Maven Commands List PDF

In this post gives you all the useful maven commands used in maven applications.

1. Create Maven Projects

Create a Java project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

Create a web project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

Create archetype from an existing project

```
mvn archetype:create-from-project
```

2. Main phases

- *clean* — delete target directory
- *validate* — validate, if the project is correct
- *compile* — compile source code, classes stored in target/classes
- *test* — run tests
- *package* — take the compiled code and package it in its distributable format, e.g. JAR, WAR
- verify — run any checks to verify the package is valid and meets quality criteria
- *install* — install the package into the local repository
- deploy — copies the final package to the remote repository

3. Maven phase commands(Project Build Commands)

clean project: will delete target directory

```
mvn clean
```

validate project: validate the project is correct and all necessary information is available

```
mvn validate
```

compile project: compile source code, classes stored in target/classes

```
mvn compile
```

test project: run tests using a suitable unit testing framework

```
mvn test
```

package project: take the compiled code and package it in its distributable format, such as a JAR /WAR

```
mvn package
```

verify project: run any checks to verify the package is valid and meets quality criteria

```
mvn verify
```

install project: install the package into the local repository, for use as a dependency in other projects locally

```
mvn install
```

deploy project: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects

```
mvn deploy
```

Skip running tests

Compiles the tests, but skips running them.

```
mvn install -DskipTests=true
```

Skips compiling the tests and does not run them.

```
mvn install -Dmaven.test.skip=true
```

4. Project Site Generation

Generate site without tests reports (tests are not executed):

```
mvn site:site
```

Generate site with unit tests reports:

```
mvn test site:site
```

Generate site with unit and integration tests reports:

```
mvn verify site:site
```

5. Code quality analysis

Analyze code quality with Sonar:

```
mvn clean install -DskipTests=true
mvn sonar:sonar
```

Read Sonar configuration guide.

6. Code coverage reporting

Notice:

> It is much more feasible to generate code coverage reports directly from IDE than from Maven. Write test, write code, run coverage for separated test, and check that all important branches are covered.

Generate Clover reports for unit tests:

```
mvn clover2:setup test clover2:aggregate clover2:clover
```

Generate clover reports for unit and integration tests:

```
mvn clover2:setup verify clover2:aggregate clover2:clover
```

Read Clover configuration guide.

7. Dependency Management

Check dependencies for newer versions:

```
mvn versions:display-dependency-updates
```

Check plugins for newer versions:

```
mvn versions:display-plugin-updates
```

Check for newer versions defined as properties:

```
mvn versions:display-property-updates
```

Display project dependencies:

```
mvn dependency:tree
```

Analyze project dependencies:

```
mvn dependency:analyze
```

8. Getting Help

Display effective Maven settings:

```
mvn help:effective-settings
```

Display effective POM:

```
mvn help:effective-pom
```

Display all profiles (from `settings.xml` and POMs hierarchy):

```
mvn help:active-profiles
```

Display plugin goals (for `m-compiler-p` in the example below):

```
mvn compiler:help
```

Display plugin's goal description (for goal `compile` in `m-compiler-p` in the example below):

```
mvn compiler:help -Dgoal=compile -Ddetail
```

Help plugin — used to get relative information about a project or the system.

- *mvn help:describe* describes the attributes of a plugin
- *mvn help:effective-pom* displays the effective POM as an XML for the current build, with the active profiles factored in. Dependency plugin — provides the capability to manipulate artifacts.
- mvn dependency:analyze analyzes the dependencies of this project
- mvn dependency:tree prints a tree of dependencies Compiler plugin — compiles your java code. Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
 artifactId>
 <version>3.6.1</version>
 <configuration>
 <source>1.8</source>
 <target>1.8</target>
 </configuration>
</plugin>
```

> Learn maven here at Apache Maven Tutorial