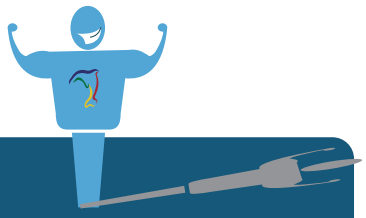
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

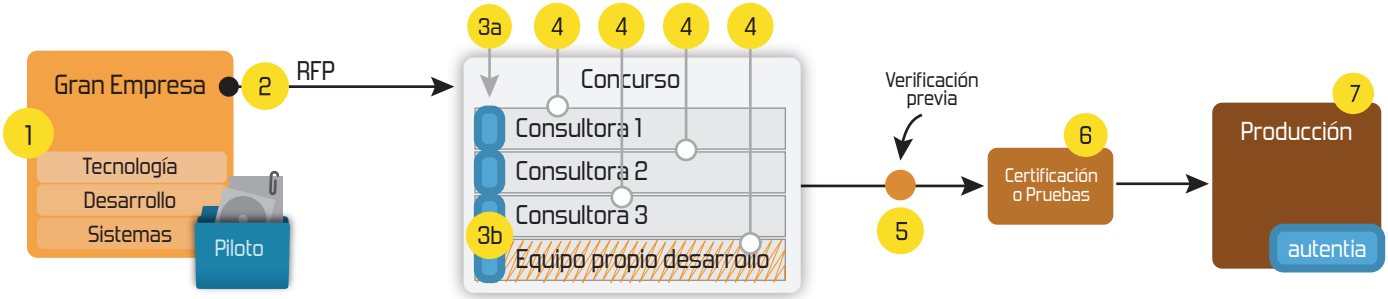
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Alejandro Perez García 2003-2006
Alejandro es Socio fundador de Autentia y nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Si te gusta lo que ves, **puedes contratarle** para impartir **cursos presenciales** en tu empresa o para ayudarte en proyectos (Madrid).

Contacta: alejandropg@autentia.com.

Descargar este documento en formato PDF maven.pdf

Firma en nuestro libro de Visitas

Maven, nunca antes resultó tan fácil compilar, empaquetar

Creación: 16-09-2006

Índice de contenidos

1. Introducción

Maven (http://maven.apache.org) es una herramienta para la gestión de proyectos de software, que se basa en el concepto de POM Object Model). Es decir, con Maven vamos a poder compilar, empaquetar, generar documentación, pasar los test, preparar las build

Ahora muchos pensaréis que eso ya los sabemos hacer con Ant (http://ant.apache.org), pero no debemos confundirnos, ya que Mav son cosas totalmente diferentes. Algunas de las principales ventajas de Maven frente a Ant, podrían ser:

- Maven se basa en patrones y en estándares. Esto permite a los desarrolladores moverse entre proyectos y no necesitan apre compilar o empaquetar. Esto mejora el mantenimiento y la reusabilidad.

- Mientras que con Ant escribimos una serie de tareas, y unas dependencias entre estas tareas; con el POM hacemos una descr proyecto. Es decir, decimos de que se compone nuestro proyecto (nombre, versión, librerías de las que depende, ...), y Mave encargará de hacer todas las tareas por nosotros.
- Maven hace la gestión de librerías, incluso teniendo en cuenta las dependencias transitivas. Es decir, si A depende de B y B d C, es que A depende de C. Esto quiere decir que cuando empaquetemos A, Maven se encargará de añadir tanto B como C en

En cualquier caso no tienen por que ser herramientas enfrentadas, sino que pueden ser complementarias. Usando cada una según n interese.

En este tutorial vamos a ver como trabajar con Maven, y como explotar algunas de sus características. El caso de ejemplo que vamo será un proyecto java normal que llamaremos autentiaNegocio y que depende del driver de MySQL (http://www.mysql.com) para p acceder a la base de datos, y un proyecto web que depende de "autentiaNegocio" y que llamaremos "autentiaWeb".

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Ahtec Signal 259M MXM (Sonoma 2.1 GHz, 2048 MB RAM, 100 GB HD).
- Sistema Operativo: GNU / Linux, Debian Sid (unstable), Kernel 2.6.17, KDE 3.5
- Máquina Virtual Java: JDK 1.5.0_08 de Sun Microsystems
- Maven 2.0.4

3. Instalación de Maven

Debemos descargarnos el paquete de Maven (por ejemplo `maven-2.0.4-bin.tar.bz2`) de http://maven.apache.org/download.html (es recomendable elegir un mirror para la descarga).

Una vez descargado, tenemos que descomprimirlo en nuestro ordenador. Por ejemplo, suponiendo que nos hemos descargado el pa Maven en `/download`, podemos hacer:

```
$ cd /opt
$ tar -xjf /download/maven-2.0.4-bin.tar.bz2
```

Esto nos creará el directorio `/opt/maven-2.0.4`.

Ahora basta con añadir el directorio `/opt/maven-2.0.4/bin` al PATH del sistema:

```
export PATH=$PATH:/opt/maven-2.0.4/bin
```

Para comprobar que está correctamente instalado podemos probar a ejecutar:

```
$ mvn --version
```

Si tenemos algún problema podemos comprobar si la variable JAVA_HOME apunta correctamente a la localización donde está instala JDK.

4. Creando proyectos y los arquetipos

Podríamos decir que un "arquetipo" para Maven es una plantilla. Es decir, gracias a un arquetipo Maven es capaz de generar una est directorios y ficheros.

Con los arquetipos se acabo "el miedo al folio en blanco" a la hora de empezar un proyecto, ya que basta con decirle a Maven que ti proyecto queremos y nos creará la estructura base.

Por ejemplo, para crear nuestro proyecto java basta con hacer:

```
$ mvn archetype:create -DgroupId=com.autentia.demoapp -DartifactId=autentiaNegocio
```

Donde `groupId` es el identificador único de la organización o grupo que crea el proyecto (se podría decir que es el identificador de la aplicación), y `artifactId` es el identificador único del artefacto principal de este proyecto (se podría decir que es el identificador del dentro de la aplicación), es decir, este será el nombre del jar.

En el comando no ha hecho falta decir que queremos construir un proyecto java, ya que esta es la opción por defecto.

El resultado de ejecutar este comando es la siguiente estructura de directorios y ficheros:

```
autentiaNegocio
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- autentia
    |               `-- demoapp
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- autentia
                    `-- demoapp
                        `-- AppTest.java
```

Esta estructura de directorios es estándar, siendo la misma en todos los proyectos. Maven nos permite cambiar esta estructura, per recomendable ya que el hecho de que sea estándar permite a los desarrolladores moverse entre proyectos con mayor comodidad, y siempre sabrán donde encontrar las cosas. Para más información sobre la estructura de directorios se puede consultar http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

5. El fichero pom.xml

El fichero pom.xml es donde vamos a describir nuestro proyecto. Vamos a echar un vistazo al fichero pom.xml que nos ha generado

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
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>com.autentia.demoapp</groupId>
    <artifactId>autentiaNegocio</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>
```

Con `packaging` se indica el tipo de empaquetado que hay que hacer con el proyecto. Podemos usar jar, war, ear, pom.

Con `version` se indica la versión del proyecto con la que estamos trabajando. Al indicar `SNAPSHOT` se quiere decir que es una versió es decir que estamos trabajando para obtener al versión `1.0`.

También podemos ver como dentro de dependecies se describen las dependencias del proyecto. Ahora tenemos una dependencia de poder compilar y ejecutar los test. Dentro de la descripción de las dependencias es interesante destacar el elemento scope que indic de librería se trata. Podemos distinguir:

- **compile** – es el valor por defecto. Se utiliza en todos los casos (compilar, ejecutar, ...).
- **provided** – también se utiliza en todos los casos, pero se espera que el jar sea suministrado por la JDK o el contenedor. Es d incluirá al empaquetar el proyecto, ni en el repositorio.
- **runtime** – no se utiliza para compilar, pero si es necesario para ejecutar.
- **test** – Sólo se utiliza para compilar o ejecutar los test.
- **system** – es similar a provided, pero eres tu el que tiene que suministrar el jar. No se incluirá al empaquetar el proyecto, ni repositorio.

Para saber más se puede consultar http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

6. Como compilar, empaquetar, ...

Ahora mismo, sin haber escrito ni una línea ya podemos hacer todas las tareas habituales (esto con Ant no sería tan fácil ;):

- `$ mvn compile` – compila el proyecto y deja el resultado en target/classes
- `$ mvn test` – compila los test y los ejecuta
- `$ mvn package` – empaqueta el proyecto y lo dejará en taget/autentiaNegocio-1.0-SNAPSHOT.jar
- `$ mvn install` – guarda el proyecto en el repositorio
- `$ mvn clean` – borra el directorio de salida (target)
- ...

Estas tareas son estándar, por lo que un desarrollador puede saltar de un proyecto a otro y siempre sabrá compilar, empaquetar, ...

Maven define un ciclo de vida por lo que al ejecutar un objetivo, antes se ejecutan los sus antecesores en el ciclo de vida. Por ejemp ejecutamos package, antes se ejecutará compile y test.

Para saber más sobre el ciclo de vida de Maven se puede leer http://maven.apache.org/guides/introduction/introduction-to-the-lifec

7. El repositorio de Maven

Ya hemos hablado en un par de ocasiones del "repositorio". Maven guarda todas las dependencias y proyectos en un único repositor

Este repositorio está situado en `<USER_HOME>/.m2/repository` (aunque Maven nos permite cambiar esta localización por defecto.

También existe un repositorio remoto desde el cual se descargan los diferentes jars según los vamos necesitando. Esto ya lo habréis ejecutar los primeros comando de Maven; seguro que habéis visto como Maven se ponía a descargar varias cosas (ojo, si salís a Int traves de un proxy, lo tendréis que configurar en Maven http://maven.apache.org/guides/mini/guide-proxies.html).

Este repositorio remoto también podemos cambiarlo, e incluso podemos tener más de uno. Sería muy interesante que dentro de nu organización tuviéramos un repositorio remoto donde publicar todos nuestros proyectos.

Para saber más podéis leer http://maven.apache.org/guides/introduction/introduction-to-repositories.html

En general esta idea es bastante buena porque no hace falta que tengamos los jar duplicados en el sistema de control de versiones, cada proyecto. Sobre todo con los jar de terceros, ya que ocupan espacio y es algo que no tiene utilidad que versionemos (lo interes saber en cada versión de nuestro proyecto que dependencias tenía pero no versionar la dependencia en sí).

Además el repositorio de Maven guardar las diferentes versiones de cada proyecto o dependencia, con lo cual podemos actualizar ur dependencia de un proyecto a una versión superior, sin que afecte al resto de nuestros proyectos (que seguirán usando la versión a

8. Creando el proyecto Web

Basándonos en los arquetipos, basta con ejecutar:

```
$ mvn archetype:create -DgroupId=com.autentia.demoapp -DartifactId=autentiaWeb
-DarchetypeArtifactId=maven-archetype-webapp
```

Como antes, indicamos el groupId y el artifactId, pero esta vez también indicamos el archetypeArtifactId. En este último atributo, c maven-archetype-webapp, estamos indicando que queremos usar la plantilla de aplicaciones web.

A parte de los arquetipos que nos proporciona Maven, es interesante saber que podemos creara nuestros propios arquetipos: http://maven.apache.org/guides/mini/guide-creating-archetypes.html

Echemos un vistazo al pom.xml generado:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
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>com.autentia.demoapp</groupId>
    <artifactId>autentiaWeb</artifactId>
```

```
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <finalName>autentiaWeb</finalName>
    </build>
</project>
```

Destacaremos como en este caso el elemento `packaging` es igual a `war`, y la aparición del elemento `finalName`.

Con `finalName` estamos indicando el nombre del archivo que se genera al empaquetar, en este caso el nombre del archivo war. En el anterior no usamos este elemento porque queríamos el comportamiento por defecto: el nombre del archivo se genera como el `artifactId` `version`. Pero este comportamiento no nos suele interesar en un archivo war, porque sino al desplegar la aplicación abría que poner de la aplicación en la URL para acceder a ella.

9. Definiendo dependencias

Hasta ahora no hemos escrito ni una sola línea, bueno ya es hora de ponerse a trabajar y hacer algo ;)

Vamos a indicar que el proyecto autentiaNegocio depende del driver de MySQL. Para ello basta con editar el fichero autentiaNegocio añadir:

```
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.0.3</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
    ...
</project>
```

Nótese como se indica scope runtime, es decir este jar sólo se tendrá en cuenta a la hora de ejecutar, pero no se usará para compila

Para indicar que autentiaWeb depende de autentiaNegocio, habrá que añadir al fichero autentiaWeb/pom.xml:

```
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>com.autentia.demoapp</groupId>
            <artifactId>autentiaNegocio</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
    ...
</project>
```

Ahora al empaquetar autentiaWeb, veremos que en el directorio WEB-INF/lib se ha añadido el jar de autentiaNegocio, y el jar del dr MySQL, ya que este último es dependencia de autentiaNegocio.

10. Un pom.xml para controlarlos a todos

En este punto podemos construir cada proyecto con sus dependencias correctas, pero tenemos las restricción de que tenemos que c primero autentiaNegocio (hay que hacer `install` para que el jar se guarde en el repositorio) y luego autentiaWeb, porque si lo hace contrario no se encontrará la dependencia correctamente.

Imaginemos que tenemos ahora 20 proyectos con dependencias cruzadas, la situación se hace inmanejable. Pero Maven ha pensado

Lo que necesitamos es un pom.xml que los controle a todos.

Este pom.xml se suele colocar en directorio padre de los proyectos (en nuestro caso autentiaNegocio y autentiaWeb), pero yo lo voy en el directorio "autentiademoapp". Este directorio está a la misma altura que autentiaNegocio y autentiaWeb (podríamos decir que hermanos). Esto lo hago para poder gestionarlo todo más fácilmente cuando esté trabajando con el Eclipse y el sistema de control d (Eclipse sólo puede interactuar con el sistema de control de versiones a través de proyectos o carpetas del workspace).

Este pom.xml tendrá el siguiente aspecto:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
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>com.autentia.demoapp</groupId>
    <artifactId>autentiademoapp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <modules>
        <module>../autentiaNegocio</module>
        <module>../autentiaWeb</module>
    </modules>
</project>
```

Nótese como el valor del elemento `packaging` es `pom`. Vease también como dentro del elemento `modules` se listan todos los módulos forman parte de la aplicación.

Al ejecutar cualquier objetivo sobre este `pom.xml`, Maven se encarga de ejecutarlo sobre todos los módulos, pero tiene en cuenta las dependencias entre ellos, de forma que compilará primero la dependencia y luego al dependiente.

Para terminar vamos a modificar el pom.xml de autentiaNegocio y autentiaWeb para indicar cual es su proyecto padre. Sí, en Maven establecer "herencia" entre proyectos, esto nos va a resultar muy útil para definir configuración compartida por los proyectos. Basta en cada pom.xml "hijo":

```
<project>
    ...
    <parent>
        <groupId>com.autentia.demoapp</groupId>
        <artifactId>autentiademoapp</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    ...
</project>
```

11. Configurando los plugins de Maven y heredando la configuración

En ciertas ocasiones nos puede interesar cambiar la manera en que Maven hace las cosas. Para ello podemos redefinir las propiedad por Maven. Por ejemplo vamos ha cambiar las propiedades para compilar nuestros proyectos con la versión 1.5 de Java.

Basta con añadir lo siguiente al fichero autentiademoapp/pom.xml (el pom "padre"):

```
<project>
    ...
    <build>
        ...
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.5</source>
                    <target>1.5</target>
                </configuration>
            </plugin>
        </plugins>
        ...
    </build>
    ...
</project>
```

Como este pom.xml está definido como padre del resto de proyectos, todos "heredarán" esta configuración.

12. Creación de perfiles

Maven nos permite crear diferentes perfiles para la construcción de nuestros proyectos. El perfil nos va a permitir definir propiedade para la construcción del proyecto en distintas situaciones. Por ejemplo, si estamos construyendo el proyecto para un entorno de pro pruebas o desarrollo, ...

Vamos a ver un ejemplo en el que crearemos un perfil para construir el proyecto sin información de depuración y con optimización. añadir lo siguiente al fichero autentiademoapp/pom.xml (el pom "padre"):

```
<proyect>
    ...
    <profiles>
        <profile>
            <id>build-release</id>
            <properties>
                <maven.compiler.debug>false</maven.compiler.debug>
                <maven.compiler.optimize>true</maven.compiler.optimize>
            </properties>
        </profile>
    </profiles>
    ...
</proyect>
```

Para activar un perfil existen varias formas (el perfil puede estar activo siempre, en función de si está definida una variable de entor dependiendo de la versión de JDK que tengamos instalada, con la opción -P, ...). En nuestro caso cuando queramos construir el proy este perfile ejecutaremos:

```
$ mvn package –P build-release
```

Nótese que "build-release" es el identificador del profile, que hemos definido en el pom.xml.

13. Como publicar una nueva "release" de nuestro proyecto

Dentro de Maven tenemos un plugin que nos gestiona la elaboración de nuevas release.

Antes de hacer nada vamos a configurar el proyecto para decirle con que sistema de control de versiones tiene que trabajar. Editam autentiaNegocio/pom.xml y añadimos:

```
<project>
    ...
    <scm>
        <connection>scm:svn:svn://servidorautentia/test/trunk/autentiaNegocio</connection>
        <developerConnection>scm:svn:svn://serveridorautentia/test/trunk/autentiaNegocio</developerC
    </scm>
    ...
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-release-plugin</artifactId>
                    <configuration>
                        <password>${scm.password}</password>
                        <tagBase>svn://servidorautentia/test/tags</tagBase>
                    </configuration>
            </plugin>
        </plugins>
        ...
    </build>
    ...
</project>
```

Con los elementos `connection` y `developerConnection` le estamos diciendo donde está el proyecto dentro del sistema de control de (en nuestro caso el Subversion). Para saber como escribir esta URL consultar http://maven.apache.org/scm/scm-url-format.html

Con el elemento `tagBase` definimos donde se tienen que guardar los diferentes tags que hagamos del proyecto, dentro del sistema d de versiones.

Con el elemento `password` definimos la clave para tiene que usar para conectar con el sistema de control de versiones (el usuario se estemos usando en el sistema, pero se le podría indicar otro con el elemento `username`). Se ve que, en vez de poner un valor fijo, s usando una propiedad.

Para dar valor a la propiedad `scm.password`, podemos pasarla con -D al ejecutar mvn o definirla en el fichero de configuración del u

Vamos a usar este último método; para ello añadimos al fichero `~/.m2/settings.xml`:

```
<settings>
    ...
    <profiles>
    ...
    <profile>
        <id>scmConfig</id>
        <activation>
            <activeByDefault/>
        </activation>
        <properties>
            <scm.password>miClaveSuperSecreta</scm.password>
        </properties>
    </profile>
    ...
    </profiles>
</settings>
```

Se puede ver como estamos definiendo un perfil que está siempre activo, y en este perfil estamos definiendo la propiedad `scm.pass`

Ahora que ya hemos configurado el proyecto para que sepa donde está nuestro repositorio podemos hacer:

```
$ mvn release:prepare
```

Este comando nos hace tres preguntas:

- What is the release version for "Proyecto de prueba de Autentia: autentiaNegocio"? (com.autentia.demoapp:autentiaNegocio)
  Nos pregunta cual es la versión de la release que estamos publicando. Por defecto Maven quita "-SNAPSHOT" a la versión que
  definida en el pom.xml en la etiqueta `version`.
- What is SCM release tag or label for "Proyecto de prueba de Autentia: autentiaNegocio"? (com.autentia.demoapp:xx) xx-1.0:
  Nos pregunta como queremos llamar a la etiqueta que se creará en el sistema de control de versiones. Por defecto Maven añ
  prefijo el `artifactId` del proyecto a lo contestado en la pregunta anterior.
- What is the new development version for "Proyecto de prueba de Autentia: autentiaNegocio"? (com.autentia.demoapp:autent
  1.1-SNAPSHOT: :
  Nos pregunta como queremos que se llame la nueva versión sobre la que seguiremos trabajando. Por defecto Maven increme
  último dígito de la versión que tenemos definida en el pom.xml en la etiqueta `version`.

En la mayoría de los casos podemos contestar a estas tres preguntas simplemente pulsando "Intro" y que coja los valores por defec

Una vez terminada la ejecución ya se habrá creado la etiqueta en el sistema de control de versiones. Ahora sólo nos queda ejecutar

```
$ mvn release:perform
```

Con esto Maven se baja la versión que acabamos de etiquetar, la compila, le pasa los test, hace un jar con los fuentes, hace un jar c
javadoc y lo publica todo en el repositorio. También lo publica en el repositorio remoto (en nuestro caso, como no hemos definido u
repositorio remoto nos dará un error). Es decir, la deja lista para que la utilice el resto del equipo.

14. Conclusiones

Como véis, con un mínimo esfuerzo por nuestra parte hemos conseguido mucho. Pero lo mejor de todo no es el haber escrito poco (
una ganancia), lo mejor es estandarizar este tipo de procesos.

Con Ant todo el mundo se acababa definiendo más o menos las mismas tareas, pero cada vez que se iba a un nuevo proyecto había
aprender que tareas concretas estaban definidas y como se comportaban estas tareas.

Otra de las grandes ventajas es la gestión de dependencias. Para hacer lo mismo con Ant nos tocaba escribir bastante.

Desde Autentia (http://www.autentia.com) siempre os recomendamos la adopción de estándares, así que esta no podía ser una exc

15. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Gestión de contenidos

Autentia S.L. Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..
y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	**Descripción**
Subversion, sistema de control de versiones, en Debian GNU/Linux	En esete tutorial aprenderemos a instalar y configurar el nuevo programa de gestión de versiones Subversion en Debian GNU/Linux

Instalación de Subversion (SVN) en Windows XP	En este tutorial os mostramos cómo instalar y utilizar la herramienta SVN en vuestro entorn XP
Desarrollo Gráfico Scripts Ant	Os mostramos como crear y ejecutar scripts Ant (para automatizar tareas en el mundo Java con las herramientas gratuitas Antelope y NetBeans
Subversive, cliente de Subversion para Eclipse	En este tutorial os enseñamos a utilizar este plugin de eclipse que permite trabajar con repositios de Subversion
Introducción a ANT	En el mundo Java, la compilación, verificación e instalación de aplicaciones se ha normalizac con este potente paquete llamado ANT.

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

www.AdictosAlTrabajo.com Opimizado 800X600