



## YCBS-257 - Data at Scale

---

# Hadoop Core Workshop

### General Instructions:

The purpose of this workshop is to get you started with Hadoop. Here you will learn how to use HDFS and how to write, compile, and execute a simple Hadoop program (MapReduce).

Start your Cloudera QuickStart VM, open a terminal windows to complete the workshop












Online resources:

<https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html>

## Part 1 - HDFS




### Exercise 1: HDFS basics operations

In this exercise you will manipulate HDFS with some basic commands such as creating directories and copying files.

1. Verify that HDFS it's running. 
2. Download this file <http://www.gutenberg.org/files/5000/5000-8.txt> (UTF-8 format) 
3. Create a new directory `/lab1` on HDFS. 
4. Put the 5000-8.txt file into HDFS under `/lab1`. 
5. View the content of `/lab1` directory. 
6. Determine the size of the file on HDFS. 
7. Print the first 5 lines to screen from the file on HDFS. 
8. Copy `lab1` to `/lab-hdfscopy` on HDFS. 
9. Copy `lab1` back to local fillesystem and name it `lab1_localcopy`. 
10. Delete the file from HDFS. 
11. Delete `/lab1` directory from HDFS. 

### Exercise 2: Merging HDFS files

In this exercise you will merge several files on HDFS. This technique is typically used to merge files resulting from MapReduce processing and place the resulting merged file on the local file system.



1. Extract the **FlumeData.zip** on the local file system
2. Create a new directory `/lab2` on HDFS.
3. Copy all extracted **FlumeData** files into this directory 
4. View the content of the `/lab2` directory (you should see 10 files)
5. Use the Merge command to merge all the files in one single file named **FlumeDataAll.txt** 
6. On the local file system, use the **wc -l** command to determine the size of the file on HDFS 



### Exercise 3: Creating a Hadoop Archive File

Hadoop Archives are used to solve small size file problem (in which files are very small compared to HDFS block size).

Hadoop Archives combines large number of small input files into one archive file. The archive will always have **.har** extension.

1. Create a Hadoop archive file named **flumedata.har** that combines all the files in the **/lab2** directory and place the new created **har** file into **/lab2\_out** directory 
2. View the content of **flumedata.har** and use the **wc -l** command to count the line numbers of the archive output file (**part-0**) 

### Exercise 4: HDFS basic administration operations

This exercise aims to familiarize you with basic HDFS administration commands such as checking the consistency of the system and printing a HDFS report.

1. Check the entire HDFS filesystem for inconsistencies/problems.
2. View and verify the consistency of HDFS files under **/user/cloudera**
3. View and verify the consistency of HDFS files blocks under **/user/cloudera**
4. Print an HDFS information report.
5. Make a backup of the NameNode metadata in a file named **nn\_metadata.txt**  
Info: (this file should appear on your local file system in **/var/log/Hadoop-hdfs**)
6. Print the HDFS safe mode status



## Part 2 – MapReduce

The goal of this lab is to familiarize you with the basic process for creating and running Hadoop MapReduce jobs.

Online resources:

<https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CommandsManual.html>

### **Exercise 1: Running a jar file**



The example given below runs a Hadoop **Pi** calculation job. The Pi application is packaged into a Hadoop MapReduce examples archive. The application just runs a map-only job and print the final result on the screen.

To run the Pi QuasiMonteCarlo calculator<sup>i</sup> you should provide two parameters:

<nMaps>       mappers numbers (how many mappers)  
<nSamples>    samples counts (how many points)

- To run the pi calculator using **5** mappers and **50000** samples enter the following command:

```
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples-2.6.0-mr1-cdh5.13.0.jar pi 5 50000
```

- Report the pi value 
- Rerun the pi calculator using **5 000 000** samples
- Report the pi value (is there any difference?) 

### **Exercise 2: Running a WordCount job**

Based on what you learned in the previous section, run the **wordcount** application (packaged in the same archive used in the previous section)

To run the **wordcount** application you should provide two parameters:

<in>    path for input files  
<out>   path for out result

1. Download this file <http://www.gutenberg.org/ebooks/28885.txt.utf-8>
2. Create a new directory **wc\_in** on HDFS
3. Place the file into this directory
4. Run the **wordcount** MapReduce job
5. View the 20 first lines of the result
6. Print the count of the word '**the**'



## Exercise 3: Compiling a MapReduce application

The purpose of this exercise is to familiarize you with the methodology of creating a Hadoop MapReduce application written in Java.

In this exercise you will not write the java code of the application but rather you will follow the steps for compiling the code, generating the jar file and running it on Hadoop.

This MapReduce application takes a data sample file which is a list of units per year. The MR job will select only those > 40.

### Part 1: Creating the **maxunits** jar archive

1. Create a new directory **hadoopdev** into **/home/cloudera/** (local file system)
2. Place the **BigUnits.java** file and the **hadoop-core-1.2.1.jar** into this directory
3. Create a new directory **units** under **hadoopdev**
4. Open a Terminal and navigate to **hadoopdev**
5. Compile the **BigUnits.java** using this command:  

```
$ javac -classpath hadoop-core-1.2.1.jar -d units BigUnits.java
```
6. Use the **tree** command and check that three files **.class** has been generated
7. Create the java archive using this command  

```
$ jar -cvf bigunits.jar -C units/ .(space character after the slash)
```
8. Verify that a **bigunits.jar** has been generated into the **hadoopdev** directory

### Part 2: Running the **maxunits** jar archive

1. Create a new directory **lab3\_in** on HDFS
2. Place the file **sample-data.txt** file into this directory
3. Verify the content of the **lab3\_in** directory
4. Run the **maxunits.jar** archive using this command  

```
$ hadoop jar bigunits.jar hadooplab.BigUnits lab3_in lab3_out
```
5. View the content of the reduce result file in **lab3\_out**. This should be similar to the screenshot below

1979	43
1984	43
1984	42
1984	41
1985	45
1985	41
1985	41
1985	41

- Copy the result file to the local file system and rename it **bigunits.txt**



## Exercise 4: Analysis of Olympics dataset

In this section, you will write, compile and run your own MapReduce application based on what you learned in the previous sections.

Olympics data set is a publicly available data. Using this dataset, you will complete the **MedalsByCountry.java** and write you map and reduce classes to find the top 5 number of medals won by each country.

Before running your application, create a new HDFS directory **lab4\_in** and place the **olympix\_data.csv** file into it.

The result should be stored into the **lab4\_out** HDFS directory

### Data Set Description

The data set consists of the following fields:

<b>Athlete:</b>	Name of the athlete
<b>Age:</b>	Age of the athlete
<b>Country:</b>	The name of the country participating in Olympics
<b>Year:</b>	The year in which Olympics is conducted
<b>Closing Date:</b>	Closing date of Olympics
<b>Sport:</b>	Sports name
<b>Gold Medals:</b>	No. of gold medals
<b>Silver Medals:</b>	No. of silver medals
<b>Bronze Medals:</b>	No. of bronze medals
<b>Total Medals:</b>	Total no. of medals

### Compiling and running your application

- To compile your application, navigate to the directory where your java class is and enter this commands: (you can ignore warning messages)

```
$ export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

```
$ hadoop com.sun.tools.javac.Main *.java
```

- Create your java archive

```
$ jar cvf olympixmedals.jar *.class
```

- Run your application and report the top 5 countries (top 5 number of medals won by each country).

Hint: To get the expected result you need to read the file on HDFS and combine with the Linux **sort** and head commands.

## Exercise 5: Run MapReduce jobs using Python

In this exercise you will use the Hadoop streaming library to run MapReduce job written in Python.

You will use the **mapper.py** and **reducer.py** files to count the words from in the **5000-8.txt** file ("*Leonardo Da Vinci Notes*") downloaded earlier.

To work properly, this library expects to provide it with four parameters:

- a) the input directory



- b) the output directory
- c) the mapper executable file
- d) the reducer executable file.

1. Create a new directory **hadoopstream** on your local file system
2. Place the **mapper.py** and **reducer.py** file into this directory
3. Set the file owner to cloudera and grant execution permissions

```
$ sudo chown cloudera mapper.py  
$ sudo chown cloudera reducer.py  
$ sudo chmod +x mapper.py  
$ sudo chmod +x reducer.py
```

4. Create the **stream\_in** directory on HDFS
5. Put the file **5000-8.txt** into this directory
6. Enter the following command to perform the word count on the input file:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-  
cdh5.13.0.jar -input stream_in -output stream_out -mapper  
/home/cloudera/hadoopstream/mapper.py -reducer  
/home/cloudera/hadoopstream/reducer.py
```

7. View the 20 first lines of the result

---

<sup>i</sup> [https://en.wikipedia.org/wiki/Quasi-Monte\\_Carlo\\_method](https://en.wikipedia.org/wiki/Quasi-Monte_Carlo_method)