1 - INTRODUCCIÓN MAVEN

OBJETIVOS

El asistente al curso será capaz de utilizar Maven como herramienta para la construcción de una aplicación, automatizar tareas, gestionar dependencias, etc.

TEMARIO

- CONCEPTOS BÁSICOS
- INSTALACIÓN Y CONFIGURACIÓN
- ARQUTECTURA Y CARACTERÍSTICAS
- ARTIFACT
- POM
- ARQUETIPOS
- COMANDOS Y CICLOS DE VIDA
- FASES, GOALS Y PLUGINS
- DEPENDECIAS

TEMARIO

- PROYECTOS MULTIMODULO
- REPOSITORIOS
- REPOSITORIO CORPORATIVO
- NEXUS
- PLUGINS BÁSICOS
- PLUGINS PERSONALIZADOS

MAVEN

Maven (<<uno que entiende>> en Hebreo) fue concebido originalmente como herramienta interna para ejecutar las compilaciones de Jakarta (proyecto de Apache)

Problema: muchos proyectos, cada uno con sus ficheros de compilación: Muchos Jar's dispersos en distintos repositorios

MAVEN: Objetivos

1- Definir claramente un proyecto

2- Proponer una manera estándar de compilar un proyecto

3- Establecer un mecanismo sencillo para compartir bibliotecas JAR a lo largo de diferentes proyectos

4- Dar soporte a buenas prácticas (TEST)

MAVEN: Objetivos

5- Mecanismo de soporte transparente, facilitando el uso de nuevas versiones de Maven sin apenas intervención del usuario final.

INSTALACIÓN

DESCARGA

https://maven.apache.org/download.cgi

VERSION

3.3.9

REQUISITOS

JDK 1.7 (mín)

INSTALACIÓN DEBIAN

LINUX DEBIAN

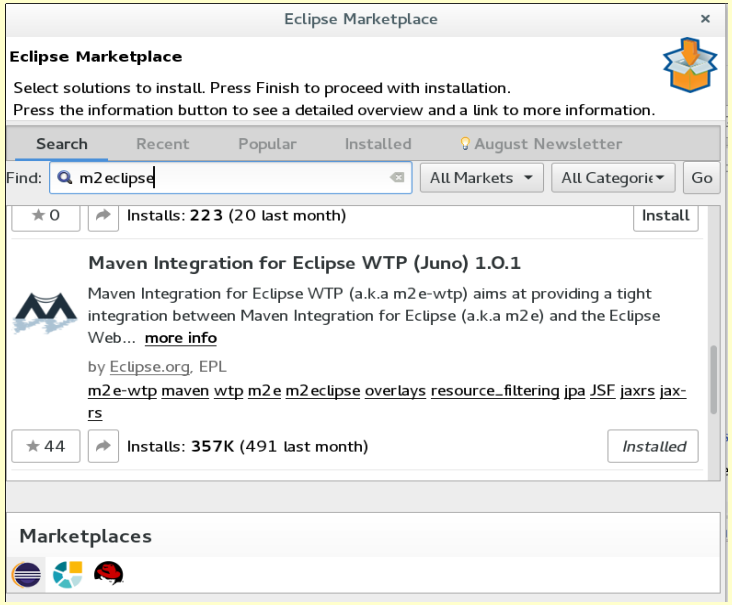
Wget http://ftp.cixug.es/apache/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.zip

unzip apache-maven-3.3.9-bin.zip

mv apache-maven-3.3.9 /opt

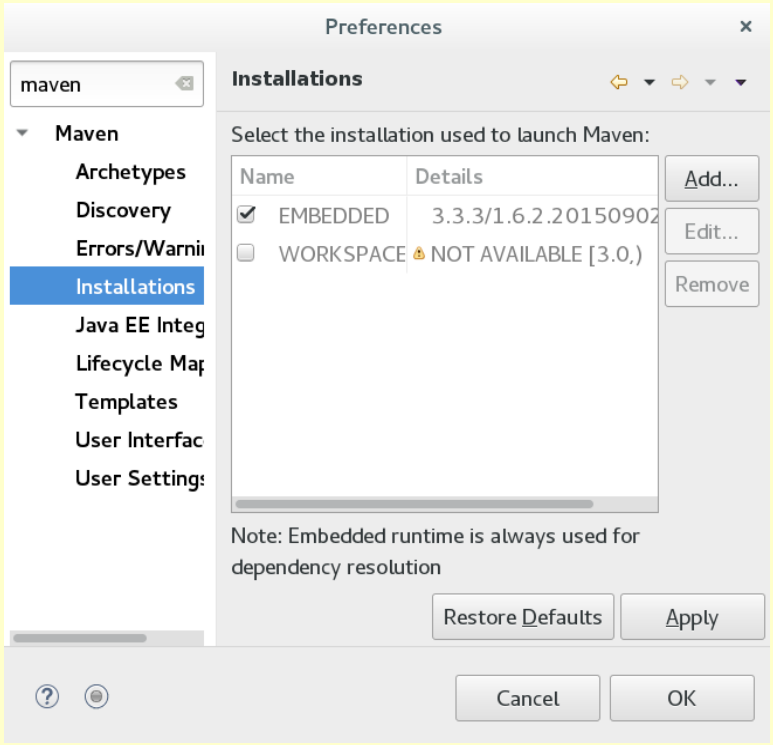
export PATH=/opt/apache-maven-3.3.9/bin:$PATH

INSTALACIÓN ECLIPSE



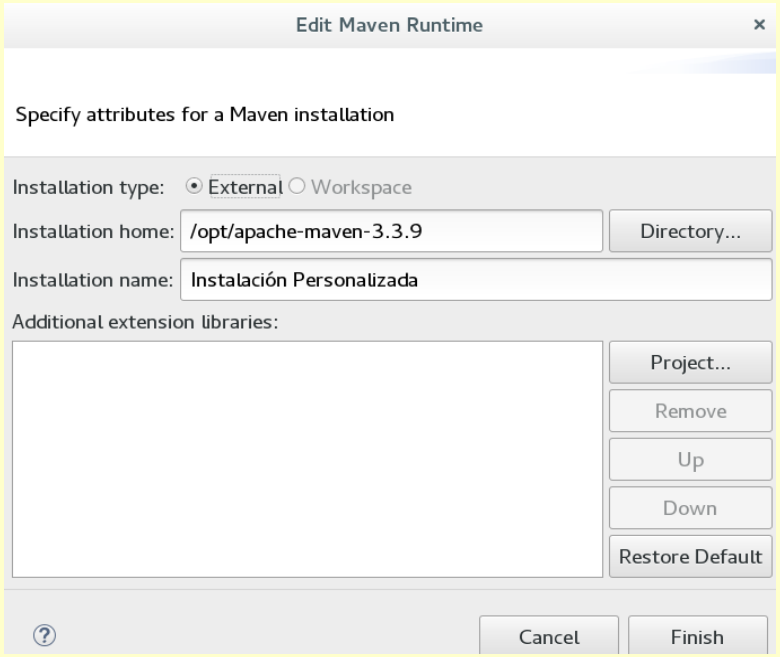
Versión por defecto m2e-WTP: Maven más plugins extras para manejar proyectos JEE

INSTALACIÓN ECLIPSE



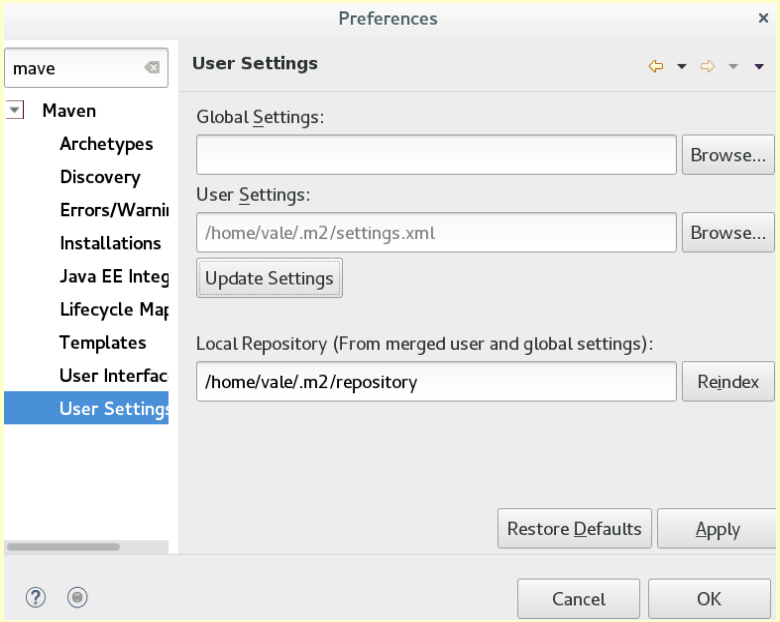
VERSIÓN por defecto Windows Preferences Maven Installations que podemos personalizar (para trabajar con la última edición)

INSTALACIÓN ECLIPSE



ADD → introducimos la nueva ruta, dando un nombre y aceptamos

INSTALACIÓN ECLIPSE



Settings → parámetros, usuarios repositorios remoto, local (jars, arquetipos)

MAVEN: Claves

Apache Maven es una herramienta para la gestión del desarrollo de un proyecto. Su funcionamiento gira alrededor de un archivo XML denominado POM.xml (Project Object Model)

Ofrece automatismos para generar compilaciones, distribuciones, la creación de documentación, gestión de publicaciones y de dependencias.

CONCEPTO GLOBAL

Maven puede extender su funcionalidad mendiante complementos o plug-ins, tanto para generar informes como mejorar el proceso de construcción/compilación.

Originalmente concebido para la gestión de proyectos JAVA, puede gestionar proyectos de otros lenguajes como C#, Ruby, Scala y otros.

CONCEPTO GLOBAL

Si bien Maven (2002) puede entenderse como sucesor de Make (C 1977) o Ant (Java 2000), recientemente han aparecido tecnologías alternativas como Gradle (Android) y SBT (Scala) que sin emplear XML como lenguaje declarativo, manejan conceptos similares.

CARACTERÍSTICAS

Estructura estándar de un proyecto

Gestión de dependencias semi-automática

Fácil acceso a librerías e info de proyectos

Plugins JAVA o script (bat / sh)

Acceso inmediato a nuevas características Maven

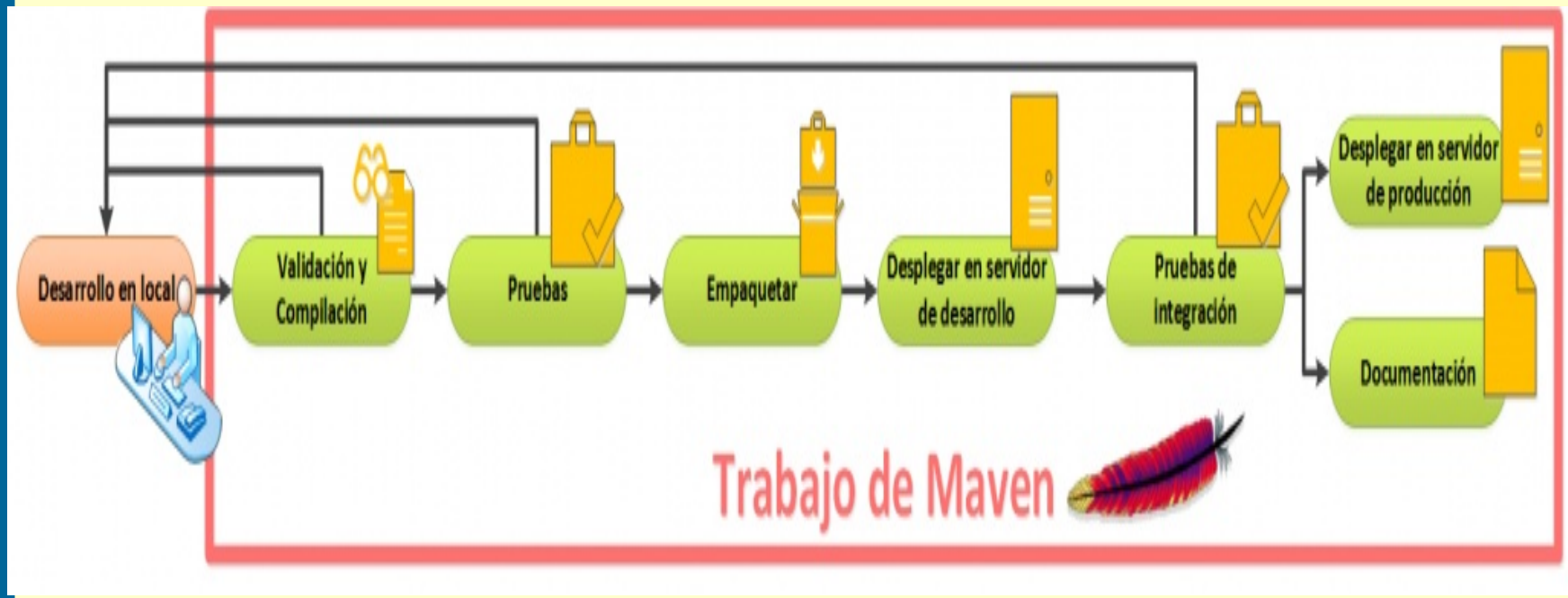
Gestión de despliegue del proyecto

Distribución y publicación de proyectos/librerías semiautomática.

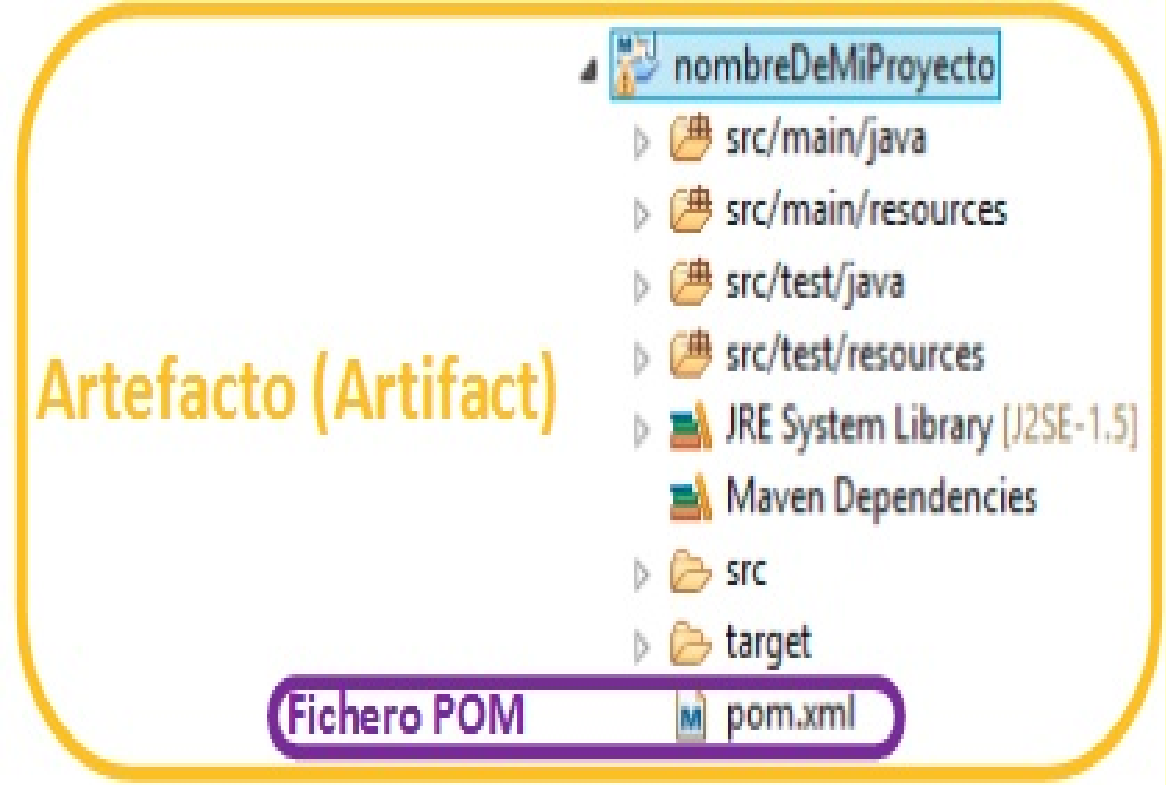
CARACTERÍSTICAS

- Generación documentación en formato Web y PDF
- Integración con CVS
- Gestión completa ciclo de vida: Creación, Testing, Despliegue, Documentación
- Descarga de arquetipos (proyectos plantilla)

CARACTERÍSTICAS



ARTIFACT


nombreDeMiProyecto
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
Artefacto (Artifact)
Fichero POM
pom.xml

ARTIFACT Y POM

ARTEFACTO : Es simplemente el nombre que recibe un proyecto gestionado por MAVEN. Para Maven, los proyectos, son “artifacts”

POM : El fichero POM (Viene de “Project Object Model”) o “pom.xml”: Fichero que describe toda la información sobre el proyecto y la configuración de forma declarativa

DEFINICIONES

GROUPID: Atributo que se demanda al crear un nuevo proyecto MAVEN. Generalmente, se corresponde con el paquete de la app (nombrado de más general a más específico) Distintos proyectos, pueden compartir idéntico groupID (ej: org.springframework)

ARTIFACTID: Nombre de la aplicación, que dará además nombre al JAR/WAR/EAR resultante de construir el proyecto (ej: spring-test)

DEFINICIONES

VERSION: Hasta 3 niveles A.B.C

PACKAGE: Generalmente, idéntico al GroupID

SNAPSHOT: Versión de la librería en desarrollo

RELEASE: Versión estable de uso público

Dependencias

De forma manual, editando la sección del POM a tal efecto <dependencies>, añado un elemento

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

Dependencias

La referencia puedo buscarla en la web https://mvnrepository.com/ y copiar /pegar la depencia

O también De forma automática, por Eclipse:

Botón derecho → Maven → Add dependency

Activar antes Windows / preferecences / download repository index para facilitar la búsqueda

Arquetipos MAVEN

Para facilitar la creación de proyectos MAVEN, la herramienta proporciona una serie de proyectos usados como moldes o puntos de partida, denominados ARQUETIPOS

Deberé elegir un arquetipo a la hora de crear un proyecto, el cual contendrá fuentes y dependencias por defecto

Arquetipos MAVEN

Con la sentencia mvn archetype:generate puedo generar un nuevo proyecto basado en un Arquetipo. Podemos definir nuestro propios Arquetipos, aunque ya viene definidos unos cuantos

https://maven.apache.org/guides/introduction/introduction-to-archetypes.html

Arquetipos MAVEN

El descriptor del arquetipo - archetype.xml en el directorio src/main/resources/META INF/maven/ -

Ficheros plantilla -fuentes, test y recursos- bajo: src/main/resources/archetype-resources/

El pom.xml prototipo, en src/main/resources/archetype-resources

El pom del arquetipo (en el directrio raíz)

Arquetipos MAVEN

ArchetypeDescriptor: Fichero que describe al arquetipo y se ubica en

src/main/resources/META-INF/maven/archetype.xml

Contiene la descripción de dónde se ubican los fuentes, los test, los recursos, el site

Arquetipos MAVEN

```
<archetype xmlns="http://maven.apache.org/plugins/maven-archetype-
    plugin/archetype/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-archetype-
    plugin/archetype/1.0.0 http://maven.apache.org/xsd/archetype-1.0.0.xsd">
  <id>quickstart</id>
  <sources>
    <source>src/main/java/App.java</source>
  </sources>
  <testSources>
    <source>src/test/java/AppTest.java</source>
  </testSources>
</archetype>
```

Arquetipos MAVEN

Puedo añadir ficheros plantilla de recursos con el elemento <resources>, como se muestra a continuación:

```
<resources>
  <resource>src/main/java/resources/imagen1.jpg</resource>
 </resources>
```

Arquetipos MAVEN

Ficheros plantilla: Serán copiados a las carpetas del destino, del nuevo proyecto, especificadas en el arquetipo

src/main/resources/META-INF/maven/archetype.xml

Arquetipos MAVEN

Ficheros plantilla: Si en el fichero arquetipo viene esta entrada

```
<sources>
    <source>src/main/java/App.java</source>
</sources>
```

App.java, deberá ubicarse en

src/main/resources/archetype-resources/src/main/java/App.java

Arquetipos MAVEN

En los fuentes, conviene definir el paquete de forma parametrizada, para que el valor concreto se establezca en tiempo de creación del proyecto

package ${package};

Arquetipos MAVEN

Pom prototipo: Será el pom de partida del proyecto creado a partir del arquetipo. Incluirá las dependencias que deseemos de inicio

src/main/resources/archetype-resources/pom.xml

Los atributos groupId, artifactId y version, quedan definidos como variables (${})

Arquetipos MAVEN

```
 <groupId>${groupId}</groupId>
<artifactId>${artifactId}</artifactId>
<version>${version}</version>
<packaging>jar</packaging>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
   </dependency>
</dependencies>
```

Arquetipos MAVEN

Pom propio del arquetipo: Será el pom utilizado para identificar al prototipo, ubicado en la carpeta raíz

```
<groupId>my.groupId</groupId>
<artifactId>my-archetype-id</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

Arquetipos MAVEN

Una vez con estos ficheros y en la carpeta raíz del arquetipo creado ejecuto

mvn install

Se producen las validaciones y una vez satisfechas, se exporta el arquetipo al repositorio local, quedando a disposición como plantilla de los potenciales proyectos

Arquetipos MAVEN

```
mvn archetype:generate -DarchetypeGroupId=my.groupId
      -DarchetypeArtifactId=my-archetype-id
      -DarchetypeVersion=1.0-SNAPSHOT
      -DgroupId=com.val.ebtm
      -DartifactId=my-app-prototipada
      -DarchetypeRepository=/root/.m2/repository
      -DinteractiveMode=false
```

Al ejecutar el comando anterior desde una carpeta nueva, creo un nuevo proyecto basado en arquetipo definido anteriormente.

Arquetipos desde Eclipse

También puedo crear un arquetipo, basándome en el arquetipo maven-archetype-archetype, para una vez generado, proceder a la instalación install) y en la perspectiva Maven respostories, botón derecho y rebuild index para que quede disponible.

Hecho lo anterior, teóricamente puedo crear mi proyecto, basándome en el repositorio (marcar archetypes snapshot) seleccionándolo entre el catálogo existente. (esta opción presenta algunos bugs)

Arquetipos MAVEN

Para la publicación de arquetipos en el repositorio central, a disposición de cualquier usuario, es necesario firmar el proyecto, sólo admitiéndose versiones RELEASE. Algo más laborioso, aunque igualmente factible. La información completa aquí:

https://maven.apache.org/guides/mini/guide-central-repository-upload.html

Proyecto simple

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false (creo)

cd my-app/

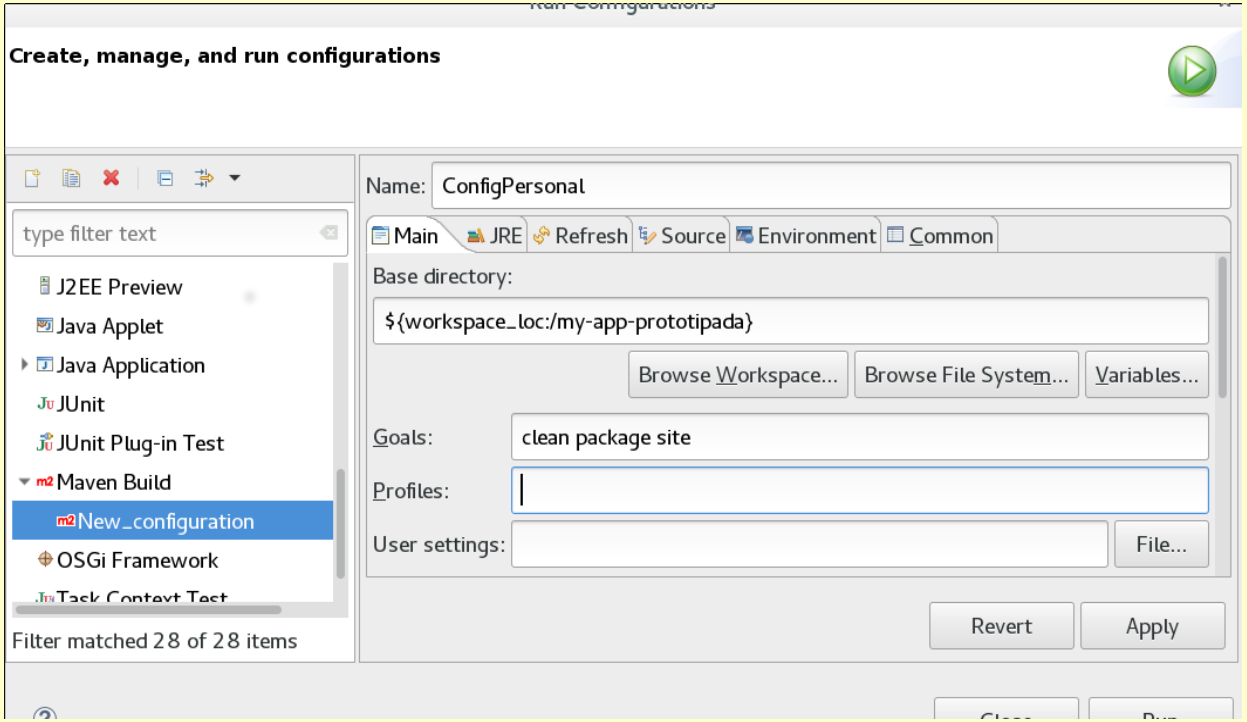
mvn package (empaqueto)

java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App (classpath) -ejecuto-

mvn site (genero documentación)

Igual desde Eclipse

Boton derecho Run → run as → run configurations → Maven Build → Nueva



COMANDOS

La primera vez que se ejecuta un comando Maven descarga todos los plugins y dependencias que necesita, por lo que tardará un poco más de tiempo, el resto de veces, únicamente descargará lo que haya cambiado

COMANDOS

La ejecución de un comando tiene el siguiente aspecto o combinaciones:

mvn <phase>
mvn <plugin:goal>

mvn clean dependency:copy-dependencies package

FUNCIONALIDAD

La funcionalidad en Maven se consigue en cuatro entidades, relacionadas e independientes entre sí:

1. Ciclos de vida (Build Lifecycles)
2. Fases (Build Phases)
3. Plugins
4. Goals

Ciclos de Vida

- Un Ciclo de Vida define un conjunto de
- FASES, con un orden de ejecución entre
- ellas.

Maven trae tres ciclos de vida predefinidos (built-in lifecycle) para su uso en un proyecto que son :default, clean y site.

Ciclos de Vida - Fases

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

Fases

Cada fase, representa un estado en el Ciclo de Vida. Al ejecutar una fase, se ejecutan todas las precedentes de forma automática.

Las fases, llevan asociadas por defecto un conjunto de goals. Dependiendo del tipo de empaquetado, una fase llevará asociada de forma predeterminada unos goals u otros

Fases de un packing JAR

process-resources resources:resources

compile compiler:compile

process-test-resources resources:testResources

test-compile compiler:testCompile

test surefire:test

package jar:jar

install install:install

deploy deploy:deploy

Fases

validate Valida el proyecto si es correcto y si toda la información necesaria está disponible

initialize Inicializa el estado de construcción (build). Por ejemplo, establece las propiedades o crea directorios

generate-sources Genera cualquier código para incluirlo en la compilación

process-sources Procesa el código. Por ejemplo, para filtrar cualquier valor

Fases

generate-resources Genera los recursos para incluirlos en el paquete

process-resources Copia y procesa los recursos del directorio de destino (los ficheros que están en la carpeta “src”), los prepara para el empaquetado

compile Compila el código del proyecto

process-classes Post-procesa los ficheros generados por la compilación. Por ejemplo: mejorar el bytecode de clases Java

Fases

generate-test-sources Genera todas las pruebas (tests) del código para incluirlo en la compilación

process-test-sources Procesa las pruebas (tests) del código. Por ejemplo: filtrar cualquier valor

generate-test-resources Crea los recursos para probar (testing)

process-test-resources Copia y procesa los recursos que están dentro del directorio de destino de pruebas (los ficheros que están en la carpeta “test”)

Fases

test-compile Compila el código de prueba (test) del directorio de pruebas (carpeta “test”)

process-test-classes Post-procesa los ficheros generados por la compilación. Por ejemplo: mejorar el bytecode de clases Java

test Ejecuta las pruebas utilizando un Framework apropiado para los test unitarios (por ejemplo “JUnit”). Estas pruebas no deben requerir que el código esté empaquetado (packaged) o desplegado (deployed)

Fases

prepare-package Realizar todas las operaciones necesarias para preparar un paquete antes del empaquetado (package) actual. Esto suele resultar en una no empaquetado (unpacked), versión procesada del paquete

package Empaquetar el código compilado en un formato distribuible, como en un JAR, WAR, EAR, EJB, POM, MAVEN-PLUGIN

pre-integration-test Realizar acciones que requieran la ejecución de pruebas de integración. Esto debería implicar cosas como configurar el entorno necesario

Fases

integration-test Procesar (process) y desplegar (deploy) el paquete si fuera necesario en un entorno donde se puedan ejecutar pruebas de integración

post-integration-test Realizar acciones requeridas después de ejecutar las pruebas de integración. Esto debería incluir limpiar el entorno.

verify Ejecutar todas las comprobaciones para verificar la validez del paquete y que cumpla con los criterios de calidad

Fases

install Instala el paquete en un repositorio local, para poder utilizarse como dependencia en otros proyectos locales

deploy Realizar en un entorno de integración o de producción (release), copia el proyecto final al repositorio remoto para compartirlo con otros desarrolladores y proyectos

Fases

Fases del ciclo de vida de limpieza (Clean Lifecycle)

pre-clean Ejecuta los procesos necesarios antes de ejecutar el proyecto de limpieza

clean Elimina todos los ficheros generados por construcciones (build) anteriores

post-clean Ejecuta los procesos necesarios para finalizar el proyecto de limpieza

Fases

Fases del ciclo de vida de documentación (Site Lifecycle)

pre-site Ejecuta los procesos necesarios antes de ejecutar de generación de la página (o sitio) web de documentación

site Genera la página web de documentación del proyecto

post-site Ejecuta el proceso necesario para finalizar la generación de la página web de documentación, y prepara para desplegar el sitio

Fases

site-deploy Despliega la página web de documentación generado en el servidor web especificado

Plugin

Los plugin agrupan goals que persiguen una funcionalidad común.

Además, al crear el plugin, éste queda asociado a una determianada fase, para la cual se ejecutarán su goals

Fase-Plugin-Goal

Si deseo ejecutar un goal o tarea para una fase, basta con añadir un elemento execution en la en el plugin que refiera a la fase.

Al ejecutar una fase, esta buscará en el interior de los plugins a qué goals aplica o debe ejecutar.

El orden de ejecución es del aparición en el POM

Fase-Plugin-Goal

```
<plugin>
  <groupId>com.mycompany.example</groupId>
  <artifactId>display-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>  <execution>
      <phase>process-test-resources</phase>
      <goals>
        <goal>time</goal>
      </goals>
    </execution> </executions>
</plugin>
```

Goal-Phase

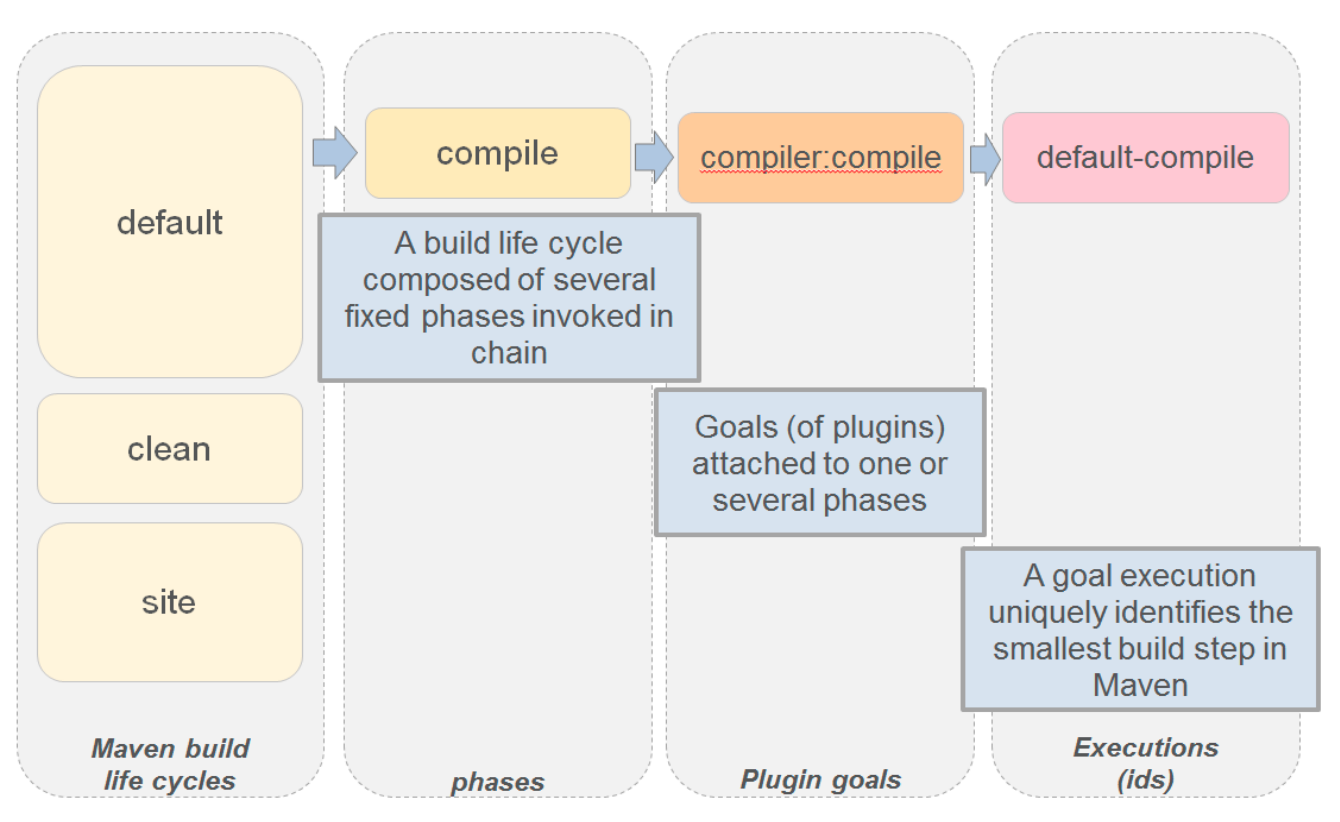
Automáticamente al ejecutarse una fase, se ejecutarán sus plugins asociados y con ello su goals.

Pero puede darse, que quiera ejecutar un goal, a una fase. Es posible, gracias a incluir el elemento <execution> en el plugin

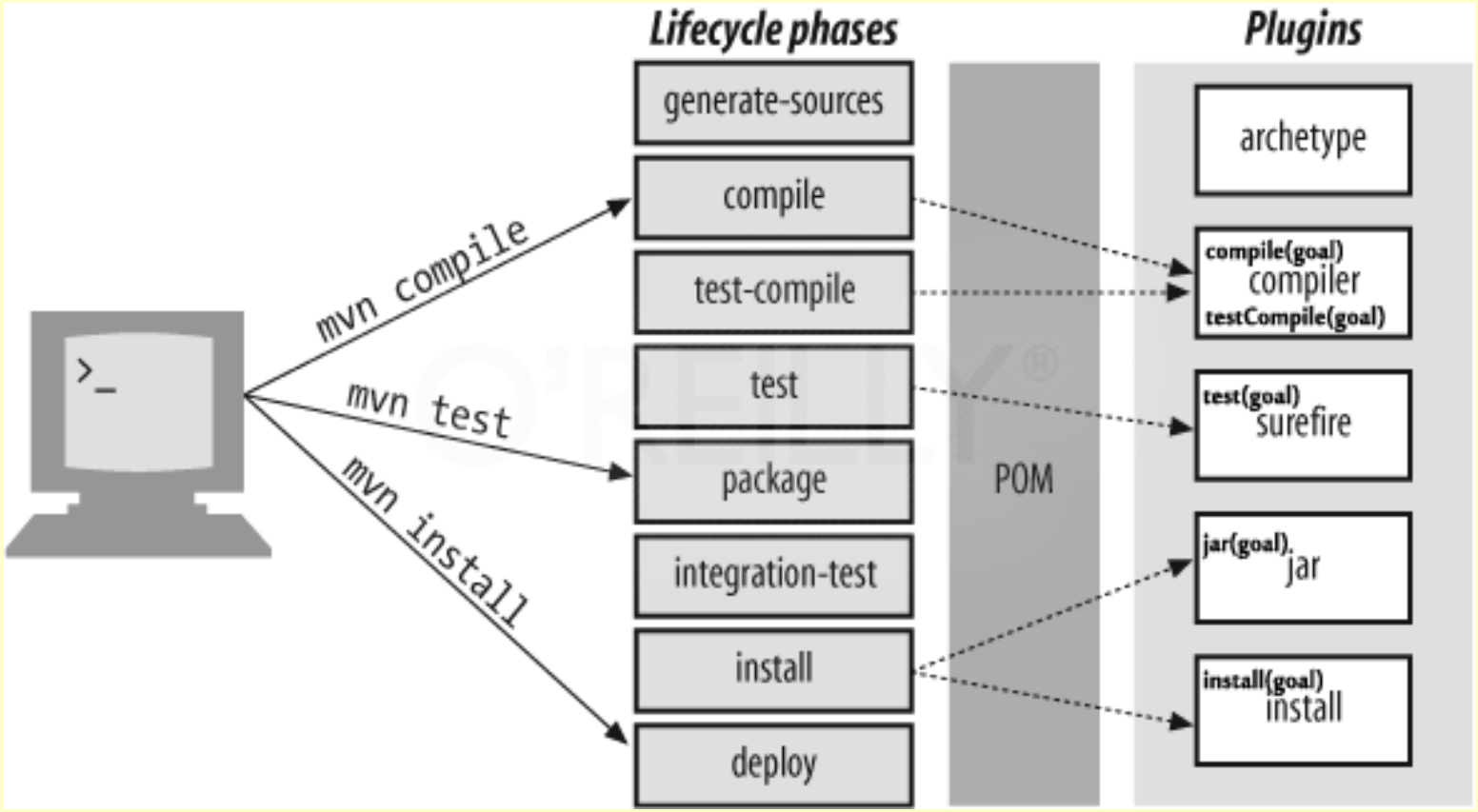
Goal-Phase

```
<plugin>
  <groupId>com.mycompany.example</groupId>
  <artifactId>display-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <phase>process-test-resources</phase>
      <goals>
        <goal>time</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Phases-Goals



Phase-Plugin-Goal



POM

El POM es el archivo xml que define el proyecto Maven. Formalmente, queda definido por el XSD (“gramática”) http://maven.apache.org/xsd/maven-4.0.0.xsd

Describe el proyecto y almacena toda la información relativa a él.

POM – Elementos básicos

Los elementos básicos son:

<groupId>...</groupId>

<artifactId>...</artifactId>

<version>...</version>

<packaging>...</packaging>

<dependencies>...</dependencies>

<parent>...</parent>

<dependencyManagement>...</dependencyManagement>

<modules>...</modules>

<properties>...</properties>

POM – Elementos básicos

```
<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>org.codehaus.mojo</groupId>
 <artifactId>my-project</artifactId>
 <version>1.0</version>
</project>
```

groupId:artifactId:version – COORDENADAS (ID del proyecto)

POM – Empaquetado

Maven supone el formato jar si no se especifica otro empaquetado. Para el atributo packaging, podemos emplear

Pom Jar Maven-plugin Ejb Ear War Rar Par

POM – Dependecias

```
<dependencies>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.0</version>
  <type>jar</type>
  <scope>test</scope>
  <optional>true</optional>
</dependency>
```

POM – Dependecias

Scope : Optimiza compilaciones (valor defecto)

Version: Qué versión emplear

1.0: Me vale cualquiera

[1.0]: Sólo me vale la 1

(,1.0]: Me vale versiones hasta la 1

[1.2,1.3]: Me valen versiones entre 1.2 y 1.3

[1.5,): Me valen versiones superiores a 1.5

(,1.1),(1.1,) Me valen todas menos la 1.1

POM – Dependecias

Si un paquete A depende de otro B.1 y a su vez, un paquete C depende de otra versión B.2, se pueden descargar dependecias transitivas que provoquen errores.

Para ello, bien puedo decir omitrilas mediante el elemento exclusion

POM – Dependecias

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <artifactId>hamcrest-core</artifactId>
      <groupId>org.hamcrest</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

POM – Herencia

Un POM hereda de otro varios atributos como las dependencias, la información de los desarrolladores, los plugins. Ello se consgiue indicando

```
<parent>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <relativePath>../my-parent</relativePath>
 </parent>
```

POM – Herencia

Por defecto, todo POM hereda de SuperPOM, donde vienen definidos los repositorios, perfiles y demás información.

Cuando quiera ver el resultado de componer el pom de proyecto con sus padres, estoy refiriéndome al POM efectivo

POM – Agregación

```
<groupId>org.codehaus.mojo</groupId>
<artifactId>my-parent</artifactId>
<version>2.0</version>
<packaging>pom</packaging>

<modules>
  <module>my-project</module>
  <module>another-project</module>
</modules>
</project>
```

POM – Propiedades

Con el elemento <properties><persona>

<propiedad>valor</propiedad></persona>

</properties> puedo definir claves que luego usar a modo de constates con {persona.propiedad}

POM – Build Settings

En esta sección indico aspectos más avanzados de mi proyecto como: qué goals ejecuto por defecto, dónde están los recursos, plugins usados en la construcción, qué rutas almacenan qué recursos, qué plugin uso para crear la documentación

POM – Info adicional

En esta sección se indican aspectos descriptivos como Licencia del proyecto, colaboradores, datos de la organización y desarrolladores

POM – Extra

La aspiración de maven es tal, que permite incluso indicar: qué sistema se emplea en gestión de errores de proyecto (BugZilla , Jira, etc), qué repositorios se emplean para almancenar consultar los artifact, datos del SCM (git), direccion del site a pubicar el proyecto, etc.

Multi-módulo

Si hacemos una app de cierto tamaño, podemos considerar dividir el código en diferentes jar y a su vez, un war para desplegarlo.

Maven permite este tipo de proyectos gracias a los multi módulo

Multi-modulo

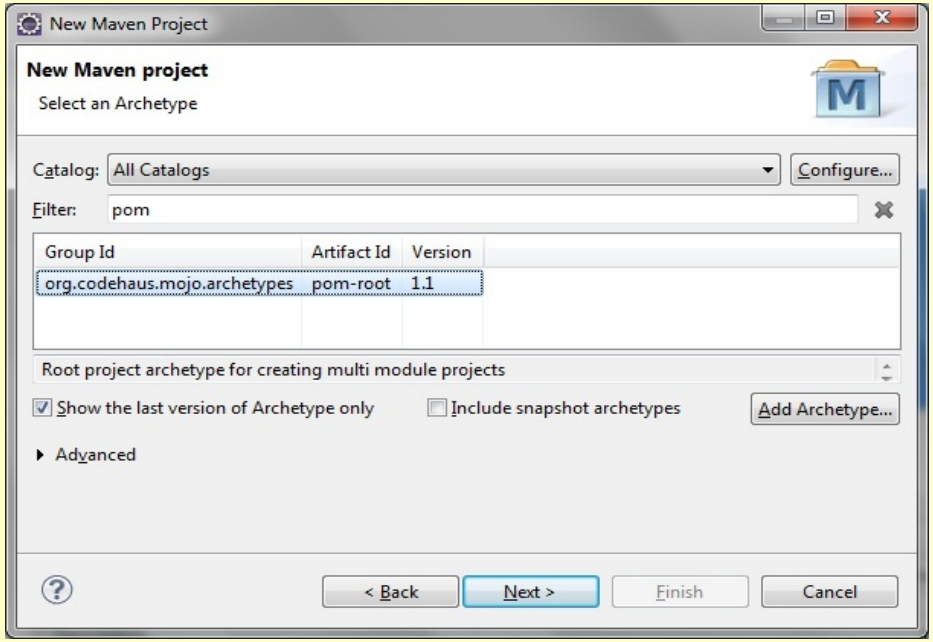
Para nuestro ejemplo, vamos a emplear

Un proyecto padre con empaquetado pom que contenga dos submódulos:

- Un proyecto jar
- Un proyecto war que emplee el jar

Multi-modulo

Creamos el proyecto padre filtrando por el arquetipo pom



Multi-modulo

Ahora creo un módulo de maven, indicando el padre. Este nuevo módulo, será el proyecto web basado en el maven-archetype-webapp y añadiendo las librerías de servlet por Target runtime

Multi-modulo

Creo otro modulo, indcando el mismo padre usando esta vez un arquetipo jar sencillo maven-archetype-quickstart

Por último, añado la dependecia al módulo web del jar

mvn clean package y se aplicará el comando a cada módulo

Repositorios Maven

REPOSITORIO LOCAL . - Donde se descargan e instalan los arquetipos y librerías

¿Qué pasa si tiro de una librería que está en otro repositorio distinto? → Añadir repositorio

Añadir repositorio

Añado estas líneas al settings.xml o al pom

```
<repositories>
  <repository>
    <id>Spark repository</id>
<url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  </repository>
</repositories>
```

Añadir repositorio

Y desde Eclipse, esa dependecia, puede ser visualizada en : Window → Show View → Maven Repositories

Añadir librerías extras

También puedo descargarme una librería (Oracle no se encuentra en ningún repo público por cuestiones de licencia) e insertarla en mi repositorio con install

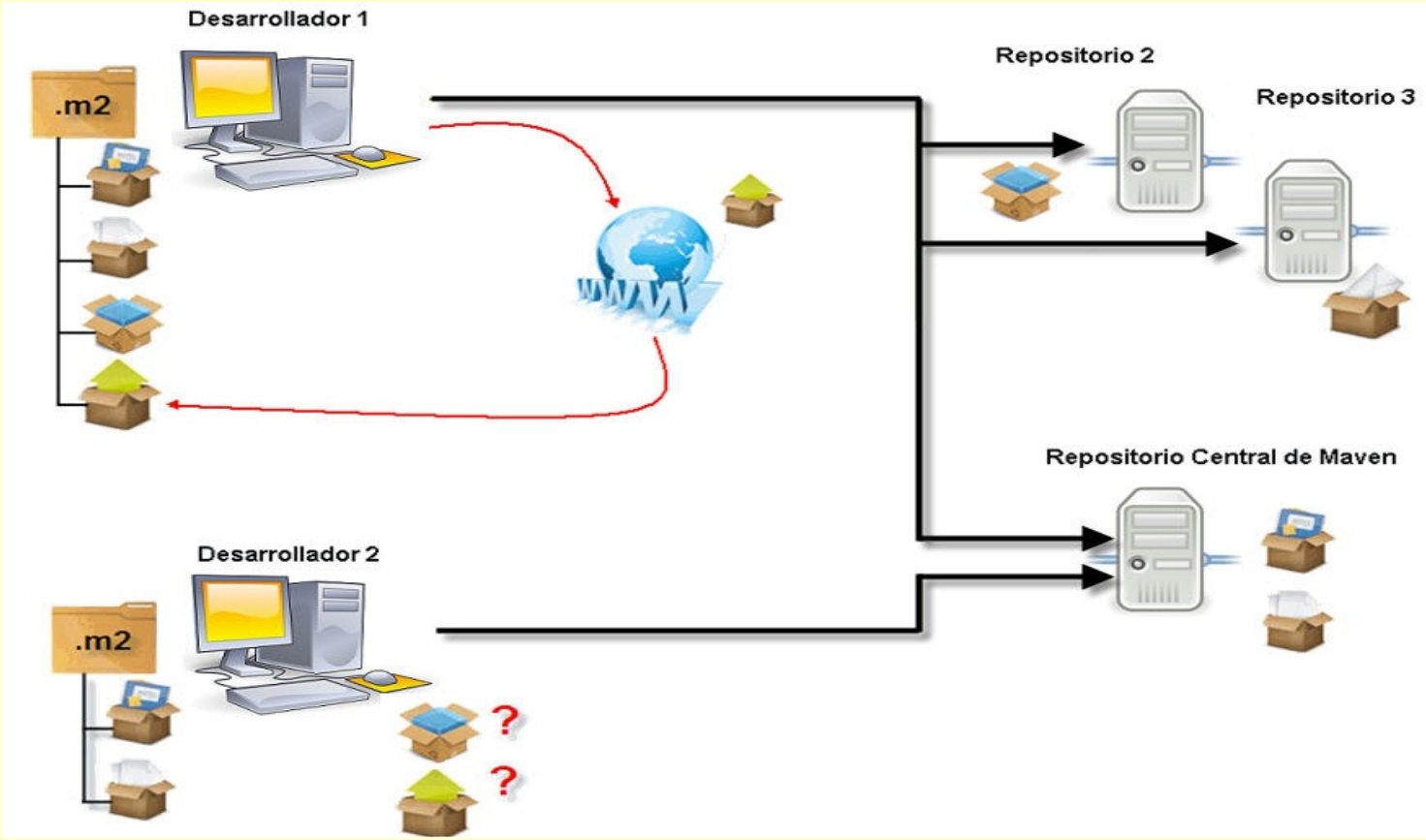
mvn install:install-file
-Dfile={Path/al/fichero/ojdbc.jar}
-DgroupId=com.oracle -DartifactId=ojdbc6
-Dversion=11.2.0 -Dpackaging=jar

Añadir librerías extras

Y después añadir la depedencia del jar instalado

```
<dependency>
  <groupid>com.oracle</groupid>
  <artifactid>ojdbc6</artifactid>
  <version>11.2.0</version>
</dependency>
```

Repositorio Corporativo



Repositorio Corporativo

En un escenario donde hay librerías NO presentes en los repositorios centrales públicos, se hace necesario a menudo la configuración y gestión de un equipo de trabajo desde un repositorio local

Repositorio Corporativo

VENTAJAS:

Almacenará las librerías propias de la organización, de manera que se puedan crear fácilmente dependencias a otros proyectos internos. Contendrá tanto artefactos en desarrollo (snapshot), como versiones liberadas (releases).

Repositorio Corporativo

VENTAJAS:

Almacenará las librerías que necesitemos y que hayamos descargado de manera manual.

Servirá de proxy a otros repositorios. Los desarrolladores únicamente se conectan al repositorio corporativo, y es dicho repositorio el que centraliza las conexiones a otros repositorios remotos.

Repositorio Corporativo

VENTAJAS:

Proporciona herramientas de control de acceso, gestión de usuarios, envío de correos, etc, que ayudan a definir y coordinar los equipos de desarrollo.

NEXUS

Nexus es un app web Java, sobre Tomcat que usaremos como implementación del repositorio corporativo

Hay otras aternativas como Archiva de Apache, pero Nexus se prefiere Nexus por ser de los creadores de Maven

NEXUS

Descargar Nexus

https://www.sonatype.com/download-oss-sonatype

Usamos la versión 2.13 (madurez)

Descomprimimos tar -zxvf

Ejecutamos ./nexus start (/bin)

Abrimos localhost:8081/nexus

Login admin admin123

NEXUS

Gestionar Repostorios

Añadir artifacts

Gestionar usuarios

Desplegar artifacts

Consumir depedencias

NEXUS

Tipos de repositorios:

Proxy.- Repositorios públicos

Hosted.- Almacén de libs propias

- 3rd Party No están en repos públicos
- Releases / Snapshots Las mías

Group.- Agrupa varios repos

NEXUS

Puedo subir librerías, que puedo etiquetar con los atributos de un artifact, y automáticamente se crea el POM que me permite usar la referencia

Selecciono el repositorio 3rd party y hago upload / Creo el proyecto y lo consumo

NEXUS

Subimos una librería de terceros a 3rd party

Artifact Upload (pesataña inferior)

Seleccionamos librería

Add Artifact

Upload Artifact

NEXUS USUARIOS

USUARIOS: Añadimos un usario para poder acceder a él desde Maven y subir nuestros artifacts, así como descargarnos las dependencias. Existen también roles para otorgar permisos

Panel lateral Securty → User → New User Nexus

CONECTAR NEXUS

Para acceder desde nuestro proyecto eclipse a Nexus, necesitamos parametrizar dos archivos:

- **El archivo settings.xml**
- **El POM**

CONECTAR NEXUS

Settings.xml Windows/preferences/Maven/User Settings $home/.m2/settings.xml

```
<servers>
        <server>
            <id>repodesarrollo</id>
            <username>valentino</username>
            <password>valentino</password>
        </server>
</servers>
```

CONECTAR NEXUS

Settings.xml Windows/preferences/Maven/User Settings $home/.m2/settings.xml

```
<mirrors>
    <mirror>
        <id>minexus</id>
        <mirrorOf>*</mirrorOf>
        <url>http://localhost:8081/nexus/content/groups/public</url>
    </mirror>
</mirrors>
```

CONECTAR NEXUS

Pom.xml Indico dónde desplegar

```
<distributionManagement>
        <snapshotRepository>
            <id>repodesarrollo</id>
            <name>Repositorio de desarrollo </name>
<url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
        </snapshotRepository>
    </distributionManagement>
```

CONECTAR NEXUS

Hecho lo anterior, ya puedo incluir dependencias a artifacts subidos en mi repositorio, así como desplegar mis proyectos/artifacts en el servidor local

```
<dependency>
        <groupId>com.val.edu</groupId>
        <artifactId>jotason</artifactId>
        <version>1</version>
    </dependency>
```

Plugins de serie

Los mavens incluídos en la distribución de la herramienta, son listados en la web del proyecto Maven:

https://maven.apache.org/plugins/

A la hora de ejecutarlos, tengo una copia presente en mi repo local

Plugins personalizados

Puedo ejecutar tareas propias o personalizadas, y definir mis propios plugins para que queden a disposición de mi proyecto y otros usuarios

Plugins personalizados

Nomenclatura: No emplear maven-minombre-plugin como id del artifact, porque está reservado para los plugins de serie

La clase Java que en realidad se ejecuta es una clase MOJO (La M de Maven por la P de POJO). Es equivalente a un “goal”. Un plugin, tendrá varios MOJOs

Custom plugins (Shell)

1 Creo un proyecto Maven con el tipo packaging <packaging>maven-plugin</packaging>

2 Creo una clase que herede de AbstractMojo en el paquete indicado, implementando execute ()

3 Dentro del proyecto, ejecuto mvn install, para que se instale en el repositorio local

4 Una vez instalado, puedo probar su ejecución

mvn paquete:idArtifact:version:goal

mvn com.val.edu:miplugin:0.0.1-SNAPSHOT:migoal

Custom plugins (Eclipse)

1 Creo un proyecto Maven con el arquetipo maven-archetype-plugin

2 Creo una clase que herede de AbstractMojo en el paquete indicado

3 Dentro del proyecto, ejecuto mvn install, para que se instale en el repositorio local

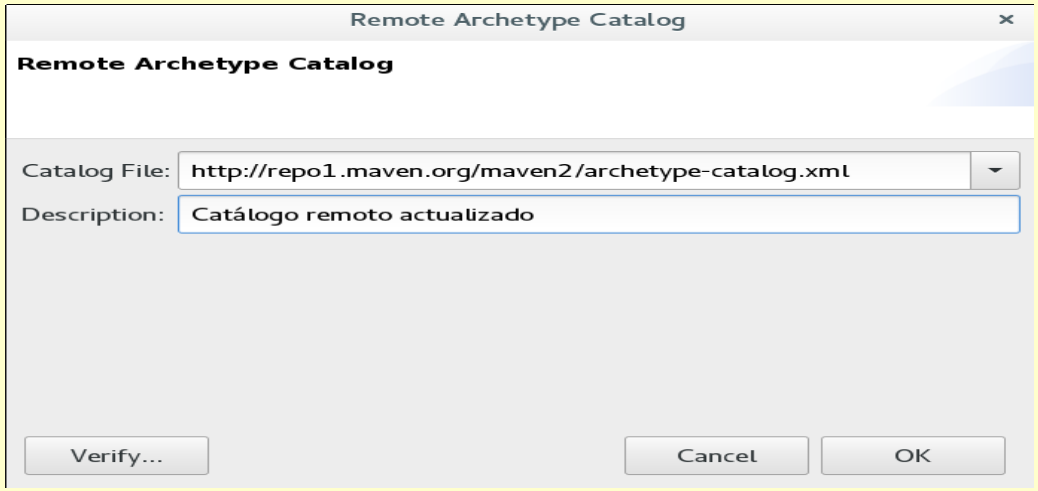
4 Una vez instalado, puedo probar su ejecución

mvn paquete:idArtifact:version:goal

mvn com.val.edu:miplugin:0.0.1-SNAPSHOT:migoal

Plugins personalizados

Puede ser necesario apuntar al catálogo original, por si el espejo no contiene algunas referencias (windows / prefereces)



Ejecución MOJOS

Creo un proyecto Maven y referencio en la sección build y ya puedo invocarlo (Add plugin)

```
<build>
  <plugins>
    <plugin>
      <groupId>sample.plugin</groupId>
      <artifactId>hello-maven-plugin</artifactId>
      <version>1.0-SNAPSHOT</version>
    </plugin>
  </plugins>
</build>
```

Ejecución MOJOS

O bien asocio la ejecución en un ciclo de vida. Por ejemplo, a la fase compile

```
<build>
  <plugins>
    <plugin>
      <groupId>sample.plugin</groupId>
      <artifactId>hello-maven-plugin</artifactId>
      <version>1.0-SNAPSHOT</version>
      <executions>
            ...
```

Ejecución MOJOS

```
...          <execution>
        <phase>compile</phase>
        <goals>
          <goal>sayhi</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
 </plugins>
</build>
```