GUÍA DE APRENDIZAJE

**GIT - Control de versiones de código**

|  |  |
| --- | --- |
| **OBJETIVO DE****APRENDIZAJE** | **OA 10.** Instalar software de productividad y programas utilitarios en un equipo personal, de acuerdo a los requerimientos de los usuarios. |
| **APRENDIZAJE** **ESPERADO** | **AE 1.** Instala aplicaciones para optimizar tareas en el puesto de trabajo considerando, propósito específico, utilidades, especificaciones de trabajo y requerimientos del usuario. |
| **CRITERIO DE** **EVALUACIÓN** | **1.2** Instala software de productividad específica, para facilitar y optimizar el rendimiento de trabajo de un usuario en actividad particular de trabajo, por ejemplo, diseño, fotografía u otros, respetando protocolos y licencias de instalación.**1.5** Instala y configura aplicaciones que permiten administrar y configurar las aplicaciones de productividad instaladas en el equipo, considerando protocolos de la industria y normativa vigente. |
| **OBJETIVOS DE** **APRENDIZAJE****GENÉRICOS** | **A.** Comunicarse oralmente y por escrito con claridad, utilizando registros de habla y de escritura pertinentes a la situación laboral y a la relación con los interlocutores.**B.** Leer y utilizar distintos tipos de textos relacionados con el trabajo, tales como especificaciones técnicas, normativas diversas, legislación laboral, así como noticias y artículos que enriquezcan su experiencia laboral.**C.** Realizar las tareas de manera prolija, cumpliendo plazos establecidos y estándares de calidad, y buscando alternativas y soluciones cuando se presentan problemas pertinentes a las funciones desempeñadas.**H.** Manejar tecnologías de la información y comunicación para obtener y procesar información pertinente al trabajo, así como para comunicar resultados, instrucciones e ideas. |

## GLOSARIO

**Directorio:** Es el comienzo de una estructura de carpetas y/o archivos específica en nuestro computador.

**Repositorio Local:** Se crea al ejecutar el comando **git init** en un directorio y es el versionado de código en nuestro computador.

**Repositorio Remoto:** Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet

**README.md:** Es un archivo que se crea en el directorio raíz de un programa computacional que sirve como guía de instalación o configuración y además de indicar de qué trata el programa. Es un símil a un archivo LEEME.txt, sin embargo, la extensión .md hace referencia a un lenguaje de marcado muy utilizado en la informática llamado [**Markdown**](https://markdown.es/)**.**

**Rama:** Sirve para hacer una copia de nuestro código y seguir trabajando ahí, sin afectar el código original.

**Rama Master:** Es la rama por defecto que usa Git cuando comenzamos a utilizarlo.

**Origin:** Es el nombre que por defecto Git le da al servidor que has clonado, ósea el Repositorio Remoto.

## SISTEMA DE CONTROL DE VERSIONES (SCV)

“Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.” (Git,s.f.)

## GIT

Git es un sistema de control de versiones **(SCV)** creado para la eficiencia, trazabilidad y mantenimiento de versiones de aplicaciones computacionales. Su propósito es mantener un historial de todos los cambios que se generan en los archivos de la aplicación y coordinar el trabajo colaborativo.

Si bien Git no es el único SCV del mercado, es el sistema de control de versiones más popular a nivel mundial gracias a su rapidez y buen rendimiento en el trabajo en equipo, lo que se ve reflejado en la siguiente gráfica que nos provee [***Google Trends***](https://trends.google.es/trends/explore?geo=ES&q=%2Fm%2F05vqwg,%2Fm%2F08441_,%2Fm%2F012ct9)que compara los principales SCV como GitHub, Mercurial y Subversion.





*Imagen extraída el 11 de noviembre de 2020*

Además, podemos ver en la página oficial de [**Git**](https://git-scm.com/) que reseña las principales grandes empresas tecnológicas que utilizan esta herramienta como control de versiones de sus proyectos.



## GITHUB

[**GitHub**](https://github.com/) es una plataforma web de desarrollo colaborativo que permite el alojamiento y gestión de repositorios remotos Git, donde cada repositorio se almacena de forma pública a menos que se pague por mantener el código privado.

Cada repositorio creado en GitHub es propiedad de una cuenta de usuario u organización y sólo usuarios autorizados pueden realizar cambios en dicho repositorio.

Así como GitHub, existen muchas otras plataformas web que nos permiten la gestión de repositorios remotos Git y que nos entregan diversos beneficios como, por ejemplo, la gestión y asignación de tareas o también llamadas *issues* entre el equipo de desarrollo.

Las más populares en el mercado son GitHub, Bitbucket y GitLab.



##

## FUNCIONAMIENTO Y COMANDOS BÁSICOS

Git es multiplataforma, esto significa que es posible utilizarlo tanto en Windows, MacOs o Linux.

Para utilizar Git en Windows es necesario que se descargue desde su página oficial [**https://git-scm.com/downloads**](https://git-scm.com/downloads)**.** Luego de su instalación tendremos dos programas; Git GUI y Git BASH, sin embargo, para este caso se usará Git Bash que es una consola de comandos de Git para Windows, que además nos permitirá utilizar comandos básicos de Linux.



A continuación, se describirán los principales comandos de Git seguido de una imagen explicativa de cada comando y su flujo.

**git init:** Con este comando creamos el **repositorio local**, basta con ejecutar **git init** dentro del directorio (directorio: ruta de la carpeta donde se encuentra la aplicación) de la aplicación computacional.

**git clone:** Crea una copia local en el **Área de trabajo** de un repositorio remoto que se encuentra alojado en GitHub por ejemplo.

**git add.:** Registrar todos los nuevos cambios (se añaden al **Área de Ensayo**)

**git add <archivo>:** Registra los cambios de un archivo al **Área de Ensayo**.

**git commit -m “mensaje commit”:** Se usa para confirmar los cambios registrados con **git add**, además registra los cambios al **Repositorio Local** con un comentario de los cambios realizados en el código.

**git push:** Envía los cambios del **Repositorio Local al Repositorio Remoto***.*

**git pull:** Se usa para actualizar el **Repositorio Local y Área de Trabajo** al último commit subido.

**git status:** Permite ver los archivos que han tenido cambios en el área de trabajo.

## ORDEN DE COMANDOS PARA TRABAJAR DE FORMA INDIVIDUAL CON GIT

A la hora de trabajar con Git en algún proyecto, es probable que tengamos que trabajar de manera individual o de forma colaborativa, para ambos casos hay ciertas diferencias que debemos identificar para no cometer errores.

Los pasos para trabajar de forma individual consisten en crear versiones **(commits)** y subirlos a GitHub sin mayores variaciones.

Los pasos a seguir son:

1. **git clone**: Esto para clonar un repositorio ya creado en GitHub (Sólo la primera vez).
2. **git add. o git add <directorio>:** Para añadir archivos creados o modificados.
3. **git commit -m “mensaje commit”:** Para agregar un mensaje al commit que será la próxima versión del sistema.
4. **git push:** Para subir el commit al **Repositorio Remoto**.
5. Repetir comandos desde **git add**.



**Flujo de comandos para trabajar con Git de manera individual**

## ORDEN DE COMANDOS PARA TRABAJAR DE FORMA COLABORATIVA CON GIT

Los pasos para trabajar con más personas en un mismo repositorio consisten en bajar nuevos posibles cambios antes de subir una versión **(commit)** a GitHub.

Ocuparemos los siguientes pasos para trabajar con más personas en un mismo aplicativo computacional:

1. **git clone:** Esto para clonar un repositorio ya creado en GitHub (Sólo la primera vez).
2. **git pull:** Este comando descarga los últimos commit que se hayan subido al servidor remoto por alguna otra persona.
3. **git add. o git add <directorio>:** Para añadir archivos creados o modificados.
4. **git commit -m “mensaje commit”:** Para agregar un mensaje al commit que será la próxima versión del sistema.
5. **git push:** Para subir el commit al **Repositorio Remoto**.
6. Repetir comandos desde **git add**.

****

**Flujo de comandos para trabajar con Git de manera colaborativa**

## COMANDOS BÁSICOS LINUX

Existen algunos comandos básicos y útiles de Linux que ayudarán a mejorar la usabilidad de la consola Bash de GitHub, las cuales son:

**pwd:** Muestra la ruta completa del directorio actual.

**cd <directorio>:** Abrir directorio (moverse una carpeta adelante).

**cd.:** Ir un directorio hacia atrás.

**ls:** Lista todo lo que hay dentro del directorio donde nos encontramos.

**clear:** Limpia la consola.

## VIDEO TUTORIAL

En el siguiente enlace se encuentra un video llamado ***“*Aprende Git en 10 Minutos”** donde se verá la instalación, configuración y uso de Git y Github. <https://youtu.be/DuYjcOZw11s>

## DESAFÍO 1. Crea tu primer repositorio en GitHub para luego clonarlo

**Descripción**

Para este desafío se requiere crear un repositorio en <http://github.com> llamado **“mi-primer-repositorio”**, dicho repositorio debe ser creado con un archivo **README.md**.

Luego, el repositorio debe ser clonado en sus computadores para luego realizar un cambio al archivo **README.md**, posterior a esto, subir los cambios como una nueva versión a GitHub, donde la descripción de dicha versión (commit), debe tener la descripción del cambio que se realizó, por ejemplo **“Se modificó el texto original del archivo readme”**.

**Buenas prácticas:** Debe tener en cuenta que el texto que se ingresa en el comando commit -m “...”, debe ser claro, conciso y descriptivo de los cambios que se realizaron.

**Habilidad**

1. Crear repositorios en GitHub.
2. Clonar un repositorio.
3. Crear un mensaje correcto para el comando commit
4. Utilizar los principales comandos para trabajar de manera individual.

## DESAFÍO 2. Subir un repositorio local ya existente a un repositorio remoto en Github

**Descripción:**

Para este desafío, se requiere crear un programa computacional que permita sumar dos números ingresados a través del teclado, para luego subir dicho programa a un nuevo *Repositorio Remoto* en Github.

Este ejercicio presta utilidad cuando tenemos un programa computacional creado o avanzado y se necesita subir a un **Repositorio Remoto**.

**Pasos a seguir**

1. Cree un programa computacional que sume dos números.
2. Guarde dicho archivo en una carpeta llamada **“Suma dos numeros”**. (Éste será su **Área de Trabajo**).
3. Cree el Repositorio Remoto en Github con el nombre: **“suma-dos-numeros-por-teclado”** (Cree el repositorio SIN un archivo README.md).

Una vez tenga listo los pasos, deberá subir su programa computacional al *Repositorio Remoto* ya creado, para esto deberá seguir los siguientes comandos git dentro de su **Área de Trabajo**:

$ git init

$ git add .

$ git commit -m “Primer commit de mi programa que suma dos números por teclado”

$ git remote add origin https://github.com/SuNombreDeUsuarioGithub/nombre-del-repositorio-remoto.git

$ git push -u origin master

**Habilidad**

Subir un programa computacional ya existente a un nuevo **Repositorio Remoto** en Github.

## DESAFÍO 2.1. Crear archivo README.md en Github para luego bajar al Repositorio Local

**Descripción:**

Para este desafío, se requiere abrir su cuenta de Github y entrar a su repositorio anteriormente creado llamado **“suma-dos-numeros-por-teclado”**, donde tendrá que crear un archivo **README.md** para luego ser bajado a su **Repositorio Local**. En dicho archivo deberá documentar de manera breve en qué consiste su programa computacional y en qué lenguaje de programación se creó.

Posterior a esto, tendrá que subir los cambios como una nueva versión (commit) a su Repositorio Remoto.

**Pasos a seguir**

1. Abra su cuenta de Github y entre al repositorio anteriormente creado **“suma-dos-numeros-por-teclado”**
2. Cree un archivo README.md dentro de su Repositorio Remoto en Github.

##

1. Baje los cambios de la última versión de su **Repositorio Remoto**con el comando git pull (esto bajará el nuevo archivo README.md creado en Github).
2. Documente de manera breve que hace su programa computacional y describa en qué lenguaje está construido.
3. Suba los cambios a Github agregando un mensaje en el commit que describa el cambio que realizó.

**Buenas prácticas:** Es importante mantener documentado nuestros proyectos o programas computacionales en un archivo README.md, esto permite conocer por ejemplo: Qué hace el programa, en qué lenguaje se encuentra programado, cómo ejecutarlo, entre otros.

**Habilidad**

* Utilizar el comando **git pull**.
* Aprender la importancia del archivo **README.md**

##

## DESAFÍO 3. Calculadora

**Descripción:**

Para este desafío, se requiere programar una calculadora que permita realizar las cuatro operaciones básicas; suma, resta, multiplicación y división con un máximo de 2 números ingresados por teclado.

Tendrá que crear un nuevo repositorio en Github y clonarlo en su computador para luego ir subiendo el código a medida que vaya avanzando, creando así un mínimo de 5 versiones **(commits)** de su calculadora.

**Pasos a seguir**

1. Abra su cuenta de Github y cree un repositorio llamado **“mi-calculadora”** para luego clonarlo en su computadora.
2. Comience a programar su calculadora.
3. Suba una nueva versión (commit) cuando su calculadora logre sumar 2 números ingresados por el teclado.
4. Suba una nueva versión (commit) cuando su calculadora logre restar 2 números ingresados por el teclado.
5. Suba una nueva versión (commit) cuando su calculadora logre multiplicar 2 números ingresados por el teclado.
6. Suba una nueva versión (commit) cuando su calculadora logre dividir 2 números ingresados por el teclado.
7. Cree un archivo README.md dentro de su **Área de Trabajo** y documente de forma concisa de qué trata su programa y en qué lenguaje está programado.
8. Suba una versión (commit) con el nuevo archivo README.md que acaba de crear.

**Buenas prácticas**: El concepto de realizar un commit, es subir siempre una versión nueva y acotada de su programa, además de ser una versión funcional, ya sea un commit que refleje una mejora o que corrija algún error.

Además, los commit deben ser soluciones pequeñas. Por ejemplo, si su programa tiene 4 errores, deberá subir 4 commit por cada corrección.

**Habilidad**

Aprender el buen uso del comando commit a la hora de subir una nueva versión de un programa computacional.

## ¿QUÉ ES UNA RAMA EN GIT?

Las ramas o **branch** son una de las principales utilidades que disponemos en Git para llevar un mejor control del código, se trata de crear caminos que podemos tomar a lo largo del desarrollo de un software, algo que ocurre naturalmente para resolver problemas, o crear nuevas soluciones o funcionalidades.

En la práctica se trata que nuestro código tenga distintos estados donde el programador pueda pasar de uno a otro o mezclar estos estados de manera ágil.

## RAMA MASTER

La **rama master**es el estado por defecto cuando usamos git, lo puedes corroborar usando el comando git branch dentro de un programa que use git. Este comando listará todas las ramas que se han creado y se mostrará en verde en la rama que nos encontramos en ese momento, en el caso que no se haya creado ninguna, sólo aparecerá la **rama master***.*

Esta rama, no es distinta a cualquier otra que se pueda crear de forma manual, sin embargo, habitualmente se utiliza para alojar el software en su última versión funcional y estable, la que se encuentra desplegada en algún servidor.

## EXPLICACIÓN GRÁFICA

En la siguiente imagen, podemos apreciar que existen 2 ramas, la rama **master** y una rama llamada **branch2**. En la gráfica, cada círculo significa que se subió un commit al **Repositorio Remoto**.

El punto naranjo muestra que fue el primer **commit**en la rama *master,* seguido de 5 commit más.

En el punto verde podemos observar que se creó la rama **branch2**, la cual tuvo 3 *commit* para luego ser juntada nuevamente con la **rama master**.



## EJEMPLO PRÁCTICO

Imaginemos que trabajamos en una empresa de desarrollo de software, y el cliente pide una nueva funcionalidad a un sistema que ya se encuentra en funcionamiento en un servidor con su última versión en la **rama master**.

La nueva funcionalidad no estaba prevista y tampoco se sabe si dará el resultado que se espera ya que será algo experimental, por lo que se pide que se desarrolle para poder hacer pruebas. El programador puede crear una rama dedicada para trabajar en dicha funcionalidad, y así no tocar ni afectar con algún posible error la versión que se encuentra funcionando en el servidor.

Una vez terminada la funcionalidad, el programador sube al servidor el código con la rama creada para que se puedan hacer las pruebas del sistema que contiene la nueva funcionalidad.

En el caso que el cliente quiera mantener la nueva funcionalidad, basta con mezclar la nueva rama a la rama *master* y así mantener actualizada a la última versión del programa, en este punto el programador puede borrar la rama creada.

De lo contrario, si el cliente pide quitar la nueva funcionalidad, basta con que el programador vuelva a poner la *rama* master en el servidor.

## COMANDOS PARA TRABAJAR CON RAMAS

**git branch:** Muestra un listado de las ramas creadas.

**git branch desarrollo:** Crea una nueva rama llamada “desarrollo”.

**git branch -d nombre-rama:** permite eliminar una rama específica.

**git checkout nombre-rama:** Permite moverse entre distintas ramas, por ejemplo: git checkout desarrollo, git checkout master.

**git merge nombre-rama:** Permite mezclar ramas, por ejemplo, si nos encontramos en la **rama master**, y queremos mezclar la rama desarrollo con la **rama master**, hacemos git merge desarrollo, esto nos traerá todos los nuevos cambios de la rama desarrollo a la rama *master*.

## VIDEO TUTORIAL

En el siguiente enlace se encuentra un video llamado “Ramas o Branch, Uniones o Merge”, podrá ver ejemplos prácticos de los principales comandos relacionados con el trabajo con ramas.

<https://www.youtube.com/watch?v=tFr0Vg1q9Eg&ab_channel=Bluuweb%21>

## DESAFÍO 4. Trabajando con ramas en el Repositorio Local

**Descripción**

Para este desafío se requiere trabajar con el programa que creó en el desafío número 2, en el cual se creará una rama paralela para trabajar en una nueva mejora y posteriormente mezclarla con la **rama master**.

**Pasos a seguir**

1. Comenzaremos usando el comando git branch en la consola dentro del **Área de Trabajo** para listar las ramas que se encuentran creadas en nuestro **Repositorio Local.**
2. Crearemos una rama llamada **tercer\_numero** con el siguiente comando **git branch tercer\_numero**.
3. Usaremos nuevamente el comando **git branch,** y observaremos que ahora se listan 2 ramas donde la rama *master* aparece en color verde, indicando que nos encontramos en dicha rama.
4. Estando en la rama *master* nos cambiaremos a la rama **tercer\_numero**recién creada con el siguiente comando: **git checkout tercer\_numero**.
5. Para verificar en la rama que nos encontramos usaremos nuevamente el comando **git branch**.
6. Ahora haremos un primer cambio en nuestro código, el cambio será agregar un tercer número ingresado por teclado para que puedan ser sumados.
7. Una vez tenga los cambios en su código, agregue los cambios al **Repositorio Local** con los comandos **git add. y git commit -m “texto commit”.**
8. Observe su código y vuelva a cambiarse a la **rama master**. Se dará cuenta que sus cambios realizados ya no están, esto ocurre porque los cambios por el momento sólo se encuentran en la rama **tercer\_numero**.
9. Ahora mezclaremos ambas ramas para luego subir los cambios al **Repositorio Remoto***.*
10. Antes de mezclar los cambios de la rama **tercer\_numero**a la **rama master**, debemos estar posicionados en la **rama master** (a la rama donde migrarán los cambios), para esto verifique que se encuentra en la **rama master***,* de lo contrario, deberá cambiarse de rama.
11. Para mezclar ambas ramas, utilizaremos el comando **git merge tercer\_numero**. En este punto, podrá verificar que en ambas ramas tenemos el mismo código.
12. Ahora posicionados en la **rama master**, subiremos los cambios al Repositorio Remoto con git push.

En este punto nuestra rama **“tercer\_numero”** sólo se encuentra en nuestros computadores, ya que el git push que hicimos fue desde la rama **“master”** y no desde la rama **“tercer\_numero”.**

Además, la mejora de agregar un tercer número ya fue creada, así que supondremos que ya no necesitamos trabajar más en dicha mejora, por lo tanto, la rama **“tercer\_numero”** ya cumplió su objetivo, así que la eliminaremos.

1. Para eliminar una rama usaremos el comando **git branch -d nombre\_rama***.*
2. Vuelva a usar el comando git branch para verificar que sólo tiene creada la **rama master**.

**Buenas prácticas**: Es importante entender que al usar Git, muchas veces usaremos distintas ramas para trabajar tareas puntuales, como por ejemplo una nueva funcionalidad, y para cuando ya se haya creado lo mejor es borrar la rama para así mantener un orden en nuestro repositorio.

**Habilidad**

* Crear ramas.
* Moverse de una rama a otra.
* Eliminar ramas.
* Mezclar ramas.

## VISUALIZAR DE MANERA GRÁFICA LOS CAMBIOS EN NUESTRO REPOSITORIO EN GITHUB

En Github podemos ver un historial de cambios, entre ellos commit, creación de ramas y merge, de manera gráfica. Dicho historial gráfico lo encontramos accediendo al menú superior Insights/Network.



##

## DESAFÍO 4.1. Trabajando con ramas y subirlas al Repositorio Remoto

**Descripción**

Para este desafío trabajaremos con el programa computacional del desafío número 4. Se requiere crear una nueva rama para trabajar en una nueva funcionalidad que permita obtener el promedio de la suma de los 3 números, además de validar que el resultado sea menor a 10, de lo contrario, mostrar un mensaje de alerta indicando que el promedio debe ser un número menor a 10.

Recordemos que el uso correcto del comando commites subir versiones funcionales y acotadas. Por lo tanto, no podrá subir toda la nueva funcionalidad en un solo commit, sino separado en bloques funcionales.

**Pasos a seguir**

1. Entrar al **Área de Trabajo**y crear una nueva rama llamada **promedio**.
2. Cambiarse a la nueva rama *promedio*, donde tendrá que programar la nueva funcionalidad.
3. Programe el cálculo de promedio de los 3 números e imprima el resultado.
4. Una vez tenga los cambios en su código, agregue los cambios al **Repositorio Local** con los comandos git add. y git commit -m “texto commit”.
5. Suba los cambios al **Repositorio Remoto** usando el comando git push origin promedio.

Si se percata, ahora usamos una variación en el comando git push a git push origin nombre\_rama, de ahora en adelante usaremos **SIEMPRE** este comando, esto va a indicar que subiremos cambios de una rama específica **nombre\_rama**hacia **origin (Repositorio Remoto)**. Si estamos posicionados en la rama *master*, usaremos git push origin master.

Para verificar de manera rápida en que rama está usted posicionado, puede mirar su consola de git como lo muestra la siguiente imagen:



1. Observe su **Repositorio Remoto** en Github, y podrá apreciar que se realizaron algunos cambios gráficos, producto de haber subido una segunda rama.
* El número 1, indica que estamos mirando los archivos y commit de la **rama master**. Si hacemos clic, podremos cambiar de rama y visualizar los cambios de dicha rama seleccionada.
* El número 2 indica que recientemente se subieron cambios en la rama **promedio***.*
* En el número 3 podemos hacer una comparación de los códigos subidos y realizar un **pull request**, (Solicitud de **merge** entre ramas, por ejemplo, una solicitud para mezclar la rama *promedio* y la **rama master**)



1. Programe la validación del promedio calculado, mostrando un mensaje de alerta si es que el resultado es igual o mayor a 10.
2. Haga un commit y súbalo al **Repositorio Remoto**.
3. Cámbiese a la rama *master* para que pueda mezclar la rama **promedio** a la **rama master***.*
4. Una vez mezcladas las ramas y estando posicionado en la **rama master**, actualice el archivo **README.md** para posteriormente realizar un commit y subirlo al **Repositorio Remoto** usando git push origin master.

En este punto, la funcionalidad se encuentra terminada, sin embargo, si queremos seguir trabajando en ella, por cualquier motivo, es importante **SIEMPRE** mantener la rama actualizada en su **Repositorio Local** con la última versión que tenga el **Repositorio Remoto**, que, en este caso, la última versión la tiene la rama *master* (un commit adelante) cuando actualizamos el **README.md.**



**\*Usted puede comparar el archivo README.md de ambas ramas haciendo git checkout nombre\_rama y podrá observar la diferencia.**

Para actualizar su rama **promedio** a la última versión en su **Área Local**, usted debe cambiarse a la rama *promedio* y ejecutar el comando git pull origin master. Este comando le permitirá bajar los nuevos cambios de la rama *master* que se encuentra en el **Repositorio Remoto (origin)** a la rama donde se encuentra posicionado, o sea a la rama **promedio***.*

En este momento su rama **promedio** del Repositorio Local se encuentra en la misma versión que la **rama master** del Repositorio Remoto, sin embargo, en Github, este cambio aún no ve reflejado, debido a que sólo se actualizó en su **Repositorio Local** y no se han subido los cambios de la rama **promedio**.

**Actualmente**



Para que su rama *promedio* quede actualizada a la última versión en el Repositorio Remoto, usted debe estar posicionado en la rama *promedio* y ejecutar el comando **git pull origin promedio**, este comando subirá los cambios de su rama al Repositorio Remoto, quedando de esta forma.

**Ahora**



**Buenas prácticas**: El uso del comando git push origin nombre\_rama y git pull origin rama es obligatoria cuando trabajamos con distintas ramas, sin embargo, es una buena práctica añadir siempre el origin nombre\_rama aunque estemos trabajando con una sola rama, de esta manera, evitaremos errores por olvido cuando comencemos a utilizar incluso más de dos ramas en un programa.

**Habilidad**

* Subir ramas al **Repositorio Remoto**.
* Mantener ramas actualizadas.

## MATERIAL DE CONSULTA

1. Página oficial de Git:

<https://git-scm.com/>

1. Página oficial de GitHub:

<https://github.com/>

1. Página oficial de Bitbucket:

<https://bitbucket.org/product/>

1. Página oficial de GitLab

<https://about.gitlab.com/>

1. git - la guía sencilla

<https://rogerdudler.github.io/git-guide/index.es.html>

1. La guía básica de Git y Github para principiantes

[https://medium.com/@sthefany/primeros-pasos-con-github-7d5e0769158c](https://medium.com/%40sthefany/primeros-pasos-con-github-7d5e0769158c)

1. Paper - GitHub como herramienta docente

<https://upcommons.upc.edu/bitstream/handle/2117/76761/JENUI2015_76-83.pdf>

1. Importancia archivo README.md

[https://docs.github.com/es/free-pro-team@latest/github/creating-cloning-and-archiving-repositories/about-readmes](https://docs.github.com/es/free-pro-team%40latest/github/creating-cloning-and-archiving-repositories/about-readmes)

1. Trabajar con ramas

<https://desarrolloweb.com/articulos/trabajar-ramas-git.html>

1. Aplicación Github de escritorio

<https://desktop.github.com/>

1. Plataforma de aprendizaje interactivo <https://learngitbranching.js.org/?locale=es_ES>

## REFERENCIAS

1. Git (n.f). Inicio - Sobre el Control de Versiones - Acerca del Control de Versiones. Recuperado el 11 de noviembre de 2020 de

<https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones>