Table of Contents

## Home

# Git "the stupid content tracker"

Wiki de git. Por que ninguém consegue decorar tudo.

* Introducao
* Básico
* Avançado

## 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

## Lista de Comandos

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

* Git (3pm) - Perl interface to the Git version control system
* git (1) - the stupid content tracker
* git-add (1) - Add file contents to the index
* git-am (1) - Apply a series of patches from a mailbox
* git-annotate (1) - Annotate file lines with commit information
* git-apply (1) - Apply a patch to files and/or to the index
* git-archimport (1) - Import an Arch repository into Git
* git-archive (1) - Create an archive of files from a named tree
* git-bisect (1) - Use binary search to find the commit that introduced a bug
* git-blame (1) - Show what revision and author last modified each line of a file
* git-branch (1) - List, create, or delete branches
* git-bundle (1) - Move objects and refs by archive
* git-cat-file (1) - Provide content or type and size information for repository objects
* git-check-attr (1) - Display gitattributes information
* git-check-ignore (1) - Debug gitignore / exclude files
* git-check-mailmap (1) - Show canonical names and email addresses of contacts
* git-checkout (1) - Switch branches or restore working tree files
* git-checkout-index (1) - Copy files from the index to the working tree
* git-check-ref-format (1) - Ensures that a reference name is well formed
* git-cherry (1) - Find commits yet to be applied to upstream
* git-cherry-pick (1) - Apply the changes introduced by some existing commits
* git-citool (1) - Graphical alternative to git-commit
* git-clean (1) - Remove untracked files from the working tree
* git-clone (1) - Clone a repository into a new directory
* git-column (1) - Display data in columns
* git-commit (1) - Record changes to the repository
* git-commit-tree (1) - Create a new commit object
* git-config (1) - Get and set repository or global options
* git-count-objects (1) - Count unpacked number of objects and their disk consumption
* git-credential (1) - Retrieve and store user credentials
* git-credential-cache (1) - Helper to temporarily store passwords in memory
* git-credential-cache--daemon (1) - Temporarily store user credentials in memory
* git-credential-store (1) - Helper to store credentials on disk
* git-cvsexportcommit (1) - Export a single commit to a CVS checkout
* git-cvsimport (1) - Salvage your data out of another SCM people love to hate
* git-cvsserver (1) - A CVS server emulator for Git
* git-daemon (1) - A really simple server for Git repositories
* git-describe (1) - Describe a commit using the most recent tag reachable from it
* git-diff (1) - Show changes between commits, commit and working tree, etc
* git-diff-files (1) - Compares files in the working tree and the index
* git-diff-index (1) - Compare a tree to the working tree or index
* git-difftool (1) - Show changes using common diff tools
* git-diff-tree (1) - Compares the content and mode of blobs found via two tree objects
* git-fast-export (1) - Git data exporter
* git-fast-import (1) - Backend for fast Git data importers
* git-fetch (1) - Download objects and refs from another repository
* git-fetch-pack (1) - Receive missing objects from another repository
* git-filter-branch (1) - Rewrite branches
* git-fmt-merge-msg (1) - Produce a merge commit message
* git-for-each-ref (1) - Output information on each ref
* git-format-patch (1) - Prepare patches for e-mail submission
* git-fsck (1) - Verifies the connectivity and validity of the objects in the database
* git-fsck-objects (1) - Verifies the connectivity and validity of the objects in the database
* git-gc (1) - Cleanup unnecessary files and optimize the local repository
* git-get-tar-commit-id (1) - Extract commit ID from an archive created using git-archive
* git-grep (1) - Print lines matching a pattern
* git-gui (1) - A portable graphical interface to Git
* git-hash-object (1) - Compute object ID and optionally creates a blob from a file
* git-help (1) - Display help information about Git
* git-http-backend (1) - Server side implementation of Git over HTTP
* git-http-fetch (1) - Download from a remote Git repository via HTTP
* git-http-push (1) - Push objects over HTTP/DAV to another repository
* git-imap-send (1) - Send a collection of patches from stdin to an IMAP folder
* git-index-pack (1) - Build pack index file for an existing packed archive
* git-init (1) - Create an empty Git repository or reinitialize an existing one
* git-init-db (1) - Creates an empty Git repository
* git-instaweb (1) - Instantly browse your working repository in gitweb
* git-interpret-trailers (1) - help add structured information into commit messages
* git-log (1) - Show commit logs
* git-ls-files (1) - Show information about files in the index and the working tree
* git-ls-remote (1) - List references in a remote repository
* git-ls-tree (1) - List the contents of a tree object
* git-mailinfo (1) - Extracts patch and authorship from a single e-mail message
* git-mailsplit (1) - Simple UNIX mbox splitter program
* git-merge (1) - Join two or more development histories together
* git-merge-base (1) - Find as good common ancestors as possible for a merge
* git-merge-file (1) - Run a three-way file merge
* git-merge-index (1) - Run a merge for files needing merging
* git-merge-one-file (1) - The standard helper program to use with git-merge-index
* git-mergetool (1) - Run merge conflict resolution tools to resolve merge conflicts
* git-mergetool--lib (1) - Common Git merge tool shell scriptlets
* git-merge-tree (1) - Show three-way merge without touching index
* git-mktag (1) - Creates a tag object
* git-mktree (1) - Build a tree-object from ls-tree formatted text
* git-mv (1) - Move or rename a file, a directory, or a symlink
* git-name-rev (1) - Find symbolic names for given revs
* git-notes (1) - Add or inspect object notes
* git-p4 (1) - Import from and submit to Perforce repositories
* git-pack-objects (1) - Create a packed archive of objects
* git-pack-redundant (1) - Find redundant pack files
* git-pack-refs (1) - Pack heads and tags for efficient repository access
* git-parse-remote (1) - Routines to help parsing remote repository access parameters
* git-patch-id (1) - Compute unique ID for a patch
* git-prune (1) - Prune all unreachable objects from the object database
* git-prune-packed (1) - Remove extra objects that are already in pack files
* git-pull (1) - Fetch from and integrate with another repository or a local branch
* git-push (1) - Update remote refs along with associated objects
* git-quiltimport (1) - Applies a quilt patchset onto the current branch
* git-read-tree (1) - Reads tree information into the index
* git-rebase (1) - Reapply commits on top of another base tip
* git-receive-pack (1) - Receive what is pushed into the repository
* git-reflog (1) - Manage reflog information
* git-remote (1) - Manage set of tracked repositories
* git-remote-ext (1) - Bridge smart transport to external command.
* git-remote-fd (1) - Reflect smart transport stream back to caller
* git-repack (1) - Pack unpacked objects in a repository
* git-replace (1) - Create, list, delete refs to replace objects
* git-request-pull (1) - Generates a summary of pending changes
* git-rerere (1) - Reuse recorded resolution of conflicted merges
* git-reset (1) - Reset current HEAD to the specified state
* git-revert (1) - Revert some existing commits
* git-rev-list (1) - Lists commit objects in reverse chronological order
* git-rev-parse (1) - Pick out and massage parameters
* git-rm (1) - Remove files from the working tree and from the index
* git-send-email (1) - Send a collection of patches as emails
* git-send-pack (1) - Push objects over Git protocol to another repository
* git-shell (1) - Restricted login shell for Git-only SSH access
* git-sh-i18n (1) - Git's i18n setup code for shell scripts
* git-sh-i18n--envsubst (1) - Git's own envsubst(1) for i18n fallbacks
* git-shortlog (1) - Summarize 'git log' output
* git-show (1) - Show various types of objects
* git-show-branch (1) - Show branches and their commits
* git-show-index (1) - Show packed archive index
* git-show-ref (1) - List references in a local repository
* git-sh-setup (1) - Common Git shell script setup code
* git-stage (1) - Add file contents to the staging area
* git-stash (1) - Stash the changes in a dirty working directory away
* git-status (1) - Show the working tree status
* git-stripspace (1) - Remove unnecessary whitespace
* git-submodule (1) - Initialize, update or inspect submodules
* git-subtree (1) - Merge subtrees together and split repository into subtrees
* git-svn (1) - Bidirectional operation between a Subversion repository and Git
* git-symbolic-ref (1) - Read, modify and delete symbolic refs
* git-tag (1) - Create, list, delete or verify a tag object signed with GPG
* git-unpack-file (1) - Creates a temporary file with a blob's contents
* git-unpack-objects (1) - Unpack objects from a packed archive
* git-update-index (1) - Register file contents in the working tree to the index
* git-update-ref (1) - Update the object name stored in a ref safely
* git-update-server-info (1) - Update auxiliary info file to help dumb servers
* git-upload-archive (1) - Send archive back to git-archive
* git-upload-pack (1) - Send objects packed back to git-fetch-pack
* git-var (1) - Show a Git logical variable
* git-verify-commit (1) - Check the GPG signature of commits
* git-verify-pack (1) - Validate packed Git archive files
* git-verify-tag (1) - Check the GPG signature of tags
* git-web--browse (1) - Git helper script to launch a web browser
* git-whatchanged (1) - Show logs with difference each commit introduces
* git-worktree (1) - Manage multiple working trees
* git-write-tree (1) - Create a tree object from the current index
* BC (3x) - direct curses interface to the terminfo capability database
* bc (1) - An arbitrary precision calculator language
* bc (1p) - arbitrary-precision arithmetic language
* gvimdiff (1) - edit two, three or four versions of a file with Vim and show differences
* vimdiff (1) - edit two, three or four versions of a file with Vim and show differences

## 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.

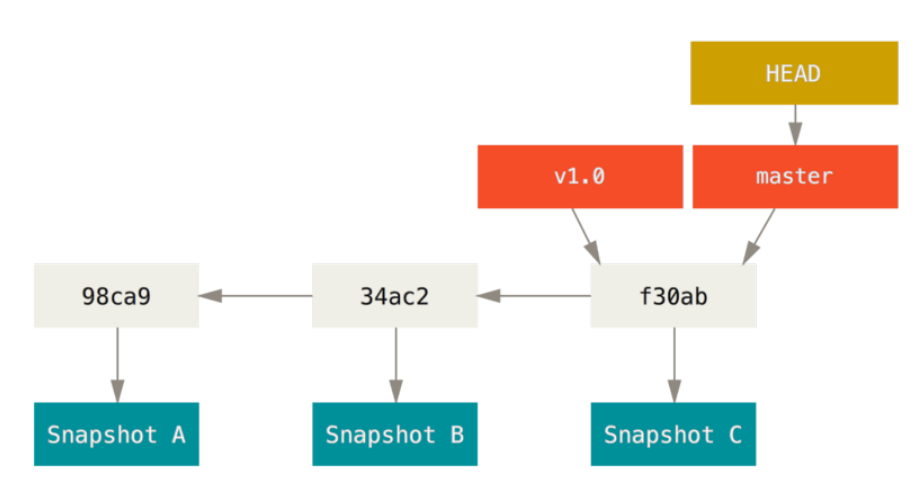
Pode-se ter uma branch para:

* 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

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

+----+ +----+ +----+ +--------+ +----+ +----+ +----+ +-------------+  
| c0 | --> | c1 | --> | c2 | --> | c3 | --> | c4 | --> | c5 | --> | c6 | <-- | new\_feature |  
+----+ +----+ +----+ +--------+ +----+ +----+ +----+ +-------------+  
 ^  
 |  
 |  
 +--------+  
 | master |  
 +--------+

##### Depois

+----+ +----+ +----+ +----+ +----+ +----+ +--------+ +-------------+  
| c0 | --> | c1 | --> | c2 | --> | c3 | --> | c4 | --> | c5 | --> | c6 | <-- | new\_feature |  
+----+ +----+ +----+ +----+ +----+ +----+ +--------+ +-------------+  
 ^  
 |  
 |  
 +--------+  
 | master |  
 +--------+

## 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

+--------+ +----+ +----+ +----+ +----+ +----+ +-------------+  
| c0 | --> | c1 | --> | c2 | --> | c4 | --> | c5 | --> | c6 | <-- | new\_feature |  
+--------+ +----+ +----+ +----+ +----+ +----+ +-------------+  
 |  
 |  
 |  
+--------+ +----+ |  
| master | --> | c3 | <-----+  
+--------+ +----+

##### Depois

+-------------+  
 | new\_feature |  
 +-------------+  
 |  
 |  
 v  
+----+ +----+ +----+ +----+ +----+ +-------------+ +----+ +--------+  
| c0 | --> | c1 | --> | c2 | --> | c4 | --> | c5 | --> | c6 | --> | c7 | <-- | master |  
+----+ +----+ +----+ +----+ +----+ +-------------+ +----+ +--------+  
 | ^  
 | |  
 v |  
 +----+ |  
 | c3 | ------------------------------------------------+  
 +----+

###### 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.

## Patches

## Format patch

Patches são mudanças em um formato padronizado que podem ser transferidas de um lugar para outro:

Para criar um patch do último commit:

git format-patch master

## Apply

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 patches a partir de uma branch

git format-patch master

##### Patch Exemplo

From 6a617a94815f3c0756c903630a8408db747f483e Mon Sep 17 00:00:00 2001  
From: Jean Carlo Machado <contato@jeancarlomachado.com.br>  
Date: Tue, 28 Feb 2017 10:03:09 -0300  
Subject: [PATCH] historia  
  
---  
 docs/historia.md | 10 ++++++++++  
 1 file changed, 10 insertions(+)  
 create mode 100644 docs/historia.md  
  
diff --git a/docs/historia.md b/docs/historia.md  
new file mode 100644  
index 0000000..1634532  
--- /dev/null  
+++ b/docs/historia.md  
@@ -0,0 +1,10 @@  
+Historia do git  
+===============  
+  
+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"  
--  
2.12.0

## Recuperando e desfazendo

## 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

##### Encontrar string no repositório

git grep -i [--untracked ] linus

## 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

O *ammend* leva em consideração a área de *staging*.

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

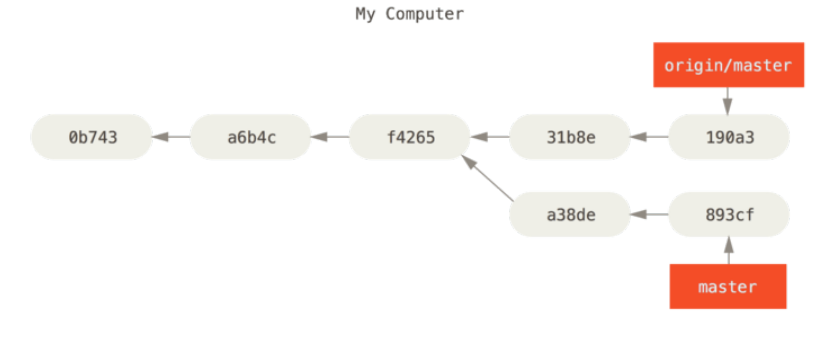
## Inspecionando remoto

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'

## Fluxos de trabalho

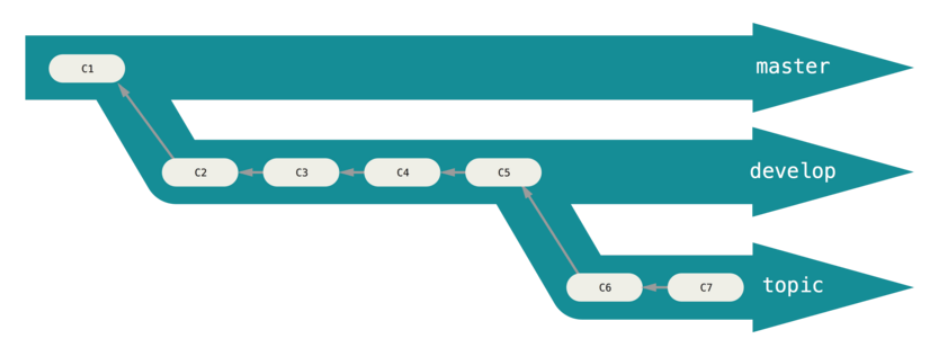
## Modelos de versionamento

* 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

## Github e Gitlab

## Github

GitHub é uma rede social.

##### Permite

* Criar repositórios
* Colaborar para repositórios existentes:
  + Criando issues
  + Enviar código

##### Outros Serviços

###### Githubio Pages

Hospedar um site com seu usuário.

###### Gists

Tipo pastebin

## Gitlab

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

## 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).

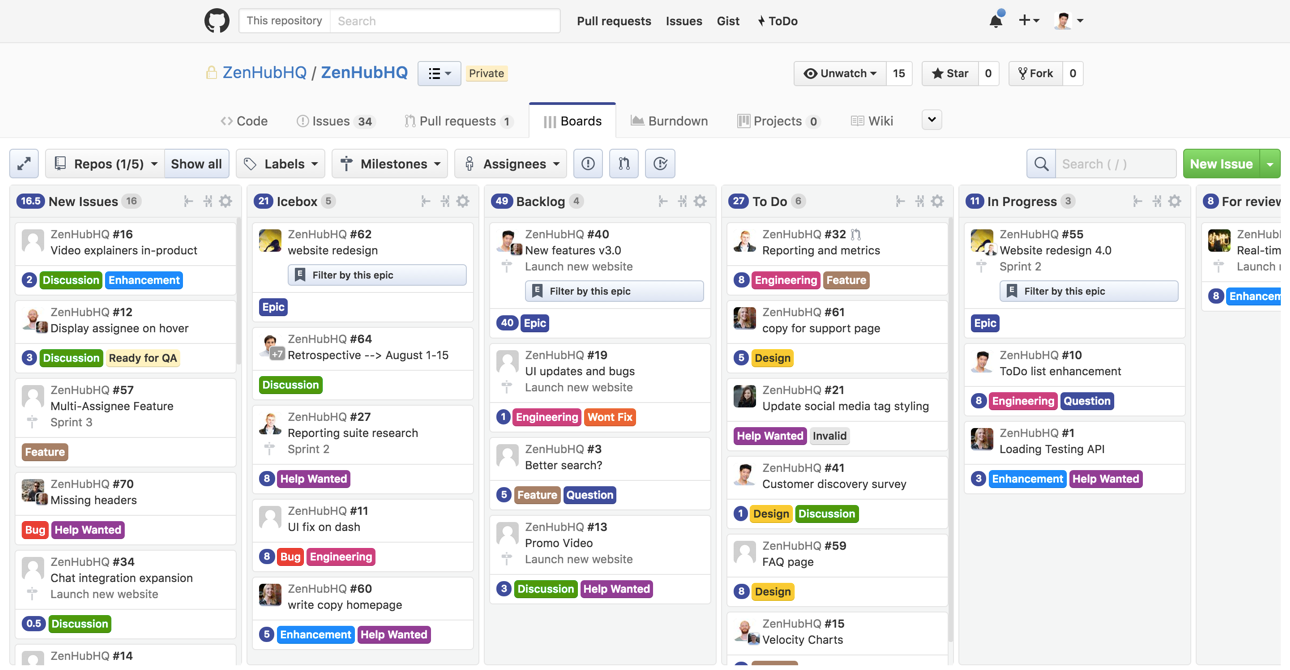
* Waffle
* Buildkite
* ZenHub

##### Codacy

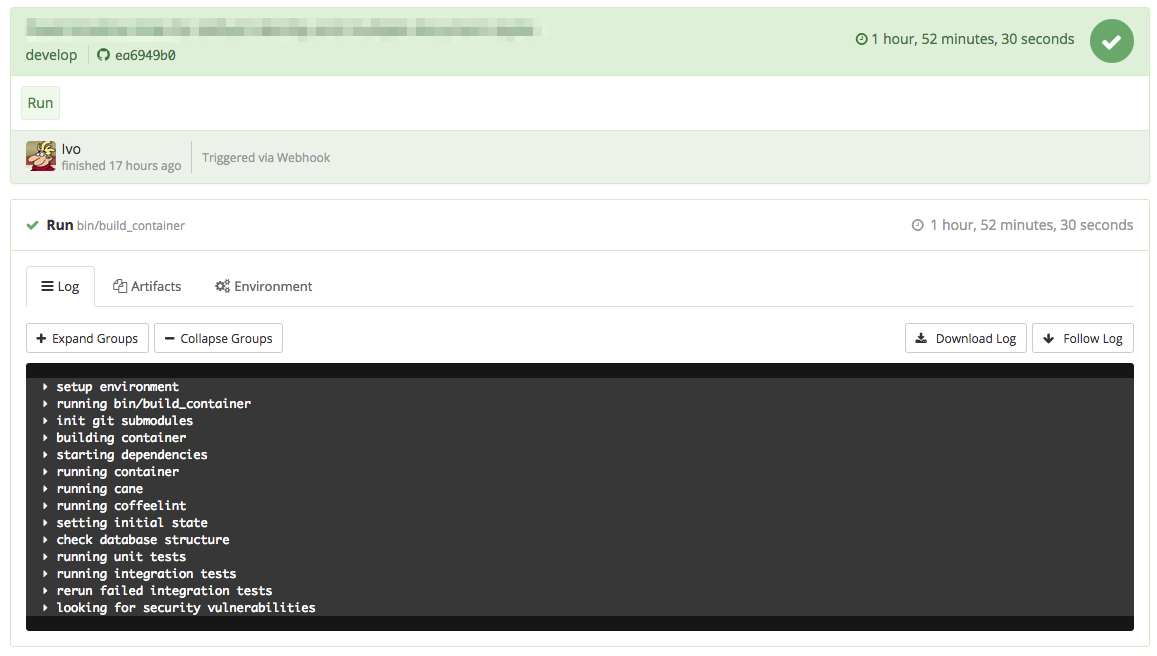
## Serviços

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

## 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/)
* [Novidades do gitlab](https://about.gitlab.com/2017/03/22/gitlab-9-0-released/)
* [Markdown ref 1](https://guides.github.com/features/mastering-markdown/)
* [Markdown ref 2](http://www.markdowntutorial.com)

## 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)