**CONFIGURAR GIT:**

* Para saber la versión

**git --version**

* Configuración global, y no por proyecto, se usan comillas dobles porque va a tener espacios

**git config --global user.name “Diana Maria Toro Lopez”**

* Configuración de correo electrónico

**git config --global user.email** [**dianamtlopez@hotmail.com**](mailto:dianamtlopez@hotmail.com)

* Configurar editor de texto, --wait es para que la terminal espere hasta que cerremos el editor de texto

**git config --global core.editor “code --wait”**

* Para los que no funcione:

**git config --global core.editor “code -w”**

* Para volver a nano o vi predeterminado

**git config --global --unset core.editor**

* Eliminar todos los valores de core.editor

**git config --global --unset-all core.editor**

* Si queremos ver la configuración global

**git config --global -e**

* Configuración de core.autocrlf: cuando varias personas trabajan sobre un mismo repositorio, todos tienen sistemas operativos diferentes, los de Windows cada vez que agregamos un salto de línea Windows va a agregar dos caracteres especiales para marcar esa línea como un salto de línea, en este caso se gregan los caracteres especiales de carriage return o CR y line feed o LF, en Linux o mac solo se agrega line feed o LF, el desarrollador cuando trabaja en Windows, y desee subir código va a tener que eliminar el carácter CR y si desea descargar código del repositorio, debe agregarlo, para esto, debemos cambiar la configuración de git y la propiedad de core.autosclf debe tener el valor en true, para los usuarios de Linux y de mac, con esta configuración, git debe ser capaz de eliminarlo y no dejar subir el carácter y para eso la propiedad de core.autocrlf debe tener la propiedad de input y de esta manera quedan configurados los saltos de línea ya sean en Windows o en Linux y mac:

**git config --global core.autolrlf true**

* Para ver el listado de todas las configuraciones escribimos:

**git config -h**

* Para listar todos los archivos y carpetas que se encuentran en un determinado directorio

**ls**

* Para saber en qué carpeta o directorio estamos, debemos presionar

**pwd**

* Para movernos entre las diferentes carpetas cd + espacio + el nombre de la carpeta donde queremos movernos.

**cd Descktop/**

* Para crear un directorio nuevo

**mkdir miweb**

* Para inicializar el repositorio, debo dar **git init** y nos va a mostrar la siguiente información:

**Initialized empty Git repository in C:/Users/User/Desktop/miweb/.git/**

**Lo que nos indica .git/ es que es un directorio oculto, allí se van a almacenar las distintas versiones de nuestro código, las ramas, los commit etc**

* Para poder visualizar archivos ocultos

**ls –a**

* Para ver el estado actual de nuestro repositorio

**git status**

nos muestra lo siguiente:

**On branch master**

**No commits yet:** no hay ningún commit aún

**Untracked files:** son archivos que git no está siguiendo

**(use "git add <file>..." to include in what will be committed)**

**archivo1.txt**

**nothing added to commit but untracked files present (use "git add" to track)**

* Para seleccionar los archivos que git deba seguir debemos colocar: git add + espacio + nombre de archivo con extensión, podemos colocar expresión regular como \*.txt lo que indicará que todos los archivos txt se anexarán, o un punto, el punto lo que hará es agregar absolutamente todos los archivos que aparezcan en el listado, este último es una mala práctica, ya que se nos pueden ir archivos que no queramos subir, que se nos olvide acciones realizadas o archivos binarios grandes con imágenes etc, en tal caso se recomienda agregar archivos por nombres para agregar a la etapa de staged.

**git add archivo1.txt**

**nos muestra lo siguiente:**

**On branch master:** seguimos en la rama master

**No commits yet:** no tenemos ningún commit

**Changes to be committed:** se encuentran cambios para ser comprometidos

**(use "git rm --cached <file>..." to unstaged)**

**new file: archivo1.txt**

**Nota: los archivos que se encuentren en verde, son archivos que se encuentran en la etapa de staged y listos para ser comprometidos**

* Creo otro archivo archivo2.txt, le agrego información, guardo, voy a git bash y hago git status, me dice que hay dos archivos en la etaba de staged, vuelvo a modificar el archivo y aparece los dos archivos en la etapa de stageds pero uno con cambios pendiente por agregar, si hacemos commit allí, no vamos a agregar los cambios de archivo2.txt.

**On branch master**

**No commits yet**

**Changes to be committed:**

**(use "git rm --cached <file>..." to unstaged)**

**new file: archivo1.txt**

**new file: archivo2.txt**

**Changes not stagedd for commit:**

**(use "git add <file>..." to update what will be committed)**

**(use "git restore <file>..." to discard changes in working directory)**

**modified: archivo2.txt**

Nota: lo que pasamos a la etapa de staged no son los archivos mismos si no las modificaciones de los mismos.

* Para hacer commit(comprometer), la forma recomendada es la siguiente:

**git commit –m “Commit inicial”**

**[master (root-commit) 8c175fa] Commit inicial**

**2 files changed, 2 insertions(+)**

**create mode 100644 archivo1.txt**

**create mode 100644 archivo2.txt**

**$ git status**

**On branch master: seguimos en la rama de main**

**nothing to commit, working tree clean: ya no hay nada disponible para hacer commit**

* Para eliminar archivos dentro del directorio

**rm archivo2.txt**

**git status**

**git add archivo2.txt**

**git commit –m “Eliminando archivo 2”**

* Para ahorrar pasos podemos escribir

**git rm archivo1.txt**

* Para deshacer el cambio

**git restore --staged archivo1.txt**

* Para deshacer los cambios y recuperar el archivo dañado

**git restore archivo1.txt**

* Para mover archivos o cambiar su nombre, mv + nombre actual + nombre nuevo:

**mv archivo1.txt archivo.txt**

**git add archivo1.txt archivo.txt**

**$ git status**

**On branch master**

**Changes to be committed:**

**(use "git restore --stagedd <file>..." to unstaged)**

**renamed: archivo1.txt -> archivo.txt**

* **Hago el commit:**

**$ git commit -m "Renombrando Archivo"**

**[master afcad4a] Renombrando Archivo**

**1 file changed, 0 insertions(+), 0 deletions(-)**

**rename archivo1.txt => archivo.txt (100%)**

* Para hacerlo directamente:

**git mv archivo.txt archivo1.txt**

**git status**

**$ git commit -m "Devolviendo el nombre del archivo"**

**[master 1877801] Devolviendo el nombre del archivo**

**1 file changed, 0 insertions(+), 0 deletions(-)**

**rename archivo.txt => archivo1.txt (100%)**

* Para ignorar archivos de configuración y que no sean incluidos en el repositorio de git, esto se hace ya que a veces vamos a tener archivos de configuración que sean específicos solo para nuestra máquina, como por ejemplo, podría ser variables de entorno, supongamos que estamos trabajando con una base de datos y esta va a estar instalada en local, en este caso los usuarios, contraseñas y cualquier acceso van a ser completamente diferente al de producción, en este caso, queremos tener ese archivo almacenado dentro de nuestra máquina pero que este no se suba por algún error al repositorio y queremos que sea configurable para que cuando la aplicación se despliegue a producción, solamente una persona tenga acceso a esas variables de entorno, las cuales servirán para configurar la aplicación y que esta se conecte finalmente con la base de datos de producción:

Creo un archivo de variables de entorno, .env allí van a estar las variables de usuario y contraseña, creo otro archivo .gitignore, a ese archivo, le agrego los archivos y carpetas que deseo que no se suban, al dar git status, me doy cuenta que solo me aparece .gitignore, lo agrego y hago commit inmediatamente.

**git add .gitignore**

**git commit -m "Agregando archivo .gitignore"**

**[master dfa1cf2] Agregando archivo .gitignore**

**1 file changed, 1 insertion(+)**

**create mode 100644 .gitignore**

* Para verificar el estado de todo al mismo tiempo:

**$ git status -s**

**M archivo1.txt: Significa que fue modificado.**

**?? archivo2.txt: Significa que no se ha agregado para hacerle seguimiento.**

* Agrego los archivos al área de staged:

**git add \*.txt**

**$ git status -s**

**M archivo1.txt: Archivo modificado**

**A archivo2.txt: Archivo agregado**

**$ git commit -m "Mostrando status corto"**

**[master 471c04e] Mostrando status corto**

**2 files changed, 2 insertions(+)**

**create mode 100644 archivo2.txt**

* Para ver los cambios a realizar y que aún no han pasado a la etapa de staged:

**git diff**

**diff --git a/archivo2.txt b/archivo2.txt**

**index 5a1832d..a72376a 100644**

**--- a/archivo2.txt**

**+++ b/archivo2.txt**

**@@ -1 +1,3 @@ : Muestra las versiones del archivo**

**-# Chanchito feliz : Muestra que se eliminó esta línea**

**\ No newline at end of file**

**+# Chanchito feliz : Muestra que se agregaron estas tres líneas**

**+# Chanchito triste**

**+# Felipe**

**\ No newline at end of file**

* Para ver los cambios que se encuentran en la etapa de staged:

**$ git diff --staged**

**diff --git a/archivo2.txt b/archivo2.txt**

**index 5a1832d..a72376a 100644**

**--- a/archivo2.txt**

**+++ b/archivo2.txt**

**@@ -1 +1,3 @@**

**-# Chanchito feliz**

**\ No newline at end of file**

**+# Chanchito feliz**

**+# Chanchito triste**

**+# Felipe**

**\ No newline at end of file**

* Para revisar el historial del repositorio:

**git log : Muestra el historial con más información**

**git log --oneline : Muestra el historial resumido con el hash más corto**

**471c04e (HEAD -> master) Mostrando status corto**

**dfa1cf2 Agregando archivo .gitignore**

**1877801 Devolviendo el nombre del archivo**

**afcad4a Renombrando Archivo**

**455546a Eliminando archivo 2**

**8657c5f Agregando un chanchito feliz**

**8c175fa Commit inicial**

* Para crear otra rama:

**git status**

**On branch master**

**Changes to be committed:**

**(use "git restore --staged <file>..." to unstage)**

**modified: archivo2.txt**

* Restauro el archivo: archivo2.txt

**git restore --staged archivo2.txt**

* Ejecuto nuevamente git status:

**git status**

**On branch master**

**Changes not staged for commit:**

**(use "git add <file>..." to update what will be committed)**

**(use "git restore <file>..." to discard changes in working directory)**

**modified: archivo2.txt**

**no changes added to commit (use "git add" and/or "git commit -a")**

* Descarto los cambios:

**git restore archivo2.txt**

* Verifico en que rama me encuentro:

**$ git branch**

**\* master**

* Para crear la rama:

**$ git checkout -b ramab**

**Switched to a new branch 'ramab'**

**git branch**

**master**

**\* ramab**

* Actualizo archivo2.txt:

**git status**

**On branch ramab**

**Changes not staged for commit:**

**(use "git add <file>..." to update what will be committed)**

**(use "git restore <file>..." to discard changes in working directory)**

**modified: archivo2.txt**

**no changes added to commit (use "git add" and/or "git commit -a")**

**$ git add archivo2.txt**

**$ git commit -m "Actualizando archivo 2"**

**[ramab 93b5202] Actualizando archivo 2**

**1 file changed, 3 insertions(+), 1 deletion(-)**

**$ git log --oneline**

**93b5202 (HEAD -> ramab) Actualizando archivo 2: Dice que me encuentro en la última parte de ramab**

**471c04e (master) Mostrando status corto : dice también que hay una rama master**

**dfa1cf2 Agregando archivo .gitignore**

**1877801 Devolviendo el nombre del archivo**

**afcad4a Renombrando Archivo**

**455546a Eliminando archivo 2**

**8657c5f Agregando un chanchito feliz**

**8c175fa Commit inicial**

* Ver el contenido del archivo2.txt:

**$ cat archivo2.txt**

**# Chanchito feliz**

**# Chanchito feliz**

* Me cambio a la rama master:

**$ git checkout master**

**Switched to branch 'master'**

* Ver el contenido del archivo2.txt:

**$ cat archivo2.txt**

**# Chanchito feliz**

**Nota: de esta manera podemos evidenciar varias versiones del código que están siendo trabajadas, en la ramab ha tenido dos cambios y en master solo uno**

* Para traer los cambios de ramab a master, me paro en master

**$ git merge ramab**

**Updating 471c04e..93b5202**

**Fast-forward**

**archivo2.txt | 4 +++-**

**1 file changed, 3 insertions(+), 1 deletion(-)**

**$ cat archivo2.txt**

**# Chanchito feliz**

**# Chanchito feliz**

**Nota: De esta manera cada programador se crea su rama. Implementa la funcionalidad y la mezcla con rama master o main.**

* Para indicar el servidor remoto para subir los cambios a GitHub:

git remote add origin https://github.com/nschurmann/miweb.git : aquí apuntas tu repositorio

en primer lugar, con add origin le decimos de donde obtiene los datos y la url es el destino, donde debe almacenarlos

* Para subir los cambios a GitHub:

git push –u origin master

Nota: los cambios se hacen respecto a la rama con la cual estamos trabajando, sin embargo esta rama no esta creada en el repositorio, por lo que debemos indicarle que queremos crearla, para hacer esto colocamos –u, seguido del origen y finalmente el nombre git push –u origin master

Ver final del video para configurar esta parte: https://www.youtube.com/watch?v=VdGzPZ31ts8

Merge se comporta de dos maneras diferentes dependiendo de la configuración del grafo

git merge picard: conseguimos todos los commit

Reset sirve para miweb una rama a donde yo quiere y la condición es que tenga el puentero head

Cambiamos a la rama 200s

Git checout 2000s

git checkout 2000s Switched to branch '2000s'

nano lp05.md

git add lp05.md

git commit -m "Modifico libro 5 en 2000s para causar conflicto"

git merge 2000s

para abortar el merge: git merge --abort

preguntar al repo como estas : git status

git log --graph : ver las ramas

git stash

$ git stash pop : saca el ultimo stash

Auto-merging lp05.md

CONFLICT (content): Merge conflict in lp05.md

On branch 2000s

Unmerged paths:

(use "git restore --staged <file>..." to unstage)

(use "git add <file>..." to mark resolution)

both modified: lp05.md

no changes added to commit (use "git add" and/or "git commit -a")

The stash entry is kept in case you need it again.

$ git add lp05.md

$ git commit -m "Termino resumen del libro 5"

[2000s 58cfa0d] Termino resumen del libro 5

1 file changed, 1 insertion(+)

Clonar repositorio git clone con la url del repositorio

Abrimos una terminal mas

https y copis la url del repositorio

para no pisar lo de los demás debo descargarme los cambios del servidor para no dañar el de los demás

git pull origin

se ha hecho un merge automáticamente de la rama master con la rama que se bajó

hace fetch y git merge

un push forzado es git push –f

git checkout master

git push

git push –u origin master

pide contraseña

ir a mi usuario settings, developper settings

generar un token nuevo, pedirá la contraseña de github, y en token se pone un nombre cualquiera

expiración si se quiere que caduque o no y el escode, los permisos

generate token

y esa es la contraseña para hacer los push

nos cambiamos a las ramas que queramos subir y

git push –u origin 90S etc

fork en mi cuenta, vscode

en mi cuenta hago cambio y subirlos

es como si hubiera hecho un clone

y ahora quiero que la aportación se incorpore a vscode

pull recuest es un merge entre repositorios

hanks son las diferencias o cosas que cambian