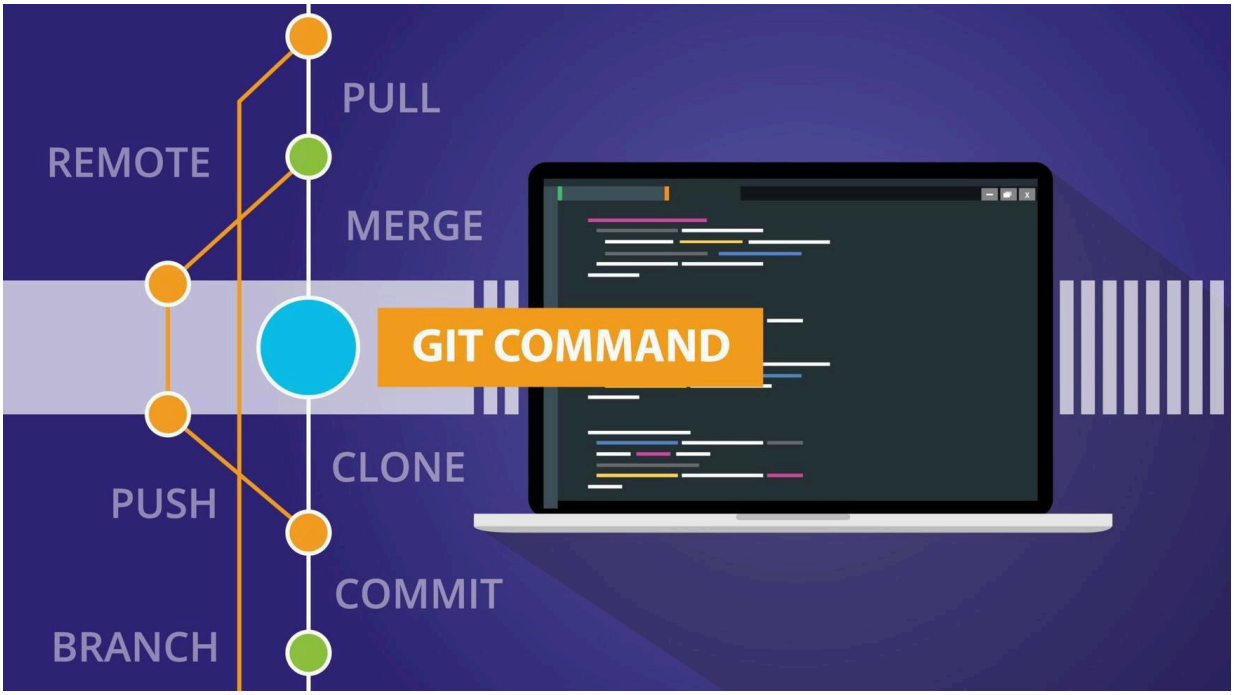
ARQUITECTURA Y SISTEMAS OPERATIVOS

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

14 o 21/05/24 (A confirmar)

ACTIVIDADES

Realizar los siguientes pasos, realizar una captura de pantalla o video de los pasos realizados (cada paso tiene que estar explicado).

1. Crear un repositorio nuevo con el nombre ‘clase 7’.
2. Crear dos carpetas: ‘readme’ y ‘planilla c’. Añadir dos documentos de texto en ambas carpeta, añadir con git add cada modificación y commitear.
3. Aplicar los comandos git checkout y seleccionar dos n° # para volver en el tiempo en el proyecto.
4. Volver al último commit que se realizó anteriormente.
5. Crear una 1ra rama auxiliar: ‘- desarrollo’.
6. Crear tres documentos de texto, añadir y commitear.
7. Crear una 2da rama auxiliar: ‘- prueba’.
8. Crear tres documentos de texto, añadir y comitear.
9. Aplicar el comando que permite ver todas las ramas y commit.
10. Eliminar una rama.
11. Aplicar nuevamente el comando que permite visualizar las ramas y los commit.
12. Generar tres etiquetas en el proyecto.
13. Desarrollar cuál es la importancia de utilizar Git (Visto en la clase n°6). Puede ser acompañado de imágenes o capturas de pantalla.

DESARROLLO DE LAS ACTIVIDADES

1- A continuación , la creación de una rama utilizando " git init" con el nombre de **Clase_7**.

```
→  WORKSPACEGIT git:(master) ✗ cd Clase7
→  Clase7 git:(master) ✗ git init
Initialized empty Git repository in /Users/macbookair/Desktop/WORKSPACEGIT/Clase7/.git/
```

2- Aquí se creó las carpetas en el repositorio y también sus determinados documentos de texto para luego al final commitear estos.

```
→  Clase7 git:(master) mkdir planila_c
→  Clase7 git:(master) mkdir readme
```

```
→  Clase7 git:(master) cd planilla_c
→  planilla_c git:(master) touch file1.txt
→  planilla_c git:(master) ✗ cd..
zsh: command not found: cd..
→  planilla_c git:(master) ✗ cd ..
→  Clase7 git:(master) ✗ cd readme
→  readme git:(master) ✗ touch file2.txt
```

3- Primero con **git log --oneline** para que me aparezca el <hash>

```
41fe4f7 (HEAD -> master, origin/master) Añadiendo primer a file2.txt
ceef83a Añadiendo primer cambio a file1.txt
dccac00 Añadiendo file2.txt vacio
f7c4092 Añadiendo file1.txt vacio
76166db first commit
(END)
```

4- Regresamos al commit anterior:

```
→  Clase7 git:(master) git checkout ceef83a
Note: checking out 'ceef83a'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at ceef83a... Añadiendo primer cambio a file1.txt
→  Clase7 git:(ceef83a)
```

5- Creamos la nueva rama llamada “Desarrollo” de esta forma:

```
→  Clase7 git:(ceef83a) git checkout -b desarrollo
Switched to a new branch 'desarrollo'
→  Clase7 git:(desarrollo)
```

6-Aquí se crearon tres documentos de texto, se agregó y luego se realizó el commit.

```
 Clase7 git:(desarrollo) touch file3.txt
 Clase7 git:(desarrollo) ✗ touch file4.txt
 Clase7 git:(desarrollo) ✗ touch file5.txt
 Clase7 git:(desarrollo) ✗ git status
n branch desarrollo
ntracked files:
 (use "git add <file>..." to include in what will be committed)

       file3.txt
       file4.txt
       file5.txt
```

```
 * [new branch]      desarrollo -> desarrollo
Branch 'desarrollo' set up to track remote branch 'desarrollo' from 'origin'.
→  Clase7 git:(desarrollo) ✗ git add file3.txt
→  Clase7 git:(desarrollo) ✗ git add file4.txt
→  Clase7 git:(desarrollo) ✗ git add file5.txt
→  Clase7 git:(desarrollo) ✗ git commit -m "Agregando file3.txt,file4.txt,file5.txt"
[desarrollo d574862] Agregando file3.txt,file4.txt,file5.txt
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3.txt
 create mode 100644 file4.txt
 create mode 100644 file5.txt
```

7- Creamos una nueva rama”Prueba”.

```
→  Clase7 git:(desarrollo) git checkout -b prueba
Switched to a new branch 'prueba'
```

8- Creamos los archivos nuevos y lo agregamos a la rama “Prueba” para al final comitearlos.

```
        file6.txt
        file7.txt
        file8.txt

nothing added to commit but untracked files present (use "git add" to track)
➜  Clase7 git:(prueba) ✗ git add file6.txt
➜  Clase7 git:(prueba) ✗ git add file7.txt
➜  Clase7 git:(prueba) ✗ git add file8.txt
```

```
➜  Clase7 git:(prueba) ✗ git commit -m "Agregamos los documentos file6.txt,file7.txt y file8.txt a la rama Prueba"
[prueba bbc76c6] Agregamos los documentos file6.txt,file7.txt y file8.txt a la rama Prueba
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file6.txt
 create mode 100644 file7.txt
 create mode 100644 file8.txt
```

9- Con el comando “**git log --all --decorate --oneline**” vemos todas las ramas y documentos de texto realizados.
Nos muestra lo siguiente.
Volvemos atrás apretando en el teclado “q”.

```
bbc76c6 (HEAD -> prueba, origin/prueba) Agregamos los documentos file6.txt,file7.txt y file8.txt a la rama Prueba
d574862 (origin/desarrollo, desarrollo) Agregando file3.txt,file4.txt,file5.txt
41fe4f7 (origin/master, master) Añadiendo primer a file2.txt
ceef83a Añadiendo primer cambio a file1.txt
dccac00 Añadiendo file2.txt vacio
f7c4092 Añadiendo file1.txt vacio
76166db first commit
(END)
```

10- En la siguiente foto eliminamos de manera local la rama llamada **“Prueba”** con el comando: “**git branch -d prueba”**

```
➜  Clase7 git:(master) git branch -d prueba
warning: deleting branch 'prueba' that has been merged to
         'refs/remotes/origin/prueba', but not yet merged to HEAD.
Deleted branch prueba (was bbc76c6).
```

11- De nuevo con el comando "**git log --all --decorate --oneline**" vemos las ramas , los documentos de texto y los commits realizados.

```
d574862 (origin/desarrollo, desarrollo) Agregando file3.txt,file4.txt,file5.txt
41fe4f7 (HEAD -> master, origin/master) Añadiendo primer a file2.txt
ceef83a Añadiendo primer cambio a file1.txt
dccac00 Añadiendo file2.txt vacio
f7c4092 Añadiendo file1.txt vacio
76166db first commit
(END)
```

12- Etiquetas realizadas del proyecto ejemplo:

```
41fe4f7 (HEAD -> master, origin/master) Añadiendo primer a file2.txt
ceef83a Añadiendo primer cambio a file1.txt
dccac00 (tag: etiqueta_prueba2, tag: etiqueta_prueba) Añadiendo file2.txt vacio
f7c4092 (tag: etiqueta_prueba.2) Añadiendo file1.txt vacio
76166db (tag: etiqueta_prueba.3) first commit
(END)
```

13- Git es una herramienta fundamental para el desarrollo de software y la programación por varios motivos, algunos de los que podemos nombrar son:

- Control de versiones: Git nos permite mantener un historial completo de los cambios realizados en el código fuente. Esto facilita la colaboración entre los desarrolladores, ya que pueden trabajar simultáneamente en diferentes partes con el uso de varias ramas(branch) del código sin temor a sobrescribir el trabajo de los demás o dañar la rama Master. Además, si algo sale mal, es posible retroceder a versiones anteriores del código.

- Colaboración: Git nos facilita la colaboración entre programadores en proyectos de software o código de varios tipos, ya que nos permite trabajar en el mismo proyecto de manera simultánea y coordinada. Como dijimos en el punto anterior, los cambios se pueden integrar de forma eficiente mediante el uso de ramas (branches) y fusiones de las mismas (merges).

- Rastreo de cambios: Git nos proporciona un registro detallado de quiénes hicieron en nuestro código cambios, cuándo y por qué. Esto es un punto crucial para la trazabilidad y la resolución de problemas que podemos encontrar en nuestro proyecto.

- Experimentación y desarrollo experimental: Las ramas creadas en Git nos permiten trabajar en nuevas características o experimentos del proyecto sin afectar el código base (Master). Esto nos

permite fomentar la innovación, la creatividad y la exploración de nuevas ideas sin comprometer la estabilidad del proyecto principal.

- Despliegue continuo: Git es esencial para las prácticas de integración continua (CI) y despliegue continuo (CD). Nos permite automatizar la entrega de software al garantizar que cada cambio que realizamos, luego de ser aprobado en el repositorio, pueda ser implementado automáticamente en un entorno de producción.

- Respaldo y redundancia: Usando Git, todo el historial de nuestro proyecto se encontrará en un servidor remoto y en los repositorios locales de cada uno de los desarrolladores. Esto nos proporciona una capa adicional de seguridad en caso de la pérdida de datos.

Para resumir, Git es importante porque nos facilita la colaboración, el control de versiones, la trazabilidad y la automatización en el desarrollo de proyectos en el sector de la programación y software, lo que lleva a darnos un tipo de proceso de desarrollo más eficiente y confiable.