

Controle de Versões com



Vagner Santana
<http://vagnersantana.com>

Agenda

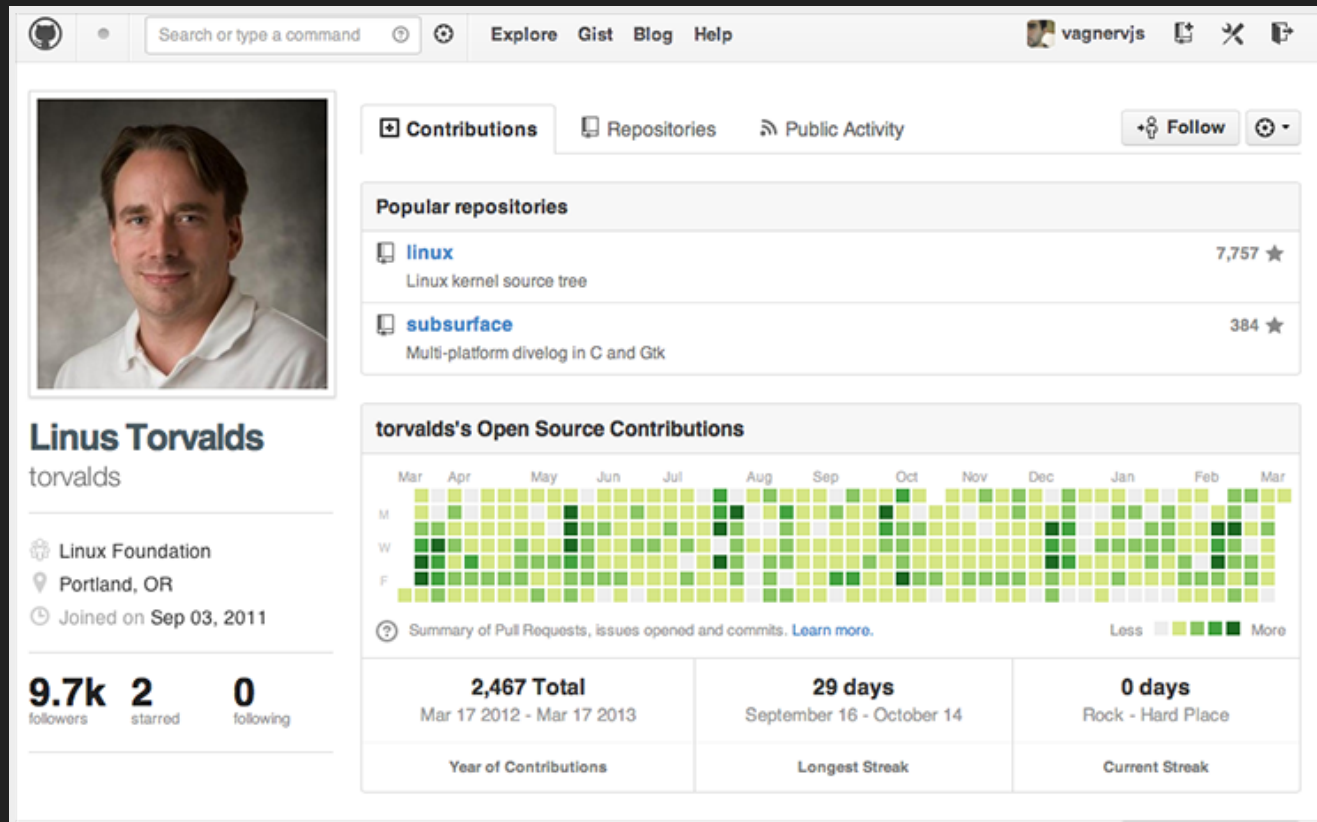
- Introdução
- Instalação
- Configuração
- Iniciando um projeto
- Áreas do git
- Commit
- Gerenciamento de índice
- Stash / Branch
- Merge / Rebase
- Repositórios remotos: Modelo Centralizado / Modelo Distribuído
- Extras

Dica:

Não se preocupe,
o início sempre é difícil!

Introdução

História: Criado por **Linus Torvalds** para gerenciar o desenvolvimento do kernel do linux (antes era utilizado o bitKeeper).



Introdução

- Git não tem nada a ver com o **Subversion**.
- Git é um filesystem distribuído. Ou seja não só códigos fontes, e sim qualquer tipo de arquivos.
- Muito eficiente e confiável.
- Utiliza **SHA1** para identificação dos commits.
- Dificilmente ocorre perda de arquivos no git.

Instalação

Linux

- Instalando git e ssh

```
sudo apt-get install git-core git-svn ssh
```

- Gerando chaves ssh:

```
ssh keygen -t rsa
```

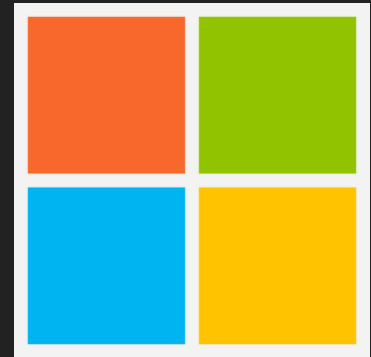


Instalação

Windows

- Download: <http://git-scm.com/download/win>
- Abrir git bash e gerar chaves ssh:

```
ssh-keygen -t rsa
```



Instalação

Mac

- Download: <http://git-scm.com/download/mac>
- Ou, usando **homebrew**

```
brew install git
```

- Gerando chaves ssh:

```
ssh keygen -t rsa
```



Configuração

- Definindo nome e email para identificar autor

```
git config --global user.name "vagnervjs"  
git config --global user.email "vagnervjs@gmail.com"
```

- Verificando

```
cat /Users/vagner/.gitconfig
```

- Highlight

```
//arquivo .gitconfig  
color: auto
```

Dica:

Sempre leia as mensagens!

Iniciando um projeto

- Iniciar git em um projeto

```
cd pasta_do_projeto  
git init
```

- Ajuda

```
git help init
```

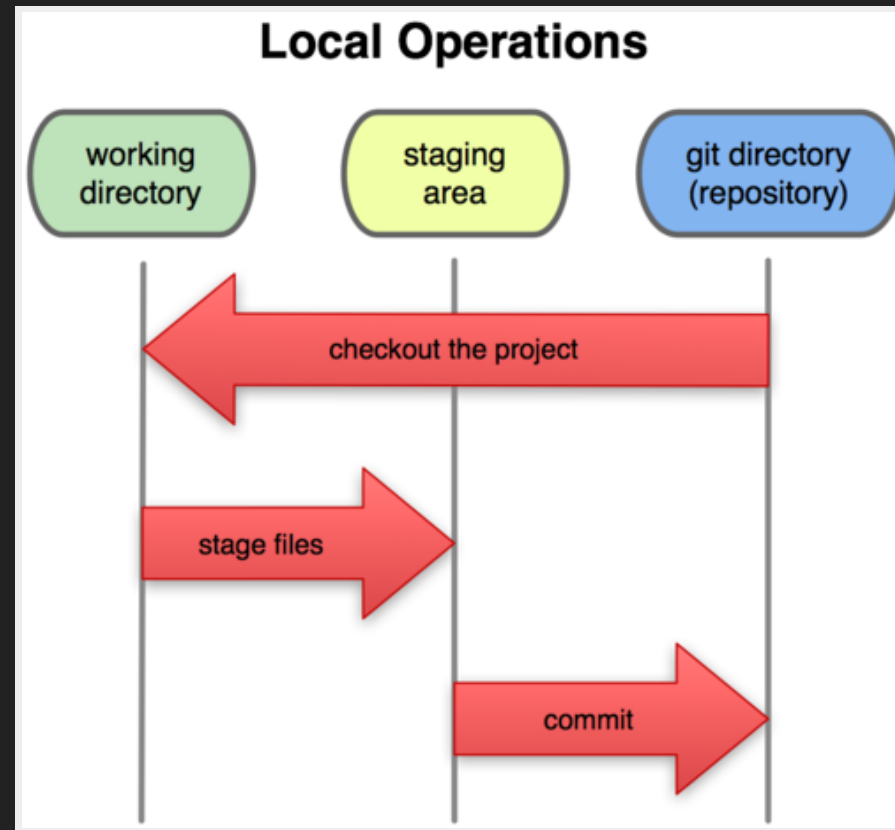
- Status

```
git status
```

Áreas do Git

- **Working directory:** diretório atual de trabalho
- **Stage area:** índice
- **.git:** repositório de dados

Áreas do Git



<http://git-scm.com/book/en/Getting-Started-Git-Basics>

Continuando...

- Adicionando arquivos ao índice

```
git add . file.txt //apenas arquivo file.txt  
git add . "*.txt" //todos arquivos .txt  
git add . //Todos arquivos do working directory
```

- Após adicionar os arquivos ficam como new file, ou seja, a serem adicionados no próximo commit.
- Retirar do índice

```
git rm --cached file.txt
```

- Commit

```
git commit -m "Primeiro commit"
```

- Log

```
git log
```

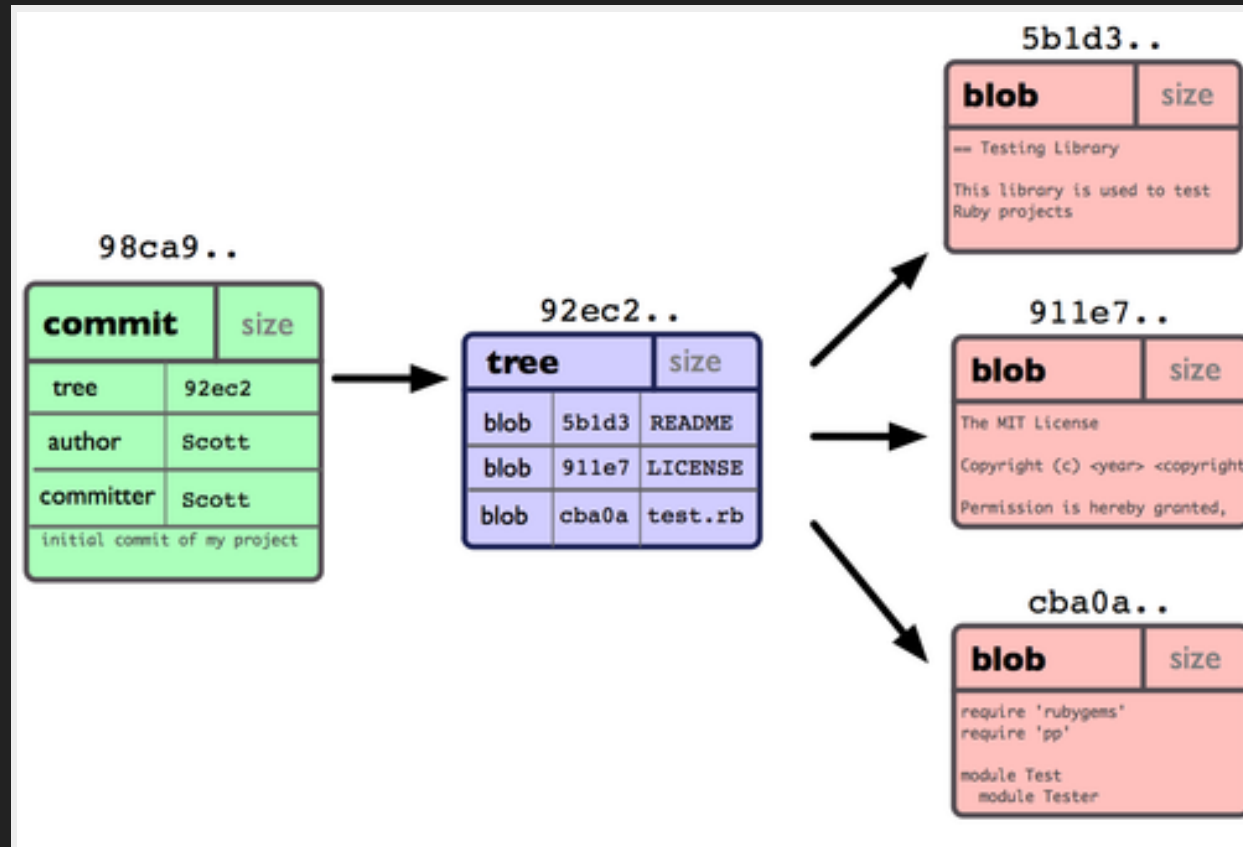
Dica:

Sempre crie commits com
mensagens descrevendo a alteração
feita no projeto!

Estrutura do commit

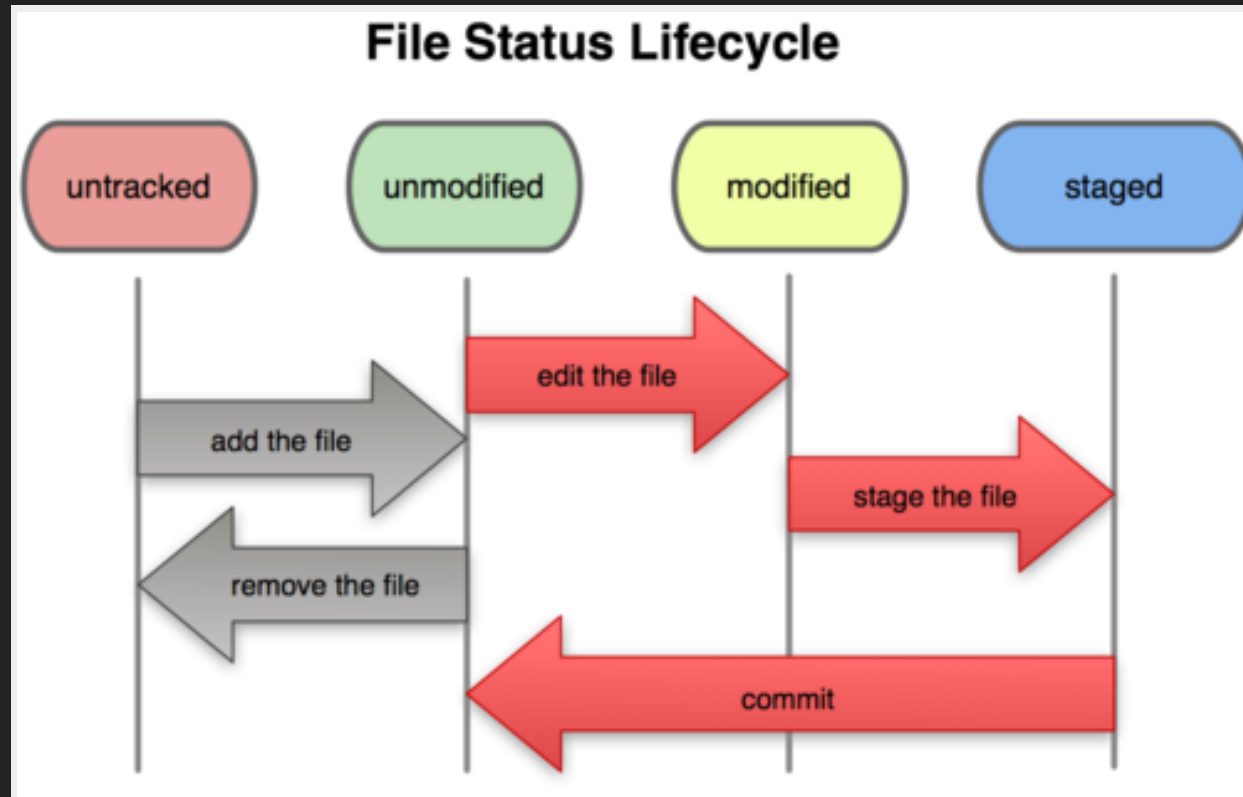
- Commit é um pacote, um envelope.
- Após adicionar os arquivos ficam como new file, ou seja, a serem adicionados no próximo commit.
- Branch e tags apontam sempre para commits
- Commits possuem um identificador em SHA1, o que torna praticamente impossível conflito de commits.
- Mesmo utilizando trechos do SHA1 (até 5 caracteres no máximo), ainda assim é possível identificar o commit.
- O git nunca duplica conteúdos entre commits, apenas faz referência ao blob do commit anterior e adiciona os novos arquivos.

Estrutura do commit



<http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is>

Gerenciando o índice



<http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository>

Gerenciando o índice

- Imagine as ações abaixo:

```
Moficação no arquivo file.txt  
git add file.txt
```

- Como desfazer?

```
git reset HEAD file.txt
```

- E para voltar ao início (antes de modificar o arquivo file.txt)

```
git checkout -- file.txt
```

- Para todos os arquivos

```
git reset HEAD  
//todos os arquivos voltam para status untracked
```

Gerenciando o índice

- Jogar fora último commit

```
git reset HEAD~1 --hard
//agora a cabeça aponta para o commit anterior
```

- Esta operação parece destrutiva mas não é:

```
git reflog
//guarda todas as coisas que foram jogadas fora
git merge < sha1 do commit desejado >
```

- Voltar arquivos de commit para o índice

```
git reset HEAD~1
```

Gerenciando o índice

- Remover arquivos do índice

```
git rm --cached < file >
```

- Limpar arquivos do working directory

```
git clean -f
```

- Mais operações de índice...

```
git add -i
```

Stash

Situação: você está desenvolvendo uma nova funcionalidade e chega um pedido para resolver um bug.

Não é viável fazer um commit neste momento pois a funcionalidade não está pronta.

Para auxiliar situações como essa, existe uma quarta área (temporária) no git, chamada stash.

Stash

- Criando um stash anônimo

```
git stash
```

- Listando

```
git stash list
```

- Para voltar ao desenvolvimento após corrigir o bug descrito na situação acima

```
git stash apply
```

- Removendo stash

```
git stash clear
```

Stash

- Criando stash e definindo uma identificação

```
git stash save "fazendo algo"
```

- Outros comandos para stash

```
git stash apply stash@[0]  
//volta para o stash desejado  
git stash pop  
//tira da lista, aplica e apaga o stash  
git stash drop stash{0}  
//retira da lista de stash
```


Branch

- Objetivo de trabalhar com branches: separar funcionalidades durante o desenvolvimento.
- Troca de contexto sem atrito
- Criando branch chamando "desenvolvimento"

```
git checkout -b "desenvolvimento"
```

- Trocando de master para "desenvolvimento"

```
git checkout desenvolvimento
```

- Listando

```
git branch
```

Branch

- Visualizando histórico

```
git log --graph
```

- Unindo alterações feitas em um branch (ex: "desenvolvimento") para o master

```
git checkout master  
git merge desenvolvimento
```

- Removendo branch

```
git branch -d desenvolvimento
```

- GUI para visualização

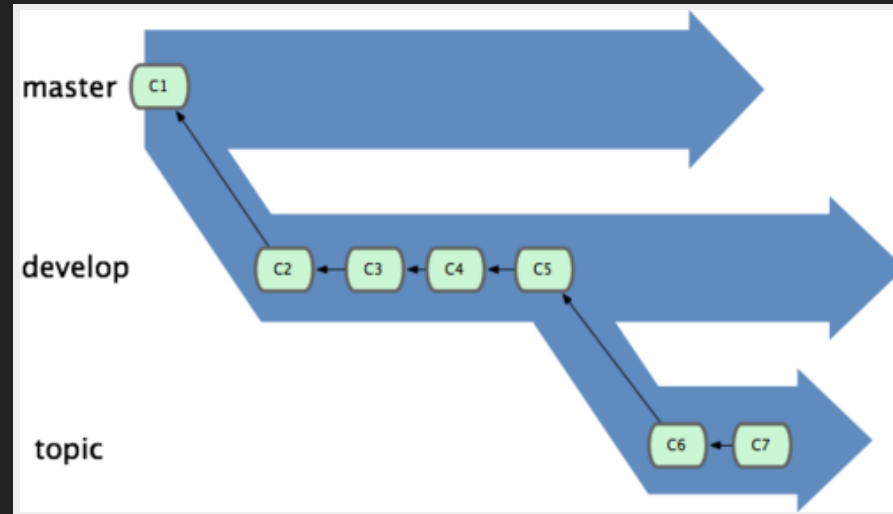
```
gitk --all &
```

- Mesclando pequenos commits em um único

```
git merge < branch > --squash
```

Dica:

Evite trabalhar no master. O ideal é que cada branch tenha uma funcionalidade.



<http://git-scm.com/book/en/Git-Branching-Branching-Workflows>

Merge / Rebase

- Situação: estou trabalhando em um branch e alguém alterou o master
- O que deve ser feito é: manter sempre o branch atual atualizado pelo master.

```
git rebase master
```

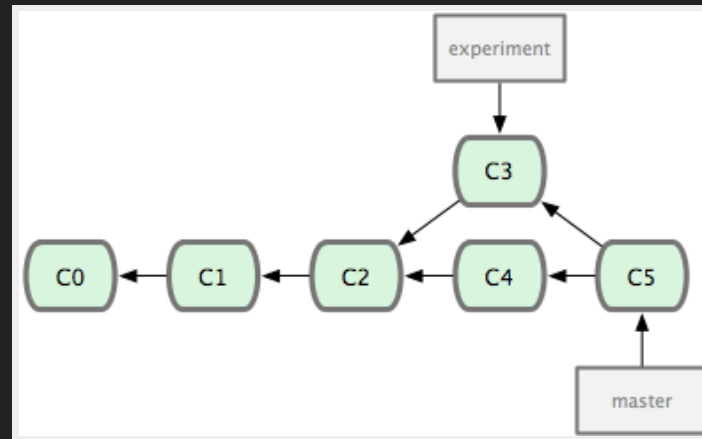
- Se existir um conflito, ou seja, arquivo local é diferente do arquivo que está no master, o arquivo deve ser analisado e depois de corrigir o conflito:

```
git rebase --continue
```

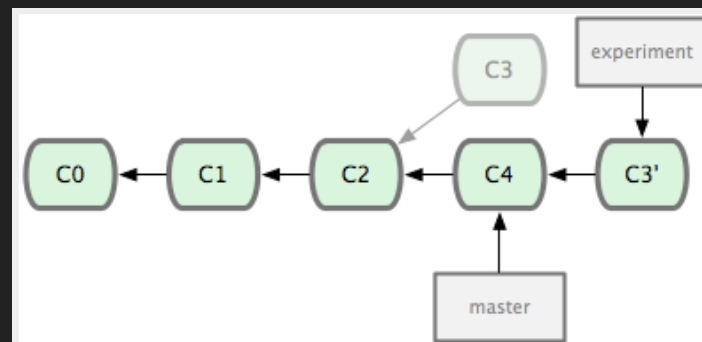
Merge / Rebase

- Se estiver em trabalhando com um servidor remoto, deve ser feito merge, pois o rebase é destrutivo.
- Com rebase a linha de histórico é reta, com merge é criado um desvio.
- Merge traz os commits e adiciona mais um no final.
- Rebase junta todos os commits linearmente.

Merge



Rebase



<http://git-scm.com/book/en/Git-Branching-Rebasing>

Respositórios remotos

- Clonando um repositório

```
git clone teste/.git teste2
```

- Mostrando origem

```
git remote  
git remote show origin
```

- Criando branch local a partir de branch remoto

```
git checkout -b desenvolvimento origin/desenvolvi  
mento
```

- Exibindo braches locais e remotos

```
git branch -a
```

- Enviando para repositório remoto

```
git push origin master
```

Modelo Centralizado

- Cenário: Dois desenvolvedores, cada um com uma cópia local e um servidor remoto.
- Criando repositório no servidor

```
mkdir teste.git  
cd teste.git  
git init --bare
```

- Adicionando permissão de acesso

```
cd ~/.ssh  
echo "chave rsa do usuario" >> authorized_keys
```

- Chave ssh no usuário

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


Modelo Centralizado

- No usuário, adicionando remote ao repositório local

```
git remote add origin vagner@172.16.100.198:/home  
/git/repositorios/teste.git
```

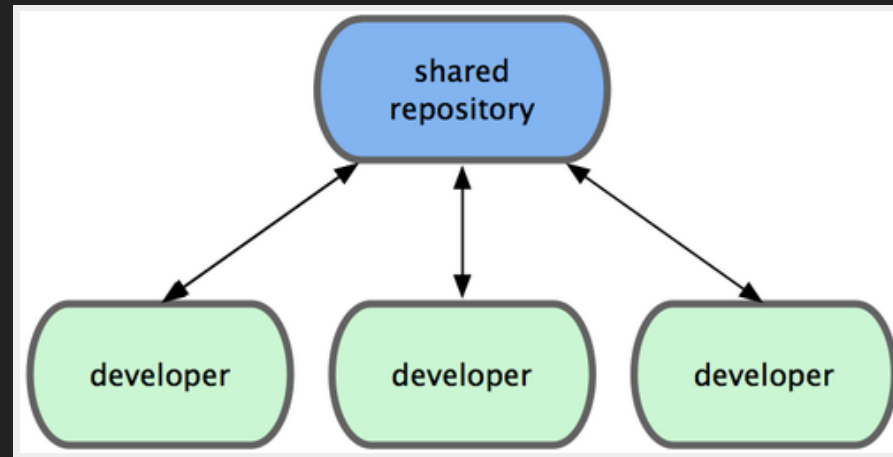
- Trazendo alterações do repositório remoto para um branch especial

```
git fetch origin master
```

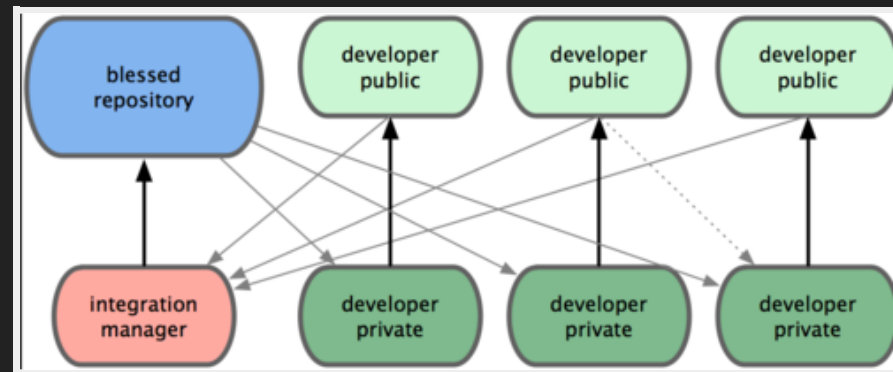
- Fech seguido de merge

```
git pull
```

Modelo Centralizado



Modelo Distribuído



<http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows>

Gitweb

Interface gráfica para visualização dos repositórios no servidor remoto.

The screenshot shows the Gitweb interface for the repository `/pub/scm / git/git.git / summary`. At the top, there are navigation links: `summary`, `shortlog`, `log`, `commit`, `commitdiff`, and `tree`. A search bar is also present with a "commit" button and a "search:" input field. Below the navigation, the repository details are listed:

- description: The core git plumbing
- owner: Junio C Hamano
- last change: Fri, 20 Feb 2009 07:44:07 +0000
- URL: [git://git.kernel.org/pub/scm/git/git.git](http://git.kernel.org/pub/scm/git/git.git)
<http://www.kernel.org/pub/scm/git/git.git>

Below the details, the "shortlog" section displays a list of recent commits. Each entry includes the time since the commit, the author's name, the commit message, and links to the commit, commitdiff, tree, and snapshot.

Time	Author	Commit Message	commit	commitdiff	tree	snapshot
33 hours ago	Junio C Hamano	Merge branch 'maint' <code>master</code>	commit	commitdiff	tree	snapshot
34 hours ago	Matthieu Moy	More friendly message when locking the index fails. <code>maint</code>	commit	commitdiff	tree	snapshot
34 hours ago	Matthieu Moy	Document git blame --reverse.	commit	commitdiff	tree	snapshot
34 hours ago	Marcel M. Cary	gitweb: Hyperlink multiple git hashes on the same commi ...	commit	commitdiff	tree	snapshot
34 hours ago	Johannes Schindelin	system_path(): simplify using strip_path_suffix(), ...	commit	commitdiff	tree	snapshot
34 hours ago	Johannes Schindelin	Introduce the function strip_path_suffix()	commit	commitdiff	tree	snapshot
2 days ago	Todd Zullinger	Documentation: Note file formats send-email accepts	commit	commitdiff	tree	snapshot
2 days ago	Junio C Hamano	Merge branch 'maint'	commit	commitdiff	tree	snapshot
2 days ago	Junio C Hamano	tests: fix "export var=val"	commit	commitdiff	tree	snapshot
2 days ago	Lars Noschinski	filter-branch -d: Export GIT_DIR earlier	commit	commitdiff	tree	snapshot
2 days ago	Jay Soffian	disallow providing multiple upstream branches to rebase ...	commit	commitdiff	tree	snapshot
2 days ago	Michael Spang	Skip timestamp differences for diff --no-index	commit	commitdiff	tree	snapshot
2 days ago	Junio C Hamano	git-svn: fix parsing of timestamp obtained from svn	commit	commitdiff	tree	snapshot
2 days ago	Marcel M. Cary	gitweb: Fix warnings with override permitted but no ...	commit	commitdiff	tree	snapshot
2 days ago	Gerrit Pape	Documentation/git-push: --all, --mirror, --tags can ...	commit	commitdiff	tree	snapshot
2 days ago	Thomas Rast	bash completion: only show 'log --merge' if merging	commit	commitdiff	tree	snapshot

<http://git-scm.com/book/ch4-6.html>

Extras

Tags

```
git tag v1.0
git push origin v1.0
git push --tags
git -b < branch > v0.8
```

Logs

```
git log --stash
git log --pretty=oneline
git log --pretty=format:"[%an %ad] %h - %s"
git log --pretty=format:"%h - %s" --graph
git log --since=30.minutes
git log --since=4.hours --until=2.hours
git log --since=4.hours --until=2.hours --before=
"2013-03-15"
```

Considerações Finais

- Controle de versões é essencial para organização do projeto
- É fácil, basta usar frequentemente
- Facilita contribuição em projetos open source
- **Github** é a melhor forma de mostrar seu trabalho como desenvolvedor, seja para empresas ou pessoas

Referências

- **Git SCM - Pro Git Book**
- **Fabio Akita - Screencast - Começando com Git**
- **Git for Computer Scientists**
- **Git Documentation**

Perguntas?



Obrigado!



Github



Facebook



Twitter



Google +



Linkedin



Slideshare

