

Maven Cheat Sheet



Maven™

Why is Software Build?

In the context of software development, Software Build refers to the process of creation of a software and the associated processes including: compiling computer source code into binary code, packaging binary code, and running automated tests.

Build automation is the act of scripting or automating a wide variety of simple/complicated, repeatable tasks:

- Compiling source code
- Packing binaries
- Running automated tests
- Deploying to production system
- Creating documentation

The advantages of build automation to software development projects include

- A necessary pre-condition for continuous integration and continuous testing
- Improve product quality
- Accelerate the compile and link processing
- Eliminate redundant tasks
- Minimize "bad builds"

- Eliminate dependencies on key personnel
- Have history of builds and releases in order to investigate issues
- Save time and money

What is Maven?

Maven is a Build & an project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plugin goals at defined phases in a lifecycle.

Before Maven provided a common interface for building software, every single project had someone dedicated to managing a fully customized build system. Developers had to take time away from developing software to learn about the idiosyncrasies of each new project they wanted to contribute to.

If a new source code analysis tool came out that would perform static analysis on source code, or if someone developed a new unit testing framework, everybody would have to drop what they were doing and figure out how to fit it into each project's custom build environment. How do you run unit tests? There were a thousand different answers. This environment was characterized by a thousand endless arguments about tools and build procedures.

While Maven provides an array of benefits including dependency management and reuse of common build logic through plugins, the core reason why it has succeeded is that it has defined a common interface for building software. Maven provides:

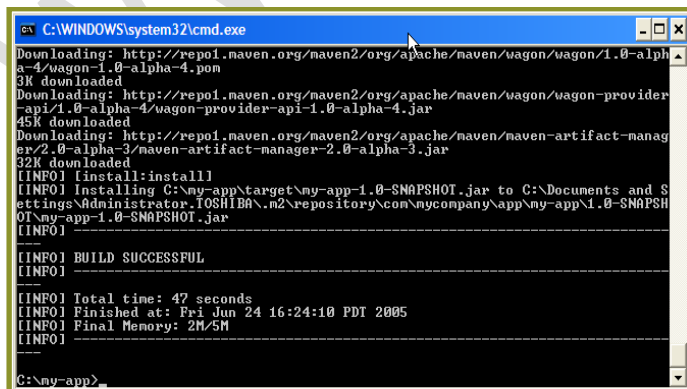
- Easy Build Process
- Uniform Build System
- Quality Project Information
- Guidelines for Best Practices Development

Achieved Characteristics:

- Visibility
- Reusability
- Maintainability
- Comprehensibility "Accumulator of Knowledge"

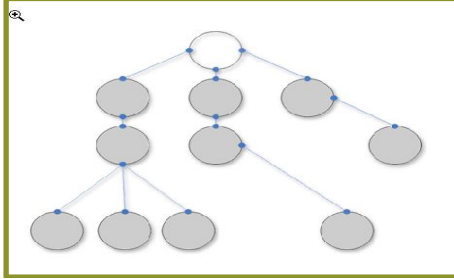
Main Features of Maven:

➤ Build Tool



```
C:\WINDOWS\system32\cmd.exe
Downloading: http://repo1.maven.org/maven2/org/apache/maven/wagon/wagon/1.0-alpha-4/wagon-1.0-alpha-4.pom
3K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/wagon/wagon-provider-api/1.0-alpha-4/wagon-provider-api-1.0-alpha-4.jar
45K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-artifact-manager/2.0-alpha-3/maven-artifact-manager-2.0-alpha-3.jar
32K downloaded
[INFO] [install:install]
[INFO] Installing C:\my-app\target\my-app-1.0-SNAPSHOT.jar to C:\Documents and Settings\Administrator\TOSHIBA\m2\repository\com\mycompany\app\my-app\1.0-SNAPSHOT\my-app-1.0-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESSFUL
[INFO]
[INFO] Total time: 47 seconds
[INFO] Finished at: Fri Jun 24 16:24:10 PDT 2005
[INFO] Final Memory: 2M/5M
[INFO]
C:\my-app>
```

➤ Dependency Management Tool



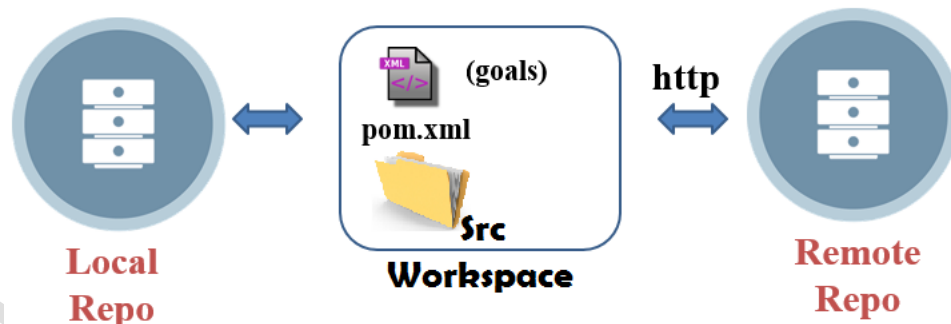
➤ Documentation Tool



Maven Architecture

In Maven terminology, a repository is a 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. The following illustration will give an idea regarding these 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. This 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.

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 – <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows –

- ✓ 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 – <https://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

Remote Repository:

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the 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.

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 perform 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 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 stops processing and throws error (Unable to find dependency).

Maven Build Lifecycle

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

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.

The 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.

```
mvn <goals> -> plugins -> task(s)
```

Maven Standard Directory Structure

If you were to create a Maven project by hand this is the directory structure that we recommend using. Having a common directory layout would allow for users familiar with one Maven project to immediately feel at home in another Maven project. The advantages are analogous to adopting a site-wide look-and-feel.

```
├── maven-project
│   ├── pom.xml
│   ├── README.txt
│   ├── NOTICE.txt
│   ├── LICENSE.txt
│   └── src
│       ├── main
│       │   ├── java
│       │   ├── resources
│       │   ├── filters
│       │   └── webapp
│       ├── test
│       │   ├── java
│       │   ├── resources
│       │   └── filters
│       ├── it
│       ├── site
│       └── assembly
```

Maven GAV

Maven uniquely identifies any project using GAV.

- **groupId:** Arbitrary project grouping identifier (no spaces or colons)
- **artifactId:** Arbitrary name of project (no spaces or colons)
- **version:** Version of project
 - Format {Major}.{Minor}.{Maintenance}
 - Add '-SNAPSHOT' to identify in development

GAV Syntax: **groupId:artifactId:version**

Environment Setup

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

Set1: Verify Java Installation on your Machine

```
$ java -version
```

Step 2 - Set JAVA_HOME Environment variable

Step 3 - Download Maven Archive from <https://maven.apache.org/download.cgi>

Step 4 - Extract the Maven Archive

Step 5 - Set Maven Environment Variables - M2_HOME, M2

Step 6 - Add Maven bin Directory Location to System Path

Setup Maven on Linux

```
% yum install java-1.8.0-openjdk-devel (install JDK, check the binary link)
```

```
% cd /usr/local
```

```
% wget https://archive.apache.org/dist/maven/maven-3/3.5.3/binaries/apache-maven-3.5.3-bin.tar.gz % tar xzf apache-maven-3.5.3-bin.tar.gz
```

```
% ln -s apache-maven-3.5.3 maven
```

```
export M2_HOME=/usr/local/maven
```

```
export PATH=${M2_HOME}/bin:${PATH}
```

POM

POM Stands for Project Object Model. As a fundamental unit of work in Maven, POM is an XML file that contains information about project and configuration details used by Maven to build the project

Some of the configuration that can be specified in the POM are following

- Name and Version
- Artifact Type
- Source Code Locations
- Dependencies
- Plugins
- Profiles (Alternate build configurations)

Creating Project

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 the command console, go to the C:\MVN directory and execute the following mvn command.

```
C:\MVN> mvn archetype:generate  
-DgroupId = wezva
```

-DartifactId = builddemo
-DarchetypeArtifactId = maven-archetype-quickstart
-DinteractiveMode = false

Create sample web project:

C:\MVN> mvn archetype:generate -DgroupId=demo -DartifactId=samplewar -
DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

Maven Plugins?

Maven is actually a plugin execution framework where every task is actually done by plugins

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

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

Plugin Types

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

- Plugins are specified in pom.xml using plugins element.
- Each plugin can have multiple goals.
- You can define phase from where plugin should start its processing using its phase element. You can configure tasks to be executed by binding them to goals of plugin.
- That's it, Maven will handle the rest. It will download the plugin if not available in local repository

Syntax:

```
<project>  
  <build>  
    <plugins>  
      <plugin>  
        # Which Plugin  
        <groupId>XXX</groupId>  
        <artifactId>YYY</artifactId>
```



```
<version>ZZZ</version>

<executions>
  <execution>
    #When to use
    <phase>PHASENAME</phase>
    <goals>
      <goal>run</goal>
    </goals>
    <configuration>
      # What to do
    </configuration>
  </execution>
</executions>

</plugin>
</plugins>
</build>
```

Example: Check class notes for more examples

Maven Build Profiles?

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 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).

```
<project>
...
  <profiles>
    <profile>
      <build>
        # plugins
      </build>
    </profile>
  </profiles>
...
</project>
```

Active the profile by passing -P<profilename> along with other options while invoking mvn

Maven Snapshot

A large software application generally consists of multiple modules and it is common scenario where multiple teams are working on different modules of same application

- For ex Demo2 team uses Demo.jar
- Now if Demo team builds a new jar
 - Demo should inform every time when they release an updated code
 - Demo2 have to update their pom.xml to get the latest Demo.jar

What is SNAPSHOT?

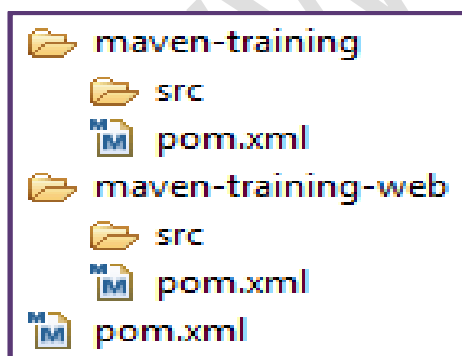
SNAPSHOT is a special version that indicates a current development copy. Unlike regular versions, Maven checks for a new SNAPSHOT version in a remote repository for every build.

Maven Multi Modules

Let's say we have an application which consists of several modules, let it be a front-end module and a back-end module. Now, we work on both of them and change functionality which affects the two. In that case, without a specialized build tool, we'll have to build both components separately or write a script which would compile the code, run tests and show the results. Then, after we get even more modules in the project, it will become harder to manage and maintain.

Besides, in the real world, projects may need certain Maven plugins to perform various operations during build lifecycle, share dependencies and profiles or include other BOM projects.

Therefore, when leveraging multi-modules, we can build our application's modules in a single command and if the order matters, Maven will figure this out for us. Also, we can share a vast amount of configuration with other modules.



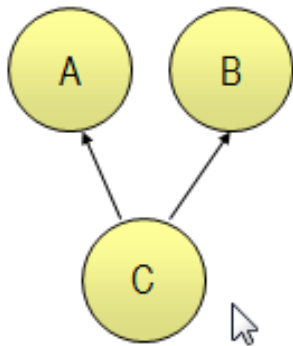
```
<project>
  <groupId>wezva</groupId>
  <artifactId>Parent-module</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <modules>
    <module>Child-jar</module>
    <module>child-war</module>
  </modules>
</project>
```

- In a multi-module project layout, a parent project contains a parent pom.xml and each sub modules (sub projects) also containing own pom.xml
- The packaging value for Parent will be "pom"
- Parent pom.xml adds the list of child modules under <modules> tag
- In the Child pom.xml, we should refer the GAV of the parent using <parent> tag

Example: Check class notes for more examples

```
<project>
...
<parent>
  <groupId>wezva</groupId>
  <artifactId>Parent-module</artifactId>
  <version>1.0-SNAPSHOT</version>
</parent>
<groupId>wezva</groupId>
<artifactId>child-jar</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
...
</project>
```

Maven Dependency Management



```
<dependencies>
  <dependency>
    <groupId>com.some.project</groupId>
    <artifactId>B</artifactId>
    <version>[1.0,)</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

« Any Version After 1.0 »

One of the core features of Maven is Dependency Management. Managing dependencies is a difficult task once we've to deal with multi-module projects (consisting of hundreds of modules/sub-projects). Maven provides a high degree of control to manage such scenarios.

It is pretty often a case, when a library, say A, depends upon other library, say B. In case another project C wants to use A, then that project requires to use library B too.

Maven helps to avoid such requirements to discover all the libraries required. Maven does so by reading project files (pom.xml) of dependencies, figure out their dependencies and so on.

We only need to define direct dependency in each project pom. Maven handles the rest automatically.

Example: Check class notes for more examples

Maven Update Version

- mvn versions:set -DnewVersion=your_new_version

ex : mvn versions:set -DnewVersion=2.1.1

- mvn versions:revert

Command Line Options

- D : defines a system property
- P : activates a profile
- o : works offline without accessing network
- f : specifies an alternate of POM.xml
- s : alternate settings.xml
- fn : never fail the build
- q : show only errors
- X : debug output

www.wezva.com

facebook

<https://www.facebook.com/wezva>

Linked in

<https://www.linkedin.com/in/wezva>



+91-9739110917

+91-9886328782