ARQUITECTURA Y SISTEMAS OPERATIVOS

Clase 6 - GIT BASH

Profesora: Lucero, Natalia.

Nombre del grupo: “Carpinchos Programando”

Integrantes del Grupo:

- Aguilera, Mariana.
- Atim, Mercedes.
- Lanatta, Wanda Oriana.
- Lizardo, Florencia.
- Mercado, Nicolás.
- Mori, Ana.
- Mulena, Enzo.
- Rico, Adriel.
- Ríos, Nelson.
- Ríos Garín, Ana Paula.
- Sandoval, Facundo

07/05/24

ACTIVIDADES

Responder las siguientes preguntas: Desarrollar y utilizar comandos vistos (se puede acompañar de capturas).

1. ¿Cómo se crea una carpeta desde Git Bash?
2. ¿Cómo se inician los Repositorios Git?
3. ¿Qué comando hay que utilizar para configurar el usuario (nombre y apellido) y email?
4. ¿Cuál es el comando que permite viajar en el tiempo de los diferentes commit?
5. ¿A qué se llama Rama Master o Main?
6. ¿Cuál es la diferencia entre Rama Auxiliar y Rama Master o Main?
7. ¿Qué comando se utiliza para cambiar de una rama a otra?
8. ¿Se puede modificar la Rama Master?
9. ¿Cuál es el comando para crear una nueva rama?
10. Mencionar 3 funcionalidades que tiene el comando Git Checkout.
11. ¿Cuál es el comando que permite ver de manera gráfica los commit y las ramas creadas?
12. Realizar una captura del último punto realizado, se puede utilizar captura de la actividad n° 4.

DESARROLLO DE LAS ACTIVIDADES

1. Para crear una carpeta (también conocida como directorio) desde Git Bash se utiliza el comando **mkdir.**

   Los pasos son los siguientes:

   a. Navega al directorio donde deseas crear la nueva carpeta utilizando el comando cd: **cd Documentos**

   b. Una vez que estamos en el directorio correcto,se crea una nueva carpeta utilizando el comando **mkdir** seguido del nombre de la carpeta que se desea crear.

   Por ejemplo, si deseas crear una carpeta llamada Proyecto, puedes escribir: **mkdir Proyect**o

2. Inicializar un nuevo repositorio con git init

   git init es el comando que hay que usar para realizar el setup inicial de un repositorio.

   Ejecutar este comando crea el subdirectorio .git en el directory en el que se ejecuta (el directorio del proyecto); no se agregan ni cambian archivos que no sean el subdirectorio .git.

   Dentro del directorio .git se crean todos los metadatos que necesita Git (branch de default, objetos, referencias, archivos template…).

   La mayoría de los comandos de Git están disponibles sólo cuando se ejecutan dentro de un repositorio inicializado.

   Obviamente, es posible inicializar un directorio diferente al actual indicando la ruta:

   $ git init /path/to/project/directory

3. Configurar la información del usuario para todos los repositorios locales

   $ git config --global user.name "[name]"

   Establece el nombre que estará asociado a tus commits

   $ git config --global user.email "[email address]"

   Establece el e-mail que estará asociado a sus commits.

   Para configurar el usuario que va a escribir en la bitácora.

   Esto mostrará datos de la identidad con la que hemos creado el usuario así como otros datos de la máquina con la que estamos trabajando. Con git config podremos configurar git para registrar diferente identidades, por si usamos un ordenador para diferentes desarrolladores o si nos interesa registrar los cambios bajo diferentes nombres. Esto lo podemos realizar mediante los comandos:

   git config --local user.name "Nombre Apellido"

   git config --local user.email "tuemail@ejemplo.com" (para configurar el usuario que va a escribir en la "Bitácora" desde una máquina).

4. **Git log**

El comando git log permite visualizar el historial de los commit, filtrarlo y buscar cambios específicos. Así como git status opera en la working directory y la staging area, git log opera en la history de los commit.
Git log ofrece muchas opciones para personalizar las visualizaciones de los commit y para filtrarlas.

- Con este comando veremos nuestro primer commit.
- será el cuaderno, la "Bitácora" que mostrará en pantalla todos los commits, todos los mensajes).
- Nos aparece el Autor del commit(nombre, apellido y el correo)
- Fecha y hora del sistema y el mensaje que realizamos en nuestro editor de texto.
- Cada commit tiene un número identificador y que permite de manera ordenara realizar cada commit en mi proyecto, de manera que es imposible que se repitan.

5. Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la **rama master**. Con la primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.
   Cuando creamos un repositorio con git init en nuestro proyecto, Git también ha creado un branch de default (cuyo nombre suele ser main o master).
   Por regla general a Main se la considera la rama principal y la raíz de la mayoría de las demás ramas. Lo más habitual es que en main se encuentre el "código definitivo", que luego va a producción, y es la rama en la que se mezclan todas las demás tarde o temprano para dar por finalizada una tarea e incorporarla al producto final.

6. Las ramas auxiliares son punteros ligeros y van a administrar una separación de la rama principal. Una vez que se realizaron los cambios en nuestra rama auxiliar, volvemos a incorporarnos en la rama principal o Master.
   - Git nos permite viajar en una línea del tiempo de nuestro proyecto y realizar cambios en él.
   - Los commit crean vínculos a otras confirmaciones, formando un gráfico del historial del desarrollo, a esto llamaremos rama.
   - Al trabajar en ramas auxiliares, estamos trabajando en un espacio temporal y esto me permite realizar cambios que no necesariamente se van a aplicar en nuestro proyecto. Y también proteger nuestro proyecto en caso de que las modificaciones sean fallidas, evitamos cualquier daño a la rama principal.

7. **git checkout**
   El comando git checkout se usa para cambiar de rama y revisar el contenido de tu directorio de trabajo.

   ```
   $ git switch -c [branch-name]
   ```

   Cambia a la rama especificada y actualiza el directorio activo

8. Los pasos básicos para modificar la **rama master**:
   a. Asegurarse de estar en la rama master con el comando: **git checkout** master.
   b. Realizar las modificaciones necesarias en los archivos.
   c. Agrega los archivos modificados al área de preparación con: **git add <nombre_del_archivo> o git add** : para agregar todos los archivos modificados.
   d. Confirma los cambios con: g**it commit -m "Mensaje del commit".**
   e. Finalmente, sube los cambios a la rama master con: **git push origin master.**

9. $ git branch [branch-name]
   Crea una nueva rama
   RAMA NUEVA
   Para Crear una Rama Nueva utilizaremos el siguiente comando
   Git Checkout -b «nombre de la rama»
   En el momento de crear mi rama auxiliar o nueva rama mi puntero se va a posicionar en mi nueva rama de mi proyecto.
   Una vez creada la rama, añadiremos las modificaciones a nuestro proyecto, cuando ya estén listas nuestras modificaciones.
   Digitaremos Git status.
   Veremos entonces cada una de las modificaciones que hemos realizado.
   Luego commitearemos dichas modificaciones.
   Git add . (agregamos archivo modificado)

10. Comando Git Checkout:

- El comando git checkout me permite borrar archivos modificados .
- No me permite eliminar archivos nuevos.

Otras funciones de Git Checkout
Con Git Checkout «más el número de identificación»
(se pueden copiar sólo los primeros 7 dígitos)
Podremos ver entonces a como estaba nuestro proyecto sin las modificaciones nuevas
Para volver al presente dónde está mi último commit ejecutamos
Git Checkout Master

GIT CHECKOUT

git checkout -- [path del archivo] Elimina un archivo del working tree. (No funciona si es un archivo untracked, ver git clean).
git checkout [branch-name] (Otra opción) Cambiar de rama.
git checkout [codigo del commit] (Detached HEAD) Volver al estado de un commit anterior y revisar el codigo o hacer cambios experimentales (también se puede lograr con el nuevo comando git switch -d [codigo del commit]).
git checkout --orphan [new-branch-name] Crea una nueva rama huérfana. Se mantendrán todos los archivos en el index tree, en caso de que no se necesiten se recomienda el uso de git rm -rf .

para eliminarlos. NOTA: la rama no figurará en la lista de ramas git branch -l hasta no poseer commits.

El comando git checkout se usa para cambiar de rama y revisar el contenido del directorio de trabajo.

En primer lugar, encontramos el comando en *Cambiar de Rama* junto con el comando git branch.

Vemos cómo usarlo para iniciar el seguimiento de ramas con el indicador --track en Hacer Seguimiento a las Ramas.

Lo usamos para reintroducir los conflictos de archivos con --conflict=diff3 en Revisando Los Conflictos.

Entramos en más detalle sobre su relación con git reset en Reiniciar Desmitificado.

Finalmente, examinamos algún detalle de implementación en La CABEZA (HEAD).

11. Git log --oneline --all --graph --decorate

Este comando nos permite ver de manera gráfica (dibujo)de las ramas que se van trabajando por separado y las modificaciones que vamos registrando.

12.

```
[➜  WORKSPACEGIT cd share_repository
[➜  share_repository git:(master) git log --oneline --all --graph --decorate
* c716ba8 (HEAD -> master, origin/master) Cambiando readme
| *   db86353 (origin/ramaWan, ramaWan) Cambiando readme
| |\
| |/
|/|
* | 72b8794 Cambiando readme
* |   c78343a Merge remote-tracking branch 'origin/ramaWan'
|\ \
| | *   d081d11 espacio
| | |\
| |_|/
|/| |
* | | b907e78  Trabajando en equipo
| | * 277a1e2 cambiando readme2
| |/
| * b434778 cambiando readme2
|/
* e887ce9 cambiando readme
* 65f291f cambiando readme
* 04c6025 Cmabiando Readme
* 2320f6d primer commit
~
~
~
~
~
~
~
~
~
~
```