Reference : <http://tutorials.jenkov.com/maven/maven-tutorial.html>

Description : Maven is most popular project and dependency managment tool(build tool) for java application. It has lot of command and option to help us during our application developemnt.

**Installing Maven**

To install Maven on your own system (computer), go to the [**Maven download page**](http://maven.apache.org/download.cgi) and follow the instructions there. In summary, what you need to do is:

1. Set the JAVA\_HOME environment variable to point to a valid Java SDK (e.g. Java 8).
2. Download and unzip Maven.
3. Set the M2\_HOME environment variable to point to the directory you unzipped Maven to.
4. Set the M2 environment variable to point to M2\_HOME/bin (%M2\_HOME%\bin on Windows, $M2\_HOME/bin on unix).
5. Add M2 to the PATH environment variable (%M2% on Windows, $M2 on unix).
6. Open a command prompt and type 'mvn -version' (without quotes) and press enter.

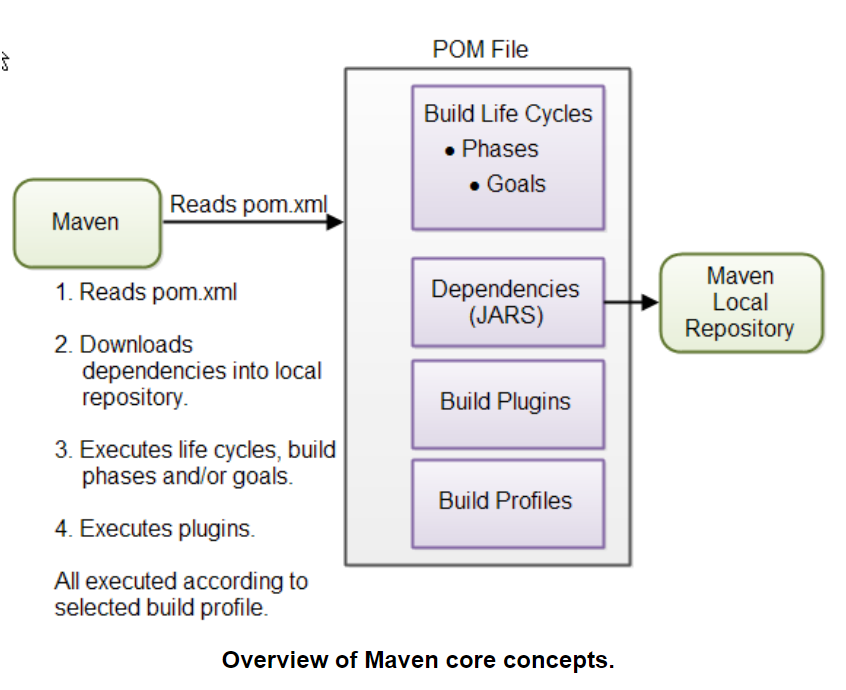
After typing in the mvn -version command you should be able to see Maven execute, and the version number of Maven written out to the command prompt.

Note: Maven uses Java when executing, so you need Java installed too (and the JAVA\_HOME environment variable set as explained above). Maven 3.0.5 needs a Java version 1.5 or later. I use Maven 3.3.3 with Java 8 (u45).

## Maven Overview - Core Concepts

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



|  |
| --- |
| **Overview of Maven core concepts.** |

These concepts are explained briefly below to give you an overview, and then in more detail in their own sections later in this tutorial.

**POM Files**  
When you execute a Maven command you give Maven a POM file to execute the commands on. Maven will then execute the command on the resources described in the POM.

**Build Life Cycles, Phases and Goals**  
The build process in Maven is split up into 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. When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal. If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too

**Dependencies and Repositories**  
One of the first goals Maven executes is to check the dependencies needed by your project. Dependencies are external JAR files (Java libraries) that your project uses. 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. The local repository is just a directory on your computer's hard disk. You can specify where the local repository should be located if you want to (I do). You can also specify which remote repository to use for downloading dependencies. All this will be explained in more detail later in this tutorial.

**Build Plugins**  
Build plugins are used to insert extra goals into a build phase. If you need to perform a set of actions for your project which are not covered by the standard Maven build phases and goals, 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 if you need to.

**Build Profiles**  
Build profiles are used if you need to build your project in different ways. For instance, you may need to build your project for your local computer, for development and test. And you may need to build it for deployment on your production environment. These two builds may be different. To enable different builds you can add different build profiles to your POM files. When executing Maven you can tell which build profile to use.

## Maven POM Files

A Maven POM file (Project Object Model) is an XML file that describe the resources of the project. This includes the directories where the source code, test source etc. is located in, what external dependencies (JAR files) your projects has etc.

Here is a minimal POM file:

<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.jenkov</groupId>

<artifactId>java-web-crawler</artifactId>

<version>1.0.0</version>

</project>

### **Super POM**

All Maven POM files inherit from a super POM. If no super POM is specified, the POM file inherits from the base POM. Here is a diagram illustrating that:

|  |
| --- |
| Super POM and POM inheritance. |
| **Super POM and POM inheritance.** |

You can make a POM file explicitly inherit from another POM file. That way you can change the settings across all inheriting POM's via their common super POM. You specify the super POM at the top of a POM file like this:

<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>

**<parent>**

**<groupId>org.codehaus.mojo</groupId>**

**<artifactId>my-parent</artifactId>**

**<version>2.0</version>**

**<relativePath>../my-parent</relativePath>**

**</parent>**

<artifactId>my-project</artifactId>

...

</project>

An inheriting POM file may override settings from a super POM. Just specify new settings in the inheriting POM file.

### **Effective POM**

With all this POM inheritance it may be hard to know what the total POM file looks like when Maven executes. The total POM file (result of all inheritance) is called the *effective POM*. You can get Maven to show you the effective POM using this command:

mvn help:effective-pom

This command will make Maven write out the effective POM to the command line prompt.

**Maven Settings File**

Maven has two settings files. In the settings files you can configure settings for Maven across all Maven POM files. For instance, you can configure:

* Location of local repository
* Active build profile
* Etc.

The settings files are called settings.xml. The two settings files are located at:

* The Maven installation directory: $M2\_HOME/conf/settings.xml
* The user's home directory: ${user.home}/.m2/settings.xml

Both files are optional. If both files are present, the values in the user home settings file overrides the values in the Maven installation settings file.

## Running Maven

Running Maven is done by executing the mvn command from a command prompt. When executing the mvn command you pass the name of a [**build life cycle, phase or goal**](http://tutorials.jenkov.com/maven/maven-tutorial.html#maven-build-life-cycles-phases-and-goals) to it, which Maven then executes. Here is an example:

mvn install

This command executes the build phase called install (part of the default build life cycle), which builds the project and copies the packaged JAR file into the local Maven repository. Actually, this command executes all build phases before install in the build phase sequence, before executing the install build phase.

You can execute multiple build life cycles or phases by passing more than one argument to the mvn command. Here is an example:

mvn clean install

This command first executes the clean build life cycle, which removes compiled classes from the Maven output directory, and then it executes the install build phase.

## Maven Directory Structure

Maven has a standard directory structure. If you follow that directory structure for your project, you do not need to specify the directories of your source code, test code etc. in your POM file.

You can see the full directory layout in the [**Introduction to the Maven Standard Directory Layout**](http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html).

Here are the most important directories:

- src

- main

- java

- resources

- webapp

- test

- java

- resources

- target

## Project Dependencies

Unless your project is small, your project may need external Java APIs or frameworks which are packaged in their own JAR files. These JAR files are needed on the classpath when you compile your project code

Luckily, Maven has built-in dependency management. You specify in the POM file what external libraries your project depends on, and which version, and then Maven downloads them for you and puts them in your local Maven repository. If any of these external libraries need other libraries, then these other libraries are also downloaded into your local Maven repository.

Example

**<dependencies>**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.8.1</version>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

## External Dependencies

An external dependency in Maven is a dependency (JAR file) which is not located in a Maven repository (neiterh local, central or remote repository). It may be located somewhere on your local hard disk, for instance in the lib directory of a webapp, or somewhere else. The word "external" thus means external to the Maven repository system - not just external to the project. Most dependencies are external to the project, but few are external to the repository system (not located in a repository).

You configure an external dependency like this:

<dependency>

<groupId>mydependency</groupId>

<artifactId>mydependency</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>${basedir}\war\WEB-INF\lib\mydependency.jar</systemPath>

</dependency>

**Maven Repositories**

Maven repositories are directories of packaged JAR files with extra meta data. The meta data are POM files describing the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has. It is this meta data that enables Maven to download dependencies of your dependencies recursively, until the whole tree of dependencies is download and put into your local repository.

Maven repositories are covered in more detail in the [**Maven Introduction to Repositories**](http://maven.apache.org/guides/introduction/introduction-to-repositories.html), but here is a quick overview.

Maven has three types of repository:

* Local repository
* Central repository
* Remote repository

Maven searches these repositories for dependencies in the above sequence. First in the local repository, then in the central repository, and third in remote repositories if specified in the POM.

Here is a diagram illustrating the three repository types and their location:

|  |
| --- |
| Maven Repository Types and Location. |
| **Maven Repository Types and Location.** |

**Local Repository**  
A local repository is a directory on the developer's computer. This repository will contain all the dependencies Maven downloads. The same Maven repository is typically used for several different projects. Thus Maven only needs to download the dependencies once, even if multiple projects depends on them (e.g. Junit).

Your own projects can also be built and installed in your local repository, using the mvn install command. That way your other projects can use the packaged JAR files of your own projects as external dependencies by specifying them as external dependencies inside their Maven POM files.

By default Maven puts your local repository inside your user home directory on your local computer. However, you can change the location of the local repository by setting the directory inside your Maven settings file. Your Maven settings file is also located in your user-home/.m2 directory and is called settings.xml. Here is how you specify another location for your local repository:

<settings>

<localRepository>

d:\data\java\products\maven\repository

</localRepository>

</settings>

**Central Repository**

The central Maven repository is a repository provided by the Maven community. By default 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 need no special configuration to access the central repository.

**Remote Repository**  
A remote repository is a repository on a web server from which Maven can download dependencies, just like the central repository. A remote repository can be located anywhere on the internet, or inside a local network.

A remote repository is often used for hosting projects internal to your organization, which are shared by multiple projects. For instance, a common security project might be used across multiple internal projects. This security project should not be accessible to the outside world, and should thus not be hosted in the public, central Maven repository. Instead it can be hosted in an internal remote repository.

Dependencies found in a remote repository are also downloaded and put into your local repository by Maven.

You can configure a remote repository in the POM file. Put the following XML elements right after the <dependencies> element: Like Nexus

<repositories>

<repository>

<id>jenkov.code</id>

<url>http://maven.jenkov.com/maven2/lib</url>

</repository>

</repositories>

## Maven Build Life Cycles, Phases and Goals

When Maven builds a software project it follows a build life cycle. The build life cycle is divided into build phases, and the build phases are divided into build goals. Maven build life cycles, build phases and goals are described in more detail in the [**Maven Introduction to Build Phases**](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html), but here I will give you a quick overview.

**Build Life Cycles**  
Maven has 3 built-in build life cycles. These are:

1. default
2. clean
3. site

The default life cycle handles everything related to compiling and packaging your project. The clean life cycle handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc. The site life cycle handles everything related to generating documentation for your project. In fact, site can generate a complete website with documentation for your project.

**Build Phases**  
Each build life cycle is divided into a sequence of build phases, and the build phases are again subdivided into goals. Thus, the total build process is a sequence of build life cycle(s), build phases and goals.

|  |  |
| --- | --- |
| Build Phase | Description |
| validate | Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded. |
| compile | Compiles the source code of the project. |
| test | Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed. |
| package | Packs the compiled code in its distributable format, such as a JAR. |
| install | Install the package into the local repository, for use as a dependency in other projects locally. |
| deploy | Copies the final package to the remote repository for sharing with other developers and projects. |

You execute one of these build phases by passing its name to the mvn command. Here is an example:

mvn package

This example executes the package build phase, and thus also all build phases before it in Maven's predefined build phase sequence.

## Maven Build Profiles

Maven build profiles enable you to build your project using different configurations. Instead of creating two separate POM files, you can just specify a profile with the different build configuration, and build your project with this build profile when needed.

You can read the full story about build profiles in the Maven POM reference under [**Profiles**](http://maven.apache.org/pom.html#Profiles). Here I will give you a quick overview though.

Maven build profiles are specified inside the POM file, inside the profiles element. Each build profile is nested inside a profile element. Here is an example:

<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.jenkov.crawler</groupId>

<artifactId>java-web-crawler</artifactId>

<version>1.0.0</version>

<profiles>

<profile>

<id>test</id>

<activation>...</activation>

<build>...</build>

<modules>...</modules>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<dependencies>...</dependencies>

<reporting>...</reporting>

<dependencyManagement>...</dependencyManagement>

<distributionManagement>...</distributionManagement>

</profile>

</profiles>

</project>

A build profile describes what changes should be made to the POM file when executing under that build profile. This could be changing the applications configuration file to use etc. The elements inside the profile element will override the values of the elements with the same name further up in the POM.

Inside the profile element you can see a activation element. This element describes the condition that triggers this build profile to be used. One way to choose what profile is being executed is in the settings.xml file. There you can set the active profile. Another way is to add -P profile-name to the Maven command line. See the profile documentation for more information.