



## D1.2

# Creation of code repository and versioning system

Adham Hashibon and Yoav Nahshon

## Document information

Project acronym:	INTERSECT
Project full title:	Interoperable Material-to-Device simulation box for disruptive electronics
Research Action Project type:	Accelerating the uptake of materials modelling software (IA)
EC Grant agreement no.:	814487
Project starting / end date:	1 <sup>st</sup> January 2019 (M1) / 31 <sup>st</sup> January 2022 (M37)
Website:	<a href="http://www.intersect-project.eu">www.intersect-project.eu</a>
Final version:	17/07/2019
Deliverable No.:	D1.2
Responsible participant:	Fraunhofer (participant number 5)
Contributing Consortium members:	FRA, EPFL, AMAT
Due date of deliverable:	31/07/2019
Actual submission date:	30/07/2019
Dissemination level:	PU - Public

**Authors:** Adham Hashibon and Yoav Nahshon

**To be cited as:** A. Hashibon and Y. Nahshon (2019): Creation of code repository and versioning system. Deliverable D1.2 of the H2020 project INTERSECT (final version as of 17/07/2019). EC grant agreement no: 814487, Fraunhofer Gesellschaft Zur Foerderung Der Angewandten Forschung E.V., Freiburg, DE.

## Disclaimer:

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

## D1.2: Code repository and versioning system

### Contents

<b>Executive Summary</b>	4
<b>Next steps</b>	4
<b>Description of Work done</b>	6
<b>Results</b>	7
<b>The Intersect Code Repository on GitLab</b>	7
Introduction to Git and its eco system	7
How to access the INTERSECT GitLab repository:	8
On Gitlab Terminology	9
Creating a new repository	10
Further information on GitLab	10
<b>General Collaborative Workflow</b>	10
<b>The INTERSECT Developers Manifest</b>	13
<b>Additional information</b>	14
Creating Issues	14
Branching and association with issues	15
How to create an issue-associated branch	15
How to work locally on the newly created branch	15
Committing & Pushing	15
How to commit and push changes	16
Creating Merge Requests	16
How to create a merge request	16
Markdown language	16
<b>Team Communication: Mattermost</b>	16
The Mattermost APP for mobile and Desktop	17
Mattermost functionality in a nutshell	18
Specifically mentioning and addressing members in messages	20
Formatting	20
<b>REFERENCES</b>	27

## Executive Summary

This task comprises the activities devoted to the definition of the quality standards for the software developed in the project, as well as the codes comprised in the **Interoperable Materials-To-Device (IM2D)** box and the quality assurance procedures, in agreement with the EU White Paper for standards of modelling software development [1].

Specific actions performed are

- (i) Creation of a central code repository and versioning system, see
- (ii) Figure 1 and
- (iii) Figure 2
- (iv) Organization of software development
- (v) Definition of a testing protocols and providing the infrastructure needed for each of the components of the **IM2D** box, of the interfaces, wrappers and plugins for interoperability;
- (vi) Central code repository and versioning are controlled through GitLab system, based at Fraunhofer
- (vii) Provision of a battery of tests to feed the protocol defined above
- (viii) Provision for production of software documentation, including description of models, physical equations and material relations; code architecture, installation and user guide, and test execution examples.

For (v) and (vi) more work needs to be done in tandem with the code developed, as the testing and documentation suits need to be customized per each case.

Furthermore, to ensure coherent and efficient collaboration a beginner's guide and general introduction to software development tools is given together with a **developer manifesto**. The manifest is to be followed by all developers and contains explanation of how to use Gitlab and related collaboration tools and will be updated after feedback from all developers with common naming conventions and semantic versioning that are most suitable to all partners in order to: 1. Ease the review process by reducing confusion stemming from non-standard naming (e.g., naming a variable as a method), and 2. To enable better long term manageability and clarity of the codes thus assisting in their upscaling and uptake by industry.

Additionally, a state-of-the-art **self-hosted team communication platform** has been provided and set up using Fraunhofer hosted Mattermost server that shares the same sign on credentials with Gitlab. This allows team members to exchange rapid short messages related to various topics before starting issues on GitLab. It is hooked up with the GitLab system so that various developers can interact and discuss related issues and other project matters on Mattermost. It is provided as an added feature.

## Next steps

Continuous maintenance and updates are performed for the entire project duration. Beyond the project, the open parts of the code will keep being maintained as open source project.

Additionally, the partners can maintain their own repositories for commercial exploitation in tandem with consortium decisions.

Next steps for the coming period constitute of

1. Conducting online tutorials for all partners that need introduction to the tools, and
2. Customization of the continuous integration, docker registry and integration and deployment workflows so that development and official releases of the various components can be automatically managed through this task. This essentially addresses the objective (v) provision of a battery of tests and (vi) provision for production of software documentation

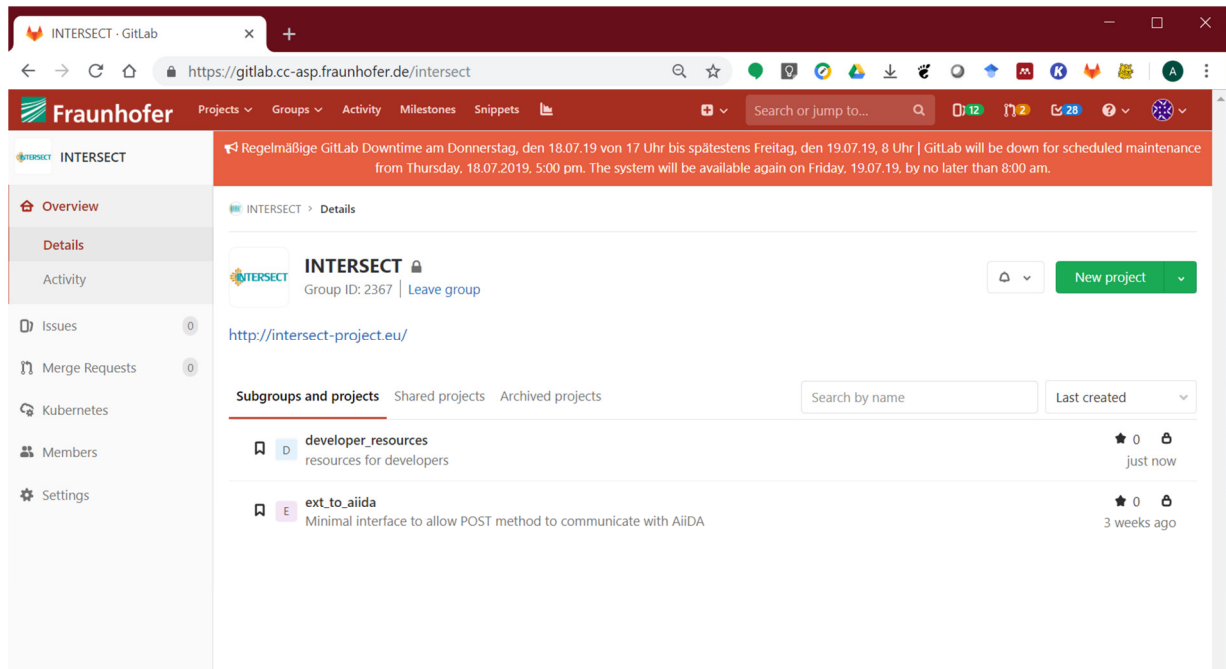


Figure 1: A snapshot showing the main settings of the INTERSECT GitLab space. Every partner has rights to open new repositories (called projects on GitLab) and add collaborators to it (from within the INTERSECT team).

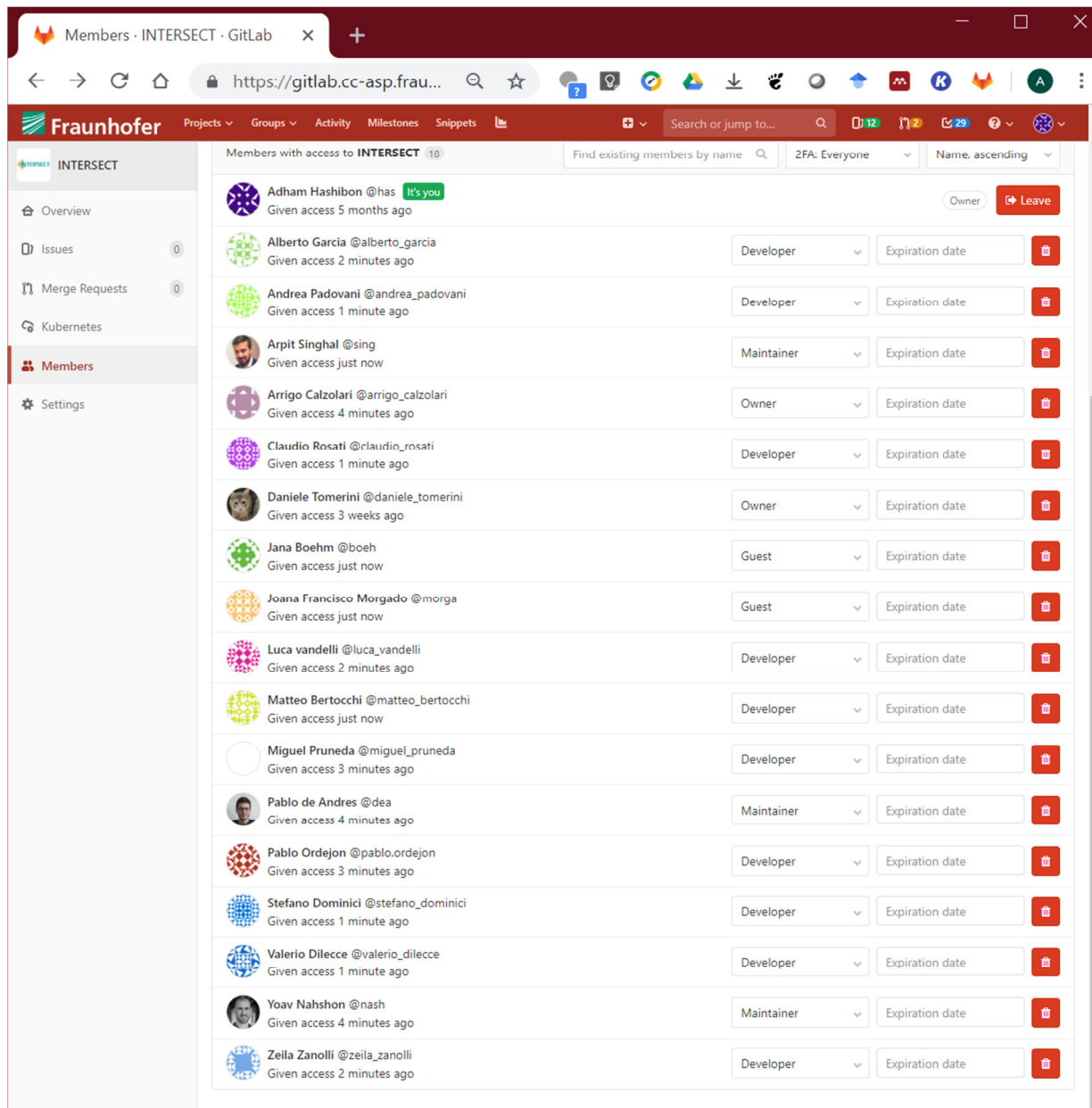


Figure 2: snapshot of the members setting page of the GitLab repository.

## Description of Work done

Work done in this task includes:

- (i) Creation and setup of a central code repository and versioning system on Fraunhofer GitLab servers hosted in Europe;
- (ii) Creating a Developer Manifest and initial how-to and documentation (as part of this deliverable) covering login and basic work to partners to rapidly get started using the tools

- (iii) Creating a developer resources repository for exchange of documentation relevant to the developers
- (iv) Providing the infrastructure needed for each of the components of the IM2D box, of the interfaces, wrappers and plugins for interoperability;
- (v) Support for project partners to login and use the system

## Results

### The Intersect Code Repository on GitLab

#### Introduction to Git and its eco system

Git is the most widely used version control system and a de-facto standard. Git is an open source project originally developed in 2005 by Linus Torvalds the famous creator of the Linux operating system kernel (according to some accounts, the first version took about in just 3 days!). Git is actively developed and supported by numerous stakeholders, including Microsoft, Google, Facebook, etc.

(see e.g., <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F> and <https://www.atlassian.com/git/tutorials/what-is-git>)

Git is based on a distributed, non-centralized architecture (hence Distributed Version Control System - DVCS).

**In Git, every developer has a full working copy of the entire repository with the full history of all changes made. Git provides powerful, yet simple, tools to merge, and track changes across developers.**

The big advantage is: every developer can work independently more rapidly, however still we need means to allow for better **enhanced collaboration between developers working on the same project** or collection of projects (like in INTERSECT) and users giving feedback designers, architects, etc. This can be done per email or phone, but best through dedicated online collaborative code development platforms based on Git.

Two such widely used platforms are *GitHub* and *GitLab*.

**What is GitHub (From Wikipedia)** GitHub is an American company that provides hosting for software development version control using Git. It is a subsidiary of Microsoft, which acquired the company in 2018 for \$7.5 billion [2]. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project [3].

**What is GitLab? (from Gitlab.com)** GitLab is a single application for the entire DevOps lifecycle. This makes GitLab unique and makes Concurrent DevOps possible, unlocking your organization from the constraints of a pieced together toolchain. Join us for a live Q&A to

learn how GitLab can give you unmatched visibility and higher levels of efficiency in a single application across the DevOps lifecycle.

Both share much of the same functionalities and have similar features but differ slightly, though for our needs, there is a major difference is that we can easily host our own Gitlab but not our own GitHub (commercial, requires a license). For our EU project it is important that the data is hosted on EU servers complying with the EC regulations (e.g., GDPR). For this, since GitLab is an open source project, users can have their own instance of Gitlab hosted on the servers of their choice. GitHub does not provide such an option (they do provide an enterprise edition that can be self-hosted, but not an open source one).

Moreover, GitLab integrates with docker registries, continuous integration and other tools that enable a full DevOps workflow to be set up on our own servers. On GitHub, even if we get a free hosted account or pay for it, we still need to configure and buy additional continuous integration and docker registry services (the free options are too limited for the project needs in the long run).

For these reasons we choose GitLab that is hosted in Europe (Germany) on Fraunhofer infrastructure, open to all partners, includes docker registry and can be configured at ease to use our own servers for unlimited continuous integration for free.

#### How to access the INTERSECT GitLab repository:

A git repository is set up and can be accessed at the address:

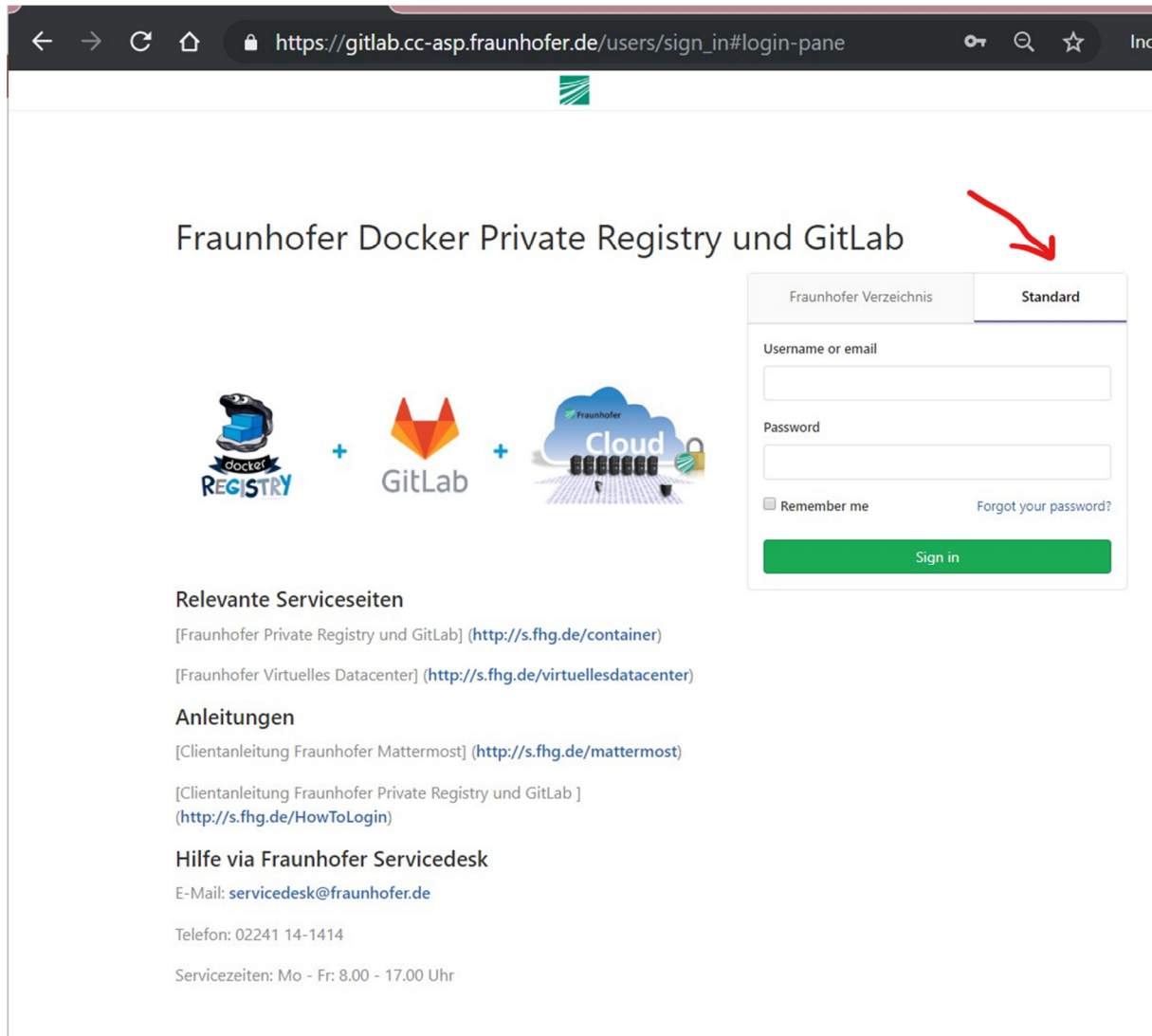
<https://gitlab.cc-asp.fraunhofer.de/intersect>

which directs users to the standard login page, see Figure 3.




Accounts have been opened to all partners. Additional accounts can be further opened by Fraunhofer, on request.

Once signed in, click on Groups and pick INTERSECT, you are then in the project's git space on GitLab (called Group in the GitLab terminology).





Fraunhofer Docker Private Registry und GitLab

Relevante Serviceseiten

[Fraunhofer Private Registry und GitLab] (<http://s.fhg.de/container>)

[Fraunhofer Virtuelles Datacenter] (<http://s.fhg.de/virtuellesdatacenter>)

Anleitungen

[Clientanleitung Fraunhofer Mattermost] (<http://s.fhg.de/mattermost>)

[Clientanleitung Fraunhofer Private Registry und GitLab ] (<http://s.fhg.de/HowToLogin>)

Hilfe via Fraunhofer Servicedesk

E-Mail: [servicedesk@fraunhofer.de](mailto:servicedesk@fraunhofer.de)

Telefon: 02241 14-1414

Servicezeiten: Mo - Fr: 8.00 - 17.00 Uhr

Figure 3: the partner login page on the GitLab repository. Note that you must activate the standard login if you are not employee at Fraunhofer. The page can be reached by either link <https://gitlab.cc-asp.fraunhofer.de/intersect>.

## On Gitlab Terminology

- **Repository:** A repository is what you use to store your codebase in GitLab and change it with version control. A repository is part of a project, which has a lot of other features.
- **Projects:** are used for hosting your codebase, use it as an issue tracker, collaborate on code, and continuously build, test, and deploy your app with built-in GitLab CI/CD.
- **Gitlab Group:** an EU Project with many projects and hence many repositories, essentially it is a collection of repositories i.e., a group of repositories and related tools, like issue tracker, collaboration, etc.

## Creating a new repository

A new repository is a software project and can be created thus as a part of a new Project in GitLab's terminology. Simply click on New project (see green arrow in Figure ) and follow the instructions. By default, all members added to a GitLab Group have access to all projects (i.e., INTERSECT repositories). This can be managed by the specific project setting: click on the repository (project) (see Figure for example) then on the side bar, click on members.

## Further information on GitLab

This deliverable document is not intended to give a thorough deep nor complete tutorial of GitLab (nor Git), however the most elementary options are mentioned here to get you started. An excellent official resource is available online for free at: <https://about.gitlab.com/handbook/>

Partners are encouraged to check it out. There are also numerous resources in the internet including tutorials on you tube etc.

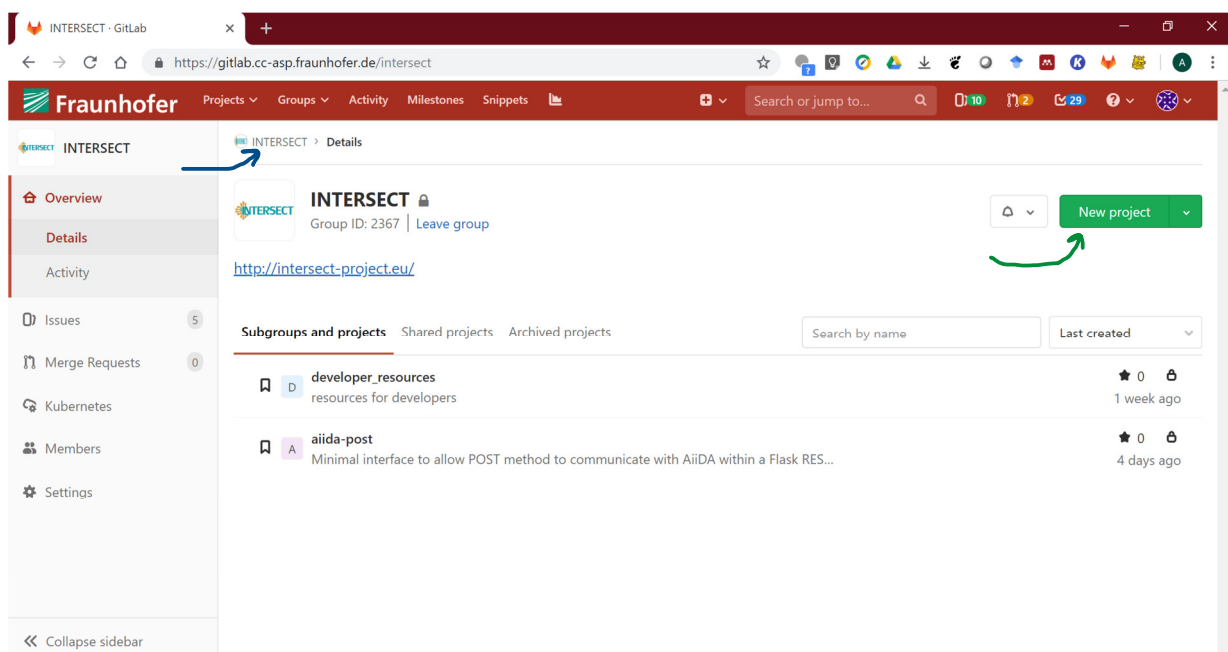


Figure 4: view after logging in. If you are not in the INTERSECT space, click on groups in the top bar (shown by a red arrow) and pick INTERSECT. If not there, please contact Task Leader.

Additionally, as this is a Git based repository, i.e., it is not centralized in the sense that each partner has always a full copy of the repository once they check it out or clone it (see below for explanation), the partners can maintain their own repositories for further exploitation in tandem with decisions in WP1 and WP2.

## General Collaborative Workflow

Recommended additional reads Ref. [4].

In general, the way we work with git is as follows:

1. You need a copy of a git repository on your file system:
  - a. Clone an existing repository from either GitLab/GitHub or another directory
    - Type the following in your command line  
> git clone <address of repo>
  - b. Or, create a new repository from scratch,
    - Type the following in your command line  
> Git init (to create a new repository in a folder in your system)
  - c. Or use the Gitlab web interface to create it first and then clone it as in 1.
2. Develop your code on your local copy– by default you will be working on the so-called Master branch, this is the main default branch
3. When you want to test various options, or work on one or more new features it makes sense to make branches, which are essentially copies of the master (or other branch) at the point of branching.
  - a. You can have as many branches as needed (see naming convention of branches below in the manifest).
  - b. You can switch between branches, or in other words activate a branch, by checking it out from your local working repository.
    - i. Type the following in your command line  
git checkout -b <New branch name> to create a new bench and Change to it (checking it out)
    - ii. Type the following in your command line  
git checkout <existing branch name> to switch (checkout) to an existing branch. NOTE: you are still working locally, on your own repository, you are just creating local branches!
4. In each branch you check out, you can now do your code development. If you are happy, you can merge back to your copy of the master (again we are still working locally, while other developers have also copy of master).
  - a. You first need to stage your changes for committing.

Git keeps track of the changes you make to source code file in each branch. You may not be interested though in merging all those changes to your local master, e.g., no backup files, or temporary ones. **Moreover, you may have a number of large binary files generated through the compilation or testing process**, or even worse, some files with passwords, or other sensitive information, so here git provides and actually demands that the user specifies

explicitly which files should be actually included in the commit, i.e., become part of the actual development (again, all are local on your hard disk or storage). This is called staging in the Git Jargon.

b. Staging is done by the “git add command”:

i. Use

```
> git add <file1> <file2>
```

for committing onto your current branch!

NOTE: never add files automatically, like `git add -A`, as you would may push unwanted changes!

5. Now you can make a commit, meaning you are registering the changes in this branch to its own history. This is done by the commit command.

a. Use

```
> git commit -c “this is a commit with a short comment”
```

b. The comment should be short, and contain information on the issue (see manifest below)

6. The next step is to merge back or push your changes in the current branch to the master branch, again the local one!

a. However, since we are doing a collaborative development (even if you are the only developer! still others need to review your code) you never merge with your local master! Instead, you push your developer branch upstream, i.e., to the GitLab project repository by the following command:

i. `git push`

- this command will push the local branch to the upstream repository on GitLab. If you get something like:

- ```
> ~/developer_resources$ git push
```

  
fatal: The current branch review has no upstream branch.  
To push the current branch and set the remote as upstream, use

```
git push --set-upstream origin review
```

- Then simply do as told! This is needed since usually when you first created the branch it was done locally, and this command published your new branch to the common repository on GitLab.

- b. Once the branch is pushed, you need to create a merge request and ask a fellow developer or collaborated to review the code. GitLab provides online tools to see the changes and compare versions, as well as auto tests.
- c. Once the branch is reviewed it can be merged to master.
- d. It is best to check the status of the repository each time to see whether your local copy is in sync, behind or after the master by doing:
  - i. Git status

### The INTERSECT Developers Manifest

The developer manifesto is also available online on the repository and ownCloud for all partners and will be continuously updated to adapt to possible changing needs of the project.

The manifesto is in the repository “Developer Resources” accessed at:

[https://gitlab.cc-asp.fraunhofer.de/intersect/developer\\_resources](https://gitlab.cc-asp.fraunhofer.de/intersect/developer_resources)

It includes definition of INTERSECT’s naming conventions (for classes, variables, methods, etc.), semantic versioning, review process workflow and guidelines, etc., to ensure coherent and efficient collaboration.

Continuous maintenance and updates are needed for the entire project duration. The entire repository will be managed, especially the open (publicly accessible) parts of the code will keep being maintained as open source project.

Feedback from all users and partners is welcome through the same procedure for code reviews outlined in this document itself. In other words, any suggestions for change or feedback should start by submitting an issue on Gitlab as explained below.

Before we start with the developer manifest itself, we introduce git and GitLab in general focusing on getting you started and most importantly, mentioning where you can access and get more information. As there is an abundance of very good and free documentation on the net, it does not make sense to get deeper in this focused deliverable.

The following are the best practices [1] proposed for all development in INTERSECT, all developers are strongly advised to adhere to these guidelines in the manifest:

1. Each feature, bug fix, no matter how small, must be associated with a Git Issue opened on Gitlab. See below how to open one.
  - a. Choose a short descriptive naming of the issue
  - b. Add a short description of what is needed and why

- c. It is recommended to use referencing using @username to invite certain developers to discuss the issue before or while you are implementing.
2. Create a branch to address the issue,
  - a. Branch naming convention: it is a good practice to add the type of issue (fix, feature) and the main issues numbers to the name of the branch, like so: *45-fix-something would be a branch to fix issue #45*. NOTE: If branches are created from issues, the GitLab will suggest a name for it, it is a good practice to use it. (e.g., 39-login-not-working).
  - b. Though often a branch may address more than one issue, in this case choose a descriptive name, like so: fix-crash-GUI
  - c. Create unit tests for your code  
(see <https://gitlab.cc-asp.fraunhofer.de/help/ci/README.md>)
3. Work on the branch and discuss the implementation with fellow developers/designed on the issue page and/or using Mattermost
4. Once you change the code, push it to the GitLab (this is the “ORIGIN” repository that all developers sync and develop against.)
5. Commit as often as you can. Open new issues as needed and link them to this branch in the description of title. This allows you to revert changes and do tests incrementally.
6. Once done, and you are happy with the code, initialize a merge request on GitLab and specifically ask other fellow developers to review the code.
7. Let the WP leader review the code as well
8. Once all tests are passed, consulting with the developer and reviewer the WP leader can merge the branch (or give the ok to the reviewer/developer to do so on the review page on Gitlab)

### Additional information

Please also check this: <https://about.gitlab.com/2016/10/25/gitlab-workflow-an-overview/#stages-of-software-development> for a more detailed explanation

As well as the GitLab user documentation: <https://gitlab.cc-asp.fraunhofer.de/help/user/index.md>

### Creating Issues

How to create/open an issue:

Go to the Gitlab page of the repository and click on ‘Issues’ on the left sidebar.

Click on 'New Issue' and fill in the details, including 'Assignee' (if known) and 'Labels':

Click on 'Submit issue'.

### Branching and association with issues

Once an issue has been opened, it can be directly associated to a new branch. It is a good practice to associate each branch to an issue, since it allows the team members to see if someone is already working on the issue, and what are the changes that have been made concerning it.

### How to create an issue-associated branch

To associate a branch to an issue, we go to the issue's page on git lab, and choose 'Create branch'. We can change the name of the branch, and the source branch we are branching from if need be.

### How to work locally on the newly created branch

To work locally on the changes, we run the following commands:

- `git pull`
- `git checkout --track origin/1-title-of-readme-file-is-too-long`

The 'git pull' command informs the local repository about the new branch.

In the second command, we use the --track flag that is shorthand for `git checkout -b [branch] [remotename]/[branch]`. It creates a local branch that is associated with the remote one, and checks out (i.e., switches into) into that branch.

### Committing & Pushing

It's important to commit often, not only for having more checkpoints to return to in case of a fallout of a certain approach, but also for the project's periodical reports in which we can show the amount of work that has already been invested in the project.

Here is such a chart that was taken from the OSP-core project:

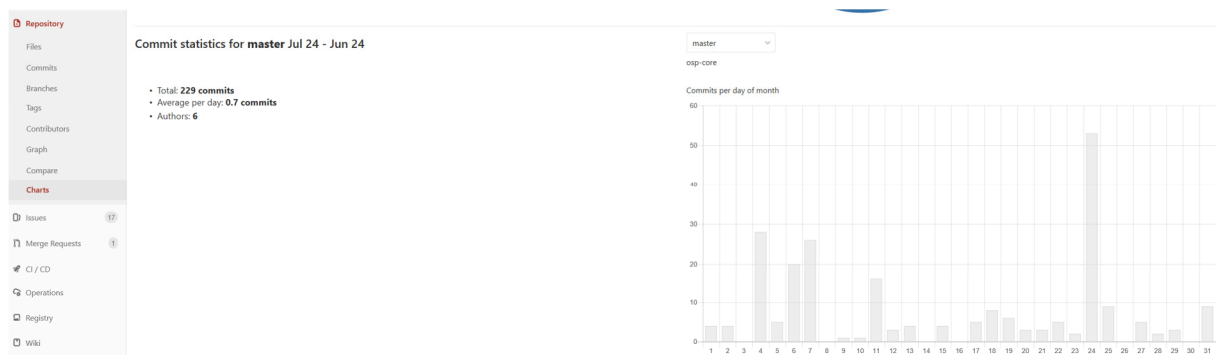


Figure 5: Snapshot of Commitment Chart

Commit often, this documents your efforts for project reporting as well as IP, and in a very good practice, it offers a backup and track of your work. Commit at least once a day at the end of the work-day.

### How to commit and push changes

After making the changes in our code, we can commit our changes and push them to the remote repository by running the following commands:

```
git commit -a -m "shortening title"
```

```
git push -u origin 1-title-of-readme-file-is-too-long
```

The flag -a in the first command includes our changes in the current commit.

Note that it does not include unstaged files. For those, use the 'git add' command.

The flag -u links local branch with the remote branch automatically. In this case it is redundant since the local branch is already associated with the remote one due to the 'git checkout' command we ran before.

### Creating Merge Requests

Once the work on the issue have been completed, we can create a merge request to the master branch.

### How to create a merge request

On the GitLab page, go to 'Merge Requests' on the left sidebar, and click on 'Create merge request'. Fill in the details as you see fit, and, importantly, assign a reviewer that will perform the merging of the request to the master branch.

### Markdown language

See above and <https://gitlab.cc-asp.fraunhofer.de/help/user/markdown>

### Team Communication: Mattermost

The accounts for GitLab are the same for the Mattermost service. It can be reached at:

<https://mattermost.cc-asp.fraunhofer.de/intersect>

To log in, you see a page directing you to the GitLab login page (see Figure 5) lick on GitLab (red arrow) and then you will get the GitLab Login page, from then click on standard (another red arrow) and then log in!



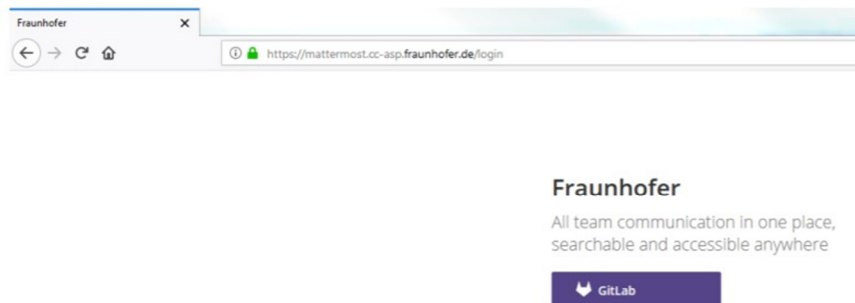


Abbildung 4: Login Fraunhofer Mattermost

Danach folgt eine Weiterleitung zum Fraunhofer GitLab. Dort unter Verwendung der »Fraunhofer-UserID« und des »Fraunhofer Primär Passworts« anmelden.

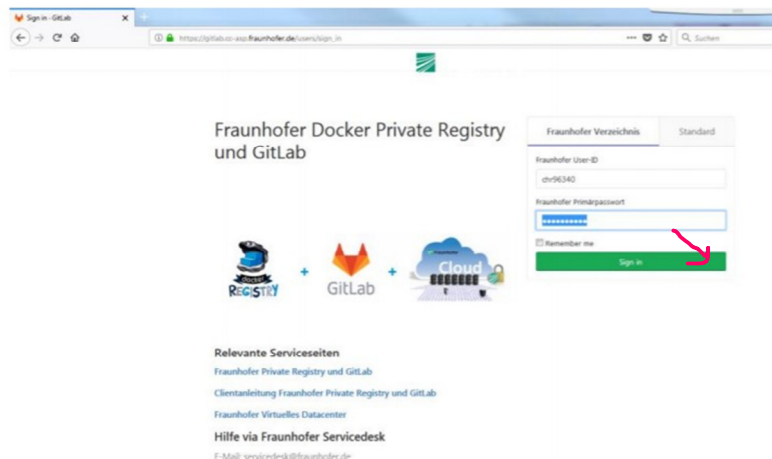


Figure 6: Login page of GitLab

**About Mattermost:** An alternative to proprietary SaaS messaging, Mattermost brings all your team communication into one place, making it searchable and accessible anywhere. It's written in Golang and React and runs as a production-ready Linux binary under an MIT license with either MySQL or Postgres. (cut&paste from mattermost.com).

### The Mattermost APP for mobile and Desktop

While Mattermost can be accessed easily on any web browser with full functionality, you can also download and install Apps for desktop or your mobile devices.

Visit: <https://mattermost.com/download-v2/#mattermostApps>

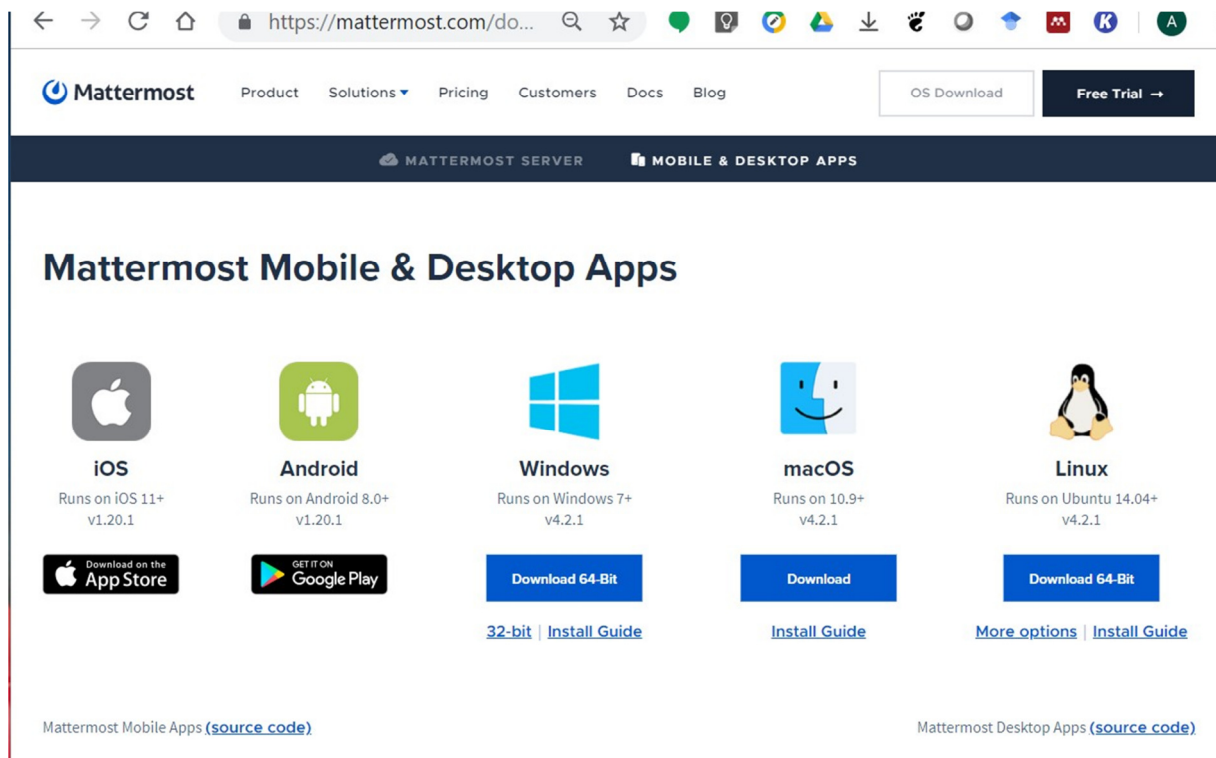


Figure 7: Mattermost client software is supported on all platform including mobile apps.

### Mattermost functionality in a nutshell

An excellent user guide (short, with many pictures) is found here:

<https://docs.mattermost.com/help/getting-started/welcome-to-mattermost.html#the-basics>

The main concepts from the above link are brought here for convenience (cut&paste):

#### Teams

A team is a digital workspace where you and your teammates can collaborate in Mattermost. Depending on how Mattermost is set up in your organization, you can belong to one team or multiple teams.

In INTERSECT Mattermost we are all in the INTERSECT team, but can be also in others (e.g., EMMO team).

#### Channels

Channels are used to organize conversations across different topics. They are located on the left-hand panel of Mattermost.

There are three types of channels: Public Channels, Private Channels, and Direct Messages.

#### Public Channels:

Public Channels are open to everyone on a team. New team members are automatically added to two Public Channels when they sign up: Town Square and Off-Topic.

Public Channels are identified with a globe icon.

Click on the More... button at the bottom of the Public Channels section to explore more channels to join!

### Private Channels:

Private Channels are for sensitive topics and are only visible to selected team members. Any member of a Private Channel can add additional members. Channel members can choose to leave at any time, but only the channel owner or Team Admin can remove other members.

Private Channels have a lock icon.

### Direct Messages (DM) and Group Messages (GM)

Direct Messages are for conversations between two people. Group Messages are Direct Messages that have conversations among three or more people. Both are visible only to the people involved.

### Messages

Sending messages and replying to messages are important ways to keep conversations active with your team. You can edit and delete messages after you have sent them. You can also share links to any message in Mattermost.

The team settings are on the top left of the side bar, there you can change settings like notifications (via email or desktop/mobile etc).

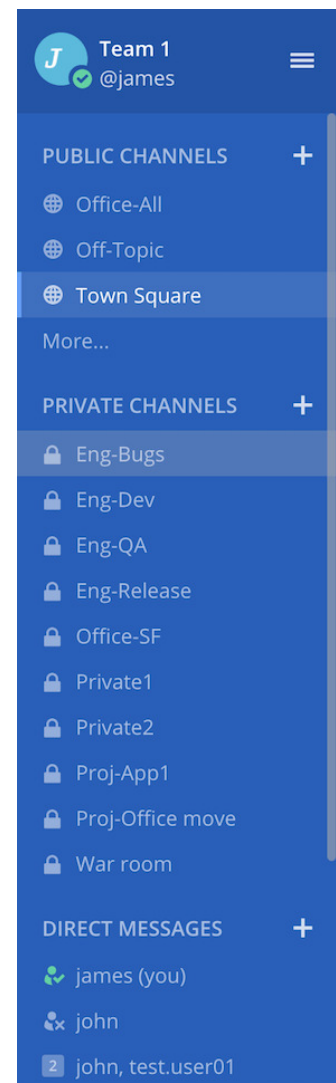


Figure 8: Side bar showing various channels (figure reproduced from:  
<https://docs.mattermost.com/help/getting-started/welcome-to-mattermost.html#the-basics> )

### Specifically mentioning and addressing members in messages

Use the “@Mentions” options, namely, use “@” in the message to bring a list of users, start typing to filter then pick the user. They will get a notification (on desktop or email) to alert them.

Example:

@arrigo\_calzolari: Hi Adham, I've got new ideas for MODA1

Use **@channel** and **@all** to address all in a channel. All members in the channel receive a notification.

You can control notifications in the **Channel Menu > Notification Preferences > Ignore mentions for @channel, @here and @all**.

**@here** allows you to mention all that are online in a channel! (they get notifications)

#### Recent mention

Click **@ next** to the search box to query for your most recent **@mentions** and words that trigger mentions. Click jump to go to them.

### Formatting

Please see thorough and short description here:

<https://docs.mattermost.com/help/messaging/formatting-text.html>

e.g., to make a headline use: #, ##, etc. like so:

# this is headline 1

## this is subsection or headline 2 etc..

to make a list use:

1. Item 1
1. Item 2
1. Item three

(yes, all are 1. And can be any number, does not need to be sequential, since Mattermost will then manage and order the numbering once you type enter and send the message!)

Mark-up enables also to use latex for equations, code syntax highlighting, and much more, see link above.

Note: same mark-up is also used in Gitlab!

### References

- 
- [1] E. Wimmer, et al., "EMMC - White Paper for standards of modelling software development", <https://emmc.info/emmc-csa-white-paper-for-standards-of-modelling-software-development>
- [2] [https://en.wikipedia.org/wiki/GitHub#cite\\_note-techcrunch-4](https://en.wikipedia.org/wiki/GitHub#cite_note-techcrunch-4)
- [3] [https://en.wikipedia.org/wiki/GitHub#cite\\_note-hugeinvestment-5](https://en.wikipedia.org/wiki/GitHub#cite_note-hugeinvestment-5)
- [4] <https://about.gitlab.com/2016/10/25/gitlab-workflow-an-overview/#gitlab-workflow>