

Experiment 06

Aim:

Install and run Selenium automation tests in Jenkins using Maven.

Theory & Key Concepts

- **Jenkins:**

Jenkins is an open-source automation server that enables developers to implement Continuous Integration (CI) and Continuous Deployment (CD) pipelines for their projects. By automating repetitive tasks—such as builds, tests, and deployments—Jenkins helps deliver reliable and robust software. Integration with testing frameworks ensures every code change is automatically validated.

- **Maven:**

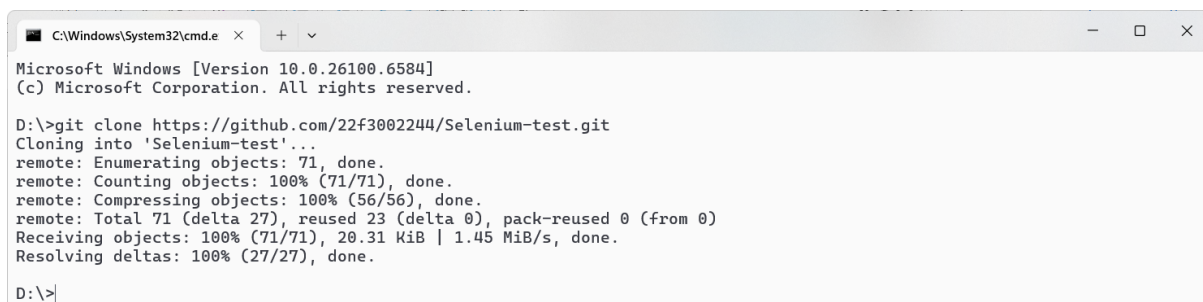
Maven is a build automation and dependency management tool for Java-based projects. It standardizes project configuration through its pom.xml file, where you define project details, dependencies (libraries required for your tests), plugins (such as Surefire and Failsafe for running unit and integration tests), and build goals. Maven makes it easy to manage dependencies and plugins and generate test reports automatically.

- **Selenium:**

Selenium lets you automate browser tasks and web application tests. Selenium scripts can be written in Java and organized as Maven projects, streamlining dependency management and portability. Maven brings in Selenium and relevant testing libraries through the pom.xml file.

Detailed Steps to Complete the Experiment**1. Create a Maven Project**

- Initialize a new Maven project on your machine or clone a starter repository.
 - Example: <https://github.com/HumayunK01/Selenium-test>

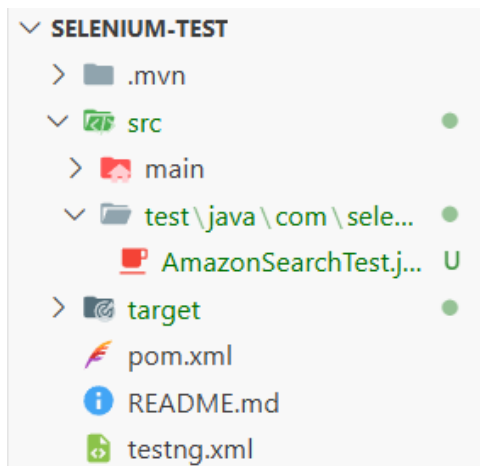


```
C:\Windows\System32\cmd.e  x  +  v
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\>git clone https://github.com/22f3002244/Selenium-test.git
Cloning into 'Selenium-test'...
remote: Enumerating objects: 71, done.
remote: Counting objects: 100% (71/71), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 71 (delta 27), reused 23 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (71/71), 20.31 KiB | 1.45 MiB/s, done.
Resolving deltas: 100% (27/27), done.

D:\>
```

- Directory structure will look like:



2. Write Your Selenium Automation Test Script

- Create a Java class for your Selenium test under src/test/java/com/selenium/.
- Example:

```
package com.selenium;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Reporter;
import org.testng.annotations.Test;

import java.time.Duration;
import java.util.List;

public class AmazonSearchTest {

    @Test
    public void searchAmazonProduct() {
        // Set Chromedriver path
        System.setProperty("webdriver.chrome.driver",
"C:\\Users\\Admin\\SeleniumDemo\\chromedriver-
win64\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();

        try {
            // Open Amazon India
            driver.get("https://www.amazon.in/");
```

```

        // Wait a bit for the page to load
        Thread.sleep(1000);

        // Locate the search box
        WebElement searchBox =
driver.findElement(By.id("twotabsearchtextbox"));

        // Type the search query and press ENTER
        searchBox.sendKeys("laptop" + Keys.ENTER);

        // Wait for results page to load and display results
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
        wait.until(ExpectedConditions.visibilityOfElementLoca
ted(By.cssSelector("div.s-main-slot")));

        // (Optional) Print titles of first few products
        List<WebElement> productTitles =
driver.findElements(By.cssSelector("span.a-size-medium.a-color-
base.a-text-normal"));
        for (int i = 0; i < Math.min(5,
productTitles.size()); i++) {
            System.out.println(productTitles.get(i).getText()
);
        }

    } catch (Exception e) {
        Reporter.log("Exception caught: " + e.getMessage());
    } finally {
        driver.quit();
    }
}
}

```

3. Configure Dependencies in pom.xml

- Add Selenium, TestNG/JUnit, and Maven plugins:

```

<dependencies>
  <!-- Selenium WebDriver -->
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>[Latest Version]</version>
  </dependency>
  <!-- TestNG -->
  <dependency>
    <groupId>org.testng</groupId>

```

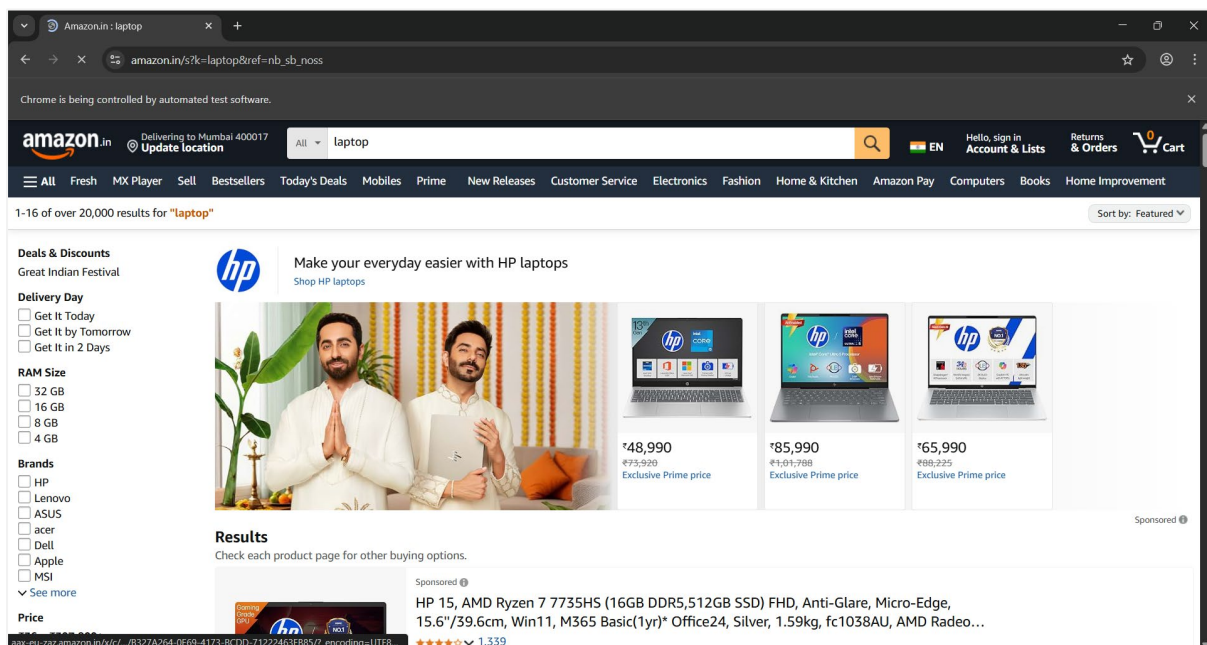
```

<artifactId>testng</artifactId>
<version>[Latest Version]</version>
</dependency>
</dependencies>
<build>
  <plugins>
    <!-- Surefire plugin for running unit tests -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>[Latest Version]</version>
    </plugin>
  </plugins>
</build>

```

4. Run Your Tests Locally

- Run test scripts in your IDE or use the terminal:



```

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 8.146 s -- in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.453 s
[INFO] Finished at: 2025-09-22T13:52:39+05:30
[INFO] -----
PS D:\Selenium-test>

```

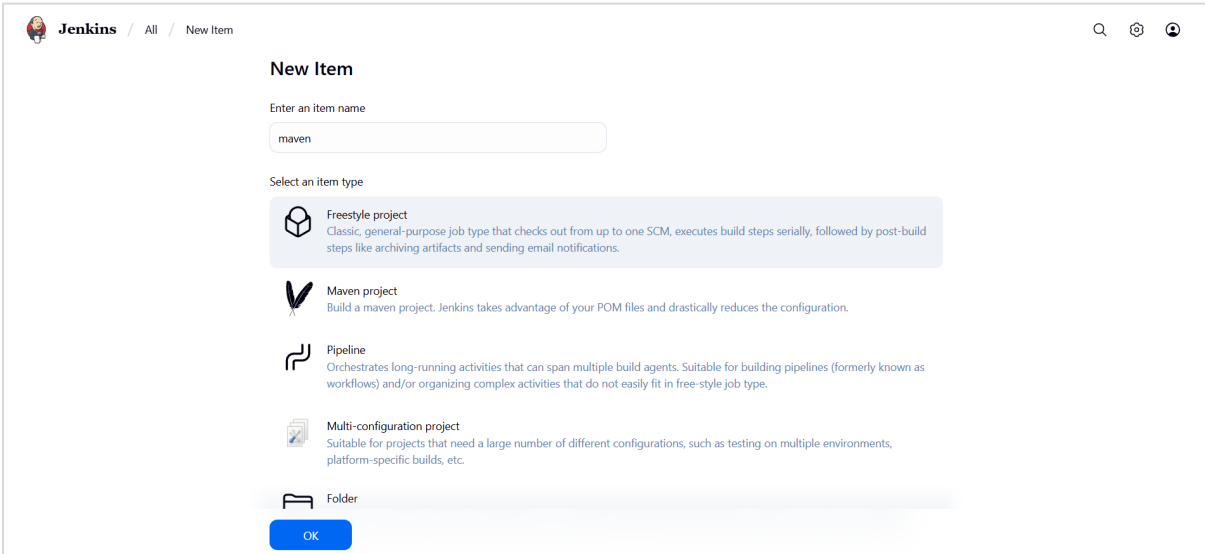
- Maven will execute all tests in the test directory and generate test reports.

5. Install and Start Jenkins

- Download and install Jenkins from jenkins.io.
- Start the Jenkins server on your machine.

6. Create and Configure a Jenkins Project

- Open Jenkins dashboard → Click “New Item” → Select *Freestyle Project*, give it a name.



Jenkins / All / New Item

New Item

Enter an item name

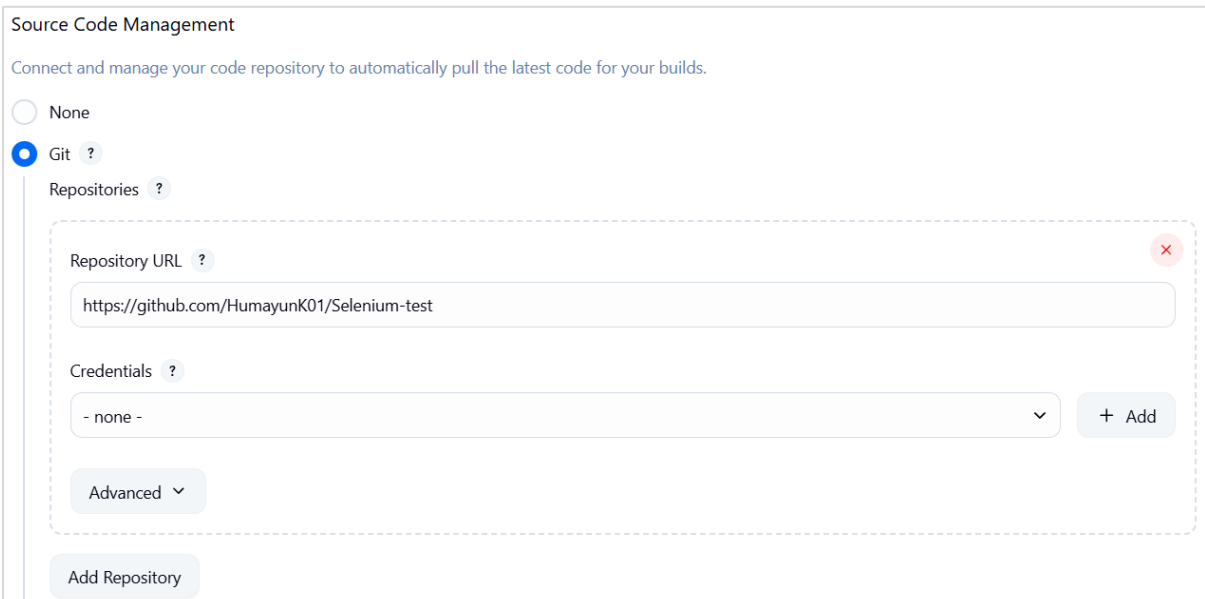
maven

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.
- Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- 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.
- Folder**

OK

- In project configuration:
 - **Source Code Management (SCM):**
 - Connect your project repository (e.g., Git).
 - Enter repository URL and change Branch Specifier to `*/fix-selenium-test`



Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://github.com/HumayunK01/Selenium-test

Credentials ?

- none -

Advanced

Add Repository

- **Build Steps:**

- Add a build step: “Invoke top-level Maven targets”
- Goals: clean test -DTest=DemoTest

Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

≡ **Invoke top-level Maven targets** ?

Maven Version

(Default) ▼

Goals

clean test -DTest=DemoTest ▼

Advanced ▼

Add build step ▼

- **Post-Build Actions:**

- Add “Publish JUnit test result report”
- Path: target/surefire-reports/*.xml
- Optionally, configure email notifications/reports.

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

≡ **Publish JUnit test result report** ?

Test report XMLs

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.

target/surefire-reports/*.xml

Test output retention ?

None ▼

☐ keep all the properties

☐ Keep original test names, even when reporting from multiple stages

Health report amplification factor ?

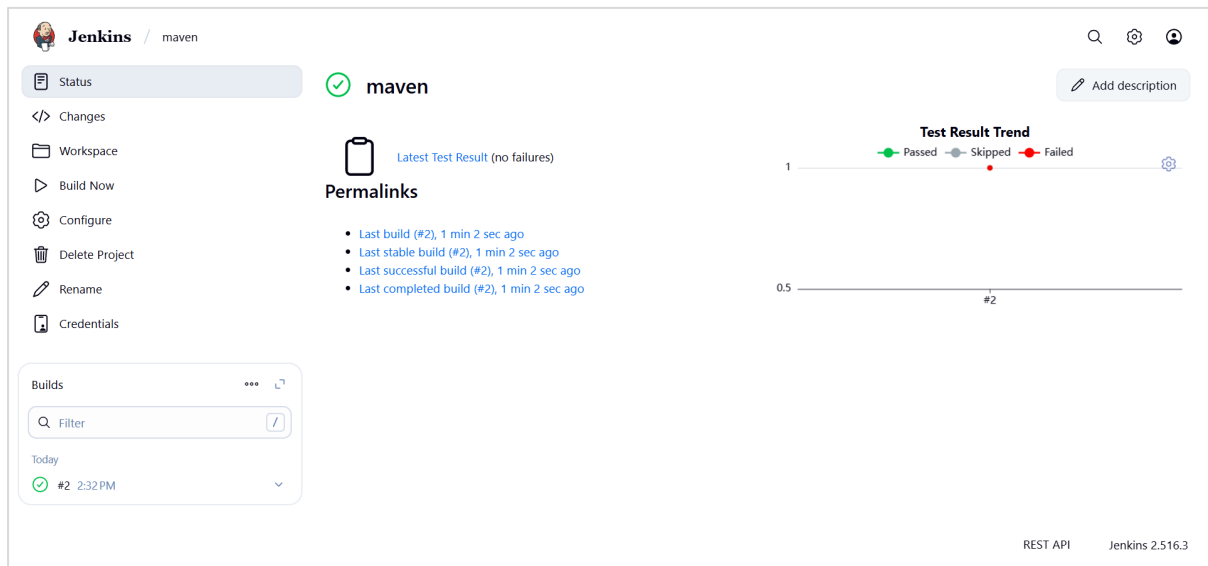
1.0

7. Apply and Save Configuration

- Review all settings and click “Save”.

8. Run Jenkins Build

- On your Jenkins job page, click “Build Now”.
- Monitor console output for compilation, test execution, and result summary.
- Check the build history, test reports, and logs for detailed feedback.



Conclusion

By integrating Selenium, Maven, and Jenkins:

- **Continuous Testing** becomes automated: tests run on every commit.
- **Faster Feedback:** Developers are instantly notified of failures.
- **Stable Builds:** Only code passing tests moves to the next stage.
- **Scalability:** You can add test environments (agents/nodes), more test cases, and complex workflows.