Maven cheat sheet

Do you use Apache Maven every day to build your Java projects? But you always have to look up specific commands or options like me?
Check out my Maven cheat sheet!

Initialize a new Maven Project

Initialize a new Maven project:

```
mvn archetype:generate -DgroupId=io.stockgeeks \
                       -DartifactId=project-example
```

Initialize a new Maven project in non-interactive mode:

```
mvn archetype:generate -DgroupId=io.stockgeeks \
                       -DartifactId=project-example \
                       -DinteractiveMode=false
```

Initialize a new Maven web project:

```
mvn archetype:generate -DgroupId=io.stockgeeks \
                       -DartifactId=webproject-example \
                       -DarchetypeArtifactId=maven-archetype-
webapp
```

Initialize a Spring Boot Maven project:

The easiest way to initialize a Spring Boot Maven project is to go https://start.spring.io/ select your dependencies and download the Maven project as a zip file.

Package and install your Maven project

Both `package`and `install` are the most used Maven phases in the build lifecycle. For a complete overview of all phases, see the Maven lifecycle documentation.

- **package:** takes the compiled code and packages it in its distributable formats, such as a JAR or WAR. Make jar not war ☺
- **install:** installs the packaged artifact into the local Maven repository, for use as a dependency in other projects locally.

Package your Maven project:

```
mvn package
```

Package but clean the target directory first:

```
mvn clean package
```

Package (clean) the build in parallel:

Building in parallel is an experimental feature since Maven 3.x and will most likely speed up your build. Make sure your project can build in parallel! Plugins you are using in your project should be thread-safe. There is no guarantee that non-thread safe plugins will run correctly and might screw up your artifact.

Example of executing your build with four concurrent threads:

```
mvn clean package -T 4
```

One thread per CPU core:

```
mvn -T 1C clean package
```

1.5 thread per CPU core:

```
mvn -T 1.5C clean install
```

For more information, check out the Maven documentation about parallel builds.

Package a specific Maven module:

```
mvn clean package -pl name-of-the-module
```

or multiple modules

```
mvn clean package -pl myproject-commons,myproject-service
```

Both `-pl` and `--project` are equivalent. Make sure there is no space between the comma and the modules to package! Otherwise, you will run into an error like:

```
[ERROR] Unknown lifecycle phase "myproject-service". You must specify a valid lifecycle phase or a goal in the format <plugin-prefix>:<goal> or <plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal>.
```

Don’t run tests:

```
mvn clean package -DskipTests
```

Note: the test classes in the project will be compiled!

Don’t compile and don’t run the tests:

```
maven clean package -Dmaven.test.skip=true
```

`maven.test.skip`is honored by the Surefire, Failsafe and the Compiler Plugin

Run a single test:

Sometimes you would like to execute a single test instead of all your tests.

```
mvn test -Dtest="NameOfYourTest"
```

Run build offline

```
mvn clean package -o
```

Both `-o` and `--offline` are equivalent. Your local Maven repository will is used to resolve dependencies. No connection to the internet is made to download dependencies. Your build will fail in case dependencies are not found in your local repository!

Dependencies

Search for dependencies in your project

List all the dependencies

```
mvn dependency:list
```

Check whether or not you have specific dependencies in your project

```
mvn dependency:list | grep log4j
```

Get a single dependency

```
mvn dependency:get -Dartifact=org.springframework:spring-core:5.3.15
```

Local repository

The default location of your local repo is `~/.m2`

Install a (3rd party) jar file into your local Maven repository since it doesn't exist in any public repository like Maven Central.

```
mvn install:install-file
   -Dfile=<path-to-file>
   -DgroupId=<group-id>
   -DartifactId=<artifact-id>
   -Dversion=<version>
   -Dpackaging=<packaging>
   -DgeneratePom=true
```

Where:

- `<path-to-file>` the path to the file to load
- `<group-id>` the group that the file should be registered under
- `<artifact-id>` the artifact name for the file
- `<version>` the version of the file
- `<packaging>` the packaging of the file, e.g., jar

Example:

```
mvn install:install-file -Dfile=lang-groovy-5.2.2.jar \
                         -DgroupId=org.elasticsearch.module \
                         -DartifactId=lang-groovy \
                         -Dversion=5.2.2 \
                         -Dpackaging=jar \
                         -DgeneratePom=true
```

Plugins

Describe the available goals of a Maven plugin:

Most recent Maven plugins have a help goal in the command line the description of the plugin, with their parameters and types. Help goal on the Spring Boot Maven Plugin:

```
mvn org.springframework.boot:spring-boot-maven-plugin:help
```

Show the defaults of a specific plugin goal:

```
mvn org.springframework.boot:spring-boot-maven-plugin:help -
Ddetail -Dgoal=start
```

For old(er) Maven plugins that don't offer a help goal, you can use:

```
mvn help:describe -DgroupId=org.apache.maven.plugins \
                  -DartifactId=maven-war-plugin \
                  -Dversion=2.0 \
                  -Ddetail=true
```

Show Maven plugins used in your project:

```
mvn dependency:resolve-plugins
```

Debug the source code of a Maven plugin:

Did you ever been in the situation you didn't understand how a goal of a Maven plugin works? Check out the source code of the plugin run it in debug mode and step through the code step by step:

1. From the command line, run a goal of a Maven plugin with `mvnDebug` instead of `mvn`

```
mvnDebug dependency-check:check
Preparing to execute Maven in debug mode
Listening for transport dt_socket at address: 8000
```

The plugin will not start but will wait until you connect.

2. Checkout and open the source of the Maven plugin you want to debug in IntelliJ and set a breakPoint in the code that is matching the goal you want to debug.

3. In IDEA, add a Remote Configuration. Under Settings, set

- Transport: Socket
- Debugger Mode: Attach
- Host: localhost
- Port: 8000 (default port of mvnDebug)

4. Run the configuration in Debug mode from IntelliJ. It should connect to the waiting mvnDebug JVM. The plugin will execute the goal, and your breakpoint will be hit.

Display parent project updates for your project

By executing the command:

```
mvn versions:display-parent-updates
```

Example output of a Spring Boot project where the parent project has a newer version available:

```
[INFO] artifact org.springframework.boot:spring-boot-starter-
parent: checking for updates from spring-milestones
[INFO] artifact org.springframework.boot:spring-boot-starter-
parent: checking for updates from central
[INFO] The parent project has a newer version:
[INFO]   org.springframework.boot:spring-boot-starter-parent
2.5.1 -> 2.5.2
```

Display dependency updates for your project

By executing the command:

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

This goal will check all the properties in your project and display a list of those properties which are used to control versions of:

- dependencies
- plugins
- and plugin dependencies

and it will display which properties have newer versions available.

Example output where there is a newer version available of Axon Framework:

```
[INFO] The following version property updates are available:
[INFO]   ${axon.version} ..................................
4.5.1 -> 4.5.2
[INFO]
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
```

Display plugin updates for your project

By executing the command:

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

Sources

Downloaded the sources of the dependencies used in the project:

```
mvn dependency:sources
```

Download the sources matching specific groupId(s):

```
mvn dependency:sources -DincludeGroupIds=org.springframework
```

Download the sources of the dependencies matching both specific groupId(s) and artifactId(s). In this example, we download the sources of spring-boot-actuator only:

```
mvn dependency:sources -
DincludeGroupIds=org.springframework.boot -
DincludeArtifactIds=spring-boot-actuator
```

Javadoc

Download the Javadoc of all dependencies in your project:

```
mvn dependency:resolve -Dclassifier=javadoc
```

Other useful commands

Show the dependency tree:

Show all the dependencies of the Maven project in a 'dependency tree' and most importantly get insights in the *transitive dependencies* (the dependencies of that comes with your dependencies)

```
mvn dependency:tree
```

Inspecting the 'dependency tree' in the command line is hard I always write the output of the command to a file so I can check the dependency tree in my favorite editor.

```
mvn dependency:tree > tree.txt
```

Show the dependency tree matching: `com.fasterxml.jackson.datatype`

```
mvn dependency:tree -Dincludes=com.fasterxml.jackson.datatype
```

In a Spring Boot project, two dependencies (jackson-datatype-jsr310 and jackson-datatype-jdk8)
matching `com.fasterxml.jackson.datatype` are found.

```
[INFO] --- maven-dependency-plugin:3.1.1:tree (default-cli) @ stock-tick-consumer-avro ---
[INFO] io.stockgeeks.kafka.stock.tick.consumer.avro:stock-tick-consumer-avro:jar:0.0.1-SNAPSHOT
[INFO] +- org.springframework.boot:spring-boot-starter-actuator:jar:2.2.3.RELEASE:compile
[INFO] |  \- org.springframework.boot:spring-boot-actuator-autoconfigure:jar:2.2.3.RELEASE:compile
[INFO] |     \- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.10.2:compile
[INFO] \- org.springframework.boot:spring-boot-starter-web:jar:2.2.3.RELEASE:compile
[INFO]    \- org.springframework.boot:spring-boot-starter-json:jar:2.2.3.RELEASE:compile
[INFO]       \- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.10.2:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.428 s
[INFO] Finished at: 2020-01-19T00:35:35+01:00
[INFO] ------------------------------------------------------------------------
```

Show the version of your Maven project:

```
mvn org.apache.maven.plugins:maven-help-plugin:evaluate \
    -Dexpression=project.version
```

Less verbose ways to show the version of the Maven project:

```
printf 'VERSION=${project.version}\n0\n' | mvn
org.apache.maven.plugins:maven-help-plugin:evaluate | grep
'^VERSION'
```

or:

```
mvn help:evaluate -Dexpression=project.version -q -DforceStdout
```

or:

```
mvn -q -Dexec.executable=echo -Dexec.args='${project.version}' -
-non-recursive exec:exec
```

Set the version of the project

```
mvn versions:set -DnewVersion=1.0.0-RELEASE
```