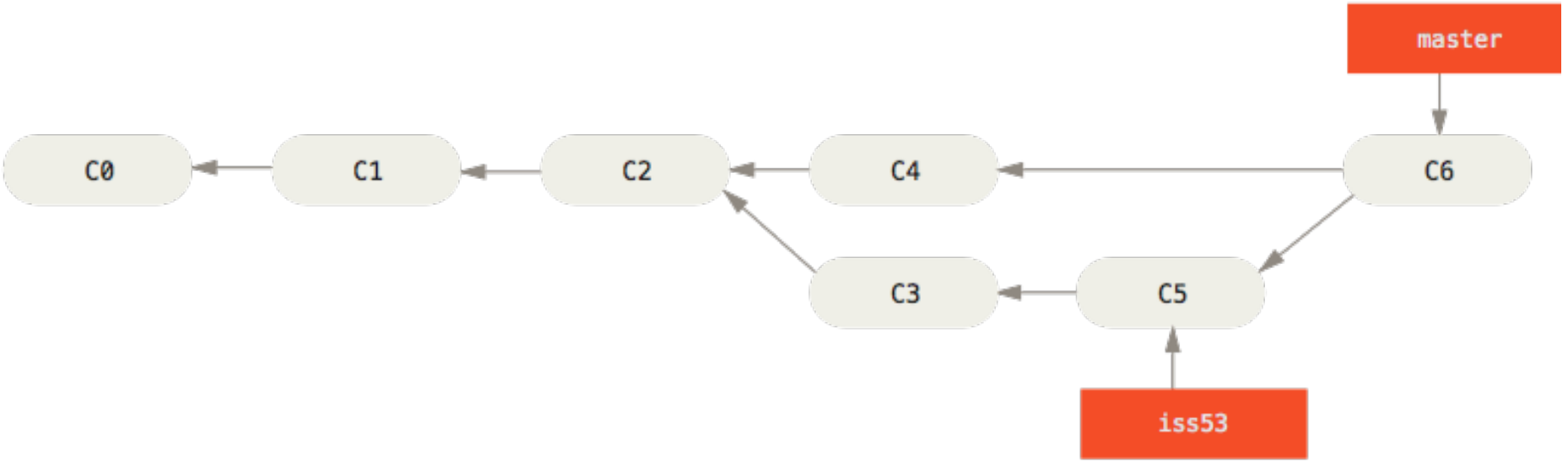
- Fazendo merge entre o iss53 e o *branch* master

```
$ git checkout master
$ git merge iss53
```



PEM - Git

- Se você não mexeu nas mesmas partes do arquivo no branch hotfix e no branch iss53, o que vai acontecer é isso:



- Nesse caso, pode apagar o branch iss53:

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

PEM - Git

- Como lidar com conflitos
  - É comum que o processo de merge não seja tão tranquilo
  - Imagine que você esteja mexendo nas mesmas linhas de código dos mesmos arquivos em *branches* diferentes
  - Quando for fazer merge, como o git vai saber qual versão é a correta?
    - Não tem como!
  - Mas ele te dá recursos para resolver os conflitos

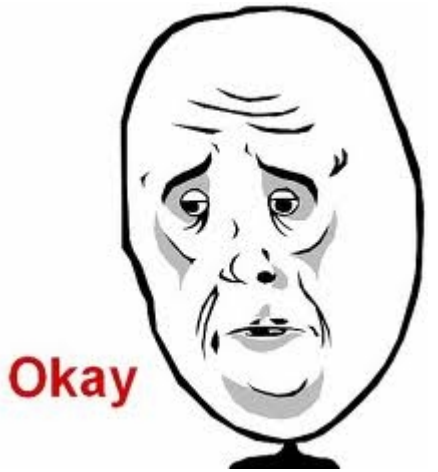
PEM - Git

```
$ git merge iss53
```

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.



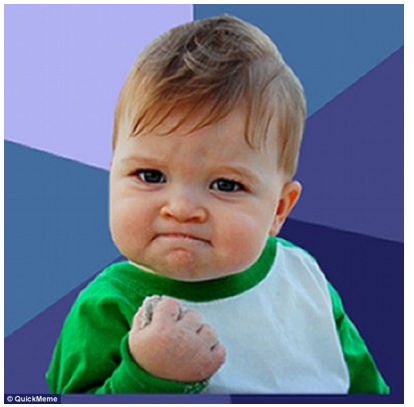
PEM - Git

```
$ 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")
```

PEM - Git

- Como lidar com conflitos
  - Resolvidos os conflitos:

```
$ 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
```



PEM - Git

- Como lidar com conflitos

```
$ git commit -m “conflito entre arquivos resolvidos”
```

  - O git irá prduzir uma mensagem dizendo que o conflito foi resolvido

PEM - Git

- Gerenciamento de *branches:*

```
$ git branch
iss53
* master
testing
```

- Lista todos os branches do repositório e indica em qual você está

PEM - Git

- Gerenciamento de *branches:*

```
git branch -v
iss53   93b412c fix javascript issue
* master 7a98805 Merge branch 'iss53'
testing 782fd34 add fulano to the author list in readmes
```

PEM - Git

- Gerenciamento de *branches:*

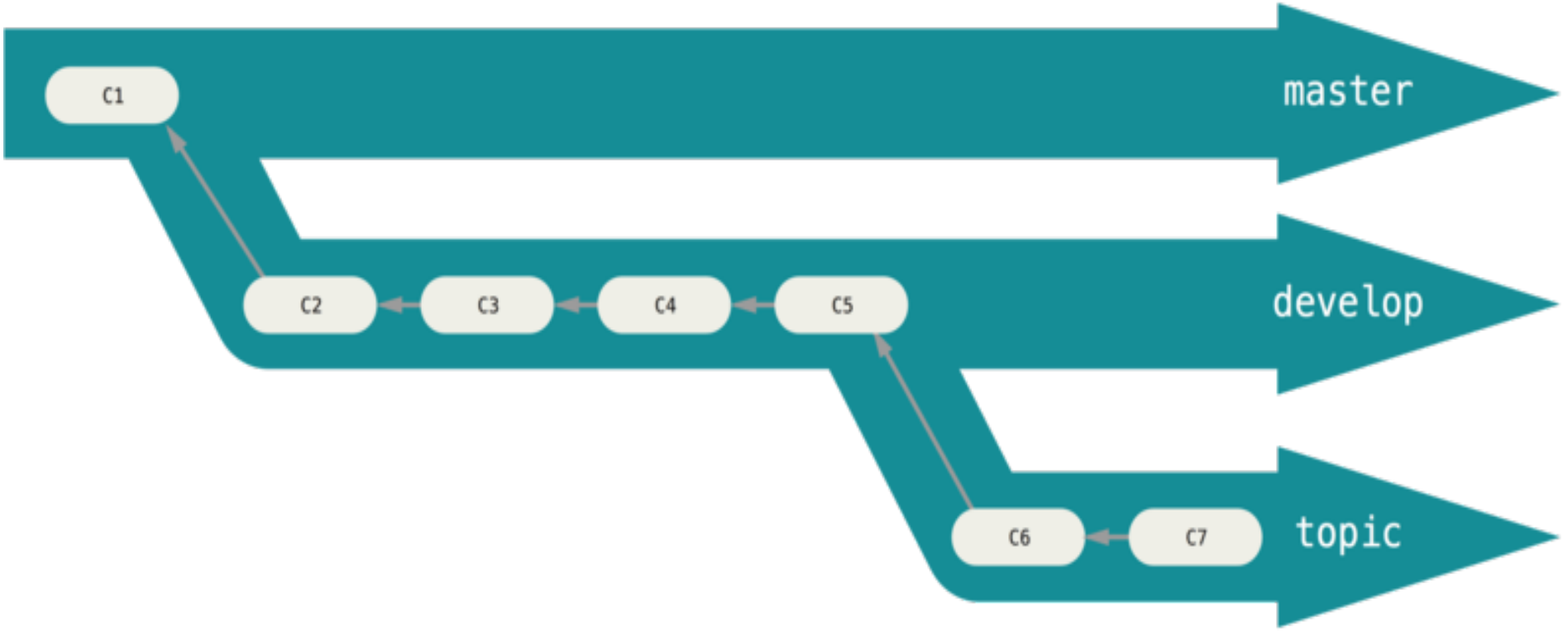
```
$ git branch --merged
iss53
* master
$ git branch --no-merged
testing
```

PEM - Git

- Fluxo de trabalho com branches:
  - *Branches* de longa duração (*long-running branches*)
    - Existe durante um longo tempo dentro do projeto
    - Pode ser feito merge com o *branch master* várias vezes
    - É uma técnica que faz parte do *workflow* de vários usuários de git

PEM - Git

- Fluxo de trabalho com branches:
  - *Branches* de longa duração

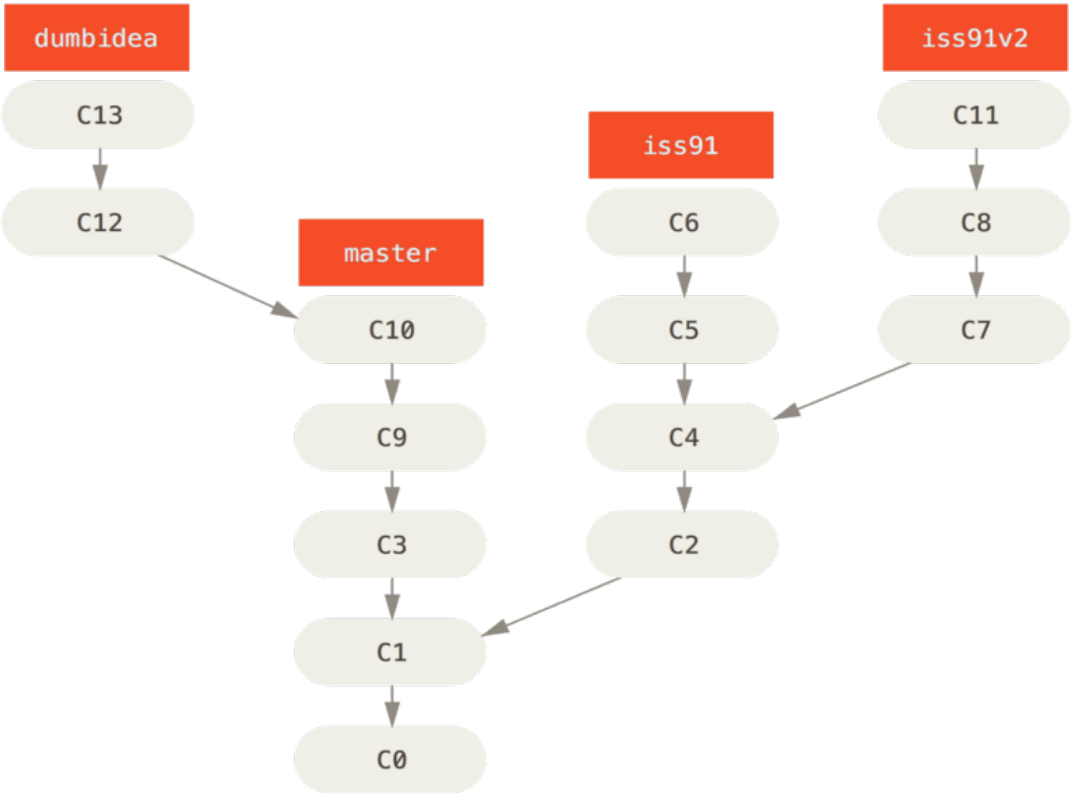


PEM - Git

- Fluxo de trabalho com branches:
  - *Branches* de curta duração (*topic branches*)
    - Existe durante um curto tempo dentro do projeto
    - O *merge* é feito tipicamente uma vez e depois o *branch* é apagado

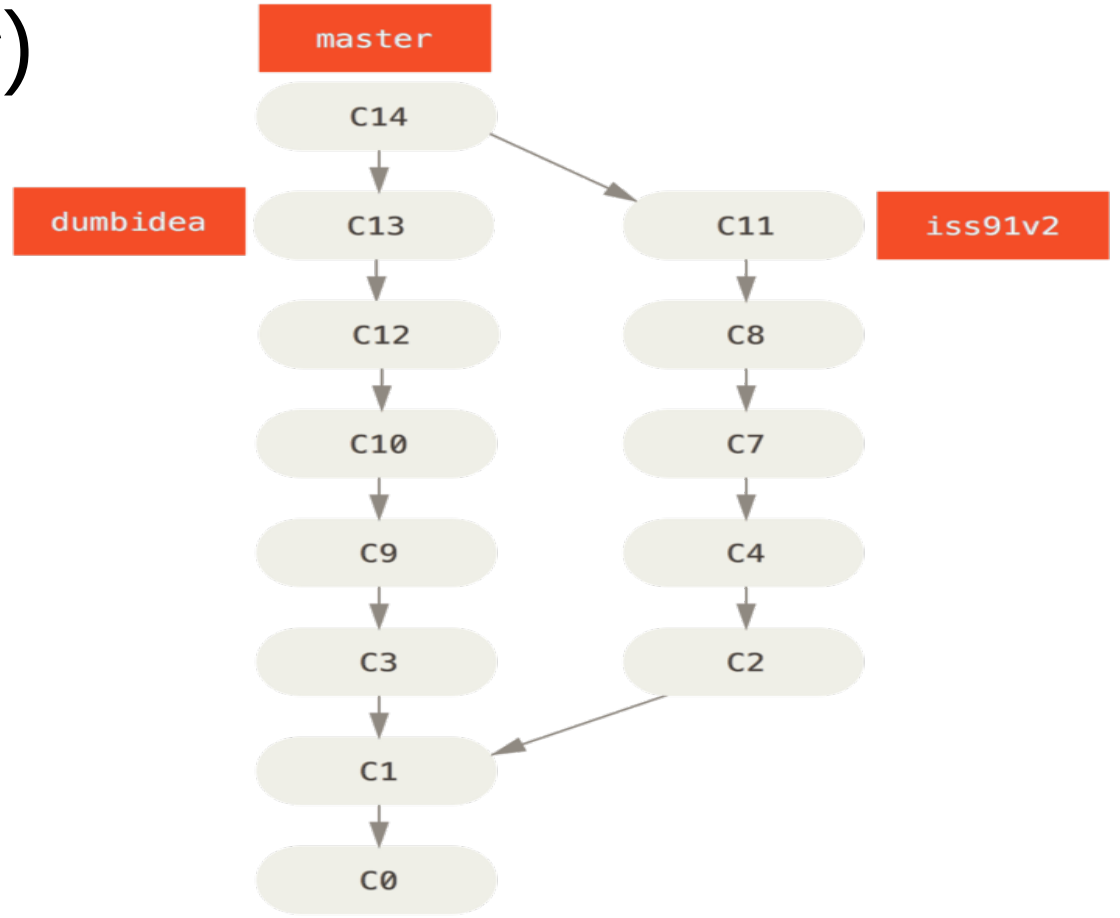
PEM - Git

- Fluxo de trabalho com branches:
  - *Branches* de curta duração (*topic branches*)



PEM - Git

- Fluxo de trabalho com branches:
  - *Branches* de curta duração (*topic branches*)



PEM - Git

- *Branches* remotos
  - Ponteiros para branches no repositório remoto
  - Você pode imprimir uma lista das referências remotas com:

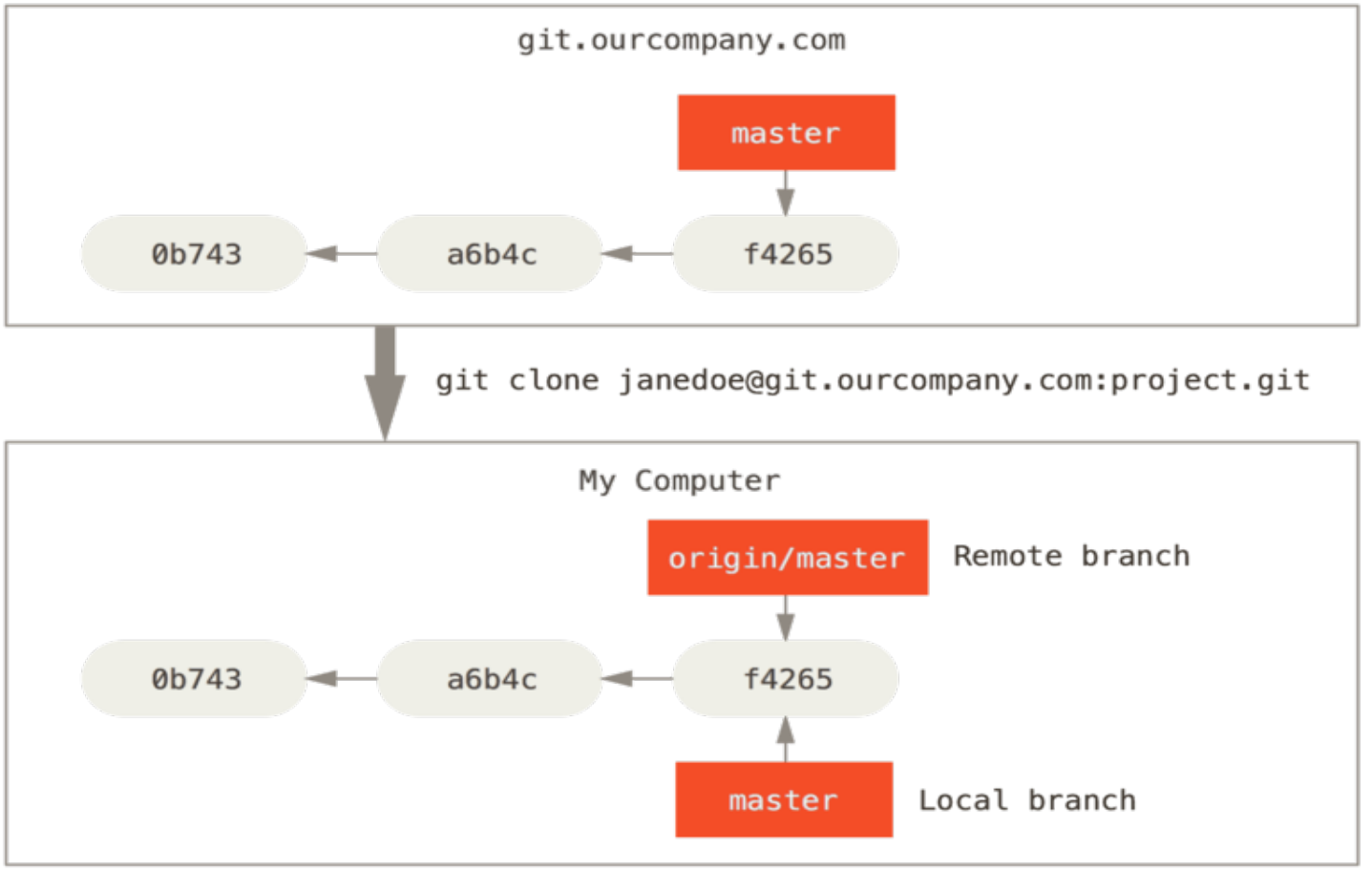
    ```
    $ git ls-remote [repositório remoto]
    ```

  - Ou:

    ```
    $ git remote show [repositório remoto]
    ```

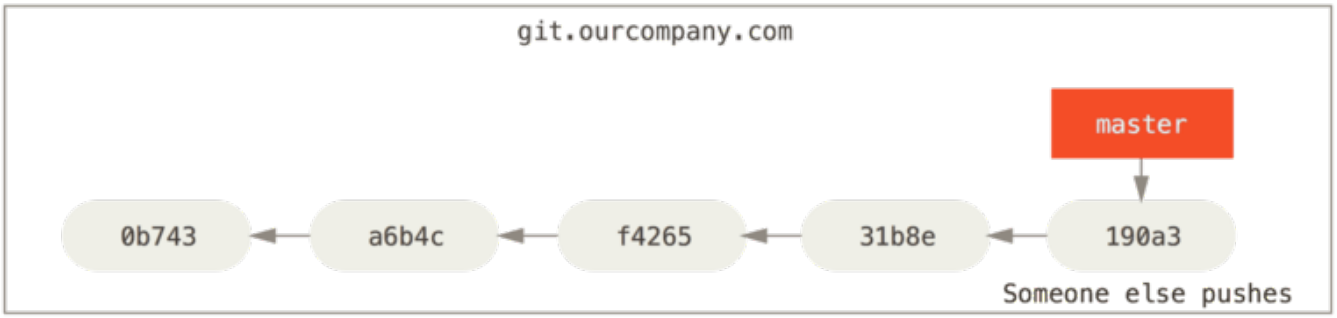
PEM - Git

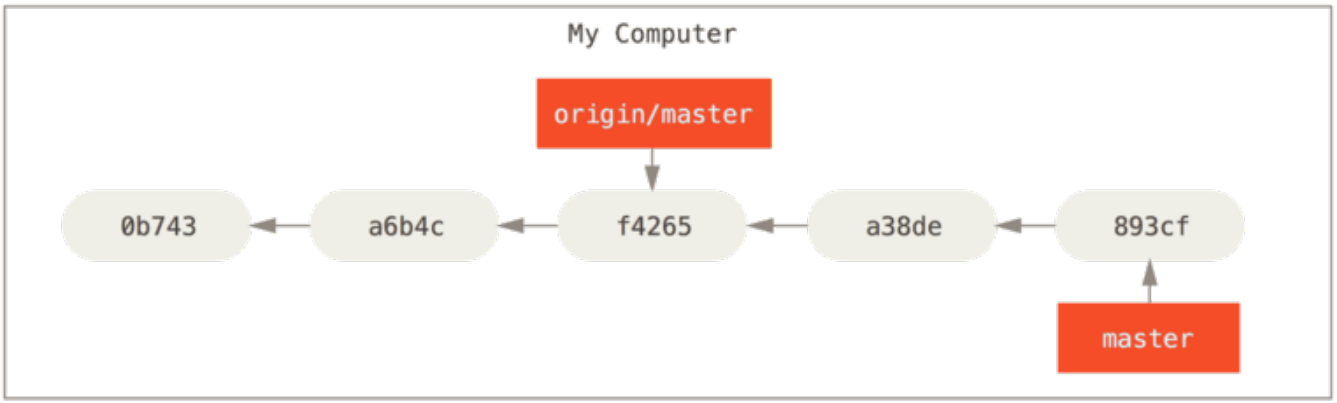
- *Branches* remotos



PEM - Git

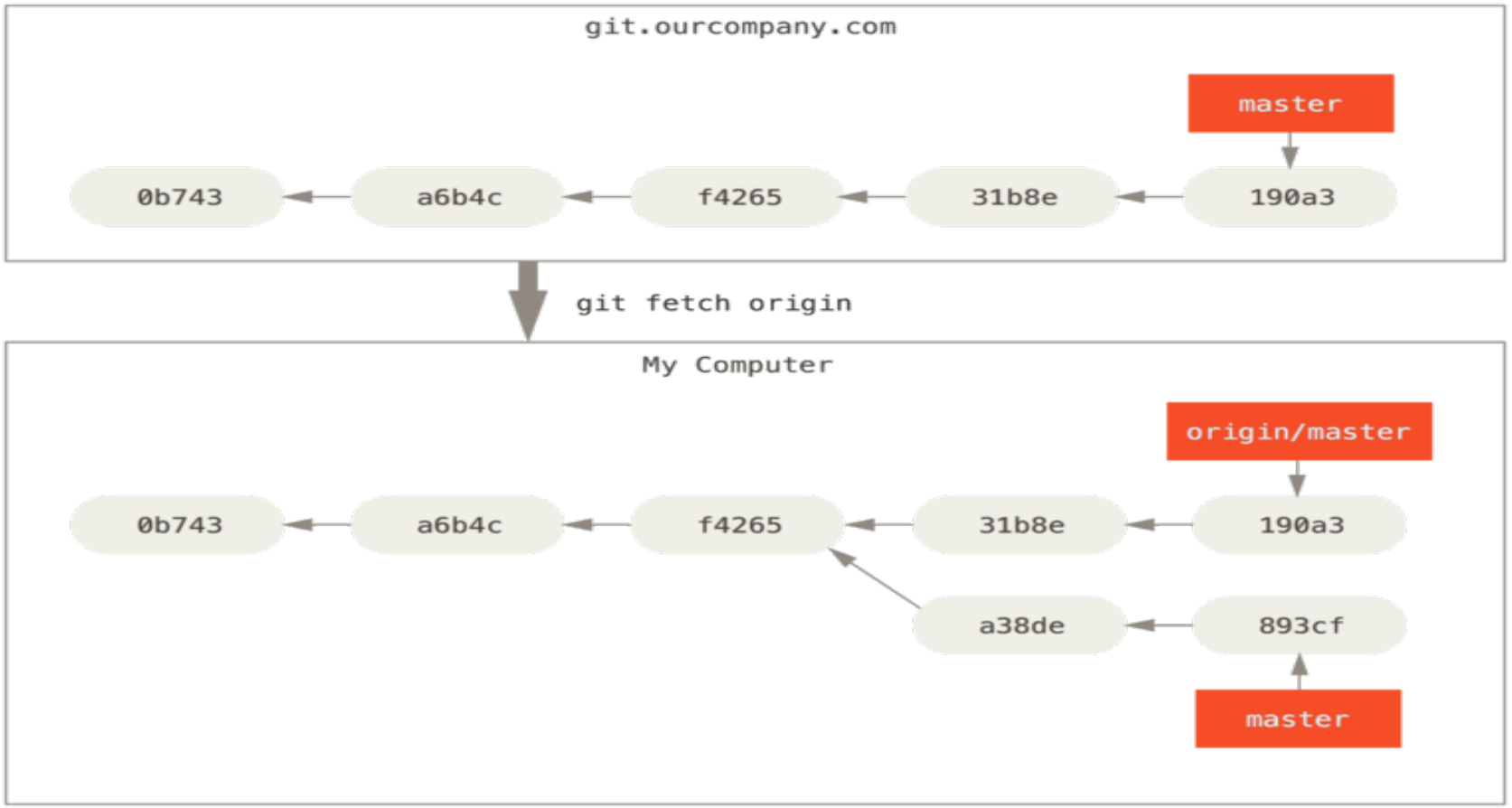
- *Branches* remotos
  - `git fetch`





PEM - Git

- *Branches* remotos
  - `git fetch origin`



PEM - Git

- push, fetch e pull (revisitado)
  - Como colocar um branch específico no servidor?

    ```
    $ git push origin serverfix
    ```

  - Quando algum colaborador pegar o que está no servidor, agora ele terá acesso ao branch serverfix
    - `$ git fetch origin`
    - Esse comando não te dá uma referência editável ao branch!

PEM - Git

- push, fetch e pull (revisitado)
  - E se eu quiser ter uma cópia editável de um *branch*?

```
$ git checkout -b serverfix
origin/serverfix
```

  - git pull é a mesma coisa de git fetch seguido de um git merge

PEM - Git

- Como colaborar em um projeto?

```
# John's Machine
$ git clone john@githost:simplegit.git
Cloning into 'simplegit'...
...
$ cd simplegit/
$ vim lib/simplegit.rb
$ git commit -am 'removed invalid default value'
[master 738ee87] removed invalid default value
 1 files changed, 1 insertions(+), 1 deletions(-)
```

PEM - Git

- Como colaborar em um projeto?

```
# Jessica's Machine
$ git clone jessica@githost:simplegit.git
Cloning into 'simplegit'...
...
$ cd simplegit/
$ vim TODO
$ git commit -am 'add reset task'
[master fbff5bc] add reset task
 1 files changed, 1 insertions(+), 0 deletions(-
```

PEM - Git

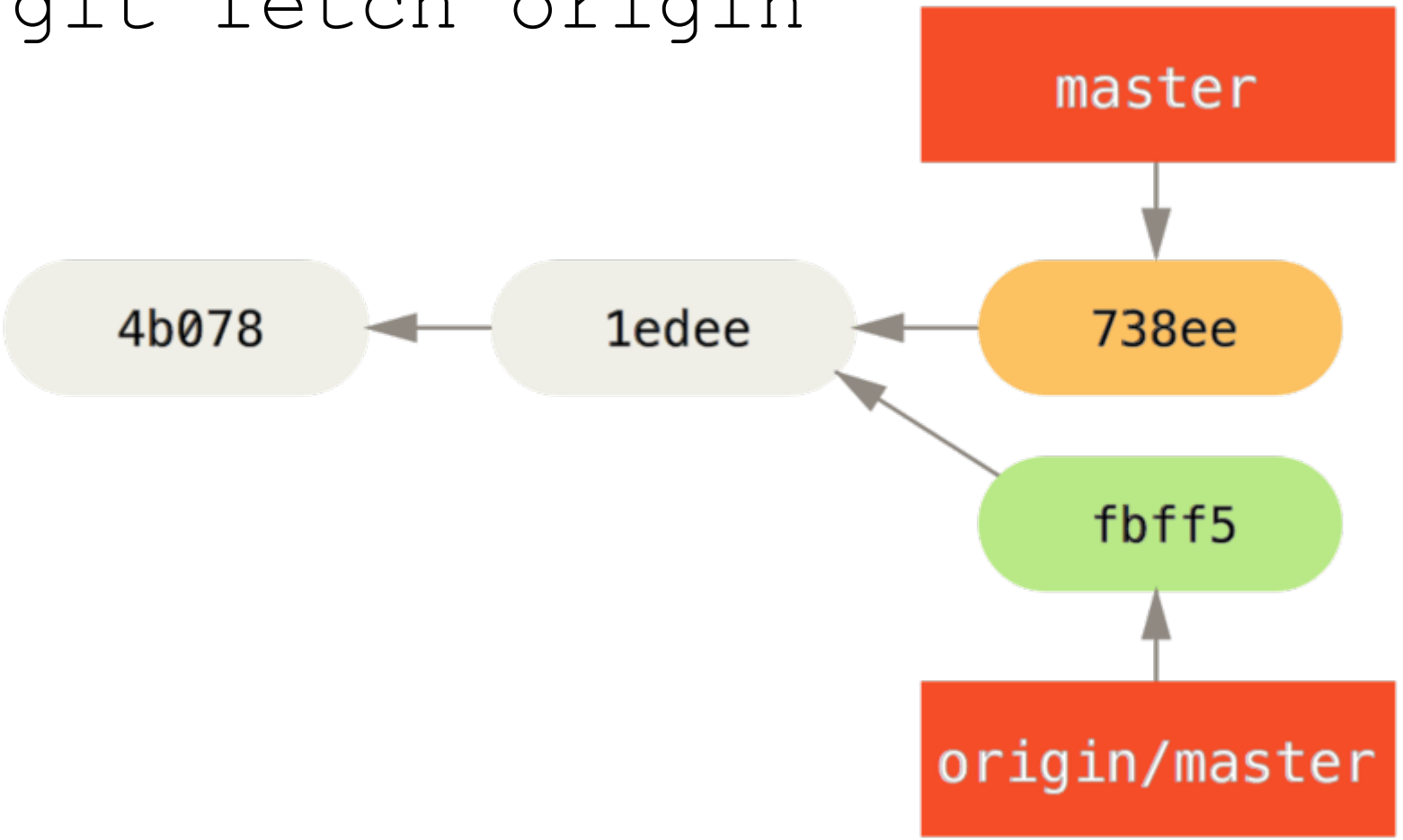
- Como colaborar em um projeto?

```
# Jessica's Machine
$ git push origin master
...
To jessica@githost:simplegit.git
   1edee6b..fbff5bc  master -> master

# John's Machine
$ git push origin master
To john@githost:simplegit.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to
'john@githost:simplegit.git'
```

PEM - Git

- Como colaborar em um projeto?

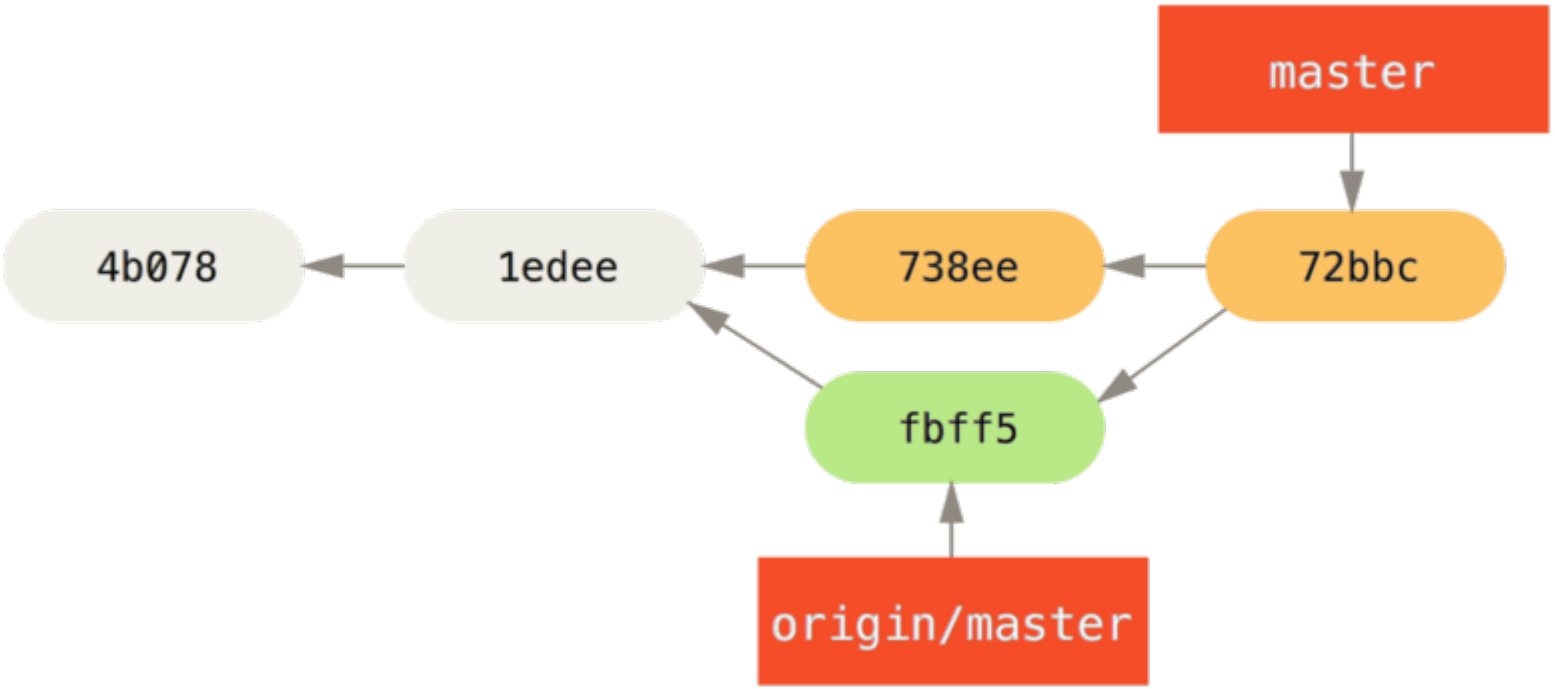
```
$ git fetch origin
```



PEM - Git

- Como colaborar em um projeto?

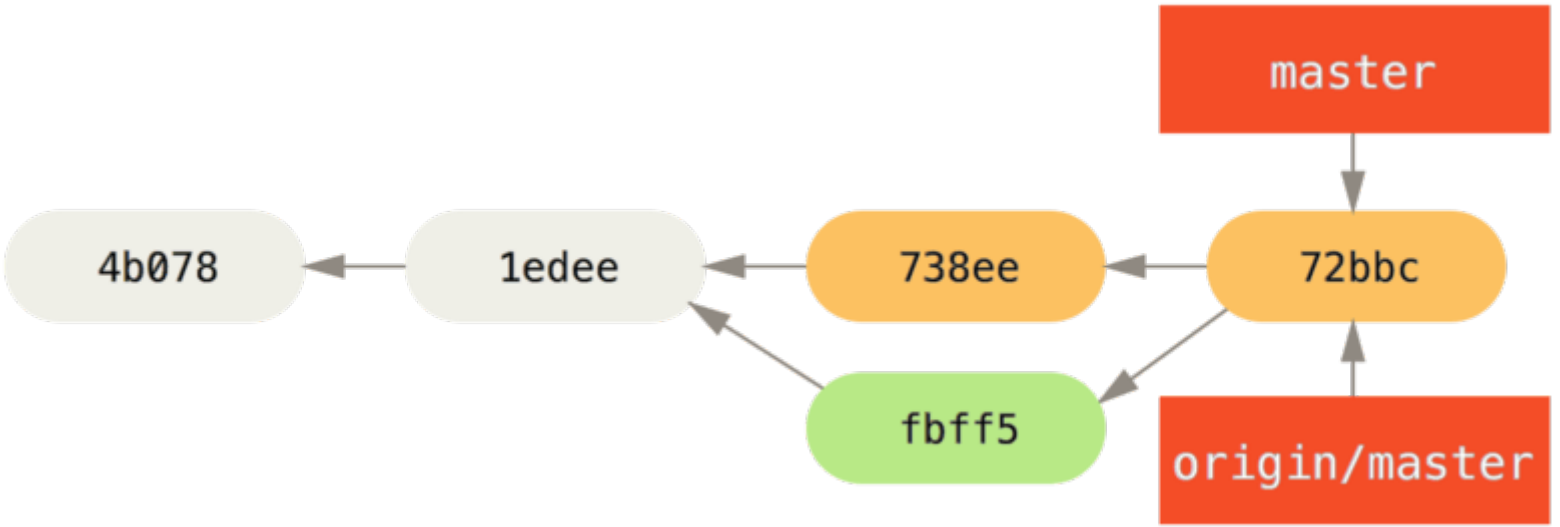
```
$ git merge origin/master
```



PEM - Git

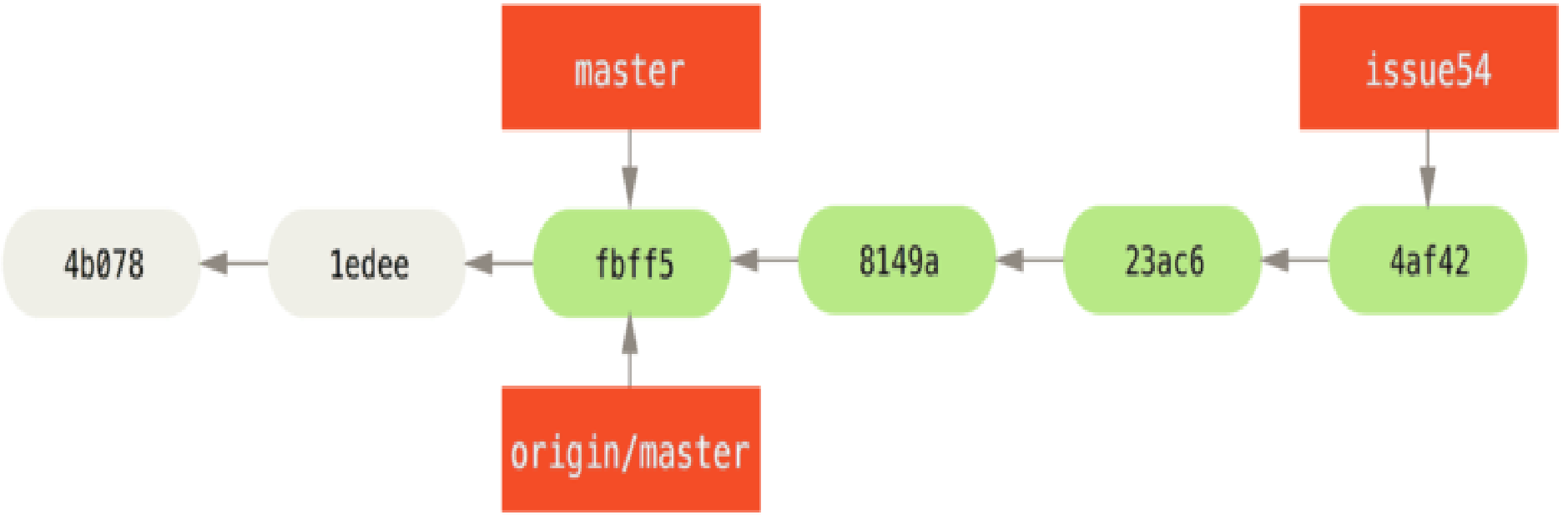
- Como colaborar em um projeto?

```
$ git push origin master
```



PEM - Git

- Como colaborar em um projeto?



PEM - Git

- Como colaborar em um projeto?

```
$ $ git log --no-merges
issue54..origin/master

commit
738ee872852dfaa9d6634e0dea7a324040193016

Author: John Smith <jsmith@example.com>

Date:   Fri May 29 16:01:27 2009 -0700


    removed invalid default value
```

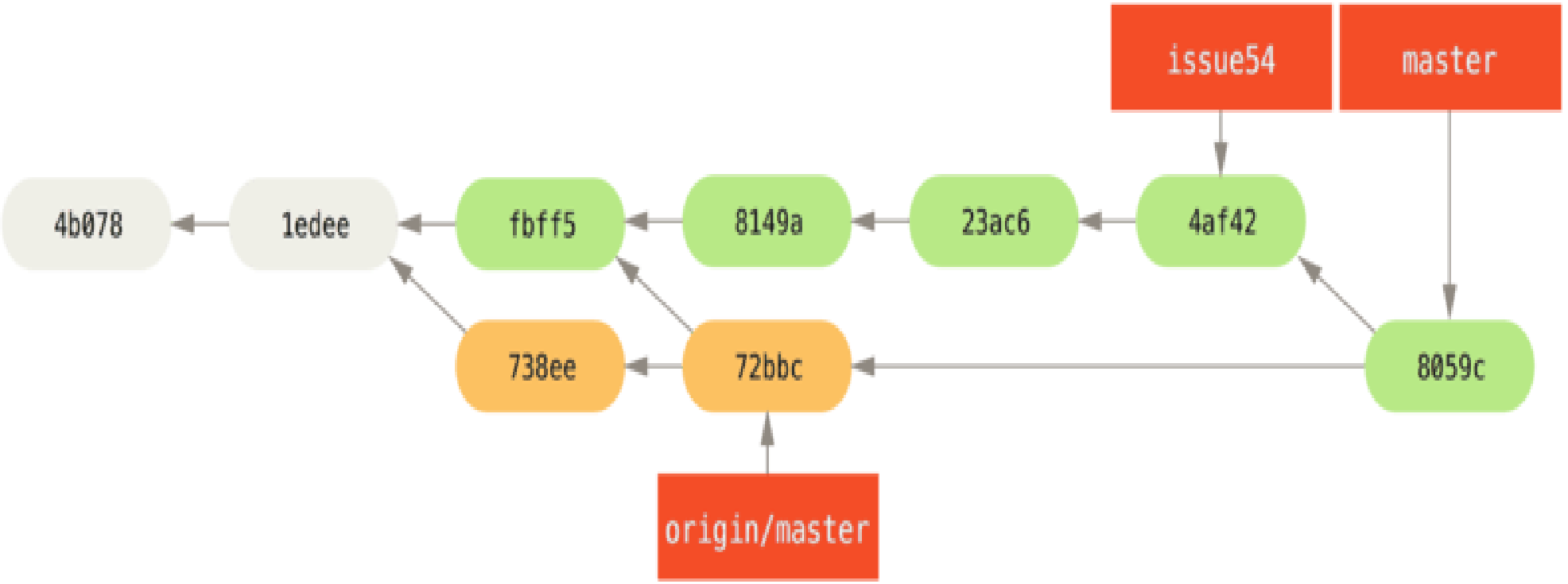
PEM - Git

- Como colaborar em um projeto?

```
$ git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 2
commits, and can be fast-forwarded.
$ git merge issue54
Updating fbff5bc..4af4298
Fast forward
 README              |    1 +
 lib/simplegit.rb    |    6 +++++-
 2 files changed, 6 insertions(+), 1 deletions(-)
```

PEM - Git

- Como colaborar em um projeto?



PEM - Git

- Como colaborar em um projeto?

```
$ git merge origin/master
Auto-merging lib/simplegit.rb
Merge made by recursive.
 lib/simplegit.rb |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

PEM - Git

- Como colaborar em um projeto?