



# Treinamento de Git

Sandro Ribeiro  
18/07/2016

# Agenda

- Definição do Git e suas principais características
- Instalação
- Comandos básicos
- Tagging
- Branching
- Stashing

Git?



# Git?


- Git é um sistema de controle de versão distribuído, gratuito e de código aberto, projetado para lidar com projetos de qualquer tamanho com velocidade e eficiência
- Além de tudo, possui uma baixa curva de aprendizado. Ele supera outras ferramentas de SCM com recursos como ramificação local simples e vários fluxos de trabalho

# Git?




- Foi criado para ser utilizado para versionar o kernel do Linux, após desistirem do BitKeeper (VCS proprietário)
- é uma gíria em inglês britânico para “cabeça dura”







# Principal showcase: GitHub



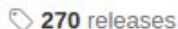

 This repository Search



Pull requests Issues Gist



  2,008  27,545  11,057






Ruby on Rails <http://rubyonrails.org>




 52,969 commits  43 branches  270 releases  2,805 contributors



 rails / + 

Merge pull request #21456 from rodzyn/clean\_requires ...


 rafaelfranca authored 7 hours ago latest commit dc5cf37d73 


 actionmailer	Removed duplicate requiring minitest/mock as it is already required i...	7 days ago
 actionpack	Remove not used requires	8 hours ago
 actionview	Fix calling cache helper with a relation	5 days ago
 activejob	Silence callback deprecation warning if testing AJ	2 days ago
 activemodel	Add missing test for #17351	14 hours ago

  345  531

SSH clone URL

git@github.com:rai 

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#). 

# Principais características do Git



# Desenvolvimento não-linear

- Permite e encoraja criações de ramos (branches) e mesclas (merges) de forma fácil e rápida
- Os branches podem ser totalmente independentes um do outro

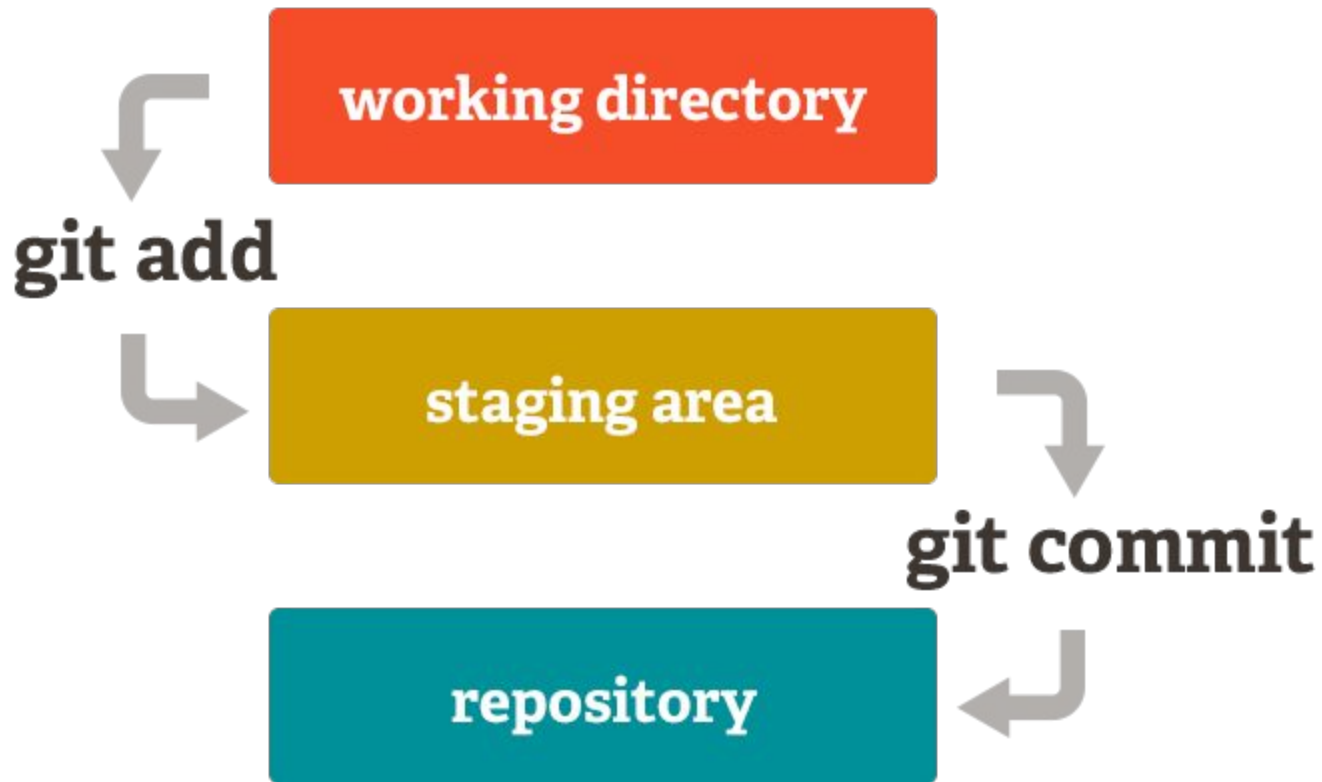
# Desenvolvimento distribuído

- O desenvolvedor possui uma cópia local de todo o histórico de desenvolvimento
- O desenvolvimento pode correr independente de um repositório central
- Podem existir múltiplos repositórios remotos
- Dessa forma, caso alguma repositório se perca ou corrompa, é muito fácil criar um novo repositório principal

# Área de *staging*

- Ao contrário dos outros VCS, o Git possui a "área de teste" ou "índice". Esta é uma área intermediária onde commits podem ser formatados e revisados antes de completar os commits e enviá-los a um repositório remoto
- Outra característica que diferencia o Git de outros VCS é que é possível criar um commit com alguns dos arquivos modificados (e processar o restante em commits separados)

# Área de *staging*



# Pequeno e rápido

- Quase todas as operações são realizadas localmente, dando-lhe uma vantagem de velocidade enorme sobre os VCS que constantemente têm de se comunicar com um servidor remoto
- O Git foi construído para versionar o kernel do Linux, o que significa que teve de lidar com grandes repositórios desde a sua concepção

# Pequeno e rápido

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push 1000 1k images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+, 6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c - 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x

<https://git-scm.com/about/small-and-fast>

# Segurança

- O Git usa um modelo de dados que garante a integridade criptográfica de cada bit de seu projeto. Cada arquivo e commit é *checksummed* e recuperados pelo seu *checksum* quando baixados. É impossível obter qualquer coisa do repositório exceto os mesmos bits que foram submetidos



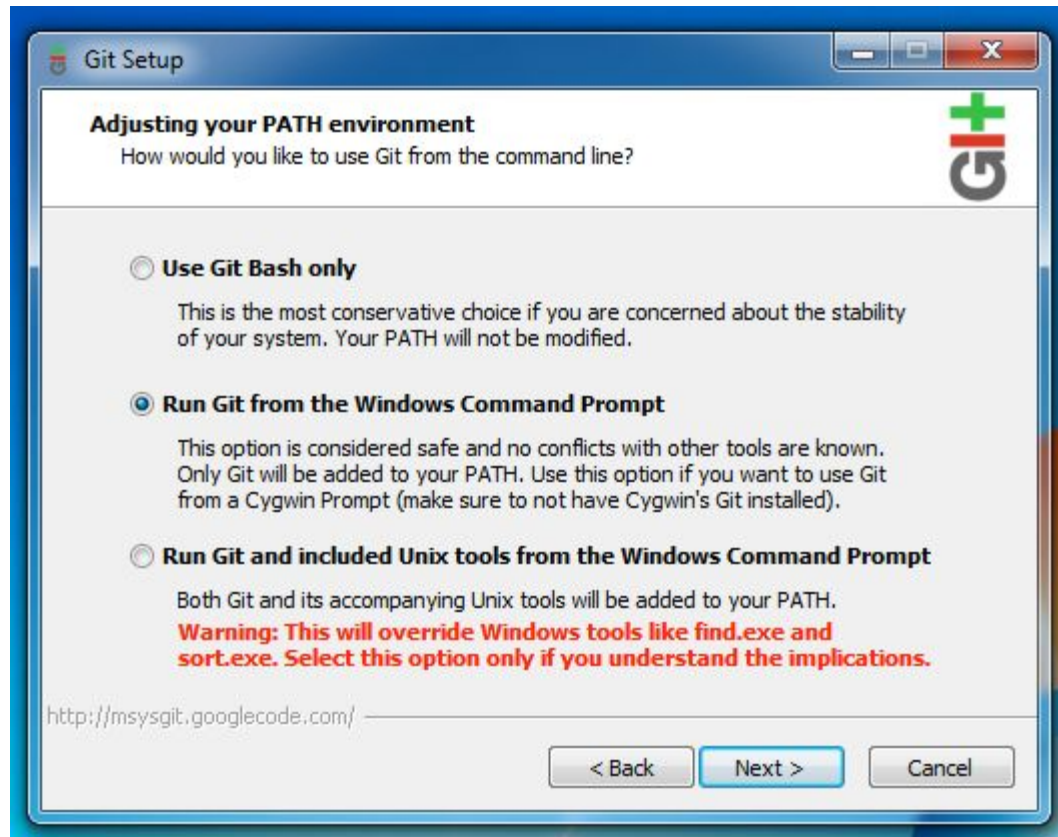
# Segurança

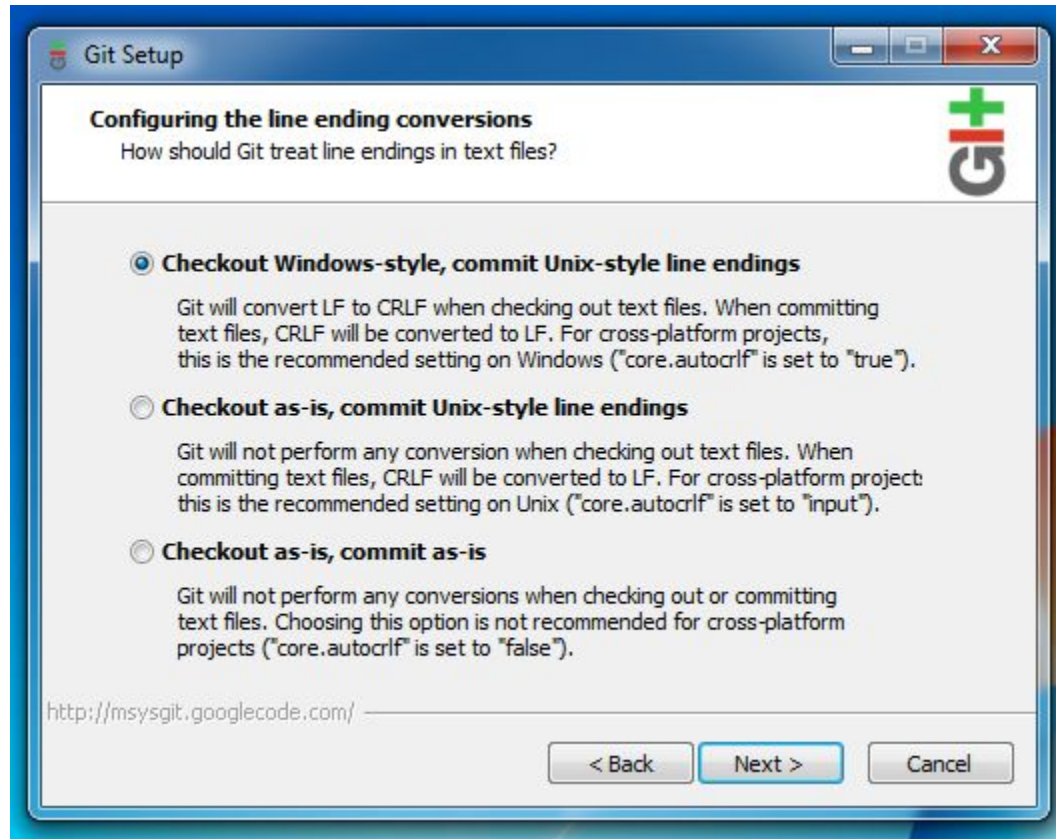


# Instalação

# Instalação

- Linux família Debian
    - `sudo apt-get install git`
  - Linux família RedHat
    - `sudo yum install git`
  - Linux família Arch
    - `sudo pacman -S git`
  - MacOS
    - Default ou através do Homebrew
  - Windows
    - <https://git-scm.com/download/win> ou Chocolatey
-





# Chaves SSH

- Abrir o Terminal (ou Git Bash no Windows) e digitar:  
`ssh-keygen -C "email@empresa.com.br" -t rsa`
- Nos ambientes Unix o par de chaves está em `$HOME/.ssh`. No Windows está em `C:\Users\SeuUsuário\.ssh`

# Comandos básicos



# Configurações básicas

- Defina o seu nome e endereço de email para que os commits sejam identificados:

```
git config --global user.name "Seu Nome"
```

```
git config --global user.email nome@empresa.com.br
```

- Outras possíveis configurações podem ser vistas na documentação oficial

<https://goo.gl/pT5GjE>

# Obter um repositório

Existem duas formas:

- **Inicializando um novo repositório:**

```
$ git init
```

```
$ git add *.rb
```

```
$ git add README
```

```
$ git commit -m 'início do repositório'
```

- **“Clonando” um repositório existente:**

```
$ git clone git@github.com:sandrocvrb/demo.git
```

# Status

A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando **git status**:

```
$ git status  
# On branch master  
nothing to commit, working directory clean
```

# Status

```
$ touch README
```

```
$ git status
```

```
# On branch master
```

```
# Untracked files:
```

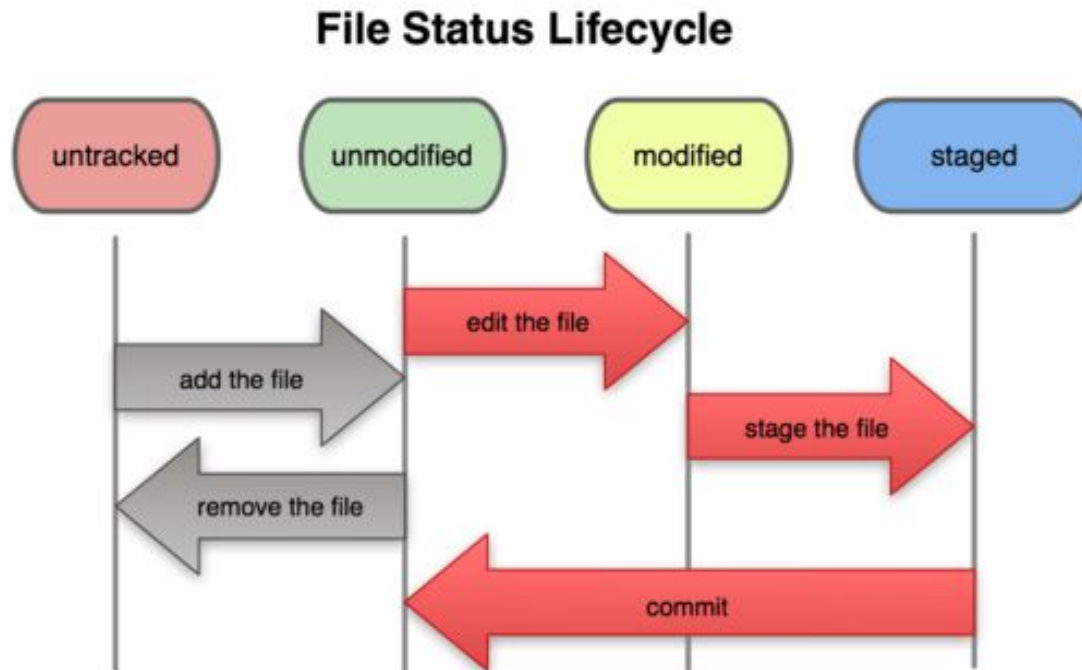
```
# (use "git add <file>..." to include in what will be  
committed)
```

```
#
```

```
#  README
```

```
nothing added to commit but untracked files present (use  
"git add" to track)
```

# Status



# Adicionar arquivo(s): novos

```
$ git add README
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
```

# Adicionar arquivo(s): modificados

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
```



# Adicionar arquivo(s): modificados

```
$ git add benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
```

# Adicionar arquivo(s): modificados

```
$ vim benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
```

---

# Ignorar arquivo(s)

- Os arquivos ignorados de um repositório são especificados dentro do arquivo .gitignore

```
$ cat .gitignore
```

```
.idea/
```

```
.ruby-version
```

```
vendor/bundle/
```

- Repositório de exemplos:

<https://github.com/github/gitignore>

# Ignorar arquivo(s)

```
# um comentário - isto é ignorado
# sem arquivos terminados em .a
*.a
# mas rastreie lib.a, mesmo que você tenha ignorado arquivos
terminados em .a acima
!lib.a
# apenas ignore o arquivo TODO na raiz, não o subdiretório TODO
/TODO
# ignore todos os arquivos no diretório build/
build/
# ignore doc/notes.txt mas, não ignore doc/server/arch.txt
doc/*.txt
```

# Verificar o que foi mudado

```
$ git diff
```

```
diff --git a/benchmarks.rb b/benchmarks.rb
```

```
index 3cb747f..da65585 100644
```

```
--- a/benchmarks.rb
```

```
+++ b/benchmarks.rb
```

```
@@ -36,6 +36,10 @@ def main
```

```
    @commit.parents[0].parents[0].parents[0]  
  end
```

```
+   run_code(x, 'commits 1') do
```

```
+     git.commits.size
```

```
+   end
```

```
+ 
```

```
    run_code(x, 'commits 2') do
```

```
      log = git.commits('master', 15)
```

```
      log.size
```

# Commit de arquivo(s)

```
$ git commit
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

# Commit de arquivo(s)

```
$ git commit -m "Story 182: Ajustes gerais"  
[master]: created 463dc4f: "Ajustes gerais"  
2 files changed, 3 insertions(+), 0 deletions(-)  
create mode 100644 README
```



# Commit de arquivo(s)

```
$ git commit -a -m "Story 182: Ajustes de performance"
```

```
[master]: created 463dc4f: "Ajustes de performance"
```

```
2 files changed, 3 insertions(+), 0 deletions(-)  
create mode 100644 README
```

# Remover arquivo(s)

```
$ rm grit.gemspec
```

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Changes not staged for commit:
```

```
#   (use "git add/rm <file>..." to update what  
will be committed)
```

```
#
```

```
#       deleted:    grit.gemspec
```

# Remover arquivo(s)

```
$ git rm grit.gemspec
```

```
rm 'grit.gemspec'
```

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
#       deleted:    grit.gemspec
```

# Mover arquivo(s)

```
$ 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
```

---

# Mover arquivo(s)

```
$ mv README.txt README
```

```
$ git rm README.txt
```

```
$ git add README
```

# Histórico de commits

```
$ git log
```

```
commit ca82a6dff817ec66f44342007202690a93763949
```

```
Author: Fulano <fulano@email.com>
```

```
Date: Mon Mar 17 21:52:11 2008 -0700
```

faltou coisa na versão inicial!

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
```

```
Author: Sicrano <sicrano@leemail.com>
```

```
Date: Sat Mar 15 10:31:28 2008 -0700
```

primeiro commit

# Histórico de commits

```
$ git log -3
```

(limita a saída para os 3 últimos commits)

```
$ git log -p
```

(mostra o diff de cada commit)

```
$ git log --stat
```

(mostra estatísticas abreviadas para cada commit)

```
$ git log --pretty=oneline
```

(mostra cada commit em uma linha)

```
$ git log --pretty=format:"%h %s" --graph
```

(graph mostra um gráfico de branches e merges)

# Histórico de commits

The screenshot displays the Git GUI interface for a repository named 'testeinit'. The top menu bar includes 'File', 'Edit', 'View', and 'Help'. The main window is divided into several sections:

- Commit Graph:** Shows a graph with three commits. The 'master' branch is highlighted in green, and the 'fix-readme' branch is highlighted in blue. The commit messages are 'Merge branch 'fix-readme'' and 'README estava vazio'. The first commit is labeled 'primeiro commit'.
- Commit List:** A table showing the commit history with columns for the author, email, and date.

Author	Email	Date
Sandro Ribeiro	<sandro.ribeiro@adtsys.com.br>	2015-09-02 00:25:03
Sandro Ribeiro	<sandro.ribeiro@adtsys.com.br>	2015-09-02 00:24:33
Sandro Ribeiro	<sandro.ribeiro@adtsys.com.br>	2015-09-02 00:23:14
- SHA1 ID:** The current commit's SHA1 ID is 'dd1ac1447b2b4c2d370e4f7133a46e4e4c1a67ef'. Navigation buttons (left and right arrows) and a 'Row' indicator (1 / 3) are present.
- Find:** A search bar with a dropdown menu set to 'commit containing:'.
- Search:** A button to initiate the search.
- Diff View:** The 'Diff' view is selected. It shows the commit details for the current commit, including the author, committer, parent commits, and the branch. The commit message is 'Merge branch 'fix-readme''.
- Comments:** A section on the right side of the diff view, currently empty.



# Modificar último commit

```
$ git commit -m 'adicionando funcionalidade'
```

```
$ git add arquivo.java
```

```
$ git commit --amend
```

# Tirar arquivo(s) de staging

```
$ git add .  
$ git status  
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
#       modified:   README.txt  
#       modified:   benchmarks.rb
```

# Tirar arquivo(s) de staging

```
$ git reset HEAD benchmarks.rb
benchmarks.rb: locally modified
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   benchmarks.rb
```

---

# Desfazer modificação em arquivo

```
$ git status
# Changes not staged for commit:
#   (use "git add <file>..." to update what will
#   be committed)
#   (use "git checkout -- <file>..." to discard
#   changes in working directory)
#
#       modified:   benchmarks.rb
#       modified:   README.txt
```

# Desfazer modificação em arquivo

```
$ git checkout benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
```

# Repo remoto: listar

```
$ git remote  
origin
```

```
$ git remote -v  
origin    git://github.com/yyz/qqq.git (fetch)  
origin    git://github.com/yyz/qqq.git (push)
```

# Repo remoto: adicionar

```
$ git remote add pb git://github.com/xxx/qqq.git
```

```
$ git remote -v
```

```
origin    git://github.com/yyz/qqq.git
```

```
pb        git://github.com/xxx/qqq.git
```

# Repo remoto: baixar

```
$ git fetch pb
```

```
remote: Counting objects: 58, done.
```

```
remote: Compressing objects: 100% (41/41), done.
```

```
remote: Total 44 (delta 24), reused 1 (delta 0)
```

```
Unpacking objects: 100% (44/44), done.
```

```
From git://github.com/xxx/qqq
```

```
* [new branch]      master      -> pb/master
```

```
* [new branch]      ticgit      -> pb/ticgit
```



# Repo remoto: baixar

- **git fetch** traz os dados para o seu repositório local  
– ele não faz o merge automaticamente com o seus dados ou modifica o que você está trabalhando atualmente. Você terá que fazer o merge manualmente no seu trabalho quando estiver pronto
- **git pull** = git fetch + git merge

# Repo remoto: enviar

```
$ git push origin master
```

# Repo remoto: renomear e remover

```
$ git remote rename pb zzz
```

```
$ git remote
```

```
origin
```

```
zzz
```

```
$ git remote rm zzz
```

```
$ git remote
```

```
origin
```

# Aliases

```
$ git config --global alias.co checkout  
$ git config --global alias.br branch  
$ git config --global alias.ci commit  
$ git config --global alias.st status  
$ git config --global alias.unstage 'reset HEAD  
-- '  
$ git config --global alias.uncommit 'reset  
--soft HEAD~1'
```

# Tagging

# Tag: o que é

- Assim como a maioria dos VCS's, Git tem a habilidade de criar tags em pontos específicos na história do repositório para indicar pontos importantes
- Geralmente esta funcionalidade é usada para marcar pontos de release (v1.0, v1.1, etc)

# Tag: listar

```
$ git tag
```

```
v0.9
```

```
v1.0
```

```
v1.1
```

```
$ git tag -l 'v1.0.*'
```

```
v1.0.0
```

```
v1.0.1
```

# Tag: criar

```
$ git tag -a v1.2 -m 'versão 1.2'
```

```
$ git tag
```

```
v0.9
```

```
v1.0
```

```
v1.1
```

```
v1.2
```



# Tag: ver informações

```
$ git show v1.2
```

```
tag v1.2
```

```
Tagger: Fulano <fulano@email.com>
```

```
Date: Mon Feb 19 08:47:13 2011 -0800
```

```
versão 1.2
```

```
commit 61422490d3f51f4b62868acfcfe52a01a8c41b6e
```

```
Merge: 9a447f7... f6b4c97...
```

```
Author: Fulano <fulano@email.com>
```

```
Date: Sun Feb 18 19:02:46 2011 -0800
```

```
Merge branch 'prova-de-conceito'
```

---

# Tag: criar para commits passados

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch
'prova-de-conceito'
9fceb02d0ae598e95dc970b74767f19372d61af8 atualizou xmls
8a5cbc430f1a9c3d00faaeffd07798508422908a atualizou README
```

```
$ git tag -a v1.1.1 9fceb02
```

```
$ git tag
```

```
v1.0
```

```
v1.1
```

```
v1.1.1
```

```
v1.2
```

# Tag: enviar para repo remoto

```
$ git push origin v1.3
```

```
$ git push origin --tags
```

# Tag: remover

```
$ git tag -d v1.0
```

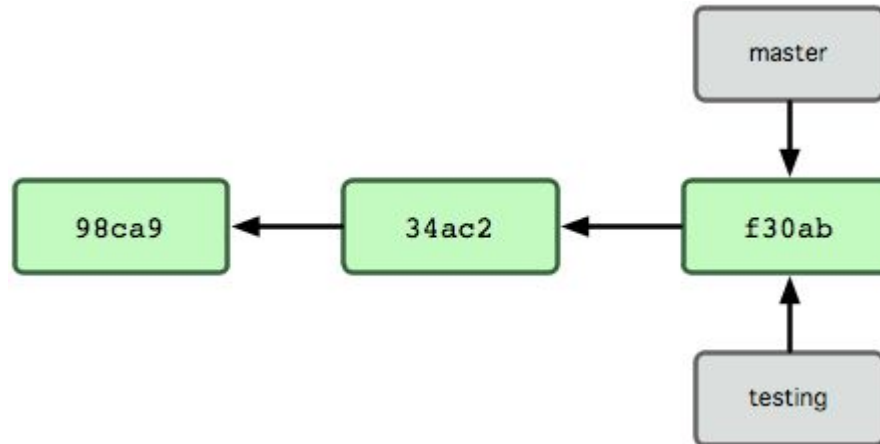
# Branching

# Branch: o que é

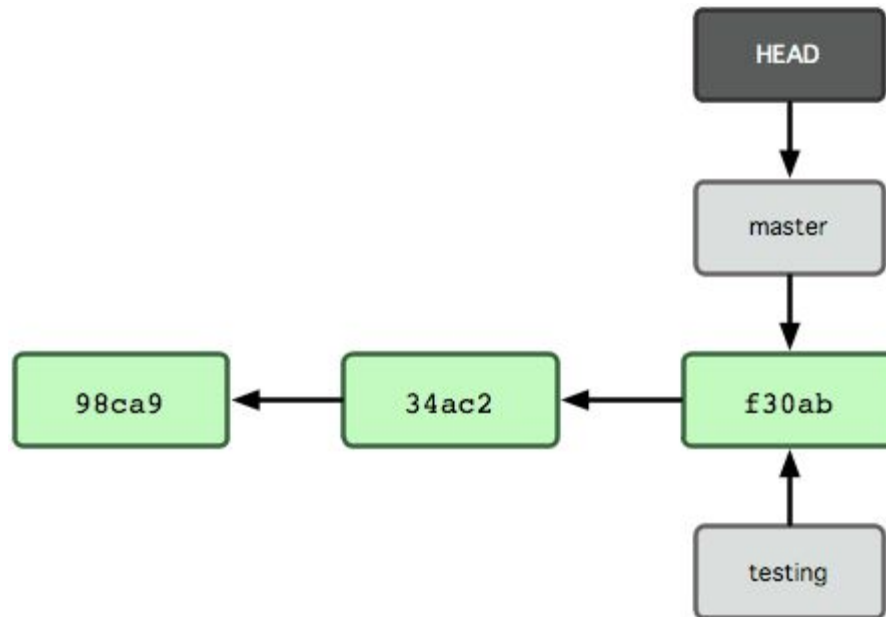
- Criar um branch significa dizer que o usuário irá divergir da linha principal de desenvolvimento e continuar a trabalhar em algo sem afetar o branch original.
- Em outros VCS este é um processo um pouco caro, muitas vezes exigindo que você crie uma nova cópia do seu diretório de código-fonte, o que pode levar um longo tempo para grandes projetos

# Branch: básico

```
$ git branch testing
```



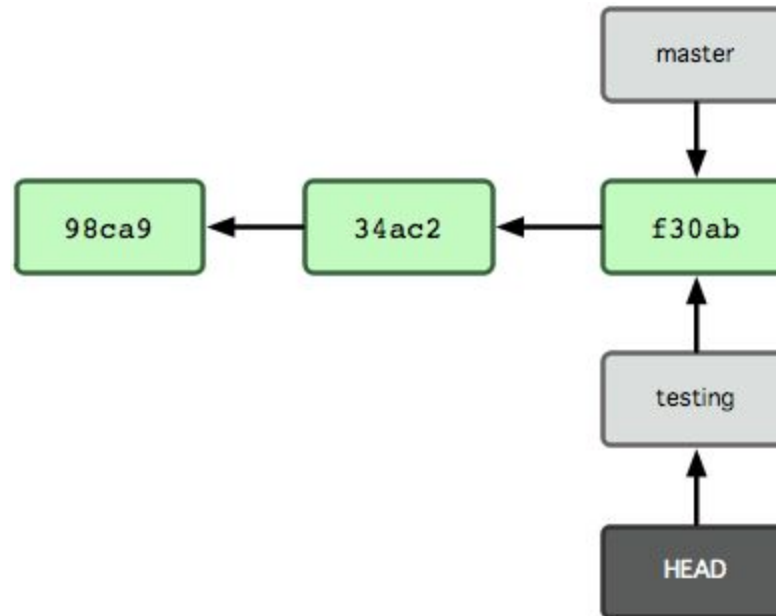
# Branch: básico





# Branch: básico

```
$ git checkout testing
```



## Branch: criar

```
$ git checkout -b req-234
```

```
Switched to a new branch "req-234"
```

```
$ git branch req-234
```

```
$ git checkout req-234
```

## Branch: alterar arquivo(s)

```
$ vim index.html
```

```
$ git commit -a -m 'req-555 adicionei um novo  
rodapé'
```

# Branch: alterar arquivo(s)

The screenshot displays the Git GUI application window titled "gitk: testeinit". The interface includes a menu bar (File, Edit, View, Help) and a commit history view. The commit history shows a merge of the 'fix-readme' branch into 'master'. The commit message is "Merge branch 'fix-readme'" and the description is "README estava vazio". The commit is attributed to Sandro Ribeiro. The SHA1 ID is dd1ac1447b2b4c2d370e4f7133a46e4e4c1a67ef. The commit is the first of three rows in the list.

SHA1 ID: dd1ac1447b2b4c2d370e4f7133a46e4e4c1a67ef

Find commit containing:

Search

Diff Old version New version Lines of context: 3 Ignore space change

Author: Sandro Ribeiro <sandro.ribeiro@adtsys.com.br> 2015-09-02 00:25:03  
Committer: Sandro Ribeiro <sandro.ribeiro@adtsys.com.br> 2015-09-02 00:25:03  
Parent: 7d89d563facc01da10030deba17dcdb2e6e2f5ea (primeiro commit)  
Parent: 7181bb71cblac4c014a02525aa75feeb5bd6fd8 (README estava vazio)  
Branch: master  
Follows:  
Precedes:

Merge branch 'fix-readme'

Comments

# Branch: merge

```
$ git checkout master
```

```
$ git merge hotfix
```

```
Updating f42c576..3a0874c
```

```
Fast forward
```

```
README |      1 -
```

```
1 files changed, 0 insertions(+), 1 deletions(-)
```

# Branch: conflitos de merge

```
$ git merge req-222
```

```
Auto-merging index.html
```

```
CONFLICT (content): Merge conflict in index.html
```

```
Automatic merge failed; fix conflicts and then  
commit the result
```

# Branch: conflitos de merge

```
$ git status
index.html: needs merge
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will
#   be committed)
#   (use "git checkout -- <file>..." to discard
#   changes in working directory)
#
#       unmerged:   index.html
```

---

# Branch: conflitos de merge

```
<<<<<< HEAD:index.html
```

```
<div id="footer">contato :
```

```
email.support@github.com</div>
```

```
=====
```

```
<div id="footer">
```

```
    por favor nos contate em support@github.com
```

```
</div>
```

```
>>>>>> iss53:index.html
```



# Branch: gerenciar

```
$ git branch
```

```
* master  
testing
```

```
$ git branch -v
```

```
* master 7a98805 Merge branch 'req-222'  
testing 782fd34 adicionar método GetCliente
```

# Branch: gerenciar

```
$ git branch --merged  
  req-123  
* master
```

```
$ git branch --no-merged  
  testing
```

# Branch: gerenciar

```
$ git branch -d testing
```

error: The branch 'testing' is not an ancestor of your current HEAD.

If you are sure you want to delete it, run ``git branch -D testing``.

# Stashing

# Stash: o que é

Muitas vezes, quando você está trabalhando em uma parte do seu projeto, você precisa mudar de branch por um tempo para trabalhar em outra coisa. O problema é, você não quer fazer o commit de um trabalho incompleto somente para voltar a ele mais tarde. A resposta para esse problema é o comando **git stash**

# Stash: criar

```
$ git status
```

```
# On branch master
```

```
# Changes to be committed:
```

```
# (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
#    modified:   index.html
```

```
#
```

```
# Changes not staged for commit:
```

```
# (use "git add <file>..." to update what will be committed)
```

```
#
```

```
#    modified:   lib/get_cliente.java
```

# Stash: criar

```
$ git stash
```

```
Saved working directory and index state \  
    "WIP on master: 049d078 novo index.html"  
HEAD is now at 049d078 added the index file  
(To restore them type "git stash apply")
```

```
$ git status
```

```
# On branch master
```

```
nothing to commit, working directory clean
```

# Stash: listar

```
$ git stash list
```

```
stash@{0}: WIP on master: 049d078 novo index.html
```

```
stash@{1}: WIP on master: c264051... Reverter  
mudança GetCPF
```

```
stash@{2}: WIP on master: 21d80a5... novo padrão
```



# Stash: aplicar

```
$ git stash apply
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will
#   be committed)
#
#       modified:   index.html
#       modified:   lib/get_dados.java
```

Pode ser utilizada a opção **pop** ao invés de **apply**

---

# Stash: criar um branch

```
$ git stash branch teste-novo-getcpf
```

# Obrigado!

sandro.ribeiro@adtsys.com.br