Workshop GIT

Aprenda Git de uma vez por todas!

Paulo Igor

Paulo Igor

pigor@idopterlabs.com.br

@pigodinho

Apresentações

Inscrição, Emails, Processo, Expectativas, ….

Dúvidas

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

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!

https://insights.stackoverflow.com/survey/2017#work-version-control

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

http://pt.slideshare.net/JessCiriloSoares/git-vs-svn-31692526

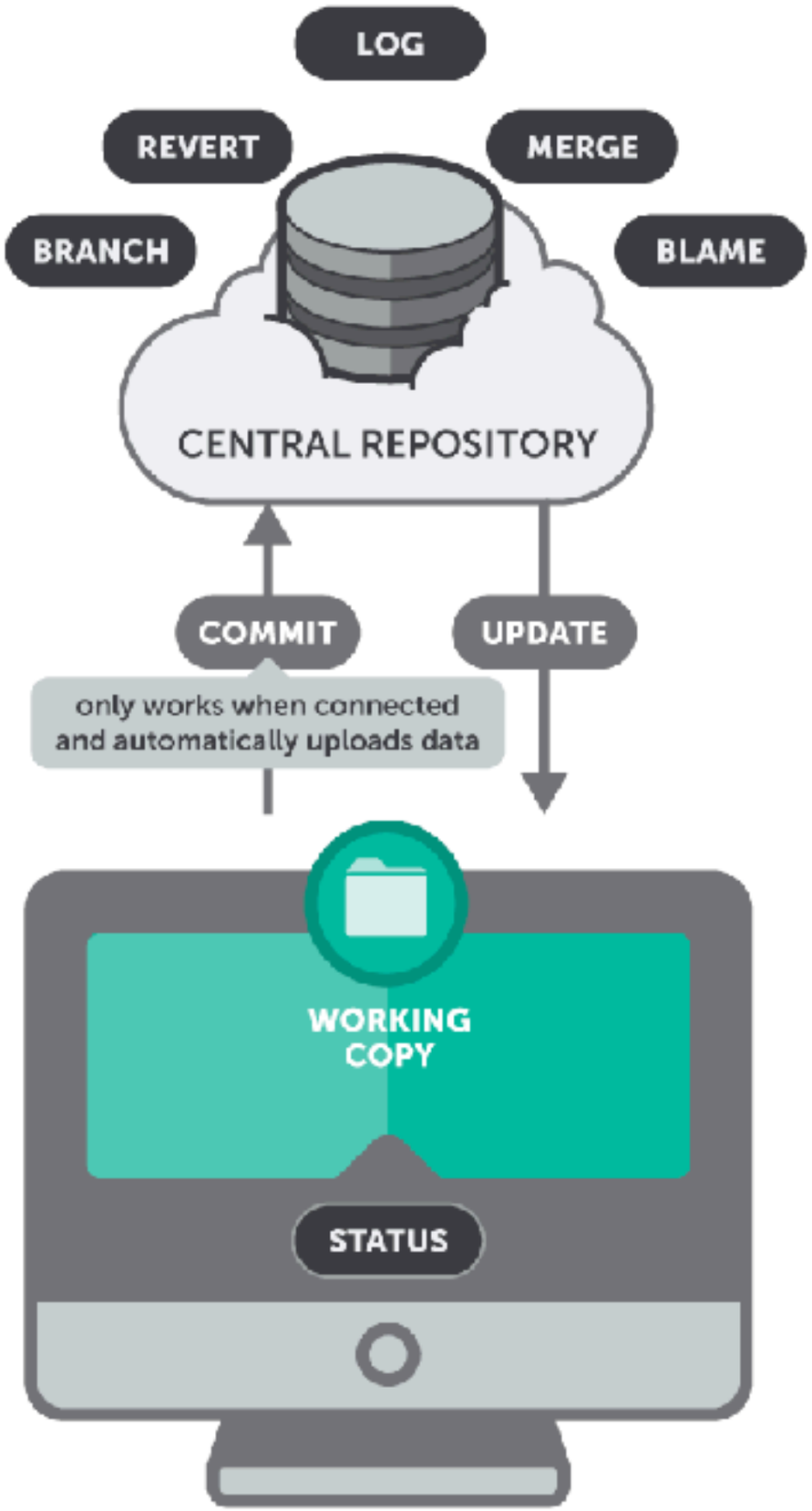
Linux Kernel Development Visualization (git commit history - past 6 weeks - june 02 2012)

https://www.youtube.com/watch?v=P_02QGsHzEQ

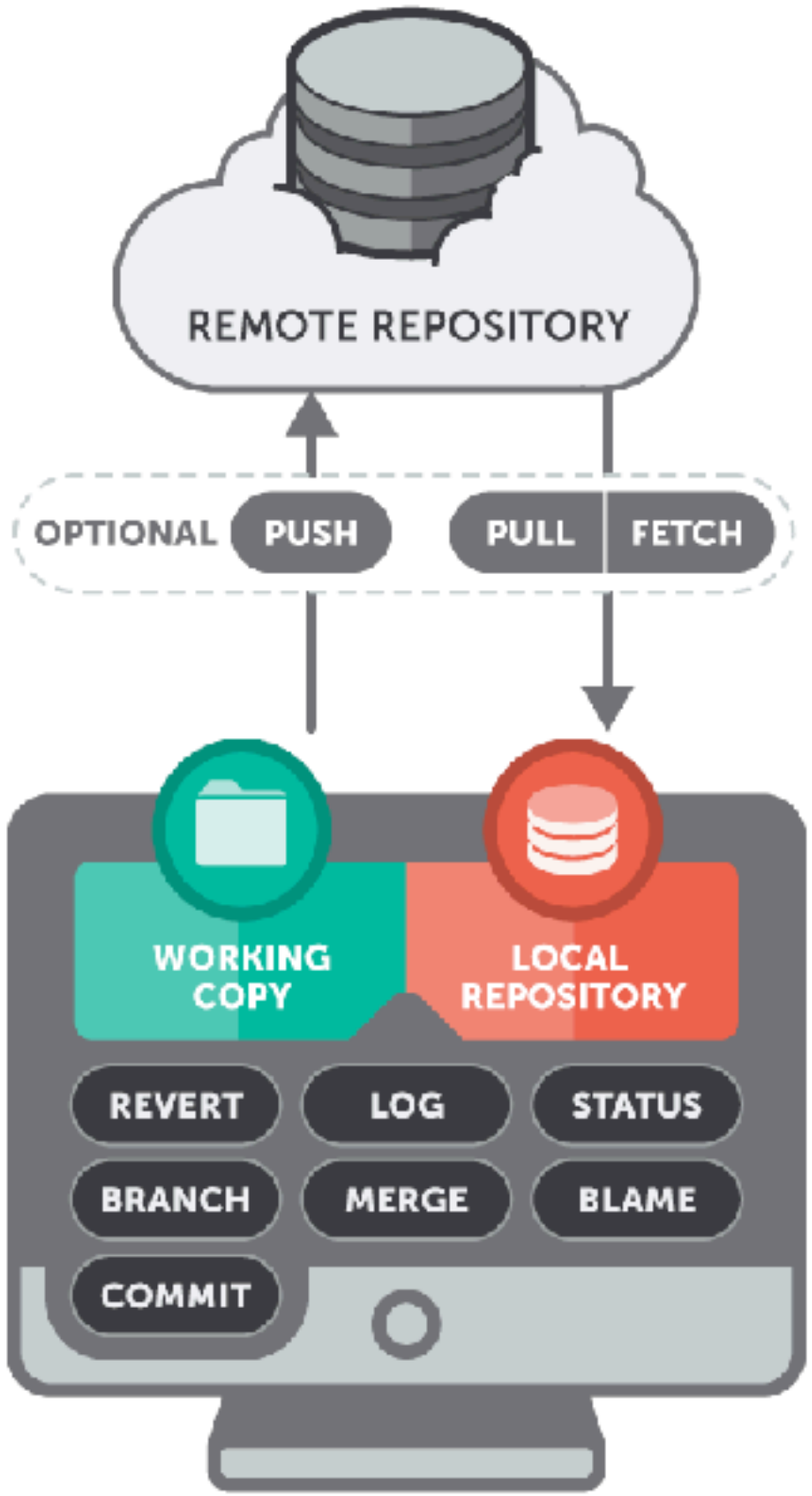
Modelos de Versionamento

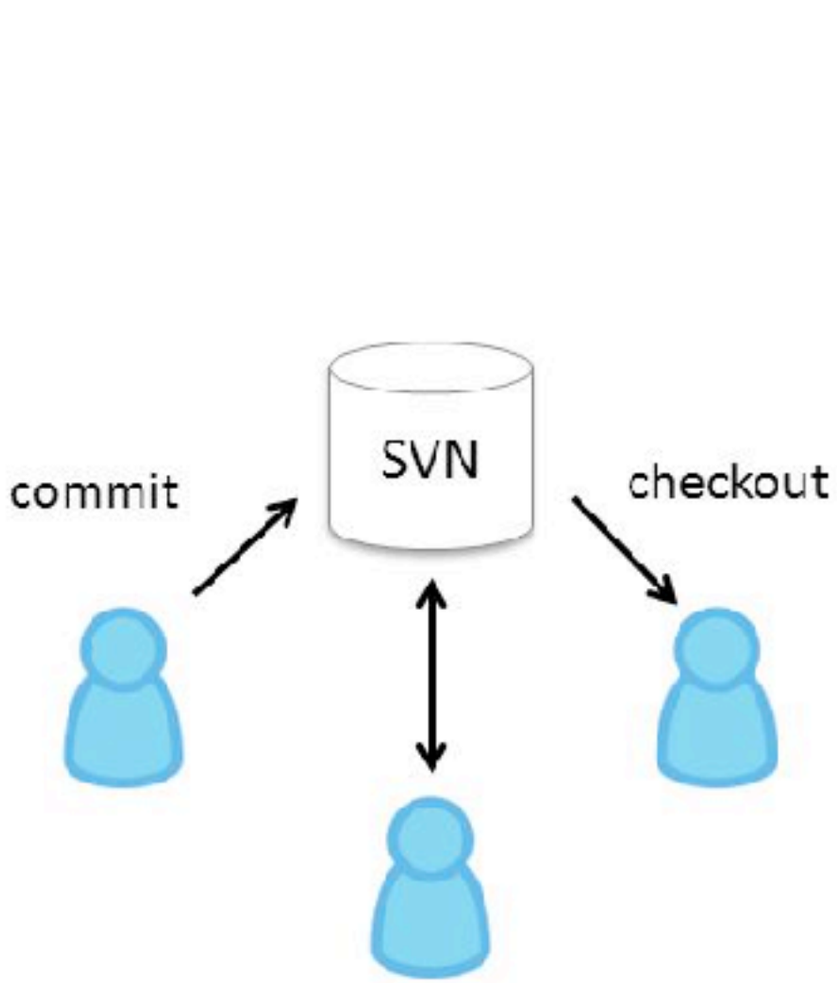
Centralizado vs Descentralizado

SUBVERSION

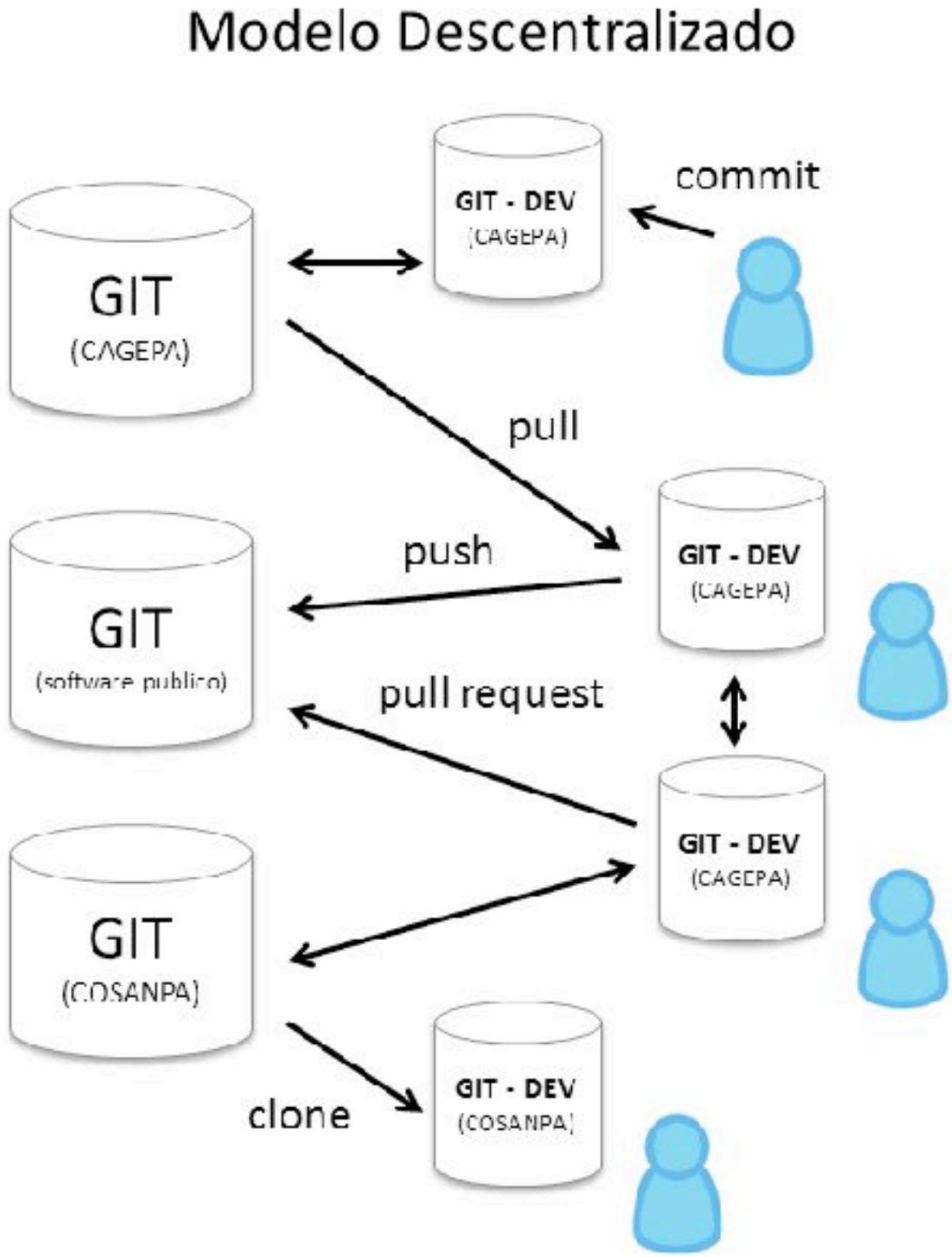


GIT

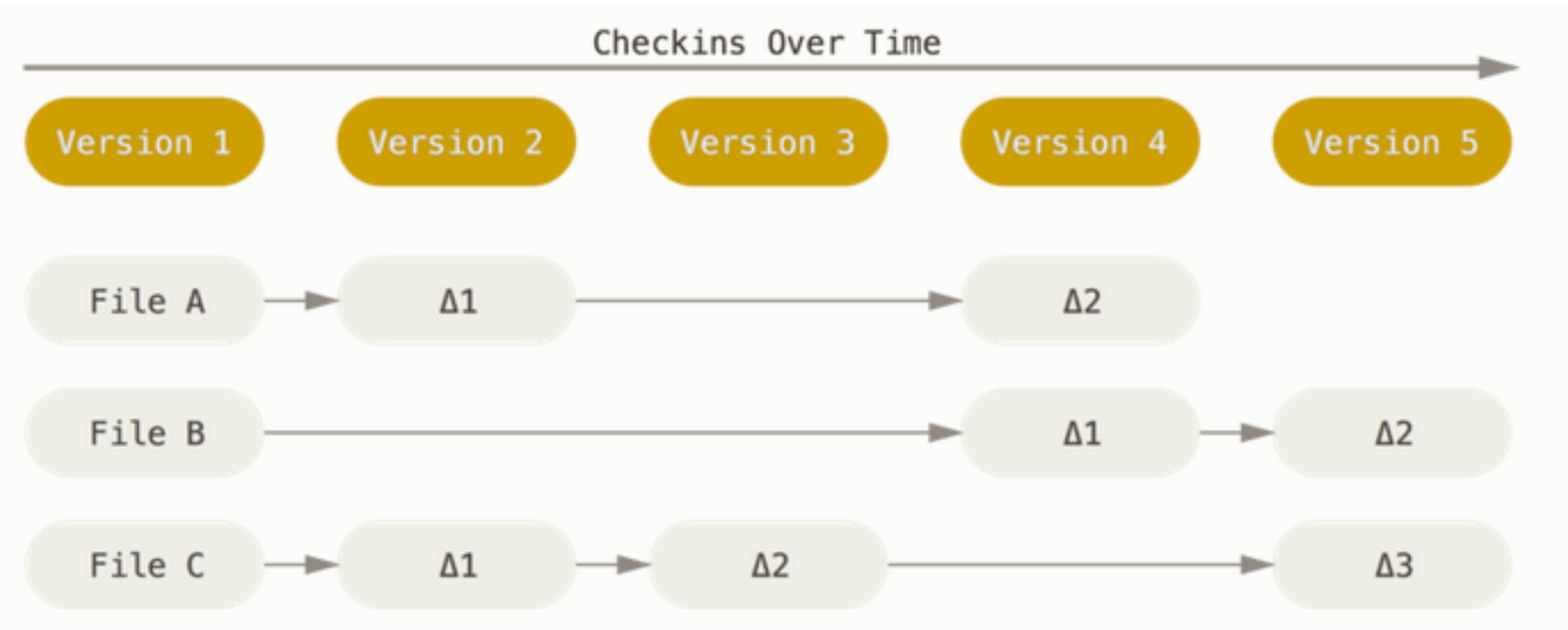


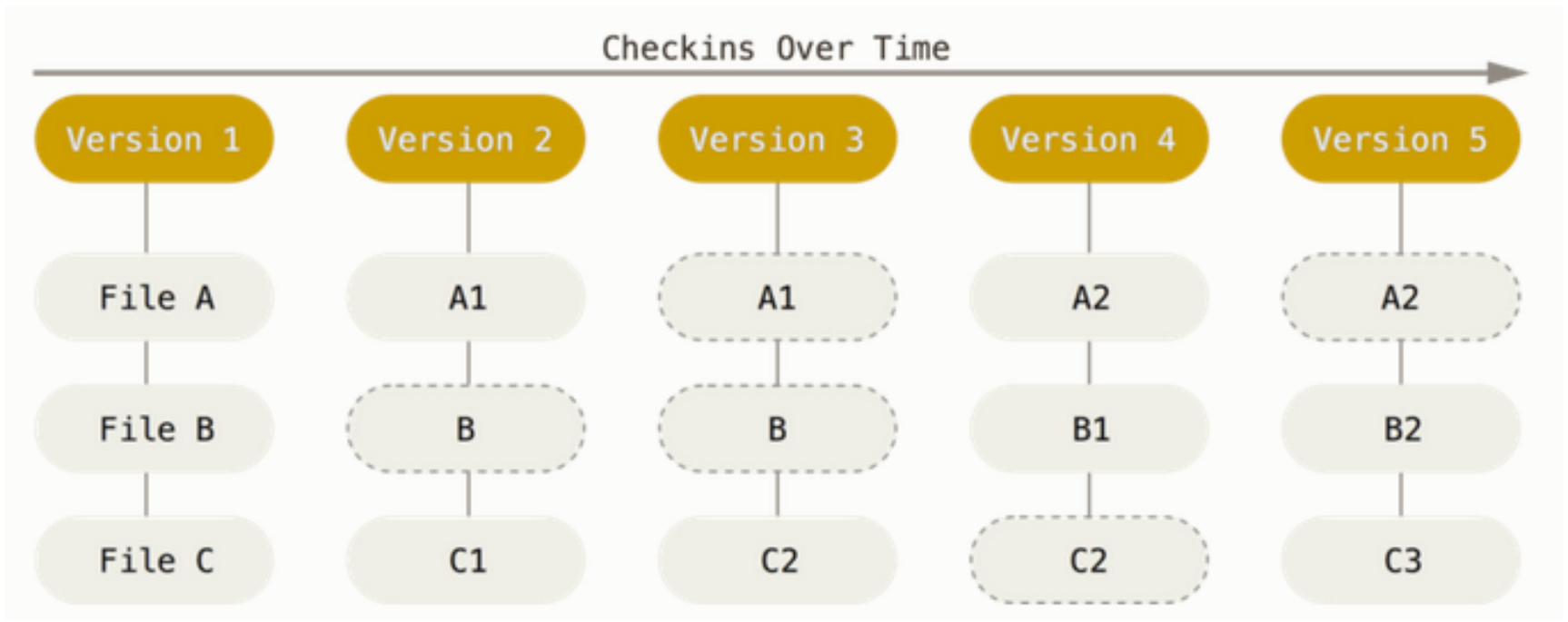


Modelo Centralizado

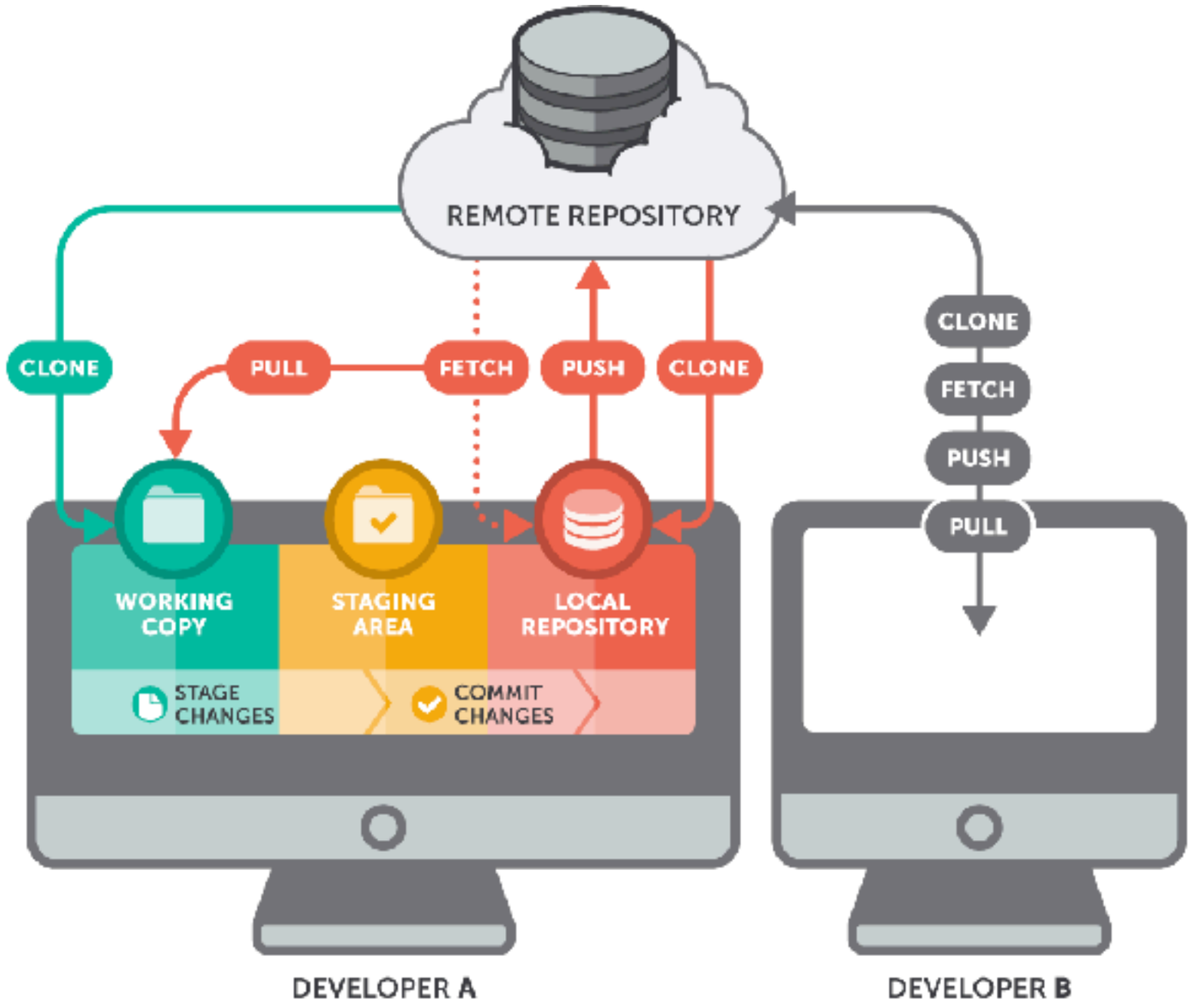


http://slideplayer.com.br/slide/6138625/

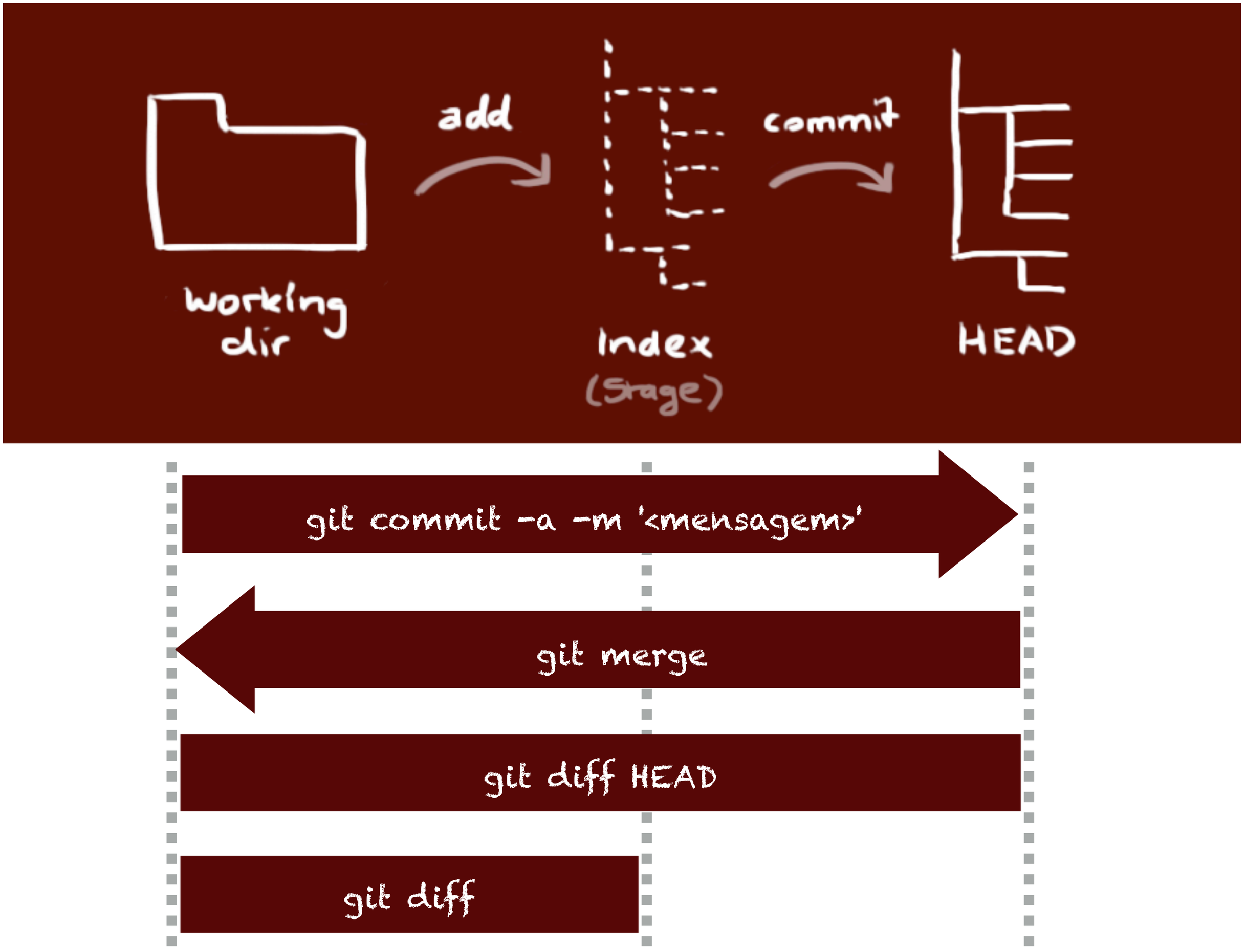

Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
Δ1
Δ2
File B
Δ1
Δ2
File C
Δ1
Δ2
Δ3

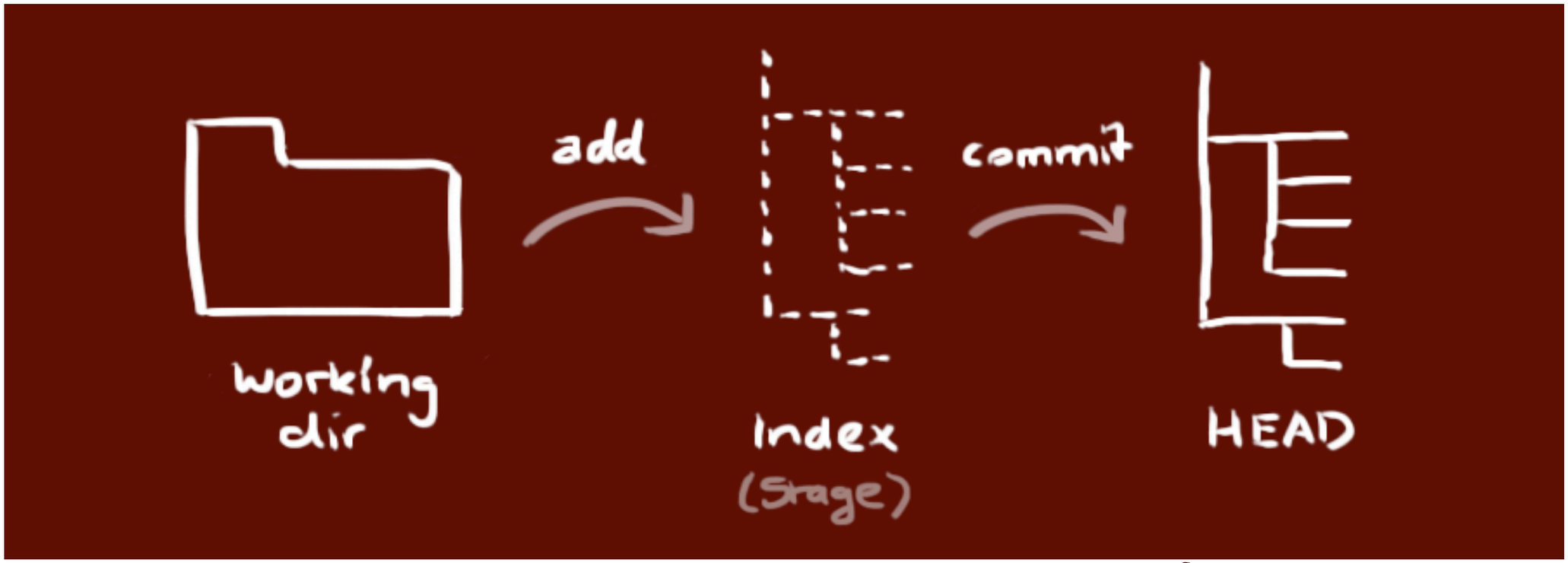

Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
A1
A1
A2
A2
File B
B
B
B1
B2
File C
C1
C2
C2
C3

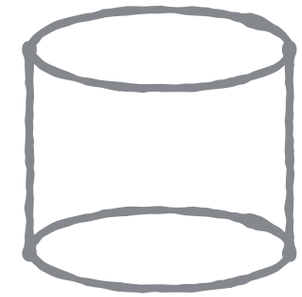
Git Workflow



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


add
commit
working dir
Index
(Stage)
HEAD
git commit -a -m '<mensagem>'
git merge
git diff HEAD
git diff





remote repository

git commit -a -m '<mensagem>'

git push

git fetch

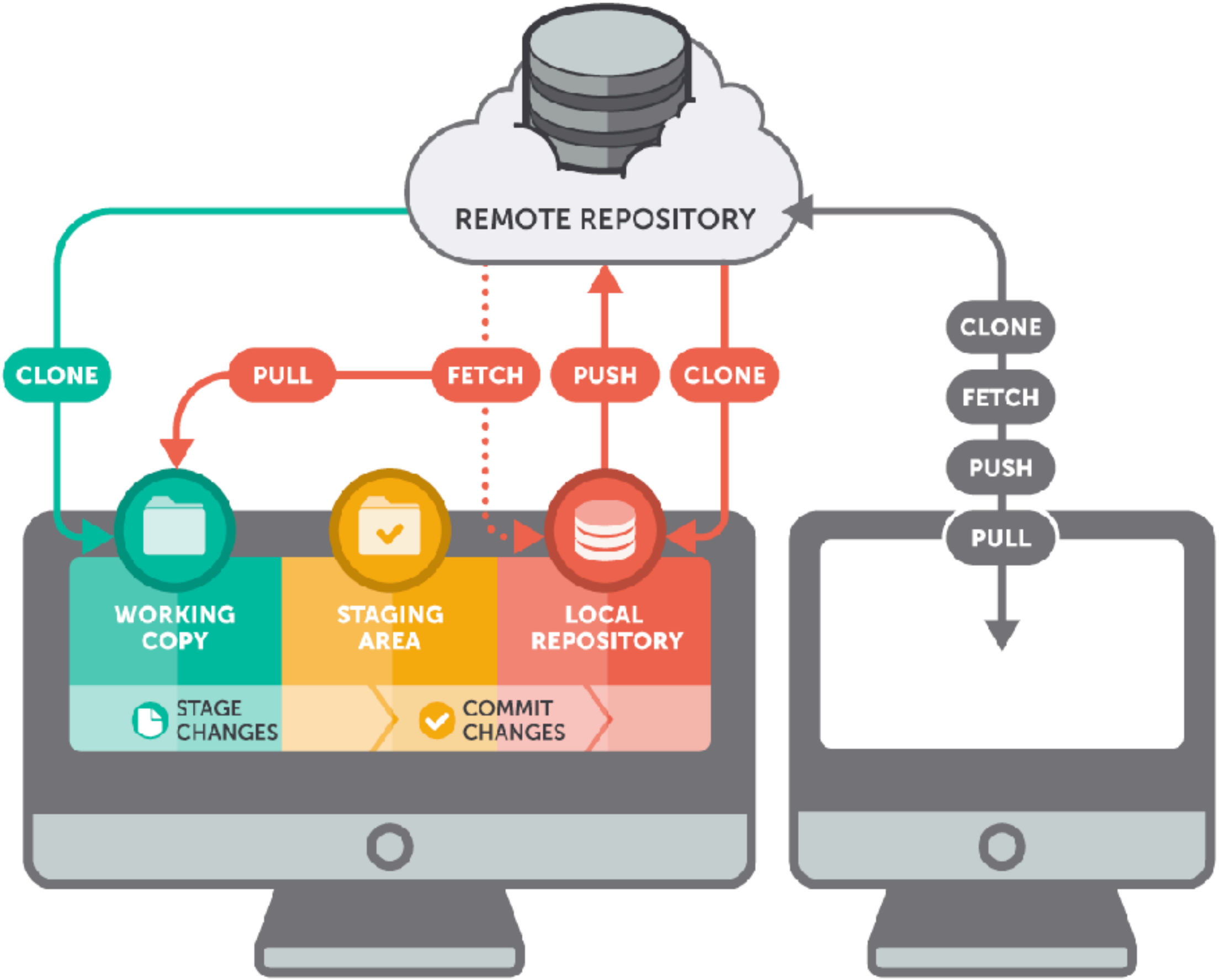
git merge

git pull / rebase

git diff HEAD

git diff

Hora de Praticar!!!


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

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 .

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

and any new files you ve created

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

commit loop

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?

feature loop

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

Git Cheat Sheet

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

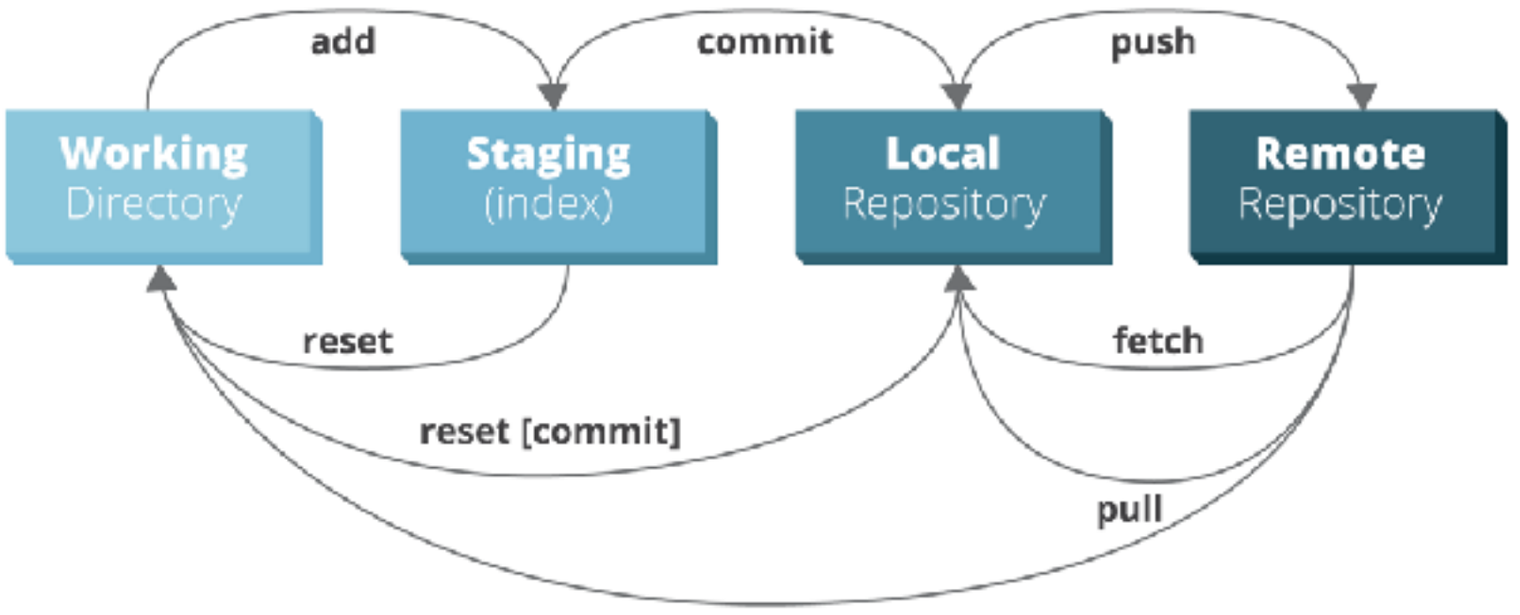
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



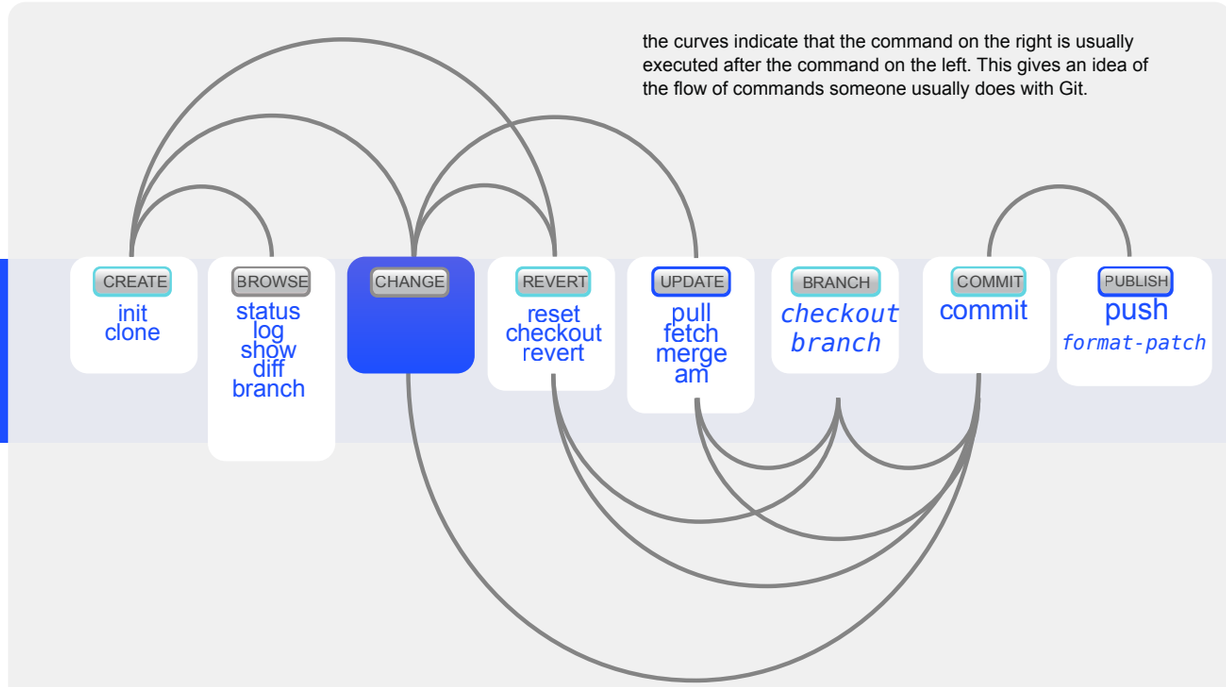
Git Cheat Sheet

http://git.or.cz/

Remember: git command --help

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

Commands Sequence



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)

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

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
```

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)
git bisect good $id       ($id is the last working version)
git bisect bad $id        ($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 conclicts

```
git diff              (complete conflict diff)
git diff --base  $file    (against base file)
git diff --ours  $file    (against your changes)
git diff --theirs $file   (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)
git rebase --continue
```

Zack Rusin
Based on the work of:
Sébastien Pierre
Xprima Corp.

Comandos Básicos

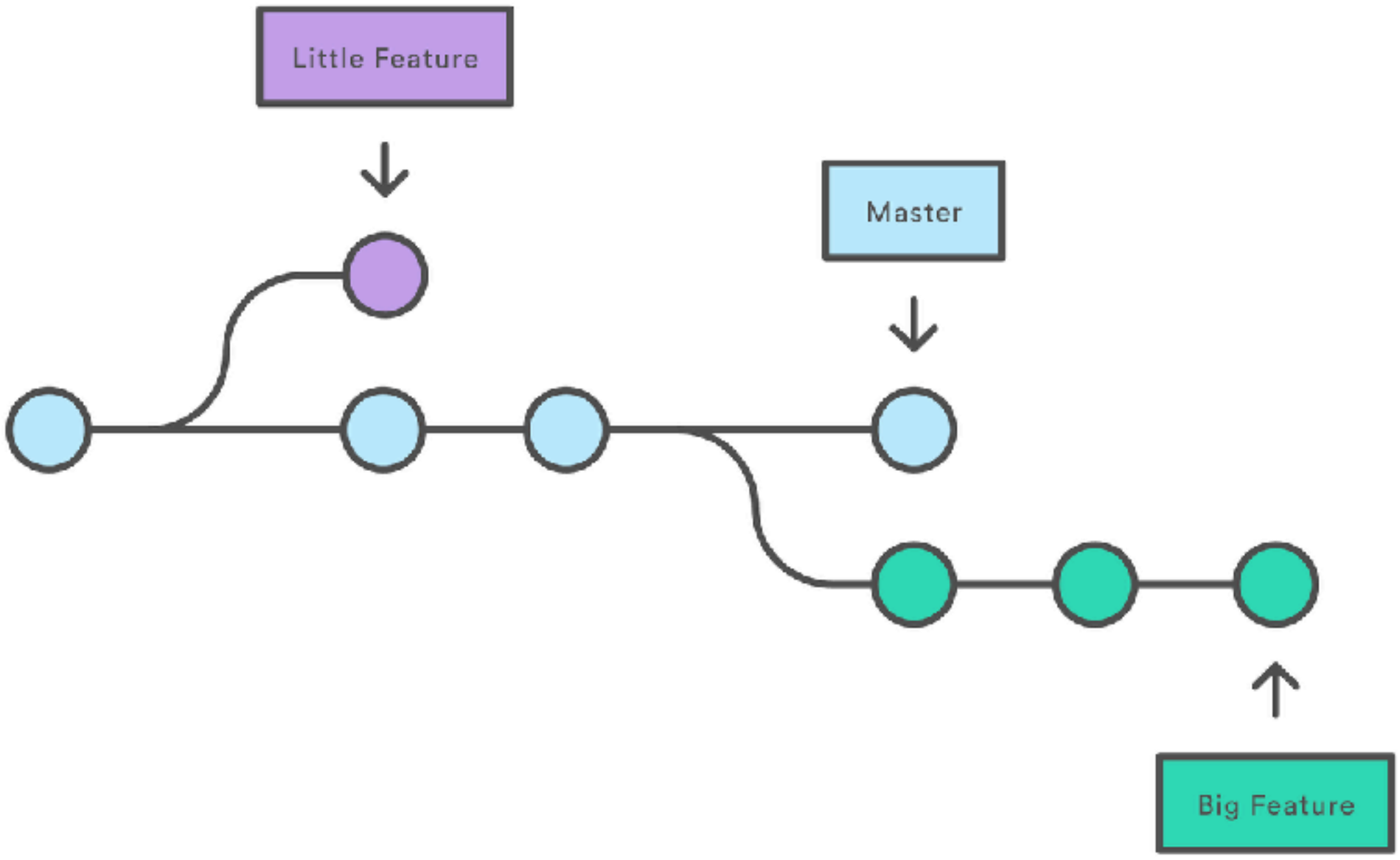
- git config
- git help
- git init
- git remote
- git clone
- git status
- git add
- git commit
- git checkout
- git reset
- git diff
- git pull
- git push
- git log (pretty e graph)
- git tag

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 .

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

and any new files you ve created

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

commit loop

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?

feature loop

git merge branch-name-here

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

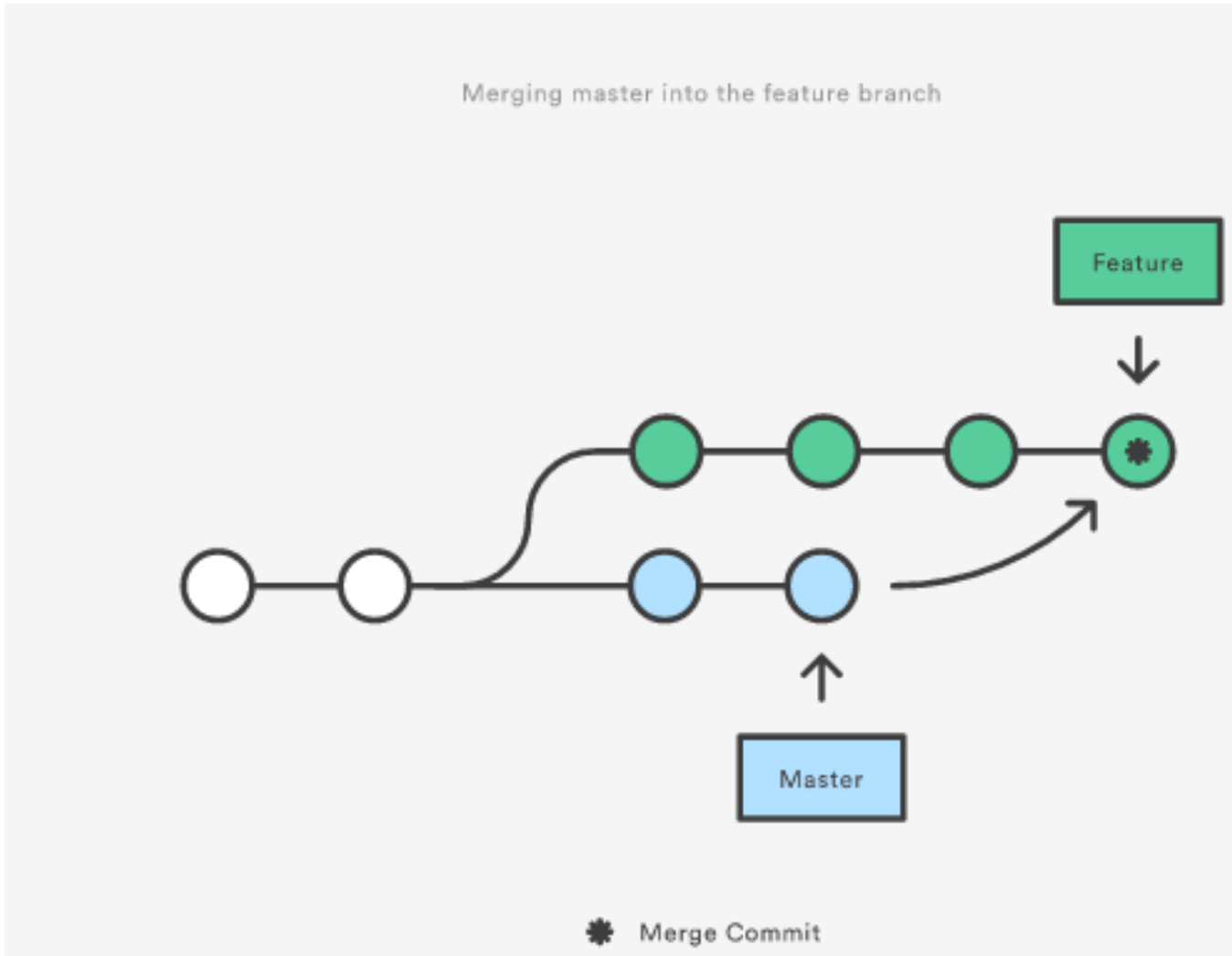
git push

send your changes up to the remote repository

Merge vs Rebase

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

Dica: https://tasafo.org/2015/12/03/git-merge-e-rebase/

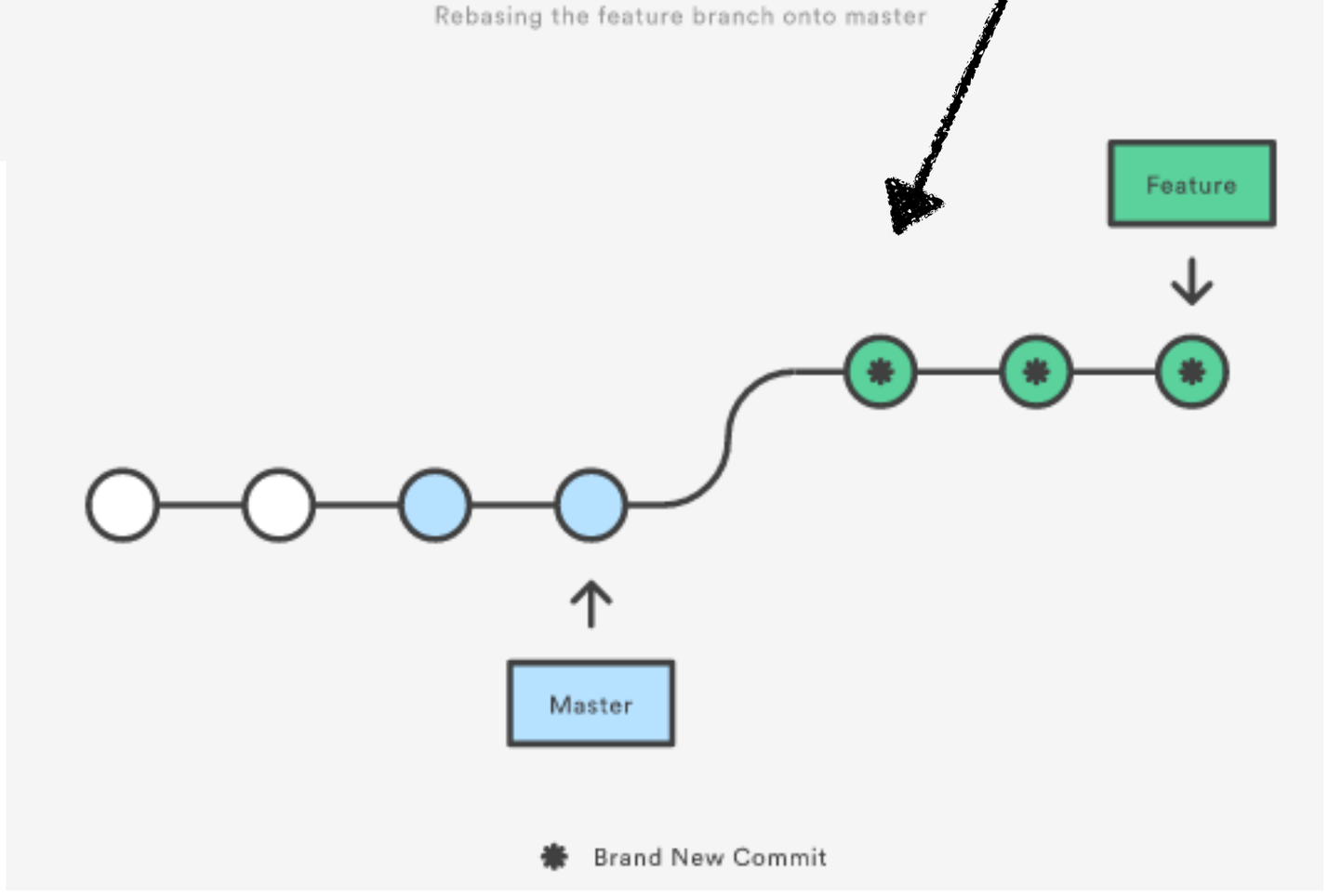


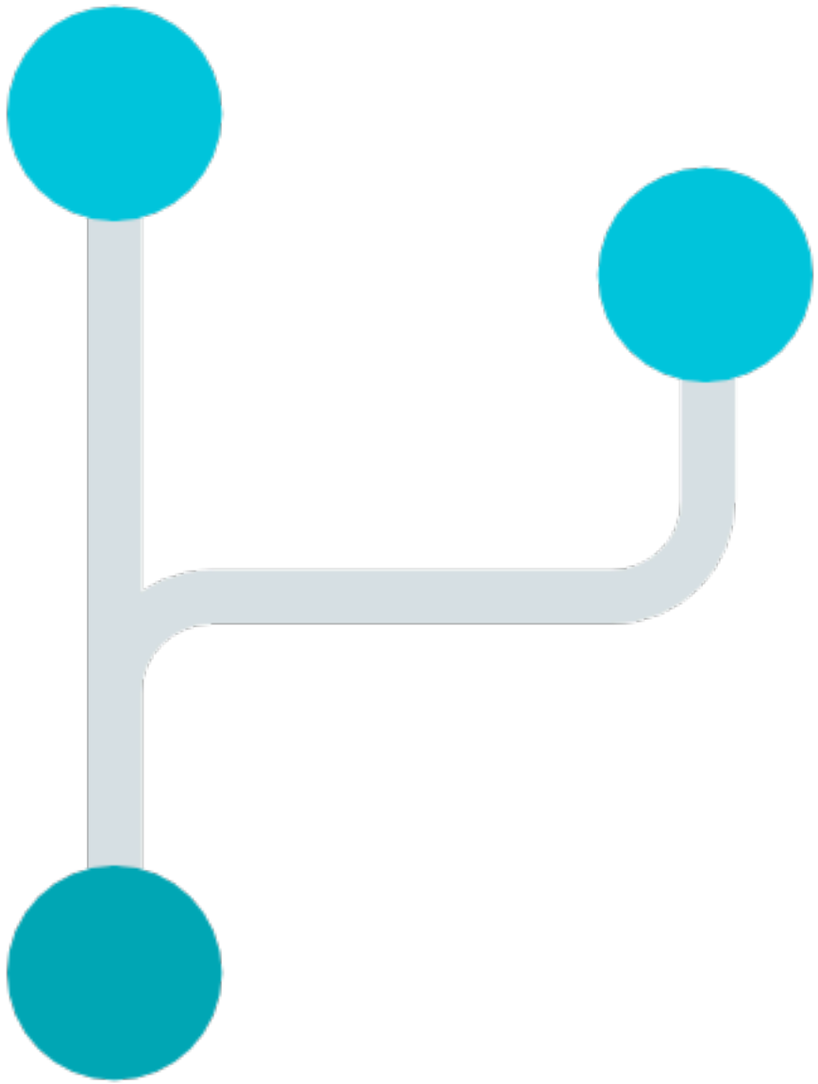
merging

rebasing

Merging vs. Rebasing

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





Fork

REMOTE

Someone else's repository.

cool_repo

Fork!

REMOTE

Your fork of the repository.

cool_repo fork

Clone to your computer from GitHub.

Push and **Pull** to your fork '**origin**'.

Pull from '**upstream**' changes to original.

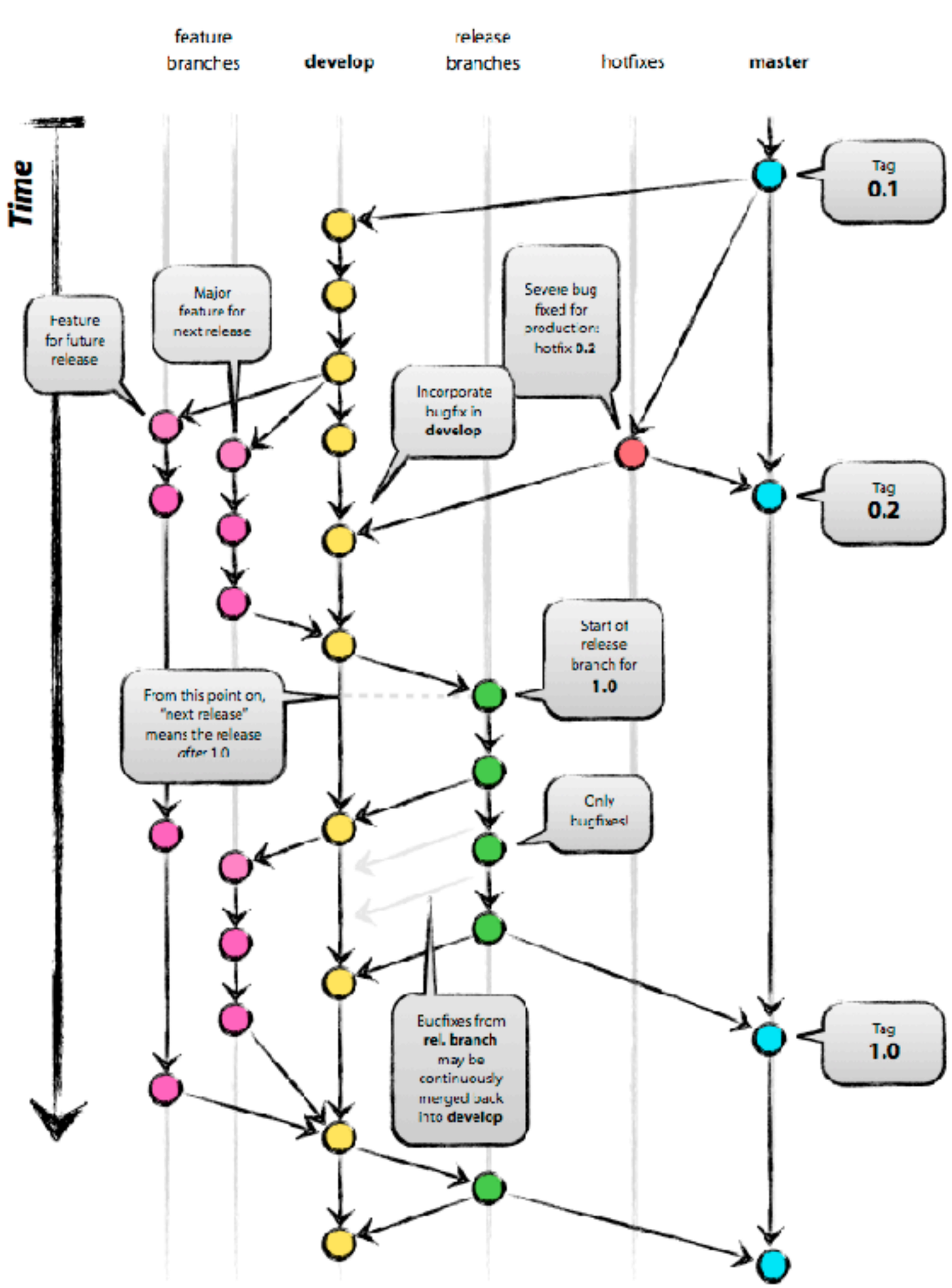
LOCAL

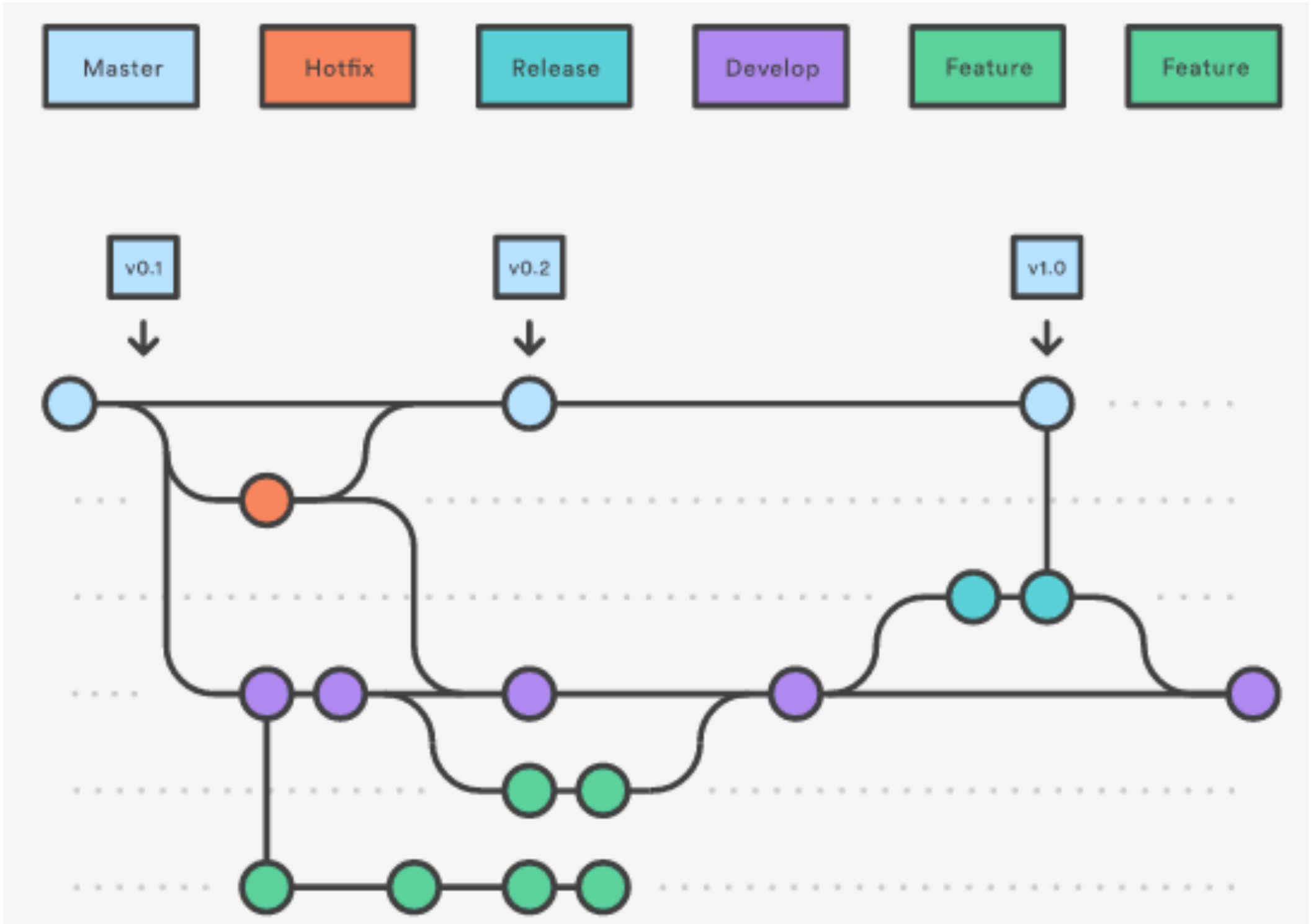
Use your computer's **terminal** to talk to two repositories via **two remotes** to the GitHub servers.

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!

Workshop GIT

Aprenda Git de uma vez por todas!

OBRIGADO!!!!

Paulo Igor