Practica 1 – Comandos básicos de Git

Para cada pregunta en un documento word añade capturas de los comandos utilizados y guardar los logs de la terminal de git bash, los log's de git (git log --oneline) y los reflogs. Entregar todo en un fichero comprimido en .zip

Pregunta 1: ***Inicio de un proyecto Big Data con Git y seguimiento de cambios***

Has sido asignado para crear un repositorio para gestionar los scripts de procesamiento de datos de un pipeline de Big Data. Debes inicializar un repositorio de Git, configurar tu identidad y realizar una serie de acciones para rastrear y gestionar los cambios en el proyecto.

Escenario:

Trabajas en un proyecto de procesamiento de datos, donde tres tipos de archivos deben ser versionados:

- **etl_script.py**: Script en Python que extrae y procesa datos de una base de datos de Azure.
- **data_pipeline.sh**: Script Bash que automatiza el pipeline de datos.
- **spark_job.scala**: Código Scala que se ejecuta en Spark para realizar análisis de grandes volúmenes de datos.

Estos archivos se encuentran en el siguiente directorio:

```
/home/user/bigdata_project/
    ├── etl_script.py
    ├── data_pipeline.sh
    ├── spark_job.scala
```

Responder con capturas:

1. **Inicializa el repositorio Git** dentro del directorio del proyecto.
2. **Configura tu usuario globalmente en Git** para que tu identidad quede registrada en cada commit que realices.
3. **Verifica que la configuración de tu usuario se haya aplicado correctamente**.
4. **Añade el archivo etl_script.py al índice de Git** para ser rastreado.
5. **Añade todos los archivos del proyecto al índice**.
6. **Verifica el estado de los archivos en el repositorio** para confirmar que los archivos han sido añadidos correctamente al índice.
7. **Realiza tu primer commit** en el repositorio con un mensaje que explique el propósito de este commit.
8. **Realiza varios commits y luego revisa el historial de commits** en formato resumido (solo las primeras líneas de cada commit).
9. **Usa git checkout para revisar una versión anterior del repositorio** y verifica cómo estaban los archivos antes de realizar algunos cambios.

10. Verifica el registro de todas las acciones realizadas en el repositorio, incluyendo los commits eliminados o revertidos usando git reflog.
11. **Crea una etiqueta (tag) en el commit actual** para marcar la primera versión del pipeline.
12. **Realiza cambios en etl_script.py y spark_job.scala** (simula cualquier cambio en el código) y **verifica las diferencias** con el estado anterior usando git diff.
13. **Verifica las diferencias con respecto a la versión más reciente confirmada (HEAD)** en todo el proyecto.

Archivos:

etl_script.py

```
import pandas as pd

def etl_process():
    data = pd.read_csv('data.csv')
    # Transformaciones de datos aquí
    data.to_csv('cleaned_data.csv')
```

data_pipeline.sh

```
#!/bin/bash
echo "Iniciando pipeline de datos"
python3 etl_script.py
```

spark_job.scala

```
val data = spark.read.csv("cleaned_data.csv")
data.show()
```

Pregunta 2: *Modificación y seguimiento de múltiples archivos con Git*

Has hecho algunos ajustes en los scripts del proyecto de Big Data y ahora necesitas asegurarte

de que esos cambios se registren adecuadamente en Git.

Escenario:

Has actualizado el script **etl_script.py** para corregir un error en el procesamiento de datos y has modificado el script **data_pipeline.sh** para que use la nueva versión de etl_script.py. Además, necesitas añadir más transformaciones en el código de **spark_job.scala.**

Responder con capturas

1. Realiza los siguientes cambios en los archivos:
    - **Actualiza etl_script.py** para incluir más transformaciones:

```python
import pandas as pd

def etl_process():
    data = pd.read_csv('data.csv')
    # Nueva transformación agregada
    data['new_column'] = data['existing_column'] * 2
    data.to_csv('cleaned_data_v2.csv')
```

    - **Modifica data_pipeline.sh** para usar la nueva versión del archivo procesado:

```bash
#!/bin/bash
echo "Iniciando pipeline de datos actualizado"
python3 etl_script.py
echo "Pipeline completado"
```

    - **Añade transformaciones adicionales a spark_job.scala:**

```scala
val data = spark.read.csv("cleaned_data_v2.csv")
val transformedData = data.withColumn("double_column", data("new_column") * 2)
transformedData.show()
```

2. **Añade y confirma los cambios de todos los archivos** usando un solo comando (git commit -

am). Este comando añade automáticamente todos los archivos modificados al área de preparación y realiza el commit.

3. **Verifica el historial de commits** en formato resumido, observando cómo los cambios han sido registrados.
4. **Realiza más cambios en el archivo spark_job.scala,** esta vez agregando un nuevo análisis de datos y vuelve a realizar un commit.
5. Revisa los cambios que has realizado entre dos commits recientes. Usa el hash de los commits para comparar.

Pregunta 3: *Gestión de cambios erróneos y restauración en Git*

En este ejercicio, vas a trabajar en un repositorio donde accidentalmente se han realizado varios cambios erróneos. Debes restaurar los archivos a un estado anterior, también se usará .gitignore para excluir archivos innecesarios del seguimiento de Git y aplicar distintos tipos de git reset para gestionar los commits incorrectos.

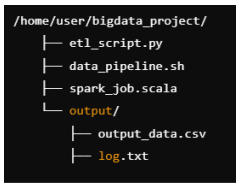
Escenario:

Tienes los siguientes archivos en tu proyecto de Big Data:

- **etl_script.py**: Script que extrae y transforma datos.

- **data_pipeline.sh**: Automación del pipeline de datos.
- **spark_job.scala**: Script que ejecuta procesos Spark.
- **output/**: Carpeta con archivos de salida generados por los scripts.

Los archivos se encuentran en el siguiente directorio:



Responder con capturas:

1. Inicializa el repositorio Git dentro del directorio del proyecto y configura tu identidad.
2. **Añade todos los archivos a Git** y realiza el primer commit.
3. **Crea un archivo .gitignore** para evitar que la carpeta output/ (y sus archivos) sea incluida en el control de versiones.
4. Realiza cambios en etl_script.py, luego verifica el estado de los archivos con git status.
5. **Añade el archivo modificado y realiza un commit** con el mensaje "Actualización con error en ETL".
6. **Revisa el historial de commits**.

7. **Verifica las diferencias entre la versión actual y la más reciente** (HEAD) para identificar el error.
8. Restaura el archivo etl_script.py al estado anterior antes del último commit usando git restore.
9. **Corrige el error en el archivo etl_script.py** y realiza un nuevo commit con el mensaje "Corrección del error en ETL".
10. **Realiza un reset suave** a la versión anterior para deshacer el último commit pero mantener los cambios en el área de trabajo.
11. **Revisa el historial de acciones** usando git reflog para ver los cambios recientes.
12. Realiza un reset duro para deshacer completamente todos los cambios, eliminando tanto los commits como las modificaciones.
13. **Verifica el estado actual del repositorio** y observa cómo todos los cambios han sido revertidos
14. **Crea una etiqueta (tag)** para marcar la primera versión estable del proyecto después de realizar las correcciones.
15. **Realiza cambios erróneos en etl_script.py. Luego verifica el estado de los archivos. Luego restaurar etl_script.py al estado anterior sin afectar el staging area, usando git restore.**

16. **Simula que el archivo spark_job.scala ha sido añadido por error al staging area,** y usa git restore --staged para quitarlo del staging area.
17. **Realiza cambios en etl_script.py,** esta vez de forma intencional, y realiza un commit.
18. **Revisa el historial de commits,** identifica un commit que se hizo por error, y usa git revert para deshacerlo sin modificar el historial del proyecto.
19. **Realiza un commit con un mensaje incorrecto,** y luego usa git commit --amend -m para corregir el mensaje del commit más reciente.
20. **Crea un nuevo directorio scripts/** para organizar mejor los archivos del proyecto.
21. **Mueve los archivos etl_script.py y data_pipeline.sh** al nuevo directorio usando git mv.
22. **Elimina el archivo log.txt** de la carpeta output/ porque no debería estar versionado, usando git rm.
23. **Confirma todos los cambios realizados** en los pasos anteriores.
24. **Verifica el historial de commits** y los cambios recientes.