Git

Wiki de git. Por que ninguém consegue decorar tudo.

Introdução

Usos do git

- Git serve para versionar informações
- Git serve para recuperar informação
- Trabalhar em múltiplas tarefas concorrentemente
- Trabalhar com múltiplas equipes concorrentemente
- Pode-se versionar todo tipo de informação: projetos de software, livros, tcc's, etc.

História

Git foi criado em 2005 por Linus Torvalds, o criador do Linux. A motivação de criar o Git foi porquê o CVS anterior (SVN) era muito lento para comportar o trabalho do kernel.

Alguns significados para o termo: - "global information tracker" - "goddamn idiotic truckload of sh*t"

Pontos-chave no design:

- Velocidade
- Design Simples
- Suporte a desenvolvimento não-linear (branches)
- Totalmente distribuído
- Capaz de lidar com projetos gigantes

Concorrentes

- Subversion
- Perforce

Lista de Comandos

Git conta com vários comandos, a lista completa se encontra em /lib/git-core.

Inicializando um projeto

Ajuda

Para ajuda genérica pode-se utilizar

```
git --help
man git
```

Para mais detalhes do que como cada comando opera pode utilizar

```
man git commando
git commando --help
```

Init

Para inicializar um projeto git

São criados arquivos na pasta .git com os dados versionados.

```
git init $DIRETORIO
```

Para uma lista dos arquivos criados pelo git:

```
cd /tmp
git init foo
find .
```

Config

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Criando Histórico

Status

Git status dá informações de como está seu repositório. Muito útil ara saber qual a próxima coisa a se fazer.

```
git status
git status -s
```

Add

Git add adiciona arquivos a uma área temporária para compor um commit (staging).

```
git add $ARQUIVO|$DIRETORIO
```

Commit

Cada mudança no histórico de um projeto é representado por um commit. `git show` mostra o último commit.

Para uma lista completa dos commits use: `git log`.

```
git commit
```

O commit transfere as informações para o repositório local.

Para pular a fase de staging em arquivos já existentes:

```
git commit -a
```

Consultando o histórico

Git log

```
git log
```

Gitk

```
gitk
```

ou

```
gitk nome_do_arquivo
```

Removendo arquivos

```
git rm --cached foo.txt//staging
git rm foo.txt
```

Gitignore

```
site/*
*.pd
*.doc
presentation.pdf
.DS_Store
*.un~
*.disabled
Backend/data/tmp*
*.eml
Backend/config/autoload/*local.php
```

Renomeando arquivos

```
git mv REDME.md README
```

é equivalente a

```
mv README.md README
git rm README.md
git add README
```

Boas Práticas de commits

Nomenclatura de commits

If you force good commit practices consistently, you will be able to drive the engineering culture and the code itself to a better state.

- Escreva na forma imperativa. Ex: *ajuste de estilo no formulário X* ao invés de *ajustado de estilo no formulário X*
- Se é difícil dar nomes talvez seja melhor quebrar o commit antes.
- Mais commits é melhor que menos commits.

- É interessante colocar o número da issue no commit para ajudar a minerar o histórico
- Commits não deveriam quebrar o build (serem atômicos).
- Commits de funcionalidade não devem conter mudança de estilo, espaçamento, etc
- Commits não deveriam necessitar mais de 5 a 10 minutos para serem compreendidos e revisados

Referências

- https://en.wikipedia.org/wiki/Atomic_commit
- https://www.alexkras.com/19-git-tips-for-everyday-use/#good-commit-message
- https://kernelnewbies.org/UpstreamMerge/MergingStrategy
- http://sethrobertson.github.io/GitBestPractices/
- http://stackoverflow.com/questions/273695/git-branch-naming-best-practices

Branches

Uma branch é uma linha de trabalho independente. Podem ser usadas para diversos propósitos.

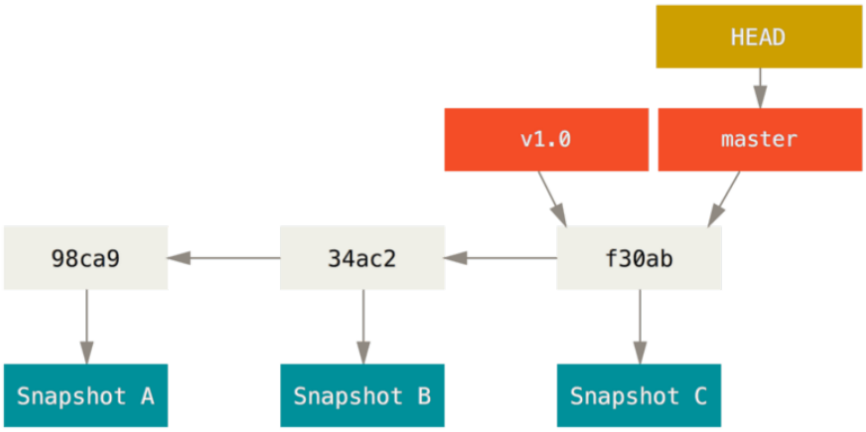
Pode-se ter uma branch para:

- experimentar uma tecnologia nova;
- uma branch para um bug-fix;
- outra para o trabalho do sprint;

A branch padrão no git é a `master`. Para listar todas as branches use: `git branch`

Branches no git são baratas

A branch atual é aquela apontada pelo objeto HEAD.



Objetos que compõem o histórico

```
cat .git/HEAD
```

Listando branches

```
git branch
git branch -v
git branch --merged
git branch --no-merged
```

Criando branches

```
git branch nova_branch
```

Movendo para a branch

```
git checkout nova_branch
```

Usando branches

```
###da master
git checkout -b "recursos_adicionais" #cria outra linha de trabalho
###adicionados recursos adicionais
git commit "recursos adicionais x,y,z"
git checkout -b "recursos_adicionais_extras"
###mais recursos adicionais
```

Merge

Merge mescla o conteúdo de branches

```
git merge recursos_adicionais
git log
git log [branch_name]
```

Fast-forward: move o ponteiro da master para o último commit da branch. Possível utilizar quando não há divergências entre as branches

```
git merge
```

Deletando branch

```
git branch -d branch_name
```

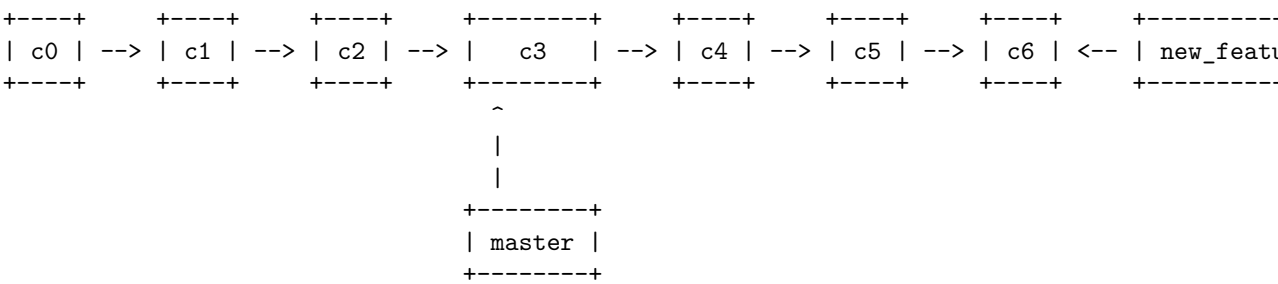
Merges

Fast-forward

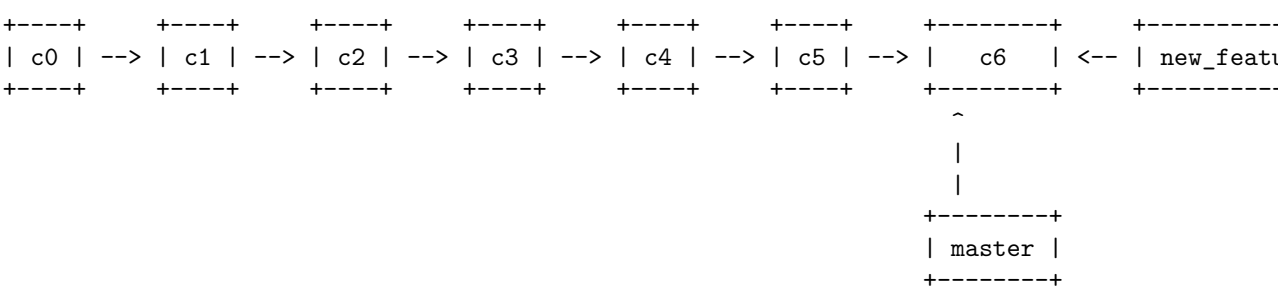
Acontece quando o histórico do merge está a frente do HEAD atual, então move-se apenas o ponteiro da branch.

Muda o ponteiro do commit.

Antes



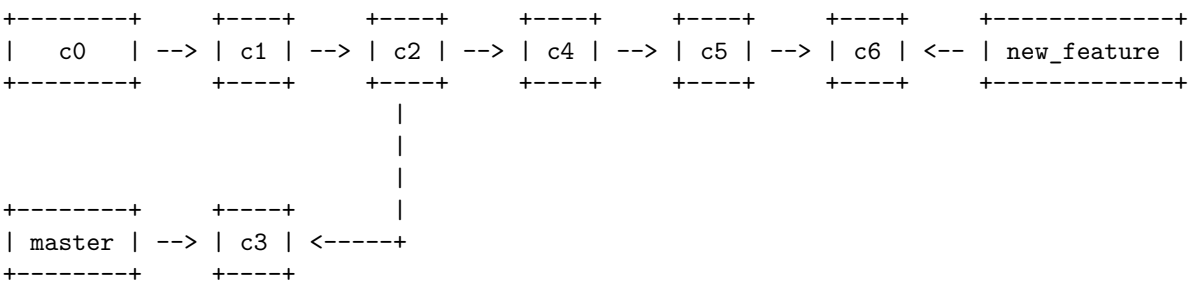
Depois



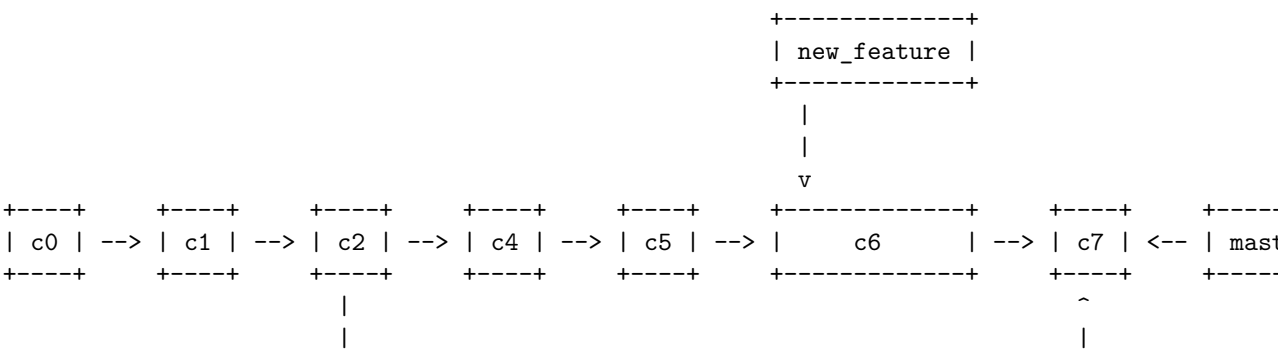
Recursive

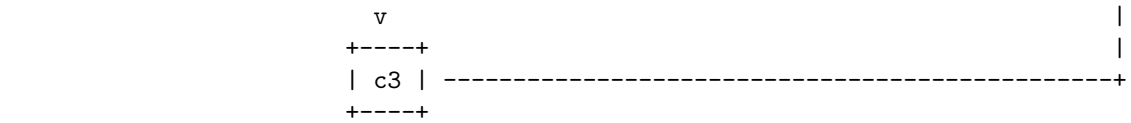
É o método usado quando as modificações são conflitantes. Usando os snapshots das duas branches e o ancestral comum dos dois. É criado um commit com dois pais.

Antes



Depois





Pseudo algoritmo

- Encontre um commit base (c2) que é ancestral de ambas as versões (c3, c6)
- Executa diffs entre c3 e c2 e entre c6 e c2.
- Percorre os blocos de mudança identificados nos diff's.
- Se ambas as versões introduziram a mesma modificação no mesmo lugar aceita uma delas.
- Se uma versão introduz uma modificação e a outra não mexe no mesmo lugar adiciona-se a modificação
- Se ambas as versões introduzem modificações diferentes no mesmo lugar marca-se a área como conflitante e pede-se para o usuário corrigir.

Remotos

Remoto no filesystem local

```
cd /path/to/git-docs
cd ..
git clone git-docs other-git-docs
cd other-git-docs
git config user.name "other user"
git config user.email other.user@gmail.com
git ls-remote

//do some work
cd ../git-docs
git remote add other ../other-git-docs
git merge other/master
```

Remoto online

Listar os repositórios remotos

```
git remote -v
```

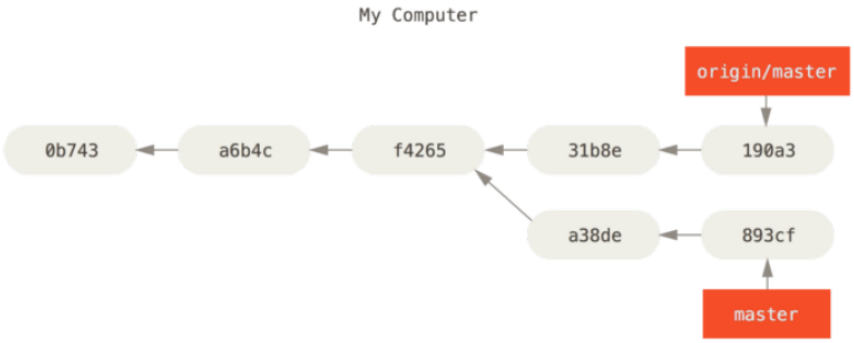
Inspecionando remoto

```
git remote show
git remote show origin
git ls-remote
```

Branches remotas

Usar o padrão [remoto/branch]

```
git show origin/master
```



Git fetch não faz merge

Git remote add origin

```
git remote add origin git@github.com:compufour/compufacil.git
git remote add origin https://github.com/user/repo.git
```

Mandar para o repositório

```
git clone https://github.com/JeanCarloMachado/git-docs
git push origin new_branch
```

Baixar as modificações remotas no local

```
git fetch origin
```

Pull

O *git pull* faz um fetch mais um merge.

```
git pull origin master
```

Setando remoto e branch padrões

Permite usar apenas `git push`, ao invés de `git push origin master`.

```
git branch --set-upstream-to myfork/master
```

Começando trabalho a partir de uma branch remota

```
git checkout nome_da_branch_remota
ou
git checkout -b branch_remota origin/branch_remota
```

Listar as braches sincronizadas com o servidor

```
git branch -vv
```

SSH

Criando chaves

```
cd ~/.ssh
ssh-keygen
```

Adicionando chaves

```
cat ~/.ssh/id_rsa.pub | copy
```

Credenciais https

```
git config --global credential.helper 'cache --timeout=3600'
```

Fluxos de trabalho

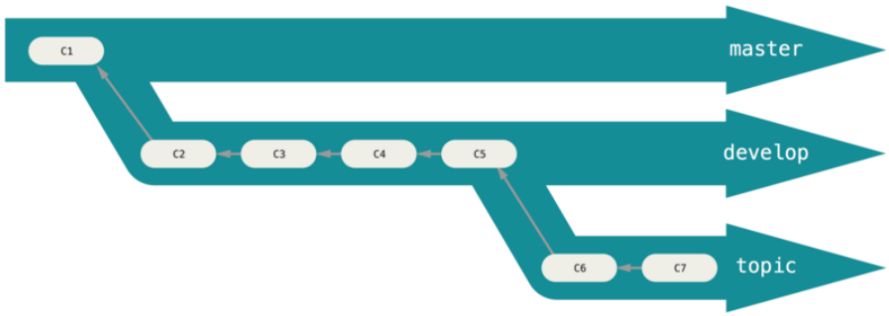
Modelos de versionamento

- Hierárquico
- Centralizado
- Distribuído

Git flow

Mais adequado para software em versões.

- master
- develop
- topic
- [pu]
- [hotfix]
- [release]



Git flow

Githubflow

Mais adequado para entrega contínua

- Cria-se uma branch

- Modifica-se conteúdo
- Envia-se um pull request
- Revisa-se o conteúdo e faz-se alterações no PR
- Faz-se o merge do pull request para master
- entrega-se a nova versão

Git flow vs Github flow

Estudando fluxos de projetos open-source

Kernel

- Mailing list + patches
- Mantenedores de sub-sistemas

PHP

- Github
- Pull-requests
- Contributing

Node

- Github
- Pull-requests
- Contributing
- Collaborator Guide

Mais sobre o assunto

- https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows#_distributed_git
- https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows
- https://lucamezzalira.com/2014/03/10/git-flow-vs-github-flow/
- http://scottchacon.com/2011/08/31/github-flow.html
- https://lucamezzalira.com/2014/03/10/git-flow-vs-github-flow/

Github e Gitlab

Github

Permite

- Criar repositórios
- Colaborar para repositórios existentes
- Gerenciar projetos
- Integrar ferramentas de terceiros

Outros Serviços

- Githubio Pages: permite hospedar um site com seu usuário.
- Gists: equivalente ao pastebin

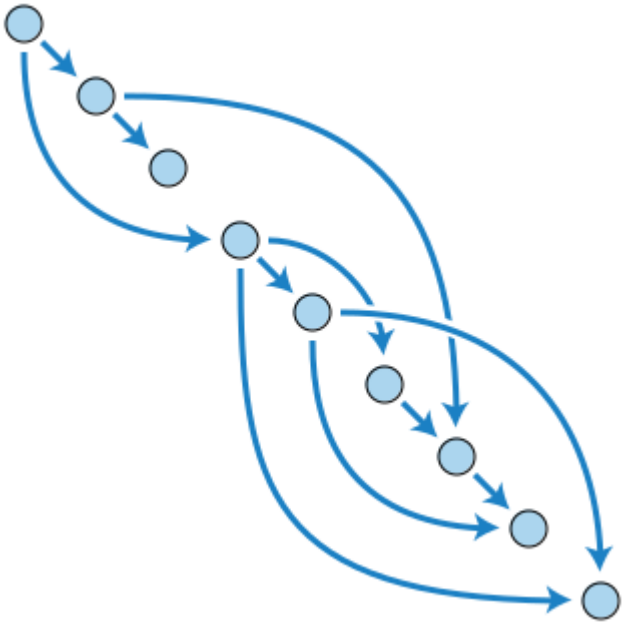
Gitlab

Permite fazer as mesmas coisas que o github Merge request / Pull request

Conceitos

Histórico em grafo

Os commits no git são estruturados em forma de grafo

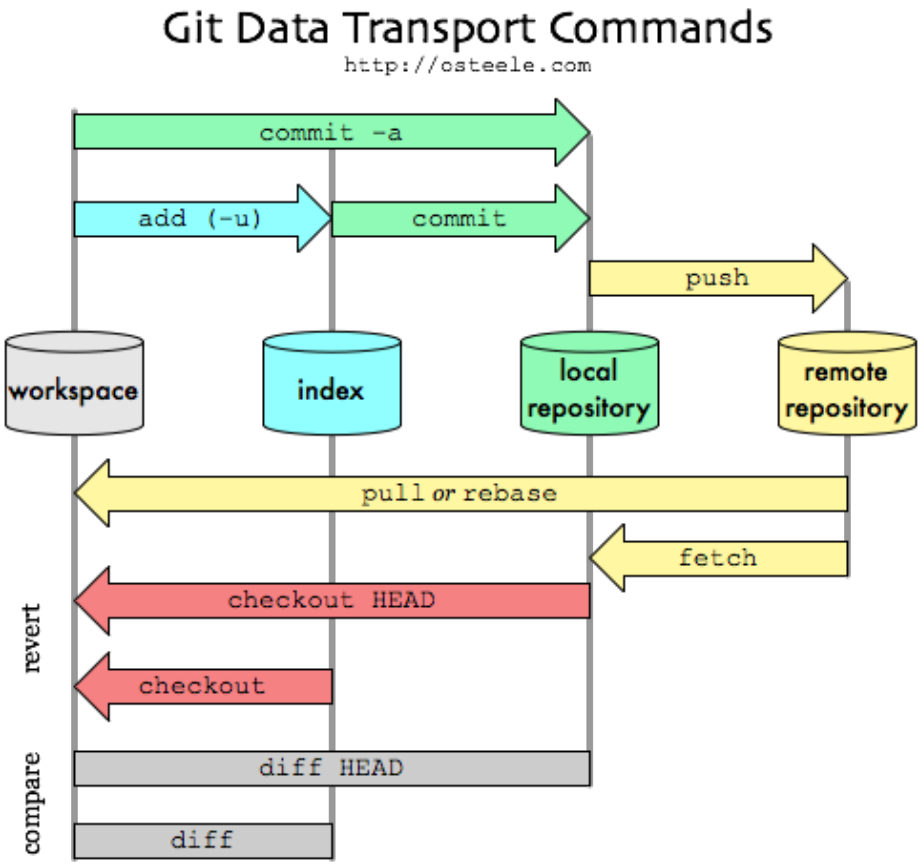


Grafo acíclico

Áreas de armazemanto do git

Git tem 4 áreas de armazenamento principais

- Área de trabalho
- Staging (index)
- Repositório Local
- Repositório Remoto



Comandos de trasporte e áreas de armazenamento

Packfiles

São arquivos "otimizados" para remover tamanho desenecessários do sistema.

```
$ git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx
```

Todo commit é uma hash sha1 e muda conforme o pai for reescrito.

HEAD

É a última versão da branch atual. Utilizada pelo comando git commit para ser o pai do novo commit.

Recursos Adicionais

Links

- Referência oficial
- Melhor tutorial de Git
- Encontrando issues no gitub
- Git para cientistas da computação
- Novidades do gitlab
- Markdown ref 1
- Markdown ref 2

Livros

- Git Pro
- Pragmatic Version Control Using Git - Pragmatic bookshelf