GIT

Aplicado ao Github

O que é Git?

- Sistema de controle de versão de arquivos
  - Controle de Versão Locais
  - Controle de Versão Centralizados
  - Controle de Versão Distribuídos

O que é Git?

Observa-se o controle de versão em um computador local, no qual as versões são patches das diferenças entre os arquivos na linha do tempo

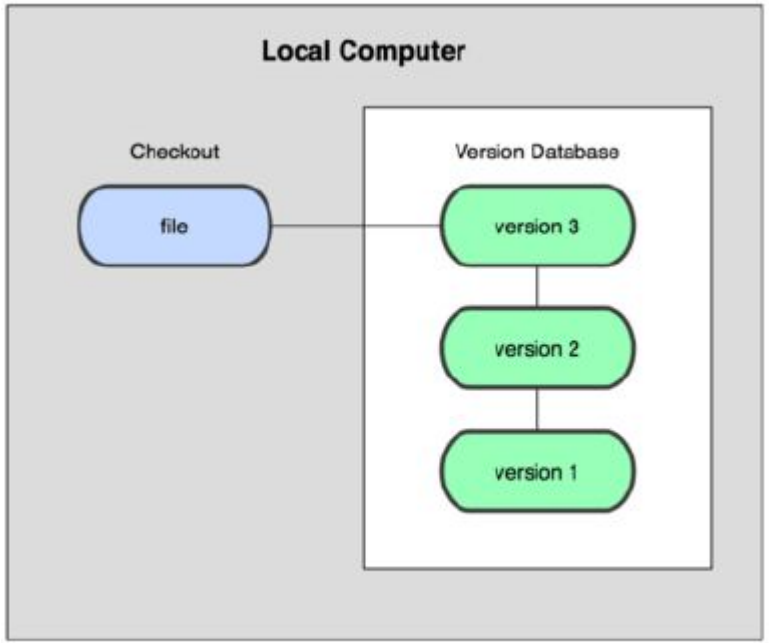


Figura 1-1. Diagrama de controle de versão local.

O que é Git?

Versão centralizado permite que mais desenvolvedores tenham acesso aos patches e os modifiquem.

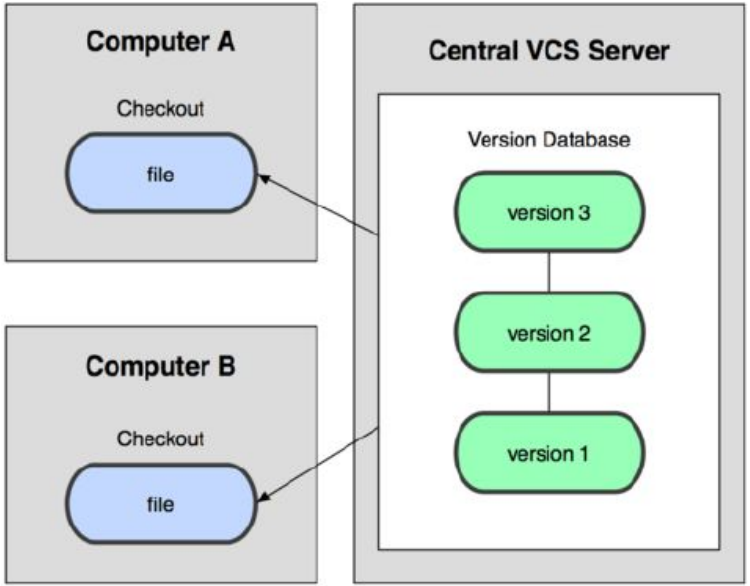


Figura 1-2. Diagrama de Controle de Versão Centralizado.

O que é Git?

Cada contribuinte possuem cópias completas do repositório, favorecendo que informações sejam perdidas tanto no servidor como localmente.

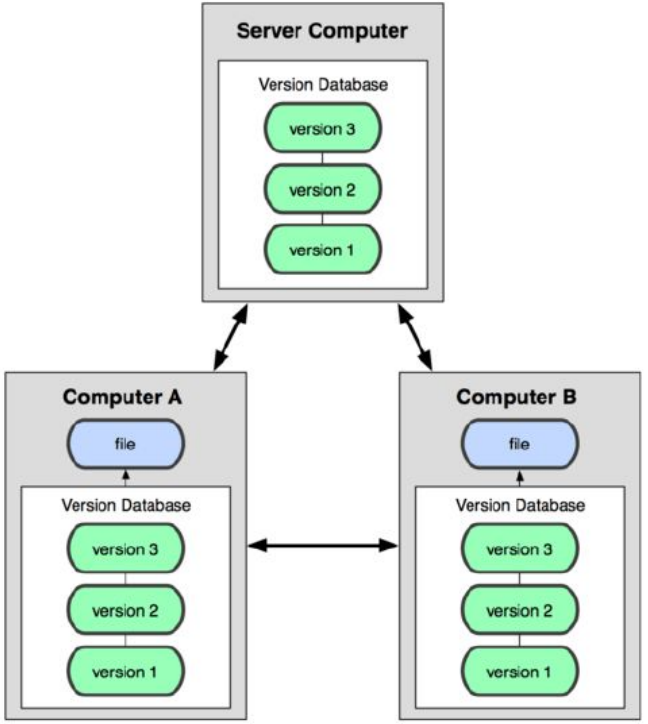


Figura 1-3. Diagrama de Controle de Versão Distribuído.

Alguns benefícios

- Quase todas operações são locais
- Integridade
  - Verificação via checksum (hash SHA-1)
- Geralmente só adiciona dados
  - Tudo é reversível depois de um commit
- Os três estados
  - Consolidado (committed)
  - Modificado (modified)
  - Preparado (staged)

Os três estados

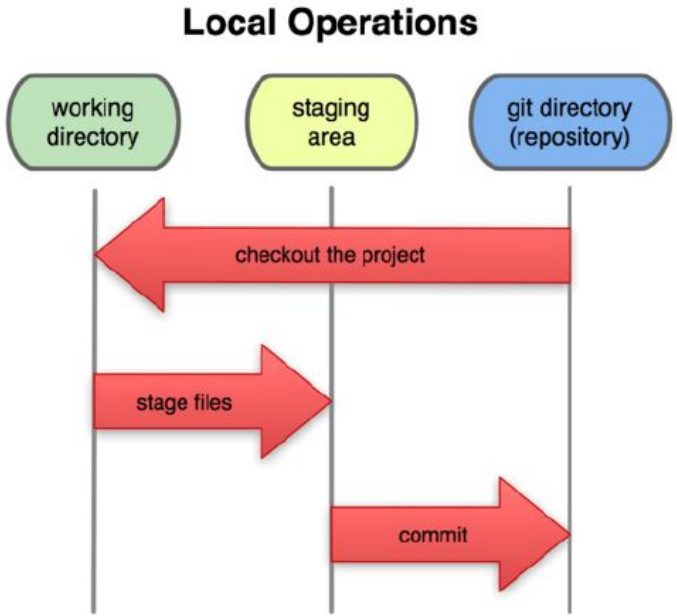


Figura 1-6. Diretório de trabalho, área de preparação, e o diretório do Git.

Primeiros passos

- Instalação do Git no Linux (Ubuntu)
  - $ sudo apt-get install git
- Configuração inicial do Git
  - Todos usuários do sistema: $ git config --system
  - Somente usuário específico: $ git config --global
  - Diretório git: $ .git/config
- Identidade
  - $ git config --global user.name “SeuNome”
  - $ git config --global user.email SeuEmail

Primeiros passos

- Obtendo ajuda
  - $ git help <verb>
  - $ git <verb> --help
  - $ man git-<verb>

E o que é o Github?

- Serviço de Web Hosting Compartilhado
- Rede social de software

Git Essencial

- Obtendo um repositório Git
  - Inicializando repositório existente: $ git init
  - Clonando repositório existente:
    - $ git clone URL
    - $ git clone URL NomeDiretórioDestino

Alterações no Repositório

Arquivos clonados são considerados inalterados e são monitorados pois estavam no último snapshot, após modificações passam a ser não monitorados até o próximo commit.

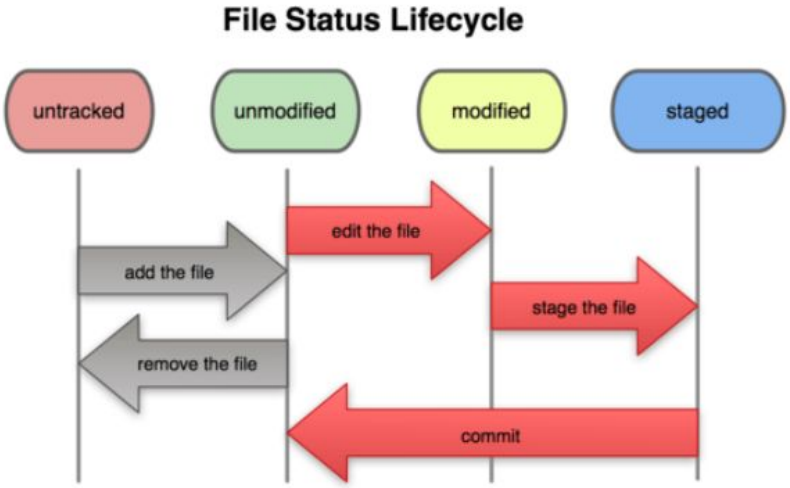


Figura 2-1. O ciclo de vida dos status de seus arquivos.

Git Essencial

- Status dos arquivos
  - $ git status
- Monitorando novos arquivos (criação do ‘README.md’)
  - $ git add README.md

Git Essencial

- Ignorando arquivos
  - $ cat .gitignore
  - *. [oa] #Ignora todos arquivos finalizados com .o ou .a
  - *~ #Ignora todos arquivos que terminam com ~
  - !lib.a #Rastreia o arquivo lib.a
  - /TODO #Ignora apenas o arquivo TODO na raiz
  - build/ #Ignora todos os arquivo no diretório build/
  - doc/*.txt #Ignora todos arquivo .txt no diretório, mas não no subdiretório

Git Essencial

- Visualizando mudanças não selecionadas
  - $ git diff #Compara o que está no seu diretório com sua área de seleção
- Visualizando mudanças selecionadas
  - $ git diff --cached ou $ git diff --staged #Compara o que está na sua área de seleção com do último commit

Git Essencial

- Realizando commit das mudanças
  - $ git commit #Realiza o commit contendo a mensagem da última saída do comando git status
  - $ git commit -v #Realiza o commit contendo a mensagem da diferença (diff) da sua mudança
  - $ git commit -m SuaMensagem #Realiza o commit contendo a sua mensagem
  - $ git commit -a #Seleciona automaticamente os arquivos monitorados e realiza o commit

Git Essencial

- Removendo arquivos
  - $ rm NomeDoArquivo #Remove da área de seleção
  - $ git rm NomeDoArquivo #Remove arquivo e deixa de monitorá-lo
  - $ git rm -f NomeDoArquivo #Força remoção do arquivo, previne remoções acidentais
  - $ git rm --cached NomeDoArquivo #Mantém arquivo no diretório e deixa de monitorá-lo
  - $ git rm log/*.log #O git cria sua própria expansão, \ necessário para informar terminações

Git Essencial

- Movendo arquivos
  $ git mv arquivo_origem arquivo_destino #É mais utilizado para renomear arquivo

Git Essencial

- Histórico de commits
  - $ git log #Exibi todos commits em ordem reversa
  - $ git log -p #Exibi todos commits com diff
  - $ git log -p -2 #Exibi os dois últimos commits com diff
  - $ git log --stat #Exibi os commits resumidamente
  - $ git log --pretty=oneline #Exibi cada commit em uma linha
  - $ git log --pretty=format: “%h - %an, %ar : %s” #Exibi o commit formatado a escolha do usuário

Git Essencial

Opção	Descrição de Saída
%H	Hash do commit
%h	Hash do commit abreviado
%T	Árvore hash
%t	Árvore hash abreviada
%P	Hashes pais
%p	Hashes pais abreviados
%an	Nome do autor
%ae	Email do autor
%ad	Data do autor (formato respeita a opção -date=)
%ar	Data do autor, relativa
%cn	Nome do committer
%ce	Email do committer
%cd	Data do committer
%cr	Data do committer, relativa
%s	Assunto

Git Essencial

Opção	Descrição
-p	Mostra o patch introduzido com cada commit.
–stat	Mostra estatísticas de arquivos modificados em cada commit.
–shortstat	Mostra somente as linhas modificadas/inseridas/excluídas do comando –stat.
–name-only	Mostra a lista de arquivos modificados depois das informações do commit.
–name-status	Mostra a lista de arquivos afetados com informações sobre adição/modificação/exclusão dos mesmos.
–abbrev-commit	Mostra somente os primeiros caracteres do checksum SHA-1 em vez de todos os 40.
–relative-date	Mostra a data em um formato relativo (por exemplo, “2 semanas atrás”) em vez de usar o formato de data completo.
–graph	Mostra um gráfico ASCII do branch e histórico de merges ao lado da saída de log.
–pretty	Mostra os commits em um formato alternativo. Opções incluem oneline, short, full, fuller, e format (onde você especifica seu próprio formato).

Git Essencial

Opção	Descrição
-(n)	Mostra somente os últimos n commits.
–since, –after	Limita aos commits feitos depois da data especificada.
–until, –before	Limita aos commits feitos antes da data especificada.
–author	Somente mostra commits que o autor casa com a string especificada.
–committer	Somente mostra os commits em que a entrada do commiter bate com a string especificada.

Git Essencial

- Modificando o último commit
  - $ git commit --amend #Realiza novo commit da área de seleção substituindo seu último commit
- Retirando arquivo da área de seleção
  - $ git reset HEAD NomeDoArquivo
- Desfazendo um arquivo modificado
  - $ git checkout -- NomeDoArquivo

Git Essencial

- Exibindo repositórios remotos
  - $ git remote
  - $ git remote -v #Exibi as URL de todos repositórios remotos

Git Essencial

- Adicionando repositórios remotos
  - $ git remote add [nomecurto] [url]
  - $ git fetch [nomecurtoescolhido] #Pega todos arquivos que você ainda não possui localmente
  - O comando $ git clone gera um repositório remoto automaticamente com nome **origin**

Git Essencial

- Fazendo o **fetch** e **pull** de seus remotos
  - $ git fetch [nome-remoto]
  - $ git pull [nome-remoto] #Realiza fetch e o merge automaticamente de um branch remoto para o seu atual

Git Essencial

- Pushing para seus remotos
  - $ git push [nome-remoto] [branch] #Se outra pessoa realizou push no mesmo repositório antes, necessário realizar um pull, incorpora aos seus arquivo e um push
- Inspecionando um remoto
  - $ git remote show [nome-remoto]

Git Essencial

- Renomeando remotos
  - $ git remote rename [nome-remoto] [novonome-remoto]
- Removendo remotos
  - $ git remote rm [nome-remoto]

Git Essencial

- Pseudônimos no git
  - $ git config --global alias.co checkout
  - $ git config --global alias.br branch
  - $ git config --global alias.ci commit
  - $ git config --global alias.st status

Ramificação (Branching)

Blob é uma versão do arquivo armazenada no repositório git **Branch** é um ponteiro móvel que aponta para o último commit

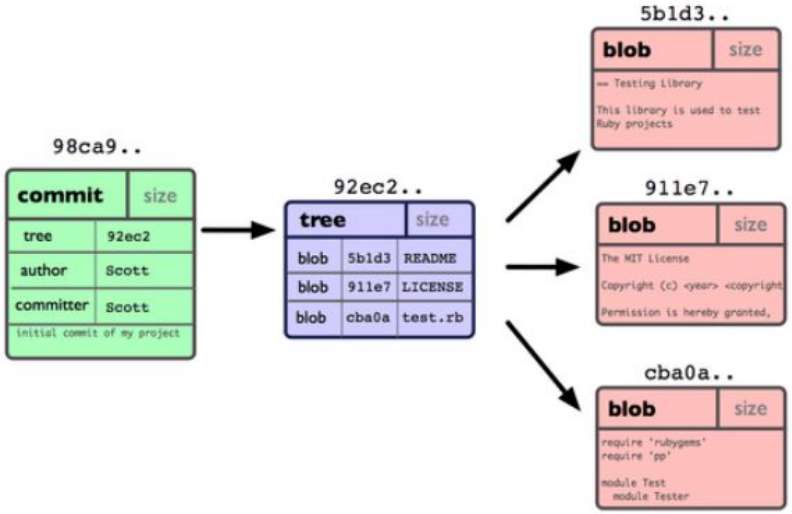


Figura 3-1. Dados de um repositório com um único commit.

Ramificação (Branching)

O **branch** padrão é o **master**

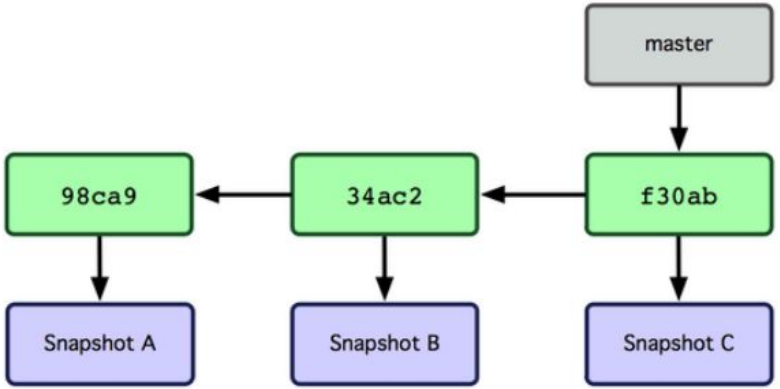


Figura 3-3. Branch apontando para o histórico de commits.

Ramificação (Branching)

Criando novo **branch**

$ git branch testing #testing é o nome do branch

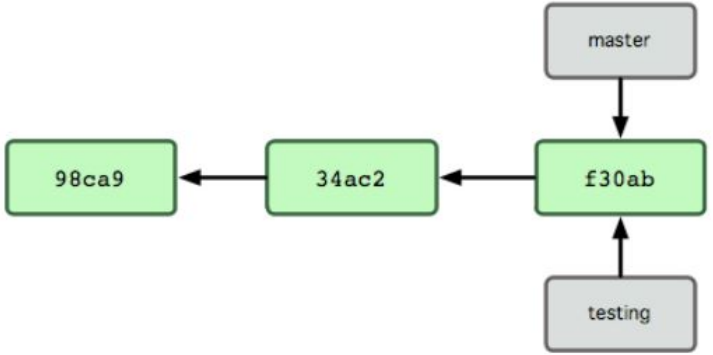


Figura 3-4. Múltiplos branches apontando para o histórico de commits.

Ramificação (Branching)

HEAD é um ponteiro especial do git que armazena o branch que você está

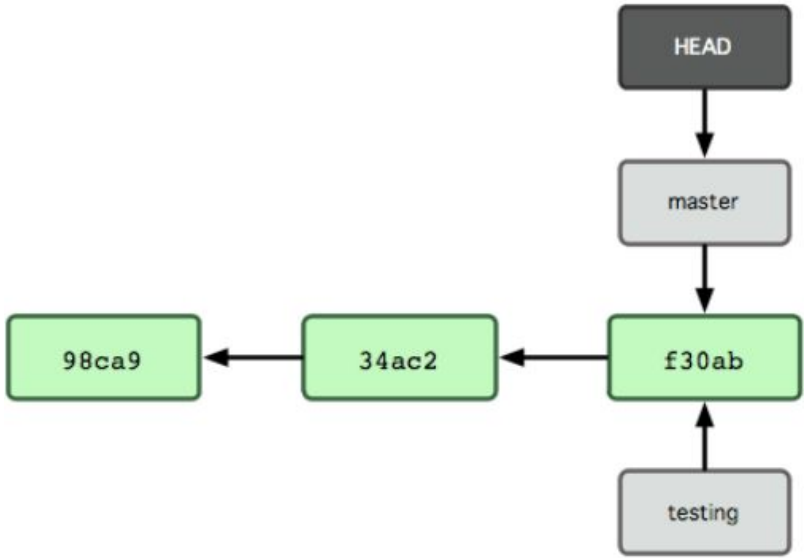


Figura 3-5. HEAD apontando para o branch em que você está.

Ramificação (Branching)

Mudando de **branch**

$ git checkout testing

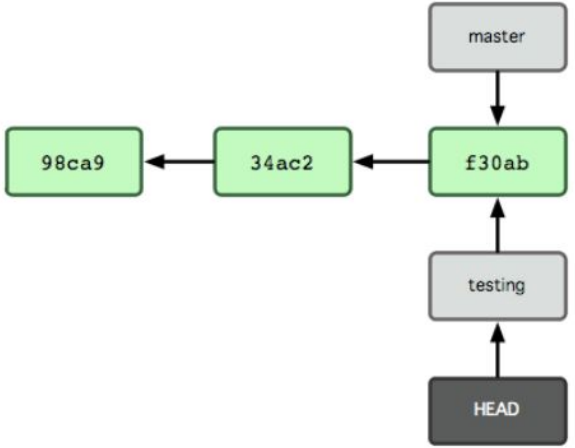


Figura 3-6. O HEAD aponta para outro branch quando você troca de branches.

Ramificação (Branching)

Realizando um novo **commit**

```
$ vim test.rb #Criei um arquivo test.rb
$ git commit -a m 'novo commit'
```

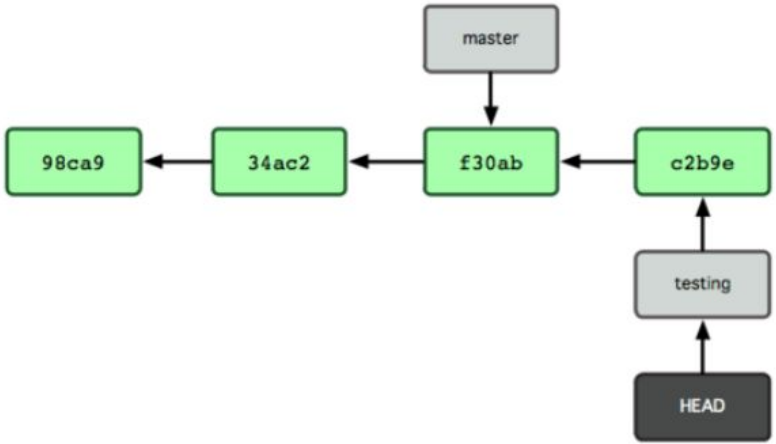


Figura 3-7. O branch para o qual HEAD aponta avança com cada commit.

Ramificação (Branching)

Alterando para master

$ git checkout master

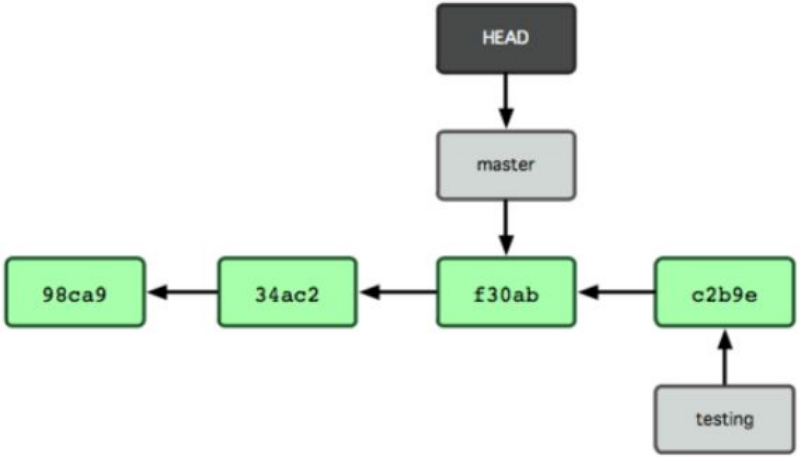
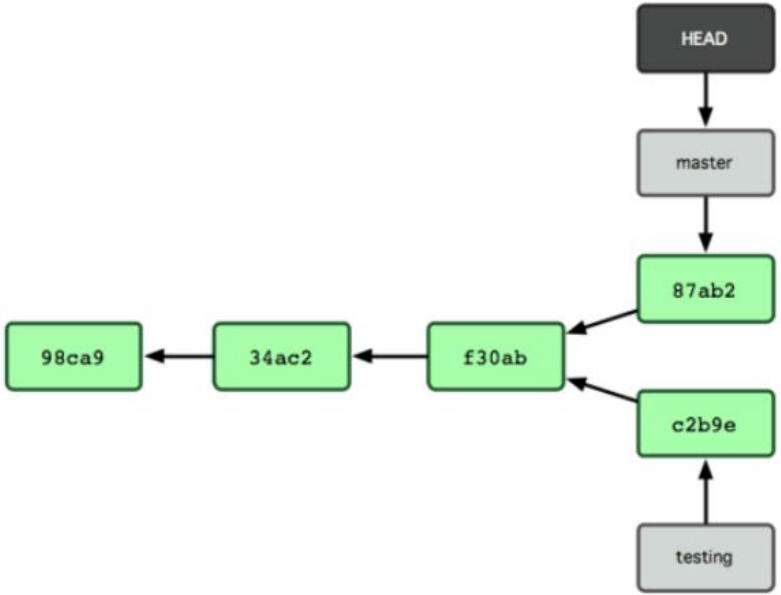


Figura 3-8. O HEAD se move para outro branch com um checkout.

Ramificação (Branching)

Realizando novo **commit**

```
$ vim test.rb
$ git commit -a -m ‘novo commit’
```



igura 3-9. O histórico dos branches diverge.

Ramificação (Branching)

Branch e **merge**

$ git checkout -b ‘hotfix’ #Cria nova branch e muda para ela automaticamente;

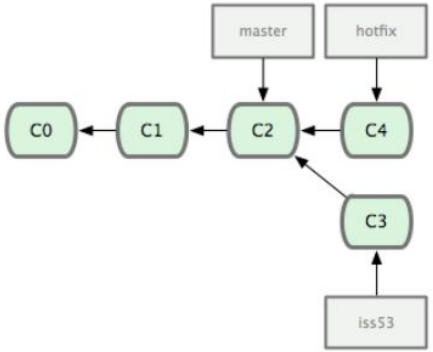


Figura 3-13. branch de correção (hotfix) baseado num ponto de seu branch master.

Ramificação (Branching)

$ git checkout master #Retorna para branch master
$ git merge hotfix #Modificação está agora no snapshot do branch master
$ git branch -d hotfix #Apaga a branch hotfix que não é mais necessária

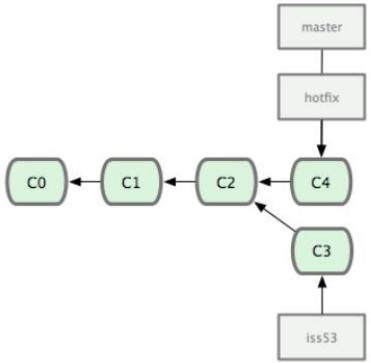


Figura 3-14. Depois do merge seu branch master aponta para o mesmo local que o branch hotfix.

Merge de três vias

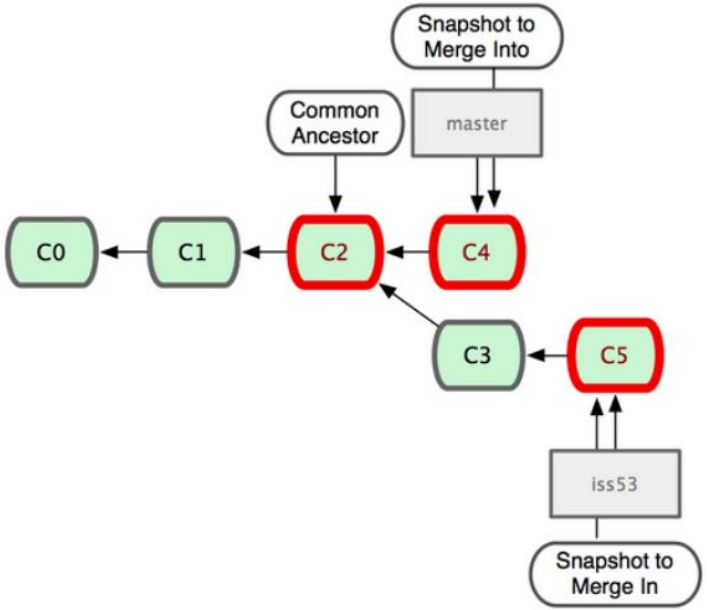


Figura 3-16. Git identifica automaticamente a melhor base ancestral comum para o merge do branch.

Merge de três vias

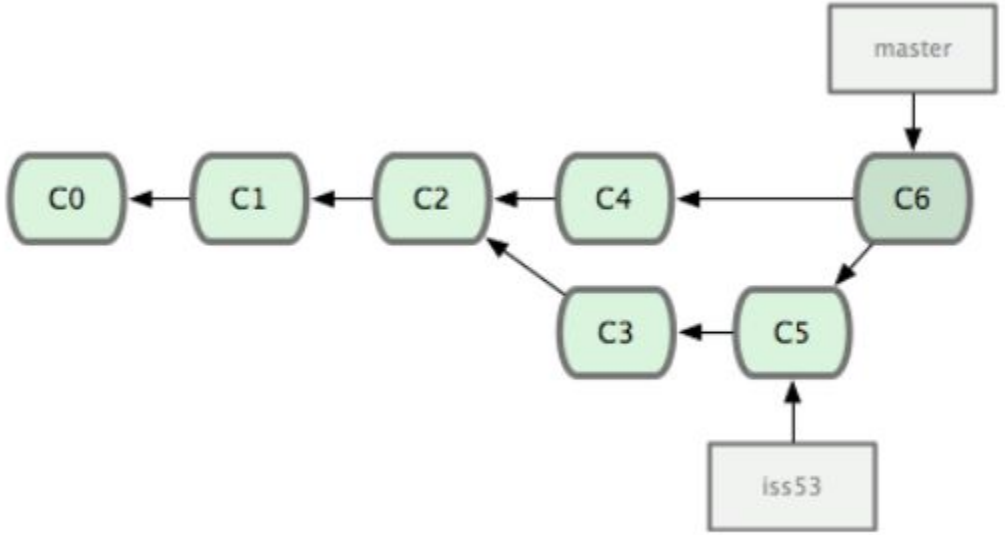


Figura 3-17. Git cria automaticamente um novo objeto commit que contém as modificações do merge.

Conflitos de merge básico

- Ocorre quando tenta realizar um merge de dois branches que possui uma alteração da mesma parte do arquivo
- Necessário um $ git status, o git te notificará o local do conflito com unmerge e uma mesclagem manual tem que ser feita para uma nova tentativa de **merge**

Gerenciamento de branches

$ git branch #Exibi todas suas branches
$ git branch -v #Exibi último commit de cada branch
$ git branch --merged #Exibi quais branches já foram mescladas na sua branch atual
$ git branch --no-merged #Exibi todos os branches que ainda não foram mesclados

- Forçando remoção de branch não mescladas
  - $ git branch -D

Fluxo de trabalho com branches

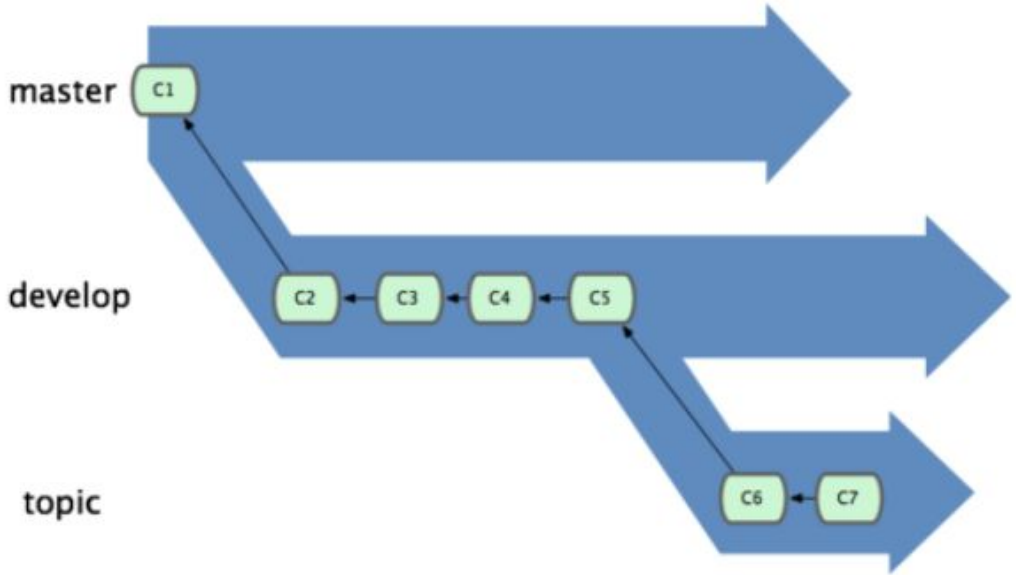


Figura 3-19. Pode ser mais útil pensar em seus branches como contêineres.

Branches remotos

- Segue o padrão (remote)/(branch)
- Servem para lembrá-lo onde estavam seus branches desde sua última conexão
- Necessário atualizar com fetch para sincronizá-lo com servidor

Fluxo de trabalho com branches

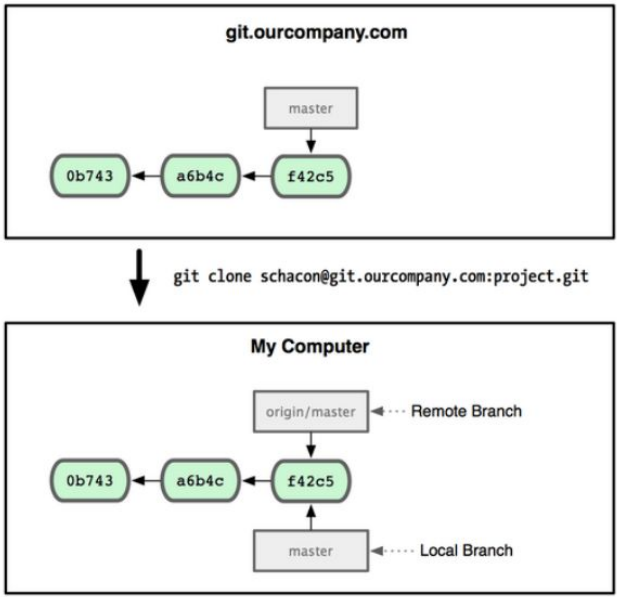


Figura 3-22. Um comando clone do Git dá a você seu próprio branch master e origin/master faz referência ao branch master original.

Fluxo de trabalho com branches

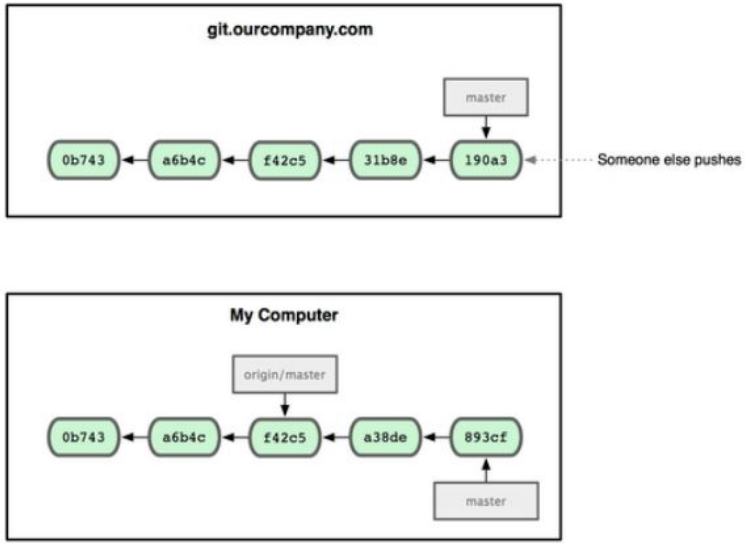


Figura 3-23. Ao trabalhar local e alguém enviar coisas para seu servidor remoto faz cada histórico avançar de forma diferente.

Fluxo de trabalho com branches

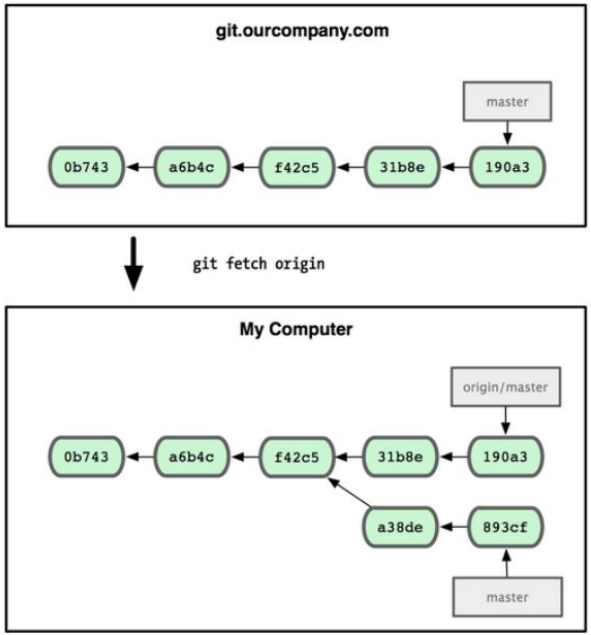


Figura 3-24. O comando git fetch atualiza suas referências remotas.

Ramificação (Branching)

Rebase (mesma finalidade do merge)
Monta um histórico mais limpo e linear

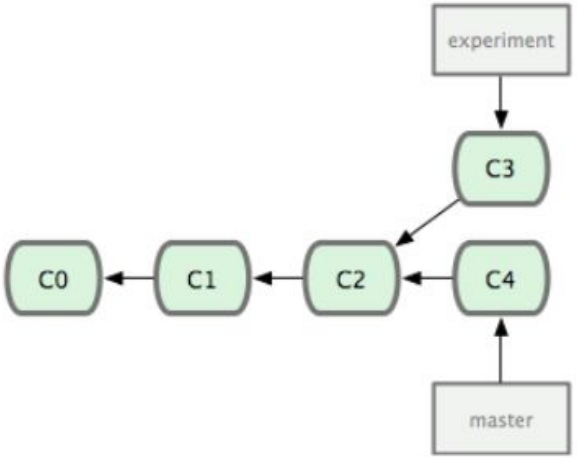


Figura 3-27. Divergência inicial no seu histórico de commits.

Ramificação (Branching)

```
$ git checkout experiment
$ git rebase master
```

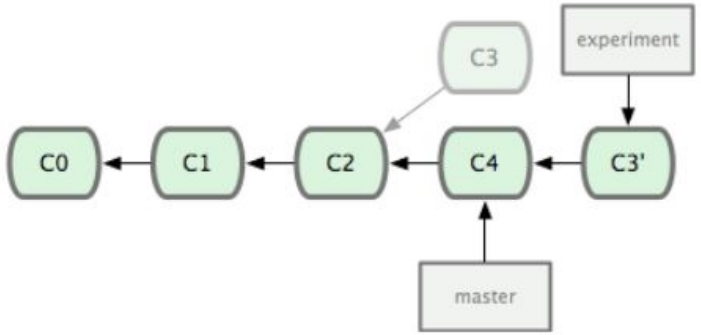


Figura 3-29. Fazendo o rebase em C4 de mudanças feitas em C3.

Ramificação (Branching)

```
$ git checkout master
$ git merge experiment
```

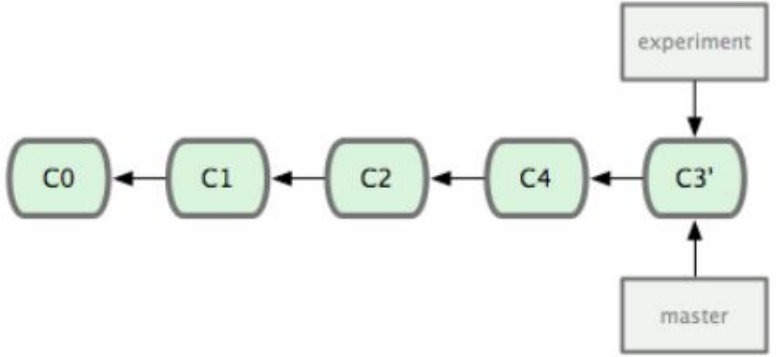


Figura 3-30. Fazendo um fast-forward no branch master.

Rebase

- Não faça rebase de commits que serão enviados para repositórios públicos
  - Rebase abandona commits existentes e cria novos similares
  - Pode ocasionar redundâncias, pois os dados são os mesmo porém o código hash SHA-1 são diferentes
  - Usuário terá que realizar novamente um merger e o git log estará confuso

Referências

CHACON, S. Pro Git. 2. ed. Apress, 2009. 288p.