Introduccióna GIT

GIT

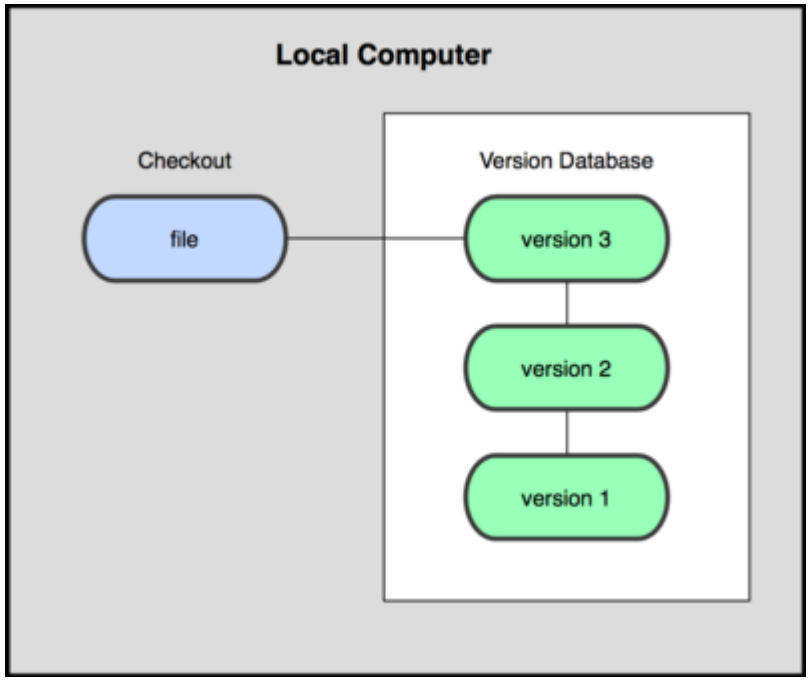
- **Control de Versiones**
  - ¿Qué es el control de versiones?
  - El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedas recuperar versiones específicas más adelante.
  - Un sistema de control de versiones (Version Control System o VCS en inglés) permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más.

GIT

- **Sistemas de control de versiones locales**
  - El método de control de versiones usado por mucha gente es copiar los archivos a otro directorio.
  - Esta técnica es muy común porque es muy simple, pero también tremendamente propensa a cometer errores.
  - Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.
  - Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos

GIT

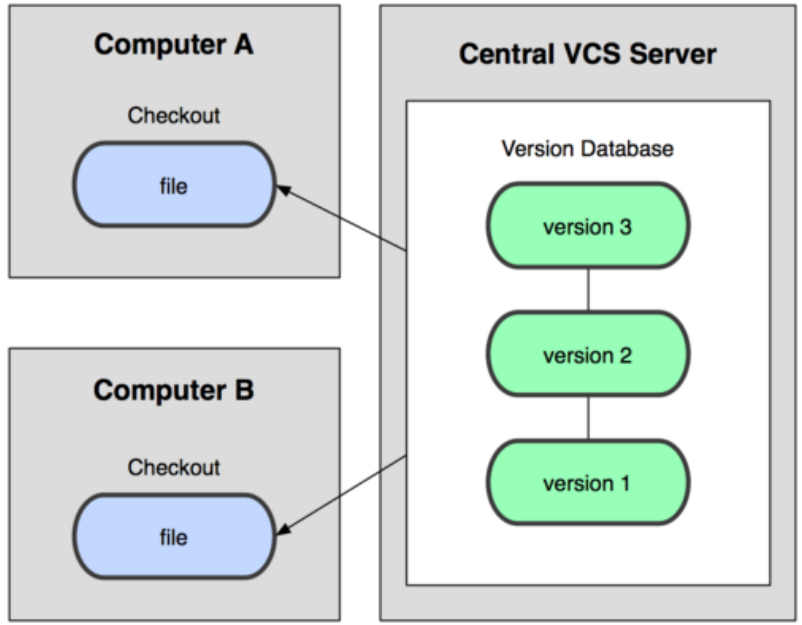
- **Sistemas de control de versiones locales**
  - Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales.
  - Hasta el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo.
  - Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco.
  - Así aw puede recrear después cómo era un archivo en cualquier momento sumando los distintos parches.



GIT

Sistemas de control de versiones centralizados

- El siguiente gran problema que encontramos es que necesitamos colaborar con otros desarrolladores.
- Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés).
- Estos sistemas, como CVS, Subversion, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos de ese lugar central.
- Durante muchos años, éste ha sido el estándar para el control de versiones



GIT

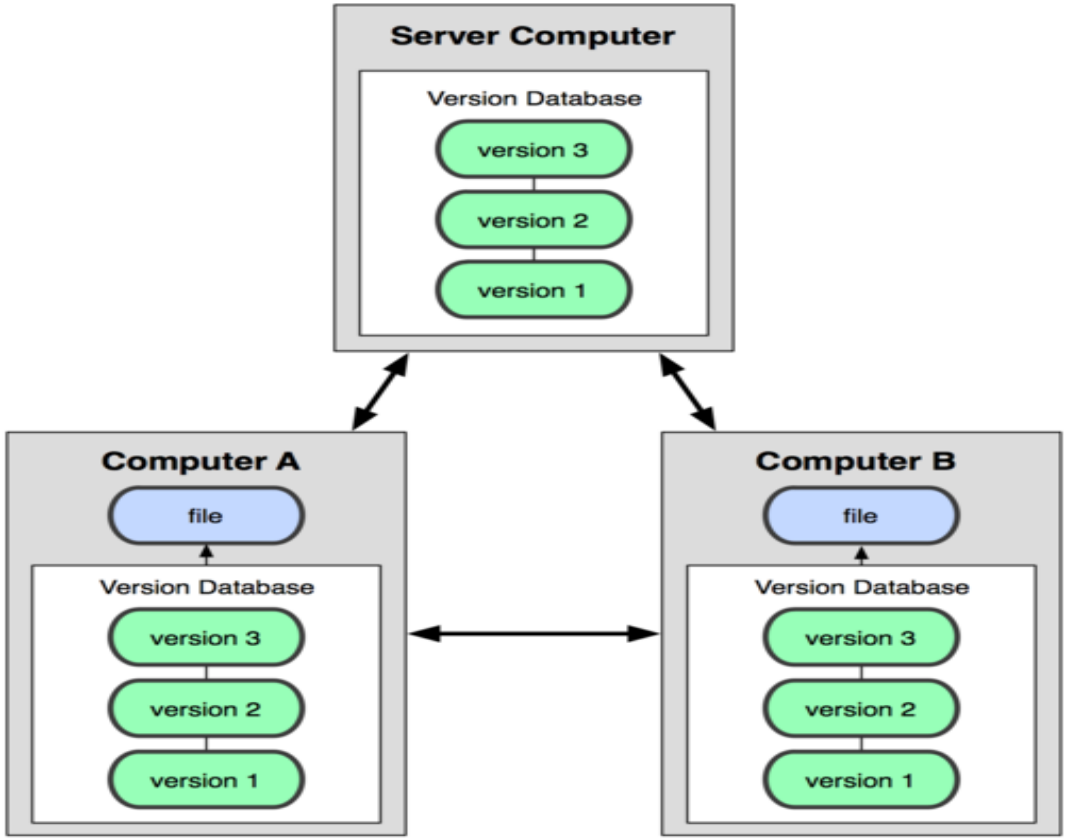
- Esta configuración ofrece muchas ventajas, especialmente frente a VCSs locales.
- Por ejemplo, todo el mundo sabe hasta cierto punto en qué está trabajando el resto de gente en el proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.
- Sin embargo, esta configuración también tiene serias 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 puede colaborar o guardar cambios versionados de aquello en que están trabajando.
- Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, pierdes absolutamente todo — toda la historia del proyecto salvo aquellas instantáneas que la gente pueda tener en sus máquinas locales.
- Los VCSs locales sufren de este mismo problema — cuando tienes toda la historia del proyecto en un único lugar, te arriesgas a perderlo todo.

GIT

- **Sistemas de control de versiones distribuidos**
- Es aquí donde entran los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés).
- En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio.
- Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo.
- Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos

GIT

- **Sistemas de control de versiones distribuidos**



GIT

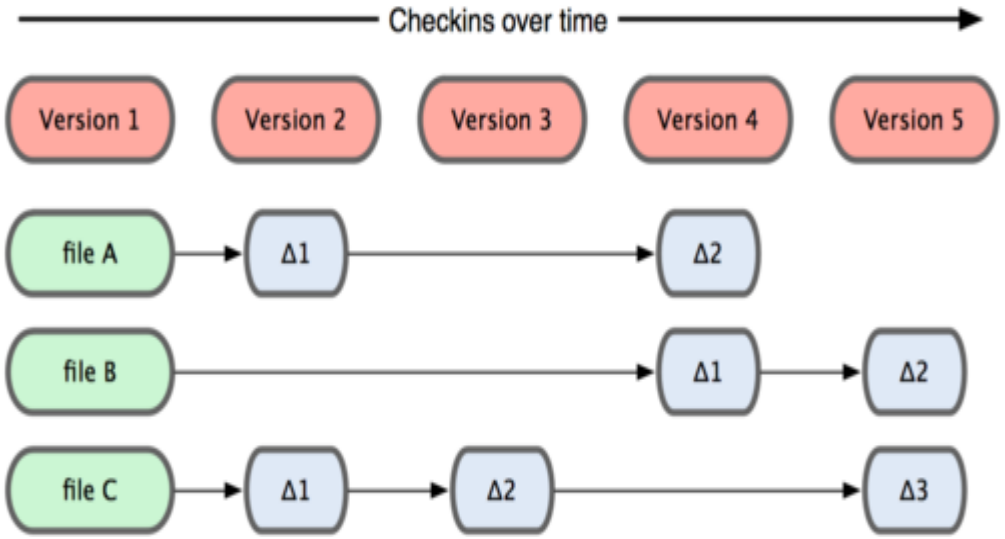
- **Historia de Git**
  - Como muchas de las grandes cosas en esta vida, Git comenzó con un poco de destrucción creativa y bastante polémica.
  - El núcleo de Linux es un proyecto de software de código abierto con un alcance bastante grande.
  - Durante la mayor parte del mantenimiento del núcleo de Linux (1991-2002), los cambios en el software se pasaron en forma de parches y archivos.
  - En 2002, el proyecto del núcleo de Linux empezó a usar un DVCS propietario llamado BitKeeper.
  - En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y la herramienta dejó de ser gratuita.
  - Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron durante el uso de BitKeeper.

GIT

- **Historia de Git**
- Algunos de los objetivos del nuevo sistema fueron los siguientes:
  - Velocidad
  - Diseño sencillo
  - Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
  - Completamente distribuido
  - Capaz de manejar grandes proyectos como el núcleo de Linux de manera eficiente (velocidad y tamaño de los datos)
- Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún así conservar estas cualidades iniciales.

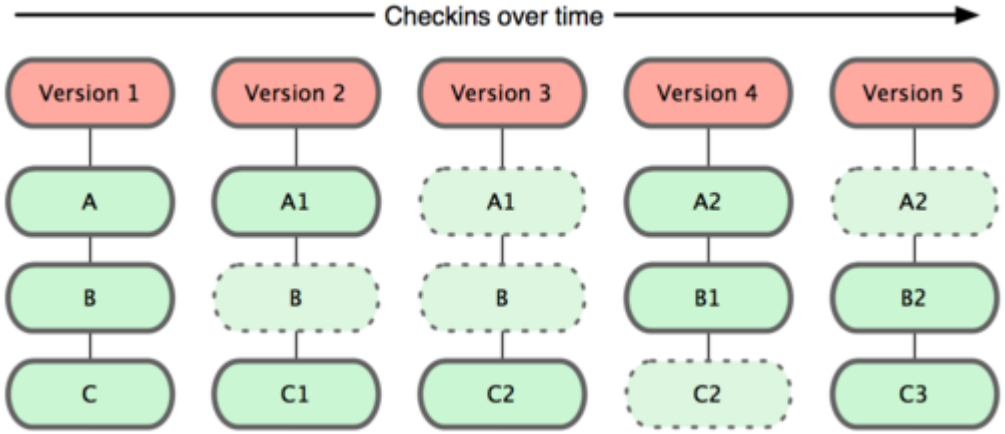
GIT

- **Fundamentos de Git**
- Instantáneas, no diferencias
- La principal diferencia entre Git y cualquier otro VCS (Subversion y compañía incluidos) es cómo Git modela 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
- Otros sistemas tienden a almacenar los datos como cambios de cada archivo respecto a una versión base



GIT

- **Fundamentos de Git**
- Git no modela ni almacena los datos de este modo. En cambio, Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos.
- Cada vez que se confirma un cambio, o se guarda el estado del proyecto en Git, se hace una foto del aspecto de todos los 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

- **Fundamentos de Git**
- Casi cualquier operación es local
  - La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar — por lo general no se necesita información de ningún otro ordenador de la red —.
  - Como esta toda la historia del proyecto en el disco local, la mayoría de las operaciones parecen prácticamente inmediatas.
  - Para navegar por la historia del proyecto, Git no necesita acceder al servidor para obtener la historia y mostrarla — simplemente la lee directamente de la base de datos local.
  - Esto significa que se puede acceder a la historia del proyecto casi al instante.
  - Si se quieren 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

- **Fundamentos de Git**
  - Esto también significa que hay muy poco que no se pueda hacer sin estar desconectado o sin VPN.
  - Si no estamos conectados, se pueden hacer cambios sin problemas hasta que se consiga una conexión de red para subirlos.
  - En muchos otros sistemas, esto es imposible o muy costoso.
  - En Perforce, por ejemplo, no se puede hacer mucho cuando no se está conectado al servidor; y en Subversion y CVS, se pueden editar archivos, pero no se pueden subir los cambios a la base de datos.

GIT

- **Git tiene integridad**
  - Todo en Git es verificado mediante una suma de comprobación antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma (checksum en inglés).
  - 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.
  - No se puede perder información durante su transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo.

GIT

- **Git tiene integridad**
  - El mecanismo que usa Git para generar esta suma de comprobación se conoce como hash SHA-1.
  - Se trata de una cadena de 40 caracteres hexadecimales (0-9 y a-f), y se calcula en base a los contenidos del archivo o estructura de directorios en Git.
  - Un hash SHA-1 tiene esta pinta:

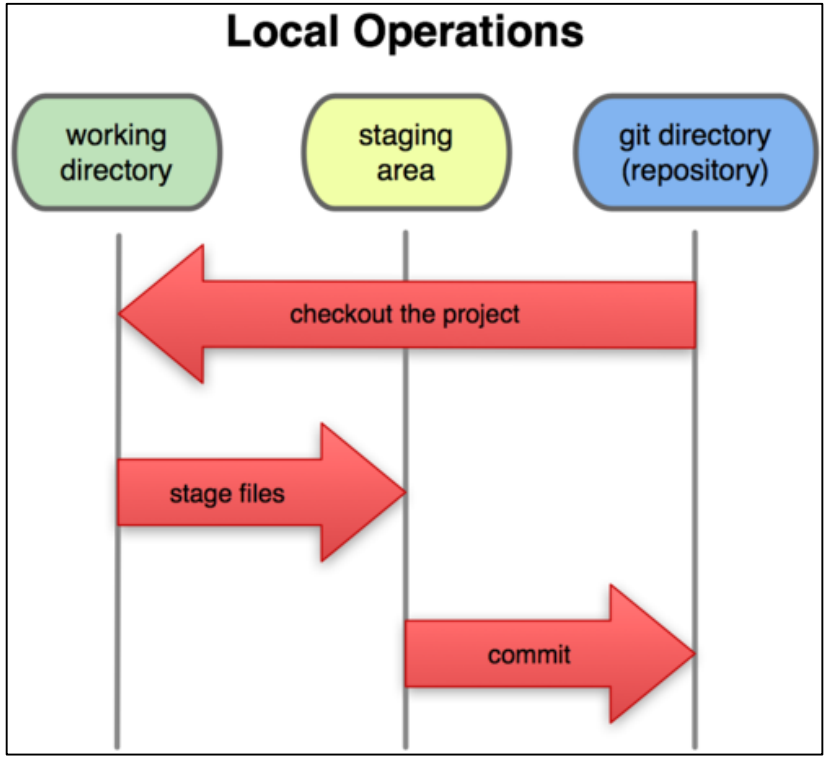
    24b9da6552252987aa493b52f8696cd6d3b00373
  - Verás estos valores hash por todos lados en Git, ya que los usa con mucha frecuencia.
  - De hecho, Git guarda todo no por nombre de archivo, sino en la base de datos de Git por el valor hash de sus contenidos.

GIT

- **Git generalmente sólo añade información**
  - Cuando se realizan acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git.
  - Es muy difícil conseguir que el sistema haga algo que no se pueda deshacer, o que de algún modo borre información.
  - Como en cualquier VCS, se puede perder o estropear cambios que no se han confirmado todavía; pero después de confirmar una instantánea en Git, es muy difícil de perder, especialmente si se envía (push) a la base de datos a otro repositorio con regularidad.

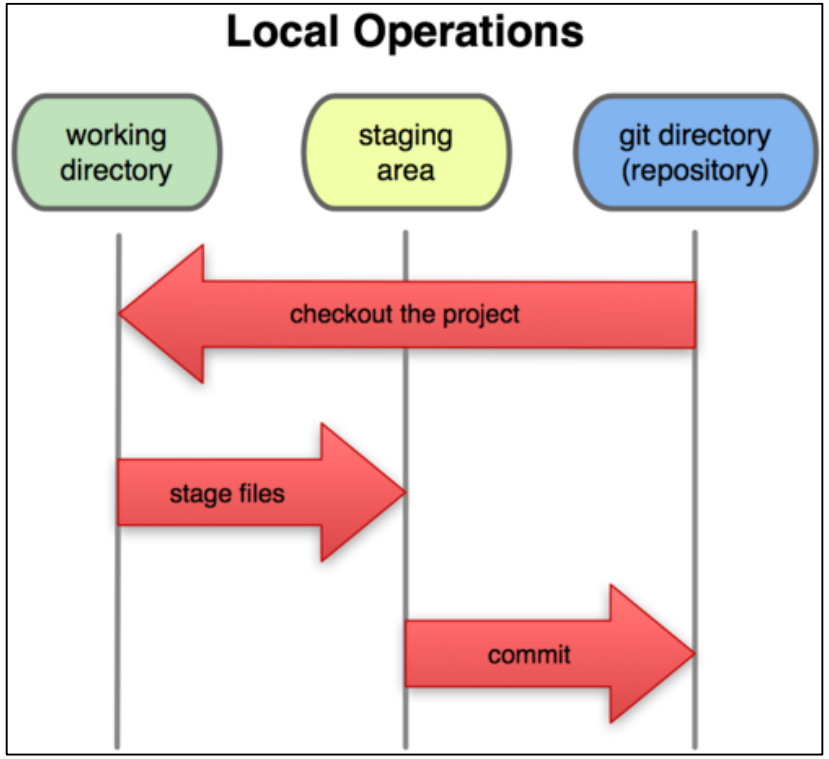
GIT

- **Los tres estados**
  - Git tiene tres estados principales en los que se pueden encontrar tus archivos:
    - confirmado (committed): los datos están almacenados de manera segura en la base de datos local
    - modificado (modified): se ha modificado el archivo pero todavía no se ha confirmado en la base de datos.
    - preparado (staged). Preparado significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.
  - Por tanto, hay tres secciones principales de un proyecto de Git:
    - El directorio de Git (Git directory)
    - El directorio de trabajo (working directory)
    - El área de preparación (staging area).



GIT

- **Los tres estados**
  - El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para el proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otro ordenador.
  - 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 se puedan usar o modificar.
  - El área de preparación es un sencillo archivo, generalmente contenido en el directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se denomina el índice, pero se está convirtiendo en estándar el referirse a ello como el área de preparación.



GIT

- **Los tres estados**
  - 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).