Gestão de Configuração e Mudanças de Software

Allan Lima

Arquitetura, Design e Implementação de Sistemas para Internet
Pós Graduação
Faculdade 7 de Setembro

arglbr@gmail.com

nov/2015

Git

Parte 1

Histórico

- Criado em 2005, por Linus Torvalds
- Substituto do BitKeeper como ferramenta utilizada para controlar as versões do kernel do Linux

Características

- Controle de versão distribuído
- Padrão *de facto* atual
- Extremamente rápido nas operações
- Ótimo gerenciamento de merges e branches

Características

- Snapshots, e não diferenças: fotografias
- Quase todas operações são locais: permite trabalho *offline*
- Tem Integridade: checksum SHA-1 string de 40 caracteres hexadecimais
- Geralmente só adiciona dados: quase impossível perder dados
- Três Estados: consolidado (*committed*), modificado (*modified*) e preparado (*staged*)

Armazenamento

- Outros SCV mantém como um conjunto de arquivos e as mudanças feitas a cada arquivo ao longo do tempo.

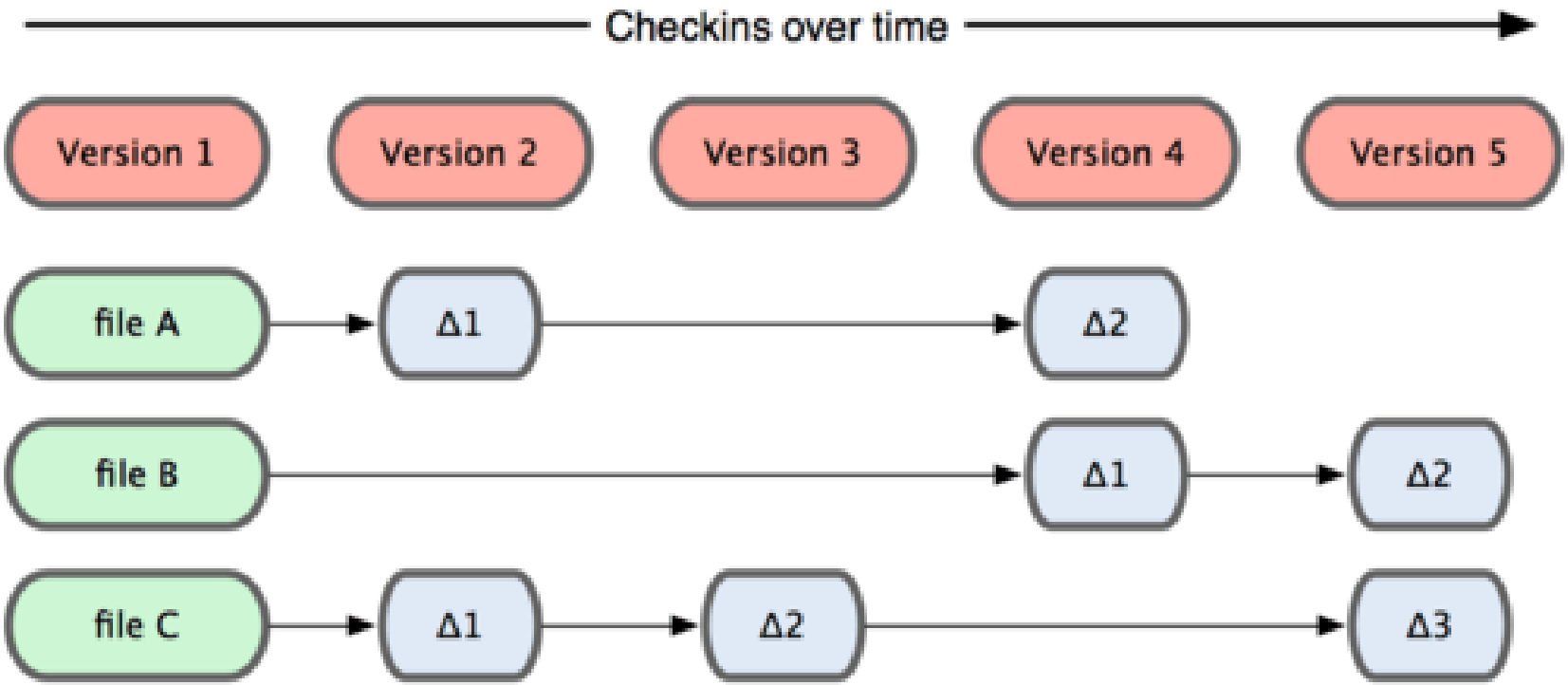


Figure: Ferramentas baseadas em diff (CVS, SVN, Perforce, Bazaar, ...)

Armazenamento

- O Git armazena a nova versão de cada arquivo, e não o que mudou entre elas.
- Considera que os dados são como um conjunto de snapshots (uma foto) de um mini-sistema de arquivos.

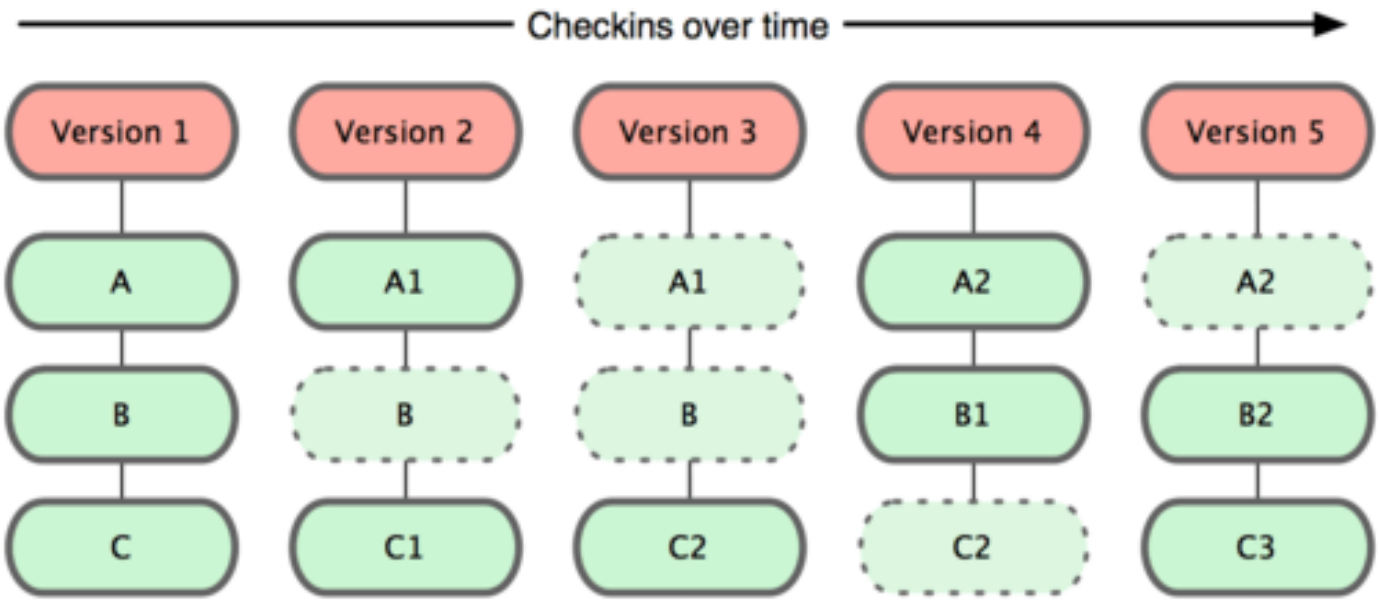


Figure: Git armazena dados como snapshots do projeto ao longo do tempo.

Estados de arquivos

- Estados de arquivos:
  - **Consolidado** (*committed*): dados são ditos consolidados quando estão seguramente armazenados em sua base de dados local
  - **Modificado** (*modified*): Modificado trata de um arquivo que sofreu mudanças mas que ainda não foi consolidado na base de dados
  - **Preparado** (*staged*): Um arquivo está como preparado quando você marca o arquivo modificado em sua versão corrente para que ele faça parte do *snapshot* do próximo *commit* (consolidação)
- Áreas de um projeto:
  - Diretório do Git (*git directory*, *repository*)
  - Diretório de trabalho (*working directory*)
  - Área de preparação (*staging area*).

Áreas de um projeto

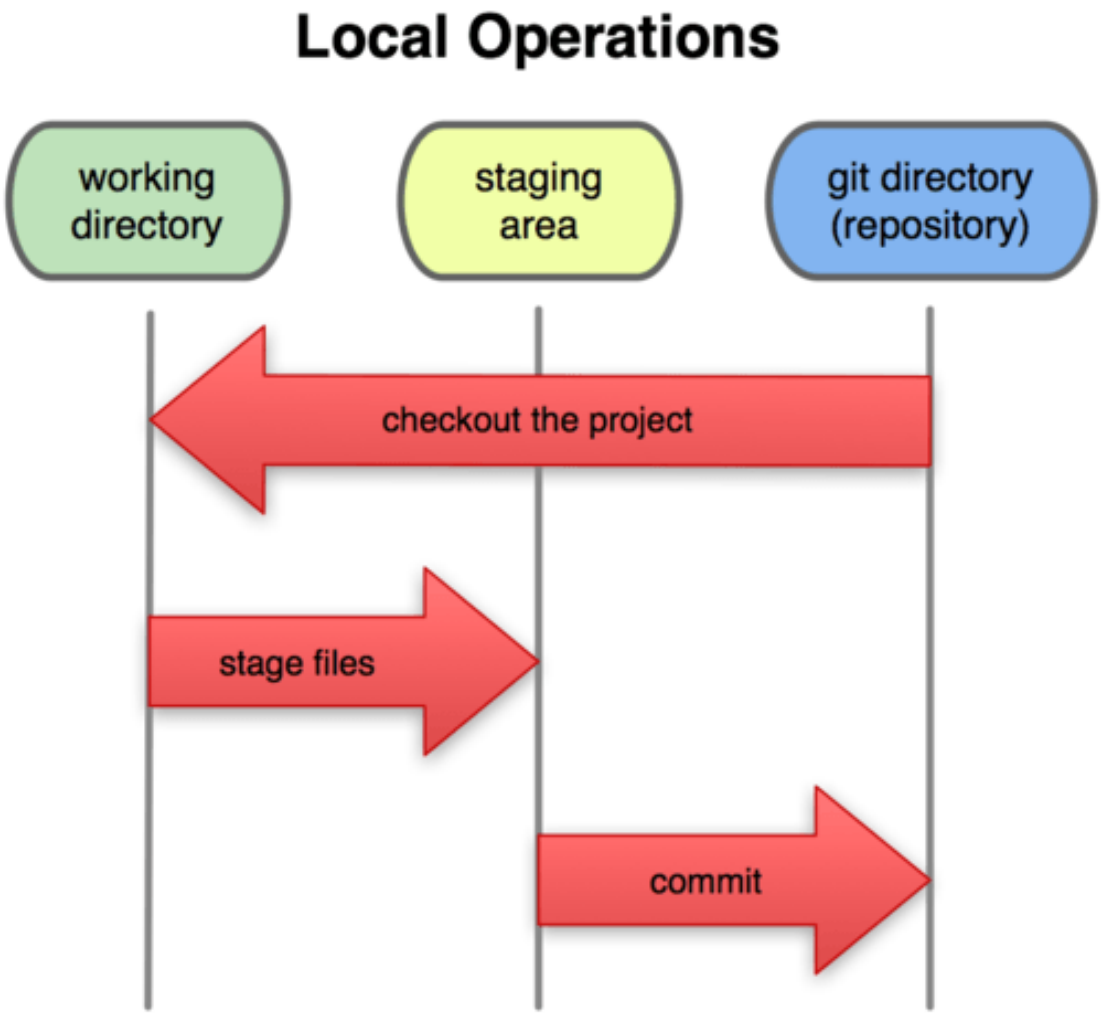


Figure: Estados X Áreas

Ciclo de Vida de um arquivo

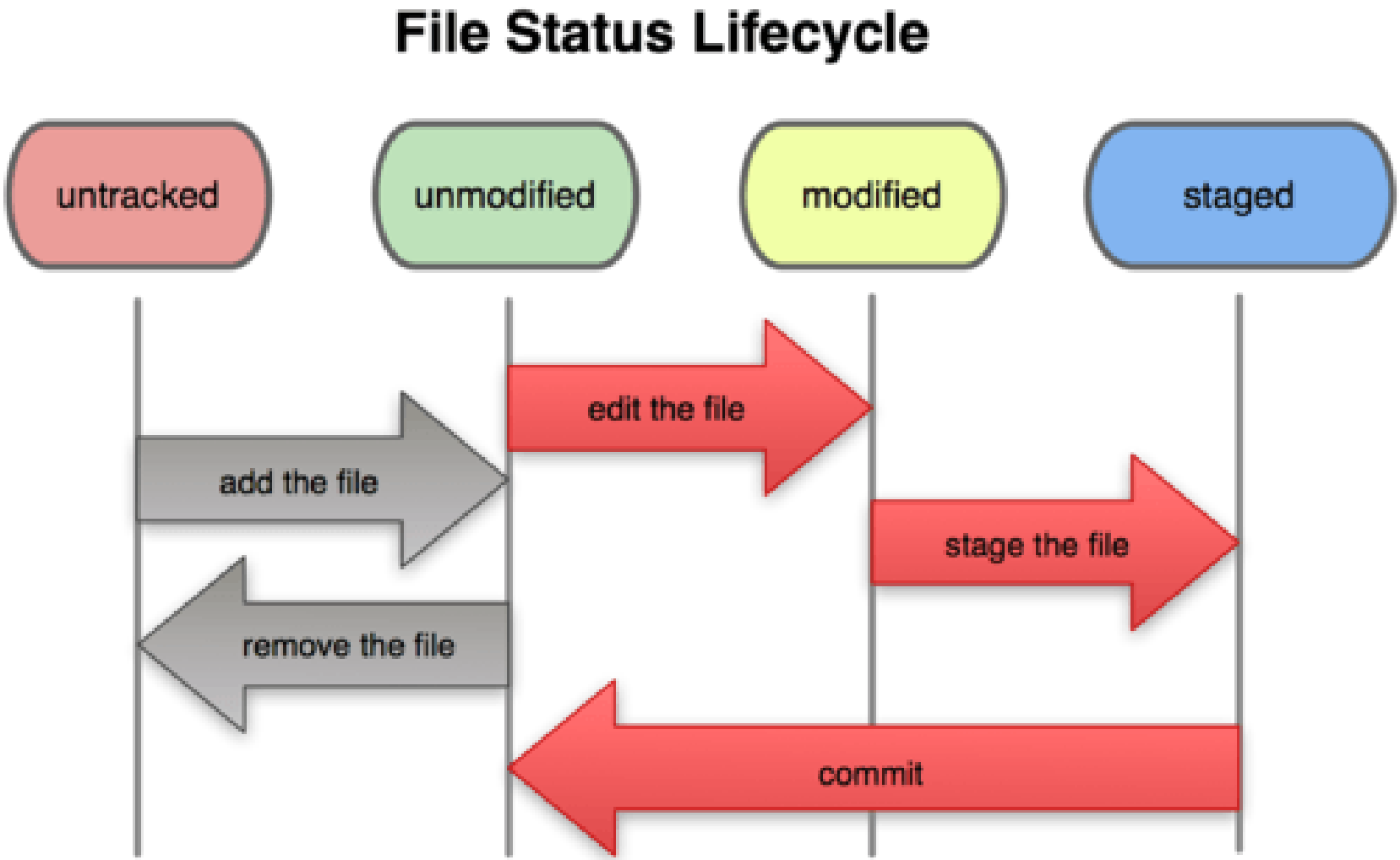


Figure: Ciclo de Vida

Tipos de Objetos

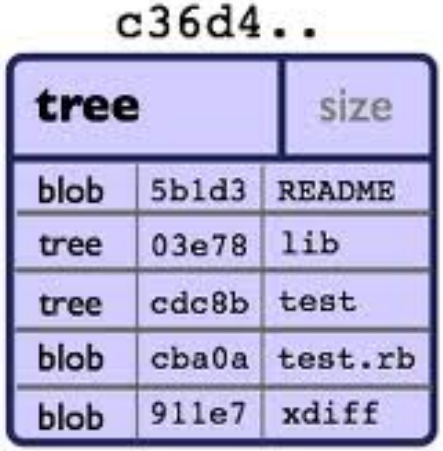
- Blob
- Tree
- Commit

Blob

- Armazena o conteúdo de um arquivo numa determinada versão
- Possui um hash

Tree

- Lista o conteúdo do diretório e especifica quais nomes de arquivos são armazenados em quais *blobs*
- Relaciona *blobs* ou outras *trees* em uma estrutura hierárquica
- Possui um *hash*



Commit

- Guarda metadados de um commit e a tree gerada por ele
- Possui um hash

Os dados no repositório...

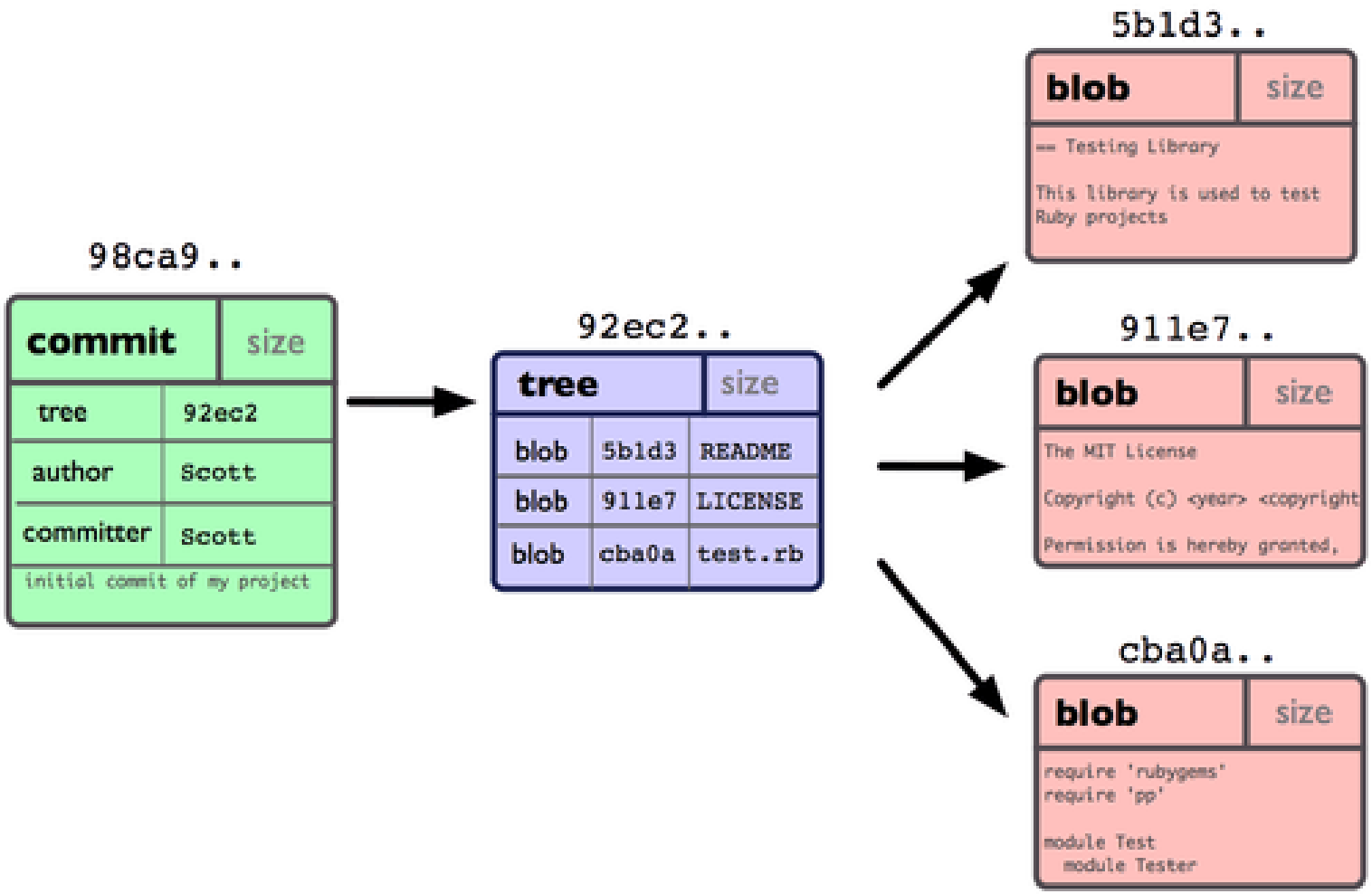


Figure: Dados de um repositório com um único commit

Múltiplos commits

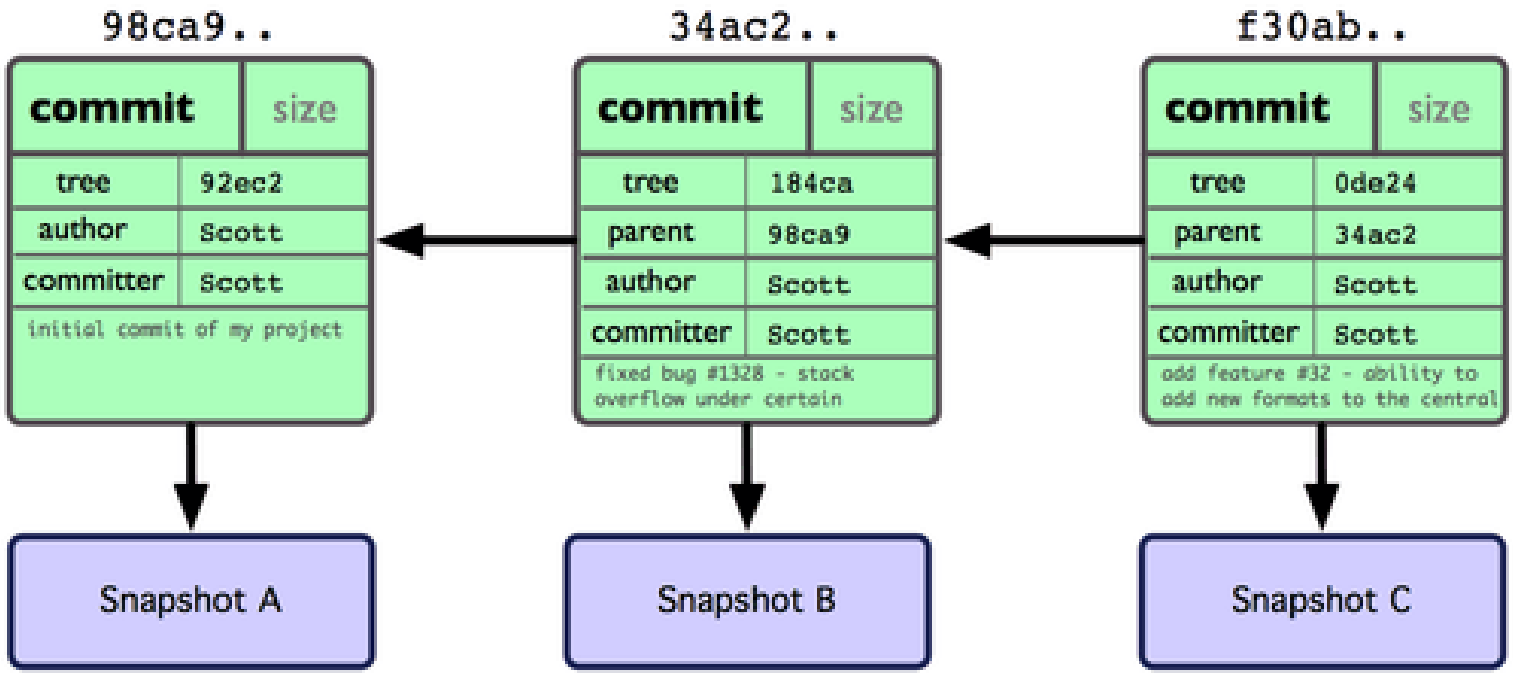


Figure: Dados dos objetos Git para múltiplos commits.

Múltiplos commits

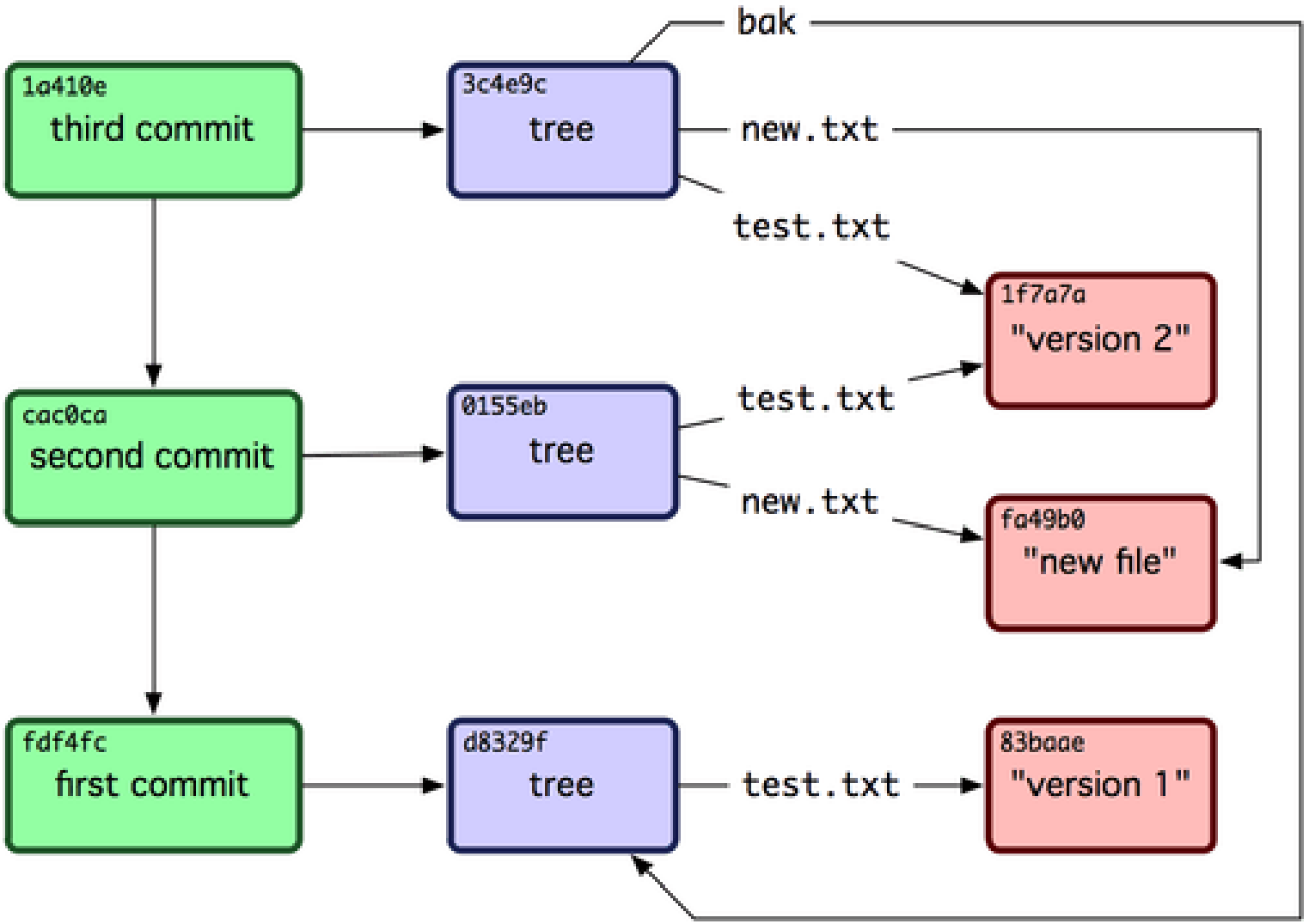


Figure: Dados dos objetos Git para múltiplos commits.

Instalação

- Inicie o VirtualBox da sua estação
- Inicie a VM Windows disponível
- Inicie a VM Linux disponível
- O usuário é **gcs**
- A senha é **gcs123**
- Essa VM funcionará como o Servidor para as práticas
- Tente sempre usar a mesma estação de trabalho

Instalação

- Para a versão Ubuntu, use o comando abaixo:
  - `apt-get install git`
- Para a versão Windows, baixe o arquivo:
  - MSysGit: `http://msysgit.github.com`
  - TortoiseGit: `https://code.google.com/p/tortoisegit/wiki/Download`
- Para a versão Mac OS, baixe o arquivo:
  - `http://code.google.com/p/git-osx-installer`

O Essencial

- Criando e obtendo projetos
  - **init**
  - **clone**
- O básico
  - **add**
  - **status**
  - **diff**
  - **commit**
  - **reset**
  - **rm**
  - **mv**

O Essencial

- Ramos (Branch) e Fusão (Merge)
  - **branch**
  - **checkout**
  - **merge**
  - **mergetool**
  - **log**
  - **stash**
  - **tag**
- Compartilhando e Atualizando projetos
  - **fetch**
  - **pull**
  - **push**
  - **remote**
  - **rebase**

Git

O Básico

Inicializando um Repositório em um Diretório Existente

- Crie uma pasta em

  `c:/curso-gcs/repositorio-git/meusite`
- Dentro da pasta `meusite`, clique o botão da direita e escolha a opção *Git Init Here*
- Ou escolha *Git Gui* e a opção *Criar novo repositório*
- Ou escolha *Git Bash* e no `shell` digite `git init`

Fazendo mudanças

- Crie um arquivo chamado *index.html*
- ```
  <html>
  <body>
      <h1>Hello World!</h1>
  </body>
  </html>
  ```
- Marque o arquivo para rastreado pelo Git: `git add index.html`
- Consolide o arquivo no repositório: `git commit -m "adicionando Hello World"`
- Git vai solicitar seu nome e email
- Veja o `log` da `commit`: `git log`
  ```

Fazendo mudanças

- Modifique o arquivo *index.html*
- ```
  <html>
  <head>
      <title>Hello World in Git</title>
  </head>
  <body>
      <h1>Hello World!</h1>
  </body>
  </html>
  ```
- Tente fazer a consolidação `git commit -m "Alterando Hello World"`
  ```

Fazendo mudanças

- Caso não consiga, use o comando `add` para marcar o arquivo para consolidação e depois consolide.
- Faça uma nova alteração no arquivo. Então para pular a Área de Prepração use o parâmetro `-a`: `git commit -a -m "mais alteracoes"`
- Para remover o arquivo do Git, use o comando `rm` e depois consolide: `git rm`

Fazendo mudanças

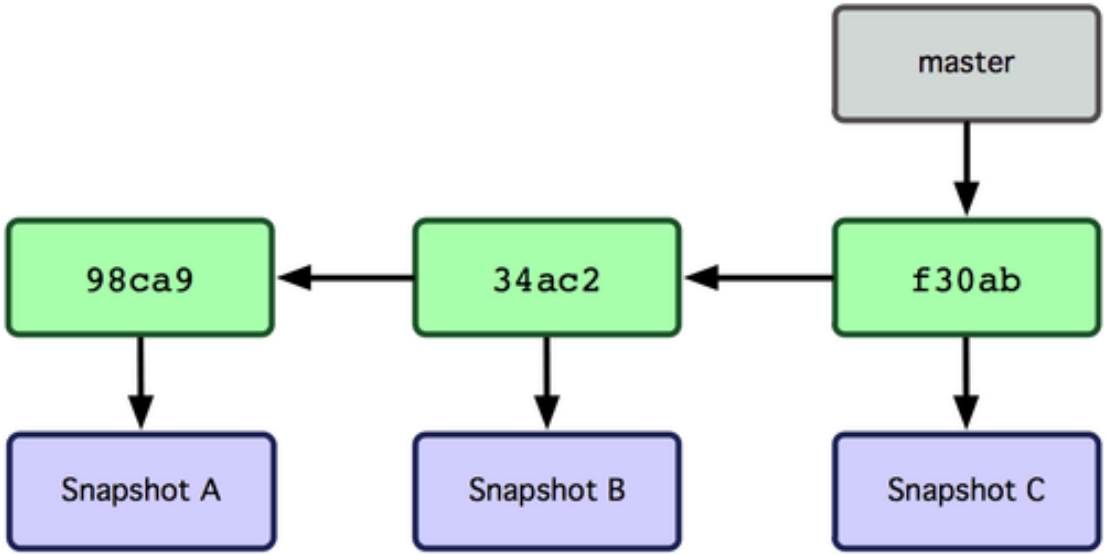
- Modificando seu último `commit`: `git commit --amend`
- Faça um `commit`, adicione um novo arquivo e faça o `git commit --amend`
- Para tirar um arquivo da área de seleção use o comando: `git reset HEAD <file>...`

Git

Branching

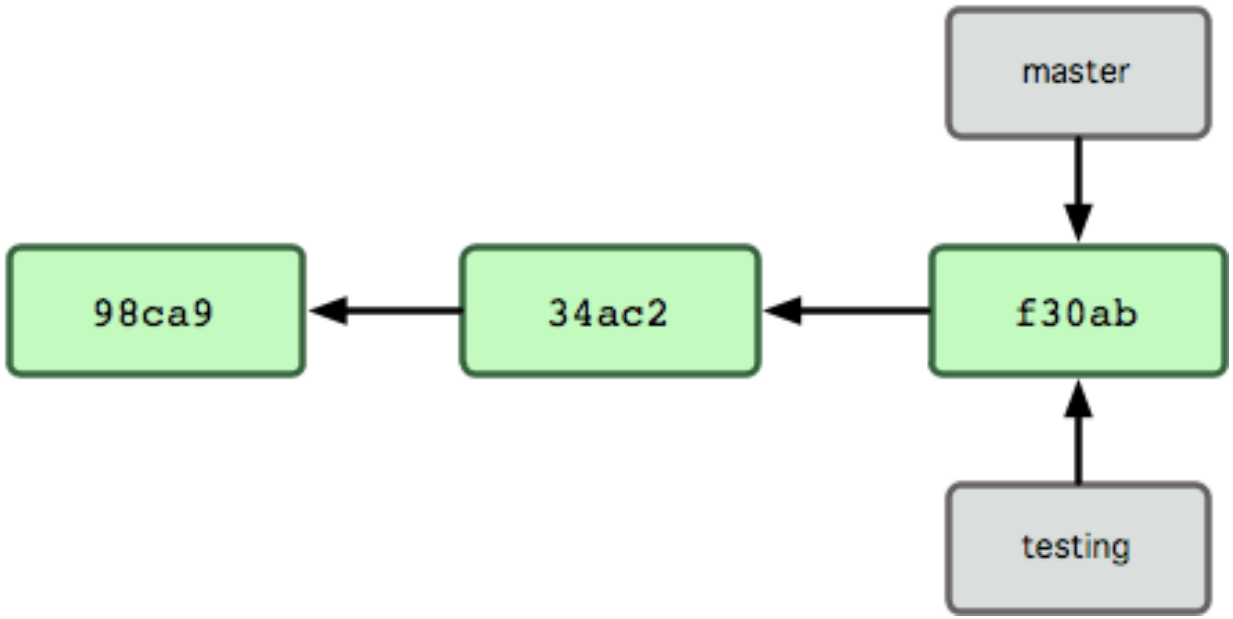
Branch

- Um branch no Git é simplesmente um ponteiro móvel para um desses commits.
- O nome do branch padrão no Git é **master**.
- O `branch` principal (`master branch`) aponta para o último `commit` feito.
- A cada novo commit ele avança automaticamente.



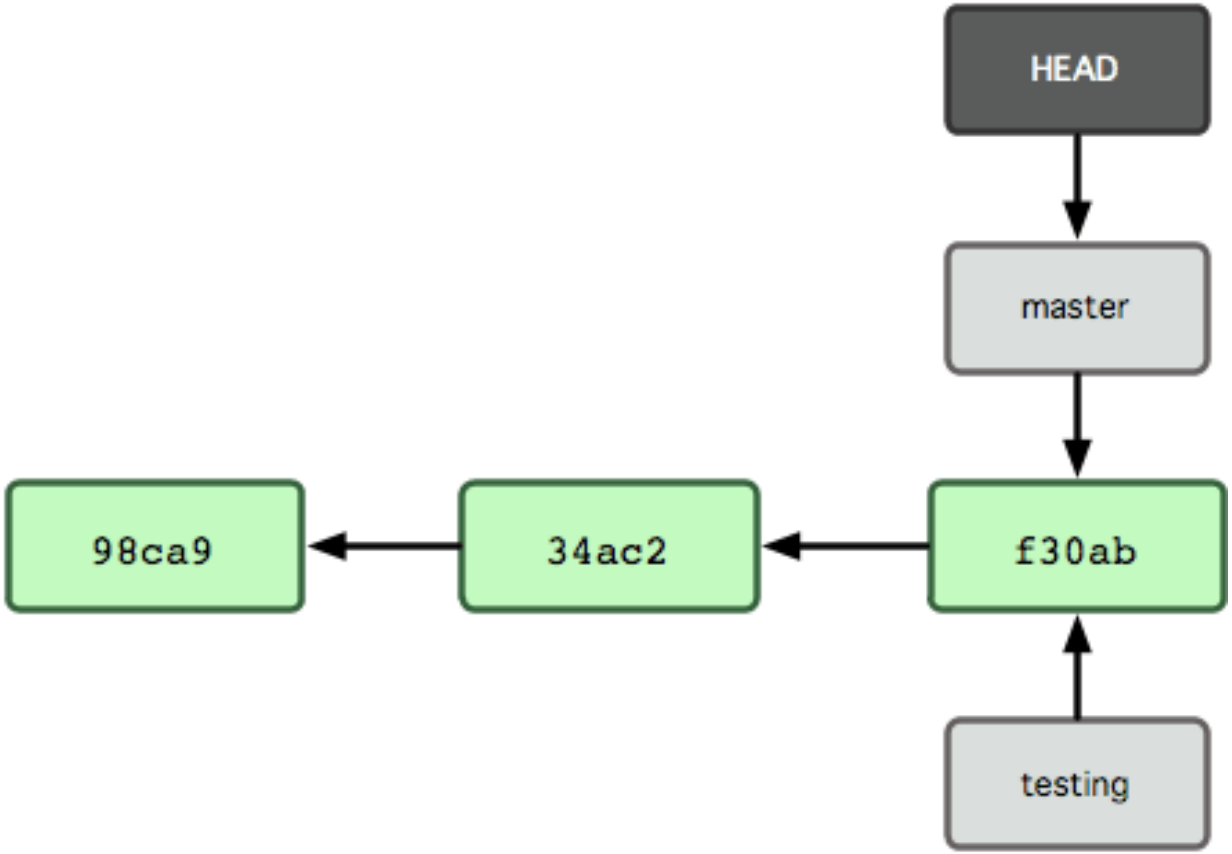
Criando um Branch

- `git branch testing`: cria um novo ponteiro para o mesmo commit em que você está no momento



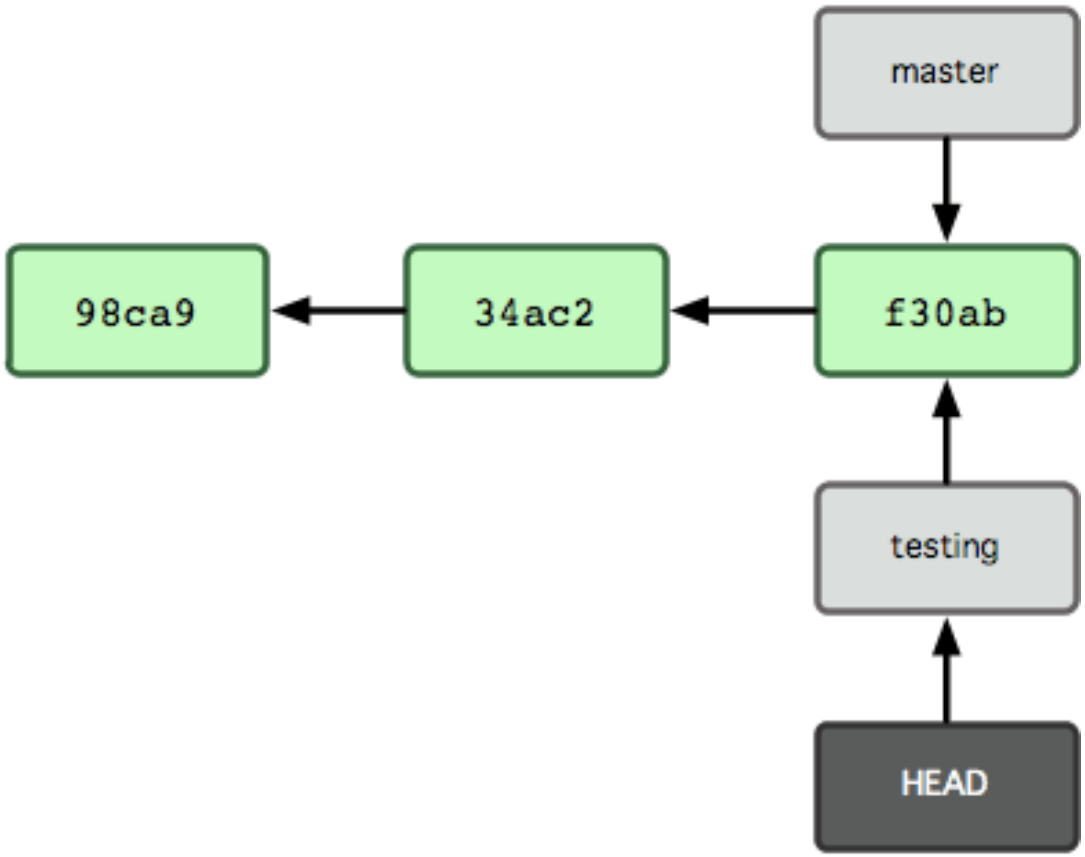
Criando um Branch

- Como o Git sabe o branch em que você está atualmente?
- Ponteiro especial chamado **HEAD**



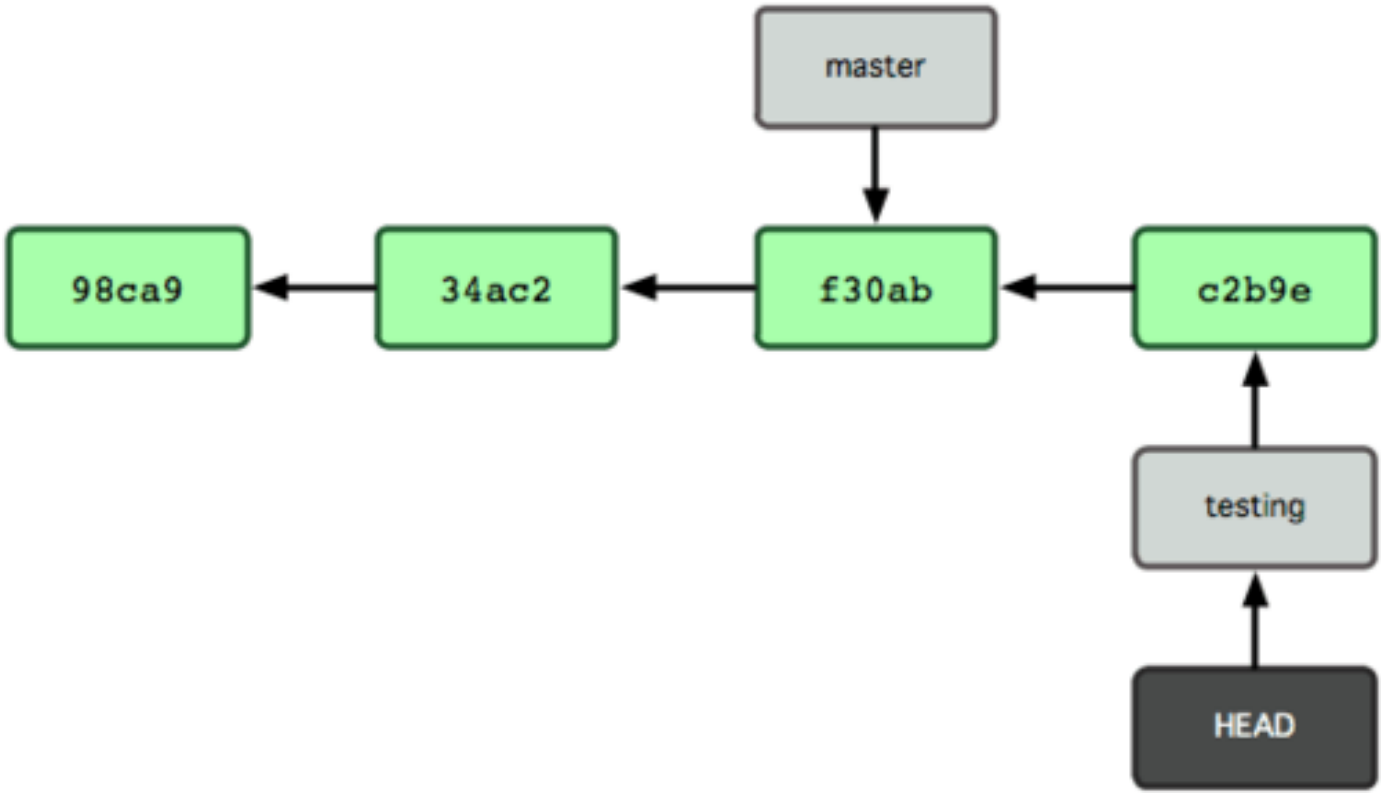
Criando um Branch

- Para mudar para um branch existente, você executa o comando `git checkout`
- `git checkout testing`



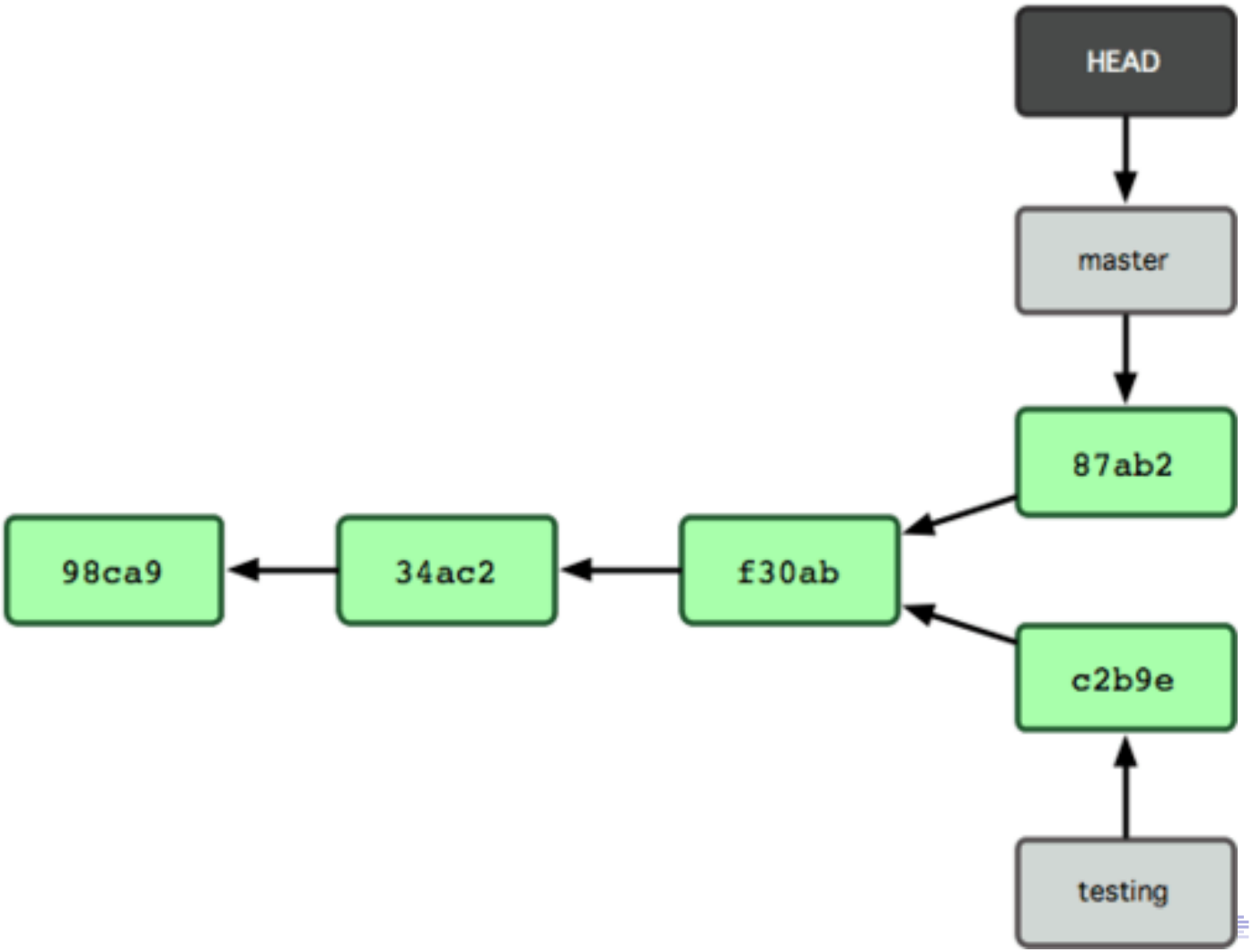
Criando um Branch

- Caso um `commit` seja realizado no `branch`, o **HEAD** acompanha, mas o `master` não.



Criando um Branch

- Voltar para o `master` (`git checkout master`) e realizar um novo `commit` muda o histórico para a figura a seguir:



Criando um Branch

- Para remover o branch `testing` e todos os commits use: `git branch -D testing`

Tagging

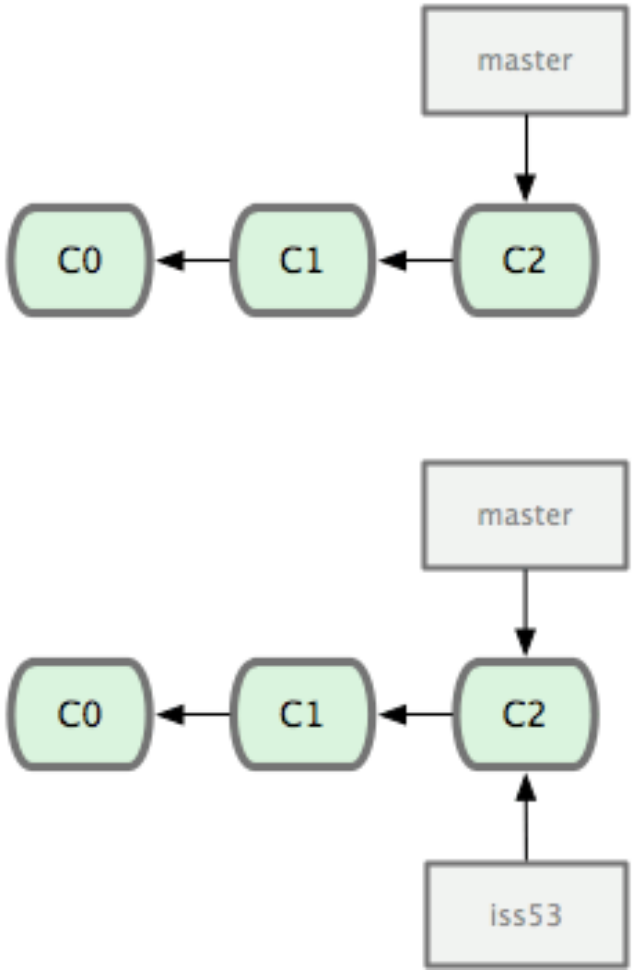
- Listando `tags`: `git tag` ou git tag -l 'v1.4.2'
- Criando `tags`:
  - **leve**: `git tag v1.4` ou `git tag v1.2 9fceb02`
    - Similar a uma branch que não muda — é um ponteiro para um commit específico.
  - **anotada**: `git tag -a v1.4 -m 'my version 1.4'`
    - São objetos inteiros
    - Chave de verificação
    - Nome de quem criou a tag, email e data
    - Mensagem relativa à tag
    - Podem ser assinadas e verificadas com o GNU Privacy Guard (GPG)

Git

Branch e Merge

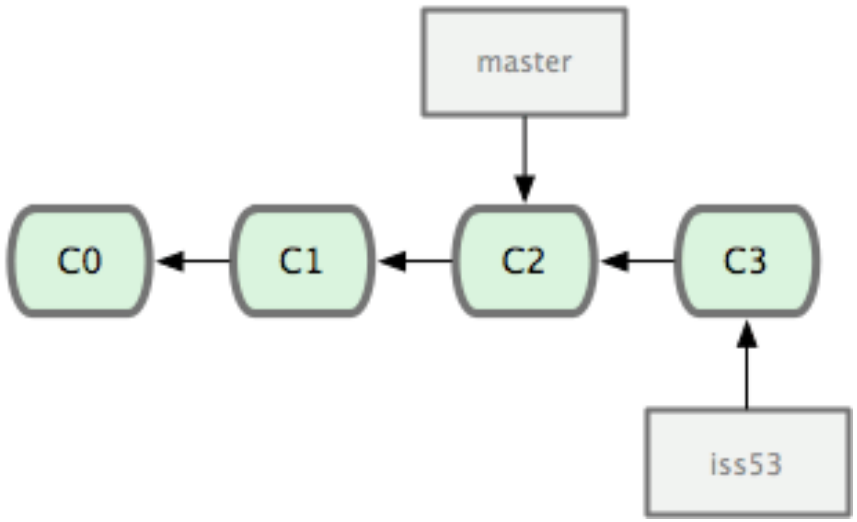
Branch e Merge

- Branch Básico: projeto que já possui alguns commits
- Apareceu em produção um bug: #53
- Será criado um `branch` para tratar a correção.
- Para criar um `branch` e mudar para ele ao mesmo tempo: `git checkout -b iss53`



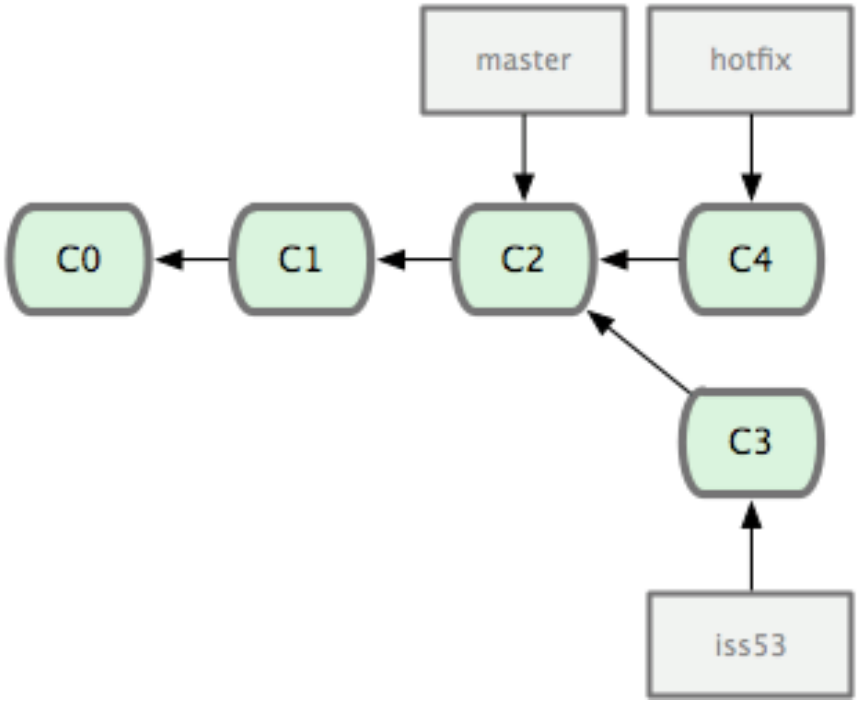
Branch e Merge

- Realize a correção: `git commit -a -m 'adicionei um novo rodapé [issue 53]'`
- Apareceu em produção um bug: #53
- Entretanto, adiaram a publicação e retornou-se para o **master** : `git checkout master`



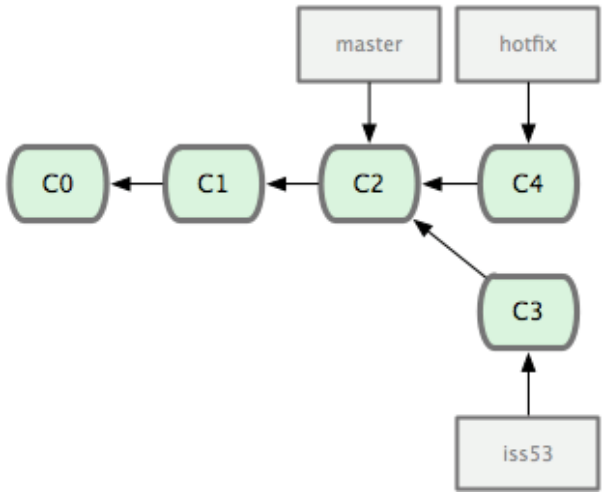
Branch e Merge

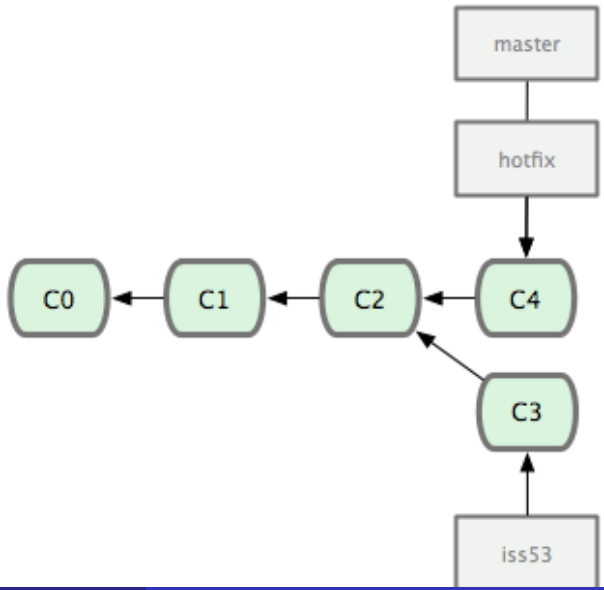
- Surgiu um novo problema urgente
- Então foi criado um `branch` para a correção (`hotfix`): `git checkout -b 'hotfix'`



Branch e Merge

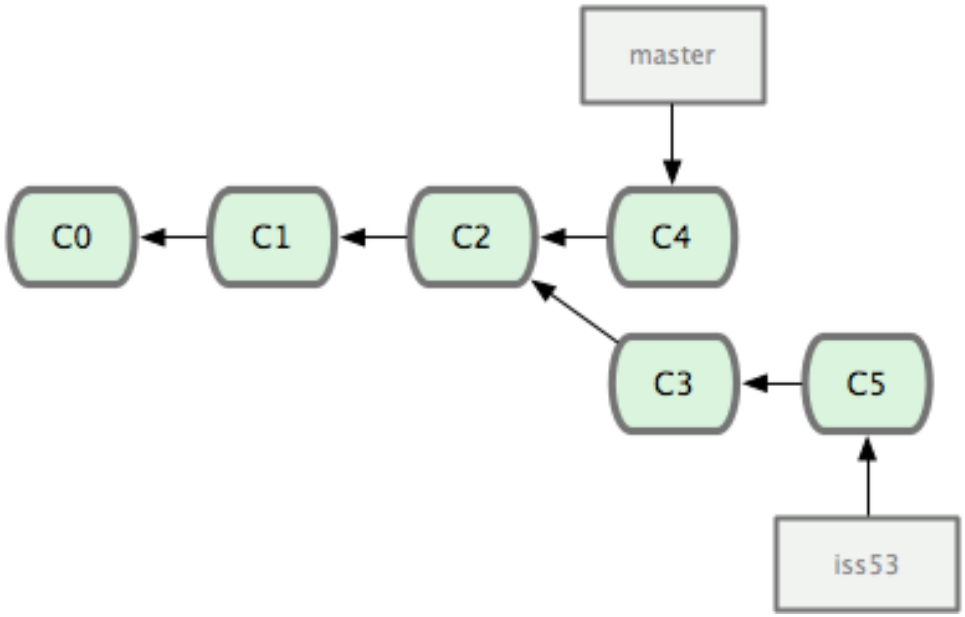
- As correções foram consolidadas, então se faz o `merge` com o `master`
- `git checkout master`
- `git merge hotfix`
- `merge` "Fast forward":
- `hotfix` não é mais necessário: `git branch -d hotfix`





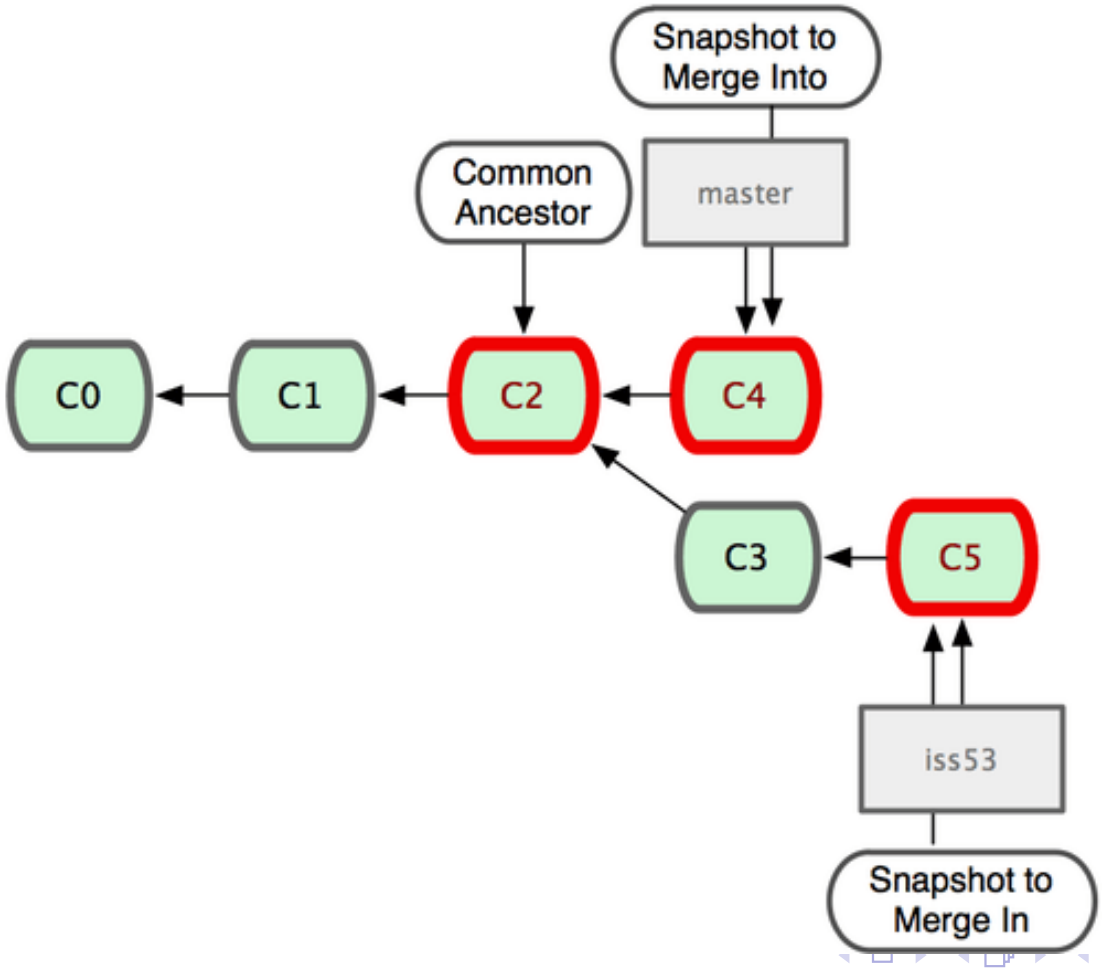
Branch e Merge

- Volta-se a trabalhar na tarefa #53
- `git checkout iss53`
- Novas alterações: `git commit -a -m 'novo rodapé terminado [issue 53]'`



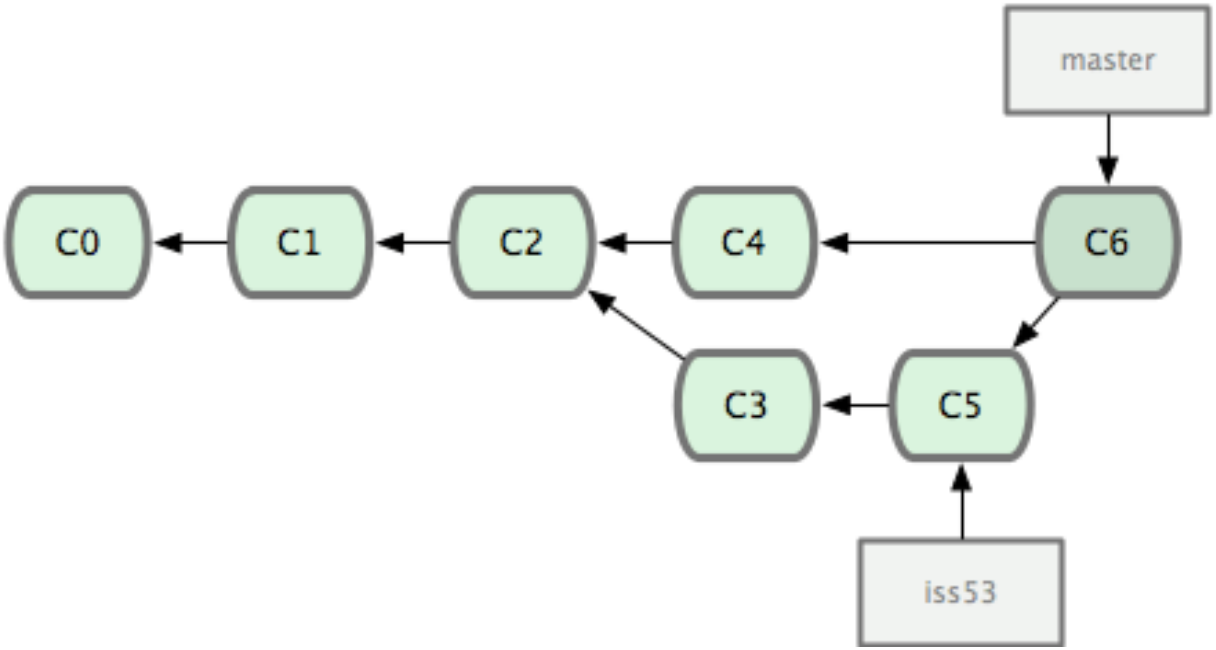
Branch e Merge

- Chegou a hora do `merge`: ancestral direto do `branch` que você está fazendo o merge não é o mesmo
- `merge` simples de três vias



Branch e Merge

- Git cria um novo snapshot que resulta do merge de três vias e automaticamente cria um novo commit que aponta para ele
- Esse commit e é especial pois tem mais de um pai.

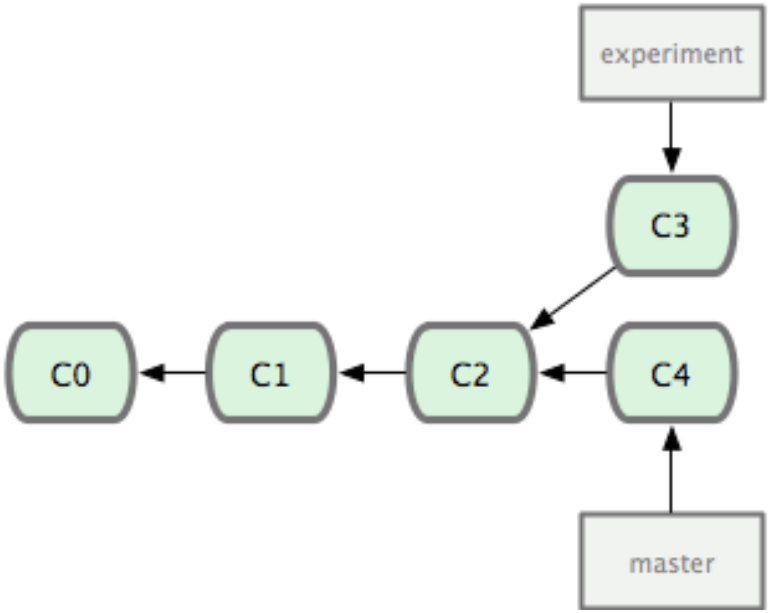


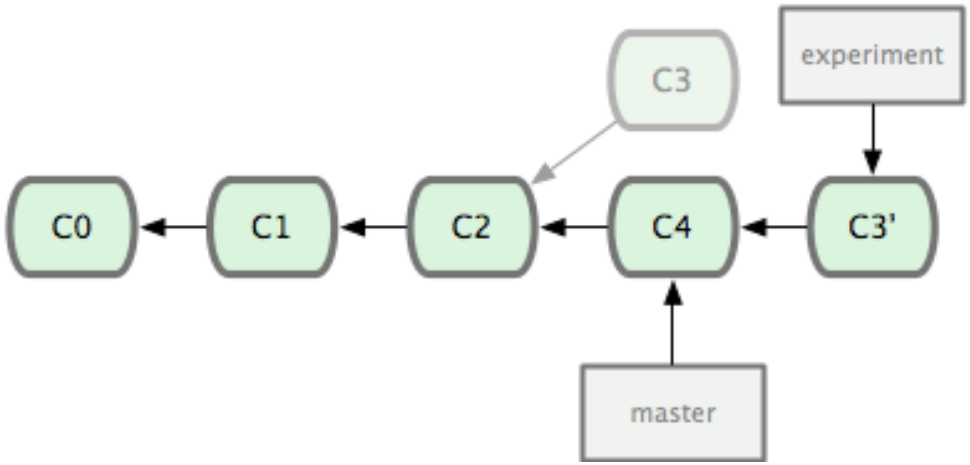
Rebasing

- Há duas forma de se integrar mudanças de branch para outro: `merge` e `rebase`
- Rebasing consiste em replicar todas as mudanças que foram realizadas num branch em outro branch
- Mantém o histórico mais limpo e linear (sem caminhos paralelos como no merge)
- O histórico aparecerá como se todo o trabalho tivesse sido em série quando na verdade foi realizado em paralelo
- Mantém a aparência do branch remoto/servidor mais limpa

Rebasing

- `git checkout experiment`
  `git rebase master`
  `First, rewinding head to replay your work on top of it...`
  `Applying: added staged command`
  `...`
  `git checkout master`
  `git merge experiment`





Git

Git no Servidor

Repositório Remotos: Comandos

- Compartilhando e Atualizando projetos
  - **clone**
  - **remote**
  - **fetch**
  - **pull**
  - **push**

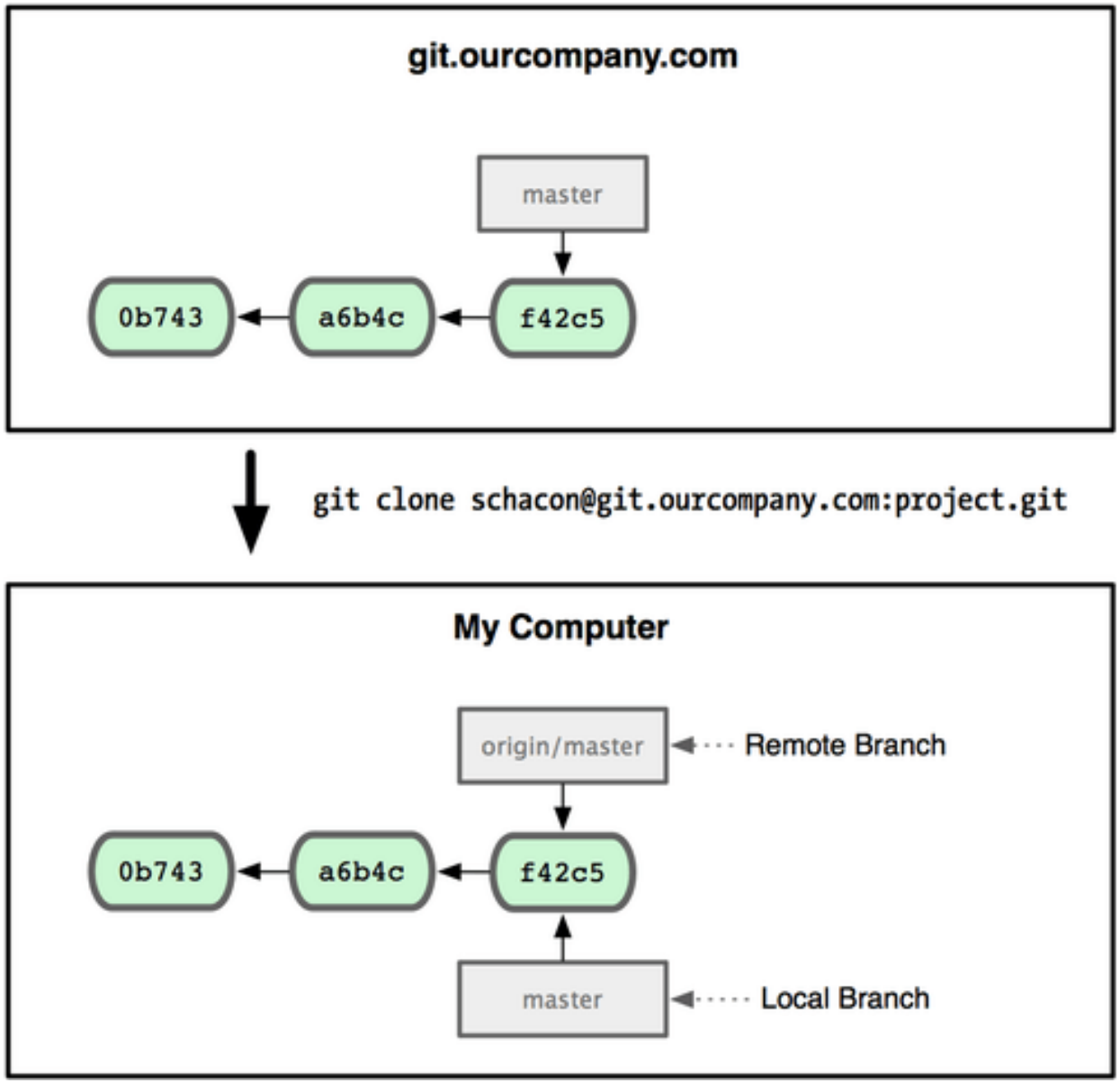
GitHub

- Crie uma conta em `https://github.com/`
- Navegue para o repositório `https://github.com/octocat/Spoon-Knife`
- No canto superior direito, clique em **Fork** (Copia o repositório e permite alterá-lo livremente sem afetar o projeto original)
- Adicione um Novo Colaborador para o projeto Spoon-Knife: pesquise pelo usuário `aluno-fa7-pos01`
- Usaremos o `aluno-fa7-pos01` para simular usuário diferente
- A senha `aluno-fa7-pos01` é `posgradfa7`

git clone

- Para copiar um repositório Git já existente use: `git clone`
- Recebe uma cópia de quase todos os dados que o servidor possui
- Exemplo:
  `git clone git://github.com/schacon/grit.git mygrit`
- Exercício: clone duas vezes o repositório SpoonKnife como abaixo
  `git clone https://github.com/arglbrce/Spoon-Knife.git Spoon-Knife`
  `git clone https://github.com/arglbrce/Spoon-Knife.git Spoon-Knife-aluno-fa7-pos01`

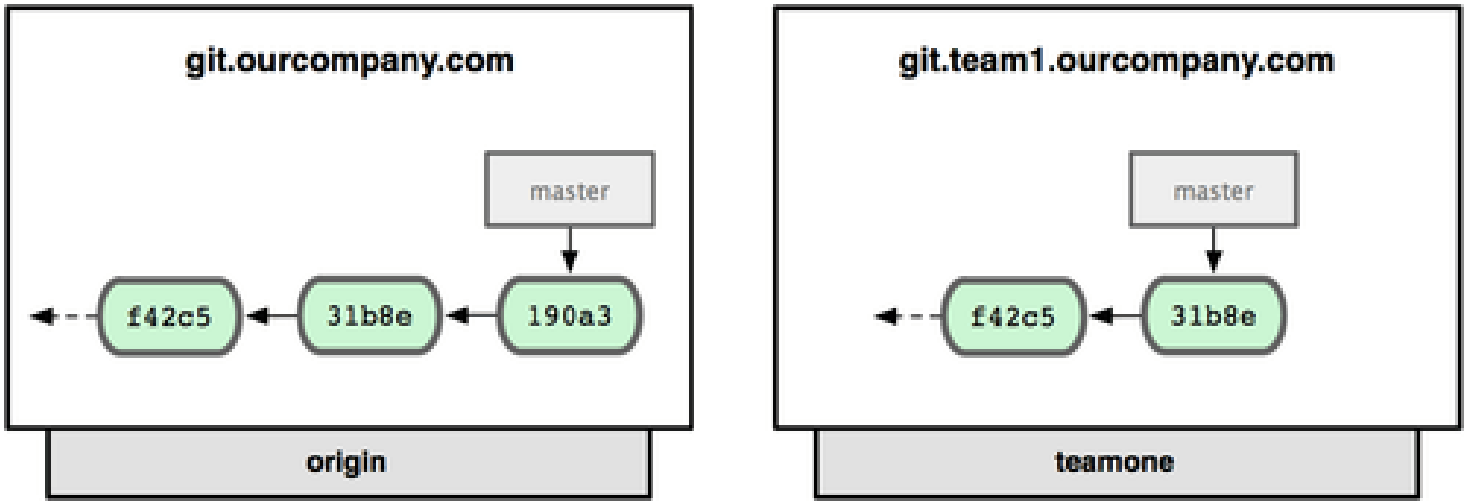
git clone


git.ourcompany.com
master
0b743
a6b4c
f42c5
git clone schacon@git.ourcompany.com:project.git
My Computer
origin/master
Remote Branch
0b743
a6b4c
f42c5
master
Local Branch

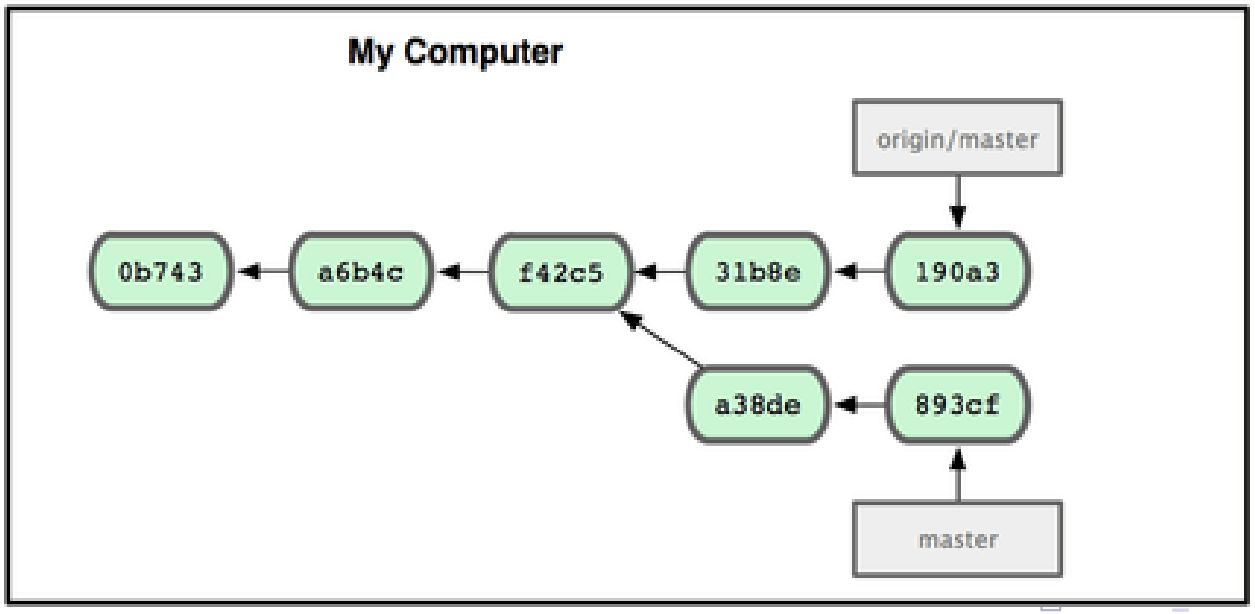
git remote

- `git remote`: lista o nome dos repositórios remotos
- `git remote -v`: lista o nome dos repositórios remotos e a URL
- `git remote add [shortname] [url]`: adiciona um repositório remoto ao repositório local
- Exemplo:
  `git remote add pb https://github.com/paulboone/ticgit`
- Exercício: adicionar o repositório do seu vizinho
  `git remote add vizinho`
  `https://github.com/vizinho/Spoon-Knife`
- `git remote rename pb paul`: renomeia um repositório remoto
- `git remote rm paul`: remove um repositório remoto
- `git remote show origin`: exibe detalhes do repositório remoto em relação ao repositório local. Quais branches estão sendo rastreadas

git remote



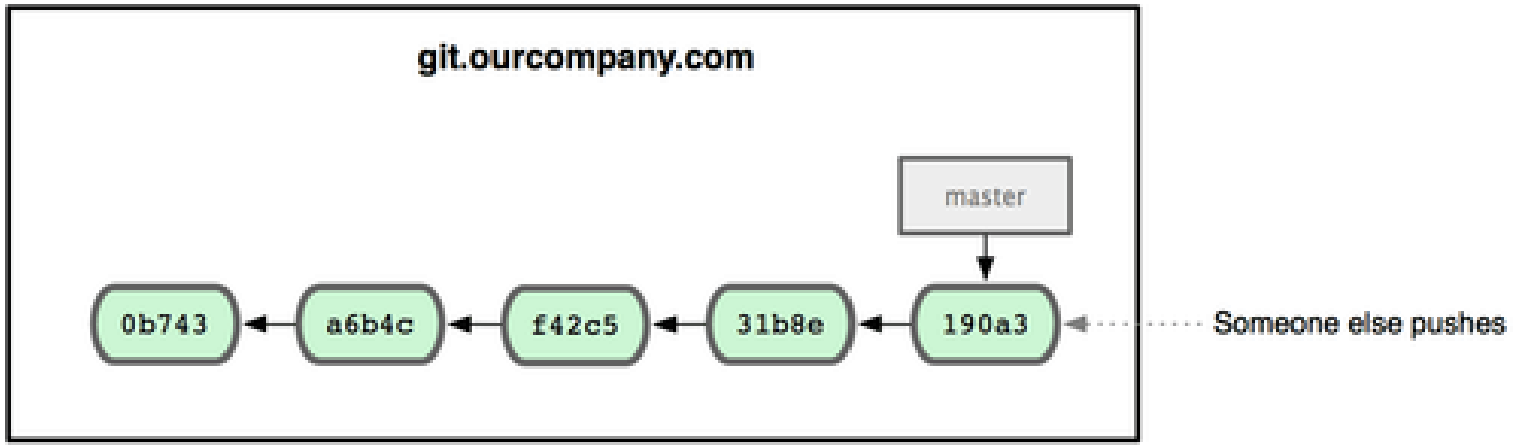

```
git remote add teamone git://git.team1.ourcompany.com
```

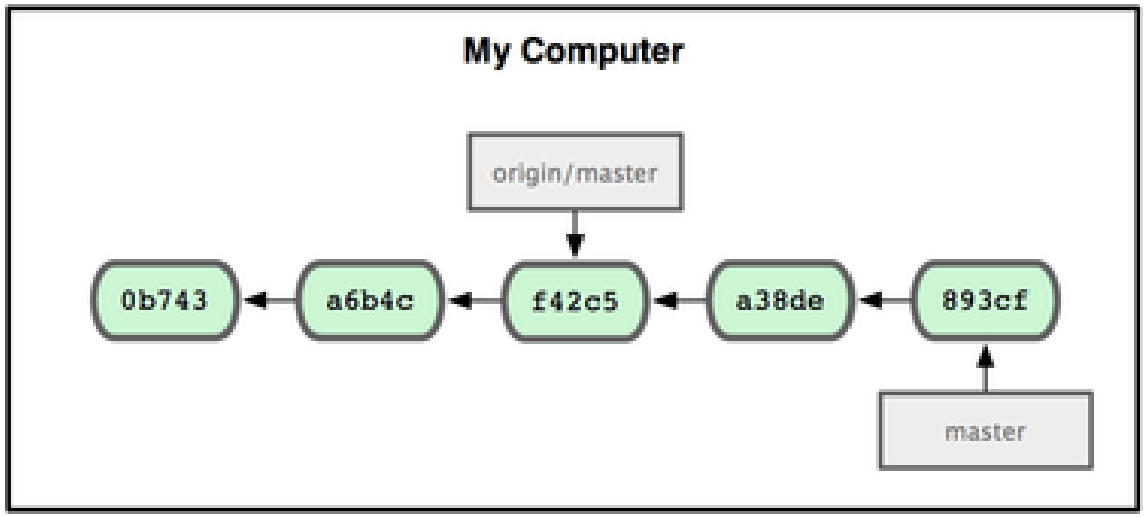


git fetch

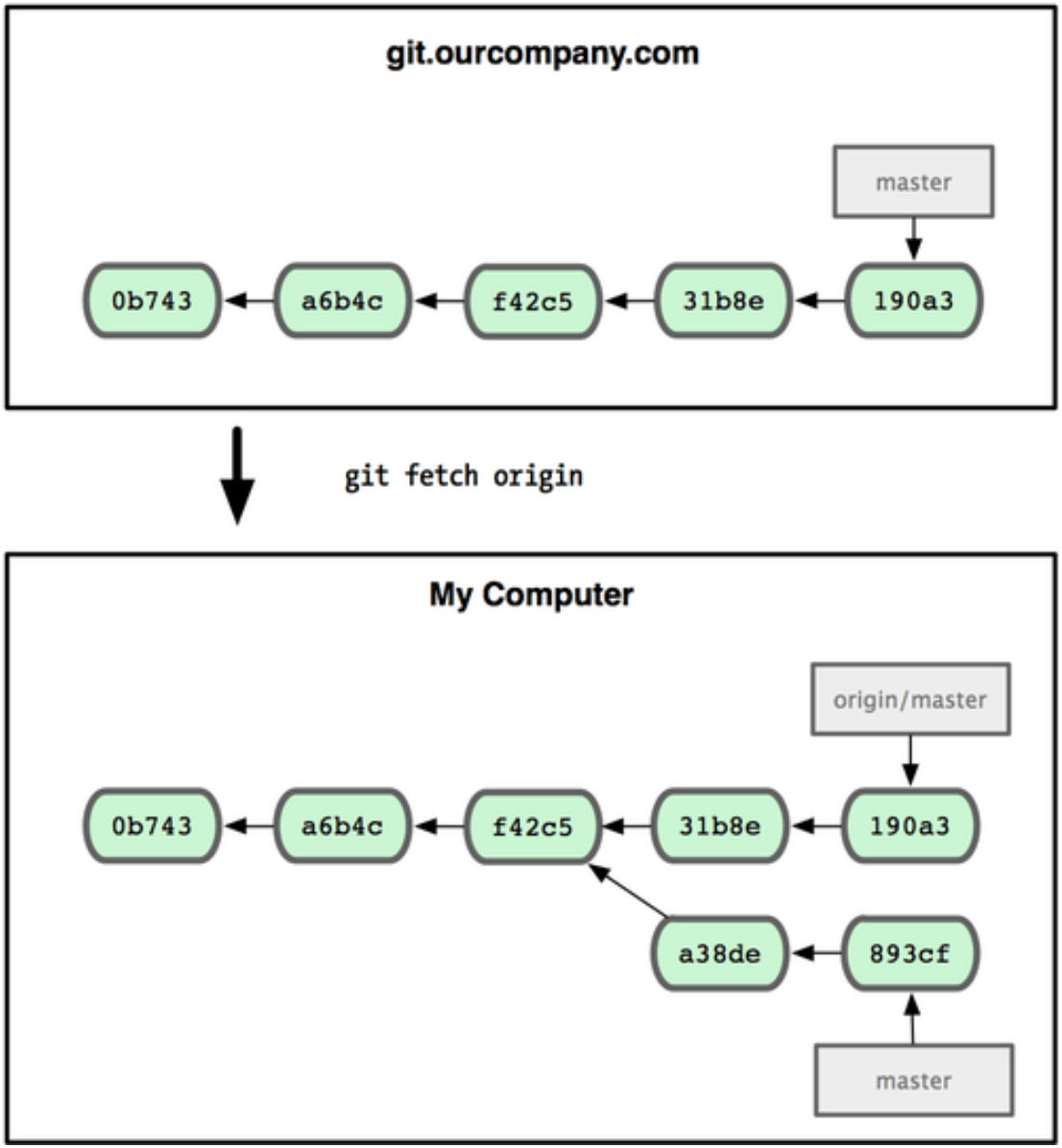
- `git fetch [remote-name]`:
  - Recupera todos os dados remotos que não se tem ainda
  - Referencia todos os branches remotos
- `git fetch origin`:
  - Verifica qual servidor `origin` aponta
  - Obtém todos os dados que o repositório local ainda não tem, desde que foi feito `git clone` ou `git fetch`
  - Atualiza o banco de dados local
  - Move o `origin/master` para a posição mais recente e atualizada
  - Não faz `merge` automático
  - As modificações locais ficam inalteradas até que se faça o `merge` manual

git fetch


git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
Someone else pushes

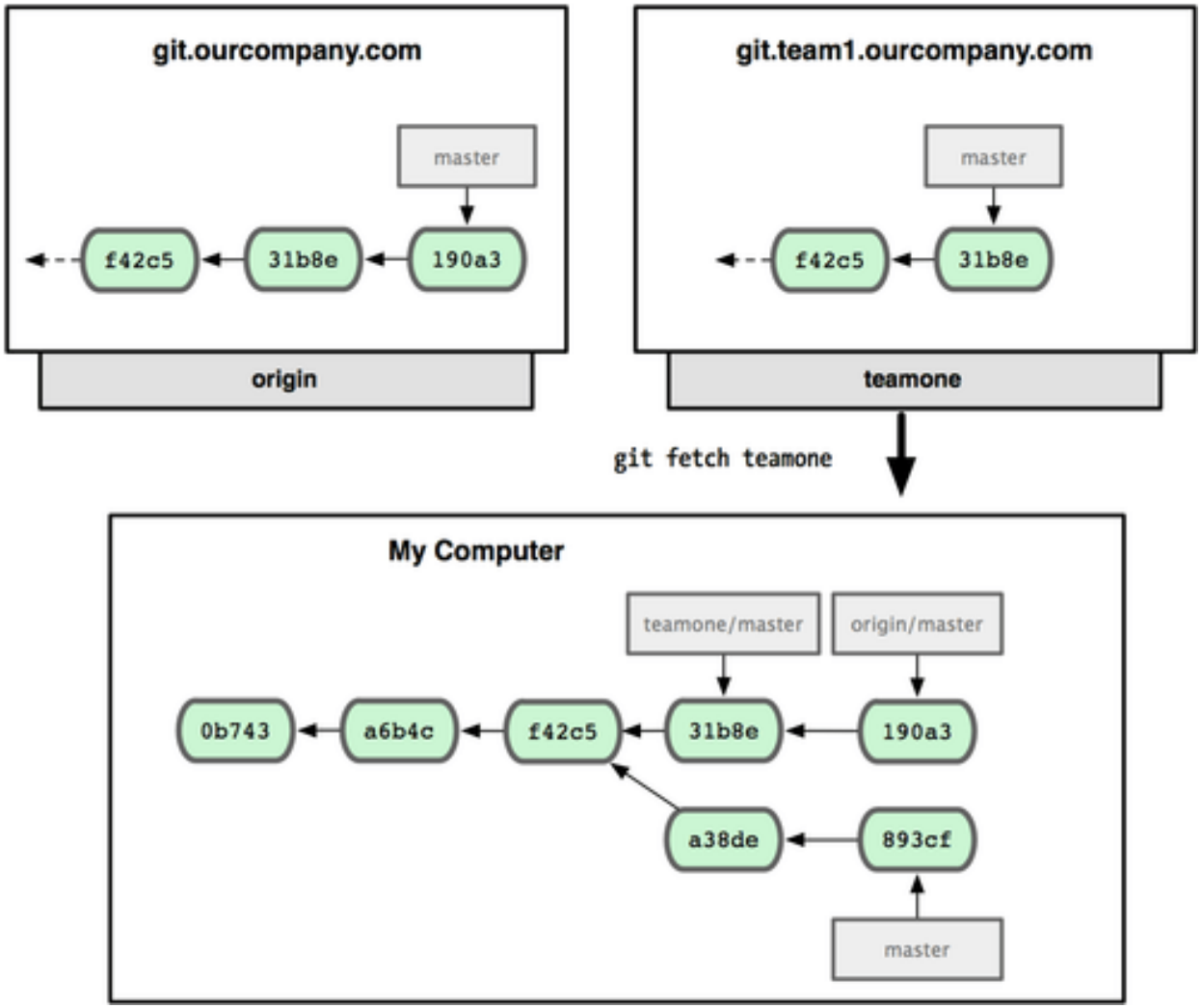

My Computer
origin/master
0b743
a6b4c
f42c5
a38de
893cf
master

git fetch


git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
git fetch origin
My Computer
origin/master
0b743
a6b4c
f42c5
31b8e
190a3
a38de
893cf
master

git fetch

- Sincronizando com o outro servidor remoto:
- `git fetch teamone`



`git pull`

- `git pull`: Recupera (`fetch`) os dados do servidor que seu repositório foi clonado e tenta automaticamente realizar a fusão (`merge`) com trabalho local

git push

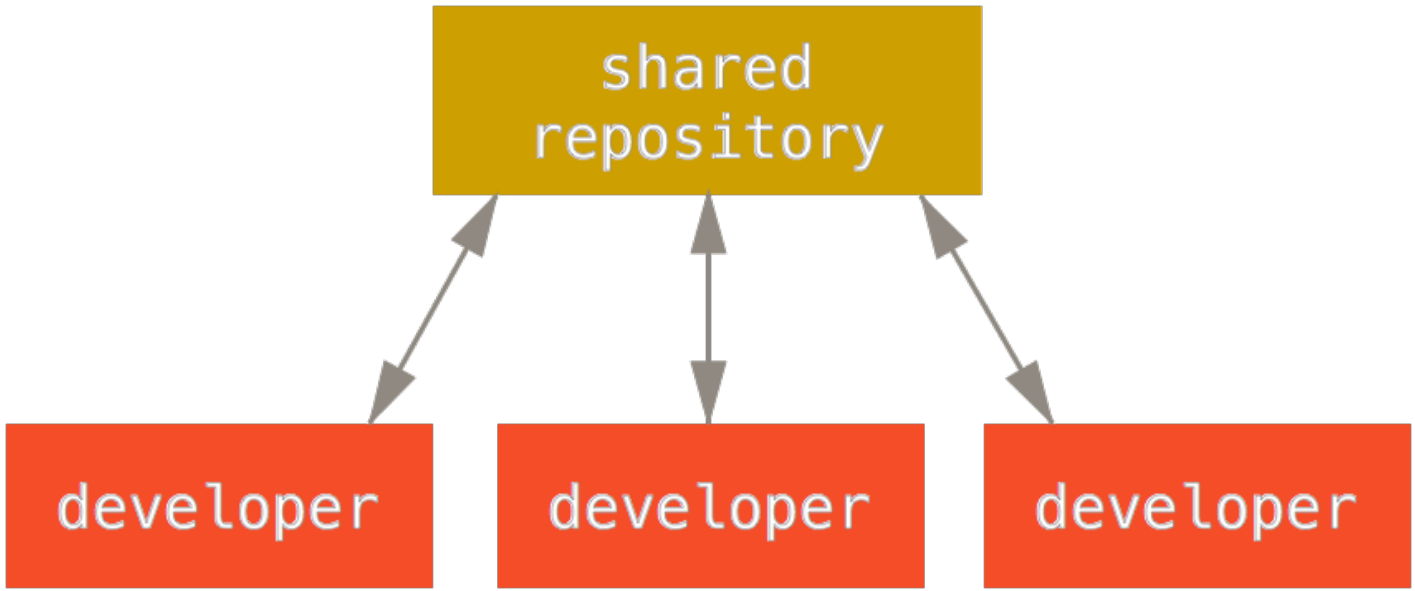
- Enviando (Pushing): quando se quer compartilhar um branch enviando-o para um servidor remoto
- `git push [remote-name] [branch-name]`
- `git push origin master`
- Possui restrições:
  - É preciso ter acesso de escrita no repositório remoto
  - Ninguém realizou um `git push` antes da sua tentativa
- Exercício:
  - Crie um branch local chamado `teste-branch-remoto`
  - Realize dois commits
  - Entregue para o servidor: `git push origin teste-branch-remoto`
  - Visualize os resultados em modo gráfico
  - No outro repositório local execute: `git remote show origin`
- `git push origin --delete serverfix`: deleta um branch remoto

Fluxo de Trabalho Distribuído

- Fluxo de Trabalho Centralizado
- Fluxo de Trabalho com Gerente de Integração
- Fluxo de Trabalho Ditador-Tenentes

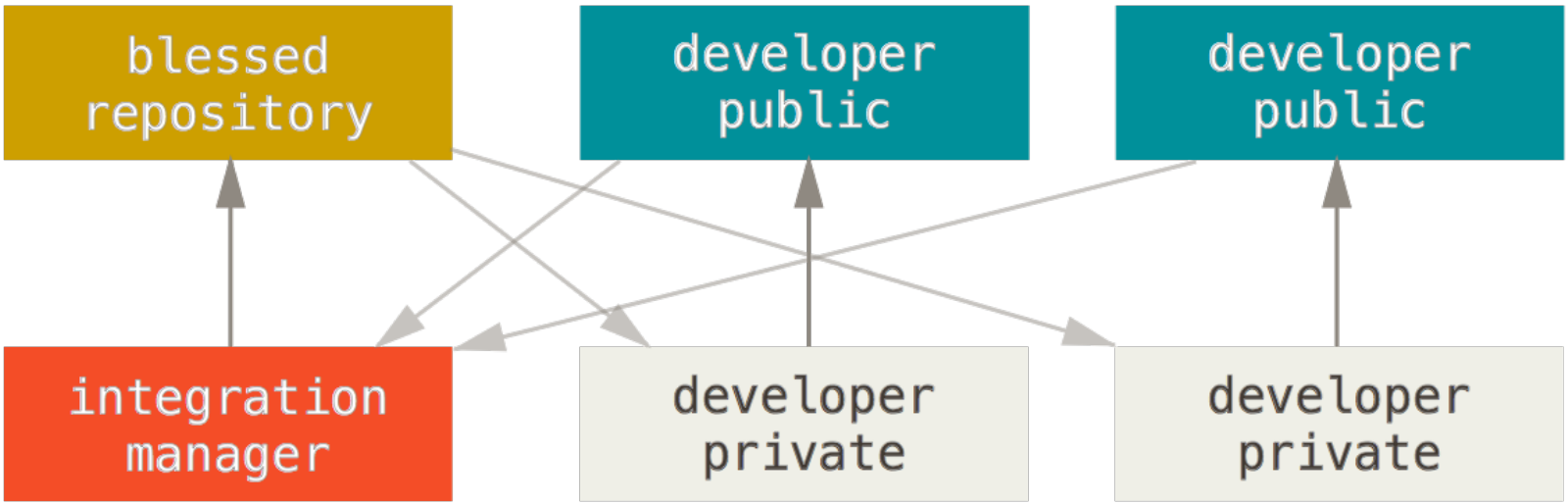
Fluxo de Trabalho Centralizado

- Estilo Subversion



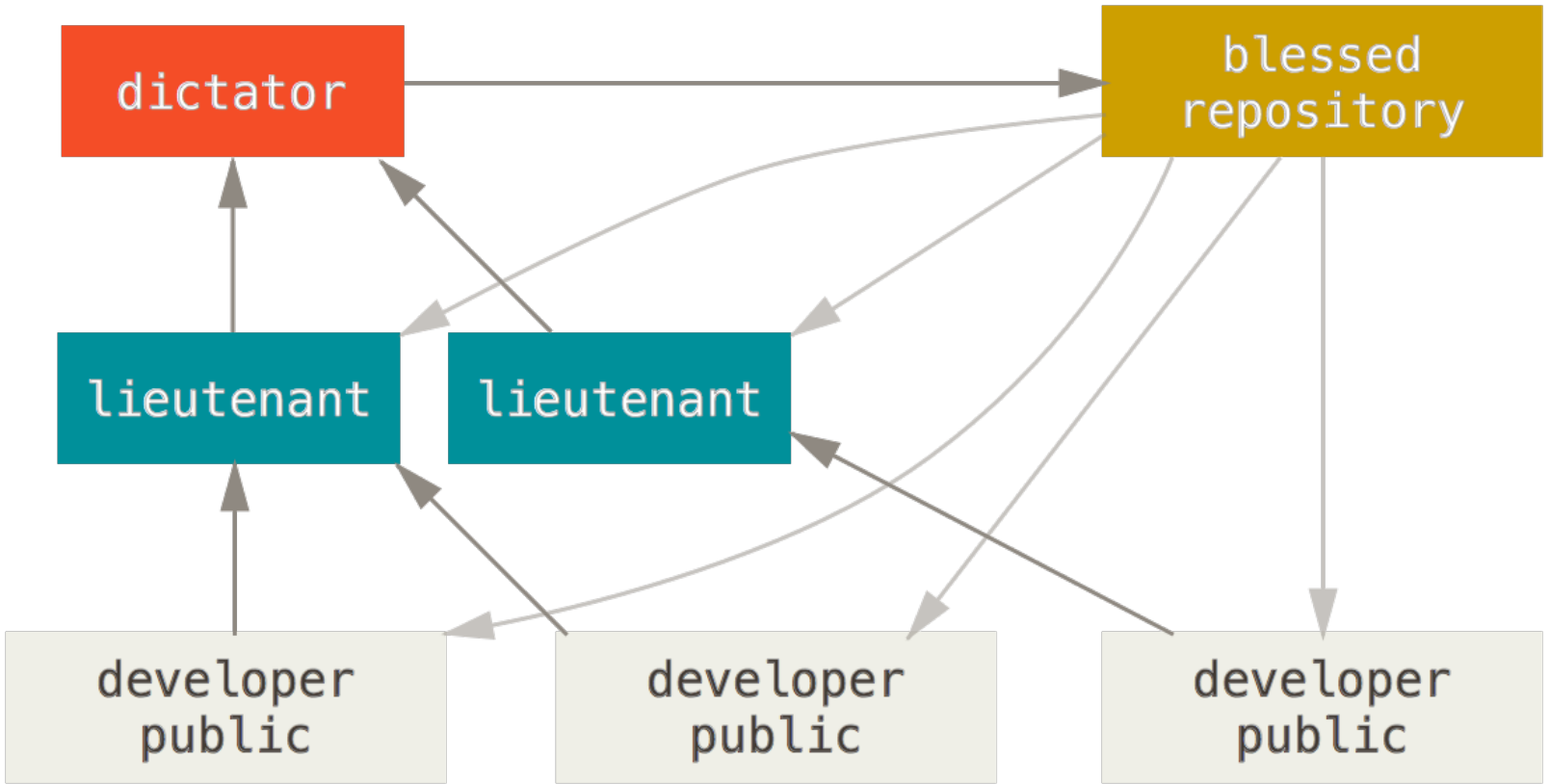
Fluxo de Trabalho com Gerente de Integração

- Estilo GitHub



Fluxo de Trabalho Ditador-Tenentes

- Estilo Kernel Linux



Fluxo de Trabalho ao Estilo Subversion

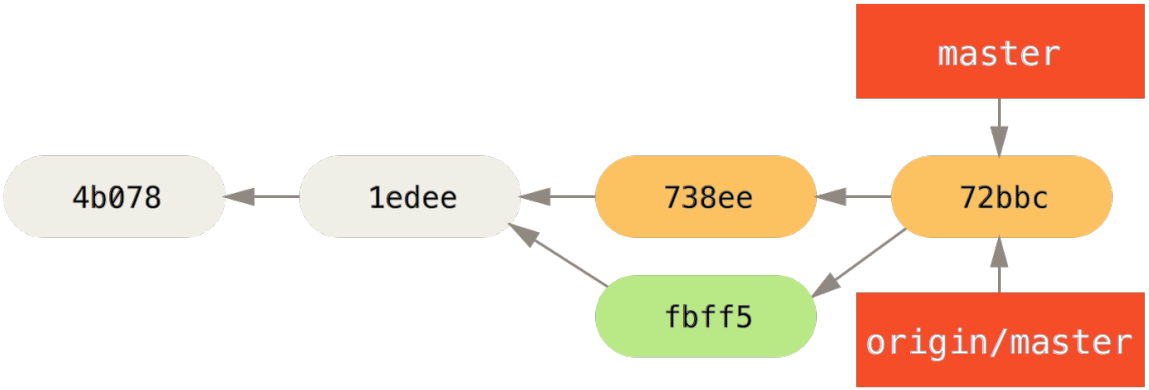


Figure: Histórico do Usuário A

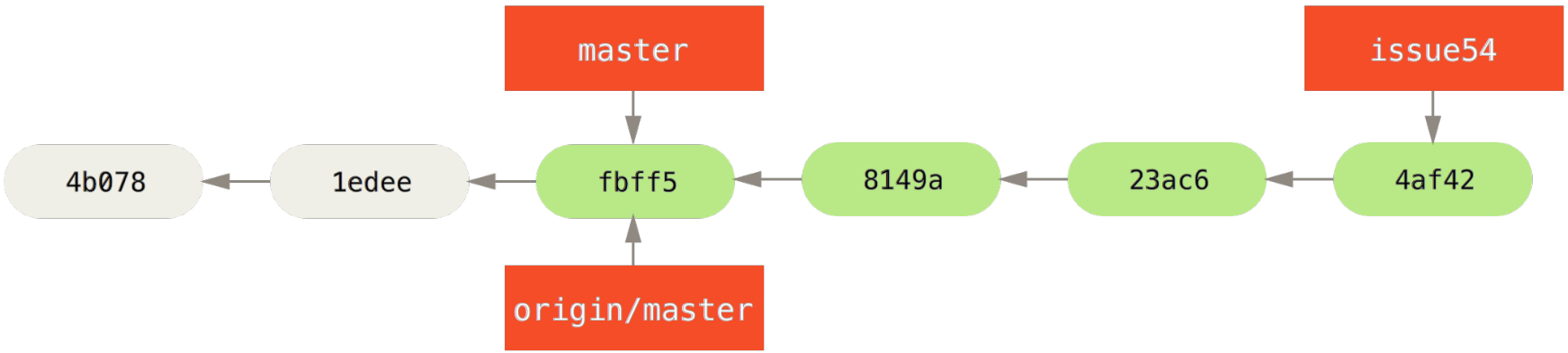


Figure: Histórico do Usuário B

Fluxo de Trabalho ao Estilo Subversion

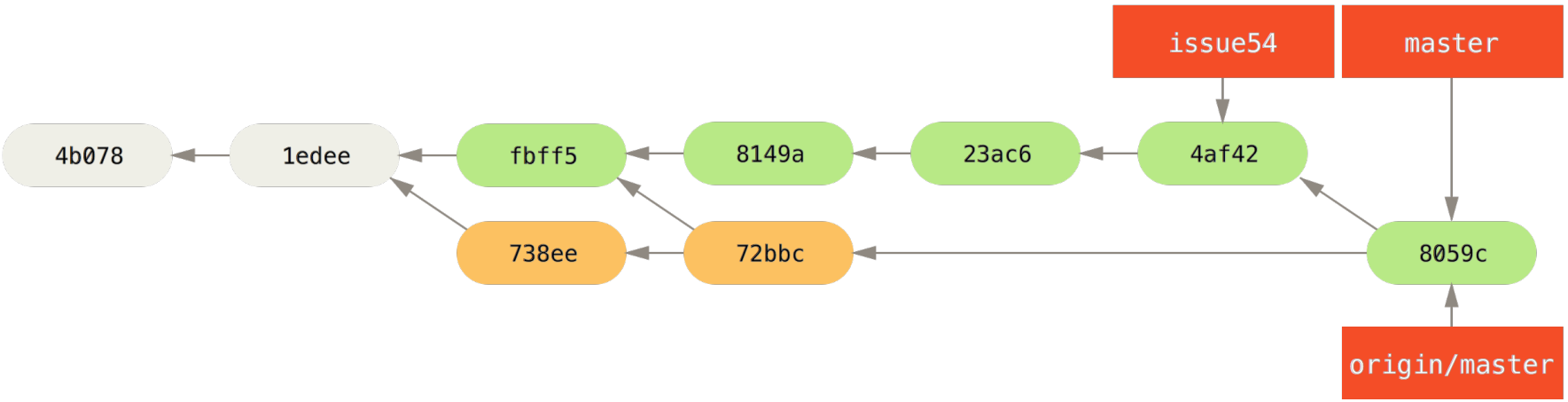
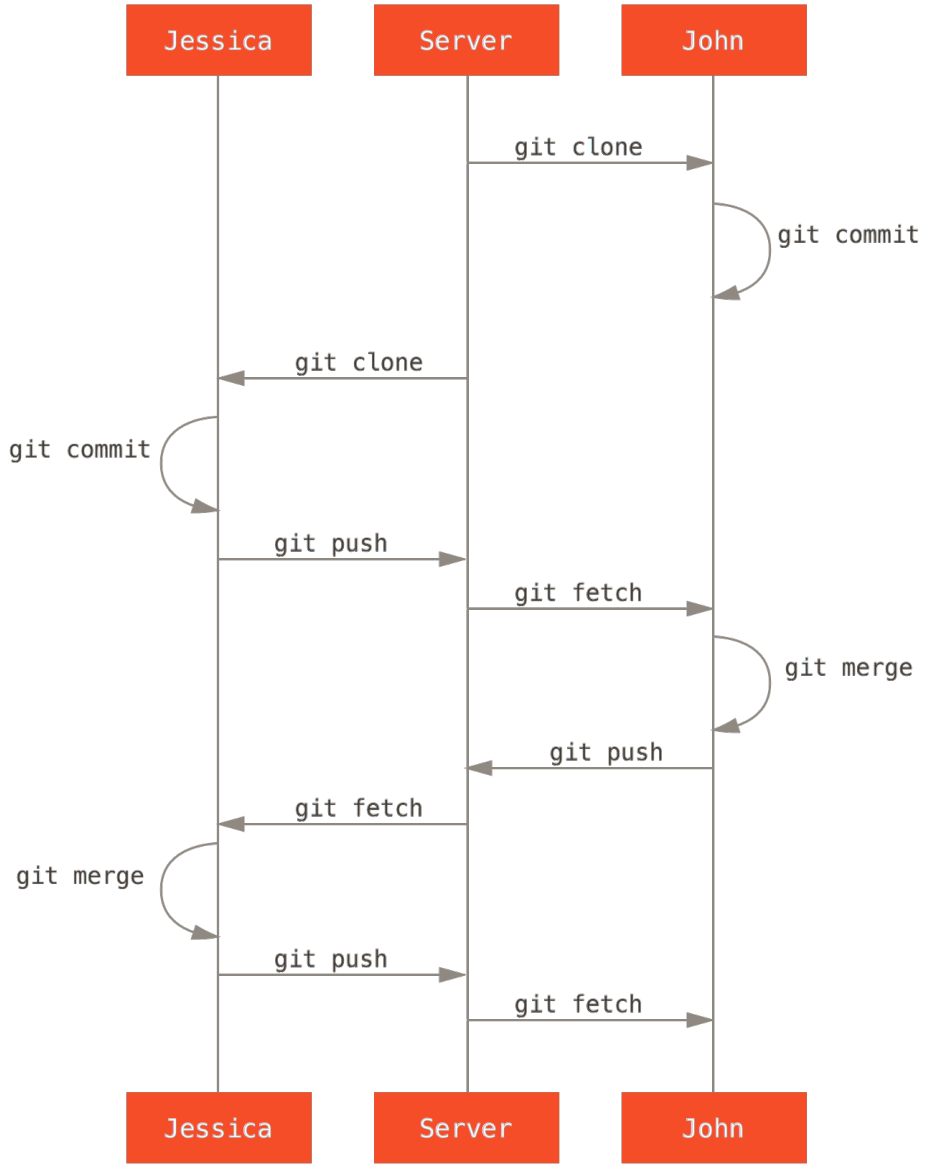


Figure: Histórico do Usuário B após a fusão

Fluxo de Trabalho ao Estilo Subversion



Referências

- Pro Git - `http://git-scm.com/book`
- Pragmatic Version Control Using Git (Pragmatic Starter Kit)
- Version Control with Git: Powerful tools and techniques for collaborative software development