Engineer

Repositorio

almacén que guarda versiones de un conjunto de archivos

Qué es

Un repositorio, casi siempre abreviado como repo en una conversación, es donde un sistema de control de versiones conserva el registro completo de un proyecto: cada estado registrado de cada archivo, quién hizo cada cambio y por qué, y los nombres que dan acceso a ese historial. Los archivos de trabajo que edita son una vista de él; el repositorio es todo lo que hay detrás de esa vista.

Git define el término con precisión en su propio glosario: «Un conjunto de referencias junto con una base de datos de objetos que contiene todos los objetos accesibles desde esas referencias». Conviene explicar ambas partes. La base de datos de objetos guarda el contenido. Cada versión de un archivo, cada listado de un directorio y cada commit se almacena como un objeto identificado por un hash de su contenido. El libro de Git lo describe como un sistema de archivos direccionable por contenido: un almacén de clave y valor donde la clave se calcula a partir del valor. Dos archivos idénticos son un solo objeto, y cualquier cambio, por pequeño que sea, produce un nombre nuevo. Las referencias son la parte humana: ramas y etiquetas, nombres cortos que apuntan cada uno a un commit y ofrecen una vía de entrada.

La mayoría de los repositorios en los que se trabaja tienen asociado un árbol de trabajo. Los archivos que abre en un editor están en el directorio del proyecto, y el propio repositorio reside junto a ellos, en un directorio oculto llamado `.git`. Un repositorio desnudo, o bare, no tiene árbol de trabajo. Por convención, el nombre de su directorio termina en `.git`; solo contiene el historial y es la forma que conserva un servidor, porque allí nadie edita archivos: se envían cambios mediante push y se recuperan mediante fetch. El glosario de Git lo indica: quienes publican repositorios públicos suelen ofrecerlos como repositorios desnudos. El libro de Git muestra cómo crear uno a partir de un proyecto existente con `git clone --bare my_project my_project.git`.

El diseño de Git se aprecia al clonar. Un clon recibe casi todos los datos del servidor: de forma predeterminada, descarga cada versión de cada archivo de todo el historial del proyecto. Por tanto, cada clon es un repositorio completo, capaz de sustituir al servidor si este se pierde. Las excepciones son pequeñas pero importantes, y el libro de Git las enumera: los hooks del servidor y otros ajustes similares no se copian.

Hay dos variantes frecuentes. Un repositorio superficial, creado al clonar con `--depth`, solo conserva los commits recientes; el glosario dice que los commits padre más antiguos quedan «cauterizados», y más adelante se puede ampliar la profundidad del historial mediante fetch. Un submódulo es un repositorio dentro de otro, con su propio historial, que el repositorio exterior fija en un commit exacto.

Utilice el concepto en una decisión

Aplique «¿Qué es un repositorio?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cambios probados, publicados y reversibles, para que lleguen a sus clientes un martes por la tarde en vez de un viernes por la noche.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Despliegue GIS SaaS con GitHub Actions — despliegue, datos e informes automatizados

Despliegue GIS SaaS con GitHub Actions Póngase en contacto →

Por qué importa

Un repositorio es la memoria de un proyecto. Si una línea de código parece incorrecta, su historial dice cuándo llegó, en qué commit y con qué explicación. Esa memoria solo es tan buena como los commits que se escriben. Por eso, los equipos que cuidan su historial establecen reglas para los mensajes de commit y las hacen cumplir mediante hooks, en vez de confiar en la buena voluntad.

También es la unidad sobre la que trabaja la mayoría de las herramientas. El acceso se concede por repositorio, los hooks se ejecutan por repositorio y la integración continua suele partir de uno. Delimitar un repositorio es delimitar lo que abarcan una revisión, una comprobación y un permiso.

Los repositorios conservan lo que reciben. Un commit que borra un archivo añade un estado nuevo; deja todos los estados anteriores donde estaban, en cada clon que ya los contenga. Esa permanencia es la razón de ser del control de

versiones, y tiene dos caras: una versión antigua siempre se puede recuperar, pero también una contraseña que alguien haya incluido por error en un commit.

Por último, un repositorio puede contener mucho más que código de aplicación. Las definiciones de infraestructura, la documentación y la configuración obtienen las mismas ventajas: un historial, un paso de revisión y una forma de reconstruir un estado conocido. Alguien que nunca haya visto un servidor puede reconstruirlo si su configuración vive en un repositorio.

Cómo se aplica en la práctica

Cuatro repositorios construidos por una persona durante dieciocho meses comparten una norma para los commits: un prefijo convencional que indica el tipo de cambio, un ámbito, un asunto cuya longitud es inferior a un límite fijado, sin emojis ni líneas finales de atribución. Un hook de mensajes de commit presente en cada repositorio donde se realiza trabajo rechaza cualquier incumplimiento. Así, la norma pasa a ser una propiedad del repositorio, no de quien hace el commit. El historial muestra entonces en qué consistió el trabajo. En la etapa de implementación de julio de 2026 que quedó documentada, el historial del repositorio más grande contenía 156 commits de funcionalidades, 133 correcciones, 128 de documentación, 81 de mantenimiento, 26 refactorizaciones, 20 de estilo, 5 de rendimiento y 3 de pruebas. El hook no llega a evaluar la calidad: comprueba el formato, no el contenido.

Para la infraestructura propia de la empresa, los servicios desplegados de acceso público y la configuración de otros componentes, el encargo era describirlos en un repositorio en vez de depender de la memoria. La implementación registrada contenía 12.065 líneas de YAML repartidas en 19 playbooks y 34 roles; esas cifras describen aquella etapa, no el repositorio actual. Allí se configura la automatización de las copias de seguridad y del correo, cuya activación operativa es un paso separado. La infraestructura se puede reproducir desde el repositorio: cada host pertenece a un grupo correspondiente a un nivel de infraestructura cuyas variables declaran los roles que ejecuta, y un host que ya ha convergido informa de cero cambios. Cada modificación lleva más tiempo que editar directamente en el servidor, y después se añadió un play de comprobaciones previas antes de la convergencia.

Un repositorio también conserva lo que recibió mientras la atención estaba en otra parte. El trabajo para otra empresa se arrastró durante una reestructuración del repositorio, incluido un registro provisional con credenciales activas en texto

claro que permaneció en el árbol sin que ningún control lo detectara durante la mayor parte del historial del repositorio. Nada de ello se ejecutaba, y precisamente por eso sobrevivió. La respuesta fue una regla de ámbito al principio de las instrucciones del repositorio: la infraestructura de una sola empresa y nada más, además de una cuarentena en lugar de un borrado. Así, el historial sigue siendo legible mientras el detector de secretos, el control de la bóveda y la comprobación de emojis continúan leyendo los archivos en cuarentena.

Dónde surgen los problemas

- **Un secreto incluido una vez en un commit sobrevive a su borrado.** Quitar el archivo en el siguiente commit lo deja en el anterior y en todos los clones creados desde entonces. Trate la credencial como pública: rótela primero y decida después si reescribir el historial compensa las molestias para todas las personas que tienen una copia.
- **Un push intenta actualizar la rama actual.** Enviar cambios a un repositorio no desnudo cuya rama actual es la que se actualiza dejaría el árbol de trabajo desincronizado con su historial. Por eso, Git rechaza de forma predeterminada ese push mediante `receive.denyCurrentBranch`. Un repositorio que recibe pushes debería ser desnudo.
- **Un clon superficial solo conoce parte del historial.** Un clon creado con `--depth` solo contiene commits recientes, de modo que `git log`, `git blame` y cualquier script que deduzca una versión a partir de la última etiqueta solo ven ese fragmento. Los procesos de construcción y publicación que necesiten el pasado deberían recuperar todo el historial.
- **Un hook local termina en el siguiente clon.** Los hooks del cliente viven en `.git/hooks`, y el libro de Git es explícito: clonar no los copia. Una política aplicada solo ahí deja de funcionar el día que alguien crea otro clon. Mantenga los scripts de hooks versionados en el repositorio y haga que `core.hooksPath` apunte a ellos, o aplique la regla en el servidor con `pre-receive`.
- **Los binarios grandes aumentan el tamaño de cada clon para siempre.** Como un clon contiene cada versión de cada archivo, cada copia sustituida de un binario grande permanece en el historial, y cada clon nuevo las descarga todas.
- **Los archivos que nada ejecuta se quedan sin leer.** Una tarea desactivada, un inventario obsoleto o un registro provisional sigue en todos los clones, pero ninguna prueba, construcción ni despliegue lo toca, de modo que escapa a la revisión. El contenido del repositorio debería corresponder a las responsabilidades del proyecto; cualquier otra cosa debería retirarse o apartarse donde los controles aún puedan verla.

Lecturas adicionales

- gitglossary: repositorio — La definición de Git, junto a las de repositorio desnudo, árbol de trabajo, repositorio superficial y submódulo.
- Obtener un repositorio Git — Las dos formas de conseguirlo, empezar en un directorio o clonar, y lo que descarga un clon.
- Objetos de Git — Cómo la base de datos de objetos almacena contenido bajo un hash de ese contenido.
- gitrepository-layout — La función de cada archivo y directorio dentro de un repositorio.
- Instalar Git en un servidor — Convertir un repositorio existente en uno desnudo y compartirlo.
- Hooks de Git — Hooks del cliente y del servidor, y por qué un clon deja atrás los del cliente.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un repositorio?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/repository/