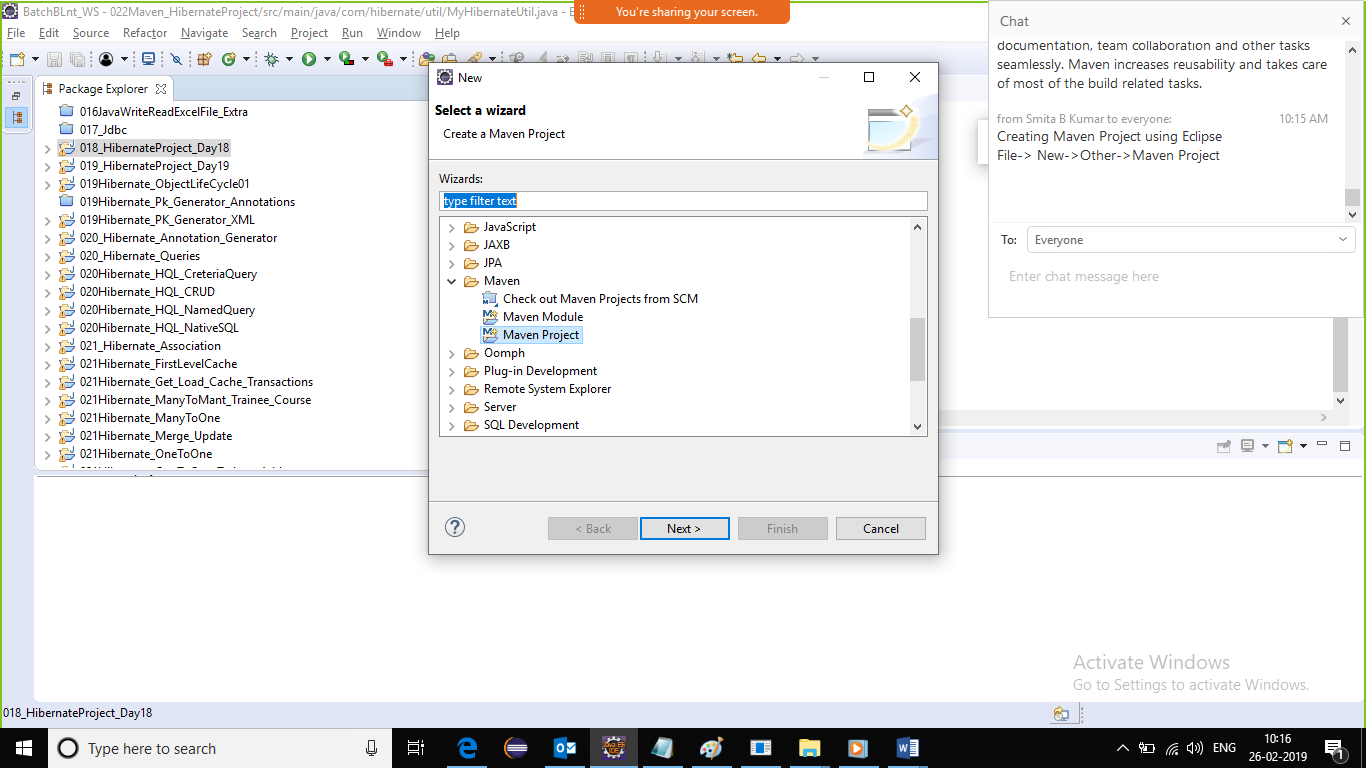
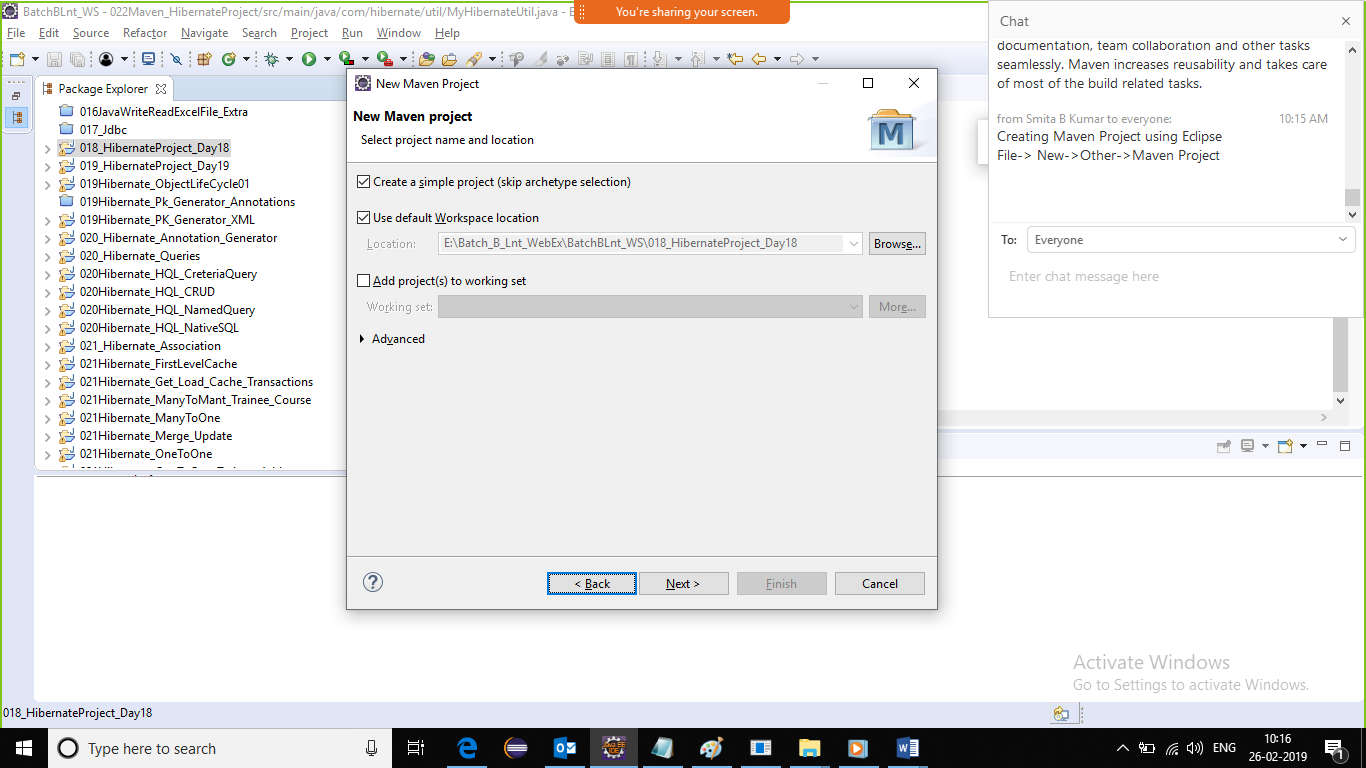
What is Maven?

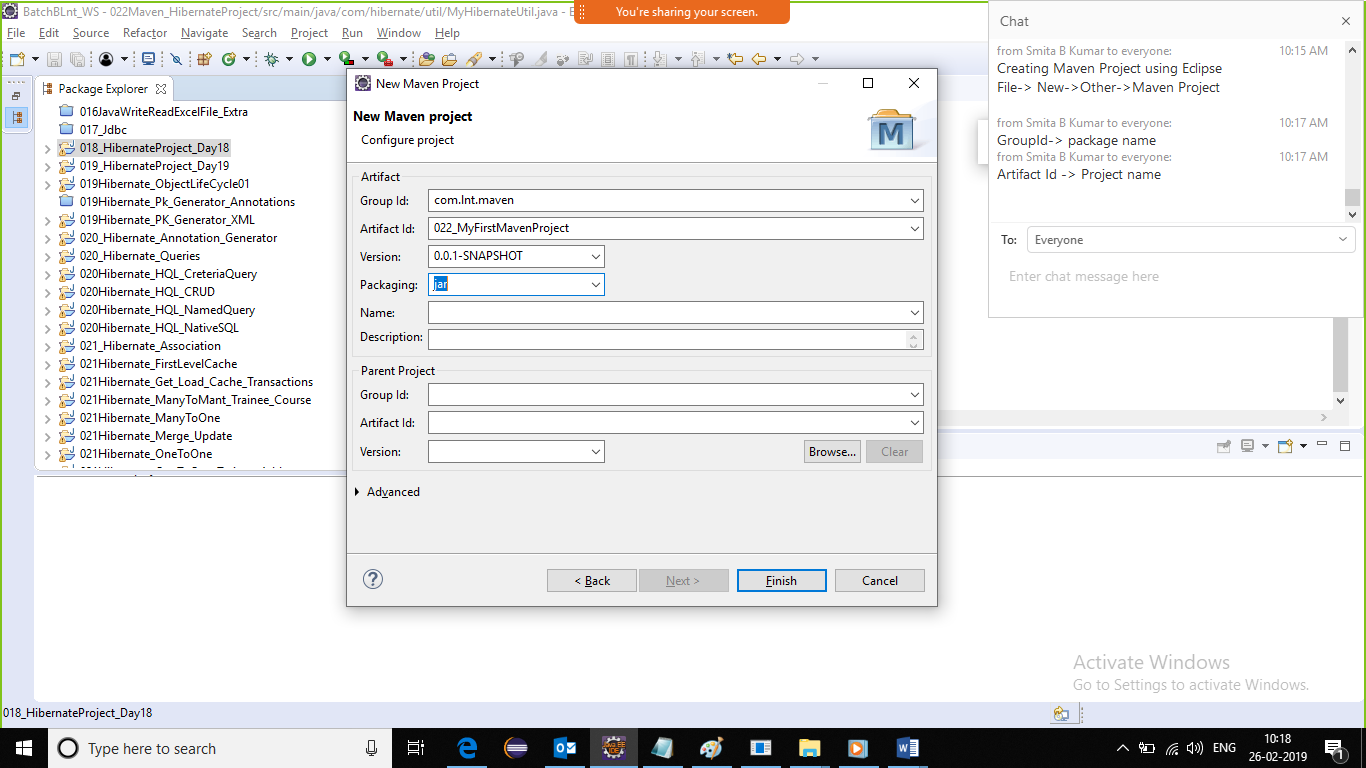
1. Maven is a Project Management, Build and deployment tool
2. It helps to Automate project Build in almost 5% time.
3. It follows Standard Directory Layout & a default build lifeycle...
4. It help in repository management.(.jar files) local, remote and central repositories.
5. Plugins management, Documentation, Reporting.
6. It follows convention over configuration.
7. 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 the build related tasks.

Creating Maven Project using Eclipse

*File-> New->Other->Maven Project*







*Finish -> update pom.xml -> rt click on project ->Maven->update project -> check force update.*

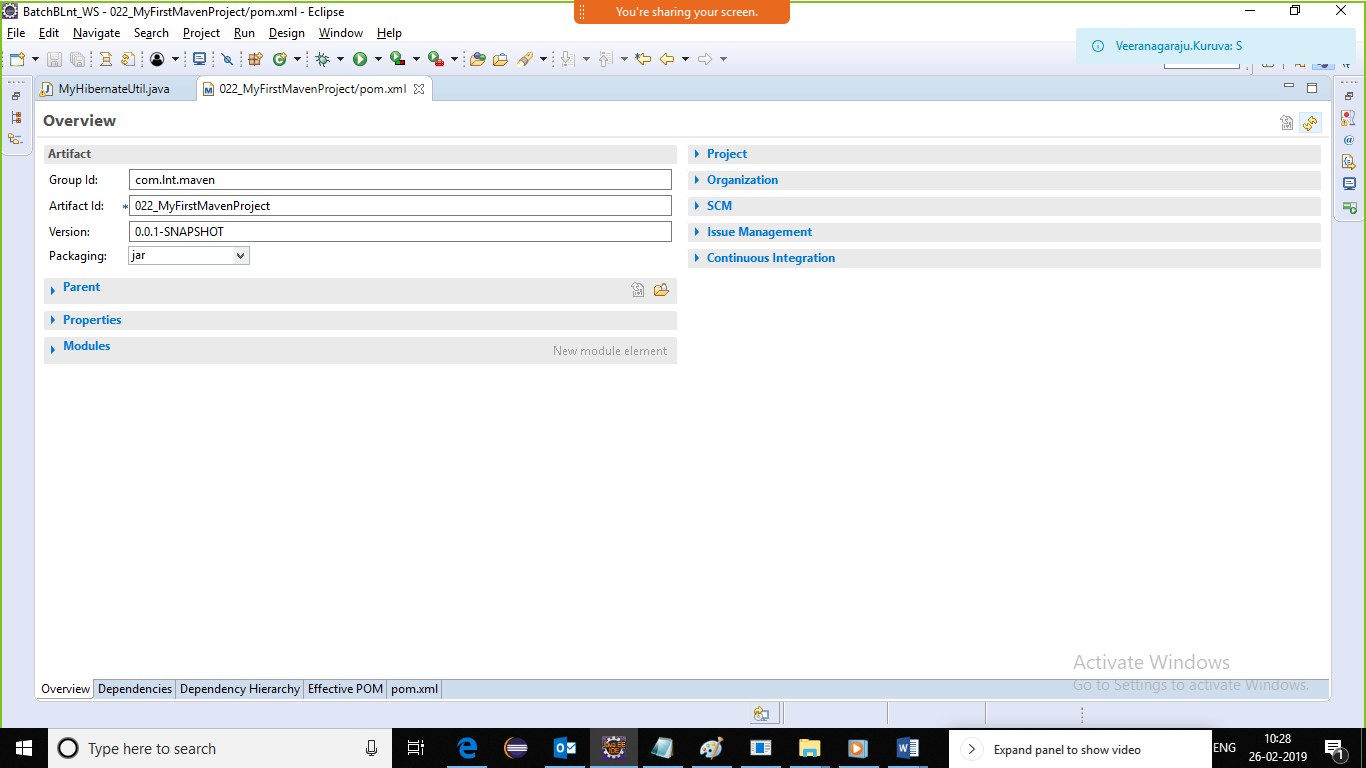
*Copy the hibernate.cfg.xml file in src/main/resources*

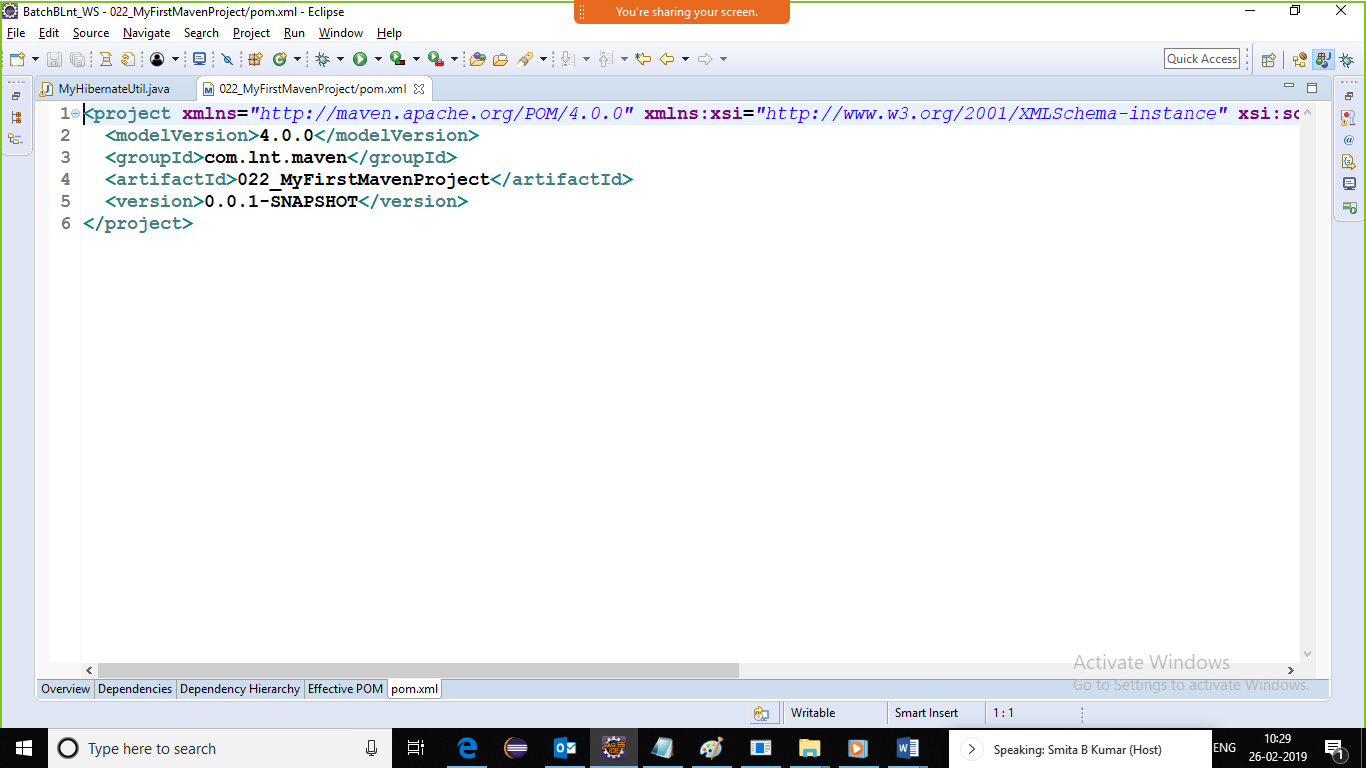
*Create all the java file in src/main/java*

1. *src/main/java -> for storing .java files*
2. *src/main/resources -> for storing ‘.propeties’,’.xml’ etc. configuration files required for .java files*
3. *src/test/java -> for storing testcases .java files*
4. *src/test/resources -> for storing ‘.propeties’,’.xml’ etc. configuration files required for execution of testcases*
5. *target* 
   1. *-> classes ->.class file from main/java*
   2. *-> test-classes->.class file from test/java*
   3. *->packaged jar/war file*

POM stands for Project Object Model. It is fundamental unit of work in Maven. It 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).





Maven is a project management and comprehension tool that 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 the 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 the build related tasks.

Maven Evolution

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.

Objective

The primary goal of Maven is to provide developer with the following −

* 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. In later chapters, we will explain POM in 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 |
| Complied byte code | ${basedir}/target |
| distributable JAR | ${basedir}/target/classes |

In order to build the project, Maven provides developers with options to mention life-cycle goals and project dependencies (that rely on Maven plugin 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 the need to understand how the individual plugins work. We will discuss Maven Plugins in detail in the later chapters.

Features of Maven

* Simple project setup that follows best practices.
* Consistent usage across all projects.
* Dependency management including automatic updating.
* A large and growing repository of libraries.
* Extensible, with the ability to easily write plugins in Java or scripting languages.
* Instant access to new features with little or no extra configuration.
* **Model-based builds** − Maven is able to build any number of projects into predefined output types such as jar, war, metadata.
* **Coherent site of project information** − Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.
* **Release management and distribution publication** − Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
* **Backward Compatibility** − You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
* **Automatic parent versioning** − No need to specify the parent in the sub module for maintenance.
* **Parallel builds** − It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.
* **Better Error and Integrity Reporting** − Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.
* Maven is a Java based tool, so the very first requirement is to have JDK installed on your machine.

## System Requirement

|  |  |
| --- | --- |
| **JDK** | 1.7 or above. |
| **Memory** | No minimum requirement. |
| **Disk Space** | No minimum requirement. |
| **Operating System** | No minimum requirement. |

Step 1 - Verify Java Installation on your Machine

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\_60"  Java(TM) SE Runtime Environment (build 1.7.0\_60-b19)  Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode) |
| Linux | java version "1.7.0\_60"  Java(TM) SE Runtime Environment (build 1.7.0\_60-b19)  Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode) |
| Mac | java version "1.7.0\_60"  Java(TM) SE Runtime Environment (build 1.7.0\_60-b19)  Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode) |

If you do not have Java installed, install the Java Software Development Kit (SDK) from <https://www.oracle.com/technetwork/java/javase/downloads/index.html>. We are assuming Java 1.7.0.60 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\_60 |
| 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\jdk1.7.0.60\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 as explained above.

Step 3 - Download Maven Archive

Download Maven 2.2.1 from <https://maven.apache.org/download.cgi>.

|  |  |
| --- | --- |
| **OS** | **Archive name** |
| Windows | apache-maven-3.3.1-bin.zip |
| Linux | apache-maven-3.3.1-bin.tar.gz |
| Mac | apache-maven-3.3.1-bin.tar.gz |

Step 4 - Extract the Maven Archive

Extract the archive, to the directory you wish to install Maven 3.3.1. The subdirectory apache-maven-3.3.1 will be created from the archive.

|  |  |
| --- | --- |
| **OS** | **Location (can be different based on your installation)** |
| Windows | C:\Program Files\Apache Software Foundation\apache-maven-3.3.1 |
| 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-3.3.1 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-3.3.1 export M2=$M2\_HOME/bin  export MAVEN\_OPTS=-Xms256m -Xmx512m |
| Mac | Open command terminal and set environment variables.  export M2\_HOME=/usr/local/apache-maven/apache-maven-3.3.1  export M2=$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 and 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 as follows −

|  |  |
| --- | --- |
| **OS** | **Output** |
| Windows | Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.7.0\_60  Java home: C:\Program Files\Java\jdk1.7.0\_60 \jre |
| Linux | Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.7.0\_60  Java home: C:\Program Files\Java\jdk1.7.0\_60 \jre |
| Mac | Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.7.0\_60  Java home: C:\Program Files\Java\jdk1.7.0\_60 \jre |

