**Experiment 1: Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup**

**Overview of Build Automation Tools**

Build automation tools help developers streamline the process of building, testing, and deploying software projects. They take care of repetitive tasks like compiling code, managing dependencies, and packaging applications, which makes development more efficient and error-free.

Two popular tools in the Java ecosystem are Maven and Gradle. Both are great for managing project builds and dependencies, but they have some key differences.

**Maven**

* What is Maven? Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called pom.xml (Project Object Model) to define project settings, dependencies, and build steps.
* **Main Features:**
  + Predefined project structure and lifecycle phases.
  + Automatic dependency management through Maven Central.
  + Wide range of plugins for things like testing and deployment.
  + Supports complex projects with multiple modules.

**Gradle**

* What is Gradle? Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.
* **Main Features:**
  + Faster builds thanks to task caching and incremental builds.
  + Flexible and customizable build scripts.
  + Works with Maven repositories for dependency management.
  + Excellent support for multi-module and cross-language projects.
  + Integrates easily with CI/CD pipelines.
* **Key Differences Between Maven and Gradle**

| **Aspect** | **Maven** | **Gradle** |
| --- | --- | --- |
| **Configuration** | XML (pom.xml) | Groovy or Kotlin DSL |
| **Performance** | Slower | Faster due to caching |
| **Flexibility** | Less flexible | Highly customizable |
| **Learning Curve** | Easier to pick up | Slightly steeper |
| **Script Size** | Verbose | More concise |
| **Dependency Management** | Uses Maven Central | Compatible with Maven too |
| **Plugin Support** | Large ecosystem | Extensible and versatile |

**Installing Maven and Gradle on Ubuntu**

**A.Installing Maven**

**Step 1**: Open a terminal and run:

sudo apt update

sudo apt upgrade

**Step 2:** Install Java (if not already installed)

Maven requires Java. Install OpenJDK (for example, version 11):

sudo apt install openjdk-11-jdk

java -version

**Expected Output (example):**

openjdk version "11.0.11" 2021-04-20

OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04)

OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode,

sharing)

**Step 3: Install Maven**

**Run the following command:**

sudo apt install maven

After installation, check the Maven version:

mvn -version

**Expected Output (example):**

Apache Maven 3.6.3

Maven home: /usr/share/maven

Java version: 11.0.11, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-

openjdk-amd64

Default locale: en\_US, platform encoding: UTF-8

OS name: "linux", version: "5.4.0-xx-generic", arch: "amd64", family: "unix"

**B. Installing Gradle**

Gradle can be installed in two primary ways: via the Ubuntu repositories (which may not be

the latest version) or by manually installing the latest version.

**Option 1: Install from Ubuntu Repositories**

sudo apt install gradle

gradle -v

**Option 2: Install the Latest Version Manually (Recommended)**

**1. Download Gradle:**

wget <https://services.gradle.org/distributions/gradle-8.0-bin.zip>

**2. Unzip the Downloaded File:**

sudo unzip -d /opt/gradle gradle-8.0-bin.zip

**3. Set Up the Environment Variables:**

Add Gradle to your system PATH by appending

the following line to your ~/.bashrc (or ~/.profile):

echo "export PATH=\$PATH:/opt/gradle/gradle-8.0/bin" >> ~/.bashrc

source ~/.bashrc

**4. Verify the Installation:**

gradle -v

Expected Output (example):

------------------------------------------------------------

Gradle 8.0

------------------------------------------------------------

Build time: 2022-05-03 12:00:00 UTC

Revision: <revision-hash>

Kotlin: 1.6.x

Groovy: 3.0.x

Ant: Apache Ant(TM) version 1.10.x compiled on ...

JVM: 11.0.11 (Ubuntu 11.0.11+9-Ubuntu-0ubuntu2.20.04)

OS: Linux 5.4.0-xx-generic amd64

**Experiment 2: Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins**

**Step 1**: Open Your Terminal

**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. Use the following command to create a new Maven project:

mvn archetype:generate -DgroupId=com.example -DartifactId=MyMavenApp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

• **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.

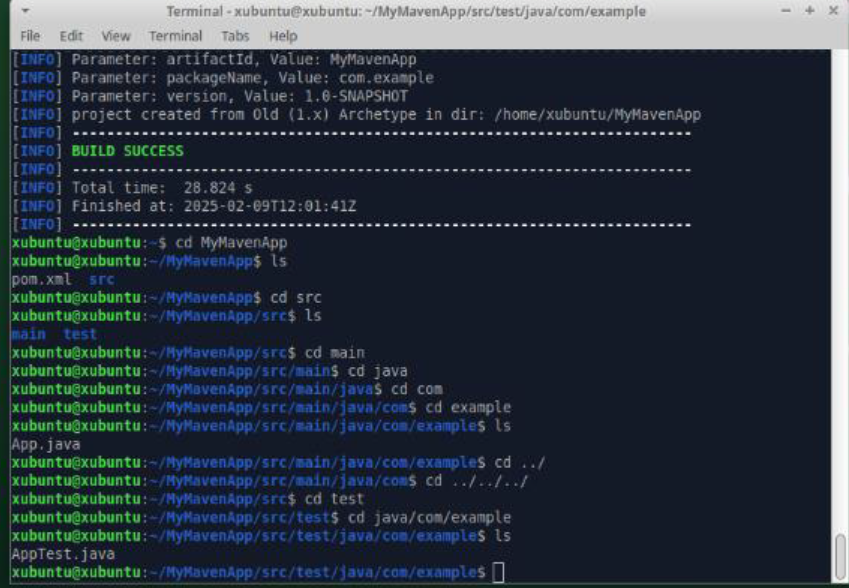
• **-DinteractiveMode=false**: Runs the command in non-interactive mode, using the

provided parameters.

**Step 3: Navigate to Your Project Directory**

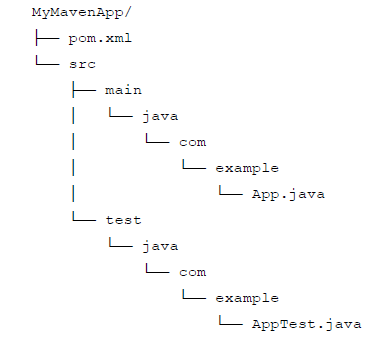
Once the command completes successfully, change your directory to the newly created project:

cd MyMavenApp



**3. Maven Project Layout and Components**

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



**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.

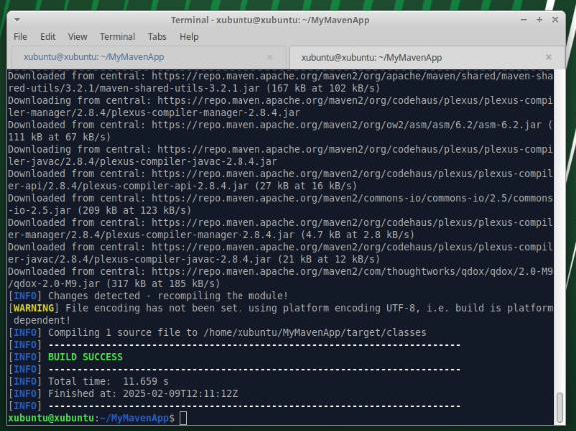
**4. Understanding the POM File (pom.xml)**

<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>MyMavenApp</artifactId>  
 <version>1.0-SNAPSHOT</version>   
 <properties>   
 <maven.compiler.source>11</maven.compiler.source> <maven.compiler.target>11</maven.compiler.target>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
</dependency>  
</dependencies>  
<build>  
<plugins>  
<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>  
<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
</plugin>  
</plugins>  
</build>  
</project>

**Common Maven Commands**

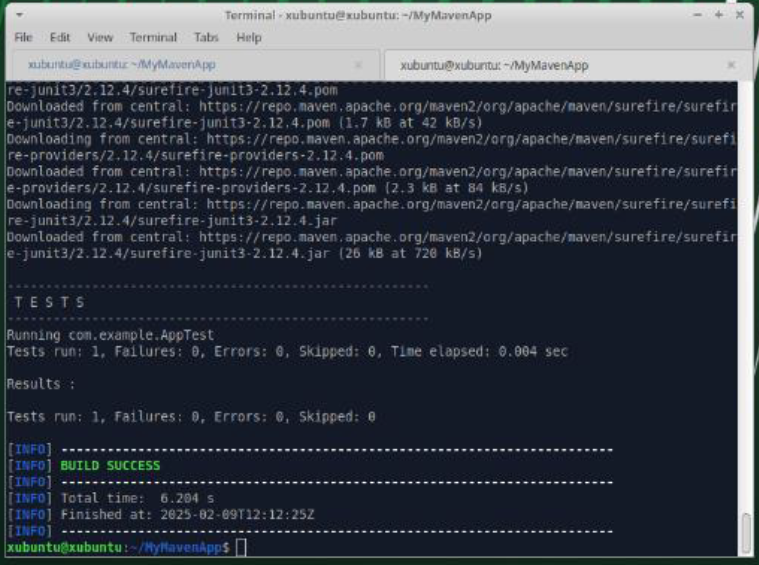
• **Compile the Project:**

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.