**Maven:**

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

[**next →**](https://www.javatpoint.com/difference-between-ant-and-maven)

What it does?

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 for new features of Maven

Apache Maven helps to manage

* Builds
* Documentation
* Reporing
* SCMs
* Releases
* Distribution

# Ant vs Maven

### shape Description

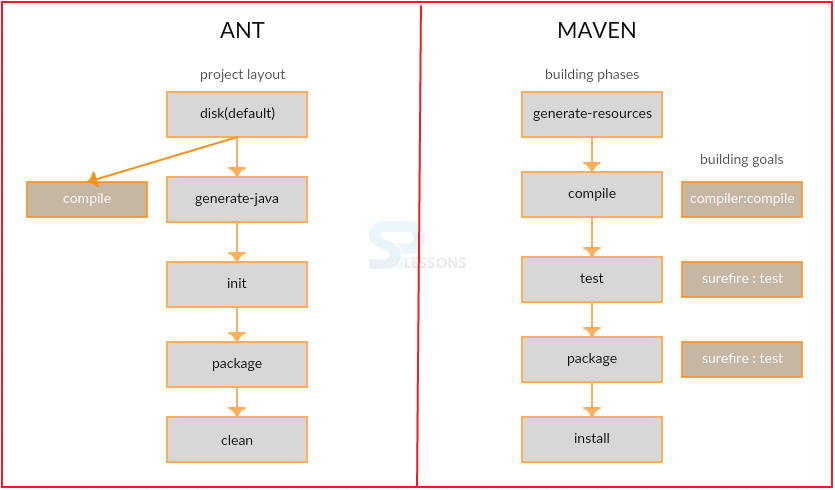
Maven is similar to the ANT. In fact, both are the build tools provided by Apache. The main purpose of these tools is to simplify the project development process. However, Maven is not a replacement for ANT.

Maven defines pattern applications to achieve characteristics like maintainability, re-usability and visibility, whereas, ANT is just a toolbox.

When examining the higher structure of ANT script, it may not be possible to infer. But with Maven, understanding projects that are independent of each other is easy.

### shape Conceptual figure

|  |  |  |
| --- | --- | --- |
| **Specification** | **ANT** | **Maven** |
| Formal Convention(project directory structure) | Instructions have to be given to the Ant about source location and output path. | Maven knows source code directory and byte code in classes and produces a JAR file in target. |
| Procedural | Ant is imperative and follows a procedure to perform tasks. | Maven is declarative. If pom.xml file is created and source file is saved in a default directory, Maven will take care of the remaining tasks. |
| Life Cycle | Goals and goal dependencies have to be defined. | Install command will execute the default plug-in goals like compile and creating jar. |
| Speed | Faster than Maven. | Slow. |
| Incremental Compilation Support | Compiles the sources that are modified. | Compiles all the sources. |
| Tool Type | Ant is a Build Tool. | Maven is a Project Management Tool. |
| Re-usability | Scripts are not re-usable | Plug-in builds are highly reusable. |
| Transitive Dependencies Support | Not possible. | Maintains dependencies of dependency. |
| Ideal use | Controlling build process is the ideal use. | Managing dependencies and building articrafts is the ideal use. |
| Complexity | Ant is Complex | Maven is less complex than Ant. |
|  |  |  |

The following is the project flow for ANT and Maven. [](https://cdn.splessons.com/spf/c81e728d9d4c2f636f067f89cc14862c/wp-content/uploads/2016/02/ant-vs-maven-splessons-2.png)

### shape Differences

**Ant vs Maven**

### shape Key Points

* Maven is a project development tool.
* Maven can handle multiple projects whereas Ant can handle only a single project.

# How to install Maven on windows

You can download and install maven on windows, linux and MAC OS platforms. Here, we are going to learn how to install maven on windows OS.

To install maven on windows, you need to perform following steps:

1. Download maven and extract it
2. Add JAVA\_HOME and MAVEN\_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

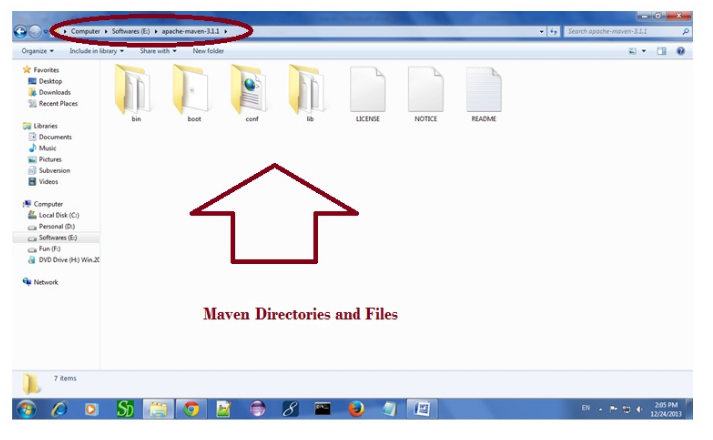
## 1) Download Maven

To install maven on windows, you need to download apache maven first.

Download Maven latest Maven software from [Download latest version of Maven](http://maven.apache.org/download.cgi)

For example: **apache-maven-3.1.1-bin.zip**

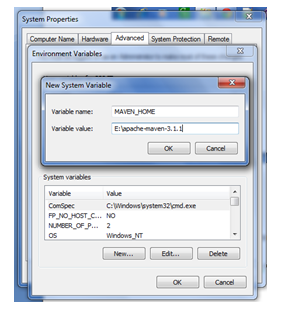
Extract it. Now it will look like this:



**2) Add MAVEN\_HOME in environment variable**

Right click on **MyComputer** -> **properties** -> **Advanced System Settings** -> **Environment variables** -> **click new button**

Now **add MAVEN\_HOME** in variable name and path of maven in variable value. It must be the home directory of maven i.e. outer directory of bin. For example: **E:\apache-maven-3.1.1** .It is displayed below:



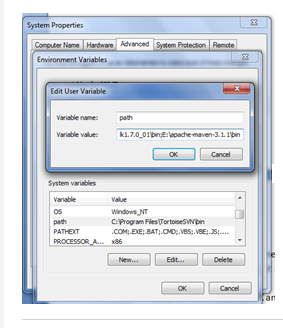
Now click on **OK** button.

## 3) Add Maven Path in environment variable

Click on new tab if path is not set, then set the path of maven. If it is set, edit the path and append the path of maven.

Here, we have installed JDK and its path is set by default, so we are going to append the path of maven.

The path of maven should be **%maven home%/bin**. For example, **E:\apache-maven-3.1.1\bin** .



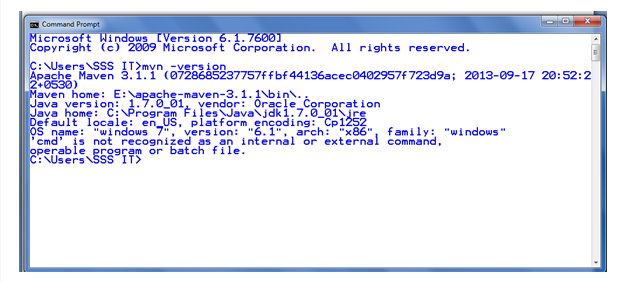
## 4)Verify maven

To verify whether maven is installed or not, open the command prompt and write:

1. mvn −version

Now it will display the version of maven and jdk including the maven home and java home.

Let's see the output:

**Maven - POM:** t is an XML file that 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, and 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.

## POM 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.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**.
* Minimal requirements for a POM −

|  |
| --- |
|  |
| **Sr.No.** | **Node & Description** |
| 1 | **Project root**  This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification. |
| 2 | **Model version**  Model version should be 4.0.0. |
| 3 | **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. |
| 4 | **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. |
| 5 | **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

The Super POM is Maven’s default POM. All POMs inherit from a parent or default (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 developers 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.

C:\MVN\project>mvn help:effective-pom

Maven will start processing and display the effective-pom.

[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'help'.

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

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

[INFO] task-segment: [help:effective-pom] (aggregator-style)

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

[INFO] [help:effective-pom {execution: default-cli}]

[INFO]

.....

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

[INFO] BUILD SUCCESSFUL

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

[INFO] Total time: < 1 second

[INFO] Finished at: Thu Jul 05 11:41:51 IST 2012

[INFO] Final Memory: 6M/15M

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

Effective POM displayed as result in console, after inheritance, interpolation, and profiles are applied.

<?xml version="1.0" encoding="UTF-8"?>

<!-- ============================================== -->

<!-- -->

<!-- Generated by Maven Help Plugin on 2015-04-09T11:41:51 -->

<!-- See: http://maven.apache.org/plugins/maven-help-plugin/ -->

<!-- -->

<!-- ==============================================-->

<!-- ==============================================-->

<!-- -->

<!-- Effective POM for project -->

<!-- 'com.companyname.project-group:project-name:jar:1.0' -->

<!-- -->

<!-- ============================================== -->

<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>

<build>

<sourceDirectory>C:\MVN\project\src\main\java</sourceDirectory>

<scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>

<testSourceDirectory>C:\MVN\project\src\test\java</testSourceDirectory>

<outputDirectory>C:\MVN\project\target\classes</outputDirectory>

<testOutputDirectory>C:\MVN\project\target\test-classes</testOutputDirectory>

<resources>

<resource>

<mergeId>resource-0</mergeId>

<directory>C:\MVN\project\src\main\resources</directory>

</resource>

</resources>

<testResources>

<testResource>

<mergeId>resource-1</mergeId>

<directory>C:\MVN\project\src\test\resources</directory>

</testResource>

</testResources>

<directory>C:\MVN\project\target</directory>

<finalName>project-1.0</finalName>

<pluginManagement>

<plugins>

<plugin>

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

<version>1.3</version>

</plugin>

<plugin>

<artifactId>maven-assembly-plugin< /artifactId>

<version>2.2-beta-2</version>

</plugin>

<plugin>

<artifactId>maven-clean-plugin< /artifactId>

<version>2.2</version>

</plugin>

<plugin>

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

<version>2.0.2</version>

</plugin>

<plugin>

<artifactId>maven-dependency-plugin</artifactId>

<version>2.0</version>

</plugin>

<plugin>

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

<version>2.4</version>

</plugin>

<plugin>

<artifactId>maven-ear-plugin</artifactId>

<version>2.3.1</version>

</plugin>

<plugin>

<artifactId>maven-ejb-plugin</artifactId>

<version>2.1</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-javadoc-plugin</artifactId>

<version>2.5</version>

</plugin>

<plugin>

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

<version>2.4.3</version>

</plugin>

<plugin>

<artifactId>maven-rar-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-release-plugin</artifactId>

<version>2.0-beta-8</version>

</plugin>

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>2.3</version>

</plugin>

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>2.0-beta-7</version>

</plugin>

<plugin>

<artifactId>maven-source-plugin</artifactId>

<version>2.0.4</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.4.3</version>

</plugin>

<plugin>

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

<version>2.1-alpha-2</version>

</plugin>

</plugins>

</pluginManagement>

<plugins>

<plugin>

<artifactId>maven-help-plugin</artifactId>

<version>2.1.1</version>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Maven Repository Switchboard</name>

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

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<releases>

<updatePolicy>never</updatePolicy>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Maven Plugin Repository</name>

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

</pluginRepository>

</pluginRepositories>

<reporting>

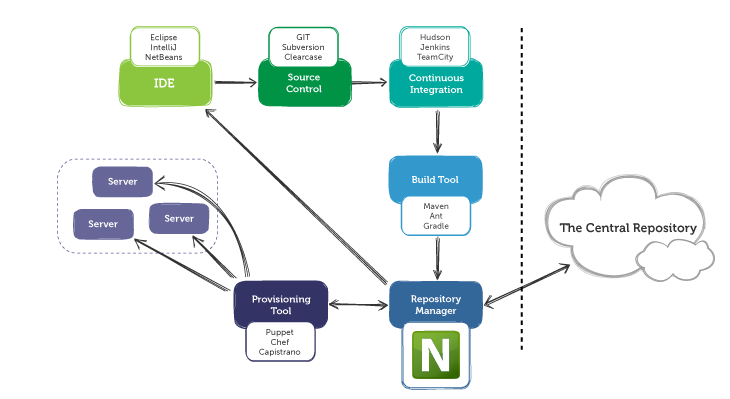
<outputDirectory>C:\MVN\project\target/site</outputDirectory>

</reporting>

</project>

In above pom.xml, you can see the default project source folders structure, output directory, plug-ins required, repositories, reporting directory, which Maven will be using while executing the desired goals.

Maven pom.xml is also not required to be written manually. Maven provides numerous archetype plugins to create projects, which in order, create the project structure and pom.xml.



Maven Life Cycle

"Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined."

" For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired."

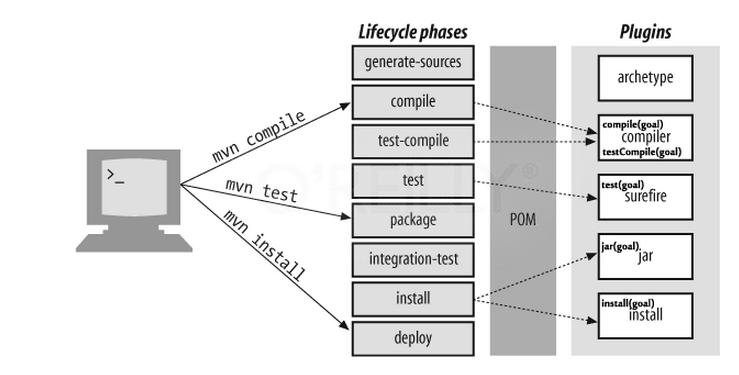
[Apache Maven Project - Introduction](https://maven.apache.org/what-is-maven.html)

There are three built-in build lifecycles:

1. **default** lifecycle handles project deployment.
2. **clean** lifecycle handles project cleaning.
3. **site** lifecycle handles the creation of project's site documentation.

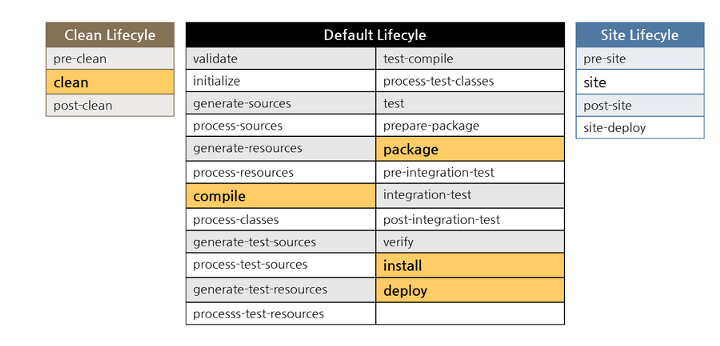
Default lifecycle

The **default** lifecycle has the following build phases:



1. **validate**: validate the project is correct and all necessary information is available.
2. **compile**: compile the source code of the project.
3. **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
4. **package**: take the compiled code and package it in its distributable format, such as a JAR.
5. **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run.
6. **verify**: run any checks to verify the package is valid and meets quality criteria.
7. **install**: install the package into the local repository, for use as a dependency in other projects locally.
8. **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle.



To do all those, we only need to call the last build phase to be executed, in this case, deploy:

$ mvn deploy

That is because if we call a build phase, it will execute not only that build phase, but also every build phase prior to the called build phase

mvn clean install

The following command tells Maven to do the **clean** action in each module before running the **install** action for each module.

$ mvn clean install

In other words, **mvn clean install** clears any compiled files we have, making sure that we're really compiling each module from scratch.

Note that **clean** is in a separate lifecycle, so it's not called by default.

A target folder holds Maven-generated temporary files and artifacts. There are times when the target folder becomes huge or when certain files that have been cached need to be cleaned out of the folder. The **clean** goal accomplishes exactly that, as it attempts to delete the target folder and all its contents.

|  |
| --- |
|  |