

Introduction to Gitlab

Introduction to Gitlab

What is GitLab

GitLab is a complete DevOps platform, delivered as a single application. This makes GitLab unique and creates a streamlined software workflow, unlocking your organization from the constraints of a pieced together toolchain. Learn how GitLab offers unmatched visibility and higher levels of efficiency in a single application across the DevOps lifecycle.

GitLab started as an open source project to help teams collaborate on software development. GitLab's mission is to provide a place where everyone can contribute. Each team member uses our product internally and directly impacts the company roadmap. This exceptional approach works because we're a team of passionate people who want to see each other, the company, and the broader GitLab community succeed and we have the platform to make that possible.

From [about gitlab](#).

Target audience

The course material presented here is meant for a technically inclined person with a basic proficiency in the Git source management system and an inclination to learn about combining the power of Git with the versatility of GitLab.

Scope & Terminology

GitLab is tightly wedded, as the name suggests, to the git source code management system, its concepts, command and terminology. It helps to be familiar with Git's interpretations of **working directory**, **staging**, **repository**, **branch**, **tag**, **commit** and **revision** among others.

For some of the exercises git command are used on the command line to update the content of the GitLab repositories. But for the most time the tutorial evolves GitLab's web interface.

This document will not cover more advanced topics such as CI/CD scripts and advanced GitOps topics.

Table 1. Terminology Summary

Application	Term	Description
Git	working directory	Location of the local copy of the git repository
Git	staging	Cached files ready for commit.
Git	repository	General term for a git project.

Git	branch	An working copy of the repository.
Git	tag	A named reference to a revision.
Git	commit	Annotated addition to the repository.
Git	revision	ditto
GitLab	merge request	Request to include changes.

GitLab setup

GitLab setup prerequisites

For the course the following prerequisites have to be met:

- Recent version of **docker.io**

All docker commands are invoked without **sudo**. It is therefore assumed that the participants user is part of the **docker** Unix group.

Gitlab docker setup

The examples in this tutorial are based on gitlab-ce (community edition). The docker image and instruction on how to install it can be found on the [gitlab docker hub page](#).

Table 2. Port mapping

Protocol	Local Port	Docker Port
HTTPS	8443	443
HTTP	8480	80
SSH	8422	22

Container start script with prefilled port numbers.

```
GITLAB_HOSTNAME=gitlab.local
GITLAB_HOME=${HOME}/var/work/gitlab
OMNIBUS=""
OMNIBUS+="external_url \"http://${GITLAB_HOSTNAME}:8480/\";"
OMNIBUS+="gitlab_rails['gitlab_shell_ssh_port'] = 8422;"

function start() {
  [[ -d ${GITLAB_HOME} ]] && mkdir -p ${GITLAB_HOME}
  docker run \
    --detach \
    --hostname ${GITLAB_HOSTNAME} \
    --env GITLAB_OMNIBUS_CONFIG="${OMNIBUS}" \
    --publish 8443:443 \
    --publish 8480:8480 \
    --publish 8422:22 \
    --name gitlab \
    --restart always \
    --volume ${GITLAB_HOME}/config:/etc/gitlab \
    --volume ${GITLAB_HOME}/logs:/var/log/gitlab \
    --volume ${GITLAB_HOME}/data:/var/opt/gitlab \
    gitlab/gitlab-ce:latest
}

start
```



The initial startup of the container is taking quite some time. Patience until the first connection is required.

For convenience add **gitlab.local** to the **/etc/hosts** file

*Add **gitlab.local** to the **/etc/hosts** file.*

```
sudo sed -i \
  -e '/gitlab.local/d' \
  -e '$a\127.0.0.1 gitlab.local' \
  -e '$a\:::1 gitlab.local' \
  /etc/hosts
```

Gitlab account setup

Configure admin account password

After starting the docker container open a browser and navigate to <http://gitlab.local:8480>.



Please create a password for your new account.



GitLab

A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

The screenshot shows a form titled "Change your password". It contains two input fields: "New password" (marked with a purple circle 1) and "Confirm new password" (marked with a purple circle 2). Below the fields is a blue button labeled "Change your password" (marked with a purple circle 3).

Didn't receive a confirmation email? [Request a new one](#)

Already have login and password? [Sign in](#)

Figure 1. Admin password setup

- 1 Provide an admin password.
- 2 Confirm the admin password.
- 3 Finish process by clicking [Change your password]

Create user account

Once the admin password is in place a normal user account may be created.

Throughout the course the following values are assumed:

Property	Value
First name	Joe
Last name	Developer
User name	jdev
Email	joe.developer@gitlab.local
Password	topsecret
Role	Software Developer



🔔 Your password has been changed successfully. ✕

GitLab

A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

The screenshot shows the GitLab registration interface. At the top, there are two tabs: 'Sign in' and 'Register'. The 'Register' tab is selected and has a purple circle with the number '1' next to it. Below the tabs, there are several input fields: 'First name' with the value 'Joe' (callout 2), 'Last name' with the value 'Developer' (callout 3), 'Username' with the value 'jdev' (callout 4), and 'Email' with the value 'joe.developer@gitlab.local' (callout 5). Below the 'Username' field, there is a green message that says 'Username is available.' Below the 'Email' field, there is a 'Password' field with the value '.....' (callout 6) and a note that says 'Minimum length is 8 characters'. At the bottom of the form, there is a green 'Register' button (callout 7).

Figure 2. User account creation

- 1 Click on tab [Register]
- 2 Joe for **First name**
- 3 Developer for **Last name**
- 4 jdev for **User name**
- 5 joe.developer@gitlab.local for **Email**
- 6 topsecret for **Password**
- 7 Create account by clicking [Register]

Welcome to GitLab Joe!

In order to personalize your experience with GitLab
we would like to know a bit more about you.

Role

Software Developer 1

This will help us personalize your onboarding experience.

Get started! 2

Figure 3. Define role

- 1 Select software developer for Role
- 2 Define role by clicking [Get started!]

Congratulations the initial setup is complete!

Module 1 - User configuration

Before diving into working with GitLab the first thing to do is go through the user settings and configure a few things that were missing from the initial setup such as adding privacy settings and **ssh public keys**.

Goals

- Navigate the user setting
- Set privacy settings.
- Manage ssh public keys.

User profile configuration

For the purpose of the excercises conducted during this tutorial no particular changes in the user profile need to be made but for better recognition the avatar, the privacy setting and the ssh keys are being discussed here.

Navigate to the user' settings

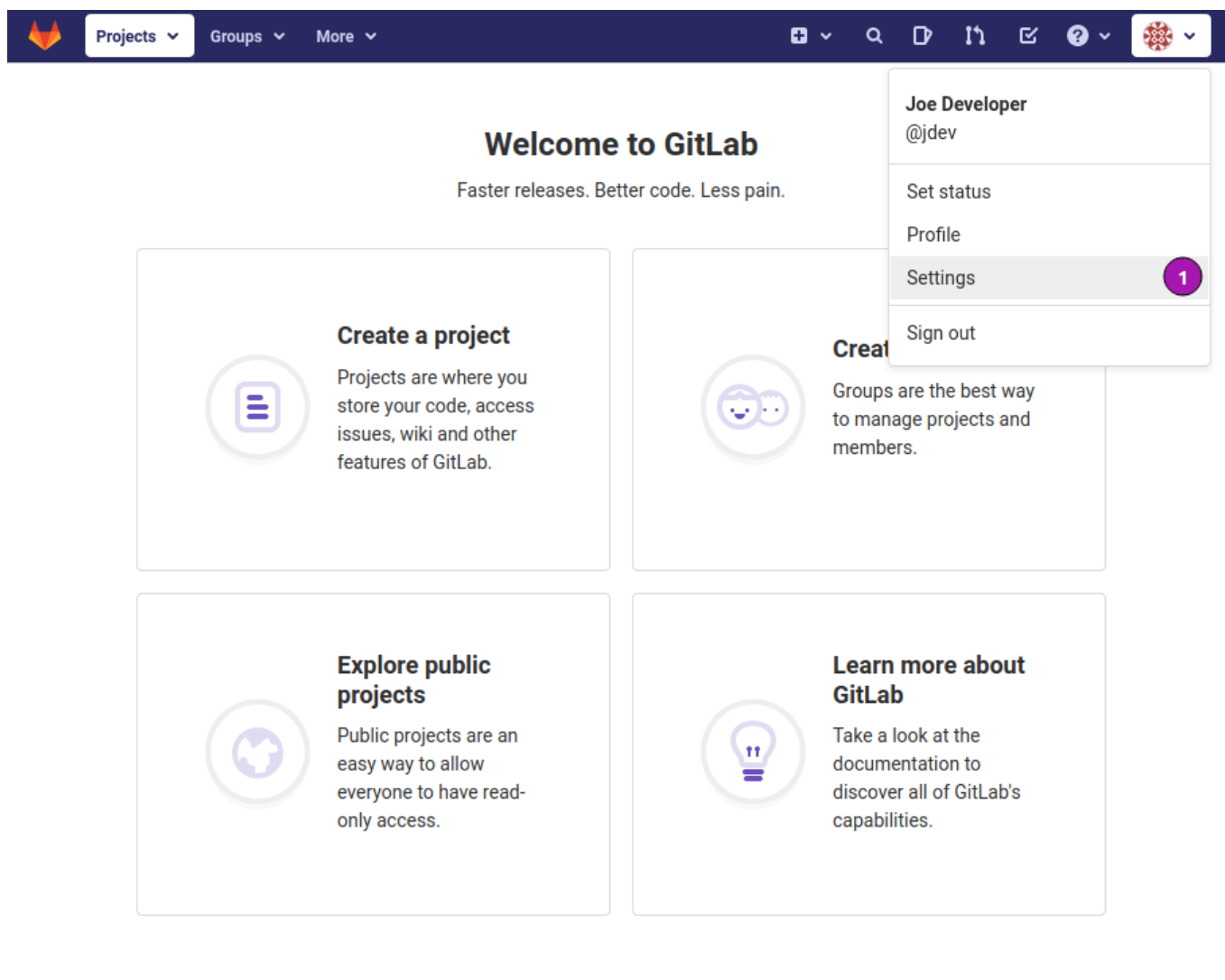


Figure 4. Navigate to the user settings

1 Click on the avatar on the right hand side and select **Settings**.

Change the avatar

The screenshot displays the GitLab 'User Settings' interface. On the left, a sidebar lists various settings categories, with 'Profile' highlighted and marked with a circled '1'. The main content area is titled 'User Settings > Edit Profile'. Under the 'Public Avatar' section, there is a circular profile picture and a 'Choose file...' button, with a 'No file chosen' message and a circled '2' next to it. Below this, the 'Current status' section features a text input field with an emoji icon and the placeholder text 'What's your status?'. The 'Main settings' section contains several form fields: 'Full name' (Joe Developer), 'User ID' (2), 'Email' (joe.developer@gitlab.local), 'Public email' (Do not show on profile), 'Commit email' (joe.developer@gitlab.local), 'Skype' (username), and 'Linkedin'. The interface is clean and modern, with a dark blue header and a light gray sidebar.

Figure 5. Change the avatar

- 1 On the left hand side navigation provides access to the various topics.
- 2 To change the avatar click on [Choose file].

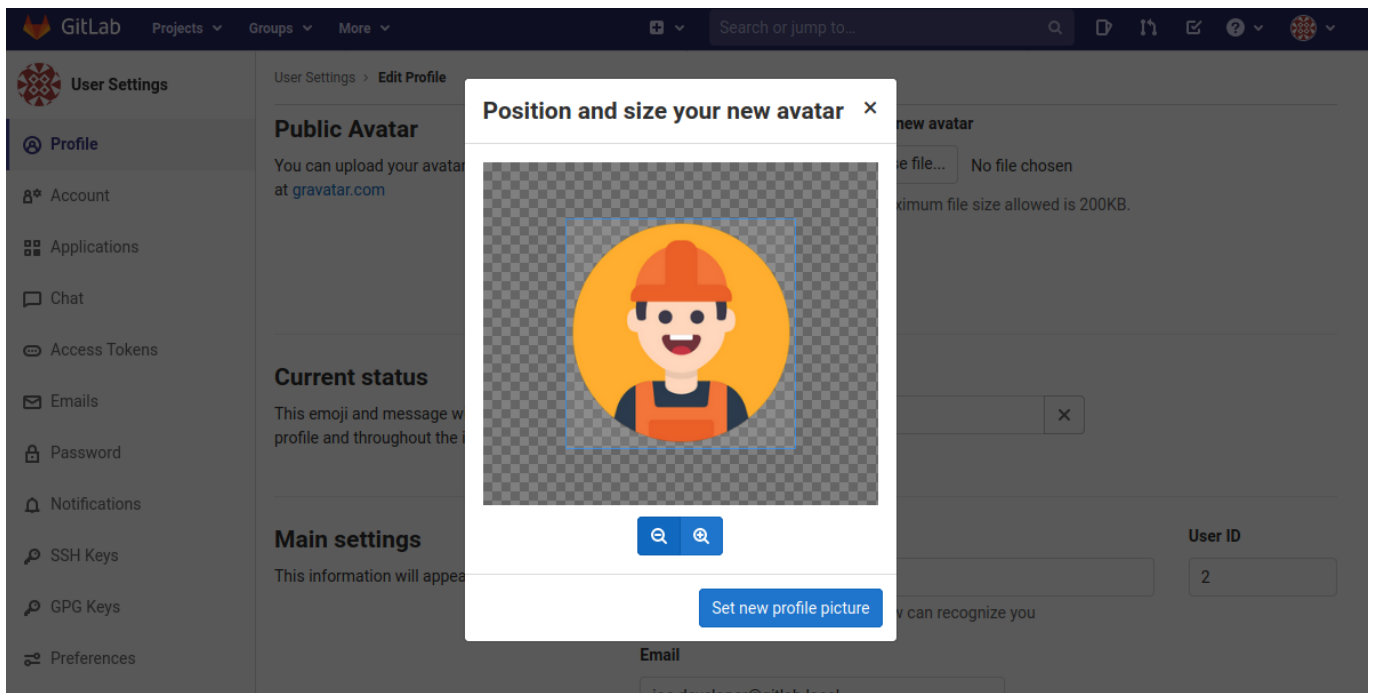


Figure 6. Upload new avatar from image file

For Joe Developer the chosen avatar is a construction worker with hard hat.

Review privacy settings

Under menu item **Profile** scroll down to the bottom of the page.

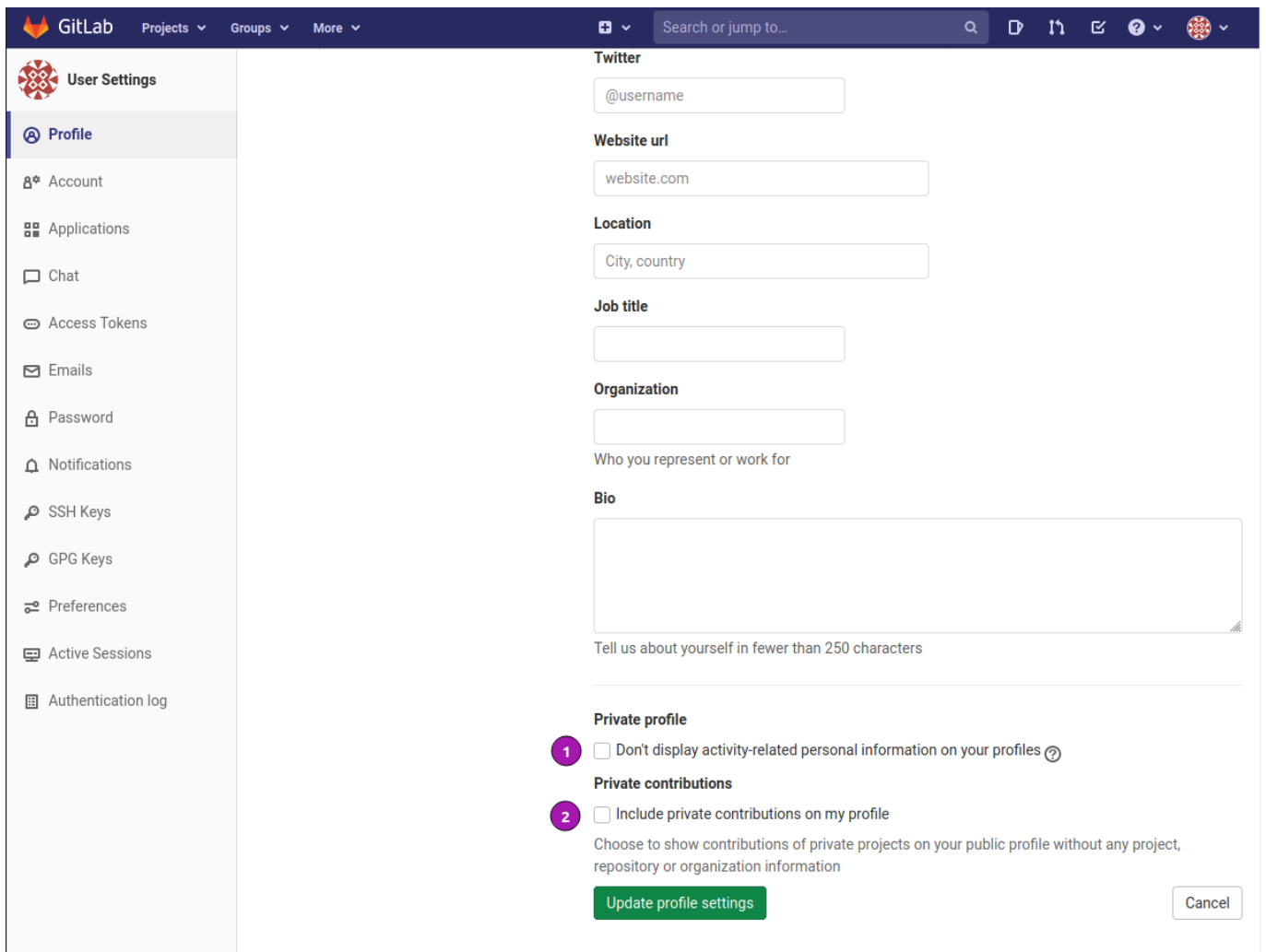


Figure 7. Privacy settings

- 1 If so desired a user can hide certain information on the profile page from being viewed by other participants. Click on the question mark ? to get a list of the information to hide. This may be necessary on a publicly accessible instance of GitLab.
- 2 This setting is only relevant for publicly accessible repositories. If your organization maintains private repositories should changes made to said repositories be shown anonymized.

Upload ssh public keys

Repositories on GitLab can be accessed via both the HTTPS protocol and the SSH protocol. To prevent constant password prompts for each remote git operation and for better efficiency one can upload a ssh public key to GitLab.

If ssh keys have already been generated previously skip the next step.

Generate a ssh public key on the command line

```
$ ssh-keygen -t ed25519 -f ~/.ssh/gitlab ❶
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase): ❷
Enter same passphrase again: ❸
Your identification has been saved in /home/uroesch/.ssh/gitlab
Your public key has been saved in /home/uroesch/.ssh/gitlab.pub
The key fingerprint is:
SHA256:hXWpMuLBU8fnrEYPNB0sDKwXaITn5Jn/Vtvdin6gJI4 uroesch@uroesch-puzzle
The key's randomart image is:
+--[ED25519 256]--+
|  o.o.o. . .      |
| . = o .+ o o    |
| * + .o + * o    |
| * .. + B =     |
|  o S + o o      |
|   ...+o..+     |
|   .+ +oo..     |
|   E + o...o .  |
|   . .o...     |
+-----[SHA256]-----+
$ ls -l ~/.ssh/gitlab*
-rw----- 1 jdev jdev 464 Nov  9 12:09 /home/jdev/.ssh/gitlab ❹
-rw-r--r-- 1 jdev jdev 104 Nov  9 12:09 /home/jdev/.ssh/gitlab.pub ❺
```

- ❶ The **ssh-keygen** command generates a new key option **-t** specifies the format **ed25519** is the easiest to handle. **-f** specifies the file location and name where to store the generated key.
- ❷ The key can be secured with a passphrase which is highly recommended.
- ❸ Confirm passphrase.
- ❹ The generated private key. Note the file permission. Keep this file safe from other users.
- ❺ The public key is the one to be copied to GitLab. Open the file and copy the content.

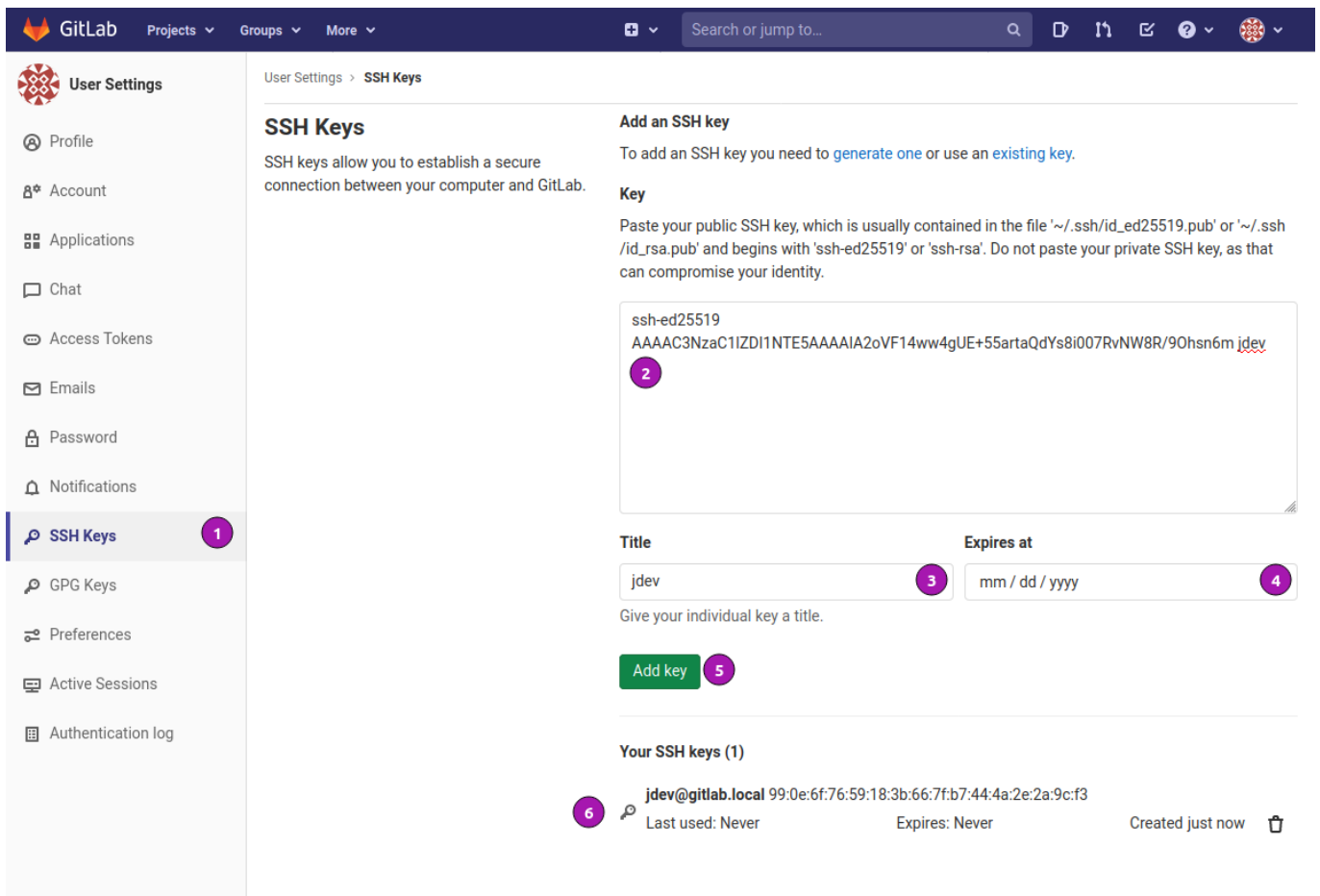


Figure 8. Upload a ssh public key

- 1 Navigate to menu item **SSH Keys**.
- 2 Copy the content of the previously created ssh public key into the text area.
- 3 Change the title of the key with a name that is uniquely identifiable.
- 4 Optionally set an expiry date for the key.
- 5 Click [Add key] to save the key.
- 6 Previously uploaded key. Modifications are not possible one can only delete the entry.

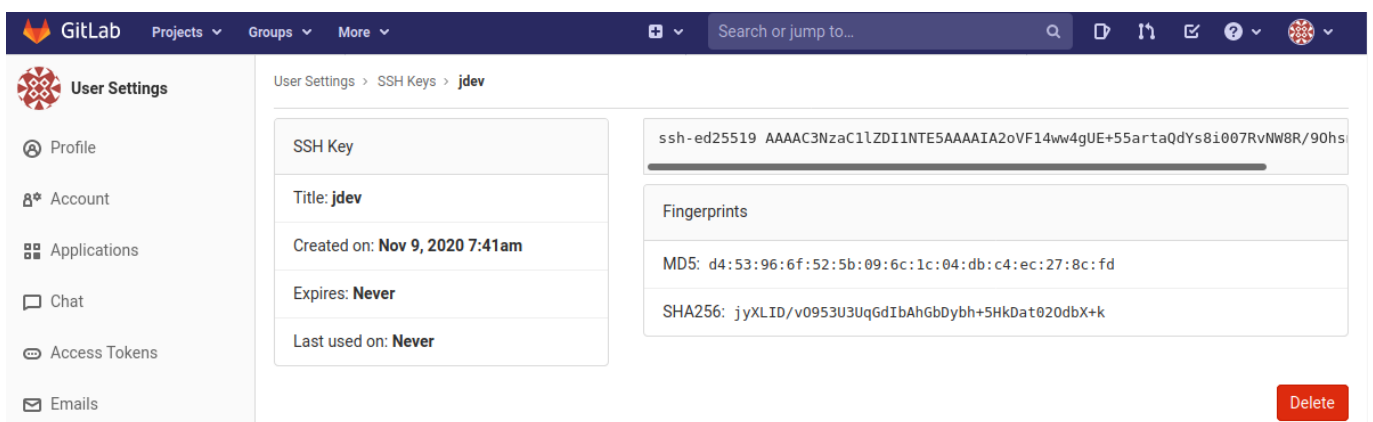


Figure 9. Upload confirmation for ssh public key

Module 2 - Gitlab project

A GitLab project has at its center a git repository and a few services that evolve around said repository. Such as an issue tracker, contributions from other developers via **merge requests** and todo lists among others.

Goals

The creation of a new project is probably not as often used as the following exercises of working with changes and merge request but it is of utmost importance to understand the configuration and the implications of certain settings.

- Create a new GitLab project.
- How to navigate a project.
- Protect certain branches like **master**.

Create a new project

In the following steps a new personal public repository is created. The process of creating a new project is assisted and straight forward.

The creation starts with clicking on the boxed plus sign left to the search field.

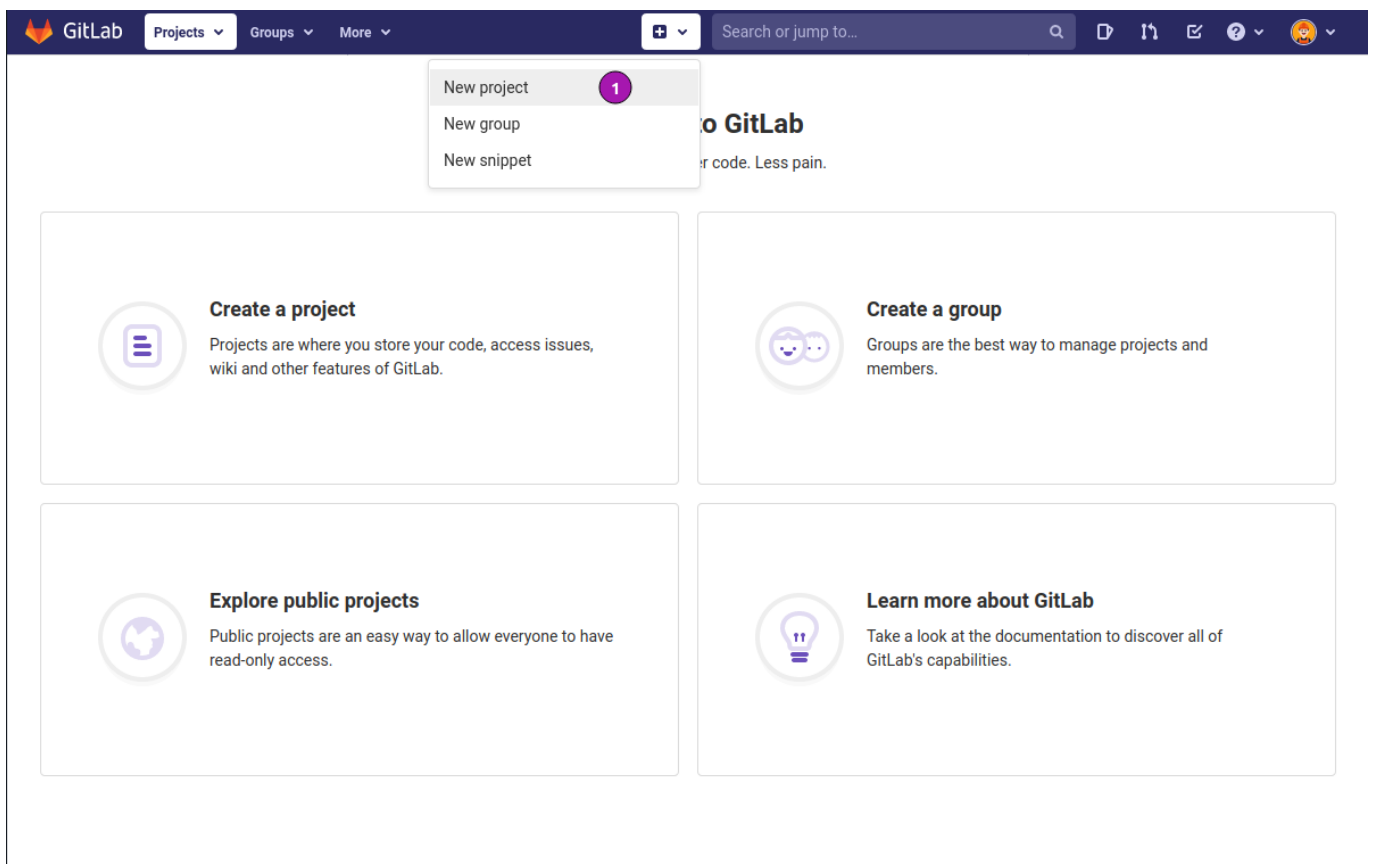


Figure 10. Create a new project from the top navigation bar

- 1 Choose **New Project** to open the creation form.

Figure 11. Fill in the required information

- ① Use gitlab-project for **Project name**
- ② Is the base URL for accessing the repository, can't be modified.
- ③ Project slug is pre-filled based on the project name. May be modified but is usually left as is.
- ④ Provide a project description. This gives visitors an overview of what the project is all about in a few sentences.
- ⑤ Choose **Public** as the visibility layer.
- ⑥ Check **Initialize repository with a README** this creates a bare bone **README.md** file in the root of the repository.
- ⑦ To finish hit the [Create project] button.

Navigate new project

Right after the project is created the next screen is the project overview. For a first time visitor the screen has quite a few elements and is probably confusing at first sight.

Here the various elements are explained shortly to provide an initial overview where to find which functionality.

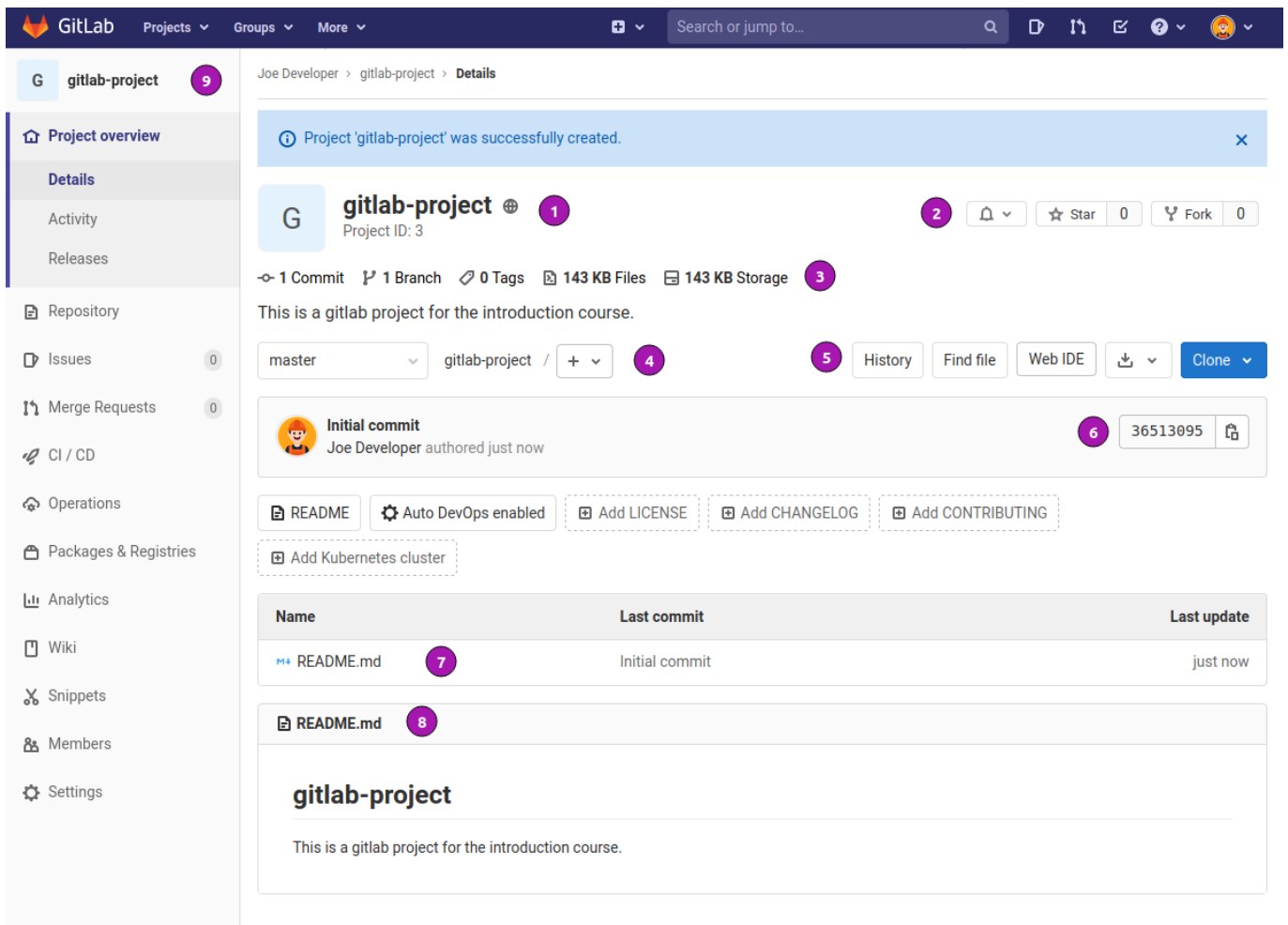


Figure 12. Initial project overview screen.

- ① Project name and visibility level marked by icon on the right hand side of the project name.
- ② Notification, star and fork buttons for other developers interested in the project.
- ③ Git repository statistics with regards to number of commits, branches tags and storage used.
- ④ Branch navigation and file, branch and tag creation.
- ⑤ Miscellaneous functionality such as file modification with Web IDE, archive downloads and clone addresses for the repository.
- ⑥ Current git revision number in shortened SHA1 format.
- ⑦ File browser area, list and navigate files and directories in the project.
- ⑧ HTML rendered content of **README . md**.
- ⑨ Navigation of further functionality and access to project settings.

Clone project

GitLab has quite a few tools allowing for changes to the git repository data via the web interface. But for most purposes the content of the git repository is worked on with an IDE or on the command line externally.

In this exercise the previously created **gitlab-project** is cloned from the command line.

Navigate to the overview page of the project → <http://gitlab.local:8480/jdev/gitlab-project>

The screenshot shows the GitLab interface for a project named 'gitlab-project'. The left sidebar contains navigation options like 'Project overview', 'Details', 'Activity', 'Releases', 'Repository', 'Issues', 'Merge Requests', 'CI / CD', 'Operations', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', 'Members', and 'Settings'. The main content area displays project statistics (1 Commit, 1 Branch, 0 Tags, 143 KB Files, 143 KB Storage) and a commit history table. A 'Clone' dropdown menu is open, showing 'Clone with SSH' and 'Clone with HTTP' options. The SSH URL is highlighted, and a 'Copy URL' button is visible. The commit history table shows a single commit for 'README.md'.

Name	Last commit	Last update
README.md	Initial commit	47 minutes ago

Figure 13. Copy project address for git clone operation.

- 1 Press the [Clone] button.
- 2 Copy the ssh address.

Change to terminal window and execute

```
$ git clone ssh://git@gitlab.local:8422/jdev/gitlab-project.git 1
Cloning into 'gitlab-project'...
X11 forwarding request failed on channel 0
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

$ cd gitlab-project 2
$ git log 3
commit 3651309571108055dcafb3daca9f9678e90c91291 (HEAD -> master, origin/master,
origin/HEAD) 4
Author: Joe Developer <joe.developer@gitlab.local>
Date: Mon Nov 9 12:10:30 2020 +0000

Initial commit
```

- 1 Clone the repository with the copied URL.
- 2 Chang into the freshly created **gitlab-project** directory.

- 3 Inspect the commit history with `git log`.
- 4 Verify the commit hash matches with the one displayed in the project overview.

To simulate different user behaviour in later modules the user information for this cloned repository should be changed to Joe Developer.

```
$ git config --local user.name "Joe Developer" ①
$ git config --local user.email "joe.developer@gitlab.local" ②
$ git config --local --get-regexp 'user' ③
user.name Joe Developer
user.email joe.developer@gitlab.local
```

- 1 Set the **user.name** for the repository to 'Joe Developer'.
- 2 Set the **user.email** for the repository to 'joe.developer@gitlab.local'.
- 3 Confirm the settings.

Repository protection

Among the many settings available for a GitLab project at this stage the one that really matters is the protection of the master branch.

Per default only the maintainer can push and as such also force push changes to the **master** branch. This can be further restricted and no one make changes to master without proper evaluation of the changes.

Protected Branches

Keep stable branches secure and force developers to use merge requests.

By default, protected branches are designed to:

- prevent their creation, if not already created, from everybody except Maintainers
- prevent pushes from everybody except Maintainers
- prevent **anyone** from force pushing to the branch
- prevent **anyone** from deleting the branch

Read more about [protected branches](#) and [project permissions](#).

Protect a branch

Branch:
Wildcards such as `*-stable` or `production/*` are supported

Allowed to merge:

Allowed to push:

Branch	Allowed to merge	Allowed to push	
master <small>default</small>	Maintainers	No one	<input type="button" value="Unprotect"/>

Protected Tags
Limit access to creating and updating tags.

Deploy Tokens
Deploy tokens allow access to packages, your repository, and registry images.

Figure 14. Restrictive master branch settings.

- 1 In the project navigate to **Settings**
- 2 Click on **Repository**.
- 3 Navigate to **Protected Branches** and hit the [Expand] button.
- 4 Go to the box for **master** under allowed to push choose **No one**.



By protecting the master branch from direct pushes one can prevent fat-fingered git pushes where master gets accidentally overwritten with content from another branch. As a disclaimer this never happened to the instructor guiding you currently, ever!!!

Module 3 - Gitlab group

A GitLab group is a way to structure and manage one or more related projects. Groups can also be compared to folders on a file system.

Groups inherit permission to their subgroups and/or projects. Say if someone has access to a group, they also get access to all the projects in the group.

This also includes the issues and merge requests for the projects in the group, and analytics for the group's activity.

Groups allow to communicate with all of the members at once.

Goals

The creation a new group and manage access to it.

- Create a new GitLab group.
- Manage access to the group.
- Basic Gitlab group settings.

Create a new group

In the following steps a new group is created. The process of creating a new group is assisted and straight forward.

The creation starts with clicking on the boxed plus sign left to the search field.

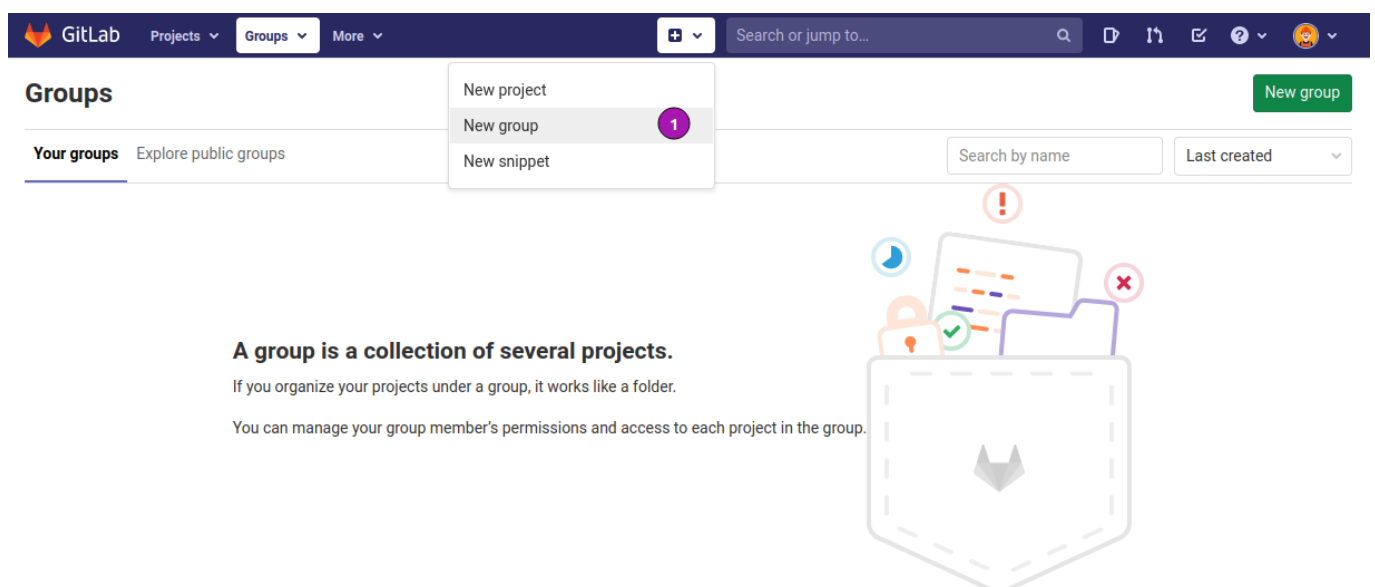


Figure 15. Create a new group from the top navigation bar

1 Choose **New Group** to open the creation form.

The screenshot shows the GitLab 'New group' form. On the left, there is a sidebar with the title 'New group' and explanatory text: 'Groups allow you to manage and collaborate across multiple projects. Members of a group have access to all of its projects. Groups can also be nested by creating subgroups. Projects that belong to a group are prefixed with the group namespace. Existing projects may be moved into a group.' The main form area is titled 'Create group' and has two tabs: 'Create group' (active) and 'Import group'. The form contains the following fields and options:

- Group name:** A text input field containing 'acme', marked with a purple circle '1'.
- Group URL:** A text input field containing 'http://gitlab.local:8480/' and 'acme', marked with a purple circle '2'.
- Group description (optional):** A text area containing 'ACME Ltd. Projects', marked with a purple circle '3'.
- Group avatar:** A file upload button labeled 'Choose file...' with the text 'No file chosen' and a note 'The maximum file size allowed is 200KB.'
- Visibility level:** A section titled 'Who will be able to see this group? [View the documentation](#)' with three radio button options:
 - Private**: The group and its projects can only be viewed by members. (marked with a purple circle '4')
 - Internal**: The group and any internal projects can be viewed by any logged in user.
 - Public**: The group and any public projects can be viewed without any authentication.

At the bottom of the form, there is a green 'Create group' button and a 'Cancel' button.

Figure 16. Fill in the required information

- 1 Use **acme** as **Group name**
- 2 Specify the **Group URL**, usually derived from the **Group name**.
- 3 Add the optional **Group description** which helps to distinguish the group's purpose.
- 4 Specify the **Visibility level** of the group. Defaults to **Private**.

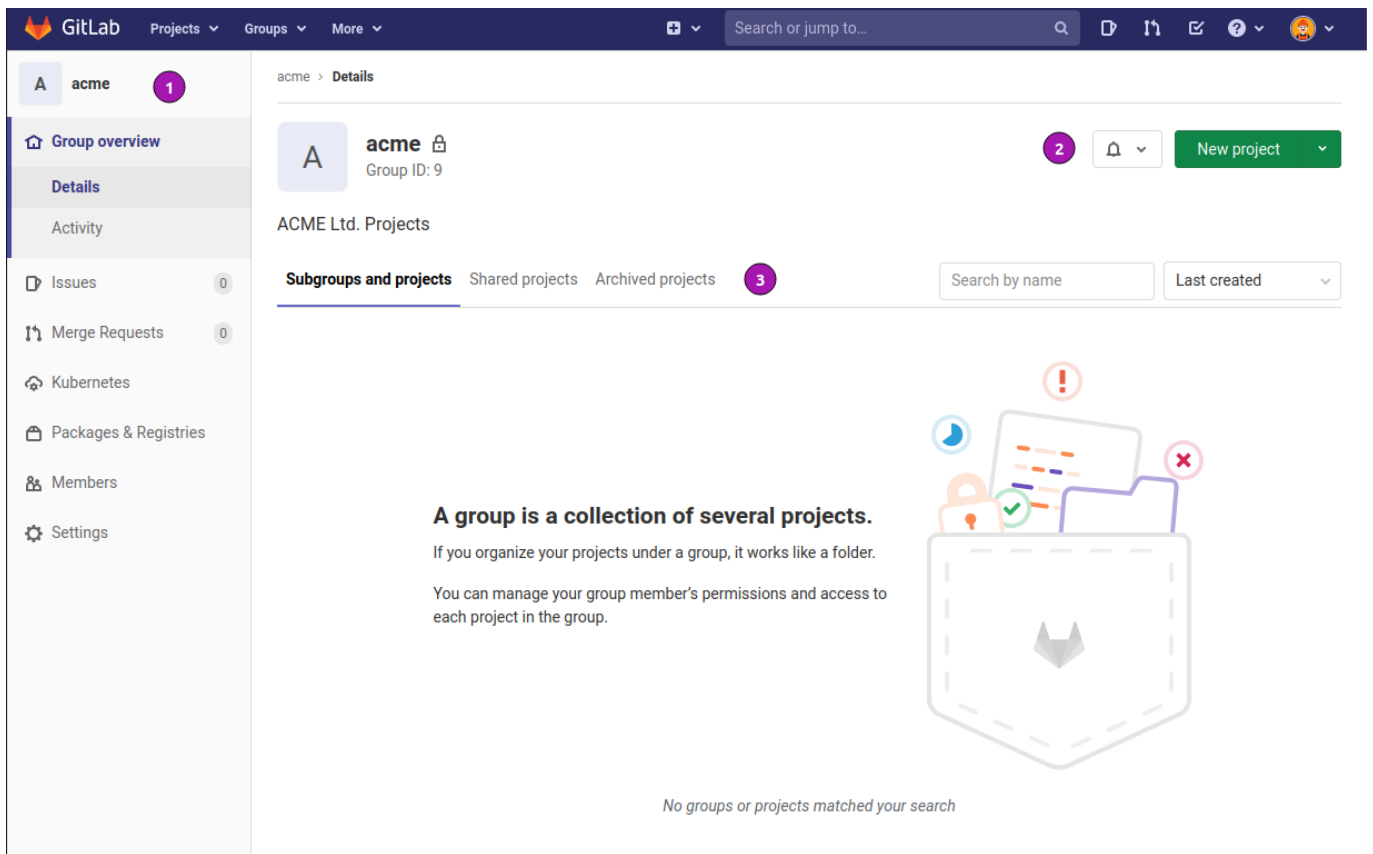


Figure 17. Initial group overview

- 1 Group navigation and configuration.
- 2 Notification and new project or subgroup creation.
- 3 Subgroup and projects tab navigation.

Group members

An integral part of groups is membership management.

The task is very straight forward but has a couple of Gitlab specific settings which need further explanation.

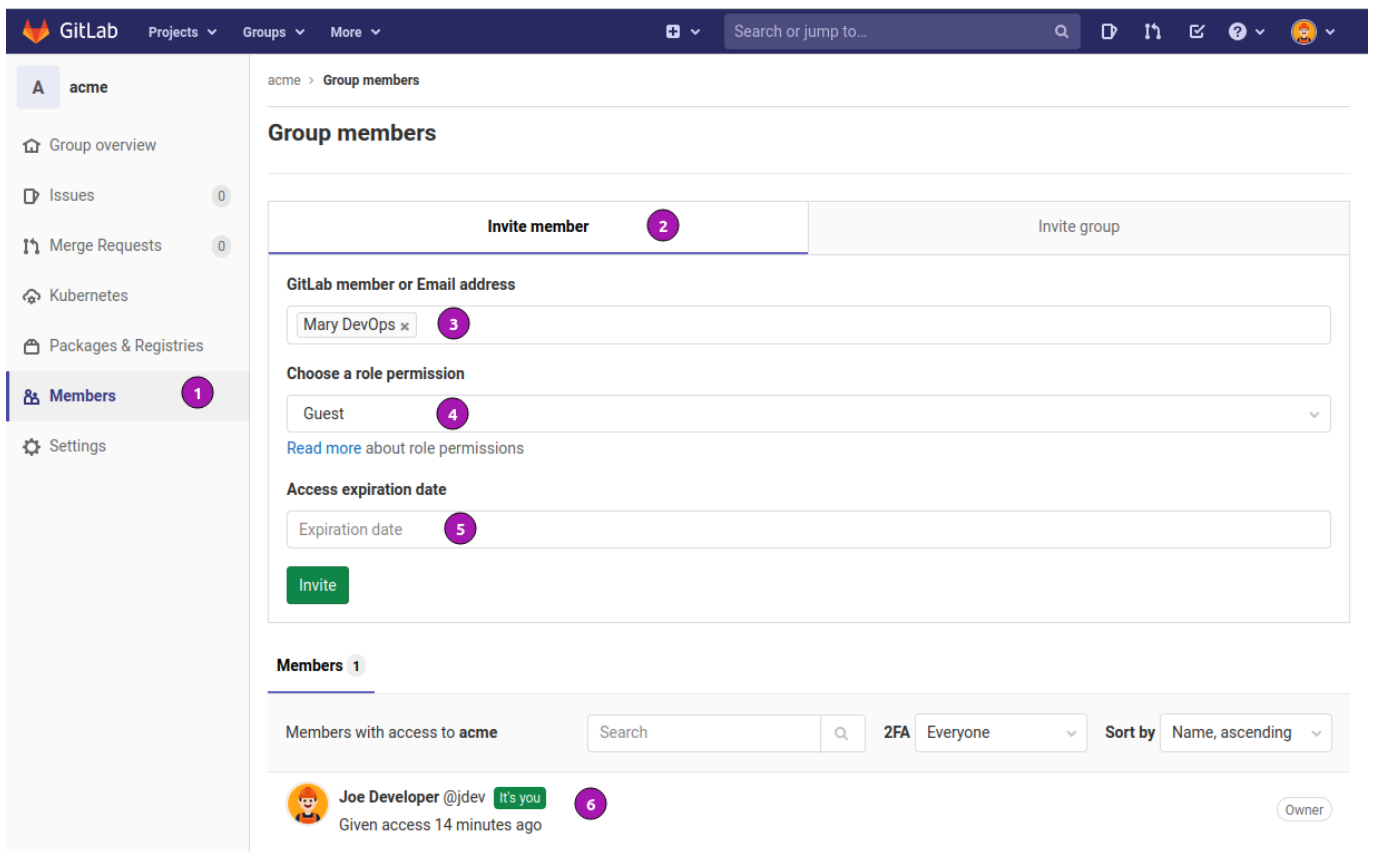


Figure 18. Manage group members

- 1 Members Navigation.
- 2 Member invitation tab.
- 3 Enter Gitlab member or email address of invitee.
- 4 Enter role permission, default is **Guest**.
- 5 For temporary group members add an expiration date of membership.
- 6 List of existing members.

Group settings

Here quite a few settings for a Gitlab group this section focused showing the bare essentials.

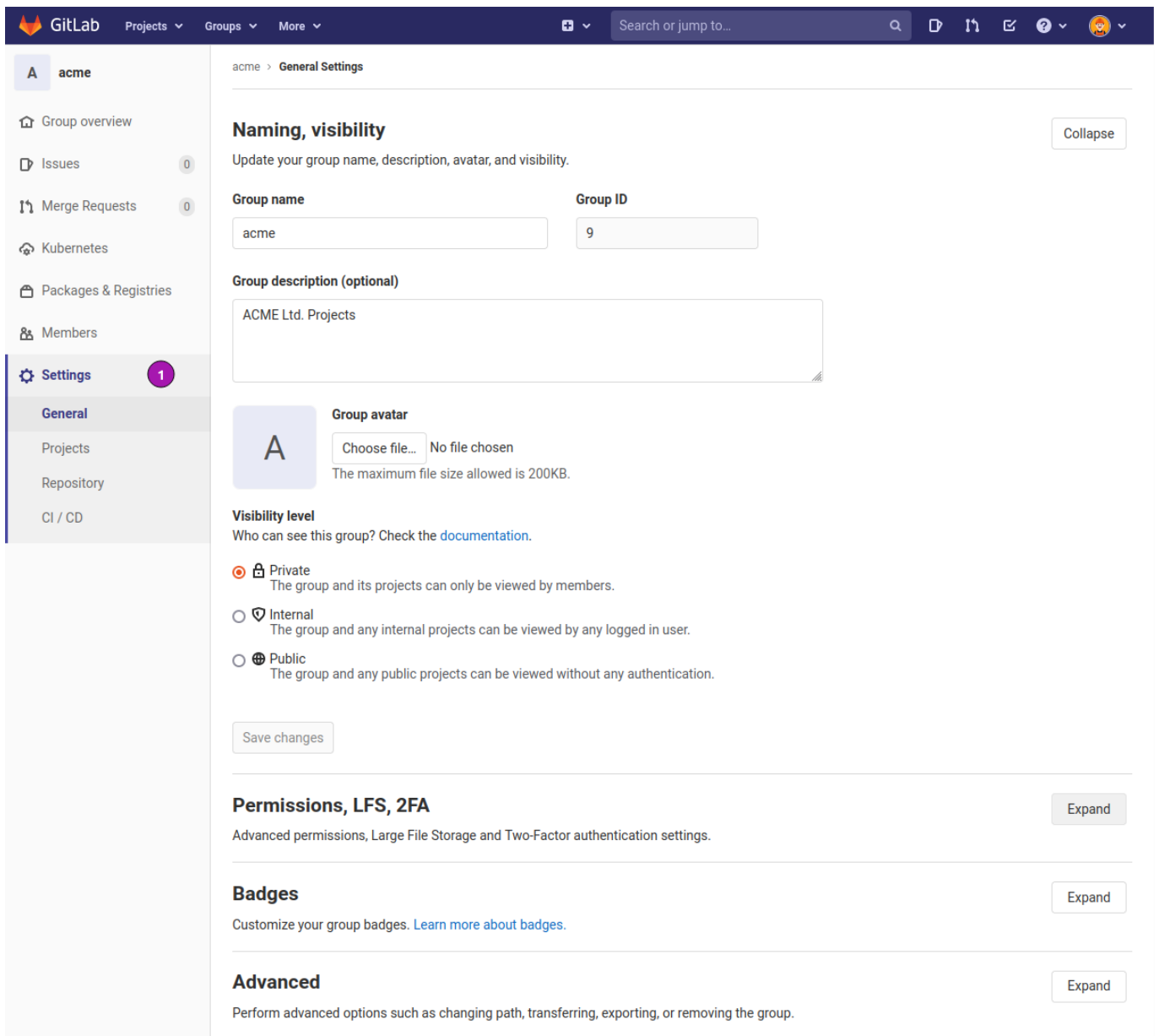


Figure 19. Settings screen for group

- 1 Choose **New Group** to open the creation form.

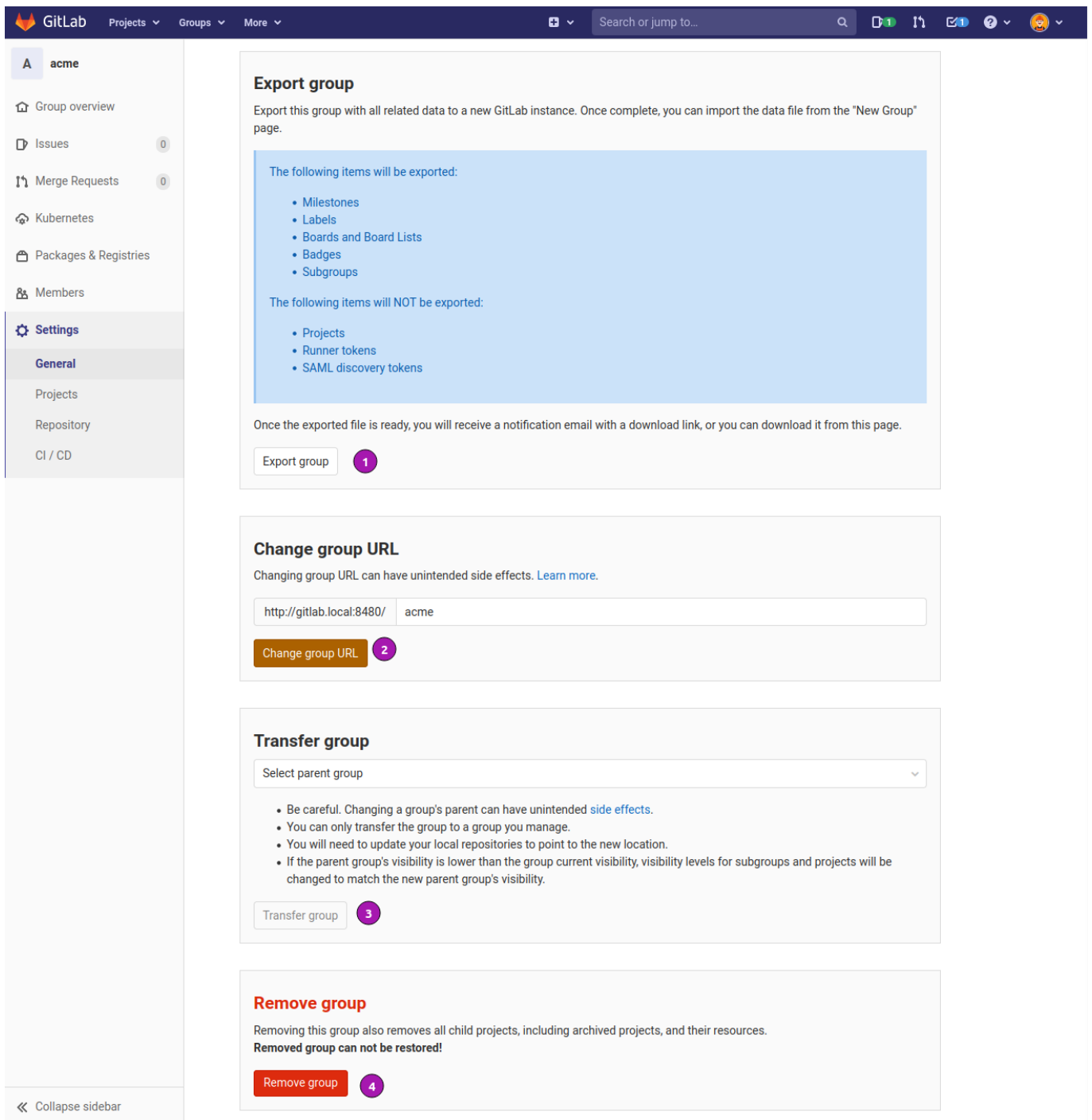


Figure 20. Advanced group settings

- 1 Export a group to be used on a other gitlab instance.
- 2 Change url. May have side-effects!
- 3 Transfer group. May have side-effects!
- 4 Remove group. No restore possible!

Module 4 - Forking projects

Forking is a way to keep a copy of a repository under the users own project. It is a way of collaboration between different users and isolates the work each user commits until a feature, bugfix or change has been completely tested or verified.

Goals

Contribute to various code bases by keeping the work performed on the repository completely isolated to one's own space.

- Create a new user
- Fork an existing project.
- Clone forked project.
- Branch local copy.
- Push changes to forked project.

Create new user (fork)

For discussing forking in detail a new user is required. Below are the instructions to do so.

Register user account

Open a new private or incognito browser window and navigate to the GitLab entry page → <http://gitlab.local:8480/>

Click on the [Register] tab and create a user with the following data:

Property	Value
First name	Mary
Last name	DevOps
User name	mdevops
Email	mary.devops@gitlab.local
Password	topsecret
Role	DevOps Engineer



A ssh public key is not required the change are made over the http protocol.

When finished change the avatar to match the description.

GitLab Projects Groups More Search or jump to...

Mary DevOps
@mdevops · Member since November 09, 2020

Overview Activity Groups Contributed projects Personal projects Starred projects Snippets

Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov

M
W
F

Issues, merge requests, pushes, and comments.

Activity Personal projects

No activities found This user doesn't have any personal projects

Figure 21. Mary DevOps user profile page

Create forked project

User Mary DevOps is going to fork the project **gitlab-project** of user Joe Developer. The steps are listed here.

Fork **gitlab-project**

As user Mary Devops navigate to URL → <http://gitlab.local:8480/jdev/gitlab-project>

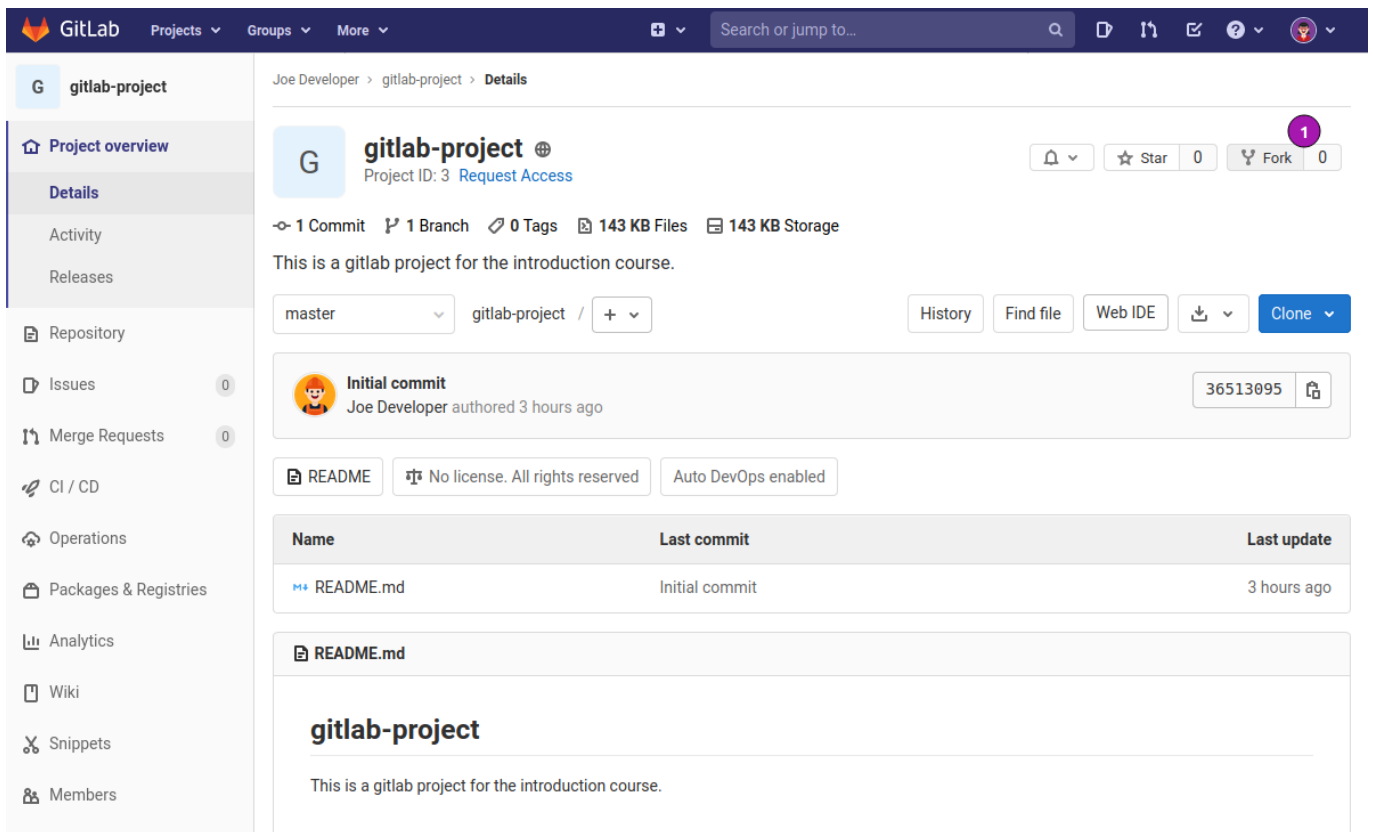


Figure 22. Fork the project **gitlab-project**

1 In the upper right corner click the **fork** button to initiate the fork.

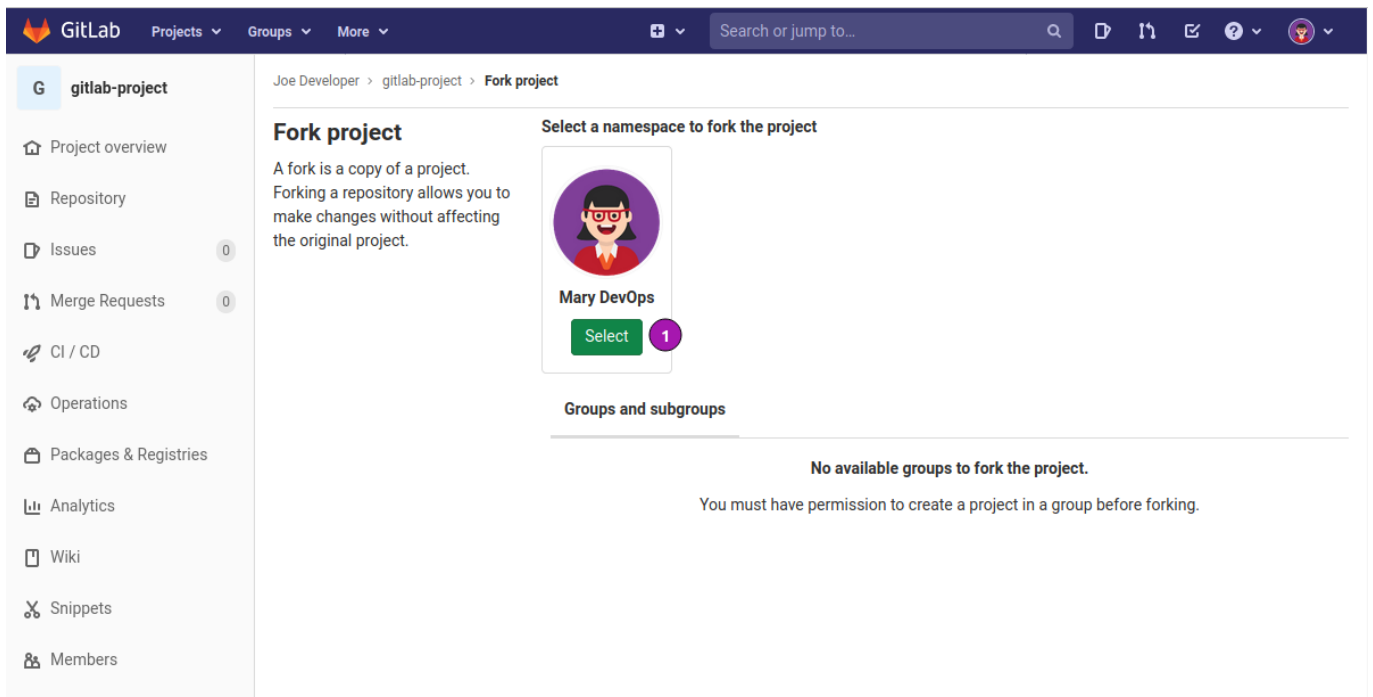


Figure 23. Select namespace for the fork.

1 Select Mary DevOps' workspace for the fork.

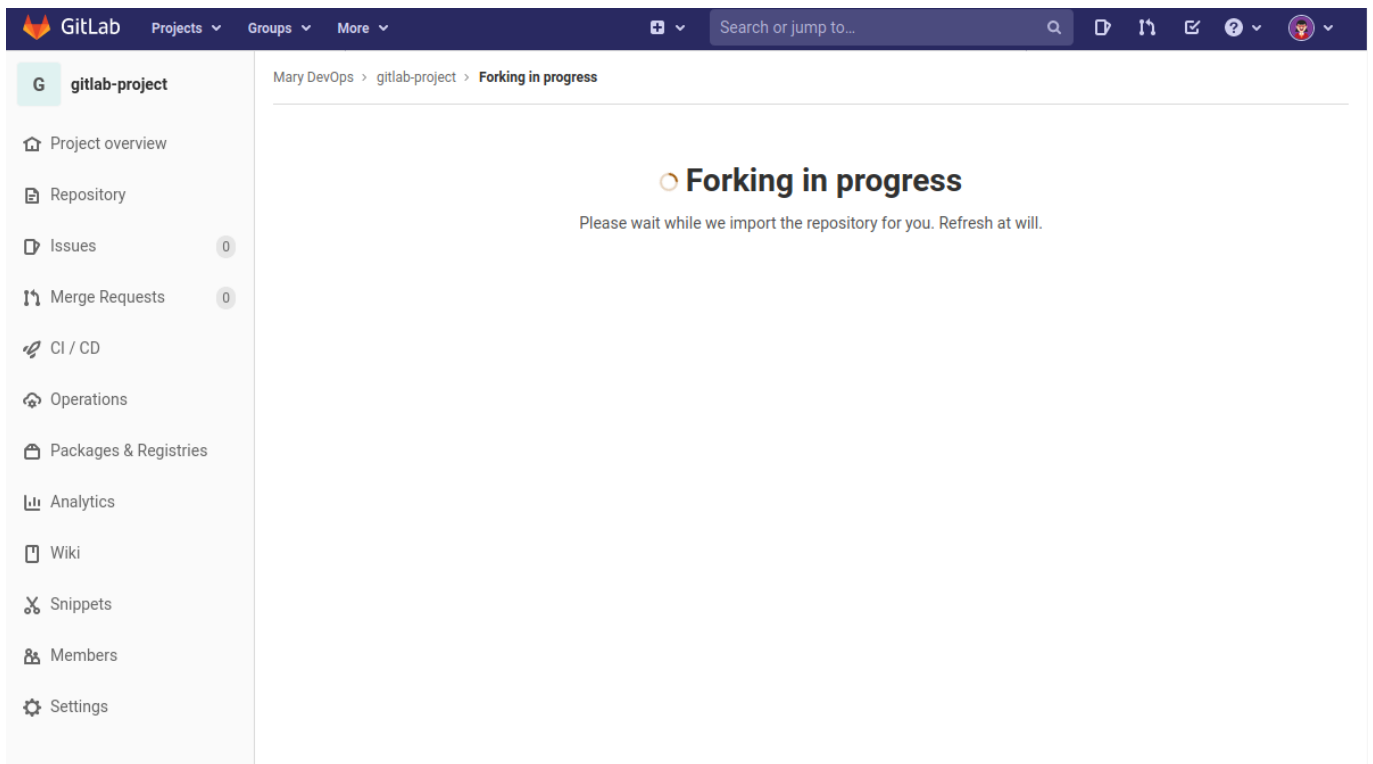


Figure 24. Fork in progress screen

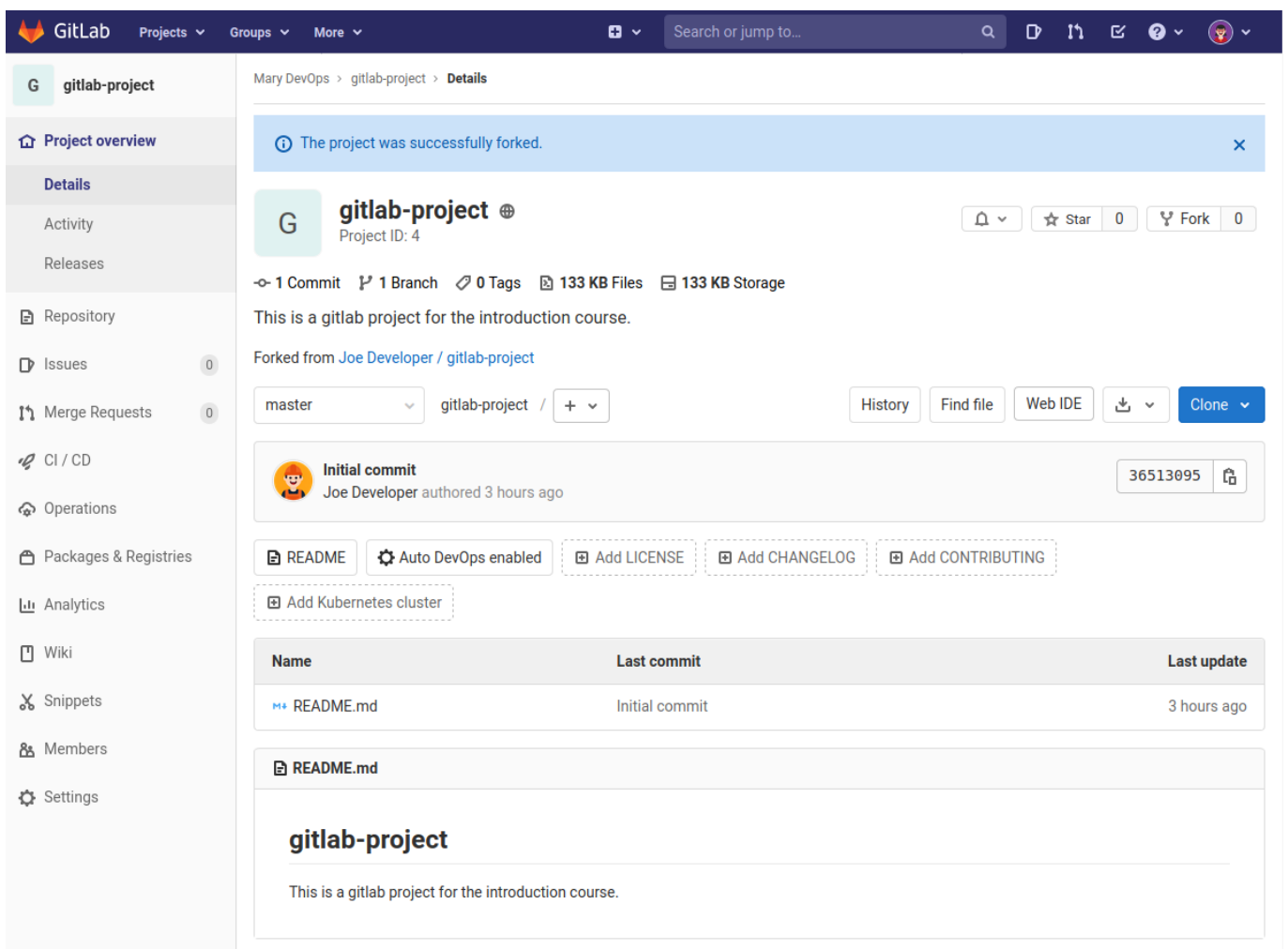


Figure 25. Forked project overview page.

Change content of forked project

With the project forked under Mary DevOps' namespace it is time to make changes to the content.

Clone forked project

Navigate to URL → <http://gitlab.local:8480/mdevops/gitlab-project/> Click on the [Clone] button to display and use the **Copy with HTTP URL** → <http://gitlab.local:8480/mdevops/gitlab-project.git>

Open terminal window to clone the project.

```
$ git clone http://gitlab.local:8480/mdevops/gitlab-project.git forked-project ①
Cloning into 'forked-project'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 258 bytes | 258.00 KiB/s, done.
```

- ① Clone project to directory **forked-project** to not mix up the already cloned copy from user Joe Developer.

To distinguish changes to the forked and original repository in case of a merge request. The freshly created **forked-project** user settings are locally modified.

```
$ cd forked-project/
$ git config --local user.name "Mary DevOps"
$ git config --local user.email "mary.devops@gitlab.local"
$ git config --local --get-regexp user
user.name Mary DevOps
user.email mary.devops@gitlab.local
```

Create branch

To work effectively encapsulate changes in git branches are recommended. When isolating each change in a branch it is possible to work on multiple different changes without independently. To follow this practice the first action in the **forked-project** repository is to create branch **documentation** and change to it.

```
$ git checkout -b documentation ①
Switched to a new branch 'documentation'
```

- ① Create and switch to branch **documentation**.

Modify existing files

Under the documentation branch Mary modifies the **README.md** file adding information about contributors.

```
$ git diff README.md ❶
diff --git a/README.md b/README.md
index 26a1ddf..3e727fd 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,7 @@
 # gitlab-project

-This is a gitlab project for the introduction course.
\ No newline at end of file
+This is a gitlab project for the introduction course.
+
+### Contributors
+* Joe Developer
+* Mary DevOps

$ git commit -m "README: Add contributors" README.md ❷
[documentation 67b94b1] README: Add contributors
 1 file changed, 5 insertions(+), 1 deletion(-)

$ git log ❸
commit 67b94b1886449c14db44524b1b67c71fe2250662 (HEAD -> documentation)
Author: Mary DevOps <mary.devops@gitlab.local>
Date:   Mon Nov 9 17:47:45 2020 +0100

    README: Add contributors

commit 3651309571108055dcafb3daca9678e90c91291 (origin/master, origin/HEAD,
master)
Author: Joe Developer <joe.developer@gitlab.local>
Date:   Mon Nov 9 12:10:30 2020 +0000

    Initial commit
```

- ❶ Display changes made to file **README.md** prior to committing.
- ❷ Commit the changes with message.
- ❸ Inspect the revision history with command **log**.

Push changes to project

With the changes made it is time to push them back onto the forked project.

```

$ git push origin documentation ❶
Username for 'http://gitlab.local:8480': mdevops ❷
Password for 'http://mdevops@gitlab.local:8480': ***** ❸
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 361 bytes | 361.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for documentation, visit:
remote:   http://gitlab.local:8480/mdevops/gitlab-project/-
remote:   /merge_requests/new?merge_request%5Bsource_branch%5D=documentation
remote:
To http://gitlab.local:8480/mdevops/gitlab-project.git
* [new branch]      documentation -> documentation ❹

```

- ❶ Explicitly provide the branch name **documentation** to push
- ❷ For the HTTP protocol the username is requested.
- ❸ The password must also be provided.
- ❹ The branch **documentation** is being created on the gitlab copy.



The username and password for HTTP operations can be cached with below command. The timeout value is seconds. **git config --global credential.helper 'cache --timeout=864000'**

Navigating once more to the project page of the fork → <http://gitlab.local:8480/mdevops/gitlab-project/> to view the changes in the interface.

The screenshot shows the GitLab web interface for a repository named 'gitlab-project'. At the top, a notification states 'You pushed to **documentation** 7 minutes ago' with a circled '1'. To the right of this notification is a 'Create merge request' button with a circled '2'. Below the notification, there is a dropdown menu for switching branches, currently set to 'master'. The dropdown menu is open, showing a search bar and a list of branches: 'documentation' (with a circled '3') and 'master' (checked). To the right of the branch dropdown, there are buttons for 'History', 'Find file', 'Web IDE', and 'Clone'. Below the branch dropdown, there is a table with columns for 'Last update' and '36513095'. The main content area shows the 'README.md' file for the 'gitlab-project' repository, which contains the text 'This is a gitlab project for the introduction course.'

Figure 26. The forked project's page after the push

- ❶ Notification about the recently pushed branch **documentation**.

- ② Convenience button to [Create merge request] right away.
- ③ The Switch branch/tag menu now contains branch **documentation**.

Switching the project's view to branch **documentation**.

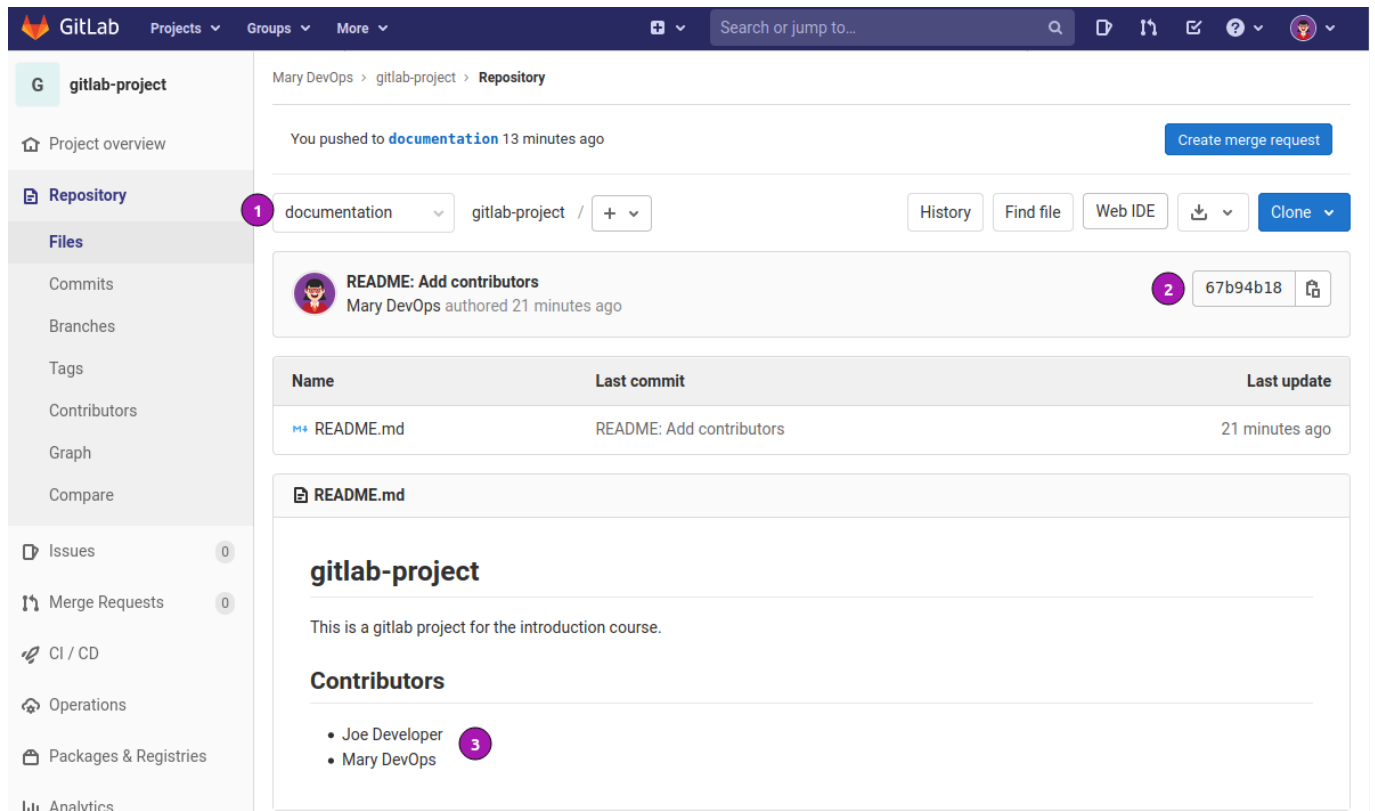


Figure 27. The branch **documentation** in the overview.

- ① Branch is set to **documentation**.
- ② The commit hash has changed and points to the latest commit in branch **documentation**. Also note the avatar of the committer has changed to Mary's.
- ③ The content of the **README.md** is rendered with the committed changes.

The next step is to propose the changes to Joe Developer with a merge request. The procedure is topic for the next module.

Summary

Summary of git commands used.

Item	Description	Resource
git checkout -b	Create and switch to newly created branch.	man git-checkout
git clone	Clone a remote repository or gitlab project.	man git-clone
git config --local	Apply configuration values to local repository.	man git-config
git diff	Display changes of uncommitted modifications.	man git-diff

Item	Description	Resource
git log	Display revision history.	man git-log
git pull	Fetch and then merge changes into the current working branch.	man git-pull
git push	Send local changes to the shared repository.	man git-push

Module 5 - Merge requests

GitLab's statement about merge requests:

Merge requests are a place to propose changes you've made to a project and discuss those changes with others

Interested parties can even contribute by pushing commits if they want to.

Goals

Learn how to collaborate, review and evaluate changes to a repository before adding the changed to the **master** branch. After a merge request has been put forward for review the process is shown of how evaluate the change comment on particulars in the change and request amendments and finally merge the code.

- Create a merge request from a forked project.
- Submit a merge request.
- Review changes in a merge request.
- Amend a merge request.
- Merge the request.

Create merge request

Initiate merge request

There is a few ways to initiate a merge request under GitLab. The variations and locations here are listed below.

For a limited time after pushing changes to a repository a message with a button is display in the upper half of the overview page.

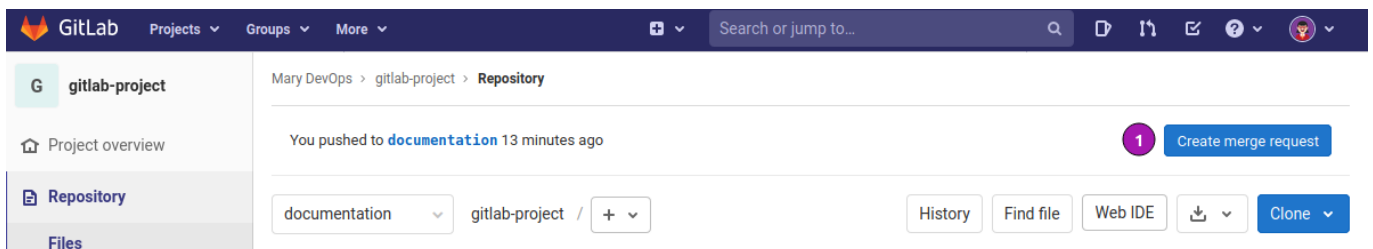


Figure 28. After pushing a branch notification appears on the overview page.

- 1 Click the [Create merge request] button to initiate.

Alternatively open a merge request via left hand side menu **Repository** → **Commits**

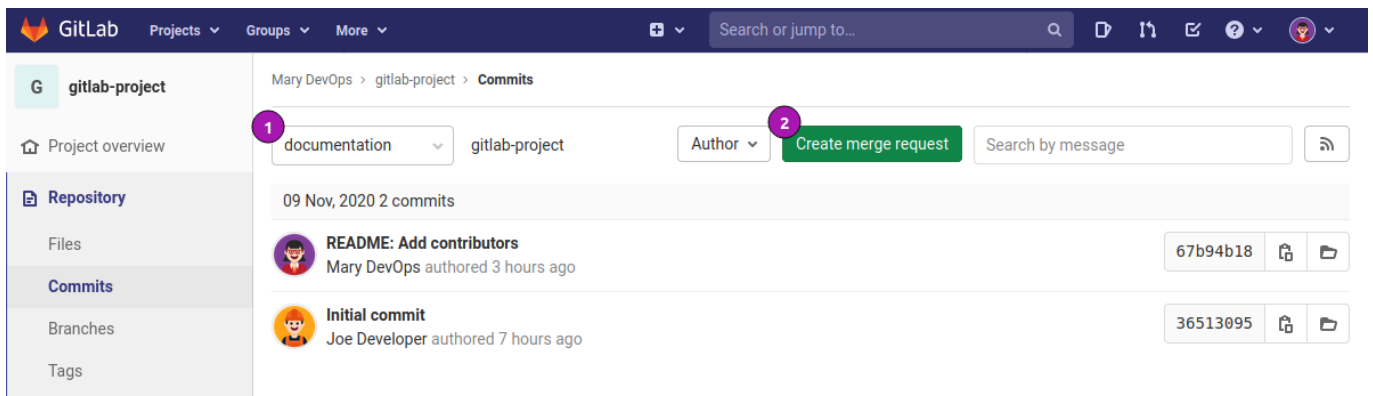


Figure 29. Create merge request via **Commits**

- ① Select the merge request branch e.g. **documentation**.
- ② Initiate via [Create merge request] button.

A third way is to go via the left hand menu **Repository** → **Branches** page.

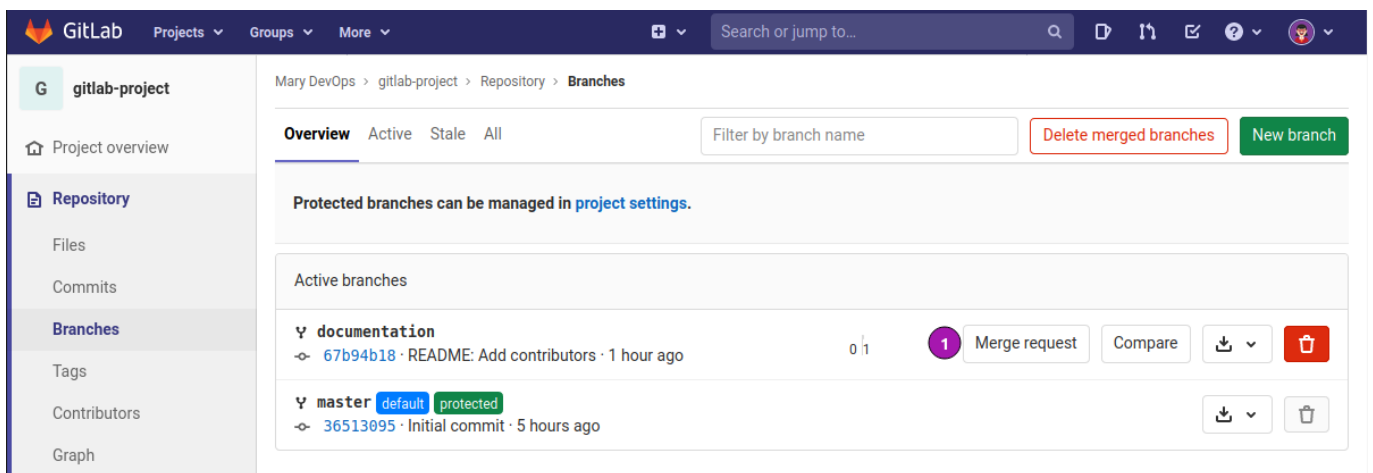


Figure 30. Create merge request via **Branches**

- ① Click on [Merge request] on the branch to be merged.

Submit merge request

To submit the merge request there are a few questions to be answered generally the defaults are fine most of the time.

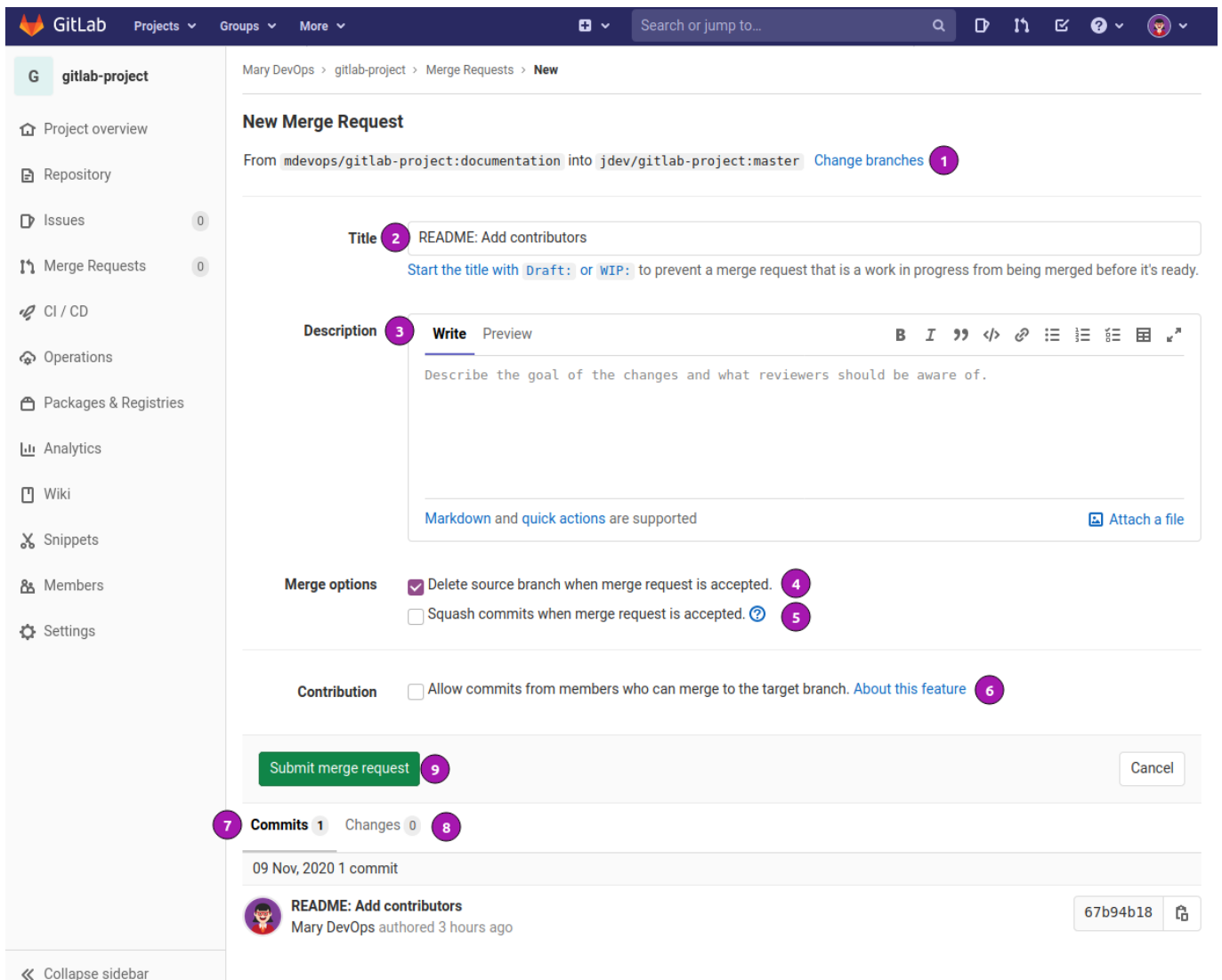


Figure 31. Create merge request via **Branches**

- 1 Change the source or / and destination branch for the merge request.
- 2 Change the title of the merge request. By default the title is the first line of the commit message.
- 3 Add a description. Per default it is filled from the commit message starting at line 2 and greater.
- 4 Deletes the source branch after the merge is done. This is checked by default and prevents out of date branches lingering around too long.
- 5 When a merge request contains more than one commit the commits are squashed into a single commit before applying to the target branch.
- 6 Allows the owner of the target branch to adjust the content of the to the merge request before merging.
- 7 List of commits contained in the merge request.
- 8 List of changes in diff format.
- 9 Submit the merge request to the target branch owner.

Review merge requests

View merge request

Once the merge request is submitted the other members of the project can view, comment and give suggestions for minor code changes.

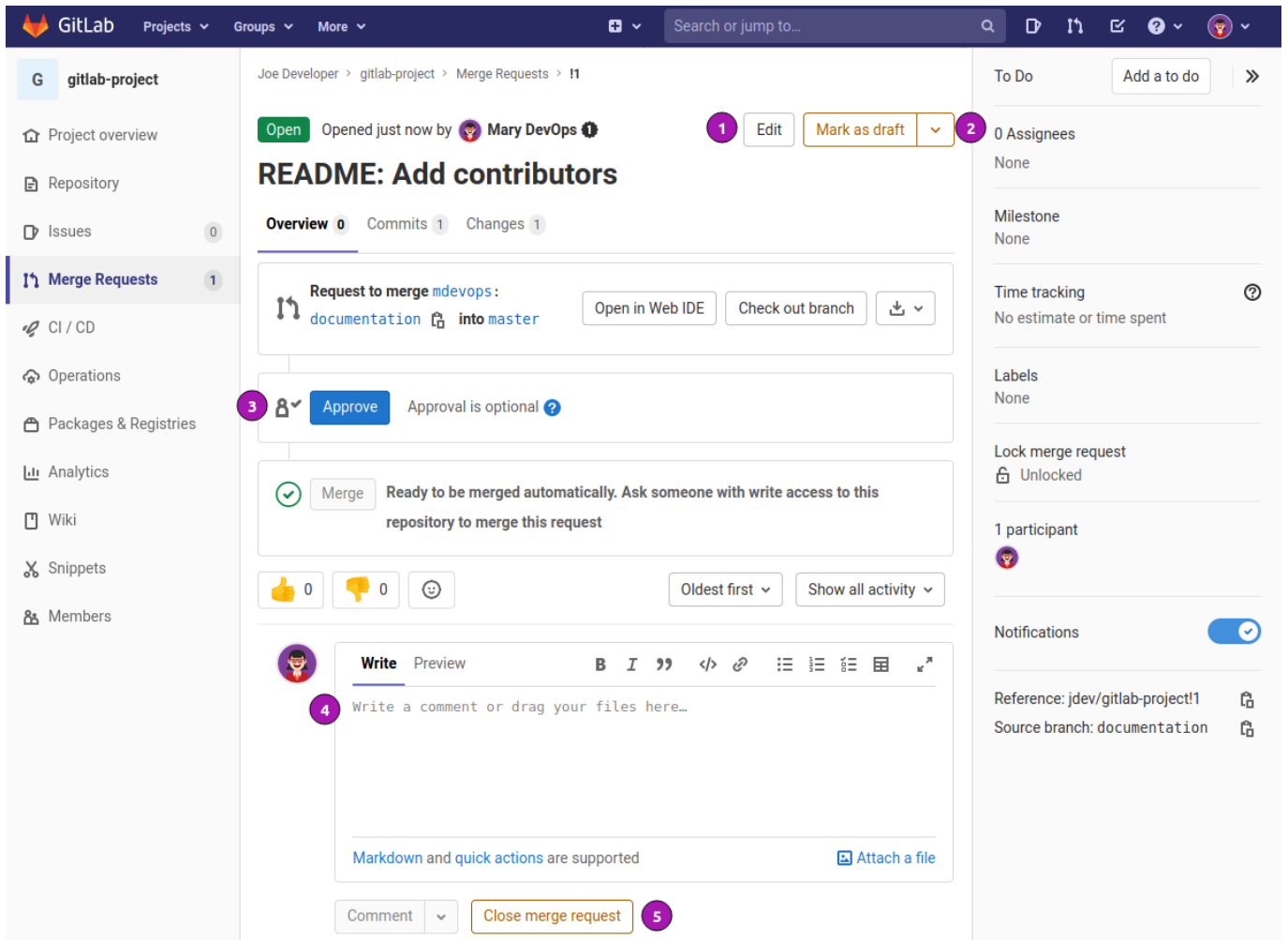


Figure 32. Merge request right after submission.

- 1 With [Edit] the title and message can be amended.
- 2 [Mark as draft] signals the maintainer that the request is not yet ready for production and requires more work before merger.
- 3 Used to [Approve] the change. Under the GitLab community edition this is gratuitous. The enterprise edition can enforce approval from stakeholders.
- 4 Add comments to the request.
- 5 [Close merge request] is another term for canceling the request. Note the request stays in history and can be reopened if required.

Add comment

Other users in the group are encouraged to review the changes and add comments.

Commenting under the [Changes] tab a reviewer can mark the lines of code the comment is referring to.

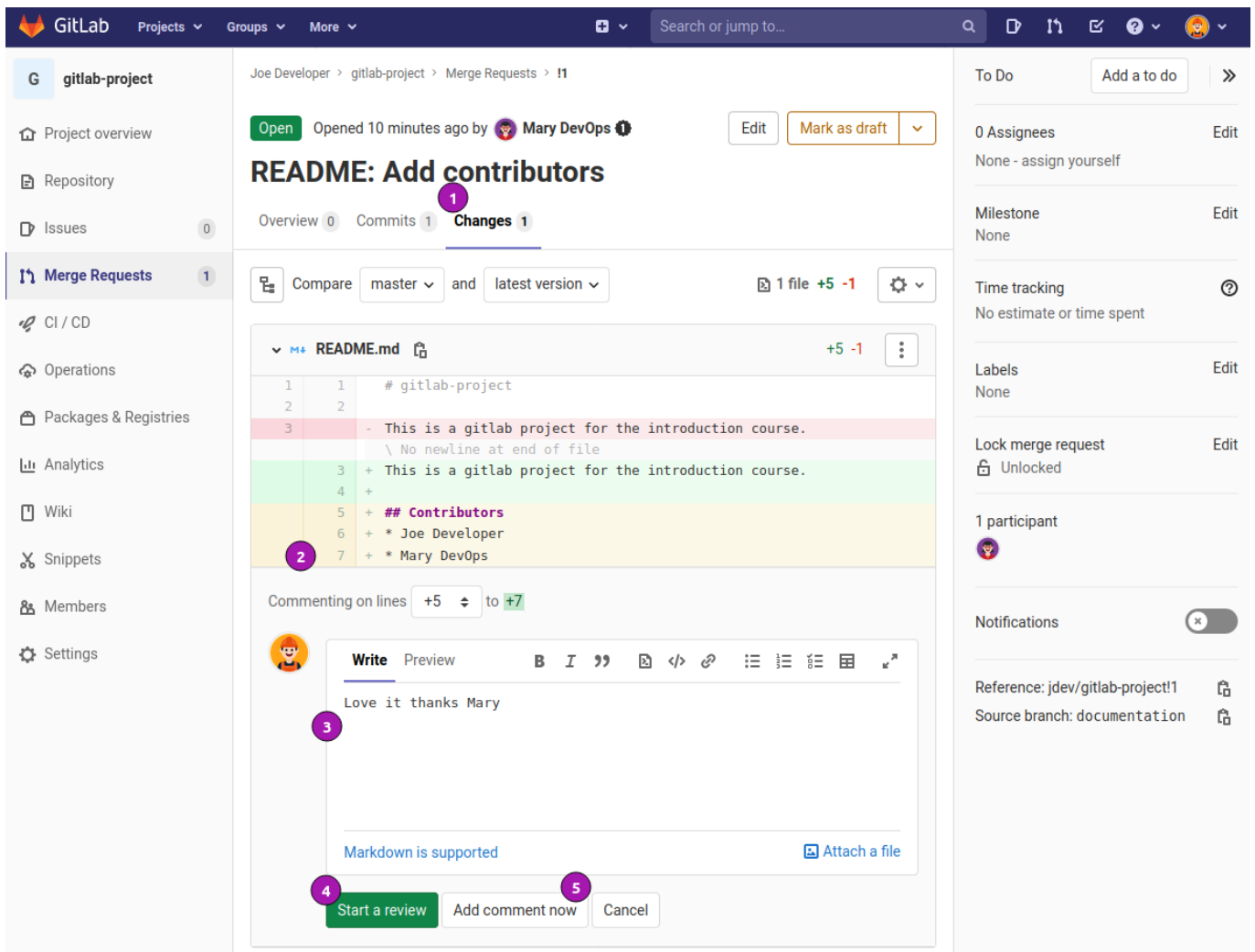


Figure 33. User Joe Developer adds a comment under changes.

- 1 Switched to tab [Changes].
- 2 Marking lines +5 ~ +7 as relevant to the comment.
- 3 Writing message about the previously marked lines.
- 4 If this change should be started as a review the [Start a review] button is clicked.
- 5 In this particular case it is a comment so [Add comment now] is clicked.

Once the comment is submitted the others can chime in. Since it is only a comment Mary DevOps is resolving the thread.

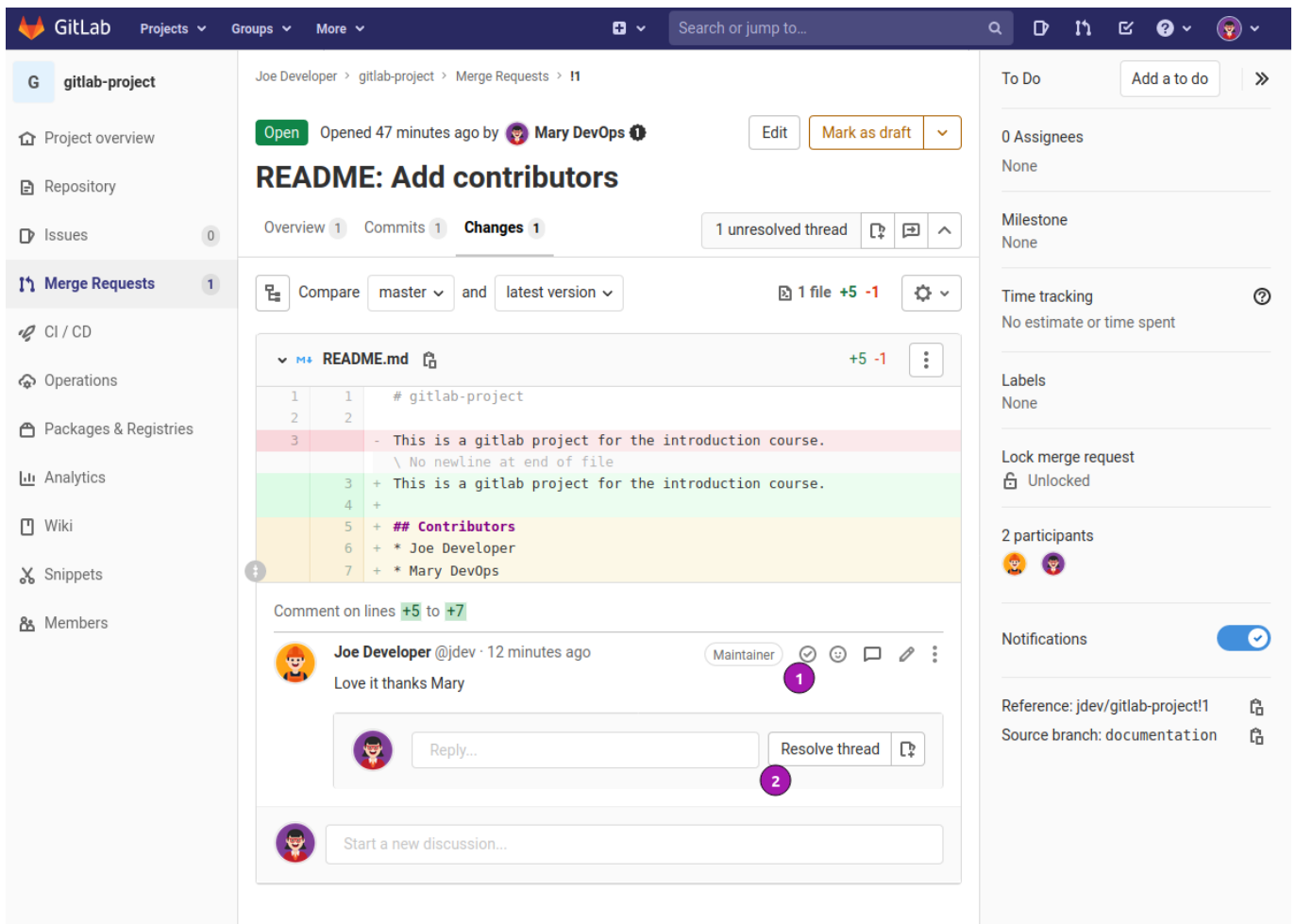


Figure 34. Mary DevOps views comment and resolves thread.

- 1 Resolving the thread via the circled check mark.
- 2 Alternatively via the button [Resolve thread].

Amend request

Maintainers have the option to make small amendments to the merge request. This can be done entirely within the merge request page.

The screenshot displays the GitLab interface for a merge request. The left sidebar shows navigation options: Project overview, Repository, Issues (0), Merge Requests (1), CI / CD, Operations, Packages & Registries, Analytics, Wiki, Snippets, Members, and Settings. The main content area shows a diff for 'README.md' with a table of changes:

Line	From	To
1	1	# gitlab-project
2	2	
3	-	This is a gitlab project for the introduction course.
		\ No newline at end of file
3	+	This is a gitlab project for the introduction course.
4	+	
5	+	## Contributors
6	+	* Joe Developer

Below the diff is a comment editor with a 'Write' tab and a 'Preview' tab. The comment text is: ``` suggestion: -0+0`
`* Joe Developer (Maintainer)`
````. The editor includes a 'Start a review' button (4), 'Add comment now', and 'Cancel' buttons. The right sidebar shows 'To Do' and 'Notifications' sections.

Figure 35. Create a suggestion

- 1 Select target line.
- 2 Click on the **insert suggestion** icon.
- 3 Modify the selected code.
- 4 [Start a review] to submit the suggestion.



Amendments are only available if the option for **Allow commits from member who can manage the target branch** has been checked during merge request submission.



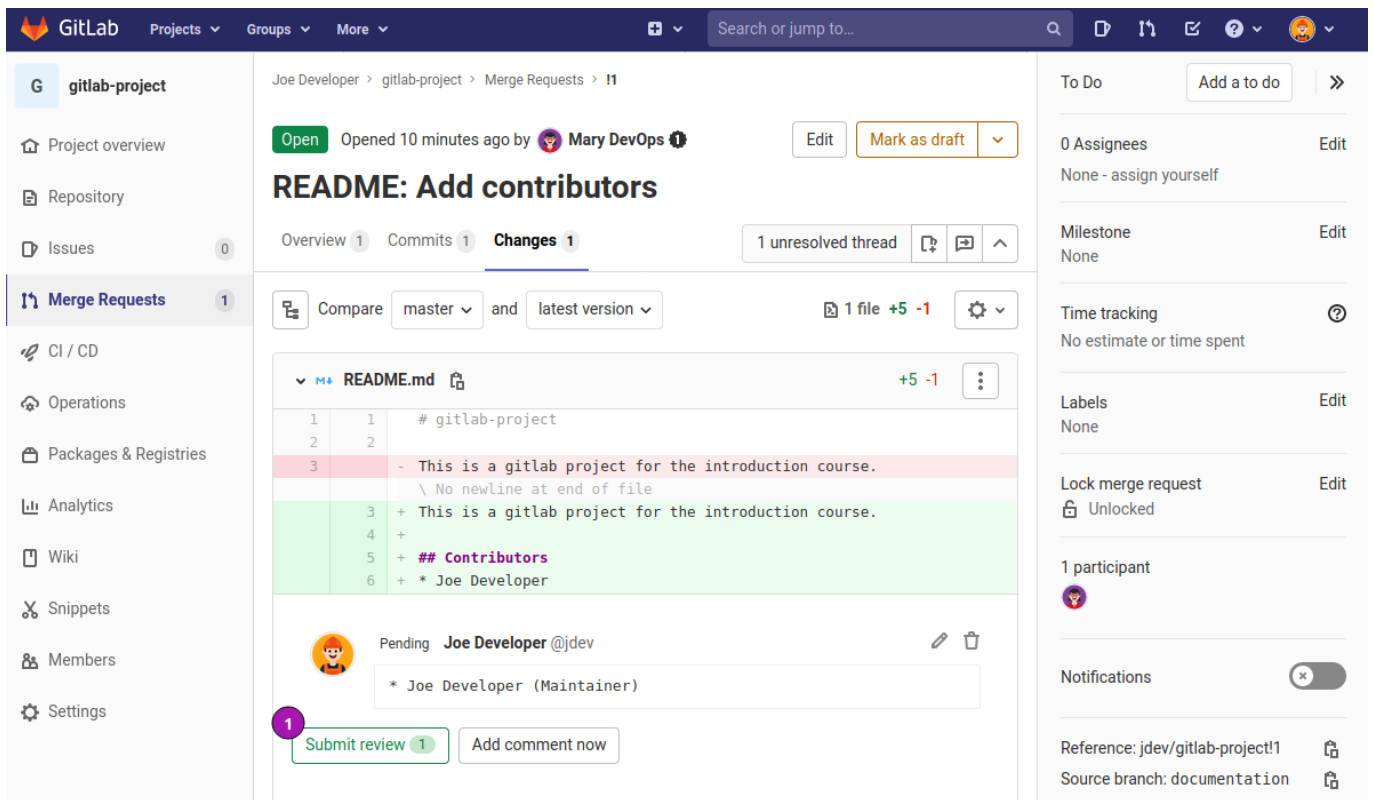


Figure 36. Submit the suggestion as review

- 1 To submit the suggestion for good click [Submit review]

Once the suggestion has been submitted other maintainers can further review the new amendment and comment.

The screenshot shows a GitLab Merge Request for a project named 'gitlab-project'. The merge request is titled 'README: Add contributors' and was opened 1 minute ago by 'Mary DevOps'. The interface shows a diff for the file 'README.md', comparing the 'master' branch with the 'latest version'. The diff highlights changes to the file's content, including the addition of a 'Contributors' section. A suggestion by 'Joe Developer' is shown, with a diff view of the change. The suggestion includes an 'Apply suggestion' button (1), a diff view of the change (2), and a 'Reply...' input field (3). The right sidebar contains various settings and options, including 'To Do', 'Assignees', 'Milestone', 'Time tracking', 'Labels', 'Lock merge request', and 'Notifications'.

Figure 37. Suggestion ready for review or application.

- 1 Clicking on [Apply suggestion] will create a new commit to the merge request.
- 2 The diff of the suggestion is shown to make suggestion review easy.
- 3 Further comments can be appended to the suggestion.

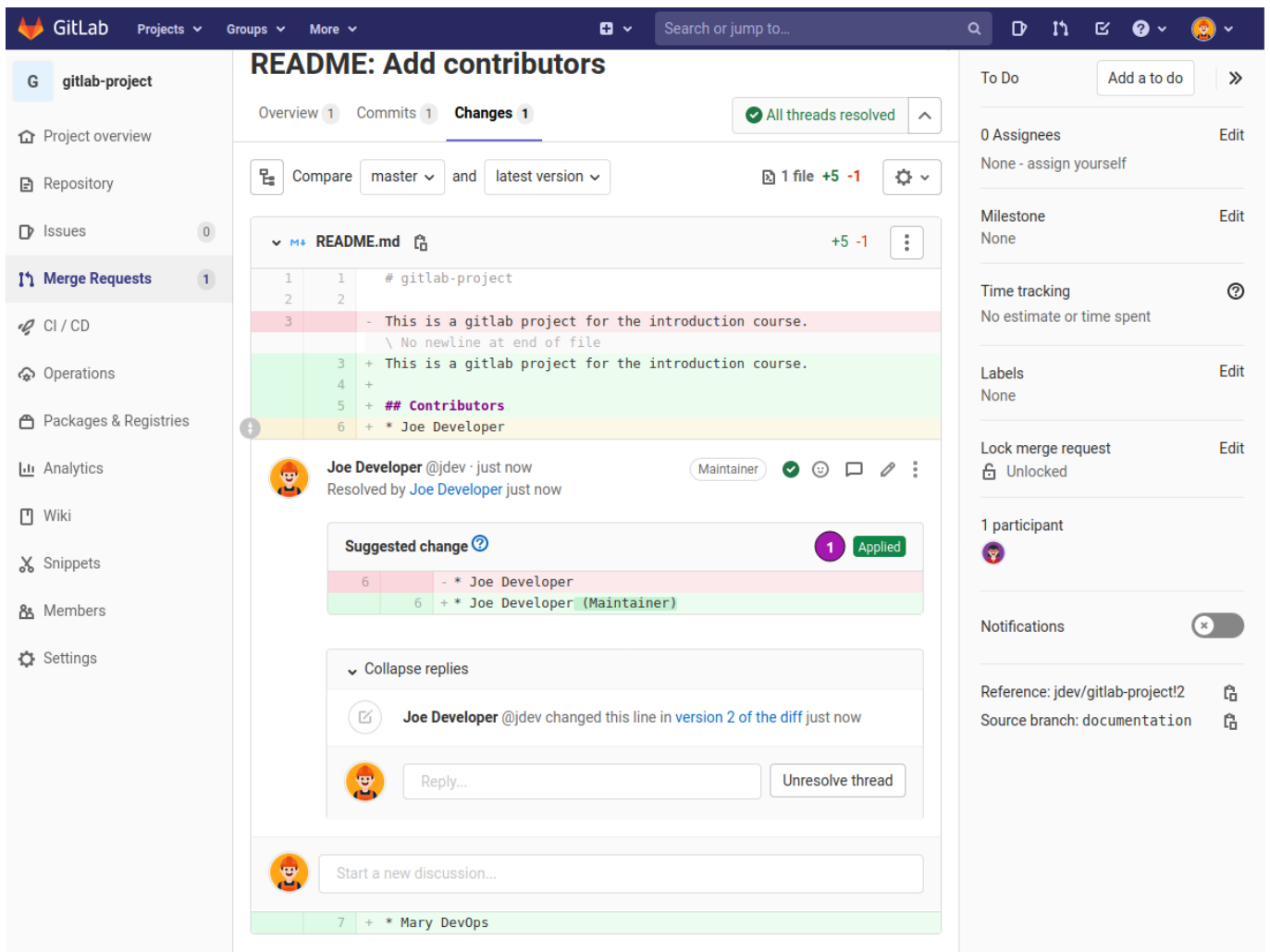


Figure 38. Applied suggestion.

- 1 After application of the suggestion the [Applied] indicator is added.

## Merge the request

Once the stakeholders are satisfied with the changes, suggestions and amendments the request can be merged by a maintainer.

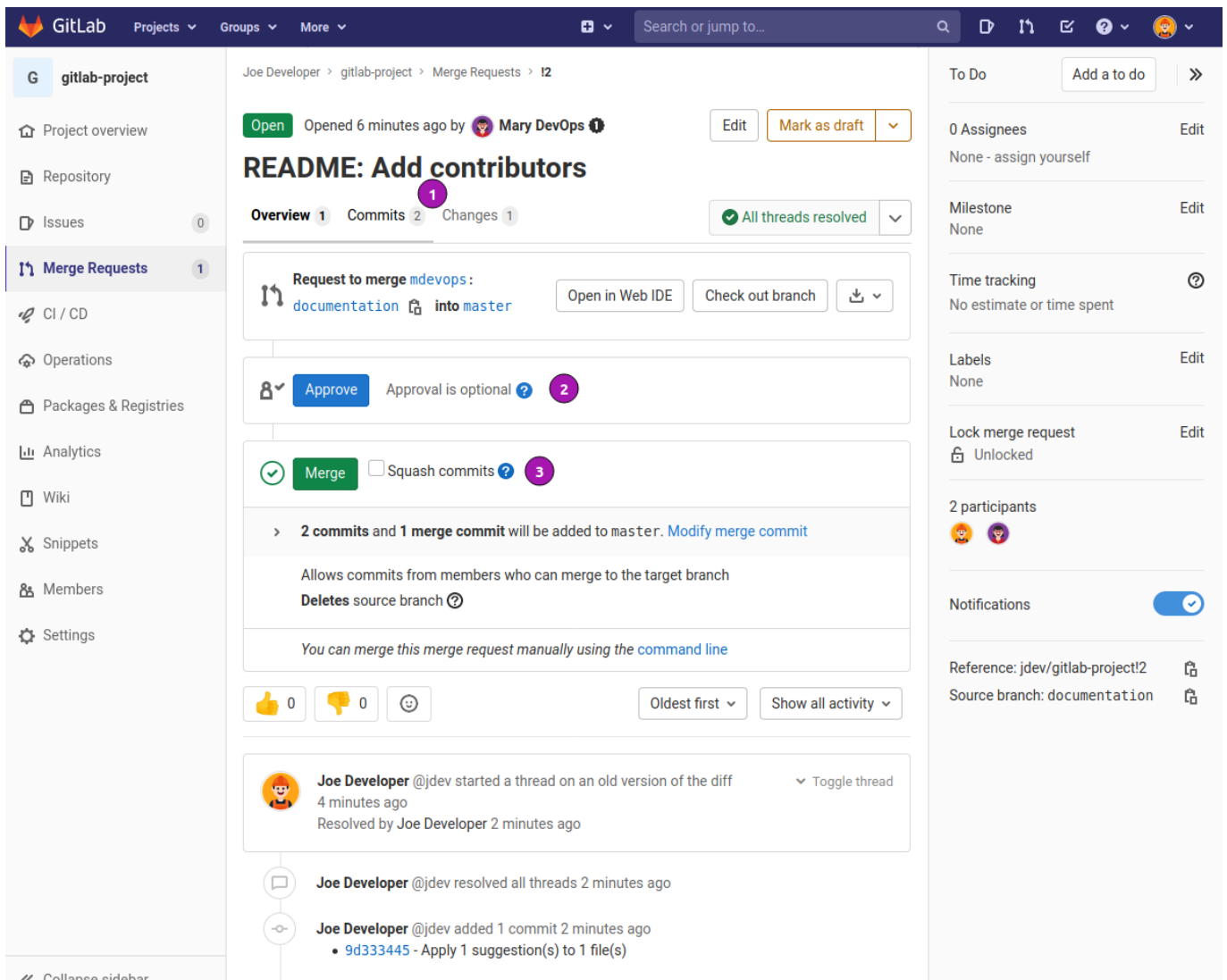


Figure 39. Merge the request

- 1 Take note with the suggestion applied there are 2 commits. At the outset of the request there was only one.
- 2 Approval of a merge request is not a requirement but can be a helpful tool to determine who has reviewed the change.
- 3 With the applied suggestion the **Squash commits** checkbox can be activated to only have one commit at merge time.

## Post merge review

Eventually the merge request was merged in this particular case with the **Squash commits** option checked.

It is now time to find out how the project repository changed.

## Web UI post merge review

In the Web UI the changes are very subtle and the complexity of the process is masked.

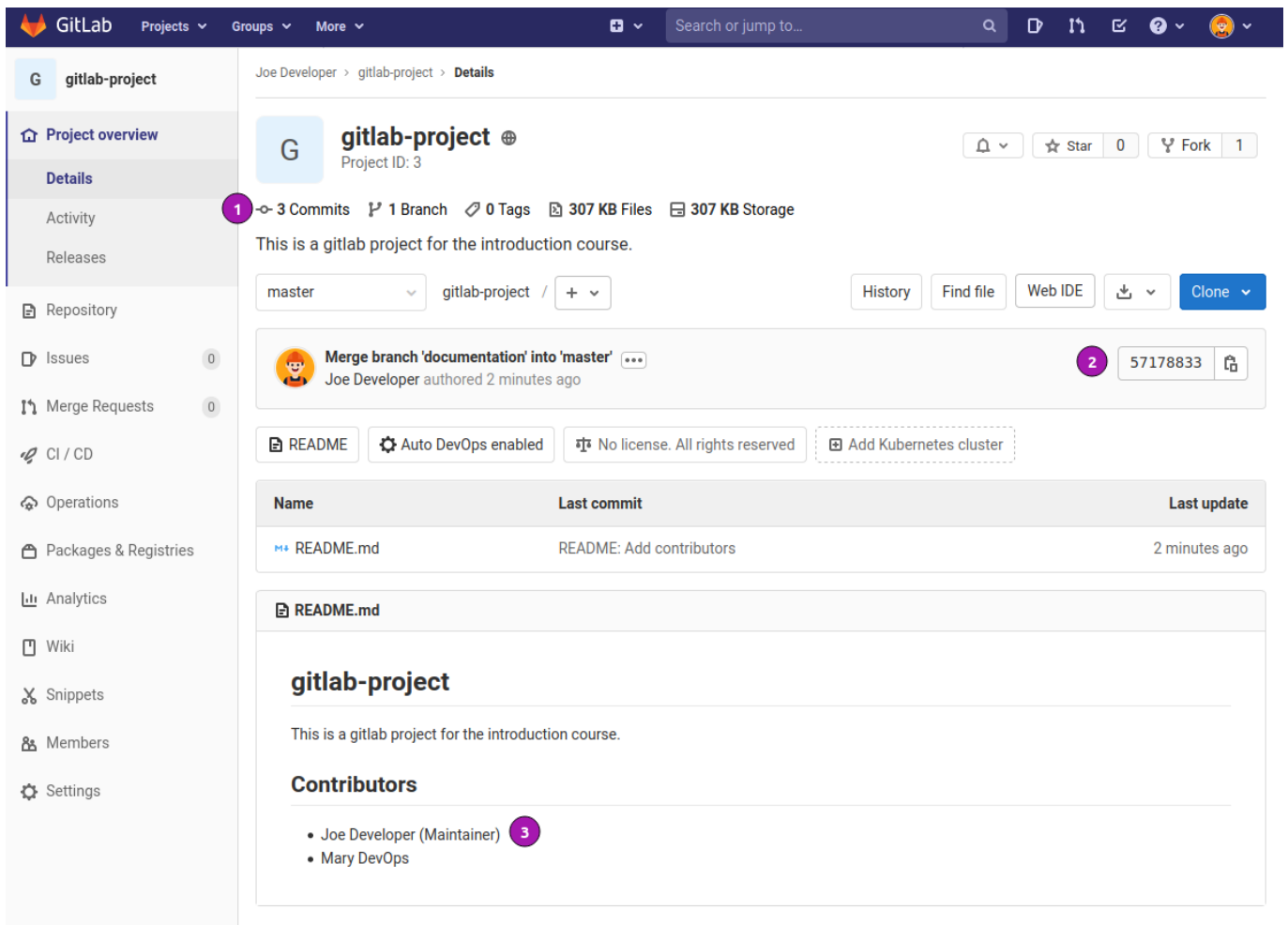


Figure 40. Project overview page post merge.

- ① Although the option **Squash commits** was checked there are now 3 commits present in the repository. It becomes more clear in as the topic evolves.
- ② The commit SHA1 is different to the one from Mary DevOps when submitting the merge request.
- ③ The changes including the suggestion from Joe Developer have are displayed in the rendered **README.md**.

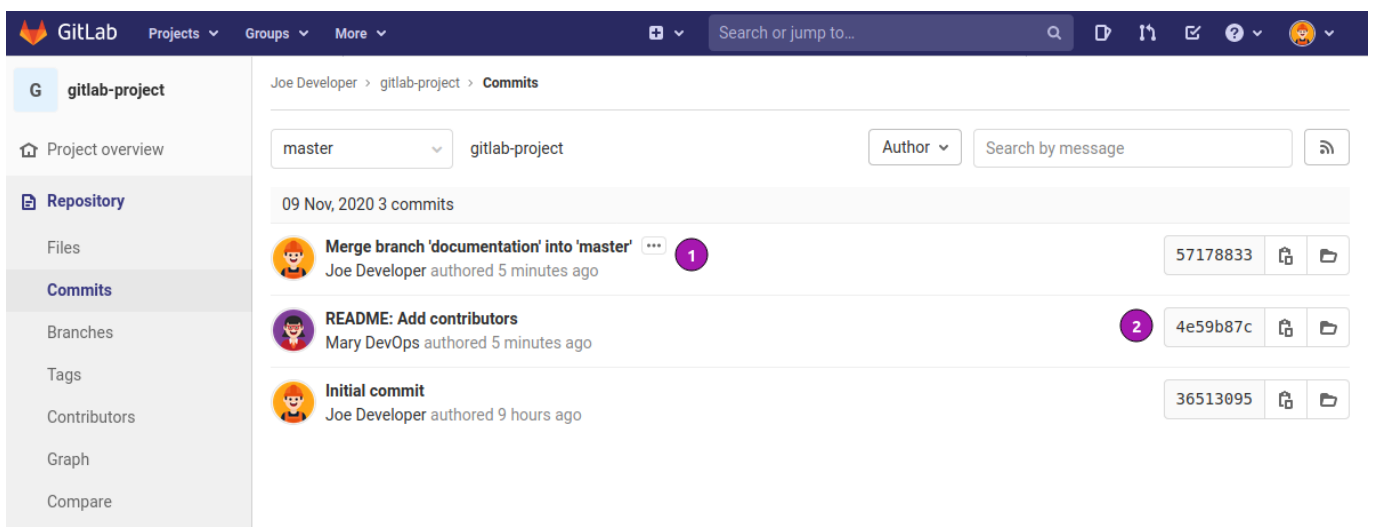


Figure 41. Review of the commits present.

- ① A new commit with a commit message never entered during the process is now the **HEAD** of the repository.

- 2 The commit hash changed compared to the submitted one. What happened?

## git command post merge review

It is sometime easier to review the changes that look confusing in the Web UI on the command line.

```
$ git pull origin master ①
X11 forwarding request failed on channel 0
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 513 bytes | 513.00 KiB/s, done.
From ssh://gitlab.local:8422/jdev/gitlab-project
 * branch master -> FETCH_HEAD
 3651309..5717883 master -> origin/master
Updating 3651309..5717883
Fast-forward
 README.md | 6 +++++- ②
 1 file changed, 5 insertions(+), 1 deletion(-)

$ git log ③
commit 57178833a3b5b0881becc92de93a45357e2c54ec (HEAD -> master, origin/master,
origin/HEAD)
Merge: 3651309 4e59b87 ④
Author: Joe Developer <joe.developer@gitlab.local>
Date: Mon Nov 9 21:39:52 2020 +0000

 Merge branch 'documentation' into 'master'

 README: Add contributors

 See merge request jdev/gitlab-project!2

commit 4e59b87c9a4cd0d41837374e5ab257ecc67d5251
Author: Mary DevOps <mary.devops@gitlab.local>
Date: Mon Nov 9 21:39:52 2020 +0000

 README: Add contributors

commit 3651309571108055dcafb3daca9678e90c91291
Author: Joe Developer <joe.developer@gitlab.local>
Date: Mon Nov 9 12:10:30 2020 +0000

 Initial commit
```

- 1 Pull the changes from the GitLab project to the previously created repository.
- 2 The file **README.md** is updated during this procedure.
- 3 The **git log** command displays a bit more information about commit **5717883**. This is a merge commit with multiple parents.

- 4 The merge line display all the parents this particular commit has. Merge commits can be prevented in under **Settings** → **General** → **Merge requests** → (\*) Fast-forward merge.

Unresolved directive in gitlab-tutorial.adoc - include::modules/06-module06/00-merge-requests.adoc[]



## Module 6 - CI/CD

CI/CD stands for Continuous Integration and Continuous Delivery and is an integral part of GitLab. The module does teach the installation, setup and registration of a GitLab runner and the creation of a simple pipeline with multiple stages.

### Goals

project in the already existing GitLab project.

Most of the action is taking place on the command line with a few peeks at the result in the GitLab frontend.

- The different types of runners.
- Installation and registration of a GitLab runner.
- CI/CD setup in GitLab.
- Creation of a simple pipeline.
- Creation of a multi stage pipeline.

### Gitlab runner types

There are 3 types of runners.

#### Shared

Available to all groups and projects.

#### Group

Available to the whole group.

#### Specific

Assigned to one or more project.

### Gitlab runner setup

#### Create a test project

The gitlab runner is written in Go which produces all in one binaries. There are packages and or binaries for all major operating systems available. The install instructions can be at this URL <https://docs.gitlab.com/runner/install/>

For the purpose of this module the GitLab runner is not installed but only run on the command line.

To start create a new repository called **gitlab-runner-test** either in the GitLab frontend or as shown in the command line instructions below.

```
git init gitlab-runner-test
cd gitlab-runner-test
git remote add origin git@<gitlab-domain>:<user>/gitlab-runner-test.git ❶
touch .gitlab-ci.yml ❷
git add .gitlab-ci.yml ❸
git commit -m "Empty pipeline" ❹
git push origin main ❺
```

- ❶ Use the domain of your GitLab server followed by either your username or a group name.
- ❷ Create and empty `.gitlab-ci.yml` file.
- ❸ Add to local git repository.
- ❹ Commit the empty file.
- ❺ Push to GitLab. This will create a new project on the path specified with the `git remote` command.

## Register a specific runner

**Runners** ❸

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

**How do runners pick up jobs?**

Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure shared runners only handle the jobs they are equipped to run. [Learn more.](#)

**Specific runners**

These runners are specific to this project.

**Set up a specific runner for a project**

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:  
`https://gitlab.com/`

And this registration token:  
`GR1348941zGxUXwUpzYSBT8syeg4R` ❹

[Reset registration token](#)

[Show runner installation instructions](#)

**Shared runners**

[These runners](#) are available to all groups and projects.

Each CI/CD job runs on a separate, isolated virtual machine.

**Enable shared runners for this project**

**Available shared runners: 50**

- #11728737 (kpqQ46jn) [🔗](#)  
1-green.shared-gitlab-org.runners-manager.gitlab.com/dind  
[gitlab-org-docker](#)
- #11574076 (8zCxmPt) [🔗](#)  
2-green.shared-gitlab-org.runners-manager.gitlab.com  
[gitlab-org](#)

- ❶ Navigate to Settings → CI/CD.
- ❷ Expand the Runners tab.
- ❸ Take note of the sites URL.
- ❹ Copy the registration token.

## Registration of the Runner

```
gitlab-runner register
Runtime platform arch=amd64 os=linux
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://<gitlab-domain>/ ①
Enter the registration token:
GR1348941zGxUXwUpzYSBT8syeg4R ②
Enter a description for the runner:
[foobar-runner]: foobar ③
Enter tags for the runner (comma-separated):
shell, linux ④
Enter optional maintenance note for the runner:

Registering runner... succeeded runner=GR1348941zGxUXwUp
Enter an executor: custom, docker, shell, virtualbox, docker-ssh+machine,
kubernetes, docker-ssh, parallels, ssh, docker+machine, instance:
shell ⑤
Runner registered successfully. Feel free to start it, but if it's running
already the config should be automatically reloaded!

Configuration (with the authentication token) was saved in
"/home/.../.gitlab-runner/config.toml"
```

- ① Provide the url for your GitLab setup.
- ② Provide the token from the registration page.
- ③ Provide a unique name for the runner.
- ④ Add tags to the runner.
- ⑤ Choose the execution type for the runner. E.g. **shell**.

## Start the runner

The runner is now configured for a single user. To start it simply type **gitlab-runner run** and let it run in the foreground of the terminal.

```
$ gitlab-runner run
Runtime platform arch=amd64 os=linux
pid=297281 revision=133d7e76 version=15.6.1
Starting multi-runner from /home/uroesch/.gitlab-runner/config.toml... builds=0
WARNING: Running in user-mode.
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

Configuration loaded builds=0
listen_address not defined, metrics & debug endpoints disabled builds=0
[session_server].listen_address not defined, session endpoints disabled
builds=0
Initializing executor providers builds=0
Checking for jobs...nothing runner=nCtzPEyx
[...]
```

## Simple pipeline

A simple pipeline is the most basic CI/CD Scripts possible. The pipeline created in this exercise is the simplest possible with a specific runner.

### .gitlab-ci.yml

Edit the `.gitlab-ci.yml` file with the content shown below.

```
first-pipeline: ❶
 tags:
 - shell ❷
 script:
 - echo "First pipeline" ❸
```

- ❶ Name of the task.
- ❷ The tag where to execute the instructions on.
- ❸ Command to execute.

### Track status

Commit and push to the GitLab server.

```
git commit -m "First pipeline" .gitlab-ci.yml
git push origin main
```

Navigate to the project CI/CD page.

The screenshot shows the GitLab interface for a project named 'gitlab-runner-test'. The left sidebar contains a navigation menu with 'Pipelines' highlighted and marked with a pink circle '1'. The main content area shows the 'Pipelines' page with a filter set to 'All' (1) and tabs for 'Finished', 'Branches', and 'Tags'. There are buttons for 'Clear runner caches', 'CI lint', and 'Run pipeline'. A search bar for 'Filter pipelines' and a 'Show Pipeline ID' dropdown are also visible. Below this is a table of pipeline runs:

| Status | Pipeline                                                   | Triggerer | Stages |
|--------|------------------------------------------------------------|-----------|--------|
| passed | First pipeline<br>#715308731<br>main -> c04d405f<br>latest |           | ❷      |

- ❶ Navigate to CI/CD → Pipelines in your project.
- ❷ See the pipeline status of the commit.

The screenshot shows the GitLab CI/CD interface. On the left is a navigation sidebar with options like Project information, Repository, Issues, Merge requests, CI/CD, Jobs, Schedules, Security & Compliance, Deployments, Packages and registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings. The main area displays a job log for 'first-pipeline' triggered 11 minutes ago by Urs Roesch. The log shows the following steps:

```

1 Running with gitlab-runner 15.6.1 (133d7e76)
2 on foobar nCtzPEyx
3 Preparing the "shell" executor 00:00
4 Using Shell executor...
6 Preparing environment 00:00
7 Running on uroesch-puzzle...
9 Getting source from Git repository 00:02
10 Fetching changes with git depth set to 20...
11 Initialized empty Git repository in /home/uroesch/Nextcloud/Documents/Technology/Tutorials/Gitlab/_src/modules/06-module06/builds/nCtzPEyx/0/uroesch/gitlab-runner-test/.git/
12 Created fresh repository.
13 Checking out c04d405f as main...
14 Skipping Git submodules setup
16 Executing "step_script" stage of the job script 00:00
17 $ echo "First pipeline"
18 First pipeline
20 Cleaning up project directory and file based variables 00:00
22 Job succeeded

```

On the right, a summary for 'first-pipeline' is shown:

- Duration: 5 seconds
- Finished: 11 minutes ago
- Queued: 0 seconds
- Timeout: 1h (from project)
- Runner: #19608584 (nCtzPEyx) foobar
- Tags: shell
- Commit c04d405f
- First pipeline
- Pipeline #715308731 for main
- test
- first-pipeline

## Staged pipeline

Building on the simple pipeline the scope is expanded to simulate several stages such as build, test and deploy.

## Stages in .gitlab-ci.yml

Edit the existing `.gitlab-ci.yml` file to look like the one below.

```

build:
 stage: build ❶
 tags: [shell]
 script: ['echo "Build project"']
test:
 stage: test ❷
 tags: [shell]
 script: ['echo "Test project"']
deploy:
 stage: deploy ❸
 tags: [shell]
 script: ['echo "Deploy project"']

stages: [build, test, deploy]

```

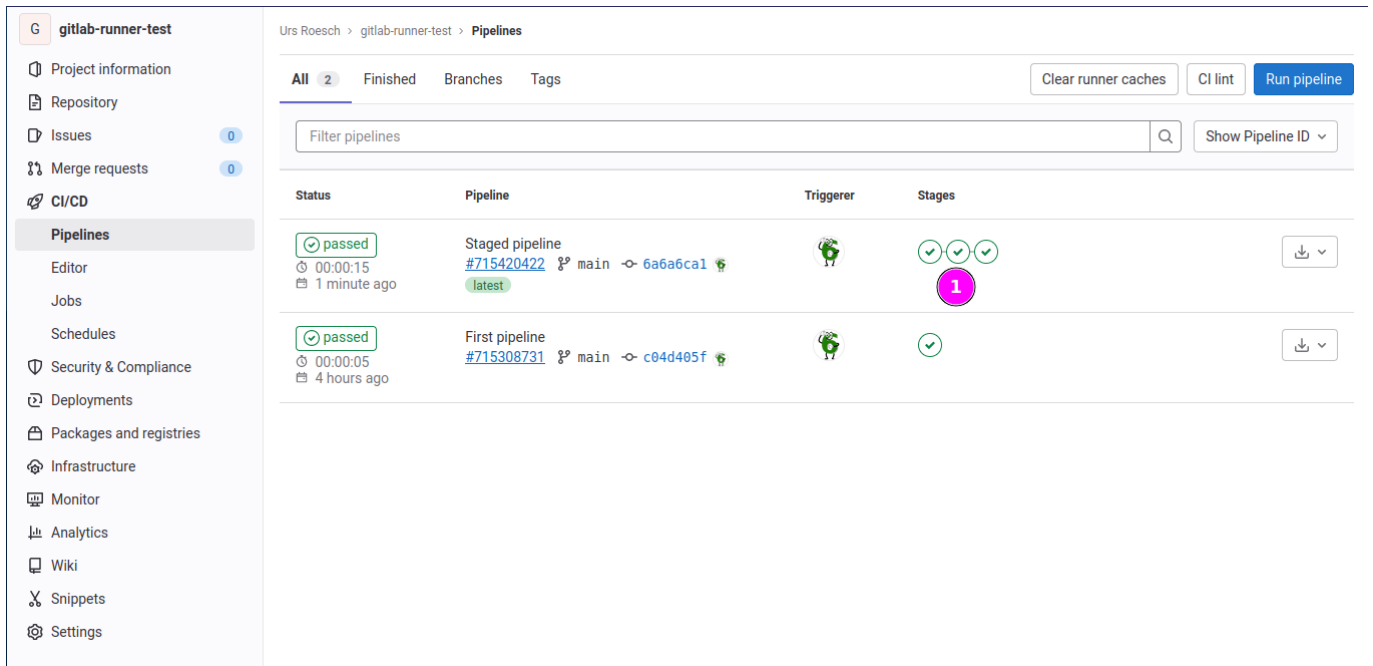
- ❶ Add the stage **build**.
- ❷ Second block is the **test** stage.
- ❸ The last block is **deploying** the code.

## Track status

Commit and push to the GitLab server.

```
git commit -m "Staged pipeline" .gitlab-ci.yml
git push origin main
```

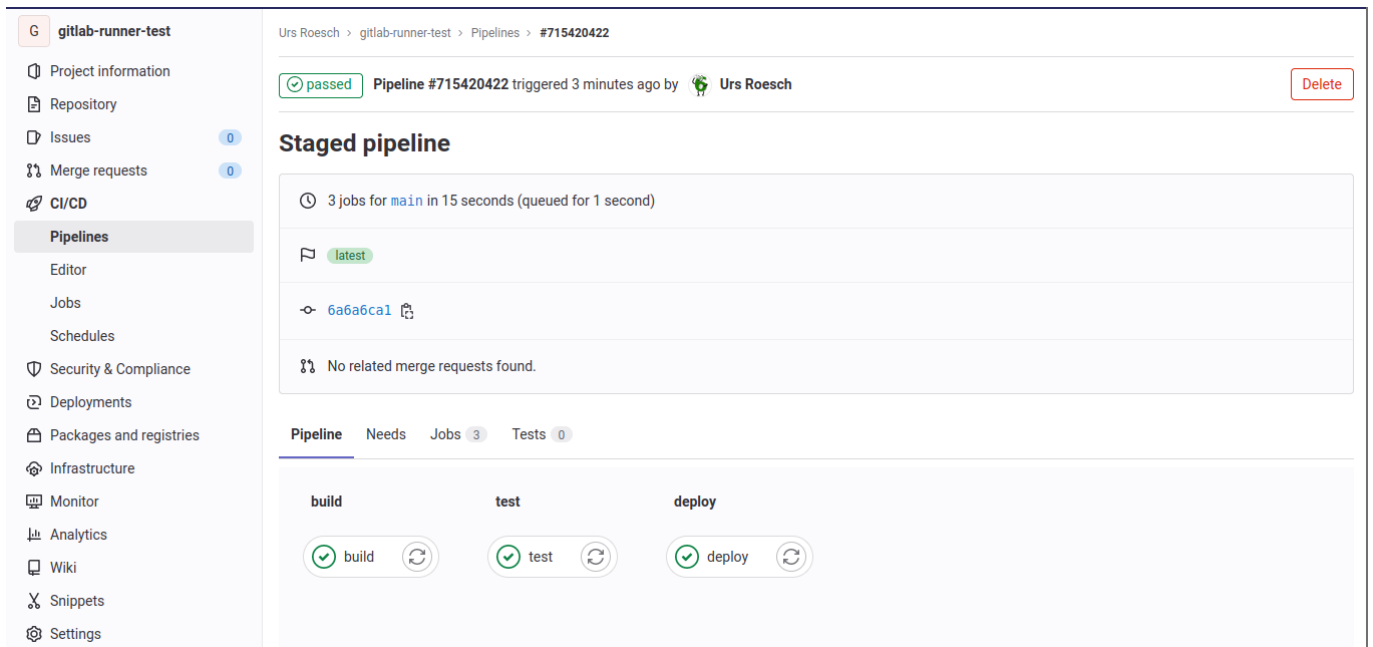
Navigate to the project CI/CD page.



The screenshot shows the GitLab CI/CD Pipelines page for the project 'gitlab-runner-test'. The left sidebar contains navigation options: Project information, Repository, Issues (0), Merge requests (0), CI/CD, Pipelines (selected), Editor, Jobs, Schedules, Security & Compliance, Deployments, Packages and registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings. The main content area shows a list of pipelines with columns for Status, Pipeline, Triggerer, and Stages. Two pipelines are listed: 'Staged pipeline' (passed, 00:00:15, 1 minute ago) and 'First pipeline' (passed, 00:00:05, 4 hours ago). The 'Staged pipeline' has a pink circle with the number '1' over its stages column, indicating a detailed view.

1 Each stage is represented by a circle with the status.

Clicking on the [passed] button shows the detailed view of the staged pipeline.



The screenshot shows the detailed view of the 'Staged pipeline' (Pipeline #715420422) triggered 3 minutes ago by Urs Roesch. The page includes a 'Delete' button and a 'passed' status indicator. The pipeline details show '3 jobs for main in 15 seconds (queued for 1 second)', the 'latest' version, and the commit '6a6a6ca1'. Below the details, the pipeline stages are listed: 'build', 'test', and 'deploy', each with a 'passed' status and a refresh icon.

Navigating to the [Jobs] tab list each job for further inspection.

- G **gitlab-runner-test**
- Project information
- Repository
- Issues 0
- Merge requests 0
- CI/CD
- Pipelines**
- Editor
- Jobs
- Schedules
- Security & Compliance
- Deployments
- Packages and registries
- Infrastructure
- Monitor
- Analytics
- Wiki
- Snippets
- Settings

passed Pipeline #715420422 triggered 3 minutes ago by Urs Roesch

Delete

## Staged pipeline

3 jobs for `main` in 15 seconds (queued for 1 second)

latest

`6a6a6ca1`

No related merge requests found.

Pipeline Needs **Jobs 3** Tests 0

| Status                                                                                 | Job                                                                                                                                           | Stage  | Name   | Duration                      | Coverage |
|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|--------|--------|-------------------------------|----------|
| <span style="border: 1px solid green; border-radius: 3px; padding: 2px;">passed</span> | #3435418996<br>🔗 main ⇨ 6a6a6ca1<br><span style="background-color: #2196F3; color: white; padding: 1px 2px; border-radius: 3px;">shell</span> | deploy | deploy | 🕒 00:00:05<br>🕒 4 minutes ago |          |
| <span style="border: 1px solid green; border-radius: 3px; padding: 2px;">passed</span> | #3435418993<br>🔗 main ⇨ 6a6a6ca1<br><span style="background-color: #2196F3; color: white; padding: 1px 2px; border-radius: 3px;">shell</span> | test   | test   | 🕒 00:00:05<br>🕒 4 minutes ago |          |
| <span style="border: 1px solid green; border-radius: 3px; padding: 2px;">passed</span> | #3435418990<br>🔗 main ⇨ 6a6a6ca1<br><span style="background-color: #2196F3; color: white; padding: 1px 2px; border-radius: 3px;">shell</span> | build  | build  | 🕒 00:00:05<br>🕒 4 minutes ago |          |

## Module 7 - Submodules

Submodules is a way to keep your repositories DRY (Don't Repeat Yourself). Instead of copying code from another repository fully to our current workspace. The option of linking a repository is both lightweight and scalable. The reason to use submodules is manifold. It can be to include common code within the repository or to create a repository collecting independent projects. A good use case is Puppet where each module is compartmentalized in its own git repository. The same can be said for other configuration management software like Ansible.

### Goals

Learn how to include and update a git submodule from another GitLab project in the already existing GitLab project.

Most of the action is taking place on the command line with a few peeks at the result in the GitLab frontend.

- Add a submodule to an existing repository.
- Inspect the **.gitmodules** file.
- Push changes to the repositories.
- Clone a repository with submodules.

### Add a submodule to repository

Create a new project **submodule** under the GitLab frontend. It will be used as the submodule repository for the following exercises.



### Project name

submodule 1

### Project URL

http://gitlab.local:8480/jdev/

### Project slug




submodule

Want to house several dependent projects under the same namespace? [Create a group](#).

### Project description (optional)

Description format

### Visibility Level ?

-  Private  
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.
-  Internal  
The project can be accessed by any logged in user.
-  Public 2  
The project can be accessed without any authentication.

### Initialize repository with a README 3

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

- 1 Name the repository **submodule**.
- 2 Make the project publicly accessible.
- 3 Create an empty **README.md** file.

## git submodule add

With the **submodule** command there are subcommands that define the action to be executed. For adding an existing repository as submodule unsurprisingly **add** is used to do so.

On the command line change into repository **gitlab-project** and issue.

```
$ cd gitlab-project 1
$ git submodule add ../../jdev/submodule 2
Cloning into '/home/jdev/var/tmp/gitlab-project/submodule'...
X11 forwarding request failed on channel 0
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

- 1 Change into the **gitlab-project** repository

- Issue to the **submodule add** command with a relative path to the freshly created **submodule** project. Git will automatically fetch the source from the GitLab server. Alternatively the full URL can be used. But when running CI/CD jobs with submodules this is the recommended way.

### Inspecting the repository's status

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
 (use "git restore --staged <file>..." to unstage)
 new file: .gitmodules ①
 new file: submodule ②
```

- A new file called **.gitmodules** was created.
- The added submodule is also ready to be committed.

### Inspecting the **.gitmodules** file

```
$ cat .gitmodules
[submodule "submodule"] ①
 path = submodule ②
 url = ../../jdev/submodule ③
```

- The file is in INI format the same as the config file under the **.git** directory. Here with the section **submodule "submodule"**
- The **path** is local path within the repository to the submodule.
- The **url** is verbatim what was provided on the command line.

### Inspecting the content of the submodule

```
$ cd submodule/
$ ls
README.md ①
$ git log ②
commit cebacd281a26b1f0503522eb4344dce1226b9383 (HEAD -> master, origin/master,
origin/HEAD)
Author: Joe Developer <joe.developer@gitlab.local>
Date: Mon Dec 7 22:52:04 2020 +0000

 Initial commit
```

- The **README.md** created in the GitLab frontend is present.
- Once inside a submodule git commands only show the history and commits of the submodule!

## Push with submodules

### git push

The command for pushing changes in the parent repository is the same as with a simple repository without any submodule attached

The changes made by the **submodule add** command must be committed first then pushed.

### Commit changes

```
$ git commit --all --message "Add submodule"
[master eb326f0] Add submodule
 2 files changed, 4 insertions(+)
 create mode 100644 .gitmodules
 create mode 160000 submodule
```

### Push from master to branch add-submodule

```
$ git push origin master:add-submodule ❶
X11 forwarding request failed on channel 0
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 363 bytes | 363.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for add-submodule, visit: ❷
remote: http://gitlab.local:8480/jdev/gitlab-project/-
/merge_requests/new?merge_request%5Bsource_branch%5D=add-submodule
remote:
To ssh://gitlab.local:8422/jdev/gitlab-project.git
 * [new branch] master -> add-submodule ❸
```

- ❶ As the commit was done in master put direct pushes to master are restricted. The push is creating a new branch called **add-submodule** in the remote repository.
- ❷ To merge the commit navigate to the URL shown and create a merge request.
- ❸ Git lets us know it created a new branch called **add-submodule**.

## Modify file content

### Modify both README files

```
$ echo '## Now with submodule' >> README.md ❶
$ echo "## I'm the submodule" >> submodule/README.md ❷
$ git status --short
 M README.md
 m submodule ❸
```

- 1 Add a new headline to the parent's README.md.
- 2 Add a new headline to the submodule's README.md.
- 3 Status only reports a change to the submodule was made but does not provide any details what exactly changed.

### *Commit and push the submodule first*

```
$ cd submodule/ ①
$ git commit -a -m "Add new content"
[master bec63ae] Add new content
 1 file changed, 1 insertion(+)

$ git push origin master:submodule ②
Entering 'submodule'
X11 forwarding request failed on channel 0
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 268 bytes | 268.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for submodule, visit:
remote: http://gitlab.local:8480/jdev/submodule/-
/merge_requests/new?merge_request%5Bsource_branch%5D=submodule
remote:
To ssh://gitlab.local:8422/jdev/submodule
 * [new branch] master -> submodule
```

- 1 Change into submodule and commit the change.
- 2 Push to repo **submodule** with branch **submodule**.

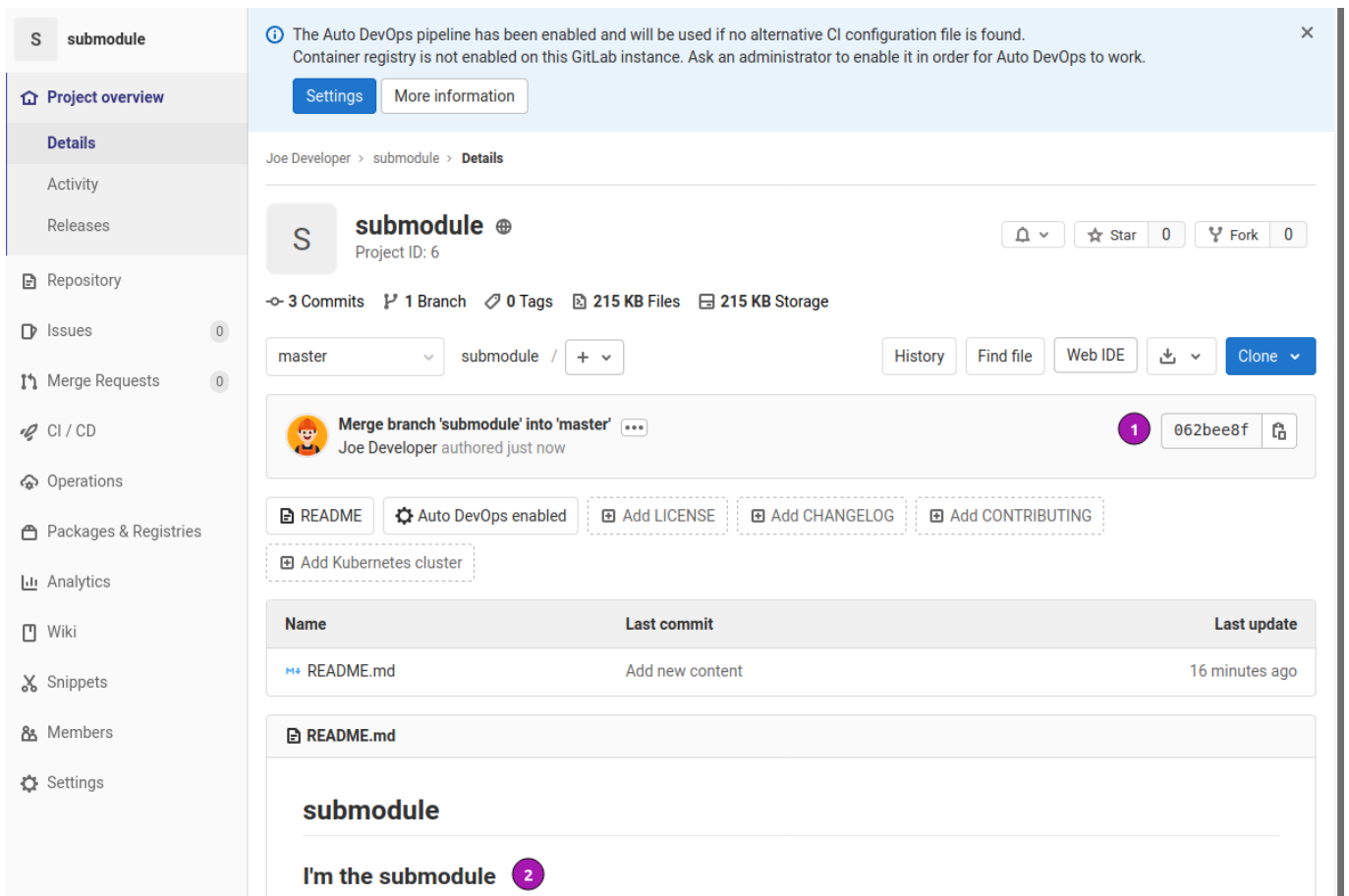


Figure 42. Repository **submodule** after merge

- ① New revision after merge.
- ② Content has been updated.

### Commit and push the submodule first

```

$ cd .. ①
$ git commit -a -m "Add submodule header"
[master 1a44056] Add submodule header
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git push origin master:submodule ②
X11 forwarding request failed on channel 0
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 254 bytes | 254.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for submodule, visit:
remote: http://gitlab.local:8480/jdev/gitlab-project/-
remote: /merge_requests/new?merge_request%5Bsource_branch%5D=submodule
remote:
To ssh://gitlab.local:8422/jdev/gitlab-project.git
* [new branch] master -> submodule

```

- ① Change back into superproject and commit the change.

## 2 Push to repo `gitlab-project` with branch `submodule`.

The screenshot displays the GitLab interface for a repository named 'gitlab-project'. The left sidebar contains navigation options like 'Project overview', 'Details', 'Activity', 'Releases', 'Repository', 'Issues', 'Merge Requests', 'CI / CD', 'Operations', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', 'Members', and 'Settings'. The main content area shows the repository details, including a merge commit for 'submodule' into 'master' by Joe Developer. Below the merge commit, there are buttons for 'README', 'Auto DevOps enabled', 'No license. All rights reserved', and 'Add Kubernetes cluster'. A table lists the commit history with columns for 'Name', 'Last commit', and 'Last update'. The README content is also visible, showing the project name 'gitlab-project' and a list of contributors: Joe Developer (Maintainer) and Mary DevOps.

| Name                 | Last commit          | Last update   |
|----------------------|----------------------|---------------|
| submodule @ bec63ae4 | Add submodule header | 1 minute ago  |
| .gitmodules          | Add submodule        | 1 hour ago    |
| README.md            | Update README.md     | 8 minutes ago |

Figure 43. Repository `gitlab-project` after merge

### 1 New revision of `submodule` after merge.

## Cloning with submodules

### git clone

There is the `--recursive` flag for clone which recursively fetches the superproject as well as all submodules.

## Recursive clone

```
$ git clone --recursive \
 http://gitlab.local:8480/jdev/gitlab-project.git \
 gitlab-project-with-submodules ❶
Cloning into 'gitlab-project-with-submodules'... ❷
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 23 (delta 10), reused 17 (delta 7), pack-reused 0
Unpacking objects: 100% (23/23), 2.46 KiB | 1.23 MiB/s, done.

Submodule 'submodule' (http://gitlab.local:8480/jdev/submodule) registered for
path 'submodule'
Cloning into '/home/uroesch/var/tmp/gitlab-project-with-submodules/submodule'...
❸
warning: redirecting to http://gitlab.local:8480/jdev/submodule.git/
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Submodule path 'submodule': checked out 'bec63ae43dd9ad434af8dcb087900424616de102'
```

- ❶ Clone into dedicated new directory **gitlab-project-with-submodules**
- ❷ Cloning the superproject repository **gitlab-project** first.
- ❸ Cloning the submodule **submode** second.

When not aware that the repository to be cloned contains submodules it can happen that the **--recursive** option is not present during clone.

In such a case the submodules need to be updated from within the superproject.

## Non-recursive clone

```
$ git clone http://gitlab.local:8480/jdev/gitlab-project.git \
 gitlab-project-non-recursive ❶
Cloning into 'gitlab-project-non-recursive'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 23 (delta 10), reused 17 (delta 7), pack-reused 0
Unpacking objects: 100% (23/23), 2.46 KiB | 841.00 KiB/s, done.
$ cd gitlab-project-non-recursive ❷
$ git submodule update --recursive --init ❸
Cloning into '/home/uroesch/var/tmp/gitlab-project-non-recursive/submodule'...
warning: redirecting to http://gitlab.local:8480/jdev/submodule.git/
Submodule path 'submodule': checked out 'bec63ae43dd9ad434af8dcb087900424616de102'
```

- ❶ Clone into dedicated new directory **gitlab-project-non-recursive**
- ❷ Change into directory **gitlab-project-non-recursive**

③ Cloning the submodules with command **update** and options **--recursive** and **--init**.