**MAVEN**

What is Maven?

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following −

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* Mailing list

# **Maven - Environment Setup**

## **Step 1** - Verify Java Installation in Your Machine

First of all, open the console and execute a java command based on the operating system you are working on.

$ java –version

output

java 11.0.11 2021-04-20 LTS

Java(TM) SE Runtime Environment 18.9 (build 11.0.11+9-LTS-194)

Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.11+9-LTS-194, mixed mode)

## **Step 2** - Set JAVA Environment

Set the **JAVA\_HOME** environment variable to point to the base directory location where Java is installed on your machine.

export JAVA\_HOME = /usr/local/java-current

Append Java compiler location to the System Path.

export PATH = $PATH:$JAVA\_HOME/bin/

Verify Java installation using the command **java -version** as explained above.

## **Step 3** - Download Maven Archive

Download Maven 3.8.4 from <https://maven.apache.org/download.cgi>

apache-maven-3.8.4-bin.tar.gz

## **Step 4** - Extract the Maven Archive

Extract the archive, to the directory you wish to install Maven 3.8.4. The subdirectory apache-maven-3.8.4 will be created from the archive.

/usr/local/apache-maven

## **Step 5** - Set Maven Environment Variables

Add M2\_HOME, M2, MAVEN\_OPTS to environment variables.

Open command terminal and set environment variables.

export M2\_HOME=/usr/local/apache-maven/apache-maven-3.8.4 export M2=$M2\_HOME/bin

export MAVEN\_OPTS=-Xms256m -Xmx512m

## **Step 6** - Add Maven bin Directory Location to System Path

Now append M2 variable to System Path.

export PATH=$M2:$PATH

## **Step 7** - Verify Maven Installation

Now open console and execute the following **mvn** command.

$ mvn –version

**PROJECT OBJECT MODEL**

POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configuration that can be specified in the POM are following −

* project dependencies
* plugins
* goals
* build profiles
* project version
* developers
* mailing list

## What is Build Lifecycle?

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical **Maven Build Lifecycle** consists of the following sequence of phases.

* VALIDATE: This step validates if the project structure is correct. For ex: it checks if all the dependencies have been downloaded and are available in the local repository.
* COMPILE: It compiles the source code, convert the .java files to .class and stores the classes in target/classes folder.
* TEST: It runs unit tests for the project.
* PACKAGE: This step packages the compiled code in distributable format like JAR/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.

## **What is a Maven Repository?**

In Maven terminology, a repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types. The following illustration will give an idea regarding these three types.

* local
* central
* remote

## **Local Repository**

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies stored on remote machine every time a project is build.

## **Central Repository**

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL − <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows −

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

To browse the content of central maven repository, maven community has provided a URL − <https://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

## **Remote Repository**

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository**, which is developer's own custom repository containing required libraries or other project jars.

## **What are Maven Plugins?**

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to −

* create jar file
* create war file
* compile code files
* unit testing of code
* create project documentation
* create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax −

mvn [plugin-name]:[goal-name]

## **Plugin Types**

Maven provided the following two types of Plugins −

**Build plugins**

They execute during the build process and should be configured in the <build/> element of pom.xml.

**Reporting plugins**

They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.

## **MAVEN COMMANDS**

* **mvn compile:** This command is used to compile the project’s source code.
* **mvn clean:** Here, the project is cleaned to remove all previous-build files generated.
* **mvn test:** With this command, one can run project testing steps.
* **mvn test-compile:** This command is used to compile the code from the test source.
* **mvn install:** This command helps deploys the packaged WAR/JAR files storing them as classes in the local repository.
* **mvn package:** With this Maven lifecycle command, one packages or creates a project WAR or JAR file to be able to use a distributable format.
* **mvn deploy:** The deploy command occurs after compilation, running project tests and project building. Here the packaged WAR/JAR files are copied to the remote repository for use by other developers.