**Testing Code Coverage**

This section will guide you to:

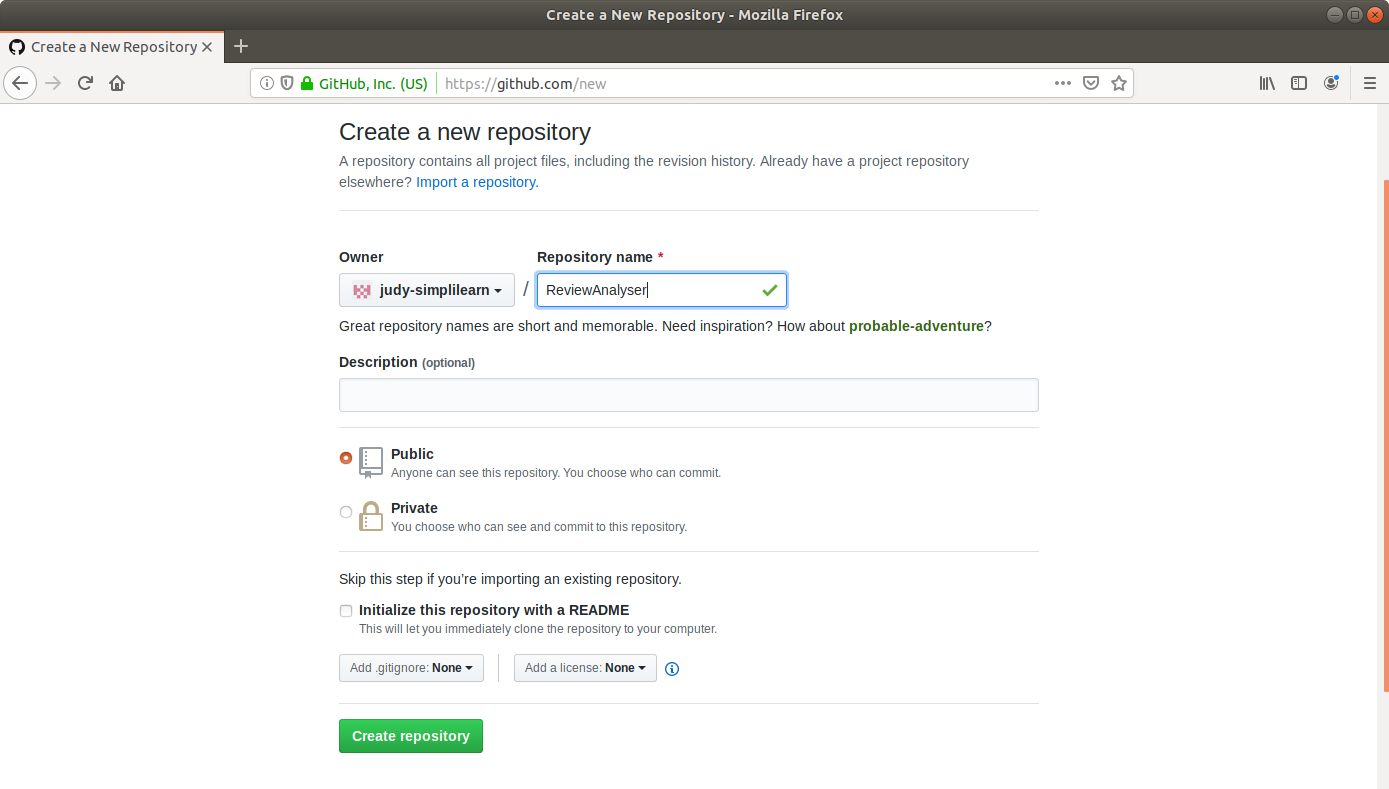
* Create and commit a Jenkinsfile which uses JaCoCo
* Configure a pipeline to run from a Jenkinsfile and publish coverage results  
    
  **Please Note**: This demo is incremental, you should have completed the previous demos in order to proceed with this demo.

**Step 1:** Creating a Git repository for the Review Analyzer program

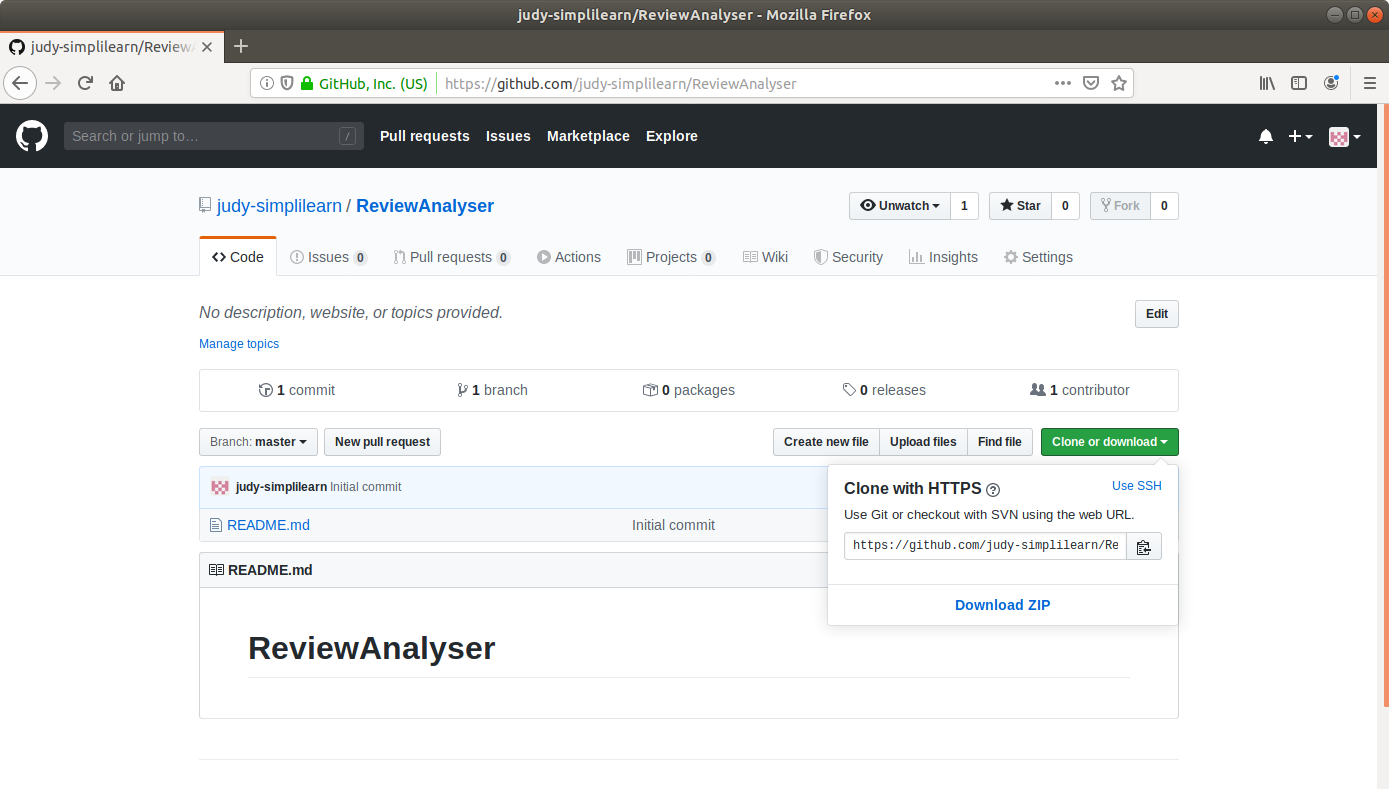
* Login to your **Github** account.
* Click on the plus icon next to the profile picture and select ***New repository*** from the drop-down menu.



* Fill the required fields in the Create Repository form.

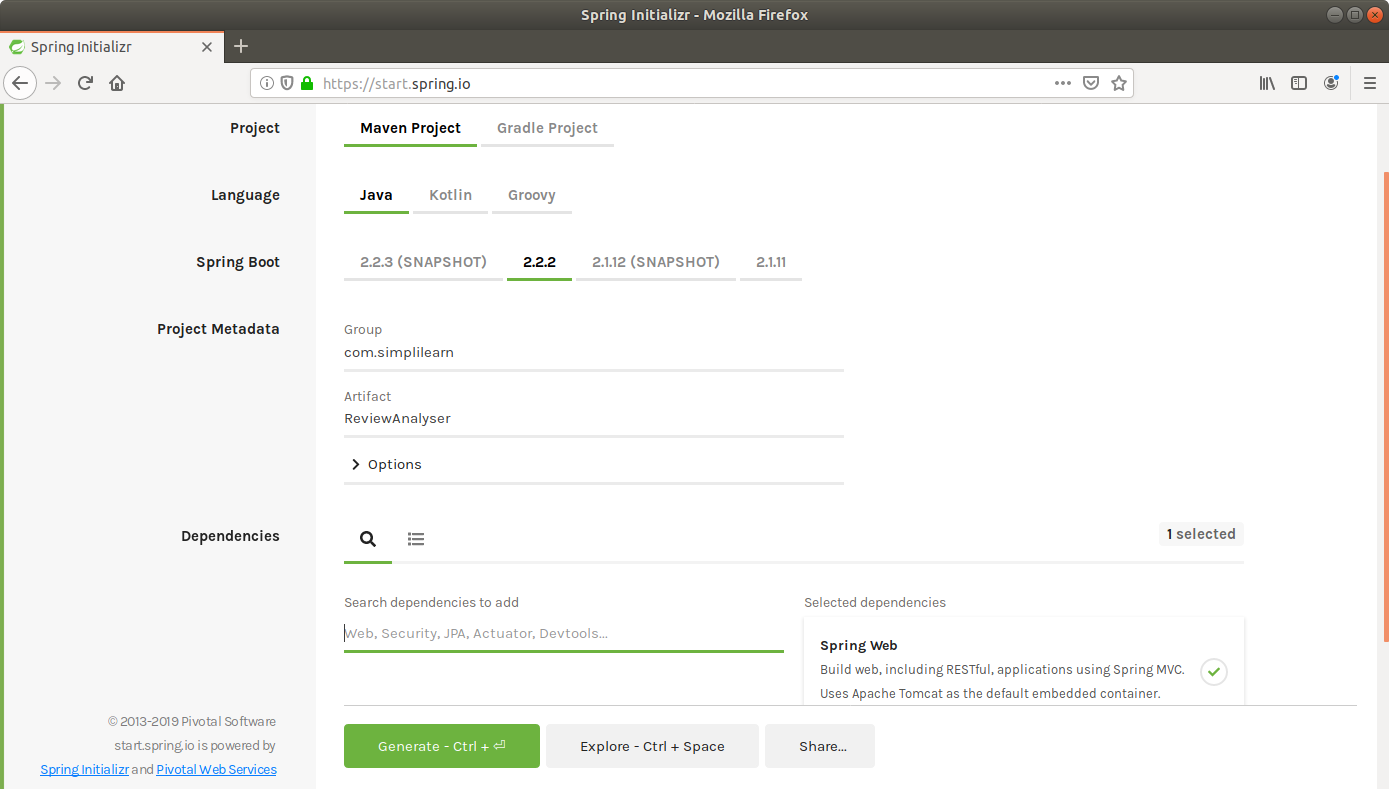


* Click on the **Create Repository** button.
* Click on the **Clone or download** button and copy the URL.



**Step 2:** Generating a spring boot project

* Go to ​start.​spring.​io/​



* Select Maven as the project type.
* Fill Group and Artifact with appropriate values. For example, *com.simplilearn* and *ReviewAnalyser.*
* Add **Web** to Dependencies.
* Select Packaging: Jar
* Select Java: 8
* Click on **Generate Project.**
* The generated skeleton project should be downloaded as a zip file.

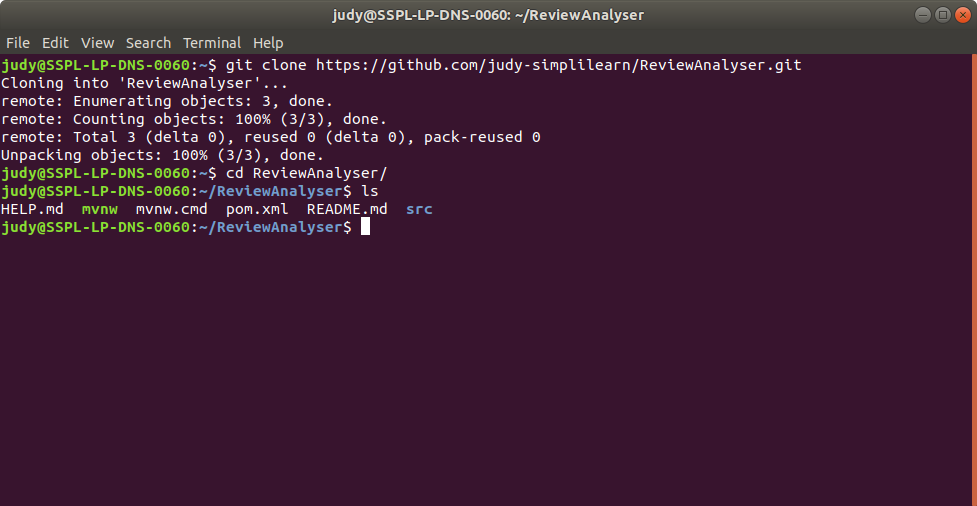
**Step 3:** Adding the code for word count to the repository

* Open the terminal and navigate to an appropriate location.
* Run **git clone [URL]** to clone the repository.
* Unzip the downloaded spring boot project to the cloned repository.

(cd Downloads

unzip Review-Analyser.zip)

Copy the contents of Review-Analyser folder present in dowsnloads and paste it into your repository folder)



* Navigate to the *ReviewAnalyser* folder within the ***src*** folder.
* Open the **ReviewAnalyserApplication.java** in a text editor.
* Add the following method to the file and save it.

**package com.simplilearn.ReviewAnalyser;**

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication;**

**@SpringBootApplication**

**public class ReviewAnalyserApplication {**

**public static double getWordCount(String review){**

**int count = 0;**

**String string[] = review.toLowerCase().split("([,.\\s]+)");**

**for(String s : string){**

**count++;**

**}**

**return count;**

**}**

**public static void main(String[] args) {**

**SpringApplication.run(ReviewAnalyserApplication.class, args);**

**}**

**}**

* Navigate to the *ReviewAnalyser* folder within the ***test*** folder.
* Open the **ReviewAnalyserApplicationTests.java** in a text editor.
* Add the following *test* method to the file and save it.

**package com.simplilearn.ReviewAnalyser;**

**import org.junit.Test;**

**import static org.junit.Assert.\*;**

**import org.springframework.boot.test.context.SpringBootTest;**

**@SpringBootTest**

**class ReviewAnalyserApplicationTests {**

**private ReviewAnalyserApplication analyser = new ReviewAnalyserApplication();**

**@Test**

**public void testWordCount() {**

**assertEquals(7,analyser.getWordCount("Train to win in the digital economy"));**

**}**

**}**

* Save the file and exit the text editor.
* Open the **pom.xml** and add the following dependency.

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit-dep</artifactId>**

**<version>4.8.2</version>**

**<scope>test</scope>**

**</dependency>**

* Add the jacoco plugin to **pom.xml** with the following xml code:

**<plugin>**

**<groupId>org.jacoco</groupId>**

**<artifactId>jacoco-maven-plugin</artifactId>**

**<version>0.8.3</version>**

**<executions>**

**<execution>**

**<id>default-prepare-agent</id>**

**<goals>**

**<goal>prepare-agent</goal>**

**</goals>**

**</execution>**

**<execution>**

**<id>default-report</id>**

**<phase>prepare-package</phase>**

**<goals>**

**<goal>report</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

* Save the file and exit the text editor.

**Step 4:** Creating and committing a Jenkinsfile

* Navigate to the *ReviewAnalyser* root directory where the pom.xml is.
* Open a new text file **vi Jenkinsfile** and add the following script to it.

**pipeline {**

**agent any**

**stages {**

**stage("Compile") {**

**steps {**

**sh "mvn compile"**

**}**

**}**

**stage("Unit test") {**

**steps {**

**sh "mvn test"**

**}**

**}**

**}**

**post {**

**always {**

**step([$class: 'JacocoPublisher',**

**execPattern: 'target/\*.exec',**

**classPattern: 'target/classes',**

**sourcePattern: 'src/main/java',**

**exclusionPattern: 'src/test\*'**

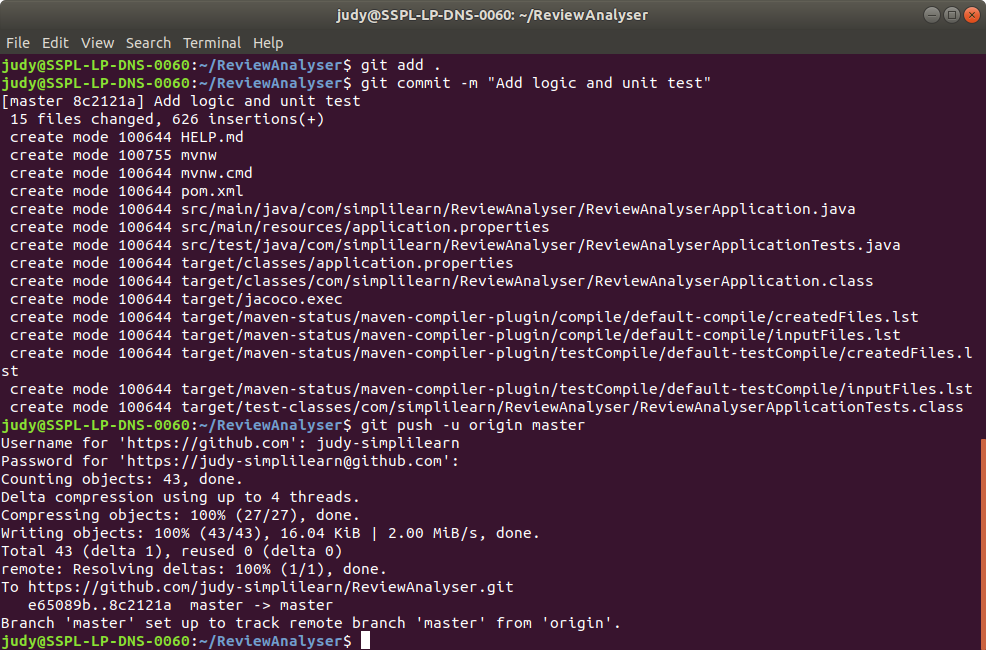
**])**

**}**

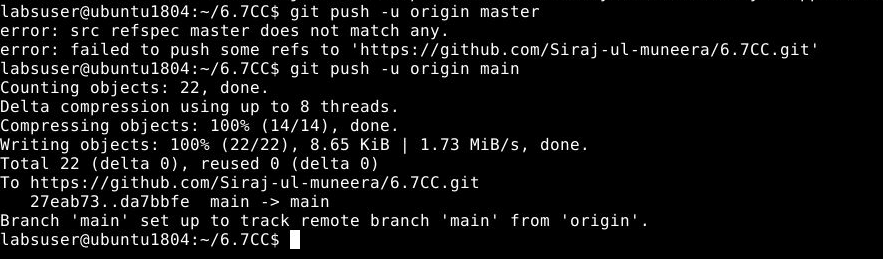
**}**

**}**

* Save the file as **Jenkinsfile** with no extension.
* Commit the changes to the remote SCM.
* Run **git add .**
* Run **git commit -m “Add logic and test”**
* Run **git push -u origin master**

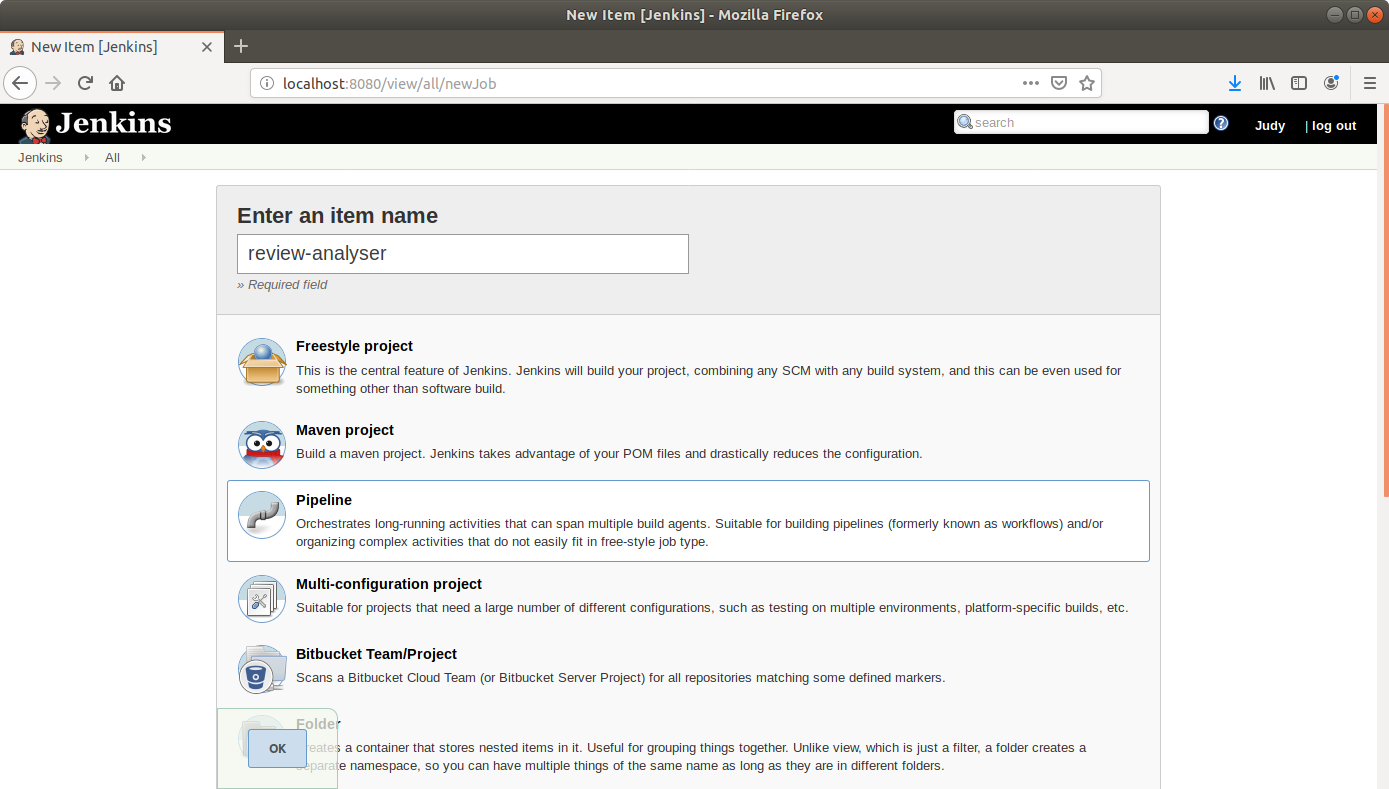


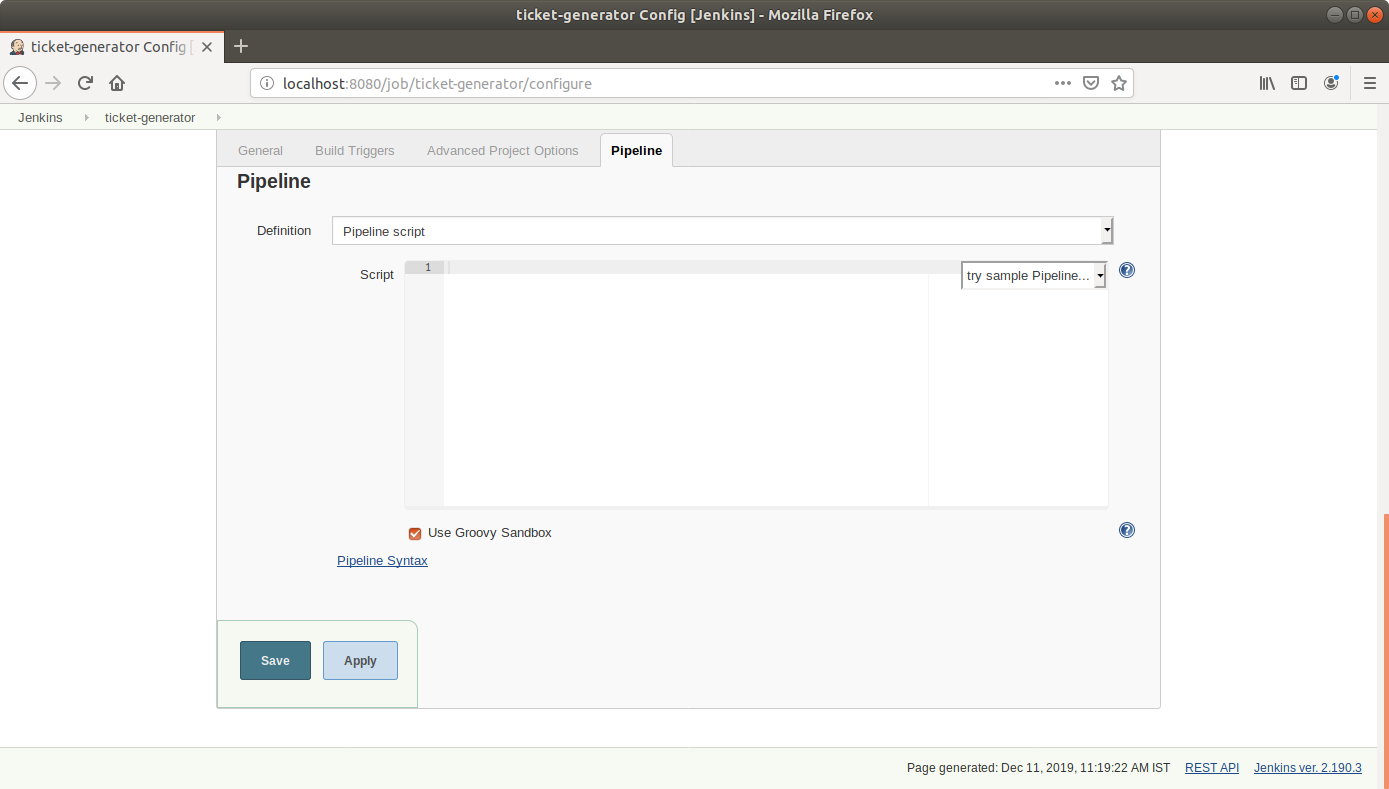
**Please Note:** In case you get an error while executing **git push -u origin master**, as shown in the screenshot below, execute **git push -u origin main**

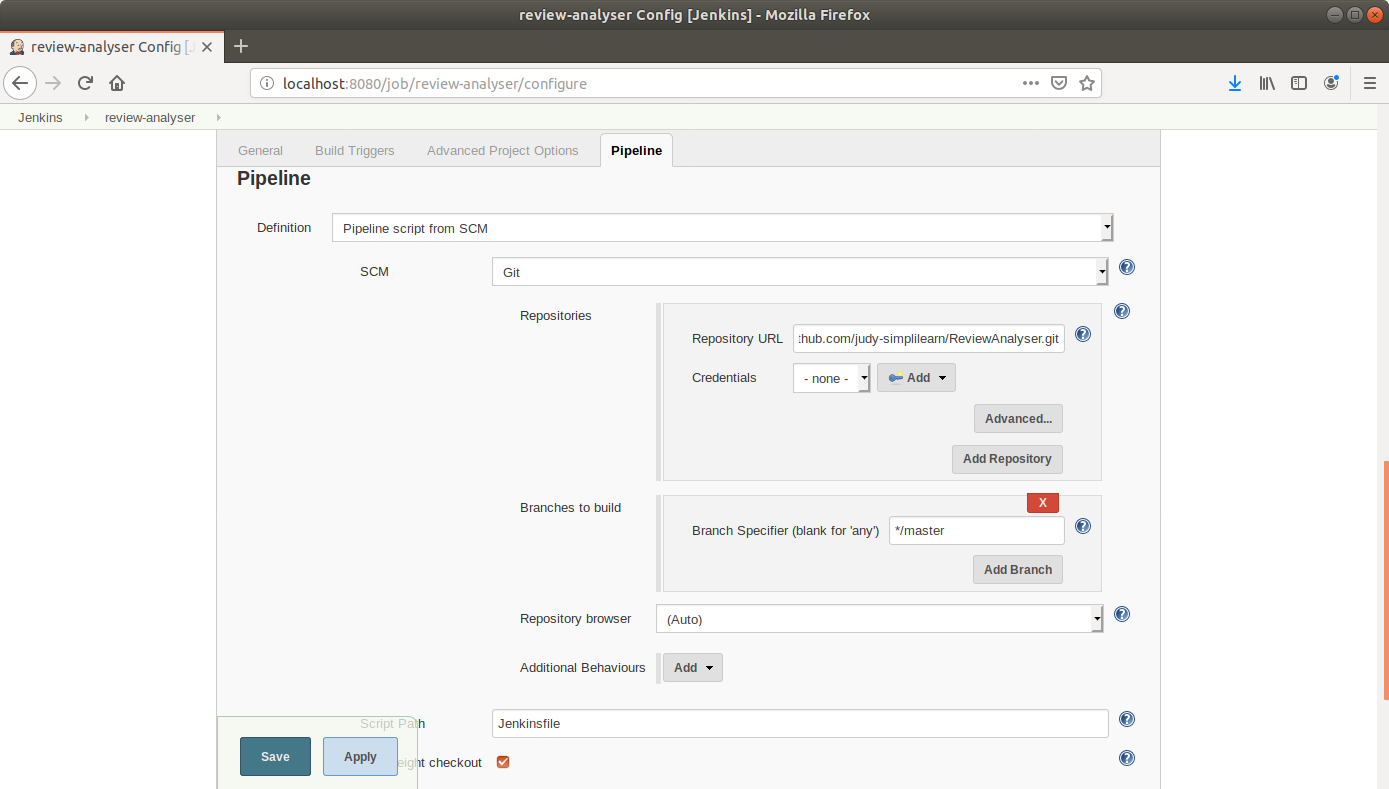


**Step 5:** Creating a multistage pipeline in Jenkins

* Go to Jenkins **dashboard**.
* Click on ***New Item***.
* Enter a **name** for your build job.
* Select ***Pipeline***as the build job type.



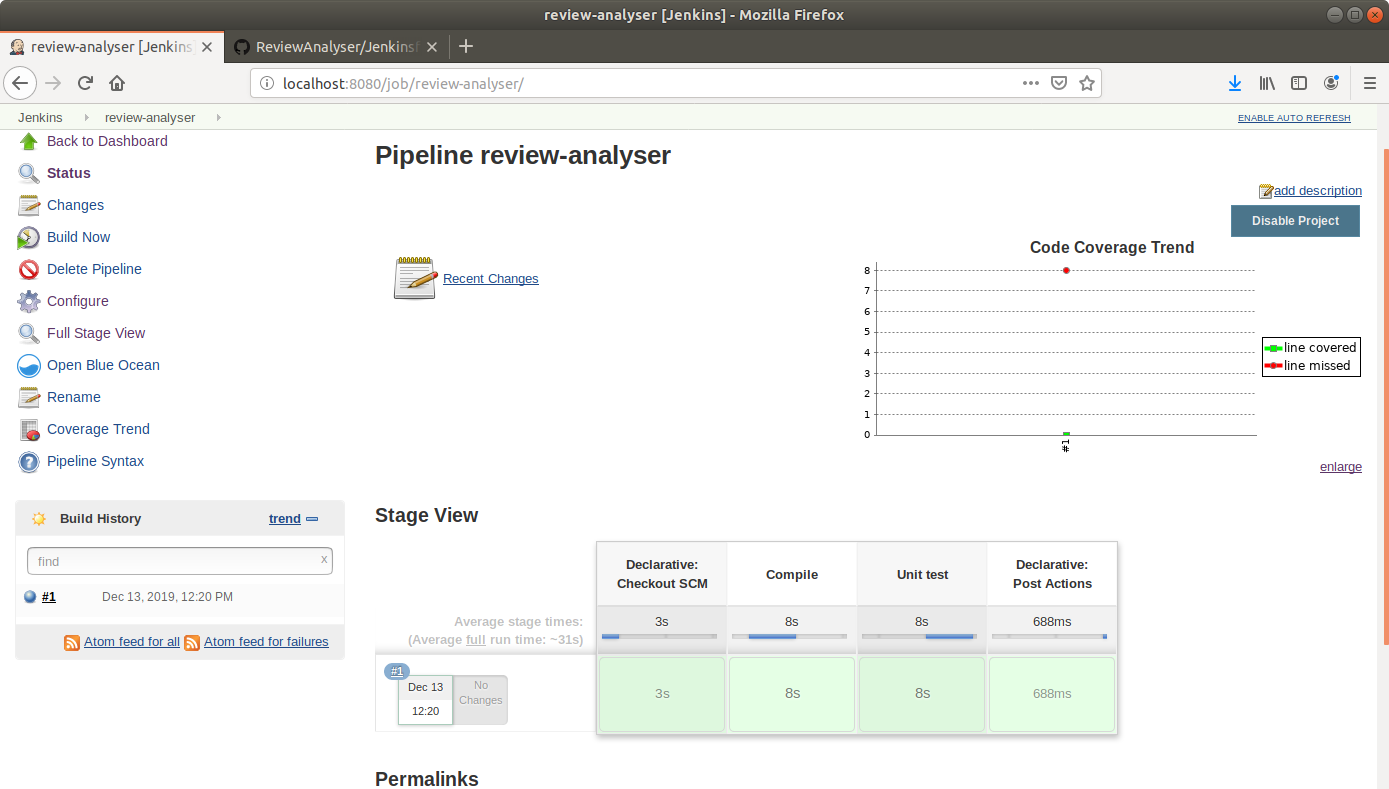
* Click **OK**.
* On the configuration page, scroll down to the **Pipeline** section.
* Change *Definition* from *Pipeline script* to ***Pipeline script from SCM****.*
* Select ***Git*** *in SCM.*
* Add the repository URL and the branch to build (master or main).



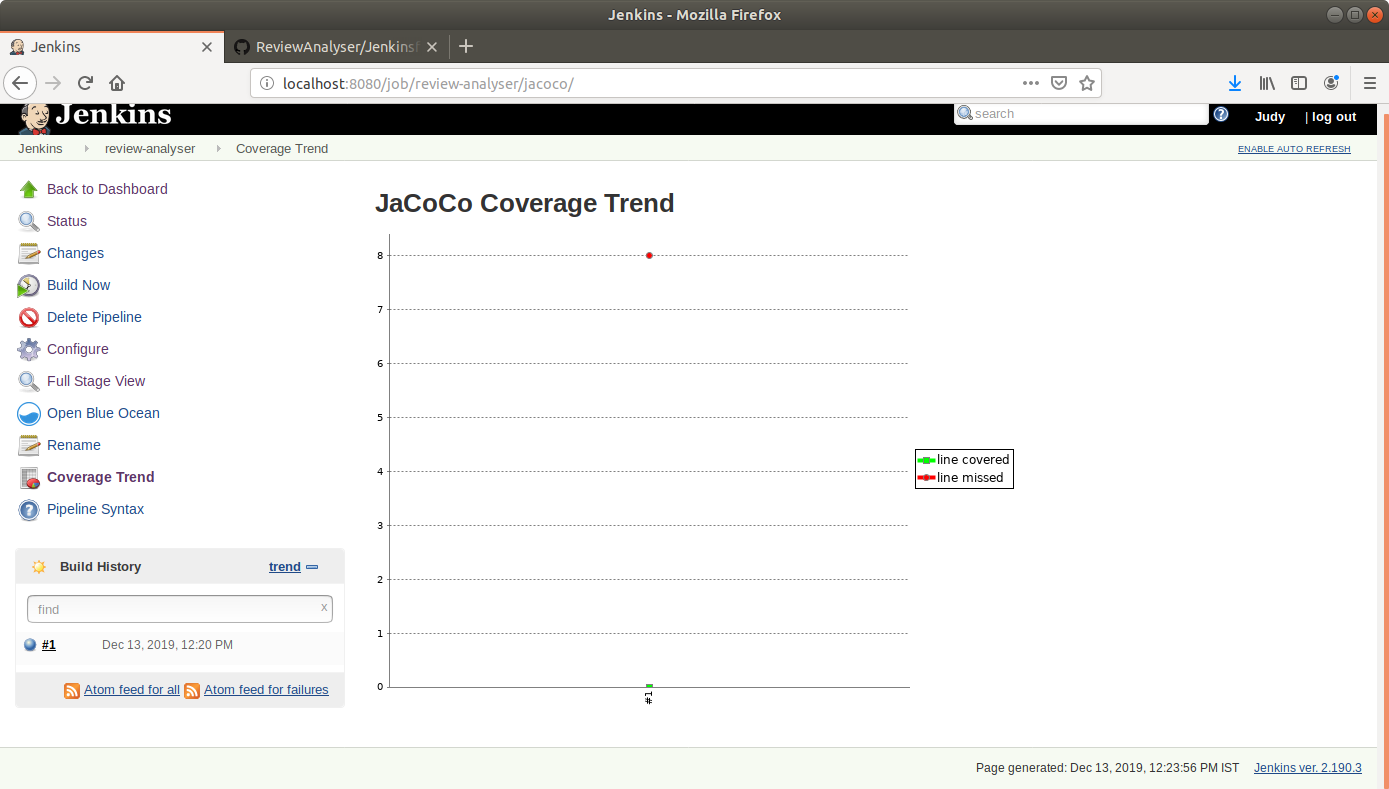
* Click **Save**.

**Step 6:** Running a multistage pipeline in Jenkins

* Click on ***Build Now***in the project window.
* Jenkins will now build your pipeline and output the logs.



* Click on *Coverage Trend* to view the coverage trend.

.