Table of Contents

## Home

# Git "the stupid content tracker"

Wiki de git. Por que ninguém consegue decorar tudo.

* Pra que git serve?
* Historia
* Lista de comandos
* Conceitos do git

## História

## Motivo

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

## Comandos

## Lista de Comandos

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

* git
* add
* add--interactive
* am
* annotate
* apply
* archimport
* archive
* bisect
* bisect--helper
* blame
* branch
* bundle
* cat-file
* check-attr
* check-ignore
* check-mailmap
* checkout
* checkout-index
* check-ref-format
* cherry
* cherry-pick
* citool
* clean
* clone
* column
* commit
* commit-tree
* config
* count-objects
* credential
* credential-cache
* credential-cache--daemon
* credential-gnome-keyring
* credential-libsecret
* credential-store
* cvsexportcommit
* cvsimport
* cvsserver
* daemon
* describe
* diff
* diff-files
* diff-index
* difftool
* difftool--helper
* diff-tree
* draw
* fast-export
* fast-import
* fetch
* fetch-pack
* filter-branch
* fmt-merge-msg
* for-each-ref
* format-patch
* fsck
* fsck-objects
* gc
* get-tar-commit-id
* grep
* gui
* gui--askpass
* hash-object
* help
* http-backend
* http-fetch
* http-push
* imap-send
* index-pack
* init
* init-db
* instaweb
* interpret-trailers
* log
* ls-files
* ls-remote
* ls-tree
* mailinfo
* mailsplit
* merge
* merge-base
* merge-file
* merge-index
* merge-octopus
* merge-one-file
* merge-ours
* merge-recursive
* merge-resolve
* merge-subtree
* mergetool
* mergetool--lib
* merge-tree
* mktag
* mktree
* mv
* name-rev
* notes
* p4
* pack-objects
* pack-redundant
* pack-refs
* parse-remote
* patch-id
* prune
* prune-packed
* pull
* push
* quiltimport
* read-tree
* rebase
* rebase--am
* rebase--interactive
* rebase--merge
* receive-pack
* reflog
* remote
* remote-ext
* remote-fd
* remote-ftp
* remote-ftps
* remote-http
* remote-https
* remote-testsvn
* repack
* replace
* request-pull
* rerere
* reset
* revert
* rev-list
* rev-parse
* rm
* send-email
* send-pack
* shell
* sh-i18n
* sh-i18n--envsubst
* shortlog
* show
* show-branch
* show-index
* show-ref
* sh-setup
* stage
* stash
* status
* stripspace
* submodule
* submodule--helper
* subtree
* svn
* symbolic-ref
* tag
* unpack-file
* unpack-objects
* update-index
* update-ref
* update-server-info
* upload-archive
* upload-pack
* var
* verify-commit
* verify-pack
* verify-tag
* web--browse
* whatchanged
* worktree
* write-tree
* mergetools

## Pra que serve

##### 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.

## 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

## 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

##### Gitk

gitk

ou

gitk nome\_do\_arquivo

##### Git log

git log

## Exercício

Crie um projeto usando o mkdocs  
Crie um arquivo resumindo como inicilizar um projeto git  
Comite esse arquivo  
Crie um arquivo resumindo como utilizar o commit  
Comite esse arquivo  
Consulte a versão anterior com o gitk

##### Setup mkdocs

$ pip install mkdocs  
$ mkdocs new git-docs  
$ cd git-docs  
$ mkdocs serve  
$ browser http://localhost:8000

## Removendo arquivos

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

## Renomeando arquivos

git mv REDME.md README

é equivalente a

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

## Branches

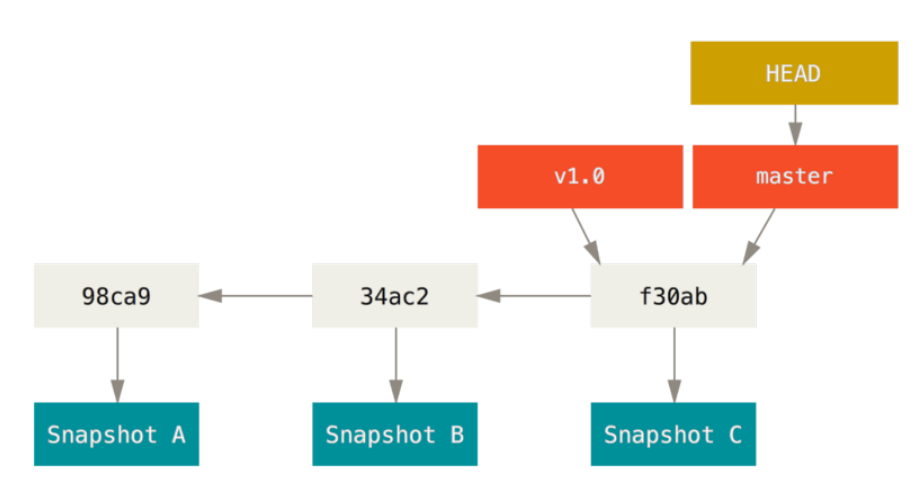
Uma branch é uma linha de trabalho independente. Podem ser usadas para diversos propósitos.

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



Objetos que compõem o histórico

A branch atual é aquela apontada pelo objeto HEAD.

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

git commit "informações sobre branches no master"  
git checkout -b "recursos\_adicionais" #cria outra linha de trabalho

O comando checkout serve para criar a branch e entrar nela.

##### Merge

Merge mescla o conteúdo de branches

git merge recursos\_adicionais  
git log

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

## Exercícios

Faça algumas modificações em uma branch de nova feature  
Volte para a master e faça uma nova branch de hotfix e trabalhe  
nela  
  
Após validar as modificações faça merge da branch de hotfix  
Volte a trabalhar na branch de nova feature  
Tragas as modificações para a master

## Merges

## Fast-forward

Muda o ponteiro do commit

## Recursive

Cria um commit adicional com dois ou mais pais

## Conflitos

git status

git mergetool

## Patches

## Format patch

Patches são mudanças que podem ser transferidas de um repositório para outro:

Para criar um patch do último commit:

git format-patch master

## Appy

Para aplicar um patch:

curl -L https://goo.gl/p1LEc7 -o 0001-historia.patch  
git apply 0001-historia.patch

## Diff e show e log

O resultado destes comandos também são compatíveis com patches do git.

git diff > my\_patch.patch  
git show > my\_patch.patch  
git diff --cached > my\_patch.patch  
git log -p

## Criar um patch de uma branch

git format-patch master

## Exercícios

Altere o conteúdo do seu projeto em uma nova branch, commite e  
conteúdo e gere um patch da branch.  
  
Em outra branch aplique as modificações.

## Recuperando

## Recuperando na área de trabalho

git log  
rm -rf docs  
browser http://localhost:8000  
git reset --hard HEAD  
browser http://localhost:8000

## Recuperando versão X do histórico

git checkout commit X  
git checkout -b "nova\_branch"  
git checkout master  
git merge nova\_branch

## Encontrar commits com string X

git log -SX

## Desfazendo

##### Remover do staging o que ainda não está comitado

git reset HEAD

ou

git reset HEAD file.txt

##### Remover arquivos criados mas não versionados

git clean

## Desfazer modificações não comitadas

git checkout -- filename

##### Trocar a última mensagem de commit

git commit --amend

##### Adicionando um arquivo ao commit anterior

git add file\_x.md  
git commit --amend

##### Revert

Desfaz um commit criando um novo commit. Recomendado em branches públicas onde o histórico não pode ser reescrito.

git checkout hotfix  
git revert HEAD~2

##### Desfazendo últimos 3 commits - sem desfazer o código

git reset --soft HEAD~3

##### Desfazendo últimos 3 commits - desfazendo o código

git reset --hard HEAD~3

## 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

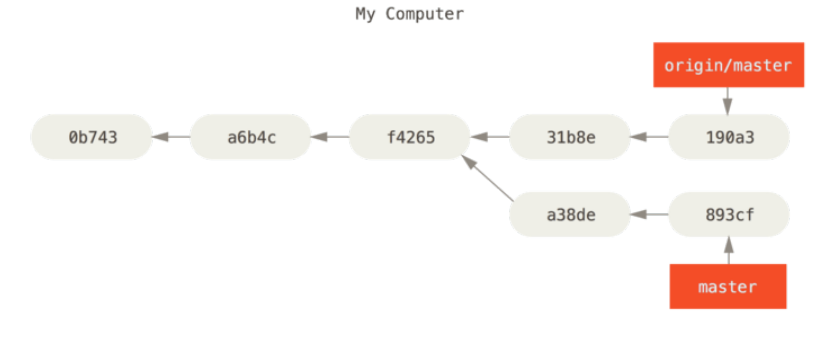
## Inspecionando remoto

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'

## Exercícios

Crie uma conta no github  
Crie um repositório para o projeto do git  
Envie as modificações para lá via http  
Configure suas chaves ssh  
Envie novas modificações via ssh

## Github & Gitlab

## Github

GitHub é uma rede social.

##### Permite

* Criar repositórios
* Colaborar para repositórios existentes:
  + Criando issues
  + Submitando pull requests

##### Outros Serviços

###### Githubio Pages

Hospedar um site com seu usuário.

###### Gists

Tipo pastebin

## Gitlab

Permite fazer as mesmas coisas que o github

## Boas Práticas

## Nomenclatura de commmits

Escreva na forma imperativa:

* *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

##### Referências

https://www.alexkras.com/19-git-tips-for-everyday-use/#good-commit-message

## Fluxos de trabalho

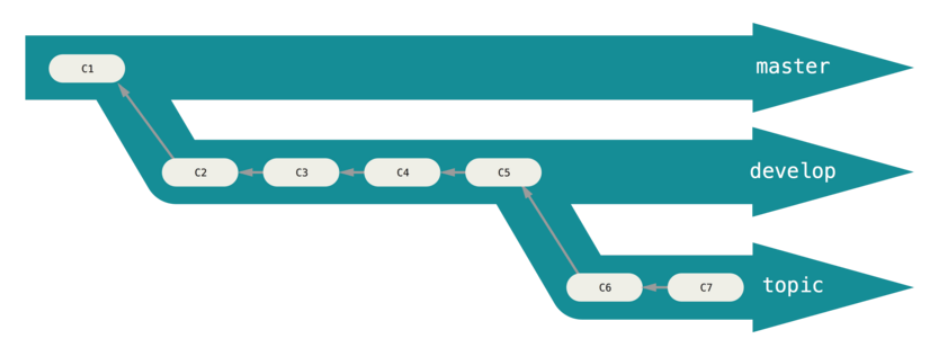
## Modelos de versionamento

* Hierárquico
* Centralizado
* Distribuído

## Git flow

Mais adequado para software em versões.

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



Git flow

## [Githubflow](https://help.github.com/articles/github-flow/)

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

## 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

## Customizando

As configurações globais ficam em

~/.gitconfig

Configurações para todos usuários podem ir em

/etc/gitconfig

Em repositórios

```sh .git/config

Para listar as configurações:

git config --list

```

## Cores

git config --global color.ui true

## Setar o editor padrão

git config --global core.editor vim

## Trocar template de mensagem de commit

git config --global commit.template vim

## Pager

git config --global core.pager 'less'

## Aliases

git config --global alias.co checkout  
git config --global alias.unstage 'reset HEAD --'

## Bash aliases

alias amend='git commit --amend'  
alias b='git branch'  
alias branch='git branch'  
alias check='export PREV\_BRANCH=`cb` && git checkout'  
alias cm='git commit'  
alias cm='git status && git add . && git commit -m "'  
alias g='git'  
alias ga='git add'  
alias gac='git-add-commit'  
alias gb='git branch'  
alias gbi='git bisect'  
alias gc='git commit'  
alias gck='git checkout'  
alias gckm='git checkout master'  
alias gckm='git checkout master'  
alias gdf='git diff'  
alias gdfn='git --no-pager diff --name-only'  
alias gf='git fetch'  
alias gfc='git fetch && git checkout'  
alias gh='git rev-parse HEAD'  
alias git-manual='chromium /home/jean/projects/git/Documentation/user-manual.html'  
alias git-show-files='git diff-tree --no-commit-id --name-only -r'  
alias git='hub'  
alias gl="git log --pretty=oneline"  
alias gm='git merge'  
alias gp='git push'  
alias gr='git remote'  
alias grh='git reset --hard HEAD'  
alias grhh='git reset --hard HEAD'  
alias gs='git status -s'  
alias gt='git tag'  
alias lcb='git pull origin `cb`'  
alias lom='echo "Consider using: [git pull --rebase] instead" ; git pull origin master'  
alias lr='git pull compufacil master --rebase'  
alias pcb='git push origin `cb`'  
alias status='git status'

##### Funções

current\_branch() {  
 git branch 2> /dev/null | grep "\*" | cut -d" " -f2  
}  
  
commit\_diff () {  
 git diff $1~ $1  
}  
  
not\_committed\_files () {  
 git ls-files --others --exclude-standard  
}  
  
commit\_count () {  
 git log --pretty=oneline | wc -l  
}  
  
  
last\_diff\_file() {  
 last\_diff=${2:-1}  
 file\_name=$1  
 git log -p --follow -$last\_diff $file\_name  
}

##### Logs

Customizar logs é uma arte.

git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"  
git config --global alias.lol "log --graph --decorate --pretty=oneline --abbrev-commit --all"  
git config --global alias.mylog "log --pretty=format:'%h %s [%an]' --graph"

###### Opções

--decorate mostra as braches apontando para determinados commits

## Ferramentas Auxiliares

# Ferramentas & Produtividade

## Stash

Para salvar trabalhos temporários

git stash  
git stash apply

##### Visualizar o stash

git stash show -p

## Gitk

Uma interface nativa para quem gosta de interfaces gráficas

## Git on GUI

[Source tree](https://www.sourcetreeapp.com/)

##### Tags

Tags anotadas guardam o commit, quem comitou e uma descrição

git tag -l "v1.\*"  
git tag -a v1.1 -m "informações adicionais" [commitId]  
git show v1.1

Uma tag leve não guarda informações adicionais a não ser o commit

git tag -l "v1.\*"  
git tag v1.2 [commitId]  
git show v1.2

##### Enviando pro server

git push origin v1.2  
git push origin --tags

## Criando um servidor git

## Criando um server remoto

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

## Bare

Não tem working dir.

cd /tmp/  
mkdir foo  
git init . --bare  
git clone root@45.55.247.185:/tmp/foo /tmp/foo\_clone

## Submódulos

## Criando um submódulo

git submodule add https://github.com/chaconinc/DbConnector  
git submodule add git@github.com:jeanCarloMachado/jeancarlomachado.github.io.git  
git submodule add ../mainrepo.git submoduleDir  
  
git status

## Buscando os arquivos

git submodule init  
git submodule update

## Inicializando os submódulos em um repositório já configurado

git clone --recursive

## Mantendo atualizado

git config -f .gitmodules submodule.githubiobranch master  
  
git submodule update --remote

## Foreach

git submodule foreach "git add . ; git commit -m 'added assets'"

## Push

git config push.recurseSubmodules on-demand  
git push

## Diff/Log

git $COMANDO --submodule

## Operadores Relativos

## Operadores relativos

HEAD@{5}  
HEAD@{yesterday}  
HEAD@{2.months.ago}

##### Sintaxe alternativa

git rev-parse 34ab345c70~2  
git rev-parse HEAD~1  
HEAD~5

##### Topo da branch atual

HEAD  
HEAD~ == HEAD~1

##### Referência a ancestrais

git rev-parse HEAD^  
git rev-parse 2aad^^

^ significa o primeiro pai do primeiro pai do primeiro pai

###### Para funcionar o (^) nos zsh e bash sem necessitar escapar

setopt NO\_NOMATCH  
setopt NO\_EXTENDED\_GLOB

## Resolvendo Conflitos

## Resolvendo com nossas alterações

git pull origin master -X ours

## Mostrando o conteúdo de um commit

git show commitId

## Tipos de estratégia de merge:

* fast-forward
* recursivo
* ???

## Resolvendo conflitos frequentes

git rerere

## Reescrevendo Histórico

## Rebase

Coloca o histórico da branch atual no topo da branch target.

git rebase target

## A regra de ouro do rebase

https://www.atlassian.com/git/tutorials/merging-vs-rebasing?section=the-golden-rule-of-rebasing

## Squash

git rebase -i

Ou

git merge --squash feature\_branch

## Editar um commit anterior

* Rebase no commit e marcar ele como editado
* Comitar as alterações
* Fazer um fix up no commit com seu pai

git rebase -i HEAD~3

## Criando dois commits a partir de 1

rebase iterativo com edit  
git reset HEAD^  
... add commit 2x  
rebase --continue

## Removendo arquivos do último commit

git rm file\_name

## Debug

## Blame

Mostra as últimas alterações de cada linha de arquivo com o autor.

$ git blame -L 141,153 simplegit.rb

Git não guarda renames explicitamente. Quando passado o -C git tenta identificar de onde o código veio.

$ git blame -C GITPackUpload.m

## Bisect

Ferramenta para descobrir qual commit quebrou uma feature.

git bisect start  
git bisect bad commitid  
git bisect good commitid

git bisect reset

Automate bisect:

$ git bisect start HEAD v1.0  
$ git bisect run test-error.sh

## Manutenção

## Removendo arquivos do histórico

git filter-branch --tree-filter 'rm -rf passwords.txt' HEAD.

git filter-branch --force --index-filter \  
'git rm --cached --ignore-unmatch PATH-TO-YOUR-FILE-WITH-SENSITIVE-DATA' \  
--prune-empty --tag-name-filter cat -- --all

## Fazendo garbage collection "packfiles"

git reflog expire --expire-unreachable=now --all  
git gc --prune=now

## Removendo branches já trabalhadas

git branch --merged | egrep -v "(^\\*|master|dev)" | xargs git branch -d

## Recuperando dados

Git só deleta objetos quando você faz um git gc.

## Encontrando objetos soltos

git fsck --unreachable

## Pumbling

Pumbling é o complemento da "porcelain". Que é um conjunto de comandos que existem para compor outros comandos.



Internal Structure

## Criando objeto

echo "test content" | git hash-object -w --stdin  
  
find\_date  
cat $(run\_function object\_path d670) | zlib-flate -uncompress  
  
  
git hash-object -w test.txt

## Cat-file

##### Retornar o tipo

git cat-file -t object\_id

##### Retornar o conteúdo

git cat-file -p object\_id

## Atualizando o index

git update-index --add --cacheinfo 100644 objectHash fileName.txt  
git write-tree

## Adicionando árvores aninhadas

git read-tree --prefix=nomeDaPasta d8329fc1cc938780ffdd9f94e0d364e0ea74f579

## Comitando

echo "first commit message" | git commit-tree treeHash

##### Commit com pai

cho "commit second" | git commit-tree 0155eb4229851634a0f03eb265b69f5a2d56f341 -p 035de1de9c497bdcf9bbbdbaab8e5902e3711ce2

## Criando branch

echo "2016cc9fb2a892886946fe2ab4354ec9d40f181d" > .git/refs/heads/master

## Criando a HEAD

echo "ref: refs/heads/master" > .git/HEAD

## Criando um objeto manualmente

## Visualizar o conteúdo comprimido

cat .git/objects/28/e697e796c79e4c86fcb62e5319a43dfcf1fb83 | zlib-flate -uncompress

## Tipos de objetos

* blob
* tree
* commit
* tag

## A árvore mais recente

git cat-file -p master^{tree}

## Commit iterativo

irb  
content = "what is up, doc?"  
header = "blob #{content.length}\0"  
require "digest/sha1"  
store = header + content  
sha1 = Digest::SHA1.hexdigest(store)  
require 'zlib'  
zlib\_content = Zlib::Deflate.deflate(store)  
path = '.git/objects/' + sha1[0,2] + '/' + sha1[2,38]  
require 'fileutils'  
FileUtils.mkdir\_p(File.dirname(path))  
File.open(path, 'w') {|f| f.write zlib\_content}

## Inspecionando objetos

cat .git/objects/28/e697e796c79e4c86fcb62e5319a43dfcf1fb83 | zlib-flate -uncompress

## Comandos Menos comuns

## Rev-parse

Retorna o commit que algo está apontando.

git rev-parse master  
git rev-parse HEAD  
cat .git/HEAD

## Reflog

Mostra um log dos comandos usados no repositório. Similar ao .bash\_history (para operações locais)

git reflog  
  
ou  
  
git log -g

## Cherry-Pick

Permite aplicar um único commit de outra branch.

git cherry-pick

## Archive the repository

git archive --format zip --output /full/path/to/zipfile.zip master  
git archive master --format=tar --output=../website-12-10-2012.tar

## Bundle

git bundle create ../repo.bundle master  
git bundle unbundle ../repo.bundle master

Mantém o histórico.

## Merge de múltiplas branches

octopus merge

ours http://stackoverflow.com/questions/16208144/how-do-i-merge-multiple-branches-into-master

## Staging parcial

git add -p

## GUI para fazer merge

git mergetool

## Git grep

##### Procura pro conteúdo de commits

git grep -i "linux" $(git rev-list --all )  
git grep <regexp> $(git rev-list --all -- lib/util) -- lib/util

## Remover arquivos não versionados

git clean -f

## Últimas N alterações em um arquivo

git log -p --follow -2 Backend/module/Api/src/Api/Service/ErrorProcessor.php

## Diferença entre branches

##### Todos os commits que estão na branch develop mas não estão na master

git log master..experiment == git log ^master experiment == git  
log experiment --not master

##### Lista de objetos

git rev-parse issue\_2..master  
  
4f298373ba0f536115051593bd149539d367937d  
^2f35c9c5cc4ad413970d666d0f90b66d3cbf89c7  
<0:git-course:/home/jean/projects/git-docs:master:> git rev-parse master..issue\_2  
  
2f35c9c5cc4ad413970d666d0f90b66d3cbf89c7  
^4f298373ba0f536115051593bd149539d367937d  
<0:git-course:/home/jean/projects/git-docs:master:>

##### Que está em uma branch mas não está na outra (xor)

git log master...experimento

Está em um mas não no outro. --left-right mostra que lado cada um se encontra.

##### Log de outra branch

git log branch\_foo  
git log <branch> -- <path/to/file>

##### Que está no local mas não está na origin

git log origin/master..master --stat

##### Que está na master mas não está local

git log origin/master..master --stat

##### Who are the children of a commit

git rev-list --all --not 7ef306ff3ef3cfe694fbf3847f2c35c86067ee87^@ --children | grep "^7ef306ff3ef3cfe694fbf3847f2c35c86067ee87"

## Integracoes

## Integrações

Software de terceiros que agregam funcionalidade ao github [lista](https://github.com/integrations).

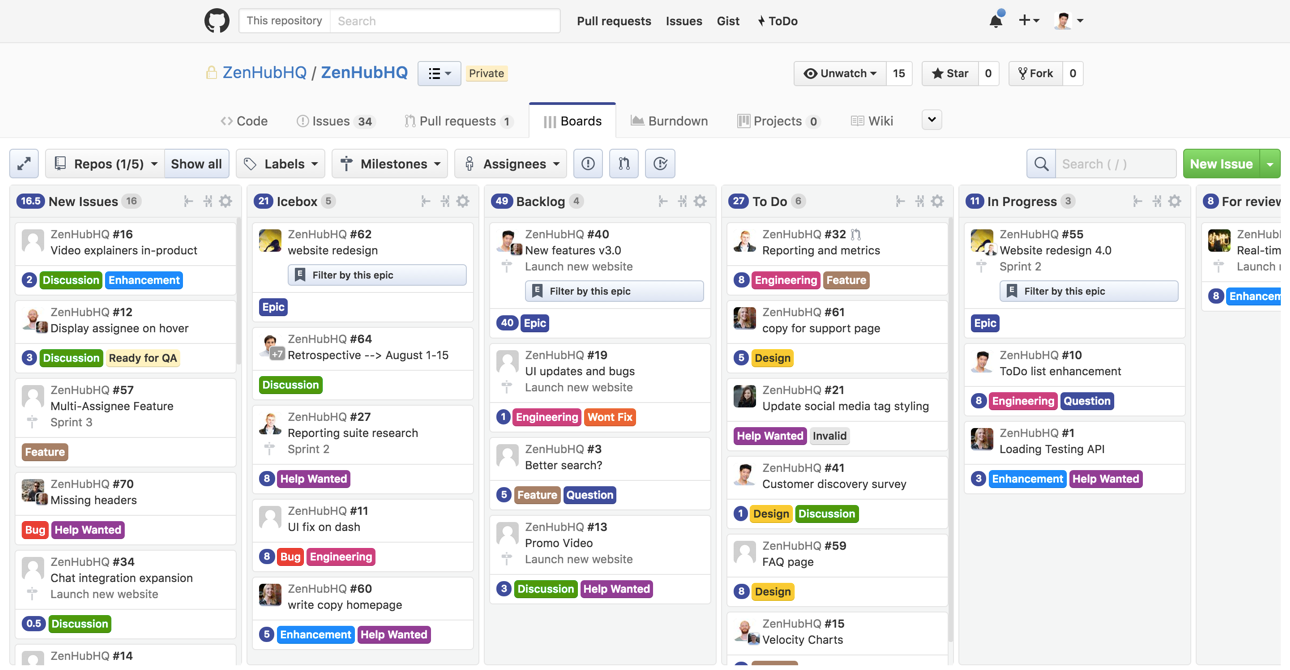
* Waffle
* Buildkite
* ZenHub

##### Codacy

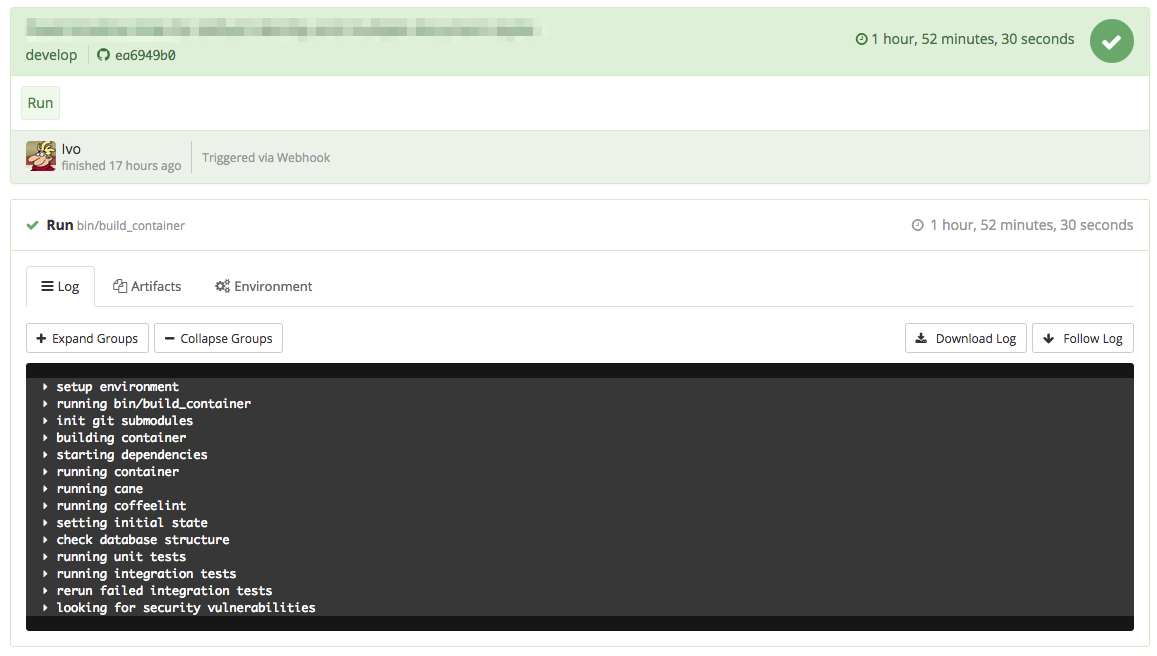
## Serviços

Executam ações quando eventos acontecem em outras ferramentas.

* Docker
* Deploy aws



zenhub



buildkite

# 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

Três estágios de informação

* Comitado
* Modificado
* Em staging

## Fast-forward merge

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

## Recursive merge

É um merge de 3 vias.

Usando os snapshots das duas branches e o ancestral comum dos dois.

É criado um commit com dois pais.

You might be best off looking for a description of a 3-way merge algorithm. A high-level description would go something like this:  
  
Find a suitable merge base B - a version of the file that is an ancestor of both of the new versions (X and Y), and usually the most recent such base (although there are cases where it will have to go back further, which is one of the features of gits default recursive merge)  
Perform diffs of X with B and Y with B.  
Walk through the change blocks identified in the two diffs. If both sides introduce the same change in the same spot, accept either one; if one introduces a change and the other leaves that region alone, introduce the change in the final; if both introduce changes in a spot, but they don't match, mark a conflict to be resolved manually.  
The full algorithm deals with this in a lot more detail, and even has some documentation (/usr/share/doc/git-doc/technical/trivial-merge.txt for one, along with the git help XXX pages, where XXX is one of merge-base, merge-file, merge, merge-one-file and possibly a few others). If that's not deep enough, there's always source code...

## 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](https://git-scm.com/docs)
* [Melhor tutorial de Git](http://rypress.com/tutorials/git/index)
* [Encontrando issues no gitub](http://www.jeancarlomachado.com.br/blog/findingissuesongithub.html)
* [Git para cientistas da computação](http://eagain.net/articles/git-for-computer-scientists/)

## Livros

* [Git Pro](https://git-scm.com/book/en/v2)
* [Pragmatic Version Control Using Git - Pragmatic bookshelf](https://pragprog.com/book/tsgit/pragmatic-version-control-using-git)