MAVEN

Maven es una herramienta open source para administrar proyectos de software. Por administrar, nos referimos a gestionar el ciclo de vida desde la creación de un proyecto en un lenguaje dado, hasta la generación de un binario que pueda distribuirse con el proyecto.

Maven nació dentro de la fundación Apache para complementar a Ant, la herramienta de compilación más usada en el mundo Java. Si no la conoces, piensa en Make para C. Esto es, Ant permite crear scripts (usando XML) que indican cómo compilar un proyecto Java y generar un binario.

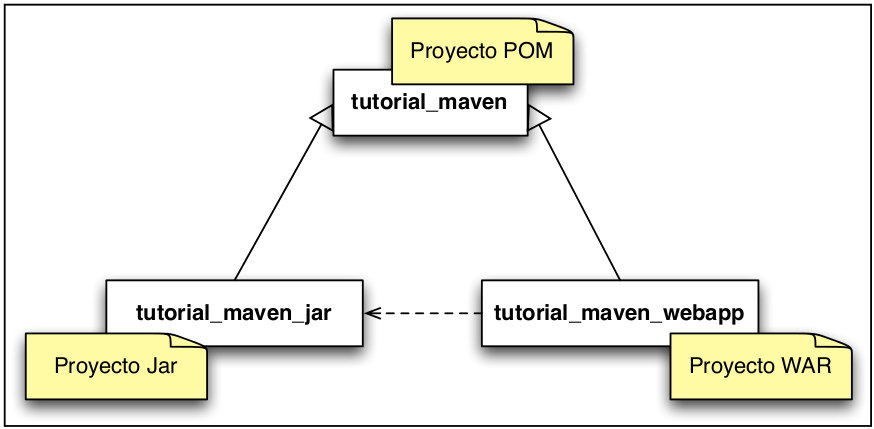
Maven complementa esta tarea brindándonos una estructura consistente de proyectos (todos los proyectos Maven tienen por default los mismos directorios) y herramientas necesarias actualmente para la complejidad de los proyectos de software: gestión avanzada de dependencias, informes sobre testing automáticos y extensibilidad vía plugins.

Requisitos:

* JDK de Java versión 5 ó superior.

Maven se ejecuta con el comando mvn (mvn.bat o mvn.sh) que se encuentra dentro de la carpeta bin de la carpeta apache-maven.

Maven promueve la modularización de los proyectos. Típicamente, en lugar de tener un proyecto monolítico, tendremos un módulo padre y varios módulos hijos.



POM son las siglas de "Project Object Model" (Modelo de Objetos de Proyecto), ya hemos dicho que Maven es una herramienta de administración de proyectos. Un POM no es más que la abstracción usada por Maven para definir dichos proyectos, como tal contiene los atributos de estos y las instrucciones para construirlo.

Un Proyecto en Maven se define mediante un archivo POM, que es un archivo llamado pom.xml con una etiqueta inicial de <project>. En dicho archivo se definen cosas como las instrucciones para compilar el proyecto, las librerías necesarias, etc.

En Maven, la ejecución de un archivo POM siempre genera un "artefacto". Este artefacto puede ser cualquier cosa: un archivo jar, un swf de flash, un archivo zip o el mismo archivo pom. **Maven trabaja modularizando tus proyectos.** De esta forma tendremos varios módulos que conforman un sólo proyecto. Para denotar esta relación en Maven, se crea un proyecto padre de tipo POM y los módulos se definen como otros archivos pom que heredan del primero.

Esta organización sirve para centralizar en el pom padre las variables (como el nombre del proyecto o el número de versión), las dependencias, los repositorios, etc. que son comunes a los módulos, eliminando duplicidad de código.

Los **arquetipos** son artefactos especiales de Maven que sirven como plantillas para crear proyectos. Maven cuenta con algunos predefinidos y terceros han hecho los suyos para crear proyectos con tecnologías específicas, como proyectos web con Spring o proyectos con Adobe Flex.

Crear proyecto maven

Desde línea de comandos y dentro del directorio donde se pondrá el proyecto.

mvn archetype:generate

Aparecerá la lista completa de artefactos en el repositorio central de Maven. Alrededor de 300 arquetipos (la 82 es maven-archetype-quickstart).

Se puede definir el arquetipo desde línea de comandos tecleando:

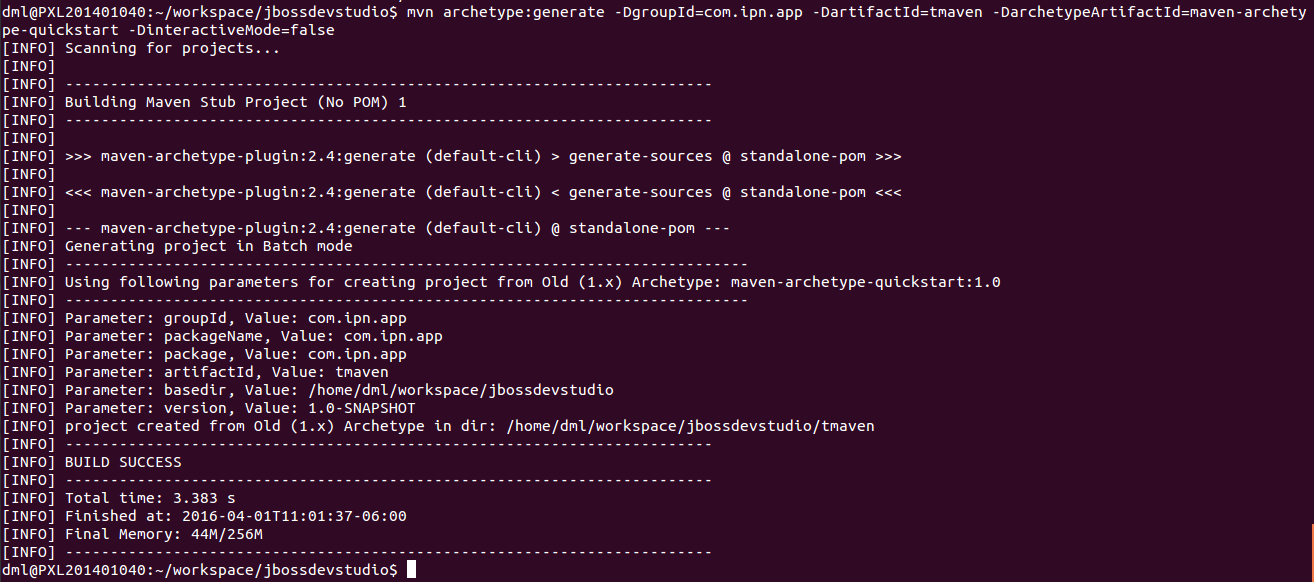
mvn archetype:generate -DgroupId=com.ipn.app -DartifactId=tmaven -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

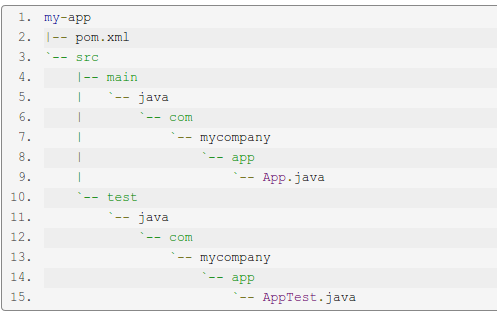
groupId: Piensa en él como en el paquete de proyecto. Típicamente aquí se pone el nombre de tu empresa u organización, ya que conceptualmente todos los proyectos con ese groupId pertenecen a una sola empresa.

artifactId: Es el nombre de tu proyecto.

version: Número de versión de tu proyecto.

package: Paquete base donde irá tu código fuente

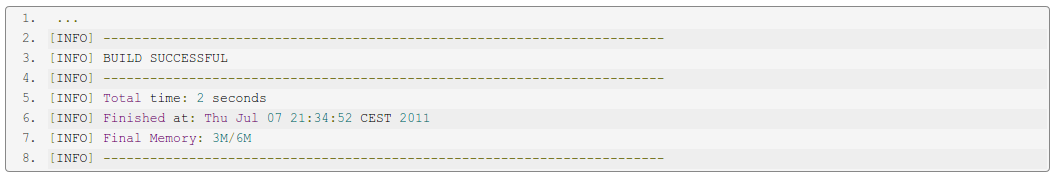






Construir el proyecto

mvn package



Una phase es un paso en el ciclo de vida de la construcción, el cual es un orden de una secuencia de phases. Cuando una phase es dada. Maven ejecutara cada phase en secuencia hacia arriba.

Ejemplo: compile

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile

Maven phases

These are the most common *default* lifecycle phases executed:

**validate**: validate the project is correct and all necessary information is available

**compile**: compile the source code of the project

**test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

**package**: take the compiled code and package it in its distributable format, such as a JAR.

**integration-test**: process and deploy the package if necessary into an environment where integration tests can be run

**verify**: run any checks to verify the package is valid and meets quality criteria

**install**: install the package into the local repository, for use as a dependency in other projects locally

**deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

There are two other Maven lifecycles of note beyond the *default* list above. They are

**clean**: cleans up artifacts created by prior builds

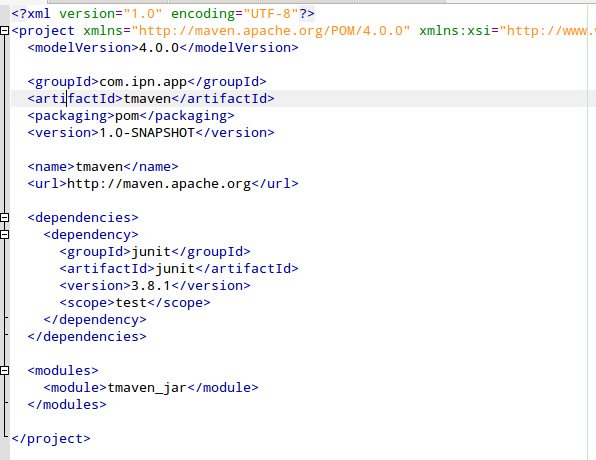
**site**: generates site documentation for this project

¿Qué es el repositorio local? Maven descarga sus dependencias y las dependencias de tus proyectos de un repositorio central. Por default, usa el repositorio central de Maven (http://repo1.maven.org/maven2) aunque puedes definir que use otros incluido alguno que tu hayas creado dentro de tu intranet. Estas dependencias las almacena en un repositorio local, que no es otra cosa que una carpeta dentro de tu computadora con el fin de no tener que volver a descargarlas otra vez.

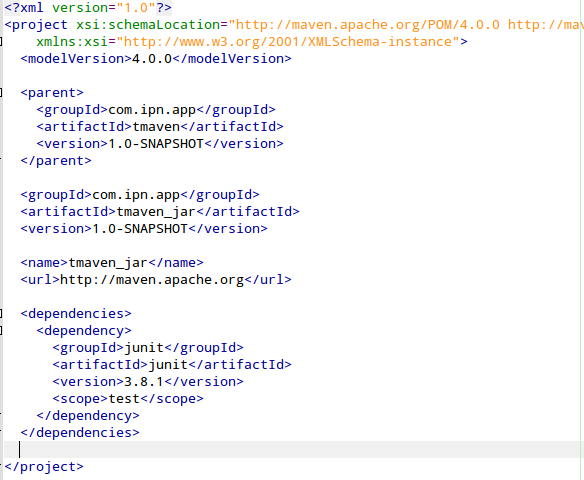
Maven copia ahí tus artefactos, los proyectos que tu generes. Por default, el repositorio local está en tu home de usuario dentro de la carpeta oculta .m2/repository.

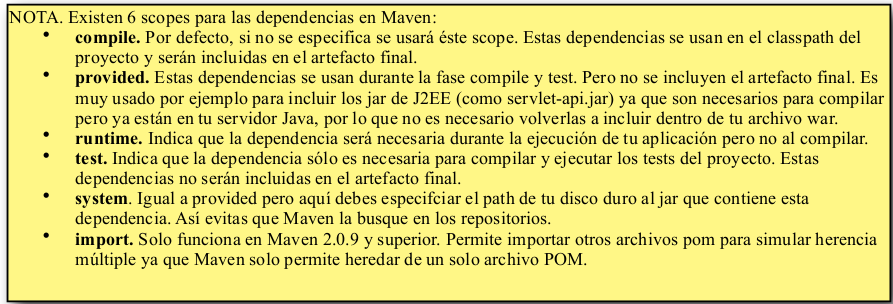
Crear Módulos

Ahora crearemos el primer módulo de nuestra aplicación empaquetado como un jar. Desde línea de comandos y dentro de la carpeta donde creamos el proyecto (tutorial\_maven).



Algo interesante de esta dependencia es que tiene un <scope> definido como de test. Esto indica a Maven que esta librería solo se usará durante la fase de testing y no será incluida en los binarios. Normalmente tus dependencias que definas no llevan este scope porque queremos que sean incluidas en el binario.





Definir que utilizara java 5 [7].

