# Maven

* Maven

Herramienta de software para la gestión y creación de proyectos.

Podemos construir proyectos simples o basados en el uso de arquetipos.

* Arquetipo

Es una plantilla, es decir, un patrón o modelo desde el que se generan otros elementos del mismo tipo.

## Pasos a seguir para crear un proyecto Maven desde eclipse

* File -> New -> Proyect -> Maven -> Majen Proyect.
* Buscar el arquetipo que queremos incluir.

En nuestro caso, hemos usado maven-archetype-quickstart (el de apache).

* Indicar los nombres:
* Groupe Id: (com.soprasteria -> el nombre de la organización invertido).
* Artefact Id: (ejemplo -> el nombre del proyecto).
* Version: (0.0.1-SNAPSHOT es la que viene por defecto).
* Package: com.soprasteria.ejemplo (se rellena solo).

En este punto ya lo tenemos creado.

Podemos observar que consta de una colección de carpetas que se agrupan en src y un archivo llamado pom.xml. Se trata del archivo de configuración del proyecto.

Ahora lo que nos queda es generarlo.

* Nos posicionamos en el proyecto o en el pom.
* Run As -> Maven install (genera un achivo .jar de nuestro proyecto en la carpeta .m2).

Obtenemos el ejecutable.

Tenemos otras opciones que podemos utilizar dependiendo de lo que queremos hacer:

* Maven build (construir determinadas partes de nuestro proyecto).
* Maven clean (limpiar las anteriores ejecuciones).
* Maven clean verify (limpiar las anteriores ejecuciones y ver que todo está correcto).
* Maven generate-source (genera la carpeta generate-source dentro del target).
* Maven test (comprueba que se pasan satisfactoriamente los test).

## Pasos a seguir para crear nuestro arquetipo desde cmd a partir del proyecto generado

* Nos posicionamos dentro de la carpeta de nuestro proyecto (primer nivel) y abrimos la consola.
* Utilizamos el comando mvn archetype:create-from-proyect para crear el arquetipo.
* Esto nos generará una carpeta dentro de target/generate-source llamada archetype con el contenido del arquetipo.
* Nos movemos a esa carpeta en el cmd y lanzamos el comando mvn install para tener nuestro arquetipo guardado en local, es decir, dentro de la carpeta .m2.

Ya lo hemos conseguido.

## Pasos a seguir para crear un proyecto Maven desde cmd usando nuestro arquetipo creado

* Nos posicionamos en la carpeta donde queremos crear el nuevo proyecto y abrimos el cmd.
* Listamos los arquetipos locales para elegir el que queremos utilizar con el comando mvn archetype:generate -DarchetypeCatalog=local
* Indicamos el que queremos (usando el número que le identifica).
* Nos pide que escribamos los datos del proyecto:
  + Groupe Id: (com.soprasteria).
  + Artefact Id: (proyectoarquetipoejemplo).
  + Version: (0.0.1-SNAPSHOT).
* Package: (com.soprasteria.proyectoarquetipoejemplo).
* Aceptamos la configuración pulsando Y y le damos a enter.

Ya lo tenemos, solo nos queda ver que el contenido es el correcto, desde cmd o importando el proyecto a eclipse.

Maven está pensado para todo el ciclo de vida del proyecto.

* Ciclo de vida

Conjunto de fases en que son divididos los proyectos para así facilitar su gestión. Cada fase representa un estado en el ciclo de vida.

Maven consta de tres ciclos de vida predefinidos:

* Default

Gestiona la construcción y el despliegue del proyecto.

A continuación, podemos ver cada una de las etapas y para qué sirven:

* validate: verifica que el proyecto sea correcto y que toda la información necesaria esté disponible.
* initialize: inicializa el estado de construcción.
* generate-sources: genera cualquier código fuente para incluirlo en la compilación.
* process-sources: procesa el código fuente.
* generate-resources: genera recursos para su inclusión en el paquete.
* process-resources: copia y procesa los recursos en el directorio de destino, listos para empaquetar.
* compile: compila el código fuente del proyecto.
* process-classes: postprocesa los archivos generados a partir de la compilación.
* generate-test-sources: genera cualquier código fuente de prueba para incluirlo en la compilación.
* process-test-sources: procesa el código fuente de prueba.
* generate-test-resources: crea recursos para realizar pruebas.
* process-test-resources: copia y procesa los recursos en el directorio de destino de la prueba.
* test-compile: compila el código fuente de la prueba en el directorio de destino de la prueba
* process-test-classes: postprocesa los archivos generados a partir de la compilación de prueba.
* test: ejecuta pruebas utilizando un marco de prueba unitario adecuado. Estas pruebas no deberían requerir que el código esté empaquetado o implementado.
* prepare-package: realiza las operaciones necesarias para preparar un paquete antes del embalaje real. Esto a menudo da como resultado una versión procesada y descomprimida del paquete.
* package: toma el código compilado y lo empaqueta en su formato distribuible, como un JAR.
* pre-integration-test: realiza las acciones necesarias antes de que se ejecuten las pruebas de integración. Esto puede involucrar cosas como configurar el entorno requerido.
* integration-test: procesa e implementa el paquete si es necesario en un entorno donde se puedan ejecutar pruebas de integración.
* post-integration-test: realiza las acciones necesarias después de que se hayan ejecutado las pruebas de integración.
* verify: ejecuta cualquier verificación para comprobar que el paquete sea válido y cumpla con los criterios de calidad.
* install: instala el paquete en el repositorio local, para usarlo como una dependencia en otros proyectos.
* deploy: copia el paquete final en el repositorio remoto para compartirlo con otros desarrolladores y proyectos.
* Clean

Gestiona la limpieza del proyecto.

Consta de las siguientes etapas:

* pre-clean: ejecuta los procesos necesarios antes de la limpieza real del proyecto.
* clean: elimina todos los archivos generados por la compilación anterior.
* post-clean: ejecuta los procesos necesarios para finalizar la limpieza del proyecto.
* Site

Gestiona la creación de la documentación en la carpeta site del proyecto.

Se diferencian las siguientes fases:

* pre-site: ejecuta los procesos necesarios antes de la generación real del site del proyecto.
* site: genera la documentación del site del proyecto.
* post-site: ejecuta los procesos necesarios para finalizar la generación del site y se prepara para la implementación del sitio.
* site-deploy: implementa la documentación del site generada en el servidor web especificado.

Un concepto clave de Maven son los plugins. Toda la funcionalidad de Maven anteriormente descrita se realiza a través de plugins. Cada uno de ellos, se encarga de una tarea distinta.

* Plugin

Complemento que sirve para añadir nueva funcionalidad a nuestro proyecto.

Se pueden configurar en el pom del proyecto. Podemos tener plugins predefinidos y también personalizados. Los plugins ofrecen su funcionalidad a través de lo que se conoce como goals.

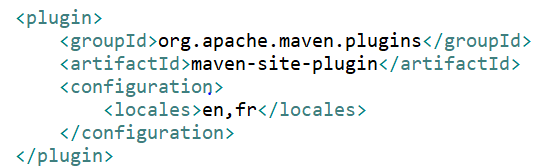
* Goal

Unidad mínima de ejecución que representa una tarea específica.

Los goals u objetivos se llevan a cabo siguiendo el orden en el que son invocados.

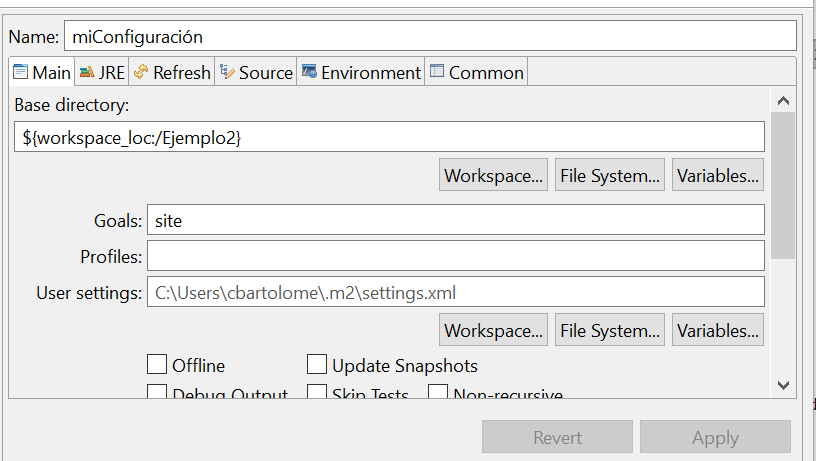
A continuación, vamos a ver algunos ejemplos:

* El plugin siguiente nos permite tener la documentación de nuestro proyecto disponible tanto en inglés como en francés.

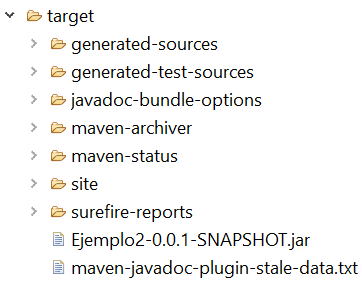


Para ello, tenemos que colocar este fragmento en el pom y después ejecutarlo de la siguiente manera:

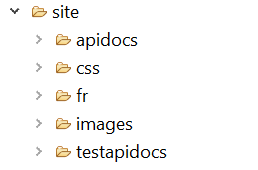
Run As -> Run Configurations -> en el apartado Goals indicar site



Una vez finalizado, podemos ver que se ha creado una carpeta site dentro de nuestro target. Ahí es donde tendremos guardada la documentación de nuestro proyecto.

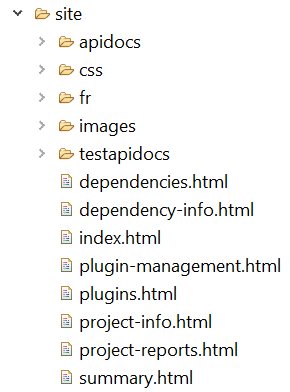
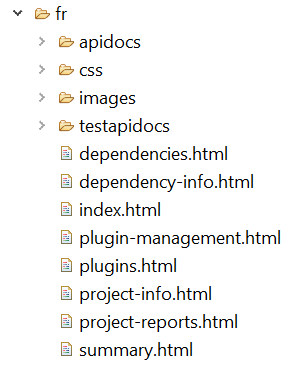


Nos queda ver que realmente nuestro plugin ha funcionado. Podemos verlo reflejado en nuestro directorio, dentro de la carpeta site.



Por defecto, nuestro sitio se genera en inglés y en la carpeta de nombre fr, vamos a contener la misma información, pero en el idioma indicado.

En inglés En francés

Si abrimos la página principal de cada uno, podemos verlo claramente:

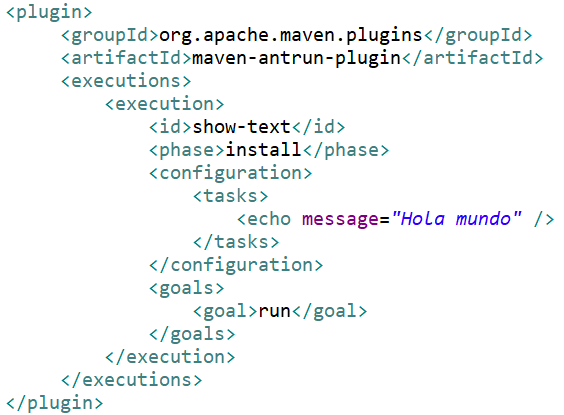
En inglés:



En francés:



* El plugin siguiente lo que va a hacer es mostrar un mensaje por consola durante una de las etapas del ciclo de vida, la que nosotros queramos.



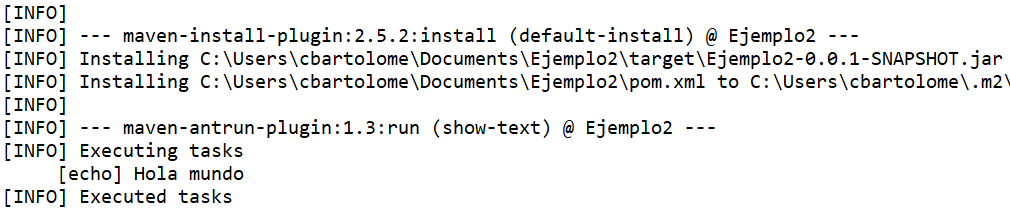
En la etiqueta <phase> ponemos el nombre de la fase. En este caso hemos indicado que sea install.

El mensaje aparece dentro de las etiquetas <tasks>. Hemos puesto un simple Hola mundo.

Nuestro goal es de tipo run.

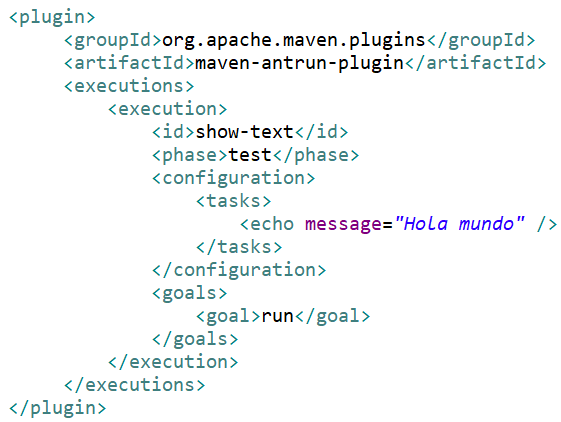
Luego procedemos a la ejecución: Run As -> Maven install

Observamos la consola y vemos el siguiente resultado:

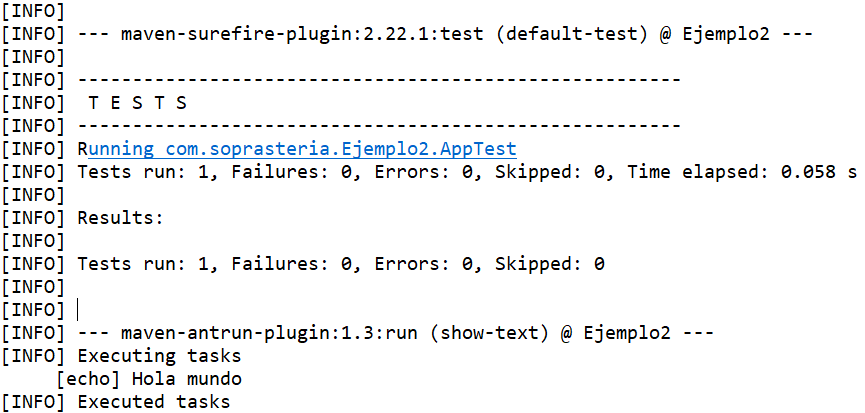


Vemos que se produce la instalación y seguidamente, se ejecuta nuestro run y se muestra el contenido que habíamos indicado.

Si lo queremos mostrar en otra etapa, solo tenemos que cambiarla.



Lo ejecutamos: Run As -> Maven test



Ahora aparece durante el test.

* MOJO = Maven + POPO = Maven plain Old Java Object

Objetivo (goal) de Maven que sirve para aumentar la funcionalidad de nuestro proyecto.

Para programar un MOJO utilizando Java como lenguaje vamos a seguir los siguientes pasos:

## Creación de nuestro MOJO desde eclipse

* File -> New -> Proyect -> Maven -> Majen Proyect.
* Buscar el arquetipo que queremos incluir.

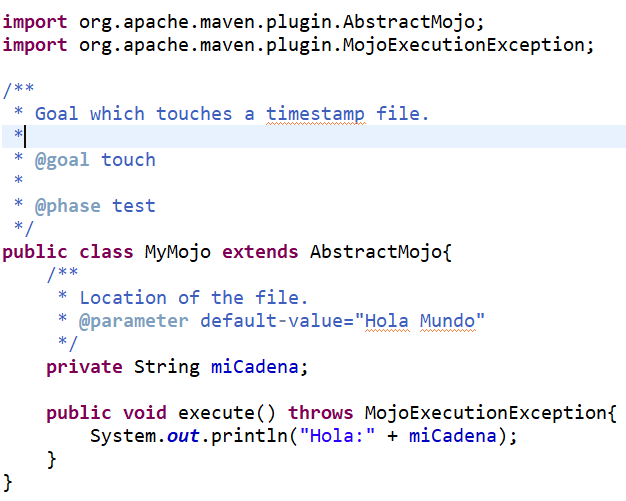
En nuestro caso, hemos usado maven-archetype-mojo (el de apache).

* Indicar los nombres:
* Groupe Id: (com.soprasteria -> el nombre de la organización invertido).
* Artefact Id: (HolaMundoMojo -> el nombre del proyecto).
* Version: (0.0.1-SNAPSHOT es la que viene por defecto).
* Package: com.soprasteria.HolaMundoMojo (se rellena solo).

En este punto ya lo tenemos creado.

Podemos observar que consta de una colección de carpetas que se agrupan en src y un archivo llamado pom.xml. Se trata del archivo de configuración del proyecto.

Hemos editado nuestra clase App para que haga lo que nosotros queramos. La única imposición que tenemos es que es necesario implementar la interface AbstractMojo En nuestro caso, solo buscamos que nos muestre un mensaje por la consola.

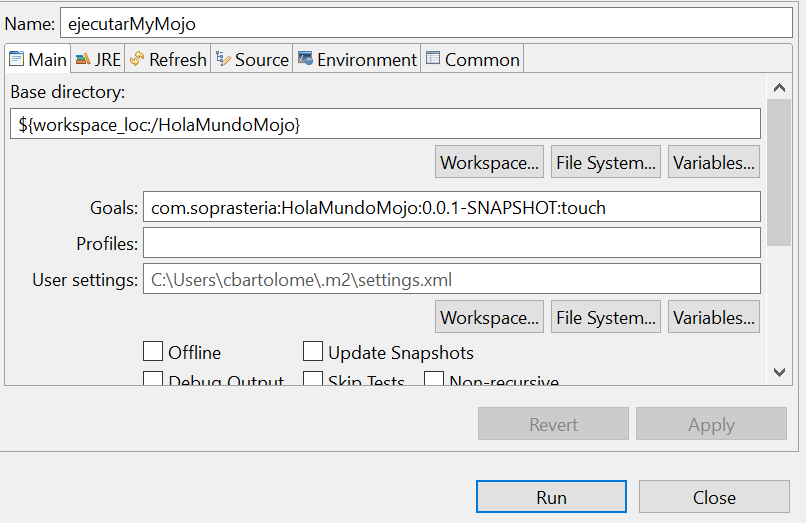


Ahora lo que nos queda es generarlo.

* Nos posicionamos en el proyecto o en el pom.
* Run As -> Maven install (genera un achivo .jar de nuestro proyecto en la carpeta .m2).

Si queremos ejecutarlo y ver lo que hace tenemos que seguir los siguientes pasos:

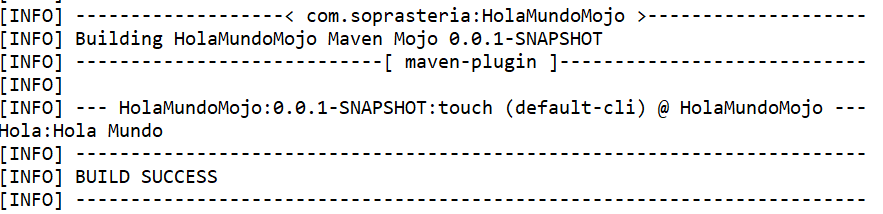
* Nos posicionamos en el proyecto o en el pom.
* Run As -> Run Configurations -> en el apartado Goals indicamos lo que queremos ejecutar.



En base directory indicamos desde dónde lo queremos ejecutar.

En Goals, indicamos el qué queremos ejecutar. En nuestro caso, es nuestro MOJO. Notar que cada uno de los ítems de la sentencia están separados por dos puntos (:). Que no se nos olvide el goal al final, que es touch en este ejemplo.

Aplicamos y procedemos a lanzarlo obteniendo el siguiente resultado en consola:



## Creación de nuestro proyecto usando nuestro MOJO desde eclipse

* File -> New -> Proyect -> Maven -> Majen Proyect.
* Buscar el arquetipo que queremos incluir.

En nuestro caso, hemos usado maven-archetype-quickstart (el de apache).

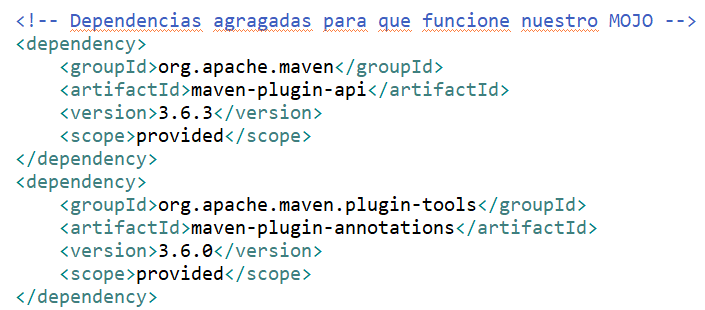
* Indicar los nombres:
* Groupe Id: (com.soprasteria -> el nombre de la organización invertido).
* Artefact Id: (EjemploCreacionMojo -> el nombre del proyecto).
* Version: (0.0.1-SNAPSHOT es la que viene por defecto).
* Package: com.soprasteria.EjemploCreacionMojo (se rellena solo).

En este punto ya lo tenemos creado.

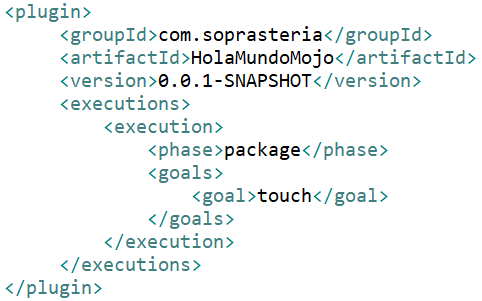
Podemos observar que consta de una colección de carpetas que se agrupan en src y un archivo llamado pom.xml. Se trata del archivo de configuración del proyecto.

En este punto, tenemos que añadir varios mandatos en el pom para poder usar nuestro MOJO.

En primer lugar, necesitamos incluir un par de dependencias para que funcione. Son las que se pueden ver a continuación:



Después, tenemos que crear el plugin que lo ejecute:

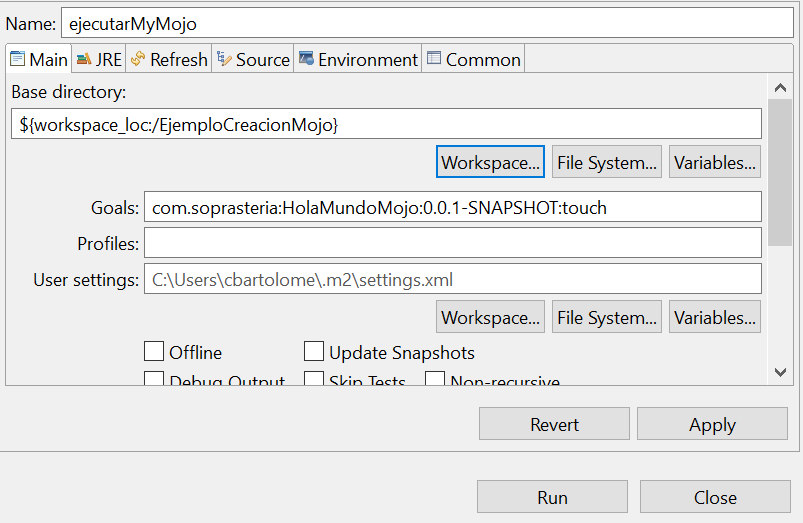


Indicamos su localización en la etiqueta groupId, el nombre de la carpeta que lo contiene en artifactId, y la versión.

Debajo ponemos el momento en el que se va a ejecutar, que en este caso es durante la etapa de package e indicamos el goal que lo identifica, que es touch (aparece en el comentario de la clase del MOJO).

Ahora lo que nos queda es generarlo.

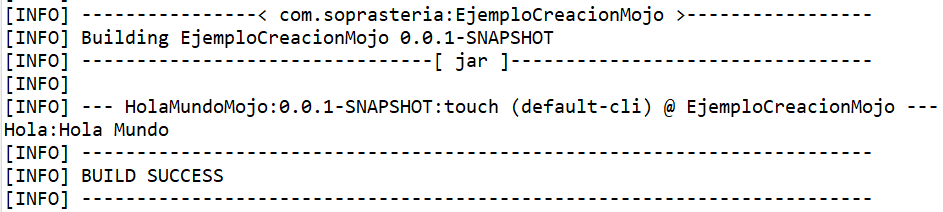
* Nos posicionamos en el proyecto o en el pom.
* Run As -> Run Configurations -> en el apartado Goals indicamos lo que queremos ejecutar.



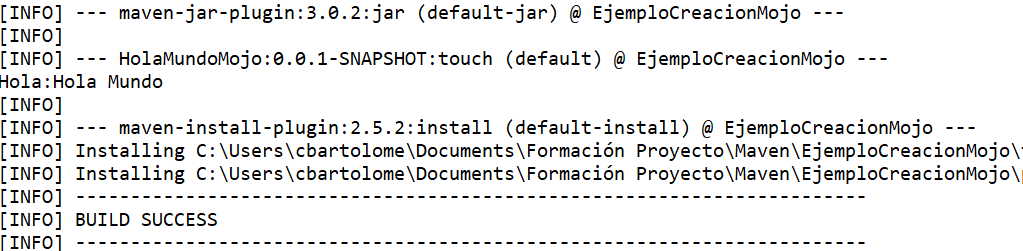
En base directory indicamos desde dónde lo queremos ejecutar.

En Goals, indicamos el qué queremos ejecutar. En nuestro caso, es nuestro MOJO. Notar que cada uno de los ítems de la sentencia están separados por dos puntos (:). Que no se nos olvide el goal al final, que es touch en este ejemplo.

Aplicamos y procedemos a lanzarlo obteniendo el siguiente resultado en consola:



En este caso, solo hemos lanzado el goal touch pero se puede ver también haciendo otras etapas del ciclo de vida.



En este caso, hemos lanzado un Maven install y se puede ver que nuestro MOJO se ejecuta durante el package (en este caso jar).

* Profile

Configuración de Maven que sólo queremos aplicar en determinadas circunstancias.

Los profiles pueden ser definidos de forma específica para un proyecto en concreto dentro del pom.xml o bien de forma general dentro del archivo settings.xml de Maven.

Vamos a ver un ejemplo. En él, vamos a crear dos profiles diferentes (en nuestro caso el profile desarrollo y el profile pre) que se van a ejecutar en diferentes etapas del ciclo de vida. Para ello, vamos a servirnos del uso de los MOJOs, que es donde tendremos su funcionamiento específico.

## Creación de los Mojos

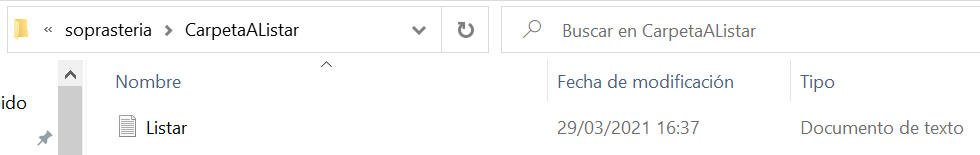
Creamos dos MOJOs de la misma manera en la que lo hemos hecho antes.

También se podría hacer un solo proyecto que los contuviera a ambos, funcionaría igual.

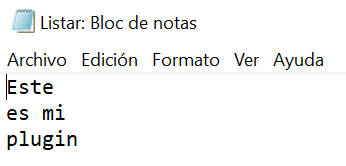
A continuación, vamos a darle a cada uno su funcionalidad.

El primer MOJO va a listar los archivos contenidos dentro de un directorio.

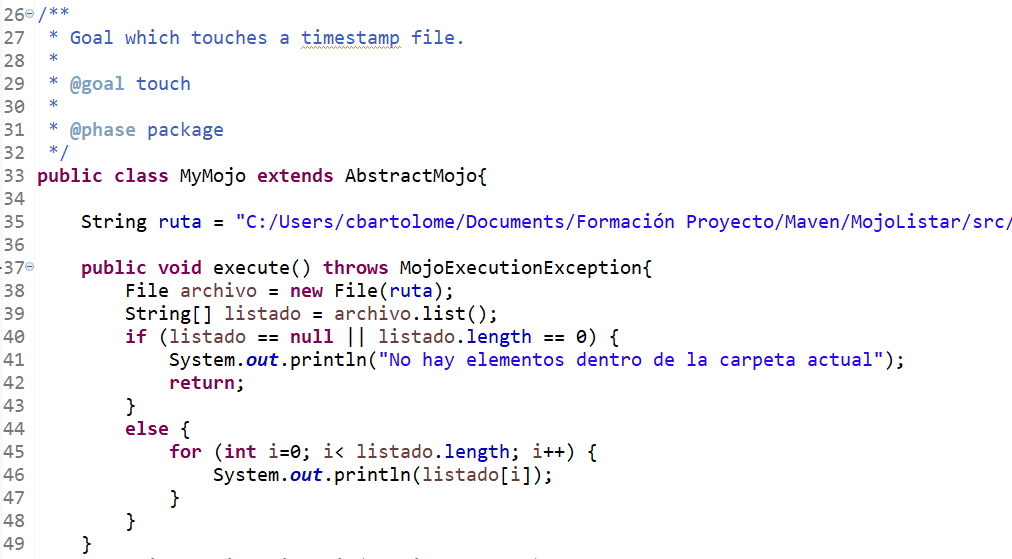
En este caso, solo va a contener uno que hemos llamado Listar.txt.



Su contenido se puede ver a continuación:



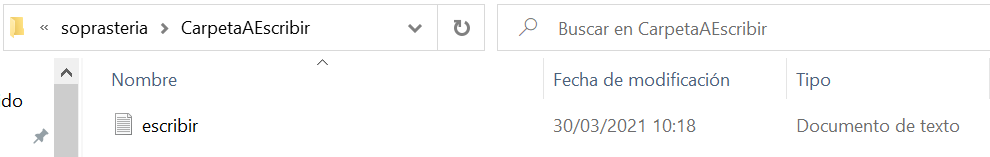
El código de nuestro MojoListar es el siguiente:



Se puede apreciar que el goal es touch y la fase en la que se va a ejecutar es la de package.

En cambio, nuestro segundo MOJO va a escribir dentro de un archivo de una ruta dada.

Si el archivo inicialmente no existe, hay que crearlo.



El código de nuestro MojoEscribir es el que sigue:



Se puede ver al inicio que nuestro goal es touch y la fase en la que se va a llevar a cabo es la de install.

Una vez los tenemos a ambos, hacemos Maven install para tenerlos en nuestro repositorio local, .m2.

## Creación de los profiles

En primer lugar, crearemos un nuevo proyecto de la misma forma que las veces anteriores, usando el arquetipo quickstart.

Como hemos dicho antes, los profiles pueden ir en el pom o en settings, pero en nuestro caso vamos a crearlos dentro de nuestro proyecto.

En primer profile, tiene el siguiente código:



En id, hemos escrito el nombre del profile, en este caso es desarrollo.

Después tenemos que indicar dónde se encuentra el plugin que vamos a utilizar. En nuestro caso, es el MojoListar que acabamos de crear,

Además, debemos indicar la etapa en la que se va a ejecutar y el goal.

El segundo profile es muy parecido, ya que solo cambia el nombre que en este caso es pre, la localización del plugin que es nuestro MojoEscribir y la etapa en la que se va a llevar a cabo (package):

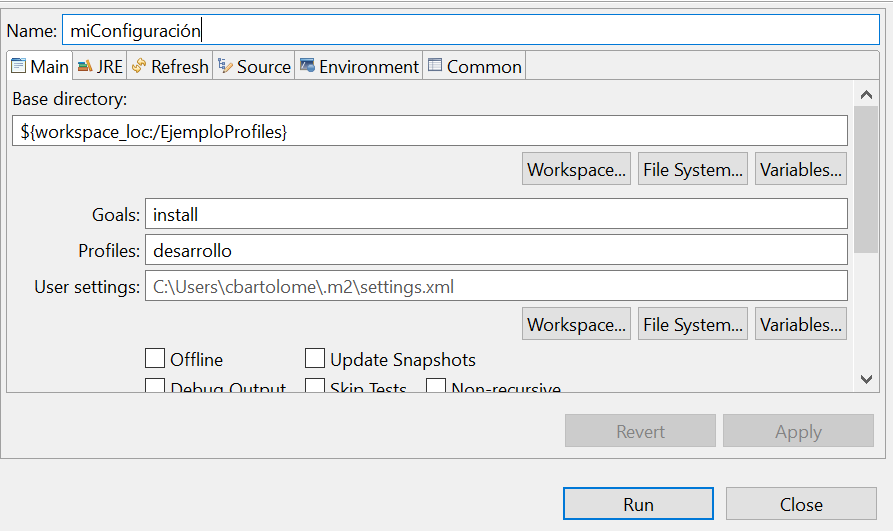


## Ejecución de los profiles

Como hemos dicho anteriormente, cada uno se va a ejecutar durante una etapa, y nunca los dos a la vez.

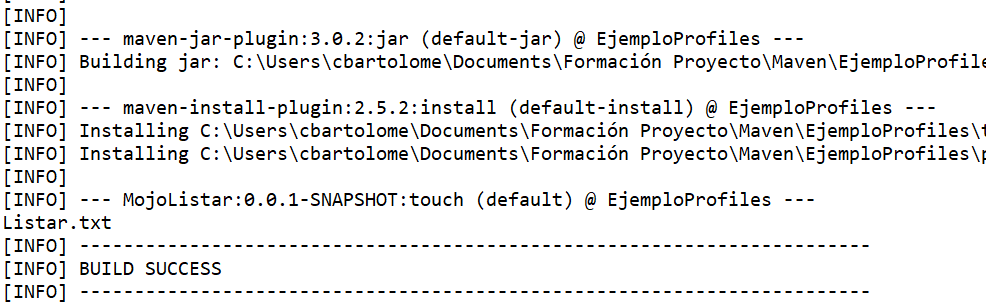
Si lo que queremos ejecutar es nuestro profile desarrollo, tenemos que proceder de la siguiente manera:

* Nos posicionamos en el proyecto o en el pom.
* Run As -> Run Configurations



En el apartado Goals indicamos la fase, y dentro de Profiles, el id del que queremos ejecutar.

Al aplicar, podemos ver el resultado por consola:

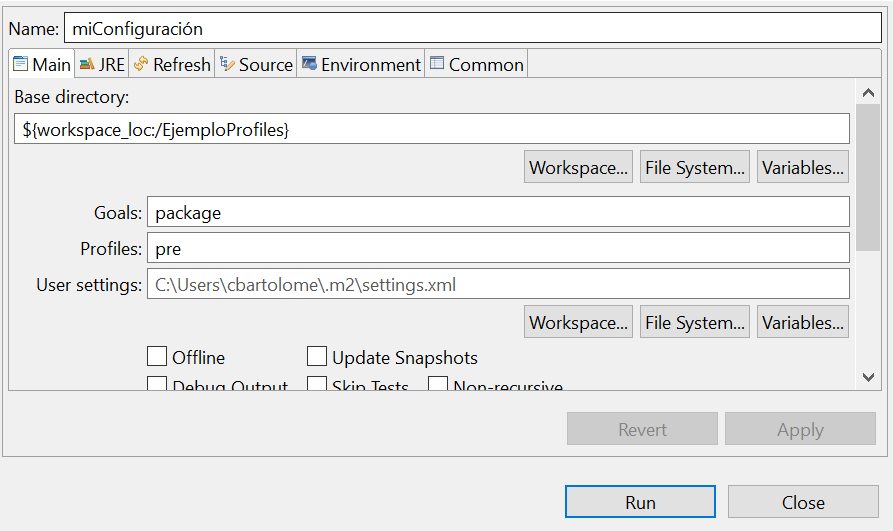


Se observa que durante el package, que por defecto es jar, no se ejecuta nuestro MojoEscribir.

Además, durante la fase del install sí que se lleva a cabo nuestro MojoListar, que es el que corresponde al profile desarrollo, el que hemos indicado.

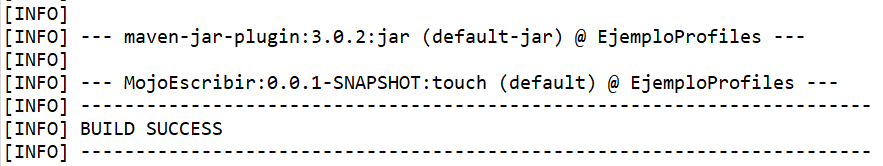
Si lo que queremos es ejecutar nuestro profile pre, se hace de la siguiente manera:

* Nos posicionamos en el proyecto o en el pom
* Run As -> Run Configurations



En el apartado Goals indicamos la fase, y dentro de Profiles, el id del que queremos ejecutar.

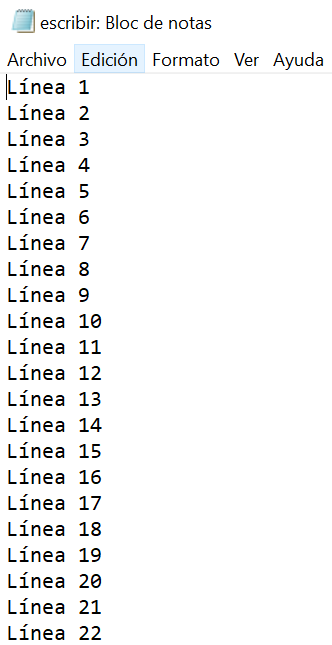
Al aplicar, podemos ver el resultado por consola:



Se observa que durante el package se ejecuta nuestro MojoEscribir.

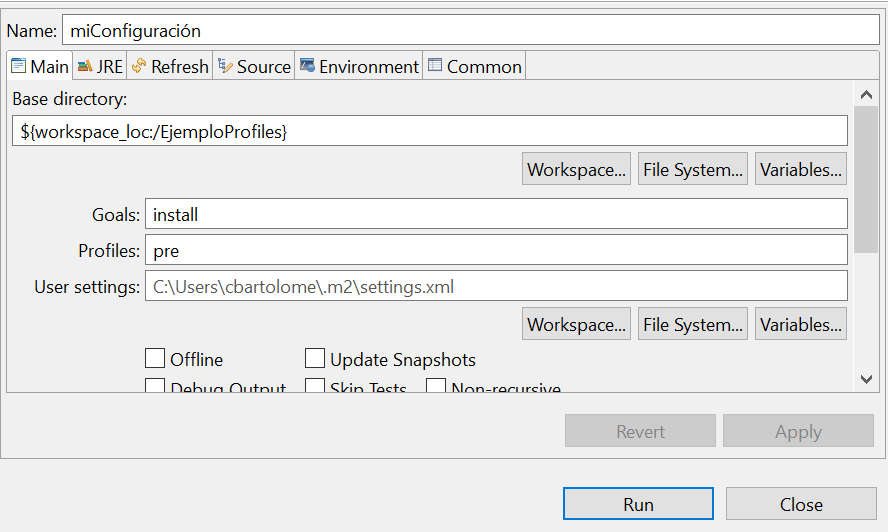
La fase de install es posterior, por lo que no tenemos nada más.

Nuestro archivo escribir se vería de la siguiente manera:

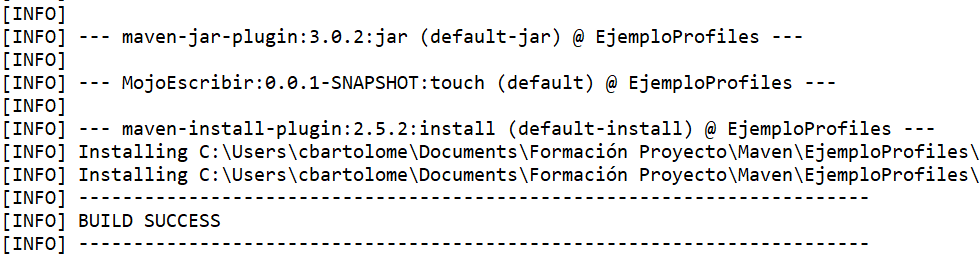


Luego se ha rellenado correctamente.

Si ahora queremos ver que durante la ejecución de este profile no se llevan a cabo ambos, vamos a indicar que la fase ahora sea install.



Lo lanzamos y miramos el resultado obtenido en la consola:



Como queríamos, solo se ha ejecutado nuestro MojoEscribir durante la etapa del package. Aunque hayamos ejecutado install, no nos lista el contenido del directorio.

Luego el uso de los profiles funciona perfectamente