

Guía rápida de Maven.

MAVEN. INSTALACIÓN Y CONFIGURACIÓN DE WILDFLY. RESTFUL EN WILDFLY

Maven como alternativa a Ant	2
Introducción	2
Propiedades	10
a) Estructura general de directorios de proyectos Maven	11
b) Instalación de Maven	15
I. Instalación de <i>Maven</i> desde el repositorio	15
II. Instalación de <i>Maven</i> desde los binarios	16
c) Creación de un Proyecto Maven	19
Creación de una aplicación Web	19

MAVEN COMO ALTERNATIVA A ANT

Apache Maven es una herramienta (*similar a Apache Ant*) Java basada en XML **para la gestión del proceso de construcción de proyectos software**, facilitando la compilación, prueba, empaquetamiento y despliegue. Entre sus ventajas está la de facilitar la **descarga de las bibliotecas** (archivos JAR) **externas** de las que depende un proyecto y agilizando la **construcción, prueba y despliegue del proyecto desarrollado**, produciendo el fichero JAR o WAR final a partir de su código fuente y del fichero **POM de descripción del proyecto**.

Como sucede con Apache Ant, herramientas de desarrollo como Netbeans, Eclipse, etc., permiten trabajar con Maven de modo visual, sin la configuración del usuario, pero también se recomienda su conocimiento de línea de órdenes.

INTRODUCCIÓN

En la página del proyecto pueden obtenerse detalles de configuración, instalación, uso, *plugins*, etc.: <http://maven.apache.org/index.html>

Como se ha comentado anteriormente, **Apache Maven** es una herramienta Java de línea de comandos (*similar a Ant, javac,...*) **basada en XML para la gestión del proceso de construcción de proyectos software**, simplificando la compilación, prueba, empaquetamiento y despliegue. Entre sus ventajas está:

- Descarga de **bibliotecas** (archivos JAR) **externas** de las que depende un proyecto.
- Facilita la **construcción, prueba y despliegue del proyecto desarrollado**, produciendo el fichero JAR o WAR (para aplicaciones Web) final a partir de su código fuente y del fichero **POM de descripción del proyecto** (pom.xml).

Maven nació en torno al 2002 en la comunidad *open source* **Apache Software Foundation**, diseñado principalmente por *Jason van Zyl* (ahora en *Sonatype*) y desarrollado inicialmente para facilitar la construcción del proyecto *Jakarta Turbine* en 2002 (<https://turbine.apache.org/>). El proyecto *Turbine* es un *Framework* basado en *servlets* que facilita la construcción rápida de aplicaciones Web, que puede emplearse para personalizar sitios Web y usar *logins* para restringir el acceso a partes de la aplicación.

En 2003 el proyecto fue aceptado como proyecto de nivel principal de Apache. En **octubre de 2005 se lanzó Maven 2**. Maven ha sido adoptado como la herramienta de desarrollo de software de muchas empresas y se ha integrado en infinidad de proyectos y entornos. En **2010** se lanzó la versión de **Maven 3.0**, en su mayoría compatible con

Maven 2. La última versión estable es **Apache Maven 3.8.6**, que requiere JDK 1.7 o superior (todas las versiones 3.3+ requieren JDK 1.7, 10MB de disco y adicional para el repositorio Maven, que, al menos, es de 500MB).

Además de emplearlo desde línea de órdenes, **es posible utilizar Maven en IDEs como Netbeans, Eclipse, Visual Studio Code o Glassfish**, pero siempre es **importante conocer la utilización de Maven en línea de órdenes y la estructura de un proyecto**, porque es la base de cualquier adaptación gráfica de gestión.

Una de las “novedades” principales de Maven es su **enfoque declarativo**, frente *make* o *Ant*, que están orientadas a tareas. En Maven, el proceso de compilación de un proyecto se basa en una **descripción de su estructura y de su contenido**.

Un proyecto Maven exige **definir un identificador único para cada proyecto que desarrollemos, así como declarar sus características** (URL, versión, bibliotecas, tipo y nombre del artefacto generado, etc.). Todas las características **se especifican en el fichero POM (Project Object Model, fichero *pom.xml* en el directorio raíz del proyecto)**.

Antes de proceder a la instalación, es necesario describir algunos conceptos de Maven. Sobre todo lo relacionado con el archivo POM.

Archivo POM

Cuando se ejecuta la orden Maven (***mvn***), ejecuta las órdenes descritas en el archivo POM (Project Object Model).

Es una **representación XML de un proyecto de Maven** guardado en un archivo llamado ***pom.xml***. Este fichero define **elementos XML preestablecidos que deben ser definidos para el proyecto concreto que estamos desarrollando**.

*“Un proyecto contiene archivos de configuración, así como los desarrolladores involucrados y los roles que desempeñan, el sistema de seguimiento de defectos, la organización y las licencias, la URL de donde reside el proyecto, las dependencias del proyecto y todas las otras pequeñas piezas que entran en juego para darle vida al código. Es una ventanilla única para todo lo relacionado con el proyecto. De hecho, en el mundo de Maven, un proyecto no necesita contener ningún código, simplemente un *pom.xml*.”*

Un archivo POM, contiene una lista de elementos bajo la etiqueta **“*project*”**. Empezando con la versión del modelo (***modelVersion***), que en la actualidad sólo admite la versión **4.0.0**:

```
<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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- Lo Básico -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>

  <!-- Configuración de construcción -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- Más información del proyecto -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Configuración del entorno -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

LO BÁSICO

El **POM** contiene toda la información necesaria sobre un proyecto, así como configuraciones de complementos que se utilizarán durante el proceso de compilación. Es la manifestación **declarativa** del "quién", "qué" y "dónde", mientras que el ciclo de vida de la compilación es el "cuándo" y "cómo".

POM Mínimo:

```
<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.pepinho.programacion</groupId>
  <artifactId>accesoDatos</artifactId>
  <version>1.0</version>
</project>
```

Coordenadas Maven

El POM mínimo que permite Maven debe tener, por lo menos, los **campos obligatorios**: **groupId:artifactId:version** (aunque, *groupId* y *version* no necesitan definirse explícitamente si se heredan de un padre). Los tres campos **actúan como una dirección** y un *timestamp* (identificador). Esto marca un lugar específico en un repositorio, actuando como un **sistema de coordenadas para proyectos de Maven**:

- **groupId** : suele ser único entre una organización o un proyecto. Por ejemplo, todos los artefactos principales de Maven están bajo groupId *org.apache.maven*, o los míos bajo *com.pepinho* ;-). No requieren notación de puntos como, por ejemplo, un proyecto junit (de pruebas unitarias). Es una buena elección que siga la estructura de paquetes empleada en el proyecto, empleando la jerarquía inversa (*com.dominio.subdominio...*)

Cuando se almacena dentro de un **repositorio**, el **groupId** actúa de manera muy similar a como lo hace la estructura de empaquetado de Java en un sistema operativo, en el que **los puntos se reemplazan por separadores de directorio** específicos del sistema operativo que se convierte en una estructura de directorio relativa del repositorio base. En el ejemplo, *com.pepinho.dam* se guardará dentro del directorio: **\$M2_REPO/com/pepinho/dam**.

- **artifactId** : *artifactId* es el **nombre por el que se conoce el proyecto**. Aunque el *groupId* es importante (todos los proyectos dentro de la organización suelen tener el mismo *groupId*), junto con el *groupId*, crea una clave que separa el proyecto de todos los demás proyectos del mundo (al menos, debería :-)). Junto con el **groupId**, **artifactId** **define completamente el alojamiento del artefacto dentro del repositorio**. En el caso del ejemplo de pom anterior, *accesoDatos*, se almacenaría en:
`$M2_REPO/com/pepinho/daw/accesoDatos`.
- **version**: ésta es la última parte del nombre. **groupId:artifactId** denota un único proyecto pero no determina que versión del proyecto se trata. Los cambios de código, esos cambios deben dar lugar a versiones diferentes, y este elemento mantiene esas versiones en línea. También se utiliza dentro del repositorio de un artefacto para separar versiones entre sí. *accesoDatos* versión 1.0 se almacena en la estructura de directorios `$M2_REPO/com/pepinho/daw/accesoDatos/1.0`.

Los tres elementos anteriores apuntan a una versión específica de un proyecto, lo que le permite a Maven saber con *quién* estamos tratando y *qué versión* en su ciclo de vida del software queremos.

Empaquetamiento (packing)

Una vez conocido el identificador de direccionamiento, *groupId:artifactId:version*, hay una etiqueta estándar más para nos informa (*qué* completo) del **tipo de empaquetamiento del proyecto**. En el ejemplo de POM `com.pepinho.dam:accesoDatos:1.0` definido anteriormente se empaquetará como un archivo **jar** (*por defecto*). Podríamos convertirlo en una **war** declarando un embalaje diferente:

```
<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
    https://maven.apache.org/xsd/maven-4.0.0.xsd" >
  ...
  <packaging>war</packaging>
  ...
</project>
```

Pues, cuando no se declara el tipo de empaquetamiento, Maven asume que el *packing* **por defecto** es **jar**.

Los tipos válidos para el packing, además de jar y war, son los roles válidos del componente [org.apache.maven.lifecycle.mapping.LifecycleMapping](https://maven.apache.org/ref/3.6.3/maven-core/default-bindings.html) (<https://maven.apache.org/ref/3.6.3/maven-core/default-bindings.html>)

En la actualidad los valores de empaquetamiento válidos son: ***pom, jar, maven-plugin, ejb, war, ear, rar***. Estos definen la lista predeterminada de objetivos que se ejecutan en cada etapa del ciclo de vida de compilación correspondiente para una estructura de paquetes en particular.

Dependencias (POM)

Una característica del desarrollo de proyectos Java es la **gran cantidad de bibliotecas** (ficheros JAR) disponibles y, a veces, necesarias para compilar y ejecutar un proyecto Java. Todas las bibliotecas que se importan deben estar disponibles (y descargadas) al compilar y ejecutar la aplicación/proyecto (web).

Mantener dependencias es lioso, sobre todo en proyectos grandes, y muy dado a provocar errores. Hay que obtener las bibliotecas, en las versiones correctas, así como las bibliotecas de las que dependen y distribuirlas en los dispositivos en los que el proyecto se va a desplegar.

Por ejemplo, si el proyecto emplea una implementación de JPA Hibernate se precisa descargar todos los JAR de Hibernate, junto con los JAR dependientes, una lista de más de 15 archivos.

Hacer estas tareas a mano para el desarrollo y despliegue es una tarea complicada. Con Maven el proceso de gestionar los paquetes de bibliotecas se precisa **declarar la dependencia con este JAR en el fichero POM**.

Por ejemplo:

```
...  
  <dependency>  
    <groupId>org.hibernate</groupId>  
    <artifactId>hibernate-entitymanager</artifactId>  
    <version>3.5.6-Final</version>  
  </dependency>  
...
```

Maven descarga todas las bibliotecas del proyecto ejecutando:

\$mvn install

Una vez descargada, la **guarda en el repositorio local**, situado en un archivo oculto (Linux), llamado **.m2** dentro del directorio del usuario. Posteriormente copia las referencias a las bibliotecas en el proyecto.

Maven permite manejar varios tipos de las relaciones de proyectos:

- **Dependencias** (y dependencias transitivas).
- Herencia.
- Agregación (proyectos de varios módulos).

La gestión de dependencias suele ser complicado a medida que crece la complejidad de los proyectos, creando un árbol de dependencias grande y complicado. Además, a veces las versiones de dependencias en un sistema no son equivalentes a las versiones desarrolladas, ya sea por la versión incorrecta proporcionada, o porque entran en conflicto con versiones con nombres de archivos jar similares.

Maven resuelve estos problemas disponiendo de **un repositorio local común** desde el que enlazar proyectos correctamente, versiones y todo.

El elemento principal del POM es la lista de dependencias. Maven descarga y vincula las dependencias en la compilación, así como en otros objetivos que las requieran. Como beneficio adicional, Maven incorpora las dependencias de esas dependencias (dependencias transitivas), lo que permite que su lista se centre únicamente en las dependencias que requiere el proyecto. Veamos un ejemplo:

```
<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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
...
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <type>jar</type>
    <scope>test</scope>
    <optional>true</optional>
  </dependency>
...
</dependencies>
...
</project>
```


Las dependencias Maven se describen por medio de coordenadas Maven (*groupId*, *artifactId*, *version*), por lo que las dependencias siempre deben ser artefactos Maven.

Si por algún motivo no se puede descargar la dependencia del repositorio central Maven (un JAR con licencia, por ejemplo), puede hacerse de tres modos:

a) **Instalar la dependencia manualmente** por medio del plugin *install*. Este método es el más sencillo de realizar.

```
mvn install:install-file -Dfile=non-maven-proj.jar -DgroupId=some.group -DartifactId=non-maven-proj -Dversion=1 -Dpackaging=jar
```

De este modo creará un POM con las coordenadas Maven indicadas.

- b) **Crear nuestro propio repositorio e implantarlo en él.** Es el método ideal para compañías con una intranet y que necesitan tener todas las bibliotecas sincronizadas. Para ese caso hay un plugin llamado *deploy:deploy-file*, similar al ejemplo anterior.
- c) **Crear un ámbito de dependencia “system”** y acceder a él median una ruta del sistema (*systemPath*). No se recomienda.

Versión de la dependencia

El elemento “**version**” de la dependencia **define los requisitos de la versión**, que se utilizan para calcular las versiones de las dependencias. Pueden indicarse requisitos estrictos de versión, obligando al uso de una versión concreta o requisitos laxos que pueden reemplazar la dependencia por diferentes versiones del mismo artefacto que se encuentran en otra parte del gráfico de dependencia.

En el caso de requisitos de versión estricta, si no hay versiones de una dependencia que satisfagan todos los requisitos estrictos para ese artefacto, la compilación falla.

Los requisitos de versión tienen la siguiente sintaxis. Ejemplos:

- 1.0: Requisito *suave* para 1.0. Utiliza 1.0 si no aparece ninguna otra versión antes en el árbol de dependencias.
- [1.0]: Requisito estricto para 1.0. Sólo utiliza esa versión.
- (,1.0]: Requisito estricto para cualquier versión ≤ 1.0 .
- [1.2,1.3]: Requisito estricto para cualquier versión entre 1.2 y 1.3 inclusive.
- [1.0,2.0): $1.0 \leq x < 2.0$; Requisito estricto para cualquier versión entre 1.0 inclusive y 2.0 exclusiva.
- [1.5,): Requisito estricto para cualquier versión mayor o igual a 1.5.
- (,1.0],[1.2,): Requisito estricto para cualquier versión menor o igual a 1.0 o mayor o igual a 1.2, pero no 1.1. Los requisitos múltiples están separados por comas.

- (,1.1),(1.1,): Requisito estricto para cualquier versión excepto 1.1; por ejemplo, porque 1.1 tiene una vulnerabilidad crítica.

Maven **elige la versión más alta de cada proyecto que satisface todos los requisitos estrictos** de las dependencias de ese proyecto. Si ninguna versión satisface todos los requisitos estrictos, la compilación falla.

Exclusión de dependencias

Mediante el elemento “exclusions”-> “exclusión”, puede indicarse a Maven que no incluya una dependencia de la dependencia (dependencias transitivas). Por ejemplo, el paquete *maven-embedder* requiere *maven-core*, y podríamos excluirla del siguiente modo:

```
...
<dependency>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven-embedder</artifactId>
  <version>2.0</version>
  <exclusions>
    <exclusion>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Propiedades

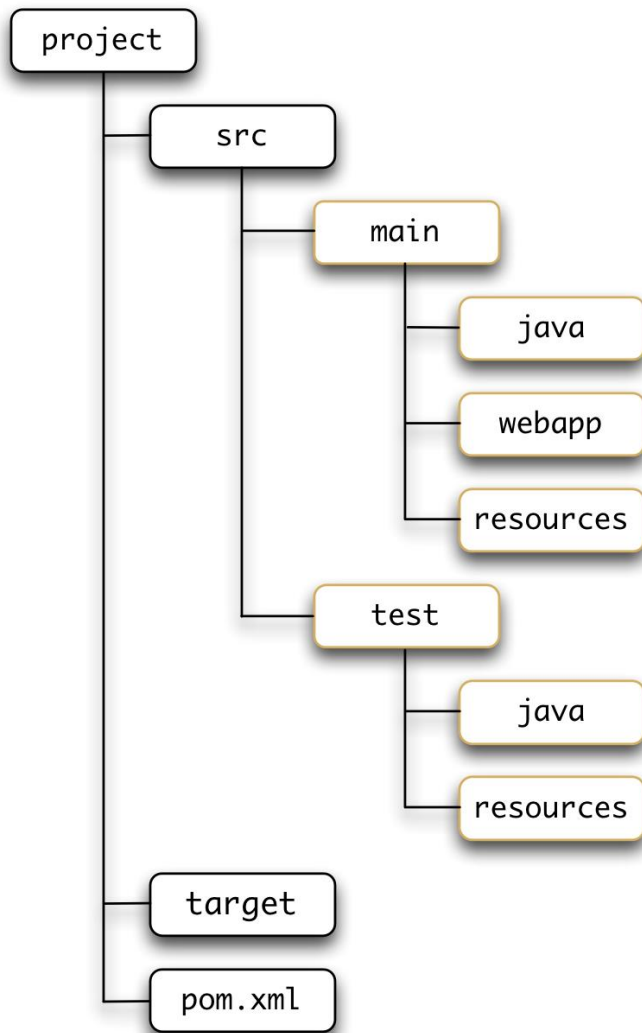
Las propiedades de Maven son marcadores de posición de valor, como las propiedades de Ant. Se **puede acceder a sus valores en cualquier lugar dentro de un POM utilizando la notación $\${X}$** , donde X es la propiedad. También pueden emplearse plugins como valores por defecto.

Ejemplo:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>14</maven.compiler.source>
  <maven.compiler.target>14</maven.compiler.target>
</properties>
```

A) ESTRUCTURA GENERAL DE DIRECTORIOS DE PROYECTOS MAVEN

Así, de modo general, la **estructura de directorios de un proyecto Maven** para guardar los distintos elementos de un programa Java **para una aplicación Web** (aunque su estructura para otro tipo de proyectos sigue la misma jerarquía, salvo *webapp*) son:



La estructura de un proyecto Maven tiene los siguientes elementos principales:

- **src: código fuente** del proyecto, tanto el código fuente de la aplicación (**main**) como clases de prueba (**test**). Ambos directorios incluyen:
 - Directorio **java**: paquetes de **código fuente** de la aplicación. El código fuente java se estructura en subdirectorios con el formato de los paquetes de la aplicación.
 - Directorio **webapp**: archivos HTML, JSP y de configuración de la aplicación web. Esto incluye las páginas de inicio (**index.jsp**, por ejemplo) y el subdirectorio **WEB-INF**, que contiene el archivo **web.xml** de la aplicación Web.

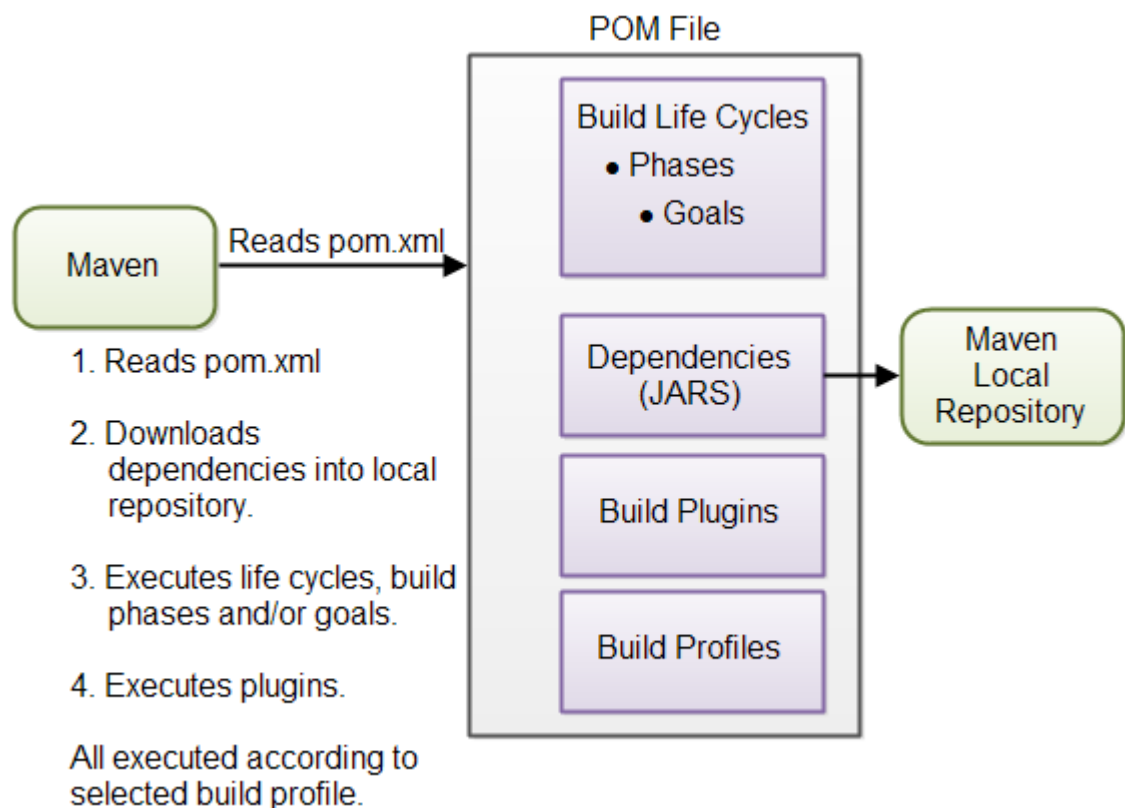
- Directorio **resources**: ficheros de configuración y recursos. Por ejemplo, contiene los archivos `images/properties`.
Ambos directorios se añaden al `classpath`.
- **target**: **clases compiladas y artefactos generados** a partir del código fuente y del resto de ficheros del directorio `src`.
- **pom.xml**: POM (Project Object Model) es fichero con la descripción de los elementos necesarios para todo el ciclo de vida del proyecto: compilación, test, empaquetado, despliegue o instalación en el repositorio de la empresa.

Un archivo POM puede heredar de otros (elemento `relativePath`), por lo que puede ser difícil en esos casos conocer el POM cuando se ejecuta Maven. **El POM resultado de la herencia se llama “POM efectivo”.**

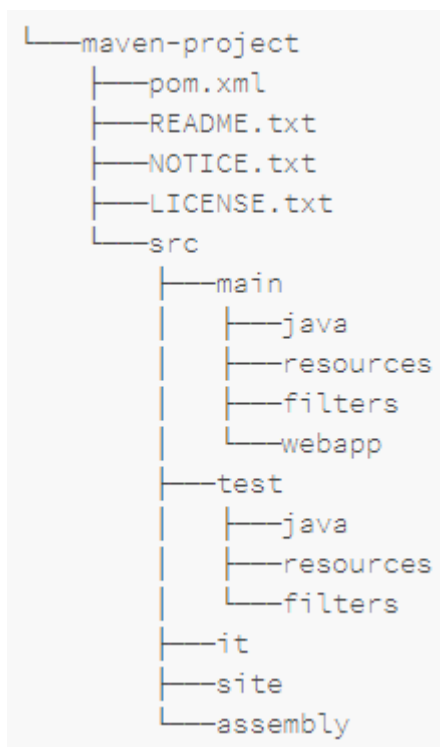
Las órdenes Maven empiezan con el comando ***mvn***, por lo que, por ejemplo, para ver el POM efectivo podría escribirse:

\$mvn help:effective-pom

El archivo POM está localizado en el directorio raíz del proyecto. En la siguiente imagen podemos ver cómo Maven trabaja con un archivo POM.



La estructura de directorios más detallada puede verse en la siguiente imagen:



Detalles

src/main/java	Código fuente de la aplicación y bibliotecas
src/main/resources	Recursos de la aplicación / biblioteca
src/main/filters	Archivos de filtro de recursos
src/main/webapp	Código fuente de aplicaciones web
src/test/java	Código fuente de prueba
src/test/resources	Recursos de prueba
src/test/filters	Archivos de filtro de recursos de prueba
src/it	Pruebas de integración (principalmente para <i>plugins</i>)
src/assembly	Descriptores de ensamblaje para empaquetar binarios.
src/site	Sitio
LICENSE.txt	Licencia del proyecto
NOTICE.txt	Avisos y atribuciones requeridas por las bibliotecas de las que depende el proyecto
README.txt	Fichero “readme” con información del proyecto.

I. Directorio raíz

- **/pom.xml** – define las **dependencias y los módulos necesarios** durante el ciclo de vida de compilación de un proyecto Maven. Es el archivo principal.
- **/LICENSE.txt** – información de **licencia del proyecto**.
- **/README.txt** – **resumen e información** sobre el proyecto.
- **/NOTICE.txt** – información sobre **bibliotecas de terceros utilizadas en el proyecto**.
- **/src/main** – contiene el **código fuente y los recursos** que se convierten en parte del artefacto.
- **/src/test** – contiene todo el **código y los recursos de prueba**
- **/src/it** – generalmente reservado para las **pruebas de integración** utilizadas por el plugin *Maven Failsafe*
- **/src/site** – **documentación del sitio** creada con el plugin *Maven Site*
- **/src/assembly** – configuración de ensamblaje para **empaquetar binarios**.

II. Directorio /src/main

Es el directorio más importante de un proyecto de Maven. Todo lo que se supone que es parte de del artefacto *[*, ved nota]*, ya sea un archivo *jar* o *war*, se guarda en este directorio.

- **src/main/java**: código fuente de Java para el artefacto
- **src/main/resources**: archivos de configuración, *internacionalización*, archivos de configuración del entorno y configuraciones XML, entre otros.
- **src/main/webapp**: para aplicaciones web, contiene recursos como JavaScript, CSS, archivos HTML, plantillas de visualización e imágenes.
- **src/main/filtros**: contiene archivos que inyectan valores en las propiedades de configuración en la carpeta de recursos durante la fase de compilación

[*] Un **artefacto Maven** es un archivo que se utiliza durante el desarrollo o la ejecución de un programa. Los tipos más comunes de artefactos son archivos JAR y WAR. Los artefactos se identifican por:

- **groupId**: se utiliza para agrupar artefactos relacionados que suelen utilizar un nombre de dominio invertido (como com.pepinho.web).
- **artifactId**: un nombre que identifica el artefacto en el *groupId*. No puede haber más de un artefacto con la misma combinación de **groupId** y **artifactId**. Viene siendo el **nombre del proyecto**.
- **version**: una versión de software normalmente en representada de con la estructura: *<mayor>.<menor>.<incremental>-<calificador>* (por ejemplo, 2.0.3-beta1).

La combinación de los valores de la **groupId**, **artifactId** y **versión** de un artefacto dado se conoce como las **coordenadas Maven** del artefacto.

III. Directorio `/src/test`

El directorio `src/test` es el lugar donde residen las pruebas de cada componente de la aplicación. **Ninguno de estos directorios o archivos se convertirá en parte del artefacto.** Similar al `main`:

- `src/test/java`: código fuente de Java para pruebas
- `src/test/resources` : archivos de configuración y otros utilizados por las pruebas.
- `src/test/filters`: contiene archivos que inyectan valores en las propiedades de configuración en la carpeta de recursos durante la fase de prueba

- a) Descarga la última versión de **Maven** (<http://maven.apache.org/download.cgi>) e instálalo en la máquina virtual con Ubuntu 20.04 LTS. *Puede ser la máquina virtual empleada en anteriores tareas, pero recomendaría que se hiciese en la clonada para evitar conflictos con otros servidores de aplicaciones y demás software empleado en tareas previas. También puedes hacer la instalación de nuevo desde cero. Importante: recuerda los prerequisites (Java instalado y variables de entorno PATH, JAVA_HOME o la nueva M2_HOME).*

B) INSTALACIÓN DE MAVEN

I. Instalación de *Maven* desde el repositorio

Para facilitar el trabajo, lo haremos inicialmente desde el repositorio, por medio de la utilidad “apt”:

```
sudo apt update
sudo apt install maven
```

```
pepecalo@totoro:~$ sudo apt update
[sudo] contraseña para pepecalo:
Obj:1 http://archive.ubuntu.com/ubuntu focal InRelease
pepecalo@totoro:~$ sudo apt install maven
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
```

Comprobación de la versión (3.6.3):

```
$mvn -version
```

```
pepecalo@totoro:~$ mvn -version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 11.0.9.1, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "5.4.0-54-generic", arch: "amd64", family: "unix"
pepecalo@totoro:~$
```

II. Instalación de *Maven* desde los binarios

La última versión estable (en la actualidad) de *maven* es la versión 3.6.3, dicha versión puede descargarse desde la página oficial:

<http://maven.apache.org/download.cgi> (<https://apache.brunneis.com/>)

La lista completa de *mirrors* puede consultarse en: <https://www.apache.org/mirrors/>

A. Prerrequisitos

- **JDK (Java Development Kit):** Maven 3.3+ requiere JDK 1.7 o superior para poder ser ejecutado, aunque existen estrategias para que funcione contra la versión 1.3 u otras (<http://maven.apache.org/guides/mini/guide-using-toolchains.html>)
- **Memoria:** no hay requisitos mínimos de memoria.
- **Disco:** aproximadamente 10MB es suficiente para la instalación. Como mantiene un **repositorio local**, el tamaño del repositorio dependerá de uso, pero se recomienda que sea, al menos, de 500MB.
- **Sistema operativo:** los scripts de arranque están disponibles para distintos Shell y archivos bat de Windows, por lo que es **multiplataforma** (en donde exista máquina virtual de Java).

Comprobamos la versión de java instalada (si no está instalada habría que instalarla):

```
sudo apt update
sudo apt install default-jdk
```

```
java -version
```

```
pepecalo@totoro:~$ java -version
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)
pepecalo@totoro:~$
```

B. Instalación

Descargamos maven en el directorio temporal, /tmp, por ejemplo:

```
wget https://apache.brunneis.com/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz -P /tmp
```

```
pepecalo@totoro:~$ wget https://ftp.cixug.es/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz -P /tmp
--2021-01-29 10:59:32-- https://ftp.cixug.es/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
Resolviendo ftp.cixug.es (ftp.cixug.es)... 193.144.61.75
Conectando con ftp.cixug.es (ftp.cixug.es)[193.144.61.75]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 9506321 (9,1M) [application/x-gzip]
Guardando como: "/tmp/apache-maven-3.6.3-bin.tar.gz"

apache-maven-3.6.3- 100%[======>] 9,07M 30,8MB/s en 0,3s

2021-01-29 10:59:32 (30,8 MB/s) - "/tmp/apache-maven-3.6.3-bin.tar.gz" guardado [9506321/9506321]

pepecalo@totoro:~$
```

Extraemos el archivo en el directorio /opt:

```
sudo tar xf /tmp/apache-maven-3.6.3-bin.tar.gz -C /opt
```

```
pepecalo@totoro:~$ sudo tar xf /tmp/apache-maven-3.6.3-bin.tar.gz -C /opt
pepecalo@totoro:~$
```

Para poder gestionar la actualización de versiones, lo mejor es **crear un enlace simbólico** a un directorio **/opt/maven**. Así, cualquier cambio de versión sólo implicaría un cambio en el enlace. Es una técnica muy común cuando se instala software desde elementos compilados o código fuente:

```
sudo ln -s /opt/apache-maven-3.6.3 /opt/Maven
```

```
pepecalo@totoro:~$ sudo ln -s /opt/apache-maven-3.6.3/ /opt/maven
pepecalo@totoro:~$
```

Un cambio de versión implica únicamente descomprimir la nueva versión y cambiar el enlace simbólico.

```
pepecalo@totoro:~$ ls /opt
apache-maven-3.6.3  maven
pepecalo@totoro:~$
```

C. Configuración de las variables de entorno

Se precisa configurar las variables de entorno, por lo que mediante un editor de texto crearemos un archivo para guardar las variables de entorno, **maven.sh**, en el directorio **/etc/profile.d/**.

```
sudo nano /etc/profile.d/maven.sh
```

```
GNU nano 4.8 /etc/profile.d/maven.sh
export JAVA_HOME=/usr/lib/jvm/default-java
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

```
export JAVA_HOME=/usr/lib/jvm/default-java
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

Así se cargará el script con el arranque del Sistema. Debemos darle permiso de ejecución y cargar las variables de entorno con “source”:

```
sudo chmod +x /etc/profile.d/maven.sh
source /etc/profile.d/maven.sh
```

```
pepecalo@totoro:~$ sudo chmod +x /etc/profile.d/maven.sh
pepecalo@totoro:~$ source /etc/profile.d/maven.sh
pepecalo@totoro:~$
```

D. Comprobación de la instalación desde los binarios

```
mvn -version
```

```
pepecalo@totoro:~$ mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /opt/maven
Java version: 11.0.9.1, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "5.4.0-54-generic", arch: "amd64", family: "unix"
pepecalo@totoro:~$
```

C) CREACIÓN DE UN PROYECTO MAVEN

Aunque no se ha indicado qué tipo de proyecto se debe realizar, por motivos obvios, crearemos un **proyecto Web Maven**.

Muchos IDE permiten haber un proyecto fácilmente (Eclipse, Netbeans,...) pero en este apartado se trata de aprender a **realizarlo manualmente**, a partir de las herramientas que dispone Maven.

Creación de una aplicación Web

Para crear una aplicación Web Java se puede emplear el *plugin/arquetipo/plantilla*: **maven-archetype-webapp**. Por ejemplo, en modo no interactivo (podría hacerse que preguntase cada uno de los elementos de configuración del proyecto):

```
mvn archetype:generate
-DgroupId=com.pepinho.web.daw
-DartifactId=tarea3.web
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

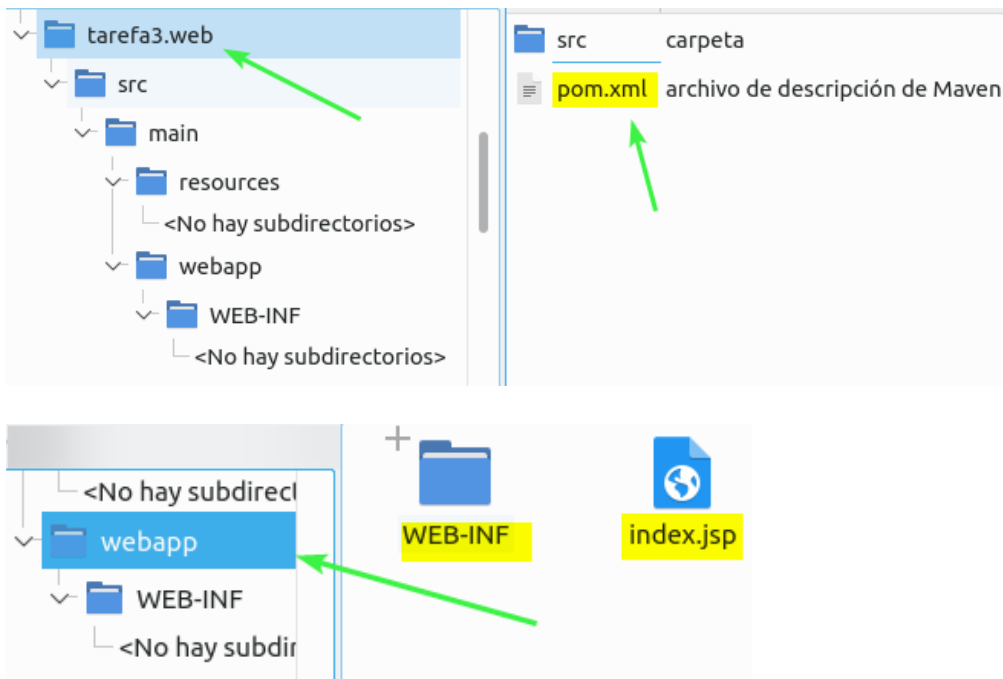
En donde:

```
mvn archetype:generate
-DgroupId={paquete del proyecto}
-DartifactId={nombre del proyecto}
-DarchetypeArtifactId={plantilla maven}
-DinteractiveMode=false
```

```
pepecalo@totoro:~/web$ mvn archetype:generate -DgroupId=com.pepinho.web.daw -D
artifactId=tarefa3.web -DarchetypeArtifactId=maven-archetype-webapp -Dinteract
iveMode=false
```

```
[INFO] Parameter: package, Value: com.pepinho.web.daw
[INFO] Parameter: groupId, Value: com.pepinho.web.daw
[INFO] Parameter: artifactId, Value: tarefa3.web
[INFO] Parameter: packageName, Value: com.pepinho.web.daw
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/pepecalo/web/tar
efa3.web
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.750 s
[INFO] Finished at: 2021- 21:05:08+01:00
[INFO] -----
pepecalo@totoro:~/web$
```

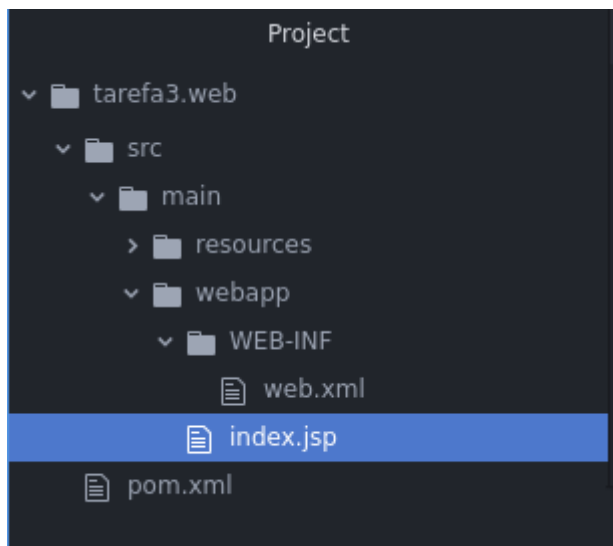
Puede verse la jerarquía de directorios:



Una vez compilado se generarían los correspondientes directorios “target” ...
El archivo POM generado queda como sigue:

```
<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>com.pepinho.web.daw</groupId>
  <artifactId>tarefa3.web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tarefa3.web Maven Webapp</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>
  <build>
    <finalName>tarefa3.web</finalName>
  </build>
</project>
```

Además, ha creado una página “/src/webapp/index.jsp” (con el famoso “¡Hola Mundo!”) y el archivo de la aplicación “/src/webapp/WEB-INF/web.xml”



El archivo **pom.xml** generado podrá ser usado como plantilla del proyecto:

```
<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>com.pepinho.web.daw</groupId>
  <artifactId>accesoDatos.web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Acceso a Datos Maven Webapp</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>
  <build>
    <finalName>accesoDatos.web</finalName>
  </build>
</project>
```

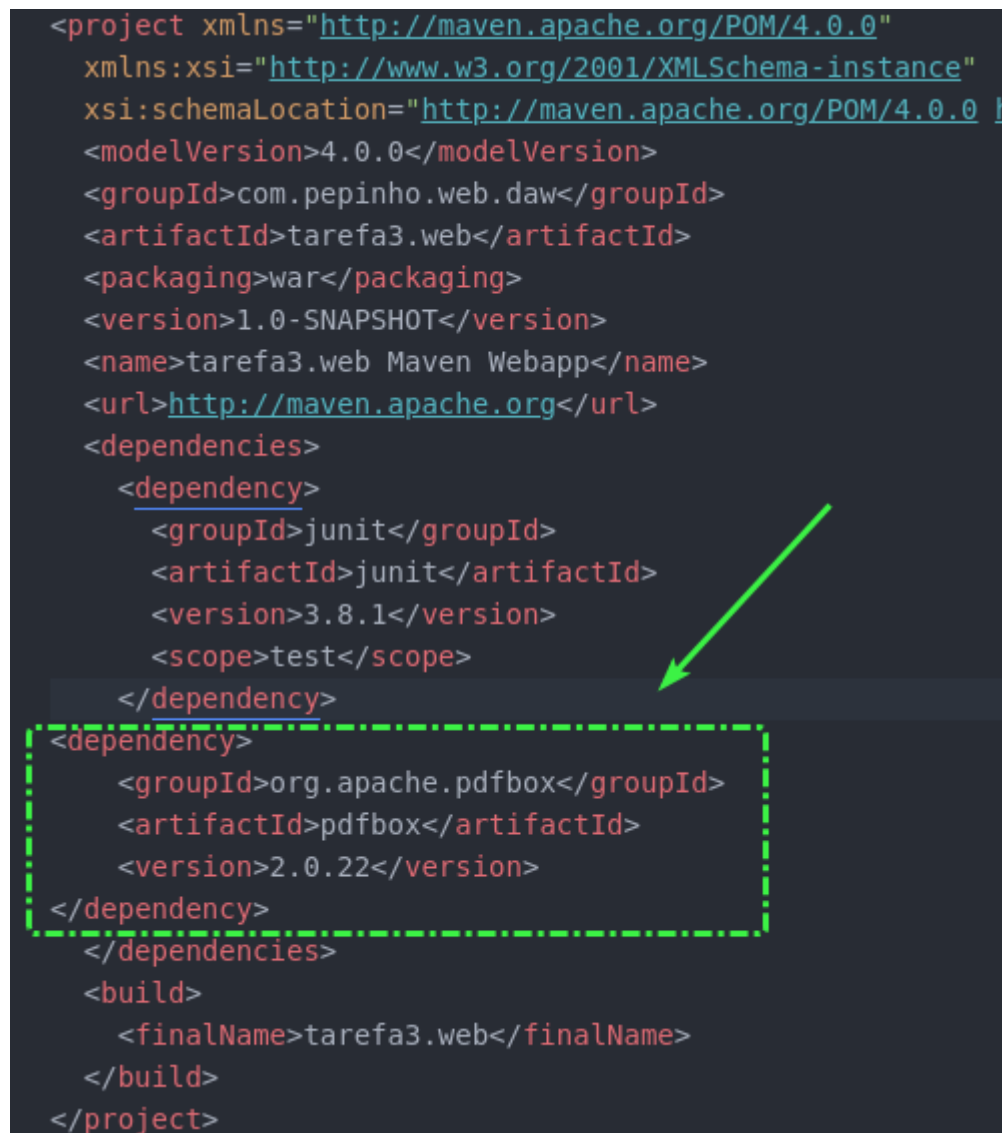
En el proyecto creado podemos ver que añade una dependencia, JUnit, que es el conjunto de bibliotecas Java para hacer pruebas unitarias de aplicaciones Java.

Añadir dependencias es tan sencillo con incorporarla en el correspondiente fichero pom.xml, por ejemplo, podríamos añadir las **bibliotecas para trabajar con PDF** y tablas u otras, directamente desde el repositorio:

<https://mvnrepository.com/artifact/org.apache.pdfbox>

```
<dependency>
  <groupId>org.apache.pdfbox</groupId>
  <artifactId>pdfbox</artifactId>
  <version>2.0.22</version>
</dependency>
```

<https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox/2.0.22>



```
<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.pepinho.web.daw</groupId>
  <artifactId>tarefa3.web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tarefa3.web Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.pdfbox</groupId>
      <artifactId>pdfbox</artifactId>
      <version>2.0.22</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>tarefa3.web</finalName>
  </build>
</project>
```

Existen muchos modos de gestionar las dependencias del proyecto:

<https://maven.apache.org/plugins/maven-dependency-plugin/usage.html>

La jerarquía de dependencias del proyecto se pueden ver con:

```
mvn dependency:tree
```

```
[INFO] com.pepinho.web.daw:tarefa3.web:war:1.0-SNAPSHOT
[INFO] +- junit:junit:jar:3.8.1:test
[INFO] \- org.apache.pdfbox:pdfbox:jar:2.0.22:compile
[INFO]     +- org.apache.pdfbox:fontbox:jar:2.0.22:compile
[INFO]     \- commons-logging:commons-logging:jar:1.2:compile
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 36.295 s
[INFO] Finished at: 2021-02-03T10:56:27+01:00
[INFO] -----
pepecalo@totoro:~/web/tarefa3.web$
```