Xebia

B.Tech Computer Science and Engineering in DevOps

Source Code Management

GitLab

Release 2.0

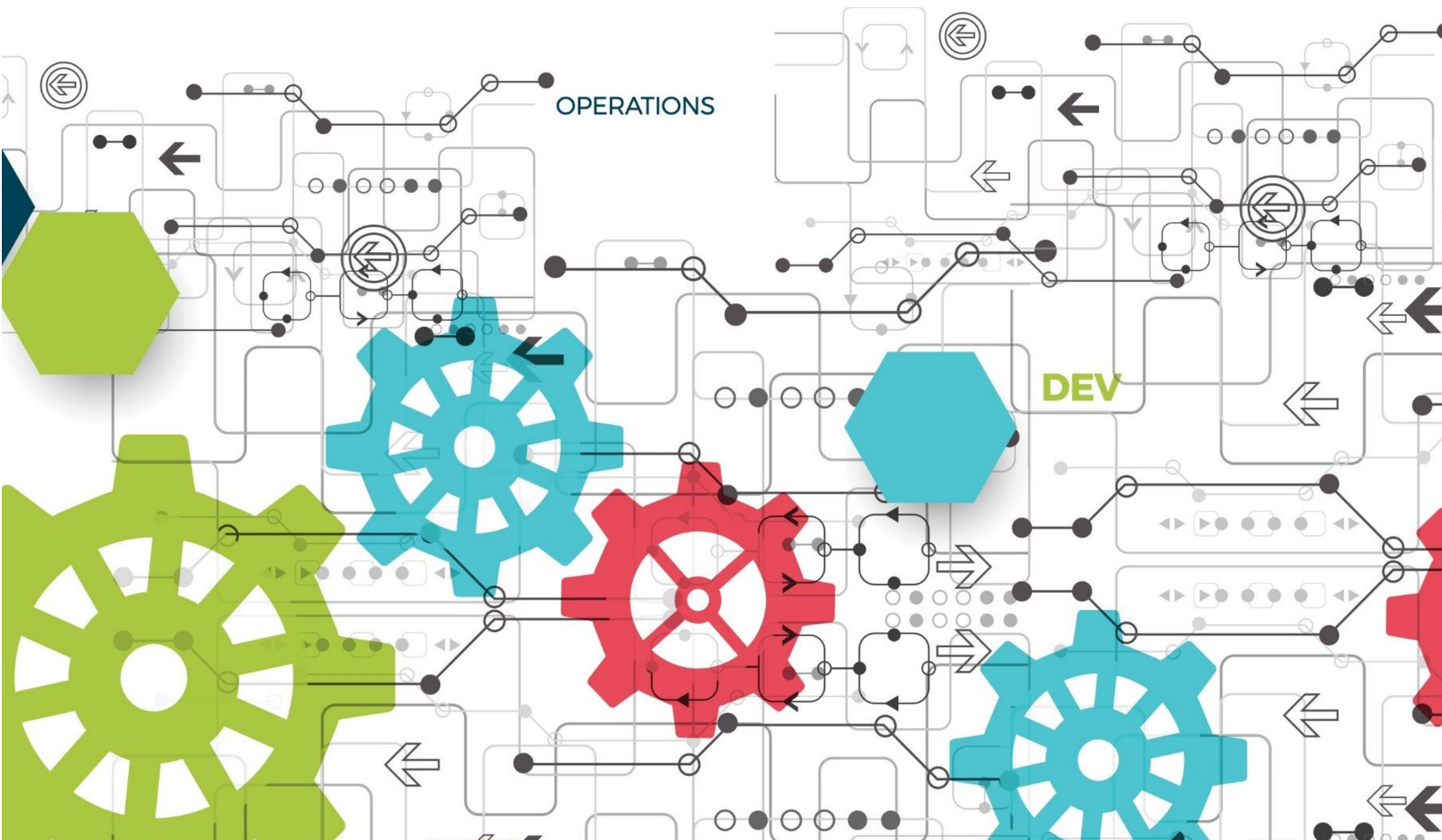


Table of Contents

GitLab

Gitlab is a service that provides remote access to Git repositories. In addition to hosting your code, the services provide additional features designed to help manage the software development lifecycle. These additional features include managing the sharing of code between different people, bug tracking, wiki space and other tools for 'social coding'.

What is Gitlab?

Before we dive into definition for Gitlab, first we need to understand few terminologies. We often come across these terms like Git, Gitlab, GitHub, and Bitbucket. Let's see definiton of all these as below −

Git - It is a source code versioning system that lets you locally track changes and push or pull changes from remote resources.

GitLab, GitHub, and Bitbucket - Are services that provides remote access to Git repositories. In addition to hosting your code, the services provide additional features designed to help manage the software development lifecycle. These additional features include managing the sharing of code between different people, bug tracking, wiki space and other tools for 'social coding'.

- **GitHub** is a publicly available, free service which requires all code (unless you have a paid account) be made open. Anyone can see code you push to GitHub and offer suggestions for improvement. GitHub currently hosts the source code for tens of thousands of open source projects.
- **GitLab** is a github like service that organizations can use to provide internal management of git repositories. It is a self hosted Git-repository management system that keeps the user code private and can easily deploy the changes of the code.

History

GitLab was found by *Dmitriy Zaporozhets* and *Valery Sizov* in October 2011. It was distributed under MIT license and the stable version of GitLab is 10.4 released in January 22, 2018.

Why to use GitLab?

GitLab is great way to manage git repositories on centralized server. GitLab gives you complete control over your repositories or projects and allows you to decide whether they are public or private for free.

Features

- GitLab hosts your (private) software projects for free.
- GitLab is a platform for managing Git repositories.
- GitLab offers free public and private repositories, issue-tracking and wikis.

- GitLab is a user friendly web interface layer on top of Git, which increases the speed of working with Git.
- GitLab provides its own *Continuous Integration* (CI) system for managing the projects and provides user interface along with other features of GitLab.

Advantages

- GitLab provides *GitLab Community Edition* version for users to locate, on which servers their code is present.
- GitLab provides unlimited number of private and public repositories for free.
- The *Snippet* section can share small amount of code from a project, instead of sharing whole project.

Disadvantages

- While pushing and pulling repositories, it is not as fast as GitHub.
- GitLab interface will take time while switching from one to another page.

Lab 1: Installation Guide: Windows, Ubuntu

You can install the GitLab runner on different operating systems, by installing *Git* versioning system and creating user account in the GitLab site.

Git is a version control system used for −

- Handling the source code history of projects
- Tracking changes made to files
- Handling small and large projects with speed and efficiency
- To collaborate with other developers on different projects

GitLab is a Git-based platform provides remote access to Git repositories and helpful for software development cycle by creating private and public repositories for managing the code.

GitLab supports different types of operating systems such as Windows, Ubuntu, Debian, CentOS, open SUSE and Raspberry Pi 2. In this chapter, we will discuss about how to install GitLab on Windows and Ubuntu operating systems −

Installation of GitLab on Windows:

Step 1 − First create a folder called 'GitLab-Runner' in your system. For instance, you can create in C drive as C:\GitLab-Runner.

Step 2 − Now download the binary for x86 or amd64 and copy it in the folder created by you. Rename the downloaded binary to *gitlab-runner.exe*.

Step 3 − Open the command prompt and navigate to your created folder. Now type the below command and press enter.

```
C:\GitLab-Runner>gitlab-runner.exe register
```

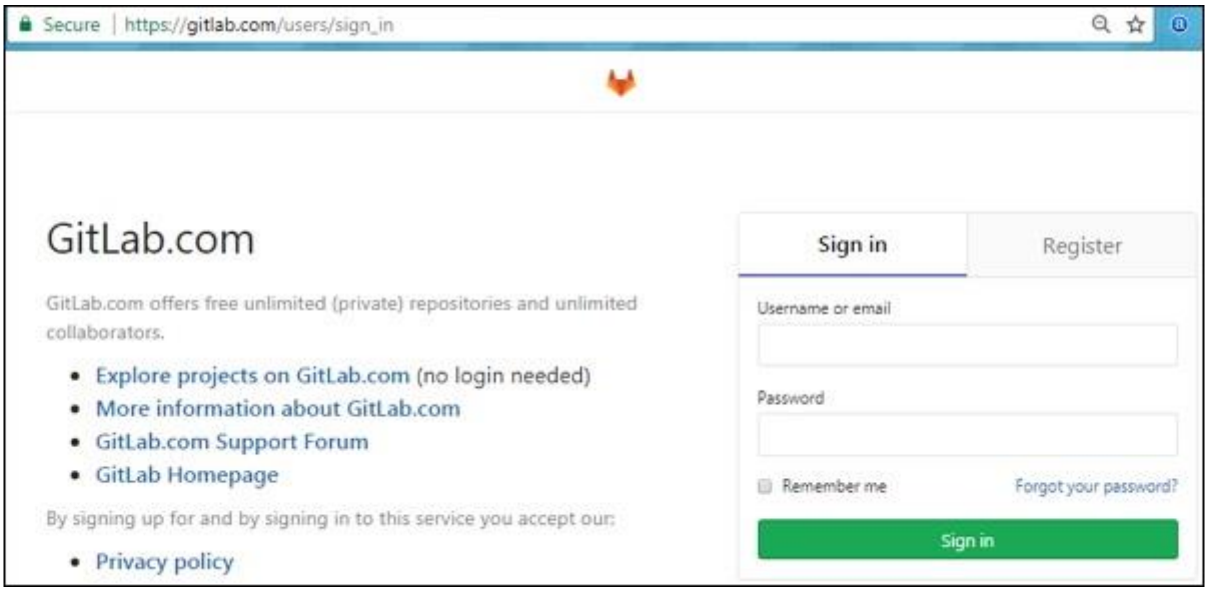
Step 4 − After running the above command, it will ask to enter the gitlab-ci coordinator URL.

```
Please enter the gitlab-ci coordinator URL (e.g.
https://gitlab.com/):
https://gitlab.com
```

Step 5 − Enter the gitlab-ci token for the runner.

```
Please enter the gitlab-ci token for this runner:
xxxxx
```

- To get the token, login to your GitLab account −


Secure | https://gitlab.com/users/sign_in

GitLab.com

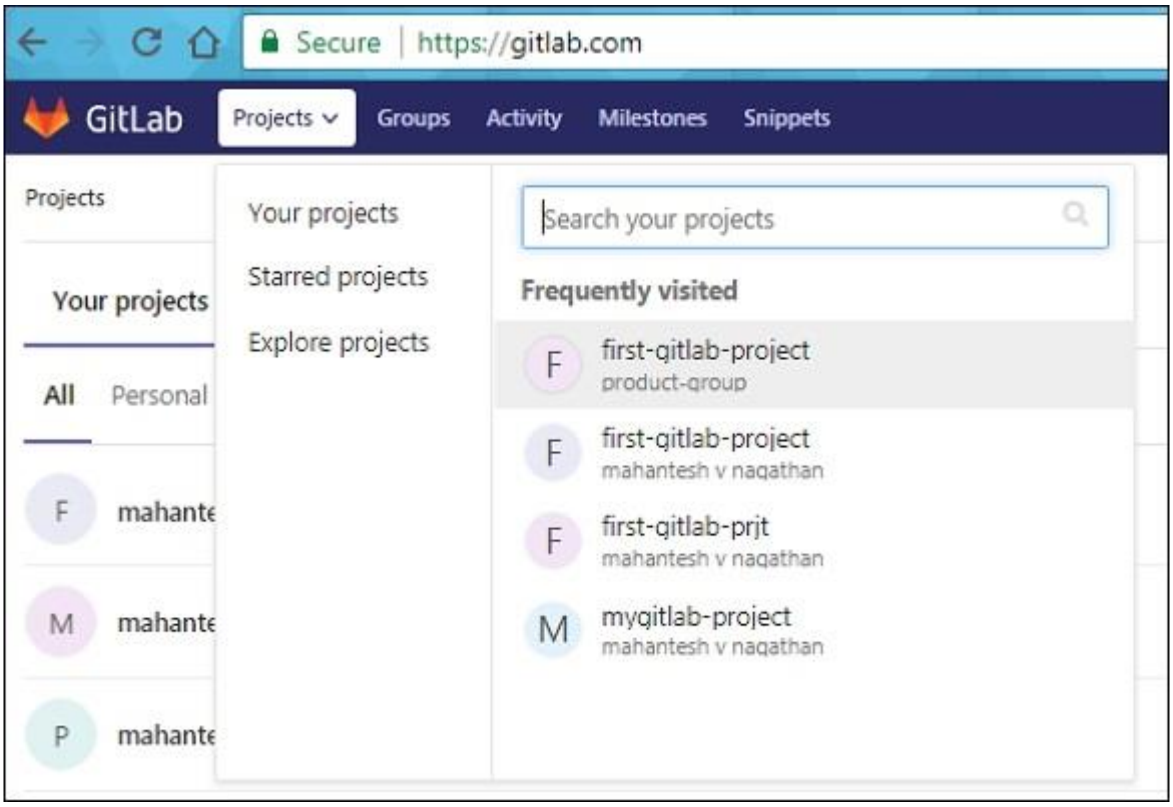
GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- Explore projects on GitLab.com (no login needed)
- More information about GitLab.com
- GitLab.com Support Forum
- GitLab Homepage

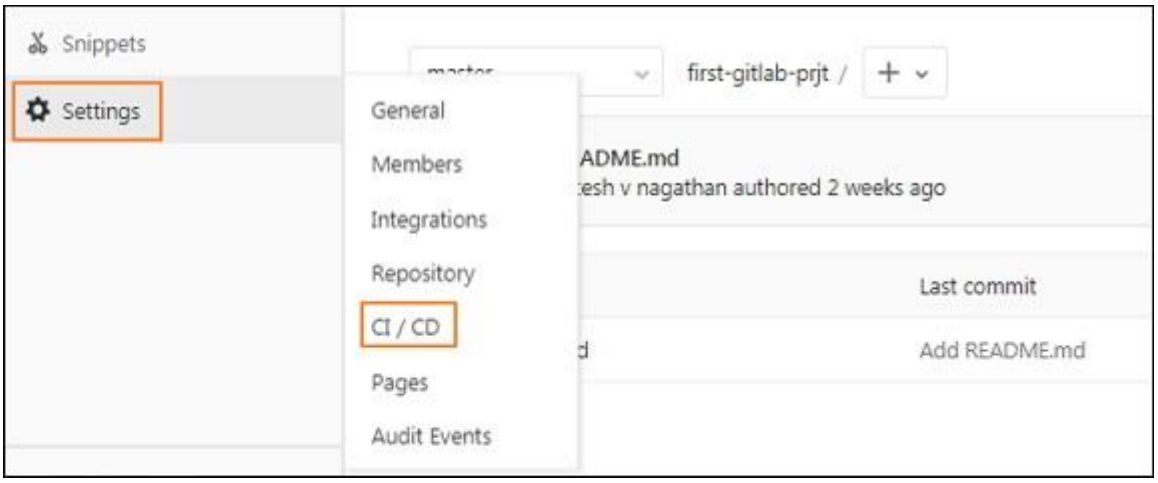
By signing up for and by signing in to this service you accept our:

- Privacy policy

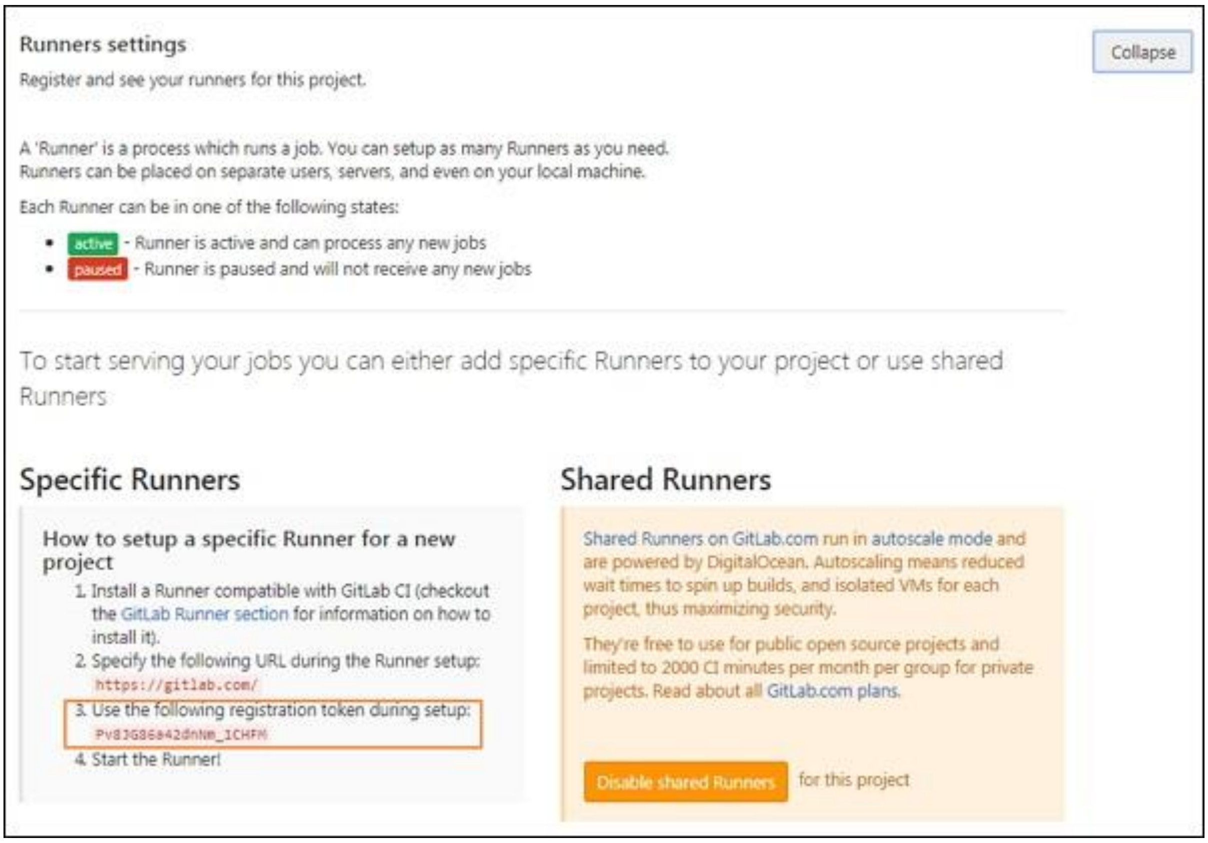
- Now go to your project −



- Click on the *CI/CD* option under *Settings* tab and expand the *Runners Settings* option.



- Under *Runners Settings* section, you will get the token as shown in the image below −


Runners settings

Register and see your runners for this project.

A 'Runner' is a process which runs a job. You can setup as many Runners as you need.
Runners can be placed on separate users, servers, and even on your local machine.

Each Runner can be in one of the following states:

- active - Runner is active and can process any new jobs
- paused - Runner is paused and will not receive any new jobs

To start serving your jobs you can either add specific Runners to your project or use shared Runners

Specific Runners

How to setup a specific Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `https://gitlab.com/`
3. Use the following registration token during setup: `Pv8JG86e42dnNm_1CHFM`
4. Start the Runner!

Shared Runners

Shared Runners on GitLab.com run in autoscale mode and are powered by DigitalOcean. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 2000 CI minutes per month per group for private projects. Read about all GitLab.com plans.

for this project

Step 6 − Enter the gitlab-ci description for the runner.

```
Please enter the gitlab-ci description for this runner:
[Admin-PC]: Hello GitLab Runner
```

Step 7 − It will ask to enter the gitlab-ci tags for the runner.

```
Please enter the gitlab-ci tags for this runner (comma
separated):
tag1, tag2
```

You can change these tags in the GitLab's user interface later.

Step 8 − You can lock the Runner to current project by setting it to true value.

```
Whether to lock the Runner to current project [true/false]:
[true]: true
```

After completing above steps, you will get the successful message as 'Registering runner... succeeded'.

Step 9 − Now enter the Runner executor for building the project.

```
Please enter the executor: parallels, shell, docker+machine,
kubernetes, docker-
ssh+machine, docker, docker-ssh, ssh, virtualbox:
docker
```

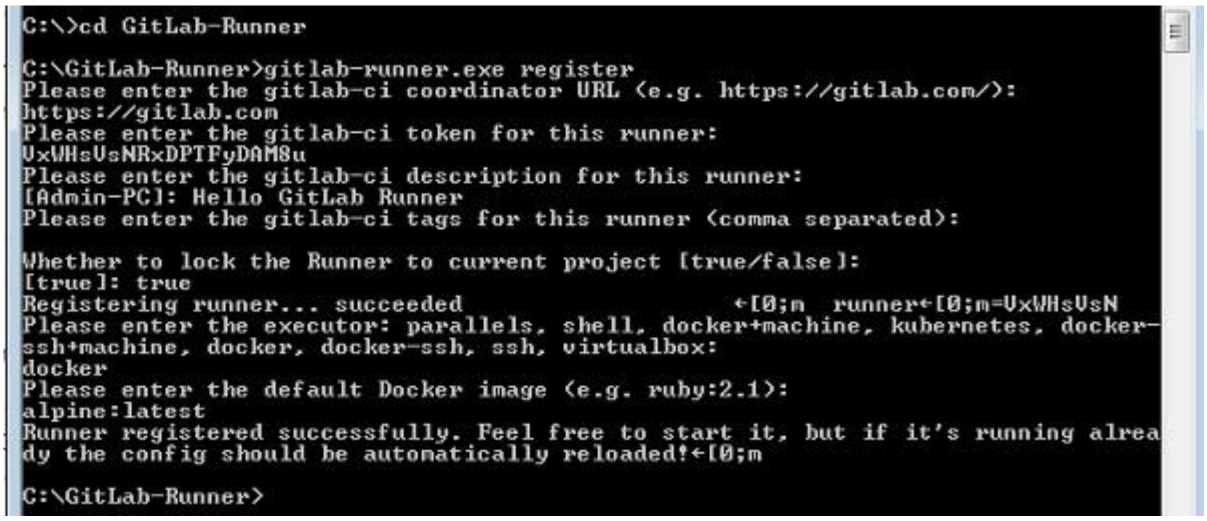
We have used the selector as 'docker' which creates build environment and manages the dependencies easily for developing the project.

Step 10 − Next it will ask for default image to be set for docker selector.

```
Please enter the default Docker image (e.g. ruby:2.1):
```

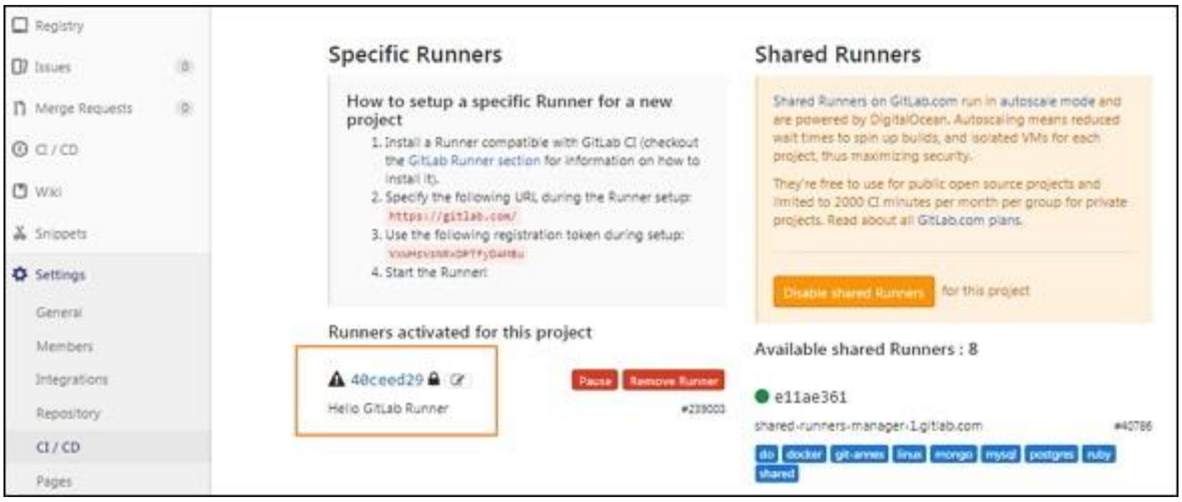
```
alpine:latest
```

Step 11 − After completing the above steps, it will display the message as 'Runner registered successfully'. The below image will describe the working flow of above commands −



```
C:\>cd GitLab-Runner

C:\GitLab-Runner>gitlab-runner.exe register
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com
Please enter the gitlab-ci token for this runner:
UxWHsUsNRxDPTFyDAM8u
Please enter the gitlab-ci description for this runner:
[Admin-PC]: Hello GitLab Runner
Please enter the gitlab-ci tags for this runner (comma separated):

Whether to lock the Runner to current project [true/false]:
[true]: true
Registering runner... succeeded                     ←[0;m  runner←[0;m=UxWHsUsN
Please enter the executor: parallels, shell, docker+machine, kubernetes, docker-
ssh+machine, docker, docker-ssh, ssh, virtualbox:
docker
Please enter the default Docker image (e.g. ruby:2.1):
alpine:latest
Runner registered successfully. Feel free to start it, but if it's running alrea
dy the config should be automatically reloaded!←[0;m

C:\GitLab-Runner>
```

Step 12 − Now go to your project, click on the *CI/CD* option under *Settings* section and you will see the activated Runners for the project.



Specific Runners

How to setup a specific Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `https://gitlab.com/`
3. Use the following registration token during setup: `VxWHsVsNRxDPTFyDAM8u`
4. Start the Runner!

Runners activated for this project

40ceed29
Hello GitLab Runner
#239003

Shared Runners

Shared Runners on GitLab.com run in autoscale mode and are powered by DigitalOcean. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 2000 CI minutes per month per group for private projects. Read about all GitLab.com plans.

Disable shared Runners for this project

Available shared Runners : 8

e11ae361
shared-runners-manager-1.gitlab.com #40786
do docker git-annex linux mongo mysql postgres ruby shared

You can see the GitLab Runner configuration in the *config.toml* file under the *GitLab-Runner* folder as shown below −

```
concurrent = 1
check_interval = 0
[[runners]]
  name = "Hello GitLab Runner"
  url = "https://gitlab.com"
  token = "40ceed29eec231fa9e306629cae4d7"
  executor = "docker"
  [runners.docker]
      tls_verify = false
      image = "alpine:latest"
      privileged = false
      disable_cache = false
      volumes = ["/cache"]
      shm_size = 0
  [runners.cache]
```

Installation of GitLab on Ubuntu

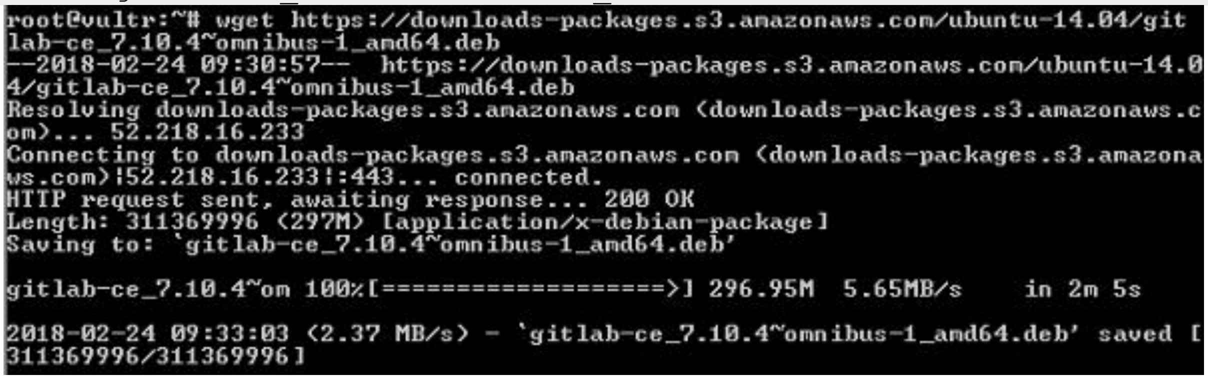
The GitLab can be installed on Ubuntu system by using *Omnibus* package which provides different services to run GitLab. The Omnibus package provides necessary components of GitLab, establishes the configurations and project metadata which can be used in user's system.

The following steps describe installation of GitLab on Ubuntu −

Step 1 − First, login to your GitLab server using SSH (Secure Shell).

Step 2 − Next, download the Omnibus package −

```
wget https://downloads-packages.s3.amazonaws.com/ubuntu-
14.04/gitlab-ce_7.10.4~omnibus-1_amd64.deb
```

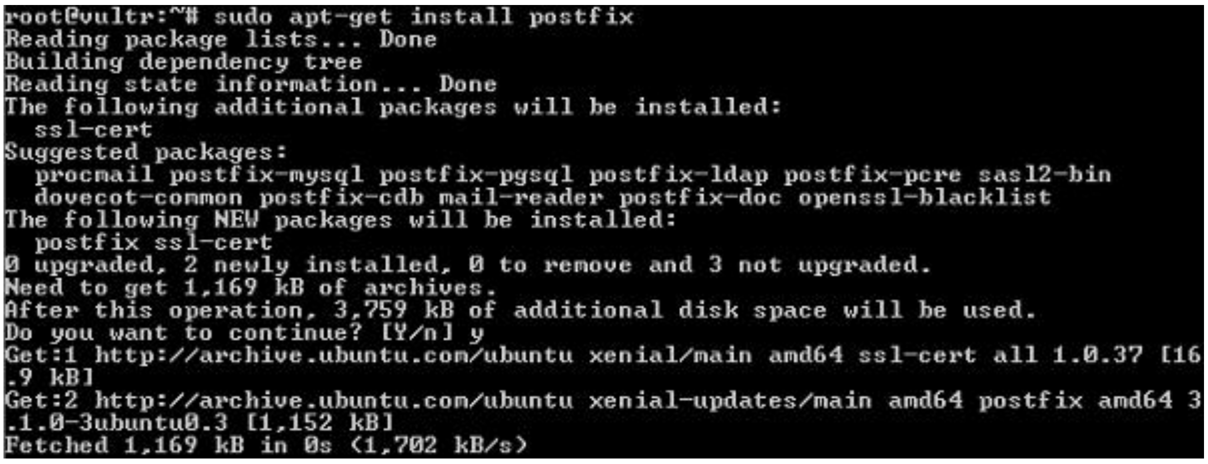


```
root@vultr:~# wget https://downloads-packages.s3.amazonaws.com/ubuntu-14.04/git
lab-ce_7.10.4~omnibus-1_amd64.deb
--2018-02-24 09:30:57--  https://downloads-packages.s3.amazonaws.com/ubuntu-14.0
4/gitlab-ce_7.10.4~omnibus-1_amd64.deb
Resolving downloads-packages.s3.amazonaws.com (downloads-packages.s3.amazonaws.c
om)... 52.218.16.233
Connecting to downloads-packages.s3.amazonaws.com (downloads-packages.s3.amazona
ws.com)|52.218.16.233|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 311369996 (297M) [application/x-debian-package]
Saving to: `gitlab-ce_7.10.4~omnibus-1_amd64.deb'

gitlab-ce_7.10.4~om 100%[===================>] 296.95M  5.65MB/s    in 2m 5s

2018-02-24 09:33:03 (2.37 MB/s) - `gitlab-ce_7.10.4~omnibus-1_amd64.deb' saved [
311369996/311369996]
```

Step 3 − Install the postfix −

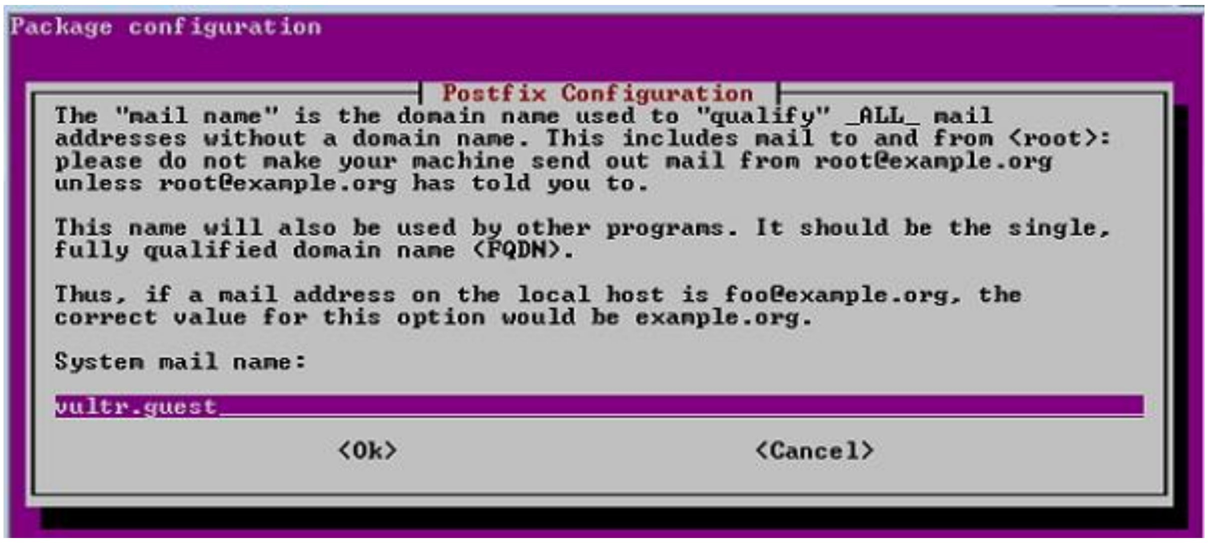
```
sudo apt-get install postfix
```

Postfix is a open source mail transfer agent used to deliver the email notifications.



```
root@vultr:~# sudo apt-get install postfix
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ssl-cert
Suggested packages:
  procmail postfix-mysql postfix-pgsql postfix-ldap postfix-pcre sasl2-bin
  dovecot-common postfix-cdb mail-reader postfix-doc openssl-blacklist
The following NEW packages will be installed:
  postfix ssl-cert
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 1,169 kB of archives.
After this operation, 3,759 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu xenial/main amd64 ssl-cert all 1.0.37 [16
.9 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 postfix amd64 3
.1.0-3ubuntu0.3 [1,152 kB]
Fetched 1,169 kB in 0s (1,702 kB/s)
```

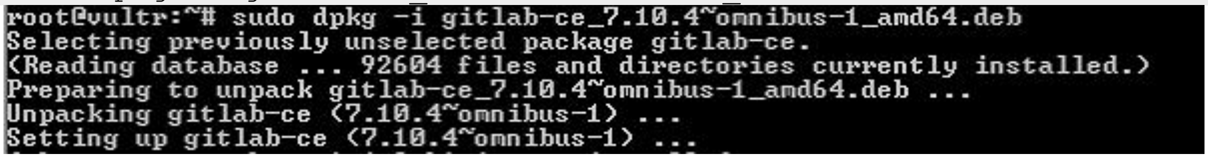
Step 4 − While installing Postfix, it will ask type of installation; then select the *Internet Site* option. Next, it will show Postfix configuration along with the system mail name as shown in the image −



```
Package configuration

                    Postfix Configuration
  The "mail name" is the domain name used to "qualify" _ALL_ mail
  addresses without a domain name. This includes mail to and from <root>:
  please do not make your machine send out mail from root@example.org
  unless root@example.org has told you to.

  This name will also be used by other programs. It should be the single,
  fully qualified domain name (FQDN).

  Thus, if a mail address on the local host is foo@example.org, the
  correct value for this option would be example.org.

  System mail name:

  vultr.guest

                  <Ok>                          <Cancel>
```

Step 5 − Install the dpkg (package manager for debian system) for managing the installed packages −

```
sudo dpkg -i gitlab-ce_7.10.4~omnibus-1_amd64.deb
```



```
root@vultr:~# sudo dpkg -i gitlab-ce_7.10.4~omnibus-1_amd64.deb
Selecting previously unselected package gitlab-ce.
(Reading database ... 92604 files and directories currently installed.)
Preparing to unpack gitlab-ce_7.10.4~omnibus-1_amd64.deb ...
Unpacking gitlab-ce (7.10.4~omnibus-1) ...
Setting up gitlab-ce (7.10.4~omnibus-1) ...
```

Step 6 − To have the changes take effect, you need to reconfigure the GitLab by using the below command −

```
sudo gitlab-ctl reconfigure
```

Step 7 − Check the status of the GitLab services by using below command −

```
sudo gitlab-ctl status
```

If you want to install GitLab from the source, then install some dependencies on the server and need to setup the database by using the PostgreSQL.

LAB 2: Using Git Commands

Description

Git commands are used for sharing and combining the code easily with other developers.

Git Commands

Following are the some basic Git commands can be used to work with Git −

The version of the Git can be checked by using the below command −

```
$ git --version
```

Add Git username and email address to identify the author while committing the information. Set the username by using the command as −

```
$ git config --global user.name "USERNAME"
```

After entering user name, verify the entered user name with the below command −

```
$ git config --global user.name
```

Next, set the email address with the below command −

```
$ git config --global user.email "email_address@example.com"
```

You can verify the entered email address as −

```
$ git config --global user.email
```

Use the below command to check the entered information −

```
$ git config --global --list
```

You can pull the latest changes made to the master branch by using the below command −

```
$ git checkout master
```

You can fetch the latest changes to the working directory with the below command −

```
$ git pull origin NAME-OF-BRANCH -u
```

Here, NAME-OF-BRANCH could be 'master' or any other existing branch.

Create a new branch with the below command −

```
$ git checkout -b branch-name
```

You can switch from one branch to other branch by using the command as −

```
$ git checkout branch-name
```

Check the changes made to your files with the below command −

```
$ git status
```

You will see the changes in red color and add the files to staging as −

```
$ git add file-name
```

Or you can add all the files to staging as −

```
$ git add *
```

Now send your changes to master branch with the below command −

```
$ git push origin branch-name
```

Delete the all changes, except unstaged things by using the below command −

```
$ git checkout .
```

You can delete the all changes along with untracked files by using the command as −

```
$ git clean -f
```

To merge the different branch with the master branch, use the below command −

```
$git checkout branch-name
```

```
$ git merge master
```

You can also merge the master branch with the created branch, by using the below command −

```
$git checkout master
$ git merge branch-name
```

LAB 3: Creating SSH Key

Description

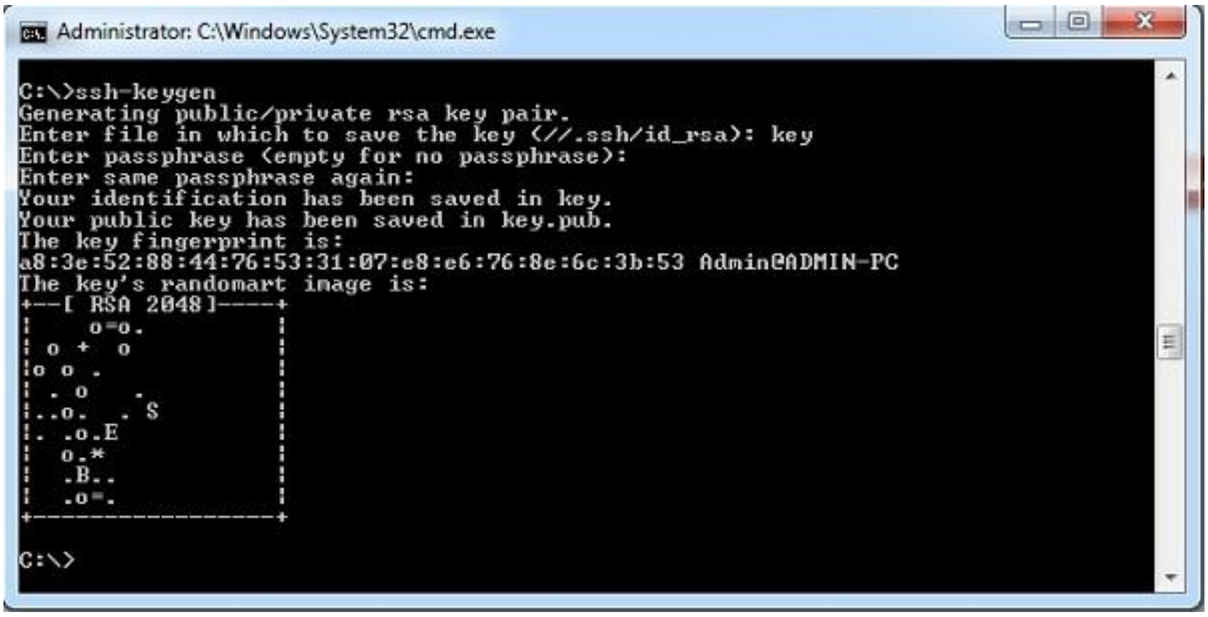
The SSH stands for *Secure Shell* or *Secure Socket Shell* used for managing the networks, operating systems and configurations and also authenticates to the GitLab server without using username and password each time. You can set the SSH keys to provide a reliable connection between the computer and GitLab. Before generating ssh keygen, you need to have Git installed in your system.

Creating SSH Key

Step 1 − To create SSH key, open the command prompt and enter the command as shown below −

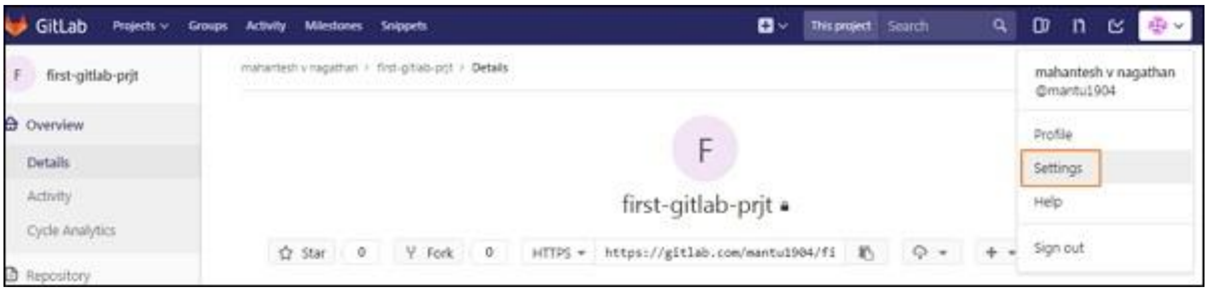
```
C:\-ssh-keygen
```

It will prompt for 'Enter file in which to save the key (//.ssh/id_rsa):', just type file name and press enter. Next a prompt to enter password shows 'Enter passphrase (empty for no passphrase):'. Enter some password and press enter. You will see the generated SSH key as shown in the below image −

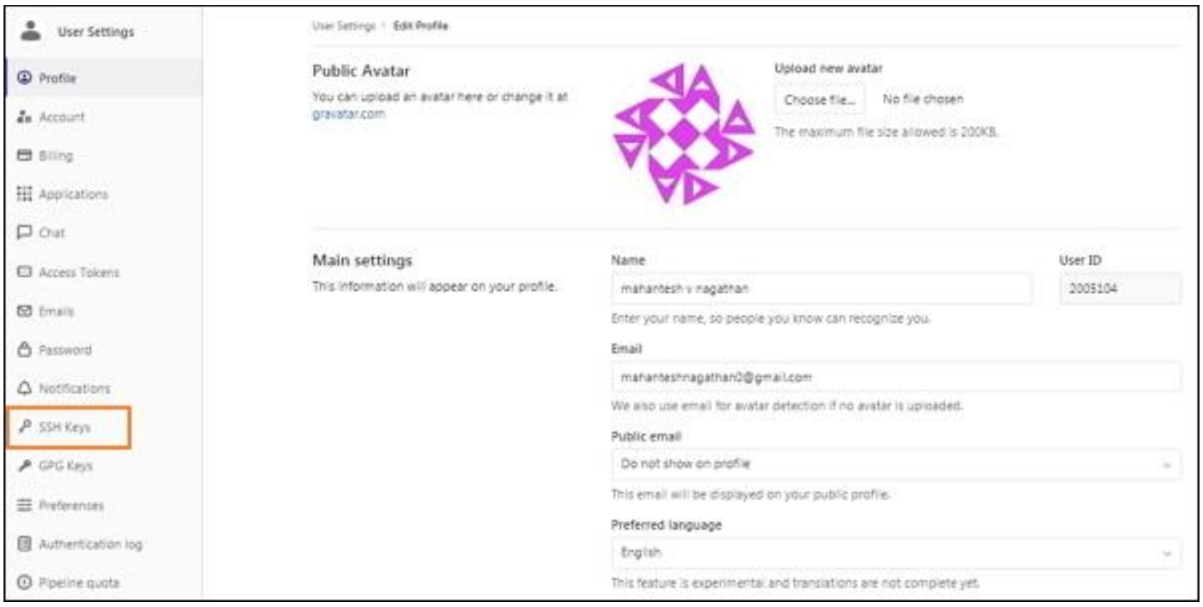


```
C:\>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (//.ssh/id_rsa): key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in key.
Your public key has been saved in key.pub.
The key fingerprint is:
a8:3e:52:88:44:76:53:31:07:e8:e6:76:8e:6c:3b:53 Admin@ADMIN-PC
The key's randomart image is:
+--[ RSA 2048]----+
|     o=o.        |
|  o +  o         |
|o o .            |
| . o     .       |
|..o.   . S       |
|. .o.E           |
|  o.*            |
|  .B..           |
|  .o=.           |
+-----------------+

C:\>
```

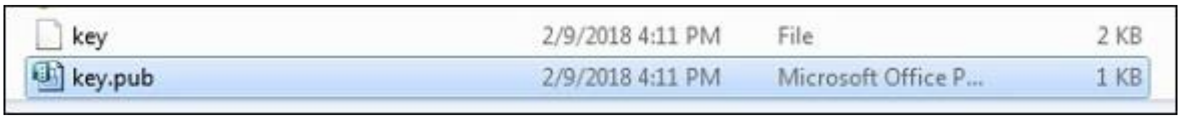
Step 2 − Now login to your GitLab account and click on the *Settings* option.



Step 3 − To create SSH key, click on the SSH keys tab at left side of the menu.

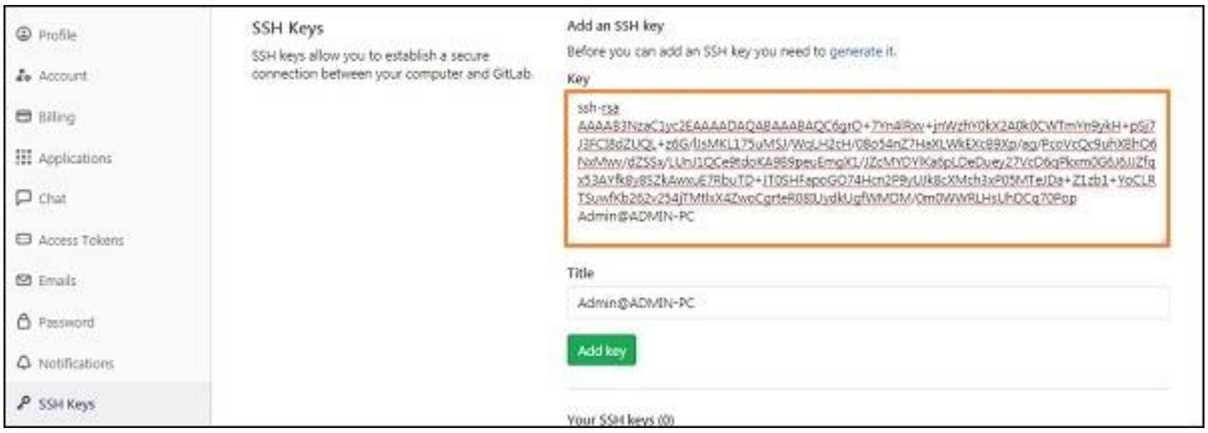

User Settings
Profile
Account
Billing
Applications
Chat
Access Tokens
Emails
Password
Notifications
SSH Keys
GPG Keys
Preferences
Authentication log
Pipeline quota
User Settings › Edit Profile
Public Avatar
You can upload an avatar here or change it at gravatar.com
Upload new avatar
Choose file...
No file chosen
The maximum file size allowed is 200KB.
Main settings
This information will appear on your profile.
Name
mahantesh v nagathan
User ID
2005104
Enter your name, so people you know can recognize you.
Email
mahanteshnagathan0@gmail.com
We also use email for avatar detection if no avatar is uploaded.
Public email
Do not show on profile
This email will be displayed on your public profile.
Preferred language
English
This feature is experimental and translations are not complete yet.

Step 4 − Now go to C drive, you will see the file with *.pub* extension which was generated in the first step.



key	2/9/2018 4:11 PM	File	2 KB
key.pub	2/9/2018 4:11 PM	Microsoft Office P...	1 KB

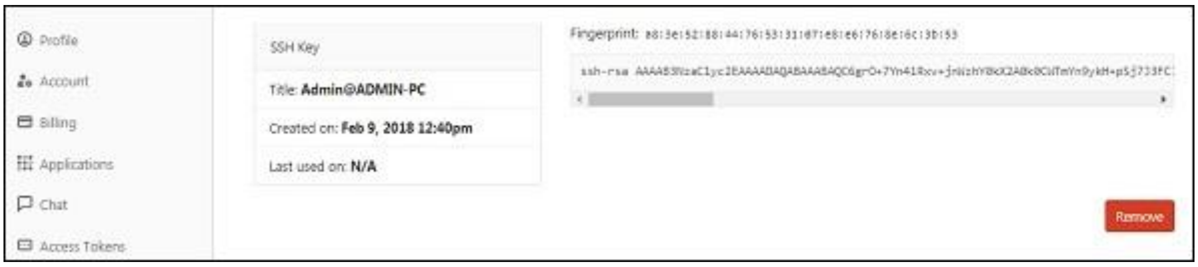
Step 5 − Next open the *key.pub* file, copy the SSH key and paste it in the highlighted *Key* box as shown in the below image −


Profile
Account
Billing
Applications
Chat
Access Tokens
Emails
Password
Notifications
SSH Keys
SSH Keys
SSH keys allow you to establish a secure connection between your computer and GitLab.
Add an SSH key
Before you can add an SSH key you need to generate it.
Key
Title
Add key
Your SSH keys (0)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC6grO+7Yn4lRxy+jnWzhY0kX2A0k0CWTmYn9ykH+pSj7
J3FC38dZUQL+z6G/lIsMKL175uMSJ/WqLH2cH/08o54nZ7HeXLWkEXc89Xp/ag/PcoVcQc9uhX8hC6
NxMwy/dZ5Sa/LUnJ1QCe8tdoKA989peuEmqX1/JZcMYDYlKa6pLDeDuey27VcD6qPkxm0G6J6JJZfq
x53AYfk8y85ZkAwxuE7RbuTD+IT0SHFepoGO74Hcn2P9yUJk8cXMch3xP05MTeJDa+Z1zb1+YoCLR
TSuwfKb262v254jTMtlxX4ZwoCgrteR08UydkUgfWMDM/0m0WWRLHsUhDCq70Pop
Admin@ADMIN-PC
```

Title: Admin@ADMIN-PC

Step 6 − Click on the *Add Key* button, to add SSH key to your GitLab. You will see the fingerprint (it is a short version of SSH key), title and created date as shown in the image below −



Title: **Admin@ADMIN-PC**

Created on: **Feb 9, 2018 12:40pm**

Last used on: **N/A**

Fingerprint: `a8:3e:52:88:44:76:53:31:67:e8:e6:76:8e:6c:3b:53`

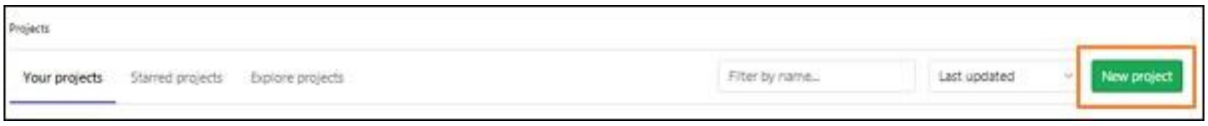
```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6grO+7Yn41Rxv+jnWzhYBkX2ABk0CUTmYn9ykH+p5j733FC
```

LAB 4: How to Create a New Project in GitLab

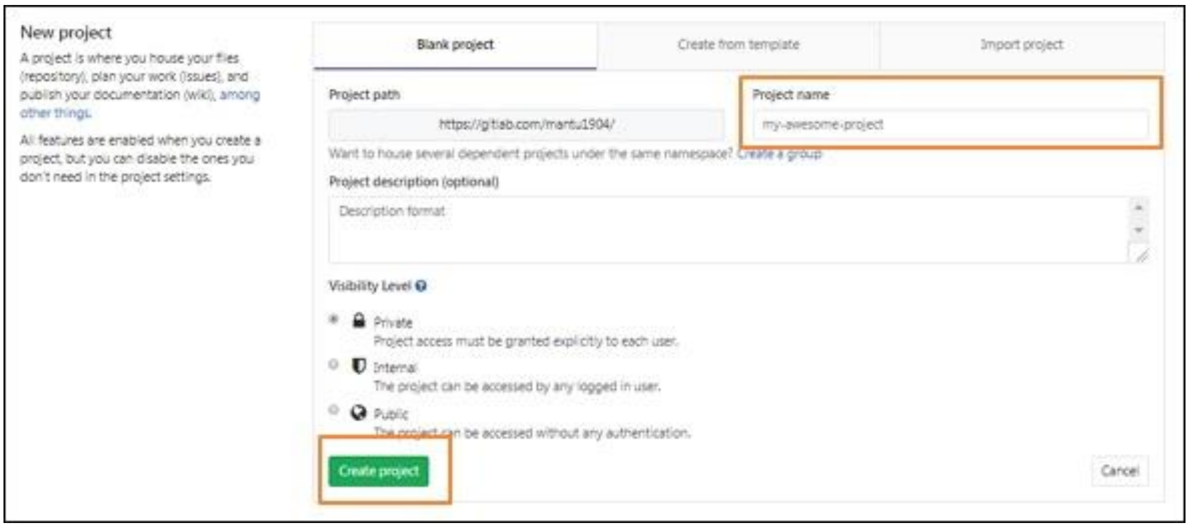
Description: We will discuss about how to create a new project in the GitLab.

Creating New Project

Step 1 − To create new project, login to your GitLab account and click on the *New project* button in the dashboard −



Step 2 − It will open the New project screen as shown below in the image −


New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled when you create a project, but you can disable the ones you don't need in the project settings.

Project path: https://gitlab.com/mantu1904/

Project name: my-awesome-project

Want to house several dependent projects under the same namespace? Create a group

Project description (optional): Description format

Visibility Level

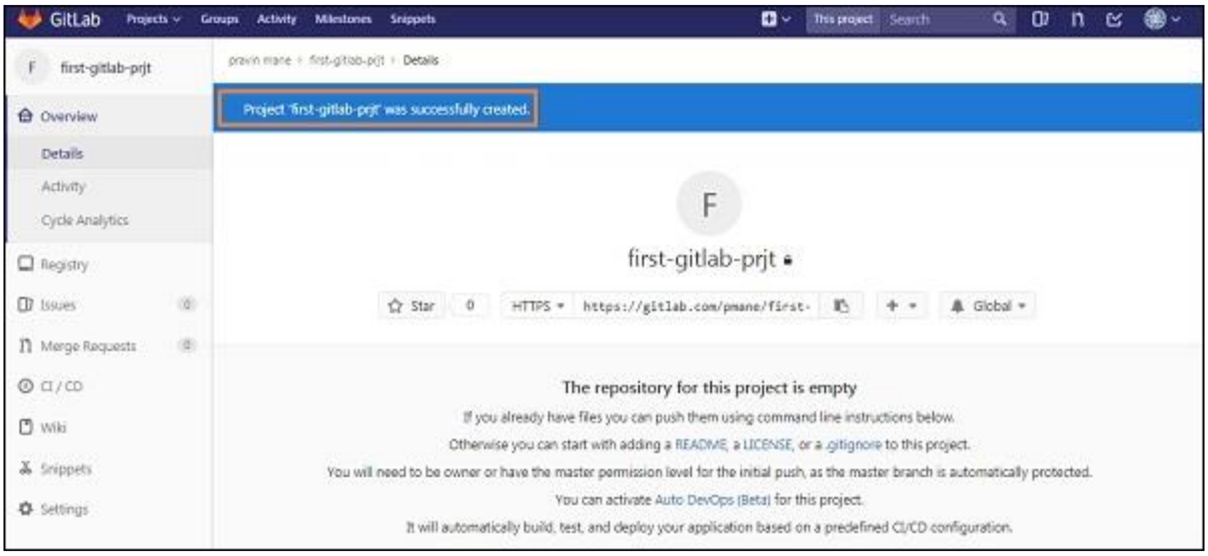
[✓ Private] Project access must be granted explicitly to each user.

Internal The project can be accessed by any logged in user.

Public The project can be accessed without any authentication.

Enter the project name, description for the project, visibility level (accessing the project's visibility in publicly or internally) and click on the *Create project* button.

Step 3 − Next it will create a new project (here given the project name as first-gitlab-prjt) with successful message as shown below −



Project 'first-gitlab-prjt' was successfully created.

first-gitlab-prjt

The repository for this project is empty

If you already have files you can push them using command line instructions below.

Otherwise you can start with adding a README, a LICENSE, or a .gitignore to this project.

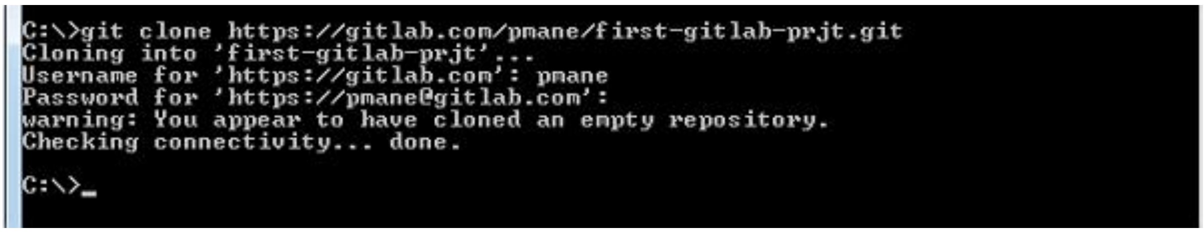
You will need to be owner or have the master permission level for the initial push, as the master branch is automatically protected.

You can activate Auto DevOps (Beta) for this project.

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Push the Repository to Project

Step 4 − You can clone the repository to your local system by using the *git-clone* command −



```
C:\>git clone https://gitlab.com/pmane/first-gitlab-prjt.git
Cloning into 'first-gitlab-prjt'...
Username for 'https://gitlab.com': pmane
Password for 'https://pmane@gitlab.com':
warning: You appear to have cloned an empty repository.
Checking connectivity... done.

C:\>_
```

The clone command makes a copy of repository into a new directory called *first-gitlab-prjt*.

Step 5 − Now go to your newly created directory and type the below command −

```
C:\>cd first-gitlab-prjt
C:\first-gitlab-prjt>touch README.md
```

The above command creates a *README.md* file in which you can put the information about your folder.

Step 6 − Add the *README.md* file to your created directory by using the below command −

```
C:\first-gitlab-prjt>git add README.md
```

Step 7 − Now store the changes to the repository along with the log message as shown below −

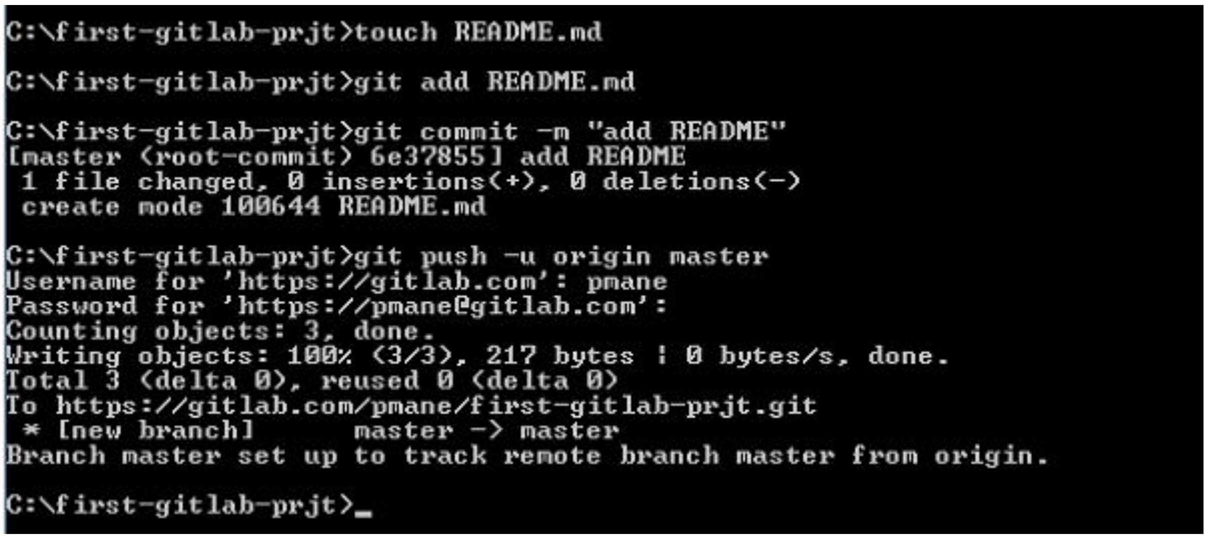
```
C:\first-gitlab-prjt>git commit -m "add README"
```

The flag *-m* is used for adding a message on the commit.

Step 8 − Push the commits to remote repository which are made on the local branch −

```
C:\first-gitlab-prjt>git push -u origin master
```

The below image depicts the usage of above commands in pushing the commits to remote repository −



```
C:\first-gitlab-prjt>touch README.md

C:\first-gitlab-prjt>git add README.md

C:\first-gitlab-prjt>git commit -m "add README"
[master (root-commit) 6e37855] add README
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

C:\first-gitlab-prjt>git push -u origin master
Username for 'https://gitlab.com': pmane
Password for 'https://pmane@gitlab.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 217 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.com/pmane/first-gitlab-prjt.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

C:\first-gitlab-prjt>_
```

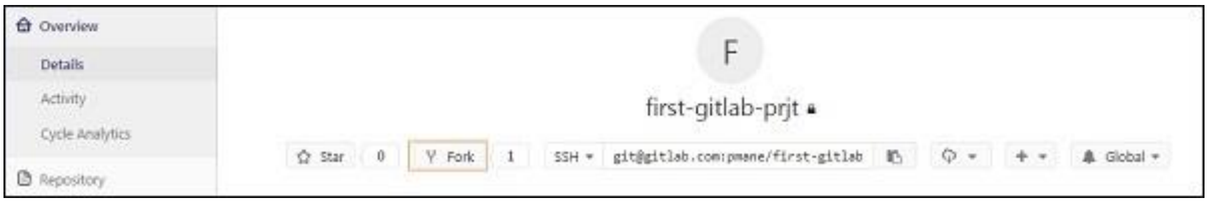
LAB 5: Forking a Project

Description

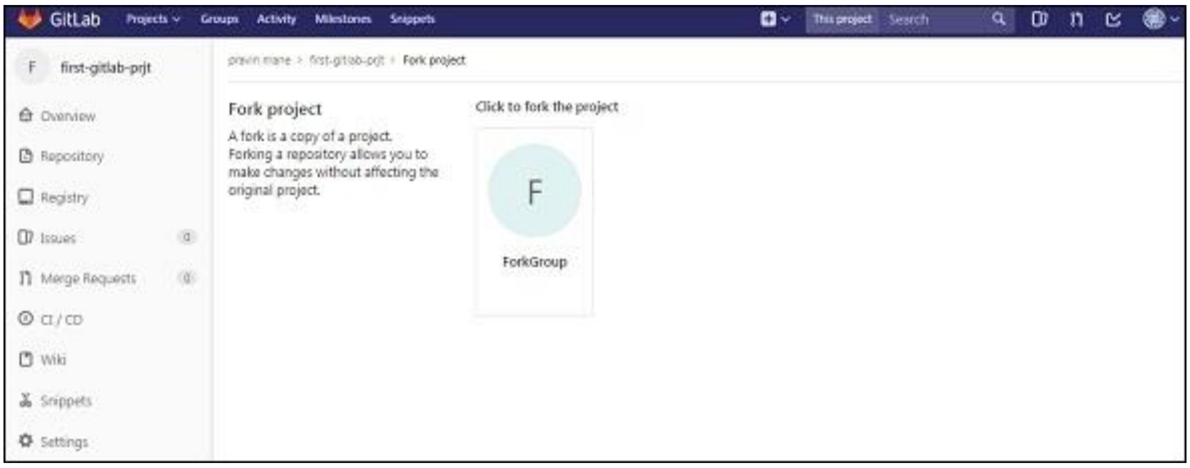
Fork is a duplicate of your original repository in which you can make the changes without affecting the original project.

Forking a Project

Step 1 − To fork a project, click on the *Fork* button as shown below −



Step 2 − After forking the project, you need to add the forked project to a fork group by clicking on it −



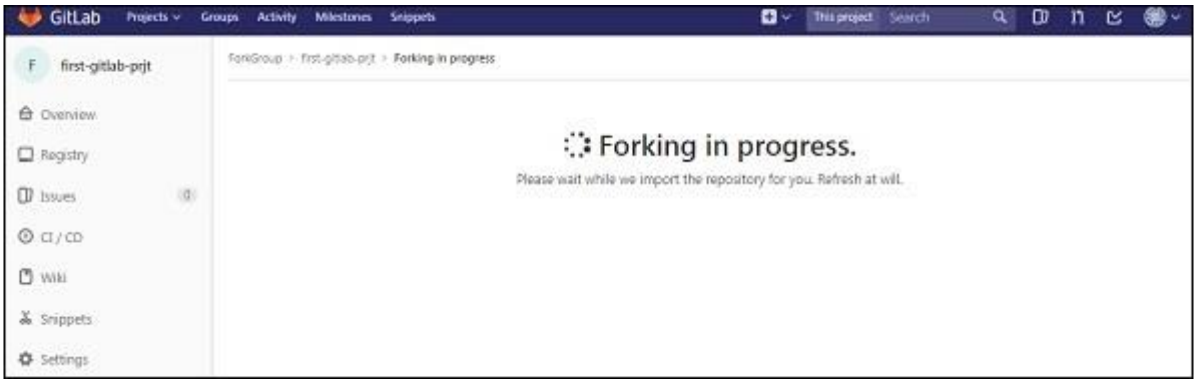
Fork project

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

Click to fork the project

ForkGroup

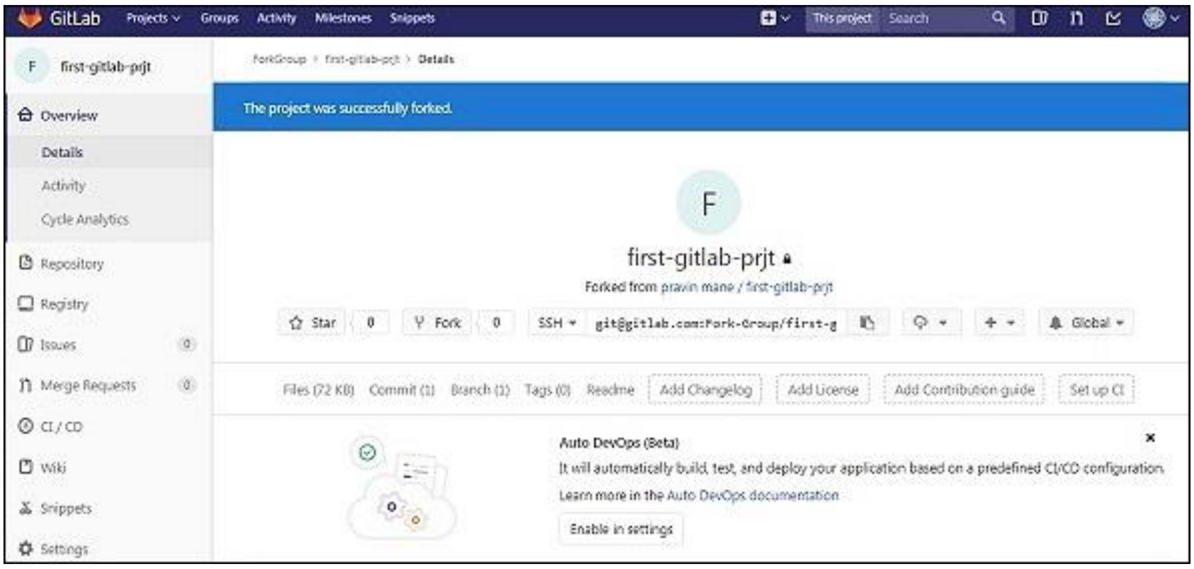
Step 3 − Next it will start processing of forking a project for sometime as shown below −



Forking in progress.

Please wait while we import the repository for you. Refresh at will.

Step 4 − It will display the success message after completion of forking the project process −



The project was successfully forked.

first-gitlab-prjt

Forked from pravin mane / first-gitlab-prjt

git@gitlab.com:Fork-Group/first-g

Auto DevOps (Beta)

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Learn more in the Auto DevOps documentation

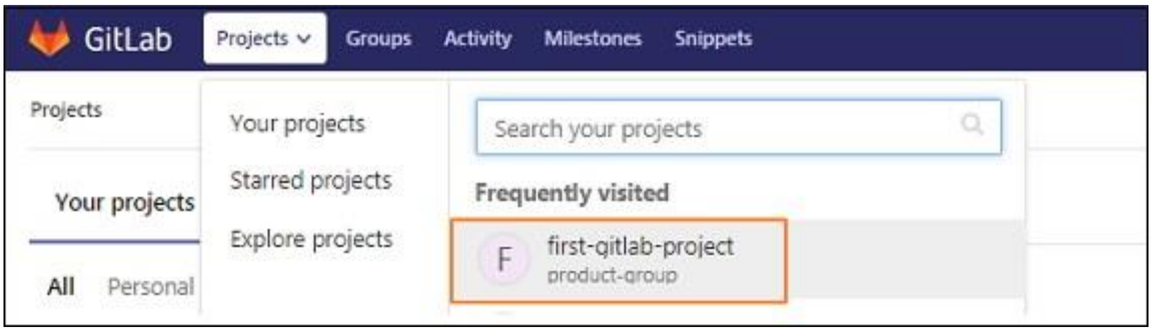
LAB 6: Creating a Branch

Description

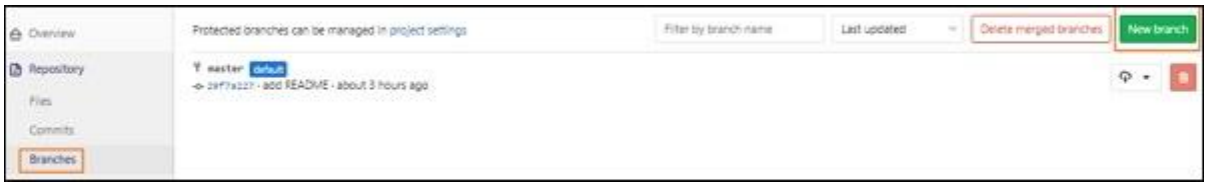
Branch is independent line and part of the development process. The creation of branch involves following steps.

Creating a Branch

Step 1 − Login to your GitLab account and go to your project under *Projects* section.



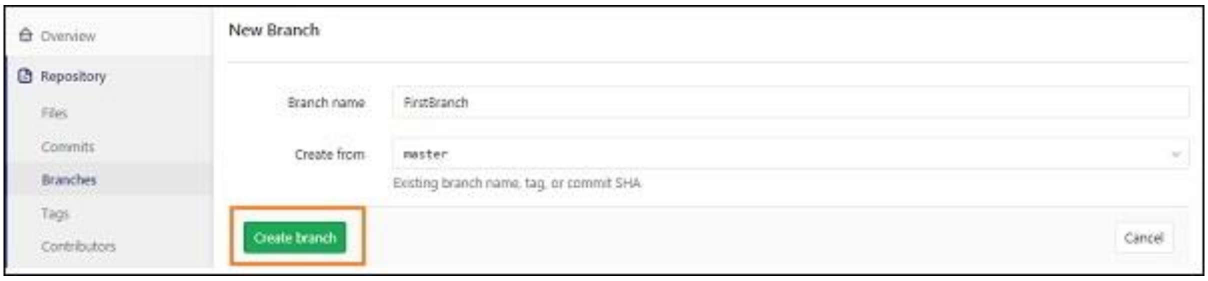
Step 2 − To create a branch, click on the *Branches* option under the *Repository* section and click on the *New branch* button.



Protected branches can be managed in project settings

master
29f7a227 · add README · about 3 hours ago

Step 3 − In the *New branch* screen, enter the name for branch and click on the *Create branch* button.

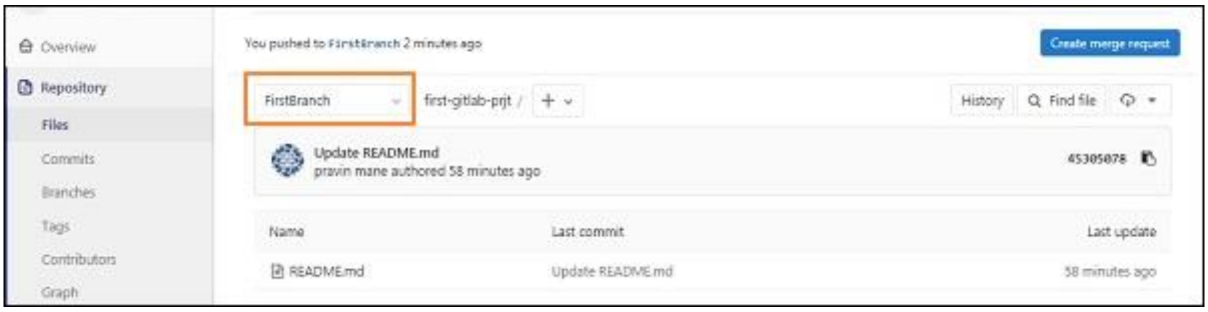


Branch name: FirstBranch

Create from: master

Existing branch name, tag, or commit SHA

Step 4 − After creating branch, you will get a below screen along with the created branch.



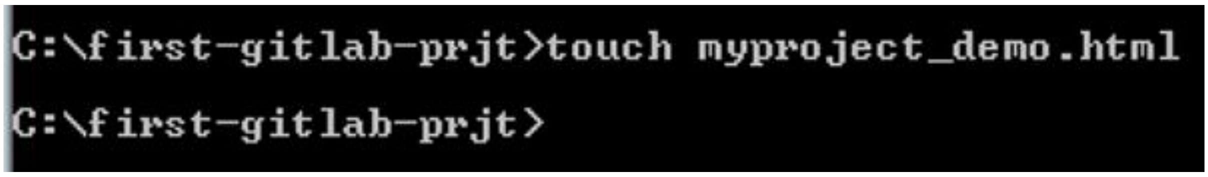
You pushed to FirstBranch 2 minutes ago

Update README.md
pravin mane authored 58 minutes ago
45305078

Name	Last commit	Last update
README.md	Update README.md	58 minutes ago

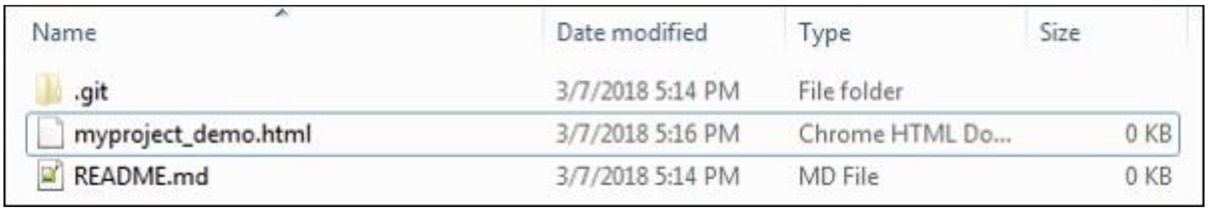
Creating a file using Command Line Interface

Step 1 − To create a file by using command line interface, type the below command in your project directory −



```
C:\first-gitlab-prjt>touch myproject_demo.html

C:\first-gitlab-prjt>
```

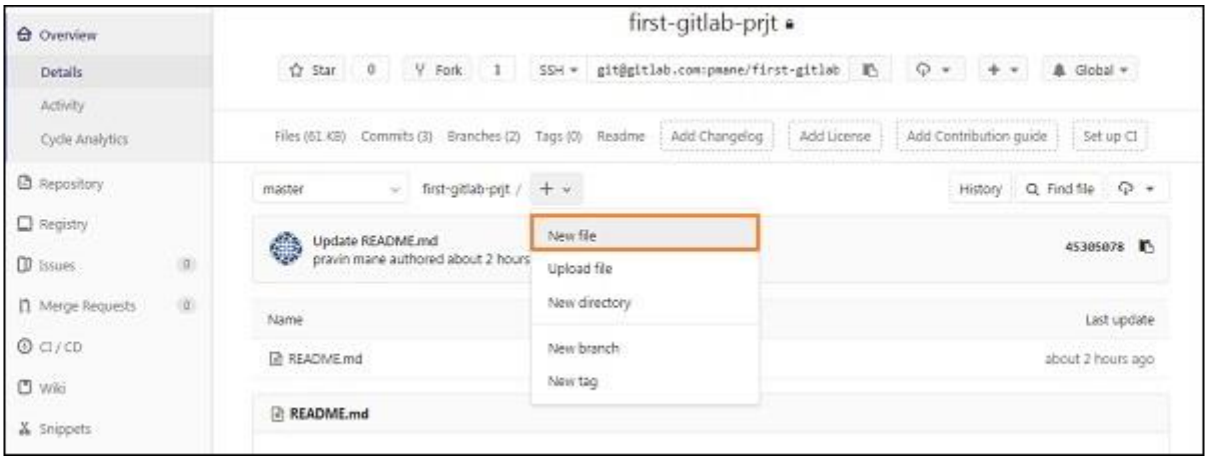
Step 2 − Now go to your project directory and you will see the created file −



Name	Date modified	Type	Size
.git	3/7/2018 5:14 PM	File folder	
myproject_demo.html	3/7/2018 5:16 PM	Chrome HTML Do...	0 KB
README.md	3/7/2018 5:14 PM	MD File	0 KB

Creating a file using Web Interface

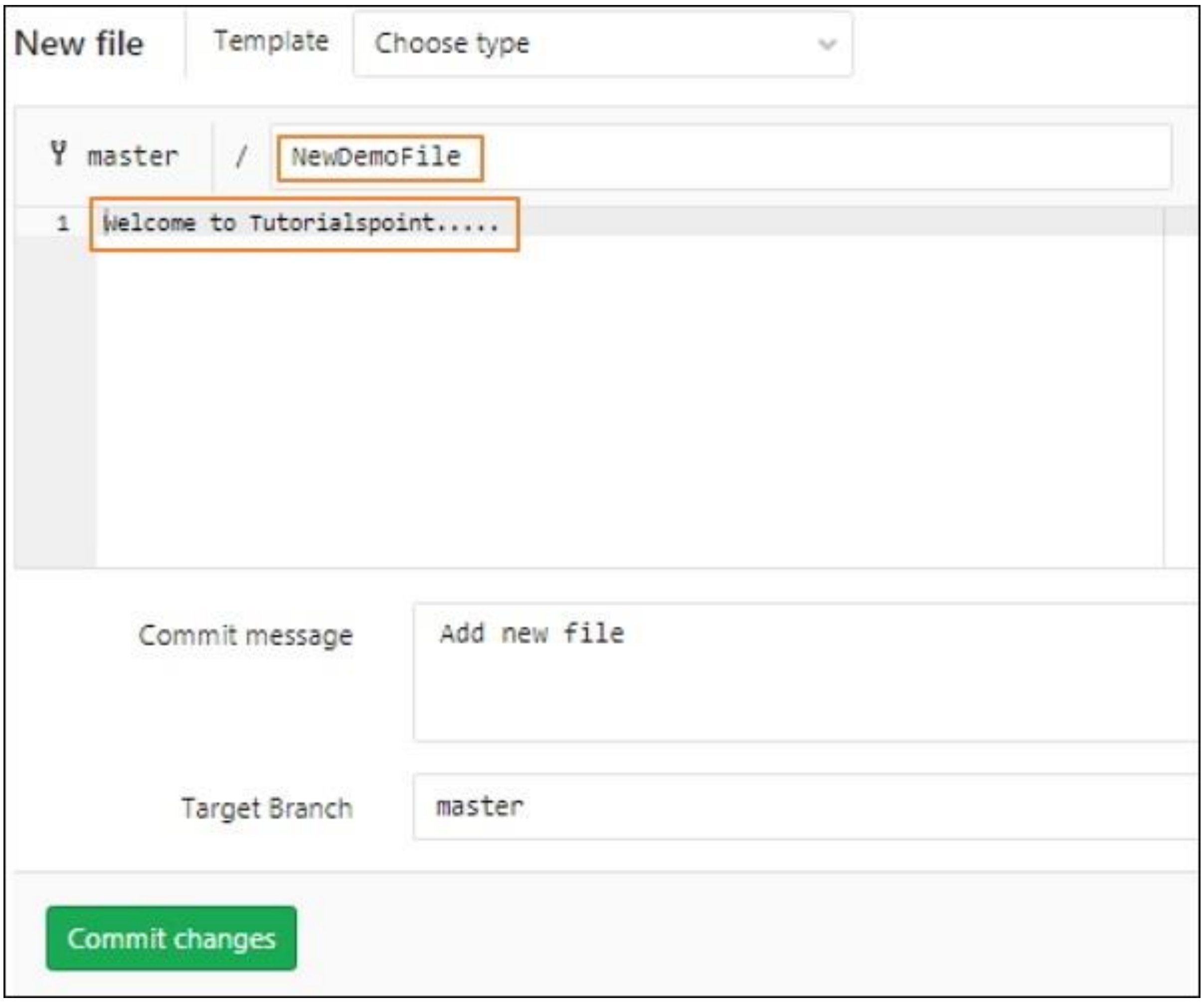
Step 1 − You can create a new file, by clicking on the '+' button which is at the right side of the branch selector in the dashboard −



Update README.md
pravin mane authored about 2 hours

README.md

Step 2 − Enter the file name, add some content in the editor section and click on the *Commit changes* button to create the file.


New file
Template
Choose type

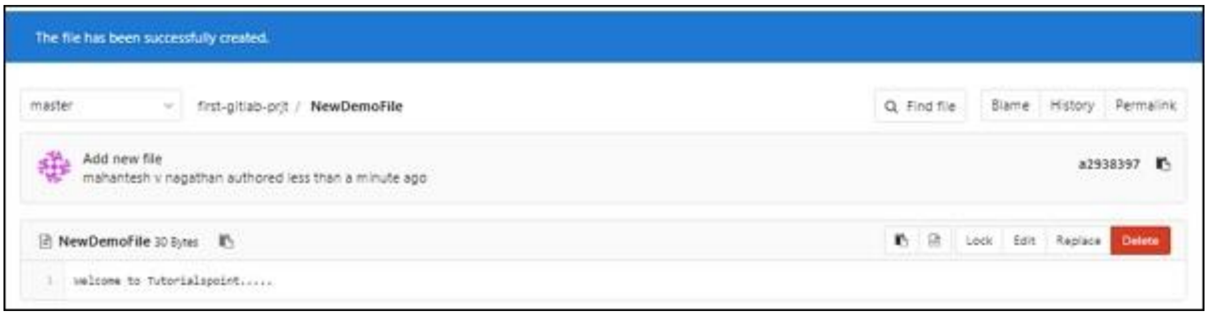
master / NewDemoFile

```
Welcome to Tutorialspoint.....
```

Commit message: Add new file

Target Branch: master

Step 3 − Now you will get a successful message after creating the file as shown below −


The file has been successfully created.

master first-gitlab-prjt / NewDemoFile

Add new file
mahantesh v nagathan authored less than a minute ago
a2938397

NewDemoFile 30 Bytes

```
welcome to Tutorialspoint.....
```

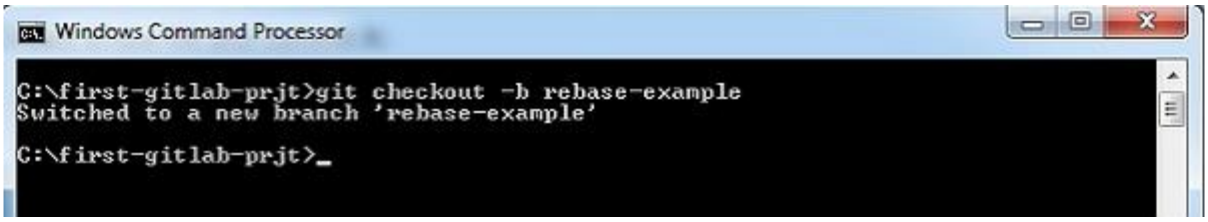
LAB 7: Exploring Rebase Operation

Description

Rebase is a way of merging *master* to your branch when you are working with long running branch.

Steps for Rebase Operation

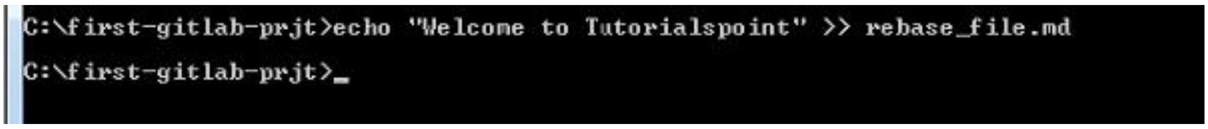
Step 1 − Go to your project directory and create a new branch with the name *rebase-example* by using the *git checkout* command −



```
C:\first-gitlab-prjt>git checkout -b rebase-example
Switched to a new branch 'rebase-example'

C:\first-gitlab-prjt>_
```

The flag *-b* indicates new branch name.

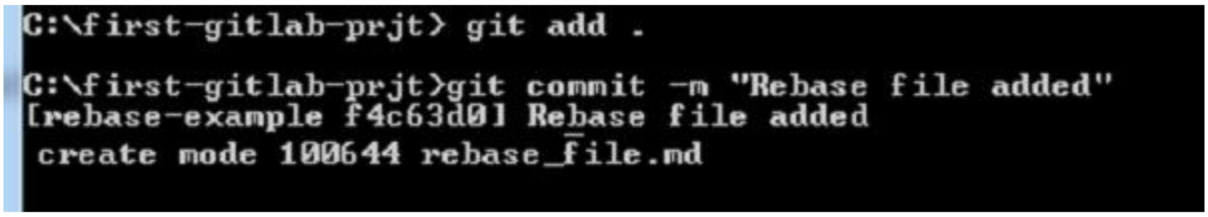
Step 2 − Now, create a new file and add some content to that file as shown below −



```
C:\first-gitlab-prjt>echo "Welcome to Tutorialspoint" >> rebase_file.md

C:\first-gitlab-prjt>_
```

The content 'Welcome to Tutorialspoint' will be added to the *rebase_file.md* file.

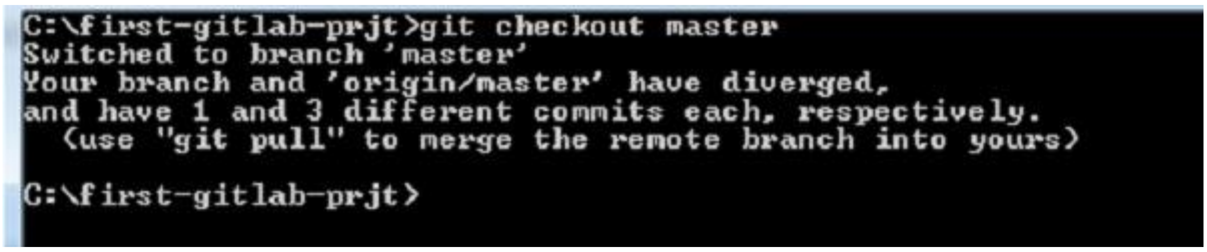
Step 3 − Add the new file to working directory and store the changes to the repository along with the message (by using the *git commit* command) as shown below −



```
C:\first-gitlab-prjt> git add .

C:\first-gitlab-prjt>git commit -m "Rebase file added"
[rebase-example f4c63d0] Rebase file added
 create mode 100644 rebase_file.md
```

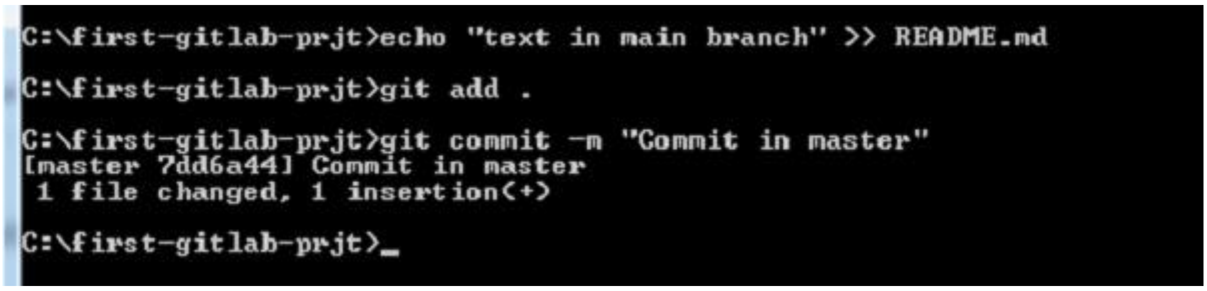
The flag *-m* is used for adding a message on the commit.

Step 4 − Now, switch to the 'master' branch. You can fetch the remote branch(*master* is a branch name) by using the *git checkout* command −



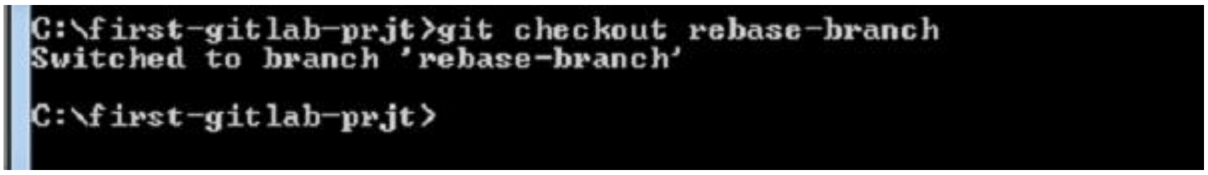
```
C:\first-gitlab-prjt>git checkout master
Switched to branch 'master'
Your branch and 'origin/master' have diverged,
and have 1 and 3 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

C:\first-gitlab-prjt>
```

Step 5 − Next, create an another new file, add some content to that file and commit it in the *master* branch.



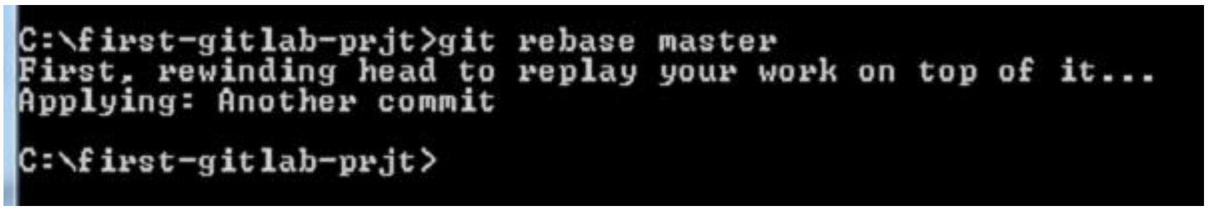
```
C:\first-gitlab-prjt>echo "text in main branch" >> README.md

C:\first-gitlab-prjt>git add .

C:\first-gitlab-prjt>git commit -m "Commit in master"
[master 7dd6a44] Commit in master
 1 file changed, 1 insertion(+)

C:\first-gitlab-prjt>_
```

Step 6 − Switch to the *rebase-branch* to have the commit of *master* branch.



```
C:\first-gitlab-prjt>git checkout rebase-branch
Switched to branch 'rebase-branch'

C:\first-gitlab-prjt>
```

Step 7 − Now, you can combine the commit of *master* branch to *rebase-branch* by using the *git rebase* command −



```
C:\first-gitlab-prjt>git rebase master
First, rewinding head to replay your work on top of it...
Applying: Another commit

C:\first-gitlab-prjt>
```

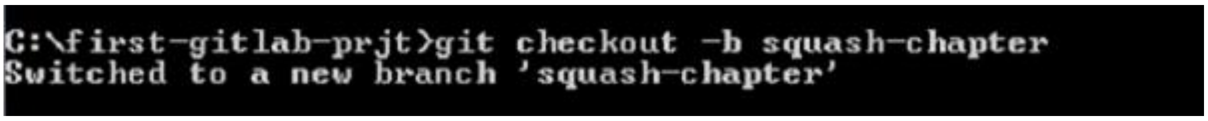
LAB 8: Squashing Commits

Description

Squashing is a way of combining all commits into one when you are obtaining a merge request.

Steps for Squashing Commits

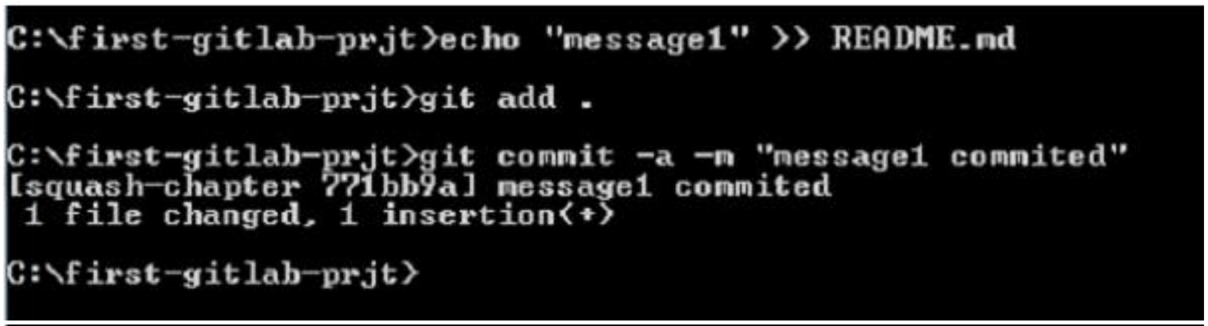
Step 1 − Go to your project directory and check out a new branch with the name *squash-chapter* by using the *git checkout* command −



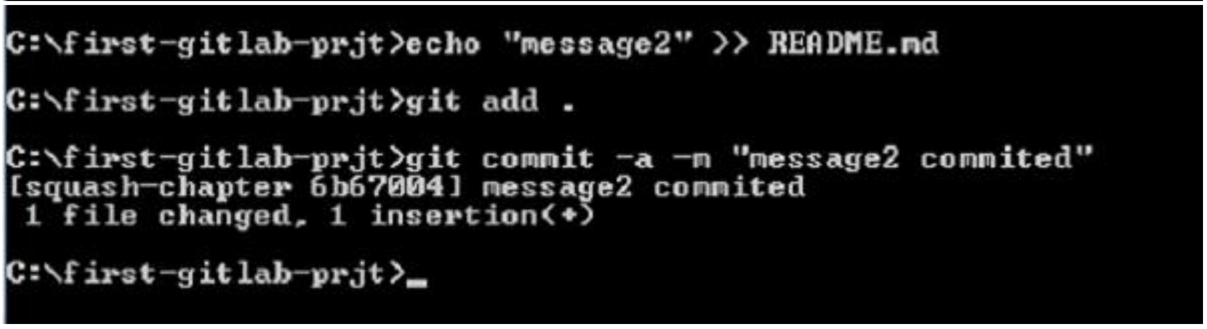
```
C:\first-gitlab-prjt>git checkout -b squash-chapter
Switched to a new branch 'squash-chapter'
```

The flag *-b* indicates new branch name.

Step 2 − Now, create a new file with two commits, add that file to working directory and store the changes to the repository along with the commit messages as shown below −



```
C:\first-gitlab-prjt>echo "message1" >> README.md

C:\first-gitlab-prjt>git add .

C:\first-gitlab-prjt>git commit -a -m "message1 commited"
[squash-chapter 771bb9a] message1 commited
 1 file changed, 1 insertion(+)

C:\first-gitlab-prjt>
```



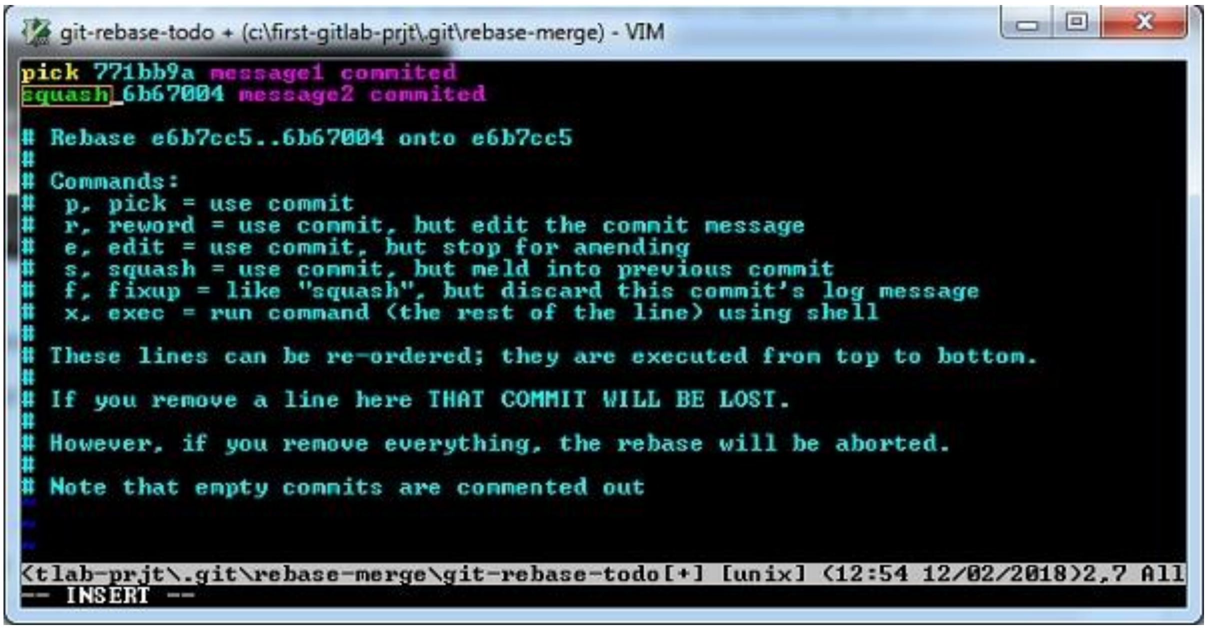
```
C:\first-gitlab-prjt>echo "message2" >> README.md

C:\first-gitlab-prjt>git add .

C:\first-gitlab-prjt>git commit -a -m "message2 commited"
[squash-chapter 6b67004] message2 commited
 1 file changed, 1 insertion(+)

C:\first-gitlab-prjt>_
```

Step 3 − Now, squash the above two commits into one commit by using the below command −

```
$ git rebase -i HEAD~2
```

Here, *git rebase* command is used to integrate changes from one branch to another and *HEAD~2* specifies last two squashed commits and if you want to squash four commits, then you need to write as *HEAD~4*. One more important point is, you need atleast two commits to complete the squash operation.

Step 4 − After entering the above command, it will open the below editor in which you have to change the *pick* word to *squash* word in the second line (you need to squash this commit).

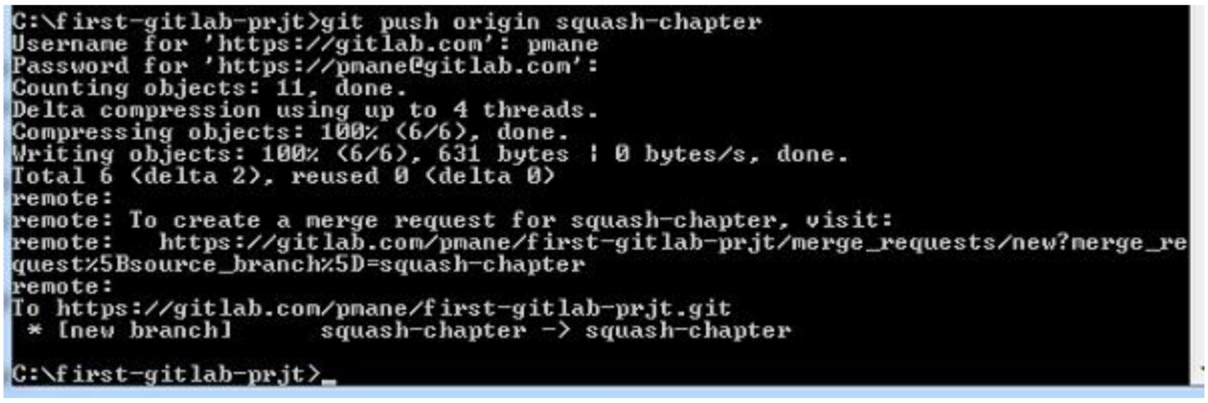


git-rebase-todo + (c:\first-gitlab-prjt\.git\rebase-merge) - VIM

```
pick 771bb9a message1 commited
squash 6b67004 message2 commited

# Rebase e6b7cc5..6b67004 onto e6b7cc5
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out

<tlab-prjt\.git\rebase-merge\git-rebase-todo[+] [unix] (12:54 12/02/2018)2,7 All
-- INSERT --
```

Now press the *Esc* key, then colon(:) and type *wq* to save and exit from the screen.

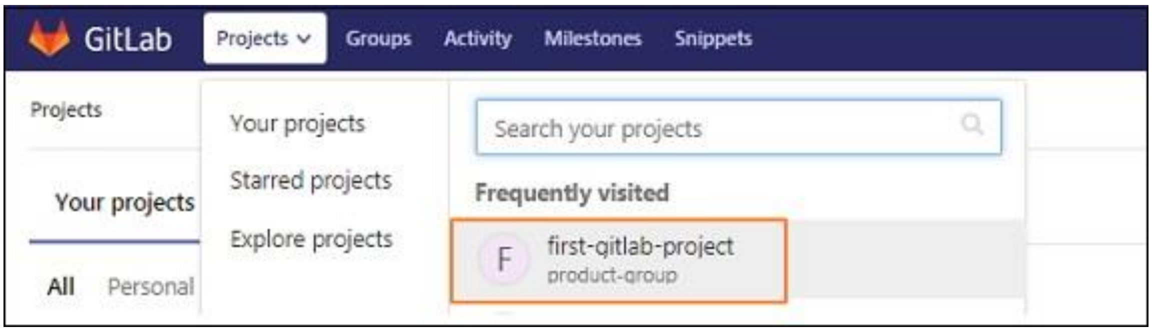
Step 5 − Now push the branch to remote repository as shown below −



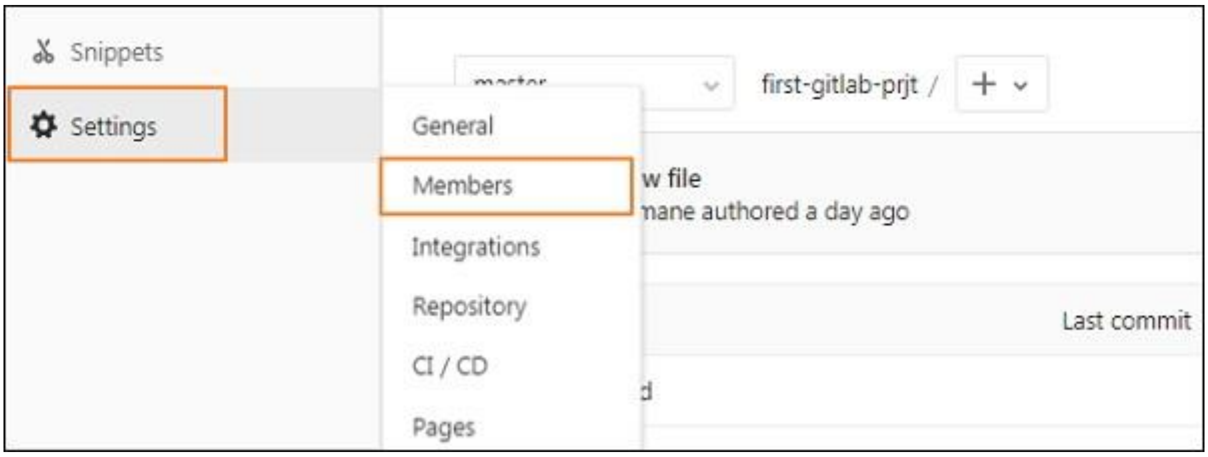
```
C:\first-gitlab-prjt>git push origin squash-chapter
Username for 'https://gitlab.com': pmane
Password for 'https://pmane@gitlab.com':
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 631 bytes | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote:
remote: To create a merge request for squash-chapter, visit:
remote:   https://gitlab.com/pmane/first-gitlab-prjt/merge_requests/new?merge_re
quest%5Bsource_branch%5D=squash-chapter
remote:
To https://gitlab.com/pmane/first-gitlab-prjt.git
 * [new branch]      squash-chapter -> squash-chapter

C:\first-gitlab-prjt>_
```

Steps for Adding User

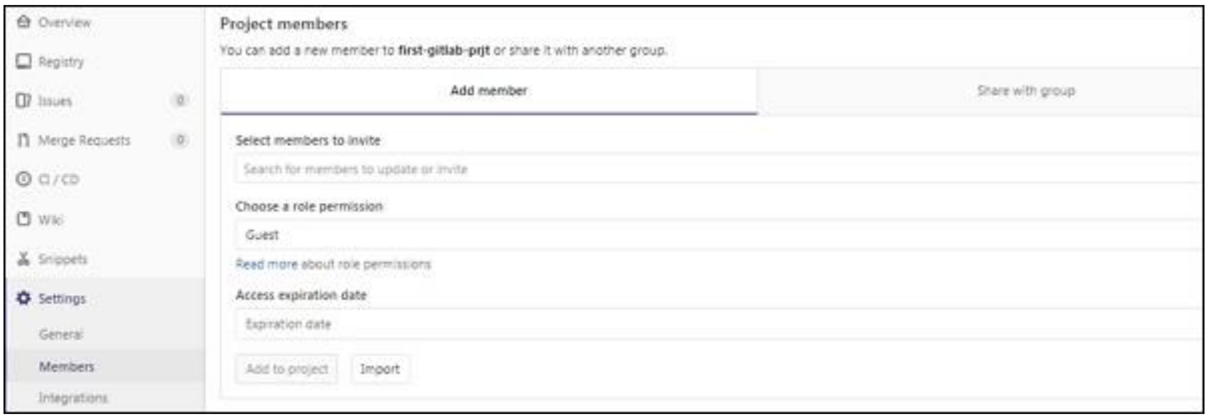
Step 1 − Login to your GitLab account and go to your project under *Projects* section.



Step 2 − Next, click on the *Members* option under *Settings* tab −



Step 3 − It will open the below screen to add the member to your project −



Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

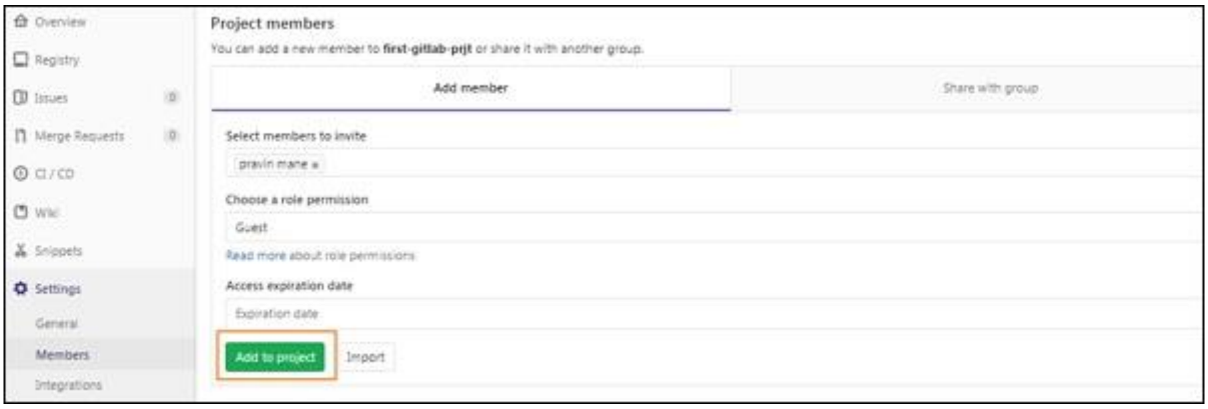
Select members to invite: Search for members to update or invite

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Step 4 − Now enter the user name, role permission, expiration date(optional) and click on *Add to project* button to add the user to project −



Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

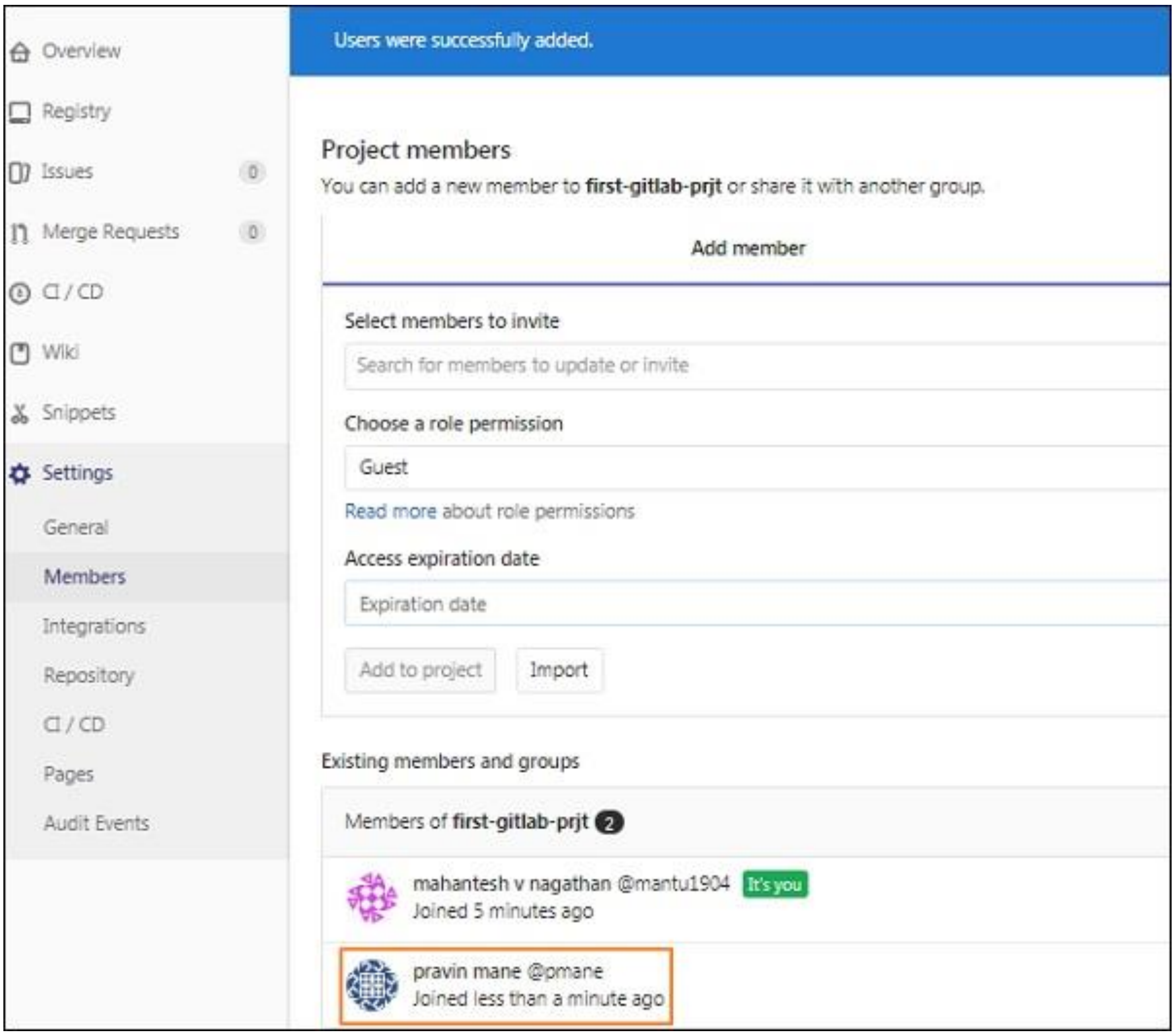
Select members to invite: pravin mane ×

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Step 5 − Next, you will get a successful message after adding user to the project.


Users were successfully added.

Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

Select members to invite: Search for members to update or invite

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Existing members and groups

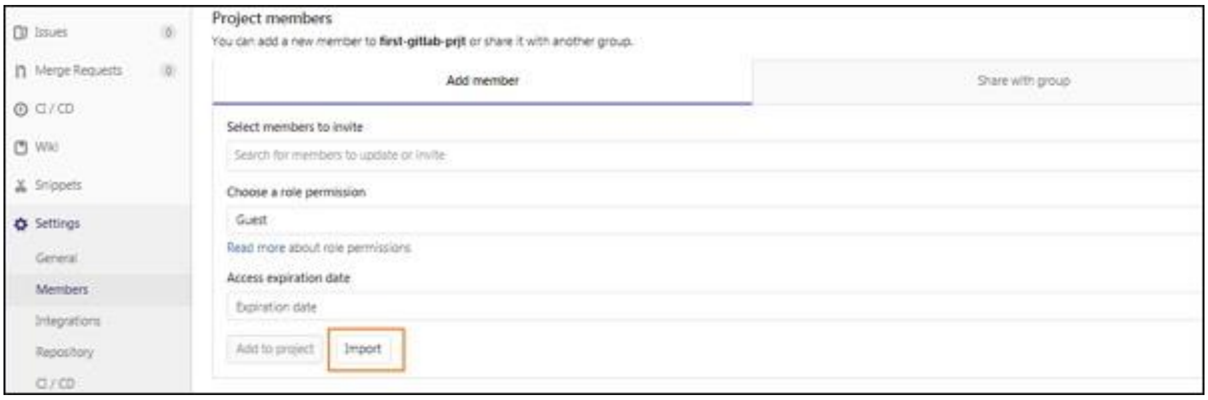
Members of **first-gitlab-prjt** 2

mahantesh v nagathan @mantu1904 It's you
Joined 5 minutes ago

pravin mane @pmane
Joined less than a minute ago

The highlighted box in the above image indicates, a new user has been added to the project −

Step 6 − You can also add user to the project by clicking on the *Import* button −


Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

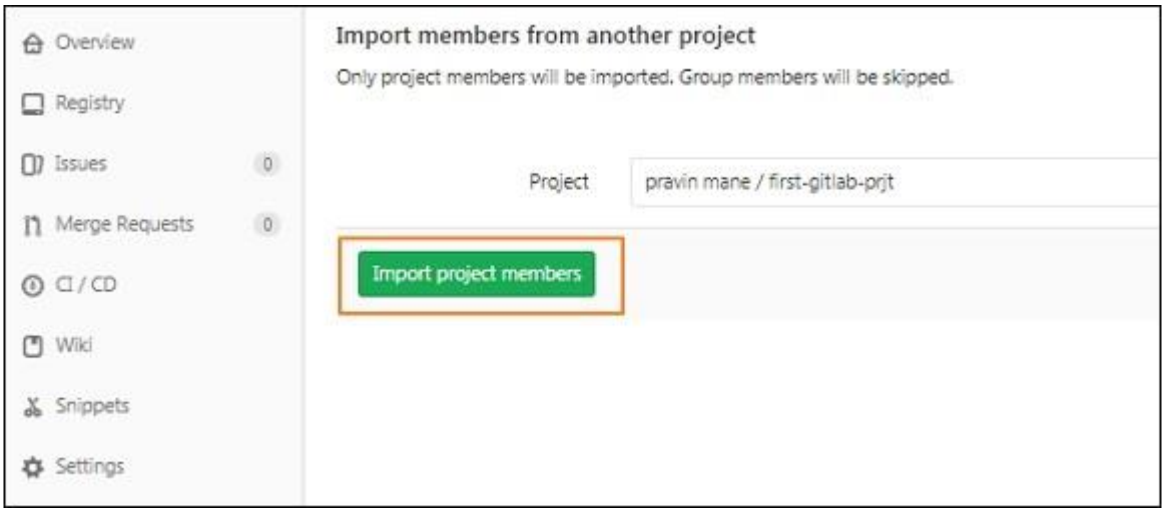
Select members to invite: Search for members to update or invite

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Step 7 − Now select the project from which you want to add the user to your project and click on the *Import project members* button −



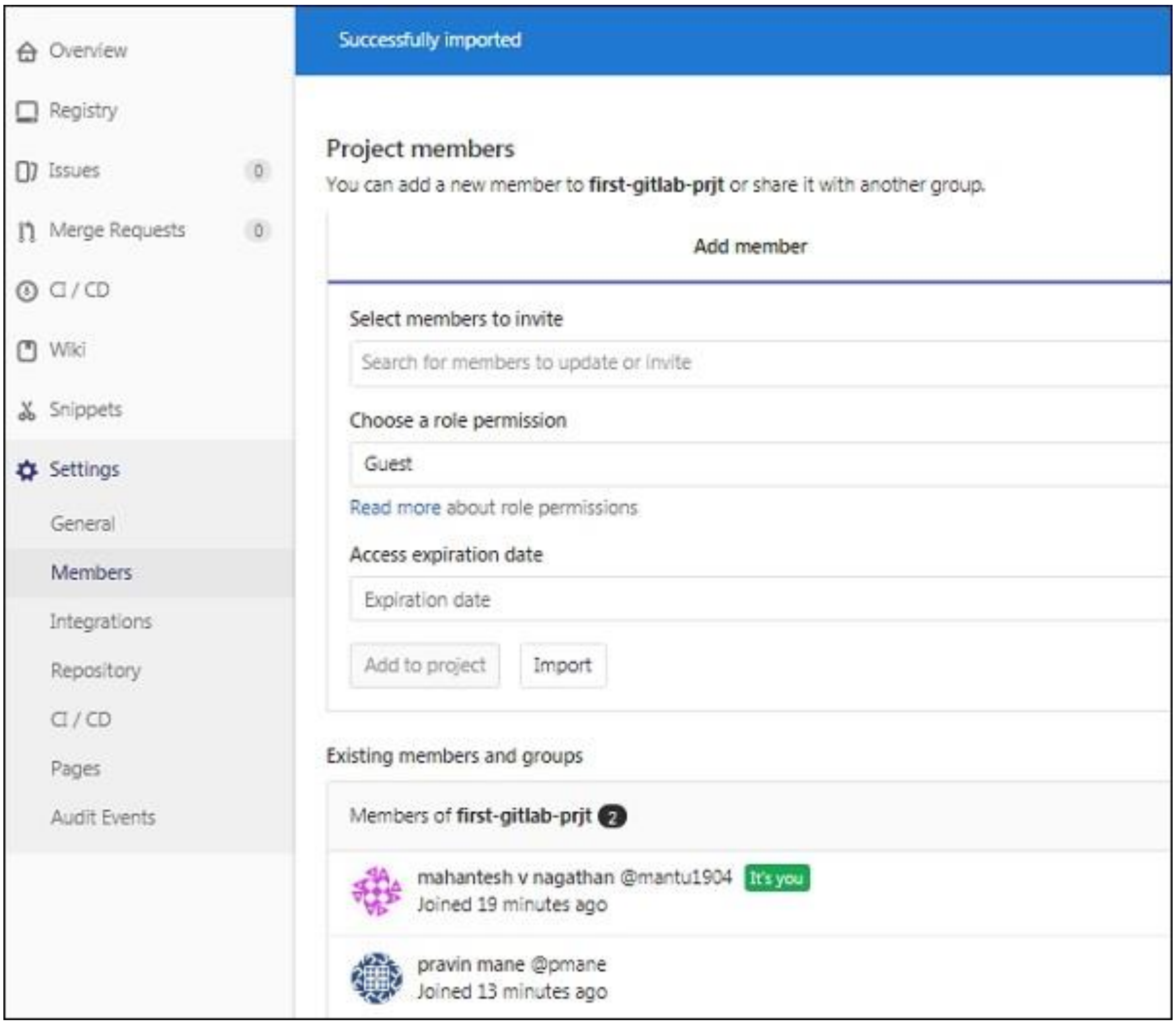
Import members from another project

Only project members will be imported. Group members will be skipped.

Project: pravin mane / first-gitlab-prjt

Import project members

Step 8 − You will get a success message after importing user to the project −



Successfully imported

Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

Add member

Select members to invite: Search for members to update or invite

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Add to project | Import

Existing members and groups

Members of **first-gitlab-prjt** 2

mahantesh v nagathan @mantu1904 It's you
Joined 19 minutes ago

pravin mane @pmane
Joined 13 minutes ago

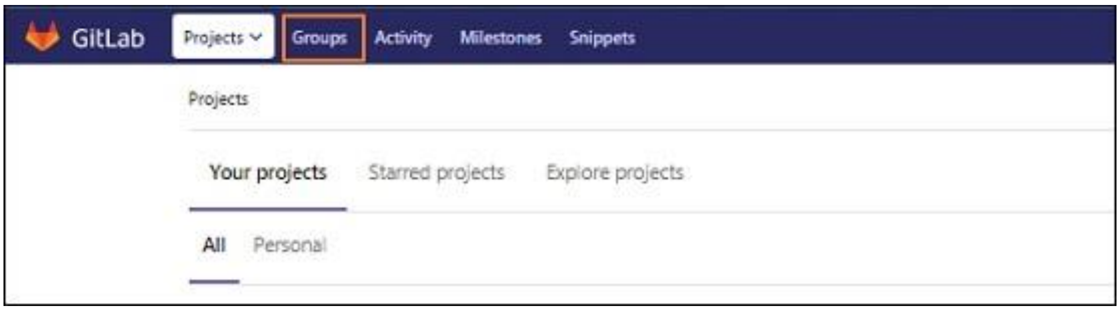
LAB 9: Creating Groups

Description

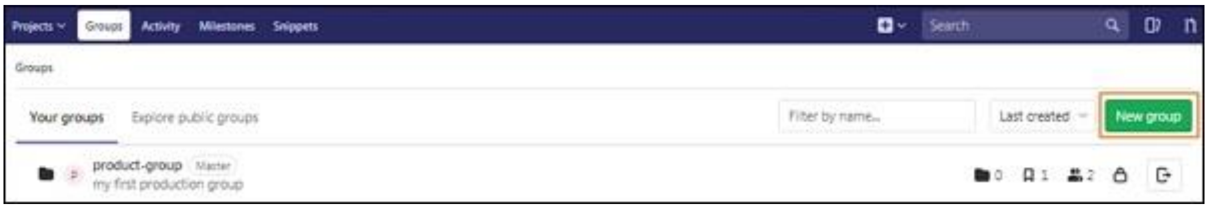
Creating group helps to connect multiple repositories and allows members to access the project by giving permissions on the group level.

Steps for Creating Group

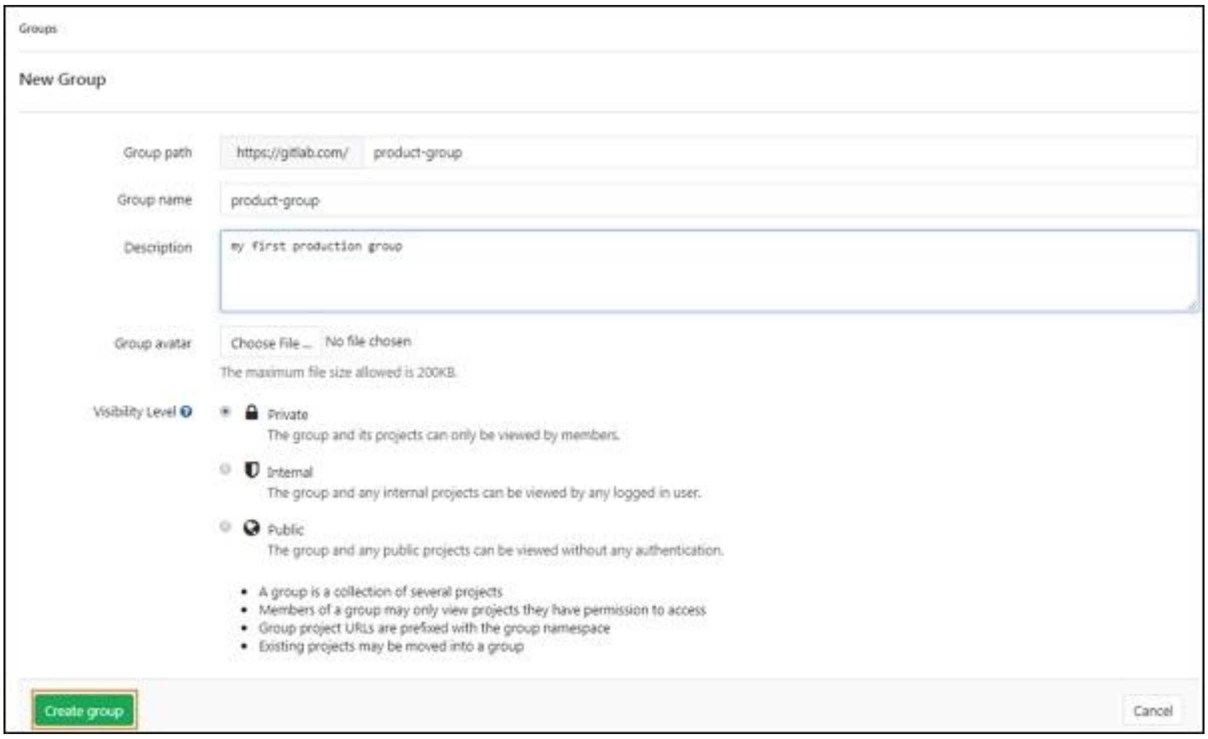
Step 1 − Login to your GitLab account and click on the *Groups* menu −



Step 2 − Next, you will get the below screen and click on the *New group* button to create a group −



Step 3 − Enter the *Group name*, *Description*, *visibility level*(Private/Public/Internal) and also you can set the image for the group of your choice which should be within 200kb in size. Now click on the *Create group* button.


Groups

New Group

Group path: https://gitlab.com/ product-group

Group name: product-group

Description: my first production group

Group avatar: Choose File... No file chosen

The maximum file size allowed is 200KB.

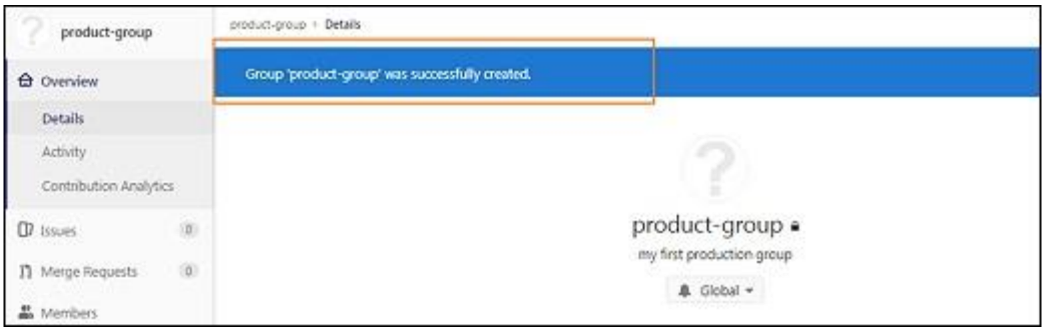
Visibility Level: [✓ Private] The group and its projects can only be viewed by members.

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.

- A group is a collection of several projects
- Members of a group may only view projects they have permission to access
- Group project URLs are prefixed with the group namespace
- Existing projects may be moved into a group

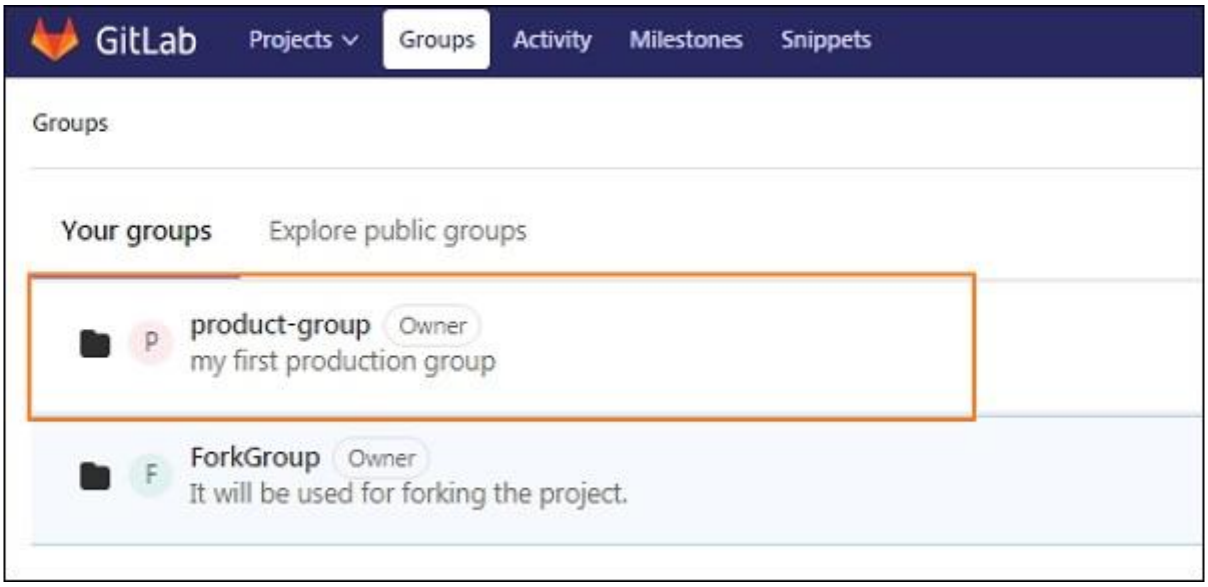
Step 4 − Next, it will display the success message after creating the group as shown below −


Group 'product-group' was successfully created.

product-group

my first production group

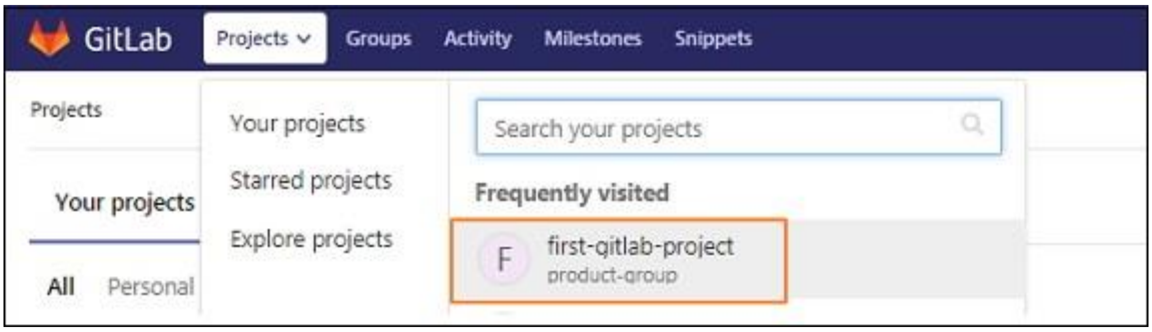
Step 5 − Now, go back to your *Groups* section and you will see the created group in the list −



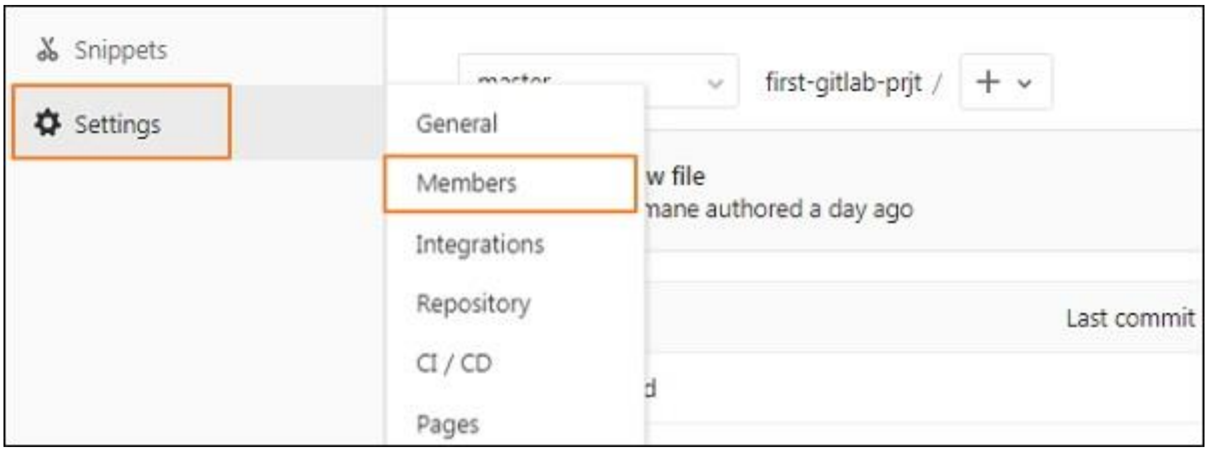
product-group Owner	my first production group
ForkGroup Owner	It will be used for forking the project.

Steps for Removing User

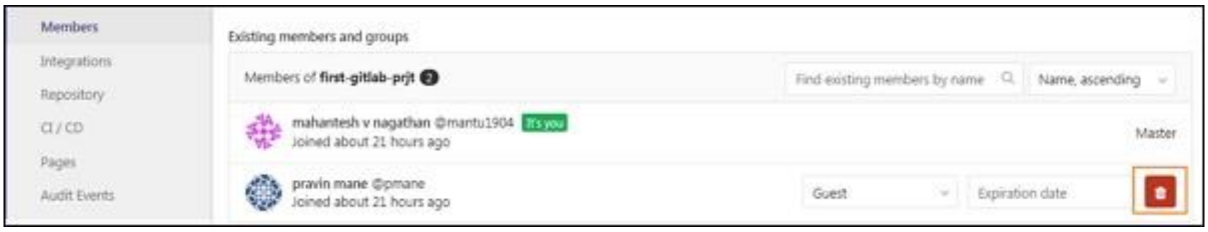
Step 1 − Login to your GitLab account and go to your project under *Projects* section −



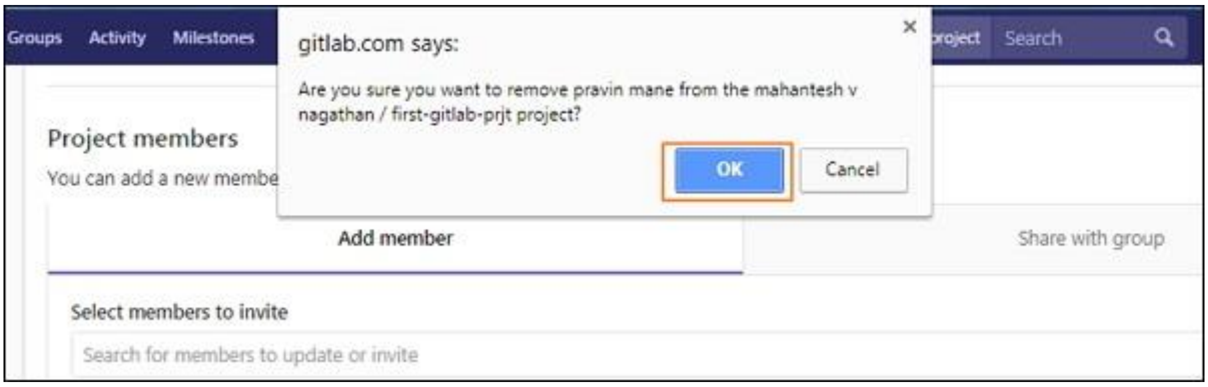
Step 2 − Now, click on the *Members* option under *Settings* tab −



Step 3 − You will see the list of users under *Existing members and groups* section and click on the delete option at right side to remove the user from project −


Members
Integrations
Repository
CI / CD
Pages
Audit Events
Existing members and groups
Members of first-gitlab-prjt 2
Find existing members by name
Name, ascending
mahantesh v nagathan @mantu1904
It's you
Joined about 21 hours ago
Master
pravin mane @pmane
Joined about 21 hours ago
Guest
Expiration date

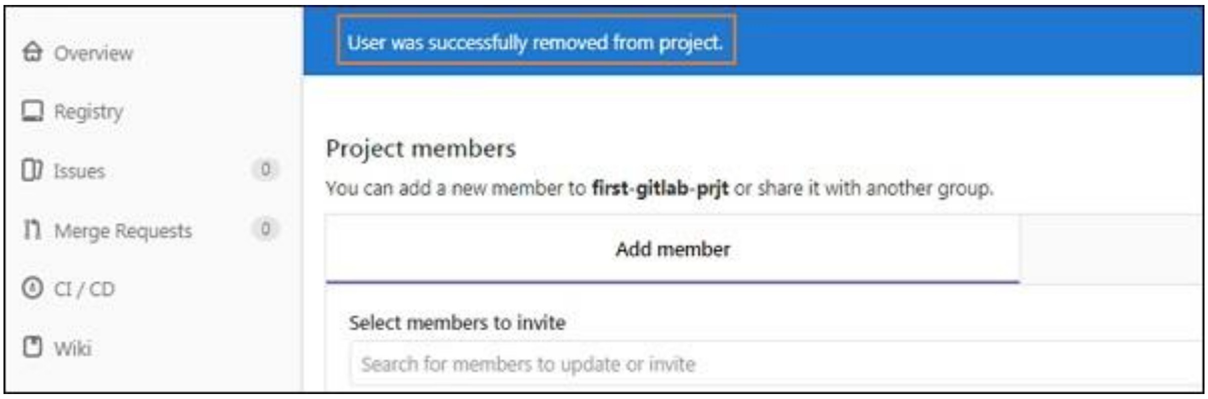
Step 4 − After clicking remove button, it will display a pop-up window saying whether to remove the selected user from the project or not. Click on *Ok* button to remove the user.



gitlab.com says:

Are you sure you want to remove pravin mane from the mahantesh v nagathan / first-gitlab-prjt project?

Step 5 − Now, it will display the success message after removing the user from the project as shown in the image below −



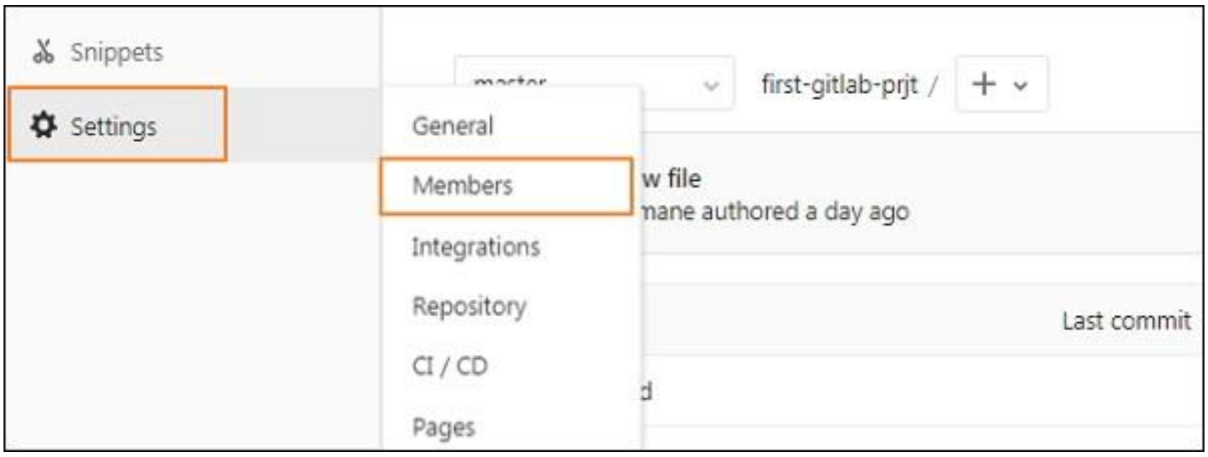
User was successfully removed from project.

Project members

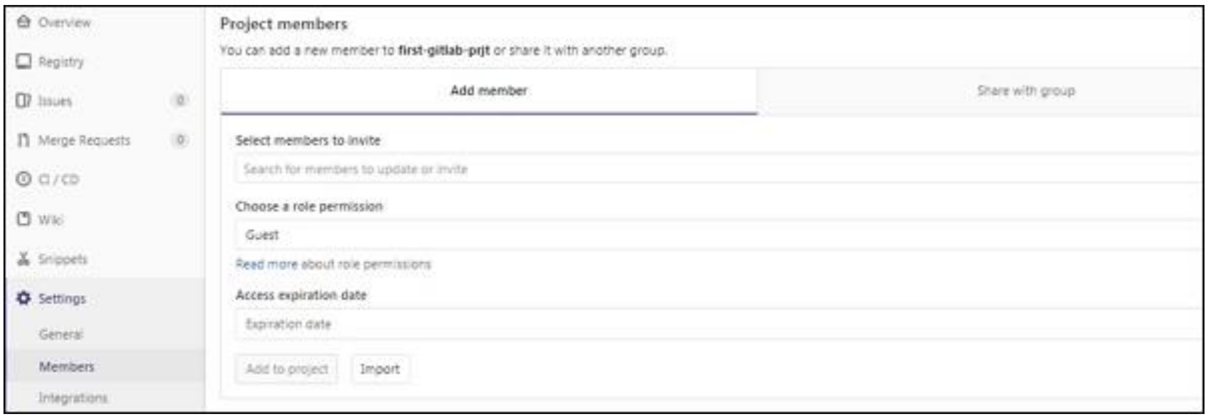
You can add a new member to **first-gitlab-prjt** or share it with another group.

Steps for creating User Permissions

Step 1 − Login to your GitLab account and click on the *Members* option under *Settings* tab −



Step 2 − It will open the below screen to add the member to your project −



Project members

You can add a new member to **first-gitlab-prjt** or share it with another group.

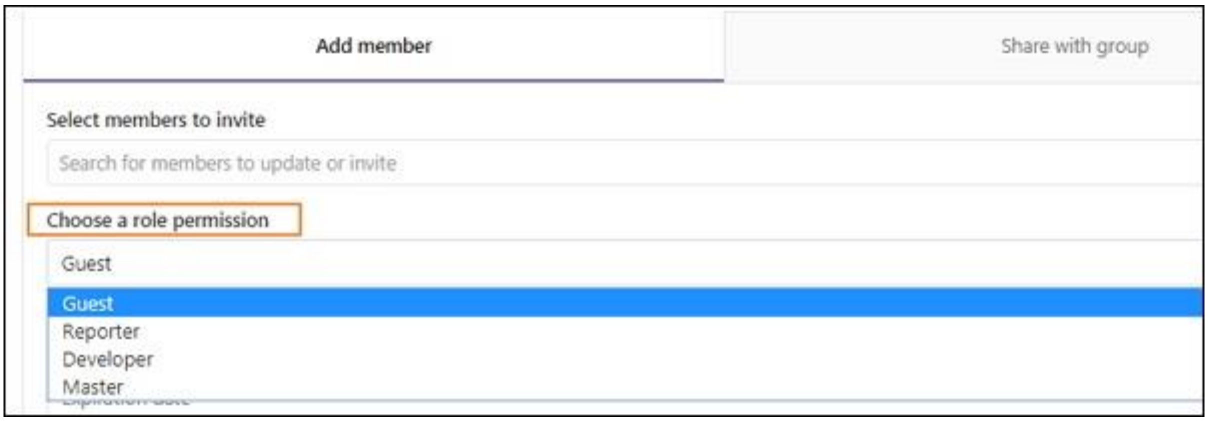
Select members to invite: Search for members to update or invite

Choose a role permission: Guest

Read more about role permissions

Access expiration date: Expiration date

Step 3 − You will see the different types of permissions when you click on a dropdown under *Choose a role permission* section −



Select members to invite: Search for members to update or invite

Choose a role permission: Guest

- Guest
- Reporter
- Developer
- Master

You can see the Adding users chapter for setting user permission and adding user to project. Here, we will briefly discuss about different user permissions which can be applied to projects.

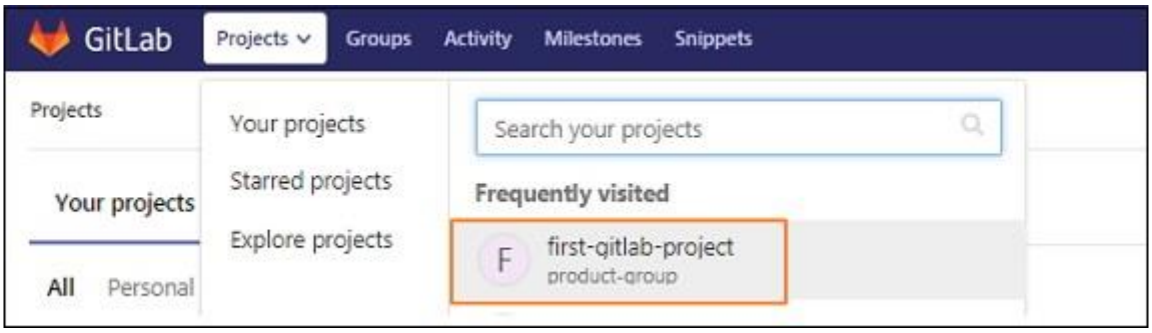
S.N.	Guest	Reporter	Developer	Master
1	Creates a new issue	Creates a new issue	Creates a new issue	Creates a new issue
2	Can leave comments	Can leave comments	Can leave comments	Can leave comments
3	Able to write on project wall	Able to write on project wall	Able to write on project wall	Able to write on project wall
4	-	Able to pull project code	Able to pull project code	Able to pull project code
5	-	Can download project	Can download project	Can download project
6	-	Able to write code snippets	Able to write code snippets	Able to write code snippets
7	-	-	Create new merge request	Create new merge request
8	-	-	Create new branch	Create new branch
9	-	-	Push and remove non protected branches	Push and remove non protected branches
10	-	-	Includes tags	Includes tags
11	-	-	Can create, edit, delete project milestones	Can create, edit, delete project milestones
12	-	-	Can create or update commit status	Can create or update commit status
13	-	-	Write a wiki	Write a wiki

14	-	-	Create new environments	Create new environments
15	-	-	Cancel and retry the jobs	Cancel and retry the jobs
16	-	-	Updates and removes the registry image	Updates and removes the registry image
17	-	-	-	Can add new team members
18	-	-	-	Push and remove protected branches
19	-	-	-	Can edit the project
20	-	-	-	Can manage runners, job triggers and variables
21	-	-	-	Add deploy keys to project
22	-	-	-	Able to manage clusters
23	-	-	-	Configure project hooks
24	-	-	-	Can enable/disable the branch protection
25	-	-	-	Able to rewrite or remove Git tags

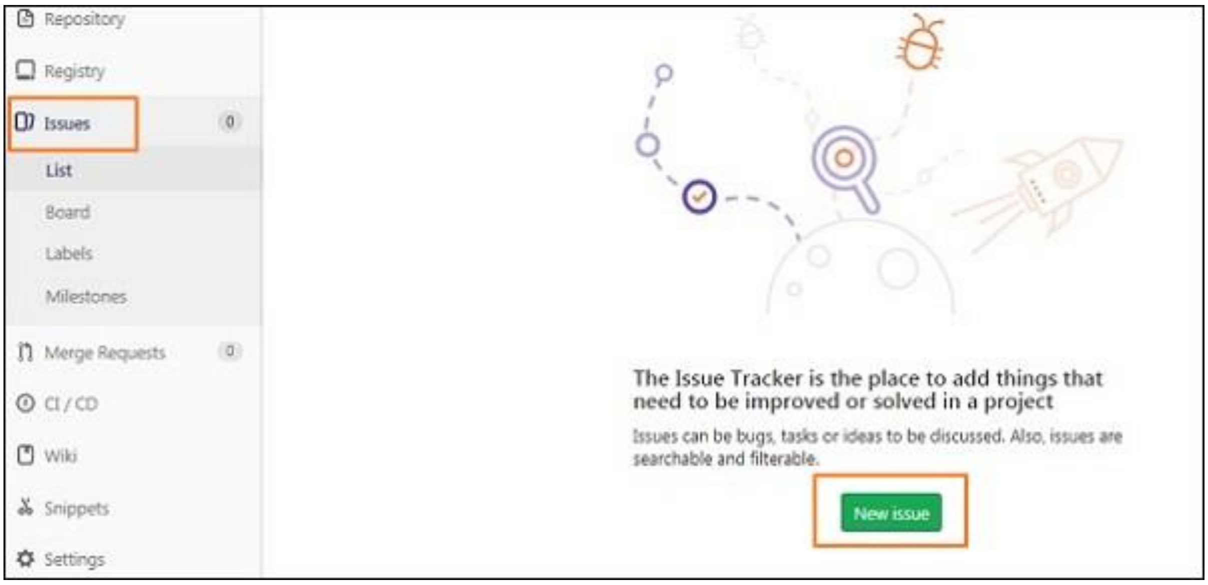
The following table shows available permission levels for different types of users −

we will discuss about how to create an issue in a project −

Step 1 − Login to your GitLab account and go to your project under *Projects* section −



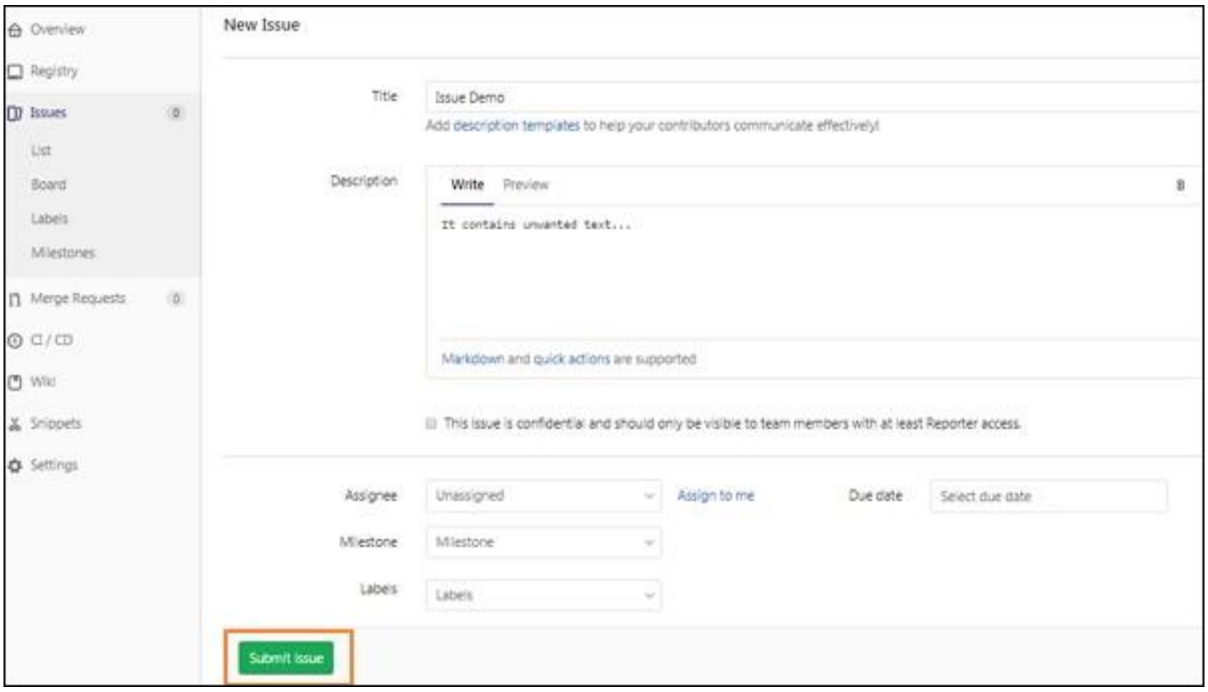
Step 2 − Go to *Issues* tab and click on the *New issue* button to create a new issue as shown below −



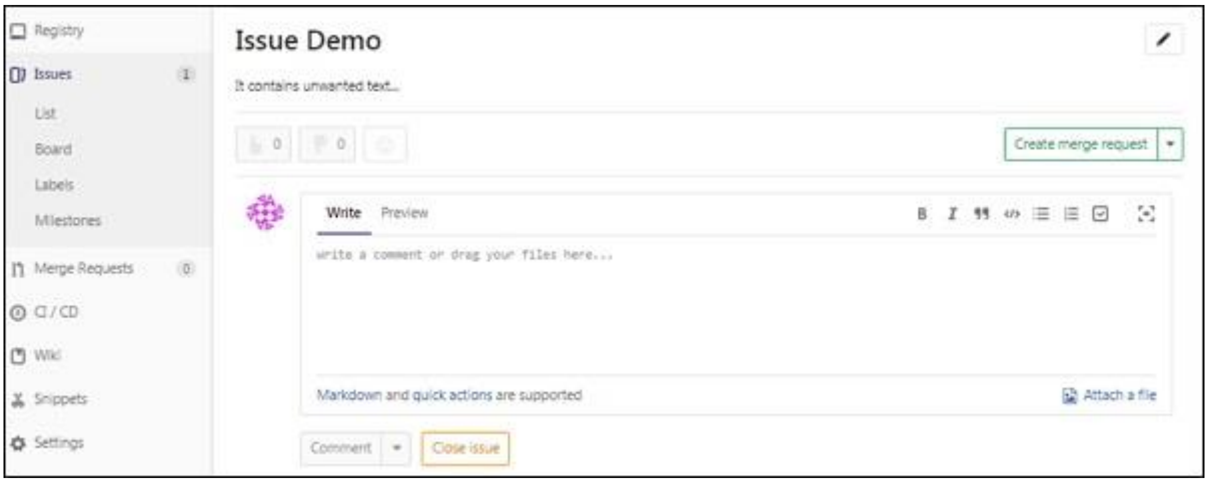
The Issue Tracker is the place to add things that need to be improved or solved in a project

Issues can be bugs, tasks or ideas to be discussed. Also, issues are searchable and filterable.

Step 3 − Now, fill the information such as title, description and if you want, you can select a user to assign an issue, milestone(refer this chapter for more information), labels upon operation or could be choose by developers themselves later.


Overview
Registry
Issues 0
List
Board
Labels
Milestones
Merge Requests 0
CI / CD
Wiki
Snippets
Settings
New Issue
Title
Issue Demo
Add description templates to help your contributors communicate effectively!
Description
Write
Preview
It contains unwanted text...
Markdown and quick actions are supported
This issue is confidential and should only be visible to team members with at least Reporter access.
Assignee
Unassigned
Assign to me
Due date
Select due date
Milestone
Milestone
Labels
Labels
Submit issue

Step 4 − Click on the *Submit issue* button and you will get an overview of an issue along with title and description as shown below −



Lab 10 Merging Requests

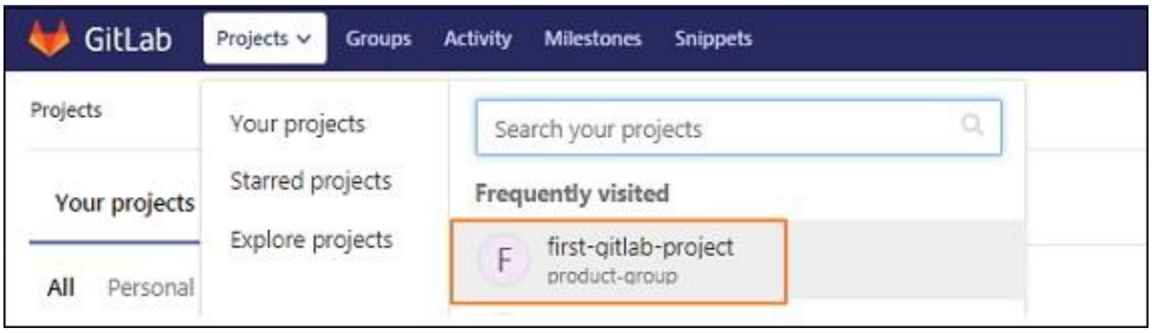
Description

Merge request can be used to interchange the code between other people that you have made to a project and discuss the changes with them easily.

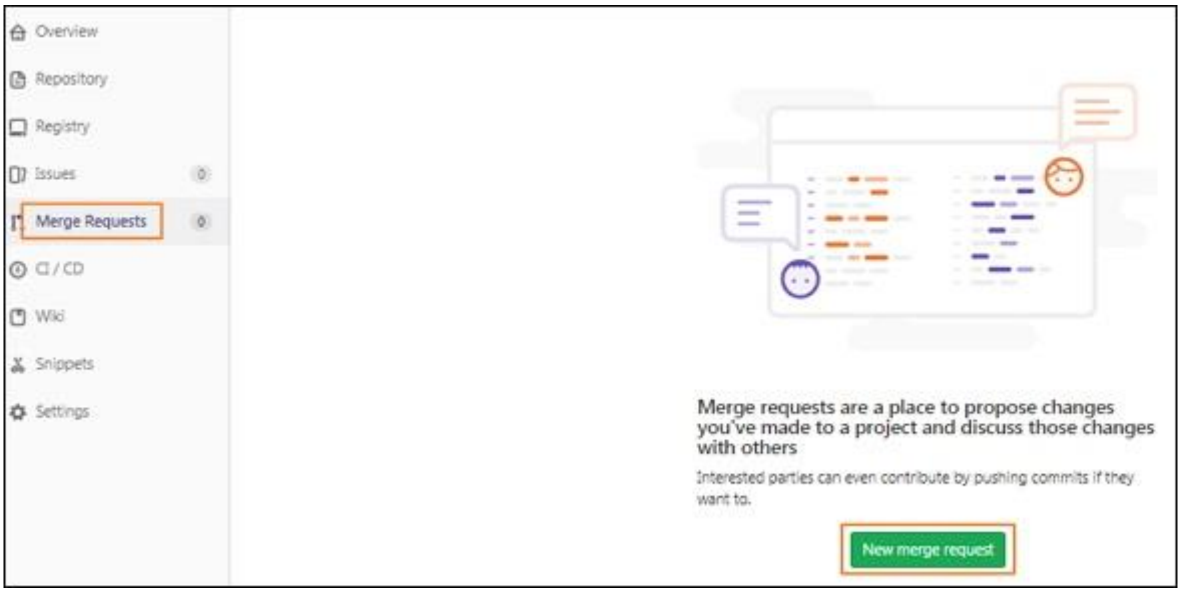
Steps for Merging Request

Step 1 − Before creating new merging request, there should be a created branch in the GitLab. You can refer this chapter for creating the branch −

Step 2 − Login to your GitLab account and go to your project under *Projects* section −



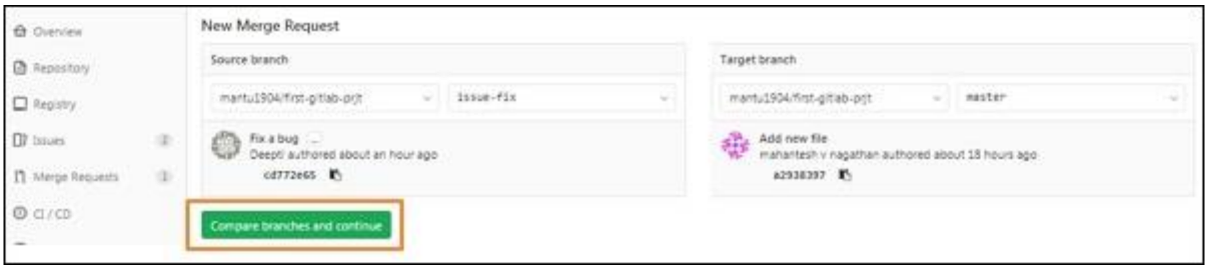
Step 3 − Click on the *Merge Requests* tab and then click on the *New merge request* button −



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.

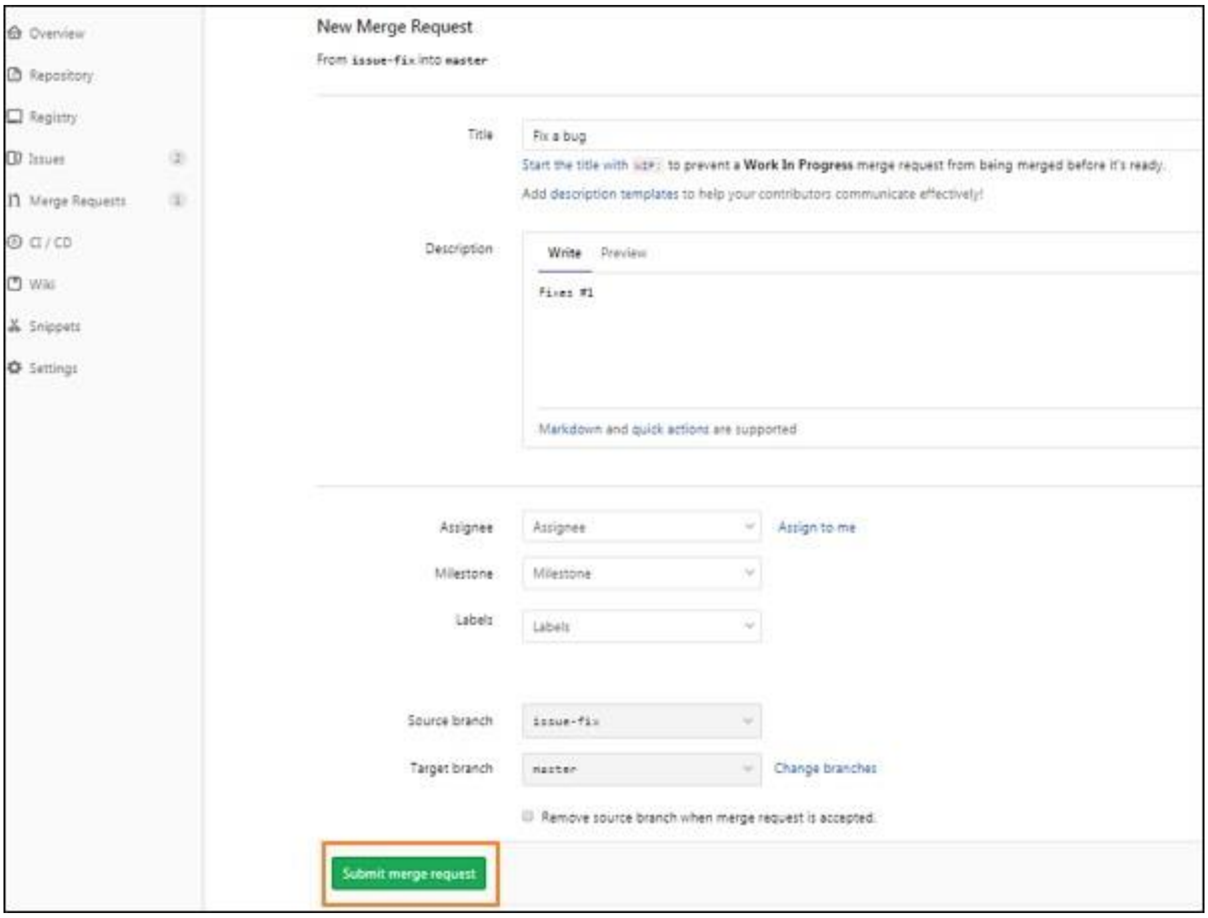
Step 4 − To merge the request, select the source branch and target branch from the dropdown and then click on the *Compare branches and continue* button as shown below −



Fix a bug
Deepti authored about an hour ago
cd772e65

Add new file
mahantesh v nagathan authored about 18 hours ago
a2938397

Step 5 − You will see the title, description and other fields such as assigning user, setting milestone, labels, source branch name and target branch name and click on the *Submit merge request* button −


New Merge Request

From issue-fix into master

Title: Fix a bug

Start the title with WIP: to prevent a Work In Progress merge request from being merged before it's ready.

Add description templates to help your contributors communicate effectively!

Description:

```
Fixes #1
```

Markdown and quick actions are supported

Assignee: Assignee — Assign to me

Milestone: Milestone

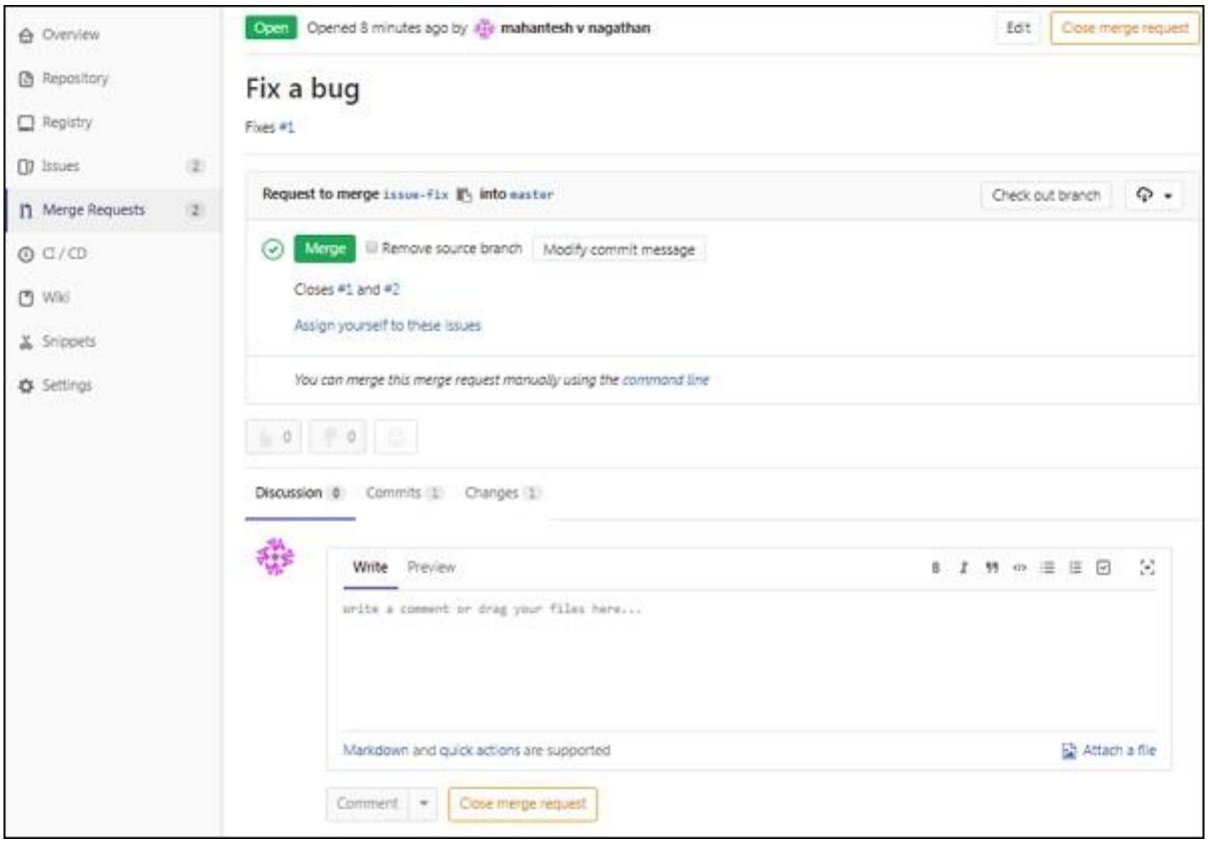
Labels: Labels

Source branch: issue-fix

Target branch: master — Change branches

Remove source branch when merge request is accepted.

Step 6 − After submitting the merge request, you will get a new merge request screen as shown below −


Open — Opened 8 minutes ago by mahantesh v nagathan

Fix a bug

Fixes #1

Request to merge issue-fix into master

Remove source branch

Closes #1 and #2

Assign yourself to these issues

You can merge this merge request manually using the command line

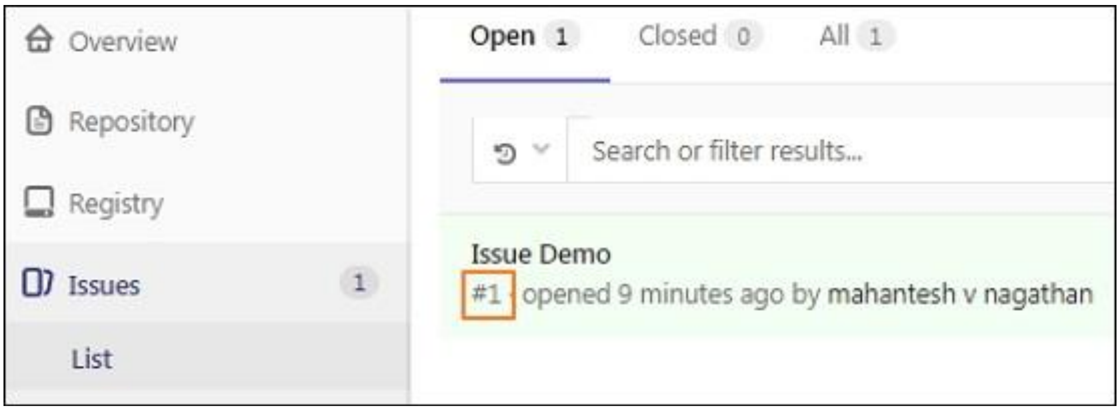
```
Write a comment or drag your files here...
```

Markdown and quick actions are supported

GitLab can be able to refer the specific issue from the commit message to solve a specific problem. In this chapter, we will discuss about how to reference a issue in the GitLab −

Step 1 − To reference a issue, you need to have an issue number of a created issue. To create an issue.

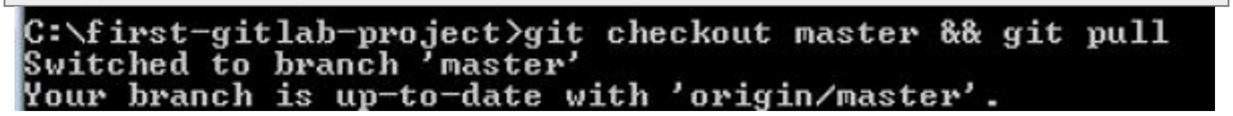
Step 2 − To see the created issue, click on the *List* option under *Issues* tab −



Issue Demo
#1 · opened 9 minutes ago by mahantesh v nagathan

Step 3 − Before making the changes in your local repository, check whether it is up to date or not by using the below command −

```
git checkout master && git pull
```

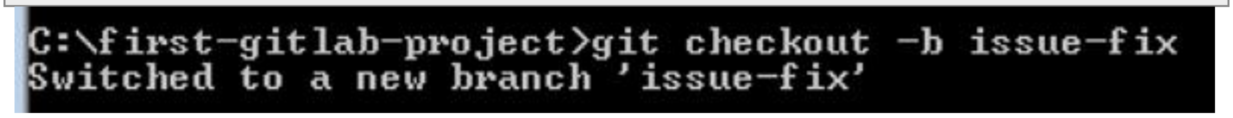


```
C:\first-gitlab-project>git checkout master && git pull
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
```

The *git pull* command downloads the latest changes from the remote server and integrates directly into current working files.

Step 4 − Now, create a new branch with the name *issue-fix* by using the *git checkout* command −

```
git checkout -b issue-fix
```



```
C:\first-gitlab-project>git checkout -b issue-fix
Switched to a new branch 'issue-fix'
```

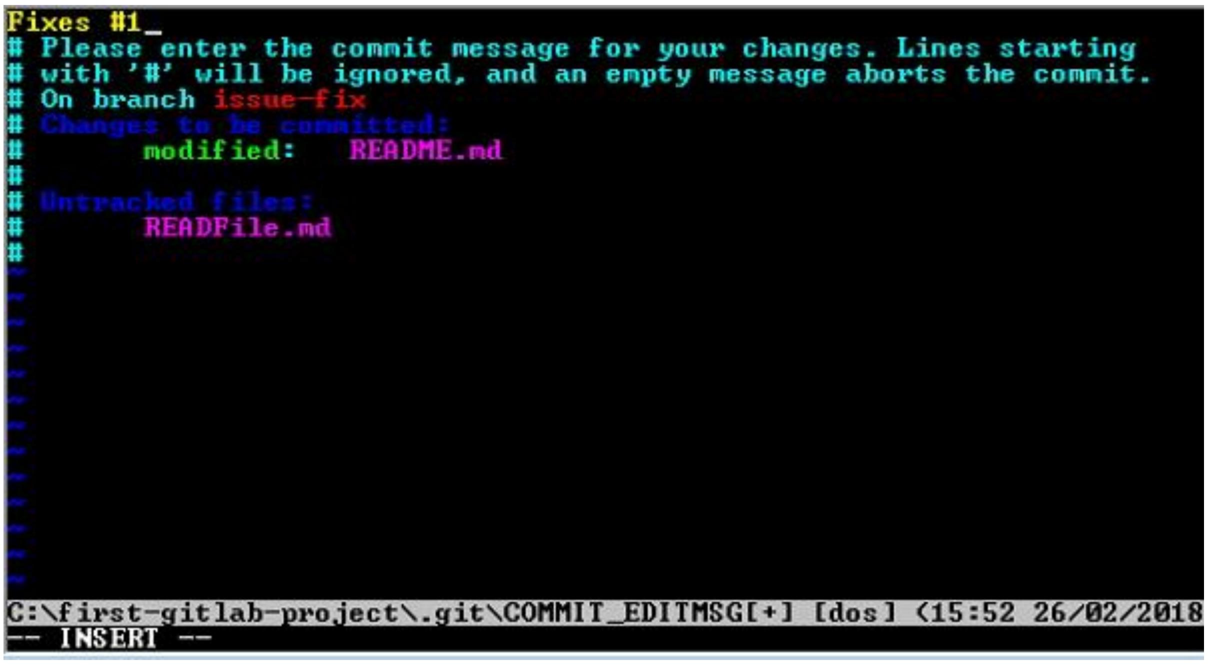
Step 5 − Now, add some content to the *README.md* file to fix the bug −

```
echo "fix this bug" >> README.md
```

Step 6 − Enter the commit message for the above change with the below command −

```
git commit -a
```

This command opens the below screen and press *Insert* key on the keyboard to add a commit message for the *issue-fix* branch.

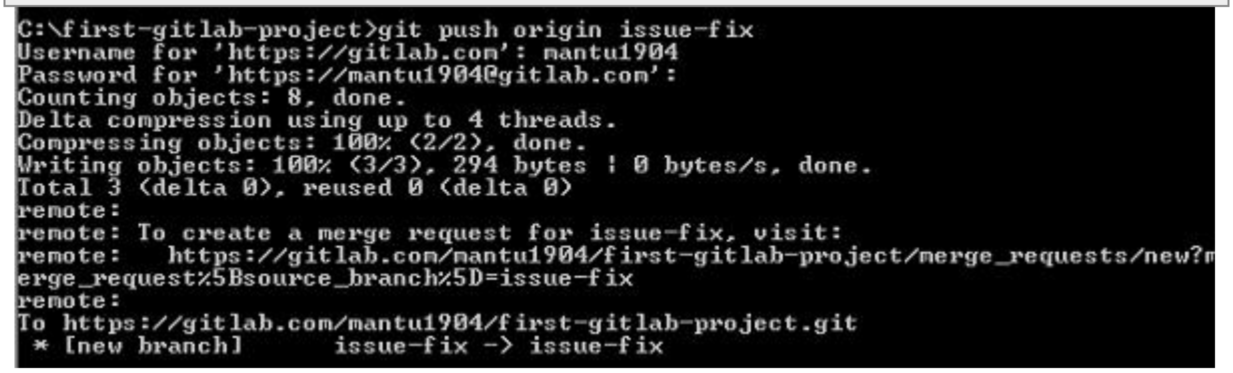


```
Fixes #1_
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch issue-fix
# Changes to be committed:
#       modified:   README.md
#
# Untracked files:
#       READFile.md
#
~
~
~
~
~
~
~
~
~
~
~
~
C:\first-gitlab-project\.git\COMMIT_EDITMSG[+] [dos] (15:52 26/02/2018
-- INSERT --
```

Now press the *Esc* key, then colon(:) and type *wq* to save and exit from the screen.

Step 7 − Now push the branch to remote repository by using the below command −

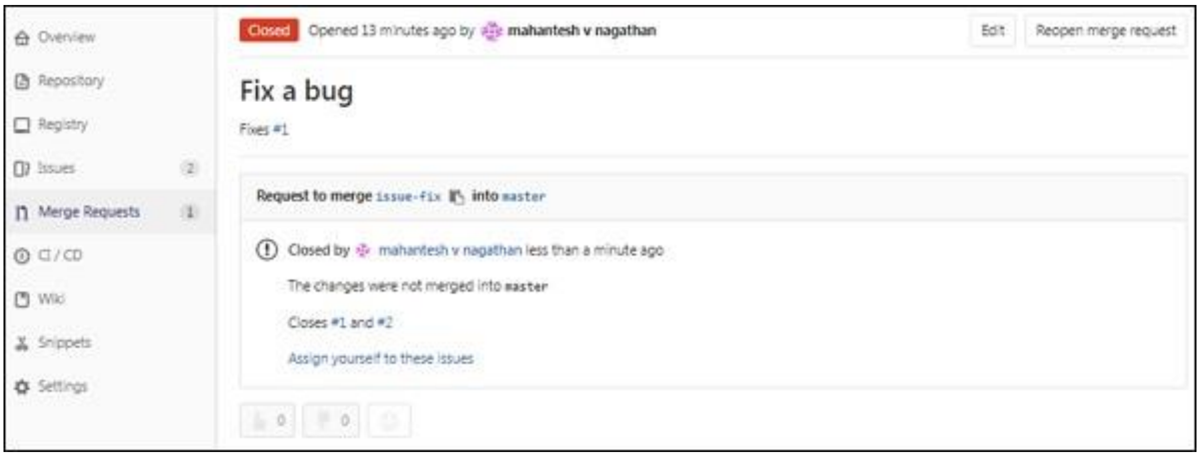
```
git push origin issue-fix
```



```
C:\first-gitlab-project>git push origin issue-fix
Username for 'https://gitlab.com': mantu1904
Password for 'https://mantu1904@gitlab.com':
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 294 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for issue-fix, visit:
remote:   https://gitlab.com/mantu1904/first-gitlab-project/merge_requests/new?m
erge_request%5Bsource_branch%5D=issue-fix
remote:
To https://gitlab.com/mantu1904/first-gitlab-project.git
 * [new branch]      issue-fix -> issue-fix
```

Step 8 − Login to your GitLab account and create a new merge request. You can refer the merge request chapter for the creation of merge request.

Step 9 − Once you create the merge request, you will be redirected to the merge request page. When you click on the *Close merge request* button (refer the screenshot in the step (6) of merge request chapter), you will see the Closed option after closing merge request.



Opened 13 minutes ago by mahantesh v nagathan

Fix a bug

Fixes #1

Request to merge issue-fix into master

Closed by mahantesh v nagathan less than a minute ago

The changes were not merged into master

Closes #1 and #2

Assign yourself to these issues

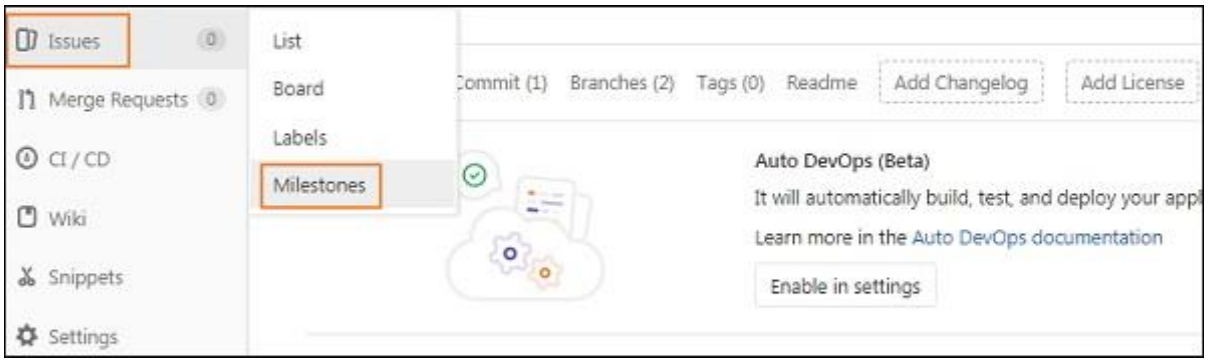
Lab 11 Creating Milestones

Description

Milestones are used for arranging issues and merge requests into a determined group which can achieved within a specified amount of time by setting a start and due date.

Steps for Creating Milestones

Step 1 − Login to your GitLab account, go to your project and click on the *Milestones* option under *Issues* tab −

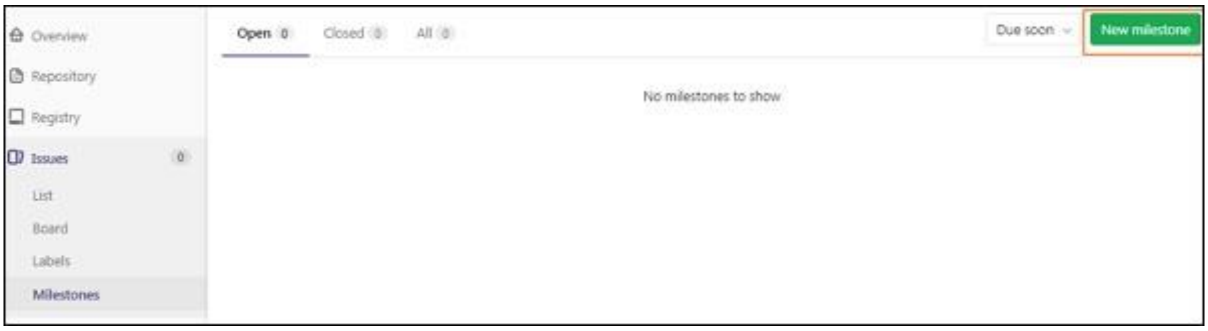


Auto DevOps (Beta)

It will automatically build, test, and deploy your appl

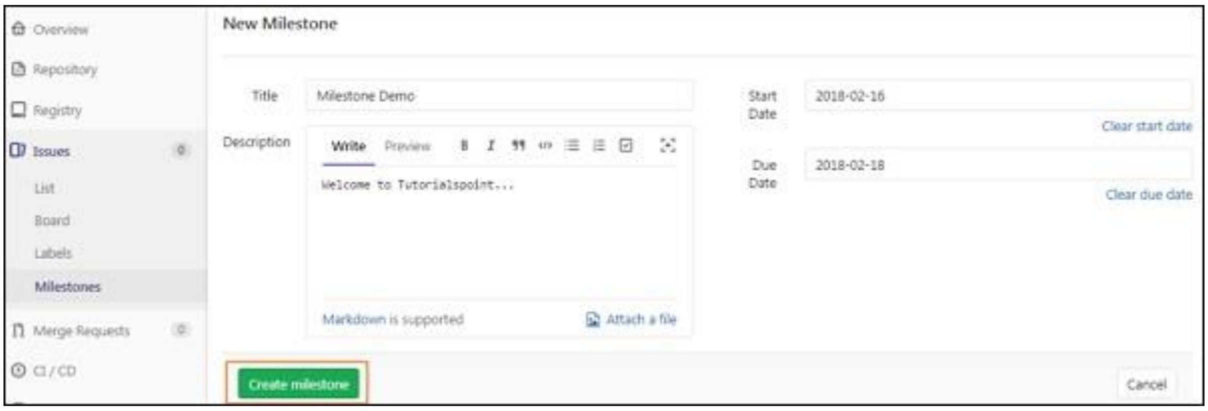
Learn more in the Auto DevOps documentation

Step 2 − Click on the *New milestone* button −



No milestones to show

Step 3 − Now enter the title, description, start and due date and click on *Create milestone* button as shown in the below image −



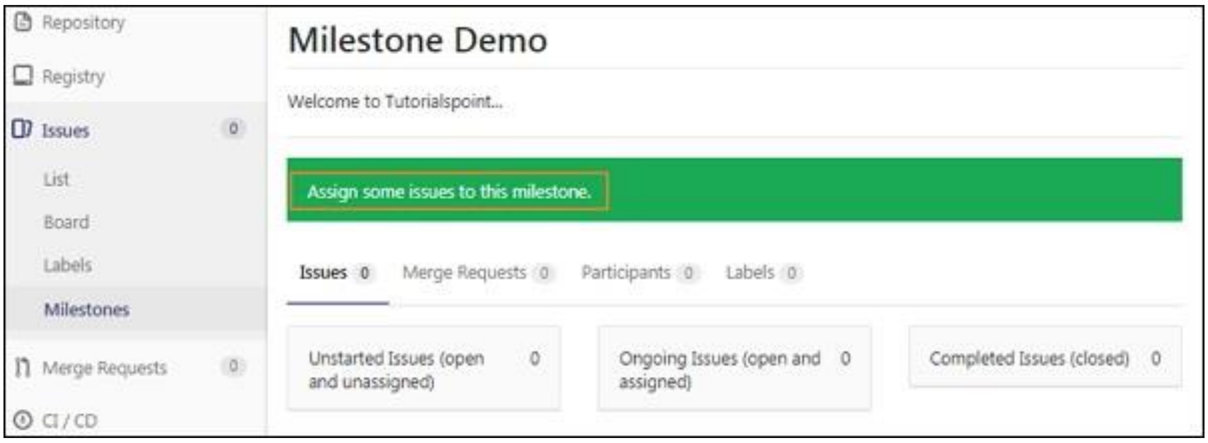
Title: Milestone Demo

Description: Welcome to Tutorialspoint...

Start Date: 2018-02-16

Due Date: 2018-02-18

Step 4 − After creating a milestone, it will display a message saying 'Assign some issues to this milestone' as shown below −



Milestone Demo

Welcome to Tutorialspoint...

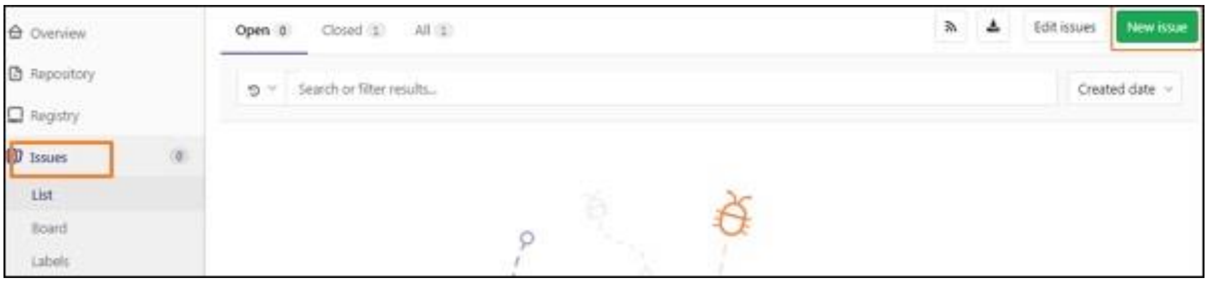
Assign some issues to this milestone.

Unstarted Issues (open and unassigned) 0

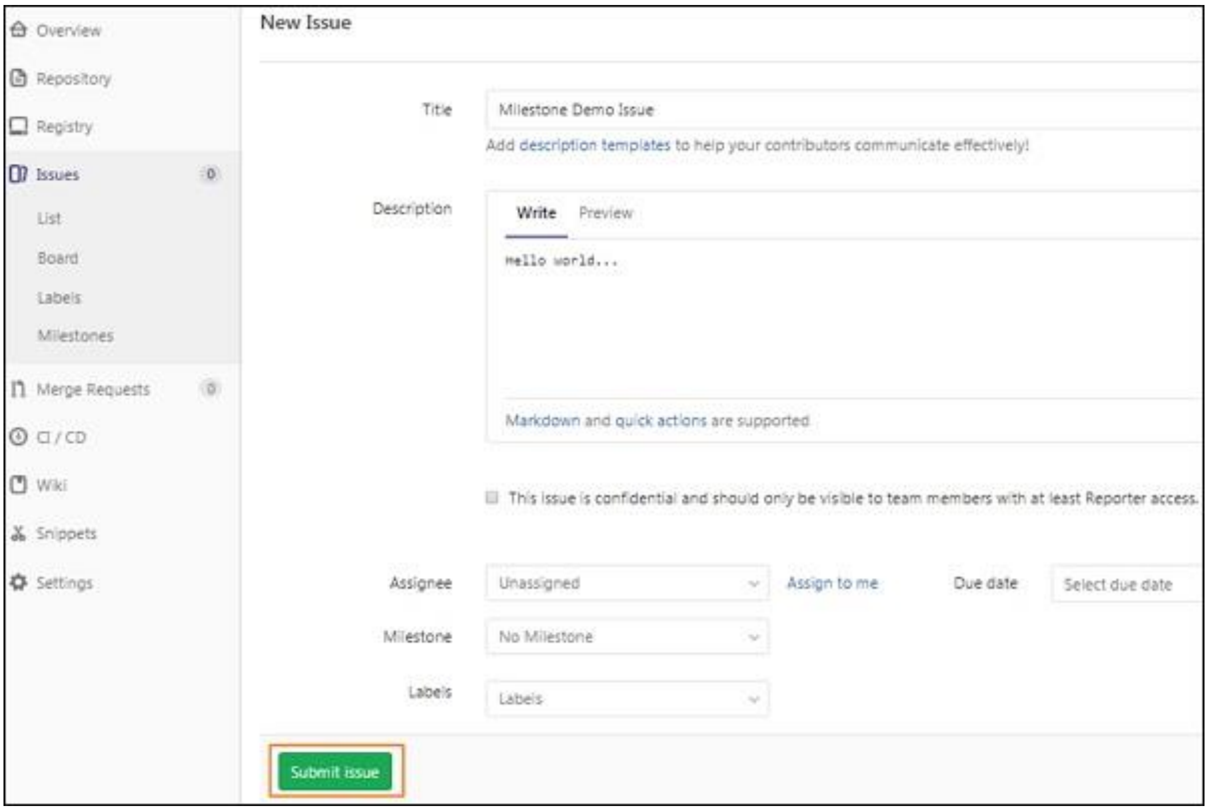
Ongoing Issues (open and assigned) 0

Completed Issues (closed) 0

Step 5 − Now go to *Issues* tab and click on the *New issue* button to create an issue for the milestone −



Step 6 − Now, fill the information such as title, description and if you want, you can select a user to assign an issue, milestone, labels upon operation or could be choose by developers themselves later. Click on the *Submit issue* button.


New Issue

Title: Milestone Demo Issue

Add description templates to help your contributors communicate effectively!

Description: Hello world...

Markdown and quick actions are supported

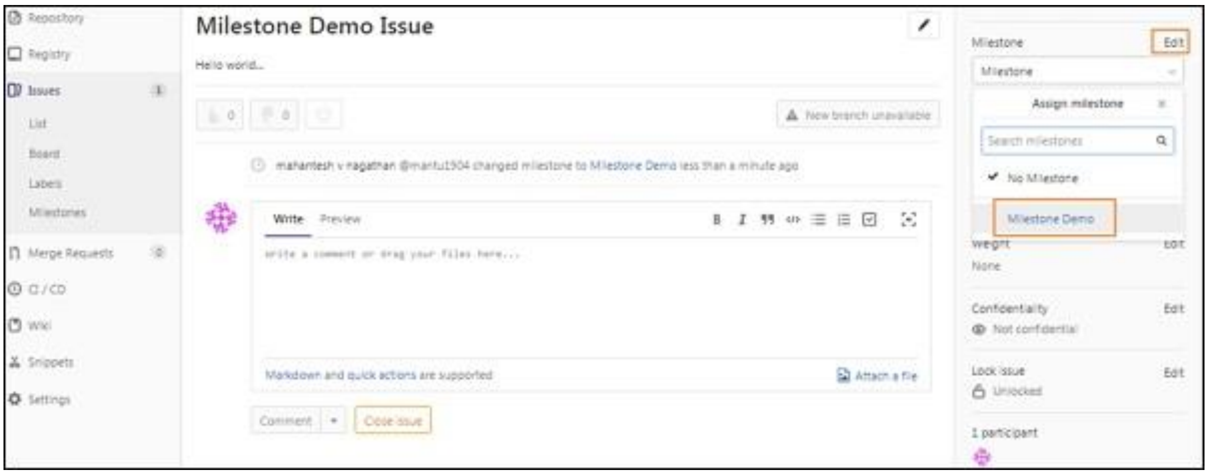
This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee: Unassigned — Assign to me — Due date: Select due date

Milestone: No Milestone

Labels: Labels

Step 7 − After creating a issue, you will get overview of an issue along with title and description. At right side, click on *Edit* option and assign milestone for the issue under *Milestone* section −


Milestone Demo Issue

Hello world...

mahantesh v nagathan @mantu1904 changed milestone to Milestone Demo less than a minute ago

Milestone: Milestone

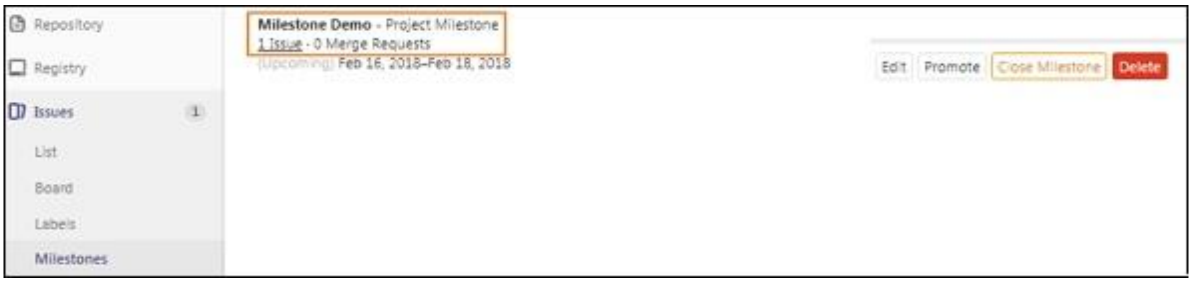
Weight: None

Confidentiality: Not confidential

Lock issue: Unlocked

1 participant

Step 8 − Now go back to Milestones section and you will see the added milestone along with created issue −


Milestone Demo - Project Milestone

1 Issue · 0 Merge Requests

(Upcoming) Feb 16, 2018–Feb 18, 2018

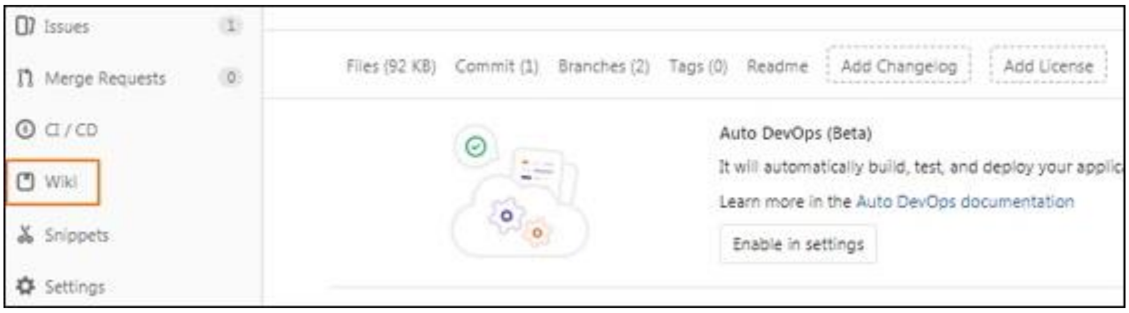
Lab 12 Creating Wiki Page

Description

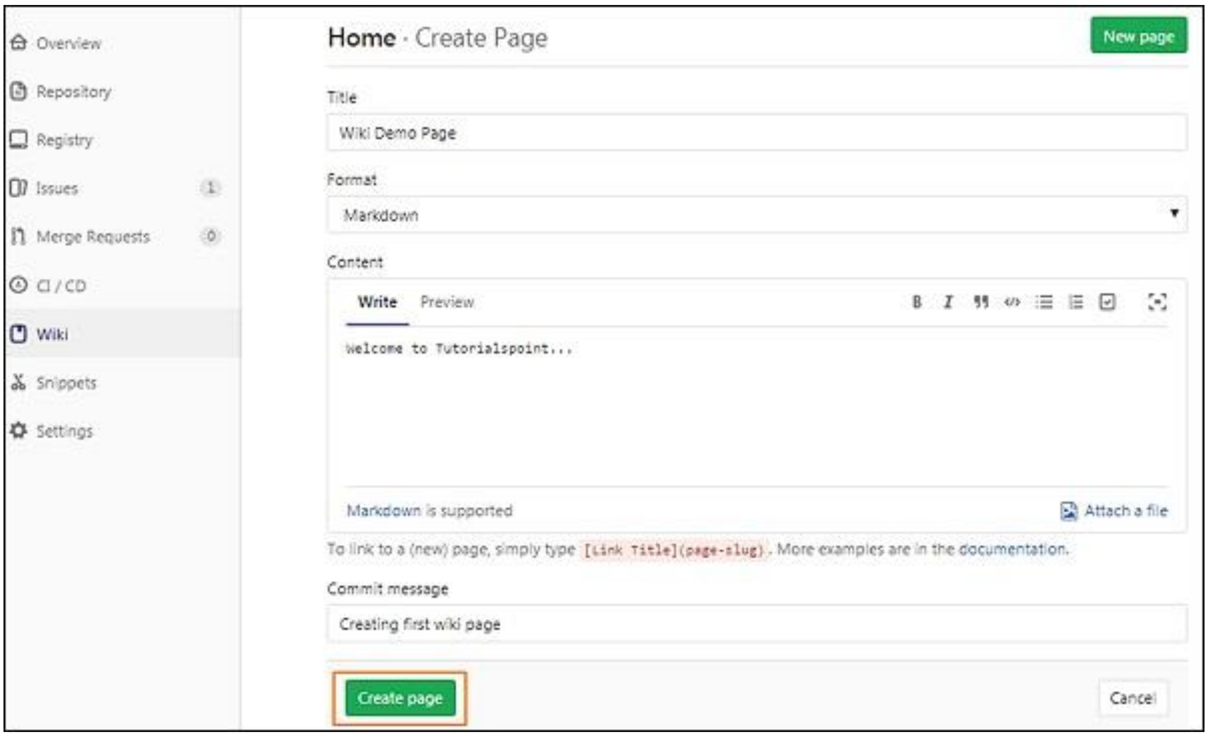
Wiki is a system for maintaining documentation for a project in the GitLab. It is like a Wikipedia which can be editable and given permissions to manage the wiki pages. A *Guest* can view a wiki page and Developer can create and edit a wiki page.

Steps for Creating Wiki Page

Step 1 − Login to your GitLab account, go to your project and click on the *Wiki* tab −


Issues 1
Merge Requests 0
CI / CD
Wiki
Snippets
Settings
Files (92 KB) Commit (1) Branches (2) Tags (0) Readme Add Changelog Add License
Auto DevOps (Beta)
It will automatically build, test, and deploy your applic
Learn more in the Auto DevOps documentation
Enable in settings

Step 2 − Now enter the title, format, fill the content section, add a commit message and then click on the *Create page* button −



Title
Wiki Demo Page

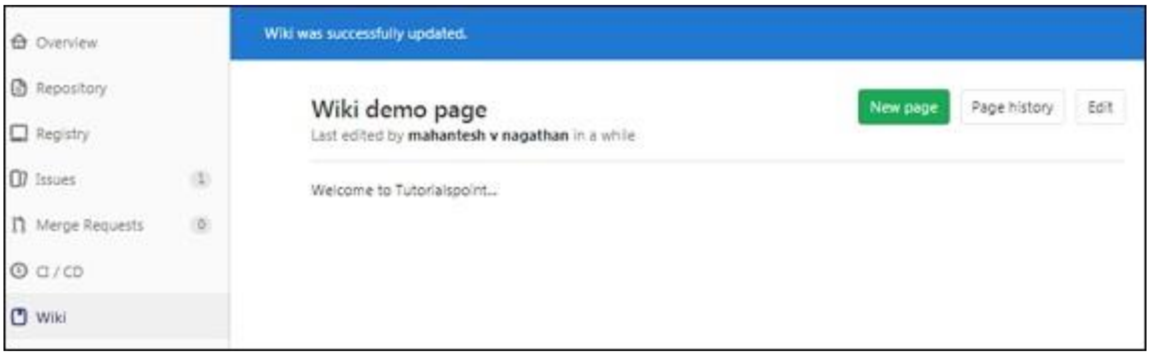
Format
Markdown

Content
welcome to Tutorialspoint...

To link to a (new) page, simply type `[Link Title](page-slug)`. More examples are in the documentation.

Commit message
Creating first wiki page

Step 3 − You will get newly created wiki page as shown in the below image −



Wiki was successfully updated.

Wiki demo page

Last edited by mahantesh v nagathan in a while

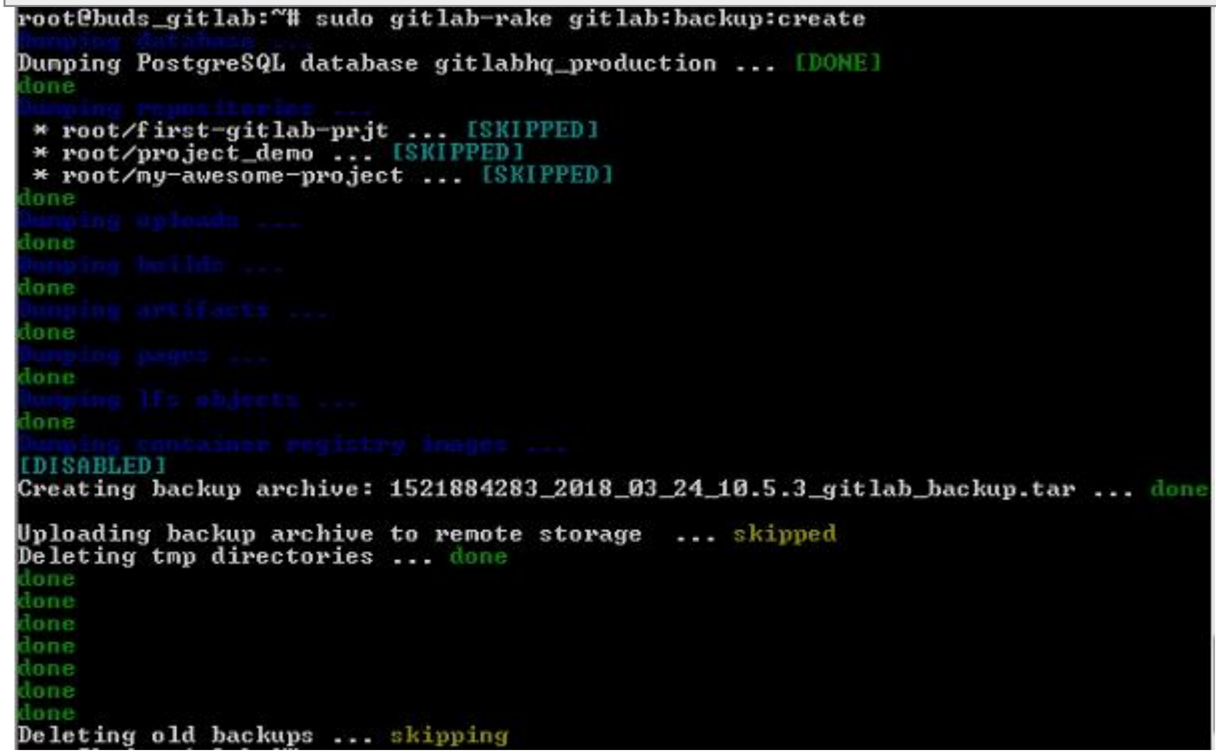
Welcome to Tutorialspoint...

GitLab allows to take backup copy of your repository by using simple command. In this chapter, we will discuss about how to take backup copy in the GitLab −

Step 1 − First, login to your GitLab server using SSH (Secure Shell).

Step 2 − Create the backup of GitLab by using the below command −

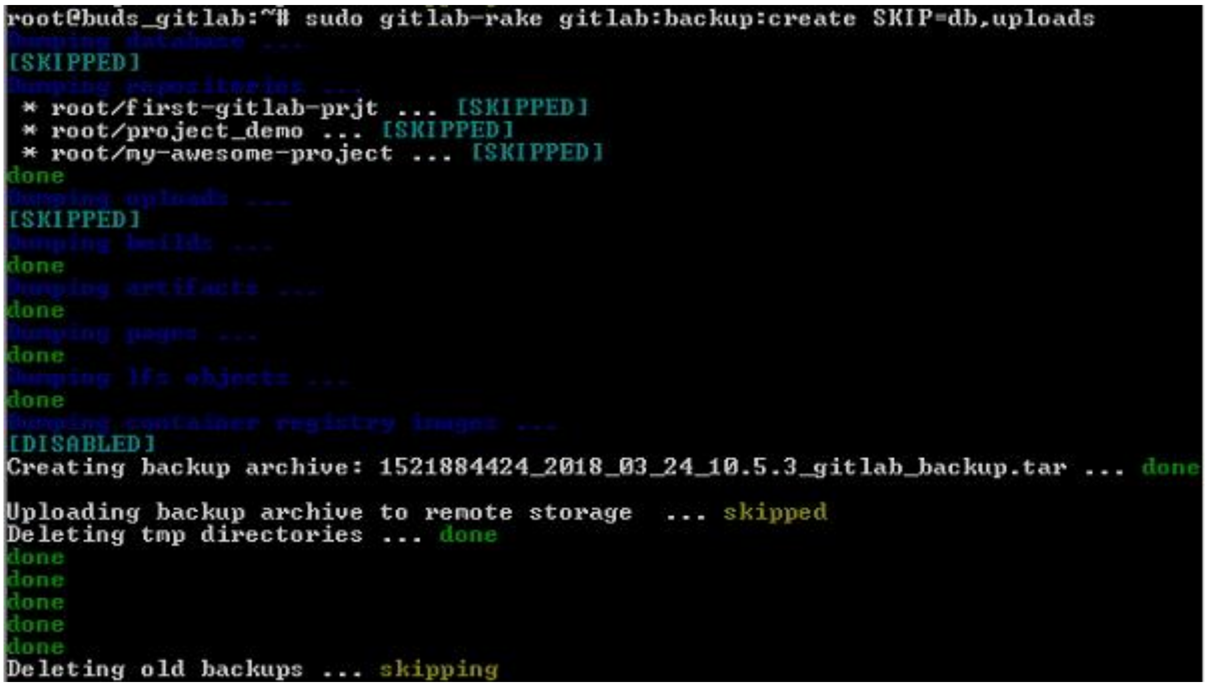
```
sudo gitlab-rake gitlab:backup:create
```



```
root@buds_gitlab:~# sudo gitlab-rake gitlab:backup:create
Dumping database ...
Dumping PostgreSQL database gitlabhq_production ... [DONE]
done
Dumping repositories ...
 * root/first-gitlab-prjt ... [SKIPPED]
 * root/project_demo ... [SKIPPED]
 * root/my-awesome-project ... [SKIPPED]
done
Dumping uploads ...
done
Dumping builds ...
done
Dumping artifacts ...
done
Dumping pages ...
done
Dumping lfs objects ...
done
Dumping container registry images ...
[DISABLED]
Creating backup archive: 1521884283_2018_03_24_10.5.3_gitlab_backup.tar ... done

Uploading backup archive to remote storage  ... skipped
Deleting tmp directories ... done
done
done
done
done
done
done
done
Deleting old backups ... skipping
```

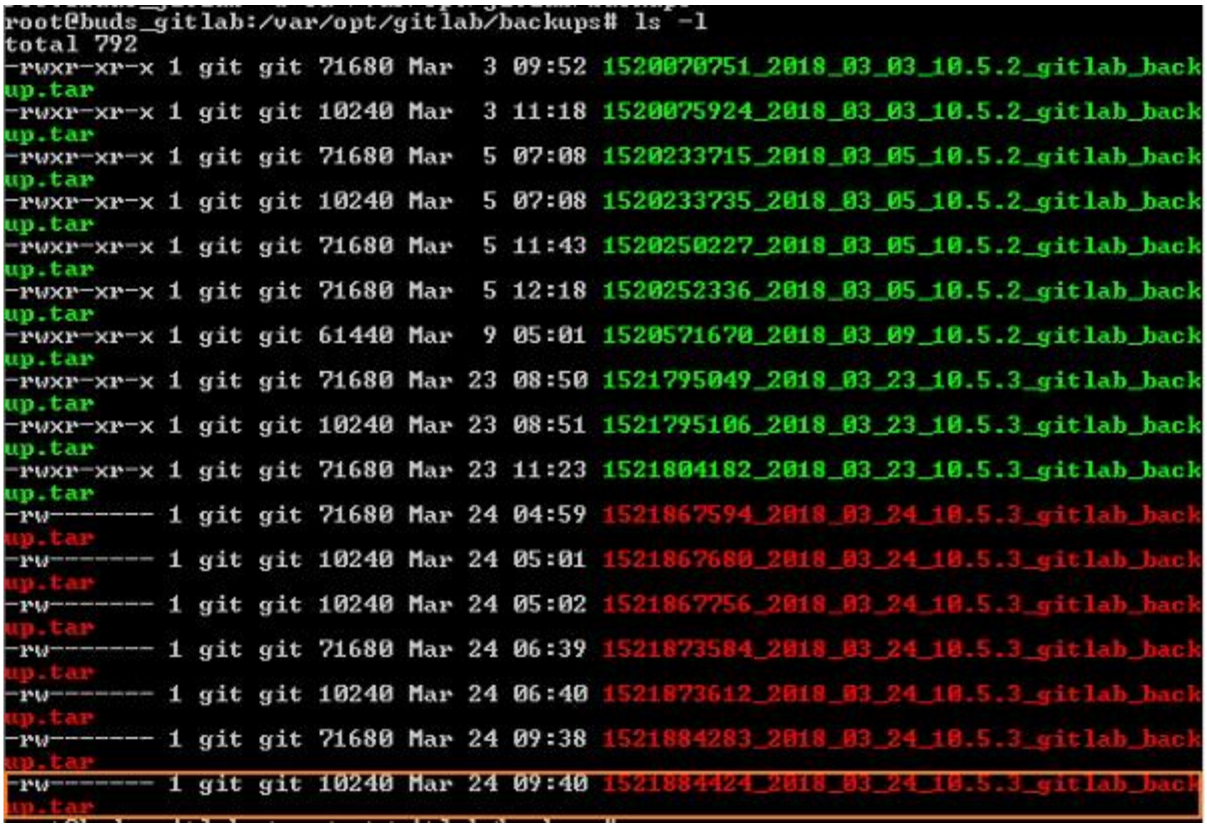
Step 3 − You can exclude some directories from the backup by adding environment variable *SKIP* as shown below −

```
sudo gitlab-rake gitlab:backup:create SKIP = db,uploads
```



```
root@buds_gitlab:~# sudo gitlab-rake gitlab:backup:create SKIP=db,uploads
Dumping database ...
[SKIPPED]
Dumping repositories ...
 * root/first-gitlab-prjt ... [SKIPPED]
 * root/project_demo ... [SKIPPED]
 * root/my-awesome-project ... [SKIPPED]
done
Dumping uploads ...
[SKIPPED]
Dumping builds ...
done
Dumping artifacts ...
done
Dumping pages ...
done
Dumping lfs objects ...
done
Dumping container registry images ...
[DISABLED]
Creating backup archive: 1521884424_2018_03_24_10.5.3_gitlab_backup.tar ... done

Uploading backup archive to remote storage  ... skipped
Deleting tmp directories ... done
done
done
done
done
done
Deleting old backups ... skipping
```

Step 4 − The backup tar file will get created in the default */var/opt/gitlab/backups* directory. Navigate to this path and type *ls -l* to see the created backup file −



```
root@buds_gitlab:/var/opt/gitlab/backups# ls -l
total 792
-rwxr-xr-x 1 git git 71680 Mar  3 09:52 1520070751_2018_03_03_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 10240 Mar  3 11:18 1520075924_2018_03_03_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 71680 Mar  5 07:08 1520233715_2018_03_05_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 10240 Mar  5 07:08 1520233735_2018_03_05_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 71680 Mar  5 11:43 1520250227_2018_03_05_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 71680 Mar  5 12:18 1520252336_2018_03_05_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 61440 Mar  9 05:01 1520571670_2018_03_09_10.5.2_gitlab_back
up.tar
-rwxr-xr-x 1 git git 71680 Mar 23 08:50 1521795049_2018_03_23_10.5.3_gitlab_back
up.tar
-rwxr-xr-x 1 git git 10240 Mar 23 08:51 1521795106_2018_03_23_10.5.3_gitlab_back
up.tar
-rwxr-xr-x 1 git git 71680 Mar 23 11:23 1521804182_2018_03_23_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 71680 Mar 24 04:59 1521867594_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 10240 Mar 24 05:01 1521867680_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 10240 Mar 24 05:02 1521867756_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 71680 Mar 24 06:39 1521873584_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 10240 Mar 24 06:40 1521873612_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 71680 Mar 24 09:38 1521884283_2018_03_24_10.5.3_gitlab_back
up.tar
-rw------- 1 git git 10240 Mar 24 09:40 1521884424_2018_03_24_10.5.3_gitlab_back
up.tar
```

GitLab allows restoring the backup copy of your repository. In this chapter, we will discuss about how to restore the backup copy in the GitLab −

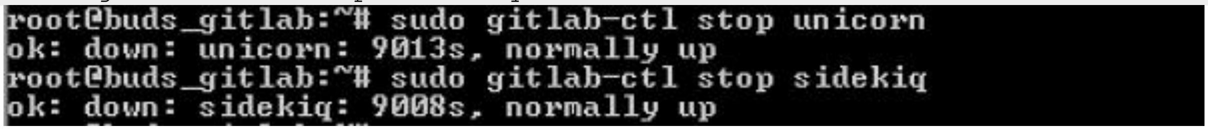
Step 1 − First, login to your GitLab server using SSH (Secure Shell).

Step 2 − Before restoring the backup copy, first make sure backup copy is in the */var/opt/gitlab/backups* directory.

Step 3 − You can check the backup copy by using the *ls -l* command which is described in the Create Backup job chapter.

Step 4 − Now, stop the processes which are related to the database by using the below commands −

```
sudo gitlab-ctl stop unicorn

sudo gitlab-ctl stop sidekiq
```



```
root@buds_gitlab:~# sudo gitlab-ctl stop unicorn
ok: down: unicorn: 9013s, normally up
root@buds_gitlab:~# sudo gitlab-ctl stop sidekiq
ok: down: sidekiq: 9008s, normally up
```

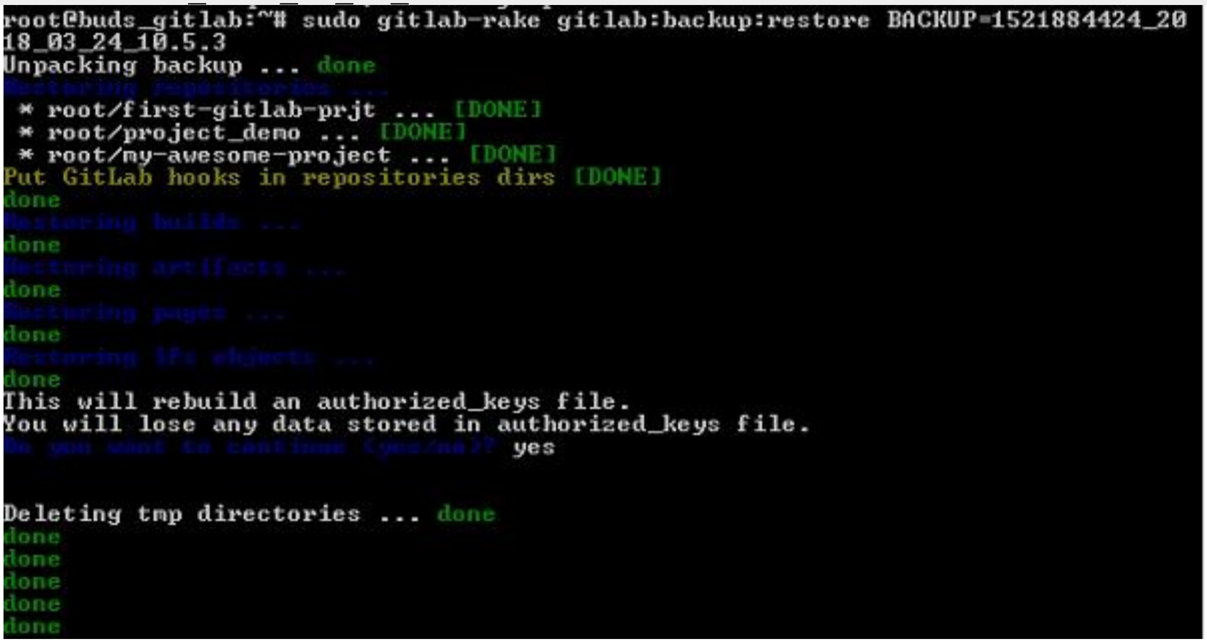
The above commands can also be used to free up some memory temporarily by shutting down them.

Step 5 − You can verify status of the GitLab services by using the below command −

```
sudo gitlab-ctl status
```

Step 6 − Now, restore the backup by using the timestamp of the backup copy −

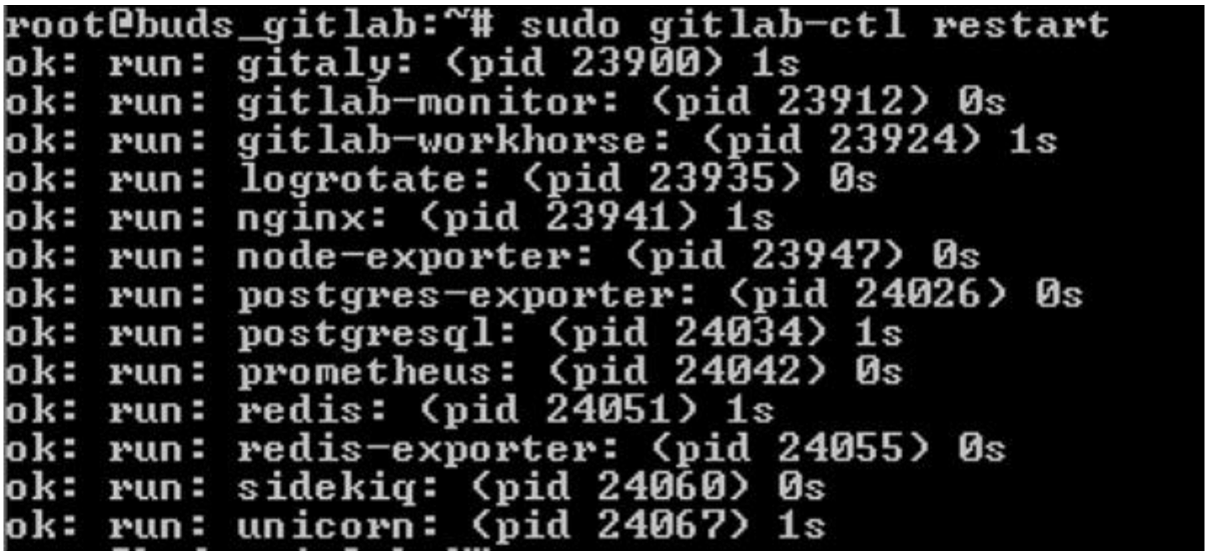
```
sudo gitlab-rake gitlab:backup:restore BACKUP =
1521884424_2018_03_24_10.5.3
```



```
root@buds_gitlab:~# sudo gitlab-rake gitlab:backup:restore BACKUP=1521884424_20
18_03_24_10.5.3
Unpacking backup ... done
Restoring repositories ...
 * root/first-gitlab-prjt ... [DONE]
 * root/project_demo ... [DONE]
 * root/my-awesome-project ... [DONE]
Put GitLab hooks in repositories dirs [DONE]
done
Restoring builds ...
done
Restoring artifacts ...
done
Restoring pages ...
done
Restoring lfs objects ...
done
This will rebuild an authorized_keys file.
You will lose any data stored in authorized_keys file.
Do you want to continue (yes/no)? yes


Deleting tmp directories ... done
done
done
done
done
done
```

Step 7 − Restart the GitLab components by using the below command −

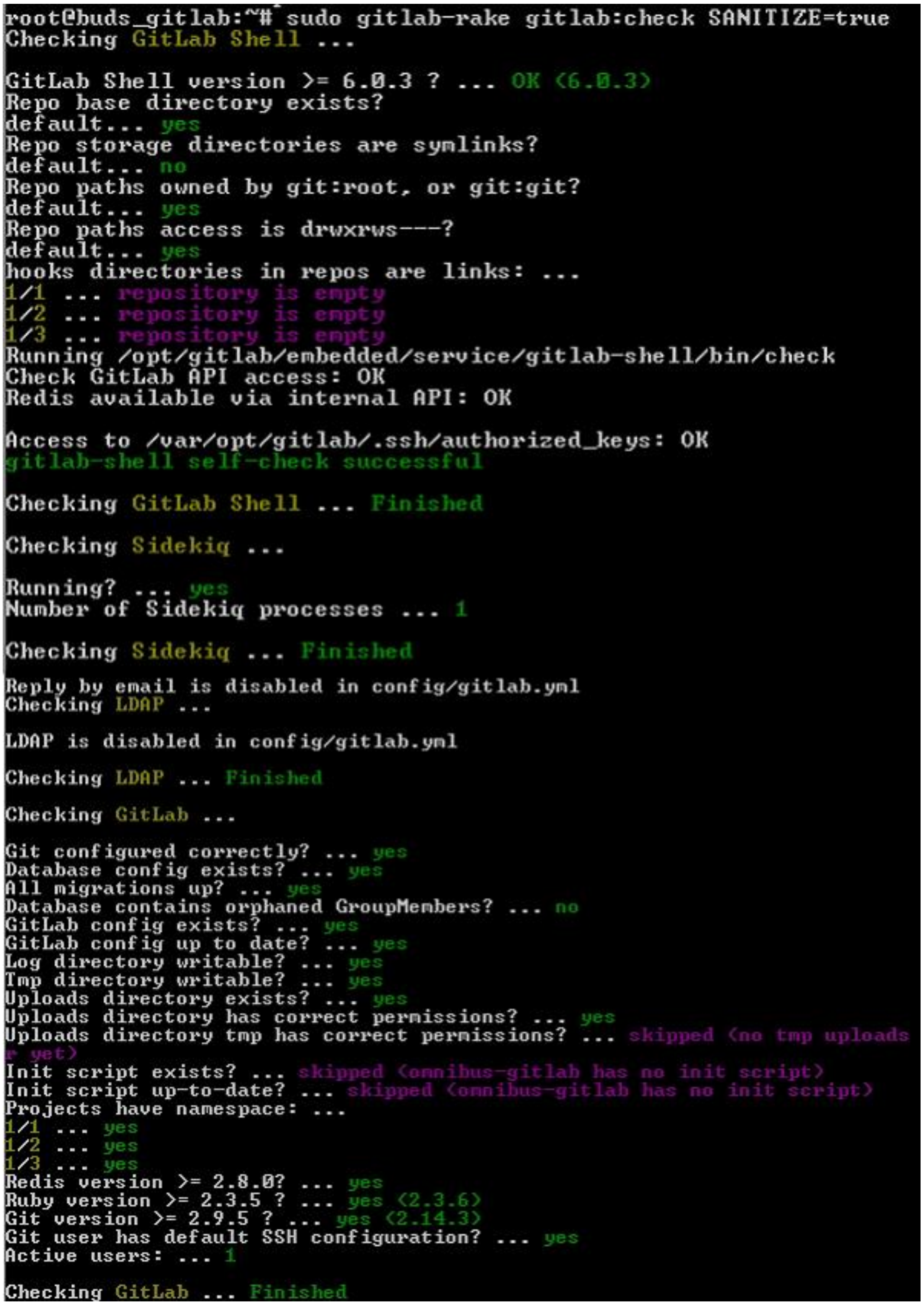
```
sudo gitlab-ctl restart
```



```
root@buds_gitlab:~# sudo gitlab-ctl restart
ok: run: gitaly: (pid 23900) 1s
ok: run: gitlab-monitor: (pid 23912) 0s
ok: run: gitlab-workhorse: (pid 23924) 1s
ok: run: logrotate: (pid 23935) 0s
ok: run: nginx: (pid 23941) 1s
ok: run: node-exporter: (pid 23947) 0s
ok: run: postgres-exporter: (pid 24026) 0s
ok: run: postgresql: (pid 24034) 1s
ok: run: prometheus: (pid 24042) 0s
ok: run: redis: (pid 24051) 1s
ok: run: redis-exporter: (pid 24055) 0s
ok: run: sidekiq: (pid 24060) 0s
ok: run: unicorn: (pid 24067) 1s
```

Step 8 − Now check the GitLab by sanitizing the database as shown below −

```
sudo gitlab-rake gitlab:check SANITIZE = true
```

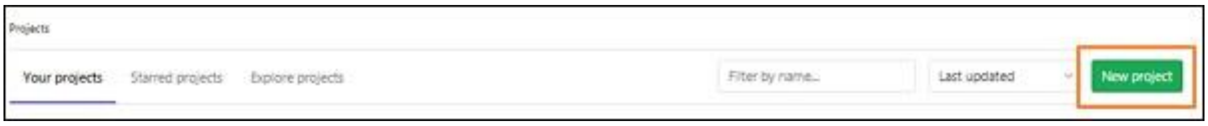


```
root@buds_gitlab:~# sudo gitlab-rake gitlab:check SANITIZE=true
Checking GitLab Shell ...

GitLab Shell version >= 6.0.3 ? ... OK (6.0.3)
Repo base directory exists?
default... yes
Repo storage directories are symlinks?
default... no
Repo paths owned by git:root, or git:git?
default... yes
Repo paths access is drwxrws---?
default... yes
hooks directories in repos are links: ...
1/1 ... repository is empty
1/2 ... repository is empty
1/3 ... repository is empty
Running /opt/gitlab/embedded/service/gitlab-shell/bin/check
Check GitLab API access: OK
Redis available via internal API: OK

Access to /var/opt/gitlab/.ssh/authorized_keys: OK
gitlab-shell self-check successful

Checking GitLab Shell ... Finished

Checking Sidekiq ...

Running? ... yes
Number of Sidekiq processes ... 1

Checking Sidekiq ... Finished

Reply by email is disabled in config/gitlab.yml
Checking LDAP ...

LDAP is disabled in config/gitlab.yml

Checking LDAP ... Finished

Checking GitLab ...

Git configured correctly? ... yes
Database config exists? ... yes
All migrations up? ... yes
Database contains orphaned GroupMembers? ... no
GitLab config exists? ... yes
GitLab config up to date? ... yes
Log directory writable? ... yes
Tmp directory writable? ... yes
Uploads directory exists? ... yes
Uploads directory has correct permissions? ... yes
Uploads directory tmp has correct permissions? ... skipped (no tmp uploads
r yet)
Init script exists? ... skipped (omnibus-gitlab has no init script)
Init script up-to-date? ... skipped (omnibus-gitlab has no init script)
Projects have namespace: ...
1/1 ... yes
1/2 ... yes
1/3 ... yes
Redis version >= 2.8.0? ... yes
Ruby version >= 2.3.5 ? ... yes (2.3.6)
Git version >= 2.9.5 ? ... yes (2.14.3)
Git user has default SSH configuration? ... yes
Active users: ... 1

Checking GitLab ... Finished
```

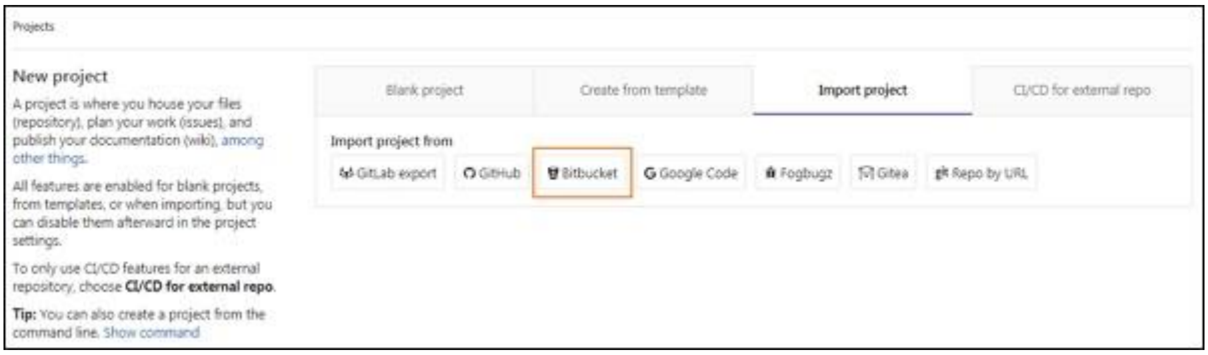
The *SANITIZE = true* flag removes all email addresses because they are confidential, removes the CI variables and access tokens as they can be used in the production instance.

we will discuss about how to import a repository from Bitbucket to GitLab −

Step 1 − Login to your GitLab account and click on the *New project* button in the dashboard −



Step 2 − Click on the *Bitbucket* button under *Import project* tab −


Projects

New project

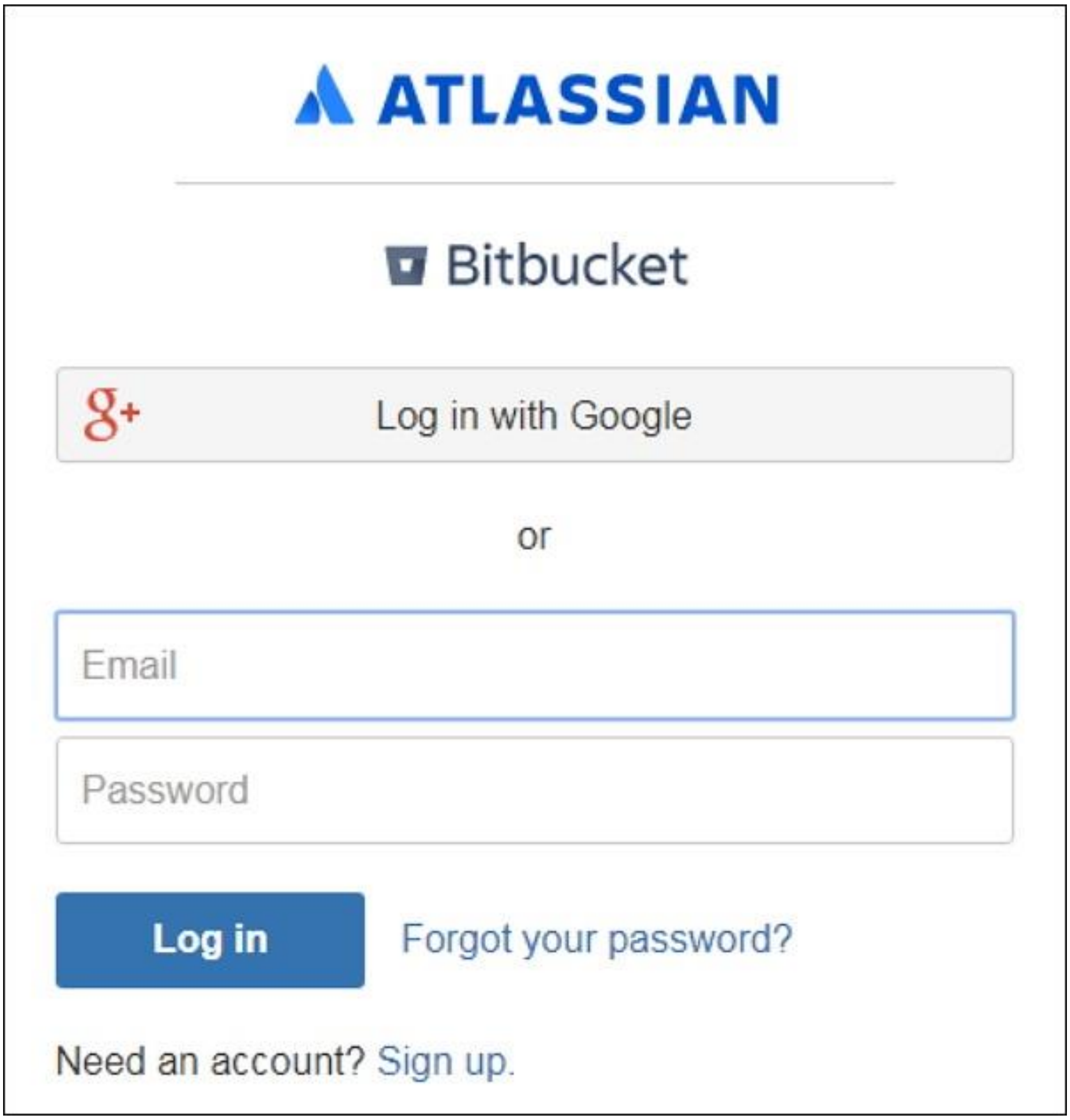
A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

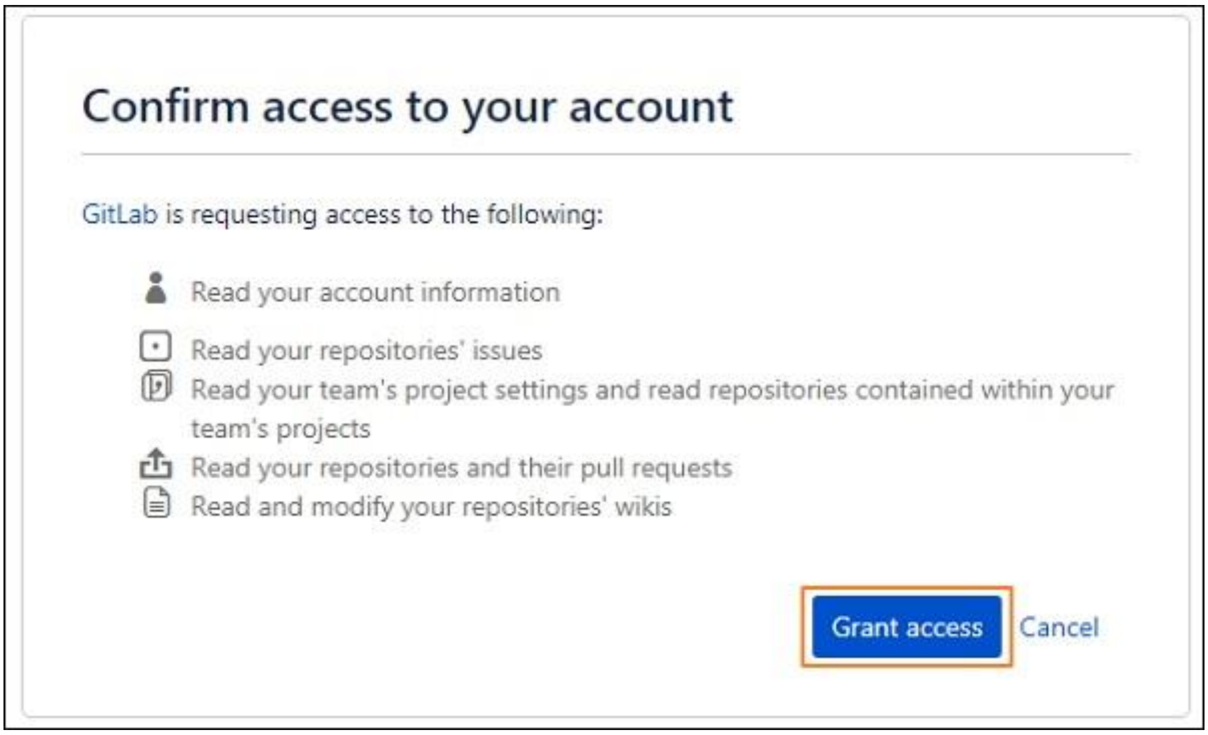
To only use CI/CD features for an external repository, choose **CI/CD for external repo**.

Tip: You can also create a project from the command line. Show command

Step 3 − Next, you need to login to your Bitbucket account. If you don't have an account, then create a new account by clicking on *Sign up* link and then login to Bitbucket account.



Step 4 − When you click on the *Bitbucket* button (shown in step 2), it will display the below screen and click on the *Grant access* button −



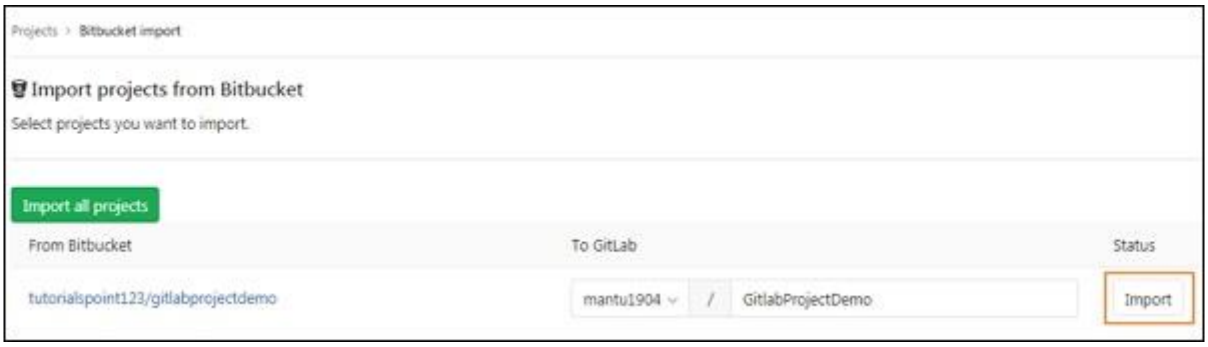
Confirm access to your account

GitLab is requesting access to the following:

- Read your account information
- Read your repositories' issues
- Read your team's project settings and read repositories contained within your team's projects
- Read your repositories and their pull requests
- Read and modify your repositories' wikis

You need to grant the access to read the account information, repository issues, project settings, and modify the repositories.

Step 5 − Click on the *Import* button to import the project from Bitbucket −

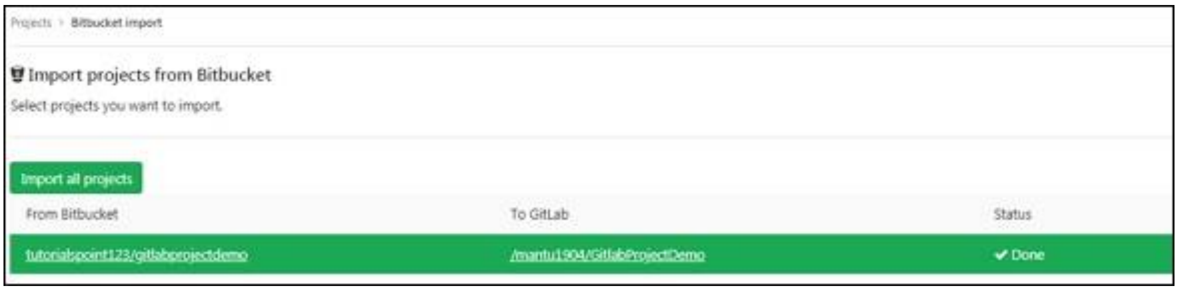


Import projects from Bitbucket

Select projects you want to import.

From Bitbucket	To GitLab	Status
tutorialspoint123/gitlabprojectdemo	mantu1904 / GitlabProjectDemo	Import

Step 6 − After importing the project successfully, it will display the status as *Done* −



Import projects from Bitbucket

Select projects you want to import.

From Bitbucket	To GitLab	Status
tutorialspoint123/gitlabprojectdemo	/mantu1904/GitlabProjectDemo	✓ Done

Lab 13: GitLab CI Service

Description

GitLab CI (Continuous Integration) service is a part of GitLab which manages the project and user interface and allows unit tests on every commit and indicates with warning message when there is an unsuccessful of build.

Features

- It is integrated in GitLab interface.
- It has earned more popularity in the past few years due to its simple usage, faster results etc.
- It allows the project team members to integrate their work daily.
- The integration errors can be identified easily by an automated build.
- It can be executed on multiple platforms such as Windows, Unix, OSX and other platforms which support Go programming language.

Advantages

- It is easy to learn, use and scalable.
- It displays the faster results as it divides the each build into multiple jobs that run on multiple machines.
- It is free and open source software which is added in both GitLab Community Edition and the proprietary GitLab Enterprise Edition.

Description

GitLab CI (Continuous Integration) service is a part of GitLab that build and test the software whenever developer pushes code to application. GitLab CD (Continuous Deployment) is a software service that places the changes of every code in the production which results in every day deployment of production.

The following points describe usage of GitLab CI/CD −

- It is easy to learn, use and scalable.
- It is faster system which can be used for code deployment and development.
- You can execute the jobs faster by setting up your own runner (it is an application that processes the builds) with all dependencies which are pre-installed.
- GitLab CI solutions are economical and secure which are very flexible in costs as much as machine used to run it.
- It allows the project team members to integrate their work daily, so that the integration errors can be identified easily by an automated build.

S.No.	Variable	GitLab	Runner	Description
1	CI	all	0.4	Specifies that job is accomplished in CI environment.
2	CI_COMMIT_REF_NAME	9.0	all	Defines the branch or tag name for project build.
3	CI_COMMIT_REF_SLUG	9.0	all	It uses the lowercased *$CI_COMMIT_REF_NAME* variable which is reduced to 63 bytes, and only 0-9 and a-z replaced with -.
4	CI_COMMIT_SHA	9.0	all	Specifies the commit revision for built project.
5	CI_COMMIT_TAG	9.0	0.5	It commits the tag name
6	CI_CONFIG_PATH	9.4	0.5	Specifies the path to CI config file. (The default path is *.gitlab-ci.yml*).
7	CI_DEBUG_TRACE	all	1.7	It enables the debug tracing.
8	CI_ENVIRONMENT_NAME	8.15	all	Defines the environment name for the job.
9	CI_ENVIRONMENT_SLUG	8.15	all	It is a environment name, suitable for DNS, URLs, Kubernetes labels, etc.
10	CI_ENVIRONMENT_URL	9.3	all	Defines the environment URL for the job.
11	CI_JOB_ID	9.0	all	Represents the unique id of the current job for GitLab CI.
12	CI_JOB_MANUAL	8.12	all	It specifies that job has been started manually.
13	CI_JOB_NAME	9.0	0.5	The job name is defined in the *.gitlab-ci.yml* file.
14	CI_JOB_STAGE	9.0	0.5	The stage name is defined in the *.gitlab-ci.yml* file.
15	CI_JOB_TOKEN	9.0	1.2	This token is used for authenticating with the GitLab Container Registry and multi-project pipelines when triggers are involved.
16	CI_REPOSITORY_URL	9.0	all	It specifies the URL to clone the Git repository.
17	CI_RUNNER_DESCRIPTION	8.10	0.5	It specifies the description for the runner.

18	CI_RUNNER_ID	8.10	0.5	It provides the unique id for runner being used.
19	CI_RUNNER_TAGS	8.10	0.5	It defines the runner tags.
20	CI_RUNNER_VERSION	all	10.6	It specifies the GitLab runner version of the current job.
21	CI_RUNNER_REVISION	all	10.6	It specifies the GitLab revision of the current job.
22	CI_PIPELINE_ID	8.10	0.5	It provides the unique id of the current pipeline.
23	CI_PIPELINE_SOURCE	9.3	all	It specifies how the pipeline was triggered by using some options such as push, web, trigger, schedule, api, pipeline.
24	CI_PIPELINE_TRIGGERED	all	all	It specifies that job was triggered.
25	CI_PIPELINE_SOURCE	10.0	all	It specifies source of the pipeline such as push, web, trigger, schedule, api, external.
26	CI_PROJECT_DIR	all	all	It defines the full path of the cloned repository, where the job is run.
27	CI_PROJECT_ID	all	all	It provides the unique id of the current project.
28	CI_PROJECT_NAME	8.10	0.5	It provides the name of the current project.
29	CI_PROJECT_PATH	8.10	0.5	It provides the name of the project along with namespace.
30	CI_PROJECT_URL	8.10	0.5	It gives the http address to retrieve the project.
31	CI_PROJECT_VISIBILITY	10.3	all	It specifies the project visibility whether it is internal, private or public.
32	CI_REGISTRY	8.10	0.5	It returns the address of GitLab's Container Registry, only if the Container Registry is enabled.
33	CI_REGISTRY_IMAGE	8.10	0.5	It returns the address of GitLab's Container Registry which is tied to specific project, only if the Container Registry is enabled.

34	CI_REGISTRY_PASSWORD	9.0	all	The password can be used to push the containers to the GitLab Container Registry.
35	CI_REGISTRY_USER	9.0	all	The username can be used to push the containers to the GitLab Container Registry.
36	CI_SERVER	all	all	It specifies that job is executed in CI environment.
37	CI_SERVER_NAME	all	all	It gives the CI server name to coordinate the jobs.
38	CI_SERVER_REVISION	all	all	It is used to schedule the jobs by using GitLab revision.
39	CI_SERVER_VERSION	all	all	It is used to schedule the jobs by using GitLab version.
40	CI_SHARED_ENVIRONMENT	all	10.1	It indicates that job is executed in a shared environment and it is set to true, if the environment is shared.
41	ARTIFACT_DOWNLOAD_ATTEMPTS	8.15	1.9	It specifies the number of attempts to download artifacts running a job.
42	GET_SOURCES_ATTEMPTS	8.15	1.9	It specifies the number of attempts to get the sources running a job.
43	GITLAB_CI	all	all	It specifies that job is accomplished in GitLab CI environment.
44	GITLAB_USER_ID	8.12	all	It specifies the id of GitLab user who is running a job.
45	GITLAB_USER_EMAIL	8.12	all	It specifies the email of GitLab user who is running a job.
46	GITLAB_USER_LOGIN	10.0	all	It specifies the login username of GitLab user who is running a job.
47	GITLAB_USER_NAME	10.0	all	It specifies the real name of GitLab user who is running a job.
48	GITLAB_FEATURES	10.6	all	It provides list of the licensed features for the GitLab instance and plan.

49	RESTORE_CACHE_ATTEMPTS	8.15	1.9	It defines number of cache attempts to restore the running a job.
50	CI_DISPOSABLE_ENVIRONMENT	all	10.1	It indicates that job is executed in a disposable environment and it is set to true, if the environment is disposable.

The following table shows list of GitLab CI/CD variables.

The following table shows list of new variables which can be used with GitLab 9.0 release −

S.No.	9.0+ name
1	CI_JOB_ID
2	CI_COMMIT_SHA
3	CI_COMMIT_TAG
4	CI_COMMIT_REF_NAME
5	CI_COMMIT_REF_SLUG
6	CI_JOB_NAME
7	CI_JOB_STAGE
8	CI_REPOSITORY_URL
9	CI_PIPELINE_TRIGGERED
10	CI_JOB_MANUAL
11	CI_JOB_TOKEN

S.N.	Guest	Reporter	Developer	Master	Owner
1	Creates a new issue	Creates a new issue	Creates a new issue	Creates a new issue	Creates a new issue
2	Can leave comments	Can leave comments	Can leave comments	Can leave comments	Can leave comments
3	Able to write on project wall	Able to write on project wall	Able to write on project wall	Able to write on project wall	Able to write on project wall
4	-	Able to pull project code	Able to pull project code	Able to pull project code	Able to pull project code
5	-	Can download project	Can download project	Can download project	Can download project
6	-	Able to write code snippets	Able to write code snippets	Able to write code snippets	Able to write code snippets
7	-	-	Create new merge request	Create new merge request	Create new merge request
8	-	-	Create new branch	Create new branch	Create new branch
9	-	-	Push and remove non protected branches	Push and remove non protected branches	Push and remove non protected branches
10	-	-	Includes tags	Includes tags	Includes tags
11	-	-	Can create, edit, delete project milestones	Can create, edit, delete project milestones	Can create, edit, delete project milestones
12	-	-	Can create or update commit status	Can create or update commit status	Can create or update commit status

13	-	-	Write a wiki	Write a wiki	Write a wiki
14	-	-	Create new environments	Create new environments	Create new environments
15	-	-	Cancel and retry the jobs	Cancel and retry the jobs	Cancel and retry the jobs
16	-	-	Updates and removes the registry image	Updates and removes the registry image	Updates and removes the registry image
17	-	-	-	Can add new team members	Can add new team members
18	-	-	-	Push and remove protected branches	-
19	-	-	-	Can edit the project	Can edit the project
20	-	-	-	Can manage runners, job triggers and variables	Can manage runners, job triggers and variables
21	-	-	-	Add deploy keys to project	Add deploy keys to project
22	-	-	-	Able to manage clusters	Able to manage clusters
23	-	-	-	Configure project hooks	Configure project hooks
24	-	-	-	Can enable/disable the branch protection	Can enable/disable the branch protection

25	-	-	-	Able to rewrite or remove Git tags	Able to rewrite or remove Git tags

User Permissions

The following table shows available user permissions levels for different types of users in a project −

The following table shows available group members permissions levels in a group −

S.N.	Guest	Reporter	Developer	Master	Owner
1	Browse group	Browse group	Browse group	Browse group	Browse group
2	-	-	-	-	Edit group
3	-	-	-	-	Create subgroup
4	-	-	-	Create project in group	Create project in group
5	-	-	-	-	Manage group members
6	-	-	-	-	Remove group
7	-	Manage group labels	Manage group labels	Manage group labels	Manage group label
8	-	-	Create/edit/delete group milestones	Create/edit/delete group milestones	Create/edit/delete group milestones
9	-	View private group epic	View private group epic	View private group epic	View private group epic
10	-	-	-	-	-

11	View internal group epic	View internal group epic	View internal group epic	View internal group epic	View internal group epic
12	View public group epic	View public group epic	View public group epic	View public group epic	View public group epic
13	-	Create/edit group epic	Create/edit group epic	Create/edit group epic	Create/edit group epic
14	-	-	-	-	Delete group epic
15	-	-	-	-	View group Audit Events

S.N.	Guest/Reporter	Developer	Master	Admin
1	Can see commits and jobs	Can see commits and jobs	Can see commits and jobs	Can see commits and jobs
2		Retry or cancel job	Retry or cancel job	Retry or cancel job
3	-	Deletes job artifacts and trace	Deletes job artifacts and trace	Deletes job artifacts and trace
4	-	-	Remove project	Remove project
5	-	-	Create project	Create project
6	-	-	Change project configuration	Change project configuration
7	-	-	Add specific runners	Add specific runners
8	-	-	-	Add shared runners
9	-	-	-	Can able to see events in the system
10	-	-	-	Admin interface

The following table shows available GitLab CI/CD permissions in the GitLab −

S.N.	Guest/Reporter	Developer	Master	Admin
1	-	Run CI job	Run CI job	Run CI job
2	-	Clone source and LFS from current project	Clone source and LFS from current project	Clone source and LFS from current project
3	-	Clone source and LFS from public projects	Clone source and LFS from public projects	Clone source and LFS from public projects
4	-	Clone source and LFS from internal projects	Clone source and LFS from internal projects	Clone source and LFS from internal projects
5	-	Clone source and LFS from private projects	Clone source and LFS from private projects	Clone source and LFS from private projects
6	-	Push source and LFS	Push source and LFS	Push source and LFS
7	-	Pull container images from current project	Pull container images from current project	Pull container images from current project
8	-	Pull container images from public projects	Pull container images from public projects	Pull container images from public projects
9	-	Pull container images from internal projects	Pull container images from internal projects	Pull container images from internal projects
10	-	Pull container images from private projects	Pull container images from private projects	Pull container images from private projects
11	-	Push container images to current project	Push container images to current project	Push container images to current project
12	-	Push container images to other projects	Push container images to other projects	Push container images to other projects

Job Permissions

The following table shows job permissions in the GitLab −

Note − LFS stands for **L**arge **F**ile **S**torage which is a Git extension that exchanges the large files such as audio, video, graphics with tiny pointers files in your repository.

Lab 14: GitLab Runners

Description

GitLab runner is a build instance which is used to run the jobs over multiple machines and send the results to GitLab and which can be placed on separate users, servers, and local machine. You can register the runner as shared or specific after installing it. The installation of runner is explained in the GitLab Installation chapter.

You can serve your jobs by using either specific or shared runners.

Shared Runners

These runners are useful for jobs multiple projects which have similar requirements. Instead of using multiple runners for many projects, you can use a single or a small number of Runners to handle multiple projects which will be easy to maintain and update.

Specific Runners

These runners are useful to deploy a certain project, if jobs have certain requirements or specific demand for the projects. Specific runners use *FIFO* (First In First Out) process for organizing the data with first-come first-served basis.

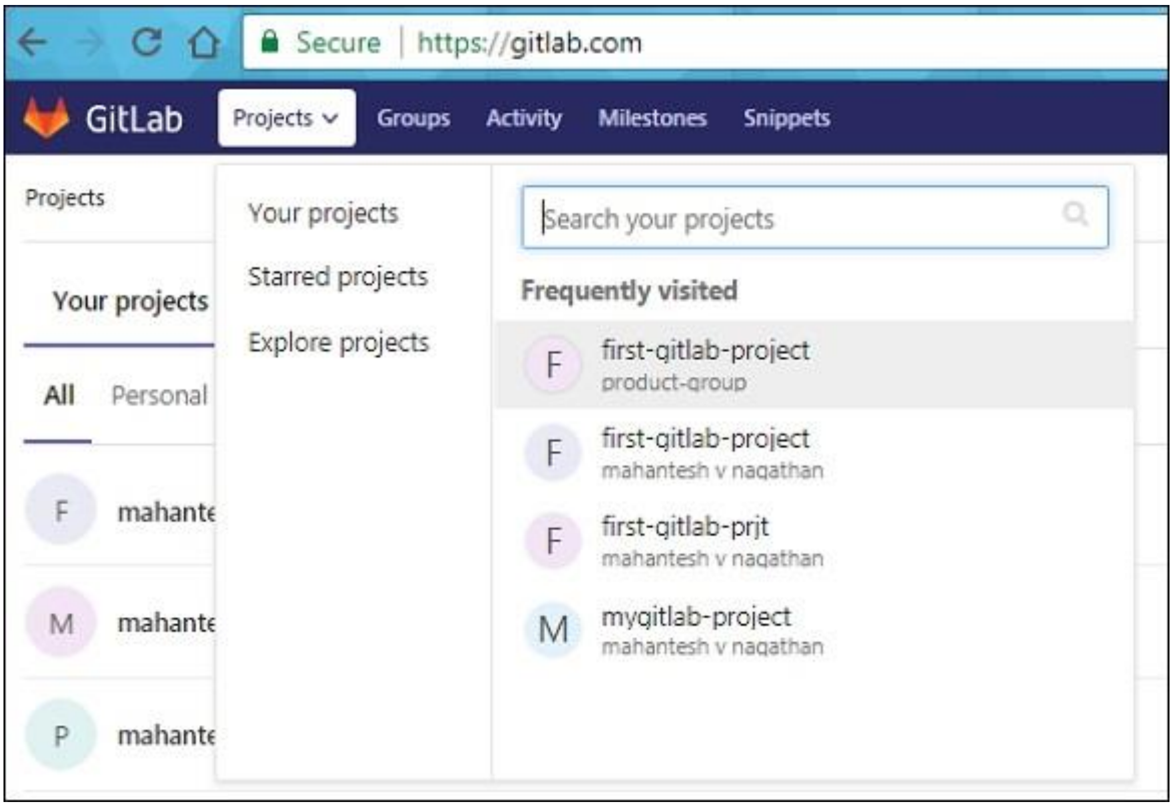
You can register a specific runner by using project registration token. The registering a specific runner is explained in the GitLab Installation chapter from step 1 to 12 under the *Installation of GitLab on Windows* section.

Locking a specific Runner

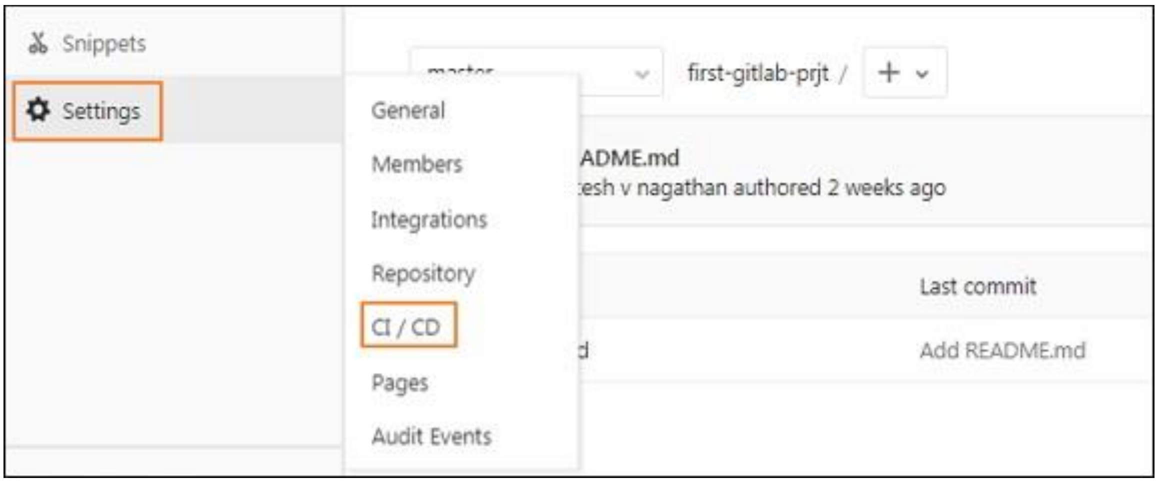
You can lock a specific runner from being enabled for other projects. To do this, you need to register a runner which is explained in the GitLab Install`ation chapter from step 1 to 12 under the *Installation of GitLab on Windows* section.

To lock runner, execute the below steps −

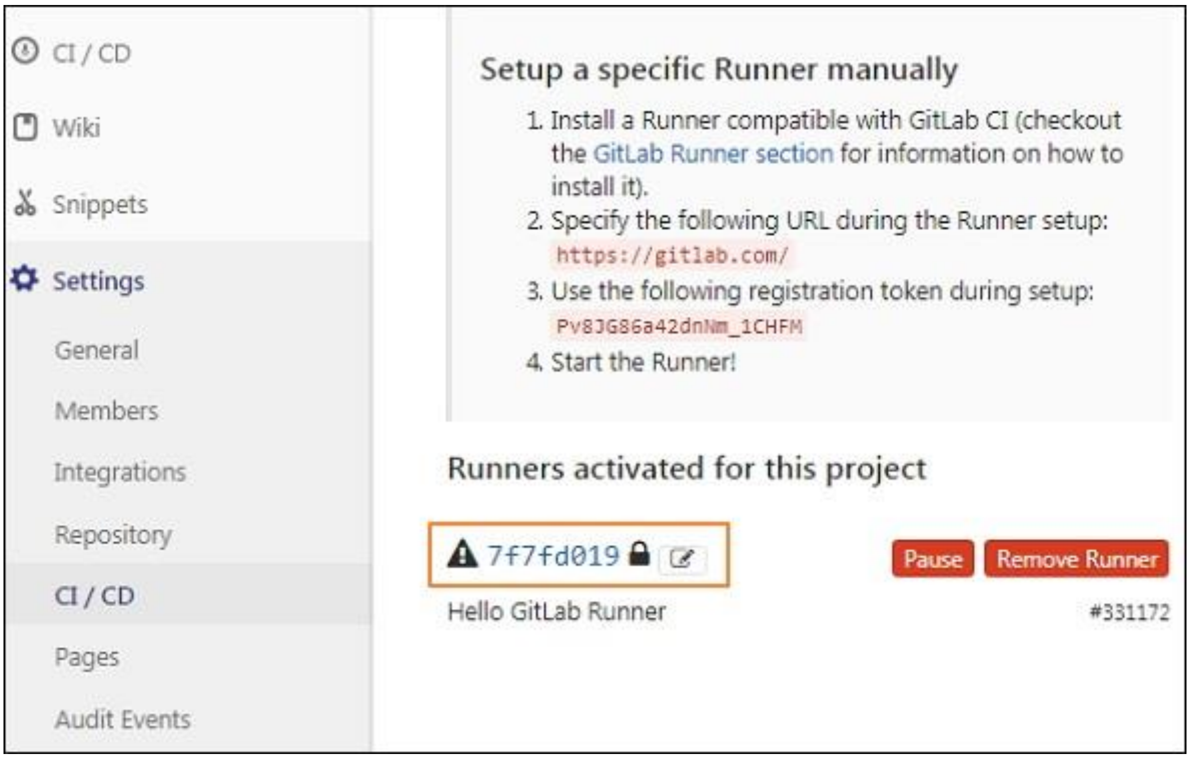
Step 1 − Login to your GitLab account and go to your project −



Step 2 − Click on the CI/CD option under Settings tab and expand the Runners Settings option. −



Step 3 − Under Runners Settings section, you will see the activated Runners for the project −



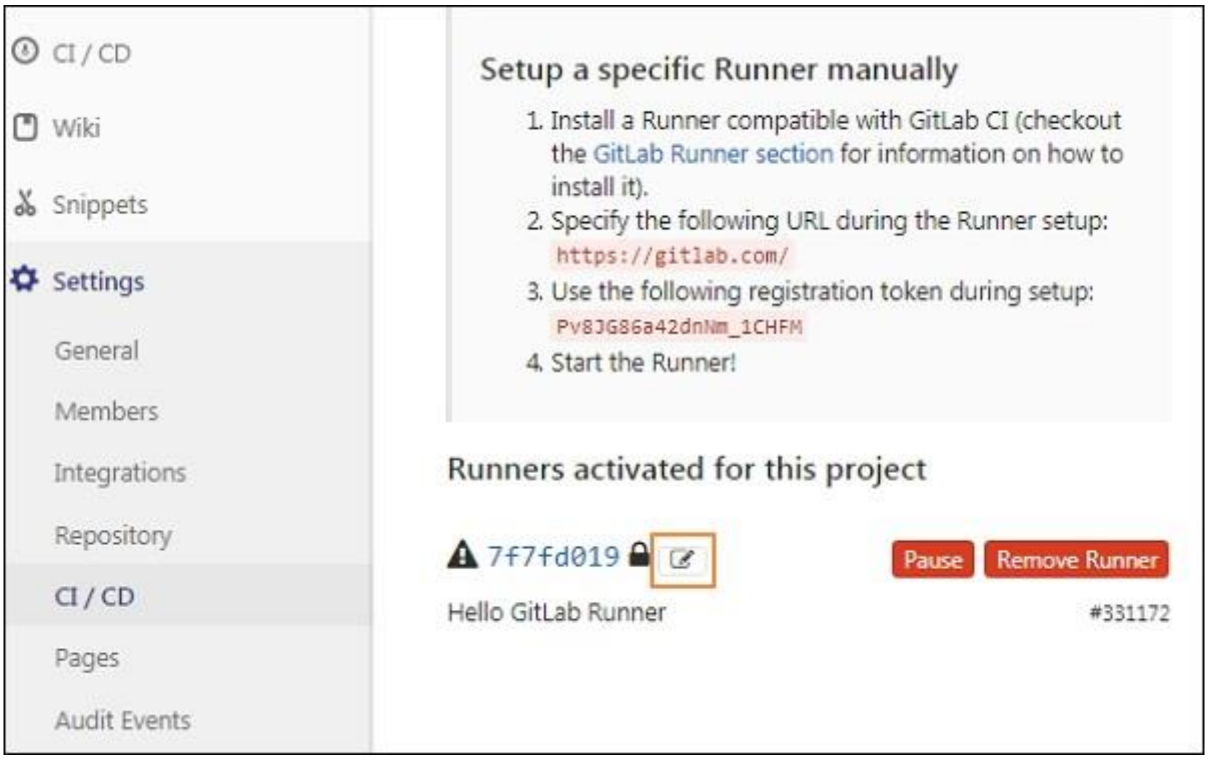
Setup a specific Runner manually

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `https://gitlab.com/`
3. Use the following registration token during setup: `Pv8JG86a42dnNm_1CHFM`
4. Start the Runner!

Runners activated for this project

7f7fd019 Hello GitLab Runner #331172

Step 4 − Now click on the pencil button −



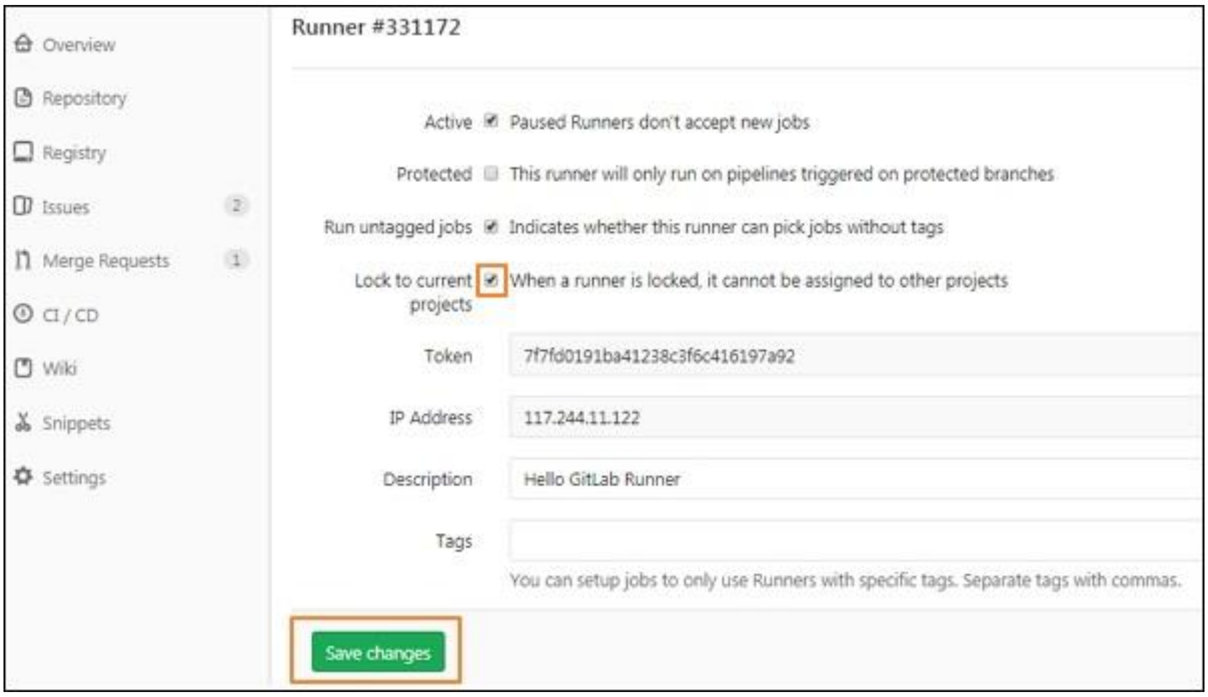
Setup a specific Runner manually

1. Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it).
2. Specify the following URL during the Runner setup: `https://gitlab.com/`
3. Use the following registration token during setup: `Pv8JG86a42dnNm_1CHFM`
4. Start the Runner!

Runners activated for this project

7f7fd019 Hello GitLab Runner #331172

Step 5 − Next it will open the Runner screen and check the *Lock to current projects* option −

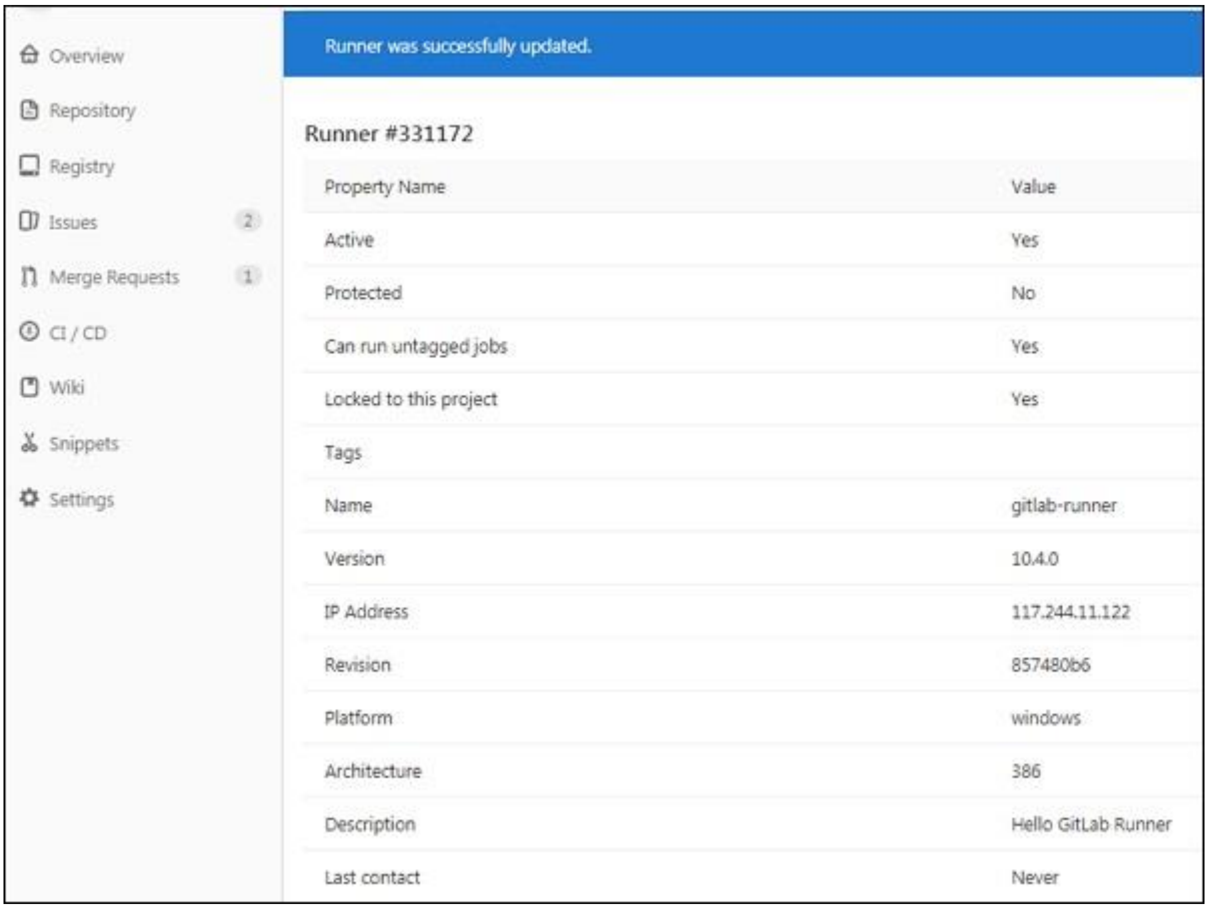


Runner #331172

Active	✓ Paused Runners don't accept new jobs
Protected	This runner will only run on pipelines triggered on protected branches
Run untagged jobs	✓ Indicates whether this runner can pick jobs without tags
Lock to current projects	✓ When a runner is locked, it cannot be assigned to other projects
Token	7f7fd0191ba41238c3f6c416197a92
IP Address	117.244.11.122
Description	Hello GitLab Runner
Tags	You can setup jobs to only use Runners with specific tags. Separate tags with commas.

Click on the *Save changes* button to take the changes effect.

Step 6 − After saving the changes, it will update the Runner successfully.



Runner was successfully updated.

Runner #331172

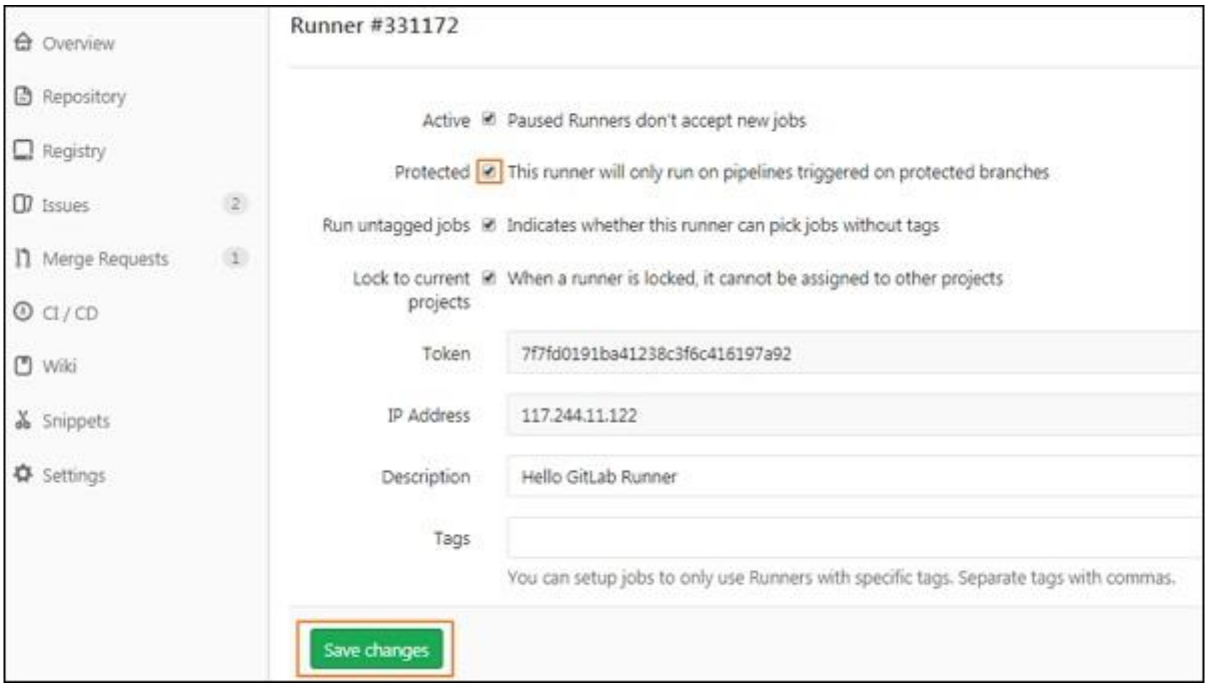
Property Name	Value
Active	Yes
Protected	No
Can run untagged jobs	Yes
Locked to this project	Yes
Tags	
Name	gitlab-runner
Version	10.4.0
IP Address	117.244.11.122
Revision	857480b6
Platform	windows
Architecture	386
Description	Hello GitLab Runner
Last contact	Never

Protected Runners

The runners can be protected to save the important information. You can protect the runner by using below steps −

Step 1 − Follow the same steps (from step 1 to 4) which are explained in the previous section (Locking a specific Runner).

Step 2 − After clicking on the pencil button, it will open the Runner screen and then check the *Protected* option −



Runner #331172

Active	✓ Paused Runners don't accept new jobs
Protected	✓ This runner will only run on pipelines triggered on protected branches
Run untagged jobs	✓ Indicates whether this runner can pick jobs without tags
Lock to current projects	✓ When a runner is locked, it cannot be assigned to other projects
Token	7f7fd0191ba41238c3f6c416197a92
IP Address	117.244.11.122
Description	Hello GitLab Runner
Tags	You can setup jobs to only use Runners with specific tags. Separate tags with commas.

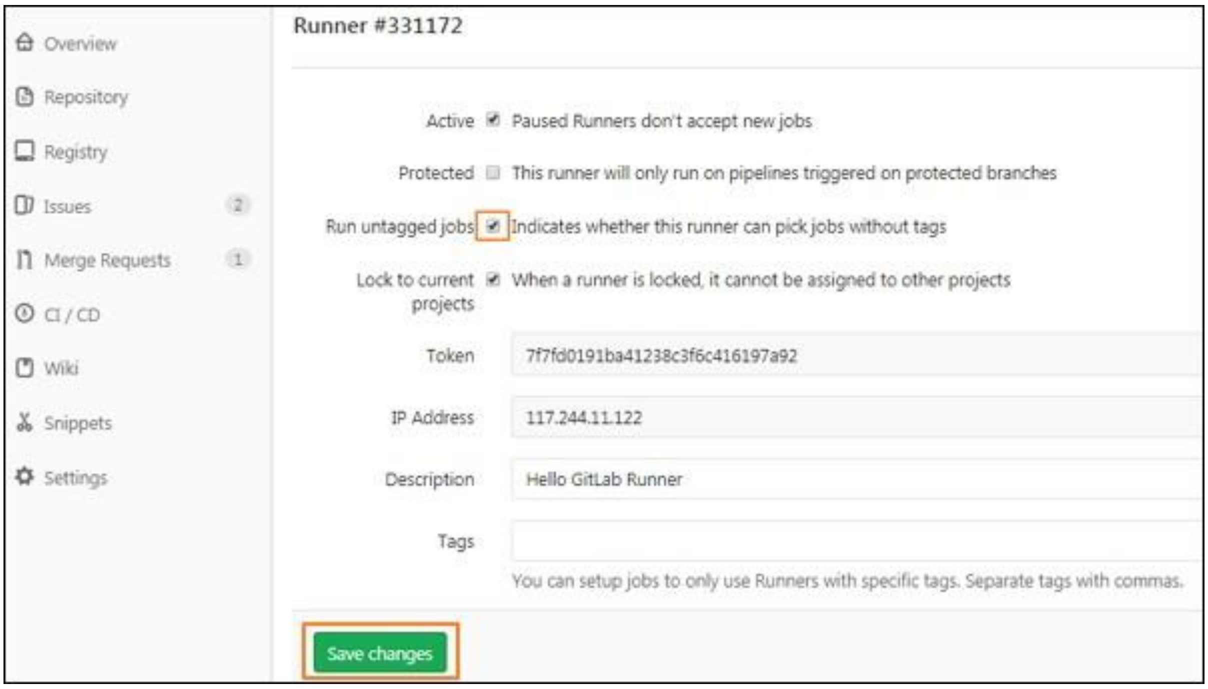
Click on the *Save changes* button to take the changes effect.

Run untagged Jobs

You can prevent runners from picking jobs with tags when there are no tags assigned to runners. Runner can pick tagged/untagged jobs by using below steps −

Step 1 − Follow the same steps (from step 1 to 4) which are explained in the *Locking a specific Runner* section.

Step 2 − After clicking on the pencil button, it will open the Runner screen and then check the *Run untagged jobs* option −



Runner #331172

Active	✓ Paused Runners don't accept new jobs
Protected	☐ This runner will only run on pipelines triggered on protected branches
Run untagged jobs	✓ Indicates whether this runner can pick jobs without tags
Lock to current projects	✓ When a runner is locked, it cannot be assigned to other projects
Token	7f7fd0191ba41238c3f6c416197a92
IP Address	117.244.11.122
Description	Hello GitLab Runner
Tags	You can setup jobs to only use Runners with specific tags. Separate tags with commas.

Click on the *Save changes* button to take the changes effect.

Lab 15: Environments and Deployments

Environments and Deployments

Environments are used for testing, building and deploying the CI (Continuous Integration) jobs and control the Continuous Deployment of software with the GitLab. GitLab CI is capable of tracking your project deployments and also you will come to know what is being deployed on your server.

The name of an environment could be defined by using *environment:name* string and contain the following −

- letters
- digits
- spaces
- -
- _
- /
- $
- {
- }

Using SSH keys with GitLab CI/CD

You can set the SSH (Secure Shell or Secure Socket Shell) keys to provide a reliable connection between the computer and GitLab. The SSH keys can be used with GitLab CI/CD when −

- You need to checkout internal sub modules.
- You need to download private packages using package manager.
- You need to install an application to your own server.
- You execute the SSH commands to remote server from build environment.
- You need to rsync files to a remote server from the build environment.

The SSH key setup is explained in the GitLab SSH Key Setup chapter.

Artifacts

Artifacts are used to attach the list of files and directories to the job after success. The artifacts contain following types −

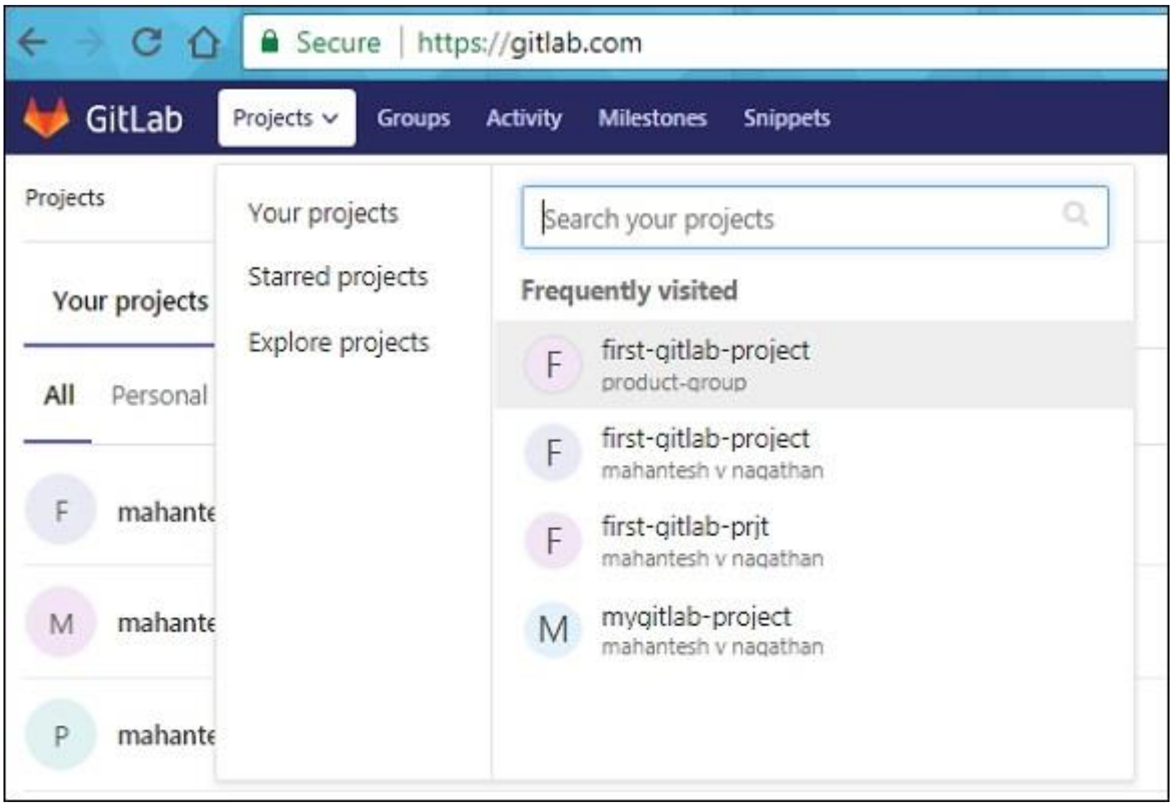
- **artifacts:name** − This directive is used to specify the name of created artifacts archive. It provides unique name for created artifacts archive which is helpful when you are downloading the archive from GitLab.
- **artifacts:when** − This directive is used to upload artifacts when there is a job failure. It contains the following values:
    - **on_success** − It is used to upload the artifacts when there is a job success.
    - **on_failure** − It is used to upload the artifacts when the job fails.
    - **always** − It is used to upload the artifacts regardless of job status.
- **artifacts:expire_in** − It defines that how long artifacts should live before they expire and therefore deleted, since they are uploaded and stored on GitLab

Triggering Pipelines

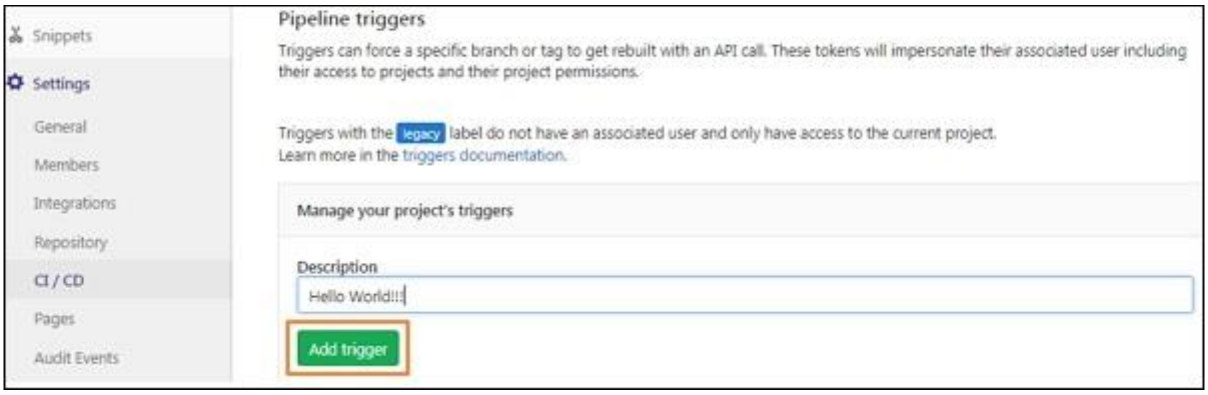
Triggers can force a specific branch or tag to get rebuilt with an API call and triggers with the *legacy* label will have access to the current project.

The new trigger can be added as shown in the below steps −

Step 1 − Login to your GitLab account and go to your project −



Step 2 − Click on the *CI/CD* option under *Settings* tab and expand the *Pipeline triggers* option −



Pipeline triggers

Triggers can force a specific branch or tag to get rebuilt with an API call. These tokens will impersonate their associated user including their access to projects and their project permissions.

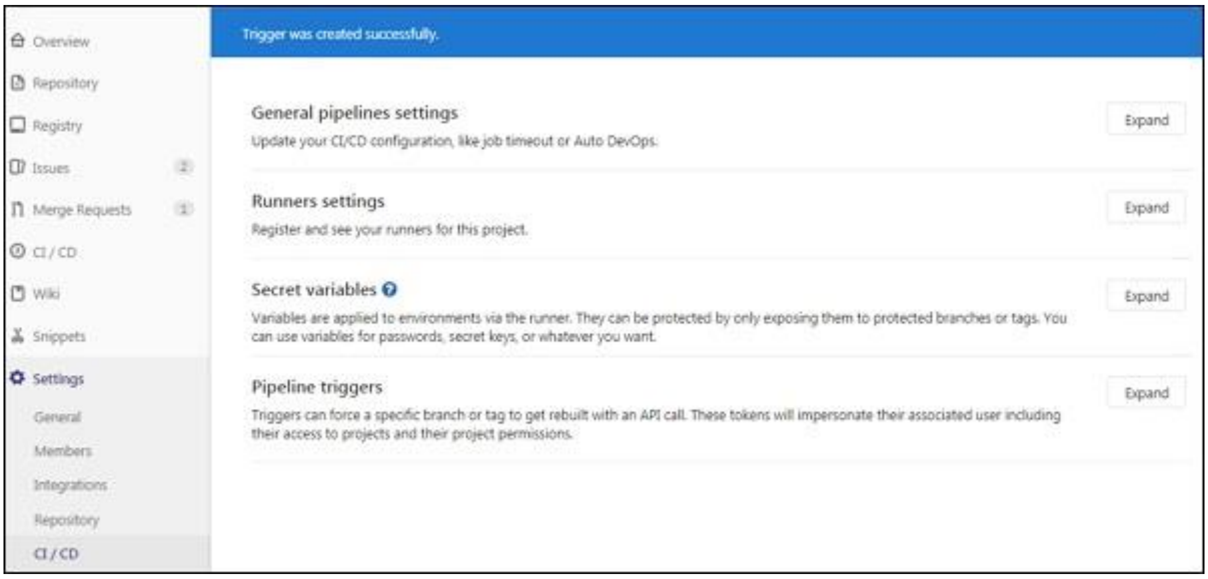
Triggers with the legacy label do not have an associated user and only have access to the current project.
Learn more in the triggers documentation.

Manage your project's triggers

Description
Hello World!!!

Enter the description for the trigger and click on the *Add Trigger* button.

Step 3 − Next, it will display the success message after creating the trigger −



Trigger was created successfully.

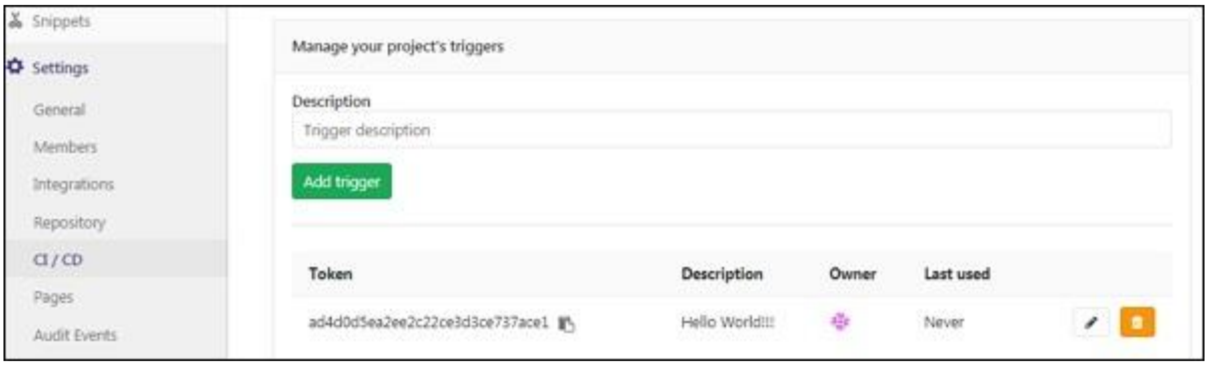
General pipelines settings
Update your CI/CD configuration, like job timeout or Auto DevOps.

Runners settings
Register and see your runners for this project.

Secret variables
Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

Pipeline triggers
Triggers can force a specific branch or tag to get rebuilt with an API call. These tokens will impersonate their associated user including their access to projects and their project permissions.

Step 4 − Now go to *CI/CD* option under *Settings* tab and expand the *Pipeline triggers* option. You will see the newly created trigger along with the token as shown in the image below −



Manage your project's triggers

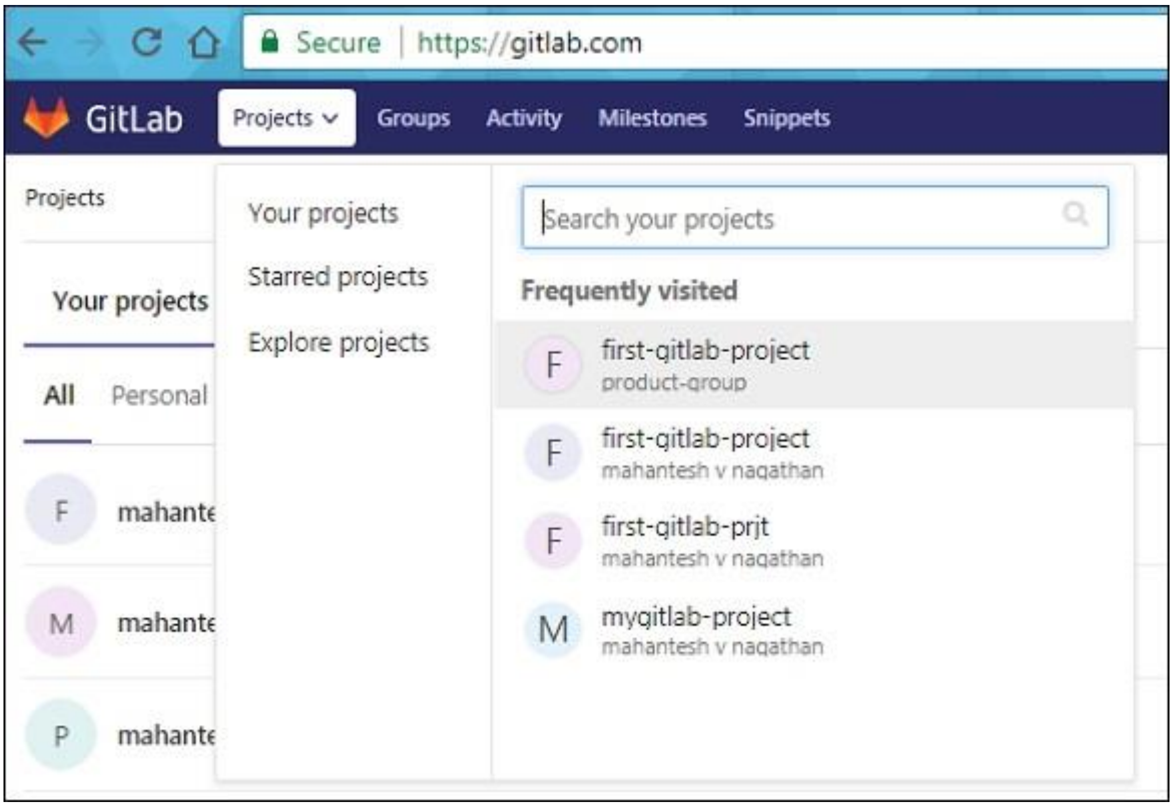
Description
Trigger description

Token	Description	Owner	Last used
ad4d0d5ea2ee2c22ce3d3ce737ace1	Hello World!!!		Never

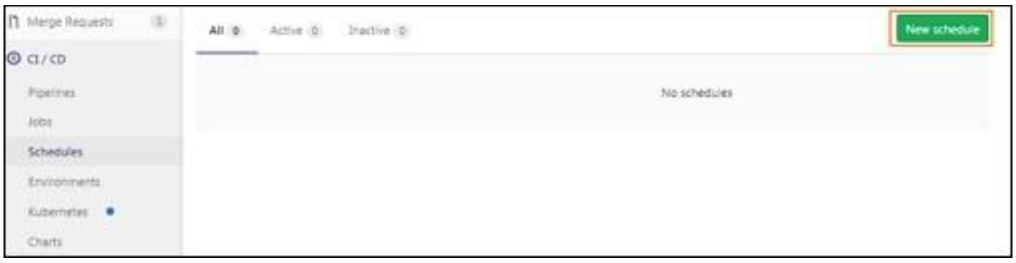
Pipeline Schedules

You can run the pipeline by using the pipeline schedules at specific intervals. To create pipeline schedule, use the below steps −

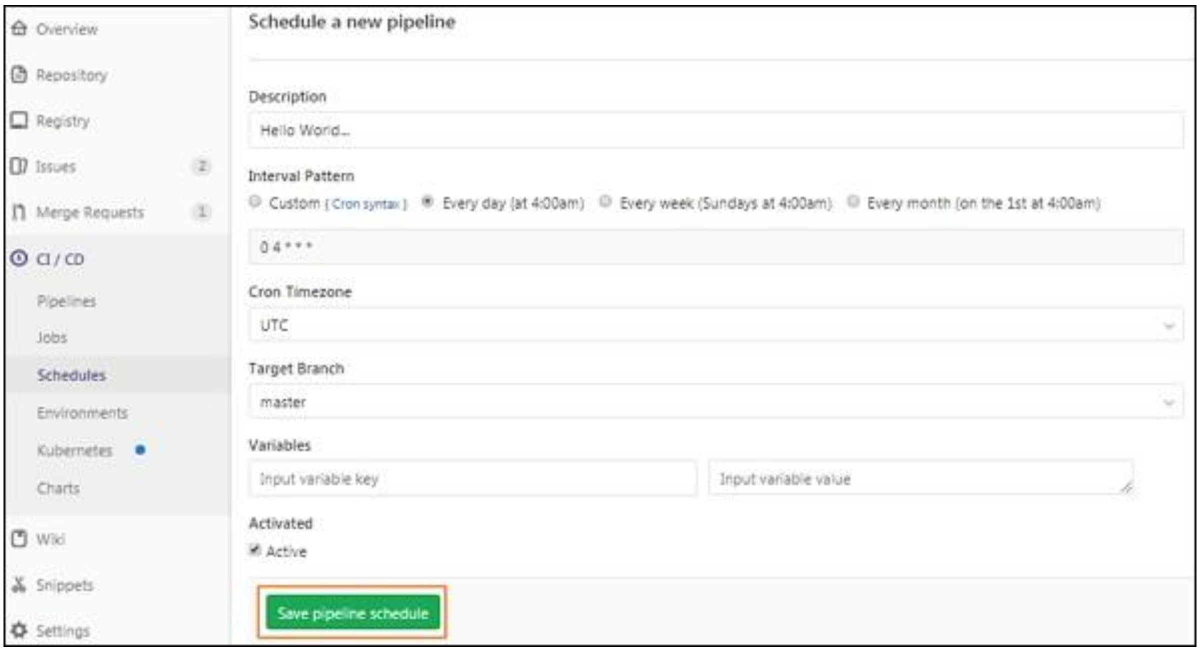
Step 1 − Login to your GitLab account and go to your project −



Step 2 − Click on the *Schedules* option under *CI/CD* tab and click on the *New schedule* button −



Step 3 − Next, it will open the Scheduling new pipeline screen, fill up the fields and click on the *Save pipeline schedule* button −



Schedule a new pipeline

Description: Hello World...

Interval Pattern: Custom (Cron syntax) / [✓ Every day (at 4:00am)] / Every week (Sundays at 4:00am) / Every month (on the 1st at 4:00am)

0 4 * * *

Cron Timezone: UTC

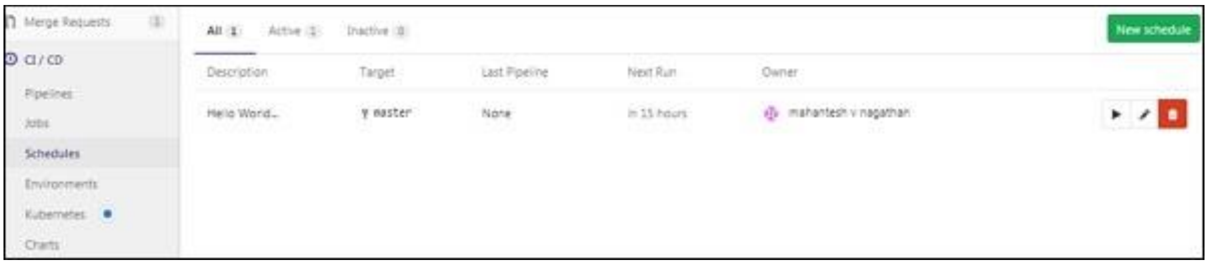
Target Branch: master

Variables: Input variable key | Input variable value

Activated: [✓ Active]

Save pipeline schedule

Step 4 − Now, you will see the pipeline which is scheduled to run −



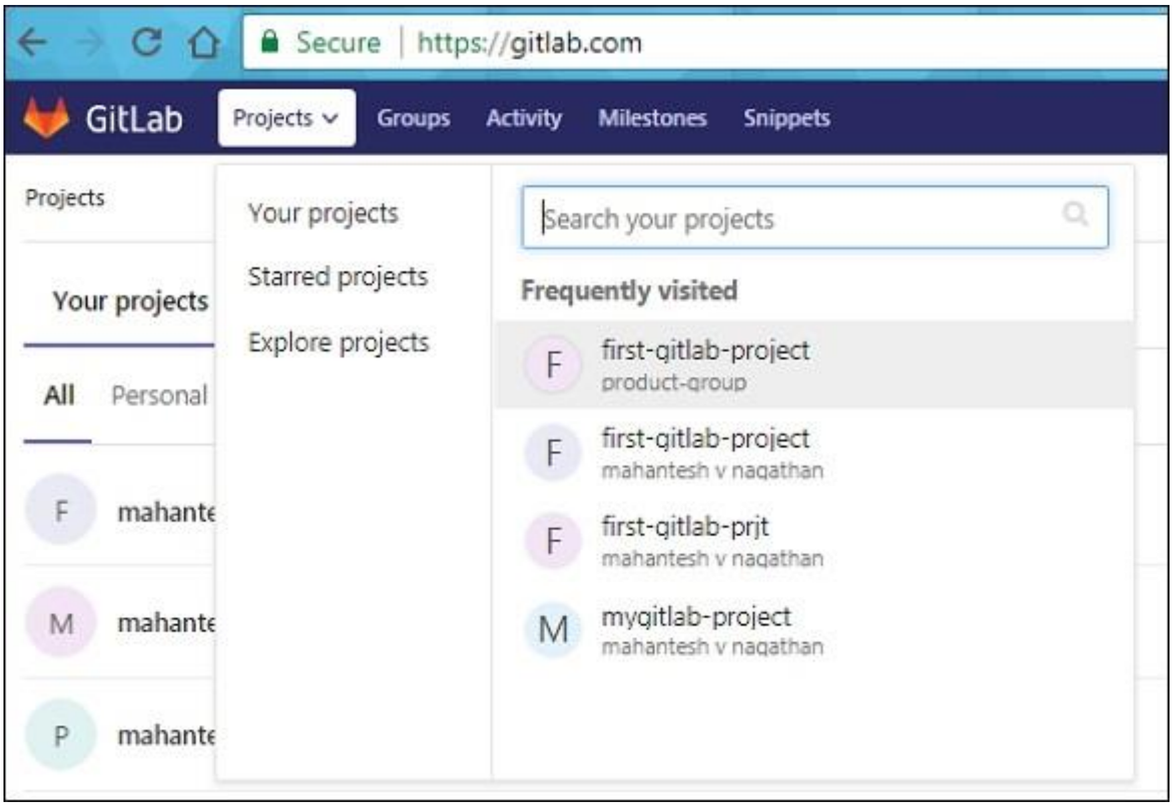
Description	Target	Last Pipeline	Next Run	Owner
Hello World...	master	None	in 15 hours	mahantesh v nagathan

Lab 16: Connecting GitLab with a Kubernetes Cluster

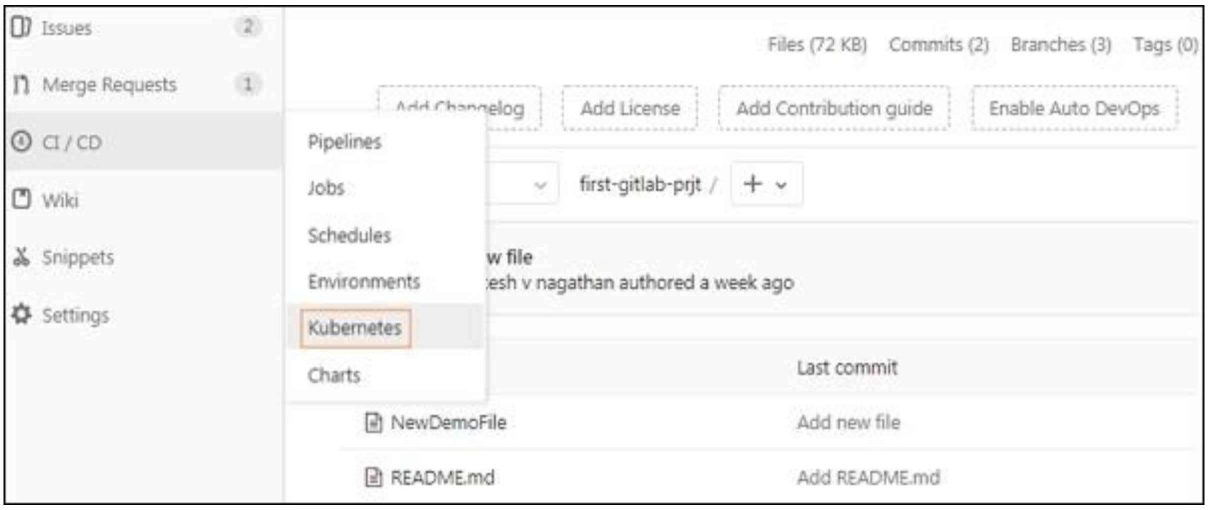
The Kubernetes cluster can be used to review and deploy the applications, running the pipeline etc in an easy method. You can create a new cluster to your project by associating your GitLab account with the Google Kubernetes Engine (GKE).

The new Kubernetes cluster can be created as shown in the below steps −

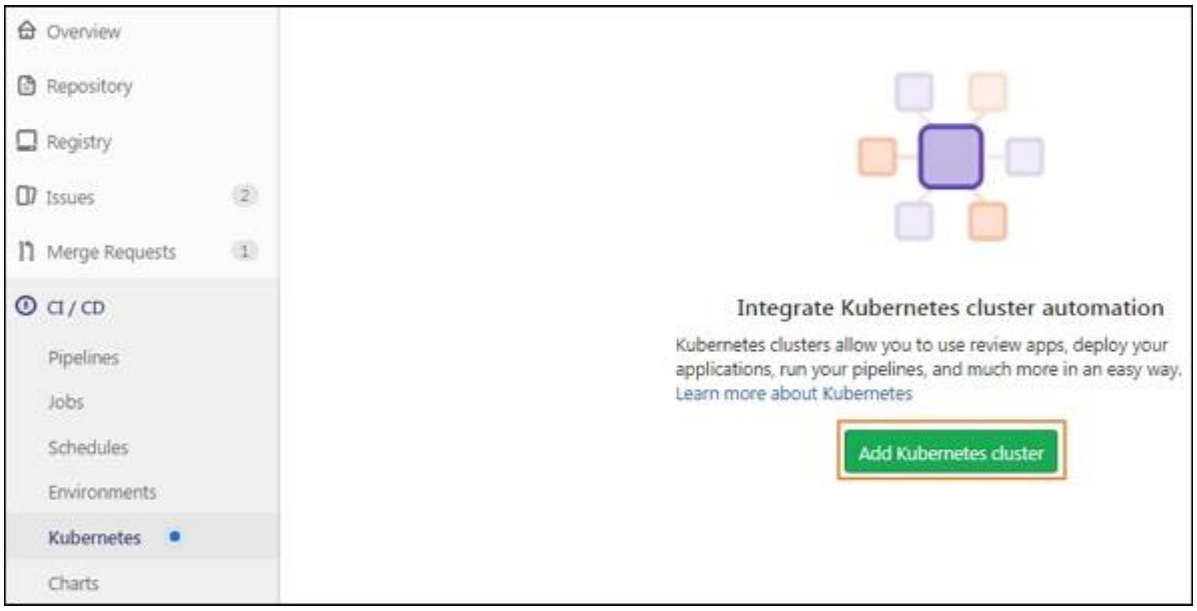
Step 1 − Login to your GitLab account and go to your project −



Step 2 − Click on the *Kubernetes* option under *CI/CD* tab −



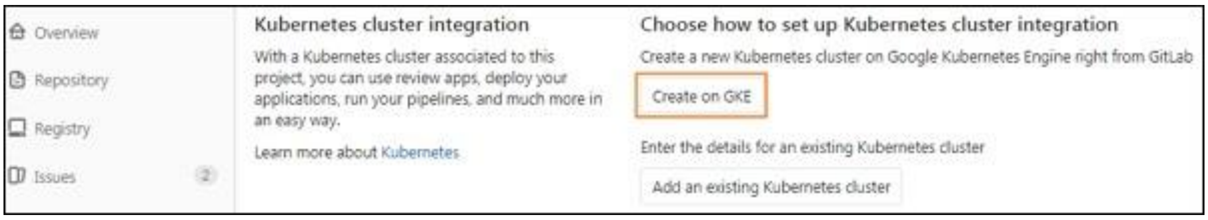
Step 3 − Next, click on *Add Kubernetes cluster* button −



Integrate Kubernetes cluster automation

Kubernetes clusters allow you to use review apps, deploy your applications, run your pipelines, and much more in an easy way.
Learn more about Kubernetes

Step 4 − Click on *Create on GKE* button to create a new Kubernetes cluster on Google Kubernetes Engine −



Kubernetes cluster integration

With a Kubernetes cluster associated to this project, you can use review apps, deploy your applications, run your pipelines, and much more in an easy way.

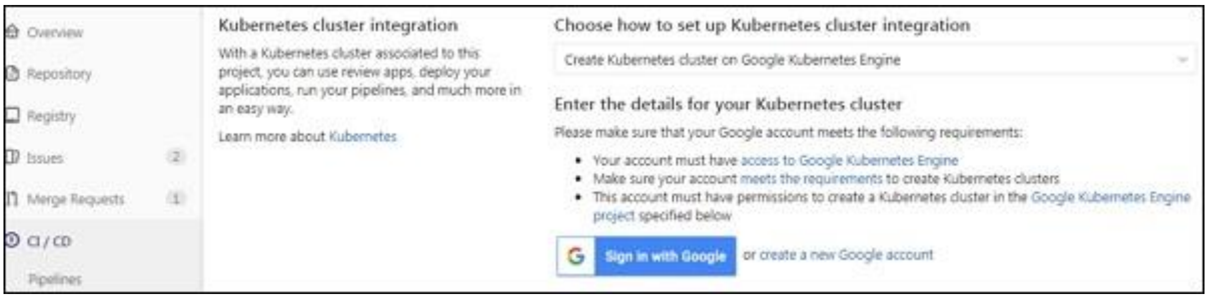
Learn more about Kubernetes

Choose how to set up Kubernetes cluster integration

Create a new Kubernetes cluster on Google Kubernetes Engine right from GitLab

Enter the details for an existing Kubernetes cluster

Step 5 − If you have a Google account, then sign with that account to enter the details for Kubernetes cluster or else create a new Google account −



Kubernetes cluster integration

With a Kubernetes cluster associated to this project, you can use review apps, deploy your applications, run your pipelines, and much more in an easy way.

Learn more about Kubernetes

Choose how to set up Kubernetes cluster integration

Create Kubernetes cluster on Google Kubernetes Engine

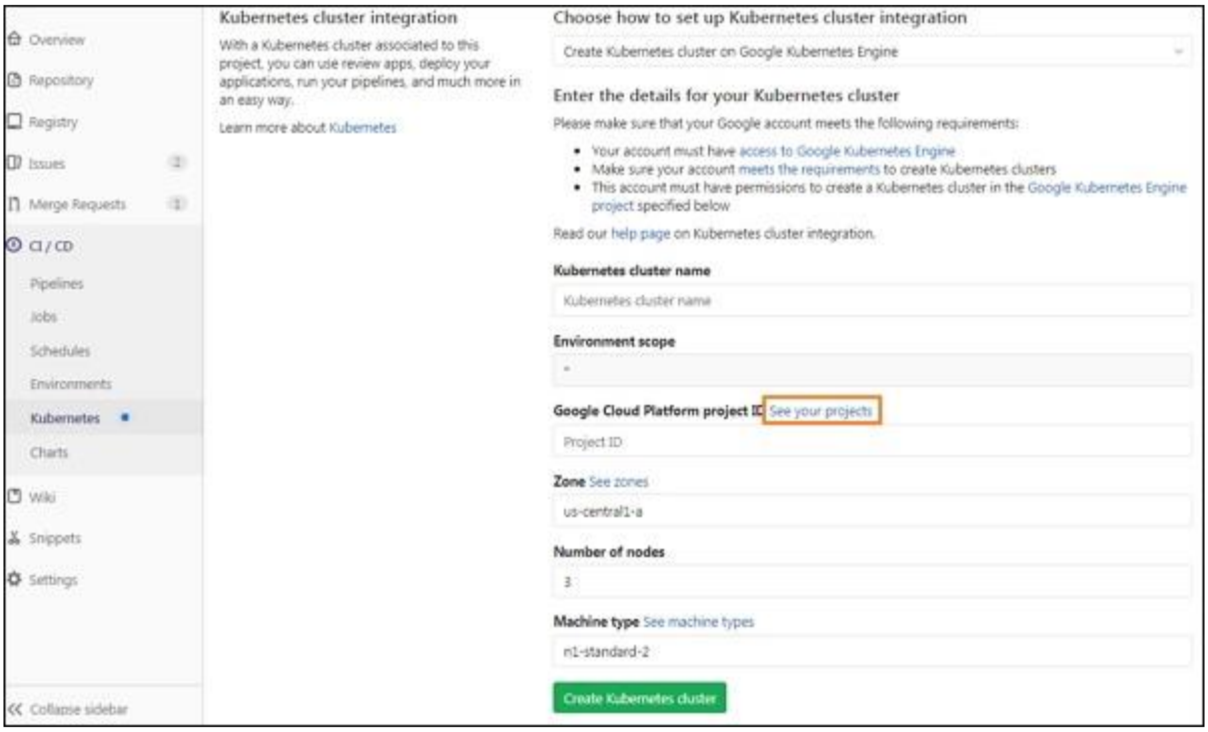
Enter the details for your Kubernetes cluster

Please make sure that your Google account meets the following requirements:

- Your account must have access to Google Kubernetes Engine
- Make sure your account meets the requirements to create Kubernetes clusters
- This account must have permissions to create a Kubernetes cluster in the Google Kubernetes Engine project specified below

or create a new Google account

Step 6 − Now enter the values in the fields for your Kubernetes cluster −



Kubernetes cluster integration

With a Kubernetes cluster associated to this project, you can use review apps, deploy your applications, run your pipelines, and much more in an easy way.

Learn more about Kubernetes

Choose how to set up Kubernetes cluster integration

Create Kubernetes cluster on Google Kubernetes Engine

Enter the details for your Kubernetes cluster

Please make sure that your Google account meets the following requirements:

- Your account must have access to Google Kubernetes Engine
- Make sure your account meets the requirements to create Kubernetes clusters
- This account must have permissions to create a Kubernetes cluster in the Google Kubernetes Engine project specified below

Read our help page on Kubernetes cluster integration.

Kubernetes cluster name: Kubernetes cluster name

Environment scope: *

Google Cloud Platform project ID See your projects: Project ID

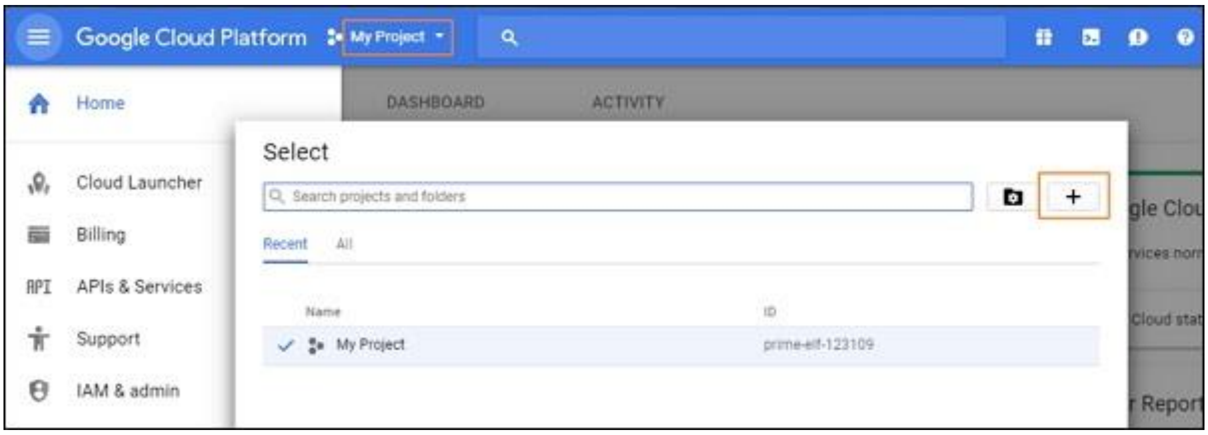
Zone See zones: us-central1-a

Number of nodes: 3

Machine type See machine types: n1-standard-2

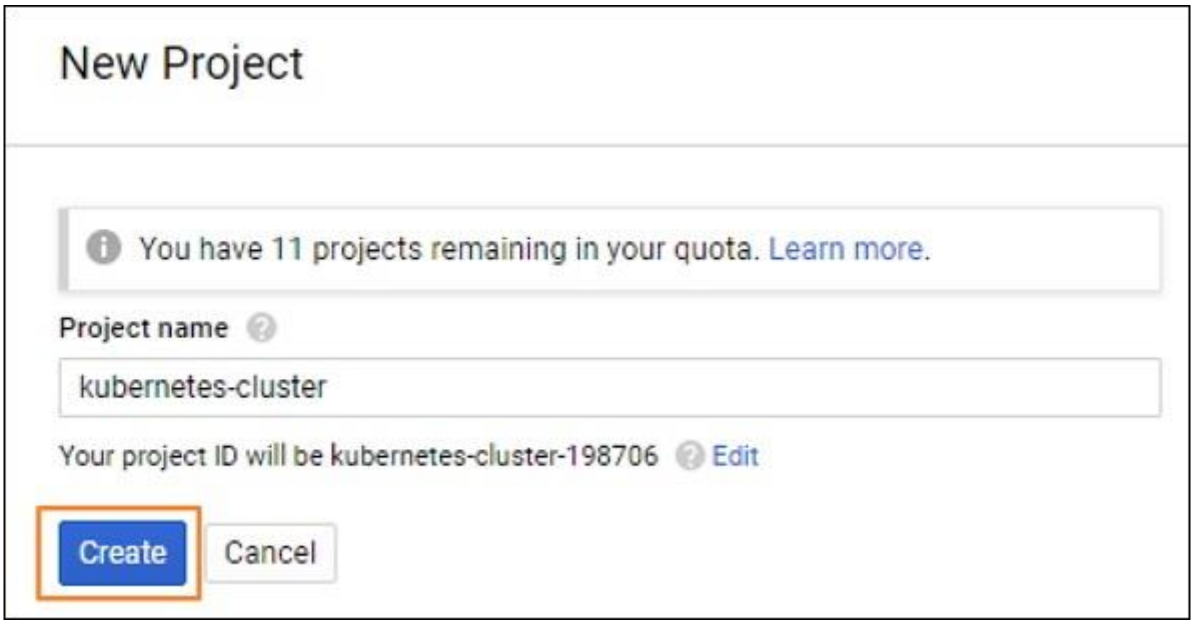
Create Kubernetes cluster

Step 7 − Before adding values in the fields, you need ID of the project which is created in the Google Cloud Platform console to host the Kubernetes cluster. To create ID, click on the *See your projects* link which is highlighted in the previous image. It will open the below screen, then click on *My Project* menu and click on the plus (+) icon to create a new project −



Name	ID
✓ My Project	prime-elf-123109

Step 8 − Now enter the project name and click on the *Create* button −

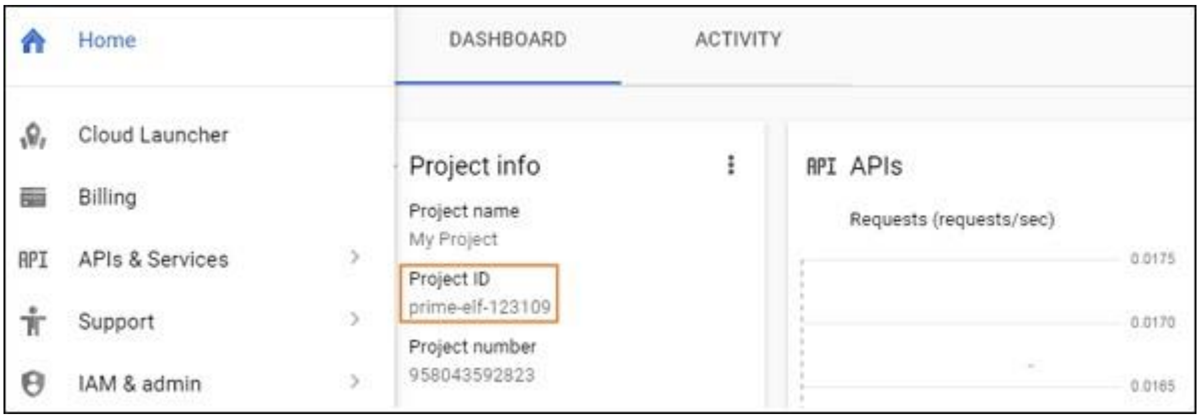

New Project

You have 11 projects remaining in your quota. Learn more.

Project name: kubernetes-cluster

Your project ID will be kubernetes-cluster-198706 Edit

Step 9 − You will get the ID of the project which will host the Kubernetes cluster −



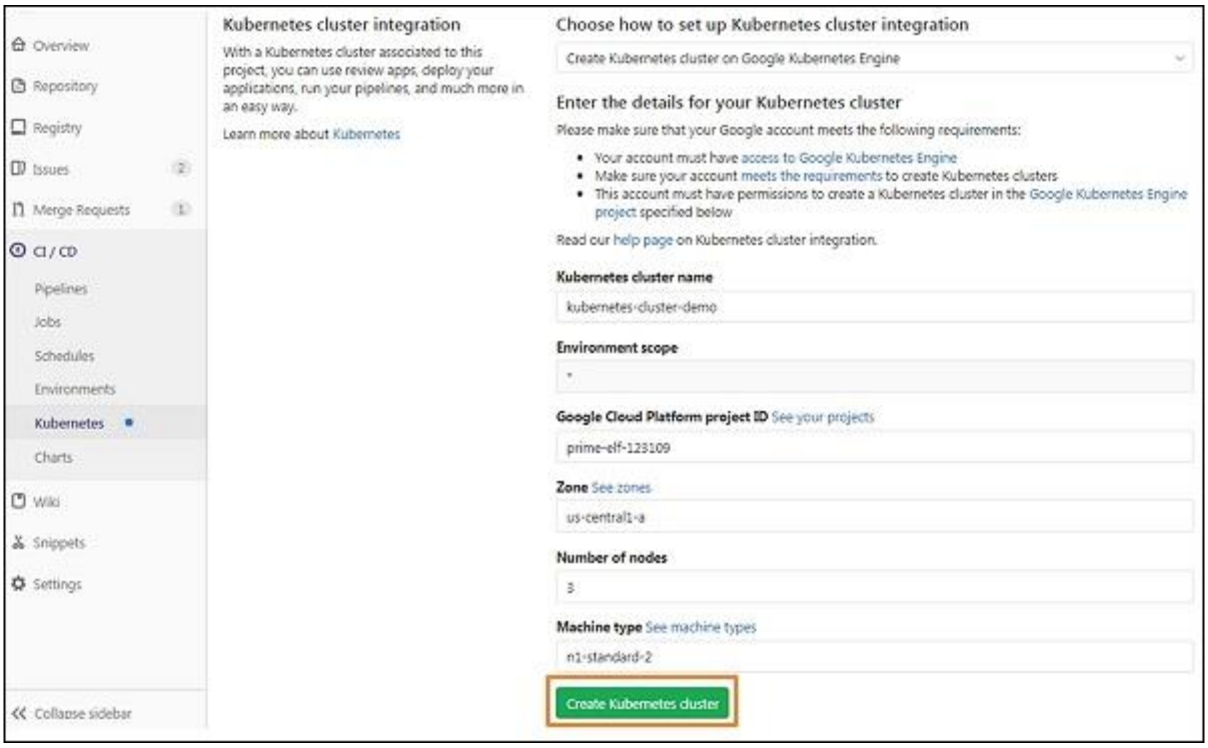
Project info

Project name: My Project

Project ID: prime-elf-123109

Project number: 958043592823

Step 10 − Enter the values in the fields for your Kubernetes cluster along with the Google Cloud Platform project ID and click on the *Create Kubernetes cluster* button −



Kubernetes cluster integration

With a Kubernetes cluster associated to this project, you can use review apps, deploy your applications, run your pipelines, and much more in an easy way.

Learn more about Kubernetes

Choose how to set up Kubernetes cluster integration

Create Kubernetes cluster on Google Kubernetes Engine

Enter the details for your Kubernetes cluster

Please make sure that your Google account meets the following requirements:

- Your account must have access to Google Kubernetes Engine
- Make sure your account meets the requirements to create Kubernetes clusters
- This account must have permissions to create a Kubernetes cluster in the Google Kubernetes Engine project specified below

Read our help page on Kubernetes cluster integration.

Kubernetes cluster name
kubernetes-cluster-demo

Environment scope
*

Google Cloud Platform project ID See your projects
prime-elf-123109

Zone See zones
us-central1-a

Number of nodes
3

Machine type See machine types
n1-standard-2

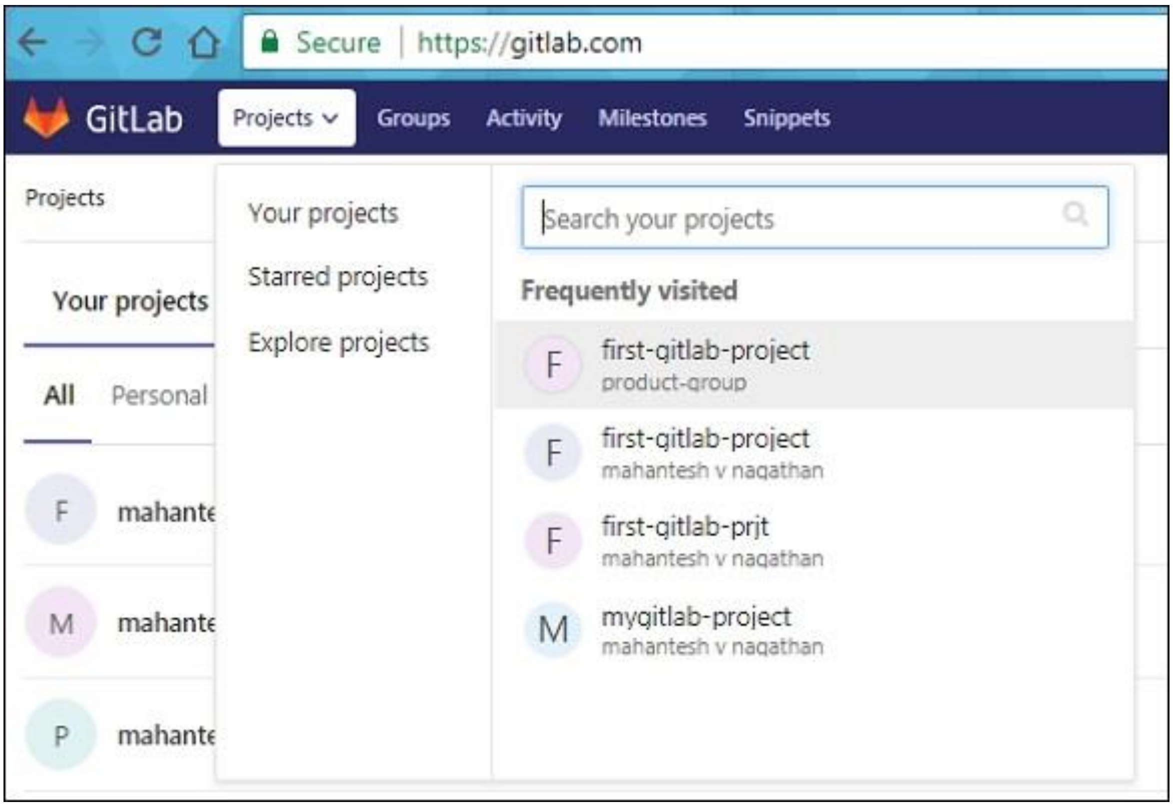
Lab 17: Cycle Analytics

Description

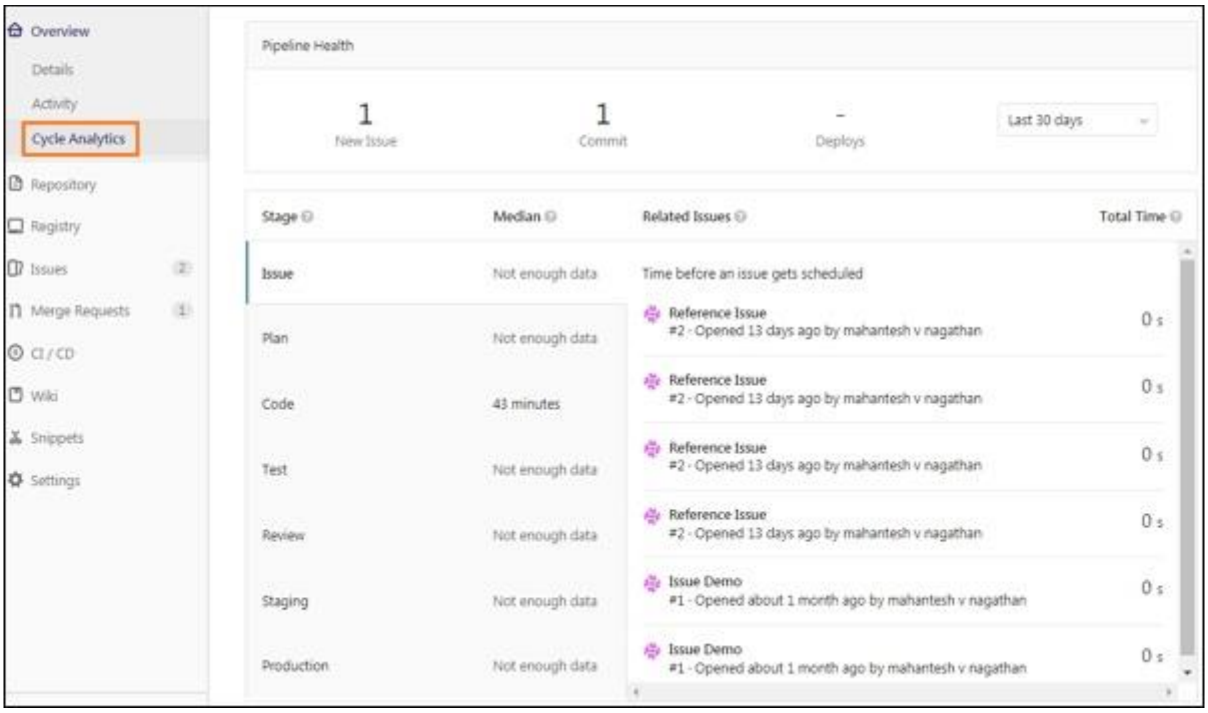
Cycle Analytics specifies how much time taken by the team to complete the each stage in their workflow and allows GitLab to store data of development efforts in one central data store.

The cycle analytics page can be found under the *Overview* section.

Step 1 − Login to your GitLab account and go to your project −



Step 2 − Click on the *Cycle Analytics* option under *Overview* tab which will open the screen as shown below −



1	1	-
New Issue	Commit	Deploys

Stage	Median	Related Issues	Total Time
Issue	Not enough data	Time before an issue gets scheduled	
Plan	Not enough data	Reference Issue #2 · Opened 13 days ago by mahantesh v nagathan	0 s
Code	43 minutes	Reference Issue #2 · Opened 13 days ago by mahantesh v nagathan	0 s
Test	Not enough data	Reference Issue #2 · Opened 13 days ago by mahantesh v nagathan	0 s
Review	Not enough data	Reference Issue #2 · Opened 13 days ago by mahantesh v nagathan	0 s
Staging	Not enough data	Issue Demo #1 · Opened about 1 month ago by mahantesh v nagathan	0 s
Production	Not enough data	Issue Demo #1 · Opened about 1 month ago by mahantesh v nagathan	0 s

The cycle analytics contains following stages −

- **Issue** − It specifies how much time taken to solve an issue.
- **Plan** − It specifies the time between pushing first commit to branch and action took for previous stage.

- **Code −** It specifies the time between pushing first commit to branch and created merge request for that commit.
- **Test −** It specifies how much time need to GitLab CI/CD to test the code.
- **Review −** It specifies time taken to review the merge request.
- **Staging −** It defines the time spent between merging and deploying to production.
- **Production −** It specifies the time taken to complete the entire process, from creating an issue to deploying code to production.

Lab 18: Container Registry

Description

Container registry is a storage and content delivery system, which stores their Docker (it is database of predefined images used to run applications.) images.

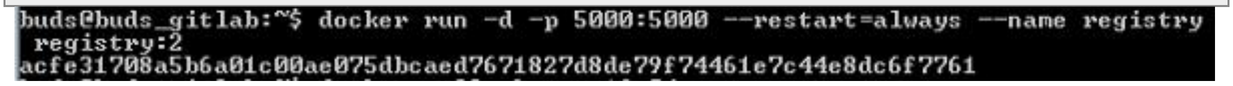
Deploying the Registry

You can deploy the registry by using the below commands −

Step 1 − First, login to your GitLab server using SSH (Secure Shell).

Step 2 − Now start the registry container by using below command −

```
$ docker run -d -p 5000:5000 --restart = always --name registry
registry:2
```

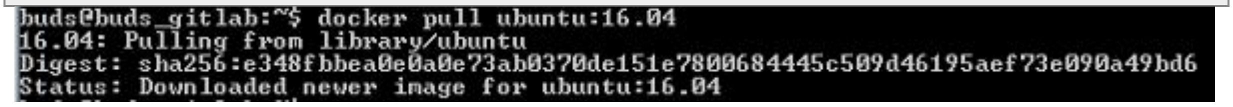


```
buds@buds_gitlab:~$ docker run -d -p 5000:5000 --restart=always --name registry
 registry:2
acfe31708a5b6a01c00ae075dbcaed7671827d8de79f74461e7c44e8dc6f7761
```

The *-p 5000:5000* specifies first part as host port and second part as port within the container. The *--restart* = *always* flag restarts the registry automatically when Docker restarts. The *registry:2* is defined as an image.

Step 3 − Now, pull the image from Docker hub to your registry −

```
$ docker pull ubuntu:16.04
```



```
buds@buds_gitlab:~$ docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
Digest: sha256:e348fbbea0e0a0e73ab0370de151e7800684445c509d46195aef73e090a49bd6
Status: Downloaded newer image for ubuntu:16.04
```

The above command pulls the *ubuntu:16.04* image from Docker Hub.

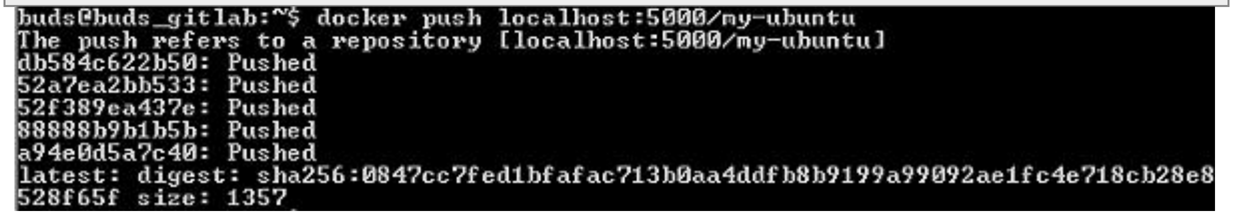
Step 4 − Next, tag the image to point your registry −

```
$ docker tag ubuntu:16.04 localhost:5000/my-ubuntu
```

Here, we are tagging the *localhost:5000/my-ubuntu* image for an existing *ubuntu:16.04* image.

Step 5 − Push the image to local registry which is executing at localhost:5000.

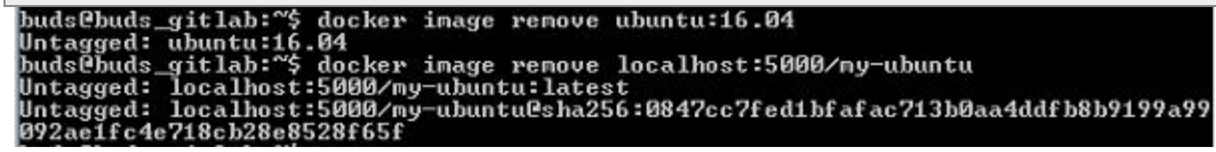
```
$ docker push localhost:5000/my-ubuntu
```



```
buds@buds_gitlab:~$ docker push localhost:5000/my-ubuntu
The push refers to a repository [localhost:5000/my-ubuntu]
db584c622b50: Pushed
52a7ea2bb533: Pushed
52f389ea437e: Pushed
88888b9b1b5b: Pushed
a94e0d5a7c40: Pushed
latest: digest: sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8
528f65f size: 1357
```

Step 6 − Now remove the cached (*ubuntu:16.04* and *localhost:5000/my-ubuntu*) images from the registry −

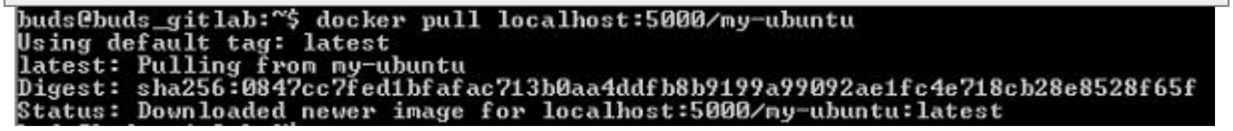
```
$ docker image remove ubuntu:16.04
$ docker image remove localhost:5000/my-ubuntu
```



```
buds@buds_gitlab:~$ docker image remove ubuntu:16.04
Untagged: ubuntu:16.04
buds@buds_gitlab:~$ docker image remove localhost:5000/my-ubuntu
Untagged: localhost:5000/my-ubuntu:latest
Untagged: localhost:5000/my-ubuntu@sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99
092ae1fc4e718cb28e8528f65f
```

Step 7 − Pull back the *localhost:5000/my-ubuntu* image from local registry −

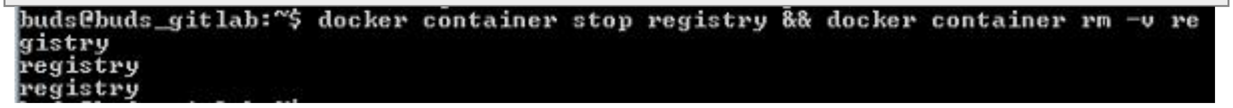
```
$ docker pull localhost:5000/my-ubuntu
```



```
buds@buds_gitlab:~$ docker pull localhost:5000/my-ubuntu
Using default tag: latest
latest: Pulling from my-ubuntu
Digest: sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f
Status: Downloaded newer image for localhost:5000/my-ubuntu:latest
```

Step 8 − Now stop the registry and remove the data −

```
$ docker container stop registry && docker container rm -v
registry
```



```
buds@buds_gitlab:~$ docker container stop registry && docker container rm -v re
gistry
registry
registry
```

Release Notes

B. TECH CSE with Specialization in DevOps Release Components.

Current Release Version.

3.0.0

Current Release Date.

23 May 2022

Course Description.

Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications

Bugs reported
Not applicable for version 2.0.0