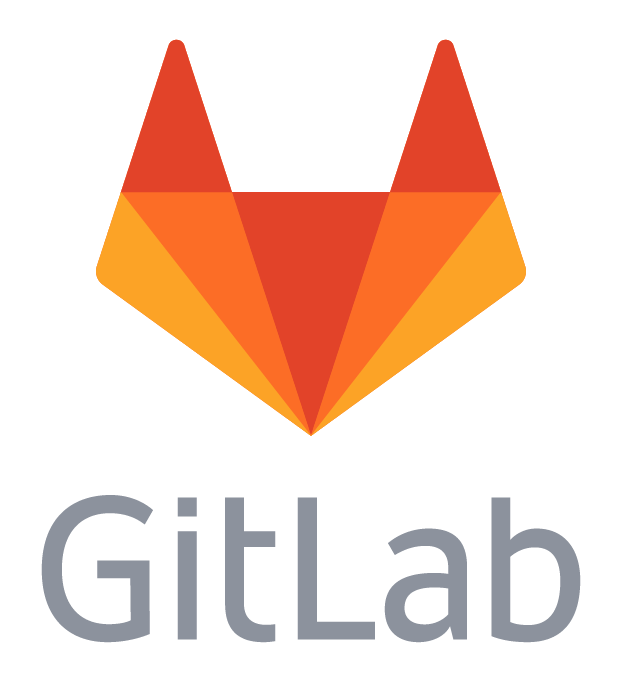
**Learn GitLab** -



Signup gitlab with respective github account :- A screenshot of a login page

Description automatically generated

**Step 1: Visit GitLab Website**

1. Open your web browser and navigate to the GitLab website: <https://gitlab.com/>

**Step 2: Registration**

1. Click on the "Sign Up" or "Register" button, usually located in the upper right corner of the page.
2. Fill in the required information, including:
   * Username: Choose a unique username.
   * Email: Provide a valid email address.
   * Password: Create a strong password.
3. Solve any CAPTCHA or security checks if prompted.
4. Review and agree to the terms of service and privacy policy.
5. Click on the "Register" or "Create account" button.

**Step 3: Email Verification**

1. Check your email inbox for a verification email from GitLab.
2. Open the email and click on the verification link provided.

**Step 4: Account Verification**

1. You may be prompted to complete additional steps for account verification, depending on GitLab's security measures.

**Step 5: Account Setup**

1. After verification, return to the GitLab website and log in with your newly created credentials.
2. Follow any on-screen prompts to set up your profile, including adding a profile picture and other optional information.

**Set up GitLab :-**

### 1. SSH Key Setup (Optional):

#### **Purpose:**

* Securely authenticate with GitLab using SSH keys.
* Convenient and secure way to connect to repositories without entering passwords.

#### **Steps:**

1. Generate SSH Key:
   * Open a terminal on your local machine.
   * Use the following command to generate an SSH key: **ssh-keygen -t rsa -C "your\_email@example.com"**
   * Follow the on-screen prompts.
2. Add SSH Key to GitLab:
   * Copy the SSH key to your clipboard: **cat ~/.ssh/id\_rsa.pub** (replace with your actual key file)
   * In GitLab, go to "Settings" > "SSH Keys."
   * Paste your SSH key into the "Key" field and click "Add Key."
3. Test the SSH Connection:
   * In the terminal, run: **ssh -T git@gitlab.com**
   * You should receive a message indicating a successful connection.

### 2. Two-Factor Authentication (2FA):

#### **Purpose:**

* Adds an extra layer of security to your GitLab account.
* Requires a second verification step in addition to your password.

#### **Steps:**

1. In GitLab, go to "Settings" > "Account" > "Two-Factor Authentication."
2. Follow the on-screen instructions to enable 2FA.
   * You can use a mobile app (e.g., Google Authenticator) or recovery codes.
3. Save recovery codes in a secure place. These codes can be used to access your account if you lose access to your 2FA device.

### 3. Account Security Settings:

#### **Purpose:**

* Additional security measures to protect your GitLab account.

#### **Steps:**

1. In GitLab, go to "Settings" > "Account."
2. Consider the following security options:
   * **Password**: Regularly update your password to a strong and unique one.
   * **Account Lockout**: Enable account lockout after a certain number of failed login attempts.
   * **Session Limit**: Set a session limit to automatically log out after a period of inactivity.

### 4. Explore GitLab Features:

#### **Purpose:**

* Understand and utilize key features of GitLab for effective collaboration.

#### **Steps:**

1. Explore the GitLab dashboard and menu options.
2. Create a new project/repository.
3. Learn about issues, merge requests, CI/CD pipelines, and other collaboration features.

### 5. Read GitLab Documentation:

#### **Purpose:**

* Stay informed about GitLab's capabilities and best practices.

#### **Steps:**

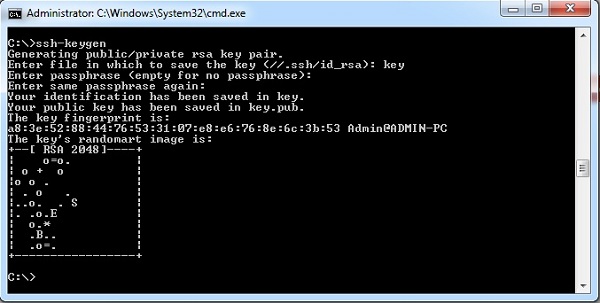
1. Visit the GitLab documentation.

## **Creating SSH Key**

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

C:\−ssh-keygen

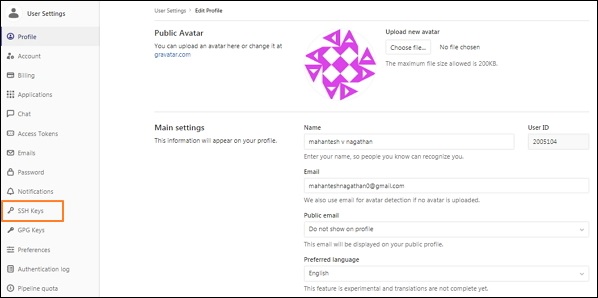
It will prompt for 'Enter file in which to save the key (//.ssh/id\_rsa):', just type file name and press enter. Next a prompt to enter password shows 'Enter passphrase (empty for no passphrase):'. Enter some password and press enter. You will see the generated SSH key as shown in the below image −



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



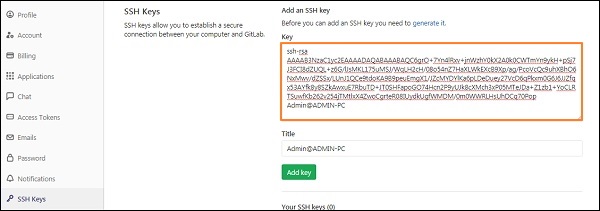
**Step 3** − To create SSH key, click on the SSH keys tab at left side of the menu.



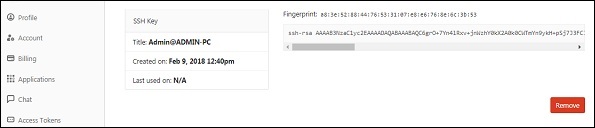
**Step 4** − Now go to C drive, you will see the file with *.pub* extension which was generated in the first step.

GitLab SSH key

**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 −



**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 −



**Setup Gitbash with two different gmail account:-**

If you want to use two different GitLab accounts on the same machine using Git Bash, you can do this by configuring multiple SSH keys and associated profiles. Here are the steps:

### 1. Generate SSH Keys for Each Account:

ssh-keygen -t rsa -C "your\_email\_account1@example.com" -f ~/.ssh/id\_rsa\_account1

Follow the on-screen prompts to generate the key.

ssh-keygen -t rsa -C "your\_email\_account2@example.com" -f ~/.ssh/id\_rsa\_account2

### 2. Add SSH Keys to the SSH Agent:

# Add the first SSH key to the SSH agent

ssh-add ~/.ssh/id\_rsa\_account1

# Add the second SSH key to the SSH agent

ssh-add ~/.ssh/id\_rsa\_account2

### 3. Add SSH Keys to GitLab Accounts:

1. Copy the contents of **~/.ssh/id\_rsa\_account1.pub** and add it to the SSH keys section of your first GitLab account.
2. Copy the contents of **~/.ssh/id\_rsa\_account2.pub** and add it to the SSH keys section of your second GitLab account.

### 4. Configure SSH Config File:

Create or edit the SSH config file (**~/.ssh/config**) using a text editor like Nano or Vim:

nano ~/.ssh/config

Add the following configuration:

# Account 1

Host gitlab.com-account1

HostName gitlab.com

User git

IdentityFile ~/.ssh/id\_rsa\_account1

# Account 2

Host gitlab.com-account2

HostName gitlab.com

User git

IdentityFile ~/.ssh/id\_rsa\_account2

### 5. Test the Connections:

# Test Account 1

ssh -T git@gitlab.com-account1

# Test Account 2

ssh -T git@gitlab.com-account2

### 6. Configure Git User for Each Repository:

For each repository, set the Git user to the corresponding GitLab account:

# For Account 1

git config user.name "Your Name Account 1"

git config user.email "your\_email\_account1@example.com"

# For Account 2

git config user.name "Your Name Account 2"

git config user.email "your\_email\_account2@example.com"

Now, you should be able to interact with GitLab repositories using different accounts on the same machine. When cloning a repository, make sure to use the appropriate SSH URL, such as **git@gitlab.com-account1:username/repo.git** or **git@gitlab.com-account2:username/repo.git**.

**Gitlab – git commands:-**

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

**Setup runner:-**

## **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](https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-386.exe) or [amd64](https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe) 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/):

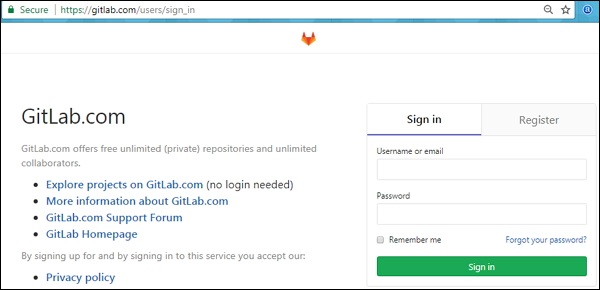
https://gitlab.com

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

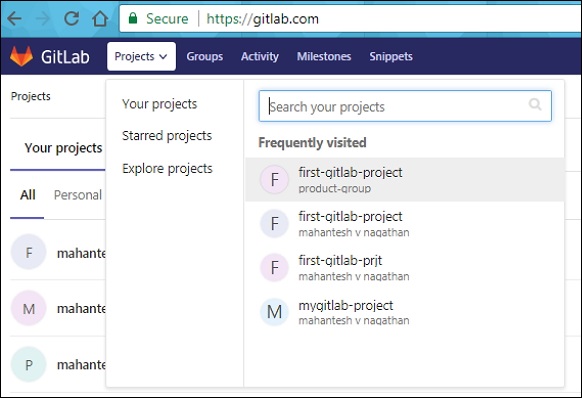
Please enter the gitlab-ci token for this runner:

xxxxx

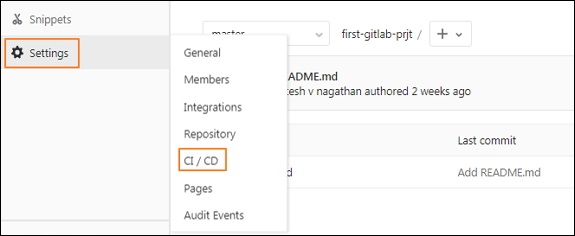
* To get the token, login to your GitLab account −



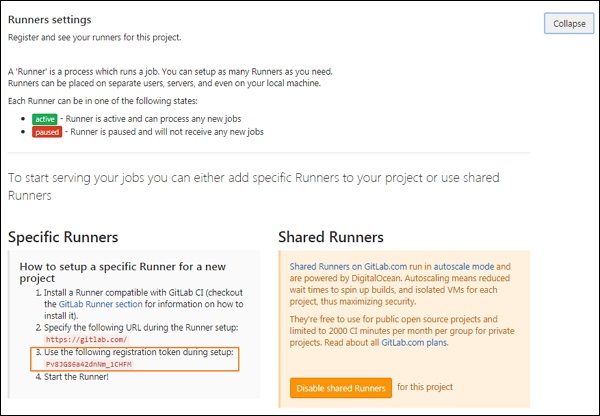
* Now go to your project −



* 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 −



**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

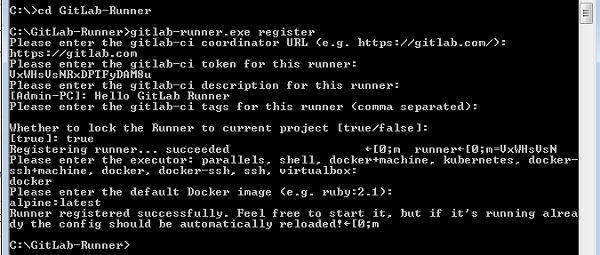
We have used the selector as 'docker' which creates build environment and manages the dependencies easily for developing the project.

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

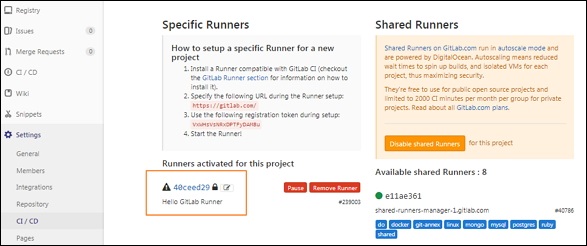
Please enter the default Docker image (e.g. ruby:2.1):

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 −



**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.



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

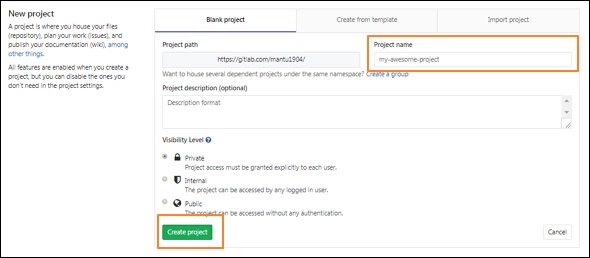
[runners.cache]

**Create project :-**

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

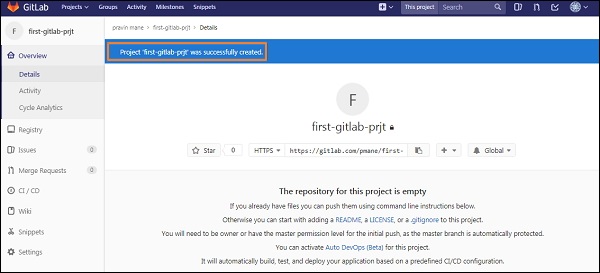
GitLab Create Project

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



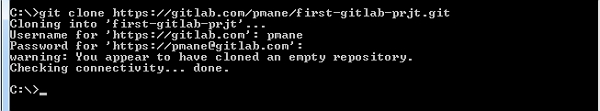
Enter the project name, description for the project, visibility level (accessing the project's visibility in publicly or internally) and click on the *Create project* button.

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



## **Push the Repository to Project**

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



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 −

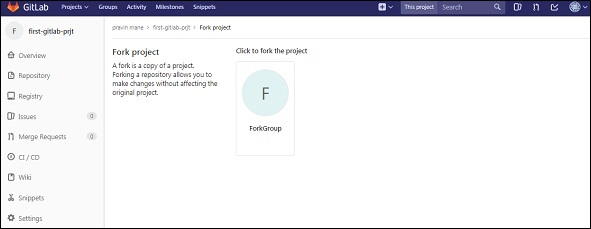


## **Forking a Project:-**

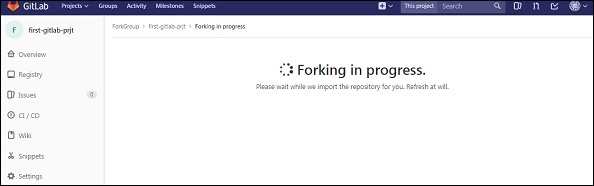
**Step 1** − To fork a project, click on the *Fork* button as shown below −



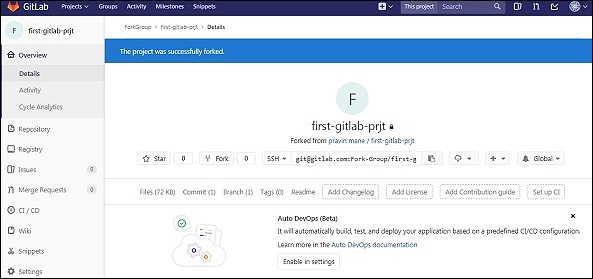
**Step 2** − After forking the project, you need to add the forked project to a fork group by clicking on it −



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

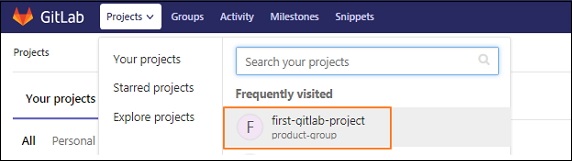


**Step 4** − It will display the success message after completion of forking the project process −



## **Creating a Branch**

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



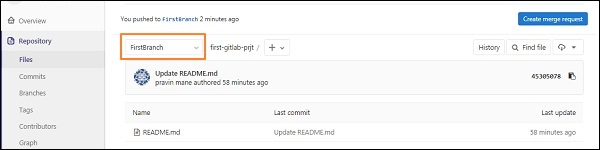
**Step 2** − To create a branch, click on the *Branches* option under the *Repository* section and click on the *New branch* button.



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

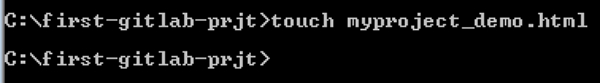


**Step 4** − After creating branch, you will get a below screen along with the created branch.

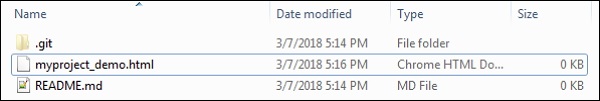


## **Creating a file using Command Line Interface**

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



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

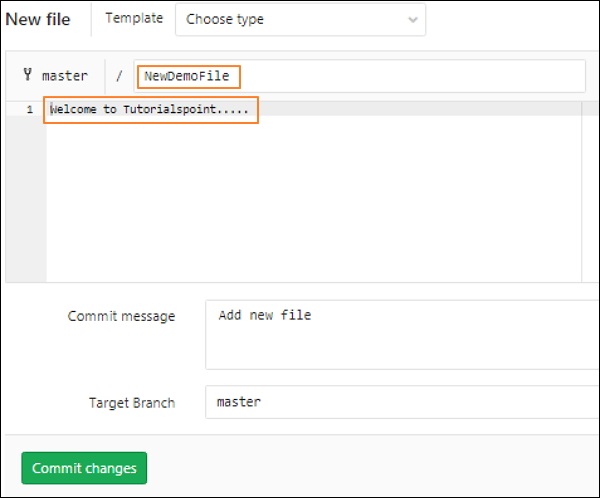


## **Creating a file using Web Interface**

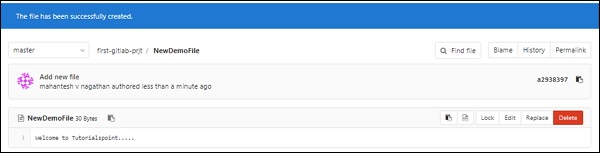
**Step 1** − You can create a new file, by clicking on the '+' button which is at the right side of the branch selector in the dashboard −



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

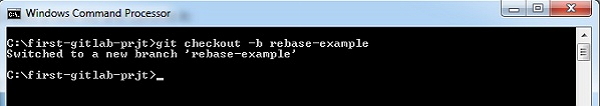


**Step 3** − Now you will get a successful message after creating the file as shown below −



## **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 −



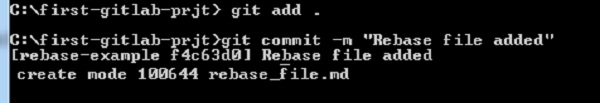
The flag *-b* indicates new branch name.

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

GitLab Rebase Operation

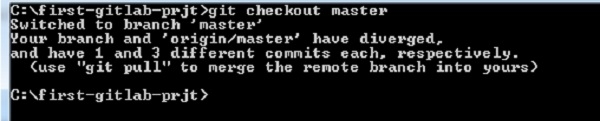
The content 'Welcome to Tutorialspoint' will be added to the *rebase\_file.md* file.

**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 −

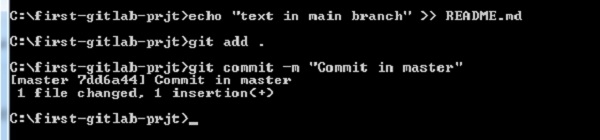


The flag *-m* is used for adding a message on the commit.

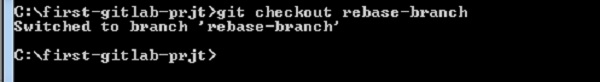
**Step 4** − Now, switch to the 'master' branch. You can fetch the remote branch(*master* is a branch name) by using the *git checkout* command −



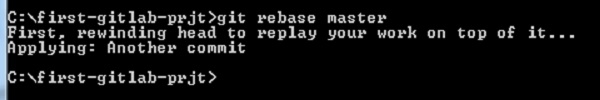
**Step 5** − Next, create an another new file, add some content to that file and commit it in the *master* branch.



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



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



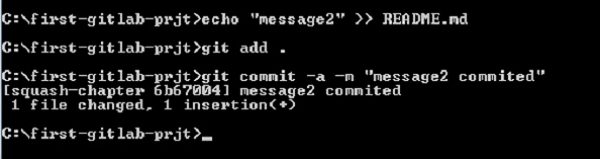
## **Steps for Squashing Commits**

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

GitLab Squashing Commits

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 −

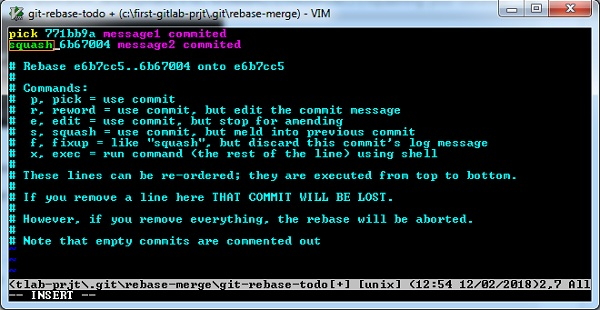


**Step 3** − Now, squash the above two commits into one commit by using the below command −

$ 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).



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 −



**CI/CD pipeline:-**

### Create a YAML Configuration File

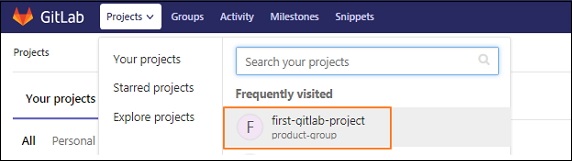
Create a file named **.gitlab-ci.yml** in the root of the project folder to define your CI/CD jobs.

GitLab takes a snapshot of data when each release is created and saves this data as a JSON file called **release evidence**, which contains information such as the name, tag name, description, project details, and reports artifact if it has been included in the **.gitlab-ci.yml** file.

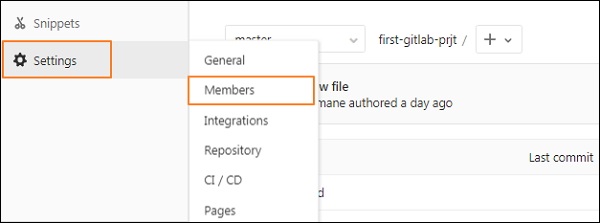
**GitLab Users and Groups**

## **Steps for Adding User**

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



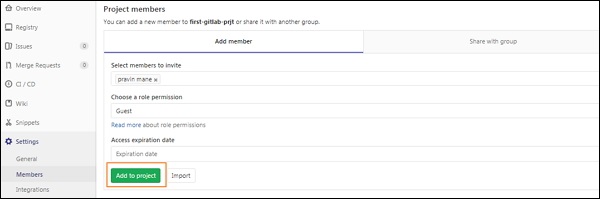
**Step 2** − Next, click on the *Members* option under *Settings* tab −



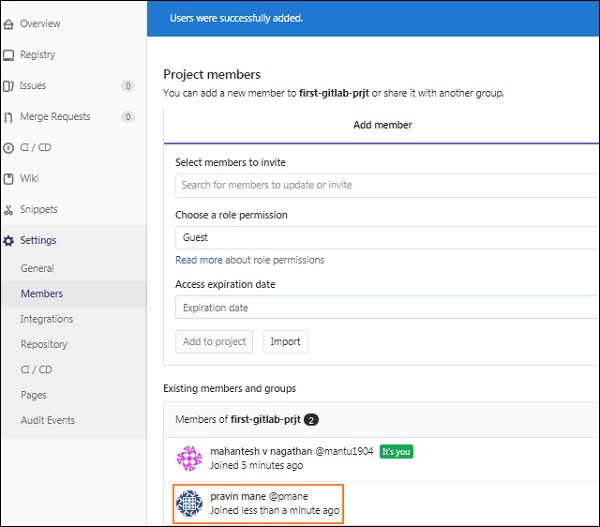
**Step 3** − It will open the below screen to add the member to your project −



**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 −

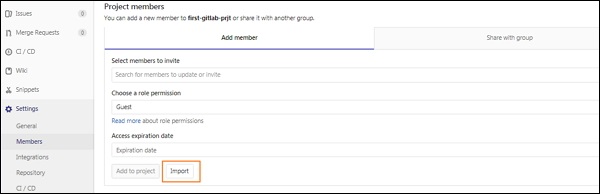


**Step 5** − Next, you will get a successful message after adding user to the project.

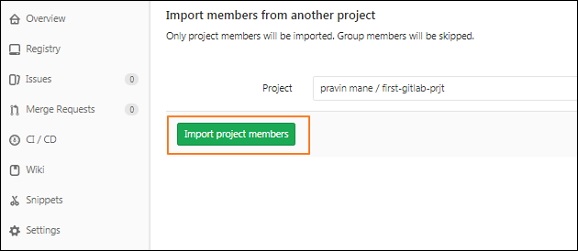


The highlighted box in the above image indicates, a new user has been added to the project −

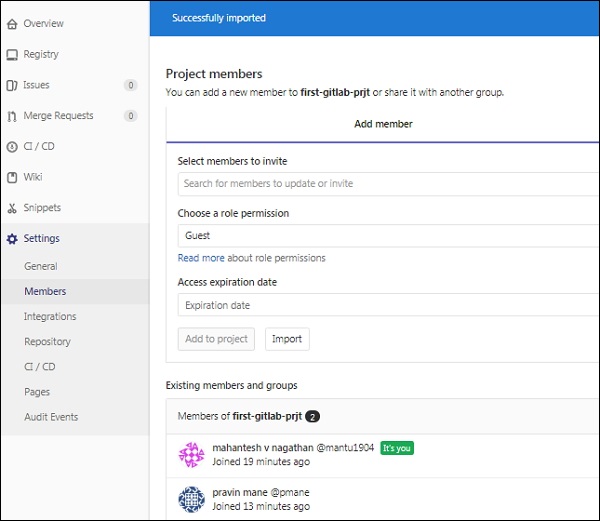
**Step 6** − You can also add user to the project by clicking on the *Import* button −



**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 −

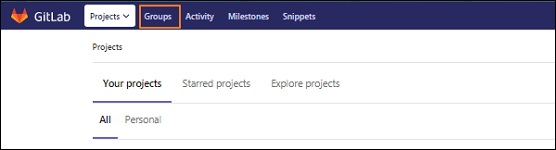


**Step 8** − You will get a success message after importing user to the project −



## **Steps for Creating Group**

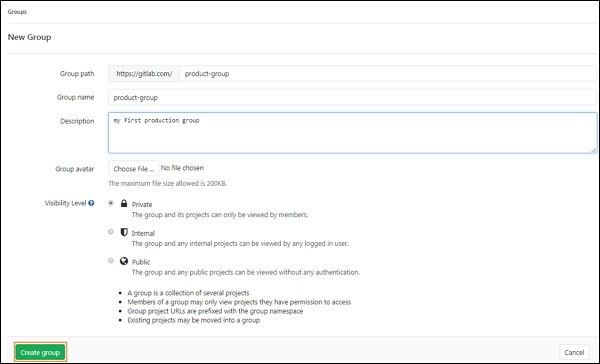
**Step 1** − Login to your GitLab account and click on the *Groups* menu −



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



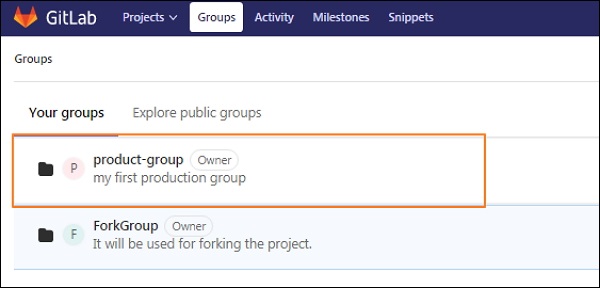
**Step 3** − Enter the *Group name*, *Description*, *visibility level*(Private/Public/Internal) and also you can set the image for the group of your choice which should be within 200kb in size. Now click on the *Create group* button.



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

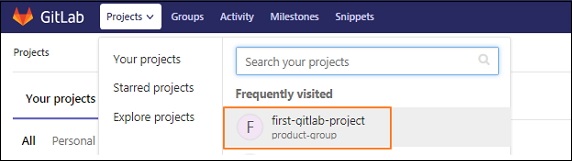


**Step 5** − Now, go back to your *Groups* section and you will see the created group in the list −

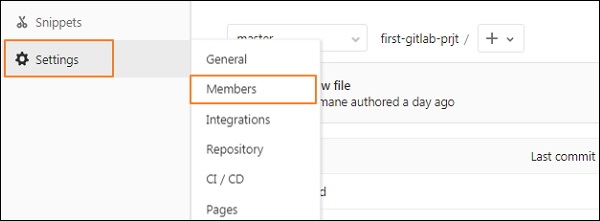


## **Steps for Removing User**

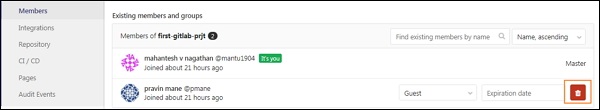
**Step 1** − Login to your GitLab account and go to your project under *Projects* section −



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



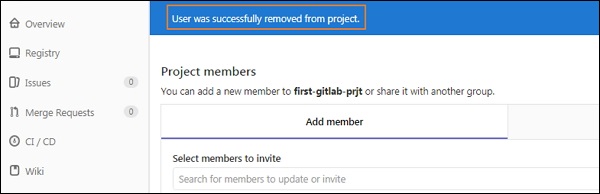
**Step 3** − You will see the list of users under *Existing members and groups* section and click on the delete option at right side to remove the user from project −



**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.

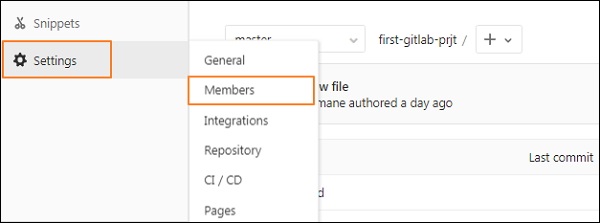


**Step 5** − Now, it will display the success message after removing the user from the project as shown in the image below −



## **Steps for creating User Permissions**

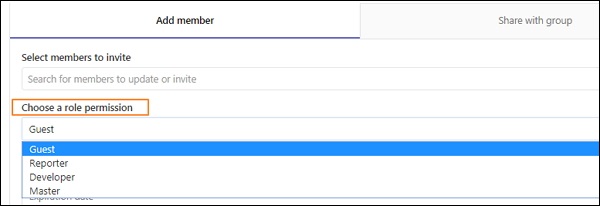
**Step 1** − Login to your GitLab account and click on the *Members* option under *Settings* tab −



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



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



You can see the [Adding users](https://www.tutorialspoint.com/gitlab/gitlab_adding_users.htm) 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.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

# **GitLab Instance Management**

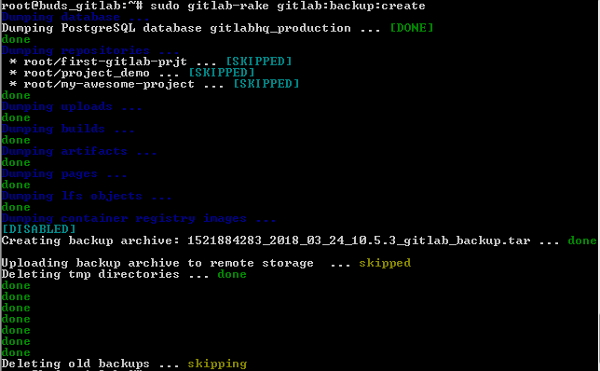
# **GitLab - Create Backup**

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).

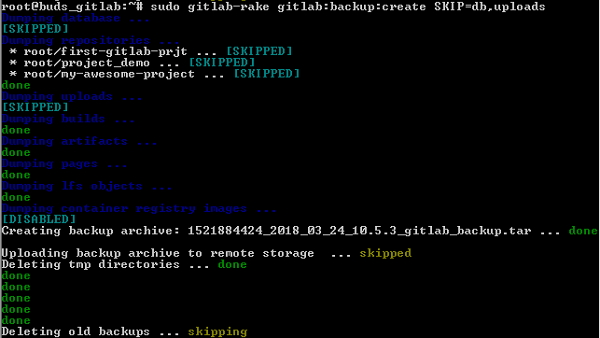
**Step 2** − Create the backup of GitLab by using the below command −

sudo gitlab-rake gitlab:backup:create

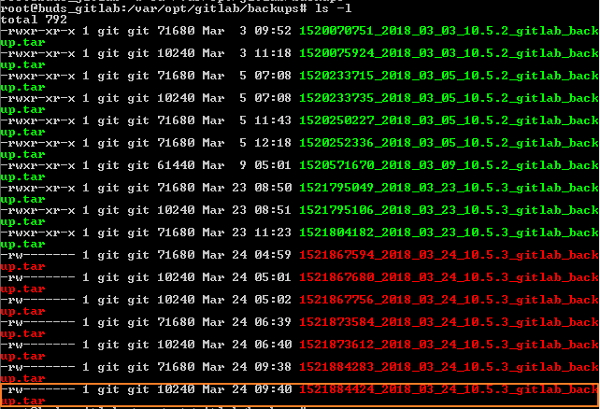


**Step 3** − You can exclude some directories from the backup by adding environment variable *SKIP* as shown below −

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



**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 −



# **GitLab - Restore Backup**

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](https://www.tutorialspoint.com/gitlab/gitlab_create_backup.htm) 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

GitLab Restore Backup

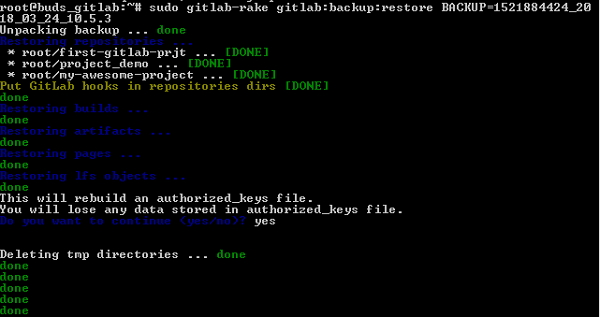
The above commands can also be used to free up some memory temporarily by shutting down them.

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

sudo gitlab-ctl status

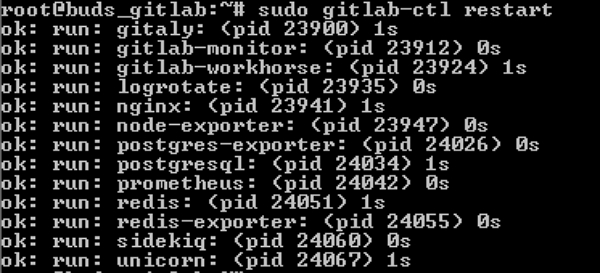
**Step 6** − Now, restore the backup by using the timestamp of the backup copy −

sudo gitlab-rake gitlab:backup:restore BACKUP = 1521884424\_2018\_03\_24\_10.5.3



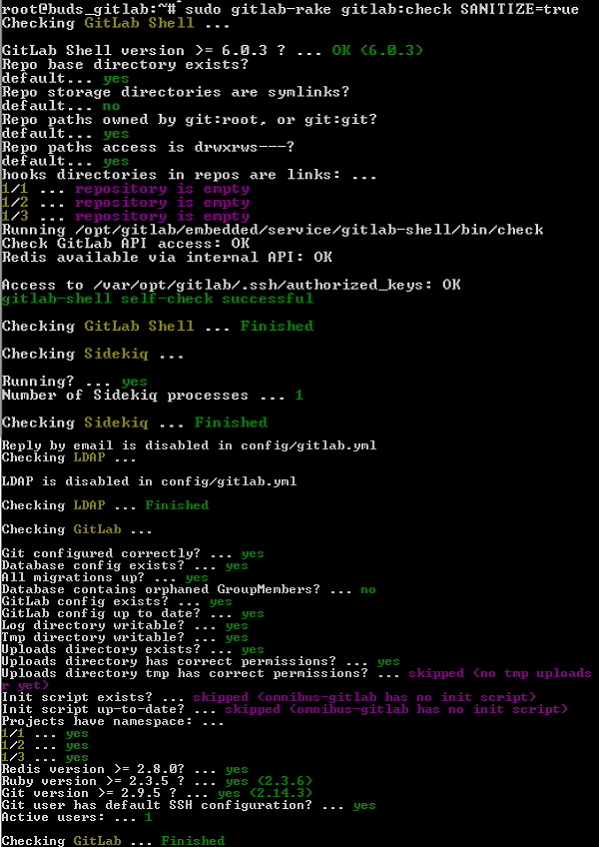
**Step 7** − Restart the GitLab components by using the below command −

sudo gitlab-ctl restart



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

sudo gitlab-rake gitlab:check SANITIZE = true



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.

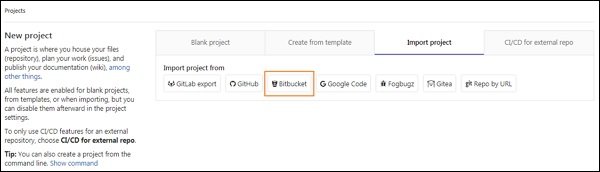
# **GitLab - Import Repository**

In this chapter, we will discuss about how to import a repository from Bitbucket to GitLab −

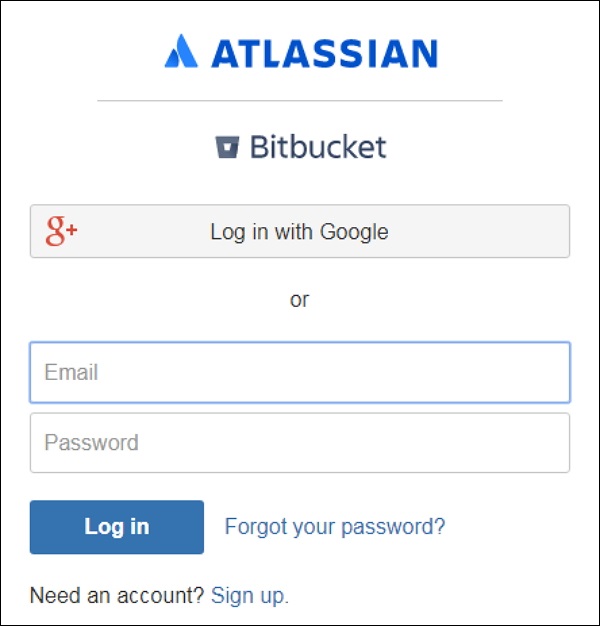
**Step 1** − Login to your GitLab account and click on the *New project* button in the dashboard −

GitLab Create Project

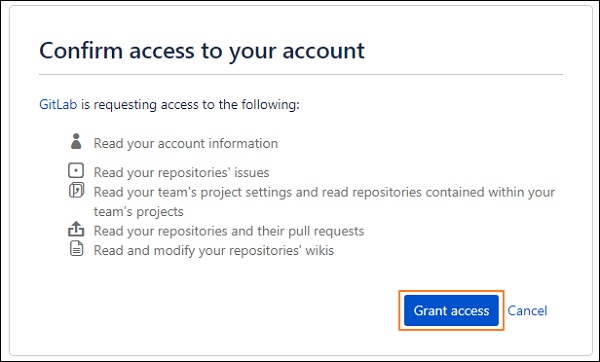
**Step 2** − Click on the *Bitbucket* button under *Import project* tab −



**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 −

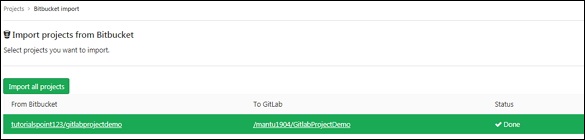


You need to grant the access to read the account information, repository issues, project settings, and modify the repositories.

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



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



**GitLab Continuous Integration**

# **GitLab - CI/CD Variables**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 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 |

# **GitLab CI - Permissions**

## **User Permissions**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

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 labels |
| 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 |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

## **Job Permissions**

The following table shows job 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 |

**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.

# **GitLab CI - Configuring 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](https://www.tutorialspoint.com/gitlab/gitlab_installation.htm) 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.

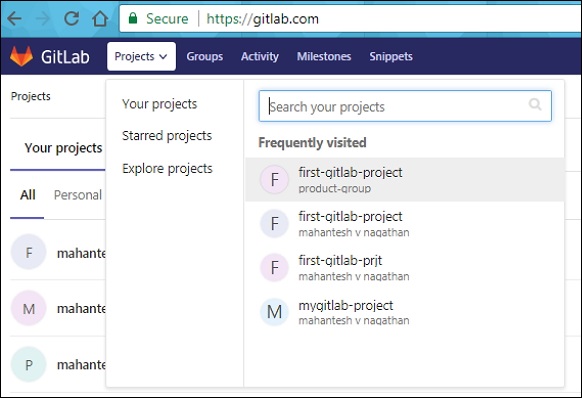
You can register a specific runner by using project registration token. The registering a specific runner is explained in the [GitLab Installation](https://www.tutorialspoint.com/gitlab/gitlab_installation.htm) chapter from step 1 to 12 under the *Installation of GitLab on Windows* section.

## **Locking a specific Runner**

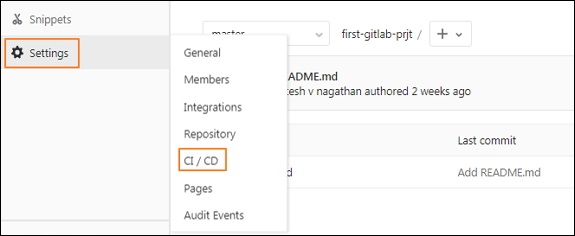
You can lock a specific runner from being enabled for other projects. To do this, you need to register a runner which is explained in the [GitLab Installation](https://www.tutorialspoint.com/gitlab/gitlab_installation.htm) chapter from step 1 to 12 under the *Installation of GitLab on Windows* section.

To lock runner, execute the below steps −

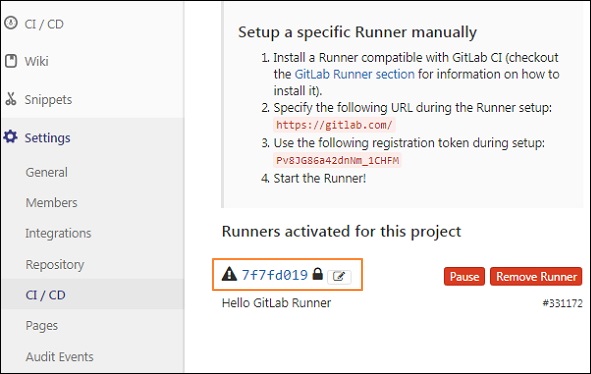
**Step 1** − Login to your GitLab account and go to your project −



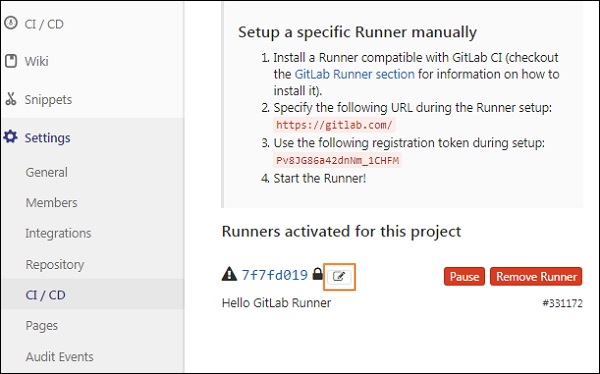
**Step 2** − Click on the CI/CD option under Settings tab and expand the Runners Settings option. −



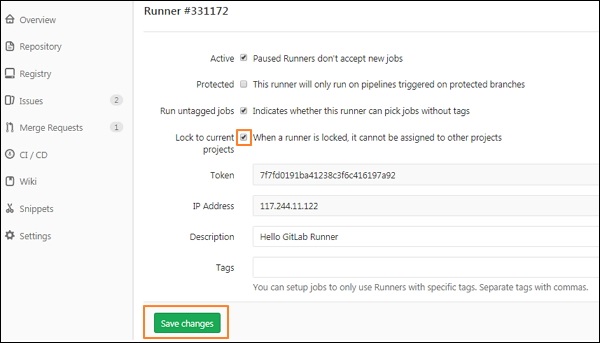
**Step 3** − Under Runners Settings section, you will see the activated Runners for the project −



**Step 4** − Now click on the pencil button −

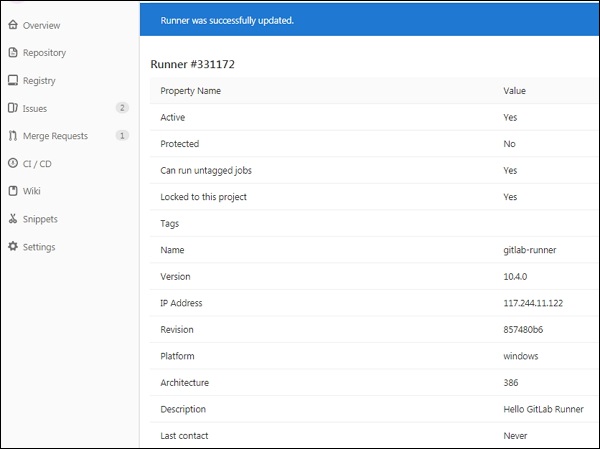


**Step 5** − Next it will open the Runner screen and check the *Lock to current projects* option −



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

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

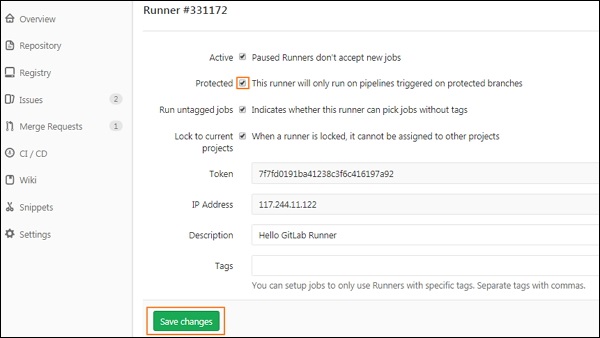


## **Protected Runners**

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 −



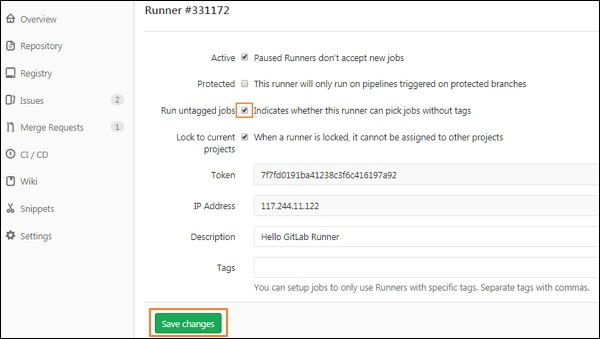
Click on the *Save changes* button to take the changes effect.

## **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 −



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

# **GitLab CI - Advanced usage of CI**

## **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](https://www.tutorialspoint.com/gitlab/gitlab_ssh_key_setup.htm) chapter.

## **Artifacts**

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

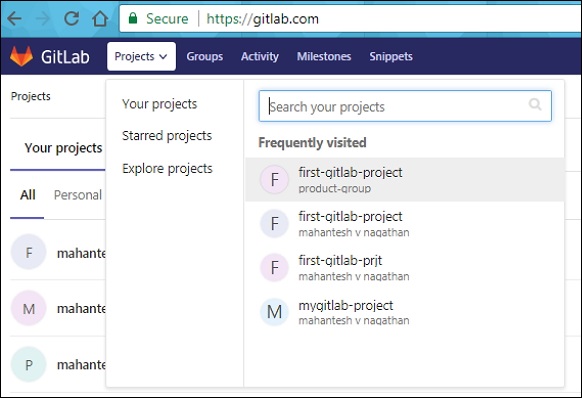
* **artifacts:name** − This directive is used to specify the name of created artifacts archive. It provides unique name for created artifacts archive which is helpful when you are downloading the archive from GitLab.
* **artifacts:when** − This directive is used to upload artifacts when there is a job 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**

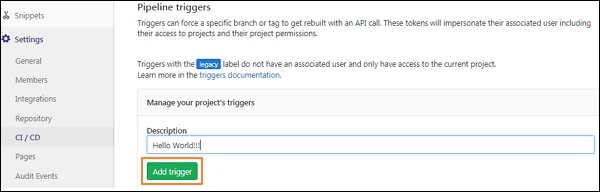
Triggers can force a specific branch or tag to get rebuilt with an API call and triggers with the *legacy* label will have access to the current project.

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

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

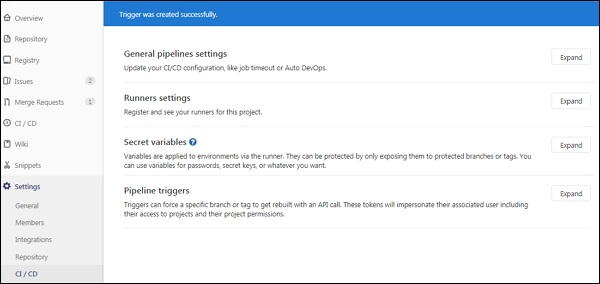


**Step 2** − Click on the *CI/CD* option under *Settings* tab and expand the *Pipeline triggers* option −

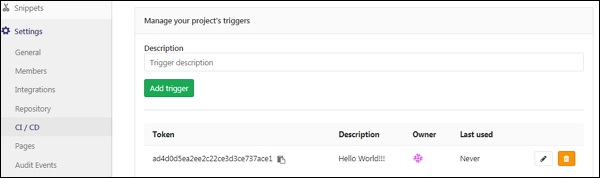


Enter the description for the trigger and click on the *Add Trigger* button.

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



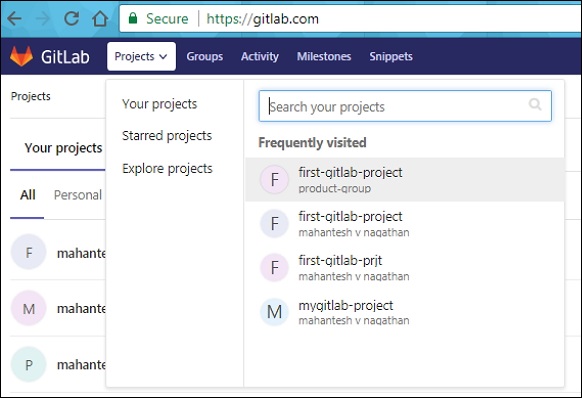
**Step 4** − Now go to *CI/CD* option under *Settings* tab and expand the *Pipeline triggers* option. You will see the newly created trigger along with the token as shown in the image below −



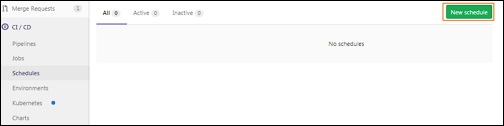
## **Pipeline Schedules**

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

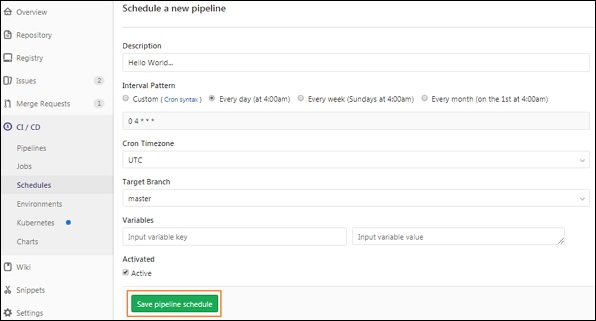
**Step 1** − Login to your GitLab account and go to your project −



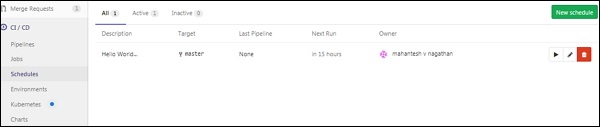
**Step 2** − Click on the *Schedules* option under *CI/CD* tab and click on the *New schedule* button −



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



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

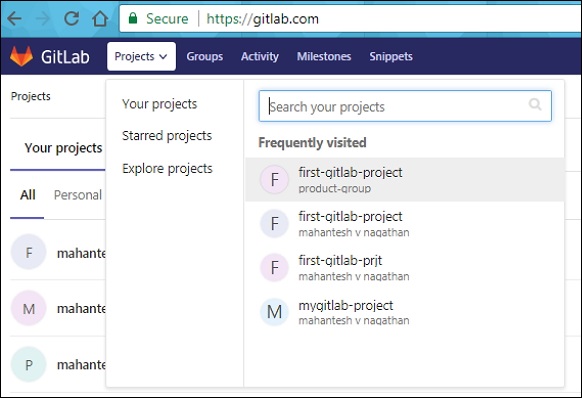


## **Connecting GitLab with a Kubernetes Cluster**

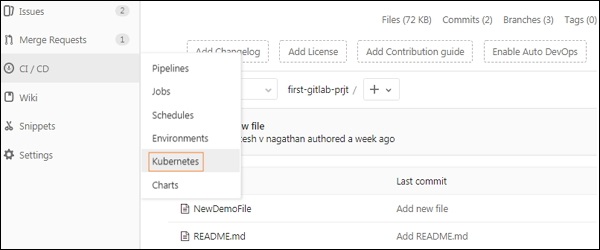
The Kubernetes cluster can be used to review and deploy the applications, running the pipeline etc in an easy method. You can create a new cluster to your project by associating your GitLab account with the Google Kubernetes Engine (GKE).

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

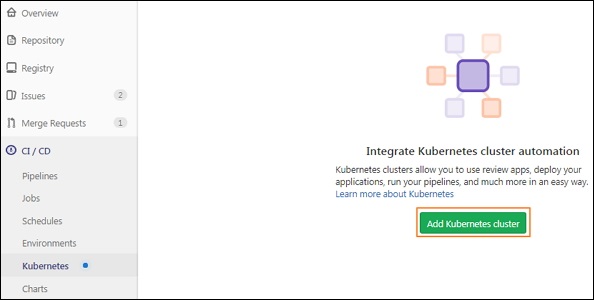
**Step 1** − Login to your GitLab account and go to your project −



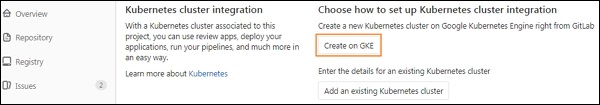
**Step 2** − Click on the *Kubernetes* option under *CI/CD* tab −



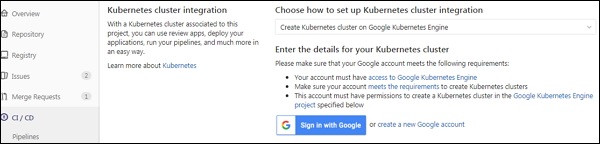
**Step 3** − Next, click on *Add Kubernetes cluster* button −



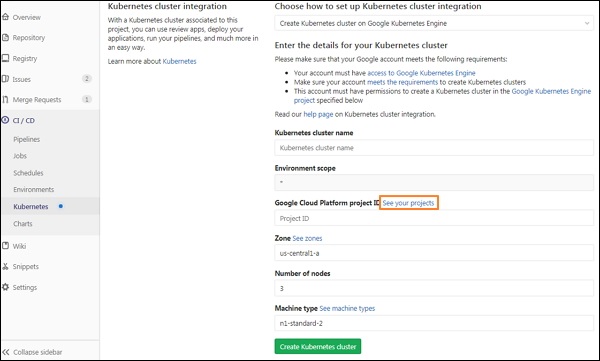
**Step 4** − Click on *Create on GKE* button to create a new Kubernetes cluster on Google Kubernetes Engine −



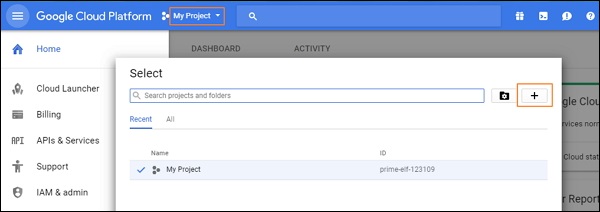
**Step 5** − If you have a Google account, then sign with that account to enter the details for Kubernetes cluster or else create a new Google account −



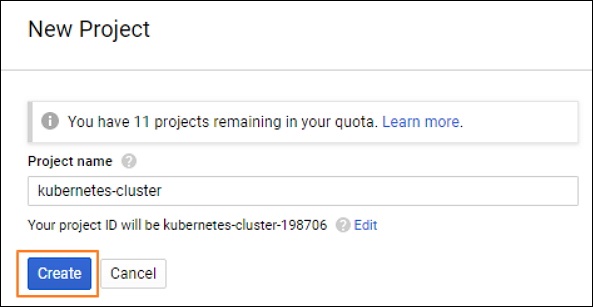
**Step 6** − Now enter the values in the fields for your Kubernetes cluster −



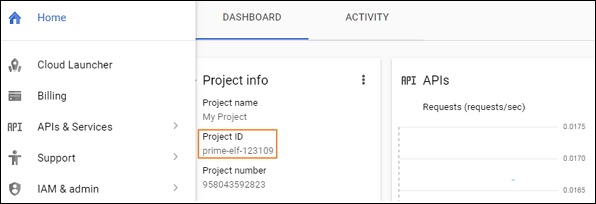
**Step 7** − Before adding values in the fields, you need ID of the project which is created in the Google Cloud Platform console to host the Kubernetes cluster. To create ID, click on the *See your projects* link which is highlighted in the previous image. It will open the below screen, then click on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



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



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



**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 −

# **GitLab Usage of CI**

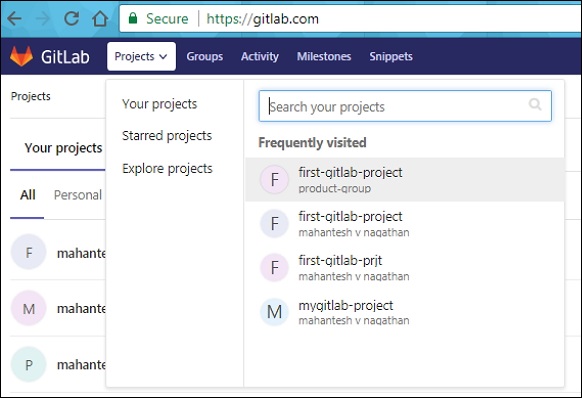
# **GitLab CI - Cycle Analytics**

## **Description**

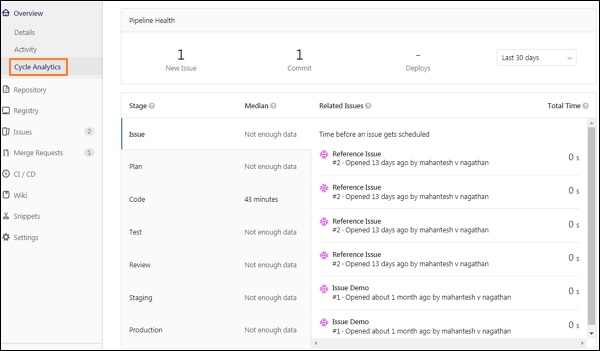
Cycle Analytics specifies how much time taken by the team to complete the each stage in their workflow and allows GitLab to store data of development efforts in one central data store.

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 −



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.

**Kickstart**

# **GitLab CI - 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

GitLab Container Registry

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

GitLab Container Registry

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

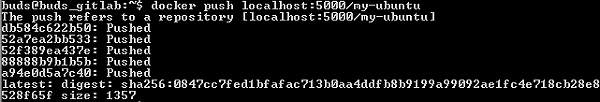
**Step 4** − Next, tag the image to point your registry −

$ 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.

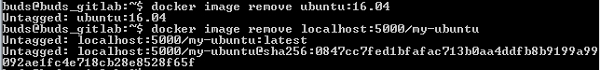
$ docker push localhost:5000/my-ubuntu



**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



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

$ docker pull localhost:5000/my-ubuntu

GitLab Container Registry

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

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

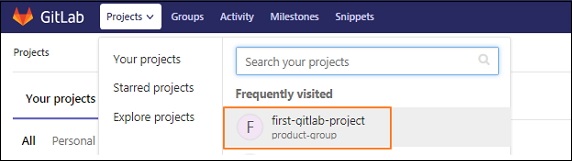
GitLab Container Registry

**GitLab Issue Tracker**

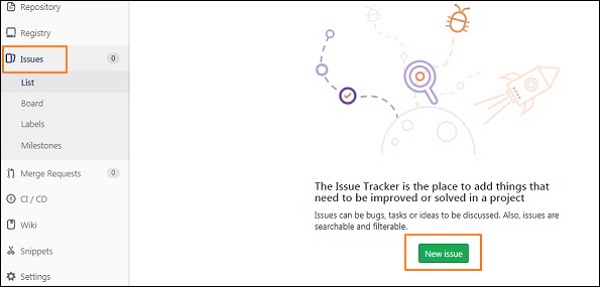
# **GitLab - Create Issue**

In this chapter, we will discuss about how to create an issue in a project −

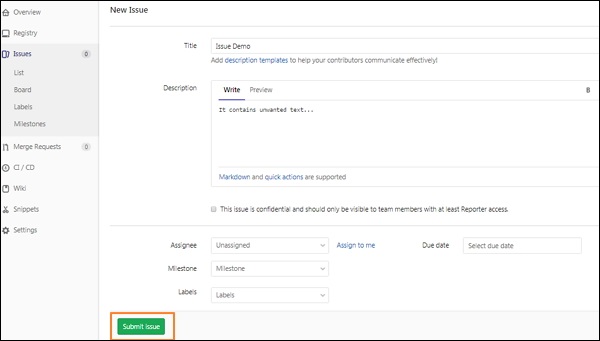
**Step 1** − Login to your GitLab account and go to your project under *Projects* section −



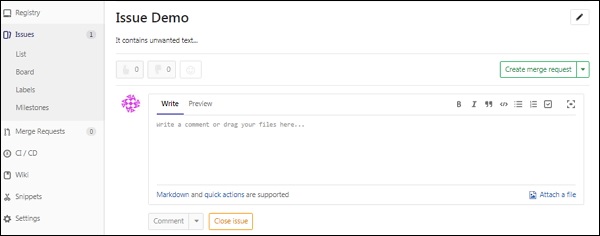
**Step 2** − Go to *Issues* tab and click on the *New issue* button to create a new issue as shown below −



**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](https://www.tutorialspoint.com/gitlab/gitlab_milestones.htm) for more information), labels upon operation or could be choose by developers themselves later.



**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 −



# **GitLab - Merge Requests**

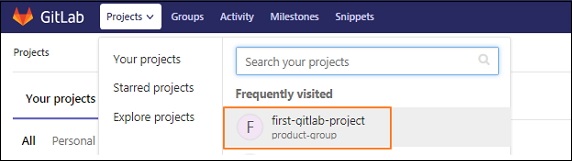
## **Description**

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.

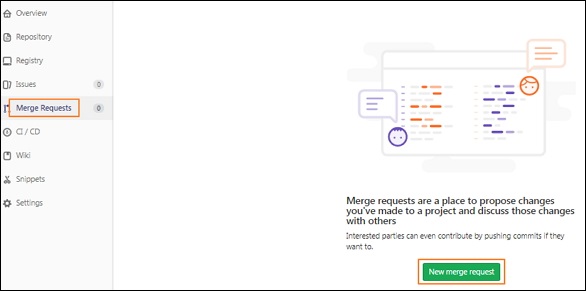
## **Steps for Merging Request**

**Step 1** − Before creating new merging request, there should be a created branch in the GitLab. You can refer this [chapter](https://www.tutorialspoint.com/gitlab/gitlab_create_branch.htm) for creating the branch −

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



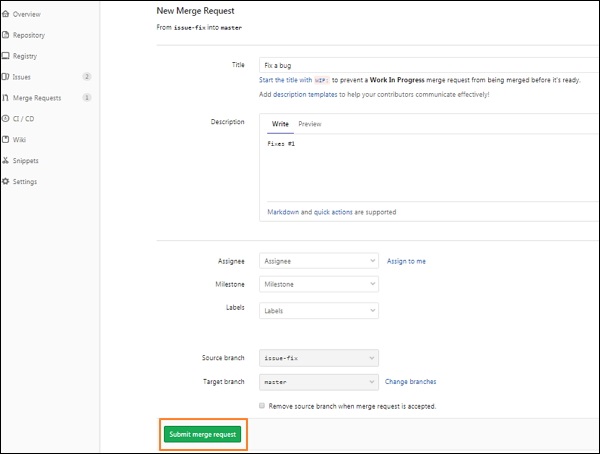
**Step 3** − Click on the *Merge Requests* tab and then click on the *New merge request* button −



**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 −



**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 −



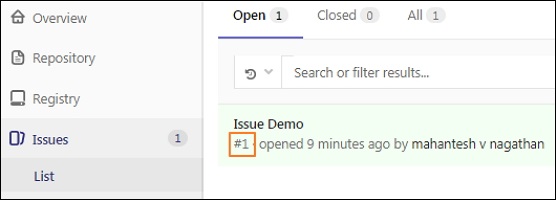
**Step 6** − After submitting the merge request, you will get a new merge request screen as shown below −

# GitLab Merge Request **GitLab - Referencing Issues**

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, refer the [creating issue](https://www.tutorialspoint.com/gitlab/gitlab_create_issue.htm) chapter.

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



**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

GitLab Reference Issue

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

GitLab Reference Issue

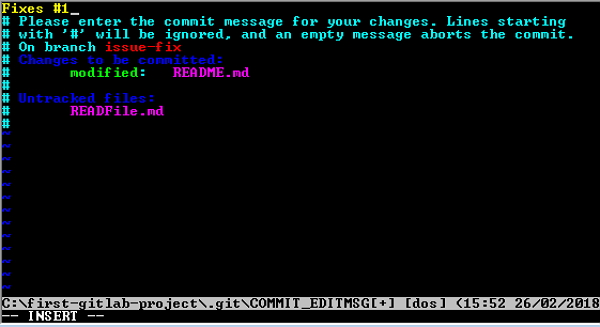
**Step 5** − Now, add some content to the *README.md* file to fix the bug −

echo "fix this bug" >> README.md

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

git commit -a

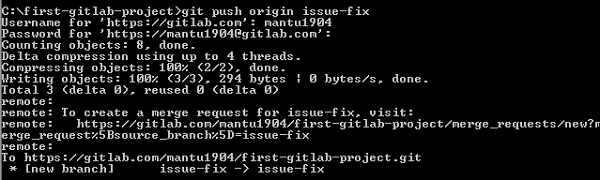
This command opens the below screen and press *Insert* key on the keyboard to add a commit message for the *issue-fix* branch.



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



**Step 8** − Login to your GitLab account and create a new merge request. You can refer the [merge request](https://www.tutorialspoint.com/gitlab/gitlab_merge_requests.htm) 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](https://www.tutorialspoint.com/gitlab/gitlab_merge_requests.htm) chapter), you will see the Closed option after closing merge request.

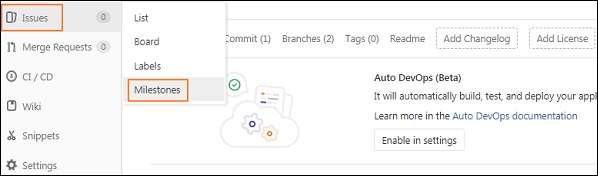
# GitLab Reference Issue **GitLab - Milestones**

## **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**

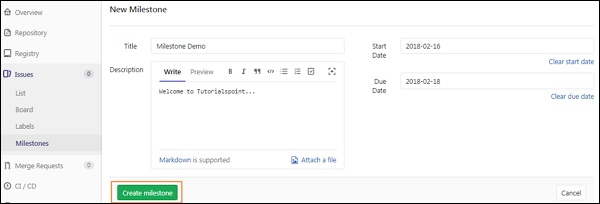
**Step 1** − Login to your GitLab account, go to your project and click on the *Milestones* option under *Issues* tab −



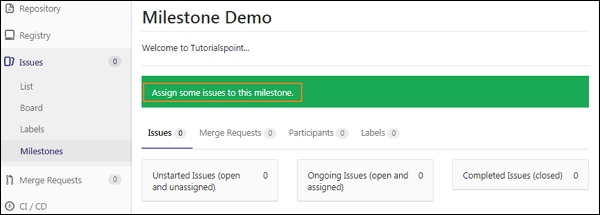
**Step 2** − Click on the *New milestone* button −



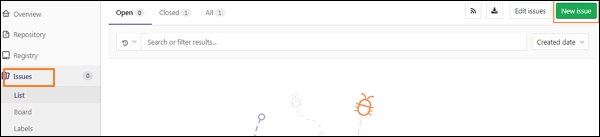
**Step 3** − Now enter the title, description, start and due date and click on *Create milestone* button as shown in the below image −



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



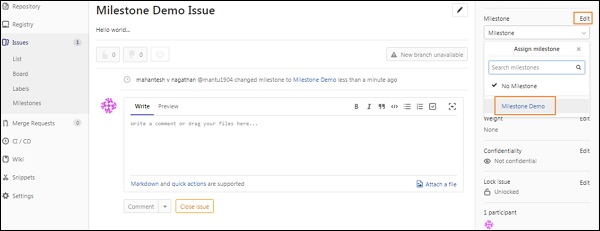
**Step 5** − Now go to *Issues* tab and click on the *New issue* button to create an issue for the milestone −



**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.



**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 −



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

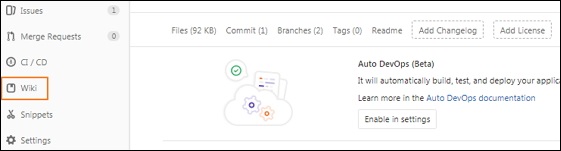
# GitLab Milestone **GitLab - Wiki Pages**

## **Description**

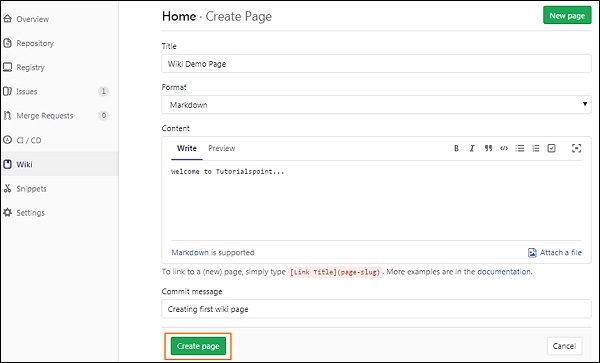
Wiki is a system for maintaining documentation for a project in the GitLab. It is like a Wikipedia which can be editable and given permissions to manage the wiki pages. A *Guest* can view a wiki page and Developer can create and edit a wiki page.

## **Steps for Creating Wiki Page**

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



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



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

