

Empezando - Fundamentos de Git

Fundamentos de Git

Entonces, ¿qué es Git en pocas palabras? Es muy importante asimilar esta sección, porque si entiendes lo que es Git y los fundamentos de cómo funciona, probablemente te sea mucho más fácil usar Git de manera eficaz. A medida que aprendas Git, intenta olvidar todo lo que puedas saber sobre otros VCSs, como Subversion y Perforce; hacerlo te ayudará a evitar confusiones sutiles a la hora de utilizar la herramienta. Git almacena y modela la información de forma muy diferente a esos otros sistemas, a pesar de que su interfaz sea bastante similar; comprender esas diferencias evitará que te confundas a la hora de usarlo.

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, como ilustra la Figura 1-4.

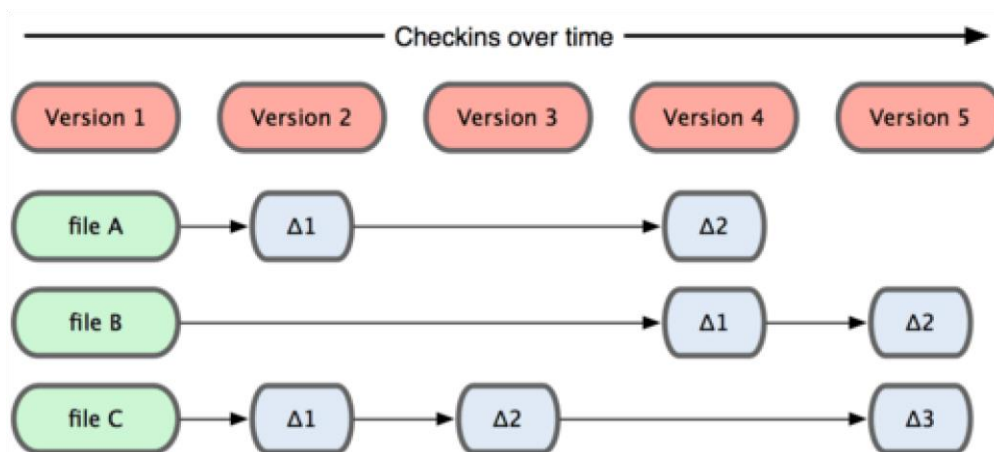


Figura 1-4. Otros sistemas tienden a almacenar los datos como cambios de cada archivo respecto a una versión base.

Git no modela ni almacena sus 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 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. Git modela sus datos más como en la Figura 1-5.

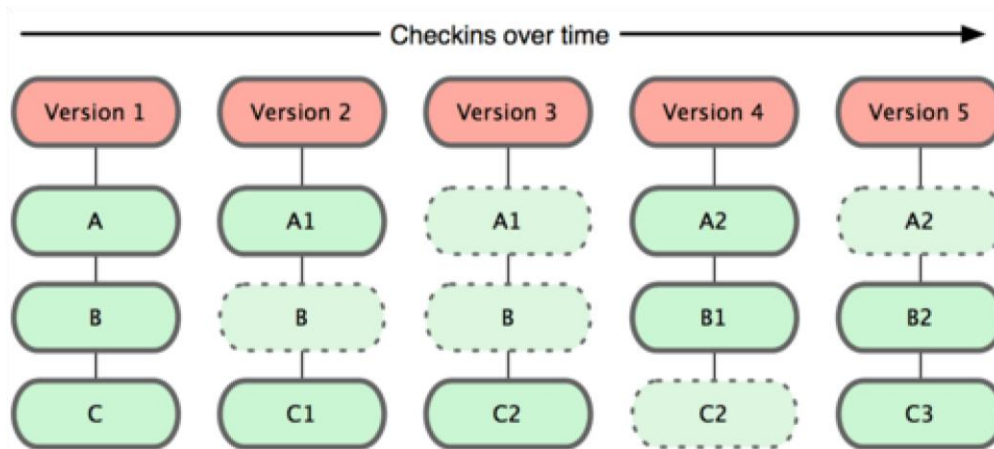


Figura 1-5. Git almacena la información como instantáneas del proyecto a lo largo del tiempo.

Esta es una distinción importante entre Git y prácticamente todos los demás VCSs. Hace que Git reconsidere casi todos los aspectos del control de versiones que muchos de los demás sistemas copiaron de la generación anterior. Esto hace que Git se parezca más a un mini sistema de archivos con algunas herramientas tremendamente potentes construidas sobre él, que a un VCS. Exploraremos algunos de los beneficios que obtienes al modelar tus datos de esta manera cuando veamos ramificaciones (branching) en Git en el Capítulo 3.

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 tu red. Si estás acostumbrado a un CVCS donde la mayoría de las operaciones tienen esa sobrecarga del retardo de la red, este aspecto de Git te va a hacer pensar que los dioses de la velocidad han bendecido Git con poderes sobrenaturales. Como tienes toda la historia del proyecto ahí mismo, en tu disco local, la mayoría de las operaciones parecen prácticamente inmediatas.

Por ejemplo, para navegar por la historia del proyecto, Git no necesita salir al servidor para obtener la historia y mostrártela, simplemente la lee directamente de tu base de datos local. Esto significa que ves la historia del proyecto casi al instante. Si quieres ver los cambios introducidos en un archivo entre la versión actual y la de 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 desde la red y hacerlo de manera local.

Esto también significa que hay muy poco que no puedas hacer si estás desconectado o sin VPN. Si te subes a un avión o a un tren y quieres trabajar un poco, puedes confirmar tus cambios felizmente hasta que consigas una conexión de red para subirlos. Si te vas a casa y no consigues que tu cliente VPN funcione correctamente, puedes seguir trabajando. En muchos otros sistemas, esto es imposible o muy doloroso. En Perforce, por ejemplo, no puedes hacer mucho cuando no estás conectado al servidor; y en Subversion y CVS, puedes editar archivos, pero no puedes confirmar los cambios a tu base de datos (porque tu base de datos no tiene conexión). Esto puede no parecer gran cosa, pero te sorprendería la diferencia que puede suponer.

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. No puedes perder información durante su transmisión o sufrir corrupción de archivos sin que Git lo detecte.

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.

Git generalmente sólo añade información

Cuando realizas 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, puedes perder o estropear cambios que no has confirmado todavía; pero después de confirmar una instantánea en Git, es muy difícil de perder, especialmente si envías (push) tu base de datos a otro repositorio con regularidad.

Esto hace que usar Git sea un placer, porque sabemos que podemos experimentar sin peligro de fastidiar gravemente las cosas. Para un análisis más exhaustivo de cómo almacena Git su información y cómo puedes recuperar datos aparentemente perdidos, ver Capítulo 9.

Los tres estados

Ahora presta atención. Esto es lo más importante a recordar acerca de Git si quieres que el resto de tu proceso de aprendizaje prosiga sin problemas. Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en tu base de datos local. Modificado significa que has modificado el archivo pero todavía no lo has 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.

Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

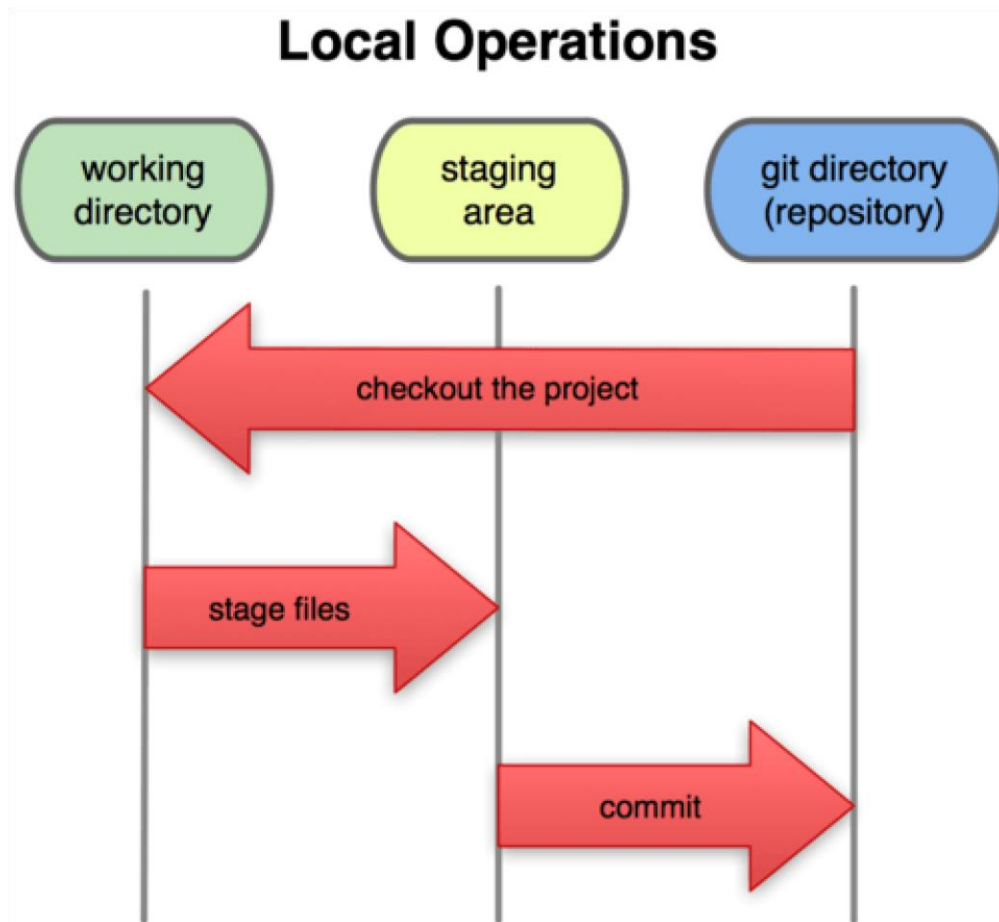


Figura 1-6. Directorio de trabajo, área de preparación y directorio de Git.

El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas 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 los puedas usar o modificar.

El área de preparación es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice, pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

El flujo de trabajo básico en Git es algo así:

1. Modificas una serie de archivos en tu directorio de trabajo.

2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. 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, 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). En el Capítulo 2 aprenderás más acerca de estos estados, y de cómo puedes aprovecharte de ellos o saltarte toda la parte de preparación. (git-scm, s.f.)

Bibliografías

git-scm. (s.f.). *Empezando - Fundamentos de Git*. Obtenido de git-fastversion-control: <https://git-scm.com/book/es/v1/EmpezandoFundamentos-de-Git>