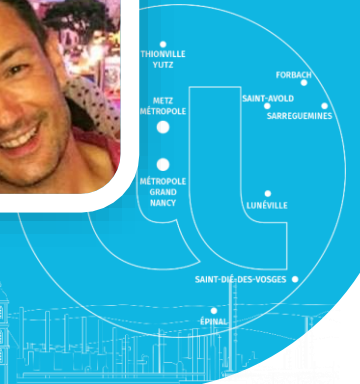


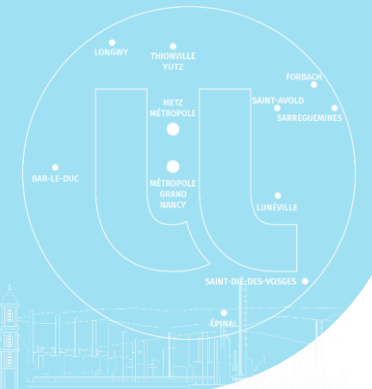
Collaborative work tools



Yann MORERE
LCOMS – SciFA – Université de Lorraine

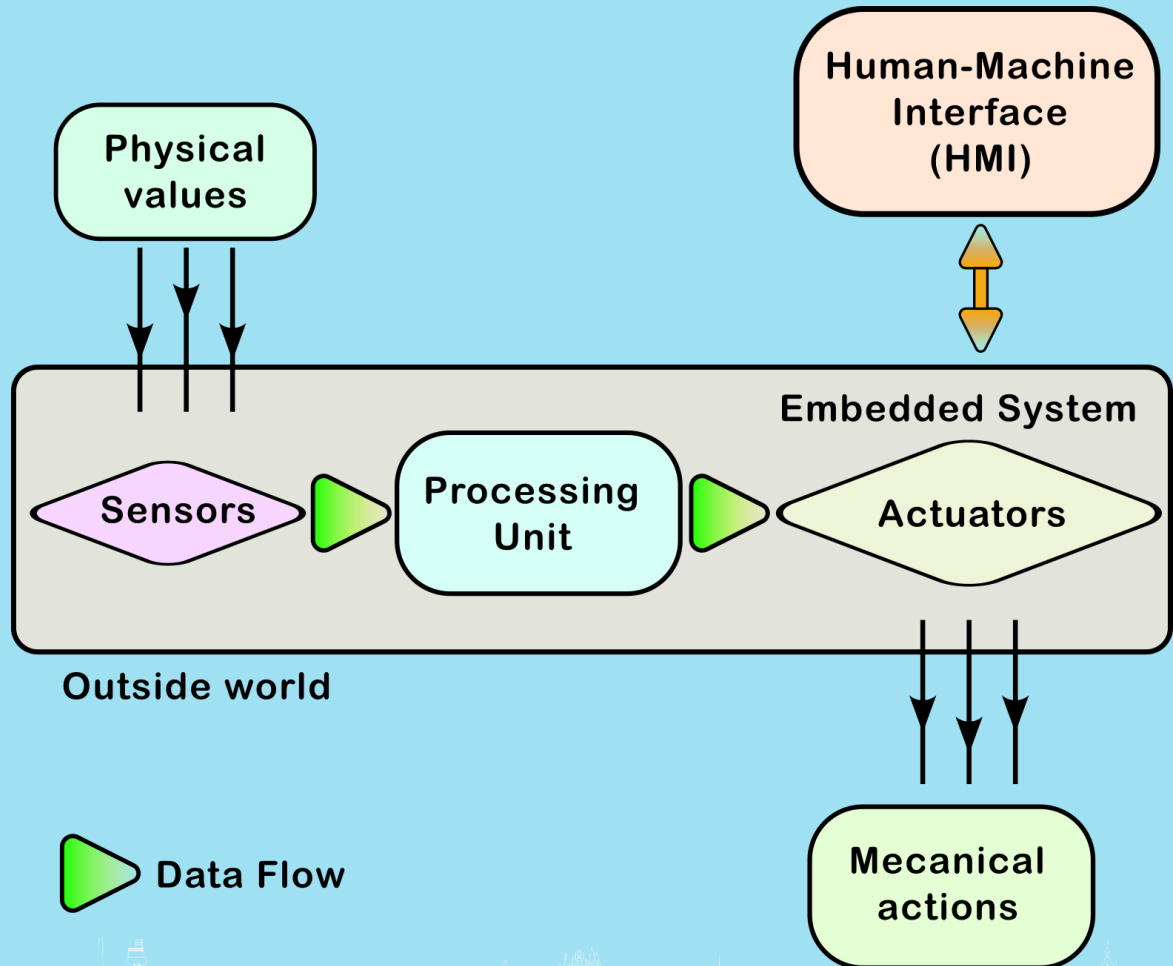


- Understand:
 - Project versioning
 - The basic concepts of git
 - Collaborative work
 - Project management using gitlab
- During the practical work, learn to use:
 - The basic git commands (workstation side)
 - The main features of gitlab (server side)
 - Good practices

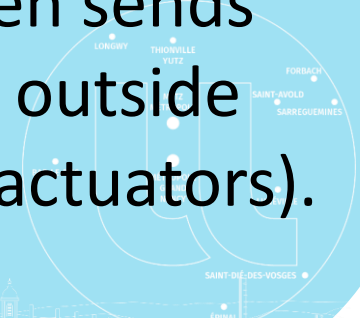


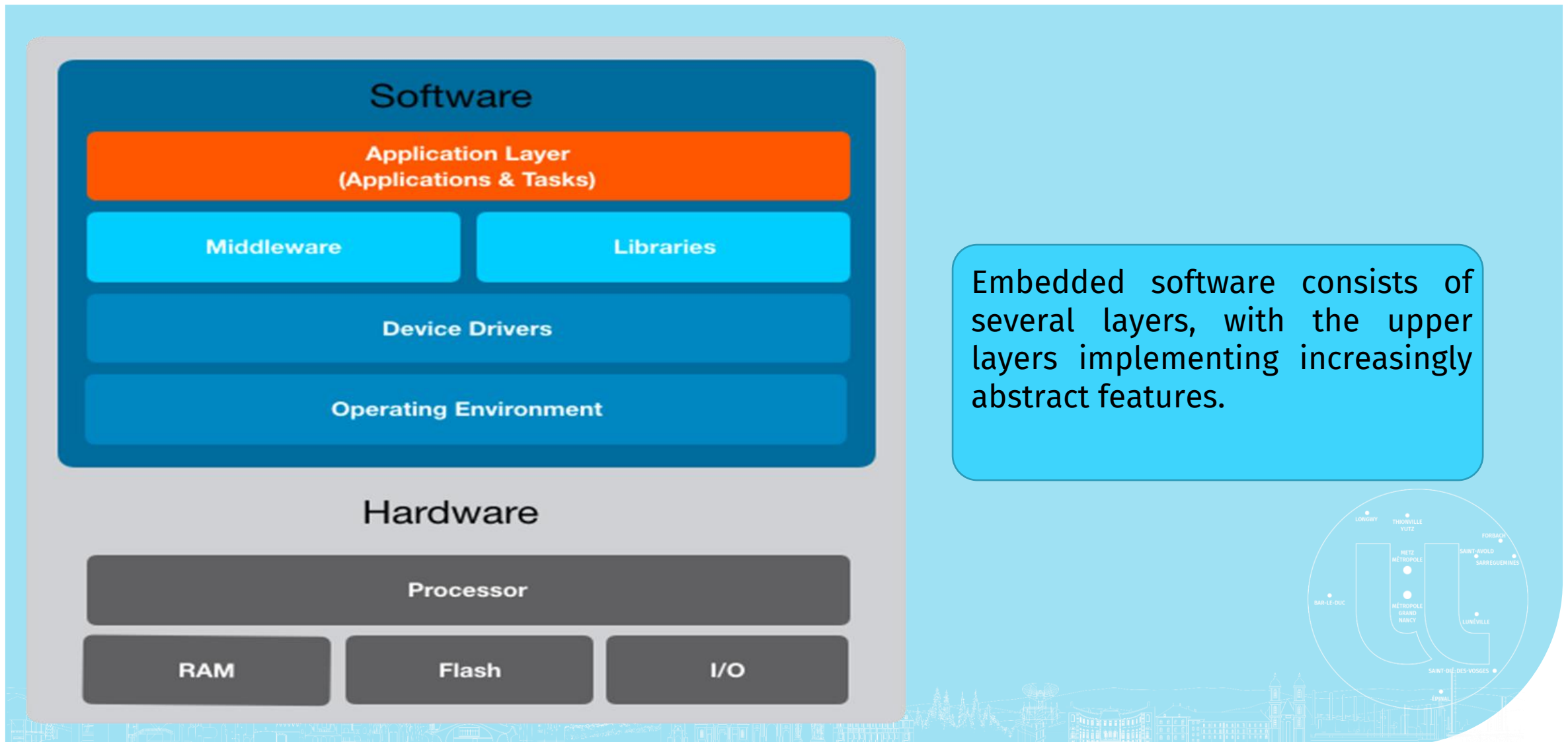
- Why a collaborative tool?
- History of VCS
- The concepts of git
- Gitlab features





- Definition: An embedded computer system receives information from the outside world by means of **sensors** (or sensors). **Its programmable board stores and processes this information by the processing unit** (the microprocessor) and then sends information back to the outside world via **actuators** (or actuators).

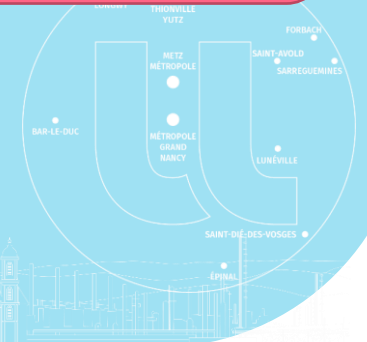


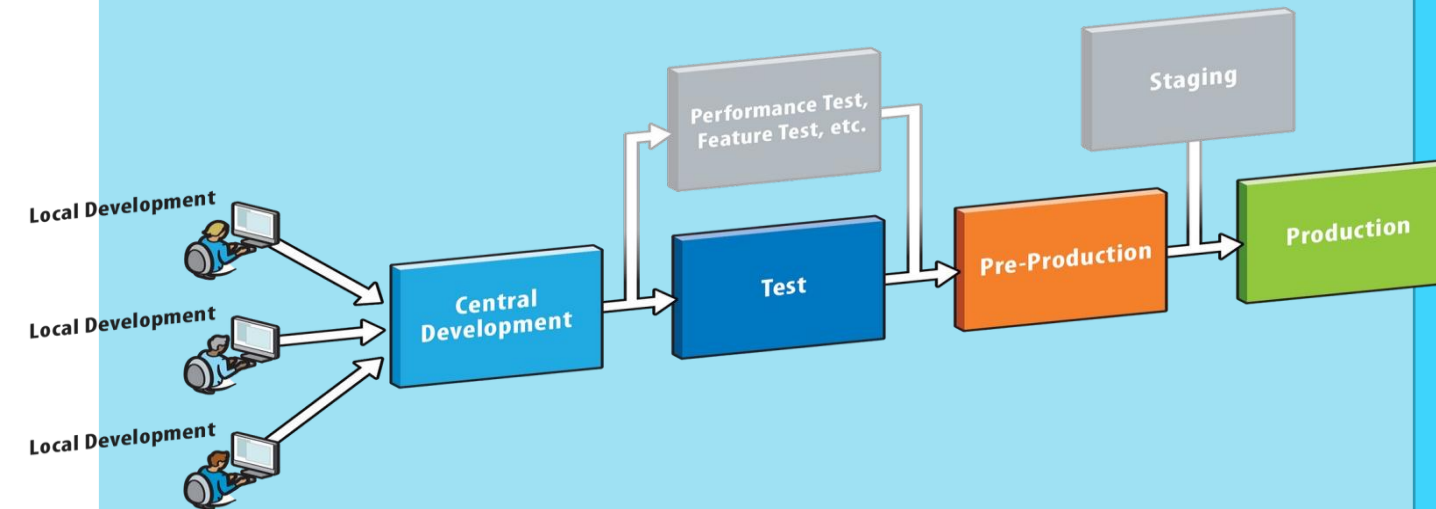




The **life cycle of a software component** can be divided into 4 phases:

- The creation of the tool (design, production and deployment)
- Its use by users (administration, use)
- Its update and evolution (maintenance)
- The decision to replace the tool (end of life)

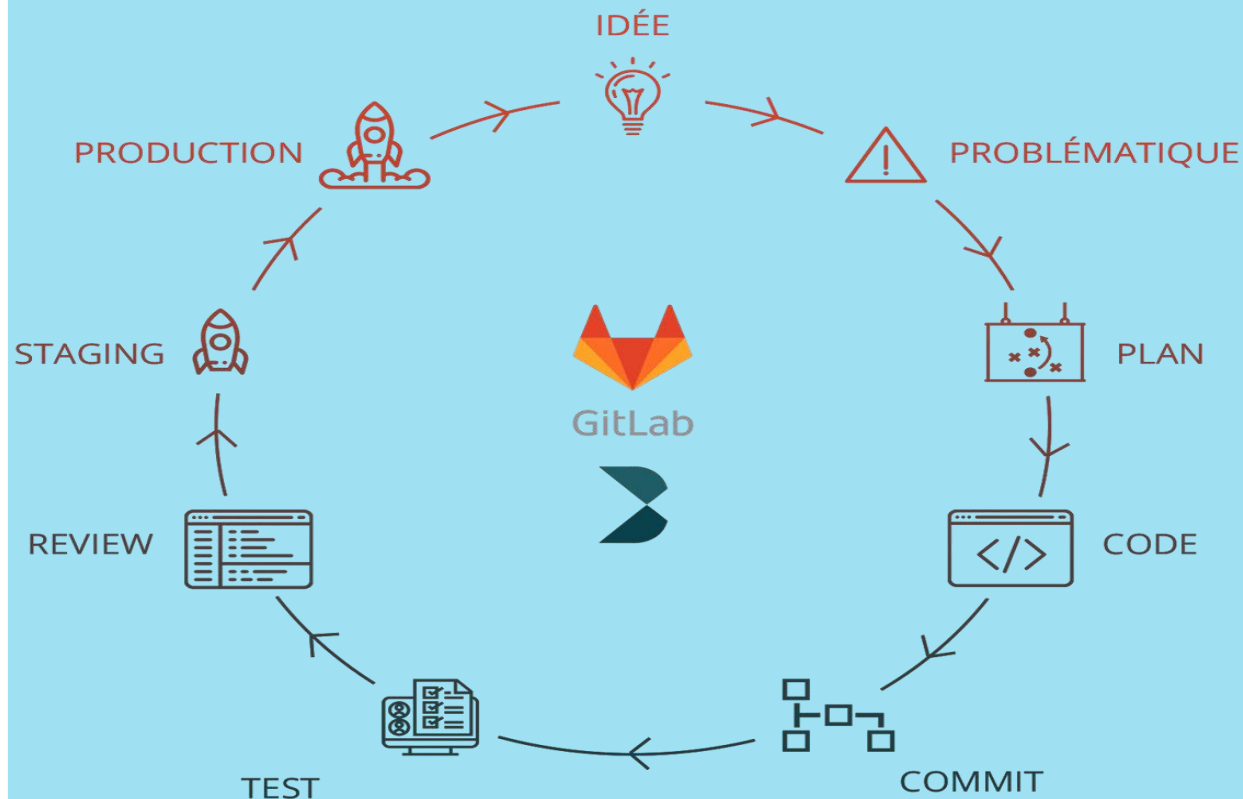




The 6 possible environments:

- **Development** also called developer's computer
- **integrated development** a remote environment used by developers to integrate their code so that they can check if they have broken someone else's code, or if someone else has broken their code
- **Test Environments:** Environments used for manual and/or automated testing
- **staging environment** used for acceptance testing before pushing the new
- **Canary environment**, a special type of environment used to test new features with a small group of users (commonly used to test things internally before reaching a wider audience)
- **Production environment** The environment that our users actually use





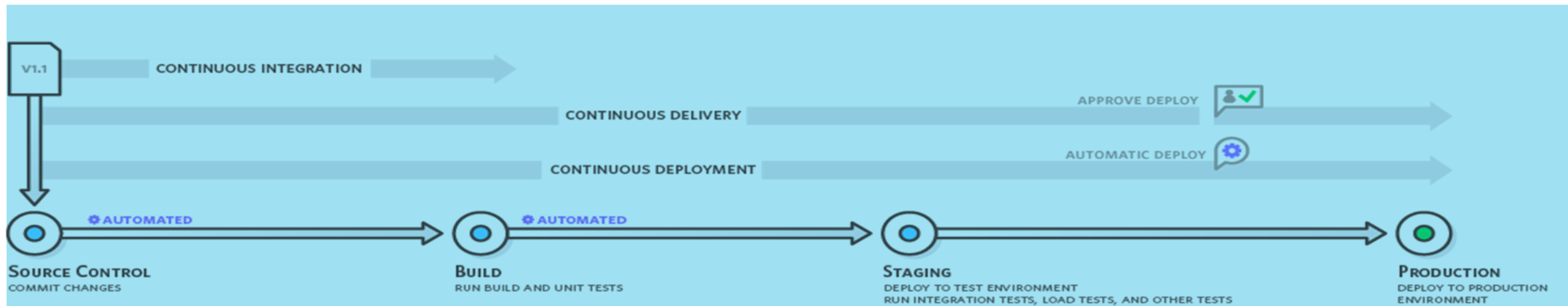
A **collaborative tool** such as GitLab allows everyone to follow the project, from the end customer who knows nothing about code to the developers of the application.

This makes it possible to manage the entire creative process from the customer idea to the delivery of the product.

Once the expression of need has been issued and the development plan has been validated, the code is developed and posted on Gitlab.

Each modification is made in the form of a "commit" and thus gives rise to tests of the code for all users. Depending on the anomalies detected, it is reviewed "review".

The validated code is then put into pre-production "staging" to finally be deployed.

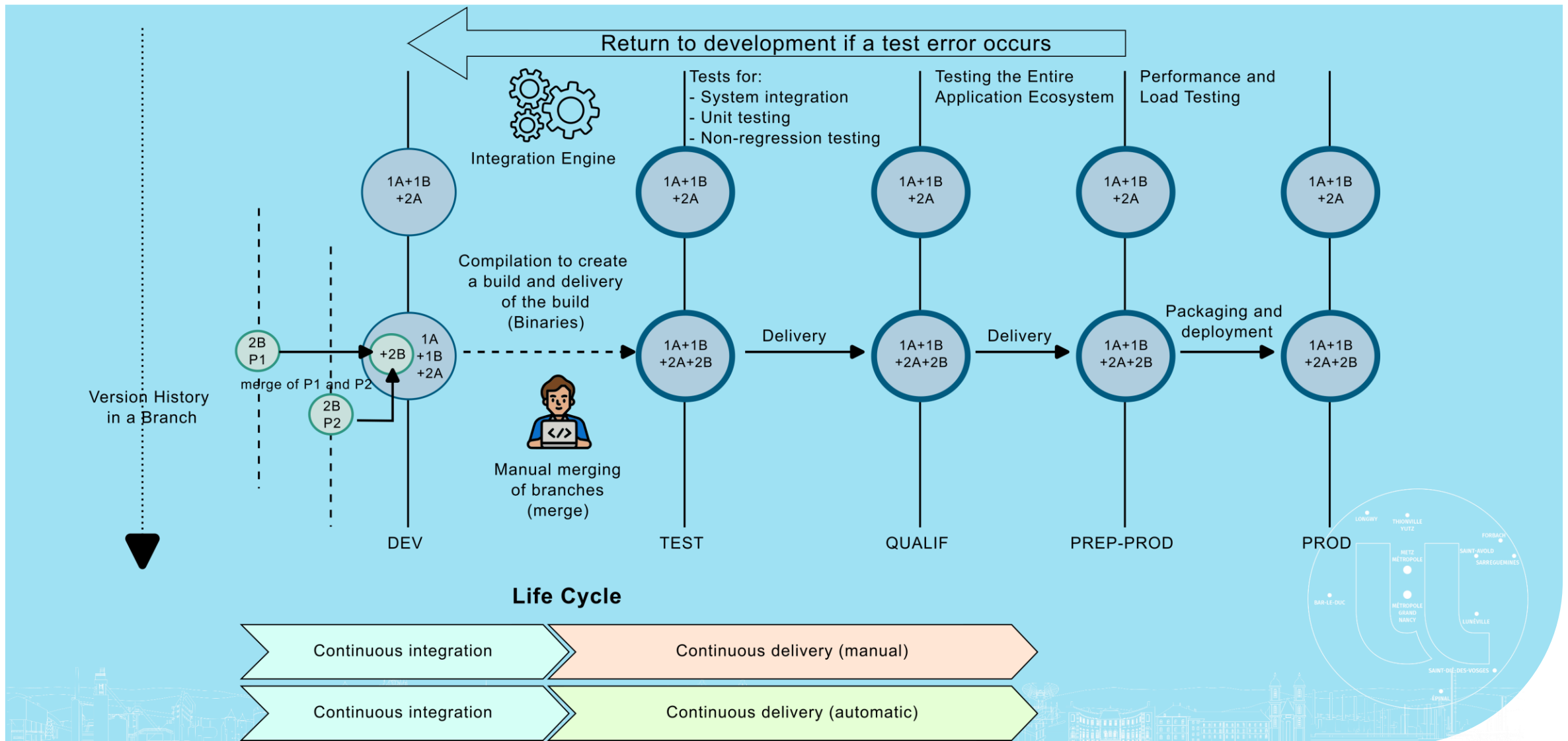


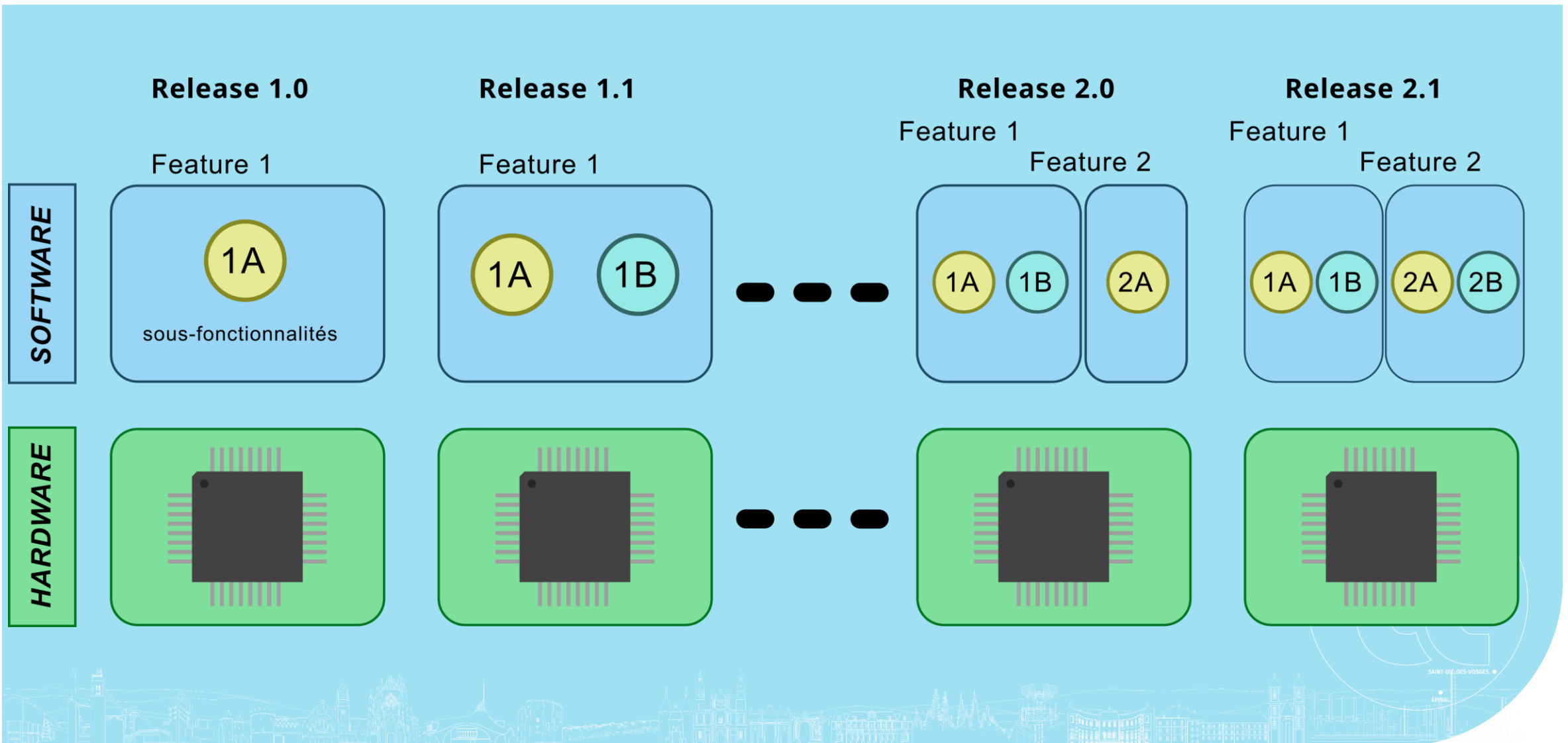
Continuous integration refers to the unit creation and testing stages of the software release process. Each revision applied triggers an automated creation and testing process.

With continuous delivery, code changes are automatically applied, tested, and prepared for production.

Continuous delivery extends the principle of continuous integration by deploying all code changes to a test environment and/or production environment after the build stage.



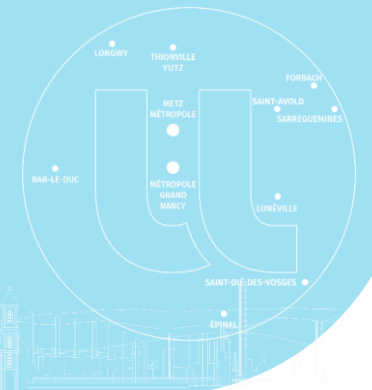






Version control system(***version control system***)

- A set of software tools for:
 - Memorize and find different versions of a project.
 - Facilitate collaborative work.
- Originally developed by Linus Torvalds to facilitate Linux kernel development.
 - Free / open source software.
 - Available on all platforms.

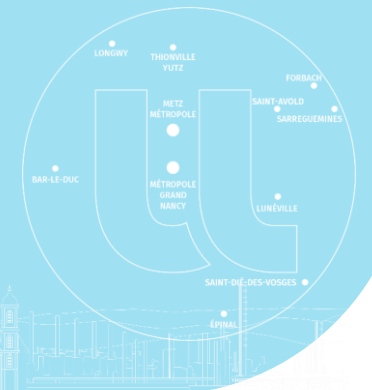


Git-based web app.

- Allows you to manage:
 - The git project lifecycle.
 - Project participants (roles, groups, etc.).
 - Communication between these participants.

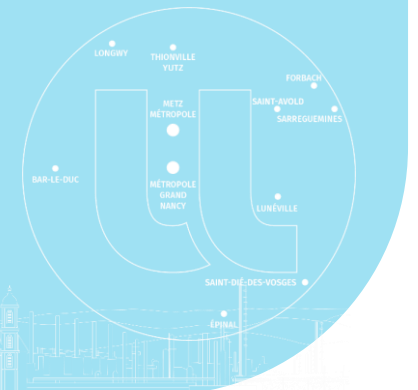
Developed by a private company

- The "community edition" is free.
- A free online service (like github)



Postulate: need a system to manage the Software Project Lifecycle.

- Manage versions.
- Manage developers (access rights).
- Manage issues (bug reports, suggestions)
- evolution, etc.) .
- Create online documentation for a project.
- Automatic build, test, and deploy.



Software Development

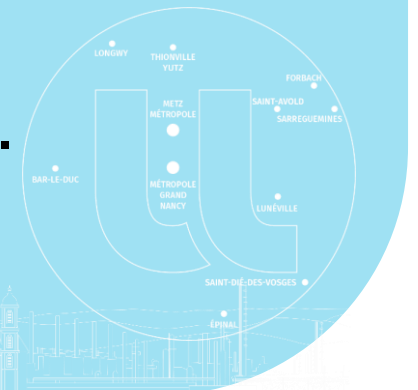
- Designed to manage software development...

Writing documents

- Articles (e.g. latex sources).
- Web pages (e.g. HTML, markdown, etc.).
- Any type of text document (ascii format).

Archive binaries (use the **git-lfs**)

- executable and "released" libraries
- Files generated by specific editors (e.g. CAD, Word).





How an SSH Key Pair Works



Private key

Securely stored
on the user's
machine

Public key

Stored on
the remote
server

Secure Shell (SSH) is both a computer program and a secure communication protocol. The connection protocol requires an exchange of encryption keys at the beginning of the connection. Thereafter, all TCP segments are authenticated and encrypted.

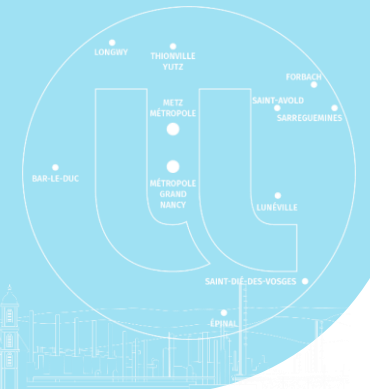
You need to generate an SSH key pair via the: **ssh-keygen**

At this point, two keys will be generated in your default 'home' folder:

- Linux: **/home/[user]/.ssh**
- **Warning: The ssh folder is a hidden folder**

The following files can then be found in the **.ssh** folder:

- **id_rsa: private key to keep on your PC and not to share**
- **id_rsa.pub: public key to send to the machines you communicate with**





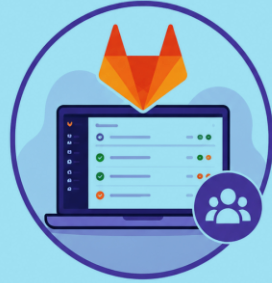
1. Create an SSH key pair with **ssh-keygen -t ed25519**
2. Make a key backup
3. Use **git clone** in SSH to retrieve the project
4. Declare the **public key** on your **gitlab** account

```
yann@lcom-is-m-maif2:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/yann/.ssh/id_ed25519):
Created directory '/home/yann/.ssh'.
Enter passphrase for "/home/yann/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yann/.ssh/id_ed25519
Your public key has been saved in /home/yann/.ssh/id_ed25519.pub
The key fingerprint is:

The key's randomart image is:
+--[ED25519 256]--+
|
|
|
|
|
|
|
|
|
|
+-----[SHA256]-----+
yann@lcom-is-m-maif2:~$ ls .ssh/
id_ed25519  id_ed25519.pub
yann@lcom-is-m-maif2:~$ cat .ssh/id_ed25519.pub
ssh-ed25519
```

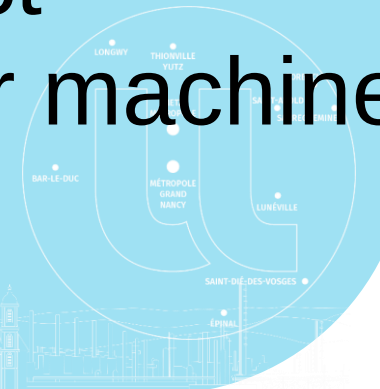


Exercise
Git



GitLab
Collaborate • Review • Deploy

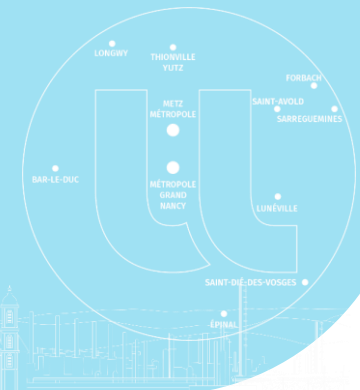
1. Create account on <https://www.gitlab.com>, log in
2. Create a first **blank project**
3. Create a helloworld.txt file locally (hard drive)
4. Upload helloworld.txt in the created project
5. Verify that the git client is installed on your machine
6. Create an EX01 directory in your home
7. Use git clone in ssh to recover the project



```
yann@lcom-is-m-maif2: ~/Gi  X + v
yann@lcom-is-m-maif2:~$ nano HelloWorld.txt
yann@lcom-is-m-maif2:~$ pwd
/home/yann
yann@lcom-is-m-maif2:~$ mkdir GitlabPractical
yann@lcom-is-m-maif2:~$ cd GitlabPractical/
yann@lcom-is-m-maif2:~/GitlabPractical$ git clone git@gitlab.com:ian57/myfirstgitlabproject.git
Cloning into 'myfirstgitlabproject'...
The authenticity of host 'gitlab.com (172.65.251.78)' can't be established.
ED25519 key fingerprint is: SHA256:eUXGGm1YGsMAS7vkcx6JOJdOGHPem5gQp4taiCfCLB8
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'gitlab.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
yann@lcom-is-m-maif2:~/GitlabPractical$ ls
myfirstgitlabproject
yann@lcom-is-m-maif2:~/GitlabPractical$ cd myfirstgitlabproject/
yann@lcom-is-m-maif2:~/GitlabPractical/myfirstgitlabproject$ ls
HelloWorld.txt  README.md
yann@lcom-is-m-maif2:~/GitlabPractical/myfirstgitlabproject$
```


The ancestor of the VCS: the CPOLD methodology

- **repository** = root folder of the project.
- Each folder/file is associated with a name + an extension.
 - Name = name of the item contained in the project
 - Extension = the version of the file/folder.
- Based on the **cp** command:
 - `cp file1.old //"old" version saved`
 - `cp -R folder folder.1.4 //assumes version 1.3 exists!`
- Sharing changes:
 - via a classic file server (or any medium)
 - Merging changes is done manually.



Local Computer

Checkout

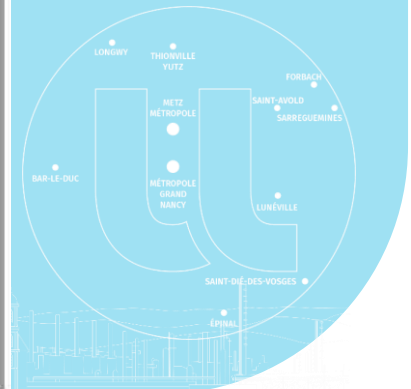
File

Version Database

Version 3

Version 2

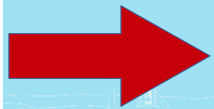
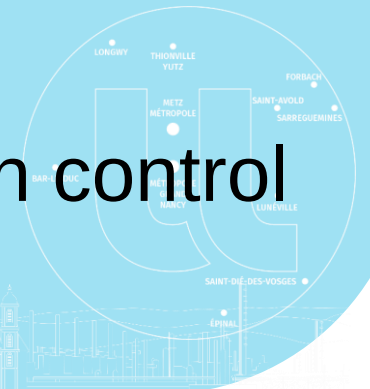
Version 1



CPOLD : limitations

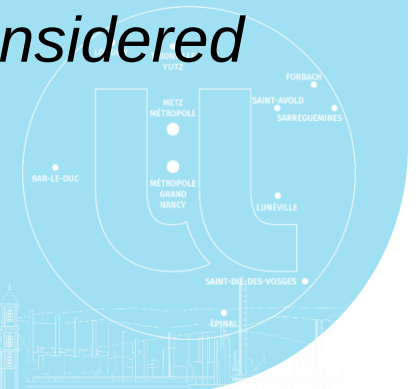
- Naming convention for extensions must be strictly followed.
- Suboptimal in terms of disk space
 - each version = full copy existing code + additions.
- Sharing changes and merging variants is **very laborious**.

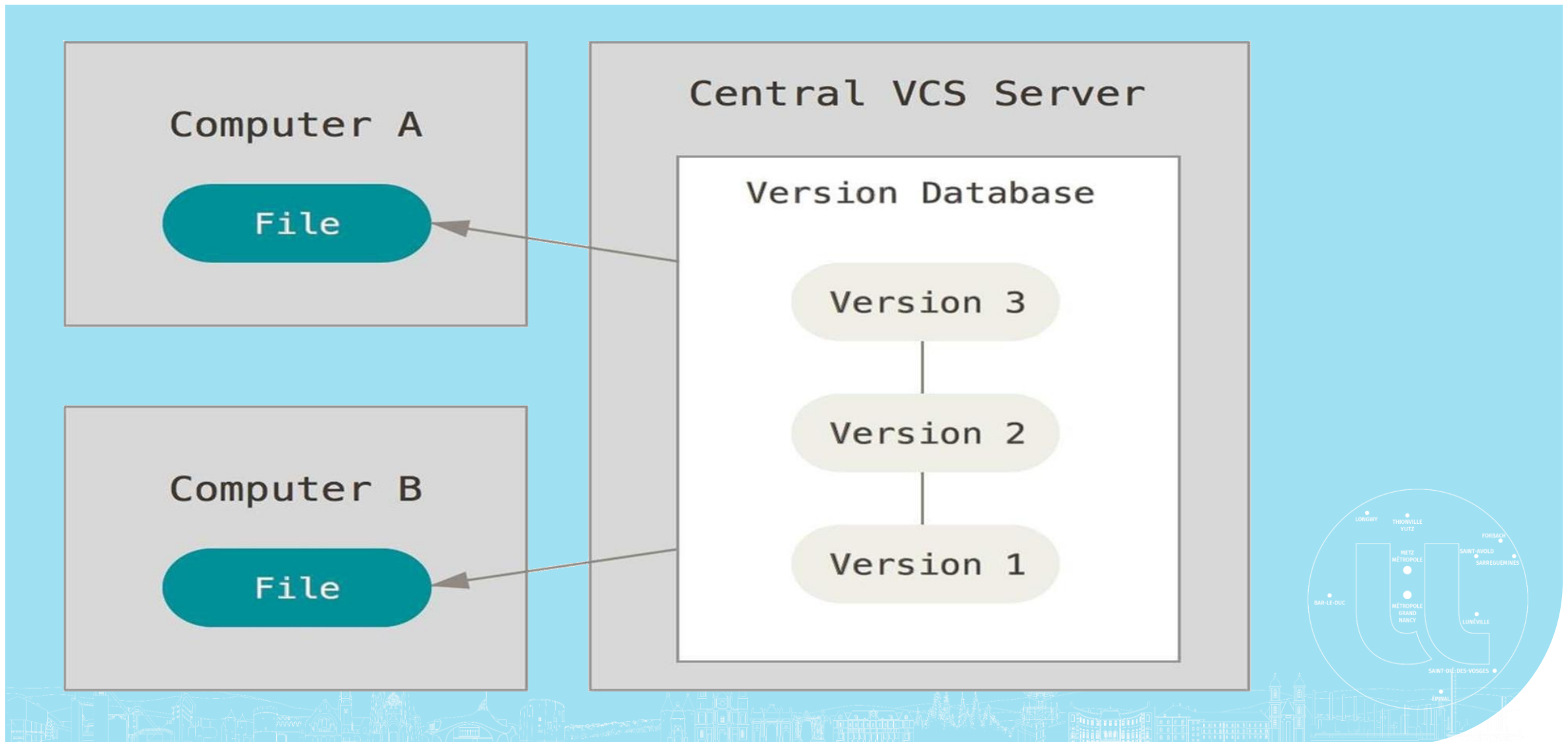
Need tools to automate, simplify, and make version control
robust.



Centralized VCS: CVS, subversion, etc.

- Software tools based on a client/server approach.
 - Server = a file server that **contains repositories** and manages user access.
 - Client = a workstation that contains a **working folder** connected to the repository.
- Principle: only **changes** are saved.
 - **Commit**: *A set of changes to files/folders in the project, considered atomic (indivisible).*
 - **repository** = *commit history.*





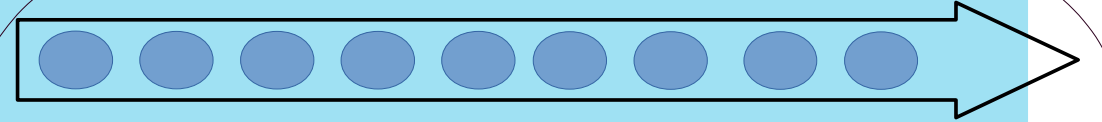
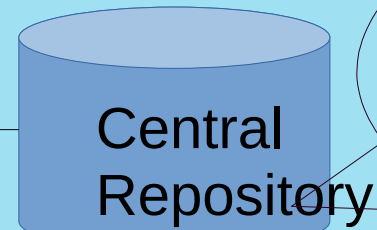
• Centralized VCS

1) Get Repository Changes:
update

 **Fusion Conflicts**

3) Publish changes
in the central repository:
commit

2) Local Changes: Operations
on project files and folders



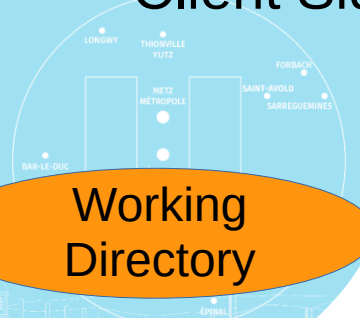
Project history
= set of commits

Server Side
Client Side

Répertoire
de travail

Working
Directory

Working
Directory



Centralized VCS: advantages

- Optimal size on disk
 - Version = changes made (commit) since previous version.
- No need for conventions.
- Version sharing is automated.

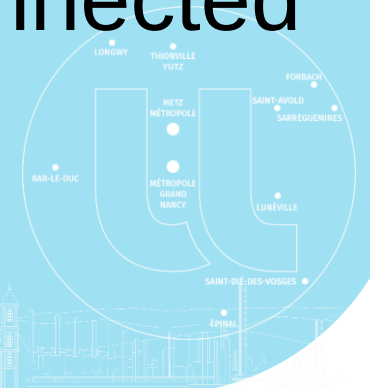
Centralized VCS: Limitations

- Difficult offline work: To create commits/merge changes, you need to synchronize with the server.
- Lack of robustness: Losing the central repository (e.g. data corruption) = losing all.

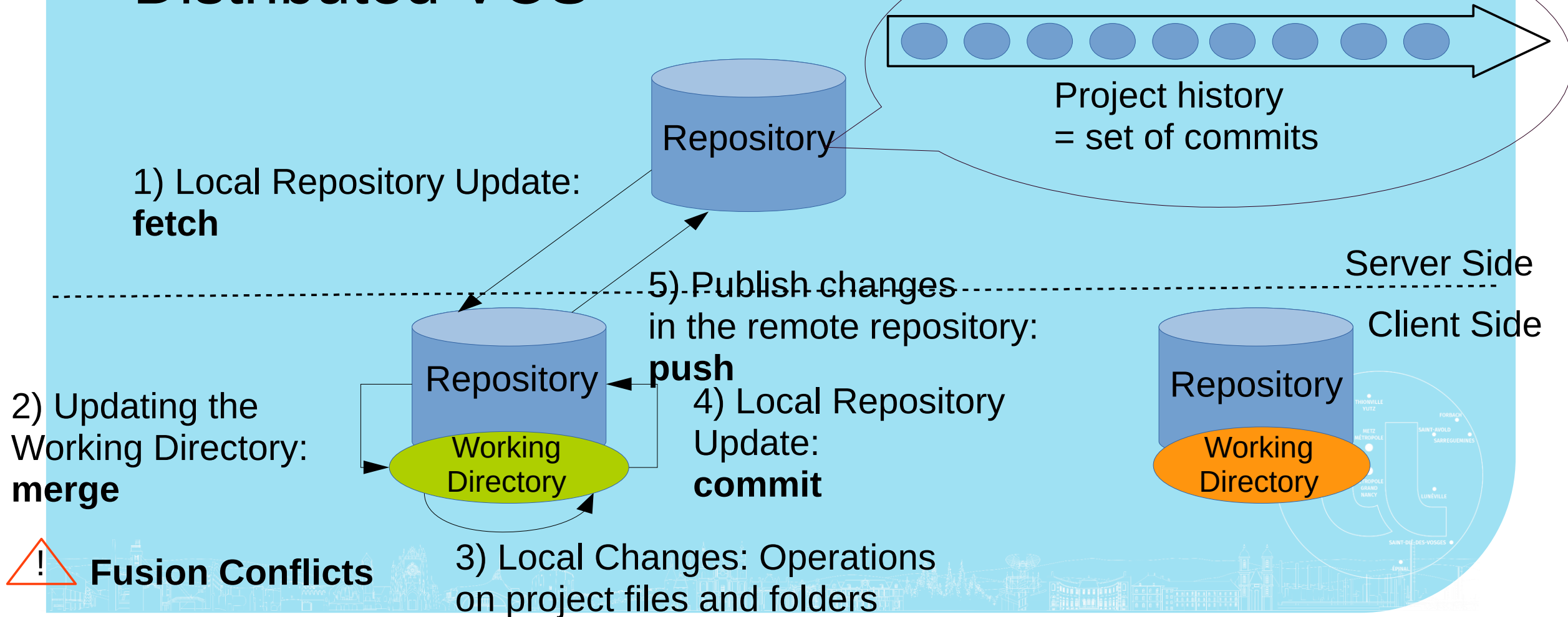


Distributed VCS: git, Bazaar, mercurial, etc.

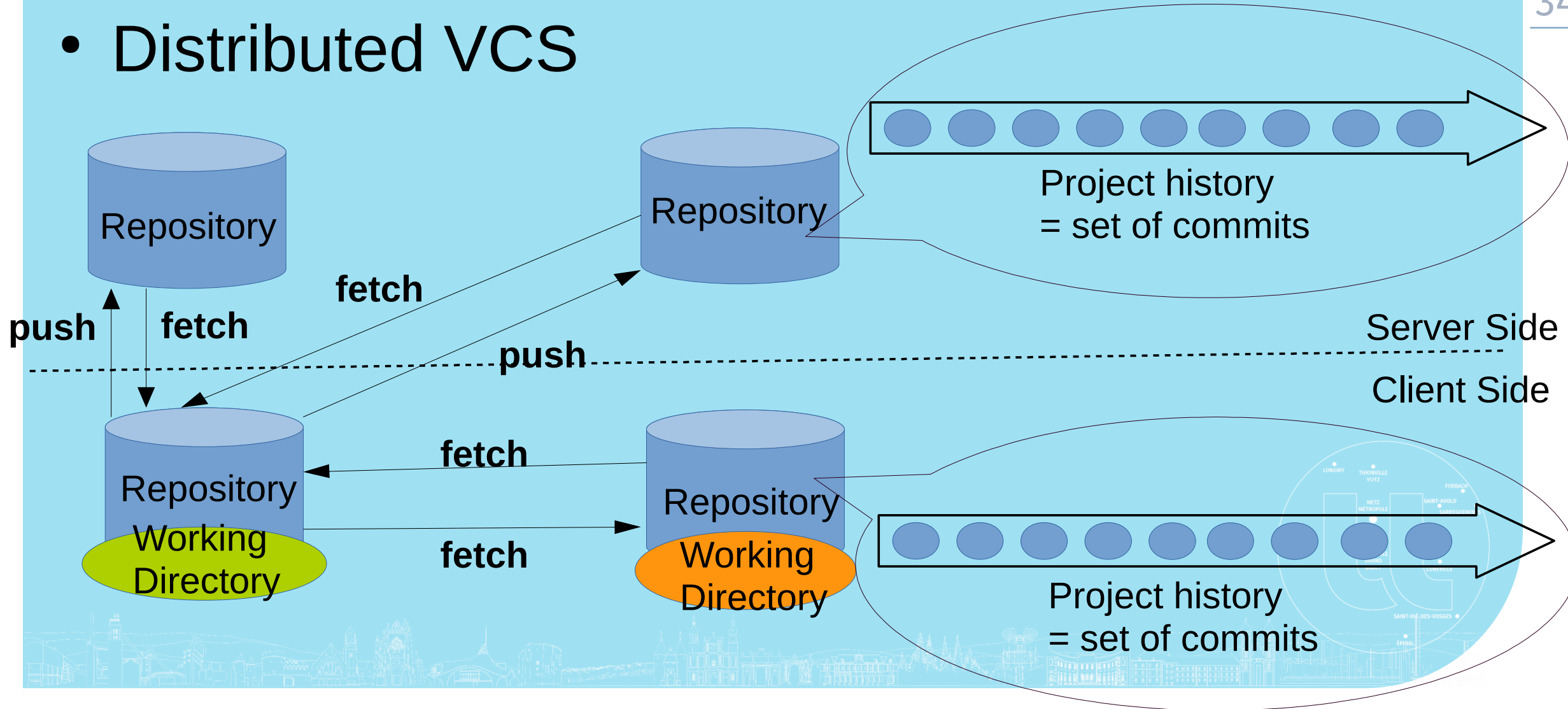
- Based on a peer-to-peer approach to synchronizing operations.
- Repositories exist on both the **server and client** sides.
- On workstations: A working directory is connected to each repository.



• Distributed VCS



- Distributed VCS



Distributed VCS: Benefits

- Robustness: multiple copies of the same deposit.
- Easy offline work: create commits and
- Merging changes is done offline.

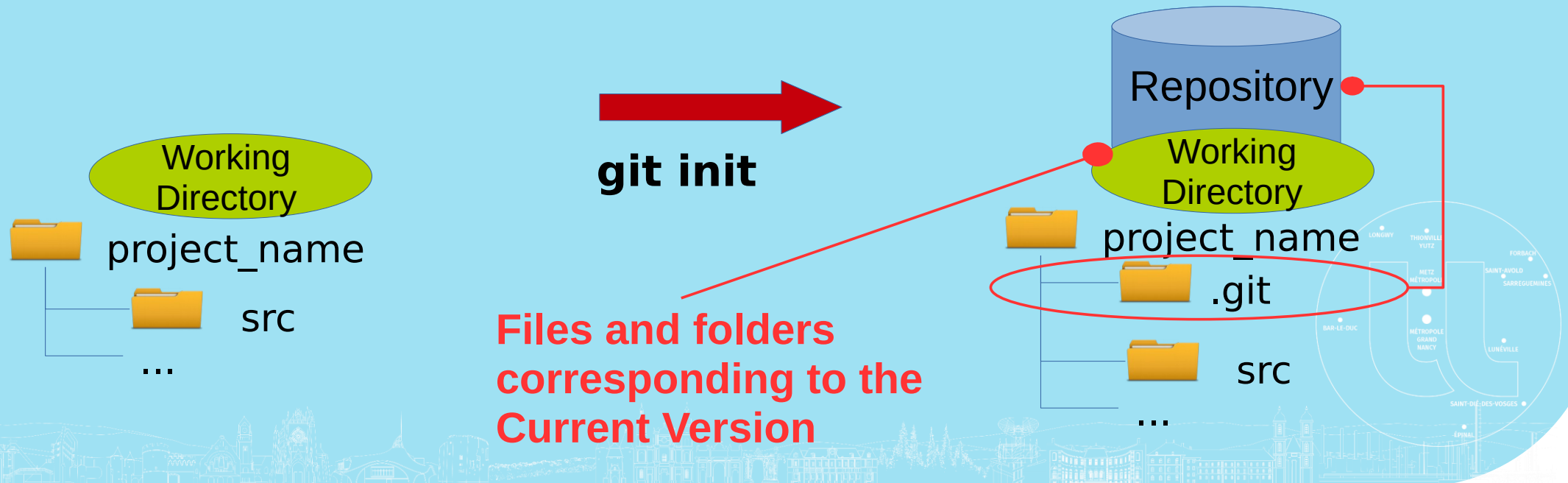
Distributed VCS: Limitations

- More commands to understand...
- **Guaranteeing a policy of access to repositories is very difficult with DVCS**
 - Each repository has a full copy of the history (since last **fetch**).
 - Each repository defines its own access rights for its customers.

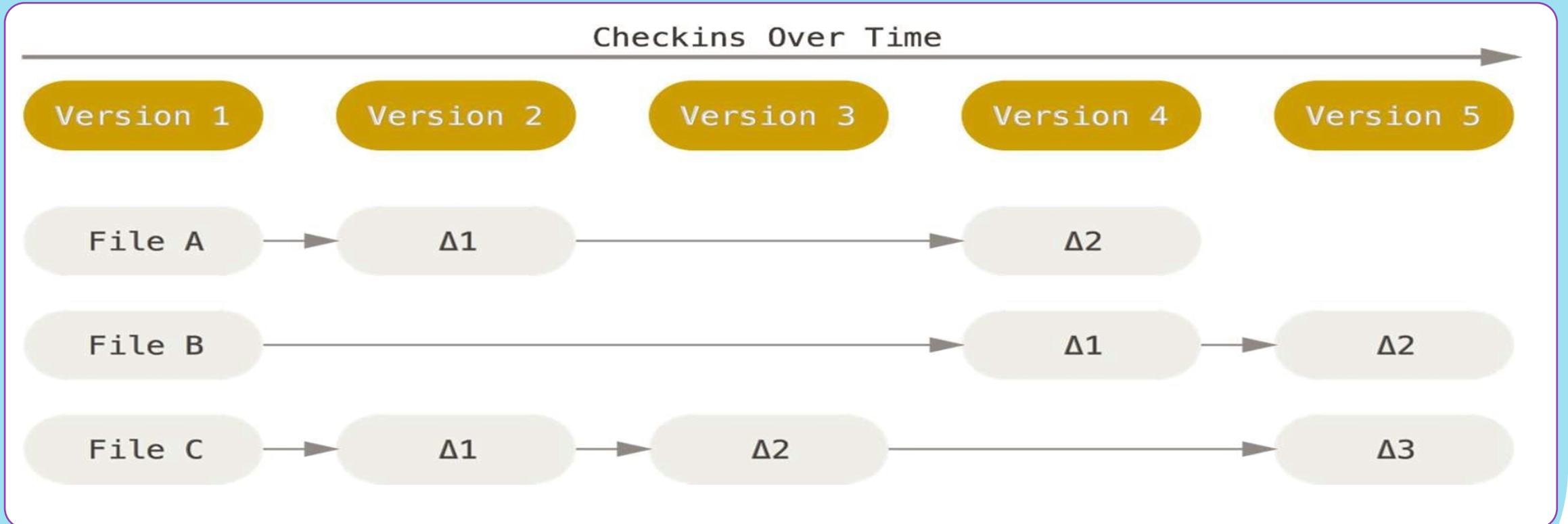


The repository

- A directory whose structure describes the commit graph, branches, and so on.
- Turning a file into a repository: **git init**

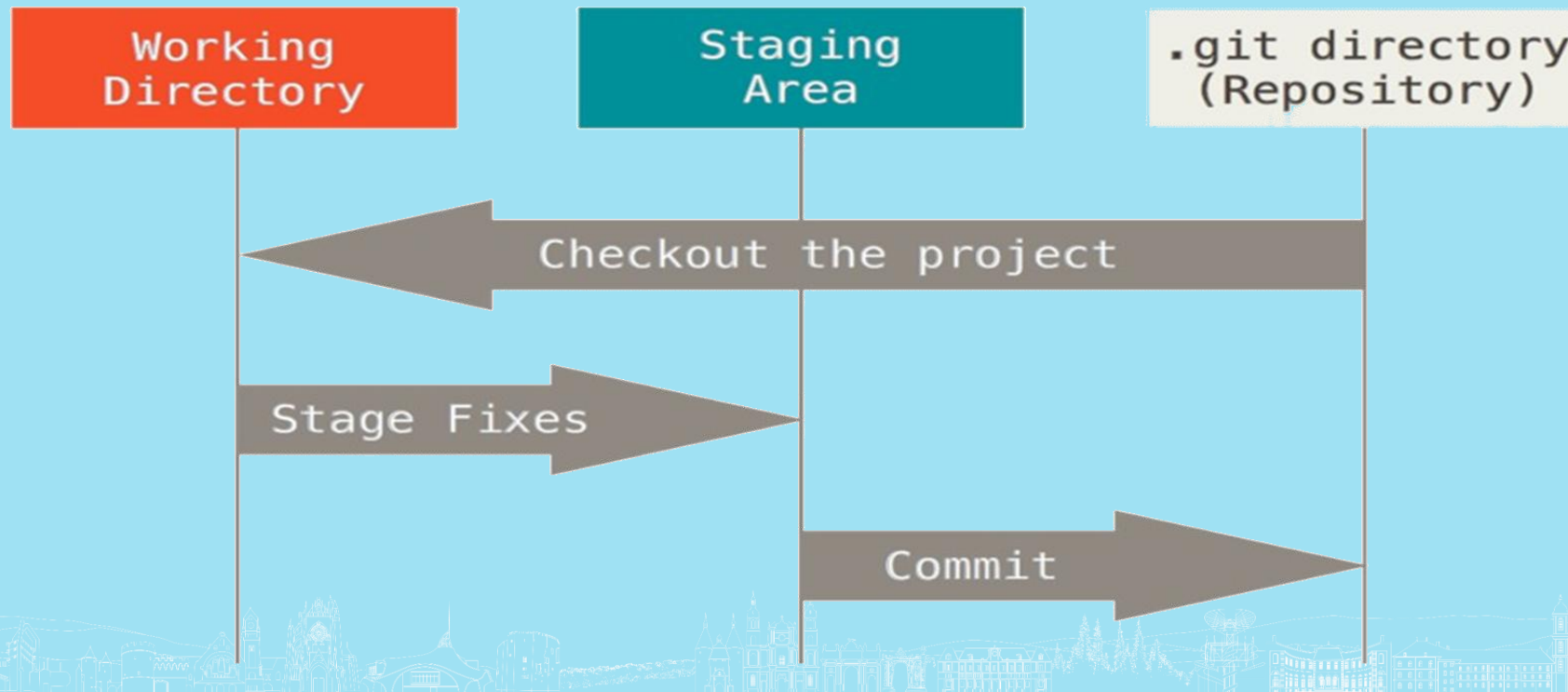


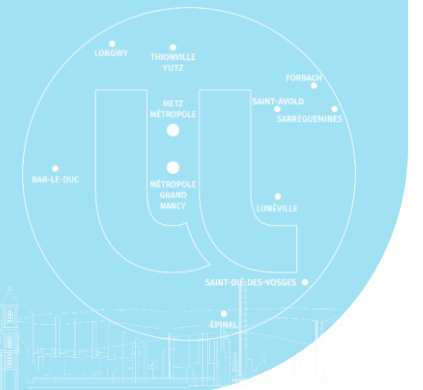
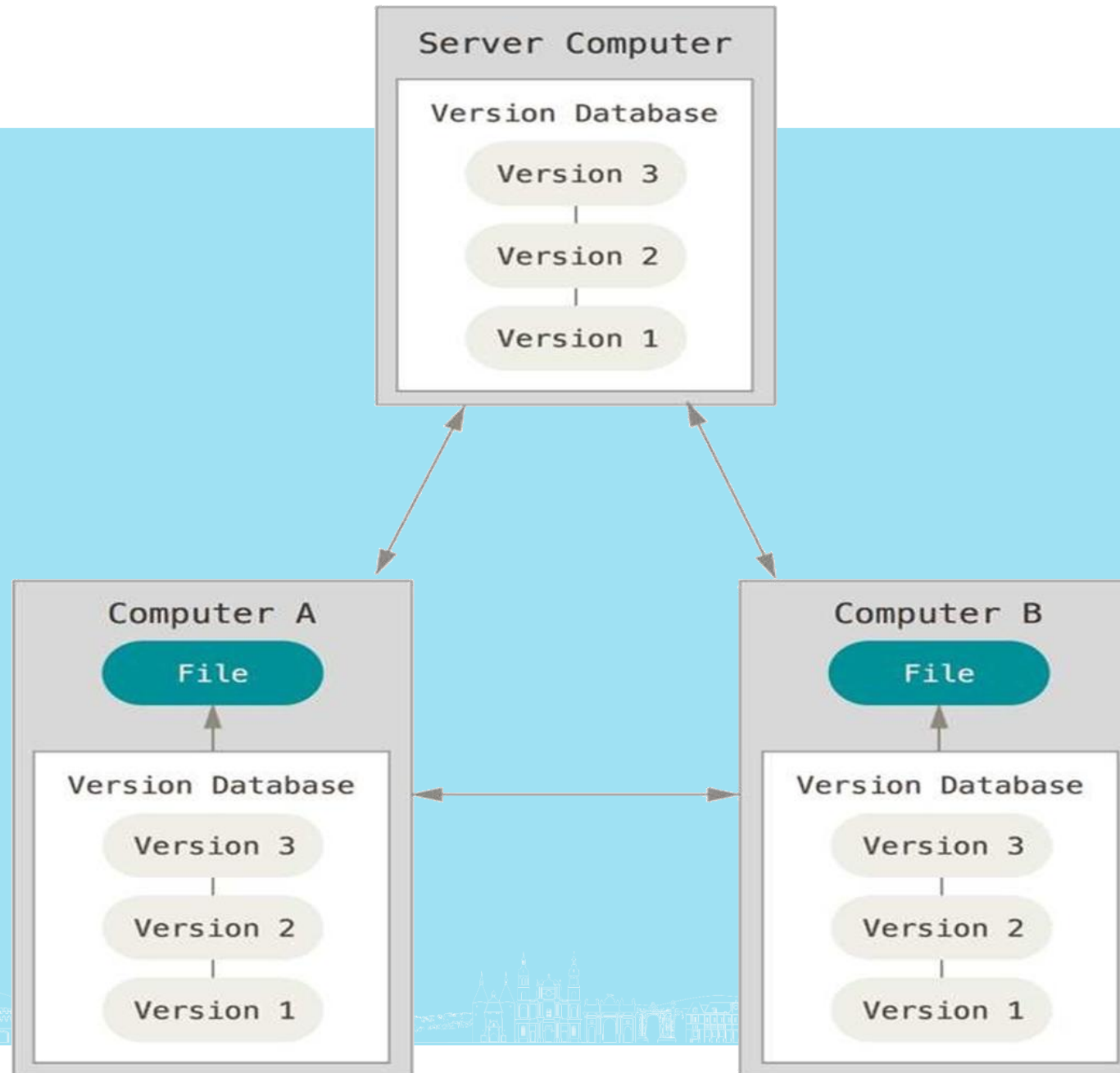
Each time you commit or save the project state, Git takes a snapshot of the contents of your workspace.

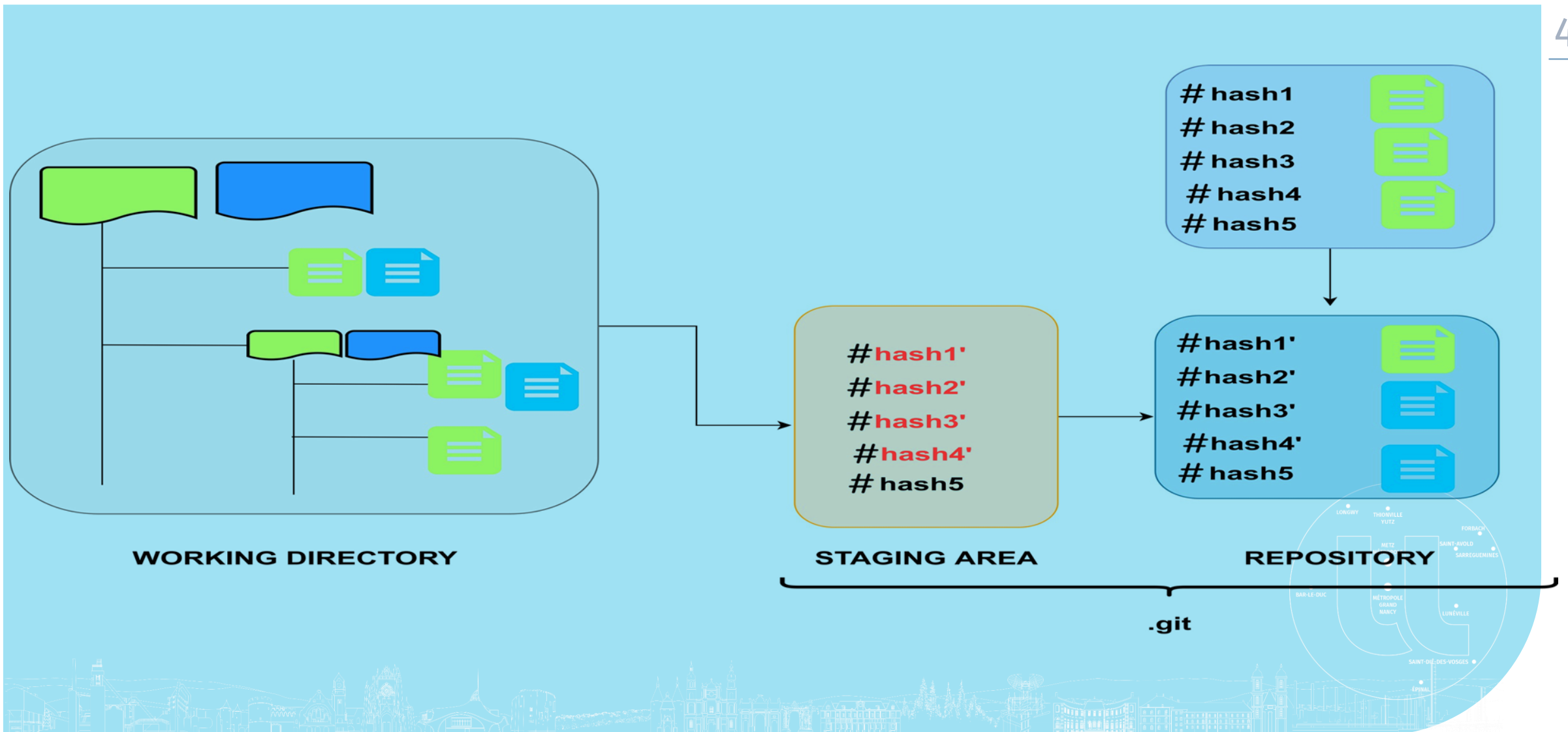


3 main sections of a GIT

- **Validated** = the .git directory
- **Modified** = the current working directory
- **Indexed** = The virtual area that lists the files that will be in the next







The **commit**

- A set of changes to files and directories.
 - Represented by a **patch** operation.
- Uniquely identified (relative to a project) by a SHA-1 hash code

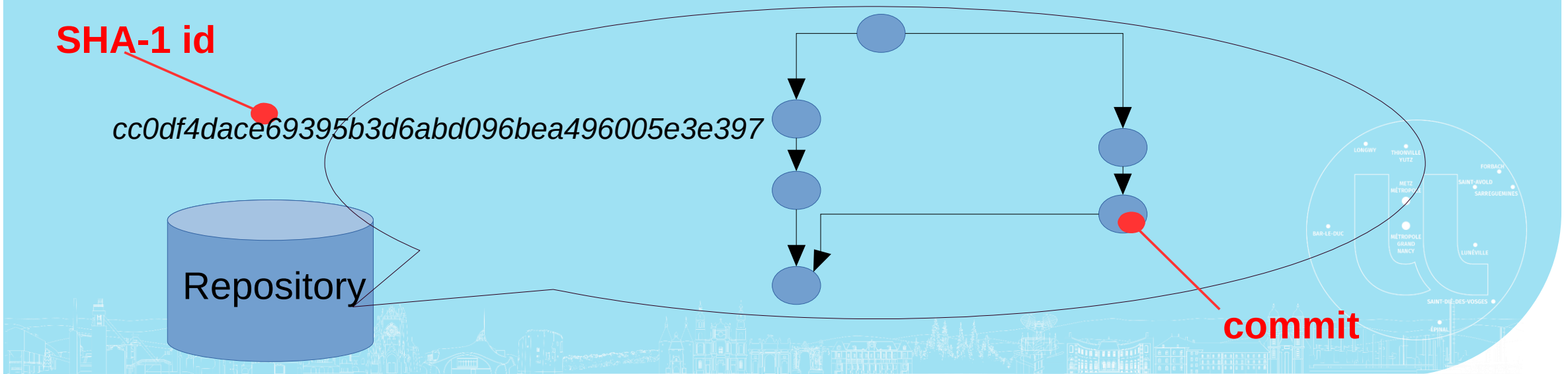
Commits are structured according to a **directed acyclic graph**

SHA-1 id

cc0df4dace69395b3d6abd096bea496005e3e397

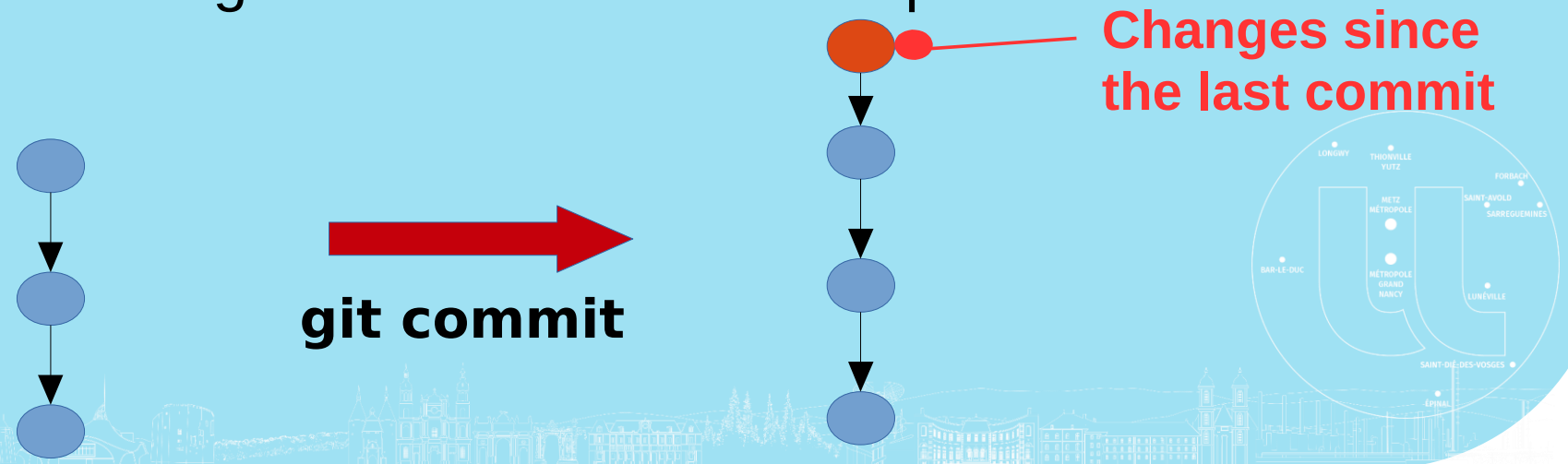
Repository

commit



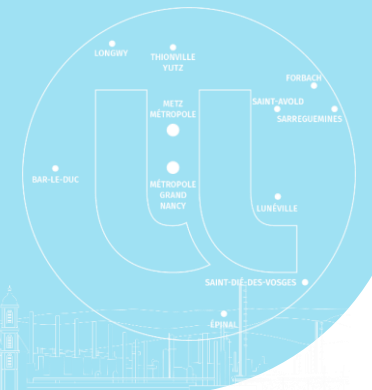
The commit

- allows you to commit your changes to create what is called a commit, i.e. a committed step in the code.
- The graph is updated each time a commit is created (e.g. commit action)
 - The new commit sets a pointer to the previous commit and contains the changes to the content of the previous commit.



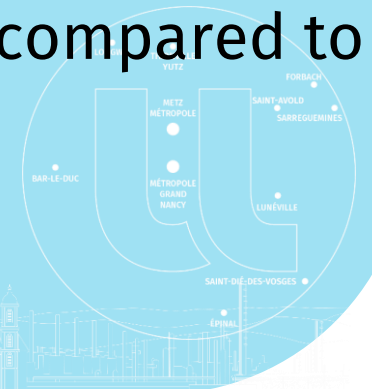
A commit is composed:

- of a commit message
- a snapshot of the code that is referred to via a unique identifier (a fingerprint or hash)
- of an author
- to a parent commit (or commits) (which allows us to make branches of the Git tree)



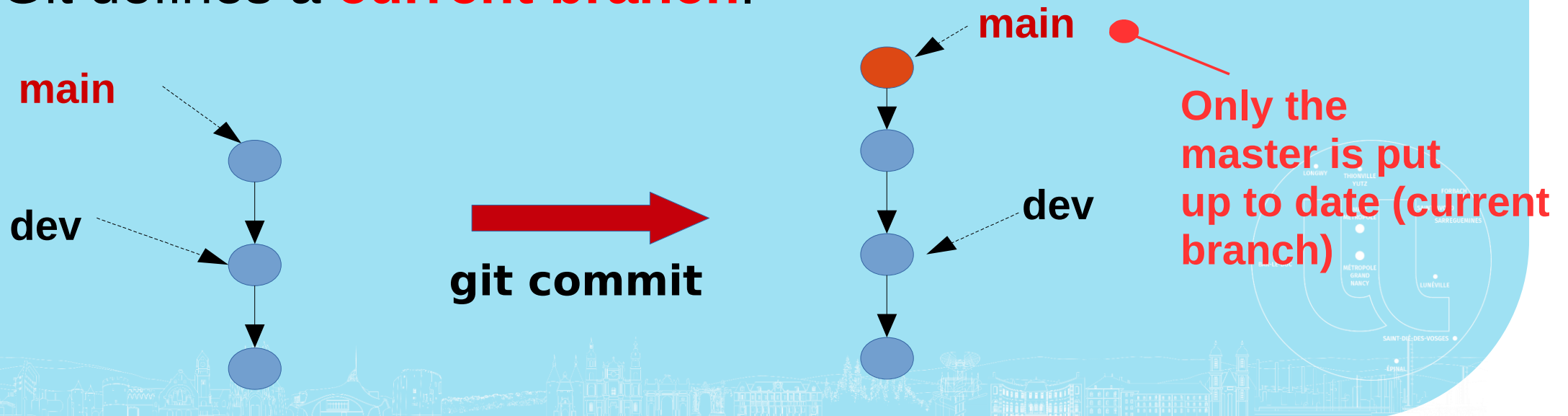
A commit is composed:

- Ideally, when you make a commit, the code should be in a roughly consistent state and the commit should gather changes that make sense to achieve a goal (e.g., fixing a bug, adding a feature, modifying documentation...)
- Always put a commit message
- Commits are stages in software development. Reading the list of these steps should help a developer understand how the code is evolving.
- a commit is always a reference to a specific version of the entire code in relation to the Git tree, it's not just additions and deletions compared to the code of the previous commit



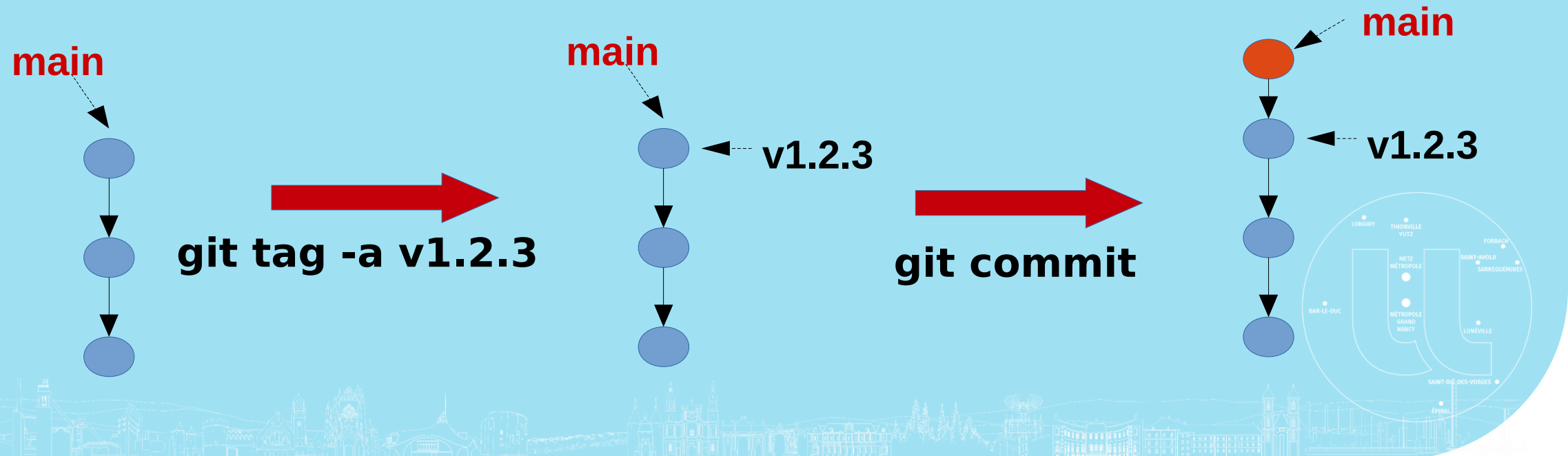
The branch

- A **pointer** to (i.e. a reference to) has a commit.
- Mis à jour chaque fois qu'un commit est créé, pointe alors vers ce nouveau commit.
- Default branch: **main (master)**.
- Git defines a **current branch**.



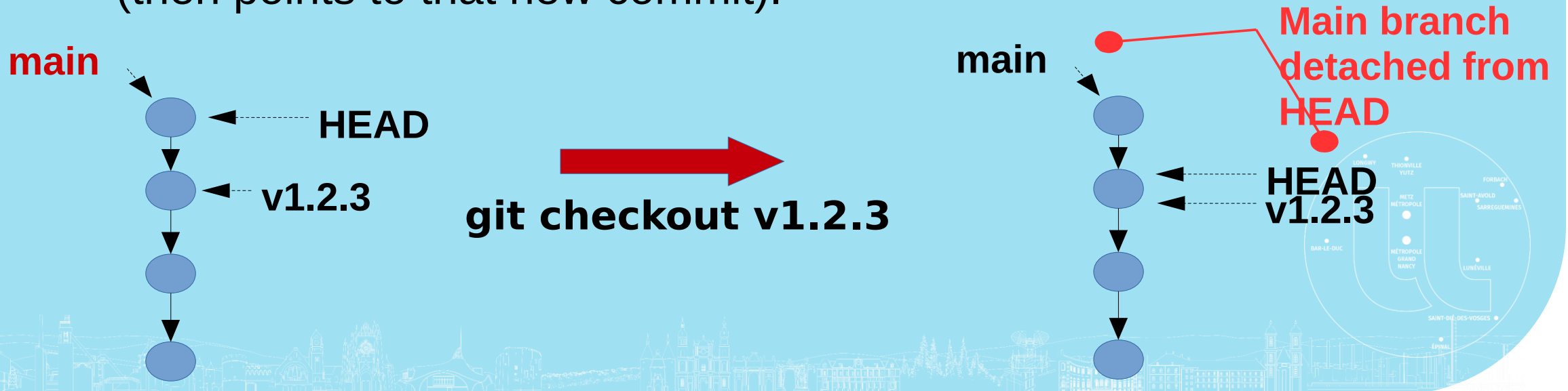
The tag

- A **pointer** to (i.e. a reference to) has a commit.
- Allows you to memorize interesting versions.



The HEAD pointer

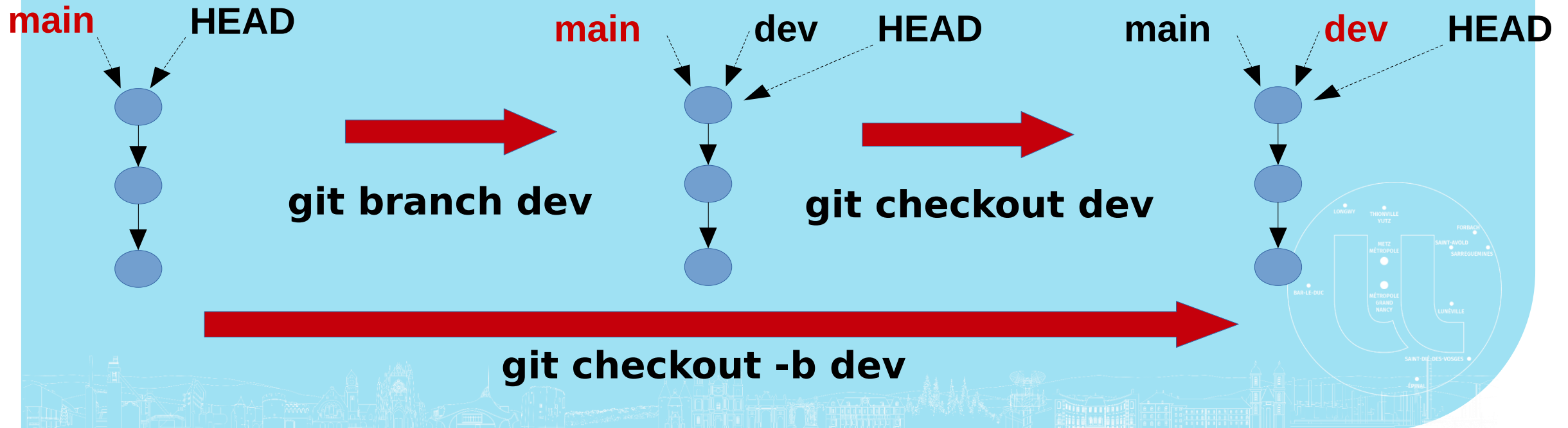
- A **pointer** to (i.e. a reference to) a commit.
- Represents the **current state of the working directory**: changes applied to the project up to the targeted commit.
- Allows you to navigate through the graph to find a given version.
- Updated on command or automatically each time a **commit** is created (then points to that new commit).



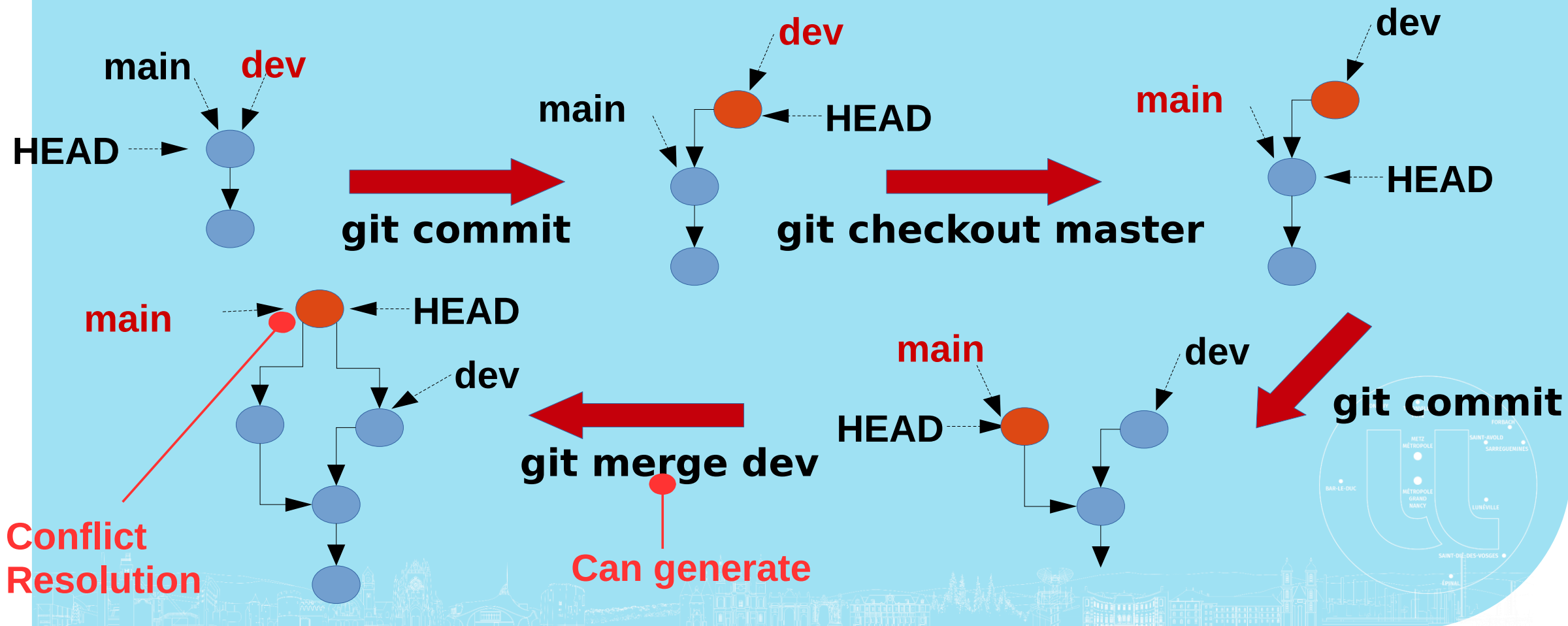
Créer des **branches**

- Plusieurs branches dans le même dépôt.

HEAD.

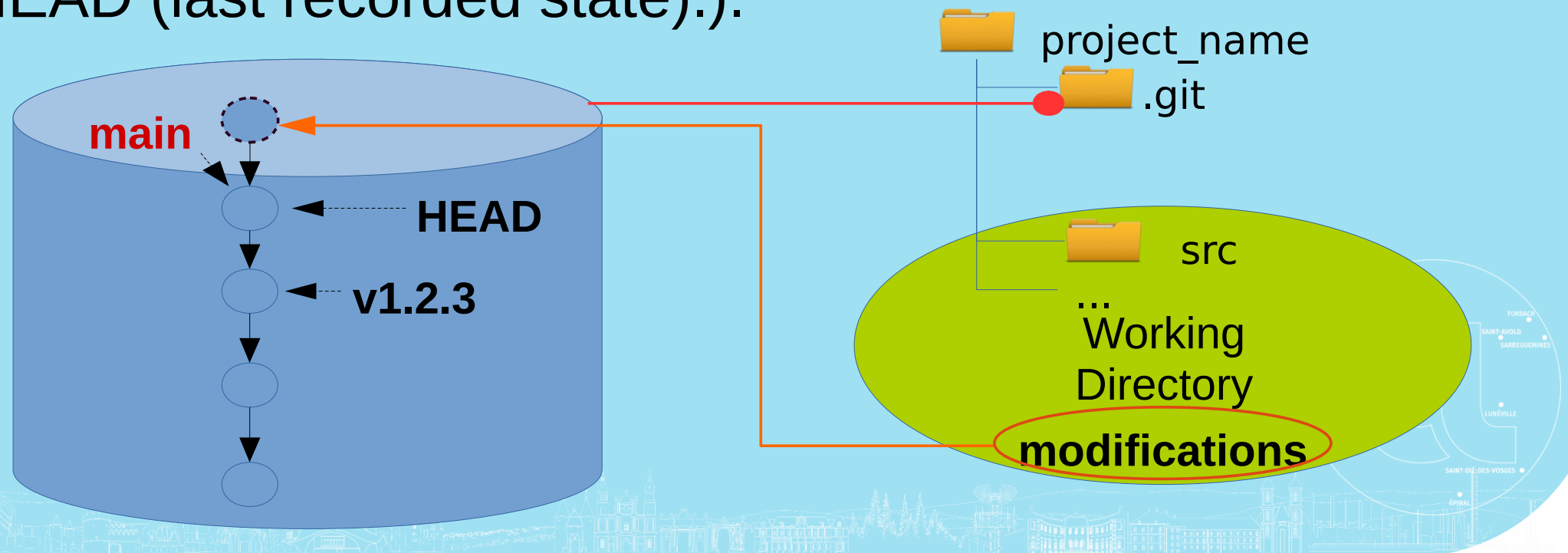


• Fusion (*Merge*) of branches



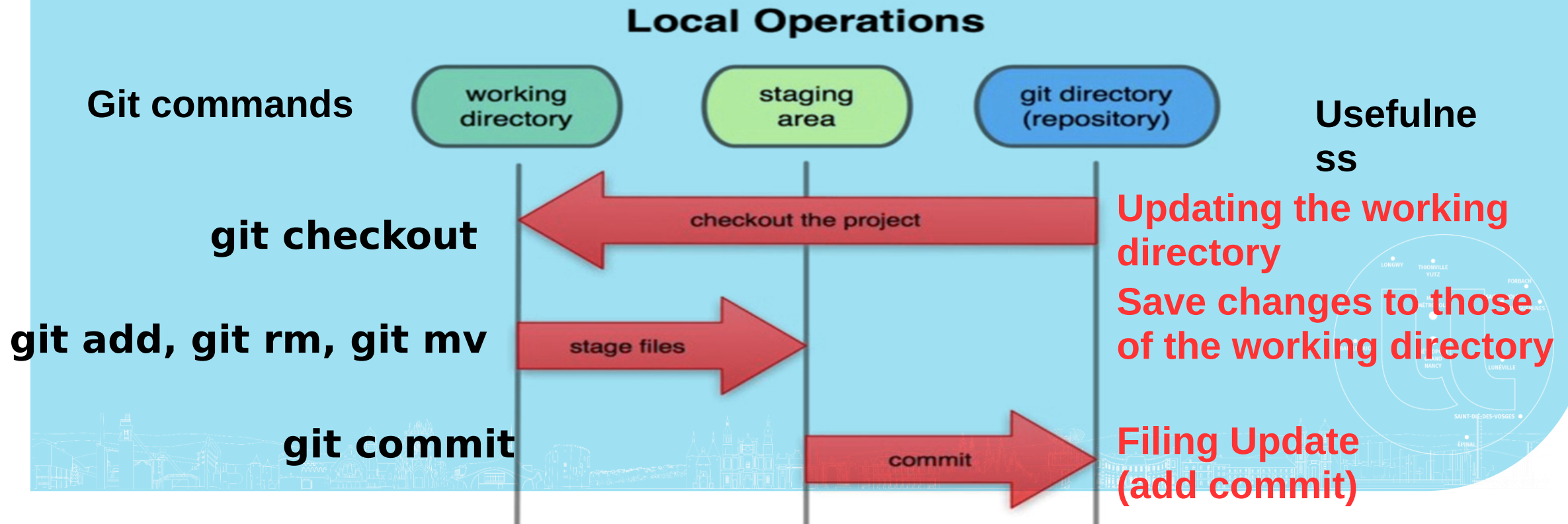
Create a commit from changes

- Goal: save changes between the contents of the working directory (folders and files) and the commit pointed out by HEAD (last recorded state).



Create a commit from changes

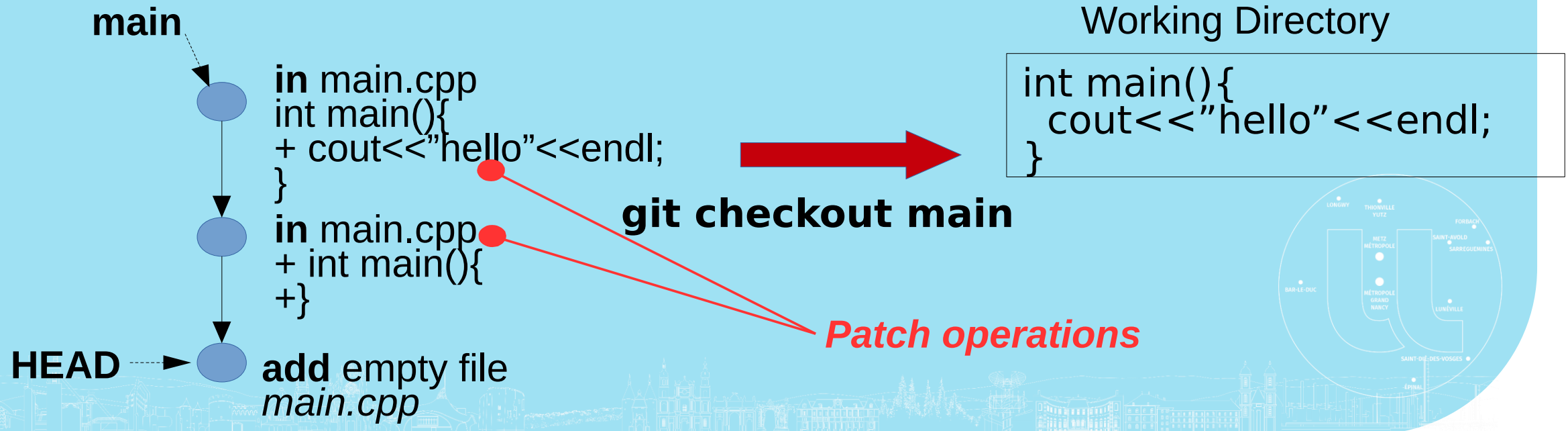
- Originality of git: 2 phases



Updating the working directory

- At each checkout or merge.
- Applies the sequence of changes (patches) and then the first one until the current commit (pointed out by **HEAD**).

main.cpp in the
Working Directory



Saving changes to the working directory

- Sélectionner les modifications qui seront contenu dans le prochain commit.
 - internal files (use **git add -p**); All changes to a file or new file (use **git add filename**)
 - All changes (use **git add -all**)
- Deselecting changes is possible (use **git reset**)

main.cpp in **HEAD** in the **repository**

```
int main(){
    cout<<"hello"<<endl;
}
```

main.cpp in the **working directory**

```
int main(){
    cout<<"hello"<<endl;
    return 0;
}
```

git add main.cpp

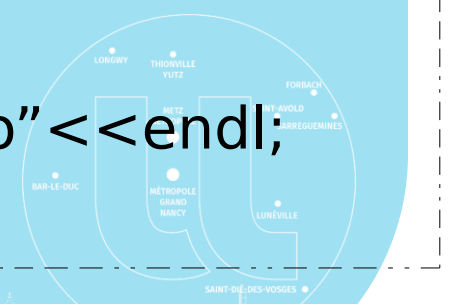


Saved Change in the
staging area

in

```
main.cpp
cout<<"hello"<<endl;
+ return 0;
}
```

Diff operation to deduce the patch.



Updating the local repository (“commit”)

- Create a new commit from the changes saved in the « *staging area* ».
- Added a message explaining what has been changed.

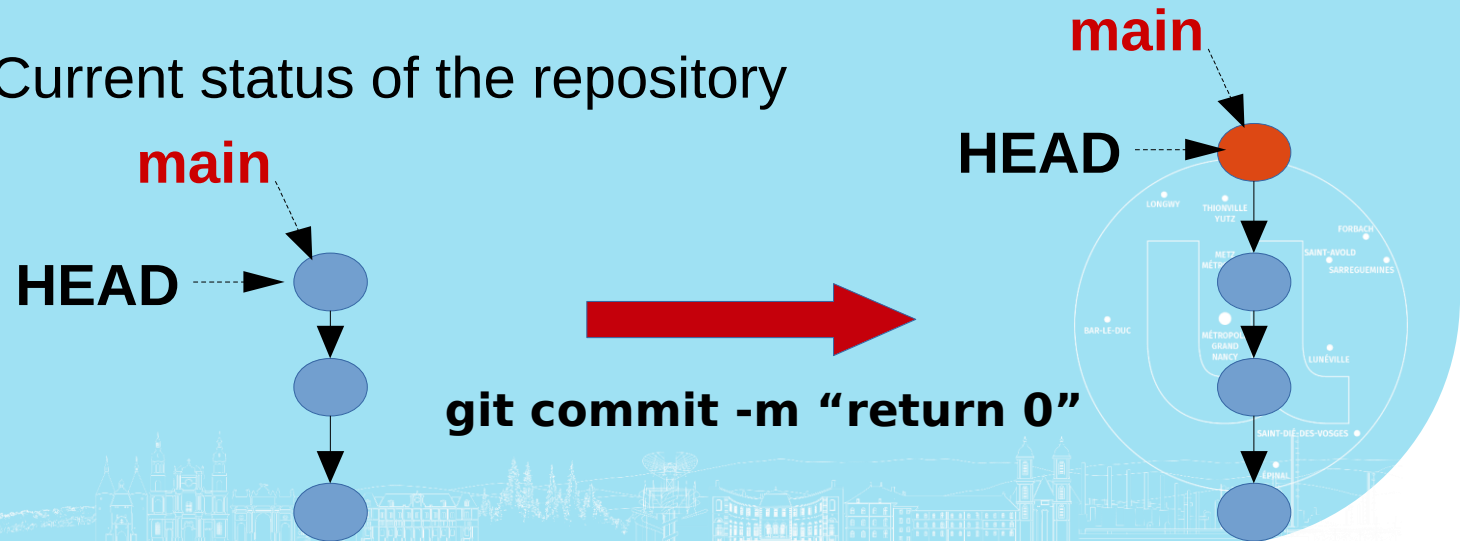
Saved changes in the
staging area

```
add other.cpp
```

```
in main.cpp
```

```
in main.cpp  
cout<<"hello"<<endl;  
+ return 0;  
}+ return 0;  
}
```

Current status of the repository



Why 2 phases to create commits?

- Create "small commits" from a large number of changes.
- Delay the commit of certain changes: Committed features can be "committed" immediately.
- isolate code that will never be committed (e.g. debug traces).

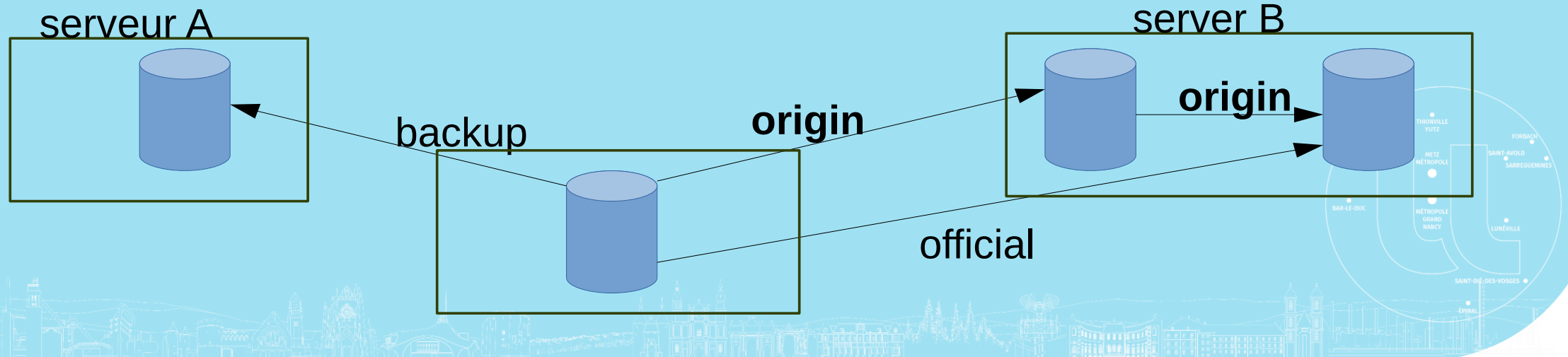
Once the interesting changes are "committed", cleaning up the staging area and working directory is possible:

- Use **git reset -hard** : permanent deletion of uncommitted changes.
- Use **git stash save** : uncommitted changes can be rolled back.



Synchronization between remote repositories

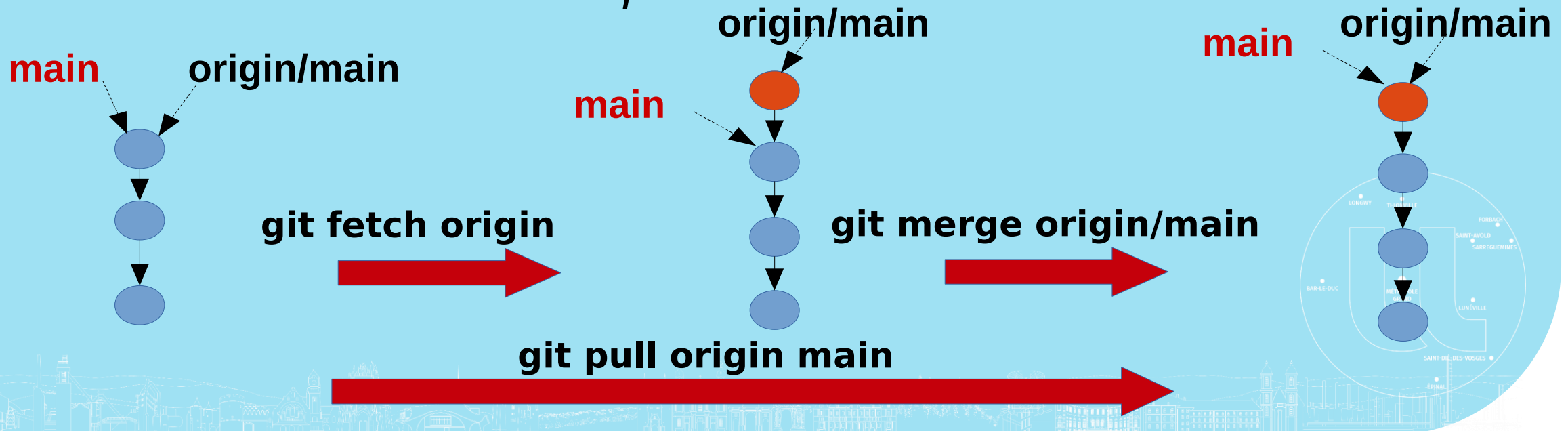
- Each repository has a set of repositories called **remotes**.
- The default *remote* is called **origin**.
- Each remote is associated with a **name** (unique in the local repository) and defines a **url** (address of a remote repository).).
- Add a *remote* : **git remote add backup <address>**
- Delete a *remote* : **git remote rm backup <address>**



-

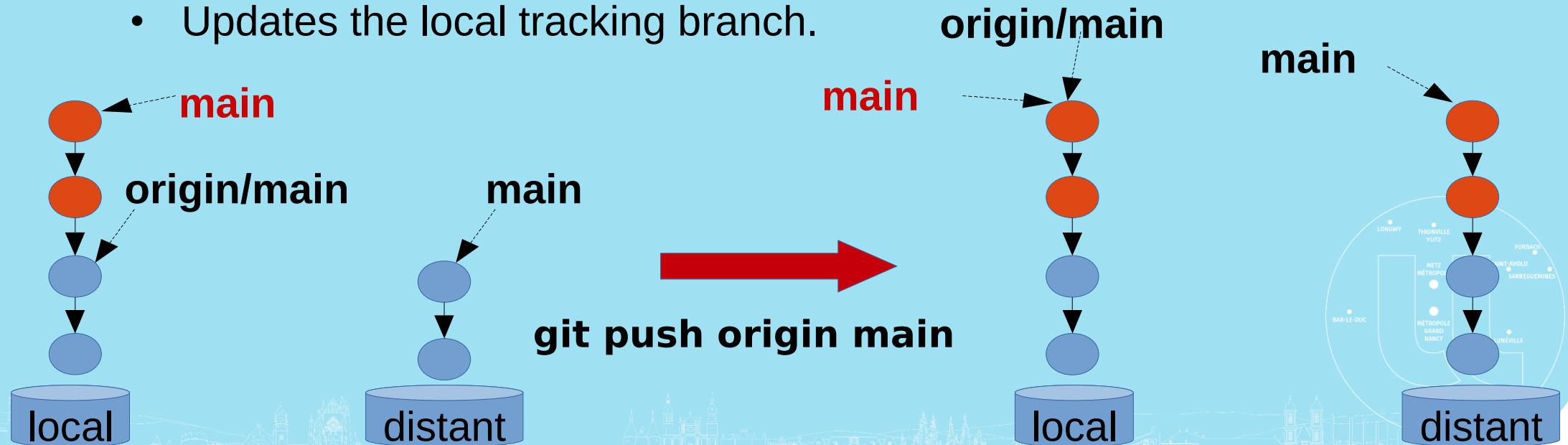
Updating a repository from one of its *remotes*.

- Update (if necessary) the **local commit graph** and **tracking branches** (*fetch* command).
- Local branches are **merged** with tracking branches (merge command).
- All-in-one command: *pull*



Update a remote branch from a local branch

- Use the command ***push*** (opération atomique)
 - Verifies that the corresponding local tracking branch is up to date.
 - Updates the remote repository's commit graph and branch
 - (of the same name).
 - Updates the local tracking branch.

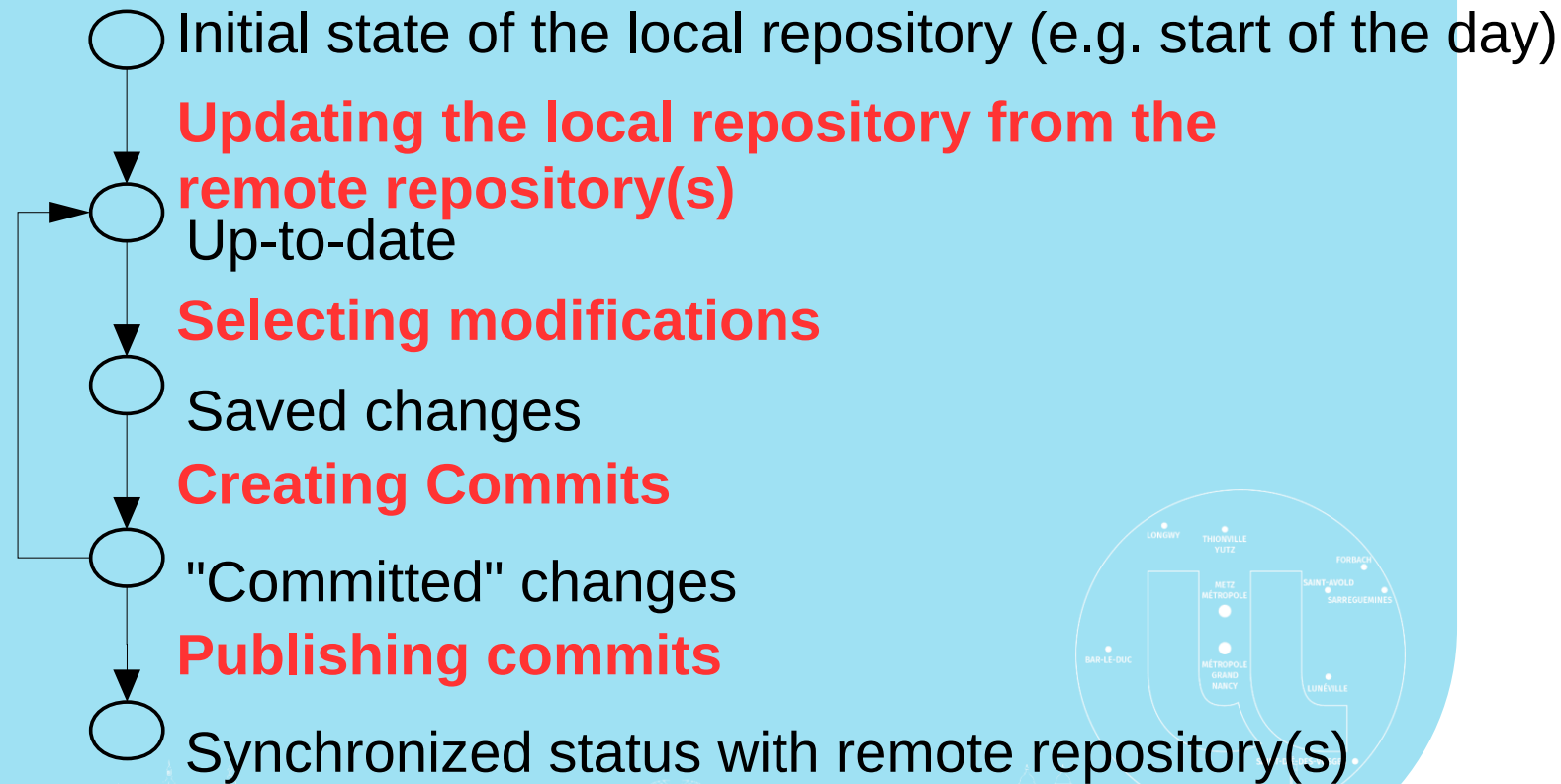


Clone, the most well-known git operation corresponds to:

- Create a local directory and initialize it as a repository (**git init**).
- Create a *remote* called **origin** (**git remote add origin <address>**)
- Create a local branch *main* (**git checkout -b main**)
- Local branch Update *main* (**git pull origin main**)

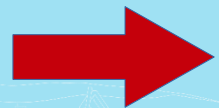


■ ■ ■

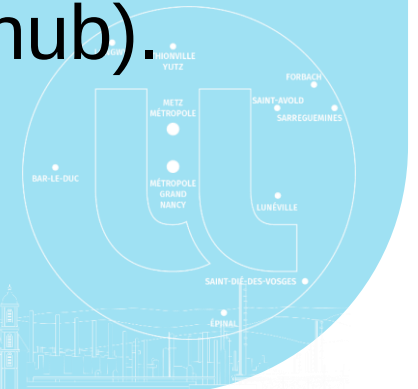


Web application to manage **server-side** repositories

- Creation/deletion of repositories.
- Registration and management of **access rights** to repositories.
- Graphical tools for viewing and editing: graph of commits, branches, tags, files, etc.
- Project documentation (README files)
- Project communication tools (**issues**)
- **Fork** mechanism and **merge requests** (like on github).



Gitlab server: <https://www.gitlab.com>



Definitions

- *Project*: git repository + other information (access rights, wiki, issues, continuous integration, etc.).
- *Workspace*: A **directory** containing git repositories.
- *User*: The person registered on the server.
- Each user has a **personal workspace** (apart from external users).
- *Group*: A group of users working on a set of common projects.
Each group has a **workspace** containing the repositories of these projects.



Direct access to this page

Home Page

Settings
Users
(SSH keys)

The screenshot shows the GitLab Home Page. The top navigation bar includes links for Home, Projects, Groups, Activity, Milestones, and Snippets. A search bar is on the right. The main content area is titled 'Your projects' and lists several projects. Red dots and lines are used to highlight specific features: a dot on the home icon points to 'Direct access to this page'; a dot on the 'Projects' link points to 'Groups to which I belong'; a dot on the 'miscellaneous / cooperative-task-controller' project points to 'List of projects in which I participate'; a dot on the 'New project' button points to 'Creation of Projects/Groups'; a dot on the 'Last updated' dropdown points to 'Research of projects'; and a dot on the top right points to 'Settings Users (SSH keys)'. The project list includes details like project name, owner, description, stars, and update time.

Project Name	Owner	Description	Stars	Updated
miscellaneous / cooperative-task-controller	Owner	A set of classes for dual-arm control using the cooperative task space representation.	0	updated 40 minutes ago
multi-contact / mc_rtc	Owner		6	updated 4 hours ago
rob-miscellaneous / eigen-extensions	Owner	Provide some extensions to the Eigen 3 library	0	updated 1 day ago
mc-hrp4 / hrp4lirmm_doc	Owner	Internal documentation for HRP4LIRMM	0	updated 1 day ago
pid / pid-workspace	Owner	This project defines a build and packaging system based on CMake and git.	2	updated 1 day ago
Robin Passama / pid-workspace	Maintainer	This project defines a build system based on CMake and git.	0	updated 1 day ago
pioneer3DX / pioneer3DX-driver	Owner	Driver software for P3DX robot, home made at LIRMM.	0	updated 1 day ago
rob-miscellaneous / ethercat-driver	Owner	A generic ethercat driver library	0	updated 4 days ago
raven2 / ultrasonix-interface	Owner		0	updated 5 days ago
raven2 / Utilisateur-UltrasonixRF	Owner		0	updated 5 days ago

Group Page

The screenshot shows the GitLab interface for a group named 'pid'. The left sidebar contains navigation links: Overview, Details, Activity, Issues (24), Merge Requests (1), Members, and Settings. The main content area displays the group's profile, description, and a list of projects. A red circle highlights the 'Members' link in the sidebar and the list of projects. A red arrow points from the 'New project' button to the text 'Create a new project'.

Management of Members (recording, right of access)

Create a new project

Access to Contained projects in the group

Project Name	Description	Stars	Created
pid-workspace	This project defines a build and packaging system based on CMake and git.	2	10 months ago
pid-rpath	A specific package of PID used to manage runtime path of executables and libraries.	0	11 months ago
pid-config	Libraries and tools to manage configuration files.	0	11 months ago
pid-log	logging libraries and tools.	0	11 months ago
ext-boost	A project to manage the port into PID system of installable versions of boost.	0	11 months ago
ext-yaml-cpp	A project to manage the port into PID system of installable versions of yaml-cpp project.	0	11 months ago
ext-qtlibs	PID external package wrapper for Qt libraries	0	2 years ago

Project page

Gitlab Features

workspace - GitLab - Google Chrome

https://gite.lirmm.fr/pid/pid-workspace

Applications dev information loisir institutionnel internet review tests en ligne achats GT CAR - Dropbo tech papers

Autres favoris

pid-workspace

Project

Details

Activity

Cycle Analytics

Repository

Issues 21

Merge Requests 1

Settings

pid-workspace

Details

pid-workspace

This project defines a build and packaging system based on CMake and git.

Project ID: 202

Unstar 2 Fork 17 SSH git@gite.lirmm.fr:pid/pid-work

Files (8.6 MB) Commits (1,375) Branches (5) Tags (2) Readme LICENSE

Add Changelog Add Contribution guide Set up CI/CD

master pid-workspace / +

History Find file Web IDE

solved BUG with remove command Robin Passama authored 1 month ago a7082335

Name	Last commit	Last update
environments	clean headers with license	1 year ago
external	changed the structure of gitignoring to preserver hierarcgy...	4 years ago
install	changed the structure of gitignoring to preserver hierarcgy...	4 years ago

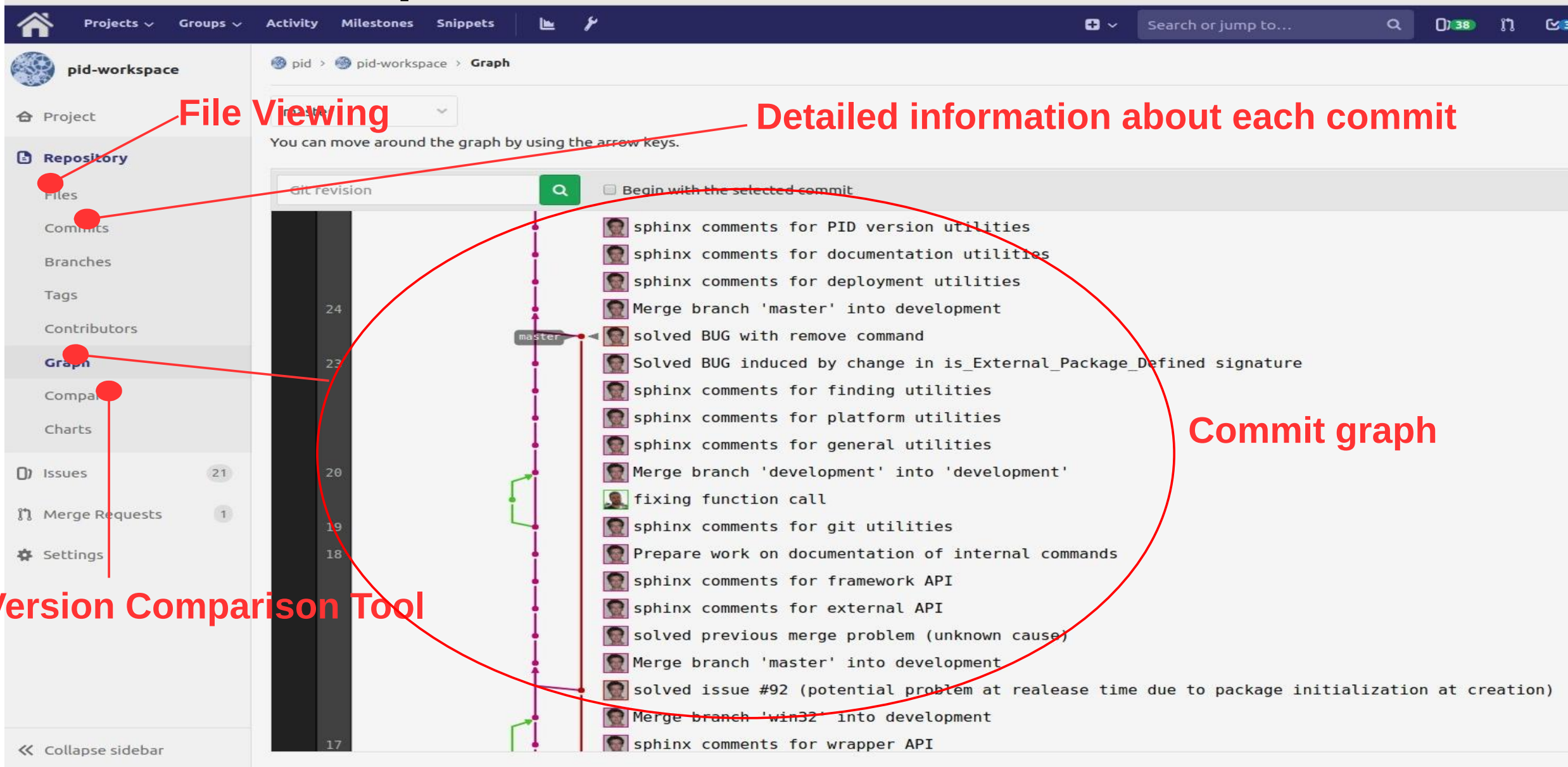
Workspace containing the project

Access to Detailed Deposit Information

Project Visibility (public, internal or private)

Address of the git repository

Issues Management



File Viewing

Detailed information about each commit

Commit graph

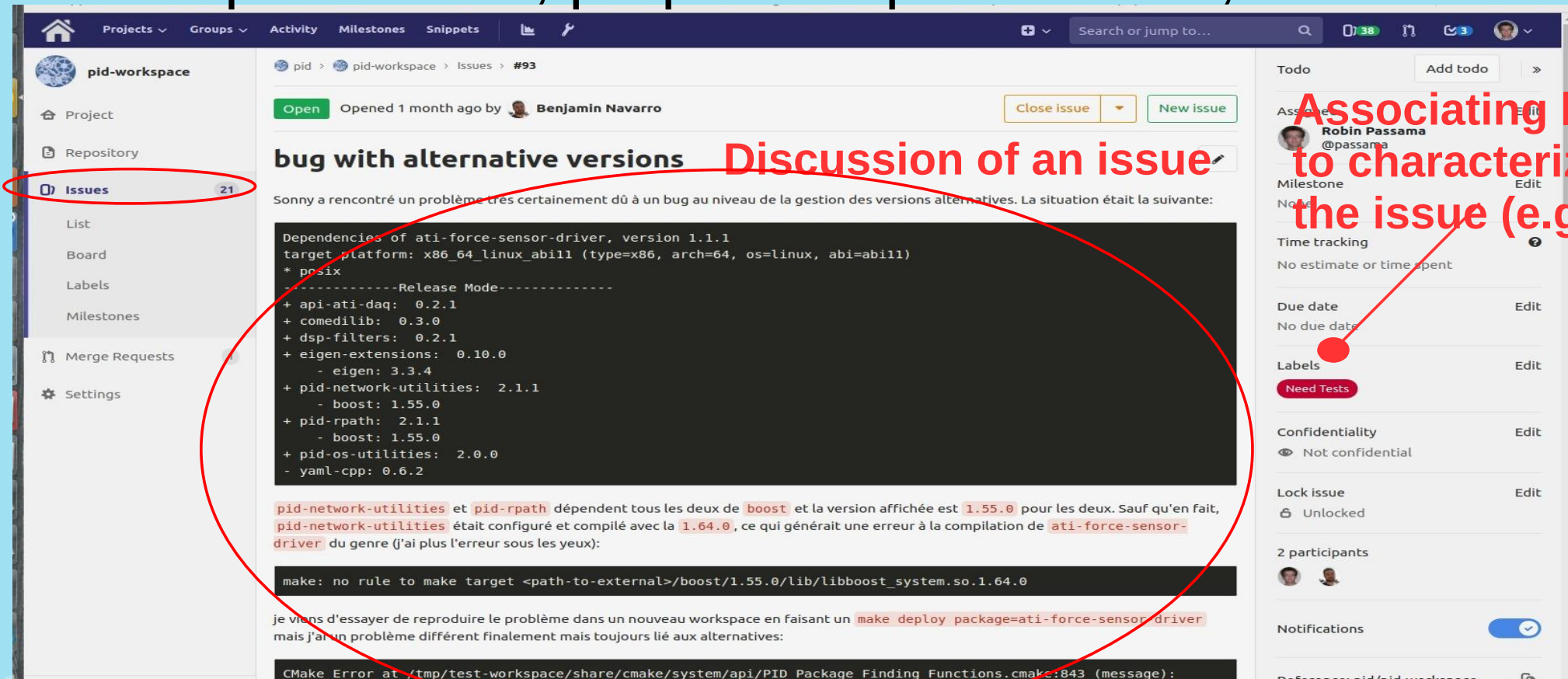
Version Comparison Tool

The screenshot shows the GitLab interface for the 'pid' repository. The sidebar on the left contains navigation links: Project, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Comparison, Charts, Issues (21), Merge Requests (1), and Settings. The main content area displays the commit graph for the 'pid' repository. The graph shows a vertical sequence of commits, with the 'master' branch highlighted. The commit messages are listed on the right, and the commit hashes are listed on the left. The commit messages include: 'sphinx comments for PID version utilities', 'sphinx comments for documentation utilities', 'sphinx comments for deployment utilities', 'Merge branch 'master' into development', 'solved BUG with remove command', 'Solved BUG induced by change in is_External_Package_Defined signature', 'sphinx comments for finding utilities', 'sphinx comments for platform utilities', 'sphinx comments for general utilities', 'Merge branch 'development' into 'development'', 'fixing function call', 'sphinx comments for git utilities', 'Prepare work on documentation of internal commands', 'sphinx comments for framework API', 'sphinx comments for external API', 'solved previous merge problem (unknown cause)', 'Merge branch 'master' into development', 'solved issue #92 (potential problem at release time due to package initialization at creation)', 'Merge branch 'win32' into development', and 'sphinx comments for wrapper API'.

Developer/User Discussion

- Report BUGS, propose improvements, etc.

68



The screenshot shows a GitHub issue page for the 'pid-workspace' repository. The issue is titled 'bug with alternative versions' and is labeled 'Open'. It was opened 1 month ago by Benjamin Navarro. The issue description is in French and mentions a problem with alternative versions. A code block shows the dependencies of 'ati-force-sensor-driver' version 1.1.1. The dependencies list includes 'boost' version 1.55.0, which is highlighted in red. The issue is assigned to Robin Passama (@passama). The right sidebar shows the issue's metadata, including labels, confidentiality, and participants.

bug with alternative versions Discussion of an issue

Sonny a rencontré un problème très certainement dû à un bug au niveau de la gestion des versions alternatives. La situation était la suivante:

```
Dependencies of ati-force-sensor-driver, version 1.1.1
target platform: x86_64_linux_abi11 (type=x86, arch=64, os=linux, abi=abi11)
* posix
-----Release Mode-----
+ api-ati-daq: 0.2.1
+ comedilib: 0.3.0
+ dsp-filters: 0.2.1
+ eigen-extensions: 0.10.0
  - eigen: 3.3.4
+ pid-network-utilities: 2.1.1
  - boost: 1.55.0
+ pid-rpath: 2.1.1
  - boost: 1.55.0
+ pid-os-utilities: 2.0.0
- yaml-cpp: 0.6.2
```

pid-network-utilities et pid-rpath dépendent tous les deux de boost et la version affichée est 1.55.0 pour les deux. Sauf qu'en fait, pid-network-utilities était configuré et compilé avec la 1.64.0, ce qui générant une erreur à la compilation de ati-force-sensor-driver du genre (j'ai plus l'erreur sous les yeux):

```
make: no rule to make target <path-to-external>/boost/1.55.0/lib/libboost_system.so.1.64.0
```

je viens d'essayer de reproduire le problème dans un nouveau workspace en faisant un make deploy package=ati-force-sensor-driver mais j'ai un problème différent finalement mais toujours lié aux alternatives:

```
CMake Error at /tmp/test-workspace/share/cmake/system/api/PID Package Finding Functions.cmake:843 (message):
```

Associating labels to characterize the issue (e.g. BUG)

Board : Organize the processing of issues.

69

The screenshot displays the 'pid-workspace' interface with a sidebar on the left containing 'Project', 'Repository', 'Issues' (21), 'List', 'Board' (highlighted with a red circle), 'Labels', 'Milestones', 'Merge Requests' (1), and 'Settings'. The main area is titled 'Issue Boards' and features three columns: 'Untreated' (Backlog, 5 items), 'Ongoing' (Work In Progress, 5 items), and 'In test' (Need Tests, 11 items). Each column contains a list of issues with their titles, IDs, and labels. Red circles and lines highlight the column headers and the 'Board' link in the sidebar.

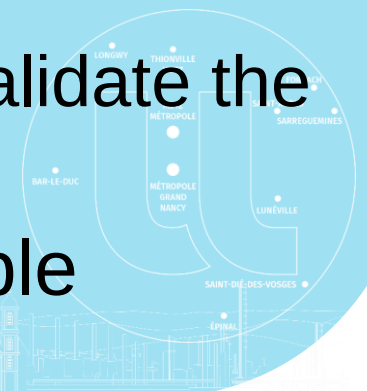
Untreated	Ongoing	In test
Command to remove unused installed versions #70 suggestion	Little improvements: #77 Work In Progress enhancement	PROBLEM with external packages version resolution #64 Need Tests bug confirmed critical
Option to directly deploy the integration branch of a package #72 suggestion	About binaries compatibility #80 Work In Progress discussion	Better dependencies resolution when considering binaries and alternatives #54 Need Tests enhancement suggestion
Conflicting C and CXX standards in .clang_complete #73 bug	Packaging compilers #67 Work In Progress discussion suggestion	lsb_release is not part of all linux distribution #82 Need Tests
Cleanup script for plugins #75	Windows support #87 Work In Progress	Manage patch version for native package if greater than currently used #78 Need Tests bug
Improvements for the external API #83	PID and mc_rtc #90 Work In Progress discussion enhancement	Profiling enabled on CI builds #76 Need Tests
		Local modifications lost #31 Need Tests bug critical
		Transform -isystem include in -I ones #86 Need Tests enhancement

Github-like Work-flow:

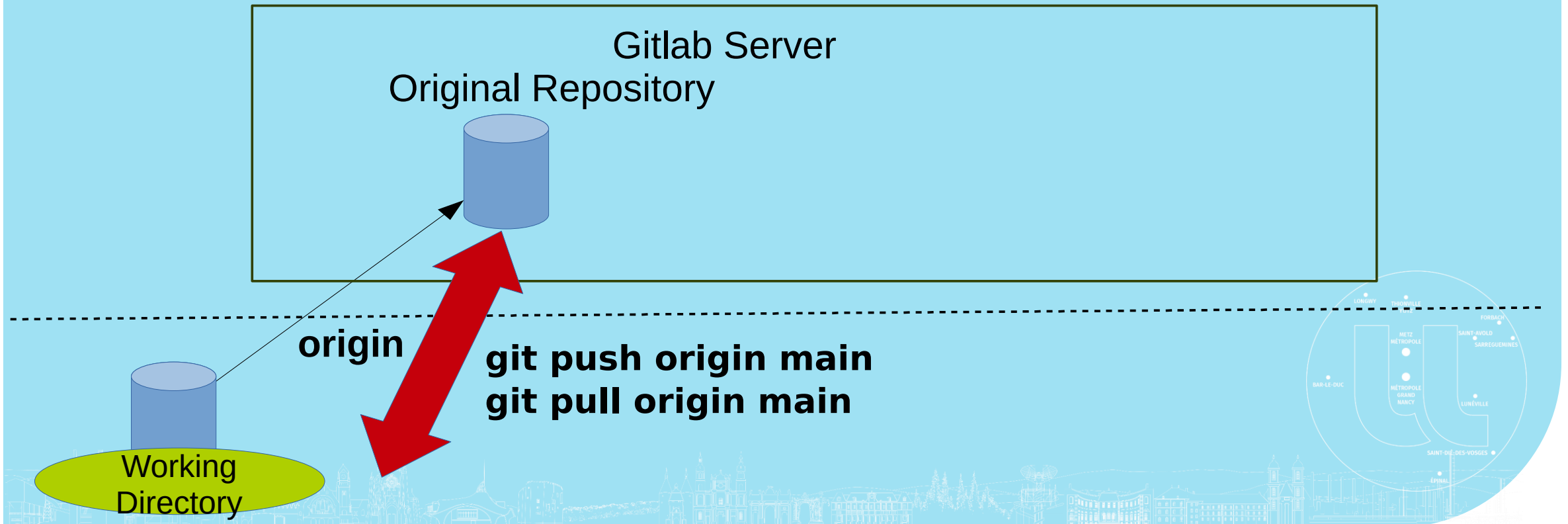
- *Fork* = clone a repository directly in Gitlab (the created repository exists in the server)
 - The **original** repository "knows" its **cloned** repositories.
- Merge request = propose to merge a branch of the **cloned** repository into a branch of the **original** repository.
 - The request can be discussed, approved or rejected.
 - Opportunity to see proposed changes.
 - Only the developers of the original repository can validate the request.



Allows you to receive contributions from people outside the project

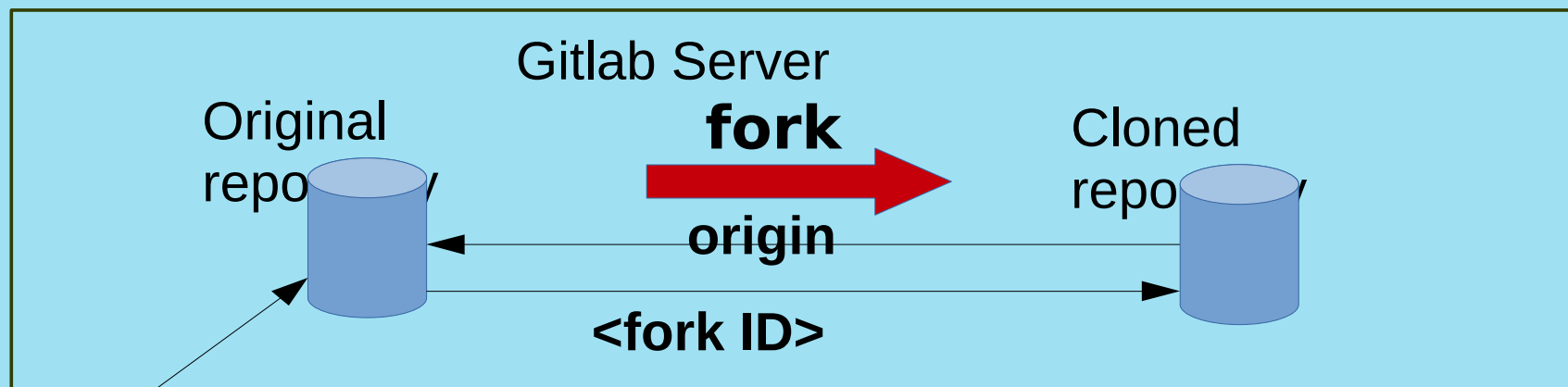


Initial situation



fork

Clone one repository into another *workspace*



forke

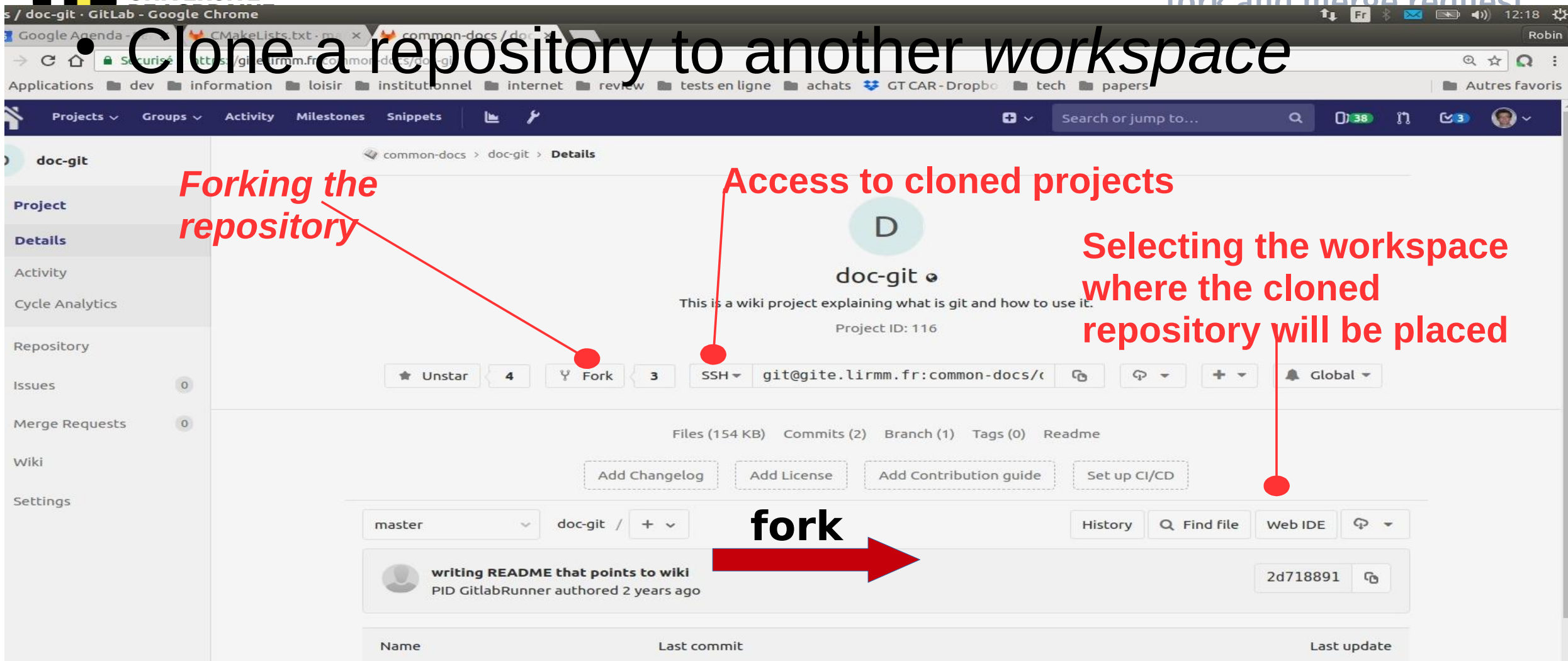
fork and merge request

• Clone a repository to another workspace

Forking the repository

Access to cloned projects

Selecting the workspace where the cloned repository will be placed



The screenshot shows the GitLab interface for a repository named 'doc-git'. The sidebar on the left contains links to Project, Details, Activity, Cycle Analytics, Repository, Issues (0), Merge Requests (0), Wiki, and Settings. The main content area displays the repository details, including the project name 'doc-git', a description 'This is a wiki project explaining what is git and how to use it.', and the Project ID '116'. Below this, there are buttons for 'Unstar', '4' stars, 'Fork' (3 forks), and 'SSH' clone URL 'git@gite.lirmm.fr:common-docs/...'. There are also buttons for 'Add Changelog', 'Add License', 'Add Contribution guide', and 'Set up CI/CD'. At the bottom, there is a commit history table with columns 'Name', 'Last commit', and 'Last update'. A large red arrow labeled 'fork' points from the 'Fork' button to the commit history section.

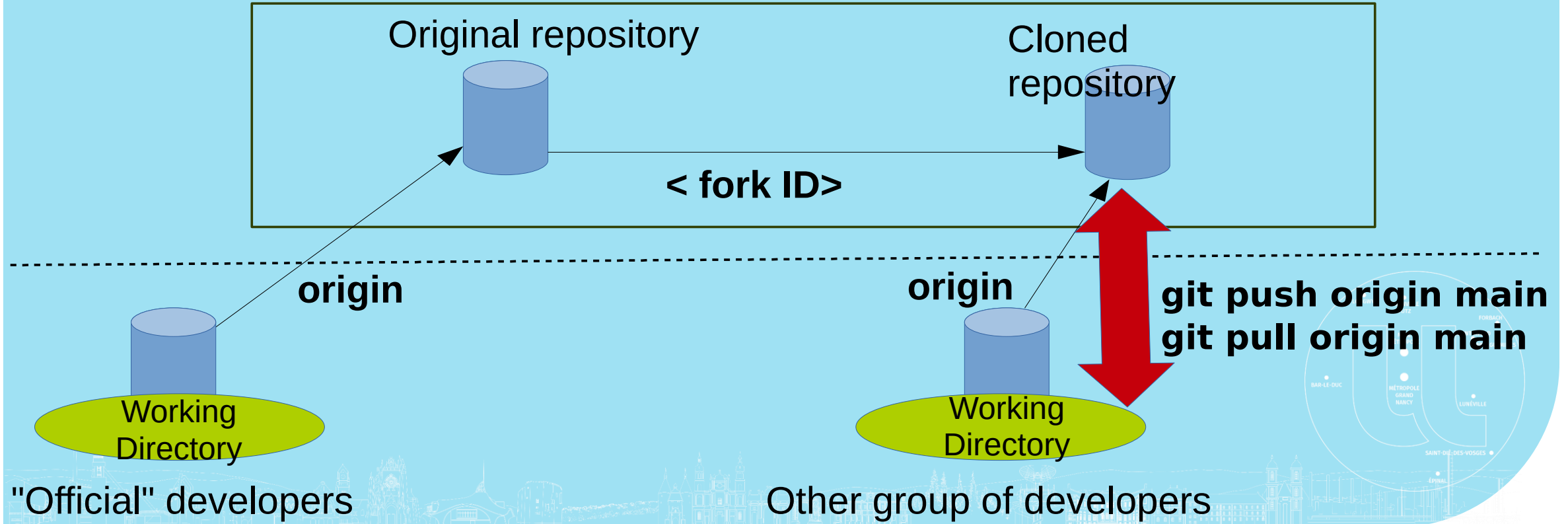
Name	Last commit	Last update
writing README that points to wiki	PID GitlabRunner authored 2 years ago	2d718891



fork

Outside developers work in isolation

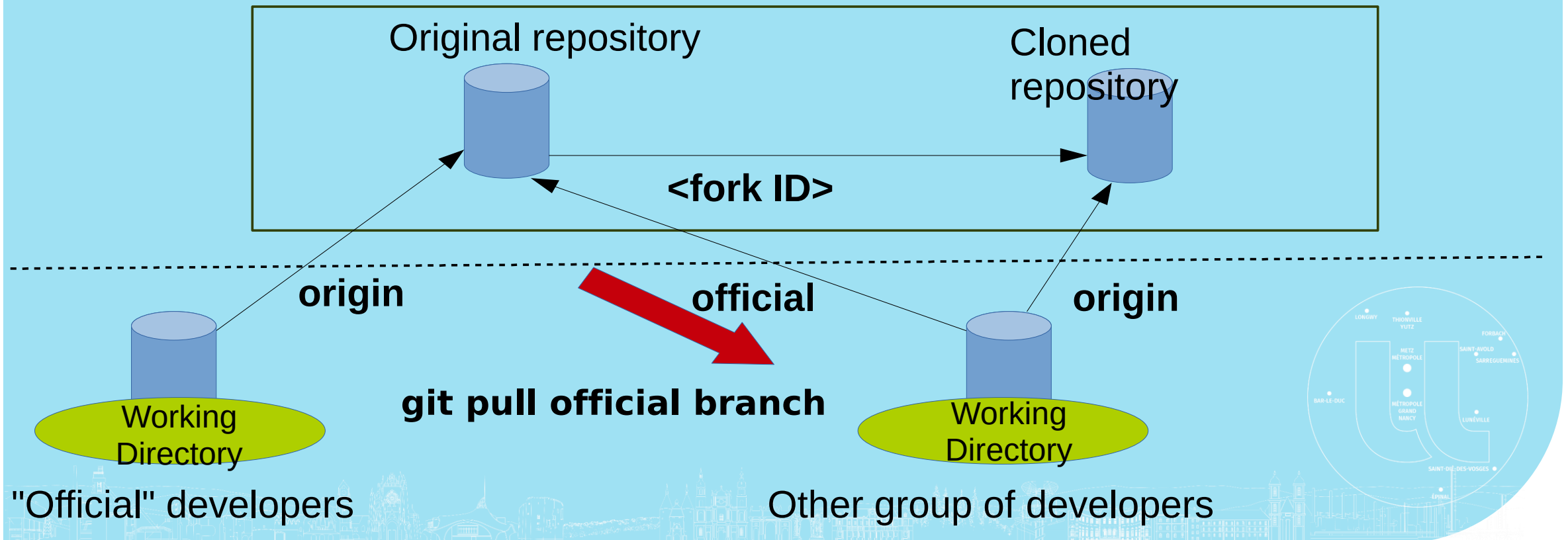
Gitlab Server



fork

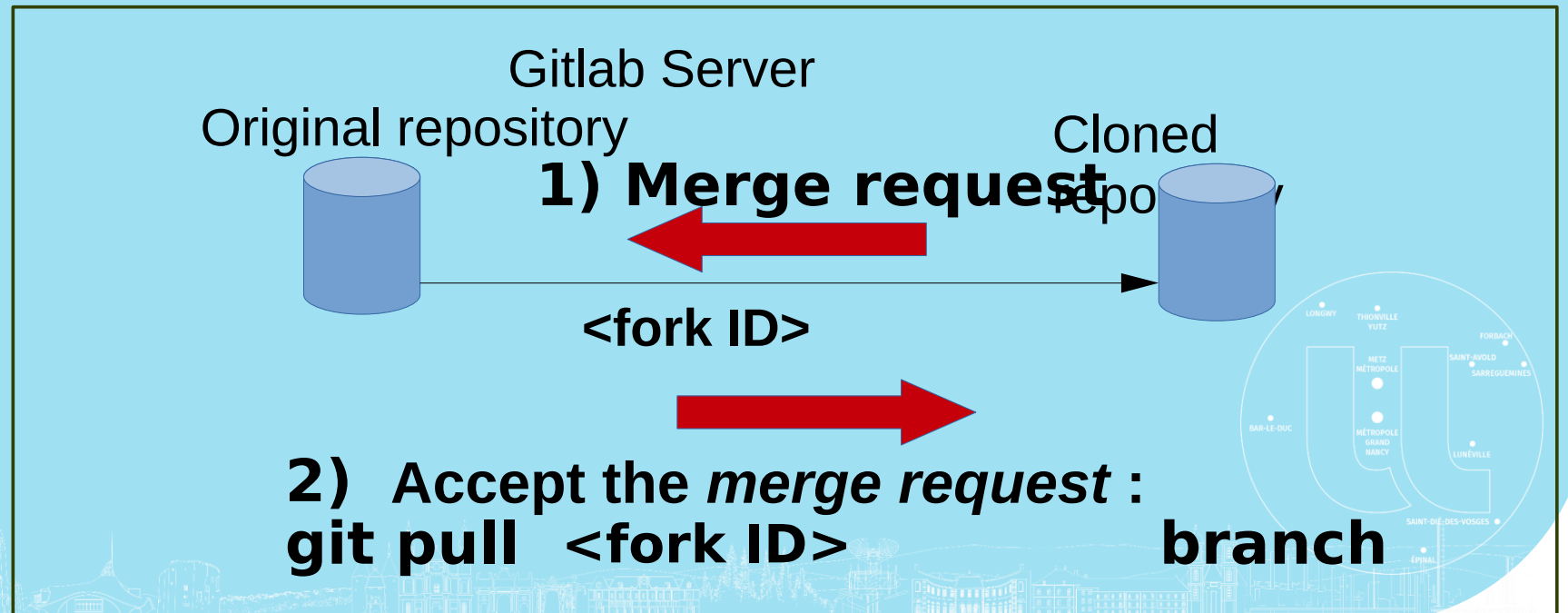
Update a cloned repository from the original repository

Serveur Gitlab



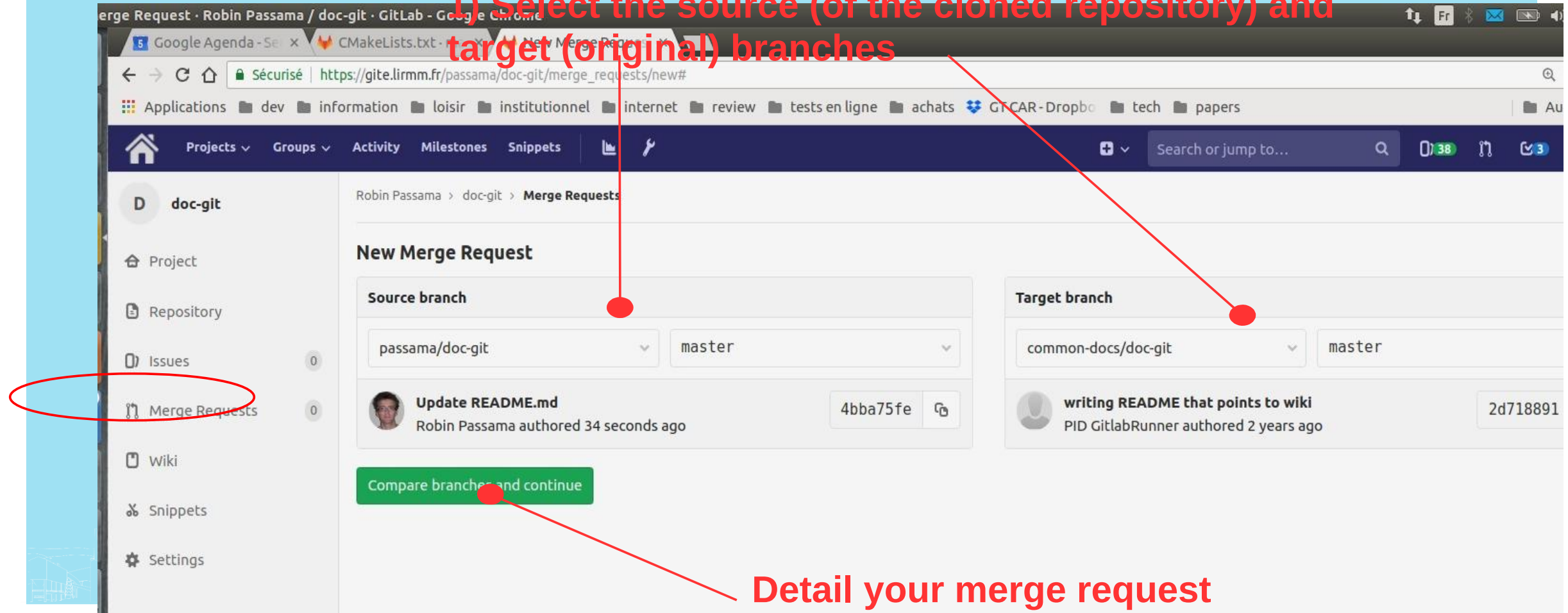
Proposal to **merge branches** from cloned repository to original repository 76

- Developers of the cloned repository propose the merging of a branch.
- Developers of the original repository decide whether the merge is completed.



Propose to merge

1) Select the source (of the cloned repository) and target (original) branches



The screenshot shows the 'New Merge Request' page in GitLab. The left sidebar contains a menu with 'Merge Requests' circled in red. The main content area has two columns for selecting source and target branches. Red dots mark the dropdown menus for 'passama/doc-git' and 'common-docs/doc-git'. A red arrow points from the text 'Detail your merge request' to the 'Compare branches and continue' button.

Source branch

passama/doc-git master

Target branch

common-docs/doc-git master

Update README.md
Robin Passama authored 34 seconds ago 4bba75fe

writing README that points to wiki
PID GitlabRunner authored 2 years ago 2d718891

Compare branches and continue

Detail your merge request

Propose to merge

request · Robin Passama / doc-git · GitLab - Google Chrome

Google Agenda - Se x CMakeLists.txt · ma x New Merge Reques x

→ ↻ 🏠 🔒 Sécurisé | https://gite.lirmm.fr/passama/doc-git/merge_requests/new?utf8=✓&merge_request%5Bsource_project_id%5D=2491&merge_request%5Bsource_branch%5D=master&merge_request%5Btarget_pr...

Applications dev information loisir institutionnel internet review tests en ligne achats GT CAR - Dropbo tech papers

Autres favoris

Projects Groups Activity Milestones Snippets Search or jump to...

doc-git

project

pository

ues

erge Requests

ki

ippets

ettings

New Merge Request

From passama/doc-git:master into common-docs/doc-git:master

Change branches

Title

Update README.md

Start the title with **WIP:** to prevent a **Work In Progress** merge request from being merged before it's ready.

Add description templates to help your contributors communicate effectively!

Description

Write Preview

bla bla bla

Markdown and quick actions are supported

Attach a file

Assignee

Robin Passama

Milestone

Milestone

Labels

Labels

Source branch

master

Target branch

master

Change branches

☒ Squash commits when merge request is accepted. [About this feature](#)

Contribution

☐ Allow commits from members who can merge to the target branch. [About this feature](#)
Not available for protected branches

Submit merge request

Cancel

Commits 1

Changes 1

05 Sep, 2018 1 commit



Update README.md

Robin Passama authored 3 minutes ago

4bba75fe



llapse sidebar

Explanation of the *merge request*

Assignee : developer of the original repository
in charge of processing the *merge request*

Commits that will be added to the original
branch if the *merge request* is accepted

Accept or reject the merging

fork and merge request

The screenshot shows a GitLab merge request interface. The browser address bar displays the URL `https://gite.lirmm.fr/common-docs/doc-git/merge_requests/2`. The page title is "Update README.md". The merge request is "Open" and was "Opened 15 seconds ago by Robin Passama". The source is "passama:master" and the target is "master". The "Merge" button is green and has a checkmark icon. The "Close merge request" button is yellow. The "Discussion" tab is selected, showing a "Write" field and a "Preview" tab. The "Commits" and "Changes" tabs are also visible. The "Developer thread" is circled in red. The "Refuse Request (Close)" button is highlighted in red. The "Origin of the request" is indicated by a red line pointing to the source branch. The "Accept the request (merge)" button is highlighted in red. The "Commits and Changes Proposed" section is highlighted in red. The "Developer thread" is highlighted in red. The "Refuse Request (Close)" button is highlighted in red.

Origin of the request

Accept the request (merge)

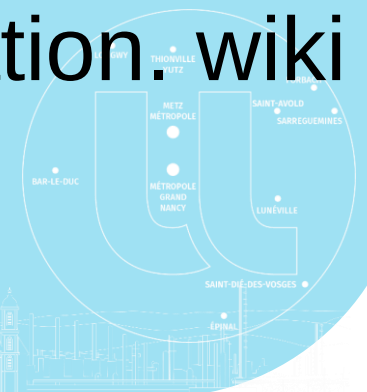
Commits and Changes Proposed

Developer thread

Refuse Request (Close)

 git **GitLab**

- The basic concepts and commands were shown...
- ... but there are a lot of refinements to discover!!
- Gitlab allows you to manage git repositories hosted on a server ...
- ... But it allows you to do many other things: continuous integration and deployment; Static site generation; wiki editing; etc.





Learn Git & GitLab

Version control for better software, together

-  **Track changes**
Never lose your work
-  **Collaborate**
Work together efficiently
-  **Ship better code**
Build great projects

Today's goals

- ✓ Clone a repository
- ✓ Create a branch
- ✓ Make a commit
- ✓ Push to GitLab
- ✓ Open a merge request
- ✓ Review code
- ✓ Merge! 😊

Work locally

```
$ git clone <url>
$ git checkout -b feature
$ git add .
$ git commit -m "Add feature"
$ git push origin feature
```

Work together

- Push your code
- Open a merge request
- Review & discuss
- Merge to main

Code. Review. Improve. Together.

Small commits. Big progress.

GitLab

AwesomeProject

- Project overview
- Issues 3
- Merge requests 2
- CI/CD
- Repositories
- Wiki
- Members

Merge request
feature/login → main Open

Great work!
Let's review and merge this.
👍 2 💬 3

GitLab

Merge requests

- Open 2 Merged Closed
- Add login feature Open
- Improve documentation Open

Changes 3 Discussions 1

app.py

```
42 *
43 + def login(user):
44 +     return authenticate(user)
45 - return None
46
```

Better code. Together.

GitLab

- Collaborate
- Review
- Build
- Deliver

Version Control
Software Engineering

