

Pre-requisites and software requirements

Pre-requisites

Java programming Knowledge: Basic understanding of Java syntax, oops concept and familiarity with building Java applications.

Operating system: Windows, macOS or Linux (64-bit recommended).

Internet connectivity: Reliable Internet connection for downloading software, accessing GitHub and Azure Resources.

GitHub account: Everyone should have a GitHub account for version control and source code management (SCM).

Software requirements

Sl. No.	Software/tool	Version	Purpose	Installation notes
1	Java JDK.	17.0 / above (preferred)	Required for building and running Java based applications	Download from Oracle or open JDK distributions.
2	IntelliJ IDEA	2022.2 / above	Integrated development environment, IDE for Java development and project management.	Download from JetBrains
3	Eclipse for developers	2022-09 / above	Integrated development environment, IDE for Java development and project management.	Download from Eclipse Packages
4	Apache Tomcat server	10 / above	Local server for basic website deployment	Download from Apache Tomcat

5	GIT	Latest stable version	Version control system for managing source code and collaborating via Github	Download from GIT
6	Maven	Latest stable version	Build automation tool for Java projects	Download from Apache Maven
7	Gradle	Latest stable version	Alternative build automation tool for managing dependencies and tasks	Download from Gradle
8	Jenkins	Latest stable version	CI-CD tool for automating builds, tests and deployments	Download from Jenkins
9	Oracle VM virtual box.	Latest stable version	Virtualization tool for creating and managing virtual machines	Download from Virtualbox
10	Vagrant	Latest stable version	Tool for managing and provisioning virtual machine environments.	Download from vagrant
11	Ansible	Latest stable version	Configuration management and automation tool	Install via ansible documentation
12	Microsoft Azure	Free Tier account	Needed for DevOps CI-CD pipelines and deployments	Install via Microsoft Azure sign up

Lab Experiment 1

Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup

What is DevOps?

DevOps is a set of cultural philosophies, practices, and tools that combine software development (Dev) and IT operations (Ops). Its goal is to shorten the system's development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.

DevOps promotes:

1. **Collaboration:** Breaking down silos between development and operations teams.
2. **Automation:** Automating repetitive tasks (builds, tests, deployments) to improve efficiency.

3. **Continuous Integration/Delivery (CI/CD):** Regularly integrating code changes and deploying them to production with minimal manual intervention.
4. **Monitoring and Feedback:** Constantly monitoring the performance and behavior of applications in production to rapidly address issues.

DevOps Used for its -

1. **Speed and Agility:** Faster development cycles and quicker time-to-market.
2. **Quality:** Automated testing and integration help catch issues early.
3. **Reliability:** Frequent, smaller updates reduce the risk of large-scale failures.
4. **Efficiency:** Automation reduces manual errors and repetitive tasks.
5. **Scalability:** Processes and infrastructure can grow with the business.

Introduction to Maven and Gradle

- Automatically manage dependencies – No need to download JARs manually.
- Simplify project configuration – A single configuration file (pom.xml for Maven, build.gradle.kts for Gradle) handles everything.
- Enable easy build & testing – Run tests and package applications using simple commands.
- Ensure consistency – The same project setup works on different machines.

Note: Both Maven and Gradle are used to automate the build process and manage project dependencies. Here's why they are so popular in the DevOps ecosystem:

- **Automated Build and Testing:** They allow developers to compile code, run tests, and package applications without manual intervention.
- **Consistent Build Environment:** By enforcing standardized project structures and dependency management, they help avoid the “it works on my machine” problem.

- Ease of Integration: Both tools integrate well with Continuous Integration (CI) servers like Jenkins, enabling automated pipelines.
- Dependency Resolution: They simplify the process of managing external libraries and ensure that all developers are using the same versions.

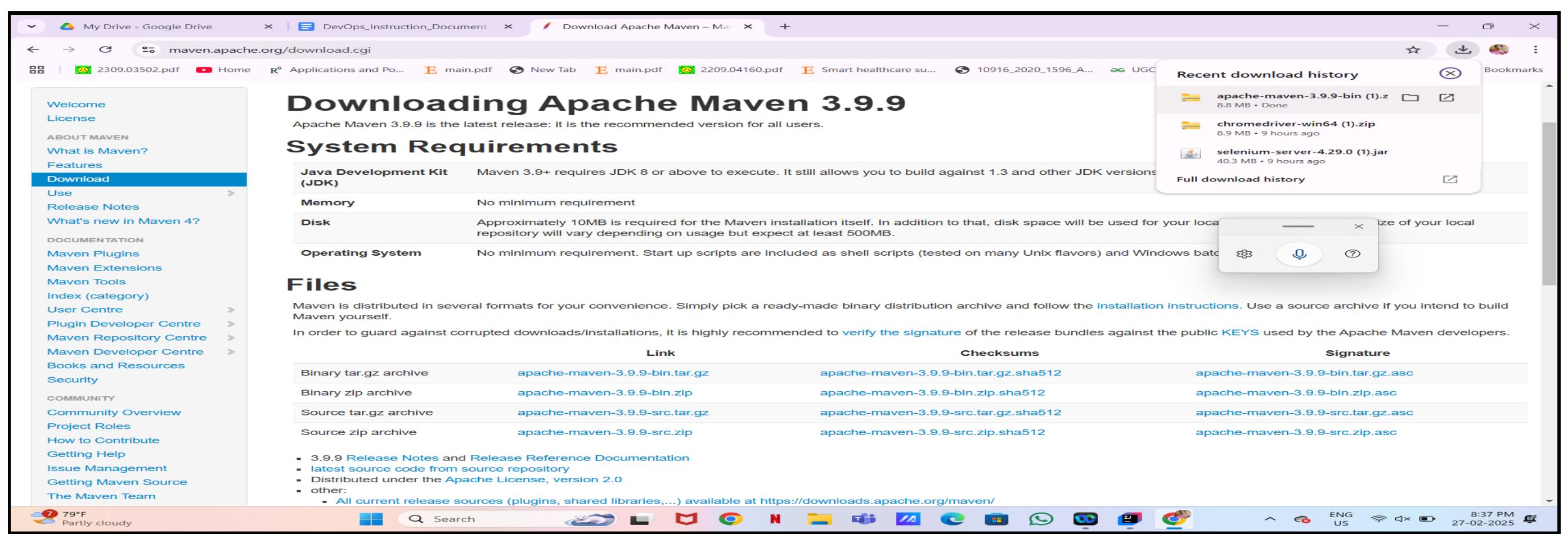
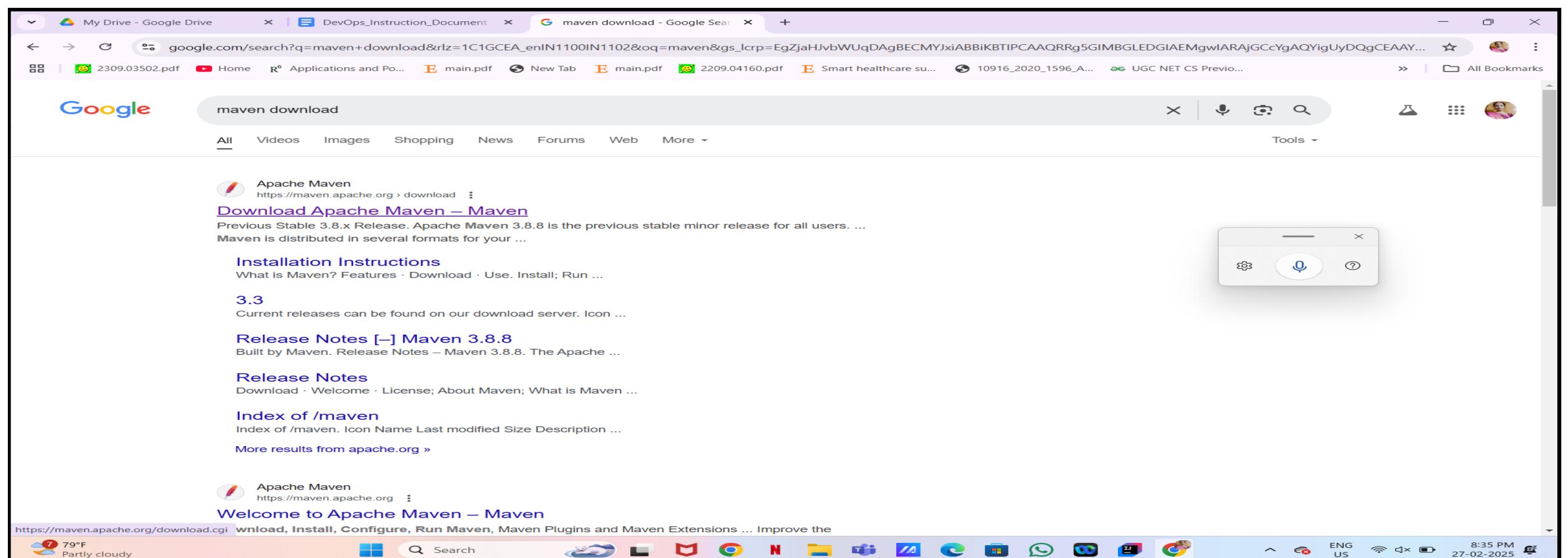
Differences between Maven and Gradle

Aspect	Maven	Gradle
Build Script Language	Uses an XML-based configuration file (<code>pom.xml</code>).	Uses a DSL based on Groovy or Kotlin (<code>build.gradle</code> or <code>build.gradle.kts</code>).
Configuration Style	Declarative and rigid – follows strict conventions (convention over configuration).	Flexible and dynamic – allows you to write custom logic and conditions within the build script.
Build Lifecycle	Provides a fixed lifecycle (e.g., validate, compile, test, package, install, deploy).	Uses a task-based approach where tasks can be defined, customized, and linked in a flexible manner.
Dependency Management	Manages dependencies through the POM file; downloads them from central repositories (e.g., Maven Central).	Similar dependency management; supports dynamic version resolution and customizable dependency configurations.
Performance	Generally slower for large projects because of the fixed build lifecycle and less	Often faster, thanks to incremental builds, caching, and parallel execution of tasks.

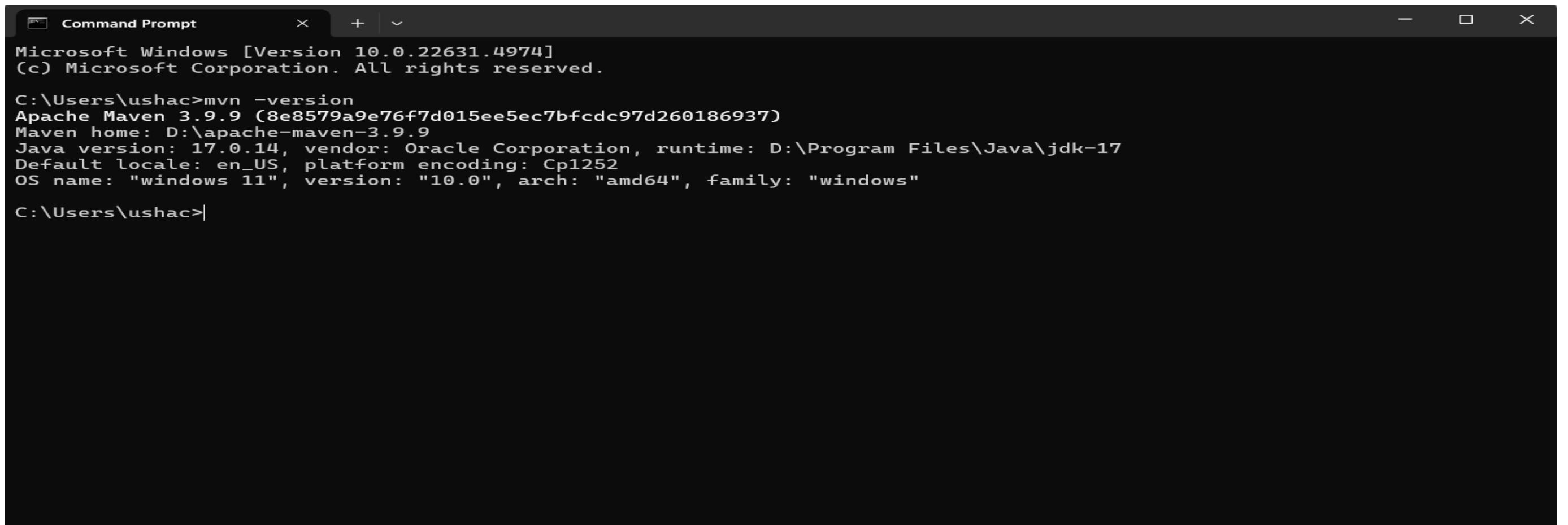
	emphasis on incremental builds.	
Extensibility & Plugins	Rich ecosystem of plugins but customization can be more challenging due to XML's verbosity and limitations in scripting.	Highly extensible through its scripting capabilities; writing custom tasks or plugins in Groovy/Kotlin is more straightforward.
Multi-Project Builds	Handles multi-module projects well but requires a strict directory layout and a parent POM for aggregating modules.	Excels in multi-project builds with simple configuration, allowing each subproject to be configured in a flexible manner.
Learning Curve	Easier for beginners due to its structured, convention-based approach but can become complex with large configurations.	More flexible but may have a steeper learning curve initially if you need to leverage its dynamic features and custom logic.
Community & Maturity	Has been around longer, so many legacy projects and extensive documentation exist.	Relatively newer; it has gained popularity due to its modern features, especially in Android and multi-language projects.
Integration with CI/CD	Well-supported by most CI/CD tools (like Jenkins, Azure Pipelines) with stable, predictable behavior.	Also integrates seamlessly with CI/CD tools and is often chosen for its faster build times and flexibility in pipeline scripting.

Installation of Maven

Browser → Maven download



After installing, check in cmd → mvn -version



```
Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ushac>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\apache-maven-3.9.9
Java version: 17.0.14, vendor: Oracle Corporation, runtime: D:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\ushac>
```

Installation of Gradle

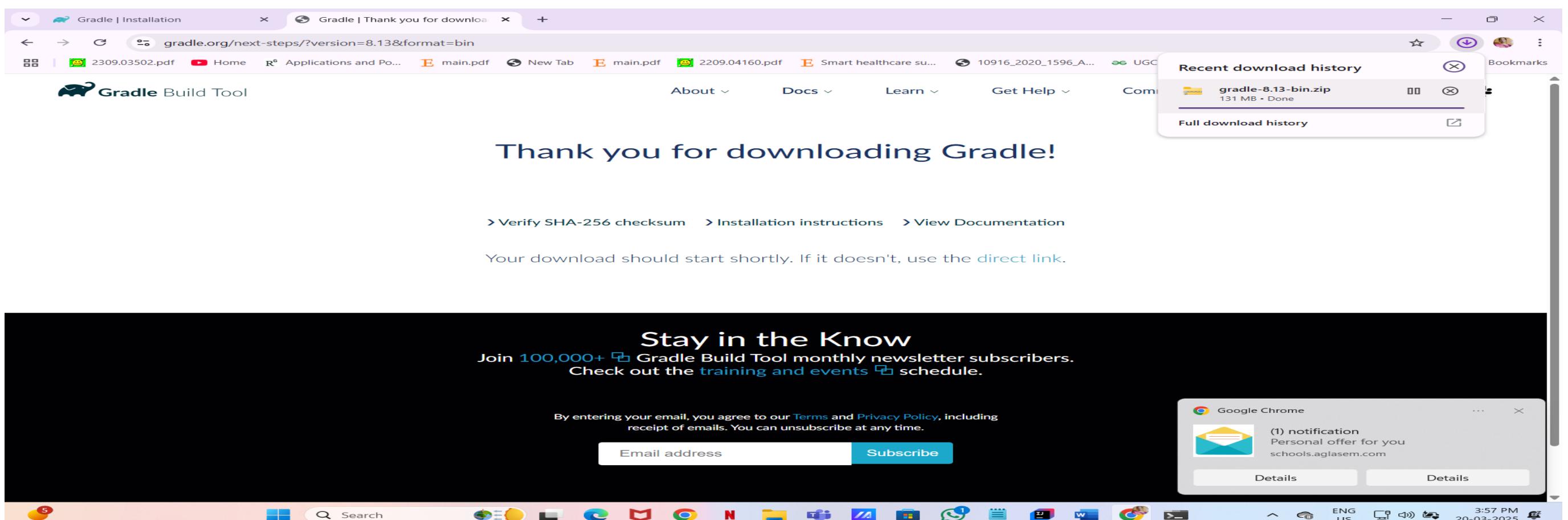
Step 1. [Download](#) the latest Gradle distribution

The current Gradle release is version 8.13, released on 25 Feb 2025. The distribution zip file comes in two flavors:

- [Binary-only](#)
- [Complete](#), with docs and sources

If in doubt, choose the binary-only version and browse [docs](#) and [sources](#) online.

Need to work with an older version? See the [releases page](#).



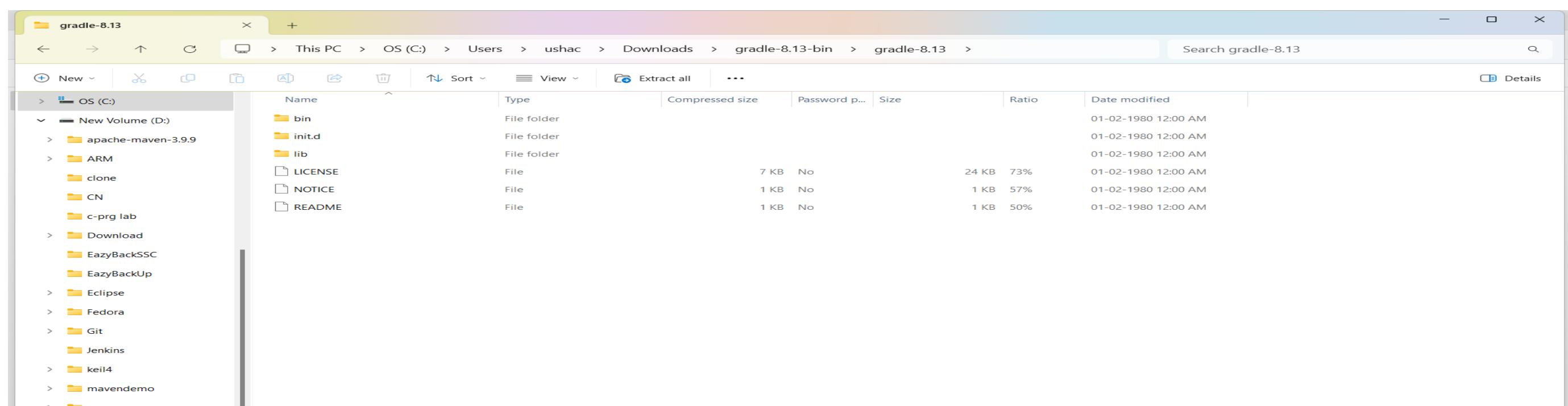
Step 2. Unpack the distribution

Microsoft Windows users

Create a new directory C:\Gradle with File Explorer.

Open a second File Explorer window and go to the directory where the Gradle distribution was downloaded. Double-click the ZIP archive to expose the content. Drag the content folder gradle-8.13 to your newly created C:\Gradle folder.

Alternatively you can unpack the Gradle distribution ZIP into C:\Gradle using an archiver tool of your choice.



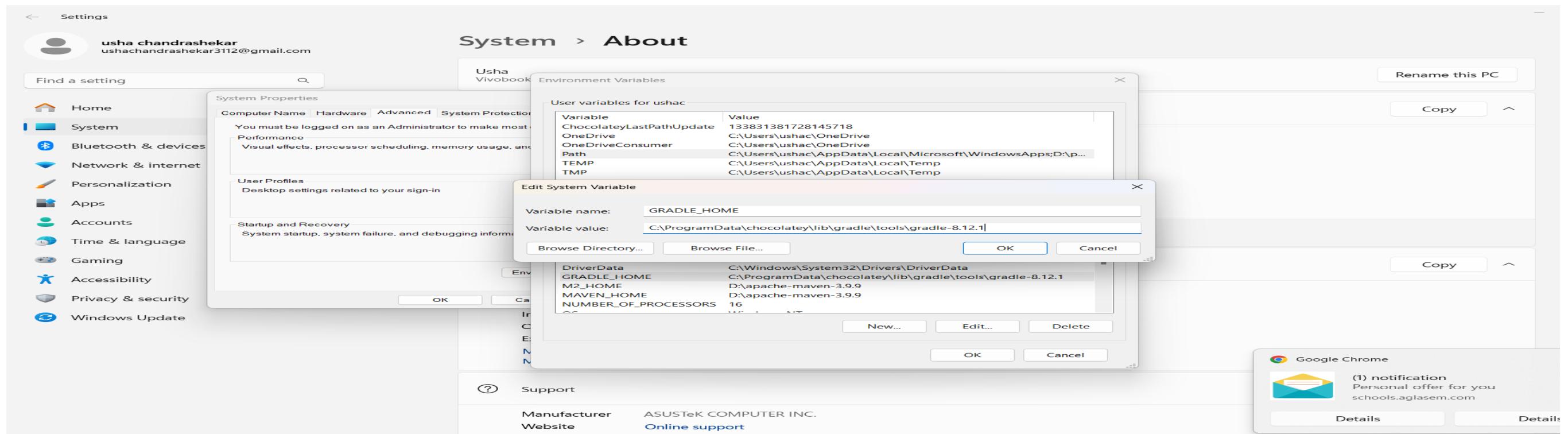
Step 3. Configure your system environment

Microsoft Windows users

In File Explorer right-click on the This PC (or Computer) icon, then click Properties -> Advanced System Settings -> Environmental Variables.

Under System Variables select Path, then click Edit. Add an entry for C:\Gradle\gradle-8.13\bin.

Click OK to save.



Step 4. Verify your installation

Open a console (or a Windows command prompt) and run gradle -v to run gradle and display the version.

```
Command Prompt
Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ushac>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\apache-maven-3.9.9
Java version: 17.0.14, vendor: Oracle Corporation, runtime: D:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\ushac>gradle -v
-----
Gradle 8.12.1
-----
Build time: 2025-01-24 12:55:12 UTC
Revision: 0b1ee1ff81d1f4a26574ff4a362ac9180852b140

Kotlin: 2.0.21
Groovy: 3.0.22
Ant: Apache Ant(TM) version 1.10.15 compiled on August 25 2024
Launcher JVM: 17.0.14 (Oracle Corporation 17.0.14+8-LTS-191)
Daemon JVM: D:\Program Files\Java\jdk-17 (no JDK specified, using current Java home)
OS: Windows 11 10.0 amd64
```

Lab Experiment 2

Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins.

Introduction to Maven

Maven is a powerful build automation and project management tool primarily used for Java projects.

It simplifies the build process by:

- Enforcing a standard project structure (convention over configuration).
- Managing dependencies automatically by downloading them from remote repositories (e.g., Maven Central).

- Defining a clear build lifecycle (compile, test, package, install, deploy).
- Allowing the integration of various plugins to extend functionality (e.g., testing, reporting).

Creating a Maven Project

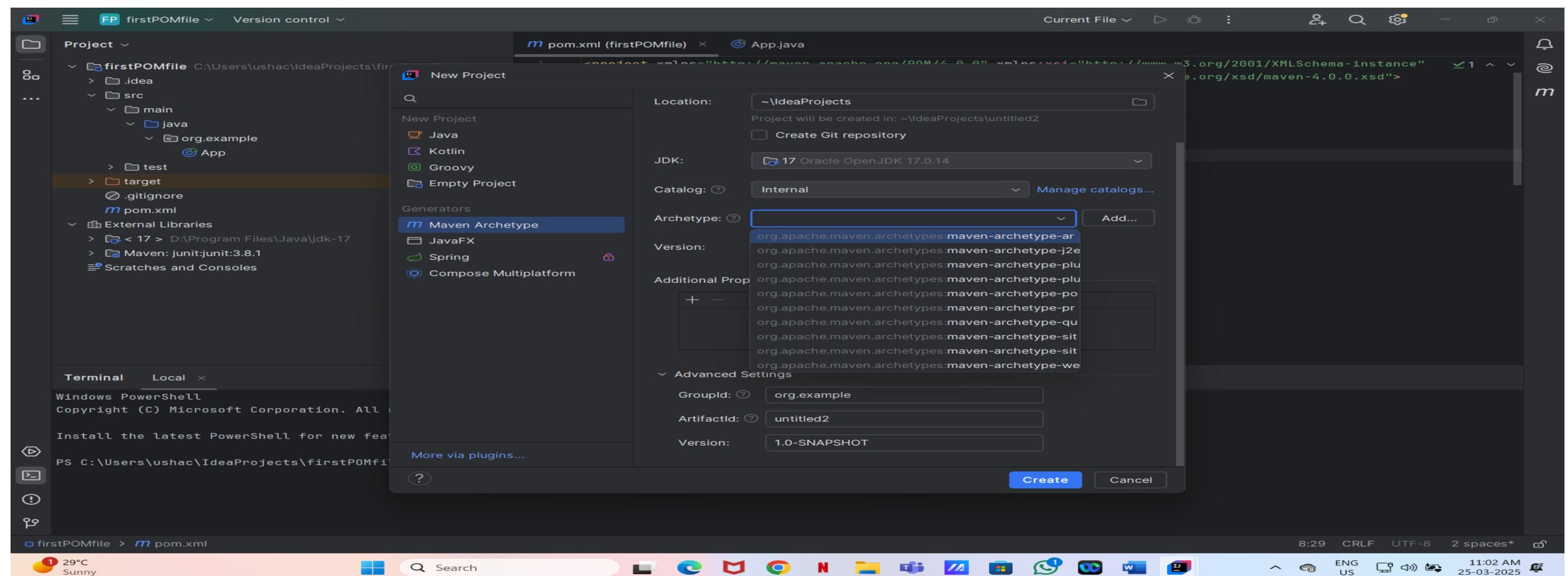
Step-by-Step Process

Step 1: Open Your IntelliJ Idea App, make sure you have Maven installed (refer to Experiment 1).

Step 2: Use Maven Archetype to Generate a New Project Maven comes with a set of archetypes that provide you with a standard project template.

File → new project → maven archetype → firstPOMfile → select maven-archetype-QuickStart → create

- **groupId:** Uniquely identifies your project's group (like a package name).
- **artifactId:** The name of your project (the resulting artifact).
- **maven-archetype-quickstart:** A simple archetype that sets up a basic Java project with a sample unit test.



Maven Project Layout and Components

After generating the project, you will notice the following standard Maven directory structure:

```

MyMavenApp/
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── example
    │               └── App.java
    └── test
        └── java
            └── com
                └── example
                    └── AppTest.java

```

Explanation of Key Components

- `pom.xml`: The Project Object Model (POM) file is the core of any Maven project. It contains configuration details such as project coordinates (`groupId`, `artifactId`, `version`), dependencies, plugins, and build settings.
- `src/main/java`: This directory holds the source code of your application. In our example, the package structure `com.example` is created, and you have an `App.java` file.
- `src/test/java`: This directory is for your test cases. The default example includes a basic test class, `AppTest.java`.

Understanding the POM File (`pom.xml`)

The `pom.xml` file is written in XML and is essential to how Maven operates. It includes several key sections:

Basic Structure of a POM File

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

```

```

<! – Project Coordinates -->
<groupId>com.example</groupId>
<artifactId>MyMavenApp</artifactId>
<version>1.0-SNAPSHOT</version>

```

```
<!-- Properties: Customize Java version or plugin versions -->
<properties>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
</properties>

<!-- Dependencies -->
<dependencies>
<!-- Example: JUnit dependency for testing -->
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.13.2</version>
<scope>test</scope>
</dependency>
</dependencies>

<!-- Build: Configuring plugins and build settings -->
<build>
<plugins>
<!-- Example: Maven Compiler Plugin to compile Java code -->
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.1</version>
<configuration>
<source>1.6</source>
<target>1.6</target>
</configuration>
</plugin>
<!-- Example: Maven Surefire Plugin to run tests -->
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.22.2</version>
</plugin>
</plugins>
</build>
</project>
```

Key Elements Explained

- **Project Coordinates:**

<groupId>: Acts like a namespace, usually following the reverse domain name convention.

<artifactId>: The name of the project.

<version>: The current version of the project.

- **Properties:**

Used to define values that can be referenced elsewhere in the POM (e.g., Java source and target versions).

- **Dependencies:**

Dependency Management: Maven downloads and manages external libraries. In the example, JUnit is added for testing.

Scope: Determines when a dependency is used (e.g., compile, test, runtime).

- **Build and Plugins:**

Maven Compiler Plugin: Ensures that your Java code is compiled with the specified Java version.

Maven Surefire Plugin: Executes unit tests during the build process.

Dependency Management with Maven

How Maven Manages Dependencies?

- **Automatic Download**: When you specify a dependency in the section, Maven automatically downloads the library from a remote repository (usually Maven Central).
- **Transitive Dependencies**: Maven also resolves dependencies required by your dependencies.
- **Version Control**: You can specify precise versions for each dependency, ensuring consistency across builds.

Example Dependency To add a dependency for JUnit, include the following snippet in your **<dependencies>** section:

```

<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>

```

- **groupId, artifactId, version**: These three elements uniquely identify the dependency.
- **scope**: The test scope ensures that this dependency is only available during the test

phase and not included in the final artifact.

Maven Plugins: Extending Maven Functionality

Plugins are key to Maven's flexibility, adding tasks to your build process.

Example : Maven Compiler Plugin

This plugin is used to compile your Java source code.

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.1</version>
<configuration>
<source>11</source>
<target>11</target>
</configuration>
</plugin>
```

- Configuration: Specifies that the project should be compiled using Java 11.

Building and Testing Your Maven Project

Once your project is set up and your pom.xml is defined, you can use Maven commands to build and test your application.

Common Maven Commands

- Compile the Project:

```
mvn compile
```

- Run Unit Tests:

```
mvn test
```

- Package the Application:

```
mvn package
```

This command compiles, tests, and packages your code into a JAR file located in the target directory.

- Clean the Project:

```
mvn clean
```

This removes any files generated by previous builds.

Source code used in our Maven-Project

Pom.xml(firstPOMfile)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>firstPOMfile</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>firstPOMfile</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <!-- Compiler Plugin -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
```

```
<configuration>
  <source>1.8</source>
  <target>1.8</target>
</configuration>
</plugin>

<!-- Jar Plugin -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.2.0</version>
  <configuration>
    <archive>
      <manifest>
        <mainClass>org.example.App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>

</plugins>

</build>
</project>
```

In Terminal execute these commands

```
mvn clean compile
mvn package
java -jar .\target\firstPOMfile-1.0-SNAPSHOT.jar
```

Output

Hello world!

The screenshot shows the IntelliJ IDEA interface with a Maven project named 'firstPOMfile'. The project structure on the left includes .idea, src (with main and test), target (containing classes, generated-sources, generated-test-sources, maven-archiver, maven-status, surefire-reports, test-classes, and a JAR file named firstPOMfile-1.0-SNAPSHOT.jar), .gitignore, pom.xml, and External Libraries. The right pane displays the contents of pom.xml, which defines a Maven project with group ID org.example, artifact ID firstPOMfile, version 1.0-SNAPSHOT, packaging jar, name firstPOMfile, URL http://maven.apache.org, properties (sourceEncoding UTF-8), and a dependency on junit 3.8.1 for test scope. Below the XML editor is a terminal window showing the output of a 'jar' command to build the JAR, followed by a 'java -jar' command to run it, resulting in the output 'Hello World!'. The status bar at the bottom indicates the time as 14:48, CRLF encoding, UTF-8 encoding, and two spaces indentation.

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>firstPOMfile</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>firstPOMfile</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

Lab Experiment 3

Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation.

Introduction to Gradle

Gradle is a modern build automation tool designed to be highly flexible, fast, and scalable. It is widely used in Java projects, Android development, and many multi-language projects.

Here's what makes Gradle stand out:

- **Flexible Build Scripts:** Gradle uses a Domain Specific Language (DSL) based on

either Groovy (by default) or Kotlin. This provides a more dynamic and expressive way to define build logic compared to static XML configurations (as used in Maven).

- Incremental Builds: Gradle optimizes build times by determining what parts of the project have changed and rebuilding only those parts.
- Task Automation: Everything in Gradle is treated as a task, allowing you to create custom tasks or reuse existing ones for compiling code, running tests, packaging, and more.
- Dependency Management: Like Maven, Gradle can automatically download and manage dependencies from remote repositories (e.g., Maven Central, JCenter, or custom repositories).

Setting Up a Gradle Project

Step-by-Step Instructions

Step 1: Ensure Gradle is Installed

Before starting, verify that Gradle is installed on your system. Open a terminal and run:

```
gradle -v
```

If you see version information, you're set. If not, follow the Gradle installation instructions from Experiment 1.

Gradle Project using Kotlin

File → New → project → select Gradle → choose → Kotlin DSL (it will generate build.gradle.kts) → Name the project → set JDK 17 → FINISH

build.gradle.kts

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile
plugins {
    kotlin("jvm") version "1.8.10" // Use latest stable Kotlin version
    application
}
group = "org.example"
version = "1.0-SNAPSHOT"
repositories {
    mavenCentral()
}

dependencies {
    implementation(kotlin("stdlib")) // Kotlin Standard Library
    testImplementation("org.junit.jupiter:junit-jupiter-api:5.8.2")
```

```

    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:5.8.2")
}

tasks.test {
    useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "17" // Match with your JDK version
}

tasks.register<Jar>("fatJar") {
    archiveClassifier.set("all")
    duplicatesStrategy = DuplicatesStrategy.EXCLUDE
    manifest {
        attributes["Main-Class"] = "MainKt"
    }
    from(configurations.runtimeClasspath.get().map { if (it.isDirectory) it else zipTree(it) })
    with(tasks.jar.get() as CopySpec)
}

tasks.register("hello") {
    doLast {
        println("Hello, Usha!")
    }
}

application {
    mainClass.set("MainKt") // Update this if using a package
}

```

Main.kt

```

fun main() {
    println("Hello, Gradle with Kotlin DSL")

}

```

Execute these commands in terminal

```

./gradlew build
./gradlew run

```

Output 1 -

> Task :run

Hello, Gradle with Kotlin DSL

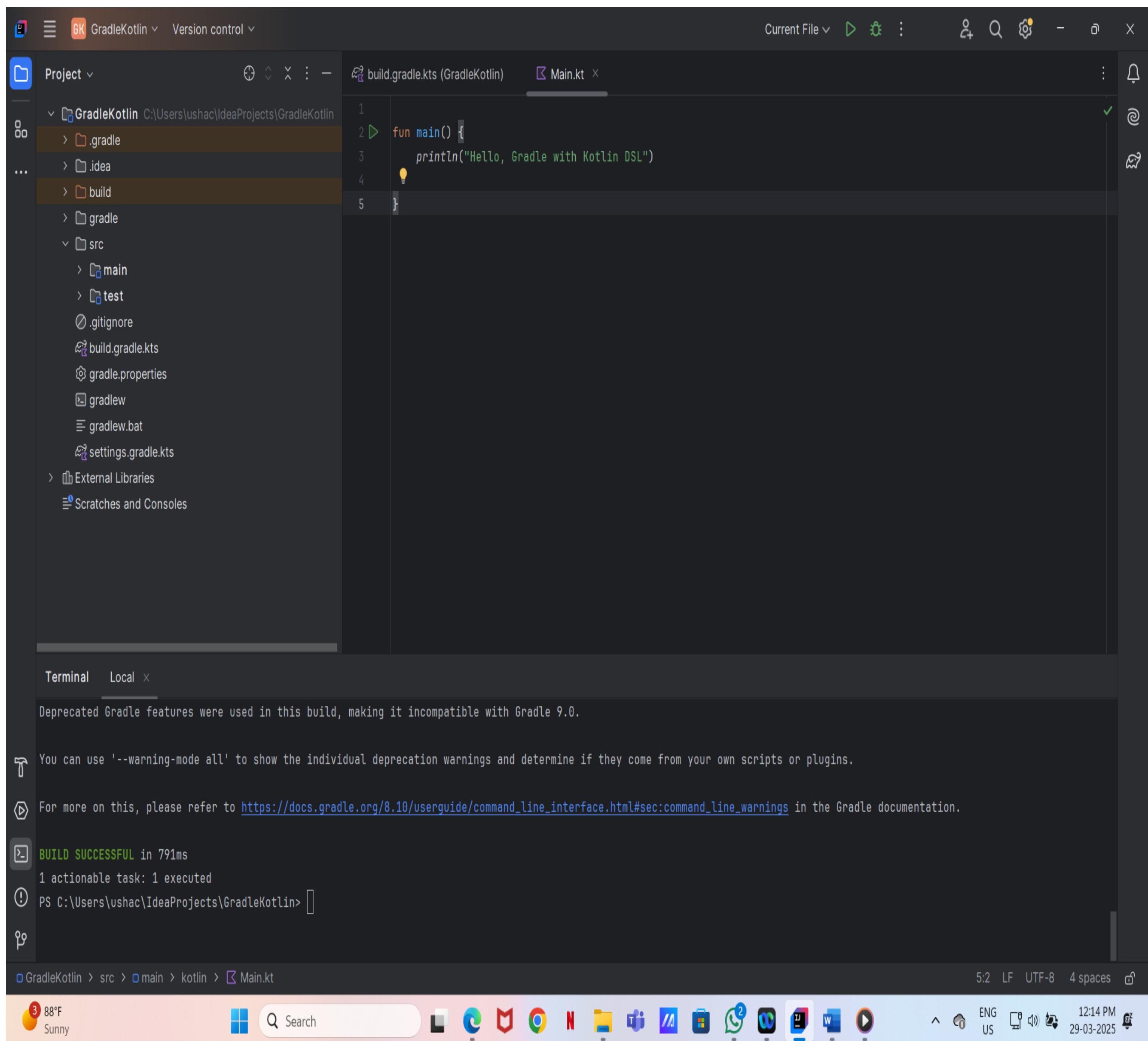
Output 2 -

```
./gradlew fatJar  
java -jar .\build\libs\GradleKotlin-1.0-SNAPSHOT.jar  
./gradlew hello
```

Output 3 -

> Task :hello

Hello, Usha!



Explanation of Components:

- **build.gradle:** This is the main build script written in Groovy (or Kotlin if you choose). It defines plugins, repositories, dependencies, and tasks.
- **settings.gradle:** A small script that defines the project's name and, in multi-project builds, the included subprojects.

- `gradlew` / `gradlew.bat`: The Gradle wrapper scripts. They allow you to run Gradle without requiring a separate installation on every machine by automatically downloading the correct Gradle version.
- `src/main/java`: Contains your application's source code.
- `src/test/java`: Contains your unit tests.

Understanding Gradle Build Scripts

A build script in Gradle is a programmatic file that instructs Gradle on how to build your project.

It can be written in two main DSLs:

- Groovy DSL (`build.gradle`): The traditional and most common syntax.
 - Kotlin DSL (`build.gradle.kts`): A statically-typed alternative that leverages Kotlin's language features.
 - Groovy DSL Example (`build.gradle`) Below is an example of a basic `build.gradle` for a Java application using Groovy DSL:

```
plugins {  
    // Apply the Java plugin for compiling Java code  
    id 'java'  
  
    // Apply the application plugin to add support for building an  
    // application  
    id 'application'  
  
}  
  
group = 'com.example'  
version = '1.0'  
  
repositories {  
    // Use Maven Central for resolving dependencies.  
    mavenCentral()  
}
```

```

dependencies {

    // Define your dependencies. For example, JUnit for testing:

    testImplementation 'junit:junit:4.13.2'

}

application {

    // Define the main class for the application.

    mainClass = 'com.example.App'

}

Department of Computer Science and Engineering
prakasha.s@rnsit.ac.in

Devops lab manual

// A custom task example: printing a greeting

task hello {

    doLast {

        println 'Hello, Gradle!'

    }

}

```

Explanation:

- **plugins block:** Declares the plugins used. The `java` plugin adds Java compilation and testing tasks, while the `application` plugin adds tasks for running the application.
- **group/version:** Sets project coordinates.
- **repositories block:** Configures the repository (Maven Central) where dependencies will be resolved.
- **dependencies block:** Lists the libraries your project depends on.
- **application block:** Specifies the main class of your application.
- **Custom task:** Defines a task named `hello` that prints a greeting when run.

Kotlin DSL Example (build.gradle.kts)

Here's how the same build configuration might look in Kotlin DSL:

Create a file named build.gradle.kts (or convert your file) with the following content:

```
plugins {  
    // Apply the Java plugin for compiling Java code  
    java  
  
    // Apply the application plugin to add support for building an  
    application  
  
    application  
}  
  
group = "com.example"  
  
version = "1.0"  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    // Define dependencies using Kotlin DSL syntax  
    testImplementation("junit:junit:4.13.2")  
}  
  
application {  
    // Set the main class for the application  
    mainClass.set("com.example.App")  
}  
  
// A custom task example using Kotlin DSL
```

```
tasks.register("hello") {  
    doLast {  
        println("Hello, Gradle with Kotlin DSL!")  
    }  
}
```

Explanation:

- **Kotlin DSL Syntax:** Uses a statically typed syntax, which can be more intuitive for developers familiar with Kotlin.
- **Plugins, Repositories, and Dependencies:** Defined similarly to the Groovy DSL but with Kotlin-style function calls and property access.
- **Custom Task Registration:** Uses `tasks.register` to define a task, similar in functionality to the Groovy DSL.

Lab Experiment 4

Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

POM.XML

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>FinalMavenToGradle</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>FinalMavenToGradle</name>
  <url>http://maven.apache.org</url>
```

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <!-- Compiler Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <!-- Jar Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.2.0</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>com.example.App</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

Build.gradle

```
plugins {
    id 'java'
}

group = 'com.example'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {

    testImplementation 'junit:junit:4.13.2'
}

jar {
    manifest {
        attributes(
            'Main-Class': 'com.example.App'
        )
    }
}
```

App.java

```
package com.example;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello from Gradle!" );
    }
}
```

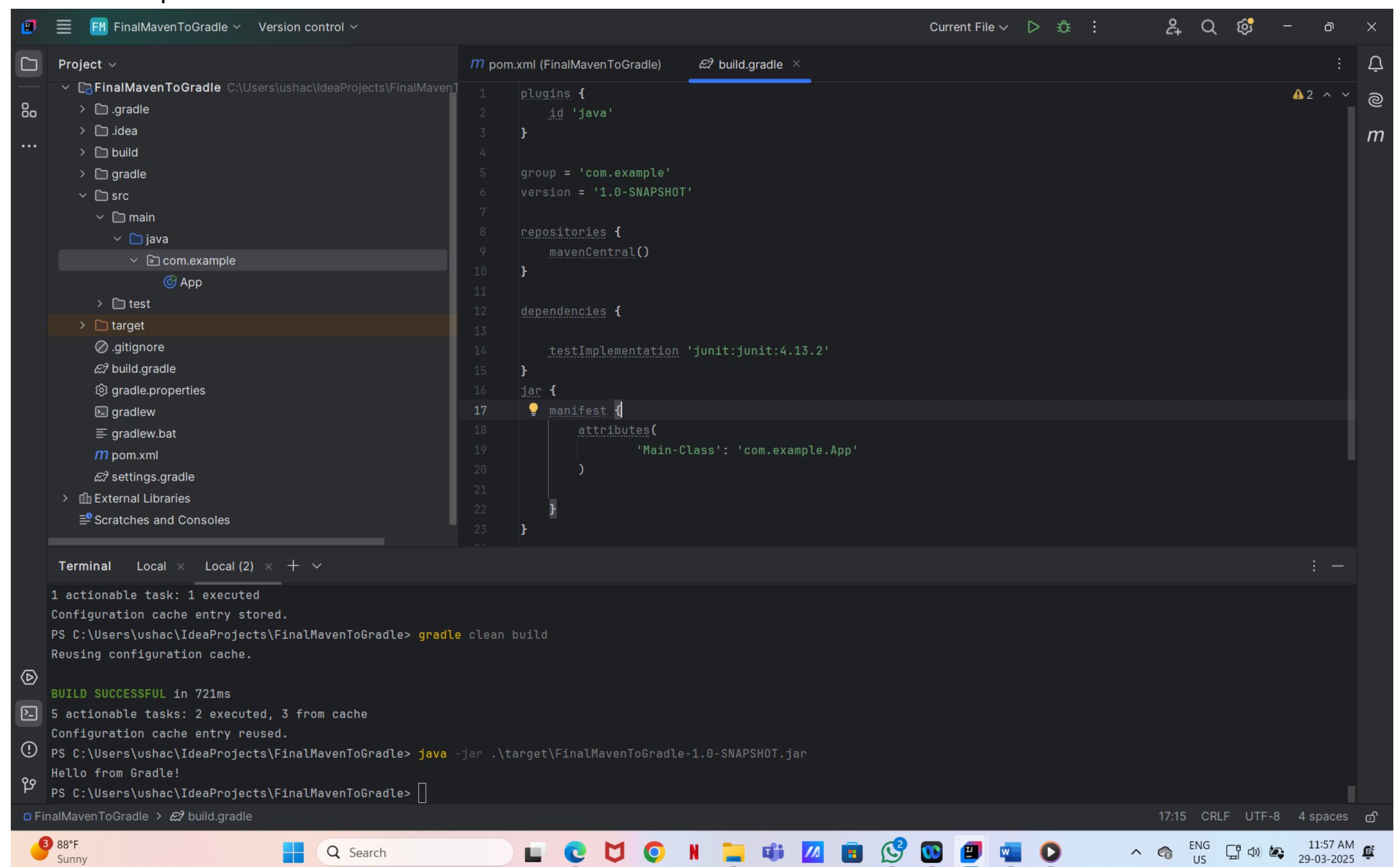
```

    }
}

```

In Terminal run these commands

1. mvn clean install
2. mvn package
3. java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar
Output- Hello from Gradle!
4. gradle init --type pom
5. Select build script DSL:
1: Kotlin
2: Groovy
Select 2
6. Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
Yes
7. gradle clean build
8. java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar
Output – Hello from Gradle!



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with a tree view of files and folders. The `build.gradle` file is selected.
- Code Editor:** Displays the `build.gradle` file content. The manifest section is highlighted with a yellow dot.
- Terminal:** Shows the command-line output of the build process. It includes:
 - Execution of `gradle clean build`.
 - Success message: `BUILD SUCCESSFUL in 721ms`.
 - Execution of `java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar`.
 - Output: `Hello from Gradle!`
- Status Bar:** Shows system information like weather (88°F, Sunny), date (29-03-2025), and time (11:57 AM).

Lab Experiment 5

Introduction to Jenkins: What is Jenkins? Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use.

1. Jenkins is an open-source automation server used for:

- Continuous Integration (CI)** – Automatically testing and integrating code changes
- Continuous Deployment (CD)** – Automating application deployment
- Building Pipelines** – Managing end-to-end software development workflows
- Plugin-Based Extensibility** – Supporting tools like Maven, Gradle, Ansible, Docker, and Azure DevOps

2. Why Use Jenkins?

- Automates builds and tests
- Reduces manual intervention
- Improves software quality
- Works with multiple tools and platforms

3. Installing Jenkins: Jenkins can be installed using multiple methods -

- 1** Windows Installer (.msi) – Recommended for Windows
- 2** Linux Package Manager – Best for Linux Users
- 3** Jenkins WAR File – Universal method using Java

Steps to install Jenkins in windows

Step 1: Download Jenkins

 Download from: [Download and deploy](#) Choose Windows Installer (.msi) for an easy setup.

Step 2: Install Jenkins

- 1** Run the downloaded .msi file.
- 2** Follow the installation wizard.
- 3** Select Run Jenkins as a Windows Service (recommended).
- 4** Choose the installation directory (default: C:\Program Files\Jenkins).
- 5** Click Install and wait for the setup to complete.

Step 3: Start Jenkins

- 1** Open Services (services.msc) and ensure Jenkins is running.

- 2** Open a web browser and go to: 1 <http://localhost:8080>

Step 4: Unlock Jenkins

1 Find the initial Admin Password in:

1. C:\Program Files\Jenkins\secrets\initialAdminPassword
2. Copy the password and paste it into the Jenkins setup page.

Step 5: Install Recommended Plugins Jenkins will prompt you to install plugins.

Click "Install Suggested Plugins".

Step 6: Create Admin User

- 1** Set up a Username, Password, and Email.
- 2** Click Save and Finish.

Jenkins is now ready! Access it anytime at: 1 <http://localhost:8080>

Configuring Jenkins for First Use

Understanding the Jenkins Dashboard After logging in, you will see:

- ◆ New Item → Create Jobs/Pipelines
- ◆ Manage Jenkins → Configure System, Users, and Plugins
- ◆ Build History → View previous builds
- ◆ Credentials → Store secure authentication details

Installing Additional Plugins Jenkins supports plugins for various tools like Maven, Gradle, Docker, and Azure DevOps.

◆ To install a plugin:

- 1** Go to Manage Jenkins → Manage Plugins
- 2** Search for the required plugin
- 3** Click Install without Restart

Setting Up Global Tool Configuration Configure Java, Maven, and Gradle in Jenkins:

- 1** Go to Manage Jenkins → Global Tool Configuration
- 2** Add paths for: JDK (C:\Program Files\Java\jdk-17)
Maven (C:\Maven\apache-maven-) Gradle (C:\Gradle\gradle-)
- 3** Click Save

Continuous Integration with Jenkins

1. Configuring Jenkins & Git Integration

Step 1: Verify Git Installation in Jenkins

1. Open Jenkins Dashboard → Manage Jenkins → Global Tool Configuration.
2. Under Git, verify the installation path (e.g., C:\Program Files\Git\bin\git.exe).

3. Click Save.

Step 2: Add GitHub Credentials in Jenkins

1. Navigate to Manage Jenkins → Manage Credentials.
2. Select Global credentials (unrestricted) → Click Add Credentials.
3. Choose Username with password or SSH Key, provide details, and click OK

2. Running a Selenium Java Test from a Local Maven Project

Step 1: Create a New Jenkins Job

1. Go to Jenkins Dashboard → Click New Item.
2. Enter a project name → Select Freestyle Project.
3. Click OK.

Step 2: Configure the Build Step

1. Scroll to Build → Click Add build step → Execute Windows Batch Command.
2. Enter the following commands (ensure correct navigation to project directory):

```
1 cd D:\Idea Projects\MVNGRDLDEMO
2 mvn test
```
- 3 Click Save → Click Build Now to execute the test.

3. Running Selenium Tests from a GitHub Repository via Jenkins

Step 1: Set Up a New Jenkins Job for GitHub Project

1. Go to Jenkins Dashboard → Click New Item.
2. Enter a project name → Select Freestyle Project.
3. Click OK.

Step 2: Configure Git Repository in Jenkins

1. Under Source Code Management, select Git.
2. Enter your GitHub repository URL (e.g., <https://github.com/your-repo-name.git>).
3. Select the Git credentials configured earlier.

Step 3: Add Build Step for Maven

1. Scroll to Build → Click Add build step → Execute Windows Batch Command.
2. Enter the Maven test command: 1 mvn test 2
3. Click Save.

Step 4: Trigger the Build

1. Click Build Now to fetch the code from GitHub and execute the Selenium tests.
2. Check the Console Output to verify test execution.

New Item

Enter an item name
jenkinsprojectwithmaven

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.
- Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

Tools

Maven Configuration

Default settings provider
Use default maven settings

Default global settings provider
Use default maven global settings

JDK installations
Add JDK

Git installations

Git
Name
Save Apply

The screenshot shows the Jenkins dashboard at localhost:8080. A context menu is open over the 'JenkinsProject' job in the list. The menu options are:

- </> Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Rename

The JenkinsProject row in the table has the following details:

S	W	Name	Last Success	Last Failure	Last Duration
		biet.build1	N/A	1 mo 10 days #5	0.11 sec
		DemoWebsite	5 days 23 hr #8	N/A	4.2 sec
		Github Connect Demo	11 days #8	N/A	2.2 sec
		JenkinsProject	2 min 12 sec #15	N/A	20 sec
		JSS-Selenium-G	4 days 15 hr #12	N/A	7.2 sec

Started by user usha
Running as SYSTEM
Building in workspace C:\Users\ushac\.jenkins\workspace\JenkinsProject
The recommended git tool is: NONE
using credential a563b68d-b523-4586-b4c1-26f0f1460774
> D:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\Users\ushac\.jenkins\workspace\JenkinsProject\.git # timeout=10
Fetching changes from the remote Git repository
> D:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/trisam3118/JenkinsProject.git # timeout=10
Fetching upstream changes from https://github.com/trisam3118/JenkinsProject.git
> D:\Program Files\Git\bin\git.exe --version # timeout=10
> git --version # 'git version 2.47.1.windows.2'
using GIT_ASKPASS to set credentials Demo on Jenkins
> D:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/trisam3118/JenkinsProject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> D:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision 766b2dafecfa3608b12fd216c67b2d4e1b9e181d (refs/remotes/origin/master)
> D:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
> D:\Program Files\Git\bin\git.exe checkout -f 766b2dafecfa3608b12fd216c67b2d4e1b9e181d # timeout=10
Commit message: "code updated in AppTest"
> D:\Program Files\Git\bin\git.exe rev-list --no-walk 766b2dafecfa3608b12fd216c67b2d4e1b9e181d # timeout=10
[JenkinsProject] \$ cmd /c call C:\Users\ushac\AppData\Local\Temp\jenkins18291852692978054275.bat

C:\Users\ushac\.jenkins\workspace\JenkinsProject>mvn test
[INFO] Scanning for projects...
[INFO]

Lab Experiment 6

Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests.

What is a CI Pipeline?

A Continuous Integration (CI) Pipeline automates the process of building, testing, and integrating code changes every time code is committed to the repository. This pipeline:

- Automatically checks out the latest code.
- Compiles the application.
- Runs tests to catch errors early.
- Notifies the team of build/test results.

Why Use Jenkins for CI?

- Automation: Jenkins automates the build and test cycle, reducing manual intervention.
- Immediate Feedback: Developers get rapid notifications of any integration issues.
- Extensibility: With hundreds of plugins available, Jenkins can integrate with version control systems, build tools (Maven, Gradle), testing frameworks, and more.
- Pipeline as Code: Using Jenkins Pipelines (defined in a Jenkinsfile), you can manage the CI process as part of your source code repository.

Setting Up a CI Pipeline with Jenkins (Freestyle Project)

This section explains how to create a CI pipeline as a Freestyle project that integrates with a Maven project.

Step 1: Create a New Jenkins Job

1. Log into Jenkins:

- o Open your web browser and navigate to your Jenkins URL (e.g., <http://localhost:8080> or your cloud instance URL).
- o Log in with your admin credentials.

2. Create a New Job:

- o On the Jenkins dashboard, click on “ New Item” .
- o Enter an Item Name: For example, Maven-CI (or Gradle-CI if you prefer Gradle).
- o Select “ Freestyle project” .
- o Click “ OK” .

Step 2: Configure Source Code Management (SCM)

1. Select SCM:

- o In the job configuration page, scroll down to the “ Source Code Management” section.

- o Select “ Git” (if using Git for version control).

2. Enter Repository Details:

- o Repository URL: Enter the URL of your Git repository (for example, <https://github.com/yourusername/your-maven-project.git>).
- o Credentials: If your repository is private, click “ Add” to provide the necessary credentials.

- o Optionally, specify the Branch Specifier (e.g., */main or master).

Step 3: Add Build Steps

A. For a Maven Project

1. Add Maven Build Step:

- o Scroll down to “ Build” and click on “ Add build step” .
- o Select “ Invoke top-level Maven targets” .
- o Goals: In the Goals field, enter:
- o clean package

This command instructs Maven to clean the previous build artifacts, compile the code, run tests, and package the application into a JAR/WAR file. Optionally, set the POM File location if it is not in the default location (**pom.xml**).

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>JenkinsProject</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>JenkinsProject</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
```

```
<dependencies>

<dependency>
    <groupId>org.testng</groupId>

    <artifactId>testng</artifactId>
    <version>7.10.2</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.25.0</version>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-plugin</artifactId>
            <version>3.2.0</version>
        <executions>
```

```
<execution>

    <phase>prepare-package</phase> <!-- Before packaging -->

    <goals>

        <goal>copy-resources</goal>

    </goals>

    <configuration>

        <outputDirectory>${project.basedir}/docs</outputDirectory>

        <resources>

            <resource>

                <directory>src/main/resources</directory>

                <includes>

                    <include>**/*</include> <!-- Copy all files in src/main/resources -->

                </includes>

            </resource>

        </resources>

    </configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>
```

Now, index.html, style.css and logo.png files from docs folder is added to your Jenkins Project

Apptest.java

```
package org.test;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import static org.testng.Assert.assertTrue;

public class WebpageTest {

    private static WebDriver driver;

    @BeforeTest
    public void openBrowser() throws InterruptedException {
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        Thread.sleep(2000);
        driver.get("https://trisam3118.github.io/demoWebsite/");// here enter static website URL from
        your GITHUB
    }

    @Test
    public void titleValidationTest(){
        String actualTitle = driver.getTitle();
        String expectedTitle = "Computer Science and Engineering(Data Science)";
    }
}
```

```
Assert.assertEquals(actualTitle, expectedTitle);

assertTrue(true, "Title should contain 'Computer Science and Engineering(Data Science)'");

}

@AfterTest

public void closeBrowser() throws InterruptedException {

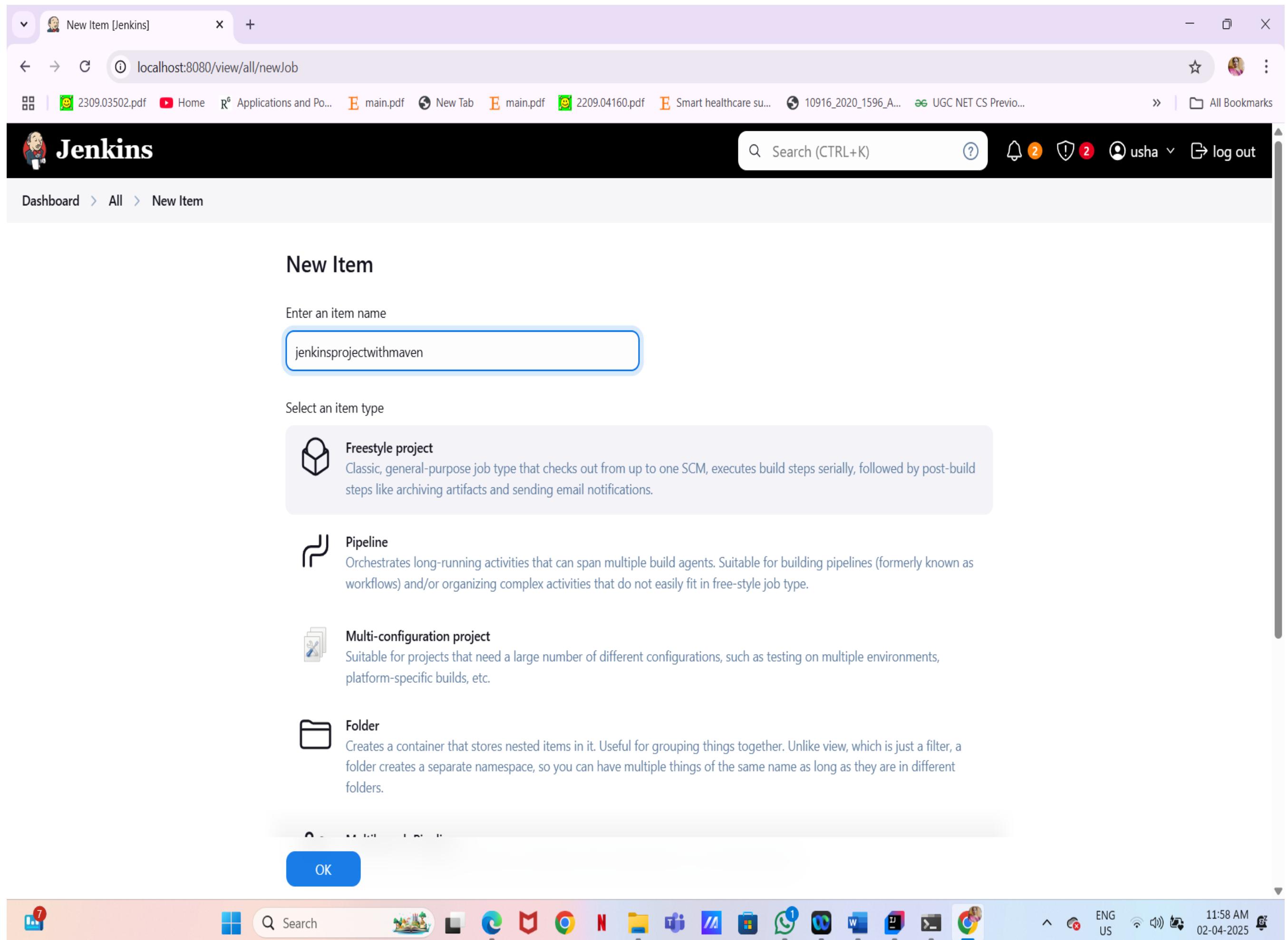
    Thread.sleep(1000);

    driver.quit();

}

}
```

Note: Run this file it should open browser and validates the title.



Lab Experiment 7

Configuration Management with Ansible: Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook

1. What Is Ansible?

Ansible is an open-source IT automation and configuration management tool. It allows you to manage multiple servers and perform tasks such as:

- Configuration Management: Automate the configuration of servers.
- Application Deployment: Deploy applications consistently.
- Orchestration: Coordinate complex IT workflows and processes. Key Concepts in Ansible
- Inventory: An inventory is a file (usually in INI or YAML format) that lists the hosts (or groups of hosts) you want to manage. It tells Ansible which machines to target.
- Playbook: A playbook is a YAML file that defines a set of tasks to be executed on your target hosts. It is the heart of Ansible automation.

In a playbook, you specify:

- Hosts: The target machines (or groups) on which the tasks should run.
 - Tasks: A list of actions (using modules) that should be executed.
 - Modules: Reusable, standalone scripts that perform specific actions (e.g., installing packages, copying files, configuring services).
- Modules: Ansible comes with a large collection of built-in modules (such as apt, yum, copy, service, etc.). These modules perform specific tasks on target hosts. You can also write custom modules if needed.

2. Why Use Ansible?

- Agentless: Ansible uses SSH to communicate with target hosts, so no agent needs to be installed on them.
- Simplicity: Playbooks use simple YAML syntax, making them easy to write and understand.
- Idempotence: Ansible tasks are idempotent, meaning running the same playbook multiple times yields the same result, ensuring consistency.
- Scalability: Ansible can manage a small number of servers to large infrastructures with hundreds or thousands of nodes.

Step 1: Vagrantfile

```
config.vm.box = "ubuntu/bionic64"
```

Step 2:

```
C:\vagrantproject1>mkdir ansible-controller
```

```
C:\vagrantproject1>cd ansible-controller
```

Step 3: vagrantfile

```
config.vm.provision "shell", inline: <<-SHELL
```

```
sudo yum install epel-release -y
```

```
sudo yum install ansible -y
```

```
SHELL
```

```
end
```

```
Vagrant.configure("2") do |config|
```

```
  config.vm.define "ansible-controller" do |controller|
```

```
    controller.vm.hostname = "controller"
```

```
    controller.vm.box = "centos/7"
```

```
  controller.vm.provision "shell", inline: <<-SHELL
```

```
    sudo yum install epel-release -y
```

```
    sudo yum install ansible -y
```

```
  SHELL
```

```
end
```

```
end
```

Step 4: vagrant ssh

Step 5: sudo vi /etc/resolv.conf

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

Step 6: ping google.com

Step 7: sudo vi /etc/yum.repos.d/CentOS-Base.repo

[base]

```
name=CentOS-7 - Base  
baseurl=http://vault.centos.org/7.9.2009/os/x86_64/  
gpgcheck=1  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

[updates]

```
name=CentOS-7 - Updates  
baseurl=http://vault.centos.org/7.9.2009/updates/x86_64/  
gpgcheck=1  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

[extras]

```
name=CentOS-7 - Extras  
baseurl=http://vault.centos.org/7.9.2009/extras/x86_64/  
gpgcheck=1  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

Step 8: sudo yum clean all

Step 9: sudo yum install epel-release -y

Step 10: ssh-keygen

Step 11: cd ~/.ssh

Step 12: vagrant@controller .ssh]\$ ls

authorized_keys id_rsa id_rsa.pub

Step 13:

```
C:\vagrantproject1\ansible-controller>vagrant validate
```

Vagrantfile validated successfully.

Step 14:

```
C:\vagrantproject1\ansible-controller>vagrant up
```

```
Bringing machine 'ansible-controller' up with 'virtualbox' provider...
==> ansible-controller: Checking if box 'centos/7' version '2004.01' is up to date...
==> ansible-controller: Machine already provisioned. Run `vagrant provision` or use
the `--provision`
==> ansible-controller: flag to force provisioning. Provisioners marked to run
always will still run.
```

Step 14: vagrant provision

Step 15:

```
C:\vagrantproject1\ansible-controller>vagrant ssh
```

```
Last login: Thu Apr 10 03:56:08 2025 from 10.0.2.2
```

Step 16:[vagrant@controller ~]\$ ansible --version

```
ansible 2.9.27
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/vagrant/.ansible/plugins/modules',
  u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Apr 2 2020, 13:16:51) [GCC 4.8.5 20150623 (Red
  Hat 4.8.5-39)]
```

ANSIBLE INSTALLED SUCCESSFULLY... ... !

Inventory file contains ansible-host machine details

vi hosts

```
webserver
[192.168.33.10]
dbserver
[192.168.33.11]
```

vi playbook.yml

```
- name: Configure Web Server
  hosts: webservers
```

```
become: true
tasks:
  - name: Install Apache
    yum:
      name: httpd
      state: latest

  - name: Start and enable Apache
    service:
      name: httpd
      state: started
      enabled: true

  - name: Create a custom index.html
    copy:
      dest: /var/www/html/index.html
      content: "<h1>Hello from Ansible Web Server!</h1>"

  - name: Configure DB Server
    hosts: dbservers
    become: true
    tasks:
      - name: Install Apache
        yum:
          name: httpd
          state: latest

      - name: Start and enable Apache
        service:
          name: httpd
          state: started
          enabled: true
```

```
C:\Windows\System32\cmd.e x + v

==> db: Clearing any previously set forwarded ports...
==> db: Fixed port collision for 22 => 2222. Now on port 2201.
==> db: Clearing any previously set network interfaces...
==> db: Preparing network interfaces based on configuration...
db: Adapter 1: nat
db: Adapter 2: hostonly
==> db: Forwarding ports...
db: 80 (guest) => 8082 (host) (adapter 1)
db: 22 (guest) => 2201 (host) (adapter 1)
==> db: Booting VM...
==> db: Waiting for machine to boot. This may take a few minutes...
db: SSH address: 127.0.0.1:2201
db: SSH username: vagrant
db: SSH auth method: private key
==> db: Machine booted and ready!
==> db: Checking for guest additions in VM...
db: No guest additions were detected on the base box for this VM! Guest
db: additions are required for forwarded ports, shared folders, host only
db: networking, and more. If SSH fails on this machine, please install
db: the guest additions and repack the box to continue.
db:
db: This is not an error message; everything may continue to work properly,
db: in which case you may ignore this message.
==> db: Setting hostname...
==> db: Configuring and enabling network interfaces...
==> db: Rsyncing folder: /cygdrive/c/vagrantproject1/ansible-controller/ansible-hosts/ => /vagrant
==> db: Machine already provisioned. Run 'vagrant provision' or use the '--provision'
==> db: flag to force provisioning. Provisioners marked to run always will still run.

C:\vagrantproject1\ansible-controller\ansible-hosts>vagrant status
Current machine states:

web           running (virtualbox)
db            running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run 'vagrant status NAME'.

C:\vagrantproject1\ansible-controller\ansible-hosts>
```

```
vagrant@controller:~/ansible  X + - 11:18 AM 23-04-2025
└ name: Configure Web Server
  hosts: webservers
  become: true
  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: latest

    - name: Start and enable Apache
      service:
        name: httpd
        state: started
        enabled: true

    - name: Create a custom index.html
      copy:
        dest: /var/www/html/index.html
        content: "<h1>Hello from Ansible Web Server!</h1>"

- name: Configure db Server
  hosts: dbservers
  become: true
  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: latest

    - name: Start and enable Apache
      service:
        name: httpd
        state: started
        enabled: true

```
"playbook.yml" 35L, 690C
```

```
vagrant@controller:~/ansible X + v
db: 80 (guest) => 8082 (host) (adapter 1)
db: 22 (guest) => 2201 (host) (adapter 1)
==> db: Booting VM...
==> db: Waiting for machine to boot. This may take a few minutes...
db: SSH address: 127.0.0.1:2201
db: SSH username: vagrant
db: SSH auth method: private key
==> db: Machine booted and ready!
==> db: Checking for guest additions in VM...
db: No guest additions were detected on the base box for this VM! Guest
db: additions are required for forwarded ports, shared folders, host only
db: networking, and more. If SSH fails on this machine, please install
db: the guest additions and repack the box to continue.
db:
db: This is not an error message; everything may continue to work properly,
db: in which case you may ignore this message.
==> db: Setting hostname...
==> db: Configuring and enabling network interfaces...
==> db: Rsyncing folder: /cygdrive/c/vagrantproject1/ansible-controller/ansible-hosts/ => /vagrant
==> db: Machine already provisioned. Run 'vagrant provision' or use the '--provision'
==> db: flag to force provisioning. Provisioners marked to run always will still run.

C:\vagrantproject1\ansible-controller\ansible-hosts>vagrant status
Current machine states:

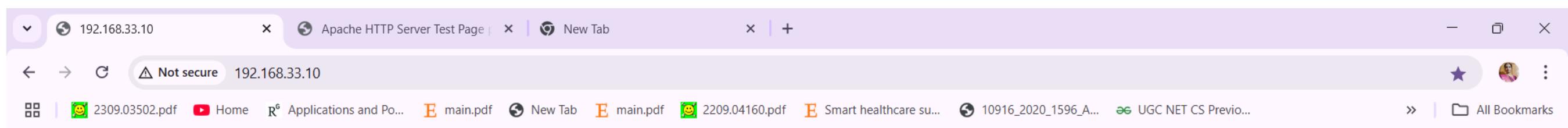
web running (virtualbox)
db running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run 'vagrant status NAME'.

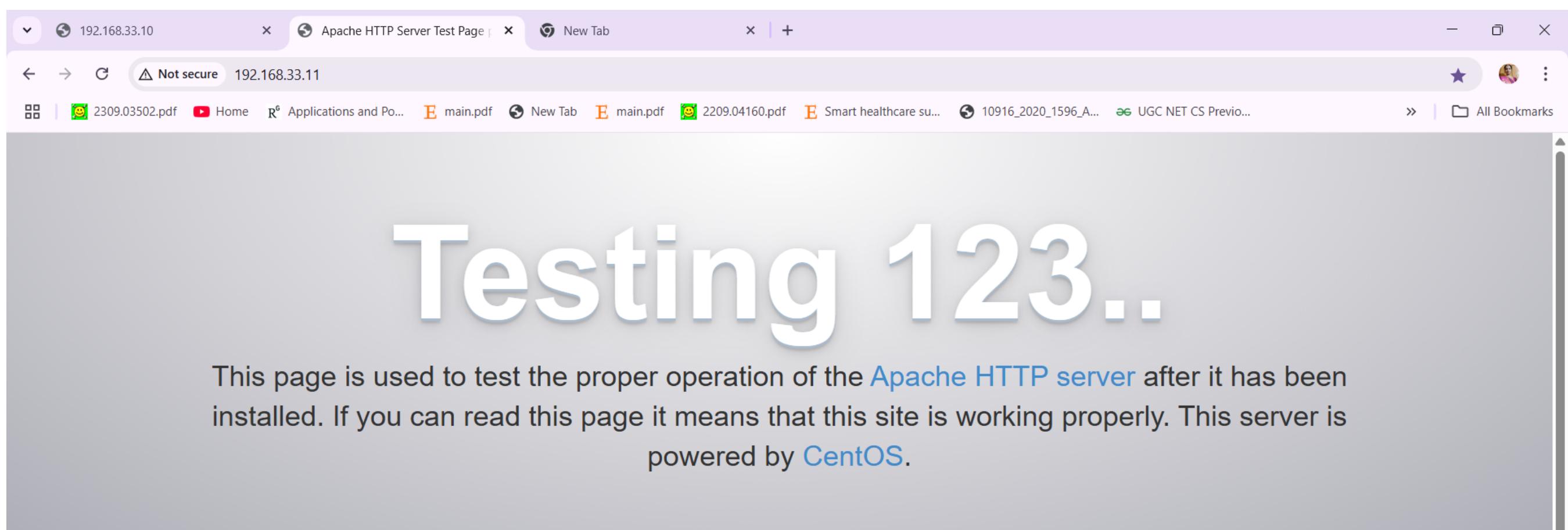
C:\vagrantproject1\ansible-controller\ansible-hosts>vagrant ssh
This command requires a specific VM name to target in a multi-VM environment.

C:\vagrantproject1\ansible-controller\ansible-hosts>cd..

C:\vagrantproject1\ansible-controller>vagrant ssh
Last login: Tue Apr 15 10:27:36 2025 from 10.0.2.2
[vagrant@controller ~]$ ls
ansible-project
[vagrant@controller ~]$ cd ansible-project
[vagrant@controller ansible-project]$ ls
hosts playbook.yml
[vagrant@controller ansible-project]$ vi hosts
[vagrant@controller ansible-project]$ vi playbook.yml
[vagrant@controller ansible-project]$ ansible-playbook -i hosts playbook.yml|
```



**Hello from Ansible Web Server!**



#### Just visiting?

The website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting [www.example.com](http://www.example.com), you should send e-mail to "webmaster@example.com".

#### Are you the Administrator?

You should add your website content to the directory `/var/www/html/`.

To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

#### Promoting Apache and CentOS

You are free to use the images below on Apache and CentOS Linux powered HTTP servers. Thanks for using Apache and CentOS!



## Lab Experiment 8

**Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins**

## INTRODUCTION

**Introduction to DevOps terminology definition and concepts, Understanding version control tool - Git, Overview of Build Automation tools, Key differences between Maven and Gradle.**

### 1. What is DevOps?

**DevOps** is a set of cultural philosophies, practices, and tools that combine software development (Dev) and IT operations (Ops). Its goal is to shorten the system's development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives. DevOps promotes:

**Collaboration:** Breaking down silos between development and operations teams.

**Automation:** Automating repetitive tasks (builds, tests, deployments) to improve efficiency.

**Continuous Integration/Delivery (CI/CD):** Regularly integrating code changes and deploying them to production with minimal manual intervention.

**Monitoring and Feedback:** Constantly monitoring the performance and behavior of applications in production to rapidly address issues.

### 2. Why is DevOps Used?

**Speed and Agility:** Faster development cycles and quicker time-to-market.

**Quality:** Automated testing and integration help catch issues early.

**Reliability:** Frequent, smaller updates reduce the risk of large-scale failures.

**Efficiency:** Automation reduces manual errors and repetitive tasks.

**Scalability:** Processes and infrastructure can grow with the business.

### 3. Examples of DevOps in Action.

**Continuous Integration (CI):** Tools like Jenkins or Azure Pipelines automatically build and test code on every commit.

**Continuous Deployment (CD):** Systems automatically deploy tested code to production environments.

**Monitoring:** Tools such as Prometheus, Grafana, or New Relic continuously monitor application performance.

### 4. Understanding software development life cycle (SDLC).

Before diving into build tools like Maven and Gradle, it is essential to understand how software has traditionally been developed. The software development life cycle provides a structured approach to build software efficiently. One of the earliest models of SDLC is the waterfall model, which follows a linear and sequential approach, where each phase must be fully completed before moving to the next phase. However, this model has several limitations which led to the adoption of agile and DevOps methodologies.

### 5. Drawbacks of Waterfall model

- Rigid & Inflexible – Changes cannot be made once a phase is completed.
- Late Bug Detection – Testing happens after coding, making bug fixes costly.
- Slow Development Process – Each phase must be completed before the next starts.
- Not Suitable for Modern Agile Environments – Does not support rapid iteration.

## 6. Introduction to agile methodology

To overcome Waterfall's limitations, the Agile Model was introduced. Agile promotes iterative and incremental development, allowing teams to deliver software faster with continuous feedback.

- a) Key Agile Concepts
- b) Sprints – Development happens in short cycles, usually lasting 7-14 days.
- c) Scrum Master – Facilitates Agile processes and removes blockers for the team.
- d) Daily Stand-up Meetings – Short meetings where team members discuss:
  - What they did yesterday?
  - What they plan to do today?
  - Any blockers?
- e) Retrospective Meetings – At the end of each sprint, the team reflects on what worked well and what can be improved.

## 7. Transition from agile to DevOps

While agile improved development, deployment and operations remained slow. This led to the birth of DevOps, which introduced:

- Continuous Integration and Deployment (CI/CD).
- Automation of Builds, Testing and Deployment.
- Faster and more reliable software releases.

**Note:** To make a DevOps sufficient, we need build tools like Meghan and Gradle. It simplifies project management.

## 8. Problems with manual Approach

- + Time consuming - Downloading, adding and adding dependencies manually is inefficient.
- + Error prone - Missing jar files or incorrect configurations can break the project
- + Difficult to manage - Dependencies are not automatically updated.
- + Not scalable - Every team member must manually configure their setup.

## 9. What is version control?

Version control also known as source control is a system that tracks and manages changes to files, especially code overtime following teams to collaborate efficiently and revert to previous version versions if needed. Before moving to Magan and Gradle, we must first understand version control systems. Like GIT partial control plays a crucial role in devops enabling

- ✓ efficient code management - track changes, revert to previous versions and collaborate seamlessly.
- ✓ Team collaboration - multiple developers can work on the same project without conflicts.
- ✓ Integration with CI/CD pipelines, automate spools, testing and deployment.

## 10. Setting Up Git on Local System

**Step1:** Check if git is already installed.

**cmd prompt → git --version**

**Step2:** If not installed download Git installer from <https://git-scm.com/>

**Step3:** Run installer and install git

## Create GitHub Account

**Step 1: Go to <https://github.com/> and sign up for a new account**

**Step 2: Login to GitHub**

**Step 3: Create a new Repository (Private or Public)**

## Basic Git Commands

Step 1: Create a new folder and open git bash and go to the folder location.

Step 2: Run the Commands for git configuration

```
git config --global user.email " sampleGitHub@email.com"
git config --global user.name " sampleGitHub_username"
```

Step 3: Initialize git using this command.

```
git init
```

Step 4: Add some files in the folder by using this command.

```
git touch <filename1.txt>
git touch <filename2.html>
git touch <filename3.py>
git touch <filename4.js>
```

Step 5: Run these commands to check status at files and commit your changes or updates.

```
git status [To check status]
git add <filename> [To add a particular file]
git add . [To add all the files]
git commit -m " Commit Message" [To commit all your changes]
git remote add origin " github-url" [To add remote repo link to local repo]
git push -u origin master [To push your local changes to remote]
```

## 11. Why do we need Build Tools (Maven & Gradle)?

- Automatically manage dependencies – No need to download JARs manually.
- Simplify project configuration – A single configuration file (pom.xml for Maven, build.gradle.kts for Gradle) handles everything.

- Enable easy build & testing – Run tests and package applications using simple commands.
- Ensure consistency – The same project setup works on different machines.

**Note:** Both Maven and Gradle are used to automate the build process and manage project dependencies. Here's why they are so popular in the DevOps ecosystem:

- Automated Build and Testing: They allow developers to compile code, run tests, and package applications without manual intervention.
- Consistent Build Environment: By enforcing standardized project structures and dependency management, they help avoid the “it works on my machine” problem.
- Ease of Integration: Both tools integrate well with Continuous Integration (CI) servers like Jenkins, enabling automated pipelines.
- Dependency Resolution: They simplify the process of managing external libraries and ensure that all developers are using the same versions.

## 12. What is Maven?

**Maven** is a build automation and project management tool primarily used for Java projects. It uses a central configuration file known as the POM (Project Object Model), written in XML, which defines the project structure, its dependencies, build order, and plugins.

### Features

- Convention over Configuration: Maven enforces a standard directory structure (e.g., src/main/java, src/test/java), which simplifies project setup.
- Dependency Management: Automatically downloads and manages external libraries and dependencies from repositories like Maven Central.
- Build Lifecycle: Defines a fixed lifecycle (e.g., compile, test, package, install, deploy) which standardizes the build process.

- **Plugin Ecosystem:** Provides a rich set of plugins to extend functionality (e.g., unit testing, code coverage, reporting).

## 13. What is Gradle?

**Gradle** is a modern build automation tool that is known for its flexibility and performance. It uses a Groovy or Kotlin DSL (Domain Specific Language) to define build logic, which allows for more dynamic and customizable configurations compared to Maven's XML-based approach.

### Features

- **Flexible Build Scripts:** Instead of a rigid XML file, Gradle build scripts written in Groovy or Kotlin allow you to include logic, conditionals, and loops.
- **Incremental Builds:** Gradle tracks changes in source files and only rebuilds what is necessary, which can significantly speed up the build process.
- **Multi-project Builds:** Easily manages complex projects with multiple modules or subprojects.
- **Extensibility:** A robust plugin system that enables you to integrate various languages, frameworks, and tools.
- **Parallel Execution:** Can run tasks in parallel, optimizing build times for large projects.

## 14. Differences between Maven and Gradle

| Aspect                       | Maven                                                                               | Gradle                                                                                               |
|------------------------------|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <b>Build Script Language</b> | Uses an XML-based configuration file ( <code>pom.xml</code> ).                      | Uses a DSL based on Groovy or Kotlin ( <code>build.gradle</code> or <code>build.gradle.kts</code> ). |
| <b>Configuration Style</b>   | Declarative and rigid – follows strict conventions (convention over configuration). | Flexible and dynamic – allows you to write custom logic and conditions within the build script.      |

|                                    |                                                                                                                          |                                                                                                                                 |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
|                                    | configuration).                                                                                                          |                                                                                                                                 |
| <b>Build Lifecycle</b>             | Provides a fixed lifecycle (e.g., validate, compile, test, package, install, deploy).                                    | Uses a task-based approach where tasks can be defined, customized, and linked in a flexible manner.                             |
| <b>Dependency Management</b>       | Manages dependencies through the POM file; downloads them from central repositories (e.g., Maven Central).               | Similar dependency management; supports dynamic version resolution and customizable dependency configurations.                  |
| <b>Performance</b>                 | Generally slower for large projects because of the fixed build lifecycle and less emphasis on incremental builds.        | Often faster, thanks to incremental builds, caching, and parallel execution of tasks.                                           |
| <b>Extensibility &amp; Plugins</b> | Rich ecosystem of plugins but customization can be more challenging due to XML's verbosity and limitations in scripting. | Highly extensible through its scripting capabilities; writing custom tasks or plugins in Groovy/Kotlin is more straightforward. |
| <b>Multi-Project Builds</b>        | Handles multi-module projects well but requires a strict directory layout and a parent POM for aggregating modules.      | Excels in multi-project builds with simple configuration, allowing each subproject to be configured in a flexible manner.       |

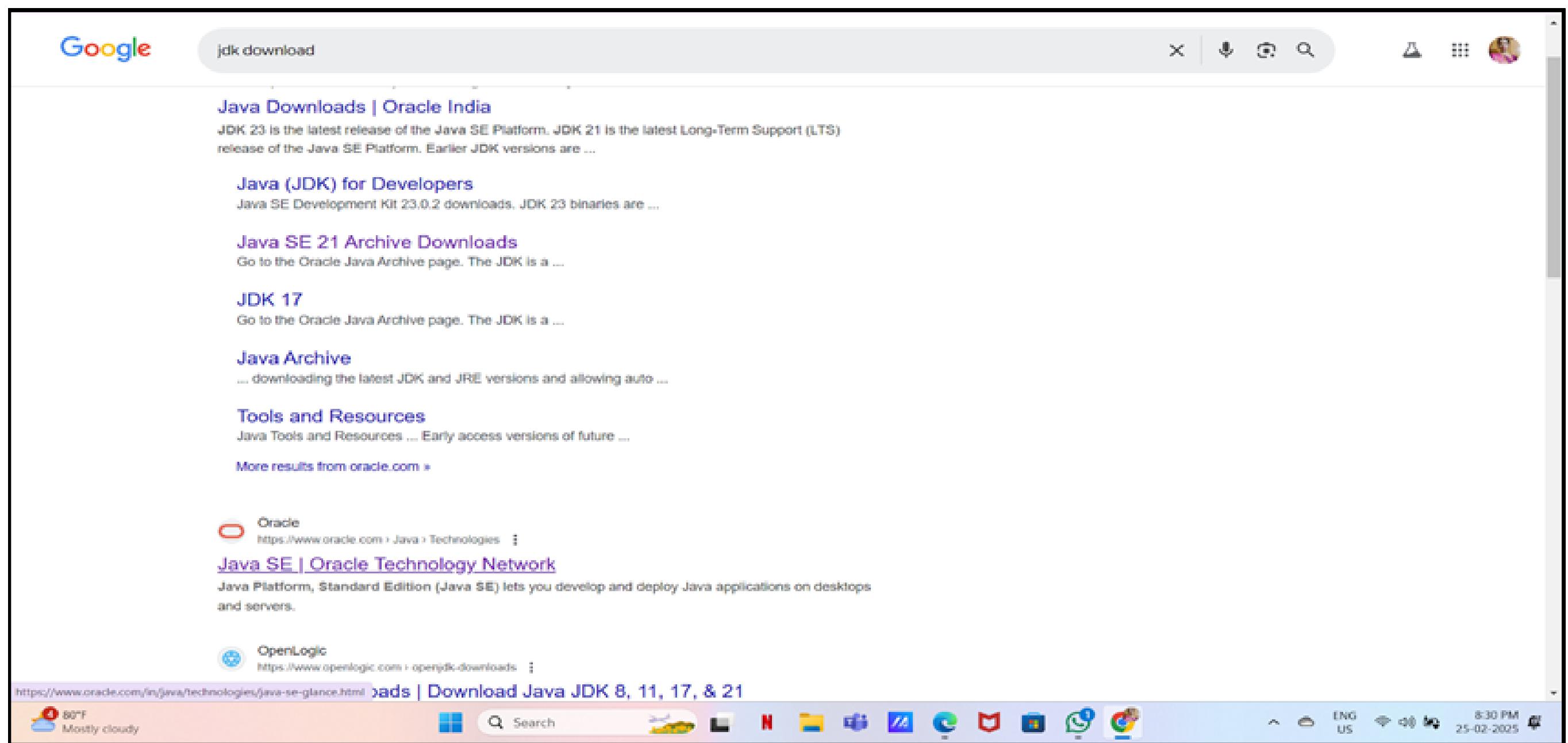
|                                 |                                                                                                                         |                                                                                                                                   |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>Learning Curve</b>           | Easier for beginners due to its structured, convention-based approach but can become complex with large configurations. | More flexible but may have a steeper learning curve initially if you need to leverage its dynamic features and custom logic.      |
| <b>Community &amp; Maturity</b> | Has been around longer, so many legacy projects and extensive documentation exist.                                      | Relatively newer; it has gained popularity due to its modern features, especially in Android and multi-language projects.         |
| <b>Integration with CI/CD</b>   | Well-supported by most CI/CD tools (like Jenkins, Azure Pipelines) with stable, predictable behavior.                   | Also integrates seamlessly with CI/CD tools and is often chosen for its faster build times and flexibility in pipeline scripting. |

# INSTALLATION

## Installation and Setup of JDK-17, IDE (IntelliJ Idea), GIT, Maven, Gradle and Jenkins.

### 1. JDK Installation steps

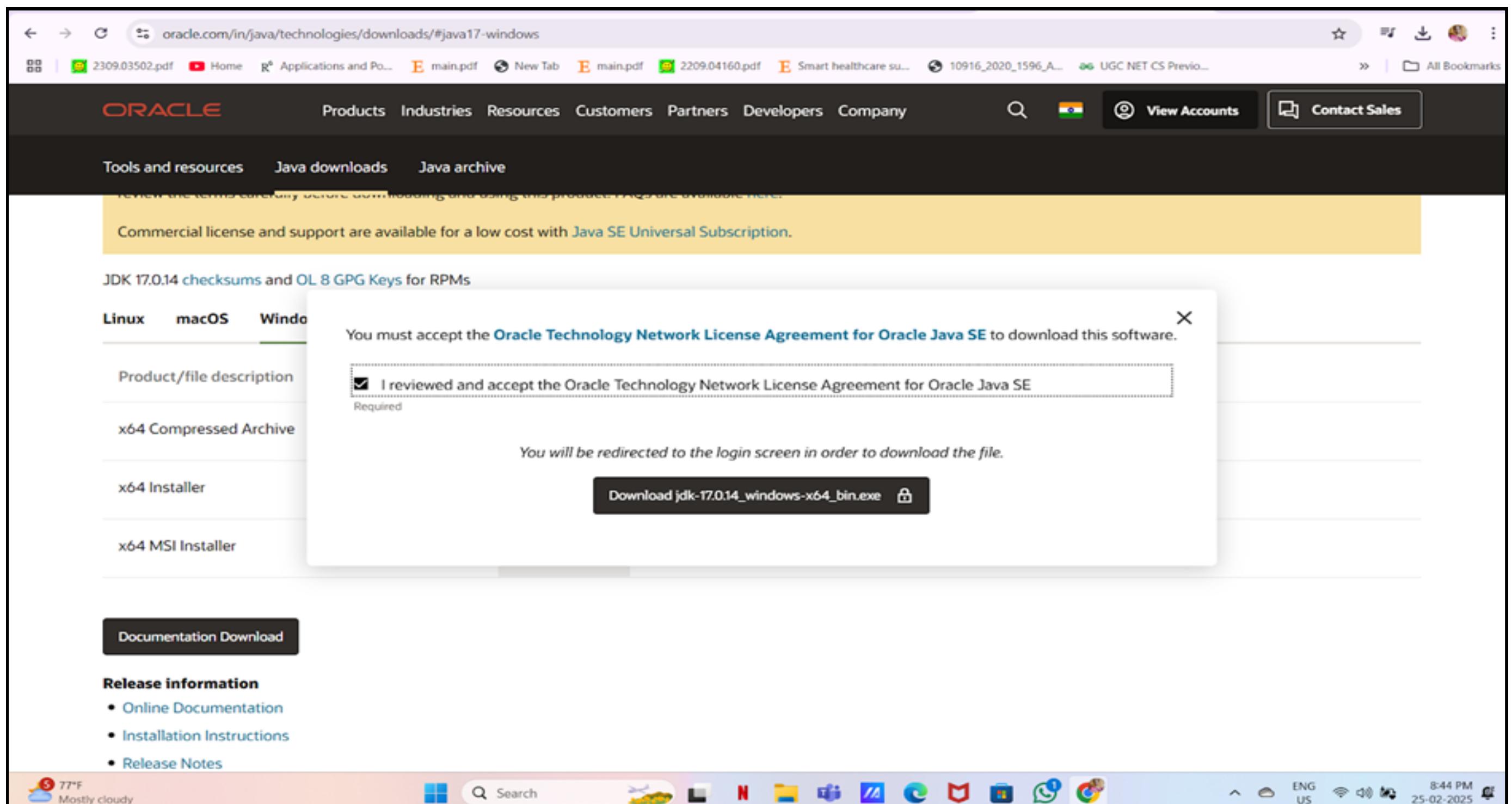
Goto → Browser → JDK download → Select-JAVA SE(Standard edition)



→

The screenshot shows the Oracle Java SE at a Glance page. At the top, there's a banner for JavaOne 2025 with a call-to-action button 'Register now and save \$100'. Below the banner, the Java logo is displayed next to the heading 'Java SE at a Glance'. A sub-section titled 'What's New' highlights 'Java Platform, Standard Edition 23', noting it's the latest release. To the right, a 'Know More' section is visible. The page footer includes a weather widget showing '77°F Mostly cloudy'.

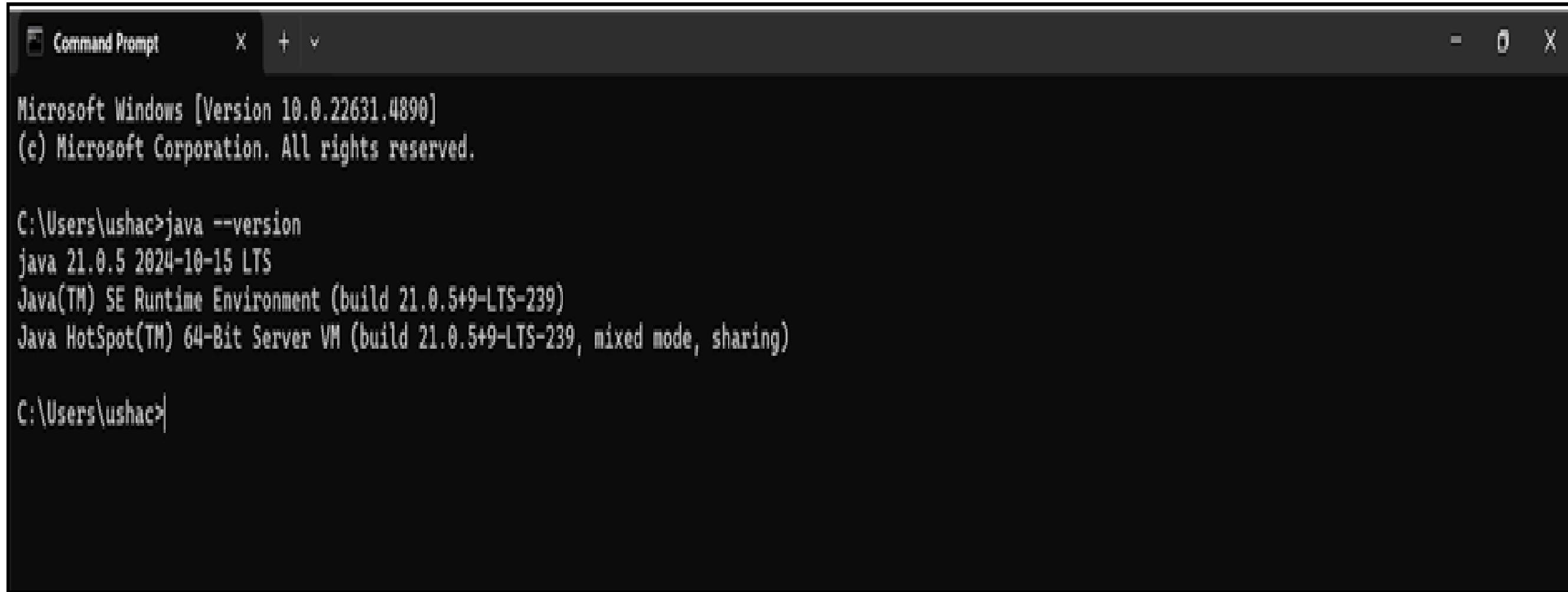
The screenshot shows the Oracle Java downloads page for JDK 17.0.14. It features a table for Windows users, listing three download options: 'x64 Compressed Archive' (172.81 MB), 'x64 Installer' (153.99 MB), and 'x64 MSI Installer' (152.75 MB). Each option has a download link. Below the table, there's a 'Documentation Download' button and a 'Release information' section with links to 'Online Documentation' and 'Installation Instructions'. The page footer includes a weather widget showing '77°F Mostly cloudy'.



JDK (java development kit) with the help of which we have compiler, JVM-convert java source code to byte code (with the help of which we can convert java program and make it compatible to any OS).

Check in -

**cmd prompt → type java --version , if installed shows the details of JDK**



```
Microsoft Windows [Version 10.0.22631.4890]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ushac>java --version
java 21.0.5 2024-10-15 LTS
Java(TM) SE Runtime Environment (build 21.0.5+9-LTS-239)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.5+9-LTS-239, mixed mode, sharing)

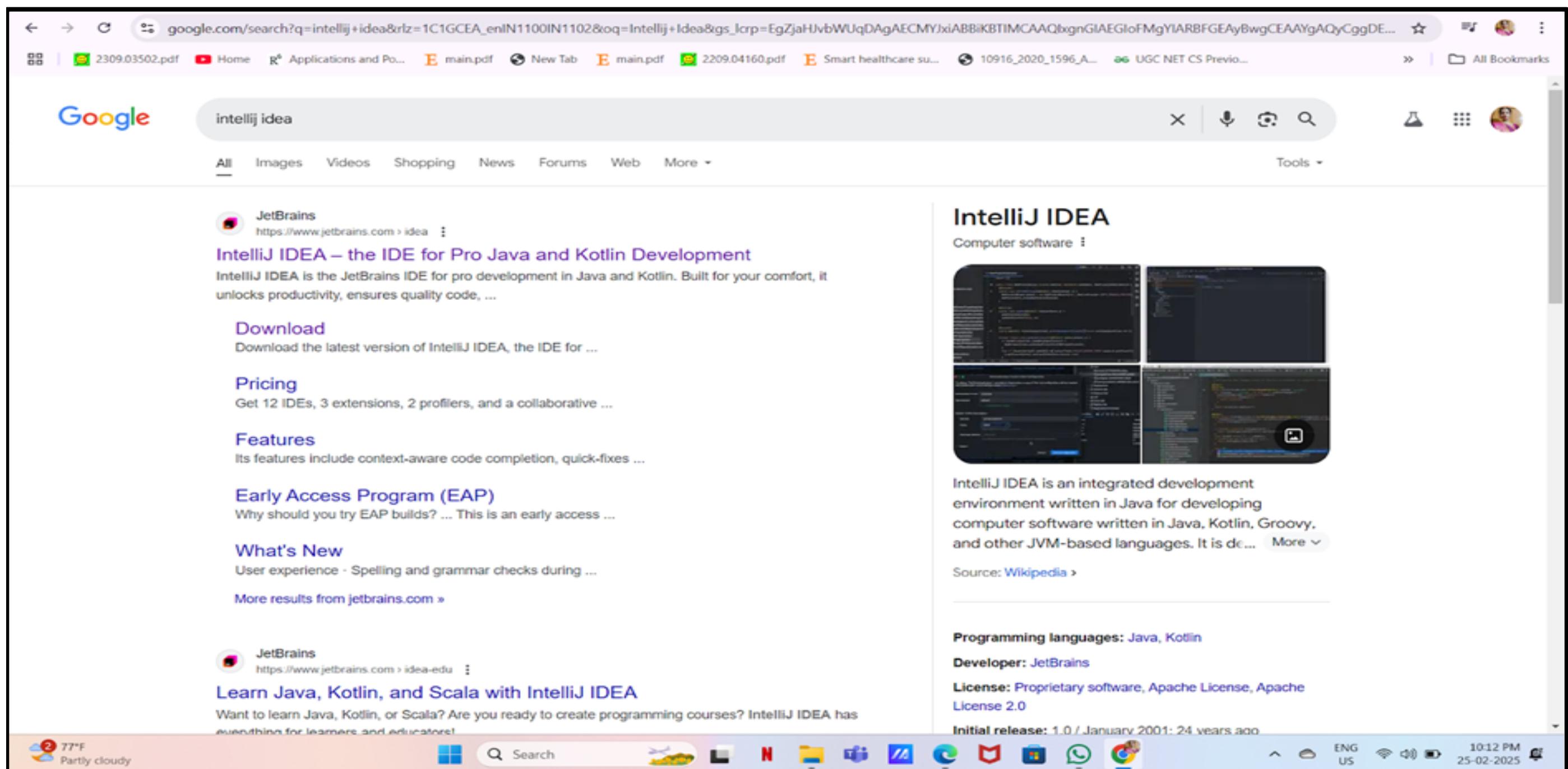
C:\Users\ushac>
```

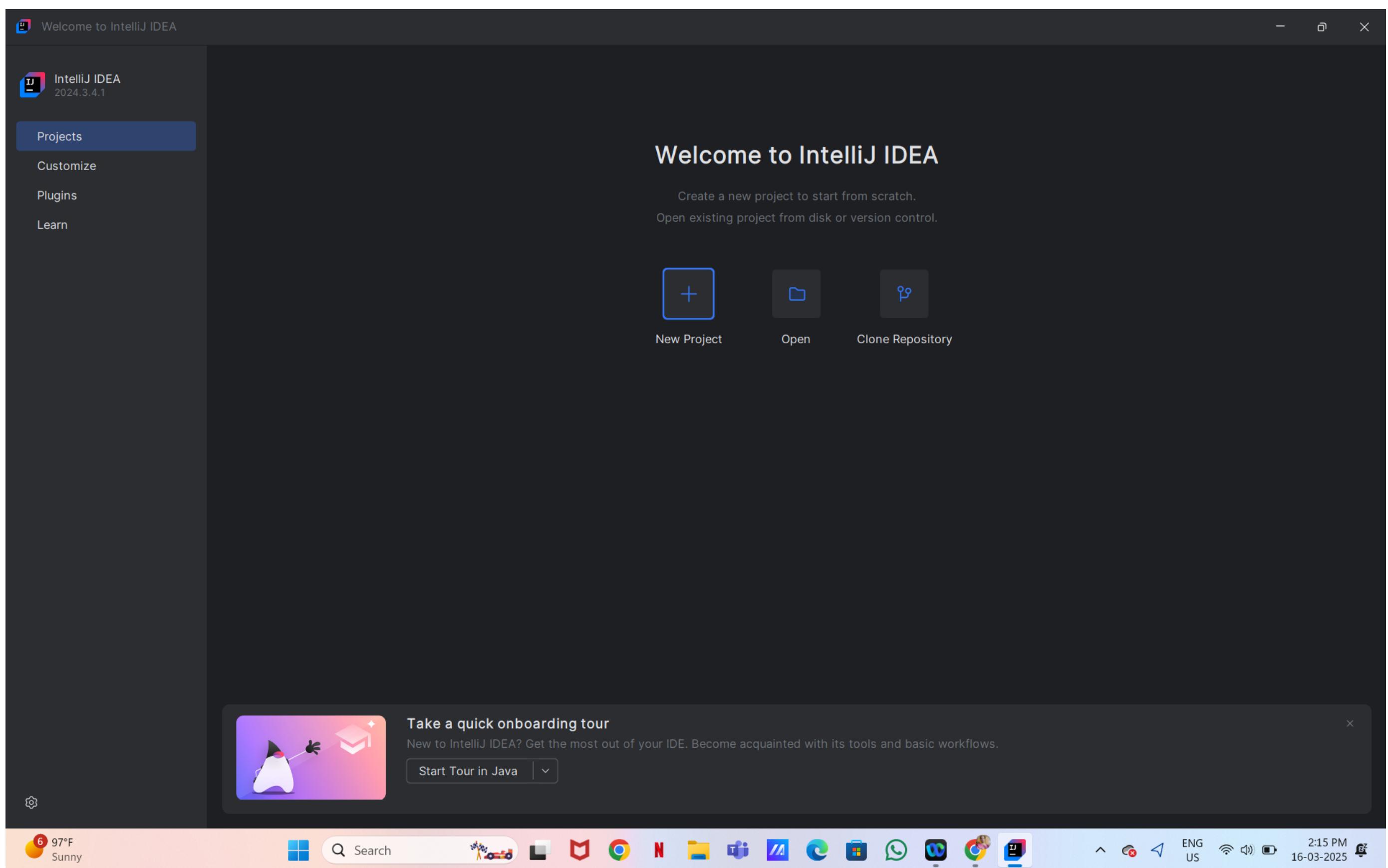
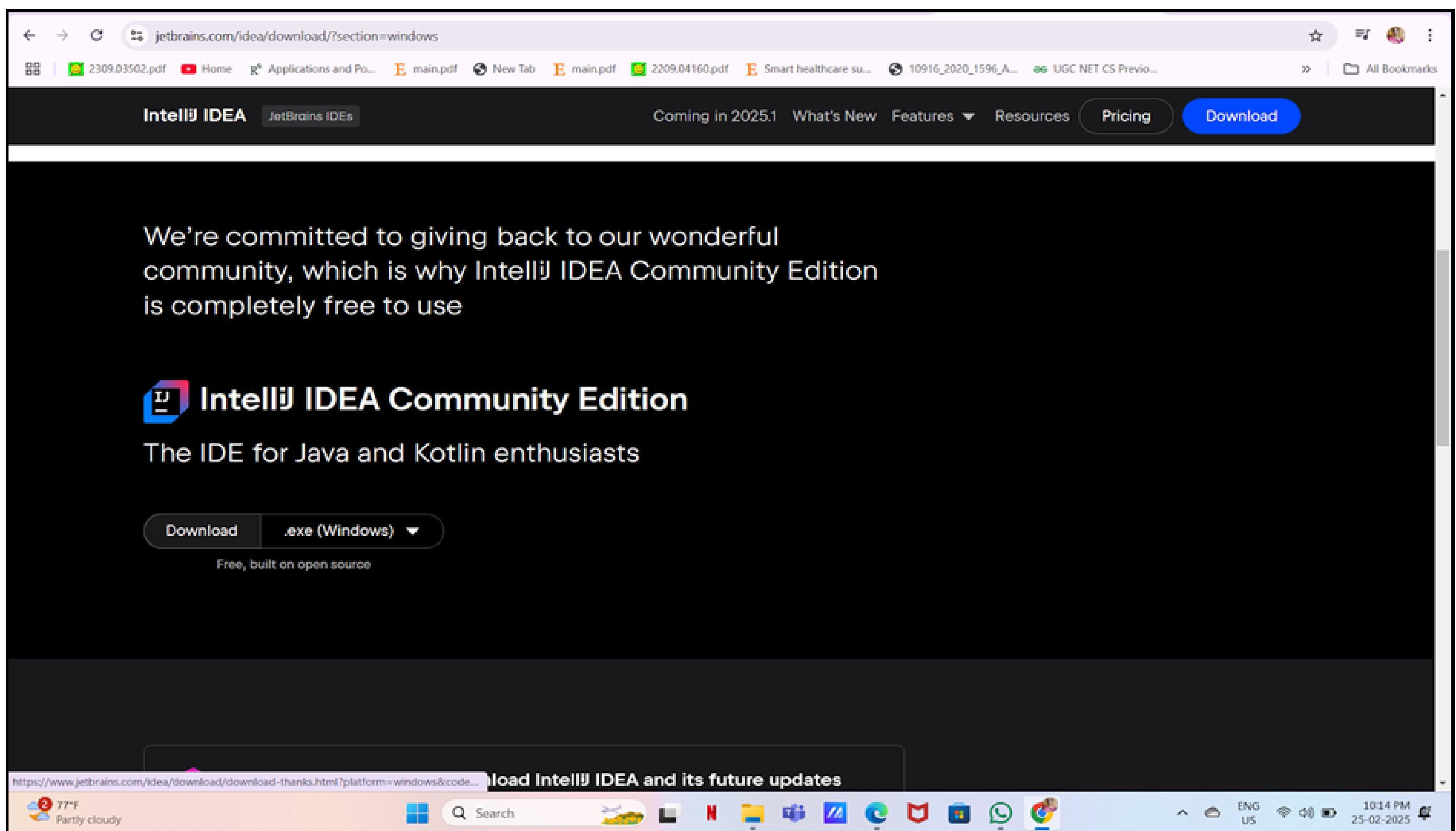
## 2. IDE (Integrated development Environment) used to write and execute JAVA programs.

**Types of IDE –**

**Eclipse, NetBeans, IntelliJ Idea (best totally dedicated for JAVA).**

### Installation of IntelliJ Idea





### 3. Anatomy of JAVA Program

- Smallest building block in a java program is **Function**.
- What is a function?
  - Function is a block of code that performs a specific task.
  - For example, if a project is given to a team, then each team member performs a specific task similarly function do the task assigned to it.
  - Declaration of a function

```
Return Type Name () {
 ...
}
```

Some functions return values like number, date, time.

Some functions return nothing called **void**

```
Void Name (){
 ...
}
```

Here **Name** of the function should be meaningful like **sendEmail()**, **saveMessage()**.

We have **Parentheses ()** where we write **Arguments or Parameters**.

Within curly braces {} we write actual java code .

- Every java program should have at least one **main()** function, it is the entry point of a java program. Functions cannot exist on their own, It belongs to a class, class is a container that has one or more related functions.
- A class is basically used to organize our code.
- Every Java program should have at least one class that is - main () class.

```
Class Main()
{
 Void main()
 {
 ...
 }
}
```

Here void main() is called a **method**.

- In JAVA, **function** is a part of the class called a **method** but in **PYTHON** we can have a method outside the class called a **function**.
- In java every class and method should have an **Access modifier**.
- Access modifiers determine whether other classes and methods in this program can access this class method or not.
- Types of access modifiers are private, public and so on.

```
Public Class Main ()
{
 Public Void main()
 {

 }
}
```

here the Main starts with the capital letter. This is because java uses a naming convention called **Pascal Naming Convention**(1st letter in the word should be upper ) in contrast java uses camel Naming Convention, which means 1st letter of every word should be uppercase except for the first word(**main()**).

#### 4. Installation of Maven

Browser → Maven download

Apache Maven  
https://maven.apache.org › download

[Download Apache Maven – Maven](#)

Previous Stable 3.8.x Release. Apache Maven 3.8.8 is the previous stable minor release for all users. ...  
Maven is distributed in several formats for your ...

**Installation Instructions**  
What is Maven? Features · Download · Use. Install; Run ...

**3.3**  
Current releases can be found on our download server. Icon ...

**Release Notes [-] Maven 3.8.8**  
Built by Maven. Release Notes – Maven 3.8.8. The Apache ...

**Release Notes**  
Download · Welcome · License; About Maven; What is Maven ...

**Index of /maven**  
Index of /maven. Icon Name Last modified Size Description ...

More results from apache.org »

Apache Maven  
https://maven.apache.org

Welcome to Apache Maven – Maven

https://maven.apache.org/download.cgi wnload, Install, Configure, Run Maven, Maven Plugins and Maven Extensions ... Improve the

79°F Partly cloudy Search ENG US 8:35 PM 27-02-2025

**Downloading Apache Maven 3.9.9**  
Apache Maven 3.9.9 is the latest release: it is the recommended version for all users.

**System Requirements**

| Java Development Kit (JDK) | Maven 3.9+ requires JDK 8 or above to execute. It still allows you to build against 1.3 and other JDK versions                                                                                   |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Memory                     | No minimum requirement                                                                                                                                                                           |
| Disk                       | Approximately 10MB is required for the Maven installation itself. In addition to that, disk space will be used for your local repository will vary depending on usage but expect at least 500MB. |
| Operating System           | No minimum requirement. Start up scripts are included as shell scripts (tested on many Unix flavors) and Windows bat files                                                                       |

**Files**

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [Installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

| Link                  | Checksums                     | Signature                            |                                   |
|-----------------------|-------------------------------|--------------------------------------|-----------------------------------|
| Binary tar.gz archive | apache-maven-3.9.9-bin.tar.gz | apache-maven-3.9.9-bin.tar.gz.sha512 | apache-maven-3.9.9-bin.tar.gz.asc |
| Binary zip archive    | apache-maven-3.9.9-bin.zip    | apache-maven-3.9.9-bin.zip.sha512    | apache-maven-3.9.9-bin.zip.asc    |
| Source tar.gz archive | apache-maven-3.9.9-src.tar.gz | apache-maven-3.9.9-src.tar.gz.sha512 | apache-maven-3.9.9-src.tar.gz.asc |
| Source zip archive    | apache-maven-3.9.9-src.zip    | apache-maven-3.9.9-src.zip.sha512    | apache-maven-3.9.9-src.zip.asc    |

- 3.9.9 [Release Notes](#) and [Release Reference Documentation](#)
- [latest source code from source repository](#)
- Distributed under the [Apache License, version 2.0](#)
- other:
  - All current release sources (plugins, shared libraries,...) available at <https://downloads.apache.org/maven/>

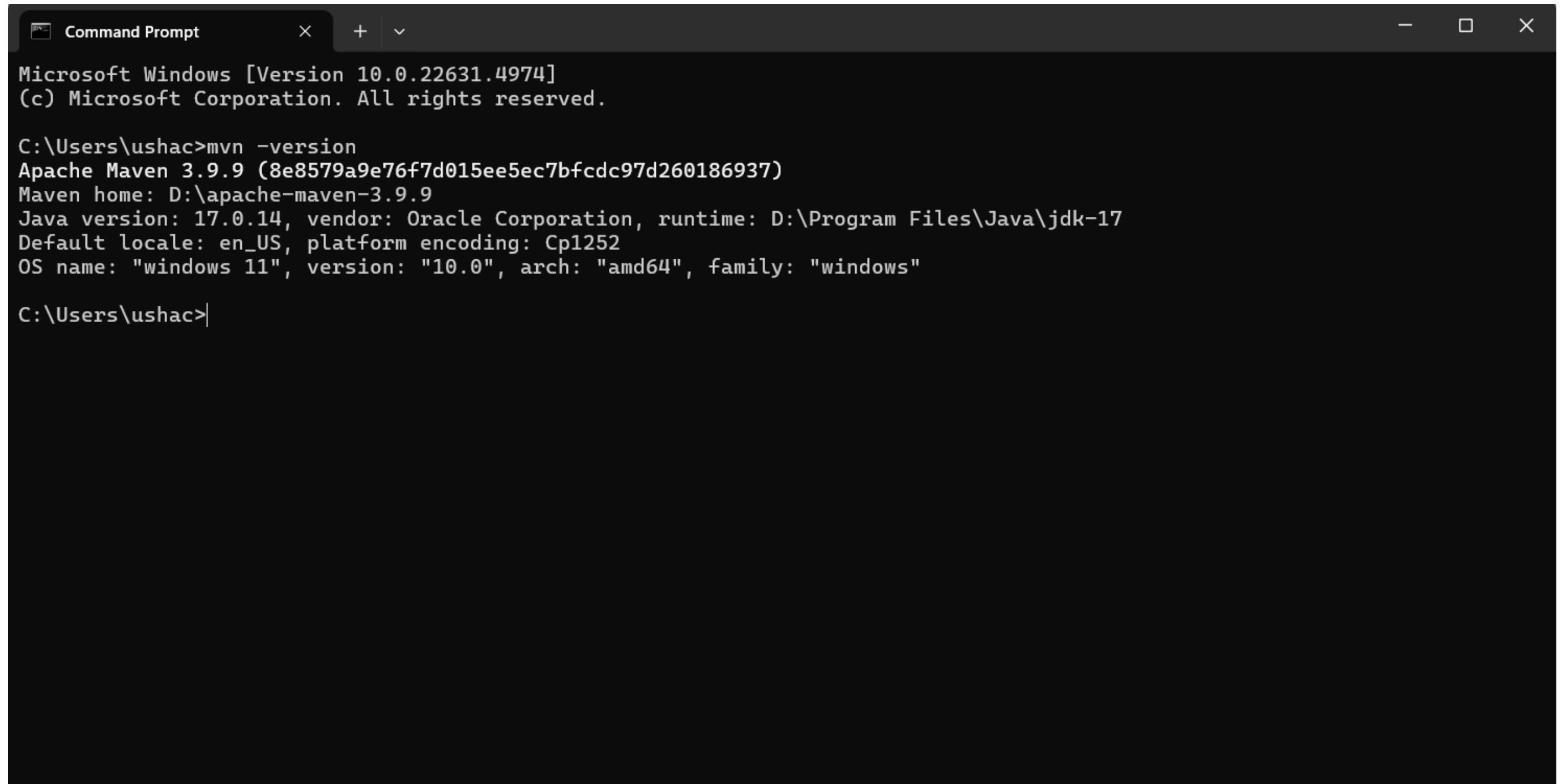
Recent download history

- apache-maven-3.9.9-bin (1).z
- chromedriver-win64 (1).zip
- selenium-server-4.29.0 (1).jar

Full download history

79°F Partly cloudy Search ENG US 8:37 PM 27-02-2025

After installing, check in cmd → mvn -version



```
Command Prompt Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ushac>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfc97d260186937)
Maven home: D:\apache-maven-3.9.9
Java version: 17.0.14, vendor: Oracle Corporation, runtime: D:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\ushac>
```

## 5. Installation of Gradle

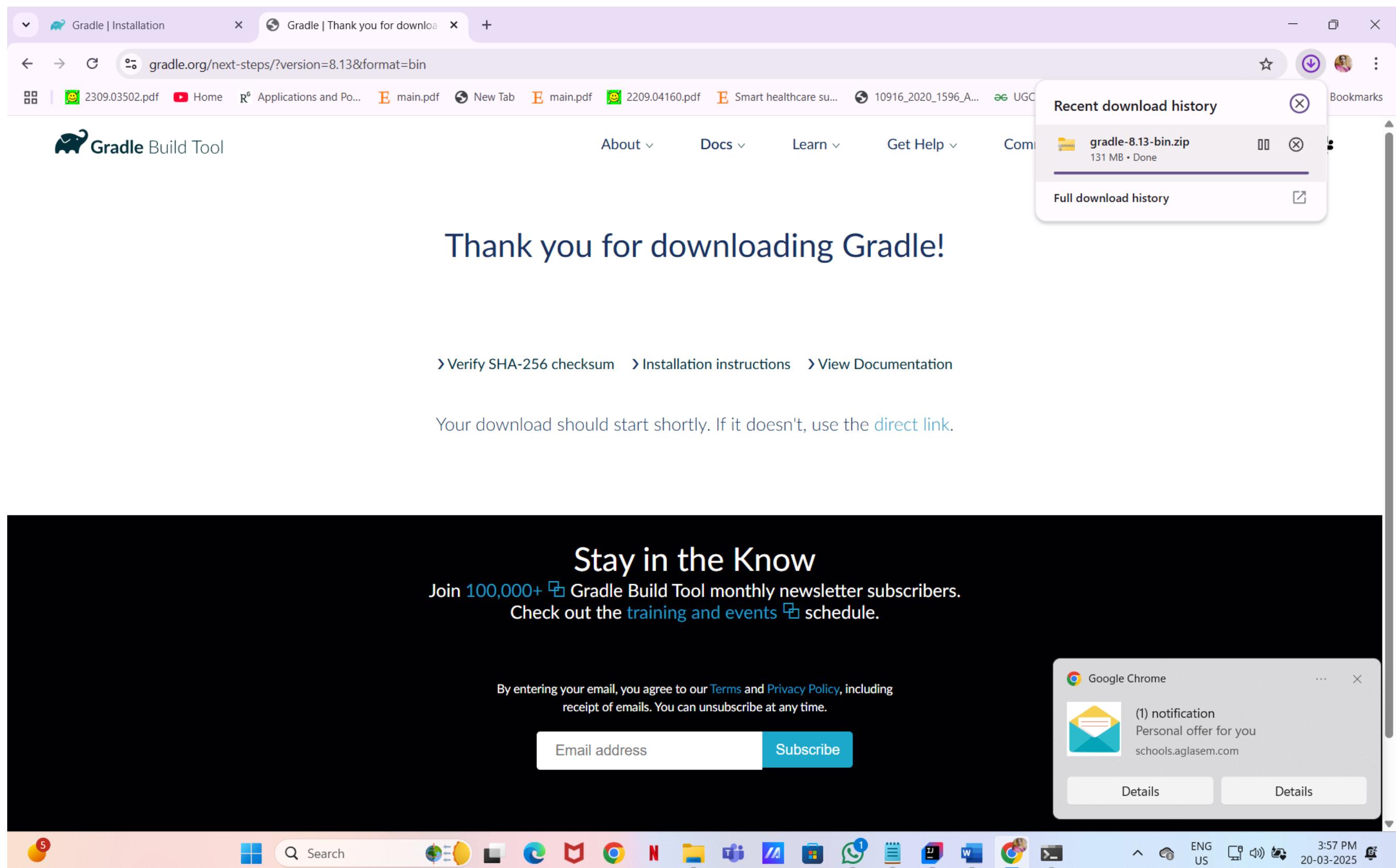
### Step 1. [Download](#) the latest Gradle distribution

The current Gradle release is version 8.13, released on 25 Feb 2025. The distribution zip file comes in two flavors:

- [Binary-only](#)
- [Complete](#), with docs and sources

If in doubt, choose the binary-only version and browse [docs](#) and [sources](#) online.

Need to work with an older version? See the [releases page](#).



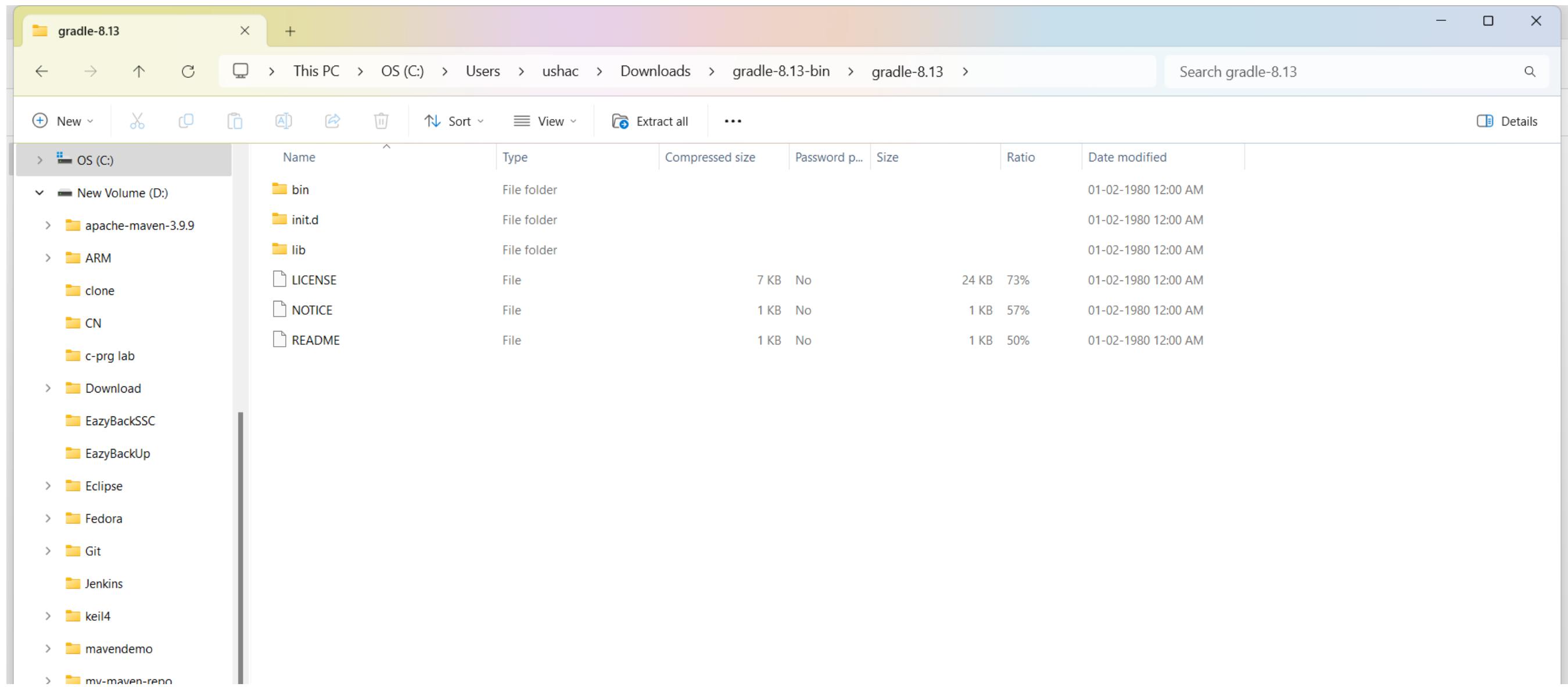
### Step 2. Unpack the distribution

Microsoft Windows users

Create a new directory C:\Gradle with File Explorer.

Open a second File Explorer window and go to the directory where the Gradle distribution was downloaded. Double-click the ZIP archive to expose the content. Drag the content folder gradle-8.13 to your newly created C:\Gradle folder.

Alternatively you can unpack the Gradle distribution ZIP into C:\Gradle using an archiver tool of your choice.

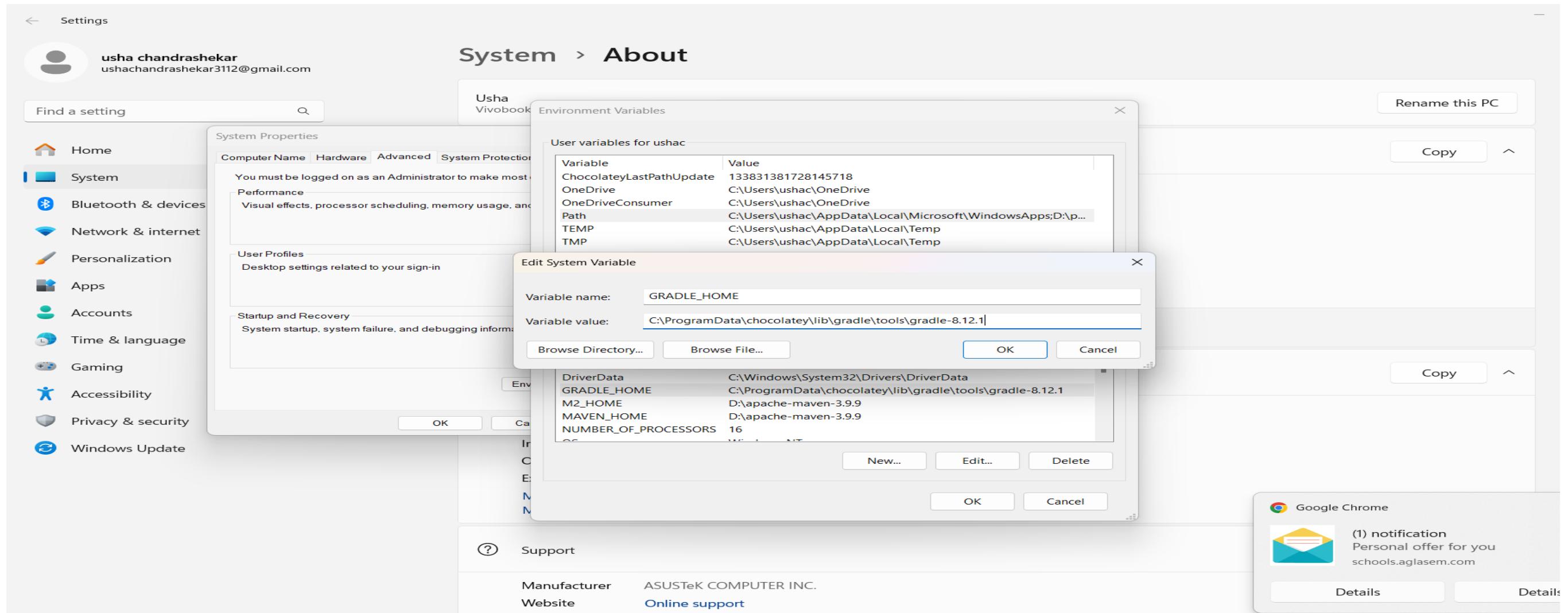


### Step 3. Configure your system environment

Microsoft Windows users

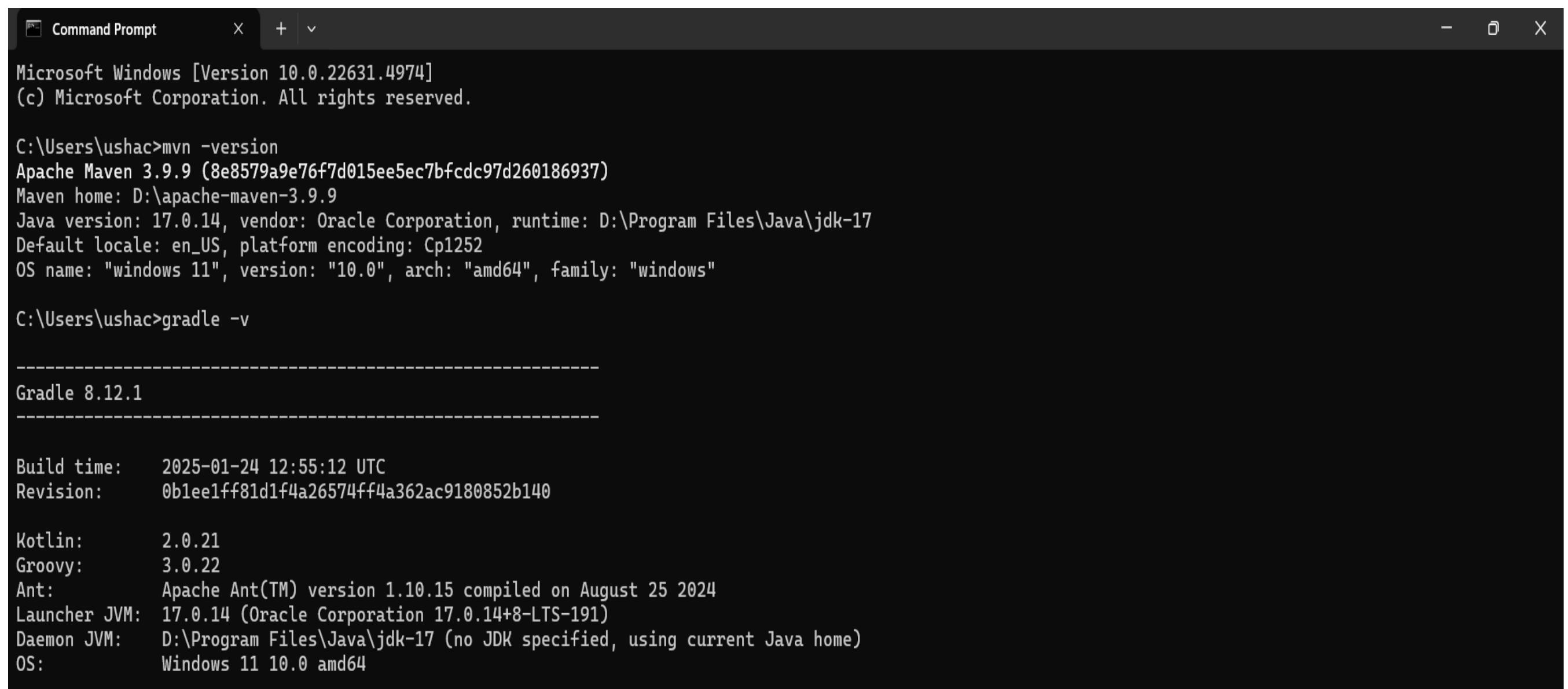
In File Explorer right-click on the This PC (or Computer) icon, then click Properties -> Advanced System Settings -> Environmental Variables.

Under System Variables select Path, then click Edit. Add an entry for C:\Gradle\gradle-8.13\bin.  
Click OK to save.



## Step 4. Verify your installation

Open a console (or a Windows command prompt) and run gradle -v to run gradle and display the version.



```

Command Prompt X +
Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ushac>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfc97d260186937)
Maven home: D:\apache-maven-3.9.9
Java version: 17.0.14, vendor: Oracle Corporation, runtime: D:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\ushac>gradle -v

Gradle 8.12.1

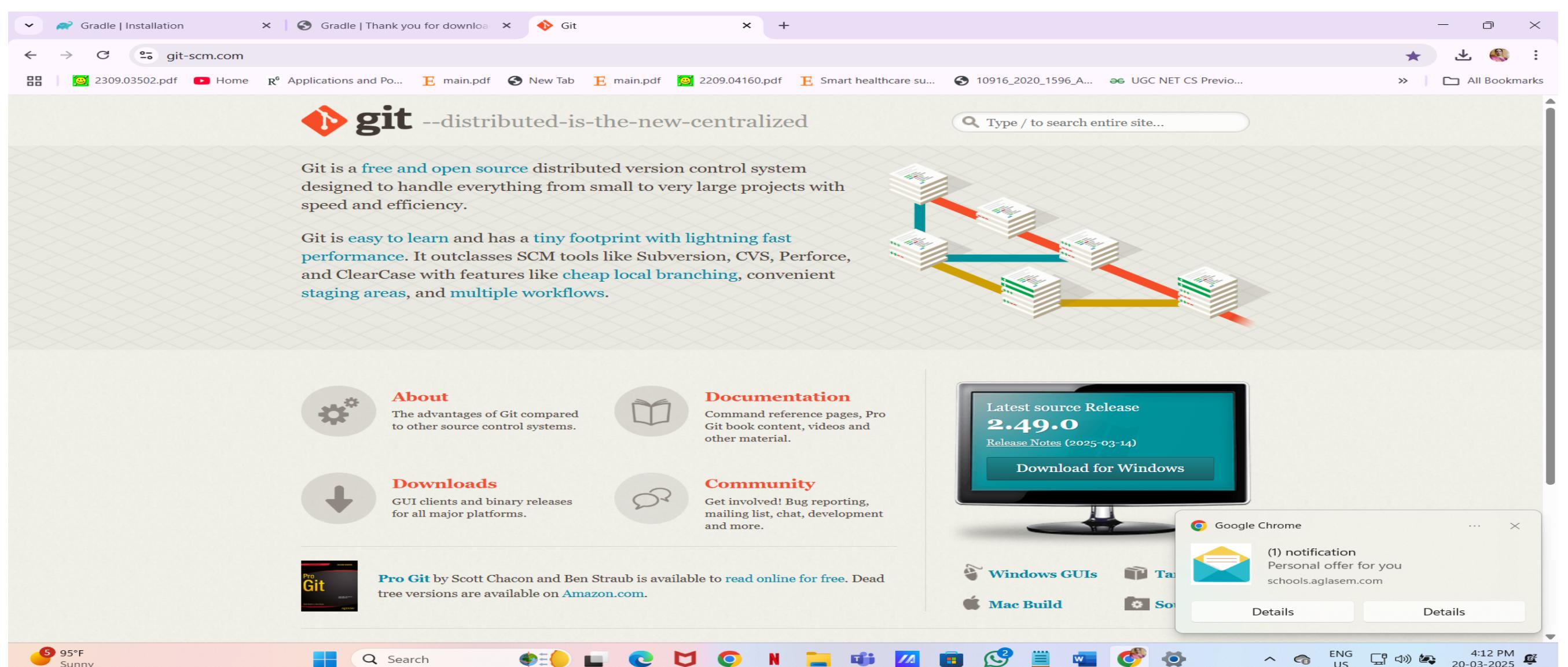
Build time: 2025-01-24 12:55:12 UTC
Revision: 0b1ee1ff81d1f4a26574ff4a362ac9180852b140

Kotlin: 2.0.21
Groovy: 3.0.22
Ant: Apache Ant(TM) version 1.10.15 compiled on August 25 2024
Launcher JVM: 17.0.14 (Oracle Corporation 17.0.14+8-LTS-191)
Daemon JVM: D:\Program Files\Java\jdk-17 (no JDK specified, using current Java home)
OS: Windows 11 10.0 amd64

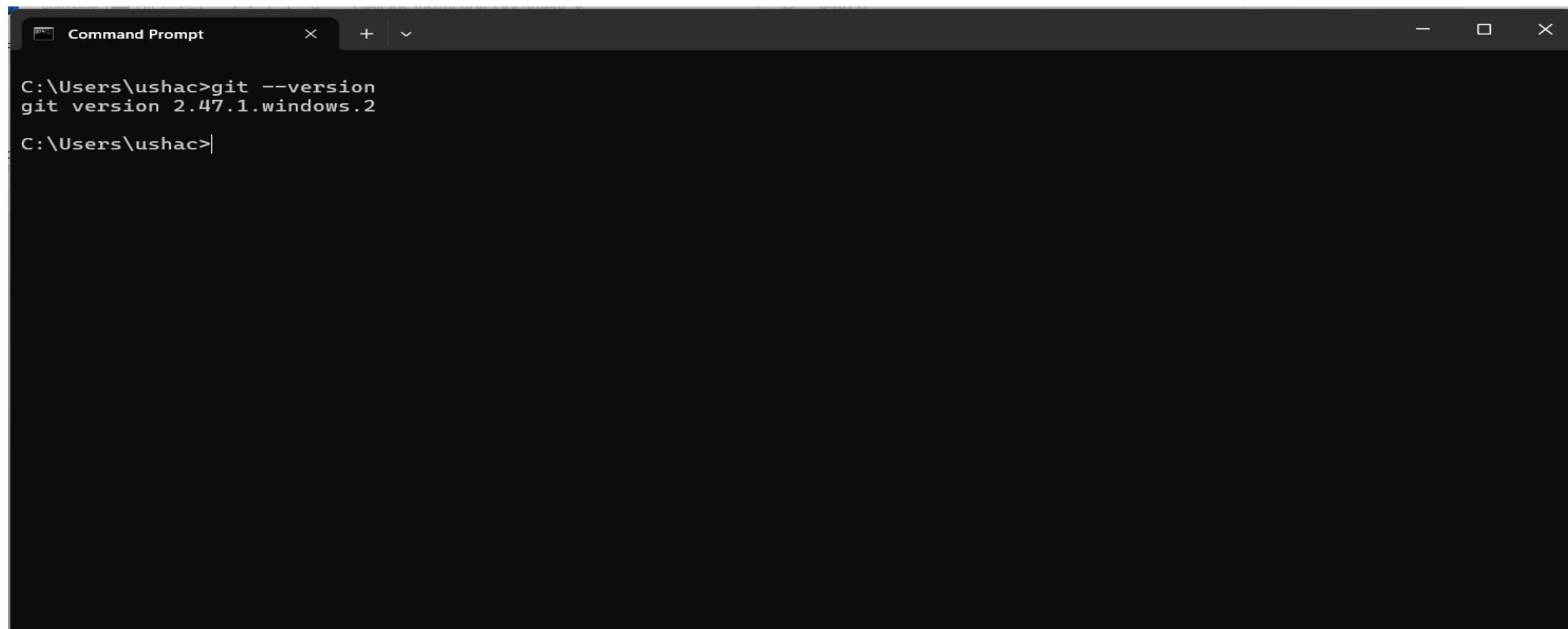
```

## 6. Installation of GIT

Download GIT from [git-scm.com](https://git-scm.com) and install.



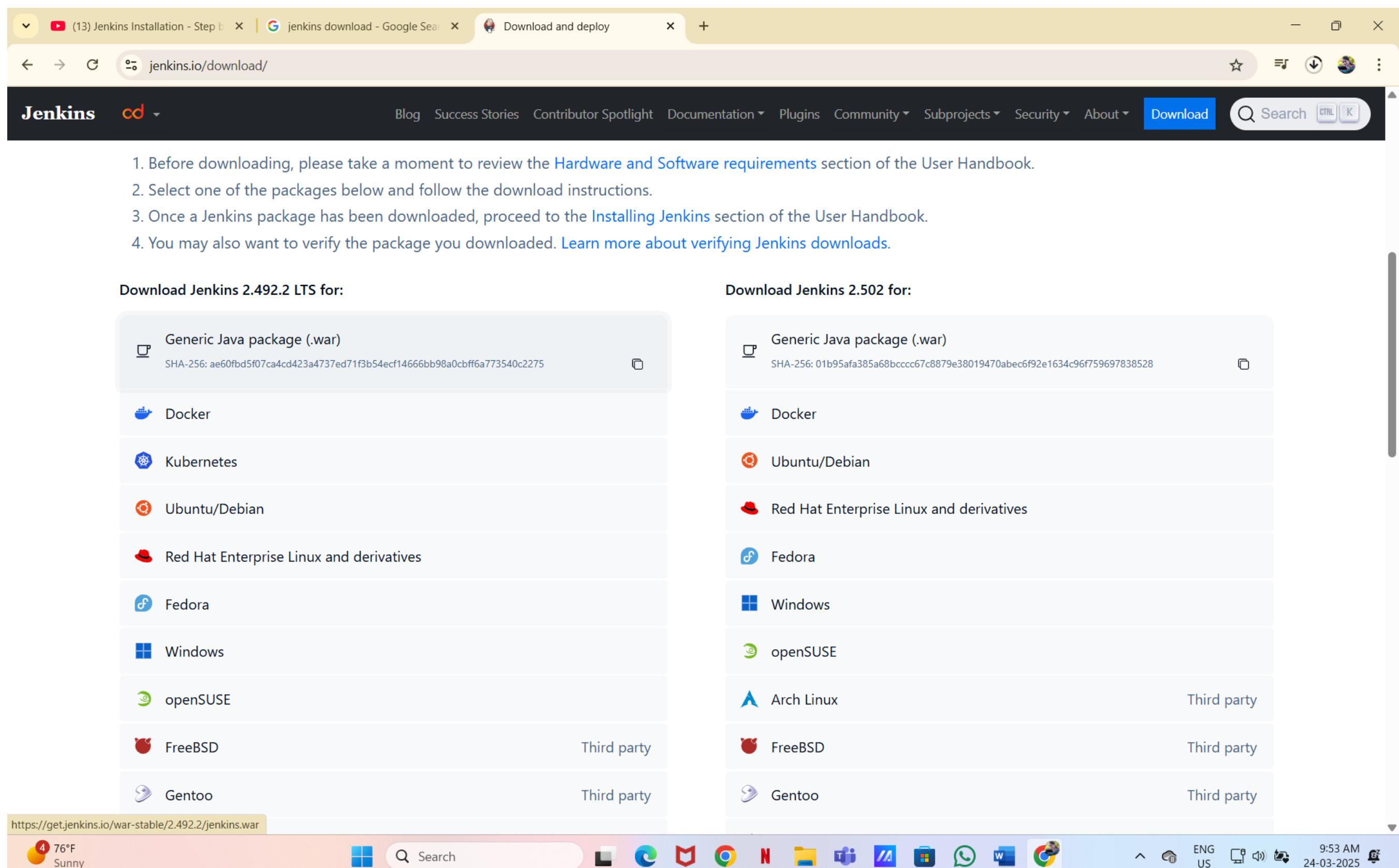
## Check git --version in cmd



```
C:\Users\ushac>git --version
git version 2.47.1.windows.2
C:\Users\ushac>
```

## 6. Installation of Jenkins

Goto Browser → jenkins.io → Download → Select Generic java package(.war), as shown ---



1. Before downloading, please take a moment to review the [Hardware and Software requirements](#) section of the User Handbook.

2. Select one of the packages below and follow the download instructions.

3. Once a Jenkins package has been downloaded, proceed to the [Installing Jenkins](#) section of the User Handbook.

4. You may also want to verify the package you downloaded. [Learn more about verifying Jenkins downloads.](#)

**Download Jenkins 2.492.2 LTS for:**

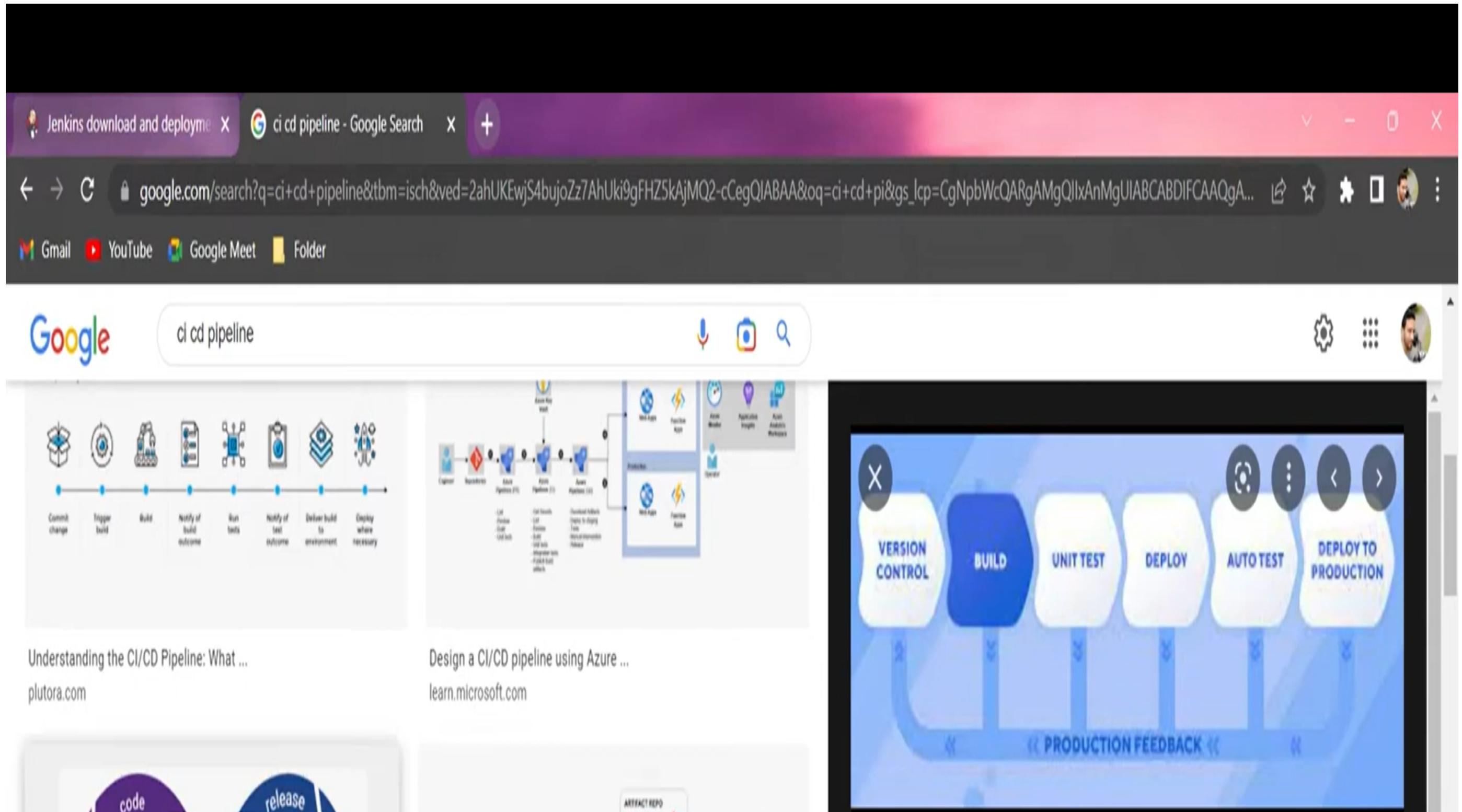
- Generic Java package (.war)
- Docker
- Kubernetes
- Ubuntu/Debian
- Red Hat Enterprise Linux and derivatives
- Fedora
- Windows
- openSUSE
- FreeBSD
- Gentoo

**Download Jenkins 2.502 for:**

- Generic Java package (.war)
- Docker
- Ubuntu/Debian
- Red Hat Enterprise Linux and derivatives
- Fedora
- Windows
- openSUSE
- Arch Linux
- FreeBSD
- Gentoo

<https://get.jenkins.io/war-stable/2.492.2/jenkins.war>

## Create new folder and save the war file.



Go to cmd → type – `java -jar jenkins.war`

```
C:\Windows\System32\cmd.e Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

D:\Jenkins>java -jar jenkins.war
Running from: D:\Jenkins\jenkins.war
webroot: C:\Users\ushac\.jenkins\war
2025-03-24 06:14:16.033+0000 [id=1] INFO winstone.Logger#logInternal: Beginning extraction from war file
2025-03-24 06:14:16.080+0000 [id=1] WARNING o.e.j.ee9.nested.ContextHandler#setContextPath: Empty contextPath
2025-03-24 06:14:16.120+0000 [id=1] INFO org.eclipse.jetty.server.Server#doStart: jetty-12.0.16; built: 2024-12-09T21:02:54.535Z; git: c3f88bafb4e393
f23204dc14dc57b042e84debc7; jvm 17.0.14+8-LTS-191
2025-03-24 06:14:16.553+0000 [id=1] INFO o.e.j.e.w.StandardDescriptorProcessor#visitServlet: NO JSP Support for /, did not find org.eclipse.jetty.ee9
.jsp.JettyJspServlet
2025-03-24 06:14:16.593+0000 [id=1] INFO o.e.j.s.DefaultSessionIdManager#doStart: Session workerName=node0
2025-03-24 06:14:16.962+0000 [id=1] INFO hudson.WebAppMain#contextInitialized: Jenkins home directory: C:\Users\ushac\.jenkins found at: $user.home/.jenkins
2025-03-24 06:14:17.037+0000 [id=1] INFO o.e.j.s.handler.ContextHandler#doStart: Started oeje9n.ContextHandler$CoreContextHandler@6e9c413e{Jenkins v2
.479.3, /, b=file:///C:/Users/ushac/.jenkins/war/, a=AVAILABLE, h=oeje9n.ContextHandler$CoreContextHandler$CoreToNestedHandler@57a4d5ee{STARTED}}
2025-03-24 06:14:17.054+0000 [id=1] INFO o.e.j.server.AbstractConnector#doStart: Started ServerConnector@1bb9aa43{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
2025-03-24 06:14:17.064+0000 [id=1] INFO org.eclipse.jetty.server.Server#doStart: Started oejs.Server@7ed7259e{STARTING}[12.0.16,sto=0] @1454ms
2025-03-24 06:14:17.064+0000 [id=36] INFO winstone.Logger#logInternal: Winstone Servlet Engine running: controlPort=disabled
2025-03-24 06:14:17.207+0000 [id=42] INFO jenkins.InitReactorRunner$1#onAttained: Started initialization
2025-03-24 06:14:17.347+0000 [id=43] INFO jenkins.InitReactorRunner$1#onAttained: Listed all plugins
2025-03-24 06:14:19.535+0000 [id=55] INFO jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2025-03-24 06:14:19.555+0000 [id=63] INFO jenkins.InitReactorRunner$1#onAttained: Started all plugins
2025-03-24 06:14:19.557+0000 [id=63] INFO jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2025-03-24 06:14:19.848+0000 [id=71] INFO h.p.b.g.GlobalTimeOutConfiguration#load: global timeout not set
2025-03-24 06:14:20.550+0000 [id=64] INFO jenkins.InitReactorRunner$1#onAttained: System config loaded
2025-03-24 06:14:20.555+0000 [id=64] INFO jenkins.InitReactorRunner$1#onAttained: System config adapted
2025-03-24 06:14:20.589+0000 [id=66] INFO jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2025-03-24 06:14:20.604+0000 [id=62] INFO jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs updated
2025-03-24 06:14:20.615+0000 [id=86] INFO hudson.util.Retrier#start: Attempt #1 to do the action check updates server
2025-03-24 06:14:20.623+0000 [id=65] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
2025-03-24 06:14:20.648+0000 [id=34] INFO hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
```

Jenkins default port number is <http://localhost:8080>

## For the first time installing Jenkins remember the Checksum

```
2022-11-07 14:31:53.186+0000 [id=87] INFO hudson.model.AsyncPeriodicWork#lambda$doRun$1: Started Download metadata
2022-11-07 14:31:53.196+0000 [id=87] INFO hudson.util.Retrier#start: Attempt #1 to do the action check updates server
2022-11-07 14:31:53.521+0000 [id=54] INFO jenkins.install.SetupWizard#init:

***** Jenkins initial setup is required. An admin user has been created and a password generated
***** Please use the following password to proceed to installation:

3c1e7ff37b0242aa979a9c1beef69edac
```

**Open browser type localhost:8080, paste the checksum in Administrator password.**

## 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:\Users\Saurav\.jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Continue

## Select the install suggested plugins



**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: Stage View |
| ○ Git         | ○ SSH Build Agents       | ○ Matrix Authorization Strategy     | ○ PAM Authentication   |
| ○ LDAP        | ○ Email Extension        | ✓ Mailer                            |                        |

```

-- Display URL API
-- Pipeline: Supporting APIs
-- Checks API
-- ...
-- Matrix Project
-- Resource Disposer
Workspace Cleanup
Ant
 -- Apache HttpComponents Client
 4.x API
 - Oracle Java SE Development Kit
 Installer
 -- Command Agent Launcher
 -- JavaScript GUI Lib: ACE Editor
bundle
 -- Pipeline: SCM Step
 -- Pipeline: Groovy
 -- Durable Task
 -- Pipeline: Nodes and Processes
 -- Pipeline: Job
 -- Jakarta Activation API
 -- Jakarta Mail API
Mailer
 -- Pipeline: Basic Steps
Gradle
 -- - required dependency

```

Jenkins 2.361.3

**Getting Started**

## Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Skip and continue as admin Save and Continue

Jenkins 2.361.3

**Getting Started**

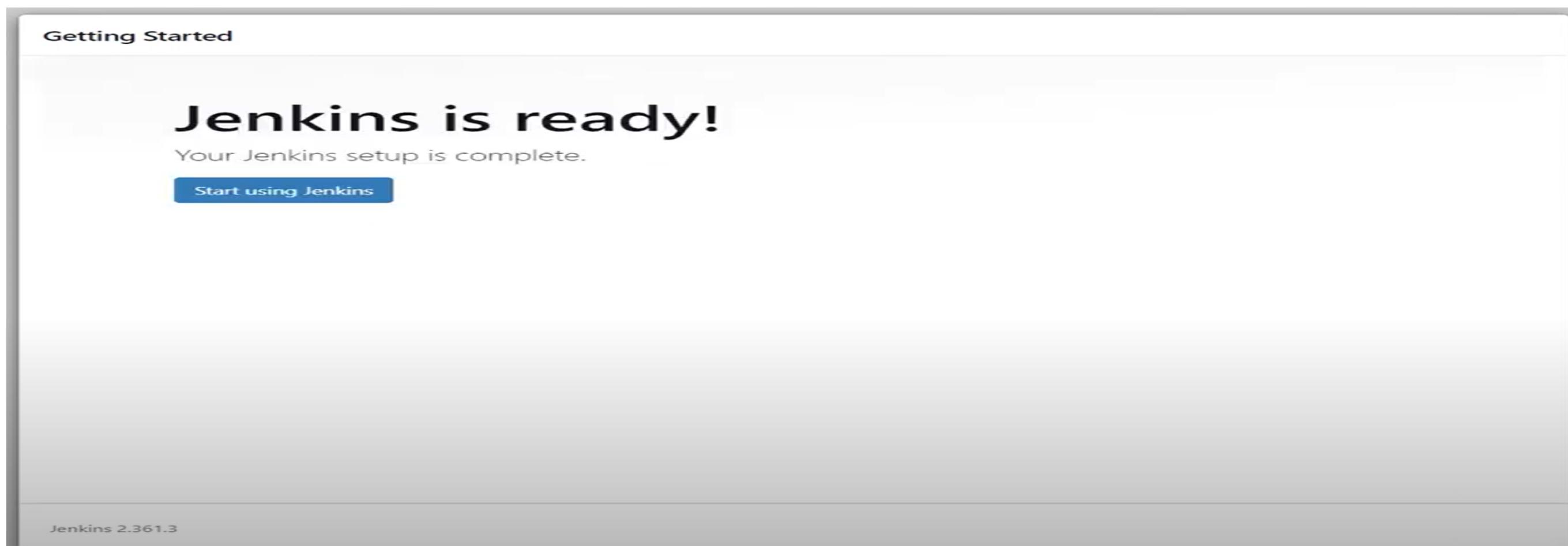
## Instance Configuration

Jenkins URL:

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 2.361.3 Not now Save and Finish



## Dashboard of Jenkins

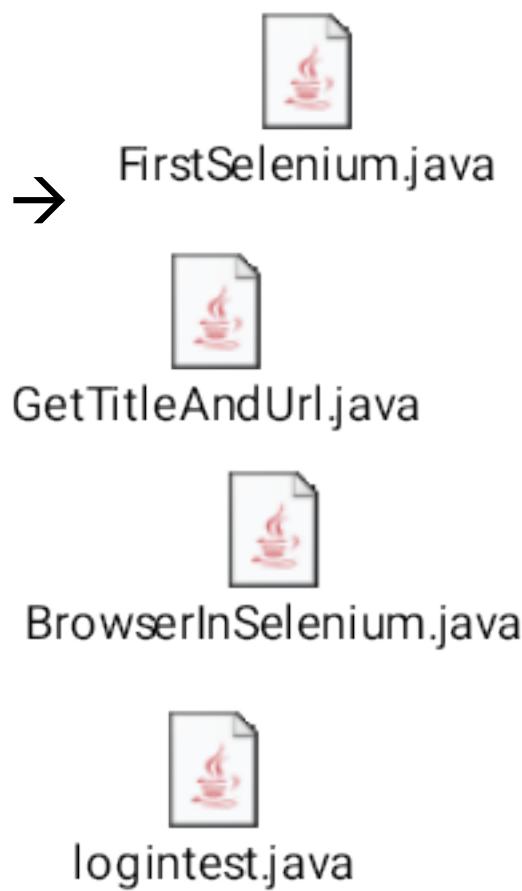
The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information for 'Saurav Ronnie'. Below the navigation bar, the dashboard has several sections: 'Welcome to Jenkins!', 'Start building your software project', 'Build Queue' (which is empty), 'Build Executor Status' (showing 1 Idle and 2 Idle executors), and 'Set up a distributed build' (with links to 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'). At the bottom right, there are links for 'REST API' and 'Jenkins 2.361.3'.

# CODING

**Build and run a Java application using Selenium - manual approach, create a Java application with Maven and migrate the same application to Gradle. Understanding the POM file, Dependency management and plugins. understanding build scripts in Gradle (Groovy and Kotlin DSL), Dependency management and Task automation.**

File → New Project → Selenium AutomationwithJava

Src → New Package → com.example → New Java class →



File → New Project → Selenium AutomationwithJava → right click → new Directory → Lib and drivers (paste the jar and chrome drivers file in directories respectively)

## FirstSelenium.java

```
package com.example;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class FirstSelenium {
 static WebDriver driver;

 public static void main(String[] args) {
 System.setProperty("webdriver.chrome.driver","./Drivers/chromedriver.exe");
 driver =new ChromeDriver();
 driver.get("https://www.google.com");
 }
}
```

```
//driver.navigate().to("https://www.google.com");
driver.quit();
}

}
```

### GetTitleAndUrl.java

```
package com.example;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class GetTitleAndUrl {
 static WebDriver driver;

 public static void main(String[] args) {
 System.setProperty("webdriver.chrome.driver", "./Drivers/chromedriver.exe");
 driver = new ChromeDriver();
 driver.get("https://www.google.com");
 driver.navigate().to("https://www.bietdvg.edu/");
 String url = driver.getCurrentUrl();
 String title = driver.getTitle();
 System.out.println("The url is: " + url);
 System.out.println("The title is: " + title);
 driver.quit();
 }
}
```

### BrowserInSelenium.java

```
package com.example;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BrowserInSelenium {
 static WebDriver driver;
```

```

public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", "./Drivers/chromedriver.exe");
 driver = new ChromeDriver();
 driver.navigate().to("https://www.google.com/");
 Thread.sleep(2000); // Hard wait not recommended
 driver.navigate().to("https://www.selenium.dev/");
 Thread.sleep(2000);
 driver.navigate().to("https://www.saucedemo.com/");
 Thread.sleep(2000);
 driver.navigate().back();
 Thread.sleep(2000);
 driver.navigate().forward();
 driver.findElement(By.name("user-name")).sendKeys("Usha");
 Thread.sleep(2000);
 driver.navigate().refresh();
 Thread.sleep(1000);
 driver.close();
}
}

```

**loginTest.java**

```

package com.example;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class logintest {
 public static void main(String[] args) throws InterruptedException {
 WebDriver driver = new ChromeDriver();
 driver.get("https://www.saucedemo.com/");
 driver.manage().window().maximize();
 Thread.sleep(2000);
 driver.findElement(By.id("user-name")).sendKeys("standard_user");
 Thread.sleep(2000);
 driver.findElement(By.id("password")).sendKeys("secret_sauce");
 Thread.sleep(2000);
 driver.findElement(By.id("login-button")).click();
 Thread.sleep(2000);
 driver.quit();
 }
}

```

```

 }
}

```

**Note:** Sauce Demo is a sample website created by Sauce Labs, a cloud-based testing platform, designed to allow users to practice browser automation and test their code against a variety of browsers and operating systems

```

package com.example;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

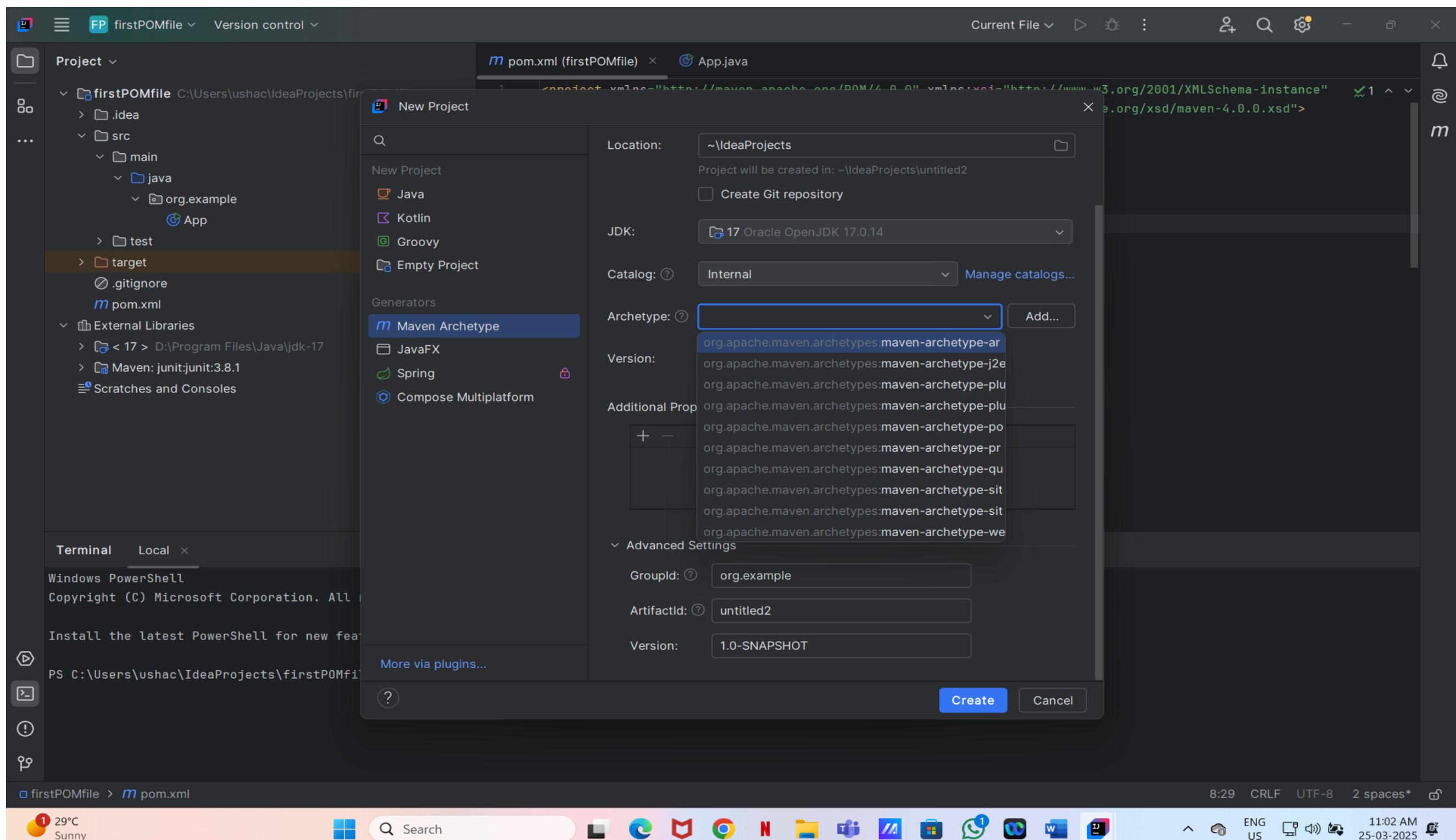
public class FirstSelenium {
 static WebDriver driver; 3 usages

 public static void main(String[] args) {
 System.setProperty("webdriver.chrome.driver", "./Drivers/chromedriver.exe");
 driver = new ChromeDriver();
 driver.get("https://www.google.com");
 //driver.navigate().to("https://www.google.com");
 driver.quit();
 }
}

```

## MAVEN PROJECT

File → new project → maven archetype → firstPOMfile → select maven-archetype-QuickStart → create



## Pom.xml(firstPOMfile)

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>

 <groupId>org.example</groupId>
 <artifactId>firstPOMfile</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>firstPOMfile</name>
 <url>http://maven.apache.org</url>

 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>

 <dependencies>

```

```
<dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
</dependency>
</dependencies>
<build>
 <plugins>
 <!-- Compiler Plugin -->
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.1</version>
 <configuration>
 <source>1.8</source>
 <target>1.8</target>
 </configuration>
 </plugin>

 <!-- Jar Plugin -->
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-jar-plugin</artifactId>
 <version>3.2.0</version>
 <configuration>
 <archive>
 <manifest>
 <mainClass>org.example.App</mainClass>
 </manifest>
 </archive>
 </configuration>
 </plugin>
 </plugins>
</build>
</project>
```

**In Terminal execute these commands**

**mvn clean compile**  
**mvn package**

```
java -jar .\target\firstPOMfile-1.0-SNAPSHOT.jar
```

## Output

Hello world!

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>org.example</groupId>
 <artifactId>firstPOMfile</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>
 <name>firstPOMfile</name>
 <url>http://maven.apache.org</url>
 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>
 <dependencies>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>
 </dependencies>
</project>

```

**File → new project → SeleniumMaven → new directory → resources → 3files → index.html, logo.png, style.css (paste)**

## POM.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>seleniuminmaven</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

```

```
<name>seleniuminmaven</name>
<url>http://maven.apache.org</url>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>

 <dependency>
 <groupId>org.seleniumhq.selenium</groupId>
 <artifactId>selenium-java</artifactId>
 <version>4.29.0</version>
 </dependency>

 <dependency>
 <groupId>org.testng</groupId>
 <artifactId>testng</artifactId>
 <version>7.11.0</version>
 <scope>test</scope>
 </dependency>
</dependencies>
<build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-resources-plugin</artifactId>
 <version>3.2.0</version>
 <executions>
 <execution>
 <phase>prepare-package</phase>
 <goals>
 <goal>copy-resources</goal>
 </goals>
 </execution>
 </executions>
 </plugin>
 </plugins>

```

```

<configuration>
 <outputDirectory>${project.basedir}/docs</outputDirectory>
 <!-- Deploy to /docs folder -->
<resources>
 <resource>
 <directory>src/main/resources</directory>
 <includes>
 <include>/*</include> <!-- Copy all files in src/main/resources -->
 </includes>
 </resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

## Index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Computer Science and Engineering(Data Science)</title>
 <link rel="stylesheet" href="styles.css">
 <link rel="icon" type="image/x-icon" href="images/favicon.ico">
</head>
<body>

<!-- Header Section -->
<header>
 <div class="container">
 <h1>Vishveshwariya Technological University</h1>

 <nav>
 Courses
 Address

```

```
</nav>
</div>
</header>

<!-- Content Section -->
<div class="content">
 <!-- Services Section -->
 <section id="Courses">
 <h2>Courses</h2>
 <p>
 Bachelor of Engineering, Computer Applications and IT, Sciences, Management and
 Business Administration, Animation and Design.....
 </p>
 <div class="Courses">
 <div class="Courses">
 <h3>UG</h3>
 <p>
 Bachelor of Engineering.
 </p>
 </div>
 <div class="Courses">
 <h3>PG</h3>
 <p>
 Post Graduate.
 </p>
 </div>
 </div>
 </section>
 <!-- Address Section -->
 <section id="address">
 <h2>Address Info</h2>
 <p>
 QFH7+497 Jnana Sangama, VTU Main Rd, Visvesvaraya Technological University,
 Machhe, Belagavi, Karnataka 590018

 Telephone: 0831 249 8100

 Email: registrar@vtu.ac.in.
 </p>
 </section>
</div>
```

```
</section>
</div>
```

## Style.css

```
/* General Styles */
body {
 font-family: Arial, sans-serif;
 margin: 0;
 padding: 0;
 background-color: #f4f4f9;
 color: #333;
}
```

```
/* Header Styles */
header {
 background-color: #004c99;
 color: white;
 padding: 20px 0;
 text-align: center;
 position: sticky;
 top: 0;
 z-index: 1000;
}
header .container {
 max-width: 1200px;
 margin: 0 auto;
 text-align: center;
}
header h1 {
 margin: 10px 0;
 font-size: 2.5rem;
}
header img {
 max-width: 250px;
 margin: 10px 0;
 border-radius: 05px;
}
header nav {
 margin-top: 15px;
}
```

```
header nav a {
 color: white;
 margin: 0 15px;
 text-decoration: none;
 font-weight: bold;
 font-size: 1rem;
}
header nav a:hover {
 text-decoration: underline;
}

/* Content Section */
.content {
 max-width: 1200px;
 margin: 20px auto;
 padding: 20px;
 background-color: white;
 border-radius: 8px;
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.content section {
 margin-bottom: 30px;
}
.content section h2 {
 color: #004c99;
 border-bottom: 3px solid #004c99;
 display: inline-block;
 padding-bottom: 5px;
 font-size: 1.8rem;
 margin-bottom: 15px;
}
.content section p {
 line-height: 1.6;
 font-size: 1rem;
 text-align: justify;
}

/* Services Section */
.services {
 display: flex;
 flex-wrap: wrap;
```

```
gap: 20px;
}

.service {
 flex: 1 1 calc(33.333% - 20px);
 background-color: #eaf3ff;
 padding: 15px;
 border-radius: 8px;
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.service h3 {
 color: #004c99;
 font-size: 1.3rem;
 margin-bottom: 10px;
}

.service p {
 font-size: 0.95rem;
}

/* Address Section */

#address a {
 color: #004c99;
 text-decoration: none;
 font-weight: bold;
}

#address a:hover {
 text-decoration: underline;
}

/* Footer Styles */

footer {
 background-color: #004c99;
 color: white;
 text-align: center;
 padding: 15px 0;
 font-size: 0.9rem;
}

footer a {
 color: white;
 text-decoration: none;
 font-weight: bold;
}
```

```
footer a:hover {
 text-decoration: underline;
}

/* Responsive Styles */
@media (max-width: 768px) {
 .services {
 flex-direction: column;
 }
 .service {
 flex: 1 1 100%;
 }
 header nav a {
 display: block;
 margin: 5px 0;
 }
}
```

## Logo.png



**Vishveshwarya Technological University**

Courses   Address

Bachelor of Engineering, Computer Applications and IT, Sciences, Management and Business Administration, Animation and Design.....

**UG**  
Bachelor of Engineering.

**PG**  
Post Graduate.

**Address Info**

**Right click on project → open in → explorer → open seleniummaven folder → right click → open git bash here**

**git init**

**git status**

**git add .**

**git commit -m " selenium test in maven"**

**git status**

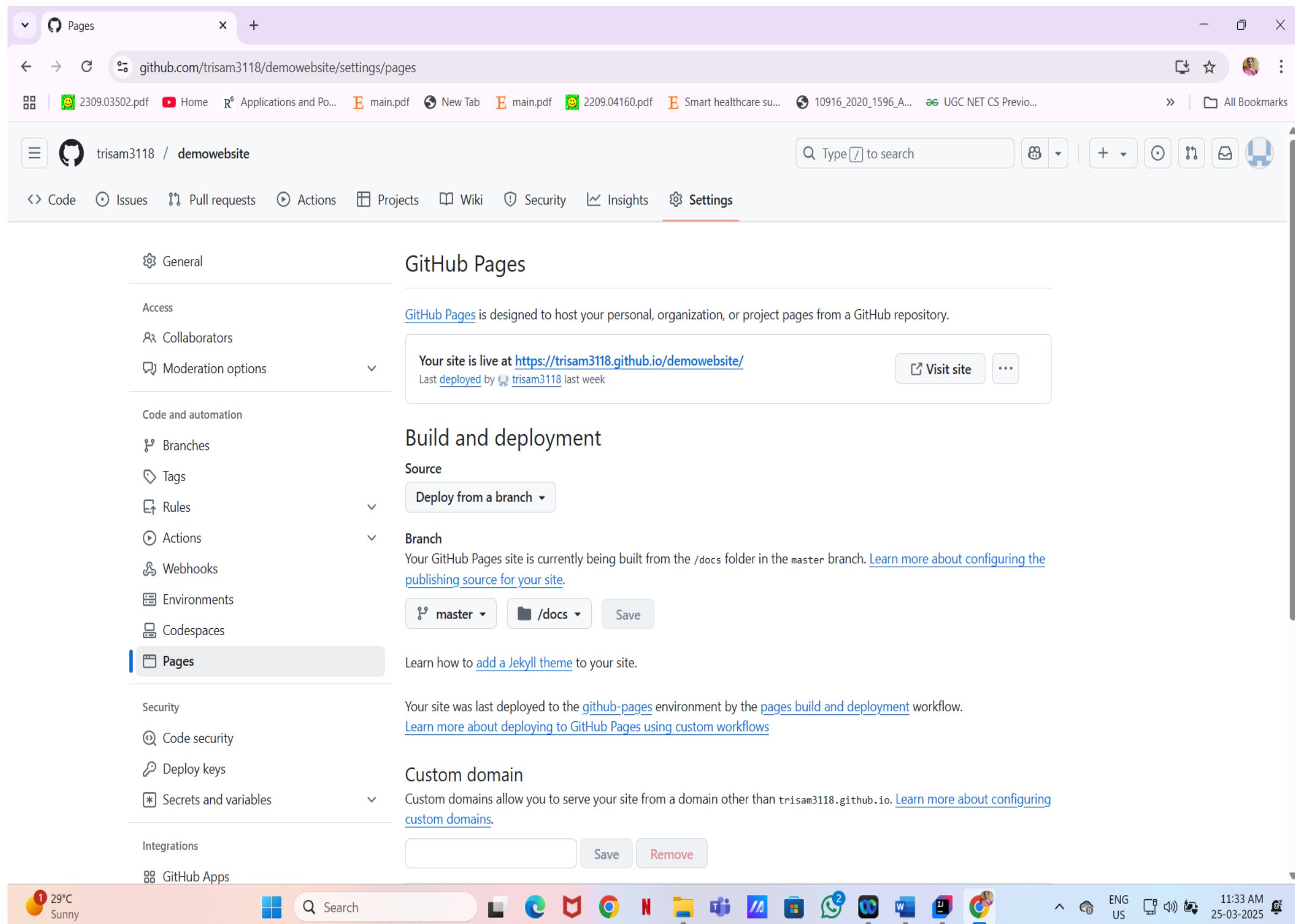
**git remote add origin URL**

**git push -u origin master**

**In GITHUB → SETTINGS → PAGES → SOURCE (DEPLOY FROM A BRANCH)**

**BRANCH → MASTER**

**ROOT →/docs → save**



## Migrate Maven project to Gradle

### POM.XML

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>FinalMavenToGradle</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>FinalMavenToGradle</name>
 <url>http://maven.apache.org</url>

<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>
</dependencies>
<build>
 <plugins>
 <!-- Compiler Plugin -->
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.1</version>
 <configuration>
 <source>1.8</source>
 <target>1.8</target>
 </configuration>
 </plugin>
 <!-- Jar Plugin -->
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-jar-plugin</artifactId>
 <version>3.2.0</version>
 <configuration>
 <archive>
 <manifest>
 <mainClass>com.example.App</mainClass>
 </manifest>
 </archive>
 </configuration>
 </plugin>
 </plugins>
</build>
</project>
```

## Build.gradle

---

```
plugins {
 id 'java'
}

group = 'com.example'
version = '1.0-SNAPSHOT'

repositories {
 mavenCentral()
}

dependencies {

 testImplementation 'junit:junit:4.13.2'
}
jar {
 manifest {
 attributes(
 'Main-Class': 'com.example.App'
)
 }
}
```

## App.java

```
package com.example;

/**
 * Hello world!
 *
 */
public class App
{
 public static void main(String[] args)
 {
 System.out.println("Hello from Gradle!");
 }
}
```

## In Terminal run these commands

1. mvn clean install
2. mvn package
3. java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar

Output- Hello from Gradle!

4. gradle init --type pom

5. Select build script DSL:

1: Kotlin

2: Groovy

Select 2

6. Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]

Yes

7. gradle clean build

8. java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar

Output – Hello from Gradle!

```

m pom.xml (FinalMavenToGradle) build.gradle
1 plugins {
2 id 'java'
3 }
4
5 group = 'com.example'
6 version = '1.0-SNAPSHOT'
7
8 repositories {
9 mavenCentral()
10 }
11
12 dependencies {
13 }
14 testImplementation 'junit:junit:4.13.2'
15 }
16 jar {
17 manifest {
18 attributes(
19 'Main-Class': 'com.example.App'
20)
21 }
22 }
23

```

Terminal

```

1 actionable task: 1 executed
Configuration cache entry stored.
PS C:\Users\ushac\IdeaProjects\FinalMavenToGradle> gradle clean build
Reusing configuration cache.

BUILD SUCCESSFUL in 721ms
5 actionable tasks: 2 executed, 3 from cache
Configuration cache entry reused.
PS C:\Users\ushac\IdeaProjects\FinalMavenToGradle> java -jar .\target\FinalMavenToGradle-1.0-SNAPSHOT.jar
Hello from Gradle!
PS C:\Users\ushac\IdeaProjects\FinalMavenToGradle>

```

88°F Sunny

## Gradle Project using Kotlin

**File → New → project → select Gradle → choose → Kotlin DSL (it will generate build.gradle.kts) → Name the project → set JDK 17 → FINISH**

### build.gradle.kts

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
 kotlin("jvm") version "1.8.10" // Use latest stable Kotlin version
 application
}

group = "org.example"
version = "1.0-SNAPSHOT"
repositories {
 mavenCentral()
}

dependencies {
 implementation(kotlin("stdlib")) // Kotlin Standard Library
 testImplementation("org.junit.jupiter:junit-jupiter-api:5.8.2")

 testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:5.8.2")
}

tasks.test {
 useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
 kotlinOptions.jvmTarget = "17" // Match with your JDK version
}
tasks.register<Jar>("fatJar") {
 archiveClassifier.set("all")
 duplicatesStrategy = DuplicatesStrategy.EXCLUDE
 manifest {
 attributes["Main-Class"] = "MainKt"
 }
}
```

```
}

from(configurations.runtimeClasspath.get()).map { if (it.isDirectory) it else zipTree(it) }
with(tasks.jar.get() as CopySpec)
}

tasks.register("hello") {
 doLast {
 println("Hello, Usha!")
 }
}

application {
 mainClass.set("MainKt") // Update this if using a package
}
```

### Main.kt

```
fun main() {
 println("Hello, Gradle with Kotlin DSL")

}
```

### Execute these commands in terminal

```
./gradlew build
./gradlew run
```

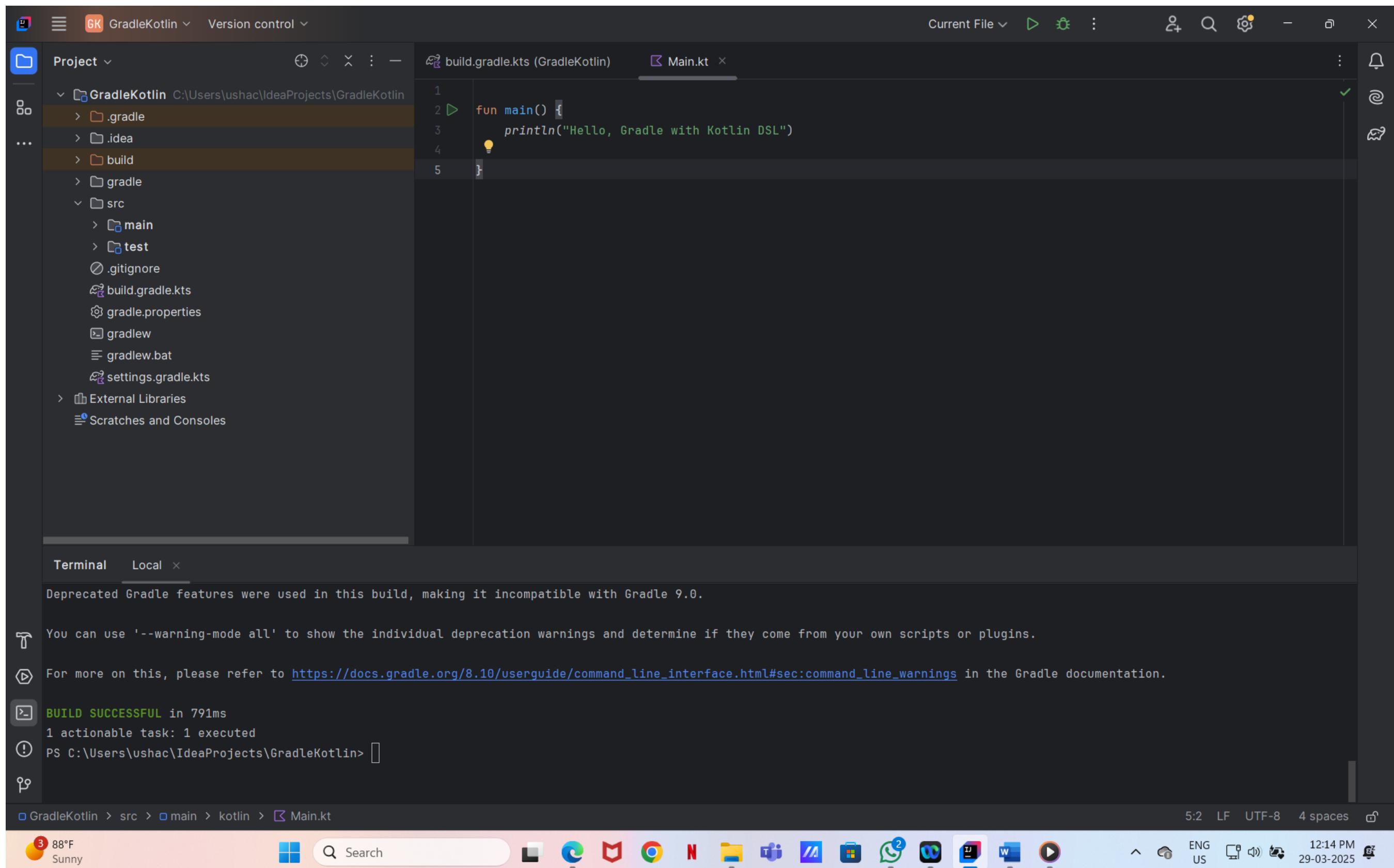
#### Output -

```
> Task :run
Hello, Gradle with Kotlin DSL
```

```
./gradlew fatJar
java -jar ./build/libs/GradleKotlin-1.0-SNAPSHOT.jar
./gradlew hello
```

#### Output-

```
> Task :hello
Hello, Usha!
```



## JENKINS

**What is Jenkins? Installation of Jenkins on local host, Configuring Jenkins, setting up a CI pipeline, Integrating Jenkins with Maven or Gradle running automated builds and tests.**

**2. Jenkins is an open-source automation server used for:**

- ✓ **Continuous Integration (CI)** – Automatically testing and integrating code changes
- ✓ **Continuous Deployment (CD)** – Automating application deployment
- ✓ **Building Pipelines** – Managing end-to-end software development workflows
- ✓ **Plugin-Based Extensibility** – Supporting tools like Maven, Gradle, Ansible, Docker, and Azure DevOps

**3. Why Use Jenkins?**

- ✓ Automates builds and tests
- ✓ Reduces manual intervention
- ✓ Improves software quality
- ✓ Works with multiple tools and platforms

#### 4. Installing Jenkins: Jenkins can be installed using multiple methods -

- 1 Windows Installer (.msi) – Recommended for Windows**
- 2 Linux Package Manager – Best for Linux Users**
- 3 Jenkins WAR File – Universal method using Java**

#### Steps to install Jenkins in windows

##### Step 1: Download Jenkins

 Download from: Download and deploy Choose Windows Installer (.msi) for an easy setup.

##### Step 2: Install Jenkins

- 1** Run the downloaded .msi file.
- 2** Follow the installation wizard.
- 3** Select Run Jenkins as a Windows Service (recommended).
- 4** Choose the installation directory (default: C:\Program Files\Jenkins ).
- 5** Click Install and wait for the setup to complete.

##### Step 3: Start Jenkins

- 1** Open Services ( services.msc ) and ensure Jenkins is running.
- 2** Open a web browser and go to: 1 <http://localhost:8080>

##### Step 4: Unlock Jenkins

- 1** Find the initial Admin Password in:
  1. C:\Program Files\Jenkins\secrets\initialAdminPassword
  2. Copy the password and paste it into the Jenkins setup page.

##### Step 5: Install Recommended Plugins Jenkins will prompt you to install plugins.

Click "Install Suggested Plugins".

##### Step 6: Create Admin User

- 1** Set up a Username, Password, and Email.
- 2** Click Save and Finish.

Jenkins is now ready! Access it anytime at: 1 <http://localhost:8080>

#### Configuring Jenkins for First Use

##### Understanding the Jenkins Dashboard After logging in, you will see:

- ◆ New Item → Create Jobs/Pipelines

- ◆ Manage Jenkins → Configure System, Users, and Plugins
- ◆ Build History → View previous builds
- ◆ Credentials → Store secure authentication details

 **Installing Additional Plugins** Jenkins supports plugins for various tools like Maven, Gradle, Docker, and Azure DevOps.

- ◆ To install a plugin:

- 1 Go to Manage Jenkins → Manage Plugins
- 2 Search for the required plugin
- 3 Click Install without Restart

 **Setting Up Global Tool Configuration** Configure Java, Maven, and Gradle in Jenkins:

- 1 Go to Manage Jenkins → Global Tool Configuration
- 2 Add paths for: JDK ( C:\Program Files\Java\jdk-17 )  
Maven ( C:\Maven\apache-maven- ) Gradle ( C:\Gradle\gradle- )
- 3 Click Save

## Continuous Integration with Jenkins

### 1. Configuring Jenkins & Git Integration

Step 1: Verify Git Installation in Jenkins

1. Open Jenkins Dashboard → Manage Jenkins → Global Tool Configuration.
2. Under Git, verify the installation path (e.g., C:\Program Files\Git\bin\git.exe ).
3. Click Save.

Step 2: Add GitHub Credentials in Jenkins

1. Navigate to Manage Jenkins → Manage Credentials.
2. Select Global credentials (unrestricted) → Click Add Credentials.
3. Choose Username with password or SSH Key, provide details, and click OK

### 2. Running a Selenium Java Test from a Local Maven Project

#### Step 1: Create a New Jenkins Job

1. Go to Jenkins Dashboard → Click New Item.
2. Enter a project name → Select Freestyle Project.
3. Click OK.

#### Step 2: Configure the Build Step

1. Scroll to Build → Click Add build step → Execute Windows Batch Command.
2. Enter the following commands (ensure correct navigation to project directory):

```
1 cd D:\Idea Projects\MVNGRDLDEMO
2 mvn test
```
- 3 Click Save → Click Build Now to execute the test.

### 3. Running Selenium Tests from a GitHub Repository via Jenkins

## Step 1: Set Up a New Jenkins Job for GitHub Project

1. Go to Jenkins Dashboard → Click New Item.
2. Enter a project name → Select Freestyle Project.
3. Click OK.

## Step 2: Configure Git Repository in Jenkins

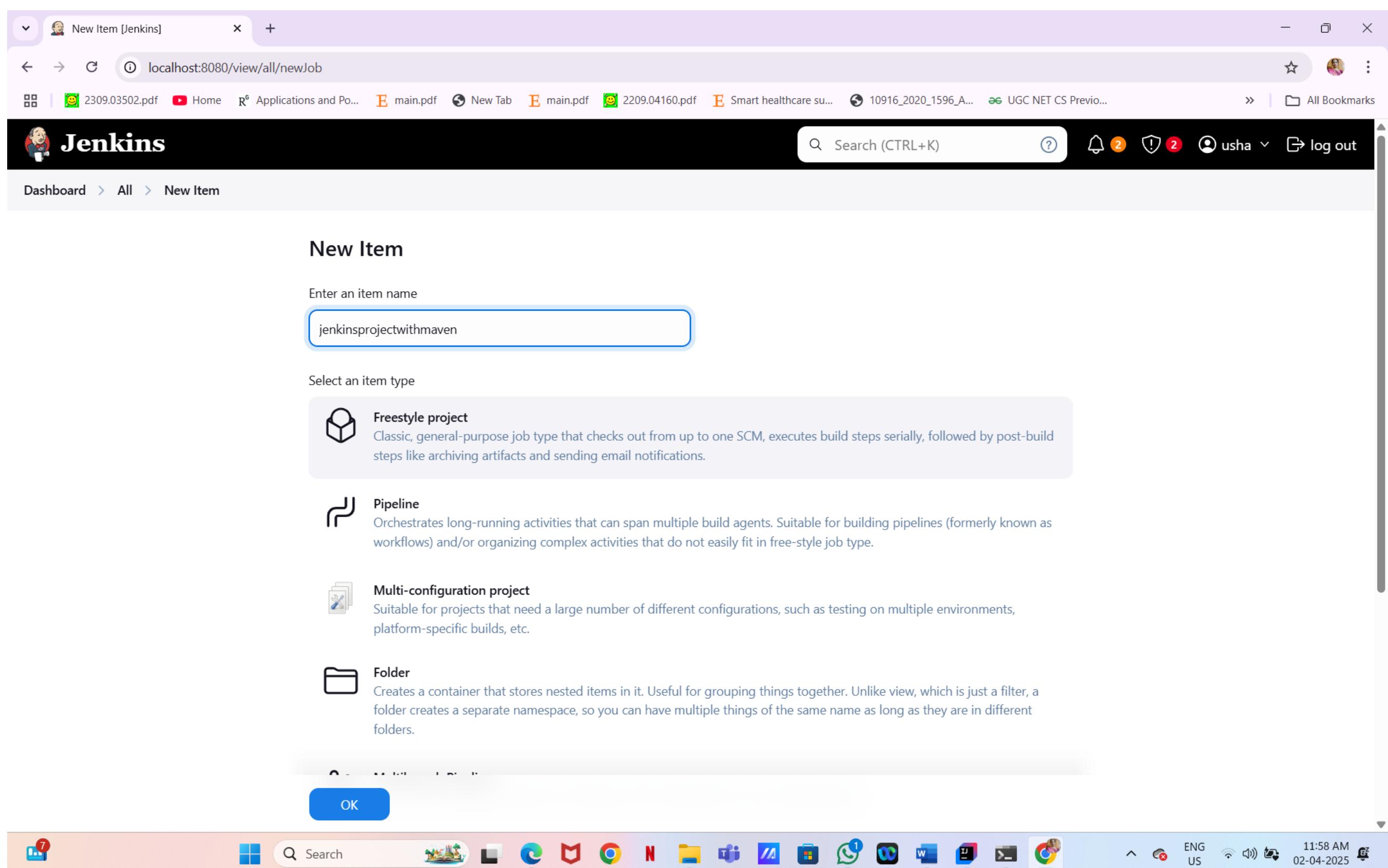
1. Under Source Code Management, select Git.
2. Enter your GitHub repository URL (e.g., <https://github.com/your-repo-name.git>).
3. Select the Git credentials configured earlier.

## Step 3: Add Build Step for Maven

1. Scroll to Build → Click Add build step → Execute Windows Batch Command.
2. Enter the Maven test command: `mvn test`
3. Click Save.

## Step 4: Trigger the Build

1. Click Build Now to fetch the code from GitHub and execute the Selenium tests.
2. Check the Console Output to verify test execution.



**Maven Configuration**

Default settings provider: Use default maven settings

Default global settings provider: Use default maven global settings

**JDK installations**

Add JDK

**Git installations**

Name:  (with a red 'X' icon)

Save Apply

+ New Item Add description

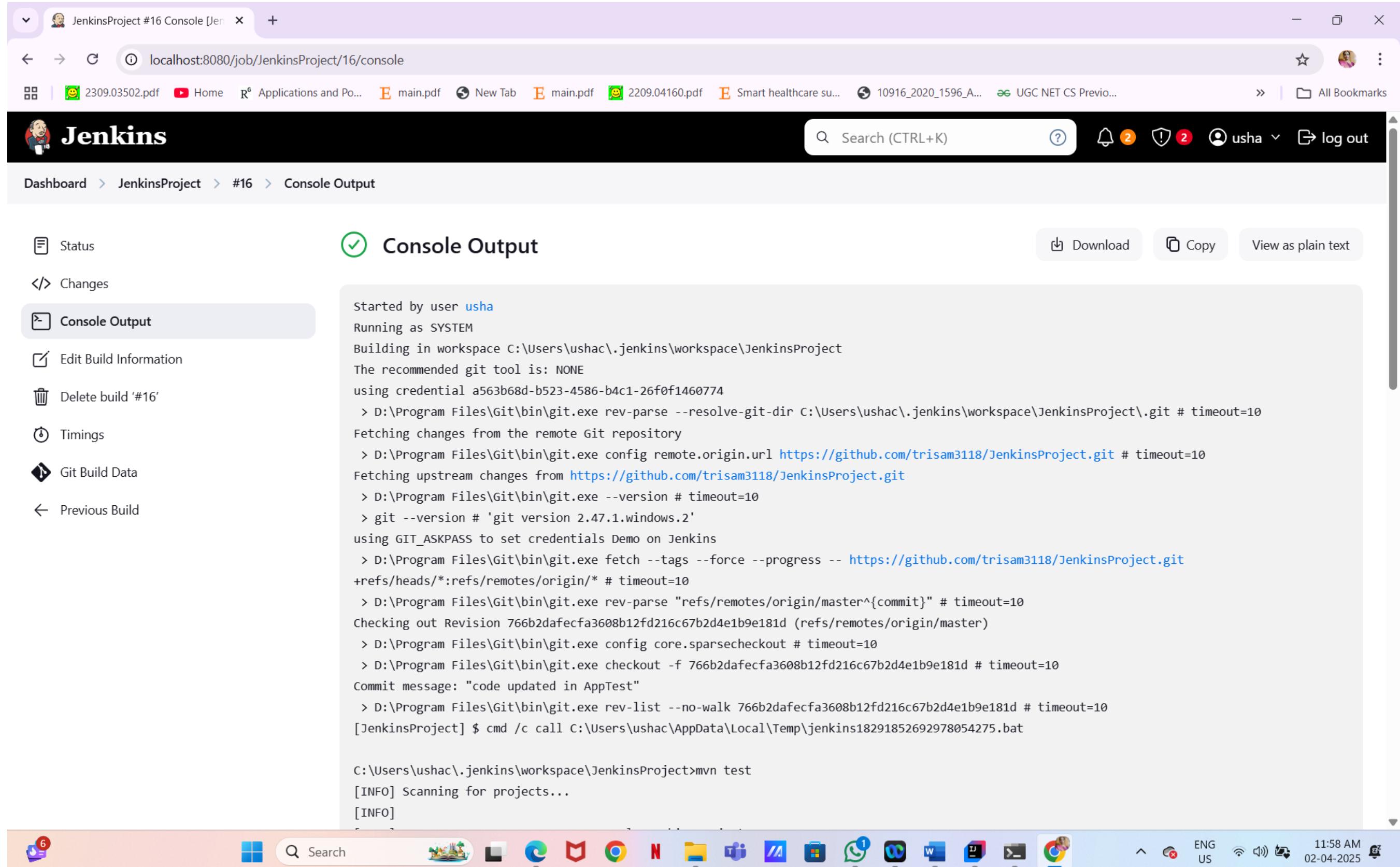
- Build History
- All
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views

S	W	Name ↓	Last Success	Last Failure	Last Duration
✗	rainy	biet.build1	N/A	1 mo 10 days #5	0.11 sec
✓	sunny	DemoWebsite	5 days 23 hr #8	N/A	4.2 sec
✓	sunny	Github Connect Demo	11 days #8	N/A	2.2 sec
✓	sunny	JenkinsProject	2 min 12 sec #15	N/A	20 sec
✓	sunny	JSS-Selenium-G	4 days 15 hr #12	N/A	7.2 sec

Icon: S M L

Changes Workspace Build Now Configure Delete Project Rename

localhost:8080/job/JenkinsProject/ REST API Jenkins 2.479.3



## Pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>JenkinsProject</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>JenkinsProject</name>
 <url>http://maven.apache.org</url>

 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>

```

```
<dependencies>
 <dependency>
 <groupId>org.testng</groupId>

 <artifactId>testng</artifactId>
 <version>7.10.2</version>
 <scope>test</scope>
 </dependency>
 <dependency>
 <groupId>org.seleniumhq.selenium</groupId>
 <artifactId>selenium-java</artifactId>
 <version>4.25.0</version>
 </dependency>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>
</dependencies>
<build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-resources-plugin</artifactId>
 <version>3.2.0</version>
 <executions>
 <execution>
 <phase>prepare-package</phase> <!-- Before packaging -->
 <goals>
 <goal>copy-resources</goal>
 </goals>
 <configuration>
 <outputDirectory>${project.basedir}/docs</outputDirectory>
 <resources>
 <resource>
 <directory>src/main/resources</directory>
 <includes>
 <include>**/*</include> <!-- Copy all files in src/main/resources -->
 </includes>
 </resource>
 </resources>
 </configuration>
 </execution>
 </executions>
 </plugin>
 </plugins>
</build>
```

```
</resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

Now, **index.html, style.css and logo.png files** from **docs** folder is added to your Jenkins Project

### Apptest.java

```
package org.test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import static org.testng.Assert.assertTrue;

public class WebpageTest {
 private static WebDriver driver;

 @BeforeTest
 public void openBrowser() throws InterruptedException {
 driver = new ChromeDriver();
 driver.manage().window().maximize();
 Thread.sleep(2000);
 driver.get("https://trisam3118.github.io/demowebsite/");// here enter static website URL from
your GITHUB
 }

 @Test
 public void titleValidationTest(){
 String actualTitle = driver.getTitle();
 String expectedTitle = "Computer Science and Engineering(Data Science)";
 Assert.assertEquals(actualTitle, expectedTitle);
 }
}
```

```
 assertTrue(true, "Title should contain 'Computer Science and Engineering(Data Science)'");
}

@AfterTest
public void closeBrowser() throws InterruptedException {
 Thread.sleep(1000);
 driver.quit();
}
}
```

**Note:** Run this file it should open browser and validates the title.

## Experiment 5

**Configuration Management with Ansible: Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook**