Maven:

Maven is a project management and comprehension tool. Maven provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage following:

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of build related tasks.

Maven History

Maven was originally designed to simplify building processes in Jakarta Turbine project. There were several projects and each project contained slightly different ANT build files. JARs were checked into CVS.

Apache group then developed *Maven* which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

Maven Objective

Maven primary goal is to provide developer

* A comprehensive model for projects which is reusable, maintainable, and easier to comprehend.
* plugins or tools that interact with this declarative model.

Maven project structure and contents are declared in an xml file, pom.xml referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. Refer to [Maven POM](https://www.tutorialspoint.com/maven/maven_pom.htm) section for more detail.

Convention over Configuration

Maven uses *Convention* over *Configuration* which means developers are not required to create build process themselves.

Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.

As an example, following table shows the default values for project source code files, resource files and other configurations. Assuming, **${basedir}**denotes the project location:

|  |  |
| --- | --- |
| **Item** | **Default** |
| source code | ${basedir}/src/main/java |
| resources | ${basedir}/src/main/resources |
| Tests | ${basedir}/src/test |
| distributable JAR | ${basedir}/target |
| Complied byte code | ${basedir}/target/classes |

In order to build the project, Maven provides developers options to mention life-cycle goals and project dependencies (that rely on Maven pluging capabilities and on its default conventions). Much of the project management and build related tasks are maintained by Maven plugins.

Developers can build any given Maven project without need to understand how the individual plugins work. Refer to [Maven Plug-ins](https://www.tutorialspoint.com/maven/maven_plugins.htm) section for more detail.

Maven is Java based tool, so the very first requirement is to have JDK installed on your machine.

System Requirement

|  |  |
| --- | --- |
| **JDK** | 1.5 or above. |
| **Memory** | no minimum requirement. |
| **Disk Space** | no minimum requirement. |
| **Operating System** | no minimum requirement. |

Step 1 - verify Java installation on your machine

Now open console and execute the following **java** command.

|  |  |  |
| --- | --- | --- |
| **OS** | **Task** | **Command** |
| Windows | Open Command Console | c:\> java -version |
| Linux | Open Command Terminal | $ java -version |
| Mac | Open Terminal | machine:~ joseph$ java -version |

Let's verify the output for all the operating systems:

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | java version "1.7.0\_75"  Java(TM) SE Runtime Environment (build 1.7.0\_75-b13)  Java HotSpot(TM) Client VM (build 24.75-b04, mixed mode, sharing) |
| Linux | java version "1.7.0\_75"  Java(TM) SE Runtime Environment (build 1.7.0\_75-b13)  Java HotSpot(TM) Client VM (build 24.75-b04, mixed mode, sharing) |
| Mac | java version "1.7.0\_75"  Java(TM) SE Runtime Environment (build 1.7.0\_75-b13)  Java HotSpot(TM)64-Bit Server VM (build 24.75-b04, mixed mode, sharing) |

If you do not have Java installed, install the Java Software Development Kit (SDK) from<http://www.oracle.com/technetwork/java/javase/downloads/index.html>. We are assuming Java 1.7.0\_75 as installed version for this tutorial.

Step 2: Set JAVA environment

Set the **JAVA\_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Set the environment variable JAVA\_HOME to C:\Program Files\Java\jdk1.7.0\_75 |
| Linux | export JAVA\_HOME=/usr/local/java-current |
| Mac | export JAVA\_HOME=/Library/Java/Home |

Append Java compiler location to System Path.

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Append the string ;C:\Program Files\Java\jdk1.7.0\_75\bin to the end of the system variable, Path. |
| Linux | export PATH=$PATH:$JAVA\_HOME/bin/ |
| Mac | not required |

Verify Java Installation using **java -version** command explained above.

Step 3: Download Maven archive

Download Maven 3.3.3 from <http://maven.apache.org/download.cgi>

|  |  |
| --- | --- |
| **OS** | **Archive name** |
| Windows | apache-maven-3.3.3-bin.zip |
| Linux | apache-maven-3.3.3-bin.tar.gz |
| Mac | apache-maven-3.3.3-bin.tar.gz |

Step 4: Extract the Maven archive

Extract the archive, to the directory you wish to install Maven 3.3.3. The subdirectory apache-maven-3.3.3 will be created from the archive.

|  |  |
| --- | --- |
| **OS** | **Location (can be different based on your installation)** |
| Windows | C:\Program Files\Apache Software Foundation\apache-maven-3.3.3 |
| Linux | /usr/local/apache-maven |
| Mac | /usr/local/apache-maven |

Step 5: Set Maven environment variables

Add M2\_HOME, M2, MAVEN\_OPTS to environment variables.

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Set the environment variables using system properties.   *M2\_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.3.3*  *M2=%M2\_HOME%\bin*  *MAVEN\_OPTS=-Xms256m -Xmx512m* |
| Linux | Open command terminal and set environment variables.  *export M2\_HOME=/usr/local/apache-maven/apache-maven-3.3.3*  *export M2=$M2\_HOME/bin*  *export MAVEN\_OPTS=-Xms256m -Xmx512m* |
| Mac | Open command terminal and set environment variables.  *export M2\_HOME=/usr/local/apache-maven/apache-maven-3.3.3*  *export M2=$M2\_HOME/bin*  *export MAVEN\_OPTS=-Xms256m -Xmx512m* |

Step 6: Add Maven bin directory location to system path

Now append M2 variable to System Path

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Append the string ;%M2% to the end of the system variable, Path. |
| Linux | export PATH=$M2:$PATH |
| Mac | export PATH=$M2:$PATH |

Step 7: Verify Maven installation

Now open console, execute the following **mvn** command.

|  |  |  |
| --- | --- | --- |
| **OS** | **Task** | **Command** |
| Windows | Open Command Console | c:\> mvn --version |
| Linux | Open Command Terminal | $ mvn --version |
| Mac | Open Terminal | machine:~ joseph$ mvn --version |

Finally, verify the output of the above commands, which should be something as follows:

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)  Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.3.3  Java version: 1.7.0\_75, vendor: Oracle Corporation  Java home: C:\Program Files\Java\jdk1.7.0\_75\jre  Default locale: en\_US, platform encoding: Cp1252 |
| Linux | Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)  Maven home: /usr/local/apache-maven/apache-maven-3.3.3  Java version: 1.7.0\_75, vendor: Oracle Corporation  Java home: /usr/local/java-current/jdk1.7.0\_75/jre |
| Mac | Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)  Maven home: /usr/local/apache-maven/apache-maven-3.3.3  Java version: 1.7.0\_75, vendor: Oracle Corporation  Java home: /Library/Java/Home/jdk1.7.0\_75/jre |

Congratulations! you are now all set to use Apache Maven for your projects.

POM stands for *Project Object Model*. It is fundamental Unit of Work in Maven. It is an XML file. It always resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal. Some of the configuration that can be specified in the POM are following:

* project dependencies
* plugins
* goals
* build profiles
* project version
* developers
* mailing list

Before creating a POM, we should first decide the project **group** (groupId), its **name**(artifactId) and its version as these attributes help in uniquely identifying the project in repository.

Example POM

<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.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

</project>

It should be noted that there should be a single POM file for each project.

* All POM files require the **project** element and three mandatory fields: **groupId, artifactId,version.**
* Projects notation in repository is **groupId:artifactId:version.**
* Root element of POM.xml is **project** and it has three major sub-nodes :

|  |  |
| --- | --- |
| **Node** | **Description** |
| groupId | This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects. |
| artifactId | This is an Id of the project.This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository. |
| version | This is the version of the project.Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example:   *com.company.bank:consumer-banking:1.0*  *com.company.bank:consumer-banking:1.1.* |

Super POM

All POMs inherit from a parent (despite explicitly defined or not). This base POM is known as the **Super POM**, and contains values inherited by default.

Maven use the effective pom (configuration from super pom plus project configuration) to execute relevant goal. It helps developer to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**

Create a pom.xml in any directory on your computer.Use the content of above mentioned example pom.

In example below, We've created a pom.xml in C:\MVN\project folder.

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

Groupid: This is an Id of project's group. This is generally unique amongst an organization or a project.

artifactId: This is an Id of the project.This is generally name of the project.

Version :

prepare-resources : resource copying

Resource copying can be customized in this phase.

Compile:

|  |  |  |
| --- | --- | --- |
| **Phase** | **Handles** | **Description** |
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| compile | compilation | Source code compilation is done in this phase. |
| package | packaging | This phase creates the JAR / WAR package as mentioned in packaging in POM.xml. |
| install | installation | This phase installs the package in local / remote maven repository. |

There are always **pre** and **post** phases which can be used to register **goals**which must run prior to or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals which are registered with each phase. Maven has following three standard lifecycles:

* clean
* default(or build)
* site

A **goal** represents a specific task which contributes to the building and managing of a project.

 It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

he order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The clean and package arguments are build phases while the *dependency:copy-dependencies* is a goal.

mvn clean dependency:copy-dependencies package

Here the *clean* phase will be executed first, and then the *dependency:copy-dependencies goal* will be executed, and finally *package* phase will be executed.

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

* pre-clean
* clean
* post-clean

Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its *clean:clean* goal deletes the output of a build by deleting the build directory. Thus when *mvn clean* command executes, Maven deletes the build directory.

We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

In the following example, We'll attach maven-antrun-plugin:run goal to the pre-clean, clean, and post-clean phases. This will allow us to echo text messages displaying the phases of the clean lifecycle.

We've created a pom.xml in C:\MVN\project folder.

We've created a pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.pre-clean</id>

<phase>pre-clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>pre-clean phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.clean</id>

<phase>clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>clean phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.post-clean</id>

<phase>post-clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>post-clean phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go to the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn post-clean

Maven will start processing and display all the phases of clean life cycle

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.pre-clean) @ project ---

[INFO] Executing tasks

[echo] pre-clean phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ project ---

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.clean) @ project ---

[INFO] Executing tasks

[echo] clean phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.post-clean) @ project ---

[INFO] Executing tasks

[echo] post-clean phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 2.078 s

[INFO] Finished at: 2015-09-26T08:03:06+05:30

[INFO] Final Memory: 7M/247M

[INFO] ------------------------------------------------------------------------

You can try tuning **mvn clean** command which will display *pre-clean* and *clean*, nothing will be executed for *post-clean* phase.

Default (or Build) Lifecycle

This is the primary life cycle of Maven and is used to build the application. It has following 23 phases.

|  |  |
| --- | --- |
| **Lifecycle Phase** | **Description** |
| validate | Validates whether project is correct and all necessary information is available to complete the build process. |
| initialize | Initializes build state, for example set properties |
| generate-sources | Generate any source code to be included in compilation phase. |
| process-sources | Process the source code, for example, filter any value. |
| generate-resources | Generate resources to be included in the package. |
| process-resources | Copy and process the resources into the destination directory, ready for packaging phase. |
| compile | Compile the source code of the project. |
| process-classes | Post-process the generated files from compilation, for example to do bytecode enhancement/optimization on Java classes. |
| generate-test-sources | Generate any test source code to be included in compilation phase. |
| process-test-sources | Process the test source code, for example, filter any values. |
| test-compile | Compile the test source code into the test destination directory. |
| process-test-classes | Process the generated files from test code file compilation. |
| test | Run tests using a suitable unit testing framework(Junit is one). |
| prepare-package | Perform any operations necessary to prepare a package before the actual packaging. |
| package | Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR file. |
| pre-integration-test | Perform actions required before integration tests are executed. For example, setting up the required environment. |
| integration-test | Process and deploy the package if necessary into an environment where integration tests can be run. |
| post-integration-test | Perform actions required after integration tests have been executed. For example, cleaning up the environment. |
| verify | Run any check-ups to verify the package is valid and meets quality criterias. |
| install | Install the package into the local repository, which can be used as a dependency in other projects locally. |
| deploy | Copies the final package to the remote repository for sharing with other developers and projects. |

There are few important concepts related to Maven Lifecycles which are worth to mention:

* When a phase is called via Maven command, for example *mvn compile*, only phases upto and including that phase will execute.
* Different maven goals will be bound to different phases of Maven lifecycle depending upon the type of packaging (JAR / WAR / EAR).

In the following example, We'll attach maven-antrun-plugin:run goal to few of the phases of Build lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.validate</id>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>validate phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.compile</id>

<phase>compile</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>compile phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.test</id>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>test phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.package</id>

<phase>package</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>package phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.deploy</id>

<phase>deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn compile

Maven will start processing and display phases of build life cycle upto compile phase.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.validate) @ project ---

[INFO] Executing tasks

[echo] validate phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ project --

-

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,

i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory C:\MVN\project\src\main\resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ project ---

[INFO] No sources to compile

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.compile) @ project ---

[INFO] Executing tasks

[echo] compile phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 3.704 s

[INFO] Finished at: 2015-09-26T08:22:05+05:30

[INFO] Final Memory: 10M/247M

[INFO] ------------------------------------------------------------------------

Site Lifecycle

Maven Site plugin is generally used to create fresh documentation to create reports, deploy site etc.

Phases

* pre-site
* site
* post-site
* site-deploy

In the following example, We'll attach *maven-antrun-plugin:run* goal to all the phases of Site lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.pre-site</id>

<phase>pre-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>pre-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site</id>

<phase>site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.post-site</id>

<phase>post-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>post-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site-deploy</id>

<phase>site-deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site-deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn site

Maven will start processing and display phases of site life cycle upto site phase.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.pre-site) @ project ---

[INFO] Executing tasks

[echo] pre-site phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-site-plugin:3.3:site (default-site) @ project ---

[WARNING] Report plugin org.apache.maven.plugins:maven-project-info-reports-plug

in has an empty version.

[WARNING]

[WARNING] It is highly recommended to fix these problems because they threaten t

he stability of your build.

[WARNING]

[WARNING] For this reason, future Maven versions might no longer support buildin

g such malformed projects.

[INFO] configuring report plugin org.apache.maven.plugins:maven-project-info-rep

orts-plugin:2.8.1

[WARNING] No project URL defined - decoration links will not be relativized!

[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 ski

n.

[INFO] Generating "Dependency Convergence" report --- maven-project-info-repo

rts-plugin:2.8.1

[INFO] Generating "Dependency Information" report --- maven-project-info-repo

rts-plugin:2.8.1

[INFO] Generating "About" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Plugin Management" report --- maven-project-info-reports-p

lugin:2.8.1

[INFO] Generating "Project Plugins" report --- maven-project-info-reports-plu

gin:2.8.1

[INFO] Generating "Project Summary" report --- maven-project-info-reports-plu

gin:2.8.1

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.site) @ project ---

[INFO] Executing tasks

[echo] site phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 11.390 s

[INFO] Finished at: 2015-09-26T08:43:45+05:30

[INFO] Final Memory: 18M/247M

[INFO] ------------------------------------------------------------------------

his is the primary life cycle of Maven and is used to build the application. It has following 23 phases.

|  |  |
| --- | --- |
| **Lifecycle Phase** | **Description** |
| validate | Validates whether project is correct and all necessary information is available to complete the build process. |
| initialize | Initializes build state, for example set properties |
| generate-sources | Generate any source code to be included in compilation phase. |
| process-sources | Process the source code, for example, filter any value. |
| generate-resources | Generate resources to be included in the package. |
| process-resources | Copy and process the resources into the destination directory, ready for packaging phase. |
| compile | Compile the source code of the project. |
| process-classes | Post-process the generated files from compilation, for example to do bytecode enhancement/optimization on Java classes. |
| generate-test-sources | Generate any test source code to be included in compilation phase. |
| process-test-sources | Process the test source code, for example, filter any values. |
| test-compile | Compile the test source code into the test destination directory. |
| process-test-classes | Process the generated files from test code file compilation. |
| test | Run tests using a suitable unit testing framework(Junit is one). |
| prepare-package | Perform any operations necessary to prepare a package before the actual packaging. |
| package | Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR file. |
| pre-integration-test | Perform actions required before integration tests are executed. For example, setting up the required environment. |
| integration-test | Process and deploy the package if necessary into an environment where integration tests can be run. |
| post-integration-test | Perform actions required after integration tests have been executed. For example, cleaning up the environment. |
| verify | Run any check-ups to verify the package is valid and meets quality criterias. |
| install | Install the package into the local repository, which can be used as a dependency in other projects locally. |
| deploy | Copies the final package to the remote repository for sharing with other developers and projects. |

There are few important concepts related to Maven Lifecycles which are worth to mention:

* When a phase is called via Maven command, for example *mvn compile*, only phases upto and including that phase will execute.
* Different maven goals will be bound to different phases of Maven lifecycle depending upon the type of packaging (JAR / WAR / EAR).

In the following example, We'll attach maven-antrun-plugin:run goal to few of the phases of Build lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.validate</id>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>validate phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.compile</id>

<phase>compile</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>compile phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.test</id>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>test phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.package</id>

<phase>package</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>package phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.deploy</id>

<phase>deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn compile

Maven will start processing and display phases of build life cycle upto compile phase.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.validate) @ project ---

[INFO] Executing tasks

[echo] validate phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ project --

-

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,

i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory C:\MVN\project\src\main\resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ project ---

[INFO] No sources to compile

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.compile) @ project ---

[INFO] Executing tasks

[echo] compile phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 3.704 s

[INFO] Finished at: 2015-09-26T08:22:05+05:30

[INFO] Final Memory: 10M/247M

[INFO] ------------------------------------------------------------------------

Site Lifecycle

Maven Site plugin is generally used to create fresh documentation to create reports, deploy site etc.

Phases

* pre-site
* site
* post-site
* site-deploy

In the following example, We'll attach *maven-antrun-plugin:run* goal to all the phases of Site lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.pre-site</id>

<phase>pre-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>pre-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site</id>

<phase>site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.post-site</id>

<phase>post-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>post-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site-deploy</id>

<phase>site-deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site-deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn site

Maven will start processing and display phases of site life cycle upto site phase.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.pre-site) @ project ---

[INFO] Executing tasks

[echo] pre-site phase

[INFO] Executed tasks

[INFO]

[INFO] --- maven-site-plugin:3.3:site (default-site) @ project ---

[WARNING] Report plugin org.apache.maven.plugins:maven-project-info-reports-plug

in has an empty version.

[WARNING]

[WARNING] It is highly recommended to fix these problems because they threaten t

he stability of your build.

[WARNING]

[WARNING] For this reason, future Maven versions might no longer support buildin

g such malformed projects.

[INFO] configuring report plugin org.apache.maven.plugins:maven-project-info-rep

orts-plugin:2.8.1

[WARNING] No project URL defined - decoration links will not be relativized!

[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 ski

n.

[INFO] Generating "Dependency Convergence" report --- maven-project-info-repo

rts-plugin:2.8.1

[INFO] Generating "Dependency Information" report --- maven-project-info-repo

rts-plugin:2.8.1

[INFO] Generating "About" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Plugin Management" report --- maven-project-info-reports-p

lugin:2.8.1

[INFO] Generating "Project Plugins" report --- maven-project-info-reports-plu

gin:2.8.1

[INFO] Generating "Project Summary" report --- maven-project-info-reports-plu

gin:2.8.1

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (id.site) @ project ---

[INFO] Executing tasks

[echo] site phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 11.390 s

[INFO] Finished at: 2015-09-26T08:43:45+05:30

[INFO] Final Memory: 18M/247M

[INFO] ------------------------------------------------------------------------

C:\MVN\project>mvn compile

C:\MVN\project>mvn compile

Maven will start processing and display phases of build life cycle upto compile phase.

hat is Build Profile?

A *Build profile* is a set of configuration values which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as *Production* v/s *Development* environments.

Profiles are specified in pom.xml file using its activeProfiles / profiles elements and are triggered in variety of ways. Profiles modify the POM at build time, and are used to give parameters different target environments (for example, the path of the database server in the development, testing, and production environments).

Types of Build Profile

Build profiles are majorly of three types

|  |  |
| --- | --- |
| **Type** | **Where it is defined** |
| Per Project | Defined in the project POM file, pom.xml |
| Per User | Defined in Maven settings xml file (%USER\_HOME%/.m2/settings.xml) |
| Global | Defined in Maven global settings xml file (%M2\_HOME%/conf/settings.xml) |

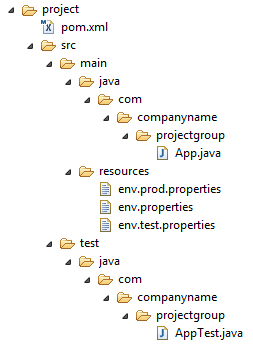
Profile Activation

A Maven Build Profile can be activated in various ways.

* Explicitly using command console input.
* Through maven settings.
* Based on environment variables (User/System variables).
* OS Settings (for example, Windows family).
* Present/missing files.

Profile Activation Examples

Let us assume following directory structure of your project:



Now, under *src/main/resources* there are three environment specific files:

|  |  |
| --- | --- |
| **File Name** | **Description** |
| env.properties | default configuration used if no profile is mentioned. |
| env.test.properties | test configuration when test profile is used. |
| env.prod.properties | production configuration when prod profile is used. |

Explicit Profile Activation

In the following example, We'll attach maven-antrun-plugin:run goal to test phase. This will allow us to echo text messages for different profiles. We will be using pom.xml to define different profiles and will activate profile at command console using maven command.

Assume, we've created following pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<profiles>

<profile>

<id>test</id>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>Using env.test.properties</echo>

<copy file="src/main/resources/env.test.properties" tofile

="${project.build.outputDirectory}/env.properties"/>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</profile>

</profiles>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

And assume, we've created following properties file in C:\MVN\project\src\resources folder.

*env.properties*

environment=debug

*env.test.properties*

environment=test

*env.prod.properties*

environment=prod

Now open command console, go to the folder containing pom.xml and execute the following **mvn** command. Pass the profile name as argument using -P option.

C:\MVN\project>mvn test -Ptest

Maven will start processing and display the result of test build profile.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building project 1.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ project --

-

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,

i.e. build is platform dependent!

[INFO] Copying 3 resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ project ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ pr

oject ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,

i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory C:\MVN\project\src\test\resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ project

---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ project ---

[INFO] Surefire report directory: C:\MVN\project\target\surefire-reports

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running com.companyname.bank.AppTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.016 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] --- maven-antrun-plugin:1.1:run (default) @ project ---

[INFO] Executing tasks

[echo] Using env.test.properties

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 4.953 s

[INFO] Finished at: 2015-09-27T11:54:45+05:30

[INFO] Final Memory: 9M/247M

[INFO] ------------------------------------------------------------------------

Now as an exercise, you can do the following steps

* Add another profile element to profiles element of pom.xml (copy existing profile element and paste it where profile elements ends).
* Update id of this profile element from test to normal.
* Update task section to echo env.properties and copy env.properties to target directory
* Again repeat above three steps, update id to prod and task section for env.prod.properties
* That's all. Now you've three build profiles ready (normal / test / prod).

Now open command console, go to the folder containing pom.xml and execute the following **mvn** commands. Pass the profile names as argument using -P option.

C:\MVN\project>mvn test -Pnormal

C:\MVN\project>mvn test -Pprod

Check the output of build to see the difference.

Profile Activation via Maven Settings

Open Maven **settings.xml** file available in %USER\_HOME%/.m2 directory where **%USER\_HOME%** represents user home directory. If settings.xml file is not there then create a new one.

Add test profile as an active profile using activeProfiles node as shown below in example

<settings 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/settings-1.0.0.xsd">

<mirrors>

<mirror>

<id>maven.dev.snaponglobal.com</id>

<name>Internal Artifactory Maven repository</name>

<url>http://repo1.maven.org/maven2/</url>

<mirrorOf>\*</mirrorOf>

</mirror>

</mirrors>

<activeProfiles>

<activeProfile>test</activeProfile>

</activeProfiles>

</settings>

Now open command console, go to the folder containing pom.xml and execute the following **mvn** command. Do not pass the profile name using -P option.Maven will display result of test profile being an active profile.

C:\MVN\project>mvn test

Profile Activation via Environment Variables

Now remove active profile from maven settings.xml and update the test profile mentioned in pom.xml. Add activation element to profile element as shown below.

The test profile will trigger when the system property "env" is specified with the value "test". Create a environment variable "env" and set its value as "test".

<profile>

<id>test</id>

<activation>

<property>

<name>env</name>

<value>test</value>

</property>

</activation>

</profile>

Let's open command console, go to the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn test

Profile Activation via Operating System

Activation element to include os detail as shown below. This test profile will trigger when the system is windows XP.

<profile>

<id>test</id>

<activation>

<os>

<name>Windows XP</name>

<family>Windows</family>

<arch>x86</arch>

<version>5.1.2600</version>

</os>

</activation>

</profile>

Now open command console, go to the folder containing pom.xml and execute the following **mvn** commands. Do not pass the profile name using -P option.Maven will display result of test profile being an active profile.

C:\MVN\project>mvn test

Profile Activation via Present/Missing File

Now activation element to include os detail as shown below. The test profile will trigger when *target/generated-sources/axistools/wsdl2java/com/companyname/group* is missing.

<profile>

<id>test</id>

<activation>

<file>

<missing>target/generated-sources/axistools/wsdl2java/

com/companyname/group</missing>

</file>

</activation>

</profile>

Now open command console, go to the folder containing pom.xml and execute the following **mvn** commands. Do not pass the profile name using -P option.Maven will display result of test profile being an active profile.

C:\MVN\project>mvn test

mvn test -Ptest

## What is a Maven Repository?

In Maven terminology, a repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types

* local
* central
* remote

## Local Repository

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository.It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in %USER\_HOME% directory. To override the default location, mention another path in Maven settings.xml file available at %M2\_HOME%\conf directory.

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0

http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>C:/MyLocalRepository</localRepository>

</settings>

When you run Maven command, Maven will download dependencies to your custom path.

## Central Repository

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL: <http://repo1.maven.org/maven2/>

Key concepts of Central repository

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

To browse the content of central maven repository, maven community has provided a URL: <http://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

## Remote Repository

Sometime, Maven does not find a mentioned dependency in central repository as well then it stopped build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository** which is developer's own custom repository containing required libraries or other project jars.

For example, using below mentioned POM.xml,Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>com.companyname.common-lib</groupId>

<artifactId>common-lib</artifactId>

<version>1.0.0</version>

</dependency>

<dependencies>

<repositories>

<repository>

<id>companyname.lib1</id>

<url>http://download.companyname.org/maven2/lib1</url>

</repository>

<repository>

<id>companyname.lib2</id>

<url>http://download.companyname.org/maven2/lib2</url>

</repository>

</repositories>

</project>

## Maven Dependency Search Sequence

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence:

* **Step 1 -**Search dependency in local repository, if not found, move to step 2 else if found then do the further processing.
* **Step 2 -**Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4 else if found, then it is downloaded to local repository for future reference.
* **Step 3 -**If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
* **Step 4 -**Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference otherwise Maven as expected stop processing and throws error (Unable to find dependency).

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to :

* create jar file
* create war file
* compile code files
* unit testing of code
* create project documentation
* create project reports

A plugin generally provides a set of goals and which can be executed using following syntax:

mvn [plugin-name]:[goal-name]

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running following command

mvn compiler:compile

Plugin Types

Maven provided following two types of Plugins:

|  |  |
| --- | --- |
| **Type** | **Description** |
| Build plugins | They execute during the build and should be configured in the <build/> element of pom.xml |
| Reporting plugins | They execute during the site generation and they should be configured in the <reporting/> element of the pom.xml |

Following is the list of few common plugins:

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| clean | Clean up target after the build. Deletes the target directory. |
| compiler | Compiles Java source files. |
| surefire | Run the JUnit unit tests. Creates test reports. |
| jar | Builds a JAR file from the current project. |
| war | Builds a WAR file from the current project. |
| javadoc | Generates Javadoc for the project. |
| antrun | Runs a set of ant tasks from any phase mentioned of the build. |

Example

We've used **maven-antrun-plugin** extensively in our examples to print data on console. See [Maven Build Profiles](https://www.tutorialspoint.com/maven/maven_build_profiles.htm) chapter. Let to understand it in a better way let's create a pom.xml in C:\MVN\project folder.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.clean</id>

<phase>clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>clean phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Next, open command console and go to the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn clean

Maven will start processing and display clean phase of clean life cycle

[INFO] Scanning for projects...

[INFO] ------------------------------------------------------------------

[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0

[INFO] task-segment: [post-clean]

[INFO] ------------------------------------------------------------------

[INFO] [clean:clean {execution: default-clean}]

[INFO] [antrun:run {execution: id.clean}]

[INFO] Executing tasks

[echo] clean phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------

[INFO] Total time: < 1 second

[INFO] Finished at: Sat Jul 07 13:38:59 IST 2012

[INFO] Final Memory: 4M/44M

[INFO] ------------------------------------------------------------------

The above example illustrates the following key concepts:

* Plugins are specified in pom.xml using plugins element.
* Each plugin can have multiple goals.
* You can define phase from where plugin should starts its processing using its phase element. We've used **clean** phase.
* You can configure tasks to be executed by binding them to goals of plugin. We've bound **echo** task with **run** goal of *maven-antrun-plugin*.
* That's it, Maven will handle the rest. It will download the plugin if not available in local repository and starts its processing.
* Maven uses **archetype** plugins to create projects. To create a simple java application, we'll use maven-archetype-quickstart plugin. In example below, We'll create a maven based java application project in C:\MVN folder.
* Let's open command console, go the C:\MVN directory and execute the following **mvn** command.
* C:\MVN>mvn archetype:generate
* -DgroupId=com.companyname.bank
* -DartifactId=consumerBanking
* -DarchetypeArtifactId=maven-archetype-quickstart
* -DinteractiveMode=false
* Maven will start processing and will create the complete java application project structure.
* [INFO] Scanning for projects...
* [INFO]
* [INFO] ------------------------------------------------------------------------
* [INFO] Building Maven Stub Project (No POM) 1
* [INFO] ------------------------------------------------------------------------
* [INFO]
* [INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources
* @ standalone-pom >>>
* [INFO]
* [INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources
* @ standalone-pom <<<
* [INFO]
* [INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom --
* -
* [INFO] Generating project in Batch mode
* [INFO] -------------------------------------------------------------------------
* ---
* [INFO] Using following parameters for creating project from Old (1.x) Archetype:
* maven-archetype-quickstart:1.0
* [INFO] -------------------------------------------------------------------------
* ---
* [INFO] Parameter: groupId, Value: com.companyname.bank
* [INFO] Parameter: packageName, Value: com.companyname.bank
* [INFO] Parameter: package, Value: com.companyname.bank
* [INFO] Parameter: artifactId, Value: consumerBanking
* [INFO] Parameter: basedir, Value: C:\MVN
* [INFO] Parameter: version, Value: 1.0-SNAPSHOT
* [INFO] project created from Old (1.x) Archetype in dir: C:\MVN\consumerBanking
* [INFO] ------------------------------------------------------------------------
* [INFO] BUILD SUCCESS
* [INFO] ------------------------------------------------------------------------
* [INFO] Total time: 03:19 min
* [INFO] Finished at: 2015-09-26T12:18:26+05:30
* [INFO] Final Memory: 15M/247M
* [INFO] ------------------------------------------------------------------------
* Now go to C:/MVN directory. You'll see a java application project created named consumerBanking (as specified in artifactId). Maven uses a standard directory layout as shown below:
* 
* Using above example, we can understand following key concepts

|  |  |
| --- | --- |
| **Folder Structure** | **Description** |
| consumerBanking | contains src folder and pom.xml |
| src/main/java | contains java code files under the package structure (com/companyName/bank). |
| src/main/test | contains test java code files under the package structure (com/companyName/bank). |
| src/main/resources | it contains images/properties files (In above example, we need to create this structure manually). |

* If you see, Maven also created a sample Java Source file and Java Test file. Open C:\MVN\consumerBanking\src\main\java\com\companyname\bank folder, you will see App.java.
* package com.companyname.bank;
* /\*\*
* \* Hello world!
* \*
* \*/
* public class App
* {
* public static void main( String[] args )
* {
* System.out.println( "Hello World!" );
* }
* }
* Open C:\MVN\consumerBanking\src\test\java\com\companyname\bank folder, you will see AppTest.java.
* package com.companyname.bank;
* import junit.framework.Test;
* import junit.framework.TestCase;
* import junit.framework.TestSuite;
* /\*\*
* \* Unit test for simple App.
* \*/
* public class AppTest extends TestCase
* {
* /\*\*
* \* Create the test case
* \*
* \* @param testName name of the test case
* \*/
* public AppTest( String testName )
* {
* super( testName );
* }
* /\*\*
* \* @return the suite of tests being tested
* \*/
* public static Test suite()
* {
* return new TestSuite( AppTest.class );
* }
* /\*\*
* \* Rigourous Test :-)
* \*/
* public void testApp()
* {
* assertTrue( true );
* }
* }
* Developers are required to place their files as mentioned in table above and Maven handles the all the build related complexities.
* In next section, we'll discuss how to build and test the project using maven [Maven Build & Test Project.](https://www.tutorialspoint.com/maven/maven_build_test_project.htm)

his tutorial will teach you how to create documentation of the application in one go. So let's start, go to C:/MVN directory where you had created your java **consumerBanking** application. Open *consumerBanking* folder and execute the following **mvn** command.

C:\MVN>mvn site

Maven will start building the project.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building consumerBanking 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-site-plugin:3.3:site (default-site) @ consumerBanking ---

[WARNING] Report plugin org.apache.maven.plugins:maven-project-info-reports-plugin has an empty version.

[WARNING]

[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.

[WARNING]

[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.

[INFO] configuring report plugin org.apache.maven.plugins:maven-project-info-reports-plugin:2.8.1

[INFO] Relativizing decoration links with respect to project URL: http://maven.apache.org

[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 skin.

[INFO] Generating "Dependencies" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Dependency Convergence" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Dependency Information" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "About" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Plugin Management" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Project Plugins" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Project Summary" report --- maven-project-info-reports-plugin:2.8.1

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

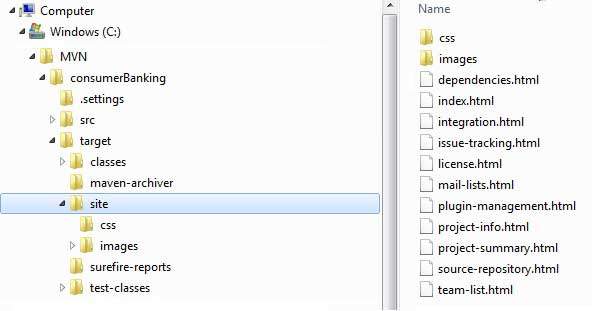
[INFO] Total time: 37.828 s

[INFO] Finished at: 2015-09-27T12:11:27+05:30

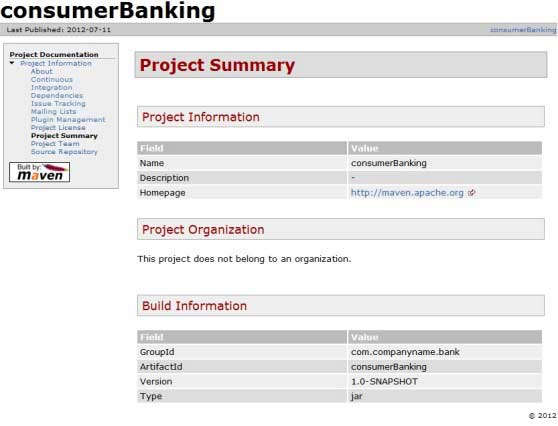
[INFO] Final Memory: 23M/247M

[INFO] ------------------------------------------------------------------------

That's it. Your project documentation is ready. Maven has created a site within target directory.



Open C:\MVN\consumerBanking\target\site folder. Click on index.html to see the documentation.



Maven creates the documentation using a documentation-processing engine called [Doxia](http://maven.apache.org/doxia/index.html" \t "_blank) which reads multiple source formats into a common document model.To write documentation for your project, you can write your content in a following few commonly used formats which are parsed by Doxia.

|  |  |  |
| --- | --- | --- |
| **Format Name** | **Description** | **Reference** |
| APT | A Plain Text document format | [**doxia format**](http://maven.apache.org/doxia/format.html) |
| XDoc | A Maven 1.x documentation format | [**jakarta format**](http://jakarta.apache.org/site/jakarta-site2.html) |
| FML | Used for FAQ documents | [**fml format**](http://maven.apache.org/doxia/references/fml-format.html) |
| XHTML | Extensible HTML | [**XHTML wiki**](http://en.wikipedia.org/wiki/XHTML) |