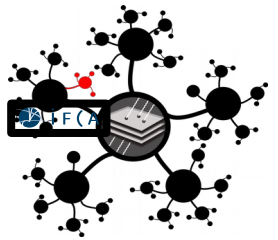


A gentle introduction to git

[gɪt]*

or

How to develop software with other folks

Jordi Duarte-Campderrós

EXCELENCIA MARÍA DE MAEZTU

IFCA Instituto de Física de Cantabria

CERN

CSIC Consejo Superior de Investigaciones Científicas

UC Universidad de Cantabria

* https://www.youtube.com/watch?v=4XpnKHJAok8%3Ft%3D1m30s

The problem

- An usual working day *algorithm* for a [PUT HERE YOUR PROFESSION]
  - **Create**

OR/AND

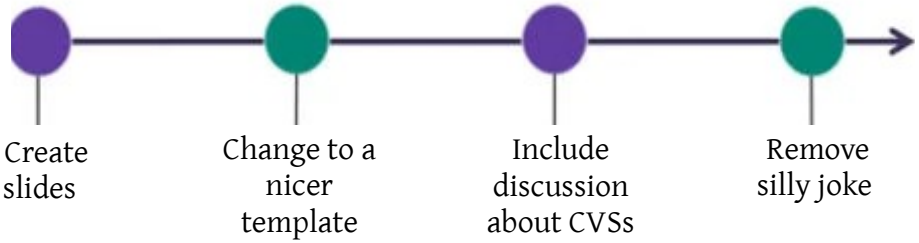
  - **Modify**

➜ **SAVE it** (and again... and again... and again...)

The problem

➡ **SAVE it** (and again... and again... and again...)

- When did you make some change
- Why did you make that change
- How did you make that change



- Keeping a detailed **TRACK HISTORY** of your important stuff

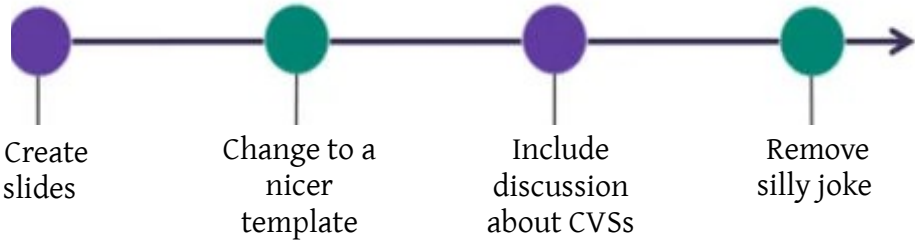
- Why is it important to keep history?
  - Well...
    "Those who cannot remember the past are condemned to repeat it."

    G. Santayana

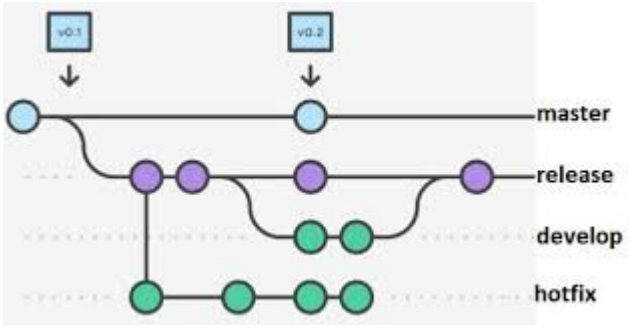
The solution

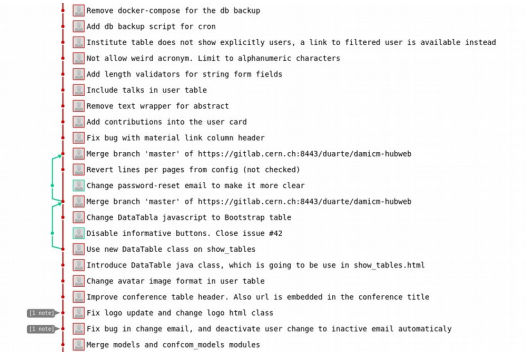
- ➡ Version Control Systems

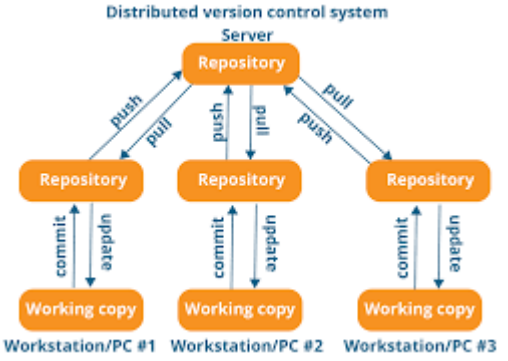
  - When you made some change
  - Why you made that change
  - How you made that change



- Keeping a detailed **TRACK HISTORY** of your important stuff

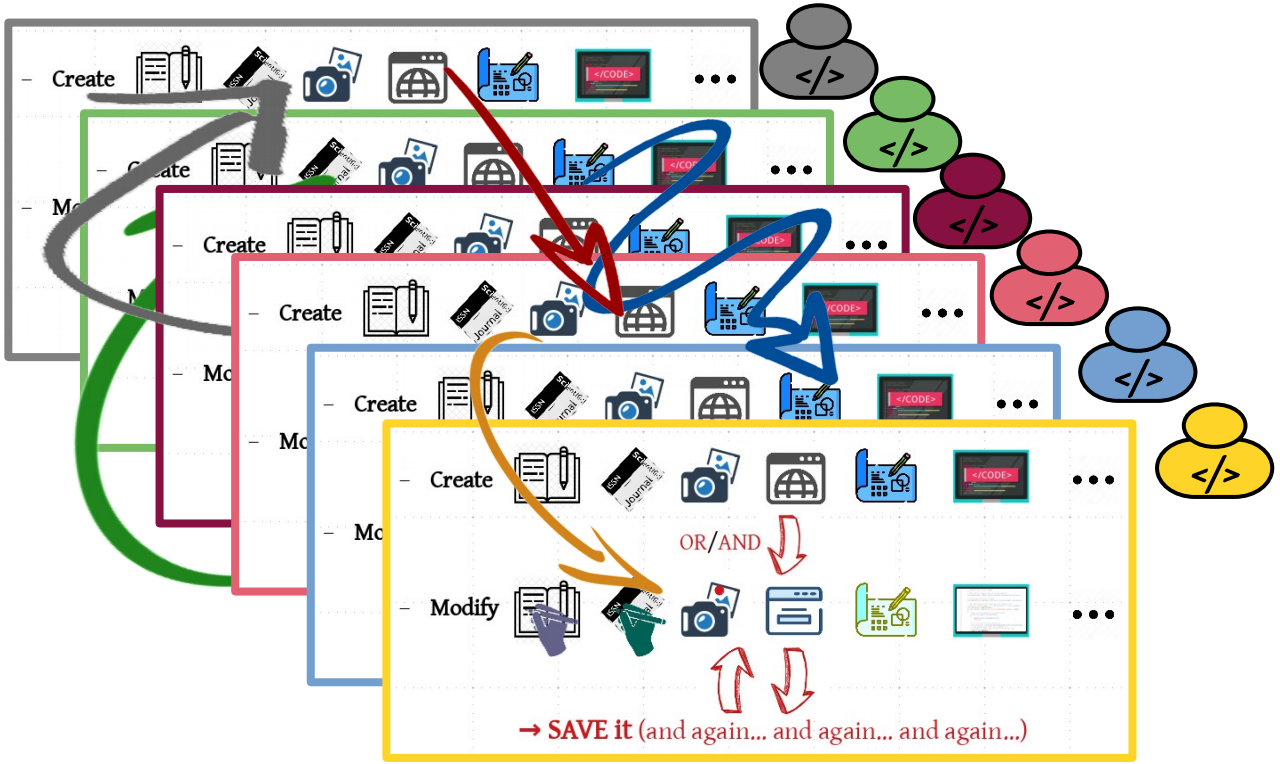






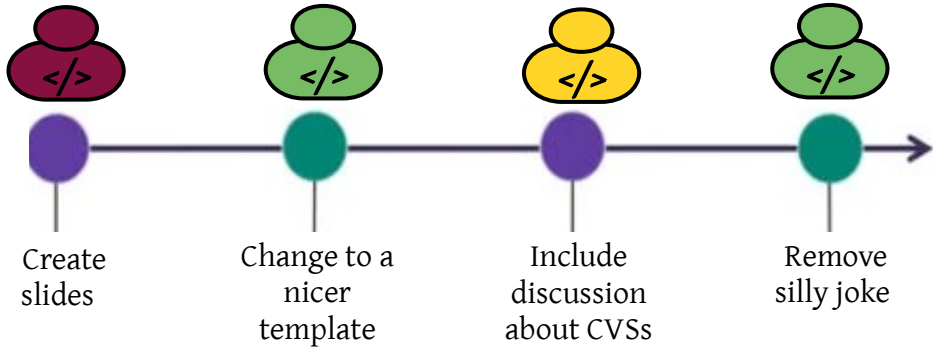
The real problem: collaborative work

- An usual working day *algorithm* for a **bunch** of [PUT HERE YOUR PROFESSIONs]



More than ever: the solution

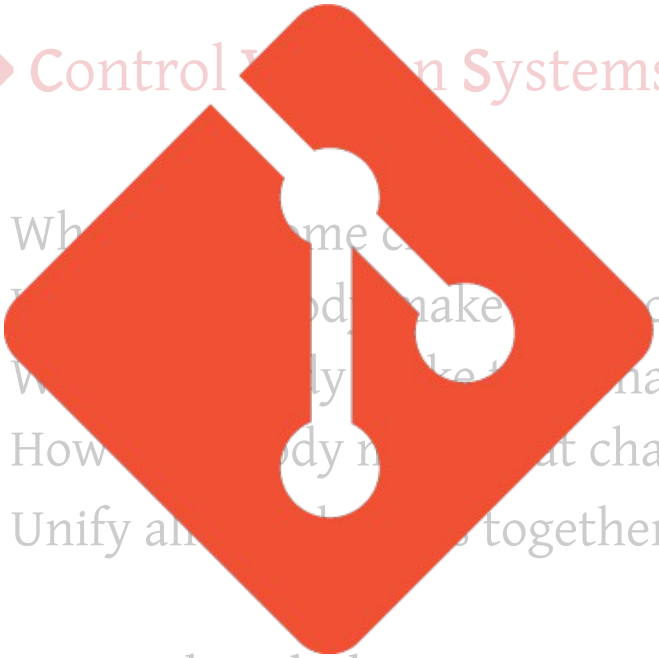
- ➡ Version Control Systems
  - Who made some change
  - When somebody made some change
  - Why somebody made that change
  - How somebody made that change
  - Unify all the changes together → **MERGE**



- Keeping a detailed **TRACK HISTORY** of all important stuff

→ Control Version Systems

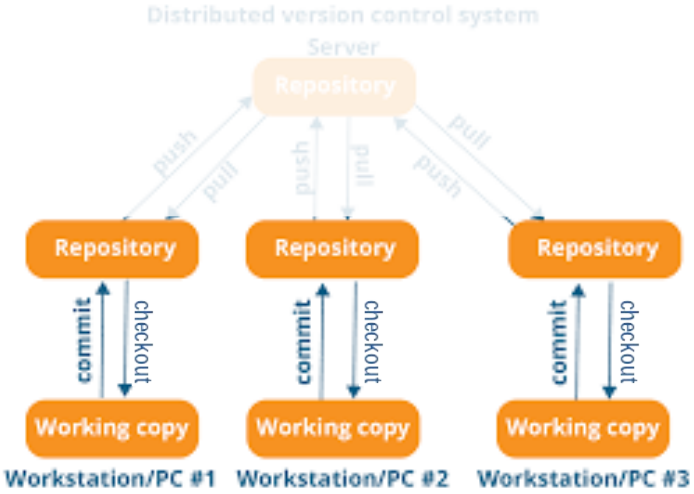
- Who made changes
- When somebody made the change
- Why somebody made that change
- How somebody made that change
- Unify all changes together → MERGE



- Keeping a detailed **TRACK HISTORY** of all important stuff

GIT: a distributed VCS

- Git is **local**
  - Allows version control in your computer
    - Just need to install git
    - Initialize the repository
    - Decide what do you want to control
      - Place them under the same directory
    - Add those files you want to version control



Exercise

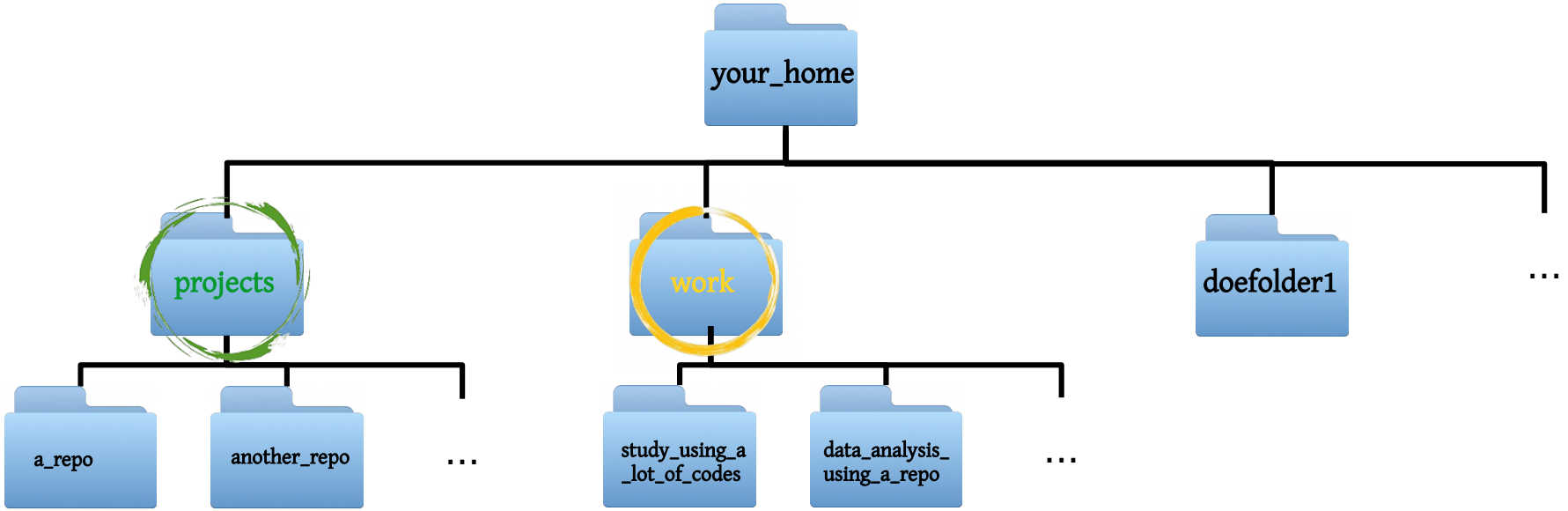
```
$ _
```

Interlude 1: code development best practice

DON'T FLIGHT A PLANE WHERE YOU BUILT IT!!

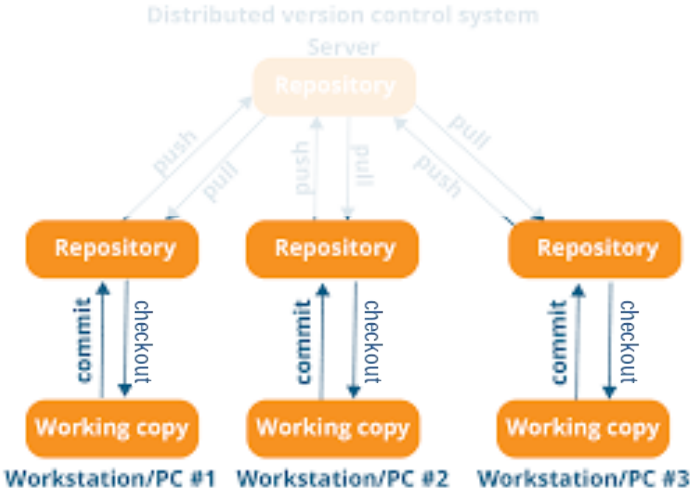
Interlude 1: code development best practice

- **NEVER** execute programs in the same folder than your development/coding area
  - Create a developing area where to put your source code
  - Create a working independent area where to execute your codes



Git: a distributed VCS

- Git is **local**
  - Allows version control in your computer
    - Just need to install git
    - Initialize the repository
    - Decide what do you want to control
      - Place them under the same directory
    - Add those files you want to version control



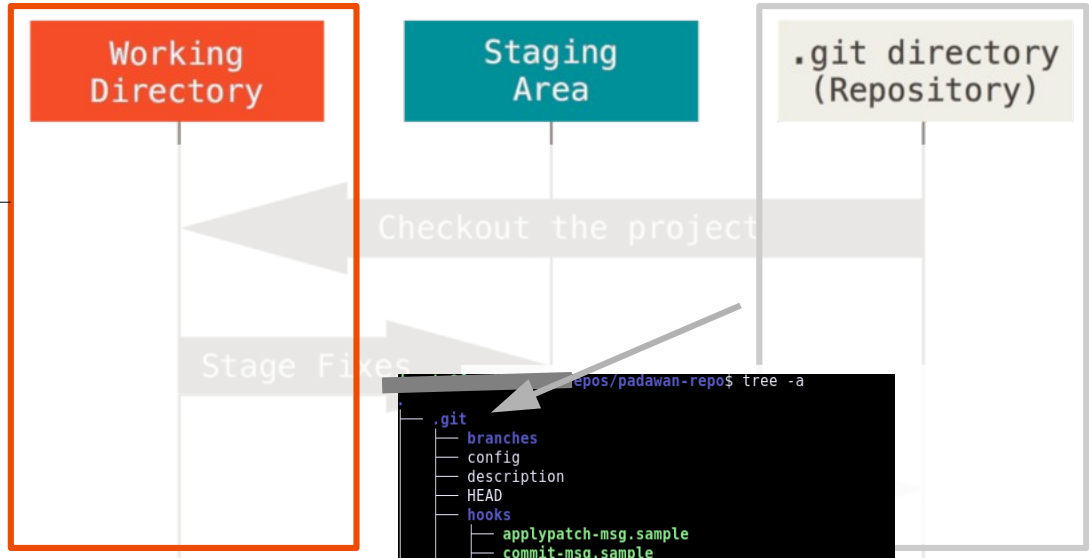
Exercise

```
$ git init padawan-repo
$ cd padawan-repo
$ echo 'May the git be with you' > keepaneyeonit.txt
$ touch idontcarethis.dat
$ git add keepaneyeonit.txt
```

Usual workflow: the areas

Exercise

```
$ git init padawan-repo
$ cd padawan-repo
```



- Creates the infrastructure to start versioning your project inside the **padawan-repo** folder (Working Directory)
  - **.git** folder: the repository, actually, a compressed database with all the files history, configuration, etc...

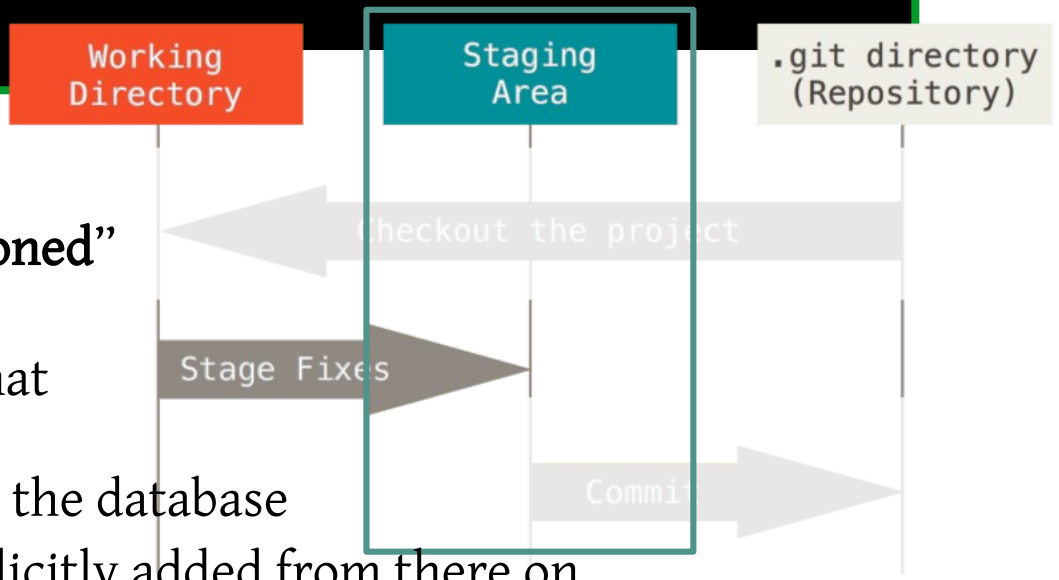
```
$ git config --global color.ui auto
$ git config --global user.name "First Last Name"
$ git config --global user.email "myemail@example.com"
```

```
epos/padawan-repo$ tree -a
.
└── .git
    ├── branches
    ├── config
    ├── description
    ├── HEAD
    ├── hooks
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── fsmonitor-watchman.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    │   ├── prepare-commit-msg.sample
    │   ├── pre-push.sample
    │   ├── pre-rebase.sample
    │   ├── pre-receive.sample
    │   └── update.sample
    ├── index
    ├── info
    │   └── exclude
    ├── objects
    │   ├── 2e
    │   │   └── ade3d82612c81ec9d871638c2514334cac138b
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```

Usual workflow

Exercise

```
$ echo 'May the git be with you' > keepaneyeonit.txt
$ touch idontcarethis.dat
$ git add keepaneyonit.txt
```



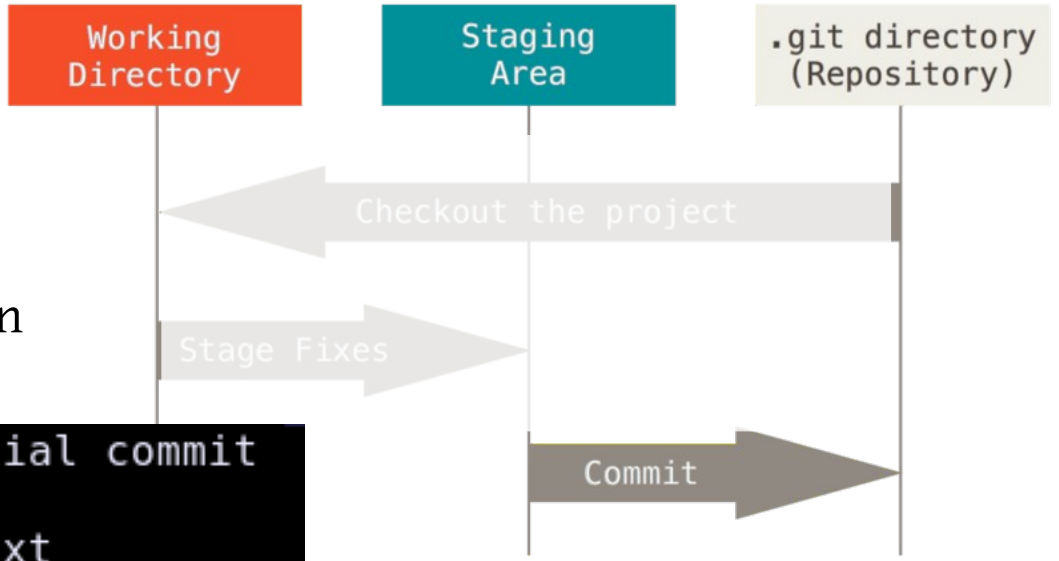
- Add the file as a "**file to be control-versioned**"
  - The file is sent to the STAGING area, a place containing the info about what it go into next commit, a kind of 'waiting room' before being store in the database
- Any modification in the file must be explicitly added from there on in order to store the changes

Usual workflow

Exercise

```
$ git commit -m "Initial commit"
```

- Creates a snapshot of the project as it is right now, and **store it permanently** in the database (.git directory, i.e. the repository)
- The snapshot is uniquely **identified** by an integer hash key, **SHA-1**



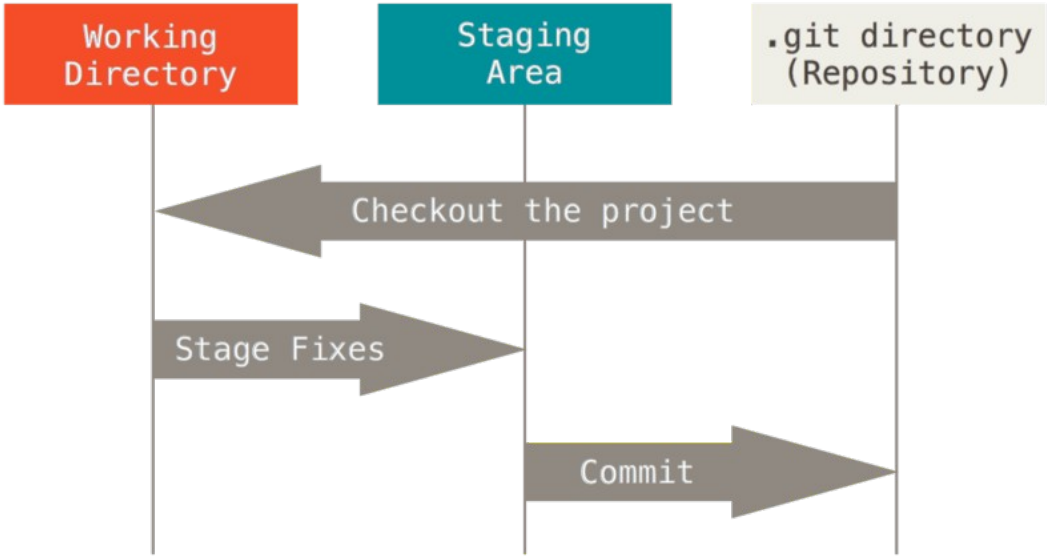

```
[master (root-commit) 444b677] Initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 keepaneyeonit.txt
```

```
commit 444b6772294167b5ab38d036a599a05c0723c481
```

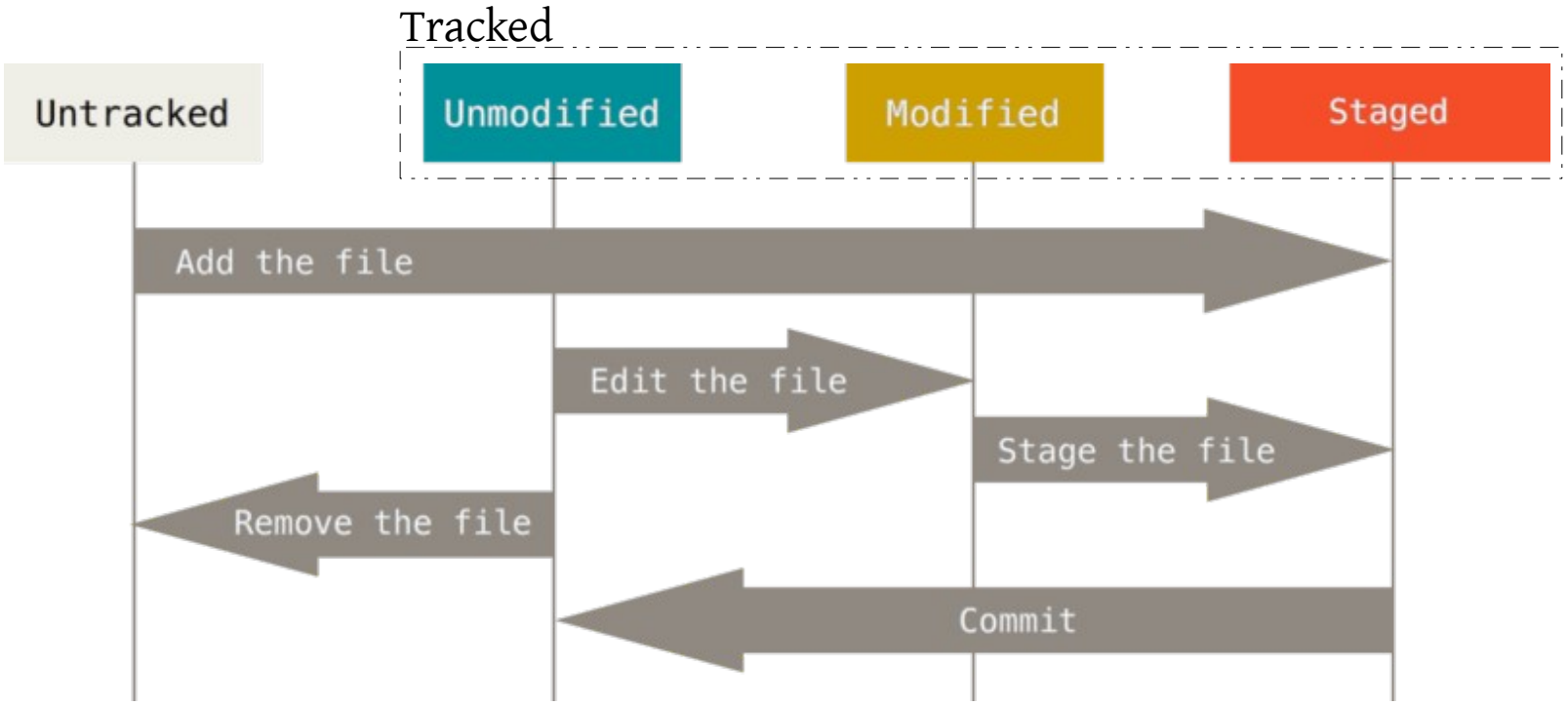
(Create-)Modify-Save in Git

The three states of a file

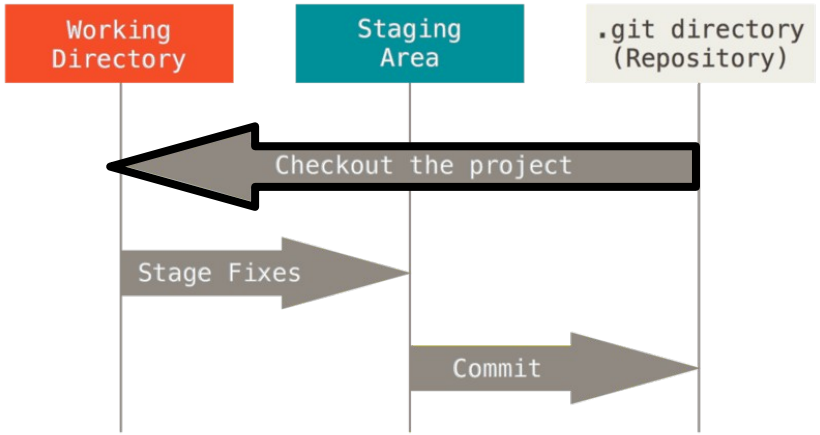
- A file being controlled by git (`git add`) will be in any of the following states:
  - **Modified:** the file contains some changes not stored in the local database
  - **Staged:** the modified file is marked to go into the next commit snapshot
  - **Committed/Unmodified:** the data is already stored in the local database



The lifecycle of a file



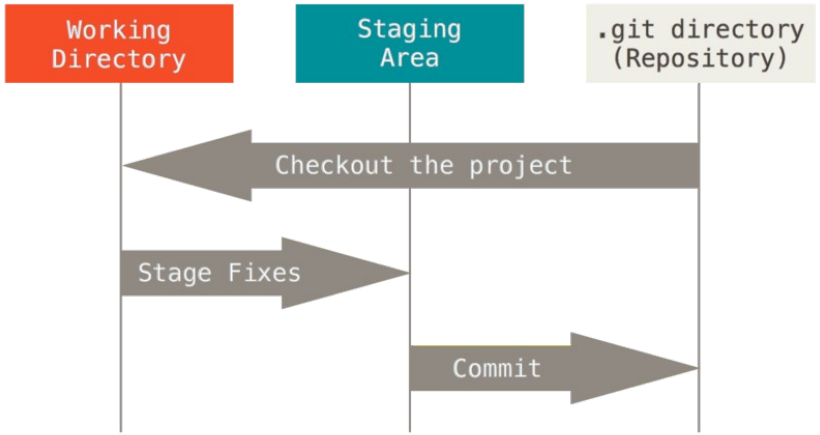
Usual workflow (revisit'd)




```
$ git checkout <version>
```

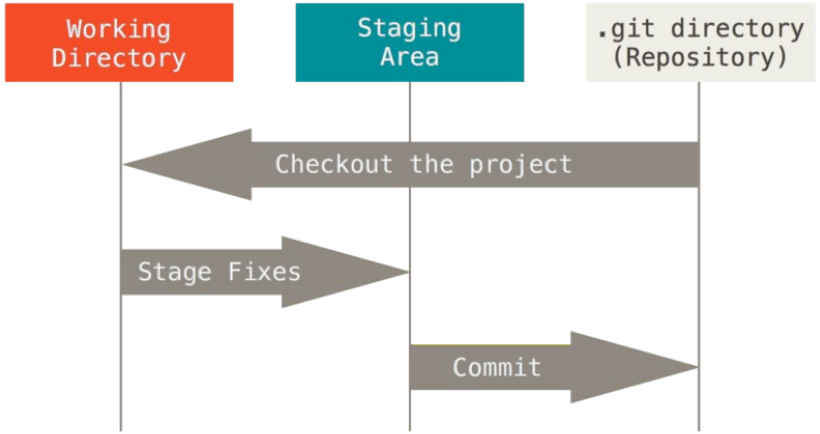
1. **Start working from a particular snapshot**: that particular version is decompressed from the local database to the working directory
   - Usually you start from the last snapshot you did, so don't do nothing

Usual workflow (revisit'd)



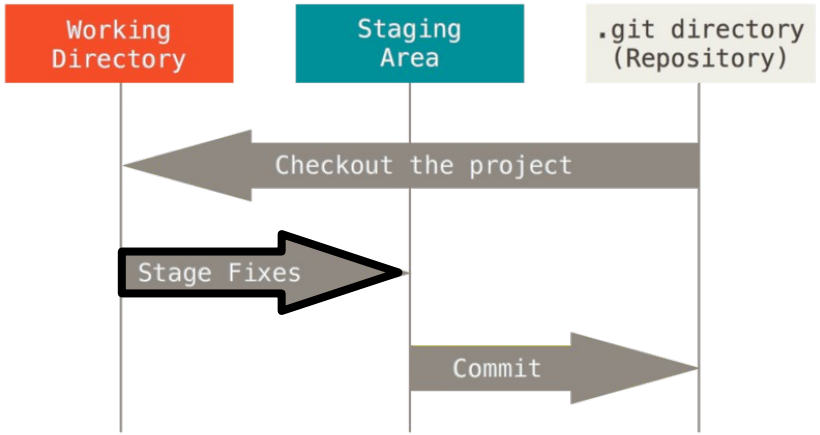
1. Start working from a particular snapshot: that particular version is decompressed from the local database to the working directory
   - **Usually you start from the last snapshot you did, so don't do nothing**

Usual workflow (revisit'd)



1. Start working from a particular snapshot: that particular version is decompressed from the local database to the working directory
   - Usually you start from the last snapshot you did, so don't do nothing
2. **Modify files from the working tree**

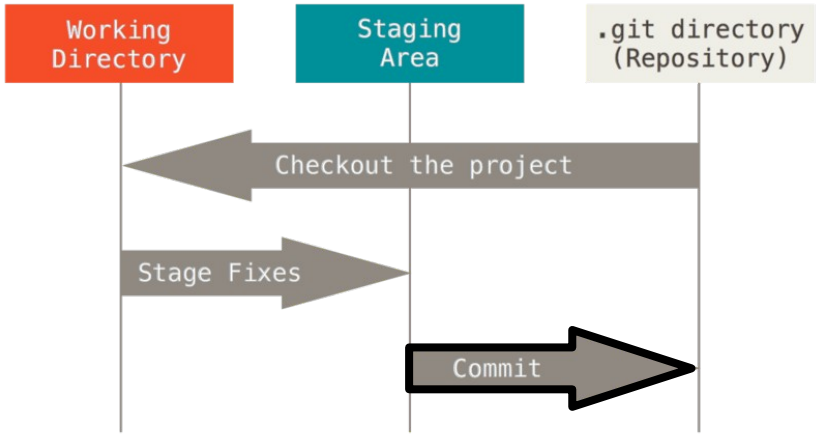
Usual workflow (revisit'd)




```
$ git add <file(s)>
```

1. Start working from a particular snapshot: that particular version is decompressed from the local database to the working directory
   - Usually you start from the last snapshot you did, so don't do nothing
2. Modify files from the working tree
3. **Selectively stage just those changes you want to be part of the next commit**

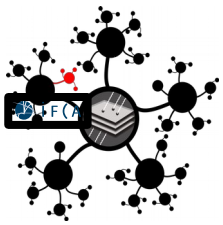
Usual workflow (revisit'd)




```
$ git commit -m "Message"
```

1. Start working from a particular snapshot: that particular version is decompressed from the local database to the working directory
   - Usually you start from the last snapshot you did, so don't do nothing
2. Modify files from the working tree
3. Selectively stage just those changes you want to be part of the next commit
4. **Do a commit**, take all the files in the staging area and stores that snapshot permanently in the database

Starting to feel the power of git...



Exercise

```
$ echo 'Your google searches can deceive you. Don't trust them
blindly' > training-01.txt
$ touch training-02.txt
$ git add training-01.txt
$ git status
$ git commit -m "Include lesson on keeping critical spirit"
$ git add training-02.txt
$ echo 'Git or Git not. There is no try.' >> training-02.txt
$ echo -ne '\n-+ Some useful subcommands:\n' >> training-02.txt
$ git add -p training-02.txt
$ git commit -m "Include the need to take sides"
$ git add training-02.txt
$ git commit -m "Prepare placeholder for command reference"
$ git log
```

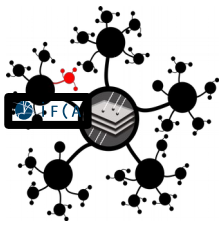
Interlude 2: commit best practices

1. Concise and granular commits
   - Commits should be focused in a particular change, or group of conceptually equivalent changes
2. Commit often
   - Easy to track changes, share quickly, ...
   - Avoid large, independent set of changes
3. Commit finalized work
   - Don't commit changes which are not logically finished, but
   - Split your changes in small chunks in order to accomplish the ultimate objective (so you can apply 1. and 2.)

Interlude 2: commit best practices

4. **Write meaningful and useful commit messages** summarizing the implemented changes, why was needed, what is different now, ...
   - Capitalized short summary (of 50 or so chars)
     - Use imperative tense, start after the sentence: "This commit will "
       - "Fix bug" not "Fixed bug", "Fixes bug" or "fix bug"
   - If needed, more detailed text: leave a blank line after the previous summary
     - Wrap it in about 72-80 characters
5. [If relevant] Test the code before commit
   - Be sure the changes do not break anything
6. **Branching**
   - Create a branch to develop a new line of development, a set of large modifications, a bugfix, ... → one of the most Git's powerful features

Starting to feel the power of git...



Exercise

```
$ echo 'Your google searches can deceive you. Don't trust
them blindly' > training-01.txt
$ touch training-02.txt
$ git add training-01.txt
$ git status
```

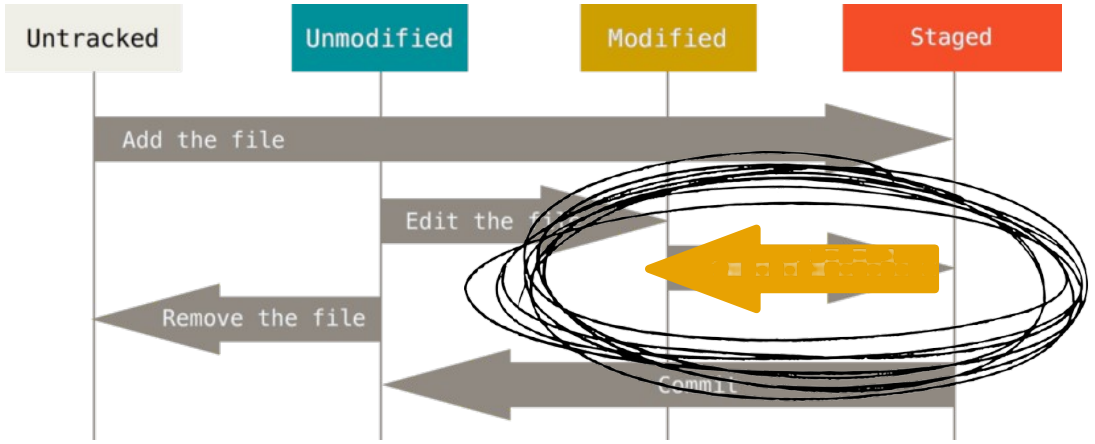
- Provides information about the working and staging areas current status.
- It shows:
  - Staged files → ready to be committed
  - Modified files → ready to be added
  - Untracked files → not version-controlled

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   training-01.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        idontcarethis.dat
        training-02.txt
```

- Also, provides you **reminders** of what you can do, especially useful if you want to undo any action...

Reconsider changes

- How to come back to modified state once you staged some changes




```
$ git reset <file>
```

```
$ git reset training-01.txt
$ git status
```

- The file is in modified state again
- Let's come back to the previous slide state...

```
$ git add training-01.txt
```

Ignore annoying files

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   training-01.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        idontcarethis.dat
        training-02.txt
```

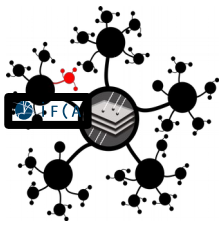
- .gitignore allows to ignore intentionally untracked files
  - Accepts wildcards/patterns (https://git-scm.com/docs/gitignore)

Exercise

```
$ cat '*.dat' > .gitignore
$ git status
$ git add .gitignore
```

- Try to follow the commit good practices: unstage training-01.txt, commit the .gitignore, and then add again training-01.txt

Starting to feel the power of git...

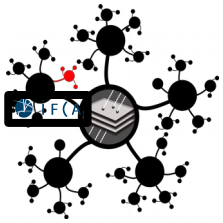


Exercise

```
$ git commit -m "Include lesson on keeping critical spirit"
$ git add training-02.txt
$ echo 'Git or Git not. There is no try.' >> training-02.txt
$ echo -ne '\n-+ Some useful subcommands:\n' >> training-02.txt
$ git add -p training-02.txt
$ git commit -m "Force to take a side"
$ git add training-02.txt
```

- Fine control over what changes to include in the next commit
  - Modified files can be partially staged
  - Interactive command: follow instructions

```
$ git add -p <file(s)>
```



Fine control over staging


```
diff --git a/training-02.txt b/training-02.txt
index e69de29..81e2e6a 100644
--- a/training-02.txt
+++ b/training-02.txt
@@ -0,0 +1,3 @@
+Git or Git not. There is no try.
+
+-+Some useful subcommands:
Stage this hunk [y,n,q,a,d,e,?]?
```

- Modifications adding a line are marked as **+** (and green color if available)
- Modifications removing a line are marked as **–** (and red color, if available)

- **y** → will stage everything between blue lines → DO NOT WANT THIS
- **n** → will not stage the shown lines → DO NOT WANT THIS
- ...
- **h/?** → HELP, shown what does it means every option

- We need to edit manually the change: **e** → it will open and editor

Fine control over staging

```
# Manual hunk edit mode -- see bottom for a quick guide.
@@ -0,0 +1,3 @@
+Git or Git not. There is no try.
+
+-+Some useful subcommands:
# ---
# To remove '-' lines, make them ' ' lines (context).
# To remove '+' lines, delete them.
# Lines starting with # will be removed.
#
# If the patch applies cleanly, the edited hunk will immediately be
# marked for staging.
# If it does not apply cleanly, you will be given an opportunity to
# edit again.  If all lines of the hunk are removed, then the edit is
# aborted and the hunk is left unchanged.
```

- Editor shown: vim
- Editor is selected depending the git configuration (see details in https://git-scm.com/docs/git-config)

- Instructions in the last lines
- We want to keep just the first line to stage in a separate commit

Fine control over staging

```
# Manual hunk edit mode -- see bottom for a quick guide.
@@ -0,0 +1,3 @@
+Git or Git not. There is no try.
# ---
# To remove '-' lines, make them ' ' lines (context).
# To remove '+' lines, delete them.
# Lines starting with # will be removed.
#
# If the patch applies cleanly, the edited hunk will immediately be
# marked for staging.
# If it does not apply cleanly, you will be given an opportunity to
# edit again.  If all lines of the hunk are removed, then the edit is
# aborted and the hunk is left unchanged.
```

- Editor shown: vim
- Editor is selected depending the git configuration (see details in https://git-scm.com/docs/git-config)

- Save the changes
- Only the *'Git or Git not. There is no try'* sentence has been staged
  - the removed lines are part of the modified changes but not staged

Look at the change details

Exercise

```
$ git commit -m "Include the needs to take sides"
$ git diff
```

- Shows what it is changed (modified) but not staged
- It can be used to compare staged changes with the last commit
- It can be used to compare changes introduced between different commits

```
$ git diff
```

```
$ git diff --staged
```

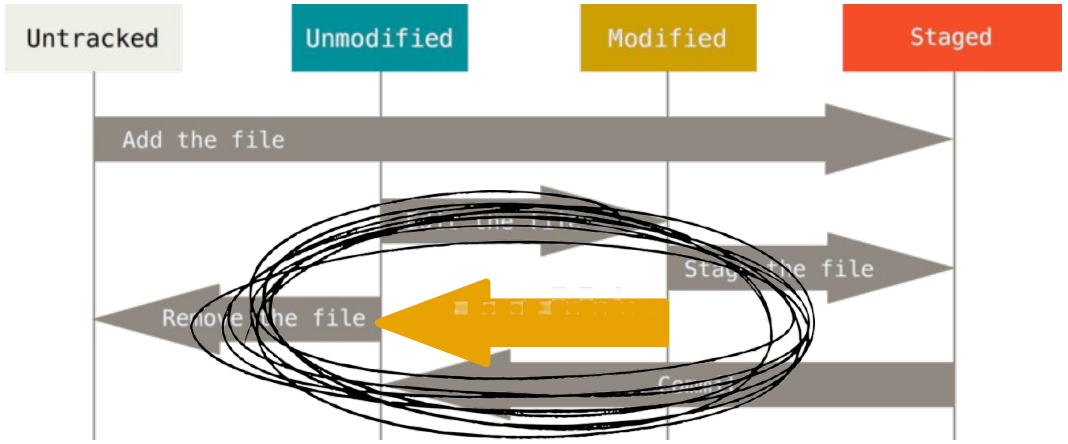
```
$ git diff <c1> <c2>
```

```
diff --git a/training-02.txt b/training-02.txt
index d1d85c7..81e2e6a 100644
--- a/training-02.txt
+++ b/training-02.txt
@@ -1 +1,3 @@
 Git or Git not. There is no try.
+
+-+Some useful subcommands:
```

Not staged modifications

Amend a mistake (unmodify)

- How to ignore changes in a modified file




```
$ git checkout -- <file>
```

```
$ git checkout -- training-02.txt
$ git status
```

```
On branch master
nothing to commit, working tree clean
```

```
$ echo -ne '\n-+Some useful subcommands:\n' >> training-02.txt
```

Reviewing history

Exercise

```
$ git add training-02.txt
$ git commit -m "Prepare placeholder for command reference"
$ git log
```

```
commit b4456d0a1fa068935ddc1ed0463be46986bb8a38 (HEAD -> master)
Author: Jordi Duarte Campderros <jorge.duarte.campderros@cern.ch>
Date:   Sun Jan 10 02:37:53 2021 +0100

    Prepare placeholder for command reference

commit c4c9d6c11e469ad3902b7cf0c92b614b1935ec37
Author: Jordi Duarte Campderros <jorge.duarte.campderros@cern.ch>
Date:   Sat Jan 9 23:44:11 2021 +0100

    Include the needs to take sides

commit f689329bc321a697b296ad03a168cc8ceb3ac643
Author: Jordi Duarte Campderros <jorge.duarte.campderros@cern.ch>
Date:   Sat Jan 9 21:13:27 2021 +0100

    Include lesson on keeping critical spirit

commit 43e6b24b9474e1380caa03a05d8f209a0fd8fbe8
Author: Jordi Duarte Campderros <jorge.duarte.campderros@cern.ch>
Date:   Sat Jan 9 21:08:57 2021 +0100

    Include .gitignore

commit 444b6772294167b5ab38d036a599a05c0723c481
Author: Jordi Duarte Campderros <jorge.duarte.campderros@cern.ch>
Date:   Sat Jan 9 19:55:04 2021 +0100

    Initial commit
```

- Shows commits made in reverse chronological order (with no options)

- Full of options to extract the exact information you're looking for

```
$ git log --patch
```

```
$ git log --pretty=oneline
```

```
...
```

- Try this one:

```
$ git log --graph --decorate --abbrev-commit --pretty=oneline
```

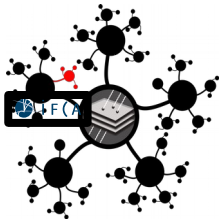
Amend a commit

- How to add forgotten files and/or changes, or recreate the last commit message
  - Just add whatever you forgot, stage them and commit again using

```
$ git commit --amend
```

    it will open your editor: you can change the message or keep it
  - Useful to minor improvements, avoiding messages in the commit history like "Forgot to include some files" or "Fix Typo in last commit" ...

    CAREFUL: **Do not amend commits pushed** somewhere else, otherwise will cause problems with your team collaborators



The awaken of the branches

Exercise

```
$ git checkout -b dark-side
$ echo 'I am your father' > approach-darkside.txt
$ git add approach-darkside.txt
$ git commit -m "Reveal the secret"
$ echo 'If you only knew the power of CVS' >> approach-darkside.txt
$ echo 'Join me and I will complete your training.' >> approach-
darkside.txt
$ git commit -a -m "Try to convince towards the dark side"
$ echo 'You were right… you were right about me' > coming-back.txt
$ git add coming-back.txt
$ git commit -m "Return to git"
$ git rm approach-darkside.txt
$ git commit -m "Culminate the redemption"
$ git checkout master
$ git merge dark-side
$ git branch -d dark-side
```

The awaken of the branches

Exercise

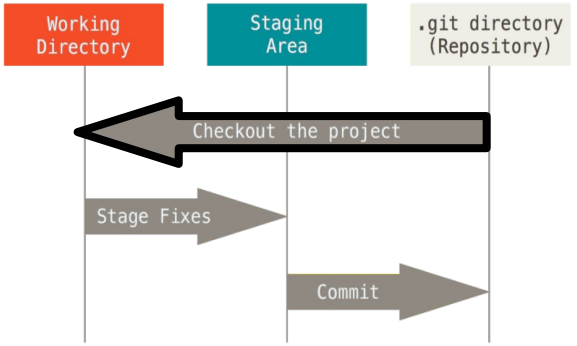
```
$ git checkout -b dark-side
```

```
* b4456d0 (HEAD -> master) Prepare placeholder for command reference
* c4c9d6c Include the needs to take sides
* f689329 Include lesson on keeping critical spirit
* 43e6b24 Include .gitignore
* 444b677 Initial commit
```

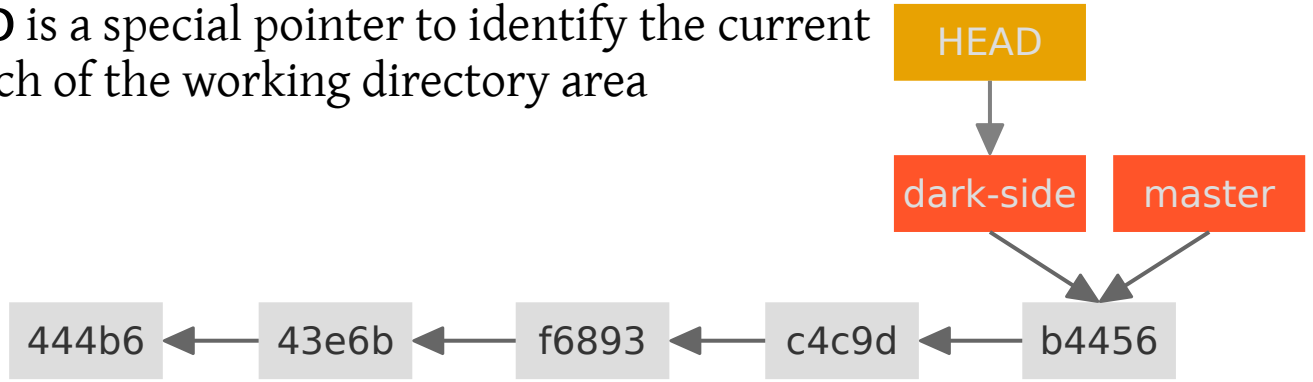
- Creates a new branch and creates a pointer to the last snapshot

```
$ git branch
```

```
* dark-side
  master
```



- A branch is a lightweight movable pointer
  - **master** is the default branch
- **HEAD** is a special pointer to identify the current branch of the working directory area

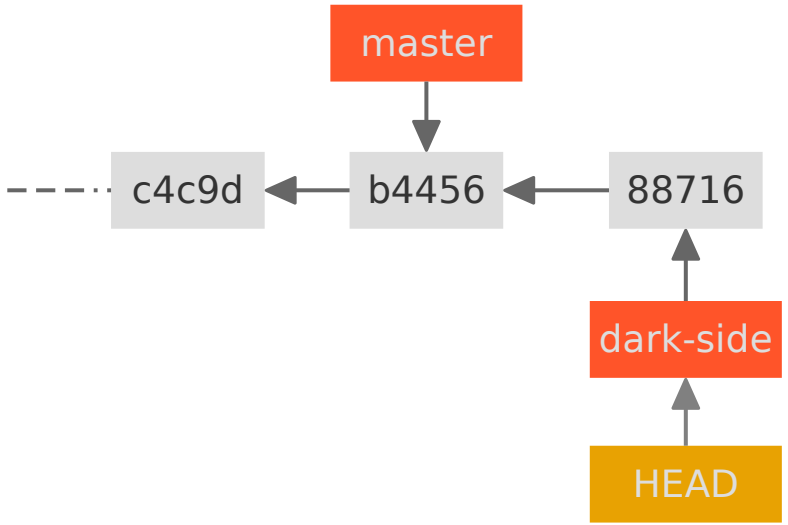


The awaken of the branches

Exercise

```
$ echo 'I am your father' > approach-darkside.txt
$ git add approach-darkside.txt
$ git commit -m "Reveal the secret"
```

```
* 8871665 (HEAD -> dark-side) Reveal the secret
* b4456d0 (master) Prepare placeholder for comm
* c4c9d6c Include the needs to take sides
* f689329 Include lesson on keeping critical sp
```

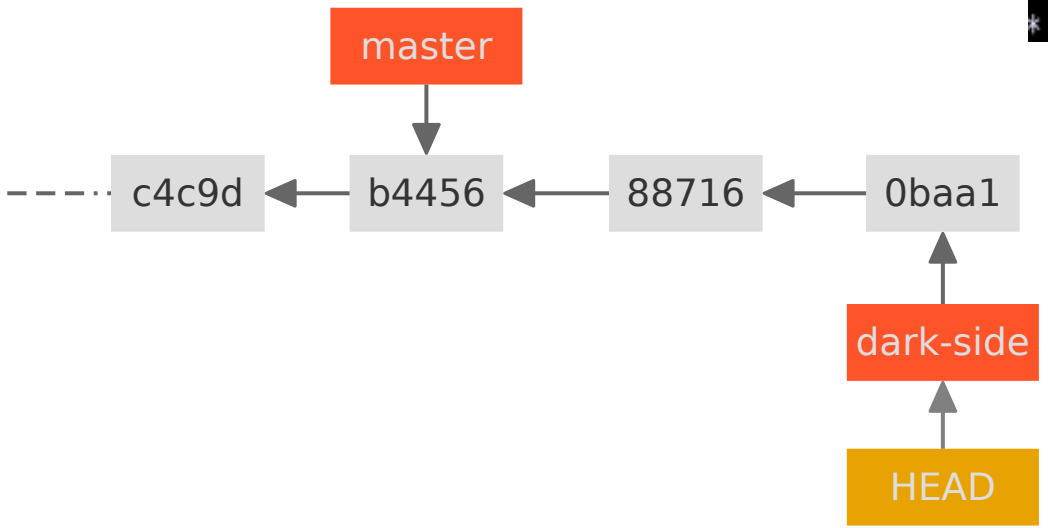


The awaken of the branches

Exercise

```
$ echo 'If you only knew the power of CVS' >> approach-darkside.txt
$ echo 'Join me and I will complete your training.' >> approach-
darkside.txt
$ git commit -a -m "Try to convince towards the dark side"
```

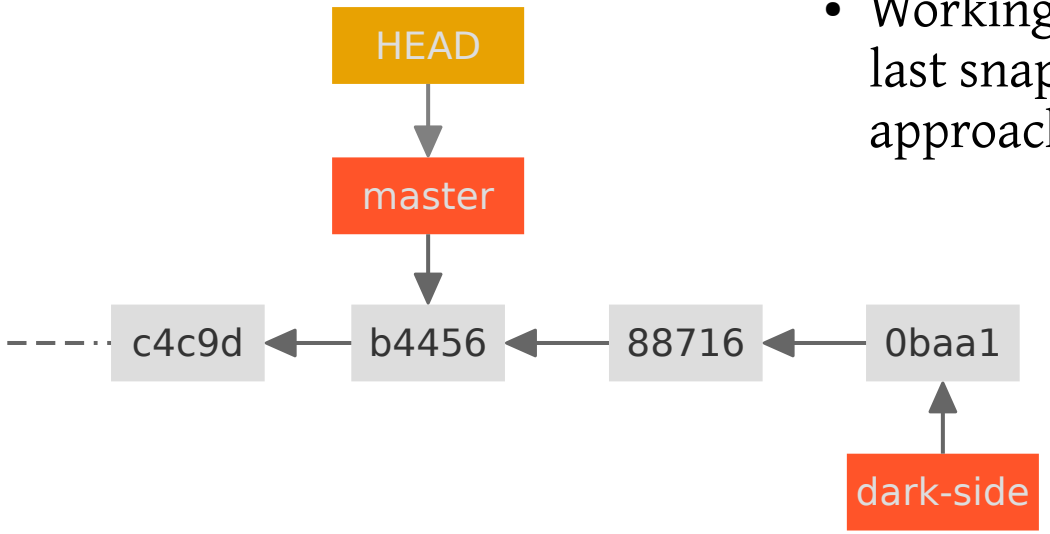
```
* 0baa19e (HEAD -> dark-side) Try to move towards
* 8871665 Reveal the secret
* b4456d0 (master) Prepare placeholder for command
```



The awaken of the branches

Exercise

```
$ git checkout master
$ ls
$ git log
```



- Working directory is now populated with the last snapshot of master (so, the file approach-darkside.txt is not there)

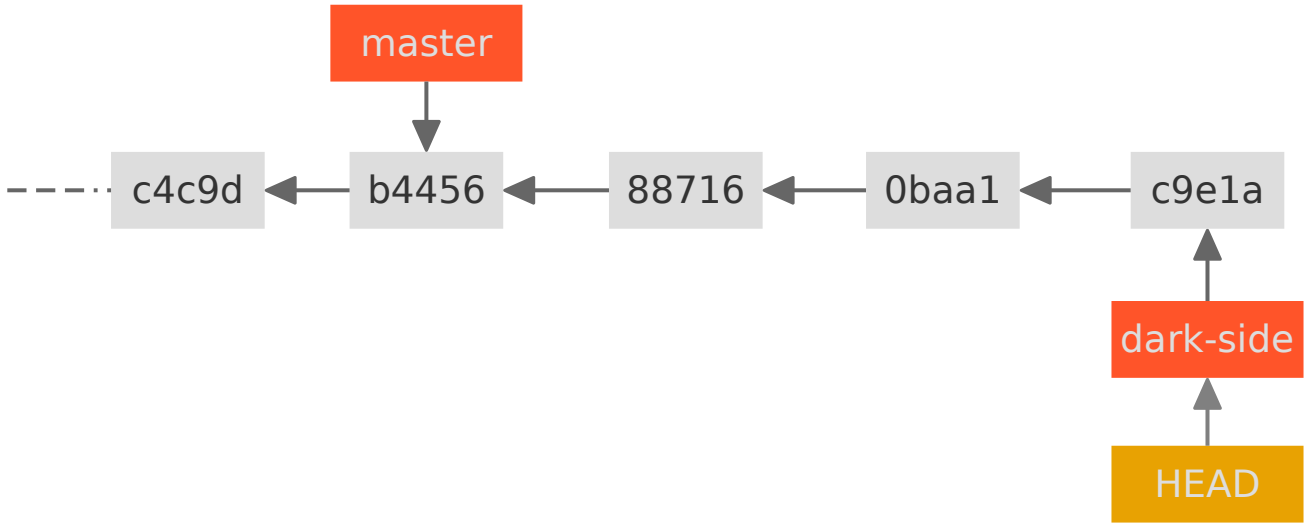
```
$ git checkout dark-side
```

The awaken of the branches

Exercise

```
$ echo 'You were right… you were right about me' > coming-back.txt
$ git add coming-back.txt
$ git commit -m "Return to git"
```

```
* c9e1abb (HEAD -> dark-side) Return to git
* 0baa19e Try to move towards the dark side
```

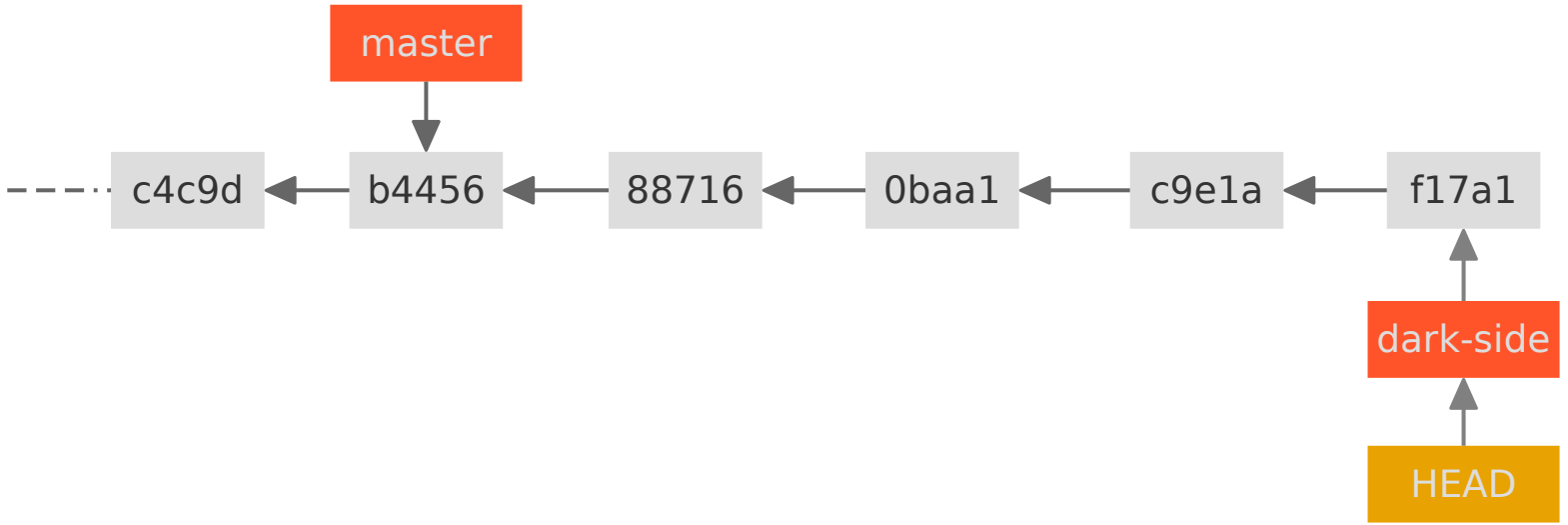


The awaken of the branches

Exercise

```
$ git rm approach-darkside.txt
$ git commit -m "Culminate the redemption"
```

```
* f17a1f9 (HEAD -> dark-side) Culminate redempti
* c9e1abb Return to git
* 0baa19e Try to move towards the dark side
```

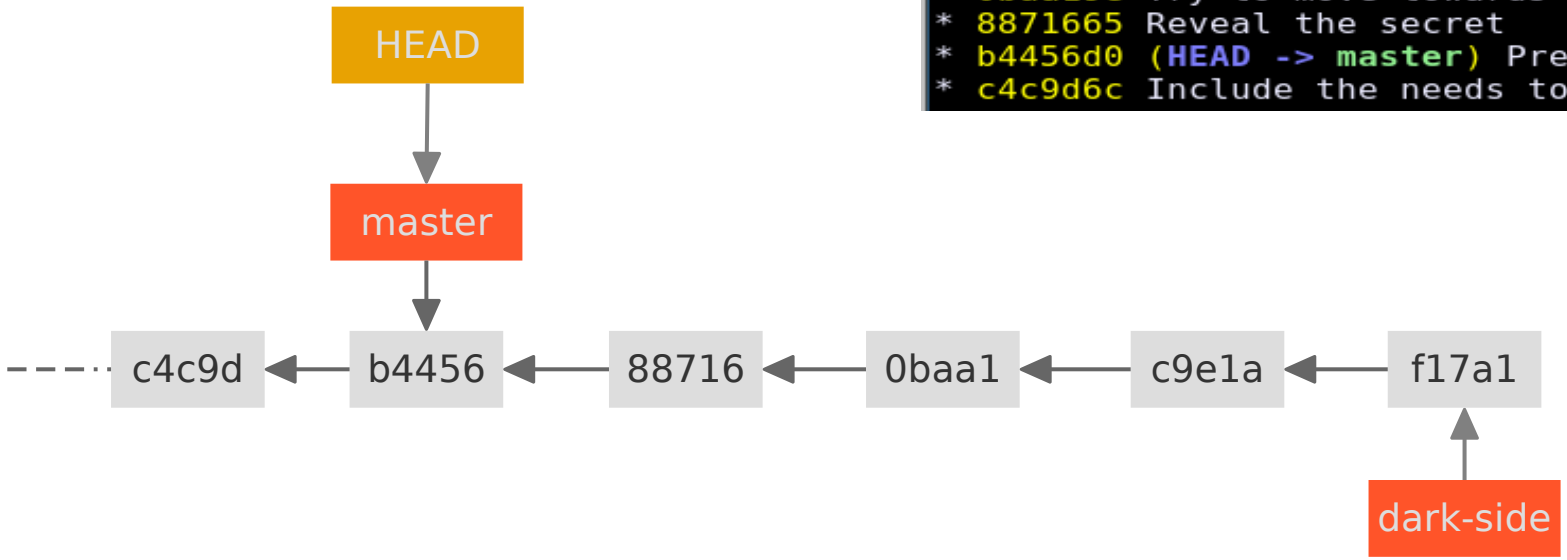


The awaken of the branches

Exercise

```
$ git checkout master
```

```
* f17a1f9 (dark-side) Culminate redemption
* c9e1abb Return to git
* 0baa19e Try to move towards the dark side
* 8871665 Reveal the secret
* b4456d0 (HEAD -> master) Prepare placeholder
* c4c9d6c Include the needs to take sides
```

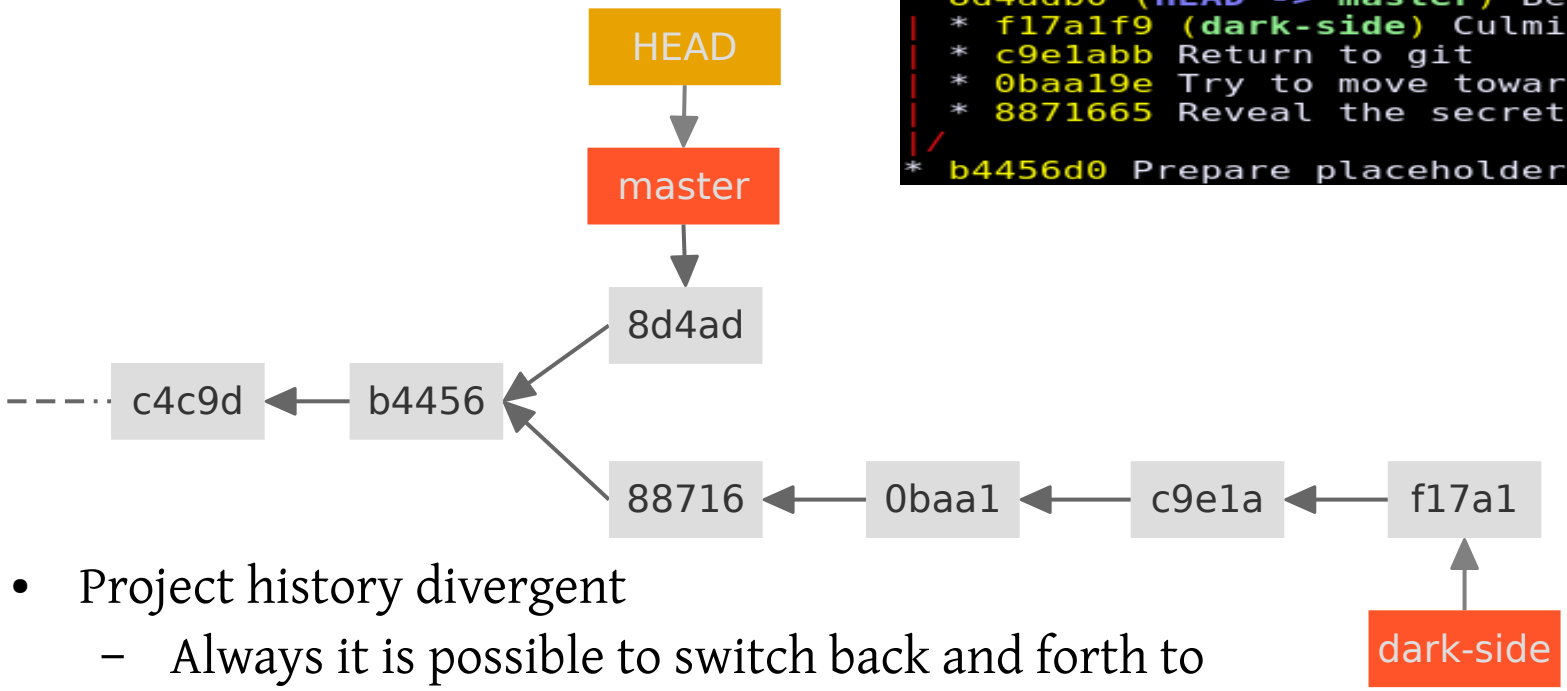


The awaken of the branches

Exercise

```
$ echo 'Fear is the path to the concurrent side' >> keepaneyeonit.txt
$ git commit -a -m "Be vigilant against the concurrent side"
```

```
* 8d4adb0 (HEAD -> master) Be vigilant agains
| * f17a1f9 (dark-side) Culminate redemption
| * c9e1abb Return to git
| * 0baa19e Try to move towards the dark sid
| * 8871665 Reveal the secret
|/
* b4456d0 Prepare placeholder for command re
```



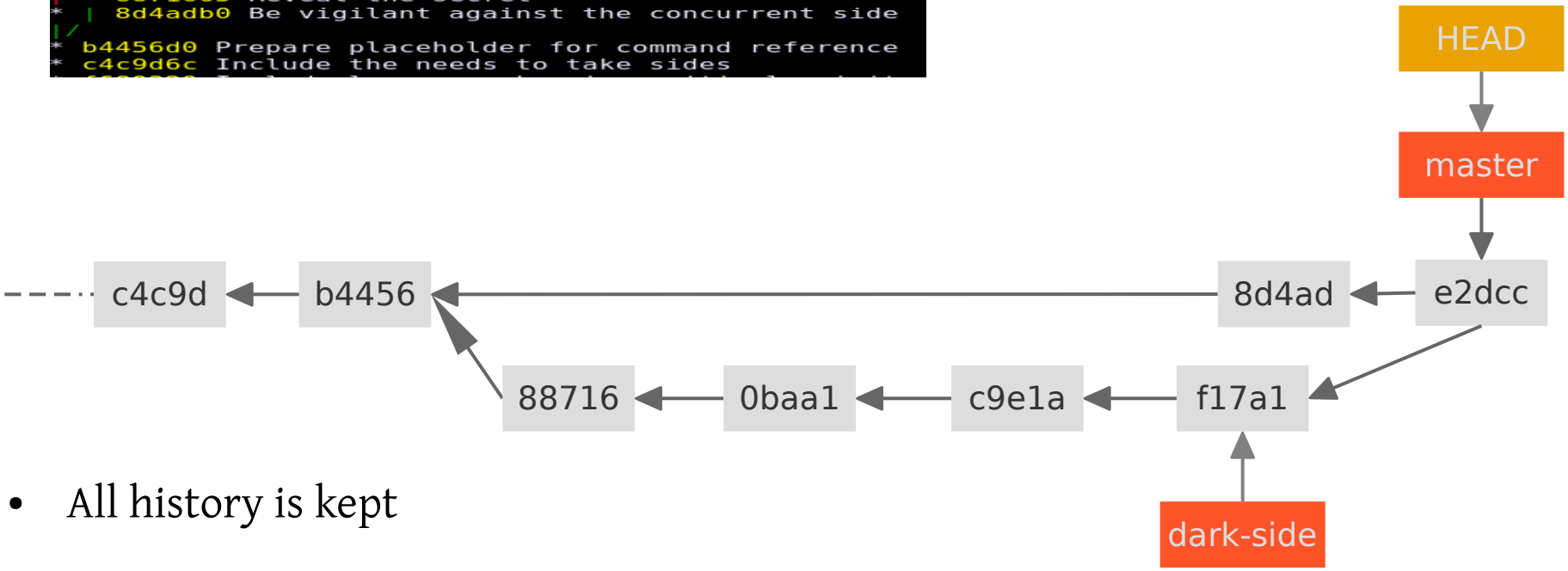
- Project history divergent
  - Always it is possible to switch back and forth to continue a history line

The awaken of the branches

Exercise

```
$ git merge dark-side
```

```
*   e2dcc2a (HEAD -> master) Merge branch 'dark-side'
|\
| * f17a1f9 (dark-side) Culminate redemption
| * c9e1abb Return to git
| * 0baa19e Try to move towards the dark side
| * 8871665 Reveal the secret
* | 8d4adb0 Be vigilant against the concurrent side
|/
* b4456d0 Prepare placeholder for command reference
* c4c9d6c Include the needs to take sides
```



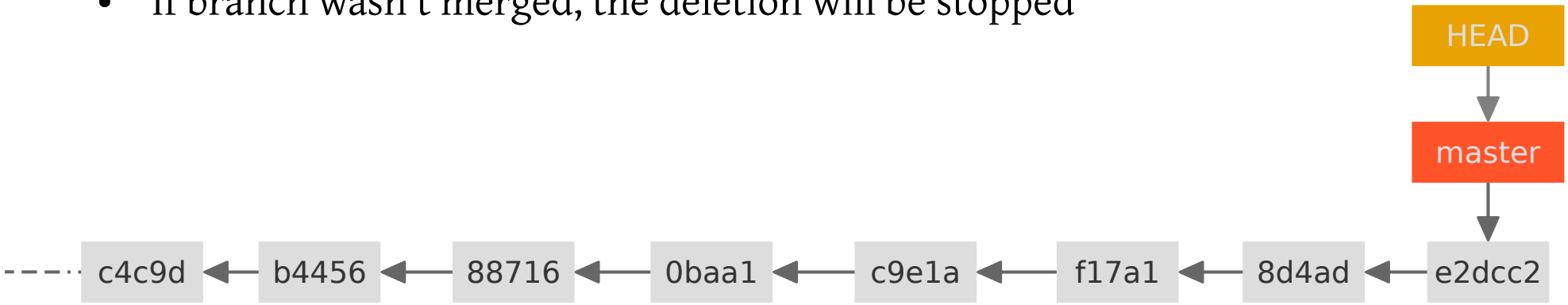
- All history is kept

The awaken of the branches

Exercise

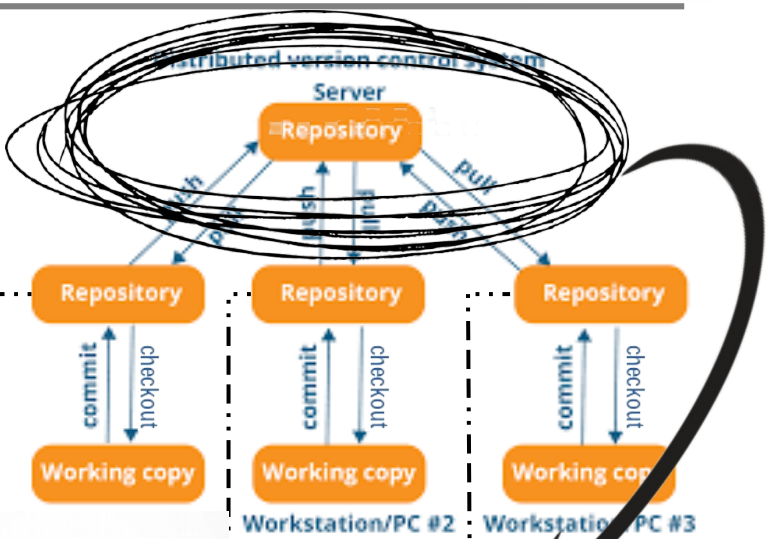
```
$ git branch -d dark-side
```

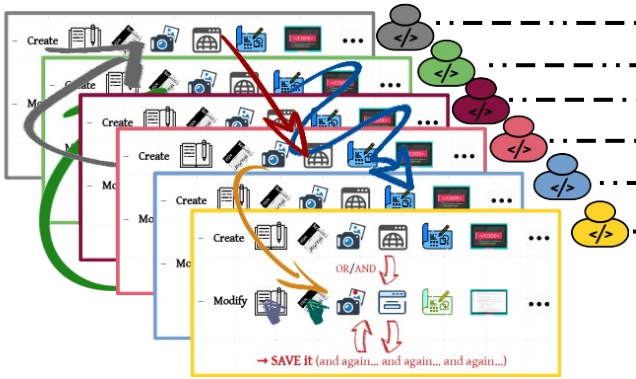
- Branch is removed and history is flatten
- If branch wasn't merged, the deletion will be stopped



Git: a distributed VCS

- Git is **distributed → collaborative**
  - Each user of project team get a local copy of the repository (from a server)
    - Use it locally (modify-save)
    - Unify changes → **MERGE**

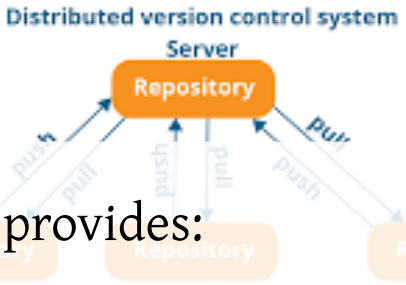




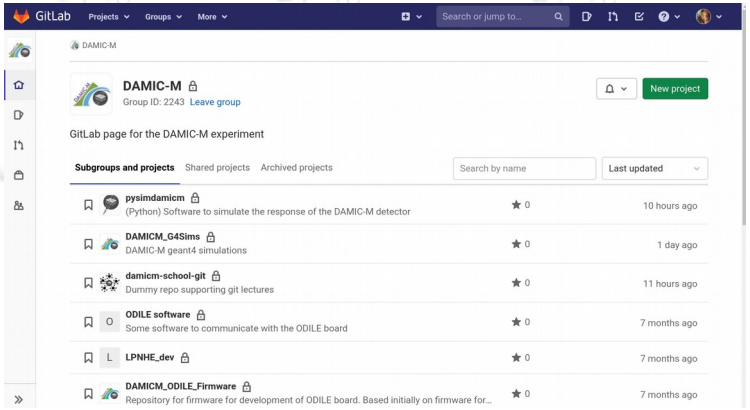
- More than git servers:
  - Web applications for complete DevOps: Issues, wiki, CI/CD, ...

Gitlab: a DevOps plattform

GitLab



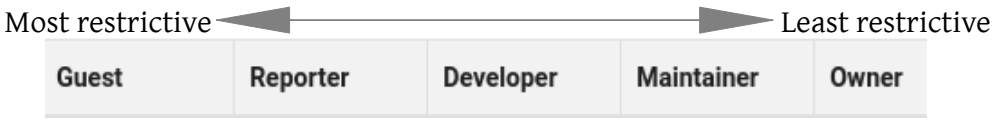
- DevOps platform. In adition to the **git repository server** it provides:
  - user management,
  - role assignation to fine control user permissions,
  - web interface to the git repositories, allowing nice commit history navigation, code reviewing, ...
  - a complete issue system,
  - a complete CI/CD framework
  - a wiki system for documentation,
  - ... and much more: https://about.gitlab.com/features



- Your server for **DAMIC-M** collaboration software projects: https://gitlab.in2p3.fr/damicm
  - It is hosted and maintained by the CC-IN2P3 at Lyon (France)
  - You should be part of the **gitlab group** DAMIC-M
    Otherwise, contact mariangela.settimo@subatech.in2p3.fr

Gitlab basics

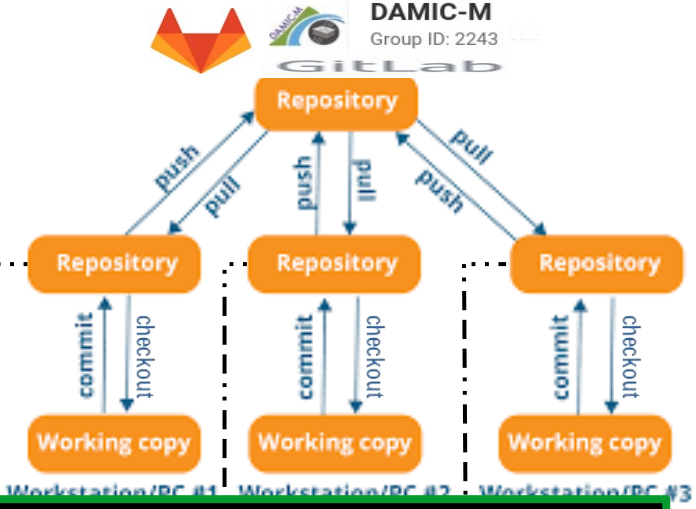
- **Project:** a repository
- **Group**: a namespace allowing to assemble project together and grant member access to several projects at once
  - You belong to the gitlab group **DAMIC-M**,
- **Role**: set of allowed actions and permissions for an user



  - Group permissions: https://docs.gitlab.com/ee/user/permissions.html#group-members-permissions
  - Project permissions: https://docs.gitlab.com/ee/user/permissions.html#project-members-permissions
  - Check if your role is suitable to do your job within the group, otherwise contact mariangela.settimo@subatech.in2p3.fr

Collaborative working

- Git is **distributed → collaborative**
  - Each user of project team get a local copy of the repository (from a server)
    - Use it locally (modify-save)
    - Unify changes → **MERGE**



Exercise

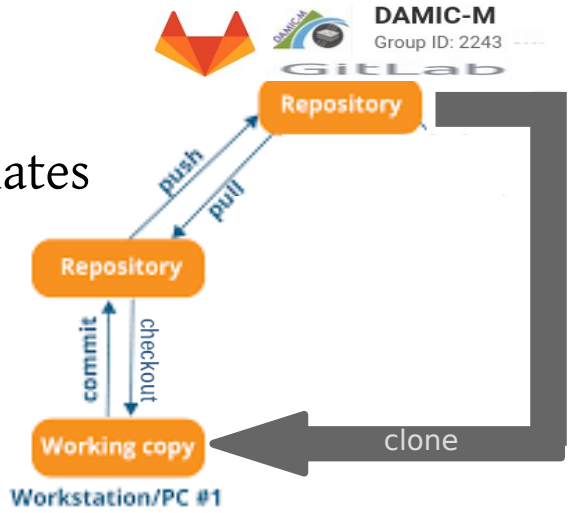
```
$ git clone git@gitlab.in2p3.fr:damicm/damicm-school-git.git
$ cd damicm-school-git
$ git checkout -b <name it yourself>
[do some changes where ever you want, create things maybe?]
$ git add <your changed files>
$ git commit -m <Your meaningful and concise message>
$ git push
```

Collaborative working

Exercise

```
$ git clone git@gitlab.in2p3.fr:damicm/damicm-school-git.git
$ cd damicm-school-git
```

- Download an exact copy of the server repository in **your local** computer, creates all areas and populates the working area with the last snapshot of the default branch (master)
  - Now you can (create-)modify-save as we've learned so far...



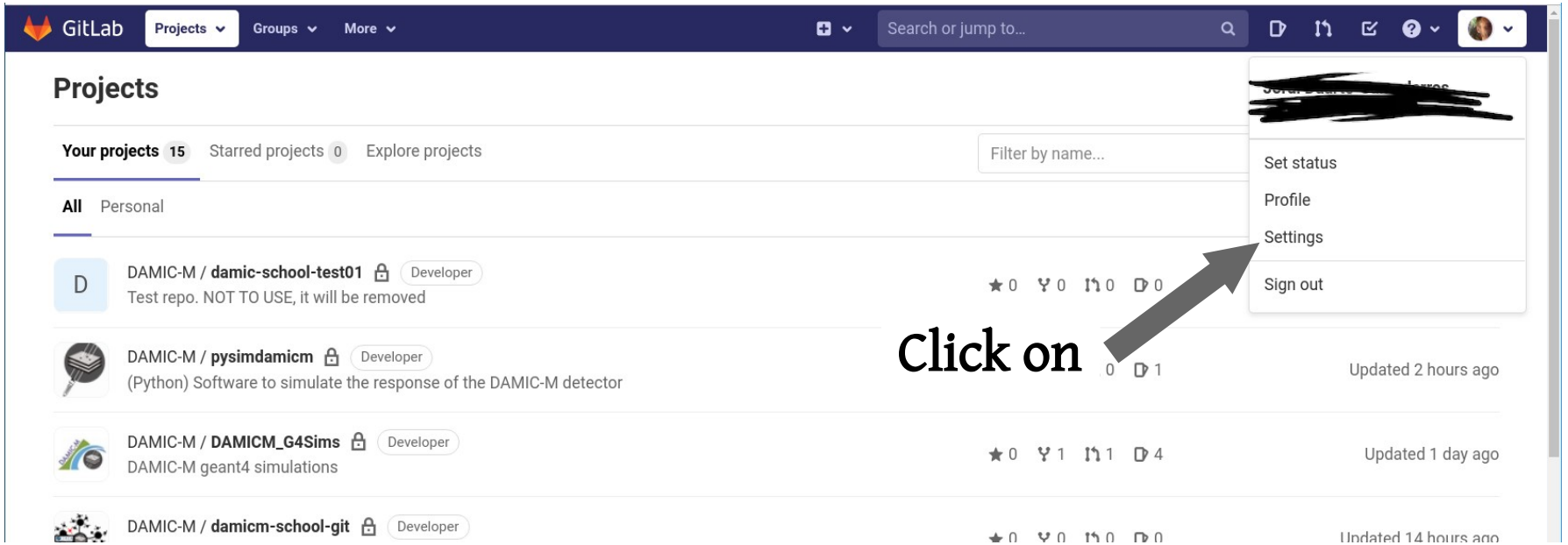
Git over SSH

- But...

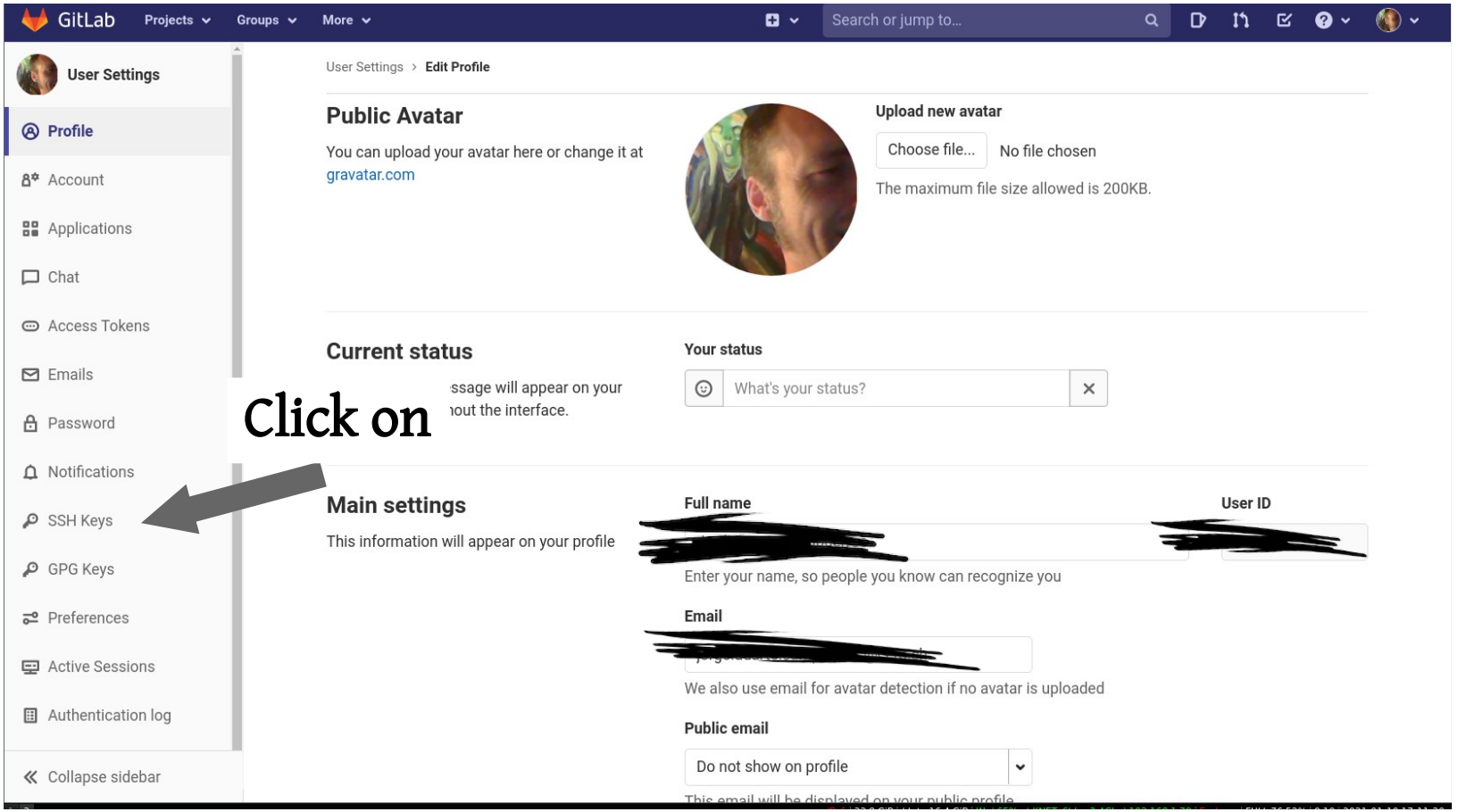
```
The authenticity of host 'gitlab.in2p3.fr (134.158.69.41)' can't be established.
ECDSA key fingerprint is SHA256:j9RRZcczB+XocN53k9R/+IAs1nLHyEjjkB4bjJiL+QU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gitlab.in2p3.fr,134.158.69.41' (ECDSA) to the list of known hosts.
git@gitlab.in2p3.fr: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

  - The server is configured to not allow authentication by asking you user/password
- You need to **add** your **SSH public key** to Gitlab, either
  a) Create the SSH key pair in the *local* computer: `$ ssh-keygen -t ed25519 -C "username@localcomputer"`
  Further instructions: https://docs.gitlab.com/ee/ssh/README.html#generating-a-new-ssh-key-pair
  b) Use an existing SSH key pair in your *local* computer, look at the ***/home/user/.ssh*** *folder*
- Get the public key (**.pub**) and copy the content into the clipboard

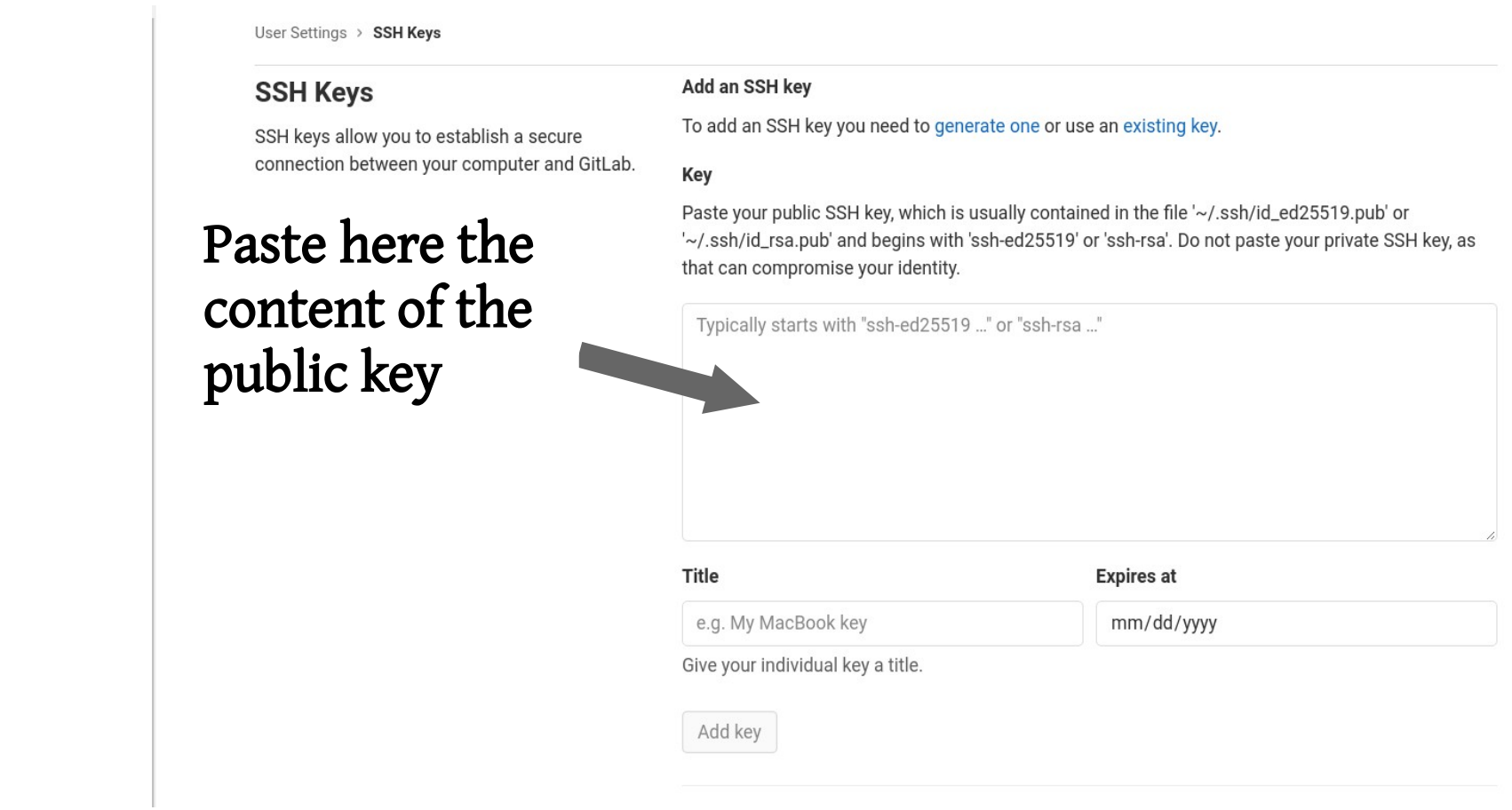
Add SSH public key



Add SSH public key

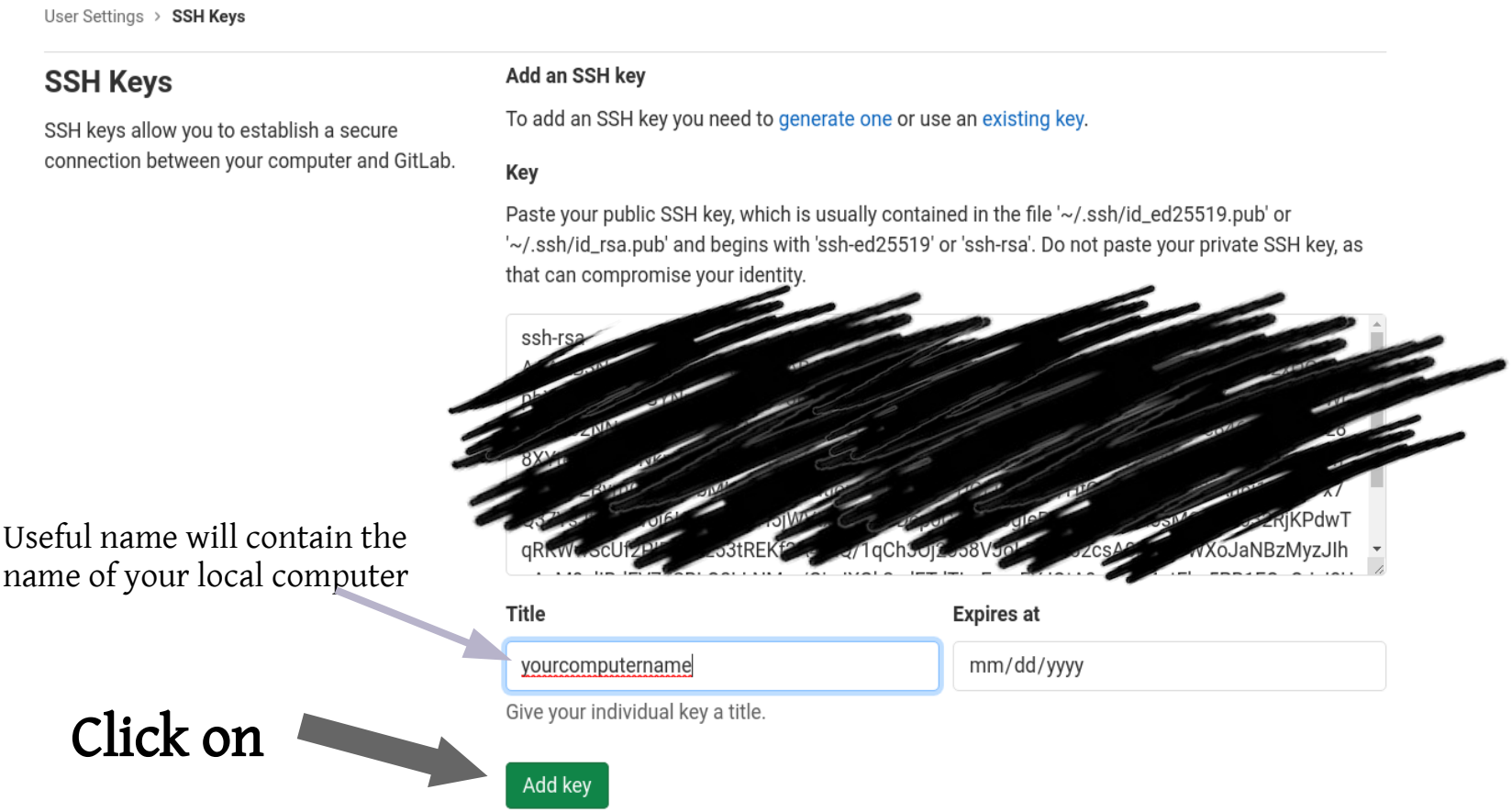


Add SSH public key



Paste here the content of the public key

Add SSH public key

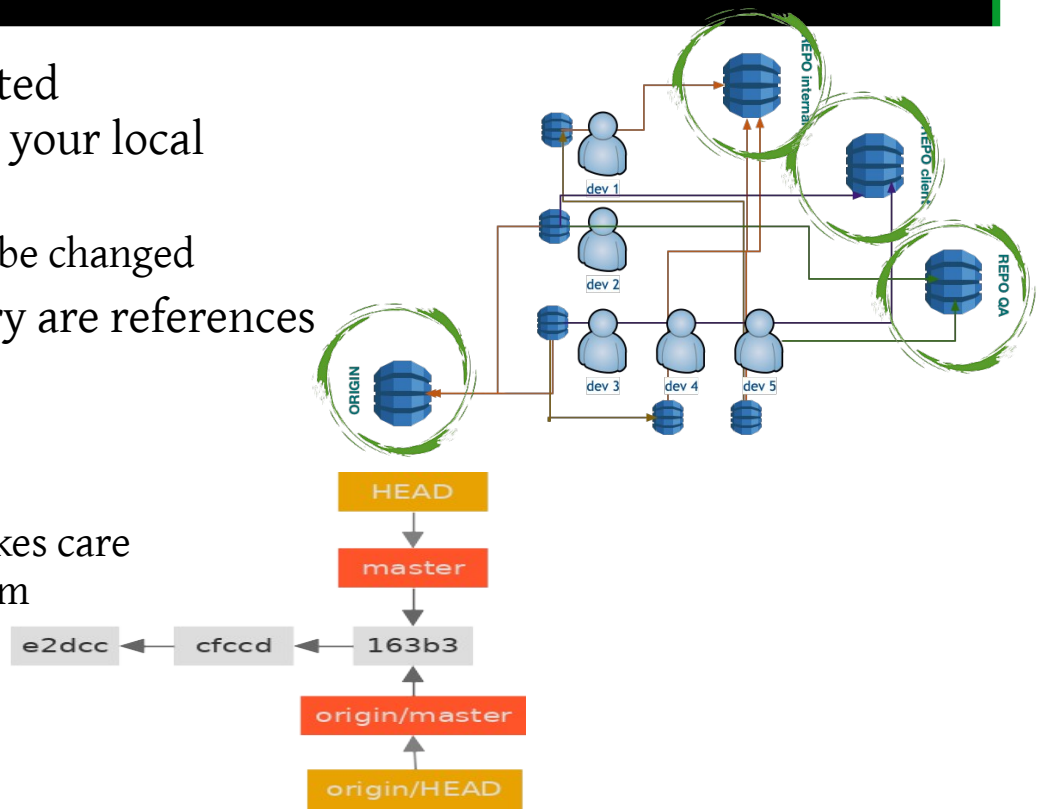


Remote repositories

Exercise

```
$ git clone git@gitlab.in2p3.fr:damicm/damicm-school-git.git
$ cd damicm-school-git
$ git remote -v
```

- Remotes are versions of the project hosted somewhere (either in the Internet or in your local computer in order location).
  - **origin**: is the per default name, but can be changed
- Remotes branches in the local repository are references to the state of the remote repository
  - **remote_name/branch_name**
  - Same behavior than before **BUT** cannot be moved locally: it's git who takes care internally whenever needs to move them



Your local repo is also a remote

- There is nothing preventing any git repository, including local to be a remote of another repository (as long as you have access to it)

```
$ git init a-copy-of-damicm-school
$ cd a-copy-of-damicm-school
$ git remote add local-damicm $HOME/repos/damicm-school-git/.git
```

  - Assuming you clone damimc-school-git at folder under your home: $HOME/repos

- This is the important thing to remember: the path ends
- And now get all the content to your copy

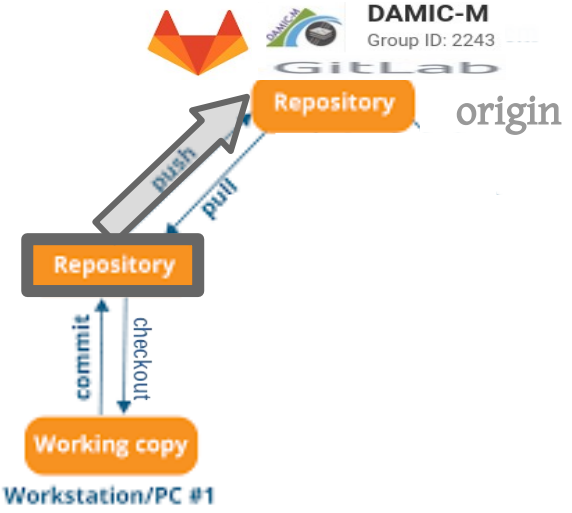
```
$ git pull local-damimc master
```

Collaborative Working

Exercise

```
$ git pull origin master
$ git checkout -b <branch_name>
[do some changes wherever you want and commit your changes]
$ git push origin <branch_name>
```

- Synchronizes your local repository with the remote
  - Once it's done, all your collaborators have access to your changes
  - If there are collapsing changes, git will prevent pushing and it will instruct you what to do (maybe pull first, merge, ...)
- **push/pull** is the mechanism to unify the collaborative infrastructure (via merge)

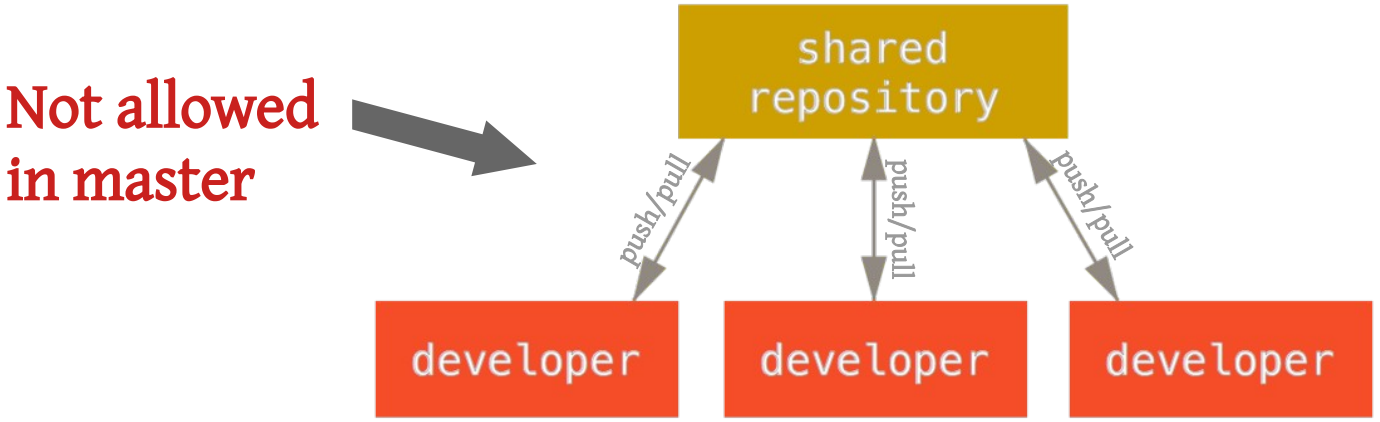


Distributed Workflows: Centralized

- If you were trying to push the **branch** master:

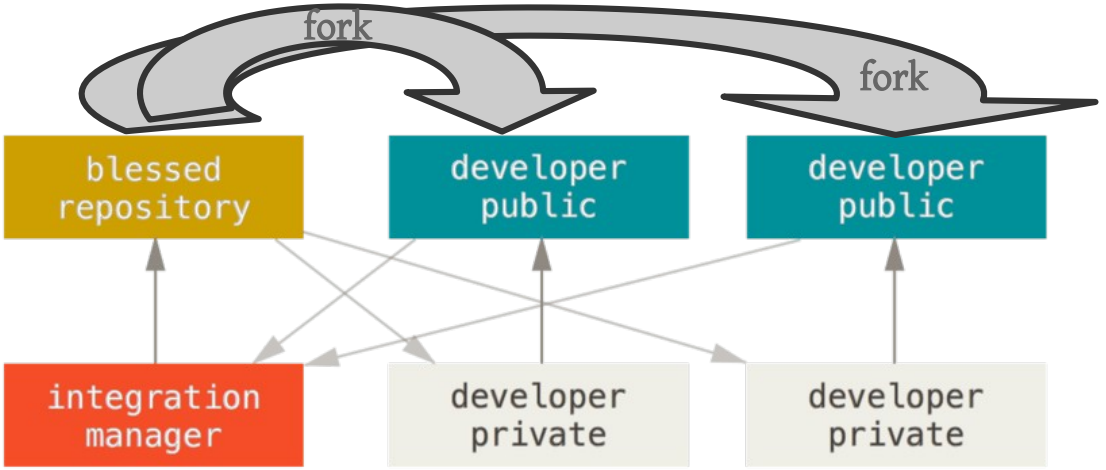
```
Writing objects: 100% (33/33), 3.15 KiB | 1.57 MiB/s, done.
Total 33 (delta 12), reused 0 (delta 0)
remote: GitLab: You are not allowed to push code to protected branches on this project.
To gitlab.in2p3.fr:damicm/damic-school-test01.git
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'git@gitlab.in2p3.fr:damicm/damic-school-test01.git'
```

  - DAMIC-M **group default** configuration prevents pushing in **master** branch (protected) without making use of the **merge request** mechanism



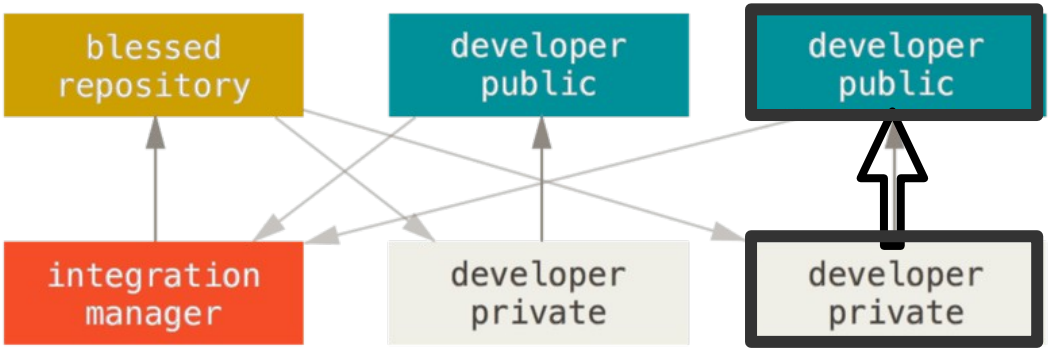
Distributed Workflows: Integration-manager

- Instead, the central repository can be push only by a maintainer. The developers must:
  - **Fork** the repository: a copy of the repository; it creates a new remote repository owned by the developer and linked to the original repo



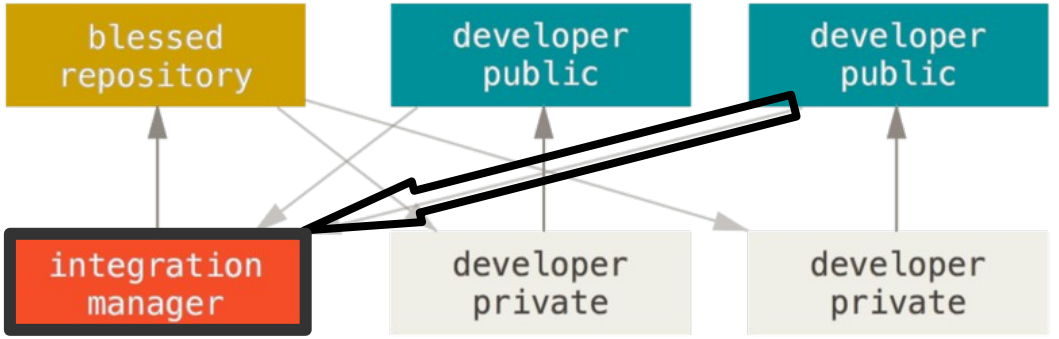
Distributed Workflows: Integration-manager

- Instead, the central repository can be push only by a maintainer. The developers must:
  - **Fork** the repository: a copy of the repository; it creates a new remote repository owned by the developer and linked to the original repo
  - Implement changes
  - Push to their public owned repo



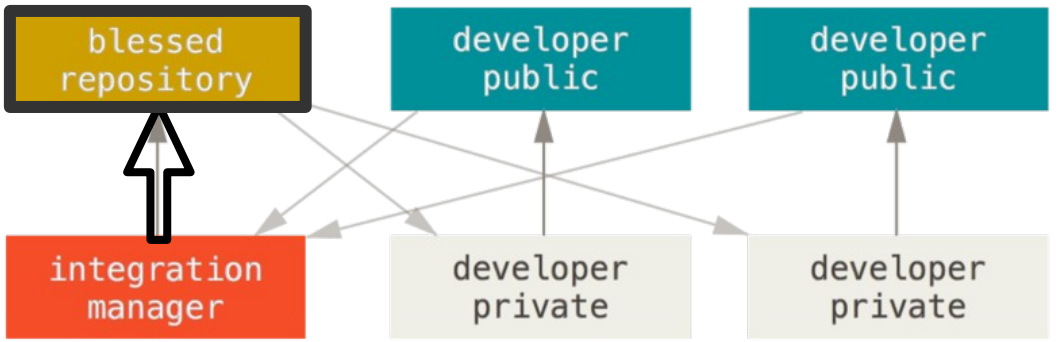
Distributed Workflows: Integration-manager

- Instead, the central repository can be push only by a maintainer. The developers must:
  - **Fork** the repository: a copy of the repository, creates a new remote repository owned by the developer
  - Implement changes
  - Push to their public owned repo
  - Merge request: request to include their changes into the central repository



Distributed Workflows: Integration-manager

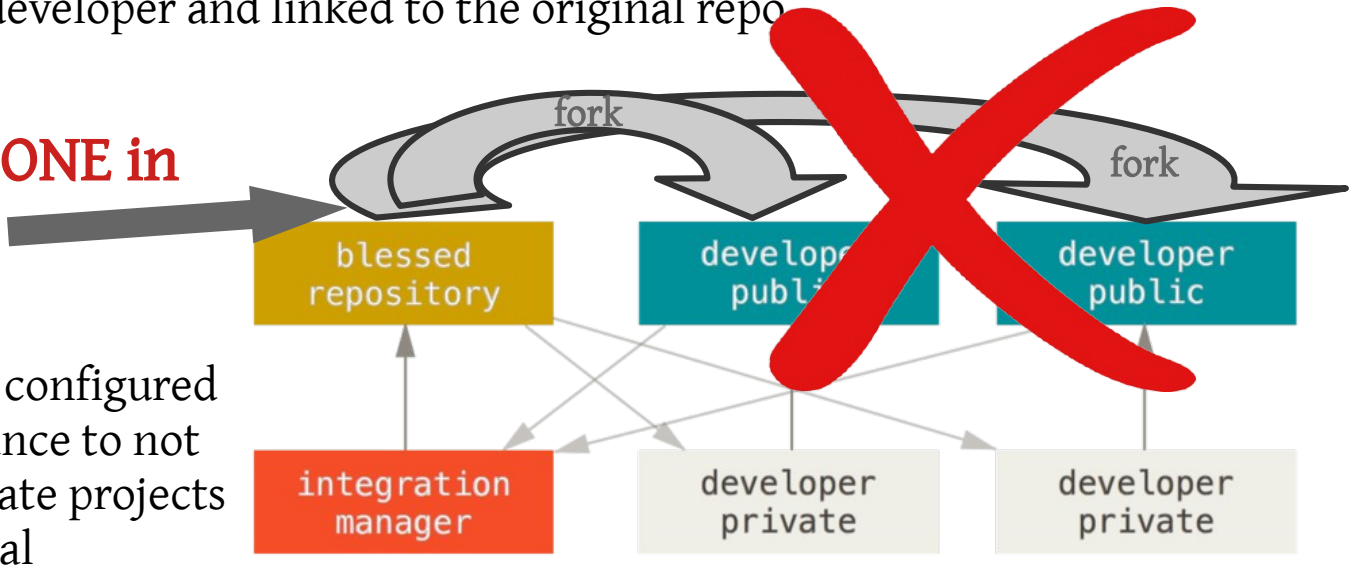
- Instead, the central repository can be push only by a maintainer. The developers must:
  - **Fork** the repository: a copy of the repository, creates a new remote repository owned by the developer
  - Implement changes
  - Push to their public owned repo
  - Merge request: request to include their changes into the central repository
  - The maintainer reviews the changes and eventually merges (or not)



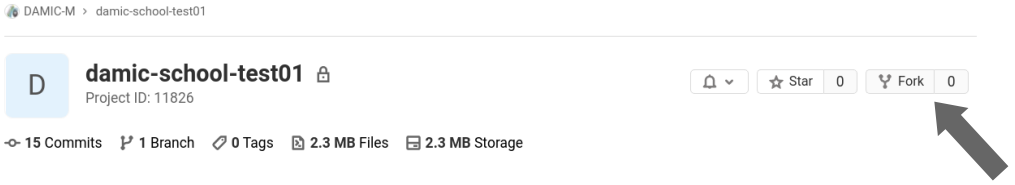
Distributed Workflows: Integration-manager

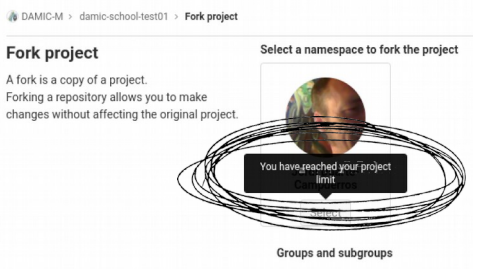
- Instead, the central repository can be push only by a maintainer. The developers must:
  - **Fork** the repository: a copy of the repository; it creates a new remote repository owned by the developer and linked to the original repo

CANNOT BE DONE in gitlab.in2p3



  - Site admin has configured the gitlab instance to not allow users create projects in their personal namespaces → **Impossible to fork!**



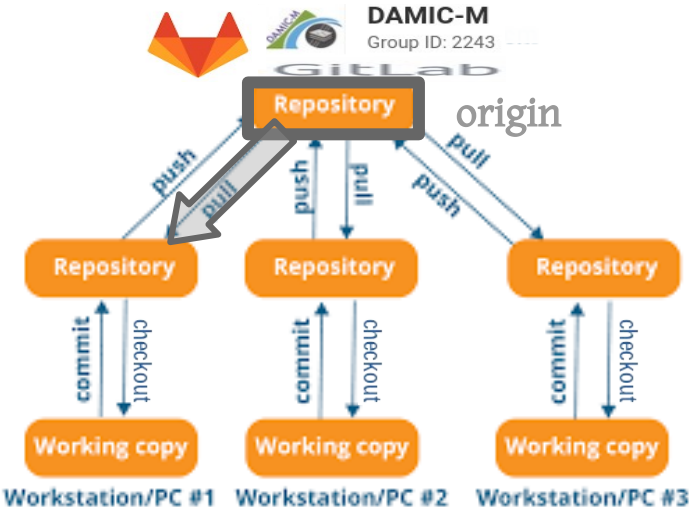


Recommended workflow

a.k.a. Feature branching

- In order to be able to apply the integration-manager workflow → use **branching approach** instead.
  1. **Pull** before start to work → copy the last changes into your local repo

```
$ git pull origin <branch>
```



Recommended workflow

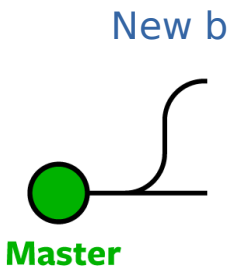
a.k.a. Feature branching

- In order to be able to apply the integration-manager workflow → use **branching approach** instead.
  1. **Pull** before start to work → copy the last changes into your local repo

```
$ git pull origin <branch>
```

  2. Create a new branch to implement your changes

```
$ git pull origin <branch>
```



Recommended workflow

a.k.a. Feature branching

- In order to be able to apply the integration-manager workflow → use **branching approach** instead.
  1. **Pull** before start to work → copy the last changes into your local repo

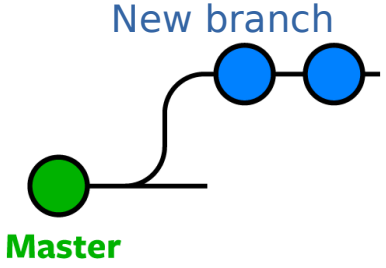
```
$ git pull origin master
```

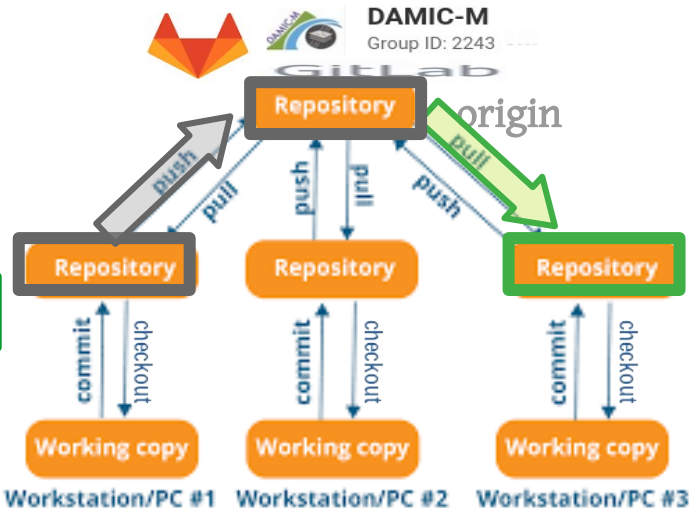
  2. Create a new branch to implement your changes

```
$ git checkout –b <new_branch>
```

  i. If working with other team mates in the same implementation, push frequently

```
$ git push origin <new_branch>
```





Recommended workflow

a.k.a. Feature branching

- In order to be able to apply the integration-manager workflow → use **branching approach** instead.
  1. **Pull** before start to work → copy the last changes into your local repo

```
$ git pull origin master
```

  2. Create a new branch to implement your changes

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

  3. Once the implementation is finished, push and create a **merge request** with the master

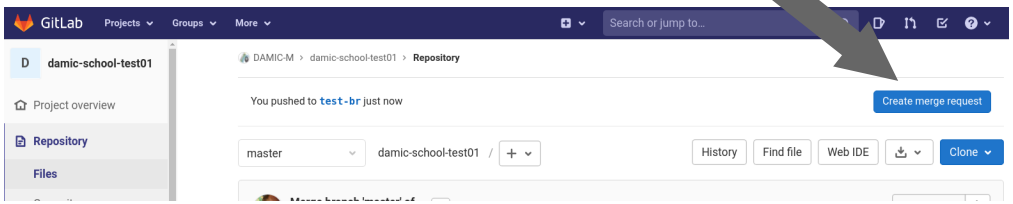
```
$ git push origin <new_branch>
```

Follow the link

```
remote:
remote:
remote: To create a merge request for test-br, visit:
remote:   https://gitlab.in2p3.fr/damicm/damic-school-test0
remote:
To gitlab.in2p3.fr:damicm/damic-school-test01.git
```

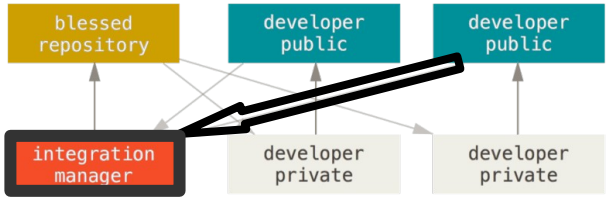
or

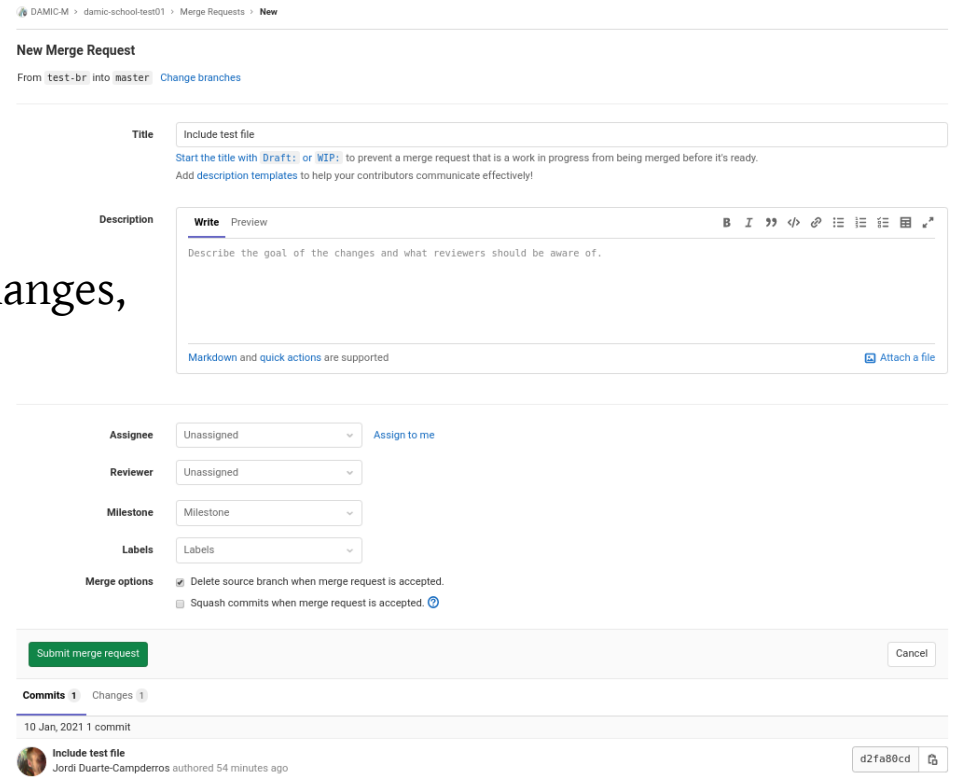
Push the button



Merge request

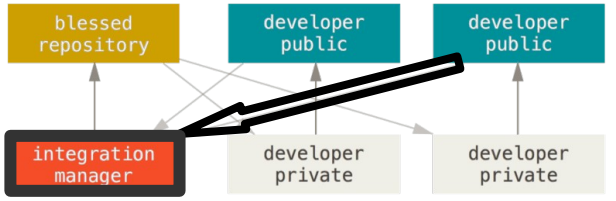
- Gitlab mechanism to discuss and review implementations between team mates
  - Fill the form request precisely as possible following the best practices described for commits:
    - **Title:** Concise summary of the changes (50 chars or less)
    - **Description:** Summarize the changes, what, why and how

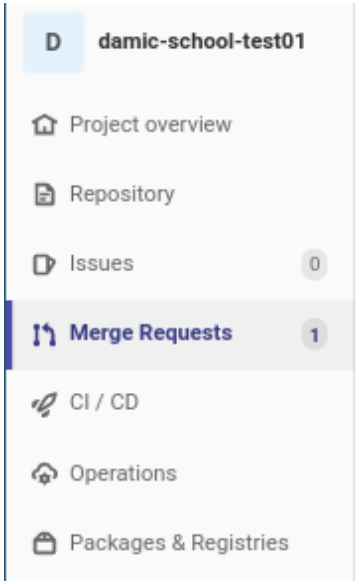


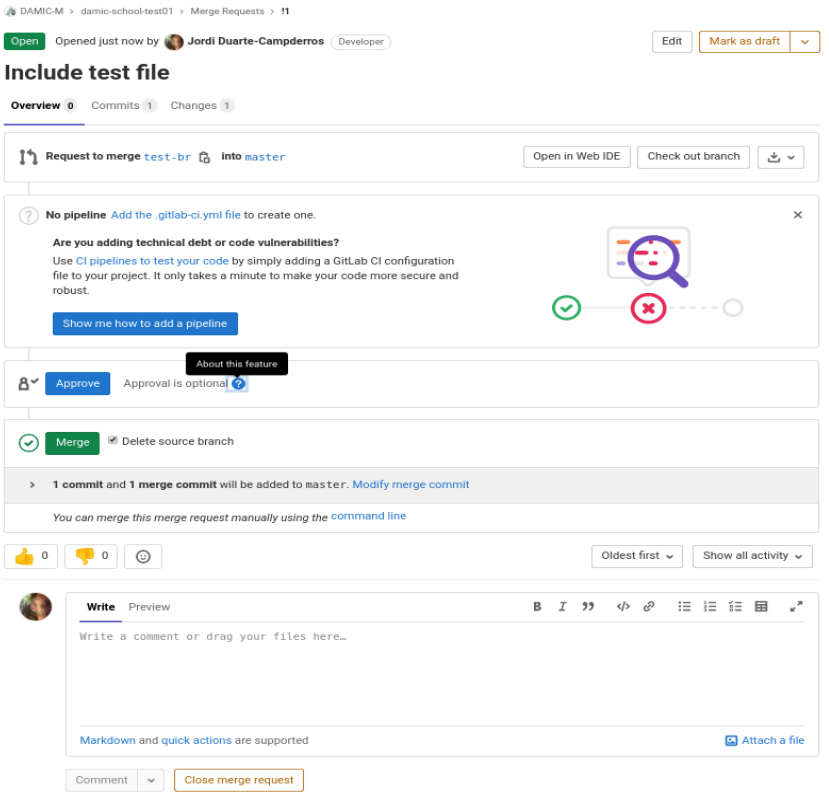


Merge request

- Gitlab provides a place where to discuss the request, with all relevant commits, files in place, and mechanisms to update the request with new commits

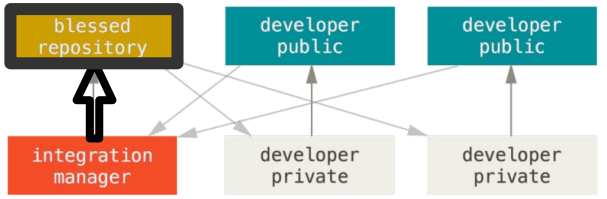


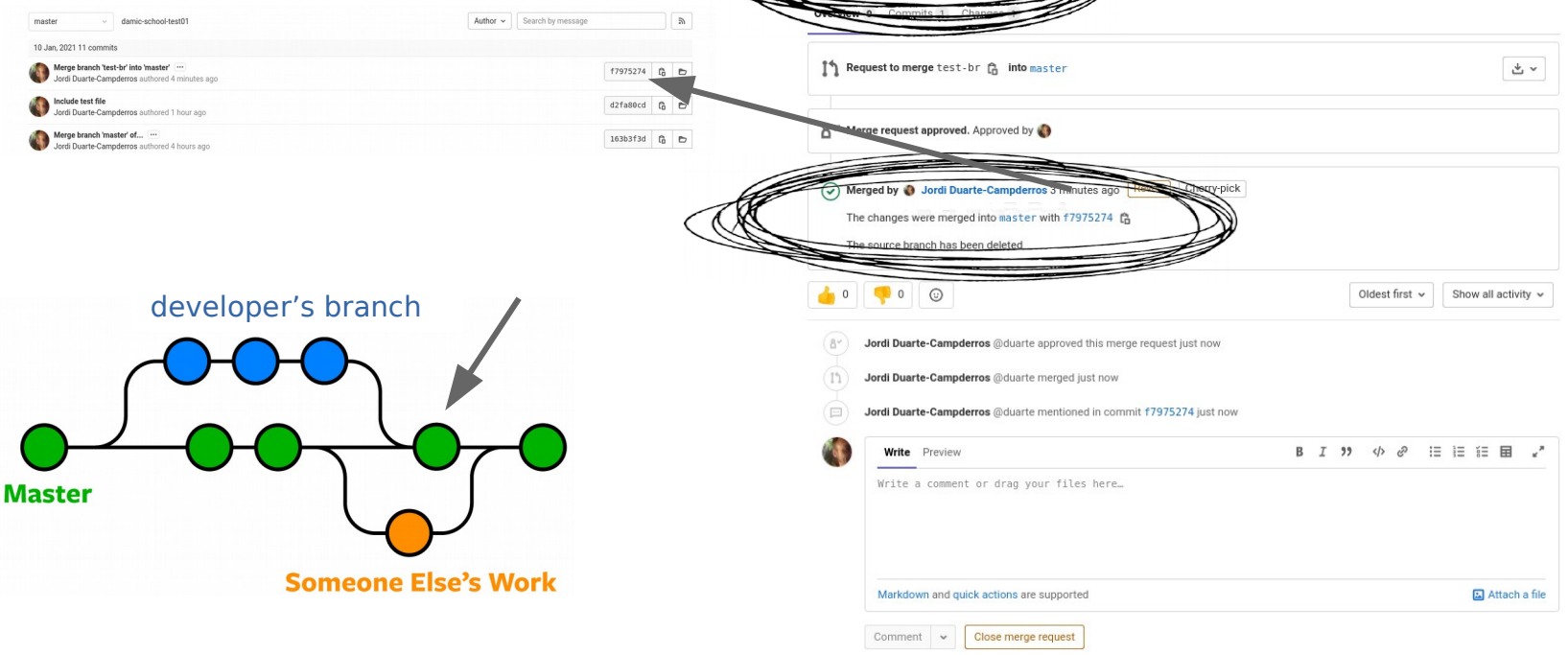




Merge request

- Once the maintainer is happy, the request is accepted and the set of commits are merged into the master branch

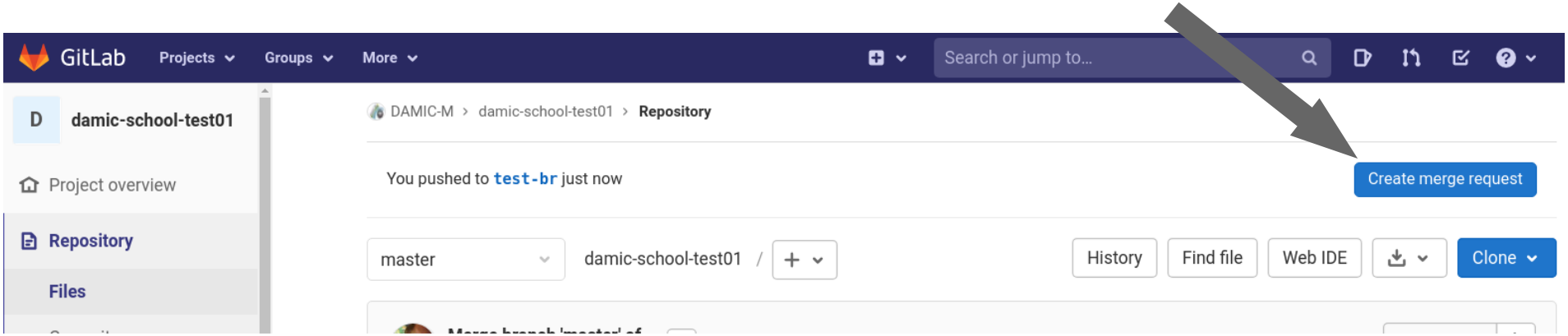




Merge request

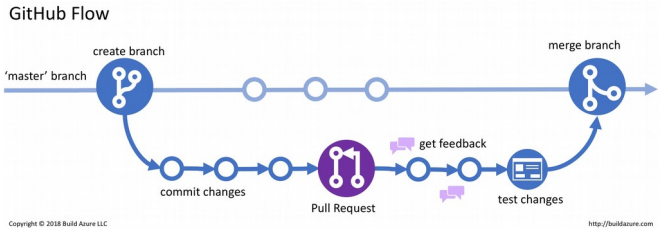
Exercise

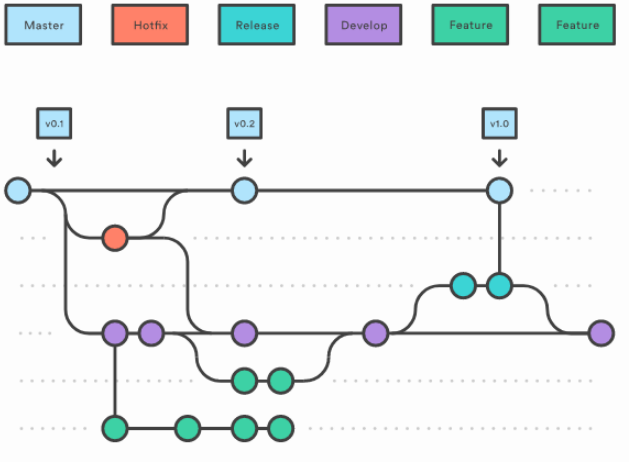
MERGE REQUEST your last push:
Go to the gitlab repository and Create a merge request



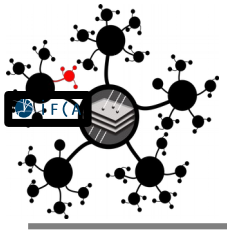
Interlude 3: Branching Strategy

- Decide which branching scheme are you going to work (or adapt yourself to what was decided in your collaboration project...)
  - All-in-master branch
  - Feature branching (already discussed)
  - GitFlow
  - GitHub flow
  - GitLab flow
  - ...





See BACKUP SLIDES for some details

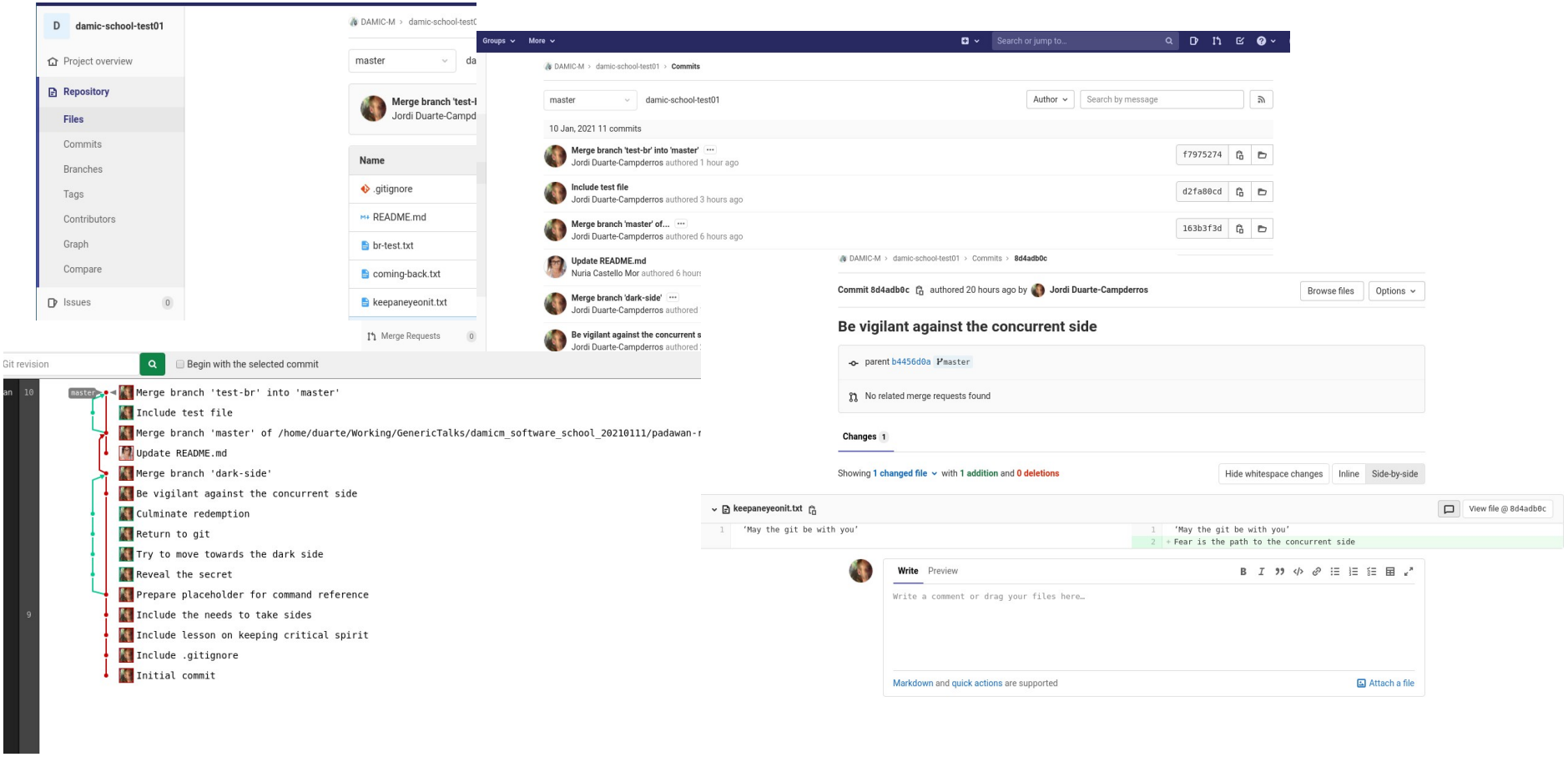


Daily work coding at Gitlab

- Some useful functionalities when developing with the gitlab.in2p3.fr instances:
  - Repository history
  - Issue tracker
  - Wiki
  - CI/CD

Repository History

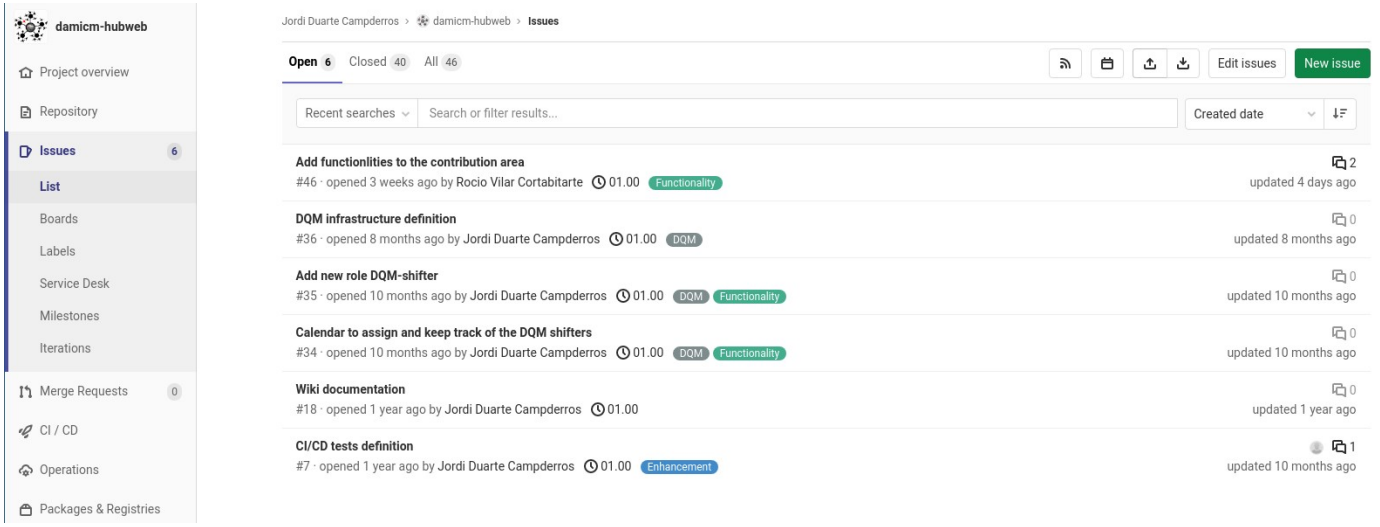
- Everything git can do in command line, plus extra functionalities... browser around



Issue Tracker

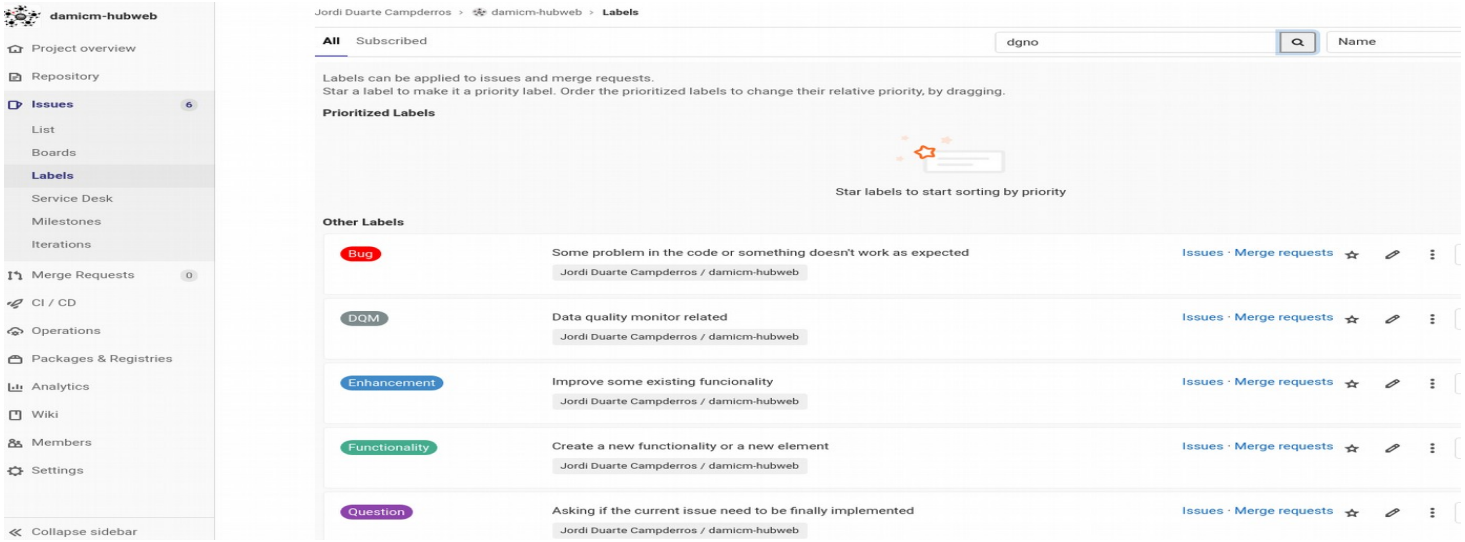
- The fundamental medium to:
  - Discuss the implementation of a new idea
  - Track tasks and work status
  - Accept feature proposals, questions, support requests or bug reports
  - Elaborate on new code implementations

- Centralized, stored for future reference



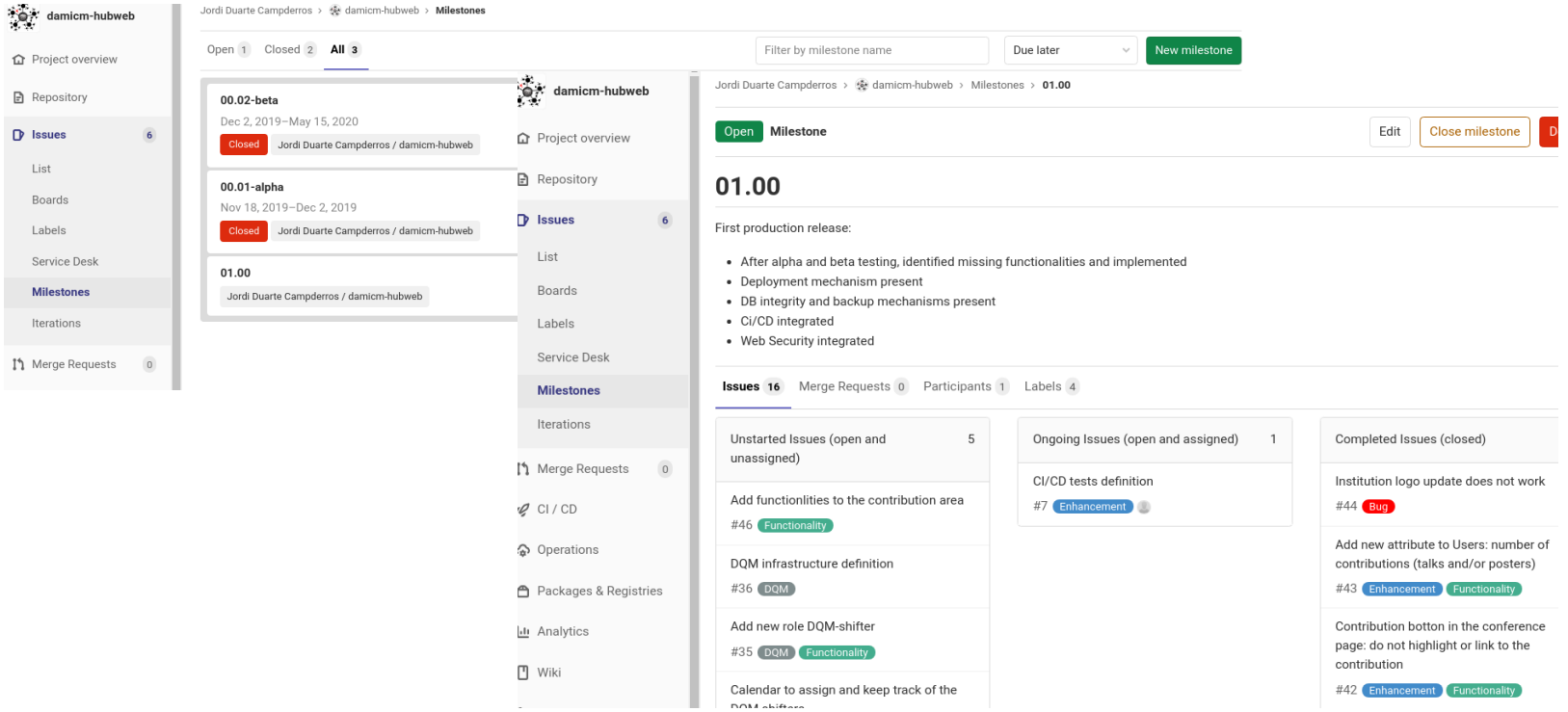
Issue Tracker

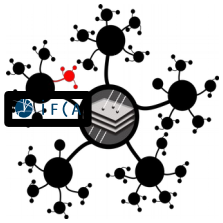
- Labels: help to quickly identify and classify issues



Issue Tracker

- Milestones: a set of issues to be solved
  - Useful to identify releases, tags snapshot, ...





Interlude 4: git tags

- Git has the ability to tag specific points of the repository history (as most of VCSs)
  - For instance, to mark release point
- Annotated tags: provide the tag name (-a) and a tagging message (-m)

```
$ git tag -a v0.1-alpha -m "Alpha release"
```

  - Don't provide the -m option if you want to write a long message, the editor will be launched as in the case of a commit without -m
  - Lightweight tags (git tag v0.1-alpha) do not store extra info, as annotated tags do
- It is possible to tag at any moment (an old commit)

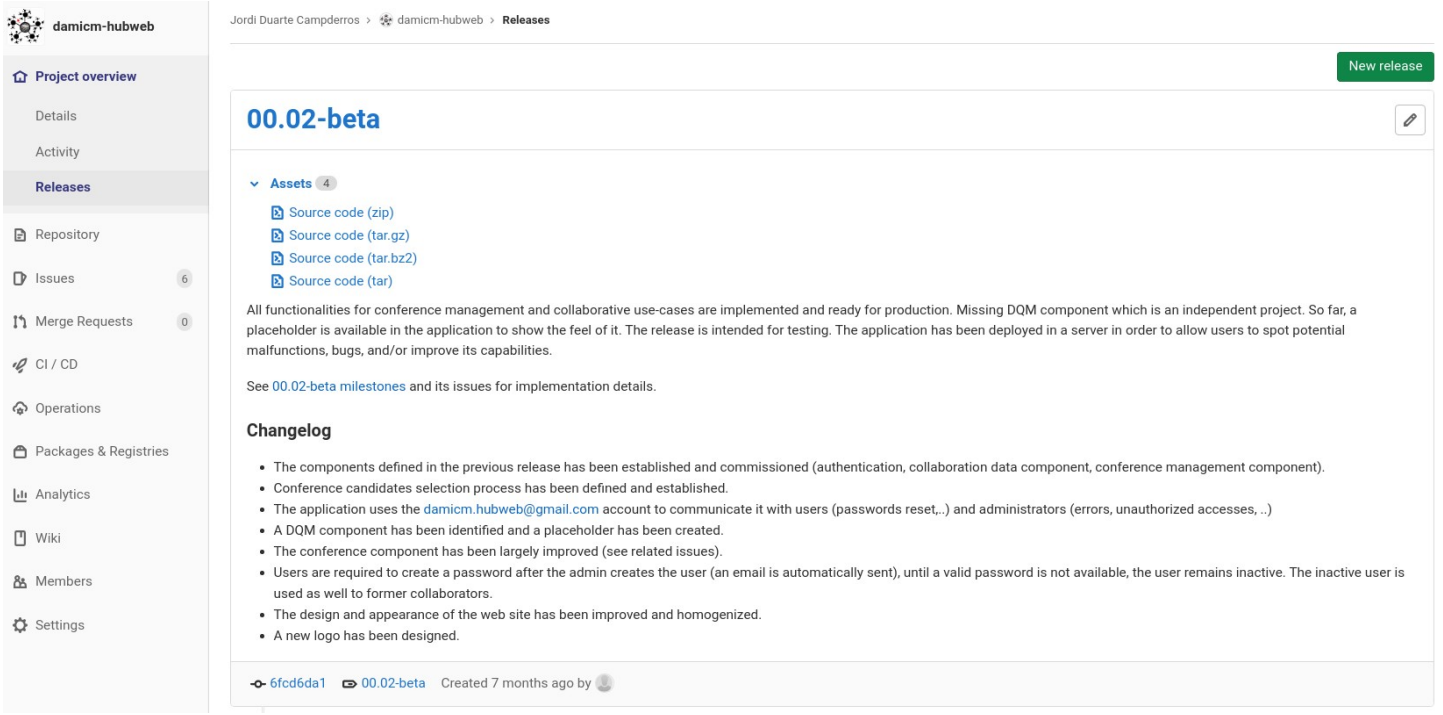
```
$ git tag -a v0.1-alpha <commit-SHA>
```

- Tags are not pushed by default, to push them:

```
$ git push <remote> --tags
```

Issue Tracker

- Releases: snapshot of the source, build output and other metadata associated to a tagged version of the code
  https://docs.gitlab.com/ee/user/project/releases/#create-a-release
  - You can create a release from an annotated tag

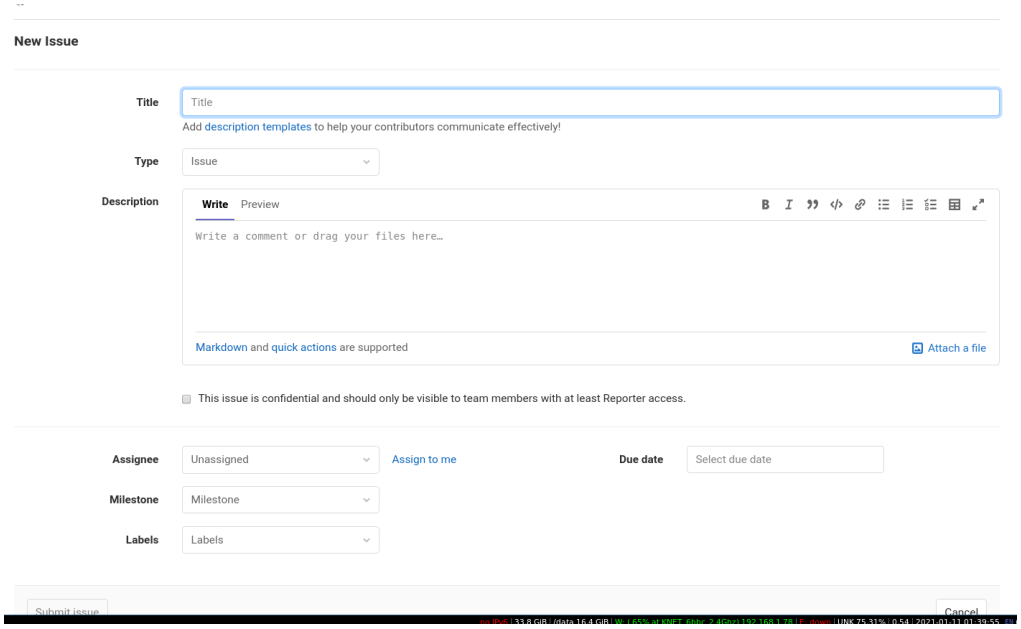


Issue Tracker

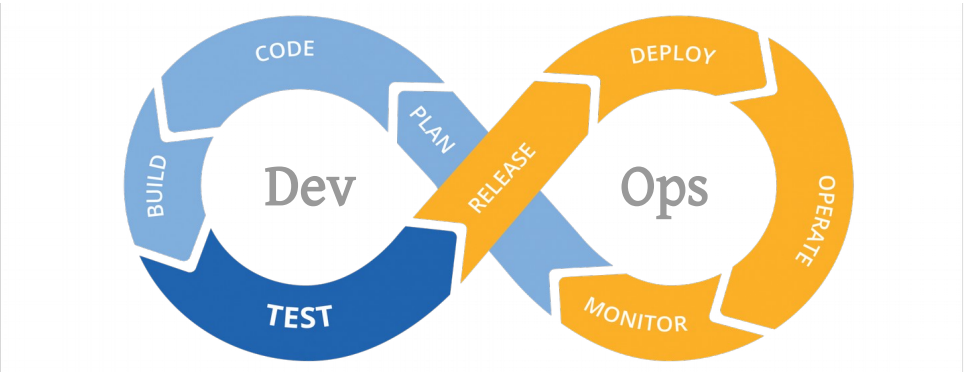
Exercise

```
* Take a look at the list of issues, choose one and try to
participate in the discussion
https://gitlab.in2p3.fr/damicm/damic-school-git/-/issues

* Create a new issue
```

- In order to create a new issue, same best practices than commits/merge request



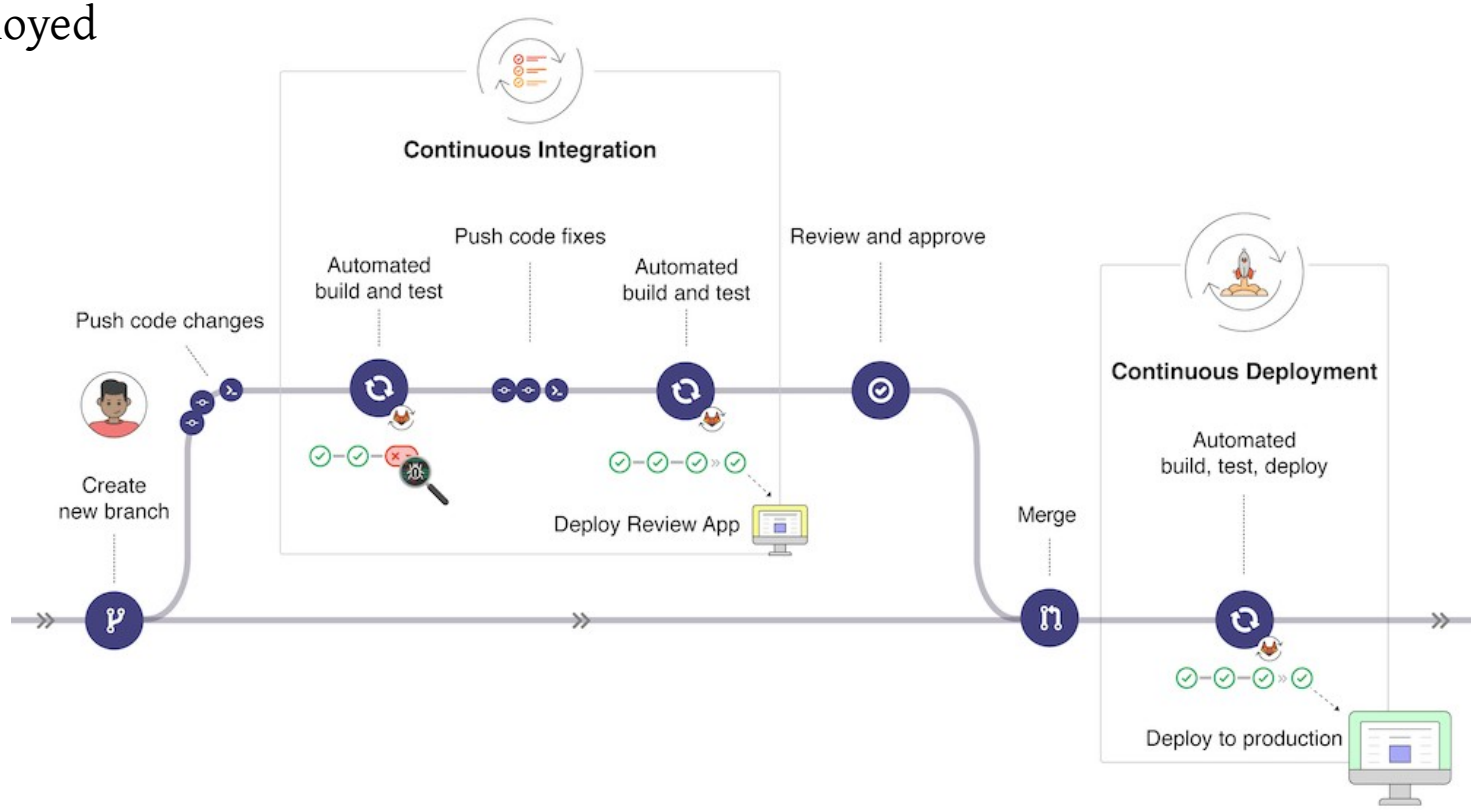
CI/CD

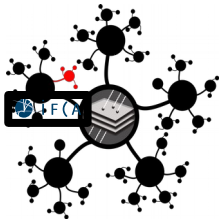


- CI: In every push, the project is tested and validated
- CD: In every push, the project is deployed

CI/CD

- The GitLab CI/CD tool will run some pre-defined scripts every time a push is sent to the repository, which builds and test the software. After revision, the software is deployed





CI/CD

Exercise

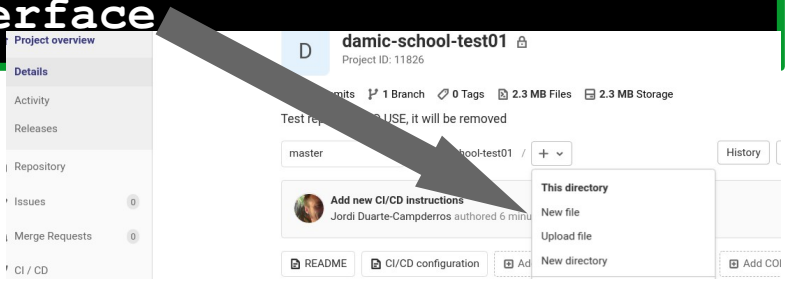
Setup CI in the repository

- Runners: agents that run CI/CD jobs
- Create a .gitlab-ci.yml file → configure instructions for the GitLab CI/CD
  - Structure and order of jobs that runner should execute
  - Decisions runner should make when specific conditions are encountered

CI/CD

Exercise

File creation on the file edit web interface



- Create a .gitlab-ci.yml file

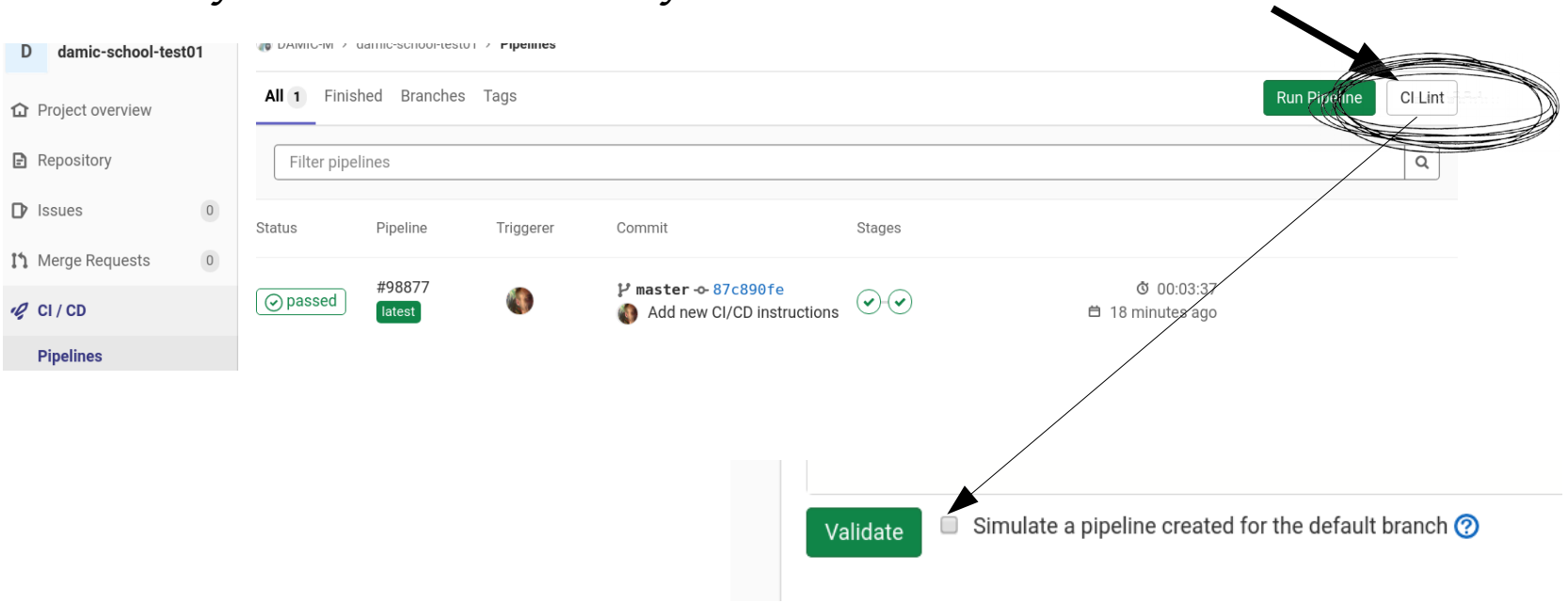
```
# Simple instructions

# it is possible to use docker images
# image: busybox:latest

preparation:
  stage: build
  script:
    - echo "Everything needed to build the software"

training01:
  stage: test
  script:
    - echo "Simulating a test, for instance the existence of the training01 file"
    - echo "ls training-01.txt"

darkside:
  stage: test
  script:
    - echo "Another parallel test here"
    - cat coming-back.txt
```

CI/CD

Exercise

File creation on the file edit web interface

- Very useful to validate the syntax and the structure of the file

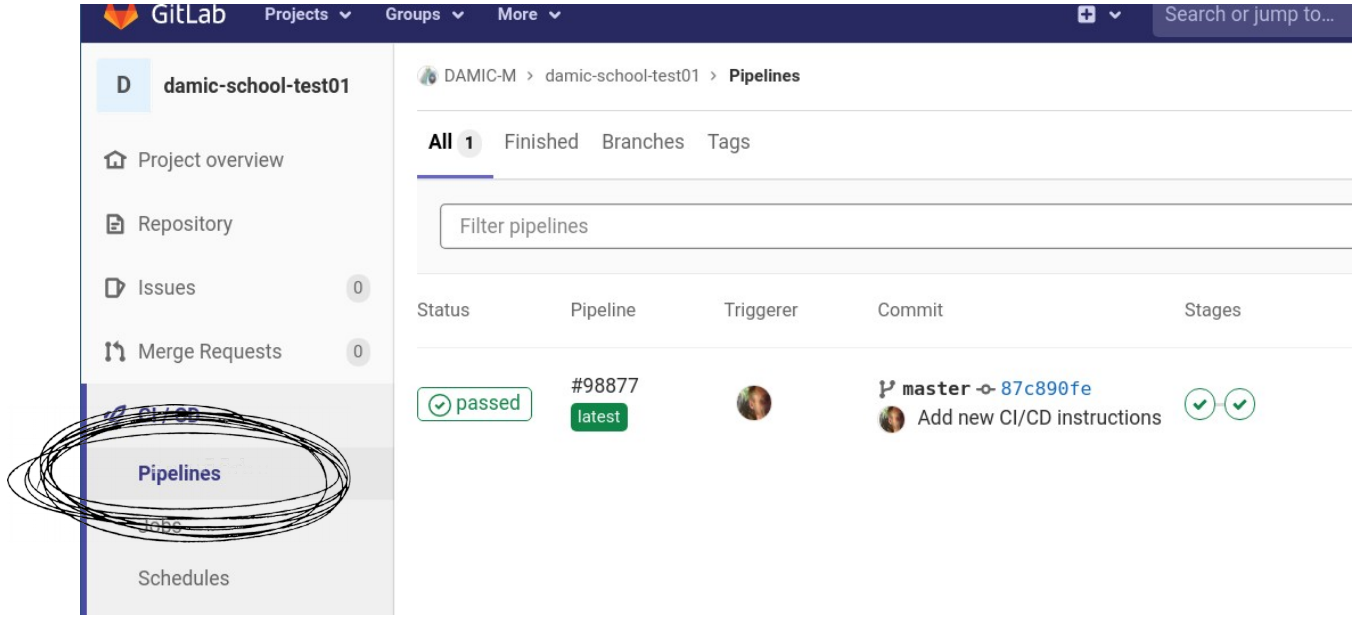


CI/CD

Exercise

Setup CI in the repository

- Once the .gitlab-ci.yml file is pushed to the repo, the Gitlab CI/CD tool is activated

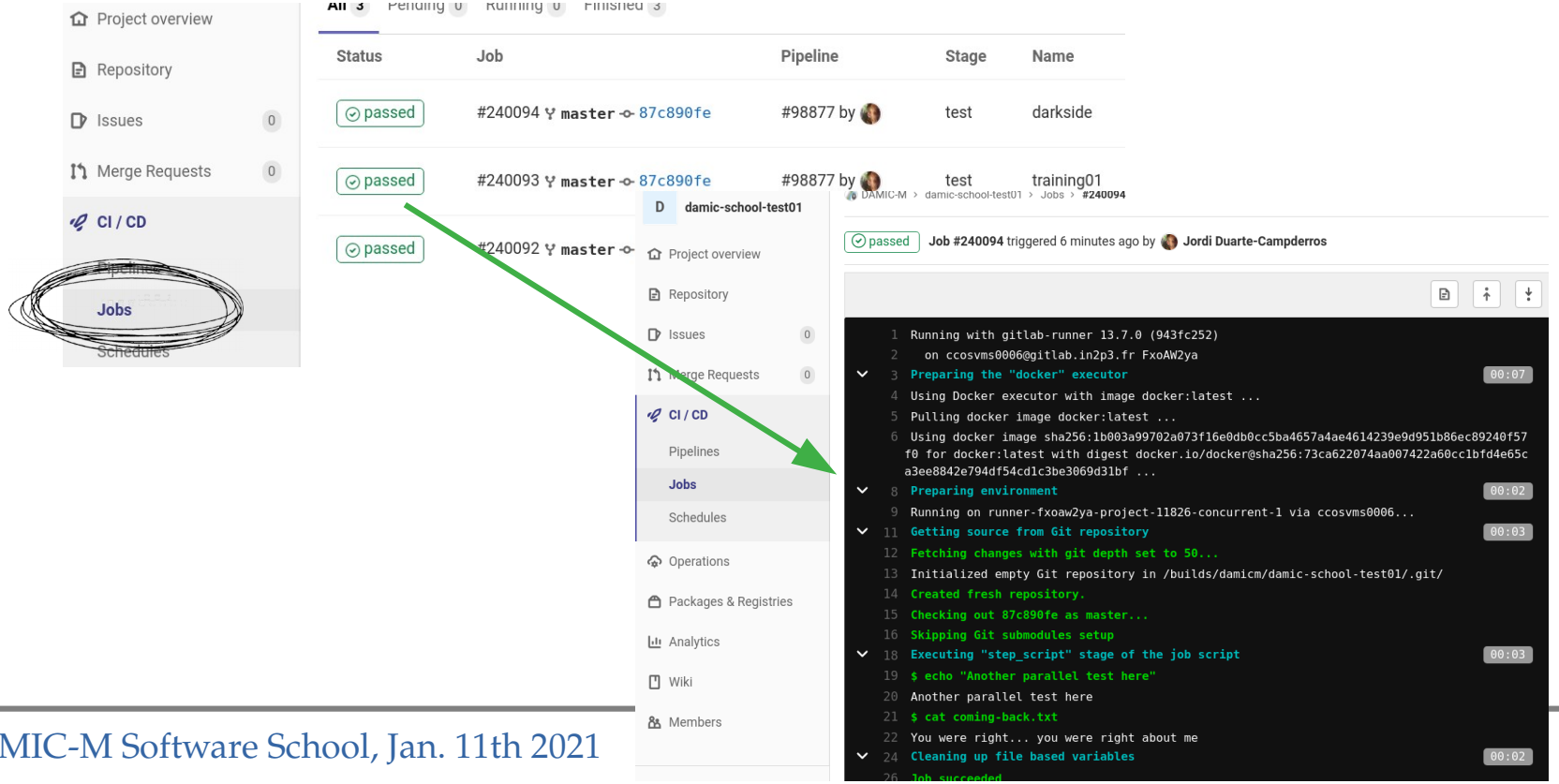


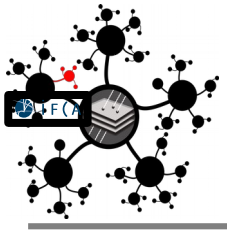
CI/CD

Exercise

Setup CI in the repository

- Jobs can be monitored, as well as test results

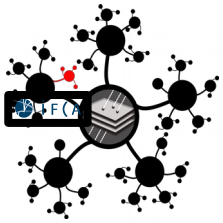




May the

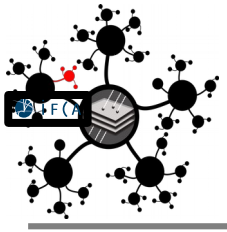
git

be with you



Selected bibliography

- Git: https://git-scm.com/book/en/v2
- Git commands reference:
  - git --help
  - https://git-scm.com/docs
- Gitlab: https://docs.gitlab.com/ee/README.html

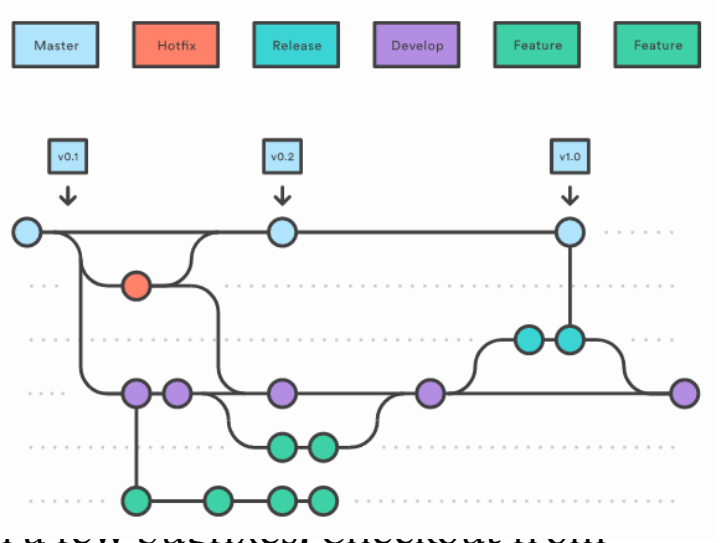


The name

- From the README.md of the project
  https://git.kernel.org/pub/scm/git/git.git/tree/README.md

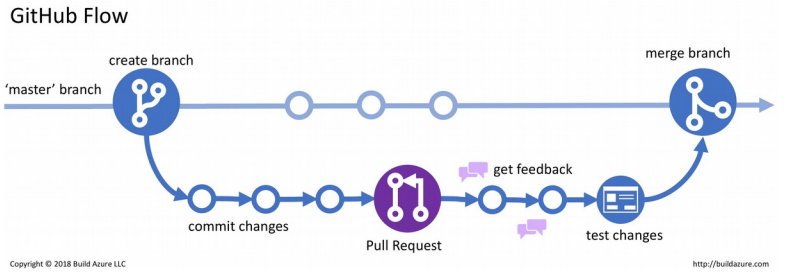
> The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your mood):
>
> - random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
> - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
> - "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
> - "goddamn idiotic truckload of sh*t": when it breaks

GitFlow branching



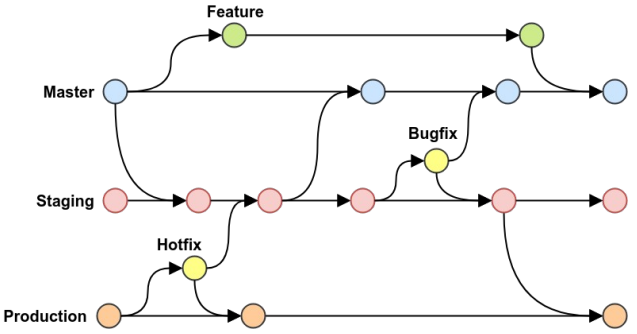
- GitFlow:
  - **Master**: Stable, direct to production
  - **Develop**: Unstable, all feature changes are pushed here
  - **Feature**: Checkout from Develop, push back to it
  - **Hotfix**: Check out from Master, push back to Master and Develop
  - **Release**: Semi-stable, ready to release, following with [illegible] Develop, push back to both Master and Develop
- Master and develop last forever
- Feature branch is created whenever a new feature, or not-urgent bug, is needed for the next release
- Release branch is used to intensively tested before merged to master
- Hotfix branch is used to fix a serious bug which should be merge immediately into master

GitHub branching



- Github flow,
  - **Master:** direct to production, every branch is checkout from here
  - Whenever a work is needed (whether is a feature, a bugfix or any other) a branch is created from master. Once the work is done, it is reviewed and fully tested, and merged into master, and pushed out to production.
    - Using **pull request** mechanism to trigger the review, tests and deployment

GitLab branching



- GitLab flow,
  - **Master:** features and fixes are created in branches starting from and merged to master
    - Using **merge request** to trigger reviews, tests, and checkout to staging branch
  - **Staging:** git stage concept, used as interface between master and production
  - **Production:** long-term, release branch. The tag mechanism is used to identify each version being released