Hadoop Exercises on macOS

Exercise 0: Start Hadoop on macOS

Start Hadoop services

start-dfs.sh

Starts HDFS daemons: NameNode, DataNode, SecondaryNameNode (storage layer).

start-yarn.sh

Starts YARN daemons: ResourceManager, NodeManager (compute layer).

Check running Hadoop processes

jps

Lists all Java processes; you should see NameNode, DataNode, ResourceManager, NodeManager.

Exercise 1: Explore Hadoop File System (HDFS)

Goal: Understand how HDFS differs from your local filesystem.

hdfs dfs -ls /

Lists contents of the HDFS root directory.

hdfs dfs -mkdir /user

Creates a directory /user in HDFS.

hdfs dfs -mkdir /user/yourname

Creates a personal subdirectory in HDFS for your files.

hdfs dfs -ls /user

Lists directories under /user.

Concept:

HDFS stores data distributed across nodes; commands are similar to Linux shell but operate on HDFS.

Exercise 2: Upload Files to HDFS

Goal: Move local files into HDFS.

echo "Hello Hadoop" > hello.txt

Creates a local file named hello.txt with content “Hello Hadoop”.

hdfs dfs -put hello.txt /user/yourname/

Uploads the local file into your HDFS directory.

hdfs dfs -ls /user/yourname

Lists files in your HDFS directory to verify upload.

hdfs dfs -cat /user/yourname/hello.txt

Displays the content of the file stored in HDFS.

Exercise 3: Download & Delete Files from HDFS

hdfs dfs -get /user/yourname/hello.txt downloaded_hello.txt

Downloads the HDFS file to your local machine as downloaded_hello.txt.

hdfs dfs -rm /user/yourname/hello.txt

Deletes the file from HDFS.

Concept:

HDFS ≠ local filesystem; files must be explicitly moved or deleted.

Exercise 4: HDFS Directory Operations

hdfs dfs -mkdir -p /user/yourname/data/input

Creates nested directories in HDFS (-p creates parent directories if needed).

hdfs dfs -rm -r /user/yourname/data

Removes a directory and all its contents recursively.

Exercise 5: Understand HDFS Replication

hdfs dfs -stat %r /user/yourname/hello.txt

Shows the replication factor (default = 3) of a file in HDFS.

Concept:

Replication ensures fault tolerance by storing multiple copies of data blocks.

Exercise 6: Run Your First MapReduce Job (WordCount)

Goal: Run Hadoop's classic WordCount example.

nano input.txt

Creates a local text file to serve as input for the WordCount job.

Input content example:

hadoop is fun

hadoop is powerful

hadoop is scalable

hdfs dfs -mkdir /user/yourname/wordcount

Creates an HDFS directory for WordCount input.

hdfs dfs -put input.txt /user/yourname/wordcount/input

Uploads the input file to HDFS.

```
hadoop jar \
/usr/local/Cellar/hadoop/*/libexec/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar \
wordcount \
/user/yourname/wordcount/input \
/user/yourname/wordcount/output
```

Runs the WordCount MapReduce job on the input directory and stores output in HDFS.

```
hdfs dfs -cat /user/yourname/wordcount/output/part-r-00000
```

Displays the WordCount results.

Expected output:

```
hadoop 3
is 3
fun 1
powerful 1
scalable 1
```

Exercise 7: Understand MapReduce Output Structure

```
hdfs dfs -ls /user/yourname/wordcount/output
```

Lists the output directory; each reducer produces a separate part file.

Concept:

Output is split into part-r-00000, part-r-00001, etc. if multiple reducers are used.

Exercise 8: Modify Input & Re-run Job

echo "hadoop is distributed" >> input.txt

Appends a new line to the input file.

hdfs dfs -put -f input.txt /user/yourname/wordcount/input

Uploads and overwrites the input file in HDFS (-f = force).

hdfs dfs -rm -r /user/yourname/wordcount/output

Deletes previous output directory (Hadoop cannot overwrite output).

hadoop jar ... (same command as before)

Re-runs WordCount job on the updated input.

Concept:

Hadoop won't overwrite existing output directories; you must delete them first.

Exercise 9: View Hadoop Web UI

Open in browser:

- **NameNode UI:** http://localhost:9870

  Monitor HDFS, directories, storage usage, and health.

- **YARN UI:** http://localhost:8088

  Monitor running jobs and cluster resources.

What to explore:

- HDFS health
- Running jobs
- Node status

Exercise 10: Mini Project – Log Analysis

Goal: Count HTTP status codes from logs.

nano access.log

Create a sample log file with HTTP requests.

Example log content:

200 GET /index.html

404 GET /missing.html

200 POST /submit

500 GET /error

200 GET /home

hdfs dfs -mkdir /user/yourname/logs

Create HDFS directory for log analysis.

hdfs dfs -put access.log /user/yourname/logs/input

Upload the log file to HDFS.

hadoop jar \

/usr/local/Cellar/hadoop/*/libexec/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar \

wordcount \

/user/yourname/logs/input \

/user/yourname/logs/output

Runs WordCount MapReduce job on log file to count occurrences of HTTP status codes.

hdfs dfs -cat /user/yourname/logs/output/part-r-00000

Displays the result; you will see counts for 200, 404, 500, etc.