

## Lab 1: Advanced GitLab CI/CD Techniques

In this lab, we will create GitLab account, run GitLab runner instance locally and create and run GitLab CI/CD pipeline.

### Task: Using SSH keys with GitLab private repositories

This task walks you through the steps of creating your own project, editing a file, and committing changes to a Git repository from the command line.

When you're done, you'll have a project where you can practice using Git.

### What you need

Before you begin:

- Ensure you can sign in to an instance of GitLab. If your organization doesn't have GitLab, create an account on <https://gitlab.com/>.
- Create SSH keys and add them to GitLab. SSH keys are how you securely communicate between your computer and GitLab.

### Create SSH key

SSH uses two keys, a public key and a private key.

- The public key can be distributed.
- The private key should be protected.

When you need to copy or upload your SSH public key, make sure you do not accidentally copy or upload your private key instead.

Let's create new ssh key using the VM. GitLab recommendation is to create SSH key type ED25519, which is more secure than RSA.

### Generate an SSH key pair

If you do not have an existing SSH key pair, generate a new one:

1. Open a terminal.
2. Run `ssh-keygen -t` followed by the key type and an optional comment. This comment is included in the .pub file that's created. You may want to use an email address for the comment. To create new key run following command. Text after `-C` option is a comment and you can change it.

```
ssh-keygen -t ed25519 -C "GitLab SSH key"
```

3. Press `Enter`. Output similar to the following is displayed:

```
Generating public/private ed25519 key pair.  
Enter file in which to save the key (~/.ssh/id_ed25519):
```

4. Accept the suggested filename and directory.

5. Do not specify a passphrase:

```
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:
```

A confirmation is displayed, including information about where your files are stored.

```
...ojktylwww-over-ssh-deploy -- azureuser@GitLab-vm: ~/.ssh -- ssh -i GitLab-vm_key.pem azureuser@104.40.185.136
azureuser@GitLab-vm:~/.ssh$ ssh-keygen -t ed25519 -C "GitLab SSH key"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/azureuser/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/azureuser/.ssh/id_ed25519
Your public key has been saved in /home/azureuser/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:tE0t9xU0Hx8KauDxzLAOLmLTroQRaHU61WeMavic8c4 GitLab SSH key
The key's randomart image is:
+--[ED25519 256]--+
|  o+*E..o. +oo |
|  o .X Boo=. + o|
|  . B.+o==.o o.|
|  ..+==+ o . .|
|  + So. . |
|  + . |
|  o |
|  E |
|  |
+-----[SHA256]-----+
azureuser@GitLab-vm:~/.ssh$ ls ~/.ssh
authorized_keys  id_ed25519  id_ed25519.pub
```

The key will be created in default directory which for linux is `~/.ssh`. You should have two new files in `.ssh` directory:

- `id_ed25519` --- private key
- `id_ed25519.pub` --- public key

### Add an SSH key to your GitLab account

To use SSH with GitLab, copy your public key to your GitLab account:

1. Copy the contents of your public key file.

```
cat ~/.ssh/id_ed25519.pub
```

2. Sign in to GitLab.
3. On the top bar, in the upper-right corner, select your avatar.
4. Select **Preferences**.
5. On the left sidebar, select SSH Keys.
6. In the Key box, paste the contents of your public key. If you manually copied the key, make sure you copy the entire key, which starts with `ssh-ed25519` and may end with a comment.
7. In the **Title** box, type a description, like `Work Laptop` or `Home Workstation`.
8. Optional. Select the **Usage type** of the key. It can be used either for Authentication or Signing or both. Authentication & Signing is the default value.
9. Optional. **Update Expiration** date to modify the default expiration date.
10. Select **Add key**.

### Verify that you can connect

Verify that your SSH key was added correctly.

Open a terminal and run this command, replacing `gitlab.com` with your GitLab instance URL:

```
ssh -T git@gitlab.com
```

If this is the first time you connect, you should verify the authenticity of the GitLab host. If you see a message like:

```
The authenticity of host 'gitlab.com (35.231.145.151)' can't be established.
ECDSA key fingerprint is SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gitlab.com' (ECDSA) to the list of known hosts.
```

Type `yes` and press `Enter`.

Run the `ssh -T git@gitlab.com` command again. You should receive a Welcome to GitLab, @username! message.

```
root@982e2c2fb78d:~/Desktop# ssh -T git@gitlab.com
Welcome to GitLab, @athertahir!
root@982e2c2fb78d:~/Desktop#
```

View your account’s SSH keys

- 1. Sign in to GitLab.
- 2. On the top bar, in the upper-right corner, select your avatar.
- 3. Select Preferences.
- 4. On the left sidebar, select SSH Keys. Your existing SSH keys are listed at the bottom of the page. The information includes:
  - The key’s:
    - Name.
    - Public fingerprint.
    - Expiry date.
    - Permitted usage types.

User Settings > SSH Keys > Work Laptop

SSH Key

Title: **Work Laptop**

Usage type: **Authentication & Signing**

Created on: **Feb 13, 2023 8:03pm**

Expires: **May 1, 2024 12:00am**

Last used on: **Never**

AAC3NzaC1LZDI1NTE5AAAAIHRh4h0iVFYu808/fnR1/5Anzzw1ShwB5xf1MV48fHKC GitLab

Fingerprints

MD5: db:df:bb:e9:93:3d:85:17:3b:1a:10:37:cf:6f:54:a5

SHA256: Kh3/Q0XySvwCsIHcvAU4j57D04CKof0bn1TYZnQHRBQ

Delete

Steps

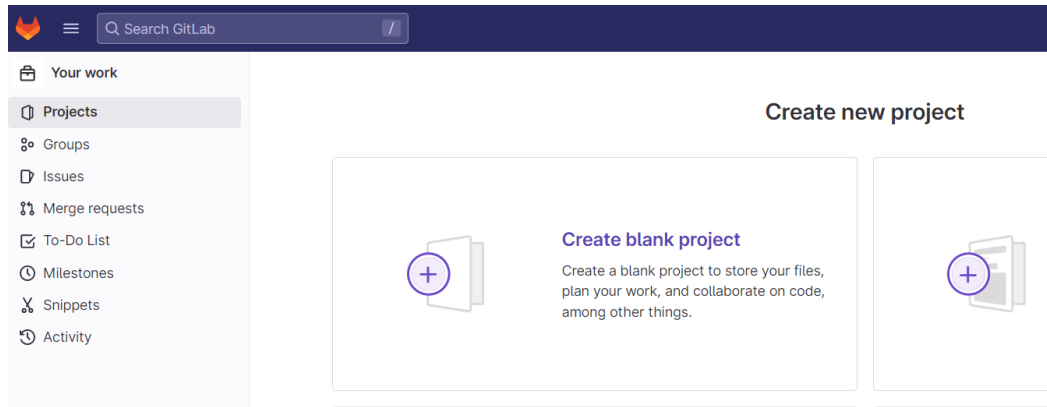
Here's an overview of what we're going to do:

- 1. Create a sample project.
- 2. Clone the repository.
- 3. Create a branch and make your changes.
- 4. Commit and push your changes.
- 5. Merge your changes.
- 6. View your changes in GitLab.

Create a sample project

To start, create a sample project in GitLab.

1. In GitLab, on the top bar, select **Main menu > Projects > View all projects**.
2. On the right of the page, select **New project > Create blank project**.

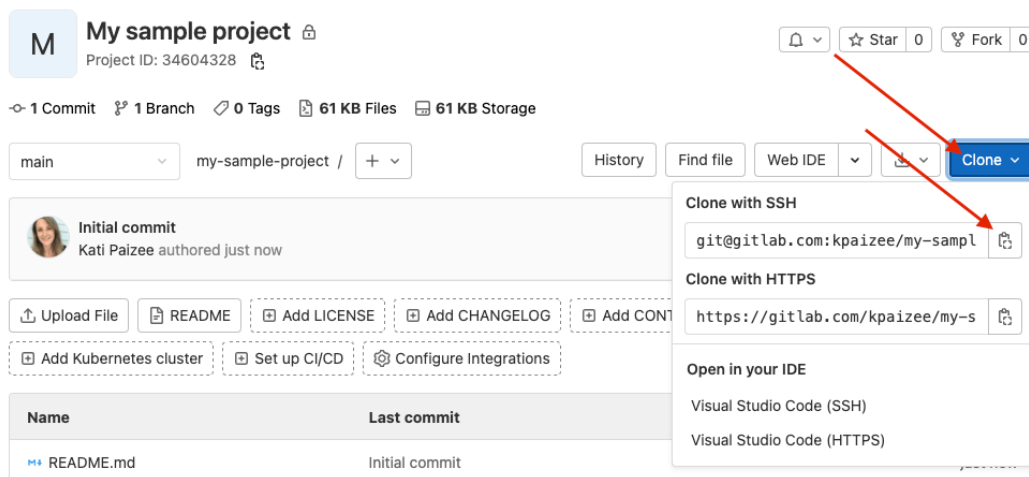


3. For **Project name**, enter `My sample project`. The project slug is generated for you. This slug is the URL you can use to access the project after it's created.
4. Ensure **Initialize repository with a README** is selected. How you complete the other fields is up to you.
5. Select **Create project**.

## Clone the repository

Now you can clone the repository in your project. *Cloning* a repository means you're creating a copy on your computer, or wherever you want to store and work with the files.

1. On your project page, select **Clone**. Copy the URL for **Clone with SSH**.



2. Open a terminal on your computer and go to the directory where you want to clone the files.
3. Enter `git clone` and paste the URL:

```
git clone git@gitlab.com:YOUR_GITLAB_USERNAME/my-sample-project.git
```

4. Go to the directory:

```
cd my-sample-project
```

5. By default, you've cloned the default branch for the repository. Usually this branch is `main`. To be sure, get the name of the default branch:

```
git branch
```

The branch you're on is marked with an asterisk. Press `Q` on your keyboard to return to the main terminal window.

## Create a branch and make changes

Now that you have a copy of the repository, create your own branch so you can work on your changes independently.

1. Create a new branch called `example-tutorial-branch`.

```
git checkout -b example-tutorial-branch
```

2. In a text editor like Visual Studio Code, Sublime, `nano`, or any other editor, open the `README.md` file and add this text:

```
Hello world! I'm using Git!
```

3. Save the file.
4. Git keeps track of changed files. To confirm which files have changed, get the status.

```
git status
```

You should get output similar to the following:

```
On branch example-tutorial-branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

## Commit and push your changes

You've made changes to a file in your repository. Now it's time to record those changes by making your first commit.

1. Add the `README.md` file to the *staging* area. The staging area is where you put files before you commit them.

```
git add README.md
```

2. Confirm the file is staged:

```
git status
```

You should get output similar to the following, and the filename should be in green text.

```
On branch example-tutorial-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

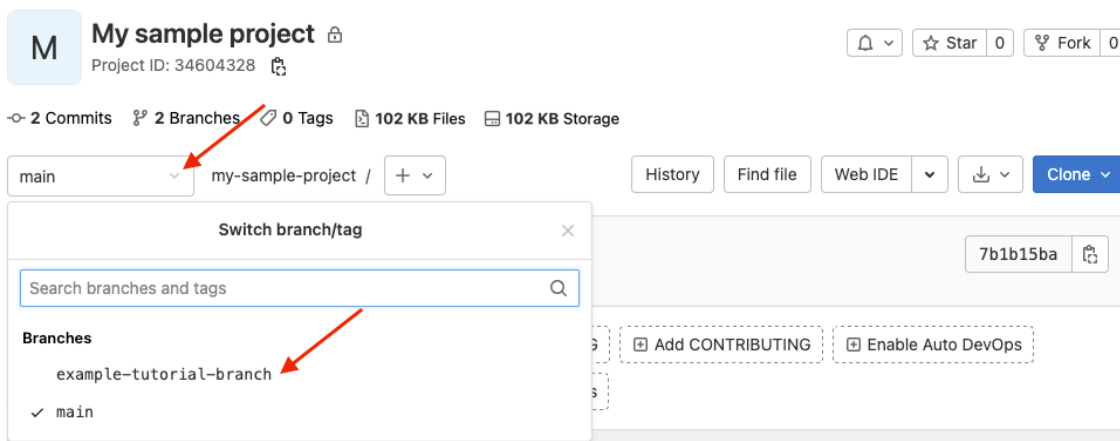
3. Now commit the staged file, and include a message that describes the change you made. Make sure you surround the message in double quotes (").

```
git commit -m "I added text to the README file"
```

4. The change has been committed to your branch, but your branch and its commits are still only available on your computer. No one else has access to them yet. Push your branch to GitLab:

```
git push origin example-tutorial-branch
```

Your branch is now available on GitLab and visible to other users in your project.



## Merge your changes

Now you're ready to merge the changes from your `example-tutorial-branch` branch to the default branch (`main`).

1. Check out the default branch for your repository.

```
git checkout main
```

2. Merge your branch into the default branch.

```
git merge example-tutorial-branch
```

3. Push the changes.

```
git push
```


For this lab, you merge your branch directly to the default branch for your repository. In GitLab, you typically use a `merge request` to merge your branch which will be created in upcoming lab.

## View your changes in GitLab

You did it! You updated the `README.md` file in your branch, and you merged those changes into the `main` branch.

Let's look in the UI and confirm your changes. Go to your project.

- Scroll down and view the contents of the `README.md` file. Your changes should be visible.
- Above the `README.md` file, view the text in the **Last commit** column. Your commit message is displayed in this column:

Name	Last commit	Last update
 README.md	I added text to the README file	16 seconds ago

Now you can return to the command line and change back to your personal branch ( `git checkout example-tutorial-branch` ). You can continue updating files or creating new ones. Type `git status` to view the status of your changes and commit with abandon.

## Run GitLab Runner in a container

We can run github runners locally. In this lab, we will register gitlab-runner on `gitlab.com`.

**NOTE:** Make sure to open new terminal and connect with your remote VM before running docker commands below:

```
`ssh ubuntu@YOUR_VM_DNS_NAME
```

**Password:** Will be provided by Instructor.

```
root@982e2c2fb78d:~# ssh root@gitlab-ansible-dev.courseware.io
root@gitlab-ansible-dev.courseware.io's password:
Welcome to Ubuntu 22.10 (GNU/Linux 5.19.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Feb 13 23:49:59 UTC 2023

System load:  0.0703125      Users logged in:      1
Usage of /:   13.0% of 154.96GB IPv4 address for docker0: 172.17.0.1
Memory usage: 15%           IPv4 address for eth0:  143.244.152.105
Swap usage:   0%            IPv4 address for eth0:  10.10.0.5
Processes:   163            IPv4 address for eth1:  10.116.0.2

94 updates can be applied immediately.
68 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Mon Feb 13 23:48:16 2023 from 172.17.0.2
```

In this lab, you can use a configuration container to mount your custom data volume.

Create the Docker volume:

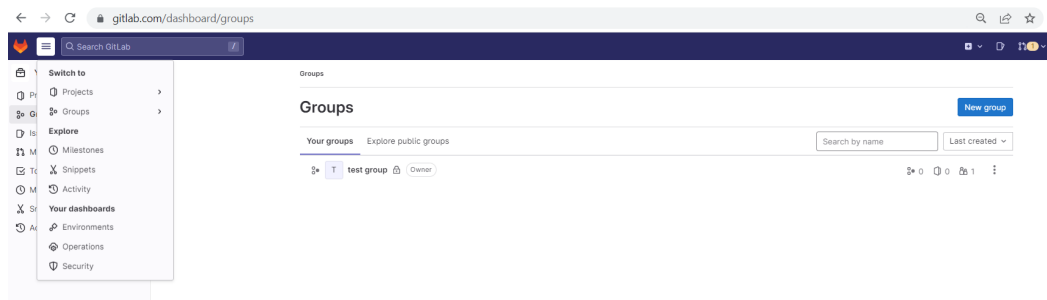
```
docker volume create gitlab-runner-config
```

Start the GitLab Runner container using the volume we just created:

```
docker run -d --name gitlab-runner --restart always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v gitlab-runner-config:/etc/gitlab-runner \
gitlab/gitlab-runner:latest
```

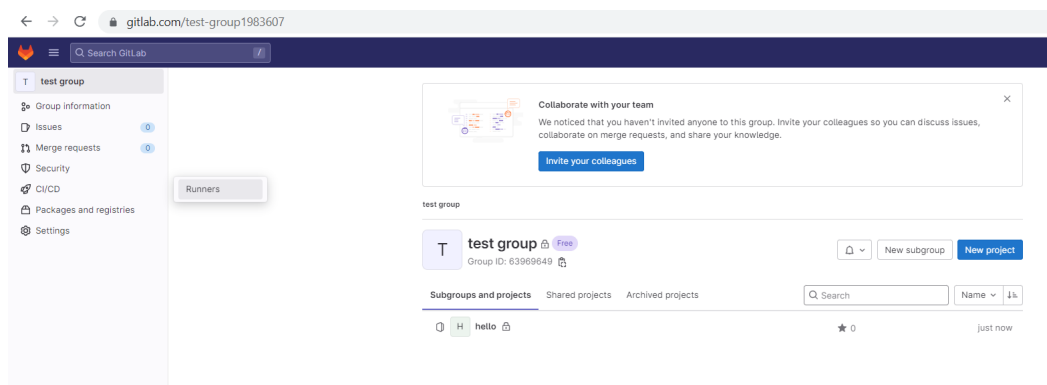
## Group Runner Setup

1. On the top bar, select **Main menu** > **Groups** and find your group.



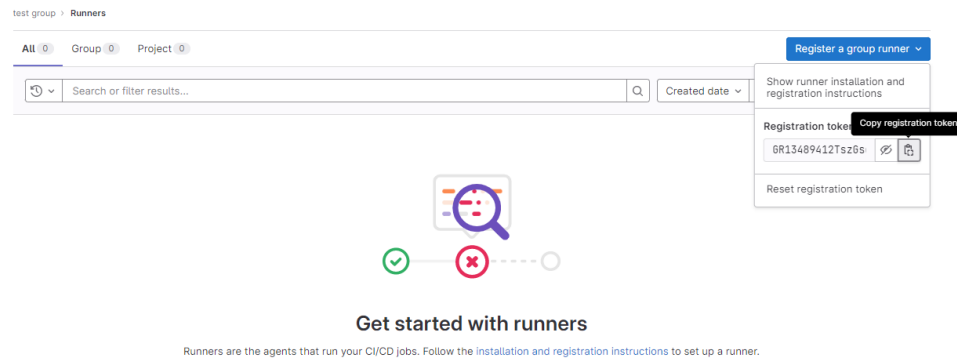
**Note:** If you don't have one, click **New Group** to create new one.

2. On the left sidebar, select **CI/CD** > **Runners**.



3. In the upper-right corner, select **Register a group runner** and copy **Registration token** from the next steps.





## Register a Runner

1. To register a runner using a Docker container:

```
docker run --rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest register
```

2. Enter your GitLab instance URL (also known as the gitlab-ci coordinator URL): `https://gitlab.com/`
3. Enter the token you obtained to register the runner.
4. Enter a description for the runner. You can change this value later in the GitLab user interface.
5. Enter the tags associated with the runner, separated by commas. You can change this value later in the GitLab user interface.
6. Enter any optional maintenance note for the runner.
7. Provide the runner executor: enter **docker**.
8. If you entered docker as your executor, you are asked for the default image to be used for projects that do not define one in `.gitlab-ci.yml`: enter **ruby:2.7**

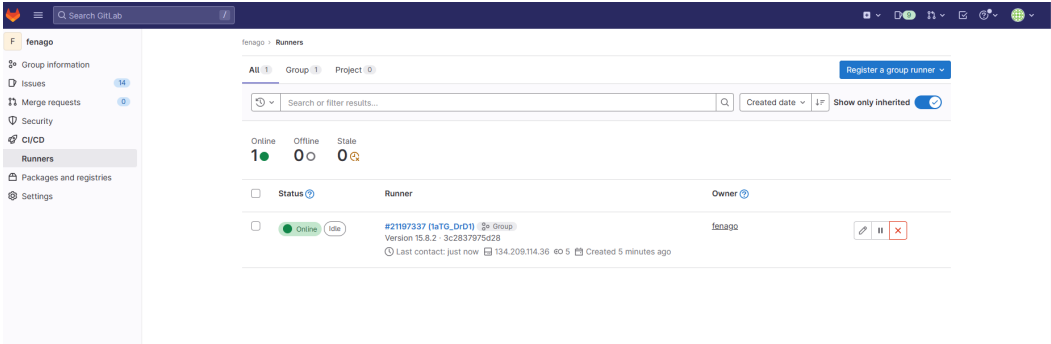
```
root@gitlabci-ansible22:~# docker run --rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest register
Runtime platform arch=amd64 os=linux pid=7 revision=4d1ca121 version=15.8.2
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR13489412Gubw5dCZPF-RA8Raz1D
Enter a description for the runner:
[5-c831975408]:
Enter tags for the runner (comma-separated):

Enter optional maintenance note for the runner:

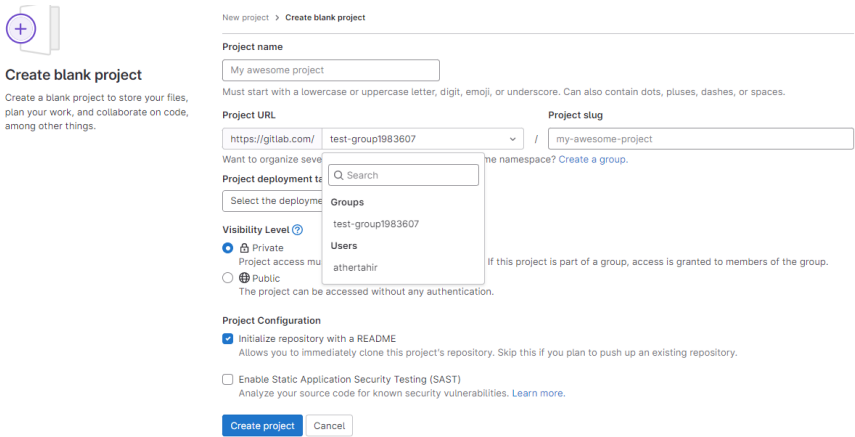
WARNING: Support for registration tokens and runner parameters in the 'register' command has been deprecated in GitLab Runner 15.6 and will be replaced with support for authentication tokens. For more information, see https://gitlab.com/gitlab-org/gitlab/-/issues/380872
Registering runner... succeeded runner=GR13489412Gubw5dC
Enter an executor: shell, ssh, virtualbox, docker+machine, instance, kubernetes, docker, docker-ssh, docker-ssh+machine, custom, parallels:
docker
Enter the default Docker image (for example, ruby:2.7):
ruby:2.7
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
Configuration (with the authentication token) was saved in "/etc/gitlab-runner/config.toml"
root@gitlabci-ansible22:~#
```

9. Reload the webpage, you should see one runner available.



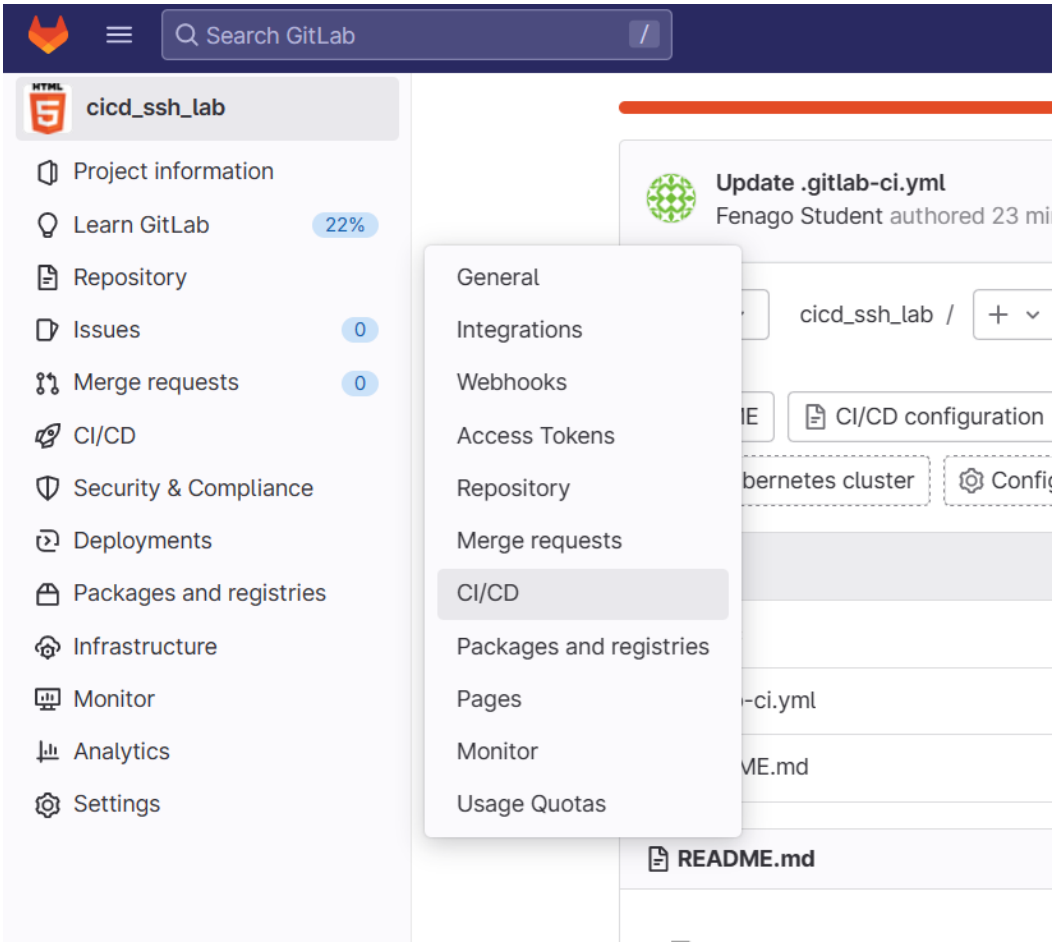
Using Gitlab Runner in CI/CD

1. Important! Make sure to create new project inside the group so that you can use the runner:



2. Disable instance runners for all the projects that you create. Otherwise, gitlab will fail your pipeline and ask for account verification.

3. On the left sidebar, select `Settings` > `CI/CD` .



4. Expand `Runners` and disable instance runners for this project.

### Runners

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

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

**How do runners pick up jobs?**

Runners are either:

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

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

#### Project runners

These runners are assigned to this project.

New project runner ⋮

#### Instance runners

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

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

Enable instance runners for this project

☒

5. Scroll to `Group runners` and confirm one runner is available.


## Group runners

These runners are shared across projects in this group.

Group runners can be managed with the [Runner API](#).

[Disable group runners](#) for this project

### Available group runners: 1

 #21197337 (1aTG\_DrD1)  
3c2837975d28

**Important!** You will need to disable shared runner for all gitlab projects.

## Task: Create and run GitLab CI/CD pipeline

This lab shows you how to configure and run your first CI/CD pipeline in GitLab.

## Prerequisites

Before you start, make sure you have:

- A project in GitLab that you would like to use CI/CD for.
- The Maintainer or Owner role for the project.

If you don't have a project, you can create a public project for free on <https://gitlab.com>.

## Steps

To create and run your first pipeline:

1. Ensure you have runners available to run your jobs.

**Important!** Disable instance runners for all the projects that you create. Otherwise, gitlab will fail your pipeline and ask for account verification.

2. Create a `.gitlab-ci.yml` file at the root of your repository. This file is where you define the CI/CD jobs.

When you commit the file to your repository, the runner runs your jobs. The job results are displayed in a pipeline.

## Ensure you have runners available

In GitLab, runners are agents that run your CI/CD jobs.

To view available runners:

- Go to **Settings > CI/CD** and expand **Runners**.

As long as you have at least one runner that's active, with a green circle next to it, you have a runner available to process your jobs.

When your CI/CD jobs run, in a later step, they will run on your local machine.

## Create a `.gitlab-ci.yml` file

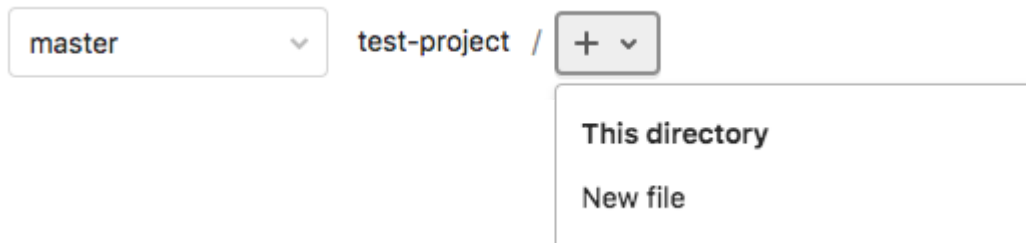
Now create a `.gitlab-ci.yml` file. It is a **YAML** file where you specify instructions for GitLab CI/CD.

In this file, you define:

- The structure and order of jobs that the runner should execute.
- The decisions the runner should make when specific conditions are encountered.

To create a `.gitlab-ci.yml` file:

1. On the left sidebar, select **Repository > Files**.
2. Above the file list, select the branch you want to commit to. If you're not sure, leave `master` or `main`. Then select the plus icon and **New file**:



3. For the **Filename**, type `.gitlab-ci.yml` and in the larger window, paste this sample code:

```
build-job:
  stage: build
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"

test-job1:
  stage: test
  script:
    - echo "This job tests something"

test-job2:
  stage: test
  script:
    - echo "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"
    - echo "which simulates a test that runs 20 seconds longer than test-job1"
```

```

- sleep 20

deploy-prod:
  stage: deploy
  script:
    - echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
  environment: production

```

This example shows four jobs: `build-job`, `test-job1`, `test-job2`, and `deploy-prod`. The comments listed in the `echo` commands are displayed in the UI when you view the jobs. The values for the [predefined variables] `$GITLAB_USER_LOGIN` and `$CI_COMMIT_BRANCH` are populated when the jobs run.





#### 4. Select **Commit changes**.

The pipeline starts and runs the jobs you defined in the `.gitlab-ci.yml` file.

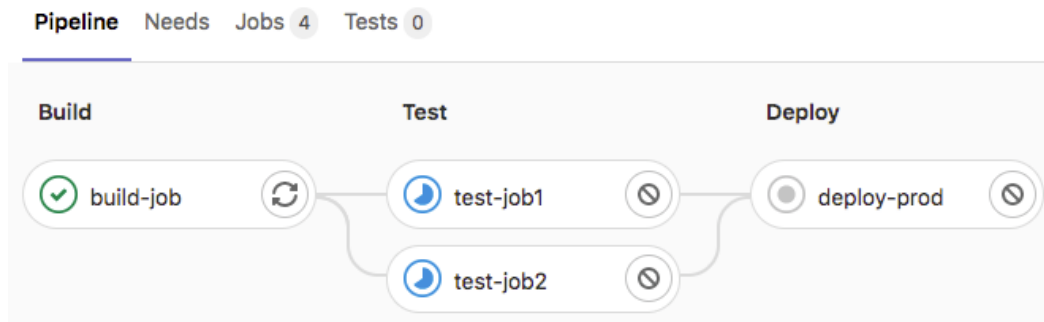
## View the status of your pipeline and jobs

Now take a look at your pipeline and the jobs within.

1. Go to **CI/CD > Pipelines**. A pipeline with three stages should be displayed:

Status	Pipeline	Triggerer	Commit	Stages
 running	#217408060 <span>latest</span>		 master → 271ad0cd Update .gitlab-ci.yml	

2. View a visual representation of your pipeline by selecting the pipeline ID:



3. View details of a job by selecting the job name. For example, `deploy-prod`:



Job #855275091 triggered 23 minutes ago by Suzanne Selhorn

```
1 Running with gitlab-runner 13.6.0-rc1 (d83ac56c)
2   on docker-auto-scale ed2dce3a
3   ✓ Preparing the "docker+machine" executor
4   Using Docker executor with image ruby:2.5 ...
5   Pulling docker image ruby:2.5 ...
6   Using docker image sha256:b7280b81558d31d64ac82aa66a9540e04baf9d15abb8fff
   ed62cd60e4fb5bf4132943d6fa2688 ...
7   ✓ Preparing environment
8   Running on runner-ed2dce3a-project-16381496-concurrent-0 via runner-ed2dce3a
9   ✓ Getting source from Git repository
10  $ eval "$CI_PRE_CLONE_SCRIPT"
11  Fetching changes with git depth set to 50...
12  Initialized empty Git repository in /builds/sselhorn/test-project/.git/
13  Created fresh repository.
14  Checking out 7353da73 as master...
15  Skipping Git submodules setup
16  ✓ Executing "step_script" stage of the job script
17  $ echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
18  This job deploys something from the master branch.
19  ✓ Cleaning up file based variables
20  Job succeeded
```

You have successfully created your first CI/CD pipeline in GitLab. Congratulations!

## Disabling GitLab CI/CD

GitLab CI/CD is enabled by default on all new projects. If you use an external CI/CD server like Jenkins or Drone CI, you can disable GitLab CI/CD to avoid conflicts with the commits status API.

## Disable CI/CD in a project

When you disable GitLab CI/CD:

- The **CI/CD** item in the left sidebar is removed.
- The `/pipelines` and `/jobs` pages are no longer available.
- Existing jobs and pipelines are hidden, not removed.

To disable GitLab CI/CD in your project:

1. On the top bar, select **Main menu > Projects** and find your project.
2. On the left sidebar, select **Settings > General**.
3. Expand **Visibility, project features, permissions**.
4. In the **Repository** section, turn off **CI/CD**.
5. Select **Save changes**.

## Enable CI/CD in a project

To enable GitLab CI/CD in your project:

1. On the top bar, select **Main menu > Projects** and find your project.
2. On the left sidebar, select **Settings > General**.
3. Expand **Visibility, project features, permissions**.
4. In the **Repository** section, turn on **CI/CD**.
5. Select **Save changes**.