1. Maven gère un localrepository là où il s'exécute. Il y recopie tout ce qu'il télécharge. le chemin (à configurer à l’instatllation) se trouve dans le fichier settings.xml, balise : localRepository
2. [Comment utiliser un proxy ?](http://java.developpez.com/faq/maven/?page=install_config#configuration1) Dans le fichier **settings.xml**, situé dans *%MVN\_HOME%/conf* ou *%USER\_HOME%/.m2/settings.xml* :

<settings>

<proxies>

<proxy>

<active>true</active>

<protocol>http</protocol>

<host>proxy.somewhere.com</host>

<port>8080</port>

<username>proxyuser</username>

<password>somepassword</password>

<nonProxyHosts>www.google.com|\*.somewhere.com</nonProxyHosts>

</proxy>

</proxies>

</settings>

1. Définition d'un artifact : Tous les artifacts gérés par Maven sont identifiés par 4 informations :

* **groupId** : Définit l'organisation ou groupe qui est à l'origine du projet. Il est formulé sous la forme d'un package Java (org.archetypejs par exemple)
* **artifactId** : Définit le nom unique dans le groupe pour le projet.
* **version** : Définit la version du projet. Les numéros de version sont souvent utilisés pour des comparaisons et des mises à jour.
* **type** : Type de projet. On trouvera le plus souvent : jar, war, maven-plugin, pom...

1. créer un projet simple : mvn archetype:create -DgroupId=com.objis.mavenapps -DartifactId=monappli

Comment créer un projet web :

mvn archetype:create -DgroupId=fr.monGroupId -DartifactId=MonApplicationWeb

-Dpackagename=fr.monGroupId -DarchetypeArtifactId=maven-archetype-webapp (pour obtenir la liste des types de projets qu’on peut créer, exécuter :  **mvn archetype:generate). Pour voir la liste complete :** <http://myjeeva.com/exclusive-maven-archetype-list.html>

1. pour transformer le projet en projet reconnaissable par eclipse : mvn eclipse:eclipse
2. $mvn compile : compile le projet
3. $mvn test-compile: Compile les tests unitaires du projet
4. $mvn test: Exécute les tests unitaires (typiquement avec Junit) dans le répertoire src/test
5. $mvn package: Mets en forme le code compilé dans son format de diffusion (JAR, WAR, etc.). En utilisant la notion de profiles, quand on veut faire une livraison en prod, on package : mvn package -Pprod. En sortie, vous aurez toujours votre common.properties, mais par contre vous aurez le specific provenant de src/main/resources\_prod.
6. $mvn integration-test: Réalise et déploie le package si nécessaire dans un environnement dans lequel les tests d'intégration peuvent être effectués.
7. $mvn install: Installe les produits dans l'entrepôt local, pour être utilisé comme dépendance des autres projets sur votre machine locale.
8. $mvn deploy: Réalisé dans un environnement d'intégration ou de production, copie le produit final dans un entrepôt distant pour être partagé avec d'autres développeurs ou projets.
9. mvn versions:set : permet d’incrémenter la version du projet.
10. On peut activer les profils souhaités en utilisant l’option -P du binaire mvn : mvn clean package -P monProfileTest,monProfilDev,monProfilReportingVerbeux
11. Utiliser les properties. Exemples

<properties>

<logback.version>0.9.15</logback.version>

<spring.version>2.5.6</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version**>${logback.version}<**/version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version**>${spring.version}**</version>

</dependency>

</dependencies>

1. Erreurs
   1. static import declarations are not supported in source 1.3 maven. Ajouter ça dans build

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

* 1. Package org.junit does not exist. Ajouter ça dans build

<dependency>

<groupId>junit</groupId>

<artifactId>junit-dep</artifactId>

<version>4.8.2</version>

</dependency>

1. Livrer des fichier ressources propres à chaque environnement. La technique consiste à créer des profile pour chaque environnement. Et préciser les actions à dans chaque environnement, exemple :

<profiles>

<profile>

<id>test</id>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>Using env.test.properties</echo>

<delete file=*"${project.build.outputDirectory}/env.properties"*/>

<copy file=*"resources/env.test.properties"* tofile=*"${project.build.outputDirectory}/env.properties"*/>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</profile>

<profile>

<id>local</id>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>Using env.properties</echo>

<delete file=*"${project.build.outputDirectory}/env.properties"*/>

<copy file=*"resources/env.properties"* tofile=*"${project.build.outputDirectory}/env.properties"*/>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</profile>

</profiles>

1. Executer chaque environment en le précisant dans l’option –P. Exemple
   1. mvn test –Plocal.
   2. mvn test –Ptest.