How to get started with Gitlab *(and GIT)*, an essential tool for research reproducibility ?

Tutorial session 1

Alizia Tarayoun

Thursday 9th November, 2023

University Grenoble Alpes, ISTerre

What we will talk about...

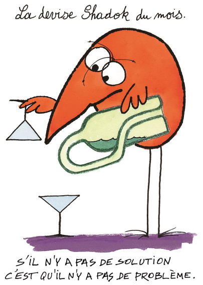
"This tutorial, intended for a novice audience, aims you to get started with the gitlab forge and a versioning system (git). We will see what is a forge and why it is today an essential tool for research reproducibility. At the end of this tutorial you will know how to use the main tools offered by the forge as well as the basic git commands to manage one or several project(s). Small demonstrations will be carried out from a simple one to illustrate the git commands to a more advanced example working with several branches on a code development project."

Objectives:

- understand what is (and what is not) a forge
- be convinced of its utility
- understand what is a version control system
- be autonomous for a (at least) basic usage
- be aware about some good practices

Organization

- Our practice through gricad-gitlab platform
- 1h15 it is short...
- Need to manipulate, experimented


La devise Shadok du mois.
S'IL N'Y A PAS DE SOLUTION C'EST QU'IL N'Y A PAS DE PROBLÈME.

→ All participants present this morning are registered in this group :
https://gricad-gitlab.univ-grenoble-alpes.fr/repro4research/sandbox
Feel free to do anything in this group (except removing someone else's work...)

Presentation (and more...) is available here: https://repro4research.gricad-pages.univ-grenoble-alpes.fr/demos/start/

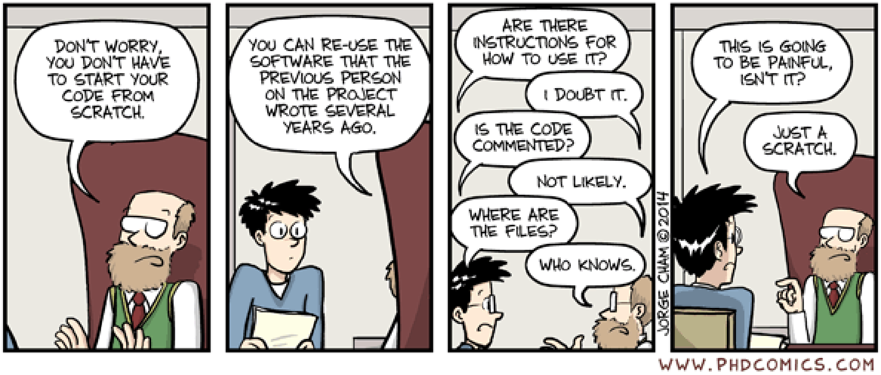
And everything will be accessible afterwards !

Outline

Outline

What is a forge: Definition

That was before...


DON'T WORRY, YOU DON'T HAVE TO START YOUR CODE FROM SCRATCH.
YOU CAN RE-USE THE SOFTWARE THAT THE PREVIOUS PERSON ON THE PROJECT WROTE SEVERAL YEARS AGO.
ARE THERE INSTRUCTIONS FOR HOW TO USE IT?
I DOUBT IT.
IS THE CODE COMMENTED?
NOT LIKELY.
WHERE ARE THE FILES?
WHO KNOWS.
THIS IS GOING TO BE PAINFUL, ISN'T IT?
JUST A SCRATCH.
JORGE CHAM © 2014
WWW.PHDCOMICS.COM

What is a forge: Definition

Today...

→ **Forges: Project management systems** (of collaborative developments)

Principle :

- Bring together users with different profiles (researchers, engineers, developers, coordinators etc.) around projects
- Provide a set of tools adapted to develop, manage, monitor, disseminate, promote projects

What is a forge: Definition

Today...

→ **Forges: Project management systems** (of collaborative developments)

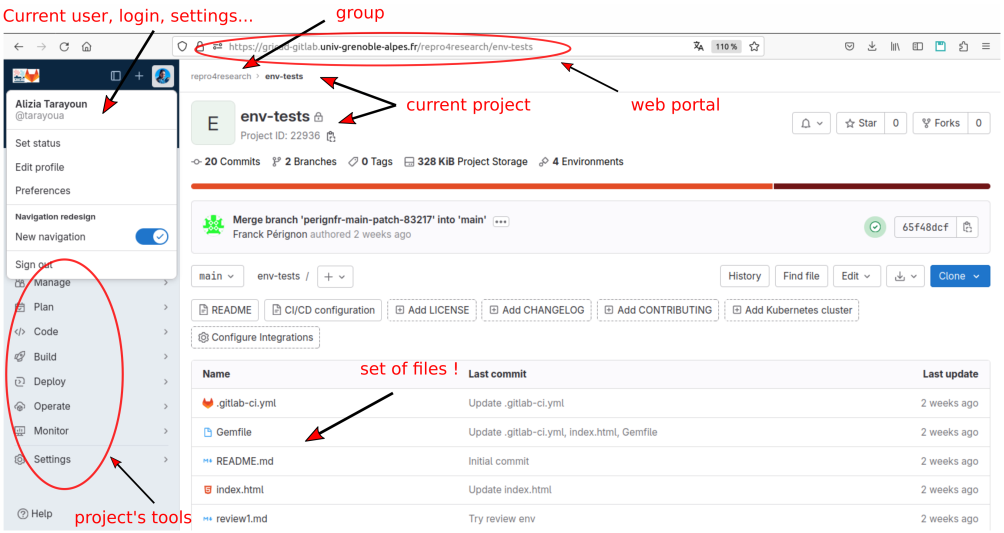
Principle :

- Bring together users with different profiles (researchers, engineers, developers, coordinators etc.) around projects
- Provide a set of tools adapted to develop, manage, monitor, disseminate, promote projects

In practice :

- a website
- users (login) associated with projects
- configurable tools via the web interface
- different levels of rights on tools and projects (visibility, writing . . .)

What is a forge: Example of the gricad-gitlab platform (ESR Grenoble)



Name	Last commit	Last update
.gitlab-ci.yml	Update .gitlab-ci.yml	2 weeks ago
Gemfile	Update .gitlab-ci.yml, index.html, Gemfile	2 weeks ago
README.md	Initial commit	2 weeks ago
index.html	Update index.html	2 weeks ago
review1.md	Try review env	2 weeks ago

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Structuring : groups, sub-groups, projects and authenticated users
  - possible collaborations with anyone
  - fine management of access rights and utilization for all tools
  - management of visibility of groups, projects, files

→ See demo1

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Structuring : groups, sub-groups, projects and authenticated users
  - possible collaborations with anyone
  - fine management of access rights and utilization for all tools
  - management of visibility of groups, projects, files

  → See demo1

- Planning, scheduling, task and problem manager (issues)
  - Build **communities** (contributors, users, students ...)
  - **Follow**, organize the team's work
  - **Collect**, log and track issues
  - **Manage** user requests and responses, tickets
  - **Share** information, exchange space, wiki

  → See demo1

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Version Control System
  Practice of keeping/maintaining all versions of a set of files.
  - Centralize activity
  - Archive and track changes, keep history, developments
  - Allow/facilitate/manage contributions (merge-requests...)

  → See part dedicated on VCS section:vcs and GIT section:git in this tutorial

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Version Control System
  Practice of keeping/maintaining all versions of a set of files.
  - Centralize activity
  - Archive and track changes, keep history, developments
  - Allow/facilitate/manage contributions (merge-requests...)

  → See part dedicated on VCS section:vcs and GIT section:git in this tutorial

- Sharing, management of documents and files
  - **Backup** (and therefore possible restoration), archiving
  - **Online edition**

  → See demo2

What is a forge: Review (non-exhaustive) of available tools and functionalities

- CI/CD Continuous Integration and Deployment
  Practice to check systematically and automatically the impact of any modification of sources.

What is a forge: Review (non-exhaustive) of available tools and functionalities

- CI/CD Continuous Integration and Deployment

  Practice to check systematically and automatically the impact of any modification of sources.
  - Allows you to create software from its source code (compilation, automatic testing, quality assurance, distribution of deliverables)
  - Production of broadcastable and usable versions
  - Generation, storage and provision of images (ready-to-use code, notebooks, etc.), "container registries"

→ See Tutorial Session 2 - Advanced Gitlab - of F. Pérignon

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Gitlab Pages : publication, hosting and maintenance of websites
    - creation and deployment of websites
    - production of documentation
    - related to CI

  → See Tutorial Session 2 - Advanced Gitlab - of F. Pérignon

What is a forge: Review (non-exhaustive) of available tools and functionalities

- Gitlab Pages : publication, hosting and maintenance of websites
  - creation and deployment of websites
  - production of documentation
  - related to CI

  → See Tutorial Session 2 - Advanced Gitlab - of F. Pérignon

- Interaction with Software Heritage

  → See Presentation of B. Chauvet

For more details about general features, see https://about.gitlab.com/features/

What is a forge: Which usages ?

- Collaborations around codes developments (softwares, applications)
  → At the origin of forge creation
  → Became an indispensable tool and widely used by the software developer community
- Collaborative (or not) redaction of scientific paper, thesis manuscript etc.
- Teaching (courses, training material, etc.)
- Generation, publication and maintenance of websites
- Share space of files, data
  ⚠a forge is not a data repository like https://recherche.data.gouv.fr/fr, a NextCloud or DropBox
  It is not intended to save large volume of data
- etc.

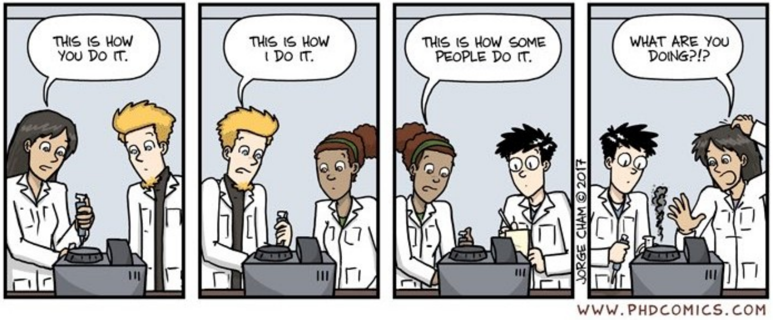
What is a forge: Why use a forge ?

- access to all tools via a single web portal
- well-integrated tools, configurable for each project, relatively intuitive use
- does not require any prior installation
- multi-site, multi-user access
- a tool suitable for:
    - varying degrees of participation: development, supervision, testing, distribution, etc.
    - very different contexts of use: research, industrial collaboration, teaching, etc.

What is a forge: Good practice and reproducibility

- Promote collaborative work, make life easier for everyone, collaborators AND users (present and future)
- Provide and promote quality, shareable and reusable products
- Be in a "reproducible research" approach
  → Grenoble Network around Reproducible Research :
  https://reproducibility.gricad-pages.univ-grenoble-alpes.fr/web/

The forges are at the heart of these processes



What is a forge: Which forge(s) to use ?

The reference GitHub ≈ 73 millions users in 2021, 100 millions of projects, redeem by Microsoft in 2019.

Equivalent alternatives :

Bitbucket
https://bitbucket.org

GitLab
https://about.gitlab.com

What is a forge: Which forge(s) to use ?

External forges to ESR, commercial or community

🙂

- very functional
- integrate many tools
- few constraints
- widely used, stable, very regularly updated and improved

☹

- commercial, non-academic
- uncontrolled hosting
  (RGPD respect, European regulations ?)
- sometimes payable
- indefinite availability period (Google code...)

What is a forge: Which forge(s) to use ?

→ A good solution: the self-hosted "ESR" platforms (39 available forges [1] in structures or laboratories)

In practice

- The choice will depend on your project, the people involved, the habits of your community...
- Possible and relatively simple to use multiple platforms
  - similar tools and uses
  - simple to transfer project

[1] Forges de l'Enseignement supérieur et de la Recherche - Définition, usages, limitations rencontrées et analyse des besoins, D. Le Berre, J.Y. Jeannas, R. Di Cosmo, F. Pellegrini, 2023, hal-04098702v2

What is a forge: Which forge(s) to use ?

→ Today: focus on GitLab ...

- A software developed by Gitlab Inc company
- An "open source" free of charge distribution (Gitlab CE) and a payable proprietary version (Gitlab EE) offering additional features
- Numerous deployments (websites) : self-hosted forges
  - https://about.gitlab.com : Gitlab Inc forge
  - 37 Gitlab instances recorded in the ESR

... and more particularly on the gricad-gitlab platform.

Outline

The gricad-gitlab platform

For today's demos we will use gricad-gitlab

- Platform created in 2017
- Academic platform intended for all establishments in the Grenoble "ESR" community.
- Hosted and administered by the UAR GRICAD
- Today: nearly 9,000 users, more than 14,000 projects, more than 2,500 groups

https://gricad-gitlab.univ-grenoble-alpes.fr

First steps with gricad-gitlab

The basics for getting started with a gitlab-type platform:

- creation/management of a user account
- creation/management of projects and groups
- visibility and rights

Prerequisites ?

- A web browser, a connection
- And that's all !

First steps with gricad-gitlab: access and connection

Who : anyone !

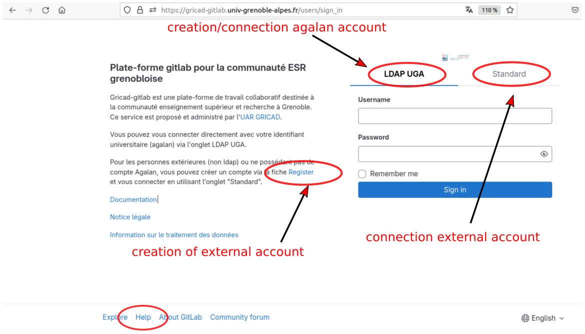
How ?

- ESR Grenoble : *agalan* account ⇒ full access to all tools
  Creation/Connection : tab **LDAP UGA**
- 'external' account, more limited ⇒ unauthorized creation of groups or personal projects
  To create an account : link **Register**
  Connection : tab **Standard**

Identification: a username and a unique email (impossible to have 2 accounts with the same email)

First steps with gricad-gitlab: access and connection

Access link : https://gricad-gitlab.univ-grenoble-alpes.fr



https://gricad-gitlab.univ-grenoble-alpes.fr/users/sign_in

Plate-forme gitlab pour la communauté ESR grenobloise

Gricad-gitlab est une plate-forme de travail collaboratif destinée à la communauté enseignement supérieur et recherche à Grenoble. Ce service est proposé et administré par l'UAR GRICAD.

Vous pouvez vous connecter directement avec votre identifiant universitaire (agalan) via l'onglet LDAP UGA.

Pour les personnes extérieures (non ldap) ou ne possédant pas de compte Agalan, vous pouvez créer un compte via la fiche Register et vous connecter en utilisant l'onglet "Standard".

Documentation

Notice légale

Information sur le traitement des données

→ Try it, connect !

First steps with gricad-gitlab: access and connection

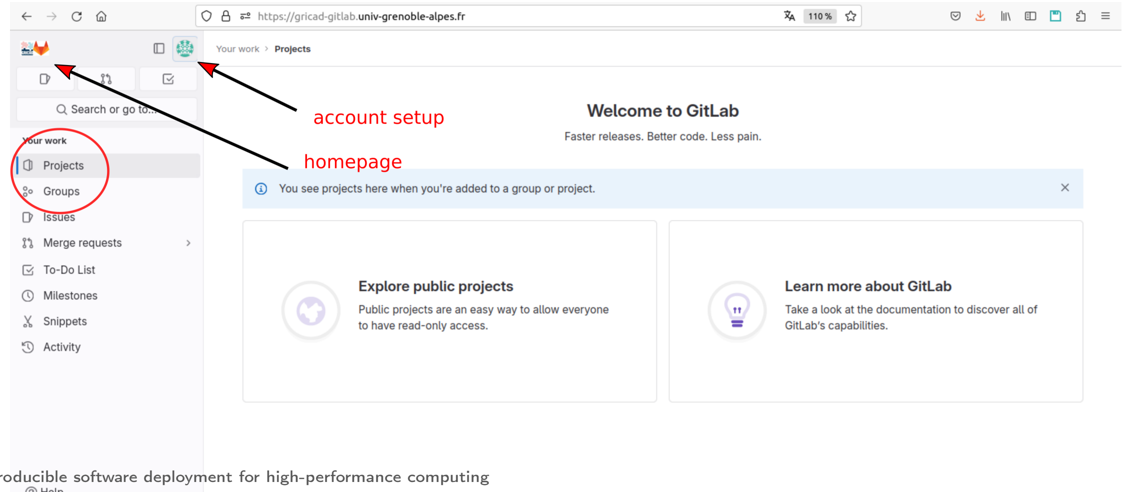
→ DEMO 1:

- Login (agalan vs external account)
- User Settings: profile
- User Settings: SSH Keys
- User Settings: Notifications
- Help

Access link : https://gricad-gitlab.univ-grenoble-alpes.fr

First steps with gricad-gitlab: DEMO 1: Dashboard

- Account setup
- Access to projects and groups
- Todo-list, notifications ...



Welcome to GitLab

Faster releases. Better code. Less pain.

You see projects here when you're added to a group or project.

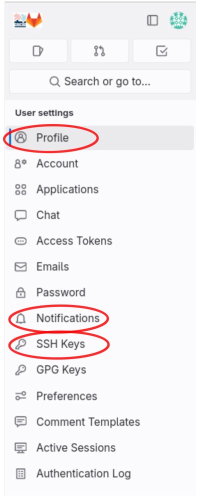
Explore public projects

Public projects are an easy way to allow everyone to have read-only access.

Learn more about GitLab

Take a look at the documentation to discover all of GitLab's capabilities.

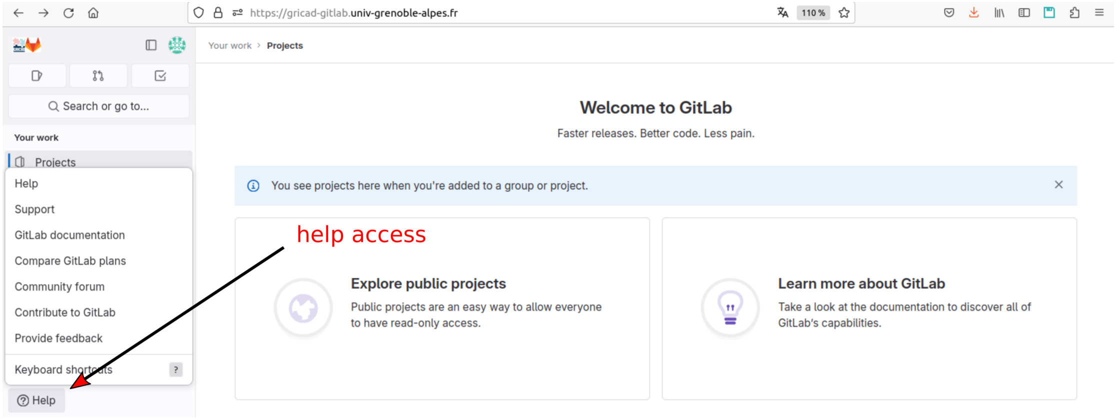
First steps with gricad-gitlab: DEMO 1: Settings Menu



Recommended first steps :

- **Profile** : check/complete the different fields
- **SSH keys** : drop an ssh key (see supplementary materials ssh_key)
- **Notifications** : control the notification level
  Visualize the different possibilities and set a default mode
  (→ “disabled” to avoid unwanted sending of emails)

First steps with gricad-gitlab: DEMO 1: where to find help



Welcome to GitLab

Faster releases. Better code. Less pain.

You see projects here when you're added to a group or project.

Explore public projects

Public projects are an easy way to allow everyone to have read-only access.

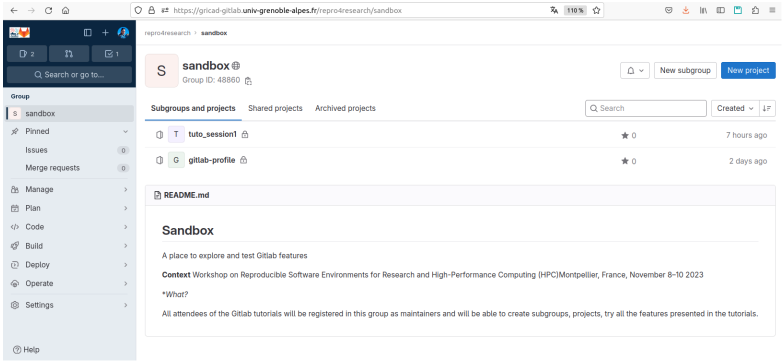
Learn more about GitLab

Take a look at the documentation to discover all of GitLab's capabilities.

First steps with gricad-gitlab: Groups

Defining as:

- Set of projects or subgroups
- Associated with a set of users
- Manage rights/permissions by default



sandbox
Group ID: 48860

tuto_session1	★ 0	7 hours ago
gitlab-profile	★ 0	2 days ago

README.md

Sandbox

A place to explore and test Gitlab features

Context Workshop on Reproducible Software Environments for Research and High-Performance Computing (HPC)Montpellier, France, November 8–10 2023

**What?*

All attendees of the Gitlab tutorials will be registered in this group as maintainers and will be able to create subgroups, projects, try all the features presented in the tutorials.

First steps with gricad-gitlab: Group creation

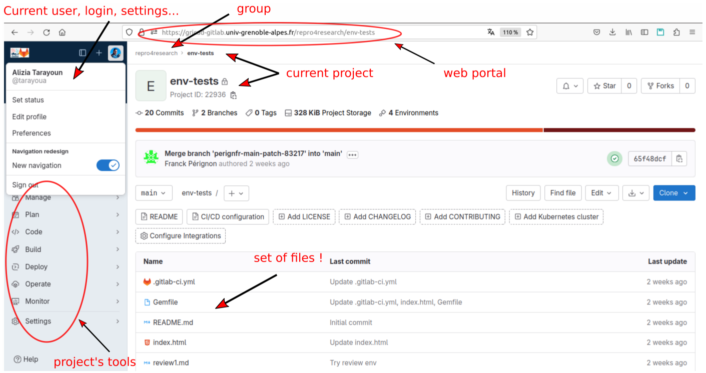
Information needed for creation:

- a name and description
- a level of **visibility**
    - private: visible only to project members (who must therefore be added by the admin)
    - internal: visible to any connected user (possibility of requesting to join)
    - public: visible to anyone

First steps with gricad-gitlab: Projects

A "space" for:

- host, save, share files
- manage a set of participants, with individually adjustable rights
- manage/configure tools



https://gricad-gitlab.univ-grenoble-alpes.fr/repro4research/env-tests

repro4research > env-tests

env-tests
Project ID: 22936

20 Commits · 2 Branches · 0 Tags · 328 KiB Project Storage · 4 Environments

Merge branch 'perignfr-main-patch-83217' into 'main'
Franck Pérignon authored 2 weeks ago · 65f48dcf

Alizia Tarayoun @tarayoua

Name	Last commit	Last update
.gitlab-ci.yml	Update .gitlab-ci.yml	2 weeks ago
Gemfile	Update .gitlab-ci.yml, index.html, Gemfile	2 weeks ago
README.md	Initial commit	2 weeks ago
index.html	Update index.html	2 weeks ago
review1.md	Try review env	2 weeks ago

First steps with gricad-gitlab: Projects

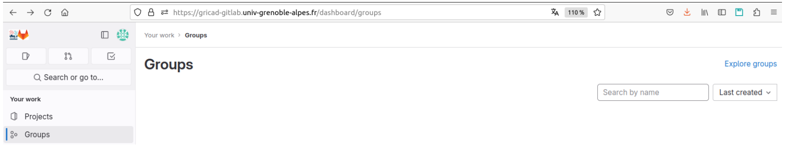
→ DEMO 2:

- join the sandbox group
- create a project in sandbox
- manage rights/permissions
- operations directly on the platform (add files, edit, save modifications...)

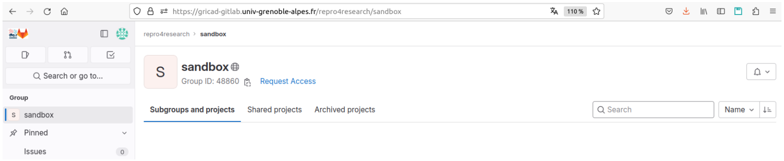
Access link : https://gricad-gitlab.univ-grenoble-alpes.fr/repro4research/sandbox (need to be accepted)

First steps with gricad-gitlab: DEMO 2: Create a project

⚠: You will see the projects only if you registered before to this tutorial.
(recall: an external user can create project only in an already existing group)
If not, need to:
1- In the link *Explore groups*, need to search for "sandbox" associated with the "repro4research" group

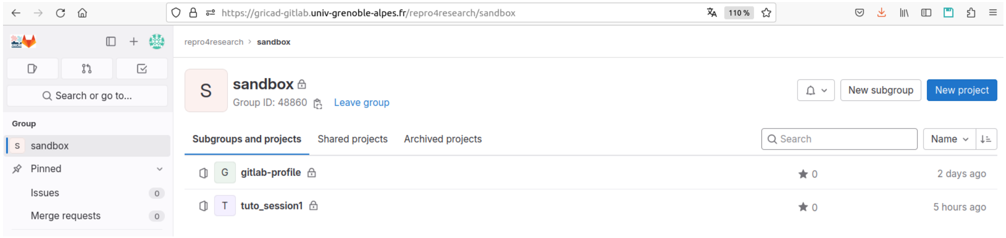


2 - Ask to join the group with the link *Request Access*



First steps with gricad-gitlab: DEMO 2: Create a project

Where? Menu *Projects* of the dashboard : projects list and tab *New project*



First steps with gricad-gitlab: DEMO 2: Create a project

Information needed for creation:

- a location (**namespace** ≈ folder), a name, a description
  - Your '**username**', personal projects (thesis manuscript, forks etc.), (**not available for external accounts**)

    https://gricad-gitlab.univ-grenoble-alpes.fr/your_login/project_name
  - A **group** or a **subgroup**

    https://gricad-gitlab.univ-grenoble-alpes.fr/group_name/project_name
- a level of **visibility**
  - private: visible only to project members (who must therefore be added by the admin)
  - internal: visible to any connected user (possibility of requesting to join the project)
  - public: visible to anyone

⚠projects inherit group visibility: for example, a private group can only contain private projects

First steps with gricad-gitlab: DEMO 2: Create a project

Roles and permissions:

Roles : guest, reporter, developper, maintainer, owner

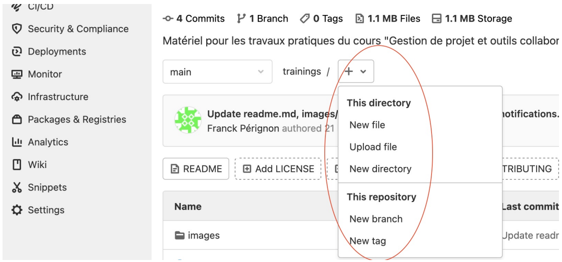
Details : https://gricad-gitlab.univ-grenoble-alpes.fr/help/user/permissions

- Creator of a group or project: automatically “owner”
- ⚠a user inherits his rights in a project from the group
- A user can be part of a project without belonging to the group

First steps with gricad-gitlab: DEMO 2: Create a project

Editing directly on the platform:

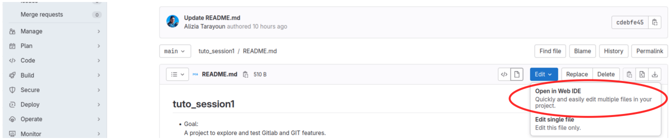
- the web editor -> creation, loading of files, creation of branches, etc.



Matériel pour les travaux pratiques du cours "Gestion de projet et outils collabor

Update readme.md, images/ ... notifications.
Franck Pérignon authored 21

- Web IDE -> online management of all repository files



Update README.md
Alizia Tarayoun authored 10 hours ago

tuto_session1

- Goal:
  A project to explore and test Gitlab and GIT features.

First steps with gricad-gitlab: DEMO 2: Create a project

Editing on the platform and markdown:
Markdown is a very simple language to learn, read and write that allows to format text for a web page. On Gitlab it is possible to use an extended version of Markdown (gitlab flavored markdown) to write comments, issues, help files, etc. A bit of documentation:

- https://docs.gitlab.com/ee/user/markdown.html
- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Advices/best practices :

- Learn the basics of Markdown
- At least one README.md per project
  (summary description of the project, content, help, useful links, etc.)
  → better readability of your project !
- When a text file is edited with markdown syntax, it will be displayed formatted (and therefore more readable and attractive)

First steps with gricad-gitlab: Group and Project: some more information

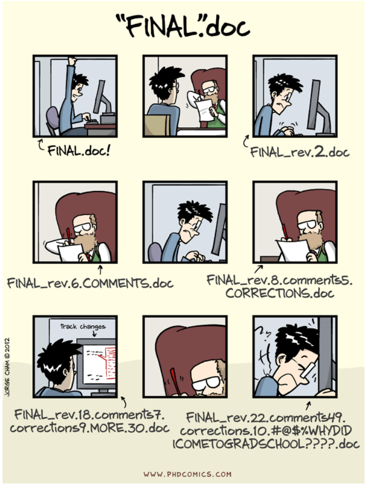
In terms of good practices:

- Take the time to correctly configure the list of members, their rights, roles, expiration dates of participation, etc.
- To group managers: only add people who are likely to participate in all the group's projects
- Organize projects into thematic groups (adds visibility)
- Pay attention to the naming of projects (impact on visibility, project reference)

Outline

In which case ?

A project with participants who collaborate for instance...


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

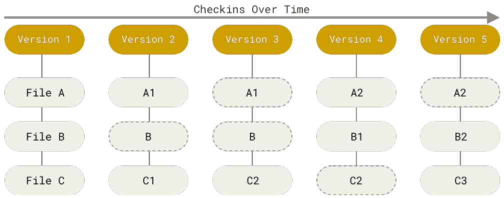
Potential needs/issues:

- simultaneous work on the same files
- need to incorporate other people's changes
- need to distribute to users an up-to-date version of the files
- need to preserve history (for possible rollbacks)
- etc.

Definition

→ Solution: Use a versioning system

A version control is a system that records changes to a file or set of files over time so that it is possible to recall specific versions later



A tool which is at the heart of the forges, and which is their main interest !
Gitlab project ≈ directories and files (**repository**) + a built-in *version manager*

Definition

Two main functions:

- allows simultaneous work of several people on a set of files:
  - synchronization of changes, automatic merging of files
  - detection and resolution (more or less automatic . . .) of conflicts
- a history management of the project:
  - access to any archived version
  - information about who made a change, when, where, why...
  - automatic notifications to participants

In short, an essential tool !

Which VCS to use ?

Different Version Control Systems (not exhaustive):

- subversion (svn, 2000): subversion.apache.org
- git (2005): git-scm.com
- mercurial (2005): mercurial.selenic.com
- bazaar (2005): bazaar.canonical.com

Today GIT overcome the others (see rhodecode.com), especially due to patforms such as github or gitlab.

Outline

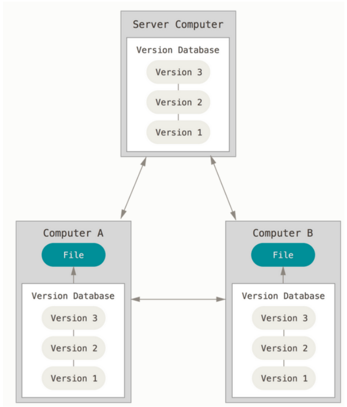
GIT definition

GIT is a distributed version control system

- when check out, users fully mirror the repository, including its full history
- every clone is really a full backup of all the data
- nearly every operation is local i.e. more possibility to work offline (access to complete history, differences between commit etc.)

Two levels of work:

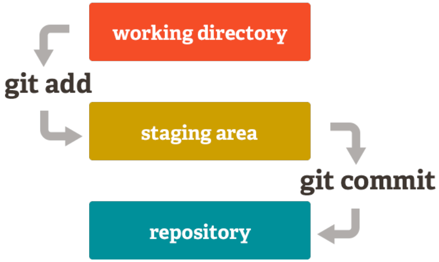
- local to each "repository" with the classic functionalities of a version manager
- remote: with the possibility of synchronizing the repository with other, called remote/distant



A GIT project

- three main components of a GIT project:
  - the repository (.git): data and meta-data concerning all "versioned" directories and files
  - the working directory: or working tree, files currently working on
  - the staging area: or index, where commits are prepared

→ Commit: state of a repository, a version, a snapshot of all files that it is recorded



Components of a GIT project

GIT: States of files

- **untracked**: on your working directory
- **modified**: the file is changed but not committed
- **staged**: a file has been marked to go into the next commit
- **committed**: the file is safely stored in the local database (i.e. the .git repository)

To get information about the state of all files in the working directory: `$ git status`

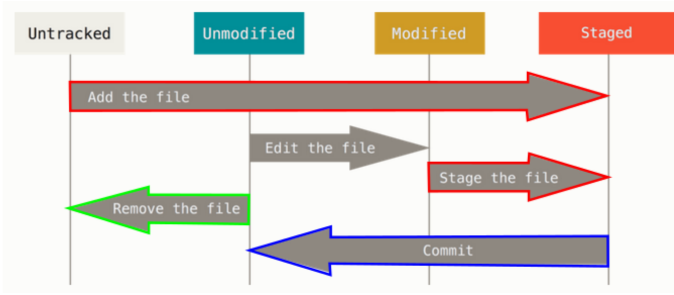
To staged files:

`$ git add <filename>` or `$ git add -A`

To commit files in the staged area:

`$ git commit` or

`$ git commit -m "<message>"`



Git configuration

Before the practice...

Git is generally used via the command line (terminal, PowerShell).
The use is identical under Windows, mac or Linux:

```
$ git <command> <argument>
```

For an overview of the different commands: `$ git help`

To obtain the documentation for a specific command: `$ git help <command>`

→ Installation, see: https://git-scm.com/downloads

Git configuration

Mandatory settings necessary for the first use of git:

```
$ git config --global user.name "Alizia Tarayoun"
```

```
$ git config --global user.email "alizia.tarayoun@univ-grenoble-alpes.fr"
```

Optional options:

choose a default editor:

```
$ git config --global core.editor vim
```

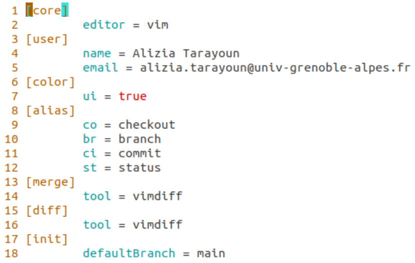
or

```
$ git config --global core.editor emacs
```

define aliases by editing the .gitconfig file:

```
$ git config --global alias.co checkout
```

→ Everything is saved in a .gitconfig file at the root of your account.



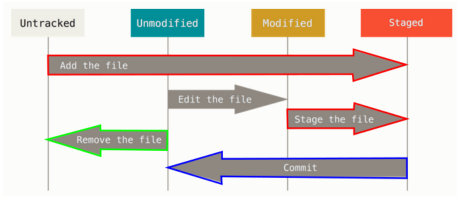
```
[core]
        editor = vim
[user]
        name = Alizia Tarayoun
        email = alizia.tarayoun@univ-grenoble-alpes.fr
[color]
        ui = true
[alias]
        co = checkout
        br = branch
        ci = commit
        st = status
[merge]
        tool = vimdiff
[diff]
        tool = vimdiff
[init]
        defaultBranch = main
```

Example of a .gitconfig file

A GIT project

→ DEMO 3:

- Create a git project in local
- Illustrate the different status
- Save modifications (i.e. commit)
- View history
- Go back to a specific version



A GIT project: DEMO 3: create a repository

To use git, you first need a working directory, which will contain, among other things, the database and the current version of your repository.

Creation (init) of an empty repository:

```
$ mkdir WorkingDir; cd WorkingDir
$ git init
```

Or copy (clone) an existing repository:

```
$ git clone <repo_address>
# examples:
# git clone /path/to/local_repo WorkingDir
# git clone username@server:/path/to/repository WorkingDir
```

"repo_address" represents a “remote” repository: another directory on your machine, a git repository on another server accessible via the network...

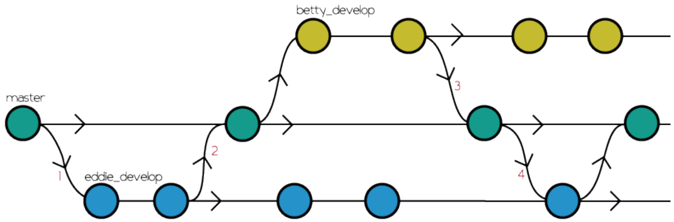
A GIT project: DEMO 3: States of file

Simplified life cycle of a project will be as follows:

- Extraction from the database of a “snapshot” of the project, i.e. a version of all the files which will constitute the working directory (git clone) or initialization of a local working directory (git init)
- Modification(s) of files in the working directory
- Staging (index) of these changes, which means that they are candidates for saving in the next version
- Validation (commit): creation of a new snapshot, saved in the database, from the index information

Notion of branches

Nearly every VCS has some form of branching support. Branching means you diverge from the main line of development and continue to do work without impacting the original branch.
In git the current default branch name is main.

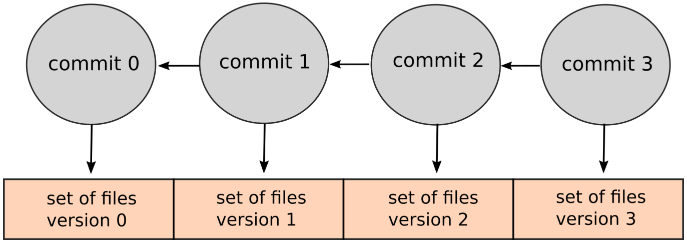


Example of connections between branches

Git: branches

Reminders of operating principles

Commit: a link (a pointer) to a **snapshot** of the state of all files. Git stacks commits one after the other that builds the project history.

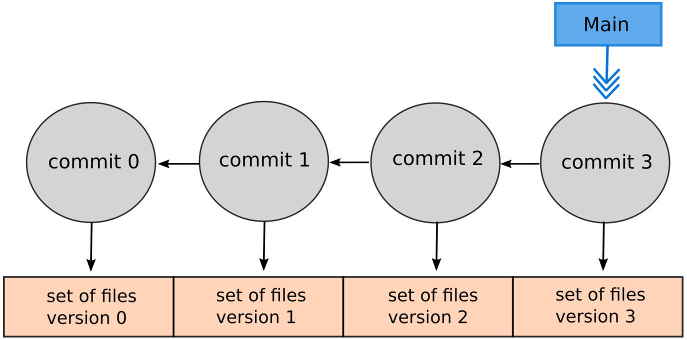


Git: branches

Branch: a pointer to a particular commit + history

A default branch: main.

Possible extraction of any commit from the branch ⇒ access to the version of the files of this commit



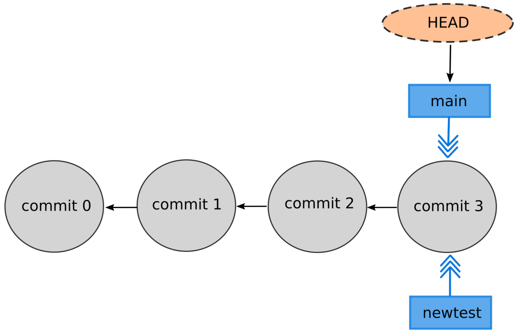
Git: branches

New branch: divergence from the main line (new functionality, bug fix...)

- To create a new branch:

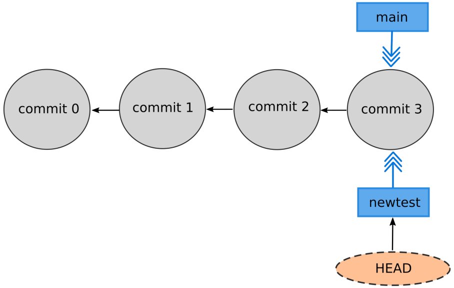
```
$ git branch newtest
```

- Identification of the current branch: the HEAD pointer



Git: branches

To switch to an existing branch: `$ git checkout newtest`



⚠Note: you will not be able to change branch if there are modified files on your current branch (check with `$git status`).

Solutions:

- Work on another "working directory" (potentially with git clone <url> -b <branch_name> --single-branch)
- Use git stash (see supplementary materials git_stash)

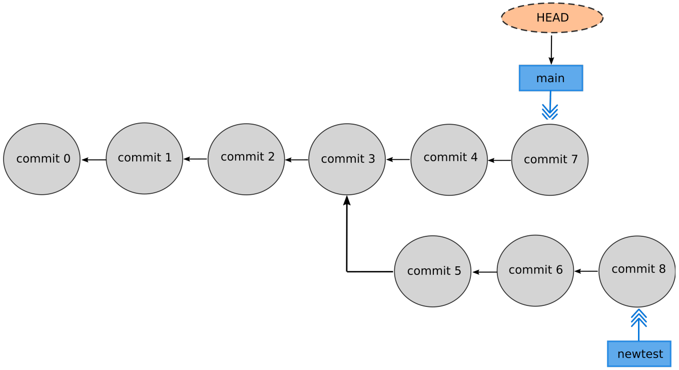
Git: branches

Independent evolution of the branches, until a possible merge...

To integrate a branch (source) into a branch (target):

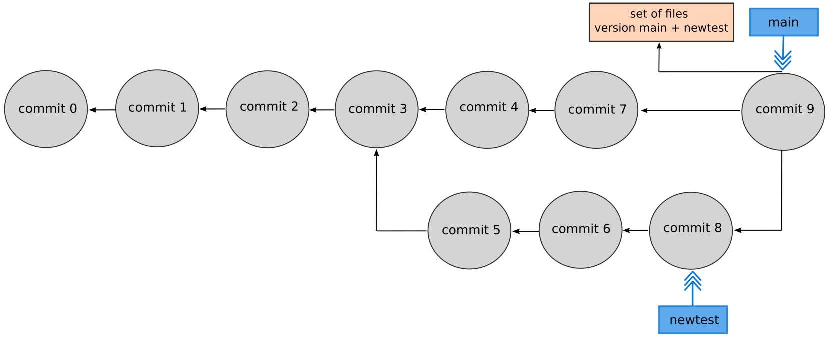
Switch to target branch: `$ git checkout main`

Merge : `$ git merge newtest`



Git: branches

Each branch can then continue to evolve independently...



Git merge: manage conflicts

During the merge, everything does not necessarily always go very well... For instance if the same part of the same file is changed differently in the two branches, Git won't be able to merge them cleanly and end up with a merge conflict.

```
(base) tarayoua@ist-156-58:~$ git merge branch_merge
Auto-merging file_hello
CONFLICT (content): Merge conflict in file_hello
Automatic merge failed; fix conflicts and then commit the result.
```

git adds markers around conflict areas:

```
(base) tarayoua@ist-156-58:~$ cat file_hello
<<<<<<< HEAD
bonjour
var=10.6789
=======
hello
var=9
>>>>>>> branch_merge
end file
```

Git merge: manage conflicts

During the merge, everything does not necessarily always go very well…

$\longrightarrow$To correct problems, need to:

- resolve conflict manually or with a dedicated tool (`$ git mergetool`)
- put the corrected files on the staged area
- validate the changes (i.e. commit)

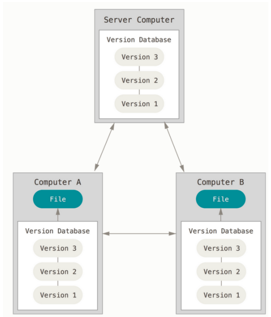
Git: branches

→ DEMO 4:

- create a new branch in the same repository
- merge branch

→ From now, work only in local.

Need to use a forge to work efficiently with collaborators

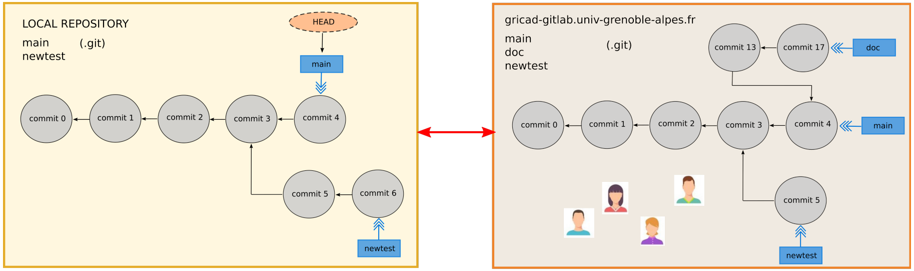


Outline

Remote repository, principle

Main steps:

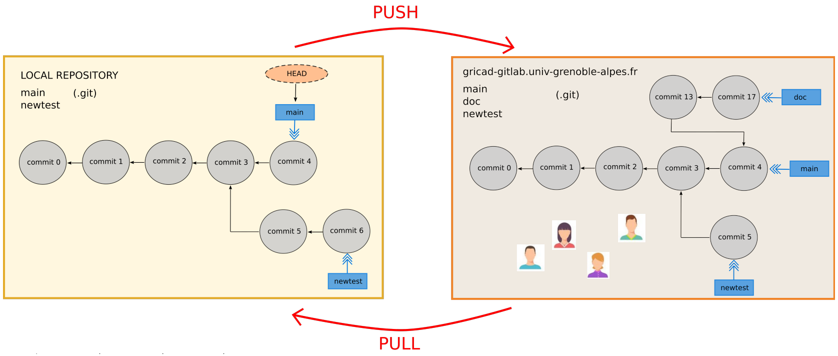
- connection between repositories: add or copy (git clone) a remote repository
- developments in each repository (independently)



Remote repository, principle

Main steps:

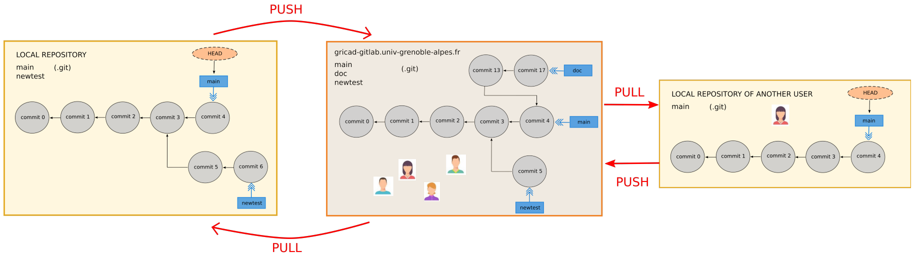
- connexion between repositories: add or copy (`git clone`) a remote repository
- developments in each repository (independently)
- integration of modifications from the remote repository (`git pull`)
- transfer from the local modifications into the remote repository (`git push`)



Remote repository, principle

Main steps:

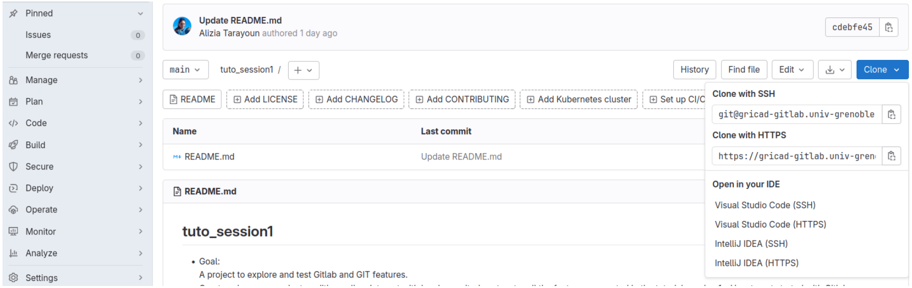
- connexion between repositories: add or copy (git clone) a remote repository
- developments in each repository (independently)
- integration of modifications from the remote repository (git pull)
- transfer of the local modifications into the remote (git push)
- push/pull from potential other users



Remote repository, interaction with ssh or https

Two protocoles are available in order to "connect" the local to the remote repository and exchange files between your machine and gitlab.

- "https": authentification by login and password
- "ssh": authentification with an ssh key (that needs to be added on the gitlab account profile)



```
git@gricad-gitlab.univ-grenoble
https://gricad-gitlab.univ-gren
```

tuto_session1

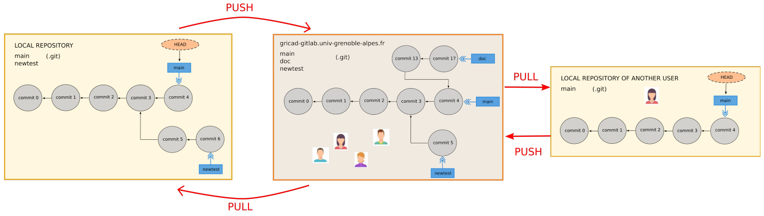
- Goal:
  A project to explore and test Gitlab and GIT features.

Working with a remote repository

→ DEMO 5:

- Connect the local repository to the remote one
- Interact with the remote repository (push/pull)
- Add other repository

link remote repository: a project in : https://gricad-gitlab.univ-grenoble-alpes.fr/repro4research/sandbox/



Working with a remote repository: DEMO 5

Recall on adding a remote repository

Case 1: from scratch

Creation of an empty repository

```
$ git init
```

Connection with a remote repo

```
$ git add remote <name_remote> <repository_address>
```

Case 2: copy

```
$ git clone git@gricad-gitlab.univ-grenoble-alpes.fr:pathproject/project.git myproject
# automatic link with the remote repository, locally named 'origin'
```

Note: it is possible to create several remote repositories with the following command:

```
$ git init --bare <remote_name.git>
```

Working with a remote repository: DEMO 5

Some useful commands working with remotes:

To list the current remotes:

```
$ git remote -v
```

You can refer to the remote repository by his name instead of the whole url.

To locally rename a remote repository:

```
$ git remote rename <current_name> <new_name>
```

To locally delete a reference to a remote (and all remote-tracking branches):

```
$ git remote rm <local_name>
```

Working with a remote repository: DEMO 5

To update your working directory regarding the remote repository:

```
$ git pull <remote_repo_name> <branch_name>
```

Note: to avoid at every pull/push/merge to specify the name of the remote repository and the name of the local branch, you can track a local branch by a remote branch (automatically done when cloning a project for the main branch, the main branch tracks origin/main):

```
$ git branch --set-upstream-to <remote_repo_name/branch_name> <branch_name>
```

git pull = `$ git fetch <remote_repo_name>` +

```
$ git merge <remote_repo_name/current_branch_name>
```

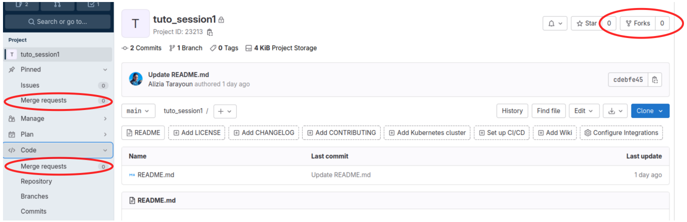
To push your modifications on the remote repository:

```
$ git push <remote_repo_name> <branch_name>
```

Working with a remote repository: DEMO 5: merge-requests

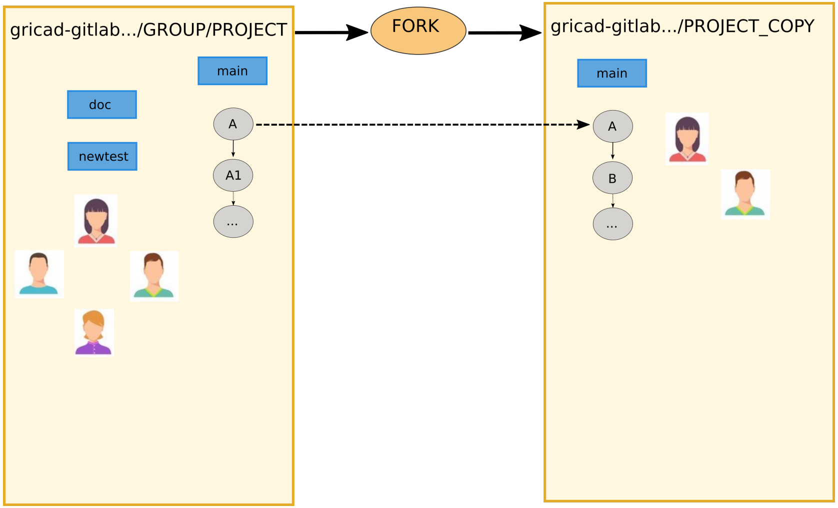
In terms of **good practices** for properly managing parallel developments (working on several simultaneous branches) a good solution is to use Gitlab functionality of the merge-requests (pull requests under github).

Merge-request: submission of a request to merge one branch into another, which will in particular lead to a review process by other developers
Fork: duplication of a project (and therefore copy of the git repository)



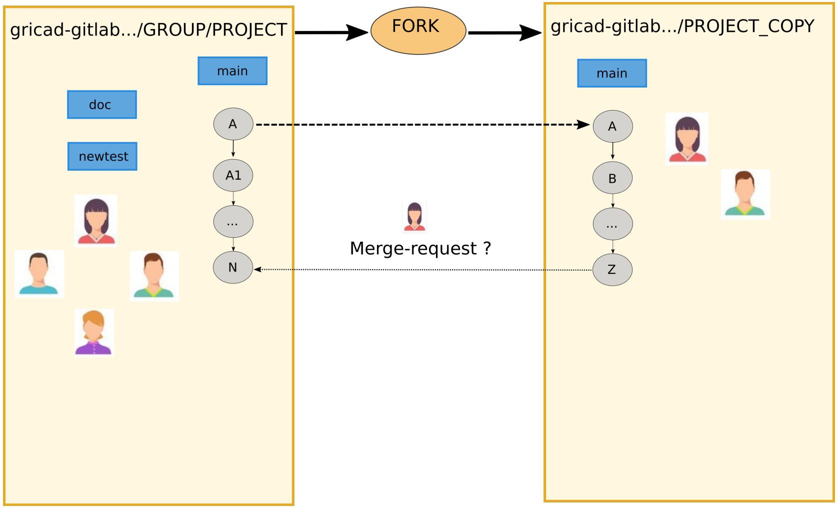
Working with a remote repository: DEMO 5: merge-requests

Copy of the gitlab project to another namespace, Fork ⟶ creation of a new project.



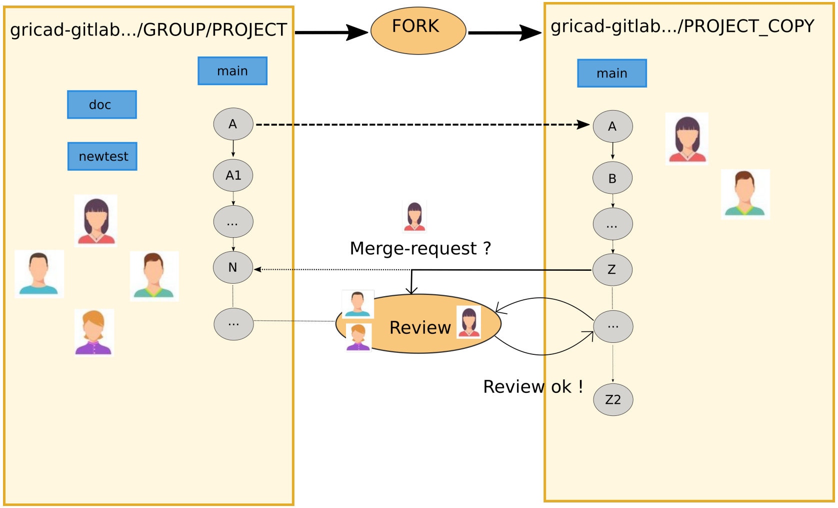
Working with a remote repository: DEMO 5: merge-requests

Submitting a merge request



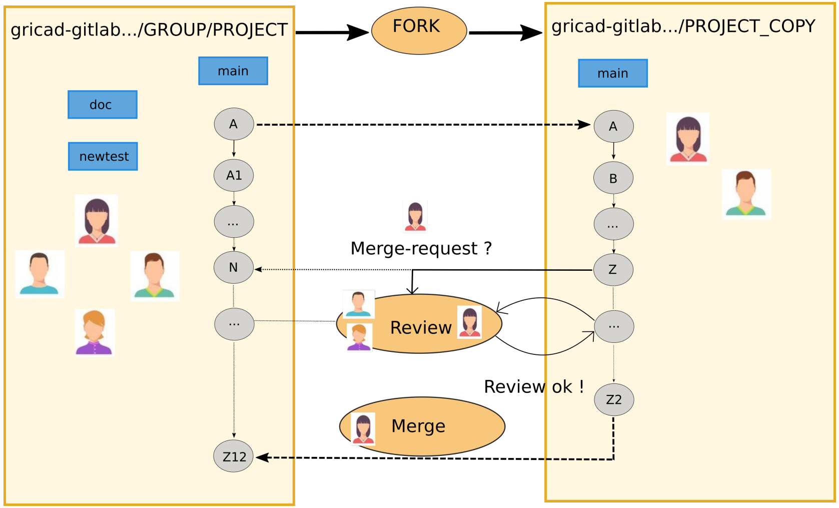
Working with a remote repository: DEMO 5: merge-requests

Review, until an acceptable and mergable solution (a commit) is obtained



Working with a remote repository: DEMO 5: merge-requests

Merge of the fork branch to the main project, possible closure of the merge request



Outline

Gricad-gitlab platform - Illustration of use

The SEISCOPE project

Develops and applies high-resolution geophysical imaging methods through seismic waveform inversion to characterize subsurface/crustal physical properties.

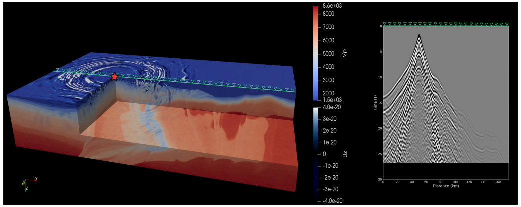
2 main codes for 3D modeling and inversion:

- TOYxDAC_TIME: anisotropic visco-acoustic based on finite difference method (MPI+MPI+OpenMP)
- SEM46: anisotropic viscoelastic (and/or acoustic) based on spectral finite elements method (MPI+MPI)

3 to 5 main contributors per code

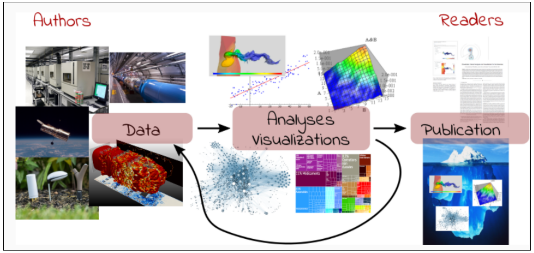
→ **need a forge (+ version control system)**

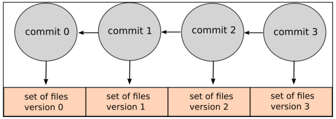
- use for code development
  example_code_SEM46
- use for collaborative redaction
  example_publication



Outline

Conclusion - reproducibility research





Top figure from https://github.com/alegrand/SMPE/blob/master/lectures/talk_23_06_01_Gricad.pdf

Outline

Git - complements, some other useful commands

Branches:

To create a new branch and switch to it at the same time:

```
$ git checkout -b <branch_name>
```

To delete a branch in local:

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

To delete a branch in a remote repository:

```
$ git push <remote_name> --delete <branch_name>
```

History:

To check the history of a branch:

`$ git log` or `$ git log -n <number>` to display only the nth last number of commits

To display modifications of a file through commits:

```
$ git log -p <filename>
```

Git - complements, some other useful commands

Display differences:

To display differences between the working tree and the last commit:

`$ git diff` or `$ git diff <filename>`

To see differences between what it is staged and the last commit:

```
$ git diff --staged <filename>
```

To see difference between commits:

```
$ git diff <commit_id1> <commit_id2>
```

To compare branches in local (at HEAD state):

`$ git diff <branch1>..<branch2>` It will show all the differences that branch2 has that are not in branch1

To compare a branch in local with the remote (at HEAD state):

```
$ git diff main remotes/origin/main
```

Git - complements, some other useful commands

To correct/ go back:

The **checkout** command:

- To go back in a specific commit with only reading rights:

```
$ git checkout <commit_id>
```

- To cancel modification on a file since a specific commit:

```
$ git checkout <commit_id> <filename>
```

need to commit it to keep it that way

The **revert** command to undo what was done on a commit (create a new commit):

`$ git revert <commit_id>` or `$ git revert <commit_id> <filename>`

The **reset** command:

- To reset the current branch HEAD to <commit_id>:

```
$ git reset <commit_id>
```

modify the history !

A lot of possibilities (git-scm.com)

Git - complements, some useful commands

GIT won't let the user switch between branches if there are modified files in the current branch.

```
(base) tarayoua@ist-156-58:~$ git checkout newtest
error: Your local changes to the following files would be overwritten by checkout:
hello_file
Please commit your changes or stash them before you switch branches.
Aborting
```

Solutions:

1) Commit the changes or 2) Create another working tree

3) Use stashing → `$ git help stash`

To store the working tree: `$ git stash save <choose_name> <-u>`

To list the stash: `$ git stash list`

To show what is in the stash: `$ git stash show <stash_name> -p`

To apply the stash: `$ git stash apply <stash_name>`

To delete the stash: `$ git stash drop <stash_name>`

Workflow git

git workflow: method of managing the git repository(s) common to all project participants.

Why choose a workflow?

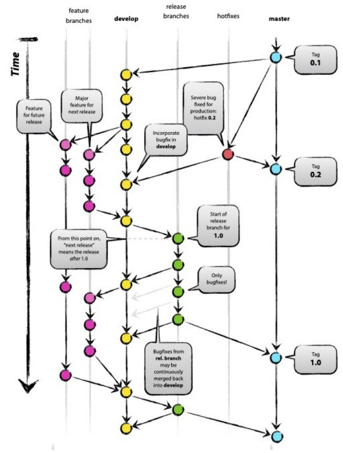
- Different speakers in several contexts, with their own habits
- Hence complicated or even chaotic management and therefore probably ineffective
- Risk of wasting time!

Which workflow to choose?

Lots of possibilities and no single answer...

- Centralized workflow: everyone works on the same branch.
  - requires a small team and a lot of discussion
  - difficult to manage releases, unstable versions . . .
- Git workflow
- GitHub flow
- OneFlow
- ...

Workflow git: one example Gitflow



feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

Figure from here

A little bit of documentation here

Working with a remote repository: SSH key

Why?

SSH keys: a necessary tool to manage secure connections between a local machine and the Gitlab server.

How?

1. Generate an SSH key pair (or check if you don't already have one).
2. Copy your public key to the Gitlab server.

1 - Generate an SSH key pair

⚠ This step is only necessary if you do not already have a set of keys !

To check if you already have a key: check the contents of the .ssh/ directory on your machine.

If it contains two files "id_rsa" and "id_rsa.pub" for instance you can proceed to the next step.

Otherwise, keys must be generated via the ssh-keygen command:

```
$ ssh-keygen -t rsa
```

After this step, two files are generated:

- .ssh/id_rsa: your private and strictly personal key
- .ssh/id_rsa.pub: the public key corresponding to your private key

Working with a remote repository: SSH key

2 - Copy your public key

Transfer your public key to the Gitlab server.

Find the SSH keys menu on the "Preferences" page of your account on Gitlab. Copy the contents of the ".ssh/id_rsa.pub" file into the associated field.

For more details: https://docs.gitlab.com/ee/user/ssh.html