

# Maven

Miguel Angel Piña Avelino

Facultad de Ciencias, UNAM

13 de febrero de 2018



# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.

# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.
- Tiene un funcionamiento similar al de Ant.

# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.
- Tiene un funcionamiento similar al de Ant.
- Trabaja sobre configuraciones en xml.

# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.
- Tiene un funcionamiento similar al de Ant.
- Trabaja sobre configuraciones en xml.
- Proyecto de nivel superior de la Apache Software Foundation.

# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.
- Tiene un funcionamiento similar al de Ant.
- Trabaja sobre configuraciones en xml.
- Proyecto de nivel superior de la Apache Software Foundation.
- Utiliza Project Object Models (POM) para describir el proyecto a construir.

# ¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos en Java.
- Tiene un funcionamiento similar al de Ant.
- Trabaja sobre configuraciones en xml.
- Proyecto de nivel superior de la Apache Software Foundation.
- Utiliza Project Object Models (POM) para describir el proyecto a construir.
- Esta listo para usarse desde la red.



- Estandarización de las construcciones.

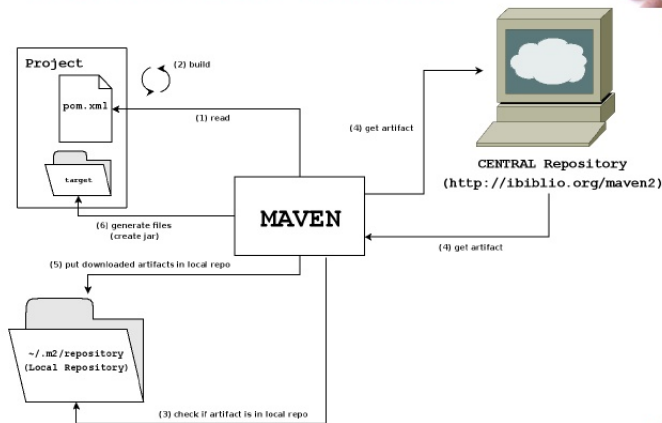
- Estandarización de las construcciones.
- Convención sobre configuración.

- Estandarización de las construcciones.
- Convención sobre configuración.
- Restringe ampliamente la variabilidad de construir proyectos de software.

- Estandarización de las construcciones.
- Convención sobre configuración.
- Restringe ampliamente la variabilidad de construir proyectos de software.
- Es ideal para proyectos nuevos.

- Estandarización de las construcciones.
- Convención sobre configuración.
- Restringe ampliamente la variabilidad de construir proyectos de software.
- Es ideal para proyectos nuevos.
- Los proyectos complejos y existentes podrían no ser adaptables a Maven.

## How Maven Works?



• Image Credit--

[http://blogs.exist.com/oching/wp-content/uploads/image/how\\_maven\\_works.png](http://blogs.exist.com/oching/wp-content/uploads/image/how_maven_works.png)

- **Compile:** Genera archivos .class desde los fuentes .java

- **Compile:** Genera archivos .class desde los fuentes .java
- **Test:** Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso de ejecución de maven si alguno de estos fallan.



- **Compile:** Genera archivos .class desde los fuentes .java
- **Test:** Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso de ejecución de maven si alguno de estos fallan.
- **Package:** Genera archivos .jar ó .war con los .class compilados.

- **Compile:** Genera archivos .class desde los fuentes .java
- **Test:** Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso de ejecución de maven si alguno de estos fallan.
- **Package:** Genera archivos .jar ó .war con los .class compilados.
- **Install:** Copia el fichero .jar (.war) a una carpeta donde Maven comparte todos los .jar generados. Esto permite que un mismo proyecto esté disponible para otros.

- **Compile:** Genera archivos .class desde los fuentes .java
- **Test:** Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso de ejecución de maven si alguno de estos fallan.
- **Package:** Genera archivos .jar ó .war con los .class compilados.
- **Install:** Copia el fichero .jar (.war) a una carpeta donde Maven comparte todos los .jar generados. Esto permite que un mismo proyecto esté disponible para otros.
- **Deploy:** Copia el .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a un servidor remoto.

Cuando se ejecuta alguno de los comandos anteriores, por ejemplo, `mvn install`, maven irá verificando que todas las fases previas a el se hayan cumplido.

Maven también tiene disponibles 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 estándar.

- **Clean:** Elimina todos los *.class* y *.jar* generados.

- **Clean:** Elimina todos los *.class* y *.jar* generados.
- **Assembly:** Genera un archivo *.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.

- **Clean:** Elimina todos los *.class* y *.jar* generados.
- **Assembly:** Genera un archivo *.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.
- **Site:** Genera un sitio web con la información de nuestro proyecto.



- **Clean:** Elimina todos los *.class* y *.jar* generados.
- **Assembly:** Genera un archivo *.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.
- **Site:** Genera un sitio web con la información de nuestro proyecto.
- **Site-deploy:** Sube el sitio web anterior.

# Creando un primer proyecto en Maven

Para crear un proyecto Maven simple, basta con ejecutar la siguiente instrucción (previamente con maven instalado).

```
mvn archetype:generate\  
-DgroupId="com.miguel.proyecto"\  
-DartifactId="mi-proyecto" -Dversion="0.0.1"
```

# Creando un primer proyecto en Maven

El código anterior va a crear un proyecto de maven con una serie de paquetes por defecto para diferenciarlo de otros proyectos `com.miguel.poyecto`, así como tener por nombre `mi-proyecto` y empezar con la versión `0.0.1`.

# Creando un primer proyecto en Maven

```
[10:21:39 miguel --> tmp]$ tree mi-proyecto/  
mi-proyecto/  
├── pom.xml  
└── src  
    ├── main  
    │   ├── java  
    │   │   ├── com  
    │   │   │   ├── miguel  
    │   │   │   │   ├── proyecto  
    │   │   │   │   │   App.java  
    │   └── test  
    │       ├── java  
    │       │   ├── com  
    │       │   │   ├── miguel  
    │       │   │   │   ├── proyecto  
    │       │   │   │   │   AppTest.java  
    └── 11 directories, 3 files  
[10:21:42 miguel --> tmp]$
```

Figura: Estructura de un proyecto de Maven

# Creando un primer proyecto en Maven

Podemos editarlo, compilarlo y jugar un poco con el proyecto. Para que pueda agregar la referencia de donde se encuentra el archivo principal (main) dentro del jar, esto una vez que sea empaquetado, hay que agregar la siguiente instrucción en el pom.xml, justo después de las dependencias del proyecto

# Creando un primer proyecto en Maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>com.miguel.proyecto.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Estableciendo el archivo main dentro del proyecto.

# Creando un proyecto web con Maven

De forma similar al ejemplo anterior, para construir un proyecto, podemos hacer uso de las herramientas que maven nos provee para construir una aplicación web.

```
mvn archetype:generate \  
-DgroupId="com.miguel.proyecto" \  
-DartifactId="mi-primer-aplicacion-web" \  
-DarchetypeArtifactId=maven-archetype-webapp \  
-DinteractiveMode=false
```

# Creando un proyecto web con Maven

```
[10:34:27 miguel --> tmp]$ tree mi-primer-aplicacion-web/
mi-primer-aplicacion-web/
├── pom.xml
└── src
    ├── main
    │   ├── resources
    │   └── webapp
    │       ├── index.jsp
    │       └── WEB-INF
    │           └── web.xml
```

5 directories, 3 files

Figura: Estructura de un proyecto web



# Incluyendo un servidor HTTP

Para tener un servidor http integrado con la aplicación, basta con agregar el siguiente snippet al archivo pom.xml

```
<build>
  <finalName>mi-primer-aplicacion-web</finalName>
  <plugins>
    <!-- Set JDK Compiler Level -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>${jdk-version}</source>
        <target>${jdk-version}</target>
      </configuration>
    </plugin>

    <!-- For Maven Tomcat Plugin -->
    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
      <artifactId>tomcat7-maven-plugin</artifactId>
      <version>2.2</version>
      <configuration>
        <path>/mi-primer-aplicacion-web</path>
      </configuration>
    </plugin>
  </plugins>
</build>
```

# Incluyendo un servidor HTTP

Para ejecutarlo, primero hay que construir el proyecto

```
mvn clean install
```

y después verificar que funciona

```
mvn tomcat:run
```

En un navegador web entramos a la url

```
http://localhost:8080/mi-primer-aplicacion-web/
```

# Estructura del proyecto

A diferencia del primer proyecto maven con el que trabajamos, en este vamos a tener un archivo especial: **web.xml**.

Este archivo es conocido Deployment Descriptor que es utilizado para determinar como las URL van a ser mapeadas a Servlets de java.

# Estructura del proyecto

Va a describir las clases, recursos y la configuración de la aplicación y cómo el servidor web va utilizar la información anterior para resolver peticiones web. Cuando un servidor web recibe una petición para la aplicación, se usa el `deployment descriptor` para mapear la URL de la petición al código que puede responder a dicha petición.

# Agregando servlets al proyecto

Un servlet, es una clase de Java que es utilizada para responder a las solicitudes http que llegan a un servidor. Estos servlets van a extender a la clase abstracta `HttpServlet` y van a sobrescribir al menos dos de los métodos utilizados para responder peticiones web: **doGet** y **doPost**.

# Agregando servlets al proyecto

Los servlets son definidos en el **deployment descriptor**, como se ejemplifica a continuación:

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>servlet</servlet-name>
    <servlet-class>com.miguel.proyecto.Servlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>servlet</servlet-name>
    <url-pattern>/foo</url-pattern>
  </servlet-mapping>
</web-app>
```

# Agregando un servlet al proyecto

En este descriptor agregamos un servlet llamado **servlet**, que tiene asociada a la clase `com.miguel.proyecto.Servlet`. Esta clase va a responder a las peticiones que lleguen a la url **/mi-proyecto/foo**, el servlet de ejemplo es el siguiente:

# Agregando un servlet al proyecto

```
package com.miguel.proyecto;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
public class Servlet extends HttpServlet {

    private void foo(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<HTML><HEAD><TITLE>Leyendo parámetros</TITLE></HEAD>");
        pw.println("<BODY BGCOLOR=\"#CCBBAA\">");
        pw.println("<H2>Leyendo parámetros desde un formulario html</H2><P>");
        pw.println("<UL>\n");
        pw.println("</BODY></HTML>");
        pw.close();
    }
}
```



# Agregando un servlet al proyecto

```
@Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    foo(request, response);
}
```

```
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException
    IOException {
    foo(request, response);
}
}
```

# Agregando servlets al proyecto

para que la clase anterior funcione, hay que agregar el siguiente snippet de código al archivo pom.xml

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>
```

# Agregando un servlet al proyecto

Lo anterior va a agregar todas las dependencias necesarias para poder construir de nueva cuenta nuestro proyecto:

```
mvn clean install  
mvn tomcat:run
```

Y entramos a la url

```
http://localhost:8080/mi-primer-aplicacion-web/foo
```