GitLab/Git

GitLab

Reference- https://docs.gitlab.com/user/get_started/

GitLab is an open-source platform built around Git, the distributed version control system developed by Linus Torvalds. It provides a comprehensive set of tools and features to manage Git repositories, project planning, continuous integration/continuous deployment (CI/CD), code review, issue tracking, and more. Essentially, GitLab consolidates the entire software development lifecycle into one seamless interface, simplifying the process and making it more efficient.

Important Points about GitLab

1. At its core, GitLab is a powerful Git repository manager. It allows developers to create, clone, push, and pull repositories, providing version control capabilities that ensure changes to code are tracked, managed, and merged efficiently.

2. GitLab offers a built-in issue tracking system, enabling teams to create, assign, and track issues, bugs, and feature requests. It also provides a project management board with customizable workflows to visualise tasks and progress.

3. One of GitLab's standout features is its integrated CI/CD pipelines. Developers can automate the testing, building, and deployment processes, ensuring code changes are thoroughly tested before being merged into the main codebase and deployed to production.

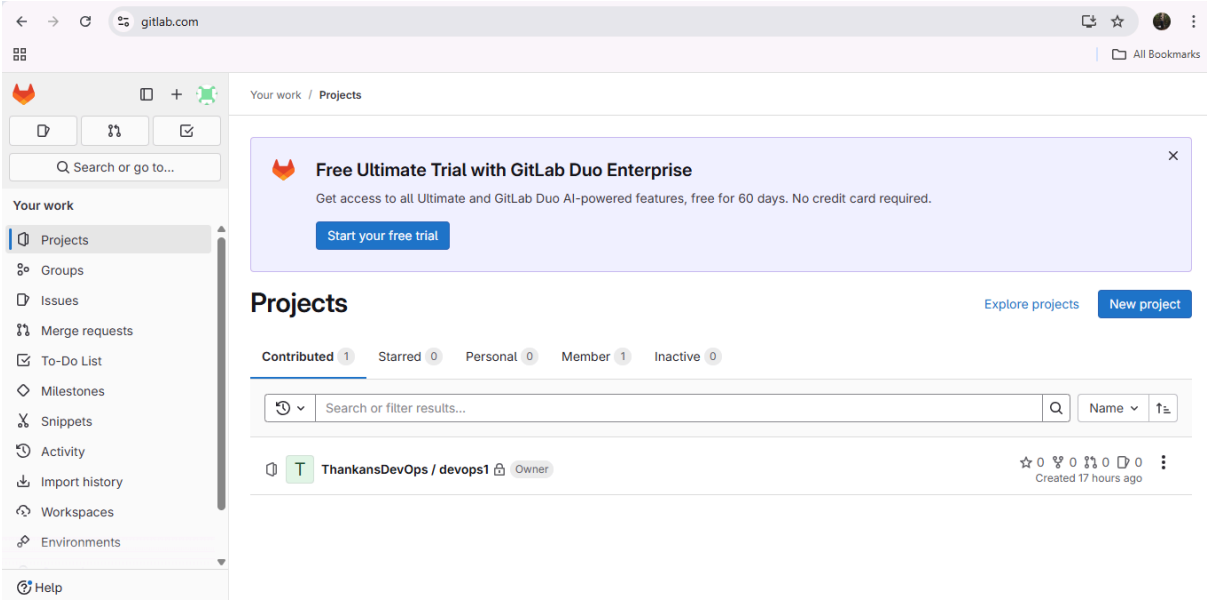
BENEFITS OF GITLAB-

- Enhanced Collaboration: GitLab centralises development activities, facilitating seamless collaboration among team members regardless of their geographical location.
- Improved Productivity: With automated CI/CD pipelines and streamlined workflows, developers can focus more on writing code and less on repetitive tasks, improving overall productivity.
- Faster Time to Market: GitLab's built-in code review process ensures that code changes are thoroughly assessed, leading to higher-quality code and reduced bugs in the production environment.
- Transparency and Accountability: The issue tracking and project management features provide transparency on project progress and help hold team members accountable for their tasks.
- Unified Platform: GitLab replaces multiple tools, reducing context shifts, enhancing speed, and cutting DevOps toolchain costs.

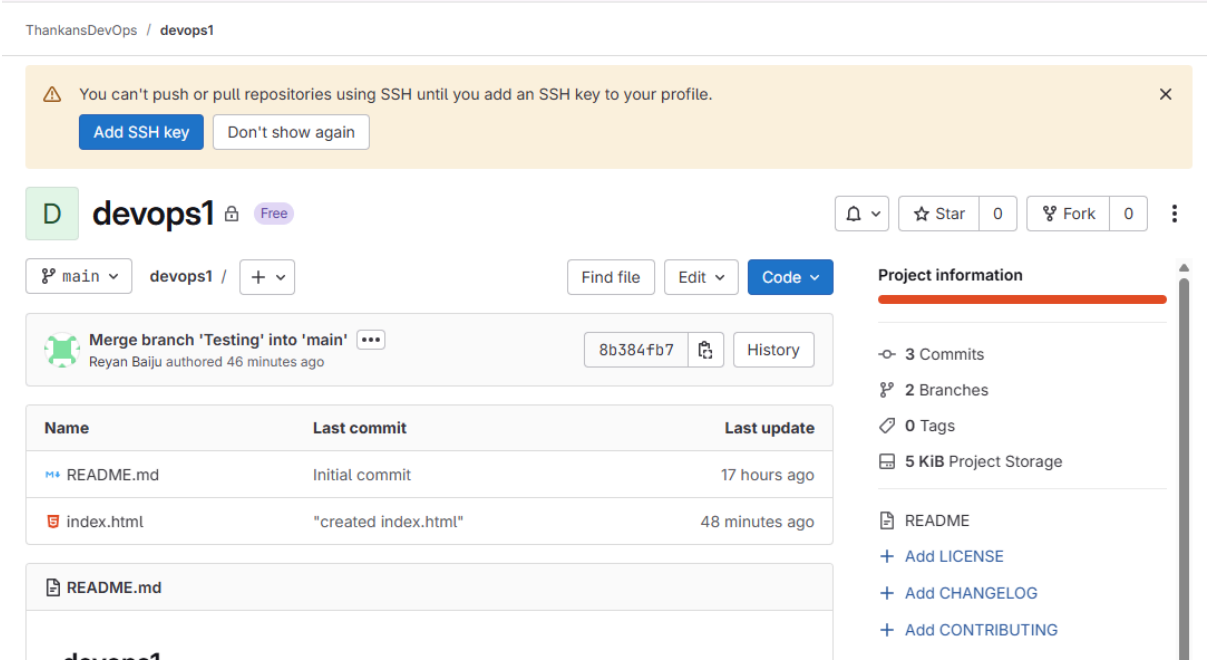
Using GitLab

GitLab can be accessed either on the gitlab cloud(on the internet) or on a self hosted server.

GitLab.com-



Repository page-



Self Hosting GitLab using Docker Compose-

Reference-
https://docs.gitlab.com/install/docker/installation/#install-gitlab-by-using-docker-compose

As gitlab uses port 22 to interact with git by default, I am going to map a new port for ssh into the server.

Just edit the sshd_config file and restart the vm.

```
ubuntu@ip-172-31-16-209: /et
GNU nano 7.2                                                    sshd_c

# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

Port 4500
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO
```

```
C:\Users\reyan>ssh -i "Downloads\cont.pem" -p 4500 ubuntu@13.213.45.156
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

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

 System information as of Mon Apr 21 08:22:02 UTC 2025

  System load:  0.81               Processes:             105
  Usage of /:   25.6% of 6.71GB    Users logged in:       0
  Memory usage: 20%                IPv4 address for enX0: 172.31.16.209
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Mon Apr 21 08:19:42 2025 from 111.92.118.119
ubuntu@ip-172-31-16-209:~$
```

Now, we can install docker from this site-
https://docs.docker.com/engine/install/ubuntu/

```
ubuntu@ip-172-31-16-209: /e
ubuntu@ip-172-31-16-209:/etc/ssh$ # Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:6 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:8 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:9 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:10 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:11 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:12 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:13 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1020 kB]
Get:14 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [223 kB]
Get:15 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [151 kB]
Get:16 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [13.5 kB]
Get:17 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1056 kB]
Get:18 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [266 kB]
Get:19 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [367 kB]
Get:20 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [26.0 kB]
Get:21 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [951 kB]
```

```
ubuntu@ip-172-31-16-209:/etc/ssh$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras libltdl7 libslirp0 pigz slirp4netns
Suggested packages:
  cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 63 not upgraded.
Need to get 120 MB of archives.
After this operation, 440 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libltdl7 amd64 2.4.7-7build1 [40.3 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
```

Now, docker is installed successfully.
We can check the docker version using docker –version.

```
ubuntu@ip-172-31-16-209: /e
ubuntu@ip-172-31-16-209:/etc/ssh$ docker --version
Docker version 28.1.1, build 4eba377
ubuntu@ip-172-31-16-209:/etc/ssh$
```

Now, we can pull the docker image from dockerhub, link-
https://hub.docker.com/r/gitlab/gitlab-ce

But, before that, we have to create some directories to use as volumes for docker compose.
I'm creating a new directory called "gitlab" in my home directory.
And making the required directories inside it.

```
ubuntu@ip-172-31-16-209:~/gitlab$ mkdir config
ubuntu@ip-172-31-16-209:~/gitlab$ mkdir logs
ubuntu@ip-172-31-16-209:~/gitlab$ mkdir data
ubuntu@ip-172-31-16-209:~/gitlab$ ls
config  data  logs
```

Create a directory for the volumes

Create a directory for the configuration files, logs, and data files. The directory can be in your user's home directory (for example `~/gitlab-docker`), or in a directory like `/srv/gitlab`.

1. Create the directory:

   Shell

   ```
   sudo mkdir -p /srv/gitlab
   ```

2. If you're running Docker with a user other than `root`, grant the appropriate permissions to the user for the new directory.

I'm giving appropriate permissions to my user to use the directory.

```
ubuntu@ip-172-31-16-209:~$ ls -la
total 36
drwxr-x--- 5 ubuntu ubuntu 4096 Apr 21 10:07 .
drwxr-xr-x 3 root   root   4096 Apr 21 07:53 ..
-rw------- 1 ubuntu ubuntu   79 Apr 21 08:20 .bash_history
-rw-r--r-- 1 ubuntu ubuntu  220 Mar 31  2024 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Mar 31  2024 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Apr 21 08:19 .cache
-rw-r--r-- 1 ubuntu ubuntu  807 Mar 31  2024 .profile
drwx------ 2 ubuntu ubuntu 4096 Apr 21 07:53 .ssh
-rw-r--r-- 1 ubuntu ubuntu    0 Apr 21 08:19 .sudo_as_admin_successful
drwx------ 5 ubuntu ubuntu 4096 Apr 21 10:11 gitlab
ubuntu@ip-172-31-16-209:~$
```

```
ubuntu@ip-172-31-16-209:~/gitlab$ ls -la
total 20
drwx------ 5 ubuntu ubuntu 4096 Apr 21 10:11 .
drwxr-x--- 5 ubuntu ubuntu 4096 Apr 21 10:07 ..
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 21 10:10 config
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 21 10:11 data
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 21 10:11 logs
ubuntu@ip-172-31-16-209:~/gitlab$
```

These are the uses of these directories.

Local location	Container location	Usage
`$GITLAB_HOME/data`	`/var/opt/gitlab`	Stores application data.
`$GITLAB_HOME/logs`	`/var/log/gitlab`	Stores logs.
`$GITLAB_HOME/config`	`/etc/gitlab`	Stores the GitLab configuration files.

Now we can mention it in the .yaml file.

I'm creating a new directory called GLD to create the docker-compose.yml file.

```
ubuntu@ip-172-31-21-189: ~
ubuntu@ip-172-31-21-189:~$ mkdir GLD
ubuntu@ip-172-31-21-189:~$ ls
GLD  gitlab
ubuntu@ip-172-31-21-189:~$
```

Create a docker-compose.yml file.

```
ubuntu@ip-172-31-16-209:~$ sudo touch docker-compose.yml
ubuntu@ip-172-31-16-209:~$ ls
docker-compose.yml  gitlab
ubuntu@ip-172-31-16-209:~$
```

Add your configuration in that file-

```
ubuntu@ip-172-31-19-116: ~/(
GNU nano 7.2                                                    docker-comp
services:
  gitlab:
    image: gitlab/gitlab-ce:latest
    container_name: gitlab
    restart: always
    hostname: '47.128.238.12'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        # Add any other gitlab.rb configuration here, each on its own line
        external_url 'http://47.128.238.12'
    ports:
      - '80:80'
      - '443:443'
      - '22:22'
    volumes:
      - '/home/ubuntu/gitlab/config:/etc/gitlab'
      - '/home/ubuntu/gitlab/logs:/var/log/gitlab'
      - '/home/ubuntu/gitlab/data:/var/opt/gitlab'
    shm_size: '256m'
```

Save the file.

Now, run the command “docker compose up -d”.

```
ubuntu@ip-172-31-21-189:~/GLD$ sudo docker compose up -d
[+] Running 10/10
 ✔gitlab Pulled                                                                    82.2s
   ✔30a9c22ae099 Pull complete                                                      2.9s
   ✔2b3d48fbd4b9 Pull complete                                                      3.0s
   ✔8e15757df0a9 Pull complete                                                     10.7s
   ✔02179878518c Pull complete                                                     10.9s
   ✔bb80053b3be1 Pull complete                                                     11.0s
   ✔f257e1f8a5ac Pull complete                                                     11.2s
   ✔0414f056ea9b Pull complete                                                     11.3s
   ✔de94400ee8c3 Pull complete                                                     11.4s
   ✔9e9bbdd9209e Pull complete                                                     79.2s
[+] Running 2/2
 ✔Network gld_default  Created                                                      0.1s
 ✔Container gitlab     Started                                                      0.8s
```

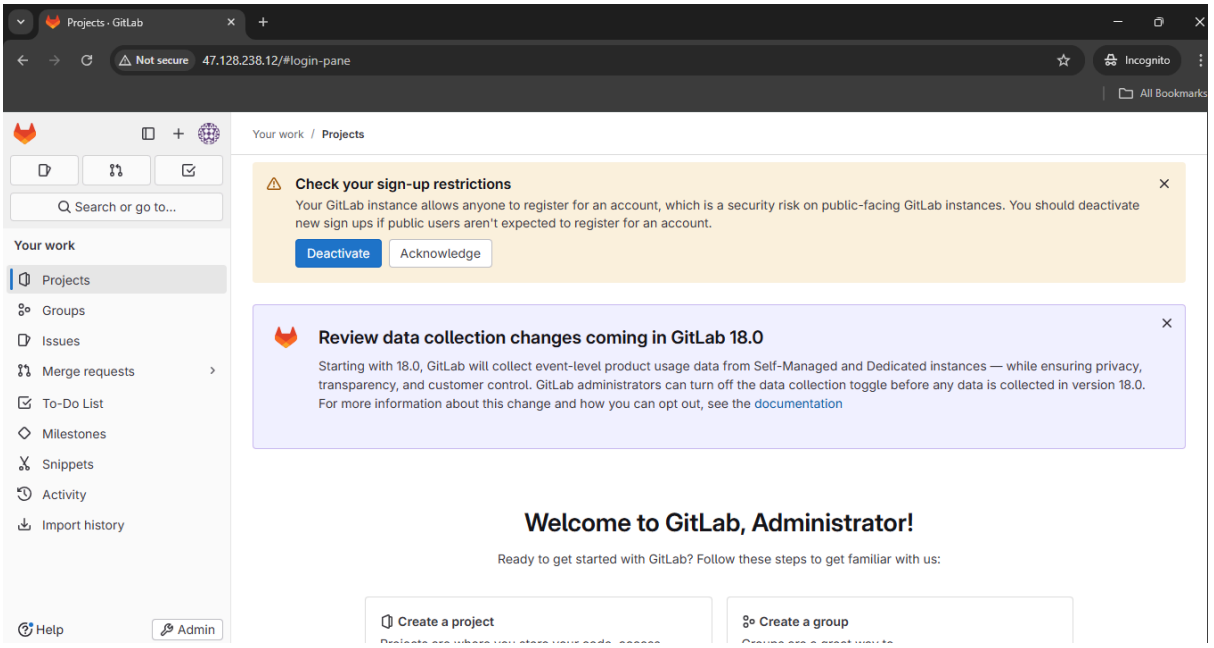
Gitlab is running successfully.
Use command sudo docker compose ps.

```
ubuntu@ip-172-31-19-116:~$ sudo docker ps -a
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS                    PORTS
                                                                                 NAMES
9aa4e966494b   gitlab/gitlab-ce:latest   "/assets/init-contai…"   29 minutes ago   Up 29 minutes (healthy)   0.0.0.0:22->22/tcp, [::]:22->22/tcp, 0.
0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   gitlab
ubuntu@ip-172-31-19-116:~$
```

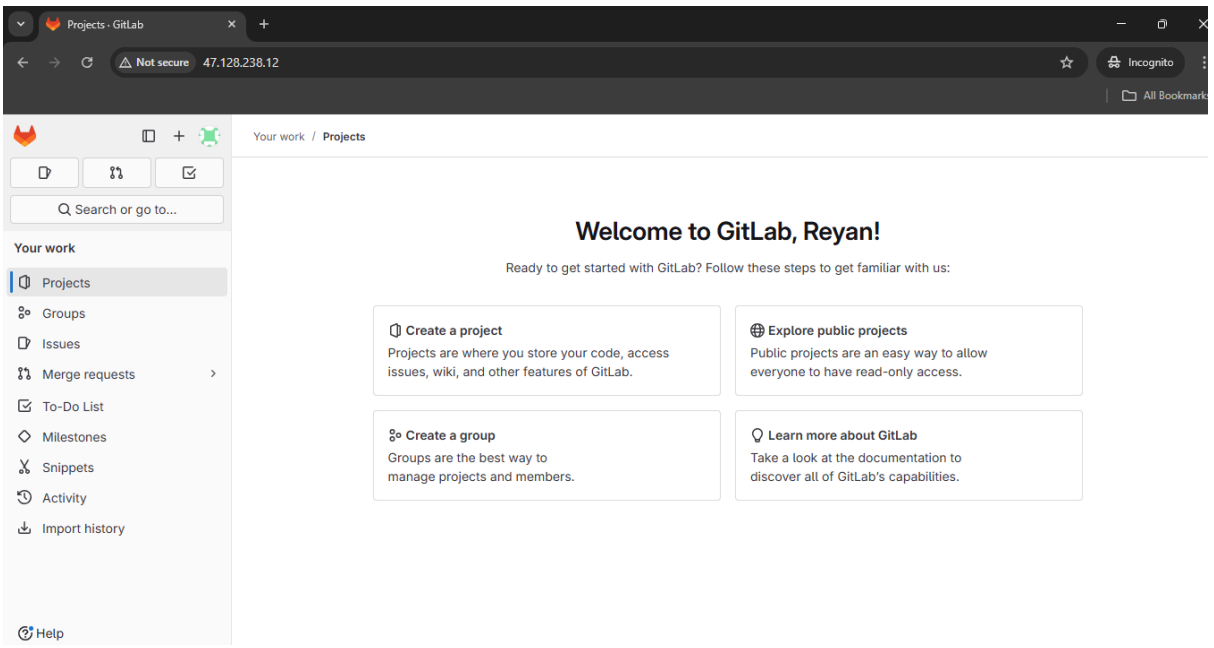
Now, we can access it from the web browser using the url http://ipaddress:80.
The initial password will be in the initial_root_
Password in the described volume.

```
ubuntu@ip-172-31-19-116:~$ cd gitlab
ubuntu@ip-172-31-19-116:~/gitlab$ ls
config  data  logs
ubuntu@ip-172-31-19-116:~/gitlab$ cd config
ubuntu@ip-172-31-19-116:~/gitlab/config$ ls
gitlab-secrets.json  initial_root_password  ssh_host_ecdsa_key.pub  ssh_host_ed25519_key.pub  ssh_host_rsa_key.pub
gitlab.rb            ssh_host_ecdsa_key     ssh_host_ed25519_key    ssh_host_rsa_key          trusted-certs
ubuntu@ip-172-31-19-116:~/gitlab/config$ cat initial_root_password
cat: initial_root_password: Permission denied
ubuntu@ip-172-31-19-116:~/gitlab/config$ sudo cat initial_root_password
# WARNING: This value is valid only in the following conditions
#          1. If provided manually (either via `GITLAB_ROOT_PASSWORD` environment variable or via `gitlab_rails['initial_root_password']` setting in
 `gitlab.rb`, it was provided before database was seeded for the first time (usually, the first reconfigure run).
#          2. Password hasn't been changed manually, either via UI or via command line.
#
#          If the password shown here doesn't work, you must reset the admin password following https://docs.gitlab.com/ee/security/reset_user_passw
ord.html#reset-your-root-password.

Password: QK0SNPlwd16IYK4+uaDOhKrOK9f8xXbnCSPZpRR00YQ=

# NOTE: This file will be automatically deleted in the first reconfigure run after 24 hours.
ubuntu@ip-172-31-19-116:~/gitlab/config$
```

Login using the username “root” and password.

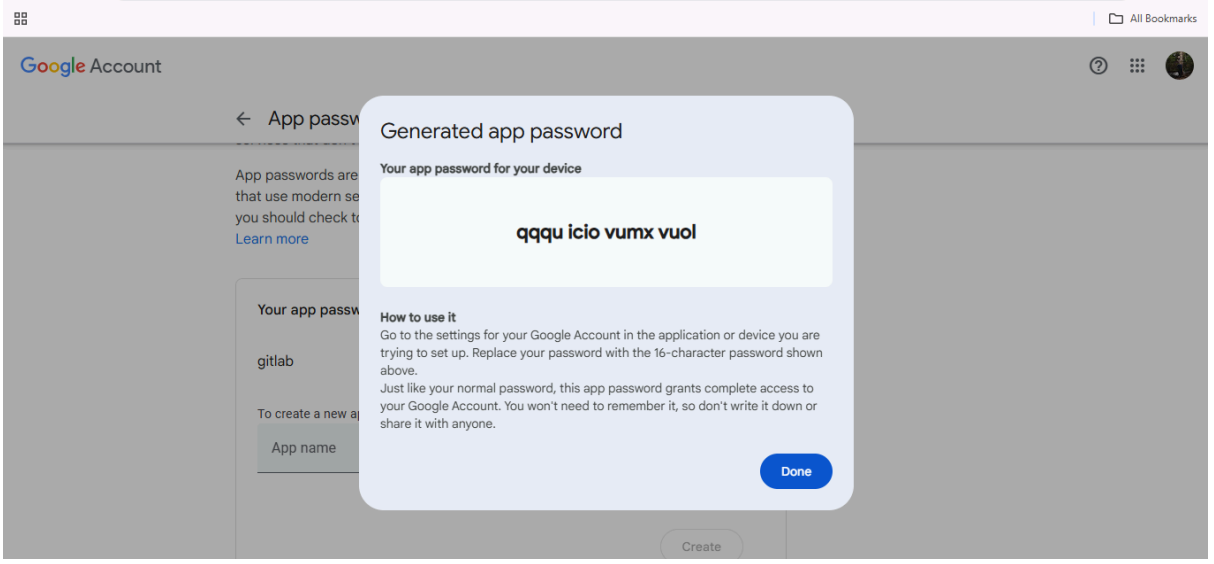


Users can now login and be approved by the administrators.
I added a user ReyaneBaiju and approved it.



[Setting up email alerts-Reference-https://docs.gitlab.com/omnibus/settings/smtp/#gmail](https://docs.gitlab.com/omnibus/settings/smtp/#gmail)

To set up email, we need to configure SMTP using gmail. First, get your gmail app password.



Then, we have to edit the /home/ubuntu/gitlab/config and edit the gitlab.rb file.

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.gmail.com"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "reyanekbaiju@gmail.com"
gitlab_rails['smtp_password'] = "rllc wlvg mnqd noua"
gitlab_rails['smtp_domain'] = "smtp.gmail.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['smtp_tls'] = false
gitlab_rails['smtp_pool'] = false

###! **Can be: 'none', 'peer', 'client_once', 'fail_if_no_peer_cert'**
###! Docs: http://api.rubyonrails.org/classes/ActionMailer/Base.html
gitlab_rails['smtp_openssl_verify_mode'] = 'peer'

# gitlab_rails['smtp_ca_path'] = "/etc/ssl/certs"
# gitlab_rails['smtp_ca_file'] = "/etc/ssl/certs/ca-certificates.crt"

### Email Settings

gitlab_rails['gitlab_email_enabled'] = true

###! If your SMTP server does not like the default 'From: gitlab@gitlab.example.com'
###! can change the 'From' with this setting.
gitlab_rails['gitlab_email_from'] = 'reyanekbaiju@gmail.com'
gitlab_rails['gitlab_email_display_name'] = 'reyanekbaiju'
gitlab_rails['gitlab_email_reply_to'] = 'reyanekbaiju@gmail.com'
# gitlab_rails['gitlab_email_subject_suffix'] = ''
# gitlab_rails['gitlab_email_smime_enabled'] = false
```

Now save and restart gitlab using both-

1. sudo docker exec -it gitlab gitlab-ctl reconfigure
2. sudo docker restart gitlab

Sending Test Email-

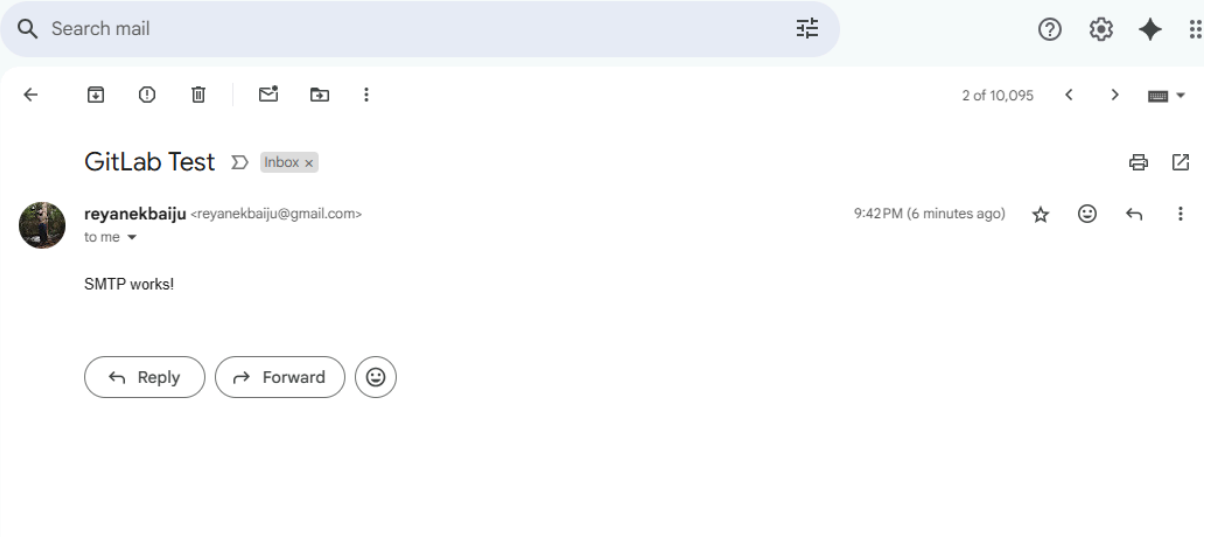
To send test email, we have to get into the gitlab ruby rails console, either use “sudo docker exec -it gitlab gitlab-rails console” or “sudo docker exec -it gitlab bash”

```
ubuntu@ip-172-31-35-142: ~/

ubuntu@ip-172-31-35-142:~/gitlab/config$ sudo docker exec -it gitlab gitlab-rails console
--------------------------------------------------------------------------------
 Ruby:         ruby 3.2.5 (2024-07-26 revision 31d0f1a2e7) [x86_64-linux]
 GitLab:       17.11.0 (5e1517f7b46) FOSS
 GitLab Shell: 14.41.0
 PostgreSQL:   16.8
------------------------------------------------------------[ booted in 32.75s ]
Loading production environment (Rails 7.0.8.7)
irb(main):001:0> Notify.test_email('your.email@gmail.com', 'GitLab Test', 'SMTP works!').deliver_now
Delivered mail 68066e404e52_29630d4-44f@52.221.222.124.mail (2531.0ms)
=> #<Mail::Message:536540, Multipart: false, Headers: <Date: Mon, 21 Apr 2025 16:11:44 +0000>, <From: reyanekbaiju <reyanekbaiju@gmail.com>>, <Reply
-To: reyanekbaiju <reyanekbaiju@gmail.com>>, <To: your.email@gmail.com>, <Message-ID: <68066e404e52_29630d4-44f@52.221.222.124.mail>>, <Subject: Git
Lab Test>, <Mime-Version: 1.0>, <Content-Type: text/html; charset=UTF-8>, <Content-Transfer-Encoding: 7bit>, <Auto-Submitted: auto-generated>, <X-Au
to-Response-Suppress: All>>
irb(main):002:0>
```

```
root@52: /
ubuntu@ip-172-31-35-142:~/gitlab/config$ sudo docker exec -it gitlab bash
root@52:/# gitlab-rails console
--------------------------------------------------------------------------------
 Ruby:         ruby 3.2.5 (2024-07-26 revision 31d0f1a2e7) [x86_64-linux]
 GitLab:       17.11.0 (5e1517f7b46) FOSS
 GitLab Shell: 14.41.0
 PostgreSQL:   16.8
------------------------------------------------------------[ booted in 32.39s ]
```

and use the send mail command-
"Notify.test_email('destination_email@address.com', 'Message Subject', 'Message Body').deliver_now"

We will get the email in our mail-



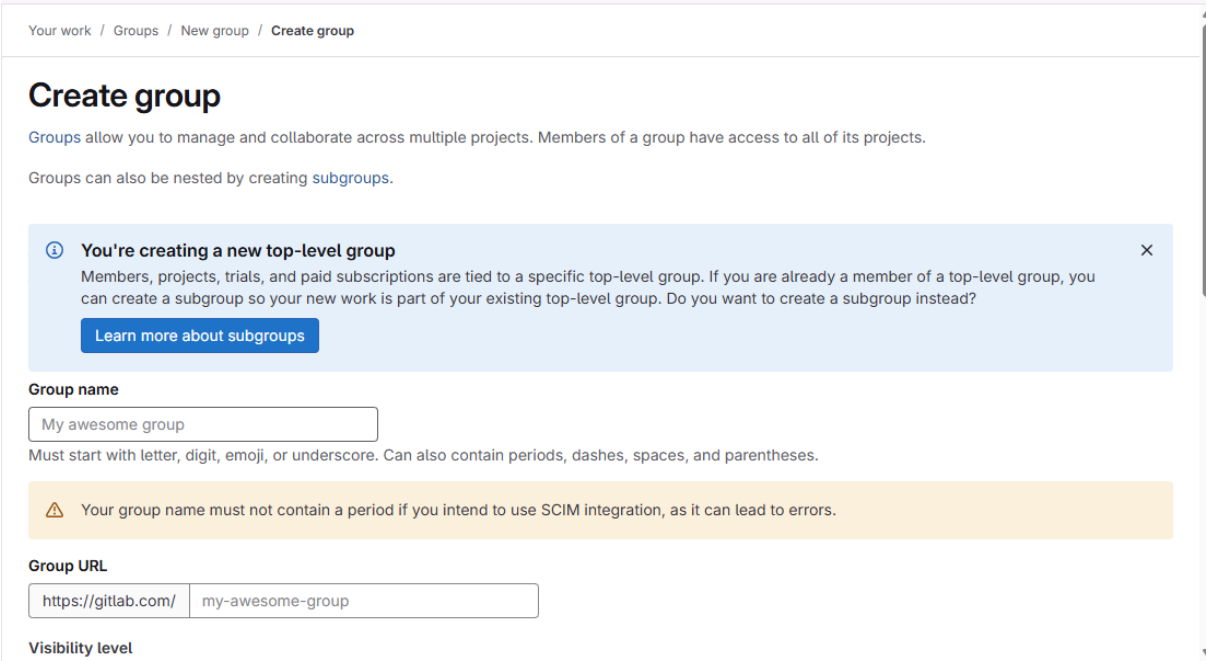
Extra info- to use git clone on self hosted gitlab servers, remember this command-
ssh://git@gitlab.example.com/user/project.git

By default, runners are not available with self-hosted servers and won't be able to run CI/CD jobs. You can add your own managed runners by registering it with GitLab.

Important GitLab Terminologies

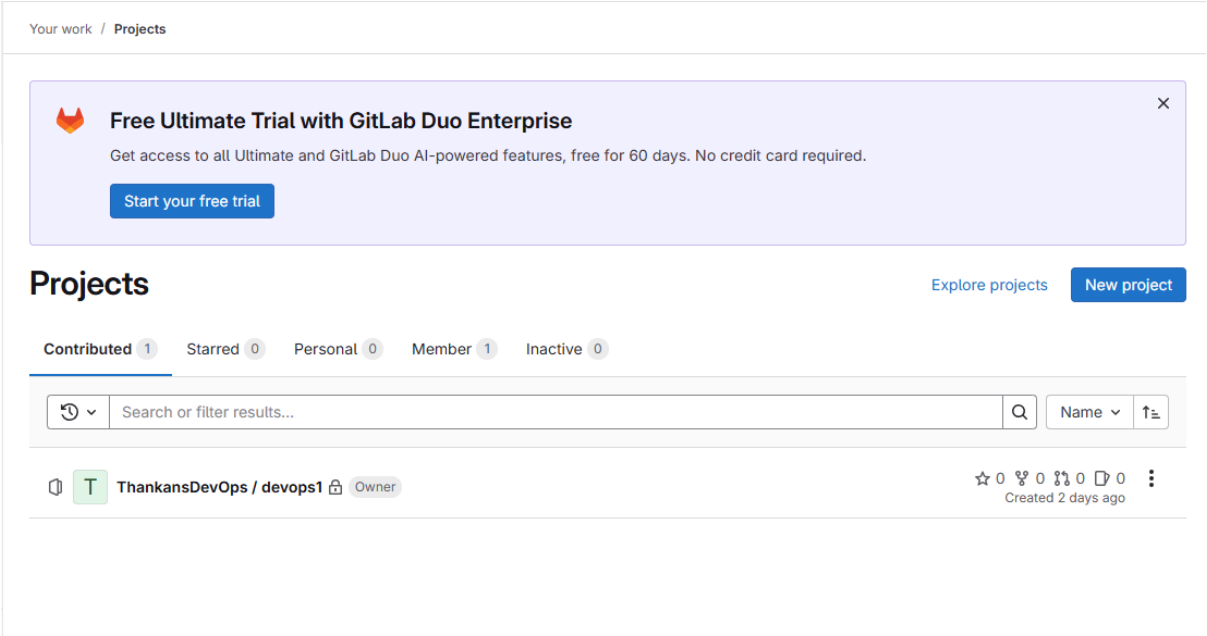
1. Gitlab Groups-

Gitlab groups are a number of different projects and repositories that are related together.
They can be used to manage these projects and its users.



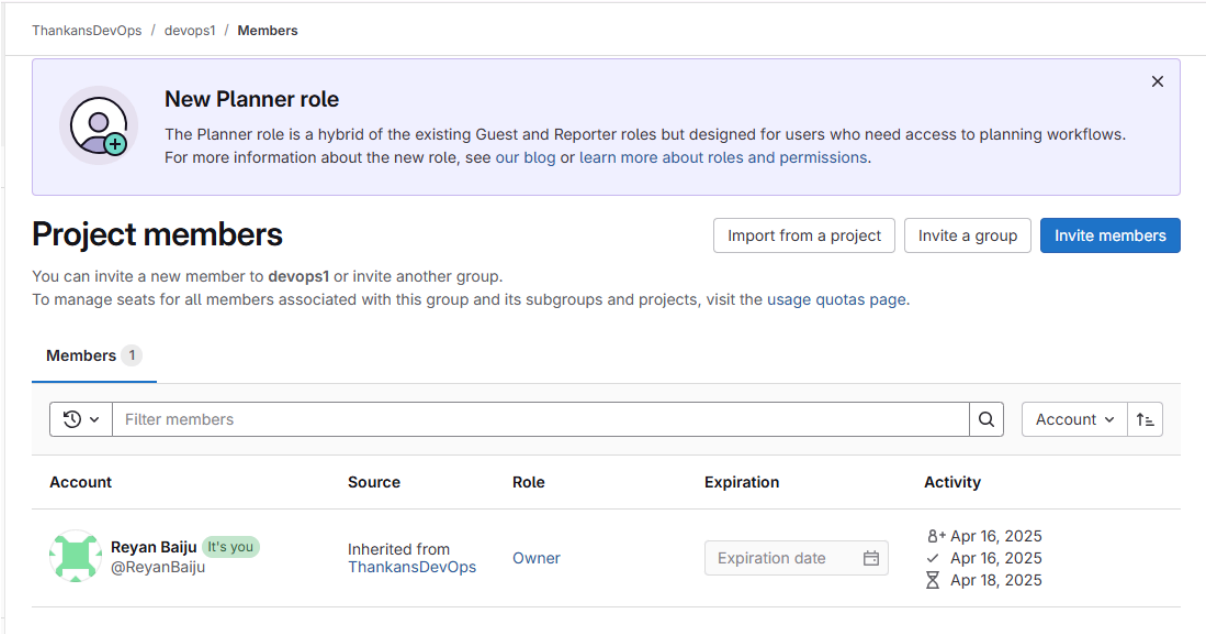
2. Gitlab project-

A gitlab project is a hub that serves as a complete environment where your team can store code, do CI/CD, check for issues etc. It is like a github repository.



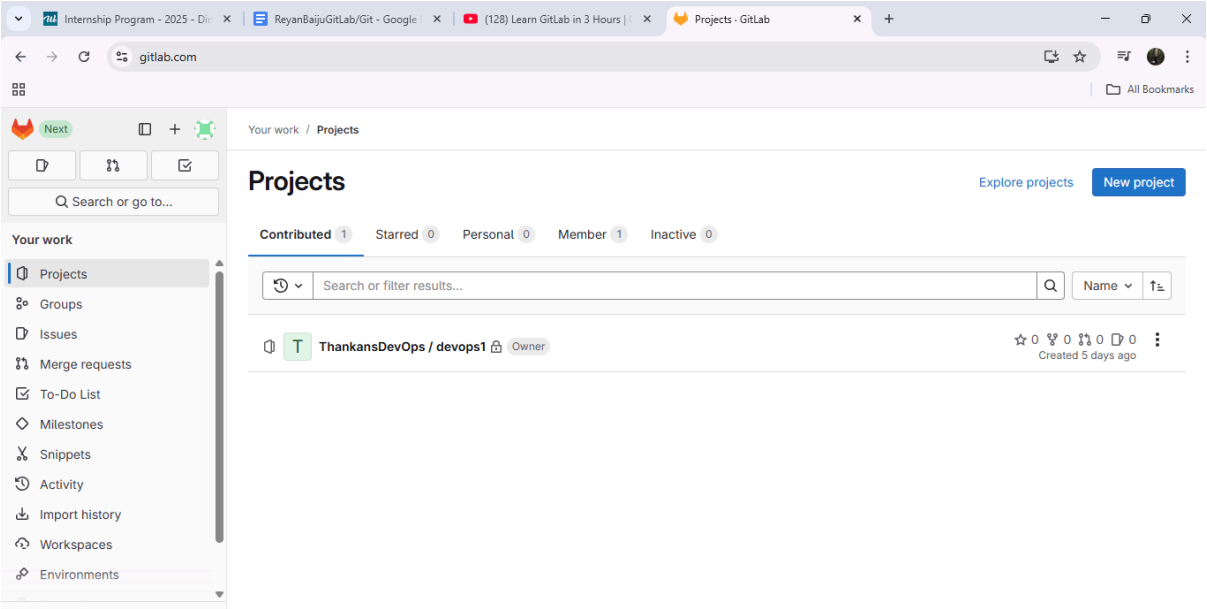
3. Gitlab Members

Gitlab members are users that have access to a gitlab project, we can specify the roles and what kind of access is allowed for the member.

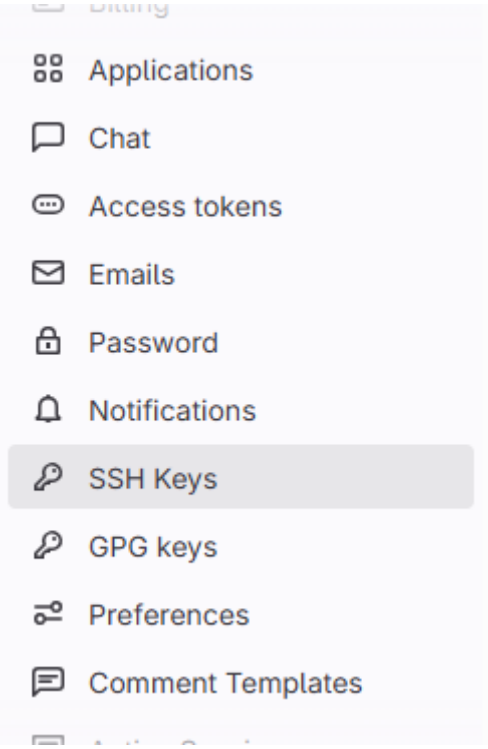


USING GITLAB-

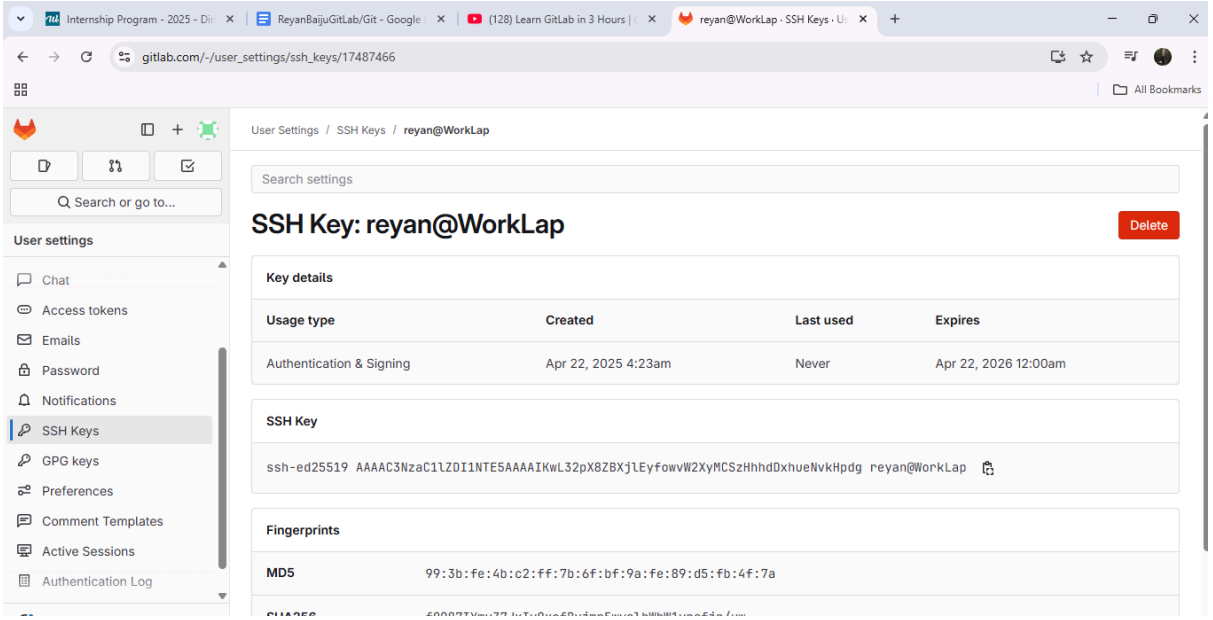
The default homepage of gitlab is-



You can go to settings to set your preferred authentication method. There are personal access tokens and also ssh keys.



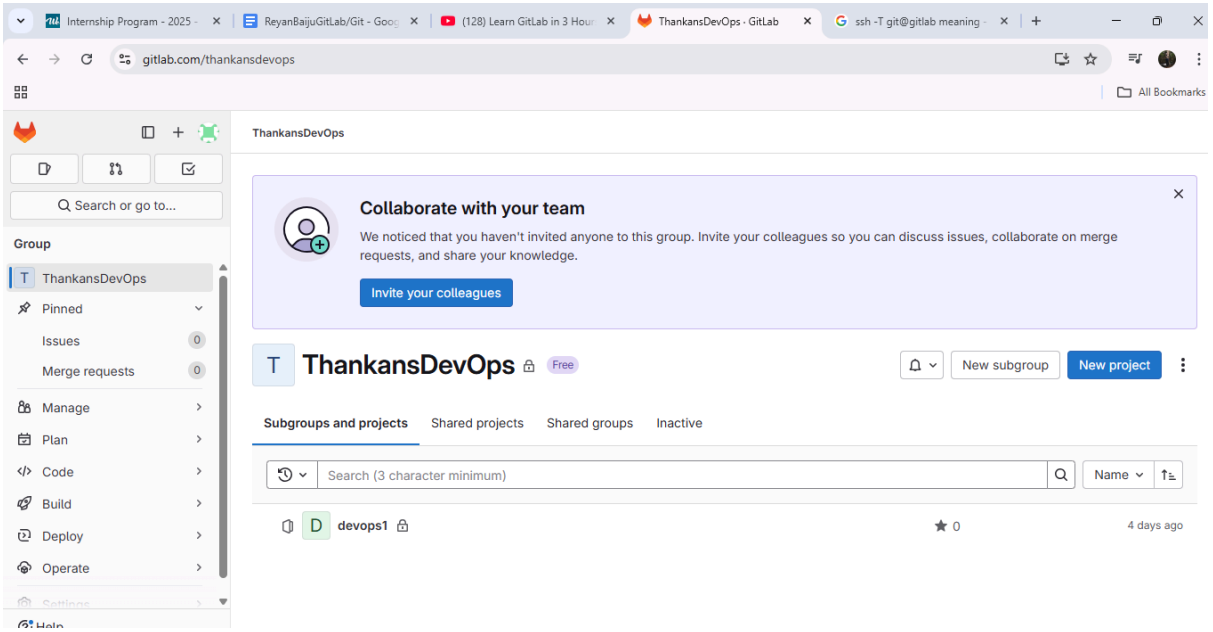
I have added SSH key to my account.



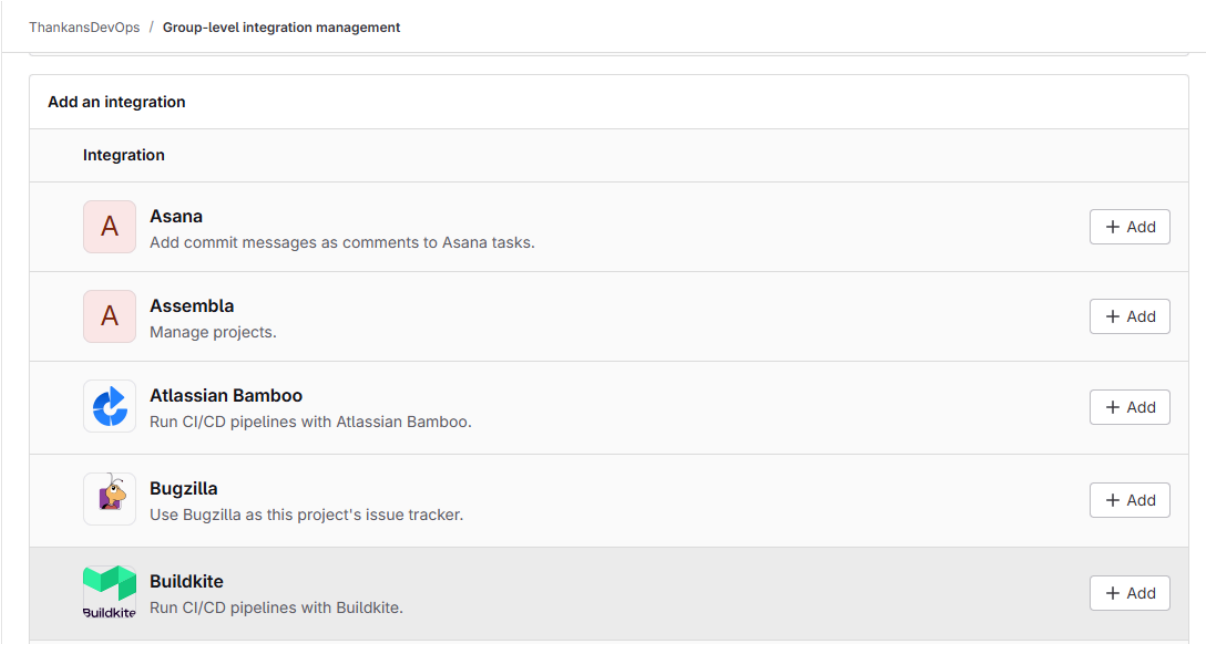
To check if it worked, use the command “ssh -T git@gitlab.com”

```
C:\Users\reyan>ssh -T git@gitlab.com
The authenticity of host 'gitlab.com (172.65.251.78)' can't be established.
ED25519 key fingerprint is SHA256:eUXGGm1YGsMAS7vkcx6JOJdOGHPem5gQp4taiCfCLB8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'gitlab.com' (ED25519) to the list of known hosts.
Welcome to GitLab, @ReyanBaiju!
```

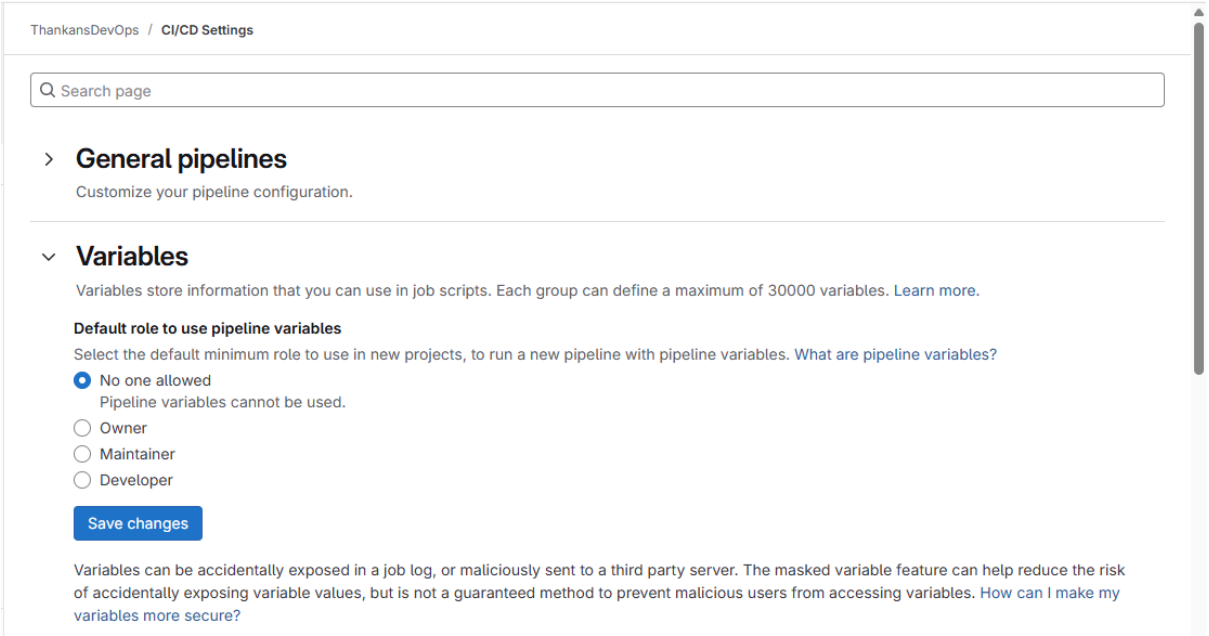
You can create Groups and integrate various tools like jira, atlassian bamboo etc.



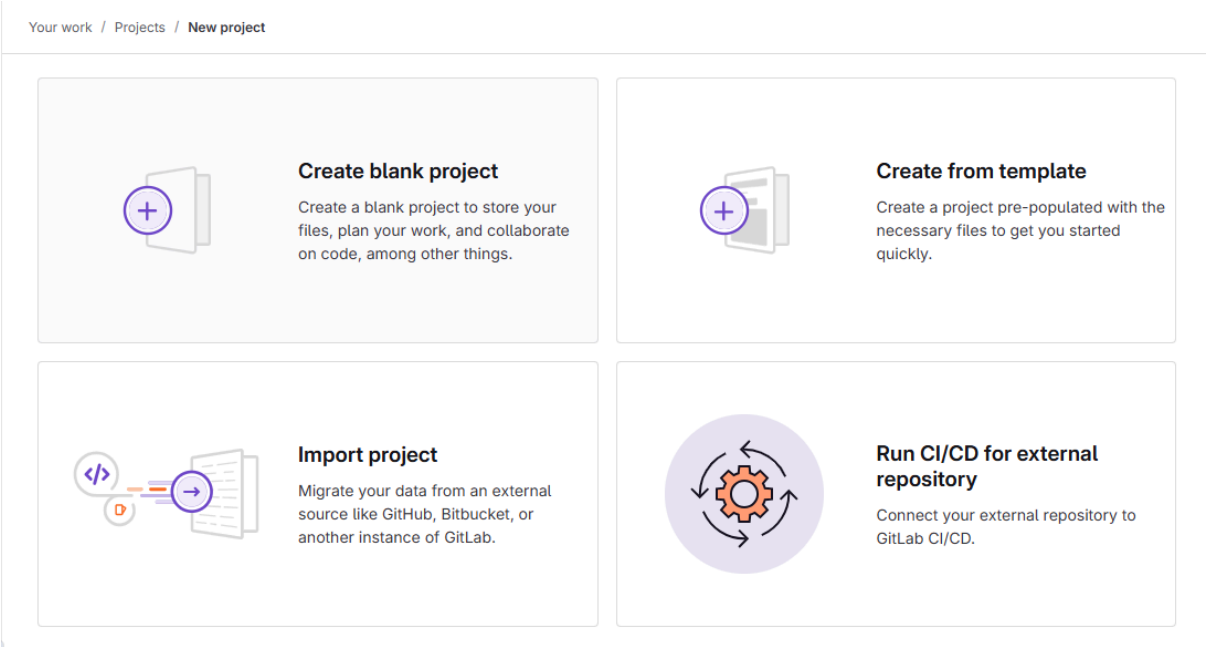
Integrations page-



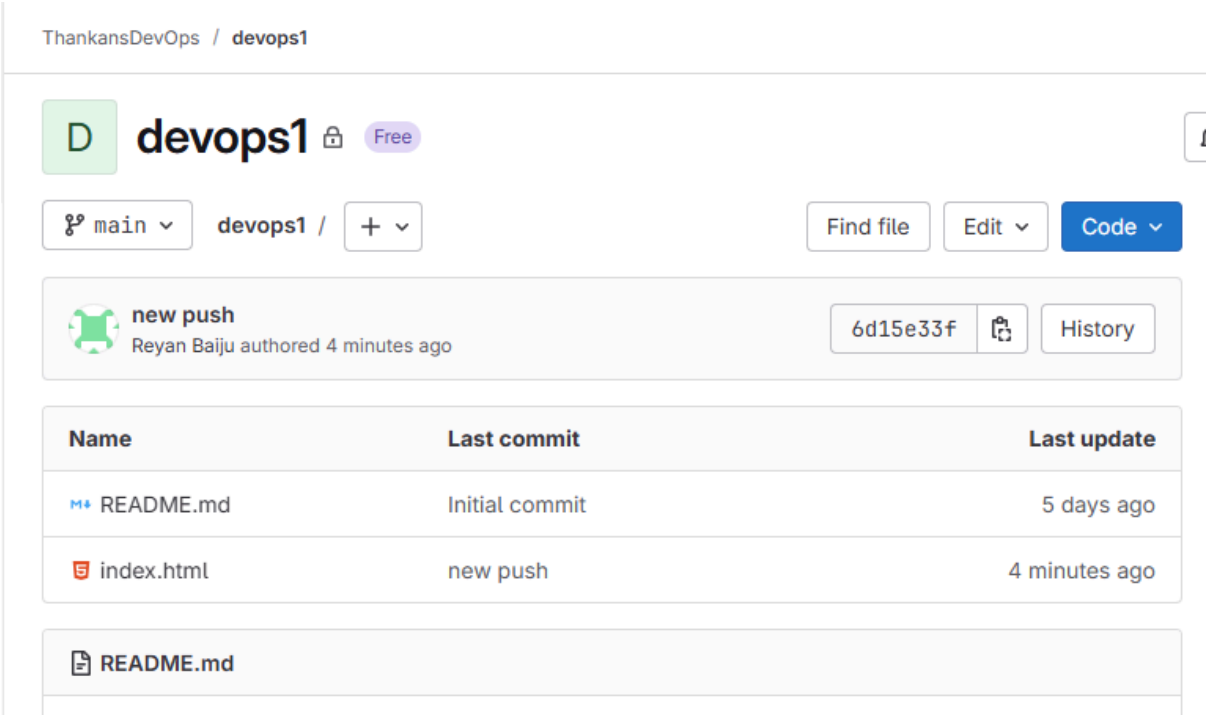
You can also added group level CI/CD variables-



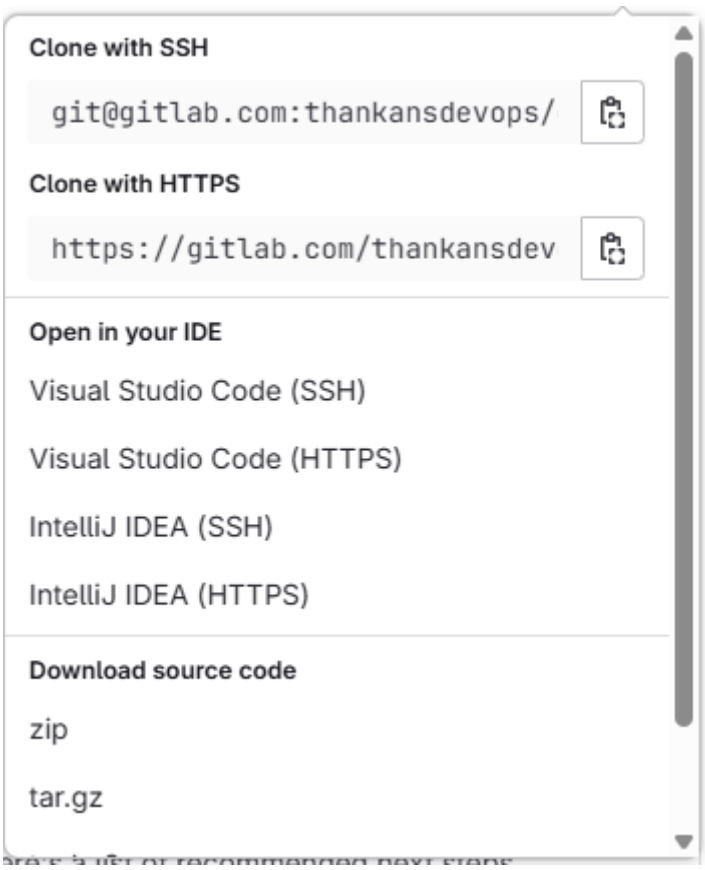
TIP- WE CAN USE GITLAB LIKE A CI/CD AGENT LIKE JENKINS WITH AN EXTERNAL REPOSITORY



After creating a project, you can use the repository. I created few files-



Im cloning it using http and adding it in my git bash.




```
reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ git remote add prac https://gitlab.com/thankansdevops/devops1.git

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ git remote -v
prac    https://gitlab.com/thankansdevops/devops1.git (fetch)
prac    https://gitlab.com/thankansdevops/devops1.git (push)

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ ls
devops1/

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ cd devops1
l
reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ ls
README.md  index.html
```

Editing and pushing the file.

```
reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ nano index.html

reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ git add .

reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html


reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ git push
Everything up-to-date

reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ git commit -m "new push"
[main 6d15e33] new push
 1 file changed, 2 insertions(+), 2 deletions(-)

reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 53.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
To gitlab.com:thankansdevops/devops1.git
   8b384fb..6d15e33  main -> main
```

The commit is successful-

Name	Last commit	Last update
README.md	Initial commit	5 days ago
index.html	new push	59 seconds ago

Imp- IF IT IS PRIVATE REPO AND YOU USE SSH KEYS TO AUTHENTICATE, REMEMBER TO PLACE THE PRIVATE KEY IN THE .SSH FOLDER.

ERROR-

```
reyan@WorkLap MINGW64 ~/Documents/new folder
$ git clone git@gitlab.com:thankansdevops/devops1.git
Cloning into 'devops1'...
git@gitlab.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

OTHER CAPABILITIES-

You also have container and package registry-

- Releases
- Feature flags
- Package registry
- Container registry
- Model registry
- Pages

Container registry-

There are no container images stored for this project

With the container registry, every project can have its own space to store its Docker images. More Information

CLI Commands

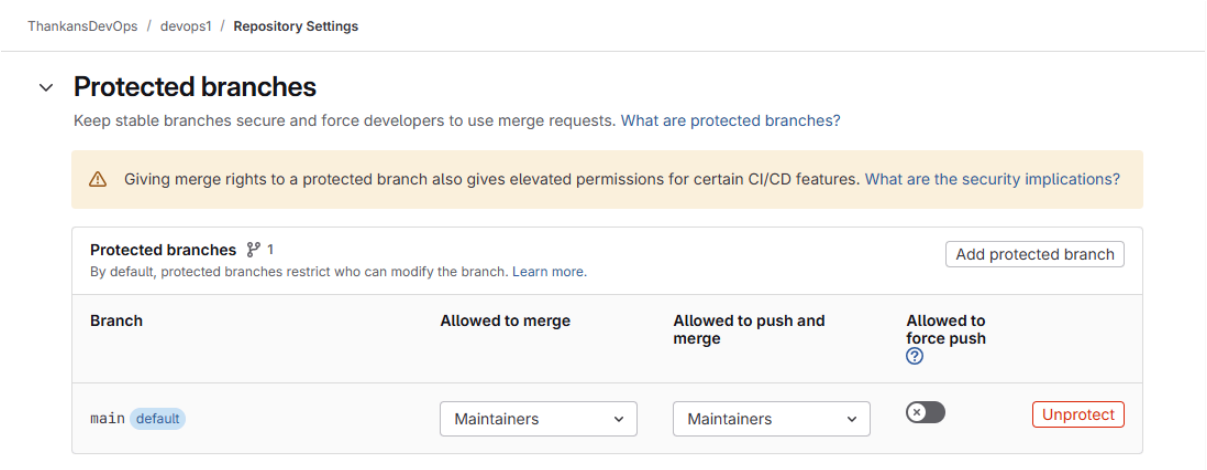
If you are not already logged in, you need to authenticate to the container registry by using your GitLab username and password. If you have Two-Factor Authentication enabled, use a personal access token instead of a password.

```
docker login registry.gitlab.com
```

You can add an image to this registry with the following commands:

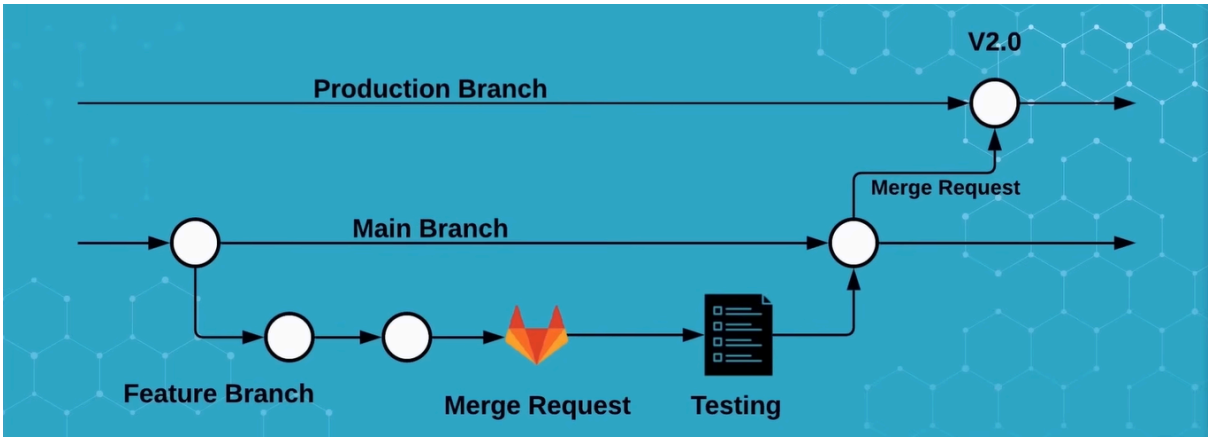
```
docker build -t registry.gitlab.com/thankansdevops/devops1
```

Protected branches-
Usually reserved for production branch-



GITLAB FLOW-

This is the most common gitlab flow, where a production branch is only updated after a feature branch is worked on.



GITLAB CI/CD

Important terminology-

Gitlab runners are open source programs that run on your vm, cloud etc that execute instructions in your ci/cd.

To install gitlab runner in a system, use the commands-

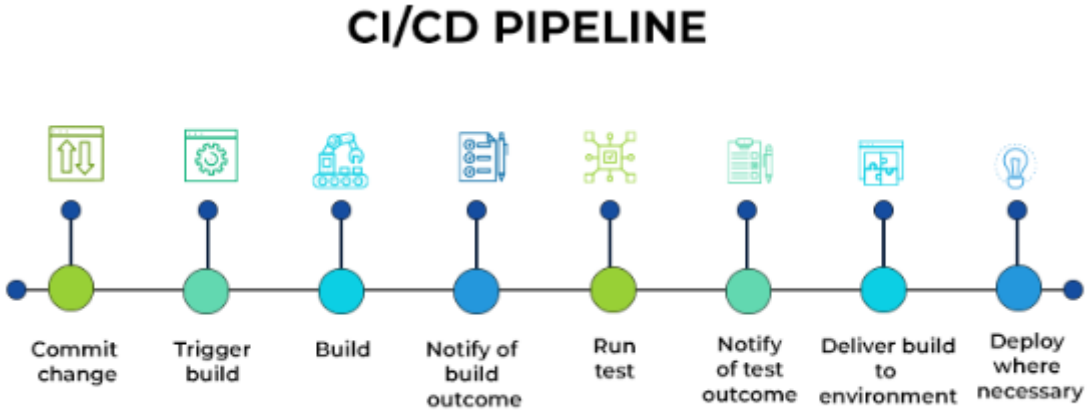
curl -L
"https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash

sudo apt install gitlab-runner

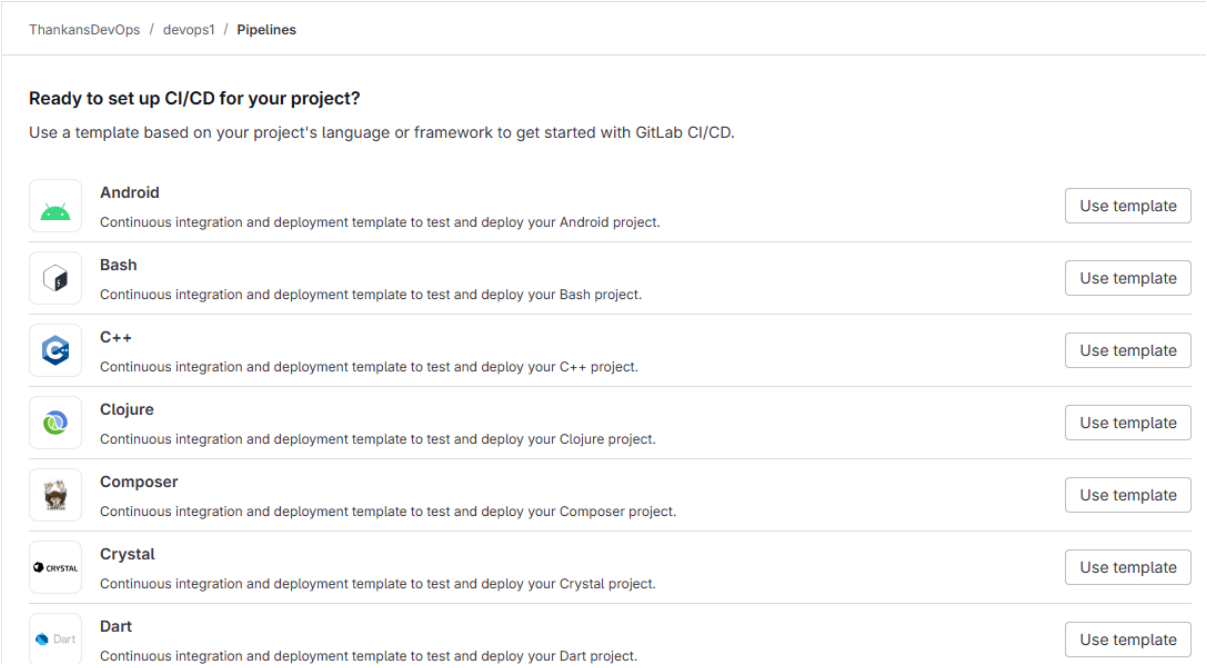
After installing a runner in your machine, you can register it in gitlab.
The details are in this documentation site-
https://docs.gitlab.com/runner/register/

SETTING UP CI/CD

Basic CI/CD steps-



To carry out a CI/CD job, you have to define a .gitlab-ci.yml file in the root directory.
You can either manually create a .yml file for ci/cd or go to the dedicated gitlab ci/cd pipeline section and select.



This is a sample CI/CD configuration file-

stages: *# List of stages for jobs, and their order of execution*

- build
- test
- deploy

build-job: *# This job runs in the build stage, which runs first.*

stage: build

script:

- echo "Compiling the code..."
- echo "Compile complete."

unit-test-job: *# This job runs in the test stage.*

stage: test *# It only starts when the job in the build stage completes successfully.*

script:

- echo "Running unit tests... This will take about 60 seconds."
- sleep 60
- echo "Code coverage is 90%"

lint-test-job: *# This job also runs in the test stage.*

stage: test *# It can run at the same time as unit-test-job (in parallel).*

script:

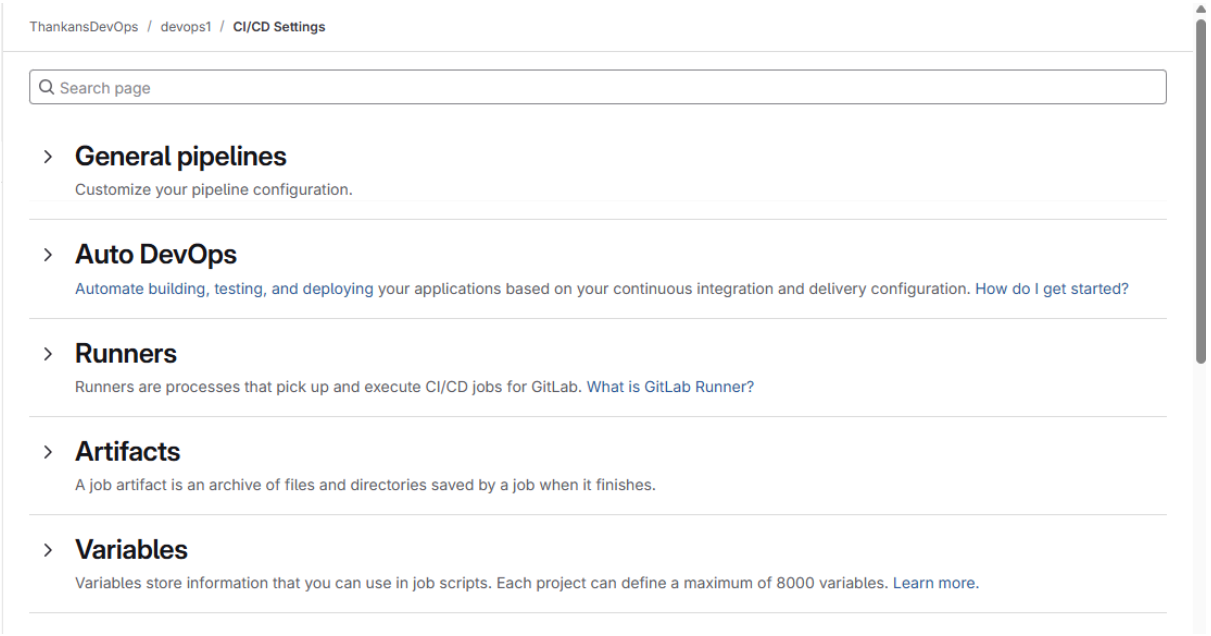
- echo "Linting code... This will take about 10 seconds."
- sleep 10
- echo "No lint issues found."

deploy-job: *# This job runs in the deploy stage.*
stage: deploy *# It only runs when *both* jobs in the test stage complete successfully.*
environment: production
script:
- echo "Deploying application..."
- echo "Application successfully deployed."

Example Docker CI/CD-

```yaml
docker-build:
  # Use the official docker image.
  image: docker:cli
  stage: build
  services:
    - docker:dind
  variables:
    DOCKER_IMAGE_NAME: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  # All branches are tagged with $DOCKER_IMAGE_NAME (defaults to commit ref slug)
  # Default branch is also tagged with `latest`
  script:
    - docker build --pull -t "$DOCKER_IMAGE_NAME" .
    - docker push "$DOCKER_IMAGE_NAME"
    - |
      if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
        docker tag "$DOCKER_IMAGE_NAME" "$CI_REGISTRY_IMAGE:latest"
        docker push "$CI_REGISTRY_IMAGE:latest"
      fi
  # Run this job in a branch where a Dockerfile exists
  rules:
    - if: $CI_COMMIT_BRANCH
      exists:
        - Dockerfile
```

You can access CI/CD settings and configure it-



Imp- Don't hard code your credentials into the CI/CD configuration, use variables of the protected kind.

In the CI/CD code, this part of the code is responsible for checking if any new code is pushed to the branch to trigger the pipeline.

```
rules:
  - if: $CI_COMMIT_BRANCH
    exists:
      - Dockerfile
```

To learn about creating docker CI/CD, go to this link-
https://www.youtube.com/watch?v=qP8kir2GUgo

GIT

Reference- https://git-scm.com/docs
https://www.atlassian.com/git/tutorials/setting-up-a-repository

Git is a distributed version control system (VCS) used to track changes in files, allowing developers to collaborate efficiently and manage different versions of code.

A VCS tracks and records changes to any file (or a group of files) allowing you to recall specific iterations later on or as needed. VCSs are sometimes called source code management (SCM) or revision control systems (RCS).

Version control allows numerous team members to work collaboratively on a project, even if they're not in the same room or even country.

To install GIT, go to the https://git-scm.com/downloads page, and select your os.

After installing GIT, you can open the GIT Bash terminal and start using GIT.

```
MINGW64:/c/Users/reyan

reyan@WorkLap MINGW64 ~
$ git --version
git version 2.49.0.windows.1

reyan@WorkLap MINGW64 ~
$ |
```

Most important GIT commands-

1. **Git init**- initialize current directory as the git repository.

Git init <repo-name>- Create a subdirectory in the same directory that is initialized as the git repository.

```
reyan@WorkLap MINGW64 ~/Documents/newgit
$ git init learn
Initialized empty Git repository in C:/Users/reyan/Documents/newgit/learn/.git/

reyan@WorkLap MINGW64 ~/Documents/newgit
$ ls
learn/
```

2. **Git config**- Git config is used to set configuration values of Git.

The main 3 flags that are used are- –local (only configured for current repository), –global (configured for user in that OS only),

Ex-

1. Git config –global init.defaultBranch main- Used to set the default branch as main.
2. git config --global user.email "reyanekbaiju@gmail.com"
3. Git config --global user.name "reyane"

3. **Git Branch**- Used to create a new branch

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (master)
$ git branch -m main

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$
```

We can see all the branches by typing git branch.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git branch testing

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git branch
* main
  testing

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$
```

To delete a branch, use the command git branch -d

4. **Git checkout**- Used to switch to a branch.

Ex- git checkout -b <branchname> to create a new branch and switch to that.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git checkout -b main
Switched to a new branch 'main'

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$
```

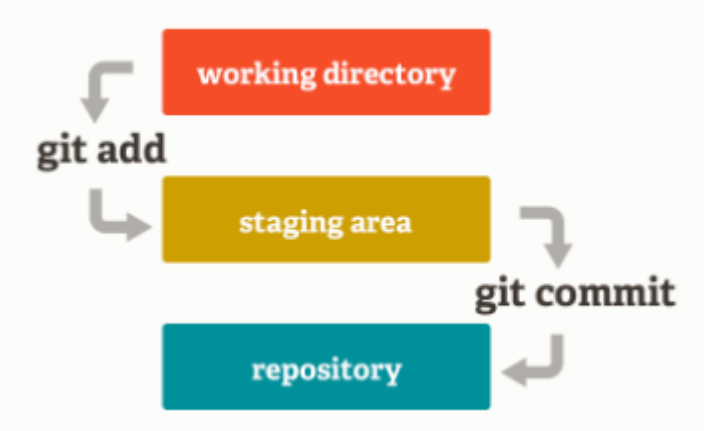
```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git checkout testing
Switched to branch 'testing'
```

IMP- When we switch to a new branch, the changes that we make in that branch stay in that branch only. To merge the changes to the main branch, we can use the git merge command.

5. **Git status**- Used to check the current status of the current directory. Shows all the changes in all the files in the directory.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$
```

6. **Git add**- Git has 2 stage commit to track changes in files. We have to add files from the working directory to the staging area.



We can use
git add <files/directories>" to add files to the staging area.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git add index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
```

To unstage files- use the command- "git rm --cached <file>"

7. **Git commit**- Used to commit changes that we are making. If we just type git commit, it will bring up the editor to type in a commit message.

We can use the git commit -m “message here”.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git commit -m "first commit"
[main (root-commit) 3bed604] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 index.html

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ |
```

When committing for the first time, GIT will ask for user metadata, like this-

git config --global user.email "reyanekbaiju@gmail.com"

git config --global user.name "reyane"

8. **Git log**- Shows history of git commits-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git log
commit 3bed60446c64dbfc4ffb46c8207f3bb3b646dd66 (HEAD -> main)
Author: reyane <reyanekbaiju@gmail.com>
Date:   Fri Apr 18 16:20:44 2025 +0530

    first commit

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$
```

Use the flag –all to see changes to all branches, –oneline to show one line summary.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git log --all
commit b84b43678628466fd8b2c8db56540b03941d28cf (HEAD -> main, testing)
Author: reyane <reyanekbaiju@gmail.com>
Date:   Fri Apr 18 16:49:16 2025 +0530

    edit text.html

commit bbc337024f7fa5d85d583f9a66101132693bdab7
Author: reyane <reyanekbaiju@gmail.com>
Date:   Fri Apr 18 16:48:00 2025 +0530

    edit index.html

commit 3bed60446c64dbfc4ffb46c8207f3bb3b646dd66
Author: reyane <reyanekbaiju@gmail.com>
Date:   Fri Apr 18 16:20:44 2025 +0530

    first commit

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git log --all --oneline
b84b436 (HEAD -> main, testing) edit text.html
bbc3370 edit index.html
3bed604 first commit

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ |
```

9. **Git merge**- To use git merge, we must change to the target branch(master or main), and then use git merge <source branch>

```
MINGW64:/c/Users/reyan/Documents/newgit/learn

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git checkout main
Switched to branch 'main'

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git status
On branch main
nothing to commit, working tree clean

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ |
```

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git merge testing
Updating 3bed604..b84b436
Fast-forward
 index.html | 2 +-
 text.html  | 1 +
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 text.html

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ |
```

10. **Git switch**- To switch to a new branch. Basically, a newer version of checkout.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git switch testing
Switched to branch 'testing'

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git status
On branch testing
nothing to commit, working tree clean
```

11. **Git rebase**- Like merge, used to merge source branch to target branch. Rebase will present conflicts one commit at a time, whereas merge will present them all at once. It's easier to handle the conflicts, but reverting a rebase is more difficult than reverting a merge if there are many conflicts.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (main)
$ git rebase testing
Successfully rebased and updated refs/heads/main.
```

In rebase, Git takes the latest version of main, then re-applies all the changes from testing one by one on top of it.

So it's like testing is rebuilt starting from main.

Always checkout to the testing branch before doing rebase.

"git checkout testing"

"git rebase main"

12. **Git Stash**- Take all the changes and stash it outside of the git repository history.

To add to the stash, use git stash push

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git stash push
Saved working directory and index state WIP on testing: c49521c checking rebase
```

```
reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git stash
Saved working directory and index state WIP on testing: c49521c checking rebase

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ ls
index.html  text.html

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git stash ls
fatal: subcommand wasn't specified; 'push' can't be assumed due to unexpected token 'ls'

reyan@WorkLap MINGW64 ~/Documents/newgit/learn (testing)
$ git stash list
stash@{0}: WIP on testing: c49521c checking rebase
```

To bring the changes to the original branch, use git stash apply or git stash pop (pop removes the changes from the list).

13. **Git clone**- Git clone is used to clone a repository into your machine. Example of a git clone command looks like this- git clone https://github.com/ReyaneBaiju/littleproject.git

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb
$ git clone https://github.com/ReyaneBaiju/littleproject.git
Cloning into 'littleproject'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 35 (delta 15), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (35/35), 11.41 KiB | 556.00 KiB/s, done.
Resolving deltas: 100% (15/15), done.

reyan@WorkLap MINGW64 ~/Documents/newgit/githb
$
```

14. **Git fetch**- Git fetch can be used to download commits, files, and refs from a remote repository into your local repo. It can fetch branches and files from the same repo or different repo.

Different Repo example-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ git fetch https://github.com/ReyaneBaiju/jenkins.git
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (50/50), done.
remote: Total 66 (delta 13), reused 16 (delta 5), pack-reused 0 (from 0)
Unpacking objects: 100% (66/66), 17.93 KiB | 41.00 KiB/s, done.
From https://github.com/ReyaneBaiju/jenkins
 * branch            HEAD       -> FETCH_HEAD
```

You can access the fetched files by using the command git checkout FETCH_HEAD for this example.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ git checkout FETCH_HEAD
Note: switching to 'FETCH_HEAD'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at e96d85f Update index.html

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject ((e96d85f...))
$ ls
README.md  default.conf  dockerfile  index.html

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject ((e96d85f...))
$ |
```

SAME REPO-

After you add your remote repository, you can fetch another branch.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ git fetch reyan
From https://github.com/ReyaneBaiju/jenkins
 * [new branch]      main       -> reyan/main

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ ls
Dockerfile  Jenkinsfile  app.js  package.json

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ git branch
* main

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject (main)
$ git checkout reyan/main
Note: switching to 'reyan/main'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at e96d85f Update index.html

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/littleproject ((e96d85f...))
$ ls
README.md  default.conf  dockerfile  index.html
```

HOW TO USE GIT FETCH

First ,we can add our remote repository and fetch the branch that we want. It will be available in FETCH_HEAD.

We can now switch or checkout to that branch and edit the files.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ git checkout FETCH_HEAD
HEAD is now at d662825 testing only

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ ls
Dockerfile  Jenkinsfile  app.js  package.json

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ git status
HEAD detached at FETCH_HEAD
nothing to commit, working tree clean

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ cat app.js
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => {
    res.send('God i hope this works for git push');
});

app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ nano app.js

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ git add .

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ git status
HEAD detached at FETCH_HEAD
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   app.js


reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((d662825...))
$ git commit -m "simoe"
[detached HEAD 19695ae] simoe
 1 file changed, 1 insertion(+), 1 deletion(-)
```

After adding and committing, we can go back to the main branch and merge it-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little ((19695ae...))
$ git checkout main
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

  19695ae simoe

If you want to keep it by creating a new branch, this may be a good time
to do so with:

 git branch <new-branch-name> 19695ae

Switched to branch 'main'
Your branch is up to date with 'test/main'.
```

We will get the code- use it-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git merge 19695ae
Updating d662825..19695ae
Fast-forward
 app.js | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ |
```

15. **Git Pull**- Git pull is a combination of the git fetch and git merge command.

The command example- git pull <remote> <branch>

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/jenkins/littleproject (main)
$ git pull reyan main
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (50/50), done.
remote: Total 66 (delta 13), reused 16 (delta 5), pack-reused 0 (from 0)
Unpacking objects: 100% (66/66), 17.93 KiB | 48.00 KiB/s, done.
From https://github.com/ReyaneBaiju/jenkins
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> reyan/main
fatal: refusing to merge unrelated histories
```

Git pull example-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little
$ git init
Initialized empty Git repository in C:/Users/reyan/Documents/newgit/githb/little/.git/

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git remote add test https://github.com/ReyaneBaiju/littleproject.git

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ ls

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git remote -v
test    https://github.com/ReyaneBaiju/littleproject.git (fetch)
test    https://github.com/ReyaneBaiju/littleproject.git (push)

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git pull test main
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 35 (delta 15), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (35/35), 11.39 KiB | 60.00 KiB/s, done.
From https://github.com/ReyaneBaiju/littleproject
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> test/main

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ ls
Dockerfile  Jenkinsfile  app.js  package.json

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ |
```

Merging example-

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/jenkins/littleproject (main)
$ ls
Jenkinsfile  README.md  app.js  default.conf  dockerfile  index.html  package.json
```

`git fetch`	**`git pull`**
Downloads new information from a remote repository without merging into the current branch.	Downloads all the changes from a remote repository and merges them into the current branch.
Updates the repository data in the *.git* directory.	Updates the local repository directly.
Allows reviewing commits and changes from other developers before committing the code.	Allows bringing and updating changes to the local repository immediately.
No possibility of merge conflicts when running the command.	Possible merge conflicts which need an immediate resolution.

16. **Git Remote** - Git remote is used to manage remote repositories.

You can add a remote repository using the command git remote add <customname> <repourl>

```
reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ git remote add prac https://gitlab.com/thankansdevops/devops1.git

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ git remote -v
prac    https://gitlab.com/thankansdevops/devops1.git (fetch)
prac    https://gitlab.com/thankansdevops/devops1.git (push)

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ ls
devops1/

reyan@WorkLap MINGW64 ~/Documents/prac (main)
$ cd devops1
l
reyan@WorkLap MINGW64 ~/Documents/prac/devops1 (main)
$ ls
README.md  index.html
```

You can see the added repositories using the git remote -v command.

```
MINGW64:/c/Users/reyan/Documents/newgit/githb/jenkins/littleproject

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/jenkins/littleproject (main)
$ git remote -v
origin  https://github.com/ReyaneBaiju/littleproject.git (fetch)
origin  https://github.com/ReyaneBaiju/littleproject.git (push)
reyan   https://github.com/ReyaneBaiju/jenkins.git (fetch)
reyan   https://github.com/ReyaneBaiju/jenkins.git (push)

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/jenkins/littleproject (main)
$ |
```

17. **Git Push**- Git push is used to push changes to the remote repository.

Command- git push <remote> <branch>.

You can use the –force tag to force a push to a branch.

You can push after editing, adding the file to the staging area and committing.

```
reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git commit -m "testing only"
[main d662825] testing only
 1 file changed, 1 insertion(+), 1 deletion(-)

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ cat app.js
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => {
    res.send('God i hope this works for git push');
});

app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ git push test main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 298 bytes | 149.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects
To https://github.com/ReyaneBaiju/littleproject.git
   a4cbf6c..d662825  main -> main

reyan@WorkLap MINGW64 ~/Documents/newgit/githb/little (main)
$ |
```

18. Git Revert

Git Revert is used to undo commits to a branch. The syntax of the git revert command is git revert <commithash>