

GIT

Gerenciamento de projeto
e versionamento Semântico.

1. Git:

1.1 O que é

1.2 Objetivos

1.3 Porque usar controle de versão

1.4 vantagens

1.5 desvantagens

2. Passos iniciais:

2.1 Instalação do GIT

2.2 Configurando git

2.3 Iniciando um repositório(github)

2.4 Commit Inicial

2.5 Adicionando e enviando arquivos

2.6 Utilizando .gitignore

3. Gerenciando seus commits:

3.1 Introduzindo os conceitos de:

- status, clone, fetch, remote, push e pull, update, diff. reset

3.2 Refazendo commits

3.3 Git Log

4. Gerenciamento de branches (git flow):

4.1 Git flow

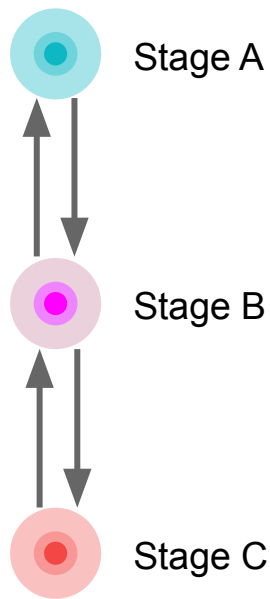
4.2 Gerenciamento de Tags

4.3 Versionamento Semântico

1. O que é git

Porque usar controle de versão?

- ▶ Você poderá ir e vir entre diferentes estágios do arquivo.
- ▶ Vai lhe dar menos dor de cabeça para gerenciar arquivos.
- ▶ Geralmente você precisa de um método backup consistente, quando a equipe é + de 1.



Ao usar um sistema de controle de versão, você tem o poder de lidar com o fluxo de mudanças acontecendo com seus documentos.

”

2.

Passos iniciais

Por onde começar!?

Configurando seu git

```
git config --global user.name "your full name"
```

```
git config --global user.email "your email"
```

```
git config -l
```

*O ultimo comando listará todas as configurações.

Configurando seu git

```
git config --local user.name "your full name"
```

```
git config --local user.email "your email"
```

Global indica um valor global para todos os repositórios criados no sistema por esse usuário do sistema, enquanto a configuração local é exatamente o oposto.

Configurando seu git

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

Gerar chaves de criptografia.

3.

Gerenciando seus commits

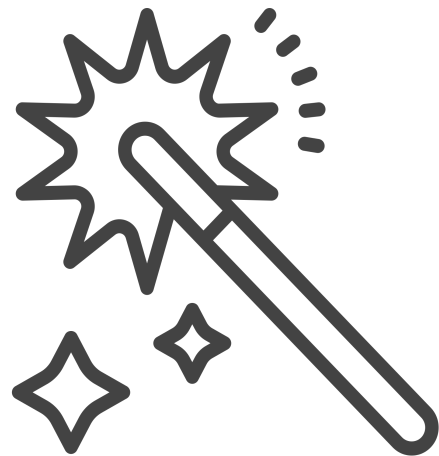
O que um mago precisa aprender ?

Palavras mágicas

Git é um grimório e você precisa usar a magia certa em cada ocasião.

(Clone, push, pull, add, commit...)

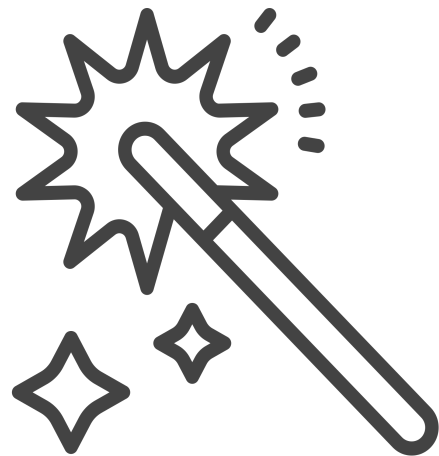
Sempre use **git <comando>**



Iniciando um repositório

`git init`

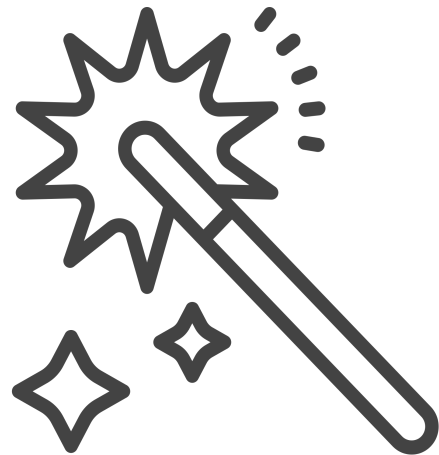
Define o diretório atual como um repositório.



Verificando arquivos

`git status`

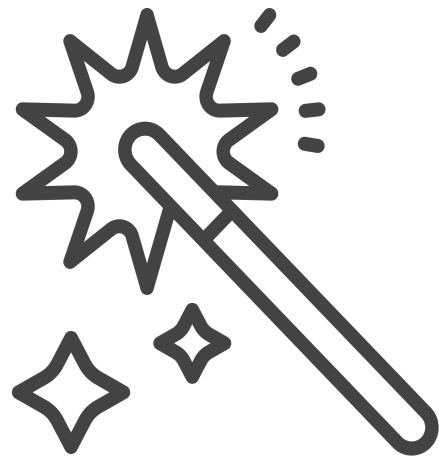
Verifica o estado dos arquivos no repositório atual.



Adicionando arquivos

`git add <nome>`

Verifica o estado dos arquivos no repositório atual.



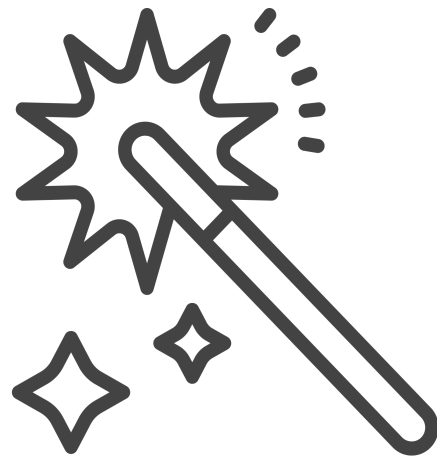
Adicionando arquivos

```
git add .
```

```
git add *.doc
```

Formas variadas de adicionar arquivos.

*mas cuidado com essas magias, elas podem adicionar arquivos que talvez você não queira.



Ignorando arquivos

.gitignore

Escreva regras e ignore
pastas, arquivos, extensões



Retirando arquivos do STAGED

`git reset <filename>`

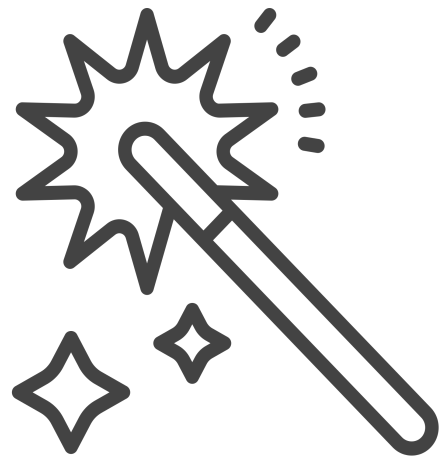
Remove o arquivo da lista que irá para o commit (STAGED)



Não queria ter modificado isso !?

`git checkout -- <filename.ext>`

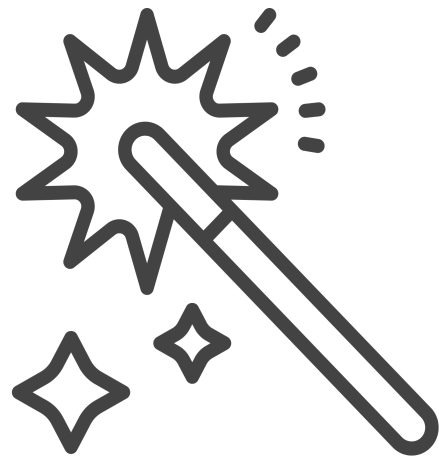
Remove alterações no arquivo apontado.



Commitando suas mudanças

`git commit -m "your comments for the commit"`

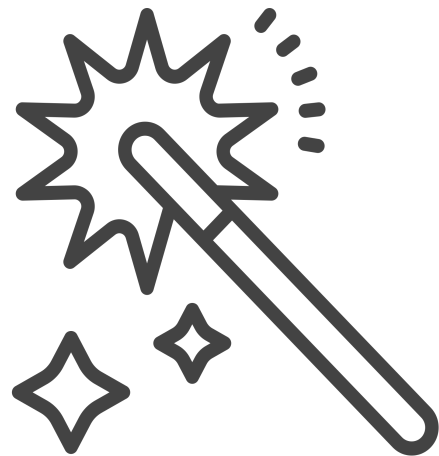
Envia seus arquivos para o repositório local com a “mensagem”.



Commitando suas mudanças

git commit

Abre um editor(vi) de texto para você escrever o commit



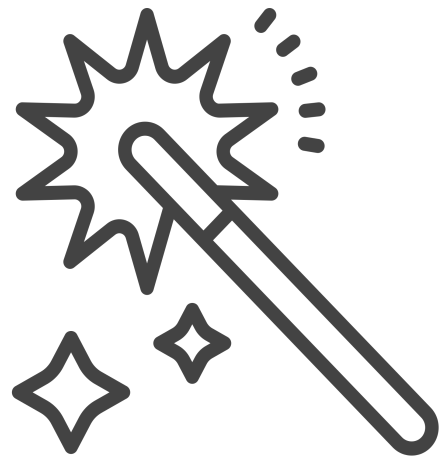
Errei o commit e agora ? (reescrevendo a historia)

[1] `git commit --amend`

[2] `git commit --amend --no-edit`

[1] Abre um editor(vi) para você
reescrever a msg do commit

[2] Apenas coloca os arquivos do
staged junto com o commit



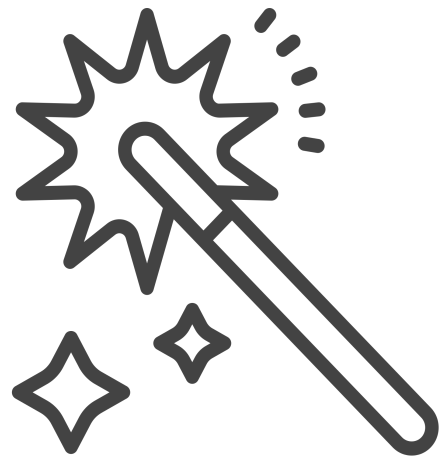
Errei o commit e agora ? (desfazendo a historia)

```
git reset HEAD^1 [--soft | --mixed| --hard ]
```

--soft desfaz commit e deixa arquivos em staged

--mixed desfaz commit e deixa arquivos fora do staged, mas com as modificações.

--hard desfaz commit e tira arquivos do staged e desfaz as modificações.

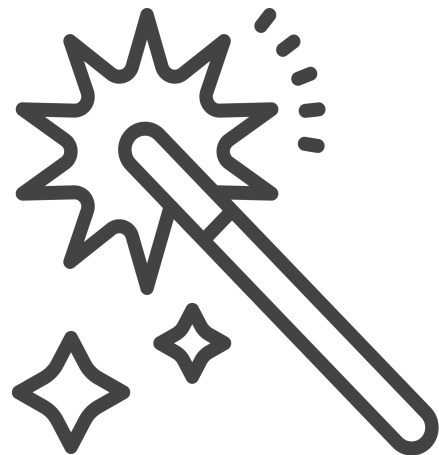


git log


git log --graph

gitk

Um grimório com histórico de ações.



git log<enter>




```
commit 93611a7f57877397425d61b0473710629f5e5d88
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:21:54 2012 +0530

    Added more text which explains why I use Git

commit 8b4fe08f90a0389879de122aa8b7846c01430031
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:16:10 2012 +0530

    Initial commit to showcase the commit functionality of Git
```


git log<enter>



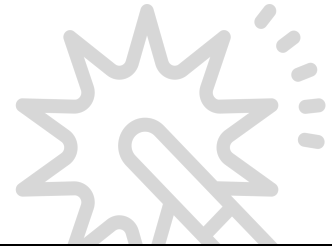
```
commit 93611a7f57877397425d61b0473710629f5e5d88
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:21:54 2012 +0530

    Added more text which explains why I use Git

commit 8b4fe08f90a0389879de122aa8b7846c01430031
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:16:10 2012 +0530

    Initial commit to showcase the commit functionality of Git
```

git log<enter>



```
commit 93611a7f57877397425d61b0473710629f5e5488
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:21:54 2012 +0530
```

Added more text which explains why I use Git

```
commit 8b4fe08f90a0389879de122aa8b7846c01430031
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:16:10 2012 +0530
```

Initial commit to showcase the commit functionality of Git

git log<enter>




```
commit 93611a7f57877397425d61b0473710629f5e5d88
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:21:54 2012 +0530

    Added more text which explains why I use Git

commit 8b4fe08f90a0389879de122aa8b7846c01430031
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:16:10 2012 +0530

    Initial commit to showcase the commit functionality of Git
```



5 primeiros

git checkout <commit ID>



```
commit 93611a7f57877397425d61b0473710629f5e5d88
Author: Ravishankar Somasundaram <raviepic3@gmail.com>
Date:   Fri Jan 27 16:21:54 2012 +0530

    Added more text which explains why I use Git
```

Voltando no tempo

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git (master)
```

```
$ git checkout 116401
```

```
Note: checking out '116401'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado a fim de evitar perdas
```

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git ((1164014...))
```

```
$ |
```

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git (master)
```

```
$ git checkout 116401
```

```
Note: checking out '116401'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado a fim de evitar perdas
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado
```

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git ((1164014...))
```

```
$ |
```




```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git (master)
```

```
$ git checkout 116401
```

```
Note: checking out '116401'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado a fim de evitar perdas
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado
```

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git ((1164014...))
```

```
$ |
```



```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git (master)
```

```
$ git checkout 116401
```

```
Note: checking out '116401'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado a fim de evitar perdas
```

```
HEAD is now at 1164014... Adicionado backup do texto que será usado
```

```
djanilson@skanp-pc MINGW64 ~/work/Pessoal/curso-git ((1164014...))
```

```
$ |
```

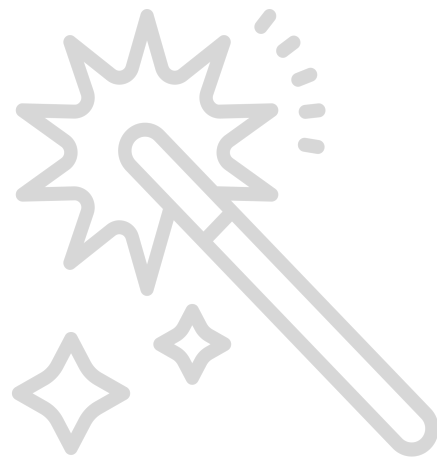


Voltando no tempo

5 primeiros
git checkout <commit ID>



```
HEAD is now at 1164014... Adicionado backup do texto que será usado  
djani1son@skanp-pc MINGW64 ~/work/Pessoal/curso-git ((1164014...))  
$ |
```



4.

Repositório *remote* e local

Seu código na internet

Senário 1 - Single Player

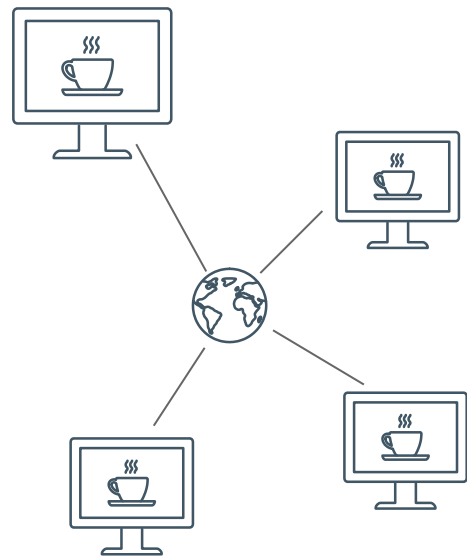
Seu game disponível para você apenas continuar.



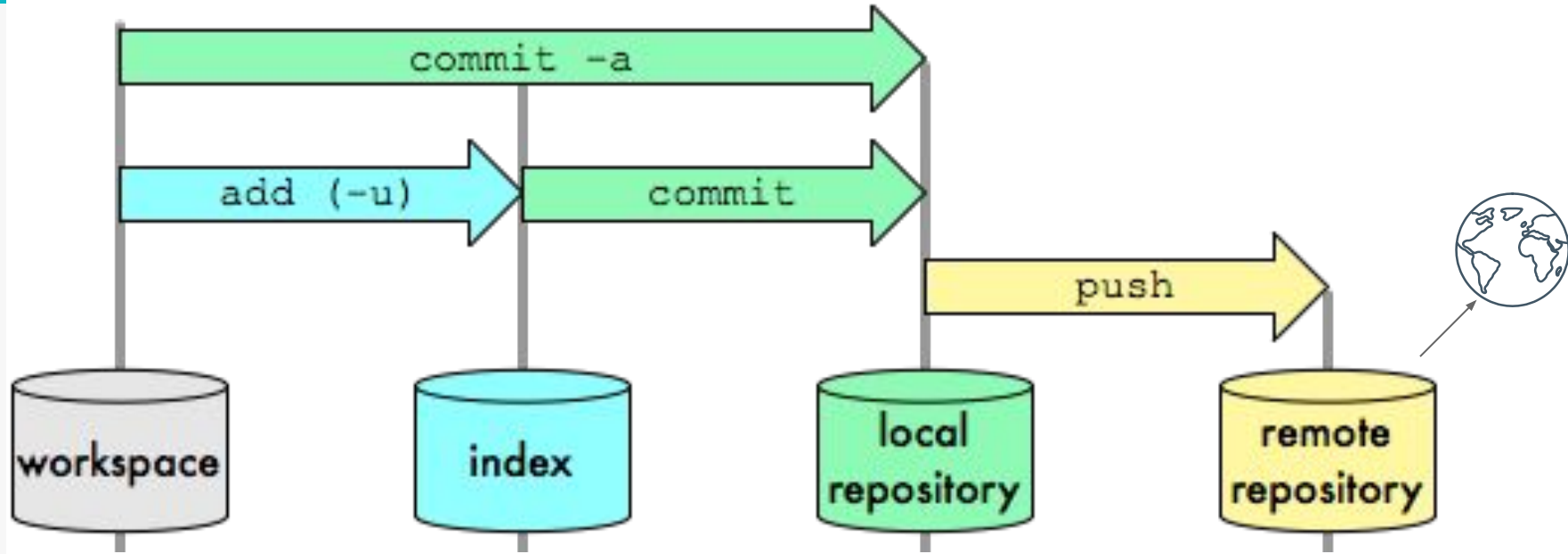
Senário 1 – Single Player

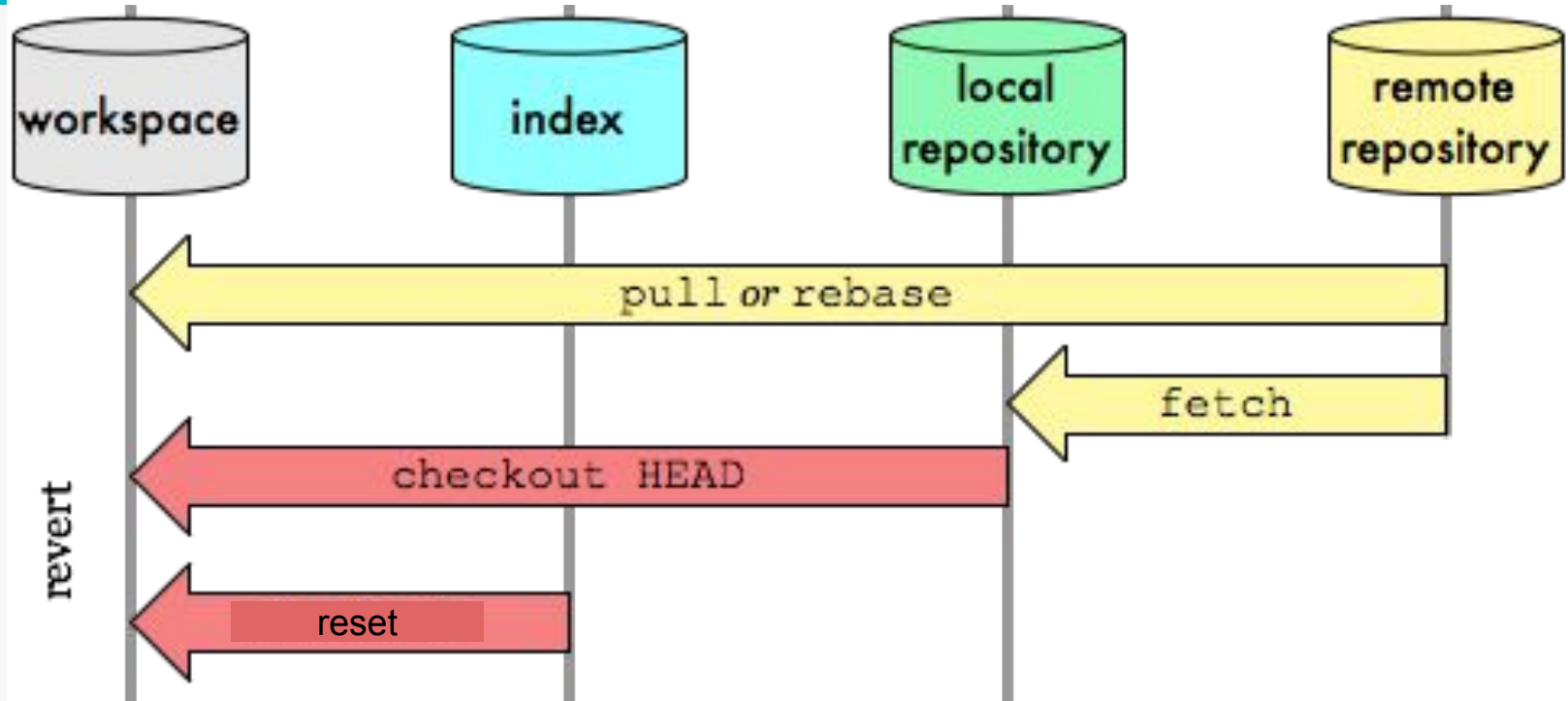
Seu game disponível para você apenas continuar.

Sem você precisar começar do zero toda vez.



Git Data Transport



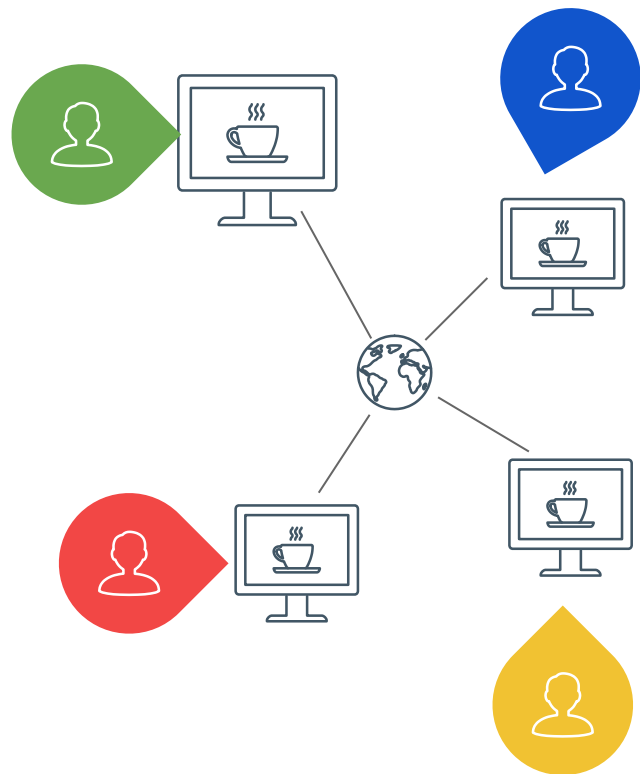


Senário 2 - Multiplayer

Seu game disponível para você apenas continuar.

Sem você precisar começar do zero toda vez.

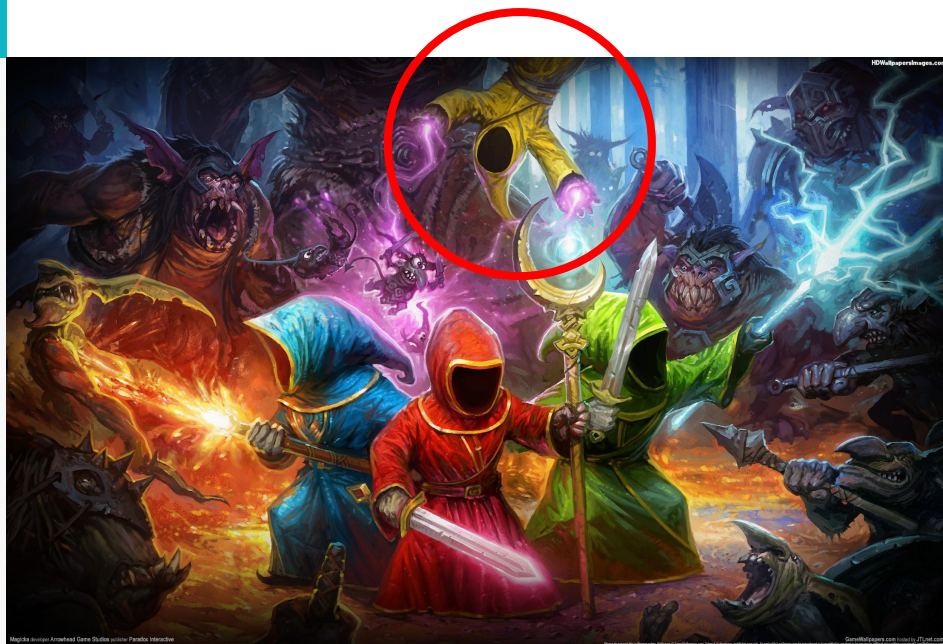
E você pode chamar outros players





Melhor que um mago, são muitos magos juntos.

”



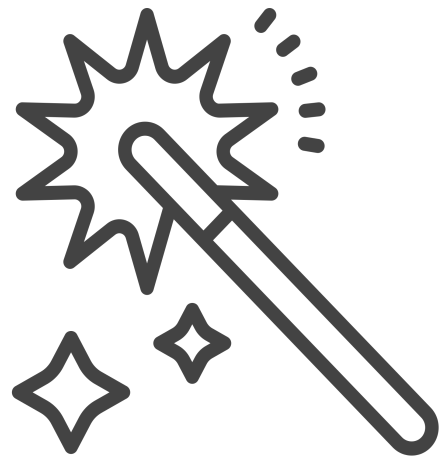
Melhor que um mago, são muitos magos juntos.

”

Superando desafios em grupo

Considere que se você não consegue seguir para o **level** seguinte, um amigo com um **build** diferente pode fazer isso para você e depois você continua.

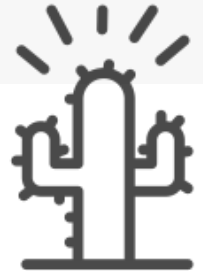
(ou fazer mais rápido)



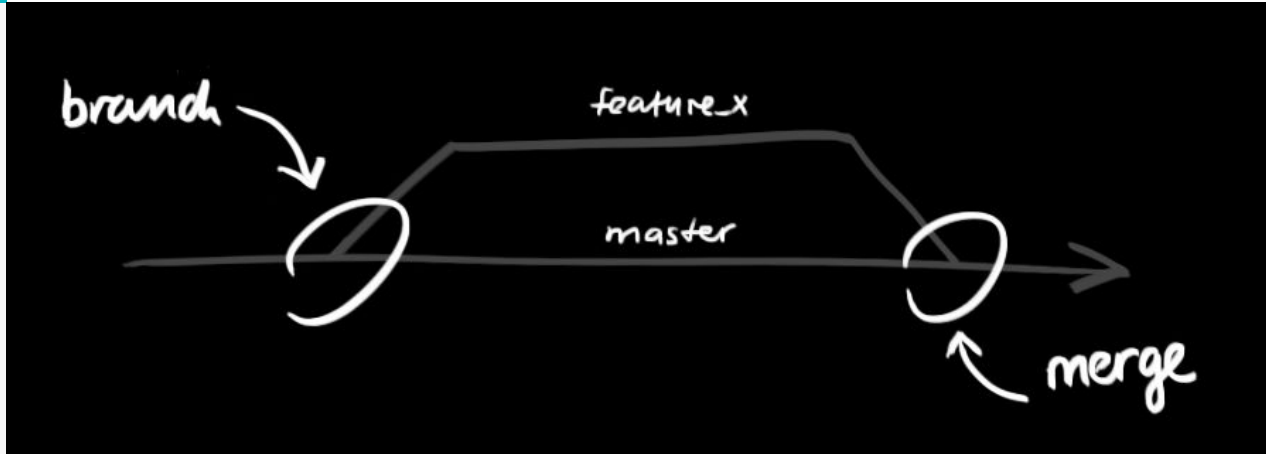
5.

Branchs

Viagem entre dimensões



Dividir para conquistar



Dividir para conquistar

Branchs

Utilizado para criar uma cópia do **workspace**, com um novo nome, a fim de trabalhar em uma tarefa específica.

Pq usar?

Experimento, resolver bugs, separar o que vai ou não para o cliente, gerenciar tarefas.



Branchs: Criando e Listando

git checkout -b <name>

Cria uma nova branch com o nome específico.

git branch [-r]

Lista as branchs disponiveis. [-r] exibirá as branch no repositório local.



Branchs: Removendo

`git branch -D <nome>`

D maiúsculo Deleta a branch localmente.

`git branch -d <nome>`

D minúsculo Deleta a branch localmente se tiver sido feito merge com a branch atual.



5 Novas magias

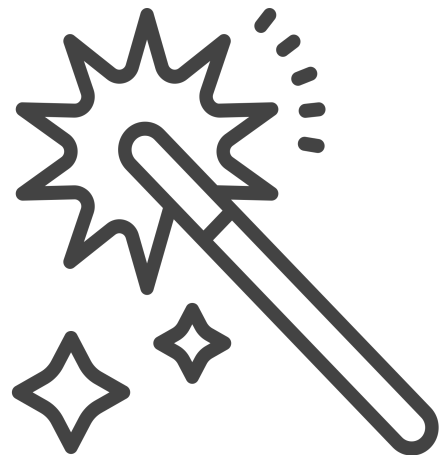
git remote

git clone

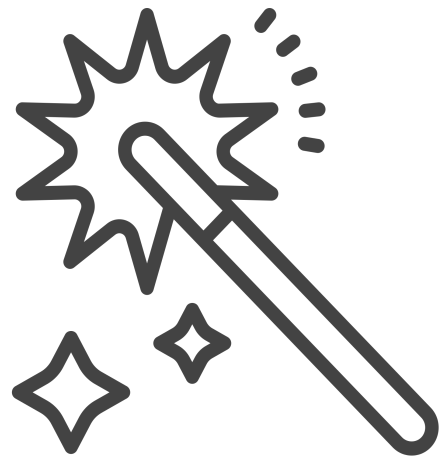
git fetch

git push

git pull

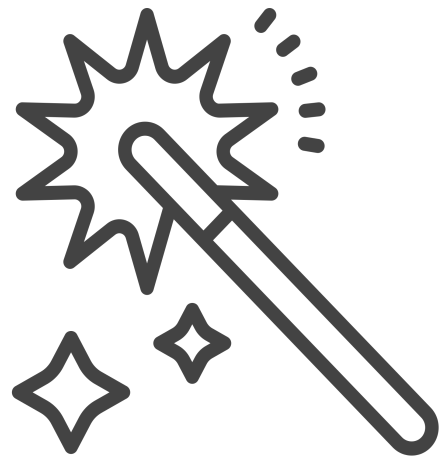



```
git remote <command> <alias> <url>
```



```
git remote <command> <alias> <url>
```

```
git remote add origin <url>
```

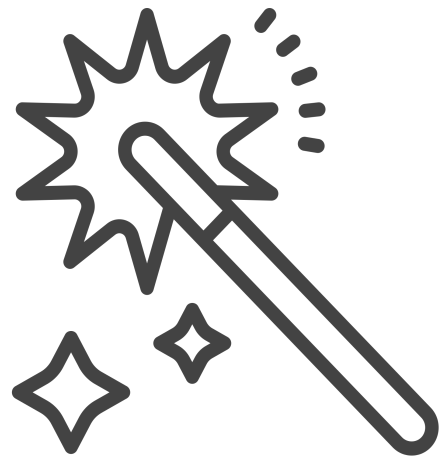


```
git remote <command> <alias> <url>
```

```
git remote add origin <url>
```

Esse comando insere um *git repository* com uma o nome de origin.

Este habilitará você de usar **push** & **pull**

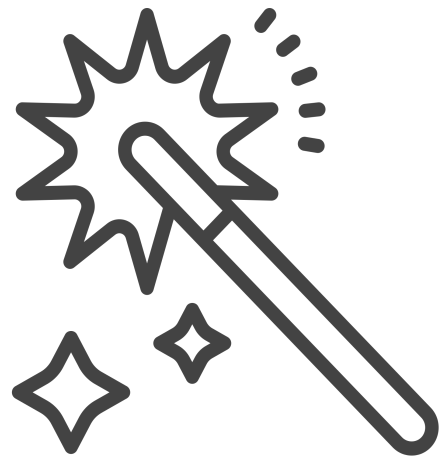


```
git remote <command> <alias> <url>
```

```
git remote add origin <url>
```

Esse comando insere um *git repository* com uma o nome de origin.

Este habilitará você de usar **push** & **pull**

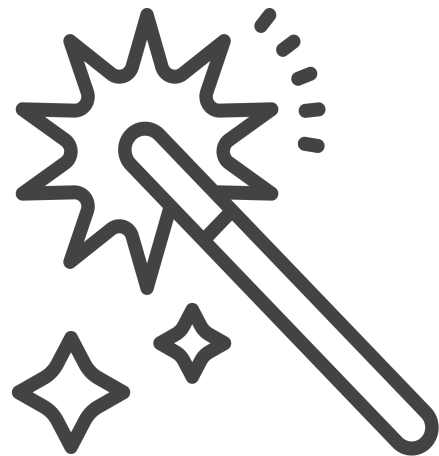


‘Catando’ os checkpoints

```
git pull <remote>
```

```
git pull origin
```

Ele irá buscar do servidor todas as modificações e inserir no seu **repo** local alterando seu workspace com as modificações.



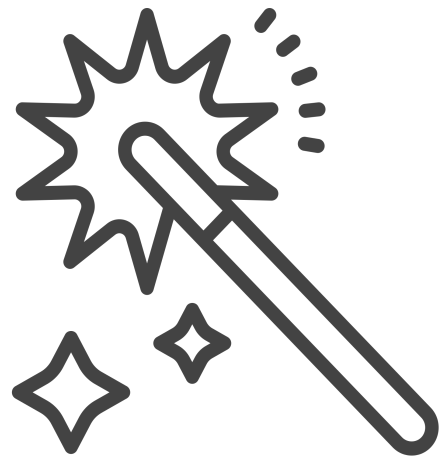
‘Catando’ os checkpoints

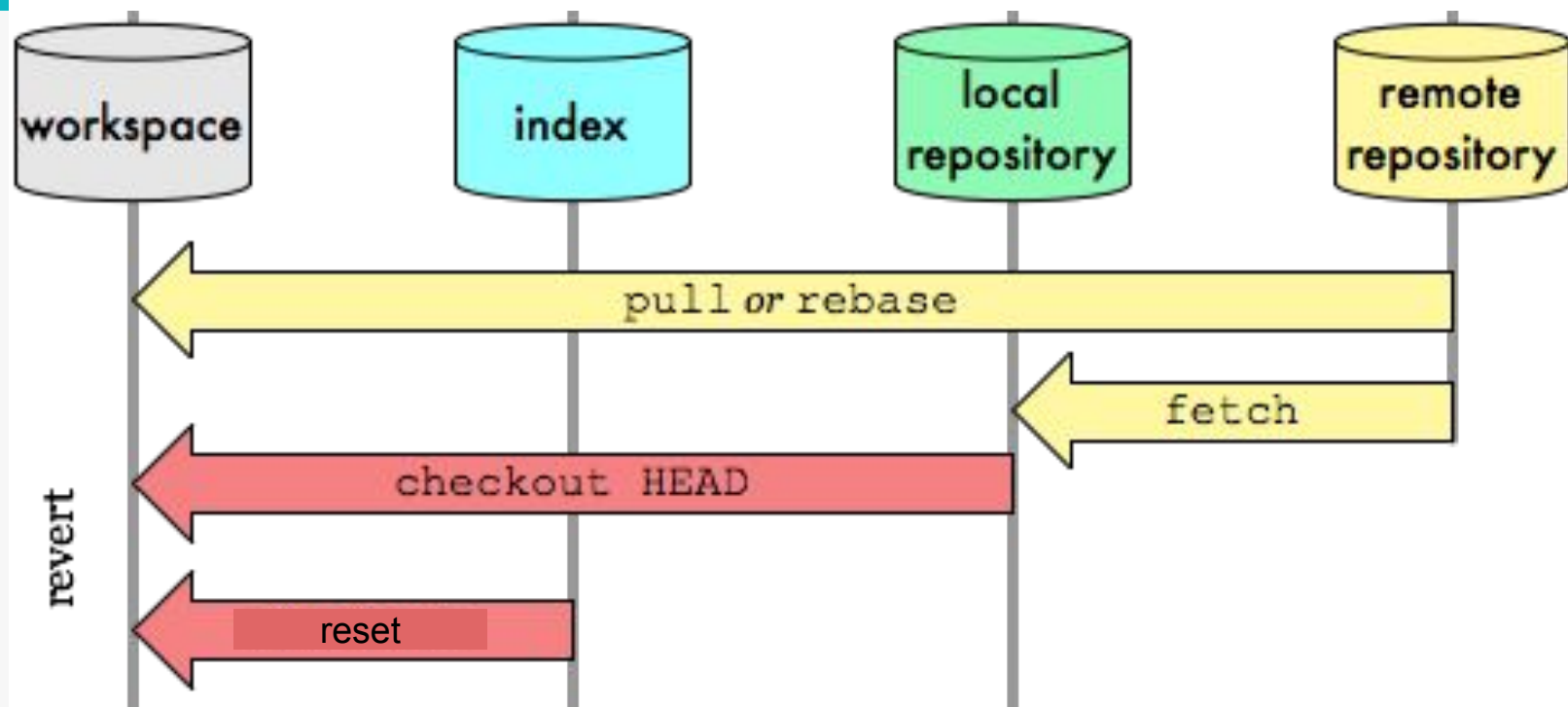
`git fetch <remote>`

`git fetch origin`

Ele irá buscar do servidor todas as modificações e inserir no seu **repo** local.

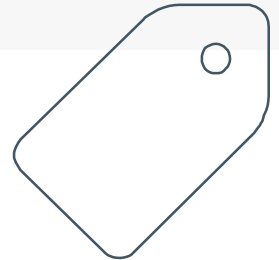
*Porém não irá modificar seu workspace;





6.

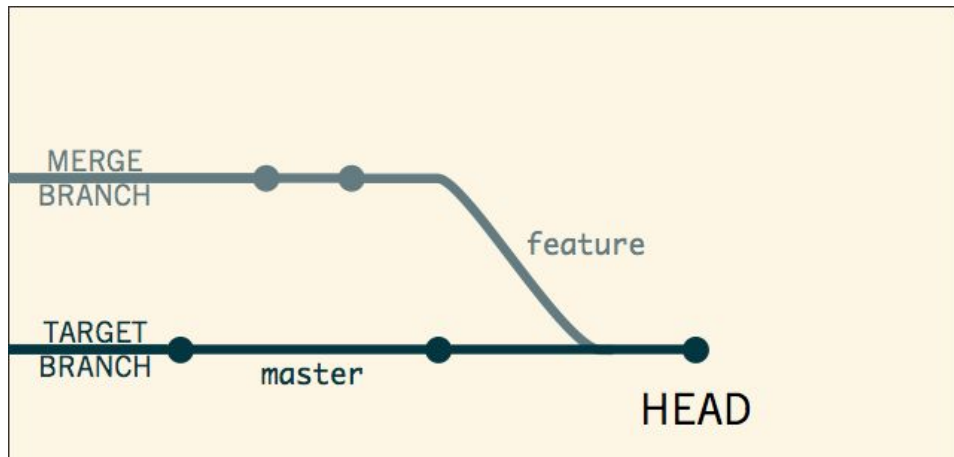
Merge – Unindo branches



Unindo branches

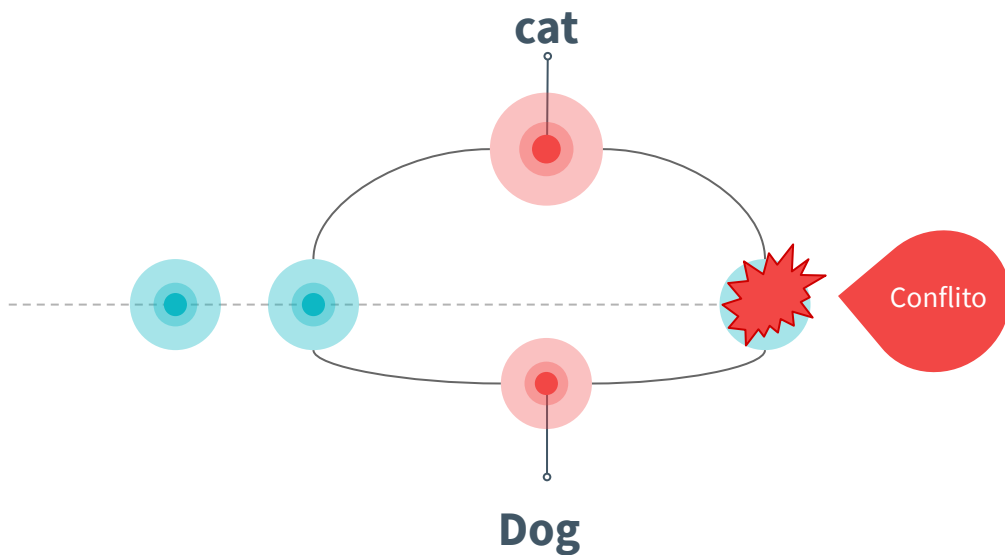
`git merge <name>`

Une a branch informada com a branch atual.



Merge CONFLICT

Quando duas pessoas mudam o mesmo trecho do arquivo o merge pode dar conflito



Merge CONFLICT

```
<<<<<< HEAD
```

```
Unchanged first line from source = Not any more ;) - Lisa
```

```
=====
```

```
First line from source - Changed by Bob
```

```
>>>>>> 9bab0336e6c9ab984b538f1f7724bf8a9703f55e
```

```
Second line
```

```
Third line
```

```
Fourth line by Lisa
```



Merge CONFLICT

```
<<<<<< HEAD
```

```
Unchanged first line from source = Not any more ;) - Lisa
```

```
-----  
First line from source - Changed by Bob
```

```
>>>>>> 9bab0336e6c9ab984b538f1f7724bf8a9703f55e
```

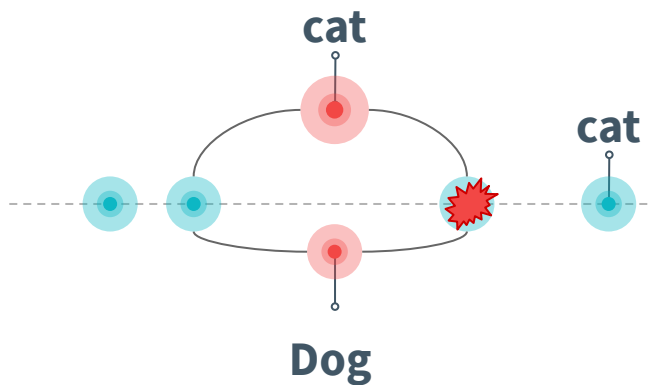
Second line

Third line

Fourth line by Lisa



Merge CONFLICT

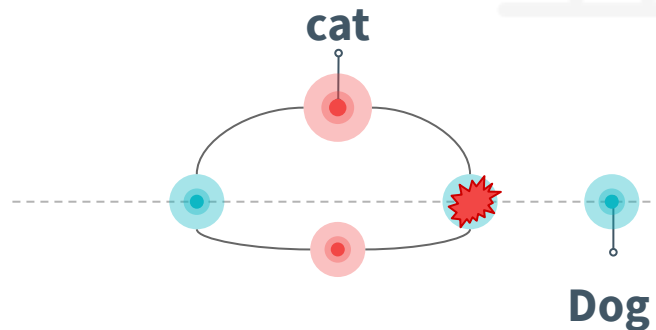


`git checkout --theirs`

Faz o **Merge** favorecendo a mudança deles.

`git checkout --ours`

Faz o **Merge** favorecendo nossas mudança.



Merge CONFLICT

```
<<<<<< HEAD
```

```
Unchanged first line from source = Not any more ;) - Lisa
```

```
-----  
First line from source - Changed by Bob
```

```
>>>>>> 9bab0336e6c9ab984b538f1f7724bf8a9703f55e
```

Second line

Third line

Fourth line by Lisa



Resolvendo conflitos de merge

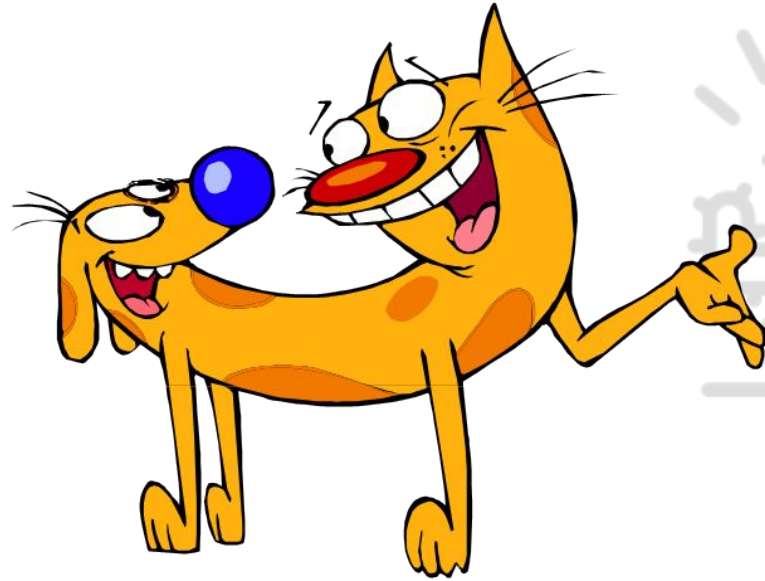
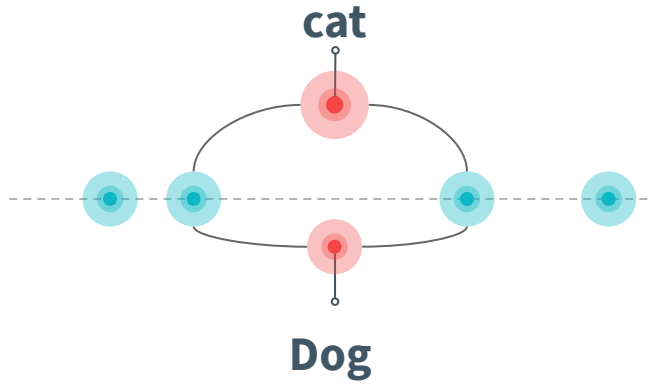
git commit

Uma mensagem automática de merge irá ser escrita.

Merge branch 'adicionar-guerreiros'



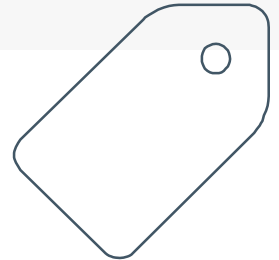
Mrege resolved



6.

Tags e Versionamento semântico

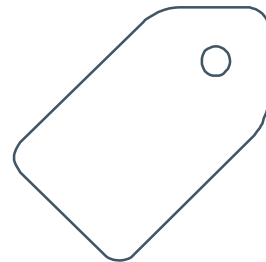
Criando números mágicos **x.y.z**



SemVer – Semantic Version

Considere uma biblioteca chamada “CaminhaoBombeiros”. Ela requer um pacote versionado dinamicamente chamado “Escada”. Quando CaminhaoBombeiros foi criado, Escada estava na versão 3.1.0. Como CaminhaoBombeiros utiliza algumas funcionalidades que foram inicialmente introduzidas na versão 3.1.0, você pode especificar, com segurança, a dependência da Escada como maior ou igual a 3.1.0 porém menor que 4.0.0.

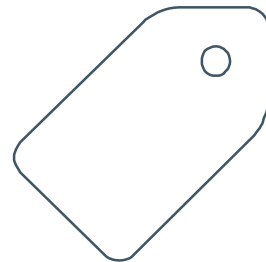
Agora, quando Escada versão 3.1.1 e 3.2.0 estiverem disponíveis, você poderá lançá-los ao seu sistema de gerenciamento de pacote e saberá que eles serão compatíveis com os softwares dependentes existentes.



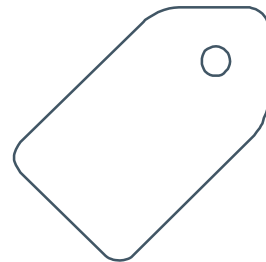
SemVer – Semantic Version

Esta não é uma ideia nova ou revolucionária. De fato, você provavelmente já faz algo próximo a isso.

O mundo real é um lugar bagunçado; não há nada que possamos fazer quanto a isso senão sermos atentos. O que você pode fazer é deixar o Versionamento Semântico lhe fornecer uma maneira sensata de lançar e atualizar pacotes sem precisar atualizar para novas versões de pacotes dependentes, salvando-lhe tempo e aborrecimento.



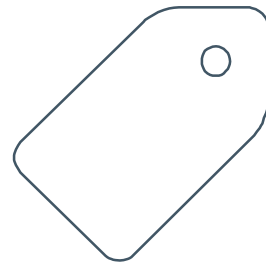
As palavras-chaves “**DEVE**”, “**NÃO DEVE**”, “**OBRIGATÓRIO**”, “**DEVERÁ**”, “**NÃO DEVERÁ**”, “**PODEM**”, “**NÃO PODEM**”, “**RECOMENDADO**” e “**OPCIONAL**” no presente documento devem ser interpretados como descrito na [RFC 2119](#).



Especificação de Versionamento

x.y.z (major.minor.patch)

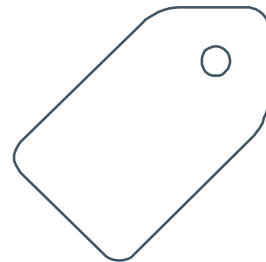
- ▶ Software usando Versionamento Semântico DEVE declarar uma API pública. Esta API poderá ser declarada no próprio código ou existir estritamente na documentação, desde que seja precisa e compreensiva.
- ▶ Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda. X é a versão **Major**, Y é a versão **Minor**, e Z é a versão de **patch**.
- ▶ Uma vez que um pacote versionado foi lançado(released), o conteúdo desta versão NÃO DEVE ser modificado. Qualquer modificação DEVE ser lançado como uma nova versão.



Especificação de Versionamento

x.y.Z (major.minor.patch)

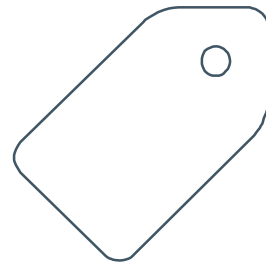
- ▶ No início do desenvolvimento, a versão *Major* DEVE ser zero (0.y.z). Qualquer coisa PODE mudar a qualquer momento. A API pública NÃO DEVE ser considerada estável.
- ▶ Versão 1.0.0 define a API como pública. A maneira como o número de versão é incrementado após este lançamento é dependente da API pública e como ela muda.
- ▶ Versão de *patch* Z (x.y.Z | x > 0) DEVE ser incrementado apenas se mantiver compatibilidade e introduzir correção de bugs. Uma correção de bug é definida como uma mudança interna que corrige um comportamento incorreto.



Especificação de Versionamento

x.Y.z (major.minor.patch)

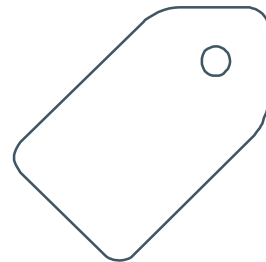
- ▶ Versão *Minor* Y ($x.Y.z \mid x > 0$) DEVE ser incrementada se uma funcionalidade nova e compatível for introduzida na API pública. DEVE ser incrementada se qualquer funcionalidade da API pública for definida como descontinuada. PODE ser incrementada se uma nova funcionalidade ou melhoria substancial for introduzida dentro do código privado. PODE incluir mudanças a nível de correção. A versão de patch DEVE ser definida para 0(zero) quando a versão Menor for incrementada.



Especificação de Versionamento

X.y.z (major.minor.patch)

- ▶ Versão Major X ($X.y.z \mid X > 0$) DEVE ser incrementada se forem introduzidas mudanças incompatíveis na API pública. PODE incluir alterações a nível de versão *Minor* e de versão de *Patch*. Versão de Patch e Versão Minor DEVEM ser redefinidas para 0(zero) quando a versão Major for incrementada.



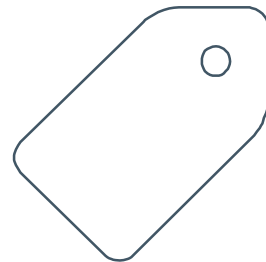
Especificação de Versionamento

X.Y.Z-alpha

- ▶ Uma versão de Pré-Lançamento (pre-release) PODE ser identificada adicionando um hífen (dash) e uma série de identificadores separados por ponto (dot) imediatamente após a versão de Correção. Identificador DEVE incluir apenas caracteres alfanuméricos e hífen [0-9A-Za-z-]. Identificador NÃO DEVE ser vazio. Indicador numérico NÃO DEVE incluir zeros à esquerda. Versão de Pré-Lançamento tem precedência inferior à versão normal a que está associada. Uma versão de Pré-Lançamento (pre-release) indica que a versão é instável e pode não satisfazer os requisitos de compatibilidade pretendidos, como indicado por sua versão normal associada.

Exemplos:

1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.



Especificação de Versionamento

X.Y.Z-alpha+metadata-to-build

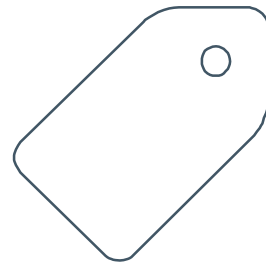
- ▶ Metadados de Build PODE ser identificada por adicionar um sinal de adição (+) e uma série de identificadores separados por ponto imediatamente após a Correção ou Pré-Lançamento.

Identificador DEVE ser composto apenas por caracteres alfanuméricos e hífen [0-9A-Za-z-].

Identificador NÃO DEVE ser vazio. Metadados de build PODEM ser ignorados quando se determina a versão de precedência. Assim, duas versões que diferem apenas nos metadados de construção, têm a mesma precedência.

Exemplos:

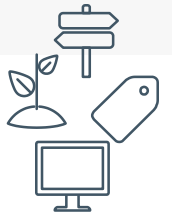
1.0.0-alpha+001, 1.0.0+20130313144700, 1.0.0-beta+exp.sha.5114f85.



7.

SemVer + Git

Git tags



Criando Tags

```
git tag x.x.x
```

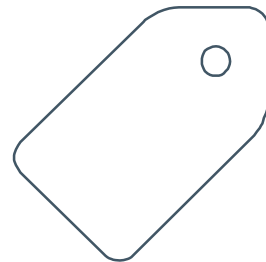
Criará uma nova tag

```
git push --tag | git push origin x.x.x
```

Manda as tags para o *remote*. | manda uma tag para o remote.

```
git tag -l
```

listar as tags



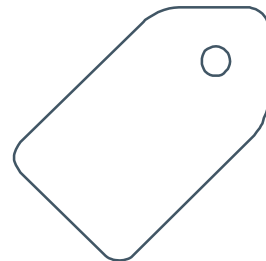
Deletando tags

`git tag -d x.x.x`

Deleta a tag localmente

`git push --delete origin x.x.x`

Deleta a tag no *remote*.



7.

Git Flow

Gerenciando o fluxo de branches



O qué git flow ?

GitFlow é um modelo de ramificação para Git.

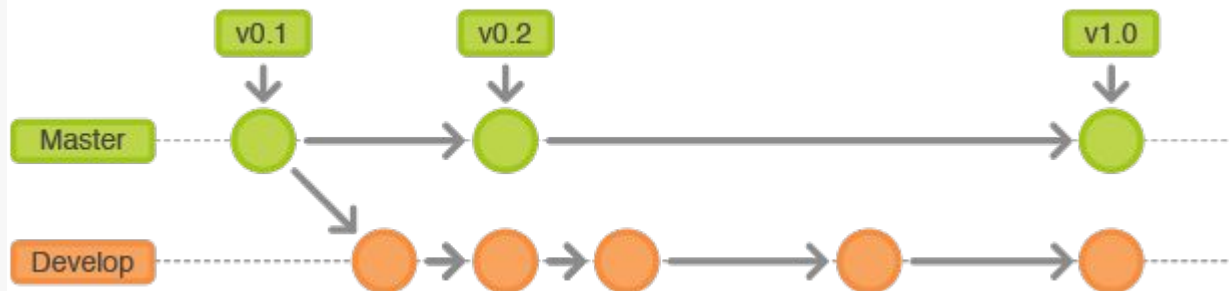
- ▶ Desenvolvimento paralelo
- ▶ Colaboração
- ▶ Area de Release
- ▶ Suporte para correções de emergência

O fluxo geral do Git Flow é:

- ▶ Um branch de **develop** é criado a partir da **main (master)**
- ▶ Um branch de **release** é criado a partir de **develop**
- ▶ **Features** são criadas a partir de **develop**
- ▶ Quando uma feature é concluída, ele é integrada com a **develop**
- ▶ Quando a branch **release** é concluída, ele é integrada com a **develop** e a **main**
- ▶ Se houver um *bug* na master. Cria-se um **hotfix** a partir do **main**
- ▶ Assim que o hotfix for concluído, ele é inserido na **develop** e **main**

O que é git flow ?

Duas Branchs com o histórico do repositório



```
$ git flow init
```

```
Initialized empty Git repository in ~/project/.git/  
No branches exist yet. Base branches must be created now.  
Branch name for production releases: [main]  
Branch name for "next release" development: [develop]
```

```
How to name your supporting branch prefixes?
```

```
Feature branches? [feature/]
```

```
Release branches? [release/]
```

```
Hotfix branches? [hotfix/]
```

```
Support branches? [support/]
```

```
Version tag prefix? []
```

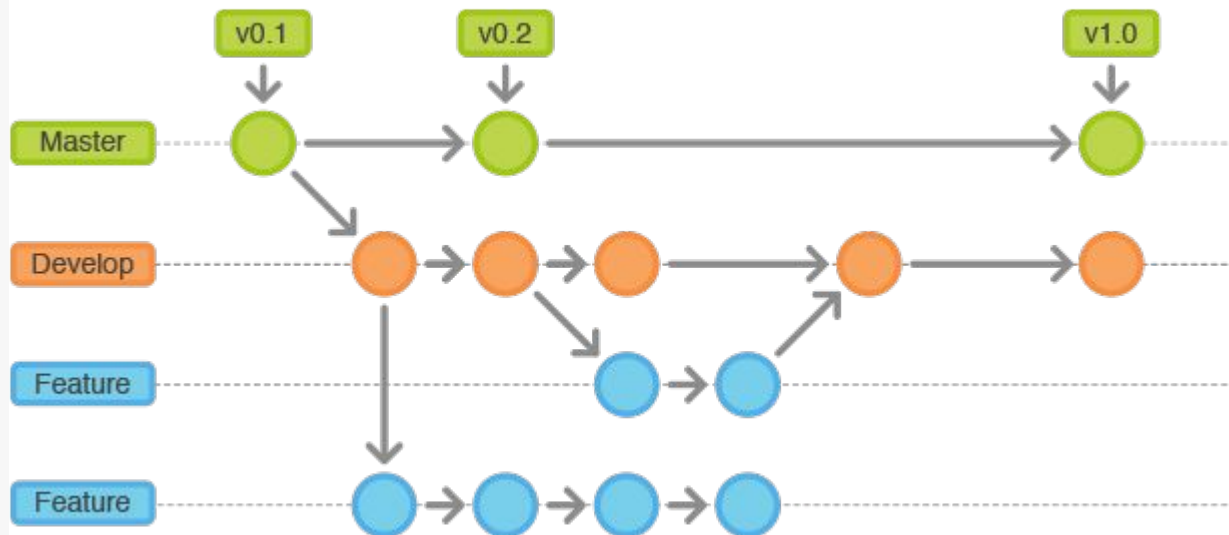
```
$ git branch
```

```
* develop
```

```
main
```

O que git flow ?

Novas *features* nascem da develop e voltam para ela,
Nunca interagem com a master



O que é git flow ?



```
#### FEATURE
```

```
# - Sem plugin
```

```
git checkout develop
```

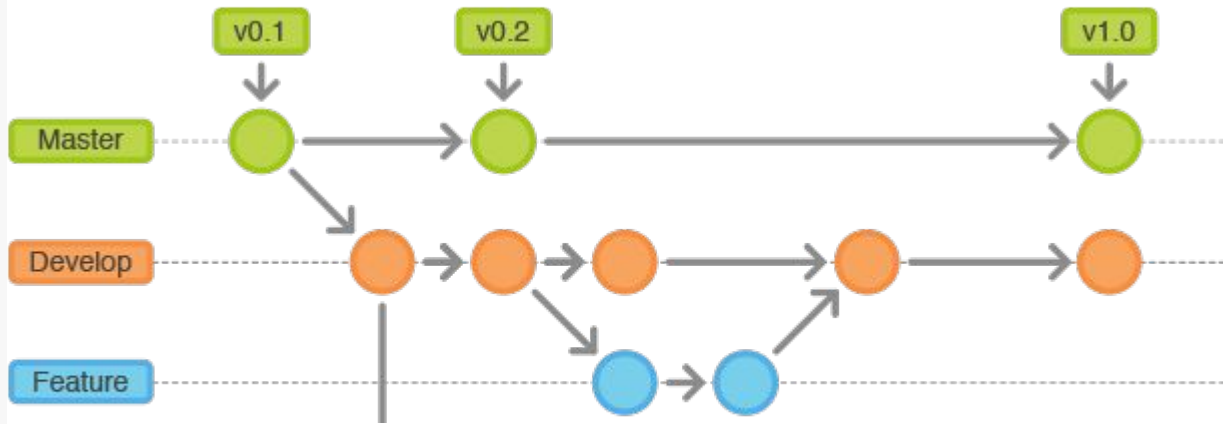
```
git checkout -b feature_branch
```

```
# - com plugin
```

```
git flow feature start feature_branch
```

¿ qué git flow ?

Quando concluída ela será integrada a develop





```
#### FEATURE
```

```
# - Sem plugin
```

```
git checkout develop
```

```
git merge feature_branch
```

```
git branch -d feature_branch
```

```
# - com plugin
```

```
git flow feature finish feature_branch
```

O que git flow ?

Correções urgentes, na **main**, cria-se a branch **hotfix**.

Este é o único branch que deve vir diretamente **main**



O que git flow ?

Correções urgentes, na **main**, cria-se a branch **hotfix**.

Este é o único branch que deve vir diretamente **main**



Semelhante a terminar um **release** branch, um **hotfix** é mesclado em ambos **main** e **develop**.

O que é git flow?

Corre

Este é

Master

Hotfix

Release

Develop



HOTFIX

- Sem plugin

```
git checkout main
```

```
git checkout -b hotfix_branch
```

- com plugin

```
git flow hotfix start hotfix_branch
```



Master

Hotfix

Release

Develop

HOTFIX

- Sem plugin

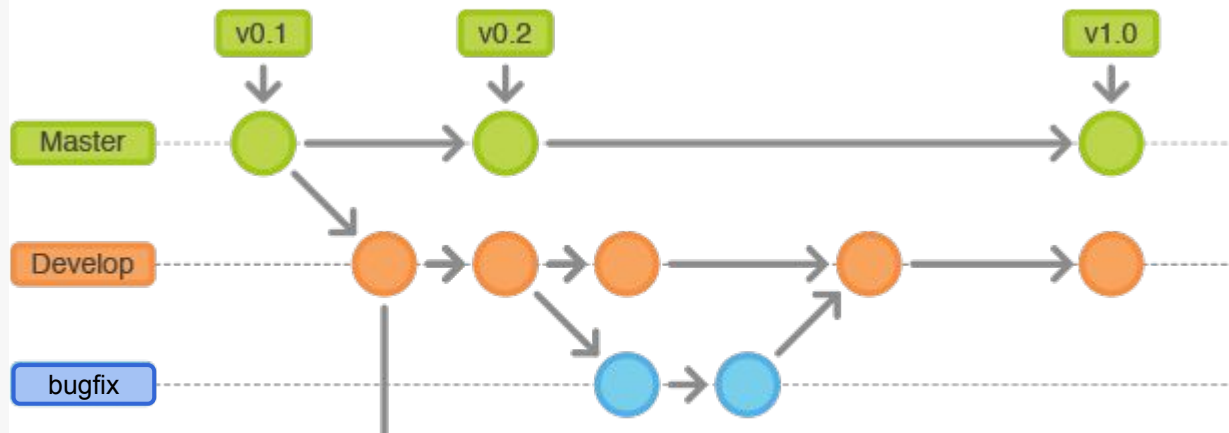
`git checkout main``git merge hotfix_branch``git checkout develop``git merge hotfix_branch``git branch -D hotfix_branch`

- com plugin

`git flow hotfix finish hotfix_branch`

O que git flow ?

Correções que ainda não estão em produção, cria-se a branch **bugfix** e corrige-se o erro, manda para **develop**.



Corre
bran



```
#### BUGFIX
```

```
# - Sem plugin
```

```
git checkout develop
```

```
git checkout -b bugfix_branch
```

```
# - com plugin
```

```
git flow bugfix start bugfix_branch
```

Correção
branch

```
#### BUGFIX
```

```
# - Sem plugin
```

```
git checkout develop
```

```
git merge bugfix_branch
```

```
git branch -D bug_branch
```

```
# - com plugin
```

```
git flow bugfix finish bugfix_branch
```

Release

Depois de develop adquirir recursos suficientes para um lançamento (ou uma data de lançamento predeterminada está se aproximando), você cria a branch de **release** partindo da **develop**.

Ao fechar a branch cria-se uma nova tag da versão

¿ qué git flow ?





```
#### RELEASE
```


```
# - Sem plugin
```

```
git checkout develop
```

```
git checkout -b release/0.1.0
```

```
# - com plugin
```

```
git flow release start 0.1.0
```

```
#### RELEASE
```

```
# - Sem plugin
```

```
git checkout main
```

```
git merge release/0.1.0
```

```
git branch -d release/0.1.0
```

```
git tag 0.1.0
```

```
# - com plugin
```

```
git flow release finish 0.1.0
```

Referências

git - the simple guide

<http://rogerdudler.github.io/git-guide/>

Git: Version Control for Everyone

<https://git-scm.com/book/en/v2>

Git: Version Control for Everyone

<https://www.packtpub.com/application-development/git-version-control-everyone>

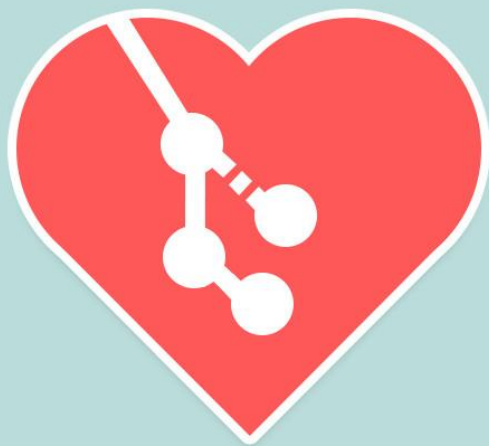
Versionamento Semântico 2.0.0

<http://semver.org/lang/pt-BR/>

Git Flow

<https://leanpub.com/git-flow/read>

<https://nvie.com/posts/a-successful-git-branching-model/>



OBRIGADO!

djanilson.alves@gmail.com