# **Apache Maven**

Last Updated : 17 Nov, 2023

**Maven** is a powerful project management tool that is based on POM (project object model). It is used for project build, dependency, and documentation. It simplifies the build process like ANT. But it is too much more advanced than ANT. In short terms we can tell maven is a tool that can be used for building and managing any Java-based project. maven makes the day-to-day work of Java developers easier and generally helps with the comprehension of any Java-based project.

## **What Is Maven?**

Maven is mostly used for the java projects to build web application packages. Maven provides different features which make it easy to build the web-application packages we manage complex projects easily.

1. **POM Files:** Project Object Model(POM) Files are XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project. When you should execute a maven command you give maven a POM file to execute the commands. Maven reads pom.xml file to accomplish its configuration and operations.
2. **Dependencies and Repositories:** Dependencies are external Java libraries required for Project and repositories are directories of packaged JAR files. The local repository is just a directory on your machine’s hard drive. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.
3. **Build Life Cycles, Phases, and Goals:** A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. Maven command is the name of a build lifecycle, phase, or goal. If a lifecycle is requested executed by giving the maven command, all build phases in that life cycle are executed also. If a build phase is requested executed, all build phases before it in the defined sequence are executed too.
4. **Build Profiles:** Build profiles a set of configuration values that allows you to build your project using different configurations. For example, you may need to build your project for your local computer, for development and test. To enable different builds you can add different build profiles to your POM files using its profiles elements which are triggered in a variety of ways.
5. **Build Plugins:** Build plugins are used to perform a specific goal. you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java.

## **What is Maven Life Cycle?**

Maven is an most powerful tool to build the artifacts of the java projects by using the POM.XML file. A build lifecycle in Apache Maven is a prearranged set of goals and phases that are carried out in a particular order to construct and oversee a project. Following are the

1. **Clean Lifecycle:** In this cycle maven will clean all the dependency which are previously stored in the local repository which have have been used for another project.

2. **Default Lifecycle:** This lifecycle had another sub cycles as mentioned below.

1. **validate:** Validates the project configuration.
2. **compile:** Compiles the source code into bytecode.
3. **test:** Runs the tests for the project.
4. **package:** Packages the compiled code and resources into an artifact (e.g., JAR, WAR).
5. **install:** Installs the artifact in the local repository.
6. **deploy:** Copies the artifact to a remote repository.

The above lifecycles can be achieved by the simple different maven commands like **mvn clean:clean or mvn compile**, or you can run an entire lifecycle using a command like **mvn clean install.**

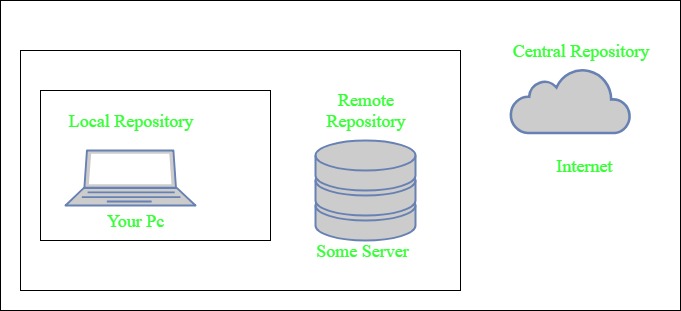
## **What is Maven In Java?**

If you want to use maven for your java project then first you need to install the maven java compiler after completion of installation you can create you first maven project for that you can use the following command.

mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart

## **What Is Maven Repository?**

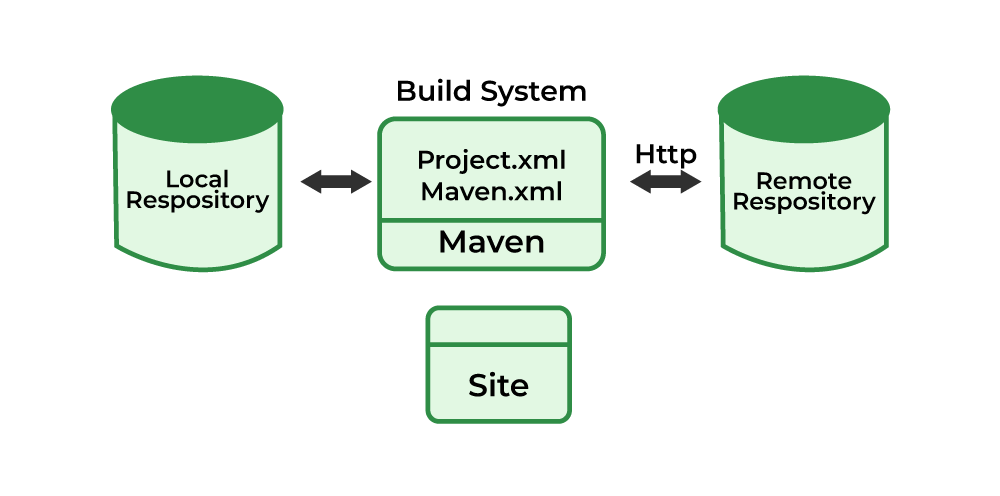
Maven repositories are directories of packaged JAR files with some metadata. The metadata are POM files related to the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has. This metadata enables Maven to download dependencies of your dependencies recursively until all dependencies are download and put into your local machine. Maven has three types of repository :Maven searches for dependencies in this repositories. First maven searches in Local repository then Central repository then Remote repository if Remote repository specified in the POM.



1. **Local repository-** A local repository is a directory on the machine of developer. This repository contains all the dependencies Maven downloads. Maven only needs to download the dependencies once, even if multiple projects depends on them (e.g. ODBC). By default, maven local repository is user\_home/m2 directory. example – **C:\Users\asingh\.m2**
2. **Central repository-** The central Maven repository is created Maven community. Maven looks in this central repository for any dependencies needed but not found in your local repository. Maven then downloads these dependencies into your local repository. You can view central repository by [this link](http://search.maven.org/#browse).
3. **Remote repository-** remote repository is a repository on a web server from which Maven can download dependencies.it often used for hosting projects internal to the organization. Maven then downloads these dependencies into your local repository.
   1. Maven can add all the dependencies required for the project automatically by reading pom file.
   2. One can easily build their project to jar, war etc. as per their requirements using Maven.
   3. Maven makes easy to start project in different environments and one doesn’t needs to handle the dependencies injection, builds, processing, etc.
   4. Adding a new dependency is very easy. One has to just write the dependency code in pom file.
   5. Maven needs the maven installation in the system for working and maven plugin for the ide.
   6. If the maven code for an existing dependency is not available, then one cannot add that dependency using maven.
   7. When there are a lot of dependencies for the project. Then it is easy to handle those dependencies using maven.
   8. When dependency version update frequently. Then one has to only update version ID in pom file to update dependencies.
   9. Continuous builds, integration, and testing can be easily handled by using maven.
   10. When one needs an easy way to Generating documentation from the source code, Compiling source code, Packaging compiled code into JAR files or ZIP files.

## **What Is Maven Architecture?**

Maven repository is a place where the maven artefacts or dependencies of the JAR file is going to store which are written in the file called the POM.XML. POM.XML contains the Java classes, resources, and other dependencies.There are the two types of repositories like



Maven reads the pom.xml file.Maven downloads the dependencies defined in the pom.xml file into the local repository from the central or remote repository.Maven executes the life cycles, phases, goals, and plugins defined in the pom.xml file.

## **What is Maven Used For?**

Maven can be used for the following:

1. We can easily build a project using maven.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven provides project information (log document, dependency list, unit test reports, etc.)
4. Maven is very helpful for a project while updating the central repository of JARs and other dependencies.
5. With the help of Maven, we can build any number of projects into output types like the JAR, WAR, etc without doing any scripting.
6. Using maven we can easily integrate our project with a source control systems (such as Subversion or Git).
7. Maven also helps in managing the project’s build lifecycle, including tasks like compiling, testing, packaging, and deploying the code.
8. Maven provides a standard project structure, making it easy for developers to understand the layout of the project and locate specific files.
9. Maven supports multi-module projects, allowing developers to work on multiple related projects simultaneously and manage their dependencies efficiently.
10. Maven plugins can be used to add additional functionality to the build process, such as code coverage analysis, static code analysis, and more.
11. Maven is highly customizable, allowing developers to configure the build process to meet their specific needs and requirements.
12. Maven simplifies the process of managing project dependencies, ensuring that the correct versions of libraries and frameworks are used throughout the project.

## **What Is Maven pom.xml File**

POM means Project Object Model is key to operate Maven. Maven reads pom.xml file to accomplish its configuration and operations. It is an XML file that contains information related to the project and configuration information such as **dependencies**, **source directory**, **plugin**, **goals etc**. used by Maven to build the project. The sample of 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.project.loggerapi </**groupId**>

<**artifactId**>LoggerApi</**artifactId**>

<**version**>0.0.1-SNAPSHOT</**version**>

<!-- Add typical dependencies for a web application -->

<**dependencies**>

<**dependency**>

<**groupId**>org.apache.logging.log4j</**groupId**>

<**artifactId**>log4j-api</**artifactId**>

<**version**>2.11.0</**version**>

</**dependency**>

</**dependencies**>

</**project**>

Elements used for Creating pom.xml file

1. **project-** It is the root element of the pom.xml file.
2. **modelVersion-** modelversion means what version of the POM model you are using. Use version 4.0.0 for maven 2 and maven 3.
3. **groupId-** groupId means the id for the project group. It is unique and Most often you will use a group ID which is similar to the root Java package name of the project like we used the groupId com.project.loggerapi.
4. **artifactId-** artifactId used to give name of the project you are building.in our example name of our project is LoggerApi.
5. **version-** version element contains the version number of the project. If your project has been released in different versions then it is useful to give version of your project.

## **What is Pom.xml file Key Components**

1. **dependencies-** dependencies element is used to defines a list of dependency of project.
2. **dependency-** dependency defines a dependency and used inside dependencies tag. Each dependency is described by its groupId, artifactId and version.
3. **name-** this element is used to give name to our maven project.
4. **scope-** this element used to define scope for this maven project that can be compile, runtime, test, provided system etc.
5. **packaging-** packaging element is used to packaging our project to output types like JAR, WAR etc.

# **Maven Build Phases and Basic Maven Commands**

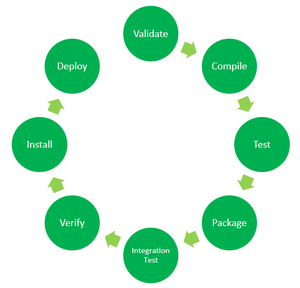
**Maven** is a powerful project management tool that is based on POM (project object model) and used for project build, dependency, and documentation. It is a tool that can be used for building and managing any Java-based project. Maven makes the day-to-day work of Java developers easier and helps with the building and running of any Java-based project. In this article, we will discuss Maven build phases and basic Maven commands.

For more details on how Maven works, how to install Maven and its applications, please visit: [Introduction to Apache Maven](https://www.geeksforgeeks.org/introduction-apache-maven-build-automation-tool-java-projects/)

## **Maven Build Phases**

**Maven build phases** are a sequence of steps or stages that Maven follows when executing a build lifecycle. These build phases are standardized and define the order in which Maven executes various goals. Each build phase represents a different stage in the lifecycle of a project.

Below are the standard Maven build phases:



* **Validate:** This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
* **Compile:** It compiles the source code, converts the .java files to .class, and stores the classes in the target/classes folder.
* **Test:** It runs unit tests for the project.
* **Package:** This step packages the compiled code in a distributable format like JAR or WAR.
* **Integration test:** It runs the integration tests for the project.
* **Verify:** This step runs checks to verify that the project is valid and meets the quality standards.
* **Install:** This step installs the packaged code to the local Maven repository.
* **Deploy:** It copies the packaged code to the remote repository for sharing it with other developers.

These build phases are executed sequentially by default when you run a Maven build.

In a Maven project, the build process begins with the Validate phase, where Maven validates the project structure. It ensures to Maven standards that the required plugins are available. Next, in the Compile phase, Maven compiles the Java source code into bytecode. Following compilation, the Test phase executes unit tests to validate the code’s functionality. Once tests pass, the Package phase bundles the compiled code, resources, and dependencies into a distributable format, like a JAR or WAR file. Finally, in the Install phase, Maven installs the packaged artifact into the local Maven repository and makes it available for other projects.

## **Maven Commands**

With a simple command-line interface, Maven manages project dependencies, compiles source code, runs tests, and packages applications. Some basic Maven commands includes:

* **mvn clean:** Cleans the project and removes all files generated by the previous build.
* **mvn compile:** Compiles source code of the project.
* **mvn test-compile:** Compiles the test source code.
* **mvn test:** Runs tests for the project.
* **mvn package:** Creates JAR or WAR file for the project to convert it into a distributable format.
* **mvn install:** Deploys the packaged JAR/ WAR file to the local repository.
* **mvn site:** generate the project documentation.
* **mvn validate:** validate the project’s POM and configuration.
* **mvn idea:idea:** generate project files for IntelliJ IDEA or Eclipse.
* **mvn release:perform:** Performs a release build.
* **mvn deploy:** Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.
* **mvn archetype:generate:** This command is used to generate a new project from an archetype, which is a template for a project. This command is typically used to create new projects based on a specific pattern or structure.
* **mvn dependency:tree:** This command is used to display the dependencies of the project in a tree format. This command is typically used to understand the dependencies of the project and troubleshoot any issues.

Generally when we run any of the above commands, we add the **mvn clean** step so that the target folder generated from the previous build is removed before running a newer build. This is how the command would look on integrating the *clean* step with *install* phase:

**mvn clean install**

Similarly, if we want to run the step in debug mode for more detailed build information and logs, we will add **-X** to the actual command. Hence, the *install* step with debug mode on will have the following command:

**mvn -X install**

Consider a scenario where we do not want to run the tests while packaging or installing the Java project. In this case, we use **-DskipTests** along with the actual command. If we need to run the *install* step by skipping the tests associated with the project, the command would be:

**mvn install -DskipTests**