WIKIPEDIA

Git

Git (pronunciado "guit"[2]) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

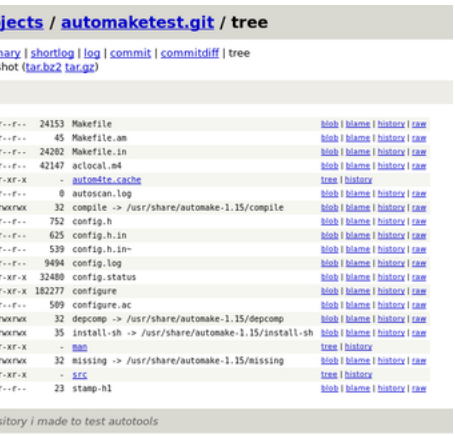
Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. [3] Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. [4] Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

Índice

Git

git-scm.com (https://git-scm.com)



Tipo de programa	Control de versiones distribuido open science tool protocolo de comunicaciones software libre
Modelo de desarrollo	Software libre
Desarrollador	Linus Torvalds Junio Hamano Software Freedom Conservancy
Autor	Linus Torvalds
Lanzamiento	7 de abril de 2005
Última versión estable	2.26.0 (info (https://git-scm.com/)) 22 de marzo de 2020 (19 días)
Género	Control de versiones
Programado en	C, Bourne Shell, Perl[1]
Sistema operativo	Unix-like, Windows, Linux
Licencia	GNU GPL v2

Idiomas	inglés

Características

El diseño de Git se basó en BitKeeper y en Monotone.[5][6] Originalmente fue diseñado como un motor de sistema de control de versiones de bajo nivel sobre el cual otros podrían codificar interfaces frontales, tales como Cogito o StGIT.[7] Desde ese entonces hasta ahora el núcleo del proyecto Git se ha vuelto un sistema de control de versiones completo, utilizable en forma directa.[8]

Linus Torvalds buscaba un sistema distribuido que pudiera usar en forma semejante a BitKeeper, pero ninguno de los sistemas bajo software libre disponibles cumplía con sus requerimientos, especialmente en cuanto a desempeño. El diseño de Git mantiene una enorme cantidad de código distribuida y gestionada por mucha gente, que incide en numerosos detalles de rendimiento, y de la necesidad de rapidez en una primera implementación.

Entre las características más relevantes se encuentran:

- Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Una presunción fundamental en Git, es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.
- Gestión distribuida. Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.
- Los almacenes de información pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH. Git también puede emular servidores CVS, lo que habilita el uso de clientes CVS pre-existentes y módulos IDE para CVS pre-existentes en el acceso de repositorios Git.
- Los repositorios Subversion y svk se pueden usar directamente con git-svn.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.
- Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial). Esto existía en Monotone.
- Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja con base en cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos.
- Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles y desastrosas coincidencias de ficheros diferentes en un único nombre.
- Realmacenamiento periódico en paquetes (ficheros). Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (con base en diferencias) no ocurre cada cierto tiempo.

Órdenes básicas

- `git init`:

Git ciclo de vida de archivos git

Esto crea un subdirectorio nuevo llamado .git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.

- `git fetch`:

Descarga los cambios realizados en el repositorio remoto.

- `git merge <nombre_rama>`:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre_rama”.

- `git pull`:

Unifica los comandos *fetch* y *merge* en un único comando.

- `git commit -am "<mensaje>"`:

Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al *commit* una breve descripción de los cambios realizados.

- `git push origin <nombre_rama>`:

Sube la rama “nombre_rama” al servidor remoto.

- `git status`:

Muestra el estado actual de la rama, como los cambios que hay sin commitear.

- `git add <nombre_archivo>`:

Comienza a trackear el archivo “nombre_archivo”.

- `git checkout -b <nombre_rama_nueva>`:

Crea una rama a partir de la que te encuentres parado con el nombre “nombre_rama_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

- `git checkout -t origin/<nombre_rama>`:

Si existe una rama remota de nombre “nombre_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

- `git branch`:

Lista todas las ramas locales.

- `git branch -a`:

Lista todas las ramas locales y remotas.

- `git branch -d <nombre_rama>`:

Elimina la rama local con el nombre “nombre_rama”.

- `git push origin <nombre_rama>`:

Commitea los cambios desde el branch local origin al branch “nombre_rama”.

- `git remote prune origin`:

Actualiza tu repositorio remoto en caso que algún otro desarrollador haya eliminado alguna rama remota.

- `git reset --hard HEAD`:

Elimina los cambios realizados que aún no se hayan hecho *commit*.

- `git revert <hash_commit>`:

Revierte el *commit* realizado, identificado por el “hash_commit”.

Flujo de trabajo

Git plantea una gran libertad en la forma de trabajar en torno a un proyecto. Sin embargo, para coordinar el trabajo de un grupo de personas en torno a un proyecto es necesario acordar como se va a trabajar con Git. A estos acuerdos se les llama **flujo de trabajo** [9] Un flujo de trabajo de Git es una fórmula o una recomendación acerca del uso de Git para realizar trabajo de forma uniforme y productiva. [10] Los flujos de trabajo más populares son git-flow, GitHub-flow, GitLab Flow y One Flow. [11]

Git-Flow

Creado en 2010 por Vincent Driessen.[11] Es el flujo de trabajo más conocido. Está pensado para aquellos proyectos que tienen entregables y ciclos de desarrollo bien definidos. [9] Está basado en dos grandes ramas con infinito tiempo de vida (ramas master y develop) y varias ramas de apoyo, unas orientadas al desarrollo de nuevas funcionalidades (ramas feature-*), otras al arreglo de errores (hotfix-*) y otras orientadas a la preparación de nuevas versiones de producción (ramas release-*). La herramienta gitflow [1] (https://github.com/nvie/gitflow) facilita la automatización de las tareas implicadas en este flujo de trabajo[11]

Master

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

Development

Es una rama sacada de *Master*. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama *Master*.

Features

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de *Development*. Una vez que la funcionalidad esté desarrollada, se hace un merge de la rama sobre *Development*, donde se integrará con las demás funcionalidades.

Hotfix

Son errores de software que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de *Master*. Una vez corregido el error, se debe hacer una unificación de la rama sobre *Master*. Al final, para que no quede desactualizada, se debe realizar la unificación de *Master* sobre *Development*.

Release

Las ramas de release apoyan la preparación de nuevas versiones de producción. Para ellos se arreglan muchos errores menores y se preparan adecuadamente los metadatos. Se suelen original de la rama develop y deben fusionarse en las ramas master y develop.[11]

GitHub-Flow

Creado en 2011 por GitHub [11] y es la forma de trabajo sugerida por las funcionalidades propias de GitHub . Está centrado en un modelo de desarrollo iterativo y de despliegue constante. Está basado en cuatro principios:[9] [11]

- Todo lo que está en la rama master está listo para ser puesto en producción
- Para trabajar en algo nuevo, debes crear una nueva rama a partir de la rama master con un nombre descriptivo. El trabajo se irá integrando sobre esa rama en local y regularmente también a esa rama en el servidor
- Cuando se necesite ayuda o información o cuando creemos que la rama está lista para integrarla en la rama master, se debe abrir una pull request (solicitud de integración de cambios).
- Alguien debe revisar y visar los cambios para fusionarlos con la rama master
- Los cambios integrados se pueden poner en producción.

GitHub intenta simplificar la gestión de ramas, trabajando directamente sobre la rama master y generando integrando las distintas features directamente a esta rama[12]

GitLab Flow

Creado en 2014 por Gitlab. [11] Es una especie de extensión de GitHub Flow acompañado de un conjunto de pautas y mejores prácticas que apuntan a estandarizar aún más el proceso. Al igual que GitHub Flow propone el uso de ramas de funcionalidad (feature) que se originan a partir de la rama master y que al finalizarse se mezclan con la rama master. Además introduce otros tres tipos de ramas:[13]

- Ramas de entorno. Por ejemplo pre-production production. Se crean a partir de la rama master cuando estamos listos para implementar nuestra aplicación. Si hay un error crítico lo podemos arreglar en un rama y luego mezclarla en la rama de entorno.
- Ramas de versión. Por ejemplo 1.5-stable 1.6-stable. El flujo puede incluir ramas de versión en caso de que el software requiera lanzamientos frecuentes.

One Flow

Creado en 2015 por Adam Ruka. En él cada nueva versión de producción está basada en la versión previa de producción. La mayor diferencia entre One Flow y Git Flow es que One Flow no tiene rama de desarrollo. [11]

Software que los usa como base

Git se ha usado como software base sobre el que el que se han desarrollado otros proyectos

- Gerrit. Aplicación web que permite la revisión de código en equipo para la aprobación o rechazo de modificaciones. Consiste de un repositorio Git que actúa como intermediario entre los gits de los desarrolladores y el repositorio Git que usa la integración continua. Los desarrolladores en lugar de enviar el código al sistema de integración continua lo envían al repositorio Gerrit donde se han establecido político. [14]
- Las plataformas de forja son plataformas web que ofrecen servicios que permiten el desarrollo colaborativo de software. Uno de los servicios básicos ofrecidos es crear poder crear repositorio de software en un sistema de control de versiones concreto. El sistema de control de versiones más utilizado es Git. Ejemplos de estos servicios son GitLab, Bitbucket y GitHub. Se han desarrollado varias soluciones que permiten crear automáticamente forjas software como por ejemplo Gogs, Gitea, RhodeCode Community Edition o las versiones de plataformas ofrecidas por GitHub (GitHub Enterprise), Gitlab (GitLab CE y GitLab EE) y Bitbucket. [15] [16]
- SparkleShare . Es un cliente de código abierto que permite usar repositorios Git como servicios de alojamiento de archivos, permitiendo la sincronización de archivos y colaboración en la nube de forma similar a Dropbox. Los repositorios Git pueden estar en una máquina Linux, generalmente creados con la aplicación Dazzle [2] (https://github.com/hbons/Dazzle) o en repositorios git alojados en la nube usando servicios como GitLab, GitHub, Bitbucket o NotABug [3] (https://notabug.org).[17] [18]
- git-crypt. Herramienta de git que permite hacer cifrado de forma completamente transparente. Cuando se hace un git push los ficheros marcados para cifrar se cifren automáticamenteno. Análogamente se hará cuando se hace un git pull. De esta forma podemos tener un repositorio que tenga parte de sus archivos cifrados y otra parte sin cifrar. El cifrado se puede realizar con claves de cifrado simétrico (con GPG) o asimétrico.[19]
- Herramientas, como gitflow [4] (https://github.com/nvie/gitflow), que facilitan la automatización de las tareas implicadas en cierto/s flujo/s de trabajo.

Véase también

- Control de versiones
- Programas para control de versiones
- GitLab
- GitHub
- Gitea
- CVS
- Subversion (software)
- Repositorio

Referencias

1. «git/git.git/tree» (https://git.kernel.org/?p=git/git.git;a=tree). git.kernel.org. Consultado el 15 de junio de 2009.
2. «Tech Talk: Linus Torvalds on git (at 00:01:30)» (https://www.youtube.com/watch?v=4XpnKHJAok8&t=1m30s). YouTube. Consultado el 20 de julio de 2014.
3. Linus Torvalds (8 de abril de 2005). «Re: Kernel SCM saga» (http://marc.info/?l=linux-kernel&m=111293537202443).
4. Linus Torvalds (23 de marzo de 2006). «Re: Errors GITtifying GCC and Binutils» (http://marc.info/?l=git&m=114314642000462).
5. Linus Torvalds (5 de mayo de 2006). «Re: [ANNOUNCE] Git wiki» (http://marc.info/?l=git&m=114685143200012). Referencias de los antecesores de Git
6. LKML: Linus Torvalds: Re: Kernel SCM saga (http://lkml.org/lkml/2005/4/8/9)
7. Torvalds, Linus (8 de abril de 2005), «Re: Kernel SCM saga (https://marc.info/?l=linux-kernel&m=111293537202443)», *lista de correo linux-kernel*, https://marc.info/?l=linux-kernel&m=111293537202443, consultado el 20 de febrero de 2008.
8. Torvalds, Linus (23 de marzo de 2006), «Re: Errors GITtifying GCC and Binutils (https://marc.info/?l=git&m=114314642000462)», *lista de correo git*, https://marc.info/?l=git&m=114314642000462.
9. Flujos de trabajo en git (https://www.yukei.net/2013/08/flujos-de-trabajo-en-git/). Felipe Lavín Z. yukei.net. 10 de agosto de 2013
10. Comparar workflows (https://www.atlassian.com/es/git/tutorials/comparing-workflows). atlassian.com
11. 4 branching workflows for Git (https://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf). Patrick Porto. medium.com. 27 de febrero de 2018
12. Git - Como gestionar y cuidar nuestro código (https://enmilocalfunciona.io/git-como-gestionar-y-cuidar-nuestro-codigo/). Antonio García Candil. enmilocalfunciona.io. 12 de junio de 2018
13. ¿Cuál es la diferencia entre GitHub Flow y GitLab Flow? (http://git.dokry.com/cul-es-la-diferencia-entre-github-flow-y-gitlab-flow.html).
14. Gerrit, un sistema de revisión de código muy jugoso (http://brigomp.blogspot.com/2011/09/gerrit-un-sistema-de-revision-de-codigo.html). Martín Pérez. http://brigomp.blogspot.com. 20 de septiembre de 2011
15. Un servidor git con frontal web: Gitea (https://www.linuxsysadmin.ml/2018/06/un-servidor-git-con-frontal-web-gitea.html). Gerard Monells. inuxsysadmin.ml. 11 de Junio de 2018
16. Gogs vs gitea : de lo conservador y cerrado a los berrinches cambiantes (http://venenuxmassenkoh.blogspot.com/2017/09/gogs-vs-gitea-de-lo-conservador-y.html). VENENUX SariSariNama/Massenkoh. 13 de septiembre de 2017
17. SparkleShare: herramienta para la sincronización de archivos en la nube (https://blog.desdelinux.net/sparkleshare-una-herramienta-para-la-sincronizacion-de-archivos-en-la-nube/). David Naranjo. desdelinux.net]]

18. File sharing with Git (https://opensource.com/article/19/4/file-sharing-git). Seth Kenlon. opensource.com. 5 de abril de 2019
19. Cifrado de repositorios Git (https://www.atareao.es/como/cifrado-de-repositorios-git/). atareao.es. 5 de abril de 2019

Enlaces externos

- Sitio web oficial (http://git-scm.com/)
