Maven Commands

Jakob Jenkov
Last update: 2021-03-10

Maven contains a wide set of commands which you can execute. Maven commands are a mix of build life cycles, build phases and build goals, and can thus be a bit confusing. Therefore I will describe the common Maven commands in this tutorial, as well as explain which build life cycles, build phases and build goals they are executing.

However, I will first list some common Maven commands with a brief explanation of what they do. After this list of common Maven commands I have a description of the Maven command structure.

Common Maven Commands

Here is a list of common Maven commands plus a description of what they do. Please note, that even if a Maven command is shown on multiple lines in the table low, it is to be considered a single command line when typed into a windows command line or linux shell.

Maven Command	Description
mvn --version	Prints out the version of Maven you are running.
mvn clean	Clears the `target` directory into which Maven normally builds your project.
mvn package	Builds the project and packages the resulting JAR file into the `target` directory.
mvn package -Dmaven.test.skip=true	Builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn clean package	Clears the `target` directory and Builds the project and packages the resulting JAR file into the `target` directory.
mvn clean package -Dmaven.test.skip=true	Clears the `target` directory and builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn verify	Runs all integration tests found in the project.
mvn clean verify	Cleans the target directory, and runs all integration tests found in the project.
mvn install	Builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository
mvn install -Dmaven.test.skip=true	Builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository

mvn clean install -Dmaven.test.skip=true	Clears the `target` directory and builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository
mvn dependency:copy-dependencies	Copies dependencies from remote Maven repositories to your local Maven repository.
mvn clean dependency:copy-dependencies	Cleans project and copies dependencies from remote Maven repositories to your local Maven repository.
mvn clean dependency:copy-dependencies package	Cleans project, copies dependencies from remote Maven repositories to your local Maven repository and packages your project.
mvn dependency:tree	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file.
mvn dependency:tree -Dverbose	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file. Includes repeated, transitive dependencies.
mvn dependency:tree -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact.
mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact. Includes repeated, transitive dependencies.
mvn dependency:build-classpath	Prints out the classpath needed to run your project (application) based on the dependencies configured in the pom.xml file.

Keep in mind, that when you execute the `clean` goal of Maven, the `target` directory is removed, meaning you lose all compiled classes from previous builds. That means, that Maven will have to build all of your project again from scratch, rather than being able to just compile the classes that were changed since last build. This slows your build time down. However, sometimes it can be nice to have a clean, fresh build, e.g. before releasing your product to the world - mostly for your own "feeling" of knowing everything was built from scratch and working.

Maven Command Structure

A Maven command consists of two elements:

- `mvn`
- One or more build life cycles, build phases or build goals

Here is a Maven command example:

This command consists of the `mvn` command which executes Maven, and the build life cycle named `clean`.

Here is another Maven command example:

```
mvn clean install
```

This maven command executes the `clean` build life cycle and the `install` build phase in the `default` build life cycle.

You might wonder how you see the difference between a build life cycle, build phase and build goal. I will get back to that later.

Build Life Cycles, Phases and Goals

As mentioned in the introduction in the section about **Build life cycles, build phases and build goals**, Maven contains three major build life cycles:

- `clean`
- `default`
- `site`

Inside each build life cycle there are build phases, and inside each build phase there are build goals.

You can execute either a build life cycle, build phase or build goal. When executing a build life cycle you execute all build phases (and thus build goals) inside that build life cycle.

When executing a build phase you execute all build goals within that build phase. Maven also executes all build phases earlier in the build life cycle of the desired build phase.

Buid goals are assigned to one or more buid phases. When the build phases are executed, so are all the goals in that build phase. You can also execute a build goal directly.

Executing Build Life Cycles, Phases and Goals

When you run the `mvn` command you pass one or more arguments to it. These arguments specify either a build life cycle, build phase or build goal. For instance to execute the `clean` build life cycle you execute this command:

```
mvn site
```

Executing the Default Life Cycle

The `default` life cycle is the build life cycle which generates, compiles, packages etc. your source code.

You cannot execute the `default` build life cycle directly, as is possible with the `clean` and `site`. Instead you have to execute a specific build phase within the `default` build life cycle.

The most commonly used build phases in the `default` build life cycle are:

Build Phase	Description
`validate`	Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded.
`compile`	Compiles the source code of the project.
`test`	Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
`package`	Packs the compiled code in its distributable format, such as a JAR.
`install`	Install the package into the local repository, for use as a dependency in other projects locally.
`deploy`	Copies the final package to the remote repository for sharing with other developers and projects.

Executing one of these build phases is done by simply adding the build phase after the `mvn` command, like this:

```
mvn compile
```

This example Maven command executes the `compile` build phase of the `default` build life cycle. This Maven command also executes all earlier build phases in the `default` build life cycle, meaning the `validate` build phase.

Executing Build Phases

You can execute a build phase located inside a build life cycle by passing the name of the build phase to the Maven command. Here are a few build phase command examples:

```
mvn package
```

Maven will find out what build life cycle the specified build phase belongs to, so you don't need to explicitly specify which build life cyle the build phase belongs to.

Next: Build a Fat JAR With Maven

Tweet

Jakob Jenkov

Sponsored Ads