**GITLAB – 02.03.2019**

What is Gitlab?

1. Online Git repository Manager
2. Great way to manage Git repository

**What is Git?**

* Distributed source code versioning system- It lets you locally track changes and push or pull changes from remote resources
* Work Coordinate in teams
* Speed and data integrity

**Gitlab vs Github vs Bitbucket**

* These are services that provide access to Git repositories
* Provide additional features to manage code
* Sharing of code, bug tracking, wiki space and other tools

1. **Github**

* Public free service
* Get suggestions for improvement

1. **BitBucket**

* Web-based version control- repository hosting service which is owned by Atlassian
* Unlimited number of private repositories

1. **GitLab**

* Github like service – that organizations can use or internal management of git repositories
* Self-hosted Git –repository Management system – that keeps the user code private and easily deploy user management of the code

**Why use Gitlab?**

* Hosts project for free i.e. public and private
* Platform for managing git repositories
* Free and public repos issue tracking and wikis
* User friendly web interface layer
* CI (Continuous Integration) for managing projects
* Option to share only a part of project
* Unlimited number of private and public repositories

Limitations of Gitlab:

1. Not as fast as Github

Some Terms with GitLab - <https://about.gitlab.com/2015/05/18/simple-words-for-a-gitlab-newbie/>

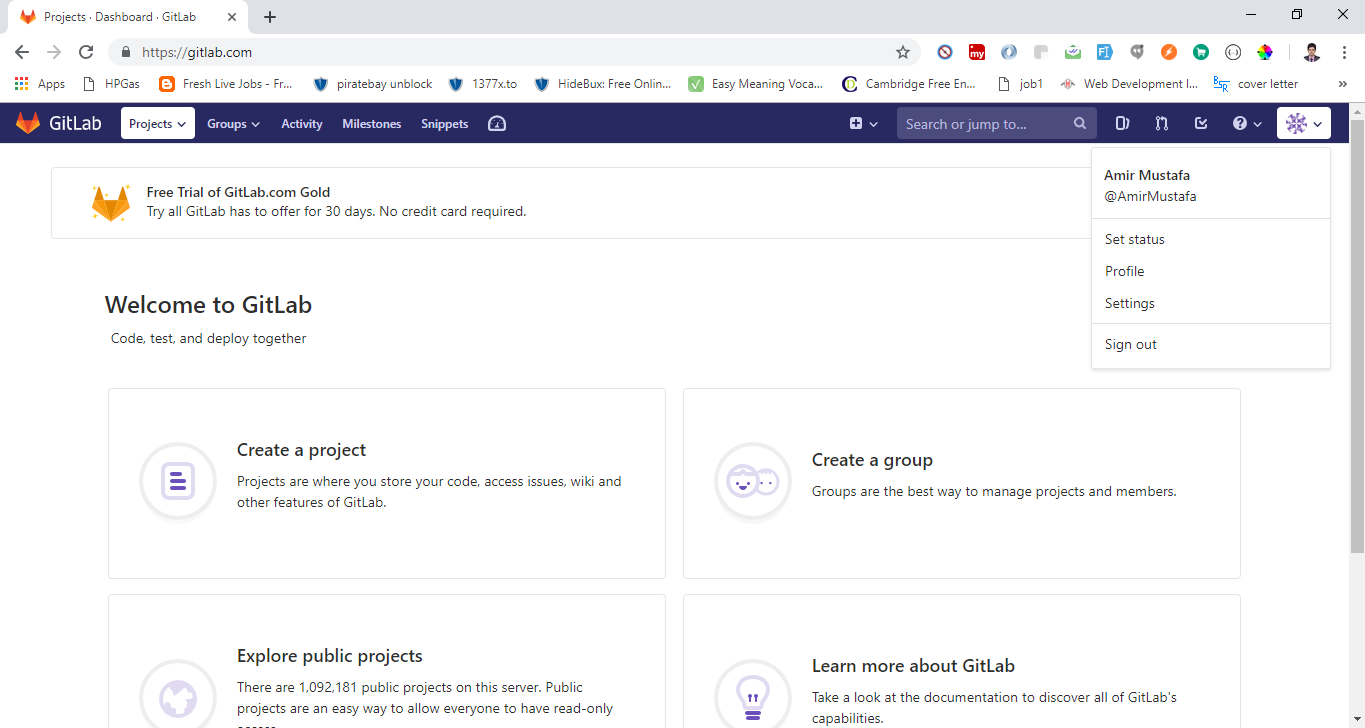
Installing in Windows

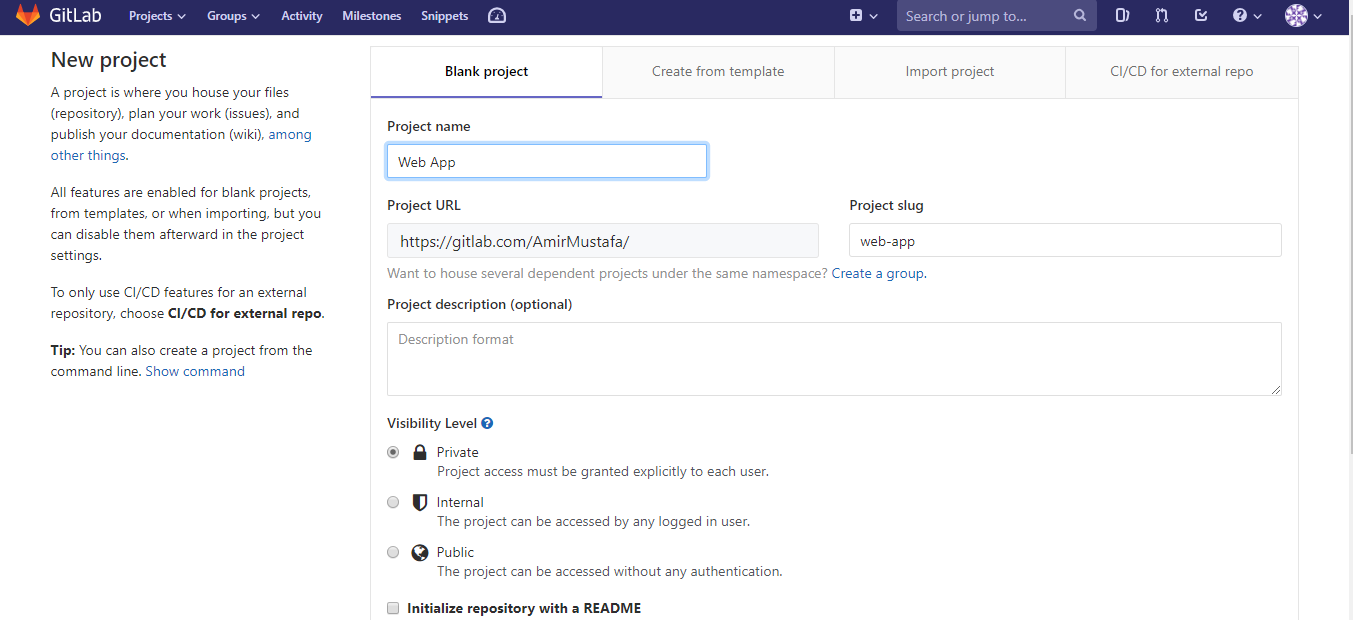
1. Create GitLab-Runner directory in your drive say C drive
2. Download file from this link:

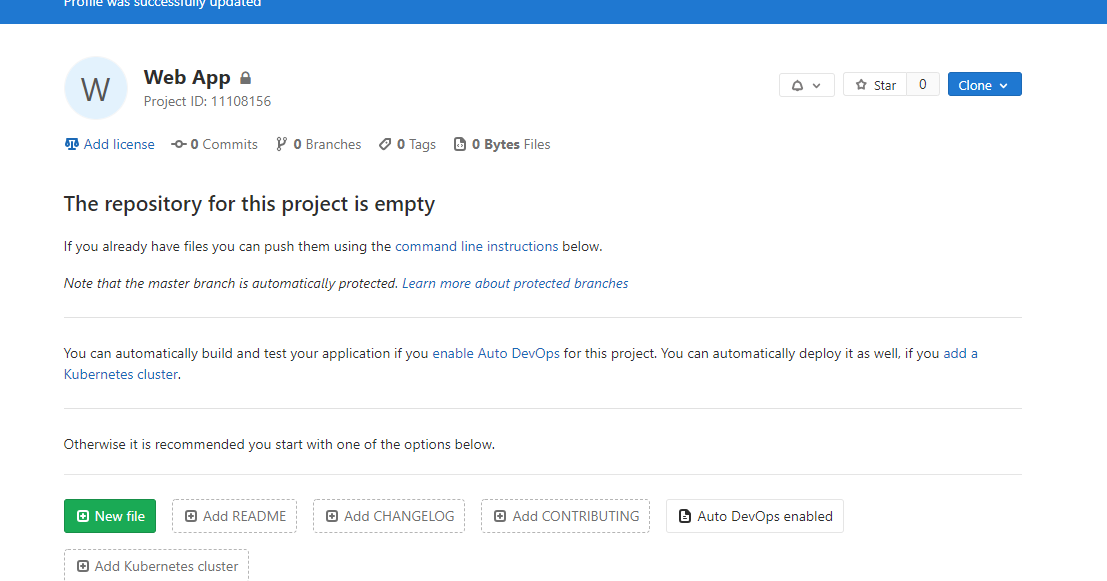
<https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe>

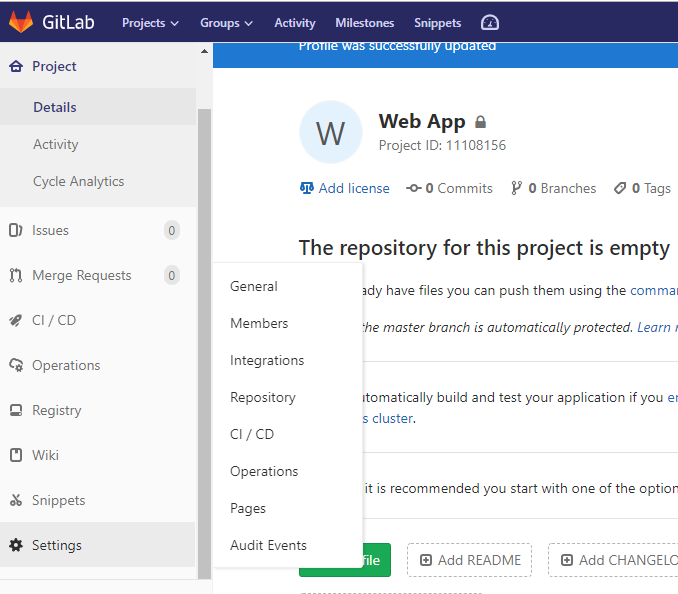
an exe file will be downloaded

1. Copy paste the download file to folder created in point 1, rename it to gitlab-runner. Open the directory in command prompt.Go to Gitlab.com
2. To generate a token first create a sample project. Click Create a project enter any project name like Web App (select private repo) **Create Button**

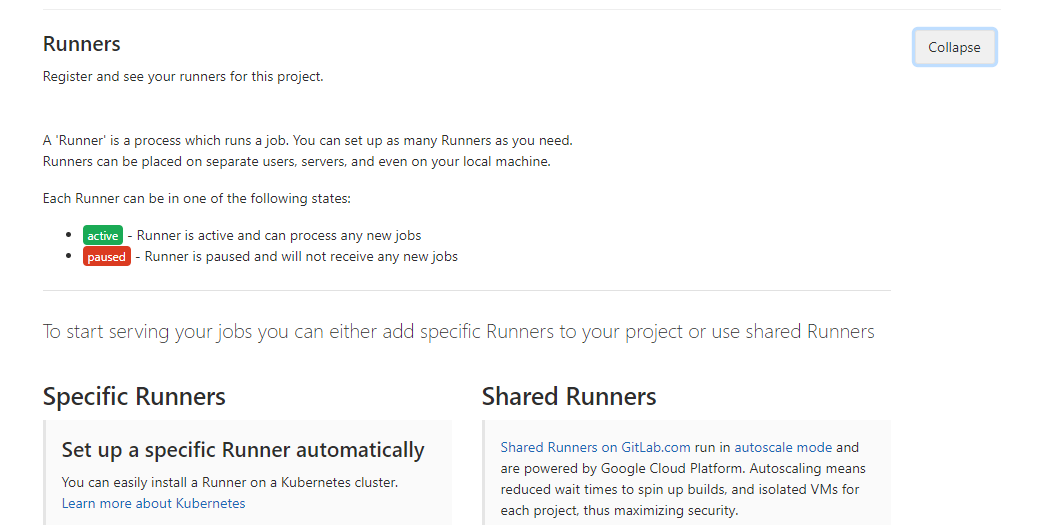








* Now goto Settings -> CI / CD and click expand Runner button:

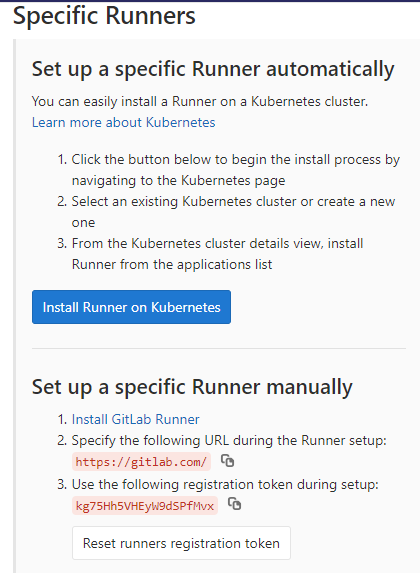


* Here you will see two types of runner: Specific runner and shared runner

What is a Runner?

1. A runner is a isolated virtual machine that pick up your jobs through the coordination API of Gitlab CI.
2. A runner can be specific to a certain project or serve any project in Gitlab CI
3. A runner that servers all project is called a **shared runner**
4. On the other hand **specific runners** are useful for jobs with special requirements or demands
5. Other than this there is another runner called **Group runners** – when you have multiple project under one group and would like all projects to have access to a set of runners. It uses FIFO (First in First out queue)

You see steps to use runner. Copy this **token** we will use in command prompt



* Now open the git lab runner folder in command prompt and where that exe file is kept and run this command:

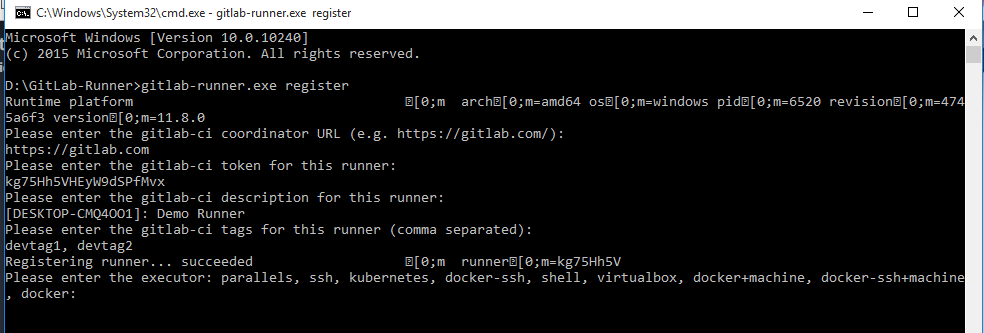
gitlab-runner.exe register to execute

It will ask basic info like:

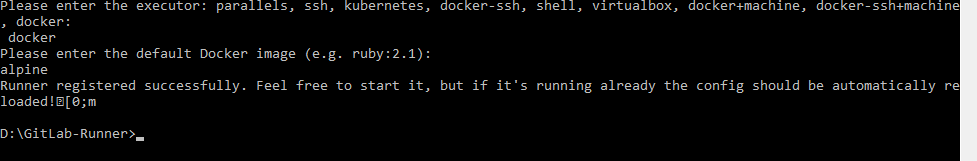
Gitlab coordinator – <https://gitlab.com>

Token – <paste\_you\_copied\_token>

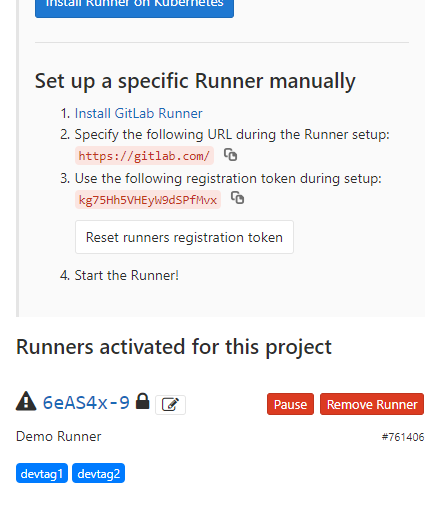
Some desc, some tags see the screenshot below. You will get registered successful message



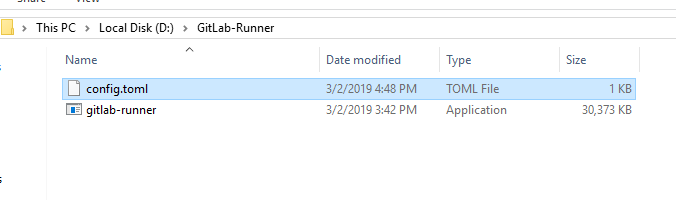
* We will chose docker and enter alpine

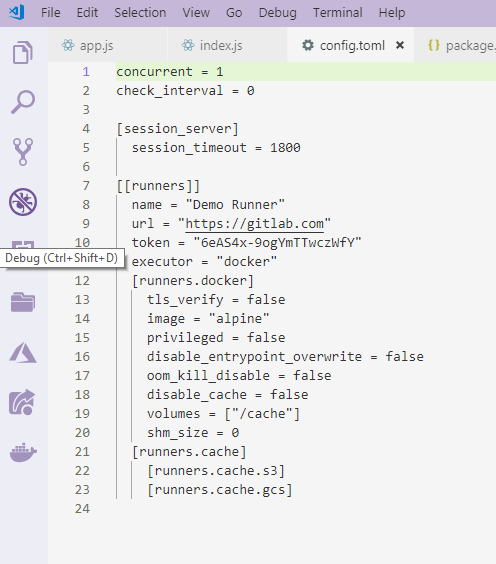


Now when you go to same setting -> CI/CD and expand runner you will see the runner we have activated:



You will see new file created config.toml and our runner is registered there:





Important Git Commands

1. To check version:

git –version

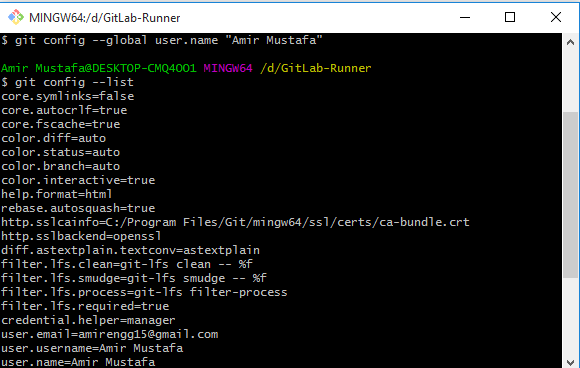
1. To give username

git config –global user.name “Amir Mustafa”

1. To check:

git config –global user.name // Amir Mustafa

git config –list // returns full object of lists assigned



1. Set email:

Git config –global user.email [amirengg15@gmail.com](mailto:amirengg15@gmail.com)

1. Create branch:

git checkout –b test1 // created new branch and switched to it

git checkout –b master // switched back to master branch

1. See status of staged/unstaged

git status

1. To push from branch to master:

git push origin master

Link - <https://orga.cat/posts/most-useful-git-commands>

SSH Setup:

* SSH – a cryptographic network protocol
* Interact with the remote repositories. In Gitlab we can either use HTTPS or SSH. In windows earlier we used putty to connect with SSH. Now windows by default have SSH client that you just need to enable it.

For Installing SSH

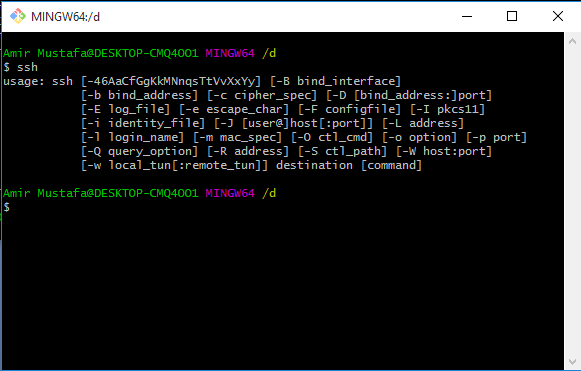
**METHOD 1:**

In windows type developer settings 🡪 select developer settings and install it

Now search in windows Search 🡪 Apps and features 🡪 Manage optional setting 🡪 click Open SSH Beta 🡪 install and restart the windows type **ssh (successfully installed)**

**METHOD 2: Open Git bash: It by default has SSH in it (easy)**

Type : ssh



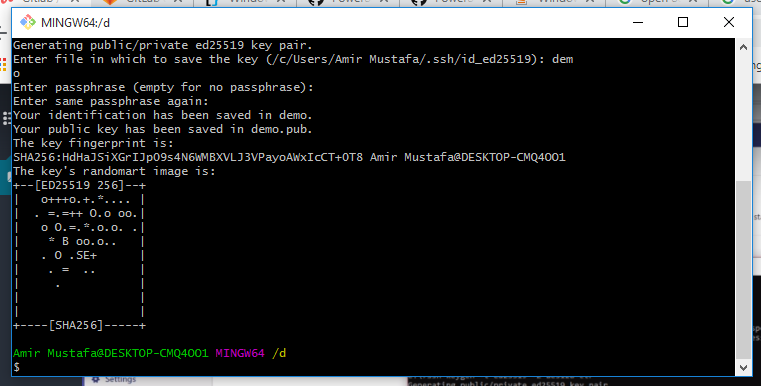
**To generate a key pair: ssh-keygen –t ed25519 –Z aes128-ctr (encryption keys public and private resp.)**

Check more here - <https://docs.gitlab.com/ee/ssh/>

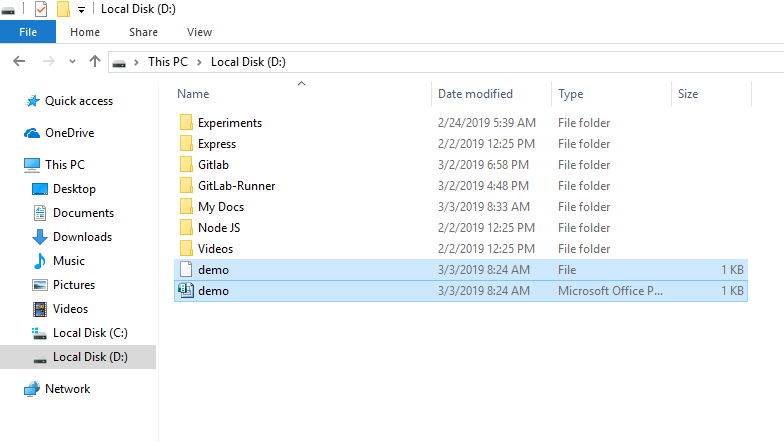
Will ask for name and passphrase as shown in the screenshot below and the key will be saved in C drive.

The public key is generated and fingerprint is shown here as well

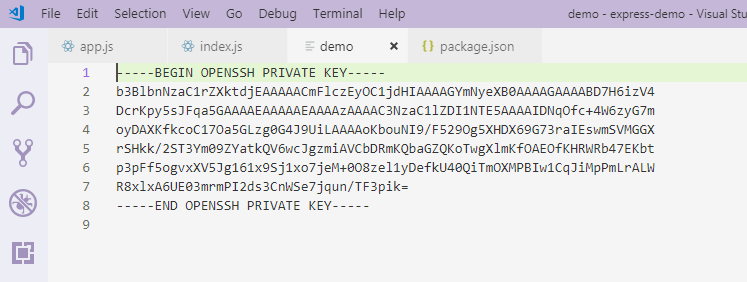
Fingerprint – It is the short sequence to a longer sequence to the longer key file.

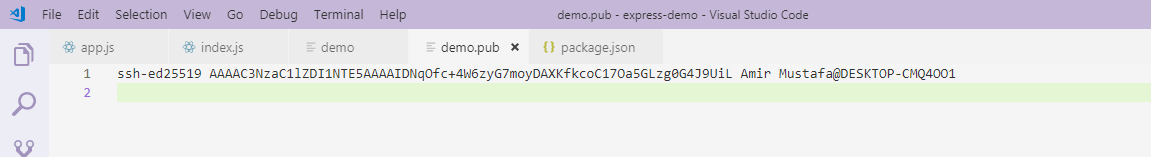


Go to the drive where command is written(I have written in D drive). There two files will be generate one for **public** and the other for **private key**



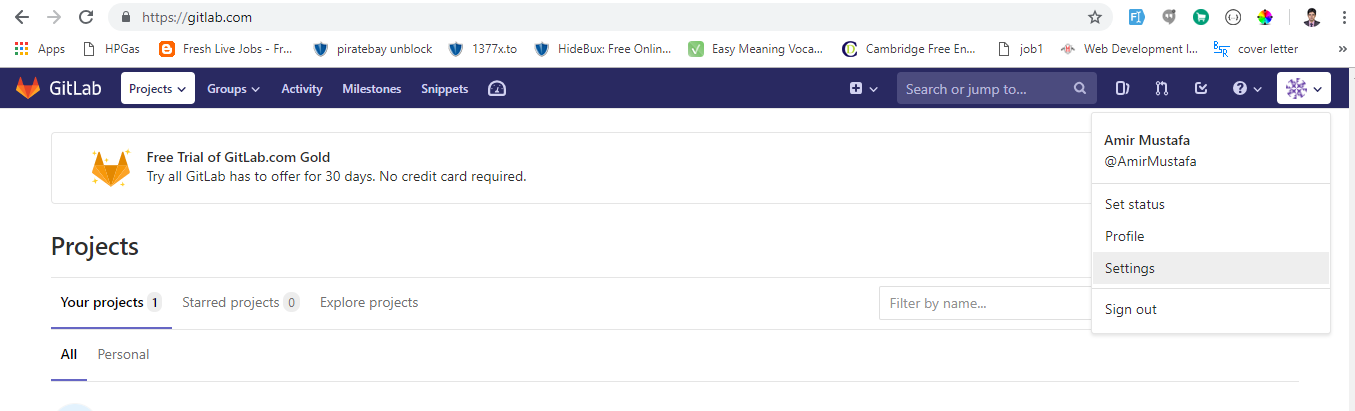
Let’s see the file:



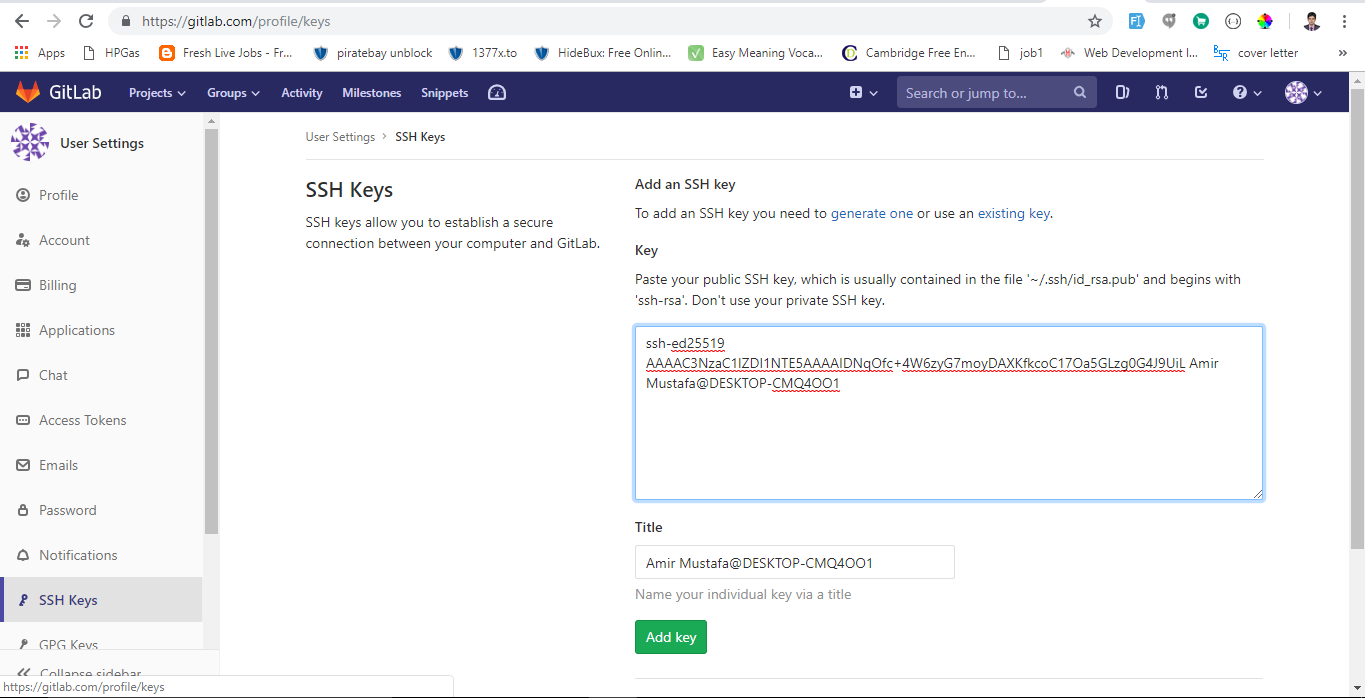


So second one is the public key generated.Copy this public key(second screenshot)

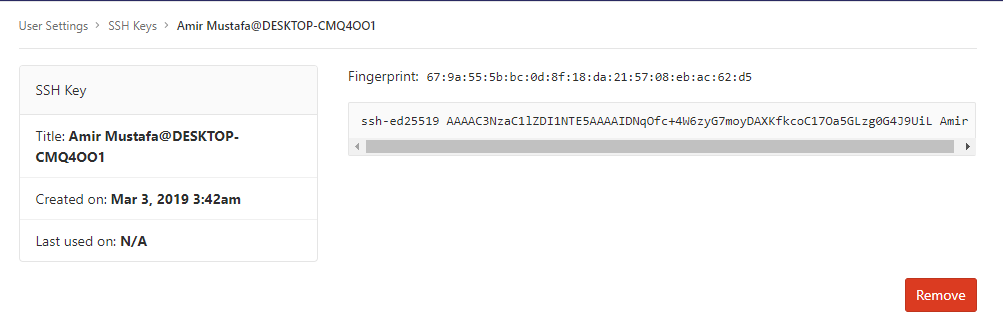
Now go to your gitlab account and click settings from user profile menu:



On the left explorer select SSH Key and paste the copied public keys there and enter Add Key (as shown in the screenshots below)



So select now key is added and you can also see fingerprint. Fingerprints are shown in hexadecimal string for better readability.



Forking a Project

1. Forking vs Branching:

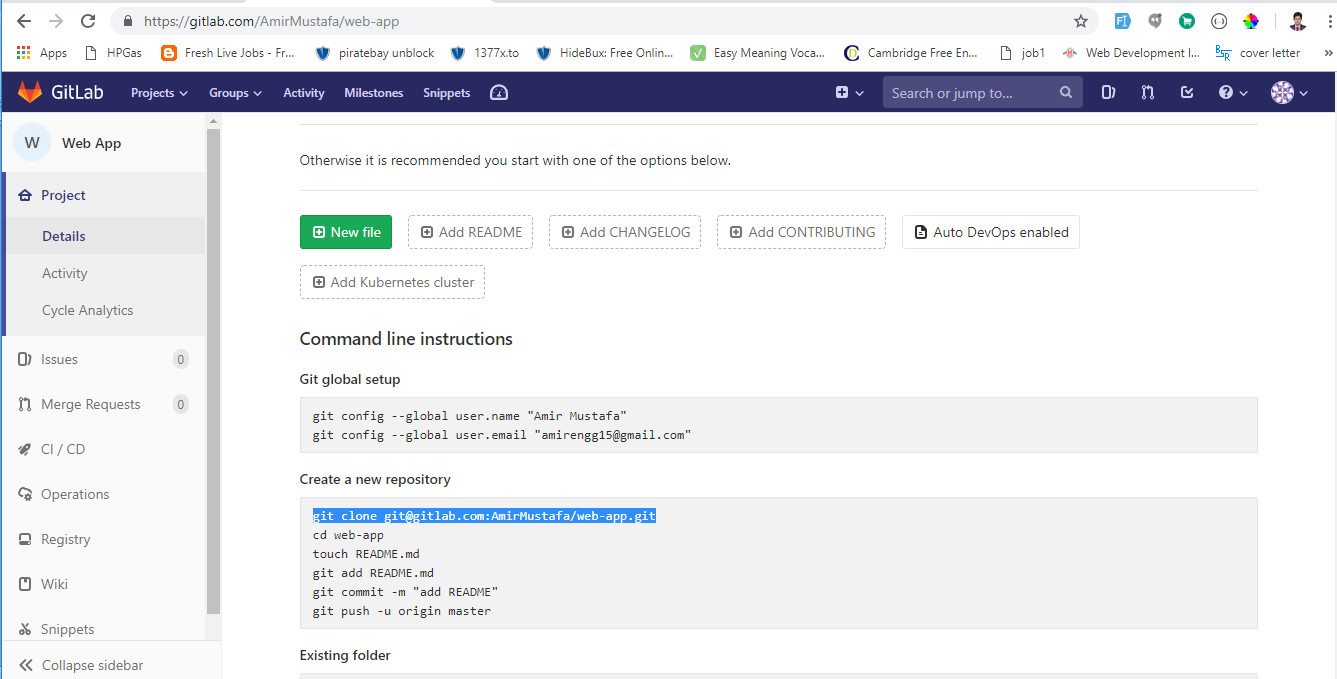
There is a bit of confusion between forking and branching.

Branch- Branch is like a branch of a tree. We have taken code from it but somewhere it is still a part of the main code.

Forking – Its just a clone/copy of the main project

If original repository is deleted branch is also deleted but fork is not.

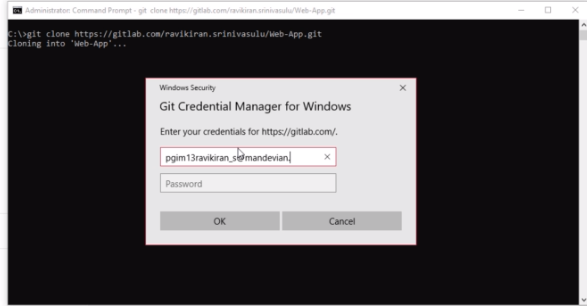
Select https in your gitlab in your project



Enter this in your SSH where you want to clone

git clone <your\_HTTPS\_url\_to\_clone>

It will ask for your gitlab credentials while creating your gitlab a/c



In Repo:

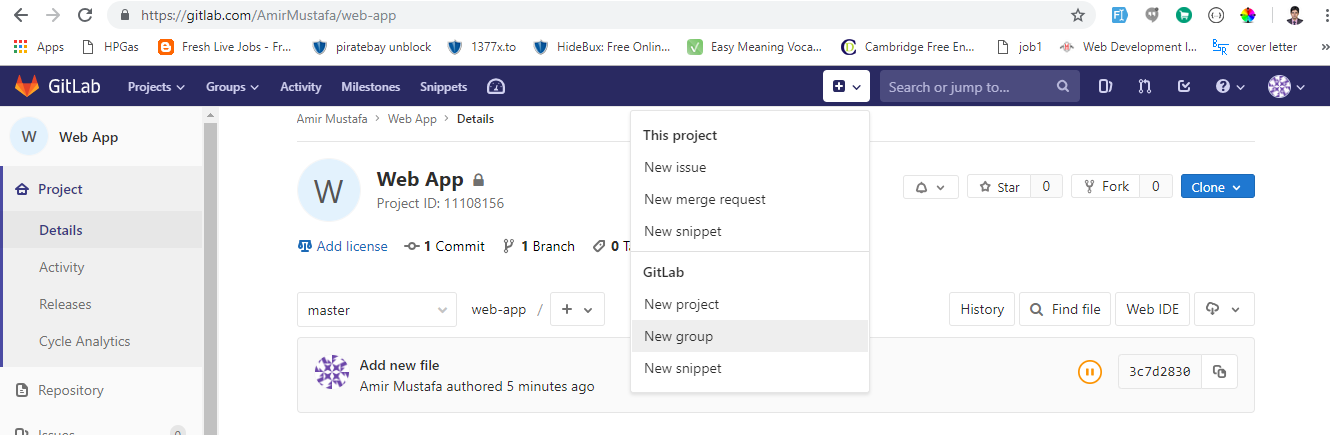
touch README.md

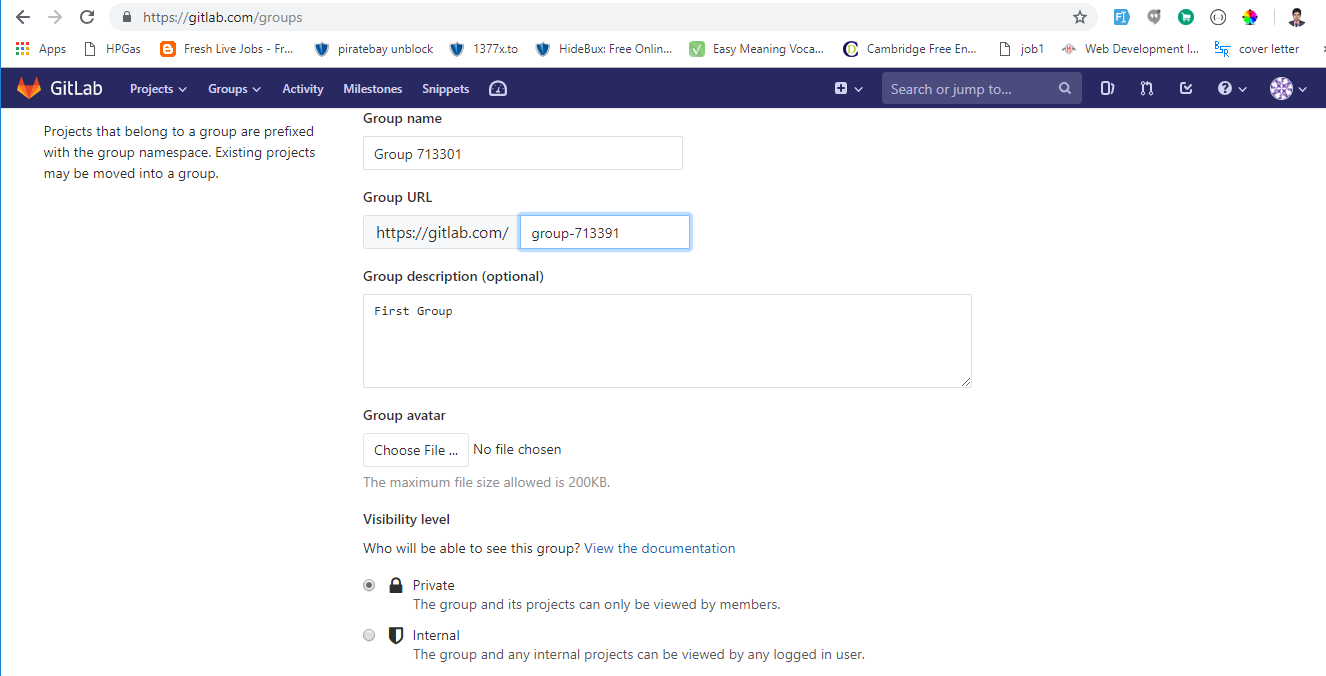
git add README.md

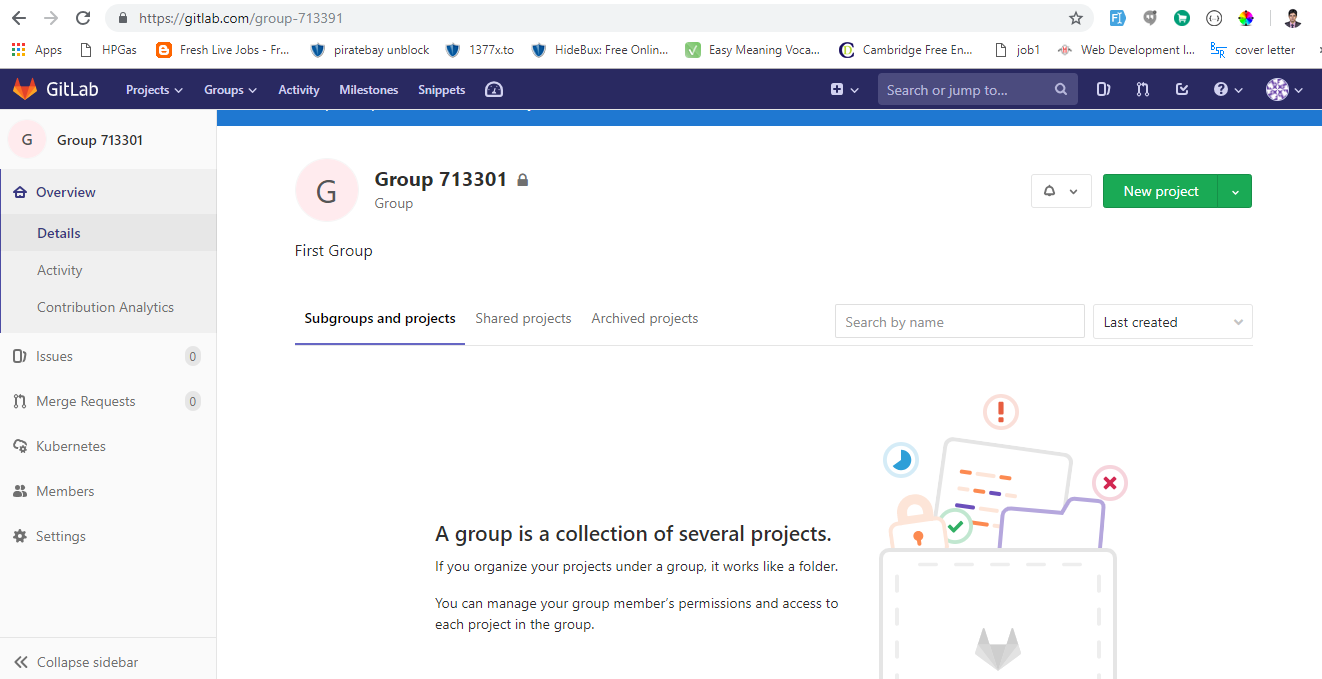
git commit –m “Adding 1 file”

git push –u origin master // origin = url we copies, master = branch name (main)

* For forking the project we need to first **create the group**. With Gitlab we can create groups where we can assemble related projects together
* Grand members can access several projects at once

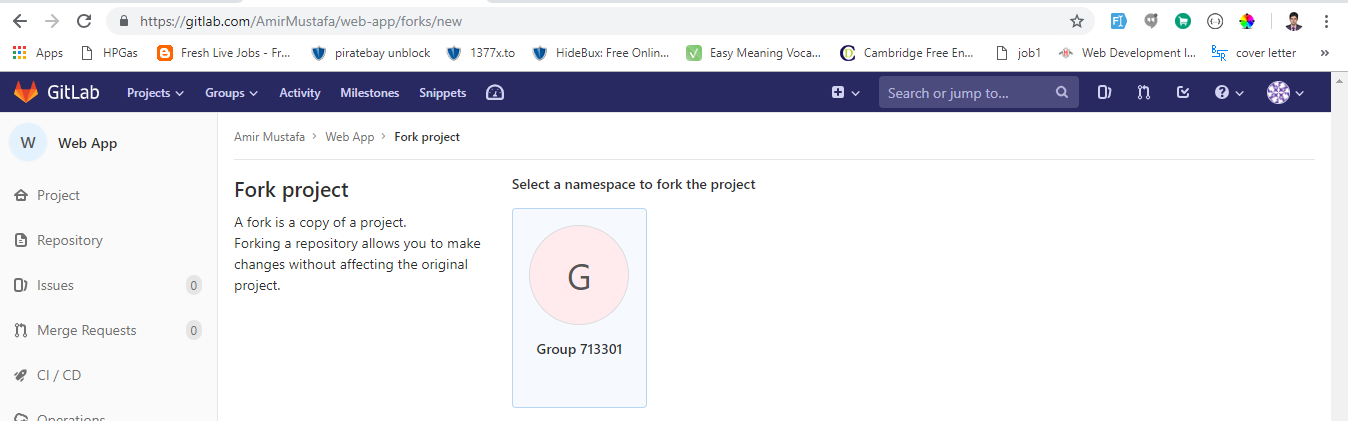


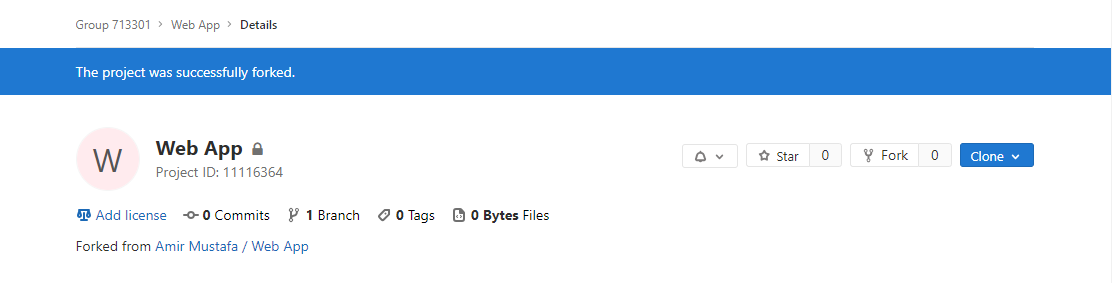




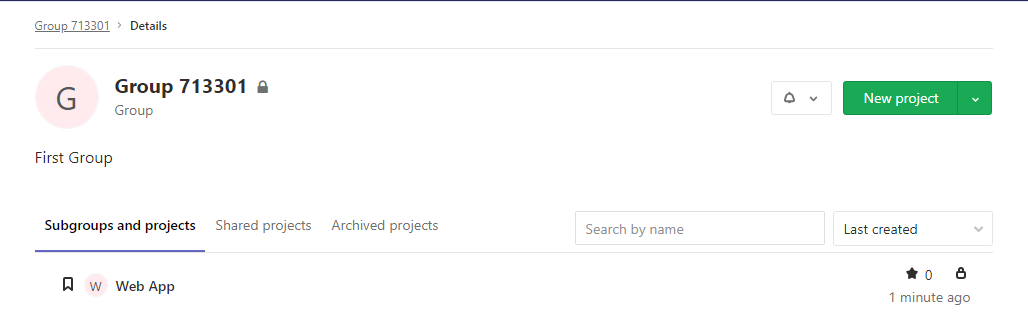
* Now go to your project and click on the fork button and select the group recently created:







* So you will find this project in new group created



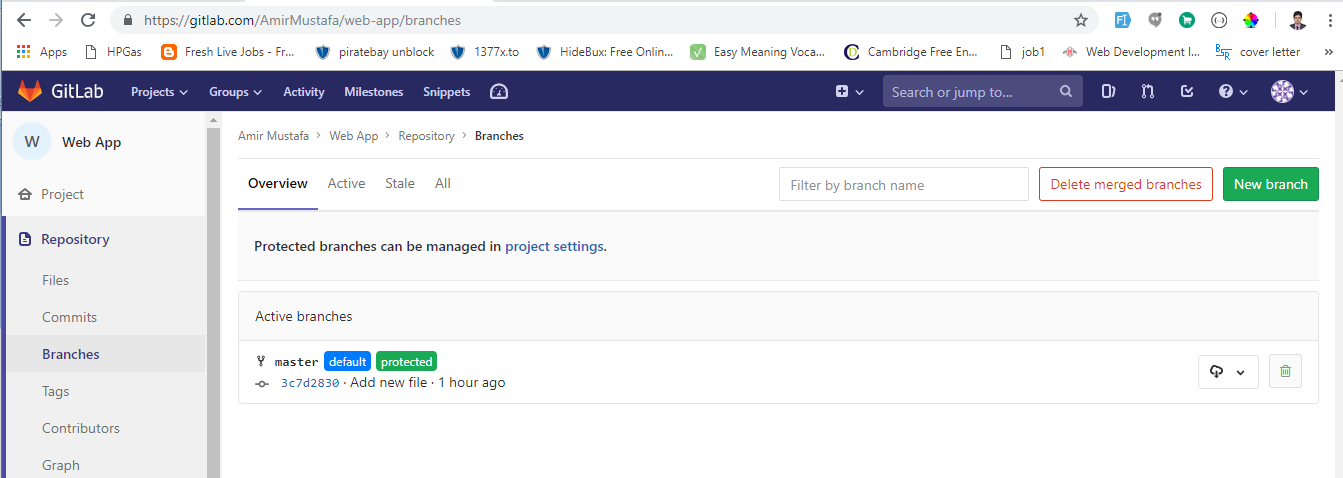
Branching

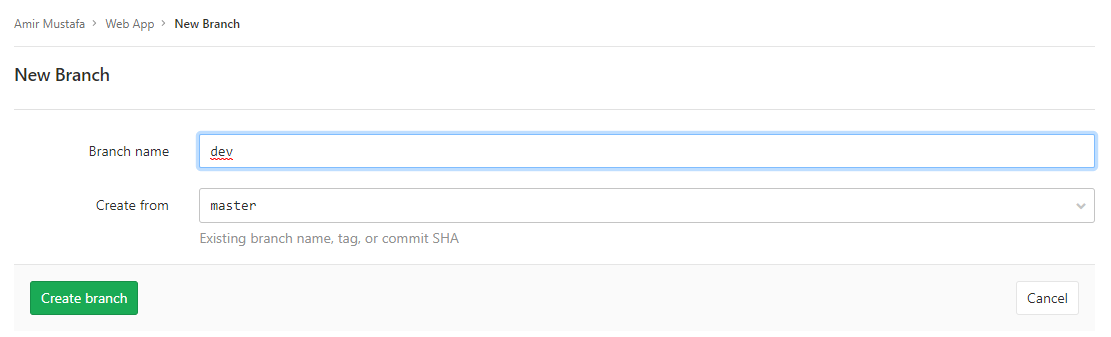
* Branching is the duplication of the source code file so that modification can happen in parallel along both the branches.
* Branching also generally implies the ability at a later stage to merge or integrate the changes back to the parent branch(i.e. master branch).
* The branch that is not intended to merge is called fork(that we have seen already)
* A branch in git is simply a light weight movable pointer to the commits. The **default branch** is **master** andas youstart making commits it points to the last commit, the branch moves forward automatically

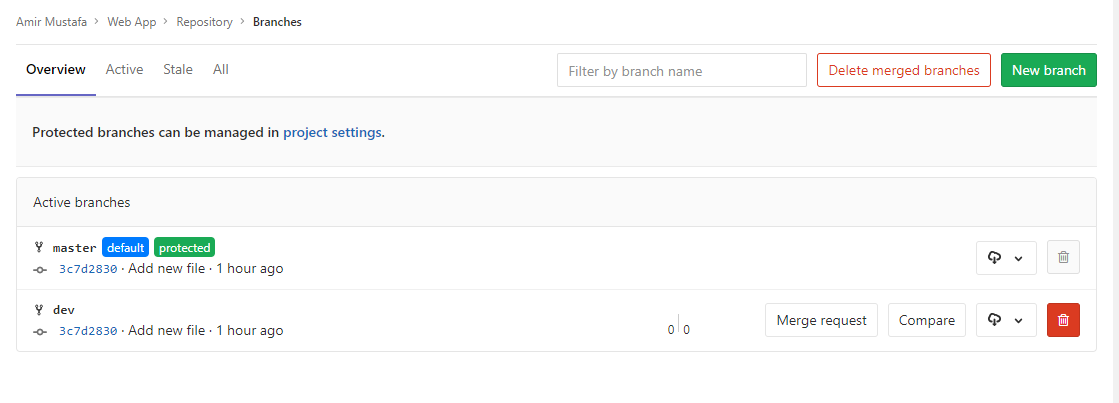
Way to create new branch:

Go to the project 🡪 click on the branches.

Default there will be only 1 branch (i.e. master). Click on New branch to create one.





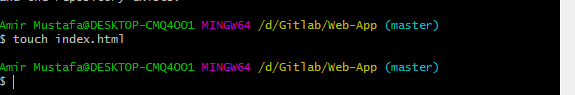


So if you go to the master or the dev branch you get the files in both repositories.

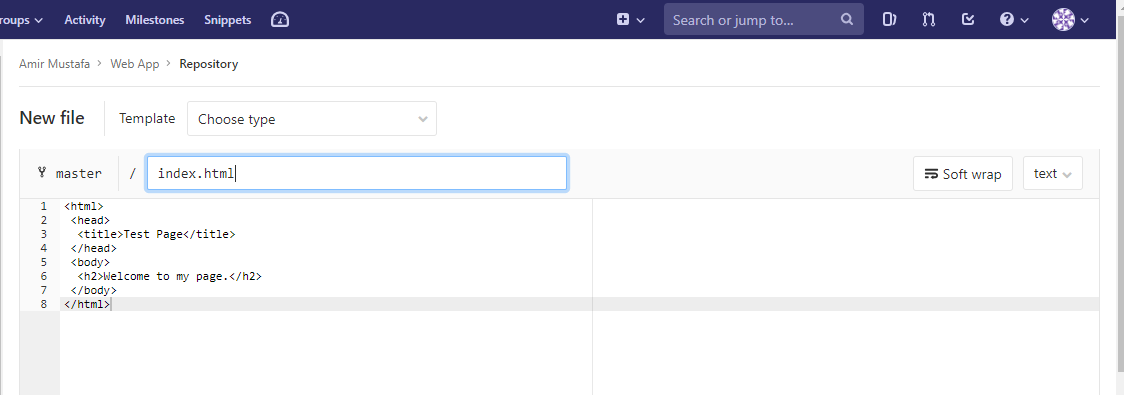
**Adding Files to the repositories:**

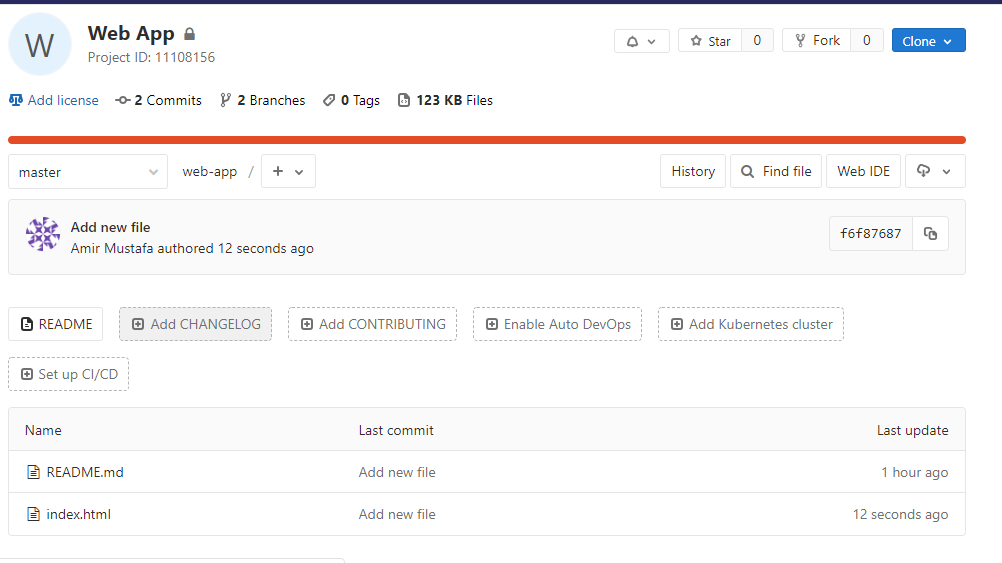
* There are two ways you can add file in repositories:

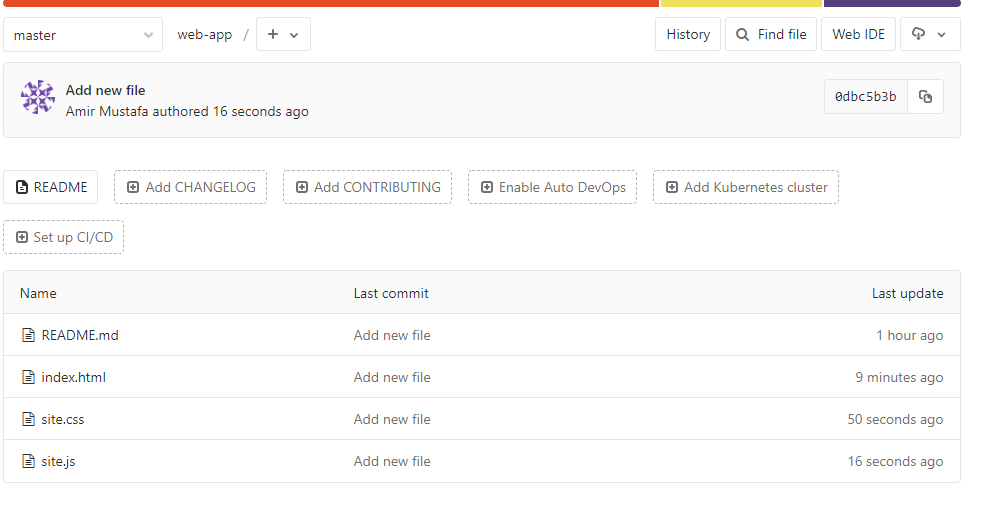
1. Command Line Interface - touch index.html



1. Web Interface - Go to the project click + sign new file







Rebasing

* Here we will see how to rebase the changes between different branches. There are basically two ways in which you can merge the changes between two branches:

1. Merge – common way of seeing changes
2. Rebase – alternate way of merging and slightly more advanced means of integration

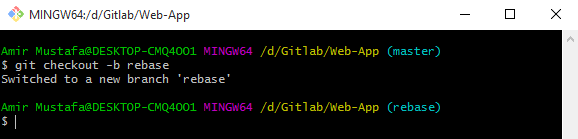
Problems with Merge:

* So when we commit changes that led this point is not preserved in a commit. Code will remain but the meaningful comments get removed. This is why people dislike merge

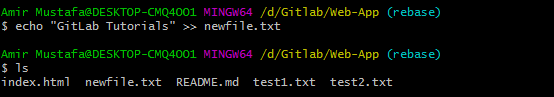
Rebase:

* So how rebase works is instead of cramming all the different changes into a merge commit, rebase preserves the original commits. So project history looks as safe and word in the single safe line.
* So let’s create a new branch through CLI

git checkout –b rebase



Lets create a new file

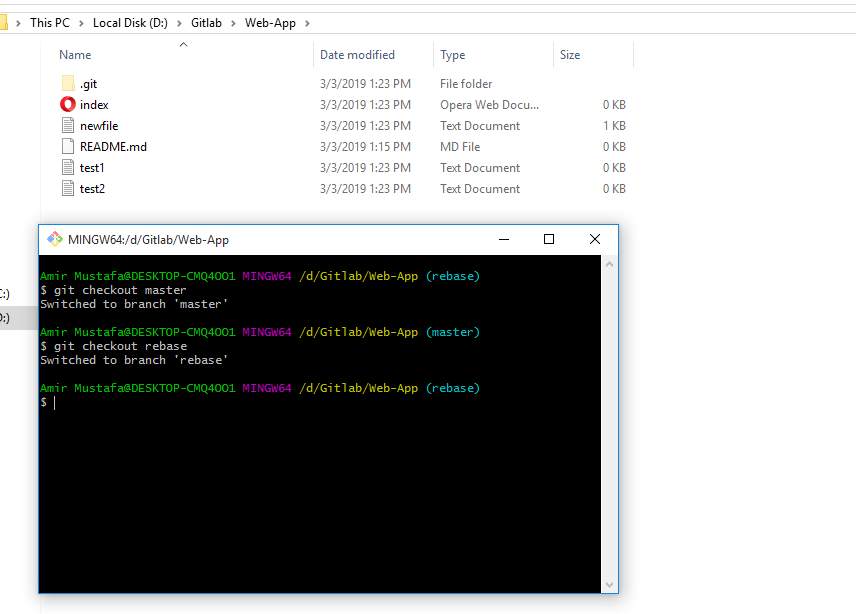


* Firstly understand one thing if you open same directory in parallel with command prompt, in the same directory files will change as per branch selected:

1. Master:



1. Rebase (another branch)



* Now understand this concept. Master and rebase branch both has empty Readme.md

We will create a new file in rebase branch with some text:

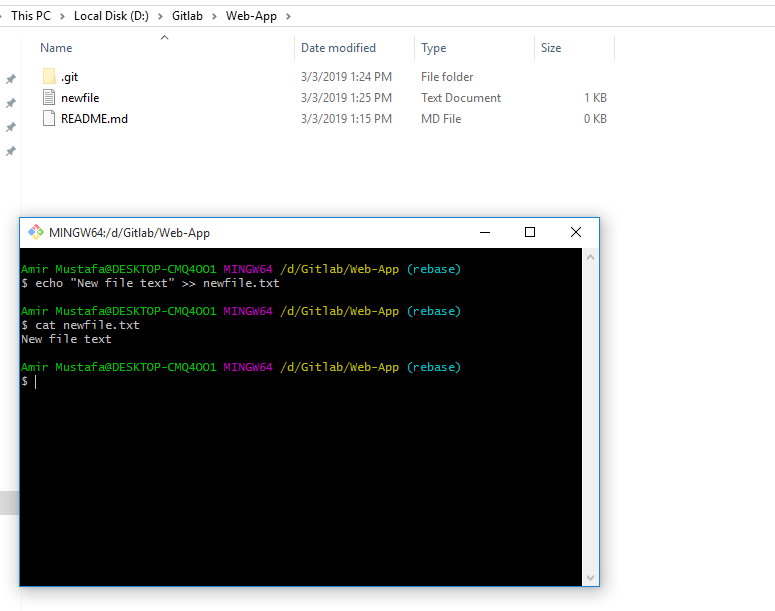
echo “new file text” >> newfile.txt

we can view text either by opening file in editor or cat command i.e.

cat newfile.txt

git add .

git commit –m “Added new file” // commit the changes



README.md file of this rebase branch is empty.

Now lets switch to the master branch: git checkout master

We will write in README.md file of the master branch:

echo “text of readme file” >> README.md // written in readme of master branch

cat README.md // you can view the text written

Now go to the branch where you want you want changes (in my case rebase)

Git checkout rebase // switched to rebase – (Branch in which will copy to)

git rebase <branch to copy from changes> // execute this in the branch you want to apply

git rebase <branch to copy from changes> // branch from where we will copy from

git rebase master // means copy all changes from master to rebase(current branch)

Now the in rebase branch the README.md will have text from master + its own newfile.txt

Squashing Commits:

* Here we will learn about how to squash commits

Squashing:

* Squashing is a way to combine all commits to one when you are obtaining a merge request.
* Squashing a commit means to move the changes introduced in a commit into its parent so that you end with one commit out of two.
* If you repeat this process multiple times you can reduce the **n commits** into a **single one**
* In git squashing is achieved with rebase with special form called **interactive rebase**

To demonstrate squashing let me create a branch and create **2 commits** with **different messages** and we will squash those commits

Lets create a new branch: git checkout –b testbranch

Lets create a file:

echo “message 1” >> test.txt

git add .

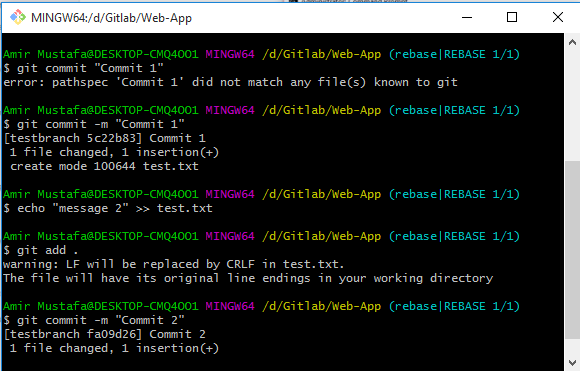
git commit –m “Msg1”

Lets repeat the step in the same file(i.e. second commit)

echo “message 2” >> test.txt

git add .

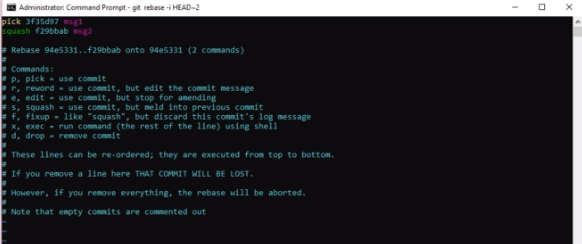
git commit –m “Msg2”



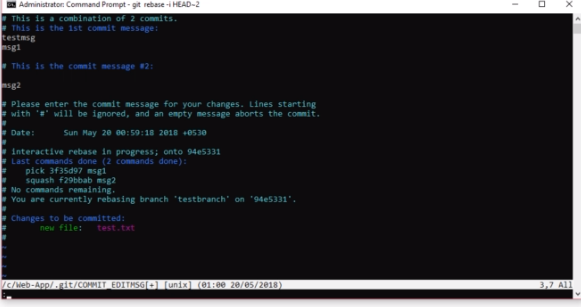
git rebase –i HEAD~2 and press enter // two squash last **n commit** HEAD~n (min 2 commit req to squash)



Now in second commit replace pick with squash (vim editor) once done press Esc 🡪 :wq (to save changes)

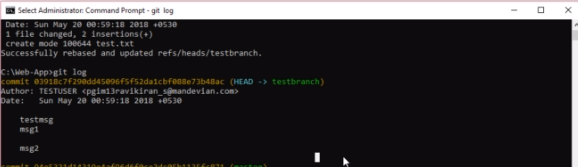


Write there your commit and save it like successfully squashed or any other message(here test msg 3rd line)



* It says successfully rebased and updated
* Type git log

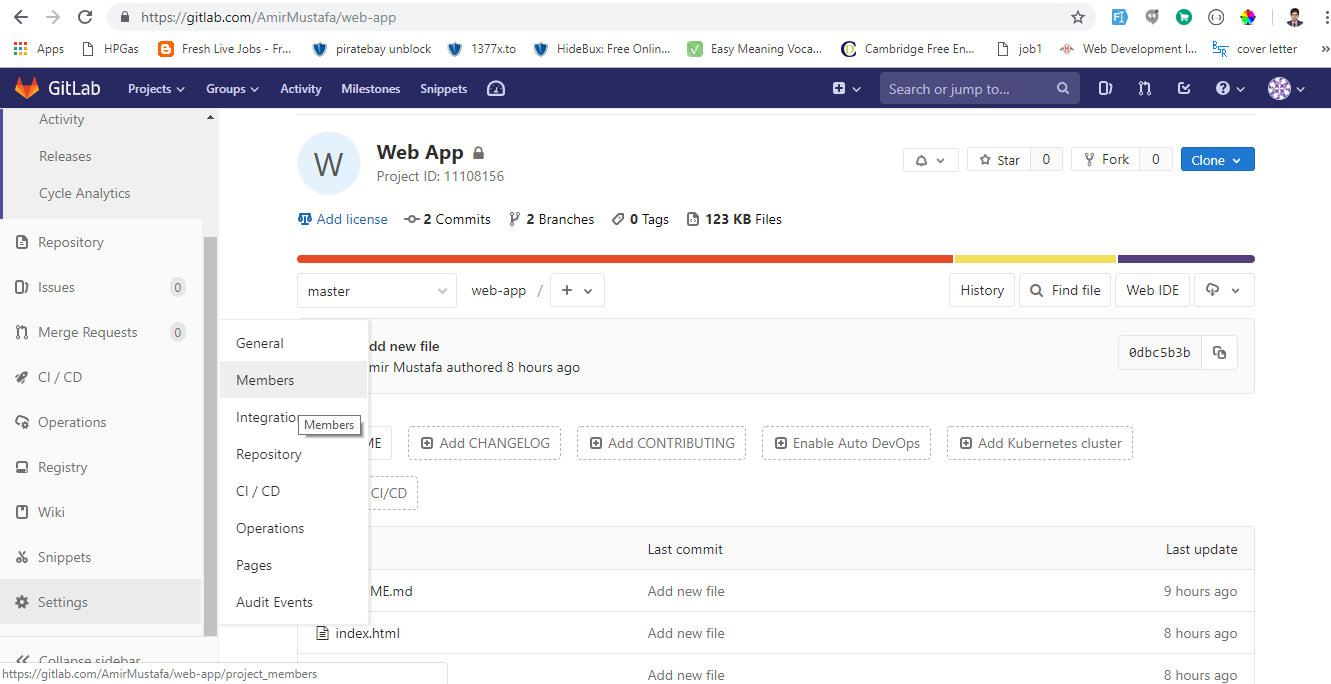
You will see only one commit to the msg1 and msg2 with additional message (test msg) written. So all in one commit

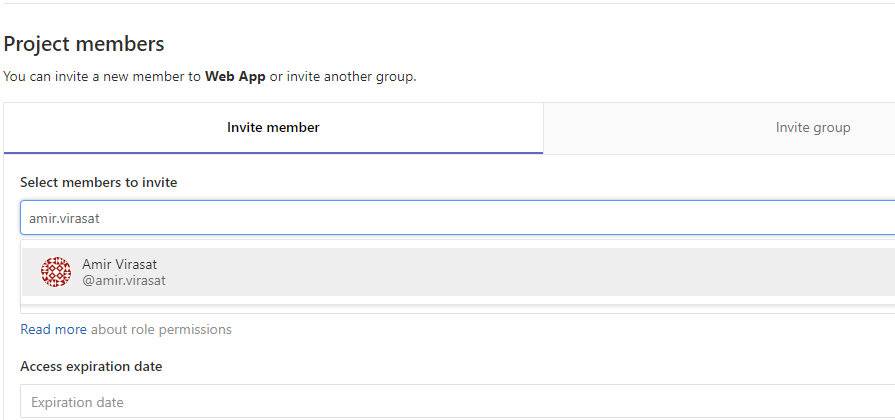


Adding Users

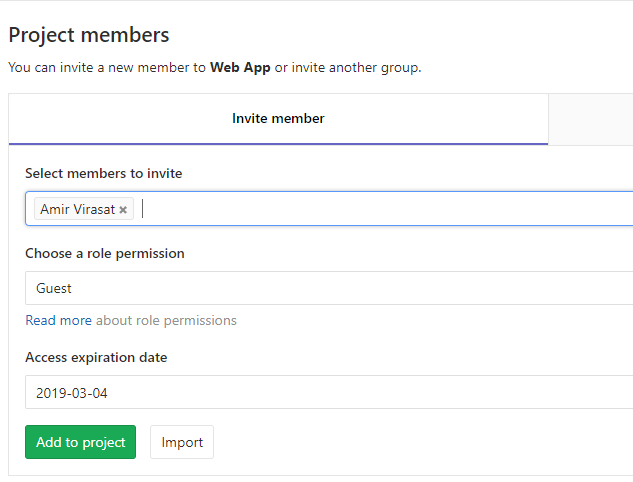
* Here we will see how to add users to the project. You can add users in Gitlab in one of the following ways:

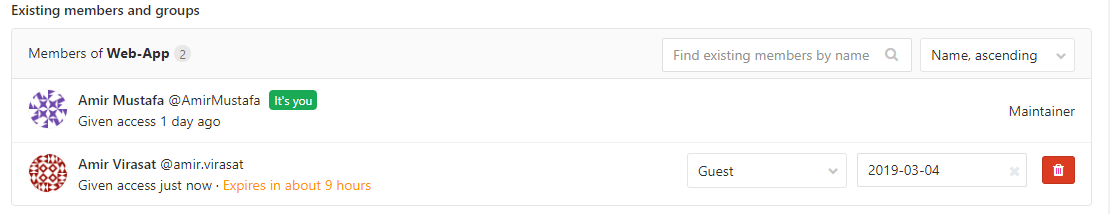
1. Add an existing users
2. Import users from existing project
3. Invite people by their email address
4. **Within your gitlab go to your project first**





You can add role, expiration date(if user is for some time in this project)

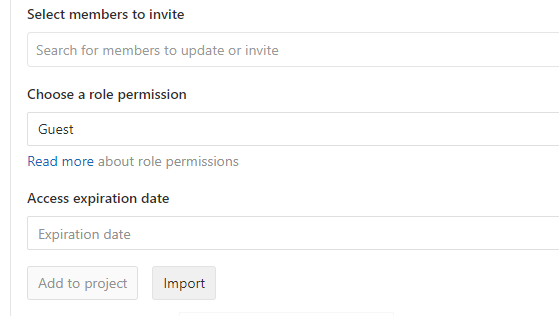


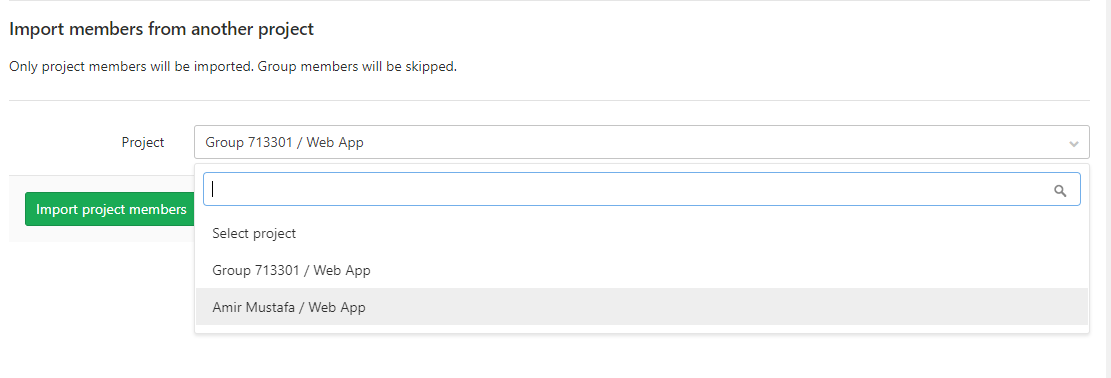


At any point of time you can delete or change the permission role of the user

1. **Import users from existing project**

If you want to import users from another project click on import button



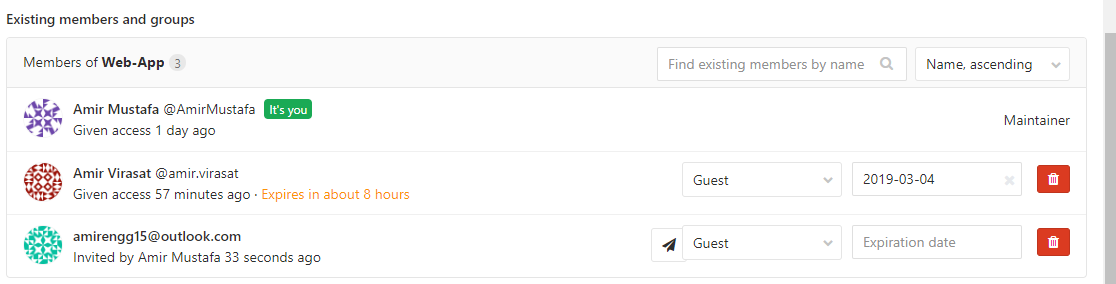


Select the project whose users you want to import. You will only change projects which have maintainer permission level.

1. **Invite people by their email address**

You can invite people by email simply by typing email address in the search user:

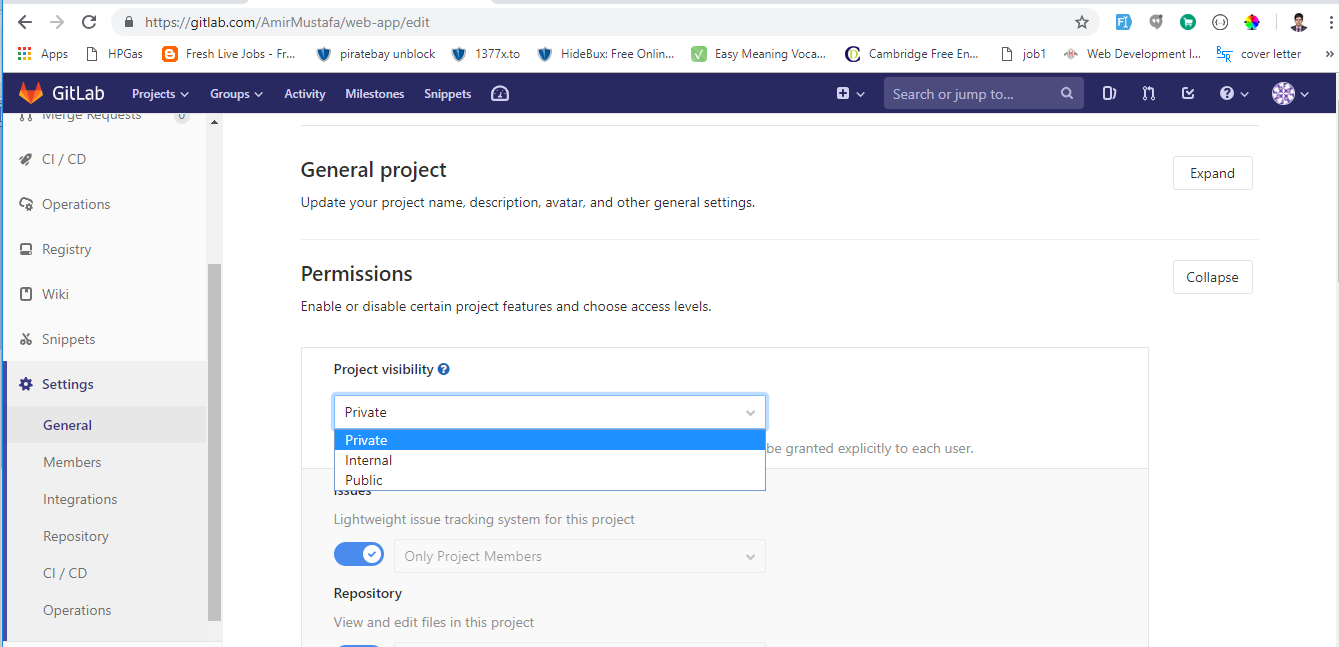


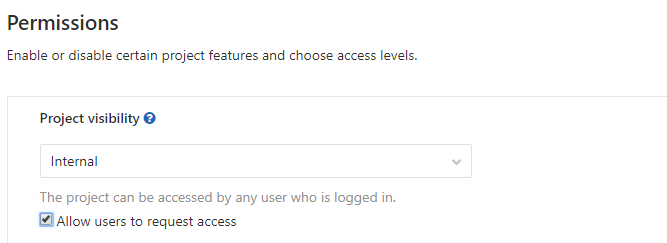


Once this new user confirms the mail, he/she will be prompted to create a/c with same email

* As a project owner you can enable or disable non members anytime to request access for your project (Project visiblility must be internal/public).

Go to General Settings





Adding Groups:

* Here we will see groups in GitLab. With gitlab you can create groups that can assemble related projects together and grant members access to several projects at once
* Groups can also be nested in sub groups

Use Cases of Groups:

* Organize related projects under the same namespace (Add members to the groups and grant members all at once)
* Create Group, add members, easier to @mention
* Create Group for company, subgroups for your teams

Eg. Group – **team** and subgroups like **frontend-team, backend-team**

**@team/backend-team** // way of calling i.e. @<group-name>/<subgroup-name>

**@team/frontend-team**

Namespaces

* In Gitlab a namespace is a unique name to be used as a **username** or a **group name** or a **sub group name**

Eg.

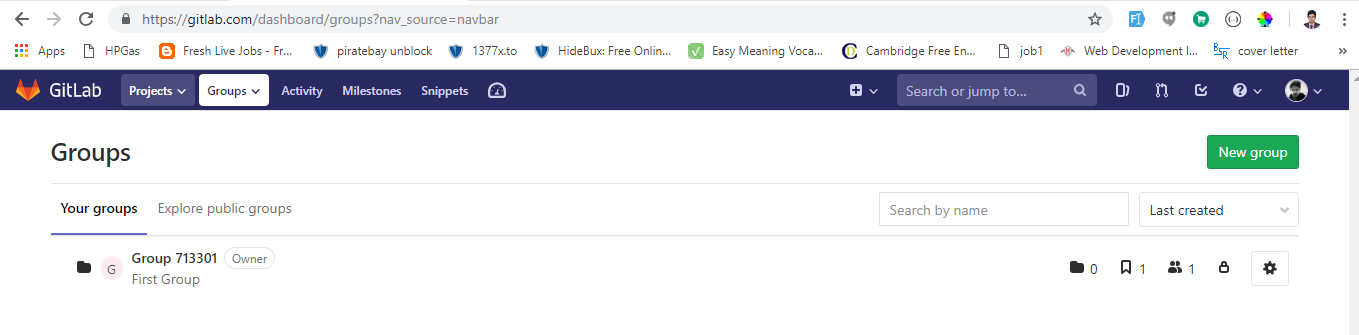
<https://gitlab.com/AmirMustafa> (Amir Gitlab Profile)

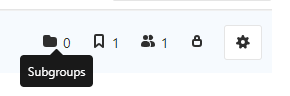
<https://gitlab.com/AmirMustafa-team> (Amir creates his group)

<https://gitlab.com/AmirMustafa-team/sales> (Subgroup named sales)

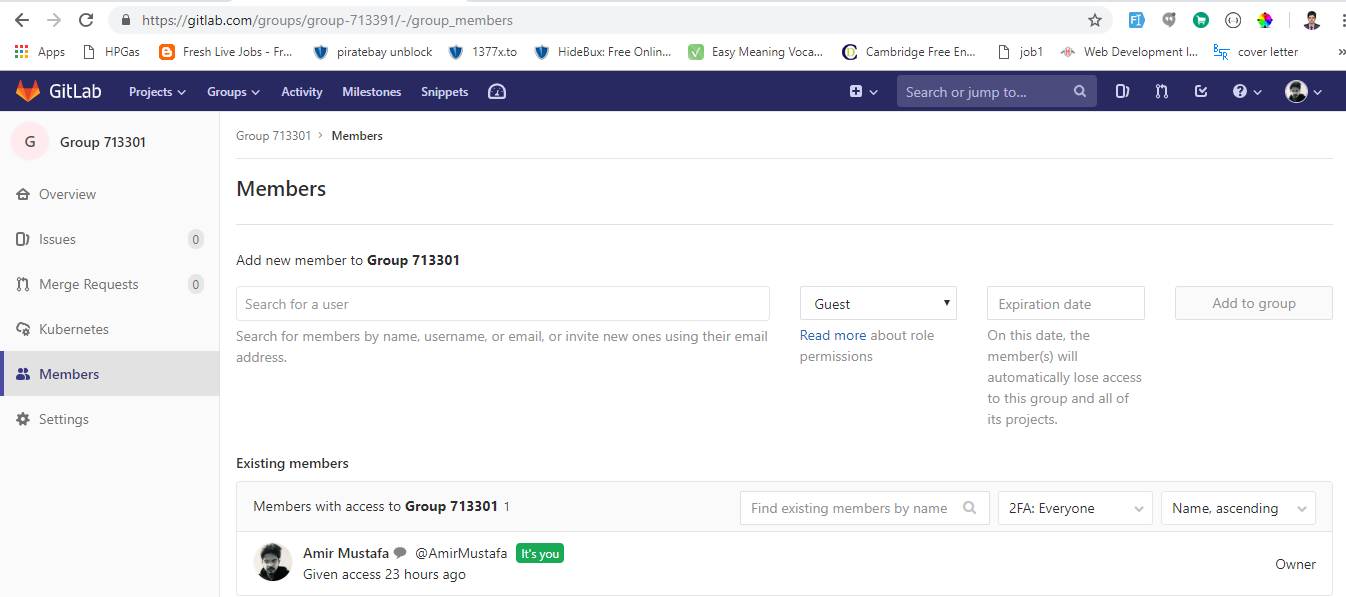
So creating groups and sub groups makes communication easier in Gitlab

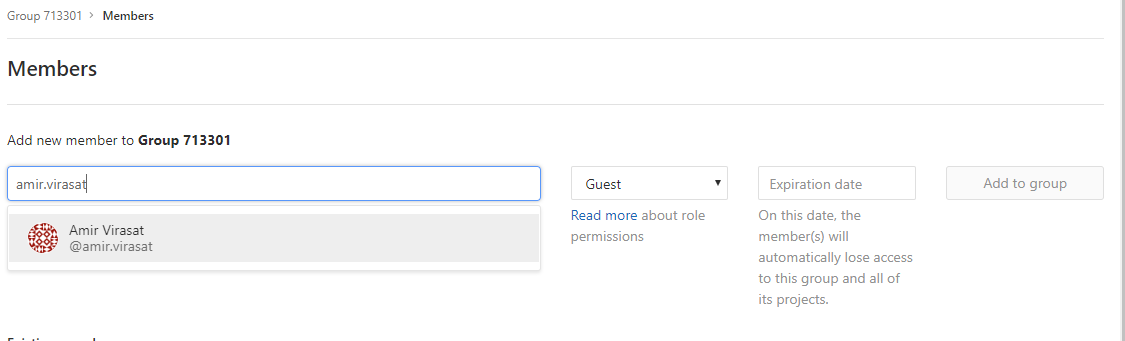
* Let’s go to the GitLab page. You can create groups from the group tabs:





* So once you create a group we have there several information in the main group such as subgroups, projects, members, visibility of the group respectively as shown in the screenshot above.
* To add member to your group 🡪 Go to your group 🡪 Click Member to your explorer and member similarly the way we have added the users in the project.





**Benefits of Multiple project in Group**

* You give a user access to all projects within a group with just one click action. Consider there is a new member to the group with developer access. You automatically give him access permission for the projects within that group.
* If necessary you can add same member to the projects with new permission to access specific projects.
* You can also change group settings similar we did In user i.e. private, internal (Allow request access) and public (Allow request access).

Gitlab Permissions

* Git is very easy to use and abuse as well. A single git push –force command can ruin the day for a lot of people.
* So before actual pushing to the main branch when working with lots of people in the group, Gitlab has a **very simple permission system**
* Fundamental idea is to have read or write permission to repo
* The following permissions are defined:

1. Guest – Have almost no access to the code (can only make issues in the project)
2. Reporter – Can only read the repository (i.e. can fork the project)
3. Developer – Can read + write to the repository
4. Master - read + write + partial administrative capabilities
5. Owner – read + write + full administrative capabilities

* If you want further protection to your code and to stop people with messing with your history or pushing code you can create what is called protected branch within gitlab

A protected branch does 3 simple things:

1. It prevents permission from everybody except the users with master permission
2. It prevents anyone from using the force push to a branch
3. It prevents anyone from deleting a branch

You can make any branch as protected branch. By default Gilab makes master branch as protected branch

More about protected branch - <https://about.gitlab.com/2014/11/26/keeping-your-code-protected/>

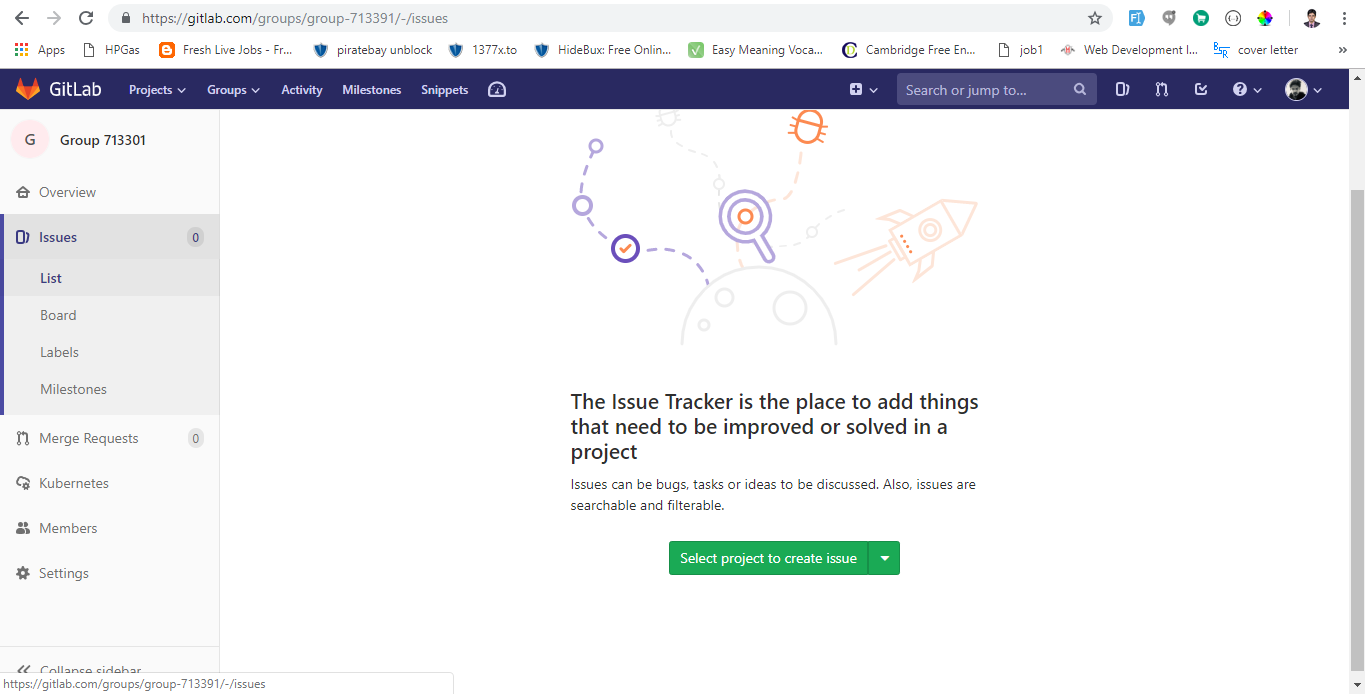
GitLab Issues:

* Issues are handeled in GitLab Issue Tracker. The GitLab Issue Tracker is advanced and complete tool for tracking the evolution of a new idea or process of solving a problem.
* Use cases of issues

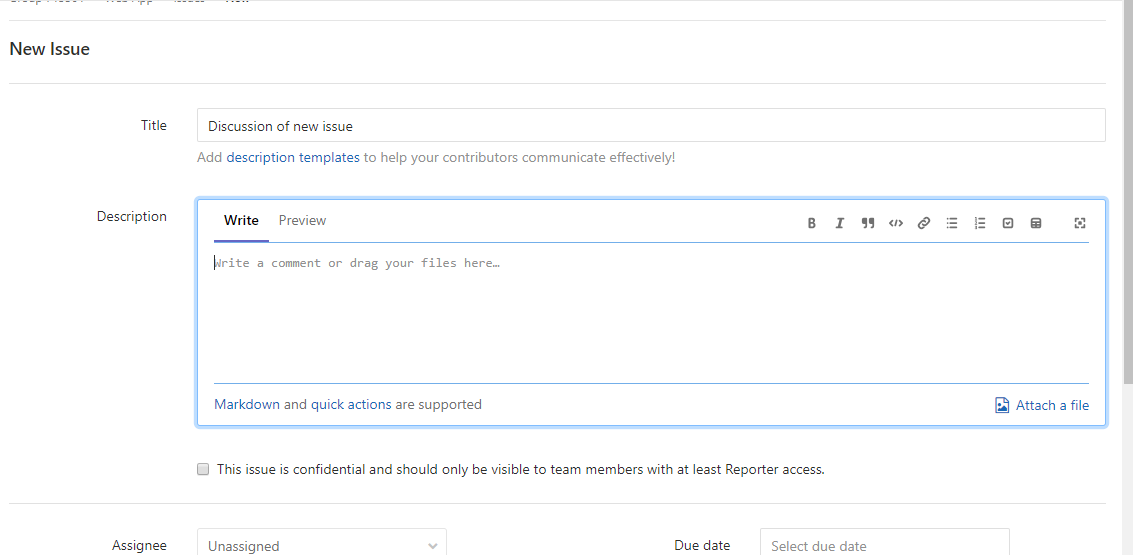
1. Discussing the implementation of a new idea.
2. Submitting feature proposal
3. Asking questions
4. Reporting a bugs and malfunction
5. Obtaining support
6. Elaborating new code implementations

* You can even have private issues (which you don’t want it to be public)
* Streamline Collaboration
* GitLab Issue tracker is the combination of **Open** and **closed Issue.**

Lets see in demo go to your project 🡪 Left explorer Click Issues



Click New issue by selecting project



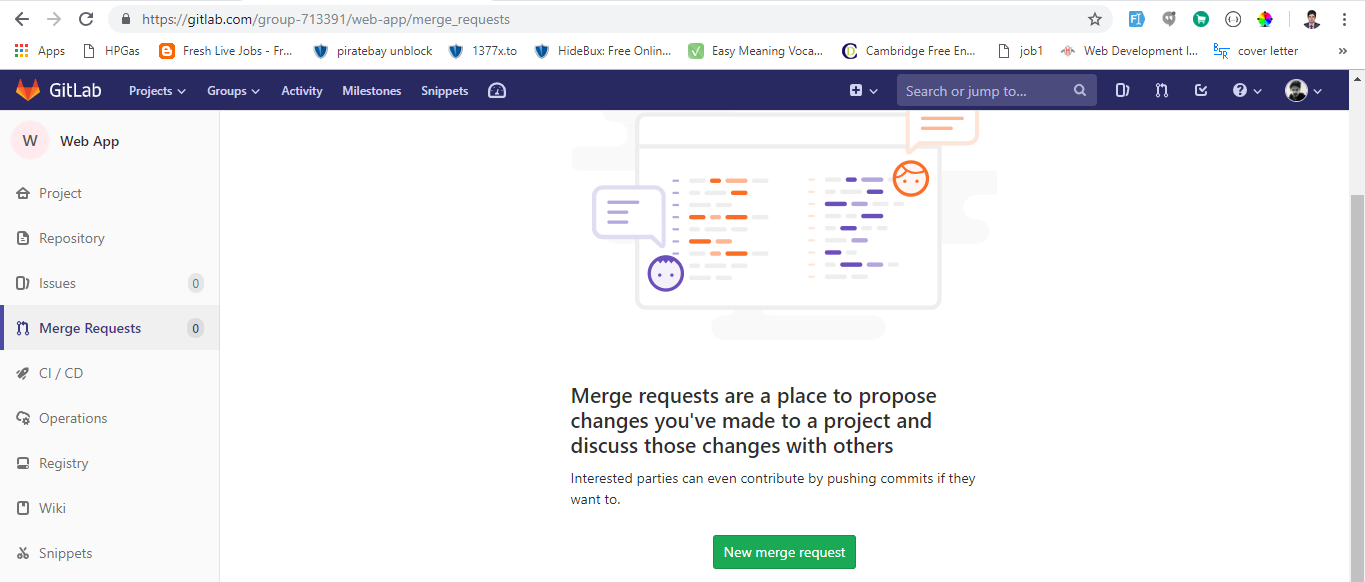
You can assign issue to specific person or keep it un assigned:

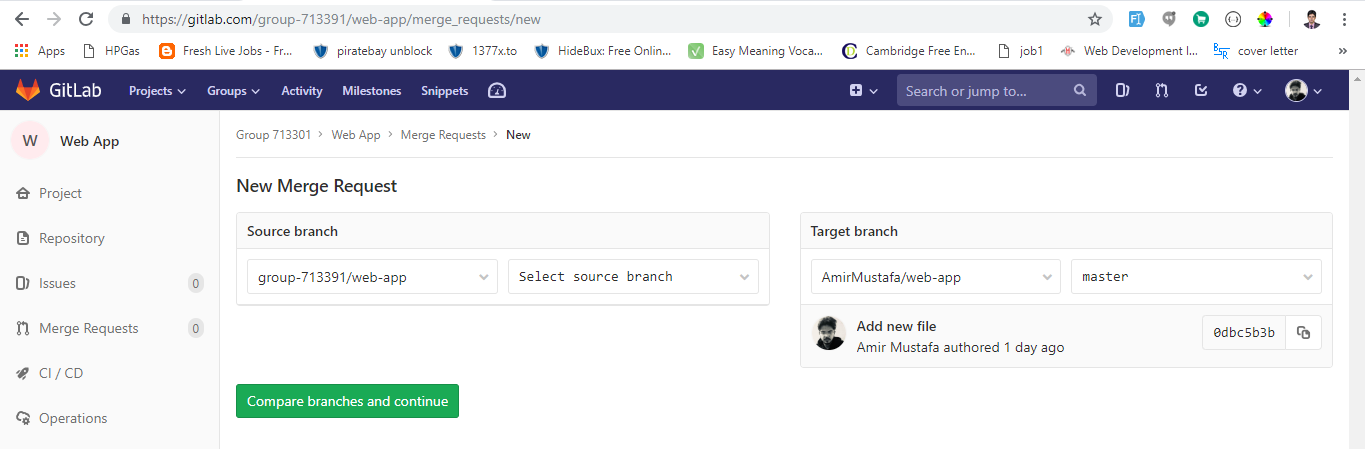


New issue is created. If your issue is resolved satisfactorily, you can close it as well.

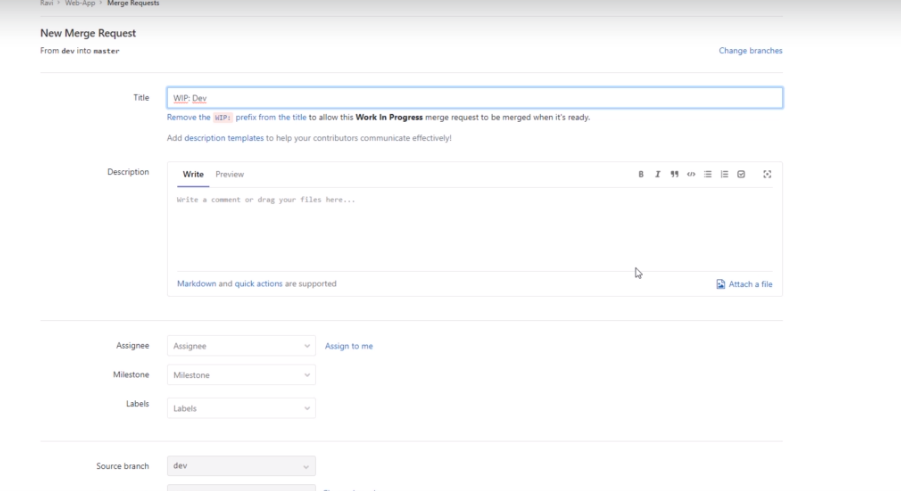
Merge Requests:

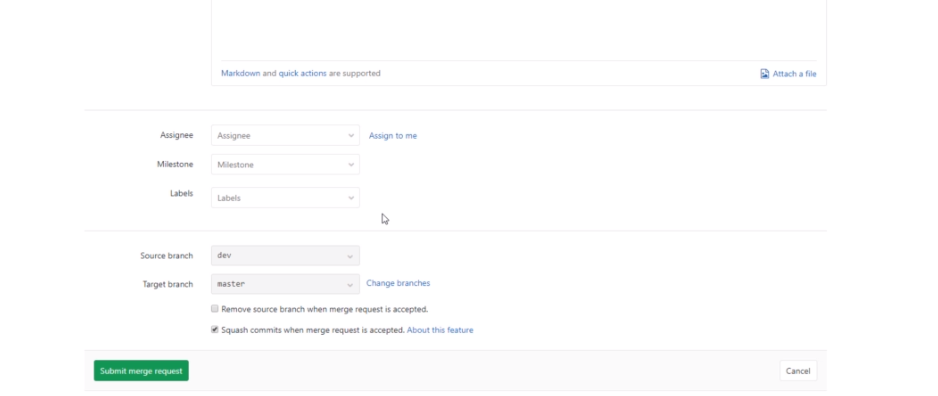
* Here we will discuss about how to create a merge request in gitlab. Merge request is used to integrate separate changes that you have made to a project on a separate branches.
* Requirement: You already need to have your branch and pushed your changes there.
* In GitLab go to you project change and click merge request





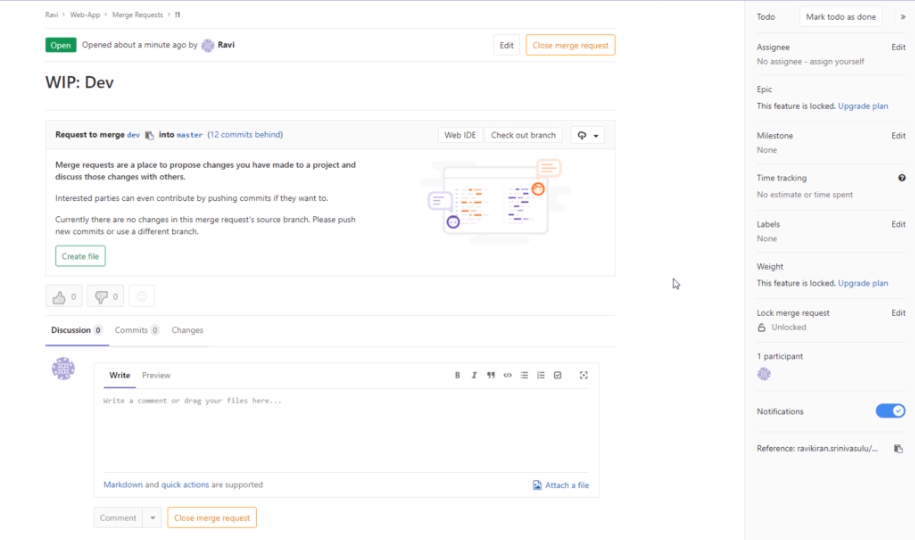
* To merge request, select the source branch and target branch and click the green button
* You will see a form to fill (request form)





Two options are there: remote source branch when merge/ squash commit (make all commit in one)

Squashing is the tidy of the merge request. So you get your page where you get details of the merge requests



Once the merge request is accepted you can go on and close the request. One person created the request and separate person who accepts it will close the request.



**Reference Issues:**

Here we will know about reference issue in gitlab.

* In Gitlab, everything is **cross-linked and referenced**. This improves discoverability and reduces duplicate effort
* Best practice to work with VCS(i.e Version Control System) is to **always start with an issue.** That way you have everthing to know about issue in one place and later as and when you make decisions keep the issue description updated as the single source of true.
* So in this concept what we will do is :

1. We will have an issue
2. We will fix that issue in the separate branch
3. We will merge changes in the main (master branch) and automatically that issue will be closed

🡪 Automatically closes after a merge request

<https://about.gitlab.com/2016/03/08/gitlab-tutorial-its-all-connected/>

**Creating Milestones**

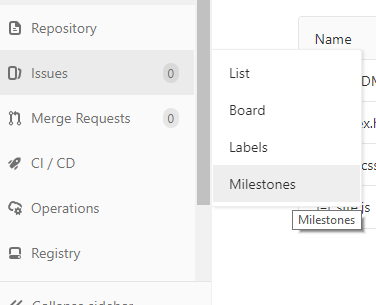
* A way to track issues and merge requests – created to achieve a broader goal in a certain period of time.
* 2 types of milestones:

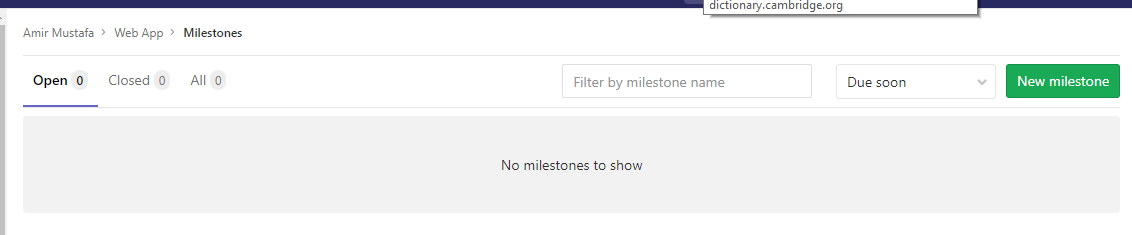
1. Project Milestones – can be assigned to issues or merge request within that project
2. Group Milestones – can be assigned to issues-----------------------------------------group

-> We will see how to create a project milestone and group milestone.

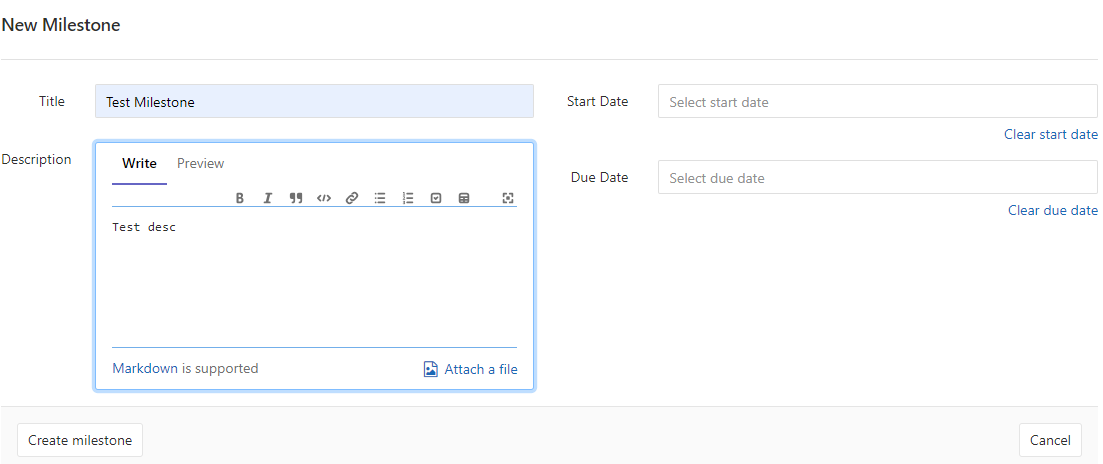
🡪 A permission level of Developer or higher is required to add/edit.

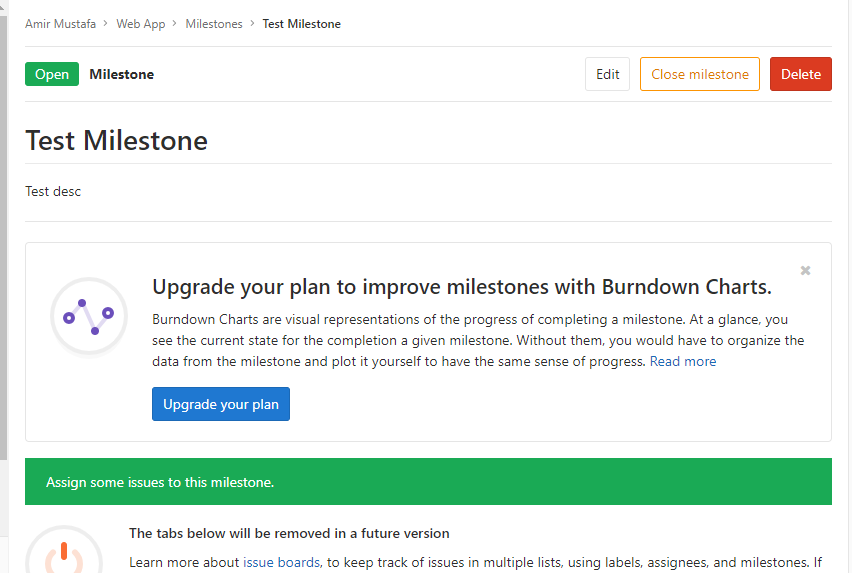
* Go to your project in gitlab and in select explorer issues 🡪 Milestone





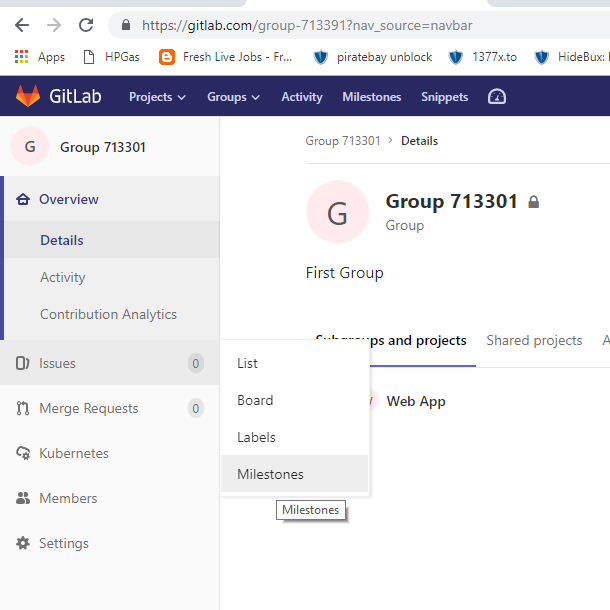
Select new milestone button

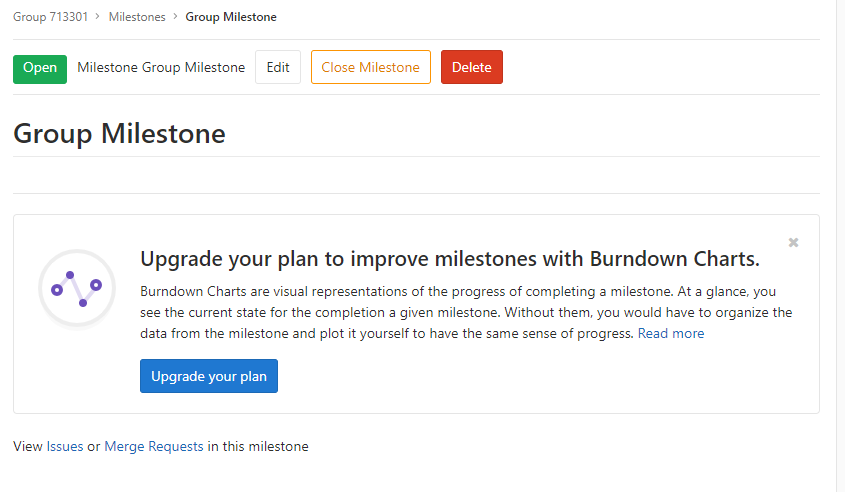




This is how **Project Milestone** is created.

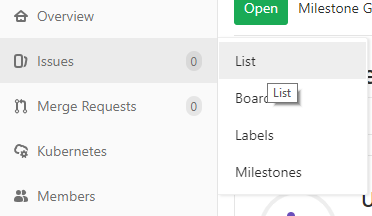
* To create the Group Milestones, you need to follow some similar steps but at a group level.

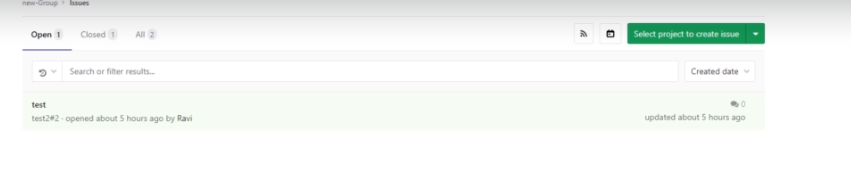


Create group in similar fashion

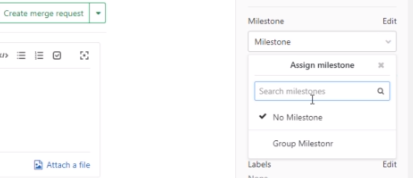
* One thing important you need to know is you can always promote a project milestone to a group milestone
* If you are expanding from smaller projects to a larger number of project within a group, you may wan to share the same milestone among same milestones among multiple projects under same group.

To promote a milestone goto Issues 🡪 List and click the issue

and 



On right hand side, select group to promote save. Done successfully and message will be shown.



Importing projects from BitBucket/Github repositories

* Import projects using importer
* The following aspects are imported.

1. Repo description
2. Git repo data
3. Issues
4. Pull requests
5. Wikis
6. Milestones
7. Labels, Release notes, Pull Requests(PR) review comments

* **A few things to check before import:**

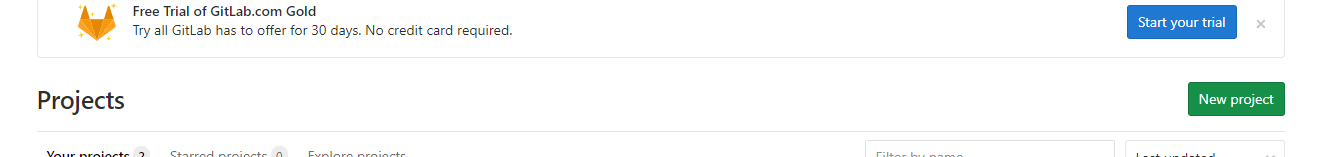
When issues and pull requests are being imported, the importer attempts to find their **Github author or assignes in the database of the gitlab instance** and

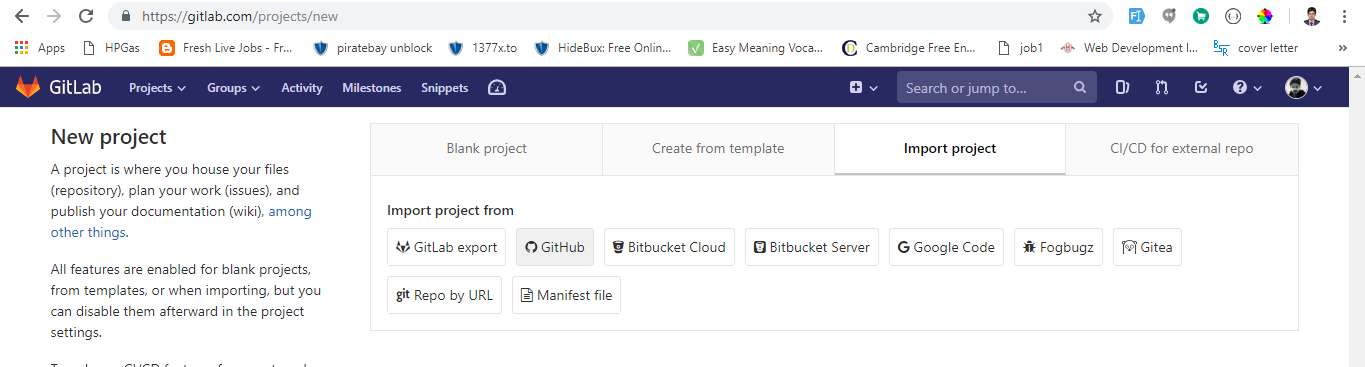
For the association to success prior to the import each gitlab or github author must have previously logged in to the github account icon or gitlab account with same email address of the github account.

Note that if the user referenced in the project is not found in the GitLab’s db the project creator typically the one who have initiated the import process is set as the author/assignee but not on the issue mentioning on the original github is added.

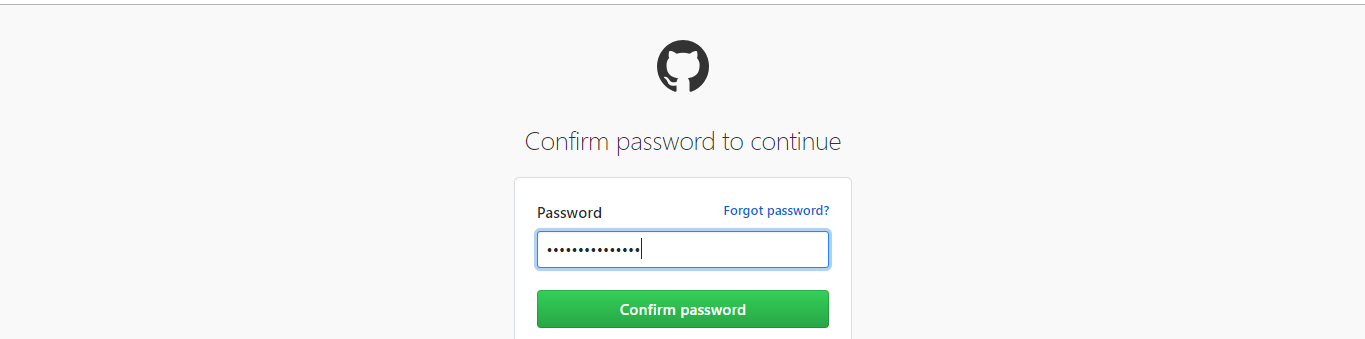
* Lets see in gitlab:

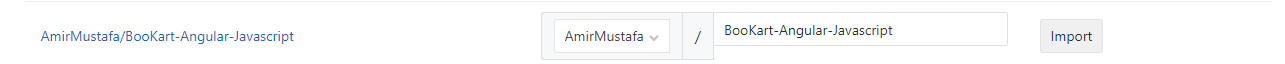
Go to the New Project 🡪 Select 3rd tab Import 🡪 You can import the repo from any Bitbucket Gitlab ..



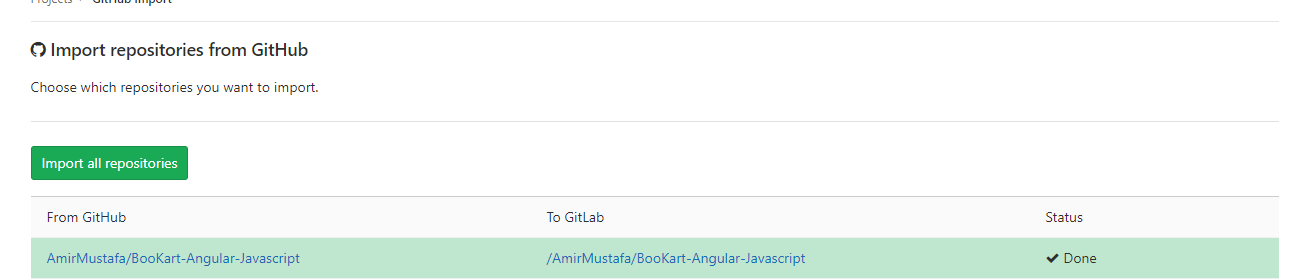


It asks to login

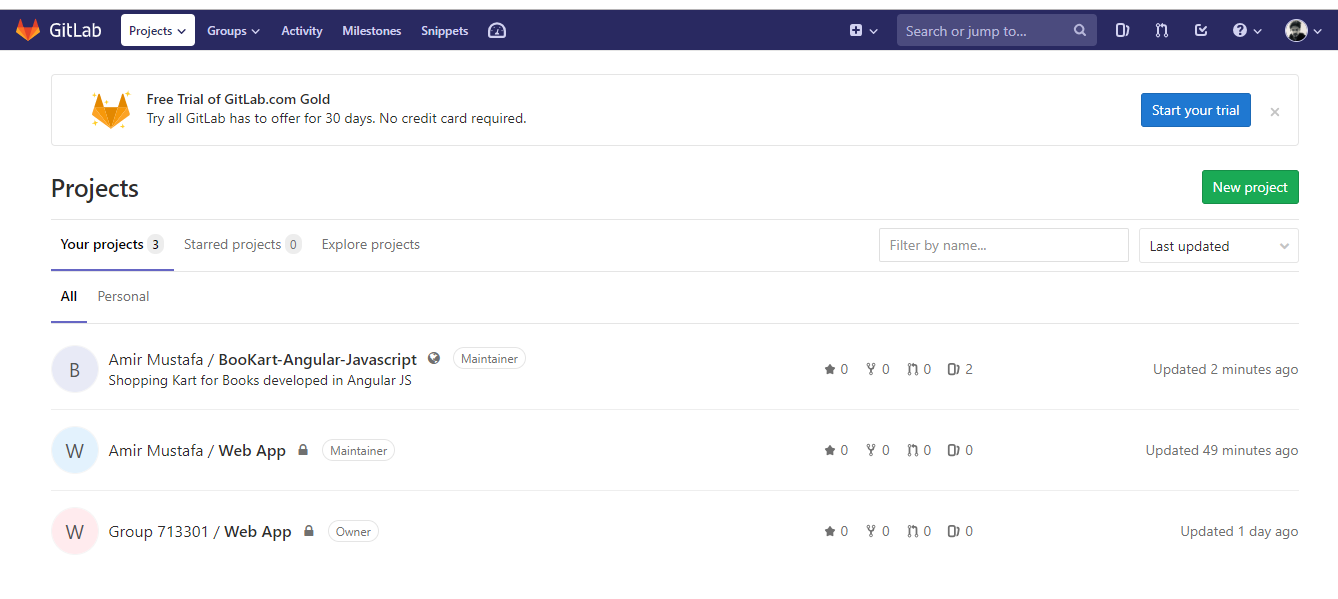




Click Import



Now click the link



We have successfully imported project from Github to GitLab.

**Continuous Integration (CI) in GitLab**

* This is the part of GitLab which allows which manages the project and allows unit test on every commit you make and indicates with warning message whether there is unsuccessful build or not

Features of CI in Gitlab:

* Integrated in the GitLab interface
* Simple usage, faster results
* Allows project team members to integrate their work daily
* Automated build- it allows integration errors to be identified easily
* Support for multiple platforms – windows, UNIX, OSX and other platforms (Gitlab CI is written Go programming language)

**Advantage:**

One advantage I have noticed using GitLab CI is that it displays the results faster as it divides each build into multiple jobs that runs on multiple machines.

In addition to the CI let’s see **pipelines and jobs**

**Pipelines and Jobs**

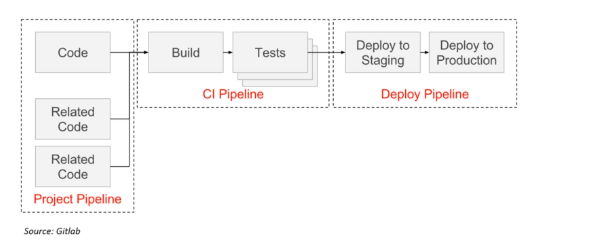
* Pipeline is a group of jobs that get executed in stages. Lets see with below example



Here this is a pipeline which has 4 different stages and each stage has one job within it. All of these stages are run in parallel and each job run concurrently (executed by **Shared Runner** or **Specific Runner**) created within a GitLab. Only when all the jobs succeed the pipelines moved into the next stage.

You can access the pipelines page in the project pipelines tab

**Types of Pipelines:**



These are of 3 types –

1. **Project pipeline** – It has cross project dependencies which are triggered via API particularly for micro services
2. **CI pipeline** – here build and test files are defined in a file called **.gitlab-ci.yml**
3. **Deploy pipeline(or Discontinuous delivery pipeline)** – These files are also defined in **.gitlab-ci.yml**

**Job configuration:**

* Job configuration with .gitlab-ci.yml file . YML/YAML file that defines a set of jobs with constraints stating when they should be run. It can have unlimited no. of jobs with and must have atleast script clause . So within project you write configure CI and CD, it will ask you to create this particular file.
* So basically if these two files (CI and CD) is created and runners are configured basically these two are only required to have continuous integration. Gitlab will automatically pick up jobs if this file is defined.

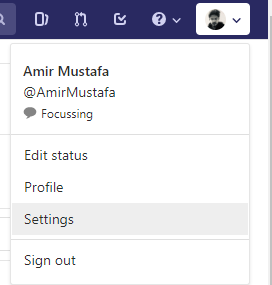
GitLab Notifications:

* GitLab has the notification system to notify the users of events that are important for workflows
* You can find notification settings under user profile. Notification settings are divided into 3 groups:
* **Global Settings** – bottom of the hierarch, any setting here will be overridden at a **group/project** level
* **Group Settings –** you can have different notification setting per group(can be configured in group page or user profile notification page) . Any setting here will be overridden at a **project level**
* **Project Settings** – This setting is on top level. Any setting over this will be overridden eg. Group setting will be overridden by project settings. This setting can be configured under the project page. Similar to the group setting they can also be configured to the user profile notifications.

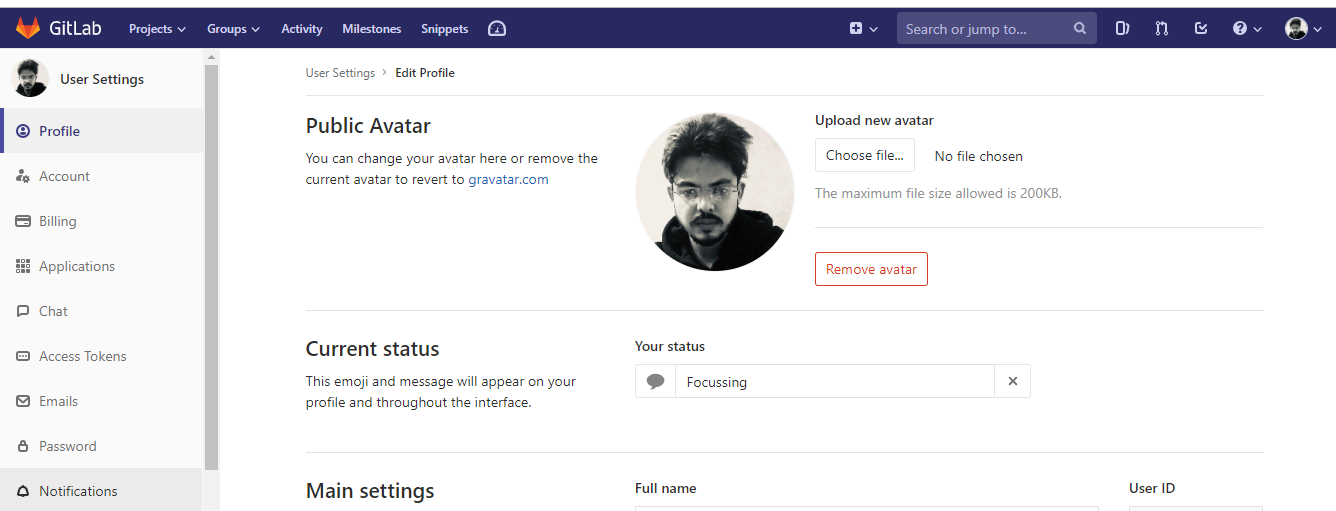
Project Setting > Group Setting > Global Setting

* Each of these settings have 5 level of notifications
* **Disabled** – if you chose this at any of the above 3 settings you will not get any notifications via email
* **Participating** – you will receive notifications for threads you have explicitly participated
* **Watch** – you will receive notifications for any activity
* **Global** – here in this setting the groups and the projects will inherit the settings that is given at the global setting.
* **Custom** – you will get notifications for specific events you chose. Eg. I only want to be notified when new merge request is made.

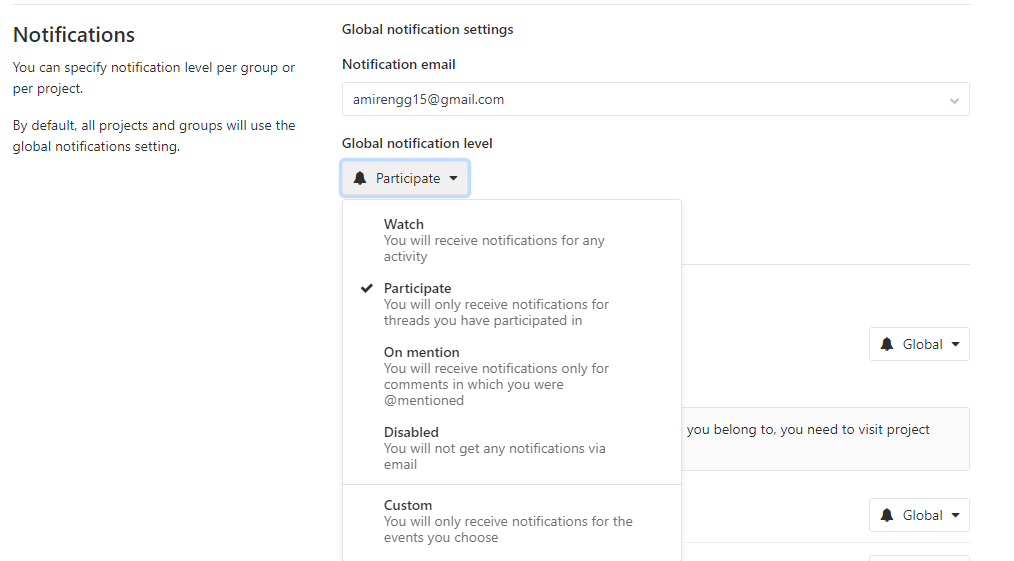
Lets see demo in GitLab : User profile 🡪 Settings



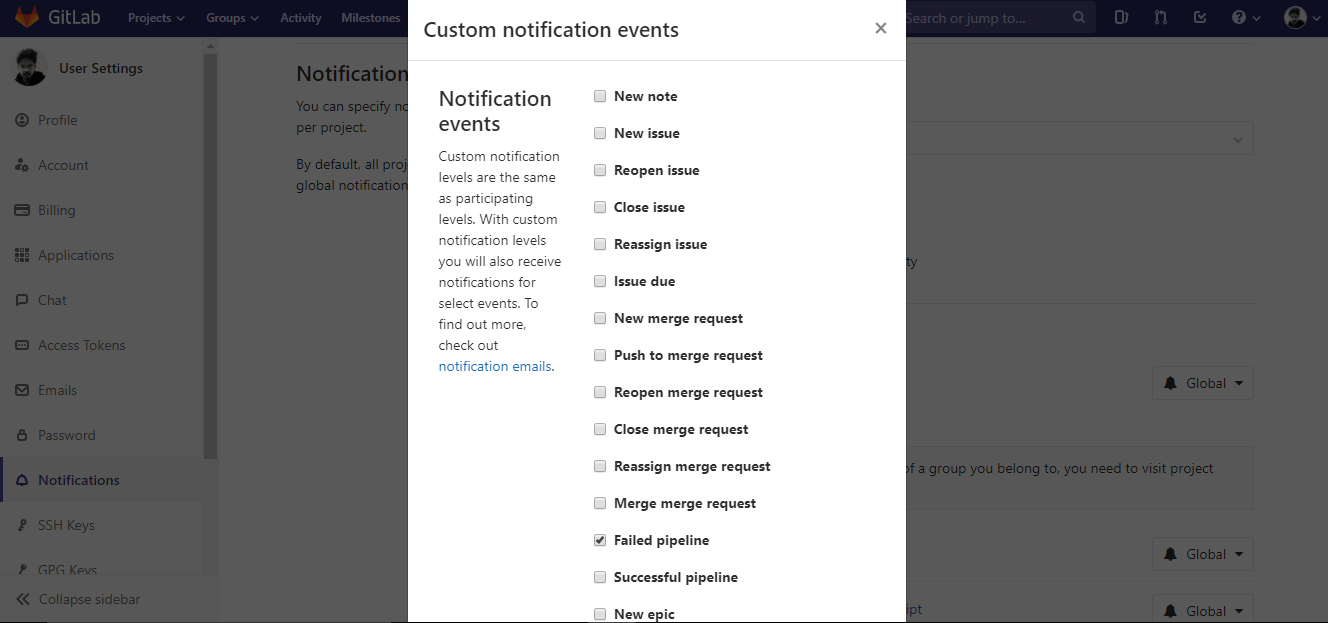
On the left panel go to notifications:



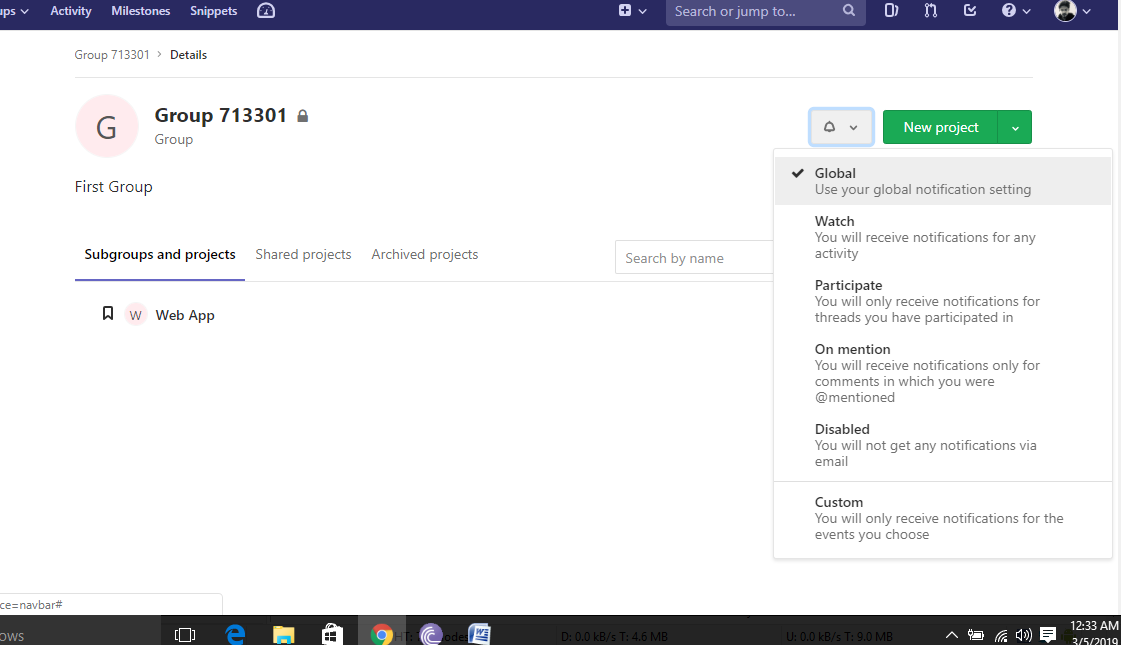
* Here we see global notification level



If you chose custom you will get further option to select the event you want to be notified of



* You can even select in group level – will auto reflect in main profile level.



* Go to project and click notification type – select as project level

**Understanding the YAML File:**

* In this we will talk about Gitlab continuous Integration YAML file

**YAML File**

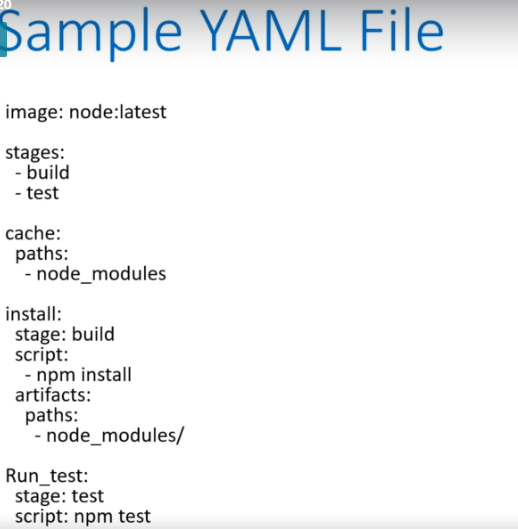
* To have a working CI in Gitlab, you need to do 2 things:

1. Add a .gitlab-ci.yml - Add this file at the root of your project repository
2. Configure a Runner (This one we have already done)

So when you have this file in root directory of the repo and configure this project in the runner, then each commit or each push triggers your continuous integration pipeline

* Gitlab CI looks for a file called **.gitlab-ci.yml**
* It tells the runner what to do
* By default it runs pipelines with 3 stages. A stage is nothing but collection of jobs
* By default it runs build tests and deploy stages.
* You don’t need to run all 3 stages. Stages with no jobs is already ignored by the gitlab runner.
* So if everything is OK, you will get nice green checkmark for every commit. This makes it easy to check if a commit caused any of the test to fail even before you look at the code. So most of the developer is using the Gitlab CI service to run the test suit so that developer gets immediate feedback if they broke something.

Let’s see a sample YAML file



* The file uses YAML configuration syntax to define the actions that need to be taken and the order they should execute, under what conditions they should be run and resources necessary to complete each task.
* When you are writing your own YAML file you can visit your syntax linter or CI linter by going to the CI/lint in your gitlab instance which we will discuss in a bit to validate whether your file is correctly formatted or not.
* For YAML page always use spaces and not tabs

Let’s discuss about this YAML file (see above image for reference):

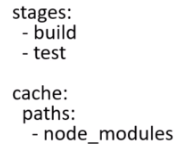
* Image: node:latest – YAML starts off by declaring the docker image.This is used to run the test suit.

So in our case our code is running in the different server instance and our tests in the different docker instance.

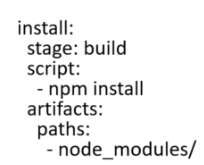
* Any CI pipeline is build in the stages and what we are doing is we are explicitly defining continues integration (CI) stages that will run. There are two stages we have defined. 1st one is build stage and 2nd one is test stage.
* The name you chose here is arbitrary but the way in which it is define (i.e. ordering) follows. Stages are task you apply to individual jobs. Gitlab will run jobs of the same stage in parallel and will execute task of the next space until all jobs of the current stage are complete.

Gitlab uses **build, test and deploy** stages. If no stages is defined there **gitlab will assign test stage by default**

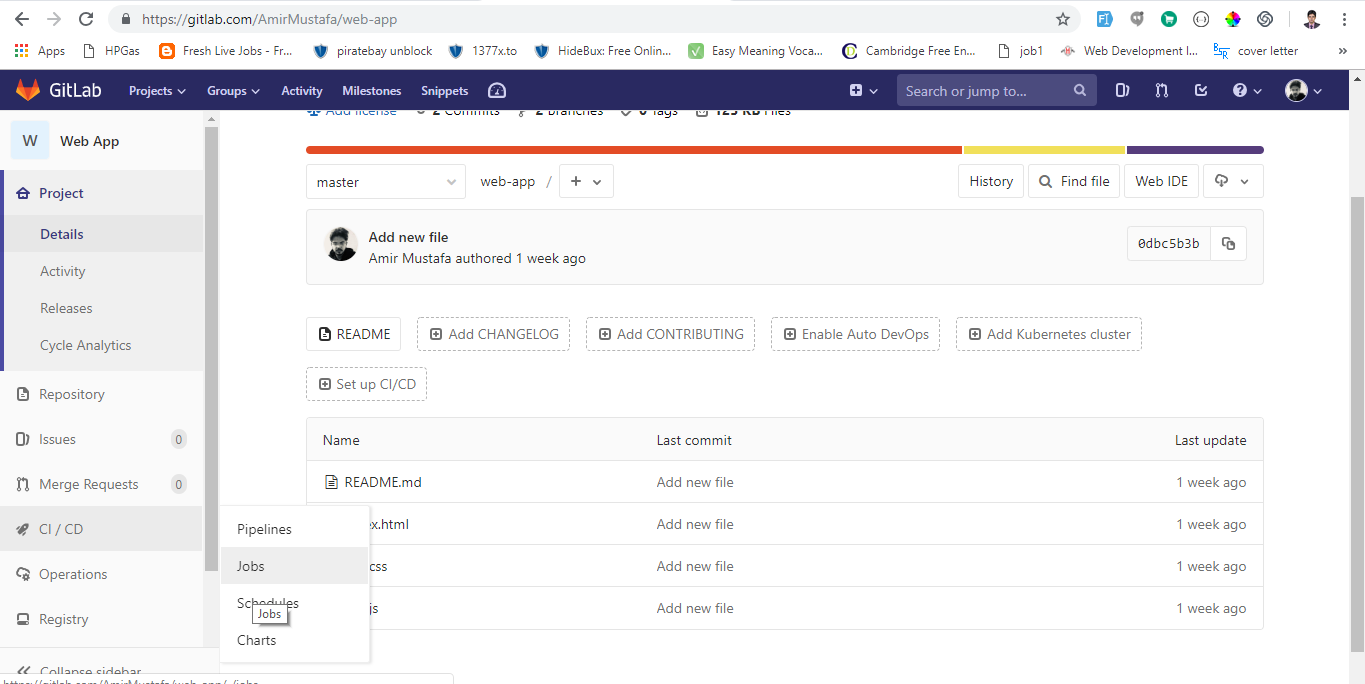
* The YAML file after defining the two stages which are build and test, it includes the **cache definition (point 2 prev image)**

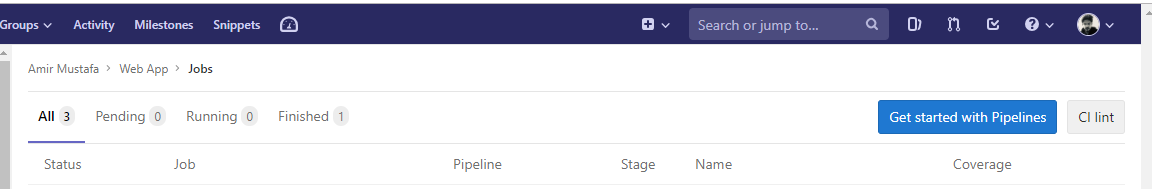


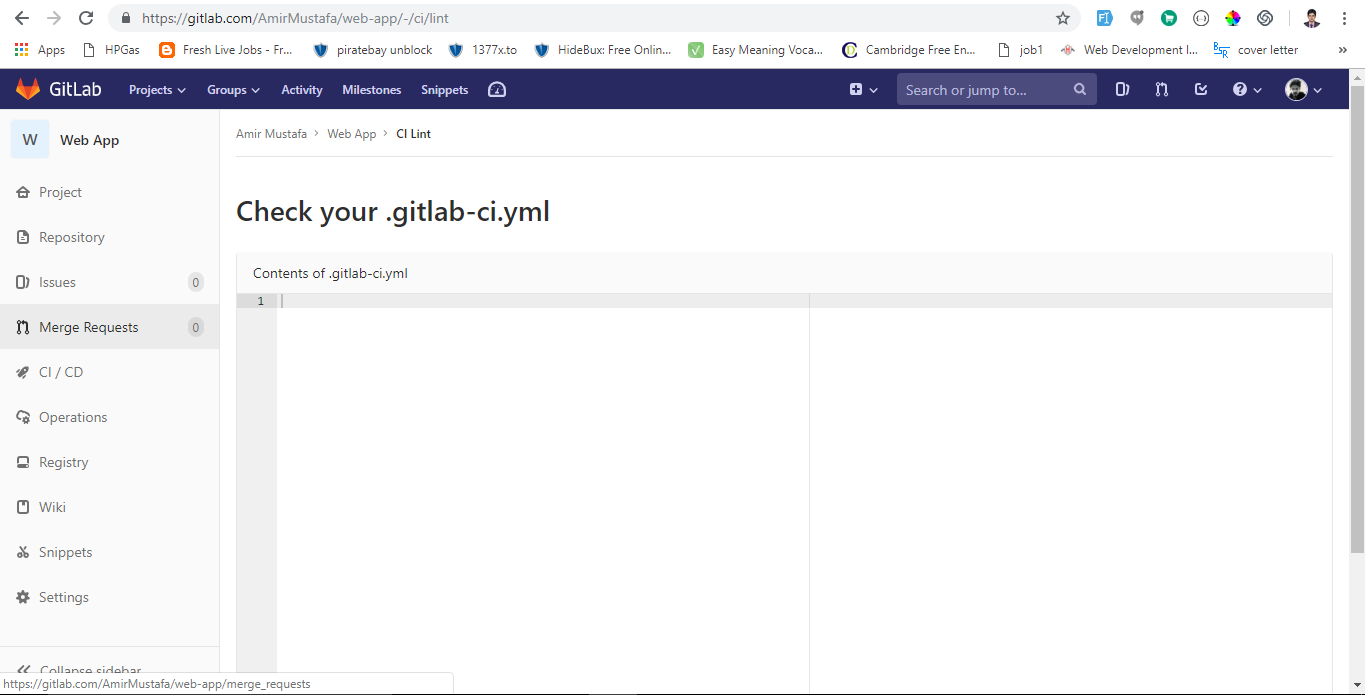
* Here we are caching the **node modules.** The part you need to indicates what particular location needs to be cached. You can think like it basically specifies files or directory that can be cached or saved for later use between different runs (means different runners) or different stages (means different stages within the same run).
* So caching like this can help the amount of time it takes to run jobs that relies on resources that might not change that might run. So node\_modules are not going to change within different runs between stages. Its better to cache all of them
* Here node\_modules is where all the dependencies will install.



* Here our first job is called **install** . Like stages your jobs can be called anything. As your name will be named in the gitlab UI descriptive names are really useful. This job has the particular i.e. to run npm script. We mark this stage as build. Usually within the job we can write additional job by adding the additional line. Here we are using only 1 command i.e. npm install
* The **artifact** section is used to specify file or directory path to save and pass between stages. Note that this YAML file defines two stages: build and test stage
* In the build stage you have one job that is install and in the test stage you have another job i.e. test. Hence to pass the data (i.e. node modules) to the installed job to the run test job or between the build stage and test stage. We are putting node\_modules inside artifacts which means this particular folder will be accessible from the other folders as well
* So currently our YAML file is not doing lot of jobs. Only two stages: first stage is installing the node modules and second stage is doing npm test
* If you want to pass everything produced to the next stage: untracked: true . So here test is the second stage and has access to the project dependency (of stage 1 i.e. all node\_modules)
* So if you are building out your own YAML file, you can actually go to the YAML Lint / CI Lint to go and check whether your syntax is correct or not







Here you just need to paste the contents of your YAML file. If everything is OK you can paste this is the root of your project repo.

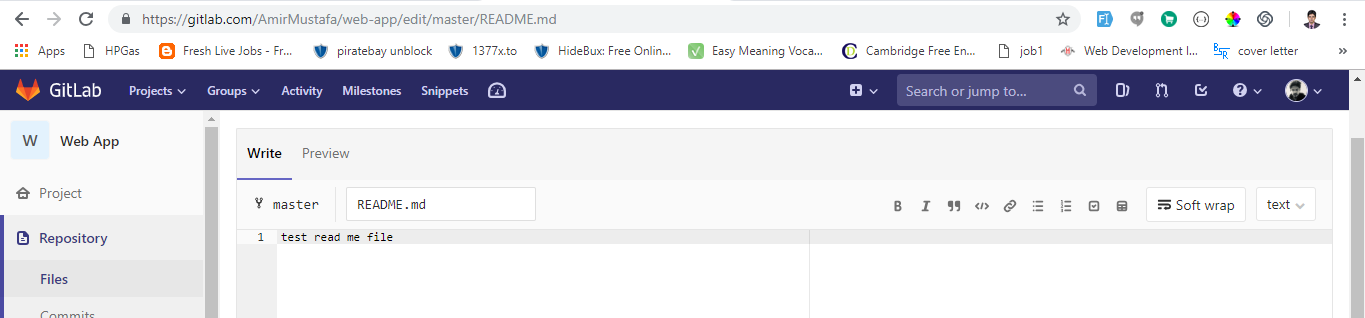
Trigger the pipelines

* Here we will discuss how to trigger CI pipeline in Gitlab

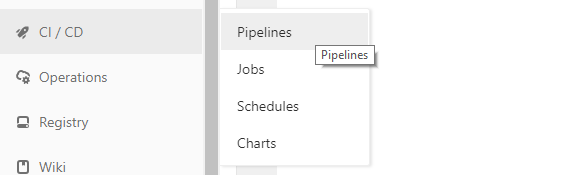
We will do the following:

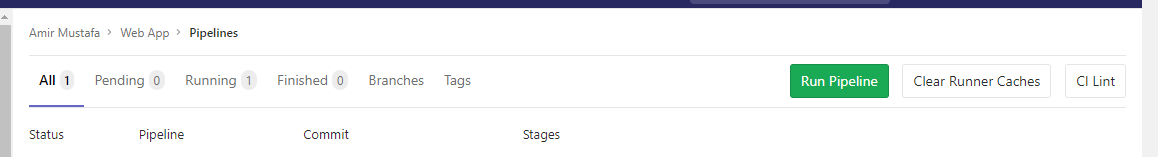
1. First commit with a file
2. See what happens
3. Create a YAML file
4. Observe the CI pipeline trigger

* Now just go to the read me file just to update something so that it will generate a commit for us.



Now go to the pipelines





i.e. no pipeline generated

* Lets create the gitlb-ci.yml file:

image: node:carbon

stages:

- build

- test

cache:

paths:

- node\_modules/

install:

stage: build

script:

- npm install

artifacts:

paths:

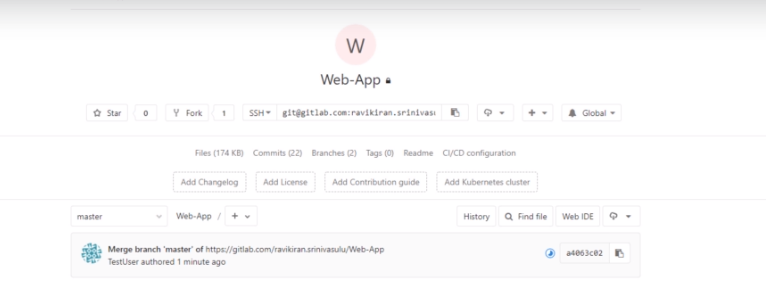
- node\_modules/

test:

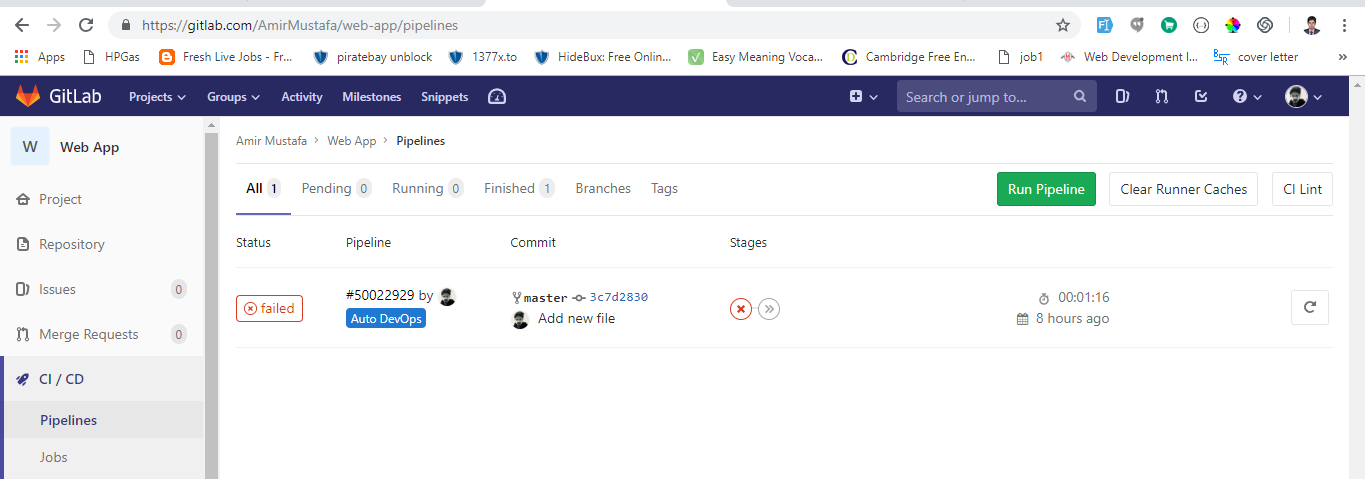
stage: test

script: npm test

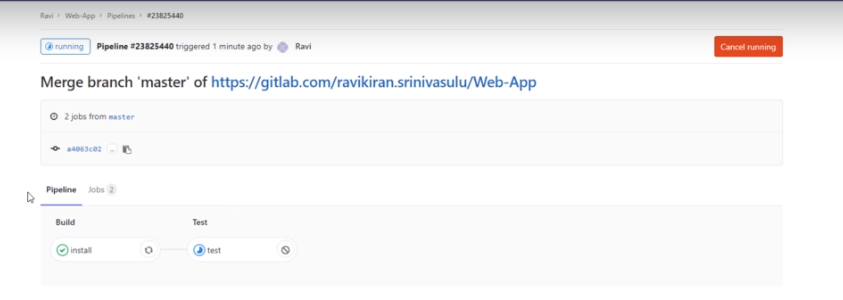
* So when you upload the YAML file the continuous integration has been triggered with that blue circle in merge branch row.



So when I go to the CI/CD pipeline, pipeline is trigerred

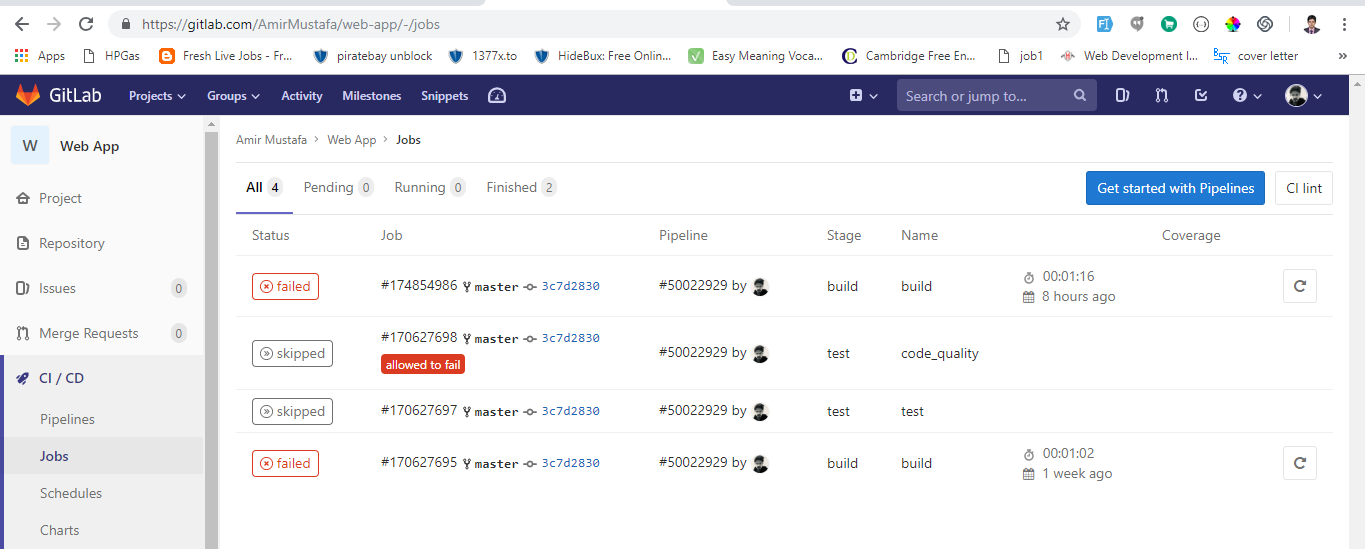


We click on the pipeline:

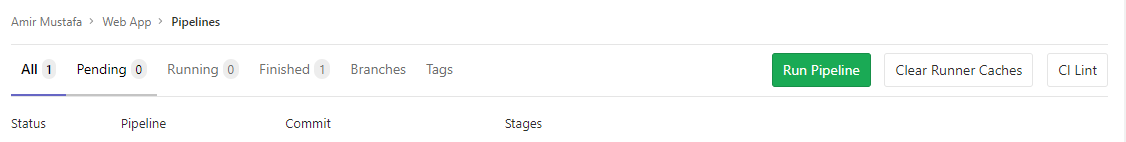


We see it has two different stages that we have defined in the YAML file: install and build. The build stage has successfully completed and the test stage is still running.

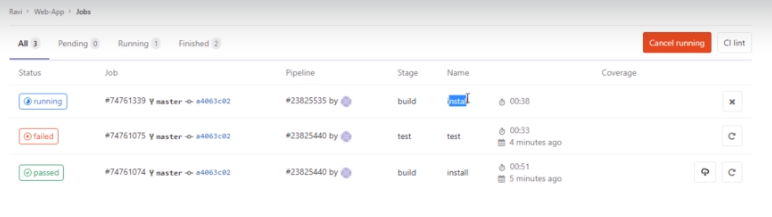
You get different messages you go and debug in CI/CD 🡪 Jobs:



If you want to run the pipeline again. You can go and in the **CI/CD 🡪 pipeline** and click **Run Pipeline.**



* By pipeline I mean what are the tasks/jobs configured as the YAML file.
* You can create pipeline for any branch
* In the jobs page you can see which job listshas been executed under which pipeline:

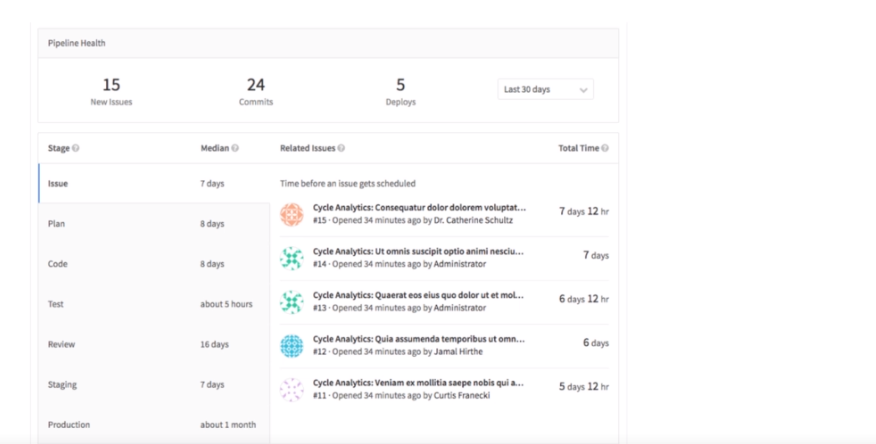


* Whether you change in the repo from command line or directly from gitab this CI (Continuous Integration) will run.

Cycle Analytics

Here we will see concepts of cycle analytics in GitLab.

* Measures the time from idea to production – for each of the projects you have. We can break time in multiple stages to see which stage is taking more time and we will optimize accordingly
* Cycle Analytics is tighly couples with the gitlab flow and calculates a separate median for each stage.
* You can find your cycle analytics page under your projects pipelines cycle analytics tab. Lets see a cycle analytics page from another project



* So this project has 15 issues, 24 commits and 5 deployment has been made
* On the left side you can see different stages and ideas gone through. The first stage is always the issue.
* So if there is idea you would like to propose, the best way is to always start with the issue.
* Cycle analytics record cycle time and data based on the project issues with the exception of staging and production cases where only data deploy to production or mature.

staging and production cases where only data deploy to production or mature

* So specifically of your CI is not setup and you have not defined production environment. You will not have this data up.

**Permissions for Cycle Analytics:**

Public – anyone can go to the cycle analytics page.

Private/Internal Projects – any member having guest member or above can access it.

So anything that will not flow the gitlab flow will not be tracked at all.

<https://docs.gitlab.com/ee/user/project/cycle_analytics.html>

Gitlab Container Registry

* A registry is nothing but storage and Content Delivery System – holding name docker images available in different tagged versions.
* Interact using docker pull and push commands
* Supports TLS and authentication

Why do you need Registry?

Running your own registry is the great solution to integrate with and complement your existing CI / CD

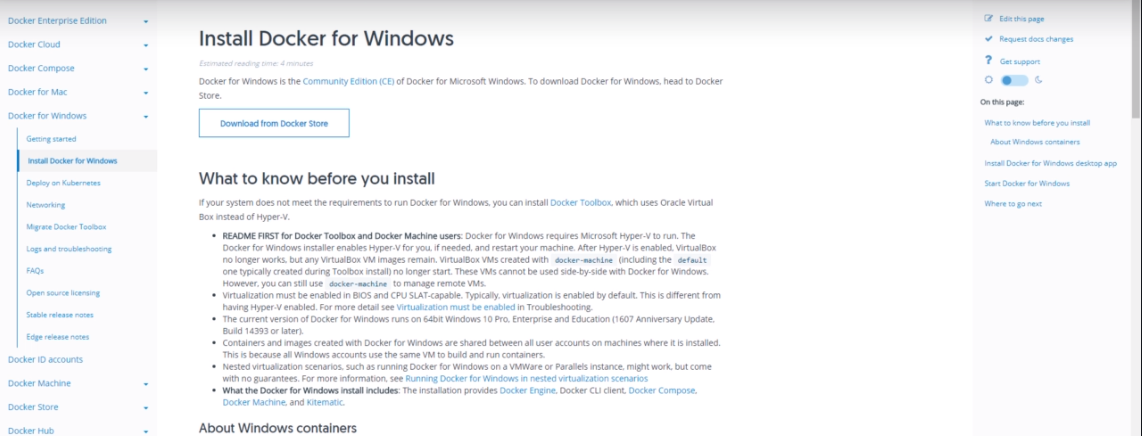
In typical workflow, a commit or source VCS (Version control System) would trigger a build to your CI system which will push a new image to your registry if the build is successful.

* So then a notification from the registry would then trigger deployment on a staging environment or notifiying other systems that a new image / commit is available
* Registry is also essential component if you want to quickly deploy a new image over a large cluster of machines
* Another advantage: It is the best way to introduce a multiple docker images inside an isolated network

Setup the Registry:

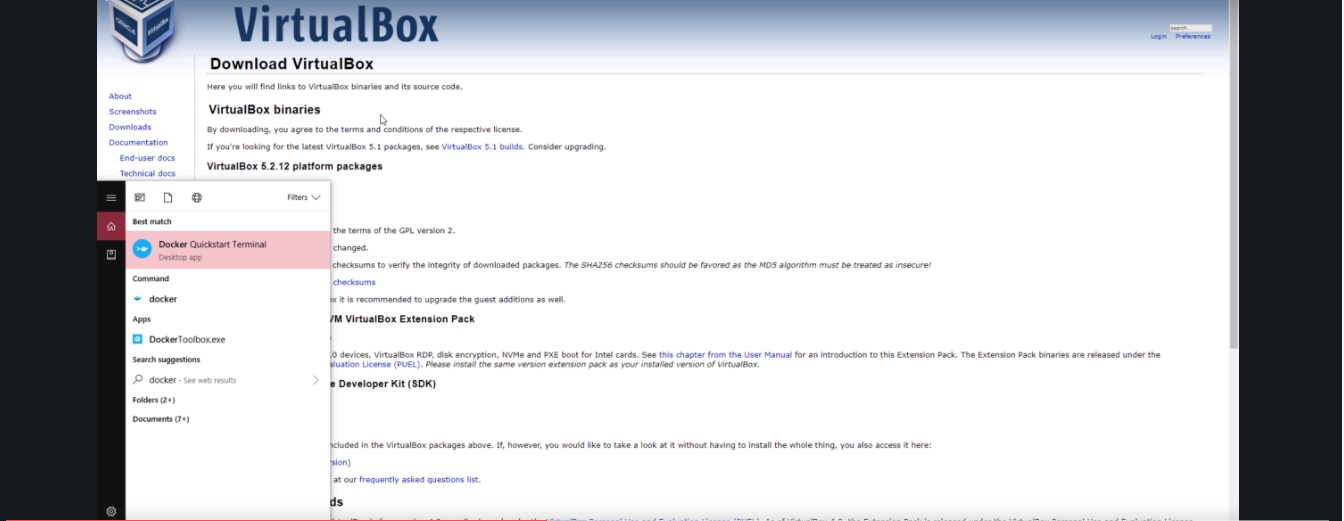
1. Install Docker for Windows
2. Install Virtalbox

Install Docker - <https://docs.docker.com/docker-for-windows/install/> ‘



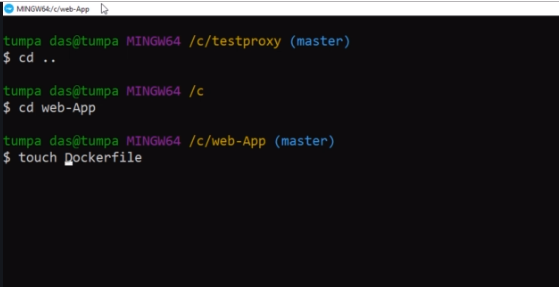
Download docker and install it. Once done Download **VirtualBox.**

* Now we are set to go to setup registry



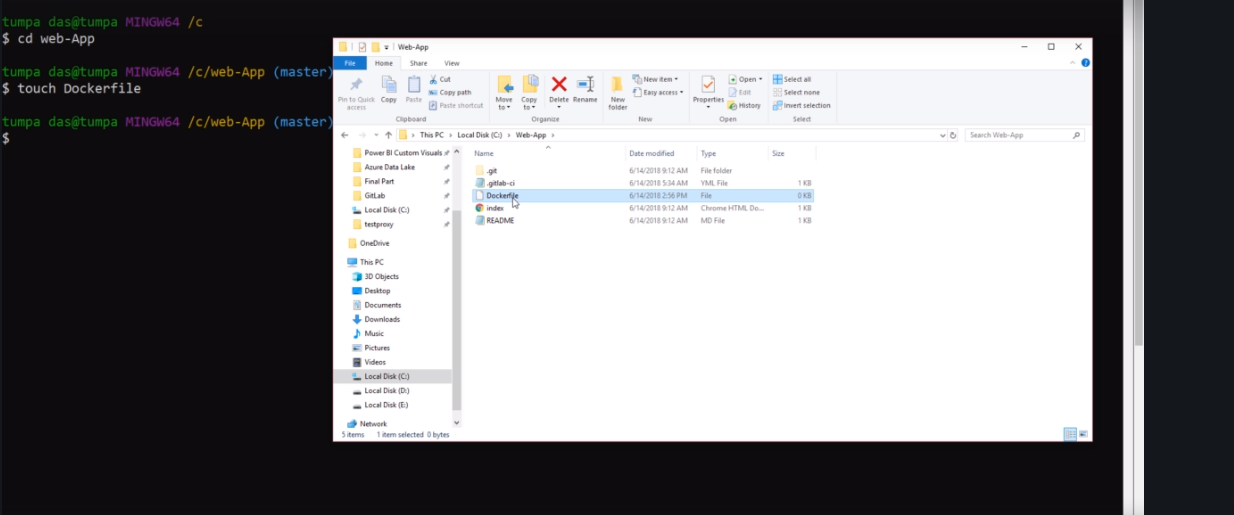
Now open the **Docker Quickstart Terminal**. So basically to work with docker image you need to work with docker terminal and not regular terminal/command prompt.

* Once you open your docker terminal just go to your project repository

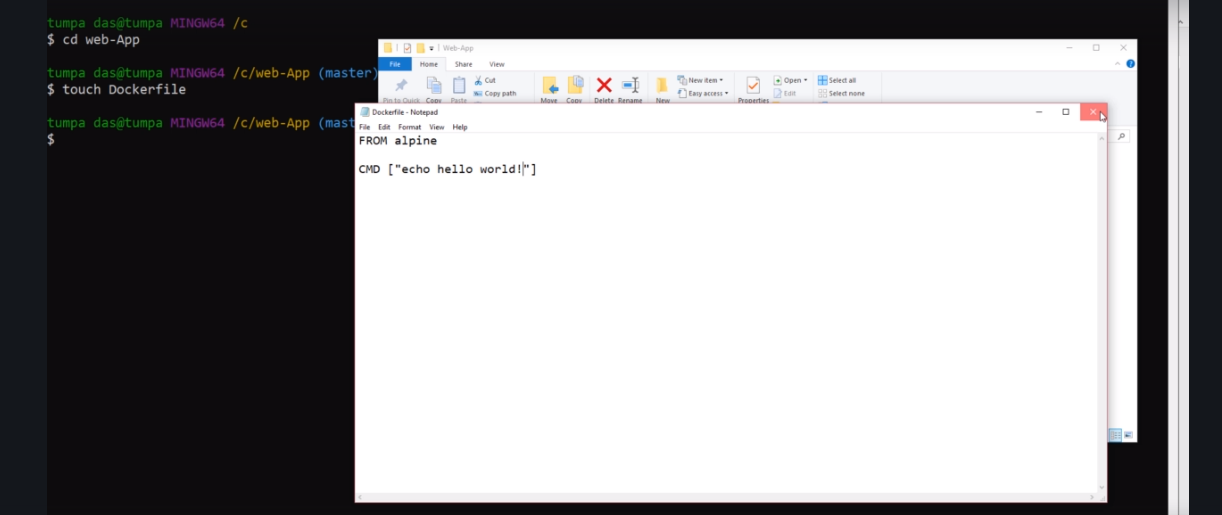


Now create the **Dockerfile** with D capital and f small. Basically we create a new docker file within your repository and it is this docker file which we will build and publish it.

So the image creation is successful, you go to your repository and see Docker image file and its type is file



We open that blank docker file and write some lines of code. We will chose **alpine** as alpine makes a good docker container because it is very small in size (around 5 MB)and completely optimized to run in your RAM.



1st line = type of docker (here alpine)

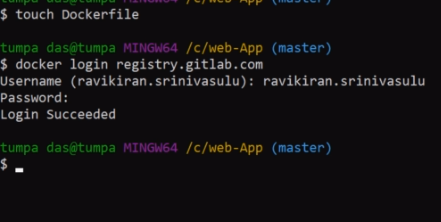
2nd line = actual command you need to execute (like printing hello world).

Now you have the docker image. So you need to login Gitlab’s container registry using your gitlab’s username and password.

If you have Multi factor authentication, you need to give additional token with it.

So let me login to your registry. To login to your registry you will be using a **docker command**

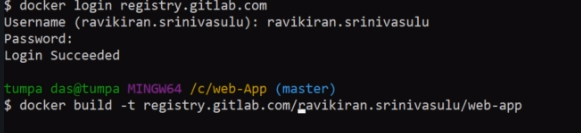
**docker login registry.gitlab.com – It will prompt for the username and password**

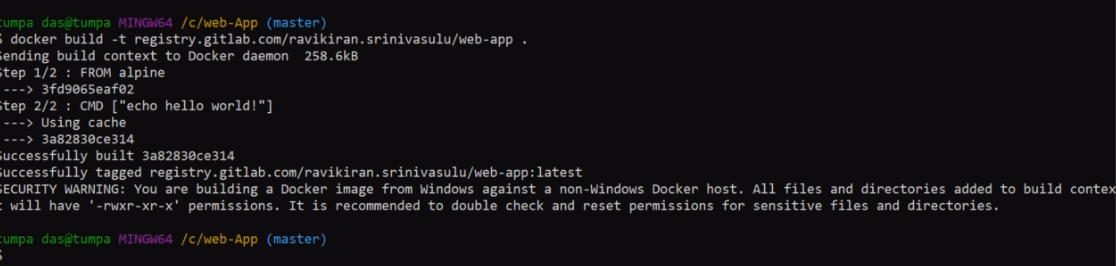
****

Once I login I am free to upload the container image using the common build and push commands.

So to build the image:

docker build –t registry.gitlab.com/ AmirMustafa/web-app .





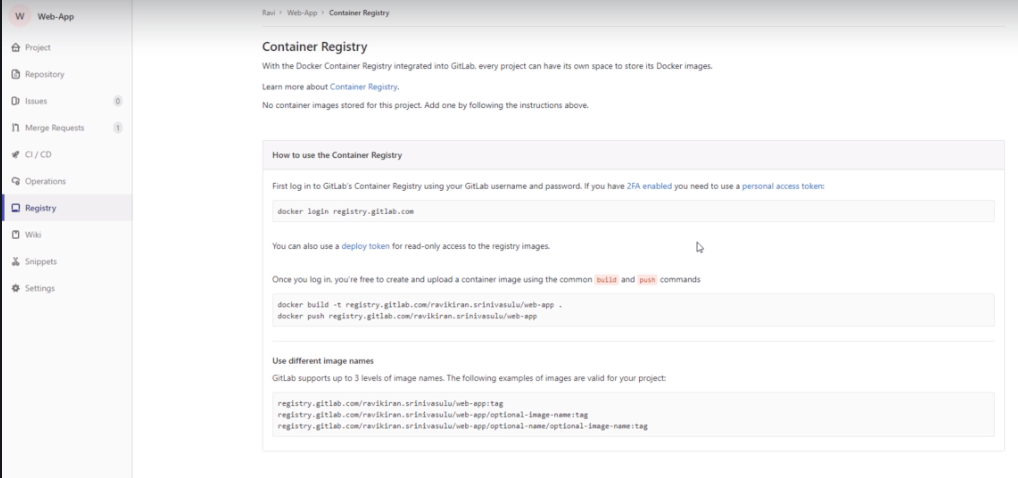
So we have successfully build the docker image and tagged the registry.

We get couple of warnings because we are on windows.

We will push the docker image now.

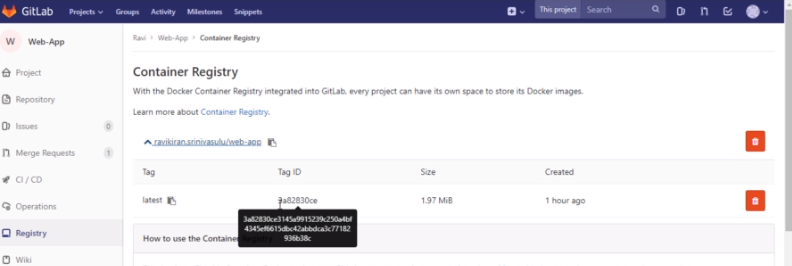
docker push registry.gitlab.com/ AmirMustafa/web-app

**Registry (Before push)**





Once we push we docker registry there



When you expand you see tag id and so on.

