Table of Contents

1. Purpose

The purpose of this document is to describe the usage of the ESO Git repository together with ESO GitLab for external Instrument Control Software (ICS) development.

2. Gitlab basic definition

2.1 Group

With GitLab Groups you can assemble related projects together and grant members access to several projects at once.

Groups can also be nested in subgroups, to better manage people and control visibility.

For example all projects related to HARMONI ICS will be grouped together in one group and HARMONI developers will be members of this group and will have access to all projects inside.

2.2 Project

In GitLab you can create projects for hosting your codebase (Git repository), collaborate on code (review, merge request).

The project's visibility can be set to **Public**, **Internal**, or **Private**:

- **Public**: The project can be accessed without any authentication over https.
- **Internal**: The project can be accessed by any logged in user except external users
- **Private**: The project can only be accessed by the members of the project.

All ICS Gitlab group/projects will be private and visible by approved members only.

2.3 Roles / Permission

Members of group/project can have different roles, which give a different permission scheme, adjusted at the level of the project.

- **Reporter**: Members with this role can only view the code and clone, but cannot carry out push operations.
- **Developer**: Members with this role can also create branches and push to non-protected branches, and submit merge requests.
- **Maintainer**: Members with this role can also push to protected branches (master branch by default) and accept merge requests.
- **Owner**: This is the member who created the group/project and can add other members and subgroups.

While Maintainer is the highest project-level role, some actions can only be performed by a personal namespace or group owner, or an instance administrator, who receives all permissions.

2.4 Fork

"Forking" is not a Git operation but a GitLab feature.

A **fork** is a personal copy of the repository and all its branches, which the user creates in a namespace of his choice. In this way the user can make changes to his own fork and submit them through a merge request to the original repository, where he cannot commit directly (not enough permission).

2.5 Merge Request

A Merge Request (**MR**) is a *request* to *merge* one branch into another.

MRs in GitLab allow to visualise and collaborate on proposed changes to source code.

There are two main ways to have a merge request flow with GitLab:

1. Working with protected branches in a single repository.
2. Working with forks of a project with restricted permission.

With the *protected branch flow* (case 1) everybody works within the same GitLab project.

The project maintainers get Maintainer access and the regular developers get Developer access.

The maintainers will define “protected” branches; by default master branch is protected.

The developers push feature branches to the project and create MRs to have their feature branches reviewed and merged into one of the protected branches. By default, only users with Maintainer access can merge changes into a protected branch.

With the *forking workflow* (case 2) the maintainers get Maintainer access and the regular developers get Reporter access to a centralised “authoritative” repository, which prohibits them from pushing any changes to it.

Developers create forks of the centralised project and push their feature branches to their own forks. To get their changes into master they need to create a merge request across forks.

The two merge request flows will be used during ICS development, protected branches for the ongoing development and the fork workflow for the delivery to ESO (see 4)

3. Gitlab at ESO

3.1 Gitlab personal account

All consortia developers will have a personal external account in the ESO Gitlab:

https://gitlab.eso.org

External users will use the “Standard” tab in login page (https://gitlab.eso.org/users/sign_in) to connect (not ESO ADS).

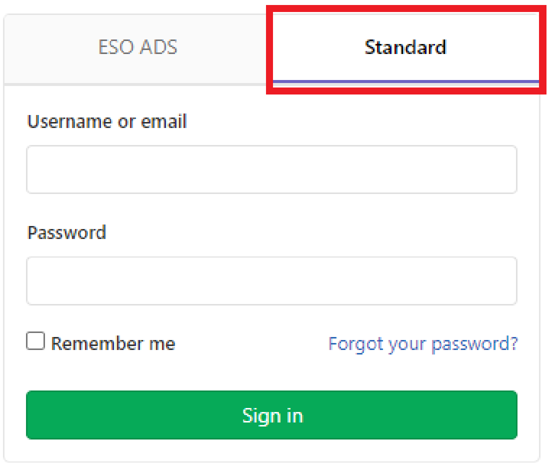


Figure 1: GitLab Sign-in

The account gives access to the ICS Framework, https://gitlab.eso.org/ifw and it is a member of the Instrument Group.

If a user needs an account, he should contact the ESO ICS contact person, assigned to the project.

3.2 Gitlab Instrument Group

For each Instrument, a **private** Gitlab group has been created: https://gitlab.eso.org/*<instrument-name>*

Example: https://gitlab.eso.org/harmoni

All instrument consortia ICS developers are members of the Instrument group with the role **Reporter.** The ESO ICS software responsible following the ICS development will have the role **Maintainer**.

The <instrument-name> namespace will host the Instrument **Git projects.** This will be the production area during all development phases, where the software will be delivered to ESO for regular deliveries approximately every 3 months and at each milestones (mid-term review, PAE, etc).

3.3 Instrument Control Software Git projects

The Instrument group will contain the Instrument Gitlab projects as described in document ESO-351071. The standard INS structure forsees at least the following Git projects:

<*prefix*>-ics : storing the source code of the ICS

<*prefix*>-aocs : storing the AOCS source code.

<*prefix*>-resource : storing configuration data of the instrument

This structure is flexible and instrument may need to add additional repositories to store the source code of other components.

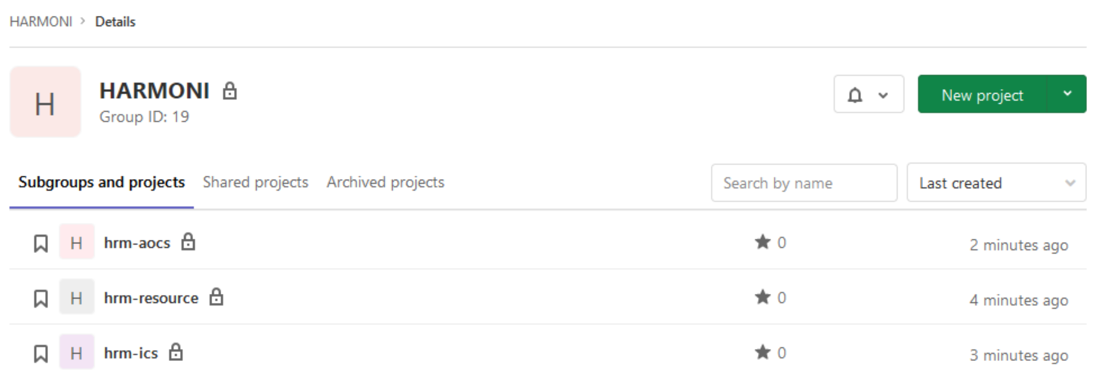
Example of basic structure for HARMONI :

- hrm-ics
- hrm-aocs
- hrm-resource

The group configuration is such that only maintainer (ESO) can create new Gitlab projects.

By default those three main Git projects will be created by ESO with a basic set of files and should be completed by consortia developers following the standard structure defined in the document ICS Software Specification (see ESO-351071 section 2.5).

Example for HARMONI, the following projects will be created and be accessible from: https://gitlab.eso.org/harmoni



hrm-aocs	0	2 minutes ago
hrm-resource	0	4 minutes ago
hrm-ics	0	3 minutes ago

Figure 2: Example of Gitlab Projects.

Each Git project will be created with an empty README.md file, which should be completed later to describe the project.

Some Git related files (.gitattributes, .gitignore) will also be added as example, and should be completed (see also https://github.com/github/gitignore).

Example:

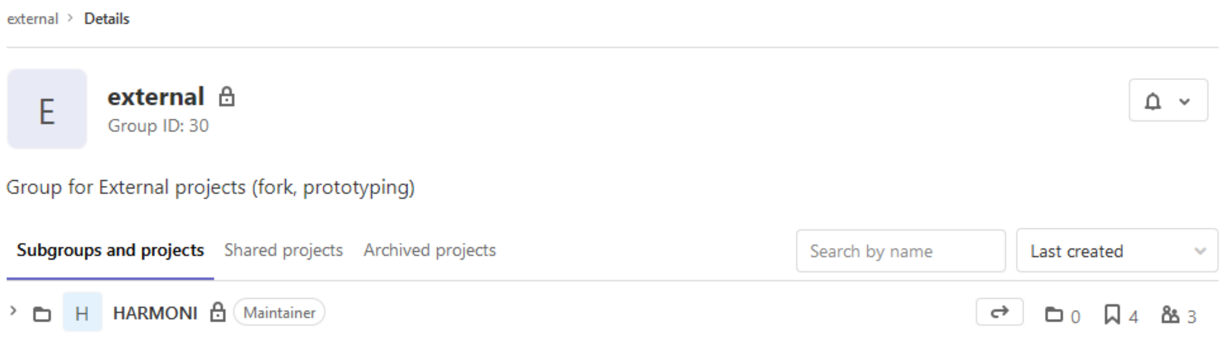
```
<prefix>-resource
    ├── .gitattributes   # example: git-lfs setting for FITS files
    ├── README.md
<prefix>-ics
    ├── .gitignore # example: ignore waf directories (build, lock-waf)
    ├── README.md
```

3.4 Gitlab Instrument private area

Consortia developers are also members of another private subgroup "*external/<instrument>*", with the role **Developer** by default, or **Maintainer** for the consortia ICS responsible.

In this area, consortia developers can create prototypes to be shared with others but also the **fork** of the production Git project (see workflow). In this group, ESO members, following the ICS development, will have the role **Reporter**.

Example: https://gitlab.eso.org/external/harmoni


external › Details

external
Group ID: 30

Group for External projects (fork, prototyping)

Figure 3: Private Area.

4. Git/Gitlab ICS Workflow

4.1 Description

The ICS workflow would be based on a forking workflow to manage the deliveries to the ESO Production Repository.

The Production Repository will be located in the <instrument-name>/<repo-name>.

To use the ESO Gitlab repository,the consortia ICS maintainer shall create a fork of the production repository in the private area external/<instrument> (see details in section 4.2).

Once the Fork Repository is created, it will be cloned by each developer to create a Local Repository on the local machine (workstation) (see

Figure 4: Creation of Fork and Local Repositories).

Example:

```
$ git clone https://gitlab.eso.org/external/harmoni/harmoni.git
```

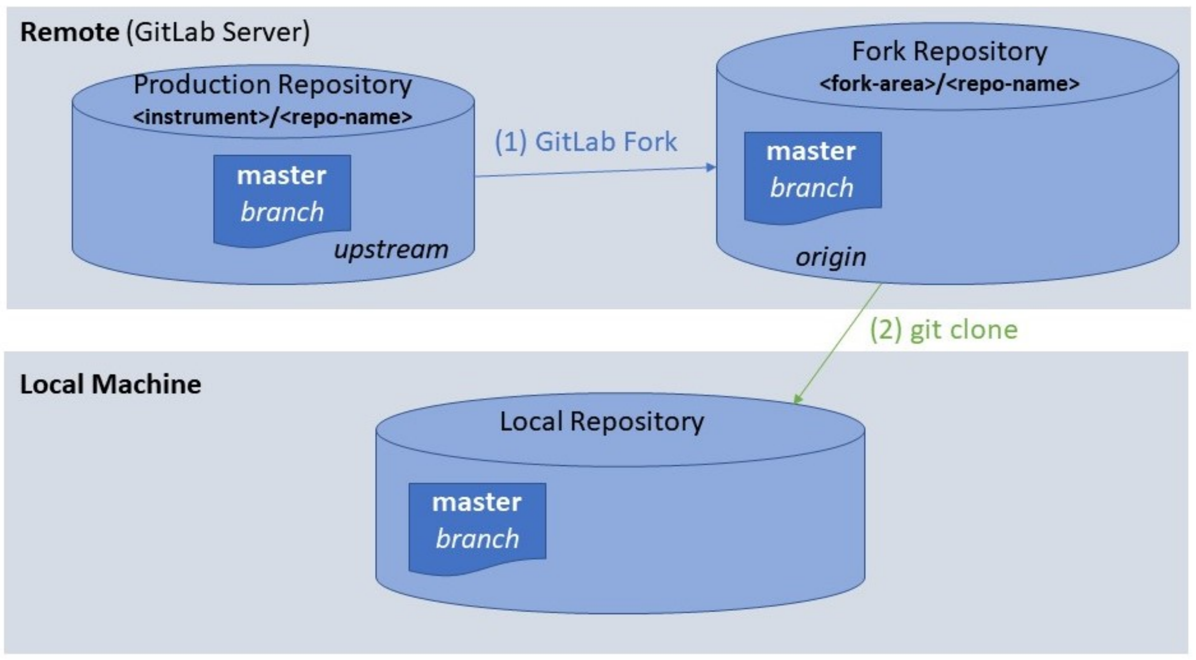


Figure 4: Creation of Fork and Local Repositories.

The following terms are used in Git to identify the different remote repositories:

Origin: Shorthand name used by Git for the remote repository that a project was originally cloned from (Fork repository in

Figure 4).

Upstream: Shorthand conventional name in Git for the remote repository that was forked (Production repository in

Figure 4).

During the development phase, the Local Repository and Remote Origin will be used following the branching strategies (or protected branch flow).

The following will describe mainly how to set up the ICS workflow for delivering to ESO, the detailed internal workflow of the Fork Repository will be defined and managed by consortia. Depending on the size of the development team or how much the ESO Gitlab will be used by the consortium (exclusively or together with another Gitlab server) different branches strategies could be used.

However as basic workflow (

Figure 5: ICS Workflow), we suggest to create *branches*(2) for new features in the Local Repository. Subsequently to *push*(3) the feature branch to the origin and then merge the feature branch into the origin master branch, by means of a Gitlab *merge request*(4). And in order to synchronise the Local Repository with Origin Master changes, a git *pull*(1) will be needed.

When the Origin Master is ready to be delivered to ESO, a Gitlab *merge request*(5) to the Production Repository should be done.

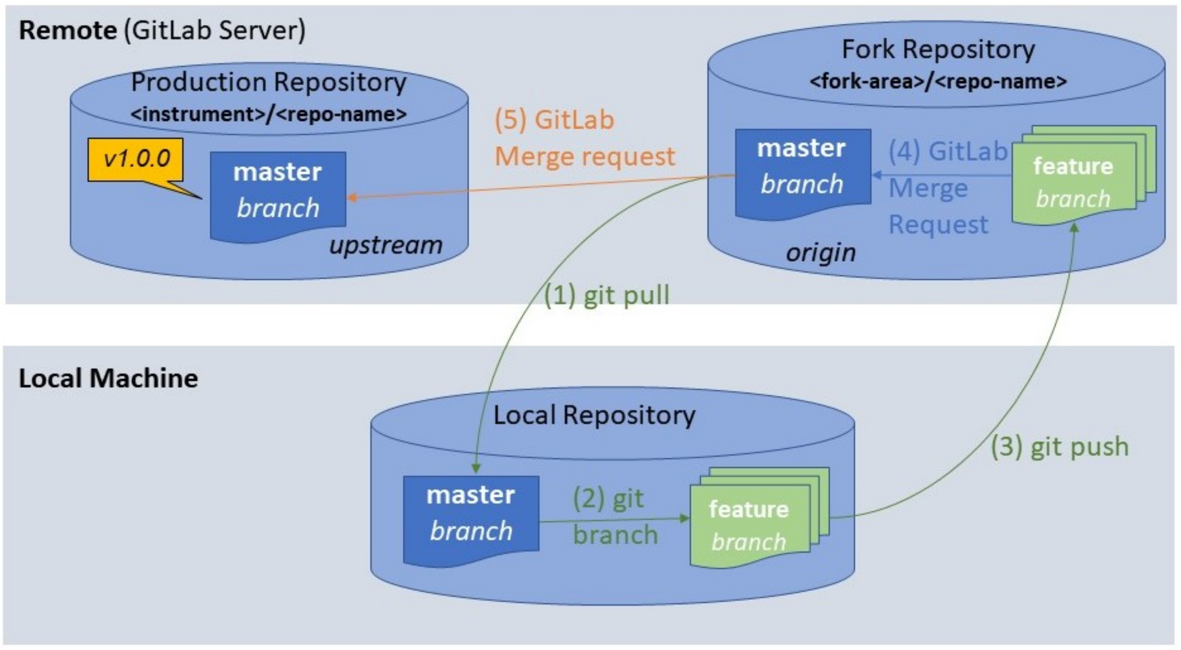


Figure 5: ICS Workflow.

During the maintenance/production phase, the workflow will certainly need to be more complex, and a synchronisation of the Local Repository with the remote Upstream Master also needed.

4.2 Creating the fork

Only one fork will be created and used as the common repository by all consortia ICS developers.

Forking a project, in most cases, is a two-step process.

1. On the project's home page, in the top right, click the Fork button.

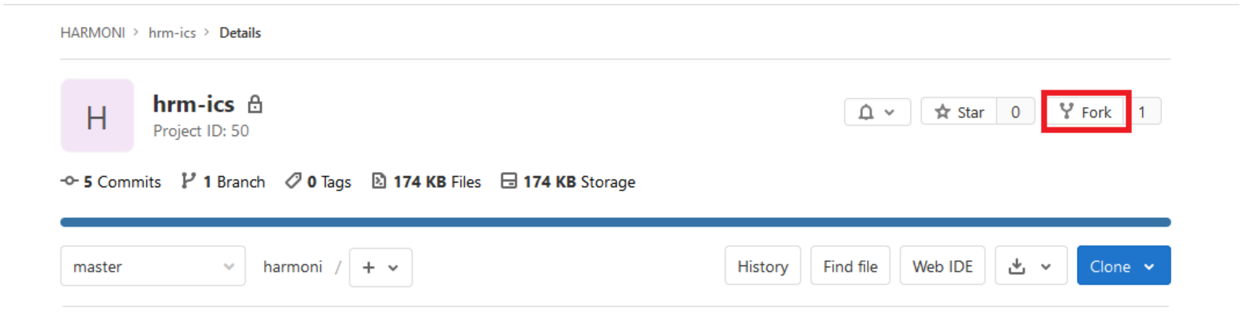
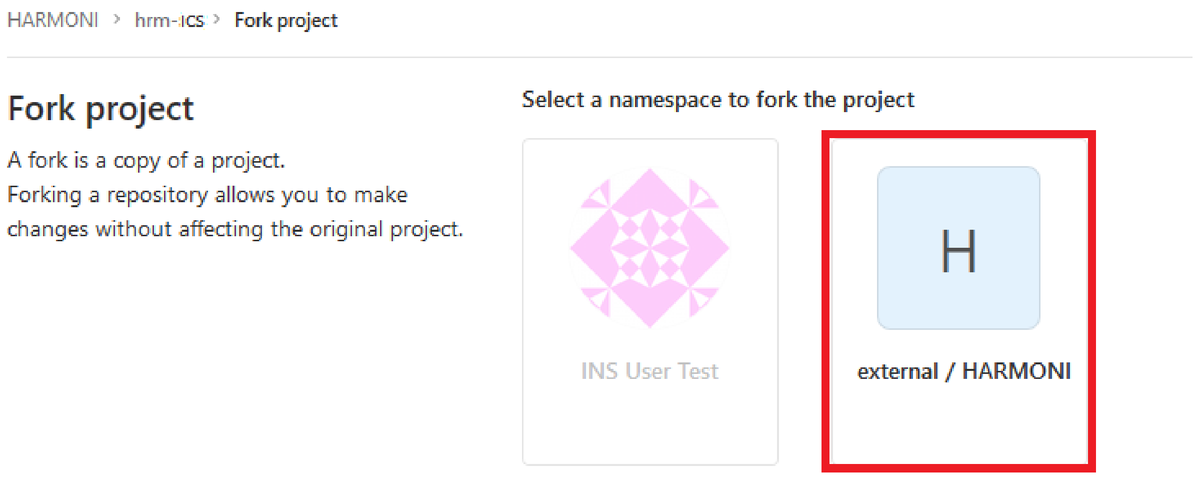


Figure 6: Fork creation.

2. Select the external namespace to fork to.


HARMONI > hrm-ics > Fork project
Fork project
Select a namespace to fork the project
INS User Test
H
external / HARMONI

A fork is a copy of a project.
Forking a repository allows you to make changes without affecting the original project.

Figure 7: Fork location.

Note: By default, an external account cannot create a project in the personal namespace.

Once the fork is created, developers can use the project as **origin** and clone, pull/push to it.

When the code in the fork is ready to be delivered to ESO, the MR can be created, to send it back to the **upstream** project.

4.3 Create the Merge Request

The Gitlab MR should be always used when merging to the Production Repository to take advantage of the Gitlab MR features, in particular code reviews and automatic testing

As shown in

Figure 5: ICS Workflow, the MR when delivering, will be created from the Fork Master Branch to Production Master Branch.

MRs will also be created from the Feature Branch (feature or bug fix branch) to the Master Branch within the Fork Repository. In this case it is recommended to delete the Fork Feature Branch after the merge, to avoid multiplication of unused branches in the Fork repository, and to use the JIRA issue number in the name of the branch when possible.

Example of Git branch naming convention

bugfix-<*issue-number*>-<*short_descriptor*>

feature-<*issue_number*>-<*short_descriptor*>

To create the MR in Gitlab, from the Fork Repository, select **Merge Requests** and **New merge request** as shown in Figure 8.

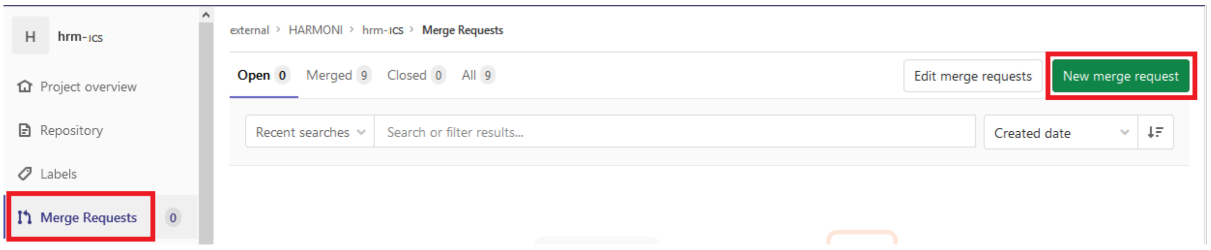


Figure 8: Merge request creation screen.

Then select the source branch to be used for the merge

For the **Source branch**, choose the forked project's master branch. For **Target branch**, choose the original project's master branch (Figure 9).

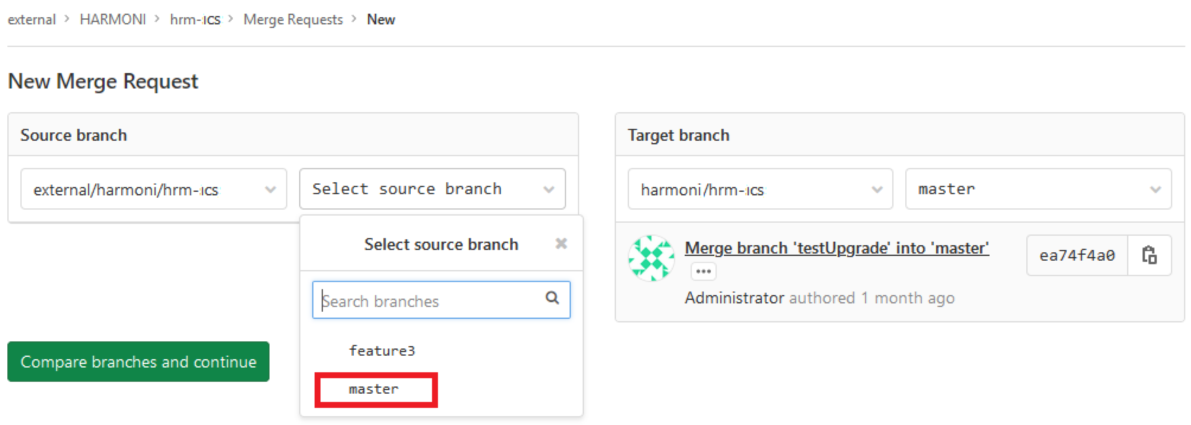


Figure 9: Merge request Source branch selection screen.

In the **New Merge Request** screen it is necessary to fill in some fields.

Assignee: The name of the ESO responsible (mandatory), who will review and accept the merge.

Milestone: Reference to the delivery milestone (optional).

Labels: To categorize the MRs (examples: bug, documentation, patch, review…) (optional).

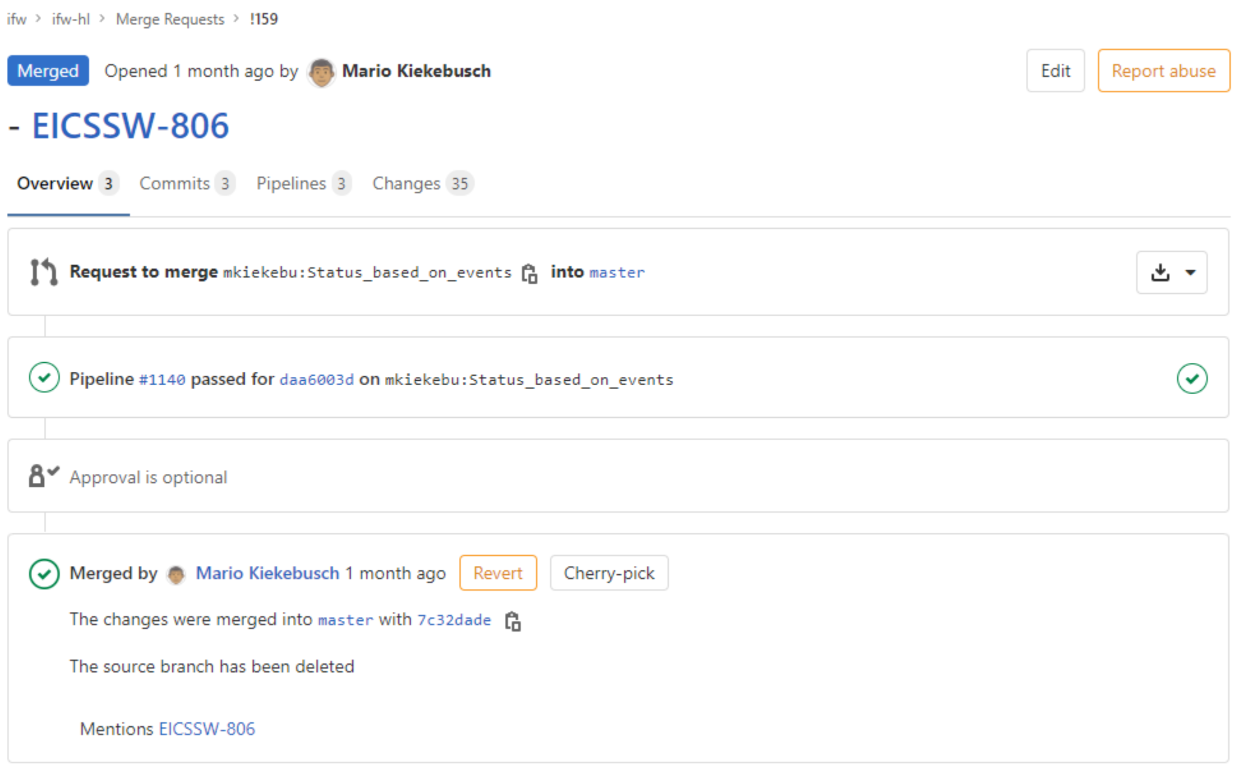
Merge options (recommended)

- Squash commits when the MR has been accepted.
- [Delete source branch when MR is accepted]. *Only when the MR is done from the Fork Branch*

Then click **Submit merge request** to conclude the process. Once successfully merged, the changes are added to the repository and branch merged into (Upstream Master).

Note: If an MR is not yet ready to be merged, it can be prevented from being accepted by flagging it as a draft (Draft prefix) and the flag shall be removed only when it is ready. This provides the possibility to propose a MR, asking for some preliminary checks (code review, test)

Example of MR approved Figure 10


ifw > ifw-hl > Merge Requests > !159

Merged Opened 1 month ago by Mario Kiekebusch

- EICSSW-806

Request to merge mkiekebu:Status_based_on_events into master

Pipeline #1140 passed for daa6003d on mkiekebu:Status_based_on_events

Approval is optional

Merged by Mario Kiekebusch 1 month ago

The changes were merged into master with 7c32dade

The source branch has been deleted

Mentions EICSSW-806

Figure 10: Merge Request approved screen.

For each MR to the Production Repository, a pipeline will be executed and will run a Jenkins job. The project is configured in a way such that the MR can be accepted only if the pipeline succeeds. The content of the pipeline (Jenkins job) is defined per Git project.

The Jenkins job, called by the MR pipeline, will execute by default the build, unit test (*waf build ,waf test),* inline documentation generation but also integration tests, and other quality control checks defined by ESO (coverage, Valgrind/sanitizers, coding standards).

In addition to the MR Jenkins Job, a Continuous Integration (CI) Jenkins job will also be executed from the master branch of the Production Repository.

The infrastructure for the CI, MR Pipeline (in Jenkins) will be provided by ESO only for the Production Repository. It is the responsibility of the consortium to replicate it for their Fork Repository, with the help of ESO if needed.