

GITLAB CI/CD TRAINING

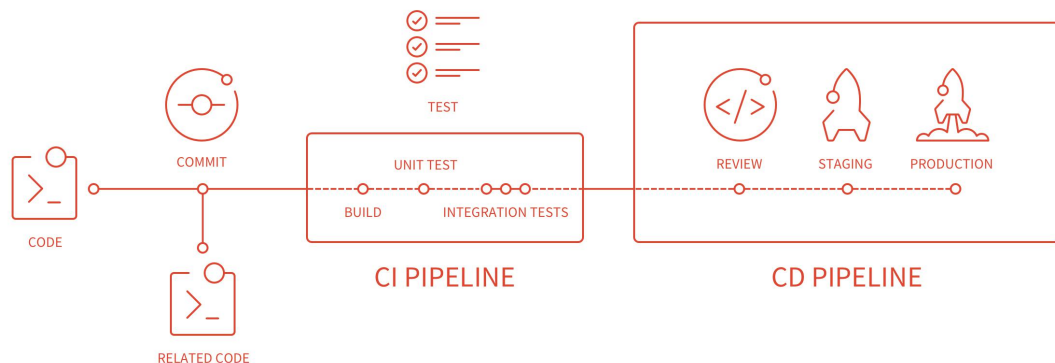


Table of content

1. Git.....	2
1.1. Git introduction.....	2
1.2. Prerequisites:.....	2
1.3. Video tutorials about Git.....	2
1.4. Git commands:.....	3
2. GitLab.....	4
2.1. Gitlab introduction.....	4
2.2. Practice GitLab:.....	5
2.3. Git and GitHub Basic Work Flow.....	6
2.4. Branching.....	6
3. GitLab CI/CD.....	7
3.1. GitLab CI/CD introduction:.....	7
3.2. Continuous Integration (CI):.....	8
3.3. Continuous Delivery (CD).....	8
3.4. Continuous Deployment (CD).....	9
3.5. GitLab CI/CD workflow.....	9
3.6. GitLab CI/CD Video series:.....	10
3.7. What is SSH.....	10
3.8. GitLab Runner.....	15
3.9. .gitlab-ci.yml file.....	21
1. .gitlab-ci.yml introduction:.....	21
2. Pipeline.....	25
3. Tag.....	26
4. Schedule (optional).....	27
4. Simple Demo GITLAB CI/CD.....	27

1. Git

1.1. Git introduction

- What is Git:

+ Git is a Free and open source version control system. Version control system is a software that helps programmers track code changes. Programmers basically save an initial version of their code into Git, and then when they update the code, they can save it into Git again and again and again.

+ Throughout time as the updated code continues to change, programmers can look back at all of the changes they have made over time. This helps to see and understand what they did when, as well as track down bugs, or go back to a previous version of code if we need to.

+ Git locally tracks changes in your project/folder and push & pull changes from remote repositories like GitHub, BitBucket, GitLab.

+ Git is the most widely used version control system in development today. Most programmers interact with Git on a daily basis.

1.2. Prerequisites:

- You should be able to use some basic commands in Command Line Interface(Terminal for Linux, Command Prompt for Window,) just for working with folders(like navigating between folders, creating folders, deleting folders), files (like opening file for editing, saving files, creating files, deleting files).

1.3. Video tutorials about Git

- Before moving to the following sections, I recommend you watch 2 Git Tutorial videos in the **first** 2 link below to have a feel about Git, how it works and how to use GitHub and some basic operations like create a repository(a repository is simply a folder) on GitHub, pull a working repository down to your local machine, change your local version of code and push it back to GitHub, and other basic notions, all will be captured in these **first** 2 videos.

- *Git and GitHub for Beginners - Crash Course:*

https://www.youtube.com/watch?v=RG0j5yH7evk&t=2401s&ab_channel=freeCodeCamp.org

- *Git Tutorial for Beginners - Git & GitHub Fundamentals In Depth:*

https://www.youtube.com/watch?v=DVRQoVRzMIY&ab_channel=TechWithTim

- *Git and GitHub (optional, if you interested in investigating more)*

<https://youtube.com/playlist?list=PLhW3qG5bs-L8OIIcBNX9u4MZ3rAt5c5GG>

1.4. Git commands:

- Below are list of Git commands that you need to know for now.

SETUP

Configuring user information used across all local repositories

git config --global user.name "[firstname lastname]"
set a name that is identifiable for credit when review version history
git config --global user.email "[valid-email]"
set an email address that will be associated with each history marker
git config --global color.ui auto
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

git init
initialize an existing directory as a Git repository
git clone [url]
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

git status
show modified files in working directory, staged for your next commit
git add [file]
add a file as it looks now to your next commit (stage)
git reset [file]
unstage a file while retaining the changes in working directory
git diff
diff of what is changed but not staged
git diff --staged
diff of what is staged but not yet committed
git commit -m "[descriptive message]"
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

git branch
list your branches. a * will appear next to the currently active branch
git branch [branch-name]
create a new branch at the current commit
git checkout
switch to another branch and check it out into your working directory
git merge [branch]
merge the specified branch's history into the current one
git log
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

git log
show the commit history for the currently active branch
git log branchB..branchA
show the commits on branchA that are not on branchB
git log --follow [file]
show the commits that changed file, even across renames
git diff branchB...branchA
show the diff of what is in branchA that is not in branchB
git show [SHA]
show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

git rm [file]
delete the file from project and stage the removal for commit
git mv [existing-path] [new-path]
change an existing file path and stage the move
git log --stat -M
show all commit logs with indication of any paths that moved

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

git remote add [alias] [url]
add a git URL as an alias
git fetch [alias]
fetch down all the branches from that Git remote
git merge [alias]/[branch]
merge a remote branch into your current branch to bring it up to date
git push [alias] [branch]
Transmit local branch commits to the remote repository branch
git pull
fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

git rebase [branch]
apply any commits of current branch ahead of specified one
git reset --hard [commit]
clear staging area, rewrite working tree from specified commit

2. GitLab

2.1. Gitlab introduction

- The difference between Git and GitLab: People also sometimes confuse **Git** and **GitHub**.

+ **Git** is the tool that tracks the changes in your code over time.

+ **GitLab** is a website where you host all of your **Git repositories**. A lot of people seem to think **Git** and **GitLab** are one, that they're the exact same thing. That is not true, these are actually different things, they obviously integrate very well with each other and they're very similar and what they actually do but they are two separate things and it's important to understand the difference.

+ **GitLab** is really just a website, think of it as kind of a nice add-on or extension to **Git** and what **GitLab** does is it gives us a nice UI and some features that **Git** doesn't have and **GitLab** lets us host remote repositories and work collaboratively with a lot of other people

+ What Git does is it actually what handles all the version control stuff so when you're doing something like merging code together or when you're doing something like making a commit or making a change or pulling or push, you're using Git.

+ Git is the version control version control software, GitLab is kind of just like a layer on top of Git that gives us a bunch of commands, it gives us a bunch of UI features, it has a fancy website and it has a bunch of other things that Git just doesn't have. So these are two separate things, please do not confuse the fact that they are the same.

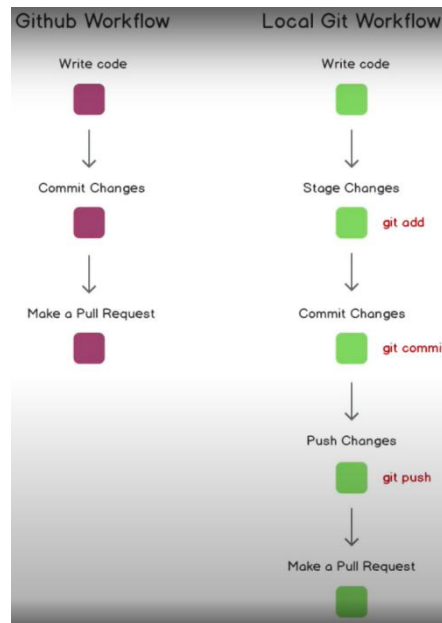
+ We can actually use Git without even having to touch GitLab because GitLab is not required, **it's not a dependency** of Git. GitLab is just something that we use on top of Git and there's lots of other tools that go on top of Git as well.

2.2. Practice GitLab:

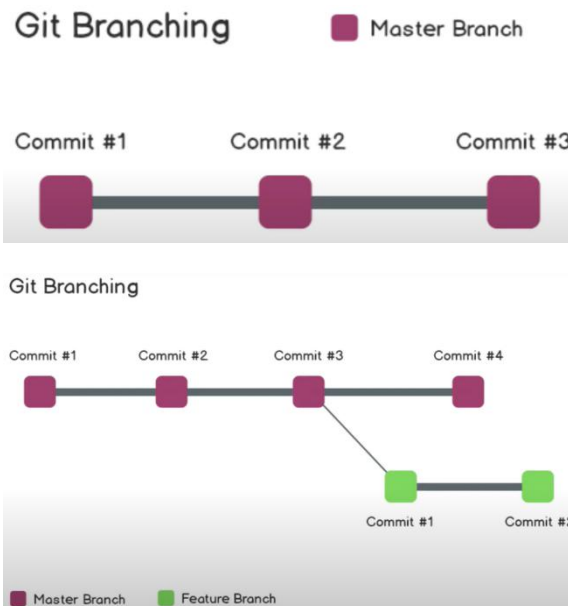
- Please create a GitLab account, then login to your GitLab to practice some basic operations in **the first 2 videos**(even though those 2 videos practice on GitHub, GitLab practice on basic operations would be the same):
- Create a blank project(repository) in GitLab.
- Generate a SSH key pair for authenticating your local device to the created project. The SSH key pair comprise a public key and a private key(you keep this key and not share it to anyone). is the one that you have to keep secure on your local machine, you don't share this key with anybody. How it works is that the public key you put on GitHub, and then every time you want to connect to GitHub or push your code on GitHub or use your account via your local machine, you use your private key to show GitHub, that you are the one that generated this public key. It's a mathematical proof that only this private key could have generated this public key.
- Practice the following Git commands yourself on the created project:
 - + Git add
 - + Git config
 - + Git commit
 - + Git push
 - + Git pull
 - + Git forking
 - + Git clone
 - + Git merge
 - + Git checkout
 - + Git diff.

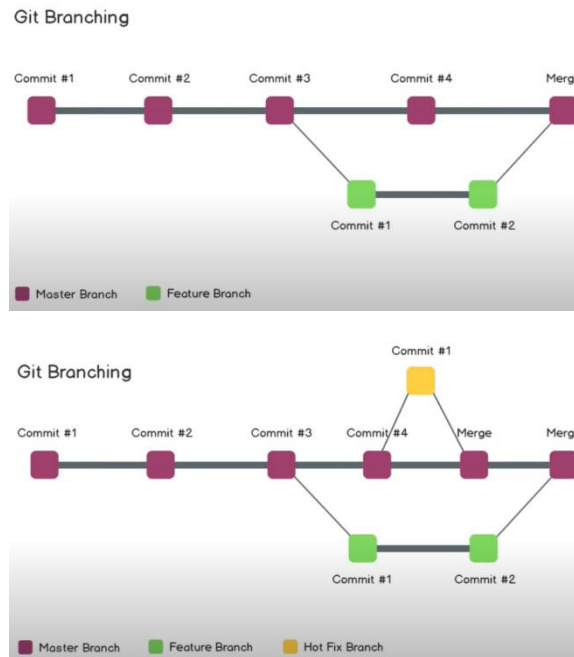
2.3. Git and GitHub Basic Work Flow

*** GitHub Work Flow also applies to GitLab



2.4. Branching





3. GitLab CI/CD

3.1. GitLab CI/CD introduction:

- When you push your code changes to a remote repository(shared by many programmers) on your personal branch that you are working on, you have to make sure that your code changes is not buggy in order to merge your code into the main branch. You do so by manually run a test suite against your code, and your code passes the test suite, then you will send a merge request and wait for the merge request to be accepted to merge your code changes into the main branch. This creates overheads since programmers make changes every day.

- You can actually automate this process with GitLab CI. CI stands for Continuous Integration, it simply means that you continuously integrate your work(that is your code changes) to the shared main code as often as possible (even though you might haven't done so much). If you have made some progress(even small), and if your work meets all the metrics, passes all test cases, then integrate it into the main code. Instead of building the source code, running all test cases and waiting for the result and sending a merge request to integrate your work into the main work, use GitLab CI/CD. It automates this repetitive process for you so you don't have to pay attention to this process anymore. GitLab CI/CD do all the tasks I've mentioned above every time you push your work to GitLab.

- GitLab CI/CD is a tool for software development using the continuous methodologies. Use GitLab CI/CD to catch bugs and errors early in the development cycle. Ensure that all the code deployed to production complies with the code standards you established for your app.

- GitLab CI/CD can automatically build, test, deploy, and monitor your applications by using Auto DevOps.

- With the continuous method of software development, you continuously build, test, and deploy iterative code changes. This iterative process helps reduce the chance that you develop new code based on buggy or failed previous versions. With this method, you strive to have less human intervention or even no intervention at all, from the development of new code until its deployment.

3.2. Continuous Integration (CI):

- Consider an application that has its code stored in a Git repository in GitLab. Developers push code changes every day, multiple times a day. For every push to the repository, you can create a set of scripts to build and test your application automatically. These scripts help decrease the chances that you introduce errors in your application.

- This practice is known as Continuous Integration. Each change submitted to an application, even to development branches, is built and tested automatically and continuously. These tests ensure the changes pass all tests, guidelines, and code compliance standards you established for your application. This helps to avoid integration challenges & check that the application is not broken whenever new commits are done.

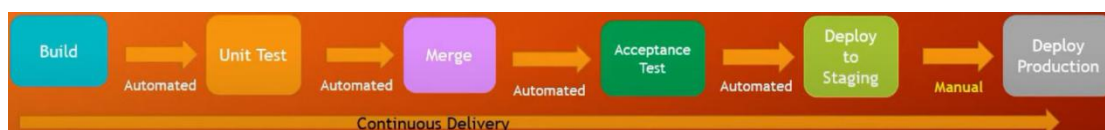
- GitLab itself is an example of a project that uses Continuous Integration as a software development method. For every push to the project, a set of checks run against the code.



3.3. Continuous Delivery (CD)

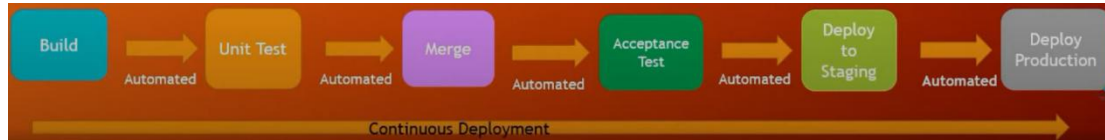
- Continuous Delivery is a step beyond Continuous Integration. Not only is your application built and tested each time a code change is pushed to the codebase, the application is also deployed continuously. However, with continuous delivery, you trigger the deployments manually.

- Continuous Delivery checks the code automatically, but it requires human intervention to manually and strategically trigger the deployment of the changes.



3.4. Continuous Deployment (CD)

- Continuous Deployment is another step beyond Continuous Integration, similar to Continuous Delivery. The difference is that instead of deploying your application manually, you set it to be deployed automatically. Human intervention is not required.



- GitLab CI/CD is the part of GitLab that you use for all of the continuous methods (Continuous Integration, Delivery, and Deployment). With GitLab CI/CD, you can test, build, and publish your software with no third-party application or integration needed.

3.5. GitLab CI/CD workflow

- GitLab CI/CD fits in a common development workflow. You can start by discussing a code implementation in an issue and working locally on your proposed changes. Then you can push your commits to a feature branch in a remote repository that's hosted in GitLab. The push triggers the CI/CD pipeline for your project. Then, GitLab CI/CD:

- + Runs automated scripts (sequentially or in parallel) to:

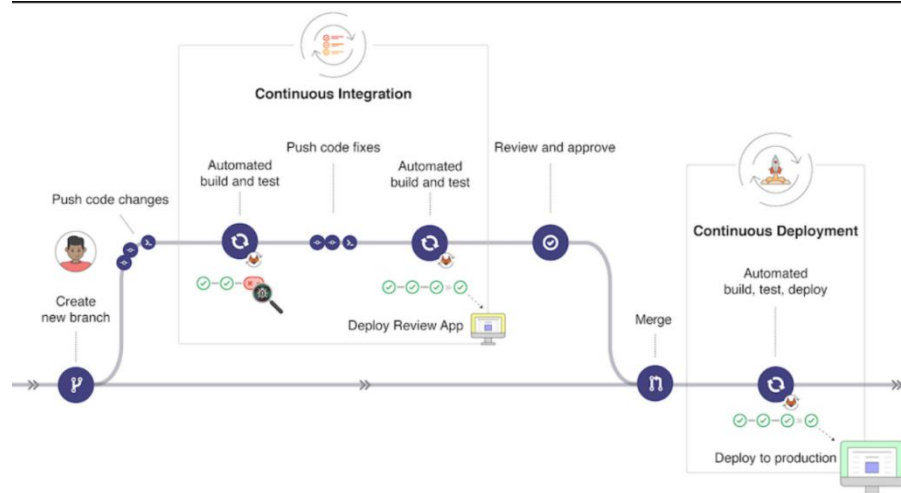
- + Build and test your application.

- Preview the changes in a Review App, the same as you would see on your localhost. After the implementation works as expected:

- + Get your code reviewed and approved.

- + Merge the feature branch into the default branch.

- GitLab CI/CD deploys your changes automatically to a production environment. If something goes wrong, you can roll back your changes.



3.6. GitLab CI/CD Video series:

- I highly recommend you to finished the videos in these links below before moving to the next steps. The purpose if to have a overview feel a about GitLab CI/CD.

- GitLab CI CD (you need to watch the first 6 videos only, the rest videos are optional):

<https://youtube.com/playlist?list=PLVx1qovxj-am2q9M8mC2CmMUfrxZGoZ9J>

- GitLab Beginner Tutorial:

https://youtube.com/playlist?list=PLhW3qG5bs-L8YSnCiyQ-jD8XfHC2W1NL_

- GitLab CI/CD Pipeline | GitLab CI/CD Tutorial | Gitlab Tutorial | DevOps Training | Edureka:

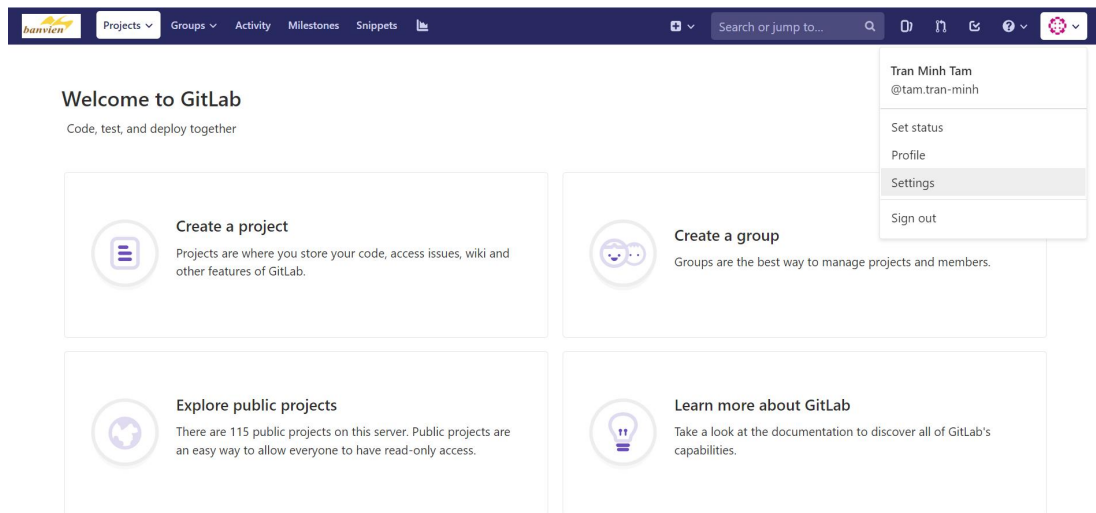
https://www.youtube.com/watch?v=HSV-Kky9N5E&t=1s&ab_channel=edureka%21

- Continuous Integration with GitLab CI:

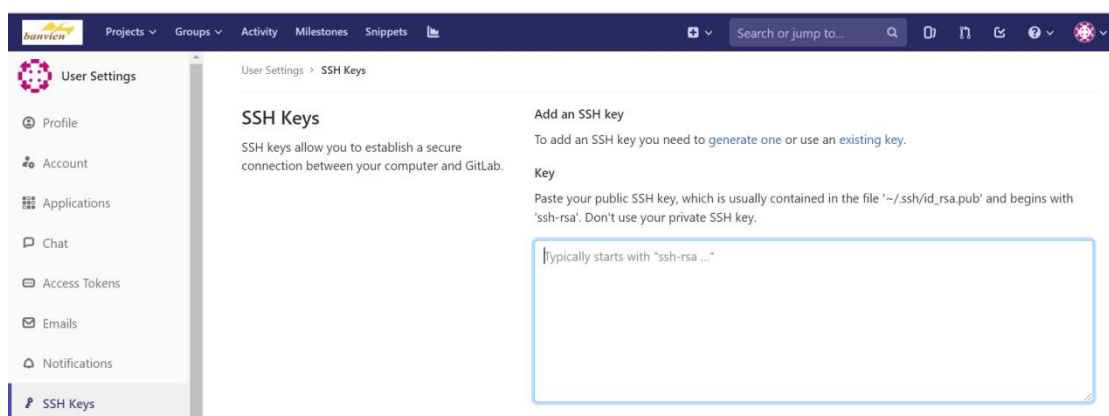
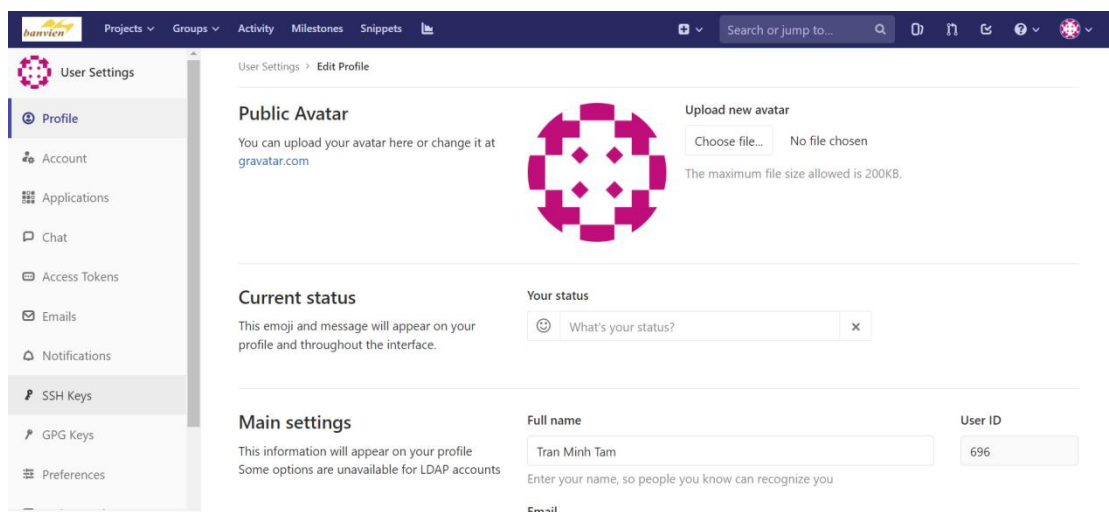
https://www.youtube.com/watch?v=EuwLdbCu3DE&ab_channel=NERDSummit(this video optional, watch the first 20 minutes only, the rest of the video is optional).

3.7. What is SSH

1. SSH is also known as Secure Shell
2. SSH or Secure Shell is a network communication protocol that enables two computers to communicate
3. SSH provide a secure way to access a computer over an unsecured network
4. Used for authentication
5. By setting ssh key you can connect to GitLab server without using username and password each
6. Steps to create a SSH key pair:
 - Login to your GitLab account, then click on your account icon on the top right of the screen and select Setting



- On the left side bar, click the SSH Keys button.



- Generate a SSH key pair, open Command Prompt and type the following command

```
cmd Command Prompt - ssh-keygen -t rsa -b 4096 -C "first time create a SSH key pair"
```

```
Microsoft Windows [Version 10.0.19043.1415]  
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Users\tam.tran-minh>ssh-keygen -t rsa -b 4096 -C "first time create a SSH key pair"
```

-t: this option specifies type of encryption

-b: this option specifies strength of encryption

-C: add message option

```
C:\Users\tam.tran-minh>ssh-keygen -t rsa -b 4096 -C "first time create a SSH key pair"  
Generating public/private rsa key pair.  
Enter file in which to save the key (C:\Users\tam.tran-minh/.ssh/id_rsa): _
```

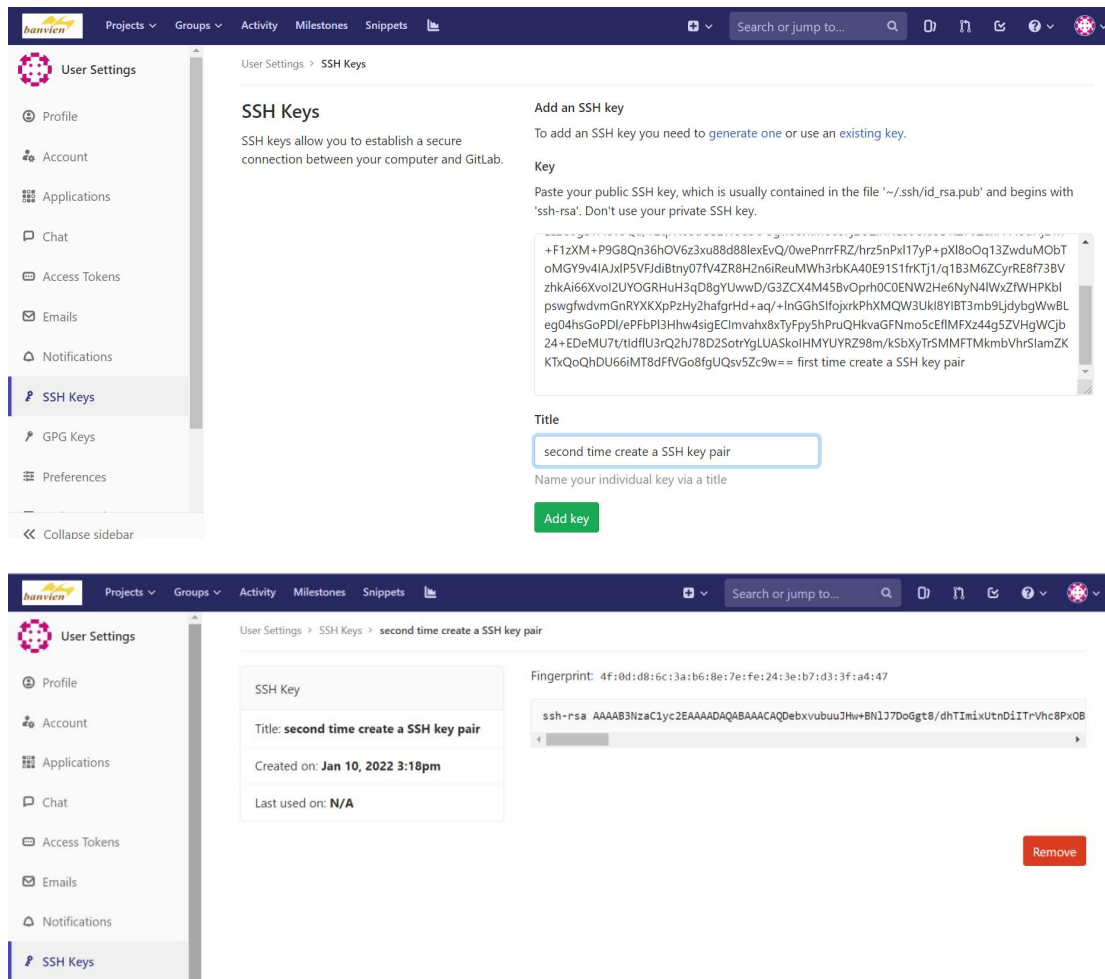
- Enter the file name to save the key pair(you can provide name for the key pair or accept the default name).

```
C:\Users\tam.tran-minh>ssh-keygen -t rsa -b 4096 -C "first time create a SSH key pair"  
Generating public/private rsa key pair.  
Enter file in which to save the key (C:\Users\tam.tran-minh/.ssh/id_rsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:
```

- When the passphrase prompt appears, just type **Enter** twice.

```
C:\Users\tam.tran-minh>ssh-keygen -t rsa -b 4096 -C "first time create a SSH key pair"  
Generating public/private rsa key pair.  
Enter file in which to save the key (C:\Users\tam.tran-minh/.ssh/id_rsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in C:\Users\tam.tran-minh/.ssh/id_rsa.  
Your public key has been saved in C:\Users\tam.tran-minh/.ssh/id_rsa.pub.  
The key fingerprint is:  
SHA256:fsPQZF50BYjR/zxv70R+KFarlP8WOnaJYGTwx3ZwjLU first time create a SSH key pair  
The key's randomart image is:  
+---[RSA 4096]---+  
| .+ ...=. |  
| ..o + o. |  
| =.+ oE |  
| = B.+ . |  
| S = +o. |  
| . o o .- |  
| . = +.O*+ |  
| . oo*o=* |  
| .oo+** |  
+---[SHA256]---+  
C:\Users\tam.tran-minh>_
```

- Navigate to the .ssh folder containing the generated key pair. Open the *.pub file(pub stands for public) by any text editor, copy the content of the file, paste it to the SSH public key field in GitLab, provide the public key title and click Add Key button



- Now, remember the another private key? It's a mathematical proof that only this private key could have generated the public key(the one that you have provided GitLab). Now, when you push your code in your local machine to GitLab, how can GitLab know that the one pushing is really you? The answer is your corresponding private key.

- You are the one who create the SSH key pair, and login to GitLab to provide the generated public key, now you need to provide Command Prompt the private key, GitLab will use this private key to authenticate access right to your local machine, I hope it makes sense.

- It means that, you can authenticate access to any personal computers(local computers) in this manner.

- Adding your SSH key to the ssh-agent. Before adding a new SSH key to the ssh-agent to manage your keys, you should have checked for existing SSH keys and generated a new SSH key. Ensure the ssh-agent is running. Open Git Bash and type the following command

eval "\$(ssh-agent -s)"

```
tam.tran-minh@L-226 MINGW64 ~/.ssh  
$ eval "$(ssh-agent -s)"  
Agent pid 1044
```

- Add your SSH private key to the ssh-agent.

ssh-add ~/.ssh/id_rsa

```
tam.tran-minh@L-226 MINGW64 ~/.ssh  
$ ssh-add ~/.ssh/id_rsa  
Identity added: /c/Users/tam.tran-minh/.ssh/id_rsa (first time create a SSH key pair)
```

- Now you should be able to push to your private GitLab project.

3.8. GitLab Runner

1. What is GitLab Runner:

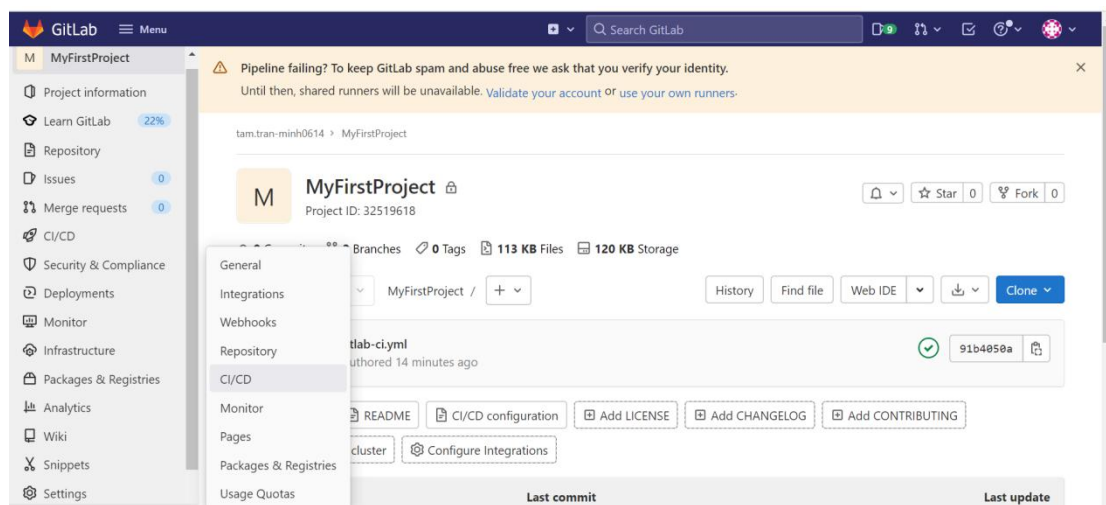
- **GitLab Runner** is an **application** that works with GitLab CI/CD. Runners are processes(what is a process?) that pick up and execute CI/CD jobs for GitLab and send results back to GitLab.

2. Specific runners, Shared Runner

- There are 2 types of **Runners: Specific runners, Shared Runner.**
- **Specific runners:** Remember that runners are processes that pick up jobs and execute jobs. Those runners could be processes on your own local machine(it means that your local machine is the one that creates processes for executing jobs, your local machine is the one who does CI/CD jobs, I hope you get what I mean)
- **Shared Runner:** To put in short, it is the Google Cloud Platform that creates processes picking up jobs and executing those jobs for you. That's it, in this case, it is Google Cloud Platform that does CI/CD jobs for you.

3. Registering Runners

- On the left side bar of your project, click **CI/CD** in **Settings** like the image below.



- Expand the **Runners** section like the following

Runners

[Collapse](#)

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

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

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

Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

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

<https://gitlab.com/>

And this registration token:

wveD4QXD1wFKxs9j5yVvk

[Reset registration token](#)

[Show Runner installation instructions](#)

Shared runners

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in **autoscale mode** and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

User validation required

Shared runners cannot be enabled until a valid credit card is on file.

- To register a runner under Windows, run the following command:

```
Command Prompt

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

- The following prompt will show up

```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
```

- In the **Runners** expand section, copy the **URL field**, and paste it to the Command Prompt

Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

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

<https://gitlab.com/>

And this registration token:

wveD4QXD1wFKxs9j5yVvk

[Reset registration token](#)

[Show Runner installation instructions](#)

Available specific runners

#13139471 (YC2y_msG)

My first Runner on Banvien internal

[Remove runner](#)

Shared runners

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in **autoscale mode** and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

Enable shared runners for this project

☐

Available shared runners: 42

#1506020 (Hs8mheX5)

windows-shared-runners-manager-1

[shared-windows](#) [windows](#) [windows-1809](#)

#11573930 (KzYhZxBv)

1-blue.shared-gitlab-org.runners-manager.gitlab.com

[gitlab-org](#)


```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
```

- The following prompt will show up

```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
```

- In the **Runners** expand section, copy the **registration token** field, and paste it to the Command Prompt

Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
<https://gitlab.com/>

And this registration token:
[wveD4QXD1wFKxs9j5yVk](#)

[Reset registration token](#)

[Show Runner installation instructions](#)

Available specific runners

[#13139471 \(YC2y_msG\)](#) [Remove runner](#)

My first Runner on Banvien internal

Shared runners

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in **autoscale mode** and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

Enable shared runners for this project

☒

Available shared runners: 42

[#1506020 \(Hs8mheX5\)](#) [Remove runner](#)

windows-shared-runners-manager-1

[shared-windows](#) [windows](#) [windows-1809](#)

[#11573930 \(KzYhZxBv\)](#) [Remove runner](#)

1-blue.shared-gitlab-org.runners-manager.gitlab.com

[gitlab-org](#)

```
Select Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wFKxs9j5yVk
```

- The following prompt will show up

```
Select Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wFKxs9j5yVk
Enter a description for the runner:
[L-226]:
```

- Then, enter a description for the runner

```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wfkxs9j5yVk
Enter a description for the runner:
[L-226]: My second Runner on Banvien internal
```

- The following prompt will show up

```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wfkxs9j5yVk
Enter a description for the runner:
[L-226]: My second Runner on Banvien internal
Enter tags for the runner (comma-separated):
```

- This Runner will run no tags jobs, so just type Enter. Then, the following prompt will show up.

```
Command Prompt - gitlab-runner.exe register

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wfkxs9j5yVk
Enter a description for the runner:
[L-226]: My second Runner on Banvien internal
Enter tags for the runner (comma-separated):

Registering runner... succeeded                      runner=wveD4QXD
Enter an executor: custom, docker-windows, virtualbox, kubernetes, docker+machine, docker-ssh+machine, docker, docker-ssh, parallels, shell, ssh:
```

- Enter **shell** executor. Then the Command Prompt informs that you have successfully registered a runner.

```
Command Prompt

C:\GitLab-Runner>gitlab-runner.exe register
Runtime platform                                arch=amd64 os=windows pid=12776 revision=5316d4ac version=14.6.0
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
wveD4QXD1wfkxs9j5yVk
Enter a description for the runner:
[L-226]: My second Runner on Banvien internal
Enter tags for the runner (comma-separated):

Registering runner... succeeded                      runner=wveD4QXD
Enter an executor: custom, docker-windows, virtualbox, kubernetes, docker+machine, docker-ssh+machine, docker, docker-ssh, parallels, shell, ssh:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!

C:\GitLab-Runner>
```

- Then, Reload the **Runners expand** page, now you should have a registered runner, but it is still not available to run CI/CD jobs.

These runners are specific to this project.

Set up a specific Runner for a project

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

And this registration token:
`wveD4QXD1wFKxs9j5yVvk`

Reset registration token

Show Runner installation instructions

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in *autoscale mode* and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

User validation required

Shared runners cannot be enabled until a valid credit card is on file.

Validate account

Available specific runners

#13141573 (zeaKZkXA)

Remove runner

My second Runner on Banvien internal

#13139471 (YC2y_msG)

Remove runner

My first Runner on Banvien internal

Available shared runners: 42

#1506020 (Hs8mheX5)

windows-shared-runners-manager-1

shared-windows windows windows-1809

#12270845 (JLqUopmM)

- The warning triangle icon means that you haven't start that runner on the **GitLab Runner** application on your local machine.

- The Green circle icon means that you have successfully *registered* and *started* that **Runner** on **GitLab Runner** application on your local machine.

- The next steps to do are to *Install GitLab Runner* application on your local machine and *start* GitLab Runner.

4. Install GitLab Runner

- go to this link <https://docs.gitlab.com/runner/install/index.html>. In the **Binaries** section, choose the install option which is compatible with your machine. For the purpose of demonstration, I will use Windows.

Binaries

- [Install on GNU/Linux](#)
- [Install on macOS](#)
- [Install on Windows](#)
- [Install on FreeBSD](#)
- [Install nightly builds](#)

- Create a folder somewhere in your system, **C:\GitLab-Runner**.

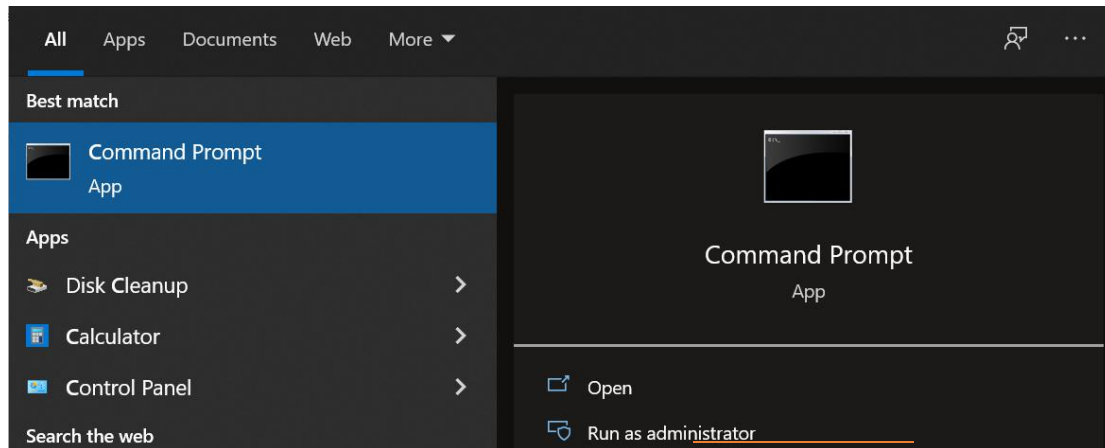
- Download the binary for 64-bit or 32-bit and put it into the folder you created. Then renamed the binary to **gitlab-runner.exe** like below.

gitlab-runner.exe	1/10/2022 11:28 PM	Application	55,032 KB
-------------------	--------------------	-------------	-----------

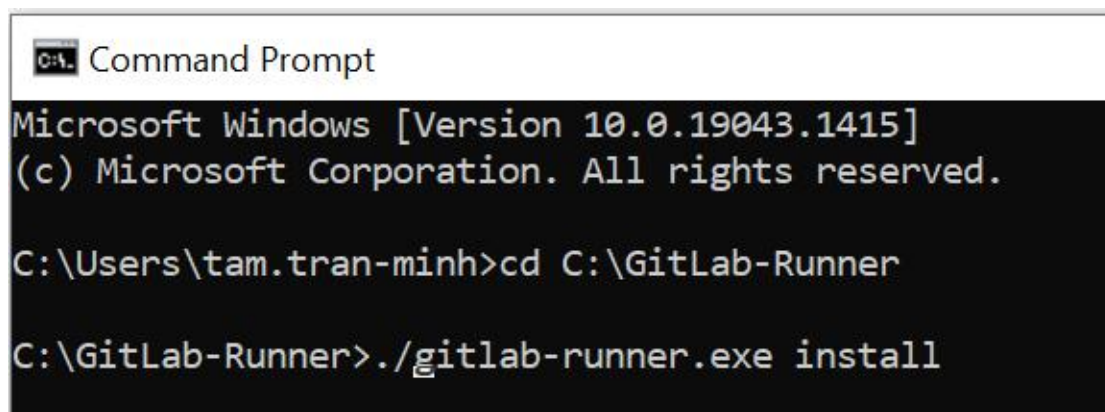
* 64-bit (<https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe>)

* 32-bit (<https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-386.exe>)

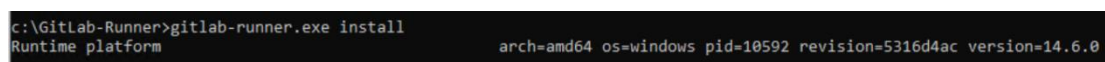
- Run an elevated command prompt(If you are on Ban Vien laptop, you will need to contact IT men for stalling GitLab-Runner).



- Navigate to the folder containing the download file and install **GitLab Runner** by the following command: `./gitlab-runner.exe install`



- If the following prompt show up, you have successfully **Install** GitLab Runner.



5. Start Runners

- Now you've installed GitLab Runner, and you've registered a runner, right? You will start this Runner by: `./gitlab-runner.exe start`

- This command starts those runners which have been registered, but haven't started yet. (This command requires to be run as administrator, so you might have to contact IT men)

```
c:\GitLab-Runner>gitlab-runner.exe start
Runtime platform arch=amd64 os=windows pid=11128 revision=5316d4ac version=14.6.0
```

- Then, reload the **Runners expand** page. The *warning triangle* icon will become the green circle one (meaning that the runner is available to run jobs)

6. Update Runners

- Suppose that you have **installed** and start **GitLab Runner**. Then you register a few more runners, these runner will not be available to execute job(because you haven't started them yet, you just have registered them).
- To update Runners, simply *stop* the **GitLab Runner application** and *restart*

```
.\gitlab-runner.exe stop
```

```
.\gitlab-runner.exe start
```

3.9. .gitlab-ci.yml file

1. .gitlab-ci.yml introduction:

- Remember the repetitive process of building source code, running a test suite on the built source code for verification, then contributing your work by initiating a merge request and waiting for the merge request to be accepted by the owners of the projects.
- With GitLab CI/CD, that whole process is automated, free you from doing those repetitive tasks, reduce the time the project owners spend on reviewing your work for validation before integrate your work into the main work.
- The details of how to automate the process is defined in **.gitlab-ci.yml** file. In this file, you define jobs need to be done, a job comprise of scripts of commands. The process has stages, those stages are defined in this file as well as dependencies between jobs. A Stage comprise many jobs.
- Jobs can run sequentially or in parallel.
- To use GitLab CI/CD, you need:
 - Application code hosted in a Git repository.
 - A file called **.gitlab-ci.yml** in the root of your repository, which contains the CI/CD configuration.

- In the `.gitlab-ci.yml` file, you can define:

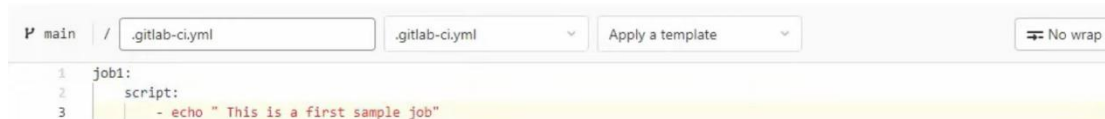
- The scripts you want to run.
- Other configuration files and templates you want to include.
- Dependencies and caches.
- The commands you want to run in sequence and those you want to run in parallel.
- The location to deploy your application to.
- Whether you want to run the scripts automatically or trigger any of them manually.

- The scripts are grouped into **jobs**, and jobs run as part of a larger **pipeline**. You can group multiple independent jobs into **stages** that run in a defined order. The CI/CD configuration needs at least one job that is **not hidden**.

- You should organize your jobs in a sequence that suits your application and is in accordance with the tests you wish to perform. To **visualize** the process, imagine the scripts you add to jobs are the same as CLI commands you run on your computer.

- When you add a `.gitlab-ci.yml` file to your repository, GitLab detects it and an application called **GitLab Runner** runs the scripts defined in the jobs.

- A simple `.gitlab-ci.yml` file

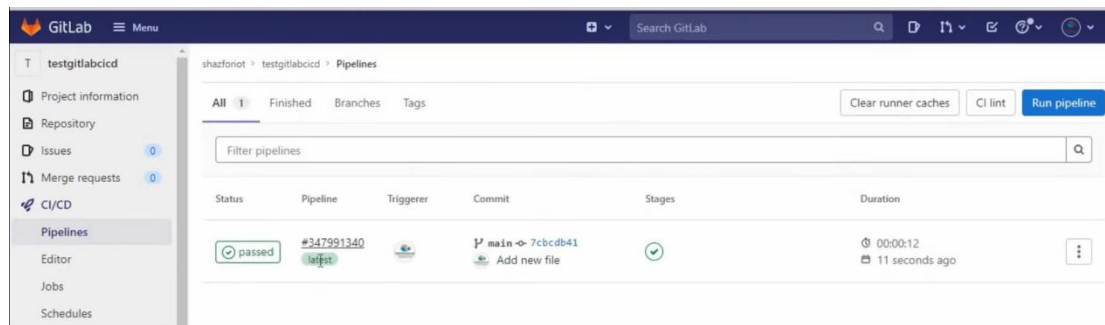


```
1 job1:
2   script:
3     - echo "This is a first sample job"
```

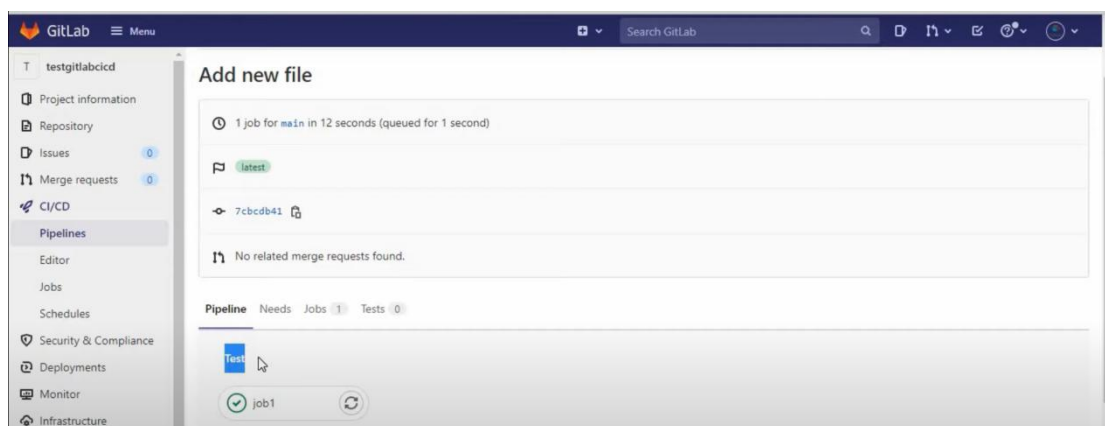
- This file contains 1 job, the job's name is "job1". This job simply print a "This is a first sample job" to terminal.

- If you save this file and "commit" to GitLab. GitLab CI/CD will automatically find this **.gitlab-ci.yml** and execute it. Remember to save this file in the project's root folder.

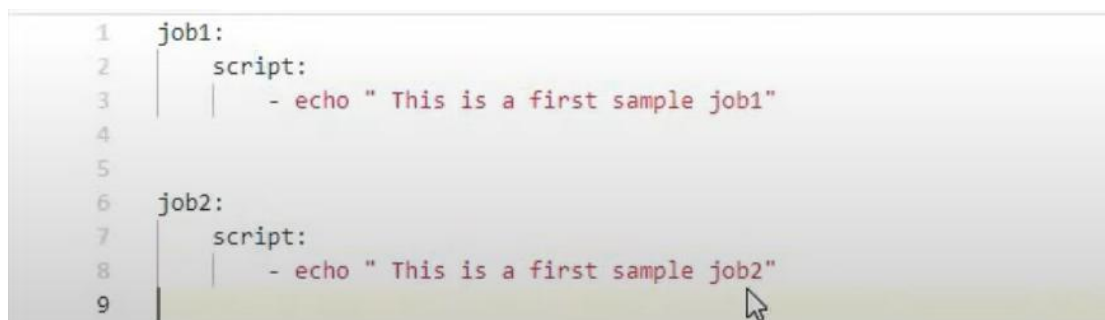
- When you commit this file to GitLab, in the **Pipelines** section will appear a running pipeline along with some information about that pipeline. Like in the **Duration** column, you can see how long the pipeline has run, when the pipeline began to run.



- When you click on the **ID** of this pipeline, you can view detail information about the pipeline. All jobs in **.gitlab-ci.yml** file and their name. How jobs are organized, what are the stages, how jobs are grouped into stages, what are the dependencies between jobs, state of the pipeline.

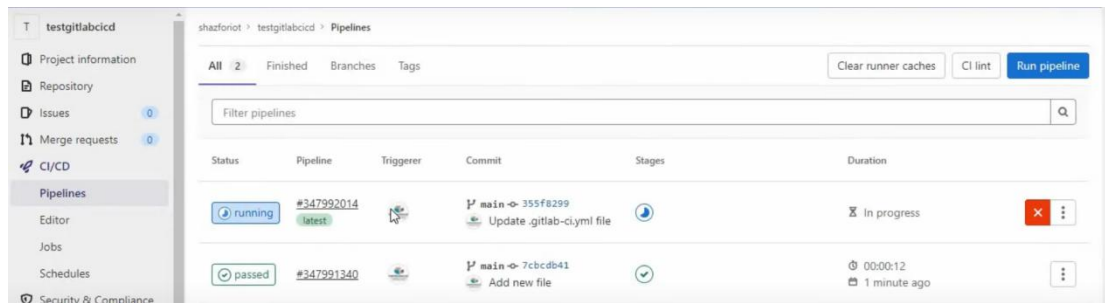


- Example 2: Update the content of the above **.gitlab-ci.yml** file with this content.

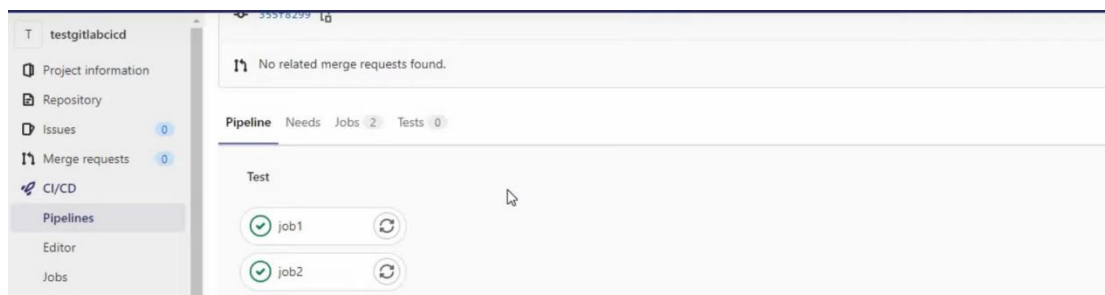


- In this file now, we have 2 jobs named *job1*, *job2*. Each job print a string to the terminal and ... that's it. Please notice that each job will be executed by one *process*. (you can google to differentiate a process and a program).

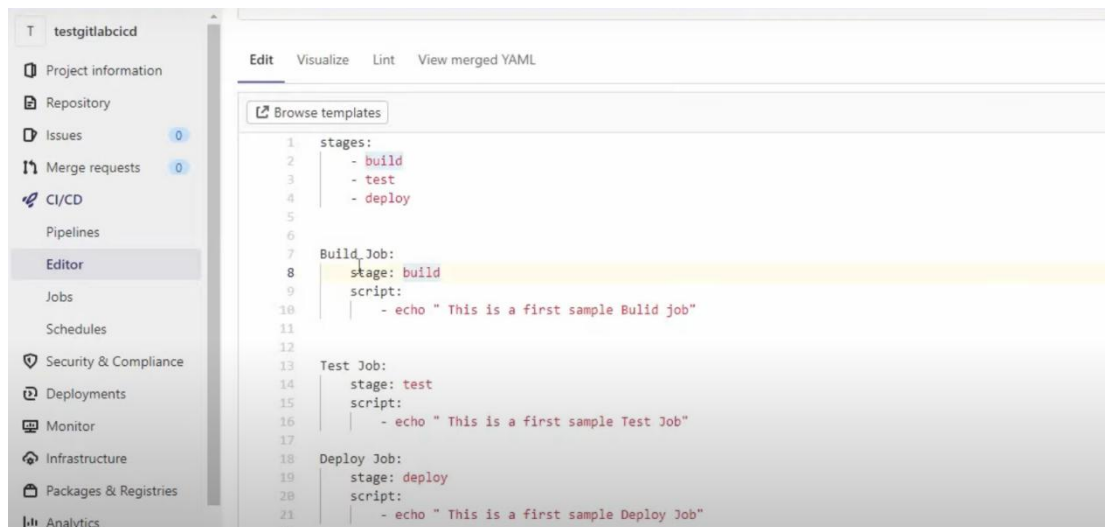
- Then Commit the updated file. After you commit, in the pipeline section will appear a second pipeline



- Click on the new pipeline to view its detail information, the default stage of jobs is *Test* (if you don't specify the stage for that job).



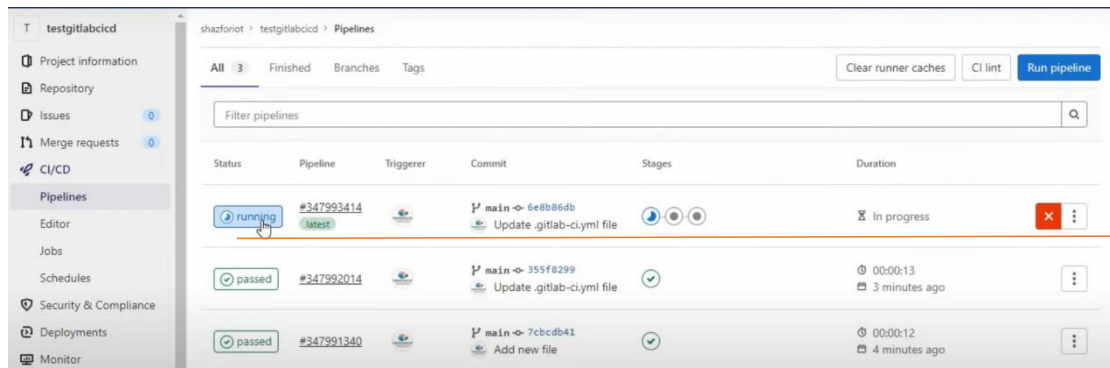
- Example 3: Update the content of the same **.gitlab-ci.yml** file with this content.



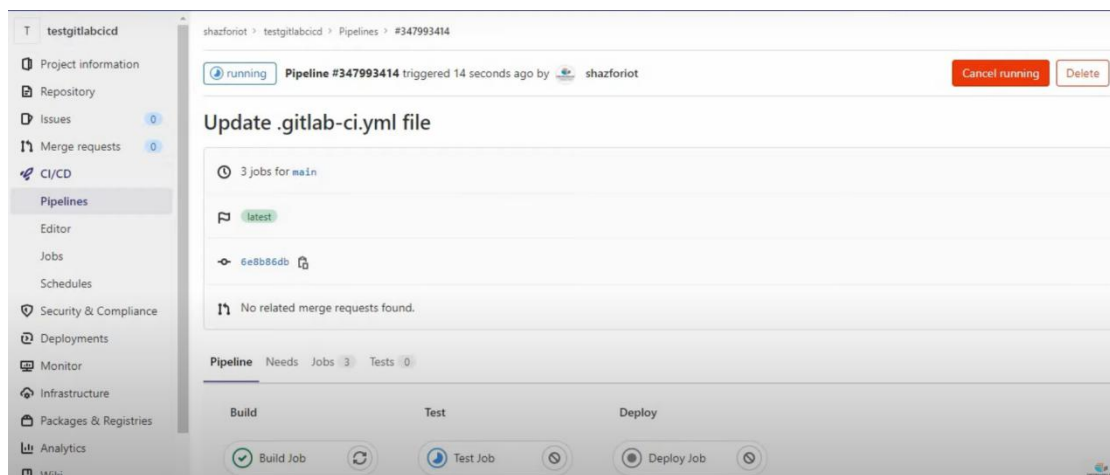
- In this new file, the keyword *stages* is used to declare stages in the pipeline, the order of states declared matters.

- In the image above, we have 3 stages: *build*, *test*, *deploy* and 3 jobs: *Build Job*, *Test Job*, *Deploy Job*. Build Job's stage is build, Test Job's stage is test, Deploy Job's stage is deploy.

- Commit this file again, in the **pipelines** section will appear a new pipeline.



- The detail of this pipeline:



- A job which has a **green** mark means that that job has successfully finished.
- A job which has a **blue** means that that job is running.
- A job which has a **grey** mark means that that job hasn't started.
- Even though each job just simply echo a string to terminal(which should take less than 1ms to run), it still takes seconds to run each job. Because each job is run by a process, the first seconds is for preparing a environment to run a job, that's why.

2. Pipeline

- Pipelines are the top-level component of continuous integration, delivery, and deployment.
- Pipelines comprise:
 - Jobs, which define what to do. For example, jobs that compile or test code.
 - Stages, which define when to run the jobs. For example, stages that run tests after stages that compile the code.

- Jobs are executed by **runners**. Multiple jobs in the same stage are executed in parallel, if there are enough concurrent runners.
- If all jobs in a stage succeed, the pipeline moves on to the next stage.
- If any job in a stage fails, the next stage is not (usually) executed and the pipeline ends early.
- In general, pipelines are executed automatically and require no intervention once created. However, there are also times when you can manually interact with a pipeline.
- A typical pipeline might consist of four stages, executed in the following order:
 - A build stage, with a job called **compile**.
 - A test stage, with two jobs called **test1** and **test2**.
 - A staging stage, with a job called **deploy-to-stage**.
 - A production stage, with a job called **deploy-to-prod**.

3. Tag

- A job can have no tag or many tags. A Runner(process) can have no tag or many tags
- When you register a runner, the Command Prompt will ask you to provide **tags** for that runner. Basically, To a runner's perspective, the tags tell the runner which jobs to pick and execute. In a job perspective, the tags tell the job which runners can pick it up and execute it.
- Imagine the following scenario:
 - + You have 3 chiefs(3 runners) A, B C.
 - + Chief A's tags: *banana fried* (Chief A makes banana fried).
 - + Chief B's tags: *banana cake, banana salad* (Chief B makes banana cake, banana salad)
 - + Chief C's tags: *banana fried, banana cake* (Chief C makes banana fried, banana cake)
 - + A batch bananas(jobs). Each banana has tags on it.
 - + Bananas which have tags: *banana fried*; can be picked up and processed by Chief A or Chief C.
 - + Bananas which have tags: *banana salad*; can be picked up and processed by Chief B.

- + Bananas which have tags: *banana fried*, *banana salad*; can be picked up and processed by either Chief A or Chief B.
- + Bananas which have tags: *banana fried*, *banana cake*; can be picked up and processed by Chief A or Chief B or Chief C.
- No tags Jobs can be picked up and executed only by no tags Runners (I hope it makes sense).

4. Schedule (optional)

- Pipelines are normally run based on certain conditions being met. For example, when a branch is pushed to repository.
- Pipeline schedules can be used to also run [pipelines](#) at specific intervals. For example:
 - Every month on the 22nd (cron example: `0 0 22 * *`) for a certain branch.
 - Every month on the 2nd Monday (cron example: `0 0 * * 1#2`).
 - Every other Sunday at 0900 hours (cron example: `0 9 * * sun%/2`).
 - Once every day (cron example: `0 0 * * *`).
- Schedule timing is configured with cron notation, parsed by [Fugit](#)
- In order for a scheduled pipeline to be created successfully:
 - The schedule owner must have [permissions](#) to merge into the target branch.
 - The pipeline configuration must be valid.
- Otherwise the pipeline is not created

4. Simple Demo GITLAB CI/CD

- Open your GitLab project, create a **.gitlab-ci.yml** file(or update your **.gitlab-ci.yml** file if it already exists) with the following content:

```

1  stages:
2    - build
3    - test
4    - deploy
5
6  build-job:
7    stage: build
8    script:
9      - echo "Hello, $GITLAB_USER_LOGIN!"
10
11 test-job1:
12   stage: test
13   script:
14     - echo "This job tests something"
15
16 test-job2:
17   stage: test
18   script:
19     - echo "This job tests something, but takes more time than test-job1."
20     - echo "After the echo commands complete, it runs the sleep command for 20 seconds"
21     - echo "which simulates a test that runs 20 seconds longer than test-job1"
22     - sleep 20
23
24 deploy-prod:
25   stage: deploy
26   script:
27     - echo "This job deploys something from the $CI_COMMIT_BRANCH branch."

```

- Notice that the .yml file has to be located at the root directory.
- The .gitlab-ci.yml file has 4 jobs: *build-job*, *test-job1*, *test-job2*, *deploy-prod*; and 3 stages: *build*, *test*, *deploy*.
- The *build-job* simply echo a hello string. *\$GITLAB_USER_LOGIN* is a variable, which hold the user name.
- The *test-job1*, *test-job2* belong to *test stage* and can be run in **parallel**.
- Commit the .yml file. After you commit, a pipeline will be trigger

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
passed	#445331118	latest	main ~> 91b4050a Update .gitlab-ci.yml	✓✓✓	00:01:16 1 day ago

- The detail of that pipeline

The screenshot shows the GitLab CI/CD interface for a project named 'MyFirstProject'. The left sidebar contains navigation links: Project information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD, Pipelines (selected), Editor, Jobs, Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, and Release & Deployment. The main content area displays the pipeline 'Update .gitlab-ci.yml' which is 'passed'. It was triggered 1 day ago by Tam Tran. The pipeline summary shows 4 jobs for 'main' in 1 minute and 16 seconds. The 'latest' commit is '91b4050a'. Below the summary, the pipeline stages are listed: Build (with jobs 'build-job'), Test (with jobs 'test-job1' and 'test-job2'), and Deploy (with job 'deploy-prod'). All jobs are marked as successful with green checkmarks.

- Below are the running results of the 4 jobs, please spend time to inspect it. **Blue lines** represent tasks; **while lines** represent running information, **green lines** are commands. Each task (blue line) has a duration time on the right corner represent the execution time of that task.

- The detail running result of *build-job* job.

The screenshot shows the detailed view of the 'build-job' from the pipeline. The left sidebar is the same as the previous screenshot. The main content area shows the job 'build-job' triggered 1 day ago by Tam Tran. The job status is 'passed'. The job log is displayed in a dark-themed terminal window, showing the following steps and their durations:

- 1 Running with gitlab-runner 14.6.0 (5316d4ac) (00:00)
- 2 on My first Runner on Banvien internal YC2y_msG
- 3 Preparing the "shell" executor (00:00)
- 4 Using Shell executor...
- 5 Preparing environment (00:02)
- 6 Running on L-226...
- 7 Getting source from Git repository (00:12)
- 8 Fetching changes with git depth set to 50...
- 9 Initialized empty Git repository in C:/GitLab-Runner/builds/YC2y_msG/0/tam.tran-minh0614/MyFirstProject/.git/
- 10 Created fresh repository..
- 11 Checking out 91b4050a as main...
- 12 git-lfs/2.13.3 (GitHub; windows amd64; go 1.16.2; git a5e65851)
- 13 Skipping Git submodules setup
- 14 Executing "step_script" stage of the job script (00:01)
- 15 \$ echo "Hello, \$GITLAB_USER_LOGIN!"
- 16 Hello, tam.tran-minh!
- 17 Cleaning up project directory and file based variables (00:01)
- 18 Job succeeded

On the right side, the job details are summarized:

- Duration:** 19 seconds
- Finished:** 1 day ago
- Timeout:** 1h (from project)
- Runner:** #13139471 (YC2y_msG) My first Runner on Banvien internal
- Commit:** 91b4050a
- Update:** .gitlab-ci.yml
- Pipeline:** #445331118 for main
- Job:** build

- The detail running result of *test1-job* job.

The screenshot displays the GitLab CI/CD interface for a project named "MyFirstProject". The left sidebar shows the navigation menu with "Jobs" selected. The main area shows the details for the "test-job1" job, which was triggered 1 day ago by Tam Tran and has a status of "passed". The job log shows the following steps:

- Running with gitlab-runner 14.6.0 (5316d4ac)
- on My first Runner on Banvien internal YC2y_msG
- Preparing the "shell" executor (00:00)
- Using Shell executor...
- Preparing environment (00:01)
- Running on L-226...
- Getting source from Git repository (00:07)
- Fetching changes with git depth set to 50...
- Reinitialized existing Git repository in C:/GitLab-Runner/builds/YC2y_msG/0/tam.tran-mi-nh0614/MyFirstProject/.git/
- Checking out 91b4050a as main...
- git-lfs/2.13.3 (GitHub; windows amd64; go 1.16.2; git a5e65851)
- Skipping Git submodules setup
- Executing "step_script" stage of the job script (00:01)
- \$ echo "This job tests something"
- This job tests something
- Cleaning up project directory and file based variables (00:01)
- Job succeeded

The right sidebar shows the job summary for "test-job1":

- Duration: 12 seconds
- Finished: 1 day ago
- Timeout: 1h (from project)
- Runner: #13139471 (YC2y_msG) My first Runner on Banvien internal
- Commit 91b4050a
- Update .gitlab-ci.yml
- Pipeline #445331118 for main
- test
- test-job1
- test-job2

- The detail running result of *test2-job* job.

The screenshot displays the GitLab CI/CD interface for a project named "MyFirstProject". The left sidebar shows the navigation menu with "Jobs" selected. The main area shows the details for the "test-job2" job, which was triggered 1 day ago by Tam Tran and has a status of "passed". The job log shows the following steps:

- Running with gitlab-runner 14.6.0 (5316d4ac)
- on My first Runner on Banvien internal YC2y_msG
- Preparing the "shell" executor (00:00)
- Using Shell executor...
- Preparing environment (00:01)
- Running on L-226...
- Getting source from Git repository (00:06)
- Fetching changes with git depth set to 50...
- Reinitialized existing Git repository in C:/GitLab-Runner/builds/YC2y_msG/0/tam.tran-mi-nh0614/MyFirstProject/.git/
- Checking out 91b4050a as main...
- git-lfs/2.13.3 (GitHub; windows amd64; go 1.16.2; git a5e65851)
- Skipping Git submodules setup
- Executing "step_script" stage of the job script (00:21)
- \$ echo "This job tests something, but takes more time than test-job1."
- This job tests something, but takes more time than test-job1.
- \$ echo "After the echo commands complete, it runs the sleep command for 20 seconds"
- After the echo commands complete, it runs the sleep command for 20 seconds
- \$ echo "which simulates a test that runs 20 seconds longer than test-job1"
- which simulates a test that runs 20 seconds longer than test-job1
- \$ sleep 20
- Cleaning up project directory and file based variables (00:01)
- Job succeeded

The right sidebar shows the job summary for "test-job2":

- Duration: 33 seconds
- Finished: 1 day ago
- Timeout: 1h (from project)
- Runner: #13139471 (YC2y_msG) My first Runner on Banvien internal
- Commit 91b4050a
- Update .gitlab-ci.yml
- Pipeline #445331118 for main
- test
- test-job1
- test-job2

- The detail running result of *deploy-prod* job.

The screenshot displays the GitLab CI/CD interface for a project named "MyFirstProject". The left sidebar shows the navigation menu with "Jobs" selected. The main area shows the details for the "deploy-prod" job, which was triggered 1 day ago by Tam Tran and has a status of "passed". The job log shows the following steps:

- Running with gitlab-runner 14.6.0 (5316d4ac)
- on My first Runner on Banvien internal YC2y_msG
- Preparing the "shell" executor (00:00)
- Using Shell executor...
- Preparing environment (00:01)
- Running on L-226...
- Getting source from Git repository (00:07)
- Fetching changes with git depth set to 50...
- Reinitialized existing Git repository in C:/GitLab-Runner/builds/YC2y_msG/0/tam.tran-mi-nh0614/MyFirstProject/.git/
- Checking out 91b4050a as main...
- git-lfs/2.13.3 (GitHub; windows amd64; go 1.16.2; git a5e65851)
- Skipping Git submodules setup
- Executing "step_script" stage of the job script (00:01)
- \$ echo "This job deploys something from the \$CI_COMMIT_BRANCH branch."
- This job deploys something from the main branch.
- Cleaning up project directory and file based variables (00:00)
- Job succeeded

The right sidebar shows the job summary for "deploy-prod":

- Duration: 11 seconds
- Finished: 1 day ago
- Timeout: 1h (from project)
- Runner: #13139471 (YC2y_msG) My first Runner on Banvien internal
- Commit 91b4050a
- Update .gitlab-ci.yml
- Pipeline #445331118 for main
- deploy
- deploy-prod