

TCDM Práctica 2

José Antonio

- Primera parte: ejecución del código filesystem_cat

Creamos el entorno virtual, descargamos las librerías y ejecutamos filesystem_cat, un script que lee el contenido de un archivo que puede estar en cualquier sistema de ficheros, gracias a pyarrow, y lo muestra por la salida estándar.

```
(.venv) luser@namenode:~$ echo "hola mundo" > a.txt
(.venv) luser@namenode:~$ ls
a.txt practica2
(.venv) luser@namenode:~$ cat a.txt
hola mundo
(.venv) luser@namenode:~$ hdfs dfs -put a.txt /user/luser/
(.venv) luser@namenode:~$ hdfs dfs -ls /user/luser/
Found 2 items
-rw-r--r-- 3 luser supergroup          11 2025-12-12 16:00 /user/luser/a.txt
drwxr-xr-x - luser supergroup          0 2025-10-22 11:42 /user/luser/libros
(.venv) luser@namenode:~$ ls
a.txt practica2
(.venv) luser@namenode:~$ rm a.txt
(.venv) luser@namenode:~$ hdfs dfs -ls /user/luser/
Found 2 items
-rw-r--r-- 3 luser supergroup          11 2025-12-12 16:00 /user/luser/a.txt
drwxr-xr-x - luser supergroup          0 2025-10-22 11:42 /user/luser/libros
(.venv) luser@namenode:~$ cd practica2/
(.venv) luser@namenode:~/practica2$ python3 filesystem_cat.py hdfs://namenode:9000/user/luser/a.txt
2025-12-12 16:01:12,401 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hola mundo
(.venv) luser@namenode:~/practica2$ |
```

- Segunda parte: código copy_half_file

El script copy_half_file.py es un programa en Python pensado para copiar solo la segunda mitad de un archivo desde un origen hasta un destino, midiendo esa “mitad” por tamaño en bytes.

Para poder trabajar igual con archivos en HDFS o en el sistema local, el script utiliza PyArrow, concretamente el módulo pyarrow.fs (pyarrow filesystem), que como ya comentamos antes ofrece una API unificada de “sistema de archivos”. Así, el usuario puede pasar rutas en formato URI como hdfs://... o file://..., y el programa se encarga de interpretar cada una, obteniendo el tipo de filesystem correcto y la ruta interna real. Antes de copiar nada, el script consulta los metadatos del archivo de origen mediante FileInfo para comprobar que existe, que realmente es un archivo (no un directorio) y para saber su tamaño exacto.

Con el tamaño ya conocido, calcula el punto medio usando división entera (si el tamaño es impar, redondea hacia abajo). Luego abre el archivo origen para lectura con posibilidad de hacer seek (mover el puntero) y abre el archivo destino como un stream de escritura. A continuación, desplaza el puntero del origen a la posición de la mitad y copia todo lo que queda hasta el final, usando shutil.copyfileobj para hacerlo por bloques, sin cargar el archivo completo en memoria.

Comprobamos que funciona el local:

```
luser@namenode:~/practica2$ python3 copy_half_file.py
Uso: copy_half_file.py <uri_origen> <uri_destino>
Ejemplos:
    copy_half_file.py hdfs:///user/luser/a.txt file:///home/luser/mitad.txt
    copy_half_file.py file:///home/luser/a.txt hdfs:///user/luser/mitad.txt
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py /home/luser/practica2/t.txt /home/luser/practica2/a.txt
(.venv) luser@namenode:~/practica2$ ls
a.txt  copy_half_file.py  filesystem_cat.py  requirements.txt  t.txt
(.venv) luser@namenode:~/practica2$ cat a.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ |
```

Comprobamos que funciona en hdfs:

```
luser@namenode:~/practica2$ python3 copy_half_file.py
Uso: copy_half_file.py <uri_origen> <uri_destino>
Ejemplos:
    copy_half_file.py hdfs:///user/luser/a.txt file:///home/luser/mitad.txt
    copy_half_file.py file:///home/luser/a.txt hdfs:///user/luser/mitad.txt
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py /home/luser/practica2/t.txt /home/luser/practica2/a.txt
(.venv) luser@namenode:~/practica2$ ls
a.txt  copy_half_file.py  filesystem_cat.py  requirements.txt  t.txt
(.venv) luser@namenode:~/practica2$ cat a.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvwxyz
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py hdfs://namenode:9000/user/luser/a.txt hdfs://namenode:9000/user/luser/b.txt
2025-12-12 16:58:12,960 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(.venv) luser@namenode:~/practica2$ hdfs dfs -ls /user/luser/
Found 3 items
-rw-r--r--  3 luser supergroup      11 2025-12-12 16:00 /user/luser/a.txt
-rw-r--r--  3 luser supergroup       6 2025-12-12 16:58 /user/luser/b.txt
drwxr-xr-x - luser supergroup      0 2025-10-22 11:42 /user/luser/libros
(.venv) luser@namenode:~/practica2$ python3 filesystem_cat.py hdfs://namenode:9000/user/luser/b.txt
2025-12-12 16:58:57,282 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
mundo
(.venv) luser@namenode:~/practica2$ |
```

- Tercera parte: Probar el comando hdfs dfsadmin

En este apartado nos piden establecer una **cuota de 4** (cuota de nombres) y comprobar cuántos ficheros se pueden copiar en ese directorio. Para ello, primero hay que entender qué son las cuotas en HDFS.

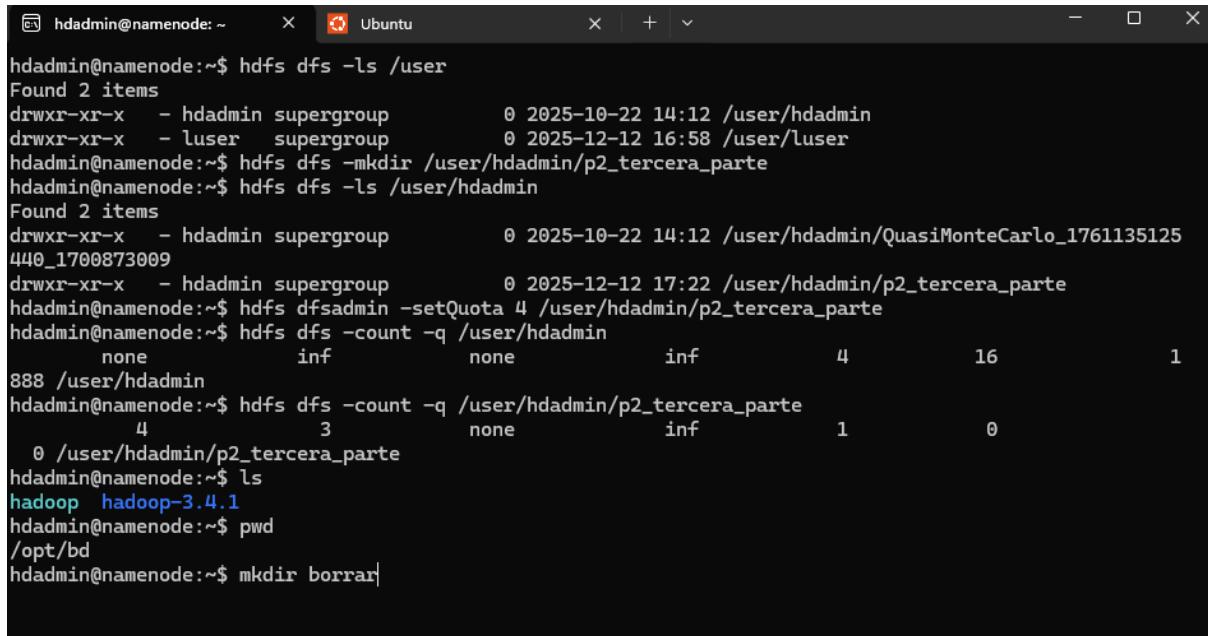
En HDFS, las cuotas son límites que el **NameNode** impone sobre un directorio HDFS (y todo su subárbol) para evitar que un usuario o aplicación utilice demasiados recursos.

Hay dos tipos principales de cuotas, cuota de nombres (namespace quota) y cuota de espacio (space quota).

- Cuota de nombres: limita cuántos “objetos” (inodos) puedes tener dentro de ese directorio y sus subdirectorios. En la práctica cada archivo y directorio consume 1. El conteo se aplica al árbol bajo ese directorio y se incluye el propio directorio, por eso al poner cuota en /ruta, el propio directorio ya cuenta como 1.

- Cuota de espacio: limita cuántos bytes pueden ocupar los datos dentro de ese directorio (y subdirectorios), cuenta el espacio replicado. Si un archivo ocupa 100 MB y el factor de replicación es 3, suele contar como ~300 MB de cuota.

Primero creamos un directorio para el ejercicio y le ponemos la cuota de 4:



```
hdadmin@namenode:~$ hdfs dfs -ls /user
Found 2 items
drwxr-xr-x - hdadmin supergroup          0 2025-10-22 14:12 /user/hdadmin
drwxr-xr-x - luser   supergroup          0 2025-12-12 16:58 /user/luser
hdadmin@namenode:~$ hdfs dfs -mkdir /user/hdadmin/p2_tercera_parte
hdadmin@namenode:~$ hdfs dfs -ls /user/hdadmin
Found 2 items
drwxr-xr-x - hdadmin supergroup          0 2025-10-22 14:12 /user/hdadmin/QuasiMonteCarlo_1761135125
440_1700873009
drwxr-xr-x - hdadmin supergroup          0 2025-12-12 17:22 /user/hdadmin/p2_tercera_parte
hdadmin@namenode:~$ hdfs dfsadmin -setQuota 4 /user/hdadmin/p2_tercera_parte
hdadmin@namenode:~$ hdfs dfs -count -q /user/hdadmin
      none        inf        none        inf         4        16         1
888 /user/hdadmin
hdadmin@namenode:~$ hdfs dfs -count -q /user/hdadmin/p2_tercera_parte
      4           3        none        inf         1         0
      0 /user/hdadmin/p2_tercera_parte
hdadmin@namenode:~$ ls
hadoop  hadoop-3.4.1
hdadmin@namenode:~$ pwd
/opt/bd
hdadmin@namenode:~$ mkdir borrar|
```

Intentamos copiar 4 ficheros (a.txt, b.txt, c.txt, d.txt) en el directorio con cuota y vemos que falla al intentar copiar el cuarto (d.txt). Es decir, nos deja copiar 3 ficheros con una cuota de 4. Este comportamiento es correcto: al establecer una cuota de nombres, el directorio también consume 1 unidad de cuota, por lo que quedan 3 unidades disponibles para ficheros (u otros inodos) dentro de ese subárbol.

```
hdadmin@namenode:~/borrar$ ls -ltr
total 16
-rw-r--r-- 1 hdadmin hadoop 2 dic 12 17:25 a.txt
-rw-r--r-- 1 hdadmin hadoop 2 dic 12 17:26 b.txt
-rw-r--r-- 1 hdadmin hadoop 2 dic 12 17:26 c.txt
-rw-r--r-- 1 hdadmin hadoop 2 dic 12 17:26 d.txt
hdadmin@namenode:~/borrar$ hdfs dfs -put ./a.txt /user/hdadmin/p2_tercera_parte/
hdadmin@namenode:~/borrar$ hdfs dfs -put ./b.txt /user/hdadmin/p2_tercera_parte/
hdadmin@namenode:~/borrar$ hdfs dfs -put ./c.txt /user/hdadmin/p2_tercera_parte/
hdadmin@namenode:~/borrar$ hdfs dfs -put ./d.txt /user/hdadmin/p2_tercera_parte/
put: The NameSpace quota (directories and files) of directory /user/hdadmin/p2_tercera_parte is exceeded: quota=4 file count=5
hdadmin@namenode:~/borrar$ |
```

- Cuarta parte: Probar el comando hdfs dfsadmin

Paso 1)

hdfs fsck / es un comando hdfs que nos permite hacer un “file system check”. Le pide al NameNode un reporte del estado de archivos/bloques bajo la ruta que le especifiquemos.

Ejecutamos *hdfs fsck* para comprobar si hay errores. Recorre el árbol desde / y reporta problemas (bloques perdidos, replicación incorrecta, etc.). Con egrep vemos solo los errores.

```
hdadmin@namenode:~$ hdfs fsck / -files -blocks -locations 2>/dev/null | egrep -i "CORRUPT|MISSING|UNDER_REPLICATED|OVER_REPLICATED|FAILED|ERROR"
Missing blocks:          0
Corrupt blocks:         0
Missing replicas:       10 (6.329114 %)
Missing block groups:   0
Corrupt block groups:  0
Missing internal blocks: 0
hdadmin@namenode:~$
```

Paso 2)

Detenemos los datanodes

```
jose@DESKTOP-VUD8TI9:~$ docker container stop datanode2 datanode3
datanode2
datanode3
jose@DESKTOP-VUD8TI9:~$ |
```

y esperamos. Comprobamos luego de 10 minutos que solo nos quedan 2 datanodes vivos:

```
Live datanodes (2):  
  
Name: 172.18.0.5:9866 (datanode4.hadoop-cluster)  
Hostname: datanode4  
Rack: /rack2  
Decommission Status : Decommissioned  
Configured Capacity: 1081101176832 (1006.85 GB)  
DFS Used: 271376384 (258.80 MB)  
Non DFS Used: 30315540480 (28.23 GB)  
DFS Remaining: 995521904640 (927.15 GB)  
DFS Used%: 0.03%  
DFS Remaining%: 92.08%  
Configured Cache Capacity: 0 (0 B)  
Cache Used: 0 (0 B)  
Cache Remaining: 0 (0 B)  
Cache Used%: 100.00%  
Cache Remaining%: 0.00%  
Xceivers: 0  
Last contact: Fri Dec 12 18:42:27 CET 2025  
Last Block Report: Fri Dec 12 15:12:31 CET 2025  
Num of Blocks: 36  
  
Name: 172.18.0.8:9866 (datanode1.hadoop-cluster)  
Hostname: datanode1  
Decommission Status : Normal  
Configured Capacity: 1081101176832 (1006.85 GB)  
DFS Used: 339488768 (323.76 MB)  
Non DFS Used: 30247428096 (28.17 GB)  
DFS Remaining: 995521904640 (927.15 GB)  
DFS Used%: 0.03%  
DFS Remaining%: 92.08%  
Configured Cache Capacity: 0 (0 B)  
Cache Used: 0 (0 B)  
Cache Remaining: 0 (0 B)  
Cache Used%: 100.00%  
Cache Remaining%: 0.00%  
Xceivers: 0  
Last contact: Fri Dec 12 18:42:27 CET 2025  
Last Block Report: Fri Dec 12 16:47:18 CET 2025  
Num of Blocks: 48  
  
Dead datanodes (3):  
  
Name: 172.18.0.2:9866 (172.18.0.2)  
Hostname: datanode5  
Decommission Status : Normal
```

Paso 3)

Volvemos a hacer un diagnóstico del sistema (esta vez sin el grep) y nos centramos en la parte de replicación:

```
Replicated Blocks:  
Total size: 336501550 B  
Total files: 44  
Total blocks (validated): 48 (avg. block size 7010448 B)  
Minimally replicated blocks: 48 (100.0 %)  
Over-replicated blocks: 0 (0.0 %)  
Under-replicated blocks: 48 (100.0 %)  
Mis-replicated blocks: 0 (0.0 %)  
Default replication factor: 3  
Average block replication: 1.75  
Missing blocks: 0  
Corrupt blocks: 0  
Missing replicas: 74 (46.83544 %)  
DecommissionedReplicas: 36  
Blocks queued for replication: 0
```

Vemos que ahora nos aparecen 48 under-replicated blocks, el 100% de los bloques. Solo nos quedan dos datanodes vivos y nuestro factor de replicación es de 3. Es por eso que nos sale este resultado.

Paso 4)

Si hacemos el *get* vemos que sigue funcionando:

```
hdadmin@namenode:~$ hdfs dfs -get /user/luser/libros/random_words.txt.bz2  
hdadmin@namenode:~$ ls  
hadoop hadoop-3.4.1 random_words.txt.bz2
```

Esto es normal, mientras cada bloque del fichero tenga al menos una réplica disponible en alguno de los dos datanodes que quedan vivos. El *get* fallaría en caso de Missing blocks (no queda ninguna réplica de ese bloque en ningún DN vivo).

Si comprobamos el fichero con *hdfs fsck* vemos que obviamente esta under-replicated, por lo que explicamos en el apartado anterior, pero Missing blocks es igual a 0, por lo que el *get* funciona perfectamente.

```
Status: HEALTHY
Number of data-nodes: 2
Number of racks: 2
Total dirs: 0
Total symlinks: 0

Replicated Blocks:
  Total size: 330326458 B
  Total files: 1
  Total blocks (validated): 5 (avg. block size 66065291 B)
  Minimally replicated blocks: 5 (100.0 %)
  Over-replicated blocks: 0 (0.0 %)
  Under-replicated blocks: 5 (100.0 %)
  Mis-replicated blocks: 0 (0.0 %)
  Default replication factor: 3
  Average block replication: 1.8
  Missing blocks: 0
  Corrupt blocks: 0
  Missing replicas: 6 (40.0 %)
  DecommissionedReplicas: 4
  Blocks queued for replication: 0

Erasure Coded Block Groups:
  Total size: 0 B
  Total files: 0
  Total block groups (validated): 0
  Minimally erasure-coded block groups: 0
  Over-erasure-coded block groups: 0
  Under-erasure-coded block groups: 0
  Unsatisfactory placement block groups: 0
  Average block group size: 0.0
  Missing block groups: 0
  Corrupt block groups: 0
  Missing internal blocks: 0
  Blocks queued for replication: 0
FSCK ended at Fri Dec 12 18:48:13 CET 2025 in 1 milliseconds
```

```
The filesystem under path '/user/luser/libros/random_words.txt.bz2' is HEALTHY
hdadmin@namenode:~$ |
```

Paso 5)

Añadimos el datanode 6 y ejecutamos el diagnostico de nuevo:

```
Replicated Blocks:  
  Total size: 336501550 B  
  Total files: 44  
  Total blocks (validated): 48 (avg. block size 7010448 B)  
  Minimally replicated blocks: 48 (100.0 %)  
  Over-replicated blocks: 0 (0.0 %)  
  Under-replicated blocks: 23 (47.916668 %)  
  Mis-replicated blocks: 0 (0.0 %)  
  Default replication factor: 3  
  Average block replication: 2.4583333  
  Missing blocks: 0  
  Corrupt blocks: 0  
  Missing replicas: 40 (25.316456 %)  
  DecommissionedReplicas: 36  
  Blocks queued for replication: 0
```

Vemos que ha funcionado correctamente porque han bajado los under-replicated blocks.