Investigación Git, Maven y Gradle

Por:

Anderson Barrientos Parra
C.c: 1017242181
Stiven Guerra Chaverra
C.c: 1037672655
Sebastián Gómez Ramírez
C.c: 1045026756
Hermen Esteban Theran Martínez
C.c: 1035875072

Robinson Coronado García
Profesor

Arquitectura de Software - Ingeniería de Sistemas
Universidad De Antioquia
Medellín 2020

Introducción

La presente investigación es sobre las herramientas de software Git, Maven y Gradle, las cuales son herramientas bastante útiles y usadas en el desarrollo de software. Se analizarán las funcionalidades y formas de uso de las herramientas anteriormente mencionadas.

Git

¿Qué es Git?

El sistema de control de versiones llamado *Git*, es usado para controlar el flujo de trabajo y los cambios hechos entre versiones de un proyecto, este permite controlar las modificaciones realizadas entre distintas versiones de un mismo proyecto.

¿En qué consiste?

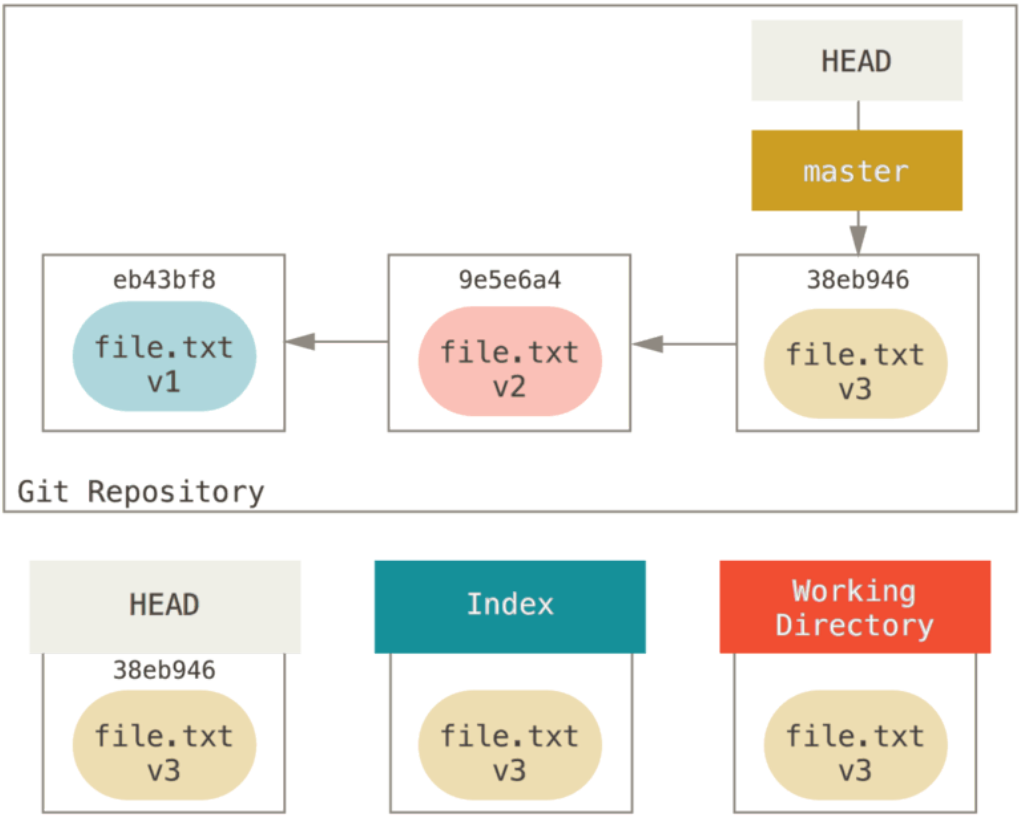
Git guarda los cambios que vamos haciendo en un proyecto, y permite hacer seguimiento de estos, controlando los cambios hechos entre una versión y otra, y además regresar a versiones anteriores, posteriores o alternativas de este.

Los proyectos en github manejan 3 “árboles” (Sectores) para guardar y garantizar una buena condición del proyecto:

1. **Área de trabajo (working area)**: Área donde se harán todos los cambios del proyecto (caja de arena)
2. **Index (staging):** Área de preparación preparación para publicar un nuevo paquete con cambios
3. **HEAD (Local repository):** Puntero que hace referencia al último paquete confirmado de cambios (commit), puede verse como la sección final donde se almacenan los commits

En el área de trabajo, git detecta todos los cambios que se hagan en el proyecto donde se está trabajando, git no se fija en los nombres de los archivos, sino en las modificaciones hechas.
En el index preparamos nuestra nueva versión del proyecto, aquí agregamos todos los cambios que hagamos en el área de trabajo y que queremos que perduren.
En estas 2 áreas los datos no están totalmente seguros, pues al cambiar de rama se perderán, para evitar esto hay que confirmarlos (commit).
Este commit se guarda en el historial de versiones (local repository) a donde apunta HEAD.

En términos generales, podemos hacer una metáfora con el funcionamiento de un pc, donde hacemos cambios (working area) que quedan guardados temporalmente en la memoria ram (Index) y que deben ser guardados en el repositorio local para que estos perduren.



[1]

¿Cómo trabajar con git?

- Descargar el controlador de versiones Git de la página: https://git-scm.com/

- Para que git pueda trabajar de una buena manera, se deben establecer los parámetros en un archivo oculto llamado **.gitconfig**, el cual en sistemas linux/mac se encuentra alojado en la carpeta raíz del sistema en windows se encuentra en la ruta "C:\users\$USUARIO\.gitconfig", podemos agregar las configuraciones aquí directamente, o bien agregarlas por la consola con los siguientes códigos (opción recomendada):

    - **git config --global user.name $NOMBRE_DE_USUARIO** -> Configura el nombre de usuario con el que quedan registrados los cambios

    - **git config --global user.email $E-MAIL** -> Configura el correo del usuario con el que quedan registrados los cambios

    - **git config --global core.editor $EDITOR_ESCOGIDO** -> Configura el editor de texto predeterminado de git
      ejm: git config --global core.editor "code --wait"

    - **git config --global merge.tool $HERRAMIENTA_DE_FUSION** -> Configura la herramienta que se usará para resolver conflictos al fusionar 2 ramas

ejm: git config --global merge.tool meld

- **git config --global mergetool.meld.path "$RUTA_DE_HERRAMIENTA"** -> Configura la ruta de la herramienta de merge escogida
  ejm: git config --global mergetool.meld.path "C:\Program Files (x86)\Meld\Meld.exe"

- **git config --global diff.tool $HERRAMIENTA_DE_DIFFERENCIACION** -> Configura la herramienta que se usará para mirar las diferencias entre 2 ramas
  ejm: git config --global diff.tool meld

- **git config --global difftool.meld.path "$RUTA_DE_HERRAMIENTA"** -> Configura la ruta de la herramienta de diff escogida
  ejm: git config --global difftool.meld.path "C:\Program Files (x86)\Meld\Meld.exe"

- **git config --global difftool.prompt false** -> Configura la consola para que no requiera confirmación antes de abrir determinado archivo

- Los siguientes son los comandos básicos para trabajar con un repositorio controlado por Git:

  - **git config --edit** -> Abre el archivo de configuración con la herramienta configurada

  - **git init** -> Crear un repositorio en la ubicación actual

  - **git remote add origin $URL_REPOSITORIO_REMOTO** -> Asocia un repositorio local con un repositorio remoto

  - **git push --set-upstream origin $NOMBRE_RAMA / git push -u origin $NOMBRE_RAMA** -> Sube un commit al repositorio remoto asociando la rama local con la rama remota indicada
    ejm: git push -u origin master -> en este ejemplo se asocia la rama actual con la rama remota **master**, por lo general se usa este comando en el proceso de asociación de un repositorio local con uno remoto

  - **git clone $URL_REPOSITORIO_REMOTO** -> Descarga un repositorio de la nube

  - **git status** -> Ver estado actual de una rama (archivos en working dir, en index y HEAD)

  - **git log** -> Muestra el registro de commits hechos en el repositorio

  - **git tag $NOMBRE_TAG $ID_COMMIT** -> Le pone el identificador indicado al commit indicado

- **git add $NOMBRE_ARCHIVO_RUTA_INCLUIDA** -> Agrega un archivo específico al index

- **git add --all / git add .** -> Agrega todos los archivos que están en el working dir al index

- **git add $NOMBRE_CARPETA/** -> Agrega una carpeta entera (con los archivos contenidos) al index

- **git commit -m "$MENSAJE"** -> Confirma un paquete de archivos, agrega y empaqueta los archivos del index en HEAD, generando así una nueva "versión" del proyecto

- **git reset $NOMBRE_ARCHIVO_RUTA_INCLUIDA** -> Elimina un archivo específico del index (copia el que está en HEAD en el index)

- **git reset** -> Elimina todos los archivos del index (copia los que están en HEAD en el index)

- **git reset --hard HEAD** -> Elimina todos los archivos del index y del working dir (copia el que está en HEAD en el index y en working dir)

- **git reset --hard $ID_COMMIT** -> Elimina todos los archivos del index y del working dir a como estaban en determinado commit (copia el que está dentro del commit indicado en el index y en working dir)

- **git reset --hard $NOMBRE_RAMA** -> Elimina todos los archivos del index y del working dir a como estaban en determinada rama (copia el que está dentro de la rama indicada en el index y en working dir)

- **git rm $NOMBRE_ARCHIVO_RUTA_INCLUIDA** -> Elimina un archivo completamente (disco duro)

- **git rm --cached $NOMBRE_ARCHIVO_RUTA_INCLUIDA** -> Elimina un archivo del repositorio pero no del disco duro, útil cuando se agregó un archivo a un commit que no debía estar

- **git stash save** -> Guarda todos los cambios temporalmente en una pila aparte del repositorio, útil para cuando no se desea pasar al index los cambios, pero es requerido trasladarse a otra rama

- **git stash pop** -> Retorna los cambios que fueron guardados temporalmente

- **git branch** -> Muestra las ramas locales

- **git branch --all** -> Muestra todas las ramas del proyecto (incluyendo las remotas)

- **git branch -d $NOMBRE_RAMA** -> Elimina la rama indicada

- **git checkout -b $NOMBRE_RAMA** -> Crea una nueva rama con el nombre indicado y con los cambios de la rama actual (genera una copia de la rama actual con el nombre dado)

- **git checkout $NOMBRE_RAMA** -> Cambia de rama (reemplaza todos los archivos por los de la rama indicada)

- **git checkout $ID_COMMIT** -> Cambia de commit (reemplaza todos los archivos por los del commit indicado)

- **git checkout -- $NOMBRE_ARCHIVO** -> Reemplaza el archivo indicado por el que está en HEAD (restaura el archivo al último commit)

- **git checkout $NOMBRE_RAMA -- $NOMBRE_ARCHIVO** -> Reemplaza el archivo indicado por el que está en la rama indicada

- **git checkout $ID_COMMIT -- $NOMBRE_ARCHIVO** -> Reemplaza el archivo indicado por el que está en el commit indicado

- **git pull** -> Descarga los cambios de la rama remota a la local

- **git pull origin $NOMBRE_RAMA** -> Descarga los cambios de la rama remota indicada a la rama local actual (indica si hay conflictos al hacer merge automático)

- **git push** -> Sube los cambios de la rama local a la remota

- **git mergetool** -> Abre la herramienta de merge para solucionar conflictos al hacer merge entre 2 ramas

- **git difftool** -> Abre la herramienta de diff para ver los cambios entre HEAD y el estado actual de los archivos

- **git difftool $NOMBRE_RAMA_A $NOMBRE_RAMA_B** -> Abre la herramienta de diff para ver los cambios entre la rama a y la rama b

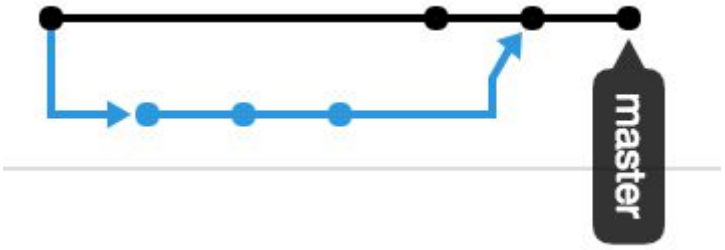
- **git diff $NOMBRE_RAMA_A $NOMBRE_RAMA_B** -> Ver diferencias entre 2 ramas

Flujo de trabajo:

Cuando se trabaja con Git, lo ideal es que cada commit tenga una funcionalidad específica implementada, es decir, cada nueva funcionalidad por más pequeña que sea debe tener un commit para sí misma. Esto se debe a la facilidad para controlar los cambios y revertirlos en caso de algún conflicto con los cambios hechos en determinada versión.

Otro punto importante a la hora de trabajar con Git, son las ramas (branch), consisten en copias del estado en el que se encontraba un proyecto en el momento de creación de la nueva rama. Estas ramas nos permiten implementar funcionalidades grandes, o realizar cambios drásticos en el proyecto sin peligro de afectar el proyecto principal.
Toda funcionalidad que se vaya a implementar, debe estar en su rama específica, y cuando se termine de implementar los cambios, se debe hacer un Pull Request (PR) hacia la rama que le siga en la jerarquía.

Un PR consiste en una petición a la rama principal para fusionar los cambios hechos en determinada rama, Cada PR debe identificar bien los cambios que se hicieron y constatar que son funcionales.



[2]

Git ignore:

En todo repositorio, hay archivos que no queremos guardar en el repositorio, ya sea local o remoto. Para esto, existe un archivo llamado **.gitignore,** este archivo guarda dentro de sí los nombres, rutas y extensiones de los archivos que queremos obviar en nuestro repositorio.

Este archivo contiene texto plano, aunque se requiere una sintaxis para indicar los archivos a ignorar en el repositorio:

- **$NOMBRE_CARPETA/** -> Indica la carpeta que quiere que sea ignorada en el repositorio
- **$NOMBRE_ARCHIVO_RUTA_INCLUIDA** -> Indica un archivo específico para ser ignorado
- ***.$EXTENSION_ARCHIVO** -> Indica que se ignorarán todos los archivos con la extensión indicada
- **#$COMENTARIO** -> Permite agregar comentarios en el archivo

GitHub:

GitHub es uno de los repositorios remotos globales más populares del momento para montar proyectos utilizando Git, aparte de ser una red social para programadores y un conjunto de servicios relacionados a los proyectos alojados.

Permite a sus usuarios compartir los proyectos alojados en él a través de su página web, publicar proyectos públicos para que cualquiera pueda colaborar y aportar a un mismo proyecto. También mantiene un tracking de las capacidades y áreas en las que se enfoca un usuario, permitiendo así conocer los lenguajes de programación y las áreas de desarrollo en las que se desempeña la persona.

GitHub dispone a sus usuarios de una interfaz gráfica bastante intuitiva, permitiendo a novatos en el mundo de los controladores de versiones manipular su proyecto sin tener que recurrir a veces a las complejas líneas de comando de Git.

Finalmente, aportándole más a las razones de la popularidad de GitHub, es presentarse como una herramienta completamente gratis, siendo la principal opción para cualquier proyecto open source.

Maven

¿Qué es Maven?

Es una herramienta de gestión y construcción de proyectos de software, escrita en Java y C#, la cual está basada en POM (Project Object Model) el cual es un archivo XML que contiene toda la información respecto al proyecto y detalles de configuración.

¿En qué consiste Maven?
Maven permite configurar proyectos bajo una estandarización en el orden de los archivos y mantener un control de la ubicación, función y dependencias de los mismos, facilitando la articulación de nuevas funcionalidades, la compilación del proyecto e incluso, la implementación de buenas prácticas.

Maven consiste en 3 componentes: el POM, directorios y repositorios.

- **POM:** Es un archivo XML albergado en cada proyecto de Maven, el cual contiene toda la información necesaria de un proyecto, como configuración de plugins que se van a usar en el proceso de construcción.

- **Directorios:** Formato estandarizado para describir un proyecto de Maven en el POM, sería la estructura del proyecto.

- **Repositorios:** Maven utiliza un “repositorio central”, el cual nos permite descubrir y publicar proyectos como dependencias. A la hora de trabajar con Maven, este nos permitirá descargar dependencias en el repositorio local e instalarlas en nuestro proyecto.

Maven nos presenta una serie de ventajas. Por ejemplo, es muy común que al trabajar en proyectos se haga el uso de librerías y se podría dar el caso que si queremos utilizar una librería en particular, esta necesite de otras librerías para su correcto funcionamiento, por lo que sería necesario el conocimiento de qué es todo lo que debemos utilizar para la correcta utilización de esta librería. Maven nos facilita este proceso, al encargarse de importar la librería en nuestro proyecto y todo lo que haría falta para poder trabajar con esta (versión, dependencias, etc).

¿Cómo instalar Maven?

Lo primero a la hora de trabajar con una nueva herramienta, evidentemente son los pasos a seguir para poder instalarla y hacerla funcionar correctamente.

Ya que Maven es una herramienta de Java, es necesario tener Java instalado para continuar. Como recomendación, instalar la versión más reciente de Java SE.

Para instalar Maven, lo primero será descargar los archivos, ya sean .zip o .tar.gz de la última versión de Maven. Tras haber descargado el archivo, se debe descomprimir en alguna ubicación deseada y finalmente se debe agregar al PATH de las variables del entorno del sistema, como mostrado a continuación.

```
C:\Program Files\Java\apache-maven-3.6.3\bin
```

Con esto, ya se podrá acceder a los comandos de Maven a través del comando de nvm y para probar si se instaló correctamente el Maven, se ha de usar el comando `mvn -v` el cual nos mostrará la versión del Maven y otros datos relevantes como enseñados a continuación:

```
C:\Users\NatoGourmet>mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\Program Files\Java\apache-maven-3.6.3\bin\..
Java version: 14, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-14
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Con esto último verificado, se podrá empezar a usar el Maven.

¿Cómo trabajar con Maven?

Una vez ya tenemos Maven instalado y configurado, lo siguiente sería crear un proyecto de Maven, ejecutando el siguiente comando:

```
mvn -B archetype:generate -DgroupId=com.mycompany.app
-DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-quickstart
-DarchetypeVersion=1.4
```

Al correr el comando previo, notaremos que se creó un proyecto llamado *my-app* y esta carpeta contendrá un archivo llamado *pom.xml*, que lucirá de la siguiente forma:

```
<build>
  <pluginManagement><!-- lock down plugins versions to avoid using Mave
    <plugins>
      <!-- clean lifecycle, see https://maven.apache.org/ref/current/ma
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
      <!-- default lifecycle, jar packaging: see https://maven.apache.o
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
      <!-- site lifecycle, see https://maven.apache.org/ref/current/mav
      <plugin>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.7.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.0.0</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

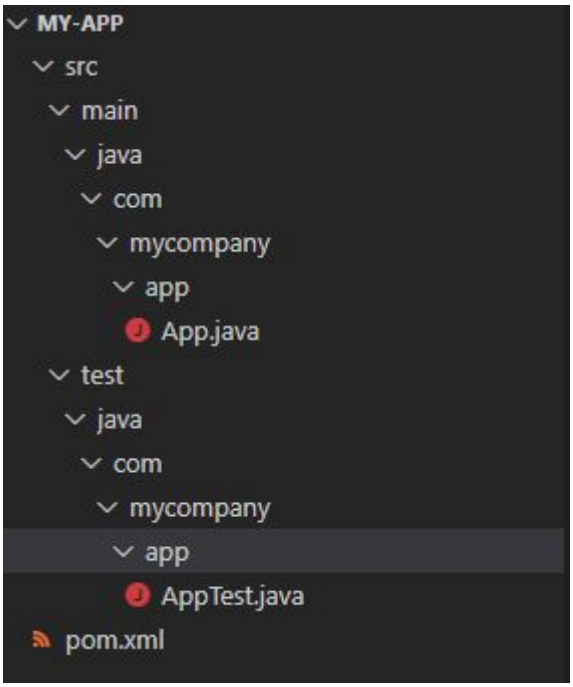
Y podremos ver que este archivo contiene la siguiente información sobre una serie de plugins:

- **artifactId:** Elemento que indica el nombre único del artefacto que ha sido generado para el proyecto.

- **version:** Elemento que indica la versión del artefacto que ha sido generado para el proyecto.

Nota: Un artefacto es un archivo, generalmente un JAR, que se implementa en un repositorio de Maven.

También notaremos, que el proyecto tiene una estructura como se ve a continuación:



Ejemplo de uso:

Por ejemplo, si quisieras integrar la librería de Google Maps en nuestro proyecto, buscaríamos la referencia de la API que buscamos integrar, que sería la siguiente:

```
<!--
https://mvnrepository.com/artifact/com.google.maps/google-maps
-services -->
<dependency>
    <groupId>com.google.maps</groupId>
    <artifactId>google-maps-services</artifactId>
    <version>0.15.0</version>
</dependency>
```

Y tras agregar esta dependencia dentro de las etiquetas de dependencias, bastaría con ejecutar el comando `mvn compile` dentro de la carpeta raíz del proyecto, lo cual descargará e instalará todas las dependencias relevantes para el proyecto, incluyendo las nuevas, como el presente caso de Google Maps.
Cabe destacar que una de las ventajas de Maven, es que al instalar las dependencias de un proyecto, no guardará los archivos relevantes en la ubicación del proyecto, en cambio, los guardará de manera local en el repositorio de Maven, de tal forma que al montar el proyecto en un repositorio remoto, no se cargarán todas estas dependencias, sino simplemente el POM de Maven. Para interés general, la ubicación de este repositorio, suele ser:

```
C:\Users\[UserName]\.m2\repository
```

Gradle

¿Que es Gradle?

Gradle, es una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento, además tiene un sistema de gestión de dependencias muy estable. Es usada como el sistema de compilación oficial de Android.

¿En qué consiste?

Gradle al ser un sistema de compilación, toma los archivos necesarios y usa las herramientas apropiadas en estos automáticamente, generando así el proyecto compilado y facilitando el desarrollo. Fue diseñado para construcciones multi-proyecto y da apoyo a la construcción del software indicando que dependencias o qué partes del proyecto están actualizadas.

Cuenta con paquetes para ser implementado en cualquier plataforma su gran versatilidad permite trabajar con monorepositorios o multirepositorios, modelando, sistematizando y construyendo soluciones exitosas, de forma rápida y precisa.

Gradle tiene las siguientes características:

- Utiliza Groovy o Kotlin DSL (Domain Specific Language) como lenguaje para sus scripts de compilación.
- 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.
- Permite compartir los resultados de la compilación para resolver en equipo de forma eficiente posibles problemas que aparezcan
- Se encarga de descargar y administrar las dependencias transitivas

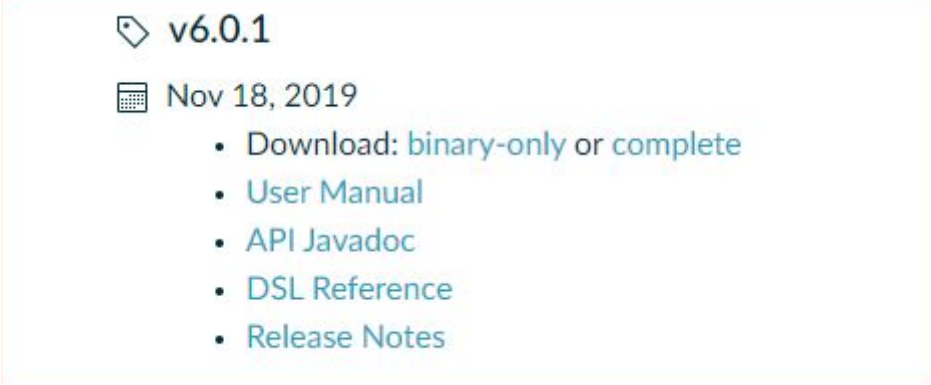
- En caso de que el código fuente o la ruta de clase cambien, Gradle cuenta con la capacidad para detectar todas las clases que se vean afectadas por dicho cambio y procederá a recompilarlas
- Permite publicar Artifacts en repositorios Ivy con diseños de directorios completamente personalizables. De igual modo, sucede con Maven en Bintray o Maven Central
- Es compatible con el formato de metadatos POM, por lo que es posible recuperar dependencias de cualquier repositorio compatible con Maven
- 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

Gradle permite facilitar la creación de compilaciones personalizadas, a diferencia de maven que es bastante rígido para esto, además mejora el rendimiento ya que trae novedades como lo son la construcción incremental, caché de compilación y gradle daemon.

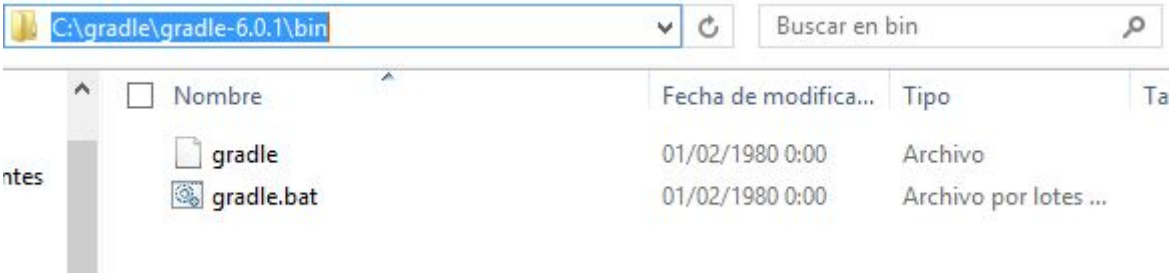
¿Cómo instalar Gradle?

Para instalar Gradle es necesario verificar que la versión del jdk de java 1.5 o superior,

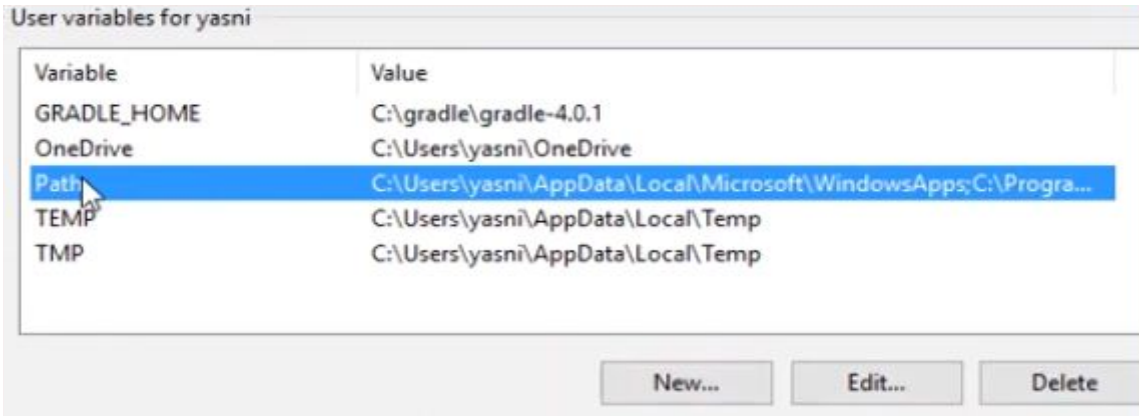
Lo primero que hacemos, es ir a la página principal de Gradle http://www.gradle.org/ y descargamos la última versión disponible.

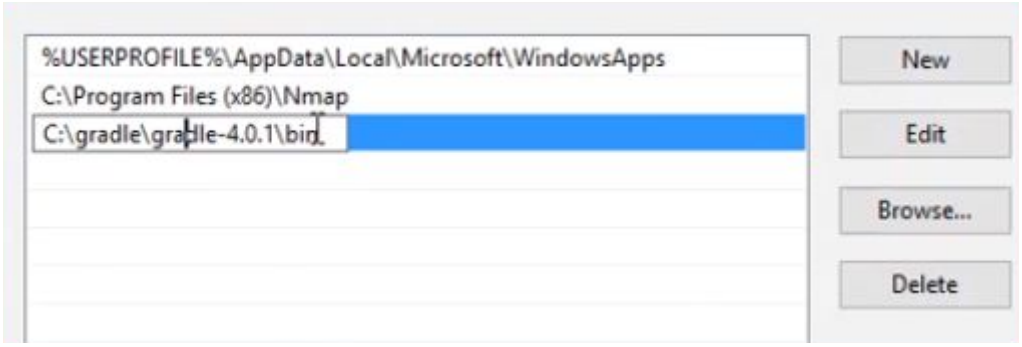


Una vez descargado, creamos la siguiente carpeta **C:\Gradle** y descomprimimos el archivo descargado



Por último configuramos la variable de entorno para poder ejecutar Gradle desde el software que utilicemos en nuestro pc





¿Cómo usar Gradle?

Creamos una carpeta donde estará ubicado nuestro proyecto y nos ubicamos en esta, aquí ejecutaremos el comando **gradle init**, esto creará un nuevo proyecto con las configuraciones puestas al ingresar este comando

```
PS C:\Users\AndersonBP\Desktop\Owl-devs\Nueva carpeta> gradle init
Starting a Gradle Daemon (subsequent builds will be faster)

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 1

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1

Project name (default: Nueva carpeta): ensayo


> Task :init
Get more help with your project: https://guides.gradle.org/creating-new-gradle-builds

BUILD SUCCESSFUL in 1m 25s
2 actionable tasks: 2 executed
```

Al estar creado el proyecto, podemos agregar las carpetas que necesitemos, por ejemplo la carpeta **src**, donde pondremos nuestros archivos de java y otros recursos.

En el archivo **build.gradle** podemos agregar tareas, plug-ins y dependencias a APIS externas, con las que se trabajarán en nuestro proyecto.

Para agregar estas, basta con buscar la sección correspondiente (excepto Task) del archivo **build.gradle** y agregar la línea apropiada para lo que necesitamos integrar, algunos ejemplos son:

- En la sección **dependencies** agregamos lo siguiente:

```
dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
    implementation 'com.google.maps.android:android-maps-utils:2.0.3'
}
```

  Con esto tendremos integrado en nuestro proyecto las librerías y clases de google maps.

- Los plug-ins se ingresan en la sección **plugins** del archivo **buil.gradle.**

```
plugins {
    id 'java'
    id "net.wooga.build-unity" version "1.0.0-rc.6"
}
```

  Con esto tendríamos a disposición herramientas para exportar proyectos de Unity3D

  Algunos ejemplos de plug-ins que podemos agregar, son los siguientes:

  - **javamuc.gradle-semantic-build-versioning:** Proporciona soporte para el versionado semántico de las compilaciones, es fácil de usar y configurar.
  - **io.freefair.maven-publish-war**: Permite crear una publicación de mavenWeb.
  - **io.freefair.maven-publish-java**: Crea una publicación mavenJava.
  - **org.mozilla.rust-android-gradle.rust-android**: Un complemento que ayuda a construir bibliotecas Rust JNI con Cargo para su uso en proyectos de Android.
  - **net.wooga.build-unity:** Este complemento proporciona tareas para exportar proyectos de plataforma desde los proyectos de Unity3D.
  - **de.db.vz.msn-plugin**: Este complemento nos permitirá ejecutar pruebas de integración de microservicio con docker.
  - **com.bmuschko.docker-remote-api:** Nos facilita la gestión de imágenes y contenedores Docker.

- **com.google.cloud.tools.jib:** Crea un contenedor para tu aplicación Java.

Conclusión

Con base en nuestra investigación, logramos observar el gran potencial que poseen estas herramientas como son Git, Gradle y Maven, a la hora de crear y desarrollar un proyecto. Son herramientas que nos ayudan a realizar un seguimiento paso a paso de una manera más fácil, y tienen como objetivo facilitar el desarrollo de una manera ordenada y eficiente.

Referencias:

[1] https://git-scm.com/book/es/v2/Herramientas-de-Git-Reiniciar-Desmitificado
[2] https://desarrolloweb.com/articulos/trabajar-ramas-git.html
[3] https://git-scm.com/book/es/v2
[4] https://ohshitgit.com/
[5]https://www.infoworld.com/article/3516426/what-is-maven-build-and-dependency-management-with-apache-maven.html
[6] https://www.genbeta.com/desarrollo/que-es-maven
[7] https://maven.apache.org/install.html
[8] https://maven.apache.org/guides/getting-started/index.html#What_is_Maven
[9] https://openwebinars.net/blog/que-es-gradle/
[10] http://chuwiki.chuidiang.org/index.php?title=Primeros_pasos_con_gradle
[11] https://guides.gradle.org/creating-new-gradle-builds/#create_a_task