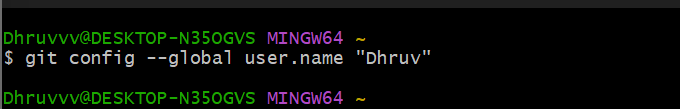
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MCAL23 DevOps Lab INDEX** | | | | |
| **Sr.No** | **Title** | **CO** | **Date** | **Sign** |
| 1 | Demonstrate basic Git commands | CO2 |  |  |
| 2 | Create and fork repositories in Git Hub. Apply branch, merge and rebase concepts. | CO2 |  |  |
| 3 | Demonstrate Git for Collabration | CO2 |  |  |
| 4 | Demonstrate Collabrating and cloning using Git | CO2 |  |  |
| 5 | Using GitLab Web IDE | CO2 |  |  |
| 6 | Demonstrate CI/CD Workflow in GitLab using .py, .bash, .java file | CO2 |  |  |
| 7 | Demonstrate setting Jenkins CI/CD pipeline. | CO3 |  |  |
| 8 | Demonstrate Setting up of a CI/CD pipeline to build and deploy a web application to a local HTTP server | CO3 |  |  |
| 9 | Demonstrate basic Docker commands | CO3 |  |  |
| 10 | Develop a simple containerized application using Docker | CO3 |  |  |
| 11 | Demonstrate add-on ansible commands | CO4 |  |  |
| 12 | Demonstrate Ansible Playbooks | CO4 |  |  |

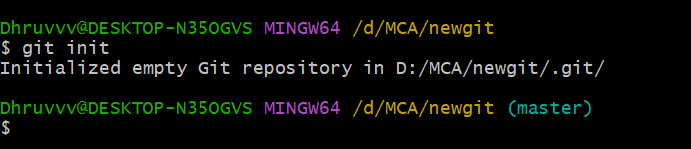
**Practical 1**

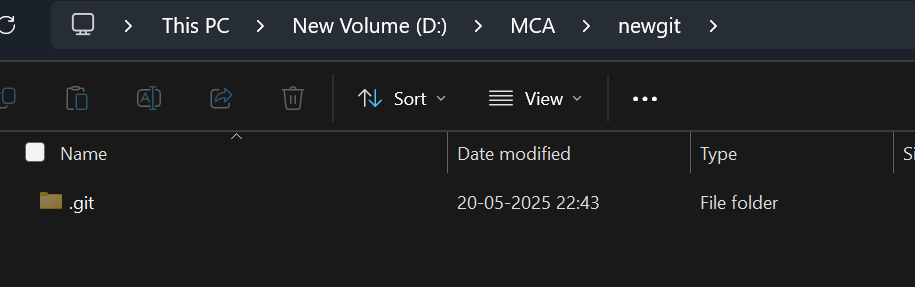
**AIM: Demonstrate basic Git commands**

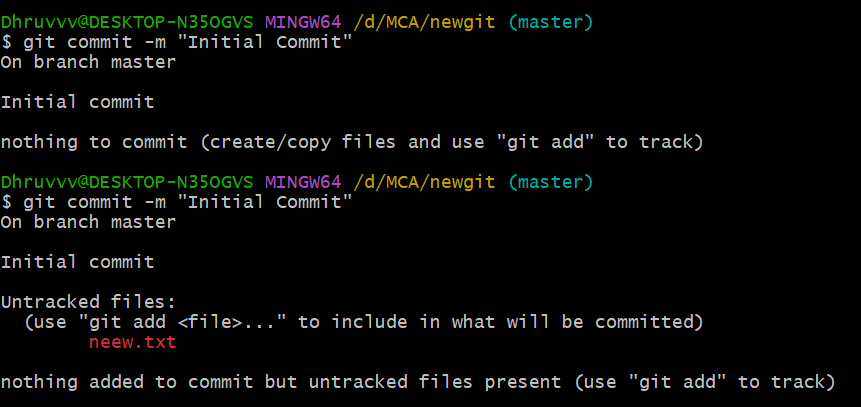
1. Set-up Git (One-time setup)
   * **git config –global user.name “Dhruv”** -- Set your name for commits.
   * **git config –global user.email “**[**dhruvpanchal547@gmail.com**](mailto:dhruvpanchal547@gmail.com)**”** -- Set your email for commits.

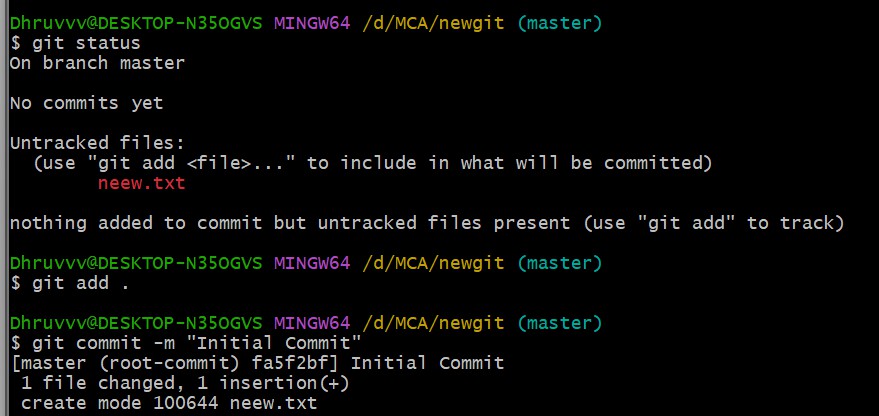


1. Start a new project
   * **git init :-** Initialize a new repository in your project folder.
   * **git status:-** Check the status of files in the working directory (untracked, modified, etc.).
   * **git add . :-** Stage all changes (new, modified, or deleted files).
   * **git commit -m “Initial Commit” :-** Commit the staged files with a message describing the changes









## Work on a New Feature

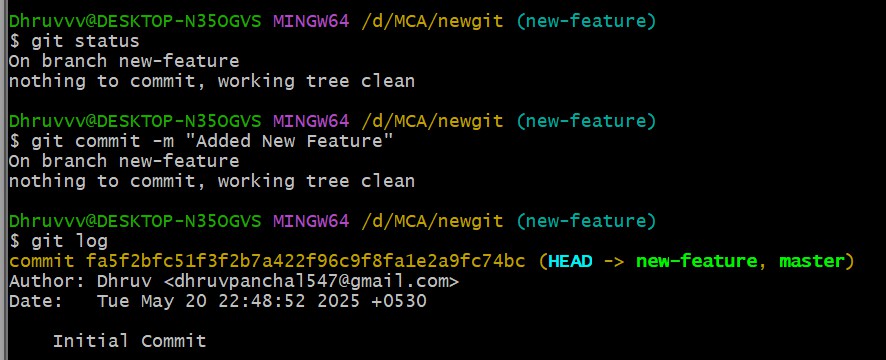
* + **git branch <branch\_name> :-** Create a new branch for a feature (e.g., feature-1).
  + **git checkout <branch\_name>** :- Switch to the new branch.



git

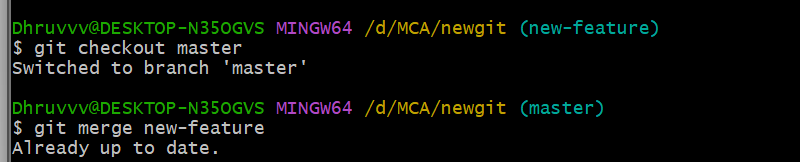
## Make Changes and Commit them

* + **git add <file>:-** Stage specific files for commit.
  + **git commit -m "Add new feature” :-** Commit changes with a descriptive message.
  + **git log :-** View the history of commits in the current branch.



## Merge changes back into main branch.

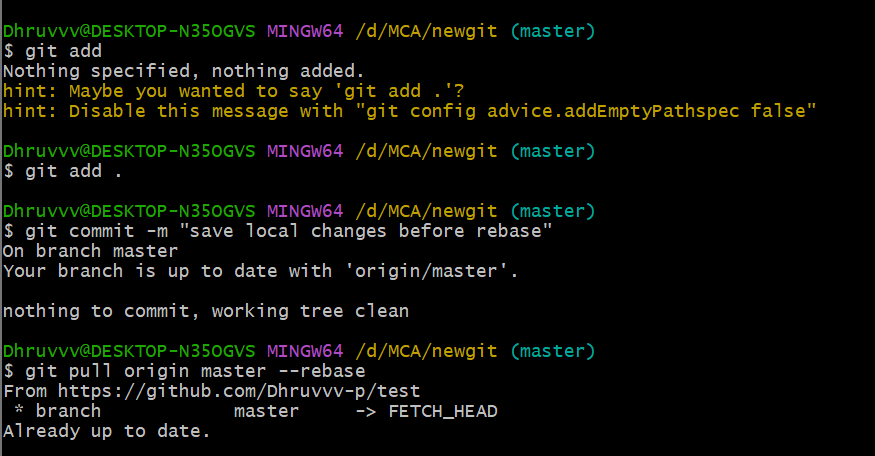
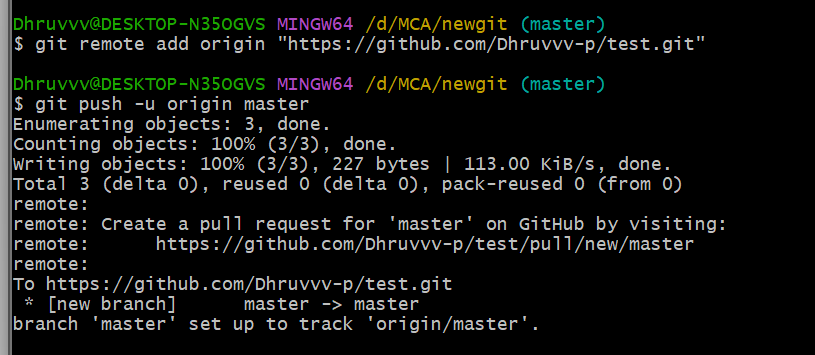
* + **git checkout main** :- Switch back to the main branch
  + **git merge <branch\_name>** :- Merge the changes from the feature branch into the main branch.



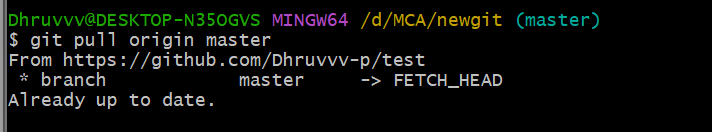
## Set up a remote repository

* + **git remote add origin <repository-url> :-** Link your local repository to a remote repository (e.g., on GitHub).

## git push -u origin main

****

1. **Collaborate with a team**
   * **git pull origin main :-** Fetch and merge changes from the remote main branch to your local branch.
   * **git branch:-** List all branches to see if new ones were created by collaborators.
   * **git fetch:-** Download updates from the remote without merging them.
   * **git diff** :- Compare changes between your working directory and the staging area, or between commits.
   * **git push** :- Push your committed changes to the remote branch



**Practical 2**

**AIM: Create and fork repositories in Git Hub. Apply branch, merge and rebase concepts.**

# Step 1: Initial Setup

## Create a Git repository (if not already created)

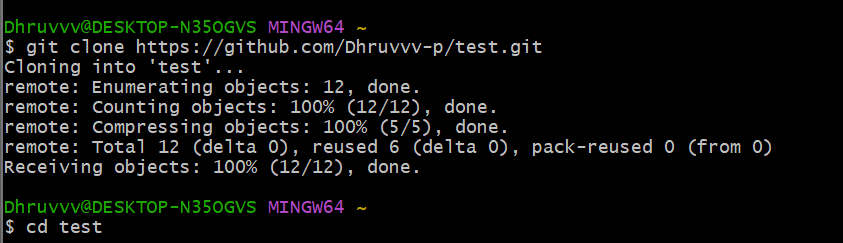
If you don’t have a repository yet, you can create one by running:

git init

## Clone an existing repository (if you're working on an existing project)

If you're working with an existing remote repository, you can clone it by running:

git clone <repository-url> cd <repository-name>



# Step 2: Working with Branches

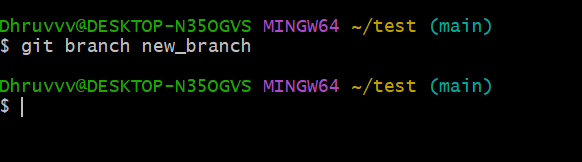
* 1. **Check the current branch** By default, Git starts with a branch named main or master. To see which branch you are currently on, use:

git branch

The current branch will be marked with an asterisk (\*).

* 1. **Create a new branch** To create a new branch, use:

git branch <branch-name>

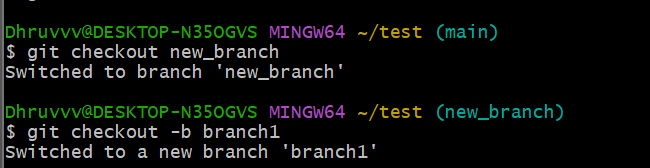


* 1. **Switch to the new branch** To start working on the new branch, use:

git checkout <branch-name>

You can also combine the creation and switch into one command:

git checkout -b <branch-name>



* 1. **View all branches** To see all branches in your repository:

git branch

The current branch will be marked with an asterisk (\*).

# Step 3: Make Changes in the Branch

## Make some changes in the code

Now that you’re on your new branch, make some changes to your files (e.g., modify code, add new features, etc.).

1. **Stage the changes** After making changes, you need to add them to the staging area:

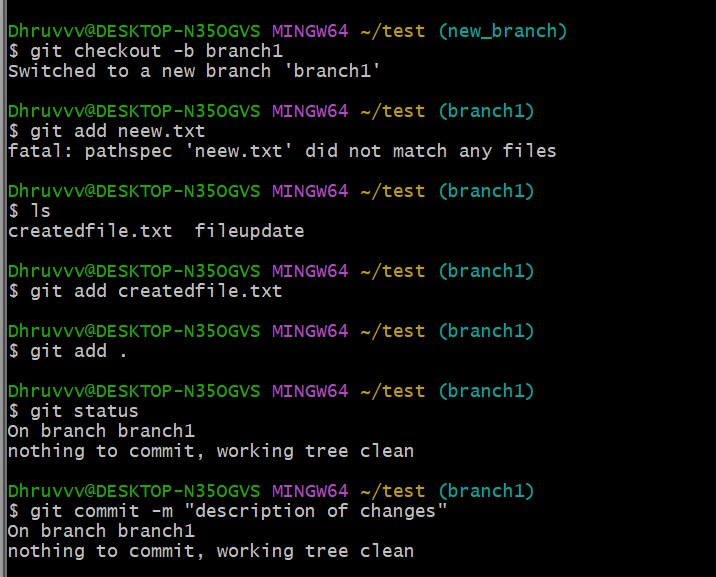
bash

git add <file-name> # Add a specific file

git add . # Add all files (recommended if you want to stage everything)

‘

1. **Commit the changes** After staging, commit the changes to your branch:



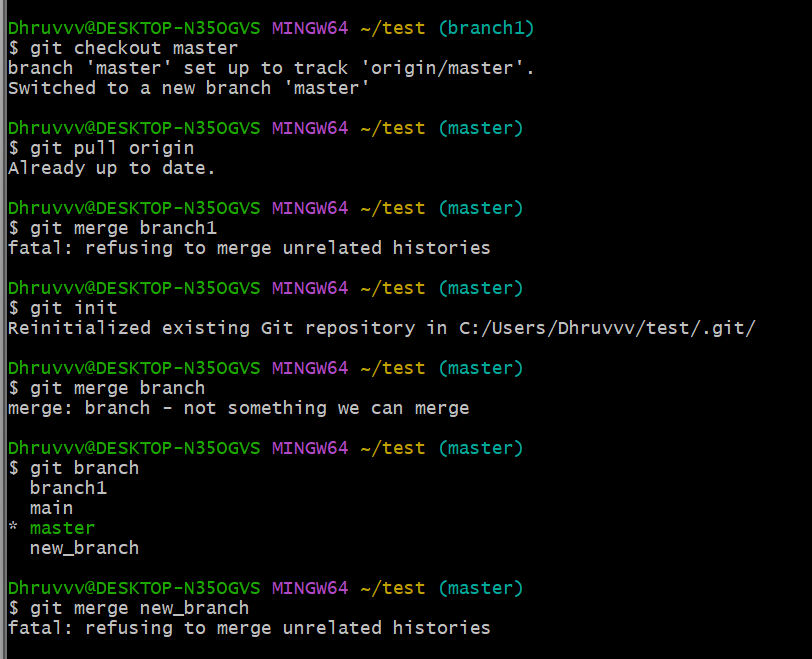
git commit -m "Description of changes"

git

# Step 4: Merge the Branch into Main

1. **Switch to the main branch (or the branch you want to merge into)** Before merging, switch back to the main branch:

git checkout main



1. **Pull the latest changes** Ensure your main branch is up to date with the remote repository:

git pull origin main

1. **Merge the feature branch into main** Now merge your branch into main: git merge <branch-name>
   * If there are no conflicts, Git will automatically complete the merge and add a merge commit.
   * If there are conflicts, Git will notify you, and you’ll need to resolve them manually.

# Step 5: Resolving Merge Conflicts (If Any)

1. **Check for conflicts** If Git encounters conflicts during the merge, it will pause and mark the conflicted files.
2. **Open the conflicted files** Conflicted sections will be marked with:

<<<<<<< HEAD

(changes from `main` branch)

=======

(changes from `<branch-name>`)

>>>>>>> <branch-name>

1. **Resolve the conflicts** Edit the file to keep the changes you want and remove the conflict markers (<<<<<<<, =======, >>>>>>>).
2. **Mark the conflicts as resolved** After resolving conflicts, stage the files as resolved:

git add <resolved-file>

1. **Complete the merge** Once all conflicts are resolved, commit the merge:

git commit

Git will automatically create a merge commit if you didn’t need to resolve conflicts manually

# Step 6: Push Changes to Remote

1. **Push the changes to the remote repository** After merging, push the changes to the remote repository:

git push origin main

This updates the remote repository with the changes from your merge.

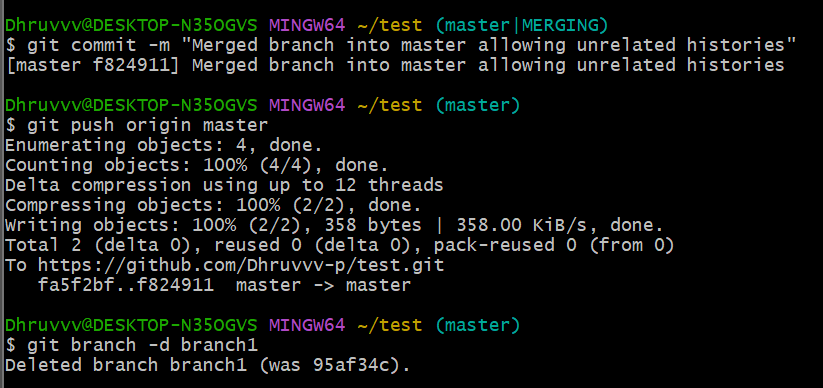
# Step 7: Clean Up (Optional)

1. **Delete the branch after merging (optional)** After merging, you can delete your feature branch if you no longer need it:

git branch -d <branch-name> # Deletes the local branch

1. **Delete the remote branch (optional)** If you want to delete the branch on the remote as well, use:

git push origin --delete <branch-name>



# Step 8: Regular Maintenance

1. **Sync your local repository with the remote regularly** To avoid conflicts, it’s good practice to frequently pull changes from the main branch into your working branch:

git checkout <branch-name> # Switch to your feature branch git pull origin main # Pull the latest changes from main

## Stay organized

* + Use descriptive branch names (e.g., feature/auth, bugfix/login).
  + Regularly merge back into main to keep your changes synchronized.

**Practical 3**

**AIM: Demonstrate Git for Collaboration**

# Set Up Git and GitHub

Before you start collaborating or cloning repositories, make sure you have the following set up:

1. **Install Git**: If you haven't already, download and install [Git](https://git-scm.com/downloads) on your machine.
2. **Create a GitHub Account**: Go to [GitHub](https://github.com/) and create an account if you don't have one.
3. **Configure Git**: Set up your Git configuration with your name and email.

git config --global user.name "Your Name"

git config --global user.email [youremail@example.com](mailto:youremail@example.com)

# Step 2: Clone a GitHub Repository

Cloning a repository allows you to create a copy of a project on your local machine, enabling you to work on it.

1. **Find a Repository to Clone**: Visit the repository page on GitHub (e.g., <https://github.com/username/repository>) and click the green **Code** button.
2. **Copy the Clone URL**: In the popup, choose either **HTTPS** or **SSH** and copy the URL. If you're using HTTPS, it will look like https://github.com/username/repository.git.

## Clone the Repository Locally:

Open a terminal on your computer and navigate to the directory where you want to clone the repository. Then run:

git clone https://github.com/username/repository.git

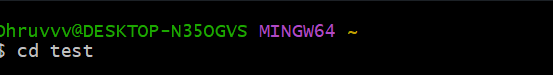
* Replace https://github.com/username/repository.git with the URL you copied.

This will create a local copy of the repository on your machine.

## Navigate to the Repository Folder:

cd repository # Navigate into the cloned directory





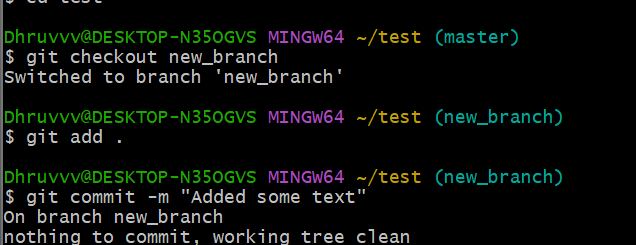
# T

**Step 3: Work on the Project Locally**

Once you've cloned the repository, you can start making changes to the code.

1. **Create a New Branch**: Before making changes, it's recommended to create a new branch. This ensures your changes don't interfere with the main codebase until you're ready to merge.

git checkout -b feature/your-feature # Create and switch to a new



1. **Make Changes**: Edit files as needed using your preferred editor or IDE.
2. **Stage Changes**: After making changes, you need to stage them before committing.

git add . # Stages all modified files

1. **Commit Changes**: Once changes are staged, commit them to your local branch.

git commit -m "Add feature X"

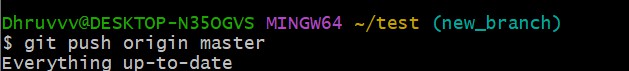
# Step 4: Push Changes to GitHub

Once you’ve committed your changes locally, you need to push them to your GitHub repository.

1. **Push Your Changes** to the remote repository:

git push origin feature/your-feature # Push the feature branch to GitHub

This uploads your local changes to your GitHub repository.



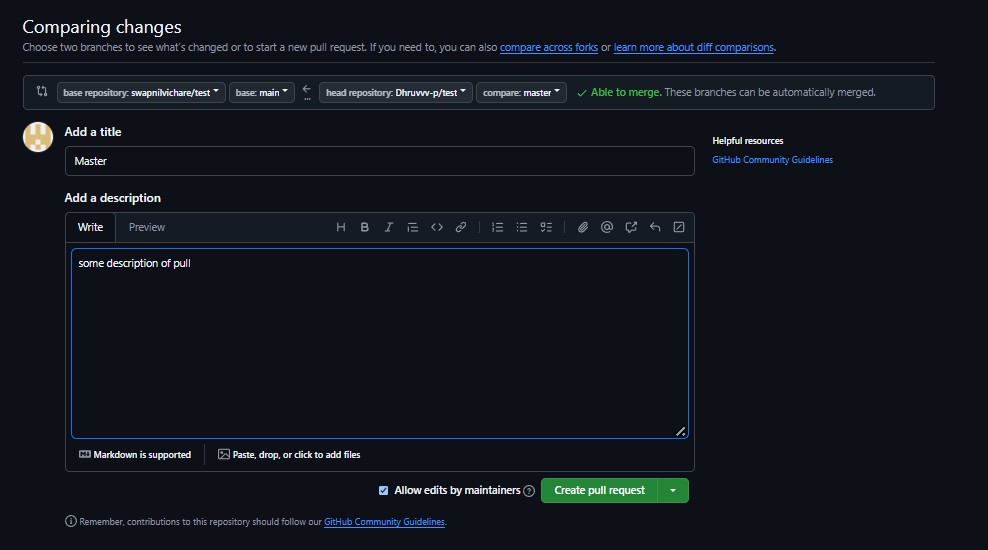
# Step 5: Create a Pull Request (PR)

To contribute your changes back to the original repository, you'll need to open a pull request (PR).

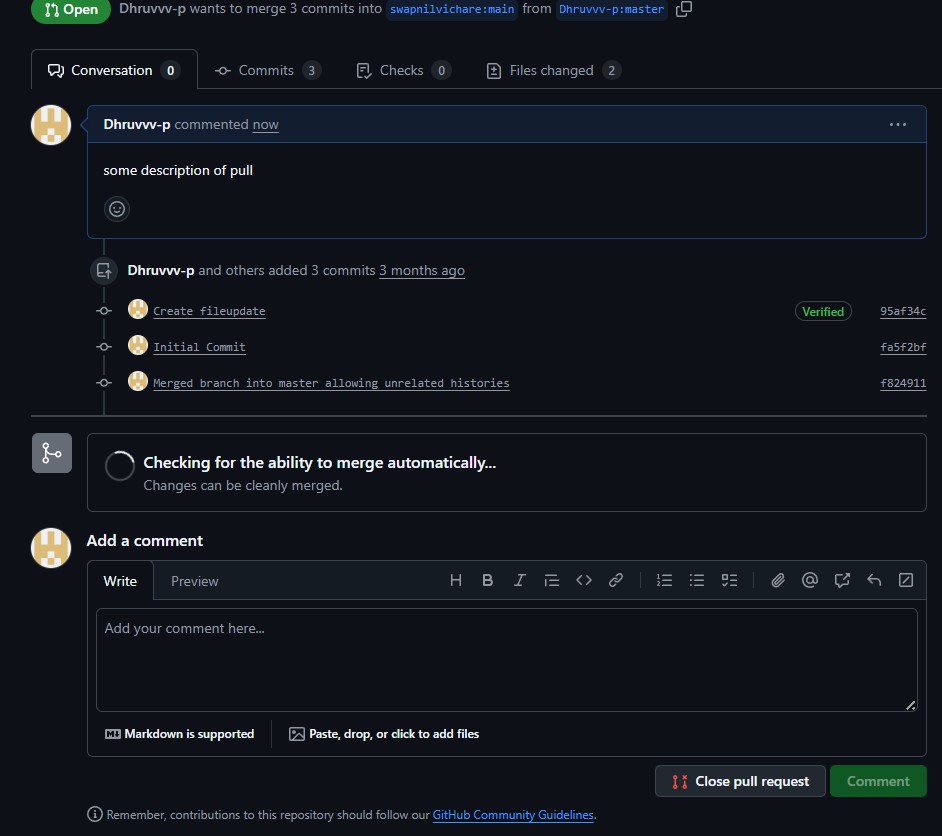
1. **Go to the GitHub Repository**: Visit the repository where you want to make the changes.
2. **Create a Pull Request**: On GitHub, you'll see an option to compare your branch (e.g., feature/your-feature) with the main branch of the repository (e.g., main). Click **New Pull Request**.

## Fill Out the Pull Request Form:

* + Add a **title** and **description** explaining the changes you’ve made.
  + Review your changes.
  + Click **Create Pull Request**.



1. **Code Review**: The repository maintainer (or other collaborators) will review your changes. They might ask for changes or approve the PR.
2. **Merge the Pull Request**: Once your changes are approved, the maintainer will merge your changes into the main branch of the project.



# Step 6: Sync Your Fork (If Working on a Forked Repo)

If you are working on a forked repository and want to keep your fork in sync with the original repository:

1. **Add the Original Repository as a Remote**: This allows you to fetch updates from the original repository.

git remote add upstream https://github.com/owner/original- repository.git

Replace https://github.com/owner/original-repository.git with the original repository's URL.

## Fetch the Latest Changes from the Original Repository:

git fetch upstream # Fetch the changes from the original repo

## Merge the Latest Changes into Your Local Branch:

git checkout main # Switch to your main branch

git merge upstream/main # Merge the latest changes from the original repo

## Push the Changes to Your Fork:

git push origin main # Push the updated main branch to your fork

# Step 7: Pull Latest Changes from the Original Repository

To keep your local repository up-to-date with the remote repository on GitHub, you can pull the latest changes.

## Switch to Your Local Main Branch:

git checkout main

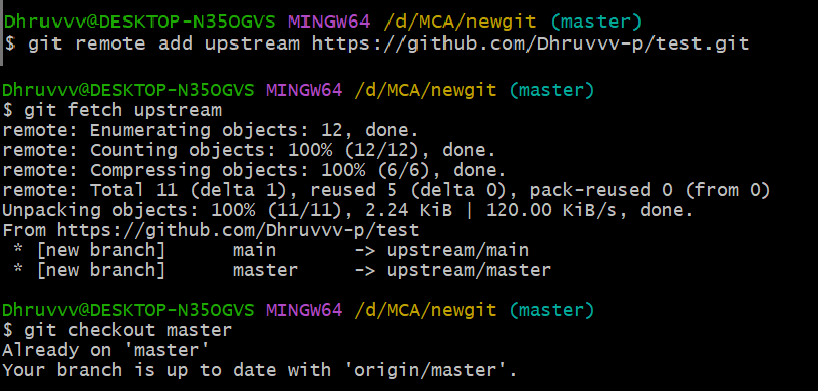
## Pull Latest Changes from GitHub:

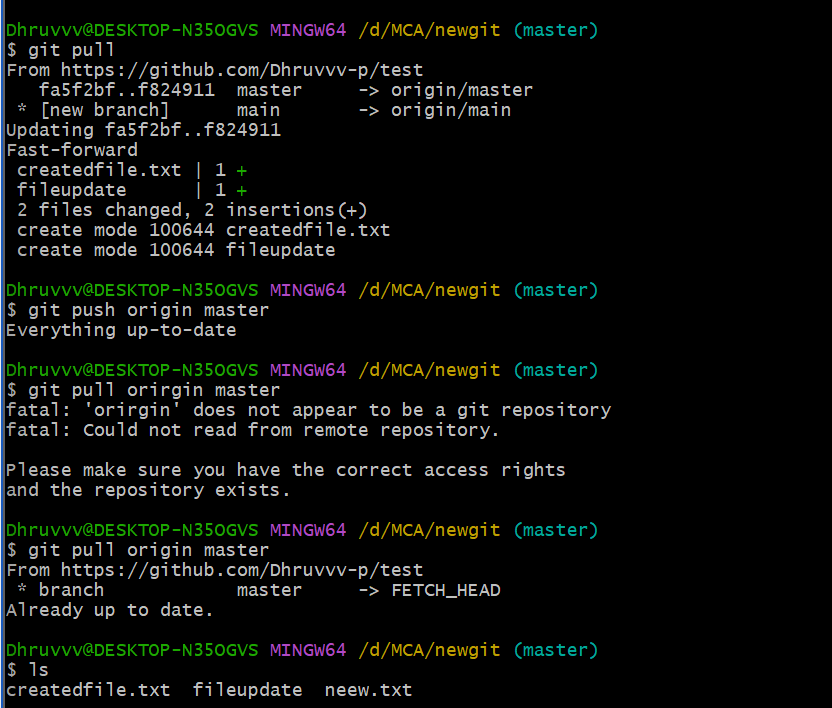
git pull origin main # Pull the latest changes from the remote repository

# Step 8: Collaborate with Other Developers

When collaborating with other developers on GitHub, you'll typically follow these best practices:

1. **Regularly Pull Latest Changes**: To ensure you're not working on outdated code, frequently pull the latest changes from the main branch (especially before you start working on new features or bug fixes).
2. **Create Feature Branches**: Always create a new branch for each feature or bug fix. This avoids conflicts and keeps the history clean.
3. **Review Pull Requests**: If you're reviewing someone else’s PR, ensure you provide feedback and approve it once you're satisfied.
4. **Resolve Merge Conflicts**: If two developers edit the same part of a file, a merge conflict will occur when merging. Resolve these conflicts manually by editing the files and then committing the changes.





**Practical 4**

**AIM: Demonstrate Collaborating and cloning using Git**

# Set Up Git and GitHub

Before you start collaborating or cloning repositories, make sure you have the following set up:

1. **Install Git**: If you haven't already, download and install [Git](https://git-scm.com/downloads) on your machine.
2. **Create a GitHub Account**: Go to [GitHub](https://github.com/) and create an account if you don't have one.
3. **Configure Git**: Set up your Git configuration with your name and email.

git config --global user.name "Your Name"

git config --global user.email [youremail@example.com](mailto:youremail@example.com)

# Step 2: Clone a GitHub Repository

Cloning a repository allows you to create a copy of a project on your local machine, enabling you to work on it.

1. **Find a Repository to Clone**: Visit the repository page on GitHub (e.g., <https://github.com/username/repository>) and click the green **Code** button.
2. **Copy the Clone URL**: In the popup, choose either **HTTPS** or **SSH** and copy the URL. If you're using HTTPS, it will look like https://github.com/username/repository.git.

## Clone the Repository Locally:

Open a terminal on your computer and navigate to the directory where you want to clone the repository. Then run:

git clone https://github.com/username/repository.git

* Replace https://github.com/username/repository.git with the URL you copied.

This will create a local copy of the repository on your machine.

## Navigate to the Repository Folder:

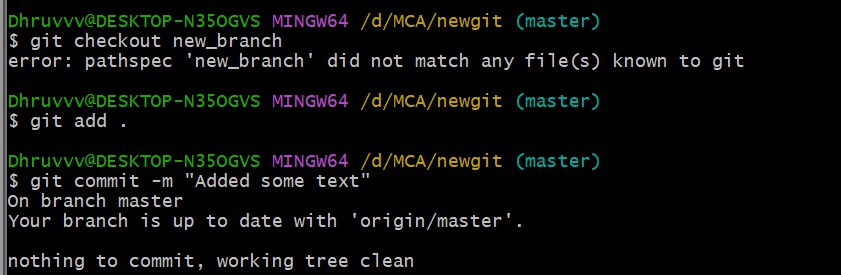
cd repository # Navigate into the cloned directory

# Step 3: Work on the Project Locally

Once you've cloned the repository, you can start making changes to the code.

1. **Create a New Branch**: Before making changes, it's recommended to create a new branch. This ensures your changes don't interfere with the main codebase until you're ready to merge.

git checkout -b feature/your-feature # Create and switch to a new



1. **Make Changes**: Edit files as needed using your preferred editor or IDE.
2. **Stage Changes**: After making changes, you need to stage them before committing.

git add . # Stages all modified files

1. **Commit Changes**: Once changes are staged, commit them to your local branch.

git commit -m "Add feature X"

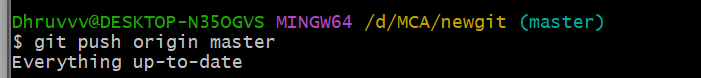
# Step 4: Push Changes to GitHub

Once you’ve committed your changes locally, you need to push them to your GitHub repository.

**Push Your Changes** to the remote repository:

git push origin feature/your-feature # Push the feature branch to GitHub

This uploads your local changes to your GitHub repository.



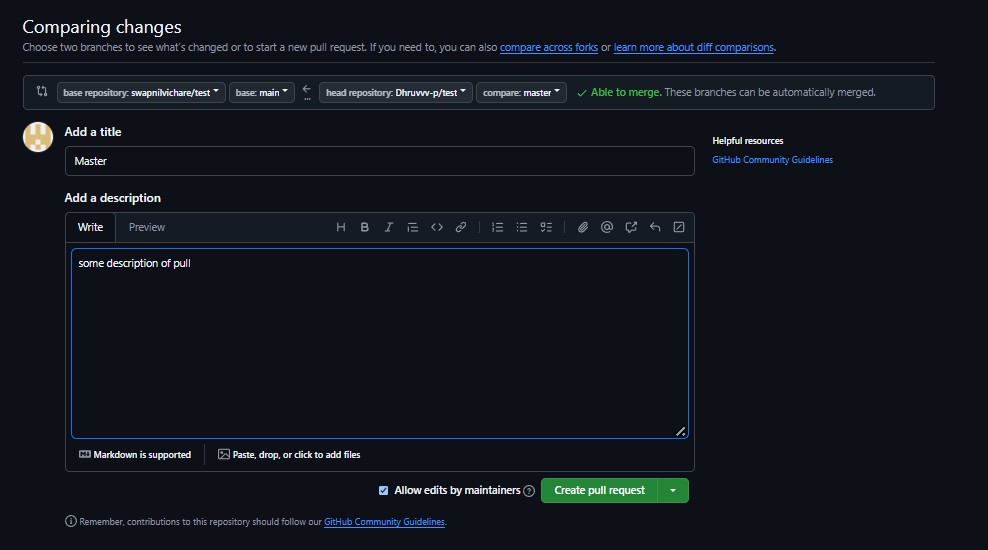
# Step 5: Create a Pull Request (PR)

To contribute your changes back to the original repository, you'll need to open a pull request (PR).

1. **Go to the GitHub Repository**: Visit the repository where you want to make the changes.
2. **Create a Pull Request**: On GitHub, you'll see an option to compare your branch (e.g., feature/your-feature) with the main branch of the repository (e.g., main). Click **New Pull Request**.

## Fill Out the Pull Request Form:

* + Add a **title** and **description** explaining the changes you’ve made.
  + Review your changes.
  + Click **Create Pull Request**.



1. **Code Review**: The repository maintainer (or other collaborators) will review your changes. They might ask for changes or approve the PR.
2. **Merge the Pull Request**: Once your changes are approved, the maintainer will merge your changes into the main branch of the project.

# Step 6: Sync Your Fork (If Working on a Forked Repo)

If you are working on a forked repository and want to keep your fork in sync with the original repository:

1. **Add the Original Repository as a Remote**: This allows you to fetch updates from the original repository.

git remote add upstream https://github.com/owner/original- repository.git

Replace https://github.com/owner/original-repository.git with the original repository's URL.

## Fetch the Latest Changes from the Original Repository:

git fetch upstream # Fetch the changes from the original repo

## Merge the Latest Changes into Your Local Branch:

git checkout main # Switch to your main branch

git merge upstream/main # Merge the latest changes from the original repo

## Push the Changes to Your Fork:

git push origin main # Push the updated main branch to your fork

# Step 7: Pull Latest Changes from the Original Repository

To keep your local repository up-to-date with the remote repository on GitHub, you can pull the latest changes.

## Switch to Your Local Main Branch:

git checkout main

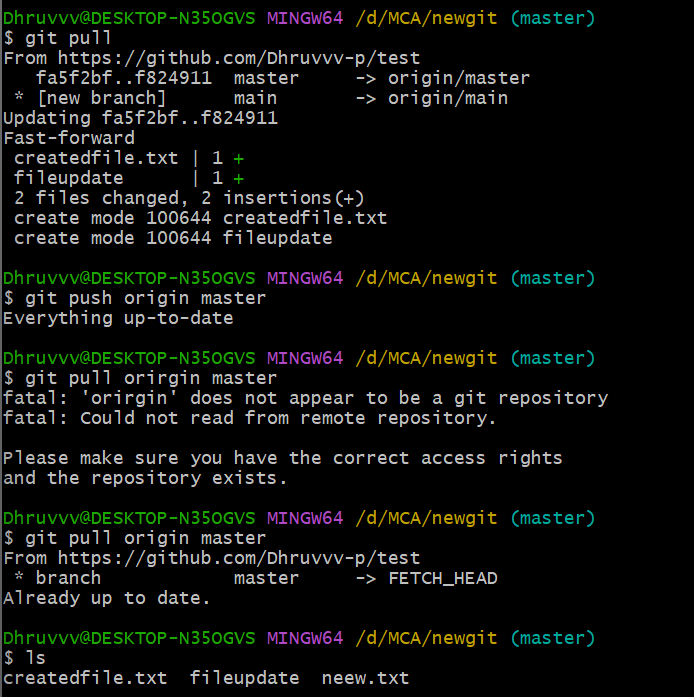
## Pull Latest Changes from GitHub:

git pull origin main # Pull the latest changes from the remote repository

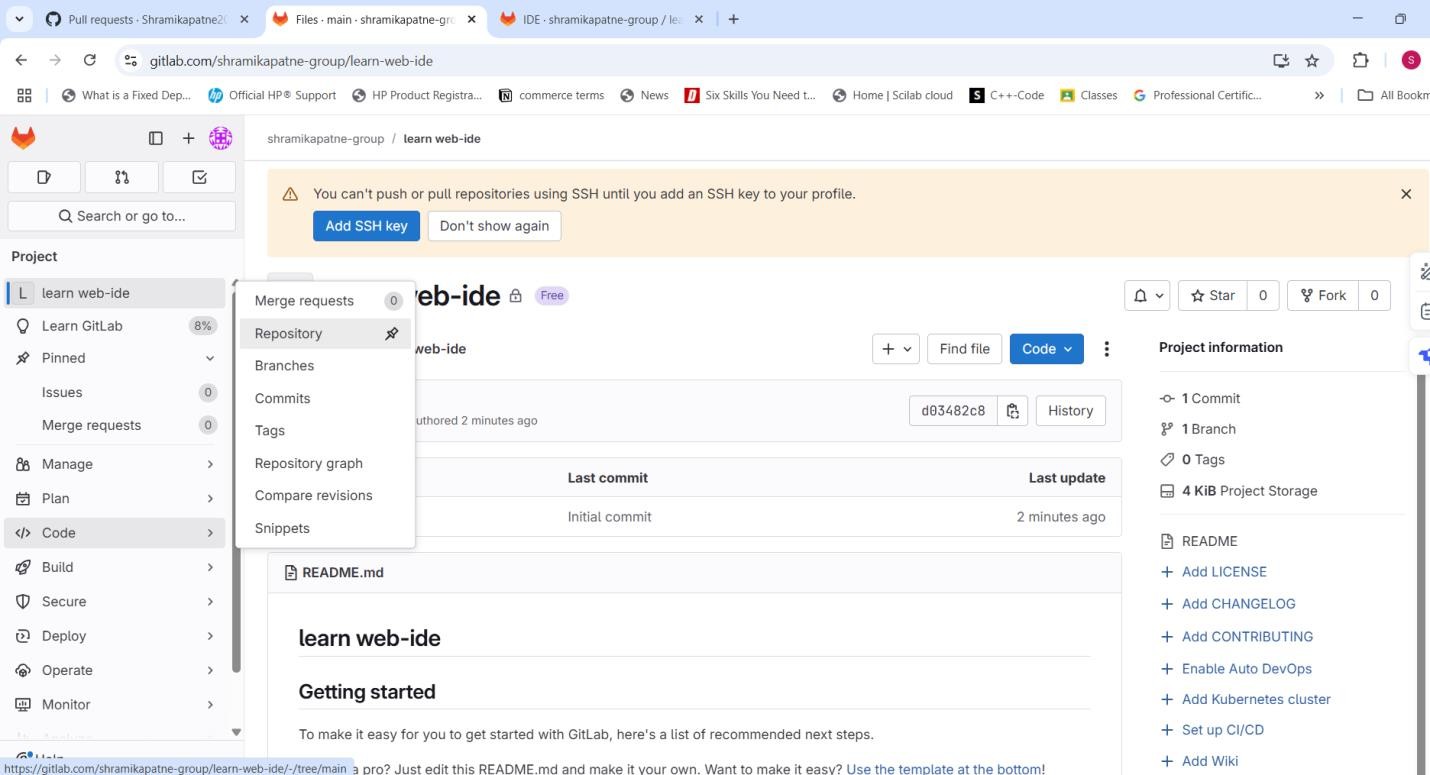
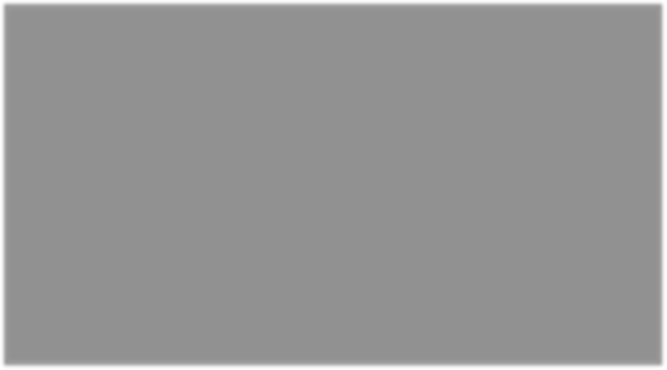
# Step 8: Collaborate with Other Developers

When collaborating with other developers on GitHub, you'll typically follow these best practices:

1. **Regularly Pull Latest Changes**: To ensure you're not working on outdated code, frequently pull the latest changes from the main branch (especially before you start working on new features or bug fixes).
2. **Create Feature Branches**: Always create a new branch for each feature or bug fix. This avoids conflicts and keeps the history clean.
3. **Review Pull Requests**: If you're reviewing someone else’s PR, ensure you provide feedback and approve it once you're satisfied.
4. **Resolve Merge Conflicts**: If two developers edit the same part of a file, a merge conflict will occur when merging. Resolve these conflicts manually by editing the files and then committing the changes.



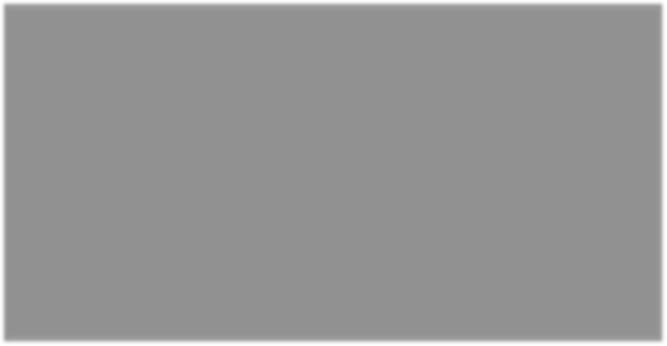
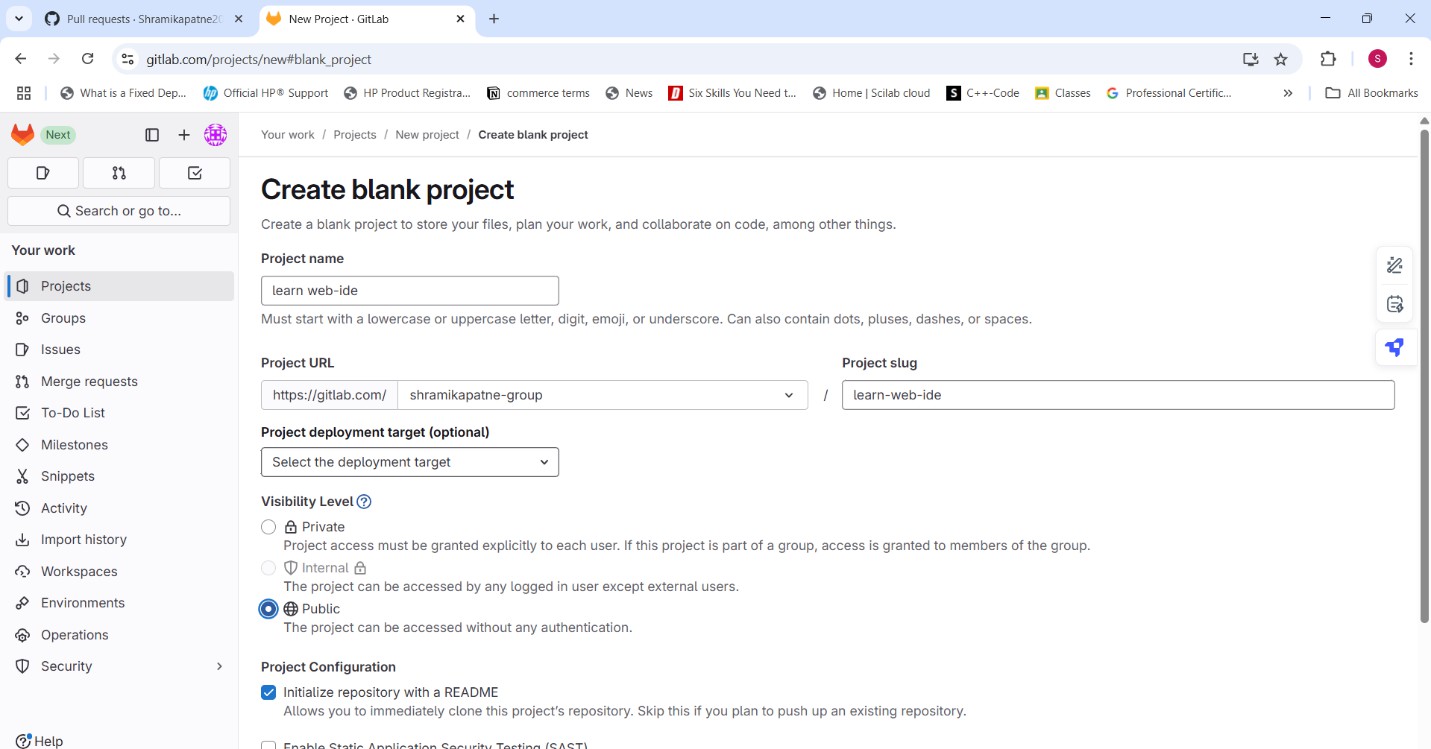
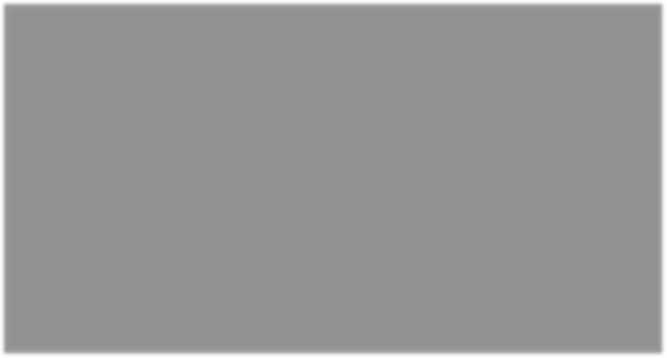
**Practical 5**



**AIM: Using GitLab Web IDE**

**Steps:**

1. Sign up at [https://gitlab.com](https://gitlab.com/)
2. Create a project.
3. Click on Web IDE in your repository.



1. Create a file (index.html):

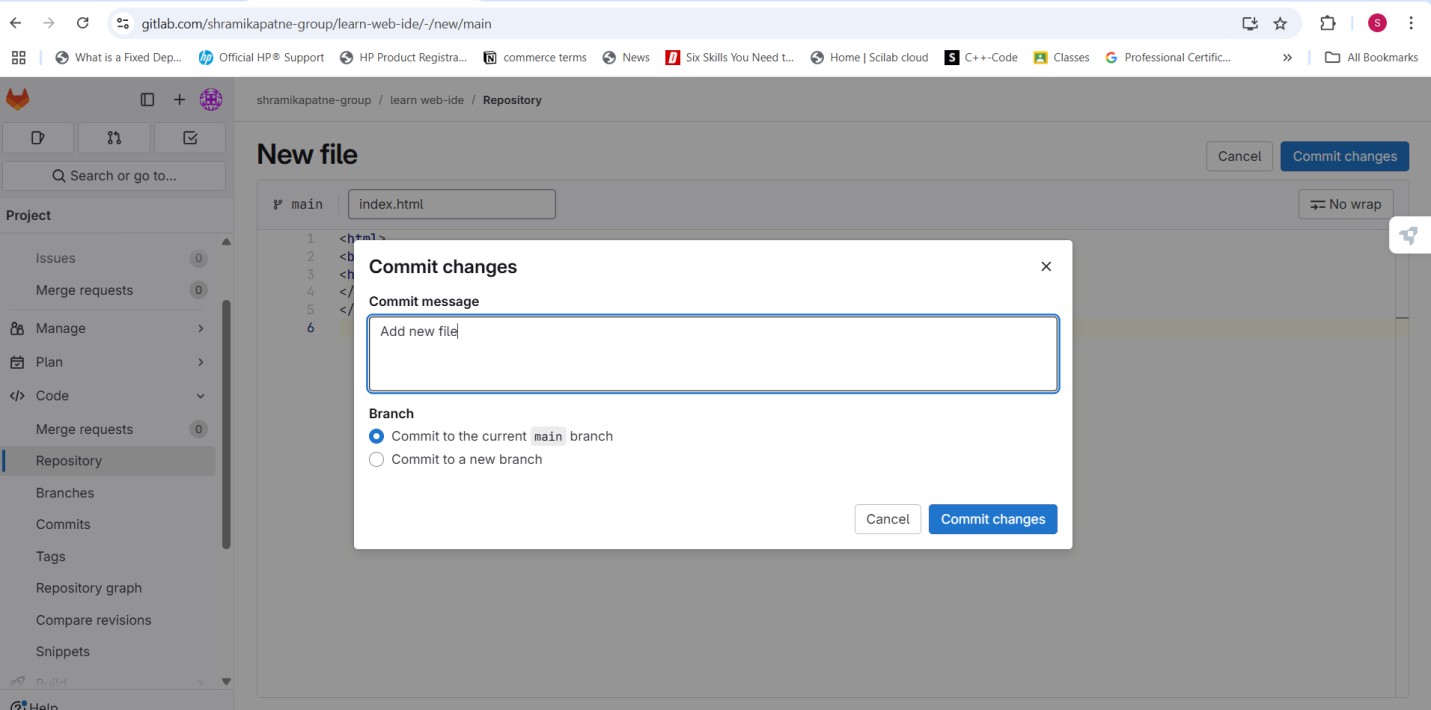
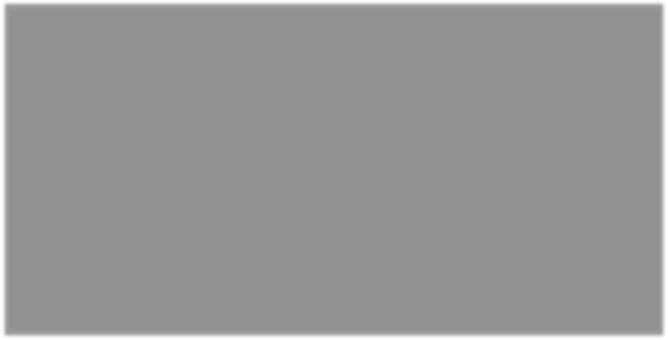
**<html>**

**<body>**

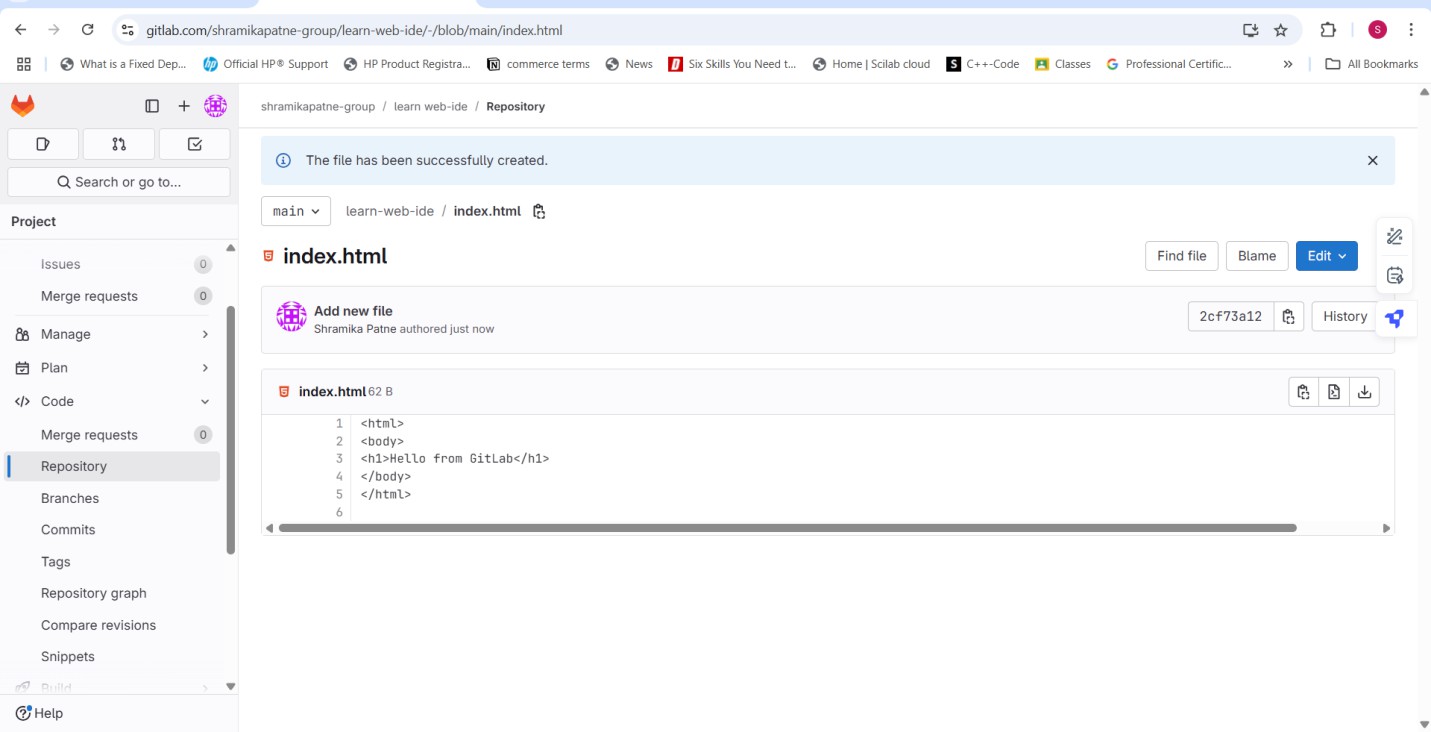
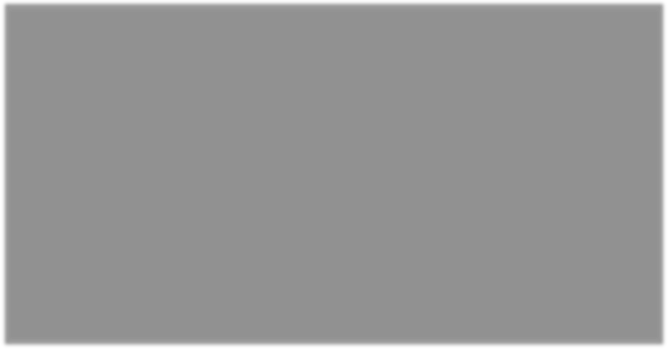
**<h1>Hello from GitLab</h1>**

**</body>**

**</html>**



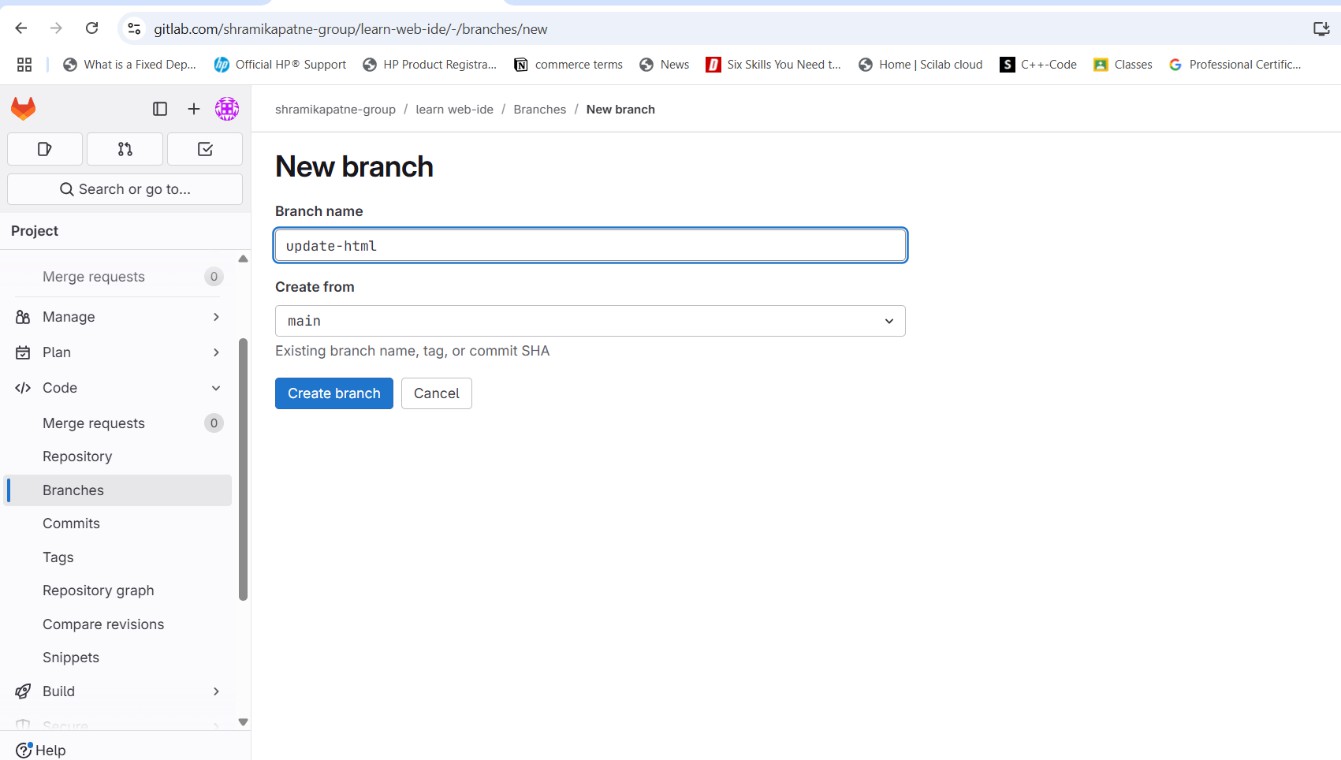
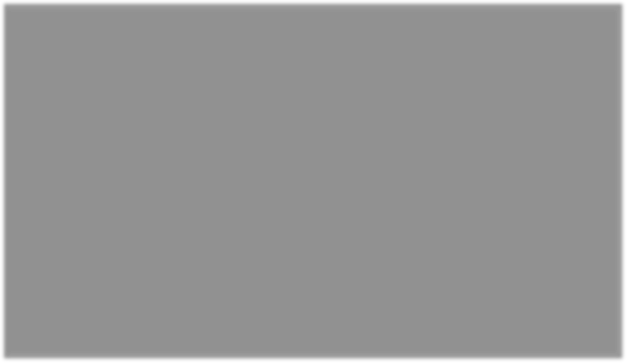
1. Click Commit and push changes.



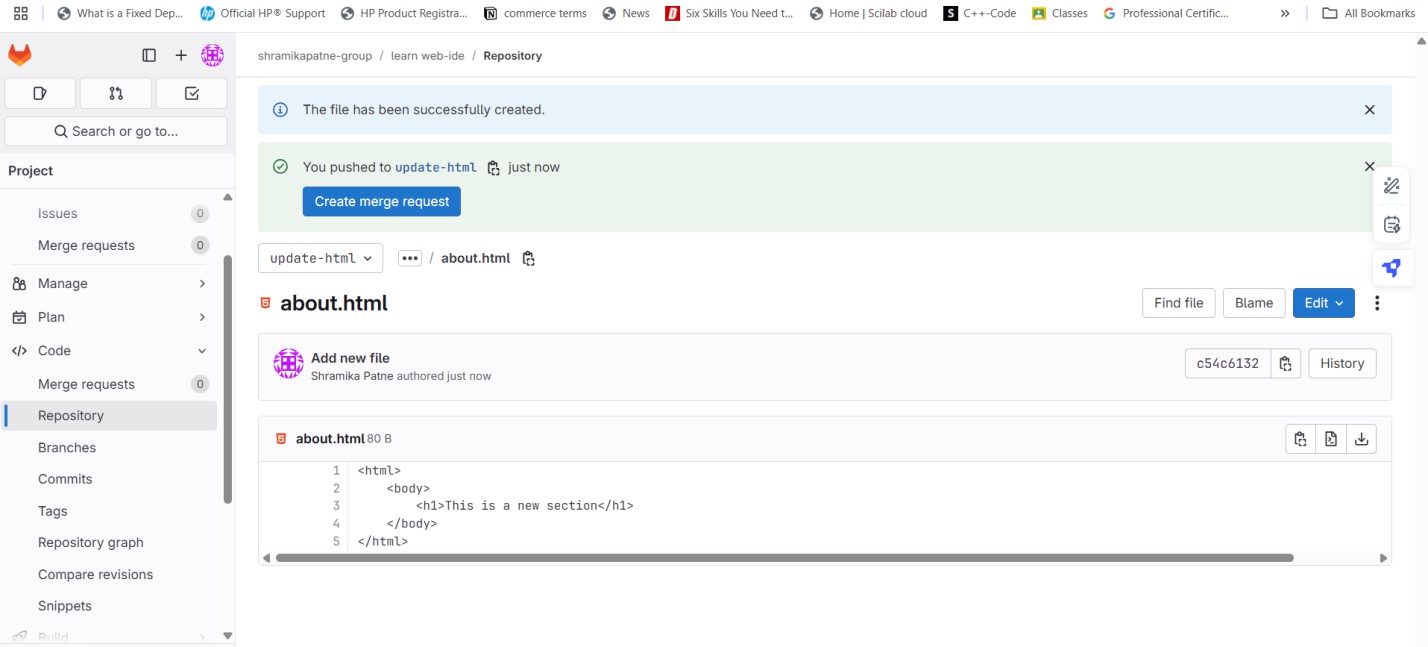
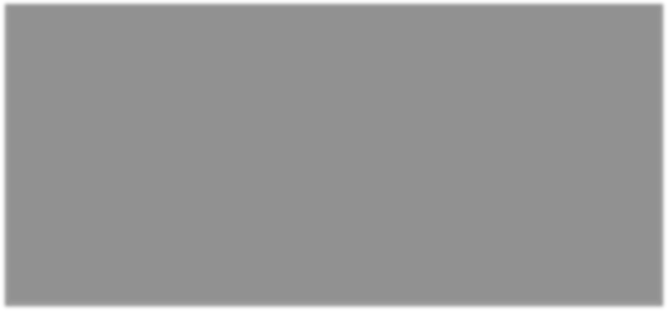
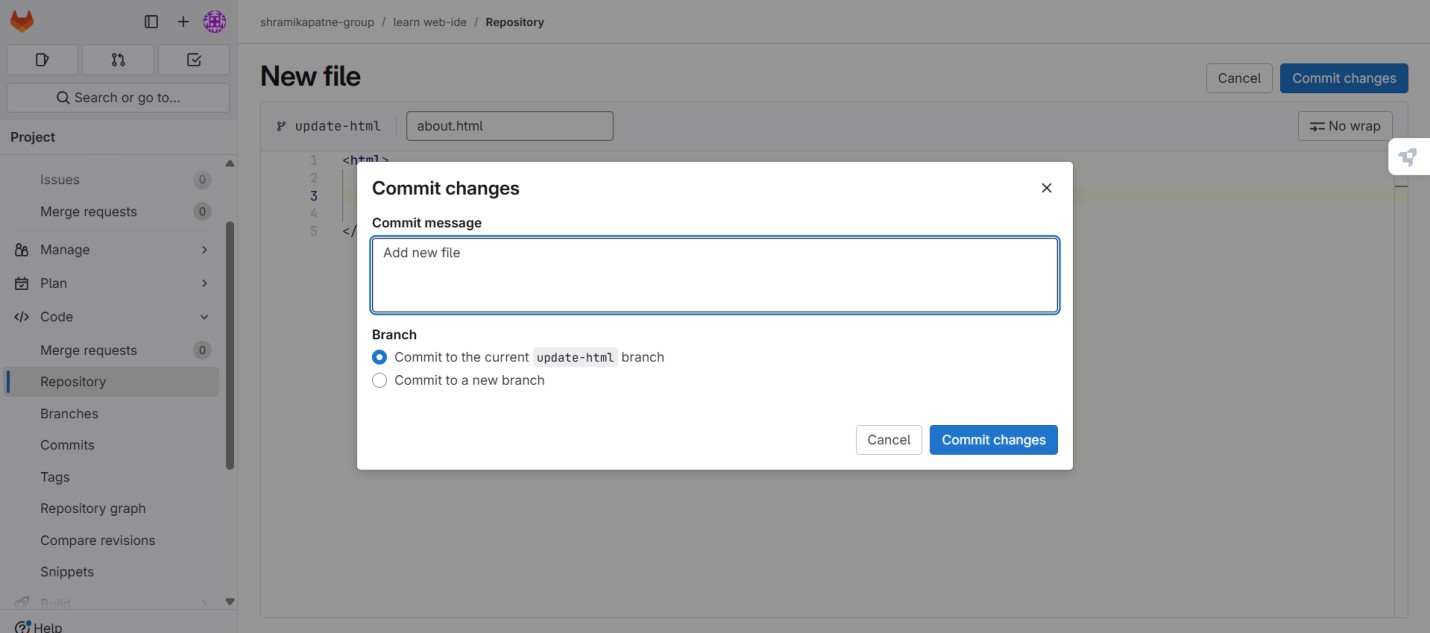
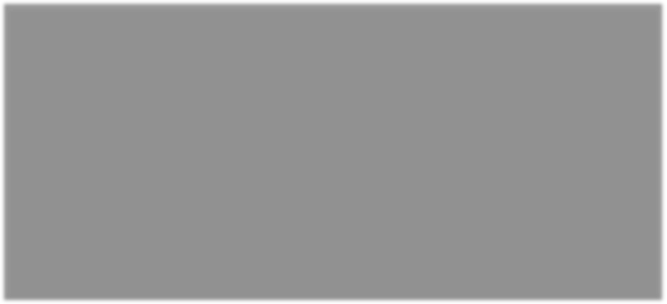
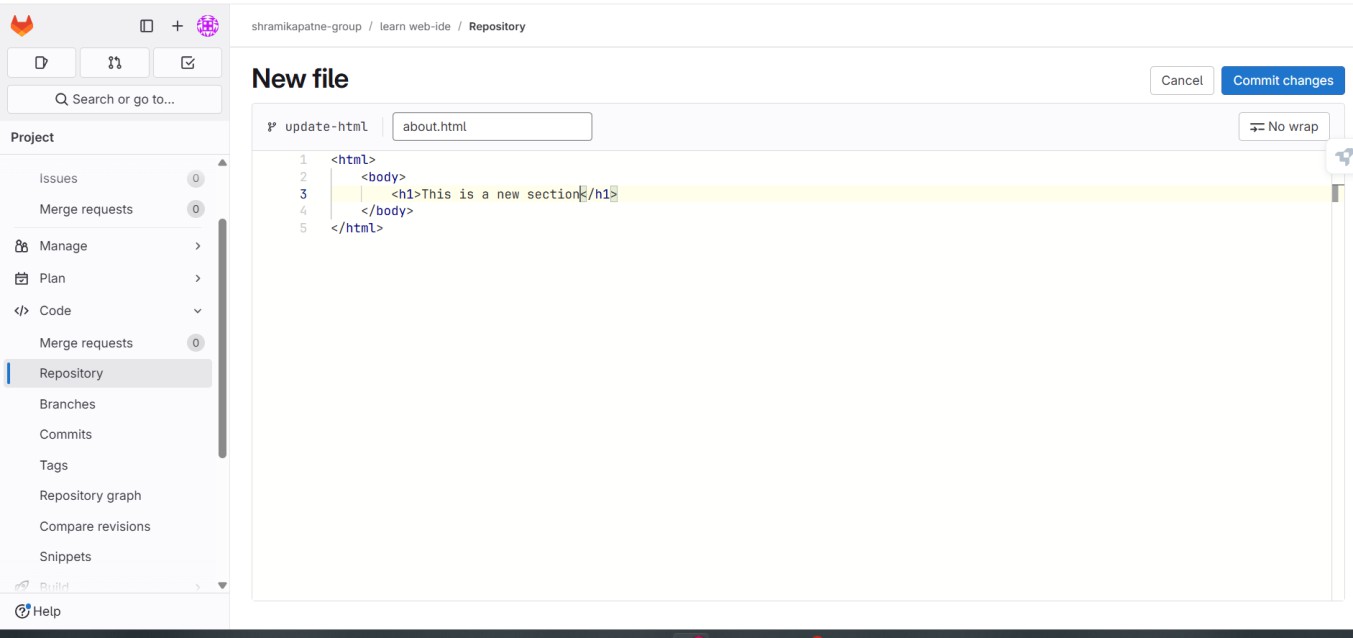
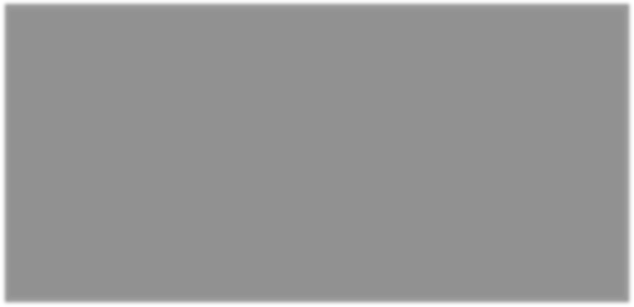
**Practical 6**

**AIM: Demonstrate CI/CD Workflow in GitLab using .py, .bash, .java file**

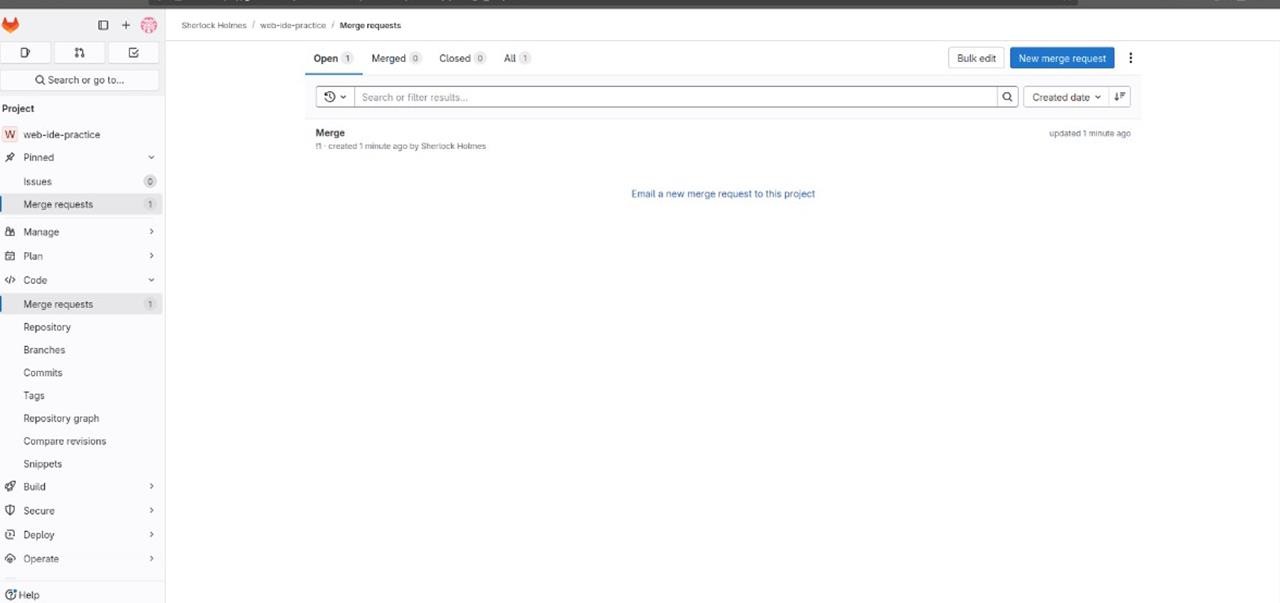
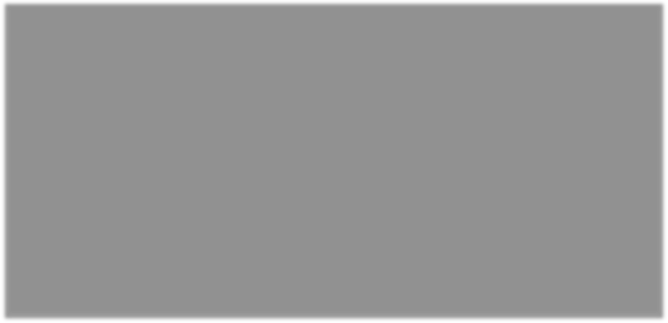
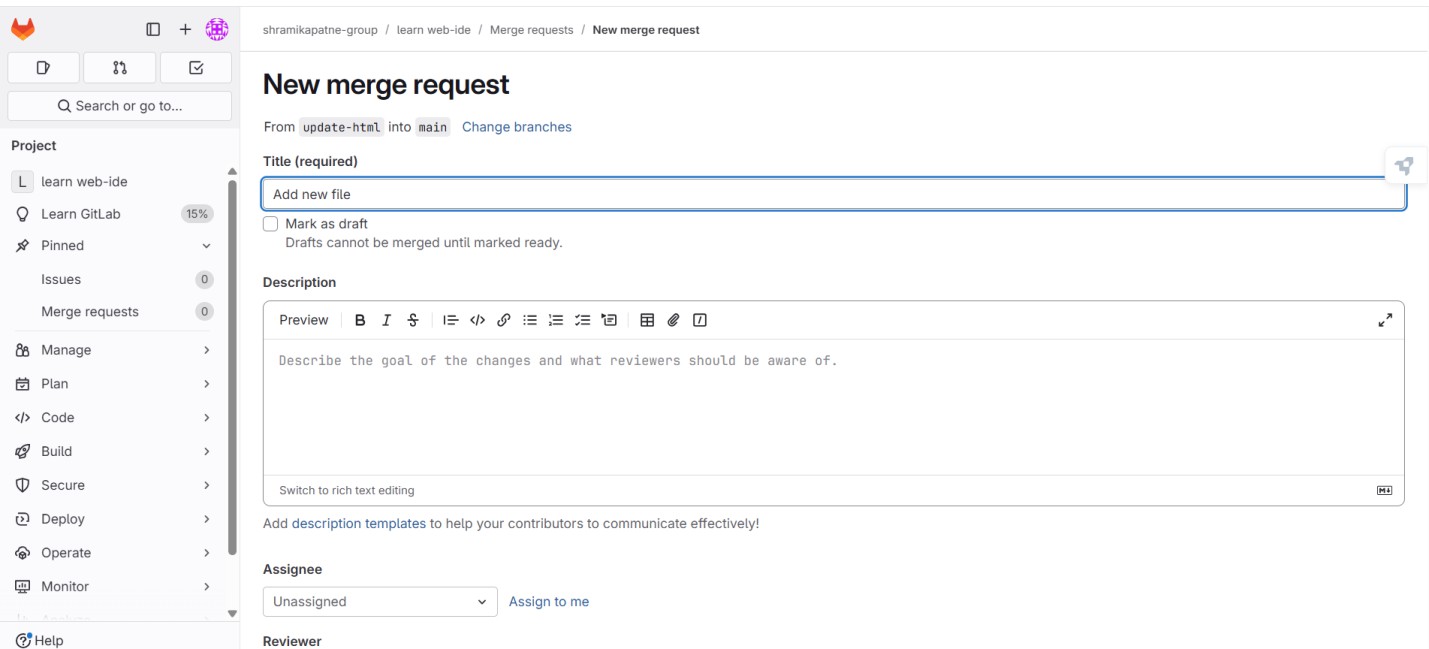
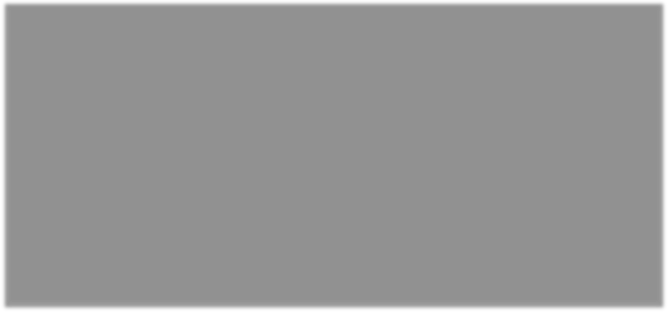
* 1. Create a new branch in Web IDE.



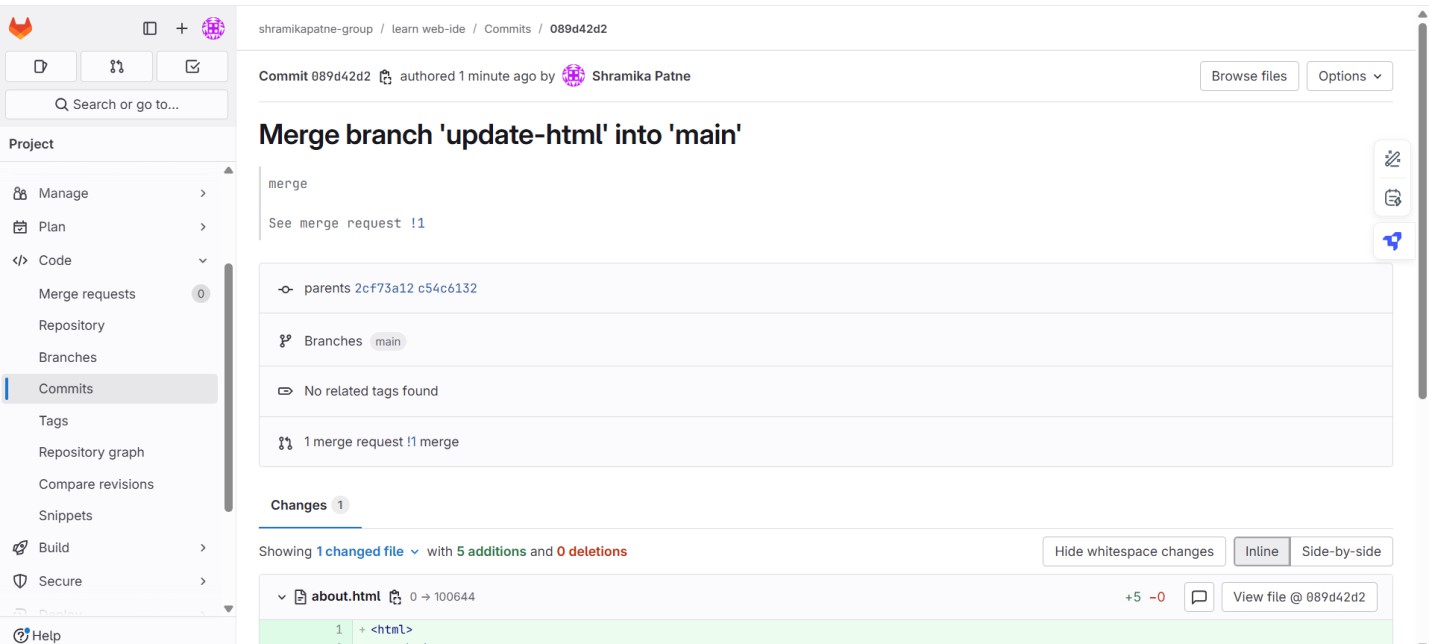
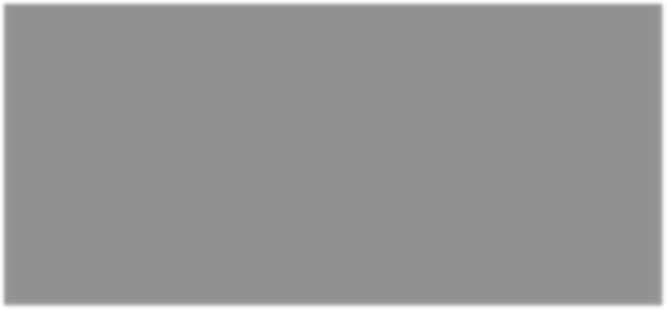
* 1. Add/edit a file and commit.



* 1. Click on Merge Requests > New Merge Request.



* 1. Select source and target branches.
  2. Submit and merge after review.



**Steps:**

1. In your repo, create .gitlab- ci.yml: stages:
   * build
   * test **build- job: stage**

**:**

**build script**

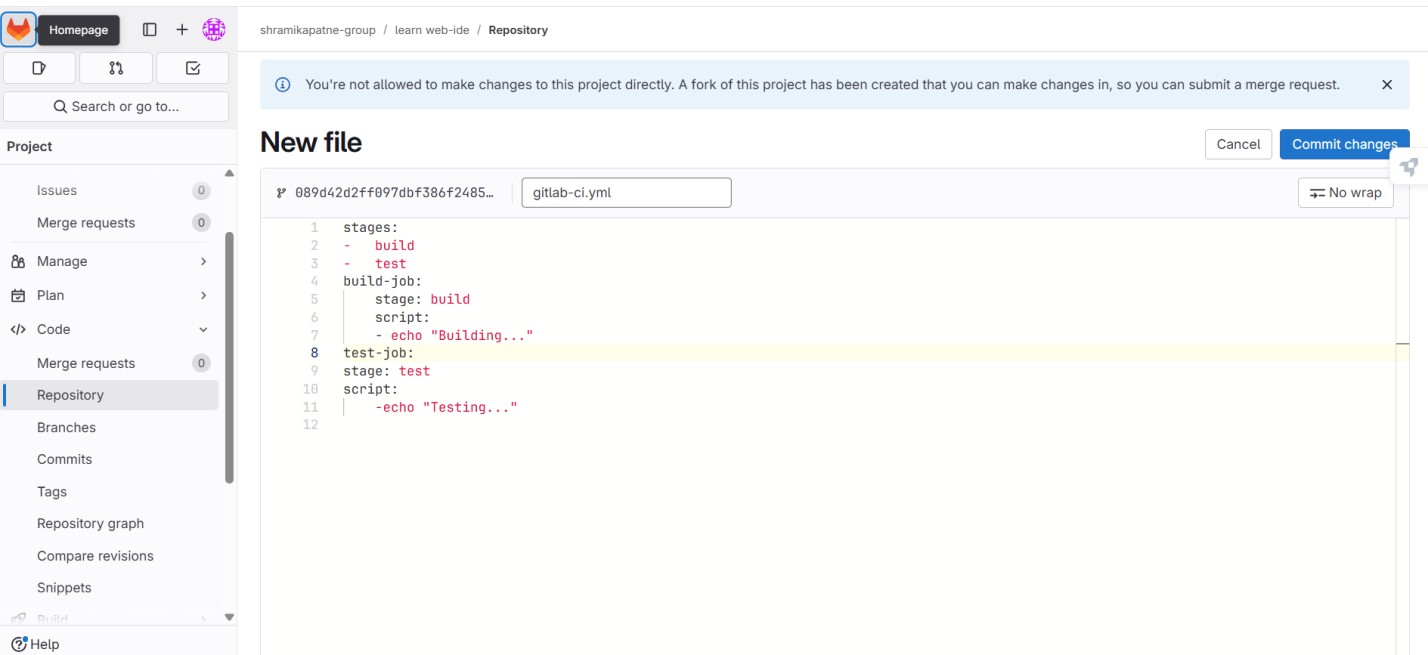
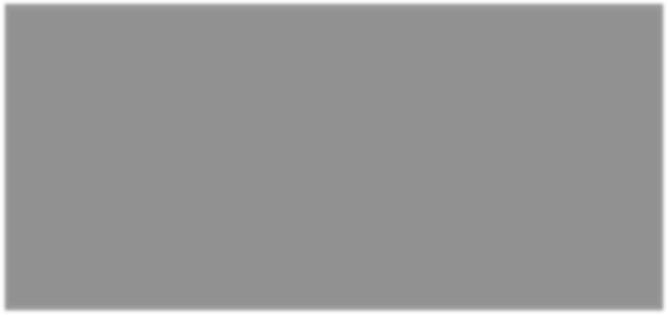
**:**

**- echo "Building..." test-job:**

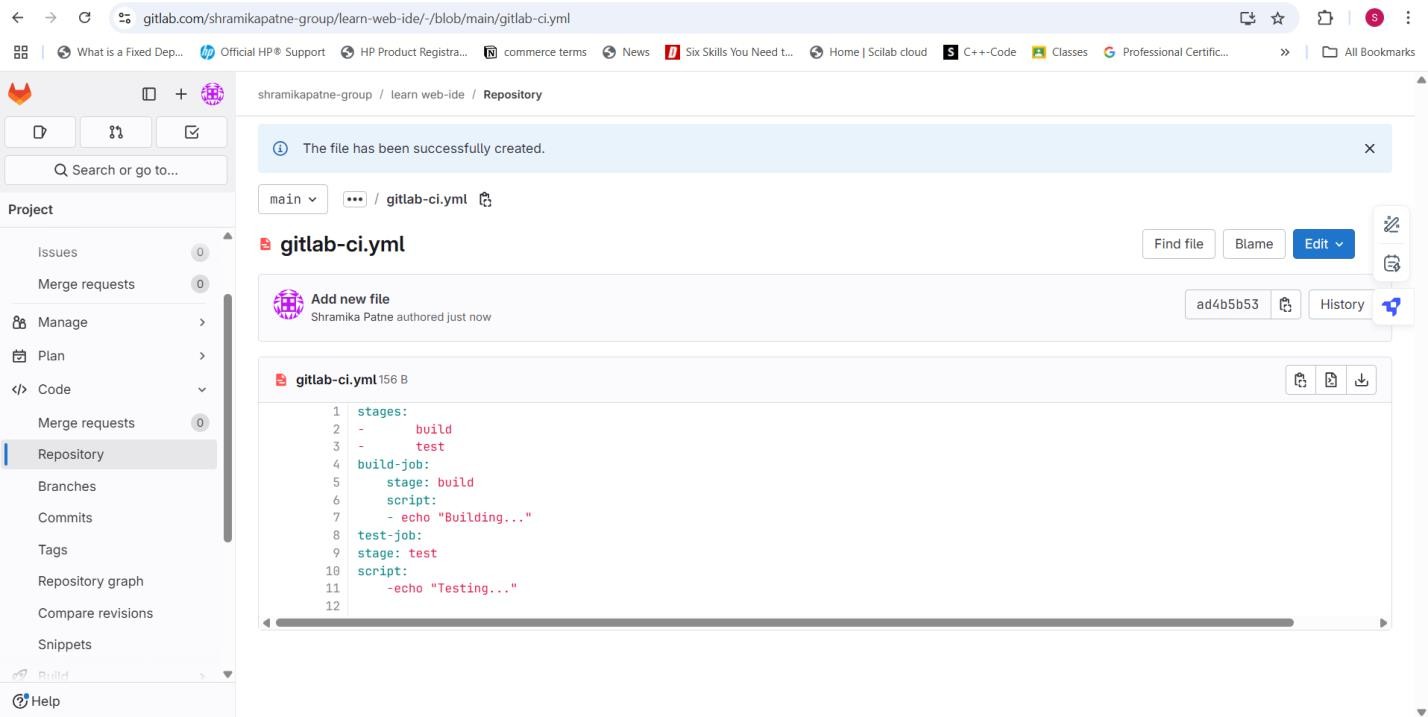
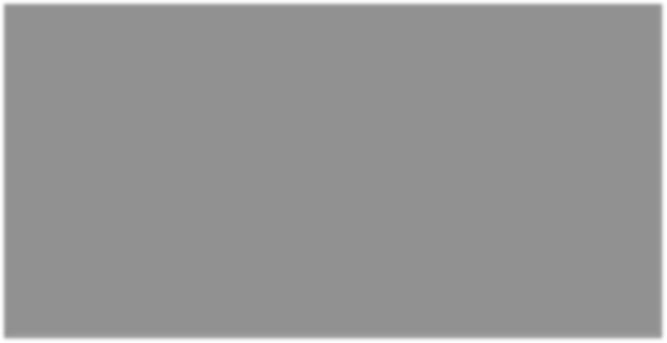
**stage:**

**test script:**

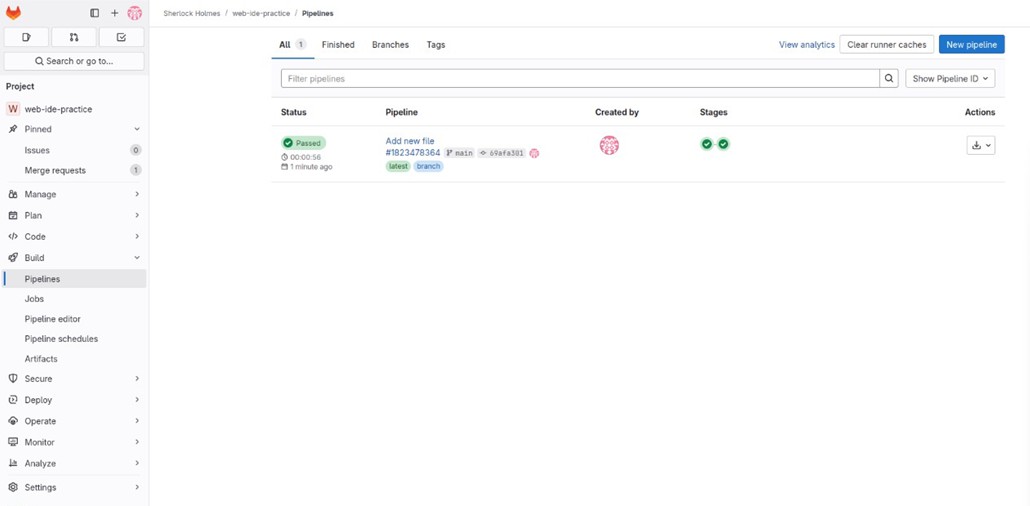
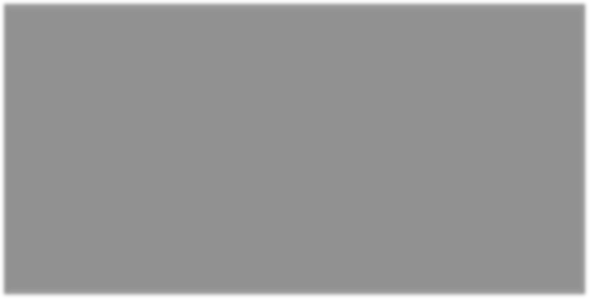
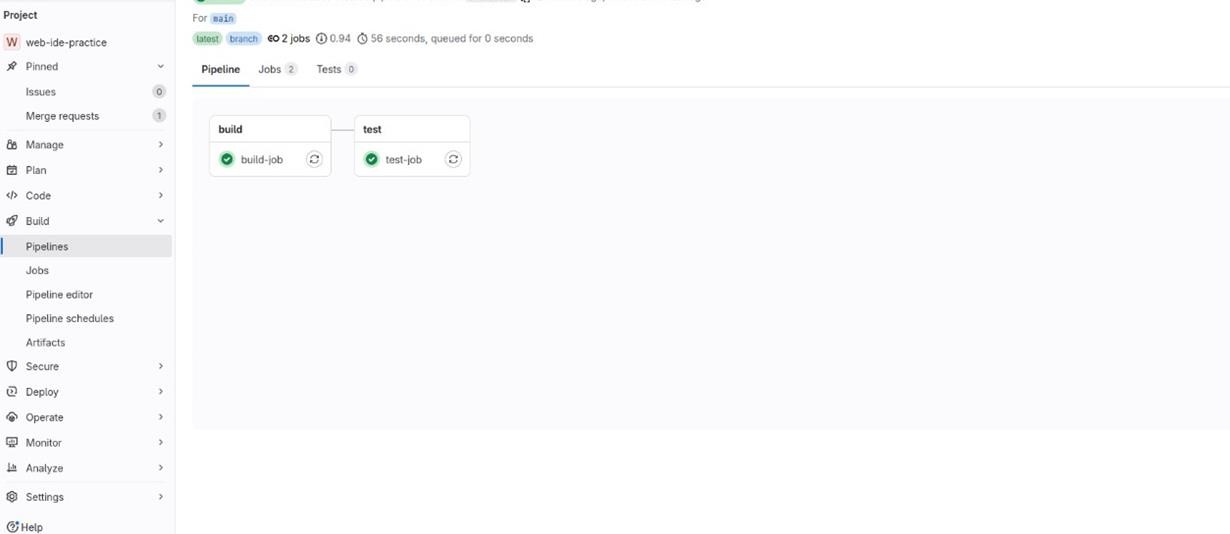
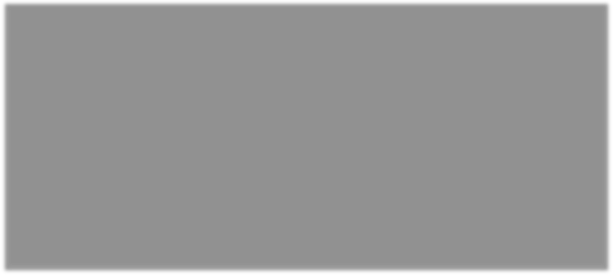
* + echo "Testing..."



1. Commit and push.



1. Go to CI/CD > Pipelines and view the build/test stages.

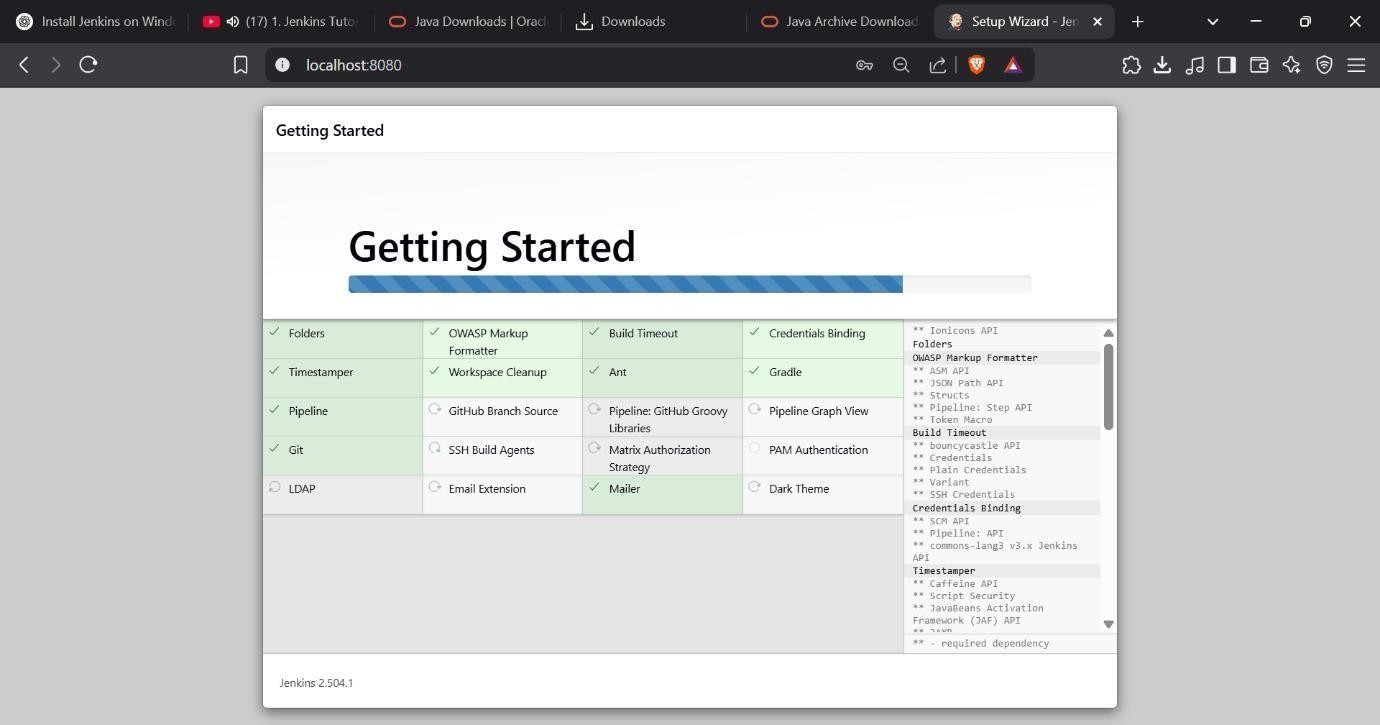


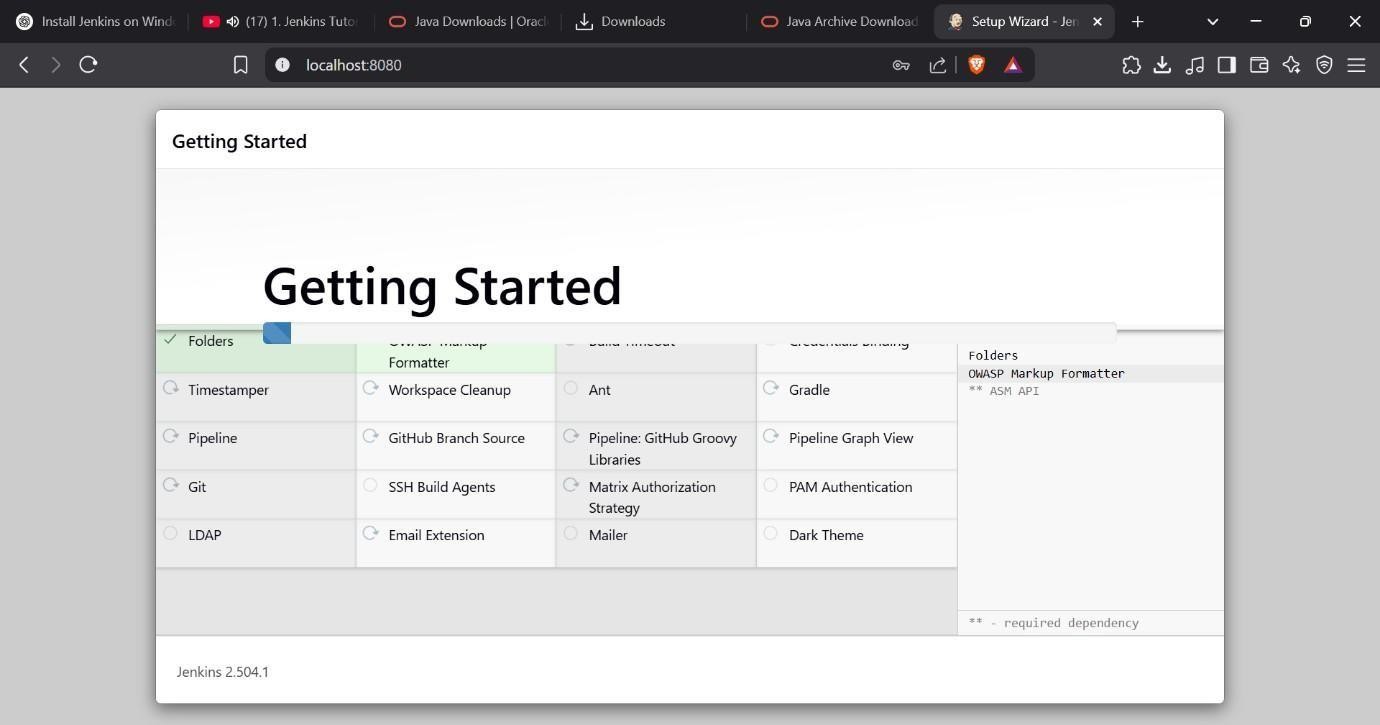
**Steps**

**Practical 7**

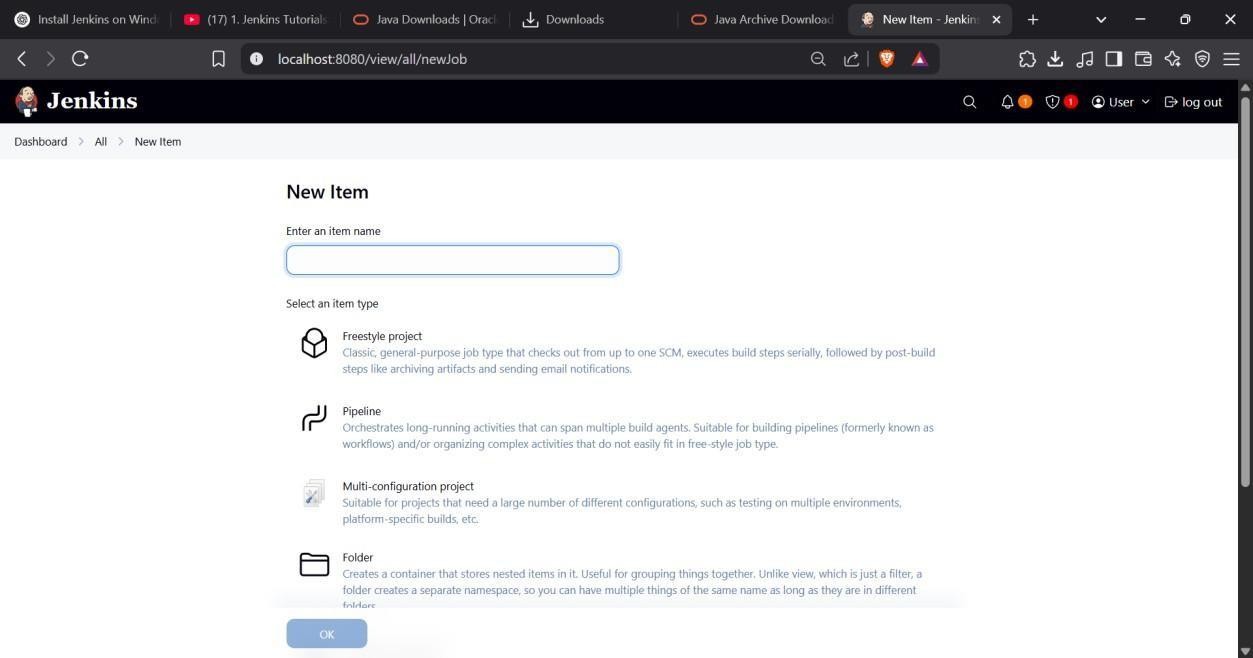
**AIM: Demonstrate setting Jenkins CI/CD pipeline.**

1. Install Jenkins (visit [https://www.jenkins.io](https://www.jenkins.io/))
2. Run Jenkins: [http://localhost:8080](http://localhost:8080/)

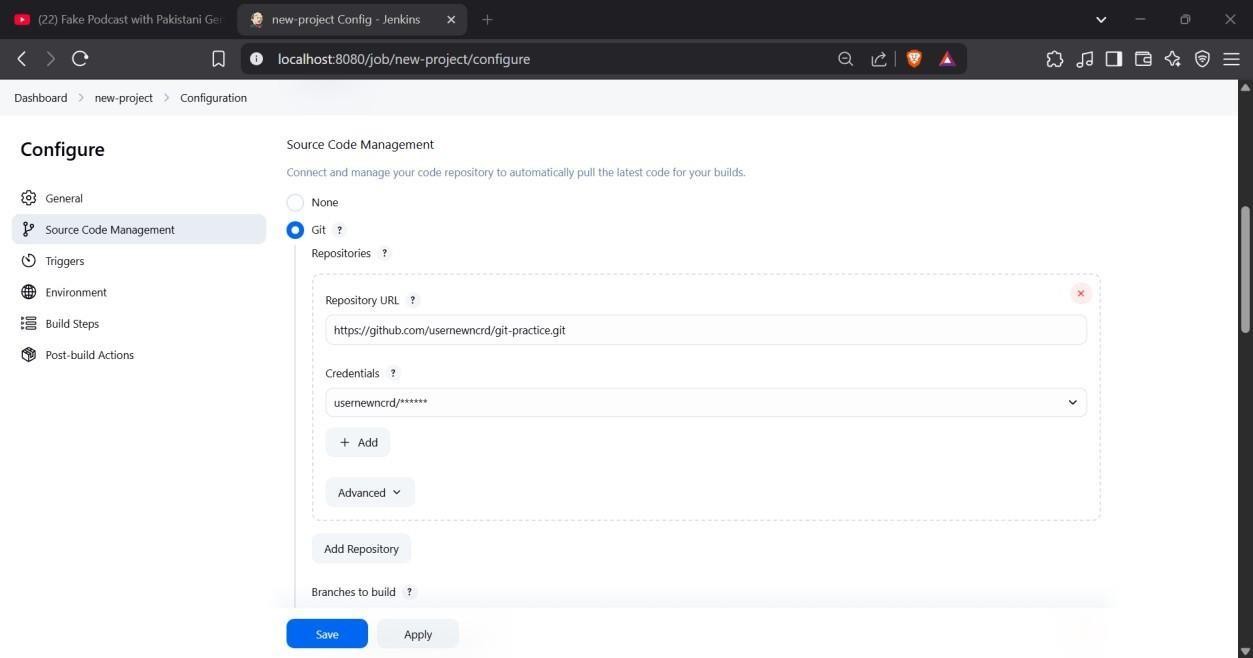




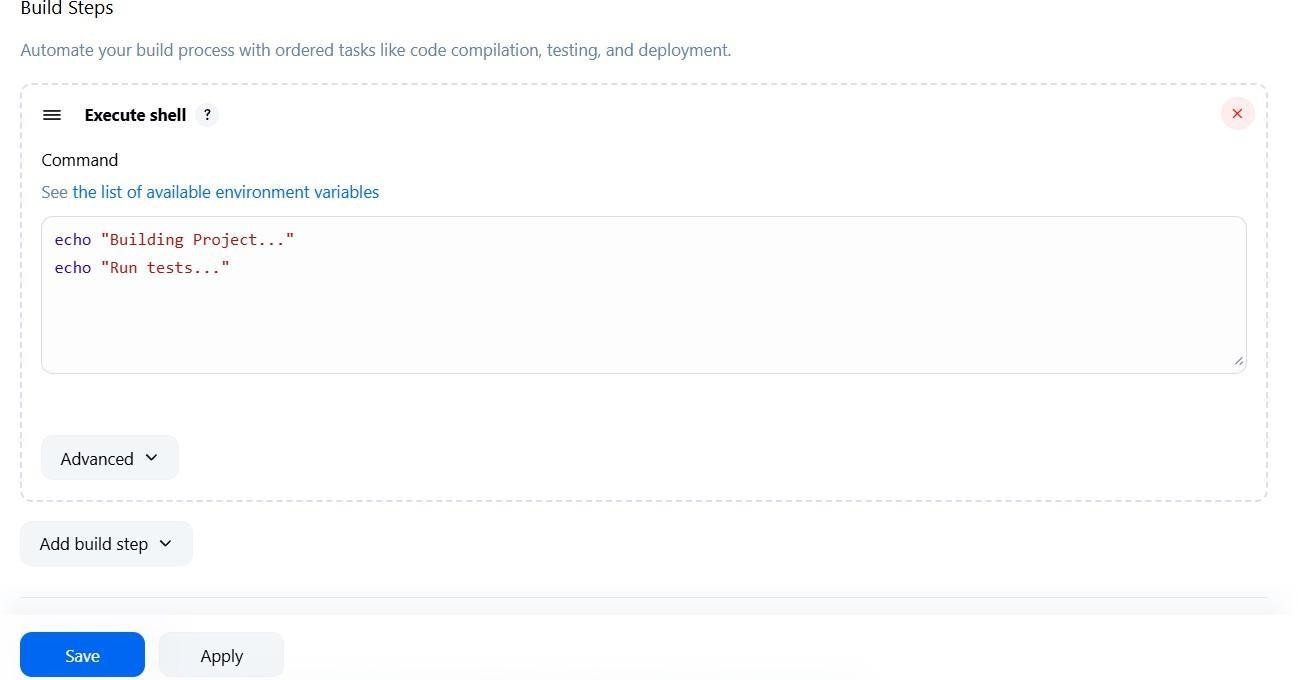
1. Create new Freestyle Project: CI-Demo



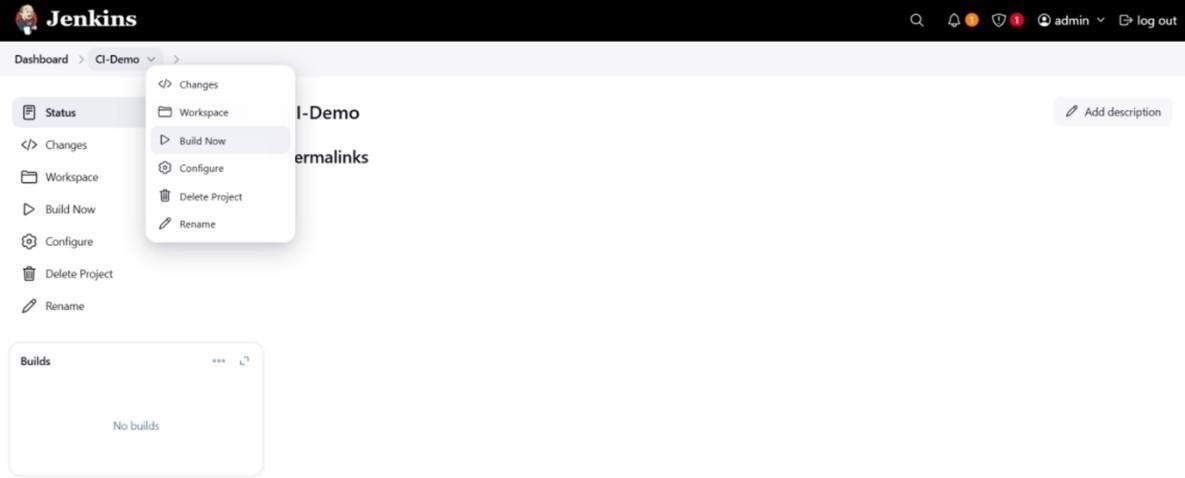
3. Under Source Code Management, choose Git and enter your repo URL.



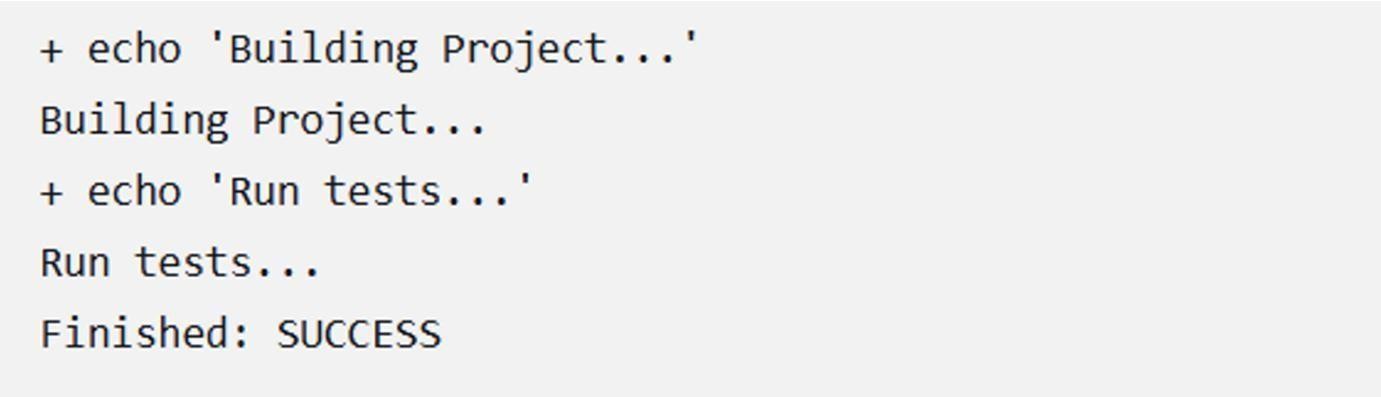
3. Add Build Step > Execute Shell: echo "Building Project..." echo "Run tests..."



3. Save and click Build Now.



3. Check output in Console Output.



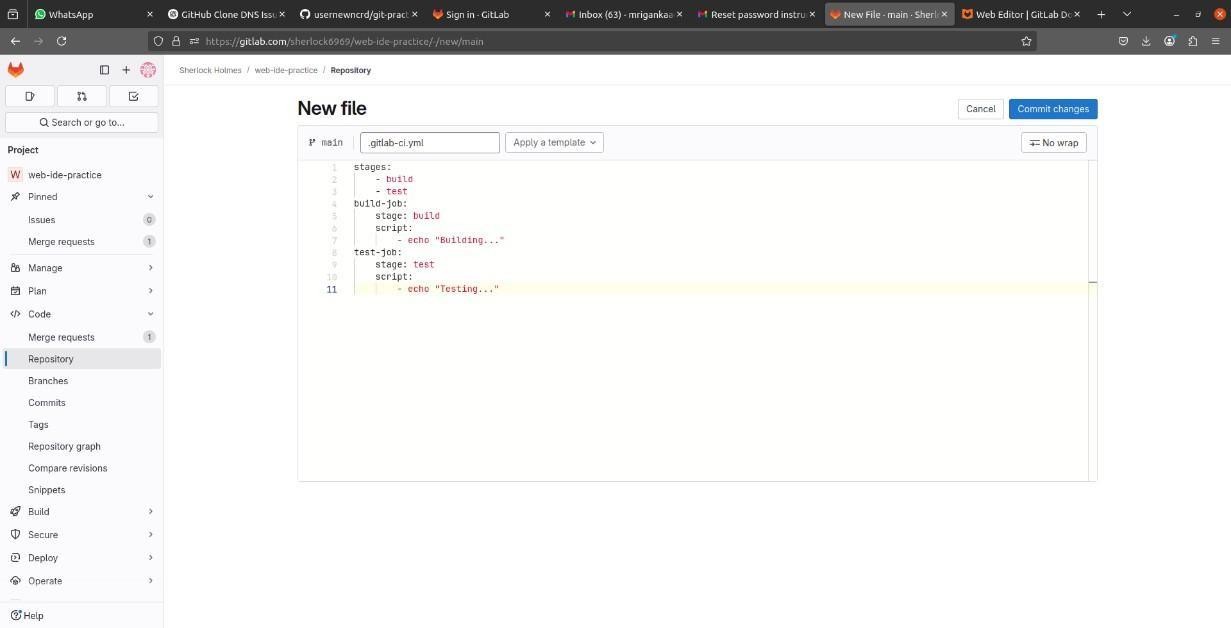
**Practical 8**

**AIM: Demonstrate Setting up of a CI/CD pipeline to build and deploy a web application to a local HTTP server**

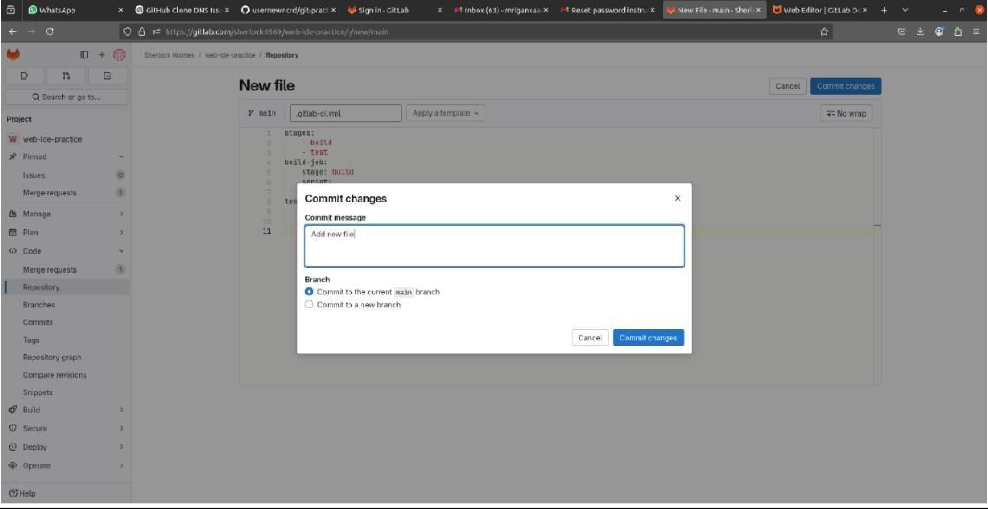
1. In your repo, create .gitlab-ci.yml: stages:
   * build - test build-job: stage: build script:
   * echo

"Building..." test-job: stage: test script:

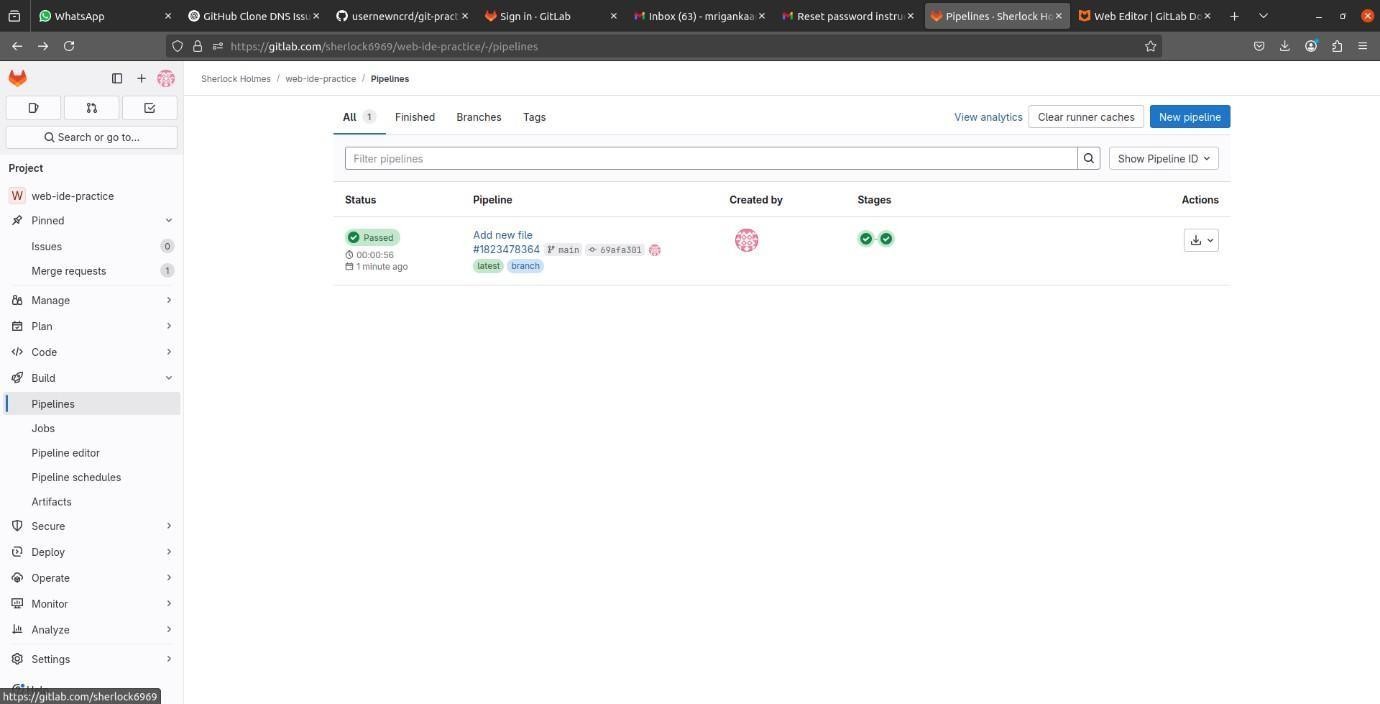
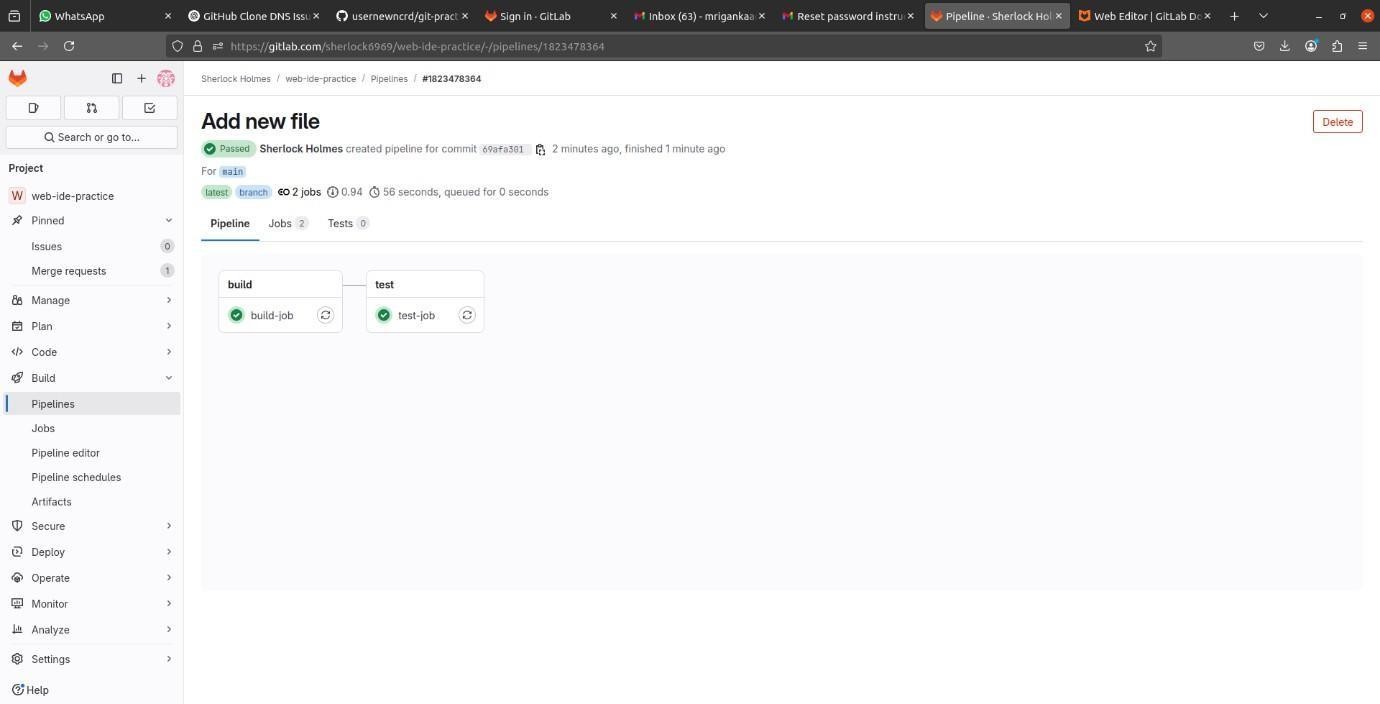
* + echo "Testing..."



1. Commit and push.



1. Go to CI/CD > Pipelines and view the build/test stages.



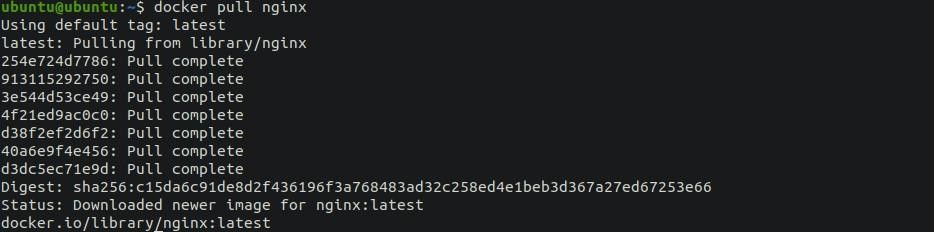
**Practical 9**

**AIM: Demonstrate basic Docker commands**

1. Check Docker version docker –version



1. Pull a Docker image from Docker Hub docker pull nginx



1. List all Docker images docker images



1. Run a container from an image docker run -d - p 8080:80 --name mynginx nginx

This will run the Nginx container and map port 80 (inside the container) to port 8080 (on your host).



1. List all running containers docker ps



1. Copy content from host to container docker cp index.html mynginx:/usr/share/nginx/html/

Replace index.html with your actual file. This copies a file into the running container.

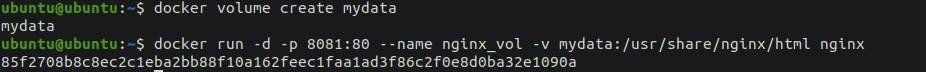
1. Copy content from container to host docker cp mynginx:/usr/share/nginx/html/index.html .



1. Create and use Docker volume for persistent content docker volume create mydata

docker run -d -p 8081:80 --name nginx\_vol -v mydata:/usr/share/nginx/html nginx

Now any data added to the /usr/share/nginx/html inside the container will persist even if the container is removed.



1. List Docker volumes docker volume ls



1. Remove a container docker rm -f mynginx Remove an image docker rmi nginx

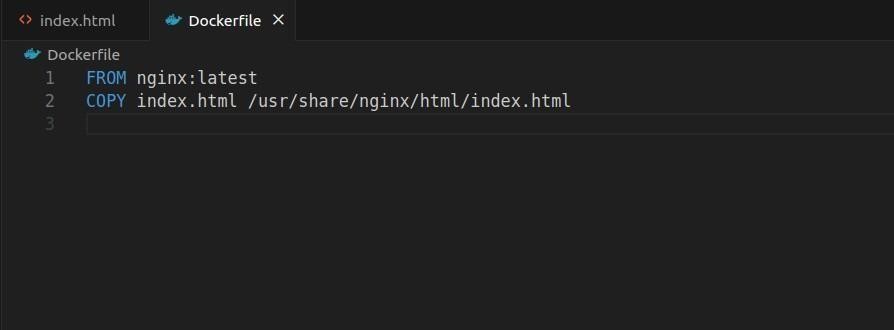
**Practical 10**

**AIM: Develop a simple containerized application using Docker**

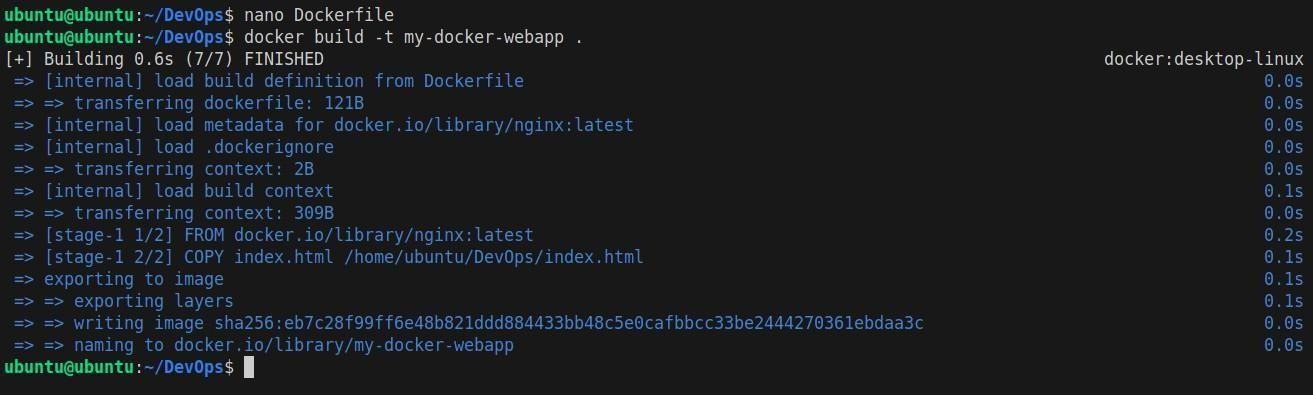
1. Index.html



1. DockerfIle :-



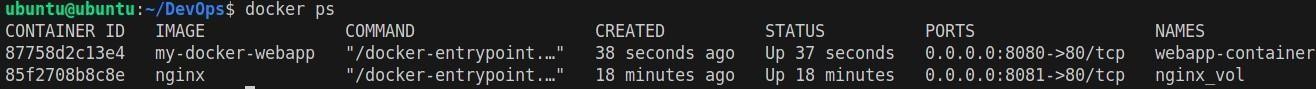
1. docker build -t my-docker-webapp .



1. docker run -d -p 8080:80 --name webapp-container my-docker-webapp



1. docker ps



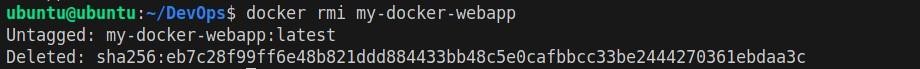
1. docker stop webapp-container

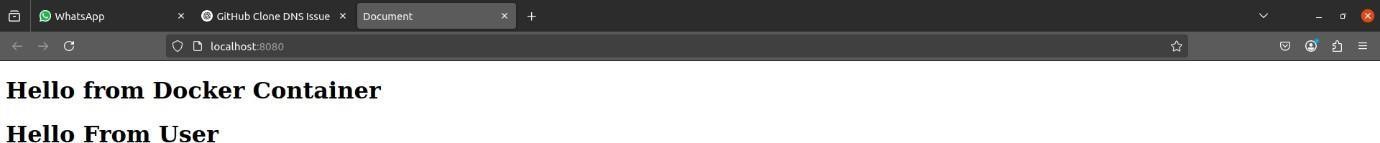


1. docker rm webapp-container



1. docker rmi my-docker-webapp

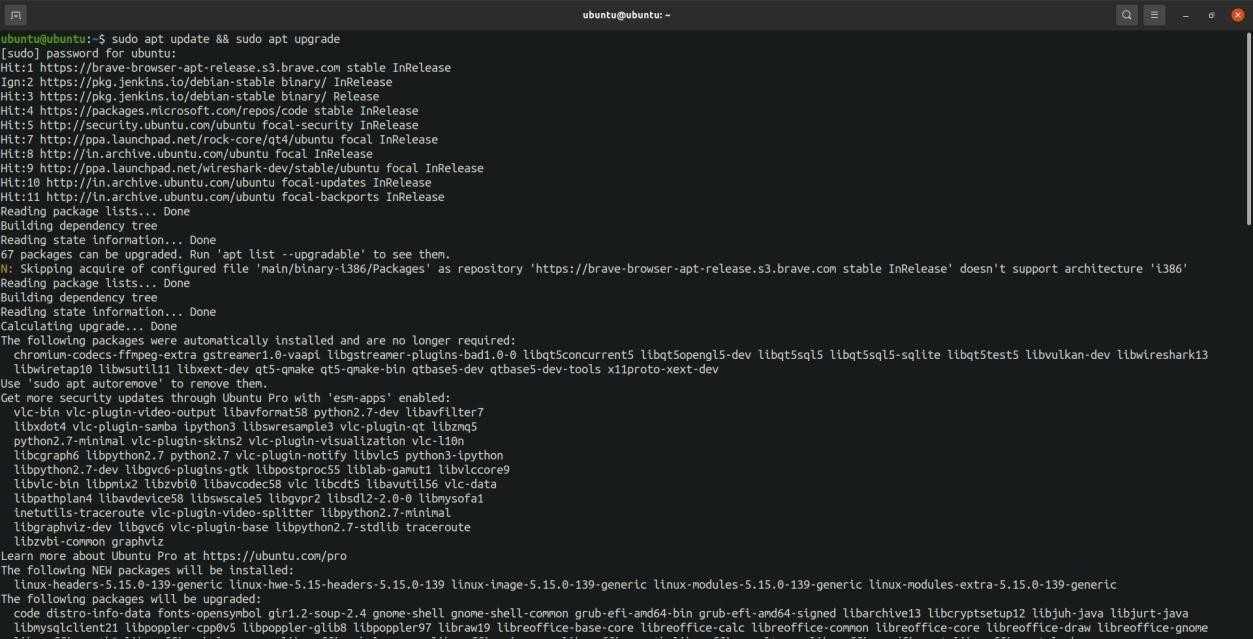




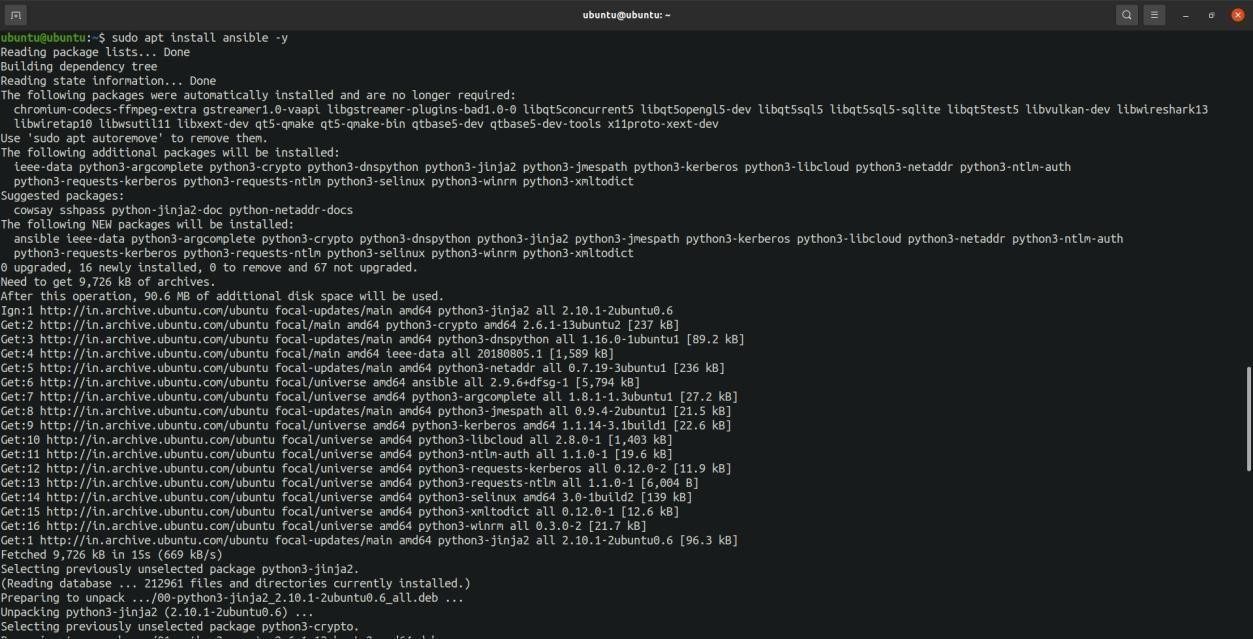
**Practical 11**

**AIM: Demonstrate add-on ansible commands**

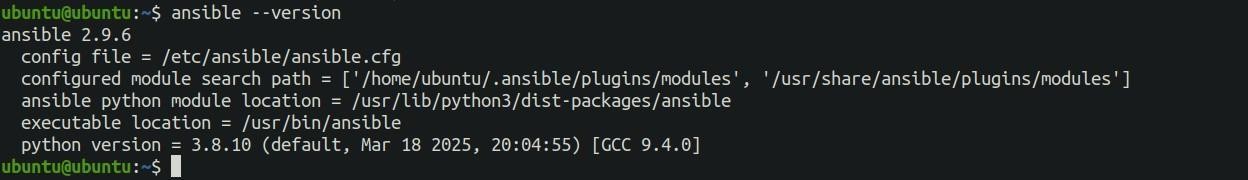
Step 1: Update your VM

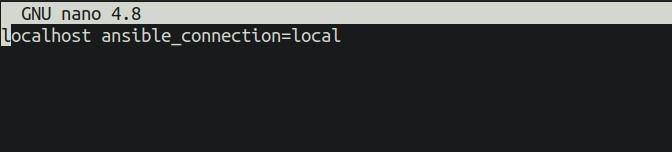


Step 2: Install Ansible

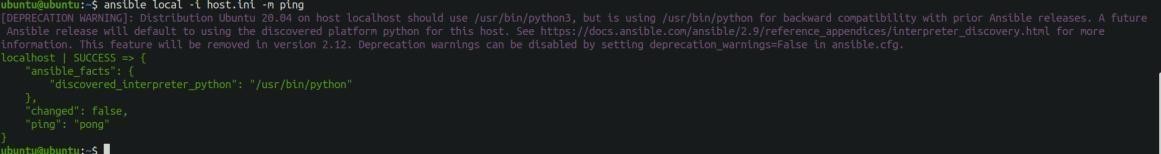


Step 3: Check version:





1. Ping the remote host ansible local -i host.ini -m ping

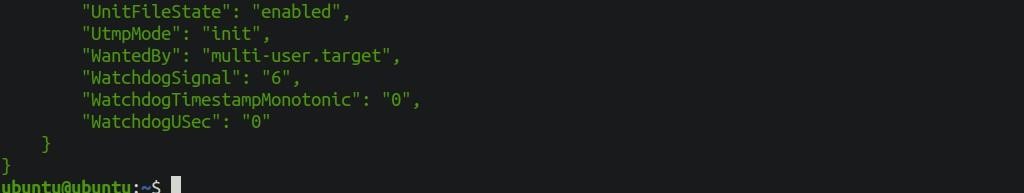


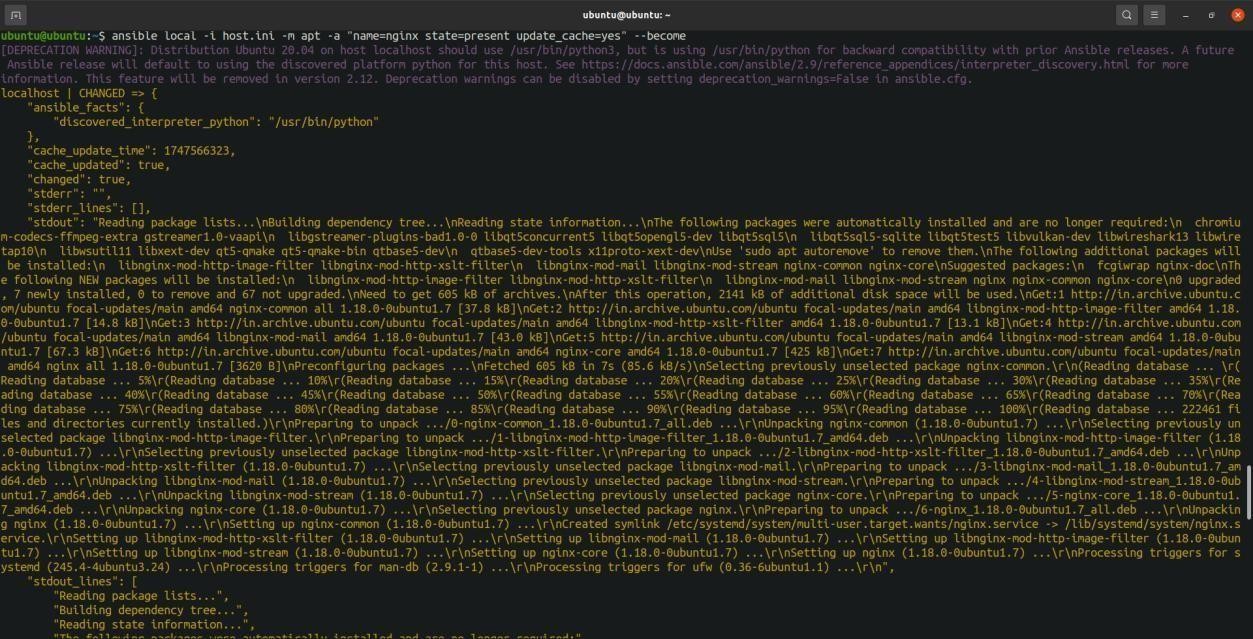
1. Check uptime

ansible local -i host.ini -a "uptime"

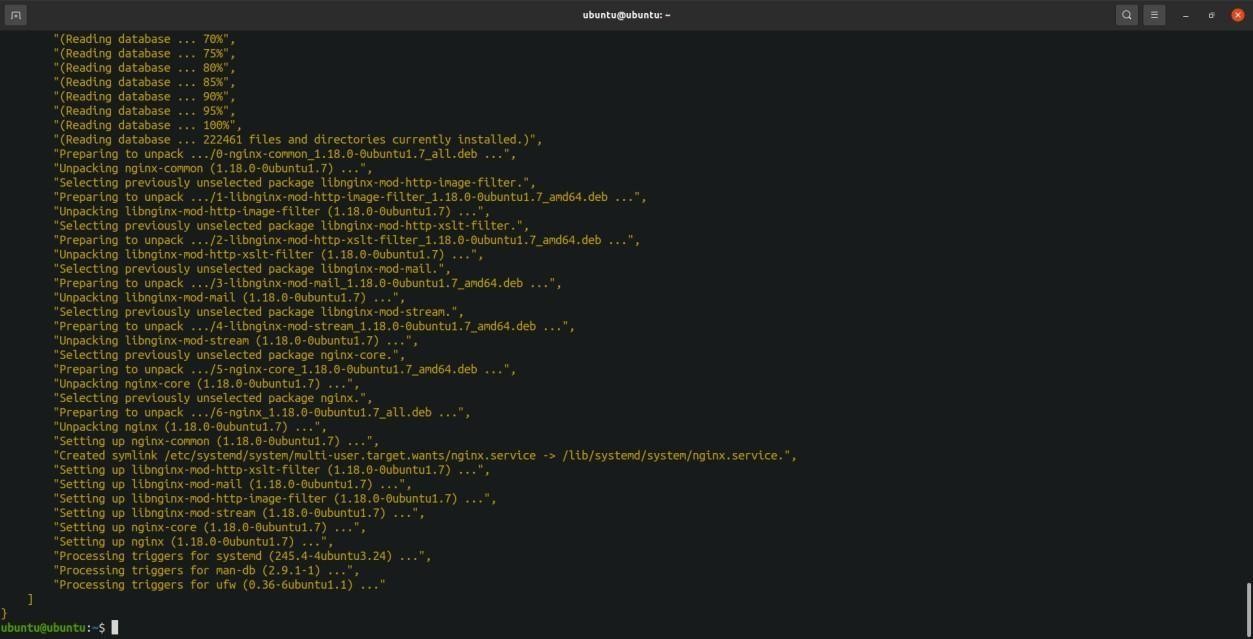


1. Install a package ansible local -i host.ini -m apt -a "name=nginx state=present update\_cache=yes" –become





\



1. Start a service ansible local -i host.ini -m service -a "name=nginx state=started" –become

**Practical 12**

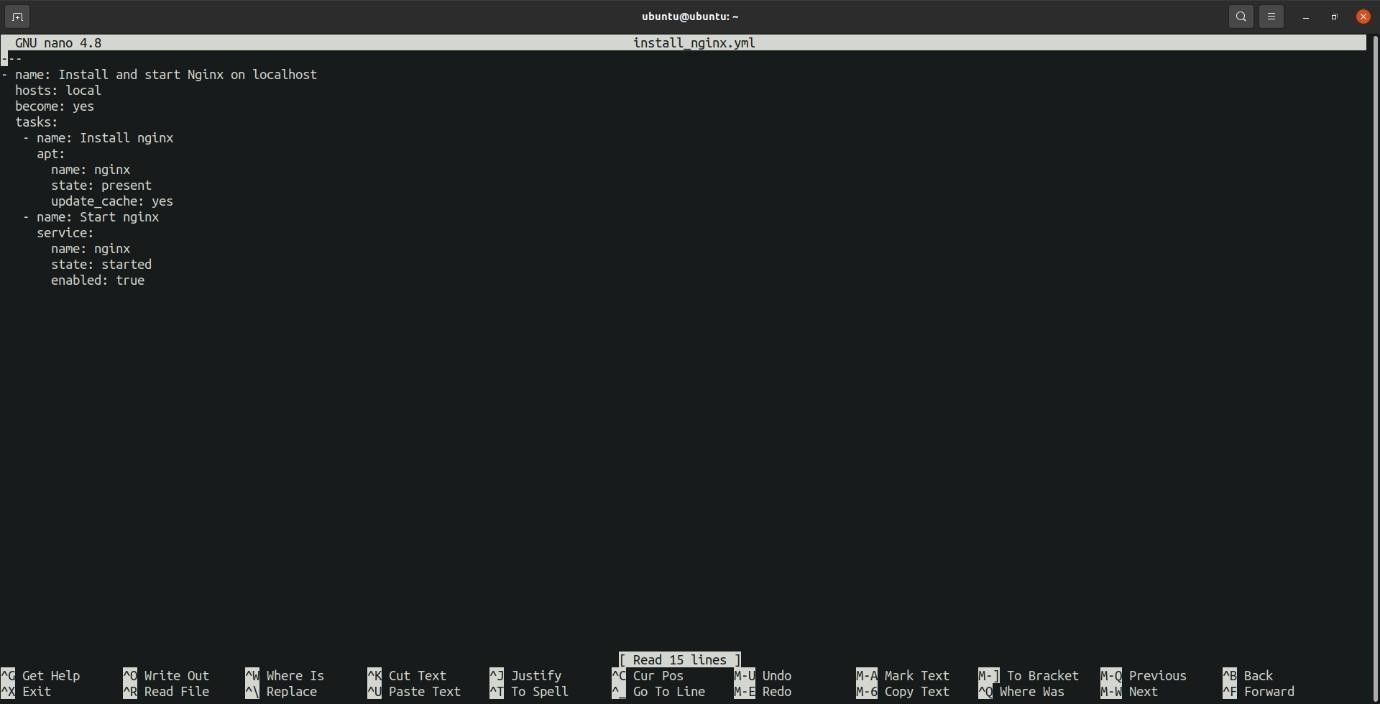
**AIM: Demonstrate Ansible Playbooks**

**Install and Start Nginx**

install\_nginx.yml:

* name: Install and start Nginx on web servers hosts: webservers become: true tasks:
* name: Install Nginx apt: name: nginx state: present update\_cache: yes
* name: Start Nginx service: name: nginx state: started enabled: true





Run the Playbook:

ansible-playbook -i hosts.ini install\_nginx.yml



