

Manual Completo de Apache Maven

Gestión y Automatización de Proyectos Java

Francisco José Teurel Teruel

27/01/2026

Índice

Introducción	4
¿Qué es Apache Maven?	4
Propósitos y Beneficios Clave	4
Limitaciones y Consideraciones	5
Historia y Evolución	5
Instalación	5
Requisitos Previos	6
Instalación en Windows	6
Instalación en macOS	6
Instalación en Linux	6
Configuración de Proxies y Repositorios	6
Configuración Inicial	7
JAVA_HOME	7
PATH	7
MAVEN_HOME y M2_HOME	7
Repositorio Local	7
Estructura de Proyecto	8
Estructura Estándar	8
Descripción de Carpetas	8
Ventajas de la Convención	8
POM - Project Object Model	9
Elementos Principales del POM	9
Ejemplo de POM Mínimo	10
Coordenadas Maven	10

Gestión de Dependencias	11
Declaración de Dependencias	11
Alcances de Dependencia (scope)	11
Repositorios Maven	11
Resolución de Conflictos y Dependencias Transitivas	12
Buenas Prácticas	12
Repositorios	12
Repositorio Local	12
Repositorio Central	12
Repositorios Remotos/Privados	12
Ciclo de Vida	12
Ciclos de Vida Principales	13
Fases del Ciclo de Vida Default	13
Ciclos Adicionales	14
Plugins y Goals	14
Plugins y Goals	14
Configuración de Plugins	15
Comandos Básicos	16
Arquetipos	17
Creación de un Proyecto Java Básico	17
Importación en IDEs	17
Perfiles y Configuración	17
Perfiles	17
Propiedades	18
settings.xml	18
Integración con IDEs y CI/CD	18
IDEs	18
CI/CD	19
Buenas Prácticas	19
Buenas Prácticas de Maven	19
Maven vs Gradle vs Ant	20
Comparativa:	20
¿Cuándo elegir Maven?	20
¿Cuándo elegir Gradle?	20
Proyectos Multi-Módulo	21
Estructura típica:	21
Gestión de Versiones	21

Testing y Calidad	22
Surefire y Failsafe	22
Configuración básica:	22
JaCoCo y Reports	22
Empaquetado y Despliegue	23
Empaquetado	23
Despliegue	23
Resolución de Problemas	24
Flags Útiles de Maven	24
Problemas Frecuentes	24
Rendimiento y Optimización	24
Generación de Documentación	24
Manual Paso a Paso	25
1. Instalación de Maven	25
2. Creación de un Proyecto	25
3. Estructura del Proyecto	25
4. Compilación	25
5. Gestión de Dependencias	25
6. Ejecución de Pruebas	25
7. Empaquetado	25
8. Instalación en Repositorio Local	25
9. Despliegue en Repositorio Remoto	26
10. Generación de Documentación	26
11. Uso de Perfiles	26
12. Comandos Comunes y Flags	26
Guía Rápida	27
Instala Java (JDK) y Maven.	27
Crea un proyecto básico:	27
Compila el proyecto:	27
Ejecuta pruebas:	27
Empaqueta el proyecto:	27
Instala en el repositorio local:	27
Agrega dependencias	27
Comandos útiles:	27
Importa el proyecto en tu IDE	27
Consulta la documentación oficial	27
Ejemplo Práctico	28
Crear el proyecto:	28
Compilar y probar:	28
Añadir una dependencia (ejemplo: Gson)	28

Empaquetar:	28
Ejecutar la aplicación (si tiene método main):	28
Recursos	29
Conclusión	29

Introducción

Apache Maven es una de las herramientas más influyentes y utilizadas en el ecosistema Java para la gestión y automatización de proyectos de software. Su adopción masiva en entornos empresariales y de desarrollo profesional se debe a su capacidad para simplificar procesos complejos, estandarizar la estructura de los proyectos y automatizar tareas repetitivas como la compilación, el testing, el empaquetado y el despliegue de aplicaciones.

Este manual exhaustivo, redactado en español, tiene como objetivo proporcionar una visión integral de Maven, desde sus fundamentos y arquitectura hasta su uso avanzado, incluyendo un manual de usuario paso a paso y una guía rápida para principiantes.

¿Qué es Apache Maven?

Apache Maven es una herramienta de gestión y automatización de proyectos de software, especialmente diseñada para proyectos Java, aunque también puede utilizarse con otros lenguajes. Su propósito principal es simplificar y estandarizar el proceso de construcción (build), gestión de dependencias, pruebas, empaquetado y despliegue de aplicaciones.

Maven se basa en el principio de “**convención sobre configuración**”, lo que significa que, siguiendo ciertas convenciones de estructura y nomenclatura, se reduce la necesidad de configuraciones manuales. Esto permite que los desarrolladores se centren en el código y la lógica de negocio, delegando en Maven la gestión de tareas repetitivas y la resolución de dependencias.

Propósitos y Beneficios Clave

- **Automatización del ciclo de vida del proyecto:** Desde la compilación hasta el despliegue, pasando por pruebas y generación de documentación.
- **Gestión centralizada de dependencias:** Descarga automática de bibliotecas y plugins desde repositorios remotos, evitando la gestión manual de archivos JAR.
- **Estandarización de la estructura del proyecto:** Facilita la colaboración y el mantenimiento, ya que todos los proyectos Maven comparten una estructura común.

- **Integración con herramientas de CI/CD e IDEs:** Soporte nativo para integración continua y compatibilidad con los principales entornos de desarrollo (IntelliJ IDEA, Eclipse, VS Code, NetBeans).
- **Extensibilidad mediante plugins:** Permite ampliar sus capacidades para cubrir necesidades específicas de cada proyecto.

Limitaciones y Consideraciones

- **Curva de aprendizaje inicial:** El uso de XML y la cantidad de conceptos pueden resultar abrumadores para principiantes.
- **Rigidez estructural:** La estandarización puede ser un obstáculo en proyectos con necesidades muy particulares.
- **Dependencia de conexión a Internet:** Para la descarga de dependencias y plugins, aunque se puede trabajar en modo offline si es necesario.

Historia y Evolución

Maven fue creado en 2002 por Jason van Zyl dentro de la Apache Software Foundation, como respuesta a la necesidad de una herramienta que unificara y simplificara la construcción de proyectos Java, superando las limitaciones de Apache Ant. Su nombre proviene del yidis y significa “experto” o “acumulador de conocimiento”, reflejando su objetivo de ser una fuente centralizada de gestión y buenas prácticas en el desarrollo de software.

A lo largo de los años, Maven ha evolucionado significativamente:

- **Maven 1.x:** Introdujo el modelo de objetos de proyecto (POM) y la gestión declarativa de dependencias.
- **Maven 2.x:** Mejoró la modularidad, la gestión de repositorios y la extensibilidad mediante plugins.
- **Maven 3.x:** Aumentó el rendimiento, la compatibilidad con proyectos multi-módulo y la integración con herramientas modernas de CI/CD.

Actualmente, Maven sigue siendo el estándar de facto en la industria Java, aunque han surgido alternativas como Gradle y SBT para necesidades más específicas o flexibles.

Instalación

La instalación de Maven es sencilla, pero requiere tener previamente instalado un Java Development Kit (JDK) compatible (Java 8 o superior).

Requisitos Previos

- **JDK instalado y configurado:** Verifica con `java -version` y `javac -version`.
- **Variables de entorno:** `JAVA_HOME` debe apuntar a la raíz del JDK, y el directorio `bin` de Maven debe estar en el `PATH`.

Instalación en Windows

1. Descarga la última versión estable de Maven desde la web oficial.
2. Descomprime el archivo ZIP en una ubicación fija, por ejemplo: `C:\apache-maven-3.9.12`.
3. **Configura las variables de entorno:**
 - `JAVA_HOME`: Apunta a la carpeta raíz del JDK (ejemplo: `C:\Program Files\Java\jdk-21`).
 - `MAVEN_HOME` (opcional): Apunta a la carpeta de Maven.
 - `PATH`: Añade `%MAVEN_HOME%\bin` o la ruta directa al directorio `bin` de Maven.
4. Abre una nueva terminal (CMD o PowerShell) y ejecuta `mvn -version` para verificar la instalación.

Instalación en macOS

- **Homebrew:** `brew install maven`
- **SDKMAN!:** `sdk install maven`
- **MacPorts:** `sudo port install maven3`

Verifica con `mvn -v`.

Instalación en Linux

- **Debian/Ubuntu:** `sudo apt update && sudo apt install maven`
- **Fedora/RedHat:** `sudo dnf install maven` o `sudo yum install maven`

Verifica con `mvn -v`.

Configuración de Proxies y Repositorios

Si tu red requiere proxy, configura el archivo `settings.xml` en `${MAVEN_HOME}/conf` o `${USER_HOME}/.m2/` con los datos del proxy en la sección `<proxies>`.

Configuración Inicial

Una correcta configuración de las variables de entorno es fundamental para el funcionamiento de Maven y la detección del JDK.

JAVA_HOME

1. Debe apuntar a la raíz del JDK, nunca al directorio `/bin` ni al JRE.
2. Ejemplo en Windows: `C:\Program Files\Java\jdk-21`
3. Verifica con `echo %JAVA_HOME%` y `java -version`.

PATH

Añade el directorio bin de Maven para poder ejecutar mvn desde cualquier ubicación.

- En **Windows**: `%MAVEN_HOME%\bin`
- En **Unix/macOS**: Añade `export PATH=$PATH:/ruta/a/apache-maven-3.9.12/bin` en tu `.bashrc` o `.zshrc`.

MAVEN_HOME y M2_HOME

Opcionales en versiones recientes, pero pueden ser útiles para algunas herramientas o scripts.

Repositorio Local

Por defecto, Maven almacena dependencias en `${USER_HOME}/.m2/repository`.

Puedes cambiar la ubicación editando `<localRepository>` en `settings.xml`.

Estructura de Proyecto

Maven impone una estructura de directorios estándar que facilita la colaboración y la integración con herramientas externas.

Estructura Estándar

```
mi-proyecto/  
  
|-- pom.xml  
|-- src/  
|   |-- main/  
|       |-- java/          # Código fuente principal  
|       |-- resources/     # Recursos (configuración, propiedades, etc.)  
|       |-- webapp/        # (solo para aplicaciones web) Archivos web  
|   |-- test/  
|       |-- java/          # Código fuente de pruebas  
|       |-- resources/     # Recursos de pruebas  
|-- target/               # Salida de la compilación (no versionar)
```

Descripción de Carpetas

Carpeta/Archivo	Descripción
pom.xml	Archivo central de configuración del proyecto.
src/main/java	Código fuente de la aplicación.
src/main/resources	Archivos de configuración, propiedades, etc.
src/main/webapp	Archivos web (HTML, JSP, etc.) para aplicaciones WAR.
src/test/java	Pruebas unitarias e integración.
src/test/resources	Recursos para pruebas.
target/	Archivos generados por la compilación (JAR, WAR, clases, reportes).

Ventajas de la Convención

- **Compatibilidad con IDEs:** Reconocimiento automático de la estructura.
- **Facilita la colaboración:** Cualquier desarrollador familiarizado con Maven puede integrarse rápidamente.
- **Automatización:** Plugins y herramientas externas funcionan sin configuraciones adicionales.

POM - Project Object Model

El POM (`pom.xml`) es el corazón de cualquier proyecto Maven. Es un archivo XML que describe la configuración, dependencias, plugins, perfiles y otros aspectos del proyecto.

Elementos Principales del POM

Elemento	Descripción	Ejemplo
<code><modelVersion></code>	Versión del modelo POM (siempre 4.0.0 para Maven 2/3)	4.0.0
<code><groupId></code>	Identificador único del grupo/organización	com.miempresa
<code><artifactId></code>	Nombre del proyecto (sin espacios)	mi-aplicacion
<code><version></code>	Versión del proyecto	1.0.0-SNAPSHOT
<code><packaging></code>	Tipo de empaquetado (jar, war, ear, pom)	jar
<code><name></code>	Nombre legible del proyecto	Mi Aplicación Demo
<code><url></code>	URL del proyecto	https://www.miempresa.com
<code><dependencies></code>	Lista de dependencias del proyecto	Ver sección 7
<code><build></code>	Configuración de construcción y plugins	Ver sección 10
<code><properties></code>	Propiedades personalizables	<code><java.version>17</java.version></code>
<code><modules></code>	Lista de módulos (para proyectos multi-módulo)	<code><module>api</module></code>
<code><parent></code>	Referencia a un POM padre (herencia)	Ver sección 16
<code><dependencyManagement></code>	Gestión centralizada de versiones	Ver sección 7.5
<code><profiles></code>	Configuraciones condicionales por entorno	Ver sección 13
<code><repositories></code>	Repositorios remotos adicionales	Ver sección 8
<code><pluginRepositories></code>	Repositorios para plugins	Similar a <code><repositories></code>
<code><reporting></code>	Configuración de reportes	Ver sección 17

Ejemplo de 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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ejemplo</groupId>
  <artifactId>demo-maven</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Coordenadas Maven

Las coordenadas (`groupId`, `artifactId`, `version`) identifican de forma única cualquier artefacto en el ecosistema Maven y son esenciales para la gestión de dependencias y la publicación en repositorios.

Gestión de Dependencias

La gestión de dependencias es uno de los pilares de Maven. Permite declarar bibliotecas externas en el POM y delegar en Maven la descarga, actualización y resolución de versiones.

Declaración de Dependencias

Las dependencias se definen en la sección `<dependencies>` del POM, especificando sus coordenadas y, opcionalmente, el alcance (scope):

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>33.1.0-jre</version>
</dependency>
```

Alcances de Dependencia (scope)

Scope	Descripción	Ejemplo de Uso
compile	Por defecto. Disponible en classpath de compilación y runtime.	Bibliotecas core del proyecto.
provided	Proveído por el contenedor o JDK en runtime. No se incluye en el empaquetado.	Servlet API, JAXB API.
runtime	Requerido solo en runtime, no para compilar.	Driver JDBC.
test	Solo disponible para compilación y ejecución de tests.	JUnit, Mockito.
system	Similar a provided, pero se referencia un JAR específico en el sistema de archivos.	Dependencias locales no en repositorios.
import	Solo usado en <code><dependencyManagement></code> para importar BOMs.	Spring Boot Dependencies BOM.

Repositorios Maven

- **Local:** Carpeta `.m2/repository` en el equipo del desarrollador.
- **Central:** Repositorio público mantenido por la comunidad Maven.

- **Remotos/Privados:** Servidores internos (Nexus, Artifactory) para dependencias corporativas o artefactos propios.

Resolución de Conflictos y Dependencias Transitivas

- **Dependencias transitivas:** Maven descarga automáticamente las dependencias de las dependencias.
- **Conflictos de versión:** Se resuelven por el principio de “nearest definition” (la versión más cercana en el árbol de dependencias).

Herramientas útiles:

- `mvn dependency:tree` - Visualiza el árbol de dependencias.
- `mvn dependency:analyze` - Detecta dependencias no utilizadas o faltantes.
- Exclusiones: `<exclusions>` para evitar dependencias no deseadas.

Buenas Prácticas

- Fijar versiones explícitas, evitar rangos abiertos o LATEST.
- Centralizar versiones en `<dependencyManagement>` o propiedades.
- Usar BOMs para ecosistemas complejos (ej. Spring Boot).

Repositorios

Repositorio Local

- Ubicación por defecto: `${USER_HOME}/.m2/repository`.
- Almacena artefactos descargados y generados localmente.
- Se puede cambiar en `settings.xml` con `<localRepository>`.

Repositorio Central

- URL: `https://repo.maven.apache.org/maven2/`
- Fuente principal de dependencias públicas.

Repositorios Remotos/Privados

Herramientas como Nexus o Artifactory permiten gestionar artefactos internos, controlar versiones y aplicar políticas de seguridad. Se configuran en el POM (`<repositories>`) o en `settings.xml` (credenciales, mirrors, proxies).

Ciclo de Vida

El ciclo de vida de Maven define una secuencia ordenada de fases por las que pasa un proyecto durante su construcción. Cada fase puede ejecutar uno o varios

“goals” (objetivos) proporcionados por plugins.

Ciclos de Vida Principales

- **default:** Construcción y despliegue del artefacto.
- **clean:** Limpieza de artefactos generados previamente.
- **site:** Generación de documentación y reportes del proyecto.

Fases del Ciclo de Vida Default

Fase	Descripción
<code>validate</code>	Valida que el proyecto sea correcto y toda la información esté disponible.
<code>initialize</code>	Inicializa el estado del build, como configurar propiedades o crear directorios.
<code>generate-sources</code>	Genera código fuente a incluir en la compilación.
<code>process-sources</code>	Procesa el código fuente (ej. filtrar valores).
<code>generate-resources</code>	Genera recursos a incluir en el paquete.
<code>process-resources</code>	Copia y procesa recursos en el directorio de destino, listo para empaquetado.
<code>compile</code>	Compila el código fuente del proyecto.
<code>process-classes</code>	Realiza post-procesamiento en los archivos <code>.class</code> generados.
<code>generate-test-sources</code>	Genera código fuente de pruebas.
<code>process-test-sources</code>	Procesa el código fuente de pruebas.
<code>generate-test-resources</code>	Genera recursos para pruebas.
<code>process-test-resources</code>	Copia y procesa recursos de pruebas en el directorio de destino de pruebas.
<code>test-compile</code>	Compila el código fuente de pruebas.
<code>process-test-classes</code>	Realiza post-procesamiento en los archivos <code>.class</code> de pruebas.
<code>test</code>	Ejecuta las pruebas usando un framework adecuado.
<code>prepare-package</code>	Realiza preparaciones necesarias antes del empaquetado.
<code>package</code>	Empaqueta el código compilado en su formato distribuible (JAR, WAR, etc.).
<code>pre-integration-test</code>	Realiza acciones requeridas antes de ejecutar pruebas de integración.

Fase	Descripción
<code>integration-test</code>	Procesa y despliega el paquete en un entorno donde se ejecutan pruebas de integración.
<code>post-integration-test</code>	Realiza acciones requeridas después de ejecutar pruebas de integración.
<code>verify</code>	Ejecuta chequeos para verificar que el paquete es válido y cumple criterios de calidad.
<code>install</code>	Instala el paquete en el repositorio local, para usarlo como dependencia en otros proyectos locales.
<code>deploy</code>	Copia el paquete final al repositorio remoto para compartirlo con otros desarrolladores y proyectos.

Ciclos Adicionales

- **clean:** `pre-clean`, `clean`, `post-clean` - Elimina el directorio `target` y artefactos previos.
- **site:** `pre-site`, `site`, `post-site`, `site-deploy` - Genera y despliega la documentación del proyecto.

Plugins y Goals

Cada fase ejecuta uno o varios goals de plugins (ej. `maven-compiler-plugin:compile`, `maven-surefire-plugin:test`).

Los plugins pueden configurarse y personalizarse en el POM.

Plugins y Goals

Los plugins extienden las capacidades de Maven y permiten automatizar tareas específicas. Algunos de los más utilizados son:

Plugin	Goals Comunes	Descripción
<code>maven-compiler-plugin</code>	<code>compile</code> , <code>testCompile</code>	Compila el código fuente del proyecto.
<code>maven-surefire-plugin</code>	<code>test</code>	Ejecuta las pruebas unitarias.
<code>maven-failsafe-plugin</code>	<code>integration-test</code> , <code>verify</code>	Ejecuta pruebas de integración.
<code>maven-jar-plugin</code>	<code>jar</code>	Crea un archivo JAR.

Plugin	Goals Comunes	Descripción
maven-war-plugin	war	Crea un archivo WAR para aplicaciones web.
maven-install-plugin	install	Instala el artefacto en el repositorio local.
maven-deploy-plugin	deploy	Despliega el artefacto en un repositorio remoto.
maven-clean-plugin	clean	Limpia los archivos generados.
maven-site-plugin	site, deploy	Genera y despliega la documentación del sitio.
maven-dependency-plugin	tree, analyze, copy-dependencies	Utilidades para gestionar dependencias.
maven-resources-plugin	resources, testResources	Copia recursos a directorios de salida.
maven-release-plugin	prepare, perform	Automatiza el proceso de release.

Configuración de Plugins

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.12.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Comandos Básicos

La interfaz de línea de comandos de Maven es poderosa y flexible. A continuación, se resumen los comandos más comunes.

Comando	Descripción
<code>mvn clean</code>	Limpia el directorio target y artefactos previos.
<code>mvn compile</code>	Compila el código fuente del proyecto.
<code>mvn test</code>	Ejecuta las pruebas unitarias.
<code>mvn package</code>	Empaqueta el código compilado en su formato distribuible (JAR, WAR, etc.).
<code>mvn install</code>	Instala el paquete en el repositorio local.
<code>mvn deploy</code>	Copia el paquete final al repositorio remoto.
<code>mvn site</code>	Genera la documentación del sitio del proyecto.
<code>mvn archetype:generate</code>	Crea un nuevo proyecto desde un arquetipo.
<code>mvn dependency:tree</code>	Muestra el árbol de dependencias del proyecto.
<code>mvn help:effective-pom</code>	Muestra el POM efectivo (incluyendo herencia e interpolación).
<code>mvn -X</code>	Ejecuta Maven en modo debug (muestra logs detallados).
<code>mvn -T 4</code>	Ejecuta build con 4 hilos (paralelismo).
<code>mvn -o</code>	Ejecuta Maven en modo offline (sin descargar dependencias).
<code>mvn -DskipTests</code>	Omite la ejecución de pruebas (útil para builds rápidos).
<code>mvn -P<perfil></code>	Activa un perfil específico.

Arquetipos

Los arquetipos son plantillas predefinidas para crear proyectos con una estructura básica y dependencias iniciales.

Creación de un Proyecto Java Básico

```
mvn archetype:generate
  -DgroupId=com.ejemplo
  -DartifactId=demo-maven
  -DarchetypeArtifactId=maven-archetype-quickstart
  -DinteractiveMode=false
```

Parámetros Clave

- `groupId`: Dominio invertido o namespace de la organización.
- `artifactId`: Nombre del proyecto.
- `archetypeArtifactId`: Plantilla a utilizar (maven-archetype-quickstart para Java básico).
- `interactiveMode=false`: Evita preguntas interactivas.

Otros Arquetipos Comunes

- `maven-archetype-webapp`: Para aplicaciones web (WAR).
- `maven-archetype-archetype`: Para crear nuevos arquetipos personalizados.

Importación en IDEs

- **IntelliJ IDEA**: Abrir la carpeta del proyecto o el pom.xml y el IDE detecta automáticamente la estructura Maven.
- **Eclipse**: Importar como “Existing Maven Project”.
- **VS Code**: Requiere la extensión “Extension Pack for Java”.

Perfiles y Configuración

Perfiles

Los perfiles (`<profiles>`) permiten definir configuraciones condicionales activables por entorno, propiedad, sistema operativo, etc. Útiles para diferenciar entornos de desarrollo, pruebas y producción.

Ejemplo de perfil en pom.xml:

```
<profiles>
  <profile>
    <id>produccion</id>
```

```

<properties>
  <db.url>jdbc:mysql://prod-server:3306/proddb</db.url>
</properties>
<activation>
  <property>
    <name>env</name>
    <value>prod</value>
  </property>
</activation>
</profile>
</profiles>

```

Se activa con: `mvn install -Pproduccion` o `mvn install -Denv=prod`

Propiedades

Variables reutilizables en el POM, útiles para centralizar versiones, codificación, rutas, etc.

```

<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

```

settings.xml

Archivo de configuración global o de usuario para Maven. Permite definir:

- `localRepository`: Ruta del repositorio local.
- `servers`: Credenciales para repositorios remotos.
- `mirrors`: Espejos de repositorios para acelerar descargas.
- `proxies`: Configuración de proxy de red.
- `profiles`: Perfiles globales para todos los proyectos.

Ubicación:

- **Global**: `${MAVEN_HOME}/conf/settings.xml`
- **Usuario**: `${USER_HOME}/.m2/settings.xml`

Integración con IDEs y CI/CD

IDEs

- **IntelliJ IDEA**: Soporte nativo para Maven, sincronización automática de dependencias y ejecución de goals desde la interfaz gráfica.

- **Eclipse:** Plugin M2Eclipse para integración total, importación de proyectos y ejecución de comandos Maven.
- **VS Code:** Extensiones específicas para Java y Maven.

CI/CD

Maven es ampliamente utilizado en pipelines de integración continua (Jenkins, GitLab CI, GitHub Actions, Bamboo, etc.).

Recomendaciones:

1. Ejecutar en modo batch (-B) para evitar prompts interactivos.
2. Cachear el repositorio local (~/.m2/repository) para acelerar builds.
3. Separar pipelines para pruebas unitarias e integración.
4. Usar perfiles y propiedades para diferenciar entornos y credenciales.

Buenas Prácticas

Buenas Prácticas de Maven

1. Seguir la convención de estructura de carpetas.
2. Centralizar versiones y configuraciones en el POM padre o en `<dependencyManagement>`.
3. Evitar duplicidad de dependencias y gestionar exclusiones de transitivas.
4. Actualizar regularmente plugins y dependencias.
5. Usar perfiles para entornos diferenciados.
6. Documentar el POM y la estructura del proyecto.

Maven vs Gradle vs Ant

Comparativa:

Característica	Maven	Gradle	Ant
Lenguaje de configuración	XML (declarativo)	Groovy/Kotlin DSL (imperativo/declarativo)	XML (imperativo)
Rendimiento	Moderado	Alto (builds incrementales, cache)	Bajo
Flexibilidad	Moderada (convención sobre configuración)	Alta (scripts personalizados)	Alta (totalmente configurable)
Curva de aprendizaje	Media	Alta (requiere aprender DSL)	Baja
Gestión de dependencias	Excelente (repositorios centrales)	Excelente (compatible con Maven)	Manual (sin gestión automática)
Ecosistema	Maduro y amplio	En crecimiento, popular en Android	En desuso
Uso típico	Proyectos Java empresariales	Proyectos multi-plataforma, Android	Proyectos legacy o con necesidades muy específicas

¿Cuándo elegir Maven?

- Proyectos Java empresariales, equipos grandes, necesidad de estandarización y trazabilidad.
- Cuando la estabilidad y la predictibilidad son prioritarias.

¿Cuándo elegir Gradle?

- Proyectos que requieren builds muy dinámicos, multiplataforma o integración con Android.
- Cuando se prioriza la velocidad y la flexibilidad de scripting.

Proyectos Multi-Módulo

Permiten organizar soluciones complejas en varios subproyectos coordinados por un POM padre.

Estructura típica:

```
plataforma/  
|-- pom.xml (packaging: pom)  
|-- api/  
|   |-- pom.xml  
|-- servicio/  
|   |-- pom.xml  
|-- cliente/  
|   |-- pom.xml
```

- El **POM padre** centraliza dependencias, plugins y perfiles.
- Los módulos hijos heredan configuración y pueden depender entre sí.

Gestión de Versiones

- Uso de `-SNAPSHOT` para versiones en desarrollo.
- Versiones fijas para releases.
- Herramientas como `maven-release-plugin` para automatizar el proceso de liberación y actualización de versiones.

Testing y Calidad

Surefire y Failsafe

- **maven-surefire-plugin**: Ejecuta pruebas unitarias (**JUnit**, **TestNG**) en la fase test.
- **maven-failsafe-plugin**: Ejecuta pruebas de integración en la fase verify.

Configuración básica:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.1.2</version>
</plugin>
```

Soporta paralelismo, forking y configuración avanzada para optimizar tiempos de ejecución.

JaCoCo y Reports

- **jacoco-maven-plugin**: Genera reportes de cobertura de código.
- **maven-pmd-plugin**, **maven-checkstyle-plugin**: Análisis de calidad y métricas.
- **maven-site-plugin**: Integra reportes en la documentación del proyecto.

Empaquetado y Despliegue

Empaquetado

- **JAR:** Por defecto, para aplicaciones y librerías Java.
- **WAR:** Para aplicaciones web (estructura en `src/main/webapp`).
- **EAR:** Para aplicaciones empresariales (múltiples módulos JAR y WAR).

Despliegue

- **mvn install:** Instala el artefacto en el repositorio local.
- **mvn deploy:** Publica el artefacto en un repositorio remoto (Nexus, Artifactory).

Configuración de `<distributionManagement>` en el POM para definir repositorios de releases y snapshots.

Resolución de Problemas

Flags Útiles de Maven

Flag	Descripción
-X	Modo depuración detallada.
-e	Muestra <i>stacktraces</i> completos.
-U	Fuerza actualización de <i>snapshots</i> .
-T	Paralelismo en la construcción.
-o	Modo <i>offline</i> .

Problemas Frecuentes

- **Conflictos de dependencias:** Usar `mvn dependency:tree` y gestionar en `<dependencyManagement>`.
- **Errores de plugins:** Verificar versiones y configuración en el POM.
- **Problemas de red/proxy:** Configurar proxies en `settings.xml`.
- **Corrupción del repositorio local:** Eliminar `.m2/repository` y reconstruir.
- **Errores de JAVA_HOME:** Verificar que apunte al JDK correcto, no al JRE ni a `/bin`.

Rendimiento y Optimización

1. Actualizar Maven a la última versión para aprovechar mejoras de rendimiento.
2. Compilaciones paralelas: `mvn install -T 4` para usar 4 hilos.
3. Cachear el repositorio local en entornos CI/CD.
4. Compilaciones incrementales: Solo recompilar módulos modificados.
5. Optimizar configuración de plugins y evitar dependencias innecesarias.

Generación de Documentación

- **mvn site:** Genera documentación HTML del proyecto, incluyendo reportes de dependencias, pruebas, cobertura, etc.
- **maven-site-plugin:** Permite personalizar el sitio, menús, logos y recursos.
- **Integración de reportes:** Surefire, JaCoCo, Javadoc, PMD, etc., pueden incluirse en el sitio generado.

Manual Paso a Paso

1. Instalación de Maven

- Instala el JDK y configura JAVA_HOME.
- Descarga y descomprime Maven.
- Añade el directorio bin de Maven al PATH.
- Verifica la instalación con `mvn -version`.

2. Creación de un Proyecto

```
mvn archetype:generate
-DgroupId=com.ejemplo
-DartifactId=demo-maven
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

3. Estructura del Proyecto

- `src/main/java`: Código fuente.
- `src/main/resources`: Recursos.
- `src/test/java`: Pruebas.
- `pom.xml`: Configuración central.

4. Compilación

```
mvn compile
```

5. Gestión de Dependencias

- Añade dependencias en `<dependencies>` del POM.
- Ejecuta `mvn dependency:tree` para visualizar el árbol.

6. Ejecución de Pruebas

```
mvn test
```

7. Empaquetado

```
mvn package
```

8. Instalación en Repositorio Local

```
mvn install
```

9. Despliegue en Repositorio Remoto

- Configura `<distributionManagement>` y credenciales en `settings.xml`.
- Ejecuta `mvn deploy`.

10. Generación de Documentación

`mvn site`

11. Uso de Perfiles

- Define perfiles en `<profiles>` del POM.
- Activa con `mvn install -P<perfil>`.

12. Comandos Comunes y Flags

Comando	Descripción
<code>mvn clean</code>	Limpia el directorio <code>target</code> .
<code>mvn compile</code>	Compila el código fuente.
<code>mvn test</code>	Ejecuta pruebas unitarias.
<code>mvn package</code>	Empaqueta el proyecto.
<code>mvn install</code>	Instala en repositorio local.
<code>mvn deploy</code>	Despliega en repositorio remoto.
<code>mvn site</code>	Genera documentación del sitio.
<code>mvn dependency:tree</code>	Muestra árbol de dependencias.
<code>mvn help:effective-pom</code>	Muestra POM efectivo.
<code>mvn -X</code>	Modo debug detallado.
<code>mvn -T 4</code>	Usa 4 hilos para build.
<code>mvn -o</code>	Modo offline.
<code>mvn -DskipTests</code>	Omite ejecución de pruebas.

Guía Rápida

Instala Java (JDK) y Maven.

Crea un proyecto básico:

```
mvn archetype:generate
  -DgroupId=com.ejemplo
  -DartifactId=mi-app
  -DarchetypeArtifactId=maven-archetype-quickstart
  -DinteractiveMode=false
```

Compila el proyecto:

```
mvn compile
```

Ejecuta pruebas:

```
mvn test
```

Empaqueta el proyecto:

```
mvn package
```

Instala en el repositorio local:

```
mvn install
```

Agrega dependencias

Edita `pom.xml` en la sección `<dependencies>`.

Comandos útiles:

- Limpiar: `mvn clean`
- Árbol de dependencias: `mvn dependency:tree`
- Modo offline: `mvn -o`
- Omite pruebas: `mvn -DskipTests package`

Importa el proyecto en tu IDE

(IntelliJ, Eclipse, VS Code).

Consulta la documentación oficial

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

Ejemplo Práctico

Crear el proyecto:

```
mvn archetype:generate
  -DgroupId=com.ejemplo
  -DartifactId=demo-maven
  -DarchetypeArtifactId=maven-archetype-quickstart
  -DinteractiveMode=false
```

Estructura generada:

```
demo-maven/
|-- pom.xml
|-- src/
    |-- main/java/com/ejemplo/App.java
    |-- test/java/com/ejemplo/AppTest.java
```

Compilar y probar:

```
cd demo-maven
mvn clean compile
mvn test
```

Añadir una dependencia (ejemplo: Gson)

En pom.xml:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
```

Empaquetar:

```
mvn package
```

Ejecutar la aplicación (si tiene método main):

```
mvn exec:java -Dexec.mainClass="com.ejemplo.App"
```

Recursos

- Documentación oficial de Maven
- Maven Central Repository
- Guía de instalación de Maven
- Guía de comandos Maven
- Comparativa Maven vs Gradle
- Solución de problemas comunes
- Generación de sitios y documentación

Conclusión

Apache Maven es una herramienta esencial para el desarrollo profesional en Java, proporcionando automatización, estandarización y gestión eficiente de dependencias y builds. Su curva de aprendizaje se ve compensada por la robustez, la integración con el ecosistema Java y la facilidad para mantener proyectos a largo plazo. Dominar Maven es una inversión clave para cualquier desarrollador o equipo que busque calidad, reproducibilidad y escalabilidad en sus proyectos de software.