

FASTBOOK 02

Cómo trabajar con repositorios en Git

Core Desarrollo



Qualentum Lab

02. Cómo trabajar con repositorios en Git

El objetivo principal de este fastbook es que aprendas a trabajar con repositorios en el cliente de Git. Veremos cómo funciona este, las ventajas que posee y así comprenderás por qué son tan importantes para nuestro trabajo los repositorios de código.

Por lo tanto, iniciaremos el tema explicando qué es un repositorio y cómo históricamente ha mejorado el tema de los versionados. A continuación, exploraremos los posibles repositorios que existen tanto en local como en la nube y en soluciones mixtas; por último, repasaremos las herramientas que facilitan la gestión de repositorios, aunque esta gestión se puede realizar desde la línea de comandos, tal y como originalmente se creó.

Autor: Jaime Morales



Qué es un repositorio



Tipos de repositorios compatibles con Git y un poco de historia



Herramientas para gestionar repositorios



Conclusiones

Qué es un repositorio



“Un repositorio de código es un lugar donde poder almacenar el software de un producto, programa o una aplicación. Esto permite que diferentes desarrolladores puedan colaborar a la vez en la modificación del software, además de permitir tener versionado el software lo que facilita la posibilidad de poder revertir los cambios o quedarse con una parte u otra.”

- The black box lab.

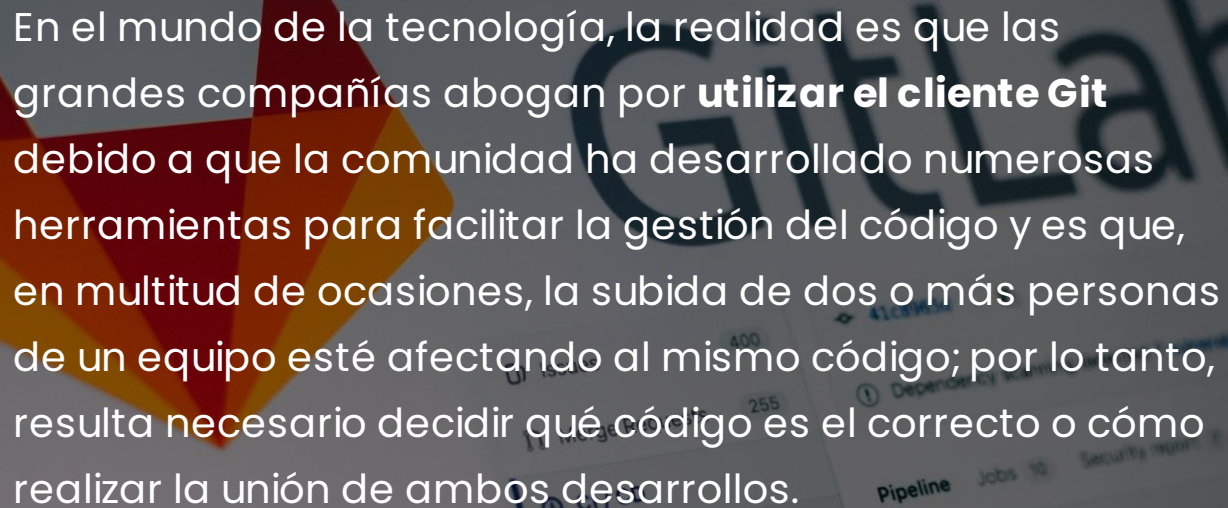
El uso de repositorios de código nos aporta un sinfín de **beneficios y ventajas** a los desarrolladores cuando trabajamos en proyectos, por ejemplo:

- Varias personas pueden trabajar en ese **proyecto de manera colaborativa o en paralelo**.
- Contamos con **la seguridad del código**, como disponer de varios accesos y de copias de seguridad que nos permita recuperar información que se haya podido perder.
- Un equipo puede estar **alineado en cuanto a cambios y pruebas** que se hayan hecho en una nueva funcionalidad desarrollada, gracias a que nos permite añadir comentarios de cada avance o actualización.
- Y algo **muy importante**: también disponemos de un historial de cambios realizados y de la recuperación de versiones anteriores en el caso de que detectemos algún error.

Nos encontramos en el mercado dos relevantes **repositorios de código** o, mejor dicho, dos grandes referentes en este sector: **Git** y **SVN**.

Veamos ahora cuáles son sus características más relevantes:

Git	SVN
Es un software libre y muy potente. Utiliza el control de versiones distribuido.	Dispone de control de versiones centralizado.
Dispone de copias locales del repositorio de código en las que los programadores trabajan directamente.	Se basa en un repositorio central en el que se generan copias de trabajo para los programadores.
Permite el acceso para la totalidad del directorio.	Permite el bloqueo de archivos si es voluntad de su desarrollador. Solo permite el acceso dependiendo del seguimiento de una ruta determinada.
Es muy rápida y permite revisiones de código muy ágiles.	El sistema de creación de ramas y etiquetas es muy eficiente.
El seguimiento de los cambios se basa en el contenido.	El seguimiento de los cambios se basa en archivos.
Cada repositorio, así como cada copia de trabajo individual incluye el historial completo de cambios realizados en la aplicación o programa. Solo precisa la conectividad a la red para la sincronización de cambios.	Solo permite acceder al historial de los cambios en el repositorio completo. Las copias de trabajo solo incluyen la última versión, es decir, la más reciente.
Es un sistema multiplataforma que se puede usar a través de la línea de comando o con múltiples clientes.	Permite la conectividad a la red con cada acceso.
Es compatible con GitHub, GitLab y Microsoft Visual Studio Code.	Se puede usar integrado en Apache.



En el mundo de la tecnología, la realidad es que las grandes compañías abogan por **utilizar el cliente Git** debido a que la comunidad ha desarrollado numerosas herramientas para facilitar la gestión del código y es que, en multitud de ocasiones, la subida de dos o más personas de un equipo esté afectando al mismo código; por lo tanto, resulta necesario decidir qué código es el correcto o cómo realizar la unión de ambos desarrollos.

Así que no es de extrañar que poco a poco, las empresas hermanadas con SVN estén **migrando a Git por la facilidad de integración** y las bondades que ofrece con respecto a su competencia.

Pero ¿qué es Git?

Git es un software de control de versiones diseñado por Linus Torvalds, cuyo objetivo es la eficiencia, compatibilidad del control de versiones de las aplicaciones. Es de licencia libre bajo la licencia pública general de GNU.

Ahora, veamos los **comandos principales** de Git:



git init

Esto crea un subdirectorio nuevo llamado *.git*, el cual contiene todos los archivos necesarios del repositorio, es decir, el 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 -m "<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 *commit*ear.



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 de 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*.

Te recomiendo que los estudies y practiques para mejorar el manejo de la herramienta.

Tipos de repositorios compatibles con Git y un poco de historia



Nos vamos a encontrar con varios tipos de repositorios: la mayoría de ellos son compatibles con el cliente Git. Los más populares son **Gitlab**, **Github** y **Bitbucket** porque poseen características muy similares y ofrecen prácticamente las mismas fortalezas.

1

GitHub

GitHub es una plataforma de desarrollo colaborativo que alojar proyectos utilizando el sistema de control de versiones Git.

Se utiliza principalmente para la **creación de código fuente de programas de ordenador**. El software que opera GitHub fue escrito en [Ruby on Rails](#) y almacena el código de los proyectos alojados generalmente de forma pública.

- El 4 de junio de 2018 Microsoft compró GitHub por la cantidad de 7500 millones de dólares.
- El cambio de propietario despertó preocupaciones y algunos proyectos salieron del entorno GitHub, sin embargo, no tuvieron grandes repercusiones para la compañía ni la plataforma.
- De hecho, continúa siendo la más importante en cuanto a la colaboración en proyectos de código abierto.

En su origen, este producto se publicó como un software completamente libre bajo la licencia MIT.

Sin embargo, tras la división del proyecto en julio de 2013 en dos versiones distintas:

- GitLab CE (*community edition*).
- GitLab EE (*enterprise edition*).

Finalmente, en febrero de 2014 GitLab EE se comenzó a desarrollar bajo una licencia privativa, con características que no están presentes en la versión libre.

En marzo de 2017, se anunció la compra del servicio de mensajería instantánea Gitter por parte de GitLab, sin que esto supusiera la fusión de ambos servicios, sino que continuarían como proyectos totalmente independientes.

Además, la compañía anunció que publicaría el código de la nueva adquisición bajo la licencia MIT antes de junio de 2017.

Pero ¿qué es y para qué sirve GitLab?

Es una plataforma DevOps, o así se define la propia plataforma GitLab, su objetivo es simplificar el desarrollo de software y evitar dar con soluciones puntuales para cada paso del ciclo de vida de la información.

3

Bitbucket

Este servicio de alojamiento basado en web está pensado para los proyectos que utilizan el sistema de control de versiones Mercurial y Git.

Bitbucket ofrece planes comerciales y gratuitos. Proporciona cuentas gratuitas con un número ilimitado de repositorios privados (que incluye hasta cinco usuarios en ellas).

Desde septiembre de 2010, los repositorios privados no se muestran en las páginas de perfil, es decir, si un usuario solo tiene depósitos privados, el sitio web envía el mensaje: "Este usuario no tiene repositorios". Por último, hay que añadir que **el servicio está escrito en Python**.



A grandes rasgos, Bitbucket es similar a GitHub, pero si quieres analizar las diferencias te recomiendo leer [este post](#) de Hostinger.com.

Herramientas para gestionar repositorios



Podemos encontrarnos como desarrolladores con diferentes herramientas que facilitan la gestión de los tipos de repositorios vistos, que evitan el uso de comandos de consola y utilizan una interfaz que se encarga de hacerlos de manera transparente.

Aquí vamos a conocer dos de ellas: **SourceTree** o **GitKraken**.

SourceTree

Esta herramienta **interactúa directamente con los repositorios Git** para que el desarrollador pueda centrarse en el desarrollo, ya que permite gestionar el repositorio a través de una interfaz donde se pueden configurar uno o varios repositorios de código.



Te recomiendo que consultes la web oficial [aquí](#), para que aprendas cómo instalarla, aunque el proceso de instalación no es muy complicado.

De todos modos, y de cara a la configuración, aquí dispones de un par de enlaces que te pueden resultar de ayuda:

Primeros pasos con SourceTree

De adictosaltrabajo.com: <https://www.adictosaltrabajo.com/2015/06/08/primeros-pasos-con-source-tree/>.

Tutorial SourceTree

Por code-fu.net.ni: <https://soporte.code-fu.net.ni/sourcetree-tutorial/>.

GitKraken

GitKraken es una herramienta multiplataforma (para Mac, Windows y Linux) que ayuda a optimizar la gestión del código, mejorando la productividad y evitando el uso de comandos directamente a través de la línea de comandos.

GitKraken fue desarrollado por Axosoft, una **compañía de desarrollo de software en Arizona**, la cual también desarrolló un software para el manejo de proyectos.

**Es una herramienta potente, sencilla y muy funcional
con respecto a la interfaz de usuarios.**

En Medium.com, encontramos a uno de sus adeptos explicando por qué GitKraken para el control de versiones, un [post](#) que te recomendamos leer.

Y en cuanto a la instalación, en la web oficial dispones de todos los detalles, también en su site podrás descargar el software para instalarlo en cualquiera de las [plataformas](#).

Conclusiones



Con este fastbook, hemos querido ofrecerte una visión global sobre cómo manejar y gestionar el control de versiones. Hemos descubierto varias herramientas con sus pros y contras cada una. Lo que sí que es cierto es que **Subversion (SVN) poco a poco se está dejando de utilizar en el mundo de software**, aunque sigue estando muy presente. Lo **más habitual es utilizar el cliente Git** en cualquiera de sus sabores, o bien a través de línea de comandos (es decir, la consola del sistema operativo), o bien mediante interfaces que facilitan la gestión en algunas ocasiones.

La realidad es que, en muchas ocasiones, el uso de la línea de comandos permite realizar **acciones más concretas que las interfaces**, pero para las tareas más habituales, como un *commit* o aceptar una *pull request*, se puede hacer perfectamente a través de cualquiera de las dos herramientas vistas.

¡Enhorabuena! Fastbook superado



[Qualentum.com](https://www.qualentum.com)