



LIÈGE université  
GIGA  
CRC In vivo Imaging

GIGA DOCTORAL SCHOOL 2020

---

# GIT IN PRACTICE

HANDS-ON TUTORIAL

## WHO ARE WE?



- ▶ Phd student at the GIGA - CRC in vivo imaging
  - ▶ "Electromagnetic modelling of a head"

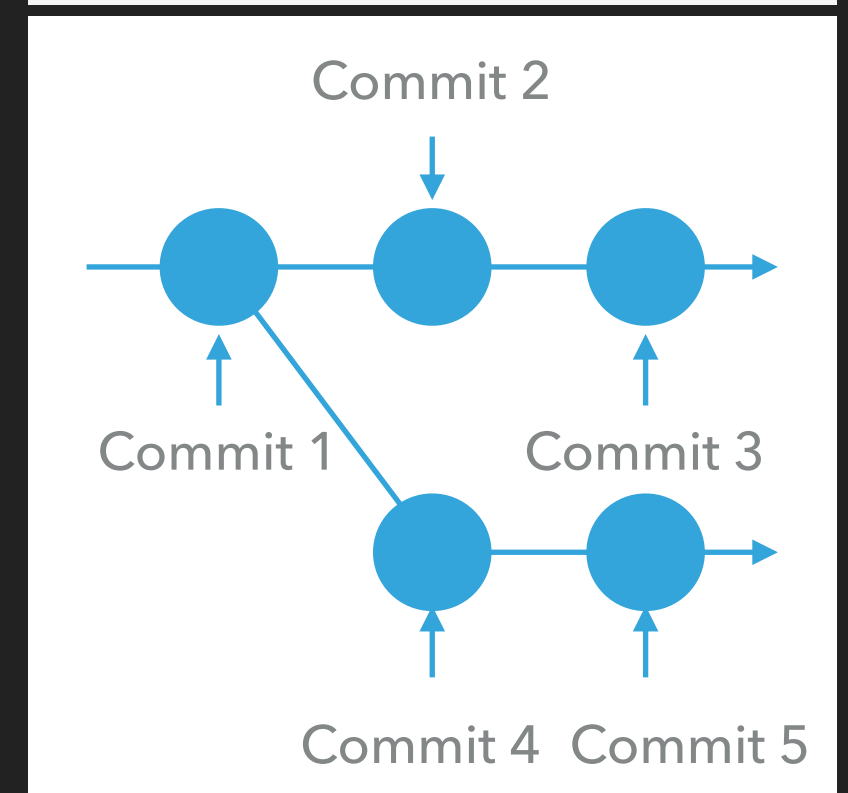
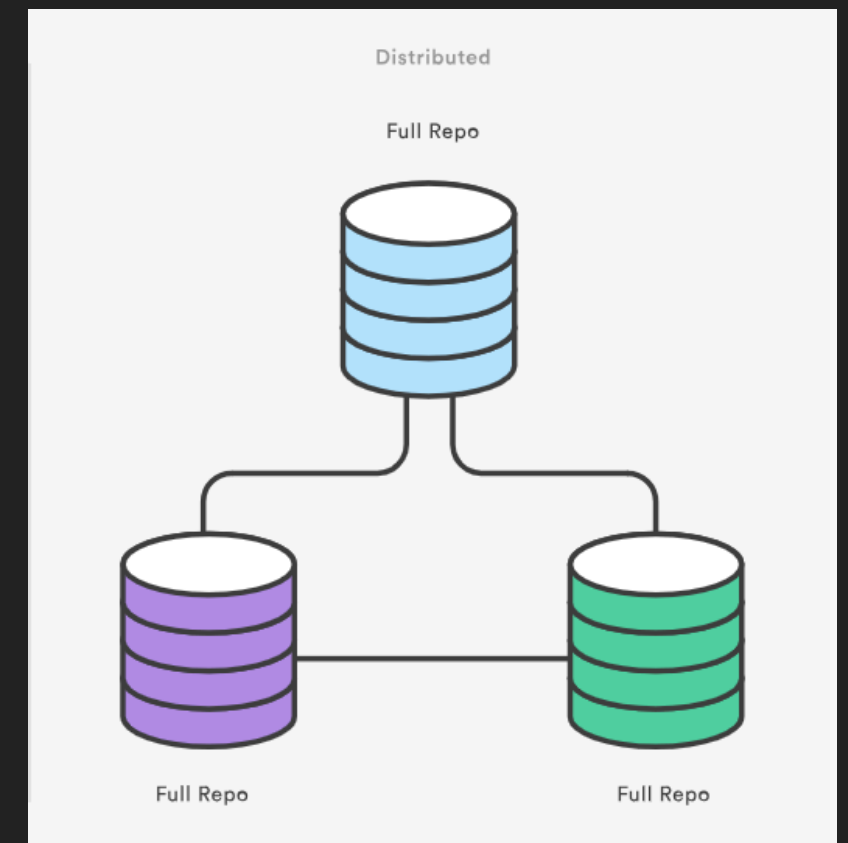
- ▶ High energy physicist, PhD&Ir
- ▶ Post-doctoral researcher at the GIGA - CRC in vivo imaging

[mar.grignard@uliege.be](mailto:mar.grignard@uliege.be)

[gregory.hammad@uliege.be](mailto:gregory.hammad@uliege.be)

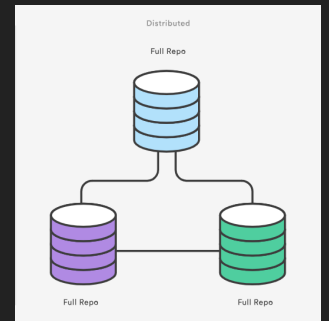
# GIT MODEL

- ▶ 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.



# 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



Local computer



.....



Remote server

# LET'S PRACTICE!

## Exercise 0

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

## 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>

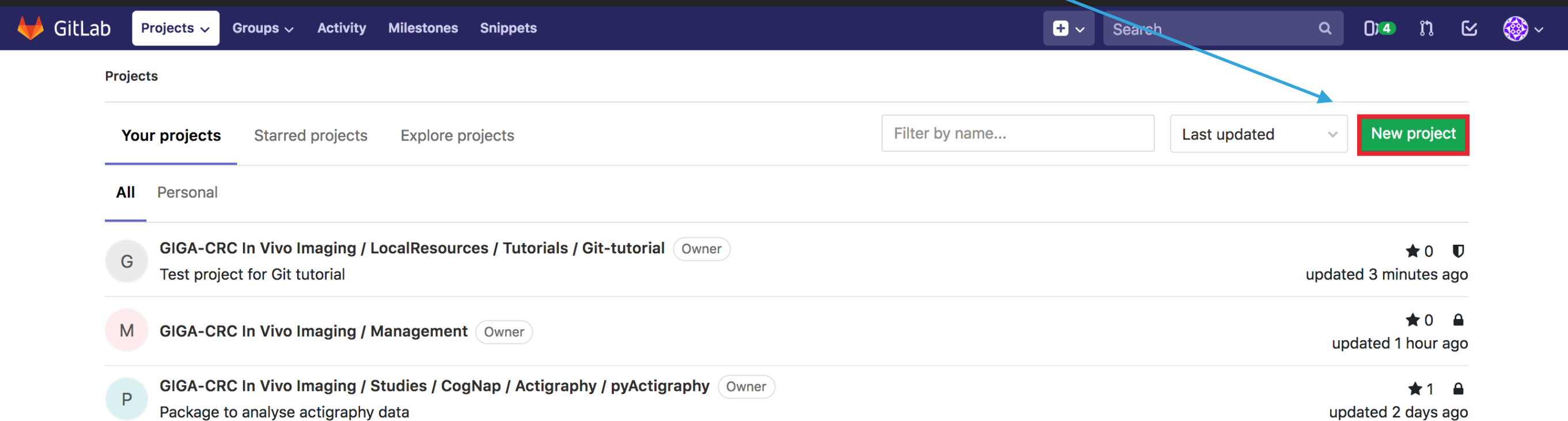
# WARM-UP

## ▶ Add a `ssh key` to your Gitlab account:

The screenshot shows the GitLab user settings page for SSH Keys. The page is titled "User Settings > SSH Keys". On the left sidebar, the "SSH Keys" option is highlighted with a red box and a red "2.". The main content area is titled "SSH Keys" and contains the following text: "SSH keys allow you to establish a secure connection between your computer and GitLab." Below this, there is a section titled "Add an SSH key" with the instruction: "To add an SSH key you need to [generate one](#) or use an [existing key](#)." The "Key" field is a large text input area, highlighted with a red box and a red "3.". Above the input area, there is a red note: "3. Typically starts with 'ssh-rsa ...'". Below the input area, there is a "Title" field with the placeholder text "e.g. My MacBook key" and the instruction "Name your individual key via a title". At the bottom of the form, there is an "Add key" button. In the top right corner, the user's profile menu is open, showing the user's name "Grégory Hammad" and the email address "@Gregory.Hammad", along with options for "Profile", "Settings", "Help", and "Sign out".

# GETTING STARTED

## ▶ Create a Git project/repository:



The screenshot shows the GitLab web interface. At the top, there is a navigation bar with the GitLab logo and several menu items: 'Projects', 'Groups', 'Activity', 'Milestones', and 'Snippets'. To the right of the navigation bar, there is a search bar and several utility icons. Below the navigation bar, the main content area is titled 'Projects'. There are three tabs: 'Your projects', 'Starred projects', and 'Explore projects'. The 'Your projects' tab is active. Below the tabs, there are filters: 'Filter by name...' and 'Last updated'. A red box highlights the 'New project' button, and a blue arrow points from the text 'Create a Git project/repository:' to this button. Below the filters, there is a list of projects. The first project is 'GIGA-CRC In Vivo Imaging / LocalResources / Tutorials / Git-tutorial' with the owner 'G' and 'Test project for Git tutorial'. The second project is 'GIGA-CRC In Vivo Imaging / Management' with the owner 'M' and 'GIGA-CRC In Vivo Imaging / Management'. The third project is 'GIGA-CRC In Vivo Imaging / Studies / CogNap / Actigraphy / pyActigraphy' with the owner 'P' and 'Package to analyse actigraphy data'.

GitLab Projects Groups Activity Milestones Snippets

Filter by name... Last updated New project

All Personal

G GIGA-CRC In Vivo Imaging / LocalResources / Tutorials / Git-tutorial Owner  
Test project for Git tutorial ★ 0 updated 3 minutes ago

M GIGA-CRC In Vivo Imaging / Management Owner  
GIGA-CRC In Vivo Imaging / Management ★ 0 updated 1 hour ago

P GIGA-CRC In Vivo Imaging / Studies / CogNap / Actigraphy / pyActigraphy Owner  
Package to analyse actigraphy data ★ 1 updated 2 days ago

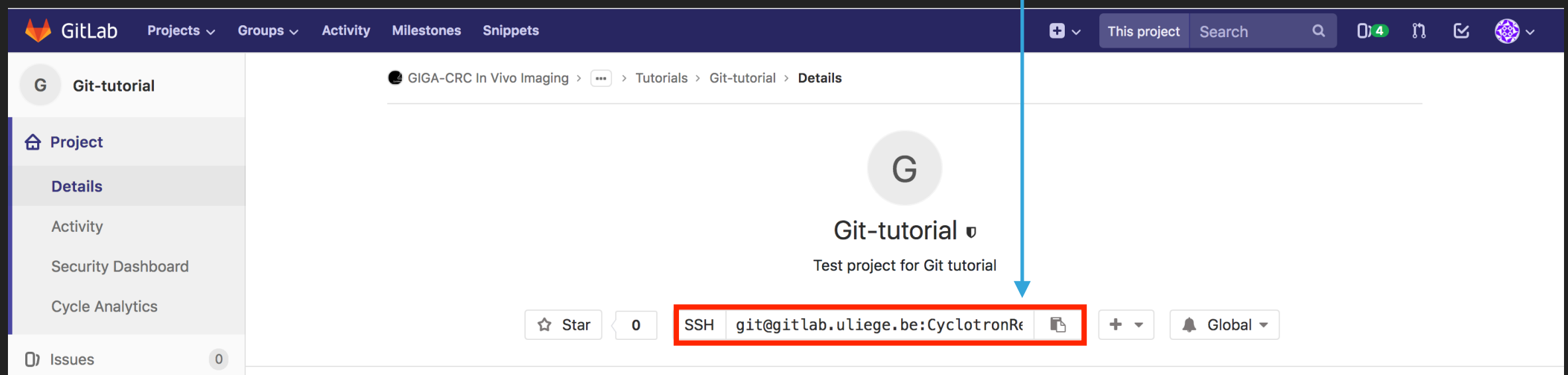


# GETTING STARTED

- ▶ 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
```



# 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
```

## GETTING STARTED

- ▶ From an existing versioned `local` folder :
  - ▶ "I want to migrate from [github.com/gitlab.giga](https://github.com/gitlab/giga) to [gitlab.uliege.be](https://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:

The screenshot shows the GitLab interface for creating a new project. The top navigation bar includes the GitLab logo, 'Projects', 'Groups', 'Activity', 'Milestones', and 'Snippets'. A search bar is on the right. The main content area is titled 'New project' and contains the following text:

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), [among other things](#).

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

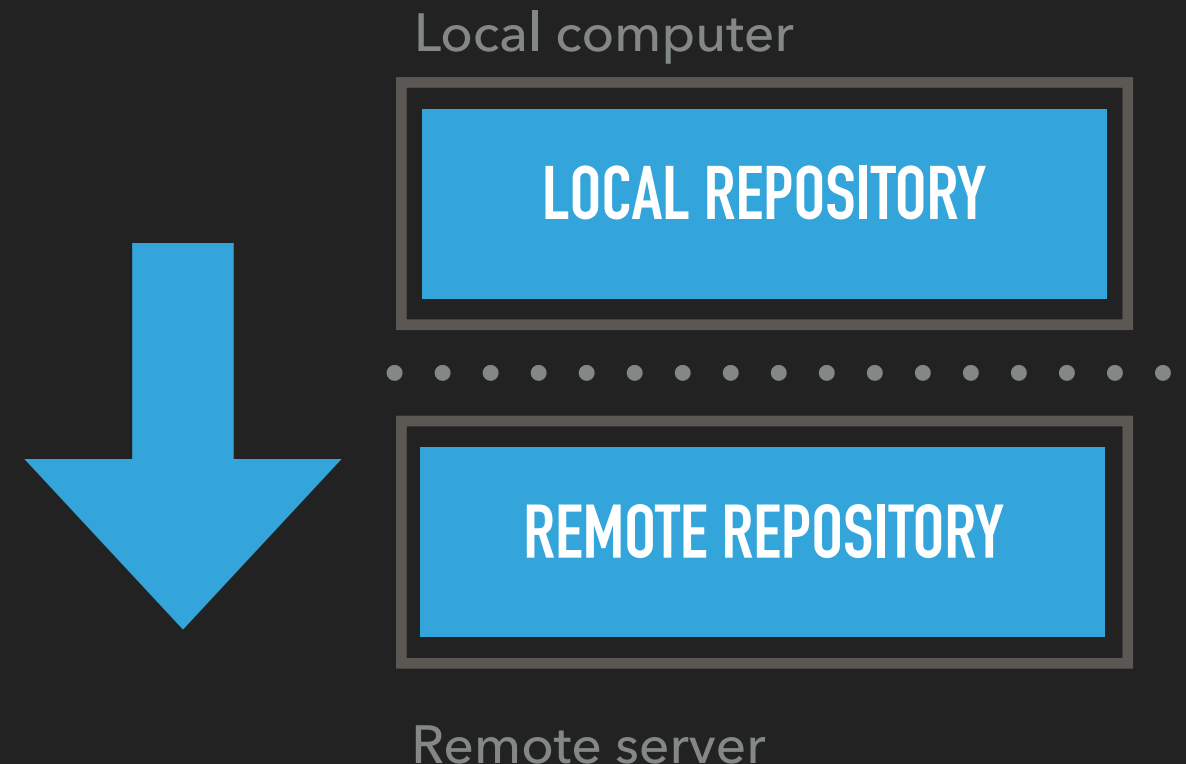
To only use CI/CD features for an external repository, choose **CI/CD for external repo**.

**Tip:** You can also create a project from the command line. [Show command](#)

The 'Import project' button is highlighted with a red box and labeled '2.'. The '+' icon in the top navigation bar is labeled '1.'.

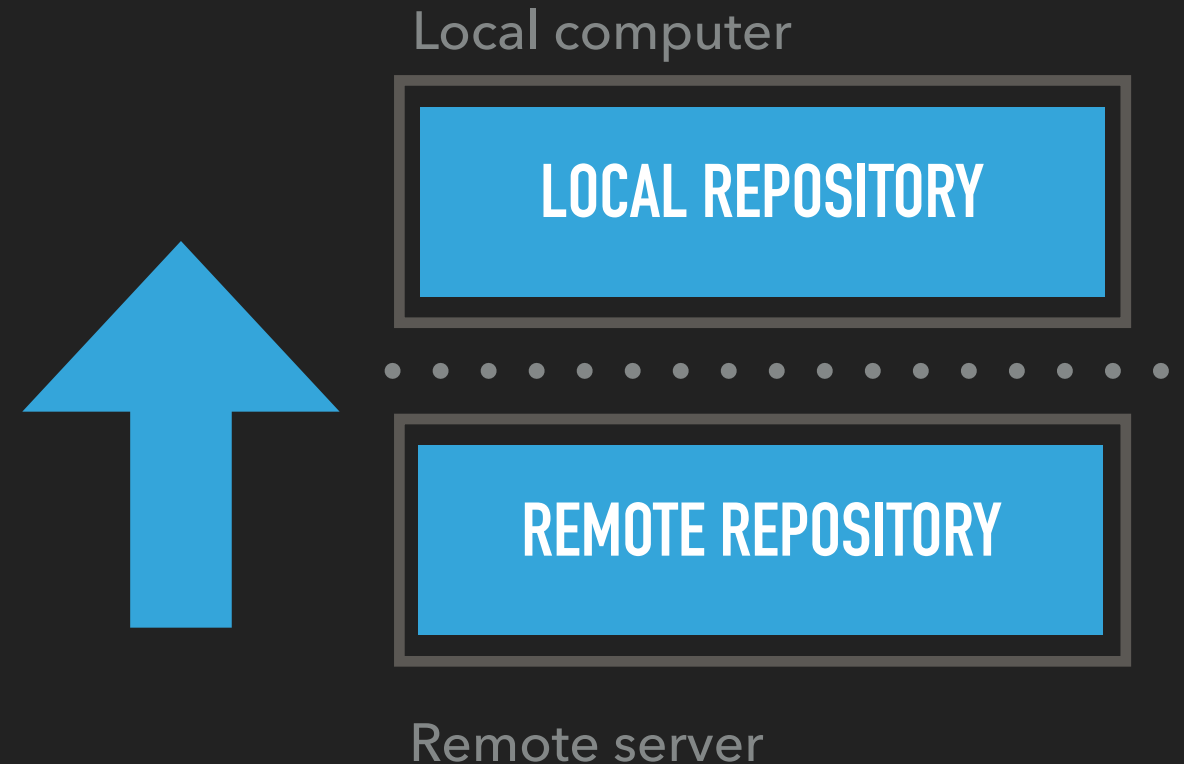
# GETTING STARTED

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



# 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

## LET'S PRACTICE!

### Exercise 2: `git pull`

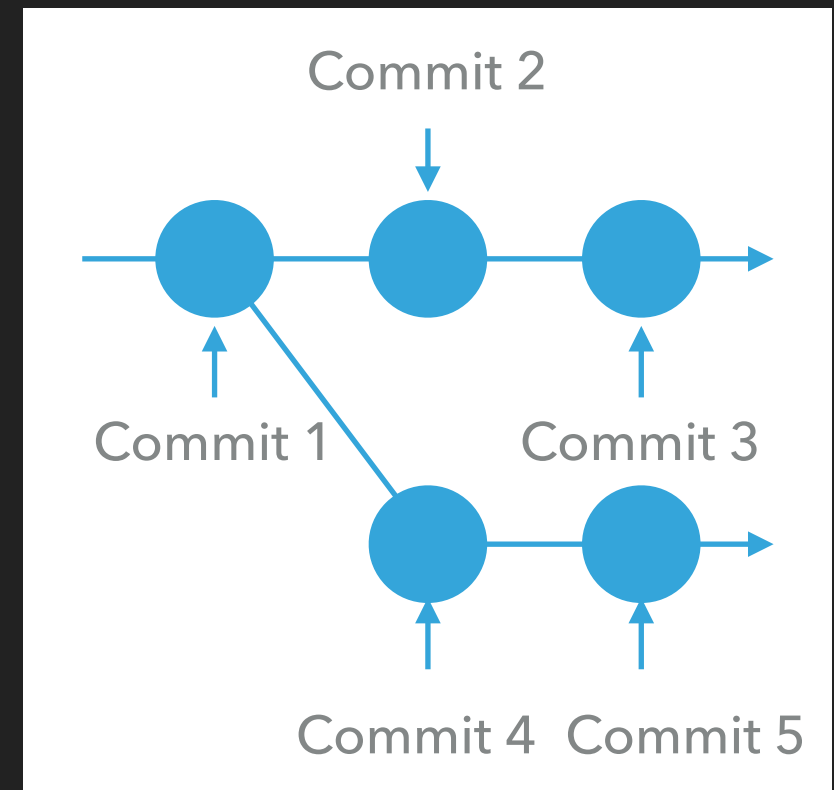
- ▶ Modify a file in the remote repository
- ▶ Fetch the corresponding commit
- ▶ Inspect the modifications
- ▶ Apply the changes to your local copy of the file
- ▶ Congratulate yourself!



# 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>``

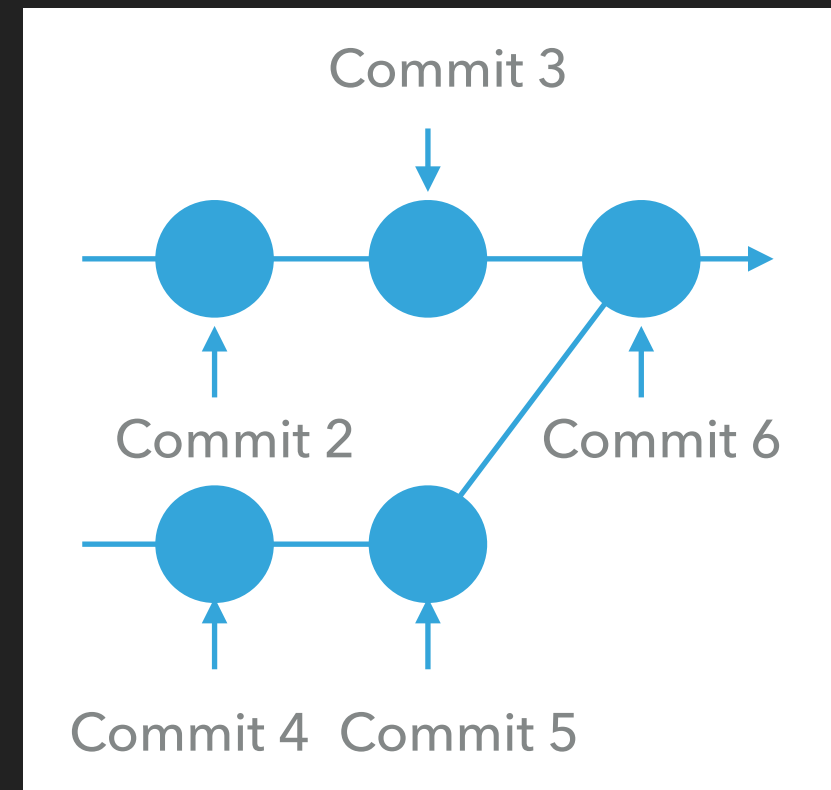


# LET'S BRANCH!

(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!

# 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

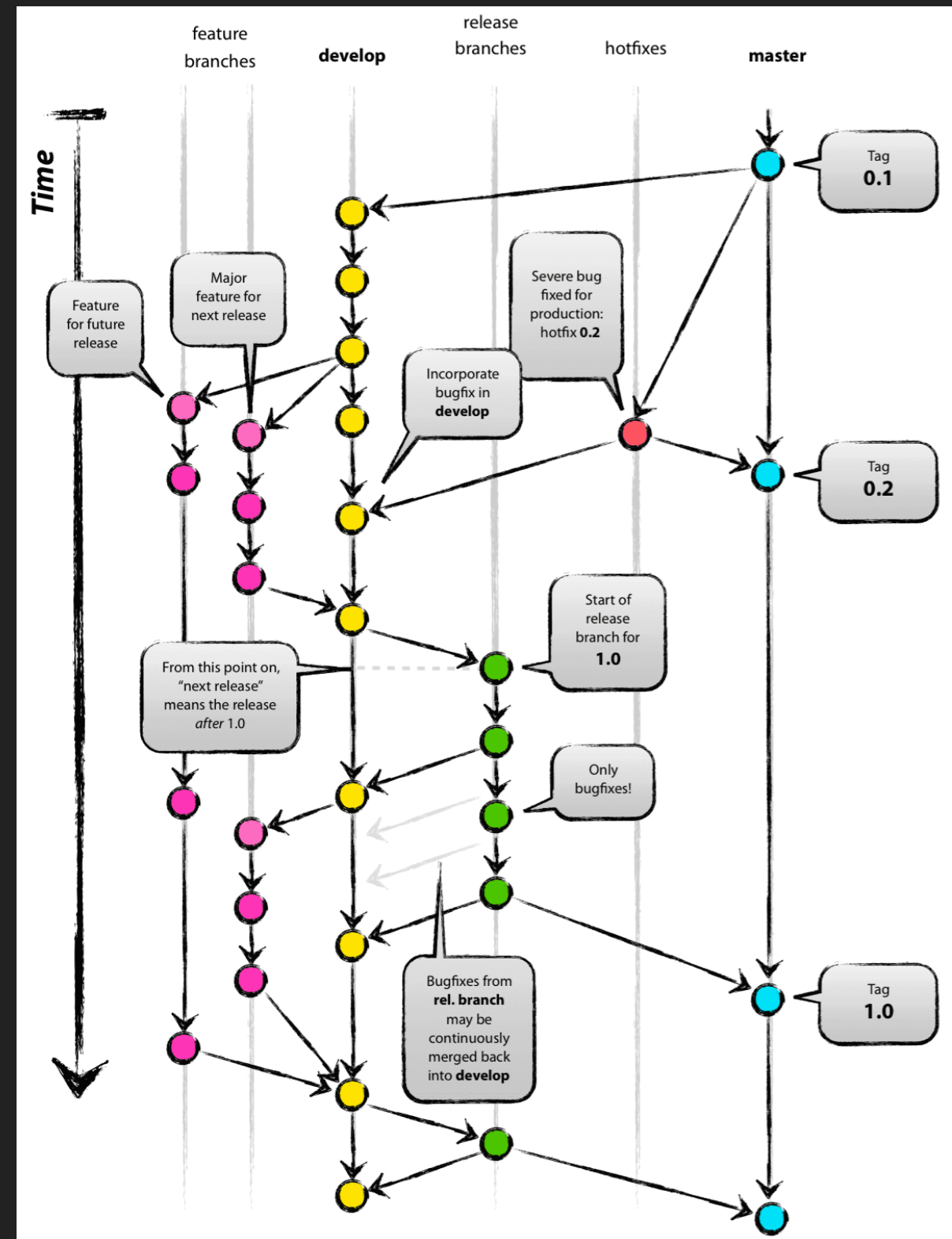
# STRUCTURING YOUR PROJECT

## ▶ 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>



## WHAT GITLAB CAN DO FOR YOU...

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

\*Not covered here

## 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
- ▶ 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

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

## Exercise 3

Local merge -> Push to remote

# 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



# 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

# 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

# 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

# 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

# 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**

# GUI TO THE RESCUE

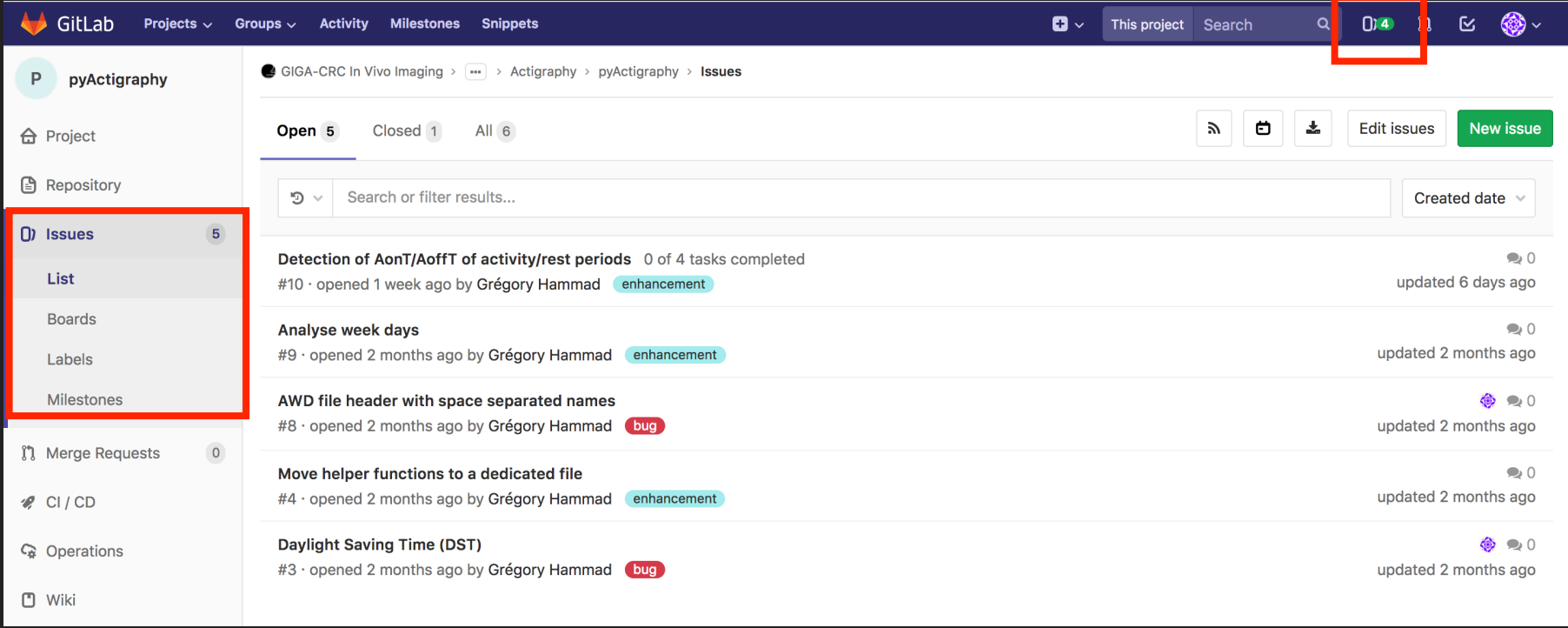
## ▶ Demo



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

# USER'S CORNER

## ▶ Issues:



The screenshot shows the GitLab interface for the 'pyActigraphy' project. The left sidebar contains a navigation menu with 'Issues' highlighted. The main content area displays a list of issues with the following details:

| Issue Title                                      | ID  | Open | Closed | All | Label       | Updated              |
|--|-----|------|--------|-----|-------------|----------------------|
| Detection of AonT/AoffT of activity/rest periods | #10 | 5    | 1      | 6   | enhancement | updated 6 days ago   |
| Analyse week days                                | #9  |      |        |     | enhancement | updated 2 months ago |
| AWD file header with space separated names       | #8  |      |        |     | bug         | updated 2 months ago |
| Move helper functions to a dedicated file        | #4  |      |        |     | enhancement | updated 2 months ago |
| Daylight Saving Time (DST)                       | #3  |      |        |     | bug         | updated 2 months ago |

▶ Report bugs

▶ Suggest improvements

▶ New functionalities

▶ Documentation...

# DEVELOPER'S CORNER

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

The screenshot displays a Kanban board interface. At the top, there is a navigation bar with a dropdown menu set to 'Development', a search bar labeled 'Search or filter results...', and buttons for 'Edit board', 'Add list', 'Add issues', and a window management icon. The board is divided into four columns: 'Backlog' (5 items), 'To Do' (1 item), 'Doing' (0 items), and 'Closed' (1 item). The 'Backlog' column contains four issues: 'AWD file header with space separated names #8' (bug), 'Analyse week days #9' (enhancement), 'Daylight Saving Time (DST) #3' (bug), and 'Move helper functions to a dedicated file #4' (enhancement). The 'To Do' column contains one issue: 'Detection of AonT/AoffT of activity/rest periods #10' (enhancement). The 'Closed' column contains one issue: 'Speed improvements #5' (enhancement).



## 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](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"

# DEVELOPER'S CORNER

The screenshot shows a GitLab Merge Request (MR) page. The breadcrumb trail is: GIGA-CRC In Vivo Imaging > Actigraphy > pyActigraphy > Merge Requests > !12. The MR is titled "Feature/docs" and is in an "Open" state, created 6 minutes ago by Grégory Hammad. The target branch is "develop".

Key elements on the page include:

- Buttons:** "Open", "Edit", and "Close merge request".
- Actions:** "Request to merge feature/docs into develop", "Open in Web IDE", "Check out branch", and a refresh dropdown.
- CI/CD:** "Pipeline #4 passed for e6e7a22a on feature/docs" with three green checkmarks.
- Approval:** "No Approval required".
- Merge Options:** "Merge" (checked), "Remove source branch", "Squash commits", and "Modify commit message".
- Instructions:** "You can merge this merge request manually using the command line".
- Reactions:** 0 thumbs up, 0 thumbs down, and 0 smiley faces.
- Summary:** Discussion: 0, Commits: 5, Pipelines: 4, Changes: 17.

On the right sidebar, the "Todo" section is visible, including fields for Assignee (None), Milestone (None), Time tracking (None), Labels (None), Lock merge request (Unlocked), 1 participant (Grégory Hammad), and Notifications (checked).

## DEVELOPER'S CORNER

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

## 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

## 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!

```
4 image: python:latest
5
6 # Change pip's cache directory to be inside the project
7 # only cache local items.
8 variables:
9   PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache"
10
11 # Pip's cache doesn't store the python packages
12 # https://pip.pypa.io/en/stable/reference/pip_install/#caching
13 # If you want to also cache the installed packages,
14 # them in a virtualenv and cache it as well.
15 cache:
16   paths:
17     - .cache/pip
18     - venv/
19
20 stages:
21   - build
22   - test
23   - deploy
24
25 before_script:
26   - python -V # Print out python version
27   - pip install virtualenv
28   - virtualenv venv
29   - source venv/bin/activate
30
31 run:
32   stage: build
33   script:
34     - pip install -e .
35
36 test:
37   stage: test
38   script:
39     - pip install pytest
40     - python -m pytest -vv --disable-pytest-warnings
41
42 pages:
43   stage: deploy
44   script:
45     - pip install sphinx sphinx-bootstrap-theme
46     - cd docs ; make html
47     - mv build/html/ ../public/
48 artifacts:
49   paths:
50     - public
51 only:
52   - feature/docs
```

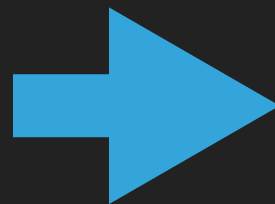
# DEVELOPER'S CORNER

## ▶ Automatic documentation

▶ Sphinx (<http://www.sphinx-doc.org/en/master/index.html>)

▶ Extract informations directly from the source code:

```
reader.py
170 def read_raw(input_path, reader_type, n_jobs=1, prefer=None, verbose=0):
171     """Reader function for multiple raw files.
172
173     Parameters
174     -----
175     input_path: str
176         Path to the files. Accept wild cards.
177         E.g. '/path/to/my/files/*.csv'
178     reader_type: str
179         Reader type.
180         Supported types: AWD (ActiWatch), MTN (MotionWatch8)
181         and RPX (Respironics)
182     n_jobs: int
183         Number of CPU to use for parallel reading
184     prefer: str
185         Soft hint to choose the default backendself.
186         Supported option:'processes', 'threads'.
187         See joblib package documentation for more info.
188         Default is None.
189     verbose: int
190         Display a progress meter if set to a value > 0.
191         Default is 0.
192
193     Returns
194     -----
195     raw : list
196         A list of instances of RawAWD, RawMTN or RawRPX
197     """
198
```



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.  
Created using Sphinx 1.7.8. [Back to top](#)

HTML

## 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 Markdown...)

## 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!!!



## 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"!

# DEVELOPER'S CORNER

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

The screenshot shows the GitLab interface for the 'pyActigraphy' repository. The left sidebar contains a navigation menu with 'Settings' highlighted and a red box around it, labeled '1.'. The main content area shows the 'Repository Settings' page with sections for 'Push Rules', 'Protected Branches', 'Protected Tags', and 'Deploy Keys'. The 'Deploy Keys' section has a 'Collapse' button highlighted with a red box and labeled '2.'. Below the 'Deploy Keys' section, there are input fields for 'Title' and 'Key'.

**BACK-UP SLIDES**

# INSTALL GIT ON WINDOWS

- ▶ <https://gitforwindows.org/>
- ▶

# GLOSSARY

| Terms     | Github       | Gitlab        |
|-----------|--------------|---------------|
| Directory | Repository   | Project       |
|           | Pull request | Merge request |
|           |              |               |

## ▶ Ressources

▶ <http://git-school.github.io/visualizing-git/>

▶