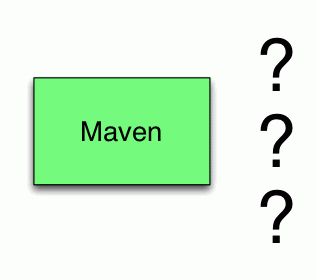
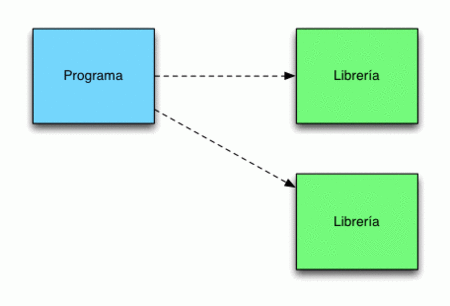
**Maven ???**

En muchas ocasiones me he encontrado con que es difícil hacer llegar a los desarrolladores la necesidad de usar una herramienta **como Maven.** Incluso a veces es difícil de explicar para que sirve exactamente. Vamos a ver una introducción.

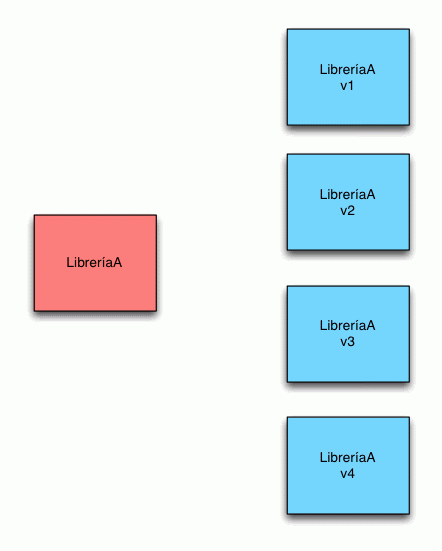


Normalmente cuando nosotros trabajamos con Java/JavaEE el uso de librerías es algo común como en cualquier otro lenguaje de programación.

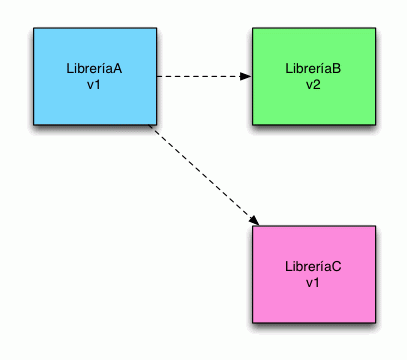


**Librerías y limitaciones**

El concepto de librería **es un concepto que a veces es limitado.** Por ejemplo nosotros podemos querer utilizar la librería A en nuestro proyecto. Sin embargo no nos valdrá con simplemente querer utilizar la librería sino que ademas necesitaremos saber que versión exacta de ella necesitamos.

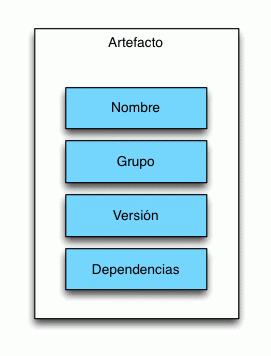


**¿Es esto suficiente?**. Lamentablemente no lo es, una librería puede depender de otras librerías para funcionar de forma correcta. Así pues necesitamos más información para gestionarlo todo de forma correcta.



**Maven y Artefactos**

Maven solventa esta problema a traves **del concepto de Artefacto. Un Artefacto puede verse como una librería con esteroides** (aunque agrupa mas conceptos). Contiene las clases propias de la librería pero ademas **incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias etc).**



**Artefactos y POM**

Para definir un **Artefacto** necesitamos crear un **fichero POM.xml (Proyect Object Model)** que es el encargado de almacenar toda la información que hemos comentado anteriormente:

<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>com.genbetadev.proyecto1</groupId>

<artifactId>proyecto1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

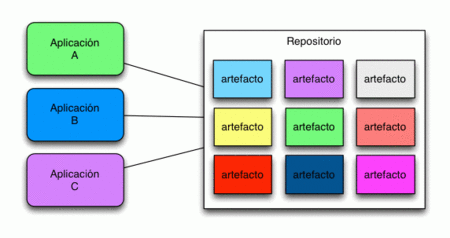
</dependencies>

</project>

La estructura del fichero puede llegar a ser muy compleja y puede llegar a depender de otros POM. En este ejemplo **estamos viendo el fichero más sencillo posible.** En el se define el nombre del Artefacto (artifactID) el tipo de empaquetado (jar) y también las dependencias que tiene (log4j). **De esta manera nuestra librería queda definida de una forma mucho más clara.**

**Maven Repositorio y Artefactos**

Una vez definidos correctamente todos los Artefactos que necesitamos, Maven nos provee de un Repositorio donde alojar, mantener y distribuir estos. Permitiéndonos una gestión correcta de nuestra librerías, proyectos y dependencias.



El uso de Maven es a día de hoy una necesitad en cualquier proyecto Java/Java EE de cierta entidad.

# Maven

|  |  |
| --- | --- |
| **Maven** | |
| [Maven logo.svg](https://commons.wikimedia.org/wiki/File:Maven_logo.svg) | |
| **Desarrollador(es)** | |
| [Apache Software Foundation](https://es.wikipedia.org/wiki/Apache_Software_Foundation) [*http://maven.apache.org*](http://maven.apache.org/) | |
| **Información general** | |
| [**Autor(es)**](https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software) | Jason van Zyl |
| **Lanzamiento inicial** | [30 de marzo](https://es.wikipedia.org/wiki/30_de_marzo) de [2002](https://es.wikipedia.org/wiki/2002) |
| [**Última versión estable**](https://es.wikipedia.org/wiki/Fases_del_desarrollo_de_software) | 3.3.9 [14 de noviembre](https://es.wikipedia.org/wiki/14_de_noviembre) de [2015](https://es.wikipedia.org/wiki/2015) (1 año, 6 meses y 1 día) |
| [**Género**](https://es.wikipedia.org/wiki/Anexo:G%C3%A9neros_de_software) | [Desarrollo Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) |
| [**Programado en**](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) | [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) |
| [**Plataforma**](https://es.wikipedia.org/wiki/Plataforma_(inform%C3%A1tica)) | [Máquina virtual Java](https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java) |
| [**Licencia**](https://es.wikipedia.org/wiki/Licencia_de_software) | [Licencia Apache 2.0](https://es.wikipedia.org/wiki/Apache_License) |
| [[editar datos en Wikidata](https://www.wikidata.org/wiki/Q139941)] | |

**Maven** es una herramienta de software para la gestión y construcción de proyectos [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) creada por Jason van Zyl, de Sonatype, en 2002.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo *de facto* de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. Maven provee soporte no solo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje. En realidad, el soporte y uso de lenguajes distintos de Java es mínimo. Actualmente existe un plugin para .Net Framework y es mantenido,[1](https://es.wikipedia.org/wiki/Maven#cite_note-1) y un plugin nativo para C/C++ fue alguna vez mantenido por Maven 1.[2](https://es.wikipedia.org/wiki/Maven#cite_note-2)

## Teoría[[editar](https://es.wikipedia.org/w/index.php?title=Maven&action=edit&section=1)]

### Convención sobre configuración[[editar](https://es.wikipedia.org/w/index.php?title=Maven&action=edit&section=2)]

La filosofía general de Maven es la estandarización de las construcciones generadas por seguir el principio de [Convención sobre Configuración](https://es.wikipedia.org/wiki/Convenci%C3%B3n_sobre_Configuraci%C3%B3n), a fin de utilizar existentes modelos en la producción de software. Esta estrategia necesariamente restringe ampliamente la variabilidad, lo que se refleja en la exhortación de Maven a adherirse a su modelo de proyecto.[3](https://es.wikipedia.org/wiki/Maven#cite_note-3) Mientras que Maven es adecuado para nuevos proyectos, los proyectos complejos ya existentes pueden ser simplemente no adaptables para que utilicen Maven. La falta de restricciones de la convención de capas del proyecto fue hecha de alguna manera más configurable con el lanzamiento de Maven 2.

### Reutilización[[editar](https://es.wikipedia.org/w/index.php?title=Maven&action=edit&section=3)]

Maven está construido alrededor de la idea de reutilización, y más específicamente, a la reutilización de la lógica de construcción. Como los proyectos generalmente se construyen en patrones similares, una elección lógica podría ser reutilizar los procesos de construcción. La principal idea es no reutilizar el código o funcionalidad (como Apache Ant), sino simplemente cambiar la configuración o también código escrito. Esa es la principal diferencia entre Apache Ant y Apache Maven: el primero es una librería de utilidades y funciones buenas y útiles, mientras que la otra es un framework configurable y altamente extensible.[4](https://es.wikipedia.org/wiki/Maven#cite_note-4)

Aunque Maven es configurable, históricamente el proyecto Maven ha enfatizado seriamente que los usuarios deben adherirse a su concepto de un modelo de proyecto estándar tanto como sea posible.

### Ciclo de vida[[editar](https://es.wikipedia.org/w/index.php?title=Maven&action=edit&section=4)]

Las partes del ciclo de vida principal del proyecto Maven son:

1. *compile*: Genera los ficheros .class compilando los fuentes .java
2. *test*: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
3. *package*: Genera el fichero .jar con los .class compilados
4. *install*: Copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
5. *deploy*: Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos maven, por ejemplo, si ejecutamos **mvn install**, maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas). Estas metas son:

1. *clean*: Elimina todos los .class y .jar generados. Después de este comando se puede comenzar un compilado desde cero.
2. *assembly*: Genera un fichero .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero xml qué se debe incluir en ese zip.
3. *site*: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
4. *site-deploy*: Sube el sitio web al servidor que hayamos configurado.
5. etc.

Pero estas metas pueden ser añadidas al ciclo de vida a través del Project Object Model (POM).

## http://www.javiergarzas.com/2014/06/maven-en-10-min.html

## [Simple y rápido. Entiende qué es Maven en menos de 10 min.](http://www.javiergarzas.com/2014/06/maven-en-10-min.html)

### Primeros pasos con Maven

En nuestros proyectos java siempre tenemos varias tareas que realizar. La primera suele ser crear una estructura de directorios para nuestro proyecto, con un hueco para los fuentes, otro para iconos, ficheros de configuración o datos, directorio para dejar los .class o el .jar, para dejar el javadoc, etc, etc.

Después, tenemos más tareas que realizamos con cierta frecuencia, como borrar los .class, compilar, generar la documentción de javadoc, el jar, incluso generar documentación web para publicar nuestro trabajo. Posiblemente acabemos haciendo algunos scripts o ficheros .bat para todas estas tareas.

Si nuestro programa es grande, incluso es posible que dependamos de otros jar externos, como drivers debase de datos, JUnit para clases de test, log4j para nuestra salida de log, etc, etc. Tendremos que copiar todos esto jar externos en algún sitio de nuestro proyecto e incluirlos.

Una primera herramienta que nos ayuda un poco con todo esto es **[ant](http://www.chuidiang.org/java/herramientas/ant.php)**. Sin embargo, con **ant** no tenemos todas estas tareas hechas y debemos reescribir posiblemente nuestro fichero **build.xml** (el de tareas que se ejecutan con **ant**) de un proyecto a otro. También tendremos que copiar los jar externos de los que dependemos.

Si vamos a una herramienta más evolucionada, llegamos a **maven**. **Maven**, con comandos simples, nos crea una estructura de directorios para nuestro proyecto con sitio para los fuentes, los iconos, ficheros de configuración y datos, etc, etc. Si a **maven** le indicamos qué jar externos necesitamos, es capaz de ir a buscarlos a internet y descargarlos por nosotros. Sin necesidad prácticamente de configurar nada, **maven** sabe como borrar los .class, compilar, generar el jar, generar el javadoc y generar un documentación web con montones de informes (métricas, código duplicado, etc). **Maven** se encarga de pasar automáticamente nuestros test de prueba cuando compilamos. Incluso **maven** nos genera un zip de distribución en el que van todos los jar necesarios y ficheros de configuración de nuestro proyecto.

En este tutorial vamos a dar nuestros primeros pasos con **Maven**.

#### ¿Qué es Maven?

Ya lo hemos contado más o menos, pero para que quede claro. **Maven** es una herramienta de línea de comandos, sin interface gráfica ni nada que se le parezca, que nos ayuda a crear los directorios de nuestro proyecto y con las tareas habituales que se realizan en él, como compilado, generar jar, documentación, distribuir, dependencias con otros jar, etc.

#### Instalar maven

Antes de que te pongas a hacer nada, necesitas tener una conexión de internet. **Maven** la primera vez que se ejecuta empiza a bajarse cosas y si no puede, no sirve para nada.

Aunque la instalación que cuento aquí es para windows, supongo que en linux es muy similar. Basta [bajarse el zip de maven](http://maven.apache.org/download.html), desempaquetarlo y poner su directorio **bin** en el path de busqueda de ejecutables.

En mi caso lo he desempaquetado en **C:\**, con lo que se me ha creado el directorio **C:\maven-2.0.4**

Para meter el subdirectorio **bin** en el path de ejecutables, le doy con el botón derecho del ratón sobre el icono de "Mi PC" y elijo "Propiedades", "Avanzado" y "Variables de Entorno". Busco "Path" y le doy a "Editar". Añado al final ";C:\maven-2.0.4\bin" (con punto y coma delante, para separlo de lo que ya haya escrito)

En linux supongo que es cuestión de coger el fichero .login, .profile, .bashrc o el que sea según el unix/linux que tengamos correspondiente del $HOME del usuario y añadirle al final la línea

PATH=$PATH;/directorio\_maven/maven-2.0.4/bin

En el caso de windows (y en el de linux) hay que abrir una ventana de comandos nueva después de haber realizado los cambios (ventana de ms-dos o bash) y para probar que todo va bien, podemos ejecutar

C:\> mvn --version  
Maven version: 2.0.4  
  
$ mvn --version  
Maven version: 2.0.4

Es posible que la primera vez que lo ejecutemos tarde un rato, ya he comentado que cada vez que ejecutamos un comando nuevo de **maven**, tiene que bajarse cosas de internet.

#### Crear un proyecto

El primer paso que podemos hacer con maven es **crear un proyecto** desde cero. El comando de **maven** que tenemos que ejecutar es

mvn archetype:create -DgroupId=chuidiang.ejemplos -DartifactId=EjemploMaven

Veamos los parámetros

* **archetype:create** es el comando/plugin, o como quieras llamarlo, de maven para crear un proyecto. Por defecto crea un proyecto de java normalito (nada de aplicación web, aunque también se puede)
* **-DgroupId=**chuidiang.ejemplos es el conjunto de proyectos al que pertenece nuestro proyecto. Por ejemplo, yo puedo meter todos mis programas de ejemplo en un grupo que llamaré "chuidiang.ejemplos". Este nombre que pongamos aquí va a servir de paquete inicial para todas las clases del proyecto. Todos los proyectos **maven** deben pertenecer a un grupo, aunque sea único para él, que se denominará **groupId**.
* **-DartifactId=**EjemploMaven es el nombre que queramos dar al proyecto. **Maven** creará un directorio con este nombre y el jar que genere para el proyecto tendrá también este nombre. Todos los proyectos **maven** tienen un nombre para identificarlos, que se denomirá **artifactId**

Una vez ejecutado este comando, **Maven** empezará a bajarse cosas de internet cuando lo ejecutemos por primera vez (en los próximos proyectos ya no necesita bajarse nada) y creará una esctructura de directorios y ficheros como la siguiente

EjemploMaven  
+---src  
¦ +---main  
¦ ¦ +---java //Para nuestros fuentes  
¦ ¦ +---chuidiang  
¦ ¦ +---ejemplos

¦ ¦ +---App.java  
¦ +---test  
¦ +---java //Para test de Junit  
¦ +---chuidiang  
¦ +---ejemplos

¦ +---AppTest.java

+---pom.xml

Crea un directorio **EjemploMaven** para nuestro proyecto.

Dentro un fichero **pom.xml** que es un fichero que contiene datos de configuración de nuestro proyecto, como dependencias con otros jar, tipos de informes que queremos en la página web de nuestro proyecto, etc.. Inicialmente contiene una serie de cosas por defecto que podremos cambiar si lo deseamos.

Crea dos subdirectorios, uno **src** y otro **test**. Dentro de **src** debemos meter todo lo que sean fuentes y ficheros de configuración o datos propios del proyecto. En la parte de **test** debemos meter todos nuestros fuentes de prueba, clases de test de JUnit, ficheros de datos o de configuración de pruebas, etc. Es decir, en **src** va lo que es del proyecto y en **test** lo que nos ayude a probar el proyecto, pero no sea propio del proyecto.

En ambos casos, crea un directorio **java**. Ahí debajo es donde van los fuentes java. En paralelo a estos directorios **java** y si lo necesitamos, debemos crear nosotros a mano otros directorios. El nombre de estos nuevos directorios es standard en **maven**. Para un proyecto normalito los más útiles puede ser

* **config** para ficheros de configuración, iconos, etc
* **resources** para ficheros que queramos que se metan dentro del jar. **Maven** meterá automáticamente todo lo que haya en este subdirectorio dentro del jar
* **assembly** para la configuración que queramos en nuestro zip de distribución.

En <http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html> tienes la estructura standard de directorios propuesta por **maven**, pero completita.

**Maven** mete directamente el paquete inicial chuidiang.ejemplos, según indicamos en el **groupId** y en él mete un par de ficheros java de muestra, **App.java** y **AppTest.java**. Normalmente estos ficheros los borraremos, salvo que queramos aprovechar el nombre.

#### Empezamos a trabajar

Ahora llega el momento duro. Debemos empezar a escribir el código, tanto de nuestro proyecto como de las clases de test de JUnit, si es que seguimos al pie de la letra las costumbres de buena programación. Desgraciadamente, todavía no hay herramientas que hagan este trabajo por nosotros, así que a ello.

#### Compilar

Una vez que tenemos todo, podemos compiliar de forma sencilla. Basta ponerse en el directorio donde está el fichero **pom.xml** y escribir

mvn compile

Esto creará un directorio **target** justo debajo de **EjemploMaven** y ahí un subdirectorio **classes** donde meterá todos los .class de nuestro compilado

EjemploMaven  
+---src  
¦ +---main  
¦ ¦ +---java  
¦ ¦ +---chuidiang  
¦ ¦ +---ejemplos  
¦ +---test  
¦ +---java  
¦ +---chuidiang  
¦ +---ejemplos  
+---target  
 +---classes  
 +---chuidiang  
 +---ejemplos //Aqui van los ficheros .class

#### Generar el jar

Para generar el jar, es igual de sencillo

mvn package

Esto primero compilará si es necesario, pasará las clases de test de JUnit y si no hay fallos, meterá en el directorio **target** nuestro jar, que por defecto tendrá un nombre tan feo como este

EjemploMaven-1.0-SNAPSHOT.jar

Bien, **EjemploMaven** es como nosotros queremos que se llame según indicamos en el **artifactId**. **Maven** añade un 1.0 para indicar que es la versión 1.0 de nuestro proyecto. Este número aparece y podemos cambiarlo en el fichero **pom.xml**. Lo de **-SNAPSHOT**es para indicar que esta versión está en construcción, que no es definitiva. **Maven** irá guardando todas las versiones del jar que generemos e irá sustituyendo **-SNAPSHOT** por la fecha y hora de la construcción. Esto de **-SNAPSHOT** también aparece en el **pom.xml** y podemos quitarlo cuando creamos que tenemos la versión definitiva.

Dentro de jar se meterán automáticamente todos los ficheros que tengamos debajo del directorio **resources**

#### Repositorios Maven

Una de las grandes ventajas de **maven** son los repositorios (almacenes) de ficheros jar que se crea.

Si miras en <http://www.ibiblio.org/maven2/> tienes el repositorio oficial de jars de maven. Ahí están los **groupId** de casi todos los jar de libre distribución que puedas encontrar en internet. Tienes el log4j, commons-logging, JFreeChart, mysql-connector, etc, etc. **Maven** es capaz de bajarse cualquiera de estos jar si tu proyecto lo necesita.

Todo lo que se baje **maven** de internet lo mete en un repositorio (almacen) local en tu pc, de forma que si lo necesita una segunda vez, no necesita descargárselo nuevamente de internet. Este directorio, habitualmente está en

* **$HOME/.m2** en unix/linux
* **C:\Documents and Settings\usuario\.m2** en windows

Adicionalmente podemos configurar **maven** para que use más repositorios, tanto de internet como otros que montemos internamente en nuestra red local si la tenemos. Por ejemplo, en internet tienes otro repositorio con jars independiente de maven en <http://java.freehep.org/maven2/>

El repositorio interno en la red local es particularmente útil si trabajamos en una empresa y hacemos proyectos que son dependientes unos de otros o tenemos nuestras propias librerías de jars. Poniendo jars en ese repositorio de empresa, todos los programadores podrán acceder o subir jars a ese repositorio y compartirlos.

#### Dependencias de nuestro proyecto

Una vez que sabemos que hay un montón de jars por el mundo a nuestra disposición, sólo tenemos que saber cómo hacer que **maven** se los baje cuando nosotros queramos.

Para decir que necesitamos un jar de los que algún alma caritativa ha puesto a nuestra disposición, tenemos que editar el fichero **pom.xml** que tiene por defecto esta pinta

<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>chuidiang.ejemplos</groupId>  
 <artifactId>EjemploMaven</artifactId>  
 <packaging>jar</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <name>Maven Quick Start Archetype</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>

Vemos que hay un apartado **dependencies** y que dentro tiene un **dependency** de junit. Esto es la opción por defecto. Para esta dependencia de Junit hay que dar el **groupId** (junit), el **artifactId** (junit otra vez), la **versión** que deseamos (3.8.1) y cuándo la necesitamos (en los test)

Imaginemos que queremos el **log4j**, pero para nuestro programa, no para el test.

Nos vamos al respositorio oficial de maven <http://www.ibiblio.org/maven2/> y navegamos hasta llegar el fichero **.pom** del log4j en la versión que queremos, es decir, vamos "pinchando" en: log4j, log4j, 1.2.13 y finalmente [log4j-1.2.13.pom](http://www.ibiblio.org/maven2/log4j/log4j/1.2.13/log4j-1.2.13.pom). Ahí vemos el **groupId**, **artifactId** y **versión** del log4j que queremos. Ahora editamos nuestro **pom.xml** y le añadimos la dependencia

<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>chuidiang.ejemplos</groupId>  
 <artifactId>EjemploMaven</artifactId>  
 <packaging>jar</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <name>Maven Quick Start Archetype</name>  
 <url>http://maven.apache.org</url>  
 <dependencies>  
 <dependency>  
 <groupId>**log4j**</groupId>  
 <artifactId>**log4j**</artifactId>  
 <version>**1.2.13**</version>  
 <scope>**compile**</scope>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>3.8.1</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

Esta vez hemos puesto "compile" en vez de "test" para indicar que lo necesitamos al compilar nuestro proyecto y no al hacerle el test. Con esto está todo listo. Ya podemos usar en nuestros fuentes el log4j. Cuando compilemos la primera vez, **maven** mirará si ya tenemos log4j en nuestro repositorio local y si no lo tenemos, se irá a buscarlo a internet y lo bajará. El log4j quedará en nuestro repositorio local.

#### Integración con los IDE

Bueno, todo esto está muy bien, pero si yo trabajo con un IDE, como eclipse, IntelliJ IDEA, netbeans o algo así, ¿cómo encuentro ahora los jar en un sitio tan escondido?. "Fácil" entre comillas. **Maven** es capaz de generar un proyecto de eclipse, de idea y creo que de netbeans. Basta ejecutar esto

mvn eclipse:eclipse  
mvn idea:idea

luego, desde nuestro IDE, creamos un proyecto importando o leyendo el fichero de proyecto que ha generado **maven**.

Digo "Fácil" entre comillas, porque al menos para eclipse, hay que hacer algo más. Debemos definir dentro de eclipse la variable **M2\_REPO** apuntando a nuestro repositorio local de **maven**. En la Chuwiki puedes ver los detalles de cómo se coge con [eclipse un proyecto maven](http://www.chuidiang.com/chuwiki/index.php?title=Integraci%C3%B3n_de_Maven_y_Eclipse#Importar_un_proyecto_maven_en_eclipse). Por lo que he visto en IntelliJ, es más inmediato y no hay que hacer nada especial.

También hay plugins en nuestros IDEs que hacen más fácil el trabajo con maven, de forma que desde nuestro IDE podamos ejecutar las tareas maven. En [codehaus](http://mevenide.codehaus.org/) tienes los plugins para los cuatro IDEs más conocidos: eclipse, netbeans, IntelliJ y JBuilder.

#### Llevar nuestro jar a los repositorios maven

Si queremos meter nuestro jar en el repositorio **maven** de jars local, es sencillo. Basta hacer

mvn install

Esto compilará si es necesario nuestros fuentes, les pasará los test, generará el jar y lo copiará en nuestro repositorio local de jars, en nuestro pc. Esta operación hace que ese jar esté disponible para otros proyectos **maven** que tengamos en nuestro ordenador. Es útil, por tanto, para proyectos **maven** que sean librerías nuestras que queramos usar en varios proyectos.

Si hemos configurardo un repositorio en red común a varios programadores en varios ordenadores, el comando es igual de simple

mvn deploy

Esto hace todo lo que hace **install**, incluido el install y luego pone nuestro jar en ese repositorio común en red. Esto lo pone a disposición de todos los programadores del equipo.

Lo de **-SNAPSHOT** en la versión tiene aquí su gracia. Si dependemos de un jar que tenga **-SNAPSHOT**, cada vez que compilemos, aunque ese jar esté en nuestro repositorio local, **maven** ira a buscarlos a los repositorios comunes o de internet, para ver si hay una versión de fecha más moderna. Si la hay, se la bajará. Por tanto, suele ser útil en un equipo de trabajo mantener la "coletilla" **-SNAPSHOT** en los jar que todavía están en desarrollo y sufren cambios frecuentes.

Si no ponemos **-SNAPSHOT**, una vez bajado el jar a nuestro repositorio local, **maven** NO se preocupará de buscar si hay versiones más modernas en los repositorios remotos.

#### Generar documentación

Generar el javadoc es fácil también entre comillas. Lo primero es que debemos editar el fichero **pom.xml** para indicarle que queremos este tipo de documentación. Puede quedar así

<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>chuidiang.ejemplos</groupId>  
 <artifactId>EjemploMaven</artifactId>  
 <packaging>jar</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <name>Maven Quick Start Archetype</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>

<reporting>  
 <plugins>  
 <plugin>  
 <groupId>**org.apache.maven.plugins**</groupId>  
 <artifactId>**maven-javadoc-plugin**</artifactId>  
 </plugin>  
 </plugins>  
 </reporting>  
</project>

Luego ya es sencillo. Basta ejecutar

mvn javadoc:javadoc

y **maven** nos generará en target un directorio **target\site\apidocs** y dentro de él metera el javadoc.

Si ejecutamos

mvn site:site

generará en **target\site** una documentación web por defecto, incluido el **javadoc**. Por supuesto, en el **pom.xml** y en algunos ficheros adicionales de configuración se podría personalizar. El aspecto de esa documentación por defecto puedes verlo [en la página de maven](http://maven.apache.org/), ya que por supuesto, la web de **maven** ha sido generada con **maven**. También en la página de [codehause](http://mevenide.codehaus.org/) o en la de [freehep](http://java.freehep.org/).

#### Para finalizar

Esto es lo básico para hacerse una idea de qué es **maven** y empezar con ello. Hay muchísimas más cosas que se pueden configurar en **maven**, pero nos alargaríamos demasiado. Por ejemplo, algunas de las cosas que se pueden hacer con **maven**:

* Generar un .jar con los fuentes
* Generar en formato web una documentación similar al javadoc, pero con los fuentes en colorines.
* Generar un .zip con todos los jar y ficheros de configuración, para distribuir nuestro programa
* Un análisis de métricas de nuestro código.
* Un análisis de cobertura de los test, indicando qué líneas de código se han ejecutado o no en los test.
* Pasar el **rmic** a los .class que lo necesitan
* Ejecutar desde **maven** tareas de **ant**
* Preparar una distrubución etiquetando todos los fuentes en cvs.
* Crear un proyecto maven compuesto de varios subproyectos
* Crear un gran jar que tenga todos los .class, tanto los nuestros como los de los jar ajenos. Evitamos así distribuir varios jar con nuestra aplicación.

En la Chuwiki tienes las [cosas de maven](http://www.chuidiang.com/chuwiki/index.php?title=Categor%C3%ADa:Maven) con las que me he ido tropezando y si no, en la [página de maven](http://maven.apache.org/) lo tienes supuestamente todo.

Tampoco me he metido, porque no lo controlo, en las utilidades para aplicaciones web con EJBs, pero las tiene. Basta crear un proyecto indicando que es una aplicación web y cuando le demos a compilar, nos generará los ficheros ear, war o los que sean.

<https://www.youtube.com/watch?v=VR-LrutW3iE&t=5s>

http://www.chuidiang.org/java/herramientas/maven.php

<https://www.youtube.com/watch?v=ZbWTvWFDsKw>

<https://www.youtube.com/watch?v=YepfC1XF1MU&t=1s>

<https://www.youtube.com/watch?v=qdwFAkQTQQU>

<https://www.youtube.com/watch?v=kMMNNEjUjEY>