

# PROYECTOS JAVA CON *Maven*<sup>TM</sup>

gestiona de manera practica y profesional tus aplicaciones



Juan Vladimir  
@juanvladimir13



**Maven** es una herramienta de software para la gestión y construcción de proyectos [Java](#)

Tiene un modelo de configuración de construcción más simple, basado en un formato [XML](#)

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

# Tus propios repositorios para un trabajo distribuido o local

El **motor** incluido en su **núcleo** puede dinámicamente descargar plugins de un repositorio.

Provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache, etc.

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.



# Preparando el entorno de desarrollo

## Tareas:

1. Descargar maven
2. Verificar requisitos mínimos para usar maven
3. Instalación y configuración de maven
  - 3.1. Configurar directorio local del **repository**
4. Seleccionar IDE favorito
5. Integrar maven al IDE

# 1. Descargar

# Maven™

## URL

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

<https://www-eu.apache.org/dist/maven/maven-3/>



# Apache Maven Project

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

## 2. Verificar requisitos mínimos

### **Java Development Kit (JDK)**

Maven 3.3+ require JDK 1.7 or above to execute

### **Disk**

Approximately 10MB is required for the Maven installation itself. In addition to that, additional disk space will be used for your local Maven repository.

### **Operating System**

No minimum requirement.

Start up scripts are included as shell scripts and Windows batch files

### 3. Instalación y configuración de



Descomprimir el archivo descargado

#### **Windows:**

Agregar JDK a las variables de entorno del sistema:

**Variable name:** JAVA\_HOME

**Variable value:** Java\jdk1.x.0\bin

Agregar al PATH

**Variable name:** Path

**Variable value:** Java\jdk1.x.0\bin; apache-maven-3.x.x\bin



### 3. Instalación y configuración de

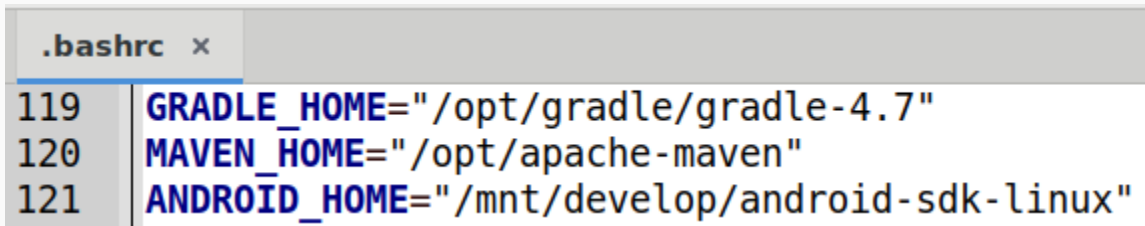


#### GNU/Linux

- ✓ Descomprimir el archivo descargado y agregar los permisos necesarios.
- ✓ Crear un acceso directo

**sudo ln -s `path_directory_maven/bin/mvn` `/usr/local/bin/`**

- ✓ Editar el archivo `/home/user/.bashrc` y agregar la variable de entorno  
**MAVEN\_HOME="/path\_directory\_maven"**

A screenshot of a text editor window showing the contents of the .bashrc file. The window title is ".bashrc x". The text is as follows:

```
119 GRADLE_HOME="/opt/gradle/gradle-4.7"
120 MAVEN_HOME="/opt/apache-maven"
121 ANDROID_HOME="/mnt/develop/android-sdk-linux"
```

#### Confirmar la instalación de maven

**mvn -v**

**mvn --version**

## 3.1. Configurar directorio local del repository

**PATH\_to\_maven** : Dirección completa donde descomprimos la carpeta de maven

**Crear carpeta**

PATH\_to\_maven/**repository**

**Editar el archivo**

PATH\_to\_maven/conf/**settings.xml**

```
<!-- localRepository
| The path to the local repository maven will use to store artifacts.
| Default: ${user.home}/.m2/repository
-->
<localRepository>PATH_to_maven/repository</localRepository>
```

## 4. Selecciona tu IDE o editor de texto favorito



## 5. Integrando maven a



### Paso 1

Windows -> Preferences -> Maven

### Paso 2

**Installations** -> Add

*Installations home* -> Directory -> ( [PATH\\_to\\_maven](#) )

**Archetypes** -> Add local catalog

*Catalog file* -> Browse ([archetype-catalog.xml](#))

### User settings

*Global settings*

[PATH\\_to\\_maven](#)/conf/[settings.xml](#)

*User settings*

[PATH\\_to\\_maven](#)/conf/[settings.xml](#)

*Local repository*

[PATH\\_to\\_maven](#)/[repository](#)



Tools -> Options -> java -> maven

**Execution** -> Maven Home -> browse (***PATH\_to\_maven***)

# Creando proyectos con **Maven**<sup>TM</sup>

1. Estructura de un proyecto
2. Crear un proyecto estándar de java
3. Proyecto con múltiples módulos
  - 3.1. Generando módulos para el proyecto padre
  - 3.2. Resultado del proyecto creado

# 1. Estructura de un proyecto

Al utilizar **maven** varias estructuras de proyectos utiliza números para identificar los proyectos:

Código	Descripción
<b>1379</b> <b>org.apache.maven.archetypes:maven-archetype-quickstart</b>	An archetype which contains a sample Maven project
<b>1673</b> <b>org.codehaus.mojo.archetypes:pom-root</b>	Root project archetype for creating multi module projects

**groupId:** Típicamente aquí se pone el nombre de tu empresa u organización, todos los proyectos con ese groupId pertenecen a una sola empresa.

**artifactId:** Es el nombre de tu proyecto.

**version:** Número de versión de tu proyecto.

**package:** Paquete base donde irá tu código fuente

## 2. Crear un proyecto estándar de java

Generando un nuevo proyecto de java desde la terminal:

```
mvn archetype:generate
```

Se despliega un programa interactivo para introducir datos del proyecto:

Opción	Ejemplo
groupId	com.empresa
artifactId	proyecto
version	1.3
package	com.empresa.paquete



### 3. Proyecto con múltiples módulos

Generando un nuevo proyecto con múltiples módulos desde la terminal:

```
mvn archetype:generate
```

Se despliega un programa interactivo para introducir datos del proyecto:

Introducir el código del proyecto **1673**

*1673 es el código de **Root project archetype for creating multi module projects***

## 3.1. Generando módulos para el proyecto padre

Generando un nuevo módulo desde la terminal:

*Ingresar al directorio del proyecto creado*

`cd proyecto_padre`

`mvn archetype:generate`

Introducir el número **1379**

*1379 es el código de **An archetype which contains a sample Maven project***

*Repetir este procedimiento según la cantidad de módulos a crear*

## 3.2. Resultado del proyecto creado

### Estructura de directorios

```
juanvladimir13@coberton:proyecto_padre$ tree
```

```
.
├── pom.xml
└── ventas
    ├── pom.xml
    └── src
        ├── main
        │   ├── java
        │   │   ├── com
        │   │   │   ├── curcusi
        │   │   │   │   ├── ventas
        │   │   │   │   │   App.java
        │   └── test
        │       ├── java
        │       │   ├── com
        │       │   │   ├── curcusi
        │       │   │   │   ├── ventas
        │       │   │   │   │   AppTest.java
```

### Archivo pom.xml del proyecto\_padre

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.curcusi</groupId>
<artifactId>proyecto_padre</artifactId>
<packaging>pom</packaging>
<version>1.0-SNAPSHOT</version>
<name>proyecto_padre Maven Multi Project</name>
<url>http://maven.apache.org</url>
<modules>
  <module>ventas</module>
</modules>
```

# Gestionando el proyecto java con **Maven**<sup>TM</sup>

1. Agregar dependencias del proyecto
2. Comandos más utilizados
3. Otros comandos
4. Ciclo de vida

# 1. Agregar dependencias del proyecto

Repositorio oficial de proyectos maven <https://mvnrepository.com/>

Localizar la librería o framework a adjuntar al proyecto ejemplo:

<https://mvnrepository.com/artifact/junit/junit-dep/4.10>

```
<!-- https://mvnrepository.com/artifact/junit/junit-dep -->  
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit-dep</artifactId>  
  <version>4.10</version>  
</dependency>
```

Agregar este contenido en el archivo **pom.xml** del proyecto dentro de:

```
<dependencies>  
</dependencies>
```

## 2. Comandos más utilizados

***compile***: Genera los ficheros **\*.class** compilando los fuentes **\*.java**

***test***: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.

***package***: Genera el fichero **\*.jar** con los **\*.class** compilados

***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.

***clean***: Elimina todos los **\*.class** y **\*.jar** generados. Después de este comando se puede comenzar un compilado desde cero.

### 3. Otros comandos

También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, (no tienen que ser siempre realizadas). Estas metas son:

***assembly***: Genera un fichero **\*.zip** con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero **xml** que se debe incluir en ese zip.

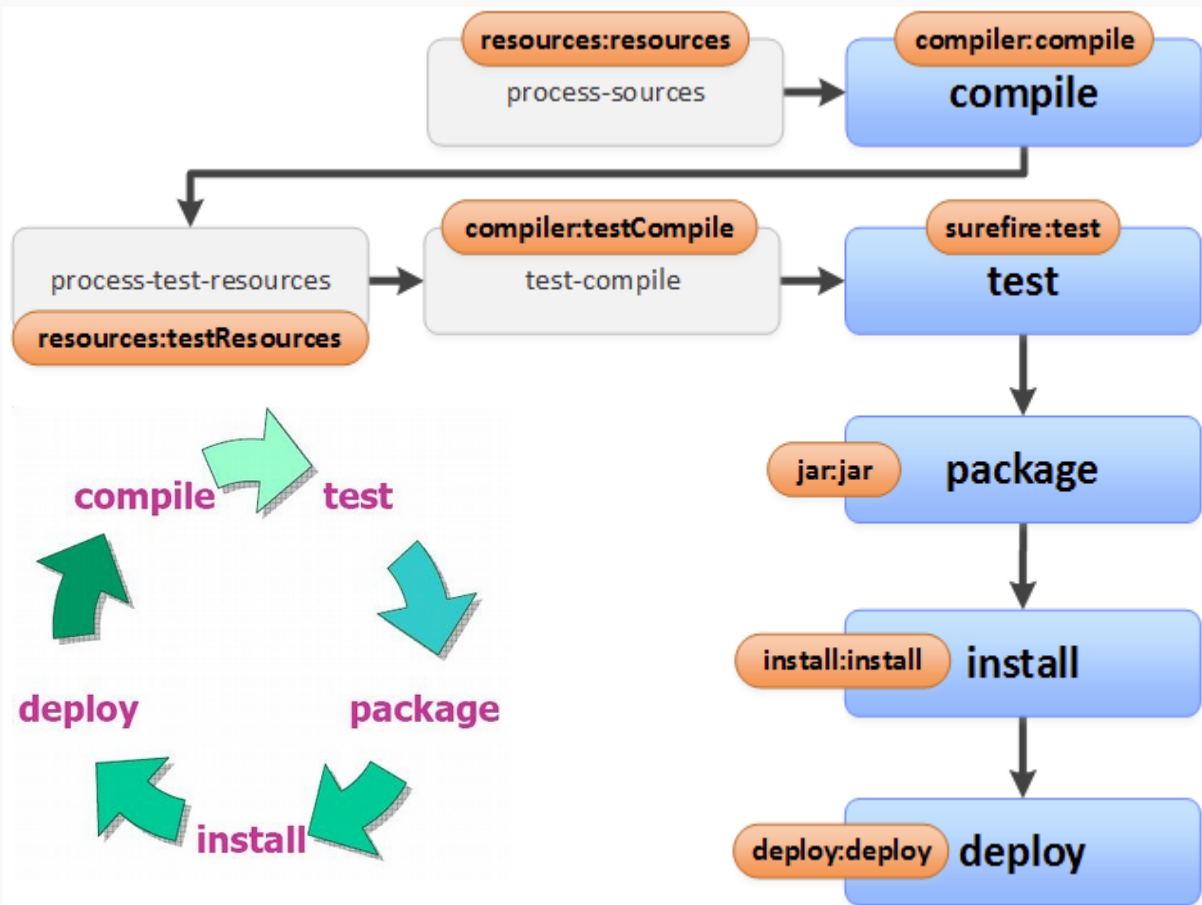
***deploy***: Copia el fichero **\*.jar** a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

***site***: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero **pom.xml** y ficheros **.apt** separados.

***site-deploy***: Sube el sitio web al servidor que hayamos configurado.

## 4. Ciclo de vida

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





# Properties and Plugins

1. Properties
2. Plugins
3. Plugins pluginManagement

# 1. Properties

Especificar versión de JDK a utilizar en el proyecto

```
<properties>
```

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

```
  <maven.compiler.source>1.8</maven.compiler.source>
```

```
  <maven.compiler.target>1.8</maven.compiler.target>
```

```
</properties>
```

## 2. Plugins

### Copiar las librerías utilizadas

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>3.0.1</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directory}/lib</outputDirectory>
        <overwriteReleases>>false</overwriteReleases>
        <overwriteSnapshots>>true</overwriteSnapshots>
        <overwriteIfNewer>>true</overwriteIfNewer>
      </configuration>
    </execution>
  </executions>
</plugin>
```

## 2. Plugins pluginManagement

Especificar mainClass del proyecto

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.0.2</version>
  <configuration>
    <archive>
      <index>true</index>
      <manifest>
        <addClasspath>true</addClasspath>
        <classpathPrefix></classpathPrefix>
        <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
        <addDefaultSpecificationEntries>true</addDefaultSpecificationEntries>
        <mainClass>com.empresa.App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

YA ERES TODO UN EXPERTO !!! :-)



# Contactos y sugerencias



<https://www.facebook.com/juanvladimir13>



<https://twitter.com/juanvladimir13>



<https://www.linkedin.com/in/juanvladimir13>



<https://www.instagram.com/juanvladimir13>



@juanvladimir13



[https://www.youtube.com/channel/UCk9R\\_mLgbcENR\\_BPF9M9asQ/videos](https://www.youtube.com/channel/UCk9R_mLgbcENR_BPF9M9asQ/videos)



juanvladimir13@gmail.com



@juanvladimir13



<http://juanvladimir13.wordpress.com>



<http://juanvladimir13.blogspot.com>



<https://github.com/juanvladimir13>



<https://bitbucket.org/juanvladimir13>



<https://www.facebook.com/groups/nucleolinux.uagrm>



<https://github.com/nucleolinux-uagrm>



[https://t.me/nucleolinux\\_uagrm](https://t.me/nucleolinux_uagrm)

