

Inicio - Sobre el Control de Versiones

Este capítulo se va a hablar de cómo comenzar a utilizar Git. Empezaremos describiendo algunos conceptos básicos sobre las herramientas de control de versiones; después, trataremos de explicar cómo hacer que Git funcione en tu sistema; finalmente, exploraremos cómo configurarlo para empezar a trabajar con él. Al final de este capítulo deberás entender las razones por las cuales Git existe y conviene que lo uses, y deberás tener todo preparado para comenzar.

Acerca del Control de Versiones

¿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.** Aunque en los ejemplos de este libro usarás archivos de código fuente como aquellos cuya versión está siendo controlada, en realidad puedes hacer lo mismo con casi cualquier tipo de archivo que encuentres en una computadora.

Si eres diseñador gráfico o de web y quieres mantener cada versión de una imagen o diseño (es algo que sin duda vas a querer), usar un sistema de control de versiones (VCS por sus siglas en inglés) es una muy decisión muy acertada. Dicho sistema te permite regresar a versiones anteriores de tus 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 VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente. Adicionalmente, obtendrás todos estos beneficios a un costo muy bajo.

Sistemas de Control de Versiones Locales

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.

Para afrontar este problema los programadores desarrollaron hace tiempo VCS locales que contenían una simple base de datos en la que se llevaba el registro de todos los cambios realizados a los archivos.

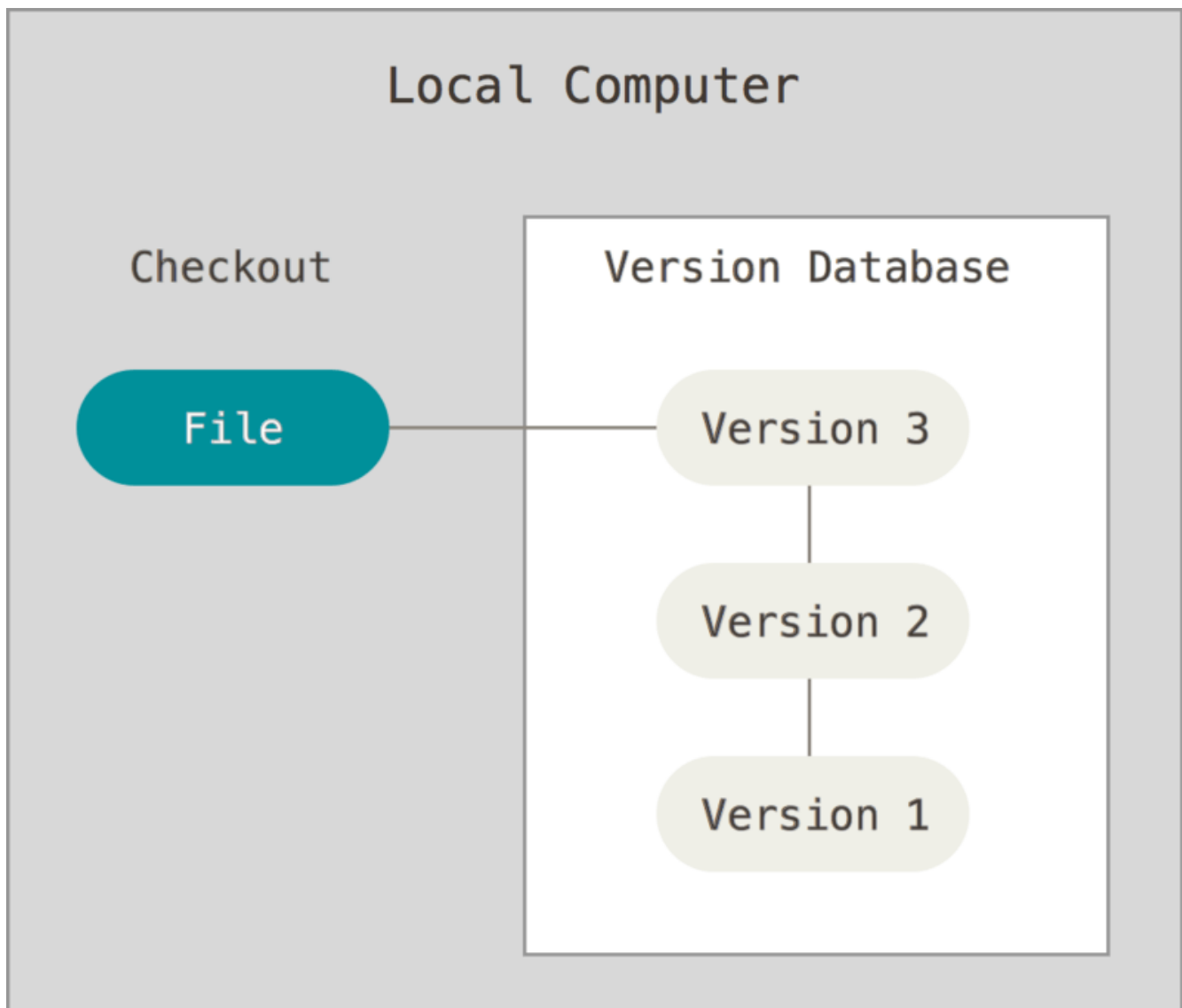


Figure 1. control de versiones local.

Una de las herramientas de control de versiones más popular fue un sistema llamado RCS, que todavía podemos encontrar en muchas de las computadoras actuales. Incluso el famoso sistema operativo Mac OS X incluye el comando `rcs` cuando instalas las herramientas de desarrollo. Esta herramienta funciona guardando conjuntos de parches (es decir, las diferencias entre archivos) en un formato especial en disco, y es capaz de recrear cómo era un archivo en cualquier momento a partir de dichos parches.

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, y 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 por muchos años.

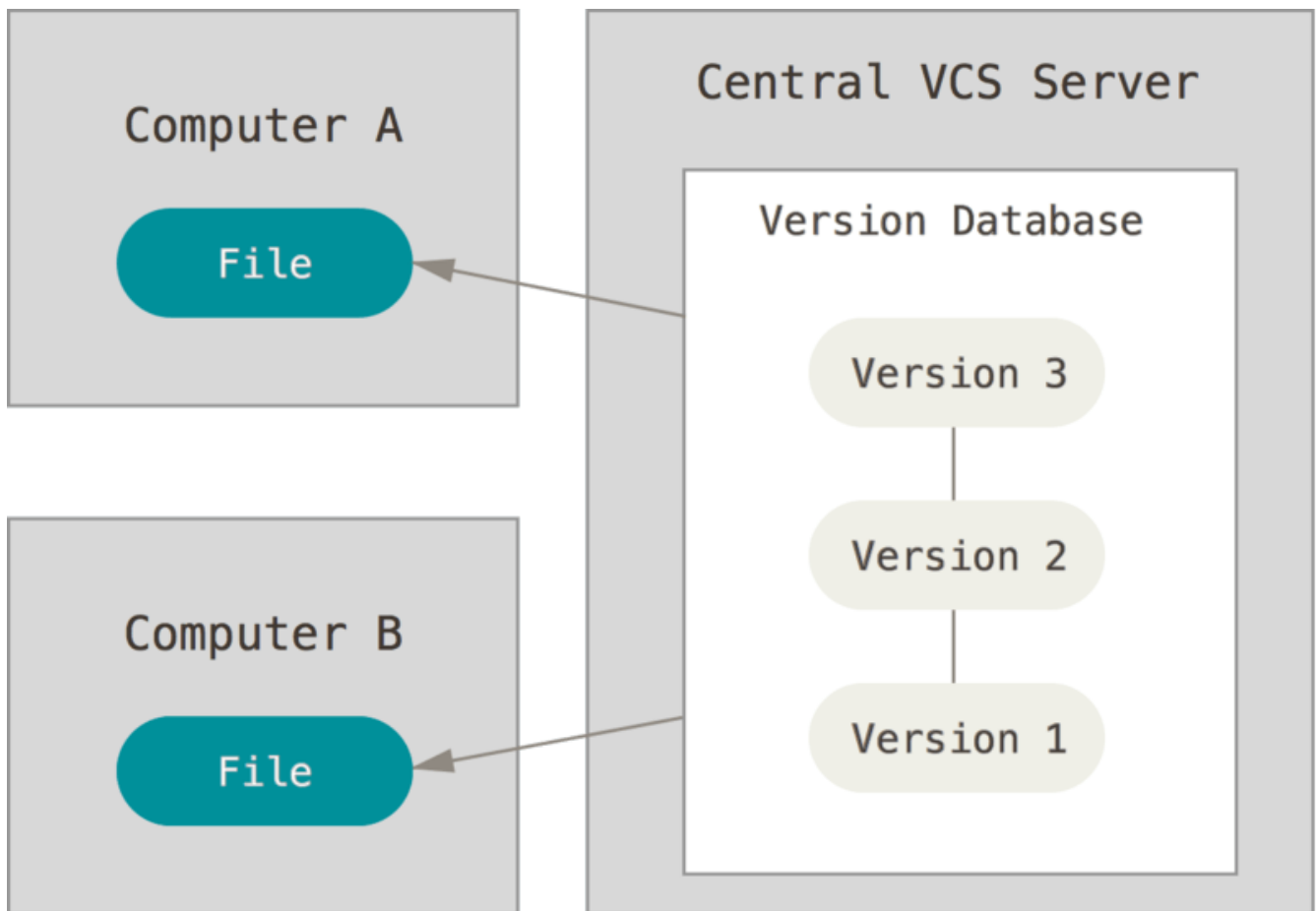


Figure 2. Control de versiones centralizado.

Esta configuración ofrece muchas **ventajas**, especialmente frente a VCS 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 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 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 VCS locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.

Sistemas de Control de Versiones Distribuidos

Los sistemas de Control de Versiones Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, 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.

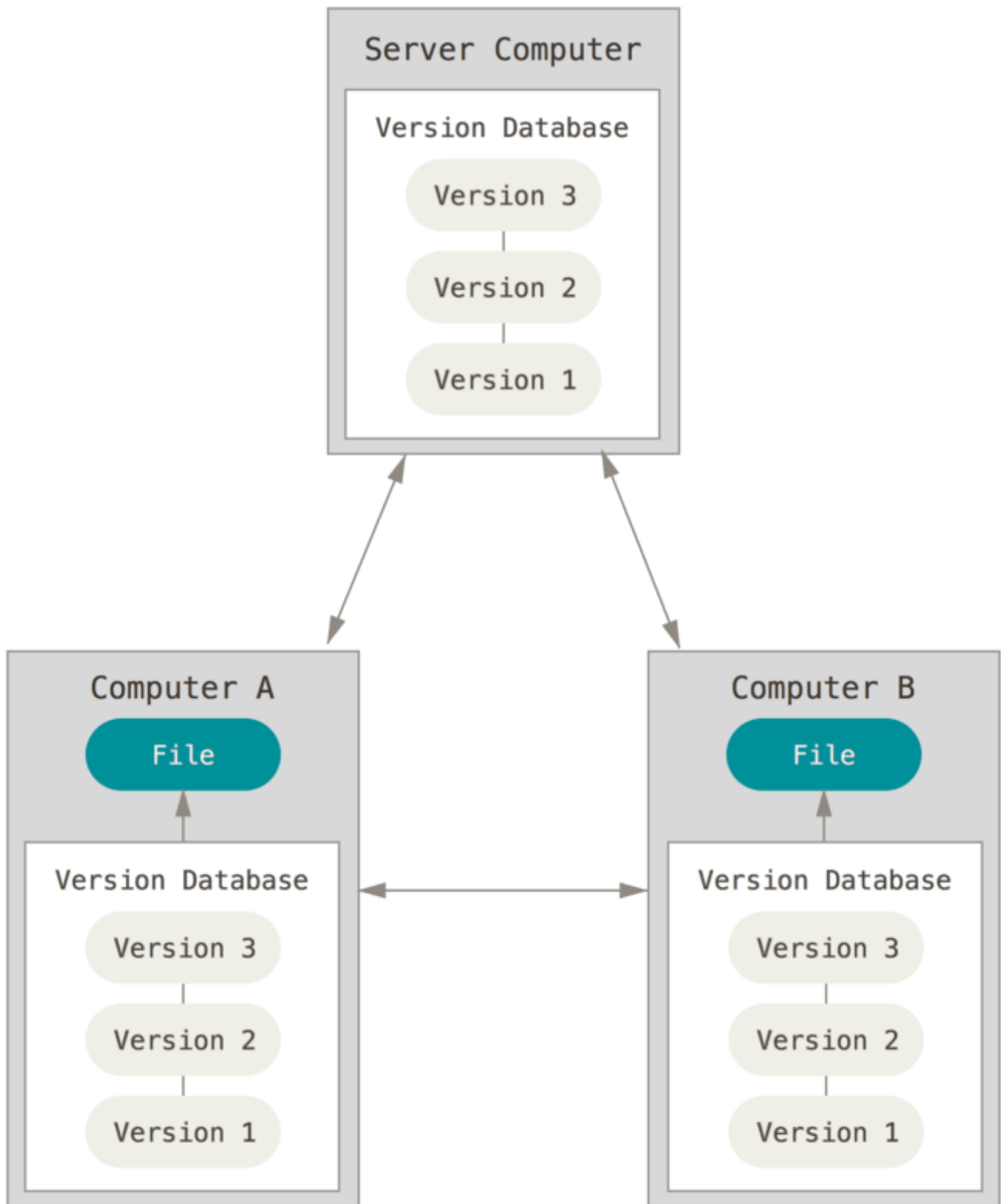


Figure 3. Control de versiones distribuido.

Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes 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.

Una breve historia de Git

Como muchas de las grandes cosas en esta vida, Git comenzó con un poco de destrucción creativa y una gran polémica.

El kernel de Linux es un proyecto de software de código abierto con un alcance bastante amplio. Durante la mayor parte del mantenimiento del kernel de Linux (1991-2002), los cambios en el software se realizaban a través de parches y archivos. En el 2002, el proyecto del kernel de Linux empezó a usar un DVCS propietario llamado BitKeeper.

En el 2005, la relación entre la comunidad que desarrollaba el kernel de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y la herramienta dejó de ser ofrecida de manera 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 mientras usaban BitKeeper. Algunos de **los objetivos del nuevo sistema** fueron los siguientes:

- 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)

Desde su nacimiento en el 2005, Git ha evolucionado y madurado para ser fácil de usar y conservar sus características iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal (véase [Ramificaciones en Git](#))

Fundamentos de Git

Entonces, ¿qué es Git en pocas palabras? Es muy importante entender bien esta sección, porque si entiendes lo que es Git y los fundamentos de cómo funciona, probablemente te será mucho más fácil usar Git efectivamente. A medida que aprendas Git, **intenta olvidar todo lo que posiblemente conoces** acerca de otros VCS como Subversion y Perforce. Hacer esto te ayudará a evitar confusiones sutiles a la hora de utilizar la herramienta. Git almacena y maneja la información de forma muy diferente a esos otros sistemas, a pesar de que su interfaz de usuario es bastante similar. Comprender esas diferencias evitará que te confundas a la hora de usarlo.

Copias instantáneas, no diferencias

La principal diferencia entre Git y cualquier otro VCS (incluyendo Subversion y sus amigos) es la forma en la que manejan sus datos. Conceptualmente, la mayoría de los otros sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.) manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.

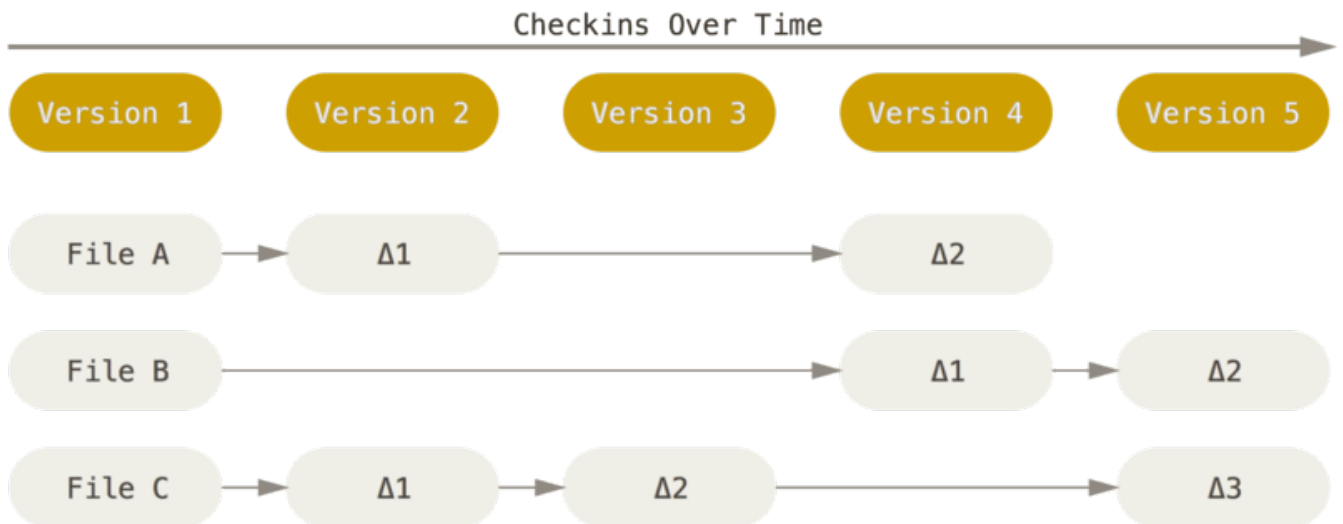


Figure 4. Almacenamiento de datos como cambios en una versión de la base de cada archivo.

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

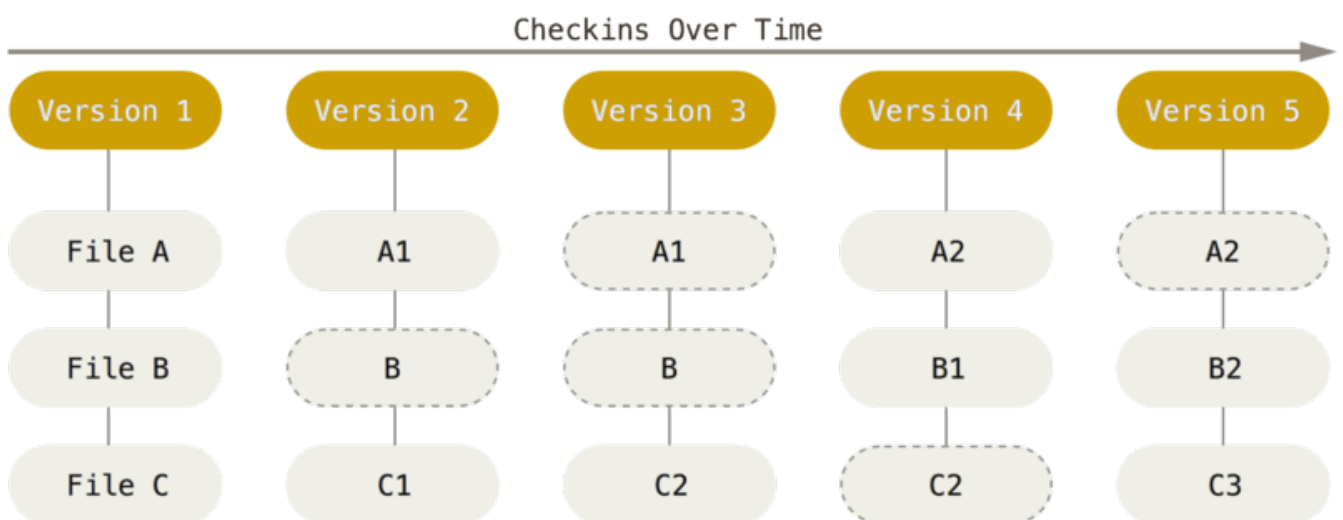


Figure 5. Almacenamiento de datos como instantáneas del proyecto a través del tiempo.

Esta es una diferencia importante entre Git y prácticamente todos los demás VCS. 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 sistema de archivos miniatura con algunas herramientas tremendamente poderosas desarrolladas sobre él, que a un VCS. Exploraremos algunos de los beneficios que obtienes al modelar tus datos de esta manera cuando veamos ramificación (branching) en Git en el (véase [Ramificaciones en Git](#)).

Casi todas las operaciones son locales

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Por lo general no se necesita información de ningún otro computador de tu red. Si estás acostumbrado a un CVCS donde la mayoría de las operaciones tienen el costo adicional 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. Debido a que **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 conectarse 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 instantáneamente. 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 puedes 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 engorroso. En Perforce, por ejemplo, no puedes hacer mucho cuando no estás conectado al servidor. 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 sea capaz de detectarlo.**

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 del directorio en Git. Un hash SHA-1 se ve de la siguiente forma:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Verás estos valores hash por todos lados en Git porque son usados con mucha frecuencia. De hecho, Git guarda todo no por nombre de archivo, sino por el valor hash de sus contenidos.

Git generalmente solo añade información

Cuando realizas acciones en Git, casi todas ellas solo añaden información a la base de datos de Git. Es muy difícil conseguir que el sistema haga algo que no se pueda enmendar, 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 copia instantánea en Git es muy difícil de perderla, especialmente si envías 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 estropear gravemente las cosas. Para un análisis más exhaustivo de cómo almacena Git su

información y cómo puedes recuperar datos aparentemente perdidos, ver [Deshacer Cosas](#).

Los Tres Estados

Ahora presta atención. **Esto es lo más importante que debes 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).

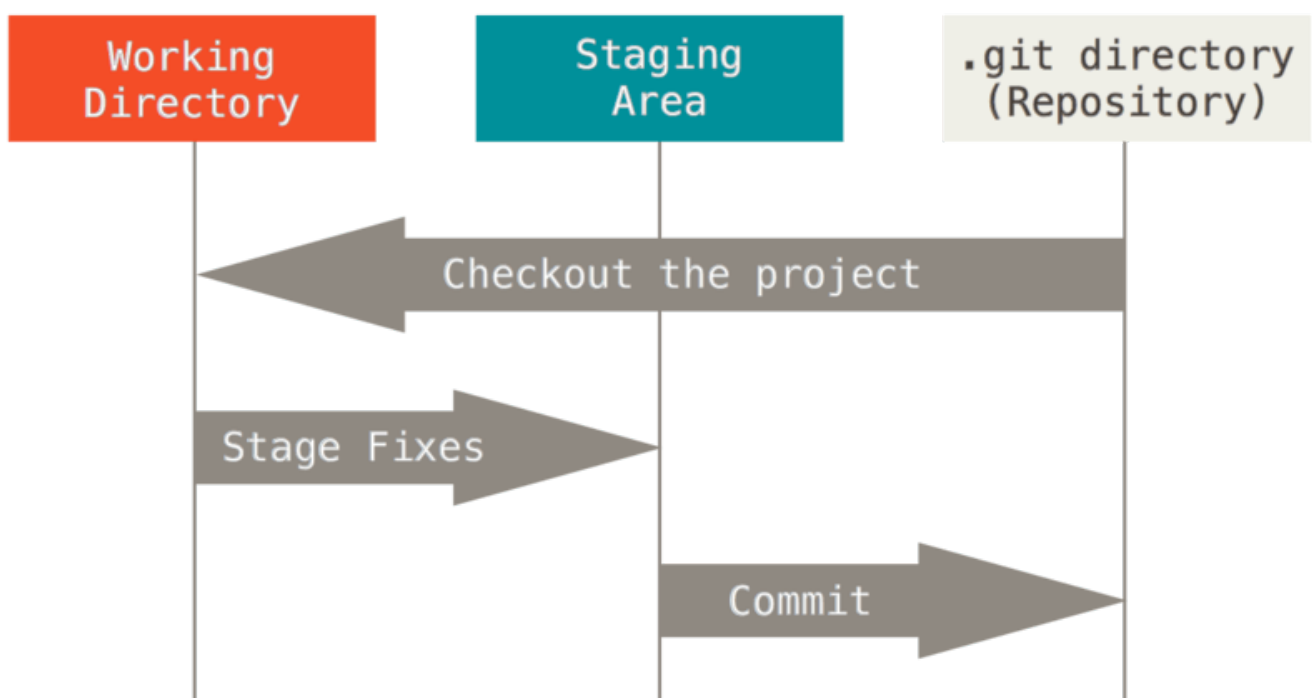


Figure 6. Directorio de trabajo, área de almacenamiento, y el directorio Git.

El directorio de Git es donde se almacenan 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 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 los puedas usar o modificar.

El área de preparación es un 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 (“index”), 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 esa copia instantánea 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 [Fundamentos de Git](#) aprenderás más acerca de estos estados y de cómo puedes aprovecharlos o saltarte toda la parte de preparación.

La Línea de Comandos

Existen muchas formas de usar Git. Por un lado tenemos las herramientas originales de línea de comandos, y por otro lado tenemos una gran variedad de interfaces de usuario con distintas capacidades. En ese libro vamos a utilizar Git desde la línea de comandos. **La línea de comandos en el único lugar en donde puedes ejecutar todos los comandos de Git** - la mayoría de interfaces gráficas de usuario solo implementan una parte de las características de Git por motivos de simplicidad. Si tú sabes cómo realizar algo desde la línea de comandos, seguramente serás capaz de averiguar cómo hacer lo mismo desde una interfaz gráfica. Sin embargo, la relación opuesta no es necesariamente cierta. Así mismo, la decisión de qué cliente gráfico utilizar depende totalmente de tu gusto, pero *todos* los usuarios tendrán las herramientas de línea de comandos instaladas y disponibles.

Nosotros esperamos que sepas cómo abrir el Terminal en Mac, o el "Command Prompt" o "Powershell" en Windows. **Si no entiendes de lo que estamos hablando aquí, te recomendamos que hagas una pausa para investigar acerca de esto de tal forma que puedas entender el resto de las explicaciones y descripciones que siguen en este libro.**

Instalación de Git

Antes de empezar a utilizar Git, tienes que instalarlo en tu computadora. Incluso si ya está instalado, este es posiblemente un buen momento para actualizarlo a su última versión. Puedes instalarlo como un paquete, a partir de un archivo instalador, o bajando el código fuente y compilándolo tú mismo.

NOTE

Este libro fue escrito utilizando la versión **2.0.0** de Git. Aun cuando la mayoría de comandos que usaremos deben funcionar en versiones más antiguas de Git, es posible que algunos de ellos no funcionen o funcionen ligeramente diferente si estás utilizando una versión anterior de Git. Debido a que Git es particularmente bueno en preservar compatibilidad hacia atrás, cualquier versión posterior a 2.0 debe funcionar bien.

Instalación en Linux

Si quieres instalar Git en Linux a través de un instalador binario, en general puedes hacerlo mediante la herramienta básica de administración de paquetes que trae tu distribución. Si estás en

Fedora por ejemplo, puedes usar yum:

```
$ yum install git
```

Si estás en una distribución basada en Debian como Ubuntu, puedes usar apt-get:

```
$ apt-get install git
```

Para opciones adicionales, la página web de Git tiene instrucciones para la instalación en diferentes tipos de Unix. Puedes encontrar esta información en <http://git-scm.com/download/linux>.

Instalación en Mac

Hay varias maneras de instalar Git en un Mac. Probablemente la más sencilla es instalando las herramientas Xcode de Línea de Comandos. En Mavericks (10.9) o superior puedes hacer esto desde el Terminal si intentas ejecutar `git` por primera vez. Si no lo tienes instalado, te preguntará si deseas instalarlo.

Si deseas una versión más actualizada, puedes hacerlo partir de un instalador binario. Un instalador de Git para OSX es mantenido en la página web de Git. Lo puedes descargar en <http://git-scm.com/download/mac>.

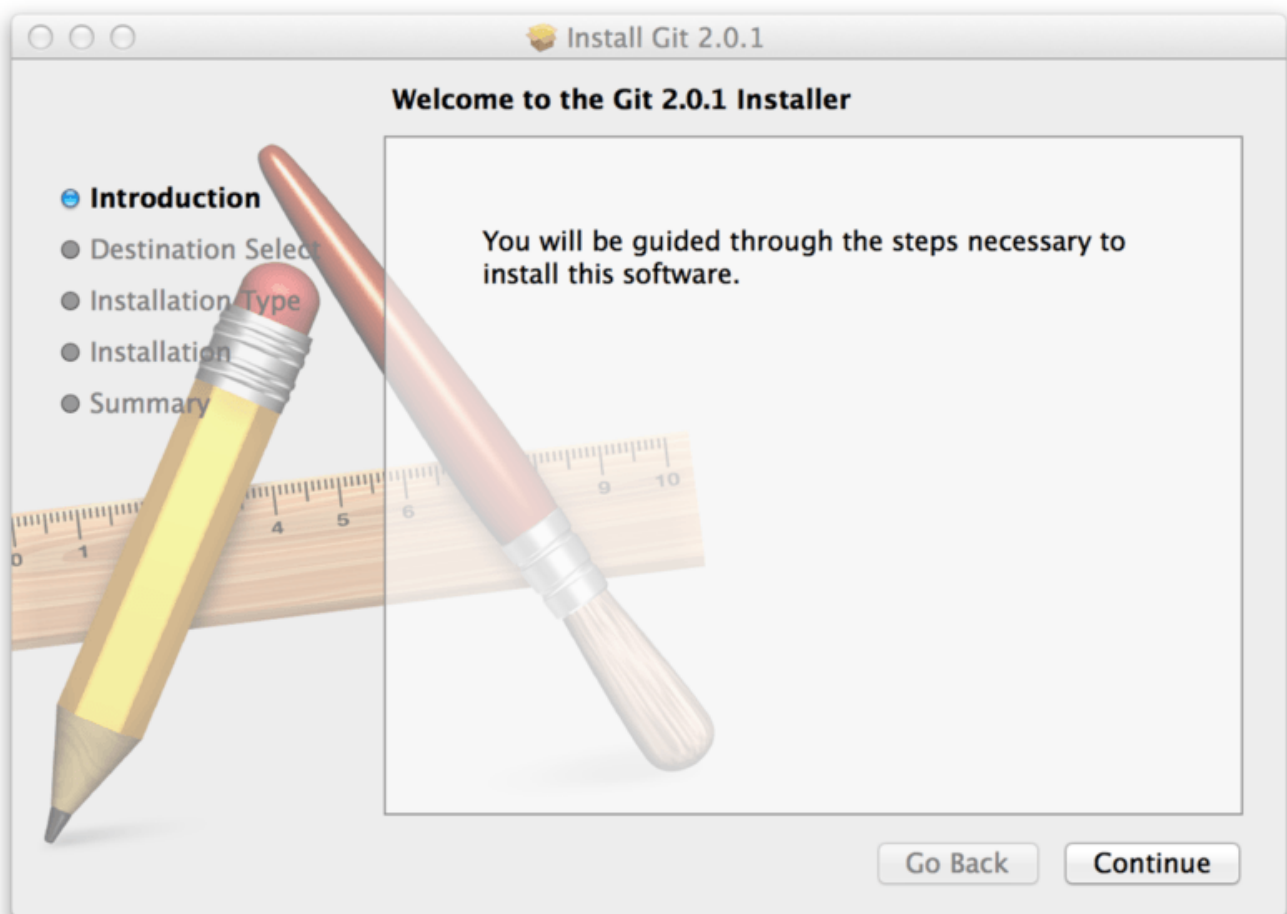


Figure 7. Instalador de Git en OS X.

También puedes instalarlo como parte del instalador de Github para Mac. Su interfaz gráfica de usuario tiene la opción de instalar las herramientas de línea de comandos. Puedes descargar esa herramienta desde el sitio web de Github para Mac en <http://mac.github.com>.

Instalación en Windows

También hay varias maneras de instalar Git en Windows. La forma más oficial está disponible para ser descargada en el sitio web de Git. Solo tienes que visitar <http://git-scm.com/download/win> y la descarga empezará automáticamente. Fíjate que éste es un proyecto conocido como Git para Windows (también llamado msysGit), el cual es diferente de Git. Para más información acerca de este proyecto visita <http://msysgit.github.io/>.

Otra forma de obtener Git fácilmente es mediante la instalación de GitHub para Windows. El instalador incluye la versión de línea de comandos y la interfaz de usuario de Git. Además funciona bien con Powershell y establece correctamente "caching" de credenciales y configuración CRLF adecuada. Aprenderemos acerca de todas estas cosas un poco más adelante, pero por ahora es suficiente mencionar que éstas son cosas que deseas. Puedes descargar este instalador del sitio web de GitHub para Windows en <http://windows.github.com>.

Instalación a partir del Código Fuente

Algunas personas desean instalar Git a partir de su código fuente debido a que obtendrás una versión más reciente. Los instaladores binarios tienen a estar un poco atrasados. Sin embargo, esto ha hecho muy poca diferencia a medida que Git ha madurado en los últimos años.

Para instalar Git desde el código fuente necesitas tener las siguientes librerías de las que Git depende: curl, zlib, openssl, expat y libiconv. Por ejemplo, si estás en un sistema que tiene yum (como Fedora) o apt-get (como un sistema basado en Debian), puedes usar estos comandos para instalar todas las dependencias:

```
$ yum install curl-devel expat-devel gettext-devel \
  openssl-devel zlib-devel
```

```
$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \
  libz-dev libssl-dev
```

Cuando tengas todas las dependencias necesarias, puedes descargar la versión más reciente de Git en diferentes sitios. Puedes obtenerlo a partir del sitio Kernel.org en <https://www.kernel.org/pub/software/scm/git>, o su "mirror" en el sitio web de GitHub en <https://github.com/git/git/releases>. Generalmente la más reciente versión en la página web de GitHub es un poco mejor, pero la página de kernel.org también tiene ediciones con firma en caso de que desees verificar tu descarga.

Luego tienes que compilar e instalar de la siguiente manera:

```
$ tar -zxf git-2.0.0.tar.gz
$ cd git-2.0.0
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```

Una vez hecho esto, también puedes obtener Git, a través del propio Git, para futuras actualizaciones:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

Configurando Git por primera vez

Ahora que tienes Git en tu sistema, vas a querer hacer algunas cosas para personalizar tu entorno de Git. Es necesario hacer estas cosas solamente una vez en tu computadora, y se mantendrán entre actualizaciones. También puedes cambiarlas en cualquier momento volviendo a ejecutar los comandos correspondientes.

Git trae una herramienta llamada `git config` que te permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git. Estas variables pueden almacenarse en tres sitios distintos:

1. Archivo `/etc/gitconfig`: Contiene valores para todos los usuarios del sistema y todos sus repositorios. Si pasas la opción `--system` a `git config`, lee y escribe específicamente en este archivo.
2. Archivo `~/.gitconfig` o `~/.config/git/config`: Este archivo es específico a tu usuario. Puedes hacer que Git lea y escriba específicamente en este archivo pasando la opción `--global`.
3. Archivo `config` en el directorio de Git (es decir, `.git/config`) del repositorio que estés utilizando actualmente: Este archivo es específico al repositorio actual.

Cada nivel sobrescribe los valores del nivel anterior, por lo que los valores de `.git/config` tienen preferencia sobre los de `/etc/gitconfig`.

En sistemas Windows, Git busca el archivo `.gitconfig` en el directorio `$HOME` (para mucha gente será `C:\Users\%USER%`). También busca el archivo `/etc/gitconfig`, aunque esta ruta es relativa a la raíz MSys, que es donde decidiste instalar Git en tu sistema Windows cuando ejecutaste el instalador.

Tu Identidad

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. **Esto es importante** porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

De nuevo, solo necesitas hacer esto una vez si especificas la opción `--global`, ya que Git siempre usará esta información para todo lo que hagas en ese sistema. Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando sin la opción `--global` cuando estés en ese proyecto.

Muchas de las herramientas de interfaz gráfica te ayudarán a hacer esto la primera vez que las uses.

Tu Editor

Ahora que tu identidad está configurada, puedes elegir el editor de texto por defecto que se utilizará cuando Git necesite que introduzcas un mensaje. Si no indicas nada, Git usa el editor por defecto de tu sistema, que generalmente es Vim. Si quieres usar otro editor de texto como Emacs, puedes hacer lo siguiente:

```
$ git config --global core.editor emacs
```

NOTE

Vim y Emacs son editores de texto frecuentemente usados por desarrolladores en sistemas basados en Unix como Linux y Mac. Si no estás familiarizado con ninguno de estos editores o estás en un sistema Windows, es posible que necesites buscar instrucciones acerca de cómo configurar tu editor favorito con Git. Si no configuras un editor así y no conoces acerca de Vim o Emacs, es muy factible que termines en un estado bastante confuso en el momento en que sean ejecutados.

Comprobando tu Configuración

Si quieres comprobar tu configuración, puedes usar el comando `git config --list` para mostrar todas las propiedades que Git ha configurado:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Puede que veas claves repetidas, porque Git lee la misma clave de distintos archivos (`/etc/gitconfig` y `~/.gitconfig`, por ejemplo). En ese caso, Git usa el último valor para cada clave única que ve.

También puedes comprobar qué valor que Git utilizará para una clave específica ejecutando `git`

config <key>:

```
$ git config user.name  
John Doe
```

¿Cómo obtener ayuda?

Si alguna vez necesitas ayuda usando Git, existen tres formas de ver la página del manual (manpage) para cualquier comando de Git:

```
$ git help <verb>  
$ git <verb> --help  
$ man git-<verb>
```

Por ejemplo, puedes ver la página del manual para el comando config ejecutando

```
$ git help config
```

Estos comandos son muy útiles porque puedes acceder a ellos desde cualquier sitio, incluso sin conexión. Si las páginas del manual y este libro no son suficientes y necesitas que te ayude una persona, puedes probar en los canales #git o #github del servidor de IRC Freenode (irc.freenode.net). Estos canales están llenos de cientos de personas que conocen muy bien Git y suelen estar dispuestos a ayudar.

Resumen

En este momento debes tener una comprensión básica de lo que es Git, y en que se diferencia de cualquier otro sistema de control de versiones centralizado que pudieras haber utilizado previamente. De igual manera, Git debe estar funcionando en tu sistema y configurado con tu identidad personal. Es hora de aprender los fundamentos de Git.