

Maven : Outils gestion de construction de projet logiciels

Une introduction

Pascal Fares © Cnam Liban



Maven c'est quoi?

What is Maven?

- Outil de gestion de projet logiciel
 - C'est plus qu'un outil de construction
- basé "Project Object Model" (POM) - pom.xml
 - pom.xml gère la construction du projet,
 - les rapports et la documentation, POM peut être hérité entre un projet parent et un projet enfant
- Basé sur le principe "Convention over configuration"
 - Un minimum de configuration au profit de conventions
- Vous spécifiez "ce qui doit être fait"
 - Pas "comment ça doit être fait"
- Architecture de plug-in
 - Éco-système Maven

Objectifs

- **Rendre le processus de construction facile**
 - Pourquoi devons-nous perdre autant de temps à entretenir la construction?
- **Fournir un système de construction uniforme**
 - Pourquoi chaque développeur doit-il maintenir son propre environnement de construction?
- **Fournir des informations de qualité sur le projet**
 - Pourquoi devons-nous faire un travail supplémentaire pour obtenir des informations sur le projet?
- **Fournir des lignes directrices pour le développement et de meilleures pratiques**
 - Comment pouvons-nous saisir les meilleures pratiques?
- **Permettre une migration transparente vers de nouvelles fonctionnalités**

Fournir des informations de qualité sur le projet

- Maven fournit de nombreuses informations utiles sur les projets partie prise à partir de votre POM et en partie généré à partir de votre sources du projet.
 - log créé directement à partir du contrôle du code source
 - Sources croisées (cross-reference)
 - Listes de diffusion
 - Liste de dépendances
 - Rapports d'essai unitaires, y compris la couverture
 - Beaucoup plus (via des plug-ins)

Caractéristiques

- Configuration de projet simple qui suit les meilleures pratiques - obtenir un nouveau projet ou module démarré en quelques secondes
- Une utilisation cohérente dans tous les projets signifie pas de perte de temps pour les nouveaux développeurs à venir sur un projet
- Gestion de la dépendance mise à jour automatique et dépendances transitives
- Capable de travailler facilement avec plusieurs projets en même temps

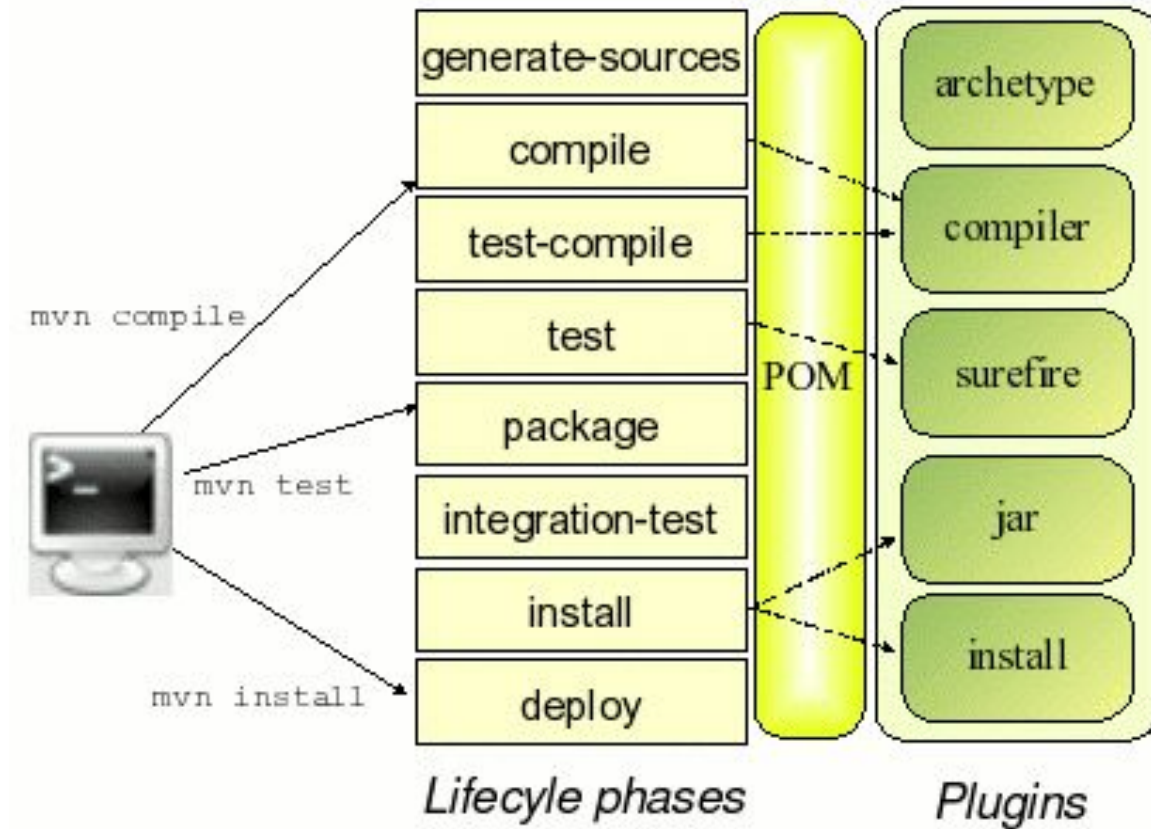
Caractéristiques

- Un référentiel grand et croissant de bibliothèques et de métadonnées à utiliser directement en référençant directement les plus grands projets Open Source pour la disponibilité en temps réel de leurs dernières versions
- Extensible, avec la possibilité d'écrire facilement des plugins en Java ou en langages de script
- Accès instantané aux nouvelles fonctionnalités avec peu ou pas d'extra configuration

Cycle de vie projet

Les phases

Les plugins



Installation de maven

Installation

Récupérer la dernière versions

<http://maven.apache.org/download.cgi>

Puis l'installer <http://maven.apache.org/install.html>

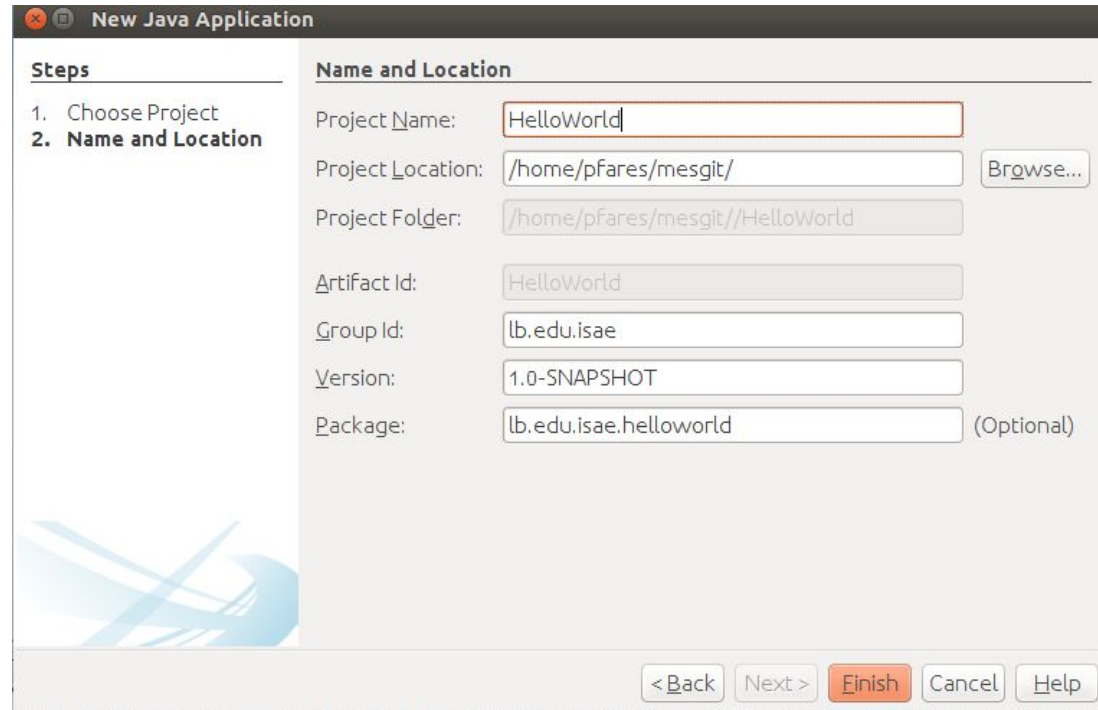
Premier projet en
ligne de commande

Étapes (vous aurez besoin d'une connexion réseau)

Vérifier que votre PATH
et JAVA_HOME sont
correcte

Faire mvn
archetype:generate

Vous devrez répondre à
certaines questions (dans
Netbeans elle
correspondent à la figure
==>

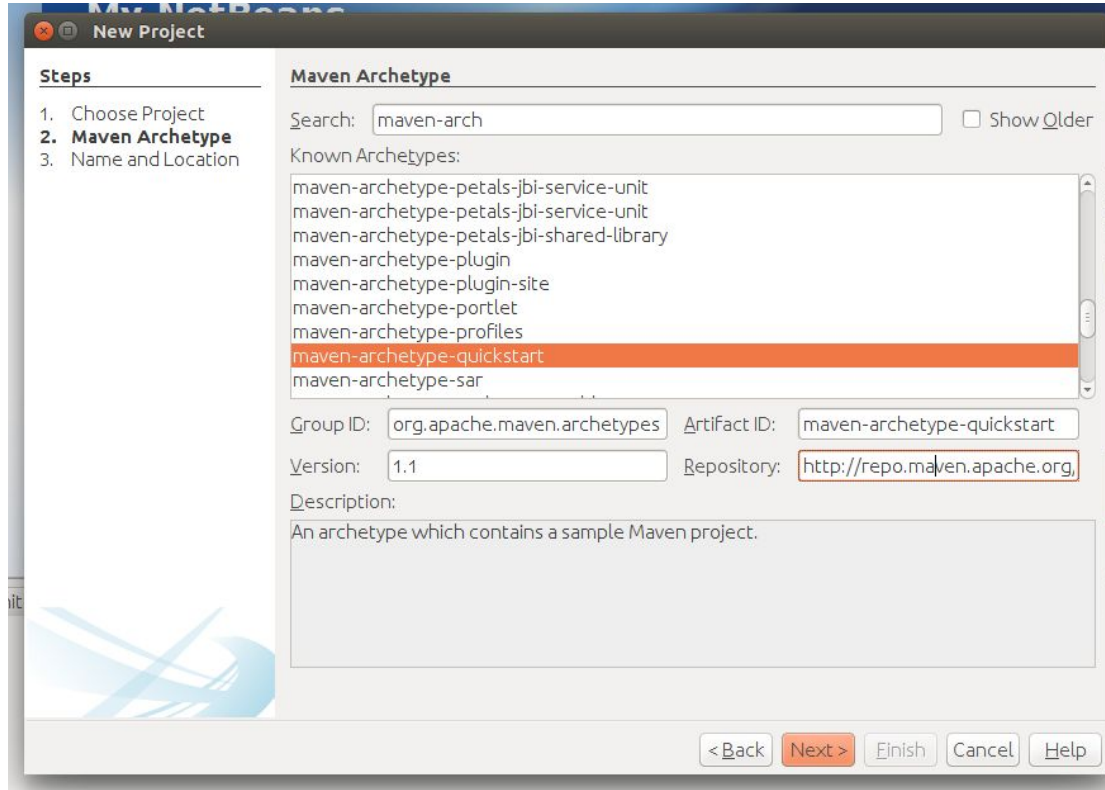


The screenshot shows the 'New Java Application' dialog box in NetBeans. The 'Steps' panel on the left indicates that step 2, 'Name and Location', is the current step. The 'Name and Location' panel contains the following fields:

- Project Name:** HelloWorld
- Project Location:** /home/pfares/mesgit/ (with a 'Browse...' button)
- Project Folder:** /home/pfares/mesgit//HelloWorld
- Artifact Id:** HelloWorld
- Group Id:** lb.edu.isae
- Version:** 1.0-SNAPSHOT
- Package:** lb.edu.isae.helloworld (Optional)

At the bottom right, there are navigation buttons: '< Back', 'Next >', 'Finish' (highlighted in orange), 'Cancel', and 'Help'.

Choisir un archetype (une sorte de canevas pour la structure de nos projets logiciel)



maven-archetype-quickstart

Puis les autres paramètres

New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: HelloWorld

Project Location: /home/pfares/mesgit/ Browse...

Project Folder: /home/pfares/mesgit//HelloWorld

Artifact Id: HelloWorld

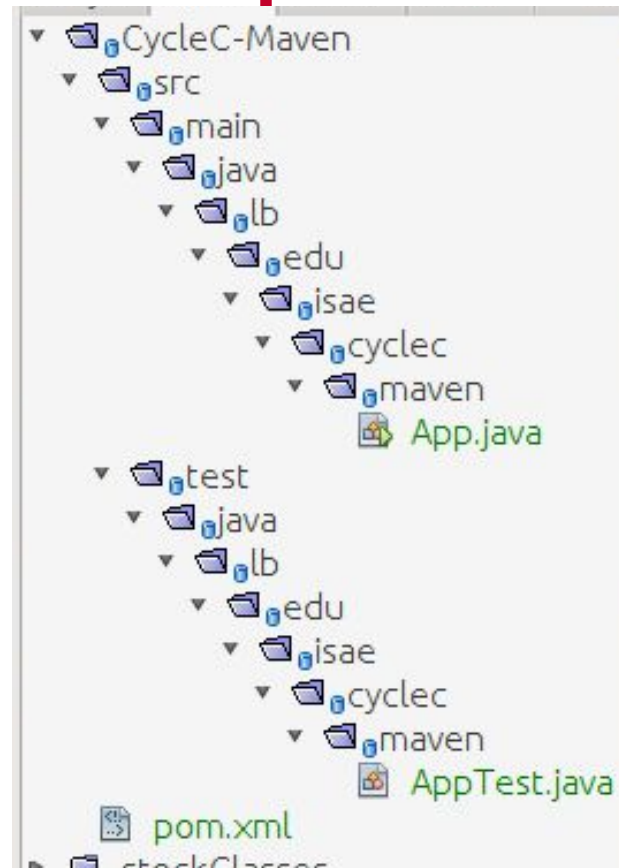
Group Id: lb.edu.isae

Version: 1.0-SNAPSHOT

Package: lb.edu.isae.helloworld (Optional)

< Back Next > **Finish** Cancel Help

Ce qui donnera ceci : un pom.xml et une structures de répertoires



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>lb.edu.isae</groupId>
  <artifactId>CycleC-Maven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>CycleC-Maven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Sens de quelques balises

<project> La balise racine du fichier pom.xml, elle contient des paramètres pour valider le contenu XML

<modelVersion> La version du POM (Project Object Model)

<groupId> L'identifiant de la personne, équipe, ou entreprise qui développe le projet.

<artifactId> Identifiant du projet (c'est aussi nom du projet au niveau du dossier racine).

<version> La version du projet.

<packaging> Représente le format de sortie (généralement : « jar » pour les applications JSE, « war » pour les applications JEE, « ear » pour les projets plus complexes). Si la balise n'est pas indiquée la valeur par défaut est « jar »

<name> Nom du projet

<url> Site web du projet

<description> Description du projet

<dependencies> Contient toutes les dépendances (les JAR) utilisées dans le projet

Identité unique pour chaque projet logiciel

Tout dans Maven est un projet et chaque projet a une identité unique, l'identité du projet est spécifiée par les "coordonnées" projet

- Considérez cela comme une adresse pour un point spécifique dans "l'espace"
- Identifie de manière unique un projet dans des référentiels de projets (publique ou privé)

Les dépendances et les références parentes sont décrites avec leurs propres coordonnées du projet

Créé avec la combinaison des éléments suivants

```
<groupId> com.javapassion.examples </ groupId>  
<artefact> helloworld_app </ artefact>  
<version> 1.0-SNAPSHOT </ version>
```

Quelques convention et précisions : identité

- **groupid**
 - > Représente généralement une organisation
 - > La convention utilise un nom de domaine inversé
 - > Exemple: lb.edu.isae
- **artifactId**
 - Un identifiant unique sous groupId
- **version**
 - Une version spécifique d'un projet

L'id projet unique est donc "groupid:artifactId:version" ou "groupid-artifactId-version"

Archétype (Archetype)

d'après le dictionnaire "Modèle original ou idéal sur lequel est fait un ouvrage, une œuvre. Manuscrit ancien qui est, par son texte, l'ancêtre d'un ou de plusieurs autres. Chez Platon, prototype des réalités visibles du monde. "

Archetype pour Maven c'est quoi?

- "Archétype" est "un modèle ou un type original après lequel des choses similaires peuvent être modelées ou prototypées "
 - Peut être pensé comme un modèle de projet logiciel
- • Capture les meilleures pratiques
 - Structure du répertoire, dépendances, plugins nécessaires
- Il y a beaucoup d'archétypes déjà fournis par la communauté Maven Application
 - Java SE simple
 - Application Spring
 - Application Hibernate
 - Application JSF
 - Application Java EE 6
 - Beaucoup plus

Configuration
utilisateur spécifique
et dépôt local

Configuration spécifique à l'utilisateur et Référentiel local

- `<répertoire_d'accueil> /. M2 / settings.xml`
 - Contient une configuration spécifique à l'utilisateur pour l'authentification, référentiels, et d'autres informations pour personnaliser le comportement de Maven
- `<répertoire_d'accueil> /. M2 / référentiel`
 - Référentiel Maven local
 - Stocke les artefacts générés localement (fichiers jar, fichiers war, etc.)
 - Stocke les copies des dépendances téléchargées à partir des référentiels distant

Exemple de contenu de ~/.m2/repository

```
pfares@cnamBackup: ~/.m2/repository
pfares@cnamBackup: ~
pfares@cnamBackup: ~
^C
pfares@cnamBackup:~/Documents$ pkill obs
pfares@cnamBackup:~/Documents$ ps aux | grep obs
pfares  11242  0.0  0.0  14352  984 pts/1    S+   08:08   0:00 grep --color=au
to obs
pfares@cnamBackup:~/Documents$ cd
pfares@cnamBackup:~$ cd .m2/
pfares@cnamBackup:~/.m2$ ls
repository
pfares@cnamBackup:~/.m2$ cd repository/
pfares@cnamBackup:~/.m2/repository$ pwd
/home/pfares/.m2/repository
pfares@cnamBackup:~/.m2/repository$ ls
antlr  com          commons-collections  commons-lang  eu    junit  org
asm    commons-codec  commons-io           dom4j         jdom  net    xml-apis
pfares@cnamBackup:~/.m2/repository$ ls junit/
junit
pfares@cnamBackup:~/.m2/repository$ ls junit/junit/3.8.1/
junit-3.8.1.jar      junit-3.8.1.pom      _maven.repositories
junit-3.8.1.jar.sha1  junit-3.8.1.pom.sha1
pfares@cnamBackup:~/.m2/repository$
```

Phase et cycle de vie

Bases du cycle de vie de construction

- • Maven est basé sur le concept central d'un cycle de vie de construction
 - Ce que cela signifie est que le processus de construction et de distribution d'un artefact particulier (projet) est clairement défini.
 - Pour la personne qui construit un projet, cela signifie que c'est seulement nécessaire d'apprendre un petit ensemble de commandes pour construire un projet "maven", et le POM s'assurera d'obtenir les résultats souhaités.
- • Il existe trois cycles de vie de construction intégrés:
 - default - gère la construction / le test / le déploiement de votre projet
 - clean - gère le nettoyage du projet
 - site - gère la création de la documentation du site de votre projet.

Phases du cycle de vie des builds

Chacun de ces cycles de vie est défini par une liste différente de phases de construction, où une phase de construction représente une étape dans le cycle de vie.

Cycle de vie "Default"

- **valider** - valider le projet est correct et toutes les informations nécessaires sont disponible
- **compile** - compile le code source du projet
- **test** - testez le code source compilé à l'aide d'un cadre de test unitaire approprié. Ces tests ne devraient pas exiger que le code soit emballé ou déployé
- **package** - récupère le code compilé et l'empaquette dans son format distribuable, comme un fichier JAR ou un fichier WAR

- **integration-test** - traiter et déployer le paquet si nécessaire dans un environnement dans lequel les tests d'intégration peuvent être exécutés
- **vérifier** - exécuter toutes les vérifications pour vérifier que le paquet est valide et répond au critères qualité
- **install** - installe le paquet dans le dépôt local, pour l'utiliser comme une dépendance dans d'autres projets localement
- **deploy** - effectué dans un environnement d'intégration ou de publication, copie le package final vers le référentiel distant pour partager avec d'autres développeurs et projets.

Build "Default"

- Ces phases de construction sont exécutées séquentiellement pour compléter le cycle de vie
 - • mvn compile
 - Toutes les phases jusqu'à la phase de compilation seront exécutées en séquence
 - • mvn test
 - Toutes les phases jusqu'à la phase de test seront exécutées en séquence
 - • mvn install
 - Toutes les phases jusqu'à la phase d'installation seront exécutées dans l'ordre
 - • mvn integration-test
 - Toutes les phases jusqu'à la phase de test d'intégration (valider, compiler, paquet, etc.) sera exécuté en séquence

Un exemple

\$ mvn --version

\$ mvn archetype:generate -DgroupId=lb.edu.isae

-DartifactId=app1

-DarchetypeArtifactId=maven-archetype-quickstart

-DinteractiveMode=false

```
AppTest.java
pfares@cnamBackup: ~/app1$ ls
pom.xml  src
pfares@cnamBackup: ~/app1$ ls src
main  test
pfares@cnamBackup: ~/app1$ ls src/main/java/lb/edu/isae/
App.java
pfares@cnamBackup: ~/app1$ ls src/test/java/lb/edu/isae/
AppTest.java
pfares@cnamBackup: ~/app1$
```

mvn package

Après un certain nombre de download (surtout la première fois)

on obtient (résultat sans les download)

```
pfares@cnamBackup: ~/app1
pfares@cnamBackup: ~
pfares@cnamBackup: ~/app1$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building app1 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ app1 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/pfares/app1/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ app1 ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /home/pfares/app1/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ app1 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/pfares/app1/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ app1 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ app1 ---
[INFO] Surefire report directory: /home/pfares/app1/target/surefire-reports

-----
T E S T S
-----
Running lb.edu.isae.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ app1 ---
[INFO] Building jar: /home/pfares/app1/target/app1-1.0-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 1.142 s
[INFO] Finished at: 2017-11-08T08:48:10+02:00
[INFO] Final Memory: 18M/299M
[INFO] -----
pfares@cnamBackup: ~/app1$
```

Phases du cycle de vie «clean»

- **pre-clean** - exécute les processus nécessaires avant le nettoyage du projet
- **clean** - supprime tous les fichiers générés par la version précédente
- **post-clean** - exécute les processus nécessaires pour finaliser le nettoyage du projet

Phases et but

- Les "plugin buts" peuvent être attachés à une phase de cycle de vie
- Comme Maven parcourt les phases dans un cycle de vie, il exécute les buts attachés à chaque phase particulière
 - Chaque phase peut avoir zéro ou plusieurs buts (de divers plugins) attachés à la phase

Lab 1 :

<https://github.com/ljulg/java-tutorials/blob/master/JavaTools/MavenSimple/README.md>

Le dépôt (référenciel)

Maven

Qu'est-ce qu'un référentiel ('repository' ou parfois j'appellerais depot)?

- Le référentiel entrepose les plugins et les artefacts
 - Les plugins et les artefacts sont extraits du dépôt distant selon les besoins
- Les référentiels distants par défaut gèrent les plugins publics et artefacts
 - <http://repo1.maven.org/maven2>
 - Appelé "Maven Central"
- Les référentiels personnalisés peuvent être configurés pour maintenir les plugins et artefacts non publique
 - Les référentiels distants par défaut peuvent être remplacés ou augmentés avec des références aux référentiels personnalisés

Structure du référentiel

- Chaque artefact est maintenu dans une structure de répertoire correspond à une coordonnée du projet
 - `/<groupId>/<artifactId>/<version>/<artifactId>-<version>.<packaging>`
- Exemple
 - "Org.apache.commons: commons-email: 1.1" (artefact)
 - `"/org/apache/commons/commons-email/1.1/commons-email-1.1.jar "` (chemin du répertoire)
 - `junit: junit: 3.8.1` est disponible sous `/junit/junit/3.8.1/junit-3.8.1.jar`
- Maven peut facilement localiser l'artefact dans un référentiel (local et distant) basé sur les coordonnées de l'artefact

Maven central

- <http://central.sonatype.org/>
- <https://search.maven.org/>

Sécurisé | <https://search.maven.org/#search%7Cga%7C1%7Cjunit>

Control Panel reseller Eurabiahosting oslm DSI ISSAE Enseignement Informations Rechercher Open Source Le Linux France S'initier au Ze

The Central Repository

SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS

[About Central](#) [Advanced Search](#) [API Guide](#) [Help](#)

All Day DevOps 2017 October 24th 100 Sessions Free Online

[All Day DevOps - Register Now!](#)

Search Results

< 1 2 3 ... 39 40 > displaying 1 to 20 of 784

GroupId	ArtifactId	Latest Version	Updated	Download
org.roboelectric	junit	3.5.1 all (9)	28-Oct-2017	pom jar javadoc.jar sources.jar
com.airbnb.okreplay	junit	1.3.0 all (3)	26-Jul-2017	pom jar javadoc.jar sources.jar
com.buschmais.jqassistant.plugin	junit	1.3 all (3)	10-May-2017	pom jar asciidoc.zip javadoc.jar sources.jar
com.airbnb.walkman	junit	1.0.3	17-Mar-2017	pom jar javadoc.jar sources.jar
com.fitbur.testifyjunit	junit	0.1.3 all (13)	18-Feb-2016	pom
org.typelevel	junit	2.11.7	20-Sep-2015	pom jar javadoc.jar sources.jar
com.github.adedayo.intelliJ.sdk	junit	142.1	13-Jul-2015	pom jar javadoc.jar sources.jar
junit	junit	4.12 all (24)	04-Dec-2014	pom jar javadoc.jar sources.jar
org.technbolts	junit	1.0.1	15-May-2014	pom jar javadoc.jar sources.jar
com.groupon.roboreMOTE.roboreMOTEclient	junit	0.4 all (3)	08-Jan-2013	pom jar javadoc.jar sources.jar
org.apache.tuscany.sca.samples	junit	2.0	19-Jun-2012	pom jar sources.jar
org.apache.isis.viewer	junit	0.2.0-incubating all (2)	10-Feb-2012	pom jar javadoc.jar site.xml sources.jar test-sources.jar tests.jar
com.kenai.nbpwr	junit	4.7-201002241900	26-Feb-2010	pom jar nbm tests.jar
org.mod4j.org	junit	3.8.2	14-Aug-2009	pom jar
org.codehaus.mevenide	junit	3.1.4 all (5)	05-Aug-2008	pom jar nbm javadoc.jar sources.jar
org.eclipse.jdt	junit	3.3.0-v20070606-0010 all (2)	27-Nov-2007	pom jar
junit	junit-dep	4.11 all (11)	14-Nov-2012	pom
org.testifyproject.junit4	junit4	0.9.9 all (9)	15-Sep-2017	pom
io.tourniquet.junit	tourniquet-junit	0.4.8 all (10)	14-Aug-2016	pom
junit-addons	junit-addons	1.4	11-Mar-2006	pom jar

< 1 2 3 ... 39 40 > displaying 1 to 20 of 784

Plugin et buts

Architecture a base de plugins

- Maven est basé sur l'architecture Plugin
 - Toutes les tâches Maven sont effectuées via des plugins
- • Le noyau de Maven est fondamentalement une coquille
 - Il analyse un fichier POM et détermine quels plugins sont nécessaires et puis le télécharge
- • Les plugins sont téléchargés, comme les dépendances sont téléchargés, à partir de dépôts distants selon les besoins et mis à jour périodiquement
 - Un plugin est un projet Maven et possède sa propre identité (coordonnées)
 - Un plugin téléchargé est ensuite maintenu dans le référentiel local

Donc un plugin c'est quoi?

- Un plugin Maven est une collection d'un ou plusieurs buts
- • Exemples de plugins "prêts à l'emploi"
 - Archétype plugin - contient des objectifs pour la création de projets Maven
 - Jar plugin - contient des objectifs pour la création de fichiers JAR
 - Plugin du compilateur - contient des objectifs pour compiler le code source et tests unitaires
 - Hibernate3 plugin - contient des objectifs pour l'intégration avec le Bibliothèque Hibernate
 - JRuby plugin - contient des objectifs pour l'exécution de JRuby dans le cadre d'une construction Maven
 - ...

Plugin personnel (customisé)

On peut créer ses propres plugins dans plusieurs langages de votre choix (un plugins est lui même un logiciel)

Java, Groovy, Ant, Ruby, etc...

Avantages de l'architecture de plugin

- Plugin commun utilisé par tout le monde à chaque projet
 - Tout le monde comprend ce que le plugin fait - pas besoin de réapprendre
- Le plugin peut évoluer / améliorer sans casser d'autres parties de la construction
- Modification / amélioration d'un plugin (par la communauté) bénéficie à tout le monde

Et un but Maven c'est quoi?

- Le but est une unité de tâche
 - > Identique à "target" dans Ant ou Makefile
- Exemples d'objectifs
 - > Objectif "generate" du plugin "archetype"
 - > "Compile" du plugin "compiler"
 - > Objectif "test" du plugin "surefire"

Un plugins peut être configuré

A partir de propriétés dans le XML du POM

Exemple changer la version du compilateur

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Ou par exemple faire un jar executable java -jar ...

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <mainClass>isae.App
            </mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Gestion des dépendances

Comment décrire les dépendances entre projets?

Les coordonnées projet permettent de référencer les dépendances

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>lb.edu.isae</groupId>
  <artifactId>app1</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>app1</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```


Dépendances transitive comme Makefile ou Ant

Votre projet dépend d'une bibliothèque A, La bibliothèque A dépend de 5 autres bibliothèques - B, C, D, E, F

- Votre projet doit spécifier une dépendance uniquement sur A

Maven va gérer le fait que A dépend de B, C, D, E, F

- Maven gère également le conflit entre les dépendances
- Vous pouvez voir l'arborescence des dépendances via la commande Maven

> mvn dependency:tree

Chaque dépendance a une portée (scope)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>lb.edu.isae</groupId>
  <artifactId>app1</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>app1</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Exemple de porté (scope) de dépendance

- Lorsqu'une dépendance a une portée de "test", elle ne sera pas disponible pour l'objectif "compile" du plugin Compiler
- Il sera ajouté au classpath pour seulement Les buts "compiler: testCompile" et "surefire: test"

Les portés (scope) de dépendances

- **compile**
 - > Portée par défaut, utilisée si aucune n'est spécifiée.
- **provided**
 - Comme pour "compile", mais indique que vous attendez de la JDK ou un conteneur pour fournir la dépendance à l'exécution.
 - Par exemple, lors de la création d'une application Web, vous devez définir la dépendance à l'API Servlet et aux API Java EE associées pour pouvoir compiler mais inutile dans le package car fourni par le conteneur Web.
 - Cette portée n'est disponible que lors de la compilation et de test et n'est pas transitive.
- **runtime**
 - La dépendance n'est pas requise pour la compilation, mais pour l'exécution. Il est dans les classes d'exécution et de test, mais pas pour le classpath lors de la compilation.

- test

- La dépendance n'est pas requise pour l'utilisation normale de l'application, et est uniquement disponible pour les phases de compilation et d'exécution du test.

- system

- Similaire à "provided" sauf que vous devez fournir le JAR qui le contient explicitement. L'artefact est toujours disponible et n'est pas recherché dans un dépôt.

- import (disponible uniquement dans Maven 2.0.9 ou version ultérieure)

- Seulement utilisé sur une dépendance de type pom dans la section <dependencyManagement>

Lors du packaging

- Lorsque vous créez un fichier JAR pour un projet
 - > Les dépendances ne sont pas regroupées avec l'artefact généré - elles sont utilisés uniquement pour la compilation
- Lorsque vous créez un fichier WAR / EAR
 - > Vous pouvez configurer POM pour que les dépendances soient regroupées avec l'artefact généré
 - > Vous pouvez également configurer pour exclure certaines dépendances en utilisant Portée "provided"

Maven et les EDI (IDE)

le support a maven

- Chaque IDE utilise ses propres métadonnées de projet d'une manière propriétaires
 - Résultats : pas facile de migrer les projets d'un EDI vers un autre
- Métadonnées de projet standardisées Maven
 - Le développeur peut utiliser n'importe quel IDE de son choix sur n'importe quel projet Maven
- Tous les principaux IDE (Eclipse, NetBeans, IntelliJ IDEA, etc) peuvent gérer des projet en Maven
 - Créer un projet Maven, Importer un projet Maven
 - Extras: éditeur POM basé sur un formulaire
- • Intégration étroite avec d'autres outils de construction IDE
 - > Gestion de version
 - > Gestion des tâches

Un exemple projetWeb Maven et netbeans

