**Gitbash:**

Comandos básicos:

**Ls:** Nos lista todo el contenido de la ubicación en donde nos encontramos.

**Cd:** es para movernos de directorio en directorio por ejemplo al usar LS nos apareció desktop que hace referencia al escritorio entonces para mover al escritorio escribimos

Cd desktop

además para ir hacia atrás y volver al directorio anterior escribimos cd ..

también para escribir nombres de carpetas o archivos que tengan espacios hay que escribir los mismos entre “” ””.

Otra cosa importante es que no importante es que no distingue entre mayúsculas o minúsculas. Escribiendo cd DESKTOP o cd desktop de las dos maneras nos lleva al escritorio.

**Pdw:** Nos muestra la ruta exacta de la computadora en donde nos encontramos.



**Ctrl + c sirve** para copiar en la terminal, pero ctrl + v no sirve para pegar.

**Para pegar utilizamos shift + Insert.**

**Touch:** Con este comando podemos crear un archivo, si el nombre del mismo tiene espacios hay que escribirlo entre “” “” y siempre al final hay que escribir la extensión del archivo.

**Mv**: podemos utilizarlo para renombrar un archivo primero escribimos el nombre y la extensión luego el nuevo nombre y la extensión.

****

**También mv** nos sirve para mover archivos y directorios de un directorio a otro.

**Mkdir:** es para crear carpetas o directorios.

Mkdir lucio

Mkdir “lucio el más grande”

Cp nos sirve para copiar y pegar un archivo o directorio en otro.

Cat: podemos arbir un archivo

Cat leila.txt



**Repositorio local:**

Lo primero que tenemos que hacer es configurar un usuario y una contraseña.

Esto lo hacemos con el comando **git config .**

**git config --global user.name "Lucio Sanfi"**

De esta manera establecimos Lucio Sanfi como nombre de usuario.

**git config --global user.password "lahijadelalagrima74"**

De esta manera establecimos lahijadelalagrima74 como contraseña.

Luego de establecer esta configuración básica hay que crear nuestro repositorio local:

**git init**

De esta manera ya configuramos usuario y contraseña y creamos un repositorio local.

Para agregar archivos carpetas o proyectos a nuestro repo local de git utilizamos el comando:

**git add**

Podemos usar **git add “nombre de archivo”.extensión** o **git add .** lo que va a agregar todos los archivos al repositorio local de git.

**git status** nos va a indicar que archivos están agregados en el repositorio o no y cuales están agregados, pero necesitan un **commit.**

Si por ejemplo hacemos un **git status** y vemos que hay archivos que necesitan comittear sus cambios escribimos **git commit -m “**aclaración del cambio realizado en el commit” luego escribimos el nombre del archivo a comittear y su extensión o también podemos escribir . para comittear todos los archivos que necesiten actualizar sus cambios.

**Ya agregamos los archivos al repositorio remoto y en caso de modificarlos ya comitteamos los cambios. ¿Qué sigue?**

Pues al trabajar en equipo seguramente tengamos que hacer un push para actualizar estos archivos en un repositorio remoto.

Con el comando **git push** vamos a llevar los cambios de estos archivos en nuestro repositorio local al repositorio local.

De forma básica nuestro repositorio local tiene 3 áreas:

**Working área:** es el área inicial antes de hacer un git add y agregar nuestros archivos al repositorio local.

**Staging área:** es el área en donde se encuentran los archivos que esta en el repositorio local pero cuyos cambios aun no fueron actualizados o confirmados.

**Commit área:** Al realizar un git commit estamos actualizando los cambios y los archivos quedan en el repositorio local con los últimos cambios ya confirmados.

Todo esto es en el repositorio local recién al utilizar un **git push** vamos a empezar a enviar cambios al repositorio remoto.

Para almacenar nuestros repositorios locales en repositorios remoto vamos a necesitar una cuenta en **GitHub** o alguna pagina similar que permita alojar repositorios remotos.

Vamos a practicar sobre repositorios remotos con Github pero se puede usar los mismos comandos con cualquier repositorio remoto.

**Conectar repositorio local con remoto:**

**git remote nombre del repositorio remoto url del repositorio remoto.**

**Git remote Upex-Academy-Automation** [**https://github.com/luciosanfi/Upex-Academy-Automation.git**](https://github.com/luciosanfi/Upex-Academy-Automation.git)

De esta manera quedó establecida la conexión entre el repositorio local y el remoto.

Luego de establecer esta conexión hay que comenzar a trabajar con ramas o branchs.

**Trabajar con ramas o Branchs en git:**

Es muy importante para hacer un push y subir nuestras actualizaciones del proyecto de nuestro repositorio local al repositorio remoto, tener ramas creadas previamente.

Las ramas nos permiten tener distintas líneas de tiempo y diferentes versiones del proyecto en cada rama.

Con el comando **git branch** en nuestro repo local, nos muestra la lista de ramas que tenemos creadas y en cual rama estamos parados. Nosotros vamos a pushear la información del repo local hacia una rama especifica del repo remoto.

Con el comando **git branch nombrerama** vamos a crear una rama nueva en nuestro repo local.

Con el comando **git checkout nombrerama** vamos a cambiar de una rama a otra dentro del repo local.

Con el comando **git branch -d “nombre de la rama”** vamos a borrar una rama.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Git push:**

En la foto de arriba se muestra que pasa cuando hacemos un push a nuestro repositorio remoto pero nuestra rama del repositorio local no existe en el remoto. Git nos avisa esto mismo y nos recomienda automáticamente el comando **git push --set-upstream Upex-Academy-Automation develop** para crear la rama develop en nuestro repositorio remoto y luego realizar el push. El hecho de que nos pida iniciar sesión es normal ya que es para justamente poder subir cambios hacia GitHub o el repositorio remoto que utilicemos.

Luego GitHub nos pide que le demos autorización a vscode:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

De esta manera en nuestro repositorio remoto aparecen los cambios de nuestro último commit.

**Pull Request:**

Una vez que tengamos un repo local conectado con un repo remoto y distintas ramas creadas en ambos, seguramente vamos a estar trabajando en equipo y antes de hacer un **merge** de nuestra rama remota con la rama main(que es la rama principal de todo el proyecto) vamos a hacer un pull Request para que alguien revise el cambio que agregamos en el código y en base a eso apruebe o no nuestro **merge** con la rama principal.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

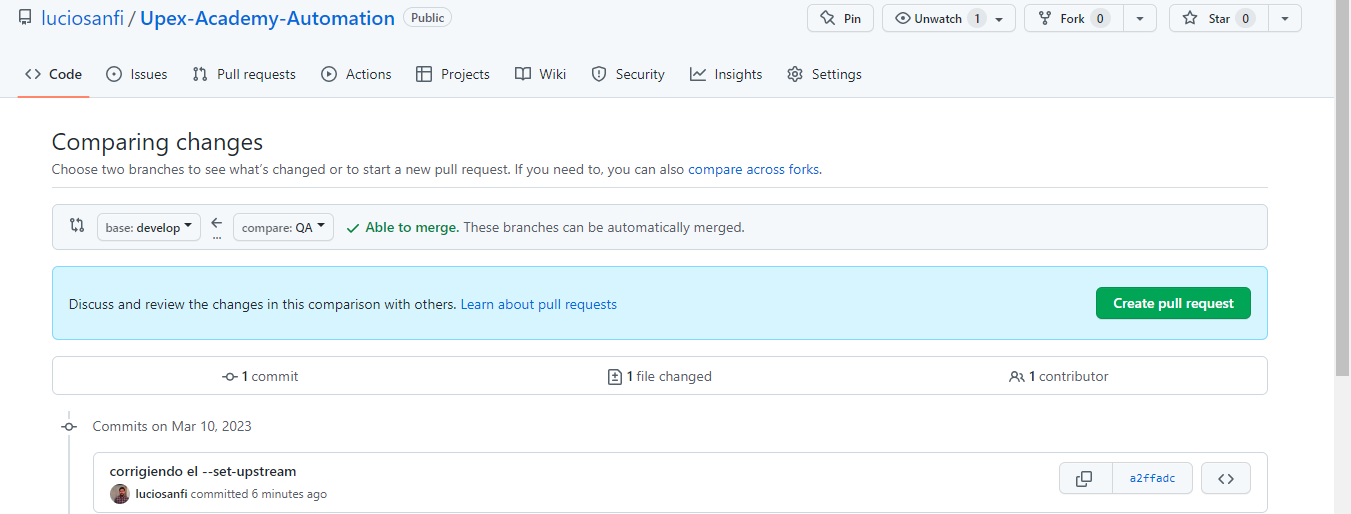
En el repositorio remoto vamos a la sección que dice Pull requests y seleccionamos nuevo pull Request.

Ahora nos va a preguntar que ramas queremos **mergear:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para el ejemplo creamos una rama llamada QA en nuestro repo local, le agregamos un commit nuevo que develop no tiene y hacemos un push. Ahora vamos a pull Request y elegimos que vamos a mershear desde QA a develop.



GitHub compara ambas ramas y nos muestra las diferencias en la rama QA indicando que tiene un archivo cambiado y un commit nuevo.

Presionamos en create pull Request.

Luego de esto la persona que analice si nuestro código esta para mergear va a aprobar o no nuestro pull Request y en caso de que nos lo apruebe vamos a presionar en merge pull Request:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**git pull:**

Utilizamos este comando para actualizar cambios en una rama desde el repositorio remoto hacia la misma rama en el repositorio local.