There are two locations where a settings.xml file may live:

* The Maven Global install: ${maven.home}/conf/settings.xml
* A user’s install: ${user.home}/.m2/settings.xml

The former settings.xml are also called global settings, the latter settings.xml are referred to as user settings. If both files exist, their contents get merged, with the user-specific settings.xml being dominant.

**Build Life Cycles, Phases and Goals**

As mentioned in the introduction in the section about [Build life cycles, build phases and build goals](http://tutorials.jenkov.com/maven/maven-tutorial.html#maven-build-life-cycles-phases-and-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.

Build 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 clean

To execute the site 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 pre-clean

mvn compile

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 cycle the build phase belongs to.

**Maven Command Line Cheatsheet**

A quick reference to Maven commands.

**mvn -Dtest=<unqualified-classname> test**

Tests only the specified class

**mvn -Dmaven.surefire.debug test**

Enables remote debugging of tests on port 5005. Surefire will block on the port until you connect with your debugger.

**mvn -Dcargo.wait=true -P int integration-test**

Runs CS in Tomcat via cargo

**mvn help:effective-pom**

Shows the logical contents of a pom.xml, including contents inherited from the parent pom.xml, up to and including the Maven super POM.

**mvn dependency:tree**

Shows all dependencies (includes transitive dependencies) of your project. This is very helpful for debugging dependency version issues.  
 **mvn -X <package-name>**

Shows all explicit and transitive dependencies for a package, helping to identify conflicts

For more cheat sheet visit **http://ronnieroller.com/maven**

[**http://maven.apache.org/ref/2.2.1/maven-settings/settings.html**](http://maven.apache.org/ref/2.2.1/maven-settings/settings.html)

[**http://www.captaindebug.com/2012/03/using-mavens-u-command-line-option.html**](http://www.captaindebug.com/2012/03/using-mavens-u-command-line-option.html)

**http://codethataint.com/blog/what-are-maven-snapshots/  
  
Multi Module Maven Project**When Maven is executed against a project with sub modules, Maven ﬁrst loads the parent POM and locates the entire sub module POMs. Maven then puts these entire project POMs into something called the Maven Reactor which analyzes the dependencies between modules. The Reactor takes care of ordering components to ensure that interdependent modules are compiled and installed in the proper order.  
  
The Reactor preserves the order of modules as deﬁned in the POM unless changes need to be made. A helpful mental model for this is to picture that modules with dependencies on sibling projects are "pushed down" the list until the dependency ordering is satisﬁed. On rare occasions, it may be handy to rearrange the module order of your build—for example if you want a frequently unstable module towards the beginning of the build.  
  
A good rule of thumb in Maven is to always declare explicit dependencies for classes referenced in your code. If you are going to be importing Commons BeanUtils classes, you should also be declaring a direct dependency on Commons BeanUtils. Fortunately, via bytecode analysis, the Maven Dependency plugin is able to assist you in uncovering direct references to dependencies

**mvn dependency : analyze**  
To analyze duplicate dependencies use : **mvn dependency : analyze-duplicate**used and declared; used and undeclared; unused and declared : **mvn dependency : analyze**to list all dependencies in project : **mvn dependency : list**To Purge local repository : **mvn dependency : purge-local-repository**Remove snapshots : **mvn dependency : purge-local-repository -DsnapshotsOnly=true -DreResolve=false -Dverbose=true**  
Sort/clean up pom.xml : **mvn com.github.ekryd.sortpom:sortpom-maven-plugin:2.8.0:sort -Dsort.expandEmptyEleme:nts=false -Dsort.predefinedSortOrder=recommended\_2008\_06**

Download Eclipse sources**: mvn eclipse: eclipse -DdownloadSources=true**if you ever wish to stop deleting a target directory then configure the clean plugin as follows:  
<plugin>   
 <artifactId>maven-clean-plugin</artifactId>   
<version>2.6</version>   
<configuration> **<skip>true</skip>** </configuration>   
</plugin>

**Skipping of test compilation and execution**Issuing an mvn command with a command-line parameter: The tests can be skipped even by issuing the following command: mvn –DskipTests but if you need to stop even compilation of test resources then use   
**mvn –Dmaven.test.skip=true** **package** This will completely skip the test compilation and test execution.

**Scopes in Maven for dependency**

|  |  |
| --- | --- |
| Scopes | Description |
| Compile | **This dependency is required for compilation. This automatically means it is required for testing as well as runtime (when the project is run).** |
| Test | **This dependency is only required for tests. This means the dependency is typically in the test code. As the test code is not used to run the project, these dependencies are not required for runtime.** |
| Runtime | **These dependencies are not required during compilation, but only required to run the project. One example would be the log back dependency if you are using Simple Logging Facade for Java (slf4j) to log and want to use log back binding** |
| Provided | **This tells Maven that dependency is required for compilation and runtime, but this dependency need not be packaged with the package for distribution. The dependency will be provided by the user. An example of this dependency is servlet-api. Typically, application servers have these libraries.** |
| System | **This is similar to the provided scope. Here, we need to explicitly provide the location of the JAR file. It is not looked up from the repository. This may be useful to specify a dependency that is not present in the repository** |
| Import | **This is only used on a dependency of the pom type in the dependency Management section. It indicates that the specified pom should be replaced with the dependencies in that pom's dependency Management section. This is intended to centralize dependencies of large multi-module projects** |

**To explode the war file generated by maven use: mvn war: exploded  
to generate web app project from command line:   
mvn archetype: generate –DinteractiveMode=false -DgroupId=com.packt.cookbook   
-DartifactId=simple-webapp -DarchetypeArtifactId=maven-archetype-webapp**