# PRACTICAL 1

**AIM: Demonstrate basic git commands**

1. Set-up Git (One-time setup)
    - **git config --global user.name “”** -- Set your name for commits.
    - **git config --global user.email “”**-- Set your email for commits.
2. 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.
3. **Work on a New Feature**
    - **git branch master:-** Create a new branch for a feature (e.g., feature-1).
    - **git checkout master** :- Switch to the new branch.
4. **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.
5. **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.
6. **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**
      Push the main branch to the remote repository for the first time.
7. **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.

```
admin@DESKTOP-AMCRBC2 MINGW64 ~/c24077 (main)
$ git pull origin main
From https://github.com/snehalparab27/demo
 * branch            main       -> FETCH_HEAD
Updating 1672d88..7df1fad
Fast-forward
 demo.txt | 2 ++
 1 file changed, 2 insertions(+)
```

# PRACTICAL 2

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

**Step 1: Initial Setup**

1. **Create a Git repository (if not already created)**

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

git init

MINGW64:/c/Users/admin/c24077

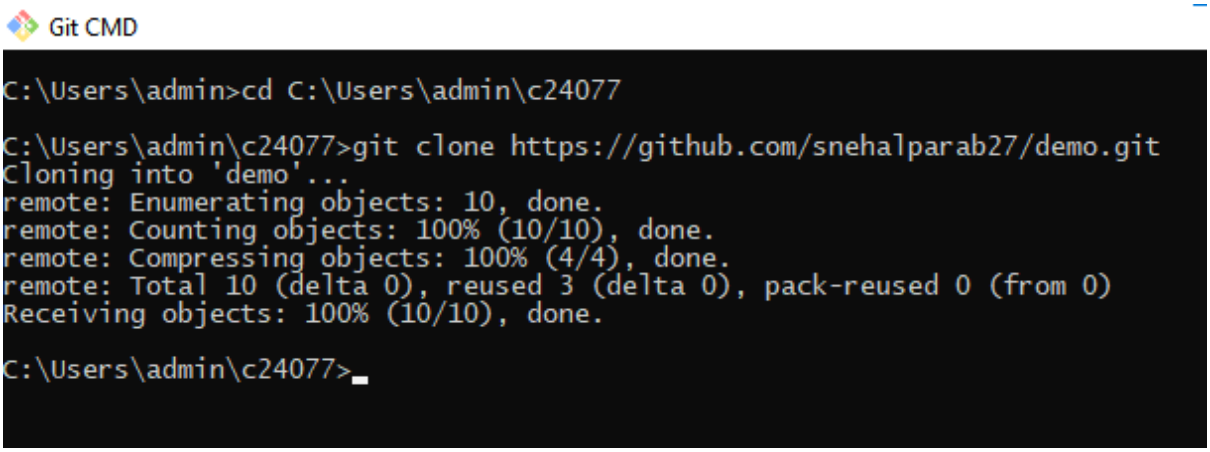
```
admin@DESKTOP-AMCRBC2 MINGW64 ~/c24077 (master)
$ git init
Reinitialized existing Git repository in C:/Users/admin/c24077/.git/
```

2. **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>


Git CMD
C:\Users\admin>cd C:\Users\admin\c24077
C:\Users\admin\c24077>git clone https://github.com/snehalparab27/demo.git
Cloning into 'demo'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 10 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (10/10), done.
C:\Users\admin\c24077>

**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 (*).

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

   git branch <branch-name>

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (main)
$ git branch new_branch
```

3. **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>

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (main)
$ git checkout new_branch
Switched to branch 'new_branch'

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git checkout -b branch1
Switched to a new branch 'branch1'
```

4. **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**

1. **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.).

2. **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)

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

   git commit -m "Description of changes"

   git

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git add myfile.txt
fatal: pathspec 'myfile.txt' did not match any files

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git add myfile.txt
fatal: pathspec 'myfile.txt' did not match any files

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ ls
hello.txt  new.txt  test.txt

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git add hello.txt

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git add .

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git status
On branch branch1
nothing to commit, working tree clean

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git commit -m "decsription of changes"
On branch branch1
nothing to commit, working tree clean
```

**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

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (branch1)
$ git checkout master
branch 'master' set up to track 'origin/master'.
Switched to a new branch 'master'

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git pull origin
Already up to date.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git merge branch1
fatal: refusing to merge unrelated histories

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git init
Reinitialized existing Git repository in C:/Users/newgit/myrepo_sam79/.git/

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git merge branch1
fatal: refusing to merge unrelated histories

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git branch
  branch1
  main
* master
  new_branch

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git merge new_branch
fatal: refusing to merge unrelated histories
```

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

   git pull origin main

3. **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.

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git merge branch1 --allow-unrelated-histories
Auto-merging hello.txt
CONFLICT (add/add): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master|MERGING)
$ git add hello.txt
```

**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>

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

   git add <resolved-file>

5. **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

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

   git push origin --delete <branch-name>

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master|MERGING)
$ git commit -m "Merged branch1 into master allowing unrelated histories"
[master f756b20] Merged branch1 into master allowing unrelated histories

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 401 bytes | 401.00 KiB/s, done.
Total 3 (delta 0), reused 1 (delta 0), pack-reused 0 (from 0)
To https://github.com/Shramikapatne20/myrepo_sam79.git
   8c490e8..f756b20  master -> master

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git branch -d branch1
Deleted branch branch1 (was 23cf05c).

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ ^[[200~git push origin --delete branch1
bash: $'\E[200~git': command not found
```

**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

2. **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 on your machine.
2. **Create a GitHub Account**: Go to GitHub 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

**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.
3. **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

```
hp@DESKTOP-RL6G3J5 MINGW64 ~ (master)
$ cd "C:\Users\newgit"

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit (master)
$ git clone "https://github.com/Shramikapatne20/myrepo_sam79.git"
fatal: destination path 'myrepo_sam79' already exists and is not an empty direct
ory.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit (master)
$ cd myrepo_sam79
```

**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

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git checkout new_branch
Switched to branch 'new_branch'

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git add .

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git commit -m "Added some text"
[new_branch 749e265] Added some text
 1 file changed, 1 insertion(+)
```

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

3. **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

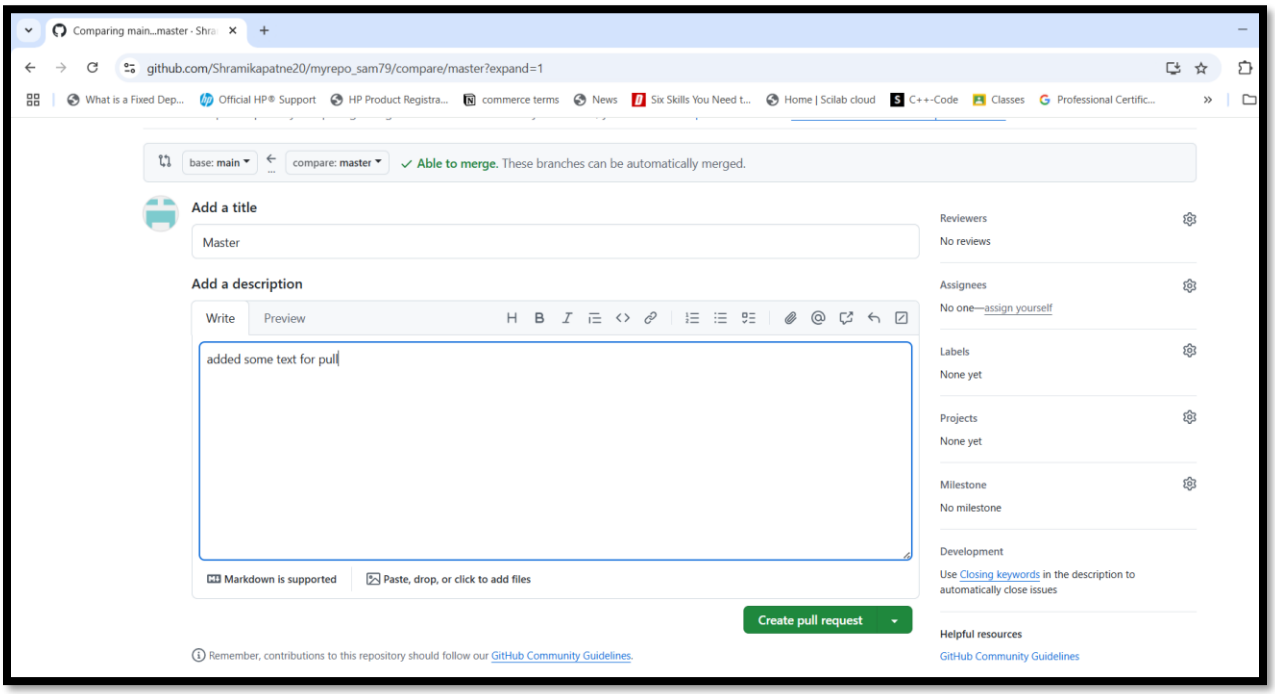
   This uploads your local changes to your GitHub repository.

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git push origin master
Everything up-to-date
```

**Step 5: Create a Pull Request (PR)**

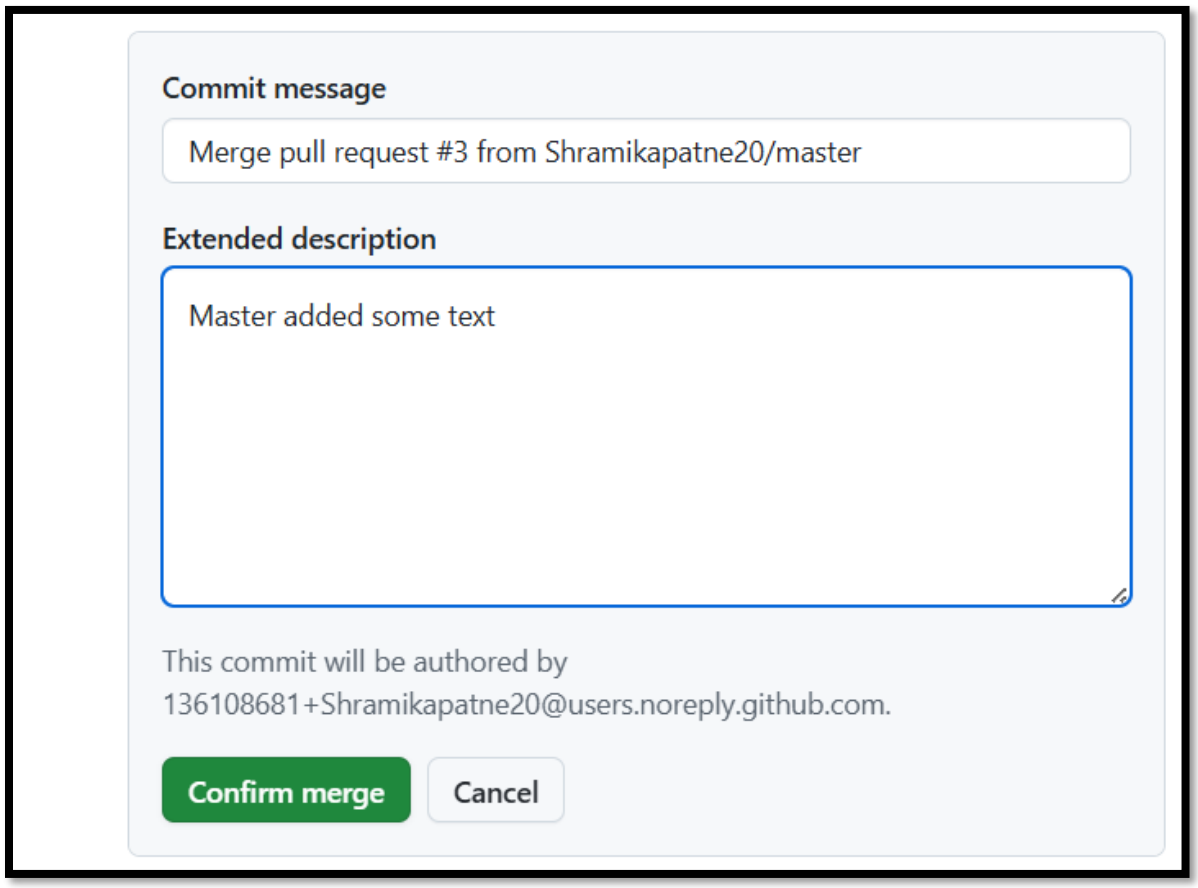
To contribute your changes back to the original repository, you'll need to open a pull request (PR).

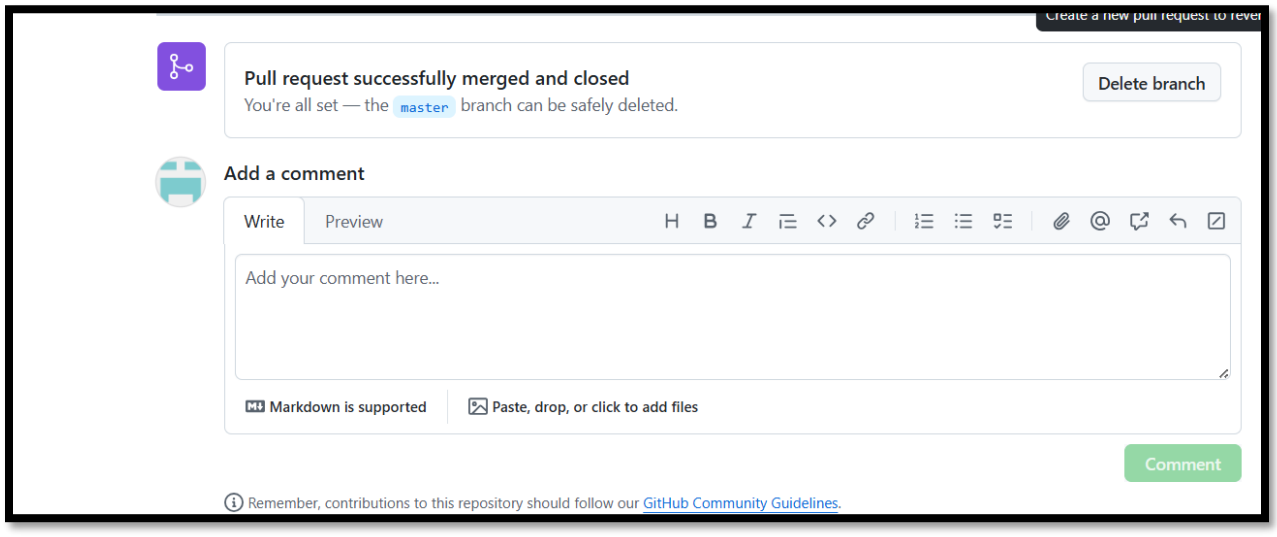
1. **Go to the GitHub Repository**: Visit the repository where you want to make the changes.
2. **Create a Pull Request**: On GitHub, you'll see an option to compare your branch (e.g., feature/your-feature) with the main branch of the repository (e.g., main). Click **New Pull Request**.
3. **Fill Out the Pull Request Form**:
    - Add a **title** and **description** explaining the changes you've made.
    - Review your changes.
    - Click **Create Pull Request**.




4. **Code Review**: The repository maintainer (or other collaborators) will review your changes. They might ask for changes or approve the PR.

5. **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.

2. **Fetch the Latest Changes from the Original Repository**:

   git fetch upstream  # Fetch the changes from the original repo

3. **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

4. **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.

1. **Switch to Your Local Main Branch**:

   git checkout main

2. **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.

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git remote add upstream https://github.com/snehalparab27/demo.git

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git fetch upstream
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 24 (delta 1), reused 5 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (24/24), 8.28 KiB | 184.00 KiB/s, done.
From https://github.com/snehalparab27/demo
 * [new branch]      branch1    -> upstream/branch1
 * [new branch]      main       -> upstream/main

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git push origin master
Everything up-to-date

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (main)
$ git pull origin master
From https://github.com/Shramikapatne20/myrepo_sam79
 * branch            master     -> FETCH_HEAD
Updating 23cf05c..f756b20
Fast-forward
 hello.txt  | 2 +-
 myfile.txt | 1 +
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 myfile.txt
```

# 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:

**Install Git**: If you haven't already, download and install Git on your machine.

**Create a GitHub Account**: Go to GitHub and create an account if you don't have one.

**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

**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.

4. **Find a Repository to Clone**: Visit the repository page on GitHub (e.g., https://github.com/username/repository) and click the green **Code** button.
5. **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.
6. **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

```
hp@DESKTOP-RL6G3J5 MINGW64 ~ (master)
$ cd "C:\Users\newgit"

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit (master)
$ git clone "https://github.com/Shramikapatne20/myrepo_sam79.git"
fatal: destination path 'myrepo_sam79' already exists and is not an empty direct
ory.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit (master)
$ cd myrepo_sam79
```

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

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

2. **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

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git checkout new_branch
Switched to branch 'new_branch'

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git add .

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git commit -m "Added some text"
[new_branch 749e265] Added some text
 1 file changed, 1 insertion(+)
```

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

   git add .  # Stages all modified files

6. **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

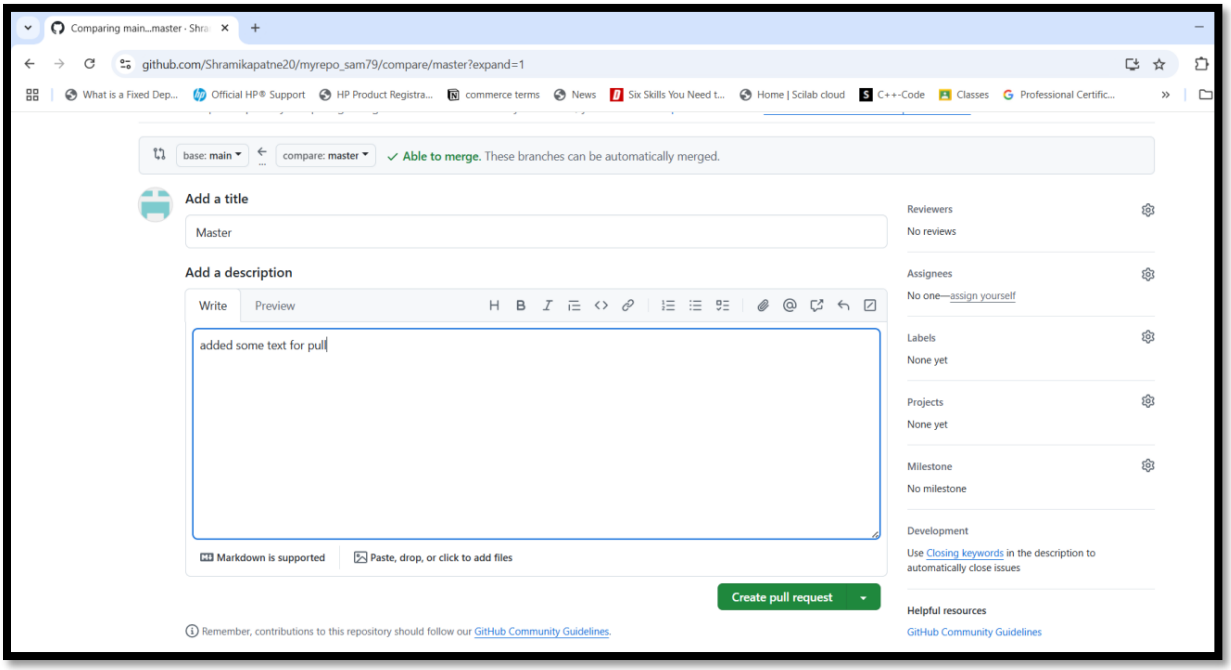
   This uploads your local changes to your GitHub repository.

```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git push origin master
Everything up-to-date
```

**Step 5: Create a Pull Request (PR)**

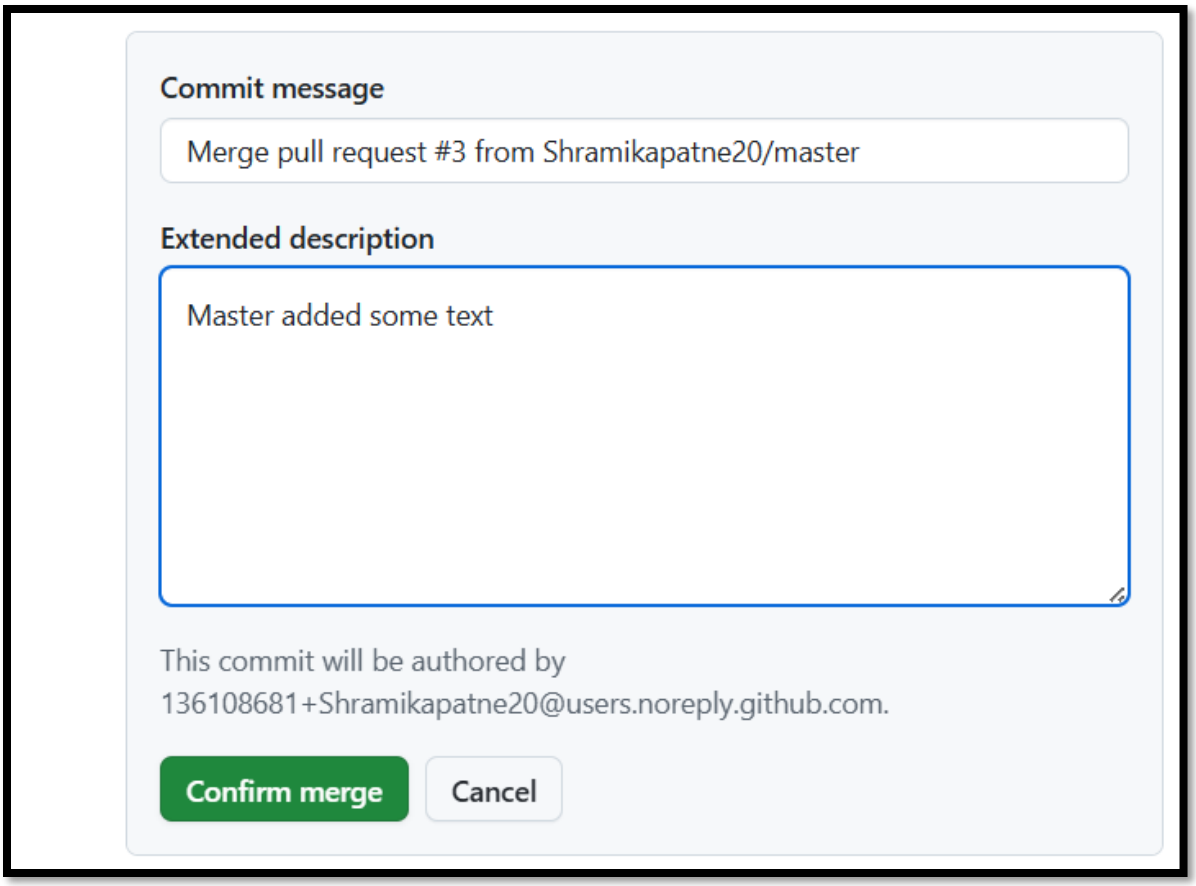
To contribute your changes back to the original repository, you'll need to open a pull request (PR).

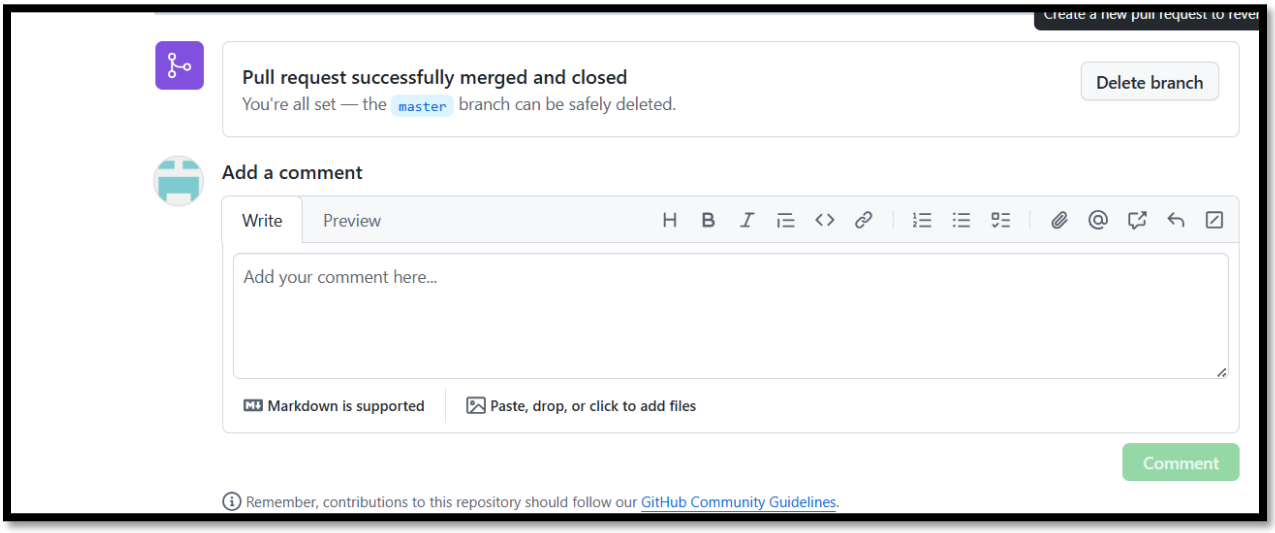
6. **Go to the GitHub Repository**: Visit the repository where you want to make the changes.
7. **Create a Pull Request**: On GitHub, you'll see an option to compare your branch (e.g., feature/your-feature) with the main branch of the repository (e.g., main). Click **New Pull Request**.
8. **Fill Out the Pull Request Form**:
    - Add a **title** and **description** explaining the changes you've made.
    - Review your changes.
    - Click **Create Pull Requ**




9. **Code Review**: The repository maintainer (or other collaborators) will review your changes. They might ask for changes or approve the PR.

10. **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:

5. **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.

6. **Fetch the Latest Changes from the Original Repository**:

   git fetch upstream  # Fetch the changes from the original repo

7. **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

8. **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.

3. **Switch to Your Local Main Branch**:

   git checkout main

4. **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:

5. **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).
6. **Create Feature Branches**: Always create a new branch for each feature or bug fix. This avoids conflicts and keeps the history clean.
7. **Review Pull Requests**: If you're reviewing someone else's PR, ensure you provide feedback and approve it once you're satisfied.
8. **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.

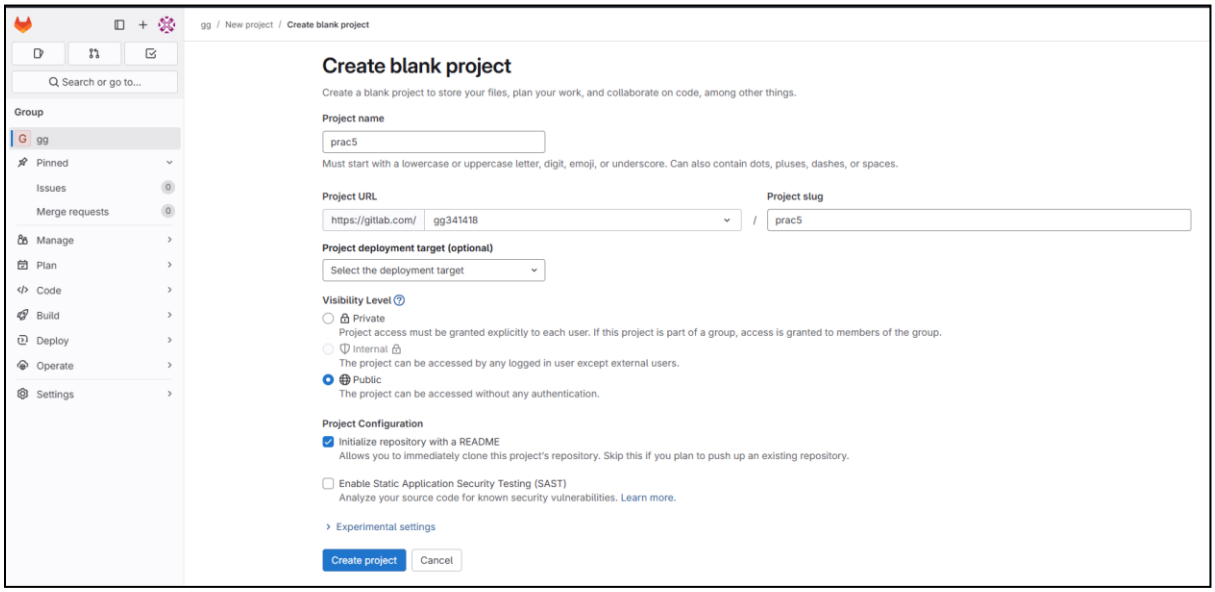
```
hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git remote add upstream https://github.com/snehalparab27/demo.git

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git fetch upstream
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 24 (delta 1), reused 5 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (24/24), 8.28 KiB | 184.00 KiB/s, done.
From https://github.com/snehalparab27/demo
 * [new branch]      branch1    -> upstream/branch1
 * [new branch]      main       -> upstream/main

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (new_branch)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git push origin master
Everything up-to-date

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (master)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

hp@DESKTOP-RL6G3J5 MINGW64 /c/Users/newgit/myrepo_sam79 (main)
$ git pull origin master
From https://github.com/Shramikapatne20/myrepo_sam79
 * branch            master     -> FETCH_HEAD
Updating 23cf05c..f756b20
Fast-forward
 hello.txt  | 2 +-
 myfile.txt | 1 +
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 myfile.txt
```

# 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.**




4. **Create a file (index.html):**

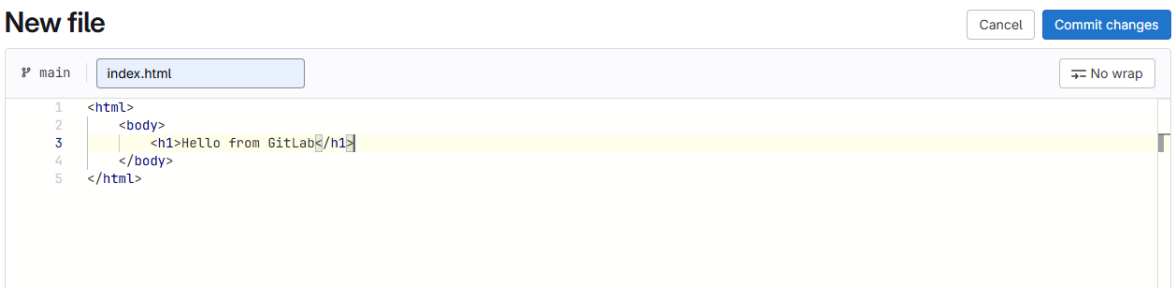
**<html>**

**<body>**

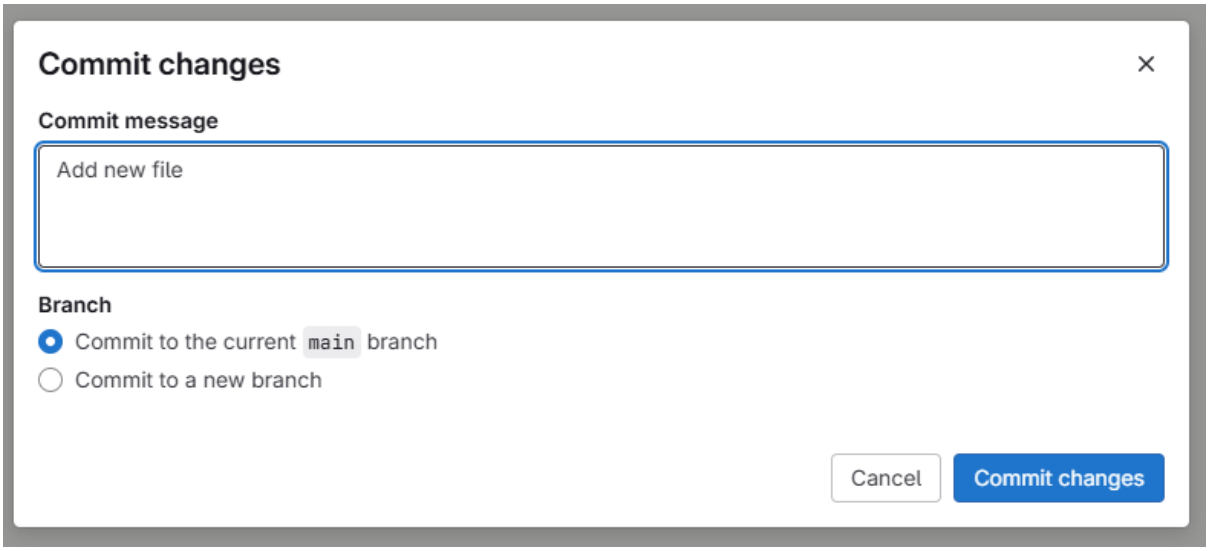
**<h1>Hello from GitLab</h1>**

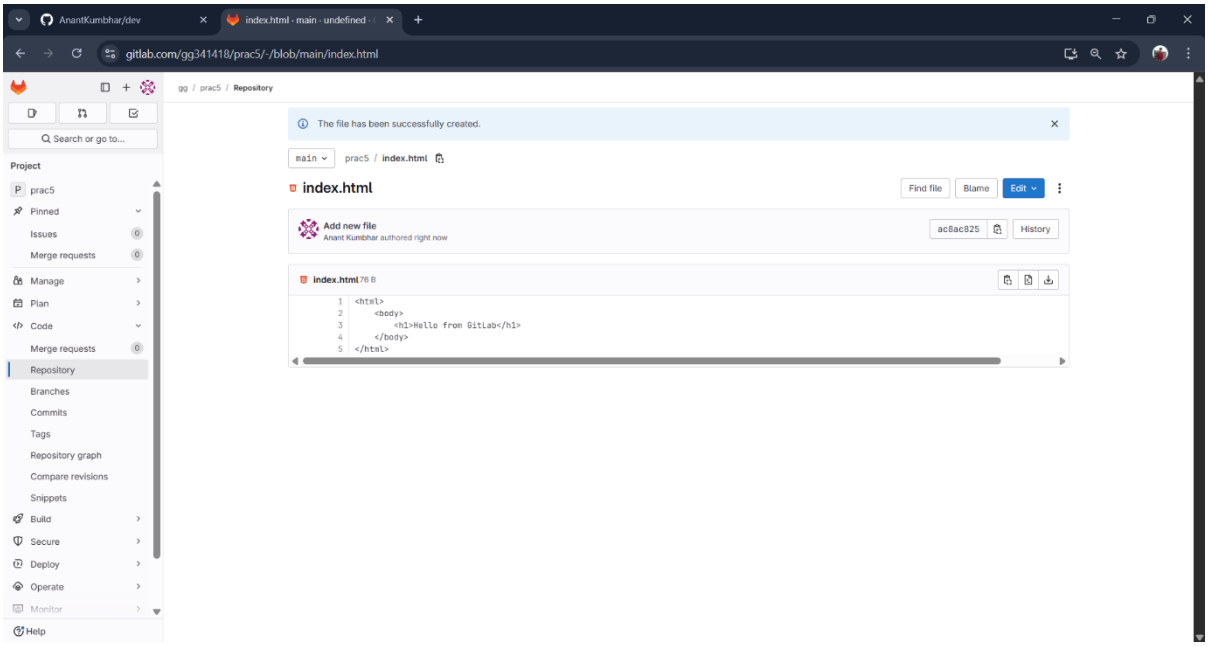
**</body>**

**</html>**



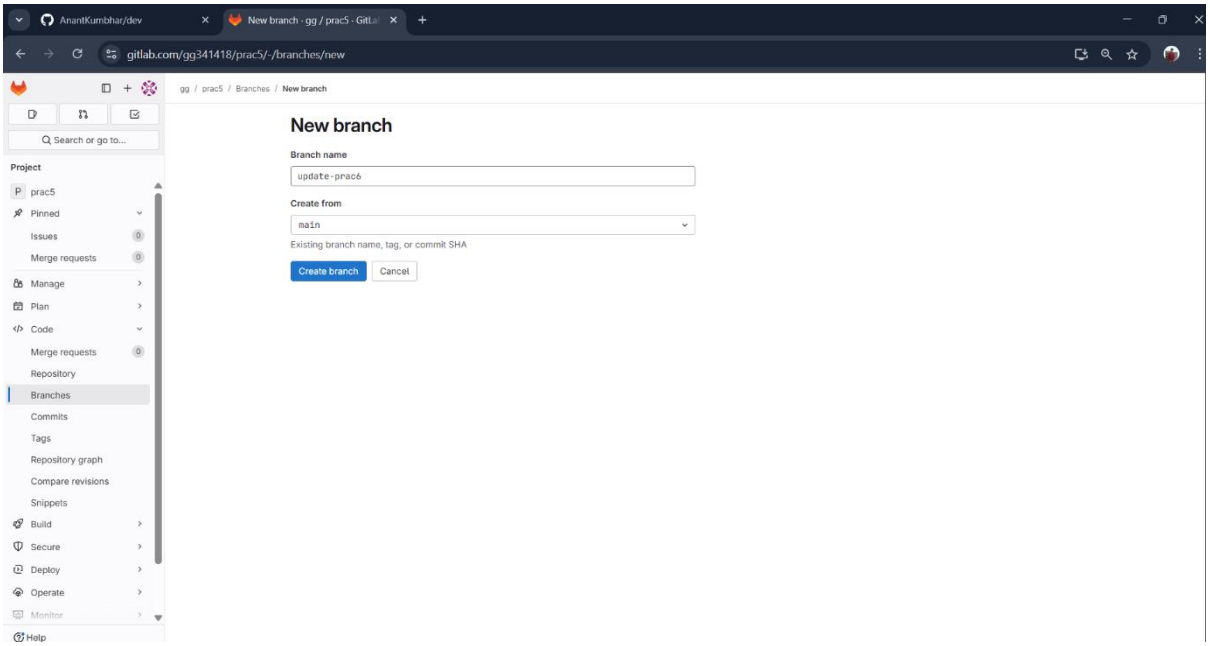

**5. Click Commit and push changes.**


Commit changes
Commit message
Add new file
Branch
Commit to the current main branch
Commit to a new branch
Cancel
Commit changes

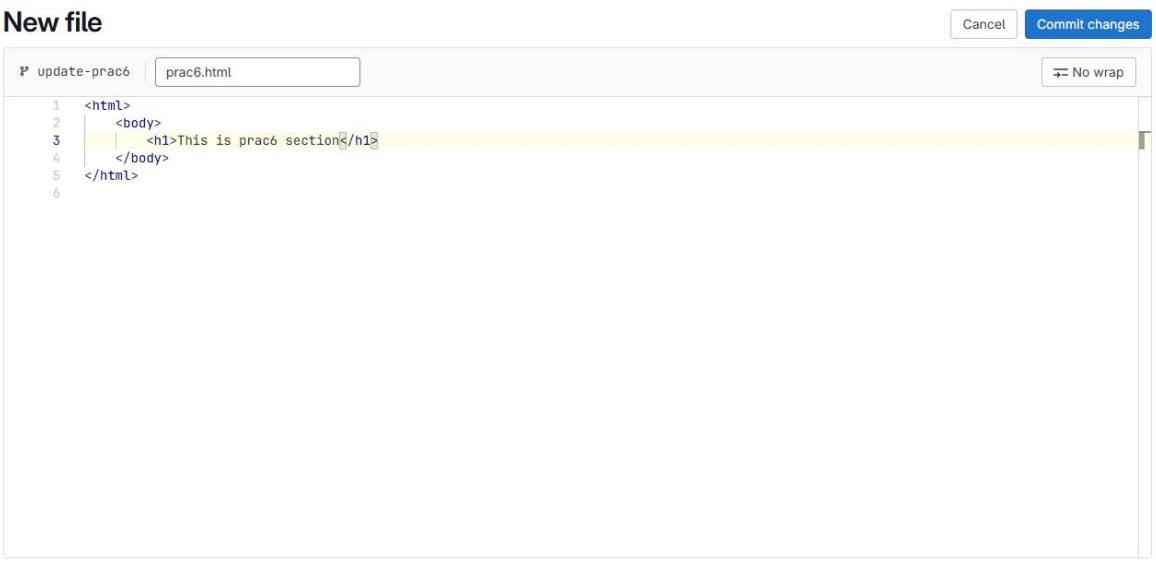

AnantKumbhar/dev
index.html · main · undefined
gitlab.com/gg341418/prac5/-/blob/main/index.html
gg / prac5 / Repository
The file has been successfully created.
main
prac5 / index.html
index.html
Find file
Blame
Edit
Add new file
Anant Kumbhar authored right now
ac8ac825
History
index.html 76 B
<html>
<body>
<h1>Hello from GitLab</h1>
</body>
</html>
Search or go to...
Project
prac5
Pinned
Issues
Merge requests
Manage
Plan
Code
Merge requests
Repository
Branches
Commits
Tags
Repository graph
Compare revisions
Snippets
Build
Secure
Deploy
Operate
Monitor
Help

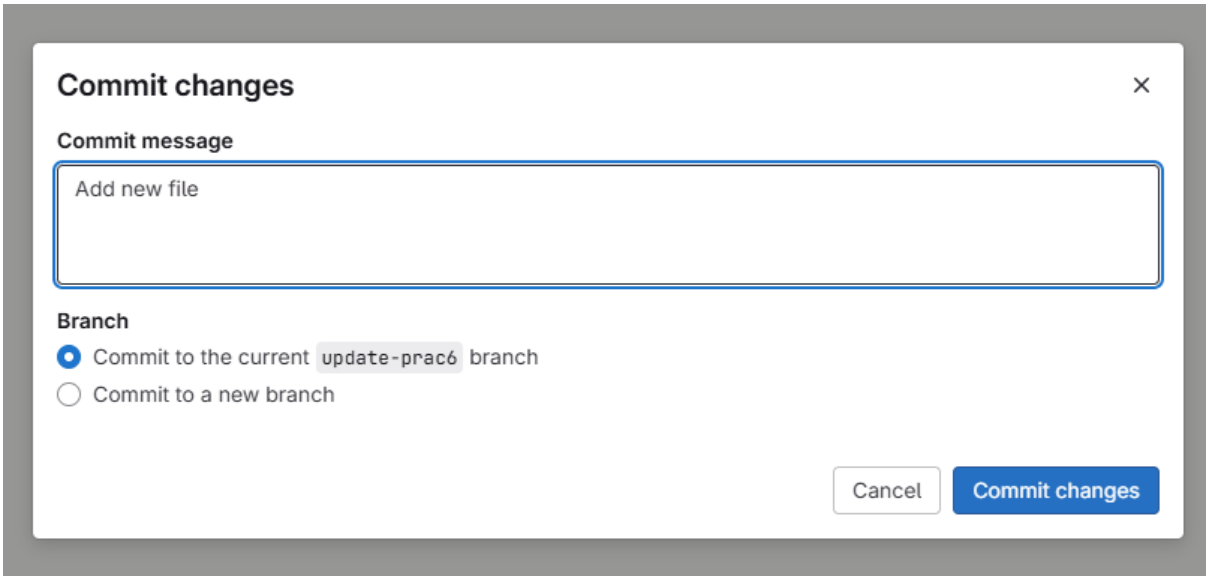
## Performing merge requests using GitLab

### 1. Create a new branch in Web IDE.

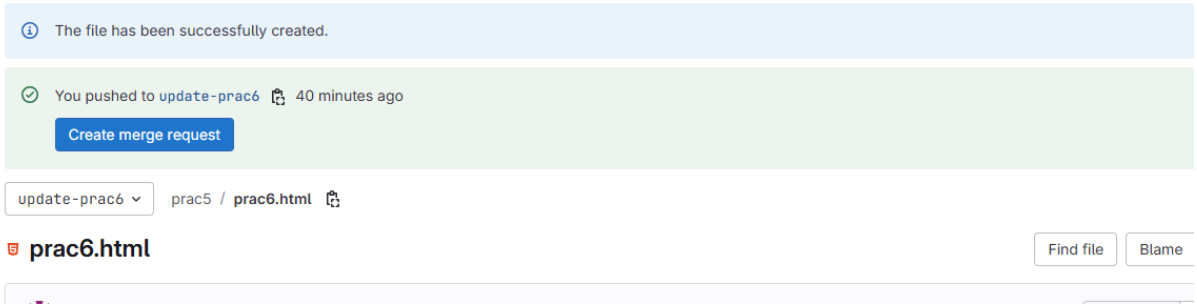



### 2. Add /Edit a File and Commit

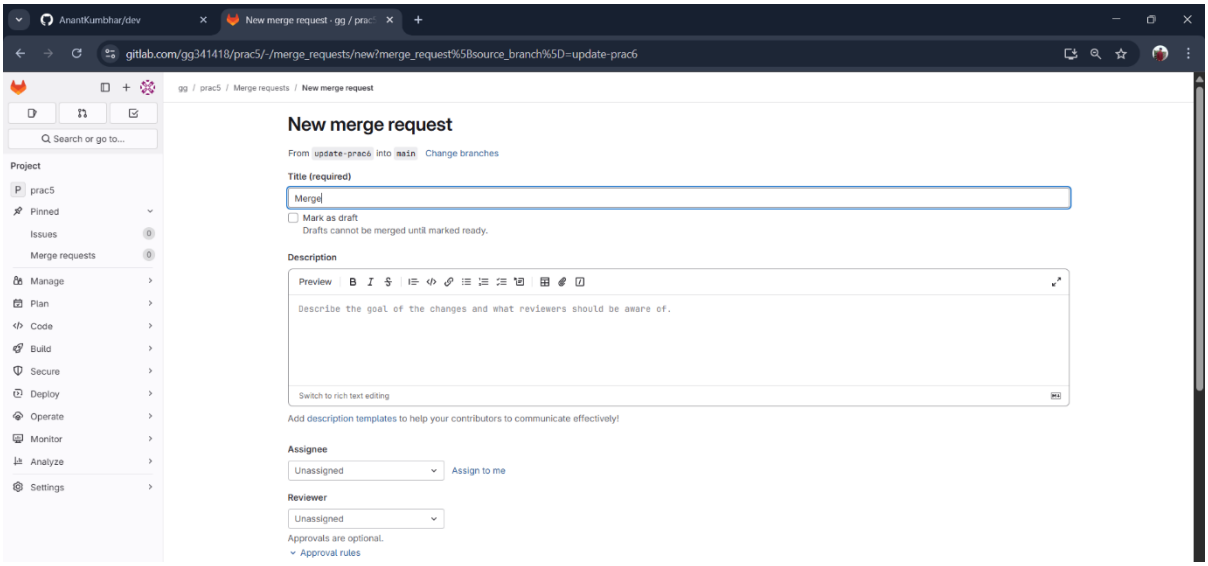


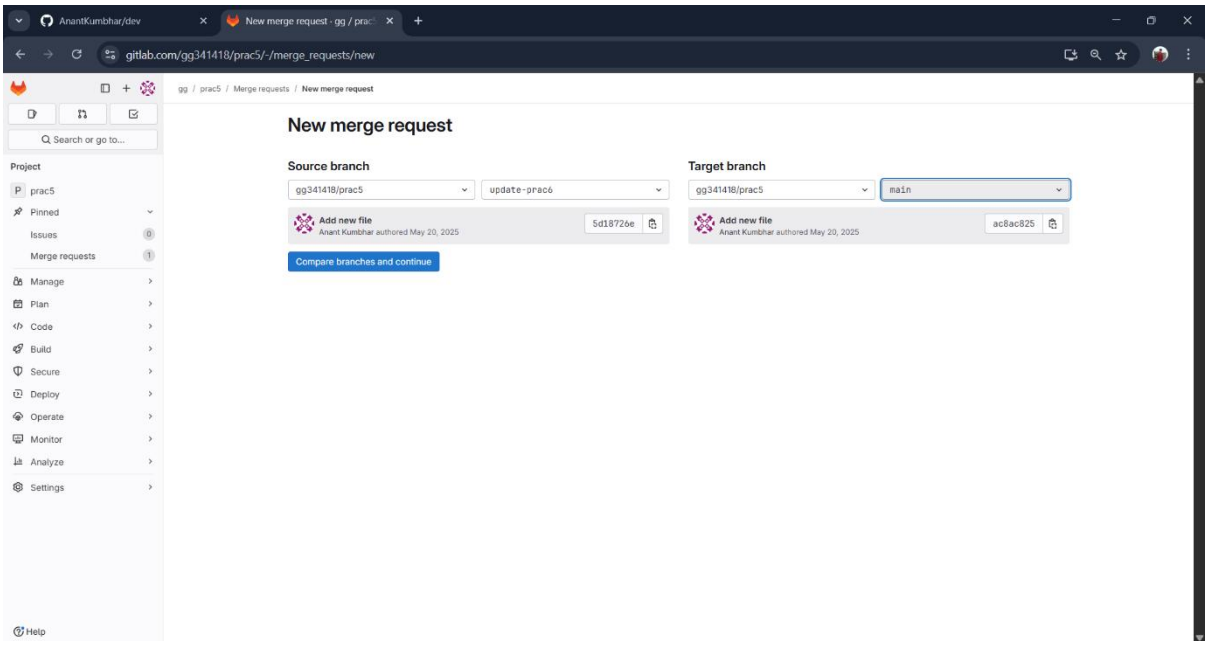



## 3. Click on merge Request > New Merge request

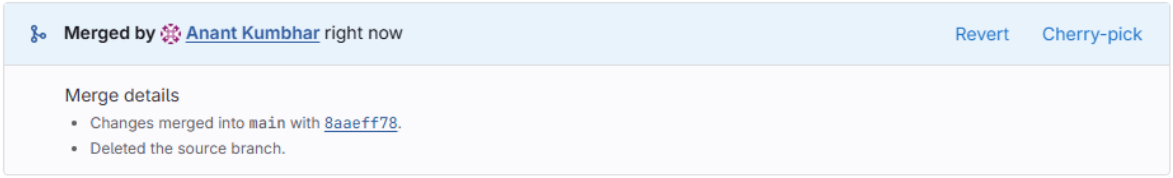



## 4. Select source and target branches






## 5. Submit and merge after review



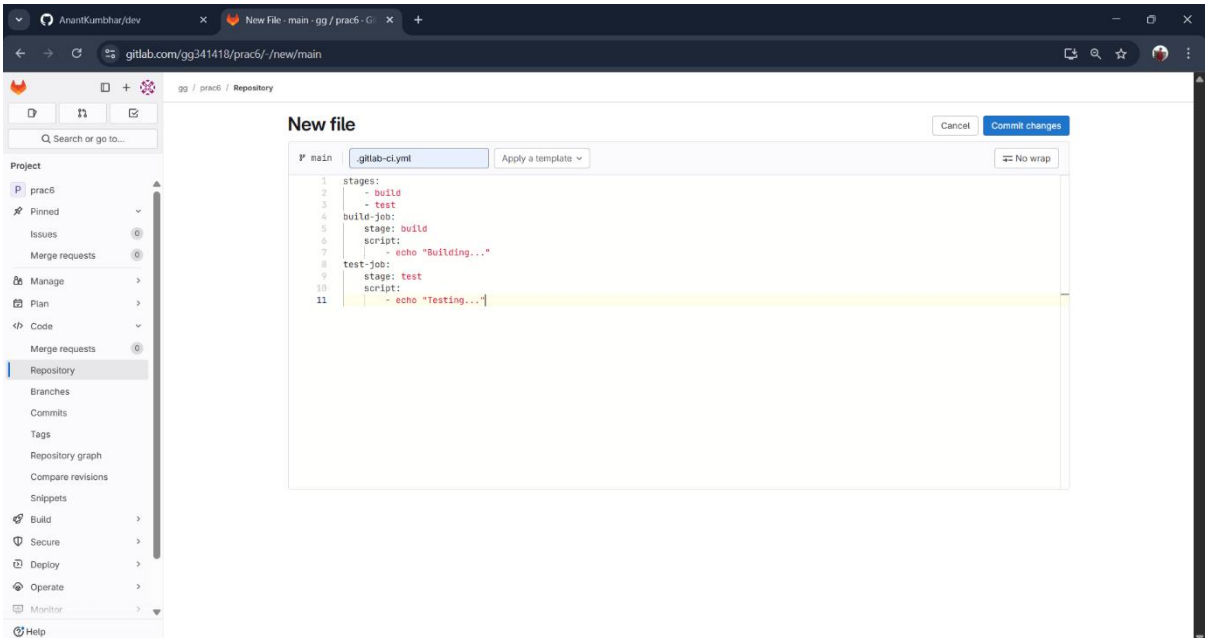
# Practical 6

**Aim: Demonstrate the CI/CD workflow in GitLab using .py, .bash, .java file**

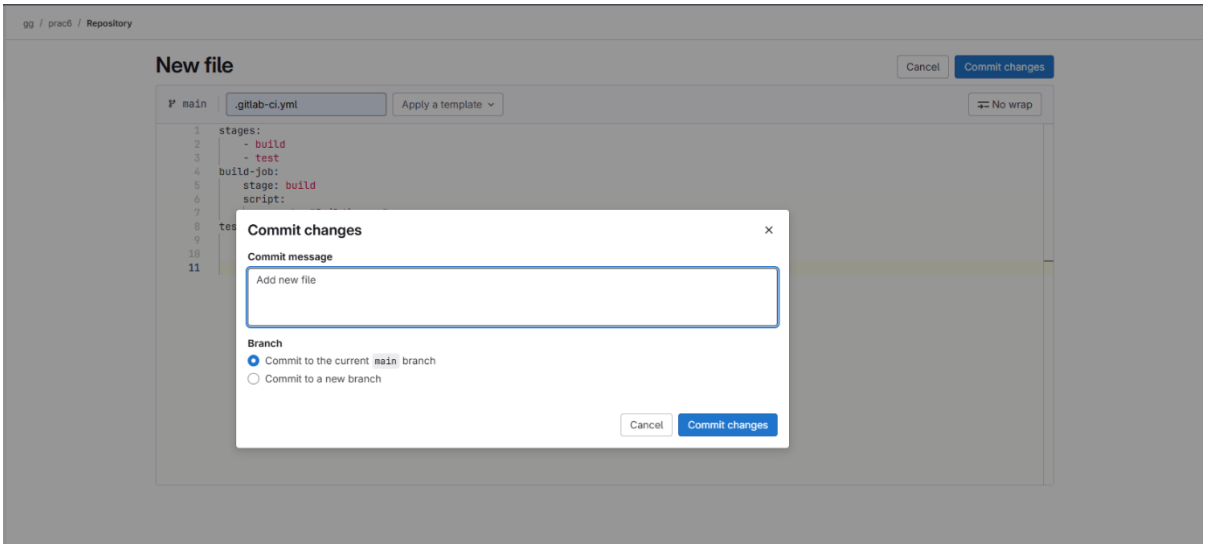
**Steps:**

**1. In your repo, create .gitlab-ci.yml:**

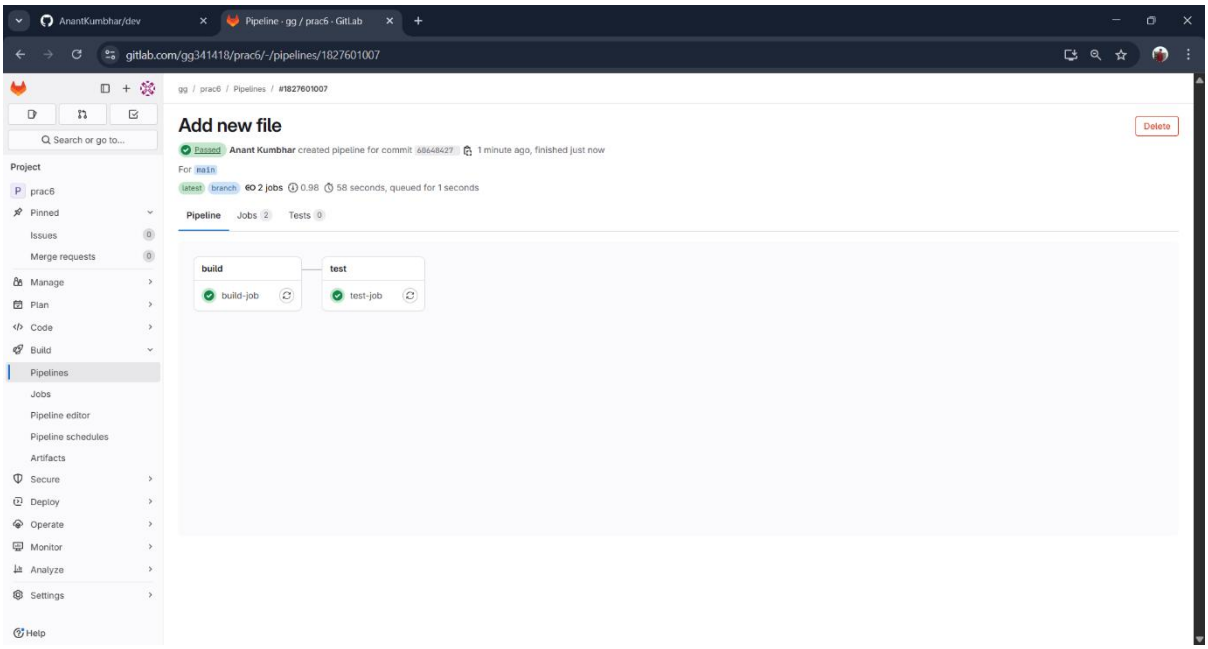
```
stages:
    - build
    - test
build-job:
    stage: build
    script:
        - echo "Building..."
test-job:
    stage: test
    script:
        - echo "Testing..."
```




**2. Commite and push**




## 3. Go To Bulid Pipeline and View the build/test stages



## build-job

Passed Started 2 minutes ago by Anant Kumbhar

Search visible log output

```
Running with gitlab-runner 17.10.0~pre.41.g5c23fd8e (5c23fd8e)
  on blue-4.saas-linux-small-amd64.runners-manager.gitlab.com/default J2nyww-s, system ID: s_cf1798852952
Preparing the "docker+machine" executor                                                                    00:21
Using Docker executor with image ruby:3.1 ...
Pulling docker image ruby:3.1 ...
Using docker image sha256:9981df1d883b246c27c62f8ccb9b57d3e07d14cee8092299e102b4a69c35ea61 for ruby:3.1 with digest ruby@sha256:91627f55e8969006aab67d15c9
2fb930500ff73948803da1330b8a853fecebb5 ...
Preparing environment                                                                                      00:05
Running on runner-j2nyww-s-project-70015884-concurrent-0 via runner-j2nyww-s-s-l-s-amd64-1747759872-2ac5077e...
Getting source from Git repository                                                                         00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/gg341418/prac6/.git/
Created fresh repository.
Checking out 60648427 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}" || echo 'Not a git repository; skipping'
Executing "step_script" stage of the job script                                                            00:00
Using docker image sha256:9981df1d883b246c27c62f8ccb9b57d3e07d14cee8092299e102b4a69c35ea61 for ruby:3.1 with digest ruby@sha256:91627f55e8969006aab67d15c9
2fb930500ff73948803da1330b8a853fecebb5 ...
$ echo "Building..."
Building...
Cleaning up project directory and file based variables                                                     00:01
Job succeeded
```

## test-job

Passed Started 1 minute ago by Anant Kumbhar

Search visible log output

```
Running with gitlab-runner 17.10.0~pre.41.g5c23fd8e (5c23fd8e)
  on blue-5.saas-linux-small-amd64.runners-manager.gitlab.com/default -AzERasQ, system ID: s_4cb09cee29e2
Preparing the "docker+machine" executor                                                                    00:20
Using Docker executor with image ruby:3.1 ...
Pulling docker image ruby:3.1 ...
Using docker image sha256:9981df1d883b246c27c62f8ccb9b57d3e07d14cee8092299e102b4a69c35ea61 for ruby:3.1 with digest ruby@sha256:91627f55e8969006aab67d15c9
2fb930500ff73948803da1330b8a853fecebb5 ...
Preparing environment                                                                                      00:04
Running on runner--azerasq-project-70015884-concurrent-0 via runner-azerasq-s-l-s-amd64-1747759901-d7593e3a...
Getting source from Git repository                                                                         00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/gg341418/prac6/.git/
Created fresh repository.
Checking out 60648427 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}" || echo 'Not a git repository; skipping'
Executing "step_script" stage of the job script                                                            00:01
Using docker image sha256:9981df1d883b246c27c62f8ccb9b57d3e07d14cee8092299e102b4a69c35ea61 for ruby:3.1 with digest ruby@sha256:91627f55e8969006aab67d15c9
2fb930500ff73948803da1330b8a853fecebb5 ...
$ echo "Testing..."
Testing...
Cleaning up project directory and file based variables                                                     00:00
Job succeeded
```

## CI/Cd for python

## Create script.py

## print("Hello NMITD!")

script.py 21 B

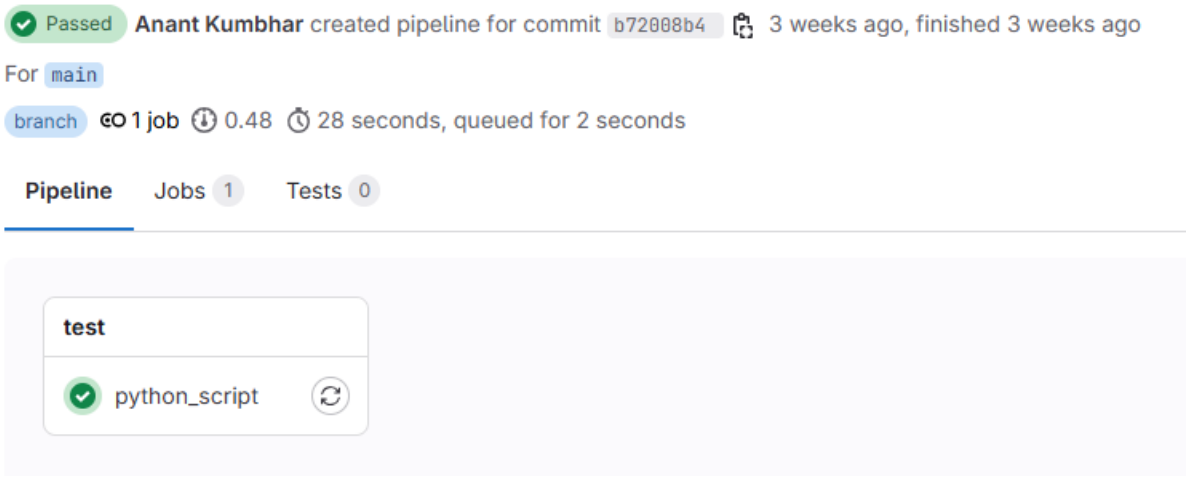
```
1  print("Hello NMITD!")
```

**Create .gitlab-ci.yml**

.gitlab-ci.yml 119 B

```yaml
stages:
    - test

python_script:
    stage: test
    image: python:3.10
    script:
        - python script.py
```

**Commit the changes  and bulid pipeline**




**CI/Cd for bash**

**Create basic.sh file**

basic.sh 112 B

```
echo "This is from bash script"
touch myfile.txt
echo "sample text" > myfile.txt
echo "this is end of script"
```

**Create .gitlab-ci.yml**

.gitlab-ci.yml 92 B

```yaml
stages:
    - build
bash_execute:
    stage: build
    script:
        - bash basic.sh
```

**CI/Cd for Java**

**Create demo.java**

demo.java 151 B

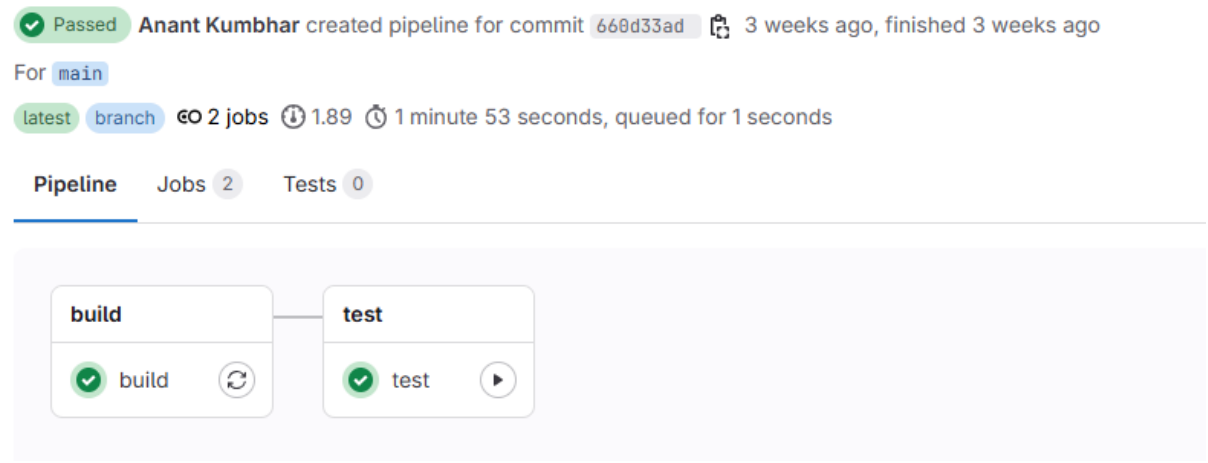
```java
class demo{
    public static void main(String a[]){
        System.out.println("Hello GG");
        System.out.println("gg guys helloo");
    }
}
```

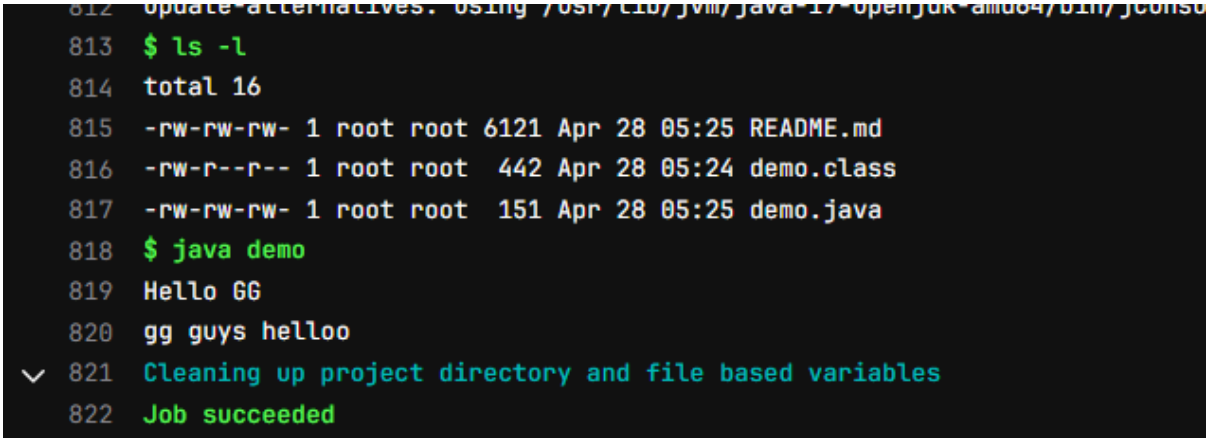
**Create .gitlab-ci.yml**

.gitlab-ci.yml 408 B

```yaml
stages:
    - build
    - test

before_script:
    - apt-get update && apt-get install -y openjdk-17-jdk

build:
    stage: build
    script:
        - javac demo.java
        - ls -ls

    artifacts:
        paths:
            - demo.class
    only:
        - main

test:
    stage: test
    when: manual
    script:
        - ls -l
        - java demo
    only:
        - main
```

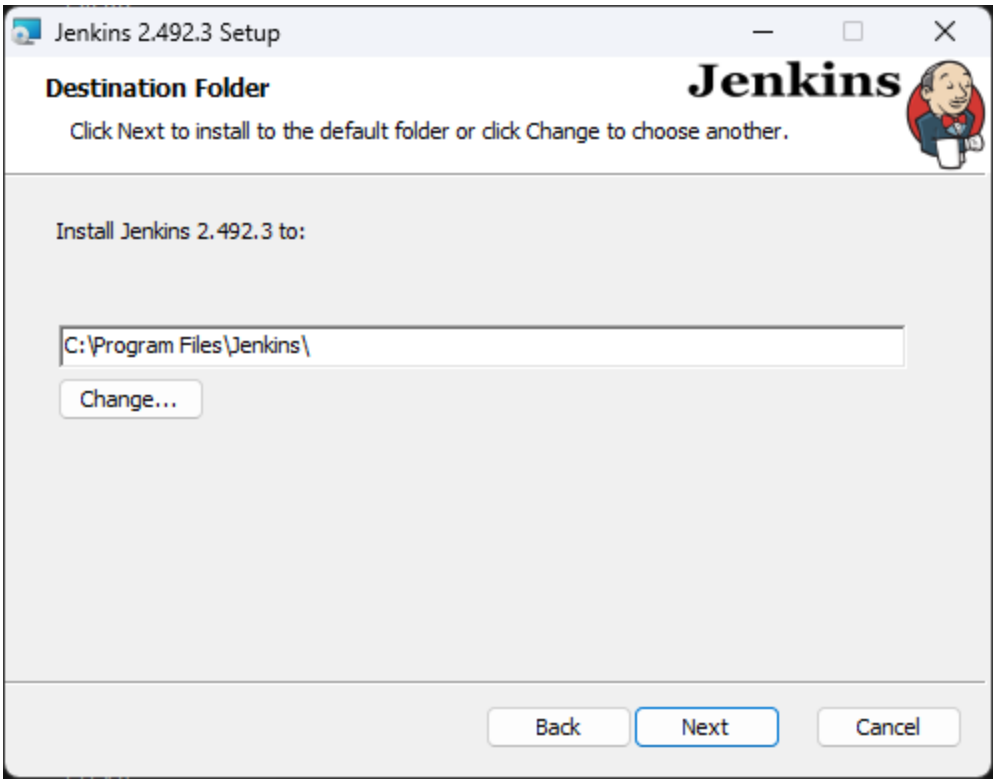
**Commite and build pipleline**

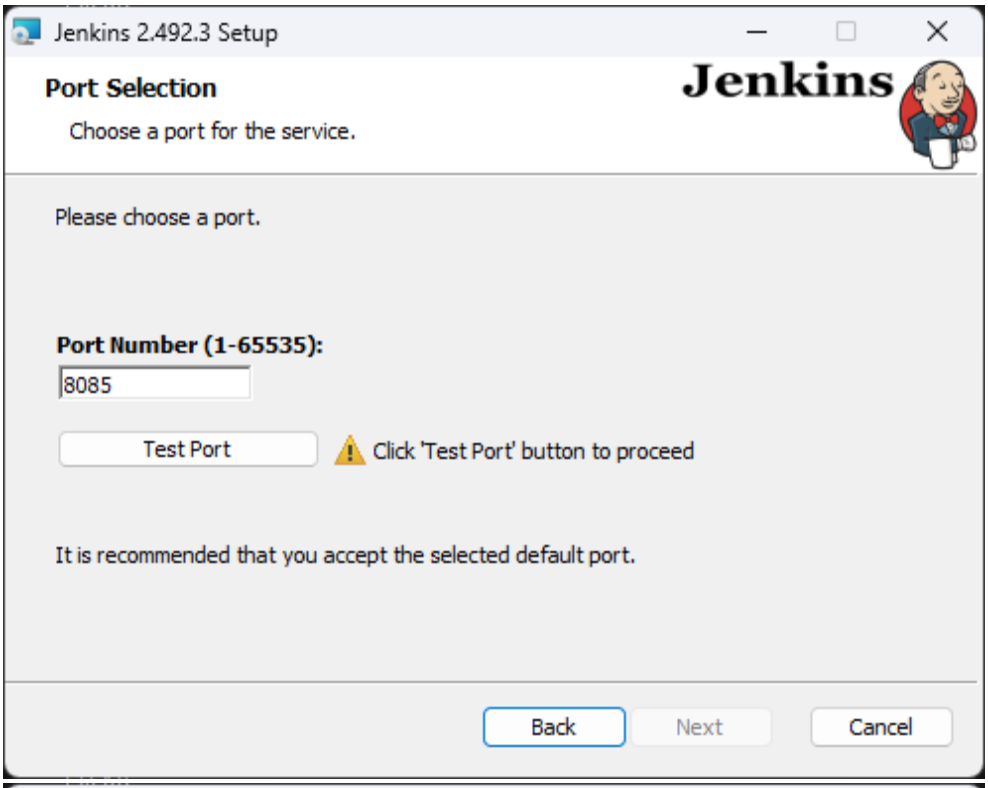

Passed
Anant Kumbhar created pipeline for commit 660d33ad
3 weeks ago, finished 3 weeks ago
For main
latest
branch
2 jobs
1.89
1 minute 53 seconds, queued for 1 seconds
Pipeline
Jobs 2
Tests 0
build
build
test
test

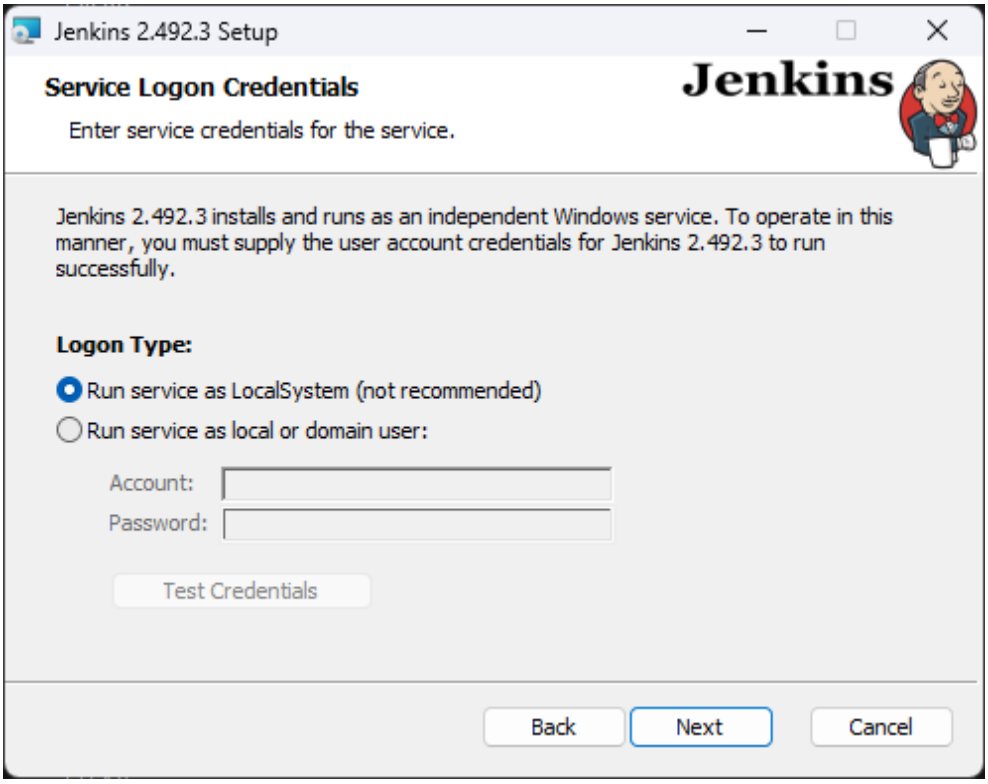

$ ls -l
total 16
-rw-rw-rw- 1 root root 6121 Apr 28 05:25 README.md
-rw-r--r-- 1 root root 442 Apr 28 05:24 demo.class
-rw-rw-rw- 1 root root 151 Apr 28 05:25 demo.java
$ java demo
Hello GG
gg guys helloo
Cleaning up project directory and file based variables
Job succeeded

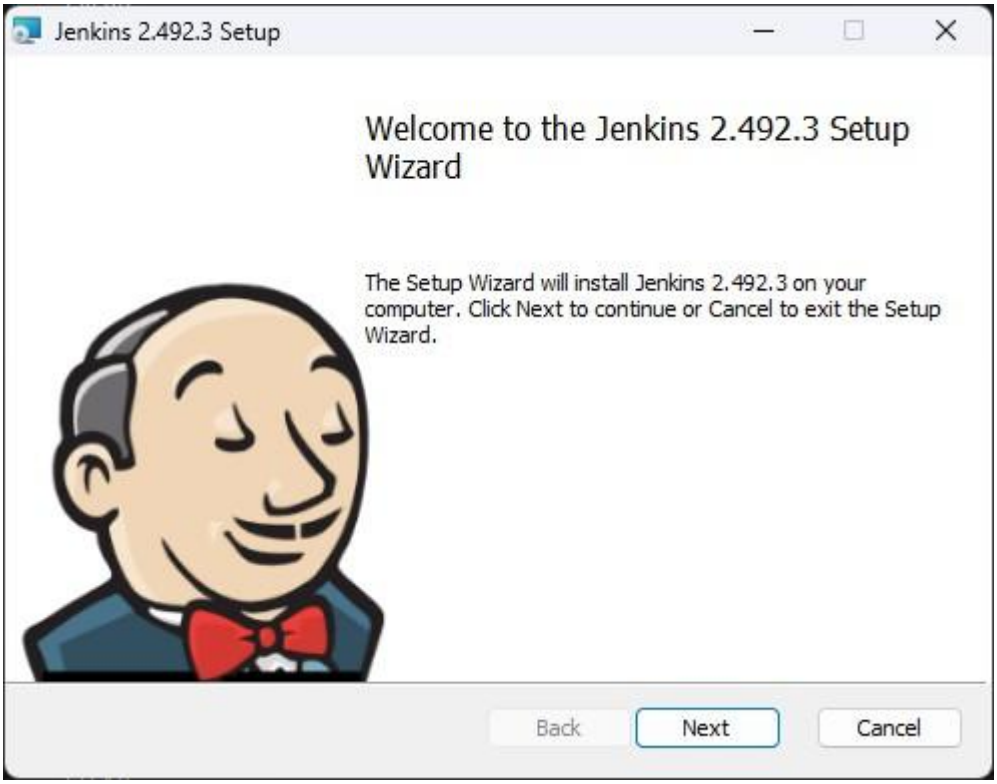
# Practical 7

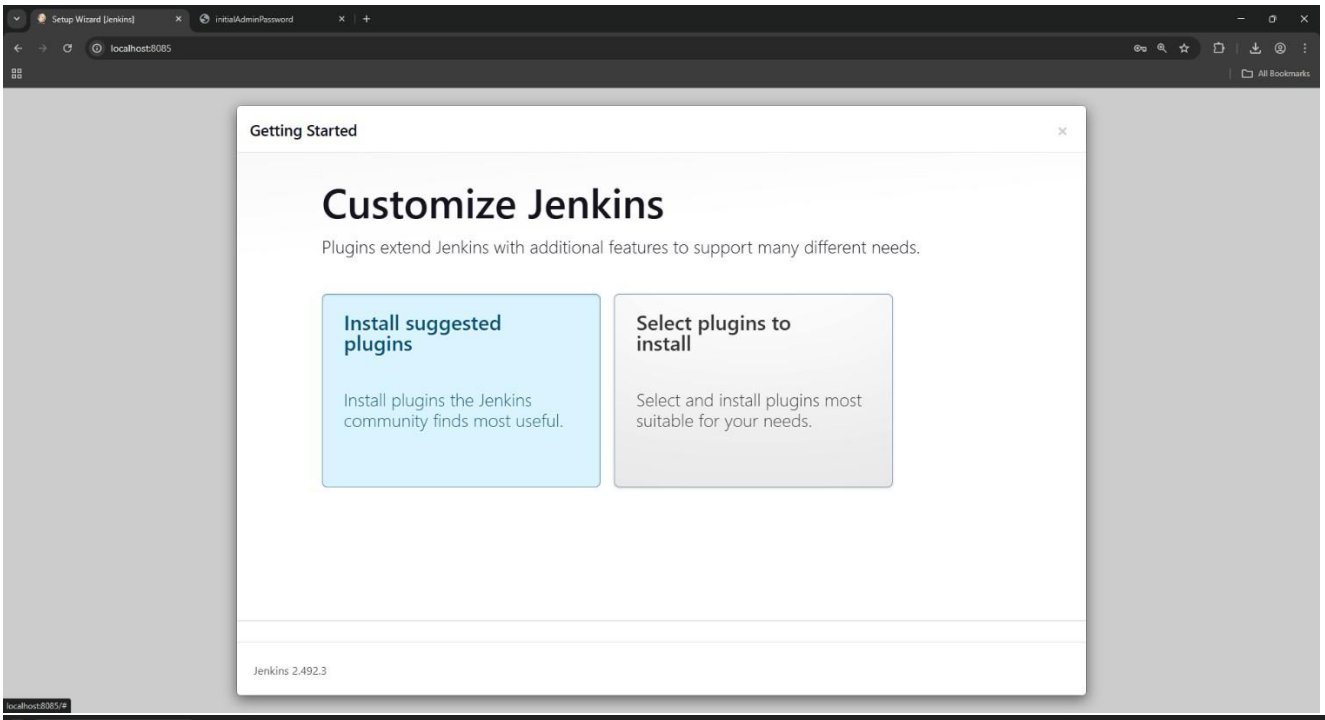
**Aim : demonstrate settings jenkins CI/CD piplline.**

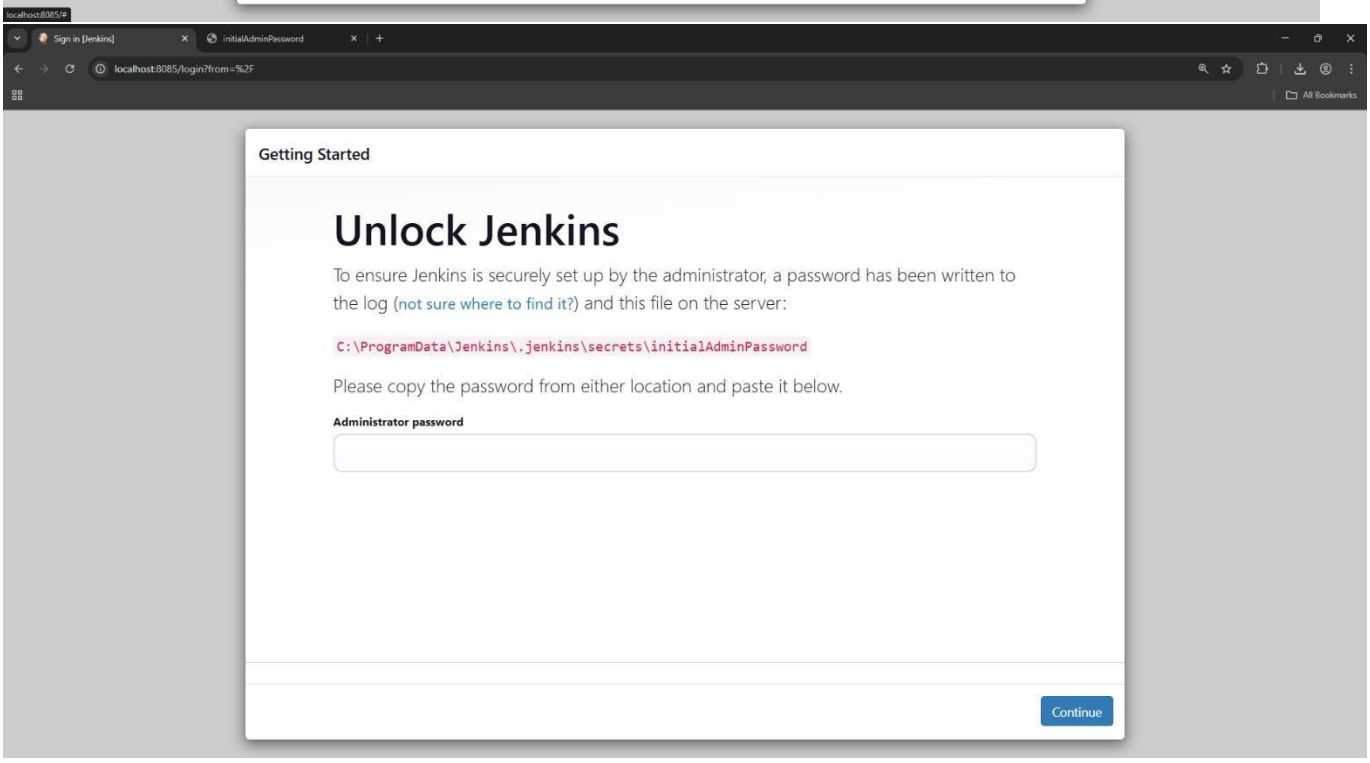



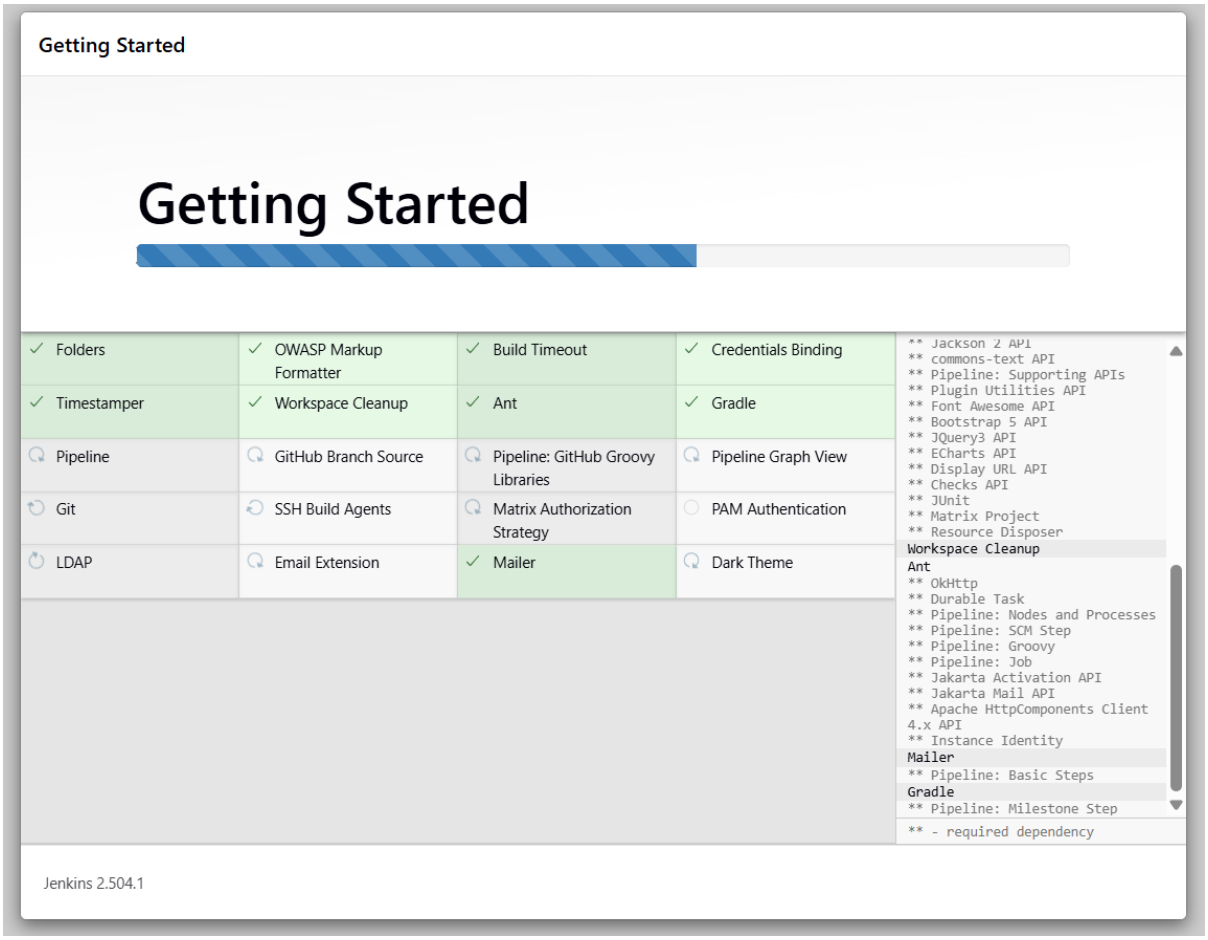

Jenkins 2.492.3 Setup
Port Selection
Choose a port for the service.
Jenkins
Please choose a port.
Port Number (1-65535):
8085
Test Port
Click 'Test Port' button to proceed
It is recommended that you accept the selected default port.
Back
Next
Cancel

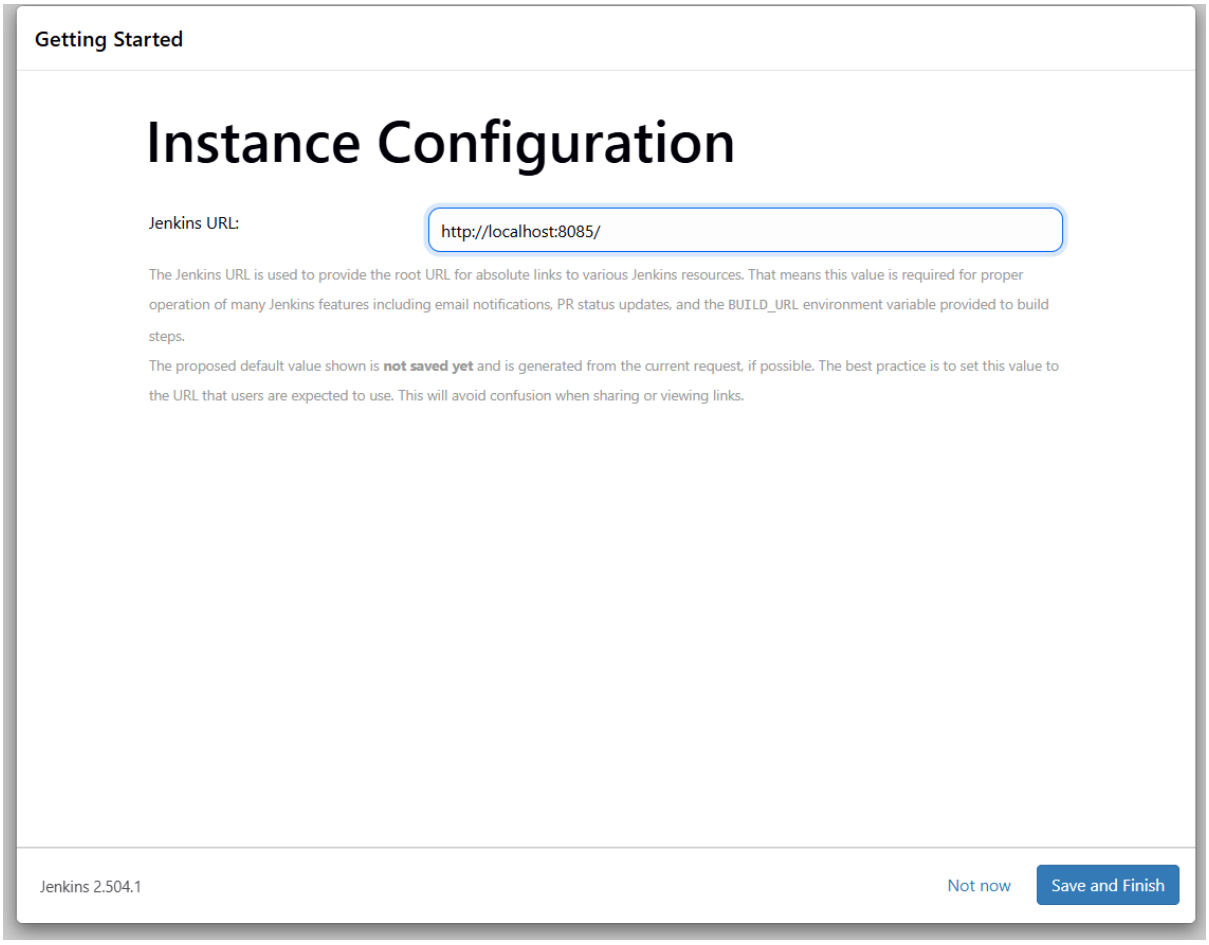

Jenkins 2.492.3 Setup
Service Logon Credentials
Enter service credentials for the service.
Jenkins
Jenkins 2.492.3 installs and runs as an independent Windows service. To operate in this manner, you must supply the user account credentials for Jenkins 2.492.3 to run successfully.
Logon Type:
Run service as LocalSystem (not recommended)
Run service as local or domain user:
Account:
Password:
Test Credentials
Back
Next
Cancel

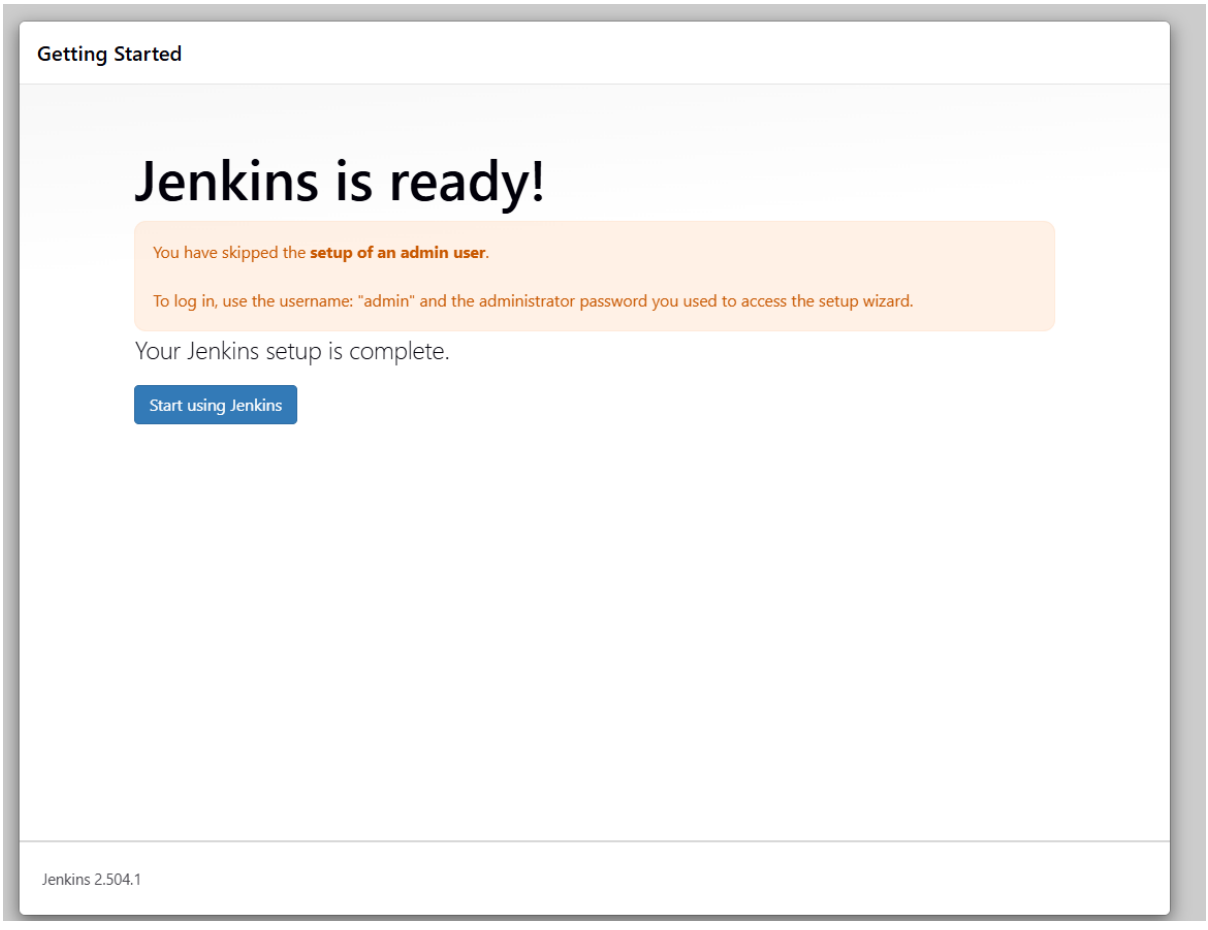

Jenkins 2.492.3 Setup
Welcome to the Jenkins 2.492.3 Setup Wizard
The Setup Wizard will install Jenkins 2.492.3 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel

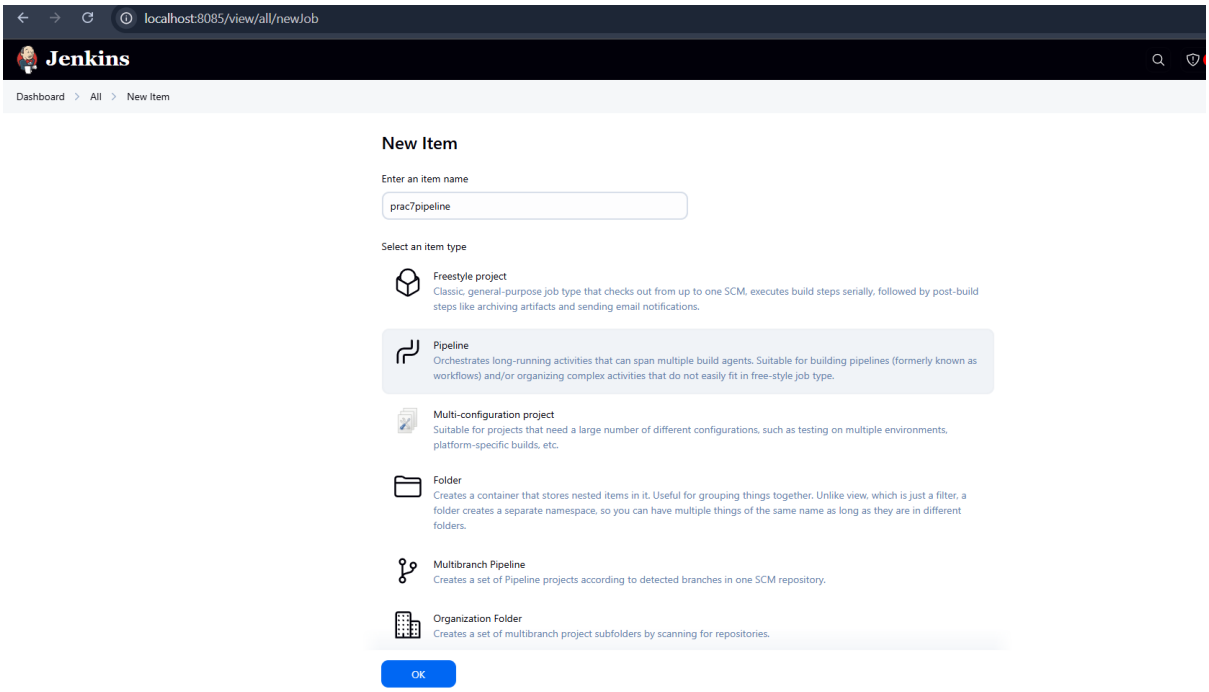

Setup Wizard [Jenkins]
initialAdminPassword
localhost:8085
All Bookmarks
Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.492.3

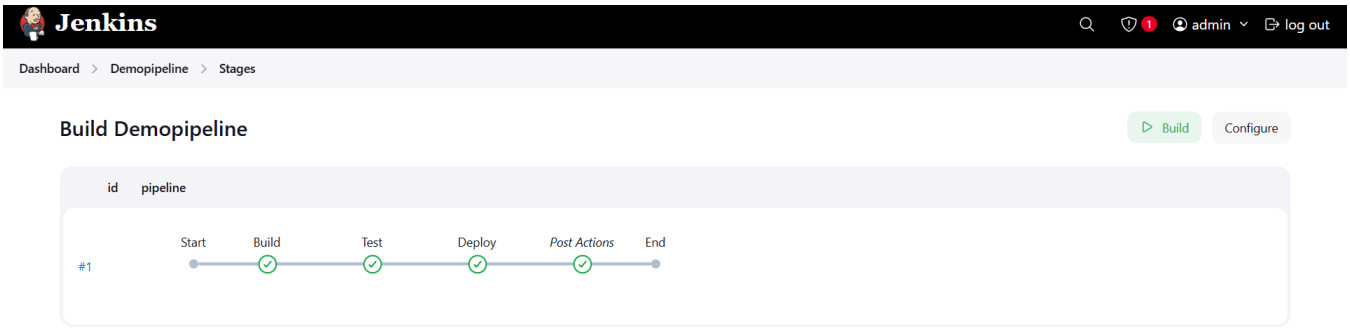

Sign in [Jenkins]
initialAdminPassword
localhost:8085/login?from=%2F
All Bookmarks
Getting Started
Unlock Jenkins
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:
C:\ProgramData\Jenkins\.jenkins\secrets\initialAdminPassword
Please copy the password from either location and paste it below.
Administrator password
Continue


Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline Graph View
Git
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
Email Extension
Mailer
Dark Theme
** Jackson 2 API
** commons-text API
** Pipeline: Supporting APIs
** Plugin Utilities API
** Font Awesome API
** Bootstrap 5 API
** JQuery3 API
** ECharts API
** Display URL API
** Checks API
** JUnit
** Matrix Project
** Resource Disposer
Workspace Cleanup
Ant
** OkHttp
** Durable Task
** Pipeline: Nodes and Processes
** Pipeline: SCM Step
** Pipeline: Groovy
** Pipeline: Job
** Jakarta Activation API
** Jakarta Mail API
** Apache HttpComponents Client 4.x API
** Instance Identity
Mailer
** Pipeline: Basic Steps
Gradle
** Pipeline: Milestone Step
** - required dependency
Jenkins 2.504.1


Getting Started
Instance Configuration
Jenkins URL:
http://localhost:8085/
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.
Jenkins 2.504.1
Not now
Save and Finish


Getting Started
Jenkins is ready!
You have skipped the setup of an admin user.
To log in, use the username: "admin" and the administrator password you used to access the setup wizard.
Your Jenkins setup is complete.
Start using Jenkins
Jenkins 2.504.1




Jenkins 2.492.3

# Practical 8

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

**Index.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Cookies.jsp" method="get"> Name:<input type="text" name="user">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

**Cookies.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String usename=request.getParameter("user"); Cookie[] cookies=request.getCookies();
int visitCount=0; boolean userExist=false; if(cookies!=null){
for(Cookie cookie:cookies){ if(cookie.getName().equals("visitCount")){
visitCount=Integer.parseInt(cookie.getValue());
}
if(cookie.getName().equals("username")){
userExist=true;
}
}
}
```

**visitCount++;**

**Cookie visitcookie=new Cookie("visitCount",String.valueOf(visitCount));**
**visitcookie.setMaxAge(60*60*24);**

**response.addCookie(visitcookie); if(!userExist&&usename!=null){**

**Cookie usercookie=new Cookie("username",usename); usercookie.setMaxAge(60*60*24);**
**response.addCookie(usercookie);**

**}**

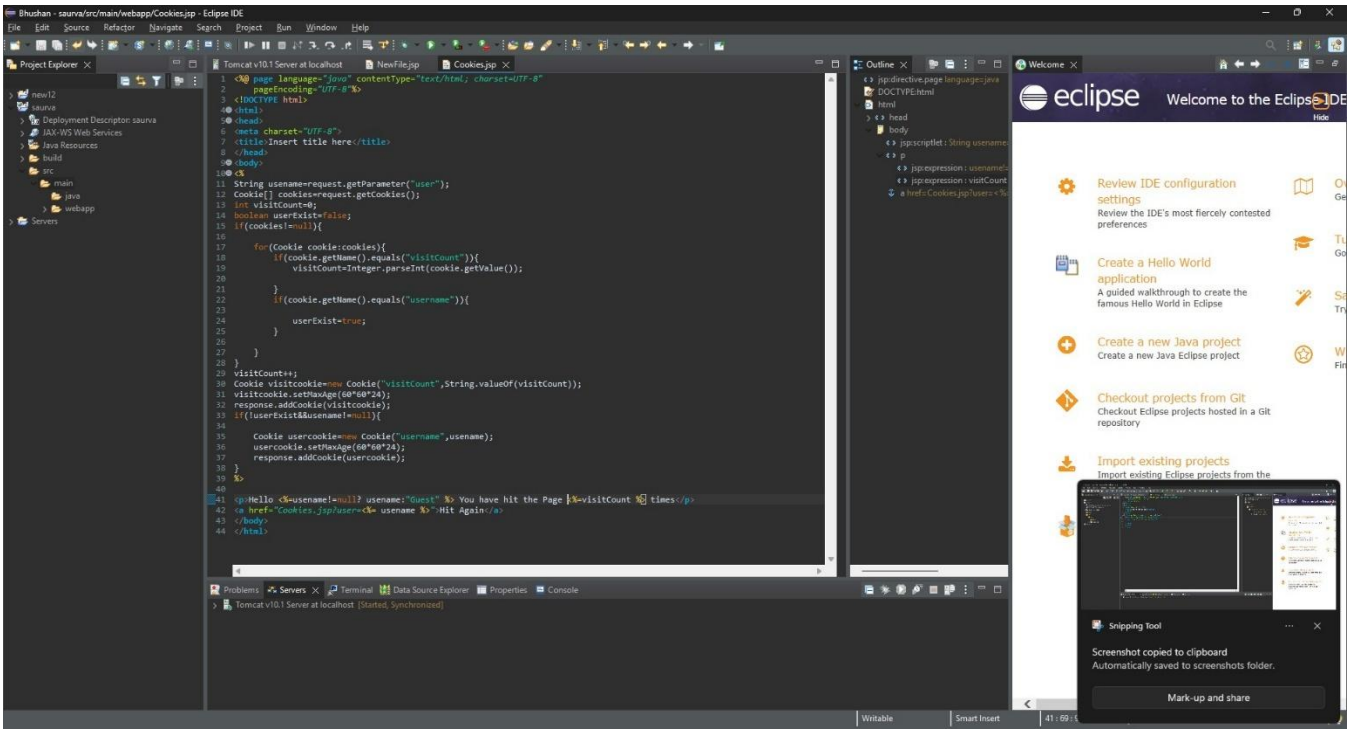
**%>**

** Hello <%=usename!=null? usename:"Guest" %> You have hit the Page <%=visitCount %> times **

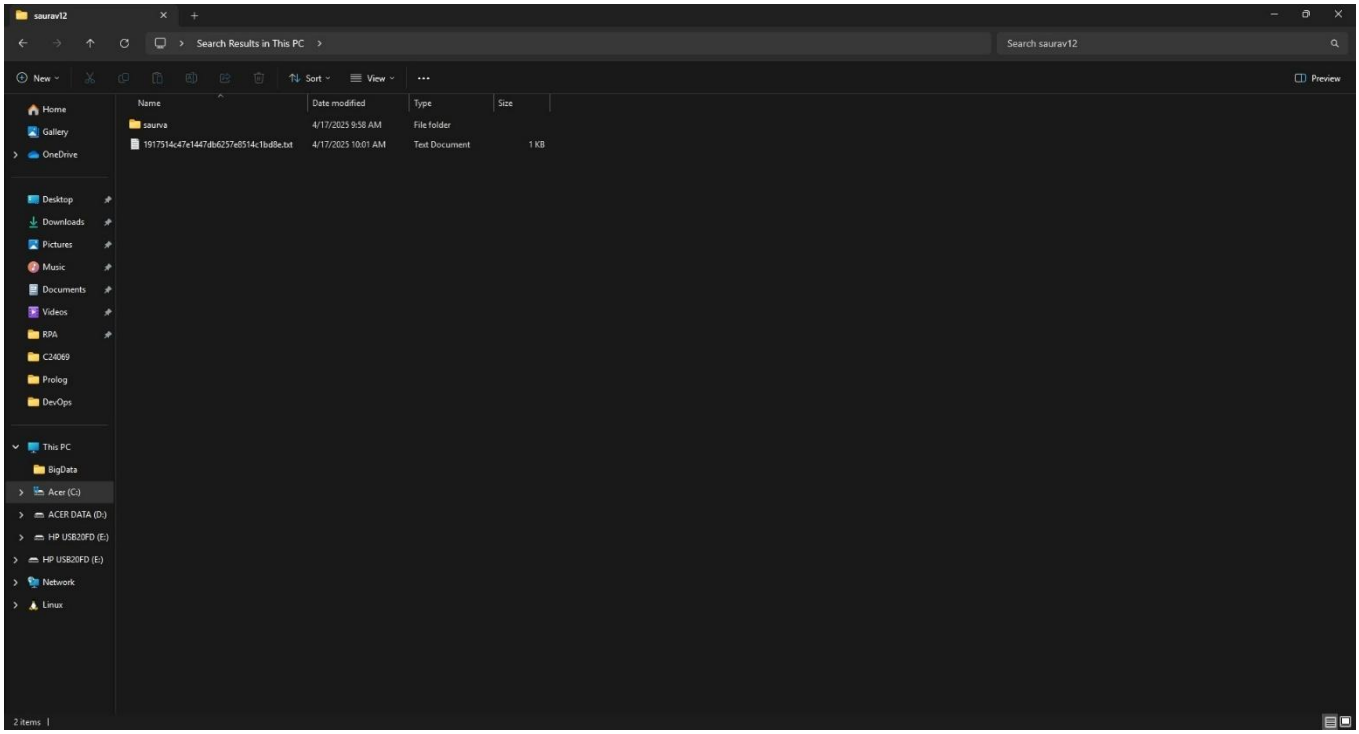
**<a href="Cookies.jsp?user=<%= usename %>">Hit Again</a>**

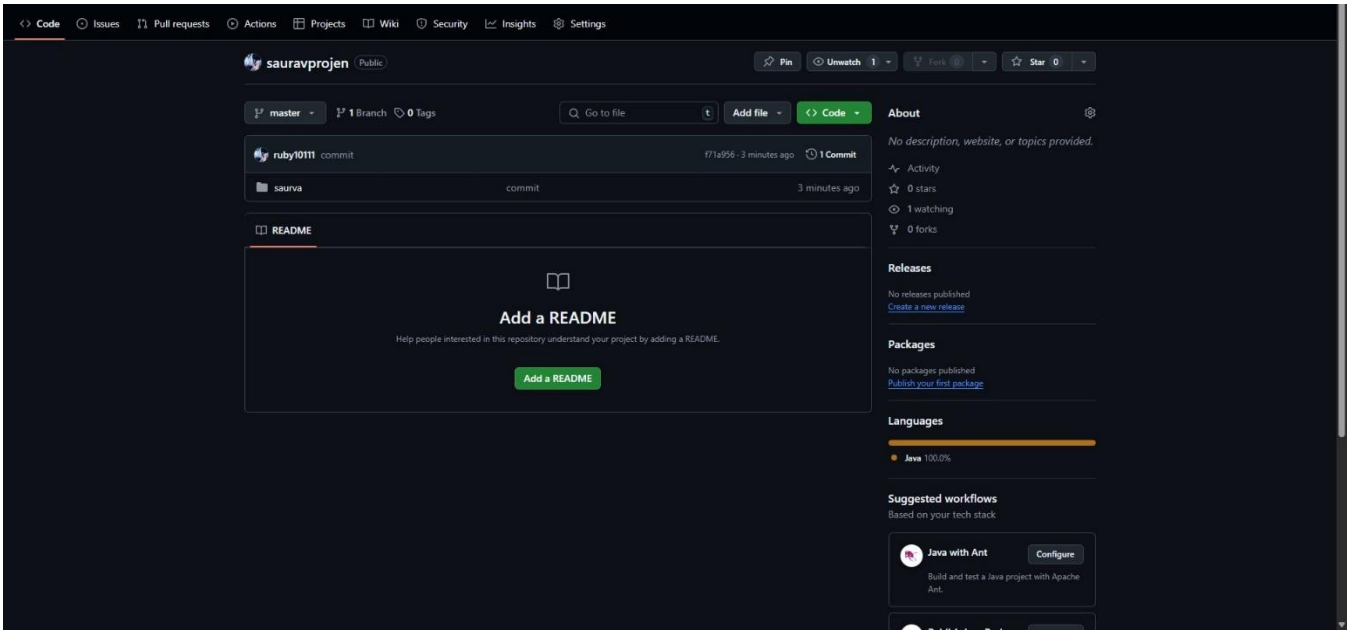
**</body>**

**</html>**

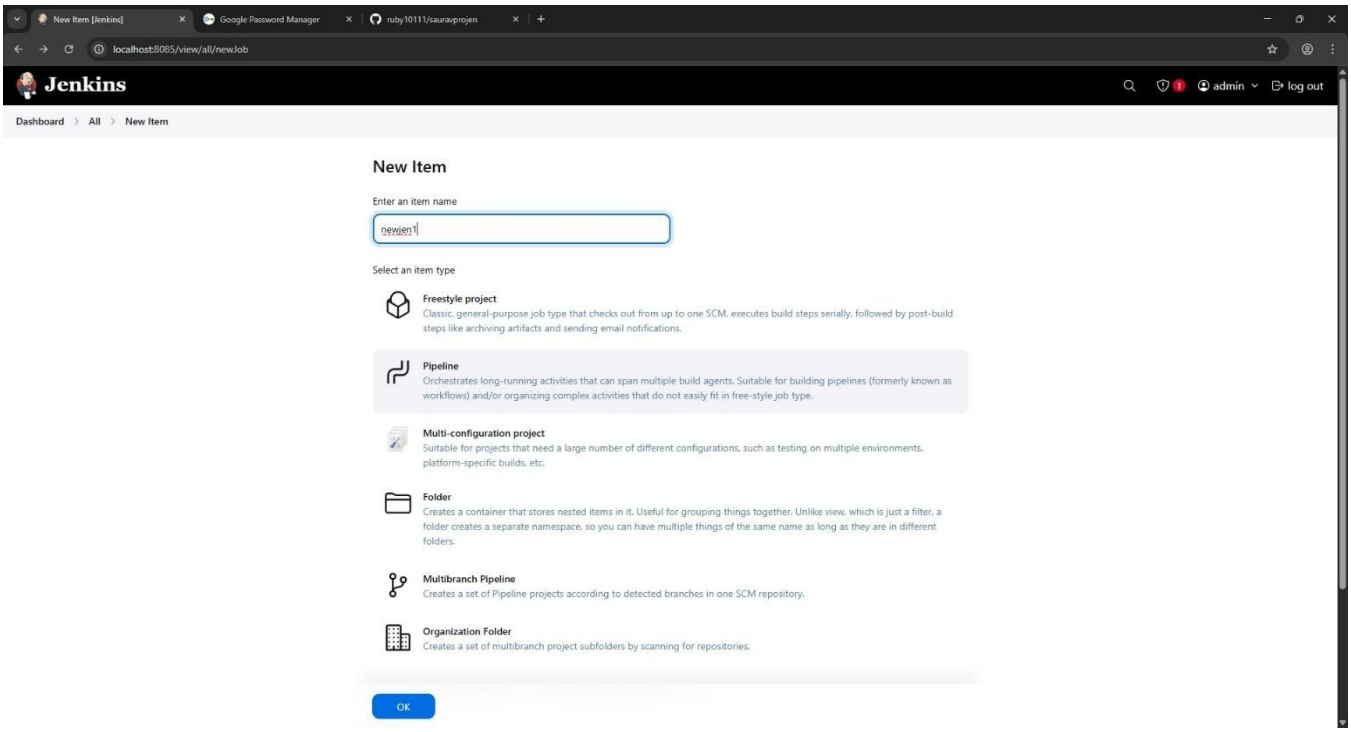


# Push project on github



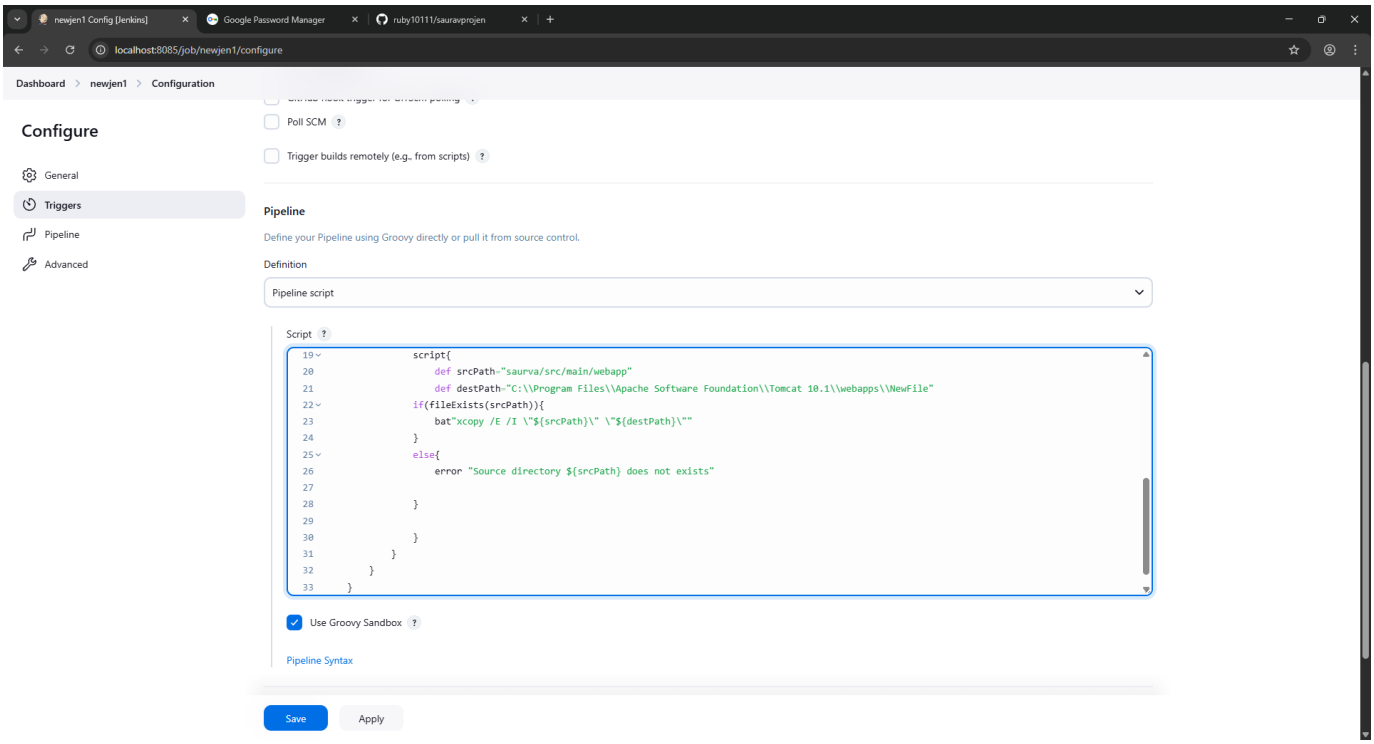



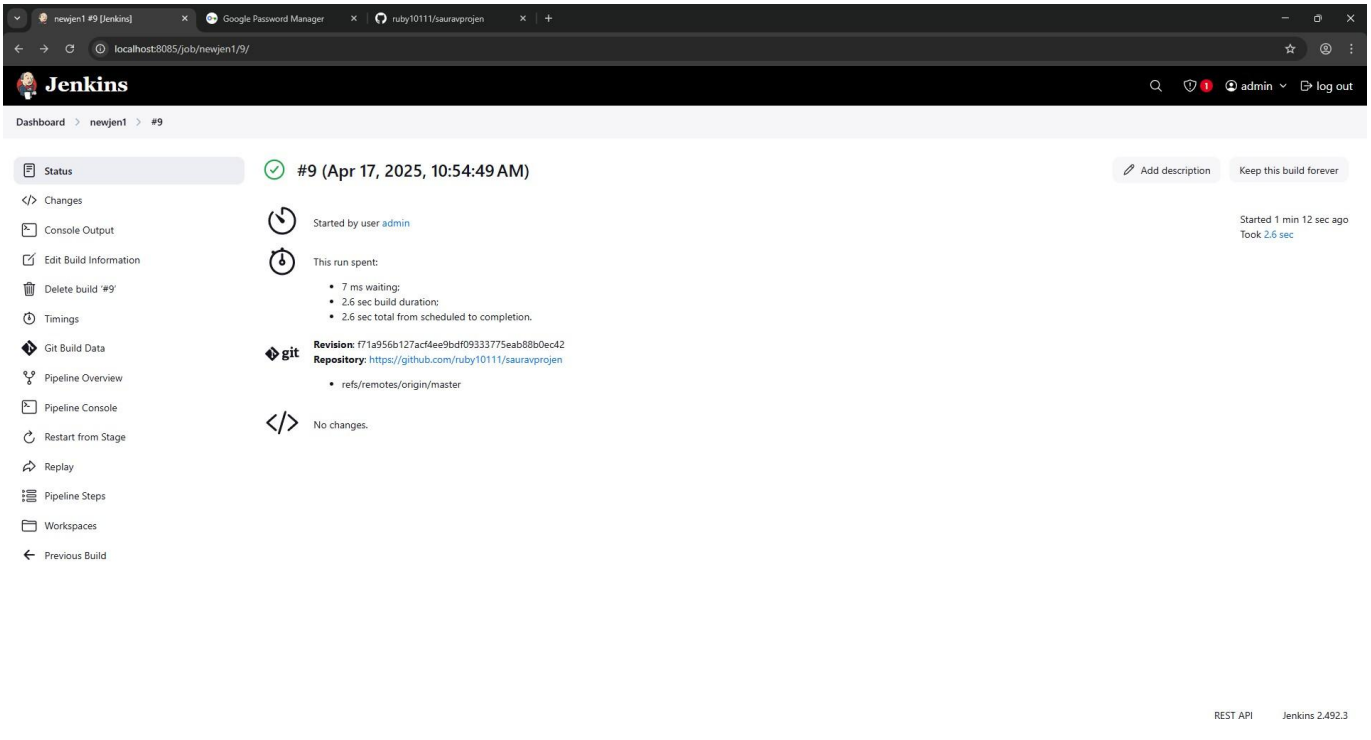
**Create a pipeline:**

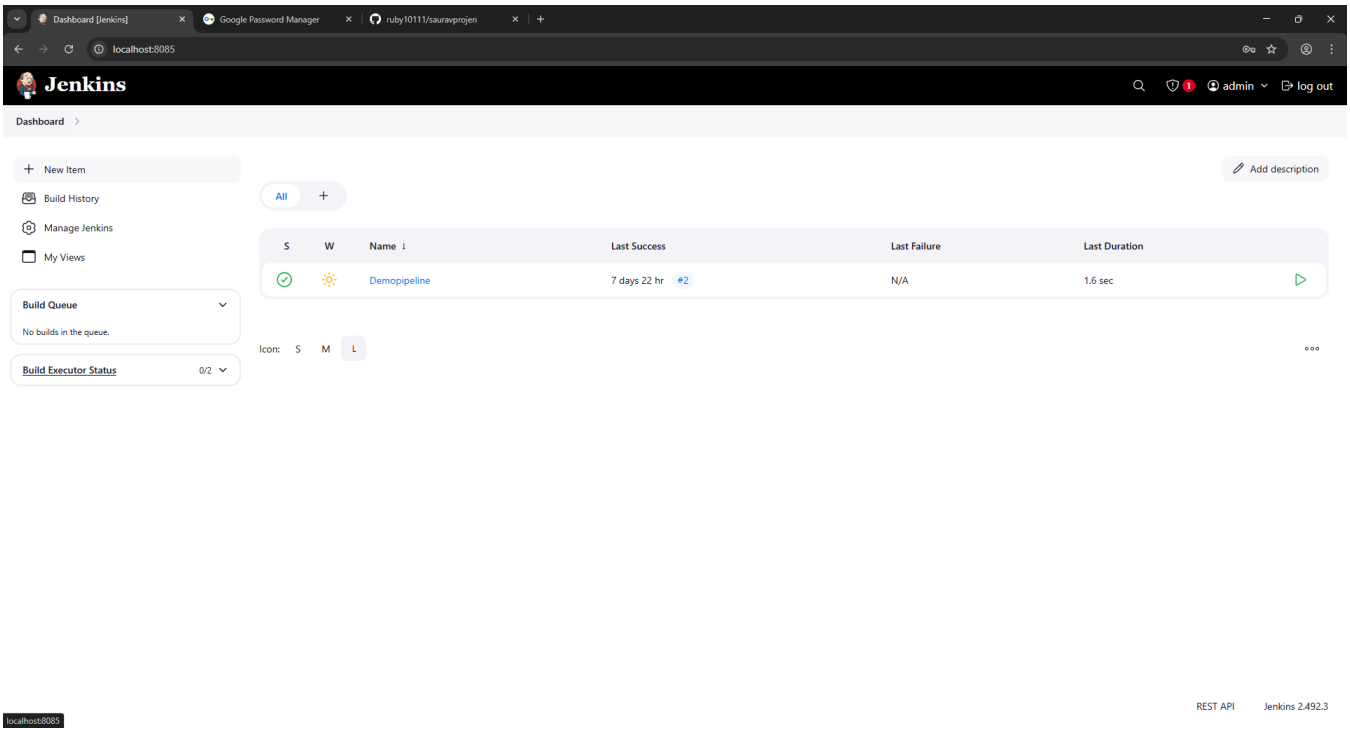



```
pipeline{ agent any
stages{
stage('Checkout Code'){ steps{
script{
git branch: 'master',url:'https://github.com/admin111/devopsprojen'
}
}
}
stage('Verify Files'){ steps{
bat 'dir /S /B'
}
}
stage('Deploy'){ steps{
script{
def srcPath="admin/src/main/webapp"
def destPath="C:\\Program Files\\Apache Software Foundation\\Tomcat 10.1\\webapps\\NewFile"
if(fileExists(srcPath)){
bat"xcopy /E /I \"${srcPath}\" \"${destPath}\""
```

```
}
else{
error "Source directory ${srcPath} does not exists"
}
}
}
```


newjen1 Config [Jenkins]
Google Password Manager
ruby10111/sauravprojen
localhost:8085/job/newjen1/configure
Dashboard > newjen1 > Configuration
Configure
General
Triggers
Pipeline
Advanced
Poll SCM
Trigger builds remotely (e.g., from scripts)
Pipeline
Define your Pipeline using Groovy directly or pull it from source control.
Definition
Pipeline script
Script
19 script{
20 def srcPath="saurva/src/main/webapp"
21 def destPath="C:\\Program Files\\Apache Software Foundation\\Tomcat 10.1\\webapps\\NewFile"
22 if(fileExists(srcPath)){
23 bat"xcopy /E /I \"${srcPath}\" \"${destPath}\""
24 }
25 else{
26 error "Source directory ${srcPath} does not exists"
27
28 }
29
30 }
31 }
32 }
33 }
Use Groovy Sandbox
Pipeline Syntax
Save
Apply


newjen1 #9 [Jenkins]
Google Password Manager
ruby10111/sauravprojen
localhost:8085/job/newjen1/9/
Jenkins
admin
log out
Dashboard > newjen1 > #9
Status
Changes
Console Output
Edit Build Information
Delete build '#9'
Timings
Git Build Data
Pipeline Overview
Pipeline Console
Restart from Stage
Replay
Pipeline Steps
Workspaces
Previous Build
#9 (Apr 17, 2025, 10:54:49 AM)
Add description
Keep this build forever
Started by user admin
Started 1 min 12 sec ago
Took 2.6 sec
This run spent:
7 ms waiting;
2.6 sec build duration;
2.6 sec total from scheduled to completion.
Revision: f71a956b127acf4ee9bdf09333775eab88b0ec42
Repository: https://github.com/ruby10111/sauravprojen
refs/remotes/origin/master
No changes.
REST API
Jenkins 2.492.3


Dashboard [Jenkins]
Google Password Manager
ruby10111/sauravprojen
localhost:8085
Jenkins
admin
log out
Dashboard
New Item
Build History
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
0/2
Add description
All
S
W
Name
Last Success
Last Failure
Last Duration
Demopipeline
7 days 22 hr #2
N/A
1.6 sec
Icon: S M L
REST API
Jenkins 2.492.3
localhost:8085

# Practical 9

**Aim: demonstrate basic Docker commands**

1. **Check Docker version docker –version**

```
ubuntu@ubuntu:~$ docker --version
Docker version 28.1.1, build 4eba377
```

2. **Pull a Docker image from Docker Hub docker pull nginx**

```
ubuntu@ubuntu:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
254e724d7786: Pull complete
913115292750: Pull complete
3e544d53ce49: Pull complete
4f21ed9ac0c0: Pull complete
d38f2ef2d6f2: Pull complete
40a6e9f4e456: Pull complete
d3dc5ec71e9d: Pull complete
Digest: sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

3. **List all Docker images docker images**

```
ubuntu@ubuntu:~$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    a830707172e8   4 weeks ago   192MB
```

4. **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).

```
ubuntu@ubuntu:~$ docker run -d -p 8080:80 --name mynginx nginx
c241fdc47993e83fe932231e1ba068b8953126eb87a89916c50ebabdc088254c
```

5. **List all running containers docker ps**

```
ubuntu@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
c241fdc47993   nginx     "/docker-entrypoint.…"   27 seconds ago   Up 26 seconds   0.0.0.0:8080->80/tcp   mynginx
```

6. **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.

```
ubuntu@ubuntu:~$ docker cp index.html mynginx:/usr/share/nginx/html/
lstat /home/ubuntu/index.html: no such file or directory
```

7. **Copy content from container to host**

   **docker cp mynginx:/usr/share/nginx/html/index.html .**

```
ubuntu@ubuntu:~$ docker cp index.html mynginx:/usr/share/nginx/html/
lstat /home/ubuntu/index.html: no such file or directory
```

8. **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

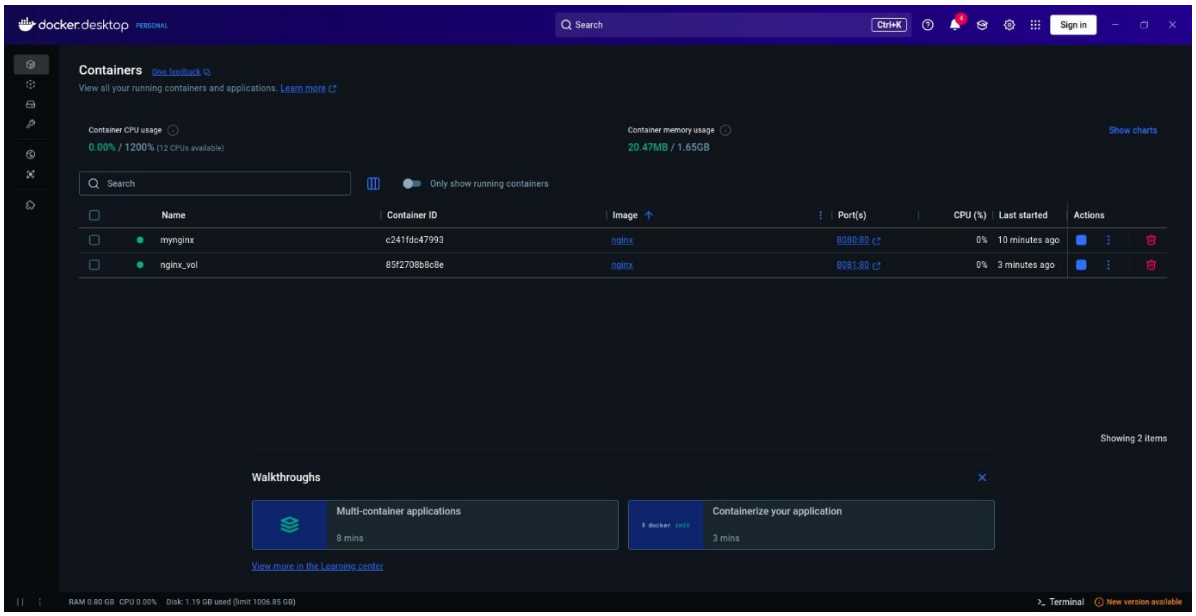
   Now any data added to the /usr/share/nginx/html inside the container will persist even if the container is removed.

```
ubuntu@ubuntu:~$ docker volume create mydata
mydata
ubuntu@ubuntu:~$ docker run -d -p 8081:80 --name nginx_vol -v mydata:/usr/share/nginx/html nginx
85f2708b8c8ec2c1eba2bb88f10a162feec1faa1ad3f86c2f0e8d0ba32e1090a
```

9. **List Docker volumes docker volume ls**

```
ubuntu@ubuntu:~$ docker volume ls
DRIVER    VOLUME NAME
local     mydata
```

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



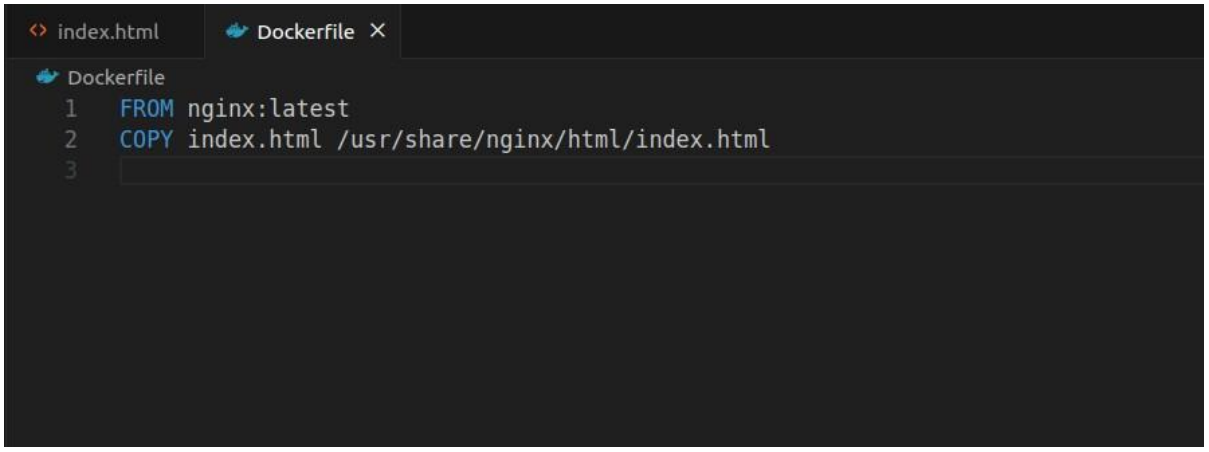
# PRACTICAL-10

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

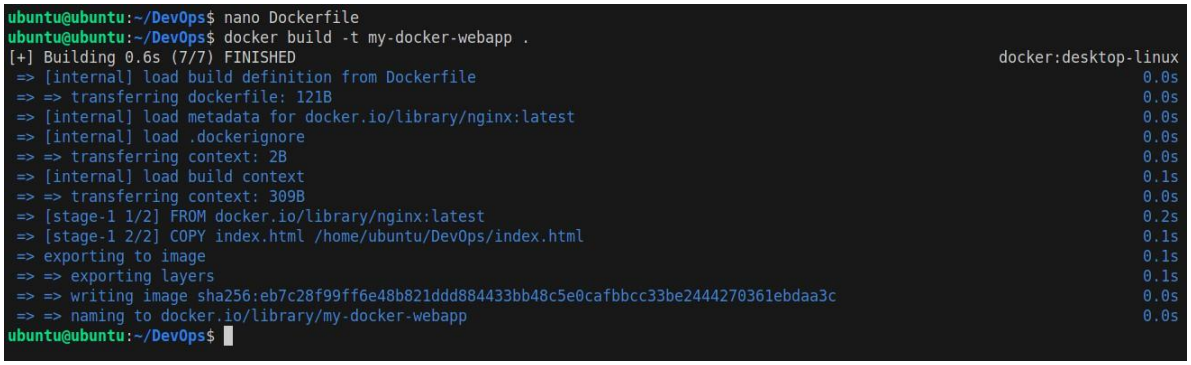
**Develop a Simple Containerized Application using Docker**

1. **Index.html**


2. **Dockerfll**




3. **docker build -t my-docker-webapp .**



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

```
ubuntu@ubuntu:~/DevOps$ docker run -d -p 8080:80 --name webapp-container my-docker-webapp
87758d2c13e4eb227c0bb149148952a661a46b92867ef336a4dd2ad74a993e3f
ubuntu@ubuntu:~/DevOps$
```

5. **docker ps**

```
ubuntu@ubuntu:~/DevOps$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS                  NAMES
87758d2c13e4   my-docker-webapp   "/docker-entrypoint.…"   38 seconds ago   Up 37 seconds   0.0.0.0:8080->80/tcp   webapp-container
85f2708b8c8e   nginx              "/docker-entrypoint.…"   18 minutes ago   Up 18 minutes   0.0.0.0:8081->80/tcp   nginx_vol
```

6. **docker stop webapp-container**

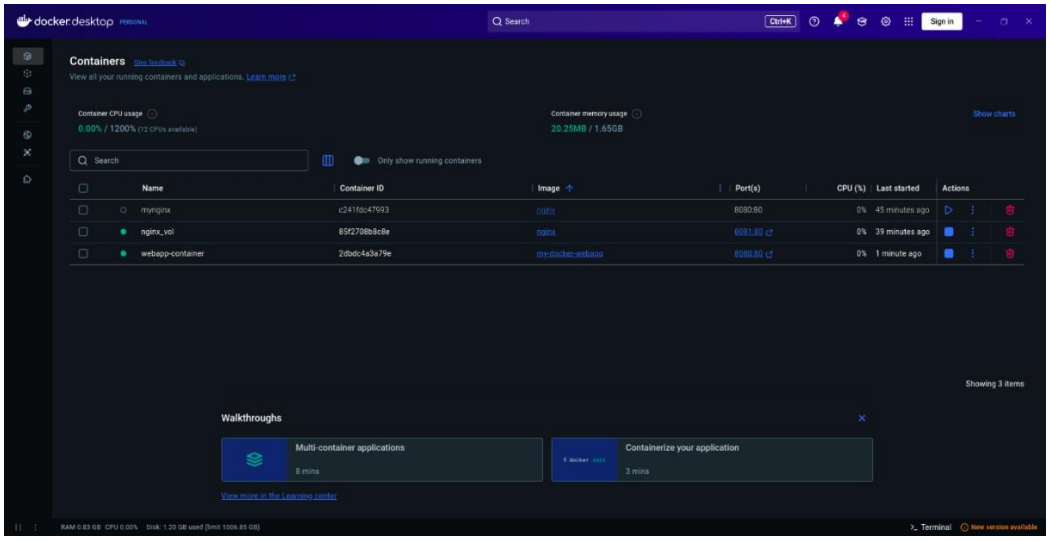
```
ubuntu@ubuntu:~/DevOps$ docker rm webapp-container
webapp-container
```

7. **docker rm webapp-container**

```
ubuntu@ubuntu:~/DevOps$ docker rm webapp-container
webapp-container
```

8. **docker rmi my-docker-webapp**

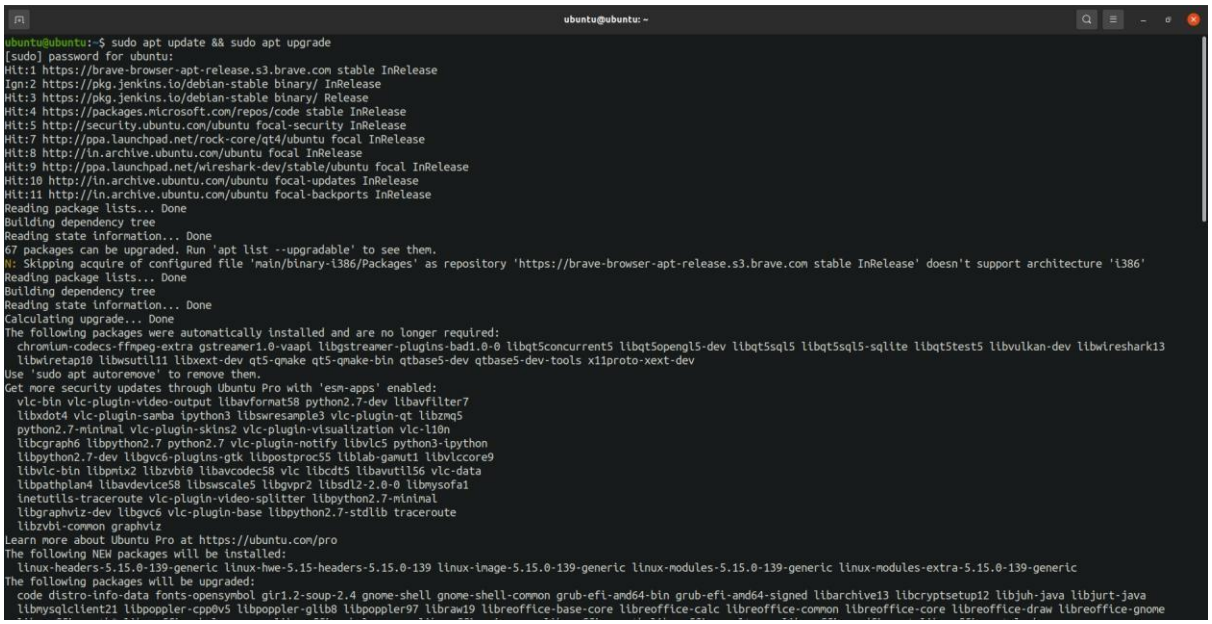
```
ubuntu@ubuntu:~/DevOps$ docker rmi my-docker-webapp
Untagged: my-docker-webapp:latest
Deleted: sha256:eb7c28f99ff6e48b821ddd884433bb48c5e0cafbbcc33be2444270361ebdaa3c
```

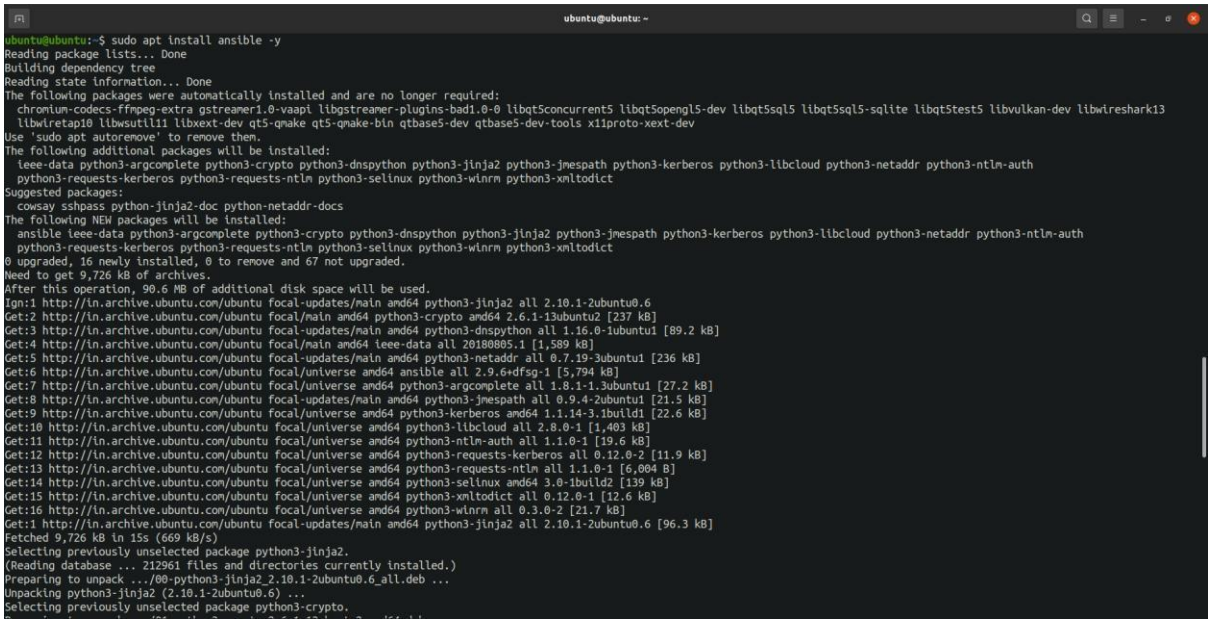
# Practical 11

## Aim: Demonstrate add-on ansible commands

## Step 1: Update your VM




## Step 2: Install Ansible



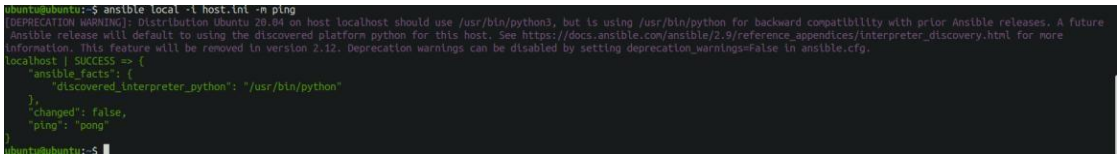

## Step 3: Check version:

```
ubuntu@ubuntu:~$ nano host.ini
ubuntu@ubuntu:~$
```

```
  GNU nano 4.8
localhost ansible_connection=local
```

1. **Ping the remote host**

**ansible local -i host.ini -m ping**

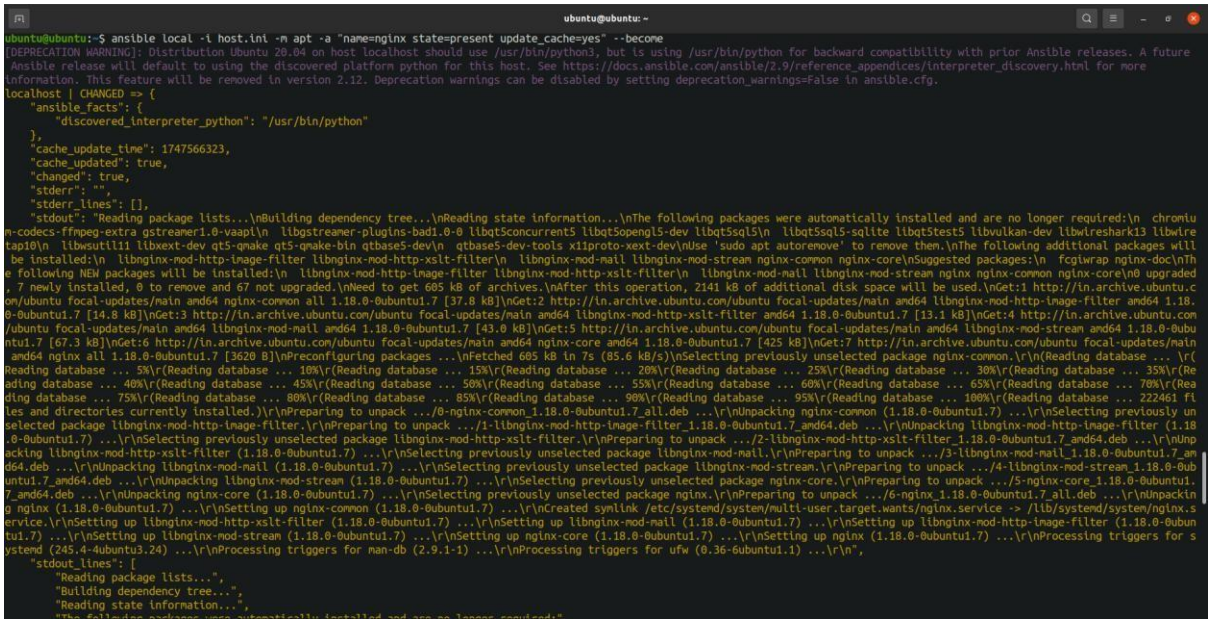


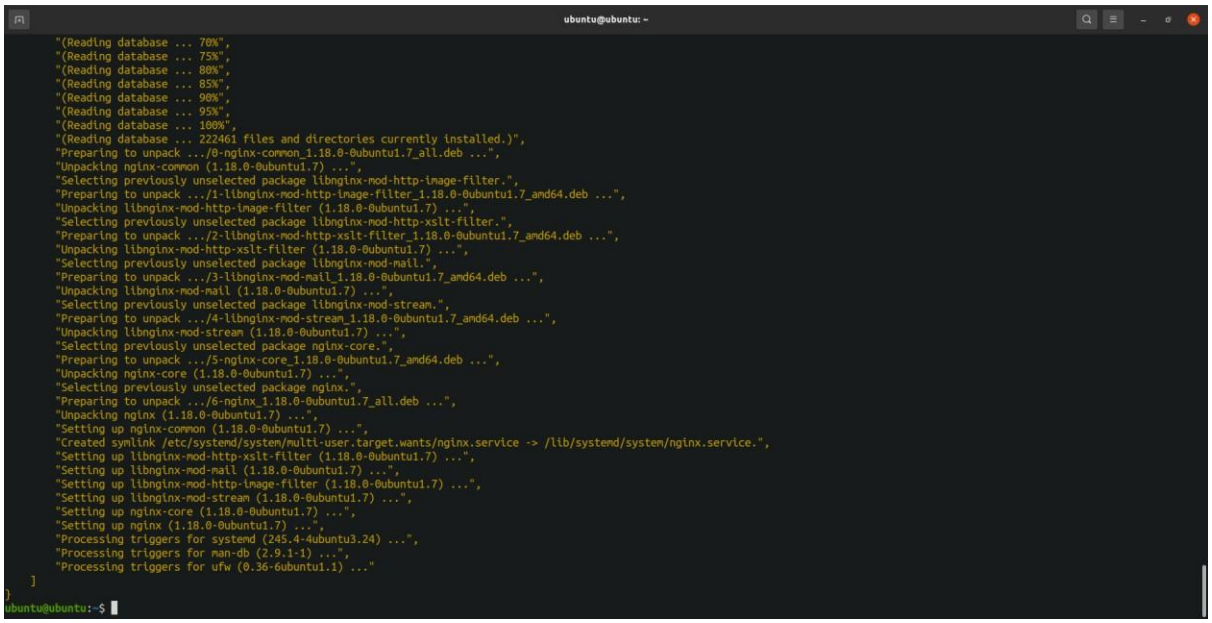

2. **Check uptime**

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


3. **Install a package**

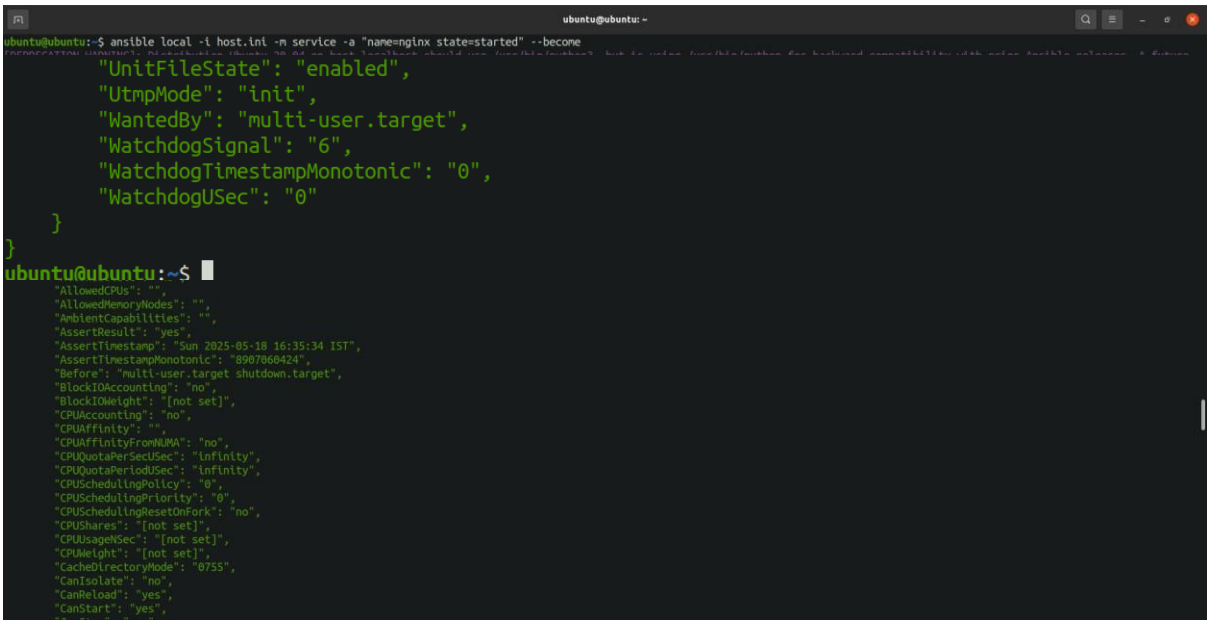
**ansible local -i host.ini -m apt -a "name=nginx state=present update_cache=yes" –become**






4. **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:**

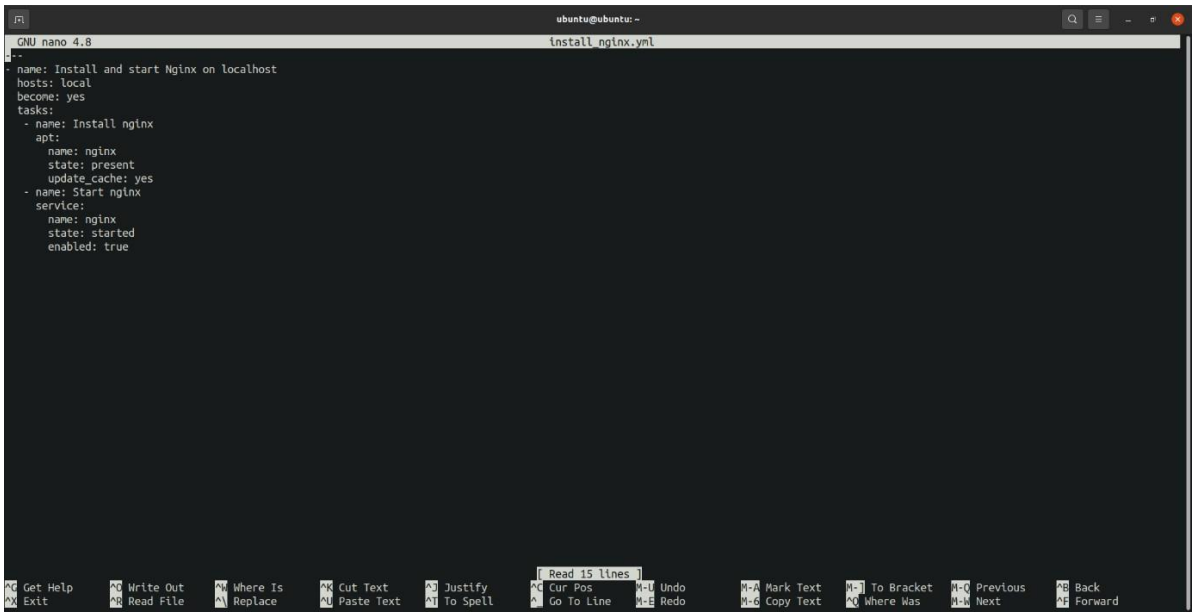
**name: nginx state: present**

**update_cache: yes**

**- name: Start Nginx service:**

**name: nginx state: started enabled: true**

```
ubuntu@ubuntu:~$ nano install_nginx.yml
```



## Run the Playbook:

## ansible-playbook -i hosts.ini install_nginx.yml

```
ubuntu@ubuntu:~$ ansible-playbook -i host.ini install_nginx.yml

PLAY [Install and start Nginx on localhost] ************************************************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************************************************
[DEPRECATION WARNING]: Distribution Ubuntu 20.04 on host localhost should use /usr/bin/python3, but is using /usr/bin/python for backward compatibility with prior Ansible releases. A future
 Ansible release will default to using the discovered platform python for this host. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information. This feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [localhost]

TASK [Install nginx] ***********************************************************************************************************************************************************************
ok: [localhost]

TASK [Start nginx] *************************************************************************************************************************************************************************
ok: [localhost]

PLAY RECAP *********************************************************************************************************************************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```