**Git**

Es un sistema de control de versiones distribuido y gratuito de código abierto diseñado para manejar cualquier tipo de proyecto., es fácil de aprender ocupa poco espacio y es muy rápido.

Sus principales características son:

**Ramificación y fusión:** permite tener múltiples ramas locales las cuales pueden ser independientes entre sí, con una fácil y rápida de fusión y eliminación de sus líneas de código, puede hacer cosas como:

* Permite crear una rama y realizar cambios, volver al lugar de origen y aplicar el parce, volver al lugar donde estas experimentando y combinarlo.
* Tener varias ramas y destinarlas para usos diferentes como producción, pruebas y trabajo diario.
* Crear nuevas ramas para cada característica y poder alternar entre ellas, luego fusionar y borrar las ramas no necesarias.
* Puedes crear rama para experimentar o probar cambios, si no te sirven simplemente borras.

**Rápido y pequeño:** Fue creado para funcionar en el kernel de Linux, ósea que maneja de forma eficaz grandes repositorios, esta escrito en C lo que reduce la carga en tiempo de ejecución, la velocidad y rendimiento han sido un objetivo en su diseño.

**Distribuido:** Una de sus mejores características es que esta distribuido, en lugar de hacer chekout de la rama actual del código fuente realiza clon de todo el repositorio.

* A pesar de ser centralizado, cada usuario tiene una copia de seguridad completa del servidor principal, cada una de estas se podría usar para reemplazar dicho servidor en caso de falla. No hay un solo punto de falla de git a menos que solo haya una copia del repositorio.
* Debido a ser distribuido y su excelente sistema de ramificación. Se puede implementar una cantidad casi infinita de flujos de trabajo.
* El flujo centralizado de trabajo es muy común, Git no le permite subir si alguien ya ha subido algo desde la ultimas que lo actualizo.
* Flujo de trabajo en común involucra un administrador de integración, el cual envía el repositorio a varias personas, luego de realizar cambios el administrador es el encargado de introducir esos cambios.

**Aseguramiento de datos**: Garantiza la integridad criptográfica de cada parte del proyecto.

**Área de Ensayo**: Es un área intermedia donde las confirmaciones se pueden revisar antes de la confirmarlas.

**Código abierto**: se publica bajo la GNU 2.0 que es licencia de código abierto para garantizar su libertad de compartir y cambiar software libre.

Los comandos mas usados en Git 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**

Agrega 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 automaticamente 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 estémos 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 un servicio de alojamiento de repositorios de Git el cual es el corazón de GitHub, pero agrega muchas características propias como interfaz grafica basada en la web, control de acceso y varias funciones de colaboración. Si Git es el corazón, Hub es como su alma, es lo que convierte una línea de comandos Git en la red social más grande para desarrolladores.

Aloja los repositorios de código y brinda herramientas muy útiles para el trabajo en equipo, puedes contribuir a la mejora del software de otros, para esto provee de algunas funcionalidades para hacer un fork y solicitar un pulls.

Su principal funcionalidad es la bifurcación, copia un repositorio de la cuenta de un usuario a otro mediante un **fork**, es la clonación de un repositorio ajeno para trabajar en él, corregir, mejorar o implementar algo, después de realizar modificaciones puedes enviar un **pull** al dueño del proyecto, este podrá analizar los cambios y los podrá adjuntar al repo original.

Además del servicio de alojamiento, ofrece varias herramientas útiles para el trabajo en equipo como:

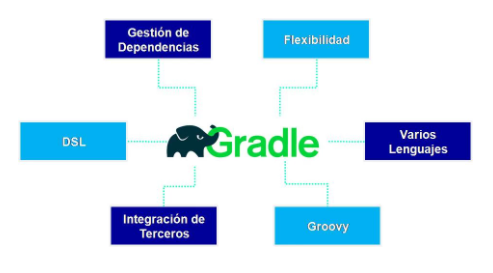
* **Wiki:** para el mantenimiento de las distintas versiones de las páginas.
* **Sistema de seguimientos de problemas**: permite a todos los miembros del equipo detallar problemas de tu software o sugerencias que quieran hacer.
* **Herramienta de revisión de código**: puede añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit.
* **Visor de ramas**: se pueden comparar los progresos realizados en las distintas ramas de nuestro repo.

**Gradle**

Es una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Nos permite usar otros lenguajes diferentes de java, cuenta con un sistema de gestión de dependencias muy estable, es personalizable y rápido, es el sistema de compilación oficial para Android y cuenta con soporte para diversas tecnologías.

Permite construir desde microservicios hasta aplicaciones móviles, puede ser utilizado por pequeñas y grandes empresas ayudando a sus equipos de trabajo a desarrollar, automatizar y entregar software de calidad en un menos tiempo.

Este sistema para construcción de proyectos toma lo mejor de lo que ya existe: puede integrar tareas de Ant, usar el manejo de dependencias de Ivy, ciclos de compilación y pruebas tipo Maven y, lo mejor de todo, sigue el paradigma de convención sobre configuración; es decir, todas las opciones configurables tienen valores por defecto con lo más común o útil, de modo que sólo es necesario modificarlos para casos especiales, pero para la mayoría de los casos se puede usar el valor por omisión, lo cual nos permite tener scripts bastante breves.



Una de las características que hacen a Gradle tan sencillo de usar, es que los scripts usan un lenguaje específico de dominio (Domain-Specific Language, DSL) que extiende el lenguaje de programación Groovy. Esto le da a Gradle a la vez sencillez y poder, ya que en los scripts se pueden utilizar elementos tanto de programación orientada a objetos como de programación funcional. Gradle cuenta también con una arquitectura de plugins y de entrada ofrece varios muy útiles: para compilar proyectos Java, proyectos Groovy (y por supuesto híbridos Java+Groovy), crear artefactos para publicar en repositorios Maven, generar documentación técnica (Javadoc/Groovydoc), realizar pruebas unitarias y generar reportes con los resultados, etc. Y por supuesto, cuenta con una API para que terceros puedan crear sus propios plugins.

Algunas características de Gradle que podemos destacar son las siguientes:

* **Depuración colaborativa:** Permite compartir los resultados de la compilación para resolver en equipo de forma eficiente posibles problemas que aparezcan.
* **Construcción incremental:** Valida 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 Artefactos.
* **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 recompilarlas.
* **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 Artefactos 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 prágramática 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 artefactos almacenados en caché para evitar el tráfico de red innecesario.

**Maven**

Se utiliza en la gestión y construcción de software, posee la capacidad de realizar ciertas tareas claramente definidas como la compilación de código y su empaquetado, hace posible la creación de software con dependencias incluidas dentro de la estructura del jar. Es necesario definir todas las dependencias del proyecto en un fichero propio del proyecto, su característica mas importante es la capacidad de trabajar en red, cuando se definen sus dependencias, este se encarga de ubicar las librerías que deseamos utilizar en Maven central, el cual es un repositorio que contiene cientos de librerías que se actualizan constantemente.

Es una herramienta de gestión de proyectos que fomenta, a través de la definición de un Project Object Model (el fichero POM que se define en cada uno de los proyectos o módulos), un conjunto de estándares que definen el ciclo de vida del proyecto. Una lógica de ejecución de plugins que permite, en determinadas las fases del ciclo de vida, poder ejecutar un plugin concreto que ofrezca una solución a un problema. Por ejemplo, antes de la ejecución de los test, en muchos casos interesa levantar un servidor de aplicaciones, ejecutar los tests y una vez hayan finalizado, bajar ese servidor para que no ocupe ningún puerto y para mejorar el rendimiento de la máquina.

Con **Maven** se simplifica enormemente la realización de tareas como borrar los “.class”, compilar, generar la documentación de javadoc, el “.jar”, generar documentación web con montones de informes (métricas, código duplicado, etc.), e incluso más. Con comandos simples, crea una estructura de directorios para el proyecto con sitio para los fuentes, los iconos, ficheros de configuración y datos, etc. Por otro lado, si le indicamos qué jars externos necesitamos, es capaz de ir a buscarlos a internet y descargarlos por nosotros. Además, se encarga de pasar automáticamente nuestros test de prueba cuando compilamos, e incluso nos genera un zip de distribución en el que van todos los jar necesarios y ficheros de configuración de nuestro proyecto.

Maven está diseñado y compuesto de pequeñas partecitas o módulos llamados **plug-ins**, tal que la arquitectura no es rígida sino extensible. De hecho, existe un API java pública, ya que Maven está hecho en java, con la cual nosotros podemos hacer nuestros propios plugins de Maven (llamados **MOJOs**). De hecho, Maven se bajará los plugins automáticamente desde internet de la misma forma en que baja las dependencias por nosotros, lo cual nos va a evitar muchísimo trabajo manual.

**Repositorios**

es básicamente un lugar donde están los **artefactos Maven**, estructurados en cierta forma estándar que permitirá al propio Maven hacer el download específico de las dependencias. Existen tres tipos de repositorios:

**Repositorio Local**: cada desarrollador, al instalarse y usar Maven (el comando) va a tener localmente, en su máquina, un repositorio. La herramienta Maven, al resolver dependencias, buscándolas y bajándolas de internet, las va a ir cacheando (es decir, se las va a guardar para recordarlas y no tener que salir a buscarlas todo el tiempo). El repositorio por defecto se encuentra en la carpeta “$HOME/.m2/repository”, pero esto se puede modificar en “$MAVEN/conf/settings.xml”.

**Repositorio Público de Internet**: es el repositorio de Maven por defecto, donde la herramienta busca los artefactos. Es un servidor donde se encuentran los proyectos open source más utilizados en la comunidad java. Uno podría publicar sus artefactos allí, aunque para eso tiene que cumplir con ciertos requerimientos de licencia (software libre y open source), y además seguir un proceso para que los artefactos sean aprobados.

**Repositorio “Empresarial”:** la empresa u organización tiene un repositorio Maven propio, ya sea para publicar sus propios artefactos solo dentro de la organización (sin hacerlo público a toda internet), o bien para evitar que cada Maven de cada desarrollador acceda a internet para bajarse los artefactos. Básicamente, se instala una aplicación en un servidor, y luego se configura Maven en cada máquina de los desarrolladores para que utilicen este repositorio.

Maven no deja de ser la base de los compiladores actuales, de IDES como Eclipse, NetBeans o IntelliJ, a los que ofrece soporte gracias a algunas de sus características:

* Un sistema de gestión dependencias.
* Un mecanismo distribuido de distribución de librerías. El comportamiento distribuido es siempre desde el repositorio local de Maven hacia los repositorios que están publicados en Internet o en la red corporativa.
* Mecanismos para ser extensible, por la creación de plugins customizables.
* Es multi-plataforma, puede funcionar tanto en entornos Linux como Windows al ser una aplicación Java.
* Es software libre, con lo cual es el código está disponible, se podría modificar y customizar en caso de que fuera necesario.
* Fomenta la reutilización de código y de librerías. El hecho de que Apache Maven ofrezca repositorios oficiales y públicos de software libre, con librerías desplegadas, que toda la comunidad de desarrolladores de software utiliza, hace que este concepto también pueda trasladarse al mundo empresarial, a través de repositorios remotos corporativos, compartidos por distintos equipos de proyectos o el propio equipo de desarrollo.
* Es compatible con múltiples IDEs.

<https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/>

<https://openwebinars.net/blog/que-es-gradle/>

<https://www.genbeta.com/desarrollo/que-es-maven>

<https://conociendogithub.readthedocs.io/en/latest/data/introduccion/>

<http://panamahitek.com/que-es-maven-y-para-que-se-utiliza/>

<http://www.7sabores.com/blog/git-comandos-basicos>

<https://sg.com.mx/revista/33/construccion-proyectos-gradle>

<https://git-scm.com/about>

<http://webipedia.es/tecnologia/cursos/maven-construyendo-proyectos/>