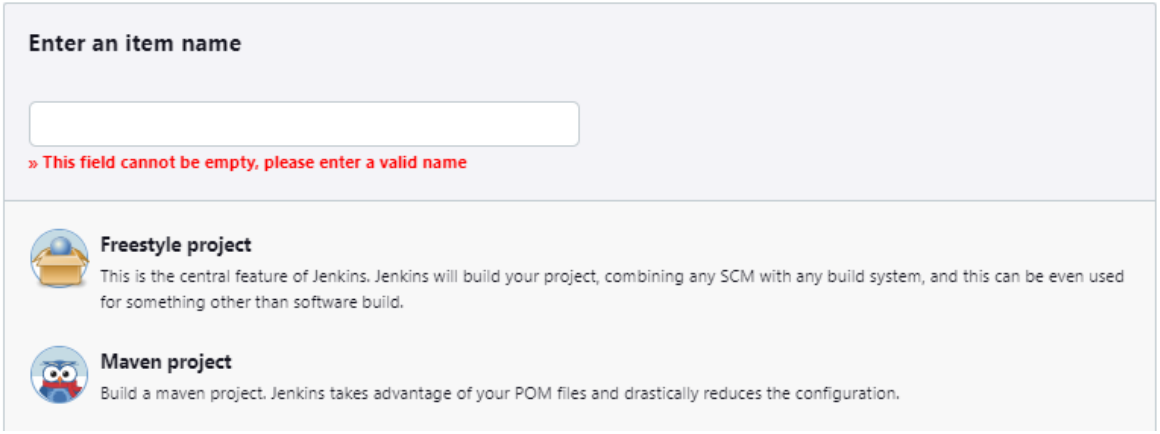
1. **Steps for running Maven Project in Jenkins (Without pipeline)**

Create a New Jenkins Job:

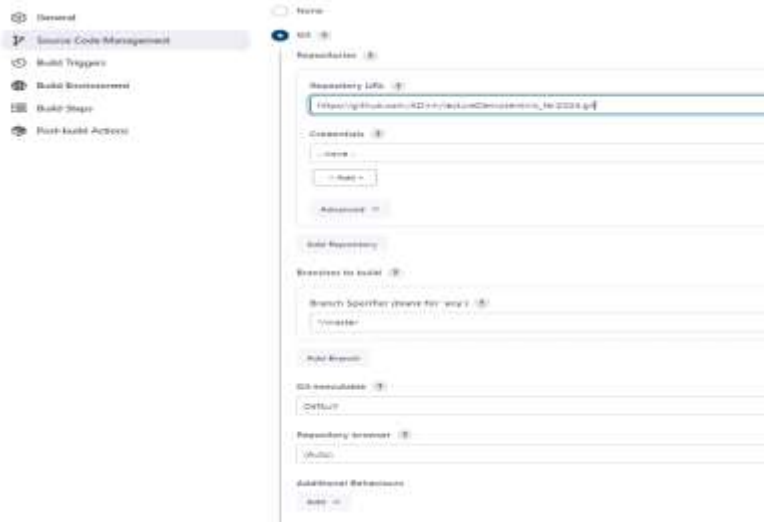
- Go to Jenkins Dashboard and select "New Item."



- Enter a name for your job and select "Freestyle project" as the job type.
- Click "OK" to create the job.

Configure Source Code Management:

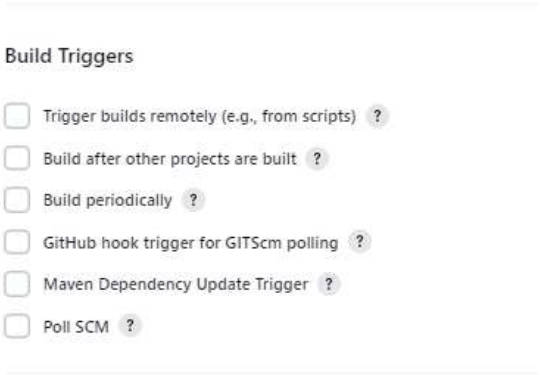
- In the job configuration page, scroll down to the "Source Code Management" section.
- Choose "Git" and enter the URL of your GitHub repository.



- Optionally, specify the branch you want to build.

Set Up Build Triggers:

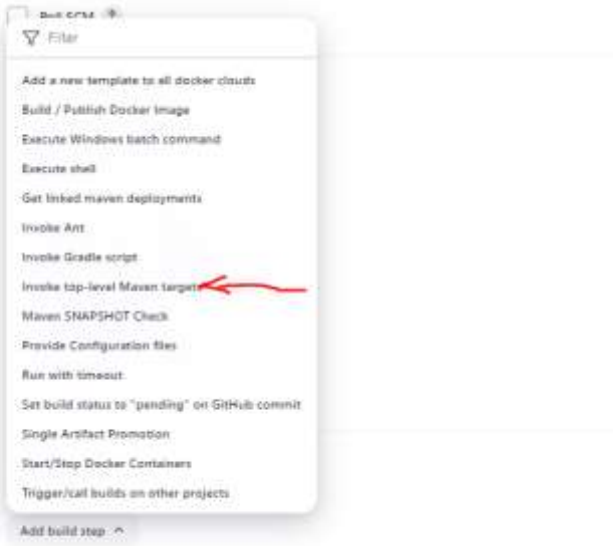
- Scroll down to the "Build Triggers" section.



- Choose how you want Jenkins to trigger a build. For example, you can choose to build periodically or trigger builds manually.

Configure Build Steps:

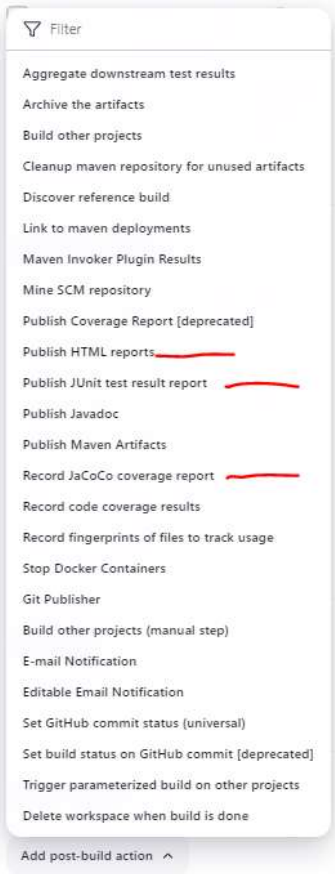
- In the "Build" section, click on "Add build step" and choose "Invoke top-level Maven targets."
- Specify the Maven version you want to use (configure Maven in Jenkins global settings if you haven't already).



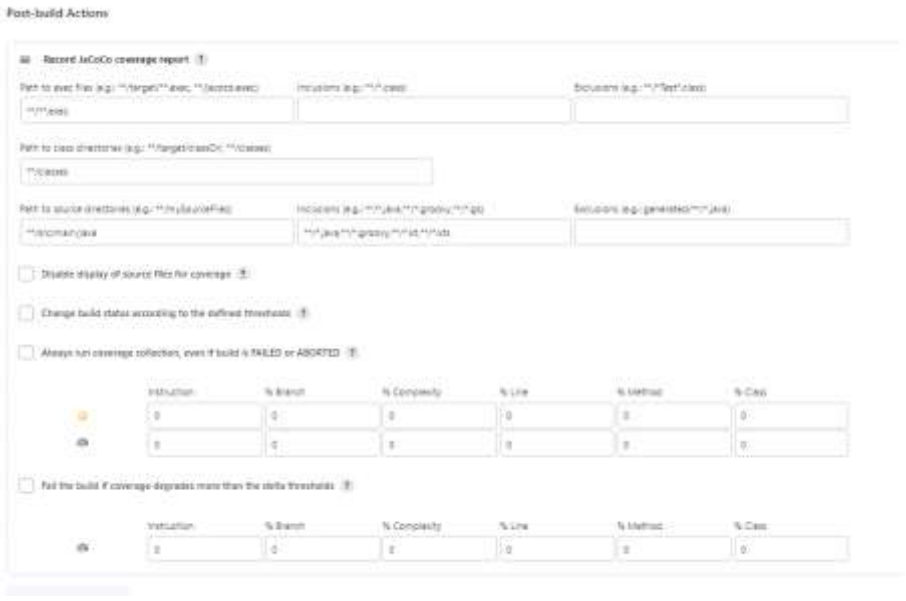
- Enter the Maven goals you want Jenkins to execute, such as clean install.

Set Up Post-Build Actions (Optional):

- If you want Jenkins to perform any actions after the build, such as archiving artifacts or sending notifications, you can configure post-build actions in the "Post-build Actions" section.

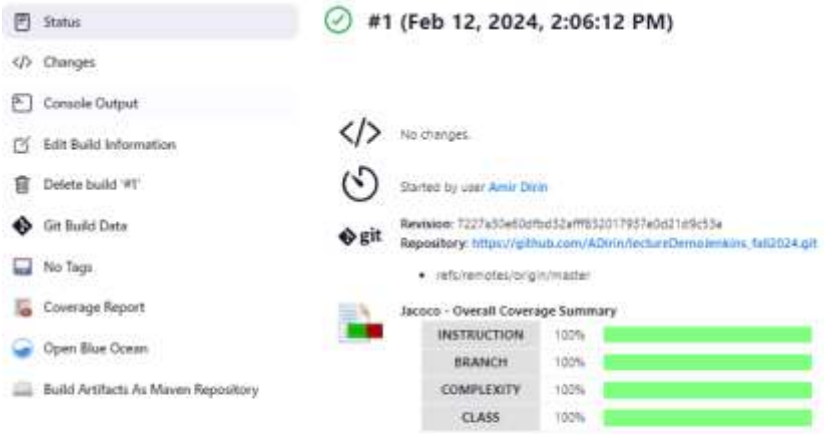


For example, JACOCO coverage report

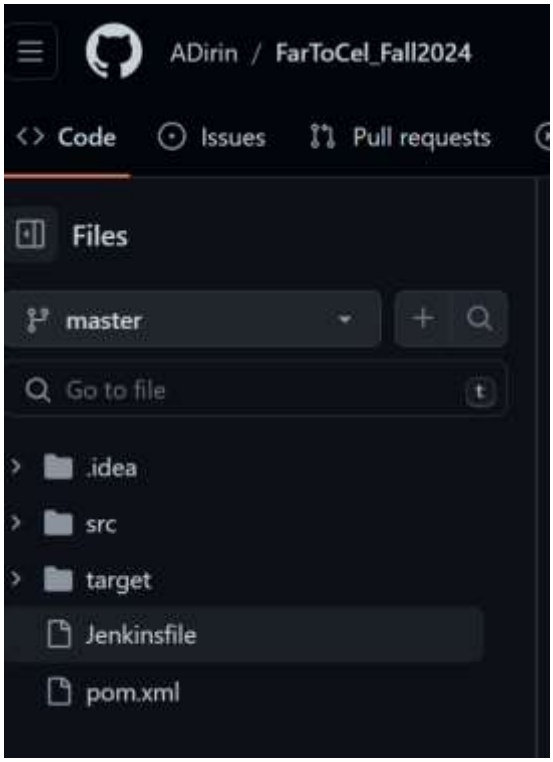


Save your Configuration: Once you've configured your job, save the configuration**.**

Run the Job: Click on "Build Now" to trigger a build of your Maven project. Jenkins will clone your repository, execute the Maven build commands specified in your job configuration, and provide feedback on the build status.



2. **Steps to create a pipeline Item to read pipeline script in GitHub**




```
pipeline {
    agent any
    environment {
        PATH = "${env.PATH};C:\\Windows\\System32" // Update the PATH to include the directory o
        GIT_CREDENTIALS = credentials('ADirin')
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'master', credentialsId: 'ADirin', url: 'https://github.com/ADirin/F
            }
        }
```

Configure GitHub Integration:

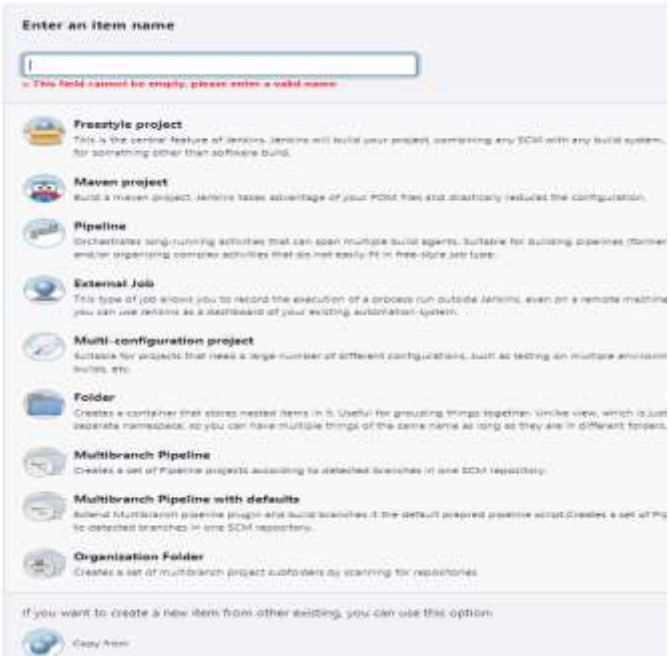
- Go to Jenkins Dashboard > Manage Jenkins > Configure System.

- Scroll down to the "GitHub" section.

- Configure GitHub server details, including GitHub API credentials. You might need to generate personal access tokens in GitHub for this purpose.
- Set Up Credentials: If Jenkins needs access to your GitHub repository, you'll need to set up credentials. These could be SSH keys or personal access tokens, depending on your security requirements.

Create a New Pipeline Job:

- Go to Jenkins Dashboard and select "New Item."



- Enter a name for your job and select "Pipeline" as the job type.
- Click "OK" to create the job.

Configure Pipeline Source:

- In the job configuration, scroll down to the "Pipeline" section.

- Choose "Pipeline script from SCM" as the Definition.
- Select "Git" as the SCM.
- Enter the URL of your GitHub repository.
- Choose the credentials you configured earlier, if necessary.
- Specify the branch or tag containing your Jenkinsfile.

Save and Run: Save your job configuration, and Jenkins will automatically start running the pipeline defined in your Jenkinsfile whenever changes are pushed to the specified branch or tag in your GitHub repository.

Verify Integration: Make a change to your Jenkinsfile in the GitHub repository, commit and push it. Jenkins should automatically detect the change and start a build based on the updated Jenkinsfile.

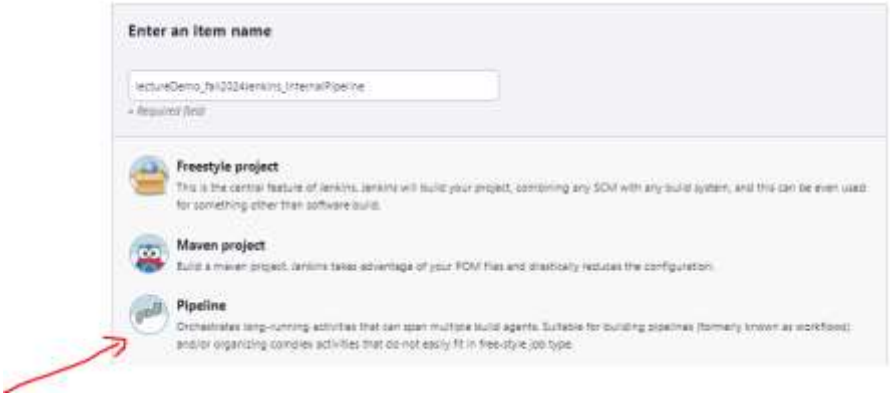
3. **Steps to create a pipeline Item to read pipeline script in GitHub**

Create a Jenkins Pipeline:

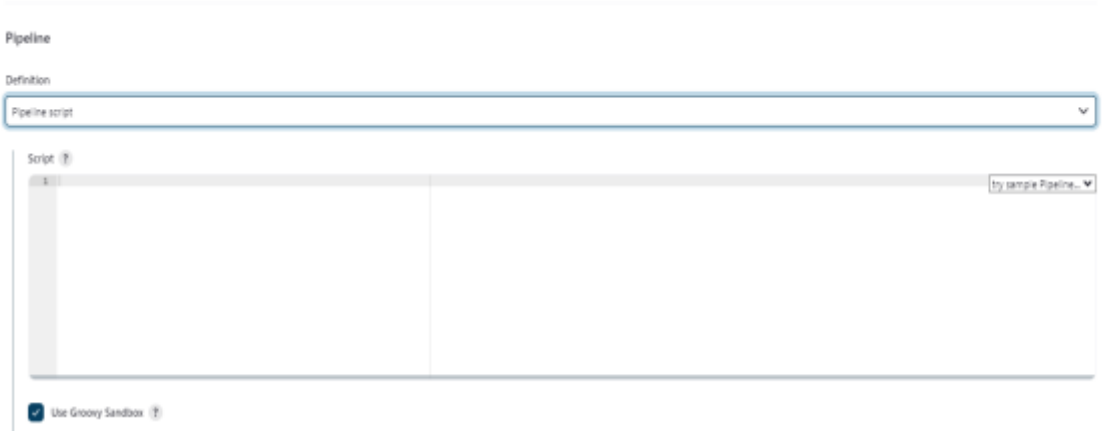
- Go to your Jenkins dashboard.
- Click on "New Item" to create a new pipeline job.
- Enter a name for your pipeline job and select "Pipeline" as the job type.
- Click "OK" to create the job.

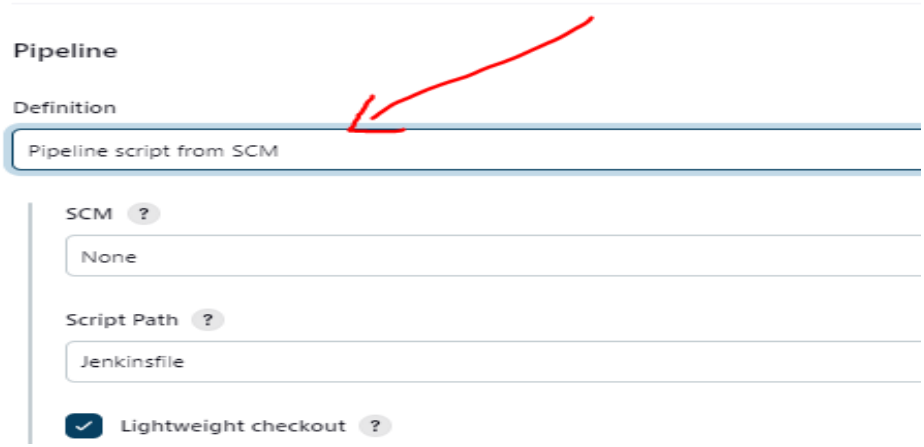
Configure Pipeline Source:

- In the pipeline configuration, scroll down to the "Pipeline" section.

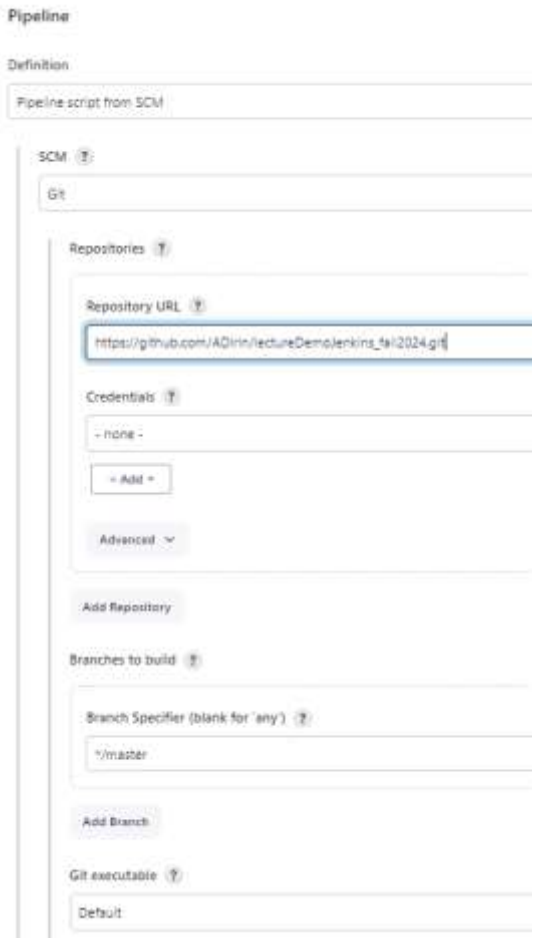


- Select "Pipeline script from SCM" as the Definition.





- Choose "Git" as the SCM.



- Enter the repository URL for your GitHub project.
- Optionally, specify the branch to build.
- Click "Save".

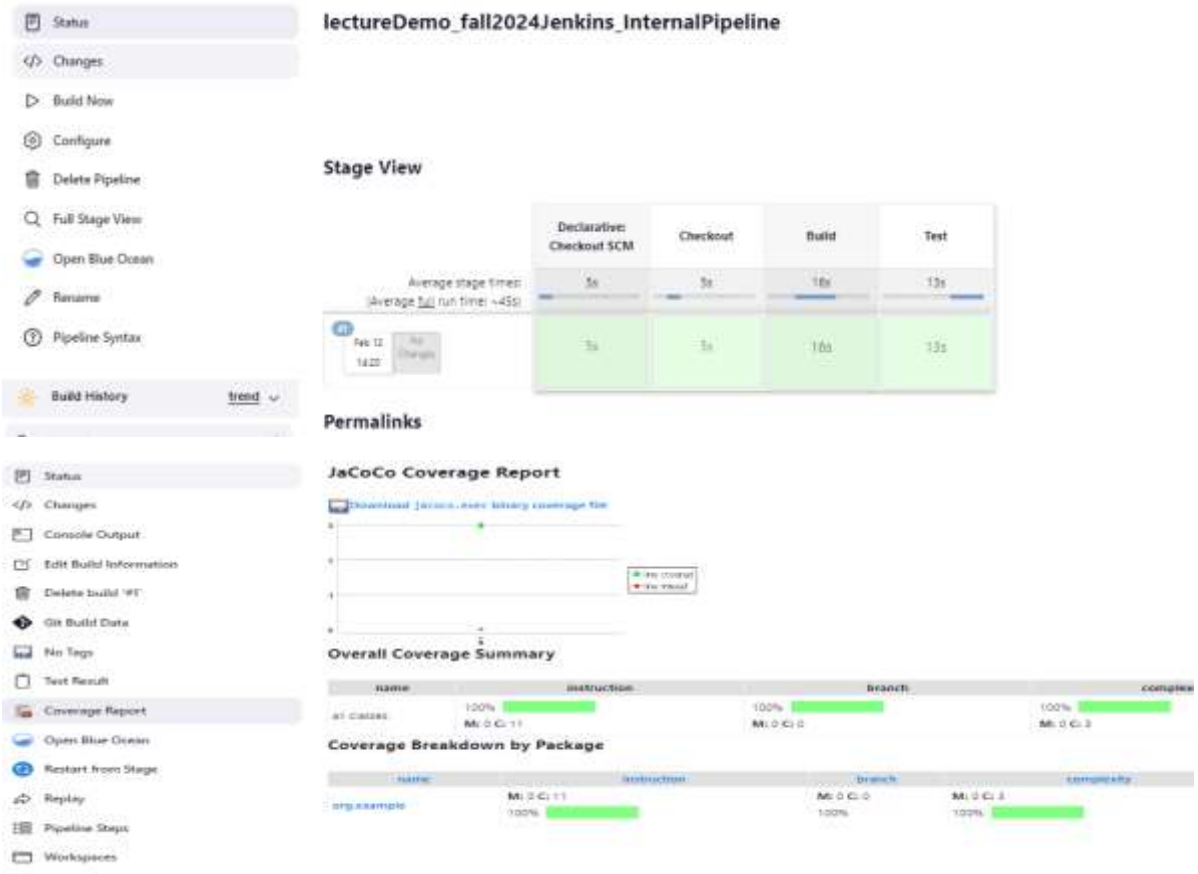
Write Pipeline Script:

- In the pipeline configuration, scroll down to the Pipeline section.
- In the Script textbox, write the Jenkinsfile script to define your pipeline stages. Below is a basic example:

"*See APPENDIX A for a Sample Pipeline*"

Save and Run Pipeline:

- Save the pipeline configuration.
- Click on "Build Now" to trigger the pipeline execution.



APPENDIX A

Sample pipeline to create a coverage report and perform unit testing in Jenkins.

```
pipeline {
    agent any
    environment {
        PATH = "${env.PATH};C:\\Windows\\System32" // Update the PATH to include the directory of cmd.exe
        GIT_CREDENTIALS = credentials('ADirin')
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'master', credentialsId: 'ADirin', url: 'https://github.com/ADirin/FarToCel_Fall2024.git'
            }
        }
        stage('Build') {
            steps {
                bat 'mvn clean install'
            }
        }
        stage('Test') {
            steps{
                bat 'mvn test'
            }
            post {
                success {
                    // Publish JUnit test results
                    junit '**/target/surefire-reports/TEST-*.xml'
                    // Generate JaCoCo code coverage report
                    jacoco(execPattern: '**/target/jacoco.exec')
                }
            }
        }
    }
}
```

Introduction To Jenkins
Amir Dirin Fall 2024

The details of the syntax are as fellow:

pipeline { //This line marks the beginning of the pipeline block.

agent any //This specifies that the pipeline can be executed on any available agent (or executor) in the Jenkins environment.

environment { // Starts the environment block, where you can define environment variables for the pipeline.

PATH = "${env.PATH};C:\\Windows\\System32" // This line updates the PATH environment variable to include the directory of cmd.exe on a Windows system. It concatenates the existing PATH with C:\Windows\System32.

GIT_CREDENTIALS = credentials('ADirin') // This sets the environment variable GIT_CREDENTIALS to the credentials stored in Jenkins with the ID 'ADirin'.

} //Closes the environment block.

stages { // Starts the stages block, where you define the different stages of the pipeline.

stage('Checkout') { // Defines the first stage of the pipeline, named 'Checkout'.

steps { // Starts the steps block for the 'Checkout' stage.

git branch: 'master', credentialsId: 'ADirin', url: 'https://github.com/ADirin/FarToCel_Fall2024.git' // This step checks out the 'master' branch of the Git repository located at the specified URL using the credentials stored in Jenkins with the ID 'ADirin'.

} // Closes the steps block for the 'Checkout' stage.

} //Closes the stage block for the 'Checkout' stage.

stage('Build') { //Defines the second stage of the pipeline, named 'Build'.

steps { //Starts the steps block for the 'Build' stage.

bat 'mvn clean install' //This step executes a Maven command (mvn clean install) using the Windows batch script.

} //Closes the steps block for the 'Build' stage.

} //Closes the stage block for the 'Build' stage.

stage('Test') { //Defines the third stage of the pipeline, named 'Test'.

steps { // Starts the steps block for the 'Test' stage.

bat 'mvn test' //This step executes the Maven command mvn test using the Windows batch script.

} //Closes the steps block for the 'Test' stage.

post { // Starts the post block, which contains actions to be performed after the stage is completed.

success { //Specifies that the actions inside this block should only be executed if the stage is successful.

*junit '**/target/surefire-reports/TEST-*.xml'* //This publishes JUnit test results by specifying the file pattern for the XML reports generated by Maven Surefire plugin.

*jacoco(execPattern: '**/target/jacoco.exec')* //This generates JaCoCo code coverage report by specifying the file pattern for the JaCoCo execution data.

} //Closes the success block.

} //Closes the post block.

} //Closes the stages block.

} //Closes the pipeline block.