

Tutorial Git (GNU/Linux)

Aplicado ao Github

Observações

- Durante o tutorial será comum a visualização de alguns casos citados abaixo:
 - Caracteres na cor azul: Comando já presente no terminal
 - Caracteres na cor verde: Linha de comando a ser digitada
 - Caracteres na cor magenta: Comando ao gosto do usuário
 - Caracteres na cor vermelha: Destaque aos pontos importantes
 - Caracteres na cor cinza: Trata-se de comentários e são ignorados no terminal

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?

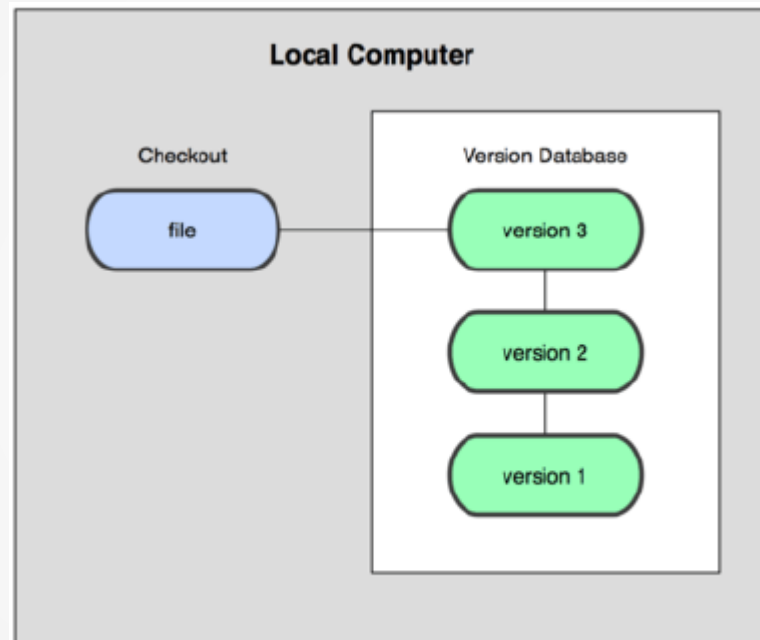


Figura 1-1. Diagrama de controle de versão local.

- 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

O que é Git?

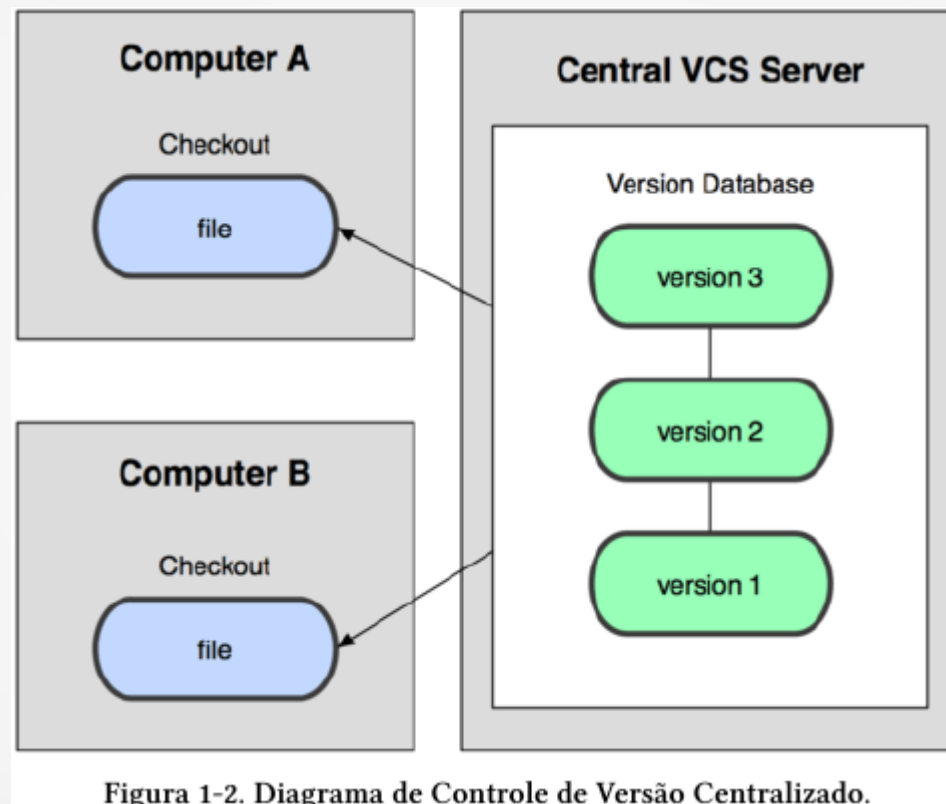


Figura 1-2. Diagrama de Controle de Versão Centralizado.

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

O que é Git?

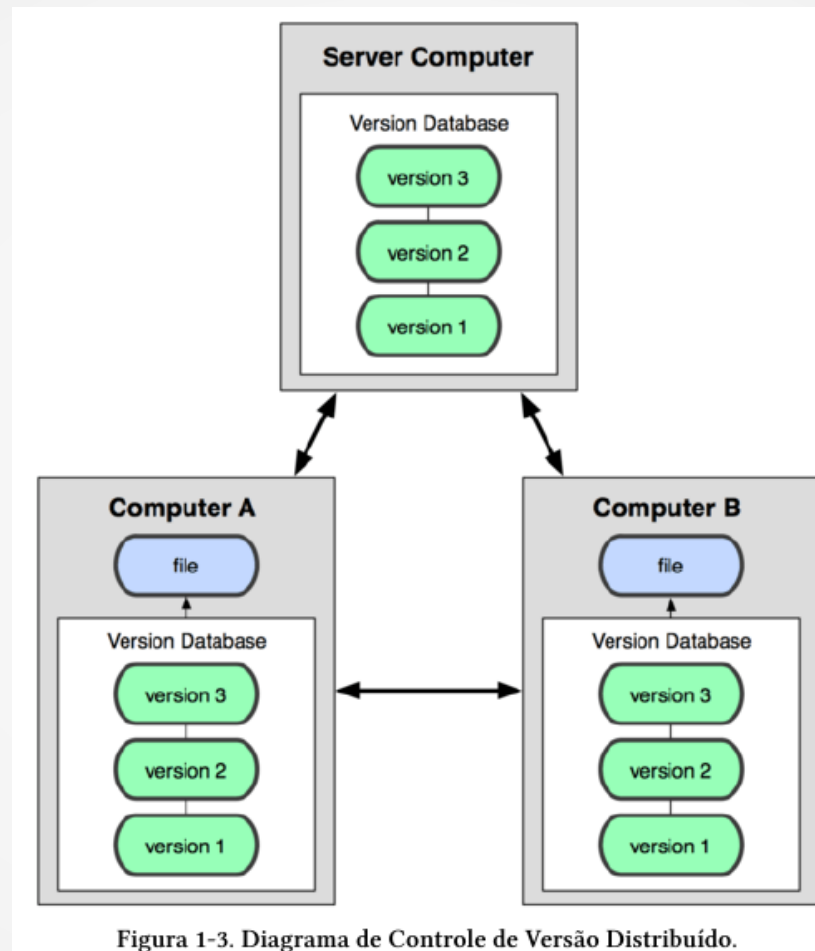


Figura 1-3. Diagrama de Controle de Versão Distribuído.

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

Git vs VCS similares

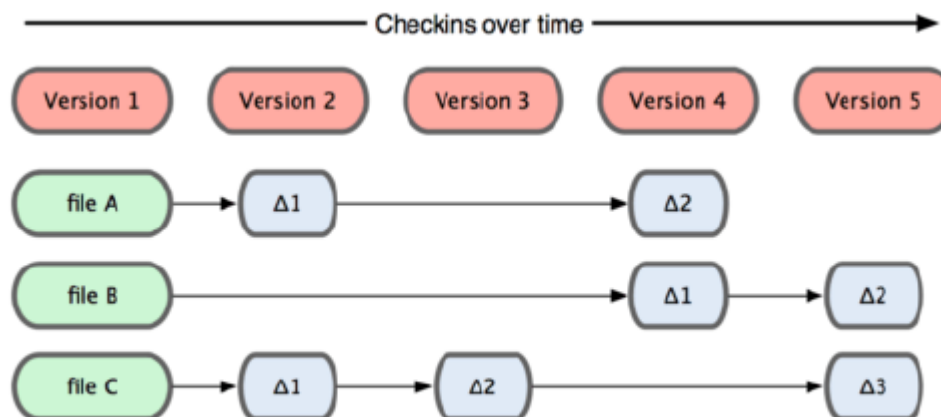


Figura 1-4. Outros sistemas costumam armazenar dados como mudanças em uma versão inicial de cada arquivo.

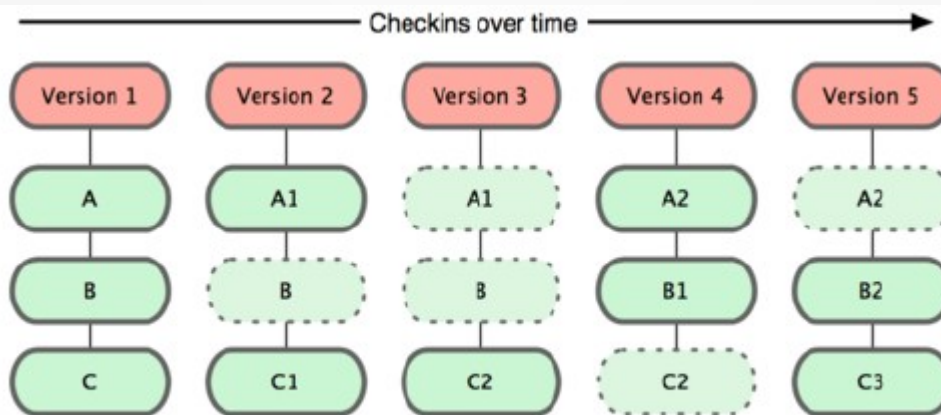


Figura 1-5. Git armazena dados como snapshots do projeto ao longo do tempo.

- O git se diferencia por realizar snapshots, cada vez que um commit é consolidado, é como se tirasse uma foto e armazenasse uma referência para essa captura

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*)

Workflow básico do Git

1. Modificação dos arquivos no diretório de trabalho
2. Seleção dos arquivos, adicionando snapshots deles para área de preparação
3. Realização do commit, que leva os arquivos modificados da área de preparação para o armazenamento permanente no diretório Git

Os três estados

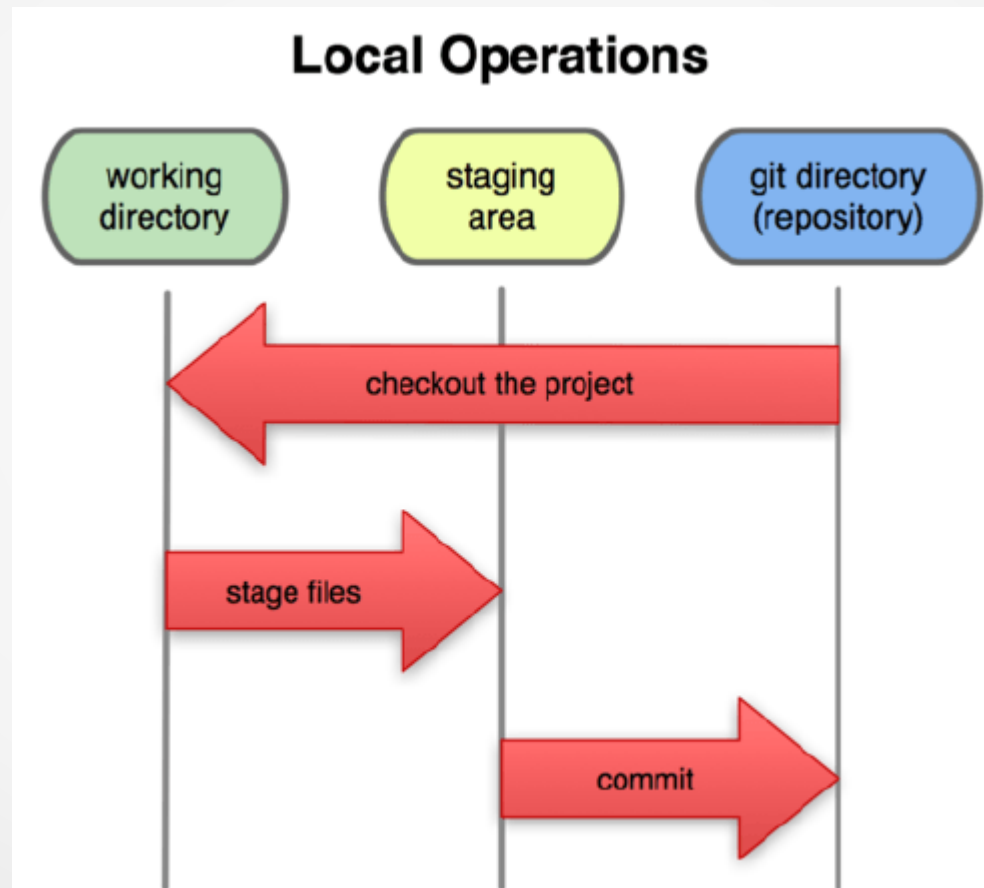


Figura 1-6. Diretório de trabalho, área de preparação, e o diretório do Git.

Primeiros passos

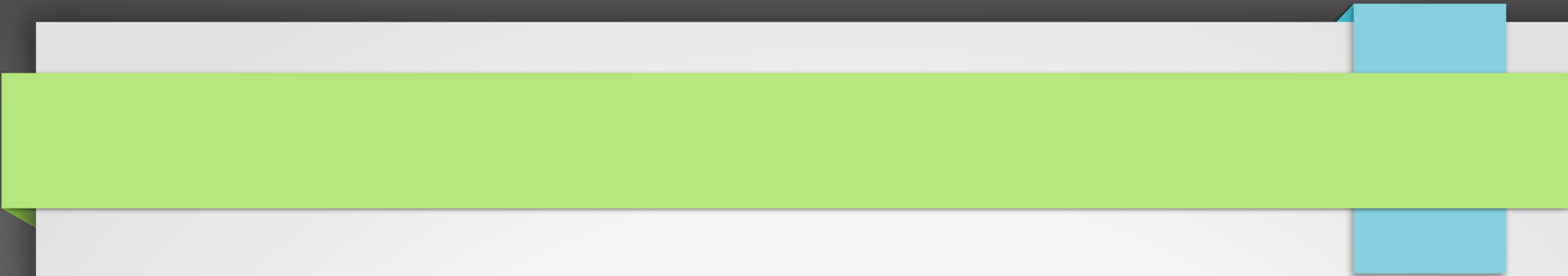
- Instalação do Git no Linux (Ubuntu)
 - `$ sudo apt-get install git`
- Configuração inicial do Git
 - Todos usuários do sistema: `$ git config --system`
 - Somente usuário específico: `$ git config --global`
 - Diretório git: `$.git/config`
- Identidade
 - `$ git config --global user.name "SeuNome"`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

Primeiros passos

- Editor (Git utiliza o padrão do sistema, no caso o Vi ou Vim)
 - `$ git config --global core.editor NomeDoEditor` #[emacs]
- Ferramenta de Diff (resolve conflitos de merge(fusão))
 - `$ git config --global merge.tool NomeDoDiff` #[vimdiff]
- Verificando suas configurações
 - `$ git config --list`
 - `$ git config user.name` #Exibi o nome na configuração

Primeiros passos

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



Antes de prosseguir, uma breve
abordagem do GitHub

O que é GitHub?

- Serviço de Web Hosting Compartilhado
- Rede social de software

GitHub

Acesse: <https://github.com>



Personal Open source Business Explore

Pricing Blog Support

Search GitHub

Sign in

Sign up

1

Join GitHub

The best way to design, build, and ship software.



Step 1:

Set up a personal account



Step 2:

Choose your plan

Create your personal account

Username

UserExemplo



This will be your username — you can enter your organization's username next.

Email Address

exemplo@exemplo.com



You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

••••••••



Use at least one lowercase letter, one numeral, and seven characters.

By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

2

3

Welcome to GitHub

You've taken your first step into a larger world, @UserExemplo.



Completed

Set up a personal account



Step 2:

Choose your plan



Step 3:

Tailor your experience

4

Choose your personal plan



Unlimited public repositories for free.



Unlimited private repositories for \$7/month. [\(view in BRL\)](#)

Don't worry, you can cancel or upgrade at any time.



Help me set up an organization next

Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.

[Learn more about organizations.](#)

Continue

5

Both plans include:

- ✓ Collaborative code review
- ✓ Issue tracking
- ✓ Open source community
- ✓ Unlimited public repositories
- ✓ Join any organization

GitHub

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @UserExemplo.



Completed
Set up a personal account



Step 2:
Choose your plan



Step 3:
Tailor your experience

How would you describe your level of programming experience?

☐ Very experienced

☐ Somewhat experienced

☐ Totally new to programming

What do you plan to use GitHub for? (check all that apply)

☐ Research

☐ Project Management

☐ Design

☐ School projects

☐ Development

☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a professional

☐ I'm a student

☐ I'm a hobbyist

☐ Other (please specify)

What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

Submit

[skip this step](#)

Esta etapa pode ser pulada

GitHub

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#)[Start a project](#)

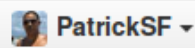
Criar novo projeto

GitHub

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

tutorialgit ✓

→ Nome do repositório

Great repository names are short and memorable. Need inspiration? How about [literate-sniffle](#).

Description (optional)

Tutorial para estudo e consulta do sistema de controle de versão (Git)

→ Breve descrição



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

→ Repositório público ou privado



Initialize this repository with a README

→ Não é necessário marcar

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**

Add a license: **None**



Create repository

GitHub

Somente PatrickSF está contribuindo

Será abordado mais a frente

Tutorial para estudo e consulta do sistema de controle de versão (Git) — Edit

1 commit

1 branch

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



PatrickSF Initial commit

Latest commit 5dcd1f7 4 minutes ago



README.md

Initial commit

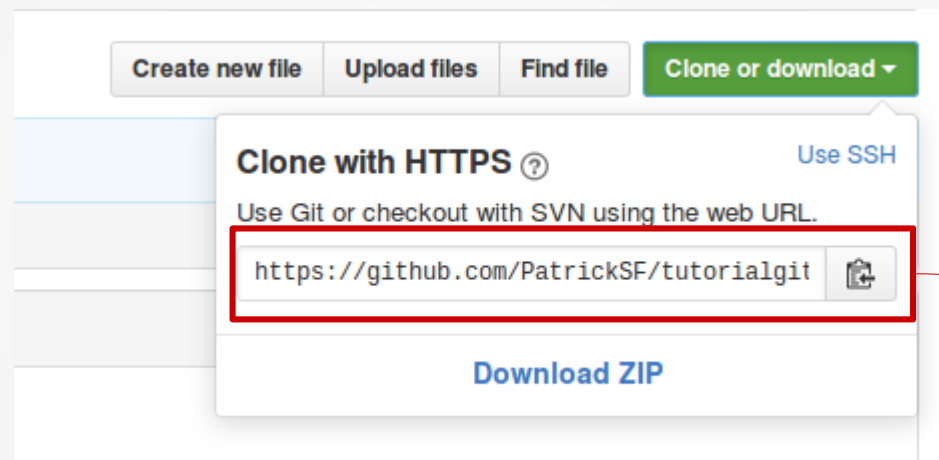
4 minutes ago

README.md

tutorialgit

Tutorial para estudo e consulta do sistema de controle de versão (Git)

GitHub



URL alvo do repositório, será necessária mais a frente



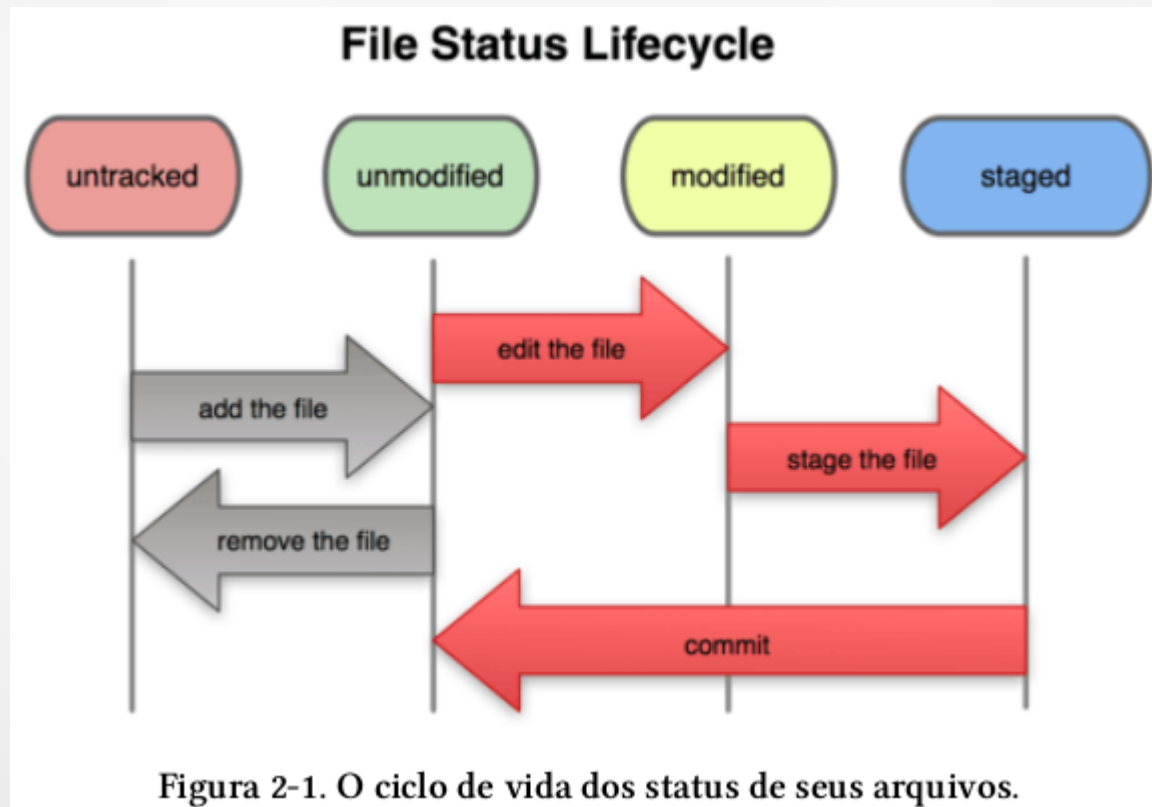
Retomando: Git Essencial

Git Essencial

- Obtendo um repositório Git
 - Inicializando repositório existente: `$ git init`
 - Clonando repositório existente:
 - `$ git clone URL #Mencionada no slide 21`
 - `$ git clone URL NomeDiretórioDestino`

Git Essencial

- 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

Git Essencial

- Status dos arquivos
 - `$ git status`
- Mensagem do status:
 - Após um clone
 - `# On branch master`
`nothing to commit (working directory clean)`
 - Após adicionar um arquivo ex: `'README'`
 - `# On branch master`
`# Untracked files:`
`# (use "git add <file>..." to include in what will be committed)`
`#`
`# README`
`nothing added to commit but untracked files present (use "git add" to track)`

Git Essencial

- Monitorando novos arquivos ('README' do slide anterior)
 - `$ git add README`
- Verificando novamente com 'git status':
 - `$ git status`
 - # On branch master
 - # Changes to be committed:
 - # (use "git reset HEAD <file>..." to unstage)
 - #
 - # new file: README
 - #

Git Essencial

- Status após modificação de arquivos monitorados
 - `$ git status`
 - # On branch master
 - # **Changes not staged for commit:**
 - # (use “git add <file>...” to update what will be #committed)
 - #
 - # modified: ‘NomeDoArquivo’
 - #
 - Basta utilizar ‘git add NomeDoArquivo’ para selecioná-lo
 - Antes de realizar um commit verifique os status

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 arquivos no diretório `build/`

- `doc/*.txt` #Ignora todos os arquivos `.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
 - Ex:
`$ git mv README.txt README`
`$ git status`
On branch master
Your branch is ahead of 'origin/master' by 1 commit

Changes to be committed
(use "git reset HEAD <file>..." to unstage)

renamed: README.txt → README

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

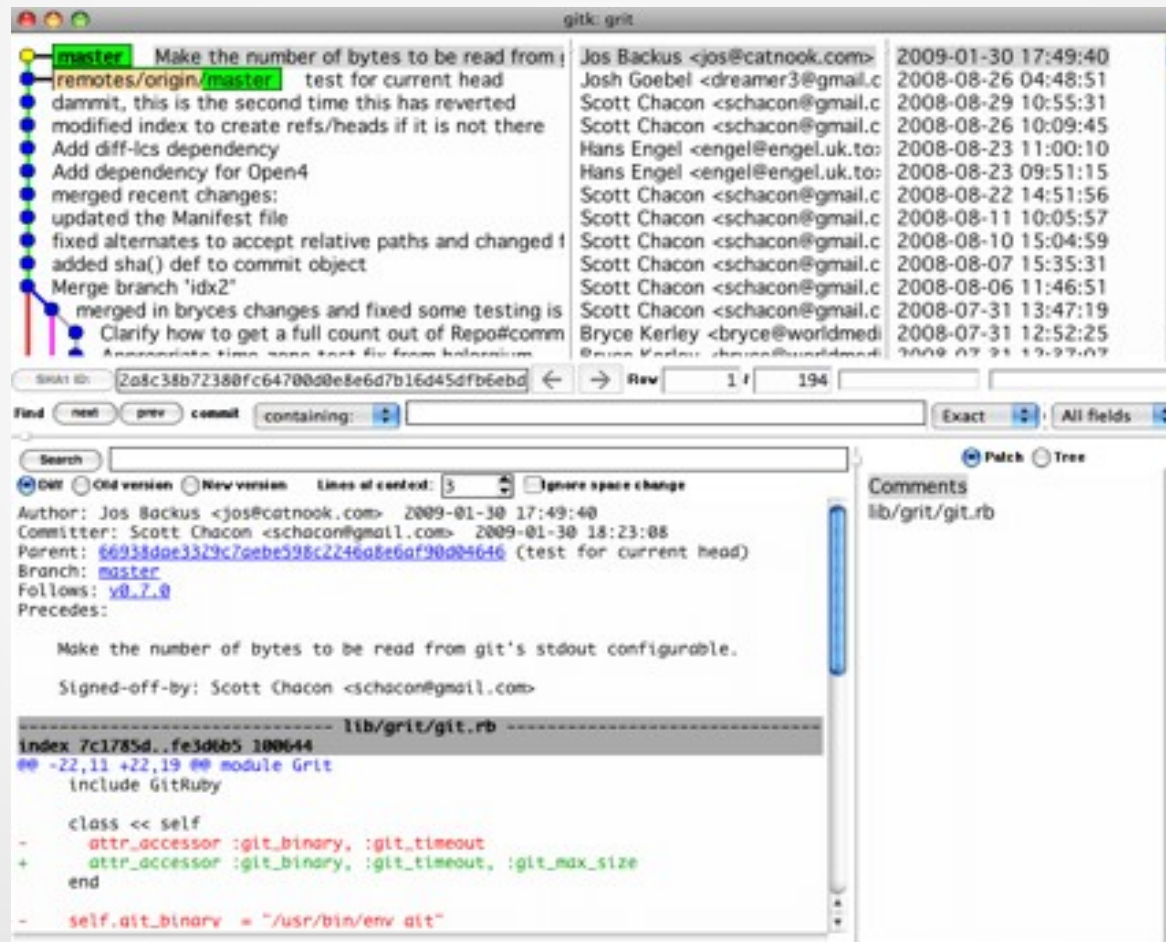
Opção	Descrição
-p	Mostra o patch introduzido com cada commit.
-stat	Mostra estatísticas de arquivos modificados em cada commit.
-shortstat	Mostra somente as linhas modificadas/inseridas/excluídas do comando -stat.
-name-only	Mostra a lista de arquivos modificados depois das informações do commit.
-name-status	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

- `$ gitk` #Ferramenta gráfica para visualizar seu histórico de commit



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
- 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
- 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**
- Inspeccionando 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

- Listando suas tags
 - `$ git tag` #Servem para marcar pontos de release (v1.0)
 - `$ git tag -l 'v1.4.2.*'` #Lista as tags com esta nomenclatura
- Tags anotadas
 - `$ git tag -a 'versão' -m 'mensagem'`
 - `$ git show 'versão'` #Exibi dados da tag junto com commit
- Tags assinadas
 - `$ git tag -s 'versão' -m 'mensagem'`
- Tags leves
 - `$ git tag 'versão'`

Git Essencial

- Verificando tags
 - `$ git tag -v 'versão'` #Necessário ter chave pública do assinados no seu chaveiro
- Taggeando após commit
 - `$ git log --pretty=oneline` #Para verificar seus commits
 - `$ git tag -a 'versão' 'chave do commit ou parte dela'`
- Compartilhando tags
 - `$ git push [nome-remoto] [nome-tag]`
 - `$ git push [nome-remoto] --tags` #Transfere todas tags

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`
 - #Cria pseudônimos para digitar ci ao invés de commit, etc..

Ramificação (Branching)

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

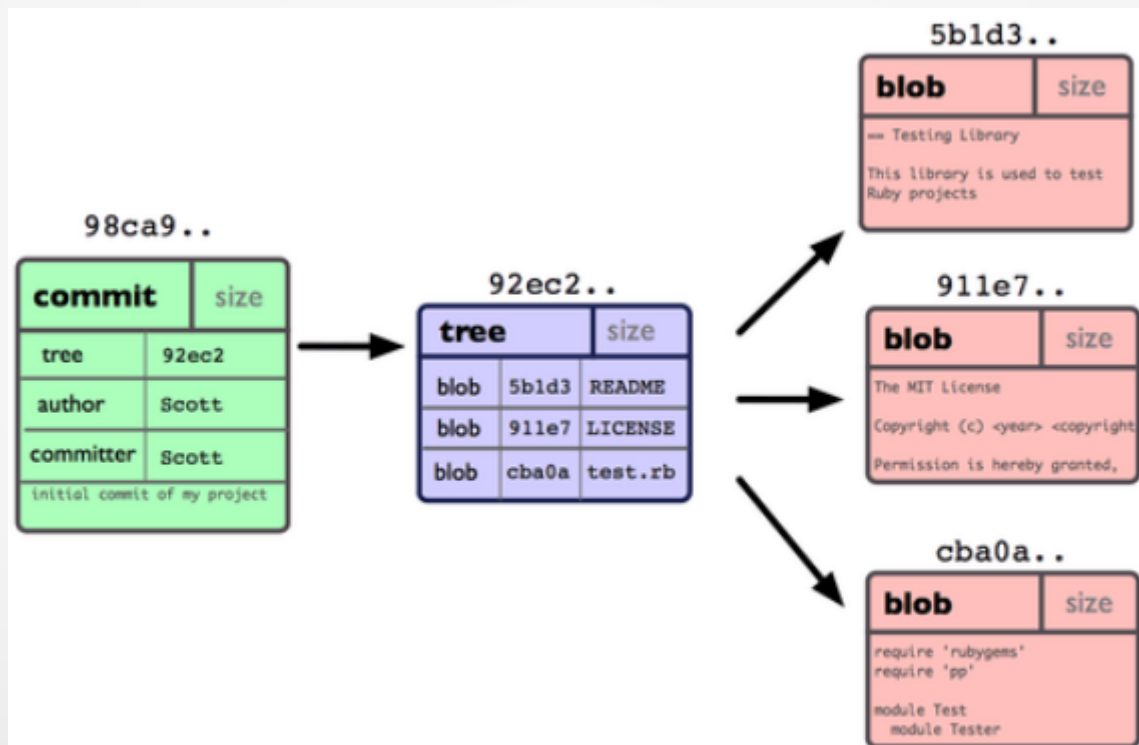


Figura 3-1. Dados de um repositório com um único commit.

Ramificação (Branching)

- O **branch** padrão é o **master**

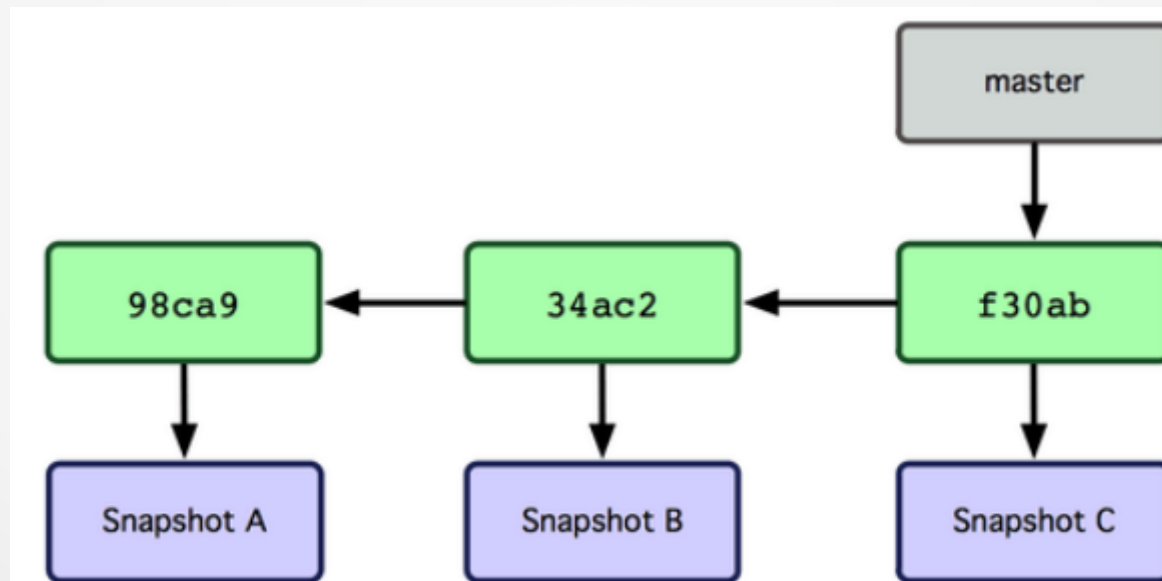


Figura 3-3. Branch apontando para o histórico de commits.

Ramificação (Branching)

- Criando novo **branch**
 - `$ git branch testing #testing` é o nome do branch

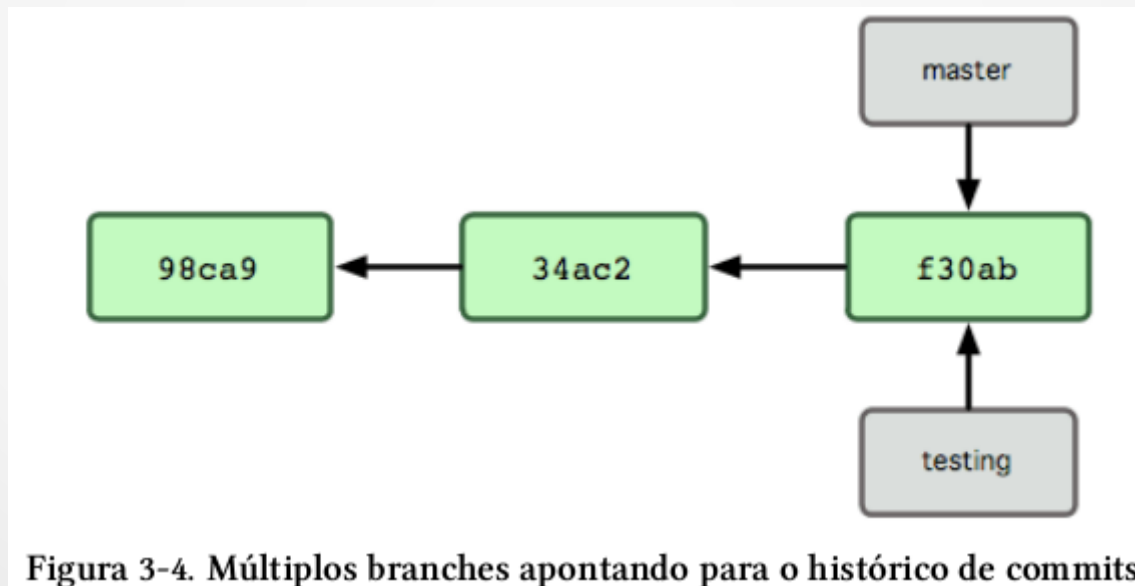


Figura 3-4. Múltiplos branches apontando para o histórico de commits.

Ramificação (Branching)

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

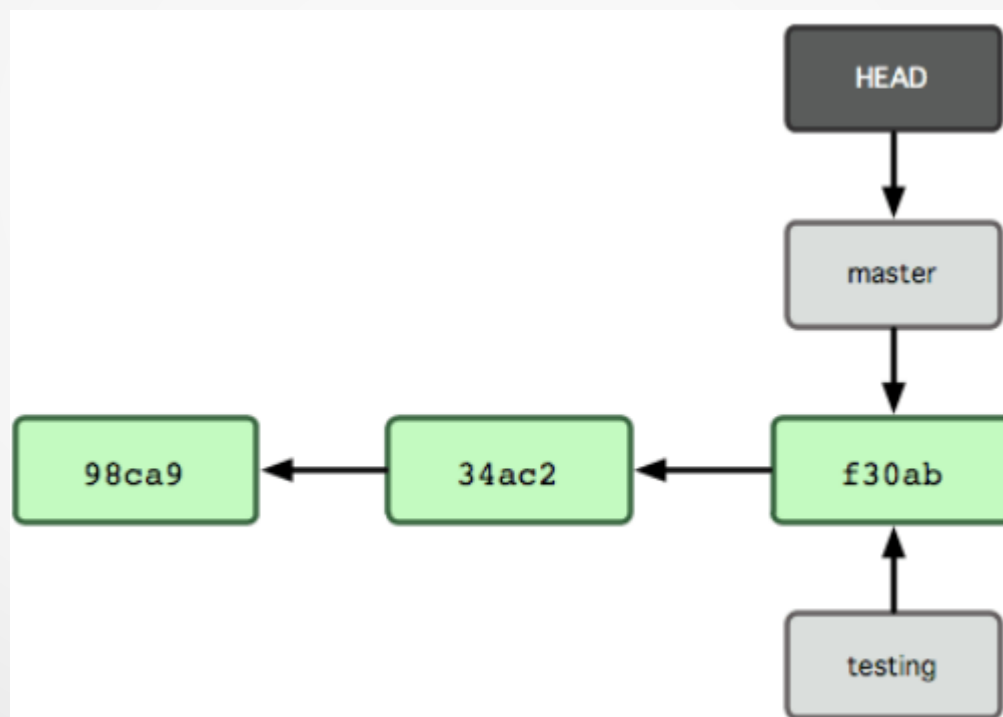


Figura 3-5. HEAD apontando para o branch em que você está.

Ramificação (Branching)

- Mudando de branch
 - \$ git checkout testing

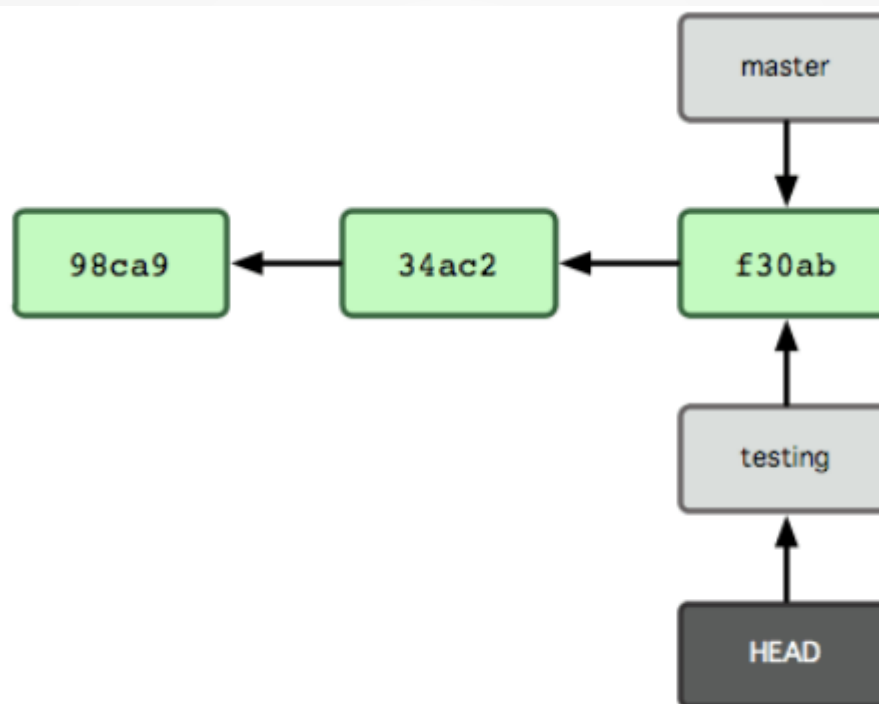


Figura 3-6. O HEAD aponta para outro branch quando você troca de branches.

Ramificação (Branching)

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

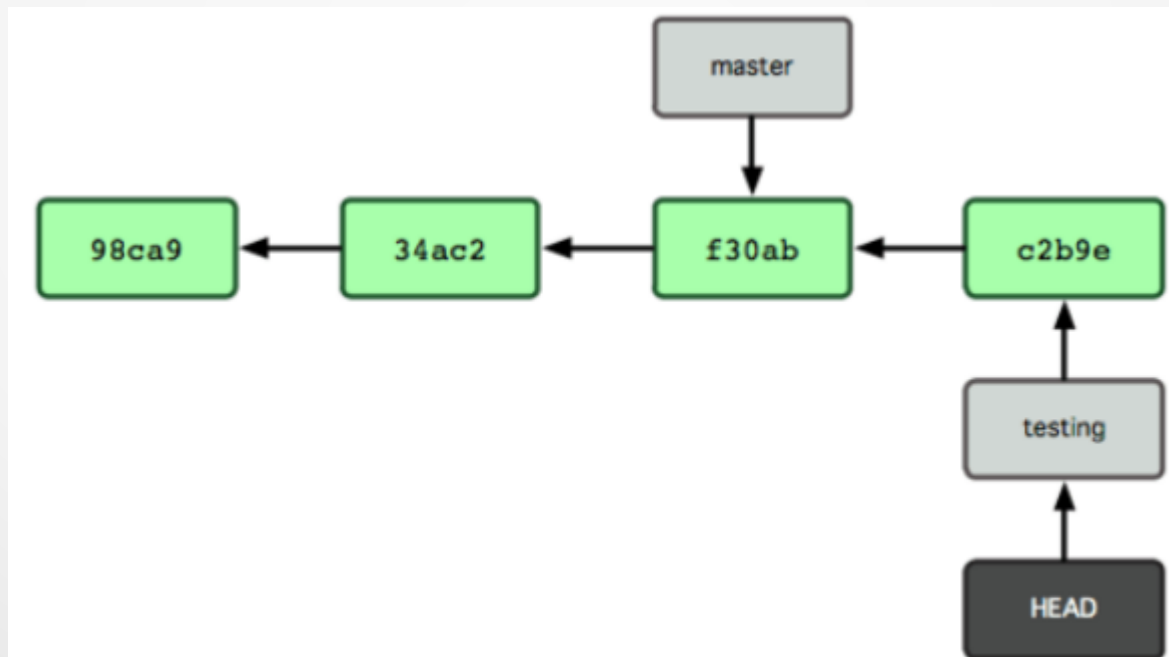


Figura 3-7. O branch para o qual HEAD aponta avança com cada commit.

Ramificação (Branching)

- Alterando para master
 - \$ git checkout master

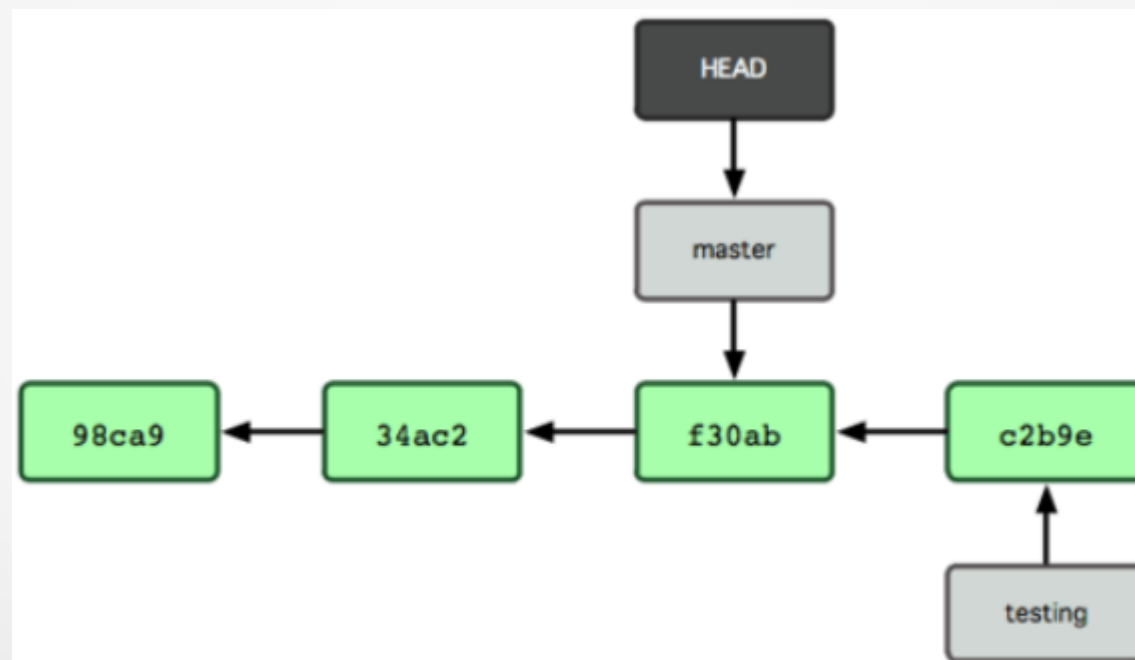


Figura 3-8. O HEAD se move para outro branch com um checkout.

Ramificação (Branching)

- Realizando novo **commit**
 - `$ vim test.rb`
 - `$ git commit -a -m 'novo commit'`

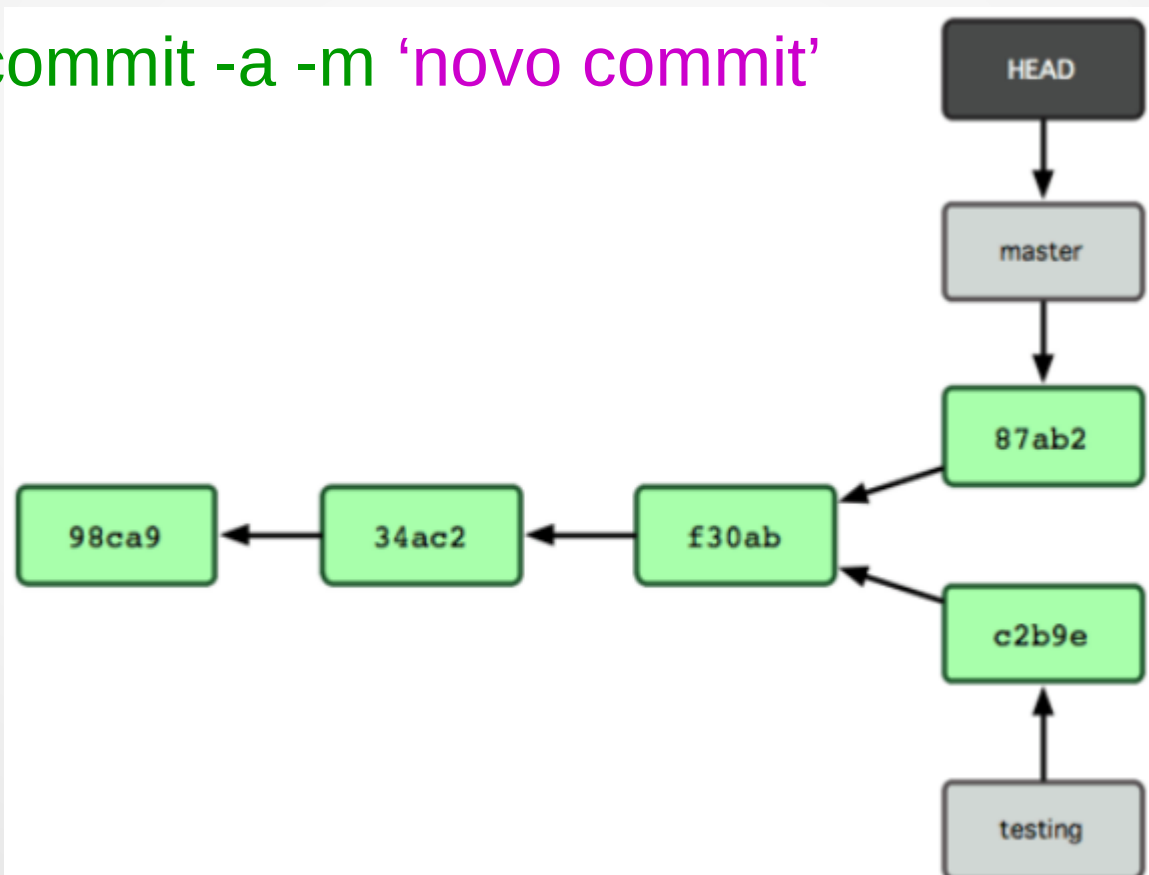


figura 3-9. O histórico dos branches diverge.

Ramificação (Branching)

- Branch e merge
 - `$ git checkout -b 'hotfix'` #Cria nova branch e muda para ela automaticamente; realize as modificações

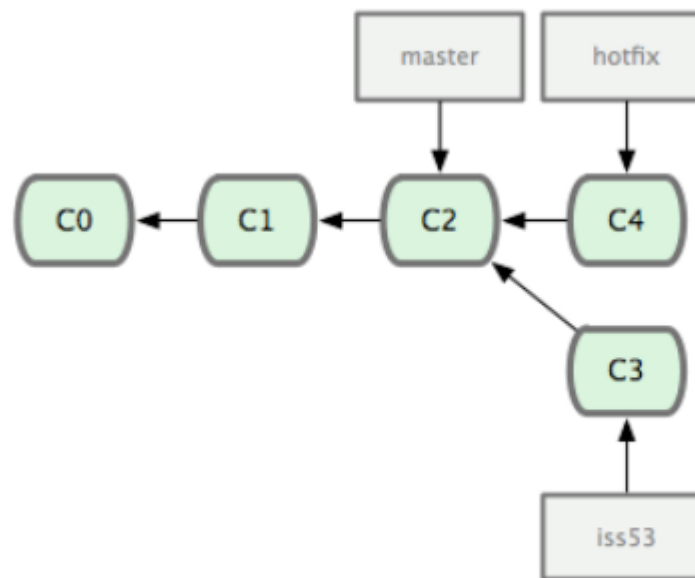


Figura 3-13. branch de correção (hotfix) baseado num ponto de seu branch master.

Ramificação (Branching)

- `$ git checkout master` #Retorna para branch master
- `$ git merge hotfix` #Modificação está agora no snapshot do branch master

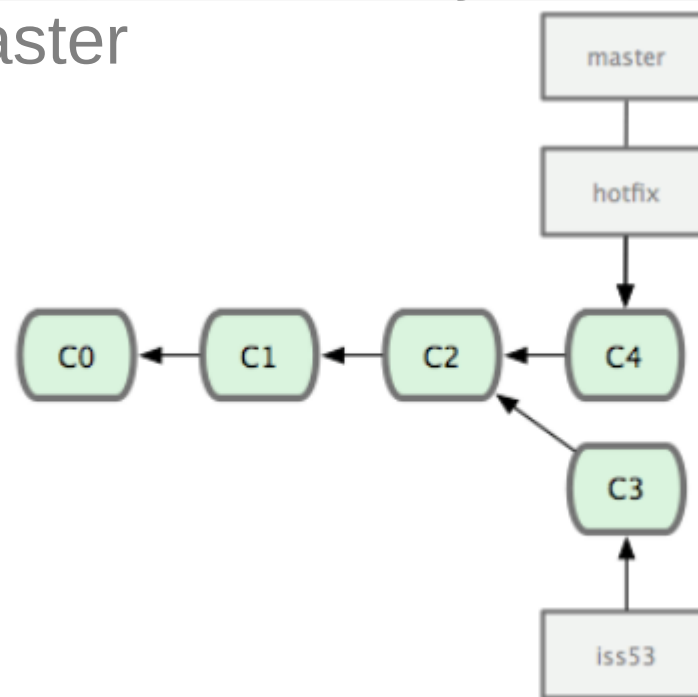


Figura 3-14. Depois do merge seu branch master aponta para o mesmo local que o branch hotfix.

- `$ git branch -d hotfix` #Apaga a branch hotfix que não é mais necessária

Ramificação (Branching)

- Merge de três vias

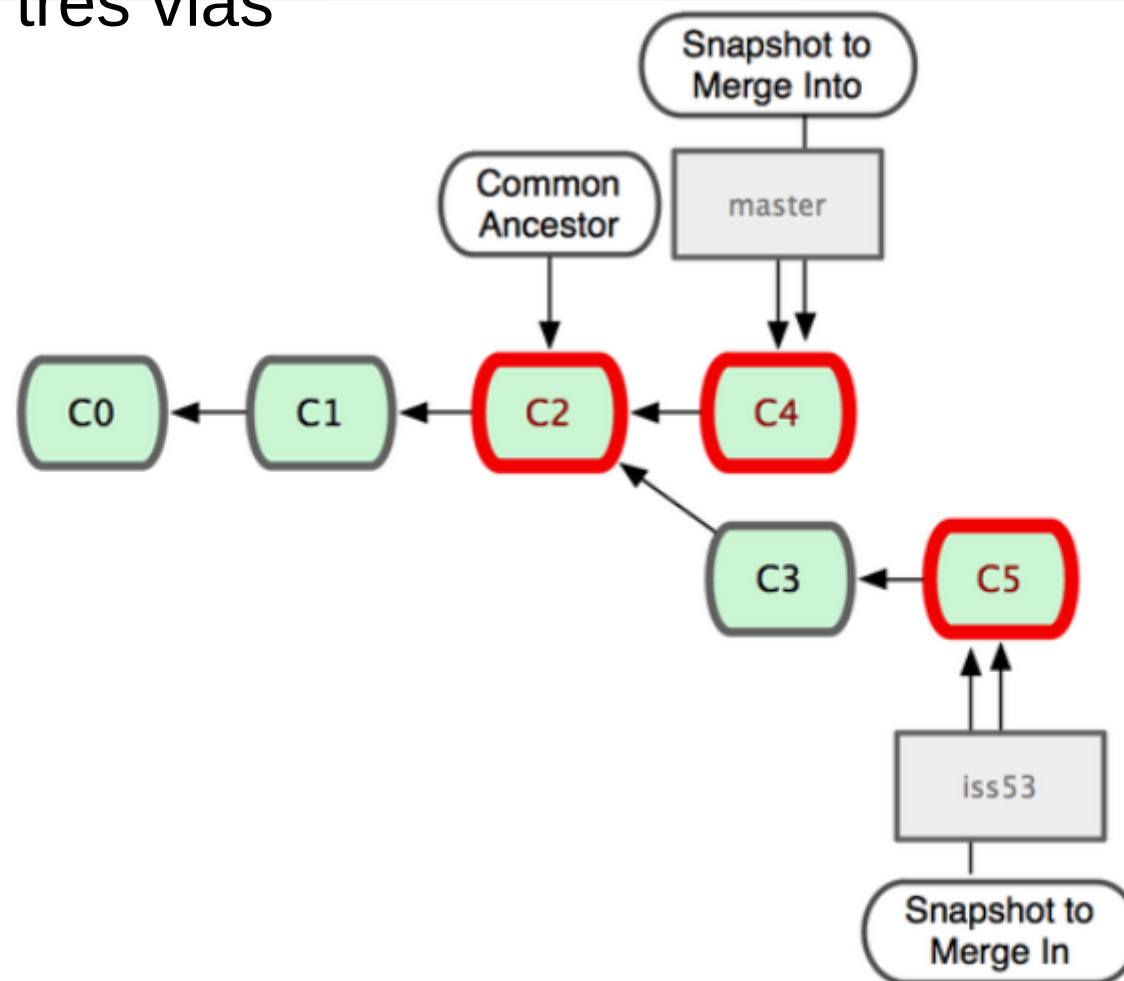


Figura 3-16. Git identifica automaticamente a melhor base ancestral comum para o merge do branch.

Ramificação (Branching)

- Merge de três vias

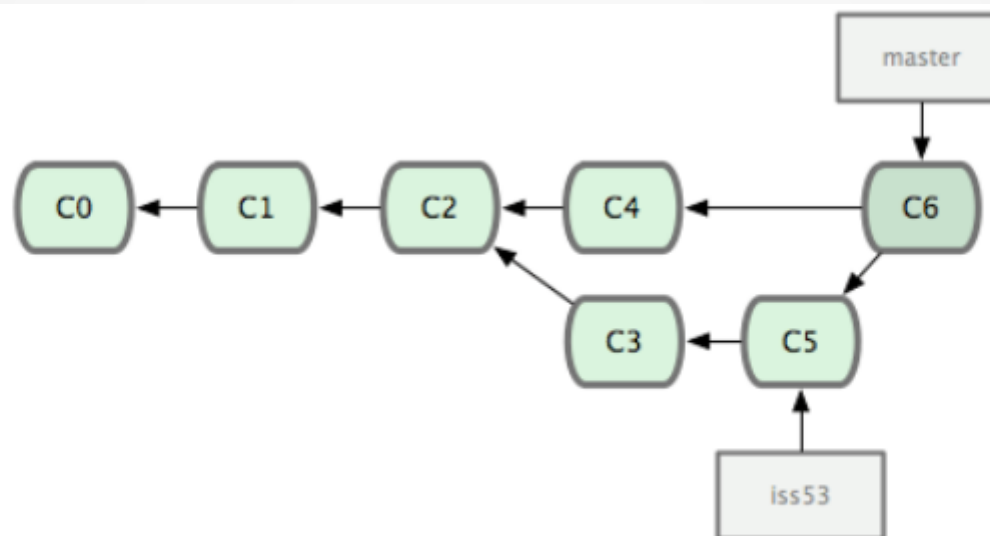


Figura 3-17. Git cria automaticamente um novo objeto commit que contém as modificações do merge.

- Git determina o melhor ancestral comum e realiza o merge de três vias

Ramificação (Branching)

- 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

Ramificação (Branching)

- 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`

Ramificação (Branching)

- Fluxo de trabalho com branches

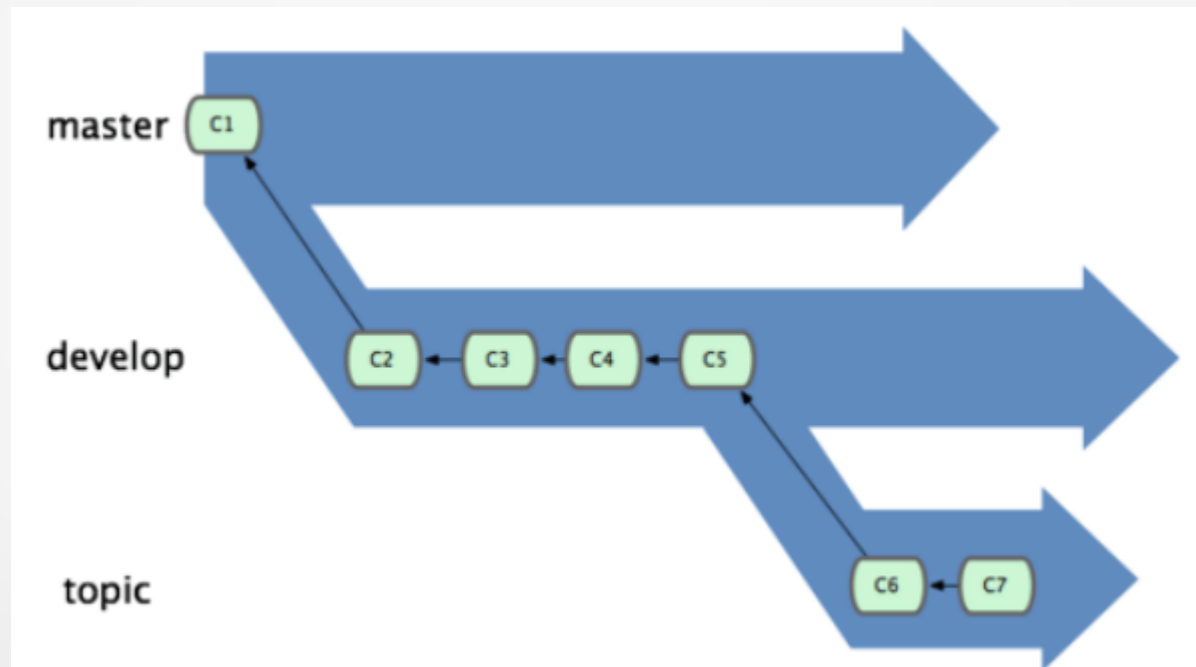


Figura 3-19. Pode ser mais útil pensar em seus branches como contêineres.

Ramificação (Branching)

- Branches tópicos
 - Branch de curta duração
- 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

Ramificação (Branching)

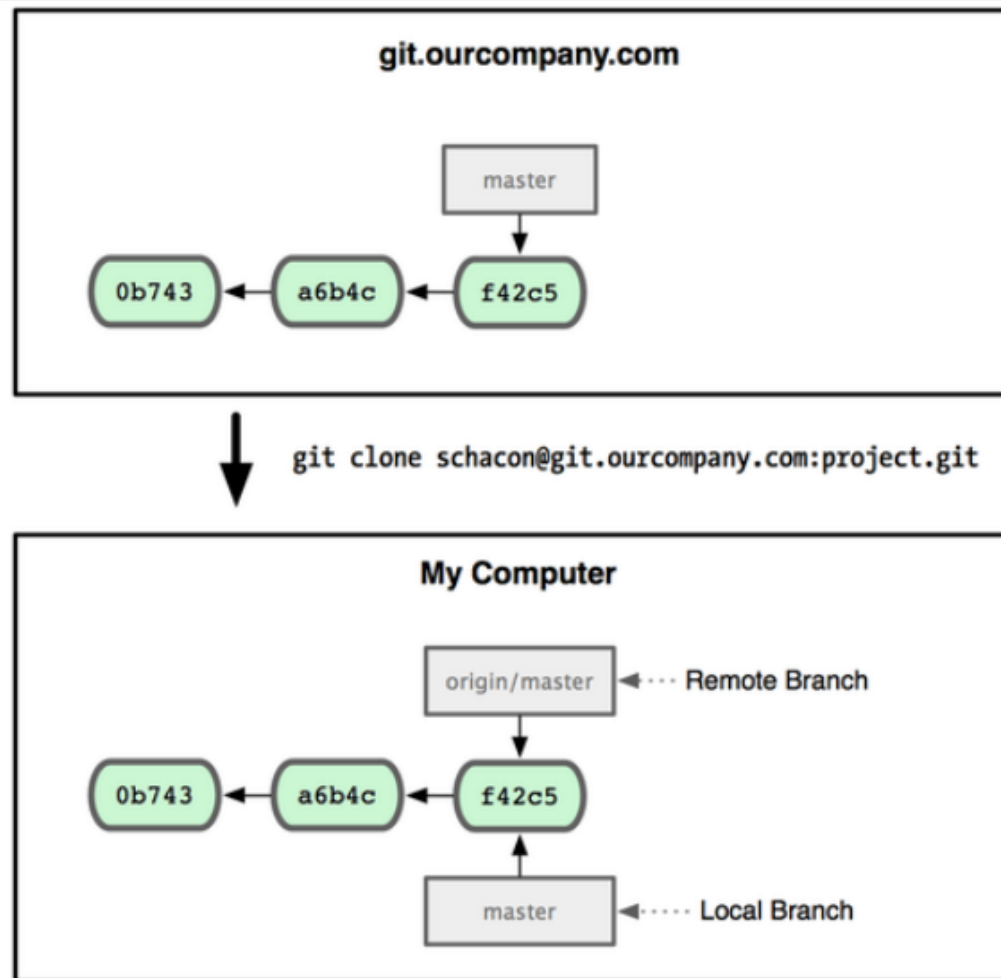


Figura 3-22. Um comando clone do Git dá a você seu próprio branch master e origin/master faz referência ao branch master original.

Ramificação (Branching)

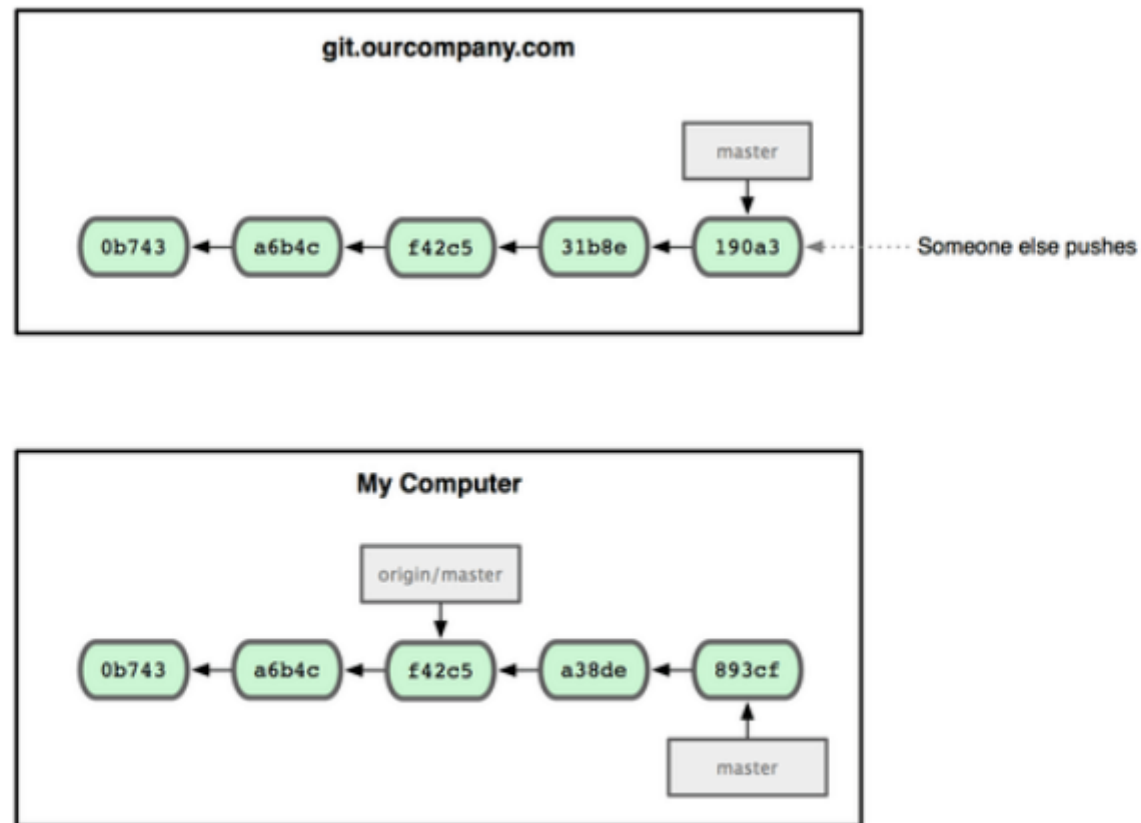


Figura 3-23. Ao trabalhar local e alguém enviar coisas para seu servidor remoto faz cada histórico avançar de forma diferente.

Ramificação (Branching)

- `$ git fetch origin` #Origin é o repositório remoto criado automaticamente após um clone

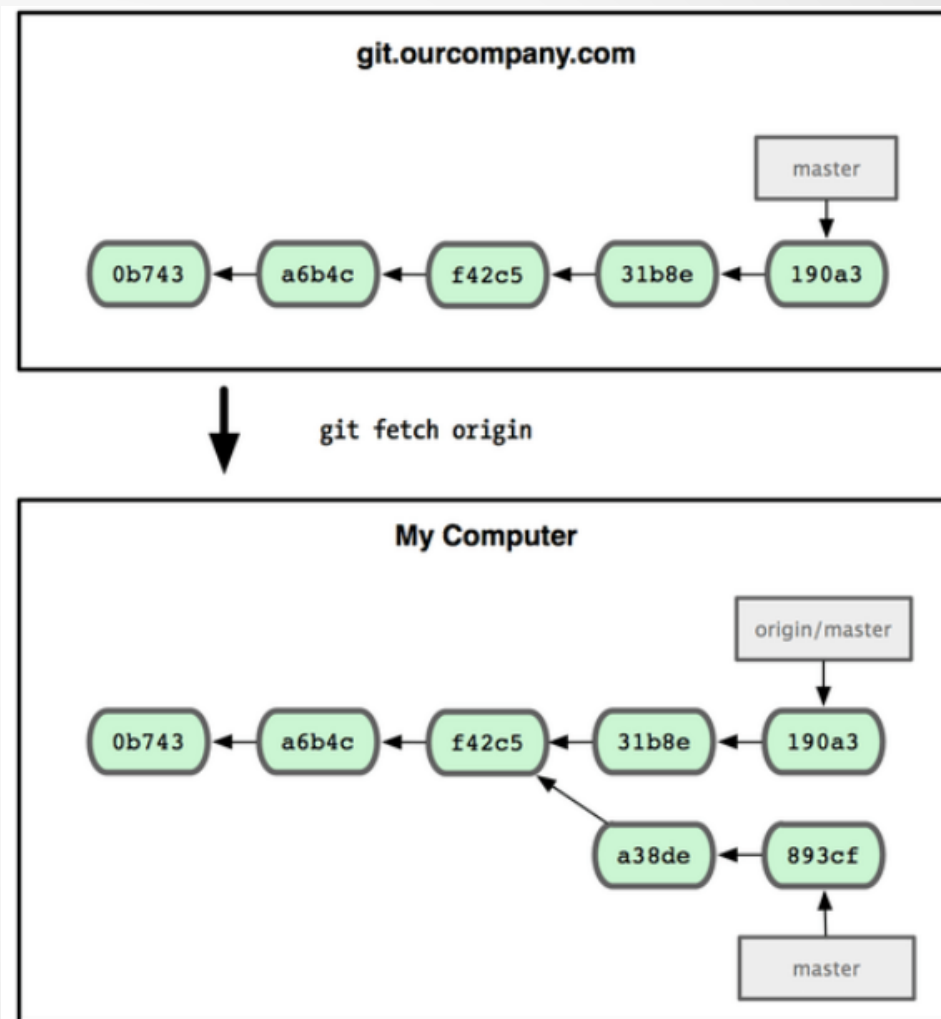


Figura 3-24. O comando `git fetch` atualiza suas referências remotas.

Ramificação (Branching)

- Trabalhando com múltiplos servidores remotos de mesma referência
 - `$ git remote add teamone git.team1.ourcompany.com`
#Adiciona um remoto da URL `git.team1.ourcompany.com` com a tag teamone que possui mesma referência que o servidor descrito anteriormente

Ramificação (Branching)

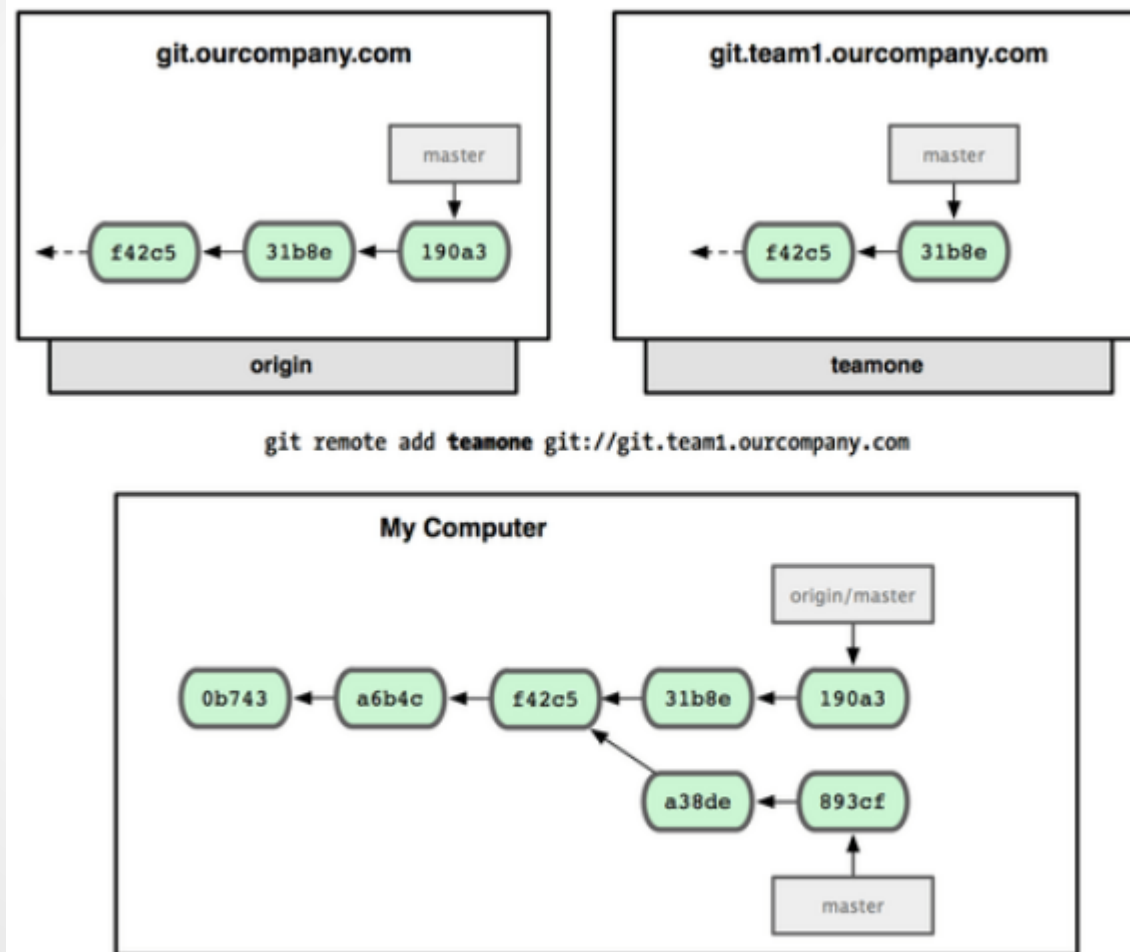


Figura 3-25. Adicionando outro servidor remoto.

Ramificação (Branching)

- Ao realizar um `$ git fetch teamone`, por conta do servidor ter um subconjunto dos dados do seu servidor origin, git cria somente um branch `teamone/master` com referência ao commit do master em teamone

Ramificação (Branching)

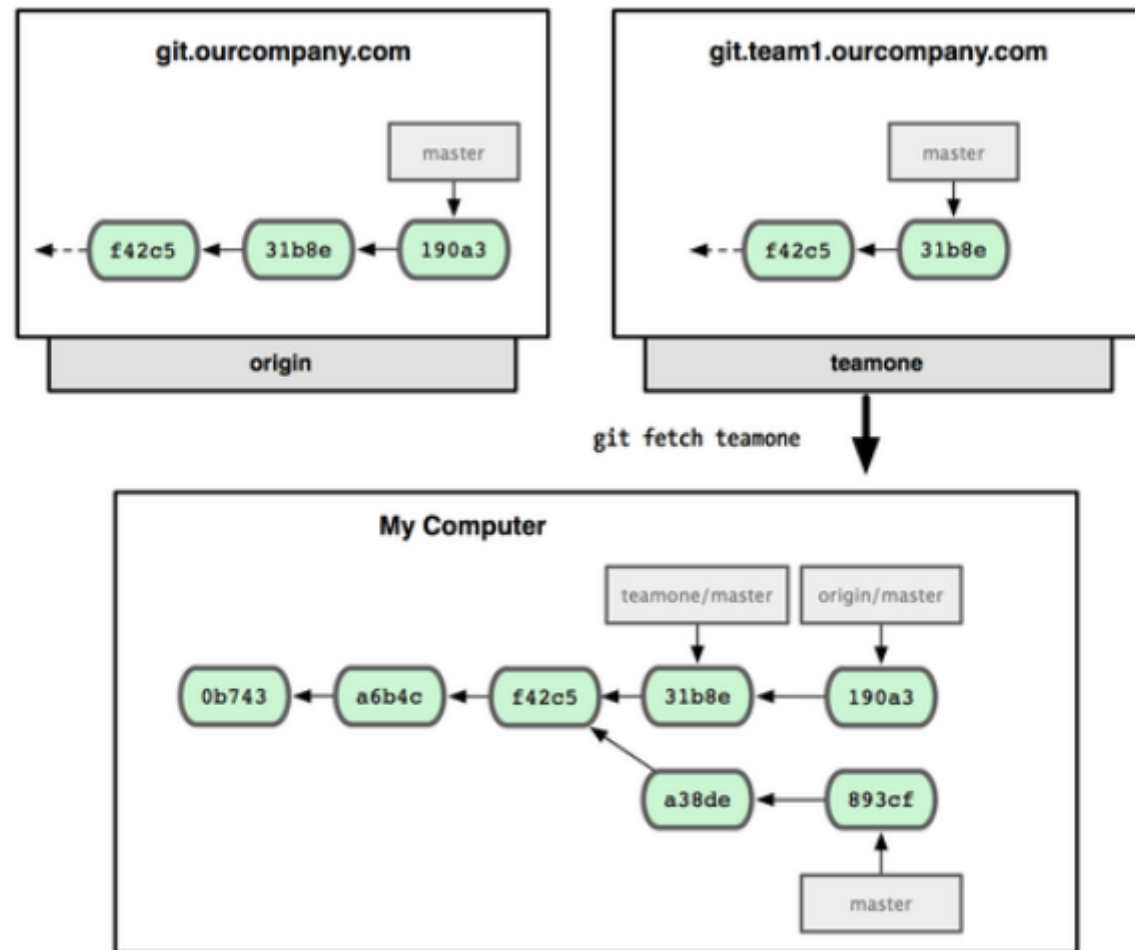


Figura 3-26. Você consegue uma referência local para a posição do branch master do teamone.

Ramificação (Branching)

- Enviando (Pushing)
 - `$ git push origin serverfix #(remote) (branch)`
 - `$ git push origin serverfix:serverfix #(remote) (branch):(branch destino)`, caso queira enviar um branch local `serverfix` para um remoto `awesomebranch`, basta `serverfix:awesomebranch`
 - Para adquirir cópias locais de novos branches remotos, necessário um `$ git merge [branch-remoto]` no branch que você está trabalhando
 - Você pode criar também um branch local para trabalhar que começa onde o remoto está
 - `$ git checkout -b [branch] [branch-remoto]` ou `$ git checkout --track [branch-remoto]` #Ex: `$ git checkout -b serverfix origin/serverfix`, ou substituir o nome local `serverfix` por outro que desejar

Ramificação (Branching)

- Apagando branches remotos
 - `$ git push [nome-remoto] :[branch]` #Apaga o branch não mais necessários do servidor

Ramificação (Branching)

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

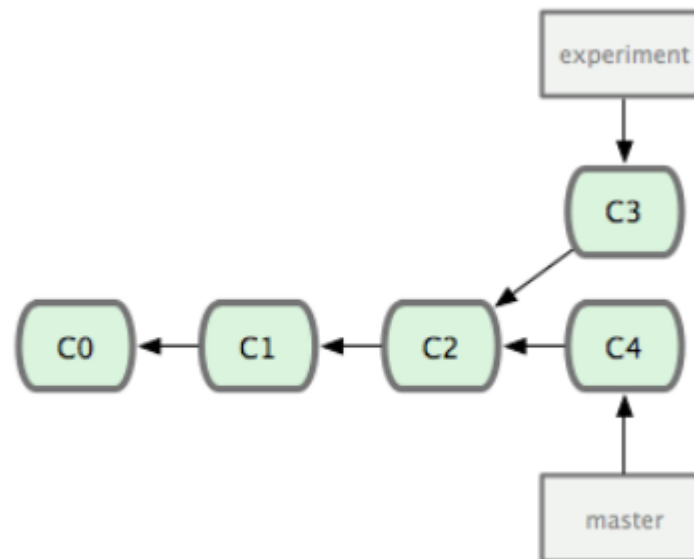


Figura 3-27. Divergência inicial no seu histórico de commits.

Ramificação (Branching)

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

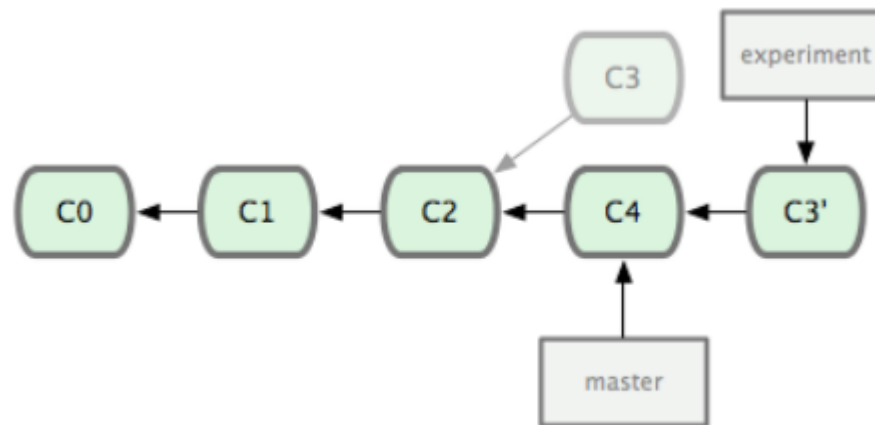


Figura 3-29. Fazendo o rebase em C4 de mudanças feitas em C3.

Ramificação (Branching)

- \$ git checkout master
- \$ git merge experiment

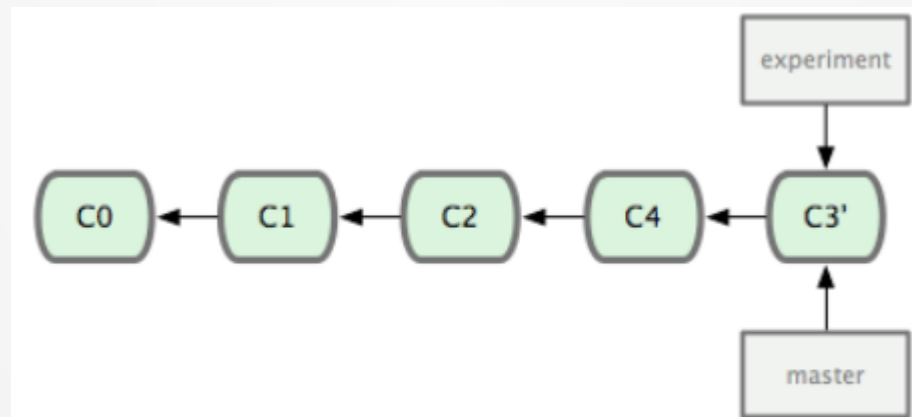


Figura 3-30. Fazendo um fast-forward no branch master.

Ramificação (Branching)

- Rebases interessantes

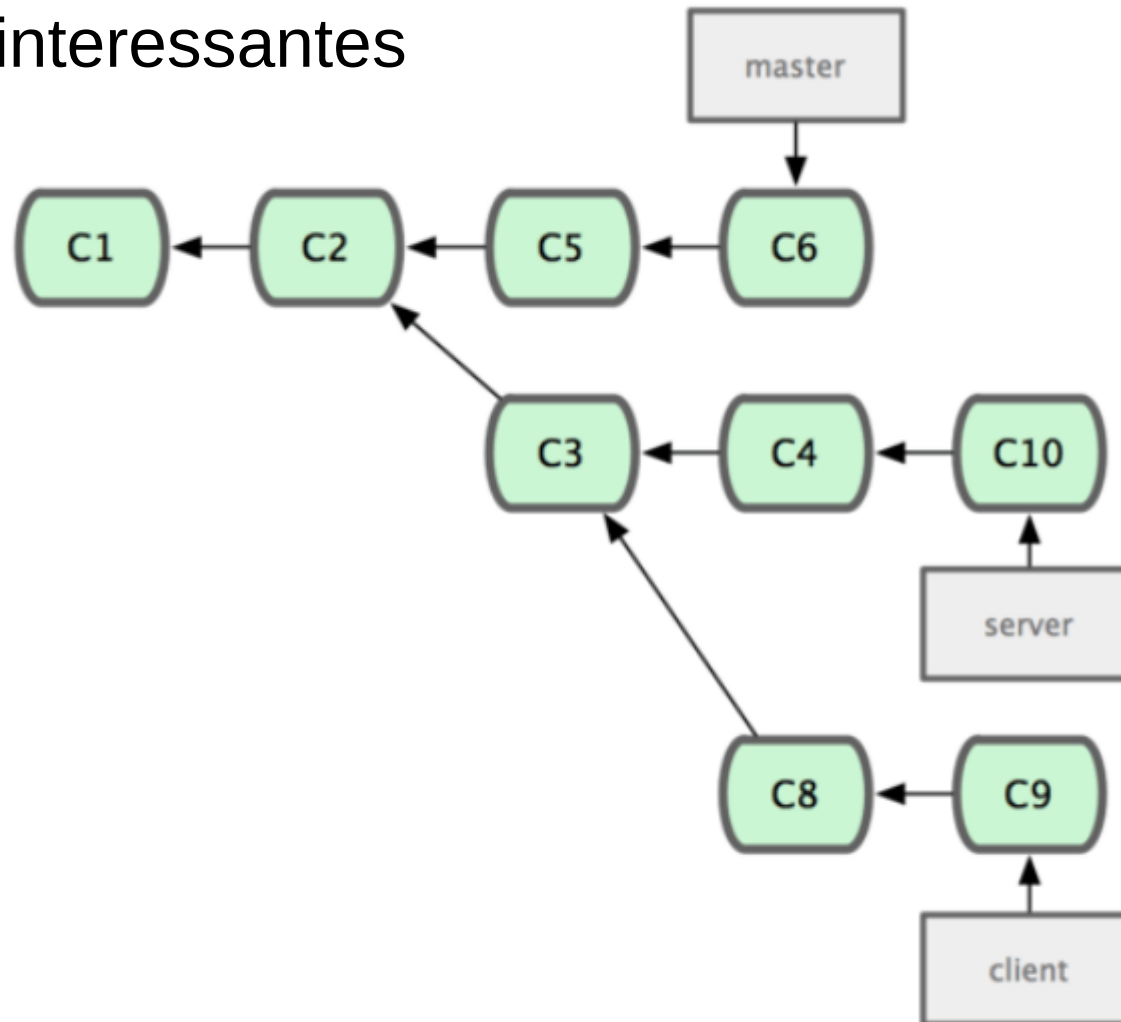


Figura 3-31. Histórico com um branch tópico a partir de outro.

Ramificação (Branching)

- Aplicando rebase somente no branch client
 - `$ git rebase --onto master server client` #Faz o checkout do branch **client**, verifica as mudanças a partir do ancestral em comum aos branches **client** e **server**, e coloca no **master**

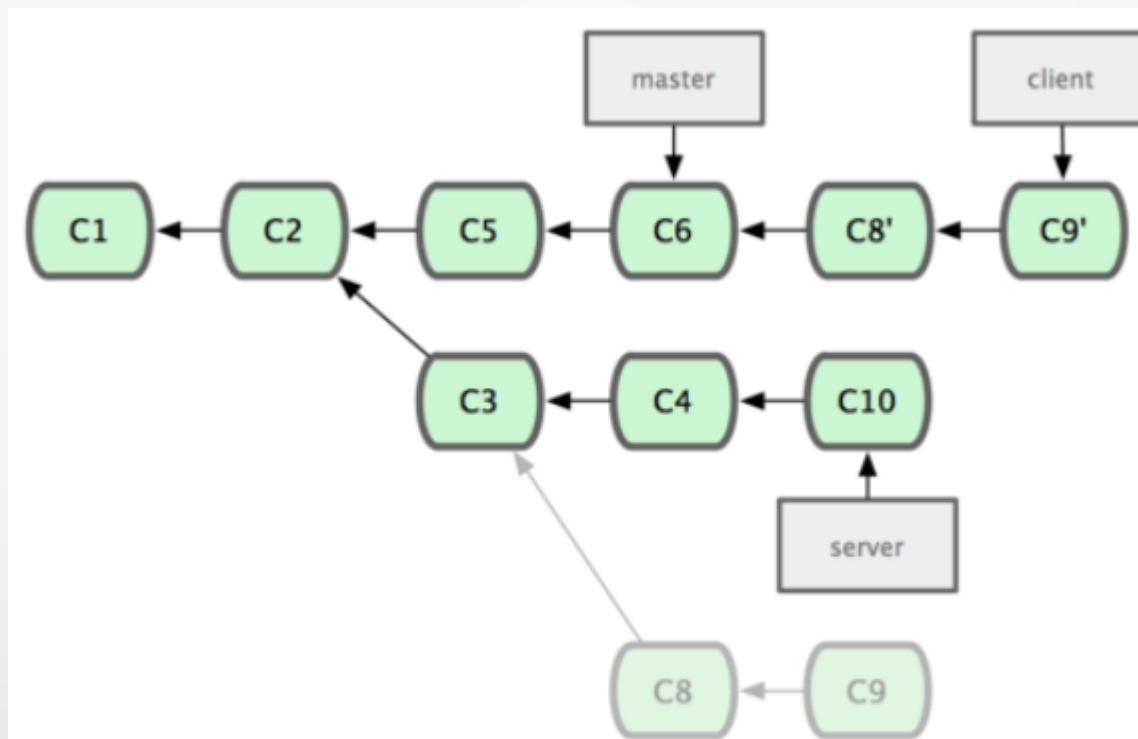


Figura 3-32. Fazendo o rebase de um branch tópico em outro.

Ramificação (Branching)

- \$ git checkout master
- \$ git merge client

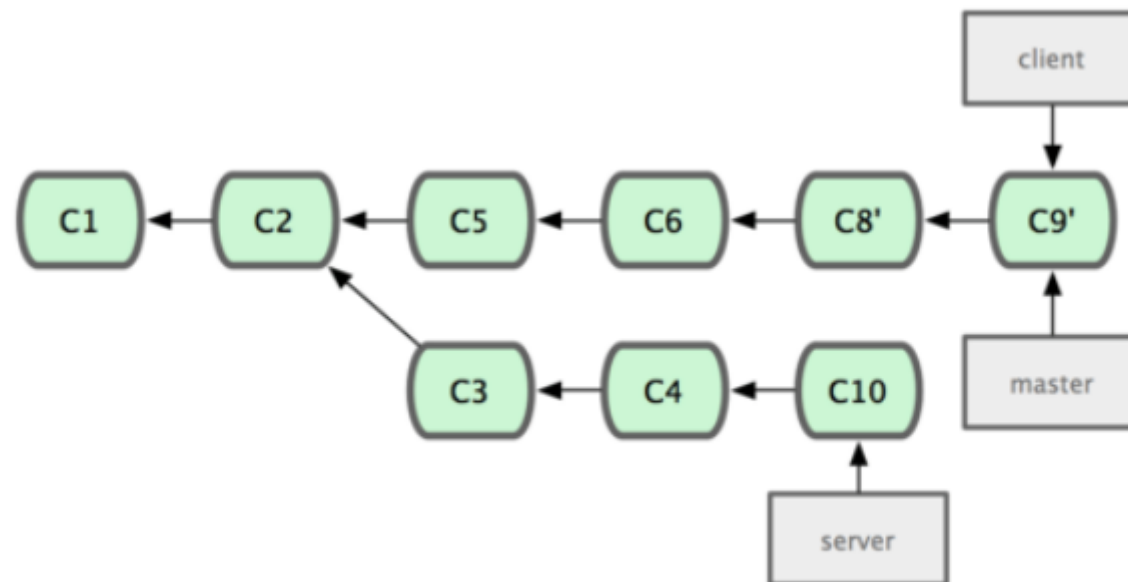


Figura 3-33. Avançando o seu branch master para incluir as mudanças do branch client.

Ramificação (Branching)

- Rebase do server sem precisar fazer checkout
 - `$ git rebase master server` #[branchbase] [branchtopico]

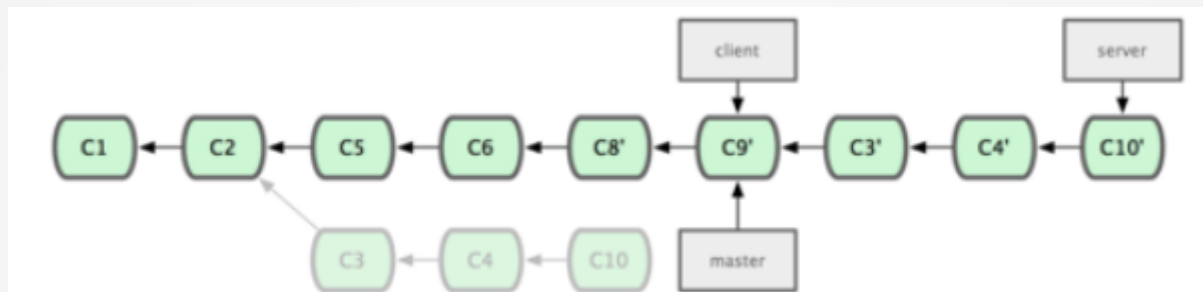


Figura 3-34. Fazendo o rebase do seu branch server após seu branch master.

- `$ git checkout master` #Vai para branch base
- `$ git merge server` #Concretiza a mesclagem
- `$ git branch -d client` #Apaga a branch client
- `$ git branch -d server` #Apaga a branch server

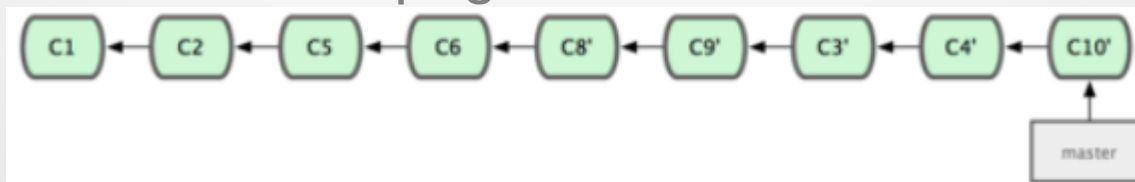


Figura 3-35. Histórico final de commits.

Ramificação (Branching)

- **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

Git no servidor

- É possível realizar um clone de um repositório local
 - \$ git clone /opt/git/project.git #Caminho do repositório
 - \$ git clone file:///opt/git/project.git
 - \$ git remote add local_proj /opt/git/project.git #Adiciona um repositório local
- Mais detalhes do git no servidor pode ser encontrado no livro texto especificado nas referências

Git distribuído

- Fluxo de trabalho centralizado
 - Hub central onde todos sincronizam seu trabalho

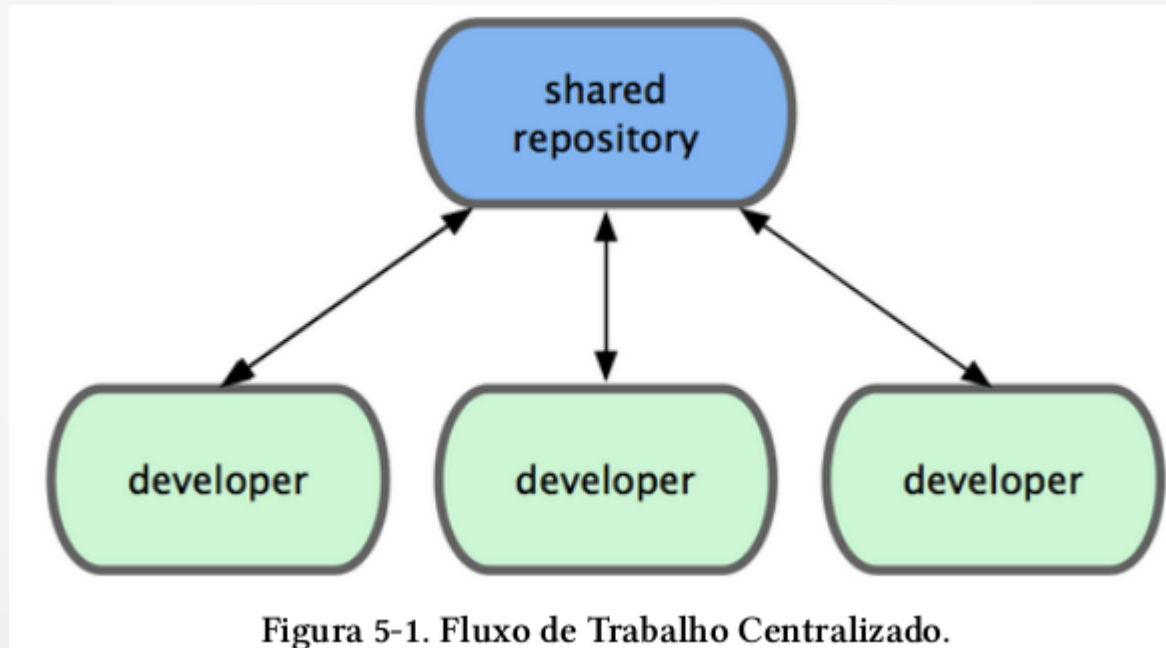


Figura 5-1. Fluxo de Trabalho Centralizado.

- Antes de cada desenvolvedor realizar o push precisa fazer um merge do trabalho do primeiro, caso exista um

Git distribuído

- Fluxo de trabalho do gerente de integração
 - O mantenedor do projeto propaga as alterações para seu repositório público
 - O desenvolvedor clona o repositório e faz alterações
 - O desenvolvedor dá push das alterações para sua própria cópia pública
 - O desenvolvedor envia um e-mail pedindo para o mantenedor puxar as alterações (pull request)
 - O mantenedor adiciona o repositório do desenvolvedor como um repositório remoto e faz merge das alterações localmente
 - O mantenedor dá push das alterações mescladas para o repositório principal

Git distribuído

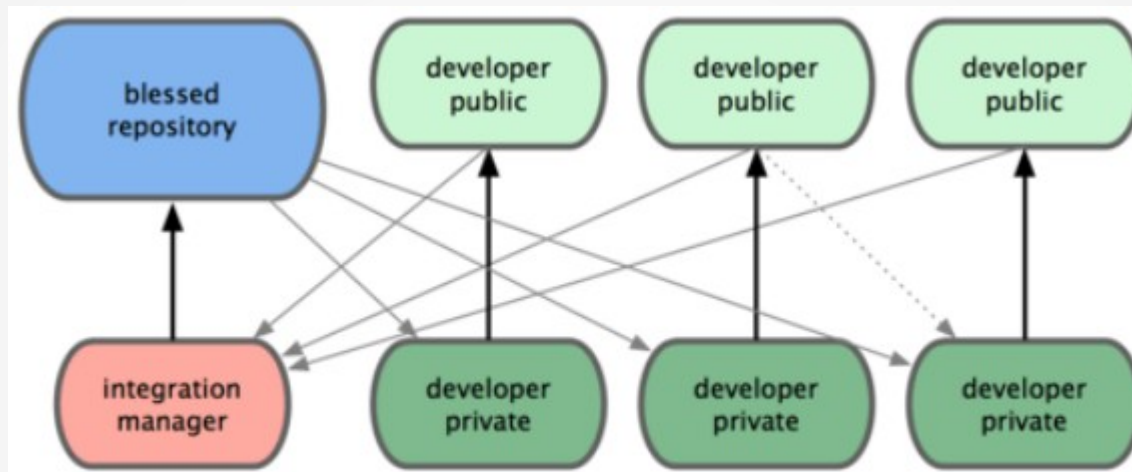


Figura 5-2. Fluxo de trabalho de Gerente de Integração.

Git distribuído

- Fluxo de trabalho de ditador e tenentes
 - Desenvolvedores regulares trabalham em seu topic branch e baseiam seu trabalho sobre o master. O branch master é o do ditador
 - Tenentes fazem merge dos topic branches dos desenvolvedores em seus master
 - O ditador faz merge dos branches master dos tenentes em seu branch master
 - O ditador dá push das alterações de seu master para o repositório de referência para que os desenvolvedores possam fazer rebase em cima dele

Git distribuído

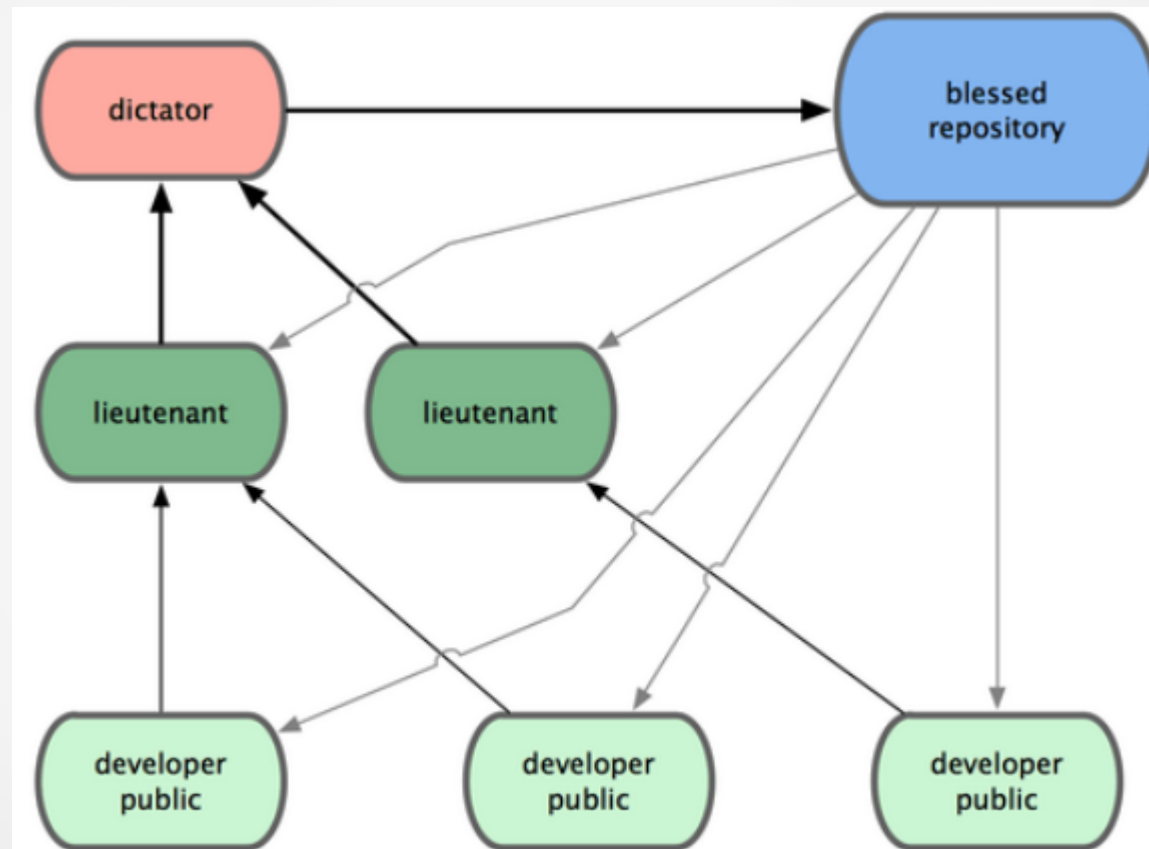


Figura 5-3. Fluxo de Trabalho do Ditador Benevolente.

Git distribuído



Figura 5-11. Sequencia geral dos eventos para um fluxo de trabalho simples para Git com múltiplos desenvolvedores.

Git distribuído

- Gerando patch
 - `$ git diff <hash-1> <hash-2> > <nome-do-patch>.patch` #Gera o patch a parti do hash dos commits especificados
 - `$ git format-patch [branch] --stdout > <nome-do-patch>.patch` #Gera patch preservando o commit
- Enviando patch (necessário configurar a seção imap no arquivo `~/.gitconfig`)
 - `$ git send-email [caminho-do-patch] #*.patch` envia todos patches
- Aplicação de patch
 - `$ git apply [caminho-do-patch]` #Aplica o patch feito com diff
 - `$ git apply --check [caminho-do-patch]` #Testa o patch antes de aplicar
 - `$ git am [caminho-do-patch]` #Aplica patch feito com format-patch

Git distribuído

- Determinando o que é introduzido
 - `$ git log contrib --not master` #Exibi os commits que estão no contrib mas não no master
 - `$ git log -p` #Exibi as mudanças que cada commit introduz
 - `$ git diff [branch]` #Exibi um diff completo da branch
 - `$ git merge-base [branch] [branch-base]` #Exibi o hash do ancestral comum
 - `$ git diff hash` #Exibi as mudanças do topic branch a partir do ancestral comum (mais preciso e evita enganos)
 - `$ git diff [branch-base]...[branch]` #Substitui os dois últimos comando acima
- Mais detalhes do git distribuído pode ser encontrado no livro texto especificado nas referências

O que é GitHub Pages?

- Um serviço de hospedagem de páginas estáticas
- Permite transformar seus repositórios do GitHub em Websites.
- Permite a criação e publicação das páginas web através do Gerador Automático de Página

O que é GitHub Pages?

- O GitHub Pages não deve ser usado para transações sensíveis tais como envio de senhas ou números de cartões de crédito.
- As páginas criadas pelo GitHub Pages seguem as seguintes limitações de uso:
 - Repositórios devem ter um limite recomendado de 1GB
 - Páginas publicadas não podem ser maiores que 1GB
 - Sites têm uma largura de banda de 100GB ou 100,000 requisições por mês
 - Sites têm um limite de 10 *builds* por hora

Páginas de Usuário, Organização e Projeto

- Há dois tipos básicos de GitHub Pages: *Páginas de Usuário/Organização* e *Páginas de Projeto*.
- É importante lembrar que as páginas sempre são publicamente acessíveis quando publicadas, até mesmo se o repositório é privado.

Páginas de Usuário, Organização e Projeto

Tipo de site GitHub Pages	Domínio padrão	Local dos arquivos fonte
Páginas de Usuário	username.github.io	master
Páginas de Organização	orgname.github.io	
Páginas de Projeto de Usuário	username.git.io/projectname	master, gh-pages ou pasta /docs como master
Páginas de Projeto de Organização	orgname.git.io/projectname	

Páginas de Usuário, Organização e Projeto

- Páginas de Usuário podem ser criadas por qualquer conta de usuário com um endereço e-mail verificado. Eles também podem usar *deploy keys* para automatizar o processo.
- Páginas de Organização podem ser criadas por qualquer membro com acesso *push* ao repositório e um endereço e-mail verificado. Para automatizar as *builds* pode-se configurar um usuário máquina como membro da organização. Páginas de Organização não dão suporte às *deploy keys*.

Páginas de Usuário, Organização e Projeto

- Diferente das páginas de usuário e organização, Páginas de Projeto são mantidas no mesmo repositório que o projeto.
- Tanto as contas pessoais quanto a de organizações podem criar Páginas de Projeto.
- Os passos para a criação das Páginas de Projeto é o mesmo dos anteriores

Páginas de Usuário, Organização e Projeto

- Páginas de Projeto são similares às páginas anteriores, com algumas pequenas diferenças:
 - Pode-se fazer e publicar sites dos ramos master ou gh-pages. Também é possível publicar seu site de uma pasta /docs no ramo master.
 - Se nenhum Domínio Customizado é usado, os sites são criados num subcaminho da Página de Usuário.
 - Um Domínio Customizado de páginas fazem que o mesmo domínio redirecione para todas as Páginas de Projeto hospedadas em uma conta.
 - 404s customizados só irão funcionar se um domínio customizado é usado. Do contrário, a Página de Usuário 404 é usada.

Configurando um arquivo publicável

- Você pode configurar o GitHub Pages para publicar os arquivos-fonte de **master**, **gh-pages** ou diretório **/docs** em seu ramo **master** das Páginas de Projeto e outras páginas que têm um certo critério.
- Se seu site é uma Página de Usuário ou de Organização que tem um repositório chamado **<username>.github.io** ou **<orgname>.github.io**, você não pode publicar seu site de uma localidade diferente. Páginas de Usuário ou Organização que têm esse nome de repositório só publicado pelo ramo **master**.

Configurando um arquivo publicável

- As configurações padrão para publicação dos arquivos fonte do site dependem do tipo de site e os ramos que se tem no repositório.
- Se o repositório do seu site não tem os ramos **master** ou **gh-pages**, o arquivo de publicação é configurado como **None** e seu site não é publicado.
- Depois de criar ou o ramo **master** ou o **gh-pages**, você pode escolher um arquivo de publicação e seu site será publicado.
- Se você manipular ou der upload no repositório do site apenas como **master** ou **gh-pages**, as configurações do código do site será ativado automaticamente pra esse ramo.

Configurando um arquivo publicável

- Habilitando o GitHub Pages para publicar seu site de **master** ou **gh-pages**
 - No GitHub, navegue até o repositório do site do GitHub Pages.
 - Abaixo do nome do repositório, clique **Settings**.
 - Use o menu drop-down **Select source** para selecionar **master** ou **gh-pages** como fonte de publicação do GitHub Pages.
 - Clique **Save**

Configurando um arquivo publicável

- Publicando sua GitHub Pages de uma pasta **/docs** em seu ramo **master**:
 - Tenha uma pasta /docs na raiz do repositório
 - Não siga o esquema de nome de repositório <username>.github.io ou <orgname>.github.io
- GitHub Pages lerá tudo para publicar seu site, incluindo o arquivo **CNAME**, da pasta **/docs**

Configurando um arquivo publicável

- Publicando sua GitHub Pages de uma pasta **/docs** em seu ramo **master**:
 - No GitHub, navegue até o repositório do site do GitHub Pages.
 - Crie uma pasta na raiz do seu repositório no ramo **master** chamada **/docs**.
 - Abaixo do nome do repositório, clique em **Settings**.
 - Use o menu drop-down para selecionar **master branch /docs folder** como sua fonte de publicação do GitHub Pages
 - Clique em **Save**

O que é o Jekyll

- Gerador de sites estáticos suportado pelo GitHub Pages
- Prever e trara erros no site
- Configura uma versão local do site
- Pode-se configurar a maior parte do site editando o arquivo `_config.yml`

Criando página por terminal

- Necessário criar um ramo órfão no repositório
- Processo mais seguro:
 - `$ git clone https://github.com/user/repository.git` #Inicie um novo clone
 - `$ cd repository` #Acessando a pasta clonada
 - `$ git branch` #Verificar se já existe a **master**
 - `$ git checkout --orphan master` #Crie um ramo **master** caso não existe
 - `$ git rm -rf` #Remova todos os arquivos para criar um novo diretório
 - `$ echo "My Page" > index.html`
`$ git add index.html`
`$ git commit -a -m "First pages commit"`
`$ git push origin master` #Adicione conteúdos e push
 - Após o push sua página pode ser acessada por:
`http(s)://<username>.github.io/<projectname>`

Removendo páginas

- Para remover uma página de projeto, delete o ramo **gh-pages**
- Para remover uma página de usuário, delete o ramo master ou o repositório **username.github.io**

Desenvolvedores

- Tutorial desenvolvido por:
Patrick Silva Ferraz
João Victor Dias Costa
- Finalidade:
Consulta e aplicação dos conceitos ao projeto de iniciação científica
- Orientador:
Leard de Oliveira Fernandes
- Instituição:
UESC - Universidade Estadual de Santa Cruz
- Localidade:
Ilhéus-Ba, Brasil
- Data:
21/10/2016

Referências

- Livro texto:
 - CHACON, S. ***Pro Git***. 1St ed. Apress, 2009. 288p.
- <https://help.github.com/categories/github-pages-basics/>