**Maven** is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.

It simplifies the build process like ANT. But it is too much advanced than ANT.

Current version of Maven is 3.

## Understanding the problem without Maven

There are many problems that we face during the project development.

* **Adding set of Jars in each project**
* **Creating the right project structure**
* **Building and Deploying the project**

**Maven simplifies the above mentioned problems. It does mainly following tasks.**

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross-referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate to new features of Maven

**Apache Maven helps to manage**

* Builds
* Documentation
* Reporting
* SCMs
* Releases
* Distribution

**A build tool takes care of everything for building a process**. It does the following:

* Generates source code (if auto-generated code is used)
* Generates documentation from source code
* Compiles source code
* Packages compiled code into JAR of ZIP file
* Installs the packaged code in local repository, server repository, or central repository

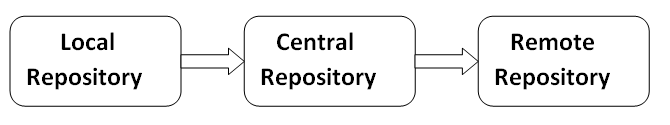
# Maven Repository

A **maven repository** is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

**Local repository** then **Central repository** then **Remote repository**.



If the dependency is not found in these repositories, maven stops processing and throws an error.

Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is %USER\_HOME%/.m2 directory. For example: **C:\Users\[USERNAME] \.m2**

## Update location of Local Repository

We can change the location of maven local repository by changing the **settings.xml** file. It is located in **MAVEN\_HOME/conf/settings.xml**, for example: **E:\apache-maven-3.1.1\conf\settings.xml**.

# Maven pom.xml file

**POM** is an acronym for **Project Object Model**. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

## Elements of maven pom.xml file

| **Element** | **Description** |
| --- | --- |
| **project** | It is the root element of pom.xml file. |
| **modelVersion** | It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0. |
| **groupId** | It is the sub element of project. It specifies the id for the project group. |
| **artifactId** | It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs. |
| **version** | It is the sub element of project. It specifies the version of the artifact under given group. |

## Maven pom.xml file with additional elements

Here, we are going to add other elements in pom.xml file such as:

| **Element** | **Description** |
| --- | --- |
| **packaging** | defines packaging type such as jar, war etc. |
| **name** | defines name of the maven project. |
| **url** | defines url of the project. |
| **dependencies** | defines dependencies for this project. |
| **dependency** | defines a dependency. It is used inside dependencies. |
| **scope** | defines scope for this maven project. It can be compile, provided, runtime, test and system. |

**<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.mycompany**</groupId>**

**<artifactId>**myecommerce**</artifactId>**

**<version>**1.0**</version>**

**<packaging>**jar**</packaging>**

**<name>**Maven Quick Start Archetype**</name>**

**<url>**http://maven.apache.org**</url>**

**<dependencies>**

**<dependency>**

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

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

**<version>**4.8.2**</version>**

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

**</dependency>**

**</dependencies>**

**</project>**

# Maven Archetype

We can create a simple maven example by executing the **archetype:generate** command of **mvn tool**.

To create a simple java project using maven, you need to open command prompt and run the **archetype:generate** command of mvn tool.

mvn archetype:generate -DgroupId=groupid -DartifactId=artifactid

-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=booleanValue

myecommerce

-src

--main

---java

----com

-----mycompany

------App.java

--test

---java

----com

-----javatpoint

------AppTest.java

-pom.xml

## How to build the maven project or how to package maven project?

The **mvn package** command completes the build life cycle of the maven project such as:

1. validate
2. compile
3. test
4. package
5. integration-test
6. verify
7. install
8. deploy

mvn clean install

mvn package

[http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html)

# Maven Plugins

The **maven plugins** are central part of maven framework, it is used to perform specific goal.

According to Apache Maven, there are 2 types of maven plugins.

1. Build Plugins
2. Reporting Plugins

#### Build Plugins

These plugins are executed at the time of build. These plugins should be declared inside the **<build>** element.

#### Reporting Plugins

These plugins are executed at the time of site generation. These plugins should be declared inside the **<reporting>** element.

## Maven Core Plugins

A list of maven core plugins are given below:

| **Plugin** | **Description** |
| --- | --- |
| clean | clean up after build. |
| compiler | compiles java source code. |
| deploy | deploys the artifact to the remote repository. |
| failsafe | runs the JUnit integration tests in an isolated classloader. |
| install | installs the built artifact into the local repository. |
| resources | copies the resources to the output directory for including in the JAR. |
| site | generates a site for the current project. |
| surefire | runs the JUnit unit tests in an isolated classloader. |
| verifier | verifies the existence of certain conditions. It is useful for integration tests. |

## List of Maven Plugins

Below are list of plugins

[http://repo.maven.apache.org/maven2/org/apache/maven/plugins/](https://repo.maven.apache.org/maven2/org/apache/maven/plugins/).

Maven plugins are also available outside the maven at **codehaus.org** and **code.google.com**.

[Dependency Management](http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Management) allows to consolidate and centralize the management of dependency versions without adding dependencies which are inherited by all children. This is especially useful when you have **a set of projects** (i.e. more than one) that inherits a common parent.

Another extremely important use case of dependencyManagement is the control of versions of artifacts used in transitive dependencies.

