**CONFIGURACION DE GIT Y GITHUB**

1.- Crearse una cuenta en github.com

2.- Descargando la aplicación de escritorio de github

<https://desktop.github.com/> (el uso de la aplicación de escritorio es completamente opcional)

Logeamos nuestra cuenta para que así podamos vincular nuestra cuenta con nuestro equipo.

3.- Despues descargamos e instalamos GitBash que es nuestra consola tipo Linux.

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

4.- Para verificar con que datos estamos logeados usamos el código: “git config --list --show-origin”, de esa forma hasta abajo nos mostrara como es que estamos logeados.

5.- También podemos configurar de manera manual nuestro perfil, para que de esta forma cuando se realicen modificaciones sepamos quienes lo hicieron, por lo tan para configurar nuestros identificadores del correo y nombre usamos los códigos:

Para el nombre de usuario o user name: git config --global user.name “Cristian Alonso Palma Sifuentes” (Que en mi caso así he crado en mi cuenta de Git Hub con mi nombre)

Y para el correo: git config --global user.email “[prof\_cristianpalma@ajalpan.tecnm.mx](mailto:prof_cristianpalma@ajalpan.tecnm.mx)”

6.- Con esto hemos configurado nuestro software de git.

7.- Git es la consola para los códigos y gitHub es el visualizador para git, básicamente git es como el backend y github como nuestro frontend.

8.- Si nunca hemos configurado el enlace entre git y github, github no sabe que computadora estoy utilizando, no podrá vincularse de forma correcta, es por ello que debemos de crear un puente para que puedan comunicarse entre si, el puente se crea mediante un protocolo de comunicación llamado “SSH”, entonces en git crearemos unas “”llaves”” para el SSH y así poder permitirle la entrada y vinculación entre la página de github y mi equipo físico, por lo tanto en la consola de git bash, usamos el código: ssh-keygen -t rsa -b 4096 -C "prof\_cristianpalma@ajalpan.tecnm.mx".

9.- Despues nos solicitara crear un passphrase, que es básicamente una contraseña, un passphrase (frase de contraseña) puede ser cualquier palabra, frase, cualquier conjunto de letras, o incluso una oración, de hasta 50 caracteres de longitud.

10.- Una vez creado, podemos entrar en la ruta que nos indica donde se creó, podemos entrar a la ruta para poder verlo.

11.- Ahora ya que tenemos creada nuestra llave, nos vamos a nuestra cuenta de github para dársela, entramos a settings, en el lado izquierdo está el apartado de “SSH and GPG keys” en este apartado podremos dar las llaves que tengamos de diferentes computadoras que pudiéramos tener, para trabajar con una sola cuenta de github, para así podre trabajar desde cualquier equipo, damos entonces click en “New SSH key” y en titulo podemos poner el nombre del equipo a donde pertenece la llave, o cualquier otro nombre que querramos, en mi caso puse LAPHP.

12.- En el siguiente apartado “Key” tenemos que insertar el archivo “id\_rsa” que creamos hace unos momentos, para poderlo insertar necesitamos del comando “clip” que realiza la función como de “copy” por lo tanto vamos a clipear el contenido de ese archivo hacia nuestro github, entonces en nuestra línea de comando escribimos: clip < ~/.ssh/id\_rsa.pub

13.- Con eso estonces el archivo se ha copiado y podemos pegarlo en la casilla de “key” simplemente con un click derecho y pastem para finalizar simplemente le damos en “Add SSH key”.

14.- Obviamente esta lleve no debe de ser pasada a nadie, es privada, puesto que si lo hacen otros usuarios modificarían su github como si fueran los dueños, pero también recuerden que esa es la llave del equipo de cada uno de ustedes.

**COMANDOS BASICOS EN GIT Y GITHUB**

1.- En GitHub creamos un repositorio, que básicamente es una carpeta en la nube.

2.- Nos vamos a GitHub y le damos en crear un nuevo repositorio, de nombre podemos ponerle el que queramos en mi caso seria “ProyectoEjemplo2025”, le agregamos una descripción cualquiera, lo dejamos como público, ya que para crearlo como privado hay que tener una cuenta de pago, activamos la casilla de “Add a README file”, y por ultimo creamos el repositorio.

3.- Observamos entonces que se ha creado en la web nuestra carpeta con el nombre y descripción que pusimos anteriormente.

4.- Ahora para poder descargar esta carpeta de mi repositorio a mi equipo físico, debemos de clonar toda la carpeta, nos dirigimos a CODE, después en la pestaña de SSH y click en el botón de copiar.

5.- En el escritorio o en el lugar donde queramos almacenar nuestro clon, creamos una carpeta, y dentro de dicha carpeta damos click derecho y ejecutamos nuestro Git Bash. NOTA: Si no nos encontramos en la carpeta y git bash fue ejecutado en otra posición, nos podemos mover con el código “cd nombre de la carpeta/”, pero lo más recomendable es ejecutar el git bash directamente en la carpeta.

6.- Ahora en git bash escribimos git clone, y posteriormente pegamos el código copiado, quedando: git clone git@github.com:ProfCristianPalma/ProyectoX.git

7.- Despues de haber ejecutado, esperamos y git nos pregunta que si queremos continuar con la conexión, solo le decimos que si. Si ingresamos contraseña nos solicitara que ingresemos la contraeña de frase. Una vez ingresado, esperamos a que termine de clonar.

8.- NOTA: Si quieren evitar que se les solcite la frase de contraseña cada vez que hagan algo, en git bash ejecutan el siguiente código: ssh-keygen –p

9.- Despues solo dan enter (para confirmar el archivo ssh), escribe la vieja contraseña, dan enter y después donde les solicite ingresar una nueva contraseña y confirmarla dejar en blanco y con eso se remueve.

10.- Entro en la carpeta que se ha creado y observo que ahí se encuentra mi archivo Readme, puedo modificarlo, primero abriéndolo, en mi caso sería con visual code, ya abierto lo modifico escribiendo algo tan sencillo como “Autor: Cristian Alonso Palma Sifuentes”.

11.- Observamos del lado izquiero, que en mi icono de Source Control, se ha activado una notificación, al darle click me muestra los cambios que he realizado, siendo estos, las modificaciones que tendría en mi proyecto local, las cuales son ya cosas diferentes a las que tendría en mi repositorio en la nube.

12.- Ahora lo que queremos es subir las modificaciones que hemos realizado, nuevamente a la nube, por lo cual primero en git bash nos movemos dentro de la carpeta donde se encuentra el archivo, ya sea cerrando el git bash y ejecutándolo en la carpeta donde se encuentra o simplemente escribiendo “cd nombre de la carpeta/” (Ejemplo: cd ProyectoX/).

13.- De esta forma ya nos encontramos en el repositorio de git, pues nos encontramos en la línea “main” que eso lo podemos observar tanto en git bash, como en la web de GitHub.

14.- En este caso al usar Visual Code, gracias al plugin, sabemos que realizamos modificaciones, pero en caso de que no usaramos visual code, para poder saber si tenemos modificaciones, en git bash escribimos: git status. De esta forma nos muestra que es lo que hemos modifcado.

15.- Para guardar las modificaciones que hemos realizado, usamos la codificación: git commit -am “Un mensaje”, en el mensaje debemos de colocar qué fue lo que modificamos, para que de esta forma, otros usuarios sepan que fue lo que hicimos en esta modificación (también llamado instantánea, fotografía, etc.), no necesariamente debe de ser una explicación larga, si no algo significativo para que sepan que se hizo. En mi caso quedaría como: git commit -am “Agregue autor del archivo”. Con eso observamos que en visual cod, en nuestro Source Code (Control de código fuente) ha desaparecido la notificación, pues ya hemos realizado los cambios.

16.- Con esto únicamente hemos guardado los cambios de forma local, aun no se verán reflejados en nuestro repositorio en la nube.

17.- Si queremos guardar los cambios en la nube, escribimos la codificación: git push origin main

18.- Esperamos a que se termine de cargar, y en la web de github podemos dar refresh y observamos que ya están las modificaciones, incluso se ha sumado un commit.

19.- Si ingresamos a la sección de commit, observamos el historial, podemos ingresar al más reciente, y veremos cuáles son las modificaciones que se realizaron y en la parte superior derecha está el hash que nos serviría para saber quién y que realizo.

20.- Ahora vamos a borrar esa modificación de autor, porque quizás no debía de ir ese dato, lo borramos entonces desde visual code.

21.- Guardamos de manera local con: git commit –am “mensaje”

22.- Ahora lo subimos a la nube con: git push origin main

23.- Ya no aparece esa línea en el readme, y también se ha creado un commit mas, podemos ingresar para visualizar los cambios.

24.-En nuestra carpeta, donde tenemos nuestro achivo README, podemos crear más archivos, nos vamos a visual code y pulsamos CTRL+N, para crear una nueva pestaña.

25.- En la nueva pestaña creada, escribimos:

int main (void)

{

printf("Prueba en GitHub, ¿Que podria malir sal?");

}

26.- CTRL+S para guardar, verificamos que la ruta de guardado sea la correcta donde está el proyecto, lo nombramos como por ejemplo: main.c

27.- Ahora si nos vamos a git bash, y escribimos: git status. Observaremos que nos muestra que el archivo creado está en rojo, eso quiere decir que solo lo hemos creado de forma local, y dicho archivo no se encuentra en el repositorio de la nube, si queremos guárdalo en el repositorio, para tenerlo guardado y también que exista en el historial, en git bash escribimos:

Git add “nombre del archivo”: en mi caso queda como: git add main.c

28.- Ya que lo agregamos al trabajo local, ahora creamos el commit: git commit –am “Agregue un archivo main”

29.- Y ahora lo subimos al repositorio: git push origin main

30.- Actualizamos github.com, y vemos que ya se a cargado el archivo a nuestro repositorio.

31.- Cada archivo tendrá una historia distinta, si cliqueamos en cada historia, será independiente para cada archivo. A menos que ambos archivos sean involucrados en un mismo commit, en la historia de igual manera serian involucrados.

32.- Ahora que git pasaría si otro usuario hace modificaciones en el repositorio en la nube, nos vamos al archivo README y le damos click en la opción EDIT THIS FILE, y hacemos modificaciones en el archivo, por ejemplo la línea 3 la dejamos en blanco y sobre la 4 escribimos algo como: Curso ITSAO.

33.- Nos vamos a la parte de abajo y en COMMIT CHANGES, anotamos el cambio que realizamos (poner cualquier cosa). Luego pulsamos el botón de commit changes.

34.- Observamos que esa nueva línea se ve reflejada, pero si nos vamos al archivo readme.md abierto en visual code, dichos cambios no se reflejan.

35.- Para actualizarlo entonces, es decir que los cambios que se realizaron en la nube, se vean reflejados en el proyecto que tenemos almacenado en nuestro equipo físico, no debemos de clonarlo nuevamente, debemos de realizar una descarga de actualización.

36.- En git bash escribimos: git fetch; este código lo que hace es descargar toda la metadata o cambios que se realizaron en el repositorio, pero solo lo descarga, no actualiza mis ficheros aún.

37.- Una vez que ya tenemos descargada la metadata, ahora si actualizamos nuestros archivos de nuestro proyecto local con el código: git pull origin main.

38.- NOTA: La diferencia entre un push y un pull, es que el push empuja o subir los datos hacia el repositorio en la nube, y el pull es para jalar o descargar los datos hacia nuestro equipo físico.

39.- Obervamos en visual code que los cambios ya se ven reflejados.

40.- RESUMEN: Cada vez que realizamos cambios en nuestro equipo físico, debemos de subir dichos cambios para que los colaboradores puedan trabajar con esas actualizaciones, de igual forma debemos de estar al pendiente de cambios que hayan hecho los demás colaboradores para actualizar nuestros archivos locales y así trabajar a la par.

**BRANCHES**

1.- Vamos a crear un branch, para de esa forma trabajar sobre otro brazo, sin afectar a la línea del tiempo original, de esa manera podríamos trabajar de forma asíncrona con uno o más colaboradres, sin que nuestros cambios nos afecten entre nosotros.

2.- En la terminal (podemos usar la de powershell o dentro de visual code abrir una de git bash) escribimos entonces: git branch “nombre del branch” por ejemplo: git branch Linea1.

3.- De esa forma ya hemos creado ese branch, pero solamente la ha creado, aun no estamos posicionados en ese branch, para movernos de branch ya que en este momento estamos en el “main”, escribimos en la consola: git checkout “nombre del branch”, que en mi caso seria: git checkout Linea1

4.- Observamos entonces en nuestra ruta de la consola, que nos ha movido del “main” hacia nuestro branch.

5.- Ahora entonces, ya estamos en nuestro propio branch, podemos hacer lo que queramos sin que nuestras actualizaciones y/o cambios afecten a los demás.

6.- Checamos previamente en github.com que de momento tenemos solo 1 branch. Modificamos un poco nuestro archivo readme:

# Branch 1 (Linea 1) “después más texto que queramos”, después realizamos el commit y lo subimos:

git commit –am “Modificacion colaborador 1”

git push origin Linea1

7.- Observamos que cambiamos el “origin main” por “origin Linea1”, porque ahora esa es la ruta de origen, donde estamos trabajando.

8.- En nuestro poyecto ya tenemos entonces 2 branches, y en cada branch tienen su propia línea de proyecto.

9.- Ahora nos movemos de branch nuevamente, nos regresamos al branch principal, tecleando: git checkout main.

10.- Lo que podemos observar es que en la ruta de la consola nuevamente nos cambió de ruta en donde estamos posicionados, y también que las modificaciones que habíamos hecho en el archivo readme no esta, ya que fueron modificados en otro branch, uno que tiene su propia línea de trabajo que es completamente independiente a la que tenemos activa actualmente.

11.- Vamos a crear un nuevo branch, pero vamos a crearlo de tal manera que al mismo tiempo que lo creamos, nos reubique inmediatamente en el, con el código: git checkout -B “nombre del branch”, que en mi caso seria: git checkout -B Linea2

12.- Modificamos nuevamente nuestro archivo readme:

# Branch 2 (Linea 2) “después más texto que queramos”

13.- Guardamos en el local y lo subimos:

git commit –am “Linea 2”

git push origin Linea2

14.- Nos vamos a github.com y observamos que se ha subido y creado nuestro segundo branch.

**GIT MERGE Y PULL REQUEST**

1.- Una vez que se han hecho las actualizaciones en alguno de los branchs, y estos ya son correctos, ahora queremos incorporar dichos cambios al main, para ello vamos a utilizar un “pull request”, de esa forma, pasar los cambios o actualizaciones que hicimos en algunos de los branchs al main.

2.- Para ello, alguien primero debe de revisar nuestros cambios, ya que por lo regular trabajamos en equipo, entonces el encargado de hacer el pull request, primero debe de revisar lo que va a aceptar para saber si es correcto o no.

3.- Entramos a github.com

4.- Normalmente, nos aparece un mensaje donde nos indica si en algún branch hemos hecho algunos pushes, entonces tendríamos que darle al botón verde que dice “COMPARE & PULL REQUEST”, para analizar dichos cambios, en caso que no aparezca dicho cuadro, nos movemos al branch que queremos verificar las modificaciones, luego damos click en el apartado de “contribute” y posteriormente al botón “open pull request”.

5.- En la parte de arriba, observamos que los cambios se van a realizar de nuestro branch, hacia nuestro main.

6.- Cambiamos el titulo por ejemplo: Modificaciones Colaborar 1, también podemos colocarle un texto con una descripción más amplia. Después damos click en “create pull reques”

7.- Ahora podemos observar todos los commits que se han hecho y en la pestaña de “files changed” podremos observar todos los cambios que se realizaron.

8.- Podríamos aprobar los cambios en el botón de “review changes” desplegandolo y seleccionando la opción de cambio, o si necesitamos que realice otros cambios, o modifique algo, pulsar en un signo en alguna o algunas líneas en especifico, pulsamos el “+” dentro de las líneas o línea de texto que queramos que modifique, dejando un comentario de que cambios queremos que realice. Por ejemplo acentos, que quite una línea, ponga más líneas, etc.

9.- Si le queremos darle un ejemplo de cómo queremos que haga los cambios, damos click en la parte de arriba a la izquierda de la “H” en el botón “Insert a suggestion”. De esa forma podemos ejemplificarle más como realizar los cambios, básicamente son sugerencias, y pulsamos en “add single comment”.

10.- De esa forma ahora nos muestra entonces la línea original, y la línea que estamos sugiriendo de cómo se modifique,

11.- Ahora supongamos que recibimos ese mensaje de sugerencia de cambios, damos click en el botón de “commit suggestion”, estamos de acuerdo con los cambios, escribimos el nombre de nuestro commit y después pulsamos “commit changes”

12.- Se han actualizado los cambios. Ahora en la parte superior izquierda en “conversation” se guarda ese historial se la conversación que tuvimos con el revisor o con el colaborar, y los cambios que se pudieron haber sugerido, y lo que se fue realizando, incluso nos aparece el commit que se aplicó en la nube.

14.- Los cambios que hicimos, de momento solo están en la nube, por lo tanto debemos de descargarlo al equipo, nos vamos a nuestra consola, si no nos encontramos en el branch deseado, nos movemos con: git checkout.

15.- Si ya nos encontramos en el branch deseado, realizamos la descarga

git fetch

git pull origin Linea1

16.- Ahora si ya tenemos los archivos actualizados también en nuestro equipo físico y también en la nube.

17.- Una vez hechos todas las modificaciones, o si no existieran modificaciones, en el apartado de “Conversation”, en la parte de abajo, nosotros o el encargado, o el revisor, nos hara un “merge”, pulsando el botón “Merge pull reques”, el mensaje lo podemos dejar por defecto, después pulsamos “confirm merge”, podemos eliminar el brach si ya no lo vamos a utilizar, pulsando el botón de DELETE BRANCH, o simplemente dejarlo así, también podemos borrarlo con línea de comando.

18.- Si queremos saber cuáles son las ramas que existen es con el código: git branch. Tambien si queremos saber las ramas que existen en local, como las de seguimiento remoto, escribimos: git branch -a.

19.- Para cambiar el nombre de una rama:

git branch -m VIEJO-NOMBRE-RAMA NUEVO-NOMBRE-RAMA

20.- Para eliminar una rama:

git branch -D RAMA-A-ELIMINAR

21.- Nos movemos a la rama main, observamos que aún no se reflejan en el local, los cambios en el main, que fusionamos con el branch, entonces actualizamos en el local:

git fetch

git pull origin main

22.- Ya realizadas las actualizaciones, ahora vamos a eliminar la rama 1, pues ya no se necesitara (en caso de que asi sea): git branch -d nombre de la rama, en mi caso: git -d Linea1.

23.- De esta forma solo la eliminamos del local, si queremos eliminarla también de la nube, nos vamos al apartado de “branches”, y darle click al botón de delete, también podemos observar que dice que el branch se re aplico un merge.

24.- Recordemos que tenemos dos colaboradores, ahora analizaremos los cambios del colaborador 2, en github.com nos movemos a la otra rama, contribute y open pull request.

25.- Observamos un mensaje en rojo que nos dice que NO PODEMOS HACER DE FORMA AUTOMATICA EL MERGE, el merge no lo podemos hacer, pero el pull request si podemos hacerlo, entonces colocamos un encabezado y un texto, por ultimo “Create pull request”.

26.- Una vez creado el pull request, observamos en la siguiente pantalla, que nos dice que este branch tiene conflictos por resolver y en que archivos se encuentran dichos conflictos. Eso es porque el contenido de este branch, es distinto al contenido que actualmente tenemos, por eso se genera ese conflicto, y github pues no puede decidir de forma automática que hacer.

27.- Ahora lo que tenemos que hacer es pasar las actualizaciones del main hacia esa rama, en caso que en el main no estuvieran reflejados los cambios, tendríamos que descargarlos y jalarlos, pero en este caso ya debe de estar actualizado.

28.- Ahora nos pasamos al branch, donde queremos mandar esos cambios, lo hacemos con: git checkout.

29.- Ahora haremos un merge de main a Linea2:

git merge main Linea2

30.- Se nos muestran entonces los conflictos que existen, y con visual code podemos resolverlos fácilmente, tenemos 3 opciones para resolverlo: La primera seria para para rechazar lo que esta en el main, o de la rama proveniente, nos guiamos por el “current” que es entonces lo que conservaríamos, la segunda opción es para aceptar y reemplazar lo que tenemos en nuestro branch y modificarlo con el merge origen, la tercera es para conservar ambas modificaciones en caso de que puedan coexistir, la cuarta opción sería para hacer una comparación antes de aceptar.

31.- Supongamos que los dos estarían bien, entonces seleccionamos la tercera opción.

32.- Una vez resuelto el conflicto, observamos que en el encabezado de línea, nos dice que nuestra rama esta MERGING, quiere decir que estamos resolviendo los conflictos.

33.- Ya que terminamos de resolver los conflictos, realizamos un commit:

git commit -am “Conflictos resueltos”

34.- De esa forma ya desaparece el MERGING, mientras se encuentre el MERGING, no podemos hacer un push, porque el mismo git nos dira que debemos de resolver los conflictos, por eso primero resolvemos y después guardamos.

35.- Ahora lo subimos a la nube:

git push origin Linea2

36.- En github.com, ya no aparece el mensaje de conflictos incluso nos aparece el commit con su código, de lo último que hicimos, resolviendo dichos conflictos.

37.- En el apartado “Files changed”, podemos ver también lo que se ha modificado.

38.- Ahora lo que nos queda, es pasar los cambios que tenemos de ese branch, al main, entonces damos click en “Merge pull request”, después en “confirm merge”, con eso los cambios se han realizado.

38.- Si queremos borrar el branch, pues lo hacemos, o simplemente lo dejamos como esta.

39.- Lo que faltaría es también descargar la actualización del main de la nube al local:

Git checkout main

git fetch

git pull origin main