TEMA 3

MÓDULO:
HERRAMIENTAS DE BIG DATA

MONITORIZACIÓN Y CONTROL

IGNACIO PÉREZ

Ingenio de Telecomunicaciones por la UAH. Master en Business administration and management por IE Business School. Gerente y arquitecto big data.

Institut de Formació Contínua-IL3
UNIVERSITAT DE BARCELONA

ÍNDICE

OBJETIVOS ESPECÍFICOS

- Entender los conceptos relacionados con los sistemas de control de versiones.
- Conocer los sistemas de control de versiones más utilizados en la actualidad: Git y Subversion (SVN).
- Conocer las plataformas web más utilizadas: GitHub, GitLab y Atlassian BitBucket, así como sus ventajas e inconvenientes.

1. CONTROL DE VERSIONES: GIT O SVN

1.1. INTRODUCCIÓN

Si empezamos buscando cuál es la definición oficial de un sistema de control de versiones, podemos tomarla del glosario de referencia de conceptos tecnológicos de Gartner.

CITA

"In a **distributed version control system (DVCS),** a full copy of the source code and version history can be on each participant's desktop. It is easier to create a new branch or variation, rapidly reflecting an individual's change actions, but this approach shifts network overhead and difference/merge overhead to a synchronized workflow stage. This separation enables different rules to be applied to the ways individuals change code, and to who has the power to merge code changes".

IT Gartner Glossary

Los **sistemas de control de versiones** son una categoría de herramientas de software que permiten a un equipo de desarrollo gestionar los cambios en el código fuente de sus desarrollos a lo largo del tiempo.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código y las almacena en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir atrás en el tiempo y comparar las versiones anteriores del código.

Existen varios sistemas de control de versiones. Los mas importantes son:

- Git.
- Subversion (SVN).
- Mercurial.
- CVS.
- Microsoft Team Foundation Version Control.

Los más utilizados en la actualidad y aquellos para los que vamos a ver más detalle son los dos primeros.

IMPORTANTE

Git es el sistema de control de versiones mas ampliamente utilizado en la actualidad, con mucha diferencia respecto al resto, incluido Subversion (SVN).

1.2. GIT

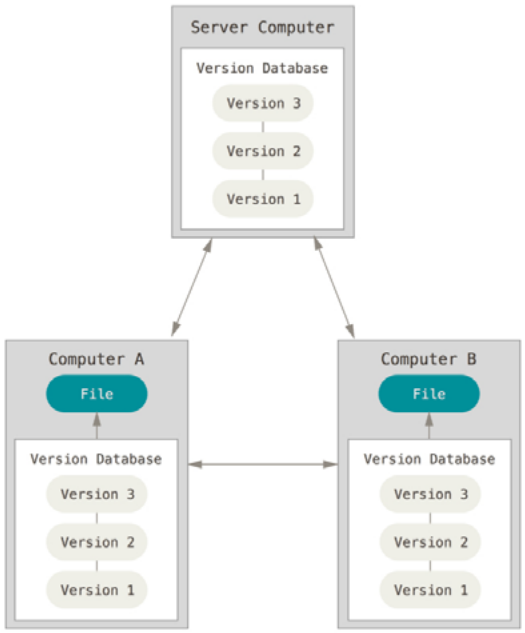
1.2.1 INTRODUCCIÓN

Git es un sistema de control de versiones cuyas principales características son:

- Es software libre proporcionado bajo “GNU general public license”.
- Fue creado, originalmente, por Linus Torvalds, creador del kernel de Linux.
- Cada directorio de trabajo de Git es un repositorio completo con historial y capacidades completas de seguimiento de revisiones.
- No depende del acceso a la red o de un servidor central.
- Es un sistema distribuido:
  - Cada desarrollador tiene una copia de toda la historia de desarrollo. Los cambios se copian de un repositorio a otro repositorio.
  - Los servidores pueden ser replicados y distribuidos en todo el mundo si es necesario.
- Las pruebas de rendimiento han demostrado que Git es mucho más rápido que otros sistemas de control de versiones. En particular, no se ralentiza a medida que crece la historia del proyecto.

Cada desarrollador tiene un “clon” del repositorio central:

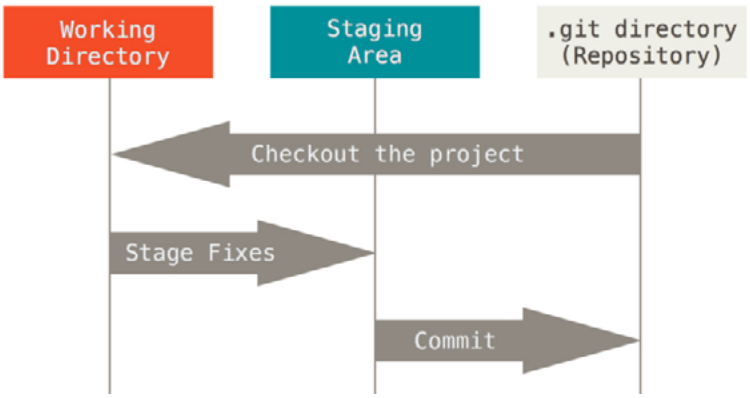
- Se trata de una copia completa del repositorio remoto y de toda su historia.
- Los cambios y “commits” se realizan localmente.
- Los “commits” se “empujan” (Push) al repositorio central para contribuir de nuevo al proyecto.



Fuente: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Los archivos pueden estar en tres estados principales: committed, modified y staged:

- **Committed:** significa que los datos se han almacenado de forma segura en la base de datos local.
- **Modified:** significa que ha cambiado el archivo, pero aún no se ha enviado a la base de datos.
- **Staged:** significa que se ha marcado un archivo modificado en su versión actual para ir en el siguiente "commit".



Fuente: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Las ventajas de Git sobre SVN son las siguientes:

- **Sistema Distribuido:** el hecho de ser un sistema distribuido significa que múltiples repositorios redundantes y ramas son conceptos de primera clase de la herramienta

  En un sistema distribuido como Git, cada usuario tiene una copia completa guardada del proyecto, haciendo que el acceso a la historia sea extremadamente rápido.

  Algo muy importante es que, debido a dicha característica, podemos utilizarlo sin conexión a Internet.

  En SVN, por otro lado, solamente el repositorio central contiene la historia completa, por lo cual, los usuarios deben comunicarse a través de la red al repositorio central para obtener el historial de los archivos.

- **Manejo de Ramas:** en Git utilizar ramas es muy común y fácil, mientras que en SVN es un proceso un poco más engorroso y no tan habitual. De hecho, la queja más común sobre SVN es lo complicado que es trabajar con ramas.
- **Rendimiento:** dado que Git trabaja con un repositorio local, no hay latencia en la mayoría de las operaciones, exceptuando push y pull, en las cuales sí necesitamos conectarnos al repositorio central.

Las desventajas de Git sobre SVN son las siguientes:

- **Complejidad:** es más complejo que los sistemas centralizados tradicionales porque entran en juego más repositorios, más operaciones y más posibilidades para trabajar en equipo.
- **Curva de aprendizaje:** es más alta. Lo básico lo podemos aprender enseguida, pero la realidad nos demuestra que no suele ser suficiente.
- Los **comandos** y algunos **conceptos** que usa pueden llegar a ser **confusos.**

1.2.2. COMANDOS PRINCIPALES

1.2.2.1. CLONE

Normalmente se hace una vez, cuando recuperamos un repositorio remoto por primera vez.

Descargamos todo el repositorio remoto localmente (almacenándolo como un remoto generalmente llamado “origin”) en un directorio con nombre. git.

Lo podemos hacer tantas veces como deseemos, si queremos tener varias copias de un repositorio localmente.

1.2.2.2. CHECKOUT

Con este comando indicamos con qué rama queremos trabajar en el repositorio local.

Vamos a tener varias ramas locales en nuestro repositorio local que tendrán su correspondencia con varias ramas remotas en el repositorio remoto.

El concepto y uso de las ramas se explica más en detalle en el siguiente apartado.

1.2.2.3. ADD

Con este comando, añadimos nuestros cambios para indicarle a Git que nos gustaría que se incluyeran en el próximo “commit”.

1.2.2.4. COMMIT

Este comando le indica a Git que confirme todos nuestros cambios de código en un “commit”.

Un “commit” registra quién hizo los cambios, cuándo y tiene asociado un mensaje.

1.2.2.5. PULL

Con este comando, mantenemos el estado actual de nuestro repositorio local, recuperando cambios del servidor remoto y aplicándolos a nuestras ramas locales.

1.2.2.6. PUSH

Este comando lleva nuestros “commits” desde nuestras ramas locales a las ramas remotas, permitiendo que otros desarrolladores vean nuestros cambios en el código.

PARA SABER MÁS

Para saber mas sobre Git y tener un listado completo de todos los comandos que puedes utilizar, accede a los siguientes enlaces:

- Tutoriales de Git.
- Comandos de Git.
- Git Cheat Sheet.

1.2.3. USO DE RAMAS

Una de las características más potentes y sencillas de utilizar en Git son las ramas.

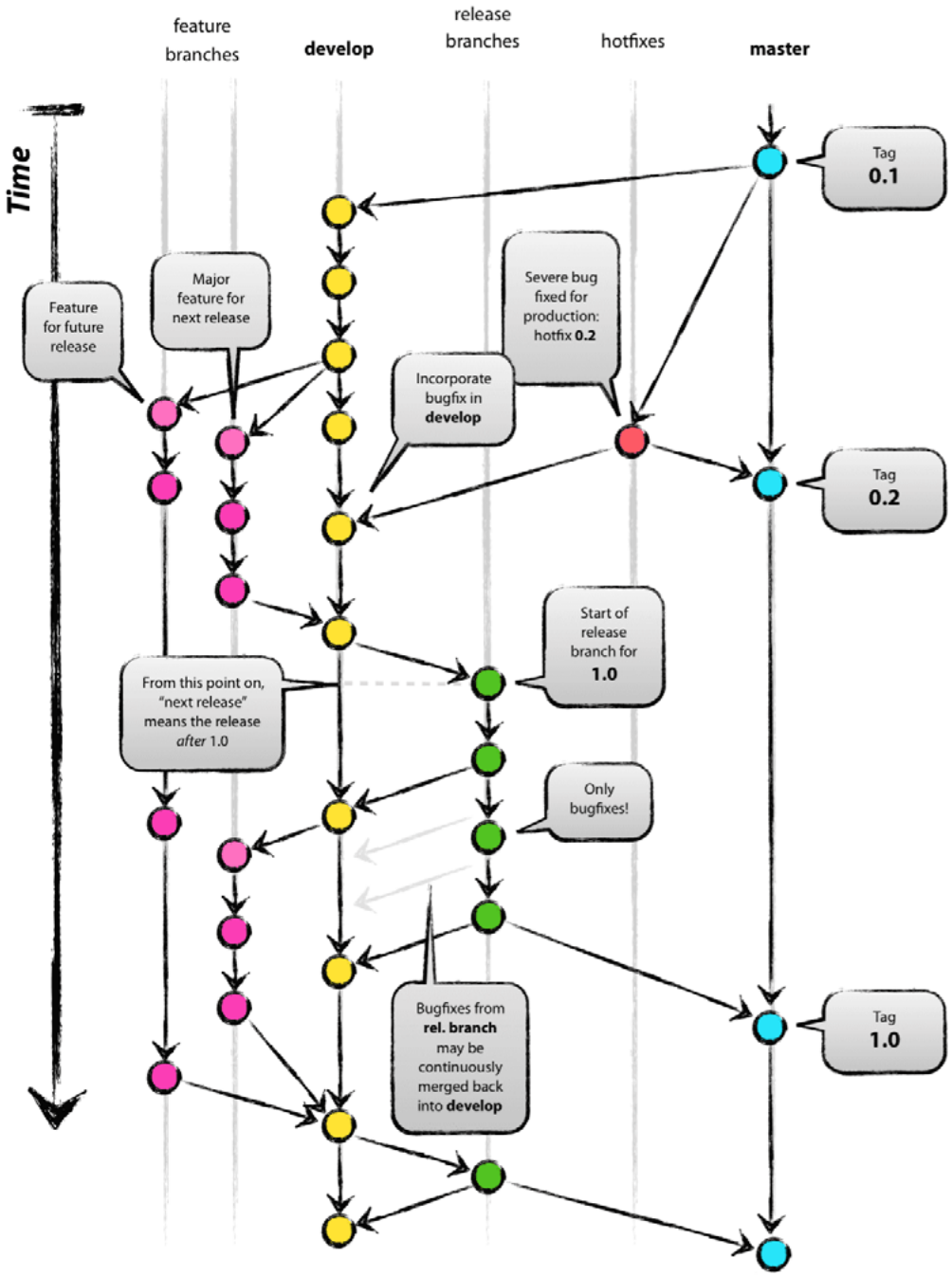
Permiten bifurcar y aislar el desarrollo del software en distintas ramas de trabajo y, posteriormente, hacer la mezcla entre ellas (“merge”).

IMPORTANTE

Una buena estrategia de creación de ramas es muy importante para conseguir el éxito en un proyecto.

A continuación, se detalla la estrategia más utilizada en el desarrollo de software:

- El modelo de ramas se basa en el principio de que hay dos ramas permanentes en un repositorio: “Develop” y “Master”.
- La rama “Develop” captura todo el trabajo de desarrollo y la rama “Master” mantiene el estado actual.
- Este modelo de ramas, también, implica el uso de ramas de soporte temporales que representan las ramas “Feature”, “Release” y “Hotfix”.



Fuente: https://nvie.com/posts/a-successful-git-branching-model/

PARA SABER MÁS

Para ampliar información sobre buenas prácticas en el uso de las ramas, puedes consultar el siguiente enlace:

A successful Git branching model.

1.3. SUBVERSION (SVN)

1.3.1 INTRODUCCIÓN

Apache Subversion, que a menudo se abrevia como SVN, es un sistema de control de versiones distribuido bajo una licencia de código abierto.

Subversion fue creado por CollabNet Inc. en 2000 pero, ahora, se desarrolla como un proyecto de Apache Software Foundation y, como tal, es parte de una rica comunidad de desarrolladores y usuarios.

SVN surgió con la intención de sustituir y mejorar a CVS (Concurrent Versions System), uno de los primeros sistemas de control de versiones. CVS, pese a sus limitaciones, constituyó el estándar de facto de los sistemas de gestión de versiones en el ámbito del software y, además, mantiene las ideas fundamentales de CVS, pero suple sus carencias y evita sus errores.

Las principales características de SVN y sus mejoras frente a CVS son:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- Atomicidad de las actualizaciones: una lista de cambios constituye una única transacción o actualización del repositorio. Esta característica minimiza el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
- Mayor eficiencia en la creación de ramas y etiquetas que en CVS.

1.3.2. COMANDOS PRINCIPALES

1.3.2.1. CHECKOUT

El comando Checkout se usa para descargar nuestro código desde el repositorio remoto en el servidor SVN a nuestra “working copy”.

Si deseamos acceder a los archivos del servidor SVN, este comando es la primera operación que debemos realizar.

Crea la “working copy” en nuestro equipo local, desde donde podemos editar, eliminar o agregar contenidos.

1.3.2.2. COMMIT

Siempre que hagamos cambios en nuestra “working copy”, estos no se reflejarán en el servidor SVN.

Para que los cambios sean permanentes, debemos realizar un “commit”.

1.3.2.3. UPDATE

Este comando actualiza nuestra “working copy” con los cambios del repositorio remoto. De esta forma, podremos tener los cambios realizados por otros miembros del equipo de desarrollo.

PARA SABER MÁS

Para saber mas sobre SVN y tener un listado completo de todos los comandos que puedes utilizar, accede a los siguientes enlaces:

- Qué es Subversion.
- Subversion Cheat Sheet.

2. VENTAJAS E INCONVENIENTES DE USAR GITHUB

Github es una plataforma web para alojar repositorios de código que usa el sistema de control de versiones Git.

Fue comprada por Microsoft en junio del 2018.

La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas y que, como usuarios, no sólo podamos descargarnos la aplicación, sino también entrar a su perfil para leer sobre ella y colaborar con su desarrollo.

Las ventajas de usar GitHub es que nos permite:

- **Versionar:** GitHub está basado en Git, por lo que podemos guardar los cambios de nuestro código en un repositorio remoto, con la posibilidad de tener acceso al historial de cambios, ya sea para volver a una de las versiones o para hacer comparaciones entre ellas.

  Lo único que necesitamos es instalar Git en nuestro equipo local, crear una cuenta en Github y ejecutar los comandos adecuados.

  La visibilidad de los repositorios puede ser pública o privada.

- **Trabajar en equipo:** GitHub permite que nuestro repositorio tenga una visibilidad a nivel global.

Podemos dar acceso a quien consideremos para trabajar sobre él.

- **Aprender:** muchos desarrolladores tienen sus repositorios públicos en GitHub. De esta forma, podemos acceder a su código y aprender de él.

  Incluso nos permite hacer modificaciones y experimentar sin afectar el código original. A esto último se le conoce como hacer un fork.

- **Contribuir:** si después de que hayamos copiado un proyecto (hacer fork) hacemos ajustes que arreglan errores o introducen nuevas funcionalidades, podemos proponerle al dueño del repositorio que integre nuestros cambios en su código. Esto se hace enviando un pull request con todas nuestras modificaciones o novedades. El dueño del repositorio podrá aceptar nuestros cambios (hacer merge) o rechazarlos.

- **Mostrar conocimientos:** GitHub es un escaparte perfecto para publicar nuestros desarrollados y mostrar nuestras habilidades.

  Los empleadores o clientes que estén buscando ciertos perfiles como el nuestro, pueden acceder a GitHub en su busca.

Como desventaja, antes teníamos un limite de colaboradores en los repositorios privados. Pero, está limitación en la actualidad ya no existe.

Si queremos utilizar GitHub a nivel empresarial, debemos optar por la versión de pago “Enterprise”.

PARA SABER MÁS

Para saber más sobre GitHub y sus distintos planes, accede a los siguientes enlaces:

- GitHub.
- Planes y Precios.

3. ALTERNATIVAS A GITHUB

Las principales alternativas a GitHub en la actualidad son:

- GitLab.
- Atlassian Bitbucket.

3.1. GITLAB

GitLab es una plataforma DevOps completa, entregada como una sola aplicación que hace todo: pla-

nificación del proyecto, administración del código fuente, CI/CD (Continuous Integration / Continuous Delivery), monitorización y seguridad.

Al ofrecer una sola aplicación, se acortan los tiempos de los ciclos de entrega del software y se aumenta la productividad.

3.2 ATLASSIAN BITBUCKET

Bitbucket es un servicio de alojamiento web para código fuente y proyectos de desarrollo que utilizan sistemas de control de revisión Mercurial (desde su lanzamiento) o Git (desde octubre de 2011) propiedad de Atlassian.

Bitbucket ofrece planes comerciales y cuentas gratuitas.

La ventaja de Bitbucket es su fácil integración con el testo de software de Atlassian como JIRA Software, HipChat, Confluence y Bamboo.

IDEAS CLAVE

- Los sistemas de control de versiones son una pieza fundamental para los equipos de desarrollo.
- Git es el sistema de control de versiones mas utilizado en la actualidad, por lo que su conocimiento en casi obligatorio para un desarrollador.
- Las plataformas web como GitHub, GitLab y Atlassian BitBucket son ampliamente utilizadas en las empresas y forman parte de un conjunto de herramientas que facilitan el desarrollo y entrega continua de software (CI/CD – Continuous Integration / Continuous Delivery).