Experiment no: 1

Exploring Git Commands through Collaborative Coding.

Experiment Steps:

Step 1: Setting Up Git Repository

- Create a new folder
- Open the vs code teminal on your computer.
- Navigate to the directory where you want to create your Git repository.
- Run the following commands:

Command 1: git init

This initialises a new Git repository in the current directory

Step 2: Creating and Committing Changes

- Create a new text file named "example.txt" using any text editor.
- Add some content to the "example.txt" file.
- In the command-line interface, run the following commands:

Command 2: git status

This command shows the status of your working directory, highlighting untracked files

Command 3: git add example.txt

This stages the changes of the "example.txt" file for commit.

Command 4: git reset example.txt

Reverts changes in the working directory and/or staging area to a previous commit.

git status

git add . (or) git add example.txt

Command 5: git commit -m "Add content to example.txt"

This commits the staged changes with a descriptive message

Step 3: Exploring History

Modify the content of "example.txt." Run the following commands:

git status
Notice the modified file is shown as "modified”

Command 6: git diff
This displays the differences between the working directory and the last commit.

Command 7: git log
This displays a chronological history of commits.

Step 4: Branching and Merging

Create a new branch named "feature" and switch to it:

Command 8: git branch feature

Command 9: git checkout feature or shorthand: **git checkout -b feature**

Make changes to the "example.txt" file in the "feature" branch.

Commit the changes in the "feature" branch.

Switch back to the "master" branch:

Command 10: git checkout master

Merge the changes from the "feature" branch into the "master" branch:

Command 11: git merge feature

Step 5: Collaborating with Remote Repositories

- Create an account on a Git hosting service like GitHub (https://github.com/).
- Create a new repository on GitHub.

Link your local repository to the remote repository:

Command 11: git remote add origin <repository_url>

Push your local commits to the remote repository:

Command 12: git push origin master

Experiment No. 2

Implement GitHub Operations using Git

Step 1: Cloning a Repository

- Sign in to your GitHub account.
- Find a repository to clone (you can use a repository of your own or any public repository).
- Click the "Code" button and copy the repository URL.
- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Run the following command:

Command 1: git clone <repository_url>

Replace <repository_url> with the URL you copied from

GitHub. This will clone the repository to your local machine.

Step 2: Making Changes and Creating a Branch

- Navigate into the cloned repository:

cd <repository_name>

- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

Command 2: git status

- Stage the changes for commit:

Command 3: git add example.txt

- Commit the changes with a descriptive message:

Command 4: git commit -m "Add content to example.txt"

- Create a new branch named "feature":

Command 5: git branch feature

- Switch to the "feature" branch:

Command 6: git checkout feature

Step 3: Pushing Changes to GitHub

- Add Repository URL in a variable

Command 7: git remote add origin <repository_url>

- Replace <repository_url> with the URL you copied from GitHub.
- Verifies remote repository URLs associated with your local repository.

Command 8: git remote –v

- Push the "feature" branch to GitHub:

Command 9: git push origin feature

Check your GitHub repository to confirm that the new branch "feature" is available.

Step 4: Collaborating through Pull Requests

- Create a pull request on GitHub:
- Go to the repository on GitHub.
- Click on "Pull Requests" and then "New Pull Request."
- Choose the base branch (usually "main" or "master") and the compare branch ("feature").
- Review the changes and click "Create Pull Request."
- Review and merge the pull request:
- Add a title and description for the pull request.
- Assign reviewers if needed.
- Once the pull request is approved, merge it into the base branch.

Step 5: Syncing Changes

Command 10: git checkout main

Command 11: git pull origin main

Step 6: Fetching changes

Command 12: git branch –r

- The command git branch -r is used to list all remote branches in a Git repository
- You can see the list of branches in git hub
- Open github
- Create new branch in github
- Branch name :feature2

Command 13: git fetch –all

- It downloads all the latest commits, branches, and tags from all remotes configured in your repository.
- After fetching again see the list of branches in git hub
- **git branch –r**

Step 7: Deleting branches

Command 14 : git push -d origin feature2

- Deletes the branch feature 2 in github (remote)

Command 15 : git branch -d feature2

Deletes a specified branch locally

git branch –r

- The command git branch -r is used to list all remote branches in a Git repository
- You can see the list of branches in git hub

Experiment No. 3

Implement GitLab Operations using Git.

Experiment Steps:

Step 1: Creating a Repository

- Sign in to your GitLab account.
- Click the "New" button to create a new project.
- Choose a project name, visibility level (public, private), and other settings.
- Click "Create project."

Step 2: Cloning a Repository

- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Copy the repository URL from GitLab.
- Run the following command:

Command 1: git clone <repository_url>

- Replace <repository_url> with the URL you copied from GitLab.
- This will clone the repository to your local machine.

Step 3: Making Changes and Creating a Branch

- Navigate into the cloned repository:
- **cd <repository_name>**
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

Command 2: git status

- Stage the changes for commit:
- **Command 3: git add example.txt**
- Commit the changes with a descriptive message:

Command 4: git commit -m "Add content to example.txt"

- Create a new branch named "feature":

  Command 5: git branch feature
- Switch to the "feature" branch:

  Command 6: git checkout feature

Step 4: Pushing Changes to GitLab

- Add Repository URL in a variable

  Command 7: git remote add origin <repository_url>
- Replace <repository_url> with the URL you copied from GitLab.
- Push the "feature" branch to GitLab:

  Command 8: git push origin feature
- Check your GitLab repository to confirm that the new branch "feature" is available.

Step 5: Collaborating through Merge Requests

1. Create a merge request on GitLab:
   - Go to the repository on GitLab.
   - Click on "Merge Requests" and then "New Merge Request."
   - Choose the source branch ("feature") and the target branch ("main" or "master").
   - Review the changes and click "Submit merge request."
2. Review and merge the merge request:
   - Add a title and description for the merge request.
   - Assign reviewers if needed.
   - Once the merge request is approved, merge it into the target branch.

Step 6: Syncing Changes

After the merge request is merged, update your local repository:

Command 9: git checkout main

Command 10: git pull origin main

Experiment No. 4

Title: Implement BitBucket Operations using Git

Step 1: Creating a Repository

- Sign in to your Bitbucket account.
- Click the "Create" button to create a new repository.
- Choose a repository name, visibility (public or private), and other settings.
- Click "Create repository."

Step 2: Cloning a Repository

- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Copy the repository URL from BitBucket.
- Run the following command:
- git clone <repository_url>
- Replace <repository_url> with the URL you copied from Bitbucket.
- This will clone the repository to your local machine.

Step 3: Making Changes and Creating a Branch

- Navigate into the cloned repository:
- cd <repository_name>
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

  git status
- Stage the changes for commit:
- git add example.txt
- Commit the changes with a descriptive message:
- git commit-m "Add content to example.txt"
- Create a new branch named "feature":
- git branch feature
- Switch to the "feature" branch:
- git checkout feature

Step 4: Pushing Changes to Bitbucket

- Add Repository URL in a variable

  git remote add origin <repository_url>
- Replace <repository_url> with the URL you copied from Bitbucket.
- Push the "feature" branch to Bitbucket:

  git push origin feature
- Check your Bitbucket repository to confirm that the new branch "feature" is available.

Step 5: Collaborating through Pull Requests

1. Create a pull request on Bitbucket:
    - Go to the repository on Bitbucket.
    - Click on "Create pull request."
    - Choose the source branch ("feature") and the target branch ("main" or
    - "master").
    - Review the changes and click "Create pull request."

2. Review and merge the pull request:

- Add a title and description for the pull request.
- Assign reviewers if needed.
- Once the pull request is approved, merge it into the target branch.

Step 6: Syncing Changes

- After the pull request is merged, update your local repository:

  git checkout main

  git pull origin

  main

Experiment No: 5

Title: Implement BitBucket Operations using Git

Step 1: Setting Up Bitbucket Repository

- Create a new folder
- Open the gitbash on your computer.
- Navigate to the directory where you want to create your Git repository.
- Run the following commands:

git init

This initialises a new Git repository in the current directory

Step 2: Creating and Committing Changes

- Create a new text file named "example.txt" using any text editor.
- Add some content to the "example.txt" file.
- In the command-line interface, run the following commands:

git status

This command shows the status of your working directory, highlighting untracked files

git add example.txt

This stages the changes of the "example.txt" file for commit.

git commit -m "Add content to example.txt"

This commits the staged changes with a descriptive message

Step 3: Exploring History

Modify the content of

"example.txt." Run the following

commands:

git status

Notice the modified file is shown as "modified

git diff

This displays the differences between the working directory and the last commit.

git log

This displays a chronological history of commits.

Step 4: Branching and Merging

Create a new branch named "feature" and switch to it:

git branch feature

git checkout

feature

or shorthand: **git checkout -b feature**

Make changes to the "example.txt" file in the "feature" branch.

Commit the changes in the "feature" branch.

Switch back to the "master" branch:

git checkout master

Merge the changes from the "feature" branch into the "master" branch: git merge feature

Step 5: Collaborating with Remote Repositories

- Create an account on a Git hosting service like GitHub (https://github.com/).
- Create a new repository on GitHub.
- Link your local repository to the remote repository:

git remote add origin <repository_url>

Push your local commits to the remote repository:

git push origin master

Step 6: Merging Conflicts

- This usually happens when changes are made to the same lines of a file or when one person deletes a file that another person edits.
- List our current branches

  git branch –vv
- Create new branch and switch to new branch

  git checkout –b branch1
- Switch to master branch

  git checkout master
- Create new branch and switch to new branch

  git checkout –b branch2
- see the list of branches

  git branch -vv

- switch to branch1

  git checkout branch1
- make changes in branch1
- add and commit changes made in branch1

  git add .

  git commit –m "commit changes"
- push branch 1 to github

  git push origin branch1
- switch to branch2

  git checkout branch2
- Make the changes in branch 2 similar to those you have done in branch 1
- Add and commit changes made in branch 2

  git add .

  git commit -m "save changes made in branch 2"
- push changes to git hub

  git push origin branch2
- switch to master branch

  git checkout master
- merge master with branch1

  git merge branch1

  git status

  git push
- merge master with branch2

  git merge branch2
- now merge conflict arrises
- now make the correct changes in code
- add and the changes and commit

  git add .

  git commit –m "add

  changes" git push

Experiment No: 7

Building a Java Application from GitHub using Jenkins

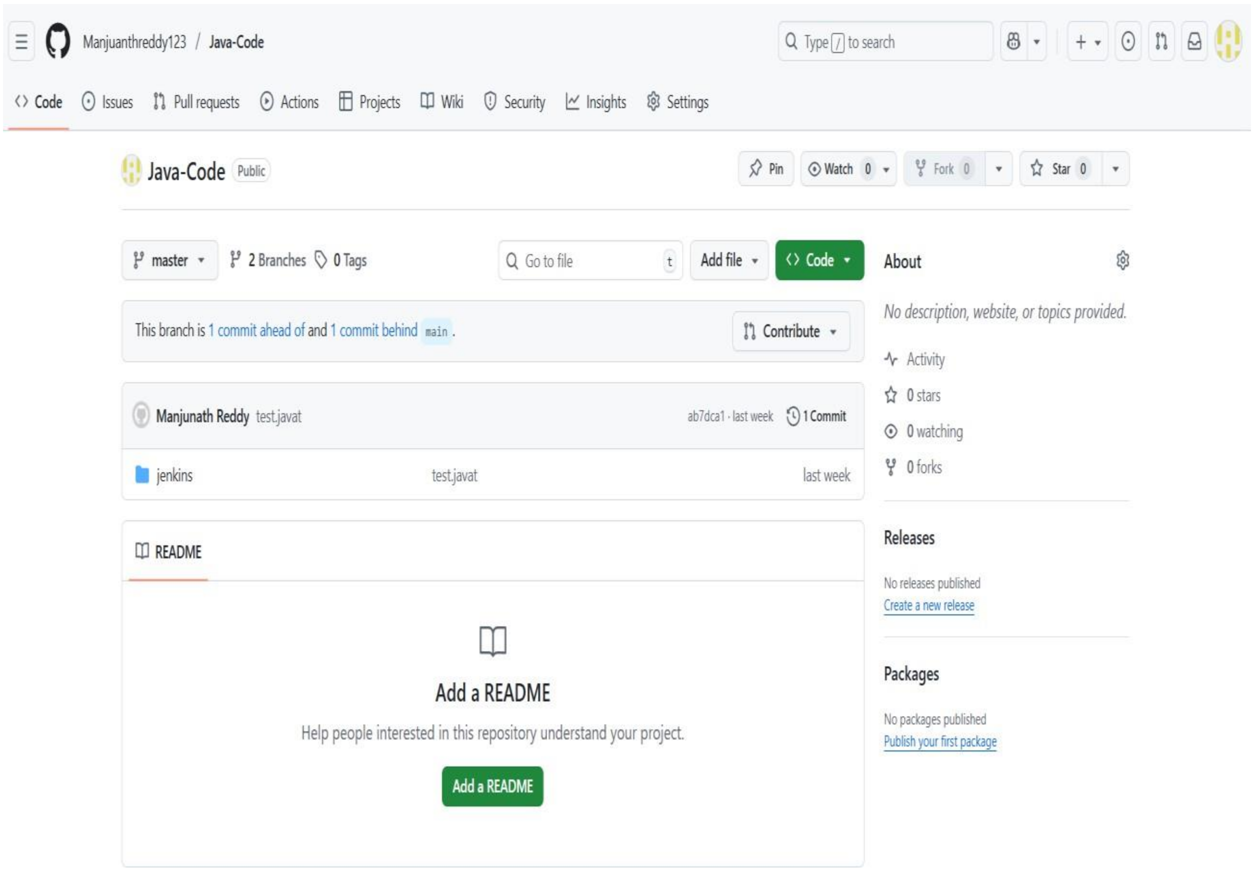
Building a Java application using Jenkins and GitHub involves setting up a continuous integration (CI) pipeline that fetches the source code from a GitHub repository, builds the application, runs tests, and possibly deploys it. Here are the general steps to achieve this:

Step1: Build a java application using any IDE and push the code git

repository Create New folder open the new folder in VS code

Create a java file in new folder by name test.java

Write the code and push the code to git hub



Manjuanthreddy123 / Java-Code

Java-Code Public

master · 2 Branches · 0 Tags

This branch is 1 commit ahead of and 1 commit behind main.

Manjunath Reddy test.javat · ab7dca1 · last week · 1 Commit

jenkins	test.javat	last week

README

Add a README

Help people interested in this repository understand your project.

About

No description, website, or topics provided.

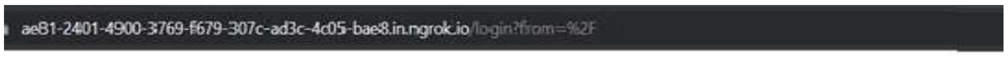
Activity · 0 stars · 0 watching · 0 forks

Releases: No releases published · Create a new release

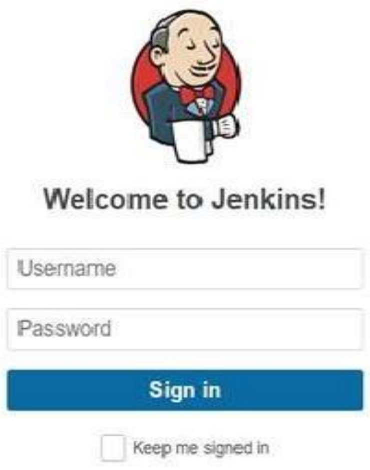
Packages: No packages published · Publish your first package

Step2: open Jenkins (loacalhost:<port

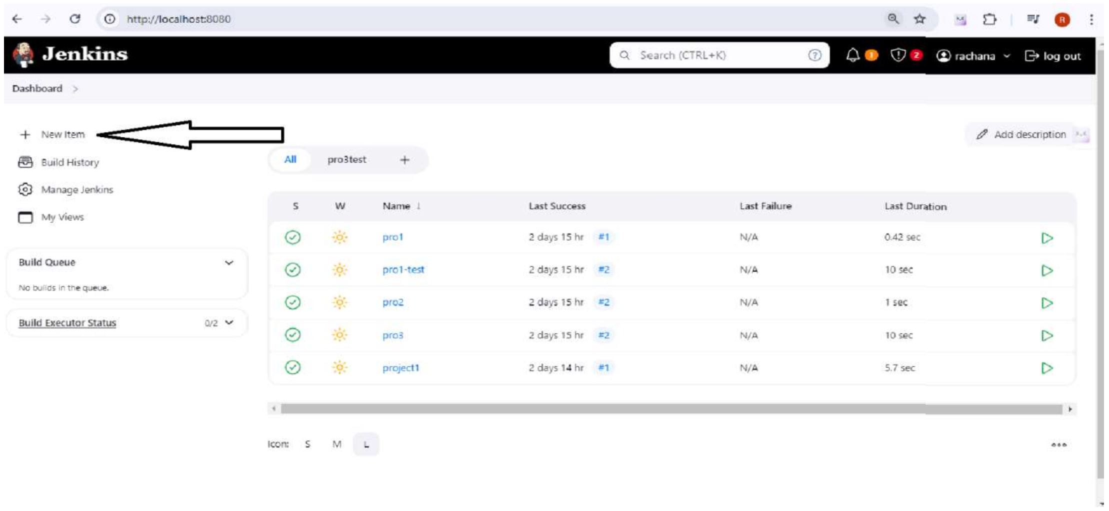
number>) Login to jenkins



ae81-2401-4900-3769-f679-307c-ad3c-4c05-bae8.in.ngrok.io/login?from=%2F

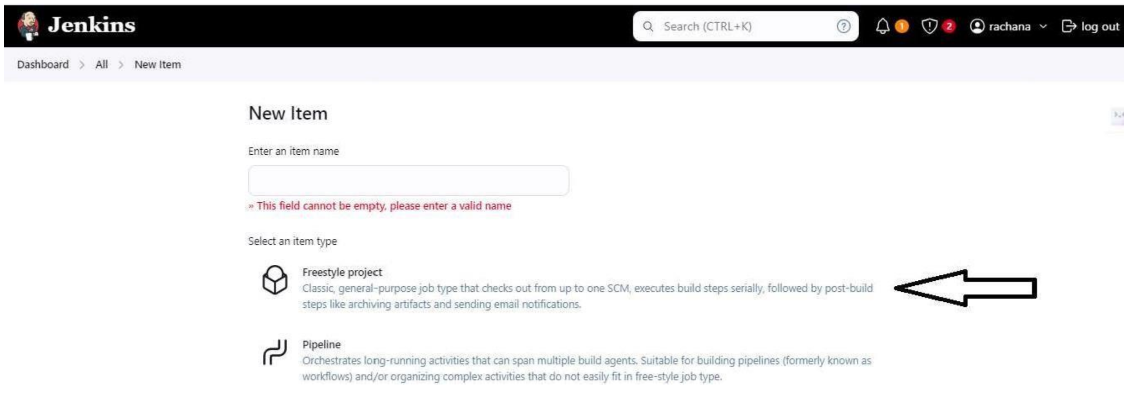


Create new Jenkins job



S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀	pro1	2 days 15 hr #1	N/A	0.42 sec
✓	☀	pro1-test	2 days 15 hr #2	N/A	10 sec
✓	☀	pro2	2 days 15 hr #2	N/A	1 sec
✓	☀	pro3	2 days 15 hr #2	N/A	10 sec
✓	☀	project1	2 days 14 hr #1	N/A	5.7 sec

Step3: Enter the name of new item and Select Free Style project



Jenkins

Dashboard > All > New Item

New Item

Enter an item name

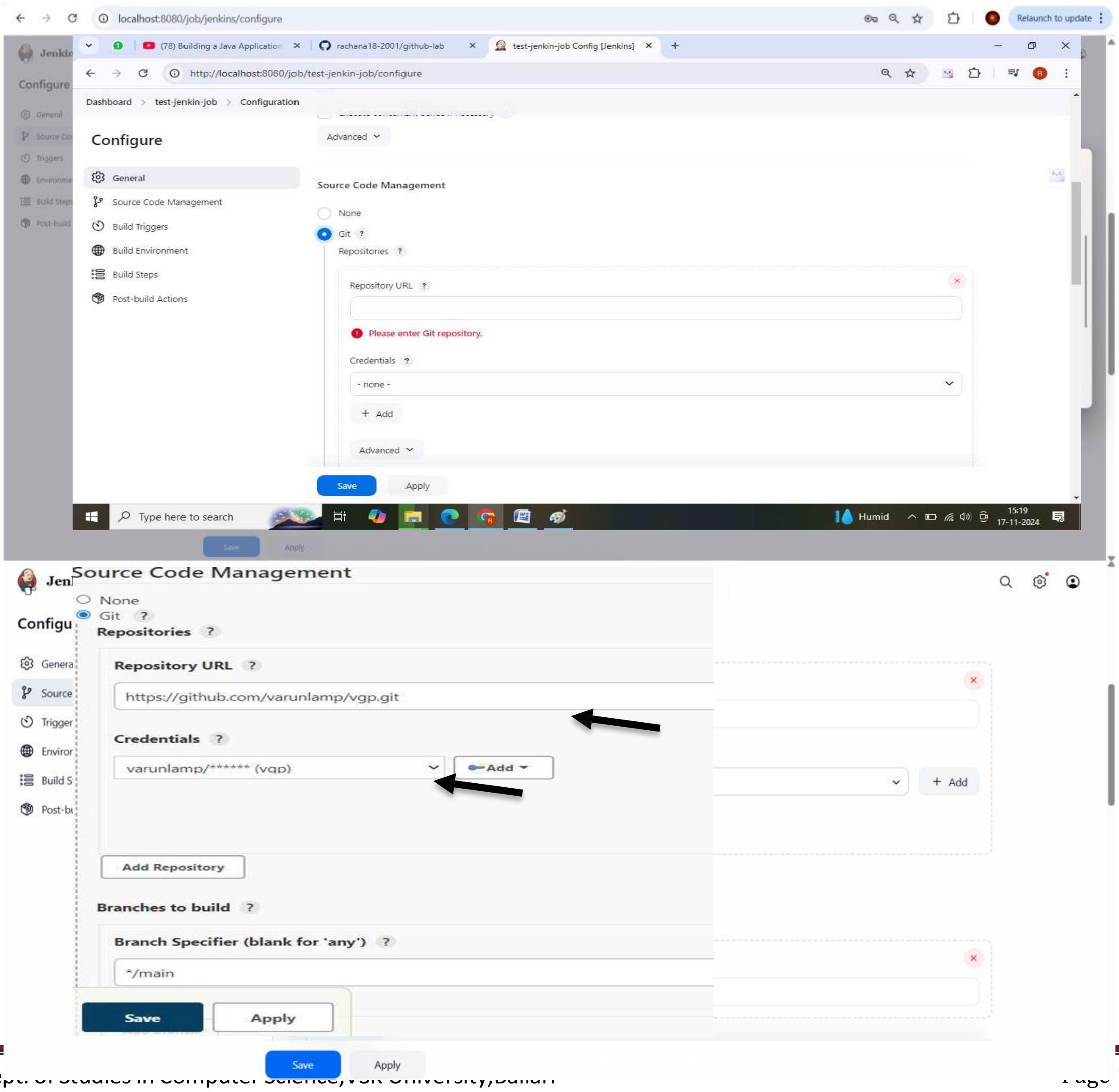
» This field cannot be empty, please enter a valid name

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Step4: Click on git and add git repository URL that has java application.



localhost:8080/job/jenkins/configure

http://localhost:8080/job/test-jenkin-job/configure

Dashboard > test-jenkin-job > Configuration

Configure

General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions

Source Code Management

None
Git

Repositories

Repository URL

Please enter Git repository.

Credentials

- none -

Source Code Management

None
Git

Repositories

Repository URL

https://github.com/varunlamp/vgp.git

Credentials

varunlamp/****** (vgp)

Add Repository

Branches to build

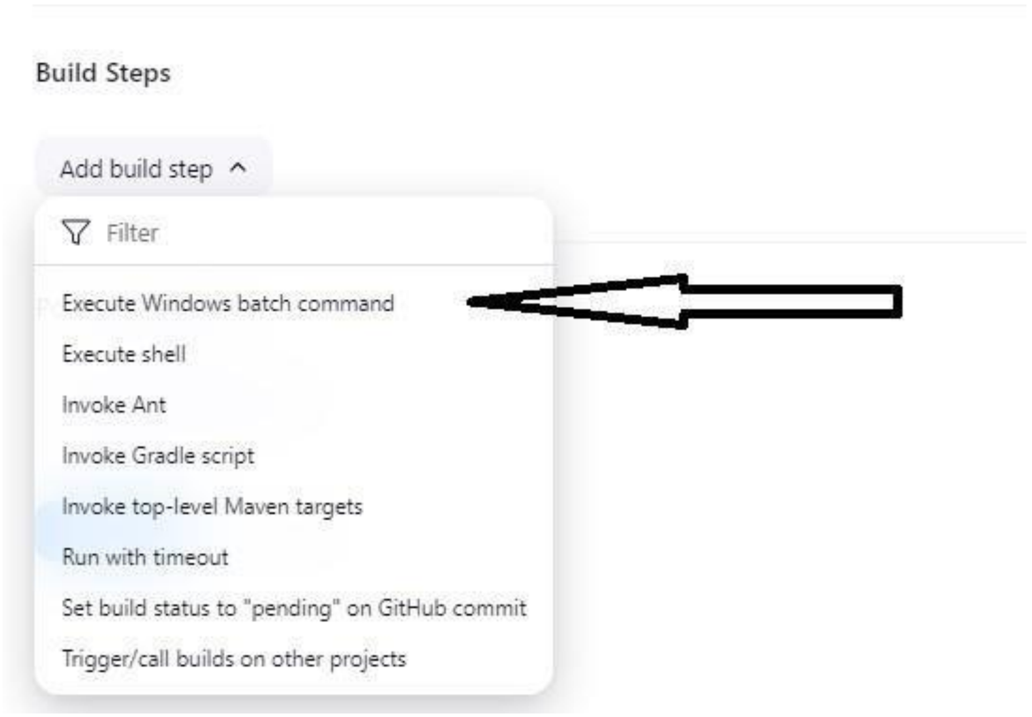
Branch Specifier (blank for 'any')

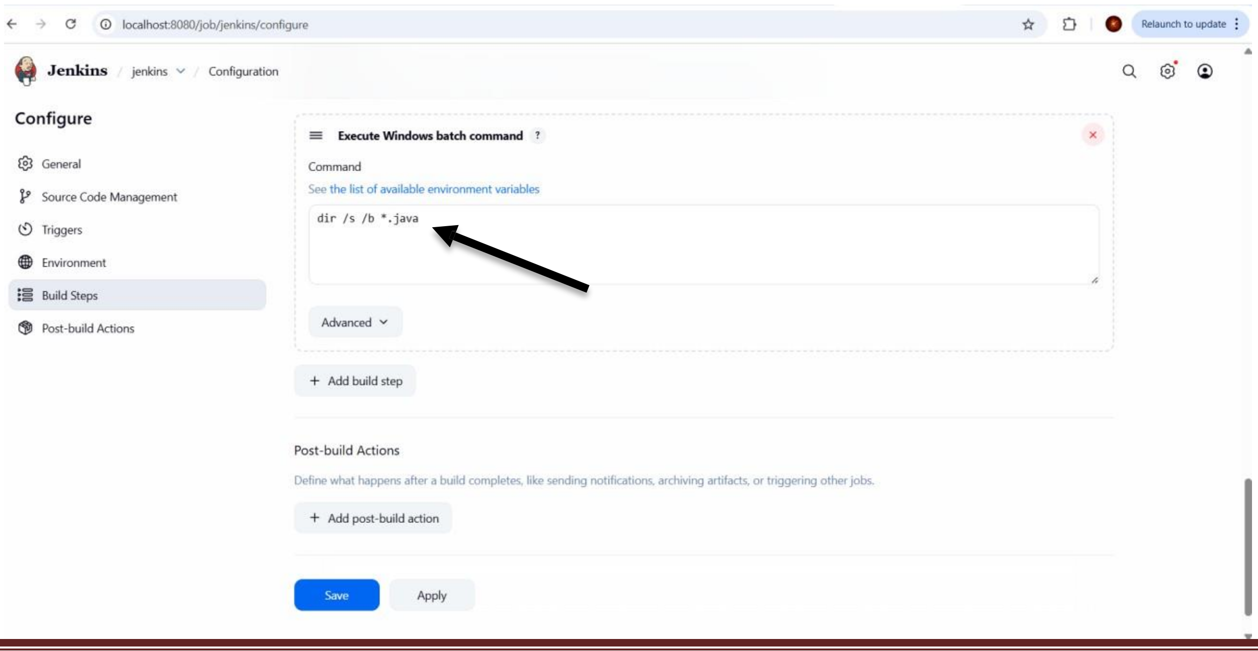
*/main

Step5: Now scroll down and select Add bulid step select **Execute Windows batch command**

Write command: javac test.java

Java test

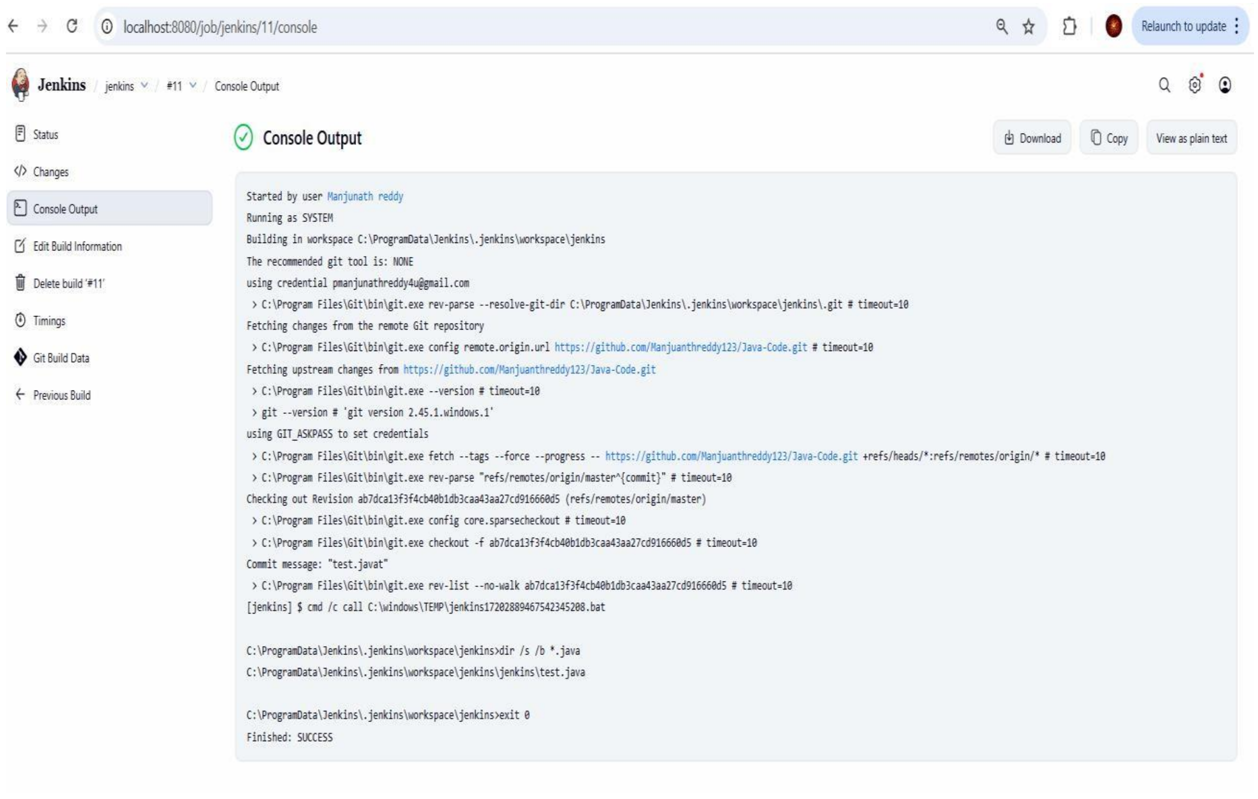

Build Steps
Add build step
Filter
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
Trigger/call builds on other projects


localhost:8080/job/jenkins/configure
Relaunch to update
Jenkins / jenkins / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Execute Windows batch command
Command
See the list of available environment variables
Advanced
Add build step
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Add post-build action
Save
Apply

```
dir /s /b *.java
```

Step7: click on Apply save Build Now

Step8: see the final output in the console



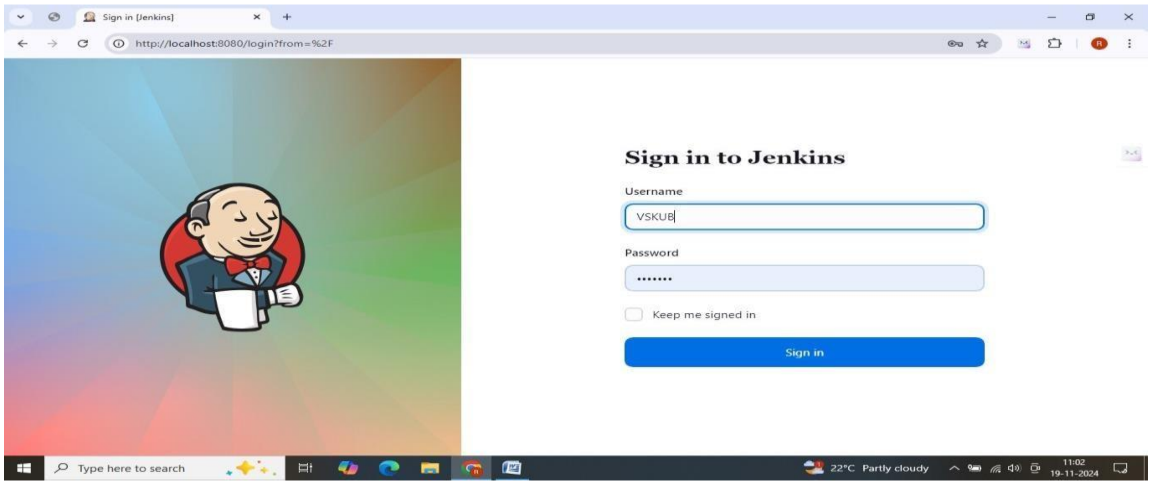
```
Started by user Manjunath reddy
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\jenkins
The recommended git tool is: NONE
using credential pmanjunathreddy4u@gmail.com
 > C:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\jenkins\.git # timeout=10
Fetching changes from the remote Git repository
 > C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/Manjuanthreddy123/Java-Code.git # timeout=10
Fetching upstream changes from https://github.com/Manjuanthreddy123/Java-Code.git
 > C:\Program Files\Git\bin\git.exe --version # timeout=10
 > git --version # 'git version 2.45.1.windows.1'
using GIT_ASKPASS to set credentials
 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/Manjuanthreddy123/Java-Code.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision ab7dca13f3f4cb40b1db3caa43aa27cd916660d5 (refs/remotes/origin/master)
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -f ab7dca13f3f4cb40b1db3caa43aa27cd916660d5 # timeout=10
Commit message: "test.javat"
 > C:\Program Files\Git\bin\git.exe rev-list --no-walk ab7dca13f3f4cb40b1db3caa43aa27cd916660d5 # timeout=10
[jenkins] $ cmd /c call C:\windows\TEMP\jenkins17202889467542345208.bat

C:\ProgramData\Jenkins\.jenkins\workspace\jenkins>dir /s /b *.java
C:\ProgramData\Jenkins\.jenkins\workspace\jenkins\jenkins\test.java

C:\ProgramData\Jenkins\.jenkins\workspace\jenkins>exit 0
Finished: SUCCESS
```

Experiment no: 8

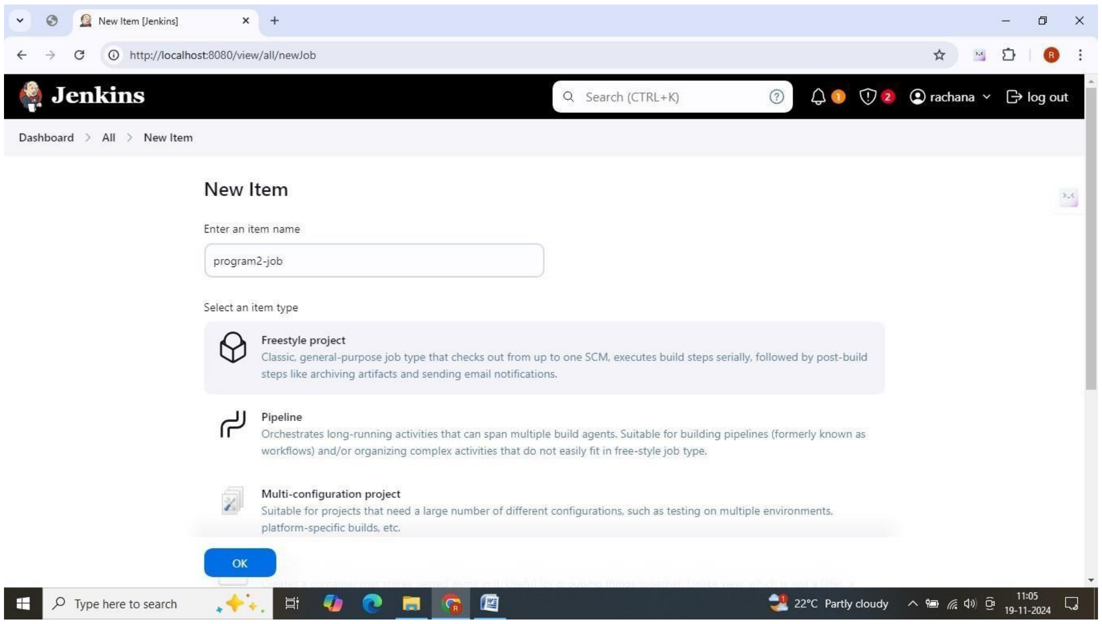
Implement jenkin program using freestyle and pipeline

Implementing Jenkins using free style project

Login to jenkins



Enter new item name and select free style project



New Item

Enter an item name

program2-job

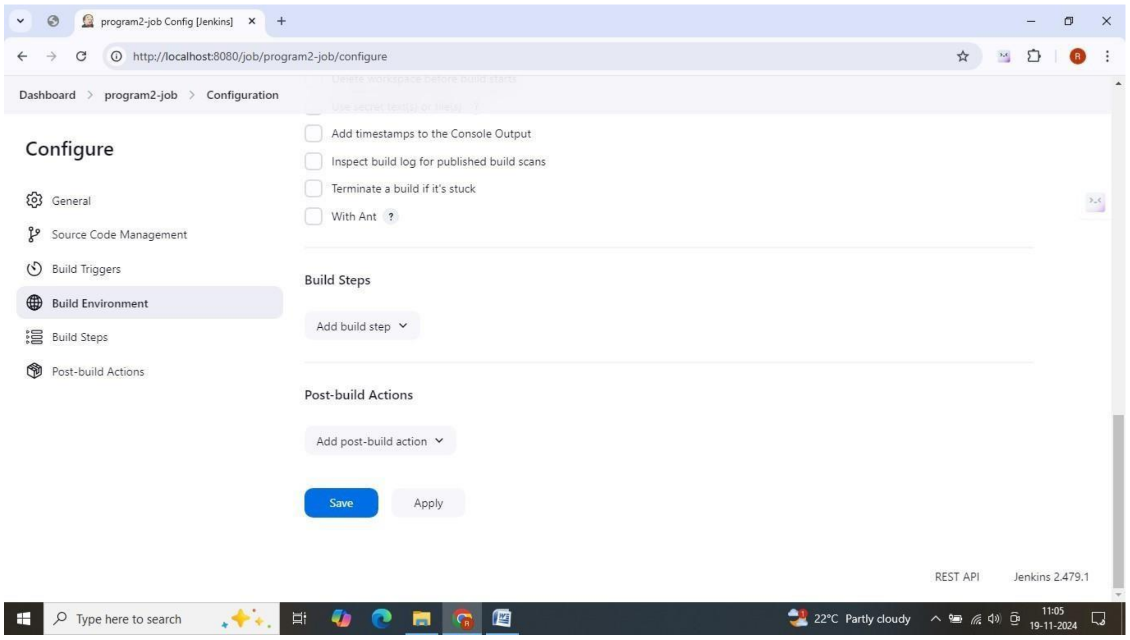
Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

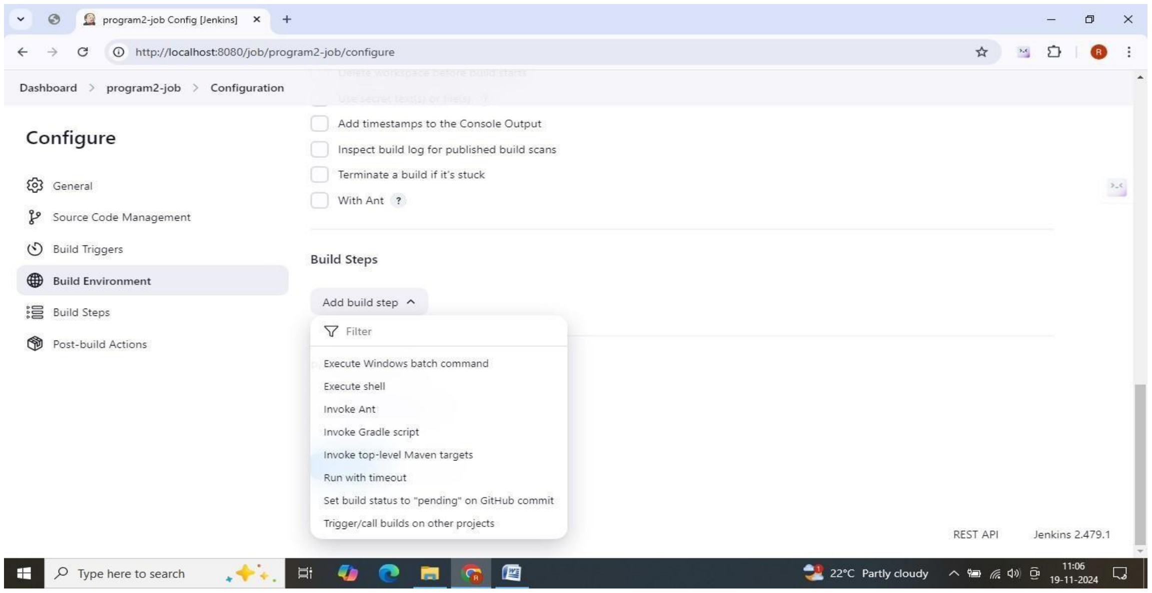
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

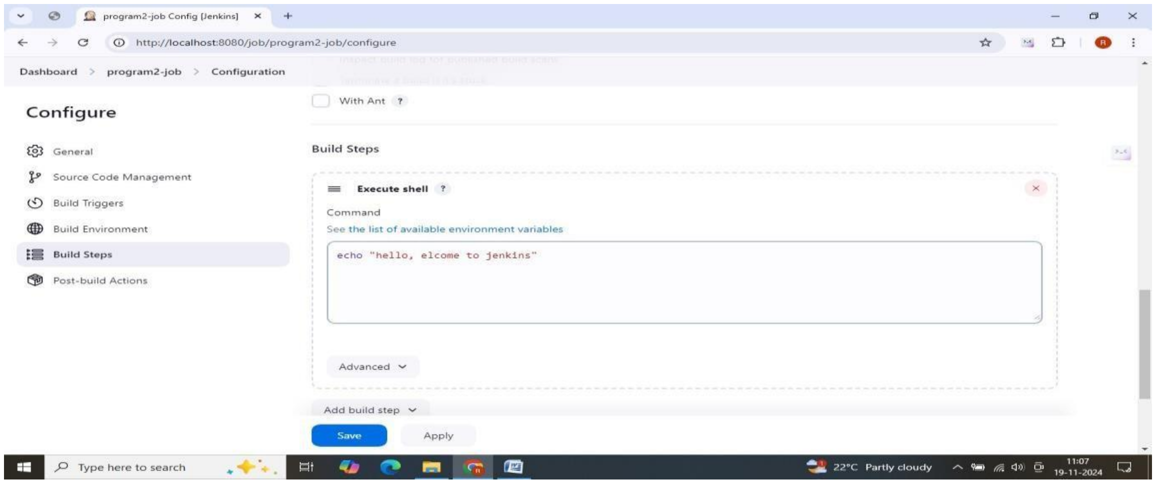
Select build steps


program2-job Config [Jenkins]
http://localhost:8080/job/program2-job/configure
Dashboard > program2-job > Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Add timestamps to the Console Output
Inspect build log for published build scans
Terminate a build if it's stuck
With Ant
Build Steps
Add build step
Post-build Actions
Add post-build action
Save
Apply
REST API
Jenkins 2.479.1

Click on execute shell

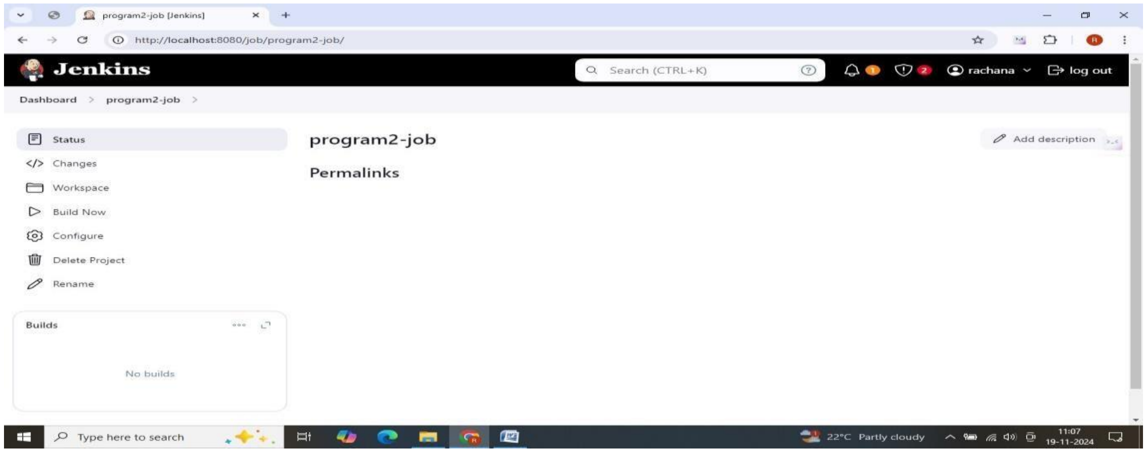

program2-job Config [Jenkins]
http://localhost:8080/job/program2-job/configure
Dashboard > program2-job > Configuration
Configure
Build Steps
Add build step
Filter
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
Trigger/call builds on other projects

Write command Save and apply

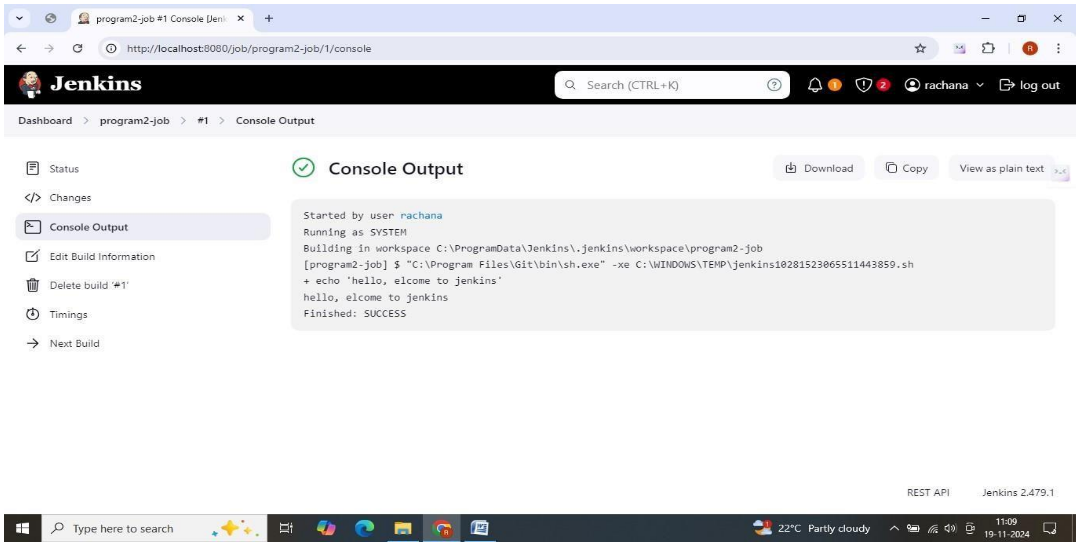


```
echo "hello, elcome to jenkins"
```

Click on build now and see the output



OUTPUT:

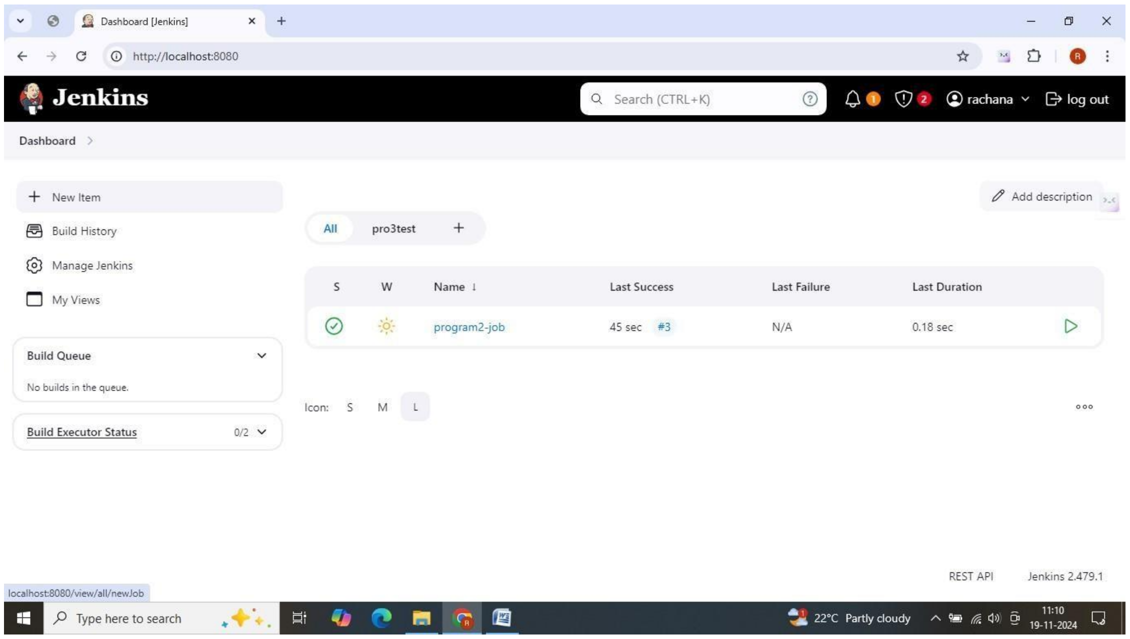


Console Output

```
Started by user rachana
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\program2-job
[program2-job] $ "C:\Program Files\Git\bin\sh.exe" -xe C:\WINDOWS\TEMP\jenkins10281523065511443859.sh
+ echo 'hello, elcome to jenkins'
hello, elcome to jenkins
Finished: SUCCESS
```

Implementing Jenkin program using pipeline

Now create new pipeline job



Dashboard [Jenkins]
http://localhost:8080

Jenkins — Search (CTRL+K) — rachana — 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 | pro3test | +

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀	program2-job	45 sec #3	N/A	0.18 sec

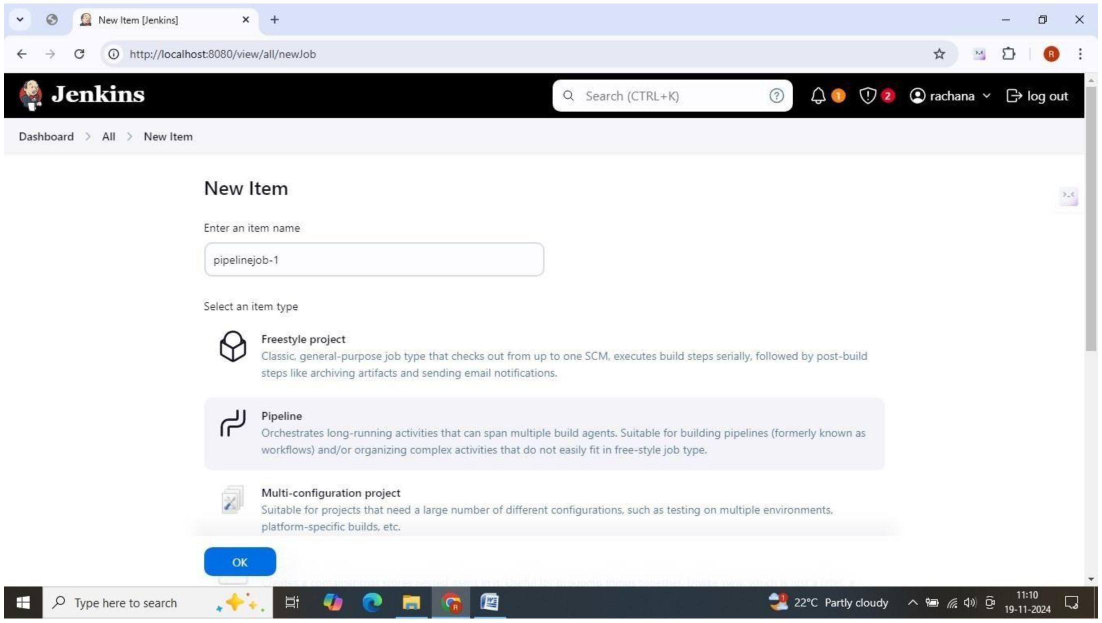
Icon: S M L

REST API Jenkins 2.479.1

localhost:8080/view/all/newJob

Click on new item

Name the job and click on pipeline



New Item [Jenkins]
http://localhost:8080/view/all/newJob

Dashboard > All > New Item

New Item

Enter an item name

pipelinejob-1

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

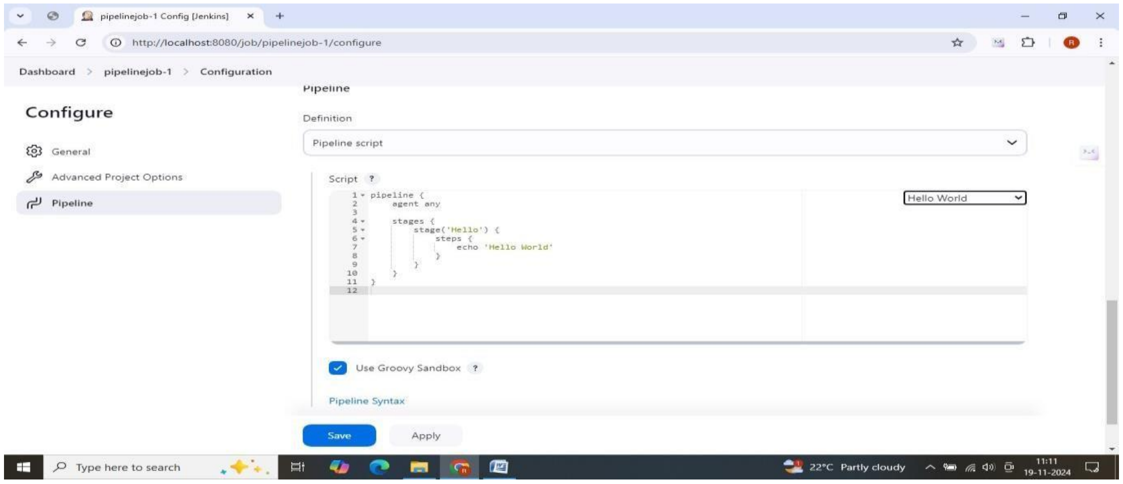
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

Write descriptive code

```
pipeline {

   agent

   any

   stages {

      stage('Hello')

         { steps {

            echo 'Hello World'
         }
      }
   }
}
```

Click on apply and save



pipelinejob-1 Config [Jenkins]

http://localhost:8080/job/pipelinejob-1/configure

Dashboard > pipelinejob-1 > Configuration

Configure

Pipeline

Definition: Pipeline script

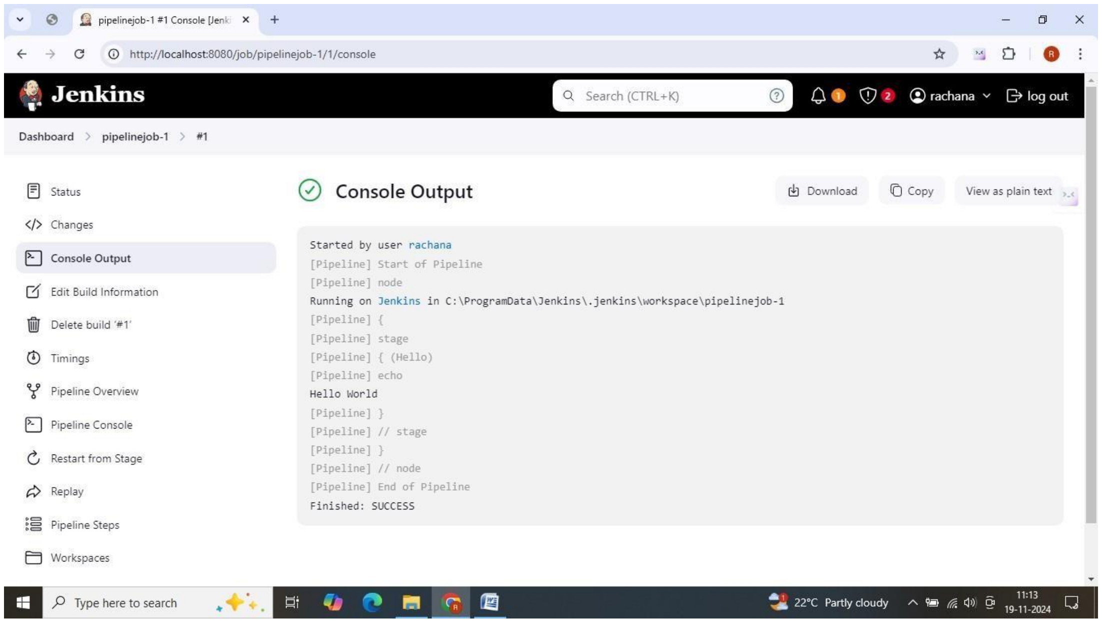
Script

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

Hello World

✓ Use Groovy Sandbox

Click on build now and see the output



Console Output

```
Started by user rachana
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\pipelinejob-1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```