**­Maven 3.0.3**

**Download & Installation:-**

* Download Maven 3.0.3 (Binary zip) from <http://maven.apache.org/download.html>
* Extract it to your choice drives or directory(Say C:)
* Set the following variable in the environment variable

path=C:\apache-maven-3.0.3\bin

JAVA\_HOME=C:\Program Files\Java\jdk1.6.0\_11

* Open CMD and type the following

C:\>mvn –version

* You will get java version,and maven version,which shows maven is installed successfully in your machine

**What is Maven?**

It is a build tool that can be used for building and managing any Java-based project.

Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system.

As a developer, we need to write the entries on a file called POM.xml,which contains all the information about

* which project to build(jar,ear,war)
* how to build the project(where are the source file, where it needs to be copied)
* what are the dependent jar file needed to build the project,and whether that jar file is needed only to compile the resources or needed to execute also

**MAVEN REPOSITORY**

Maven maintains a repository where all the dependencies of the project will be stored.Dependencies means needed jar files and plugins

By default the repository will be created under “C:\Documents and Settings\<Your\_User>\.m2” directory

The repository will be created after you execute pom.xml or any plugins

You can change the location of the repository by modifying C:\apache-maven-3.0.3\conf\settings.xml

<localRepository>d:/mavenRepository</localRepository>

Will set the repository in D:

**Life Cycle of Maven**

Maven has 3 Life Cycle.Each Life Cycle contains number of Phases

* Clean
* default
* site

***Clean:-***This Life Cycle contains 3 phases

|  |  |
| --- | --- |
| Pre-clean | Executes processes needed prior to the actual project cleaning |
| Clean | Remove all the file generated in previous build |
| Post-clean | Executes processes needed to finalize the project cleaning |

***Default:*** This Life Cycle contains 23 phases

|  |  |  |
| --- | --- | --- |
| 1 | validate | validate the project is correct and all necessary information is available. |
| 2 | initialize | initialize build state, e.g. set properties or create directories. |
| 3 | generate-sources | generate any source code for inclusion in compilation. |
| 4 | process-sources | process the source code, for example to filter any values. |
| 5 | generate-resources | generate resources for inclusion in the package. |
| 6 | process-resources | copy and process the resources into the destination directory, ready for packaging. |
| 7 | compile | compile the source code of the project. |
| 8 | process-classes | post-process the generated files from compilation,for example to do bytecode enhancement on Java classes. |
| 9 | generate-test-sources | generate any test source code for inclusion in compile |
| 10 | process-test-sources | process the test source code, for example to filter any values. |
| 11 | generate-test-resources | create resources for testing. |
| 12 | process-test-resources | copy and process the resources into the test destination directory. |
| 13 | test-compile | compile the test source code into the test destination directory |
| 14 | process-test-classes | post-process the generated files from test compilation,  for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above. |
| 15 | test | run tests using a suitable unit testing framework.These tests should not require the code be packaged or deployed. |
| 16 | prepare-package | perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above) |
| 17 | package | take the compiled code and package it in its distributable format, such as a JAR. |
| 18 | pre-integration-test | perform actions required before integration tests are executed. This may involve things such as setting up the required environment. |
| 19 | integration-test | process and deploy the package if necessary into an environment where integration tests can be run. |
| 20 | post-integration-test | perform actions required after integration tests have been executed. This may including cleaning up the environment. |
| 21 | verify | run any checks to verify the package is valid and meets quality criteria. |
| 22 | install | install the package into the local repository, for use as a dependency in other projects locally. |
| 23 | deploy | done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects. |

**Site:**It contains 4 phases

|  |  |
| --- | --- |
| Pre-site | executes processes needed prior to the actual project site generation |
| Site | generates the project's site documentation |
| Post-site | executes processes needed to finalize the site generation, and to prepare for site deployment |
| Site-deploy | deploys the generated site documentation to the specified web server |

**Sample POM.xml**

<**project** xmlns="...maven-v4\_0\_0.xsd">

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**org.nats**</groupId>**

**<artifactId>**JspApp**</artifactId>**

**<packaging>**war**</packaging>**

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

**<name>**JspApp Maven Webapp**</name>**

**<build>**

**<finalName>**JSPApp**</finalName>**

**<directory>**test**</directory>**

**<outputDirectory>**test/classes/${pack}**</outputDirectory>**

**<sourceDirectory>**src**</sourceDirectory>**

**<plugins>**

**<plugin>**

**</plugin>**

**</plugins>**

**</build>**

**<dependencies>**

**<dependency>**

**</dependency>**

**</dependencies>**

**<properties>**

**<!—user defined variable- ->**

**<pack>**nats**</pack>**

**</properties>**

**</project>**

**Note**: - groupId, ArtifactId, Version are just like package in java.They are given to uniquely identify each plugins,project,dependencies.For each dependency you can see a separate directory in the repository,holding that jar files

**Note:-**

* All these phases are executed serially.
* To perform each operation,we need to take the support of a plugins.
* If you want to perform any operation in a particular phase,while configuring plugins specify the phase and goal,so that when maven executes that phase,it will perform the operation as specified by plugins.
* If you execute the pom.xml using mvn package,then it will execute all the phases from 1 to 17,but doesnot deploy the war.

**Dependencies:**

Some times our java class depends on certain jar files,In these case we can use those jar files using 2 ways

1>Manually arrange jar file in your web-resources

2>Just give information to Maven about the jar file,Maven will download the jars from local repository(If its not available in local repository,maven will download the same from internet repository

To gather jar from maven repository(Local or Internet),we need to include the following code In pom.xml

<dependencies>

<dependency>

<groupId>junit</groupId> 🡪 organisation name

<artifactId>junit</artifactId> 🡪represents jar file name

<version>3.8.1</version> 🡪Reprents version of jar

<scope>test</scope> 🡪 scope of the dependency

</dependency>

</dependencies>

**Scope:**

It defines how project uses that dependency.It is of 6 type

**1>compile:-**This is the default scope, used if none is specified.It means the jar file is needed to compile the resources,and hence added to classpath dynamically

**2>Provided:-**This indicates that the jar file will be provided by underlaying server/container /JDK at runtime.

E.g:-If resource is servlet/jsp,Maven needs those jar only to compile those resources,while execution those jar will be provided by Servers.

**3>runtime:-** This indicates that the dependency is not required for compilation, but is for execution. It is in the runtime path but not the compile classpath.

**4>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.

**5>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.

**6> import**:- (only available in Maven 2.0.9 or later)

This scope is only used on a dependency of type pom in the <dependencyManagement>section. It indicates that the specified POM should be replaced with the dependencies in that 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.

**List of Important Plugins and goals**

|  |  |  |
| --- | --- | --- |
| Plugin Name | Goals Available | Purpose |
| maven-compiler-plugin | compile,testCompile | Compile java file and testCase file |
| exec-maven-plugin | exec,java | Compiles and executes java classes |
| maven-antrun-plugin | run | Execute ant tasks |
| tomcat-maven-plugin | Start,stop,deploy, redeploy,undeploy | Deploys the project into tomcat server |
| jboss-maven-plugin | --do-- | Deploys the project into jboss |

**How to Gather dependency details of any jar file**

Search for your jar here <http://mirrors.ibiblio.org/pub/mirrors/maven/>

Navigate inside , search and open pom.xml,which contains dependency details

**Info About all the plugins** <http://mvnrepository.com/plugins.html>