Sistemas de control de versiones

Índice

1. Que es un sistema de control de versiones

Un sistema de control de versiones (SCV) es aquel sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueden recuperar versiones específicas más adelante.

Aunque es habitual el uso de SCV en el ámbito del desarrollo de aplicaciones informáticas, con los ficheros de código fuente, estos sistemas pueden emplearse con casi cualquier archivo que se pueda almacenar en una computadora. En general, en cualquier proyecto colaborativo que requiera trabajar con equipos de personas de forma concurrente, es útil el uso de SCV. El control de versiones de código está integrado en el proceso de desarrollo de software de muchas empresas, sobre todo cuando hay más de un programador trabajando en el mismo proyecto.

Un SCV permite regresar a versiones anteriores de archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un SCV también significa generalmente que, si se estropean o pierden archivos, será posible recuperarlos fácilmente. Además, estos beneficios serán a un coste muy bajo.

En resumen, los principales beneficios de utilizar una herramienta SCV son:

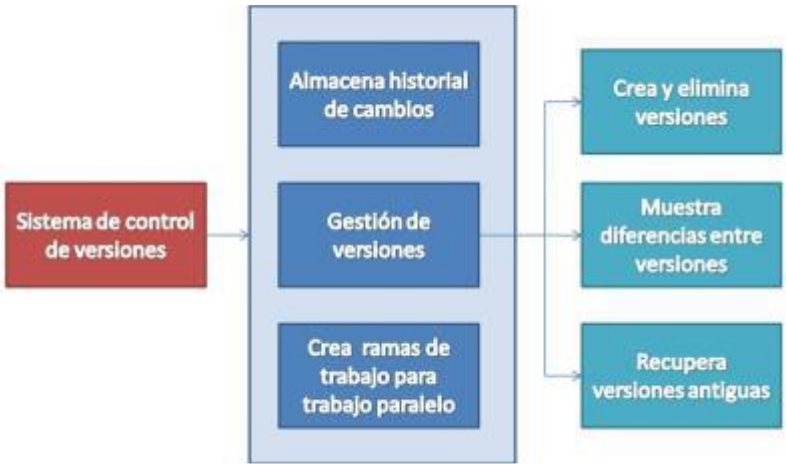
- Cualquier revisión almacenada de un archivo puede ser recuperada para visualizarse o modificarse.
- Pueden desplegarse las diferencias entre distintas versiones.
- Las correcciones pueden ser creadas automáticamente.
- Múltiples desarrolladores pueden trabajar simultáneamente en el mismo proyecto o archivo sin pérdida de datos. Se permite el control de los usuarios que trabajarán en paralelo en el proyecto.
- Los proyectos pueden ser divididos para permitir el desarrollo simultáneo en varias versiones. Estas divisiones pueden ser fusionadas para alcanzar el objetivo principal del desarrollo.
- El desarrollo distribuido, es soportado a través de las redes de datos con diferentes mecanismos de autentificación.

1.1 Características

Los programas de control de versiones realizan funciones durante la vida de un proyecto, entre las que destacan:

- Permitir el control de los usuarios que trabajarán en paralelo en el proyecto:
  - Establecer los usuarios que tendrán acceso.
  - Asignarles el tipo de acceso.

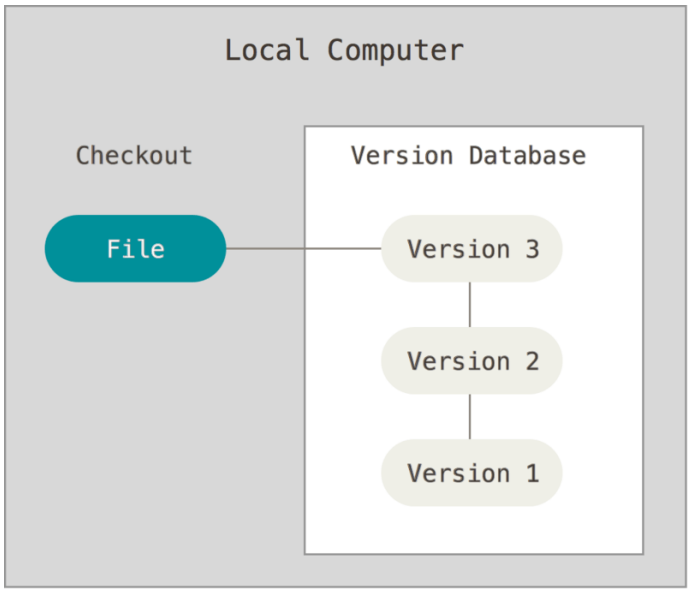
- Con relación a los ficheros:
  - Permitir el almacenamiento de los ficheros.
  - Permitir realizar cambios sobre los ficheros almacenados: modificar parcialmente un archivo, borrarlo, cambiarle el nombre, moverlo, etc.
  - Disponer de un histórico detallado (cambios, fecha, motivo, usuario...) de las acciones realizadas en el tiempo.
- Con relación a las versiones:
  - Etiquetar los archivos en un punto determinado del desarrollo del proyecto para señalar una versión estable y así poder identificarla posteriormente mediante esa etiqueta.
  - Disponer de un histórico detallado de las versiones (etiqueta, usuario responsable, fecha, etc.).
  - Permitir la recuperación de todos o alguno de los archivos de una versión.
  - Comparar versiones teniendo una vista o informe de los cambios entre ellas.
- Con relación al proyecto, permitir la creación de ramas, es decir, bifurcar el proyecto en dos o más líneas que pueden evolucionar paralelamente por separado. Las ramas pueden utilizarse para hacer pruebas de forma independiente sin perturbar la línea principal del desarrollo. Si lo realizado en la la rama de nuevo desarrollo funciona correctamente, puede ser fusionada con la rama principal.



2. Clasificación de los SCV

2.1 Sistemas de control de versiones locales

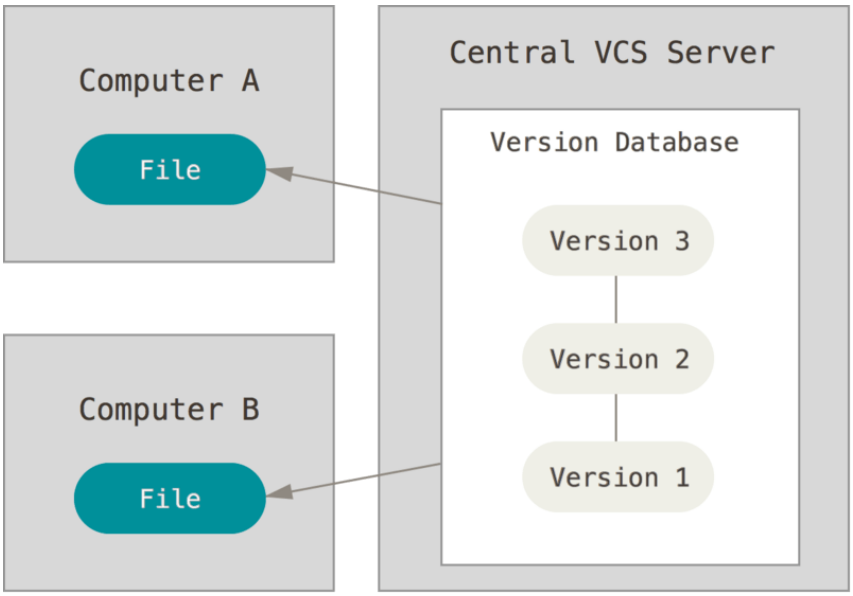
Los SCV locales contenían una simple base de datos en la que se llevaba el registro de todos los cambios realizados a los archivos.



Una de las herramientas de control de versiones locales más popular fue un sistema llamado **RCS**. Esta herramienta funcionaba guardando conjuntos de parches (es decir, las diferencias entre archivos) en un formato especial en disco, y era capaz de recrear cómo era un archivo en cualquier momento a partir de dichos parches. Prácticamente está en desuso.

2.2 Sistemas de control de versiones centralizados

El siguiente gran problema con el que se encuentran las personas es que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar este problema. Estos sistemas, como **CVS**, **Subversion** o **Perforce**, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones durante años.



En un SCV centralizado todas las diferentes versiones de un proyecto están almacenadas en un único repositorio de un servidor central. Para que los desarrolladores puedan acceder a esas

versiones o códigos fuente deben solicitar al SCV una copia local, en la cual realizan todos los cambios necesarios, y al finalizar recurren nuevamente al SCV para que almacene las modificaciones realizadas como una nueva versión. Es entonces cuando esa versión generada estará a disposición de los demás desarrolladores, al igual que las versiones anteriores.

Esta configuración ofrece muchas ventajas, especialmente frente a SCV locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un SCV centralizado que tener que lidiar con bases de datos locales en cada cliente.

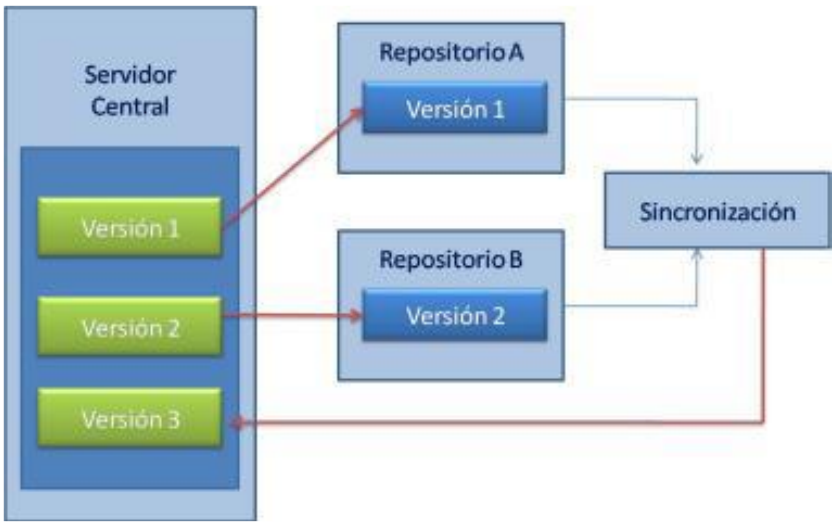
Sin embargo, esta configuración también tiene desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales. Los SCV locales sufren de este mismo problema. Cuando se tiene toda la historia del proyecto en un mismo lugar, se podría perder todo.

2.3 Sistemas de control de versiones distribuidos

Los SCV Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que se mencionan en los puntos anteriores. Algunos ejemplos de estos sistemas son **Git**, **Mercurial**, **Bazaar** o **Darcs.**

En el SCV distribuido cada desarrollador realiza una **copia del repositorio de proyectos completo** a su computadora, generándose un repositorio local del proyecto. Este repositorio local incluye todos los archivos del proyecto y el historial de cambios generado en previas actualizaciones. Cuando cada desarrollador realice modificaciones a los archivos de su repositorio local, el contenido de este repositorio se irá distanciando de los repositorios locales de otros desarrolladores, lo que provocará que se generen ramas de un mismo proyecto en los repositorios de cada desarrollador. De esta manera los desarrolladores pueden trabajar paralela e independientemente, guardando sus propias versiones en un repositorio local.

En la etapa del proceso de desarrollo del proyecto que requiera unir los repositorios locales de los desarrolladores, el SCV realizará una sincronización de copias, con lo cual se generará una nueva versión del proyecto conteniendo todos los cambios realizados por cada desarrollador. Este tipo de SCV mantiene un repositorio como referencia para realizar la sincronización de repositorios locales. En el SCV distribuido no es necesario mantener una conexión de red permanente a un servidor que contenga el repositorio de referencia, las conexiones de red por parte de los desarrolladores solo se requieren cuando se realiza el proceso de sincronización. Además, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.



Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que se puede colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

3. Conceptos relacionados con el control de versiones

Hay una serie de conceptos generales que se manejan cuando se está trabajando con sistemas de control de versiones.

- **Repositorio**: Lugar en donde se almacenan las revisiones. Físicamente puede ser un archivo, una colección de archivos, una base de datos, etc. Los repositorios pueden ser locales o remotos, en función de donde estén ubicados. Un usuario o desarrollador puede tener un repositorio local y un repositorio remoto. Por cada proyecto hay un repositorio que almacena los archivos del proyecto.
- **Revisión**: Es una visión estática en el tiempo del estado de un grupo de archivos y directorios. Posee una etiqueta que la identifica. Suele tener asociado metadatos ("datos sobre datos" o "informaciones sobre datos") como pueden ser:
  - Identidad de quién hizo las modificaciones.
  - Fecha y hora en la cual se almacenaron los cambios.
  - Razón o motivo por el cual se realizaron los cambios.
  - De qué revisión y/o rama se deriva la revisión.
  - Palabras o términos clave asociados a la revisión.
- **Rama de trabajo** (o desarrollo) **Branch**: En el más sencillo de los casos, una rama es un conjunto ordenado de revisiones. La revisión más reciente se denomina principal

(main) o cabeza (head). Las ramas se pueden separar y juntar según como sea necesario, formando un grafo de revisión.

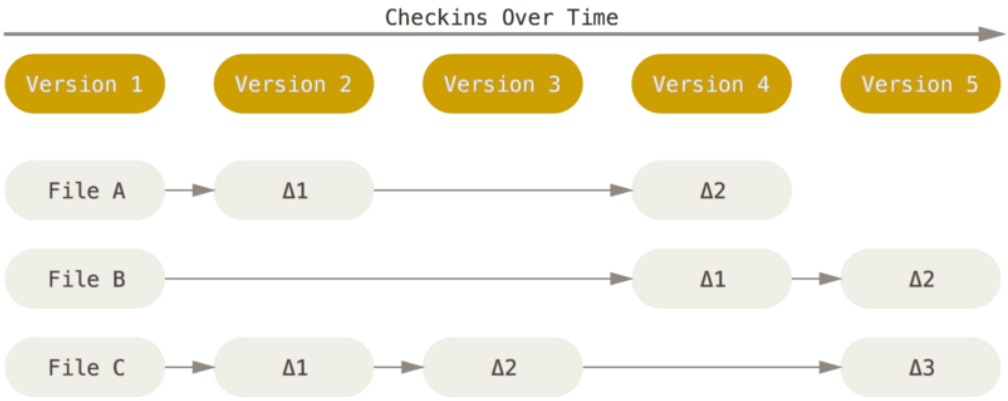
- **Área de trabajo**: es el conjunto de directorios y archivos controlados por el sistema de control de versiones, y que se encuentran en edición activa. Está asociado a una rama de trabajo concreta.
- **Conflicto**: Ocurre cuando varias personas han hecho cambios contradictorios en un mismo documento (o grupo de documentos); **los SCV solamente alertan de la existencia del conflicto**. El proceso de solucionar un conflicto se denomina **resolución**.
- **Cambio (diff)**: Modificaciones realizadas en un archivo que está bajo control de revisiones. Cuando se unen los cambios en un archivo (o varios), generando una revisión unificada, se dice que se ha hecho una fusión o integración.
- **Integración, fusión o merge**: Une dos conjuntos de cambios sobre un fichero o conjunto de ficheros en una revisión unificada de dicho fichero o conjunto de ficheros. Permite aplicar todos los cambios realizados en una rama a otra cualquiera del repositorio. La operación patch realiza lo mismo que merge pero se limita a modificaciones en ficheros, mientras que merge permite directorios y ficheros.
- **Sincronizar**: Operación que permite actualizar el directorio de trabajo con los cambios realizados en el repositorio desde la última actualización.
- **Commit**: Operación que permite actualizar el contenido de un repositorio con los cambios del área de trabajo.
- **Push**: Operación que permite actualizar el contenido de un repositorio remoto (si lo hay) con el contenido del repositorio local.
- **Pull**: Operación que permite actualizar el contenido de un repositorio local con el contenido del repositorio remoto (si lo hay).

4. SCV Git

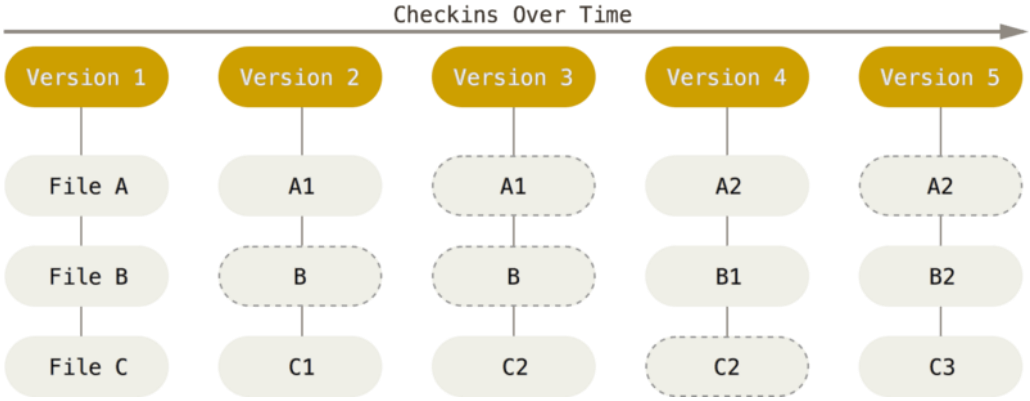
Git es un SCV creado por Linus Torvalds en el año 2005. Algunas de las características que se perseguían con Git eran:

- Velocidad.
- Diseño sencillo.
- Gran soporte para desarrollo no lineal (miles de ramas paralelas).
- Completamente **distribuido**.
- Capaz de manejar grandes proyectos (como el kernel de Linux) eficientemente (velocidad y tamaño de los datos).

La principal diferencia entre Git y cualquier otro SCV es cómo Git maneja sus datos. Conceptualmente, la mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.) modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo.



Git no maneja ni almacena sus datos de este modo. Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que se confirma un cambio, o se guarda el estado de un proyecto en Git, él básicamente hace una "foto" del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado. Git maneja sus datos como una secuencia de copias instantáneas



En Git, **la mayoría de las operaciones son locales**, es decir, sólo necesitan archivos y recursos locales para operar; por ejemplo, para navegar por la historia del proyecto, no se necesita salir al servidor para obtener la historia y mostrarla, simplemente se lee directamente de la base de datos local.

Esto significa que se ve la historia del proyecto casi al instante. Si es necesario ver los cambios introducidos entre la versión actual de un archivo y ese archivo hace un mes, Git puede buscar el archivo hace un mes y hacer un cálculo de diferencias localmente, en lugar de tener que pedirle a un servidor remoto que lo haga, u obtener una versión antigua del archivo del servidor remoto y hacerlo de manera local.

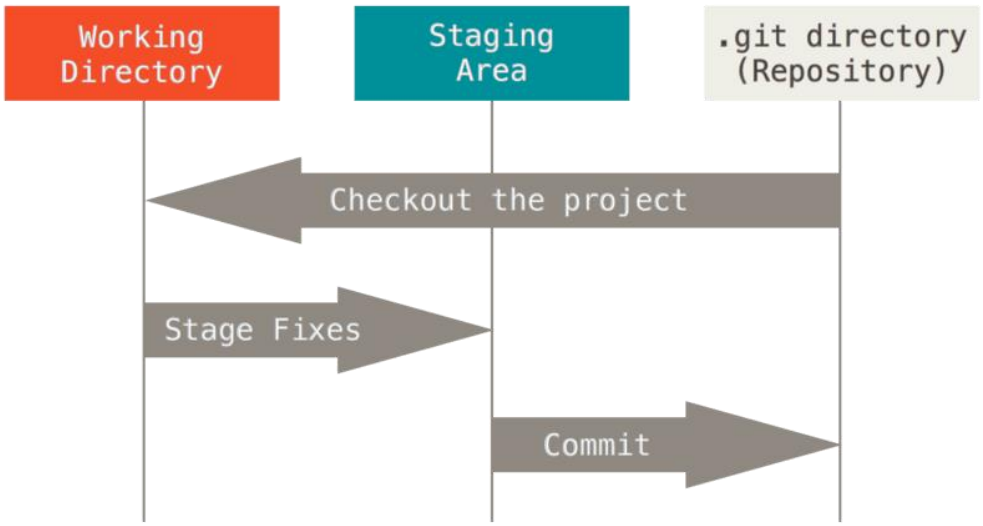
Git tiene integridad. Todo en Git es verificado mediante una suma de comprobación (checksum en inglés) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa. Esta funcionalidad está integrada en Git al más bajo nivel y es parte integral de su filosofía.

4.1 Los tres estados de Git

Git tiene tres estados principales en los que se pueden encontrar nuestros archivos: confirmado (committed), modificado (modified), y preparado (staged).

Confirmado significa que los datos están almacenados de manera segura en la base de datos local (Repositorio Local). **Modificado** significa que se ha modificado el archivo, pero todavía no se ha confirmado a tu base de datos. **Preparado** significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Hay, por tanto, tres secciones principales de un proyecto de Git: El directorio de Git (Git directory o repositorio), el directorio de trabajo (working directory), y el área de preparación (staging area).



El **directorio de Git** es donde se almacenan los metadatos y la base de datos de objetos para un proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otra computadora.

El **directorio de trabajo** es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que puedan ser usados o modificados.

El **área de preparación** es un archivo, generalmente contenido en el directorio de Git, que almacena información acerca de lo que va a ir en una próxima confirmación. En algunos casos se le denomina índice ("index"), pero lo estándar es referirse a ella como el área de preparación.

El flujo de trabajo básico en Git contempla las siguientes acciones:

- Se realizan modificaciones en una serie de archivos en el directorio de trabajo.
- Se preparan los archivos, añadiéndolos al área de preparación.

- Se confirman los cambios, lo que implica que se toman los archivos tal y como están en el área de preparación y se almacena esa copia instantánea de manera permanente en el directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (**committed**). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (**staged**). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (**modified**).

4.2 GitHub

GitHub es un servicio de alojamiento de repositorios Git.

Git es un sistema descentralizado. Por lo tanto, no necesita de un servidor donde subir los cambios ya que los desarrolladores pueden enviarse los cambios los unos a los otros desde sus equipos.

Pero lo más habitual es que un equipo de trabajo funcione mejor disponiendo de un servidor central, ya que así se evita tener que descargar los cambios de cada ordenador perteneciente a un miembro del equipo de manera individual.

Una de las ventajas de GitHub, y que fue propósito con el que nació, es alojar el repositorio de código en el que trabaja todo un equipo. Al alojar el código, se puede decidir si éste será público o su visibilidad estará limitada a unos usuarios determinados, que son los únicos que tendrán permiso para acceder al mismo y subir cambios.