

Vous êtes Thésée...

Vous êtes Thésée...

Ariane est votre allié !

Vous êtes Thésée...

Ariane est votre allié !

La poésie entre vous :

Vous 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!

# Menu du jour

## Matinée

Git en théorie

Git en pratique en solo

Git vs SVN via des cas d'études

## Après-midi

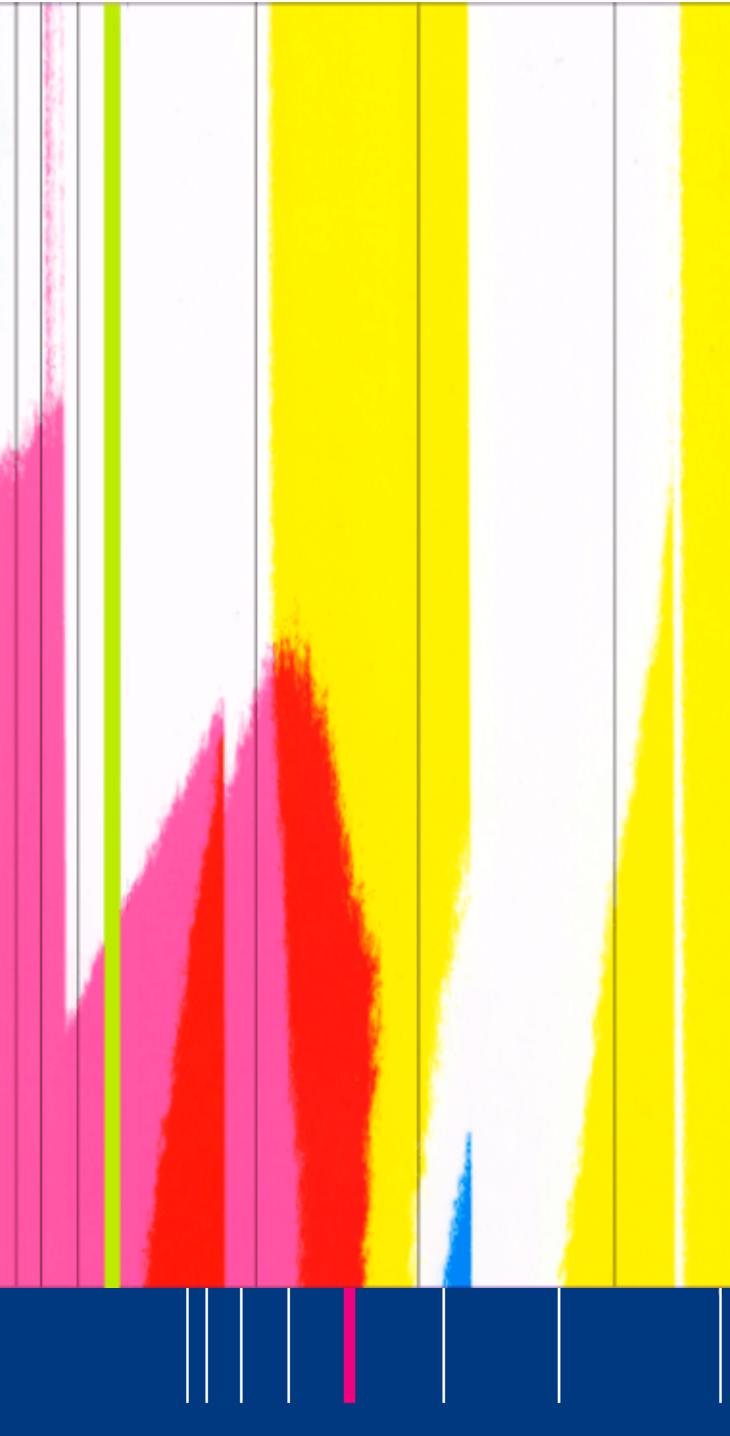
Git et vous?

Panorama des forges

Les forges et vous?

Un cas pratique sous GitHub

Optionnel : Mise en place et travail sur votre projet



Git :

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

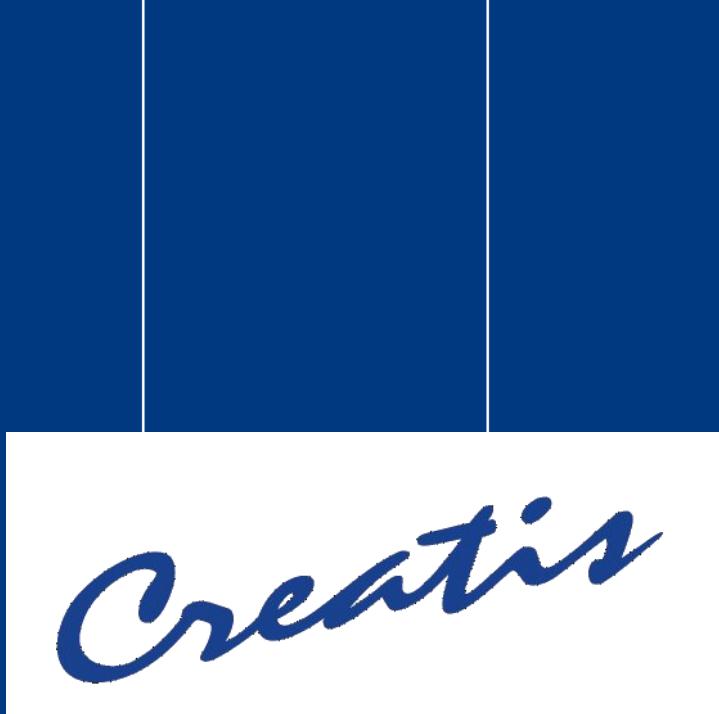
Claire Mouton

[claire.mouton@creatis.insa-lyon.fr](mailto:claire.mouton@creatis.insa-lyon.fr)

EN VOL 2016 – 1er décembre 2016

**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

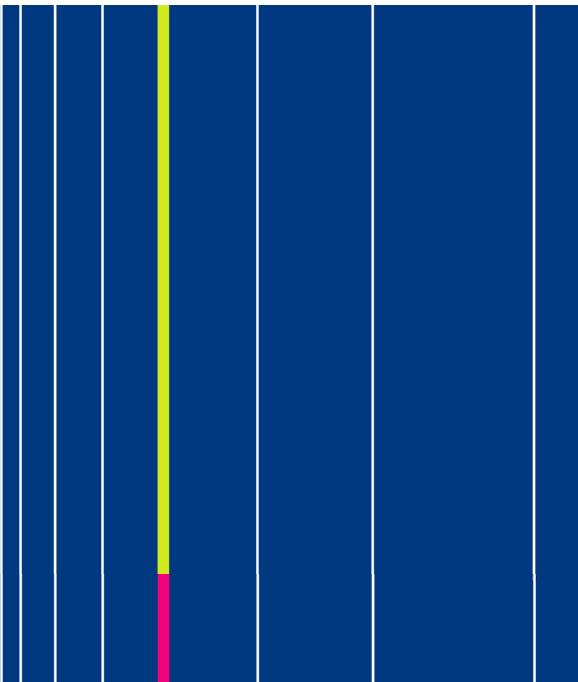
## Product management

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

# What are SCM usages?

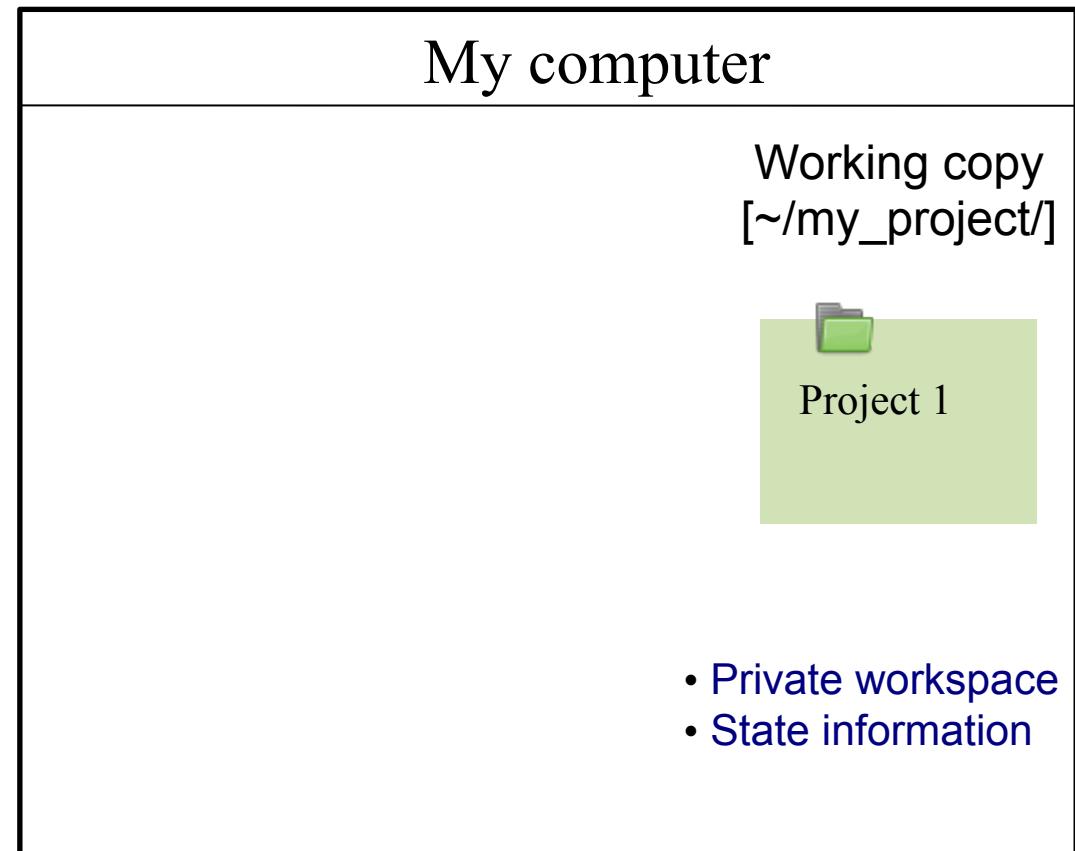
- Version history
  - Navigation
  - Retrieve older versions
  - ...
- Collaborative work
  - Simultaneous work of several people on the same project
    - ex: Conflicts resolution
  - Fork / work on another branch
    - ex: A PhD student who wants to start an experimentation
- Applies to text files (code in any language, article/documentation .tex, web site, system configuration files, .odt) (based on diff)
- Less suitable for binaries, images, .dll, ...
- Shouldn't be used to backup! Even if it saves a copy of your work, tux is there for backup...

# Core Notions

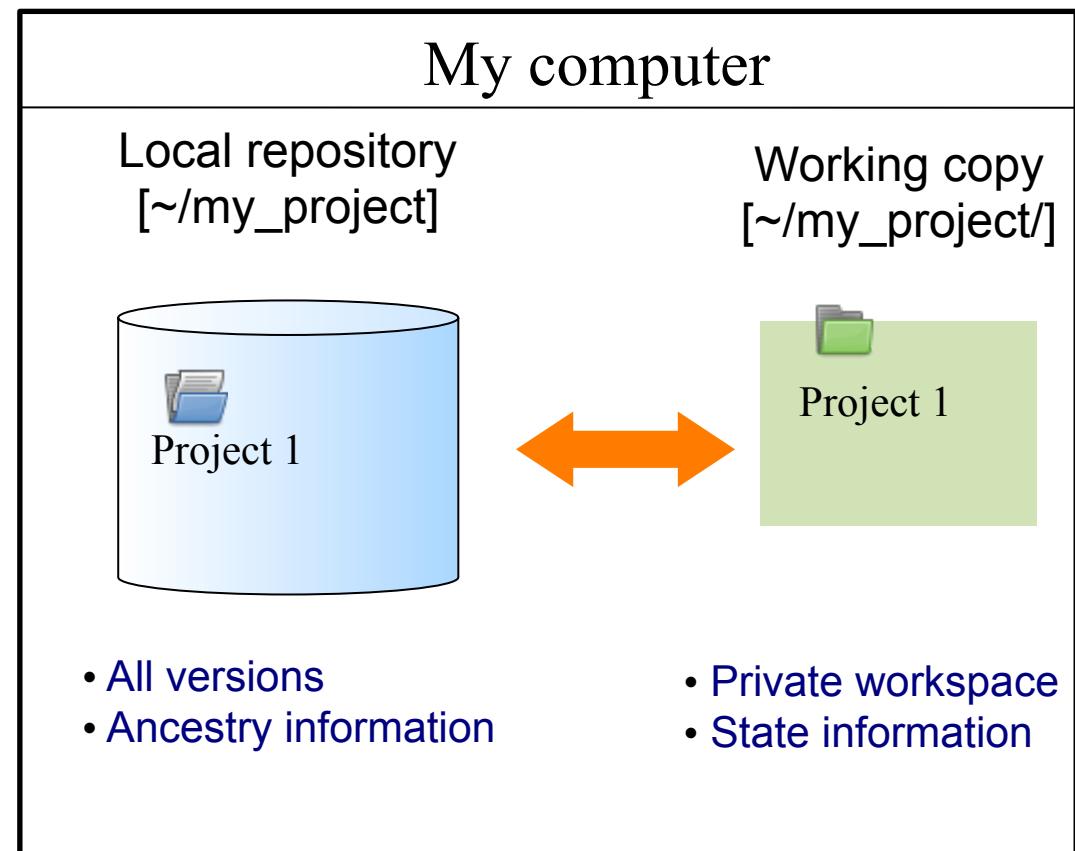
A series of vertical lines of varying heights and colors (white, yellow, red) are positioned on the left side of the slide.

Creatis

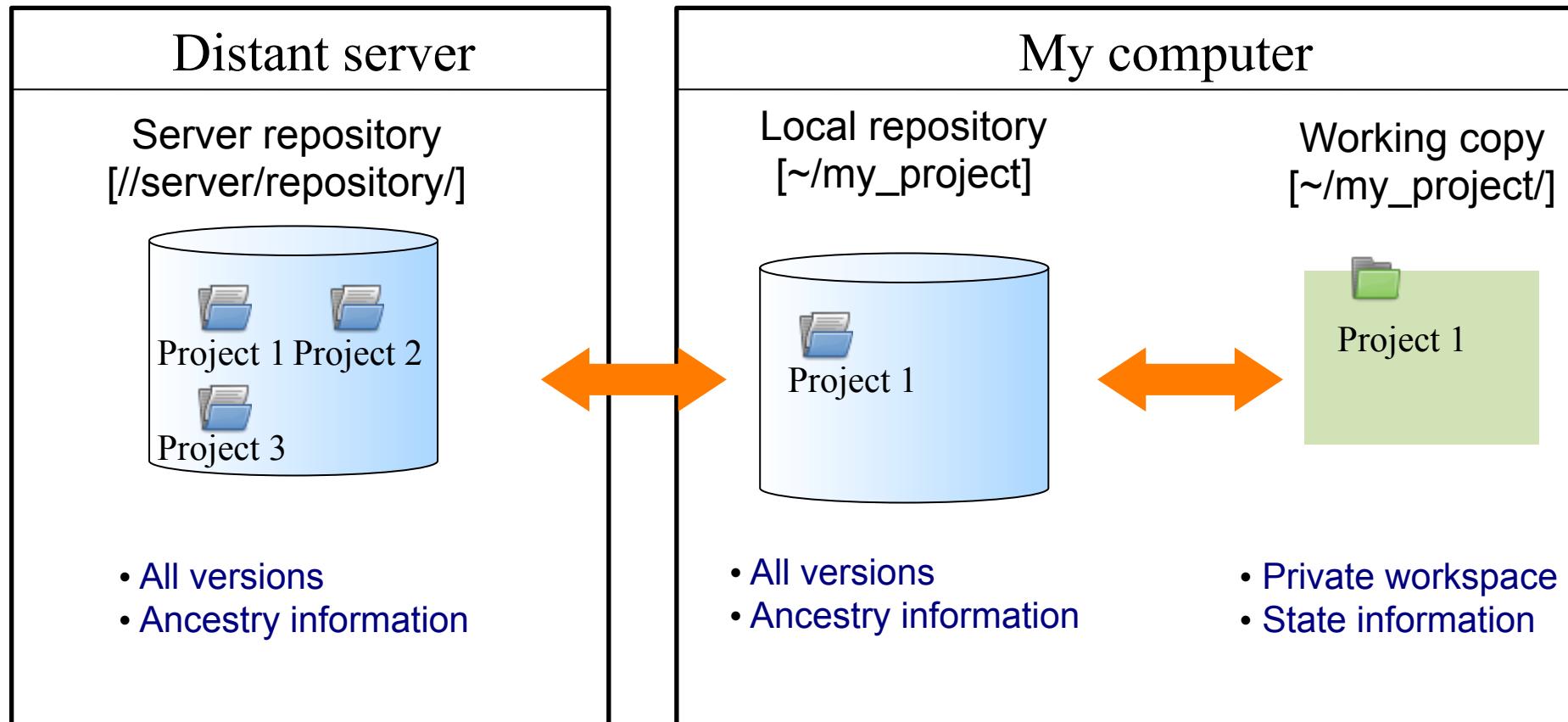
## Core notions > *Organization*



## Core notions > *Organization*



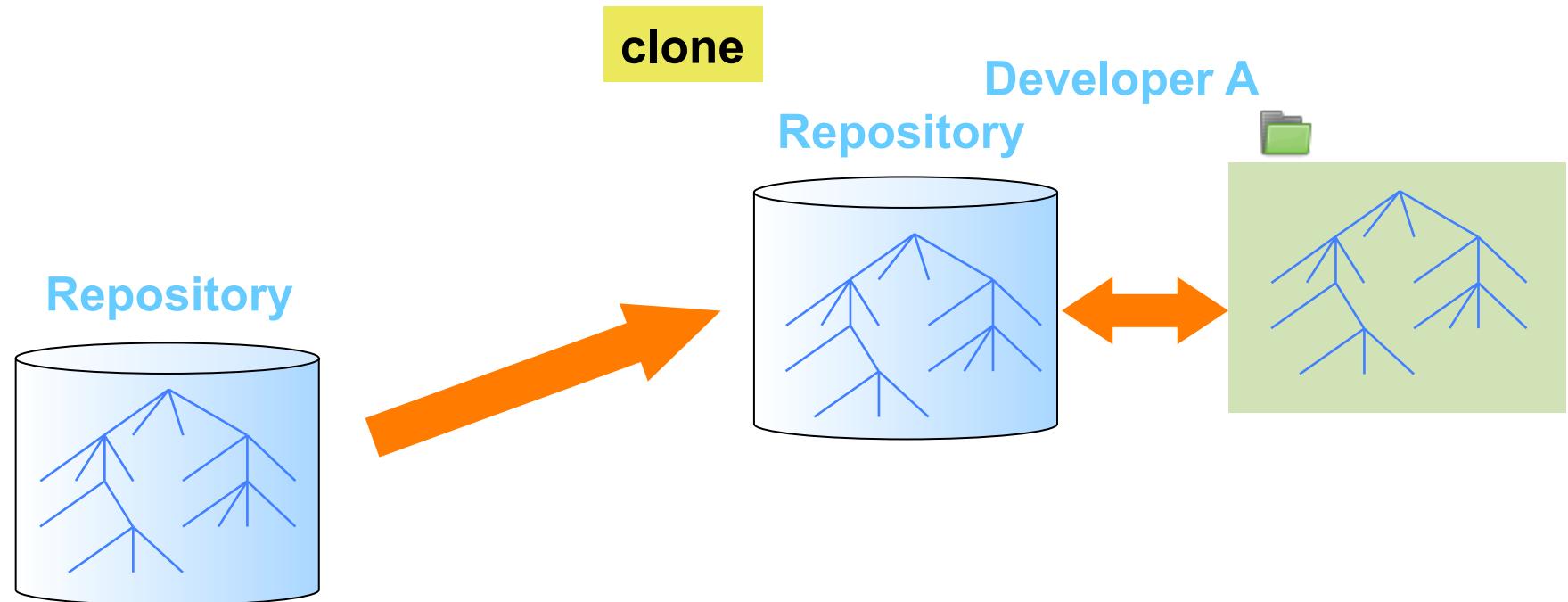
# Core notions > Organization



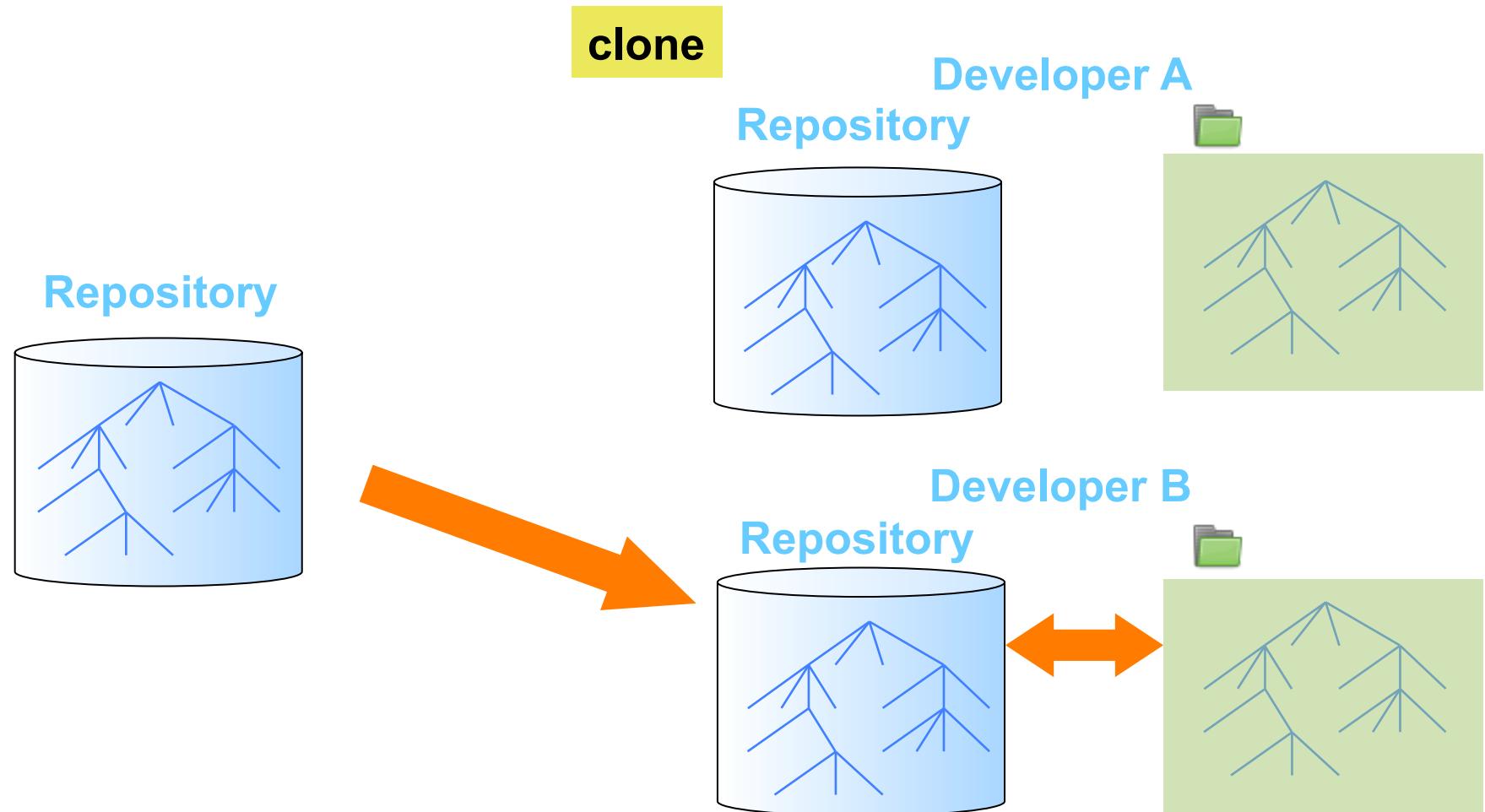
# Core notions >

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

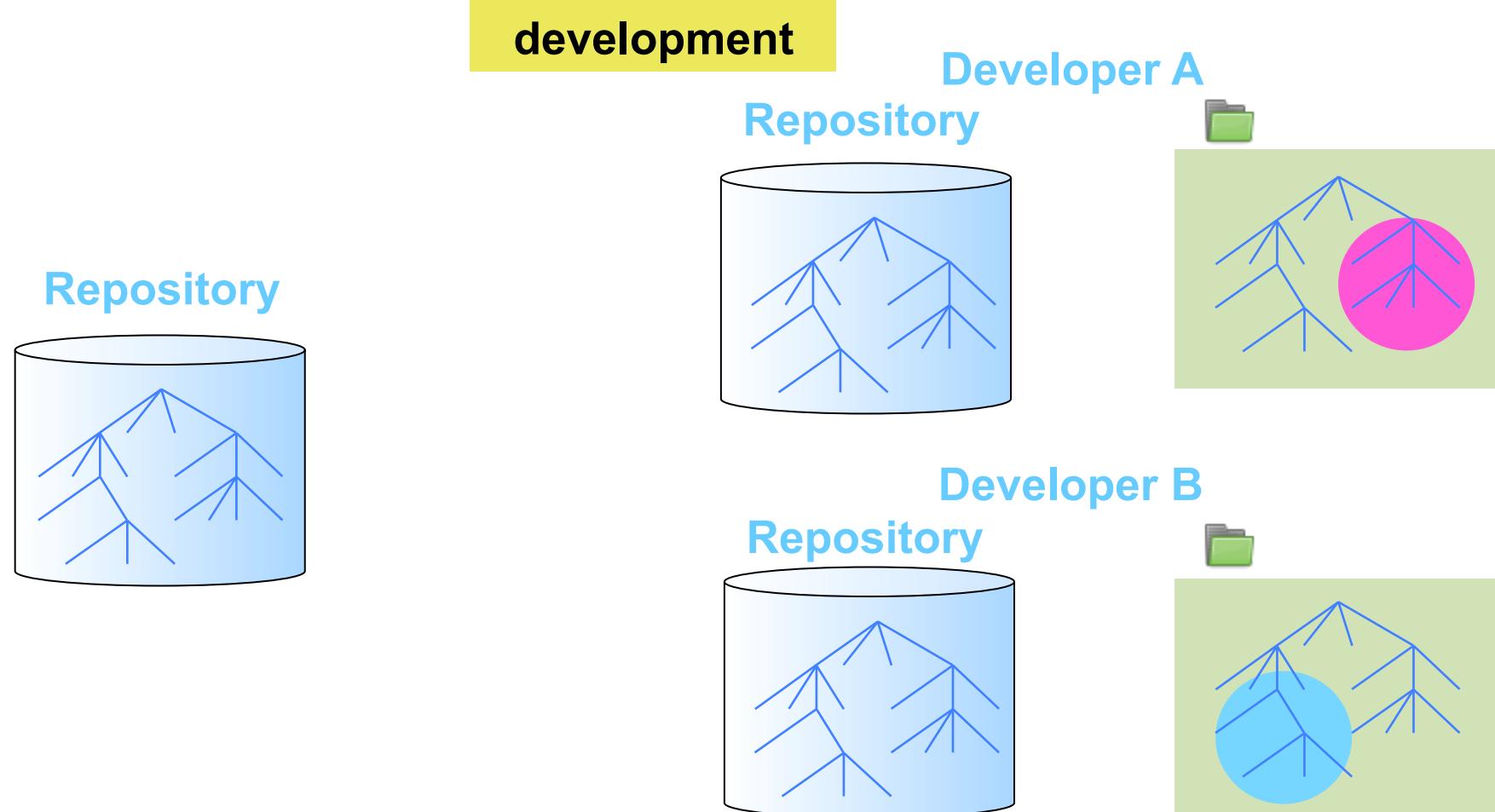
## Core notions > Get a project



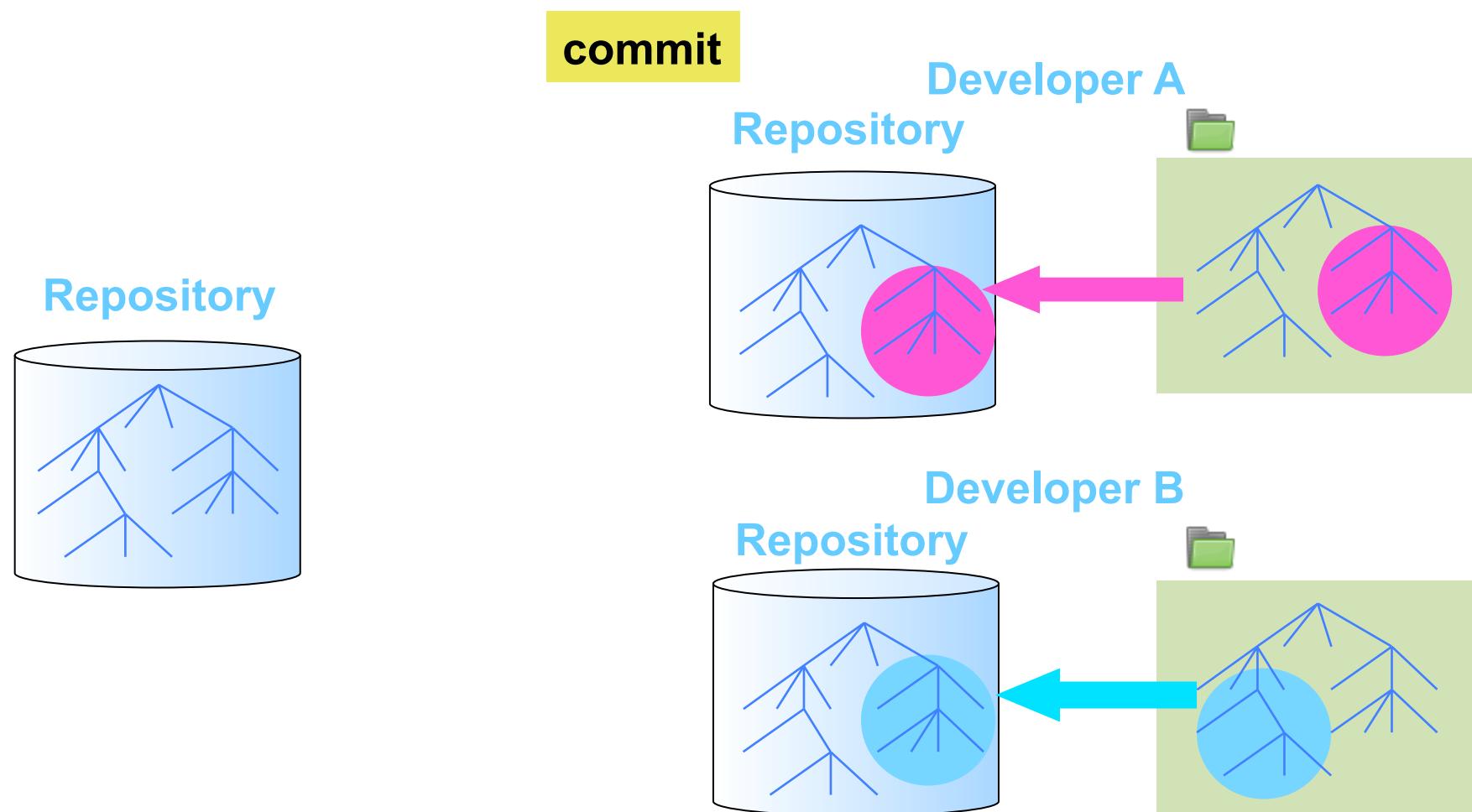
## Core notions > Get a project



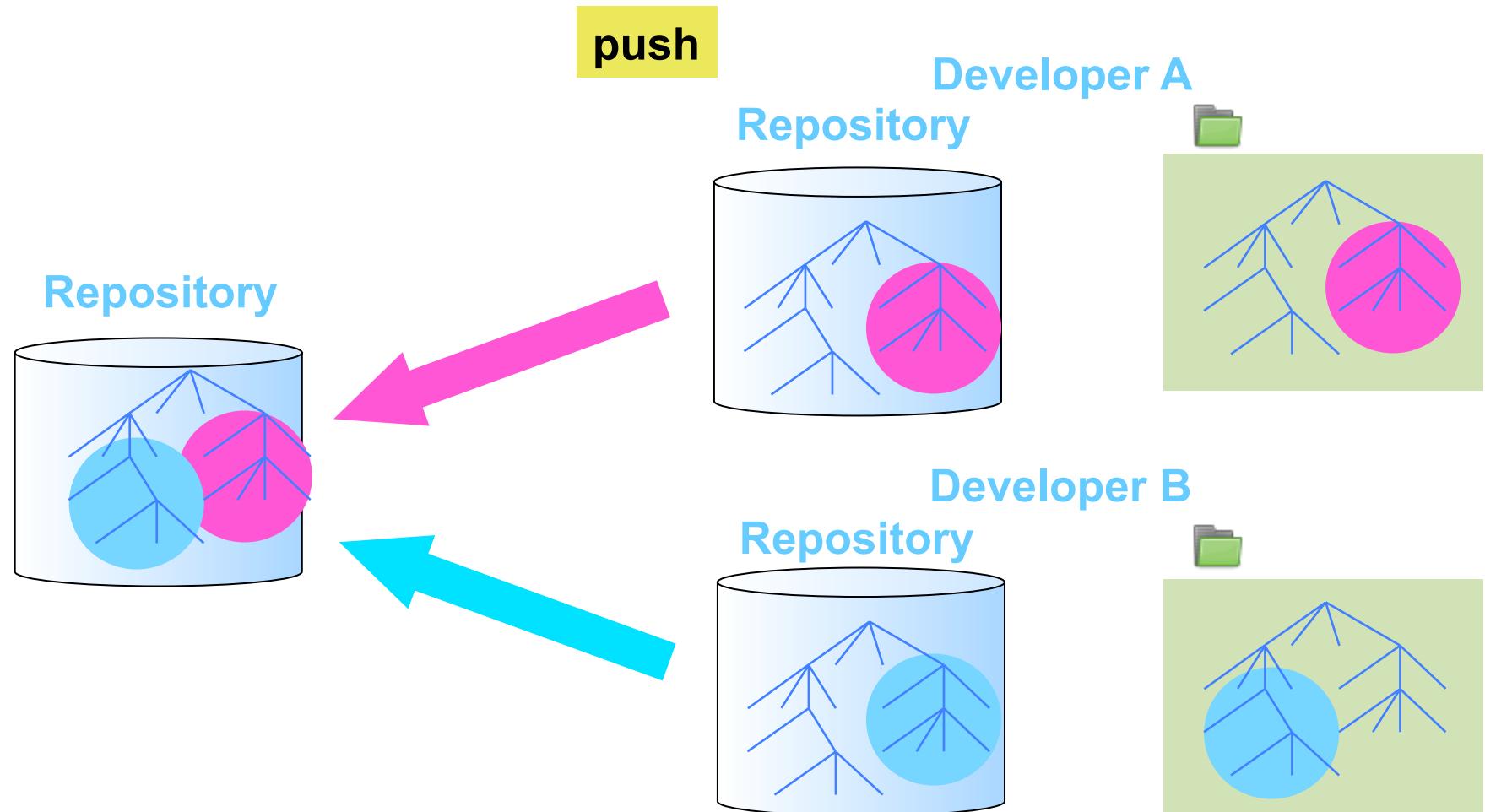
# Core notions > Commit



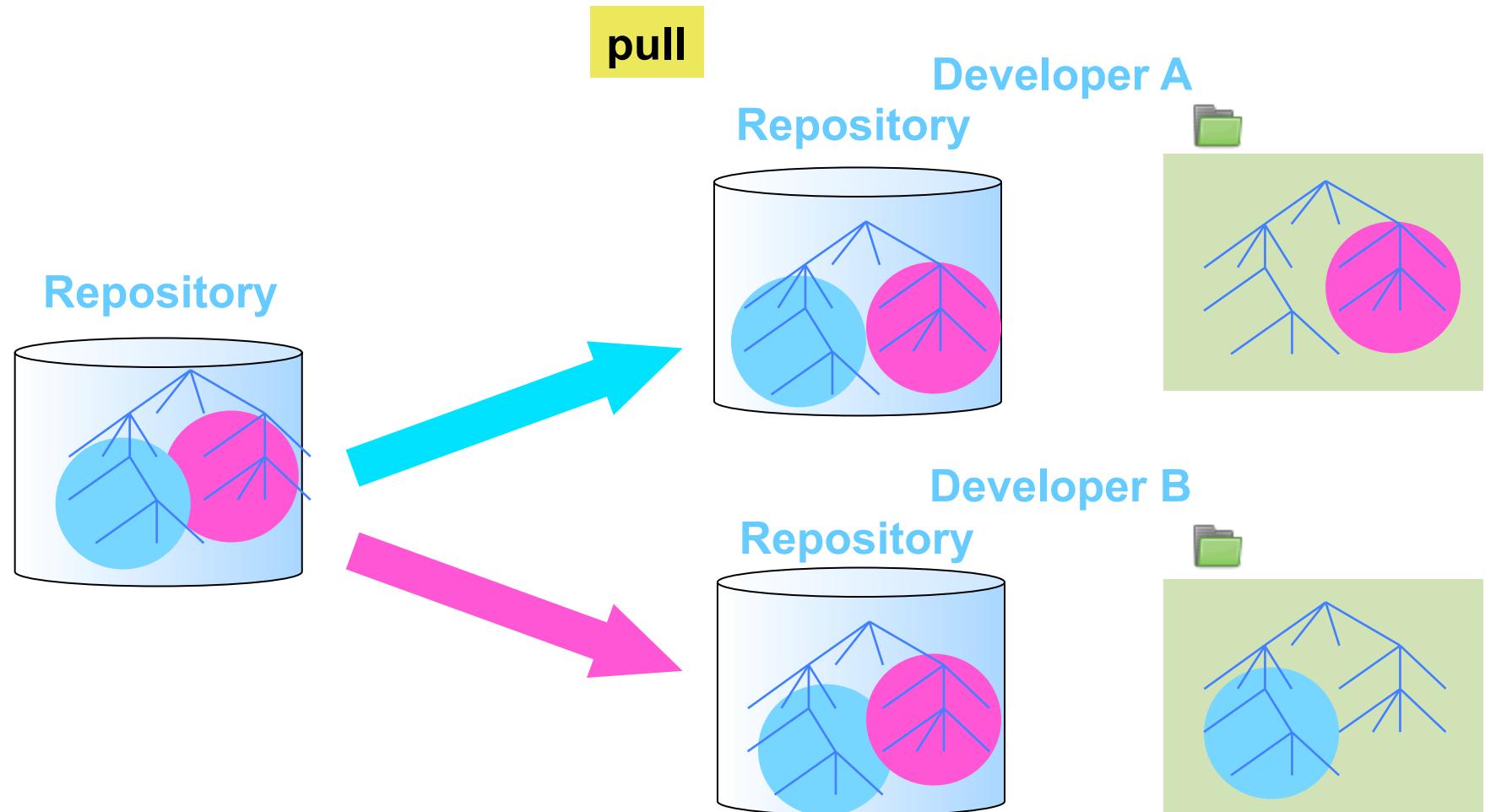
## Core notions > Commit



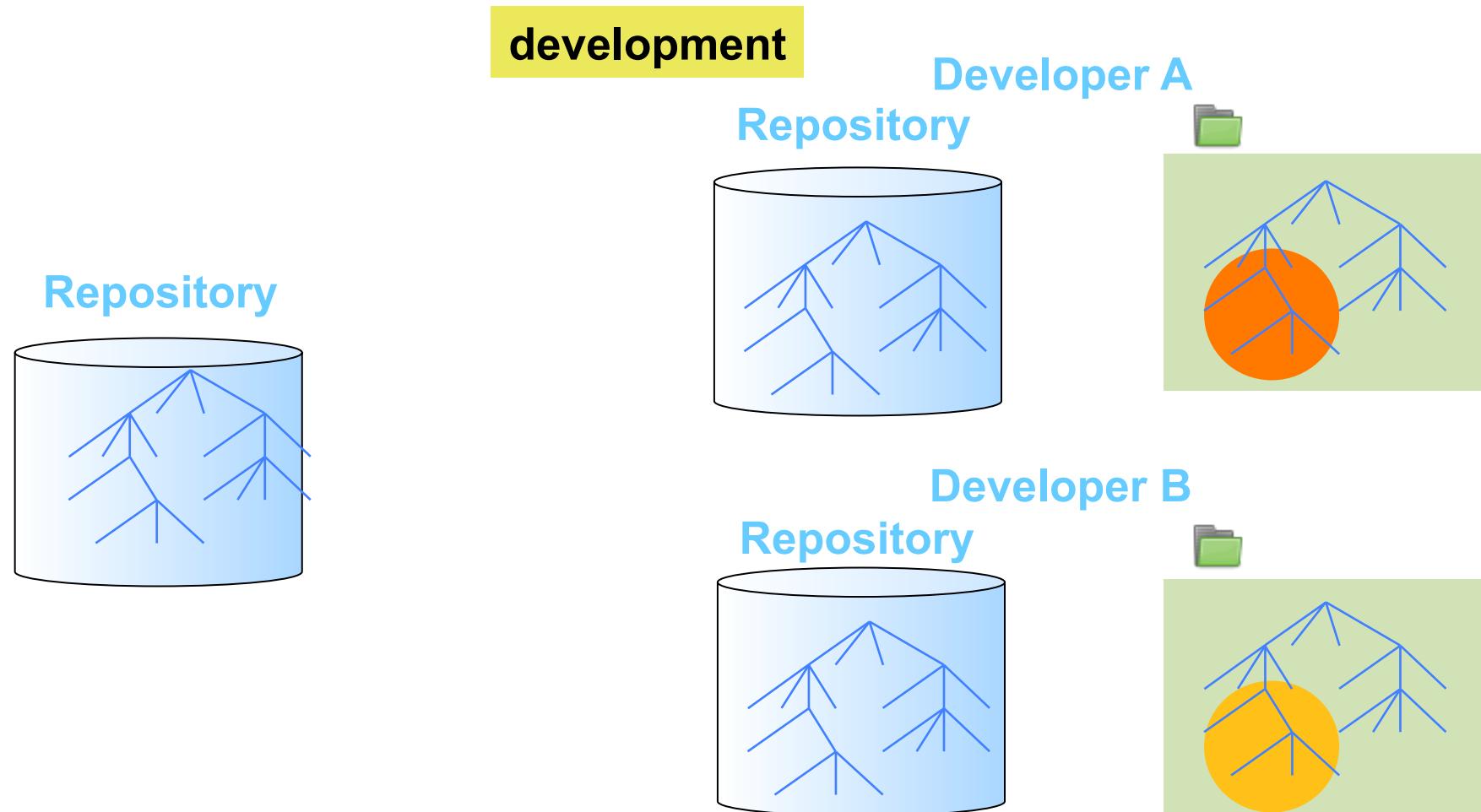
## Core notions > Commit



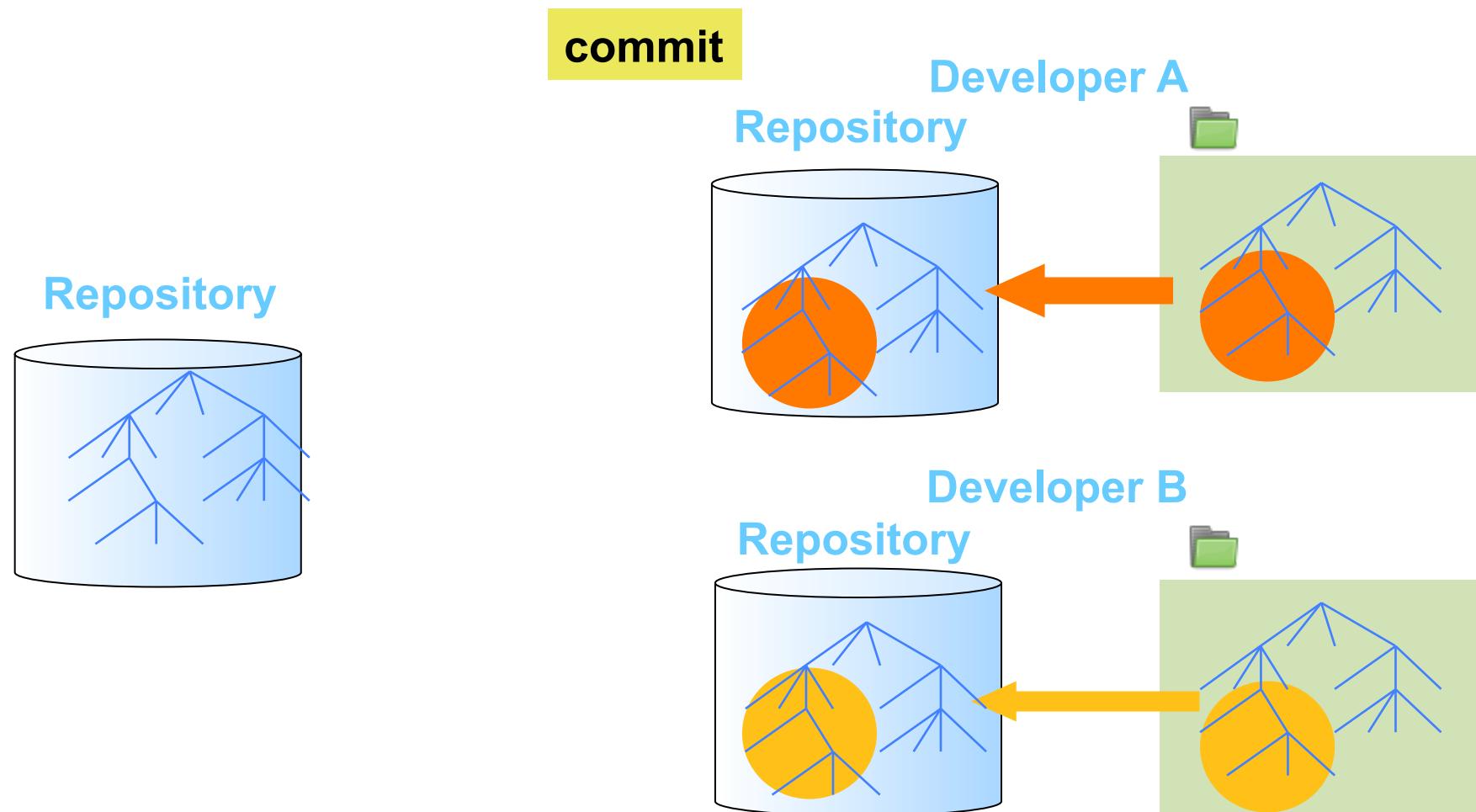
## Core notions > Commit



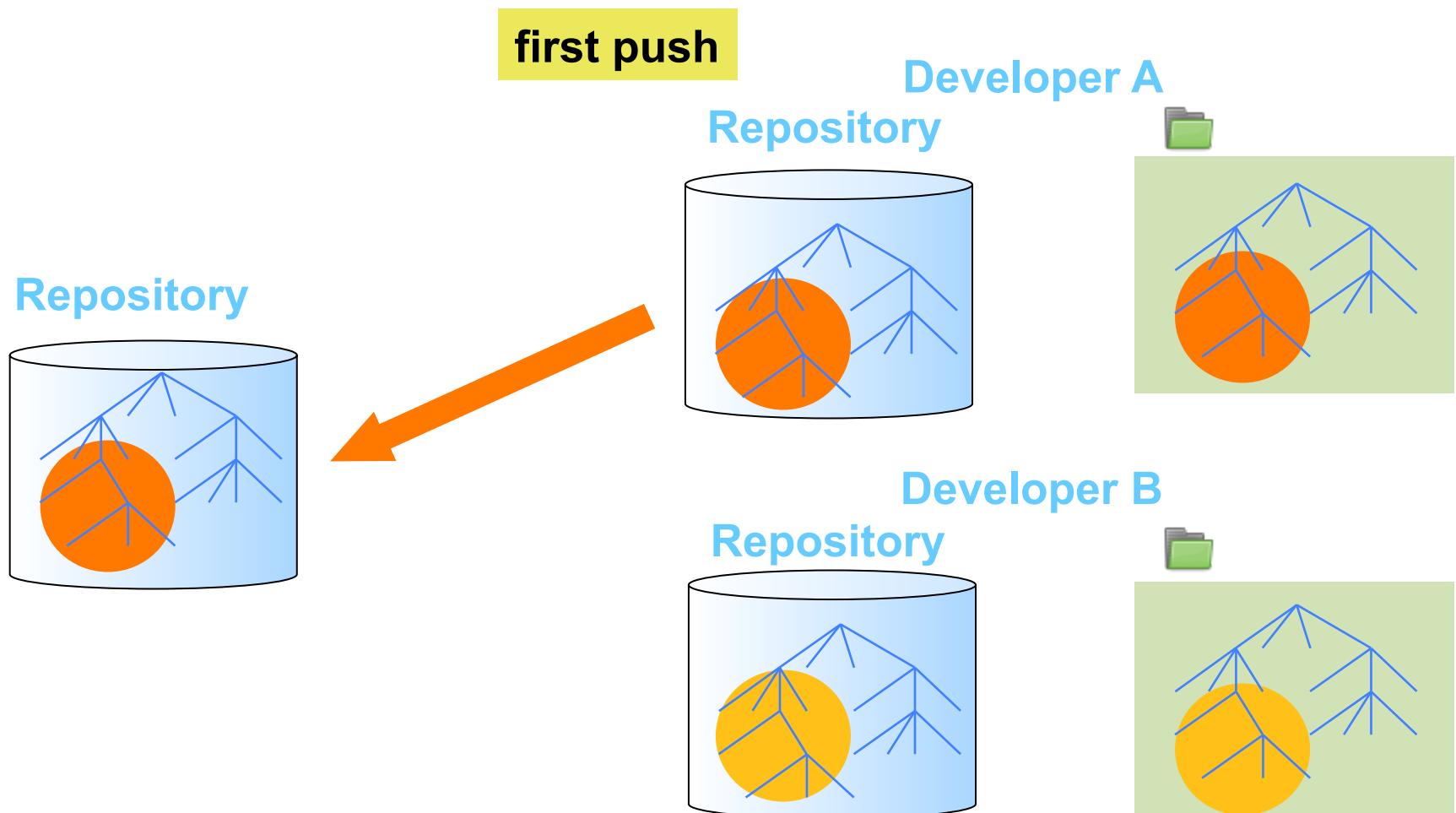
## Core notions > Conflict



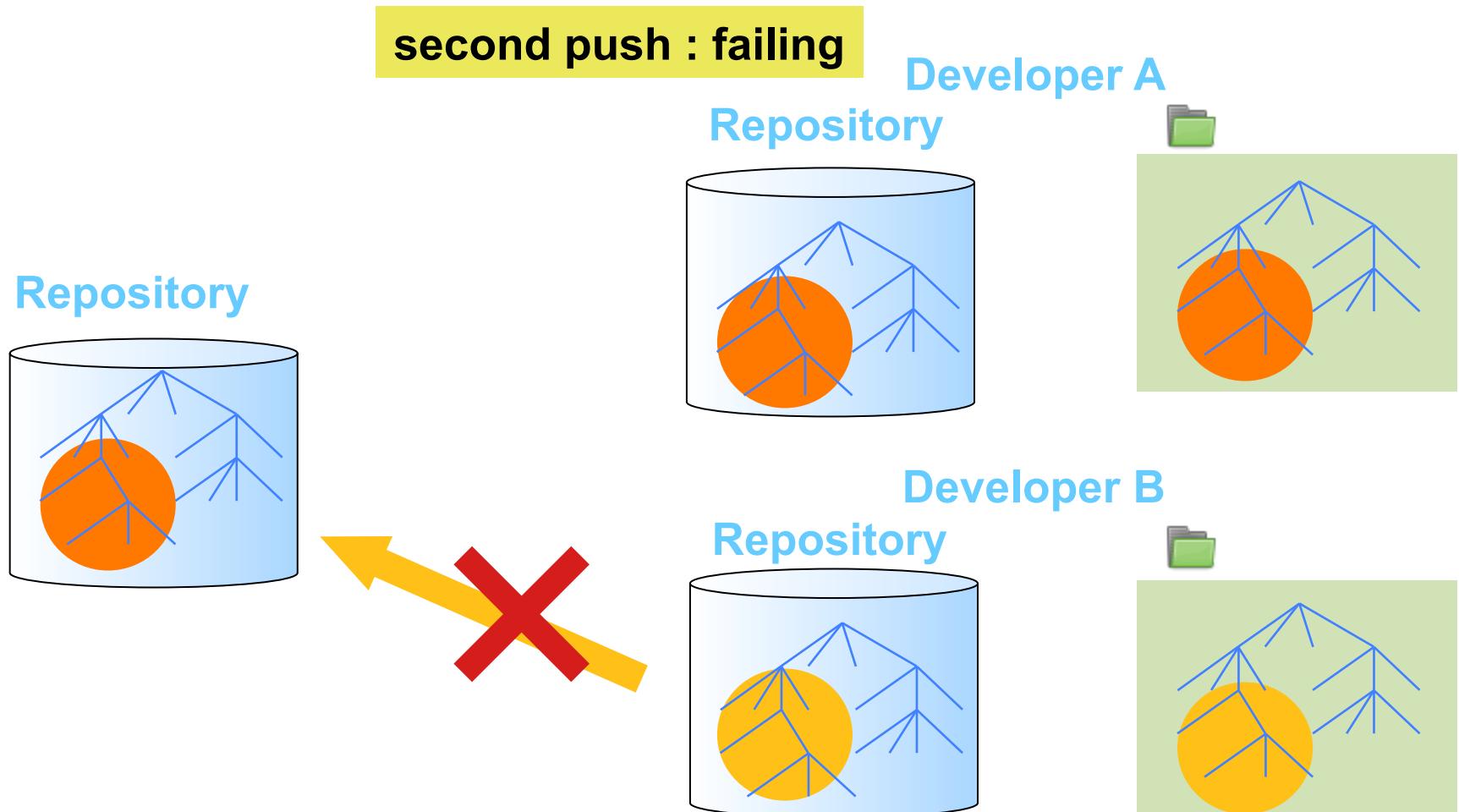
## Core notions > Conflict



## Core notions > Conflict

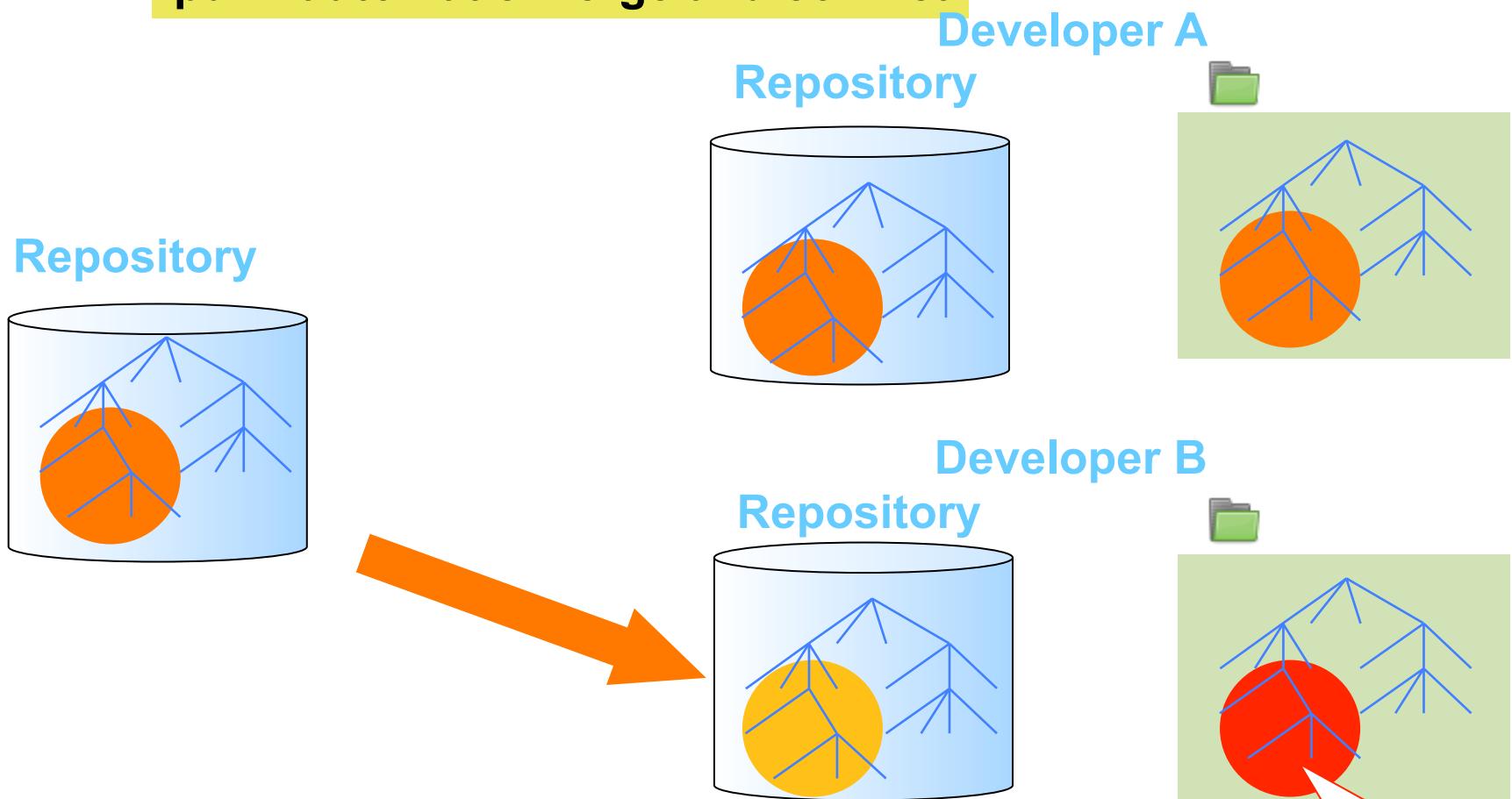


## Core notions > Conflict

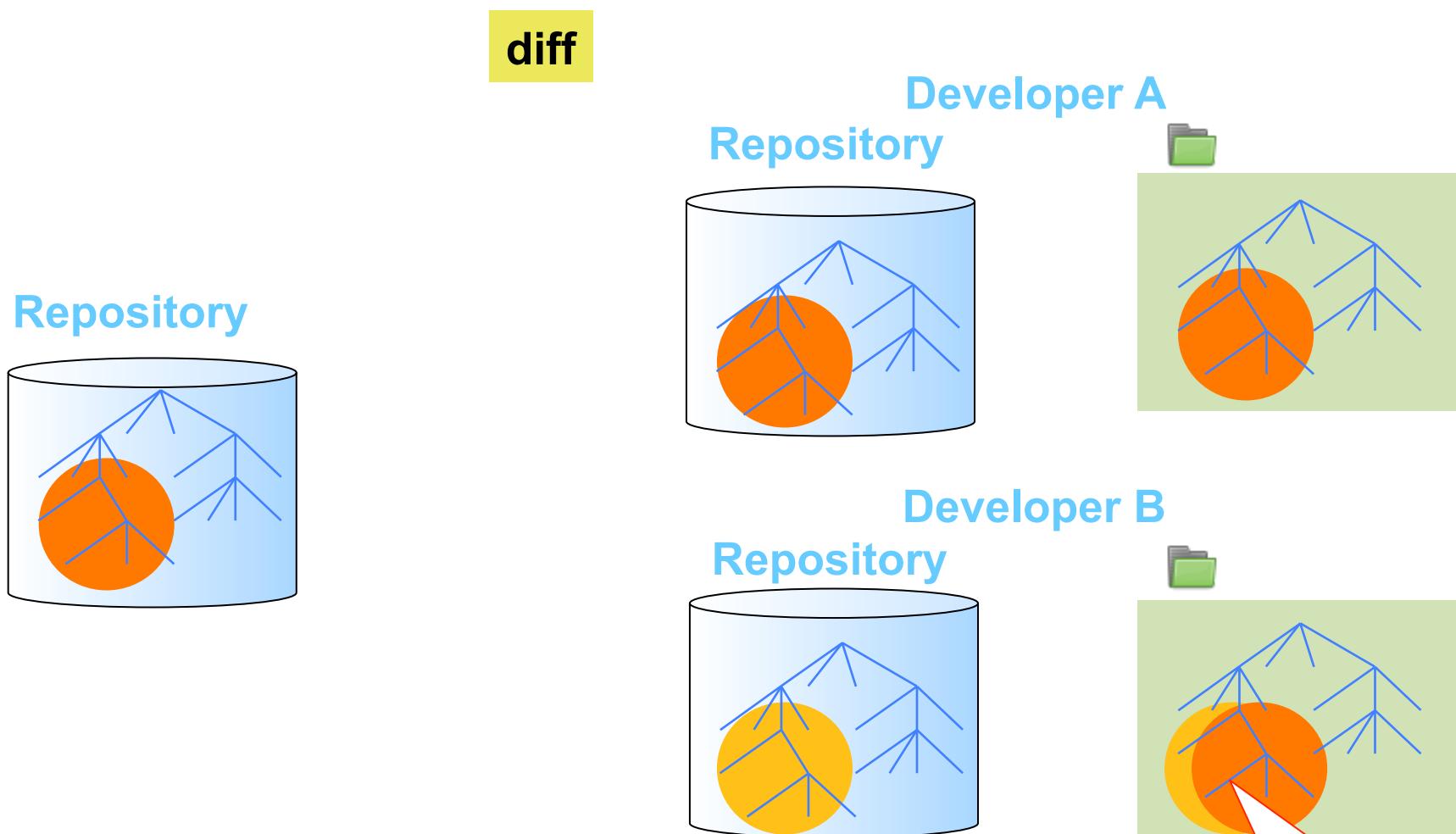


## Core notions > Conflict

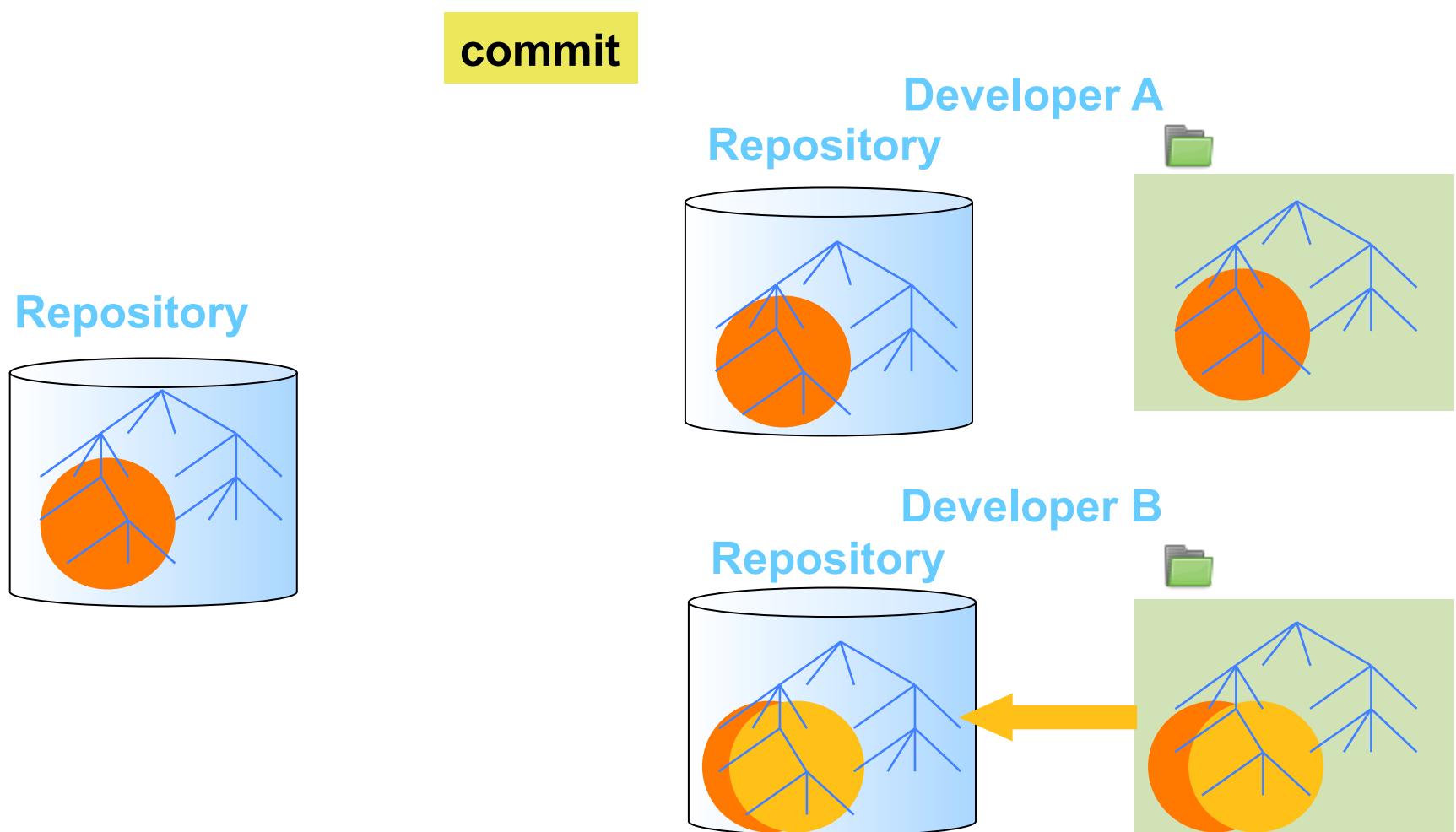
**pull : automatic merge and conflict**



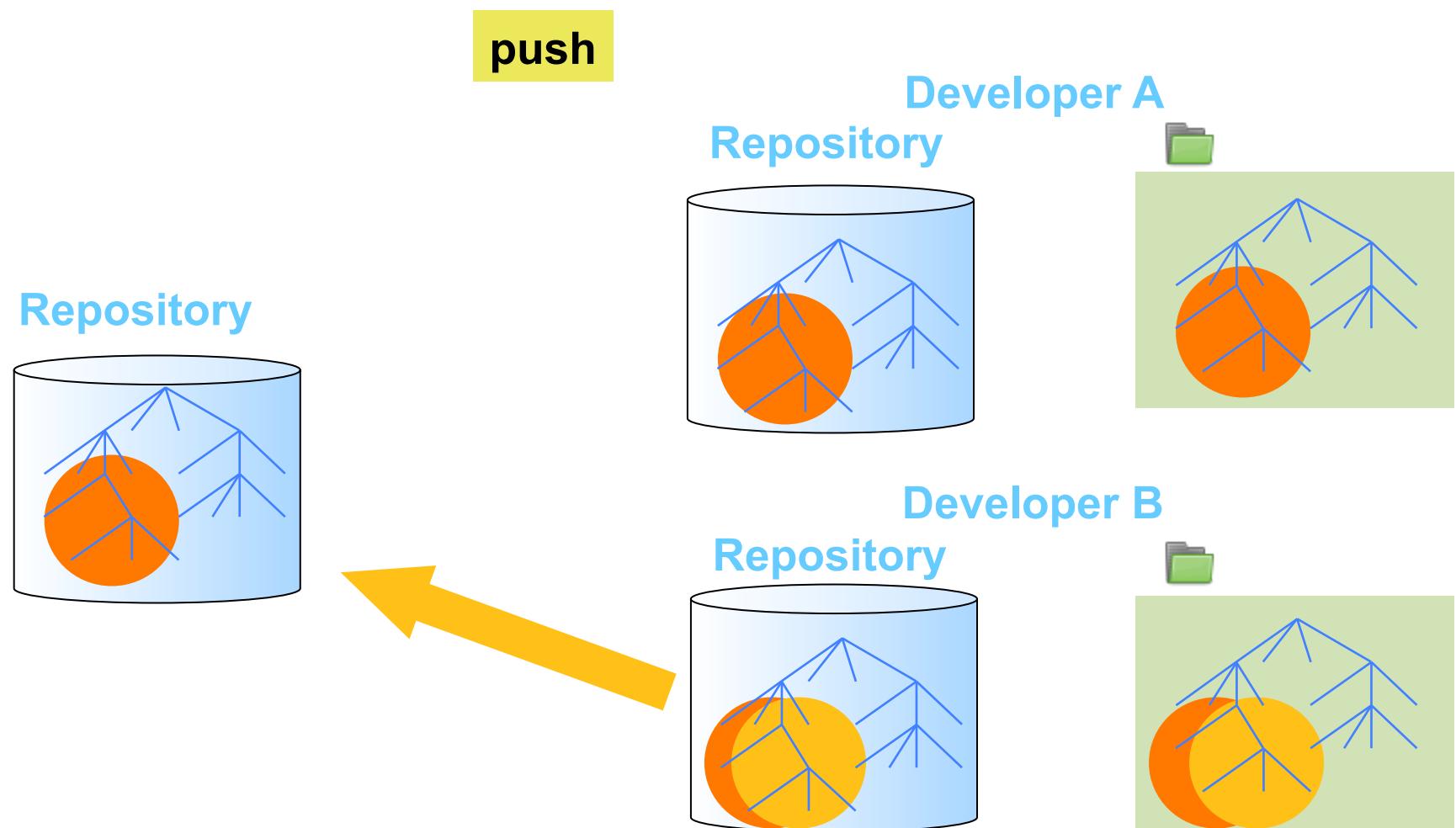
## Core notions > Conflict



## Core notions > Conflict



## Core notions > Conflict



## Core notions > *Version tagging*

tag

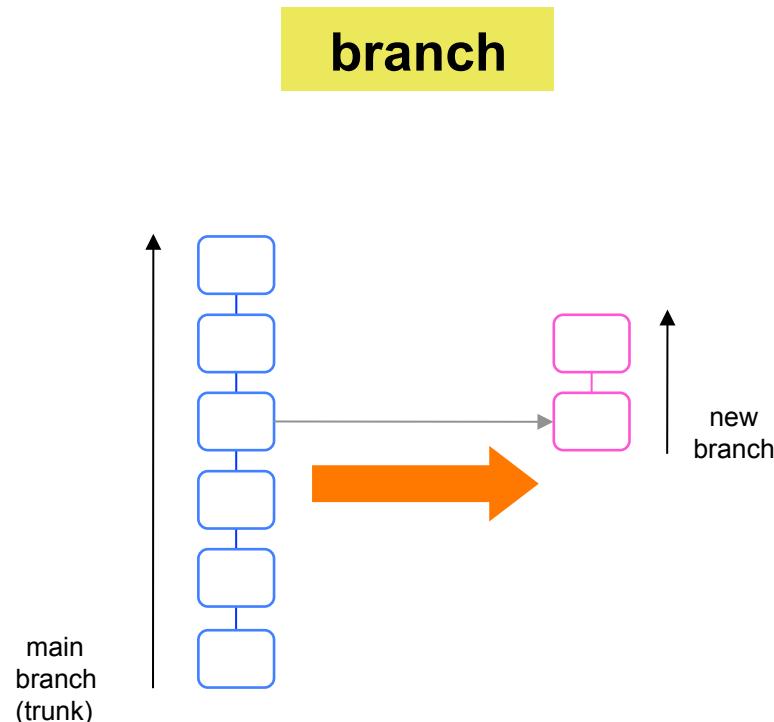
- Often used to tag the repository on important events such as a software release or article submission
- Allows you to easily retrieve a specific version of the software/article
  - Use / distribute a given release
  - Reproduce bugs for a given release

my\_project-v1.3

## Core notions > *Branch*

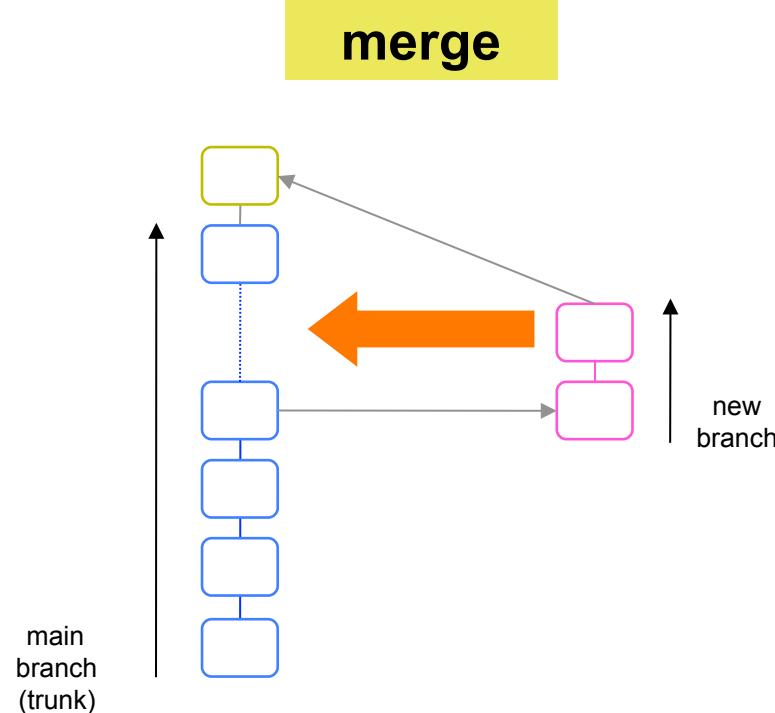
Why?

# Core notions > *Branch*



- 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 changeset is all modifications made on one or several files and is atomic

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

A changeset is recorded in the *repository* via a *commit*. It creates a new *revision* of the project.

A commit has a unique identifier :

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

# Centralized SCM

## > CVS and Subversion (SVN)



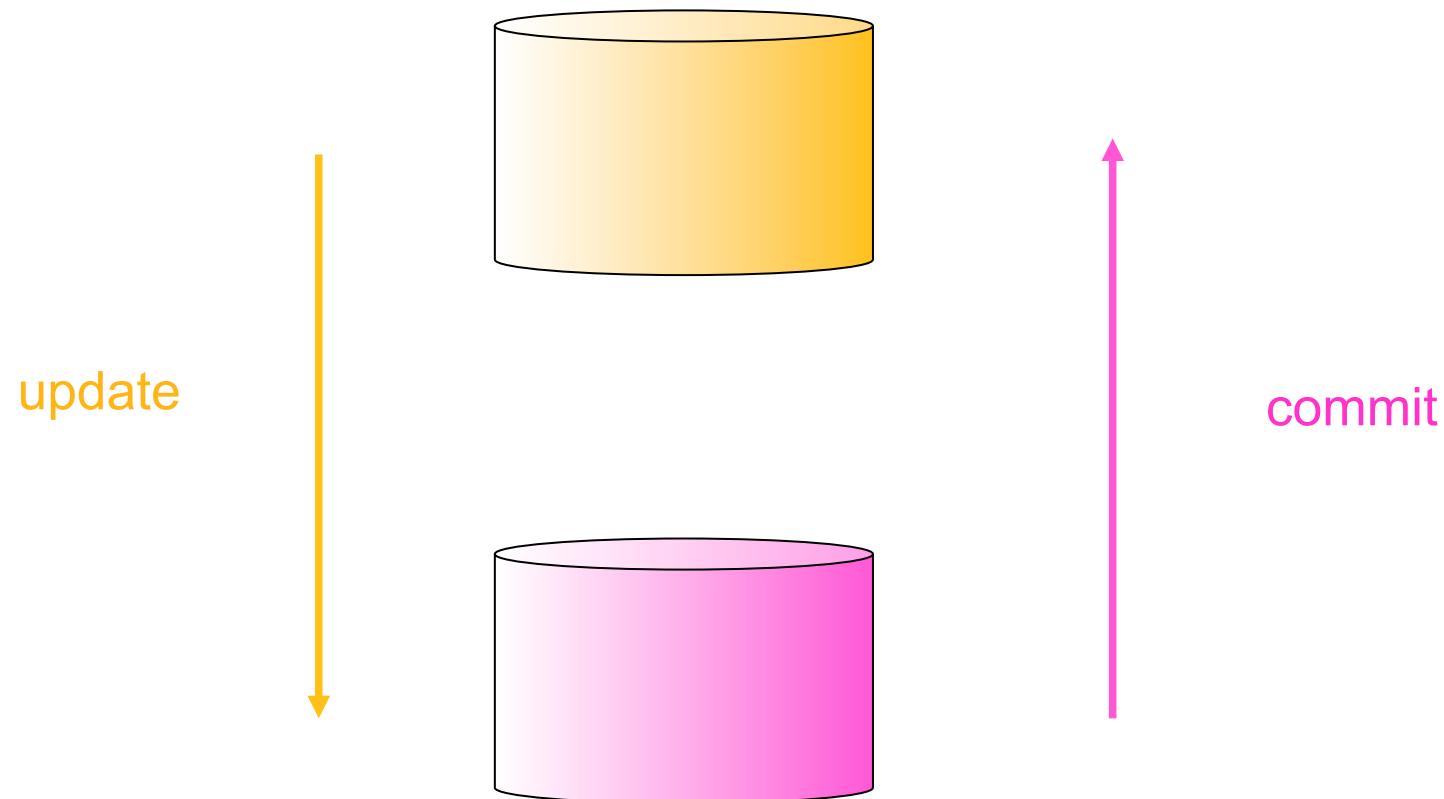
Creatis

# 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
  
- Parallel work (merge) is difficult
  - No memorizing of successive merges information
  - The user has to remember! (to avoid conflict while merging)
- Need to be on line for almost commands
- Privileged users (committers)

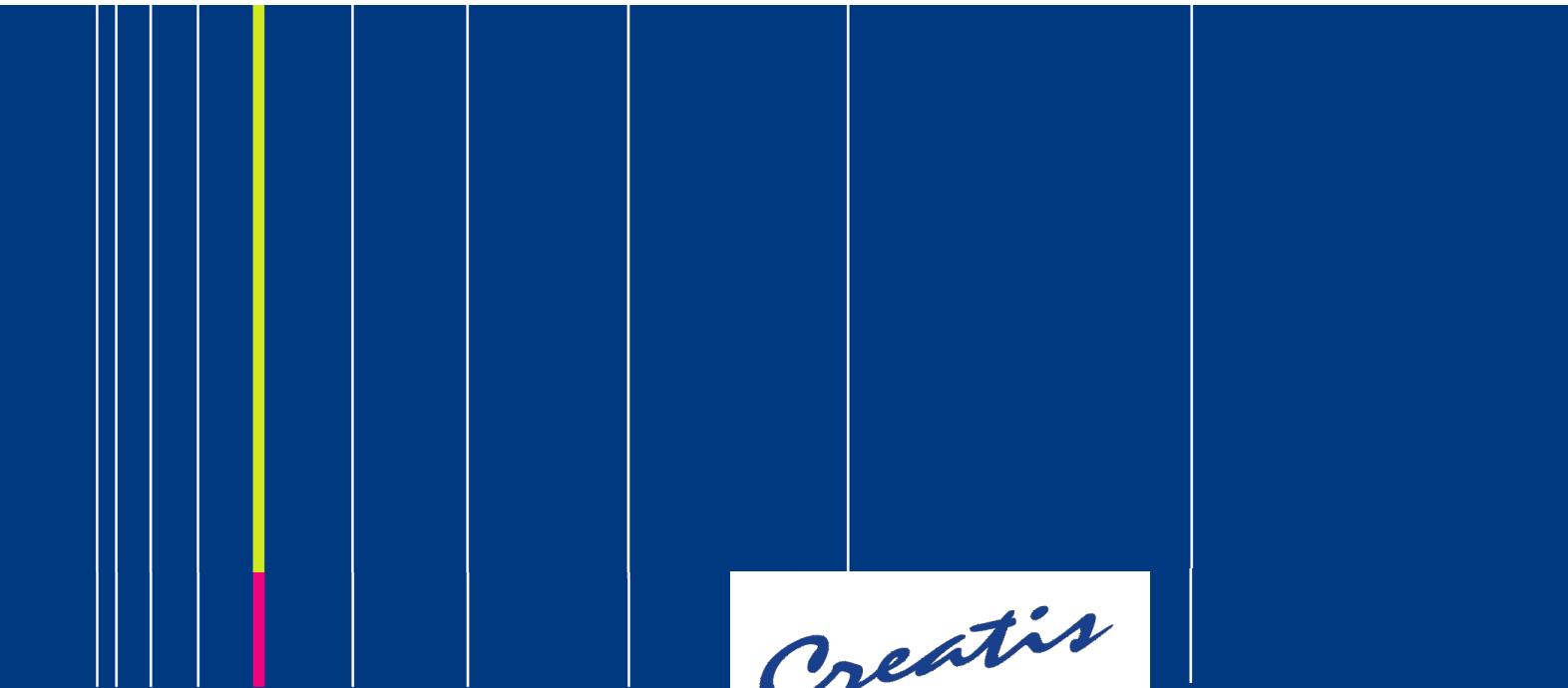


past

present

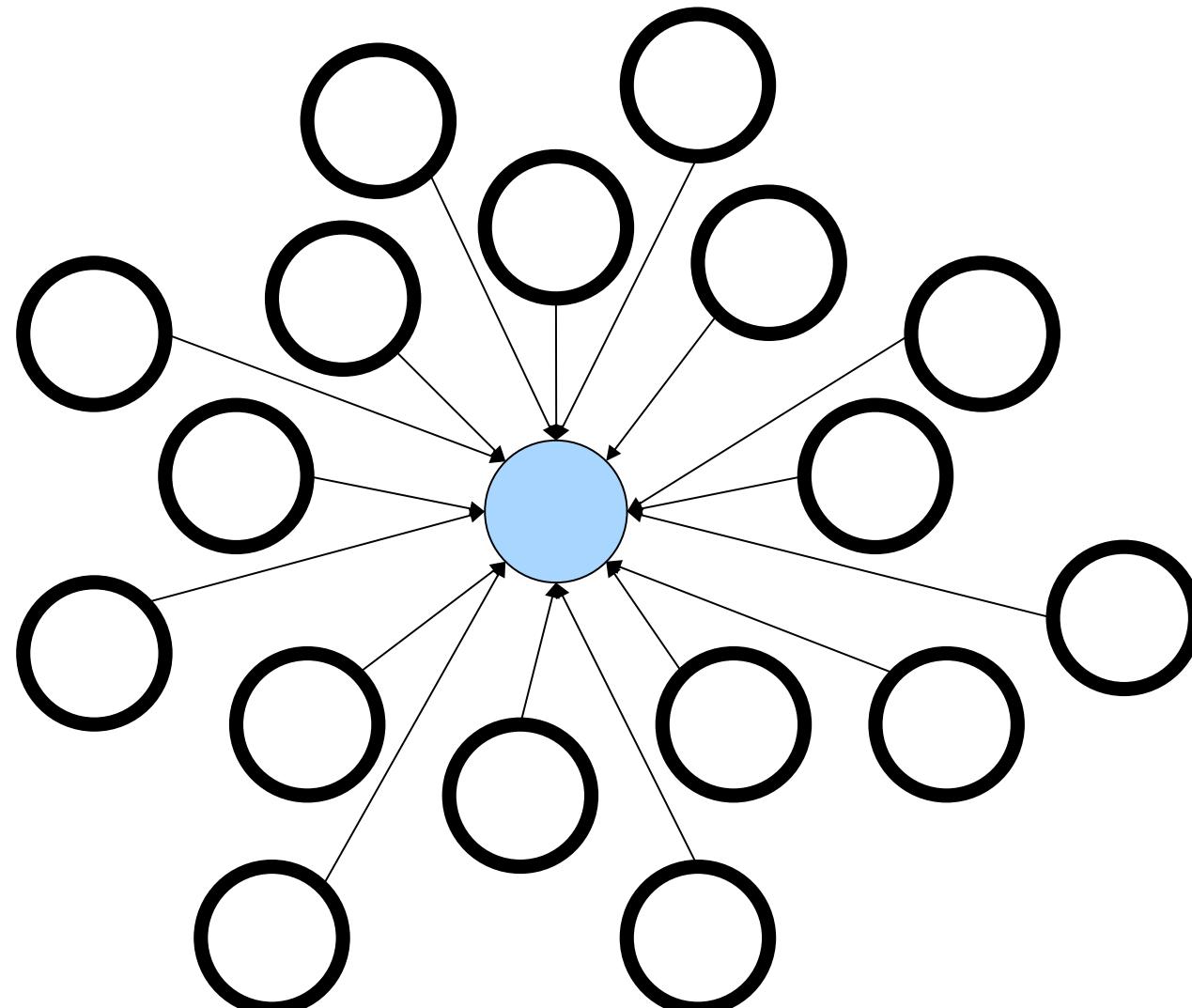
future

# Distributed SCM

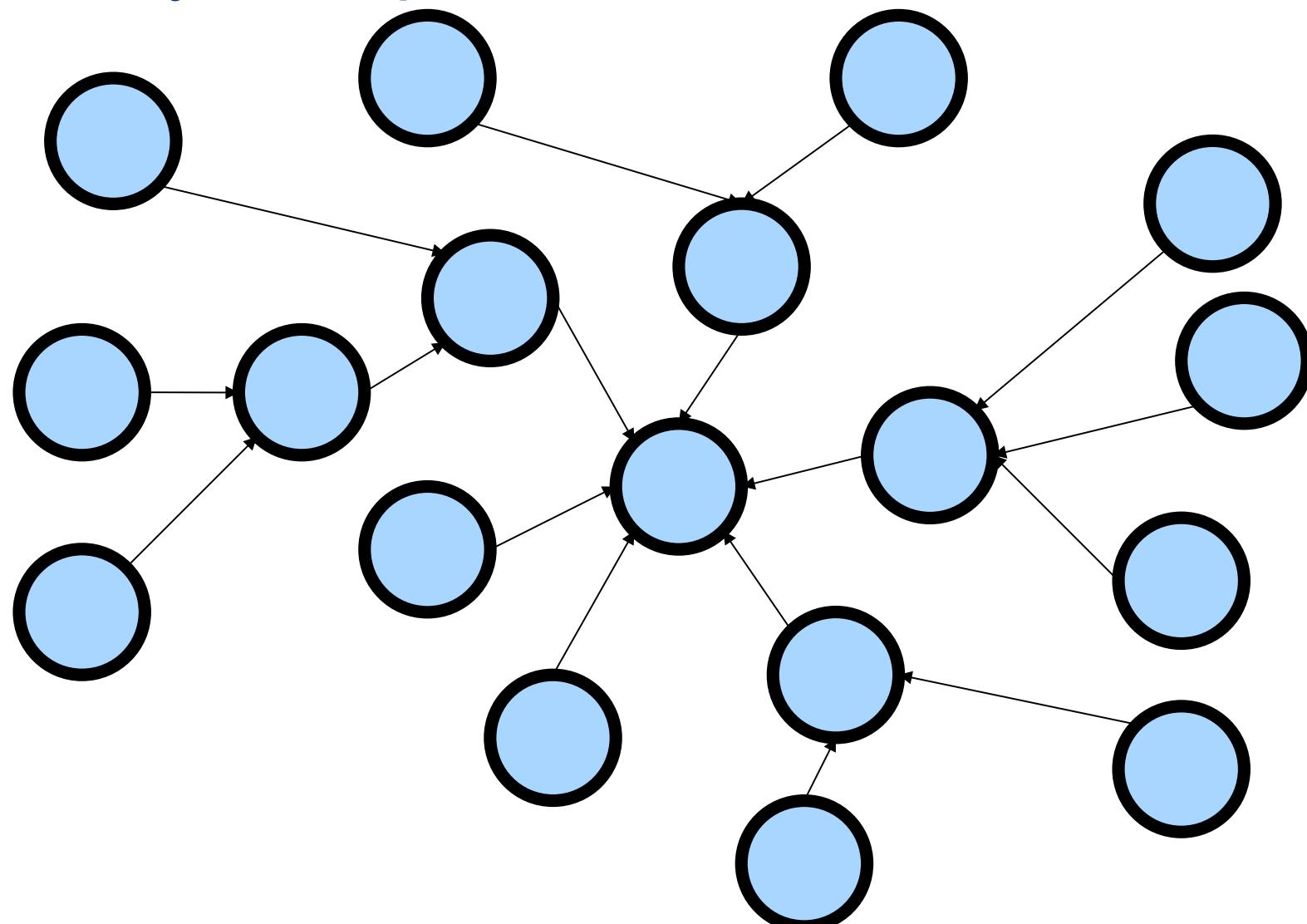


*Creatis*

## Policy example: centralized



# Policy example: distributed

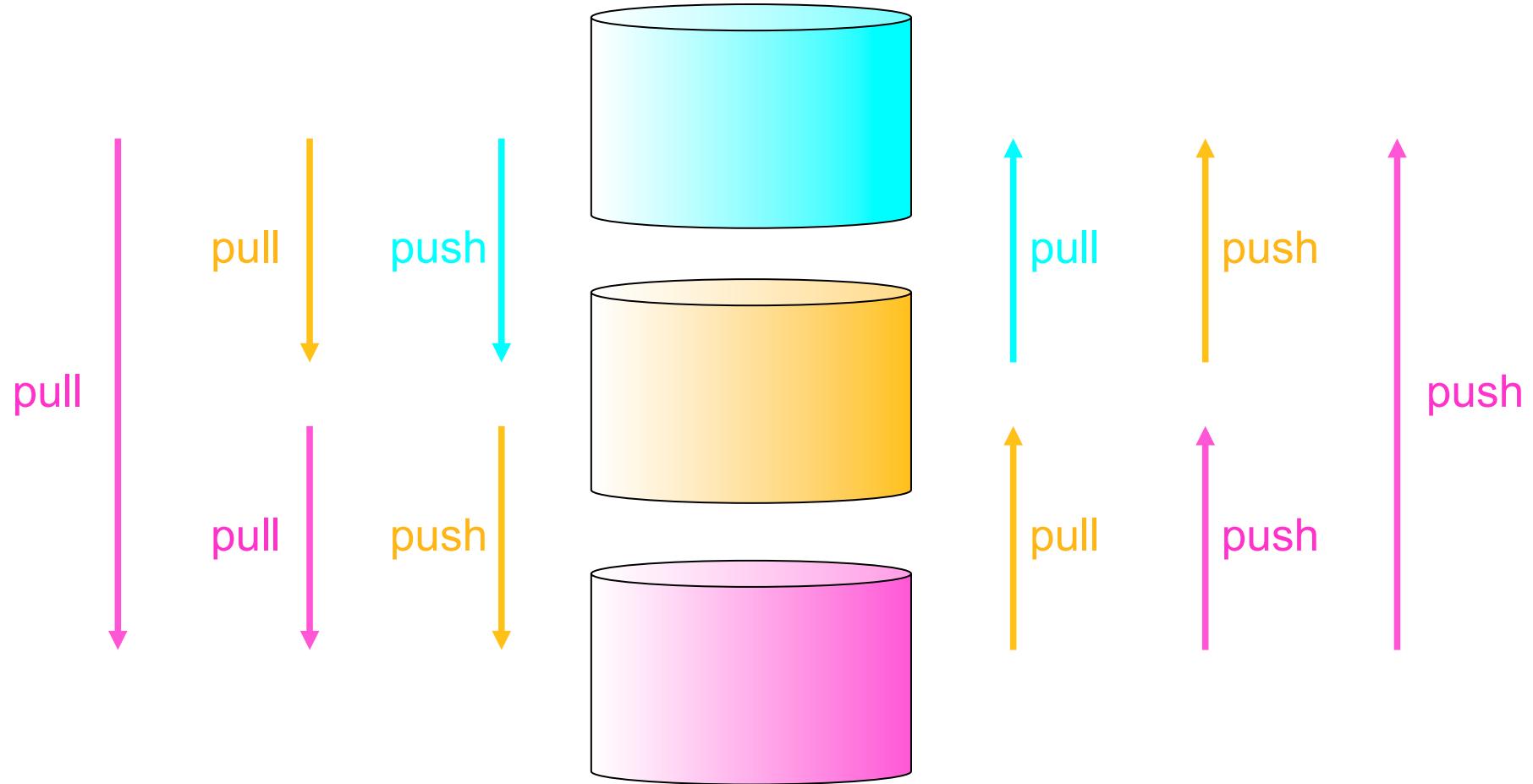


# Decentralized benefits ?

# 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

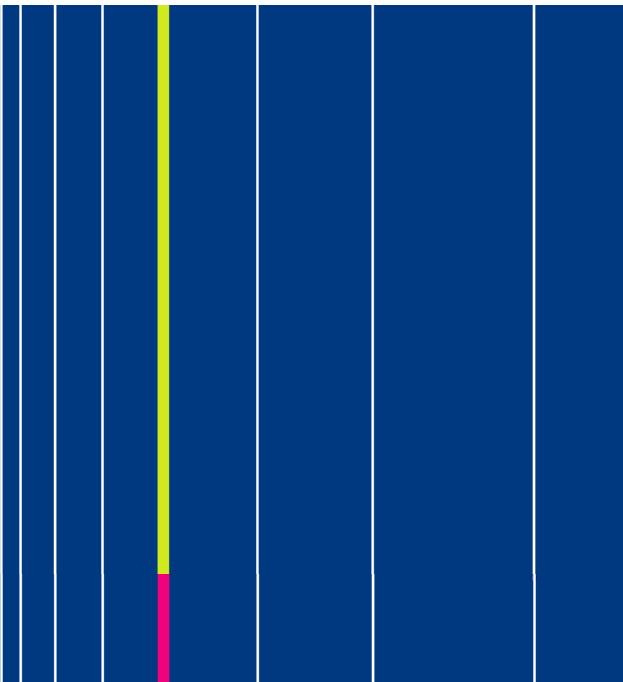


Outil	Type	Description	Projets qui l'utilisent
<a href="#">CVS</a>	Centralisé	C'est un des plus anciens logiciels de gestion de versions. Bien qu'il fonctionne et soit encore utilisé pour certains projets, il est préférable d'utiliser SVN (souvent présenté comme son successeur) qui corrige un certain 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>

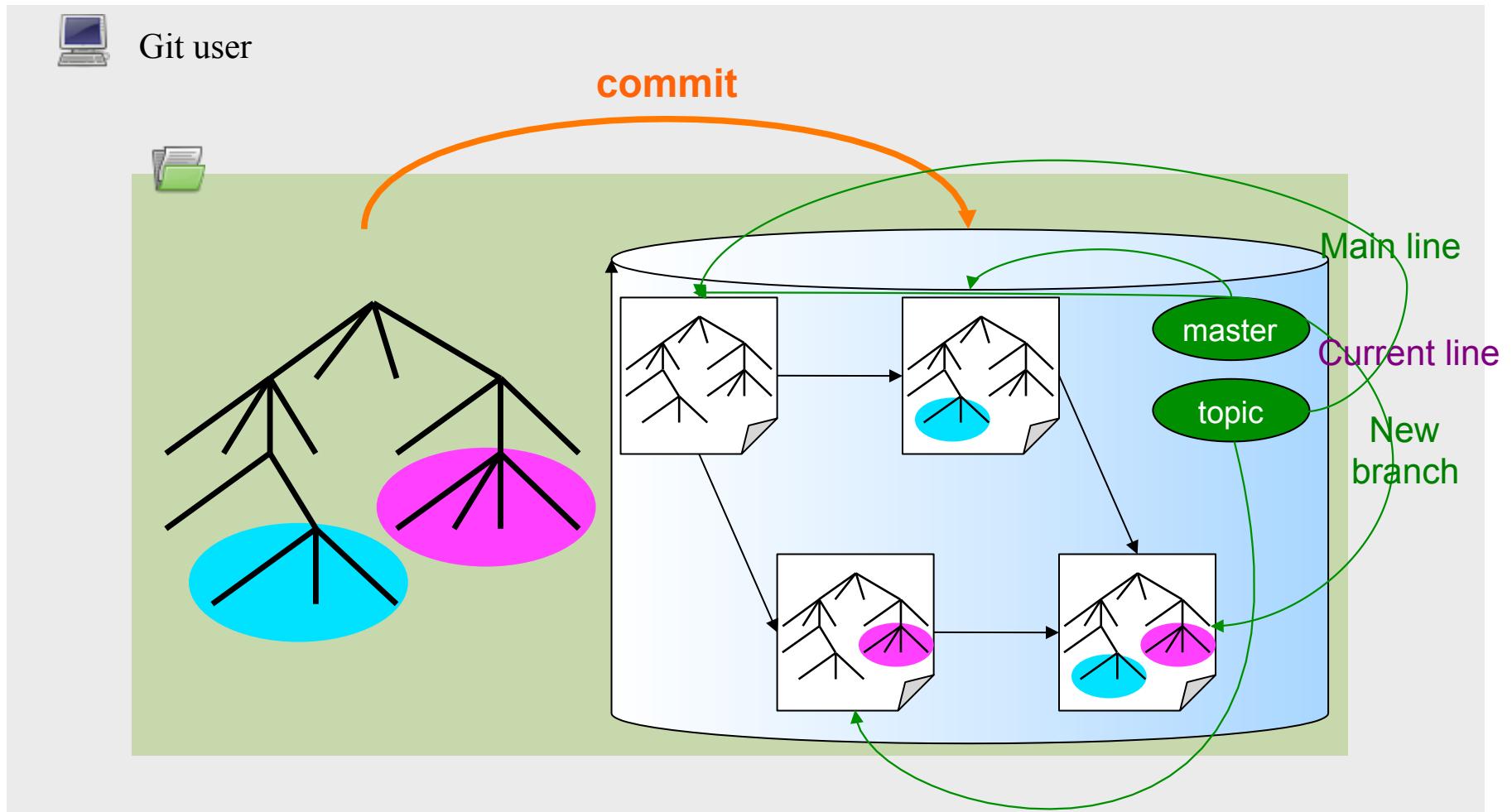
# Decentralized SCM

## > Git basic

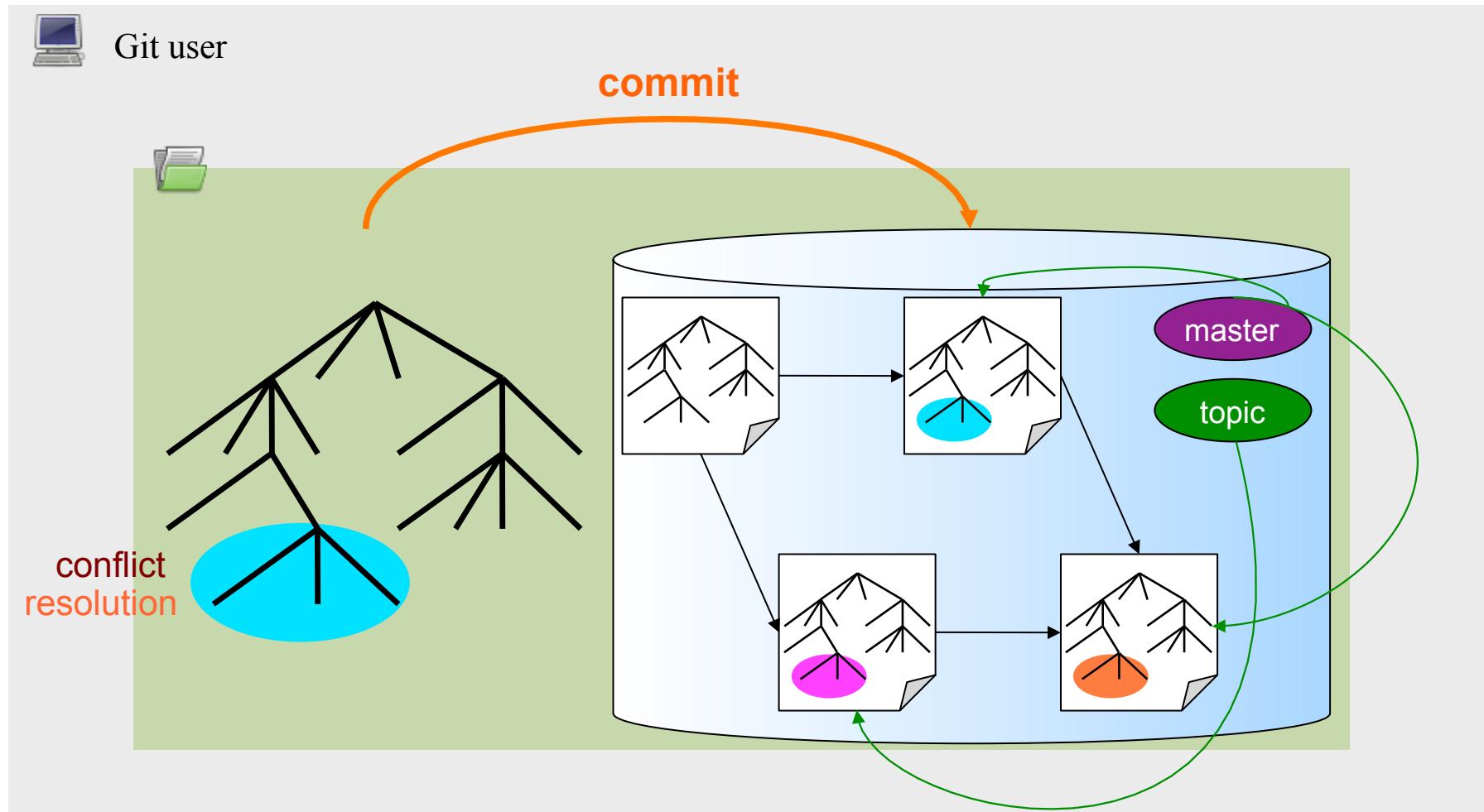
A series of vertical bars of varying heights and colors (yellow, red, white) are positioned along the left edge of the slide.

*Creatis*

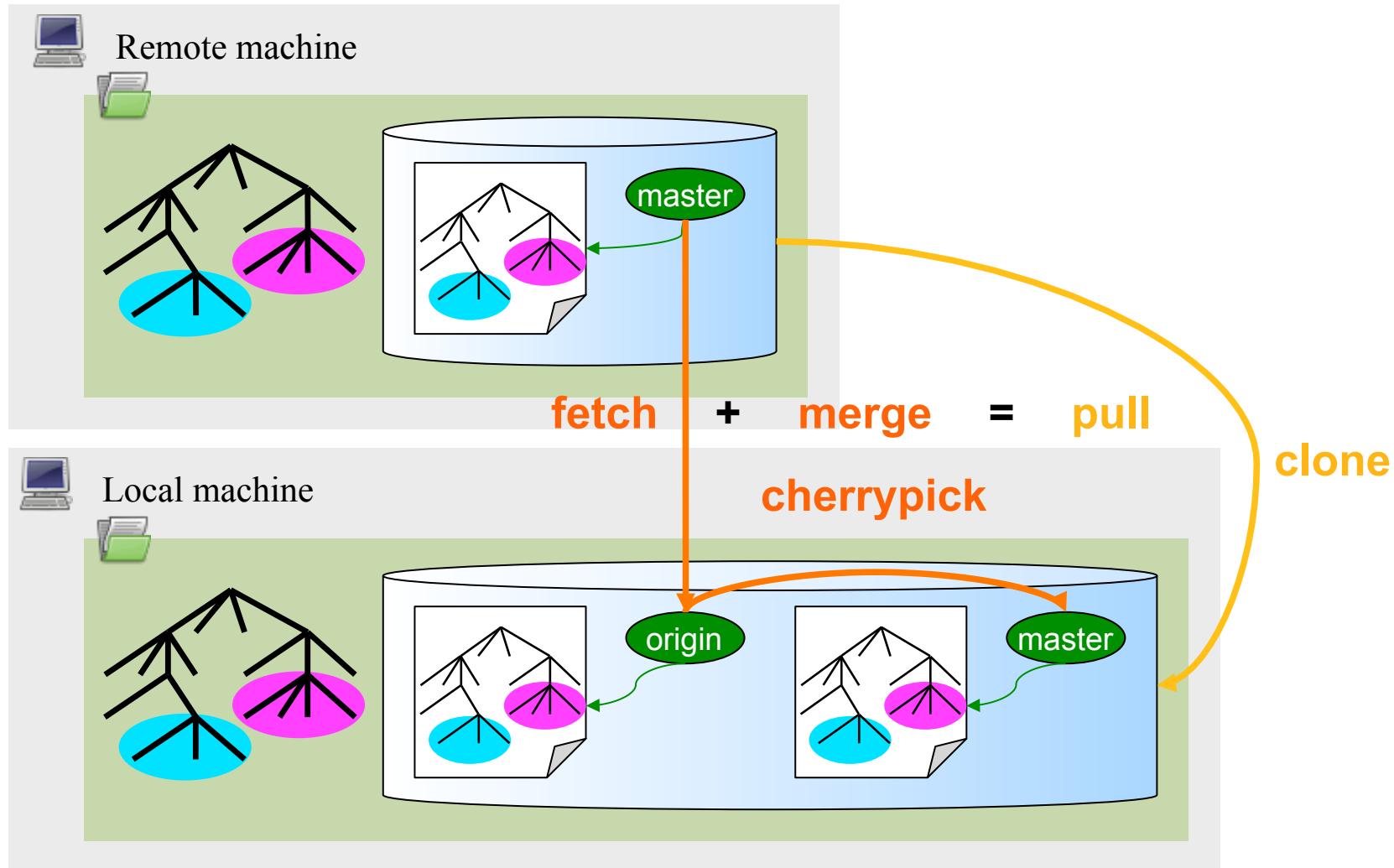
# Git > Local operations



# Git > Conflict

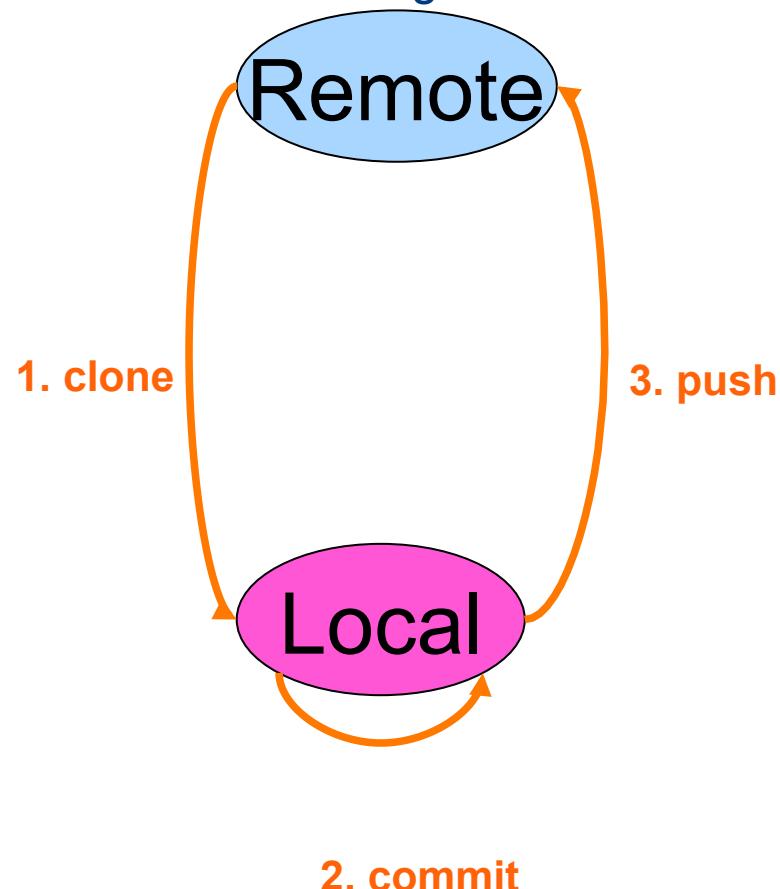


## Git > Distant operations



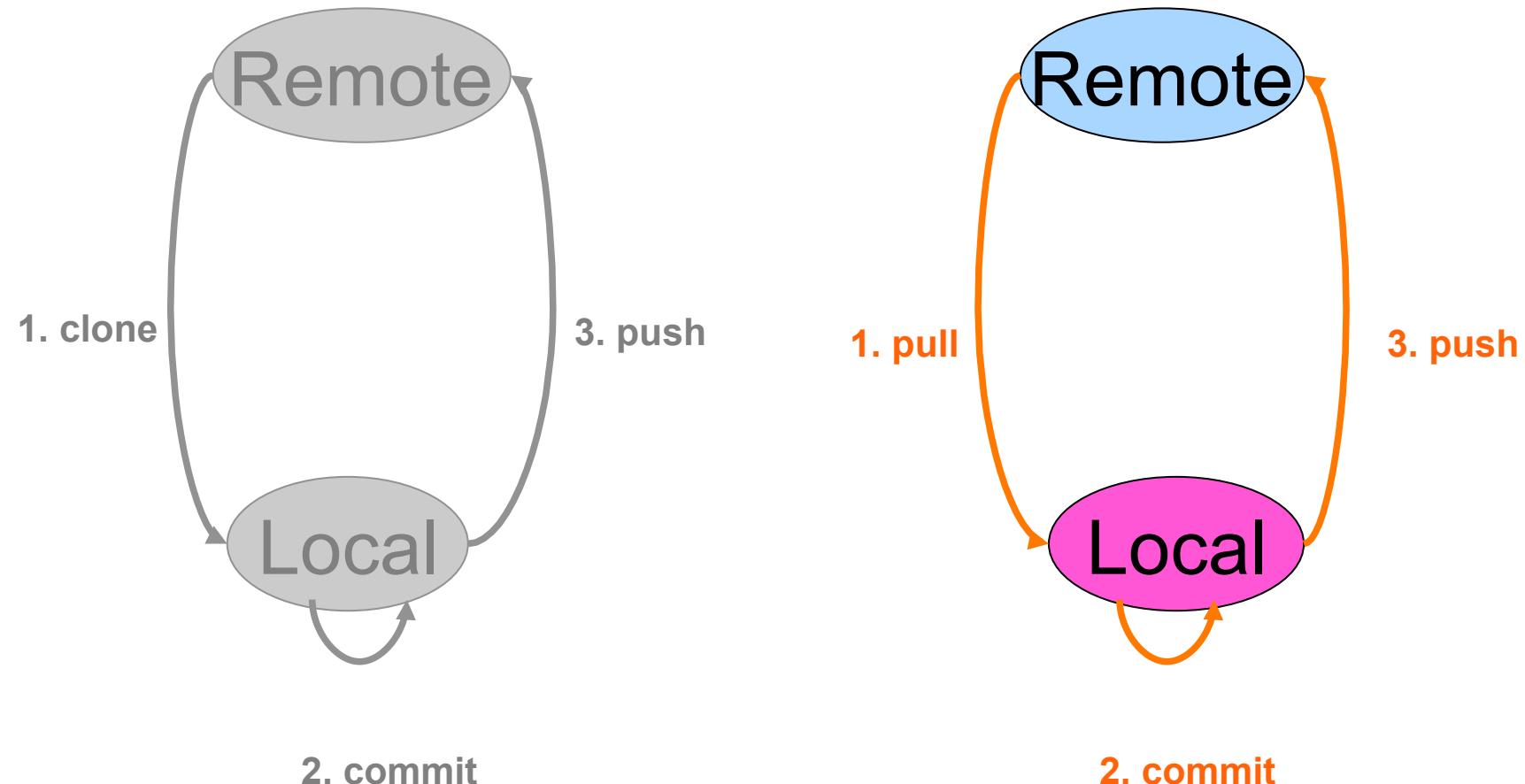
# Git > Workflow

> Classic working



# Git > Workflow

> Classic working





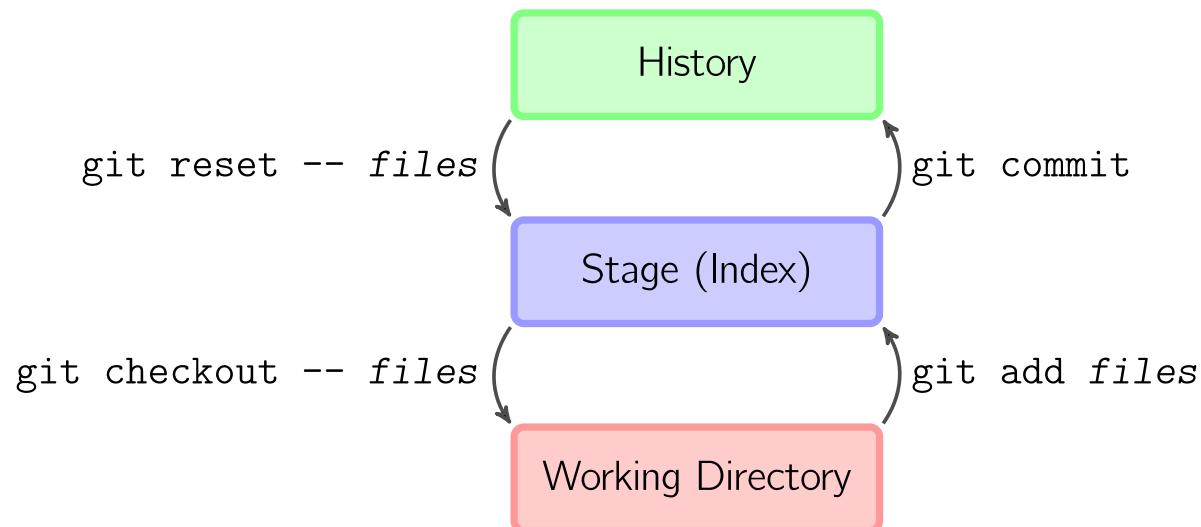
# Decentralized SCM

> Git more advanced

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

*Creatis*

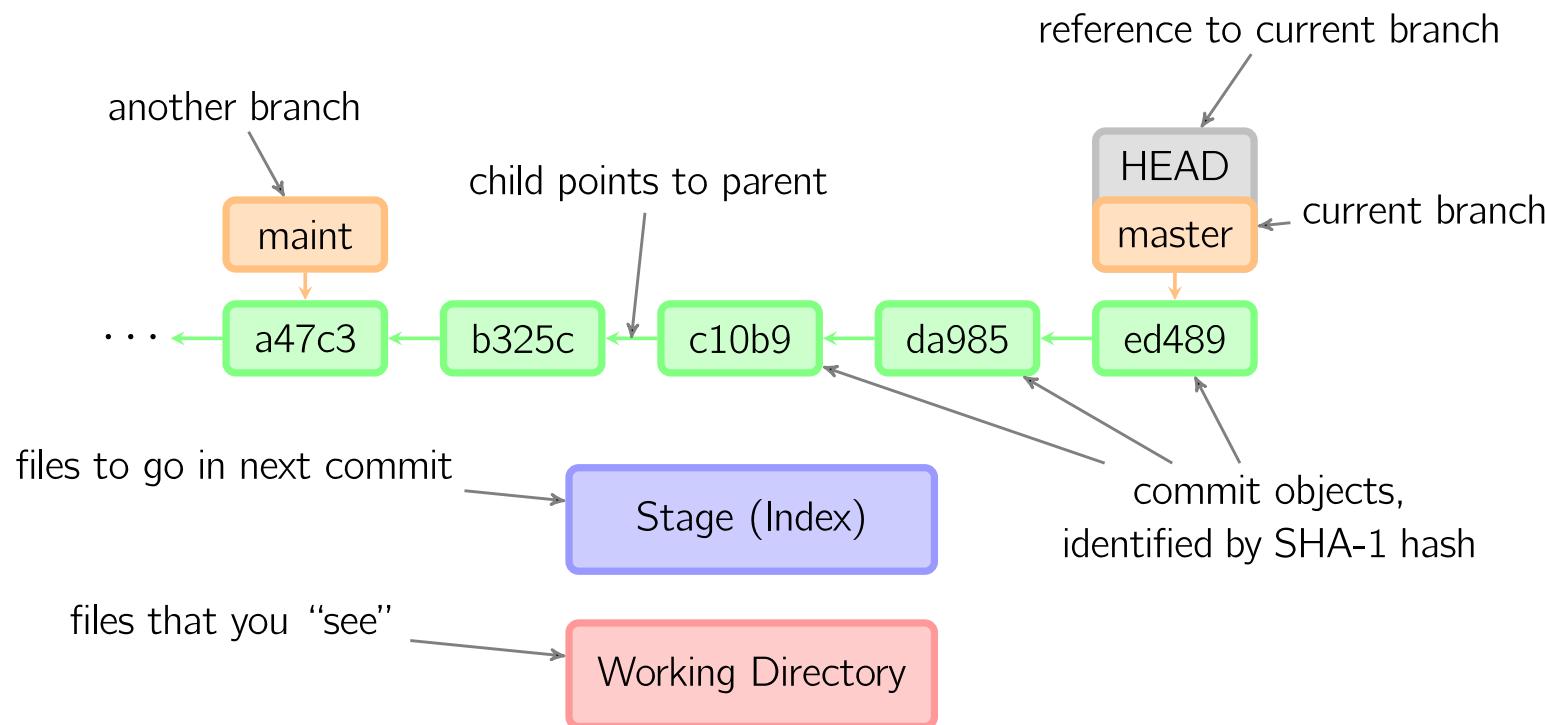
# Zones and transitions



Source :

<http://marklodato.github.io/visual-git-guide/index-en.html>

# Zones and transitions



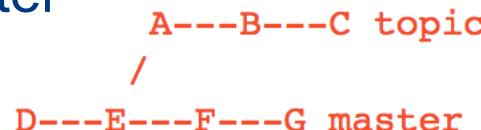
Source :

<http://marklodato.github.io/visual-git-guide/index-en.html>

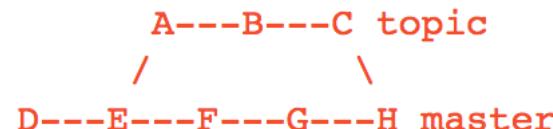
# Merge versus rebase > Merge

git-merge - Join two or more development histories together

Current branch is master



git merge topic



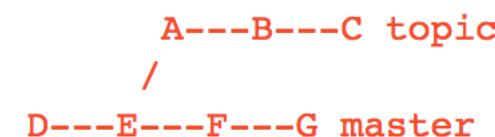
Replays the changes made on the topic branch since it diverged from master (i.e., E) until its current commit (C) on top of master, and records the result in a new commit along with the names of the two parent commits and a log message from the user describing the changes.

<https://git-scm.com/docs/git-merge>

# Merge versus rebase > Rebase

git-rebase - Reapply commits on top of another base tip

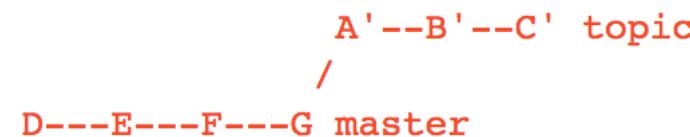
Current branch is topic



git rebase master

OR

git rebase master topic



The commit tree for the topic branch has been rewritten so that the master branch is a part of the commit history. This leaves the chain of commits linear and much easier to read.

# Merge versus rebase > *When use what ?*

*Don't use rebase ...*

If the branch is public and shared with others. Rewriting publicly shared branches will tend to screw up other members of the team.

*Use rebase ...*

When the exact history of the commit branch is important (since rebase rewrites the commit history).

*Advice*

Use rebase for short-lived, local branches ;

Use merge for branches in the public repository.

[http://gitimmersion.com/lab\\_34.html](http://gitimmersion.com/lab_34.html)

# Merge > Conflict management

*CONFLICT (content): Merge conflict in <nom-fichier>*

*# Unmerged paths:*

*# (use "git reset HEAD <some-file>..." to unstage)*

*# (use "git add/rm <some-file>..." as appropriate to mark resolution)*

*#*

*# both modified: <some-file>*

**Fix** the conflict by editing <some-file> (look for <<<< ===== >>>>)

*git add <some-file>*

*git commit -m « Merged master fixed conflict »*

# Cherrypick

- Selects changes to apply from a branch to another one

# How to get back ? > *When use what ?*

If you haven't pushed your work yet :

*git commit --amend*

To modify the last commit.

*git reset HEAD^*

To cancel the last commit. The modifications will remain in the working directory.

*git reset --hard HEAD^*

To remove the last commit and the modifications of the files.

# How to get back ? > *When use what ?*

If you haven't pushed your work yet :

*git checkout file\_name*

file\_name will be at its state in the last commit.

*git reset HEAD -- file\_to\_remove\_from\_the\_index*

To revert the *git add* on file\_to\_remove\_from\_the\_index.

*git rebase -i HEAD~3*

To modify the message commits, their order or their number.

# How to get back ? > *When use what ?*

If you have already pushed your work :

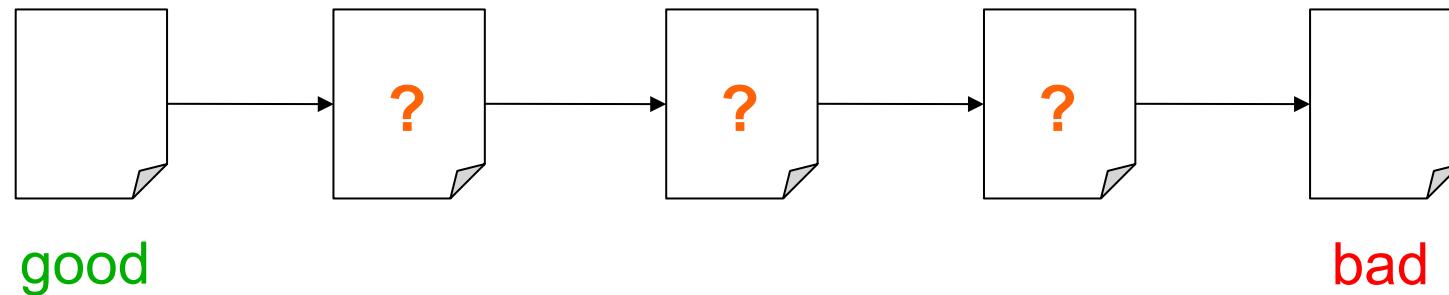
Do not modify the history !!!

*git revert 6261cc2*

To create the inverse commit of *6261cc2*.

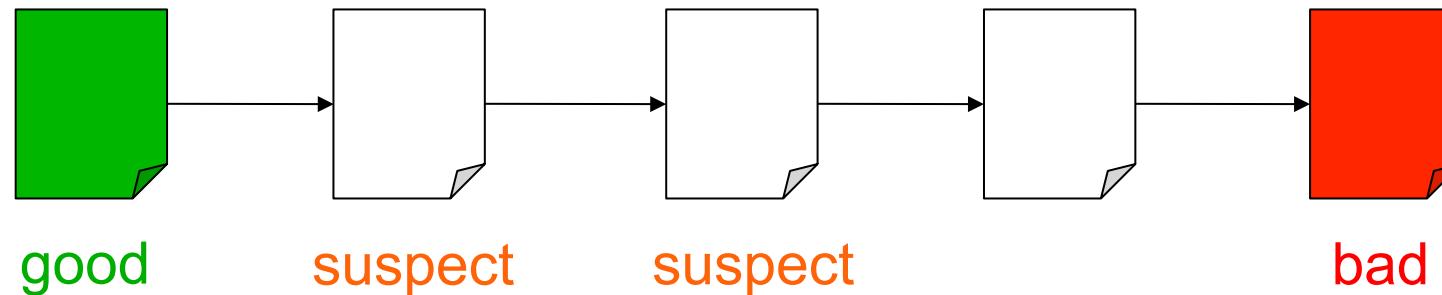
## Git > Bisection

Finds the commit introducing a bug by dichotomy



## Git > Bisection

Finds the commit introducing a bug by dichotomy



## Git > Submodule

*Link your project and its dependencies.*

A submodule is a repository seen as a subdirectory of another repository.

`git submodule add git://github.com/chneukirchen/rack.git rack`  
-> the project Rack is in the subdirectory rack

The containing project keeps track of the commit of the submodule : it helps collaborators recreating the same environment.

All Git commands work independently in the two repositories.

# Git > Submodule

Reference documentation:

<https://git-scm.com/book/fr/v1/Utilitaires-Git-Sous-modules>

Good practice:

<http://www.git-attitude.fr/2014/12/31/git-submodules/>

Tutorial:

<https://git.wiki.kernel.org/index.php/GitSubmoduleTutorial>

Wikibook:

[https://fr.wikibooks.org/wiki/Git/Sous-modules\\_et\\_Super-projets](https://fr.wikibooks.org/wiki/Git/Sous-modules_et_Super-projets)

## Git > Submodule

Submodules generate complexity and confusion.

Synchronization is not always obvious... Avoid submodules!

# Git > Hooks

*Scripts in .git/hooks to be launched on specific events.*

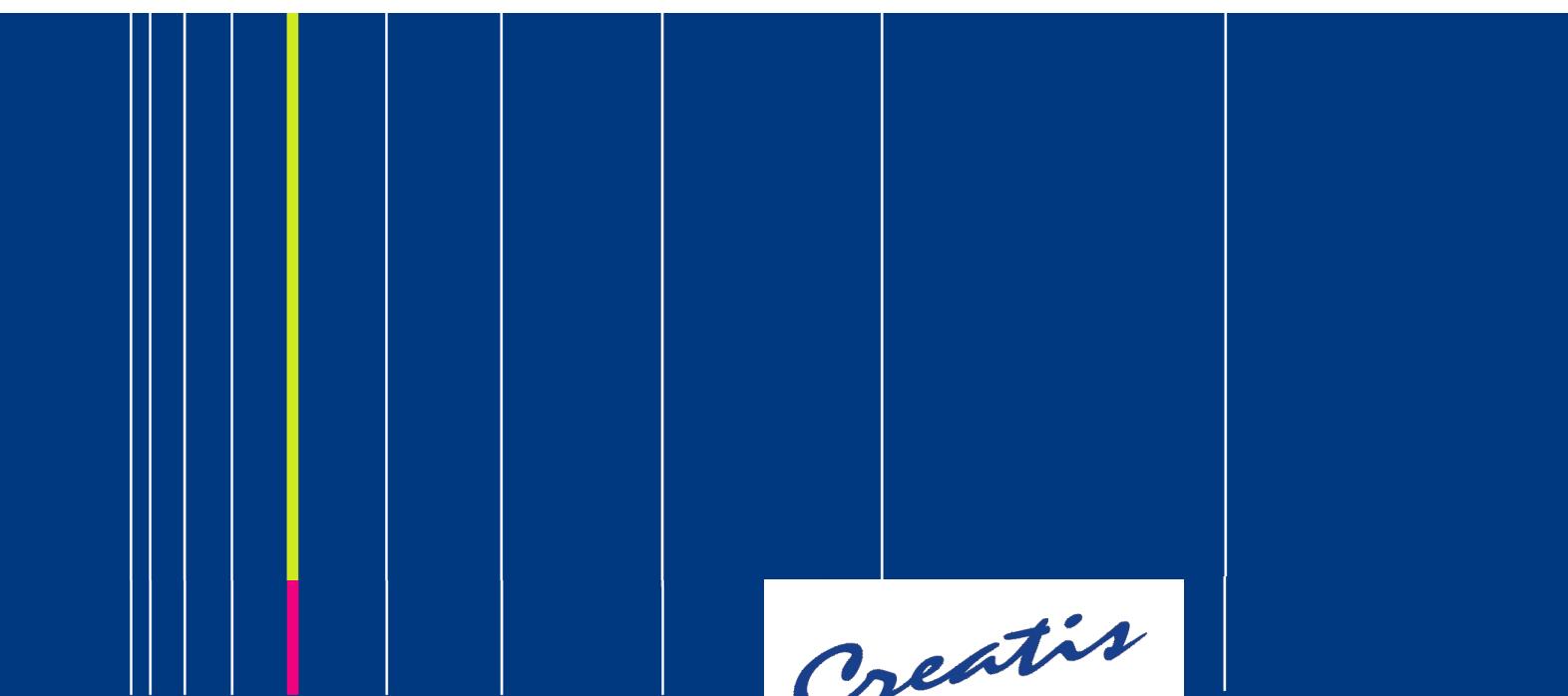
- **Client-side hooks:** on events such as commit or merge
  - *pre-commit*: to run test, to format the code or check the doc
  - *prepare-commit-msg*: to edit the default commit message
  - *commit-msg*: to check commit message compliance to a pattern
  - *post-commit*: for notification
  - *post-checkout, post-merge, pre-rebase, pre-push*
- **Server-side hooks:** on events such as reception of a pushed commit
  - *pre-receive*: access control, no non-fast-forwards
  - *update*: similar (pre-receive runs only once, whereas update runs once per branch)
  - *post-receive*: notification to a list by e-mail, to a continuous integration server, to update a ticket-tracking system

<https://git-scm.com/book/it/v2/Customizing-Git-Git-Hooks>

<https://fr.atlassian.com/git/tutorials/git-hooks/>

# 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, 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

- SmartGit (Windows, Linux, MacOS)
- TortoiseGit via Putty (Windows)
- qgit (Qt)
- Gitk (Windows, Linux, Mac OS)

## Users

- Linux kernel
- Wine
- X.org
- Android
- Kitware

## Useful links

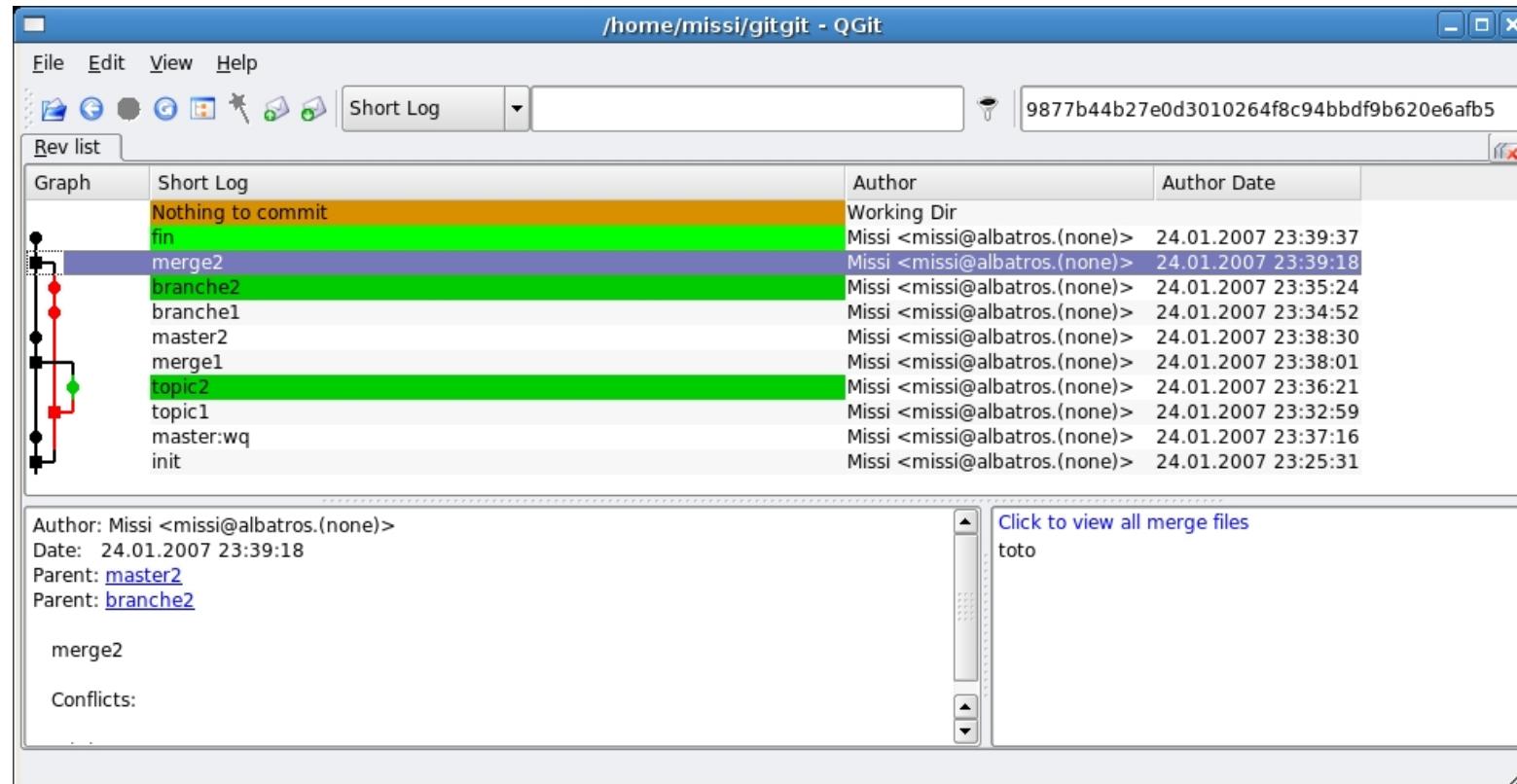
- <http://git-scm.com/>

# Around Git > qgit

Browse revisions history

View patch content and changed files

Graphically follow different development branches



## 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 top right corner features a green "git" logo with three red '+' symbols. Below the address bar is a search bar with a placeholder, a checkbox labeled "re", and a "Search" button. A red link labeled "List all projects" is centered above a table. The table has four columns: "Project", "Description", "Owner", and "Last Change". Each row represents a project with its details and links to various logs and summaries.

Project	Description	Owner	Last Change
CreaPhase.git	Propagation-based phase contra...	Loriane Weber	2 months ago
FrontAlgorithms.git	Generic implementation of...	Maciej Orkisz	2 days ago
GRIDA.git	Grid Data Management Agent	Sorina Pop	No commits
bbtk.git	Black Box Tool Kit	Eduardo Davila	6 days ago

# Around Git > *online visualization of your repository*

Gitolite CREATIS example :  
The log for one project

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

summary | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#) commit [?](#) search:   re

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)

**shortlog**

5 days ago	Leonardo Flórez...	...	<a href="#">master</a>	<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>



Git : le fil d'Ariane de vos projets, pilier des forges modernes - ENVOL 2016 - 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>
parent	<a href="#">86a6d5df2aa1aa5292a5fa851d98bfc13939bdf3</a>

[tree](#) | [snapshot](#)  
[commit](#) | [diff](#)

---

...

---

<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 - ENVOL 2016 - 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)

---

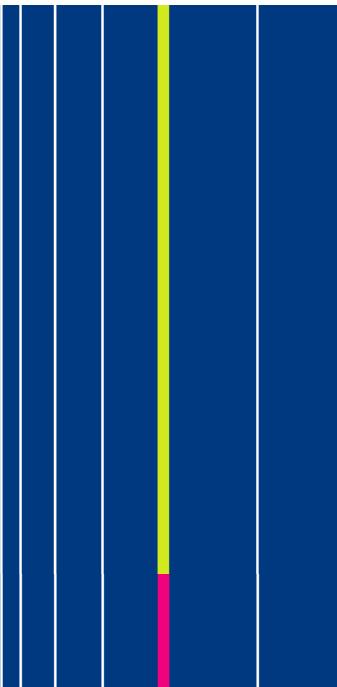
lib/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)  
lib/fpaInstances/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)  
plugins/CMakeLists.txt      [patch](#) | [blob](#) | [history](#)

---

```
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 - $RCSfile$
```

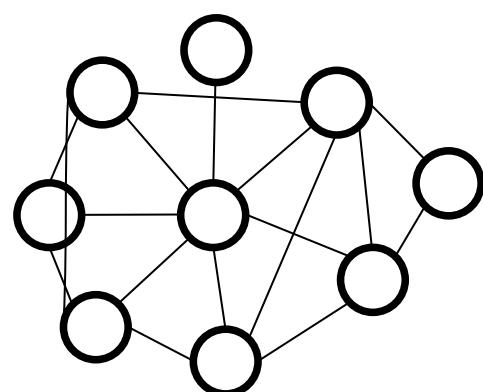
# Conclusion

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

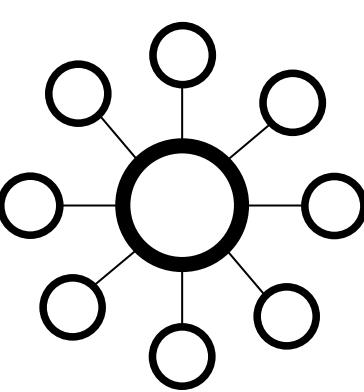
Creatis

# Conclusion

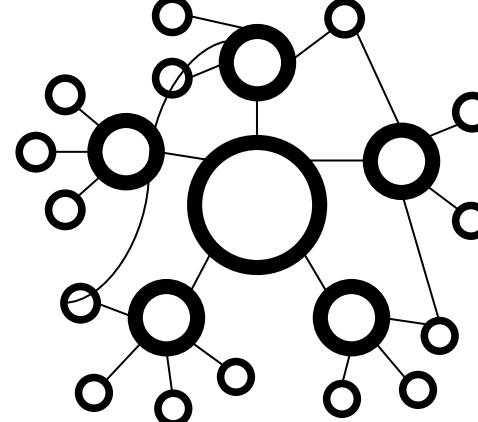
- A decentralized SCM remains a **tool**
  - No default usage policy
  - Policy to be defined
    - From centralized to decentralized
    - Pull-only vs shared-push



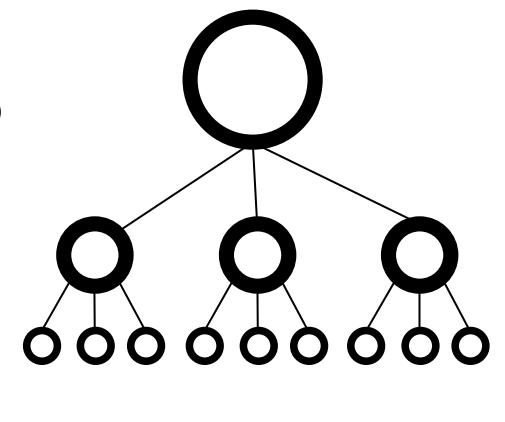
Anarchic



Centralized



Linux kernel model

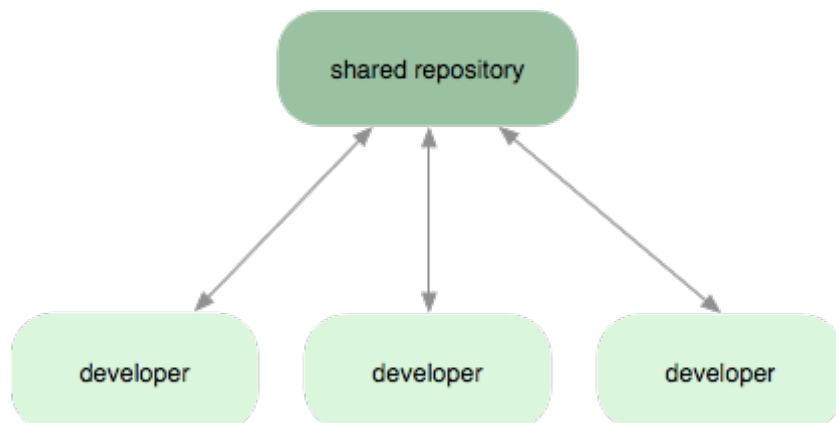


Branch by functionality  
or  
Release train

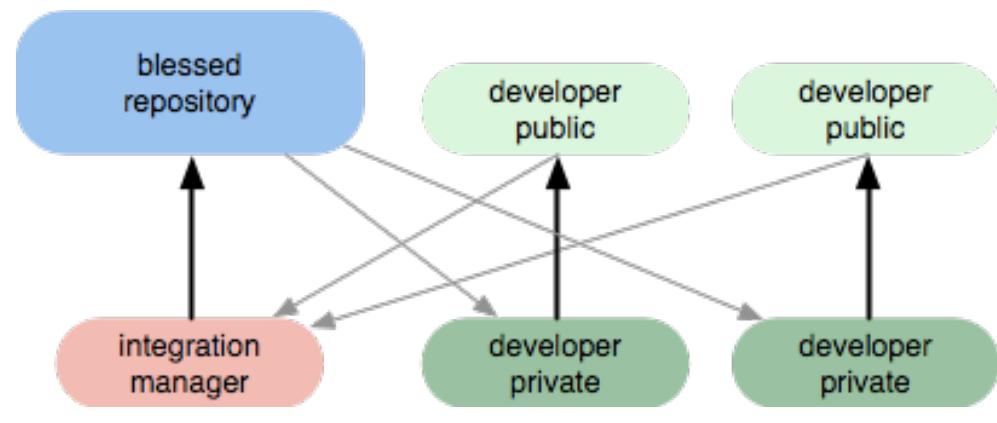
# Conclusion

- 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



# Example 1, with CVS or SVN

- We are working on an improvement, not already ready to be published
  - A bug is reported, it needs to be fixed quickly
  - How would you do with CVS or SVN?
- 
- Do a second checkout, fix the bug there and update the first checkout in which we developed

OR

- Save the current patch (`cvs diff > file`), restore the unmodified version, fix the bug, reapply the patch in the hope that conflicts will be avoided...
  - Do not forget the removed/added files
  - Not easily scalable...

# Example 1, with Git

- We are working on an improvement, not already ready to be published
  - A bug is reported, it needs to be fixed quickly
  - How would you do with Git?
- 
- Quickly *commit* the current state of the improvement
    - It will be modified later
  - Create a *branch* of the public version and *checkout* to it
  - Fix the bug and *push* the commit
  - *Checkout* to get back to the initial branch
  - *Merge* to get the bug fixing if necessary

## Example 2, with CVS or SVN

- We are working on a new feature in the plane or in the train
- Save intermediary versions
  - cvs diff > patch1\_doingX\_butnotY
- When back on-line, commit the different steps of the work
  - cvs up to the revision at the working time
  - Apply a patch
  - cvs up
  - Resolve conflicts, especially if new files were added
  - cvs commit
  - Do it again for each patch...

## Example 2, with Git

- We are working on a new feature in the plane or in the train
- Save intermediary versions
  - git commit
- Modify a previous version
  - git commit –amend
- Publish the different steps of the work
  - git push

# Example 3

- We maintain a modified version of a software A
- How to update from a new version of A?
- With CVS or SVN:
  - Find the latest revision X that has been retrieved
  - Find the current revision Y that we want to retrieve
  - Merge between X and Y
  - cvs commit
- With Git:
  - git pull A Y
    - Automatically finds out what should be merged

# Example 4

- We would like to test a modification on several machines
- With CVS or SVN:
  - We create a branch for a single (or some) commits ?
  - We send patch(es) manually to each machine?
    - Do not forget cvs up each time
- With Git:
  - Create a branch and git push/pull from/to each machine
    - A central repository is useless

# SVN to Git migration tools

## Subversion

When choosing git migration tools you need to clearly grasp the difference between an *gateway* (supporting live operation on a Subversion repository through git) and an *importer* (designed to move an entire Subversion history to git). The programs in this section were usually designed for one of these purposes and may have serious hidden flaws if used for the other.

Importers and gateways are listed first, then exporters, then auxiliary tools. See [git-svn](#) on how to use Git as an Subversion client. Here is a feature matrix of the production-quality importers:

	reposurgeon	git-svn	svn2git	SubGit	agito
role?	importer	gateway	importer	gateway	importer
handles branching?	yes	yes	yes	yes	yes
makes annotated tags?	yes	no	yes	yes	yes
splits mixed-branch commits?	yes	no	no	yes	yes
makes .gitignore files?	yes	no	no	yes	yes
documentation	excellent	good	good	excellent	passable
written in	Python	Perl	Ruby	Java	Python
last activity?	2015	2012	2012	2013	2012
maintainer	Eric S. Raymond	Eric Wong	James Coglin, Kevin Menard	TMate Software	Simon Howard

[https://git.wiki.kernel.org/index.php/Interfaces,\\_frontends,\\_and\\_tools#Subversion](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Subversion)

# 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

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`

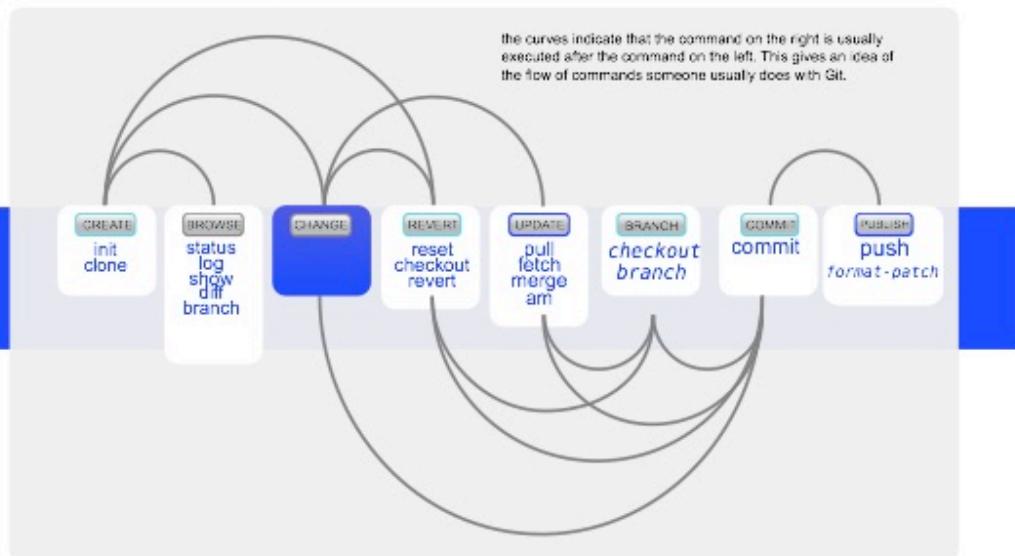
Merge branch1 into branch2  
`git checkout $branch2  
git merge $branch1`

Create branch \$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 someone 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  
git rebase --continue  
(do for all resolved files)
```

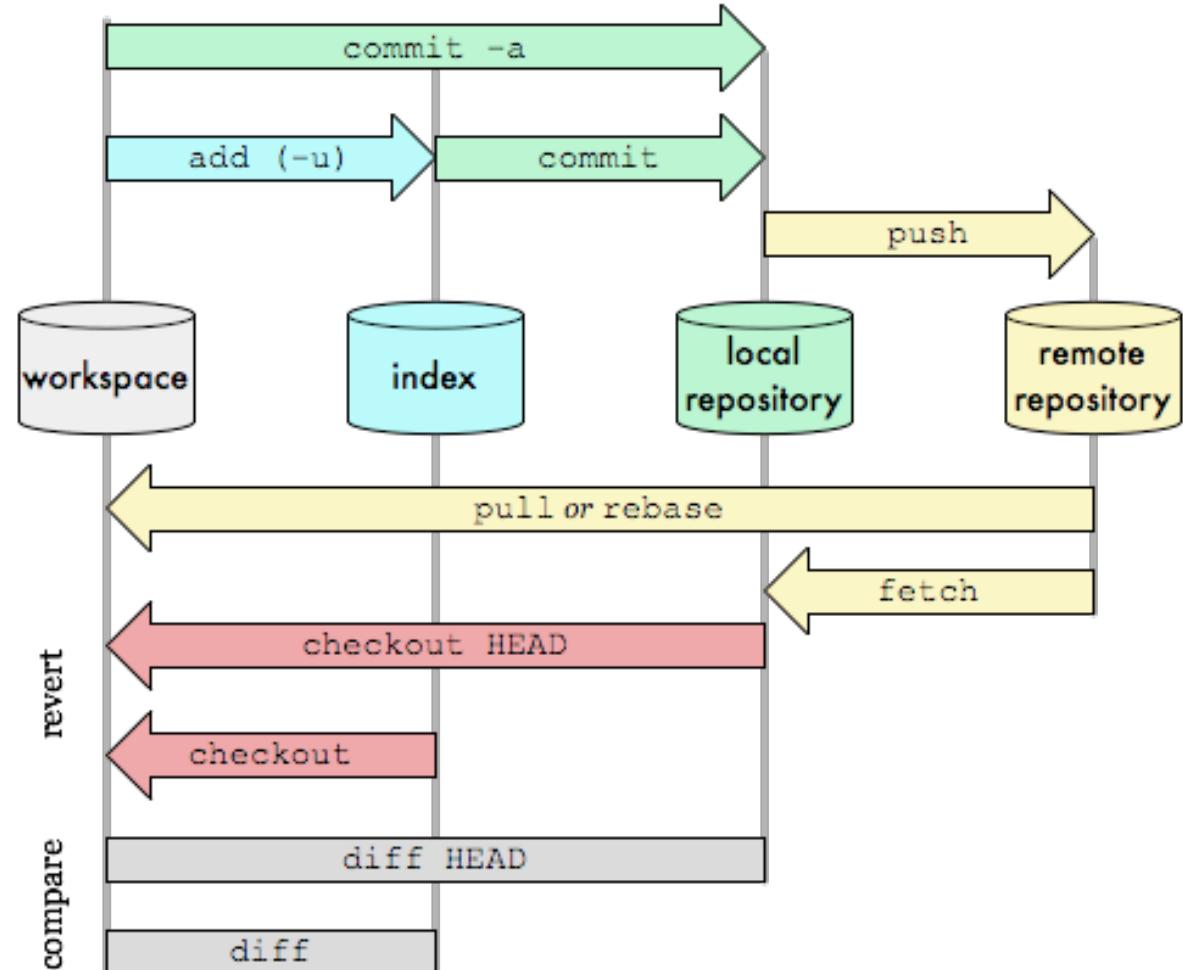
## Cheat Sheet Notation

\$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

# Git Data Transport Commands

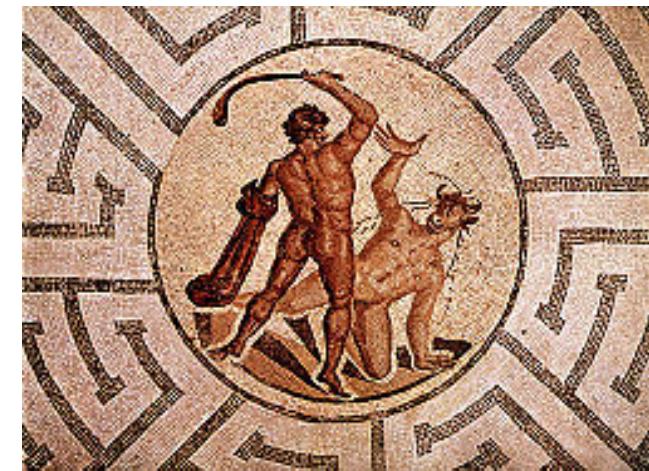
<http://osteelle.com>

10  
0



Source :

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



Git : le fil d'Ariane de vos projets, pilier des forges modernes - ENVOL 2016 - Claire MOUTON

# Un moment pour échanger !

# Un moment pour échanger !

Et toi, tu fais comment pour contrôler les versions de ton code ?!?

- Pas de gestionnaire de version ?
- Un autre que Git ?
- Quelle utilisation de Git ?
- Votre migration ?

Vos pratiques ?

Vos retours d'expérience ?

Des erreurs à ne pas faire / des conseils ?

Des compléments à ma présentation ?

# TP !

<https://ccwiki.in2p3.fr/developpements:formation:git>