Basics

`mvn <plugin>:<goal> [-Doption1 -Doption2 ... ]`	Basic maven invocation of <goal> (multiple goals possible)
`mvn <phase> [-Doption1 -Doption2 ... ]`	Execute all maven goals until phase <phase> (multiple phases possible)
`mvn -h`	Getting help on command line parameters
`mvn install`	Standard mvn call: build the package files (jar, war, etc), run unit and integration tests, submit files to local mvn repository (this call works by default recursively).
`mvn -o`	Working offline
`mvn -U`	Ignore earlier failures to download latest snapshots (sometimes useful when mvn does not find a snapshot that seems to be present)
`mvn -N`	Skip visiting of child artifacts (the given goal or phase will be executed only for the current artifact - no recursive descent)
`mvn -DtestFailureIgnore=true <goal>`	To continue the maven build even if a test fails
`mvn install -DskipTests=true`	To make maven install without launching the tests (but it generates the test-jar)
`mvn test`	To run unit tests
`mvn clean`	To clean up
`mvn -P<profile>`	Activate a profile
`mvnrec <plugin>:<goal>` or `<phase>`	Recursively execute the goal or the phase (like LEAF build.rec or EL4Ant? build.rec) (available from EL4J 1.3)

Important *phases* of maven (a concatenation of phases is called lifecycle)

- **Build** lifecycle: *validate, compile, test, package, integration-test, verify, install, deploy*
- **Clean** lifecycle: *clean*
- **Site** lifecycle: *site, site-deploy*

Configuration

Environment variable	Significance
M2_HOME	Root of where the `mvn` executable is located
MAVEN_OPTS	Command line options of the JVM launching maven

Environment variable	Significance
M2_REPO	Location of the local maven repository

Configuration file for maven: `~/.m2/settings.xml`.

We recommend to set the following parameter on the command line of maven: `-fae` "fail at end" Rationale: (1) you see explicitly what tasks have been run and (2) maven continues until the end (without breaking on intermediate errors) BTW: We added this parameter to the default maven launch script.

Eclipse Plugin

`mvn eclipse:eclipse`	To generate Eclipse project descriptor after configuring the dependencies in pom.xml (see also next point)
`mvn clean eclipse:eclipse -DdownloadSources=true`	To setup the source versions of all included libraries (for debugging and documentation convenience)
`mvn eclipse:configure-workspace`	To add the classpath variable M2_REPO to Eclipse

Sometimes the eclipse plugin does not download the sources. In that case execute `mvn eclipse:remove-cache` or look for a `mvn-eclipse-cache.properties` file in the target directory of the current folder **or one of its parent folders** and delete it! (often you can find it in `external/target` or `internal/target`)

Cargo Plugin (to deploy artifacts in a WEB or EJB container)

`mvn cargo:start`	To start the configured container. By default this is Tomcat 6x
`mvn cargo:deploy`	To deploy the configured deployable. If the current artifact is of type war this war will be deployed
`mvn cargo:undeploy`	To undeploy the configured deployable. If the current artifact is of type war this war will be undeployed
`mvn cargo:stop`	To stop the configured container. By default this is Tomcat 6x

How we recommend to use the cargo plugin

- Start the container (i.e. Tomcat) with `mvn cargo:start` in a separate bash terminal.
- In another bash terminal: Go to the topmost directory where you have applied changes (typically the `web` pom that has the `jar` and the `war` artifacts as its subdirs). Execute `mvn clean install cargo:undeploy cargo:deploy`
- For a redeployment it is mostly enough to execute `mvn install cargo:deploy`
  BTW, executing `mvn cargo:deploy` in a `jar` or `pom` artifact does not result in a failure.

Launch and init the database with the database plugin

`mvn db:prepare db:block`	Initializes and launches the db for the data of the currently active project (indicated via the current directory). It collects recursively the db scripts to launch for all projects this project depends on. **CAVEAT:** `db:prepare` alone does not block (even with db.wait flag)
`mvn db:start db:silentDrop db:create`	Does the same as `db:prepare` (in the currently active directory)
`mvn db:prepare cargo:undeploy cargo:deploy cargo:start`	Initialize the database, install the deployable in the web container and start the web container (cd to the correct `war` directory before launching this command)

Env support (from EL4J 1.6)

`mvn envsupport:list`	print out info about env configuration

Installing and deploying many libraries in repositories

Have a look at the repohelper plugin http://el4j.sourceforge.net/plugins/maven-repohelper-plugin/index.html BTW, install means update the local repository, deploy means update the local and the remote repository.

Site generation

`mvn site`	To create a complete documentation website containing Javadoc, Checkstyle, JUnit and other report pages for the current artifact. JUnit tests will be directly executed for the current artifact. The generated site can be found at "target/site"
`mvn -DskipTests=true clean site`	To generate site documentation without running the tests (handy while updating the APTs)
`mvn javadoc:javadoc`	To generates Javadoc into directory "target/site/apidocs"
`mvn checkstyle:checkstyle`	To check the style of the code by applying the EL4J's Checkstyle rules and generate reports at "target/site/checkstyle.html"

Varia

`mvn -Djetty.port=80 -Djee-web.context='/' db:prepare jetty:run`	deploy your application to the root path (means: deploy to `localhost`)
`mvn exec:java -Dexec.args="\"A single argument\""` [-Dexec.executable="maven"] [-Dexec.workingdir="/tmp"]	To execute the default java class of this module

`mvn dependency:tree`	Get a textual dependency graph.
`mvn depgraph:fullgraph -Ddepgraph.groupFilter="ch.elca"`	Get a dependency graph. **You must install the Graphviz application first!** See http://www.graphviz.org/
`mvn depgraph:fullgraph -Ddepgraph.groupFilter="(ch.elca.el4j.modules)|(ch.elca.el4j.demos)|(ch.elca.el4j.apps)" -Ddepgraph.filterEmptyArtifacts=true -Ddepgraph.dotFile=el4j.dot`	Another dependency graph
`mvn assembly:assembly -DdescriptorId=src`	To distribute the source code
`mvn install:install-file -Dfile=foo.jar -DgroupId=bar -Dversion=x.y -Dpackaging=jar -DartifactId=blah`	To install a jar file on local repo
`mvn install -DperformRelease=true`	Installs all artifacts of the project in the current directory in local repo (includes src, javadoc, tests)
`mvn assembly:assembly -DdescriptorId=jar-with-dependencies`	To generate a single jar files with all its dependencies
`mvn archetype:generate -DarchetypeGroupId=ch.elca.el4j.archetypes -DarchetypeArtifactId=EL4JCore -DarchetypeVersion=1.7 -DgroupId=ch.elca.el4j -DartifactId=myFirstProject -DremoteRepositories=http://el4.elca-services.ch/el4j/maven2repository`	To create a new EL4J project Note that you have to adapt the version number of the archetype
`mvn install:install-file -Dfile=foo.jar -DgroupId=org.foosoft -DartifactId=foo -Dversion=1.2.3 -Dpackaging=jar`	Installing a 3rdParty jar in local repository.
`cd $EL4J_ROOT/external ; mvn -N deploy:deploy-file -DgroupId=org.springframework -DartifactId=spring -Dversion=2.0.5 -Dpackaging=jar -Dfile=D:/tools/spring-framework-2.0.5/dist/spring.jar -DrepositoryId=ftpEl4ElcaServices -Durl=ftp://el4.elca-services.ch/htdocs/el4j/maven2repository`	Install 3rdParty jar to remote repository (requires pom.xml in . and requires login/password for repo (in your settings.xml)).
`cd $EL4J_ROOT/external ; mvn -N deploy:deploy-file -DgroupId=org.springframework -DartifactId=spring -Dversion=2.0.5 -Dpackaging=source -Dfile=D:/tools/spring-framework-2.0.5/dist/spring-src.zip -DrepositoryId=ftpEl4ElcaServices -Durl=ftp://el4.elca-services.ch/htdocs/el4j/maven2repository`	Install a 3rdParty source zip to remote repository.
`cd $EL4J_ROOT/external ; mvn -N deploy:deploy-file -DpomFile=D:/Projects/dashboard/dashboard-maven-plugin/pom.xml -Dfile=D:/Projects/dashboard/dashboard-maven-plugin/target/dashboard-maven-plugin-1.0.0-beta-1-el4j_20081022_0930.jar -DrepositoryId=ftpEl4ElcaServices -Durl=ftp://el4.elca-services.ch/htdocs/el4j/maven2repository`	Install a 3rdParty plugin to remote repository (requires pom.xml in . and requires login/password for repo (in your settings.xml)).

Debugging mvn pom files, settings and plugins

Please be aware that there is typically more than one JVM involved when maven executes your project. The hints here only apply for debugging the "first" JVM (i.e. the one maven

runs in). Please refer to DebuggingHowTo for further hints on debugging the other JVMs!

`mvn -X`	Enable debug output
`mvn -N help:effective-settings`	Prints the currently effective maven settings on the console
`mvn -N help:effective-pom`	Prints the effective pom on console (merges all pom sections that currently apply)
`mvn -N help:active-profiles (-P.. -D..)`	To test what profiles of the current artifact are currently active. In addition you can set profiles (-P) or system properties (-D) on the command line to see what profiles would be active in that case.
`mvn project-info-reports:dependencies`	Makes a report on dependencies. Then refer to target/site/dependencies.html
`mvn -N help:describe -DgroupId... -DartifactId... -Dfull=true`	Describes all goals of the given plugin (groupId & artifactId).
`mvn -N help:describe -Dplugin=repohelper -Dmojo=deploy-libraries -Dfull=true`	Instead of groupId & artifactId you can use the parameter plugin with format groupId:artifactId and you can even use the plugin prefix.
set `MAVEN_OPTS="-Xmx768M -XX:MaxPermSize=512M -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000"`	To be able to debug a running maven with e.g. eclipse (you need then to connect to the JVM from eclipse)

Integration test practices and skipping

To allow faster local builds of el4j, we put all integration tests that run in a default build (mvn install in the root folder) in a profile named `integrationTests` and make it `activeByDefault`. This includes the actual tests (phase integration-test), and their preparation and cleanup (pre-integration-test and post- respectively.) We should now be able to use

- `mvn -DskipTests=true -P-integrationTests`

to skip all tests and integration tests. Note, however,

- skipTests by itself skips all unit tests that run in the test phase.
- Deactivating the integrationTests profile without skipping tests would cause maven to run the tests (without preparation or cleanup) in the test phase. This is a recipe for failure.
- It is possible to replace skipTests by maven.test.skip which also skips compiling the tests. Although this is faster, any dependencies that require the test-jars may fail if you do this.
- Due to a bug in maven 2.0.9, deactivating profiles does not work by default. A patch is available.

Activating and Deactivating Profiles from the Command Line

We have patched maven to fix one bug and make one change to the profiles syntax. Here is what the options do:

Option	Normal Maven	Our Patch

Option	Normal Maven	Our Patch
`mvn -Pprofile`	Activate profile `profile` and deactivate all *activeByDefault* profiles.	Like normal maven.
`mvn -P-profile`	Does not work in 2.0.9 due to a bug. (It is documented to work like in our patched version.)	Deactivates profile `profile`. It will not be executed even if it was *activeByDefault*.
`mvn -P+profile`	Like without the `+`, activates profile `profile` and deactivates defaults.	Activates profile `profile` but does not deactivate the defaults.

Overriding Resource Filtering

Maven's default order for applying properties is :

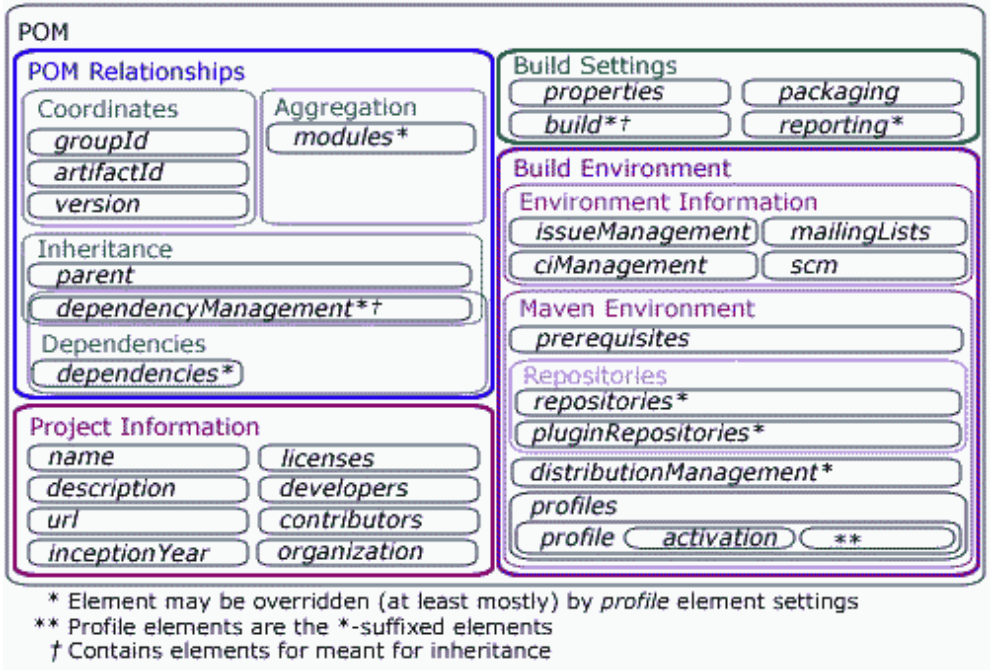
1. Properties in the environment or system properties.
2. Command line properties.
3. Properties in settings.xml.
4. Properties in the pom file.

Later ones override earlier ones. This means in particular that if you set a property on the command line (`-Dfoo=bar`) and the property is already set in your pom file, the pom one and not your command line one takes effect.

For the special case of copying resource files with filtering - this applies to both maven-resources-plugin and env-plugin - we have made patches that define the following additional syntax: Setting a property with the prefix "override." on the command line overrides settings.xml and pom.xml properties. For example, `mvn -Doverride.username=elcauser install` will use the property "username=elcauser" during resource filtering even if it is set otherwise in your pom. (This overriding does not work for the variables within other parts of maven.)

This requires our patched version of the maven-resources-plugin.

Overview of pom-file structure



References

- Extensive maven presentation: http://el4j.sourceforge.net/docs/pdf/Maven2Course_v1_2.pdf
- Documentation of standard Maven 2 plugins: http://maven.apache.org/plugins/
- Documentation of Maven 2 plugins at codehaus (a bit out of date): http://docs.codehaus.org/display/MAVEN/Maven+Plugin+Matrix
- Maven book: http://www.mergere.com/m2book_download.jsp
- EL4J: http://el4j.sourceforge.net/
- PluginDatabase
- PluginDepGraph

History

- In EL4J 1.1.3, db:prepareDB was replaced by db:prepare and db:cleanUpDB was replaced by db:cleanUp