Maven is a s/w management and comprehension tool based on the concept of Project Object Model which can manage build, reporting and documentation from central piece of information.

Role of build tools:

* Compile source code
* Copy Resource
* Compile and Run test
* Package project
* Deploy project
* Clean up

POM:

It is an XML file that contains information about configuration

* Describes project
* Name .Version ,Artifact Type,Scouce code location,Dependencies
* Plugins
* Profiles (alternate build configurations)

Maven objectives:

* Making build process easy
* Providing a uniform build system
* Providing quality project information
* Providing guidelines for best practices development
* Allowing transaparent migration to new features

2.

Maven installation and path setting - MVN –version

Creating project at command prompt

c\> mvn archetype:generate

Ex2:

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

cd my-app

* Display project structure and POM.xml

mvn package

java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App

* Hello World!

4.creating project in Eclipse ID

5. cleaning project : rightclick on project -> Run as -> Maven Clean

In command prompt

C\> mvn clean

6.compiling project : right click on project -> run as -> Maven build... -> goals give compile

C\> mvn compile

7.compile Testing cases : right click on project -> run as -> Maven build... -> goals give test- compile (need to add junit for build path)

C\> mvn test- compile

8.Testing project : right click on project -> run as -> Maven build... -> goals give test

(need to add junit for build path)

C\> mvn test

9.Creating a jar file : right click on project -> run as -> Maven build... -> goals give install

Or directly right click on project -> run as -> Maven

10. Creating jar file for a maven project ====🡺 mvn install

right click on project -> run as -> Maven build... -> goals give install

run as -> Maven install

11. Deploy a maven project ====🡺 mvn deploy

right click on project -> run as -> Maven build... -> goals give deploy

run as -> Maven deploy

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.8.2</version>

<executions>

<execution>

<id>deploy-file</id>

<phase>deploy</phase>

<goals>

<goal>deploy-file</goal>

</goals>

<configuration>

<file><!-- path-to-file --></file>

<url><!-- url-of-the-repository-to-deploy --></url>

<groupId><!-- group-id --></groupId>

<artifactId><!-- artifact-id --></artifactId>

<version><!-- version --></version>

<packaging><!-- type-of-packaging --></packaging>

</configuration>

</execution>

</executions>

</plugin>

Maven build life cycle:

* *validate:* check if all information necessary for the build is available
* *compile:* compile the source code
* *test-compile:* compile the test source code
* *test:* run unit tests
* *package:* package compiled source code into the distributable format (jar, war, …)
* *integration-test:* process and deploy the package if needed to run integration tests
* *install:* install the package to a local repository
* *deploy:* copy the package to the remote repository

Maven : dependencies

Maven :Exclusions

<exclusions><exclusion>

<group-id><group-id>

<artifactId>< artifactId >

</exclusion>

</ exclusions>

Maven : Scopes : limiting the scope of dependencies

<scope>test</scope>

<scope>compile</scope> --default

<scope>runtime</scope>

<scope>provided</scope> - compile , container to provide runtime

**Dependency Scope**

Dependency scope is used to limit the transitivity of a dependency and to determine when a dependency is included in a classpath.

There are 6 scopes:

* **compile**  
  This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.
* **provided**  
  This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes
* **runtime**  
  This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.
* **test**  
  This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.
* **system**  
  This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
* **import**  
  This scope is only supported on a dependency of type pom in the <dependencyManagement> section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

Creating war 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>CrunchifyTutorial</groupId>

<artifactId>CrunchifyTutorial</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

</dependency>

</dependencies>

</project>

Right click on project -> run as -> Maven build... -> goal as - clean install