**PERÍODO ACADÉMICO:**2019-B

**ASIGNATURA:** Aplicaciones en Ambientes Libres

**PROFESOR:** Ing. Julián Galindo

**Práctica No.:** 3

**INTEGRANTES:**

* Jersson Andrango
* Grace Borja
* Jefferson Collantes
* David Cruz
* Jair Quiñónez
* David Yánez

**FECHA:** 2 de diciembre de 2019

**TEMA: USO DE GIT**

**Objetivos:**

-Conocer el funcionamiento de sistema de control de versiones

-Aprender el flujo de trabajo de git

-Ejercicio práctico

**MARCO TEÓRICO**

1. ***Principales sistemas de control de versiones***

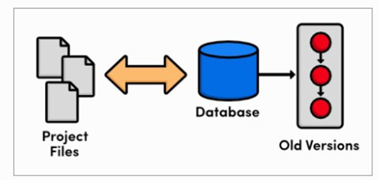
El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que sea posible recuperar versiones específicas más adelante.

Los sistemas de control de versiones han ido evolucionando a lo largo del tiempo y podemos clasificarlos en tres tipos:

**Sistemas de Control de Versiones Locales**

Los sistemas de control de versiones locales como se muestra en la Figura 1 en vez de mantener las versiones como archivos independientes, los almacenaban en una base de datos. Cuando era necesario revisar una versión anterior del proyecto se usaba el sistema de control de versiones en vez de acceder directamente al archivo, de esta manera en cualquier momento solo se tenía una copia del proyecto, eliminando la posibilidad de confundir o eliminar versiones.

En este punto el control de versiones se llevaba a cabo en el computador de cada uno de los desarrolladores y no existía una manera eficiente de compartir el código entre ellos. [1]



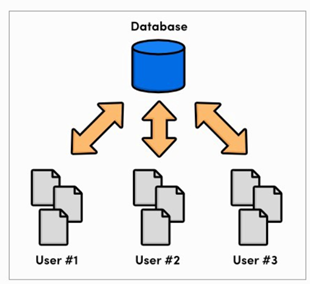
*Figura 1: Sistemas de Control de Versiones Locales*

**Sistemas de Control de Versiones Centralizados**

Para facilitar la colaboración de múltiples desarrolladores en un solo proyecto los sistemas de control de versiones evolucionaron: en vez de almacenar los cambios y versiones en el disco duro de los desarrolladores, estos se almacenaban en un servidor como se muestra en la Figura 2.

Pero existía el problema de que si dos personas editaban el mismo archivo y se presentaba un conflicto alguien debía solucionar este problema de manera manual y el desarrollo no podía continuar hasta que todos los conflictos fueran resueltos y puestos a disposición del resto del equipo.

Esta solución funcionó en proyectos que tenían relativamente pocas actualizaciones y por ende pocos conflictos, pero resultó muy engorroso para proyectos con docenas de contribuyentes activos que realizan actualizaciones a diario. [1]

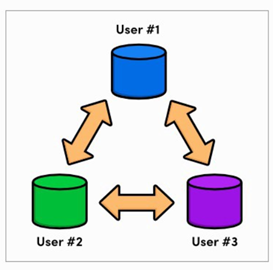


*Figura 2: Sistemas de Control de Versiones Centralizados*

**Sistemas de Control de Versiones Distribuidos**

La siguiente generación de sistemas de control de versiones se alejó de la idea de un solo repositorio centralizado y optó por darle a cada desarrollador una copia local de todo el proyecto como se muestra en la Figura 3, de esta manera se construyó una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada, pero teniendo un mecanismo de resolución de conflictos mucho más elegante que una su versión anterior.

Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclar los conflictos que se presenten sólo cuando se requiera. Cómo cada usuario tiene una copia completa del proyecto el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de pérdida de datos es mucho menor que en cualquiera de sus predecesores. [1]



*Figura 3: Sistemas de Control de Versiones Distribuidos*

1. Definición de:
   1. ***Git***

Git, es un software de control de versiones diseñado por Linus Torvalds.

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente, es decir Git nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún software o página que implique código el cual necesitemos hacerlo con un grupo de personas. [2]

Algunas de las características más importantes de Git son:

* Rapidez en la gestión de ramas, debido a que Git nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
* Gestión distribuida; Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
* Gestión eficiente de proyectos grandes.
* Realmacenamiento periódico en paquetes.
  1. ***Buddy***

Es un software de integración y entrega continúa basado en web y autohospedado para desarrolladores de Git que se puede usar para construir, probar e implementar sitios web y aplicaciones con código de GitHub, Bitbucket y GitLab. Emplea contenedores Docker con lenguajes y marcos preinstalados para compilaciones, junto con DevOps, acciones de monitoreo y notificación. [3]

Algunas de las características que ofrece Buddy son:

o Implementaciones automáticas en push to branch

o Construcciones y pruebas basadas en Docker

o Configuración de 10 minutos del entorno completo

* 1. ***Bitbucket***

Bitbucket se sitúa como una de las principales alternativas al conocido y utilizado GitHub. Bitbucket permite a todos los usuarios gratuitos elegir entre utilizar Git o Mercurial como controlador de versiones a la vez que les ofrece una serie de herramientas para el control de errores (como Jira), integración con Google Analytics, HipChat como software de comunicación entre colaboradores y opciones de control de actividad de cada repositorio en cualquier momento para que el desarrollador no tenga que utilizar herramientas externas para esto. [4]

Una de las principales ventajas de Bitbucket de cara a GitHub es que este dispone de un número ilimitado de repositorios privados para los usuarios gratuitos de manera que cualquier desarrollador podrá subir a su plataforma sus códigos, establecerlos como privados y no tener que pagar por ello. A cambio de ello, las versiones gratuitas de Bitbucket limitan a tan solo 5 colaboradores en cada proyecto y los usuarios que necesiten un mayor número de estos tendrán que pasarse a un plan de suscripción. [4]

En GitHub, al contrario, los usuarios disponen de un número ilimitado de colaboradores en su versión gratuita y por lo que pagan los usuarios es por el número de repositorios privados que necesiten.

Bitbucket dispone de un cliente oficial para Windows y para Mac, quedando Linux fuera del soporte oficial de esta plataforma. También su plataforma es totalmente accesible desde la web e incluso compatible con el cliente oficial de Git. [4]

* 1. ***Source forge***

SourceForge es un página web enfocada en el trabajo colaborativo de proyectos software. Es una herramienta muy versátil puesto que posee una gran cantidad de aplicaciones de software libre útiles para el desarrollo de software.

SourceForge actúa como un repositorio pero su rol principal es de central de desarrollo de software que ayuda con la gestión de varios proyectos de software libre.

* 1. ***Gogs***

Es la abreviatura de Go Git Service, y es una plataforma basada en git escrita en Go, que funciona como una forja para el desarrollo colaborativo de software. Gogs esta realizado en código libre, con licencia del MIT, y se caracteriza por ser ligero siendo compatible con ARM.

Gogs permite gestionar repositorios Git, tanto a sus usuario y los accesos de los mismos. Permite desarrollar repositorios privados en los cuales se puede dar acceso a cierto grupo de colaboradores.

Una funcionalidad muy atractiva de Gogs es la de efectuar exámenes de código. Su manejo de solicitudes de fusión mejora enormemente el trabajo colaborativo y las distintas versiones generadas.

* 1. ***Github***

Es una plataforma de desarrollo colaborativo que permite almacenar proyectos o cualquier tipo de documento mediante el sistema de manejo de versiones de Git. Es muy utilizado en el ámbito de la programación puesto que la mayoría de los documentos albergados en repositorios contiene código fuente.

Los proyectos alojados en el repositorio se almacenan de forma pública para que cualquier usuario pueda descargar el código y utilizarlo de la manera que sea conveniente. Se puede obtener repositorios privados con la opción pagada de Github.

Una de las características especiales de Github es su visualización de graficas sobre el avance del proyecto y la cantidad de operaciones Git que fueron realizadas, así como los usuarios que han aportado de mejor manera en el proyecto.

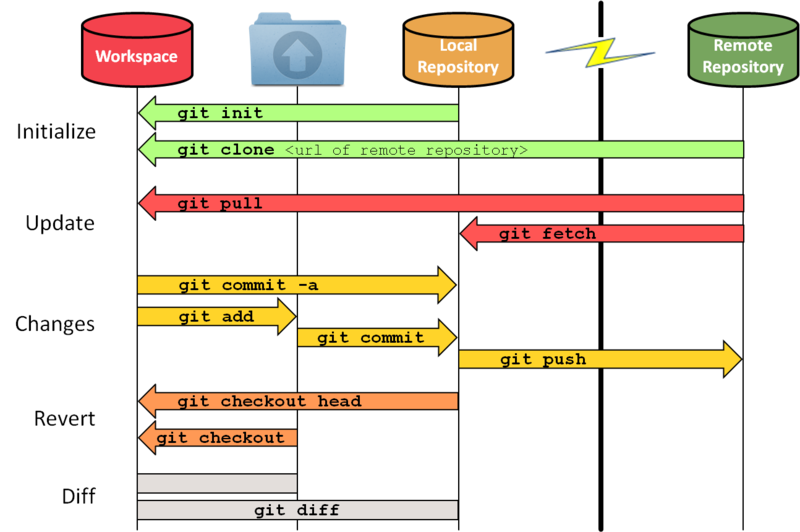
* 1. ***GitLab***

GitLab es un servicio web para el manejo de versiones y desarrollo de software orientado en el desarrollo colaborativo del repositorio de Git. La característica principal es su sistema de seguimiento de errores que se encuentra publicada bajo una licencia de código abierto.

El desarrollo de GitLab a sido financiado por sus usuarios. Lo que permite una correcta interacción con el modelo de negocio open core de la empresa, debido a ello es que muchas características de GitLab se incluyen para la versión de pago.

1. Operaciones en un repositorio: commit, branch, pull, push y merge

El flujo de trabajo es el que se muestra en la figura de la parte inferior



Commit: permite pasar todos los archivos que están el directorio de trabajo al repositorio local. Para poder hacer un comentario se usa –m, es aconsejable, utilizar un mensaje que referencie el cambio realizado.

Fetch: El comando git fetch <repositorio> permite recuperar todos los ficheros de un repositorio remoto que hayan sido modificado por otros colaboradores. Esta información es ubicada en una rama oculta del repositorio por lo que no se fusionará automáticamente.

Merge:Permite fusionar ramas, en el caso de utilizarlo con fetch\_head fusiona la rama oculta con el directorio actual. Para fusionar otra rama a la rama activa se utiliza merge <branch>.

Pull: Permite actualizar el repositorio local al commit más nuevo. Este es un comando que hace un atajo de los comandos git fetch y git merge por lo que permite ahorrar tiempo, sincroniza y trae los cambios del repositorio remoto a la rama en al cual se está trabajando. Cuando se trabaja en grandes proyectos es aconsejable primero comprobar los cambios.

Branch: permite crear ramas para desarrollar funcionalidades aisladas, una vez realizado el trabajo en una rama, se la debe fusionar con la rama principal

* Para crear una nueva rama y cambiar el puntero a la misma: git checkout -b rama\_x
* Para volver a la rama principal git checkout master
* Para borrar una rama git branch -d rama\_x
* Para que la rama esté disponible en el repositorio remoto: git push origin <branch>

Push: Permite enviar los commits realizados a la rama remota. Tiene 2 argumentos:

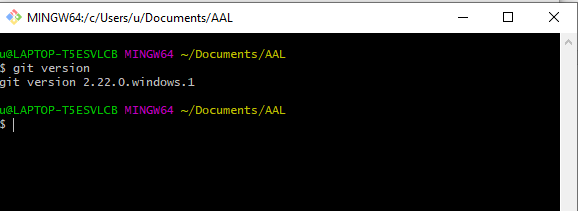
1. Nombre del repositorio remoto.
2. El nombre de la rama del repositorio local

Si no se indica argumentos se toma por defecto la rama sobre la cual se está trabajando.

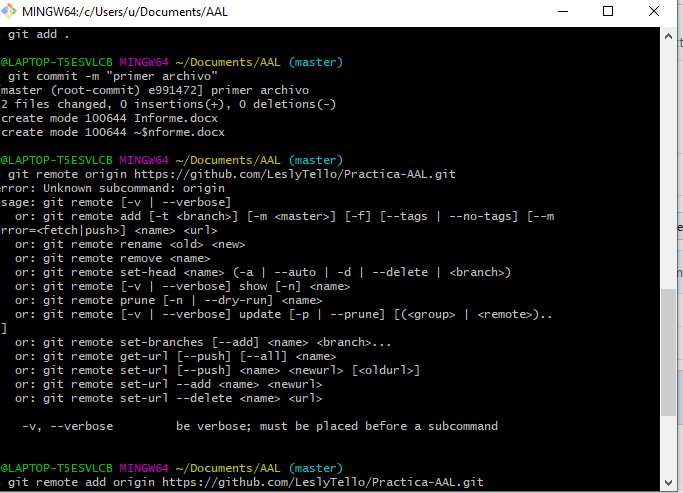
En caso de dificultad en el proceso, el problema es la falta de sincronización con el repositorio remoto para solucionarlo, es necesario hacer un git pull.

También es posible forzar el envío, aunque los repositorios no estén sincronizados, por medio del comando git push –forcé, sin embargo, no es aconsejable en el caso de trabajo colaborativo porque es probable que se sobre-escriba o elimine cualquier cambio realizado por los demás integrantes del grupo.

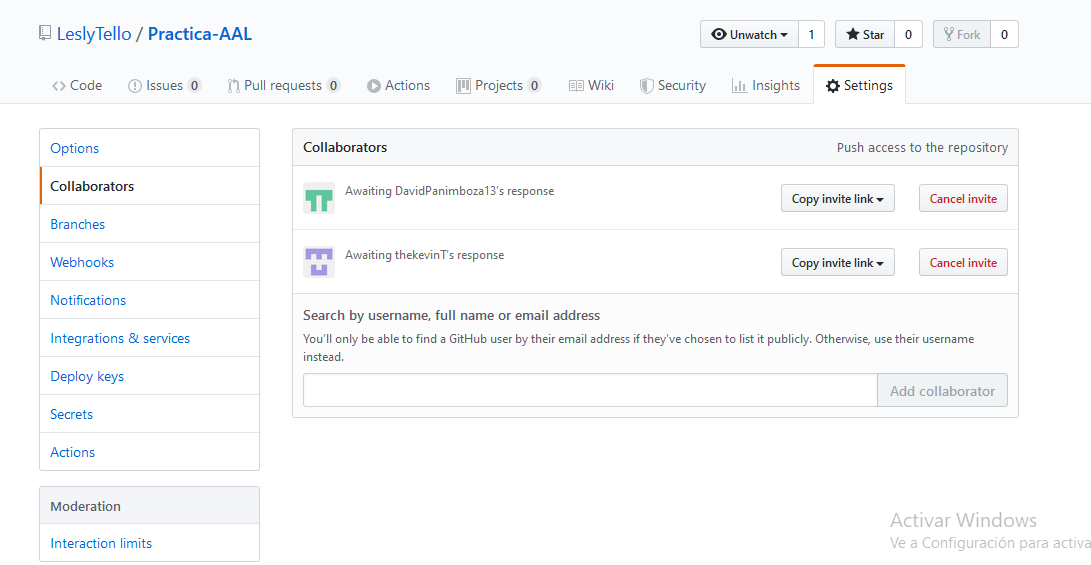
**DESARROLLO DE LA PRÁCTICA**

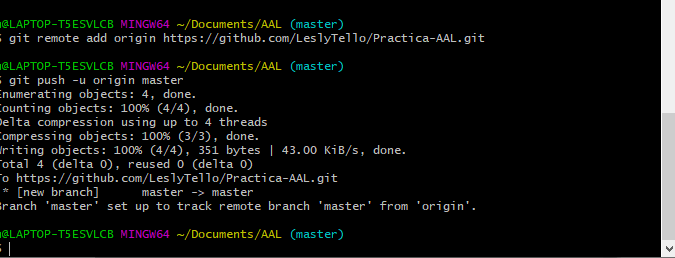
Determinar la versión de Git

Añadir todos los archivos y hacer commit con un mensaje

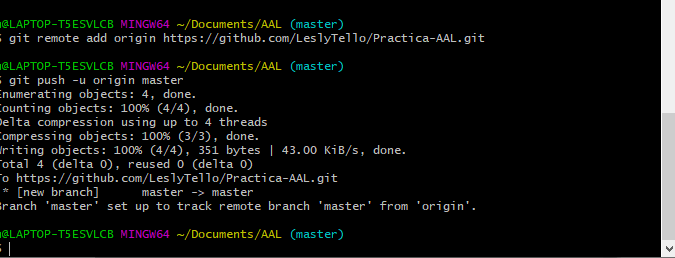


Asociar con la cuenta de github creada





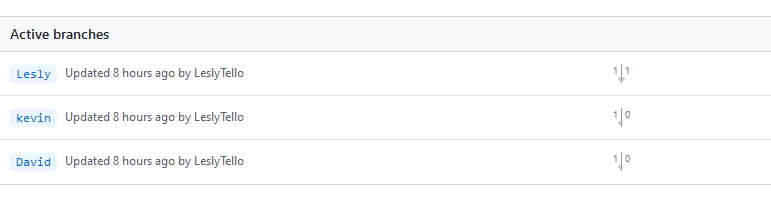
Añadir al repositorio remoto



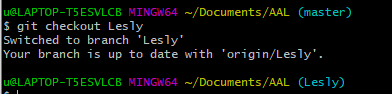
Crear las ramas

Git branch nombreRama

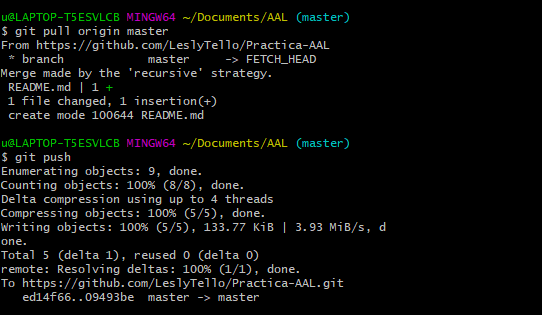
Git push –u origin nombreRama



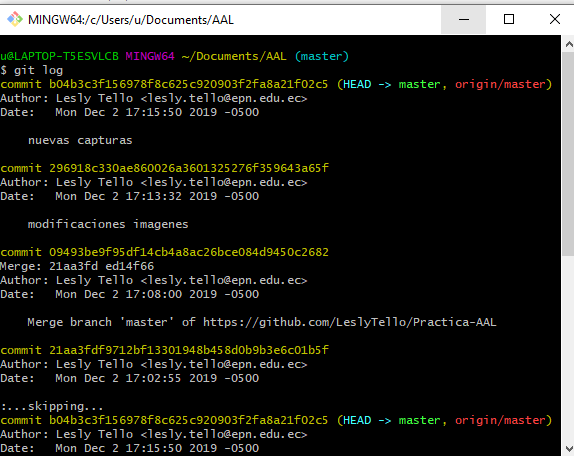
Para poder cambiar de rama se usa git checkout nombre Rama



Realizar actualizaciones



Visualizar el log de los commits realizados

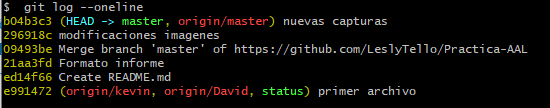


El valor hash en amarillo representa el identificador de cada commit, y es útil para regresar a una versión anterior.

Para poder recuperar a una versión anterior del código se utiliza

git checkout HEAD prueba.txt

La misma que permite regresar en un commit, pero si se desea regresar a una versión más antigua es necesario conocer el valor del commit



Y el comando a utilizar es git checkout hash nombreArchivo. Ejm: git checkout 21aa3fd informe.docx

**CONCLUSIONES**

* Es decisión del usuario decidir qué sistema de control de versiones o la plataforma utilizar, esto dependerá de sus gustos y necesidades teniendo en cuenta si el número de colaboradores será elevado o, por el contrario, se prefiere disponer de repositorios privados en vez de colaboradores de un proyecto.
* El desarrollo de software no es una tarea que se realice por una sola persona, sino, por un equipo de trabajo y por ello Git-GitHub cobra mucha importancia ya que permite realizar un trabajo colaborativo organizado y eficaz. Así, varias personas pueden estar trabajando en el código al mismo tiempo y obtener un programa funcional.
* La rama maestra es creada por defecto y el flujo dentro de git ha sido previamente establecido, por lo que hacer saltos indebidos entre áreas llevará a conflictos.
* El desarrollo ágil de software mediante grupos de trabajo se ha visto impulsado por sistemas de gestión de versiones. Lo importante de tener un repositorio común es que toda la información necesario se encuentra en el mismo, facilitando enormemente la tarea de refactorizar código.

**BIBLIOGRAFÍA**

[1] Joint, D. (28 enero 2017). “Sistemas de Control de Versiones, qué son y por qué amarlos”. [En línea] Recuperado el: 02 dic 2019. Disponible en: https://medium.com/@jointdeveloper/sistemas-de-control-de-versiones-qué-son-y-por-qué-amarlos-24b6957e716e

[2] Andrés. (16 agosto 2015). “Qué Es Git”. [En línea] Recuperado el: 02 dic 2019. Disponible en: https://codigofacilito.com/articulos/que-es-git

[3] Stackshare. “Buddy vs GitLab: ¿Cuáles son las diferencias?”. [En línea] Recuperado el: 02 dic 2019. Disponible en: https://stackshare.io/stackups/buddy-vs-gitlab

[4] Velasco, R. “Bitbucket, una alternativa al conocido controlador de versiones GitHub” [En línea] Recuperado el: 02 dic 2019. Disponible en: <https://www.redeszone.net/2014/05/27/bitbucket-una-alternativa-al-conocido-controlador-de-versiones-github/>

[5] Red Code. (2019) Git Push, Enviando Commits a Repositorios Remotos. Extraído el 2 de diciembre de 2019 de <https://blog.artegrafico.net/git-push-enviando-commits-a-repositorios-remotos>

[6]G. Padak, «The GitHub Blog,» 2 Diciembre 2019. [En línea]. Available: https://github.blog/2019-11-05-internal-repositories-are-now-generally-available-for-github-enterprise/. [Último acceso: 2 Diciembre 2019].

[7] J. Elder, «GitLab Blog,» 2 de Diciembre 2019. [En línea]. Available: https://about.gitlab.com/blog/2019/12/02/gitlab-iconography-where-mvc-meets-visual-design/. [Último acceso: 2 Diciembre 2019].

[8] Community Team, «SourceForge Blog,» 2 de Diciembre 2019. [En línea]. Available: https://sourceforge.net/blog/facing-future-open-source/. [Último acceso: 2 Diciembre 2019].