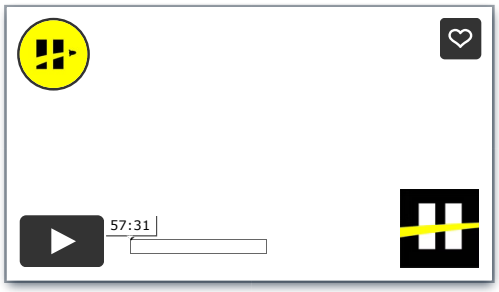
Contenido de la clase

Tiempo de lectura
14 min

Uso de Git



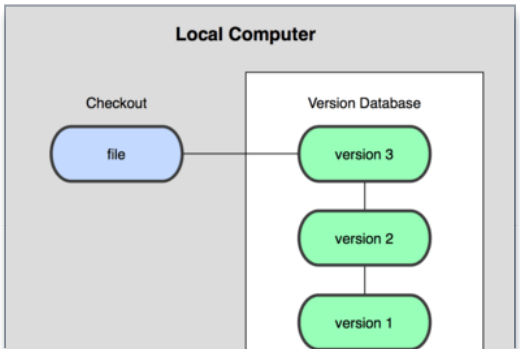
Version Control System

¿Qué es un control de versiones, y por qué debería importarte? Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Si eres diseñador gráfico o web, y quieres mantener cada versión de una imagen o diseño (algo que sin duda quieres), un sistema de control de versiones (Version Control System o VCS en inglés) es una elección muy sabia. Te 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. Usar un VCS también significa generalmente que si rompes o pierdes archivos, puedes recuperarlos fácilmente.

Hay varios tipos de sistemas de versionado, estos pueden ser:

Locales

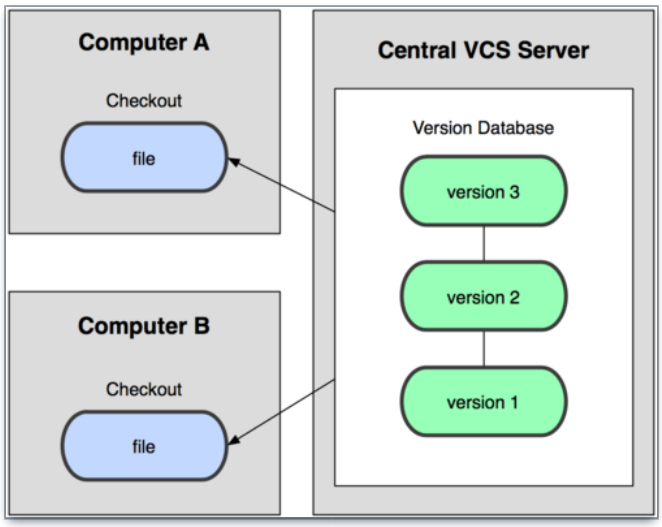


Dejanos tu feedback!👍

Contenido de la clase

Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son ingeniosos). Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.
Como se pueden imaginar, este sistema funciona *bien* para trabajar solos, pero si queremos incorporar otra gente al equipo van a empezar a surgir problemas.

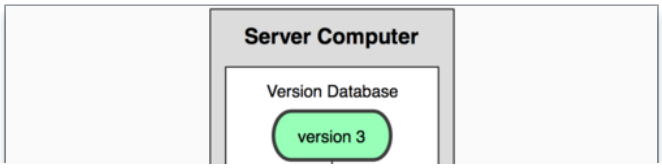
Centralizados



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 descargan los archivos desde ese lugar central. Durante muchos años éste ha sido el estándar para el control de versiones.

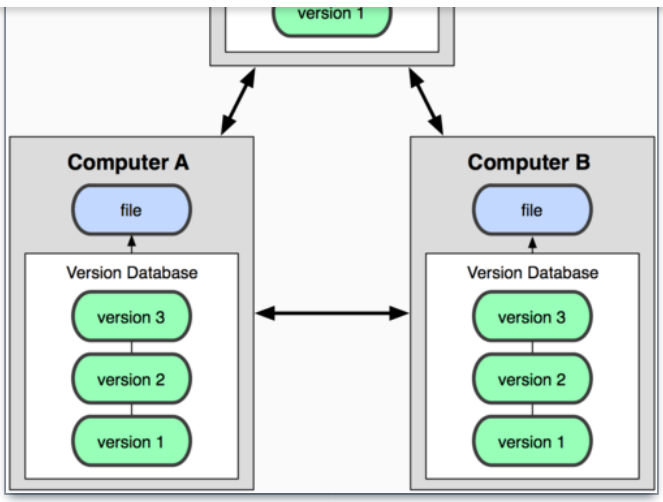
Este sistema ofrece varias ventajas, como por ejemplo: Todo el mundo puede saber en qué están trabajando los demás colaboradores y los administradores tienen control sobre qué archivos pueden ver/modificar cada colaborador. Pero tambien presenta un *problema importante*: que hay un **punto único de falllo**. ¿Si éste server se cae? Nadie puede seguir trabajando ni trackeando sus cambios. ¿O si se rompe y no hay backups? Se pierde absolutamente *todo* el trabajo realizado.

Distribuido



Dejanos tu feedback!👍

Contenido de la clase



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.

Historia de Git

Como muchas de las grandes cosas en esta vida, **Git** comenzó con un poco de destrucción creativa y encendida 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 ofrecida gratuitamente. 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. Algunos de los objetivos del nuevo sistema:

- **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)**

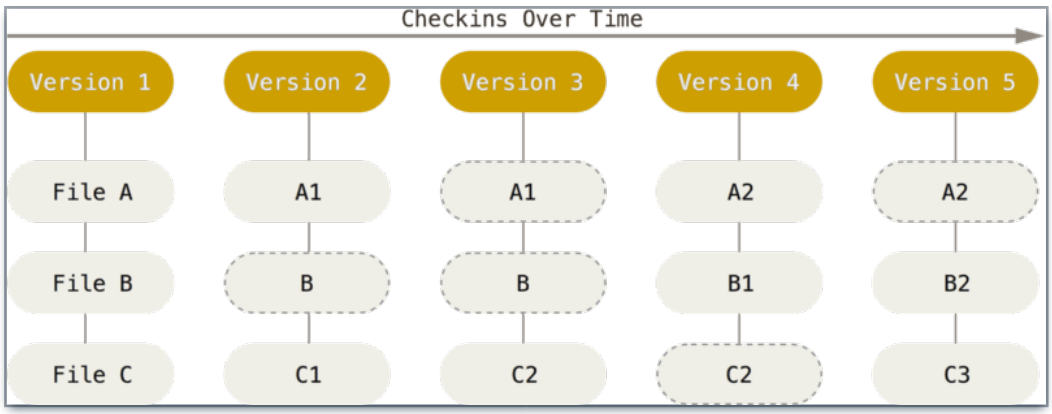
Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente a gran escala, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal.

Dejanos tu feedback!👍

Contenido de la clase

Conceptos de Git

Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu 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.



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 tu red. Como tienes toda la historia del proyecto ahí mismo, en tu disco local, la mayoría de las operaciones parecen prácticamente inmediatas (con otros sistemas el proceso involucra llamados por red que generan retardos importantes).

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.**

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. 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 por el valor hash de sus contenidos.

Vamos a distinguir dos directorios, primero el *directorio de git*: que es donde almacena los metadatos y la base de datos de tu proyecto, y segundo el *directorio de trabajo* que es una copia

Dejanos tu feedback!👍

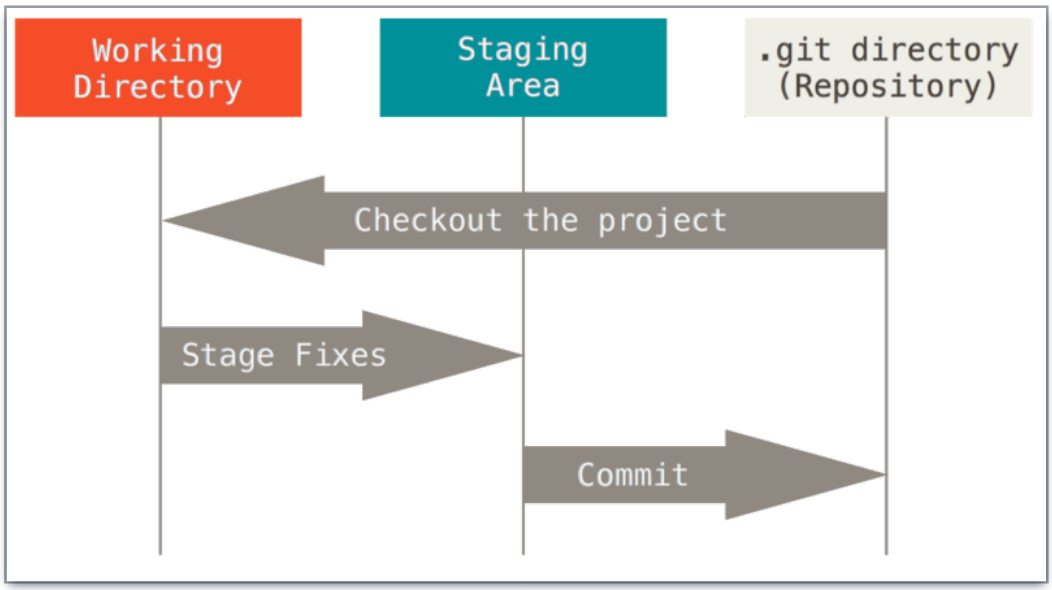
Contenido de la clase

dentro del *directorio de trabajo* pueden estar en unos de los siguientes *estados*:

Estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos:

- **committed: significa que los datos están almacenados de manera segura en tu base de datos local.**
- **modified: significa que has modificado el archivo pero todavía no lo has commiteado a tu base de datos.**
- **staged: significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima commiteada.**



Hay un archivo simple, generalmente contenido en tu directorio de Git, llamado que almacena información acerca de lo que va a ir en tu próxima confirmación, al contenido de este archivo. O al archivo mismo se lo conoce como **staging area**.

Sabiendo esto, el flujo de trabajo básico en Git sería algo así:

- **Modificas una serie de archivos en tu *directorio de trabajo*.**
- ***Stageas* los archivos, añadiendolos a tu staging area o área de preparación.**
- **Commiteas o Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu 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, y 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 (no se incluyó

Dejanos tu feedback!👍

Contenido de la clase

Github

Github.com es una red para almacenar tus repositorios, esencialmente es un repositorio de repositorios. Es uno de los tantos disponibles en internet, y el más popular. Git != Github, aunque funcionen muy bien juntos. Github es un lugar donde puedes compartir tu código o encontrar otros proyectos. También actúa como portfolio para cualquier código en el que hayas trabajado. Si planeas ser un desarrollador deberías tener cuenta en Github. Usaremos Github extensivamente durante tu tiempo en Henry.

Lectura recomendada

- Git: sitio oficial
- Github: tutorial oficial
- Git: tutorial oficial
- Terminal turorial

Si tienes dudas sobre este tema, puedes consultarlas en el canal *01b_git* de Slack

Dejanos tu feedback!👍