

Vous êtes Thésée...

Vous êtes Thésée...

Ariane est votre allié !

Vous êtes Thésée...

Ariane est votre allié !

La poésie entre vous :

You êtes Thésée...

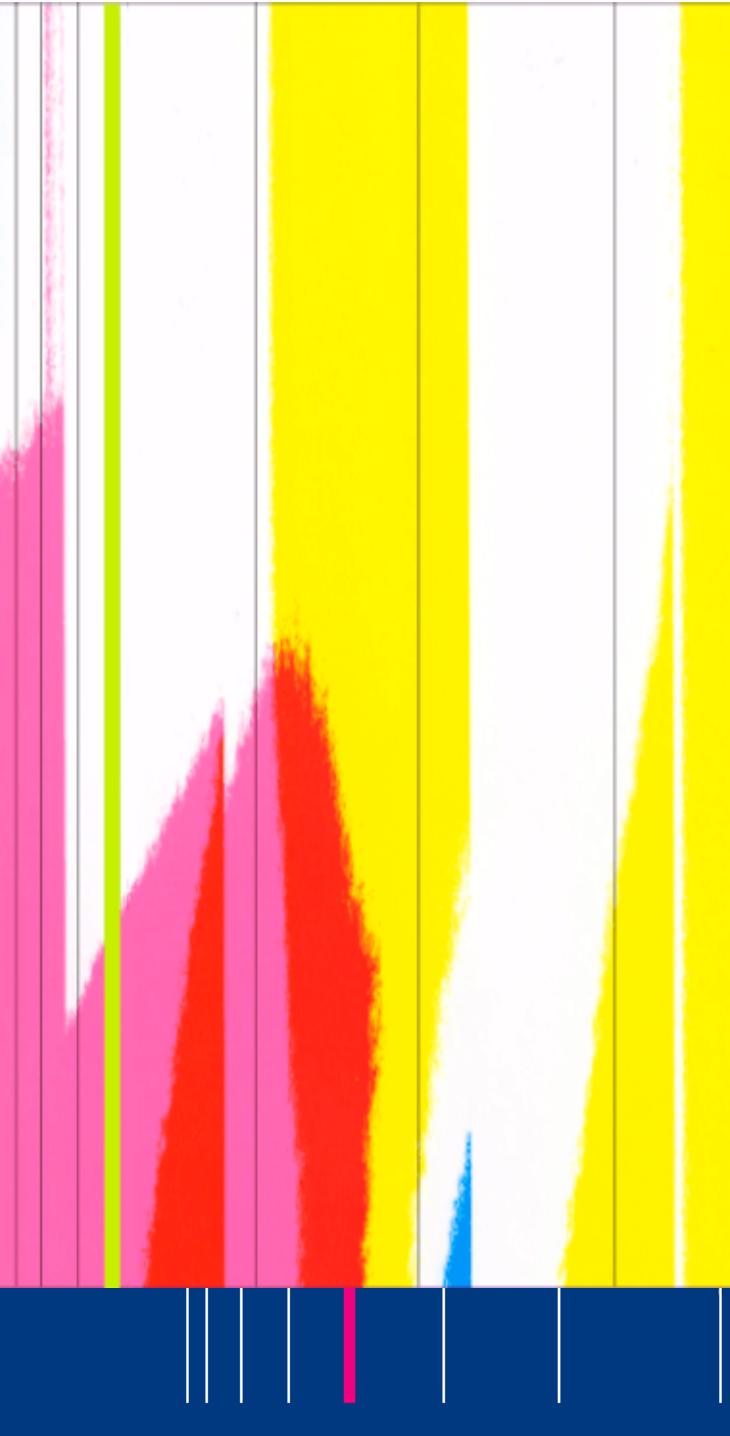
Ariane est votre allié !

La poésie entre vous :

Git et son langage !

# Les questions que se pose Thésée!

- « Qui a modifié le fichier X, il marchait bien avant et maintenant il provoque des bugs ! » ;
  - « Robert, tu peux m'aider en travaillant sur le fichier X pendant que je travaille sur le fichier Y ? Attention à ne pas toucher au fichier Y car si on travaille dessus en même temps je risque d'écraser tes modifications ! » ;
  - « Qui a ajouté cette ligne de code dans ce fichier ? Elle ne sert à rien ! » ;
  - « À quoi servent ces nouveaux fichiers et qui les a ajoutés au code du projet ? » ;
  - « Quelles modifications avions-nous faites pour résoudre le bug de la page qui se ferme toute seule ? »
- > Thésée aurait dû utiliser un fil d'Ariane alias un logiciel de gestion de versions!



Git :

# Le fil d'Ariane de vos projets, pilier des forges modernes

Claire Mouton

Basé sur les travaux de :  
Christophe Demarey, Julien Vandaele  
Research Centre INRIA Lille – Nord Europe



*Creatis*

# Why using SCM (Source Code Management)?

## Old school

```
cp myAlgo.c myAlgoWithFunctionalityB
```

```
cp myAlgo.c myAlgoWithFunctionalityC
```

=> Which one is the latest?

```
cp myAlgo.c myAlgo-v1.c
```

```
cp myAlgo.c myAlgo-v2.c
```

=> Which one has functionality B ?

## SCM

- One revision per functionality
- Author name + date

# Why using SCM?

- Retrieve a previous version
- Know when a feature / a bug has been introduced
  - Be able to diff files to easily find a bug, ...

```

template-1.py - /Users/schwehr/Desktop
template-2.py - /Users/schwehr/Desktop

class Template
#... (code from both files)

#_doc__ = ''
Example python file that is all tricked out. Designed for
unittest and doctest. Please keep updating to make this
possible template file.

Decimate these requirements to meet what you need
@requires: U{Python<http://python.org/>} >= 2.5
@requires: U{epydoc<http://epydoc.sourceforge.net/>} >= 3.1
@requires: U{psycopg2<http://initd.org/projects/psycopg2/>} >= 2.0.1

@undocumented: __doc__ parser success
@since: 2008-Feb-09
@status: under development
@organization: U{CCOM<http://ccom.unh.edu/>}

status: 3 differences
Actions
Choose left
Choose right
Choose both (left first)

```

# Why using SCM?

## Cooperative work

- Share the same vision of a software / document
  - A single reference repository
  - Avoid sending files via email to your colleagues
  - Be able to work in parallel on the same files
    - No manual diff
    - Automatic merge
    - Conflict management

## Product management

- Mainline for the development release
- Branches for experiments, maintenance revisions
- Tags for releases

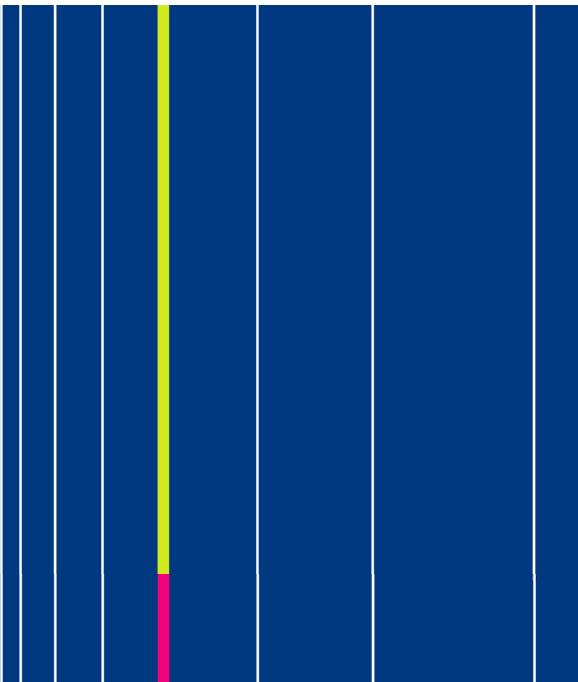
# What are SCM usages?

- Version history
  - Navigation
  - Retrieve older versions
  - ...
- Collaborative work
  - Simultaneous work of several people on the same project
    - ex: Conflicts resolution
  - Fork / work on another branch
    - ex: A PhD student who wants to start an experimentation
- Shouldn't be used to backup! Even if it saves a copy of your work...

# What are SCM usages?

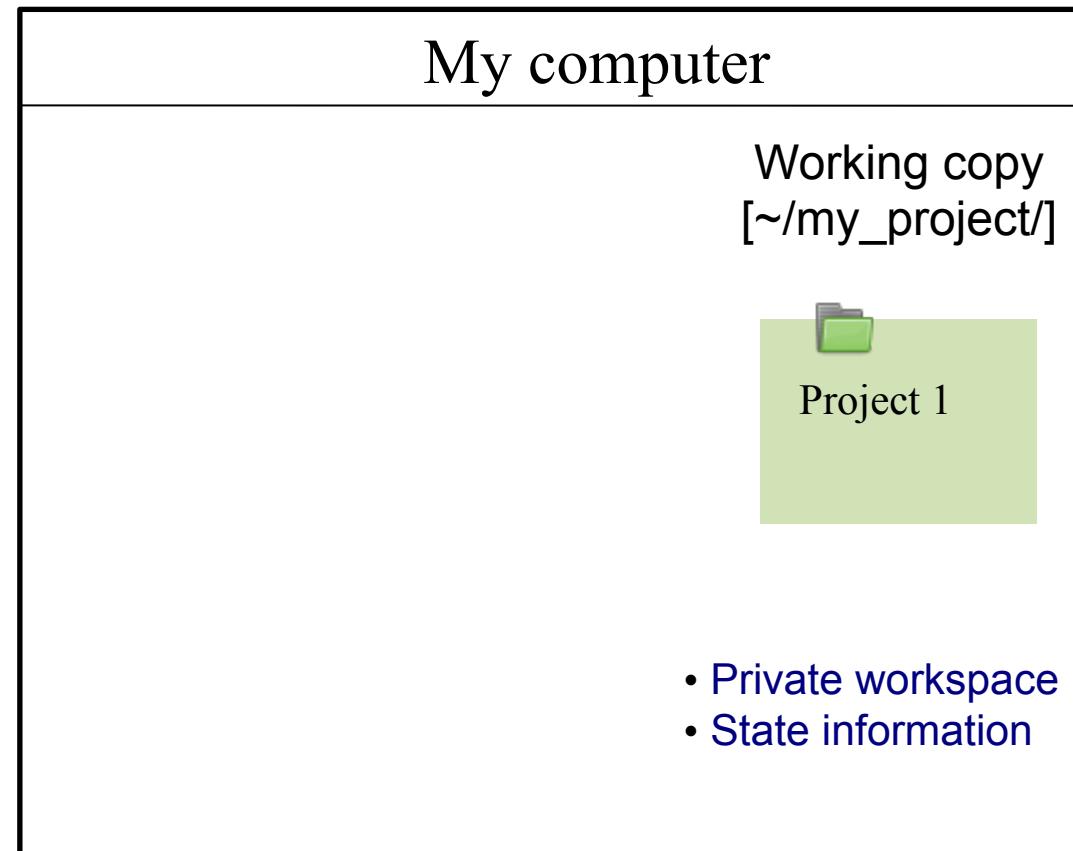
- Git is good to manage text files (based on diff) :
  - Scripts or code in any language
  - Article/documentation .tex
  - Web site in html
  - System configuration files
- Less suitable for :
  - Binaries
    - Solutions : git-annex, git-lfs, git-fat, git-bigfile, git-bigstore, git-sym, ...
  - Formatted text (Microsoft Word, Open Office, PDF etc.)
    - Solution : Combine git with a diff tool such as Word Diff or Pandoc
  - Images
  - .dll
  - Databases (like MySQL)
    - Solution : Use Git for the output of mysql dump
  - ...

# Core Notions

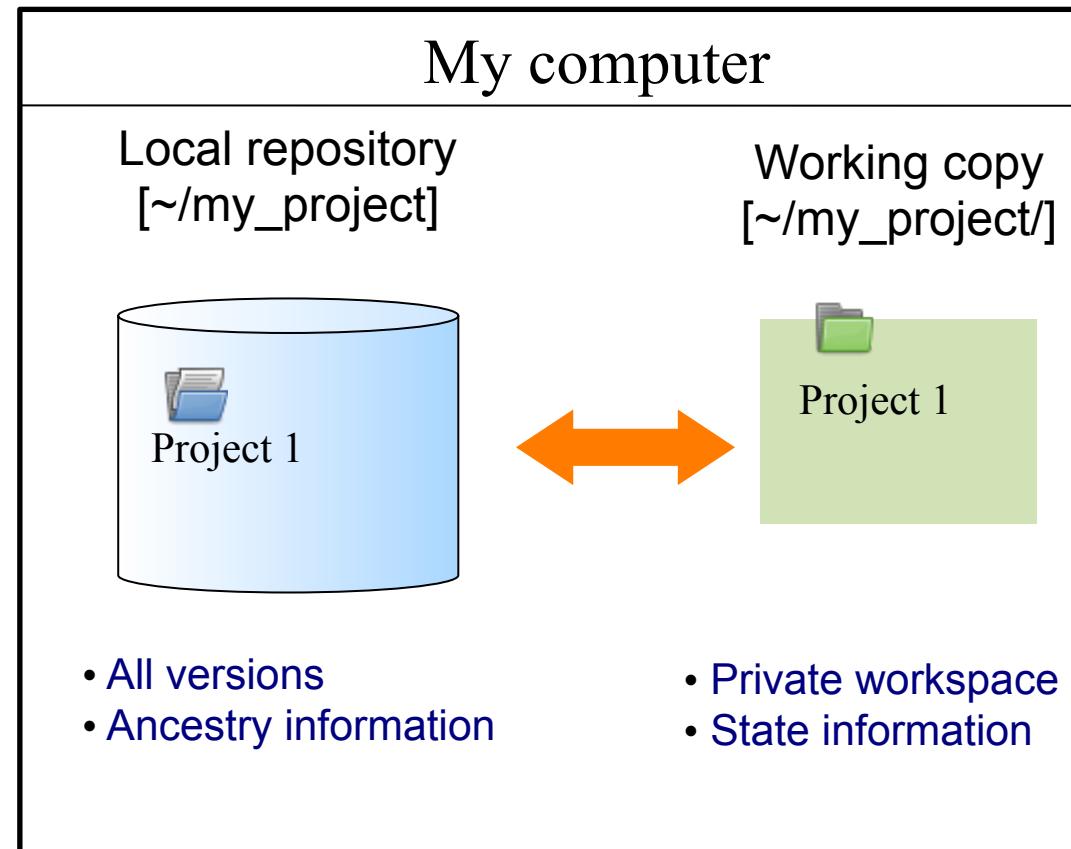
A series of vertical bars of varying heights and colors (white, yellow, red) are positioned along the left edge of the slide.

Creatis

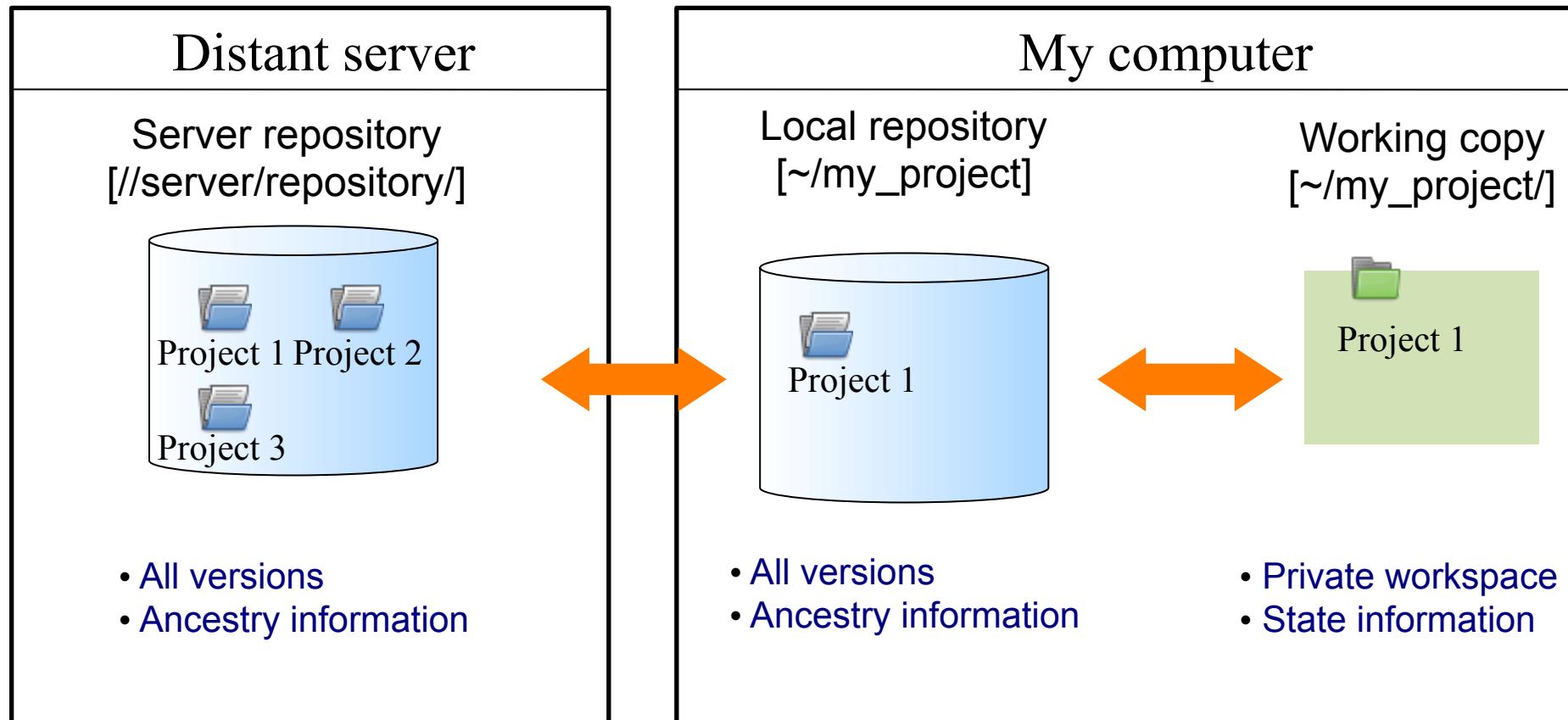
## Core notions > *Organization*



## Core notions > *Organization*



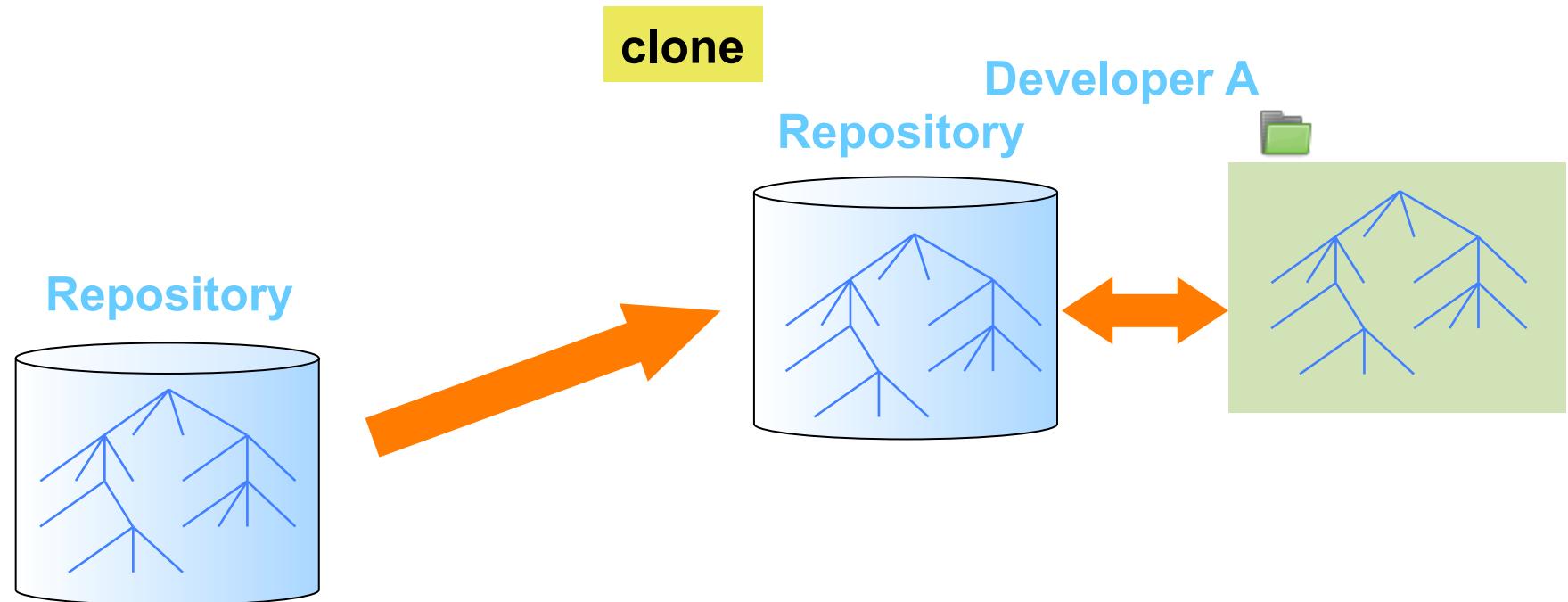
## Core notions > *Organization*



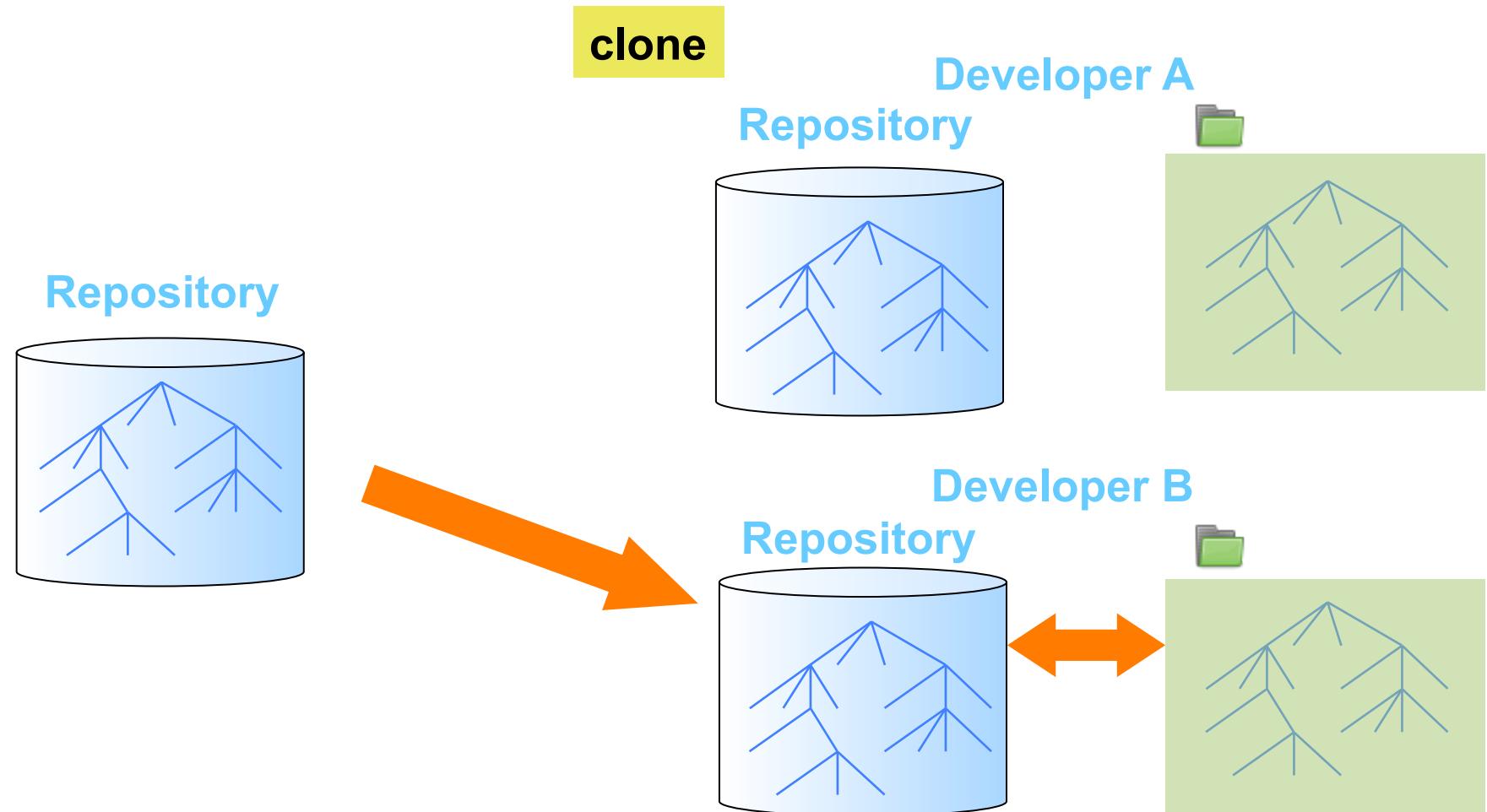
# Core notions >

## A scenario with 2 developers and a remote repository?

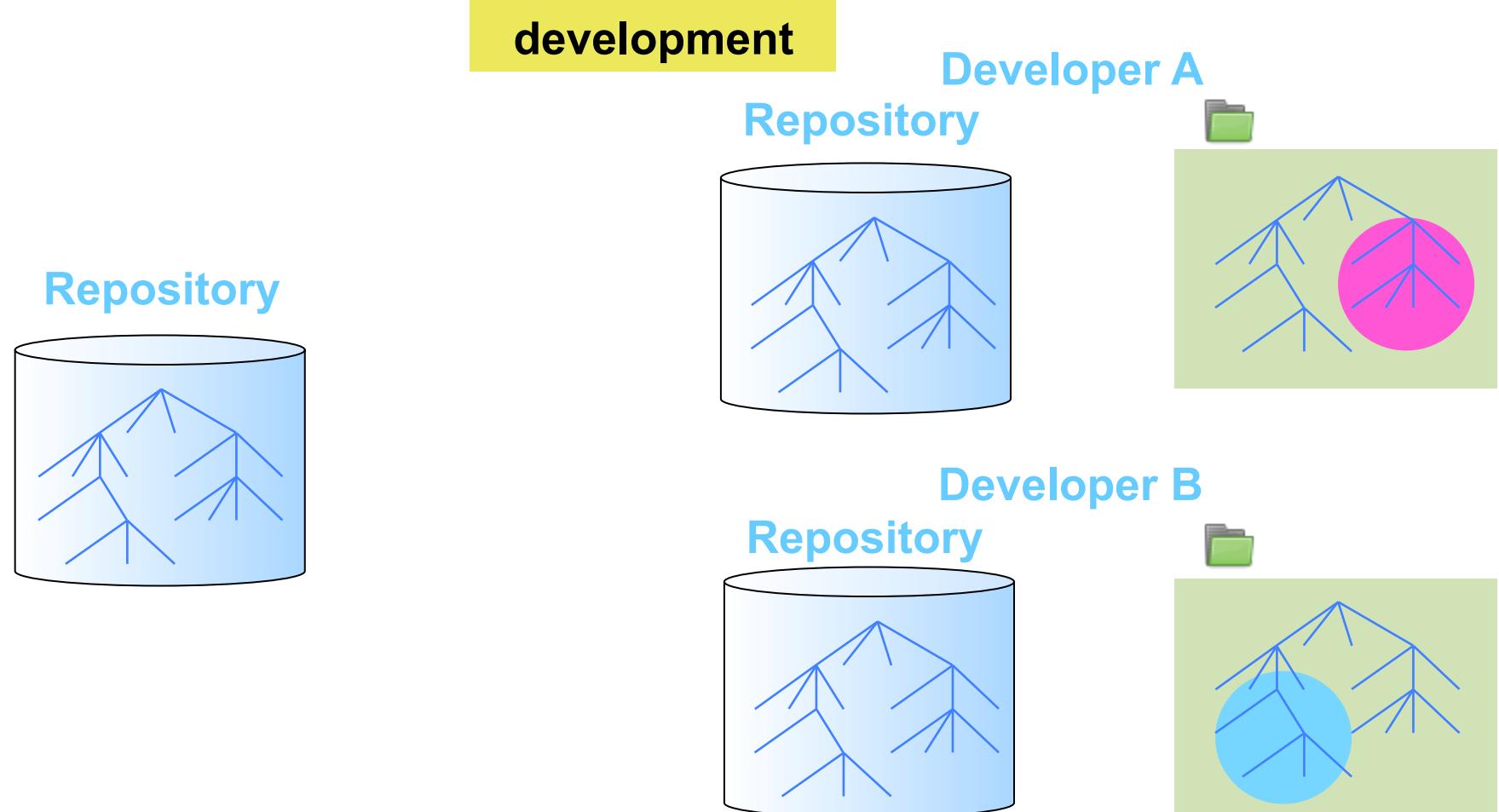
## Core notions > Get a project



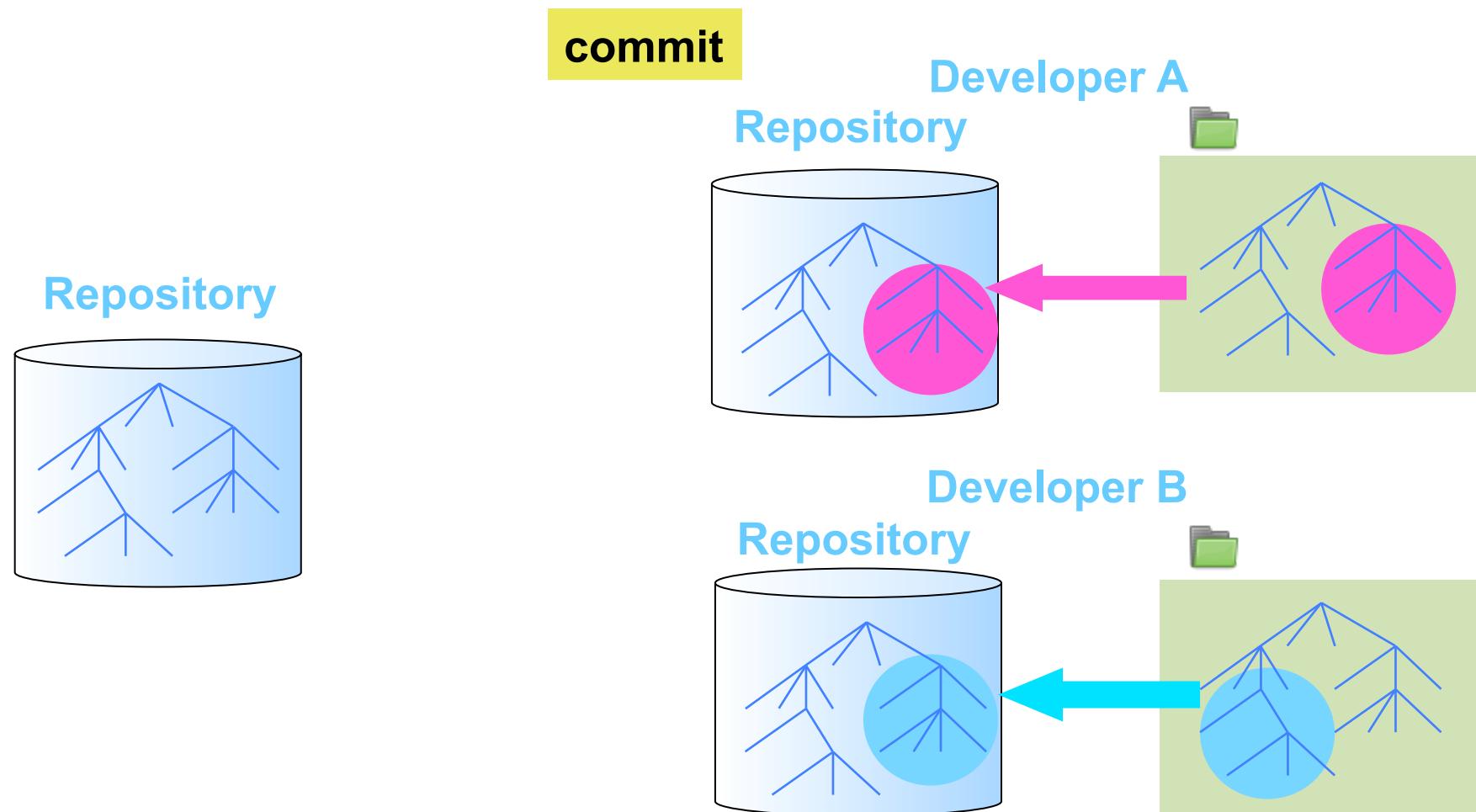
## Core notions > Get a project



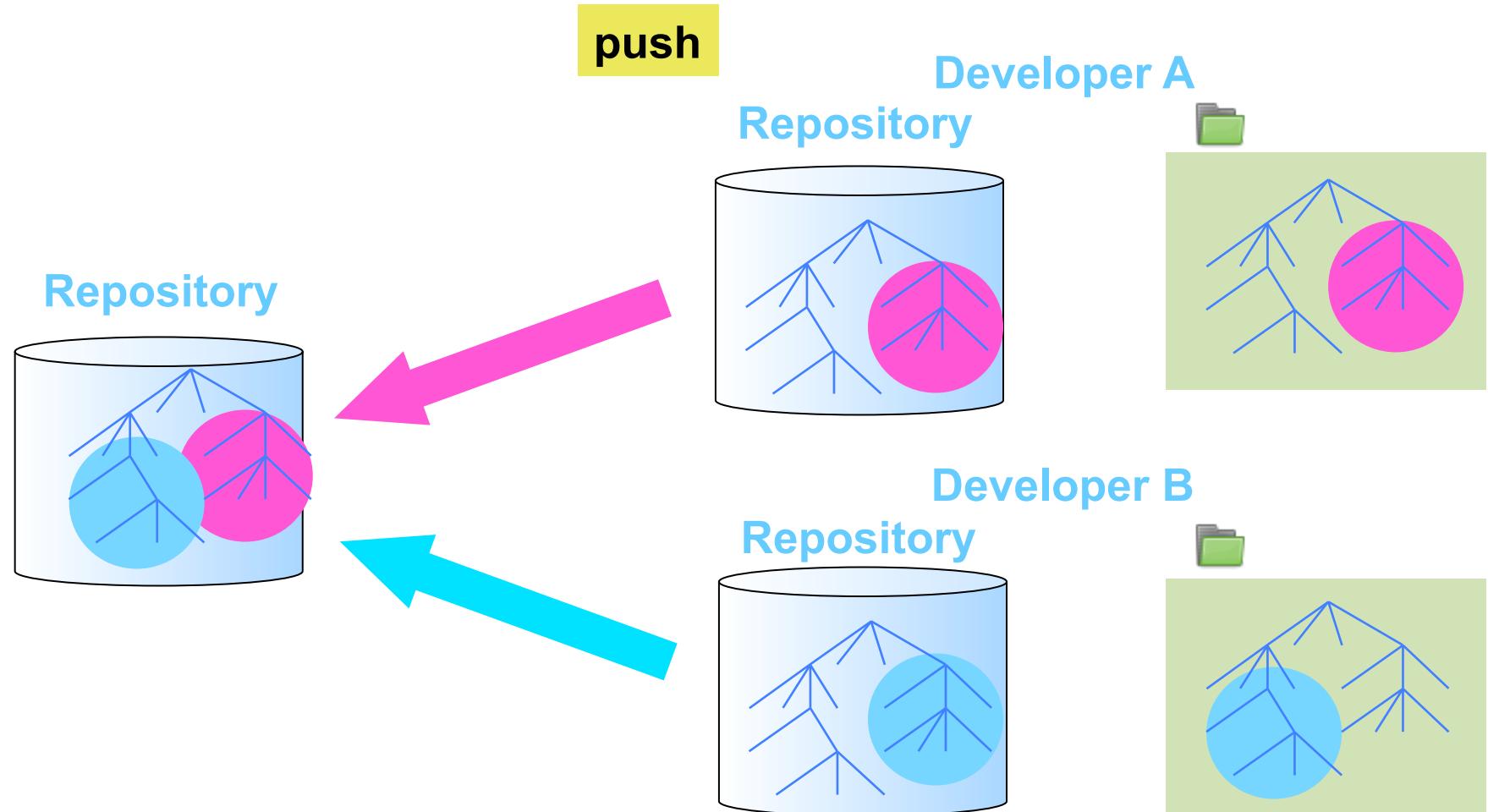
# Core notions > Commit



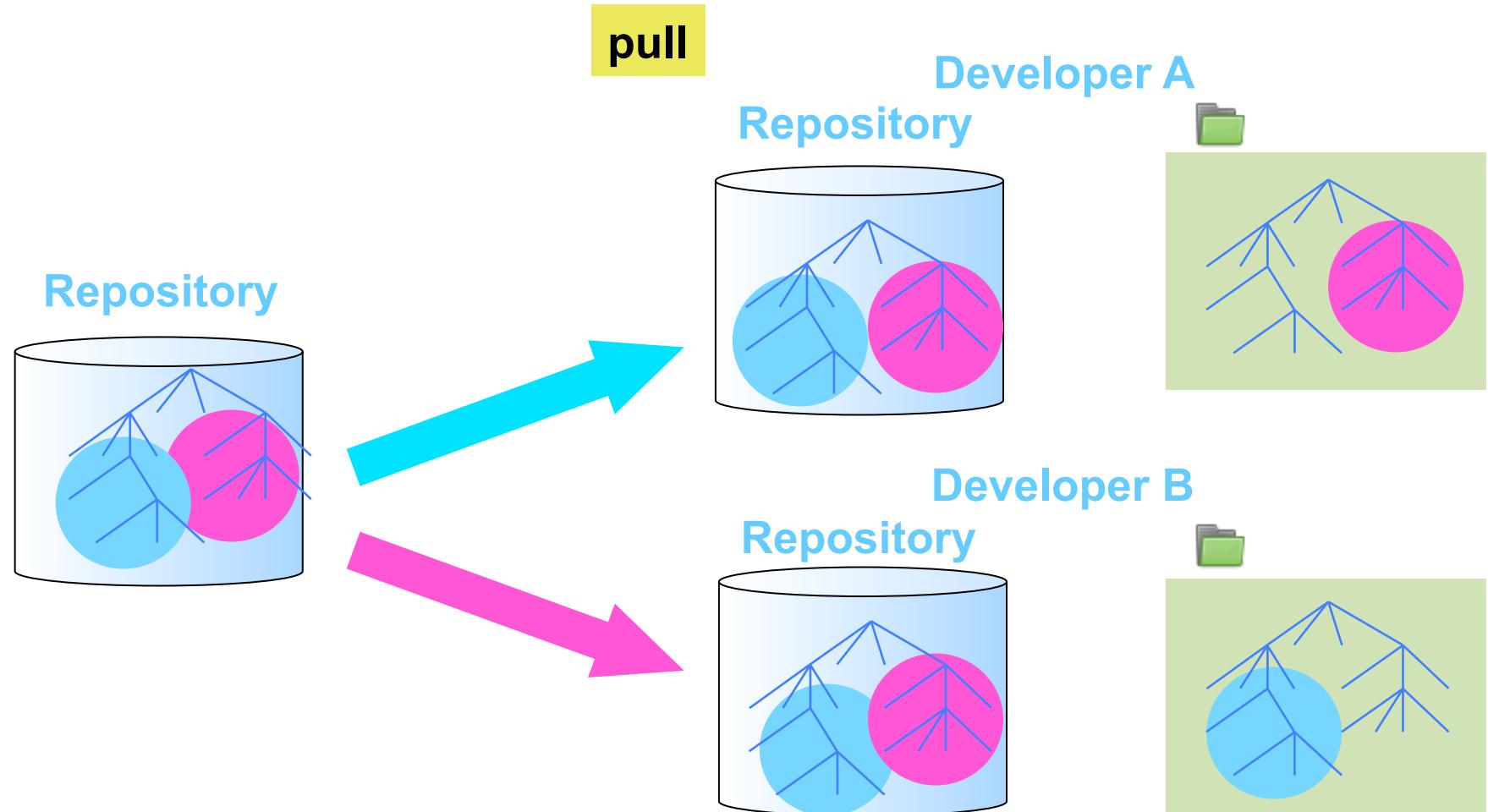
## Core notions > Commit



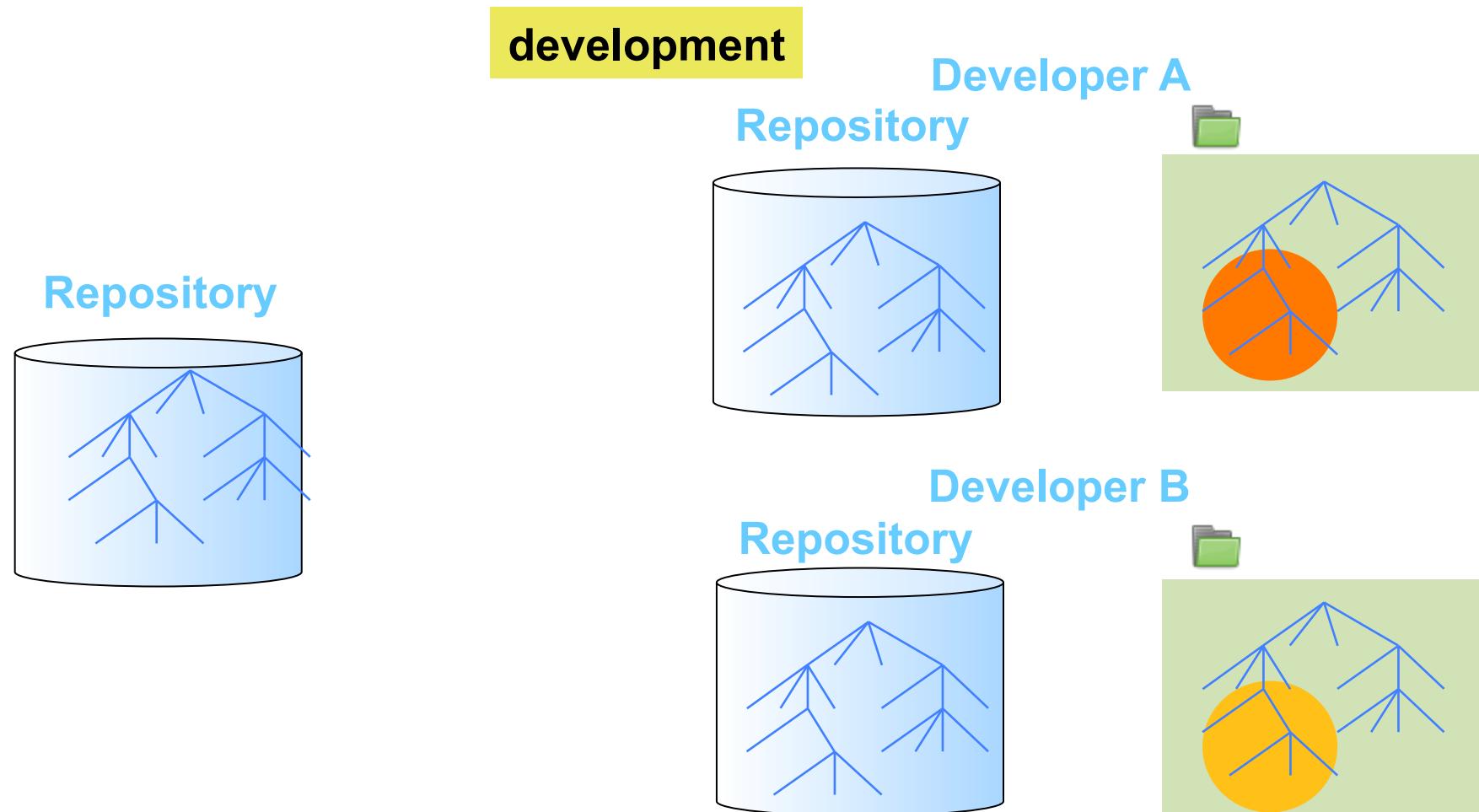
## Core notions > Commit



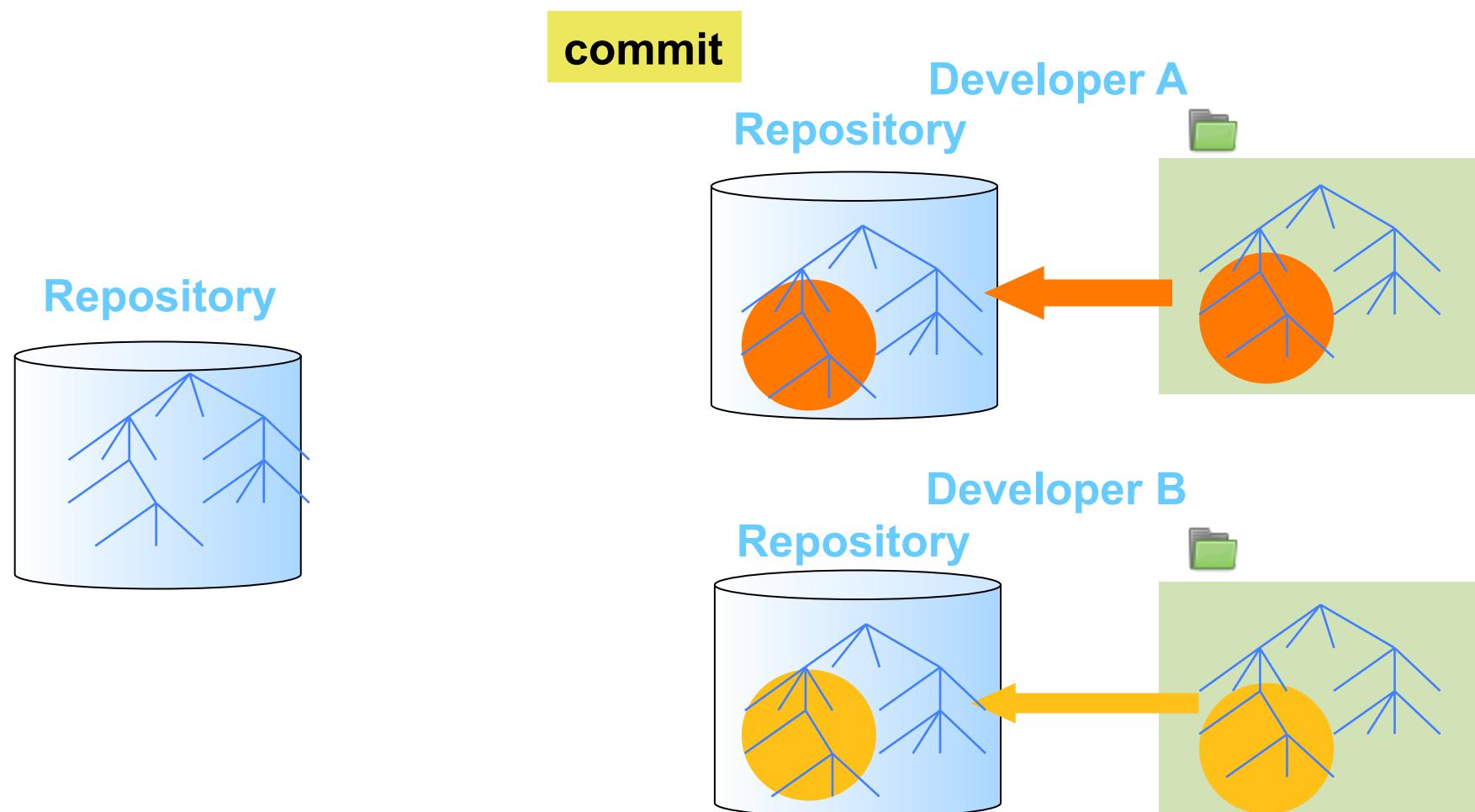
## Core notions > Commit



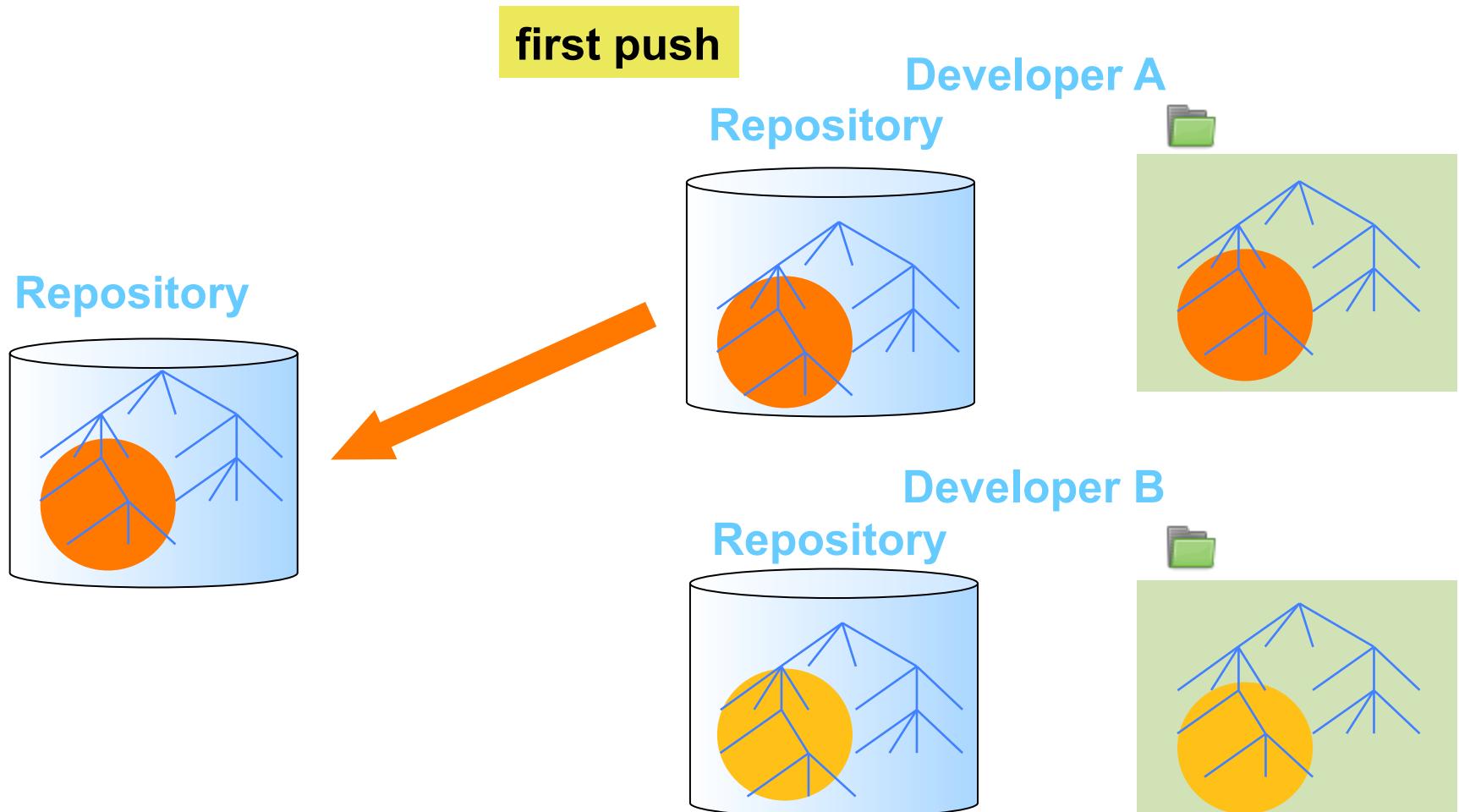
## Core notions > Conflict



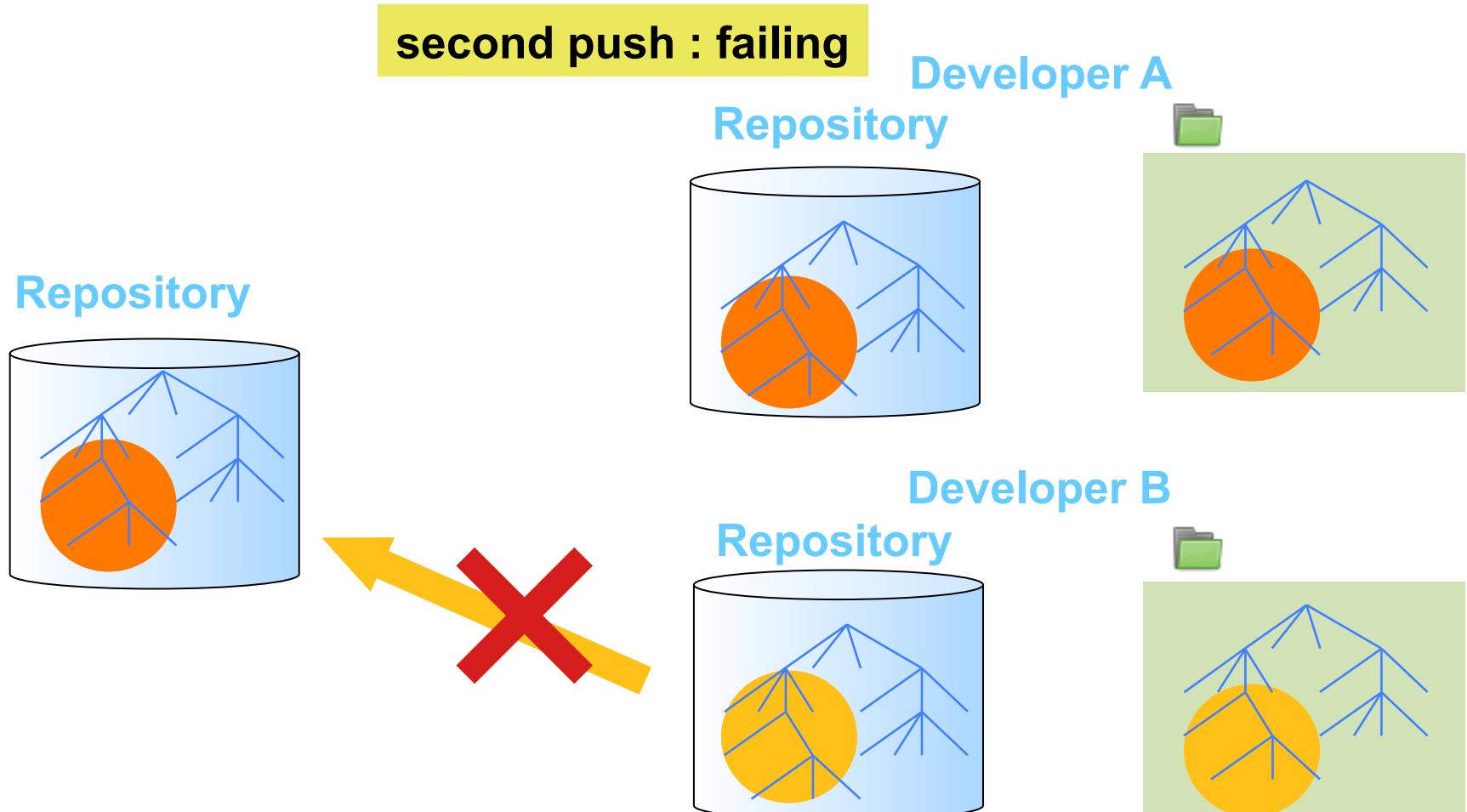
## Core notions > Conflict



## Core notions > Conflict

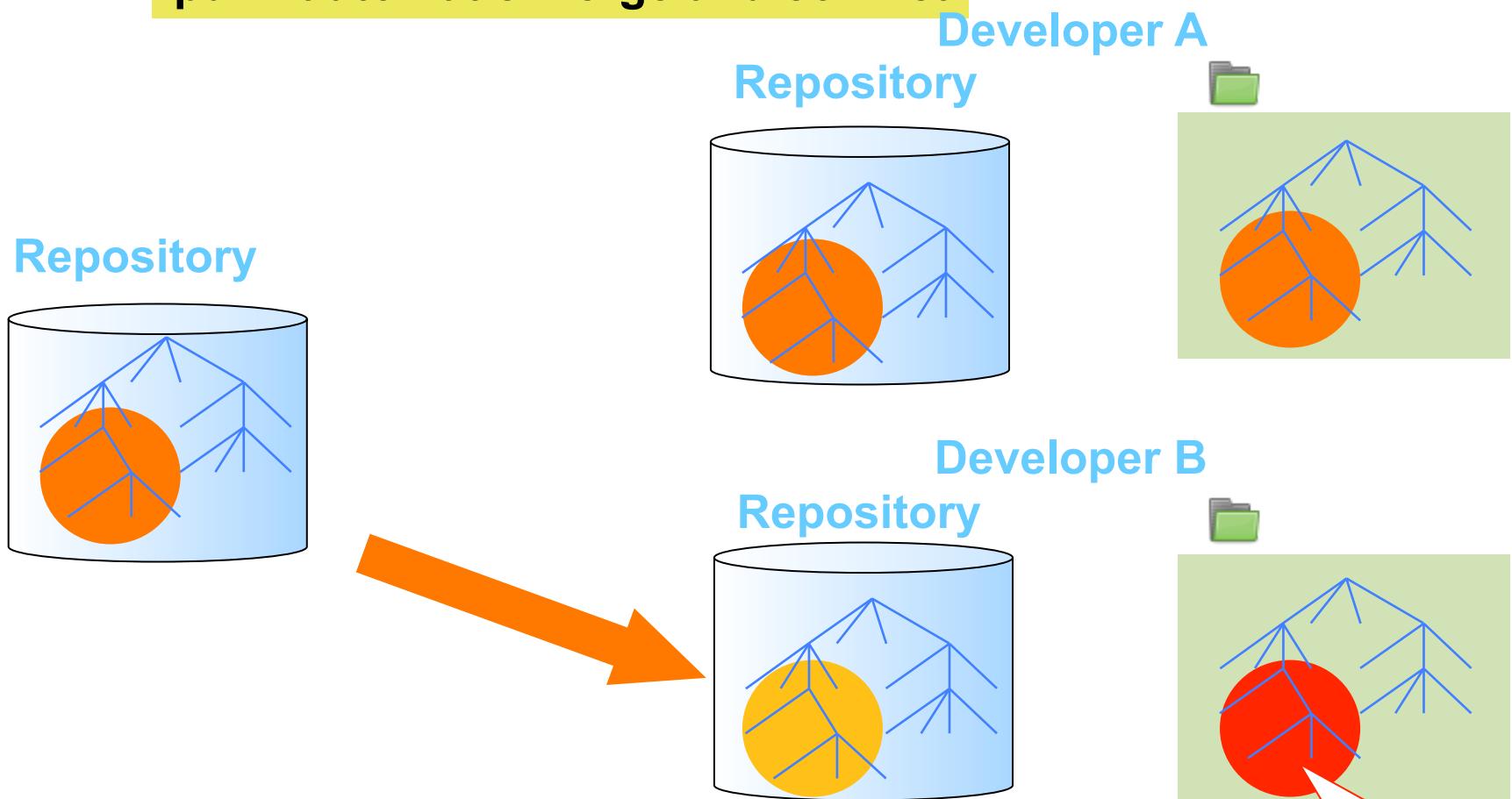


## Core notions > Conflict

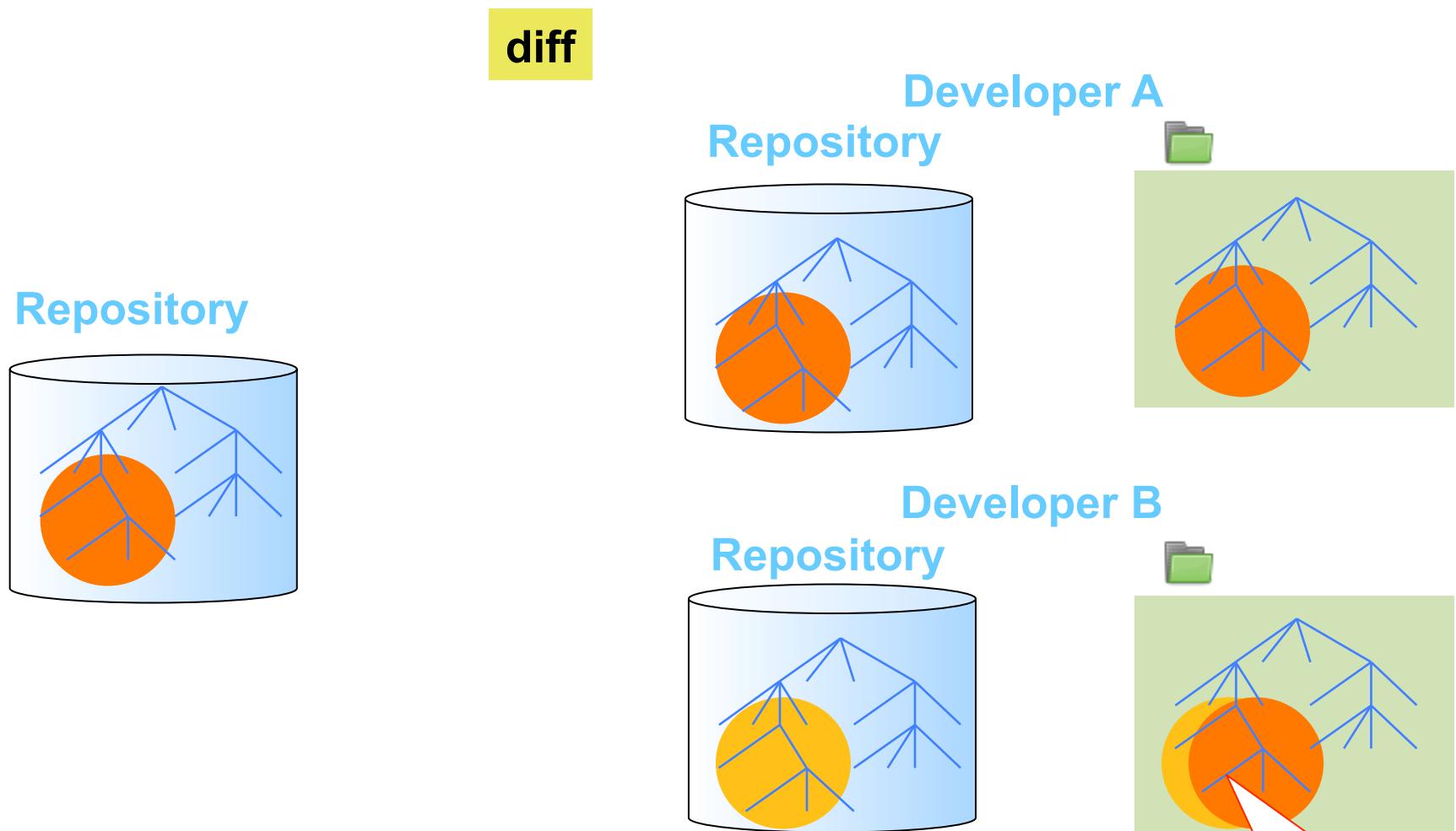


## Core notions > Conflict

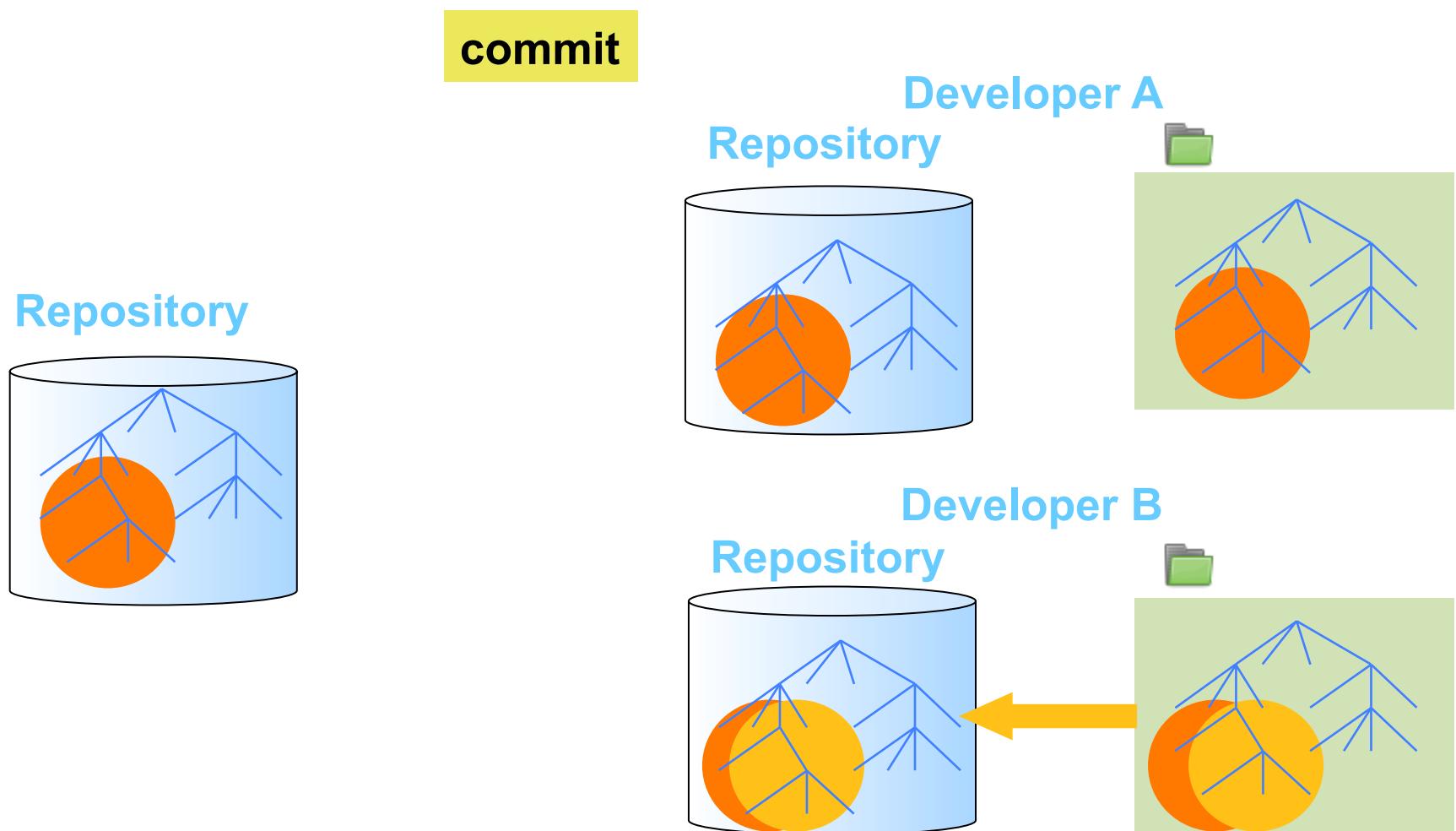
**pull : automatic merge and conflict**



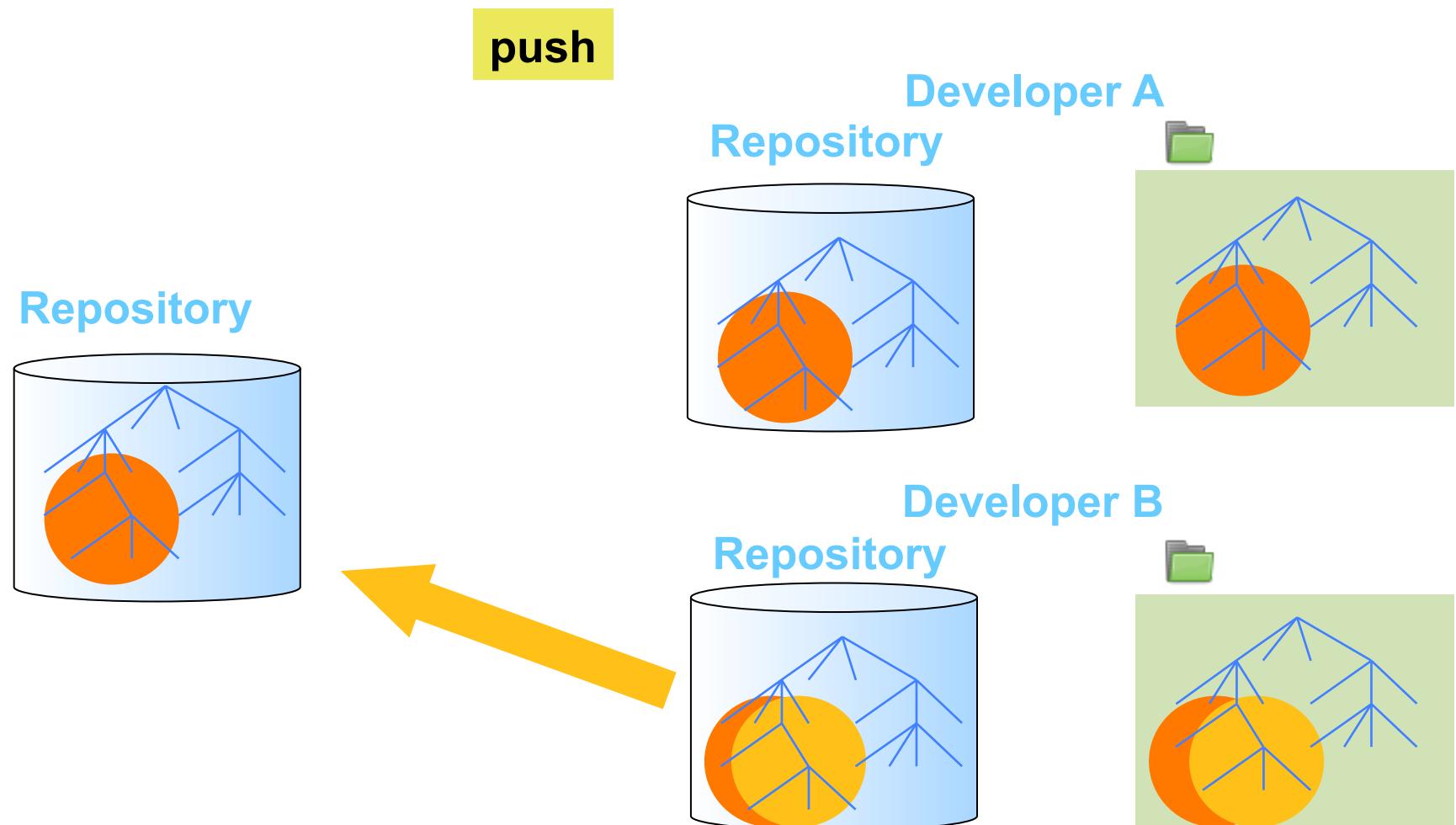
## Core notions > Conflict



## Core notions > Conflict



## Core notions > Conflict



## Core notions > Version tagging

tag

- Often used to tag the repository on important events such as a software release or article submission
- Allows you to easily retrieve a specific version of the software/article
  - Use / distribute a given release
  - Reproduce bugs for a given release

my\_project-v1.3

- Push a tag:

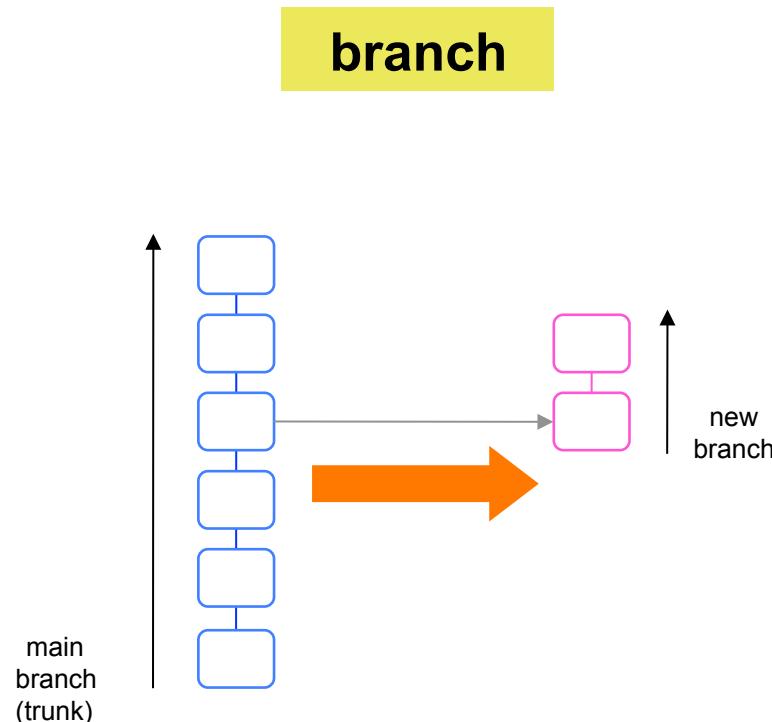
git push origin mytag

or git push --tags

## Core notions > *Branch*

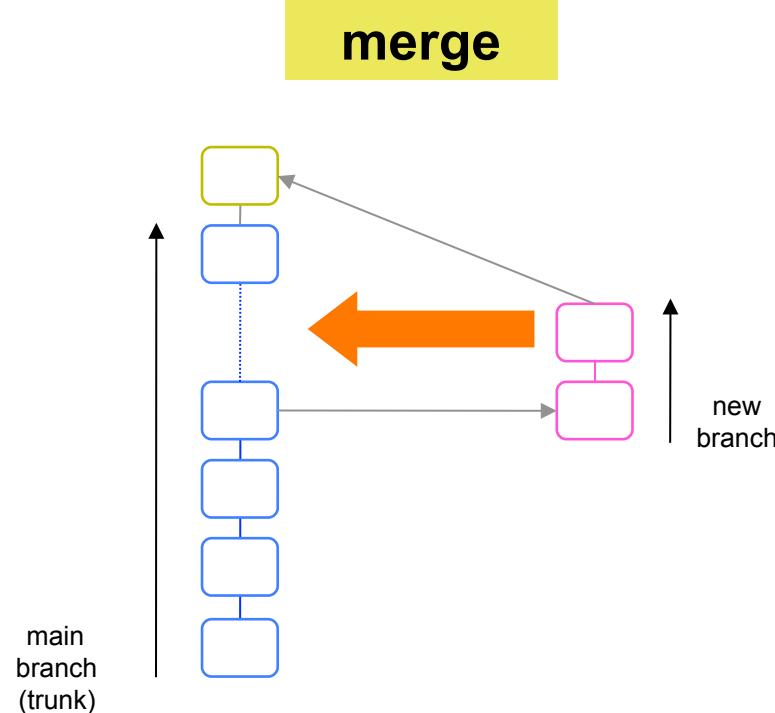
Why?

# Core notions > *Branch*



- A branch = a reference to a commit
- Maintenance and development (maintain version N and develop version N+1)
- Work on a project sub-set (one branch per feature)
- Experimental development
- Save commits temporarily

## Core notions > *Branches merging*



"Merge branches" :

Bring changes from a branch to another branch

# Work unit of Source Code Management

A *repository* contains the complete history of the project (i.e. all the revisions)

A *revision* (a.k.a. *commit* or *version*)

- Is a snapshot of all the tracked files
- Is usually based upon one other revision
- Corresponds to an identified author
- Contains a message that explains the rationale for the modifications introduced by the revision and any other info the author considers relevant
- Is atomic

For example, changeset = modifications on « calc.c AND calc.h ».

# Identifying a commit

A commit has a unique identifier :

- Revision number (SHA-1 identifier)
- String (commit message)

A commit can be identified in multiple ways, e.g.:

- Its SHA1 (possibly abbreviated)
- A reference (e.g. HEAD)
- An indirect reference : HEAD^, HEAD~, ...
- A branch name
- ...
- A tag

# Centralized SCM

## > CVS and Subversion (SVN)



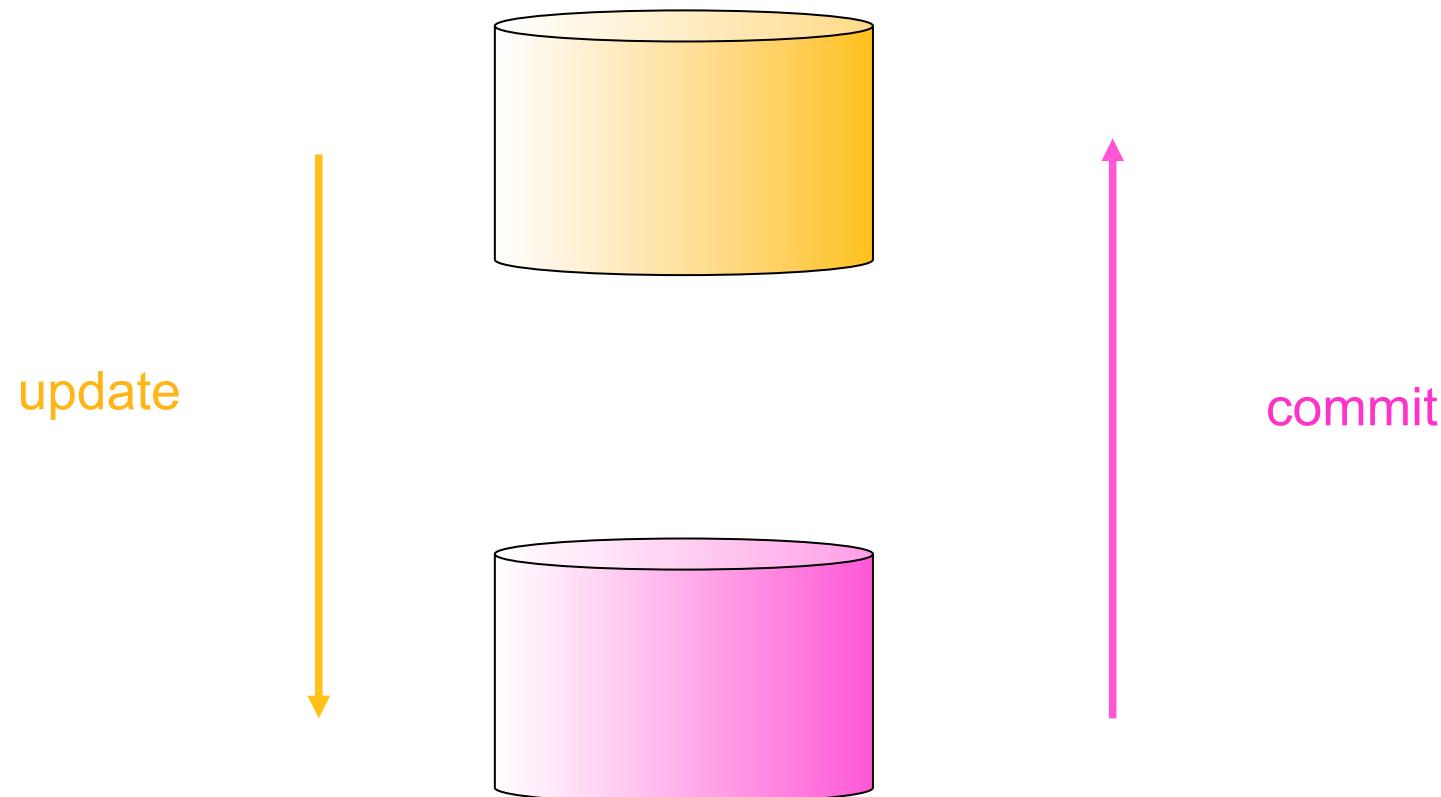
Creatis

# CVS and its evolution, SVN

- Are versioned :
  - Files
  - Directories
  - Meta-data (properties)
- Possible to move / rename elements (no history loss)
- Atomic commit
  - Done only if the whole operation is a success
  - One revision number by commit (per file for CVS)

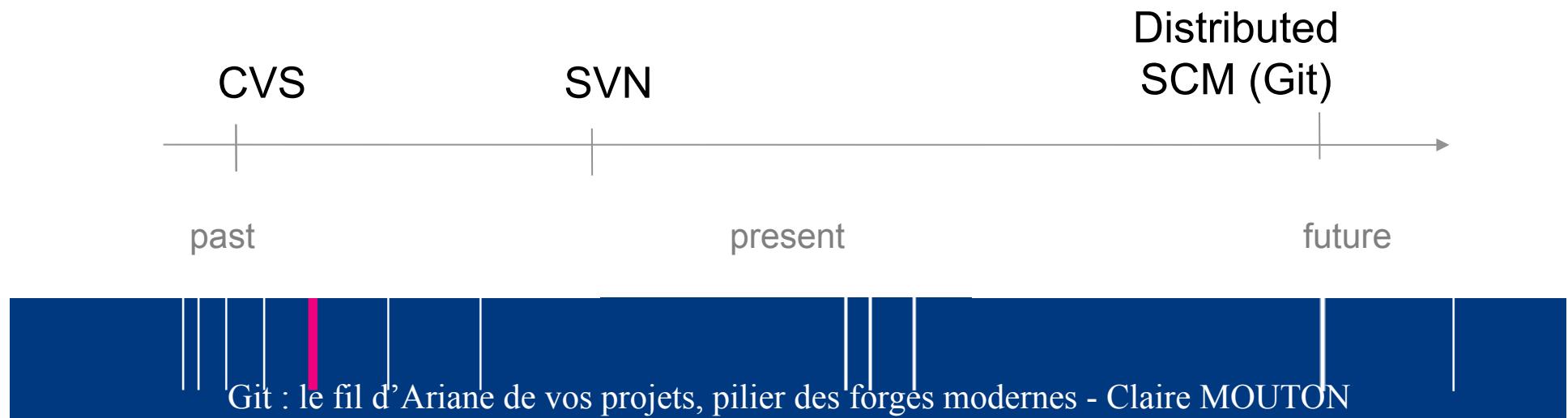
# SVN

## > *Exchanges with (remote) repository*

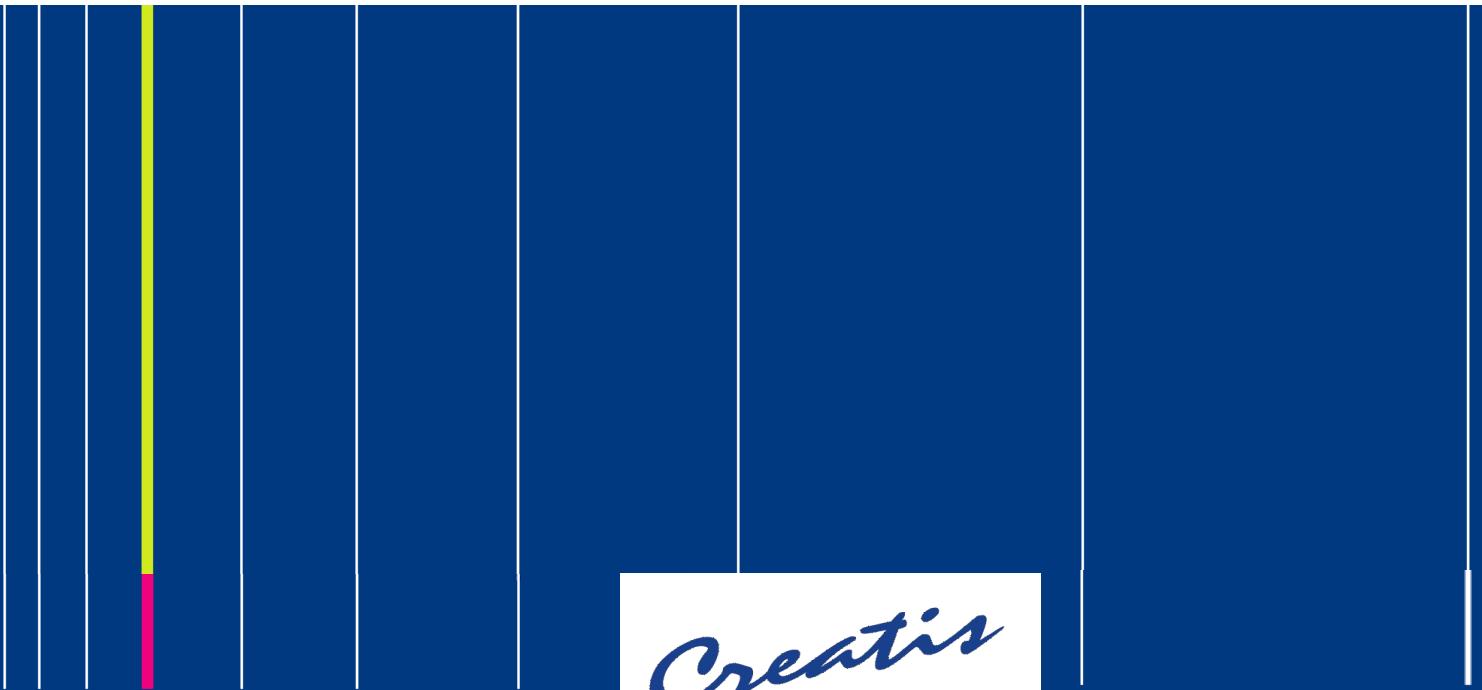


# Conclusion on centralized SCM

- + A central/core repository
  - + Easy to use  
  - Need to be on line for almost commands
  - Privileged users (committers)

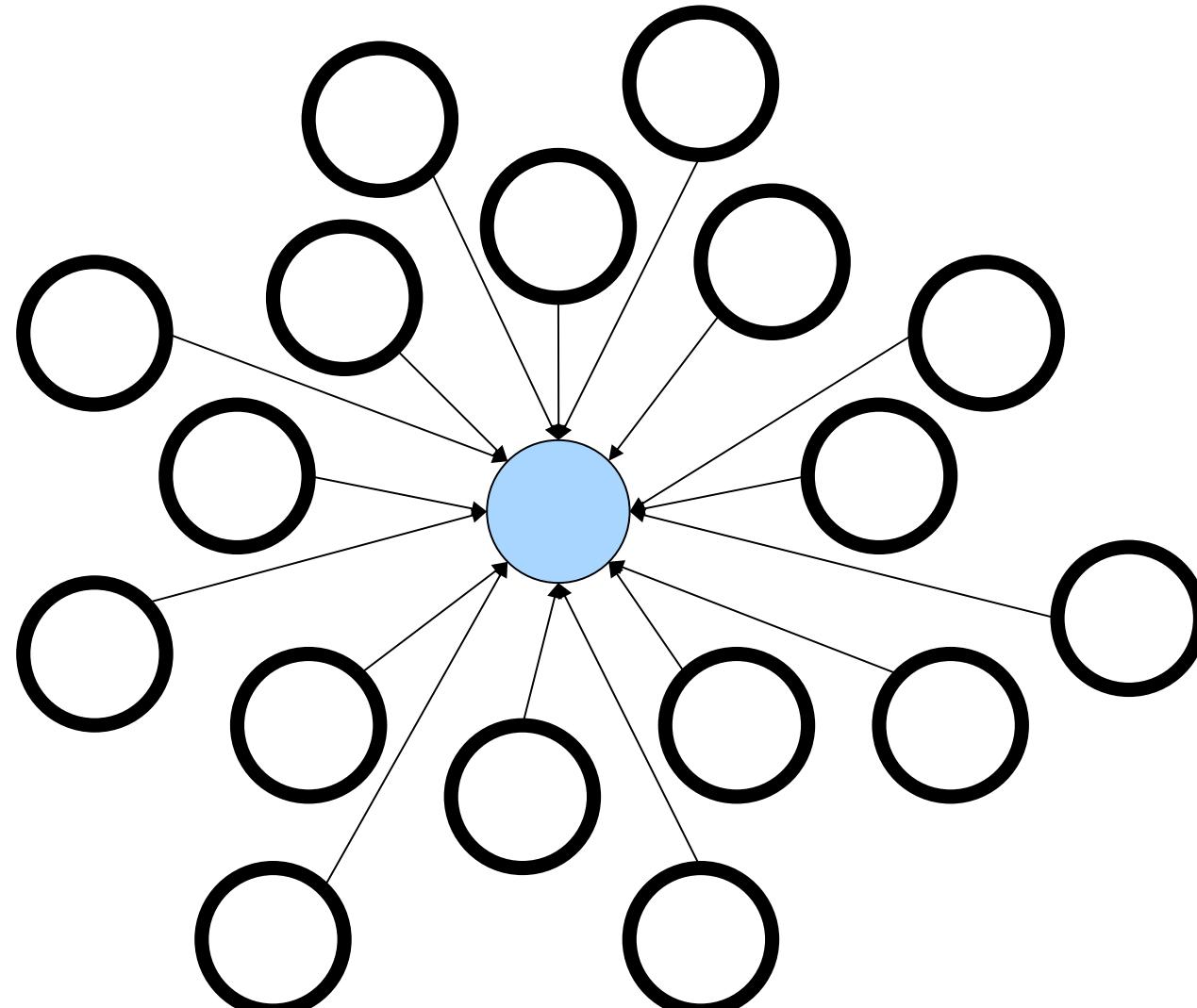


# Distributed SCM

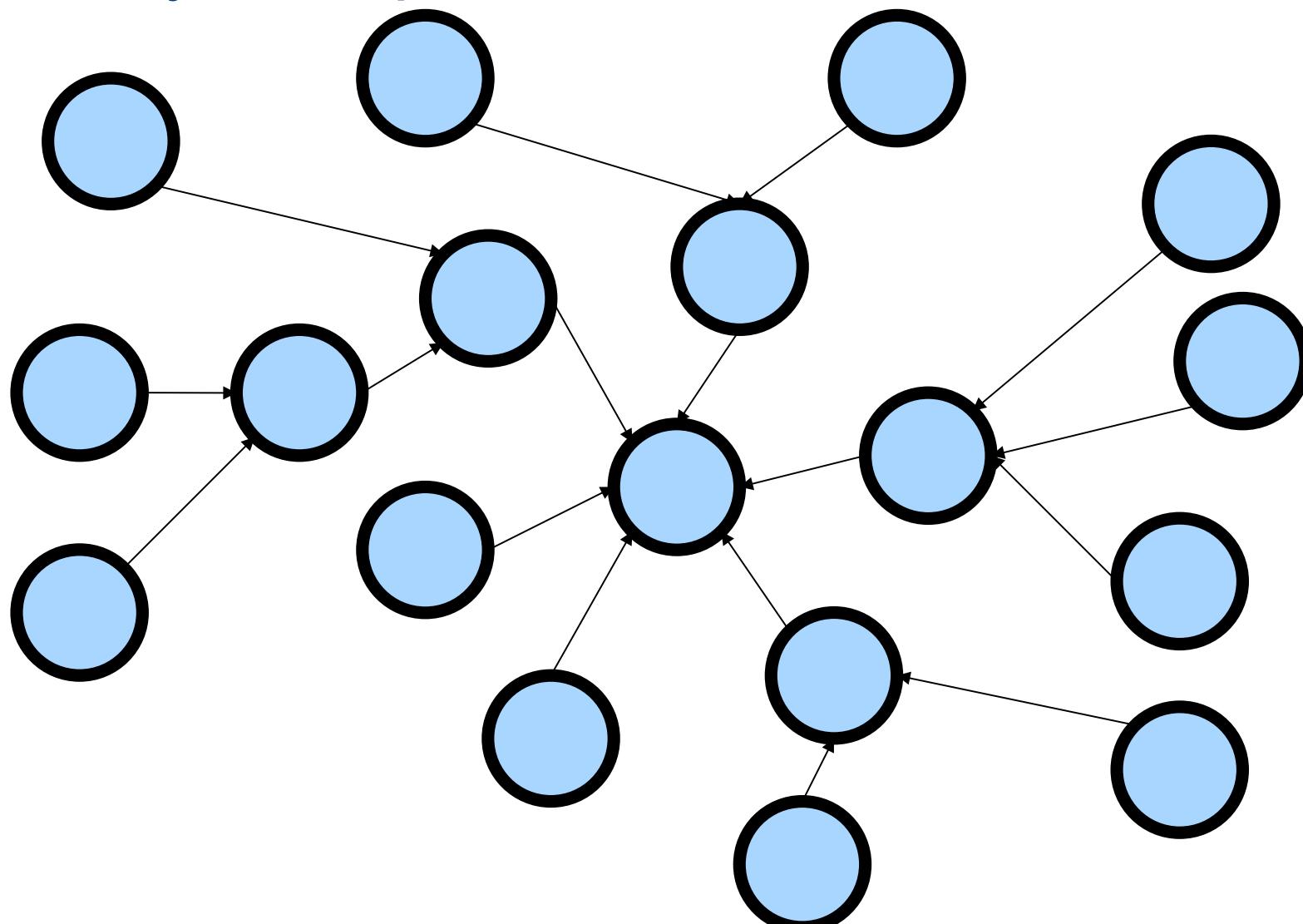


*Creatis*

## Policy example: centralized



## Policy example: distributed

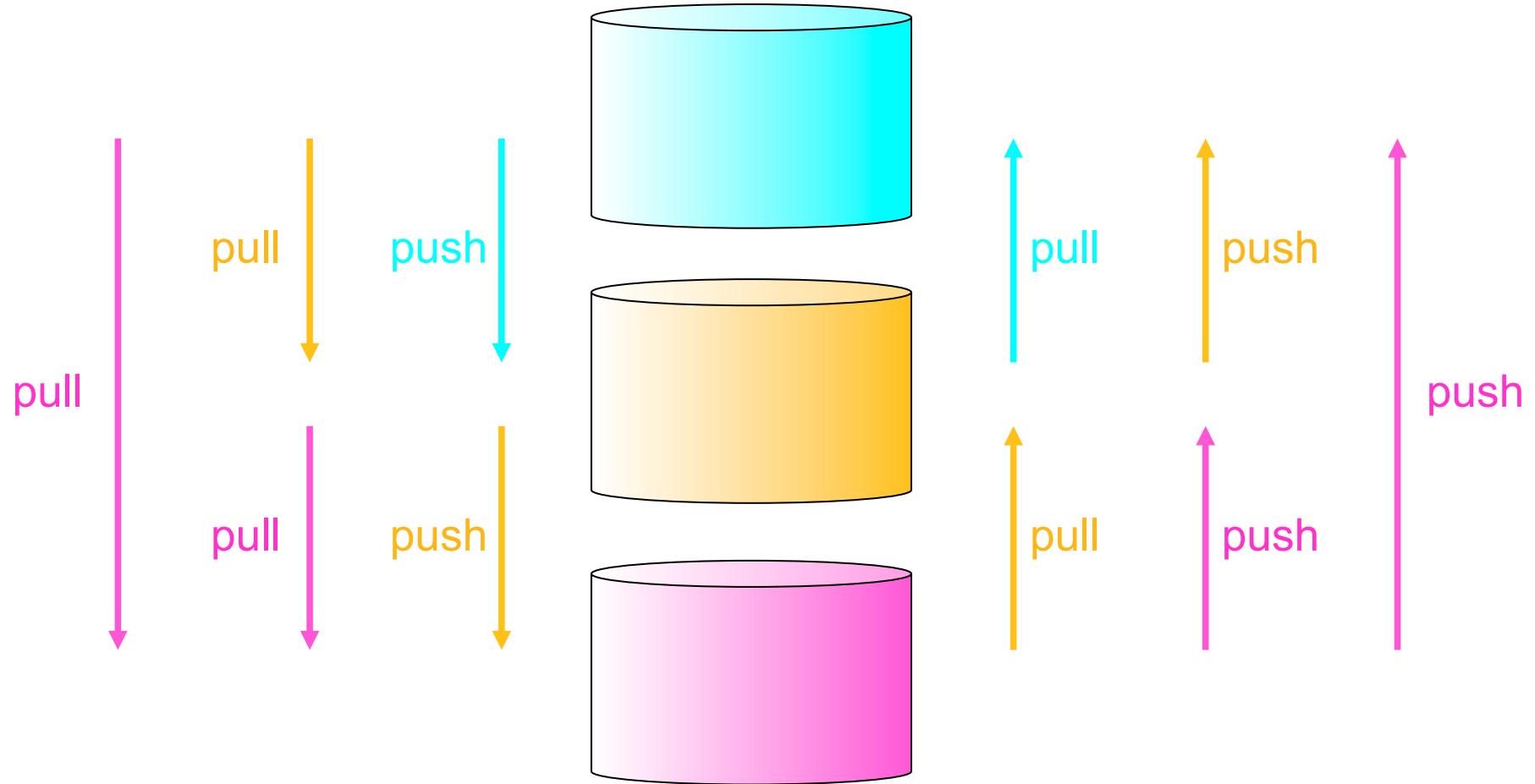


# Decentralized benefits ?

# Decentralized benefits

- Each developer can have his own repository
  - Off-line use (commands available offline)
    - ex: Do a local commit
  - Create a branch without having to ask authorization
    - ex: Open Source community
- Synchronisation needed between repositories
  - pull = get changes from a remote repository to your local repository
  - push = post your changes to a remote repository

# Exchanges between (remote) repositories



Outil	Type	Description	Projets qui l'utilisent
<a href="#">CVS</a>	Centralisé	C'est un des plus anciens logiciels de gestion de versions. Bien qu'il fonctionne et soit encore utilisé pour certains projets, il est préférable d'utiliser SVN (souvent présenté comme son successeur) qui corrige un certain nombre de ses défauts, comme son incapacité à suivre les fichiers renommés par exemple.	OpenBSD...
<a href="#">SVN (Subversion)</a>	Centralisé	Probablement l'outil le plus utilisé à l'heure actuelle. Il est assez simple d'utilisation, bien qu'il nécessite comme tous les outils du même type un certain temps d'adaptation. Il a l'avantage d'être bien intégré à Windows avec le programme <a href="#">Tortoise SVN</a> , là où beaucoup d'autres logiciels s'utilisent surtout en ligne de commande dans la console. Il y a un <a href="#">tutoriel SVN</a> sur le Site du Zéro.	Apache, Redmine, Struts...
<a href="#">Mercurial</a>	Distribué	Plus récent, il est complet et puissant. Il est apparu quelques jours après le début du développement de Git et est d'ailleurs comparable à ce dernier sur bien des aspects. Vous trouverez un <a href="#">tutoriel sur Mercurial</a> sur le Site du Zéro.	Mozilla, Python, OpenOffice.org...
<a href="#">Bazaar</a>	Distribué	Un autre outil, complet et récent, comme Mercurial. Il est sponsorisé par Canonical, l'entreprise qui édite Ubuntu. Il se focalise sur la facilité d'utilisation et la flexibilité.	Ubuntu, MySQL, Inkscape...
<a href="#">Git</a>	Distribué	Très puissant et récent, il a été créé par Linus Torvalds, qui est entre autres l'homme à l'origine de Linux. Il se distingue par sa rapidité et sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.	Kernel de Linux, Debian, VLC, Android, Gnome, Qt...

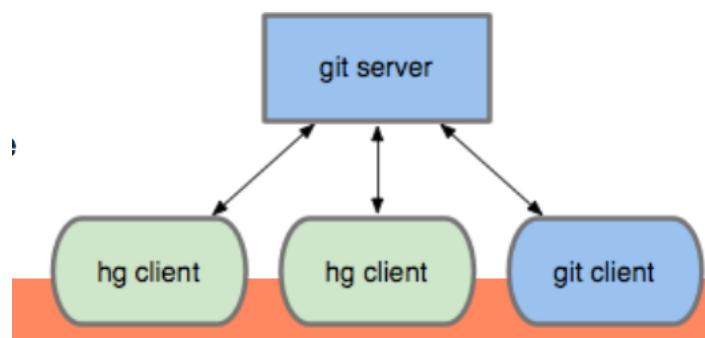
<https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git>

Git : le fil d'Ariane de vos projets, pilier des forges modernes - Claire MOUTON

# Hg-Git mercurial plugin

Adds the ability to push to and pull from a Git server repository from Mercurial.

This means you can collaborate on Git based projects from Mercurial, or use a Git server as a collaboration point for a team with developers using both Git and Mercurial.

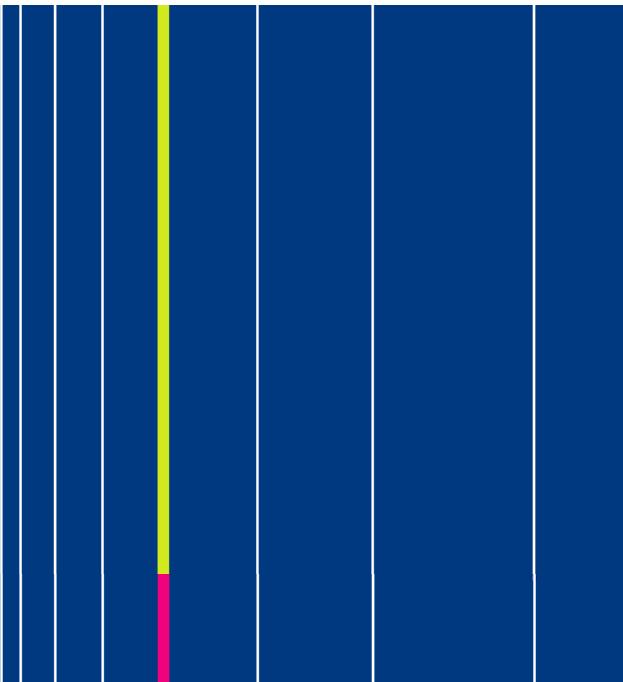


<http://hg-git.github.io/>

<https://www.mercurial-scm.org/wiki/HgGit>

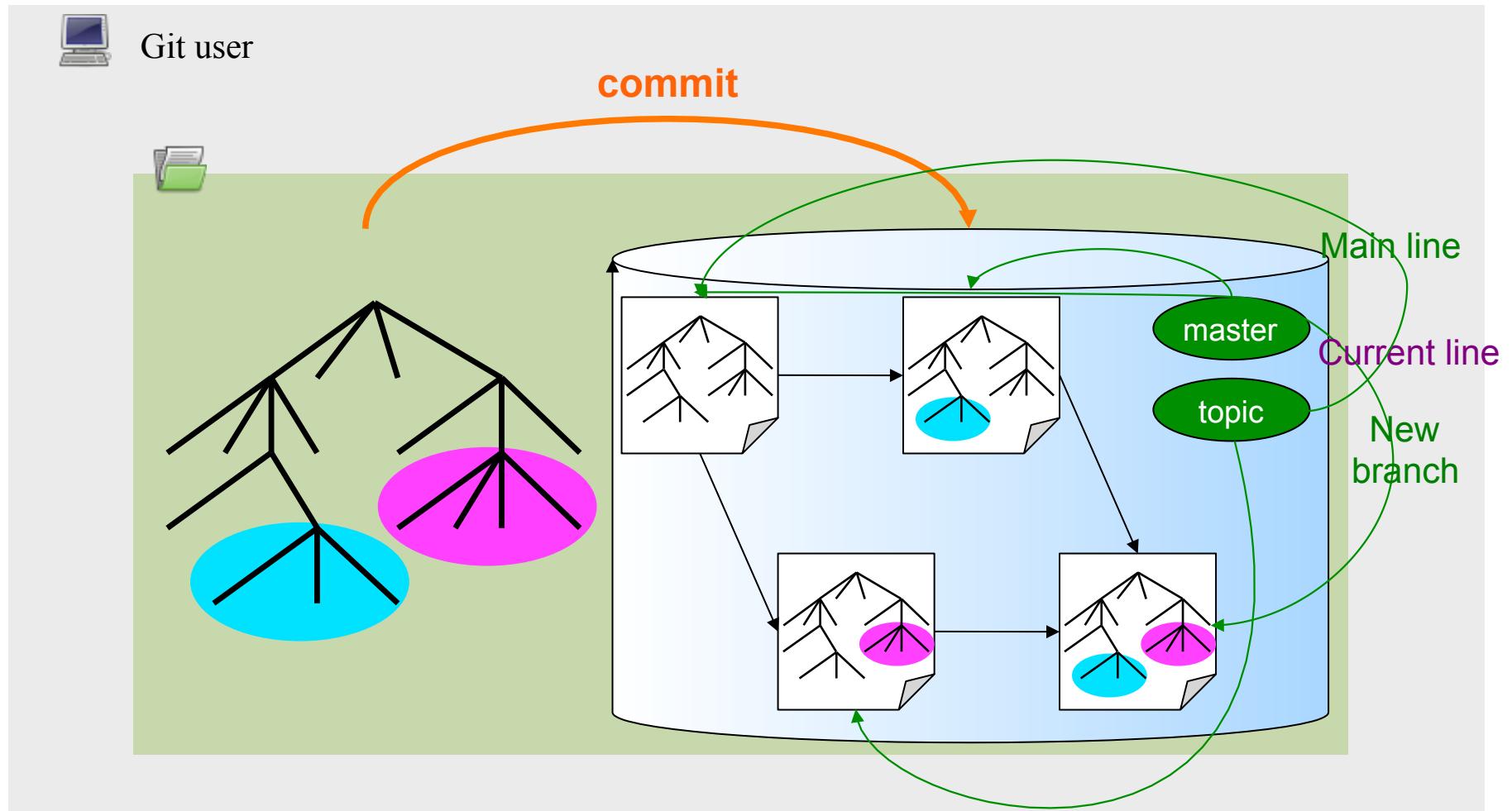
# Decentralized SCM

## > Git basic

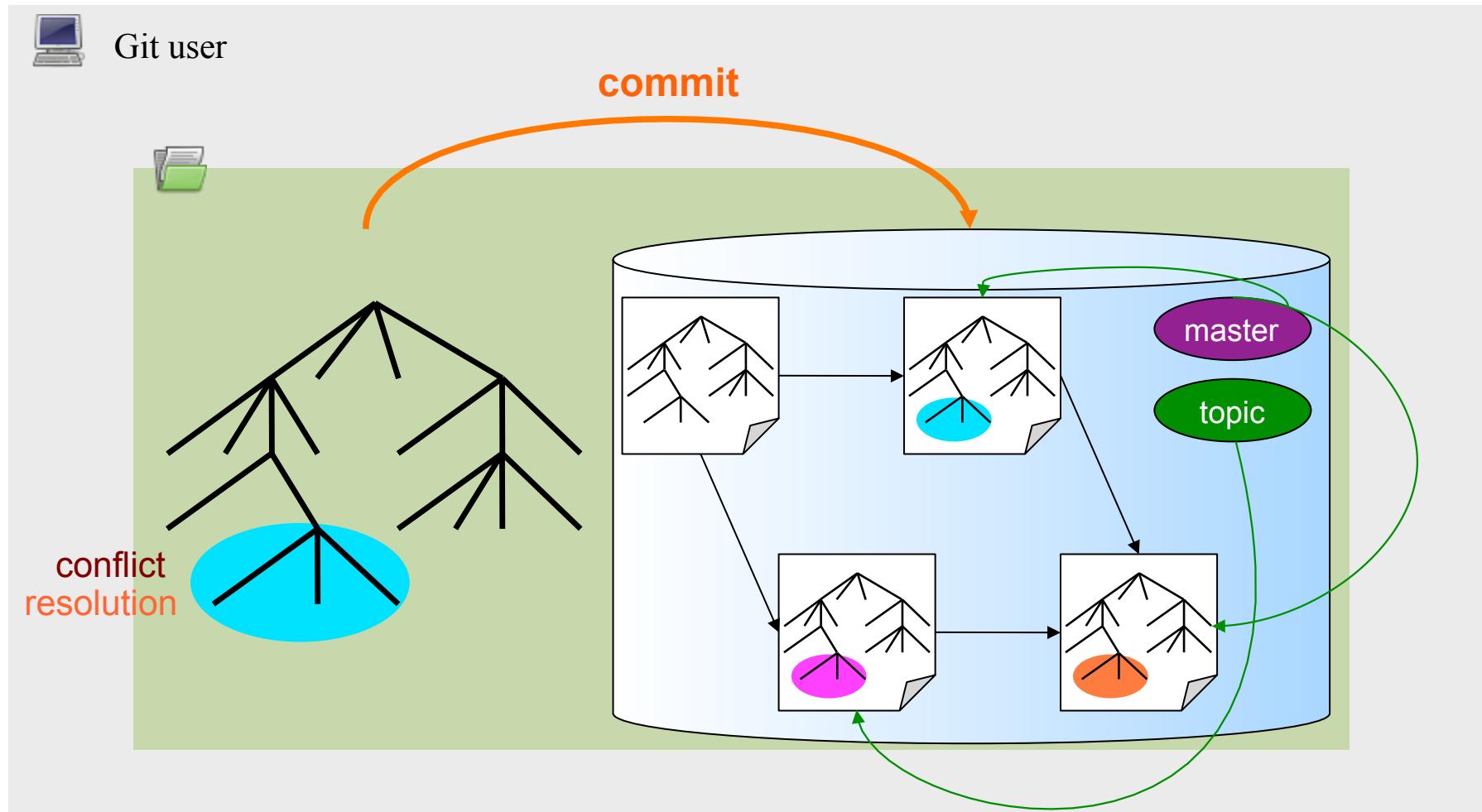
A series of vertical bars of varying heights and colors (yellow, red, white) are positioned along the left edge of the slide.

*Creatis*

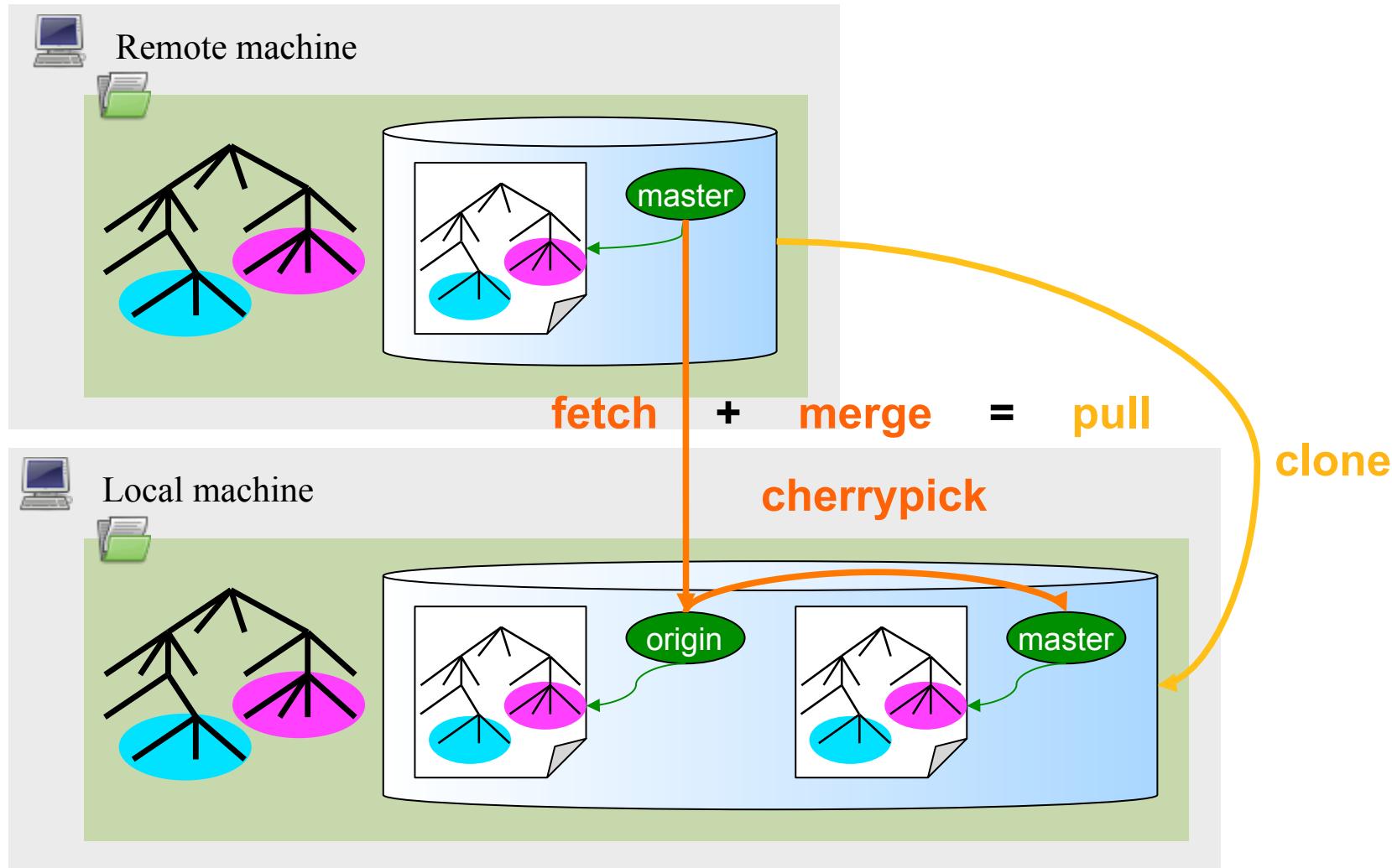
## Git > Local operations



# Git > Conflict

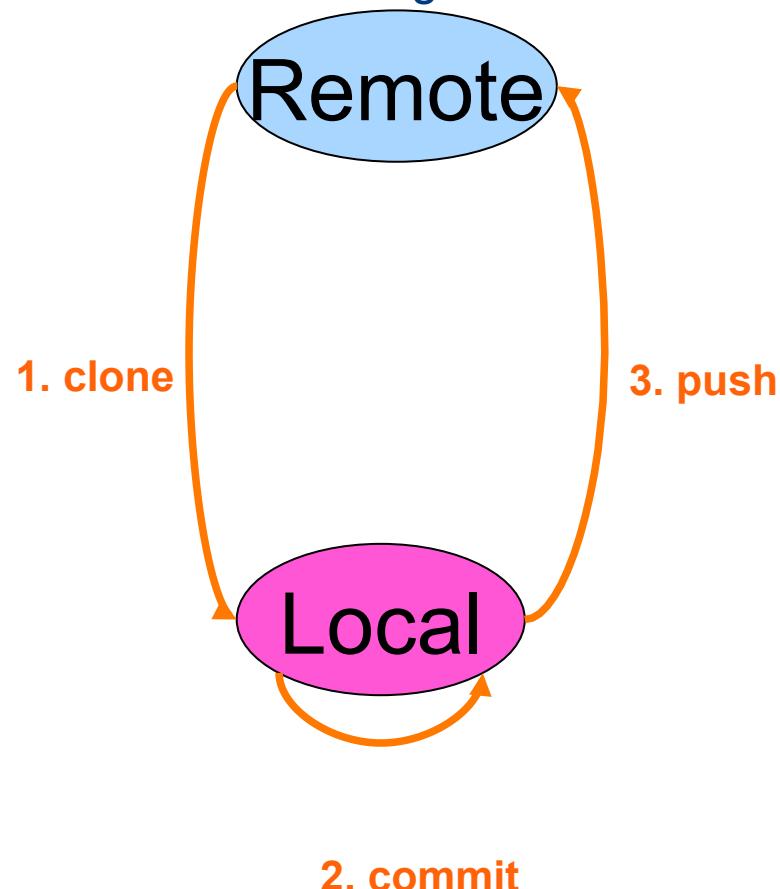


## Git > Distant operations



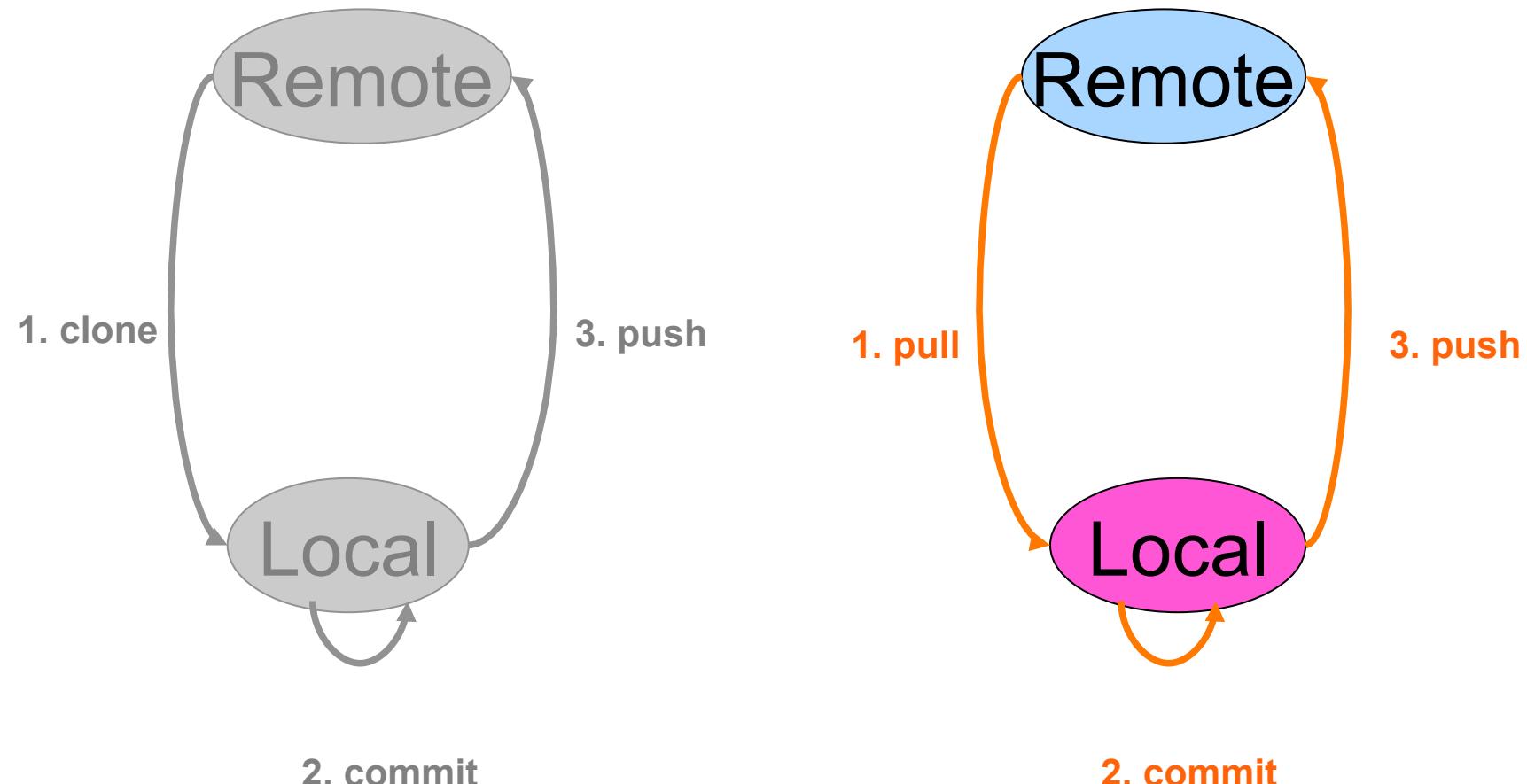
# Git > Workflow

> Classic working



# Git > Workflow

> Classic working





# Decentralized SCM

## > Git synthesis



*Creatis*

# Git

- + Good visualization and branch handling, available locally:  
saving of intermediary states, working on distinct features,  
experiments, ...
- + Easy merge of branches, cherrypick for one single commit
- + All operations available locally in a sandbox, except push and pull
- + Flexible: undo / modify commits, locally before sharing
- + Hooks pre/post (commit, update ...)
- + Bisection: finds the guilty commit (the one that introduced the bug)
- + Efficient on big projects (Linux kernel)
- + Available on Linux, OS X and Windows

# Around Git

## GUIs

- Built-in GUI tools (Windows, Linux, MacOS):
  - for browsing : [gitk](#)
  - for committing : [git-gui](#)
- [qgit](#) (a repository browser written in C++ using Qt)
- [SmartGit](#) (Windows, Linux, MacOS)
- [GitKraken](#) (Windows, Linux, MacOS)
- [SourceTree](#) (Windows, MacOS)
- [TortoiseGit](#) via Putty (Windows)
- [GitHub Desktop](#) (Windows, MacOS)
- [SourceTree](#) (Windows, MacOS)
- <https://git-scm.com/downloads/guis/>

## Useful links

- <http://git-scm.com/>

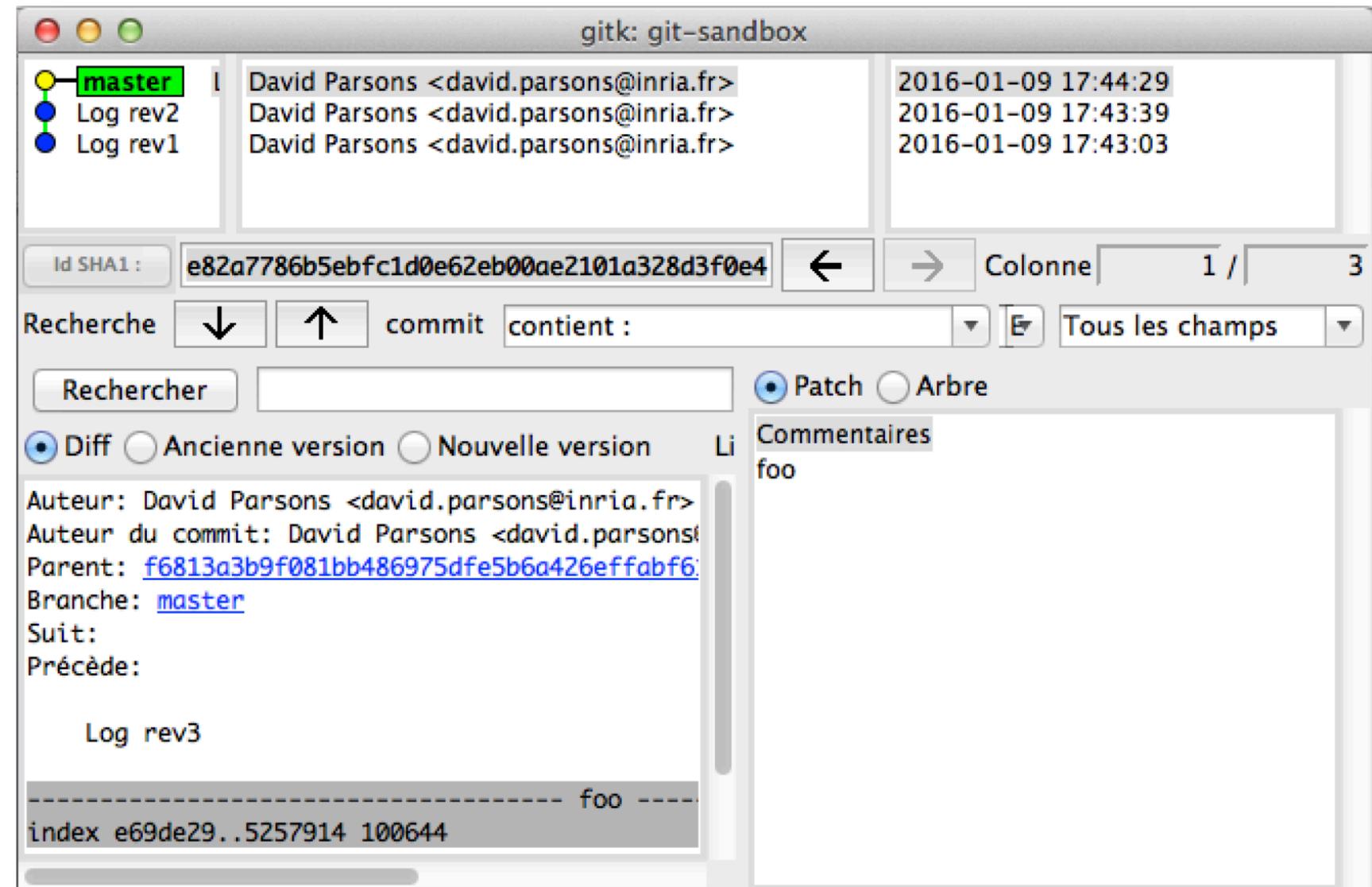
## Plugins for IDE

Vim  
Emacs  
Eclipse  
Visual Studio  
TextMate  
...

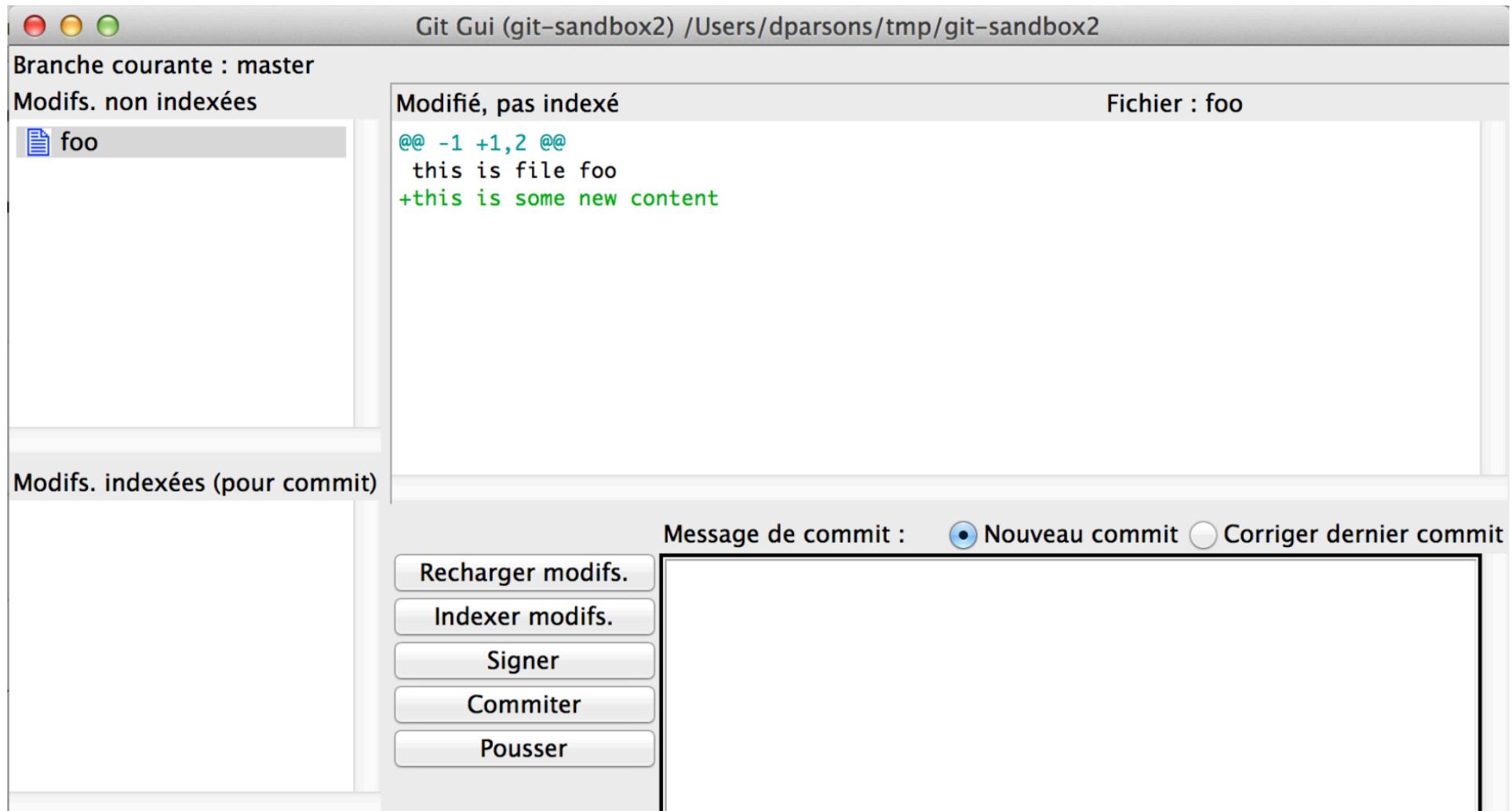
## Users

Linux kernel  
Wine  
X.org  
Android  
Kitware

# Around Git > *gitk*



# Around Git > git-gui

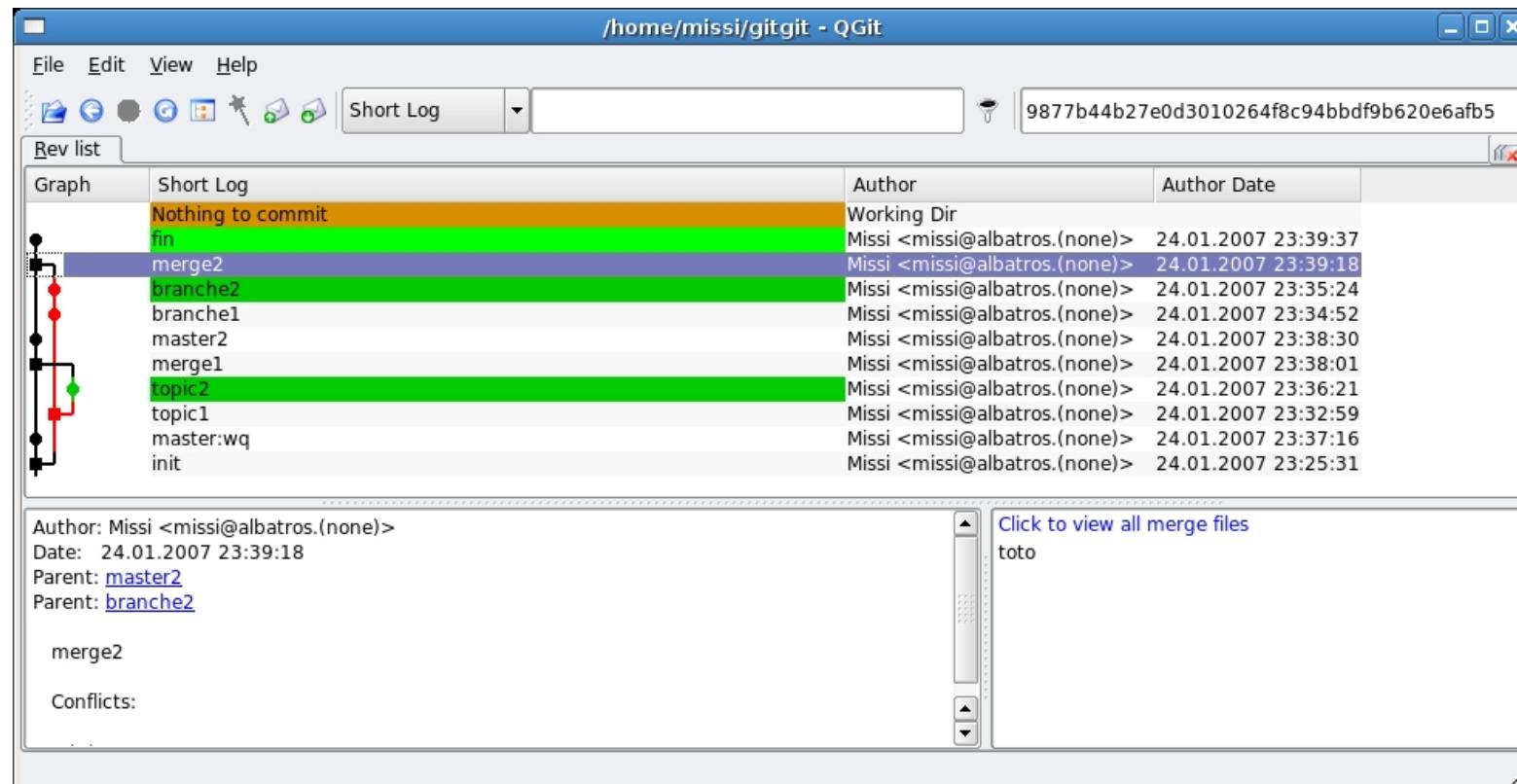


# Around Git > qgit

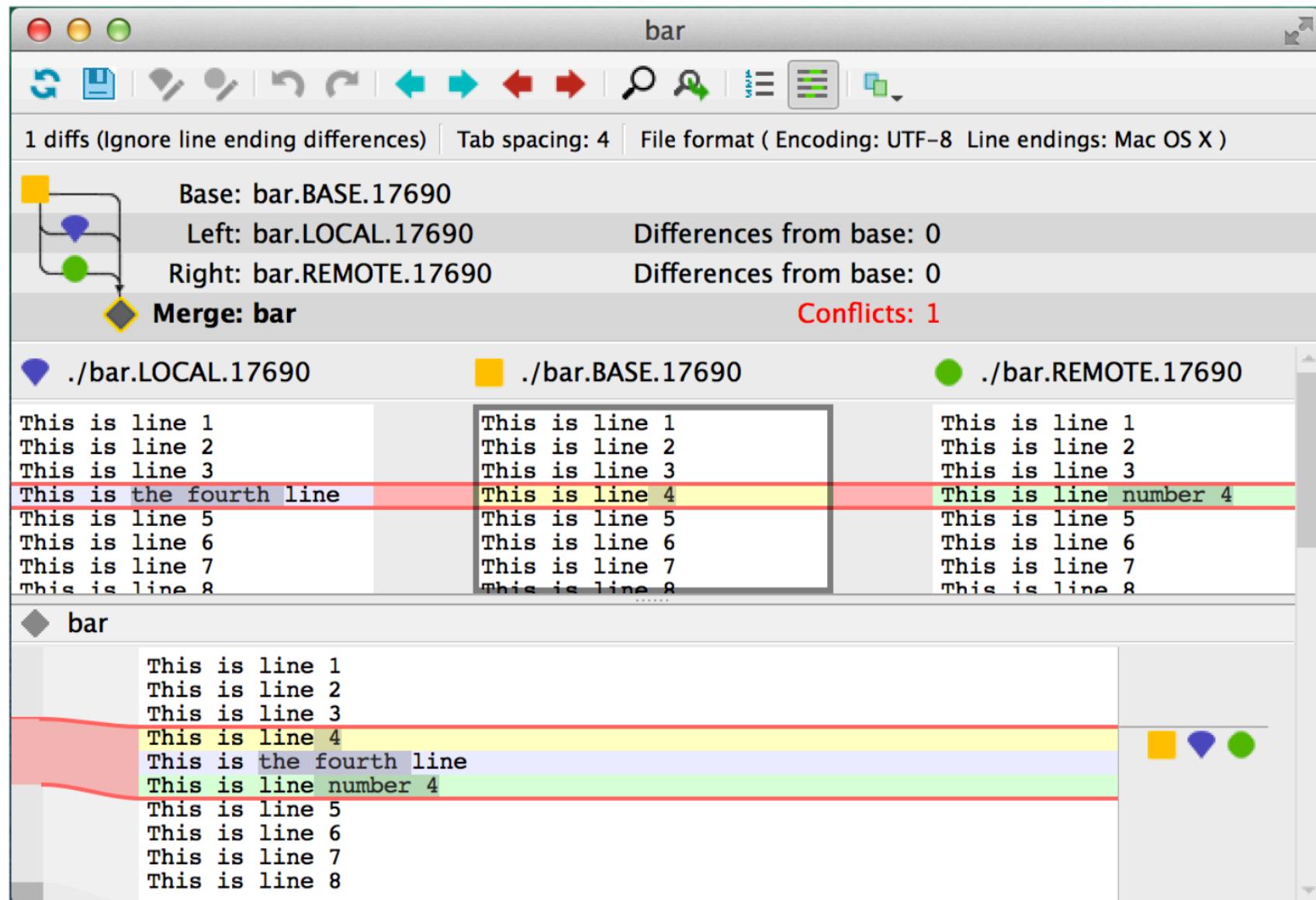
Browse revisions history

View patch content and changed files

Graphically follow different development branches

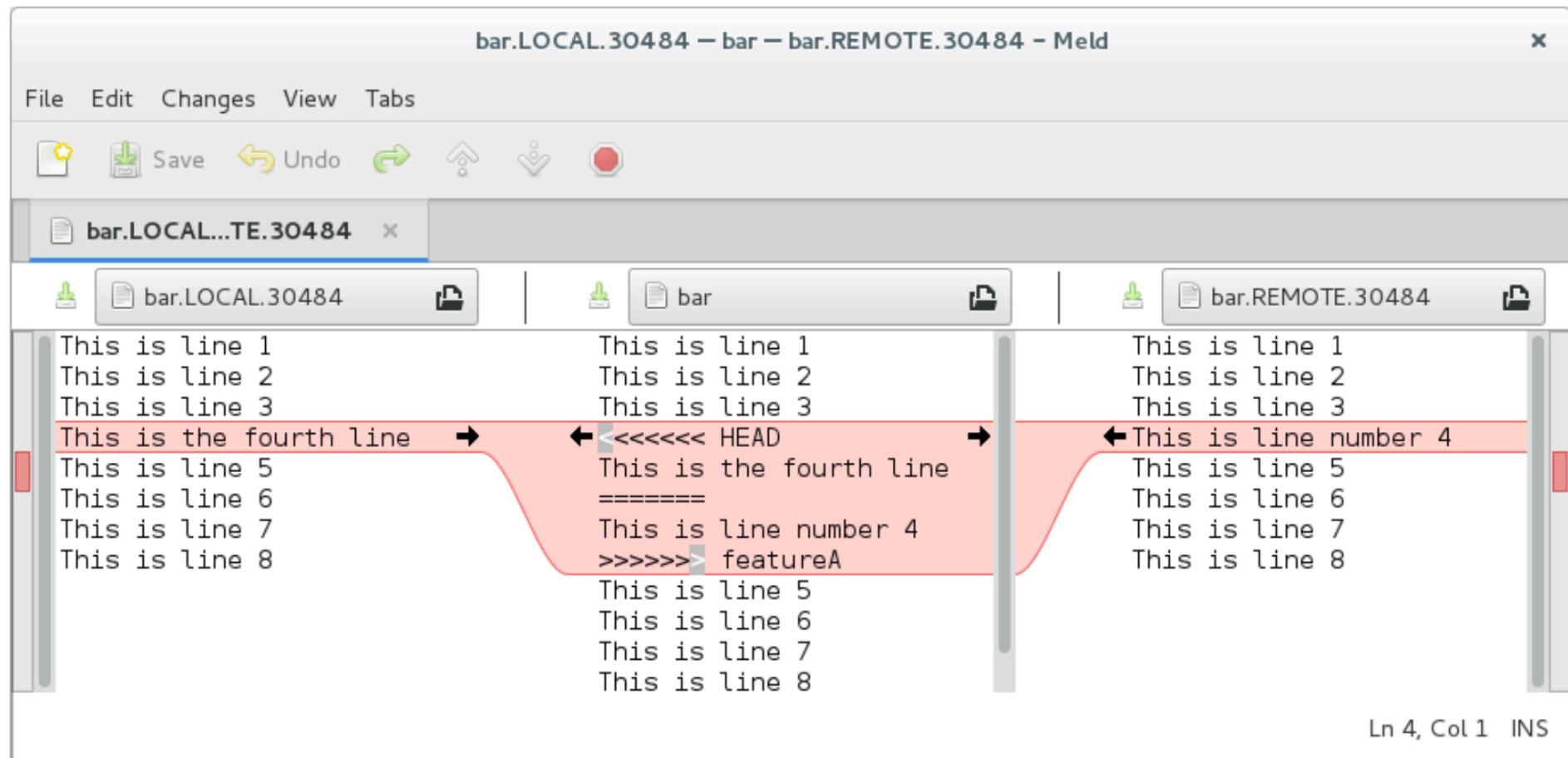


# Around Git > mergetool



# Around Git > configuring mergetool

```
$ git config --global merge.tool meld
$ git config --global mergetool.meld.cmd 'meld $LOCAL $MERGED $REMOTE'
$ git config --global mergetool.meld.trustExitCode false
$
```



## Around Git > *online visualization of your repository*

- gitweb by default
- gitolite
- gitblit
- gitbucket
- gogs
- kallithea

# Around Git > *online visualization of your repository*

## Gitolite CREATIS example : The list of projects

The screenshot shows a web-based Git interface for the repository `git://git.creatis.insa-lyon.fr /`. The interface includes a search bar with a 're' checkbox and a 'Search' button. A red link labeled 'List all projects' is visible. Below the search area, there is a table listing four projects:

Project	Description	Owner	Last Change	Actions
CreaPhase.git	Propagation-based phase contra...	Loriane Weber	2 months ago	<a href="#">summary</a>   <a href="#">shortlog</a>   <a href="#">log</a>   <a href="#">tree</a>
FrontAlgorithms.git	Generic implementation of...	Maciej Orkisz	2 days ago	<a href="#">summary</a>   <a href="#">shortlog</a>   <a href="#">log</a>   <a href="#">tree</a>
GRIDA.git	Grid Data Management Agent	Sorina Pop	No commits	<a href="#">summary</a>   <a href="#">shortlog</a>   <a href="#">log</a>   <a href="#">tree</a>
bbtk.git	Black Box Tool Kit	Eduardo Davila	6 days ago	<a href="#">summary</a>   <a href="#">shortlog</a>   <a href="#">log</a>   <a href="#">tree</a>

# Around Git > *online visualization of your repository*

Gitolite CREATIS example :  
The log for one project

The screenshot shows a web-based Git interface for the repository `git://git.creatis.insa-lyon.fr / FrontAlgorithms.git`. The top navigation bar includes links for `summary`, `shortlog`, `log`, `commit`, `commitdiff`, and `tree`. A search bar is available with a `commit` button and a `re` checkbox. The repository details section includes:  
description: Generic implementation of front propagation algorithms with some extra features  
owner: Maciej Orkisz  
last change: Tue, 22 Nov 2016 00:02:28 +0100 (18:02 -0500)  
The `shortlog` section displays the following commit history:

Date	Author	Message	Branch	Actions
5 days ago	Leonardo Flórez...	...	master	<a href="#">commit</a>   <a href="#">commitdiff</a>   <a href="#">tree</a>   <a href="#">snapshot</a>
6 days ago	Leonardo Flórez...	...		<a href="#">commit</a>   <a href="#">commitdiff</a>   <a href="#">tree</a>   <a href="#">snapshot</a>
6 days ago	Leonardo Flórez...	...		<a href="#">commit</a>   <a href="#">commitdiff</a>   <a href="#">tree</a>   <a href="#">snapshot</a>
2016-11-11	Leonardo Flórez...	...		<a href="#">commit</a>   <a href="#">commitdiff</a>   <a href="#">tree</a>   <a href="#">snapshot</a>

At the bottom, a decorative footer features a timeline of vertical bars and the text: "Git : le fil d'Ariane de vos projets, pilier des forges modernes - Claire MOUTON".

# Around Git > *online visualization of your repository*

## Gitolite CREATIS example : A commit

[git://git.creatis.insa-lyon.fr / FrontAlgorithms.git](git://git.creatis.insa-lyon.fr/FrontAlgorithms.git) / commit 

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)  
 (parent: [86a6d5d](#)) | [patch](#)

commit  [? search:](#)   re

... [master](#)

---

author	Leonardo Flórez-Valencia <florez-l@javeriana.edu.co>	
	Tue, 22 Nov 2016 00:02:28 +0100 (18:02 -0500)	
committer	Leonardo Flórez-Valencia <florez-l@javeriana.edu.co>	
	Tue, 22 Nov 2016 00:02:28 +0100 (18:02 -0500)	
commit	<a href="#">3e69c5942ef8dd71c4e25da906eac97ffb63a79d</a>	
tree	<a href="#">9226bc46572e17f1c1c42a02c0d3f3b90b1c2b23</a>	<a href="#">tree</a>   <a href="#">snapshot</a>
parent	<a href="#">86a6d5df2aa1aa5292a5fa851d98bfc13939bdf3</a>	<a href="#">commit</a>   <a href="#">diff</a>

---

...

---

<a href="#">lib/CMakeLists.txt</a>	<a href="#">diff</a>   <a href="#">blob</a>   <a href="#">history</a>
<a href="#">lib/fpaInstances/CMakeLists.txt</a>	<a href="#">diff</a>   <a href="#">blob</a>   <a href="#">history</a>
<a href="#">plugins/CMakeLists.txt</a>	<a href="#">diff</a>   <a href="#">blob</a>   <a href="#">history</a>

||| Git : le fil d'Ariane de vos projets, pilier des forges modernes - Claire MOUTON |||

# Around Git > *online visualization of your repository*

## Gitolite CREATIS example : A commitdiff

[git://git.creatis.insa-lyon.fr / FrontAlgorithms.git](git://git.creatis.insa-lyon.fr/FrontAlgorithms.git) / commitdiff

summary | shortlog | log | commit | commitdiff | tree  
[raw](#) | [patch](#) | [inline](#) | [side by side](#) (parent: [86a6d5d](#))

... master

author Leonardo Flórez-Valencia <florez-l@javeriana.edu.co>  
 Tue, 22 Nov 2016 00:02:28 +0100 (18:02 -0500)  
 committer Leonardo Flórez-Valencia <florez-l@javeriana.edu.co>  
 Tue, 22 Nov 2016 00:02:28 +0100 (18:02 -0500)

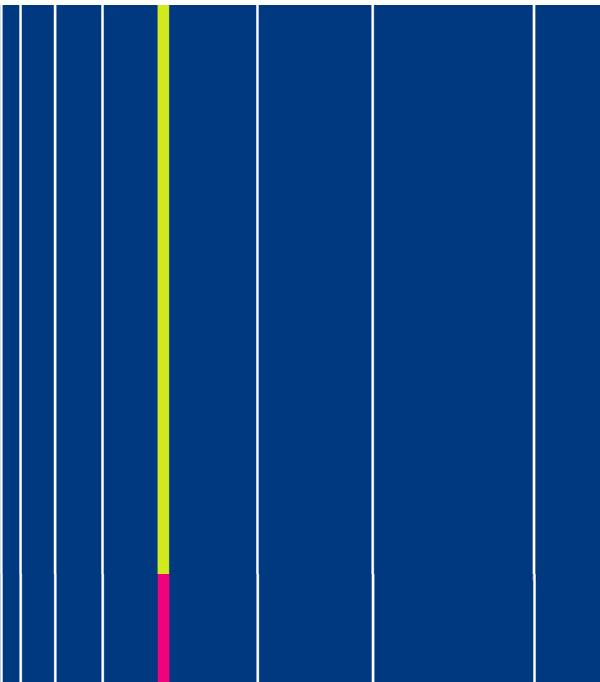
---

lib/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)  
 lib/fpaInstances/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)  
 plugins/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)

```
diff --git a/lib/CMakeLists.txt b/lib/CMakeLists.txt
index d65d98b..20f87ed 100644 (file)
--- a/lib/CMakeLists.txt
+++ b/lib/CMakeLists.txt
@@ -8,6 +8,8 @@ CompileLibFromDir(fpa SHARED fpa)
 ## == Build instances for cpPlugins ==
 ####
 
-SUBDIRS(fpaInstances)
+IF(USE_cpPlugins)
+  SUBDIRS(fpaInstances)
+ENDIF(USE_cpPlugins)

## eof - $RCSfile$
```

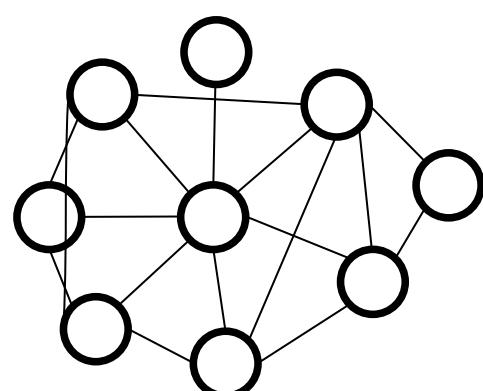
# Conclusion

A series of vertical bars of varying heights and colors (yellow, red, white) are positioned along the left edge of the slide.

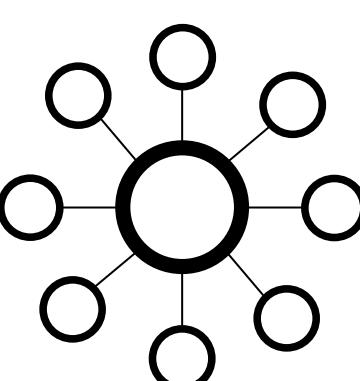
Creatis

# Conclusion

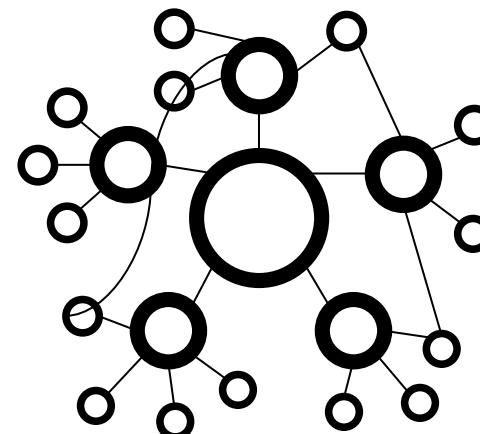
- A decentralized SCM remains a **tool**
  - No default usage policy
  - Policy to be defined
    - From centralized to decentralized
    - Pull-only vs shared-push



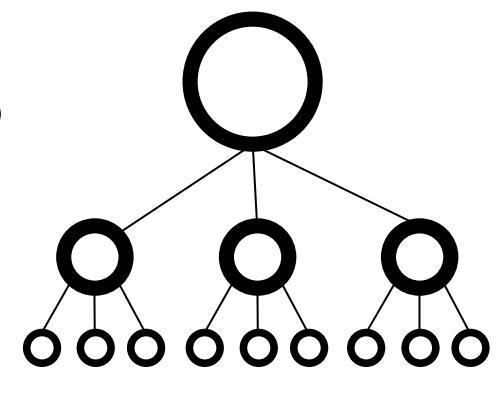
Anarchic



Centralized



Linux kernel model

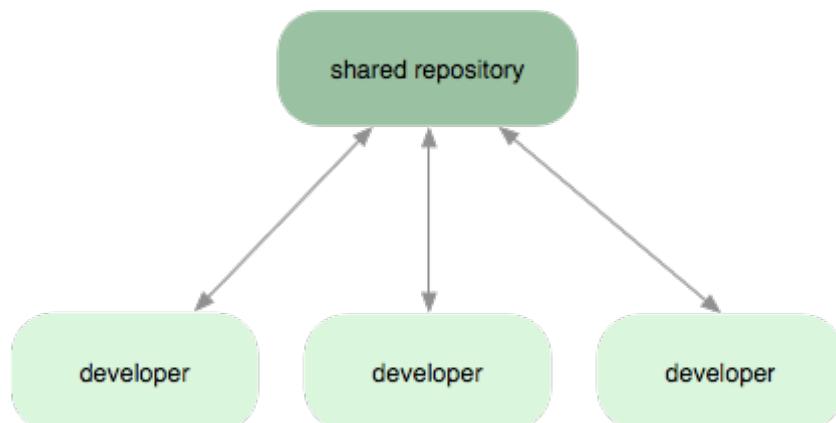


Branch by functionality  
or  
Release train

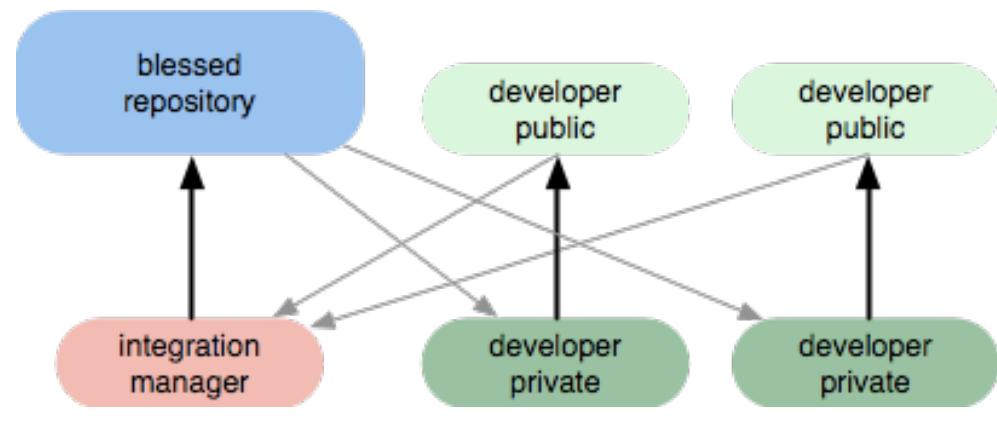
# Conclusion

- Workflow examples :
  - Centralized workflow
  - One branch per functionality
  - Workflow Gitflow (branches *master*, *develop*, *release-v\**)
  - Duplication workflow (fork to get your own public repository)

<https://fr.atlassian.com/git/tutorials/comparing-workflows/>  
<http://nvie.com/posts/a-successful-git-branching-model/>



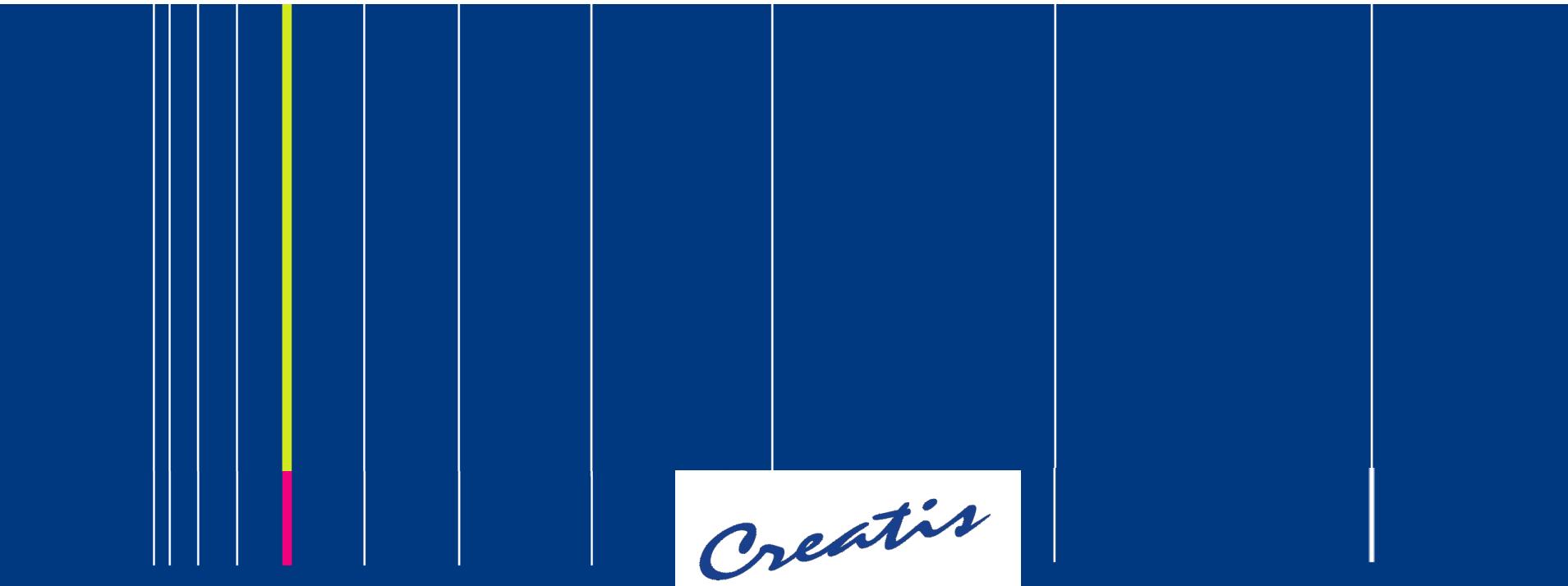
Centralized-style workflow



Integration manager workflow



# Examples and Demonstration



*Creatis*

# Demonstration with the GUI SmartGit

- History browsing
- Commit locally
- Push to remote repository
- Pull from remote repository

## Advised practice > *How to commit properly?*

# Advised practice > *How to commit properly?*

- A commit is not a backup!
- A commit should be atomic: it corresponds to one specific feature (a bug correction, a new function...).
- Before a commit:
  - Clean the code (explicit variable names, comments, Doxygen documentation, ...).
  - Check the compilation and the execution.
  - Pass the tests.
  - Git diff to choose what you are committing.
- Commit message:
  - Concise and precise. For example:
    - # IssueNb Added the method FunctionName to the class ClassName.
    - # IssueNb Removed the file BadClass.c.
    - # IssueNb Fixed a bug in Class::Method : the method performed bad access.

# Advised practice

- Commits should be atomic, with pertinent messages
- Synchronize frequently to avoid conflicts
- Bug / task manager allowing to link a commit to an issue

# Classic workflow

80

Every one has a local repository on his machine, a reference repository exists on a server.

*git clone repository\_URL* – to retrieve a module

*git pull origin master* – to retrieve the latest version from the server to update your local version

*git status* – is recommended to see the status of your local repository and the modifications ready for the commit (i.e. in the index)

*git diff* – to check the current modifications since the last commit

*git add modified\_file* – to add a new file to the module

*git commit -a -m"Appropriate message describing the fixed bug or the feature added."* – to create a local commit for all added files

*git push origin master* – to post your local commits to a remote repository

*git log* – to view the history with commit messages and authors

*git command --help* – integrated help

# 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/ecitory/`

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<sup>\*</sup> : 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`

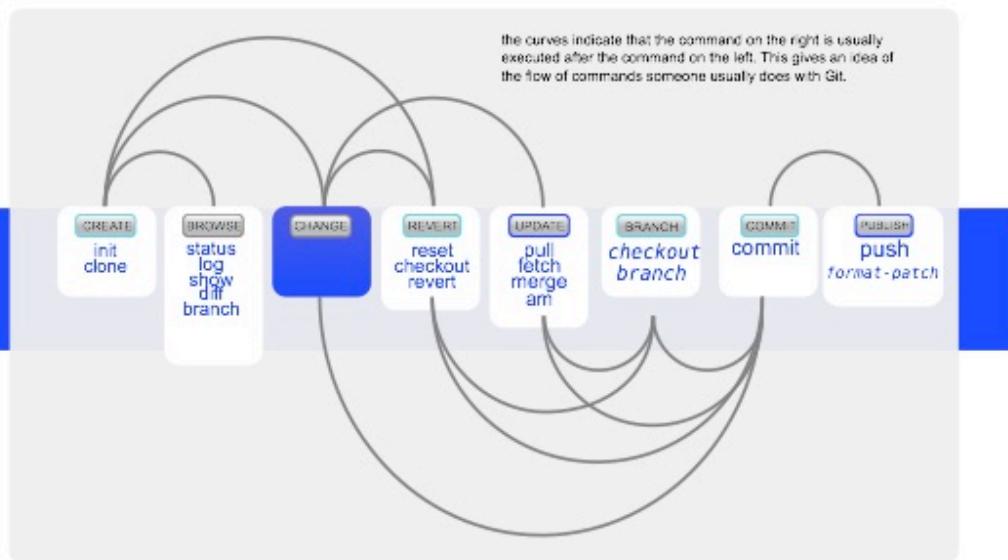
Merge branch1 into branch2  
`git checkout $branch2  
git merge $branch1`

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`  
`git bisect good $id`  
`git bisect bad $id`  
(to start)  
(\$id is the last working version)  
(\$id is a broken version)

`git bisect bad/good`  
`git bisect visualize`  
`git bisect reset`  
(to mark it as bad or good)  
(to launch gitk and mark it)  
(once you're done)

Check for errors and cleanup repository

`git fsck`  
`git gc --prune`

Search working directory for foo()  
`git grep "foo()"`

### To view the merge conflicts

`git diff` (compare 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`

## 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

## Configuration

```
git config --global user.name "nom"
git config --global user.email "prenom.nom@cnrs.fr"
git config --global color.ui auto
git config --global credential.helper cache
git config --global http.postBuffer 524288000
• Vérifier la configuration
git config --global -l
```

## Créer un dépôt

- Créer un dépôt vide  
`git init projet`
- Créer un dépôt en clonant un dépôt distant  
`git clone url`
- Créer un dépôt local dans un répertoire local existant  
`cd repertoire_projet`  
`git init git add -A`

## Ignorer des fichiers

- Créer la liste des fichiers à ignorer et la publier  
`git config --global core.excludefiles **/*.log`  
Modifier le fichier `.gitignore`  
`git add .gitignore`  
`git commit -m "Partage des fichiers à ignorer"`
- Afficher la liste de tous les fichiers ignorés  
`git ls-files --other --ignored --exclude-standard`

## Historique

- Afficher tous les commits (format par défaut ou court)  
`git log`  
`git log --pretty=--short`
- Afficher les  $x$  derniers commits  
`git log -n x`
- Afficher les commits d'un fichier ou d'un répertoire  
`git log fichier`  
`git log repertoire/`
- Afficher des statistiques pour chaque fichier modifié  
`git log --stat`
- Afficher le contenu d'un commit  
`git show id_commit`

## Modifications locales

- Annuler les modifications réalisées dans un fichier  
`git checkout -- fichier`  
`git reset [--mixed] HEAD fichier`
- Ajouter des fichiers au prochain commit  
`git add fichier1 fichier2 fichier3`
- Enlever un fichier du prochain commit  
`git rm --cached fichier`
- Supprimer un fichier  
`git rm fichier`
- Supprimer récursivement les fichiers d'un répertoire  
`git rm repertoire/ -r`
- Renommer un fichier  
`git mv fichier nouveau_nom`
- Déplacer un fichier  
`git mv fichier destination/`
- Afficher l'état des fichiers nouveaux ou modifiés  
`git status`
- Afficher les modifications des fichiers suivis modifiés  
`git diff`
- Afficher les modifications du prochain commit  
`git diff --cached`
- Effectuer un commit  
`git commit`  
`git commit -a (ajouter automatiquement les fichiers)`  
`git commit -m "Message du commit"`
- Modifier le dernier commit  
`git commit --amend`
- Etiqueter le dernier commit  
`git tag nom_tag`
- Annuler les  $n$  derniers commit  
`git revert HEAD (dernier commit)`  
`git revert HEAD~ (2 derniers commit)`  
`git revert HEAD~2 (3 derniers commit)`
- Retourner à la version du dernier commit  
(Supprime les nouveaux fichiers et les modifications)  
**ATTENTION : Cette opération ne peut pas être annulée**  
`git reset --hard HEAD`

## Branches

- Afficher la liste des branches  
`git branch`
- Créer une nouvelle branche  
`git branch nom_branch`
- Basculer sur une branche  
`git checkout nom_branch`
- Fusionner une branche dans la branche courante  
`git merge nom_branch`
- Supprimer localement une branche  
`git branch -d nom_branch`
- Afficher les différences entre deux branches  
`git diff nom_branch1...nom_branch2`

## Dépôts distants

- Afficher la liste des dépôts déclarés  
`git remote -v`
- Afficher des informations sur un dépôt  
`git remote show nom_depot` (ex:`origin`)
- Déclarer un dépôt  
`git remote add chemin|url`
- Déclarer le dépôt origin  
`git remote add origin url`
- Récupérer les données d'un dépôt déclaré  
`git fetch nom_depot`
- Récupérer les données de la branche d'un dépôt et fusionner dans la branche courante  
`git pull [nom_depot] [nom_branch_distant]`
- Publier les modifications locales d'une branche  
`git push [nom_depot] [nom_branch_locale]`
- Supprimer une branche dans un dépôt déclaré  
`git push nom_depot :nom_branch_distant`  
`git push nom_depot --delete nom_branch_distant`
- Publier les informations de tags  
`git push nom_depot --tags`



Git est le système de gestion de version décentralisé open source qui facilite les activités GitHub sur votre ordinateur.  
Cet aide-mémoire permet un accès rapide aux instructions des commandes Git les plus utilisées.

### INSTALLER GIT

GitHub fournit des clients desktop qui incluent une interface graphique pour les manipulations les plus courantes et une "an automatically updating command line edition of Git" pour les scénarios avancés.

#### GitHub pour Windows

<https://windows.github.com>

#### GitHub pour Mac

<https://mac.github.com>

Les distributions de Git pour Linux et les systèmes POSIX sont disponibles sur le site web officiel de Git SCM.

#### Git pour toutes les plate-formes

<http://git-scm.com>

### CONFIGURATION DES OUTILS

Configurer les informations de l'utilisateur pour tous les dépôts locaux

**\$ git config --global user.name "[nom]"**

Définit le nom que vous voulez associer à toutes vos opérations de commit

**\$ git config --global user.email "[adresse email]"**

Définit l'email que vous voulez associer à toutes vos opérations de commit

**\$ git config --global color.ui auto**

Active la colorisation de la sortie en ligne de commande

### CRÉER DES DÉPÔTS

Démarrer un nouveau dépôt ou en obtenir un depuis une URL existante

**\$ git init [nom-du-projet]**

Crée un dépôt local à partir du nom spécifié

**\$ git clone [url]**

Télécharge un projet et tout son historique de versions

### EFFECTUER DES CHANGEMENTS

Consulter les modifications et effectuer une opération de commit

**\$ git status**

Liste tous les nouveaux fichiers et les fichiers modifiés à commiter

**\$ git diff**

Montre les modifications de fichier qui ne sont pas encore indexées

**\$ git add [fichier]**

Ajoute un instantané du fichier, en préparation pour le suivi de version

**\$ git diff --staged**

Montre les différences de fichier entre la version indexée et la dernière version

**\$ git reset [fichier]**

Enlève le fichier de l'index, mais conserve son contenu

**\$ git commit -m "[message descriptif]"**

Enregistre des instantanés de fichiers de façon permanente dans l'historique des versions

### GROUPER DES CHANGEMENTS

Nommer une série de commits et combiner les résultats de travaux terminés

**\$ git branch**

Liste toutes les branches locales dans le dépôt courant

**\$ git branch [nom-de-branche]**

Crée une nouvelle branche

**\$ git checkout [nom-de-branche]**

Bascule sur la branche spécifiée et met à jour le répertoire de travail

**\$ git merge [nom-de-branche]**

Combine dans la branche courante l'historique de la branche spécifiée

**\$ git branch -d [nom-de-branche]**

Supprime la branche spécifiée



**GitHub**

## AIDE-MÉMOIRE GITHUB GIT

Git est le système de gestion de version décentralisé open source qui facilite les activités GitHub sur votre ordinateur. Cet aide-mémoire permet un accès rapide aux instructions des commandes Git les plus utilisées.

### INSTALLER GIT

Github fournit des clients desktop qui incluent une interface graphique pour les manipulations les plus courantes et une "an automatically updating command line edition of Git" pour les scénarii avancés.

**GitHub pour Windows**  
<https://windows.github.com>

**GitHub pour Mac**  
<https://mac.github.com>

Les distributions de Git pour Linux et les systèmes POSIX sont disponibles sur le site web officiel de Git SCM.

**Git pour toutes les plate-formes**  
<http://git-scm.com>

### CONFIGURATION DES OUTILS

Configurer les informations de l'utilisateur pour tous les dépôts locaux

```
$ git config --global user.name "[nom]"
Définit le nom que vous voulez associer à toutes vos opérations de commit
```

```
$ git config --global user.email "[adresse email]"
Définit l'email que vous voulez associer à toutes vos opérations de commit
```

```
$ git config --global color.ui auto
Active la colorisation de la sortie en ligne de commande
```

### CRÉER DES DÉPÔTS

Démarrer un nouveau dépôt ou en obtenir un depuis une URL existante

```
$ git init [nom-du-projet]
Crée un dépôt local à partir du nom spécifié
```

```
$ git clone [url]
Télécharge un projet et tout son historique de versions
```

### EFFECTUER DES CHANGEMENTS

Consulter les modifications et effectuer une opération de commit

```
$ git status
Liste tous les nouveaux fichiers et les fichiers modifiés à commiter
```

```
$ git diff
Montre les modifications de fichier qui ne sont pas encore indexées
```

```
$ git add [fichier]
Ajoute un instantané du fichier, en préparation pour le suivi de version
```

```
$ git diff --staged
Montre les différences de fichier entre la version indexée et la dernière version
```

```
$ git reset [fichier]
Enlève le fichier de l'index, mais conserve son contenu
```

```
$ git commit -m "[message descriptif]"
Enregistre des instantanés de fichiers de façon permanente dans l'historique des versions
```

### GROUER DES CHANGEMENTS

Nommer une série de commits et combiner les résultats de travaux terminés

```
$ git branch
Liste toutes les branches locales dans le dépôt courant
```

```
$ git branch [nom-de-branche]
Crée une nouvelle branche
```

```
$ git checkout [nom-de-branche]
Bascule sur la branche spécifiée et met à jour le répertoire de travail
```

```
$ git merge [nom-de-branche]
Combine dans la branche courante l'historique de la branche spécifiée
```

```
$ git branch -d [nom-de-branche]
Supprime la branche spécifiée
```

### CHANGEMENTS AU NIVEAU DES NOMS DE FICHIERS

Déplacer et supprimer des fichiers sous suivi de version

```
$ git rm [fichier]
Supprime le fichier du répertoire de travail et met à jour l'index
```

```
$ git rm --cached [fichier]
Supprime le fichier du système de suivi de version mais le préserve localement
```

```
$ git mv [fichier-nom] [fichier-nouveau-nom]
Renomme le fichier et prépare le changement pour un commit
```

### EXCLURE DU SUIVI DE VERSION

Exclure des fichiers et chemins temporaires

```
*.log
build/
temp-*
```

Un fichier texte nommé `.gitignore` permet d'éviter le suivi de version accidentel pour les fichiers et chemins correspondant aux patterns spécifiés

```
$ git ls-files --other --ignored --exclude-standard
Liste tous les fichiers exclus du suivi de version dans ce projet
```

### VÉRIFIER L'HISTORIQUE DES VERSIONS

Suivre et inspecter l'évolution des fichiers du projet

```
$ git log
Montre l'historique des versions pour la branche courante
```

```
$ git log --follow [fichier]
Montre l'historique des versions, y compris les actions de renommage, pour le fichier spécifié
```

```
$ git diff [première-branche]...[deuxième-branche]
Montre les différences de contenu entre deux branches
```

```
$ git show [commit]
Montre les modifications de métadonnées et de contenu incluses dans le commit spécifié
```

### REFAIRE DES COMMITS

Corriger des erreurs et gérer l'historique des corrections

```
$ git reset [commit]
Annule tous les commits après "[commit]", en conservant les modifications localement
```

```
$ git reset --hard [commit]
Supprime tout l'historique et les modifications effectuées après le commit spécifié
```

### ENREGISTRER DES FRAGMENTS

Mettre en suspens des modifications non finies pour y revenir plus tard

```
$ git stash
Enregistre de manière temporaire tous les fichiers sous suivi de version qui ont été modifiés ("remiser son travail")
```

```
$ git stash pop
Applique une remise et la supprime immédiatement
```

```
$ git stash list
Liste toutes les remises
```

```
$ git stash drop
Supprime la remise la plus récente
```

### SYNCHRONISER LES CHANGEMENTS

Référencer un dépôt distant et synchroniser l'historique de versions

```
$ git fetch [nom-de-dépot]
Récupère tout l'historique du dépôt nommé
```

```
$ git merge [nom-de-dépot]/[branche]
Fusionne la branche du dépôt dans la branche locale courante
```

```
$ git push [alias] [branche]
Envoye tous les commits de la branche locale vers GitHub
```

```
$ git pull
Récupère tout l'historique du dépôt nommé et incorpore les modifications
```

### GitHub Training

Formez-vous à l'utilisation de GitHub et Git. Contactez l'équipe de formation ou visitez notre site web pour connaître les dates de formation et les disponibilités pour des cours privés.

[training@github.com](mailto:training@github.com)  
[training.github.com](mailto:training.github.com)

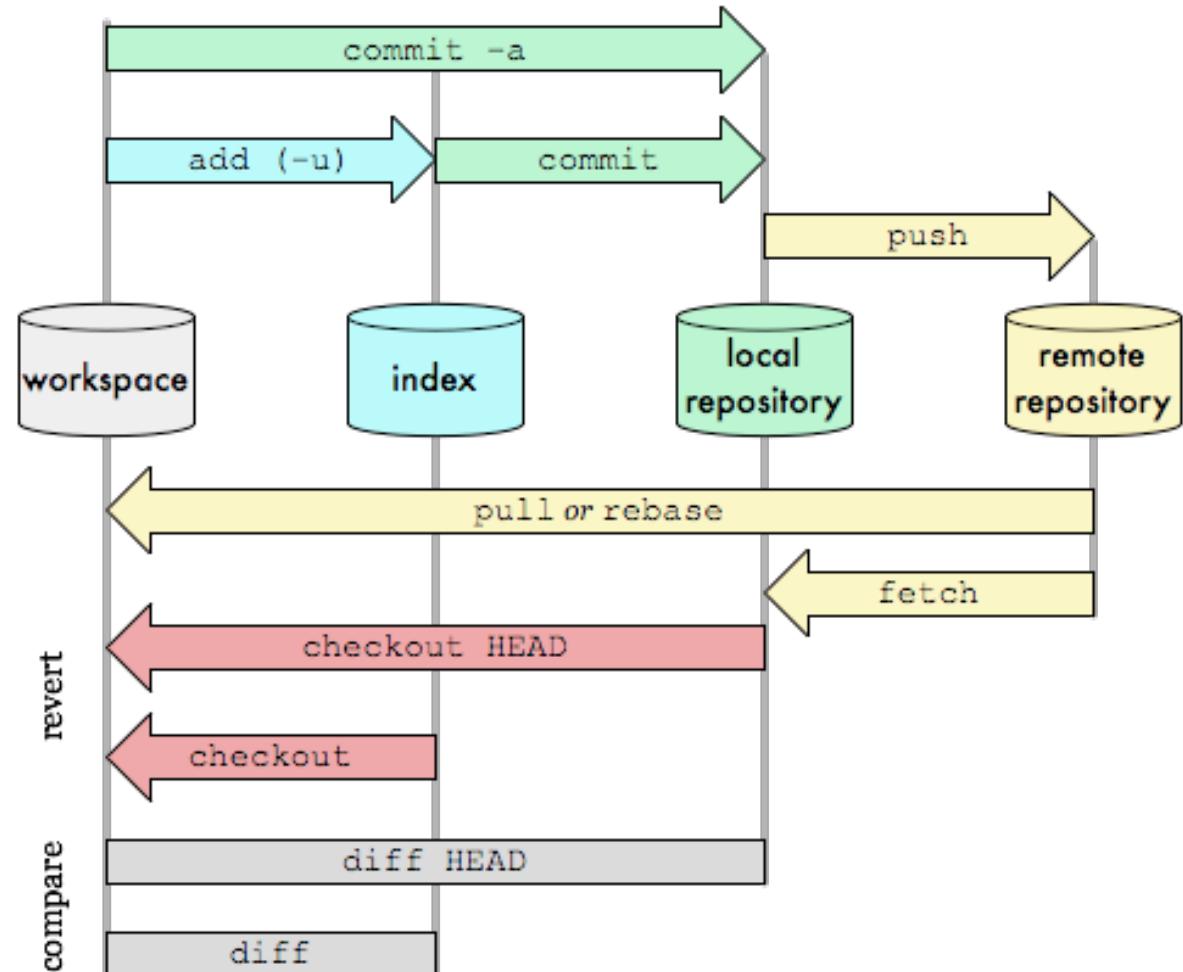
Source :

<https://services.github.com/on-demand/downloads/fr/github-git-cheat-sheet.pdf>

# Git Data Transport Commands

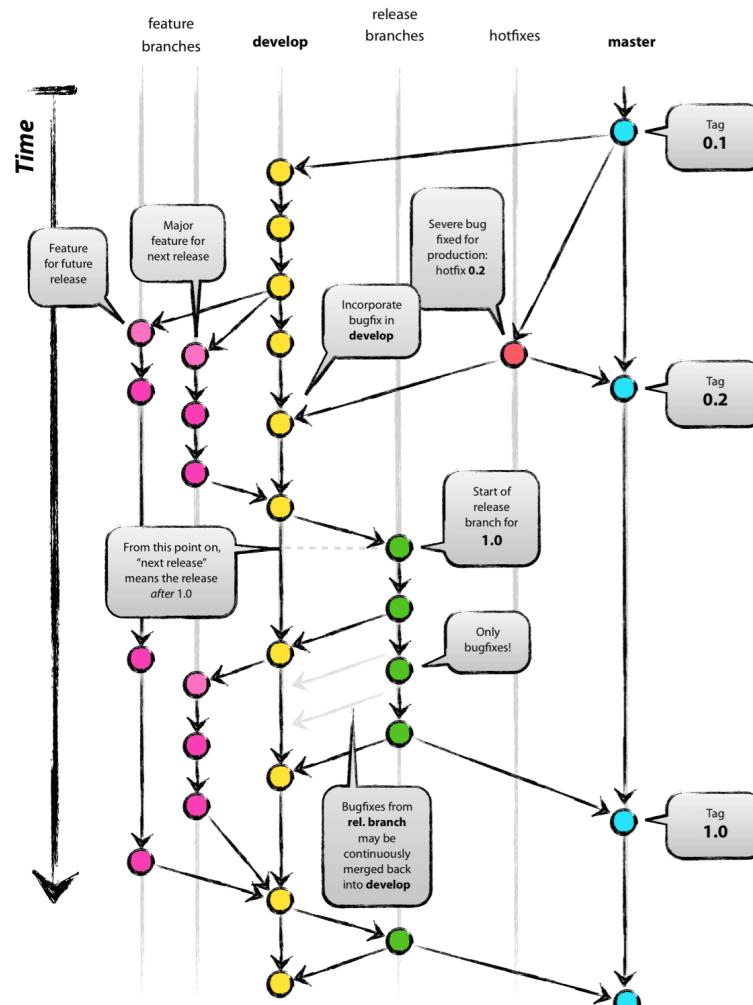
<http://osteele.com>

85



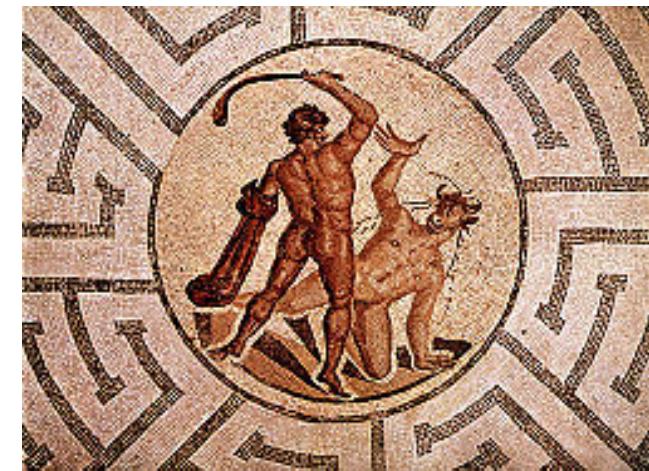
Source :

<http://stackoverflow.com/questions/3689838/difference-between-head-working-tree-index-in-git>



Source :

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



Git : le fil d'Ariane de vos projets, pilier des forges modernes - Claire MOUTON