GESTORES DE VERSIONAMIENTO Y DEPENDENCIA DE CÓDIGO FUENTE

por

Jaider Fabian Molina

Luis Fernando Rios

Profesor:

Robinson Coronado Garcia

Arquitectura de Software - 20201

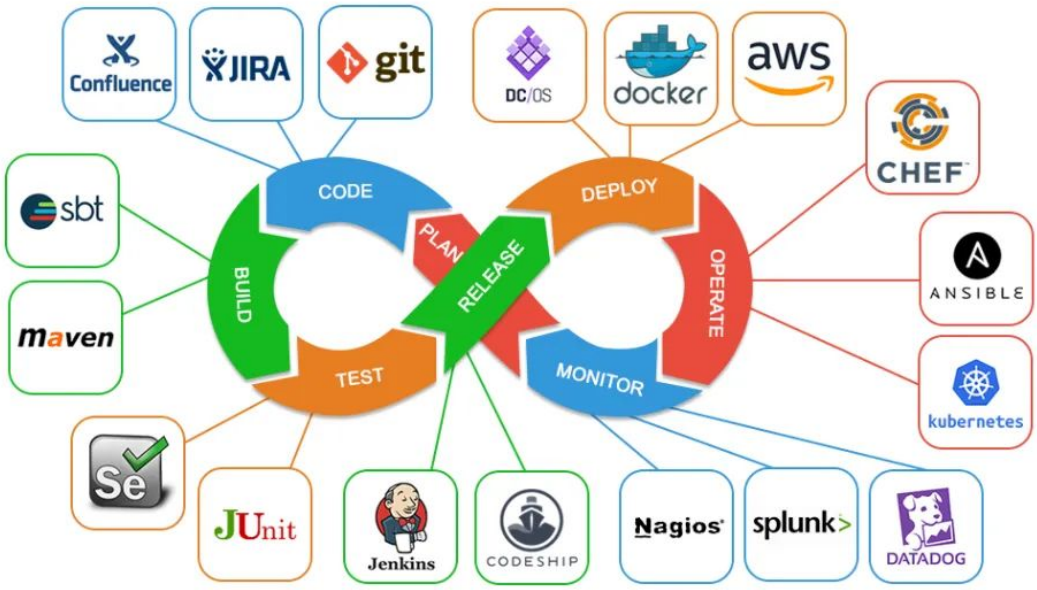
UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS
2020

GESTORES DE VERSIONAMIENTO Y DEPENDENCIA DE CÓDIGO FUENTE

Introducción

El desarrollo de software ha venido evolucionando durante los últimos 60 años donde inicialmente se hacía de manera empírica y sin ninguna metodología estándar. A raíz de problemas de tiempos de entrega, costo y calidad se fueron creando metodologías para solucionar estos problemas. Actualmente tiene mucha acogida las metodologías de desarrollo ágil donde los equipos de trabajo deben tener altas habilidades de comunicación y coordinación. Por lo tanto dentro del ciclo de desarrollo donde intervienen muchas personas para hacer entregas más rápidas se diseñaron herramientas devops, entre ellas el control de versiones, que ayudan a la gestión de la integración de código de un grupo de desarrolladores para tener alta respuesta y una buena administración de la calidad de este. Entre estas herramientas encontramos las más populares como Mercurial, CVS, Monotone, Jira y de la que hablaremos Git .

Adicionalmente dentro de las herramientas devops también encontramos los gestores de dependencias, que ayudan al equipo de desarrollo a mantener la cohesión de las versiones de frameworks y librerías en las que se apoya el app en desarrollo para que sea funcional en el tiempo y evite problemas al momento del despliegue, entre estos tenemos a Maven y Gradle.



Objetivo General

Conocer las herramientas dentro del devops (Desarrollo de software y Operaciones) para la gestión de versiones y dependencias del código en marcos de trabajo ágil.

Objetivos Específicos

- Saber que es Git, sus características, para que se aplica y cómo funciona.
- Conocer qué es Maven y Gradle, sus diferencias y cómo funciona.

Gestor de Versiones GIT

Git es un sistema de gestión de proyectos y control de versiones de código, así como una plataforma de red social diseñada para desarrolladores. ¿Pero para qué se usa GitHub? Bueno, en general, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo. Es un modelo de repositorio distribuido compatible con sistemas y protocolos existentes como HTTP, FTP, SSH y es capaz de manejar eficientemente proyectos pequeños a grandes. Fue desarrollado por Linus Torvals, el mismo padre del kernel Linux, en el año 2005. Git surge como alternativa a BitKeeper, un control de versiones privado que usaba en ese entonces para el kernel. Es liberado bajo una licencia GNU GPLv2 y su última versión estable fue publicada a inicios de Abril de 2020. Se ha convertido en uno de los más usados alrededor del mundo.

Características fundamentales

- Resolución de conflictos: Es muy probable que los miembros del equipo tengan la necesidad de realizar cambios en el mismo archivo de código fuente al mismo tiempo. Un VCS monitoriza y ayuda a poder resolver los conflictos entre varios desarrolladores.

- Revertir y deshacer los cambios en el código fuente: Al empezar a monitorizar un sistema de archivos de códigos fuente, existe la posibilidad de revertir y deshacer rápidamente a una versión estable conocida.

- Copia de seguridad externa del código fuente: Se debe crear una instancia remota del VCS que se puede alojar de forma externa con un tercero de confianza y con ello, se conservará una copia del código fuente.

Git dispone de gran libertad en la forma de trabajar en torno a un proyecto. Sin embargo, para coordinar el grupo de trabajo en un desarrollo de app es necesario acordar como se va a trabajar con Git. A estos acuerdos se les llama flujo de trabajo. Un flujo de trabajo de Git es una fórmula o una recomendación acerca del uso de Git para realizar trabajo de forma uniforme y productiva. Los flujos de trabajo más populares son git-flow, GitHub-flow, GitLab Flow y One Flow.

Los principales comandos son:

- git help: Muestra una lista con los comandos más utilizados en GIT.
- git init: Podemos ejecutar ese comando para crear localmente un repositorio con GIT y así utilizar todo el funcionamiento que GIT ofrece. Basta con estar ubicados dentro de la carpeta donde tenemos nuestro proyecto y ejecutar el comando. Cuando agreguemos archivos y un commit, se va a crear el branch master por defecto.
- git add + path: Agrega al repositorio los archivos que indiquemos.
- git add -A: Agregar al repositorio TODOS los archivos y carpetas que estén en nuestro proyecto, los cuales GIT no está siguiendo.
- git commit -m "mensaje" + archivos: Hace commit a los archivos que indiquemos, de esta manera quedan guardados nuestras modificaciones.
- git commit -am "mensaje": Hace commit de los archivos que han sido modificados y GIT los está siguiendo.
- git checkout -b NombreDeBranch: Crea un nuevo branch y automáticamente GIT se cambia al branch creado, clonando el branch desde donde ejecutamos el comando.
- git branch: Nos muestra una lista de los branches que existen en nuestro repositorio.
- git checkout NombreDeBranch: Sirve para moverse entre branches, en este caso vamos al branch que indicamos en el comando.
- git merge NombreDeBranch: Hace un merge entre dos branches, en este caso la dirección del merge sería entre el branch que indiquemos en el comando, y el branch donde estemos ubicados.
- git status: Nos indica el estado del repositorio, por ejemplo cuales están modificados, cuales no están siendo seguidos por GIT, entre otras características.
- git clone URL/name.git NombreProyecto: Clona un proyecto de git en la carpeta NombreProyecto.
- git push origin NombreDeBranch: Luego de que hicimos un git commit, si estamos trabajando remotamente, este comando va a subir los archivos al repositorio remoto, específicamente al branch que indiquemos.
- git pull origin NombreDeBranch: Hace una actualización en nuestro branch local, desde un branch remoto que indicamos en el comando.

GitHub es también uno de los repositorios online más grandes de trabajo colaborativo en todo el mundo que usa la tecnología Git. Otras plataformas que usan Git son Gitlab, Bitbucket.

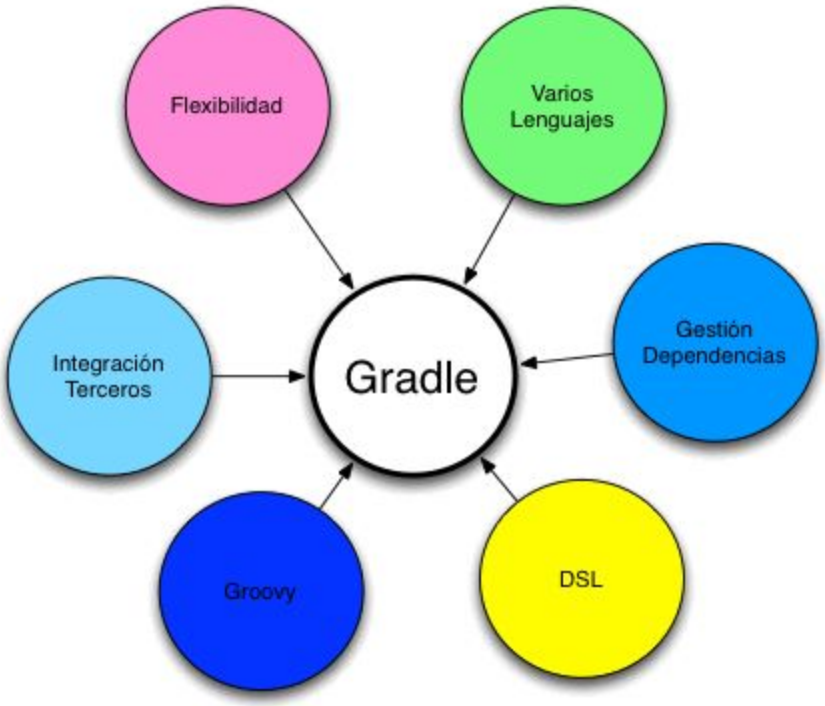
Gestor de dependencias GRADLE

Gradle es una herramienta de automatización de la construcción (build) de nuestro código y de gestión de dependencias sólido que hereda las aportaciones que han realizado herramientas como ant y maven pero intenta llevarlo todo un paso más allá. Para empezar se apoya en Groovy y en un DSL (Domain Specific Language) para trabajar con un lenguaje sencillo y claro a la hora de construir el build comparado con Maven. Por otro lado, dispone de una gran flexibilidad que permite trabajar con ella utilizando otros lenguajes y no solo Java. Los plugins iniciales están principalmente centrados en el desarrollo y despliegue en Java, Groovy y Scala, pero existen más lenguajes y workflows de proyecto en el futuro.

Características de Gradle

- Depuración colaborativa: Permite compartir los resultados de la compilación para resolver en equipo de forma eficiente posibles problemas que aparezcan.
- Construcción incremental: Válida en el proceso de compilación si la entrada, salida o implementación de una tarea ha cambiado, en caso de no existir algún cambio la considera actualizada y no se ejecuta.
- Diseño de repositorio personalizado: Podremos tratar prácticamente cualquier estructura de directorios del sistema de archivos como un repositorio de Artifacts.
- Dependencias transitivas: Es uno de los principales beneficios que ofrece al utilizar la gestión de dependencias ya que se encarga de descargar y administrar las dependencias transitivas.
- Soporte a Groovy y Scala incorporado: Compatibilidad con los proyectos de Groovy, permitiendo trabajar con código Groovy o código Scala e inclusive desarrollar código mixto Java y Groovy o Java y Scala.
- Compilación incremental para Java: En caso de que el código fuente o la ruta de clase cambien, Gradle cuenta con la capacidad para detectar todas las clases que se vean afectadas por dicho cambio y procederá a recompilar las.
- Embalaje y distribución de JAR, WAR y EAR: Cuenta con herramientas para empaquetar el código basado en JVM (Java Virtual Machine) en archivos de archivo comunes.
- Integración con Android Studio: Android Studio no cuenta con un generador interno, sino que delega todas las tareas de compilación en Gradle, garantizando la corrección en todas las construcciones, ya sea que se ejecuten desde Android Studio, la línea de comandos o un servidor de construcción de integración continua.
- Soporte de MS Visual C ++ y GoogleTest: Gradle acepta la construcción con el compilador de Visual C de Microsoft en Windows. (VS 2010, VS 2013 y VS 2015 compatibles), así como también realizar pruebas de aplicaciones C con GoogleTest.

- Publicar en repositorios Ivy y Maven: Permite publicar Artifacts en repositorios Ivy con diseños de directorios completamente personalizables. De igual modo, sucede con Maven en Bintray o Maven Central.
- TestKit para pruebas funcionales: Permite la ejecución programática de builds inspeccionando los resultados de compilación, ésta es una prueba de compatibilidad entre versiones.
- Distribuciones personalizadas: En Gradle cada distribución cuenta con un directorio init.d en el que se pueden colocar scripts personalizados que pre configuran su entorno de compilación.
- Lee el formato POM: Es compatible con el formato de metadatos POM, por lo que es posible recuperar dependencias de cualquier repositorio compatible con Maven.
- Compara builds: Resalta de forma rápida las diferencias entre compilaciones, lo que hace que el análisis de la causa raíz sea mucho más rápido y eficaz.
- Compilador daemon: Gradle crea un proceso de daemon que se reutiliza dentro de una compilación de múltiples proyectos, cuando necesita bifurcar el proceso de compilación, mejorando la velocidad de compilación.
- Personalizar y extender escaneos: Ofrece la opción de agregar sus propios datos para construir escaneos como etiquetas, valores y enlaces, integrando escaneos de compilación en la cadena de herramientas.
- Caché de dependencia de terceros: Las dependencias de repositorios remotos se descargan y almacenan en caché localmente, las compilaciones posteriores utilizan los artifacts almacenados en caché para evitar el tráfico de red innecesario.



Ejemplo archivo build.gradle:

```
/*
 * This file was generated by the Gradle 'init' task.
 */
plugins {
    // Apply the java plugin to add support for Java
    id 'java'
    // Apply the application plugin to add support for building a CLI application.
    id 'application'
}
repositories {
    // Use jcenter for resolving dependencies.
    jcenter()
}
dependencies {
    // This dependency is used by the application.
    implementation 'com.google.guava:guava:28.0-jre'
    // Use JUnit test framework
    testImplementation 'junit:junit:4.12'
}

application {
    // Define the main class for the application.
    mainClassName = 'co.edu.udea.arquisoft.app'
}
```

Gestor de dependencias MAVEN

Maven, una palabra yiddish que significa acumulador de conocimiento , comenzó como un intento de simplificar los procesos de construcción en el proyecto Jakarta Turbine. Había varios proyectos, cada uno con sus propios archivos de compilación Ant, que eran todos ligeramente diferentes. Los JAR se registraron en CVS. Queríamos una forma estándar de construir los proyectos, una definición clara de en qué consistía el proyecto, una forma fácil de publicar información del proyecto y una forma de compartir los archivos JAR en varios proyectos.

El resultado es una herramienta que ahora se puede utilizar para crear y gestionar cualquier proyecto basado en Java.

Objetivos de Maven

El objetivo principal de Maven es permitir que un desarrollador comprenda el estado completo de un esfuerzo de desarrollo en el período de tiempo más corto. Para lograr este objetivo, Maven se ocupa de varias áreas de interés:

- Facilitando el proceso de construcción
- Proporcionar un sistema de construcción uniforme
- Proporcionar información de proyectos de calidad
- Fomentar mejores prácticas de desarrollo

Facilitando el proceso de construcción: Si bien el uso de Maven no elimina la necesidad de conocer los mecanismos subyacentes, Maven protege a los desarrolladores de muchos detalles.

Proporcionar un sistema de construcción uniforme: Maven crea un proyecto utilizando su modelo de objetos de proyecto (POM) y un conjunto de complementos. Una vez que se familiarice con un proyecto de Maven, sabrá cómo se construyen todos los proyectos de Maven. Esto ahorra tiempo al navegar por muchos proyectos.

Características:

- Configuración de proyecto simple que sigue las mejores prácticas: inicie un nuevo proyecto o módulo en segundos
- Uso constante en todos los proyectos: significa que no hay tiempo de aceleración para los nuevos desarrolladores que ingresan a un proyecto
- Gestión superior de dependencias que incluye actualización automática, cierres de dependencias (también conocidas como dependencias transitivas)
- Capaz de trabajar fácilmente con múltiples proyectos al mismo tiempo

- Un repositorio grande y creciente de bibliotecas y metadatos para usar fuera de la caja, y acuerdos establecidos con los proyectos de código abierto más grandes para la disponibilidad en tiempo real de sus últimas versiones.
- Extensible, con la capacidad de escribir complementos fácilmente en Java o lenguajes de scripting
- Acceso instantáneo a nuevas funciones con poca o ninguna configuración adicional
- Tareas de Ant para la gestión de dependencias y la implementación fuera de Maven
- Compilaciones basadas en modelos: Maven puede compilar cualquier número de proyectos en tipos de salida predefinidos como JAR, WAR o distribución basada en metadatos sobre el proyecto, sin la necesidad de realizar ningún script en la mayoría de los casos.
- Sitio coherente de información del proyecto: utilizando los mismos metadatos que para el proceso de compilación, Maven puede generar un sitio web o PDF que incluya cualquier documentación que desee agregar, y agrega informes estándar sobre el estado de desarrollo del proyecto. Se pueden ver ejemplos de esta información en la parte inferior de la navegación de la izquierda de este sitio en los submenús "Información del proyecto" e "Informes del proyecto".
- Publicación de distribución y administración de versiones: sin mucha configuración adicional, Maven se integrará con su sistema de control de código fuente (como Subversion o Git) y administra la publicación de un proyecto basado en una etiqueta determinada. También puede publicar esto en una ubicación de distribución para que lo utilicen otros proyectos. Maven puede publicar salidas individuales, como un JAR, un archivo que incluye otras dependencias y documentación, o como una distribución de origen.
- Gestión de dependencias: Maven fomenta el uso de un repositorio central de JAR y otras dependencias. Maven viene con un mecanismo que los clientes de su proyecto pueden usar para descargar cualquier JAR requerido para construir su proyecto desde un repositorio JAR central muy parecido al CPAN de Perl. Esto permite a los usuarios de Maven reutilizar los archivos JAR en todos los proyectos y fomenta la comunicación entre proyectos para garantizar que se resuelvan los problemas de compatibilidad con versiones anteriores.

Ejemplo archivo pom.xml

```
<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>co.edu.udea.arquisoft</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>
```