Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

Git-GitLab tutorial

DU Data Science (UCA) – March 2025

Gino Marchetti (CC-IN2P3)

Git and the “Git*b”

Git

- *Free and open-source distributed version control system.*
- **Branching and Merging:** developers may create separate branches for features, bug fixes, or experiments, seamlessly merging them back into the main codebase after review.
- **Distributed nature:** Every developer has a complete local repository, eliminating single points of failure and enabling offline work.
- **Staging area:** The staging area provides granular control over what changes are committed, enhancing code organization and ensuring only intended modifications are recorded.

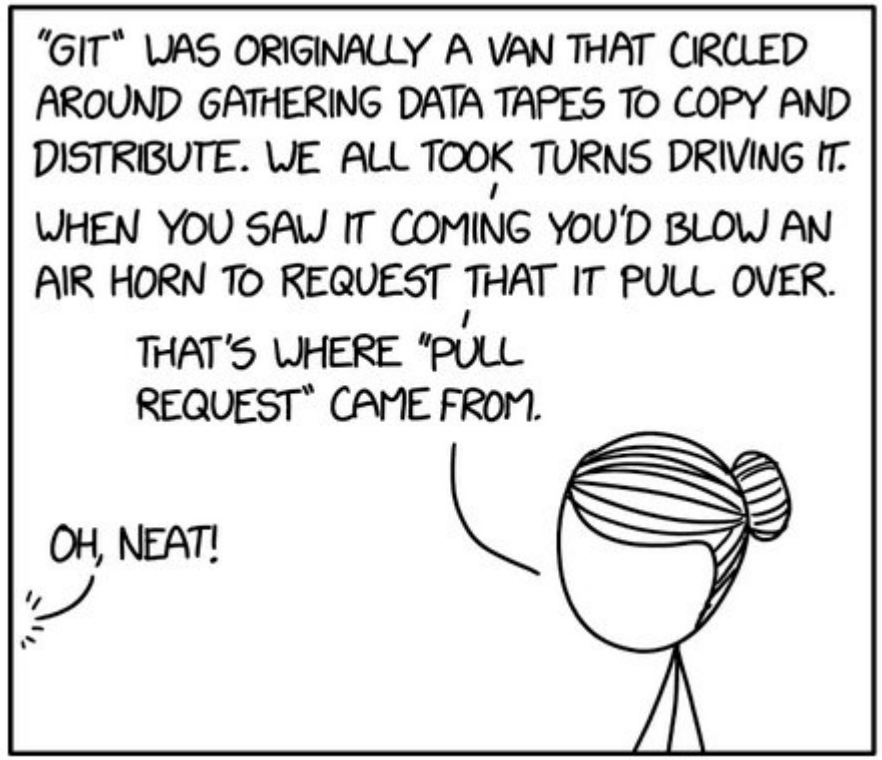
GitHub

- *Cloud-based hosting service that provides a user-friendly web interface for managing Git repositories.*
- **Remote repository hosting:** hosting Git repositories remotely, providing a centralized location for developers to push and pull code changes.
- **Collaboration and Social coding:** allowing developers to follow projects, contribute code, and interact through discussions, issues, and pull requests.
- **Issue tracking:** enabling teams to report bugs, propose new features, and manage project tasks effectively.
- **Pull requests and Code review:** allowing developers to propose changes through pull requests, receive feedback, and merge code into the main codebase.
- **Project management tools:** boards, wikis, and project tracking tools.

GitLab

- *Web-based platform that enables development and deployment workflows from Git repositories.*
- **Similar to GitHub for:**
  - Repository hosting,
  - Issue tracking,
  - Code review and collaboration.
- **Continuous Integration / Continuous Deployment (CI/CD):** allowing teams to automate the entire software delivery process, from code commit to production deployment.
- **Integrated DevOps tools:** features for container management, monitoring, and security scanning. The entire development lifecycle within a single platform.

The Old Days...



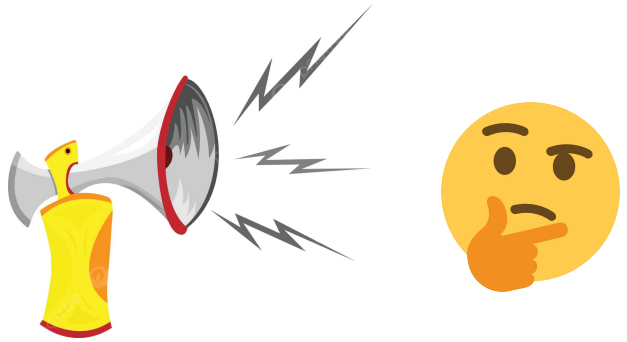
https://xkcd.com/2324/

➢ True

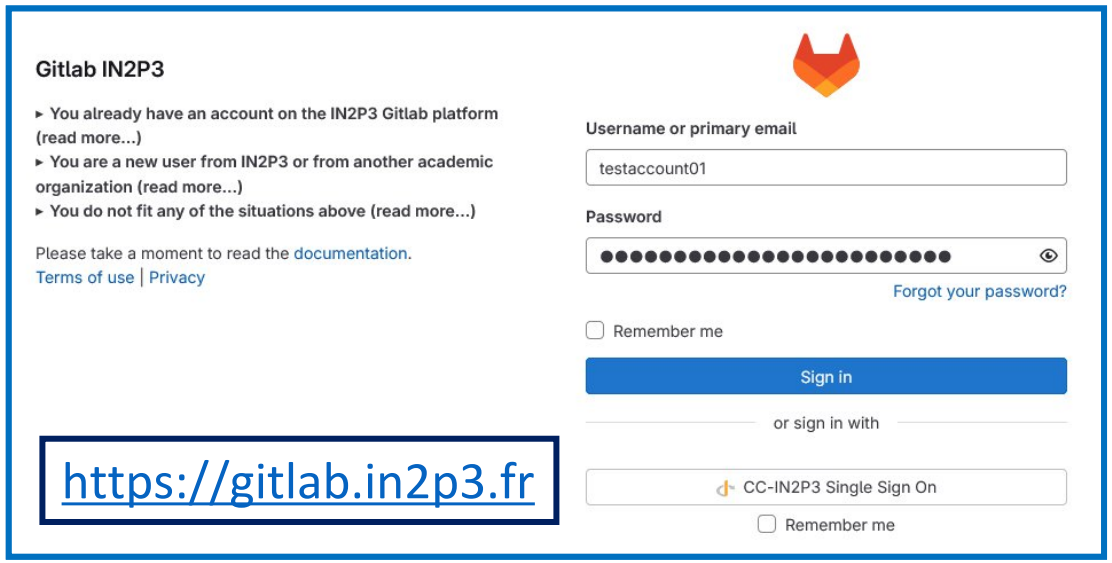
- Indeed there was (for example) a van transferring data tapes from CC-IN2P3 to CERN and vice-versa.

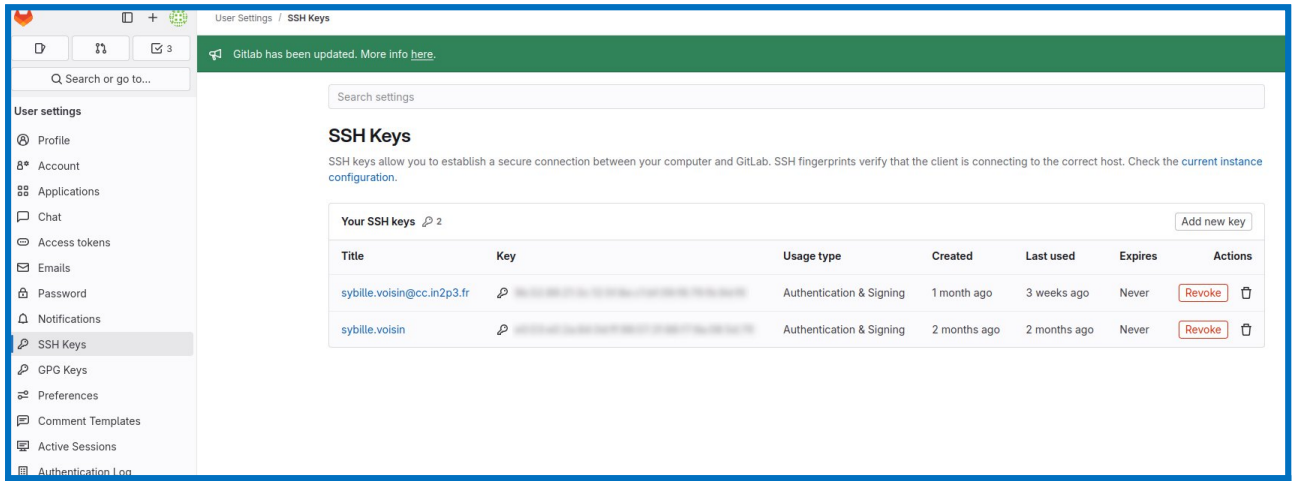
➢ Obviously false

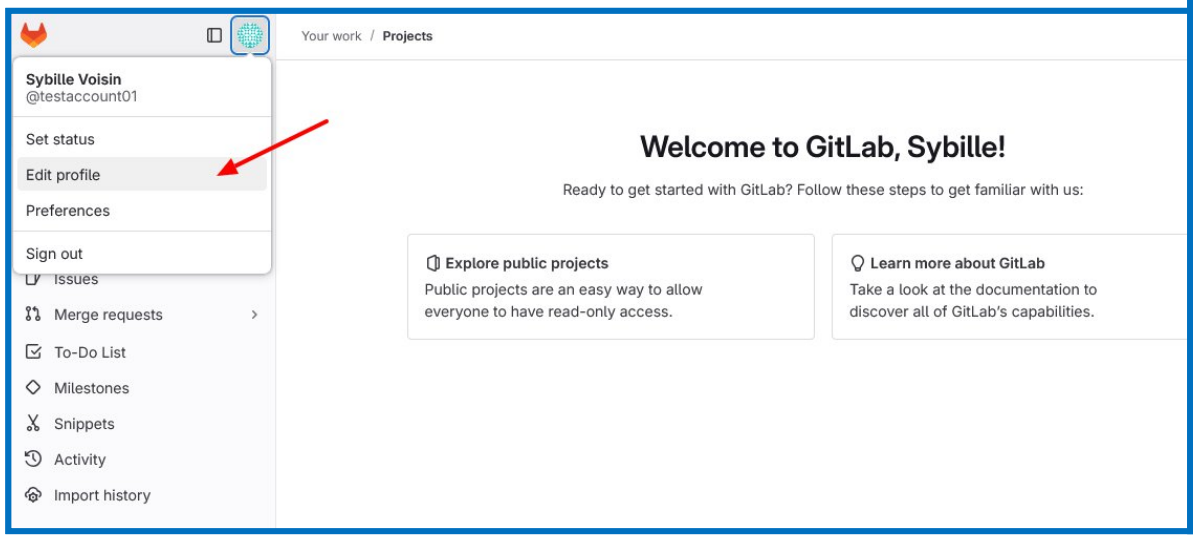
- Do you really think giving a air-horn to a bunch of bona fide physicists / computer scientists would be a good idea?

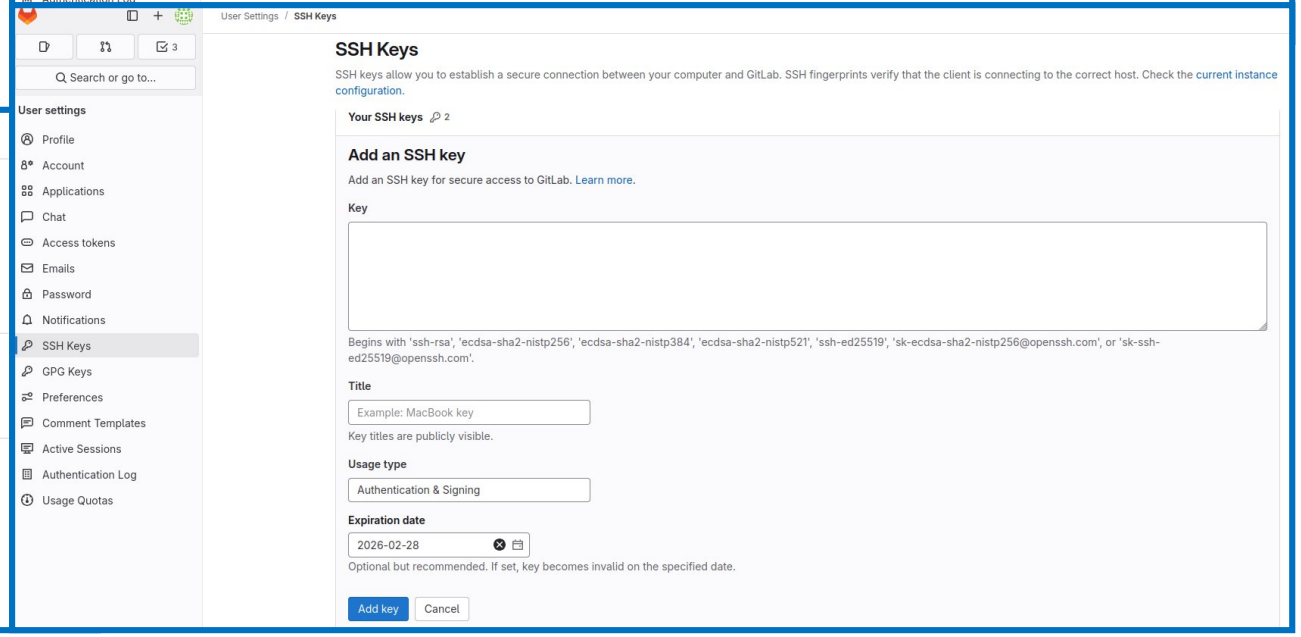


You logged in GitLab and added an SSH key...

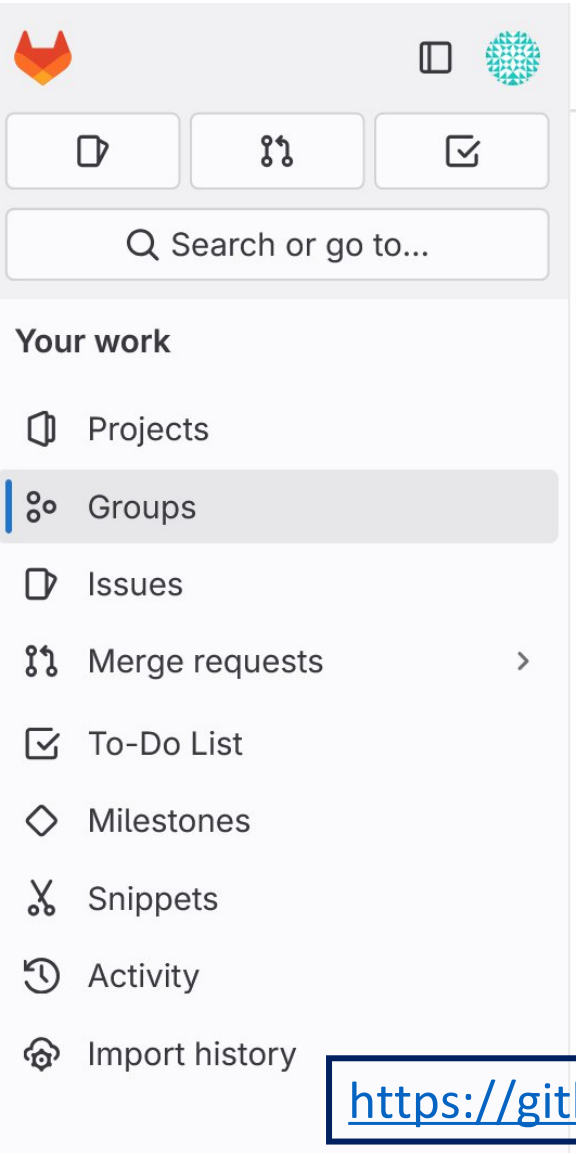


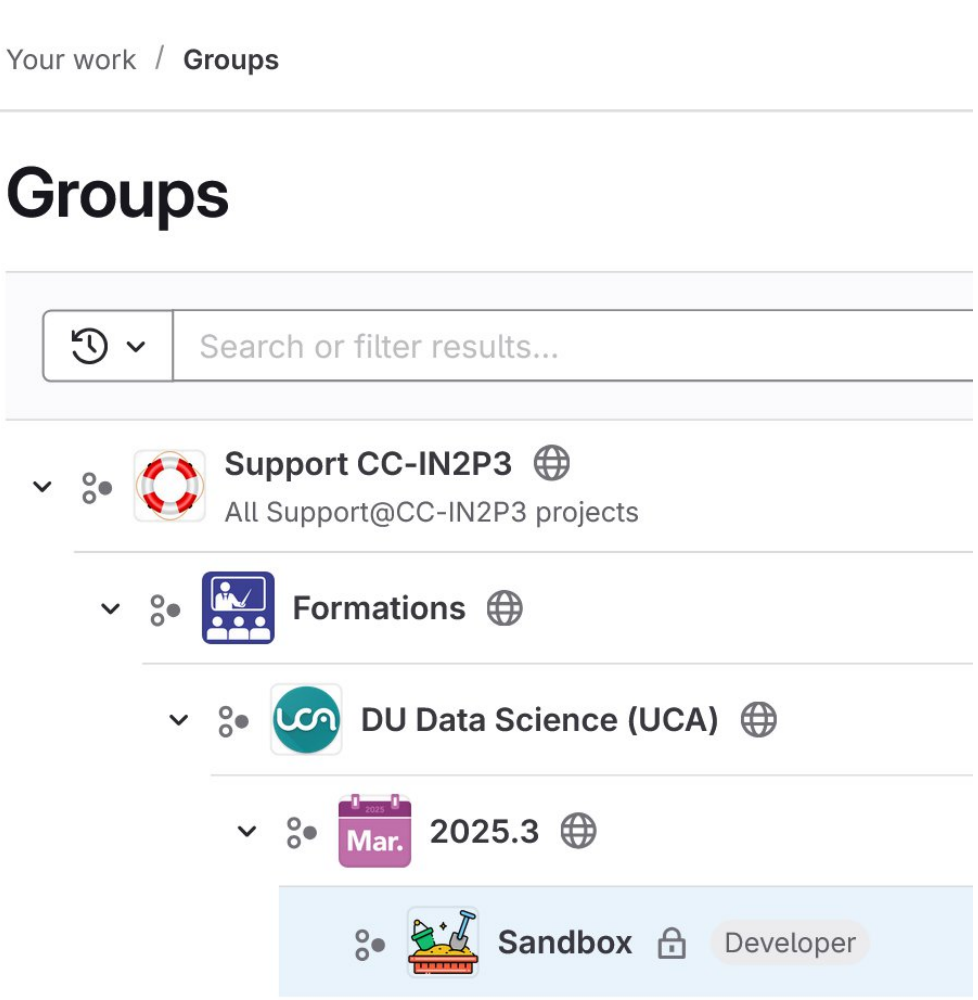






Create a project on GitLab





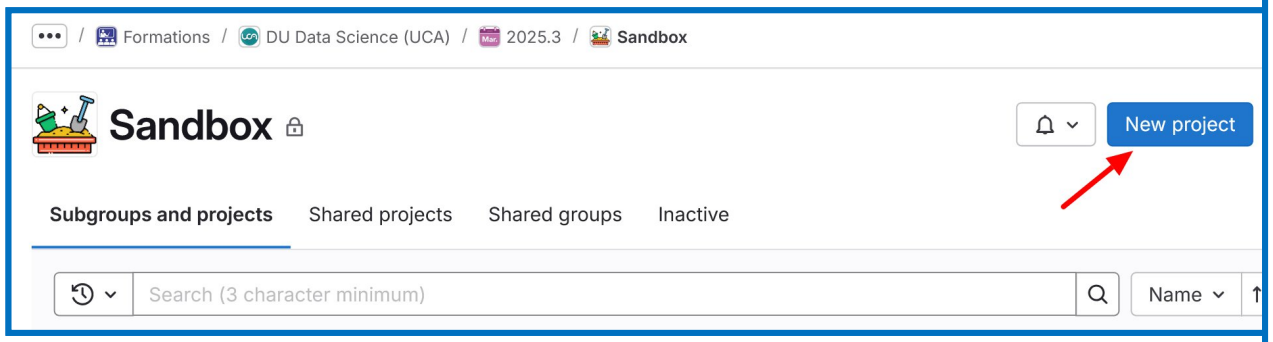
➢ Step 1

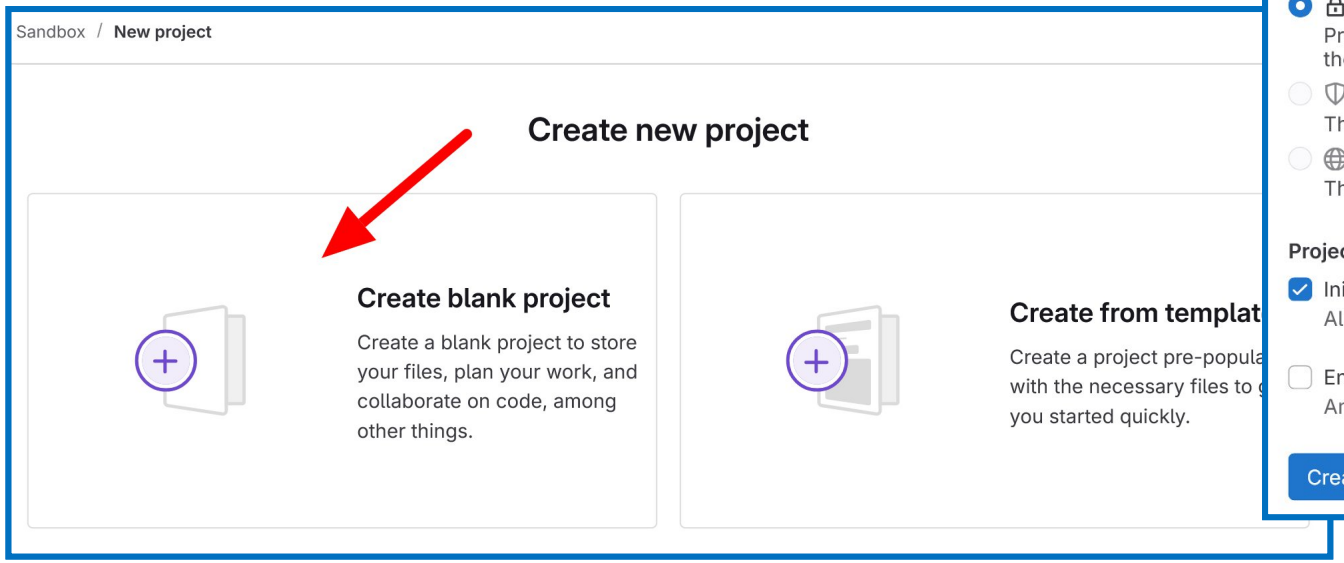
- ➢ Check in the **Sandbox** sub-group
- ➢ Training accounts are allowed to create projects in there!

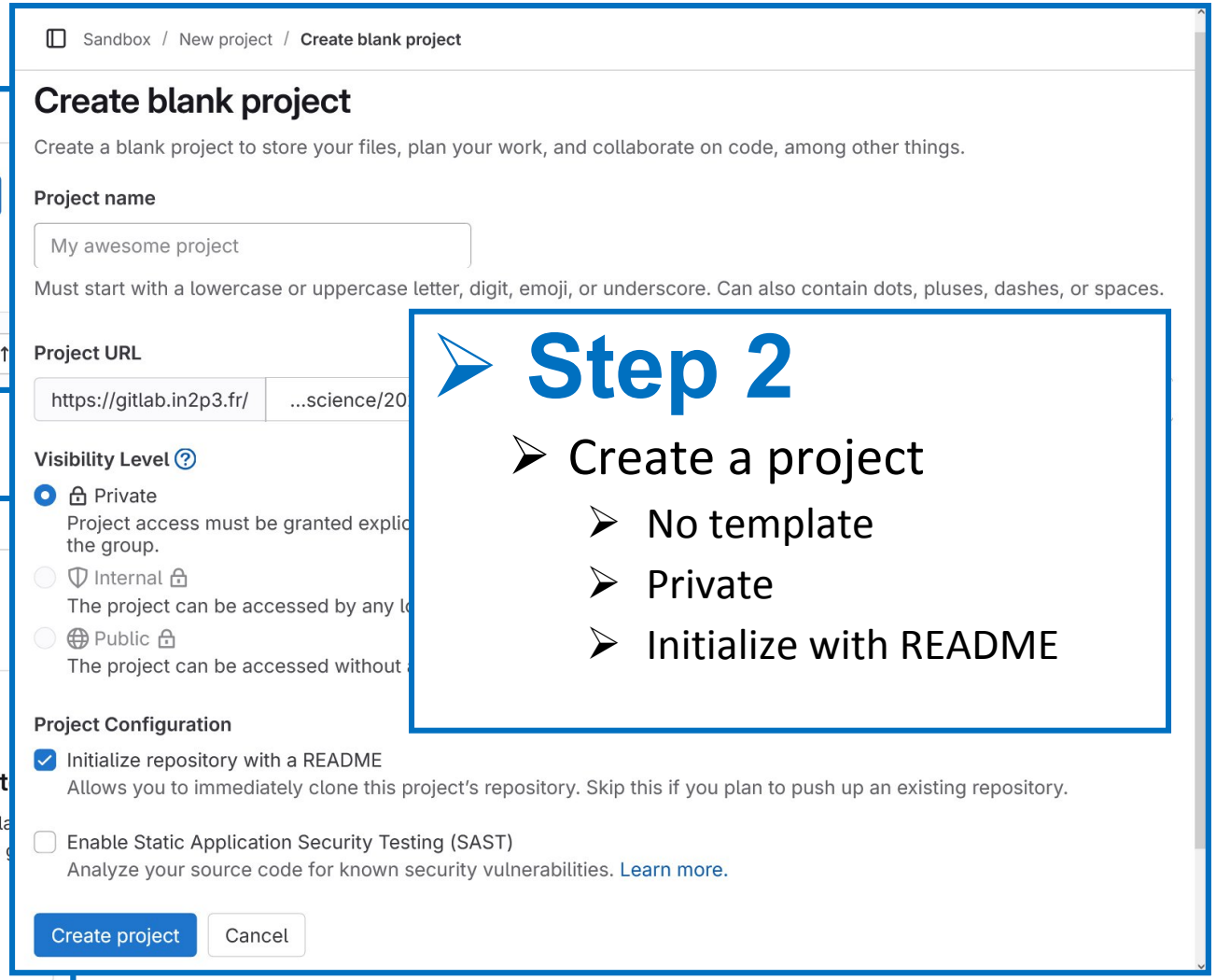
https://gitlab.in2p3.fr/ccin2p3-support/formations/du-data-science/2025.3/sandbox

Create a project on GitLab

https://gitlab.in2p3.fr/ccin2p3-support/formations/du-data-science/2025.3/sandbox



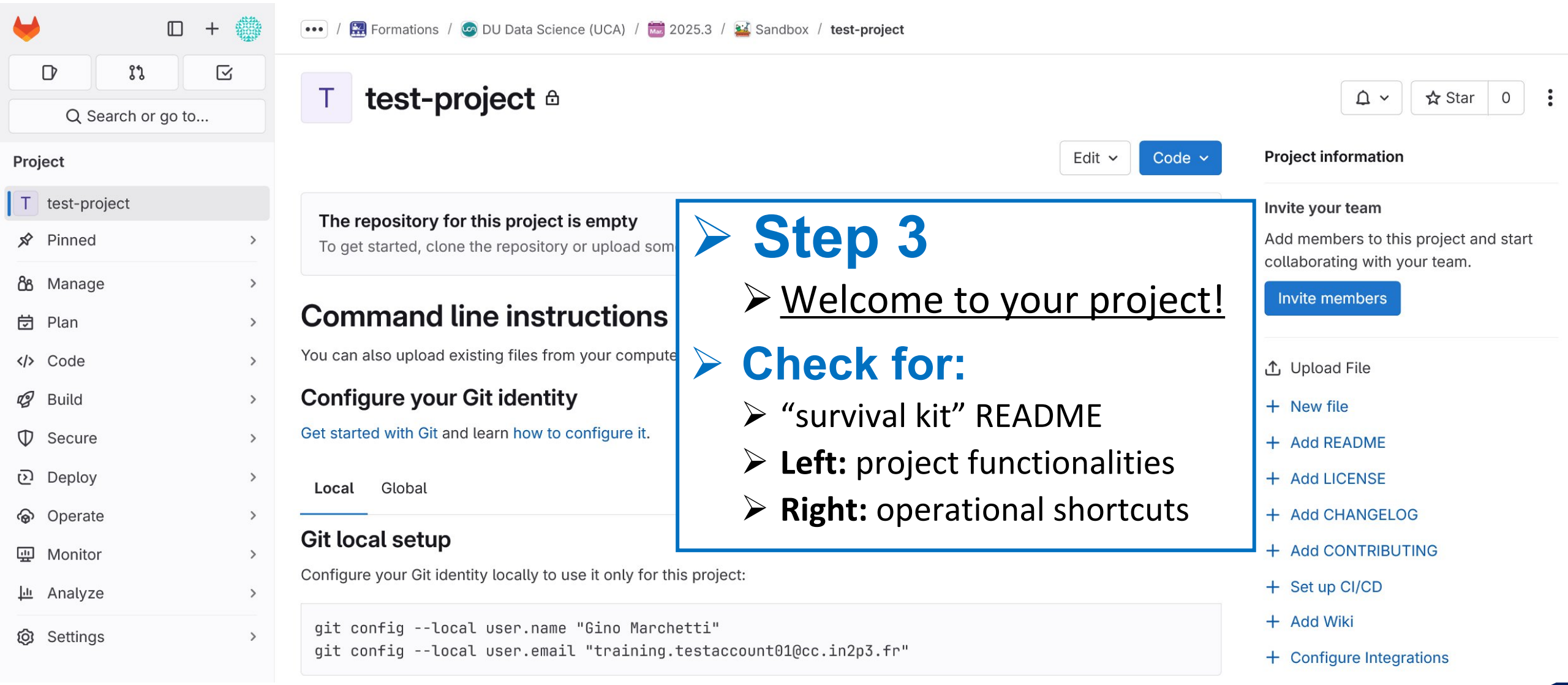




Step 2

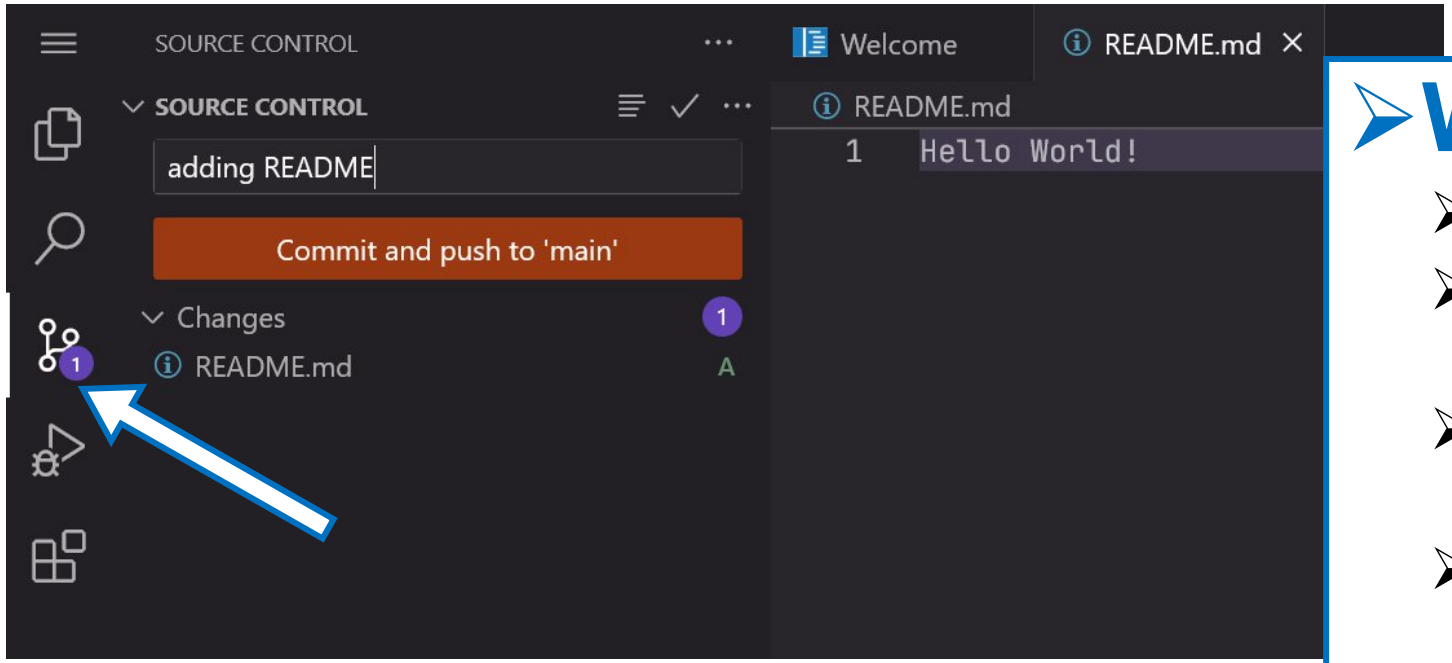
- Create a project
  - No template
  - Private
  - Initialize with README

Create a project on GitLab



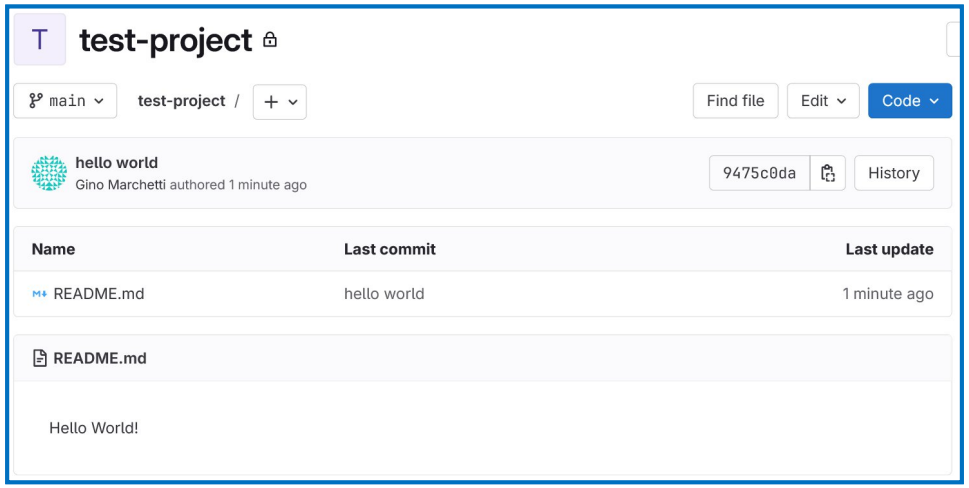
- **Step 3**
  - Welcome to your project!
- **Check for:**
  - "survival kit" README
  - **Left:** project functionalities
  - **Right:** operational shortcuts

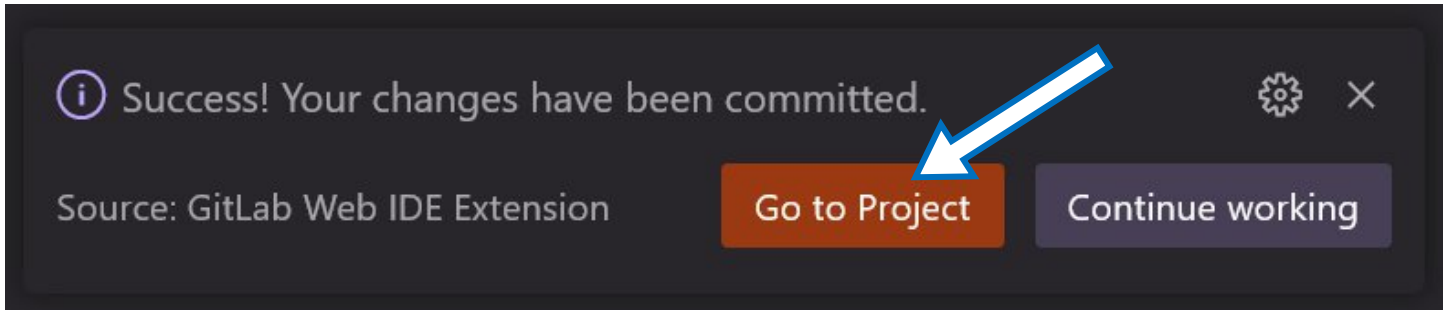
Add your first file from the interface + Add README



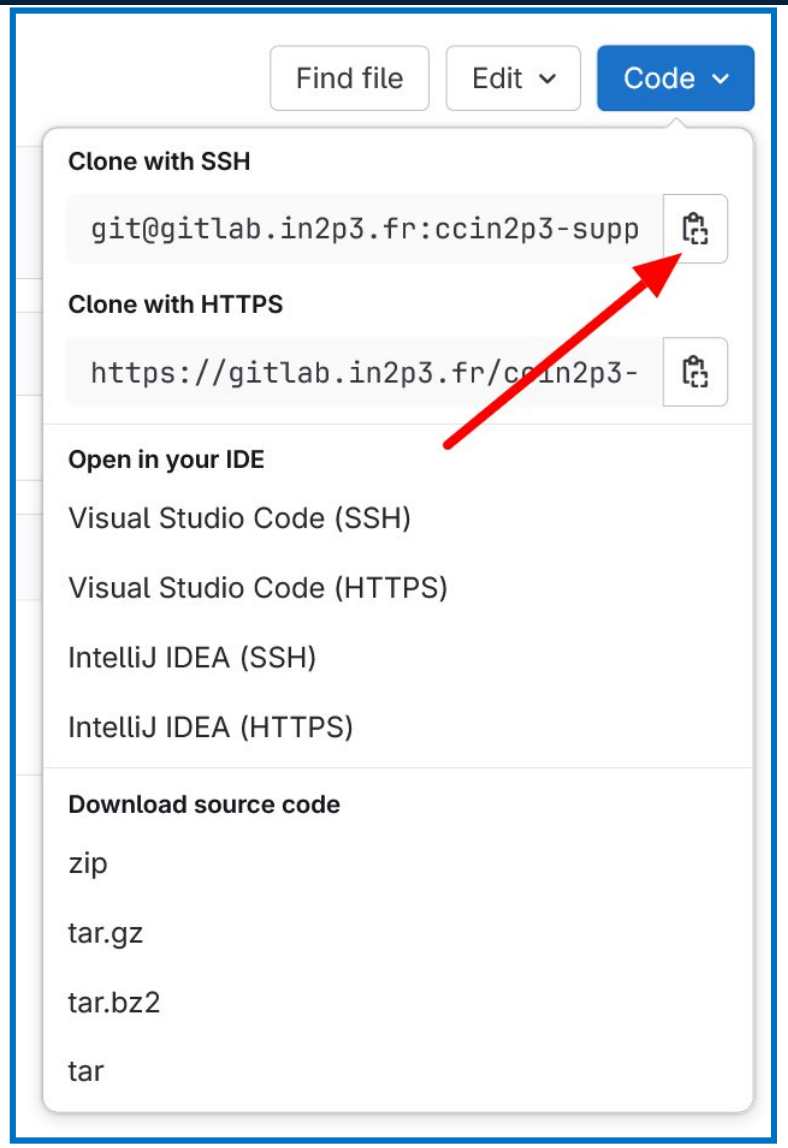
Web IDE

- Write your text in Markdown format
- Changes will be highlighted for your actions
- Add a commit message to “push”your modifications
- Use the pop-up message to get back to your project





Clone your remote project on your local workstation

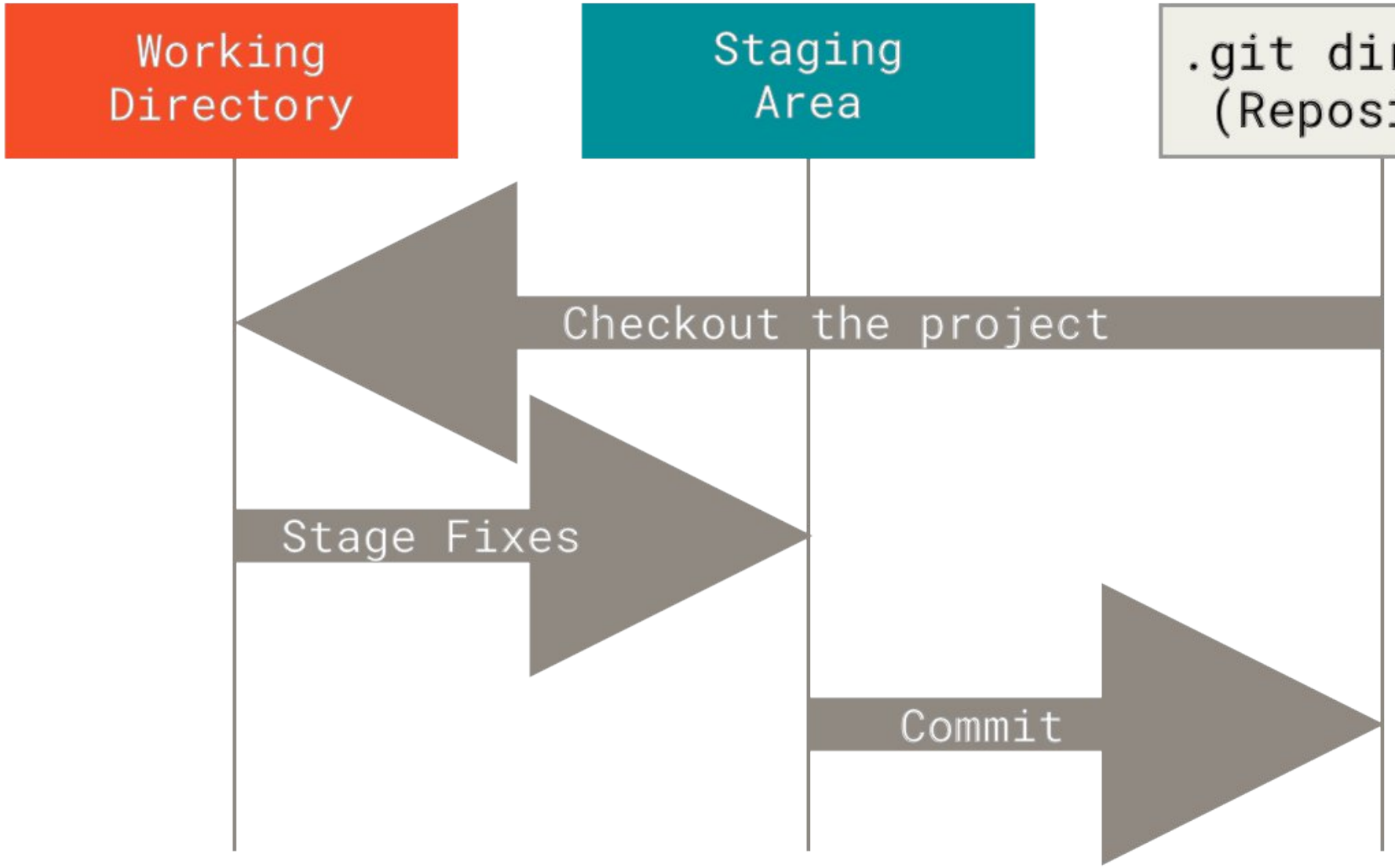


On your GitLab web-interface

On your notebook terminal

```
marchett@jns-marchett:~/Dl
[marchett@jns-marchett ~]$ mkdir DU-Datascience
[marchett@jns-marchett ~]$ cd DU-Datascience/
[marchett@jns-marchett DU-Datascience]$ git clone git@gitlab.in2p3.fr:ccin2p3-support/formations/du-
Cloning into 'test-project'...
Enter passphrase for key '                                                    ':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
Resolving deltas: 100% (1/1), done.
[marchett@jns-marchett DU-Datascience]$ cd test-project/
[marchett@jns-marchett test-project]$ ls
README.md
[marchett@jns-marchett test-project]$ cat README.md
Hello World!
[marchett@jns-marchett test-project]$ ls -a
.  ..  .git  README.md
[marchett@jns-marchett test-project]$ ls .git
branches  config  description  HEAD  hooks  index  info  logs  objects  packed-refs  refs
```

The Git workflow



- *While on your workstation*
- You modify files in your working tree.
- You selectively stage just those changes you want to be part of your next commit
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Working with Git - monitoring

```
$ git status
On branch develop
Your branch is up to date with 'origin/develop'.

nothing to commit, working tree clean
```

- Shows you the files on your branch in the repository, both tracked and untracked by Git.
- Answers the question:
  What's going on?

```
$ git diff
```

- Shows you changes between versions. Without arguments it shows all changes made on the tracked file since the last commit.
- With a **branch name** as argument, it shows the differences between the working branch and the argument.

```
$ git log
```

- Shows the commit history on your current branch.

```
$ git log -<N>
```

- With `-<N>` as argument, it shows the `<N>` last commits.

Suggestion

Create the hidden file `.gitignore` containing file patterns you want Git to ignore. Example:

```
cat << EOF >> .gitignore
*.log
*.tmp
test_data/
my_personal_notes.txt
EOF
```

These files won't show up in `git status` output.

documentation :
- https://git-scm.com/docs/git-status
- https://git-scm.com/docs/git-diff
- https://git-scm.com/docs/git-log

Working with Git – Setting your working branch

```
$ git branch
  branch_1
* branch_2
  branch_4
  ...
```

- Lists the branches available locally and identifies the branch you are on.

```
$ git branch <branch name>
```

- Creates locally a new branch but does not change your working branch

```
$ git branch -d <branch name>
```

- Deletes locally the target branch

documentation :

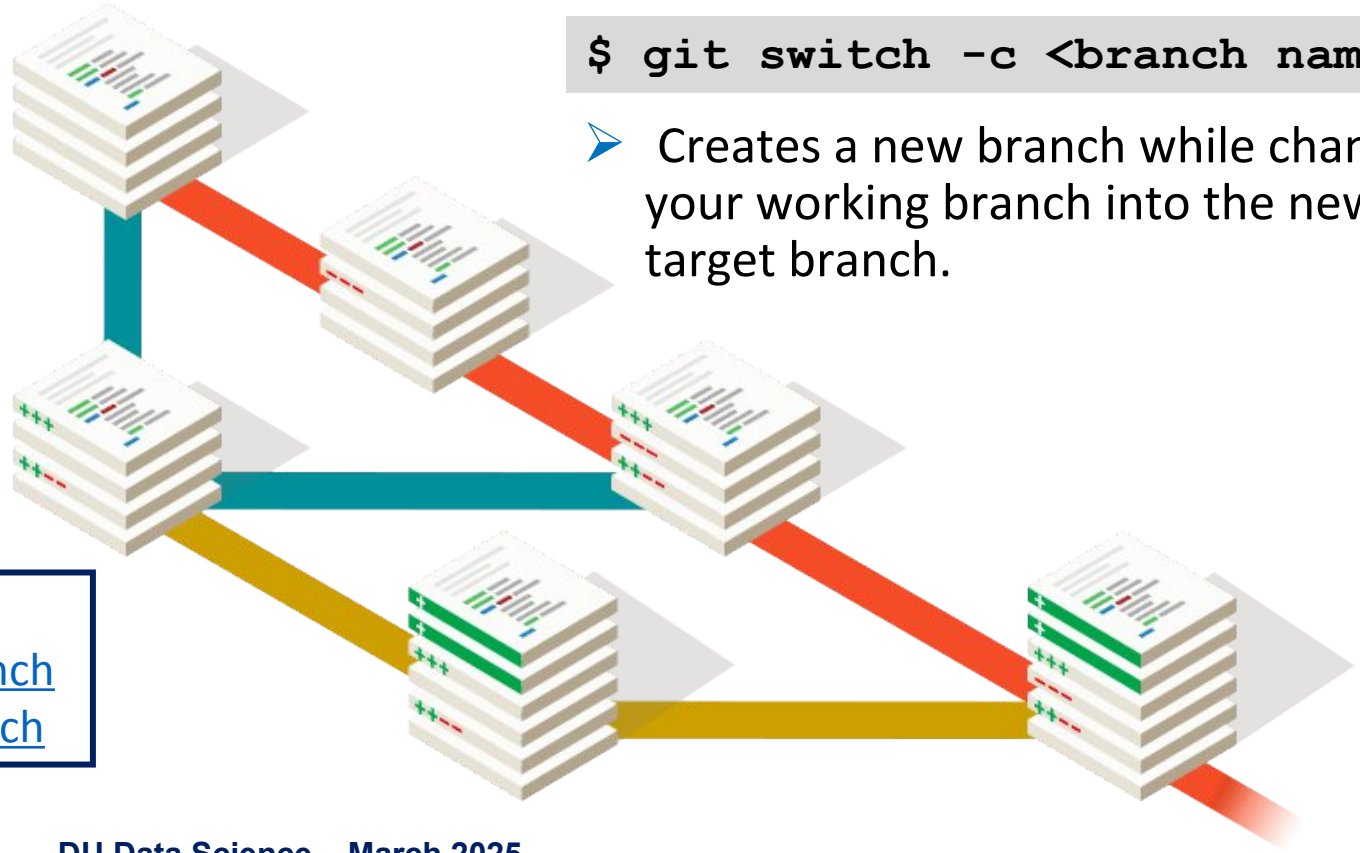
- https://git-scm.com/docs/git-branch
- https://git-scm.com/docs/git-switch

```
$ git switch <branch name>
```

- Changes your working branch into the target branch. The target branch must exist

```
$ git switch -c <branch name>
```

- Creates a new branch while changing your working branch into the new target branch.

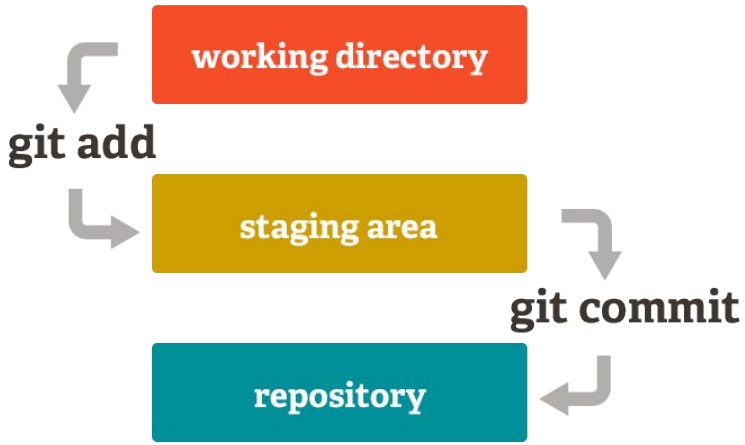


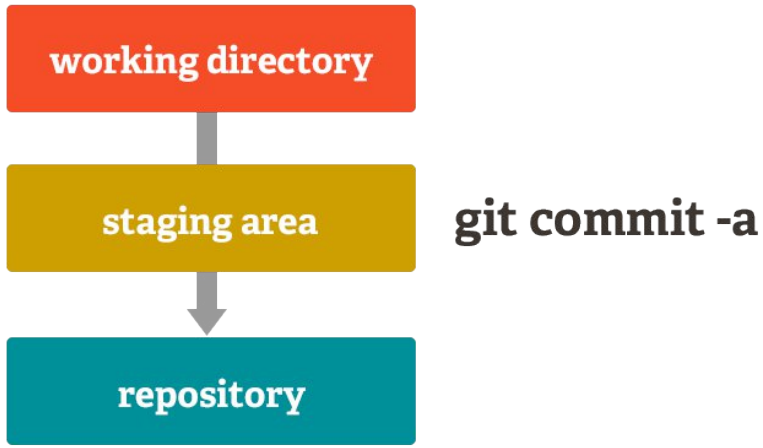
Working with Git – Making changes

```
$ git add <file>
```

- Adds the target file(s) in the staging area
- With the `--all` option all modified or newly created files are added to the staging area.

```
$ git add --all
```





git commit -a

```
$ git restore <file>
```

- Restores the file status to unmodified
- With the `--staged` option, restores stages file to “unstaged” status

```
$ git restore --staged <file>
```

documentation :
- https://git-scm.com/docs/git-add
- https://git-scm.com/docs/git-restore
- https://git-scm.com/docs/git-commit

```
$ git commit
```

```
$ git commit -a
```

```
$ git commit -m ".."
```

- Record chances to the repository. Without options the command will open a text editor to write a commit message
- With the `-a` option all modified files are recorded. Newly created files are ignored.
- With the `-m` option you may write a one-liner commit message

Git commit Messages

https://xkcd.com/1296/

The 7 rules!

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

https://cbea.ms/git-commit/

Working with Git – Collaborating 1/2

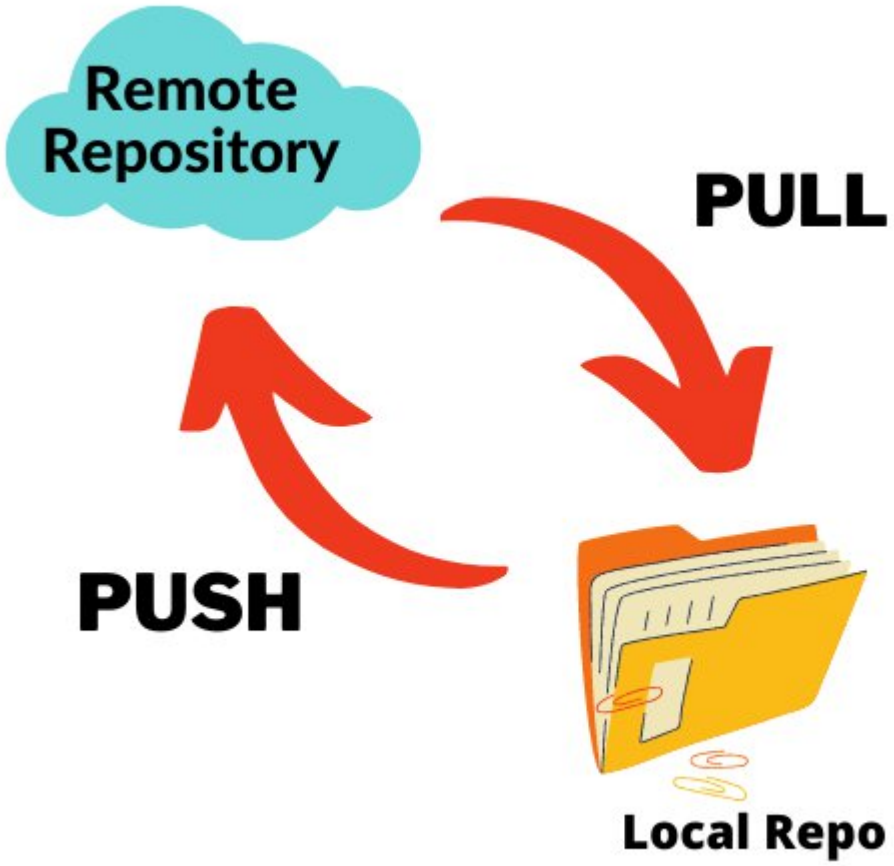
```
$ git merge <branch name>
Auto-merging <file>
CONFLICT (content): Merge conflict in <file>
Automatic merge failed; fix conflicts and then commit the result
```

- Incorporate the changes on the target branch into the current branch
- ~~When~~ If a conflict arises it is notified in the output
  - Open in a text editor the involved file(s) and check for the pattern shown **right**
  - Choose the most pertinent version (or write a new one) and save the file. Then:
  - `git add <file>`
  - `git commit –m "conflict fixed"`
  - `git push`

```
Here are lines that are either unchanged from the
common ancestor, or cleanly resolved because only
one side changed, or cleanly resolved because both
sides changed the same way.
<<<<<<< HEAD
Conflict resolution is hard; let's go shopping.
=======
Git makes conflict resolution easy.
>>>>>>> branch-to-be-merged
And here is another line that is cleanly resolved
or unmodified.
```

Working with Git – Collaborating 2/2

`$ git pull` ➢ Fetch and integrate with the remote repository.



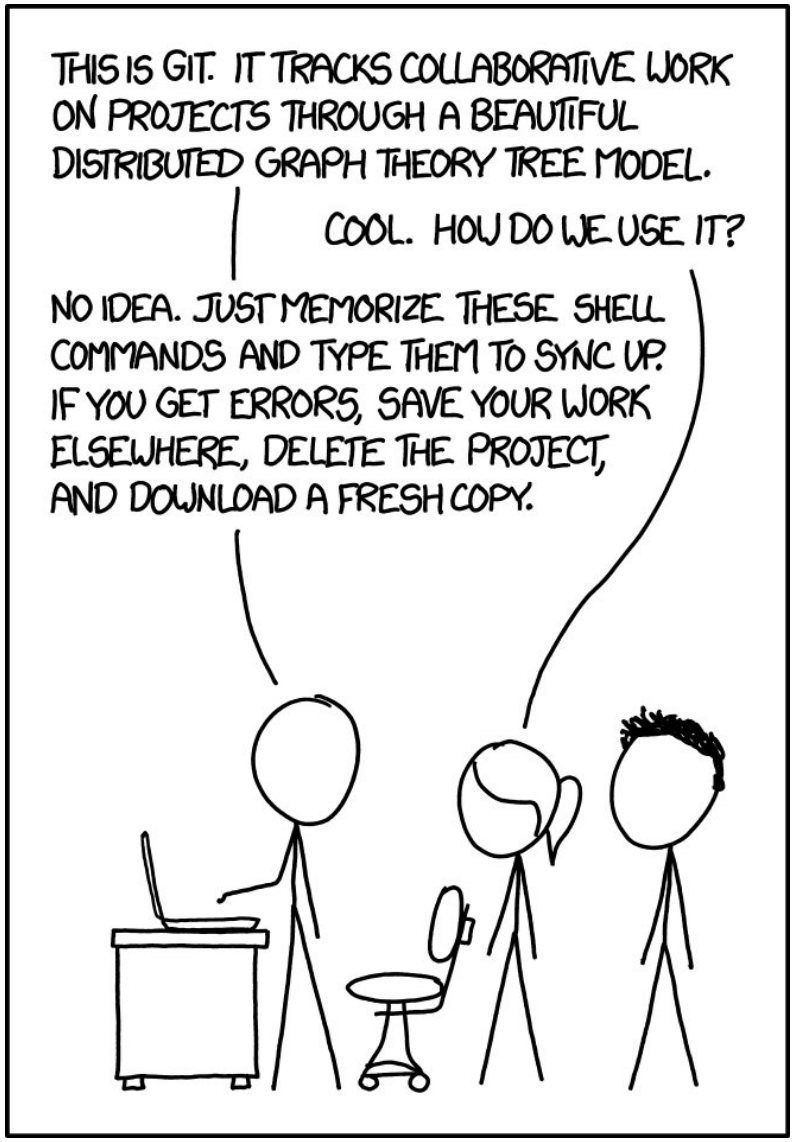
`$ git push` ➢ Update the remote repository.

First push

```
$ git branch <branch name>
...modifications...
$ git commit -am "first commit"
$ git push --set-upstream origin <branch name>
```

➢ On your fist push from a newly created branch you will need to create the branch on the remote repository.

To summarize...



https://xkcd.com/1597/

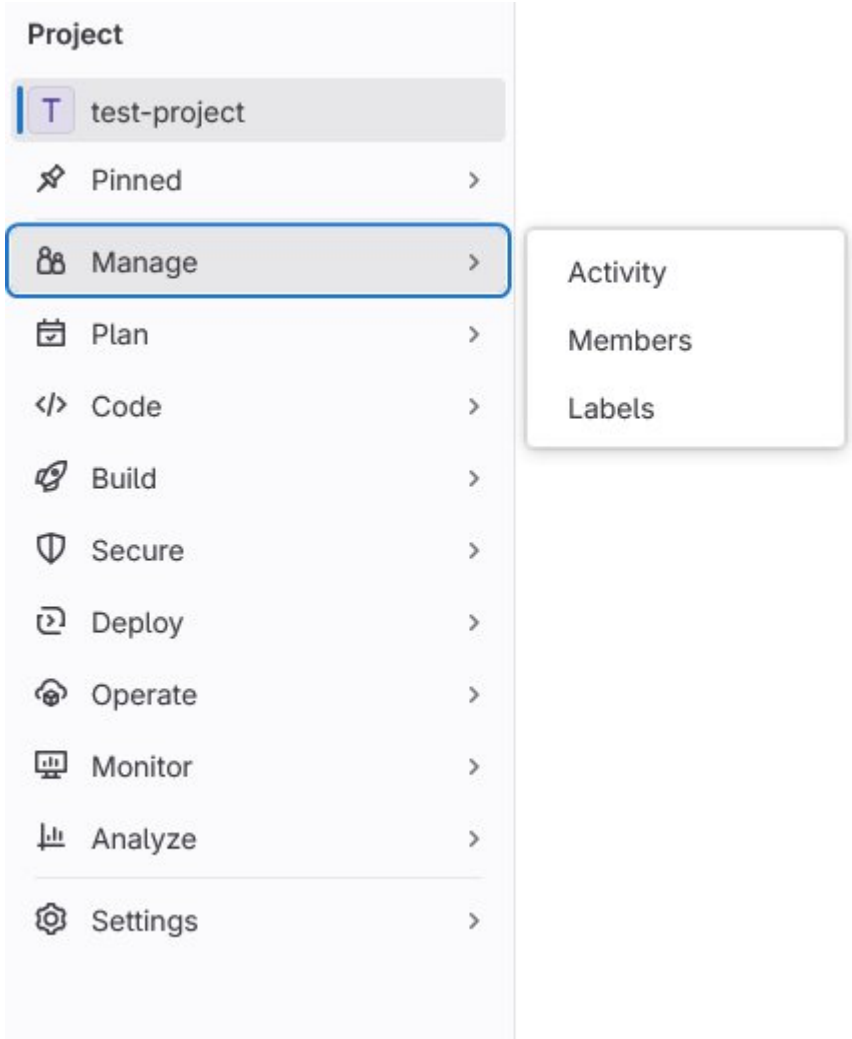
➢ We ALL did that!

- ➢ At least once...
- ➢ ...and we are not much proud about it

➢ YOU need to do better

- ➢ This course allows you to get started and work on a collaborative project
  - ➢ the famous “shell commands”
- ➢ Git is solid and “foolproof” for a fair amount of foolishness
- ➢ Practice, put yourself in a difficult position, ask questions, solve your issue and learn using Git

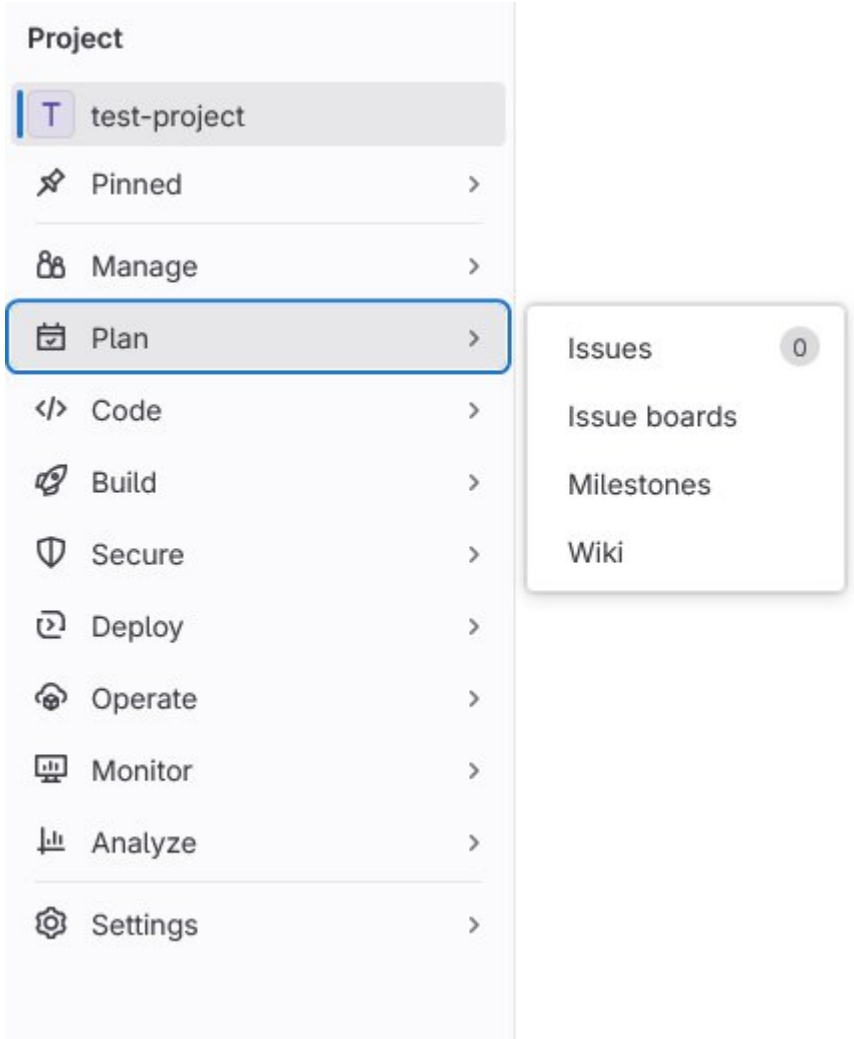
GitLab environment overview 1/5



➢ Project Management

- ➢ Activity
  - ➢ Branches, Commits
- ➢ Members
  - ➢ Invite, grant access rights
- ➢ Labels for Issues, Milestones, MR...
  - ➢ Create, Rename

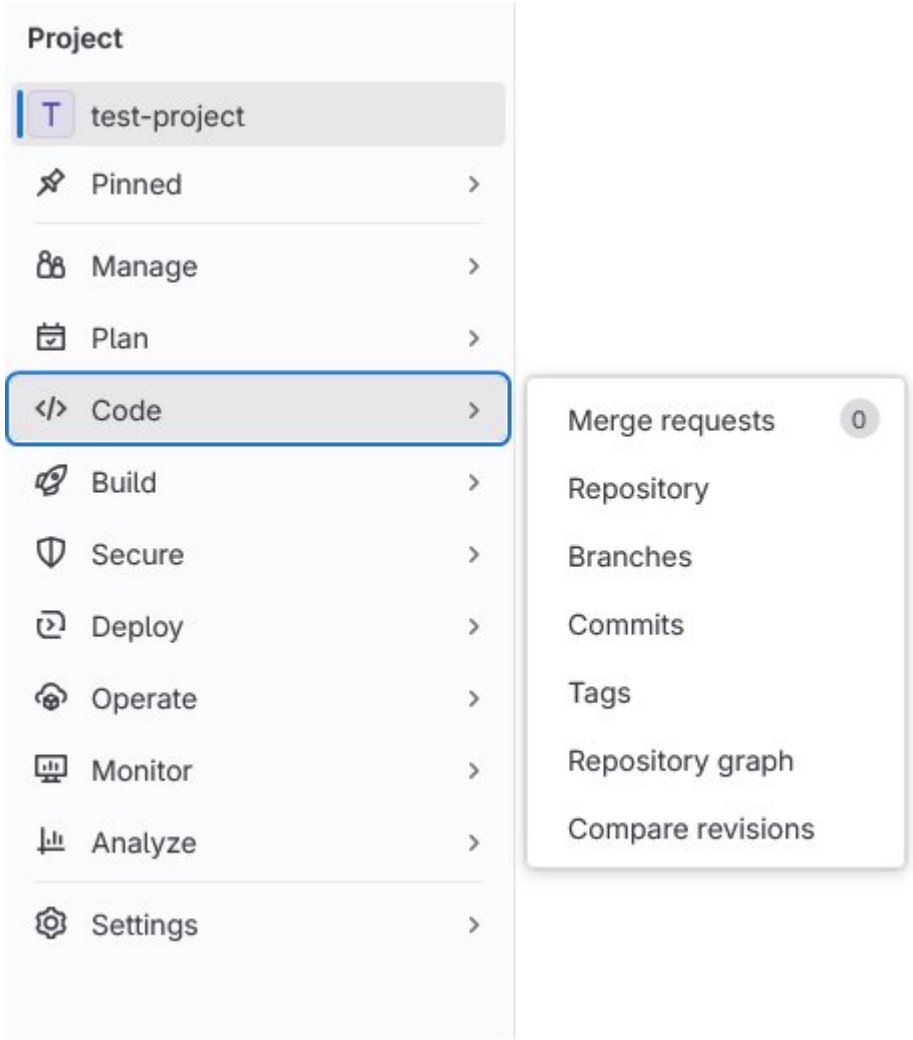
GitLab environment overview 2/5



➢ Organize collaboration

- ➢ Issues, Issue boards
  - ➢ A ticketing system for the project
  - ➢ Labels may be added to Issues
  - ➢ Members may be assigned to issues
- ➢ Milestones
  - ➢ Organize development progress setting deadlines
- ➢ Wiki
  - ➢ Project documentation space internal to GitLab

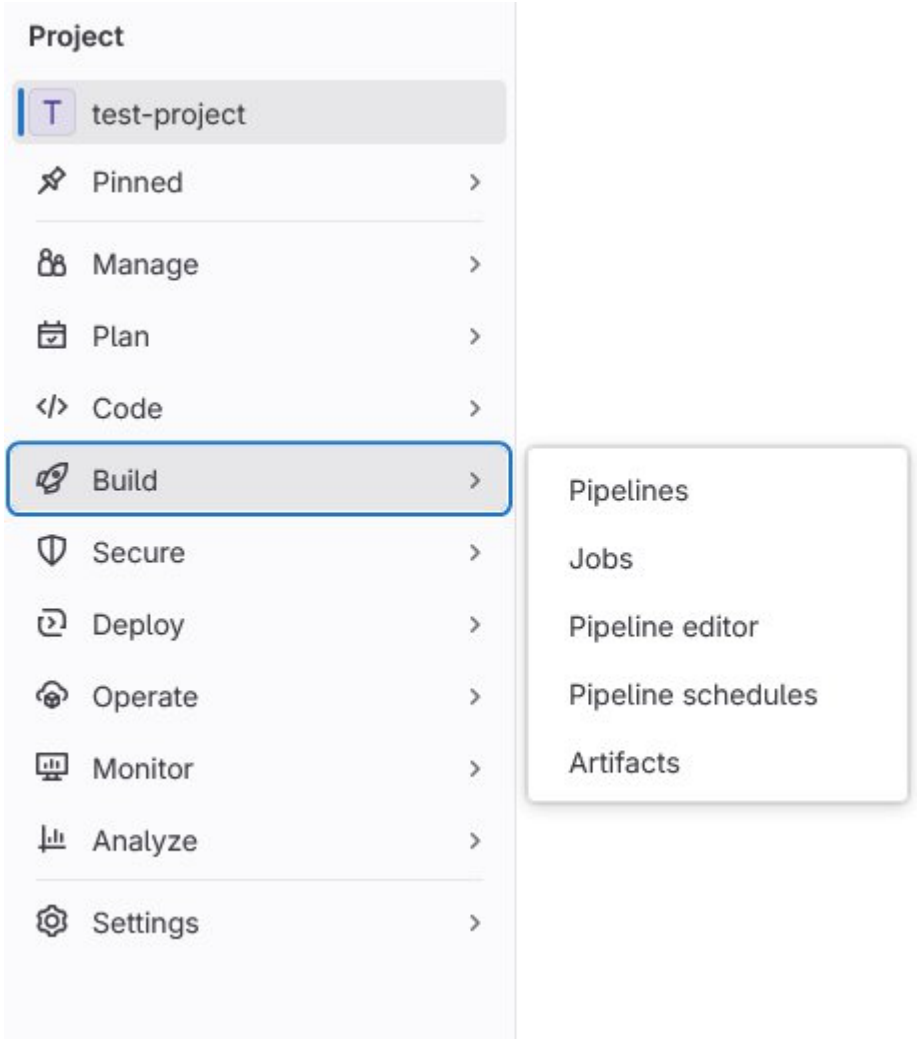
GitLab environment overview 3/5



➢ Day-to-day work

- ➢ Repository
  - ➢ Project files and directories
- ➢ Tags
  - ➢ Flag a commit (to prepare a release or highlight an important development phase)
- ➢ Repository graph
  - ➢ Project commit tree (check which branch is the most “forward”)
- ➢ Compare revision
  - ➢ Graphical version of `git diff`

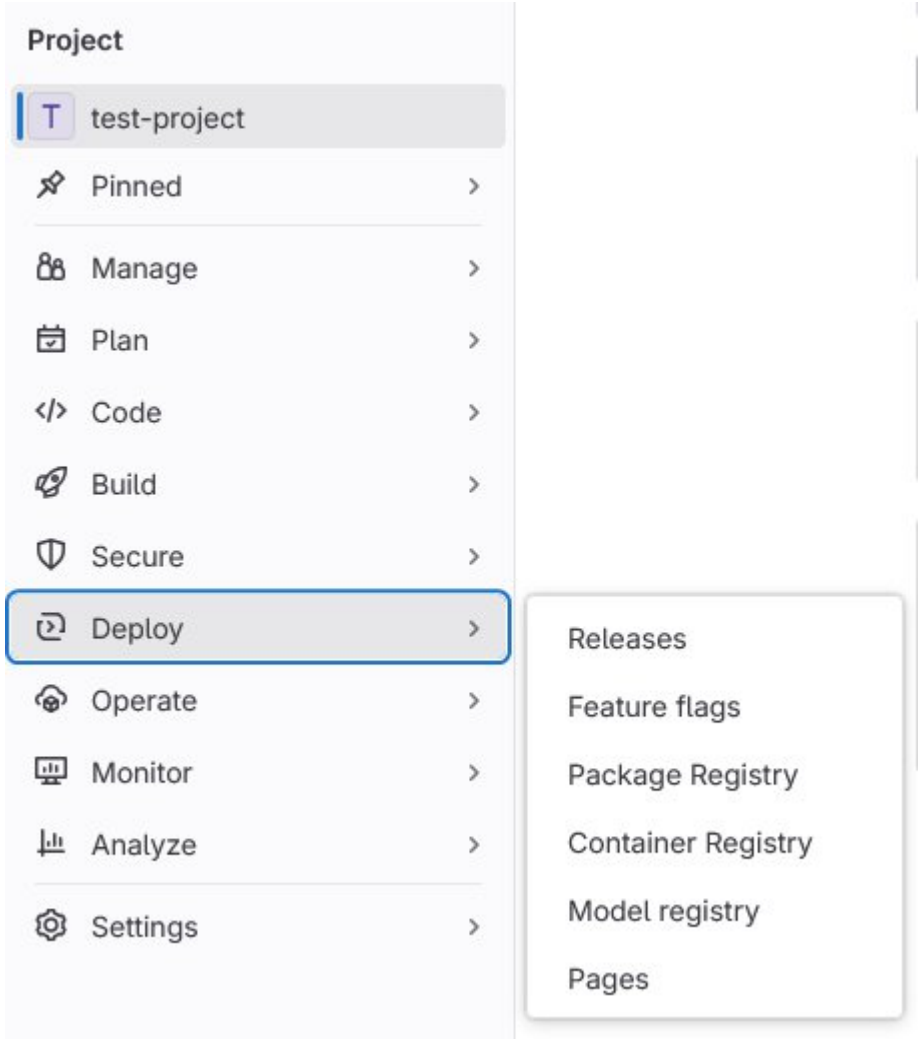
GitLab environment overview 4/5



➢ CI/CD

- ➢ Pipelines, Jobs
  - ➢ Check the CI task status and workflow
  - ➢ P. editor and schedules
- ➢ Artifacts
  - ➢ Output files of CI tasks

GitLab environment overview 5/5



- **Deploy**
  - Releases
    - Write release notes
    - Associate Milestone and Tags
  - Pages
    - Allows to publish a project website

Enjoy collaborative development!