Maven in 5 Minutes

Prerequisites

You must have an understanding of how to install software on your computer. If you do not know how to do this, please ask someone at your office, school, etc or pay someone to explain this to you. The Maven mailing lists are not the best place to ask for this advice.

Installation

Maven is a Java tool, so you must have Java (http://www.oracle.com/technetwork/java/javase/downloads/index.html) installed in order to proceed.

First, download Maven (../../download.html) and follow the installation instructions (../../install.html). After that, type the following in a terminal or in a command prompt:

```
1. mvn --version
```

It should print out your installed version of Maven, for example:

```
1. Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-24T20:41:47+02:00)
2. Maven home: D:\apache-maven-3.6.0\bin\..
3. Java version: 1.8.0_161, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_
   161\jre
4. Default locale: nl_NL, platform encoding: Cp1252
5. OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Depending upon your network setup, you may require extra configuration. Check out the Guide to Configuring Maven (../mini/guide-configuring-maven.html) if necessary.

If you are using Windows, you should look at Windows Prerequisites (./windows-prerequisites.html) **to ensure that you are prepared to use Maven on Windows.**

Creating a Project

You will need somewhere for your project to reside, create a directory somewhere and start a shell in that directory. On your command line, execute the following Maven goal:

```
1. mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=
   maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```

If you have just installed Maven, it may take a while on the first run. This is because Maven is downloading the most recent artifacts (plugin jars and other files) into your local repository. You may also need to execute the command a couple of times before it succeeds. This is because the remote server may time out before your downloads are complete. Don't worry, there are ways to fix that.

You will notice that the *generate* goal created a directory with the same name given as the artifactId. Change into that directory.

```
1. cd my-app
```

Under this directory you will notice the following standard project structure (../introduction/introduction-to-the-standard-directory-layout.html).

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

The `src/main/java` directory contains the project source code, the `src/test/java` directory contains the test source, and the `pom.xml` file is the project's Project Object Model, or POM.

The POM

The `pom.xml` file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want. The POM is huge and can be daunting in its complexity, but it is not necessary to understand all of the intricacies just yet to use it effectively. This project's POM is:

```
<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.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

What did I just do?

You executed the Maven goal *archetype:generate*, and passed in various parameters to that goal. The prefix *archetype* is the plugin (../../plugins/index.html) that provides the goal. If you are familiar with Ant (http://ant.apache.org), you may conceive of this as similar to a task. This *archetype:generate* goal created a simple project based upon a maven-archetype-quickstart (/archetypes/maven-archetype-quickstart/) archetype. Suffice it to say for now that a *plugin* is a collection of *goals* with a general common purpose. For example the jboss-maven-plugin, whose purpose is "deal with various jboss items".

Build the Project

```
mvn package
```

The command line will print out various actions, and end with the following:

```
 ...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Thu Jul 07 21:34:52 CEST 2011
[INFO] Final Memory: 3M/6M
[INFO] ------------------------------------------------------------------------
```

Unlike the first command executed (*archetype:generate*) you may notice the second is simply a single word - *package*. Rather than a *goal*, this is a *phase*. A phase is a step in the build lifecycle (../introduction/introduction-to-the-lifecycle.html), which is an ordered sequence of phases. When a phase is given, Maven will execute every phase in the sequence up to and including the one defined. For example, if we execute the *compile* phase, the phases that actually get executed are:

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile

You may test the newly compiled and packaged JAR with the following command:

```
java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App
```

Which will print the quintessential:

```
Hello World!
```

Java 9 or later

By default your version of Maven might use an old version of the `maven-compiler-plugin` that is not compatible with Java 9 or later versions. To target Java 9 or later, you should at least use version 3.6.0 of the `maven-compiler-plugin` and set the `maven.compiler.release` property to the Java release you are targetting (e.g. 9, 10, 11, 12, etc.).

In the following example, we have configured our Maven project to use version 3.8.1 of `maven-compiler-plugin` and target Java 11:

```
    <properties>
        <maven.compiler.release>11</maven.compiler.release>
    </properties>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
```

To learn more about `javac`'s `--release` option, see JEP 247 (https://openjdk.java.net/jeps/247).

Running Maven Tools

Maven Phases

Although hardly a comprehensive list, these are the most common *default* lifecycle phases executed.

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

There are two other Maven lifecycles of note beyond the *default* list above. They are

- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

Phases are actually mapped to underlying goals. The specific goals executed per phase is dependant upon the packaging type of the project. For example, *package* executes *jar:jar* if the project type is a JAR, and *war:war* if the project type is - you guessed it - a WAR.

An interesting thing to note is that phases and goals may be executed in sequence.

```
mvn clean dependency:copy-dependencies package
```

This command will clean the project, copy dependencies, and package the project (executing all phases up to *package*, of course).

Generating the Site

```
mvn site
```

This phase generates a site based upon information on the project's pom. You can look at the documentation generated under `target/site`.

Conclusion

We hope this quick overview has piqued your interest in the versatility of Maven. Note that this is a very truncated quick-start guide. Now you are ready for more comprehensive details concerning the actions you have just performed. Check out the Maven Getting Started Guide (./index.html).
