HERD SW WORKSHOP

INTRO TO GITLAB

BEFORE WE START...

It is common practice for a piece of software to be under a Version Control System, to allow different people to efficiently collaborate on the same codebase together.

The VCS of choice for HerdSoftware is *git* (https://git-scm.com) and the whole codebase is hosted on a *gitlab* instance at RECAS.

What this is not:

- A tutorial on how to use *git*.
  We assume all of you have at least a basic working knowledge on how to work with *git*. In case you need a cheatsheet here are a couple of useful links:
  - https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html
  - https://docs.gitlab.com/ee/gitlab-basics/command-line-commands.html

What this is:

- An overview of what *gitlab* offers on top of *git* to facilitate and improve the experience of working together on a big software project

WHAT IS GITLAB?

Gitlab is a web-application designed to improve and enhance collaborative development on a piece of software.

- It allows people to interact with one or more *git* repositories by using a common and friendly web interface
- It extends the functionality of the repository with dedicated tools (that actually make life a lot easier…)
- It is currently used and deployed by:
  - CERN (gitlab.cern.ch)
  - INFN (baltig.infn.it)
  - RECAS (git.recas.ba.infn.it)

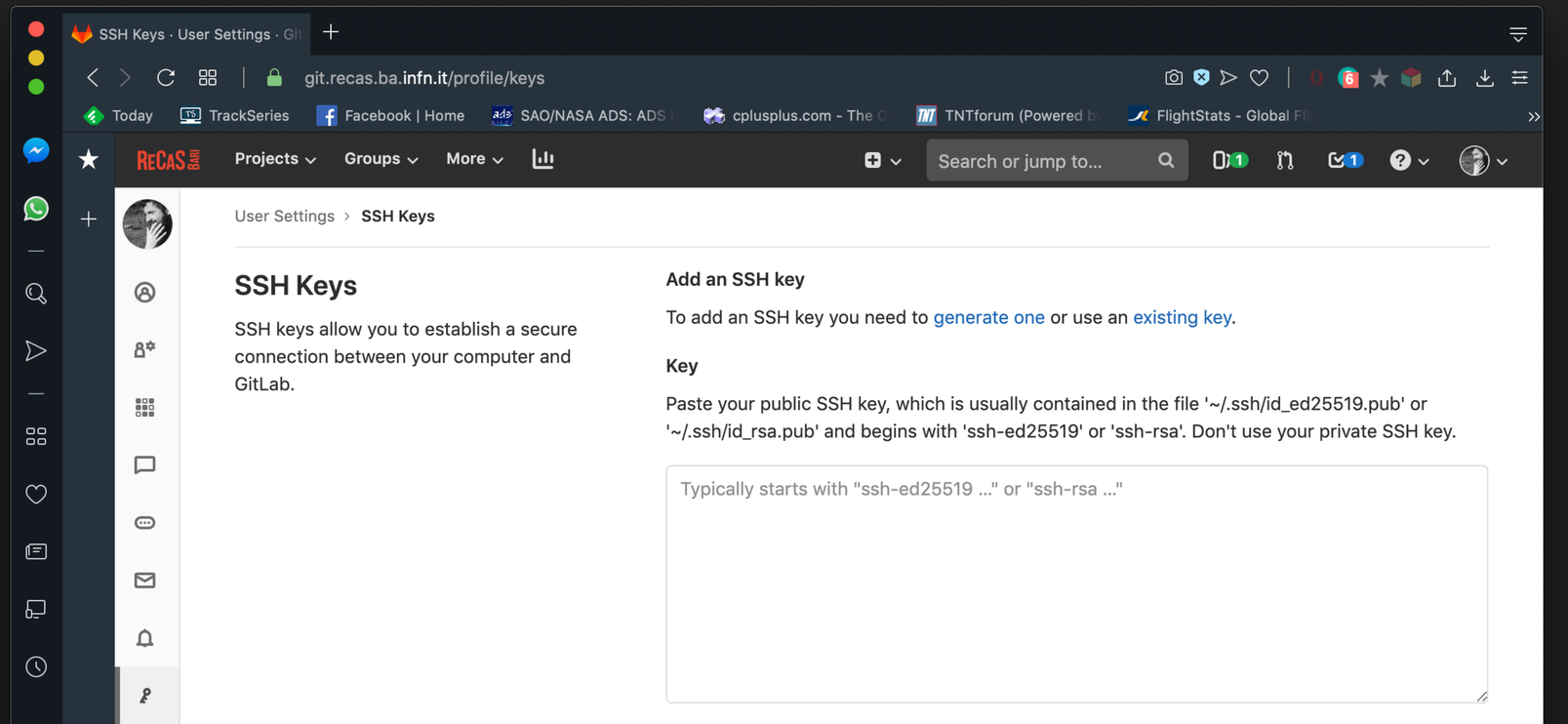
There are alternative projects that offer similar functionalities (*github*, *BitBucket*, etc…) but so far *gitlab* seems to be the most complete all-in-one solution.

Let's take a look at our instance… git.recas.ba.infn.it/herd

FIRST STEPS

My personal suggestion as a first step:

- Add your public SSH key to your *gitlab* profile. This will allow you to operate on the *git* repo from the command line without the need of constant authentication.
  (if you don't have a *gitlab* account then obviously the first step is to create one)



EXAMPLE: ADD A NEW SSH KEY

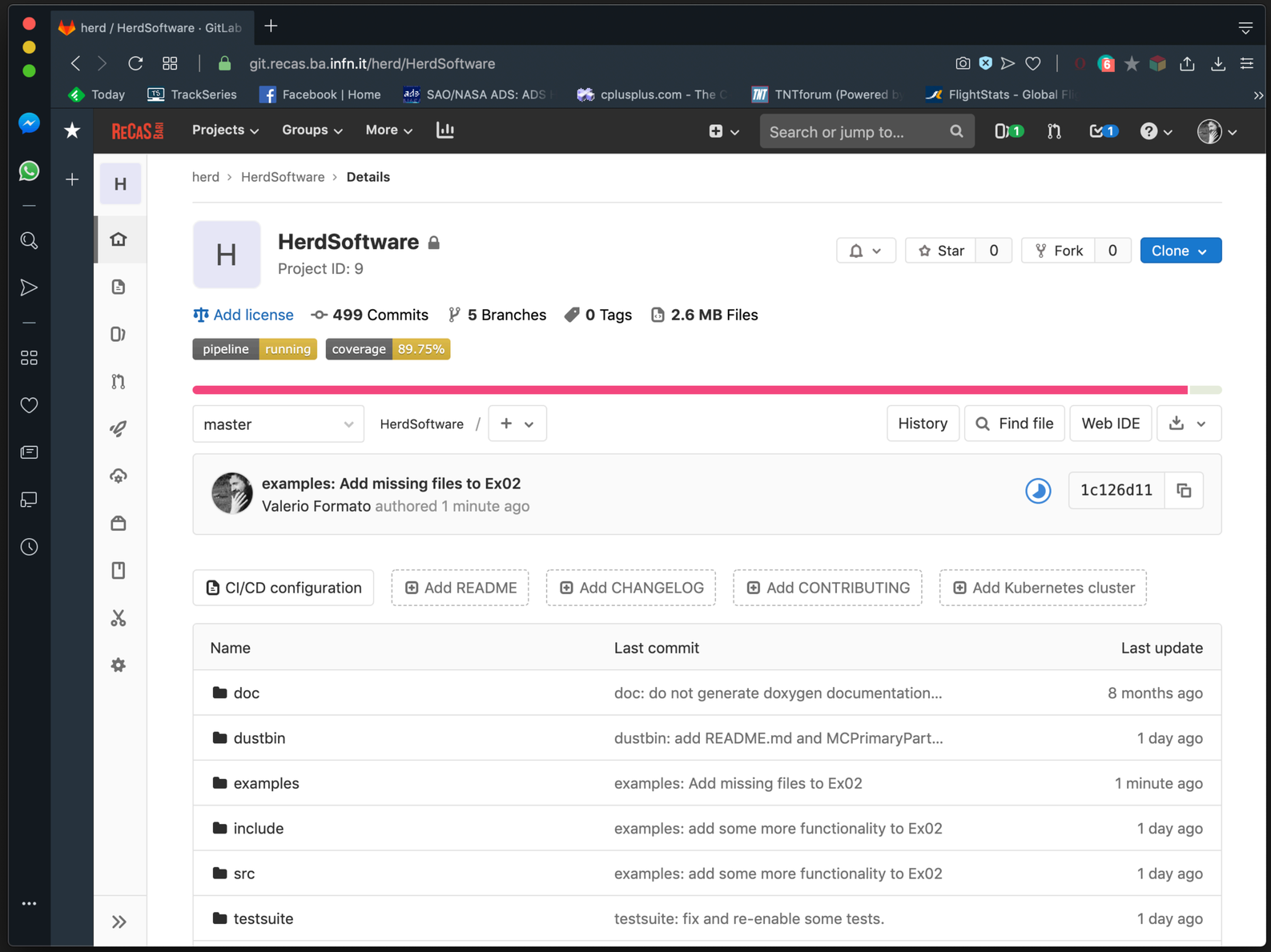
GITLAB STRUCTURE

Every *git* repository in *gitlab* is associated to a **project** (https://docs.gitlab.com/ee/user/project/).

Projects encompass a *git* repo and offer additional tools around it:

- Issue tracker
- Merge requests
- Continuous integration
- Wiki documentation
- ...and many more

We'll touch briefly some of these tools, just for general knowledge, but in principle you won't have to deal directly with most of them.

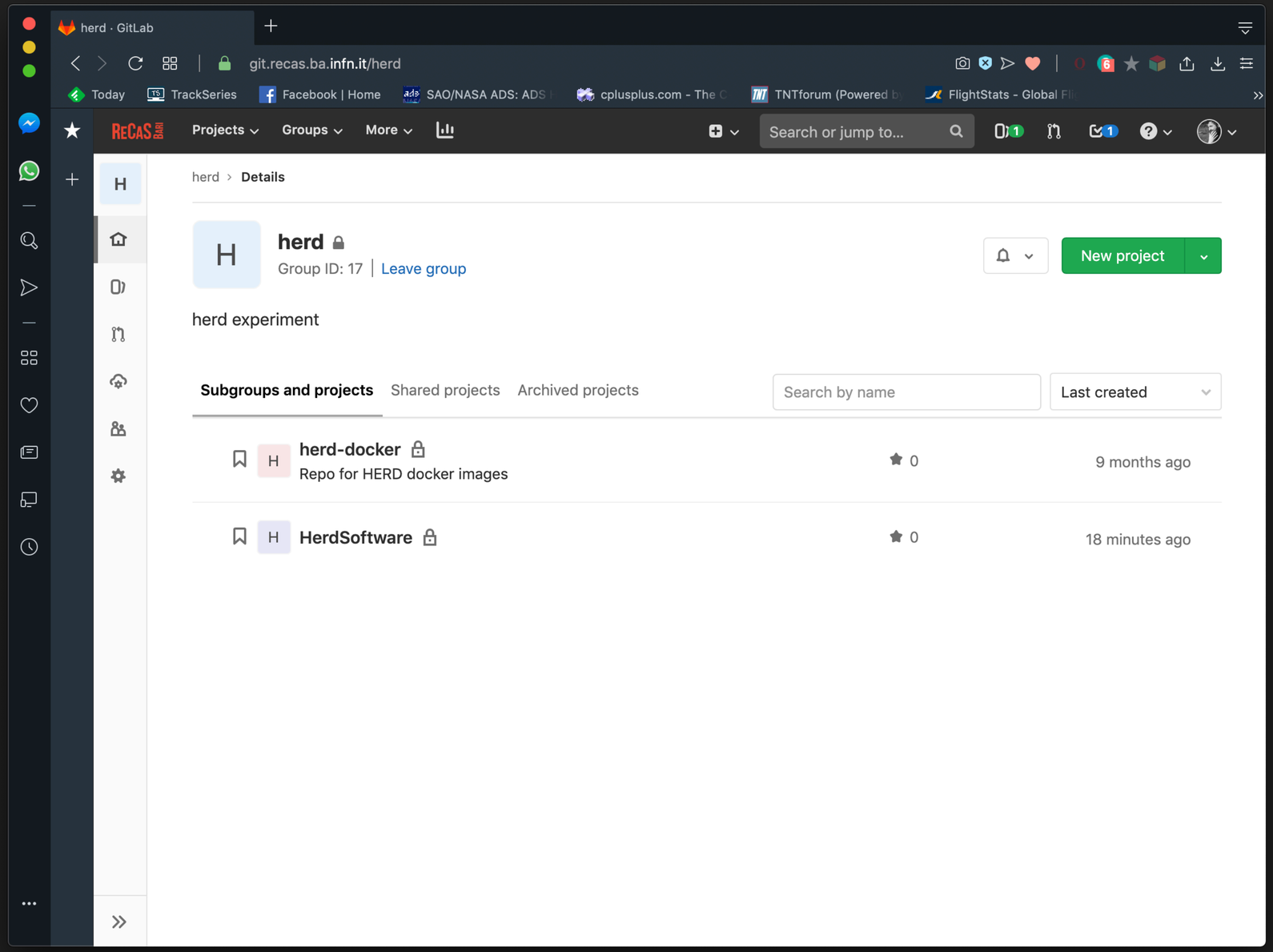


GITLAB STRUCTURE

Several **projects** can be put together in the same **group** (https://docs.gitlab.com/ee/user/group/).

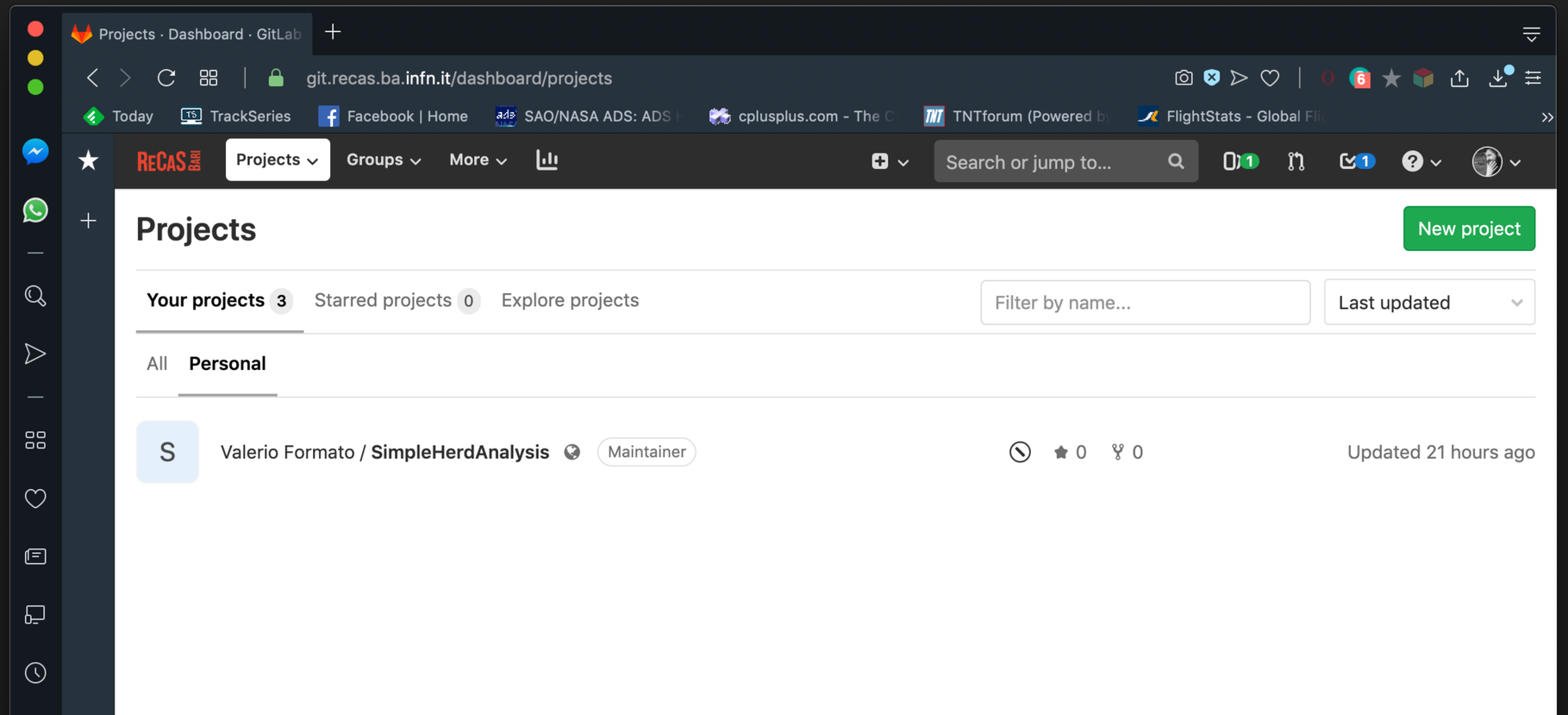
Groups are used for several different reasons, among which:

- Grant access to multiple projects and multiple team members in fewer steps by organizing related projects under the same namespace and adding members to the top-level group.
- Make it easier to mention all of your team at once in issues and merge requests by creating a group and including the appropriate members.



GITLAB STRUCTURE

You can also create **personal projects**, outside of any group scope. This is particularly useful if you want to work on your own personal analysis code and you don't need to share it with the whole group: just create a **project**!



EXAMPLE: CREATE A NEW PROJECT

EXAMPLE: CLONING HERDSOFTWARE

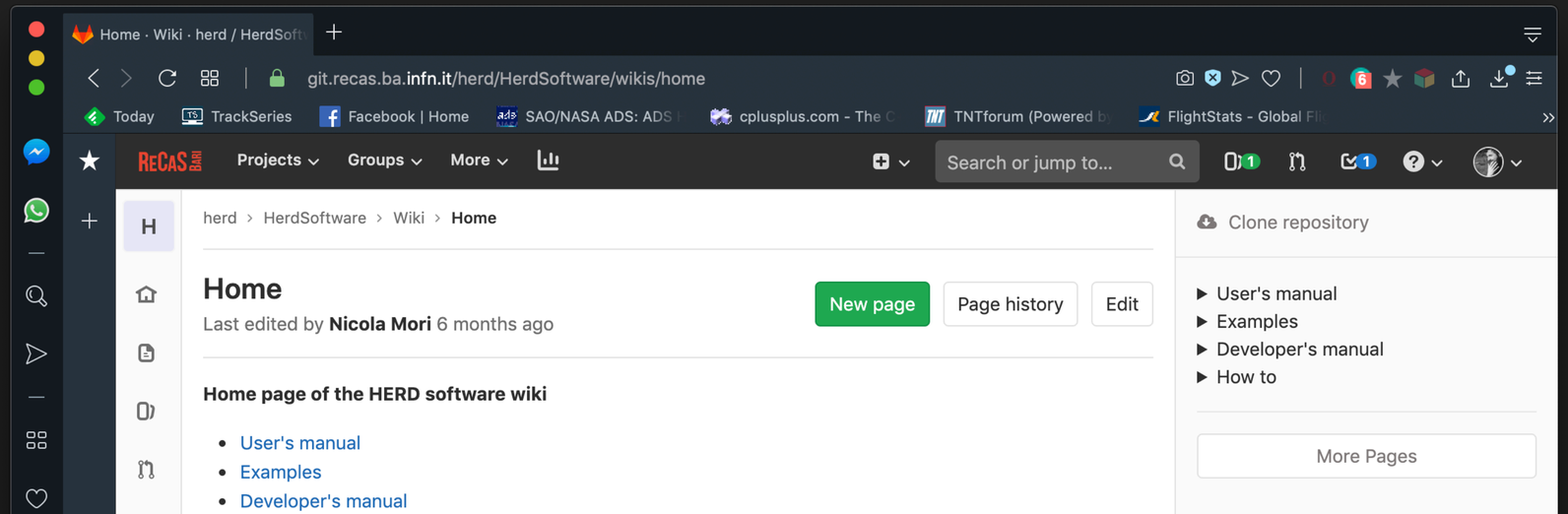
WIKI

A separate system for documentation called **Wiki**, is built right into each GitLab project. It is enabled by default on all new projects and you can find it under **Wiki** in your project.

Wikis are very convenient if you don't want to keep your documentation in your repository, but you do want to keep it in the same project where your code resides.

You can create **Wiki** pages in the web interface or locally using *git* since every **Wiki** is a separate *git* repository.

HerdSoftware comes with its own **Wiki** documentation that we try to keep always updated. You are encouraged to consult it whenever you have doubts about the software. If you think that some section is not described in enough detail, don't hesitate to write a mail or open an issue (see later...)



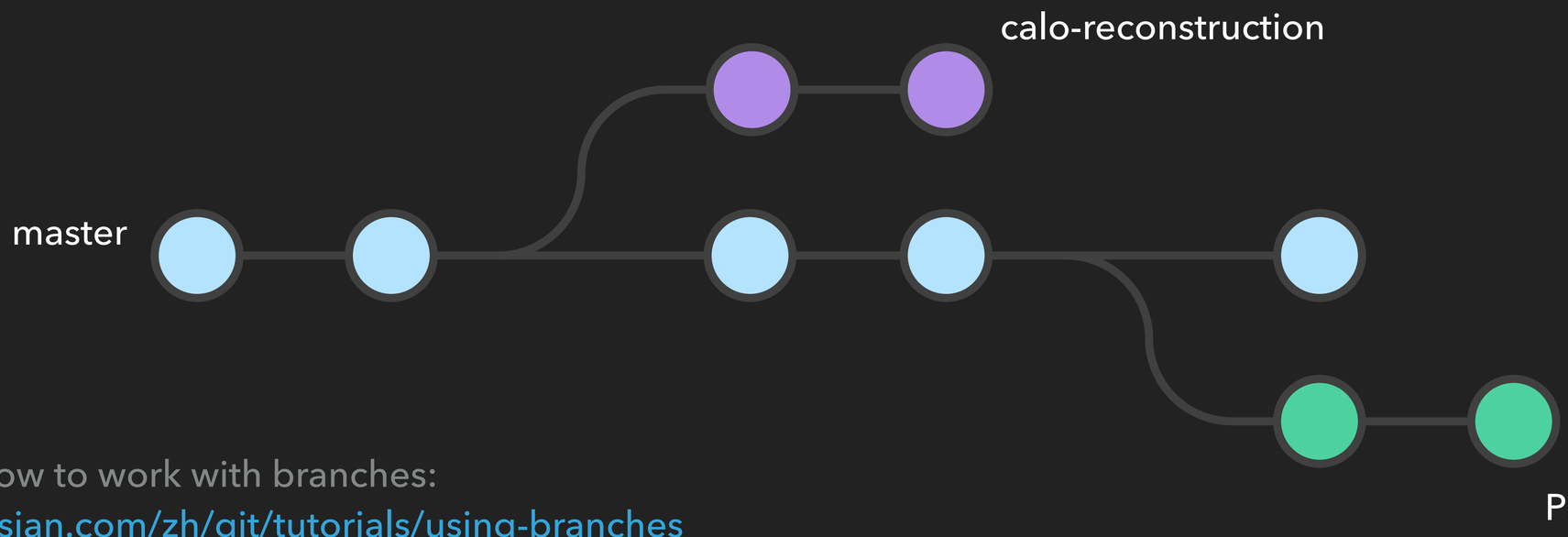
EXAMPLE: TAKING A LOOK AT THE WIKI

WORKING WITH BRANCHES

We expect that at the beginning the work will consist of both data analysis from the simulated files (which can be done in personal projects), as well as improvement and development of new reconstruction code in HerdSoftware.

In HerdSoftware the default branch is the *master* branch.

In order to allow everyone to work freely and keep the code in a nice clean and working state it is preferable to develop new features or fixes on dedicated branches and merge them into *master* only when the work is done.



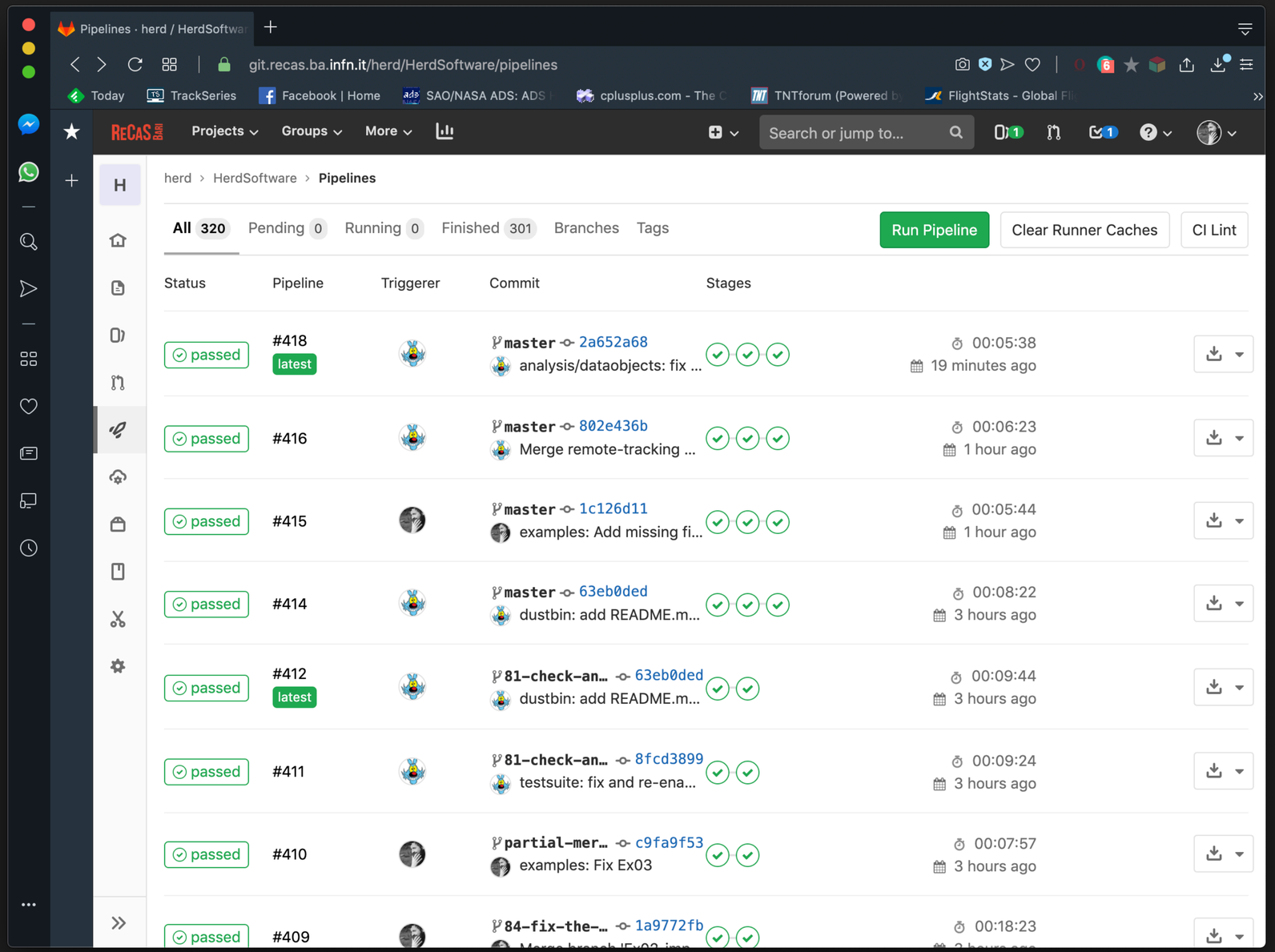
Quick guide on how to work with branches:
https://www.atlassian.com/zh/git/tutorials/using-branches

CONTINUOUS INTEGRATION

Continuous Integration (CI - https://about.gitlab.com/product/continuous-integration/) is the practice of integrating code into a shared repository and building/testing each change automatically, as early as possible - usually several times a day.

Practically this translates in a set of checks and tests that will be automatically performed at each push and will alert the developer if any of these checks fails.

In HerdSoftware both the build process and the unit-tests are checked using the CI, over 4 possible environments (centos7, ubuntu18, ubuntu16 and OSX High Sierra)

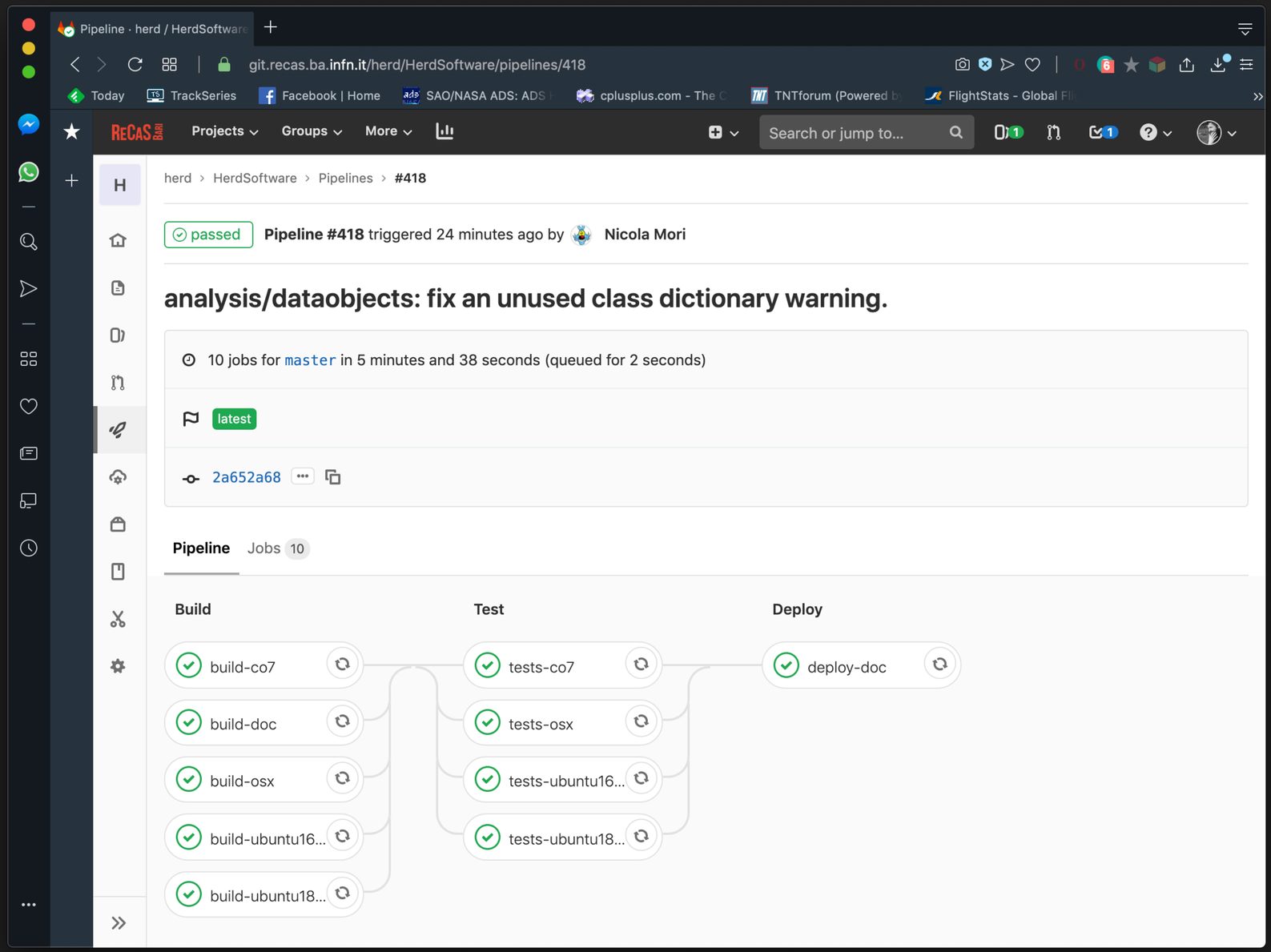


CONTINUOUS INTEGRATION

Continuous Integration (CI - https://about.gitlab.com/product/continuous-integration/) is the practice of integrating code into a shared repository and building/testing each change automatically, as early as possible - usually several times a day.

Practically this translates in a set of checks and tests that will be automatically performed at each push and will alert the developer if any of these checks fails.

In HerdSoftware both the build process and the unit-tests are checked using the CI, over 4 possible environments (centos7, ubuntu18, ubuntu16 and OSX High Sierra)



EXAMPLE: PUSH A NEW COMMIT AND TRIGGER THE PIPELINE

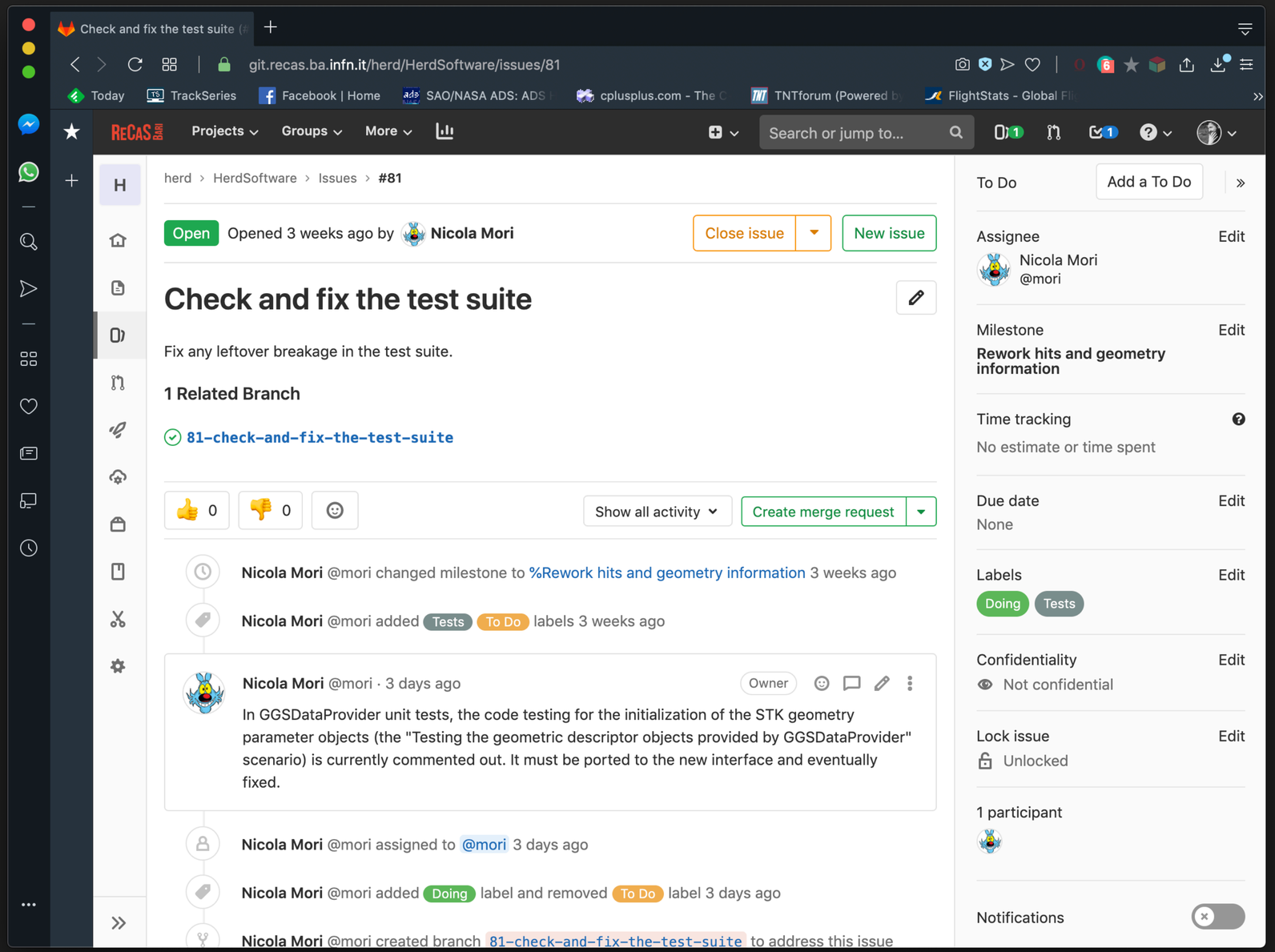
ADVANCED TOOLS: ISSUES

Issues are the fundamental medium for collaborating on ideas and planning work in GitLab.

In HerdSoftware they are used for:

- Report a bug
- Propose a new feature or work item
- Organise and split the work on big tasks between different developers

Each issue comes with a title and a description. Comments can be added as the work progresses. Gitlab can automatically create a dedicated branch (and a merge request) for a given issue. Issues can be labelled with custom with tags.

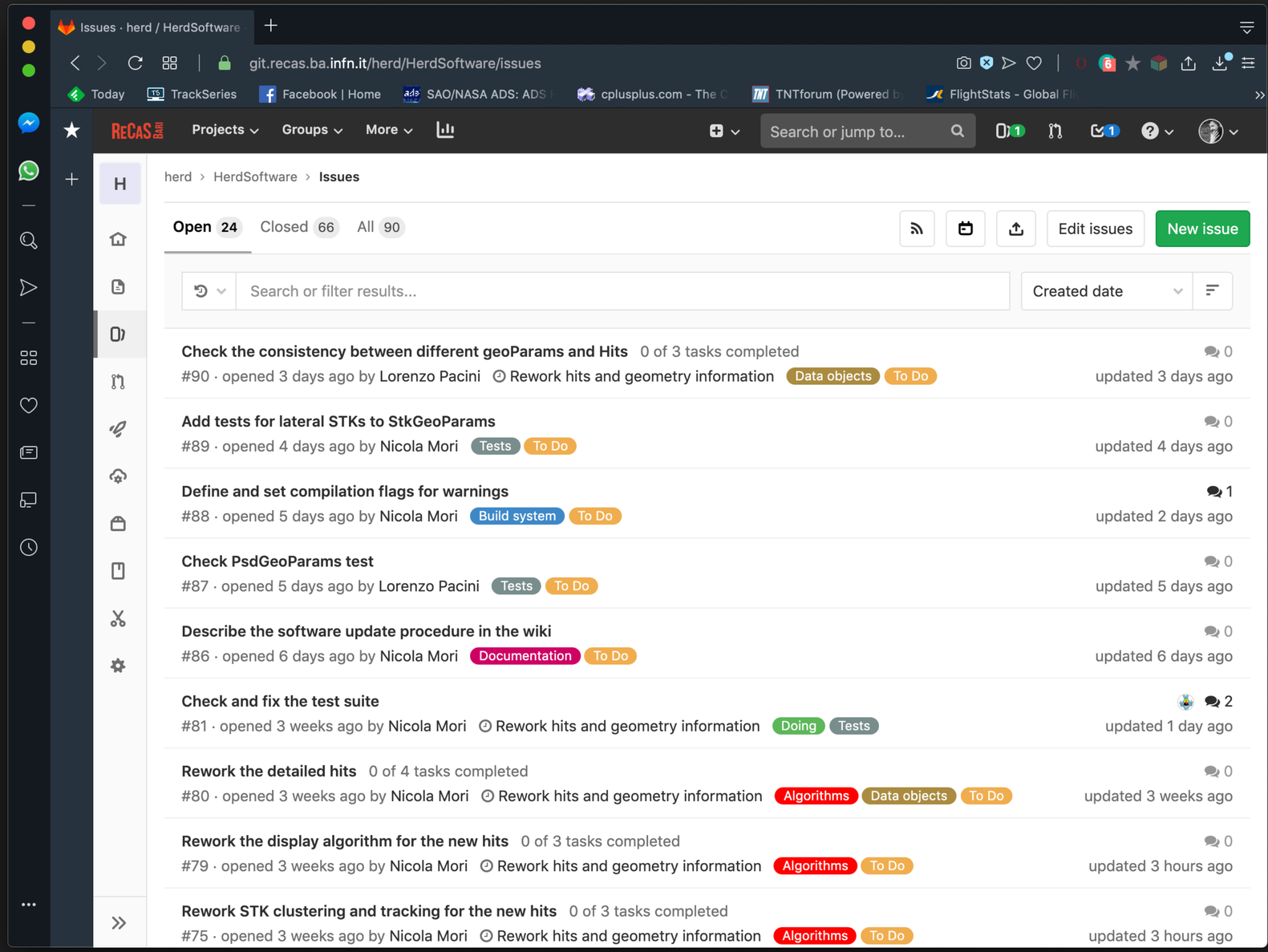


ADVANCED TOOLS: ISSUES

In HerdSoftware they are used for:

- Report a bug
- Propose a new feature or work item
- Organise and split the work on big tasks between different developers

You can check/add new issues from the Issue list page.

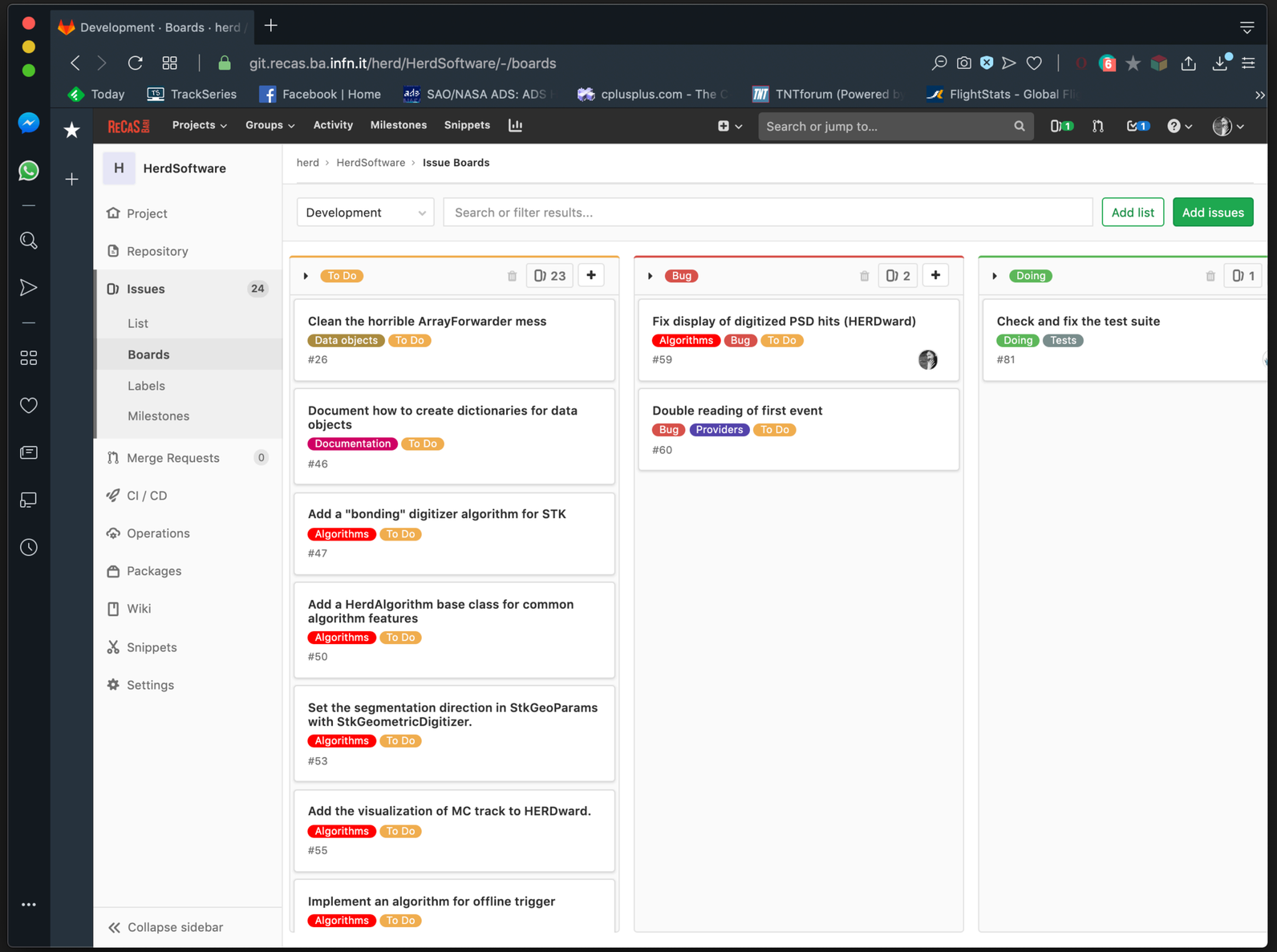


ADVANCED TOOLS: ISSUES

In HerdSoftware they are used for:

- Report a bug
- Propose a new feature or work item
- Organise and split the work on big tasks between different developers

You can check/add new issues from the Issue list page.
Or from the Issue board.

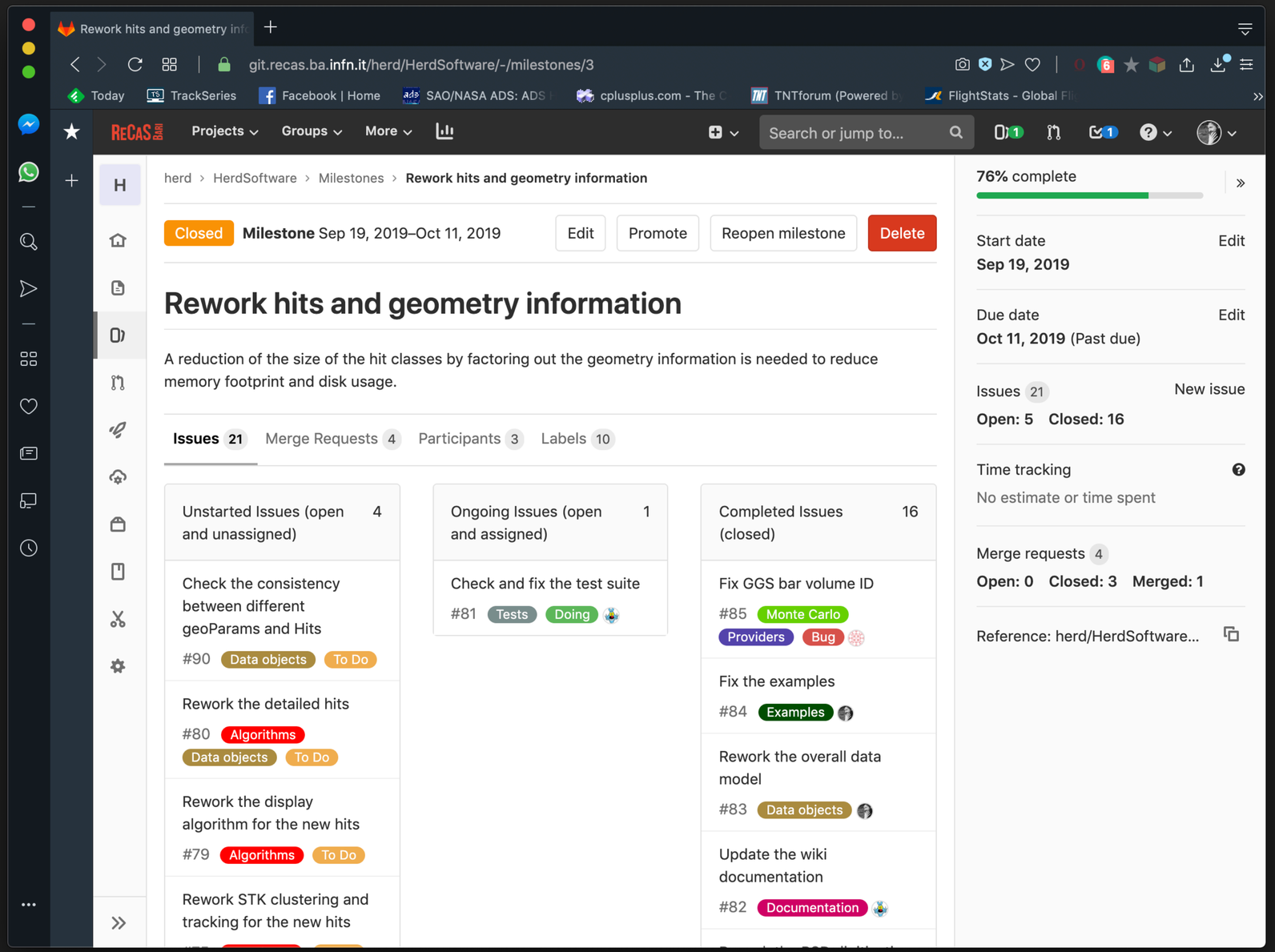


ADVANCED TOOLS: ISSUES

In HerdSoftware they are used for:

- Report a bug
- Propose a new feature or work item
- Organise and split the work on big tasks between different developers

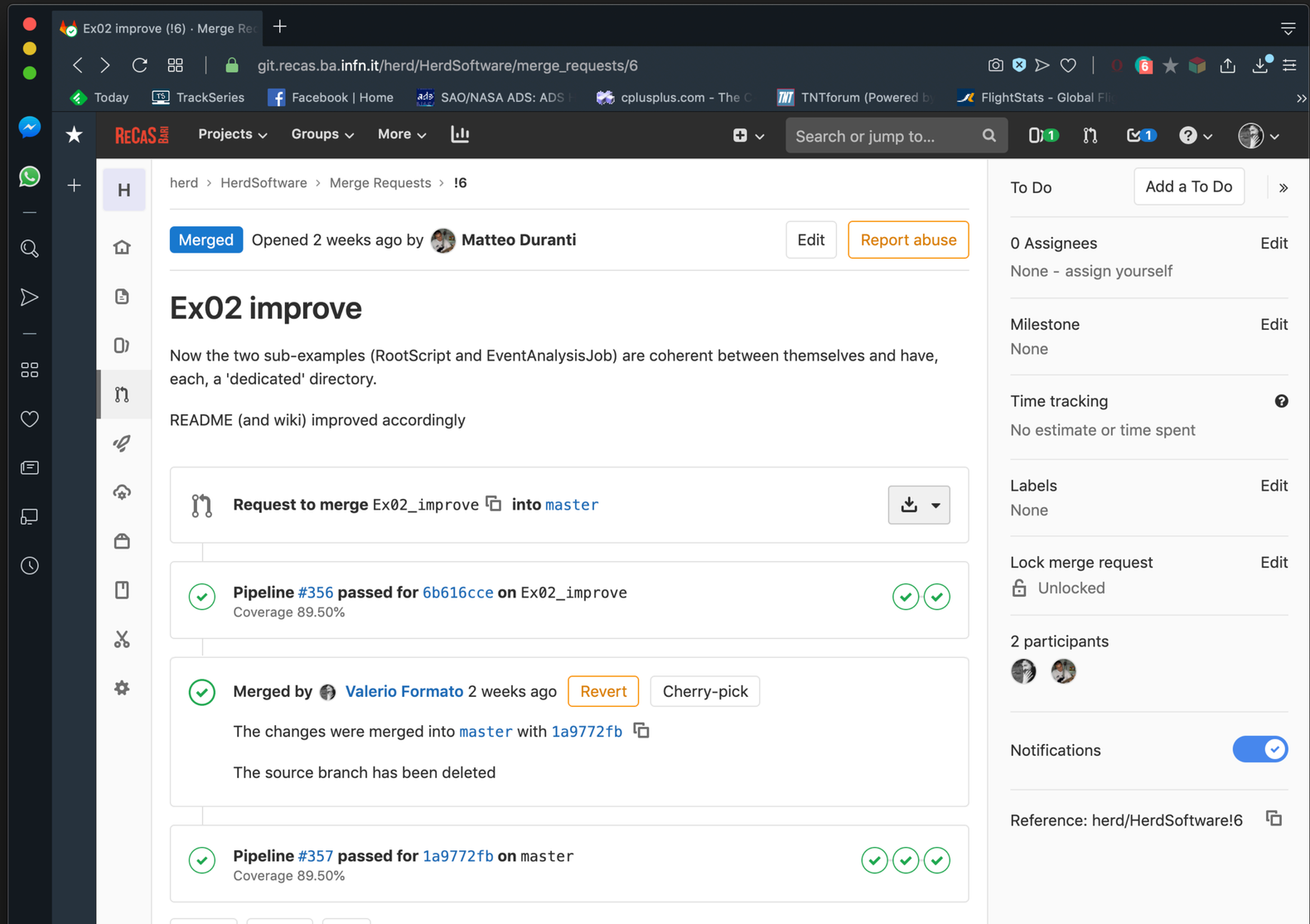
Issues can be grouped into Milestones, a collection of Issues all related to the same unit of work.



ADVANCED TOOLS: MERGE REQUESTS

A Merge Request (MR) is the basis of GitLab as a code collaboration and version control platform. It is as simple as the name implies: a request to merge one branch into another.

- Compare the changes between two branches
- Review and discuss the proposed modifications inline
- Build, test, and deploy your code in a per-branch basis with built-in GitLab CI/CD
- Automatically close the issue(s) that originated the implementation proposed in the merge request
- Organize your issues and merge requests consistently throughout the project with labels
- Resolve merge conflicts from the UI
- Enable fast-forward merge requests



ADVANCED TOOLS: MERGE REQUESTS

A Merge Request (MR) is the basis of GitLab as a code collaboration and version control platform. It is as simple as the name implies: a request to merge one branch into another.

- Compare the changes between two branches
- Review and discuss the proposed modifications inline
- Build, test, and deploy your code in a per-branch basis with built-in GitLab CI/CD
- Automatically close the issue(s) that originated the implementation proposed in the merge request
- Organize your issues and merge requests consistently throughout the project with labels
- Resolve merge conflicts from the UI
- Enable fast-forward merge requests