Que es maven

Maven nacio con el proposito de simplificar el proceso del build de un projecto software.
Existen una gran cantidad de proyectos java basados en ant y en sus ficheros project.xml. Cada uno de estos proyectos difiere del resto en sus ficheros de configuracion, son ligeramente diferentes. Asi como tambien cada uno de los proyectos tiene el servidor de control de versiones (CVS/SVN) repleto de ficheros jar.

Los objetivos que persigue maven son los siguientes:

- Hacer el proceso de construcion facil.
- Proveer un proceso de construccion uniforme.
- Proveer una cantida de informacion sobre el proyecto.
- Servir como guia de buenas practicas de desarrollo de software.
- Permitir una migracion transparente a nuevas funcionalidades.
- Setup simple de proyetos siguiendo buenas practicas de software. Generar un proyecto nuevo en pocos segundos.
- Manejo de dependencias incluyendo actualizaciones automaticas. Tanto dependencias primarias como transitivas.
- Permite trabajar de una forma facil con multiples proyectos al mismo tiempo.
- Grande y creciente repositorio de librerias y metadatos externo a nuestro proyecto. Liberando asi el sistema de control de versiones de contener jar´s.
- Es extensible. Haciendo uso de sus sistema de pluginsen java o en lenguajes de scripts.
- Acceso instantaneo a nueva funcionalidades con una minima o ninguna configuracion.
- Posibilidad del uso de tareas ant para manejar dependencias y despliegue fuera de maven.
- Maven esta preparado para un gran numero de builds para proyectos, ya sea tipo jar, war, ear, etc..
- Usando los metadatos asociados al proyecto, maven es genera un sitio web o pdf incluyendo cualquier documentacion que se quiera. Ademas de toda la informacion que maven añade como api, java doc, informacion sobre desarrolladores, informes de test, etc.
- Manejo de release y publicacion de distribuciones. Maven pude ser integrado con el sistema de control de versiones y manejar las releases de un proyecto en un tag concreto. Maven puede publicar distribuciones basadas en jar, en un archivo incluyendo dependencias y documentacion, o una distribucion de fuentes.
- Manager de dependencias: Maven impulsa el uso de un repositorioi central de JAR´s y otras dependencias. Maven viene con un mecanismo por el cual tu proyecto puede descargar otras dependencias requeridas para el proyecto desde un repositorio central de JAR´s.Esto permite a los usuario de maven la reutilizacion de JAR´s entre proyectos y impulsa la comunicación entre proyectos. Maven colabora con el repositirio Ibiblio.

¿Que no es Maven?

Posiblemente haya oido decir que maven es para:

- Que maven es una herramienta para generar sitios web y documentacion.
- Maven extiende la funcionalidad de ant incluyendole manejo de dependencias.
- Maven es un conjunto de scripts de ant reusables.

Maven impulsa el uso de buenas practicas en el proceso de generacion de software.

Como podria Maven ayudar a un desarrollo software

Maven impulsa el uso de buenas practicas en el desarrollo de software:

- Sistema automatico de build.
- Manejo de dependencias automatico.
- Versionado de librerias.
- Liberacion de espacio en el servidor de control de versiones. Ya que no existiria nigun JAR dentro de el. Estarian organizados en el repositiorio de maven.
- Paso de test unitarios automaticamente. Con informacion de donde ha fallado.
- Test de covertura de funciones.
- Ayuda en la distribucion de fuentes
- Ayuda en la generacion de instaladores.
- Ayuda en la generacion de documentacion asociada al proyecto y sitio web de desarrollo.
- Etc..

Primeros pasos con maven

A continuacion se va a mostras un pequeño tutorial de primeros pasos con maven. La parte de instalacion de maven que se cita a continuacion, tal y como se ha pensado para GvSIG no sera necesaria ya que se va a integrar dentro de GvSIG para no tener que instalarlo.

Instalar maven

Antes de que te pongas a hacer nada, necesitas tener una conexión de internet. **Maven** la primera vez que se ejecuta empiza a bajarse cosas y si no puede, no sirve para nada.

Aunque la instalación que cuento aquí es para windows, supongo que en linux es muy similar. Basta bajarse el zip de maven, desempaquetarlo y poner su directorio **bin** en el path de busqueda de ejecutables.

En mi caso lo he desempaquetado en **C:**, con lo que se me ha creado el directorio **C:\maven-2.0.4**

Para meter el subdirectorio **bin** en el path de ejecutables, le doy con el botón derecho del ratón sobre el icono de "Mi PC" y elijo "Propiedades", "Avanzado" y "Variables de Entorno". Busco "Path" y le doy a "Editar". Añado al final ";C:\maven-2.0.4\bin" (con punto y coma delante, para separlo de lo que ya haya escrito)

En linux supongo que es cuestión de coger el fichero .login, .profile, .bashrc o el que sea según el unix/linux que tengamos correspondiente del $HOME del usuario y añadirle al final la línea

PATH=$PATH;/directorio_maven/maven-2.0.4/bin

En el caso de windows (y en el de linux) hay que abrir una ventana de comandos nueva después de haber realizado los cambios (ventana de ms-dos o bash) y para probar que todo va bien, podemos ejecutar

C:\> mvn --version
Maven version: 2.0.4

$ mvn --version
Maven version: 2.0.4

Es posible que la primera vez que lo ejecutemos tarde un rato, ya he comentado que cada vez que ejecutamos un comando nuevo de **maven**, tiene que bajarse cosas de internet.

Crear un proyecto

El primer paso que podemos hacer con maven es **crear un proyecto** desde cero. El comando de **maven** que tenemos que ejecutar es

mvn archetype:create -DgroupId=*org.cit.gvsig* -DartifactId=*EjemploExtension*

Veamos los parámetros

- **archetype:create** es el comando/plugin, o como quieras llamarlo, de maven para crear un proyecto. Por defecto crea un proyecto de java normalito (nada de aplicación web, aunque también se puede)
- **-DgroupId=***org.cit.gvsig* es el conjunto de proyectos al que pertenece nuestro proyecto. Por ejemplo, yo puedo meter todos mis programas de ejemplo en un grupo que llamaré "*org.vit.gvsig*". Este nombre que pongamos aquí va a servir de paquete inicial para todas las clases del proyecto. Todos los proyectos **maven** deben pertenecer a un grupo, aunque sea único para él, que se denominará **groupId**.
- **-DartifactId=***EjemploExtension* es el nombre que queramos dar al proyecto. **Maven** creará un directorio con este nombre y el jar que genere para el proyecto tendrá también este nombre. Todos los proyectos **maven** tienen un nombre para identificarlos, que se denomirá **artifactId**

Una vez ejecutado este comando, **Maven** empezará a bajarse cosas de internet cuando lo ejecutemos por primera vez (en los próximos proyectos ya no necesita bajarse nada) y creará una esctructura de directorios y ficheros como la siguiente

```
EjemploMaven
+---src
¦    +---main
¦    ¦   +---java         //Para nuestros fuentes
¦    ¦       +---org.cit.gvsig
¦    ¦           +---EjemploExtension
¦    ¦                   +--App.java
¦    +---test
¦        +---java         //Para test de Junit
¦            +---org.cit.gvsig
¦                +---EjemploExtension
¦                        +---AppTest.java
+---pom.xml
```

Crea un directorio **EjemploExtension** para nuestro proyecto.

Dentro un fichero **pom.xml** que es un fichero que contiene datos de configuración de nuestro proyecto, como dependencias con otros jar, tipos de informes que queremos en la página web de nuestro proyecto, etc.. Inicialmente contiene una serie de cosas por defecto que podremos cambiar si lo deseamos.

Crea dos subdirectorios, uno **src** y otro **test**. Dentro de **src** debemos meter todo lo que sean fuentes y ficheros de configuración o datos propios del proyecto. En la parte de **test** debemos meter todos nuestros fuentes de prueba, clases de test de JUnit, ficheros de datos o de configuración de pruebas, etc. Es decir, en **src** va lo que es del proyecto y en **test** lo que nos ayude a probar el proyecto, pero no sea propio del proyecto.

En ambos casos, crea un directorio **java**. Ahí debajo es donde van los fuentes java. En paralelo a estos directorios **java** y si lo necesitamos, debemos crear nosotros a mano otros directorios. El nombre de estos nuevos directorios es standard en **maven**. Para un proyecto normalito los más útiles puede ser

- **config** para ficheros de configuración, iconos, etc
- **resources** para ficheros que queramos que se metan dentro del jar. **Maven** meterá

automáticamente todo lo que haya en este subdirectorio dentro del jar

- **assembly** para la configuración que queramos en nuestro zip de distribución.

En http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html tienes la estructura standard de directorios propuesta por **maven**, pero completita.

Maven mete directamente el paquete inicial *org.cit.gvsig*, según indicamos en el **groupId** y en él mete un par de ficheros java de muestra, **App.java** y **AppTest.java**. Normalmente estos ficheros los borraremos, salvo que queramos aprovechar el nombre.

Empezamos a trabajar

Ahora llega el momento duro. Debemos empezar a escribir el código, tanto de nuestro proyecto como de las clases de test de JUnit, si es que seguimos al pie de la letra las costumbres de buena programación. Desgraciadamente, todavía no hay herramientas que hagan este trabajo por nosotros, así que a ello.

Compilar

Una vez que tenemos todo, podemos compiliar de forma sencilla. Basta ponerse en el directorio donde está el fichero **pom.xml** y escribir

mvn compile

Esto creará un directorio **target** justo debajo de **EjemploMaven** y ahí un subdirectorio **classes** donde meterá todos los .class de nuestro compilado

```
EjemploMaven
+---src
¦   +---main
¦   ¦   +---java
¦   ¦       +---com.cit.gvsig
¦   ¦           +---EjemploExtension
¦   +---test
¦       +---java
¦           +---com.cit.gvsig
¦               +---EjemploExtension
+---target
    +---classes
        +---com
            +---cit
                +---gvsig
                    +---EjemploExtension   //Aqui van los ficheros .class
```

Generar el jar

Para generar el jar, es igual de sencillo

mvn package

Esto primero compilará si es necesario, pasará las clases de test de JUnit y si no hay fallos, meterá en el directorio **target** nuestro jar, que por defecto tendrá un nombre tan feo como este

EjemploExtension-1.0-SNAPSHOT.jar

Bien, **EjemploExtension** es como nosotros queremos que se llame según indicamos en el **artifactId**. **Maven** añade un 1.0 para indicar que es la versión 1.0 de nuestro proyecto. Este número aparece y podemos cambiarlo en el fichero **pom.xml**. Lo de **-SNAPSHOT** es para indicar que esta versión está en construcción, que no es definitiva. **Maven** irá guardando todas las versiones del jar

que generemos e irá sustituyendo **-SNAPSHOT** por la fecha y hora de la construcción. Esto de **-SNAPSHOT** también aparece en el **pom.xml** y podemos quitarlo cuando creamos que tenemos la versión definitiva.

Dentro de jar se meterán automáticamente todos los ficheros que tengamos debajo del directorio **resources**

Repositorios Maven

Una de las grandes ventajas de **maven** son los repositorios (almacenes) de ficheros jar que se crea.

Si miras en http://www.ibiblio.org/maven2/ tienes el repositorio oficial de jars de maven. Ahí están los **groupId** de casi todos los jar de libre distribución que puedas encontrar en internet. Tienes el log4j, commons-logging, JFreeChart, mysql-connector, etc, etc. **Maven** es capaz de bajarse cualquiera de estos jar si tu proyecto lo necesita.

Todo lo que se baje **maven** de internet lo mete en un repositorio (almacen) local en tu pc, de forma que si lo necesita una segunda vez, no necesita descargárselo nuevamente de internet. Este directorio, habitualmente está en

- **$HOME/.m2** en unix/linux
- **C:\Documents and Settings\usuario\.m2** en windows

Adicionalmente podemos configurar **maven** para que use más repositorios, tanto de internet como otros que montemos internamente en nuestra red local si la tenemos. Por ejemplo, en internet tienes otro repositorio con jars independiente de maven en http://java.freehep.org/maven2/

El repositorio interno en la red local es particularmente útil si trabajamos en una empresa y hacemos proyectos que son dependientes unos de otros o tenemos nuestras propias librerías de jars. Poniendo jars en ese repositorio de empresa, todos los programadores podrán acceder o subir jars a ese repositorio y compartirlos.

Dependencias de nuestro proyecto

Una vez que sabemos que hay un montón de jars por el mundo a nuestra disposición, sólo tenemos que saber cómo hacer que **maven** se los baje cuando nosotros queramos.

Para decir que necesitamos un jar de los que algún alma caritativa ha puesto a nuestra disposición, tenemos que editar el fichero **pom.xml** que tiene por defecto esta pinta

```
<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>org.cit.gvsig</groupId>
  <artifactId>EjemploExtension</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</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>
</project>
```

Vemos que hay un apartado **dependencies** y que dentro tiene un **dependency** de junit. Esto es la opción por defecto. Para esta dependencia de Junit hay que dar el **groupId** (junit), el **artifactId** (junit otra vez), la **versión** que deseamos (3.8.1) y cuándo la necesitamos (en los test)

Imaginemos que queremos el **log4j**, pero para nuestro programa, no para el test.

Nos vamos al respositorio oficial de maven http://www.ibiblio.org/maven2/ y navegamos hasta llegar el fichero **.pom** del log4j en la versión que queremos, es decir, vamos "pinchando" en: log4j, log4j, 1.2.13 y finalmente log4j-1.2.13.pom. Ahí vemos el **groupId**, **artifactId** y **versión** del log4j que queremos. Ahora editamos nuestro **pom.xml** y le añadimos la dependencia

```
<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>chuidiang.ejemplos</groupId>
  <artifactId>EjemploMaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.13</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Esta vez hemos puesto "compile" en vez de "test" para indicar que lo necesitamos al compilar nuestro proyecto y no al hacerle el test. Con esto está todo listo. Ya podemos usar en nuestros fuentes el log4j. Cuando compilemos la primera vez, **maven** mirará si ya tenemos log4j en nuestro repositorio local y si no lo tenemos, se irá a buscarlo a internet y lo bajará. El log4j quedará en nuestro repositorio local.

Integración con los IDE

Bueno, todo esto está muy bien, pero si yo trabajo con un IDE, como eclipse, IntelliJ IDEA, netbeans o algo así, ¿cómo encuentro ahora los jar en un sitio tan escondido?. "Fácil" entre comillas. **Maven** es capaz de generar un proyecto de eclipse, de idea y creo que de netbeans. Basta ejecutar esto

```
mvn eclipse:eclipse
mvn idea:idea
```

luego, desde nuestro IDE, creamos un proyecto importando o leyendo el fichero de proyecto que ha generado **maven**.

Digo "Fácil" entre comillas, porque al menos para eclipse, hay que hacer algo más. Debemos definir dentro de eclipse la variable **M2_REPO** apuntando a nuestro repositorio local de **maven**. En la Chuwiki puedes ver los detalles de cómo se coge con eclipse un proyecto maven. Por lo que he visto en IntelliJ, es más inmediato y no hay que hacer nada especial.

También hay plugins en nuestros IDEs que hacen más fácil el trabajo con maven, de forma que desde nuestro IDE podamos ejecutar las tareas maven. En codehaus tienes los plugins para los cuatro IDEs más conocidos: eclipse, netbeans, IntelliJ y JBuilder.

Llevar nuestro jar a los repositorios maven

Si queremos meter nuestro jar en el repositorio **maven** de jars local, es sencillo. Basta hacer

mvn install

Esto compilará si es necesario nuestros fuentes, les pasará los test, generará el jar y lo copiará en nuestro repositorio local de jars, en nuestro pc. Esta operación hace que ese jar esté disponible para otros proyectos **maven** que tengamos en nuestro ordenador. Es útil, por tanto, para proyectos **maven** que sean librerías nuestras que queramos usar en varios proyectos.

Si hemos configurardo un repositorio en red común a varios programadores en varios ordenadores, el comando es igual de simple

mvn deploy

Esto hace todo lo que hace **install**, incluido el install y luego pone nuestro jar en ese repositorio común en red. Esto lo pone a disposición de todos los programadores del equipo.

Lo de **-SNAPSHOT** en la versión tiene aquí su gracia. Si dependemos de un jar que tenga **-SNAPSHOT**, cada vez que compilemos, aunque ese jar esté en nuestro repositorio local, **maven** ira a buscarlos a los repositorios comunes o de internet, para ver si hay una versión de fecha más moderna. Si la hay, se la bajará. Por tanto, suele ser útil en un equipo de trabajo mantener la "coletilla" **-SNAPSHOT** en los jar que todavía están en desarrollo y sufren cambios frecuentes.

Si no ponemos **-SNAPSHOT**, una vez bajado el jar a nuestro repositorio local, **maven** NO se preocupará de buscar si hay versiones más modernas en los repositorios remotos.

Generar documentación

Generar el javadoc es fácil también entre comillas. Lo primero es que debemos editar el fichero **pom.xml** para indicarle que queremos este tipo de documentación. Puede quedar así

```
<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>chuidiang.ejemplos</groupId>
  <artifactId>EjemploMaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</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>
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
      </plugin>
```

```
    </plugins>
  </reporting>
</project>
```

Luego ya es sencillo. Basta ejecutar

mvn javadoc:javadoc

y **maven** nos generará en target un directorio **target\site\apidocs** y dentro de él metera el javadoc.

Si ejecutamos

mvn site:site

generará en **target\site** una documentación web por defecto, incluido el **javadoc**. Por supuesto, en el **pom.xml** y en algunos ficheros adicionales de configuración se podría personalizar. El aspecto de esa documentación por defecto puedes verlo en la página de maven, ya que por supuesto, la web de **maven** ha sido generada con **maven**. También en la página de codehause o en la de freehep.