



# Maven Tutorial

By durgeshkumar8896 • Mon Aug 12 2024



Need help? Chat with  
us on WhatsApp!



# Maven Tutorial

## Introduction to Maven



### What is Maven?

- **Maven** is a build automation tool that manages various aspects of building software: dependency management and

Need help? Chat with us on WhatsApp!



project build lifecycle management.

- It simplifies the build process by managing dependencies, compiling source code, packaging it into a deliverable (such as a JAR file), and deploying it to a repository.
- Maven is based on the concept of a Project Object Model (POM), which is a central piece of information that manages a project's build, reporting, and documentation.

His

Need help? Chat with us on WhatsApp!

Maven



- Maven was developed by the Apache Software Foundation

and released in 2004.

- It was created to address the problems with Apache Ant, a popular build tool at the time. Maven was designed to simplify the build process, provide a uniform build system, and offer project information that could be shared among developers.
- Over the years, Maven has become one of the most widely used build tools in the Java ecosystem.

**Set**

Need help? Chat with us on WhatsApp!



This section will guide your trainees through installing and setting up Maven on their

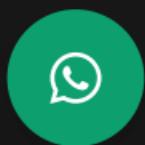
machines and integrating it with popular IDEs like IntelliJ IDEA and Eclipse.

## Installing Maven

- **Prerequisites:**

- Ensure that Java is installed on the machine. Maven requires Java to run. The trainees should have the Java Development Kit (JDK) installed and the JAVA\_HOME environment variable set.

- **S** Need help? Chat with us on WhatsApp!



1. **Download Maven:**

- Go to the [Maven official download page](#).

- Download the latest binary zip archive or tar.gz file.

## 2. Extract the Archive:

- Extract the downloaded archive to a directory of your choice (e.g.,  
C:\\Program  
Files\\Apache\\maven on  
Windows or  
/usr/local/apache-maven  
on macOS/Linux).

## 3. Set Environment

Need help? Chat with us on WhatsApp!



### • Windows:

- Right-click on 'This PC' or 'My Computer',

then go to Properties  
→ Advanced System  
Settings →  
Environment  
Variables.

- Add a new system variable named MAVEN\_HOME pointing to the Maven directory.
- Append the bin directory of Maven to the Path system variable: C:\\Program Files\\Apache\\maven\\bin.

Need help? Chat with us on WhatsApp!



- Open .bash\_profile or .zshrc file in your

home directory and  
add:

Copy

```
export MAVEN_HOME=  
export PATH=$MAVEN
```

- Run source  
~/.bash\_profile or  
source ~/.zshrc to  
apply the changes.

#### 4. Verify Installation:

- Open a terminal or

Need help? Chat with  
us on WhatsApp!



- You should see the  
Maven version  
information along with

Java details if Maven is correctly installed.

## Setting up Maven in IDEs

- **IntelliJ IDEA:**

1. **Maven Integration:**

- IntelliJ IDEA has Maven integration built-in, so no additional setup is required.

- When creating a new project, you can select

Need help? Chat with us on WhatsApp!



automatically generate the pom.xml file and

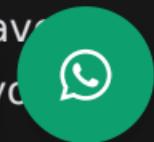
manage the project structure.

## 2. Importing a Maven Project:

- If you already have a Maven project, you can import it by opening IntelliJ and selecting File → Open, then navigating to the pom.xml file of your project.

## 3. Maven Tool Window:

Need help? Chat with us on WhatsApp!



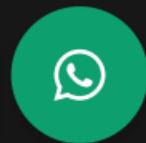
can run Maven goals, view the lifecycle, and manage dependencies directly from the IDE.

- **Eclipse:**

1. **Installing the Maven Plugin:**

- Eclipse requires the m2e plugin for Maven integration, though many versions of Eclipse come with this plugin pre-installed.
- To install it manually:  
Help → Eclipse Marketplace → Search for "m2e" → Install.

2 Need help? Chat with us on WhatsApp!



- Go to File → New → Project → Maven → Maven

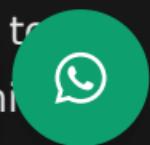
Project.

- Follow the wizard to set up your project, specifying details like GroupId, ArtifactId, and packaging type.

### 3. Importing a Maven Project:

- If you have an existing Maven project, import it by selecting File → Import → Maven → Existing Maven

Need help? Chat with us on WhatsApp!



the pom.xml.

### 4. Maven Integration:

- Eclipse provides a "Maven" perspective that allows you to run Maven commands, manage dependencies, and configure the build lifecycle from within the IDE.

## **Understanding the .m2 Directory and settings.xml**

- **The .m2 Directory:**

Need help? Chat with us on WhatsApp!



C:\\\\Users\\\\YourName\\\\.m2  
on Windows or

/home/YourName/.m2 on  
macOS/Linux).

- Contains the repository folder where Maven stores downloaded dependencies. This is known as the local repository.
  - You can delete this folder if you want to force Maven to re-download dependencies, but usually, Maven manages it automatically.
- 
- **settings.xml File:**

Need help? Chat with us on WhatsApp!



customize the behavior of Maven globally across all projects.

- Common configurations include:
  - **Mirror configuration** to point Maven to a specific repository.
  - **Proxy settings** if you're behind a firewall or using a corporate network.
  - **Local repository** location customization.

Example `settings.xml`:

Copy

Need help? Chat with us on WhatsApp!



```
<mirror>
  <id>central</id>
  <mirrorOf>centr
  <url><http://my
</mirror>
```

```
</mirrors>
<proxies>
    <proxy>
        <id>example-pro
        <active>true</a
        <protocol>http<
        <host>proxy.myc
        <port>8080</por
    </proxy>
</proxies>
</settings>
```

## Project Object Model (POM)

The POM (Project Object Model) is the central configuration file for a Maven project. Understanding the POM file is essential because it contains all the configuration details for a Maven project.

Need help? Chat with us on WhatsApp!



# Structure of a pom.xml File

- The pom.xml file is an XML file that contains information about the project and configuration details used by Maven to build the project.

Here's a basic structure of a pom.xml file:

Copy

```
<project xmlns="http://mav  
xmlns:xsi="http:/
```

Need help? Chat with us on WhatsApp!



```
<groupId>com.example</g  
<artifactId>my-app</art  
<version>1.0-SNAPSHOT</
```

```
<packaging>jar</packagi
```

```
<name>My App</name>
<url><http://maven.apac
```

```
<dependencies>
```

```
    <dependency>
```

```
        <groupId>junit<
        <artifactId>jun
        <version>4.12</
        <scope>test</sc
```

```
    </dependency>
</dependencies>
```

```
<build>
```

```
    <plugins>
```

```
        <plugin>
```

```
            <groupId>or
            <artifactId
            <version>3
```

Need help? Chat with  
us on WhatsApp!



```
        <target
```

```
    </configura
```

```
        </plugin>
</plugins>
```

```
</build>  
</project>
```

## Key Elements in POM

### 1. modelVersion:

- This specifies the version of the POM model being used. Currently, 4.0.0 is the default and most widely used version.

### 2. groupId:

Need help? Chat with us on WhatsApp!

facilitates your project

across all projects. Typically, it follows the reverse



domain name convention  
(e.g., com.example).

### 3. **artifactId:**

- The artifactId is the name of the jar without the version.  
It is the name of the project.

### 4. **version:**

- The version is the project's current version. Maven uses this version to distinguish between different versions

Need help? Chat with us on WhatsApp!



1.0.0, or they can include SNAPSHOT, which indicates a version in development.

## 5. packaging:

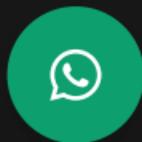
- The type of artifact this project produces. Common packaging types are jar, war, pom, etc.
- If omitted, jar is assumed by default.

## 6. name and url:

- These are optional elements used to provide a human-readable name and project URL.

## 7. d

Need help? Chat with us on WhatsApp!



- The dependencies section lists all the external libraries your project depends on.

Maven will download these libraries automatically from the central repository or other specified repositories.

#### 8. build:

- The build section is used to specify how the project should be built. This includes specifying plugins, custom build directories, resources, etc.

## Dependencies and

Dep  
Ma

Need help? Chat with us on WhatsApp!



- Dependencies:**

- Dependencies are external libraries that your project needs to work. You specify these in the dependencies section of the POM.
- Maven automatically downloads the specified versions of dependencies and any transitive dependencies they require.

Example:

Copy

```
<dependencies>
```

Need help? Chat with us on WhatsApp!



```
    <dependency>spring-
```

```
        <version>2.5.4</ver
```

```
            </dependency>
```

```
</dependencies>
```

- **Transitive Dependencies:**

- Maven resolves transitive dependencies automatically. For example, if your project depends on A, and A depends on B, Maven will include B as a dependency in your project.

## **Maven Dependency Scopes and Profiles**

1.

Need help? Chat with us on WhatsApp!



Maven provides several scopes for dependencies, each determining at what phase of the build process the dependency is

available and whether it is included in the final artifact (e.g., a JAR or WAR file). Understanding these scopes is crucial for managing project dependencies effectively.

Here's a detailed explanation of each scope, with examples:

## a. compile Scope

- **Default Scope:** If no scope is specified, Maven uses compile by default.
- A Need help? Chat with us on WhatsApp! is part of the build lifecycle, including compilation, testing, and packaging.

Need help? Chat with us on WhatsApp!



- **Inclusion in Final Artifact:**  
The dependency is included in the final artifact.
- **Typical Use:** Used for dependencies required to build, test, and run the project.

### Example:

Copy

```
<dependency>
    <groupId>org.apache.com
    <artifactId>commons-lan
    <version>3.12.0</versio
    <scope>compile</scope>
</dependency>
```

Need help? Chat with us on WhatsApp!

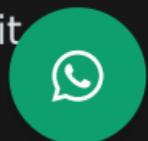


- In this example, commons-lang3 is a utility library used

during all phases of the build, so it's specified with compile scope.

## b. provided Scope

- **Availability:** The dependency is available at compile-time but is not included in the final artifact.
- **Inclusion in Final Artifact:** Not included in the final artifact. The assumption is that the runtime environment (e.g.,
  - a) Need help? Chat with us on WhatsApp!
- **Type:** It includes dependencies that are provided by the runtime environment, such as servlet



APIs provided by an application server.

## Example:

Copy

```
<dependency>
    <groupId>javax.servlet<
    <artifactId>javax.servl
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

- Here, the javax.servlet-api dependency is only needed for the application to run. It's expected to be provided by the server (e.g., Tomcat) in production, so it's marked with provided scope.

Need help? Chat with us on WhatsApp!



## c. runtime Scope

- **Availability:** The dependency is not required for compilation but is needed during runtime, including execution and tests.
- **Inclusion in Final Artifact:** The dependency is included in the final artifact.
- **Typical Use:** Used for dependencies that are needed only at runtime, like JDBC drivers.

E

Need help? Chat with us on WhatsApp!



Copy

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-conne
```

```
<version>8.0.26</versio  
<scope>runtime</scope>  
</dependency>
```

- In this example, the MySQL JDBC driver is not required for compiling the code but is needed at runtime when the application connects to a MySQL database.

#### d. test Scope

- **Availability:** The dependency is included in the final artifact only during the test phases.
- **Inclusion in Final Artifact:** Not included in the final artifact.

Need help? Chat with us on WhatsApp!



artifact.

- **Typical Use:** Used for libraries that are required only for testing, such as JUnit or Mockito.

### Example:

Copy

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

Need help? Chat with us on WhatsApp!



- Here, JUnit is only needed for running unit tests and isn't included in the final

build output, so it's scoped as test.

## e. system Scope

- **Availability:** The dependency is available for compilation and testing but must be explicitly provided by the developer on the system.
- **Inclusion in Final Artifact:** Not included in the final artifact.
- **Typical Use:** Used rarely when the dependency is in a repository and must be provided locally.

Need help? Chat with us on WhatsApp!



- **Important:** You must explicitly provide the path to the JAR file using the `<systemPath>` element.

## Example:

Copy

```
<dependency>
    <groupId>com.example</g
    <artifactId>custom-lib<
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${project.b
</dependency>
```

Need help? Chat with  
us on WhatsApp!



lib.jar is not available in  
any Maven repository and is  
instead provided manually

in the lib directory of the project.

## Maven Profiles

Maven profiles allow you to customize the build process for different environments or situations. A profile can modify the default behavior by adding, removing, or overriding certain parts of the POM, such as dependencies, plugins, properties, etc.

Need help? Chat with us on WhatsApp!



## How Profiles Work

- **Activation:** Profiles can be activated explicitly using the `-P` flag in the Maven command or automatically based on certain conditions (e.g., OS type, JDK version, environment variables).
- **Scope:** Profiles can modify the build process for a particular environment, such as development, testing, or production.

## Creating and Using

### Profiles

Need help? Chat with us on WhatsApp!



#### a. Simple Profile Example

This profile will activate different configurations based on whether you're in a development or production environment.

## POM with Profiles:

Copy

```
<project>
  ...
  <profiles>
    <profile>
      <id>development</i
      <properties>
        <environment.n
          <db.url>jdbc:h
        </properties>
      <dependency>
        <groupId>c
        <artifactI
          <version>1
        </dependency>
      </dependencies>
```

Need help? Chat with us on WhatsApp!



```
</profile>

<profile>
    <id>production</id>
    <properties>
        <environment.name>prod
        <db.url>jdbc:mysql://127.0.0.1:3306/test
    </properties>
    <dependencies>
        <dependency>
            <groupId>maven
            <artifactId>mysql-connector-java
            <version>8.0.23
        </dependency>
    </dependencies>
</profile>
</profiles>
</project>
```

Need help? Chat with us on WhatsApp!



- [Explanation](#)

- **Development Profile:**

Activates a H2 in-memory

database for development. This is useful for quick setup and teardown without affecting the production database.

- **Production Profile:** Uses a MySQL database for the production environment.

- **Activating a Profile:**

- To activate the development profile:

Copy

Need help? Chat with us on WhatsApp!



- To activate the production profile:

Copy

```
mvn clean install -Pprod
```

## b. Profile Activation Based on Conditions

Profiles can also be activated based on various conditions, such as the operating system, JDK version, or environment variables.

### POM with Conditional Profiles:

Need help? Chat with us on WhatsApp!

```
<project>
  ...
  <profiles>
    <profile>
      <id>windows</id>
```



```
<activation>
    <os>
        <family>Wi
    </os>
</activation>
<properties>
    <env.name>wind
</properties>
</profile>

<profile>
    <id>jdk-11</id>
    <activation>
        <jdk>11</jdk>
    </activation>
    <properties>
        <jdk.version>1
    </properties>
</profile>

```

Need help? Chat with  
us on WhatsApp!



- **Explanation:**

- **Windows Profile:** Activated automatically when Maven is run on a Windows operating system.
  - **JDK 11 Profile:** Activated automatically when Maven detects that the project is being built with JDK 11.
- **Activating Automatically:**

- If Maven is running on a Windows machine, the windows profile is activated automatically.

Need help? Chat with us on WhatsApp!

activated automatically.

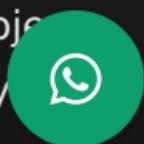


## Maven Build Lifecycle

The Maven build lifecycle is a sequence of phases that define the order in which tasks are executed during the build process of a Maven project. Maven has three built-in build lifecycles:

## 1. Default Lifecycle (Build)

This is the main lifecycle that handles project deployment. It consists of 23 phases, the most important of which are:

- **validate**: Checks the project's configuration and dependencies.
- **Need help? Chat with us on WhatsApp!**  Information is available.
- **compile**: Compiles the source code of the project.

- **test**: Runs the tests using a suitable unit testing framework (like JUnit).
- **package**: Packages the compiled code into a distributable format, such as a JAR or WAR file.
- **verify**: Runs any checks to verify the package is valid and meets quality criteria.
- **install**: Installs the package into the local repository, for use as a dependency in other projects locally.
- **d** Need help? Chat with us on WhatsApp! repository for sharing with other developers and projects.



## 2. Clean Lifecycle

This lifecycle is used to clean up artifacts created by the previous build. It consists of three phases:

- **pre-clean**: Executes processes needed before the actual project cleaning.
- **clean**: Removes files generated during the previous build.
- **post-clean**: Executes processes needed after the project cleaning.

3.

Need help? Chat with us on WhatsApp!



This lifecycle is used to create the project's site documentation. It consists of four phases:

- **pre-site**: Executes processes needed before the actual project site generation.
- **site**: Generates the project's site documentation.
- **post-site**: Executes processes needed after the site generation, preparing for site deployment.
- **site-deploy**: Deploys the generated site to a web server.

## How to Use the Build Lifecycle

You phases using the Maven command:

Need help? Chat with us on WhatsApp!



Copy

```
mvn <phase>
```

For example, running `mvn install` will run all the phases leading up to and including the `install` phase.

Understanding these phases allows you to control the build process and ensure that your project is built, tested, packaged, and deployed correctly.

Need help? Chat with us on WhatsApp!



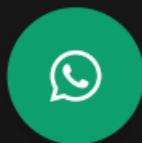
Key ~~phases~~ in the

## Default Lifecycle

When you run a Maven command, such as `mvn package`, Maven executes all the phases up to and including the package phase:

- `mvn validate`: Validates the project.
- `mvn compile`: Compiles the source code of the project.
- `mvn test`: Runs the tests using a suitable unit testing framework.
- `mvn package`: Takes the compiled code and packages it into JAR/WAR files.
- `mvn install`: Packages the JAR/WAR file into the local repository, making it available for other projects locally.

Need help? Chat with us on WhatsApp!

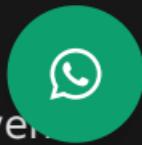


- `mvn deploy`: Copies the final package to the remote repository.

## Phases of the Clean Lifecycle

- `pre-clean`: Executes processes needed prior to cleaning.
- `clean`: Removes files generated during the previous build.
- `post-clean`: Executes processes needed after cleaning.

Need help? Chat with us on WhatsApp!



When you run `mvn clean`, it executes all phases of the Clean lifecycle.

# Phases of the Site Lifecycle

- **pre-site**: Executes processes needed prior to generating the site documentation.
- **site**: Generates the project's site documentation.
- **post-site**: Executes processes needed after generating the site documentation.
- **site-deploy**: Deploys the generated site documentation to the server.

When you run the `mkdocs build` command, it follows the site lifecycle to generate the project documentation.

Need help? Chat with us on WhatsApp!



# **Maven Repositories**

Maven repositories are a central part of the Maven build system. They are where Maven stores and retrieves project dependencies, plugins, and other artifacts. Understanding how repositories work and how to manage them is crucial for effectively using Maven.

## **Types of Maven Repositories**

Maven has three types of repositories:

Need help? Chat with us on WhatsApp!



### **1. Local Repository:**

- The local repository is a directory on your computer where Maven stores downloaded dependencies, plugins, and other artifacts.
- When you build a Maven project, Maven first checks the local repository to see if the required dependencies are already available. If they are, Maven uses them; if not, Maven downloads them from a remote repository (such as the

Need help? Chat with us on WhatsApp!



- The default location of the local repository is

`~/.m2/repository` (on Unix-based systems) or

Copy

`C:\\Users\\YourUsername\\`

You can configure the location of the local repository in the `settings.xml` file:

Copy

- 

```
<settings>
    <localRepository>C:/</localRepository>
</settings>
```

Need help? Chat with us on WhatsApp!

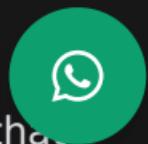


## 2. Central Repository:

- The central repository is a global, publicly available repository hosted by the Maven project. It contains a vast collection of open-source libraries, plugins, and other project artifacts.
- The central repository is the default source from which Maven downloads dependencies if they are not found in the local repository.
- Maven automatically connects to the central repository to download dependencies that are not available locally.

### 3. Remote Repository:

Need help? Chat with us on WhatsApp!



- A remote repository is any repository that is hosted on a web server other than the central repository. Organizations often maintain their own remote repositories to store proprietary artifacts, plugins, or third-party libraries that are not available in the central repository.

- You can configure Maven to use a remote repository by

Need help? Chat with us on WhatsApp!



- Example in pom.xml:

Copy

```
<repositories>
  <repository>
    <id>my-company-r
    <url><http://rep
  </repository>
</repositories>
```

## How Maven Resolves Dependencies

Maven follows a specific order to resolve dependencies:

- Locally first. If the dependency is found, Maven uses it.

Need help? Chat with us on WhatsApp!



**2. Remote Repositories:** If the dependency is not found in the local repository, Maven checks any configured remote repositories.

**3. Central Repository:** If the dependency is not found in any remote repositories, Maven checks the central repository.

**4. Fail:** If Maven cannot find the dependency in any repository, the build fails with an error.

Ma  
Dep  
Reposito

Need help? Chat with  
us on WhatsApp!



## Adding Dependencies

- You add dependencies to your project by specifying them in the dependencies section of the pom.xml file. Maven then resolves these dependencies by checking the repositories in the order described above.

Example:

Copy

```
xmlCopy code
<dependencies>
    <dependency>
        <groupId>org.springframework
        <artifactId>spring-
        <version>2.5.4</ver
```

Need help? Chat with us on WhatsApp!



# Excluding Transitive Dependencies

- Sometimes, a dependency might bring in transitive dependencies that you don't want. You can exclude these unwanted transitive dependencies using the `<exclusions>` tag.

Example:

Copy

```
<dependencies>
```

Need help? Chat with us on WhatsApp!



```
    <version>2.5.4</ver
```

```
    <exclusions>
```

```
        <exclusion>
```

```
            <groupId>or
```

```
<artifactId  
      </exclusion>  
    </exclusions>  
  </dependency>  
</dependencies>
```

## Using Repository Managers

Organizations often use repository managers like Nexus or Artifactory to:

- **Cache Artifacts:** Repository managers cache dependencies from remote repositories, reducing download times and providing redundancy in case a remote repository becomes unavailable.

Need help? Chat with us on WhatsApp!



- **Store Internal Artifacts:**

Organizations can store their internal or proprietary libraries and plugins in a secure location.

- **Enforce Policies:** Repository managers can enforce security policies, licensing, and artifact versioning rules.

## Example of Configuring a Nexus Repository:

Copy

```
<repositories>
```

Need help? Chat with us on WhatsApp!



```
    <repository><url>http://nexus.mycompany.com/nexus/content/repositories/releases</url></repository>
```

```
  </repository>
```

```
</repositories>
```

# Using Mirrors to Redirect Repositories

Mirrors allow you to redirect Maven to use a different repository for a given URL pattern. This is useful for routing all requests that would go to the central repository to an internal mirror.

Example in `settings.xml`:

Copy

Need help? Chat with us on WhatsApp!



```
<id>central-mirror</id>
<mirrorOf>central</mir
<url><http://mycompany
</mirror>
```

```
</mirrors>
```

In this example, all requests to the central repository are redirected to <http://mycompany.com/maven/central>.

## Maven Dependency Management

Maven's dependency management system is one of its most powerful features. It automatically handles the inclusion of dependencies, including their transitive dependencies (dependencies of your dependencies). Understanding

Need help? Chat with us on WhatsApp!



how to manage these dependencies effectively is key to maintaining a clean and efficient build.

## Dependency Scopes (Recap)

As we covered earlier, Maven supports different dependency scopes (`compile`, `provided`, `runtime`, `test`, `system`). These scopes determine when a dependency is available during the build process and whether it is included in the final artifact.

Here's a quick recap:

- **compile**: Available in all phases and included in the final artifact.

Need help? Chat with us on WhatsApp!



final artifact.

- **provided**: Available at compile-time but not included in the final artifact (e.g., servlet APIs provided by the server).
- **runtime**: Not needed for compilation but required during execution (e.g., JDBC drivers).
- **test**: Available only for testing, not included in the final artifact.
- **system**: Similar to provided,

b  
o Need help? Chat with us on WhatsApp!



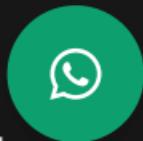
## Transitive Dependencies

Transitive dependencies are dependencies that are required by the libraries your project depends on. Maven resolves these automatically, so you don't have to manually include every single dependency in your pom.xml.

## **Example of Transitive Dependencies:**

- Suppose your project depends on spring-core, which in turn depends on commons-logging.
- M Need help? Chat with us on WhatsApp! d a transitive dependency when g a project.

Need help? Chat with us on WhatsApp!



Copy

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.9</version>
</dependency>
```

- **Explanation:** When Maven processes this dependency, it will also download and include any libraries that spring-core depends on, such as commons-logging.

Dependency Management in Multi-Module Projects

Need help? Chat with us on WhatsApp!



In multi-module projects, you often have a parent POM that defines dependencies for all modules. Maven provides a `<dependencyManagement>` section that allows you to centrally manage dependency versions across multiple modules.

## **Using `<dependencyManagement>` in the Parent POM:**

Copy

```
<project>
    <groupId>com.example</grou
```

Need help? Chat with us on WhatsApp!



```
        <dependencyManagement>
            <dependencies>
                <dependency>
```

```
<groupId>org.s
<artifactId>sp
<version>5.3.9
</dependency>
<dependency>
    <groupId>org.s
    <artifactId>sl
    <version>1.7.3
</dependency>
</dependencies>
</dependencyManagement>

<modules>
    <module>module-a</modu
    <module>module-b</modu
</modules>
</project>
```

- **E** Need help? Chat with us on WhatsApp!



defined in

<dependencyManagement> are not automatically included in the child modules. Instead,

they provide a version and scope template that child modules can use.

- **Child Module POM:**

Copy

```
<project>
  <parent>
    <groupId>com.example</groupId>
    <artifactId>parent-module</artifactId>
    <version>1.0.0</version>
  </parent>

  <artifactId>module-a</artifactId>
  <version>1.0.0</version>

  <dependencies>
    <dependency>
      <groupId>com.example</groupId>
      <artifactId>parent-module</artifactId>
      <version>1.0.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

Need help? Chat with us on WhatsApp!



- **Explanation:** The child module inherits the version and scope of spring-core from the parent POM, thanks to the `<dependencyManagement>` section.

## Excluding Unwanted Transitive Dependencies

Sometimes, you might want to exclude a transitive dependency that is not needed by your project.

Need help? Chat with us on WhatsApp!



## Example of Excluding Transitive Dependencies:

Copy

```
<dependency>
    <groupId>org.springframework
    <artifactId>spring-boot-starter
    <version>2.5.4</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework
            <artifactId>spring-boot-starter-tomcat
            </exclusion>
        </exclusions>
    </dependency>
```

- **E** Configuration excludes the spring-boot-starter-tomcat dependency, which might be useful if you're using a

Need help? Chat with us on WhatsApp!



different server (e.g., Jetty) and don't want Tomcat libraries included.

## Managing Conflicts with Dependency Mediation

When multiple versions of the same dependency are included via transitive dependencies, Maven uses a process called "dependency mediation" to determine which version to use. Maven usually selects the nearest definition (i.e., the version specified in the dependency tree).

Need help? Chat with us on WhatsApp!



## Example of Dependency Conflict:

- If A depends on B:1.0 and C depends on B:2.0, and both A and C are included in your project, Maven will choose one version of B.
- You can control which version is selected by explicitly declaring the desired version in your pom.xml.

Copy

```
<dependency>
    <groupId>com.example</group
    <artifactId>B</artifactId>
    <version>2.0</version>
```

Need help? Chat with us on WhatsApp!



- **Explanation:** By explicitly declaring the version of B, you

override the transitive versions and ensure that B:2.0 is used.

## Best Practices for Dependency Management

### 1. Minimize the Number of Dependencies:

Only include the libraries that are essential to your project. More dependencies can lead to larger builds and more complex dependency graphs, increasing the risk of conflicts.

2. Use `<dependencyManagement>` In multi-module projects, use the `<dependencyManagement>` section to centralize the

Need help? Chat with us on WhatsApp!



management of dependency versions.

### 3. **Exclude Unnecessary**

**Transitive Dependencies:**

Use the `<exclusions>` tag to remove dependencies that are not needed by your project.

### 4. **Resolve Conflicts Explicitly:**

When facing version conflicts, explicitly declare the version you want to use in your `pom.xml`.

### 5. **Regularly Update**

**Dependencies:** Keep your

dependencies up-to-date to

be aware of security vulnerabilities and security patches.

Need help? Chat with us on WhatsApp!



**Share this article ...**

[WhatsApp](#)

[Facebook](#)

[LinkedIn](#)

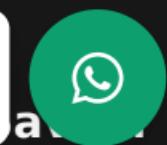
[X](#)

## Trending Blogs...



I Used  
Prompts

Need help? Chat with  
us on WhatsApp!



Discover 3 powerful ChatGPT prompts that  
save you hours daily. Boost your productivit...

[Read More →](#)

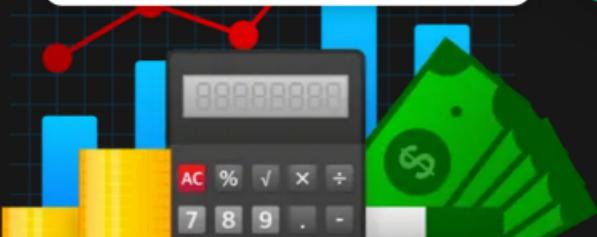


## Top 10 CSS Tips That Will Instantly Improve Your Design...

Before writing CSS, know how it connects with HTML and why using external...

[Read More →](#)

Need help? Chat with us on WhatsApp!





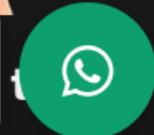
## Best Investment Apps in India (2025): Invest Smart, Grow...

Investing today has become easier than ever — thanks to powerful mobile apps that put...

[Read More →](#)



Need help? Chat with  
us on WhatsApp!



**Top 1**

### Learn in 2025

Hey there! If you're feeling the pressure of keeping up with today's crazy-fast job...

[Read More →](#)



## **iphone 17 Launch 2025: Features, Price in India, Pros ...**

Apple has officially unveiled the \*\*iPhone 17 series\*\* on \*\*September 9, 2025\*\*, at its...

[Read More →](#)

Need help? Chat with  
us on WhatsApp!



JS



# Async/Await vs Promises in JavaScript: Which Should You...

Asynchronous programming is one of the most important topics in modern JavaScript...

[Read More →](#)

## Share this article ...

[WhatsApp](#)

[Facebook](#)

[LinkedIn](#)

[X](#)

Need help? Chat with us on WhatsApp!





# Learn Code With Durgesh

Offering free & premium coding  
courses to lakhs of students via  
YouTube and our platform.



## Explore

About

Course

Blog

Contact

FlexBox Game

Need help? Chat with  
us on WhatsApp!



## **Legal**

Privacy Policy

Terms & Conditions

Refund Policy

Support

## **Contact**

+91-9839466732

[support@learncodewithdurgesh.com](mailto:support@learncodewithdurgesh.com)

Substri  
R Duber  
R

Need help? Chat with  
us on WhatsApp!



Matiyari Chinhat, Lucknow, UP, INDIA  
226028

© 2025 Made with ❤ by Substring Technologies. All rights reserved.

Need help? Chat with us on WhatsApp!

