CURSO DE GIT-GITHUB

¿Para qué sirve Git?

* Permite al desarrollador alterar el código tanto en tiempo real (online) como en un dispositivo local (offline).
* Varios desarrolladores pueden trabajar en el proyecto, teniendo cada uno su respectiva copia.
* Permite realizar modificaciones y notificar al resto del equipo los cambios realizados.
* Permite descargar las versiones más recientes del software que se está creando.
* Permite controlar las diferentes versiones de código que puedan existir para obtener la más reciente.

Repositorios distribuidos: Tener la copia de un proyecto o repositorio en una computadora local, que brinde la oportunidad de trabajar sobre ella y enviar los cambios a otros repositorios.

--------------------------------------

Afirmaciones sobre Git:

1. Nos deja organizar el trabajo en equipo, manteniendo los cambios en los archivos en un servidor específico para eso: Git nos permite almacenar los cambios realizados en cada archivo en un servidor propio para eso. Toda la gestión de cambios la realiza Git y solo debemos preocuparnos por crear un código que funcione, y no por quién lo cambió antes.

2. Permite almacenamiento y acceso a un historial de cambios: Cada cambio que realices se registra en un historial y se puede ver y restaurar en cualquier momento.

COMANDOS GIT

Comando git init: Inicializa un repositorio en el directorio donde se ejecuta el comando. Desde este comando, Git podrá administrar los cambios realizados en los archivos.

Comando git status: Permite revisar el estado actual del repositorio, si fueron agregados nuevos archivos, si hay archivos pendientes por cargar, si fueron eliminados elementos, entre otras alertas.

Comando git config --local (user.email - user.name): Permite configurar el nombre y el correo electrónico del autor de forma local, o sea, en la computadora donde tiene clonado el proyecto. Para rectificar si ambos datos son correctos se usan los comandos: git config user.email // git config user.name

Comando git config --global (user.email - user.name): Antes de cualquier interacción con git, es necesario informar quién eres para que almacene correctamente los datos del autor de cada uno de los cambios en el código. Permite agregar el correo electrónico y el nombre del dueño del repositorio actual. Esto será necesario realizarlo siempre que se utilice por primera vez Git.

Comando git commit -m "Mensaje...": Permite agregar un mensaje o comentario sobre los cambios realizados en el elemento que se está trabajando. IMPORTANTE: NO REALIZAR UN COMMIT DE UN CÓDIGO QUE NO ESTÉ FUNCIONANDO. EN CADA ALTERACIÓN SIGNIFICATIVA SE TIENE QUE GENERAR UN COMMIT.

Comando git add . / "Nombre del archivo": Prepara los elementos a cargar dentro del repositorio para después ser enviados.

Comando git log: Permite ver el histórico de modificaciones dentro del proyecto seleccionado. Se pueden observar datos como el HASH del proyecto, el branch o rama de ese proyecto, el autor de la modificación, la fecha de la modificación y el mensaje que se dejó para esa actualización.

Comando git log --oneline: Brinda una información resumida sobre las modificaciones realizadas. Se tienen los primeros siete digitos del HASH, el branch en donde está alojada, y el commit creado.

Comando git log -p: Brinda toda la información disponible sobre las modificaciones hechas en un proyecto elegido, esto gracias a un diff que logra mostrar cada uno de los cambios generados por la consola de Git.

Comando clear: Limpia la pantalla de toda la información que pueda aparecer en la consola Git.

Comando mkdir "nombre nuevo archivo": Permite crear un nuevo archivo dentro del directorio que se encuentre abierto actualmente.

Comando git init --bare: La palabra "bare", indica que este repositorio es puro. O sea, solo contiene las modificaciones de los archivos. No tiene una copia de cada uno de los archivos físicos de forma que pueda acceder fácilmente a ellos, solo controla y trae unas facilidades que hacen que podamos acceder a ese repositorio como remoto. Con este comando creamos un repositorio que no tendrá la working tree, o sea, no contendrá una copia de nuestros archivos. Como el repositorio servirá solo como servidor, para que otros miembros del equipo sincronicen sus trabajos, disminuimos espacio de almacenamiento de esta forma.

Comando git remote: Lo que está haciendo es listar todos los servidores remotos que nuestro repositorio conoce.

git remote add nombre-repositorio camino/del/repositorio: De esta forma tendremos un link de nuestro repositorio local con el repositorio remoto, que llamamos de nombre-repositorio, que está almacenado en camino/del/repositorio.

Comando git remote -v: Aparece la dirección completa del servidor, un elemento fetch y push.

Comando git clone: Permite obtener los datos de un repositorio externo dentro de un repositorio local o propio.

Comando cd ../..: Permite salir de dos carpetas a la vez.

Comando git push: te permite subir los commits desde tu rama (branch) local en tu repositorio git local al repositorio remoto. Para poder subir a tu repositorio remoto, te debes asegurar de hacer commit a todos tus cambios al repositorio local. La sintaxis de ese comando es la siguiente: git push <nombre del repositorio> <nombre de la rama>

Comando git pull: es un comando de Git utilizado para actualizar la versión local de un repositorio desde otro remoto. Es uno de los cuatro comandos que solicita interacción de red por Git. Por default, git pull hace dos cosas. Actualiza la rama de trabajo actual (la rama a la que se ha cambiado actualmente). 2- Actualiza las referencias de rama remota para todas las demás ramas. Sintaxis: git pull NOMBRE-REMOTo NOMBRE-RAMA

Comando git branch: El comando git branch permite crear una rama nueva dentro de un desarrollo. Si quieres empezar a trabajar en una nueva función, puedes crear una rama nueva a partir de la rama main con git branch "nombre\_branch".

Comando git checkout "nombre\_branch": Permite realizar un cambio desde, por ejemplo, la branch principal master hasta la branch titulo.

Comando git checkout -b "nombre\_branch": Crea la rama o branch y también ingresa de forma automática a la misma para comenzar a trabajar.

Comando git merge: El comando git merge permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama. La sintaxis es: git merge <branch> (con el nombre de la rama individual que se desea combinar). Cómo traer el trabajo realizado en una branch a otra branch, como la master, usando el comando git merge. Que git merge genera un nuevo commit, informando que hubo una mezcla entre dos branches.

Comando git rebase "nombre\_branch": Aplica los commits de otra branch de desarrollo en la Branch actual o que se está utilizando en el momento. Cómo traer commits de una branch a otra con git rebase. Que git rebase no genera un commit de merge, lo que simplifica nuestro log.

Comando git log --graph: Permite visualizar los cambios generados desde cada una de las Branchs que se tienen creadas.

Comando git restore "nombre\_archivo": El comando restore de Git se usa para restaurar archivos en el árbol de trabajo desde el índice u otra commit. Este comando no actualiza la rama actual. También puede restaurar archivos en el índice desde otra commit. deshacemos una modificación que aún no fue agregada al index o stage, o sea, antes de hacer git add.

Comando git restore --staged "nombre\_archivo": Es usado para quitar los archivos del area provisional (stagged area), el contenido se restaura desde el HEAD. Por ejemplo cuando por accidente los agregaste a el area provisional y deseas deshacer el cambio. Después de agregar con git add, para deshacer una modificación, necesitamos sacarlo de este estado, con git restore --staged.

Comando git revert "numero completo del hash": El comando git revert es una operación para deshacer de forma progresiva que ofrece una forma segura de deshacer los cambios. En vez de eliminar confirmaciones o dejarlas huérfanas en el historial de confirmaciones, la reversión creará una nueva confirmación que invierte los cambios especificados.

Comando git stash: El comando git stash guarda las modificaciones en un lugar temporal (guardar para después). Después de usar stash, las modificaciones no van a quedar realmente aplicadas a nuestro código hasta que recuperemos esas modificaciones.

¿En qué momento el git stash es útil?: Cuando necesitamos pausar el desarrollo de alguna funcionalidad, o corrección, antes de finalizar, y tal vez no sea interesante realizar un commit, pues nuestro código puede no estar funcionando aún. En este caso es interesante guardar el trabajo para poder volver a él después.

Comando git stash list: tengo una lista de todos los stash que yo puedo agarrar y colocar ahí dentro de nuestro proyecto.

Comando git stash pop: Agarra la última modificación que agregué al stash y la elimina después de esa lista de stash. agarra del stash la última modificación agregada ahí en ese stash, la coloca en nuestra carpeta y después la elimina del stash.

Comando git checkout "Primeros siete caracteres del hash de la rama": No está diciendo que ahora estamos en un modo desanexado del HEAD, de nuestro estado HEAD que es el lugar en donde nosotros estamos parados. Dice que podemos por ejemplo mirar, experimentar, hacer cambios, commitear sin problemas y esos commits, todos esos cambios van a ser descartados cuando por ejemplo volvamos a nuestra branch master.

Comando git diff: Con el comando git diff, vimos que es posible visualizar las modificaciones realizadas en determinado código. Podemos ver las diferencias entre commits, branches, etc.

Comando git diff "primer hash" .. "segundo hash": Permite visualizar las modificaciones que fueron agregadas desde un commit de inicio hasta uno final, que en este caso sería el commit en donde se encuentra el desarrollo actual del proyecto.

Comando git log -n 2: Con esto vamos a ver los últimos dos commits que hemos hecho.

Comando git tag -a "v0.1.0" -m "mensaje...":

Comando git tag: Permite visualizar todas las tags que se tienen creadas para un proyecto.

¿Cómo se muestran las modificaciones con el comando git diff?:

+ línea agregada

- línea eliminada

- línea modificada (versión antigua)

+ línea modificada (nueva versión)

¿Qué es un Tag?: Un tag marca un punto en nuestra aplicación, un punto que no puede ser modificado, fijo, por ejemplo el lanzamiento de una versión y la versión 0.1, una vez lanzada nunca más es modificada. Cualquier modificación que yo quiera agregar va a ser agregada a la 0.2 y no a la 0.1. Entonces la tag sirve exactamente para eso, para definir un punto que no cambia nunca más en nuestra aplicación. Solo voy a crear un punto al que puedo llegar y tener un código exactamente en ese estado.

¿Qué resultado genera el envío de una tag para Github?: Genera una Release, o sea, conseguiremos bajar un archivo compactado con nuestro código en este punto.

Resumidamente, ¿para qué sirve el comando git checkout?: Para dejar nuestro código en un estado determinado. La descripción del comando git checkout --help, en una traducción libre es: “Actualizar los archivos en working tree para que queden en la versión especificada. […]”. Básicamente, podemos dejar nuestro código en el estado de último commit de una branch, de un commit específico, o incluso tags (que veremos más adelante).

Significado de la palabra WIP: (Work in progress) Estado de trabajo en progreso, el cual será guardado para ser utilizado después con el comando git stash.

Que son las branches: Las branches ("ramas") se utilizan para desarrollar funcionalidades aisladas entre sí. La branch master es la branch "predeterminada" cuando creas un repositorio. Es interesante separar el desarrollo de funcionalidades en diferentes branches, para que los cambios en el código de una no influyan en el funcionamiento de otra. Que una branch (o rama) es una línea de commits separada, y que se puede usar para desarrollar funcionalidades independientes. Que con branches separadas, podemos evitar que el código de una funcionalidad interfiera con otra.

FETCH: Significa el lugar de donde nosotros vamos a obtener las informaciones de ese servidor remoto.

PUSH: Es el lugar al que nosotros vamos a enviar informaciones.

Repositorios remotos: Servidor que va a ser un repositorio de Git que solo va a controlar y guardar las modificaciones sin necesidad de entrar a un archivo y editarlo.

Archivo .gitignore: Aquí se nombran o mencionan los archivos y carpetas que no queremos se tengan en cuenta dentro de Git. Será realmente necesario tenerlo dentro de los repositorios que se creen.

HASH: Es una identificación única que recibirá el proyecto, por ende no podrán existir un proyecto con dos hash distintos.

HEAD: Estado actual de nuestro código, es decir, donde nos colocó Git.

Working tree: Lugar donde los archivos realmente están siendo almacenados.

index: Lugar donde Git almacena lo que será commiteado, es decir, la ubicación entre el working tree y el repositorio de Git en sí.

Link en donde se pueden revisar los diferentes estados con Git: https://git-scm.com/book/es/v2/Fundamentos-de-Git-Guardando-cambios-en-el-Repositorio

Link para customizar comandos del git log: https://devhints.io/git-log