Coding Bootcamp

Maven

TECHNOLOGY

Getting Started with Maven

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Analyze the basic concepts of Maven and its features to manage projects more effectively and improve build processes
- Identify the steps to install Maven to leverage its full capabilities in managing the lifecycle of Java projects
- Illustrate the concept of Maven lifecycle to manage project builds, dependencies, and deployments efficiently
- Define the usage of Maven repository for efficient and automated dependency management

TECHNOLOGY

Maven: Overview

Maven

Maven is a comprehensive build tool that simplifies project management and dependency management and builds processes for Java projects.

It enables developers to focus on writing code and delivering high-quality software by standardizing and automating many aspects of project development.

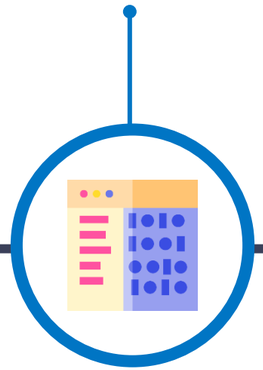
What Is a Build Tool?

It is essential in modern software development, providing automation, consistency, and efficiency in transforming source code into deployable software. It helps to:

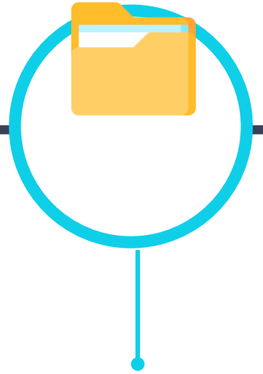
Generate source code

Generate documentation from source code

Compile the source code



Compile code packages into JAR of the ZIP file



Install the packages in repositories

Build Tool: Uses

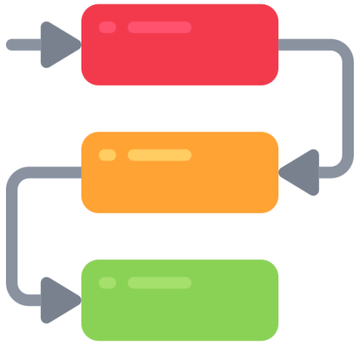
It helps developers manage:

Project builds

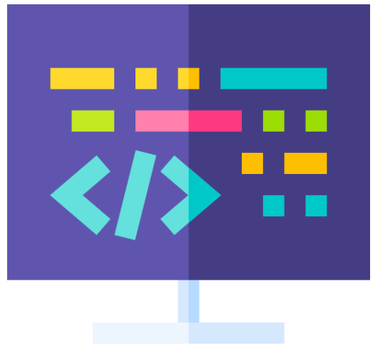
Project documentation

Reporting

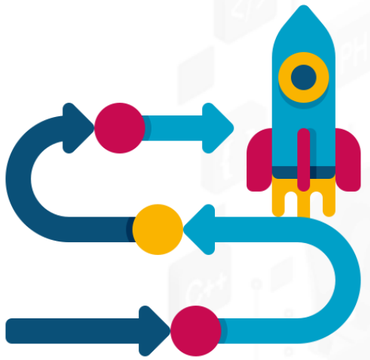
Building Tool: Uses



Maven dependencies



Software Configuration Management (SCMs)



Project releases

Primary Objectives of Using Maven

Maven helps in:

- Maintaining uniformity in the build generations system
- Making a simpler build process
- Maintaining quality project information
- Providing better development guidelines and processes
- Maintaining transparent migration to new features

Project Object Model (POM) is the most important unit.

Maven: Advantages

Maven provides uniformity in the build generation process.

It provides:

A POM.xml file to configure the build of the project

Plugins that are shared by all projects

Why Use Maven?

Its primary purpose is to simplify the build process by managing dependencies, compiling source code, packaging the project, running tests, and generating reports.

Build automation	Maven compiles source code into binary formats (like JAR, WAR, and EAR) and manages project packaging efficiently.
Dependency management	It ensures consistency by managing and standardizing versions of external libraries and frameworks.
Project management	It enforces a standardized project directory structure, making projects organized and predictable.
Lifecycle management	It defines a series of build phases (validate, compile, and test) with predefined goals for each phase.
CI/CD integration	It integrates seamlessly with CI/CD tools like Jenkins, automating build, test, and deployment processes.

Where to Use Maven?

Here are some key areas where Maven is used:

Software development	Maven automates the build process for Java projects. It manages project dependencies and ensures consistency.
Enterprise applications	It is ideal for managing large-scale enterprise applications with multiple modules and helps to streamline the build process.
API development	It integrates with tools like Swagger for API documentation and JUnit for testing, streamlining the API development process.

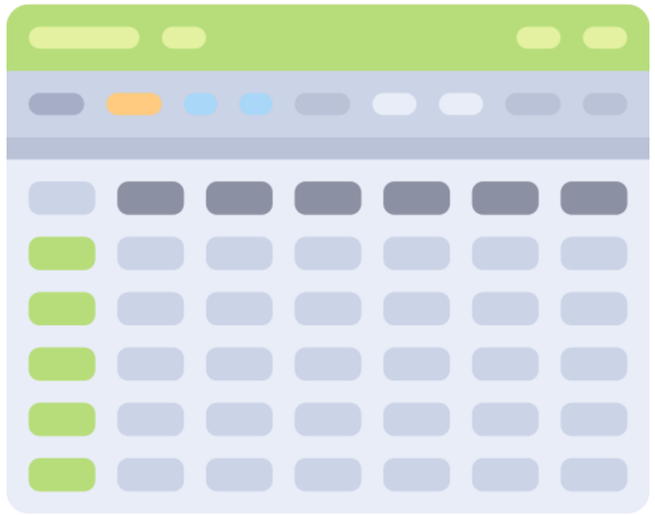
Where to Use Maven?

Here are some key areas where Maven is used:

Open source projects	Maven is used to standardize projects because its extensive plugin ecosystem supports various tasks.
Web applications	It supports web frameworks like Spring and Hibernate and simplifies web application development and deployment.
Microservices	It is used in Android projects for building and managing dependencies.

Maintaining Quality Project Information

Maven helps fetch project information from POM and other sources. The key project information that Maven can provide is the change-log document.

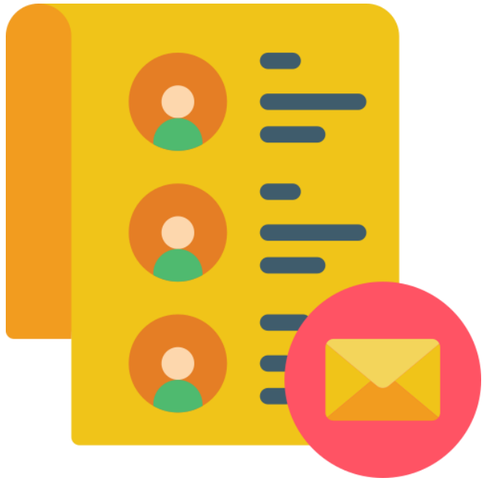


This is created from the source control and helps to fetch an updated list of project changes.

Maintaining Quality Project Information

Maven facilitates the maintenance of quality project information through various means, such as:

- Cross-referenced sources
- Dependency list
- Unit test reports



- This helps the project managers to create and manage subscriptions or unsubscriptions.
- The mailing list contains a list of the required stakeholders of the project.

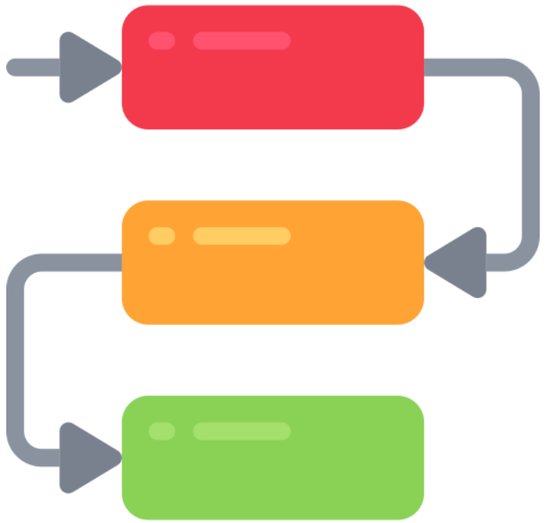
Maintaining Quality Project Information

Maven facilitates the maintenance of quality project information through various means, such as:

Cross-referenced sources

Dependency list

Unit test reports



- Dependencies refer to the artifacts the project needs to build and function effectively.
- Maven helps to manage direct dependencies through these dependencies.

Maintaining Quality Project Information

Maven facilitates the maintenance of quality project information through various means, such as:

Cross-referenced sources

Dependency list

Unit test reports

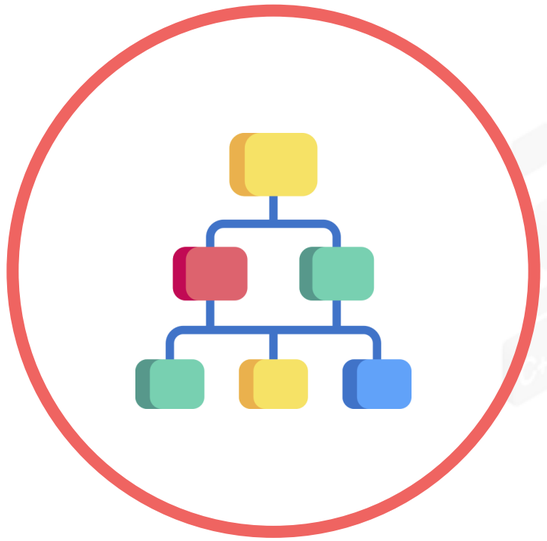
- Maven helps to execute the unit test suite packaged automatically.
- It also provides the unit test report, which is the cross-referenced results linked to the source code.

Convention over Configuration

Convention over Configuration

Maven uses convention over configuration in which developers don't have to create the build process.

It provides a sensible default behavior for projects.



It creates a default project structure.

The role of a developer is to place files as needed.

Convention over Configuration

Here are the default values for project source code files, resource files, and other configurations:

Assuming the ${root} denotes the project location:

Item	Default
Source code	${root}/src/main/java
Resources	${root}/src/main/resources
Tests	${root}/src/test
Compiled byte code	${root}/target
Distributable JAR	${root}/target/classes

Maven plugins are responsible for managing project developers' objectives and project dependency tasks.

Ant vs. Maven

Ant vs. Maven

Here are the differences between Ant and Maven:

Ant	Maven
It does not follow any formal conventions.	It follows convention to place the source or compile code.
It uses a build.xml file.	It uses a pom.xml file.
Users need to provide information about the project structure.	Users need not provide information about the project structure.
Ant scripts are not reusable.	Maven scripts are reusable.

Maven is preferred over Ant.

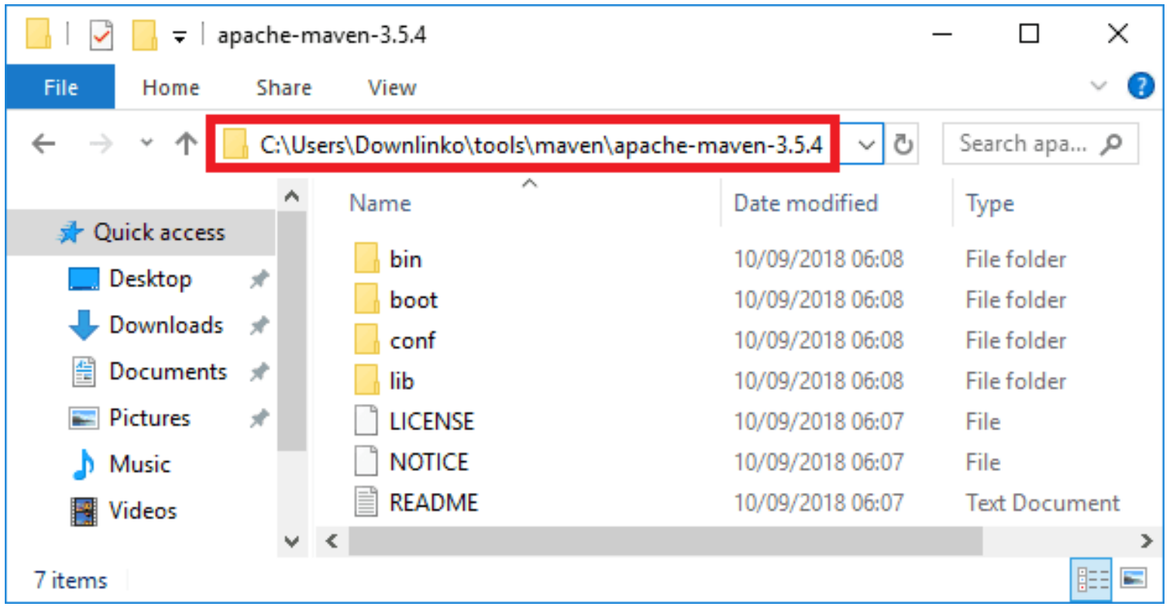
Installing Maven

Installing Maven

Maven can be downloaded and installed on Windows, Linux, and macOS platforms.

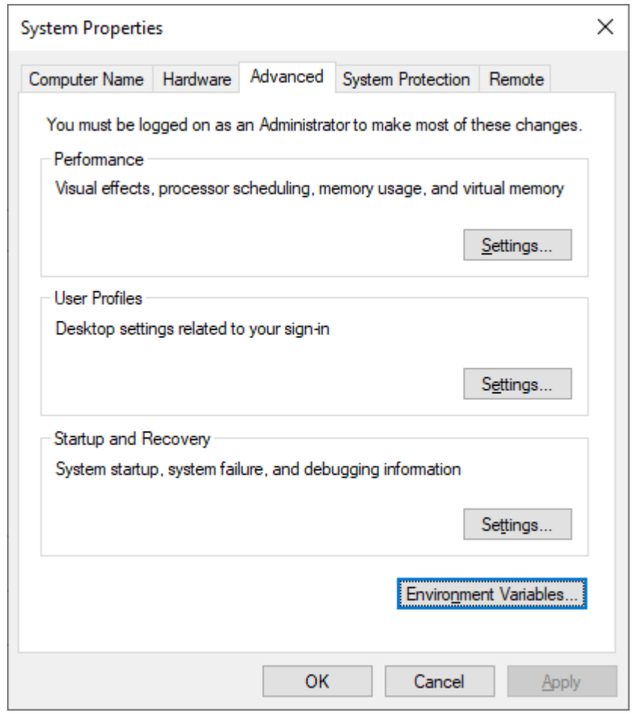
1. Download the latest version of Maven

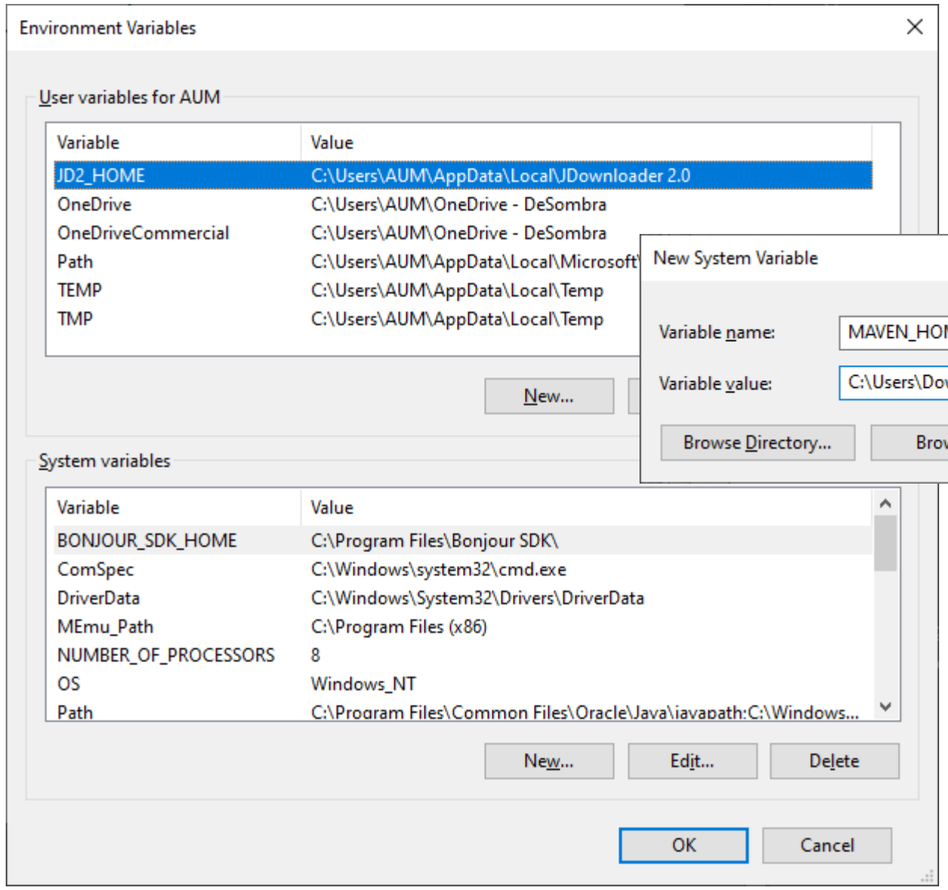
https://Maven.apache.org/download.cgi

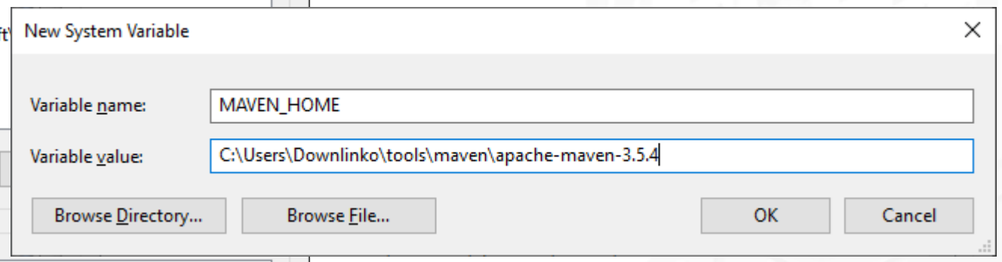


Installing Maven

2. Add Maven_HOME in the environment variable

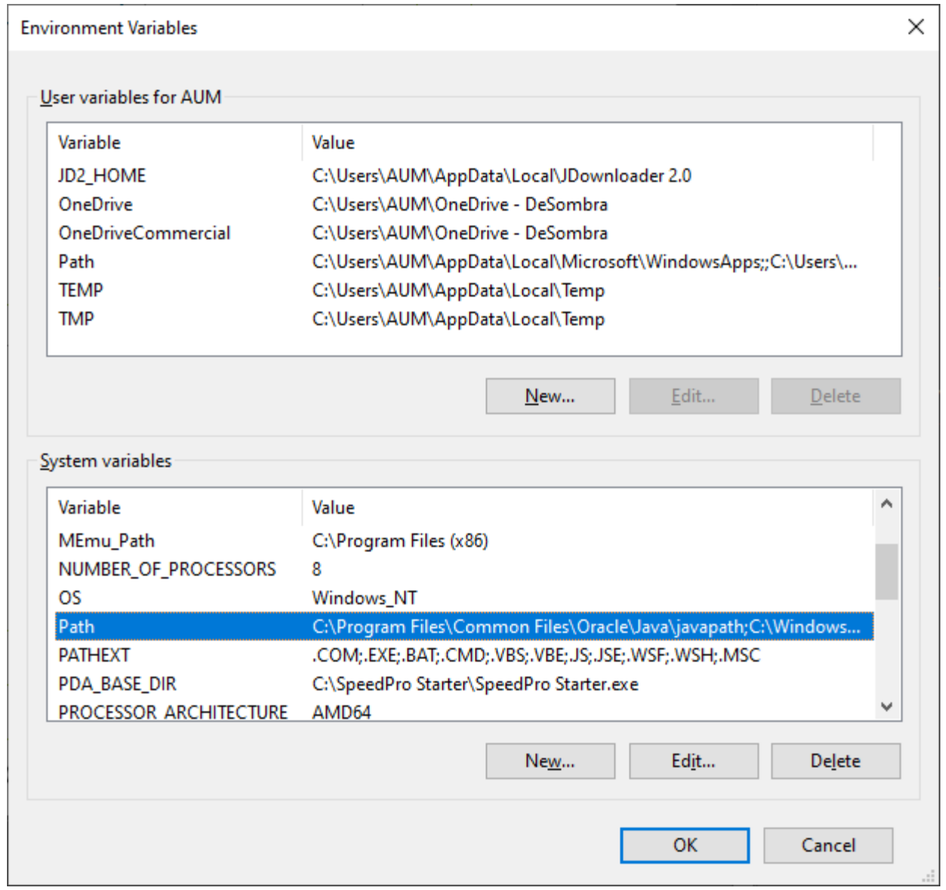


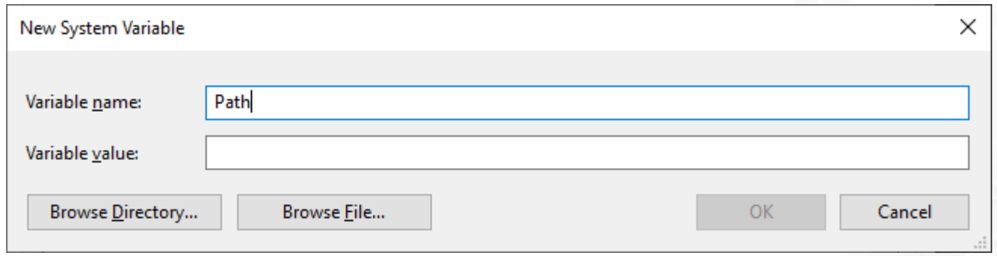




Installing Maven

3. Add Maven Path in the environment variable



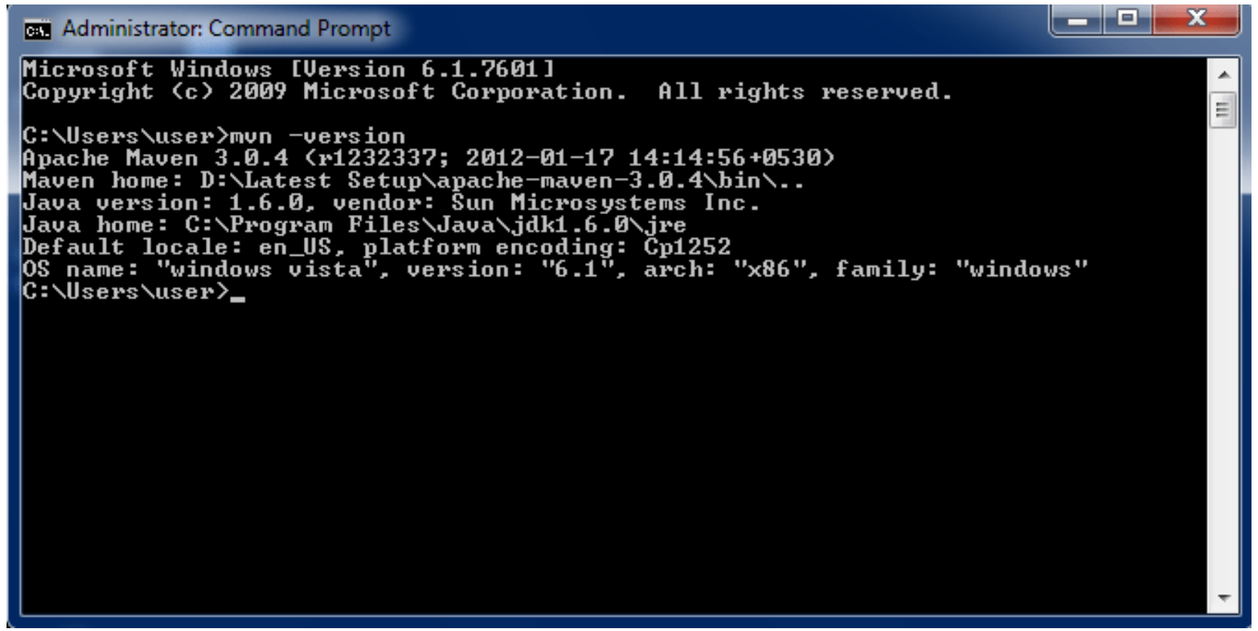


The path for Maven should be %Maven home%/bin.

Installing Maven

4. Verify Maven

Use mvn -version command to verify the Maven installation.



It displays the Maven and the JDK versions.

TECHNOLOGY

Maven Environment Setup

Maven Environment Setup: System Requirements

Maven requires the Java Development Kit (JDK) to be pre-installed.

Here are the minimum system requirements for Maven to function:

System requirements	
JDK	1.7 or the above
Memory	No minimum requirement
Disk space	No minimum requirement
Operating system	No minimum requirement

Maven Environment Setup: Steps

1. Install Java

Open the command prompt and execute the Java command **java -version**

Output:

```
java version"1.X.X_XX"
Java (™) SE Runtime Environment(build 1.X.X_XX)
```

To install Java, visit https://www.oracle.com/java/technologies/downloads/#jdk17-windows..

Maven Environment Setup: Steps

2. Set the JAVA environment variable

Set the JAVA_HOME environment path variable

↓

Add the Java compiler location to the system path

↓

Add the string C:\Program Files\Java\jdk1.X.X\bin
to the end of the system variable path

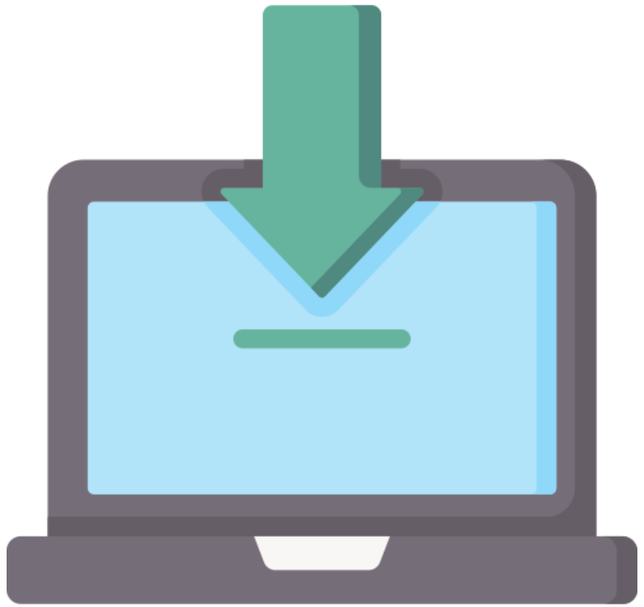
↓

Use the java-version command to verify the Java installation

Maven Environment Setup: Steps

3. Download the Maven archive

Download the Maven archive from https://Maven.apache.org/download.cgi.



Maven Environment Setup: Steps

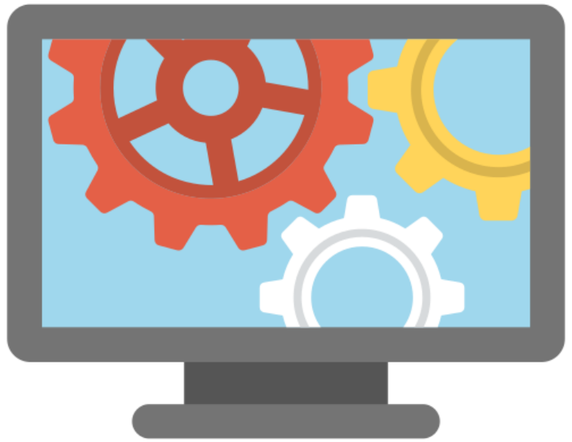
4. Extract the Maven archive

The Maven directory can be found in this location:

C:\Program Files\Apache SoftwareFoundation\apache-Maven-XXX

Maven Environment Setup: Steps

5. Set the Maven environment variables using system properties



CM2_HOME=C:\ProgramFiles\ApacheSoftwareFoundation\apache-Maven-XXX
M2=%M2_HOME%\bin Maven_OPTS=-Xms256m-Xmx512m

Maven Environment Setup: Steps

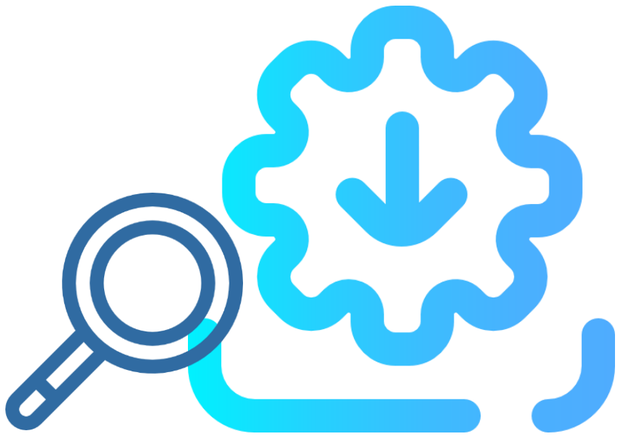
6. Add the Maven bin directory location to the system path

Maven Environment Setup: Steps

7. Verify the Maven installation

Open the terminal or command shell and execute the command:

```
mvn –version
```



Using Maven in Eclipse

Problem Statement:

You have been asked to explore Maven in Eclipse by installing the software and executing the project in different ways.

Outcome:

By exploring Maven in Eclipse through software installation and project execution, you'll gain proficiency in managing project dependencies and building Java applications efficiently. This hands-on experience enhances your ability to streamline project workflows and ensure consistent project management practices.

Note: Refer to the demo document for detailed steps: 01_Using_Maven_in_Eclipse

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

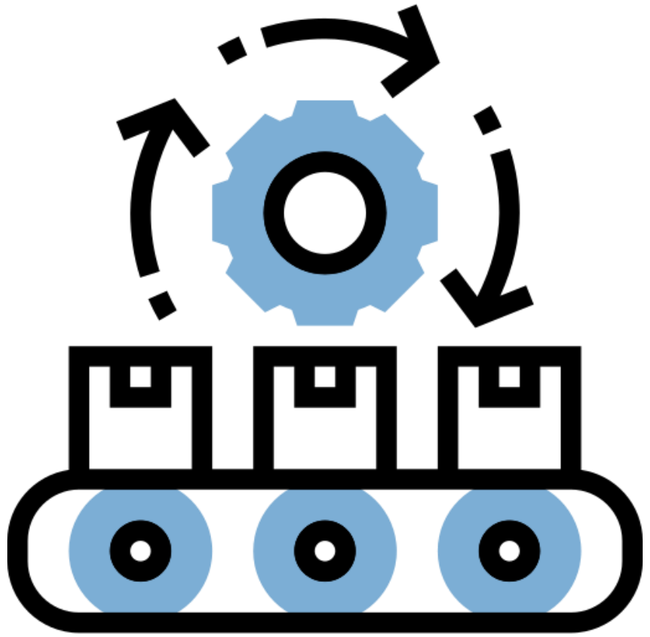
1. Install the software
2. Filter the Maven
3. Execute the Maven build
4. Run as Maven install
5. Create a Maven project
6. Add dependency in the pom.xml file

TECHNOLOGY

Maven Build Lifecycle

Maven Build Lifecycle

It is a sequence of phases that define the order in which tasks are executed during the build process.



It provides a structured and standardized sequence of phases that automate various aspects of the build process, from validating the project to deploying the final package.

Maven Build Lifecycle: Stages

The following are the stages in the lifecycle:

- Prepare resources
- Validate
- Compile
- Test

Initializes build state, sets up properties, and prepares the environment

Maven Build Lifecycle: Stages

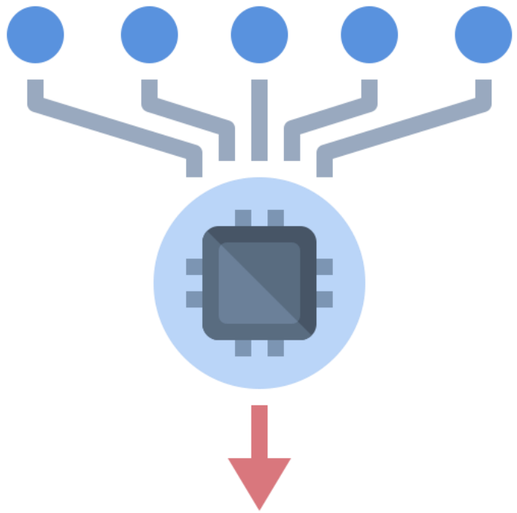
- Prepare resources
- **Validate**
- Compile
- Test

Validates the project's correctness and the availability of necessary information

Maven Build Lifecycle: Stages

- Prepare resources
- Validate
- Compile
- Test

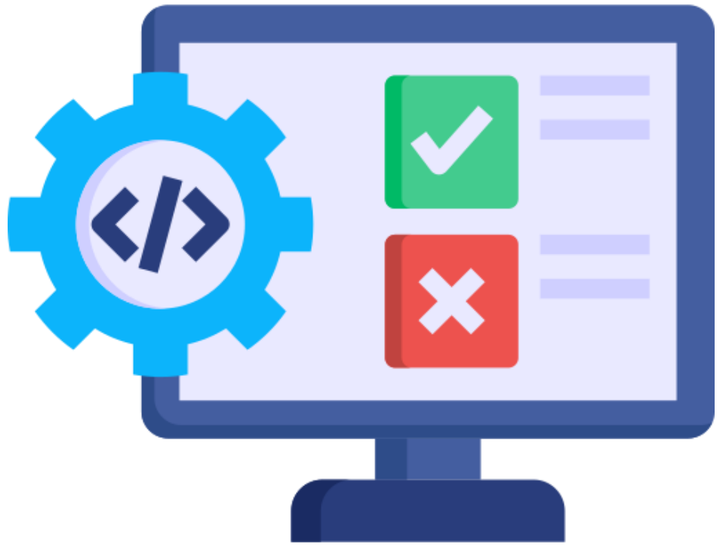
Compiles the source code



Maven Build Lifecycle: Stages

- Prepare resources
- Validate
- Compile
- Test

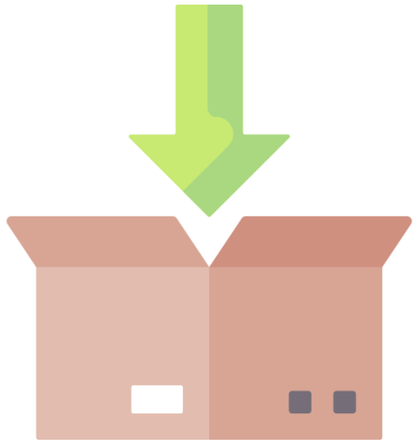
Tests the compiled source code



Maven Build Lifecycle: Stages

- Package
- Install
- Deploy
- Goals

Creates the JAR/WAR package available in POM.xml



Maven Build Lifecycle: Stages

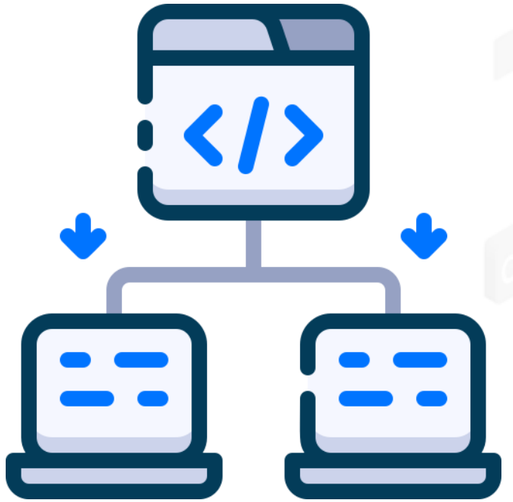
- Package
- Install
- Deploy
- Goals

Installs the package in the local or remote Maven repository

Maven Build Lifecycle: Stages

- Package
- Install
- Deploy
- Goals

Copies the final package to the remote repository



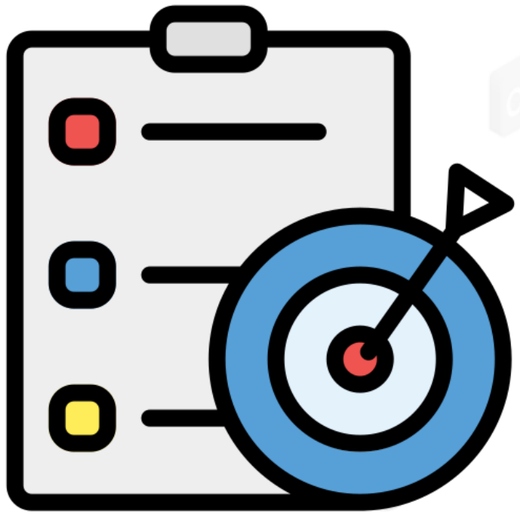
The defined sequence of the phases is used to execute goals.

Maven Build Lifecycle: Stages

- Package
- Install
- Deploy
- Goals

Points to a specific task that helps build and manage a project

Can be executed outside of the build lifecycle by direct invocation



Maven Build Lifecycle: Stages

- Package
- Install
- Deploy
- Goals

This command can be used to copy the project dependencies from the repository to a defined location.

```
mvn clean dependency:copy-dependencies
package
```

Execution:

```
clean phase → dependency:copy-dependencies
goal → package phase
```

Creating a Project Using Maven CLI

Problem Statement:

You have been asked to demonstrate how to create and execute a Maven application project using the CLI.

Outcome:

By demonstrating the creation and execution of a Maven application project using the command-line interface (CLI), you'll learn to initialize project structures, manage dependencies, and compile and run applications seamlessly. This knowledge is essential for efficiently building and managing Java projects outside of integrated development environments (IDEs).

Note: Refer to the demo document for detailed steps: 02_Creating_a_Project_Using_Maven_CLI

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Run the mvn package command
2. Open the CMS project

Using Maven CLI to Create Java Web Project

Problem Statement:

You have been asked to create a Java web project using Maven CLI and work with the Maven commands to configure and execute the project.

Outcome:

By using Maven CLI to create and manage a Java web project, you'll learn to efficiently set up dependencies and execute tasks, enhancing your ability to develop robust web applications with ease.

Note: Refer to the demo document for detailed steps: 03_Using_Maven_CLI_to_Create_Java_Web_Project

ASSISTED PRACTICE

Assisted Practice: Guidelines

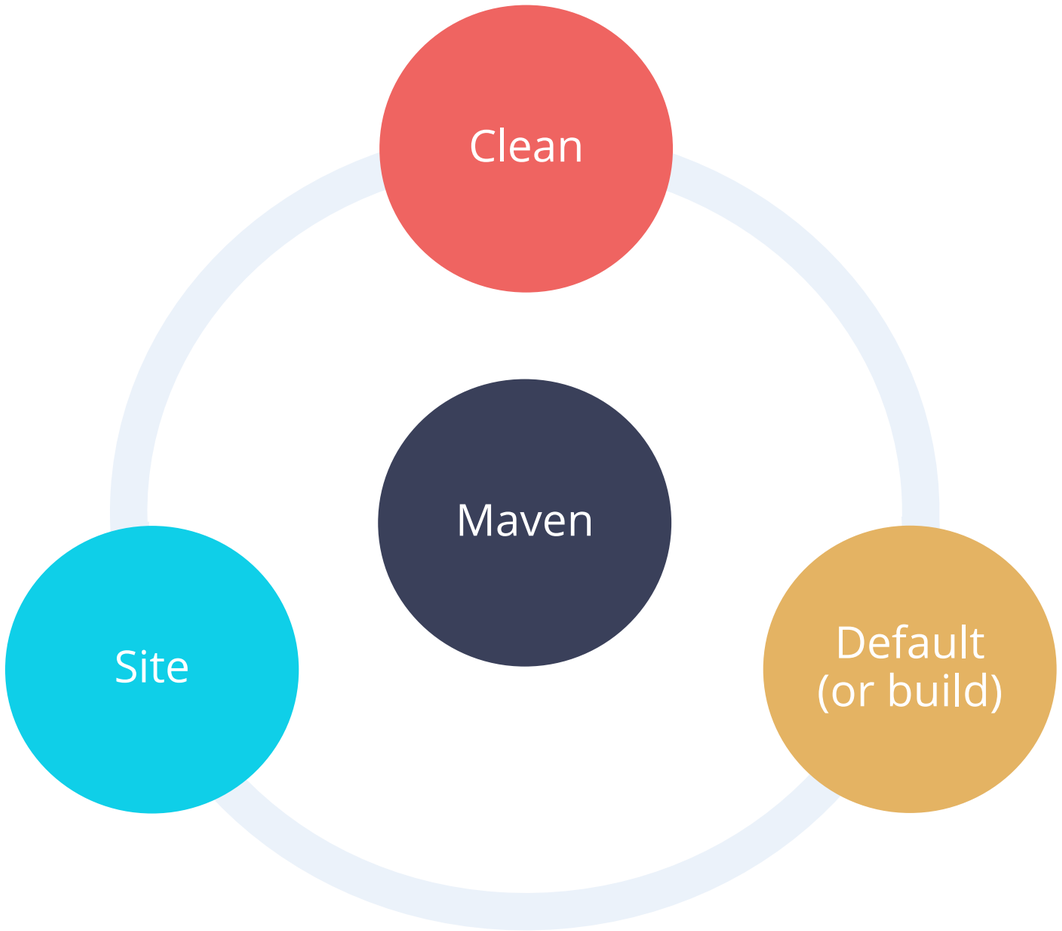
Steps to be followed are:

1. Run the mvn package command
2. Open the CMS project
3. Add the dependency

Maven Lifecycles

Maven Lifecycle

Maven consists of three standard lifecycles:



Clean Lifecycle

The clean lifecycle has three phases:

- Pre-clean
- Clean
- Post-clean

It helps to execute processes needed before the actual project cleaning.

Clean Lifecycle

The clean lifecycle has three phases:

Pre-clean	Clean	Post-clean

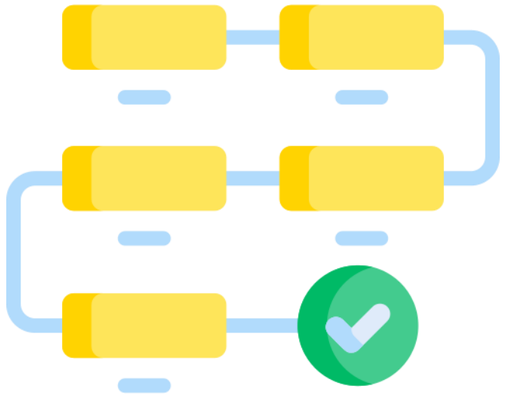
It helps to remove all the files generated by the previous build.

Clean Lifecycle

The clean lifecycle has three phases:

Pre-clean	Clean	Post-clean

It helps to execute processes required after project cleaning.



Default Lifecycle

The default or build lifecycle is employed to build an application.

It consists of twenty-three phases:

Phases	Description
validate	Verifies if the project is correct and whether all necessary information is available
initialize	Initializes build state
generate-sources	Generates any source code required for compilation
process-sources	Processes the source codes
generate-resources	Generates resources to include in a package
process-resources	Copies and processes the resources into the destination directory

Default Lifecycle

Phases	Description
compile	Compiles the source code for a project
process-classes	Post-processes the generated files from the compilation
generate-test-sources	Generates any test source code needed for compilation
process-test-sources	Copies and processes the test source code
generate-test-resources	Generates resources for testing

Default Lifecycle

Phases	Description
process-test-resources	Copies and processes the resources into the test destination directory
test-compile	Compiles the test source code into the test destination directory
process-test-classes	Post-processes the generated files from the test compilation
test	Runs tests with the right unit testing framework
prepare-package	Performs any operations required to prepare a package before the final packaging
package	Packages the compiled code in its distributable format
pre-integration-test	Performs actions needed before execution of integration tests

Default Lifecycle

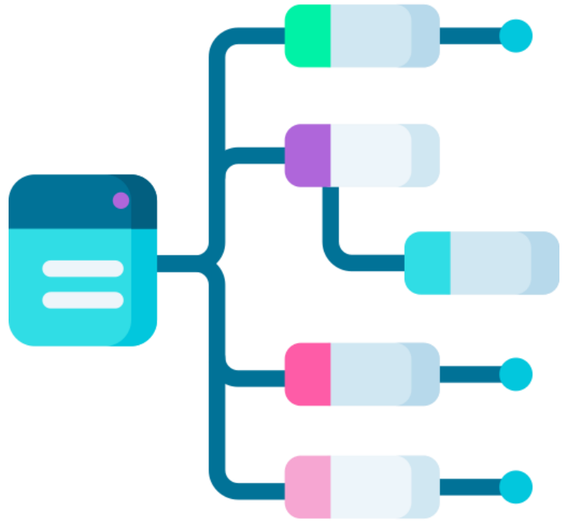
Phases	Description
integration-test	Processes and deploys the package to an environment where integration tests can be run
post-integration-test	Performs actions required after integration tests
verify	Runs any checks to ensure that the package is valid and the quality condition is met
install	Installs the package into the local repository
deploy	Deploy the final package to the remote repository

Site Lifecycle

The site lifecycle consists of four phases:

pre-site	site	post-site	site-deploy

Executes processes required before generating site documentation

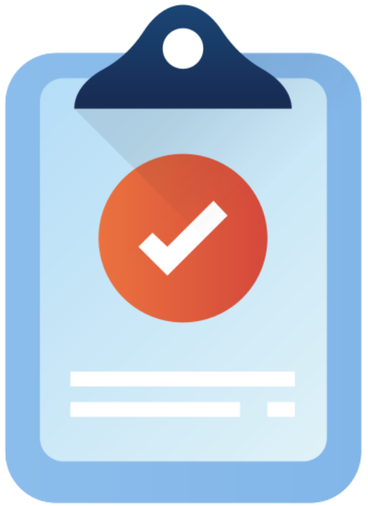


Site Lifecycle

The site lifecycle consists of four phases:

pre-site	**site**	post-site	site-deploy

Generates the project's documentation

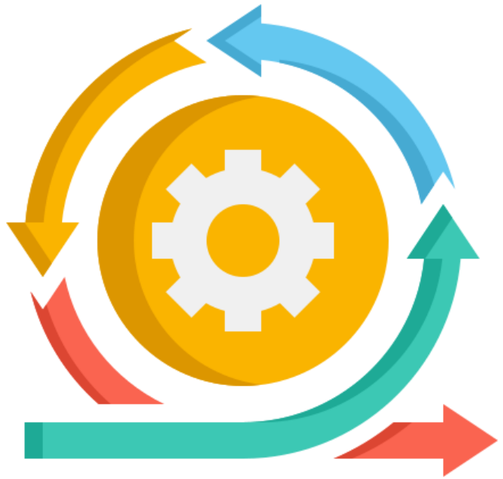


Site Lifecycle

The site lifecycle consists of four phases:

pre-site | site | **post-site** | site-deploy

Executes processes required after site generation



Site Lifecycle

The site lifecycle consists of four phases:

pre-site	site	post-site	site-deploy

Deploys the generated site documentation to the specified web server

Maven Repository

Maven Repository: Overview

The Maven repository is a directory of the packaged JAR file with the pom.xml file.

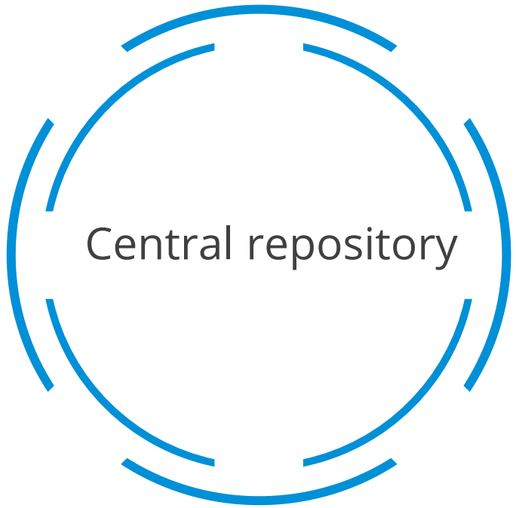
The Maven repository is categorized as:

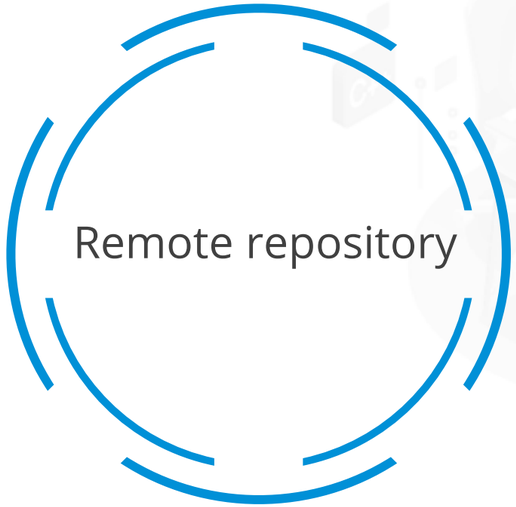
If it is unable to locate any dependency in these repositories, it stops processing and throws an error.

Maven Repository: Overview

It is a directory where project artifacts (such as libraries, plugins, and dependencies) are stored and from which they can be retrieved.

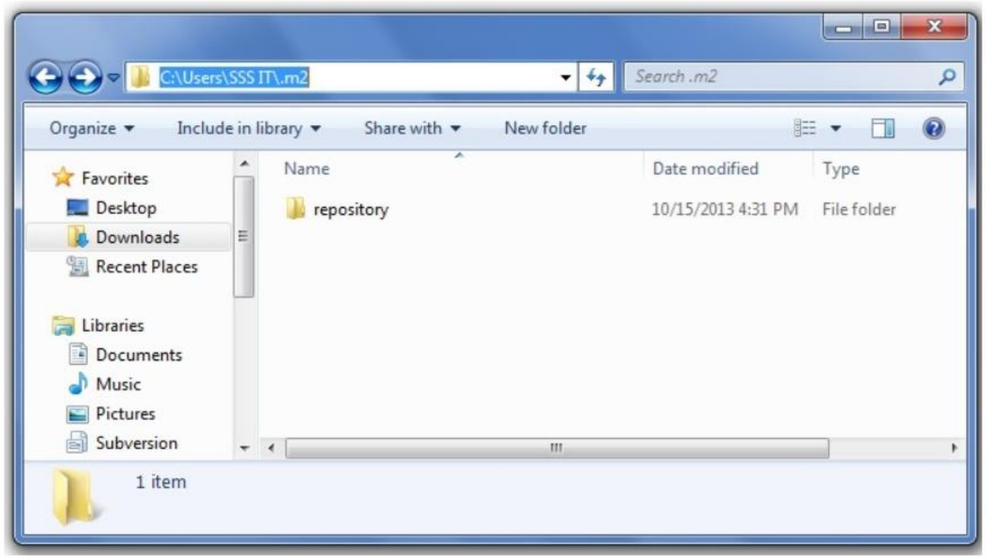
There are three main types of Maven repositories:





Local Repository

It is stored on the developer's machine, typically in the .m2 directory within the user's home directory.



When a build requires a dependency, Maven first looks in the local repository before searching remote repositories.

Central Repository

It is automatically accessed by Maven when dependencies are not found in the local repository.

It is hosted by the Maven community at repo.maven.apache.org.

Remote Repository

It is used to store custom or proprietary artifacts that are not available in the central repository.

It is defined in the pom.xml file or settings.xml file. Maven can download dependencies and upload artifacts to these repositories.

Working with Repositories

Problem Statement:

You have been asked to work with repositories and search for the Super POM file in the local repository.

Outcome:

By working with repositories and searching for the Super POM file in the local repository, you'll gain insight into Maven's dependency management system and the foundational structure of Maven projects.

Note: Refer to the demo document for detailed steps: 04_Working_with_Repositories

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Execute mvn command
2. Create a package in the file manager
3. List down files in the directory
4. Quit the configuration
5. Search for Super POM
6. Create command repository
7. Add ID and URL

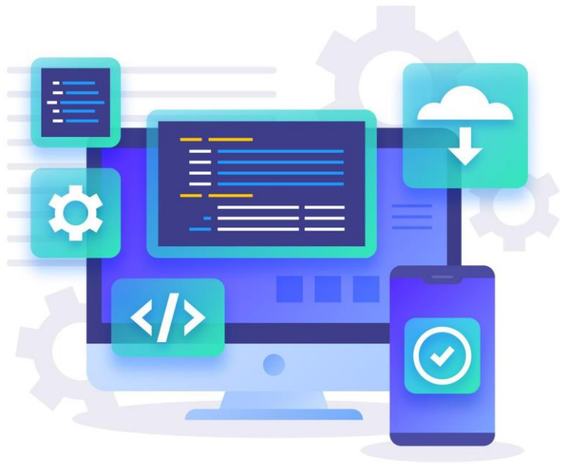
Dependency Scope

Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

Shows that the dependency is contained in the classpath of the project



Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

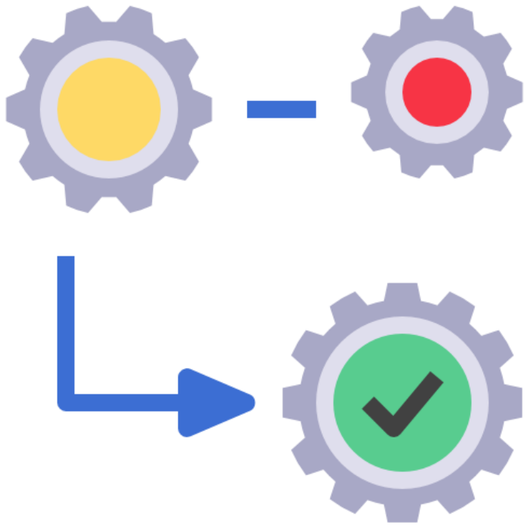
Shows that dependency is to be provided by JDK or a web server

Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

Shows that dependency is required during execution, not the compilation

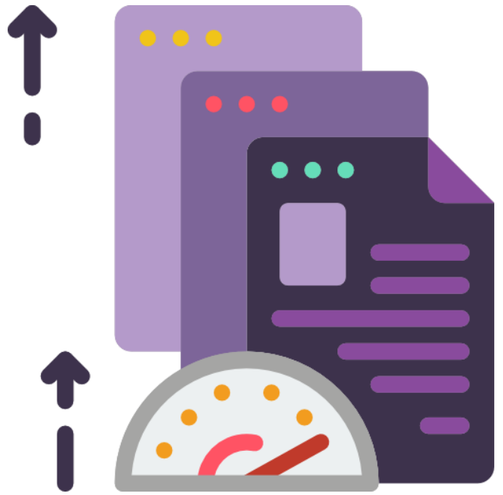


Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

Shows that dependency is only available for the test compilation and execution phases



Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

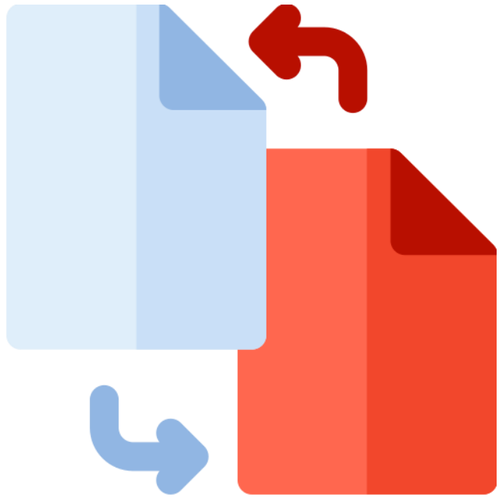
Indicates that the system path should be specified

Dependency Scope

Transitive dependencies can be restricted using various dependency scopes.

- Compile
- Provided
- Runtime
- Test
- System
- Import

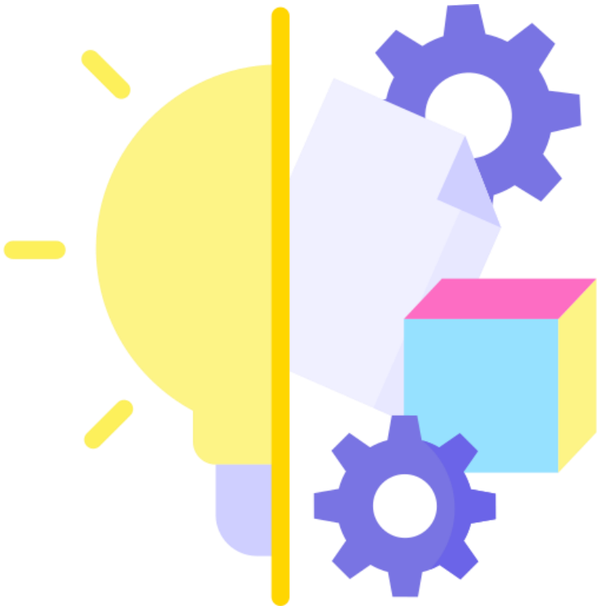
Indicates that POM must be replaced with the dependencies in the POM's <dependencyManagement> section



Maven Pom.xml

Maven Pom.xml

POM stands for Project Object Model.



- The pom.xml file contains information about the project.
- It also provides configuration information.

Maven reads the pom.xml file to execute the goal.

Elements to Create POM File

To create a pom.xml file, a few elements are used:

Elements	Description
Project	Refers to the root property of the pom.xml file
modelVersion	Refers to the sub-element of the project that specifies the modelVersion and set to 4.0.0
groupId	Refers to the sub-element of the project that helps to specify the ID for a project group
artifactId	Refers to the sub-property of the project property that specifies the ID for the artifact
Version	Refers to the sub-element of the project that specifies the version of the artifact in a specific group

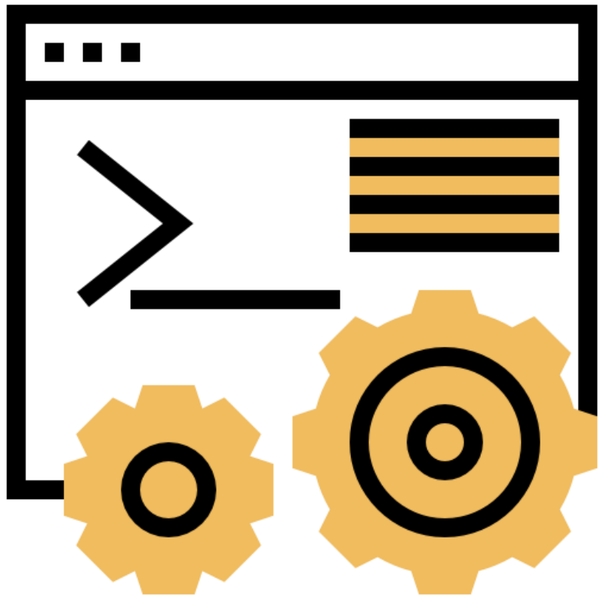
Maven Pom.xml: Syntax

```
<project xmlns="..............................."
xmlns:xsi="..................................."
xsi:schemaLocation="..............................
........">
<modelVersion>4.0.0</modelVersion>
<groupId>com.company.name</groupId>
<artifactId>app</artifactId>
<version>1</version>
</project>
```

Maven Web App

Maven Web App

To create a Maven web application, use the archetype: generate command and the Maven-archetype-webapp plugin.



Maven Web App: Syntax

```
C:\MVN> mvn archetype:generate
-DgroupId=groupid
-DartifactId=artifactid
-DarchetypeArtifactId=Maven-archetype-webapp
-DinteractiveMode=booleanValue
```

Maven Web App: Example

```
C:\MVN>mvn archetype:generate
-DgroupId = com.compname.education
-DartifactId = books
-DarchetypeArtifactId = Maven-archetype-webapp
-DinteractiveMode = false
```

Exploring Maven Clean Plugin

Problem Statement:

You have been asked to explore the Maven Clean Plugin by creating a target directory and removing the temp file from the dev directory.

Outcome:

You've been asked to explore the Maven Clean Plugin by creating a target directory and removing the temporary file from the development directory. This involves understanding how Maven plugins can automate project cleanup tasks to maintain project integrity and efficiency.

Note: Refer to the demo document for detailed steps: 05_Exploring_Maven_Clean_Plugin

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a target directory
2. Create a directory
3. Remove the temp file from the dev directory

Exploring Maven Compiler Plugin

Problem Statement:

You have been asked to explore the Maven Compiler Plugin to generate and compile the test classes.

Outcome:

By exploring the Maven Compiler Plugin to generate and compile test classes, you'll understand how Maven automates compilation tasks, ensuring the efficiency and reliability of test execution within your project.

Note: Refer to the demo document for detailed steps: 06_Exploring_Maven_Compiler_Plugin

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Perform maven compilation
2. Generate test class
3. Create a Java file
4. Compile the test

Configuring Maven Surefire Plugin

Problem Statement:

You have been asked to explore the Maven Surefire plugin by defining test cases and adding different configurations for test cases.

Outcome:

By exploring the Maven Surefire plugin and defining test cases with various configurations, you'll gain insight into how Maven automates the execution of unit tests. This exploration allows for comprehensive testing of your Java applications with different configurations, ensuring robustness and reliability in your test suite.

Note: Refer to the demo document for detailed steps: 07_Configuring_Maven_Surefire_Plugin

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Check Maven Surefire plugin
2. Define test cases
3. Test defined test cases
4. Add different configurations for test cases
5. Work with the security manager

Key Takeaways

- Apache Maven is a project management and building tool.
- Maven uses convention over configuration in which developers don't have to create the build process.
- Maven requires the Java Development Kit (JDK) to be pre-installed.
- The Maven Build Lifecycle refers to a sequence of phases that describes how the goals are to be executed or achieved.
- The Maven repository is a directory of the packaged JAR file with the pom.xml file.

TECHNOLOGY

Thank You

simplilearn