**MAVEN**

**What is Maven & How does it works?**

Maven The java build tool provided by Apache to help in the build, documentation and dependency process of projects with any level of complexity written in Java and C# that uses Project Object Model (POM) and that follows the convention of source code, compiling code and so on is called Maven.

The great majority of Maven users are going to call Maven a “build tool”: a tool used to build deployable artifacts from source code. Build engineers and project managers might refer to Maven as something more comprehensive: a project management tool. What is the difference? A build tool such as Ant is focused solely on preprocessing, compilation, packaging, testing, and distribution. A project management tool such as Maven provides a superset of features found in a build tool. In addition to providing build capabilities, Maven can also run reports, generate a web site, and facilitate communication among members of a working team.

**Why we use it ….**

Advantages of Maven Helps manage all the processes, such as building, documentation, releasing, and distribution in project management. Simplifies the process of project building. Increases the performance of the project and the building process

**Conceptual Model of a "Project"**

Maven maintains a model of a project. You are not just compiling source code into bytecode, you are developing a description of a software project and assigning a unique set of coordinates to a project. You are describing the attributes of the project. What is the project’s license? Who develops and contributes to the project? What other projects does this project depend upon? Maven is more than just a "build tool", it is more than just an improvement on tools like make and Ant, it is a platform that encompasses a new semantics related to software projects and software development. This definition of a model for every project enables such features as:

***Dependency Management Maven:*** The Complete Reference 5 / 316 Because a project is defined by a unique set of coordinates consisting of a group identifier, an artifact identifier, and a version, projects can now use these coordinates to declare dependencies.

***Remote Repositories:***

Related to dependency management, we can use the coordinates defined in the Maven Project Object Model (POM) to create repositories of Maven artifacts.

***Universal Reuse of Build Logic:***

Plugins contain logic that works with the descriptive data and configuration parameters defined in Project Object Model (POM); they are not designed to operate upon specific files in known locations.

***Tool Portability / Integration:***

Tools like Eclipse, NetBeans, and IntelliJ now have a common place to find information about a project. Before the advent of Maven, every IDE had a different way to store what was essentially a custom Project Object Model (POM). Maven has standardized this description, and while each IDE continues to maintain custom project files, they can be easily generated from the model.

***Easy Searching and Filtering of Project Artifacts:***

Tools like Nexus allow you to index and search the contents of a repository using the information stored in the POM.

What is pom.xml In Maven

That’s all you need in your pom.xml. Running mvn install from the command line will process resources, compile source, execute unit tests, create a JAR, and install the JAR in a local repository for reuse in other projects. Without modification, you can run mvn site and then find an index.html file in target/site that contains links to JavaDoc and a few reports about your source code. Admittedly, this is the simplest possible example project containing nothing more than some source code and producing a simple JAR. It is a project which closely follows Maven conventions and doesn’t require any dependencies or customization. If we wanted to start customizing the behavior, our pom.xml is going to grow in size, and in the largest of projects you can see collections of very complex Maven POMs which contain a great deal of plugin customization and dependency declarations. But, even when your project’s POM files become more substantial, they hold an entirely different kind of information from the build file of a similarly sized project using Ant. Maven POMs contain declarations: "This is a JAR project", and "The source code is in src/main/java". Ant build files contain explicit instructions: "This is project", "The source is in src/main/java", "Run javac against this directory", "Put the results in target/classes", "Create a JAR from the . . . .", etc. Where Ant had to be explicit about the process, there was something "built-in" to Maven that just knew where the source code was and how it should be processed.

**Apache Maven –** Maven has conventions. It knows where your source code is because you followed the convention.

Maven’s Compiler plugin put the bytecode in target/classes, and it produces a JAR file in target.

Maven is declarative. All you had to do was create a pom.xml file and put your source in the default directory.

Maven took care of the rest.

Maven has a lifecycle which was invoked when you executed mvn install. This command told Maven to execute a series of sequential lifecycle phases until it reached the install lifecycle phase. As a side-effect of this journey through the lifecycle, Maven executed a number of default plugin goals which did things like compile and create a JAR.

Maven has built-in intelligence about common project tasks in the form of Maven plugins. If you wanted to write and execute unit tests, all you would need to do is write the tests, place them in ${basedir}/src/test/java

add a test-scoped dependency on either TestNG or JUnit, and run mvn test. If you wanted to deploy a web application and not a JAR, all you would need to do is change your project type to war and put your docroot in ${basedir}/src/main/webapp. Sure, you can do all of this with Ant, but you will be writing the instructions from scratch. In Ant, you would first have to figure out where the JUnit JAR file should be. Then you would have to create a classpath that includes the JUnit JAR file. Then you would tell Ant where it should look for test source code, write a goal that compiles the test source to bytecode, and execute the unit tests with JUnit. Without supporting technologies like antlibs and Ivy (even with these supporting technologies), Ant has the feeling of a c`ustom procedural build. An efficient set of Maven POMs in a project which adheres to Maven’s assumed conventions has surprisingly little XML compared to the Ant alternative. Another benefit of Maven is the reliance on widely-shared Maven plugins. Everyone uses the Maven Surefire plugin for unit testing, and if someone adds support for a new unit testing framework, you can gain new capabilities in your own build by just incrementing the version of a particular Maven plugin in your project’s POM. The decision to use Maven or Ant isn’t a binary one, and Ant still has a place in a complex build. If your current build contains some highly customized process, or if you’ve written some Ant scripts to complete a specific process in a specific way that cannot be adapted to the Maven standards, you can still use these scripts with Maven. Ant is made available as a core Maven plugin. Custom Maven plugins can be implemented in Ant, and Maven projects can be configured to execute Ant scripts within the Maven project lifecycle

**<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.javatpoint.application1**</groupId>**

**<artifactId>**my-app**</artifactId>**

**<version>**1**</version>**

**</project>**

**Prerequisites**

We assume you are going to use Maven to handle enterprise level Java projects development. So it is beneficial to have the knowledge of software development, Java SE, overview of Java EE development and deployment process.

**Environment Set-Up**

<https://www.tutorialspoint.com/maven/maven_environment_setup.htm>

### **Downloading Apache Maven**

Apache Maven 3.6.0 is the latest release and recommended version for all users from here: <https://maven.apache.org/downloadable>.

**Basic Maven Commands:**

The following are the basic commands of which you should be aware.

***Mvn –version:***

This command helps us in knowing the current version of Maven that is installed

***Creating a project:***

To create a project using MVN architecture below maven command should be used.

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

The templates that maven uses are called archetypes. By using the above command Maven will get all archetypes. It will do all configurations and will generate a working project. This architecture can be simplified by providing an archetypeArtifactId property which helps in creating apps easily.

***MVN package:***

This command is used to execute all Maven phases until the package phase. It does the job of compiling, verifying and building the project. It builds the jar file and places it in the specified folder under the specified project.

***mvn clean install:***

This maven command helps in executing a clean build life cycle and installs build phase in the default build cycle. This build life cycles may have its build phases and inside each build, there are different build goals. Also, this ensures that the build target is being removed for a new build and adds the clean target.

***mvn compile:***

This command is used to compile the source code. It also compiles the classes that are stored at a particular target or class.

***mvn test:***

Maven also provides the facility of unit testing particular codes. It runs the tests using suitable [testing frameworks](https://www.educba.com/testing-frameworks-for-java/).

***mvn deploy:***

Maven also has the facility of deploying the code for the project. This deployment is done in an integration or release environment. It copies all final package to the remote repository and it becomes available for sharing with other developers.

***mvn site :***

This command builds a site which is based upon the information on the project’s pom.

the document that is generated can be seen under the target/site.

**Run a maven build job**

Once the maven project is complete, the build activities like compiling code, running tests and packaging are done here. Running the below command will take care of all these activities.

***maven.javadoc.skip:***

This command is used to skip the generation and packaging of any specified Javadoc jar file.

### **Intermediate Commands**

Let us discuss the intermediate commands.

***mvn help: describe***– This command helps in getting the attributes of the plugin. All information regarding the plugin can be found by this maven command.

***mvn help:effective-pom***– By making use of this command a user can get an effective POM as an XML. This XML can be used for current build and for active profiles which are present in the project.

***mvn dependency: analyze***– To analyze the dependencies that are present in the project in respect to prints that are unused, outdated dependencies and so on this command can be used.

***mvn dependency: tree***– This command prints the complete tree of dependencies of the complete project. This is useful in getting all the transitive dependencies and gets all conflicts if there are any due to version. This command brings all the different dependencies.

***source. skip*:** This maven command skips the complete packaging of source jars in the project.

***maven.test. skip***– If a user wants to skip a particular test script from compilation and the execution of all tests then this command can be used. It can skip both unit and integration for the test for any application

***groups={TestNG Group Name(s)}***– This command specifies the [TestNG group](https://www.educba.com/install-testng/) of unit tests that are present and which will be executed in the current build. It runs the entire group of the test. If there is no group specified then all the unit tests will be executed.

***prepare-package***– Using this maven command all activities are performed for any operation which is required to prepare any package.

***pre-integration-test***– It performs all activities which are necessary for the integration tests which will be executed. It may also clean up the environment.

***mvn help:effective-pom***– Run this command to get the most minimal Maven project pom.xml.

### **Advanced Commands**

Below is the list of advanced commands.

***Resuming builds:*** To get this option working and resume the builds the -rf or -resume option can be used. It can be used if a user is working with a large multi-module project and the user wants to restart the build any project.

***Making a subset of projects:*** When a user specifies the -am option Maven builds all the different projects that a specified project has. It can have projects that are dependent on the main project directly or indirectly.

***Creating a Maven Wrapper***: There are two way to create a maven Wrapper

cd {your-project}

mvn -N io.takari:maven:wrapper

This command helps in creating a [Maven wrapper](https://www.educba.com/install-maven/) for a particular project with the latest available Maven versions

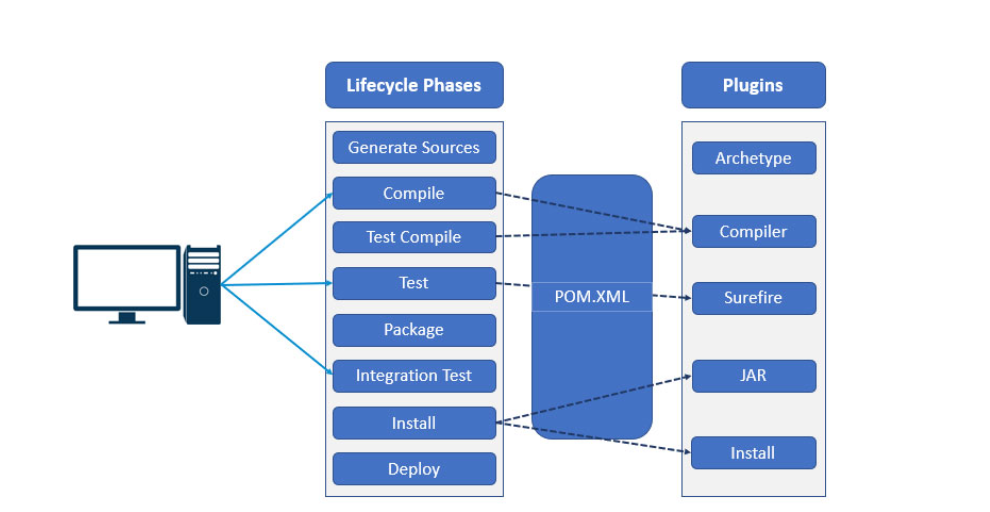
### **Tips and Tricks to Use Command**

***Maven pl option:*** This command is used to build specific reactor projects. There is no need to run the entire project.

***Debugging Unit tests:***The maven-surefire-plugin helps in running unit tests. This plugin invokes the test phase and also helps in building the life cycle.

***Maven Life Cycle***

<https://www.educba.com/maven-life-cycle/?source=leftnav>



**What is Maven plugin**

<https://www.educba.com/maven-plugins/?source=leftnav>

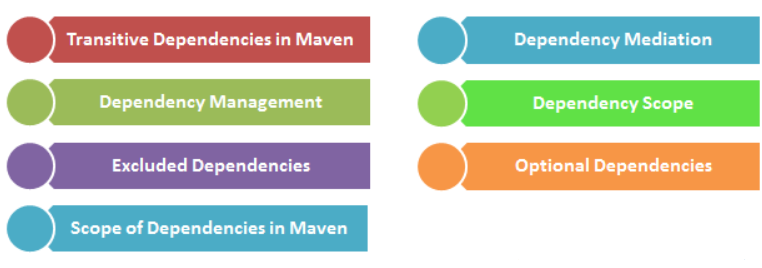
**Introduction to Maven Dependency Scope**

Maven is a building tool that is most often used in java projects for defining and maintaining the build process. Besides that maven provides us with full-proof dependency management features that helps in building a single module and multi-module building of projects and deploying the same. [Maven has maintained](https://www.educba.com/what-is-maven/) a central repository where all the jars and javadocs are available and can be added by adding the dependency tag corresponding to your dependency in your pom.xml. Besides that, there are some dependencies of the dependencies that you have added in your pom.xml. This is called transitive dependencies that are automatically added by maven. In this article, we will discuss transitive dependencies and various scope of dependency in maven.

**Types of Dependency Scope in Maven**

There are six different dependency scopes used in the maven that will be described one by one in the upcoming session.

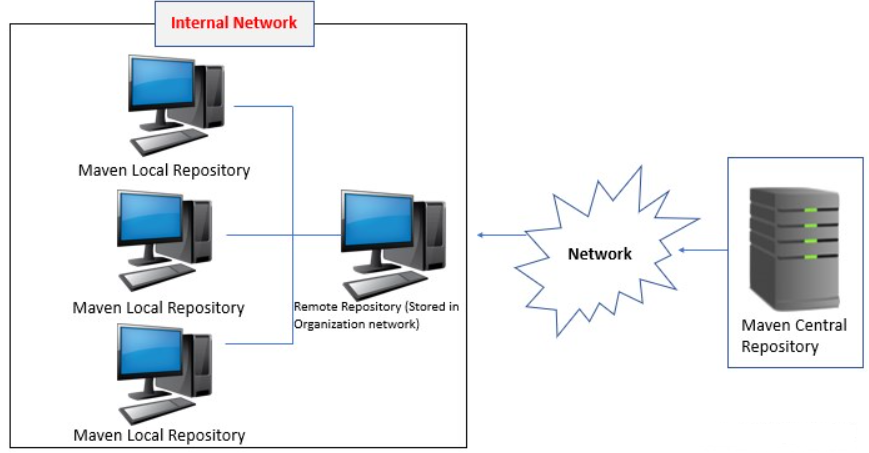
<https://www.educba.com/maven-dependency-scope/?source=leftnav>



**Introduction to Maven Repository**

Maven repository is a directory where all the packages, JAR files, plugins or any other artifacts are stored with POM.xml. Repository in maven holds build artifacts and dependencies of various types. It provides three types of repositories.

<https://www.educba.com/maven-repository/?source=leftnav>



**Maven Deploy**

<https://www.educba.com/maven-deploy/?source=leftnav>

**References**

1. <https://www.educba.com/maven-interview-questions/?source=leftnav>
2. [https://books.sonatype.com/mvnref-book/reference/index.html#:~:text=While%20there%20are%20a%20number%20of%20references%20for,Maven%2C%20%22Maven%20by%20Example%22%20and%20this%20reference%20book.](https://books.sonatype.com/mvnref-book/reference/index.html%23:~:text=While%20there%20are%20a%20number%20of%20references%20for,Maven%2C%20%22Maven%20by%20Example%22%20and%20this%20reference%20book.)
3. <https://maven.apache.org/pom.html>
4. <https://www.tutorialspoint.com/maven/maven_tutorial.pdf>
5. <https://maven.apache.org/general.html>