**¿QUE ES GIT?**

Hoy en día, Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

**CARACTERISTICAS**

**Rendimiento**

Las características básicas de rendimiento de Git son muy sólidas en comparación con muchas otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado en favor del rendimiento. Los algoritmos implementados en Git aprovechan el profundo conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente, cómo suelen modificarse con el paso del tiempo y cuáles son los patrones de acceso.

A diferencia de algunos programas de software de control de versiones, Git no se deja engañar por los nombres de los archivos a la hora de determinar cuál debería ser el almacenamiento y el historial de versiones del árbol de archivos; en lugar de ello, se centra en el contenido del propio archivo. Al fin y al cabo, los archivos de código fuente se cambian de nombre, se dividen y se reorganizan con frecuencia. El formato de objeto de los archivos del repositorio de Git emplea una combinación de codificación delta (que almacena las diferencias de contenido) y compresión, y guarda explícitamente el contenido de los directorios y los objetos de metadatos de las versiones.

Su arquitectura distribuida también permite disfrutar de importantes ventajas en términos de rendimiento.

Por ejemplo, supongamos que una desarrolladora, Alice, hace cambios en el código fuente (añade una función para la próxima versión 2.0) y, luego, los confirma con mensajes descriptivos. Después, trabaja en una segunda función y confirma también esos cambios. De forma natural, estos se almacenan como elementos independientes de trabajo en el historial de versiones. A continuación, Alice cambia a la rama de la versión 1.3 del mismo software para corregir un error que afecta únicamente a esa versión anterior. El objetivo es permitir al equipo de Alice lanzar una publicación de corrección de errores, la versión 1.3.1, antes de que la 2.0 esté lista. Tras ello, Alice puede volver a la rama 2.0 para seguir trabajando en las nuevas funciones de la versión. Todo esto puede tener lugar sin necesidad de acceso a la red y, por consiguiente, es un proceso rápido y fiable. Alice podría incluso hacerlo mientras viaja en avión. Cuando esté lista para enviar al repositorio remoto todos los cambios confirmados de modo individual, bastará con que utilice un solo comando.

**Seguridad**

Git se ha diseñado con la principal prioridad de conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, todos ellos objetos del repositorio de Git, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se salvaguarda el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, y se garantiza que el historial sea totalmente trazable.

Con Git, puedes tener la certeza de contar con un auténtico historial de contenido de tu código fuente.

Algunos otros sistemas de control de versiones carecen de protección contra las modificaciones ocultas realizadas con posterioridad, algo que puede suponer una grave vulnerabilidad de seguridad de la información para cualquier organización que se base en el desarrollo de software.

**Flexibilidad**

Uno de los objetivos clave de Git en cuanto al diseño es la flexibilidad. Git es flexible en varios aspectos: en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos.

Git se ha ideado para posibilitar la ramificación y el etiquetado como procesos de primera importancia (a diferencia de SVN) y las operaciones que afectan a las ramas y las etiquetas (como la fusión o la reversión) también se almacenan en el historial de cambios. No todos los sistemas de control de versiones ofrecen este nivel de seguimiento.

**Control de versiones con Git**

Git es la mejor opción para la mayoría de los equipos de software actuales. Aunque cada equipo es diferente y debería realizar su propio análisis, aquí recogemos los principales motivos por los que destaca el control de versiones de Git con respecto a otras alternativas:

* **Git es una excelente herramienta**

Git tiene la funcionalidad, el rendimiento, la seguridad y la flexibilidad que la mayoría de los equipos y desarrolladores individuales necesitan. Estas cualidades de Git se detallan más arriba. En las comparaciones directas con gran parte de las demás alternativas, Git resulta muy ventajoso para muchos equipos.

* **Git es un estándar de facto**

Git es la herramienta con el mayor índice de adopción de su clase, lo que la hace muy atractiva por las siguientes razones. En Atlassian, casi todo el código fuente de nuestros proyectos se gestiona en Git.

Un gran número de desarrolladores ya tienen experiencia con Git y una parte importante de los graduados universitarios puede que solo haya aprendido a usar dicha solución. Aunque algunas organizaciones puedan necesitar escalar la curva de aprendizaje al migrar a Git desde otro sistema de control de versiones, muchos de sus desarrolladores actuales y futuros no precisan de formación para utilizar esta herramienta.

Además de las ventajas que brinda disponer de una amplia base de talentos, el predominio de Git también implica que muchos servicios y herramientas de software de terceros ya están integrados con Git, incluidos los IDE, y nuestras propias herramientas, como el cliente de escritorio de DVCS, Sourcetree; el software de seguimiento de proyectos e incidencias, Jira; y el servicio de alojamiento de código, Bitbucket.

Si eres un desarrollador sin experiencia que quiere adquirir valiosas destrezas con herramientas de desarrollo de software, en lo que se refiere al control de versiones, Git debería estar en tu lista.

* **Git es un proyecto de código abierto de calidad**

Git es un proyecto de código abierto muy bien respaldado con más de una década de gestión de gran fiabilidad. Los encargados de mantener el proyecto han demostrado un criterio equilibrado y un enfoque maduro para satisfacer las necesidades a largo plazo de sus usuarios con publicaciones periódicas que mejoran la facilidad de uso y la funcionalidad. La calidad del software de código abierto resulta sencilla de analizar y un sinnúmero de empresas dependen en gran medida de esa calidad.

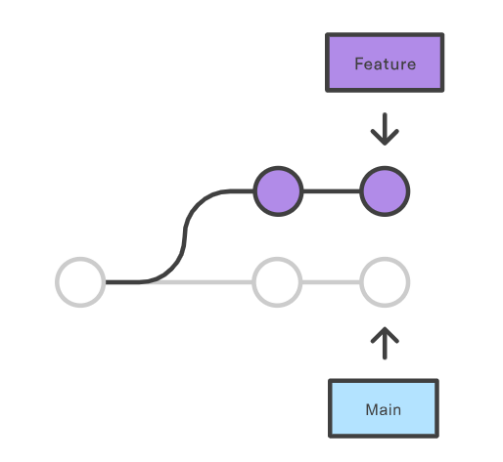
Git goza de una amplia base de usuarios y de un gran apoyo por parte de la comunidad. La documentación es excepcional y para nada escasa, ya que incluye libros, tutoriales y sitios web especializados, así como podcasts y tutoriales en vídeo.

El hecho de que sea de código abierto reduce el coste para los desarrolladores aficionados, puesto que pueden utilizar Git sin necesidad de pagar ninguna cuota. En lo que respecta a los proyectos de código abierto, no cabe duda de que Git es el sucesor de las anteriores generaciones de los exitosos sistemas de control de versiones de código abierto, SVN y CVS.

**USOS:**

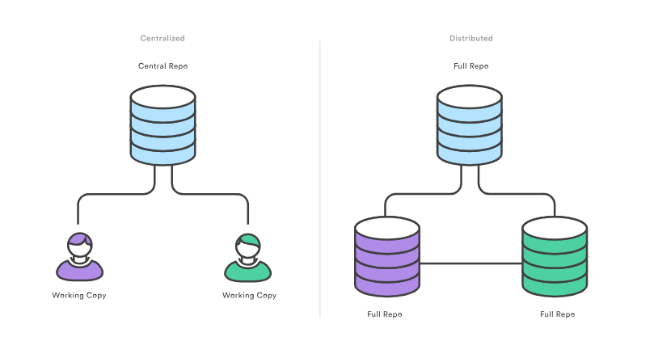
**Git para desarrolladores**

* **Flujo de trabajo de rama de función**

Una de las mayores ventajas de Git son sus capacidades de ramificación. A diferencia de los sistemas de control de versiones centralizados, las ramas de Git son baratas y fáciles de fusionar. Esto facilita el flujo de trabajo de ramas de función tan popular entre muchos usuarios de Git.

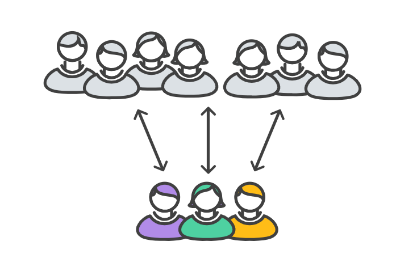
* **Desarrollo distribuido**

En SVN, cada desarrollador obtiene una copia de trabajo que apunta a un único repositorio central. No obstante, Git es un sistema de control de versiones distribuido. En lugar de una copia de trabajo, cada desarrollador obtiene su propio repositorio local con un historial completo de confirmaciones.



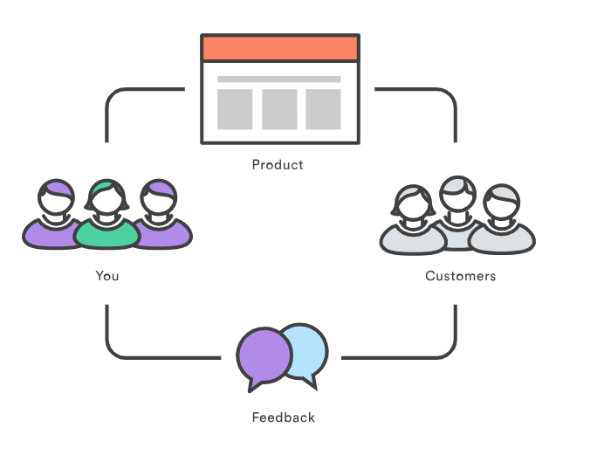
* Comunidad

En muchos círculos, Git se ha convertido en el sistema de control de versiones esperado para nuevos proyectos. Si tu equipo usa Git, lo más probable es que no tengas que formar a los nuevos empleados sobre tu flujo de trabajo, porque ya estarán familiarizados con el desarrollo distribuido.



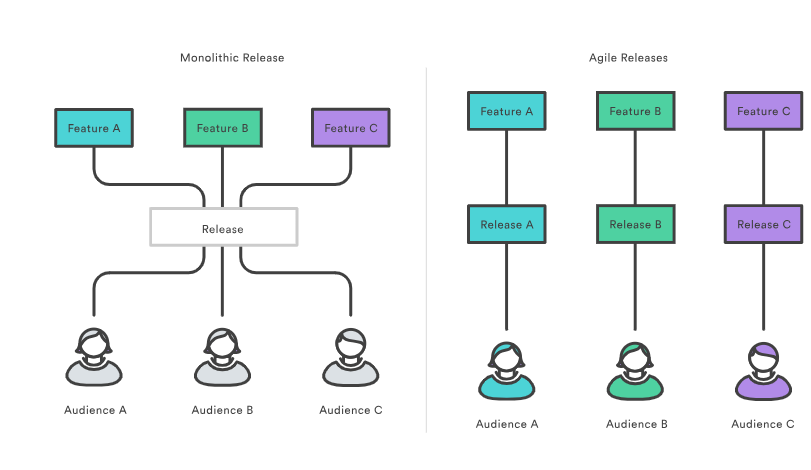
* Ciclo de publicación más rápido

El resultado final de las ramas de función, el desarrollo distribuido, las solicitudes de incorporación de cambios y una comunidad estable es un ciclo de publicación más rápido. Estas capacidades facilitan un flujo de trabajo ágil en el que se anima a los desarrolladores a compartir cambios más pequeños con mayor frecuencia. A su vez, los cambios pueden aplicarse a través de la canalización de implementación más rápido que con las publicaciones monolíticas comunes con los sistemas de control de versiones centralizados.



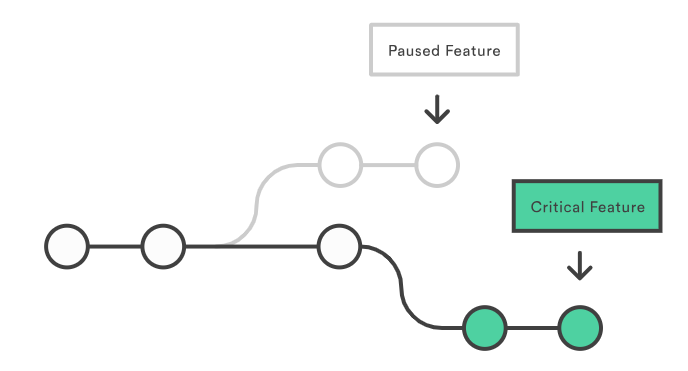
**Git para marketing**

Por ejemplo, podrían preparar un gran envío de solicitudes de incorporación de cambios para la función innovadora, una entrada de blog corporativo y un blurb de boletín informativo para la función de Mary, así como algunas entradas de invitado sobre la teoría de Rick en cuanto a la experiencia de usuario subyacente para enviarlas a blogs de diseño externos. Todas estas actividades se pueden sincronizar con una publicación independiente.



**Git para gestión de productos**

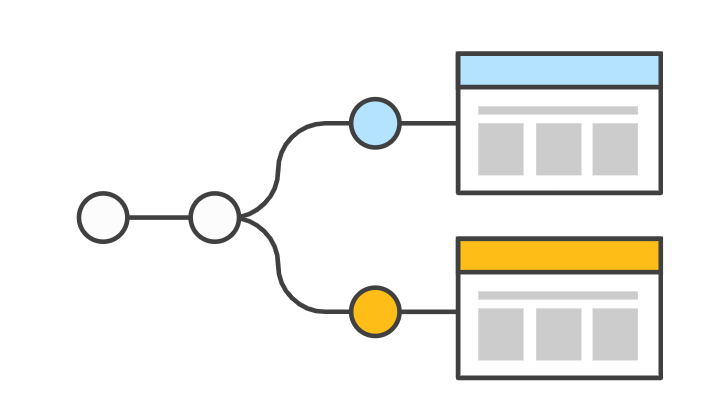
Las ventajas de Git para la gestión de productos son prácticamente las mismas que para el marketing. Las publicaciones más frecuentes se traducen en comentarios más frecuentes de los clientes y actualizaciones más rápidas en respuesta a esos comentarios. En lugar de esperar a la próxima publicación dentro de 8 semanas, puedes enviar una solución a los clientes tan pronto como sus desarrolladores puedan escribir el código.



El flujo de trabajo de la rama de función también proporciona flexibilidad cuando cambian las prioridades. Por ejemplo, si estás a mitad de un ciclo de publicación y quieres posponer una función en lugar de otra en la que el tiempo es crítico, no hay problema. Esa función inicial puede permanecer en su propia rama hasta que el equipo de ingeniería tenga tiempo de volver a ella.

**Git para diseñadores**

Las ramas de función se prestan a la creación rápida de prototipos. Tanto si tus diseñadores de experiencia o interfaz de usuario quieran implementar un flujo de usuario completamente nuevo como si simplemente quieren reemplazar algunos iconos, al extraer una nueva rama conseguirán un entorno de espacio aislado con el que experimentar. De este modo, los diseñadores podrán ver qué aspecto tendrán los cambios en una copia de trabajo real del producto sin temor a que la funcionalidad deje de funcionar.



Quizás la mejor parte de la creación de prototipos con ramas es que es tan fácil fusionar los cambios en la producción como desecharlos. No hay presión por hacer ninguna de las dos cosas. Esto anima a los diseñadores y desarrolladores de interfaces de usuario a experimentar y, al mismo tiempo, garantizar que solo las mejores ideas lleguen al cliente.

**Git para atención al cliente**

Los equipos de atención al cliente y éxito del cliente suelen tener una perspectiva distinta de las actualizaciones que los gestores de productos. Cuando un cliente los llama, por lo general tienen algún tipo de problema. Si ese problema está causado por el software de tu empresa, es necesario enviar una corrección del error lo antes posible.

El ciclo de desarrollo optimizado de Git evita posponer las correcciones de errores hasta la siguiente publicación monolítica. Un desarrollador puede solucionar el problema aplicando un parche directamente en producción. Las correcciones más rápidas equivalen a clientes más satisfechos y menos tickets de asistencia repetidos. En lugar de quedarse atascado con un "Lo siento, nos ocuparemos de esto", el equipo de atención al cliente puede comenzar respondiendo con "¡Ya lo hemos arreglado!".

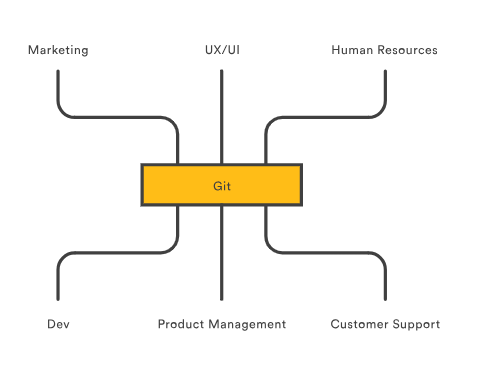
**Git para recursos humanos**

Hasta cierto punto, tu flujo de trabajo de desarrollo de software determina a quién contratas. Siempre ayuda contratar a ingenieros que estén familiarizados con tus tecnologías y flujos de trabajo, pero el uso de Git también proporciona otras ventajas.

Los empleados se sienten atraídos por las empresas que ofrecen oportunidades de crecimiento profesional, y entender cómo aprovechar Git tanto en organizaciones grandes como en pequeñas es una bendición para cualquier programador. Al elegir Git como tu sistema de control de versiones, estás tomando la decisión de atraer a desarrolladores con visión de futuro.

**Git para gestores de presupuestos**

Git se basa en la eficiencia. Para los desarrolladores, lo elimina todo, desde el tiempo perdido pasando confirmaciones a través de una conexión de red hasta las horas de trabajo necesarias para integrar los cambios en un sistema de control de versiones centralizado. Incluso hace un mejor uso de los desarrolladores júnior al brindarles un entorno seguro para trabajar. Todo esto afecta a los resultados de tu departamento de ingeniería.



Ser ágil consiste en descubrir qué funciona lo más rápido posible, incrementar las labores que tienen éxito y eliminar las que no lo tienen. Git sirve como multiplicador de todas tus actividades empresariales y garantiza que cada departamento haga su trabajo con más eficiencia.

**10 COMANDOS DE GIT QUE TODO DESARROLLADOR DEBERÍA SABER**

**1. Git clone**

Git clone es un comando para descargarte el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu ordenador.

git clone <https://link-con-nombre-del-repositorio>

**2. Git Branch**

Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

Creando una nueva rama:

git branch <nombre-de-la-rama>

Este comando creará una rama en local. Para enviar (push) la nueva rama al repositorio remoto, necesitarás usar el siguiente comando:

git push <nombre-remoto> <nombre-rama>

Borrar una rama:

git branch -d <nombre-de-la-rama>

**3. Git checkout**

Este es también uno de los comandos más utilizados en Git. Para trabajar en una rama, primero tienes que cambiarte a ella. Usaremos git checkout principalmente para cambiarte de una rama a otra. También lo podemos usar para chequear archivos y commits.

git checkout <nombre-de-la-rama>

Hay también un comando de acceso directo que te permite crear y cambiarte a esa rama al mismo tiempo:

git checkout -b <nombre-de-tu-rama>

**4. Git status**

El comando de git status nos da toda la información necesaria sobre la rama actual.

git status

Podemos encontrar información como:

* Si la rama actual está actualizada
* Si hay algo para confirmar, enviar o recibir (pull).
* Si hay archivos en preparación (staged), sin preparación(unstaged) o que no están recibiendo seguimiento (untracked)
* Si hay archivos creados, modificados o eliminados

**5. Git add**

Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

Añadir un único archivo:

git add <archivo>

Añadir todo de una vez:

git add -A

Importante: El comando git add no cambia el repositorio y los cambios que no han sido guardados hasta que no utilicemos el comando de confirmación git commit.

6. Git commit

Este sea quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios (quizás después de una tarea o asunto específico).

Git commit es como establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario.

git commit -m "mensaje de confirmación"

Importante: Git commit guarda tus cambios únicamente en local.

7. Git push

Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto.

git push <nombre-remoto> <nombre-de-tu-rama>

De todas formas, si tu rama ha sido creada recientemente, puede que tengas que cargar y subir tu rama con el siguiente comando:

git push --set-upstream <nombre-remoto> <nombre-de-tu-rama>

or

git push -u origin <nombre-de-tu-rama>

Importante: Git push solamente carga los cambios que han sido confirmados.

8. Git pull

El comando git pull se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

git pull <nombre-remoto>

Esta operación puede generar conflictos que tengamos que resolver manualmente.

9. Git revert

A veces, necesitaremos deshacer los cambios que hemos hecho. Hay varias maneras para deshacer nuestros cambios en local y/o en remoto (dependiendo de lo que necesitemos), pero necesitaremos utilizar cuidadosamente estos comandos para evitar borrados no deseados.

Una manera segura para deshacer nuestras commits es utilizar git revert. Para ver nuestro historial de commits, primero necesitamos utilizar el git log -- oneline

Entonces, solo necesitamos especificar el código de comprobación que encontrarás junto al commit que queremos deshacer:

git revert 3321844

10. Git merge

Cuando ya hayas completado el desarrollo de tu proyecto en tu rama y todo funcione correctamente, el último paso es fusionar la rama con su rama padre (dev o master). Esto se hace con el comando git merge.

Git merge básicamente integra las características de tu rama con todos los commits realizados a las ramas dev (o master). Es importante que recuerdes que tienes que estar en esa rama específica que quieres fusionar con tu rama de características.

Por ejemplo, cuando quieres fusionar tu rama de características en la rama dev:

Primero, debes cambiarte a la rama dev:

git checkout dev

Antes de fusionar, debes actualizar tu rama dev local:

git fetch

Por último, puedes fusionar tu rama de características en la rama dev:

git merge <nombre-de-la-rama>

Pista: Asegúrate de que tu rama dev tiene la última versión antes de fusionar otras ramas, si no, te enfrentarás a conflictos u otros problemas no deseados.

LINKS DE REFERENCIA:

* <https://www.atlassian.com/es/git/tutorials/what-is-version-control>
* <https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/>
* <https://git-scm.com/book/es/v2>