



git

# Workshop GIT

Aprenda Git de uma vez por todas!

**Paulo Igor**

# Apresentações



Inscrição, Emails, Processo, Expectativas, ....

# Dúvidas

- O que já viram?
- O que deu pra experimentar?
- Como foi a experiência até aqui?





git

# Introdução ao Git

História, Topologia e Funcionamento

# História do Git

- **Criado por:**
  - Linus Torvalds
- **Desenvolvido por:**
  - programadores da comunidade Linux
- **Criado em:**
  - abril de 2005
- **Versionado em:**
  - Github (<https://github.com/git/git/graphs/contributors>)
- **Mais informações:**
  - <https://en.wikipedia.org/wiki/Git>



# Características

- Forte suporte ao desenvolvimento não linear
- Desenvolvimento distribuído
- Compatível com os protocolos e sistemas existentes
- Projetado para suporte a grandes projetos
- Autenticação criptografada do histórico
- Projeto baseado em ToolKit
- Estratégias de merge plugáveis
- Garbage Collector
- Utilização eficiente do espaço

# Git e o Mercado!

<http://stackoverflow.com/research/developer-survey-2015#tech-sourcecontrol>

[https://trends.google.com/trends/explore?  
cat=5&q=cvs,svn,git](https://trends.google.com/trends/explore?cat=5&q=cvs,svn,git)

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

# GIT x SVN

## Caso de Uso

Repositório da Mozilla

240.000 arquivos

10 anos de histórico do projeto

**12 GB**

Espaço usado no SVN

**420 MB**

Espaço usado no Git

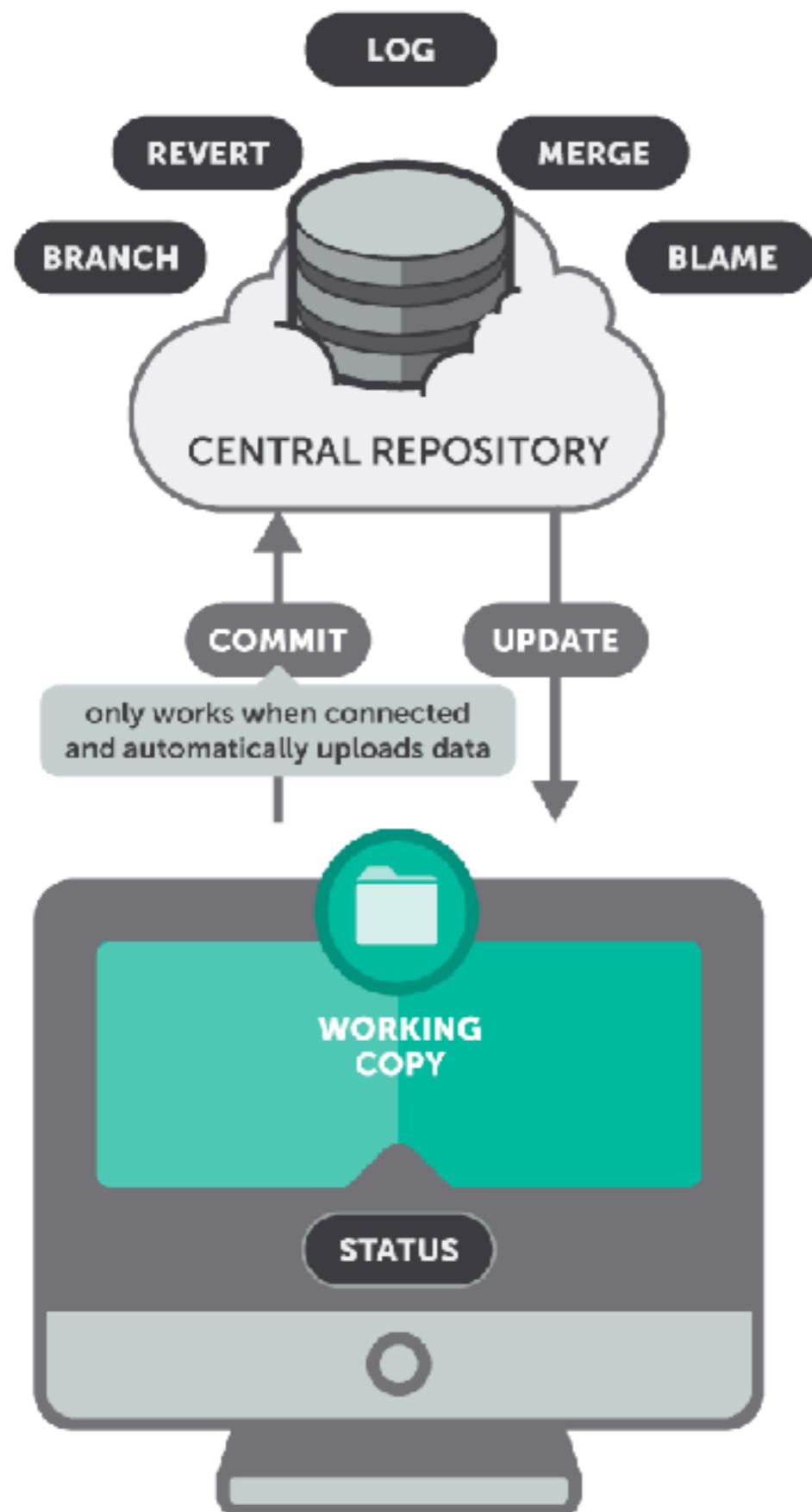
Linux Kernel Development  
Visualization (git commit history  
- past 6 weeks - june 02 2012)

[https://www.youtube.com/watch?v=P\\_02QGsHzEQ](https://www.youtube.com/watch?v=P_02QGsHzEQ)

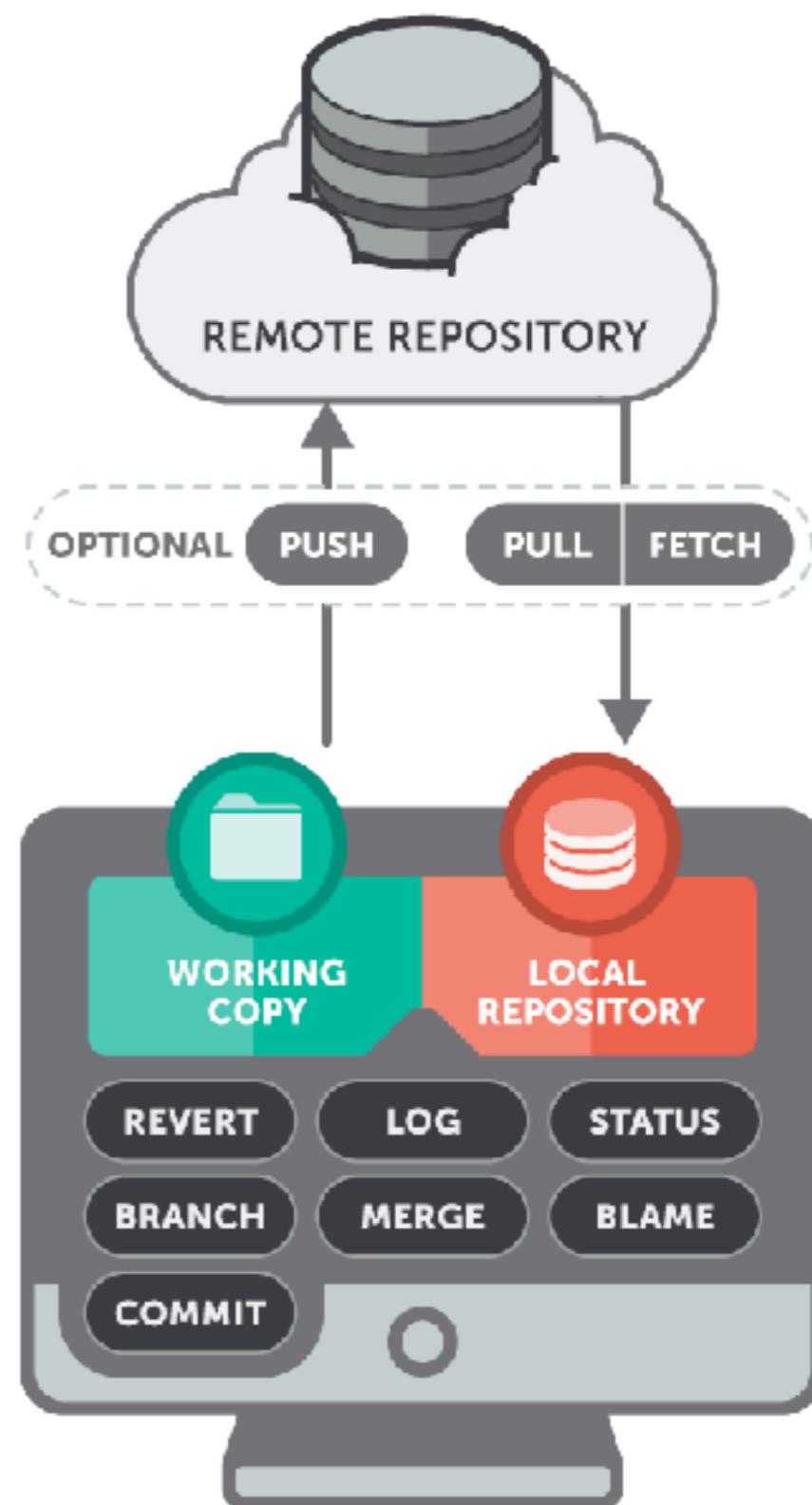
# Modelos de Versionamento

Centralizado vs Descentralizado

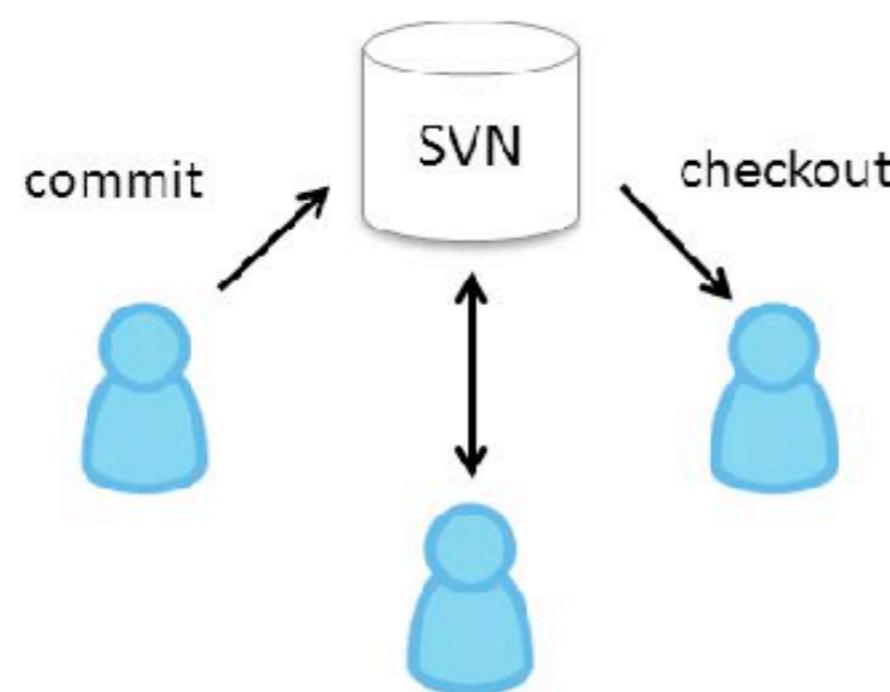
# SUBVERSION



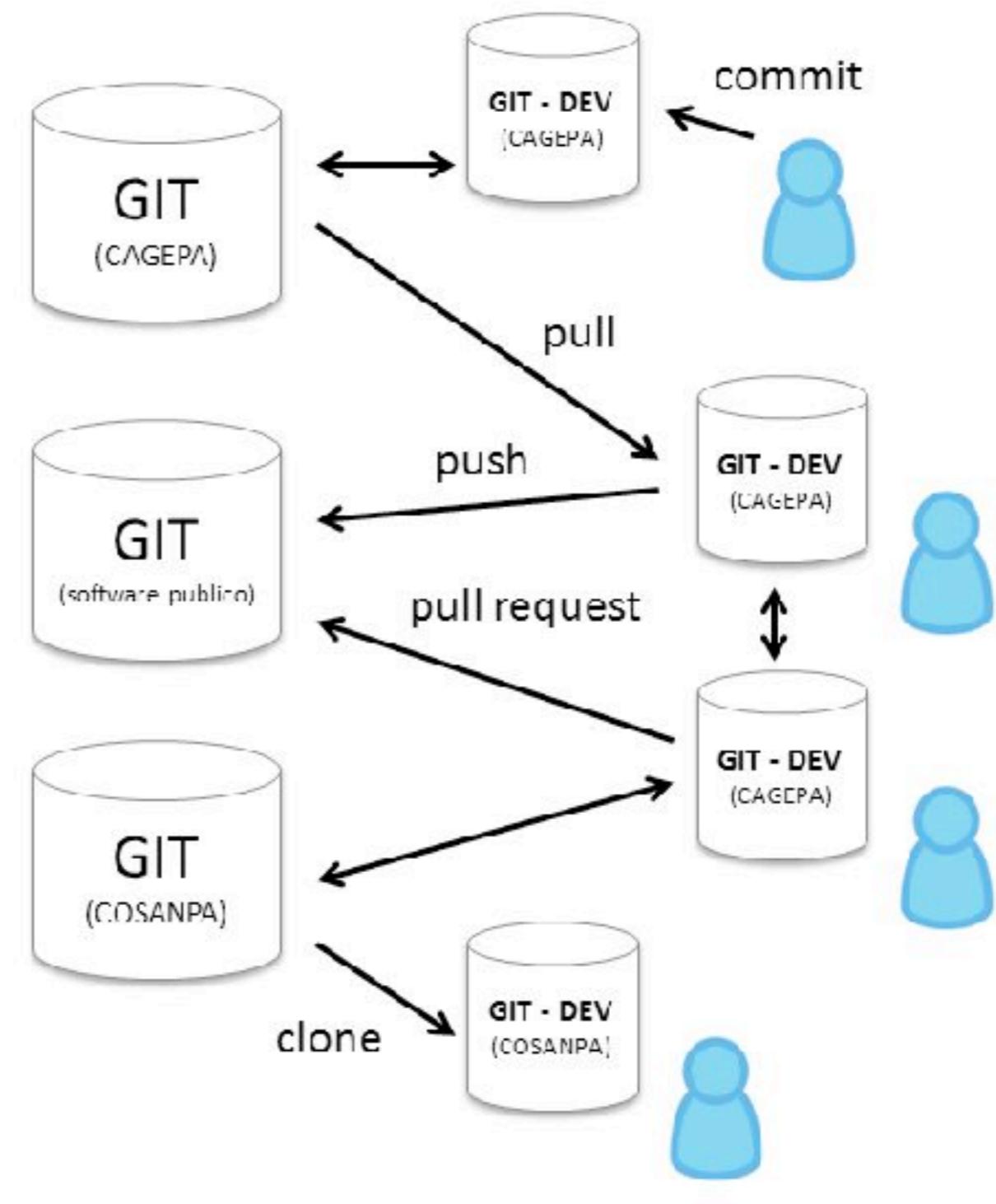
# GIT

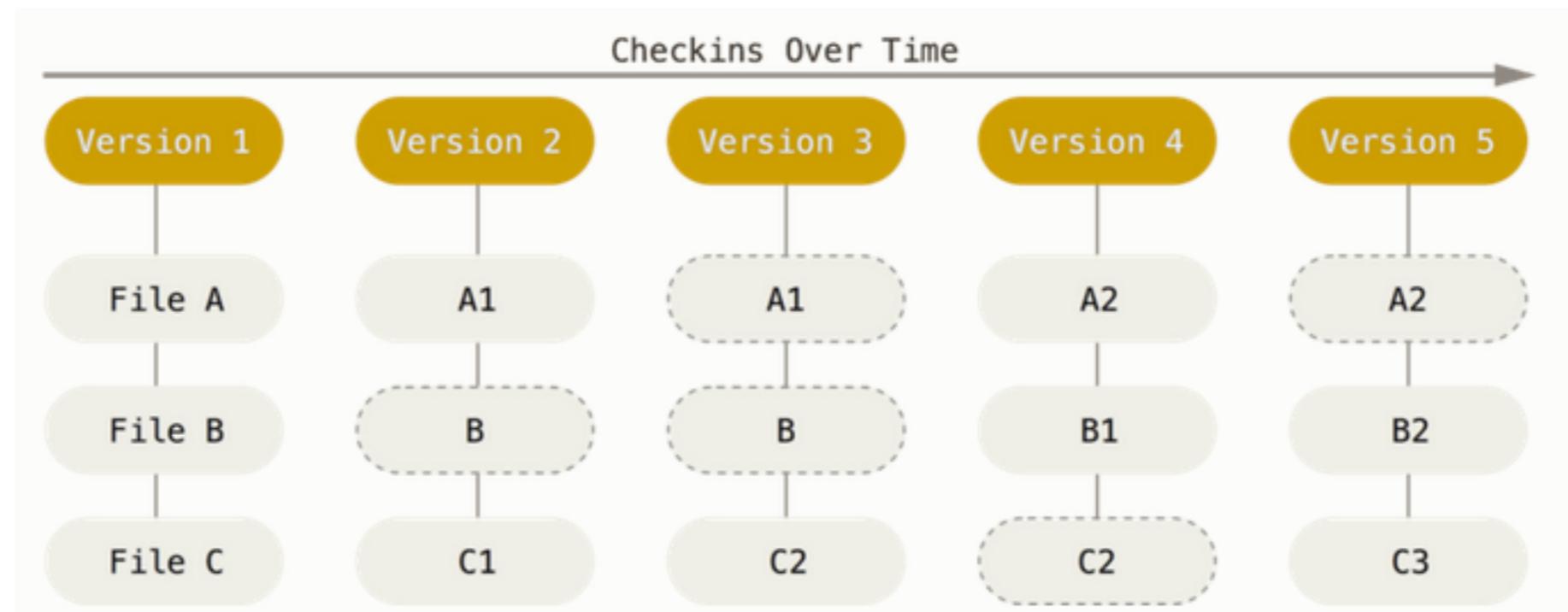
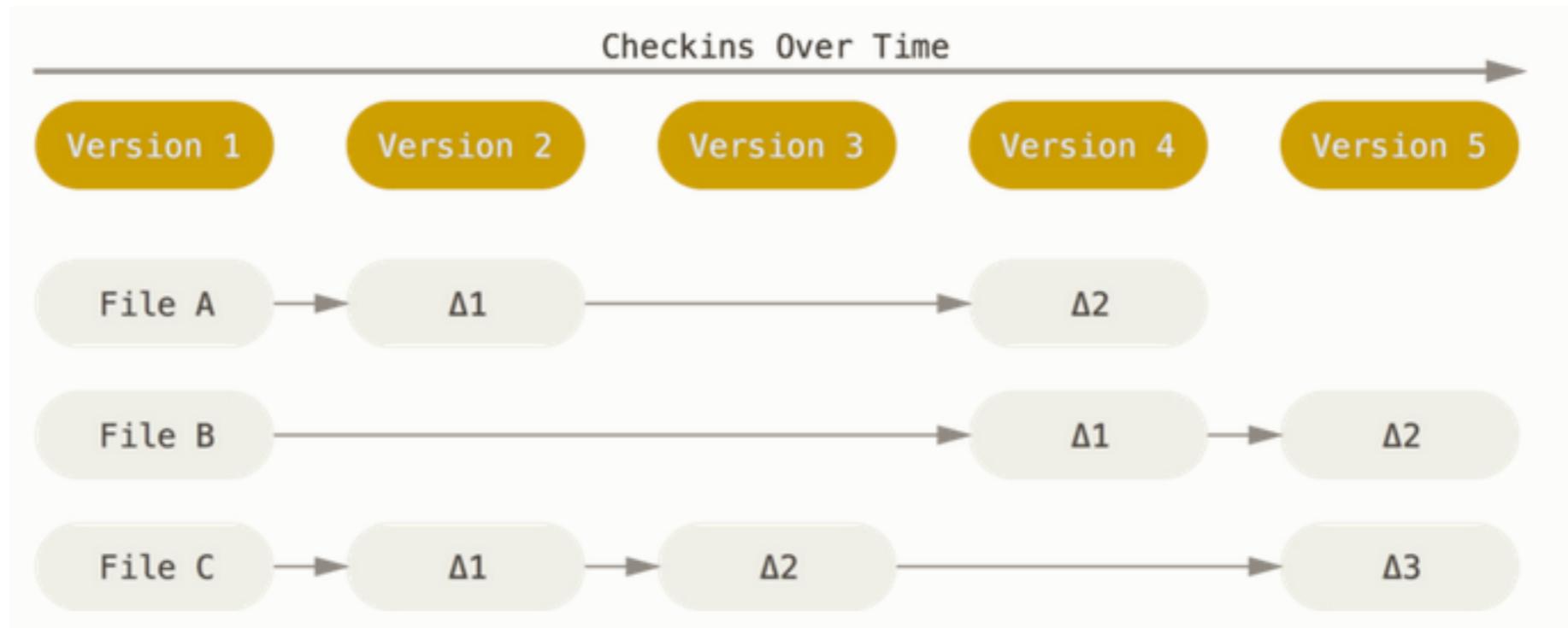


## Modelo Descentralizado

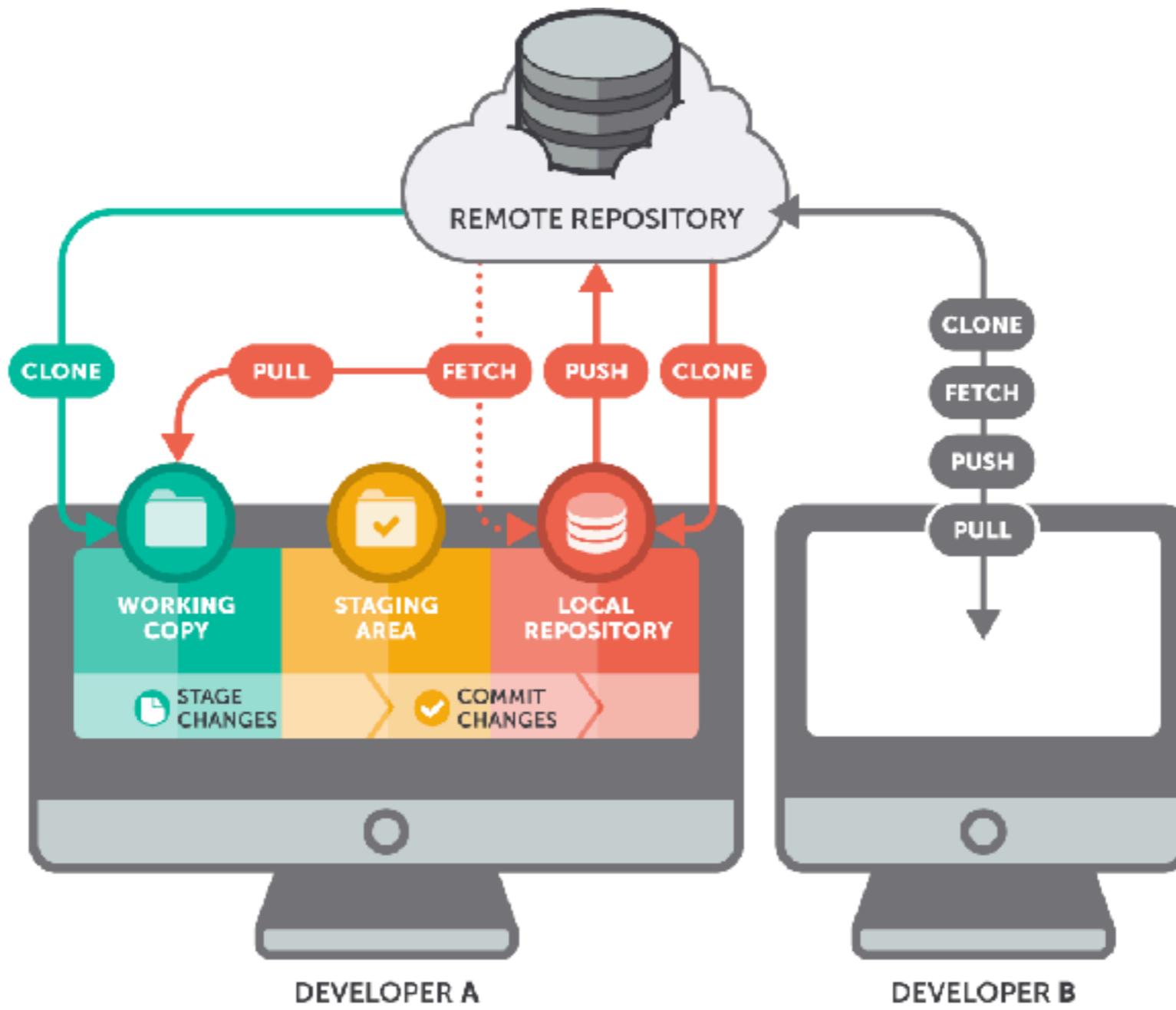


## Modelo Centralizado



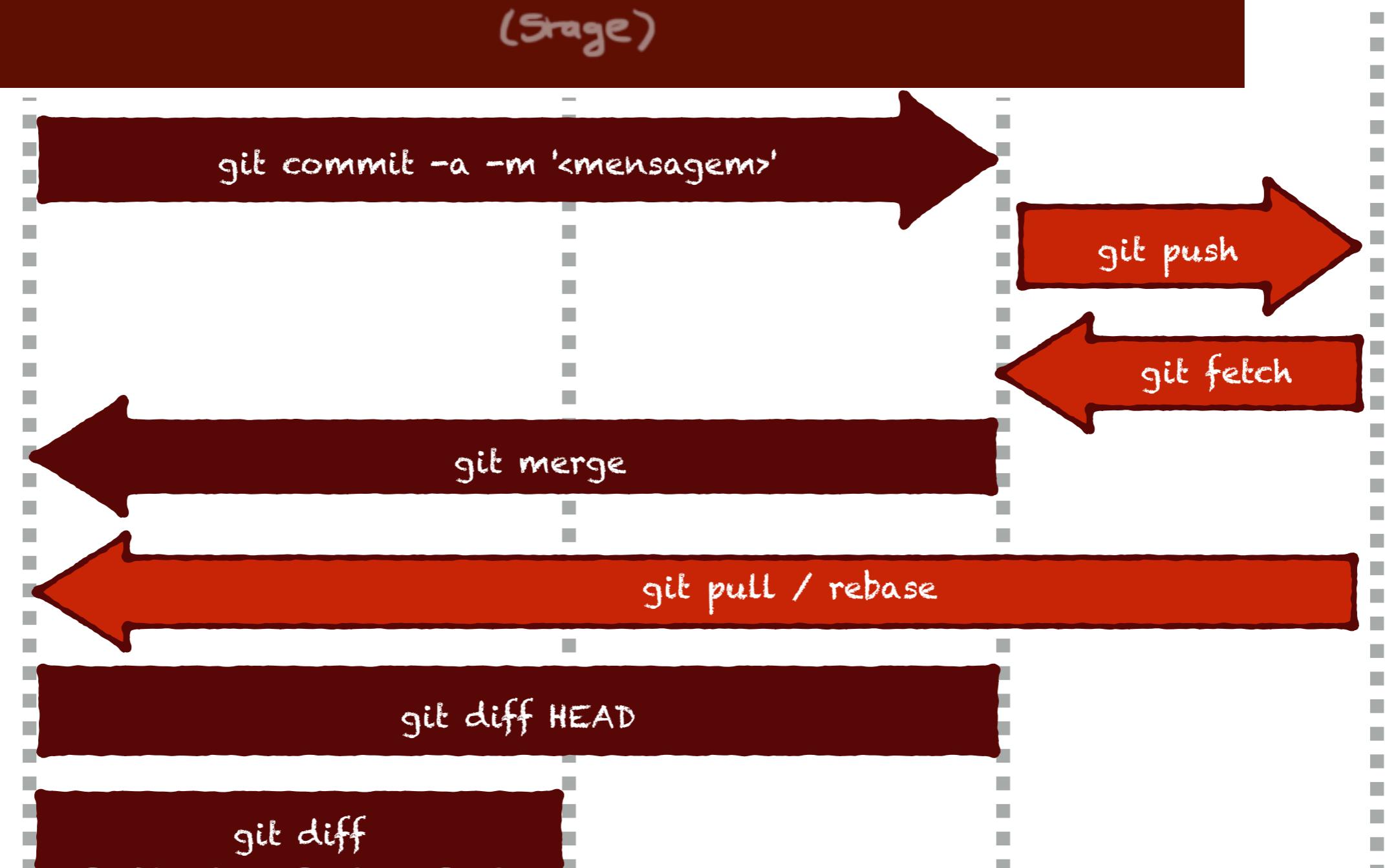


# Git Workflow



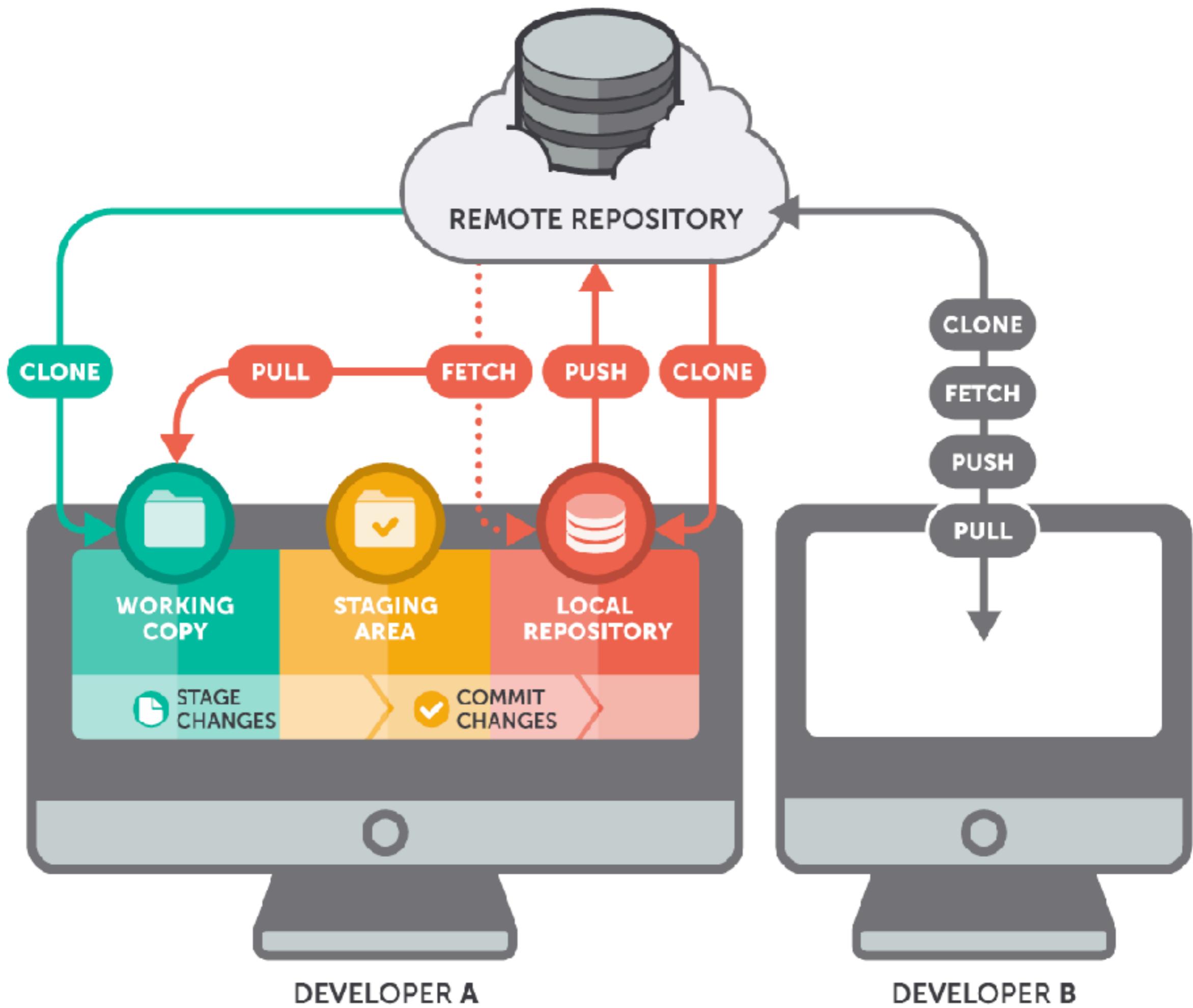
<https://www.youtube.com/watch?v=3a2x1iJFJWc>







Hora de Praticar!!!



# simple daily git workflow



## git pull

pull all the changes from the remote repository

## git checkout -b branch-name-here

create a new branch for your bug/feature/issue

## DO YOUR WORK HERE

keep it in small chunks, the smaller your commits the better, in case things go wrong

### git add .

and any new files you've created

Better: add individual files:  
`git add foo.txt`  
`git add bar.txt`

### git status and/or git diff

see the changes you're going to commit    If already added, use:  
`git diff --cached`

## git commit -m "Detailed message here"

make the commit with a nice detailed message

## git checkout master

Maybe use "git fetch"  
first, to be sure you have  
the latest master

switch back to the master branch when the feature is done, your tests pass right?

## git merge branch-name-here

update the master branch to update the master with all your changes

## git push

send your changes up to the remote repository

feature loop

commit loop

# Git Cheat Sheet

<http://git.or.cz/>

Remember: `git command --help`

Global Git configuration is stored in `$HOME/.gitconfig` (`git config --help`)

## Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

## Show

Files changed in working directory

`git status`

Changes to tracked files

`git diff`

What changed between \$ID1 and \$ID2

`git diff $id1 $id2`

History of changes

`git log`

History of changes for file with diffs

`git log -p $file $dir/ec/tory/`

Who changed what and when in a file

`git blame $file`

A commit identified by \$ID

`git show $id`

A specific file from a specific \$ID

`git show $id:$file`

All local branches

`git branch`

(star \* marks the current branch)

## Concepts

### Git Basics

master : default development branch  
origin : default upstream repository  
HEAD : current branch  
HEAD^ : parent of HEAD  
HEAD~4 : the great-great grandparent of HEAD

### Revert

Return to the last committed state

`git reset --hard`

⚠ you cannot undo a hard reset

Revert the last commit

`git revert HEAD` Creates a new commit

Revert specific commit

`git revert $id` Creates a new commit

Fix the last commit

`git commit -a --amend`

(after editing the broken files)

Checkout the \$id version of a file

`git checkout $id $file`

### Branch

Switch to the \$id branch

`git checkout $id`

Merge branch1 into branch2

`git checkout $branch2`

`git merge branch1`

Create branch named \$branch based on the HEAD

`git branch $branch`

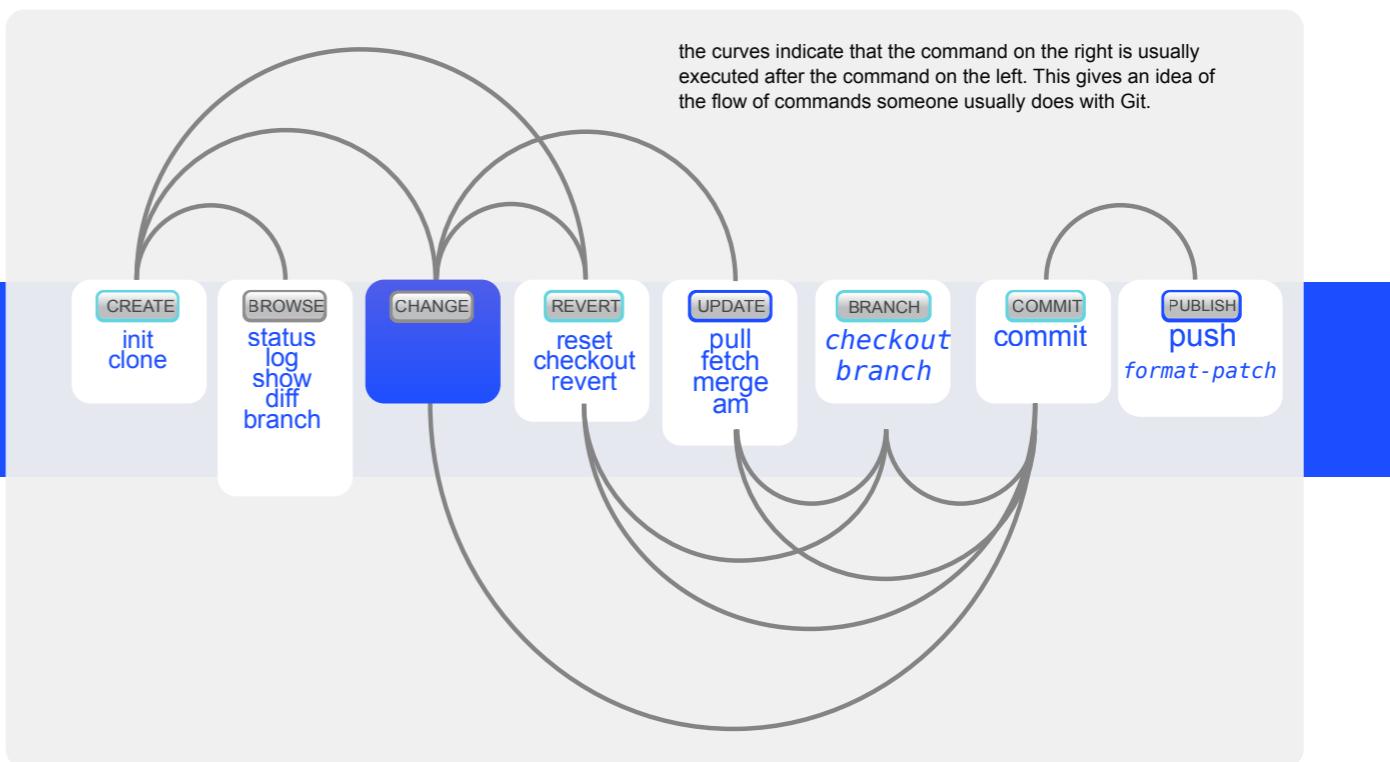
Create branch \$new\_branch based on branch \$other and switch to it

`git checkout -b $new_branch $other`

Delete branch \$branch

`git branch -d $branch`

## Commands Sequence



## Update

Fetch latest changes from origin

`git fetch`

(but this does not merge them).

Pull latest changes from origin

`git pull`

(does a fetch followed by a merge)

Apply a patch that some sent you

`git am -3 patch.mbox`

(in case of a conflict, resolve and use

`git am --resolved`)

## Publish

Commit all your local changes

`git commit -a`

Prepare a patch for other developers

`git format-patch origin`

Push changes to origin

`git push`

Mark a version / milestone

`git tag v1.0`

## Useful Commands

### Finding regressions

`git bisect start`

(to start)  
(\$id is the last working version)

`git bisect good $id`  
(\$id is a broken version)

`git bisect bad/good`

(to mark it as bad or good)

`git bisect visualize`  
(to launch gitk and mark it)

`git bisect reset`  
(once you're done)

Check for errors and cleanup repository

`git fsck`

`git gc --prune`

Search working directory for foo()

`git grep "foo()"`

## Resolve Merge Conflicts

### To view the merge conflicts

`git diff` (complete conflict diff)

(against base file)

`git diff --base $file` (against your changes)

(against other changes)

### To discard conflicting patch

`git reset --hard`

`git rebase --skip`

### After resolving conflicts, merge with

`git add $conflicting_file`

(do for all resolved files)

## Cheat Sheet Notation

\$id : notation used in this sheet to represent either a commit id, branch or a tag name

\$file : arbitrary file name

\$branch : arbitrary branch name



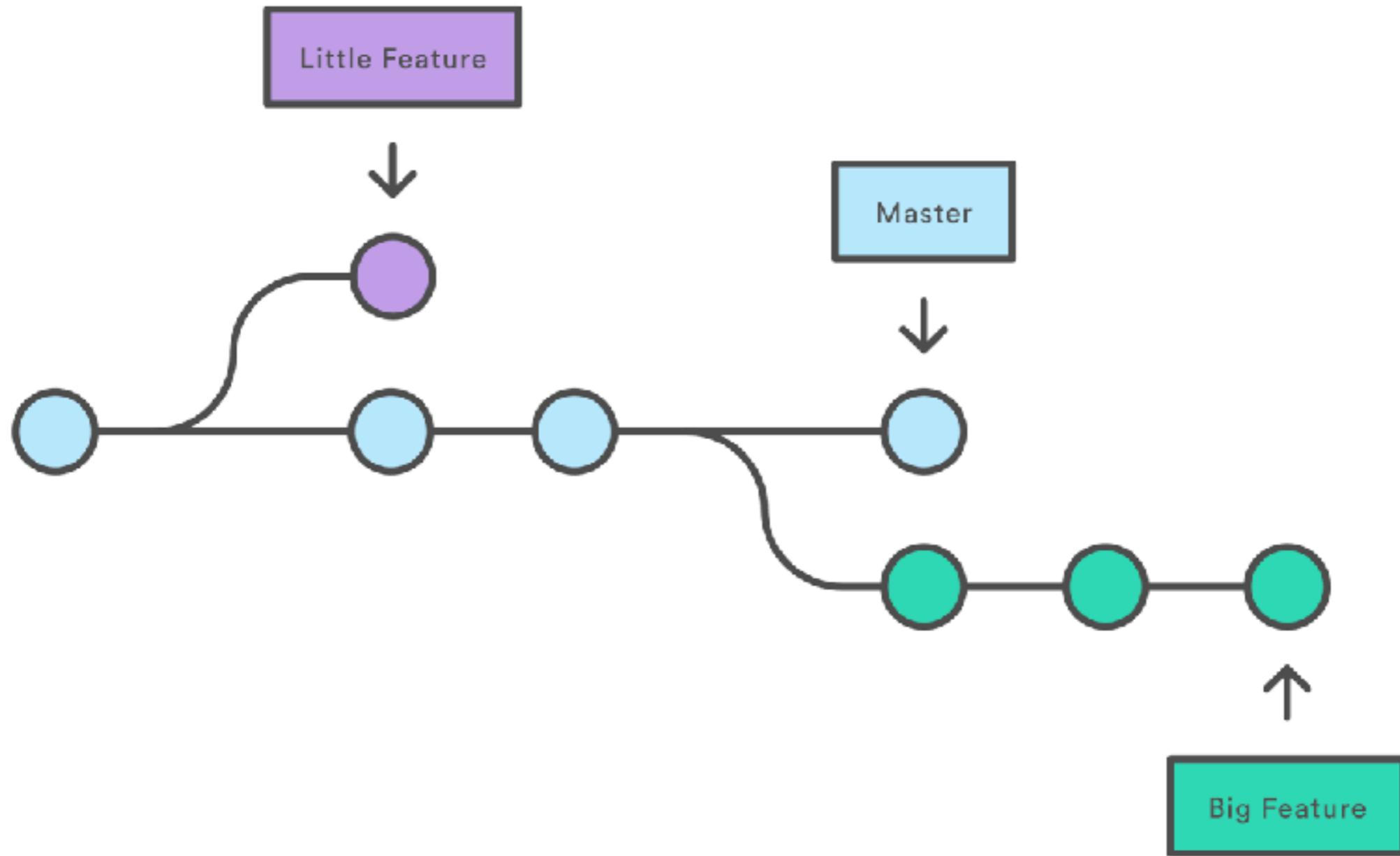
# Comandos Básicos

- git config
- git help
- git init
- git remote
- git clone
- ~~git status~~
- ~~git add~~
- ~~git commit~~
- ~~git checkout~~
- ~~git reset~~
- ~~git diff~~
- ~~git pull~~
- ~~git push~~
- ~~git log (pretty e graph)~~
- git tag

# Dúvidas

- O que já viram?
- O que deu pra experimentar?
- Como foi a experiência até aqui?





# Branches

git checkout / git branch / git merge

# Criação de Branches

- `git branch <nome da branch>`
- `git branch <nome da branch> <branch de referência>`
- `git checkout -b <nome da branch>`
- **Branch Remota**
  - `git push origin <nome da branch>`

# Removendo Branches

- `git branch -d <nome da branch>`
- `git branch -D <nome da branch>`
- **Branches remotas**
  - `git push <referência remota> :<nome da branch>`

# Navegando entre as Branches

- `git checkout <nome da branch>`
- Dica:
  - o *git checkout* também pode ser usado para criação de branches com o parâmetro `-b`
  - `git checkout -b <nome da branch>`

# Merge

- git merge <nome da branch>
  - seleciona o algoritmo de merge automaticamente

--ff

When the merge resolves as a fast-forward, only update the branch pointer, without creating a merge commit. This is the default behavior.

--no-ff

Create a merge commit even when the merge resolves as a fast-forward. This is the default behaviour when merging an annotated (and possibly signed) tag.

--ff-only

Refuse to merge and exit with a non-zero status unless the current HEAD is already up-to-date or the merge can be resolved as a fast-forward.

# simple daily git workflow



## git pull

pull all the changes from the remote repository

## git checkout -b branch-name-here

create a new branch for your bug/feature/issue

## DO YOUR WORK HERE

keep it in small chunks, the smaller your commits the better, in case things go wrong

### git add .

and any new files you've created

Better: add individual files:  
`git add foo.txt`  
`git add bar.txt`

### git status and/or git diff

see the changes you're going to commit    If already added, use:  
`git diff --cached`

## git commit -m "Detailed message here"

make the commit with a nice detailed message

## git checkout master

Maybe use "git fetch"  
first, to be sure you have  
the latest master

switch back to the master branch when the feature is done, your tests pass right?

## git merge branch-name-here

update the master branch to update the master with all your changes

## git push

send your changes up to the remote repository

feature loop

commit loop

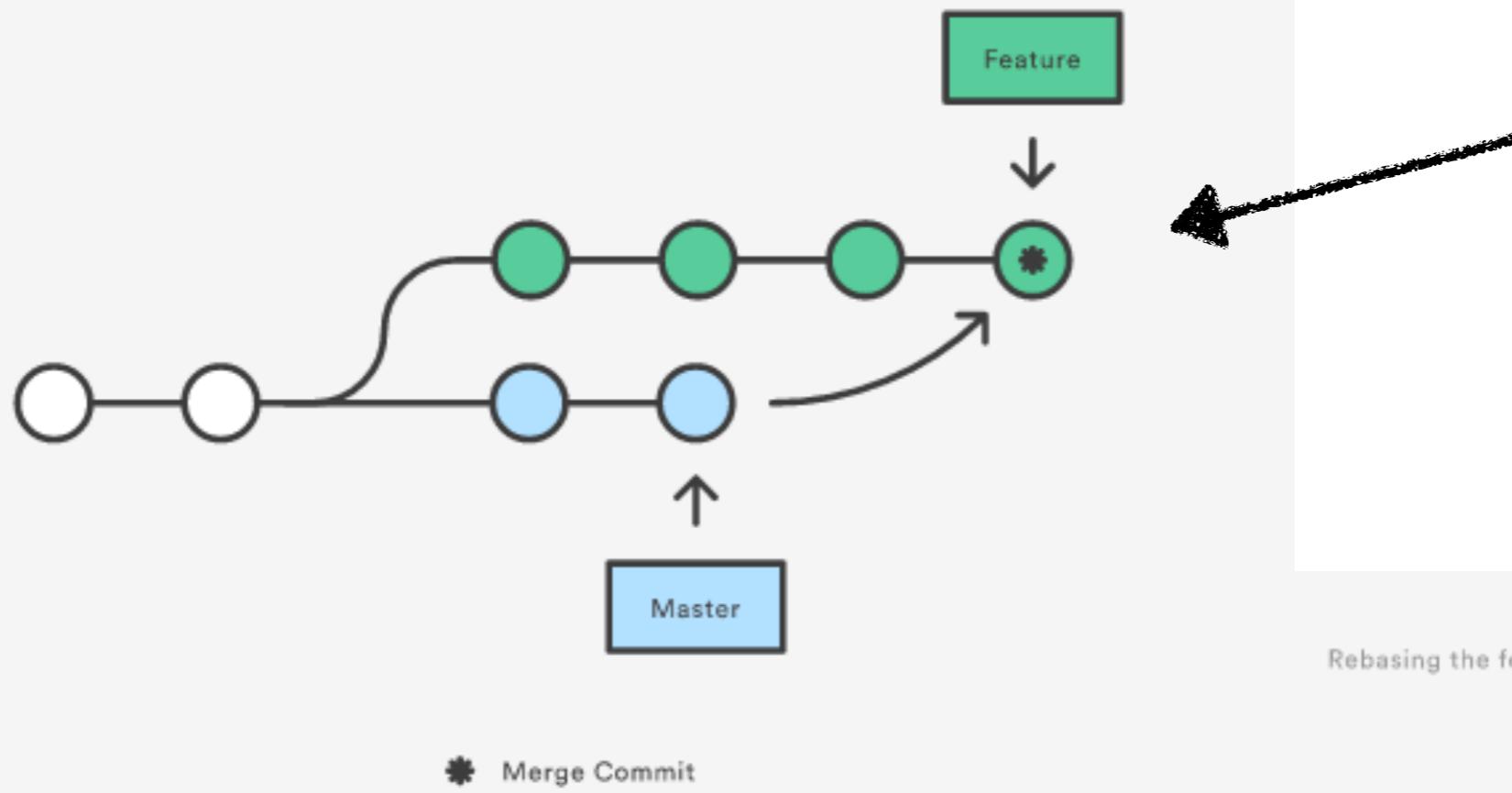


# Merge vs Rebase

<https://www.atlassian.com/git/tutorials/merging-vs-rebasing>

**Dica:** <https://tasafotutorial.org/2015/12/03/git-merge-e-rebase/>

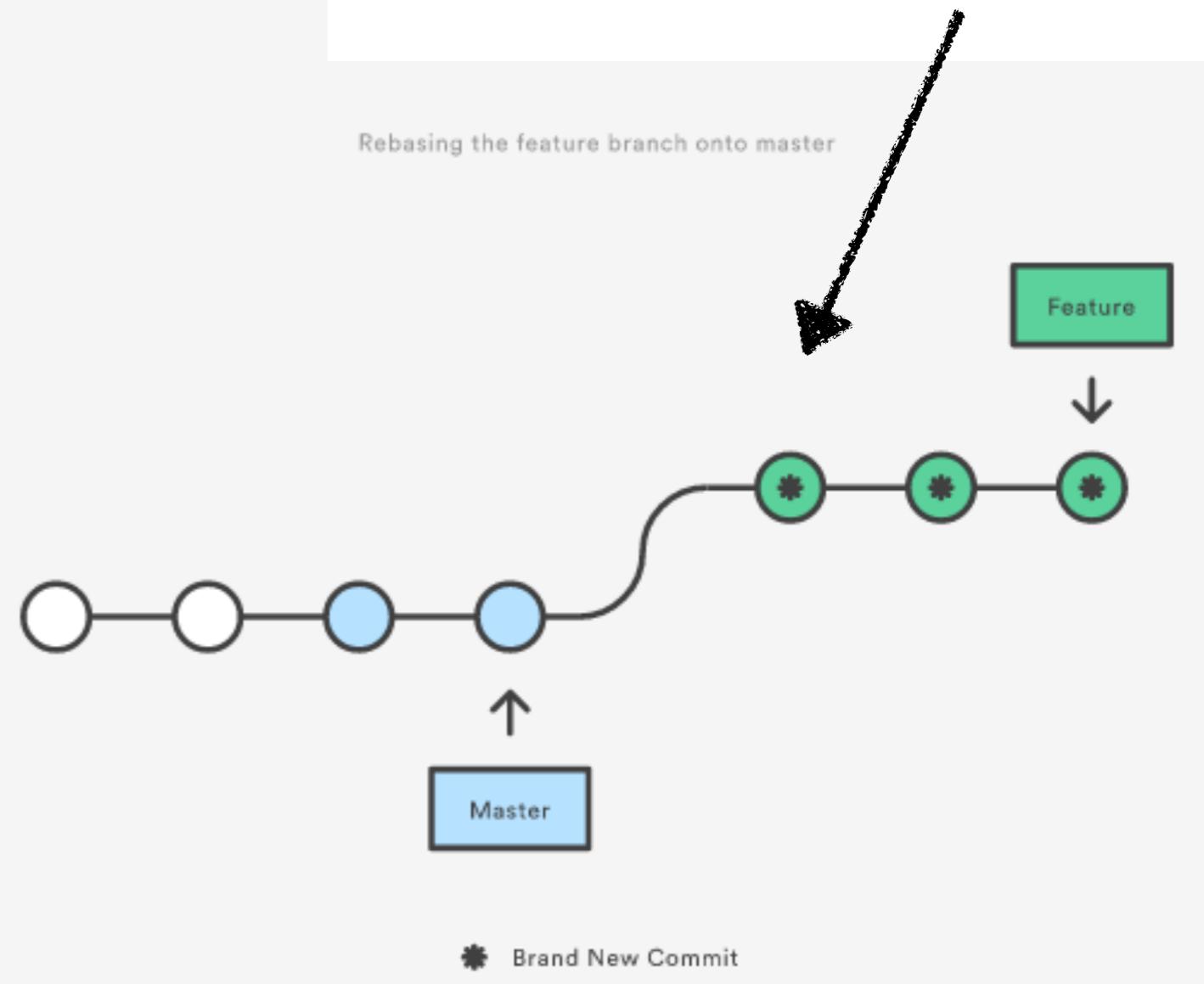
Merging master into the feature branch



**merging**

**rebasing**

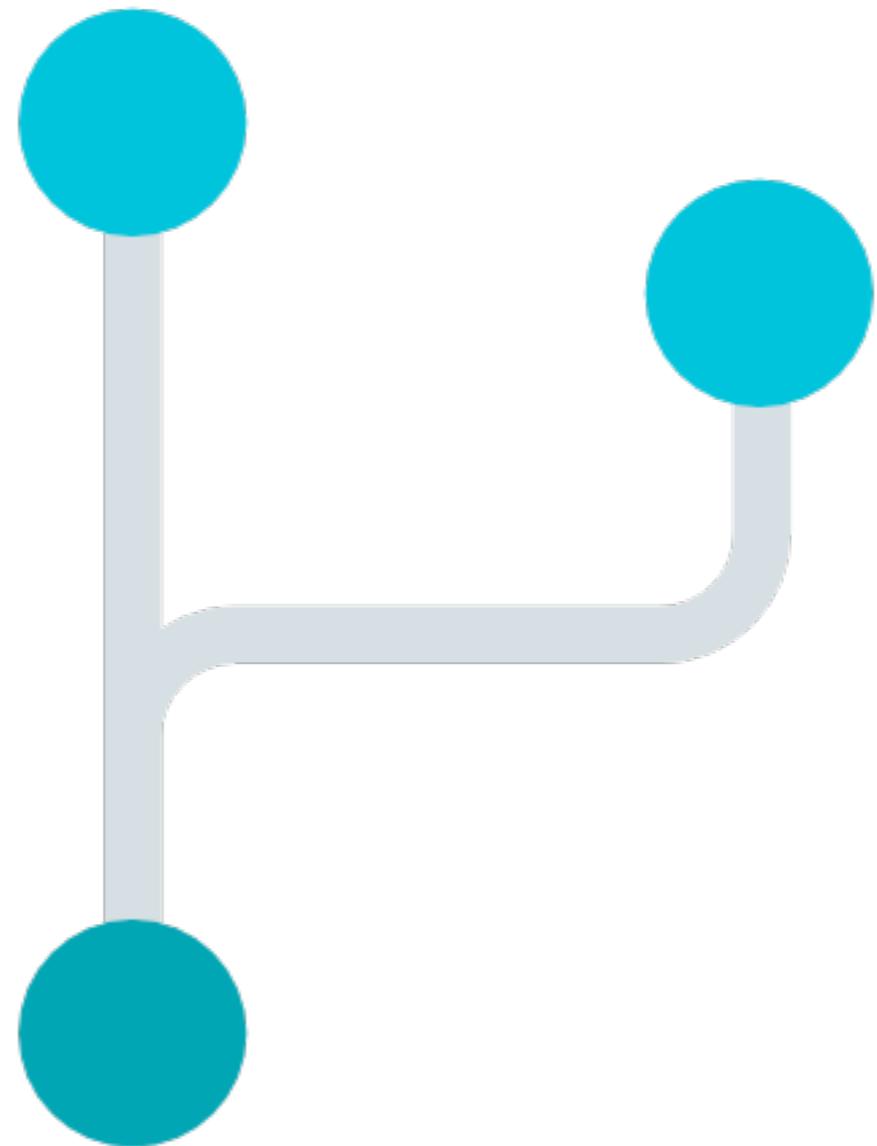
Rebasing the feature branch onto master



## Merging vs. Rebasing

<https://www.atlassian.com/git/tutorials/merging-vs-rebasing>

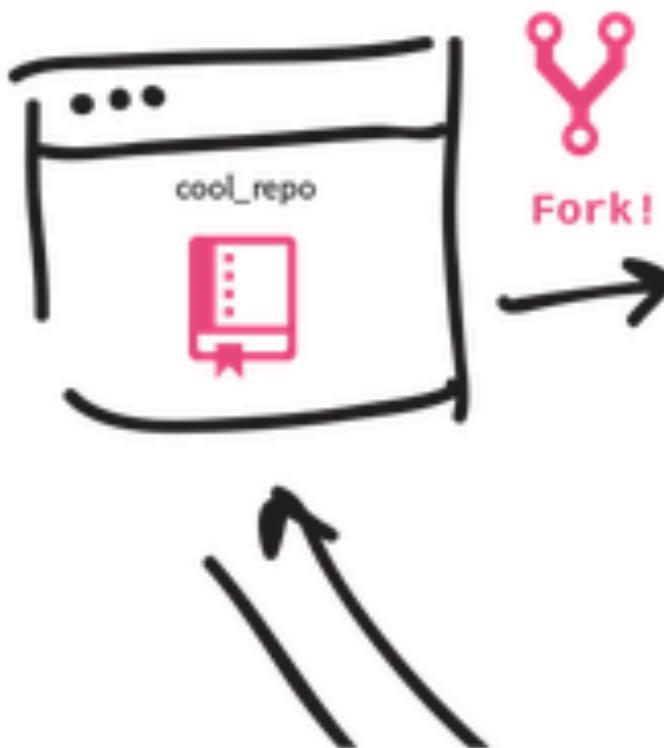
\* Brand New Commit



Fork

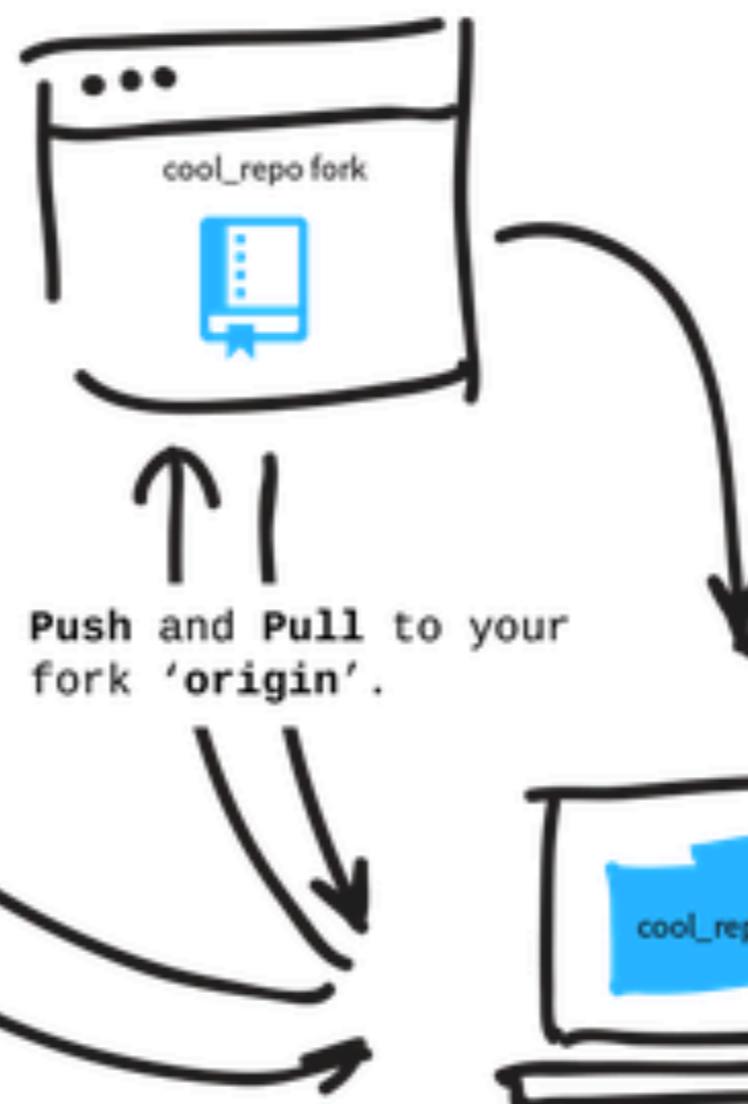
## REMOTE

Someone else's repository.



## REMOTE

Your fork of the repository.



## LOCAL

Use your computer's terminal to talk to two repositories via two remotes to the GitHub servers.

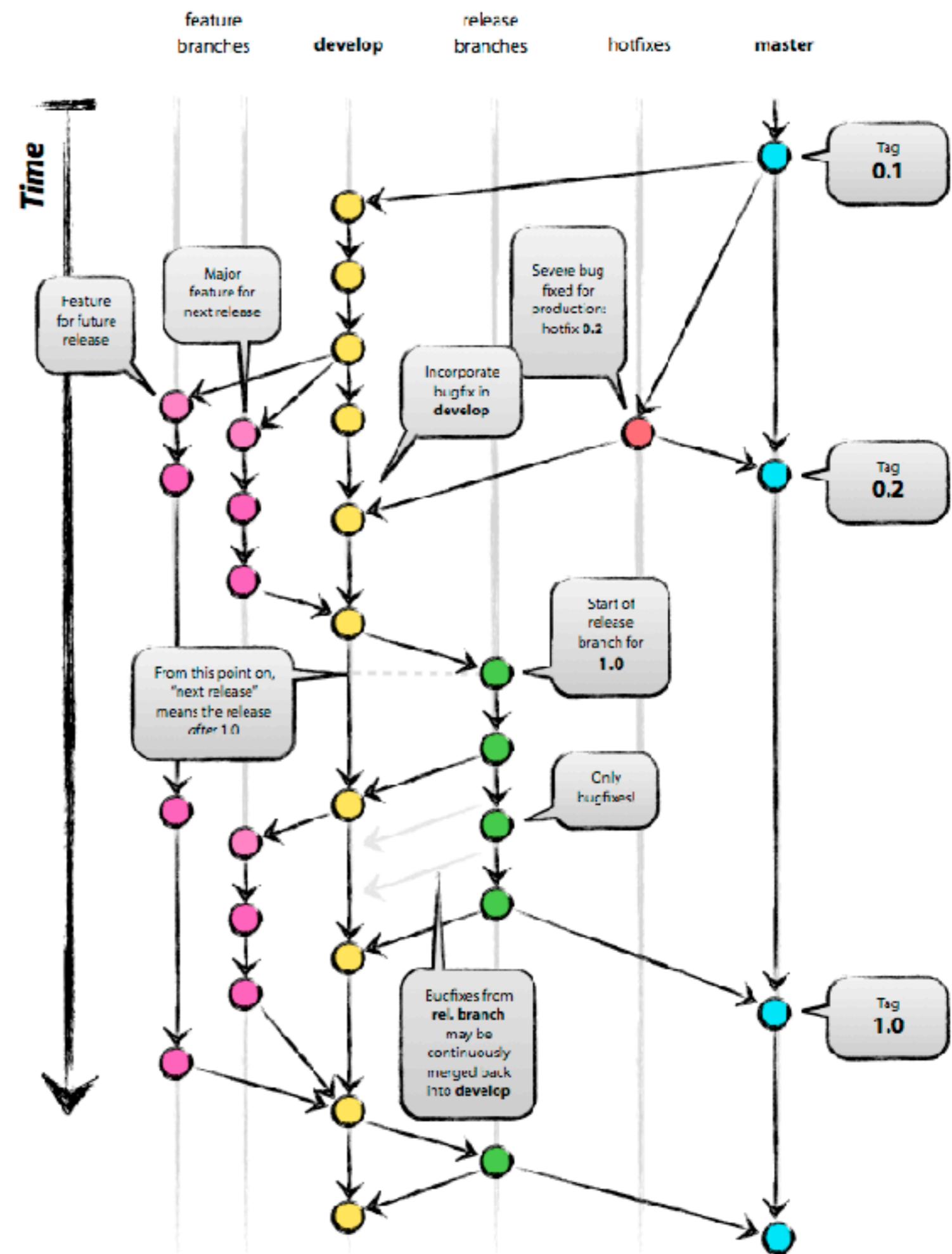


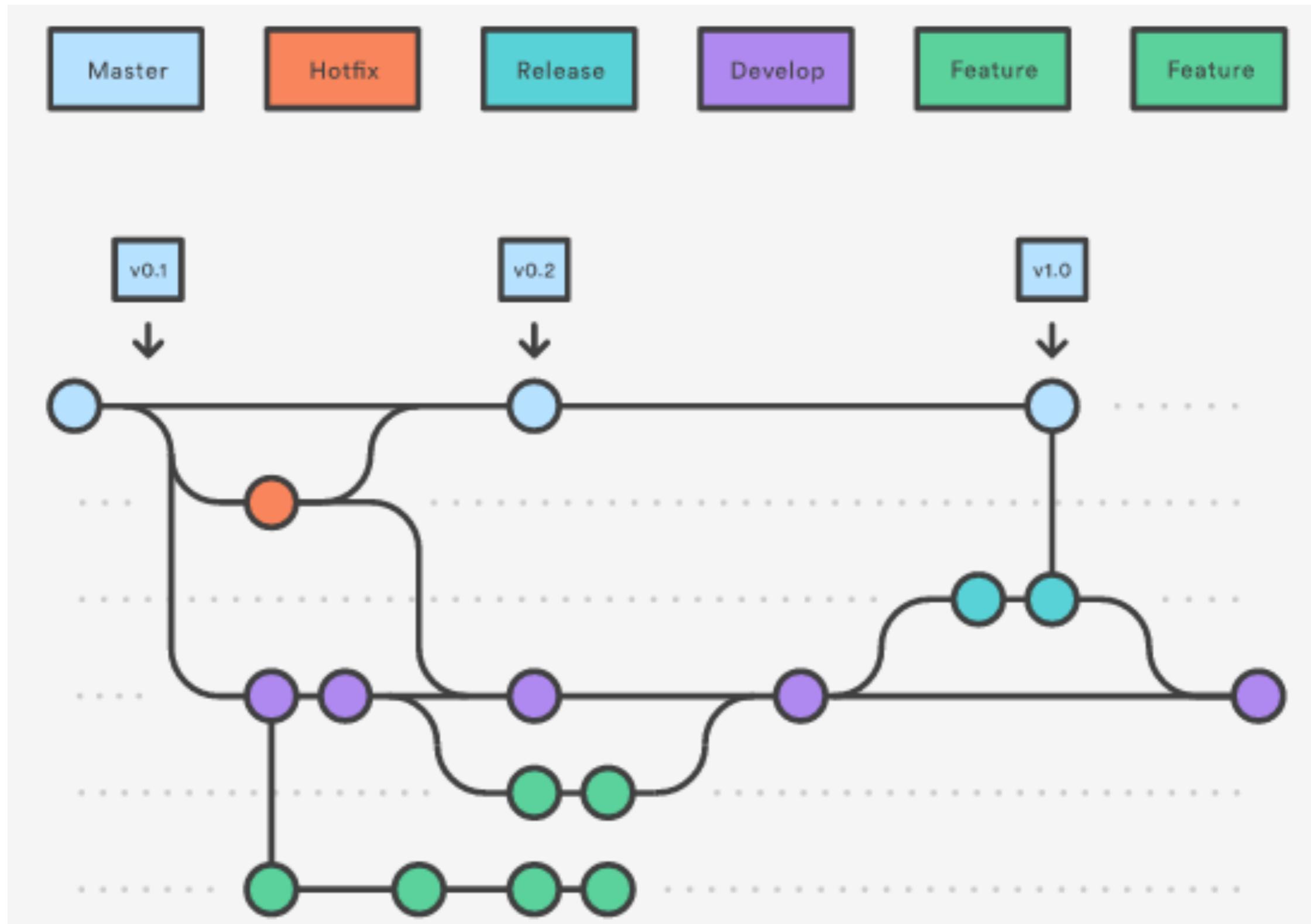
# Github / Gitlab / Bitbucket



# Git Flow

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





<https://www.atlassian.com/git/tutorials/comparing-workflows>



# Comandos Úteis

- git stash
- git cherry-pick
- git fetch -p
- git merge --squash
- git commit --amend
- git rev-list --no-merges branch..branch

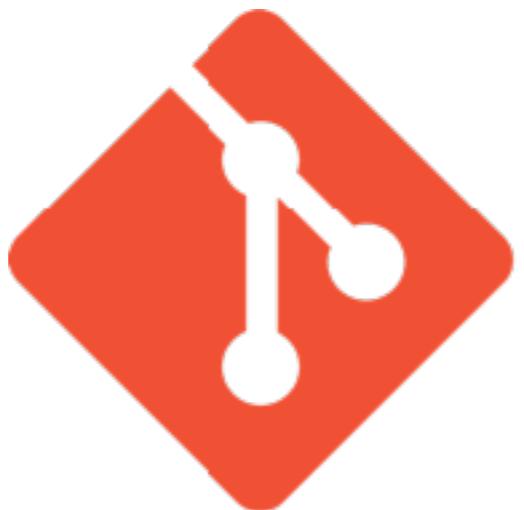
# Dúvidas

- O que já viram?
- O que deu pra experimentar?
- Como foi a experiência até aqui?





**YOUR  
FEEDBACK  
MATTERS!**



git

# Workshop GIT

Aprenda Git de uma vez por todas!

**OBRIGADO!!!!**

**Paulo Igor**