TAREA MOCHILA VIAJERO 1

a. ¿Quién inventó el sistema de control de versión Git y por qué?

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

FUENTE: https://es.wikipedia.org/wiki/Git

b. ¿A quién pertenece actualmente Github y por qué?

El 1 de junio de 2018, Microsoft compró GitHub por la cantidad de 7500 millones de dólares.

c. ¿Hay otra forma que no sea la terminal para trabajar con Github?

Se puede trabajar con Github desde la web https://github.com/

TAREA MOCHILA VIAJERO 2

$ git -- version	Para saber qué versión de Git tenemos.
$ git help	Proporciona ayuda/orientación sobre el comando que indiquemos.
$ clear	Limpia la terminal.
$ pwd	Para saber en qué directorio estamos ubicados.
$ mkdir	Para crear un directorio/carpeta. **$ mkdir "Carpeta 4"** Al poner el nombre del directorio entre comillas "" es posible colocar un espacio entre palabras diferentes pero que sean un mismo directorio. **$ mkdir Carpeta1 Carpeta2 Carpeta3** Crea las carpetas "Carpeta1", "Carpeta2" y "Carpeta3" dentro del directorio en el cual estamos ubicados en un solo comando.
$ cd nombre-carpeta	Para ingresar a la carpeta "nombre_carpeta". Si indico **$ cd r*** irá a la única carpeta que se inicialice con la letra "r" en ese directorio.
$ cd ..	Para salir de la carpeta en la que nos encontramos y subir un nivel de directorio. Si indico por ejemplo **$ cd ../Carpeta2** sube un nivel y entra a a la Carpeta2.
$ touch nombre-archivo.txt	Para crear un archivo. Para crear un archivo dentro de otro directorio diferente al cual estamos ubicados, lo hacemos indicando la ruta, por ejemplo: **$ touch Carpeta3/archivo.txt "Carpeta 4"/archivo.txt "Carpeta 5"/archivo.txt**
$ nano archivo.txt	Ingresa a un editor de texto. Para salir de ese editor primero hay que guardar con **Ctrl+O** (pregunta si quiero cambiar el nombre del archivo) y luego salir con **Ctrl+X.** De esta forma podemos ingresar a este editor desde otro directorio: **$ nano ../"Carpeta 2"/archivo.txt**
$ cat archivo.txt	Permite ver el contenido del archivo.
$ cp nombre-archivo.txt	Permite copiar un archivo que está en una carpeta y copiarlo en otra; por ejemplo: **$ cp archivo2.txt ../"Carpeta 5"** ó **$ cp ./repaso.txt ../servidores/**
$ rmdir Carpeta	Elimina el directorio "Carpeta".
$ rm archivo.txt	Elimina el archivo.
$ rm -r nombre-directorio	Elimina el directorio y todos los archivos que contiene.
$ rm -rf nombre	Elimina el directorio y todos los archivos que contiene en forma forzada.

$ ls	Lista el contenido de un directorio.
$ ls -R	Muestra directorios + lo que tienen adentro.
$ ls -Ra	Muestra directorios + lo que tienen adentro + los archivos ocultos.
$ mv	Tiene dos funciones: renombrar (si el destino no existe) y mover (si el destino existe). Ejemplos: **$ mv ./"historia de la contabilidad"/repaso.png ./"historia de la contabilidad"/repaso.txt** **$ mv ./"historia de la informatica ./repaso_clase_1y2/** <esta última barra significa que "lo ponga adentro" **$ mv "Carpeta 5"/Carpeta1 ./** ← El punto barra ./ es para decir que quiero estar ubicado ahí
	Ruta relativa y absoluta: ¿cómo dar un comando para que suceda algo en una carpeta diferente a la que estamos? Por ejemplo, estando en Desktop no hace falta entrar a la carpeta dh para crear una carpeta repaso_clase_1y2 ahí adentro, se puede hacer así: **$ mkdir dh/repaso_clase_1y2** **$ mkdir ./dh/repaso_clase_1y2** **$ mkdir c:/dh/repaso_clase_1y2**
$ git init	Inicializa un repositorio local vacío en la carpeta donde tenemos los archivos del proyecto y estamos ubicados.
$ git config user.name "mi-usuario"	Permite agregar nombre de usuario con el que se firma; para corroborar si se hizo bien escribir **$ git config user.name** (sin nombre) y ahí debería verse el texto ingresado con anterioridad.
$ git config user.email "miCorreo@email.com"	Para informar el correo electrónico con el que nos registramos en GitHub. Para corroborar si lo hizo bien escribir **$ git config user.email** (sin el mail) y ahí debería verse el texto ingresado con anterioridad.
$ git config --global user.name "mi-usuario" **$ git config --global user.email "miCorreo@email.com"**	Para configurar por única vez usuario y e-mail de modo que cualquier repositorio existente en la computadora tendrá esa misma firma (nombre+email).
git config --global --unset user.name "nombre de usuario" **git config --global --unset user.email email@email.com**	Para eliminar todos los registros que se refieren al usuario y al e-mail.
$ git status	Dice el estado de los archivos y respecto del repositorio, permitiendo el seguimiento. Indica en qué rama estamos, en qué versión del commit y si hay algún commit que enviar.
$ git add nombre-archivo	Indica que queremos agregar el archivo "nombre-archivo".

$ git add .	Indica que queremos agregar todos los archivos presentes en el repositorio, así no hay que agregar uno por uno. Si después de agregado un archivo los modificamos, Git asume que ese archivo es “nuevo” y por lo tanto pasará a ser un archivo que no tiene seguimiento; de esta forma se hace el control de versiones.
$ git restore –stage nombre_archivo	Deshace los cambios que se hicieron a un archivo.
$ git commit -m “acá va un mensaje entre comillas”	Git no commitea directorios, solo archivos. El mensaje entre comillas debe describir de manera resumida el trabajo hecho hasta ese momento.
$ git rm --cached	Mueve los archivos que se indiquen al estado untracked, es decir, los saca del commit; vuelven a un estado anterior.
$ git log	Registra un historial de los cambios del proyecto, es decir, un historial de los commits (indica fecha + autor + mensaje resumen breve del cambio).
$ git log -- raw	Muestra el commint y lo que tiene adentro.
$ git remote add origin https://github.com/ItziarSebedio/DigitalHouse.git	Conecta el repositorio local con el repositorio remoto. El repositorio local no puede estar conectado a más de un repositorio remoto. Si todo está ok la terminal no manda ningún mensaje. Para verificar que está todo ok, escribir el comando **$ git remote -v** y debería aparecer esto: *origin - https://github.com/ItziarSebedio/DigitalHouse.git (fetch)* *origin - https://github.com/ItziarSebedio/DigitalHouse.git (push)*
$ git remote -v	Indica con qué repositorio remoto está conectado el repositorio local.
$ git clone url	Crea una copia exacta en la computadora de todos los archivos existentes en un repositorio remoto. Se ejecuta una sola vez, al descargar los archivos de GitHub y cuando los mismos no estén presentes en la computadora, es decir, para clonar por primera vez un proyecto. Al clonar un repositorio, no hace falta ejecutar git init.
$ git pull origin main	No baja todos los archivos de nuevo, sino que actualiza en el repositorio local aquellos que hayan sufrido cambios y baja nuevos archivos existentes que no tengamos en la computadora.
$ git pull origin nombre-branch	Actualiza el repositorio local según el repositorio remoto, descargando cambios de la branch.
$ git push origin main	Este comando le pide a Git que lleve los archivos del repositorio local al repositorio remoto (para Git este repositorio remoto se llama “origin”) y que los lleve a la rama principal que llama “main”. Se suele decir “pushear” a esta operación. Antes de subir los archivos a GitHub hay que tener todo commiteado. Es decir que a la nube sólo se suben los commits ya cerrados.

	Al ejecutar este comando va a pedir nombre de usuario y la contraseña (que aunque se tipee no se ve en pantalla): este mensaje solicitando esta info puede aparecer en la terminal o en otra ventana emergente, dependiendo del sistema operativo.
$ git push origin nombre_rama	Para pushear un archivo a una rama distinta de la main. Al trabajar nunca vamos a pushear a la rama main porque ahí va solo lo que está validado y funcionando; pusheamos a la rama en la que estamos trabajando.
$ git branch nombre_rama	Crear una nueva rama.
$ git branch ó **git branch --list**	Enumera todas las ramas de tu repositorio.
$ git branch -v	Lista branches con información de los últimos commits.
$ git branch -d nombre-branch	Elimina la rama llamada nombre-branch. Git evita que eliminemos la rama si tiene cambios que aún no se han fusionado con la rama main.
$ git branch -D nombre-branch	Fuerza la eliminación de la rama especificada, incluso si tiene cambios sin fusionar.
$ git checkout	Para salir de la rama en la que estamos y subir a la main.
$ git checkout main	Vuelve a la branch principal main.
$ git checkout nombre_rama	Para moverse a la rama nombre_rama.
$ git checkout -b nombre-rama	En un solo comando se crea una rama y se cambia a ella.
$ git merge	Para fusionar ramas. Sucede donde estemos ubicados, generando que el último commit de esa rama sea el principal. Si queremosfusionar otra rama (nombre.rama) con la main, primero git checkout main y luego git merge nombre.rama
$ git merge nuevaBranch_nombre	Resuelve la unión (merge) entre las branches. Para realizar la unión se debe estar en la branch que debe recibir los cambios.
$ git config --global init.defaultBranch main	Esto es para cambiar el nombre de la rama ”master” para que se llame “main”.