LIÈGE université
GIGA
CRC In vivo Imaging

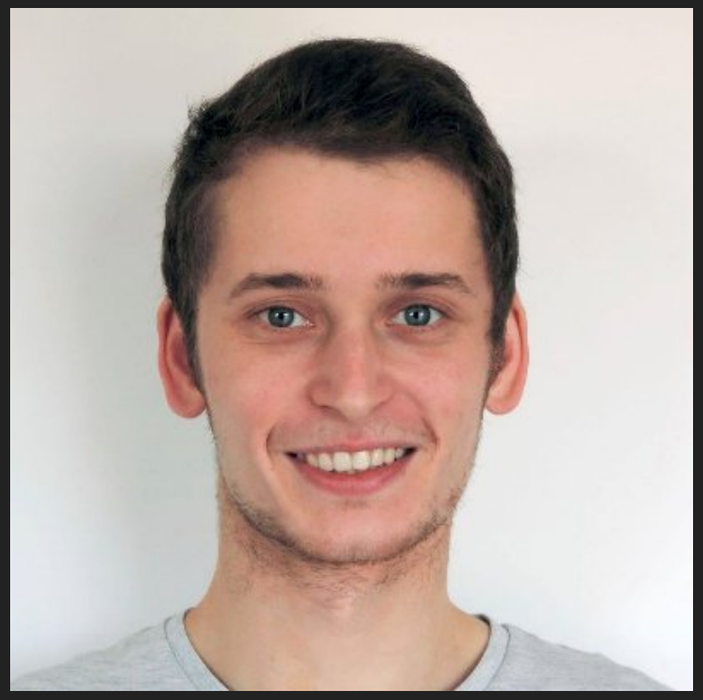
GIGA DOCTORAL SCHOOL 2022

GIT IN PRACTICE

HANDS-ON TUTORIAL

WHO ARE WE?

LIÈGE université
GIGA
CRC In vivo Imaging



- Phd student at the GIGA - CRC in vivo imaging
  - Freshly graduated!
  - “Electromagnetic modelling of a head”

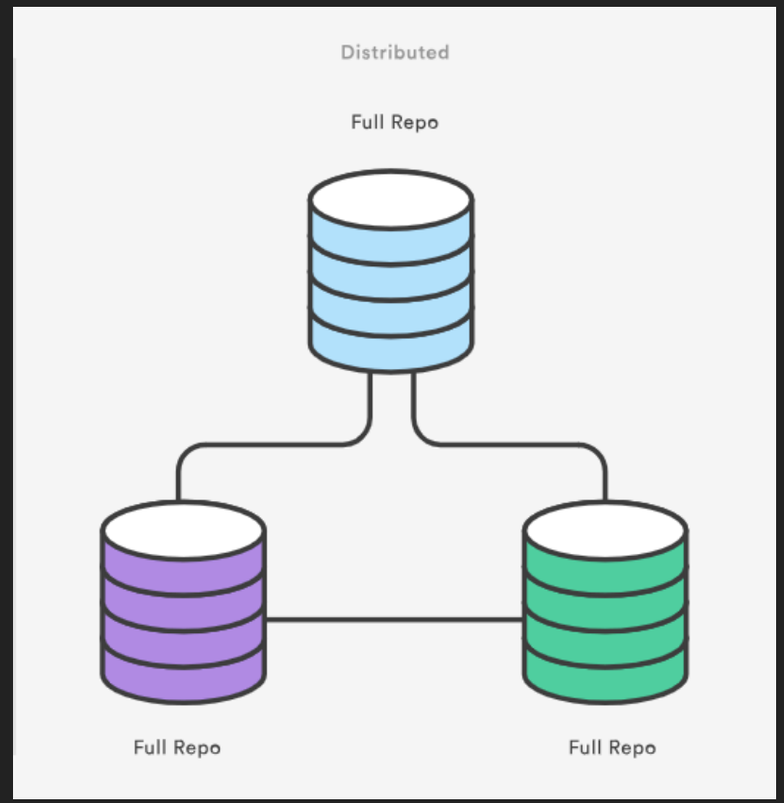
mar.grignard@uliege.be

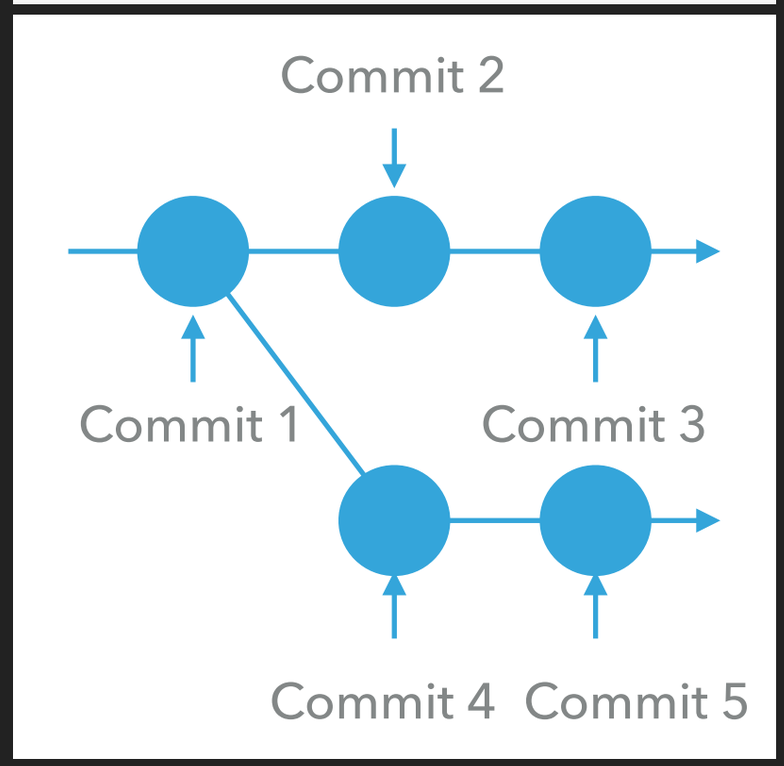
- High energy physicist, PhD&Ir
- Post-doctoral researcher at the GIGA - CRC in vivo imaging

gregory.hammad@uliege.be

GIT MODEL

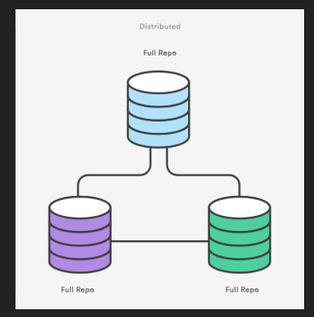
- Distributed system
  - Each user has a full copy of the project, its own history and structure.
  - That's why Git works offline too!
- Snapshots
  - Instead of keeping tracks of each file individually, Git creates a "commit snapshot" (commit) of your working copy of the project
  - Allows users to go back to a certain state of the project if necessary…
- Branch:
  - A series of commit snapshots from a given working directory.





mar.grignard@uliege.be gregory.hammad@uliege.be

GIT'S STRUCTURE



- Working directory:
  - List modified files
  - Modifications are not tracked (i.e not added to the history of modifications)
- Staging area:
  - List of "candidate" files whose modifications are meant to be tracked.
- Local repository:
  - Contains the whole history of changes (ie. commits)
- Remote repository:
  - Copy of the local repository, located on a distant server

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE!

Exercise 0

- Initialise a git repository:
  - git init
- Add a file
  - git add "myfile.csv"
- Commit the file:
  - git commit

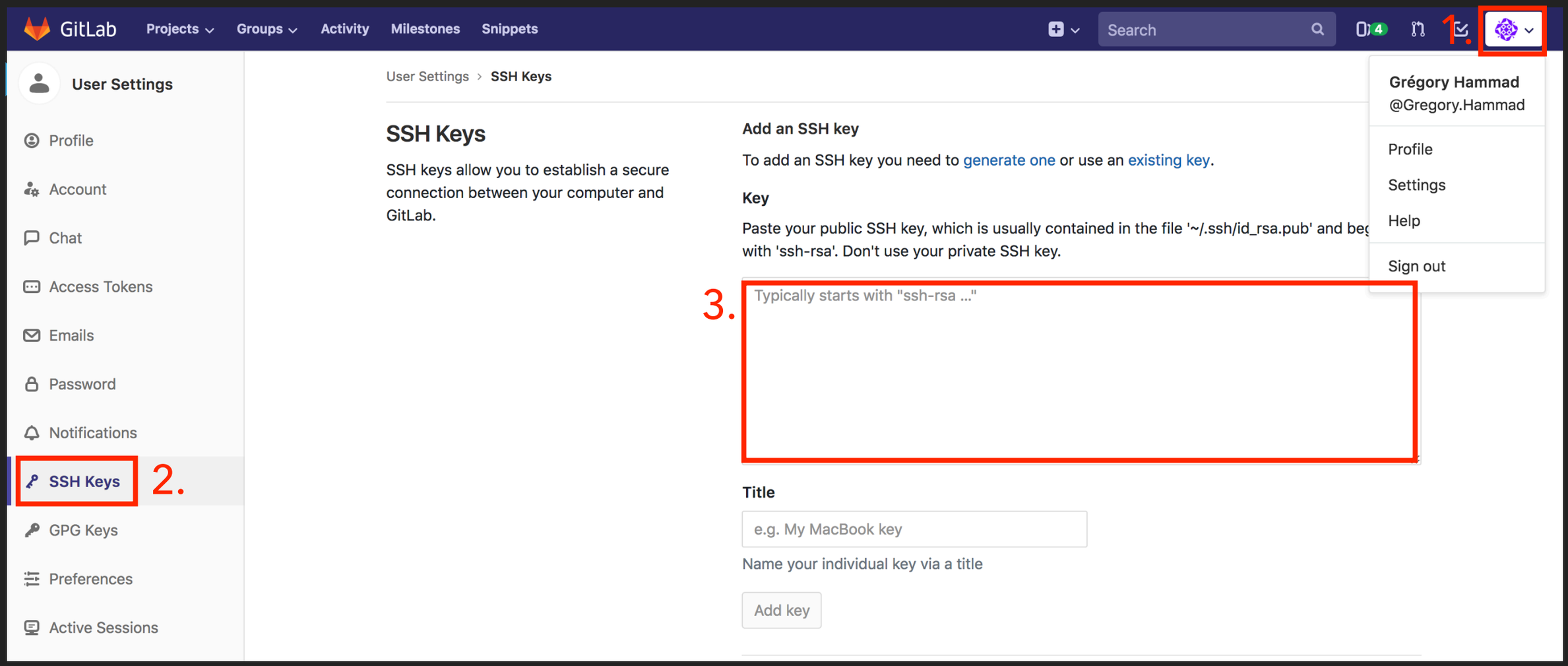
mar.grignard@uliege.be gregory.hammad@uliege.be

WARM-UP

- Add a `ssh key` to your Gitlab account:
  - SSH keys:
    - Private/public key pair
    - Identify yourself to a SSH server (like gitlab)
- Create a new key or use an existing one:
  - https://gitlab.uliege.be/help/ssh/README

WARM-UP

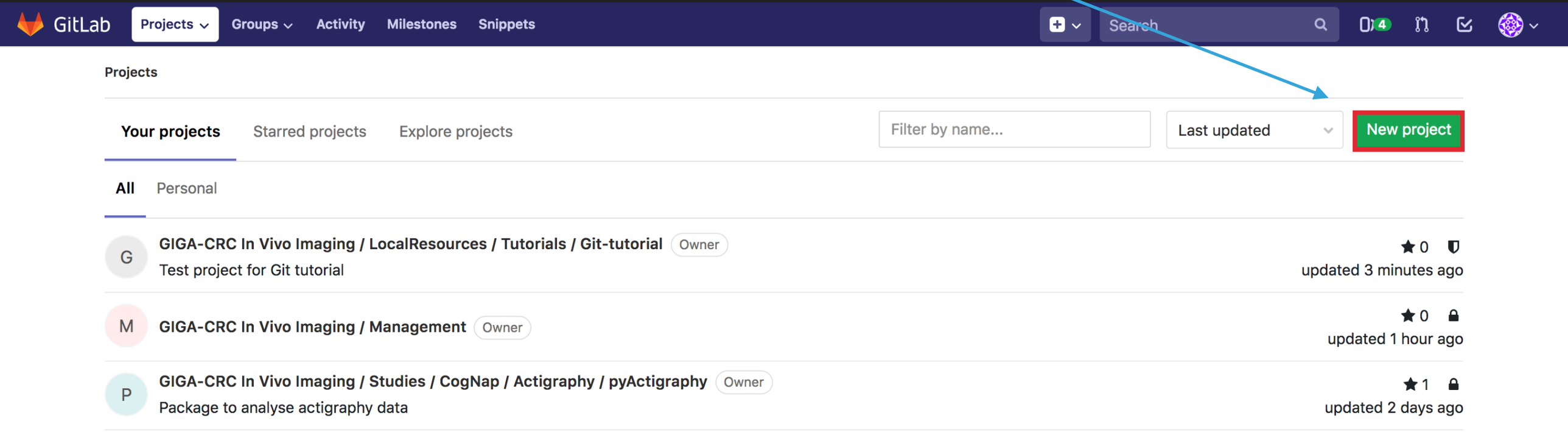
- Add a `ssh key` to your Gitlab account:



mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

- Create a Git project/repository:

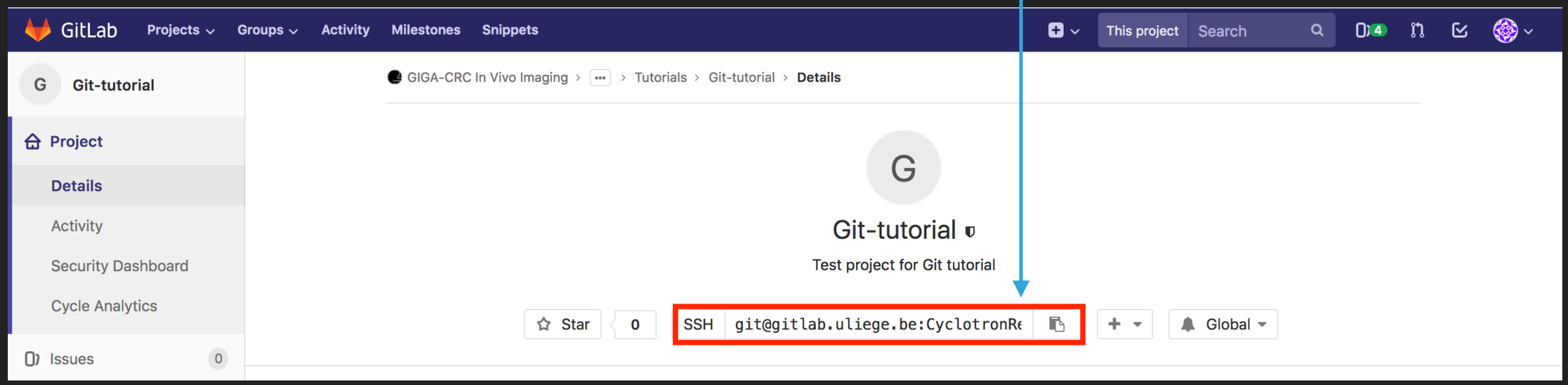


mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

- From an existing “local” folder:
  - “I want to put my existing code on Git“
  - `git init`

```
cd existing_folder
git init
git remote add origin git@gitlab.uliege.be:CyclotronResearchCentre/LocalResources/Tutorials/Git-tutorial.git
git add .
git commit -m "Initial commit"
git push -u origin master
```



mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

- From an existing project:
  - “I want to use and/or collaborate to an existing piece of code.”
  - `git clone`

```
git clone git@gitlab.uliege.be:CyclotronResearchCentre/LocalResources/Tutorials/Git-tutorial.git
cd Git-tutorial
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

- From an existing versioned `local` folder :
  - “I want to migrate from github.com/gitlab.giga to gitlab.uliege.be"
  - `git remote add`

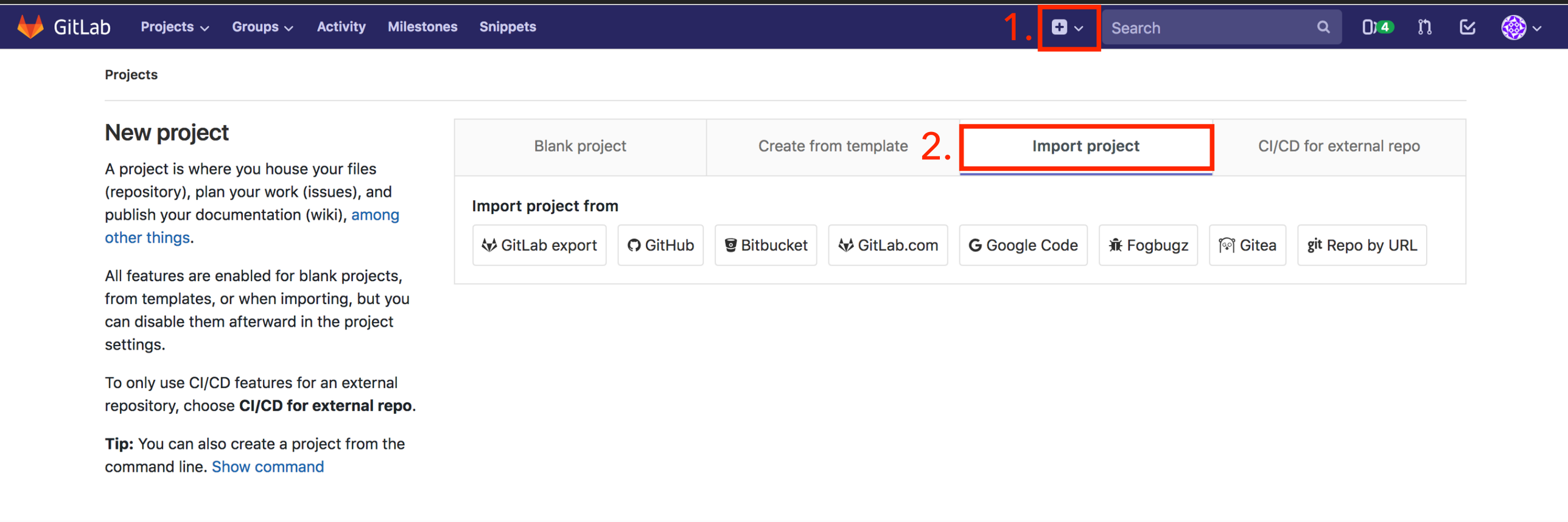
```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.uliege.be:CyclotronResearchCentre/LocalResources/Tutorials/Git-tutorial.git
git push -u origin --all
git push -u origin --tags
```

git remote set-url origin git@gitlab.uliege.be:CyclotronResearchCentre/Studies/CogNap/Actigraphy/pyActigraphy-Tutorial.git

- Or migrate the project via the interface…

GETTING STARTED

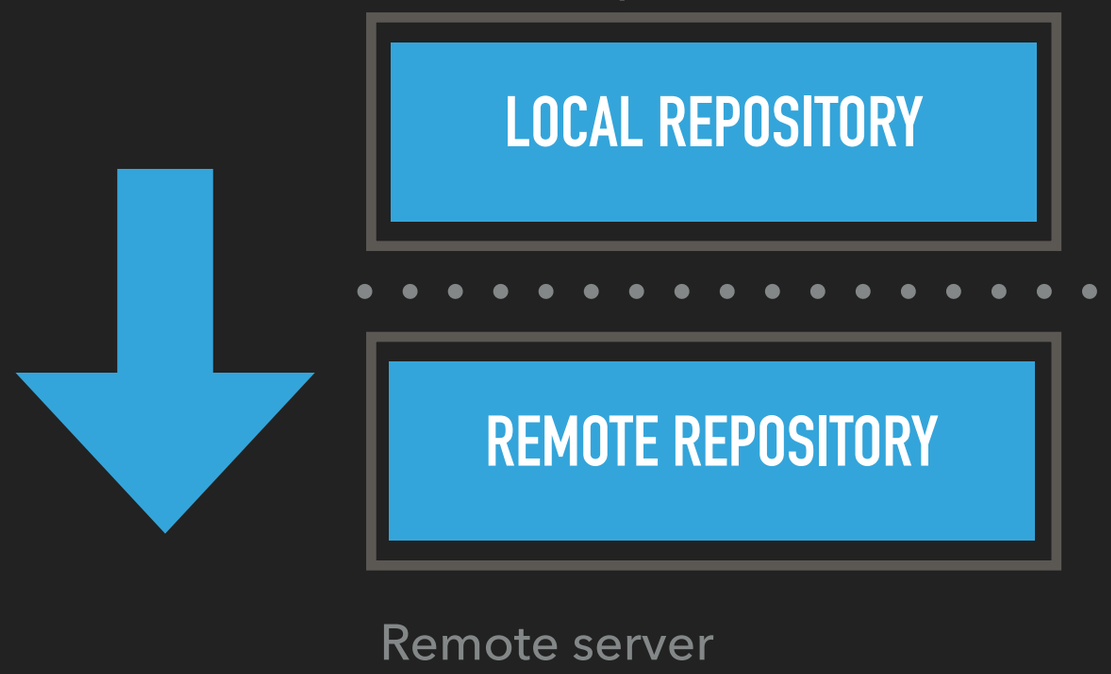
- Migrating existing projects:



mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

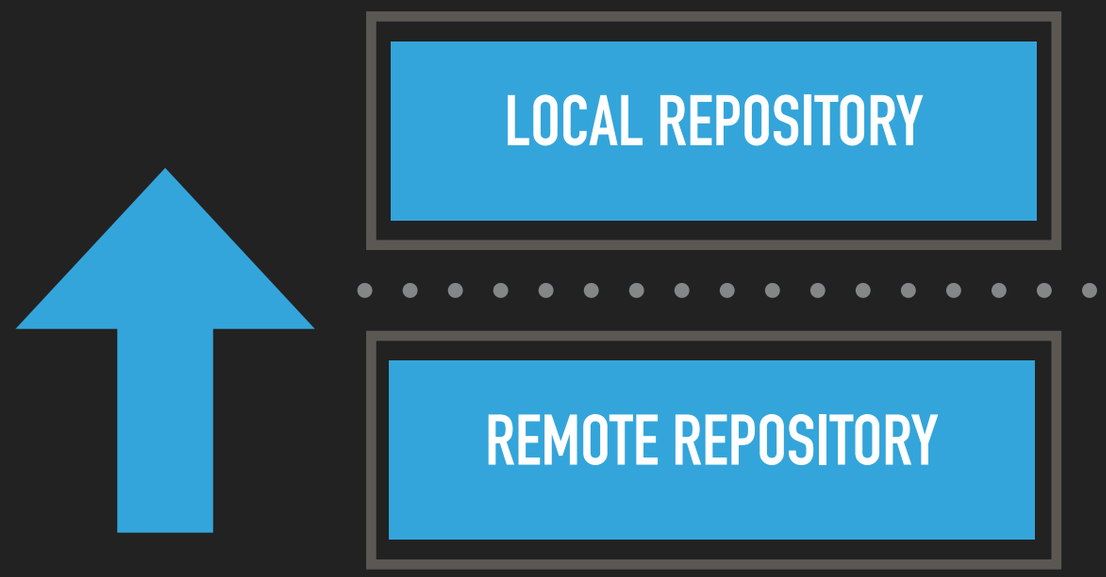
- Save your modifications:
  - `git push`
  - update the remote repository with the local changes (i.e. commits).



mar.grignard@uliege.be gregory.hammad@uliege.be

GETTING STARTED

- Update your working copy:
  - `git fetch; git merge` or `git pull`
  - apply changes recorded in the remote repository to your local copy



LET'S PRACTICE!

Exercise 1: `git push`

- Connect your local folder to an existing git repository:
  - git remote add origin git@gitlab.uliege.be:mygitrepo.git
- Add a file
- Commit the file
- Push the file to the remote server

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE!

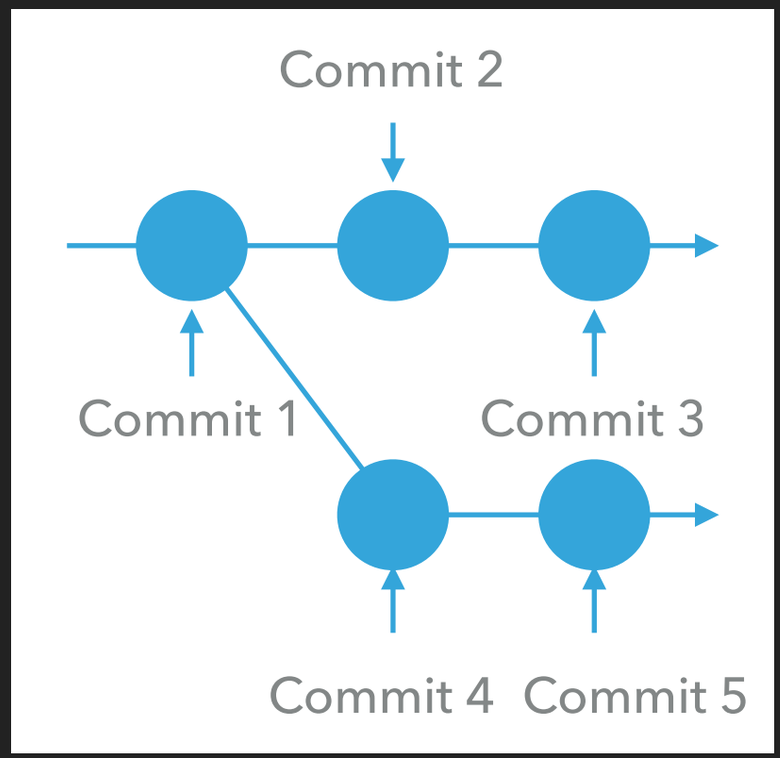
Exercise 2: `git pull`

- Modify a file in the remote repository
- Fetch the corresponding commit
- Inspect the modifications
- Apply the changes to your local copy of the file
- Congratulate yourself!

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S BRANCH! (DAVID BOWIE, 1983)

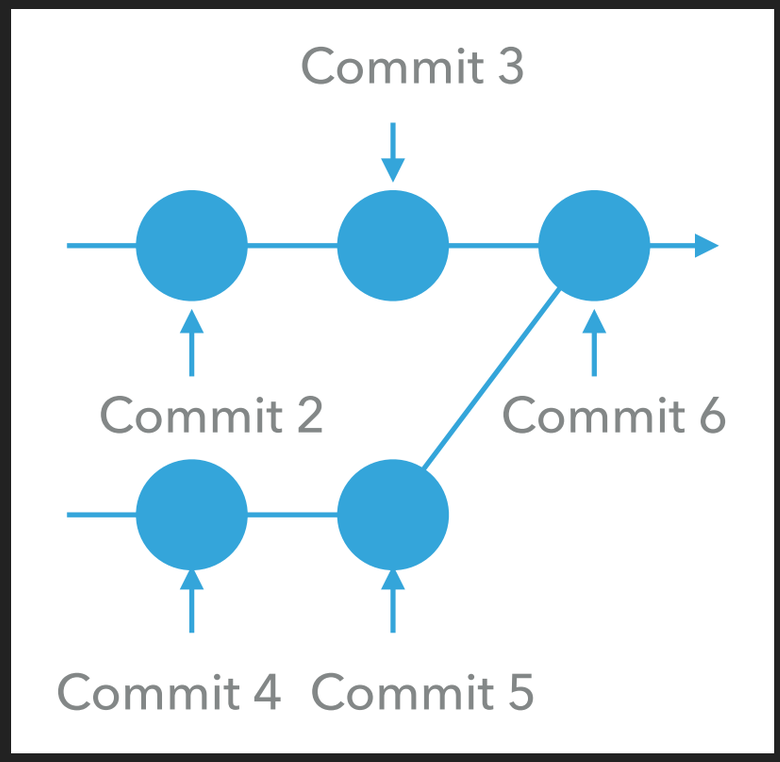
- Branch:
  - A series of commit snapshots from a given working directory.
  - Create a branch: `git branch <branch-name>`
  - Create a branch and check it out (ie. point that branch to your working dir): `git checkout -b <branch-name>`
- Push your branch to the remote server:
  - `git branch –set-upstream-to=origin/<branch-name>` + `git push`
  - or simply `git push -u origin <branch-name>`
- Delete branch: `git branch -d <branch-name>`



mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S BRANCH! (DAVID BOWIE, 1983)

- Merge:
  - Put the commit histories of two (or more) branches back together.
  - `git checkout master`
  - `git merge <branch-name>`
- Types of merging:
  - Fast-forward: the 2 branches did not diverge.
  - 3-way merging: there are commits that do appear in only one branch.
  - Good news; git choose what to do for you! Unless…




```
Some text without conflicts

<<<<<<< master

this tutorial is awesome!

=======

this tutorial is crappy!
```

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE!

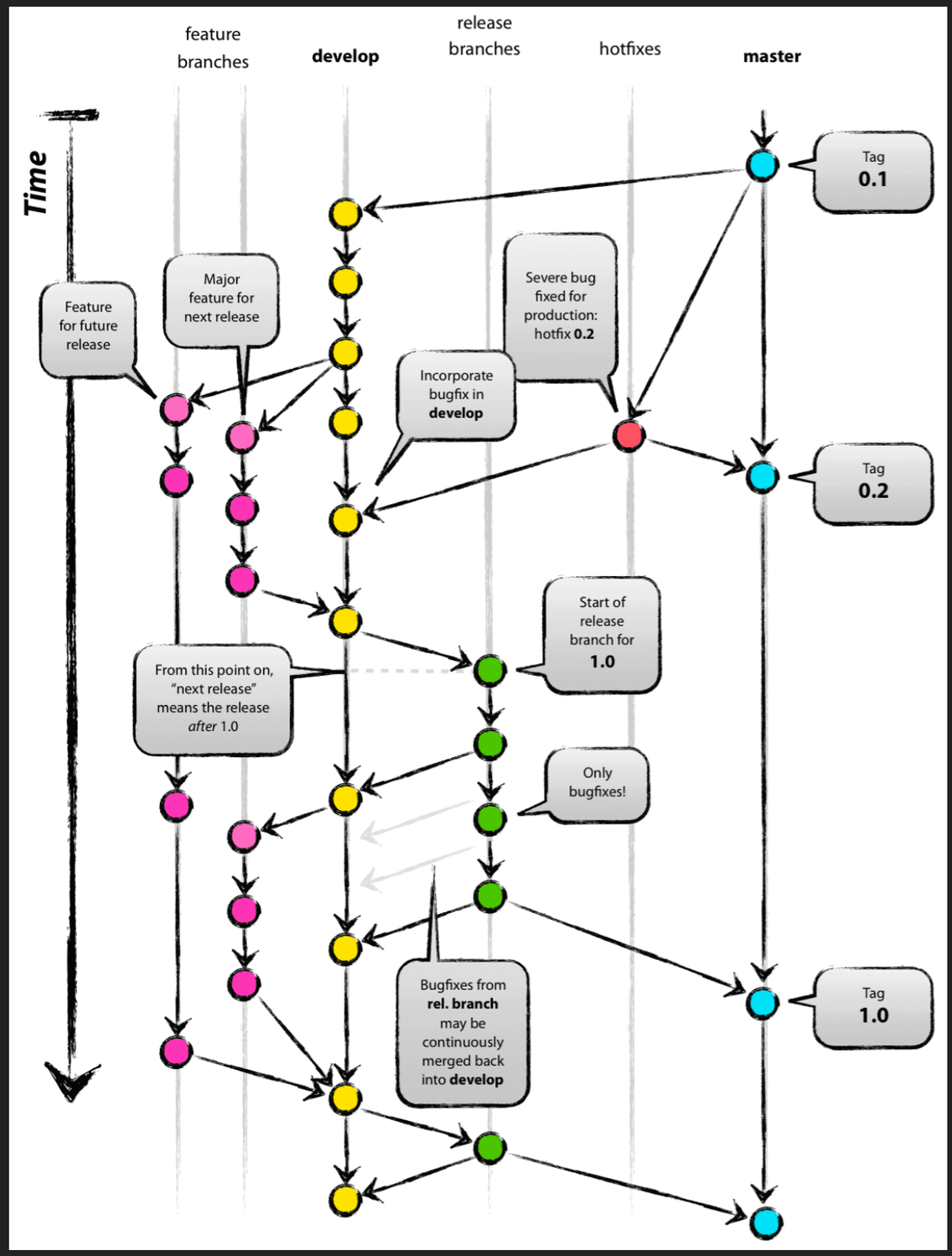
Exercise 3

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

mar.grignard@uliege.be gregory.hammad@uliege.be

STRUCTURING YOUR PROJECT

- Git flow
  - https://nvie.com/posts/a-successful-git-branching-model/
  - https://danielkummer.github.io/git-flow-cheatsheet/
- Other ways…
  - https://www.atlassian.com/git/tutorials/comparing-workflows



mar.grignard@uliege.be gregory.hammad@uliege.be

WHAT GITLAB CAN DO FOR YOU...

- Pull requests
- Issues
- CI/CD
  - Tests
  - Docs with Sphinx
  - Code coverage*
- Private code, public computer!?
- ...

*Not covered here

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Pull requests (PR):
  - "Please, use my modifications"
  - Request the merge of a branch into another one (usually master).
- Advantages:
  - Code review
  - Protection about unwanted changes
  - Nice interplay with issues

mar.grignard@uliege.be

- Typical workflow:
  - Create a "feature" branch locally
  - Publish it to the remote server (i.e set an upstream branch)
  - Work on your "feature"
  - Commit and push the modifications.
  - Merge the "feature" branch into the "master" branch via PR

gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

mar.grignard@uliege.be

gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

Exercise 4

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

Exercise 4

Push to remote -> remote merge

mar.grignard@uliege.be gregory.hammad@uliege.be

LET'S PRACTICE THE "PULL REQUEST" (PR)

Exercise 3

Local merge -> Push to remote

- Create your own branch and check it out:
  - git branch "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Switch back (i.e check out) to the main branch
- Merge the modifications from "NameOfYourBranch" onto the main branch
- Submit the resulting modifications to the remote server

Exercise 4

Push to remote -> remote merge

- Create your own branch, track its modifications on the remote server and check it out:
  - git branch -set-upstream-to=origin/ "NameOfYourBranch" "NameOfYourBranch"
  - git checkout "NameOfYourBranch"
- Modify a local file and commit the modifications (cf previous Exo) to the new branch
- Submit the resulting modifications to the remote server
- Merge the modifications from "NameOfYourBranch" onto the main branch **directly on the remote server via a PR**

mar.grignard@uliege.be gregory.hammad@uliege.be

GUI TO THE RESCUE

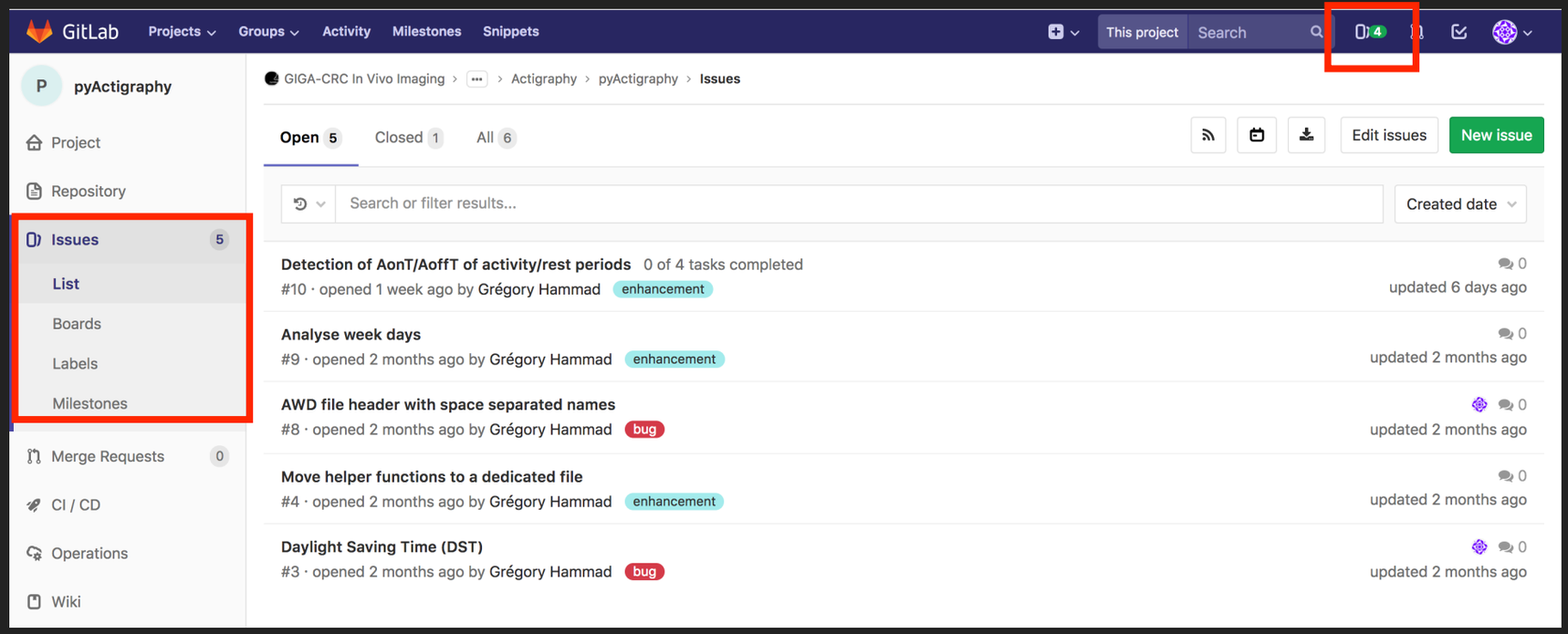
- Demo

https://gitahead.github.io/gitahead.com/

mar.grignard@uliege.be gregory.hammad@uliege.be

USER'S CORNER

- Issues:

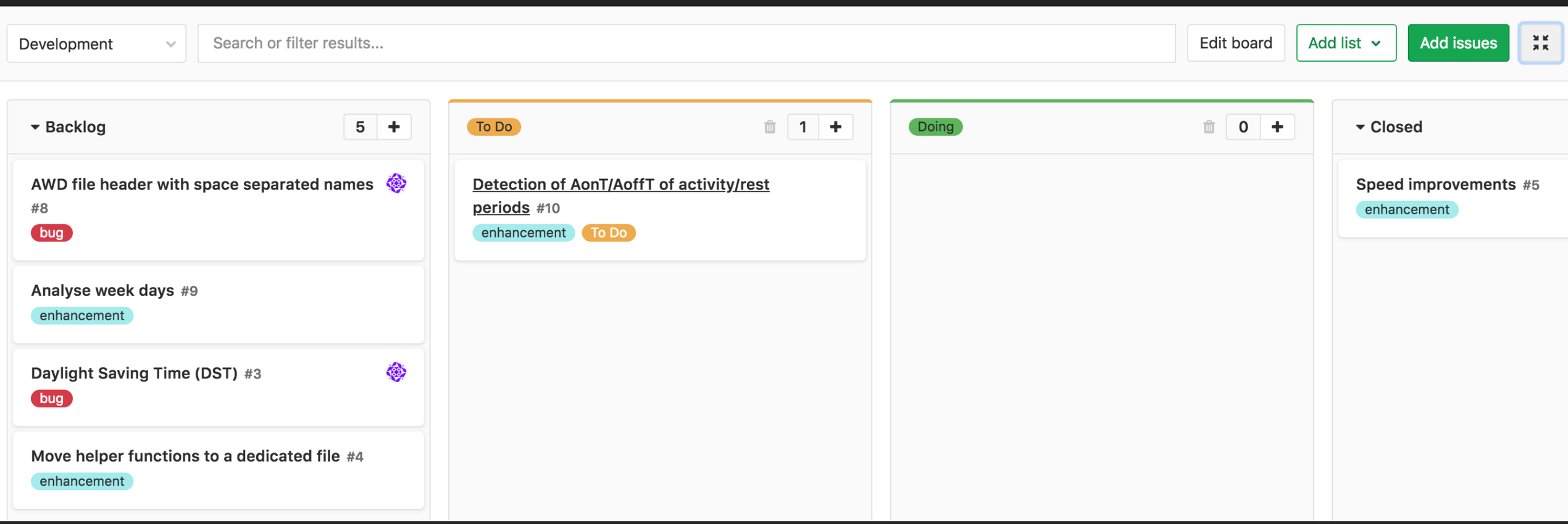


- Report bugs
- Suggest improvements
  - New functionalities
  - Documentation…

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Issue board "Kanban":
  - Prioritise and organise your developments



mar.grignard@uliege.be gregory.hammad@uliege.be

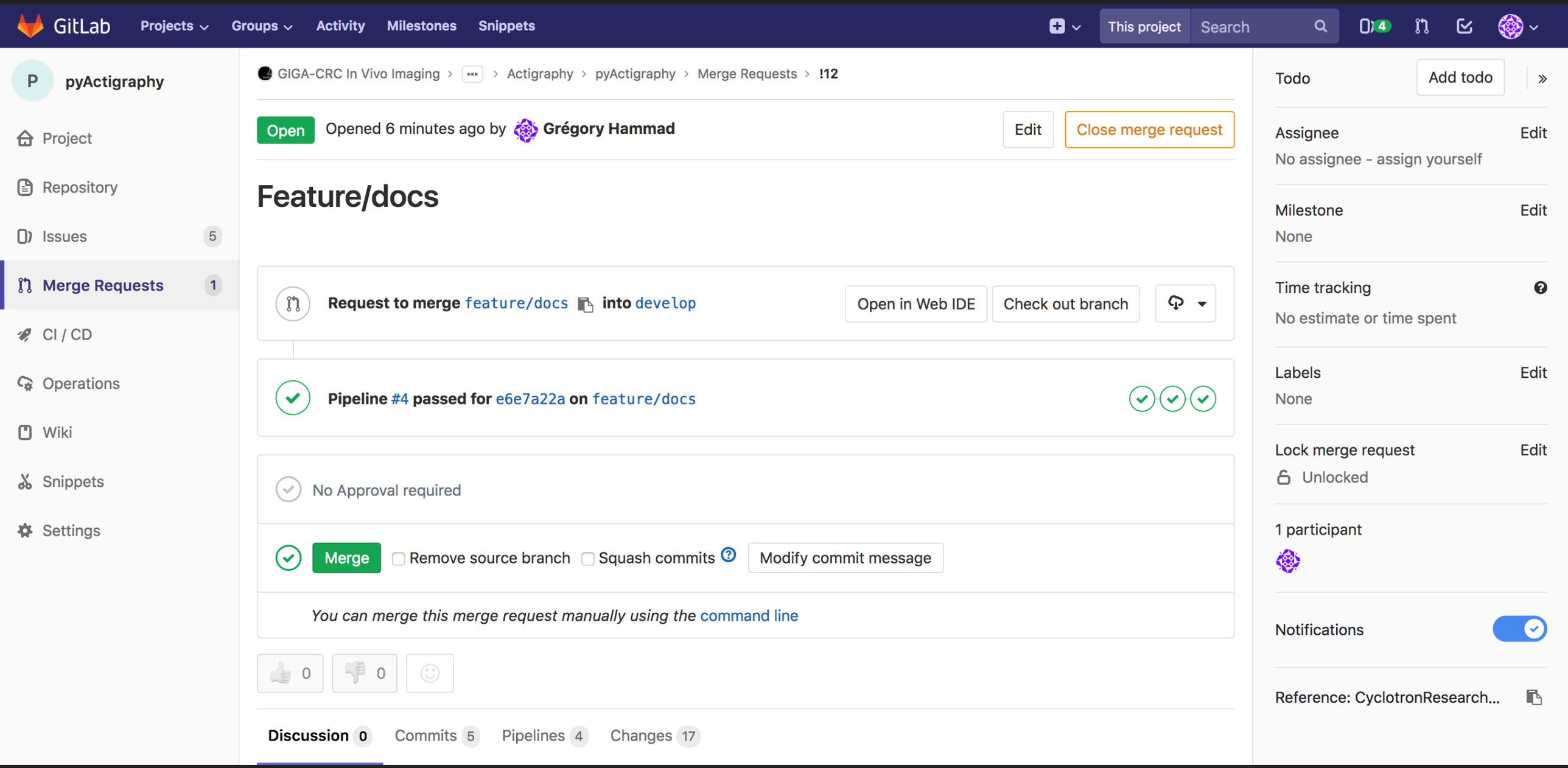
DEVELOPER'S CORNER

- Continuous integration/deployment:
  - Run a set of jobs (=pipeline), each time a commit is pushed
    - Mostly "test" jobs but not only (build, doc, …)
  - Ensure the quality* of the code does not degrade…
  - Typically;
    - Protect "master" (https://docs.gitlab.com/ee/user/project/protected_branches.html)
    - Configure CI/CD jobs to run on develop
    - Use output of CI/CD jobs to validate Merge request from "develop" to "master"

mar.grignard@uliege.be gregory.hammad@uliege.be

*broad sense

DEVELOPER'S CORNER



mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Continuous integration/deployment:
  - Jobs are run on:
    - local ressources
    - shared "runners" (external machines)
      - Not configured yet in gitlab.uliege.be
      - Ongoing discussion with the SEGI…
      - Target: 31/01/2019

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Continuous integration/deployment:
  - How to set up a pipeline?
    - Simply add a .gitlab-ci.yml file to your repository
    - Configure (or choose if available) a runner (i.e. a machine to run the pipeline jobs)
      - Project's Gitlab page > Settings > CI / CD

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Continuous integration/deployment:
  - Example:
    - Using a docker image
    - Define 3 jobs:
      - "build": ~ compilation test
      - "test": run a test suite
      - "deploy": create doc and make it available
- What about Matlab? Use Octave!

mar.grignard@uliege.be

```yaml
image: python:latest

# Change pip's cache directory to be inside the pro
# only cache local items.
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache"

# Pip's cache doesn't store the python packages
# https://pip.pypa.io/en/stable/reference/pip_insta
# If you want to also cache the installed packages,
# them in a virtualenv and cache it as well.
cache:
  paths:
    - .cache/pip
    - venv/

stages:
  - build
  - test
  - deploy

before_script:
  - python -V               # Print out python vers
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate

run:
  stage: build
  script:
  - pip install -e .

test:
  stage: test
  script:
  - pip install pytest
  - python -m pytest -vv --disable-pytest-warnings

pages:
  stage: deploy
  script:
  - pip install sphinx sphinx-bootstrap-theme
  - cd docs ; make html
  - mv build/html/ ../public/
  artifacts:
    paths:
    - public
  only:
  - feature/docs
```

DEVELOPER'S CORNER

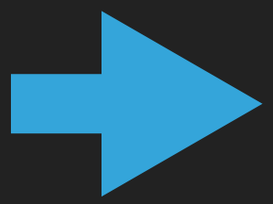
- Automatic documentation
  - Sphinx (http://www.sphinx-doc.org/en/master/index.html)
- Extract informations directly from the source code:

reader.py

```python
def read_raw(input_path, reader_type, n_jobs=1, prefer=None, verbose=0):
    """Reader function for multiple raw files.

    Parameters
    ----------
    input_path: str
        Path to the files. Accept wild cards.
        E.g. '/path/to/my/files/*.csv'
    reader_type: str
        Reader type.
        Supported types: AWD (ActiWatch), MTN (MotionWatch8)
        and RPX (Respironics)
    n_jobs: int
        Number of CPU to use for parallel reading
    prefer: str
        Soft hint to choose the default backendself.
        Supported option:'processes', 'threads'.
        See joblib package documentation for more info.
        Default is None.
    verbose: int
        Display a progress meter if set to a value > 0.
        Default is 0.

    Returns
    -------
    raw : list
        A list of instances of RawAWD, RawMTN or RawRPX
    """
```



pyActigraphy 0.1.dev0 Quick Start Documentation API Examples Source Search

pyActigraphy.io.read_raw

pyActigraphy.io. **read_raw** (*input_path, reader_type, n_jobs=1, prefer=None, verbose=0*) [source]

Reader function for multiple raw files.

Parameters:
- **input_path** (*str*) – Path to the files. Accept wild cards. E.g. '/path/to/my/files/*.csv'
- **reader_type** (*str*) – Reader type. Supported types: AWD (ActiWatch), MTN (MotionWatch8) and RPX (Respironics)
- **n_jobs** (*int*) – Number of CPU to use for parallel reading
- **prefer** (*str*) – Soft hint to choose the default backendself. Supported option:'processes', 'threads'. See joblib package documentation for more info. Default is None.
- **verbose** (*int*) – Display a progress meter if set to a value > 0. Default is 0.

Returns: **raw** – A list of instances of RawAWD, RawMTN or RawRPX

Return type: list

© Copyright 2018-2018, Grégory Hammad. Back to top
Created using Sphinx 1.7.8.

HTML

mar.gnghard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Automatic documentation
  - Advantages:
    - Catchy and user-friendly online documentation
    - Supports multiple languages (including Matlab: https://pypi.org/project/sphinxcontrib-matlabdomain/)
  - Caveats:
    - Quite complex file structure
    - Use reStructuredText directives (not Markdow…)

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Private code but public computers? How to deploy code?
  - A not-so fictional example:
    - "I want to maintain up-to-date the code I put on the MRI computer (aka: cogent). For this, I cloned the project to the MRI computer and 'git pull' each time I need to update the code".
- For this to work, you just need to install your private key on the cogent computer. Easy…
- Wait a minute! Cogent is a public computer. Everyone will have access to your private key!!!

mar.grignard@uliege.be gregory.hammad@uliege.be

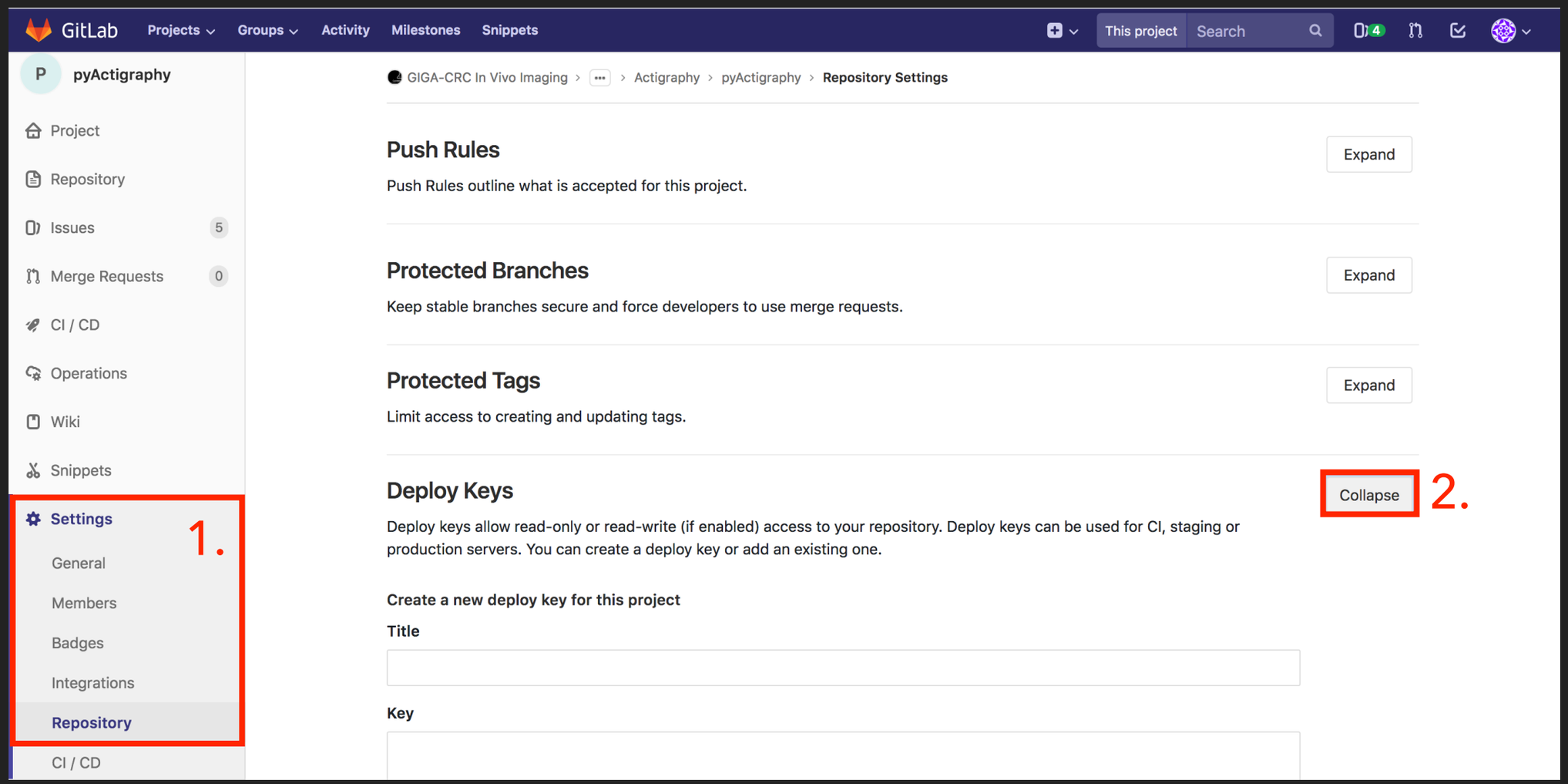
DEVELOPER'S CORNER

- Private code but public computers? How to deploy code?
  - Deploy keys to the rescue
    - https://docs.gitlab.com/ee/ssh/#global-shared-deploy-keys
  - In brief:
    - Generate a dedicated ssh key pair
    - Put the private on the "public" computer
    - Link the key to the project and assign it as "read only"!

mar.grignard@uliege.be gregory.hammad@uliege.be

DEVELOPER'S CORNER

- Private code but public computers? How to deploy code?



mar.grignard@uliege.be gregory.hammad@uliege.be

BACK-UP SLIDES

INSTALL GIT ON WINDOWS

- https://gitforwindows.org/
-

mar.grignard@uliege.be gregory.hammad@uliege.be

GLOSSARY

Terms	Github	Gitlab
Directory	Repository	Project
	Pull request	Merge request

- Ressources

  - http://git-school.github.io/visualizing-git/

  -

mar.grignard@uliege.be gregory.hammad@uliege.be