BIG DATA ANALYTICS LAB:

----

1. To Study of Big Data Analytics and Hadoop Architecture

(i) know the concept of big data architecture

(ii) know the concept of Hadoop architecture

(i) Know the concept of big data architecture

Big data tools and techniques demand special big data tools and techniques. When it comes to managing large quantities of data and performing complex operations on that massive data, big data tools and techniques must be used. The big data ecosystem and its sphere are what we refer to when we say using big data tools and techniques. There is no solution that is provided for every use case and that requires and has to be created and made in an effective manner according to company demands. A big data solution must be developed and maintained in accordance with company demands so that it meets the needs of the company. A stable big data solution can be constructed and maintained in such a way that it can be used for the requested problem.

Introduction

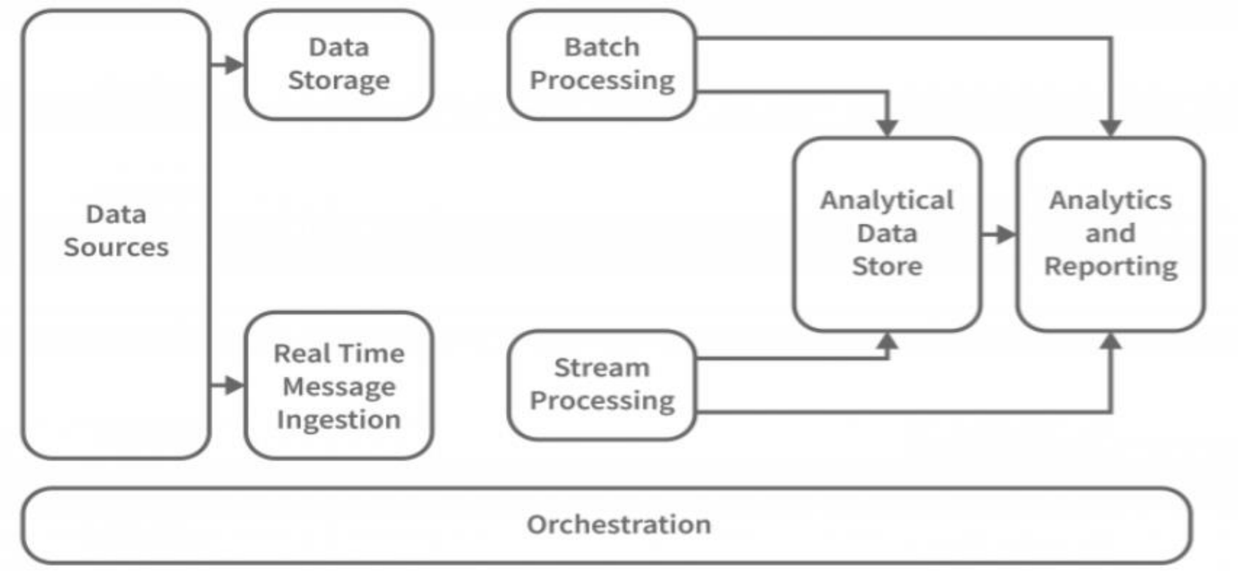
Big data architecture is a comprehensive solution to deal with an enormous amount of data. It details the blueprint for providing solutions and infrastructure for dealing with big data based on a company's demands. It clearly defines the components, layers, and methods of communication. The reference point is the ingestion, processing, storing, managing, accessing, and analysing of the data.

What is Big Data Architecture?

There is more than one workload type involved in big data systems, and they are broadly classified as follows:

1. Merely batching data where big data-based sources are at rest is a data processing situation.
2. Real-time processing of big data is achievable with motion-based processing.

3. The exploration of new interactive big data technologies and tools.
4. The use of machine learning and predictive analysis.



- The big data architectures include the following components:

✓ **Data sources**: All big data solutions start with one or more data sources.

Example,

- Application data stores, such as relational databases.
- Static files produced by applications, such as web server log files.
- Real-time data sources, such as IoT devices.

✓ **Data storage**: Data for batch processing operations is stored in a distributed file store that can hold high volumes of large files in various formats (also called data lake).

Example,

- Azure Data Lake Store or blob containers in Azure Storage.

✓ **Batch processing:** Since the data sets are so large, therefore a big data solution must process data files using long-running batch jobs to filter, aggregate, and prepare the data for analysis.

✓ **Real-time message ingestion:** If a solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing.

✓ **Stream processing:** After capturing real-time messages, the solution must process them by filtering, aggregating, and preparing the data for analysis. The processed stream data is then written to an output sink. We can use open-source Apache streaming technologies like Storm and Spark Streaming for this.

- ✓ **Analytical data store:** Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. Example: Azure Synapse Analytics provides a managed service for large-scale, cloud-based data warehousing.
- ✓ **Analysis and reporting:** The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modelling layer. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts.
- ✓ **Orchestration:** Most big data solutions consist of repeated data processing operations that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report.

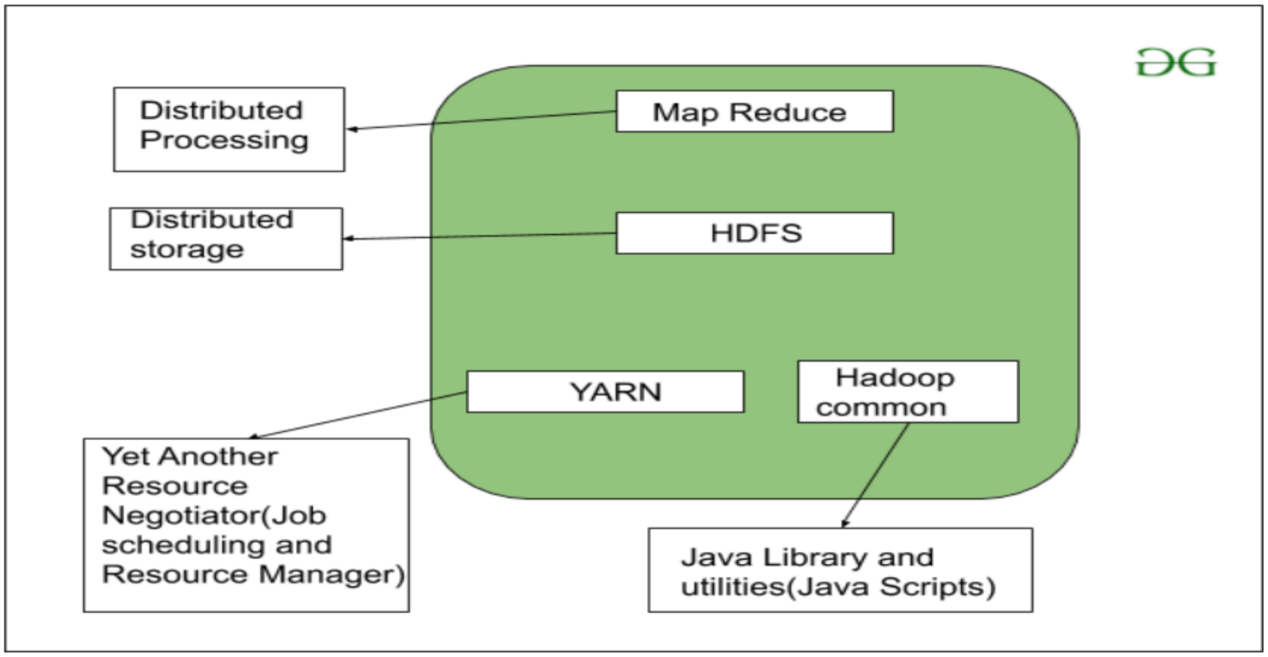
(ii) Know the concept of Hadoop architecture

Theory:

Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data. Hadoop works on MapReduce Programming Algorithm that was introduced by Google. Today lots of Big Brand Companies are using Hadoop in their Organization to deal with big data, eg. Facebook, Yahoo, Netflix, eBay, etc. The Hadoop Architecture Mainly consists of 4 components.

MapReduce

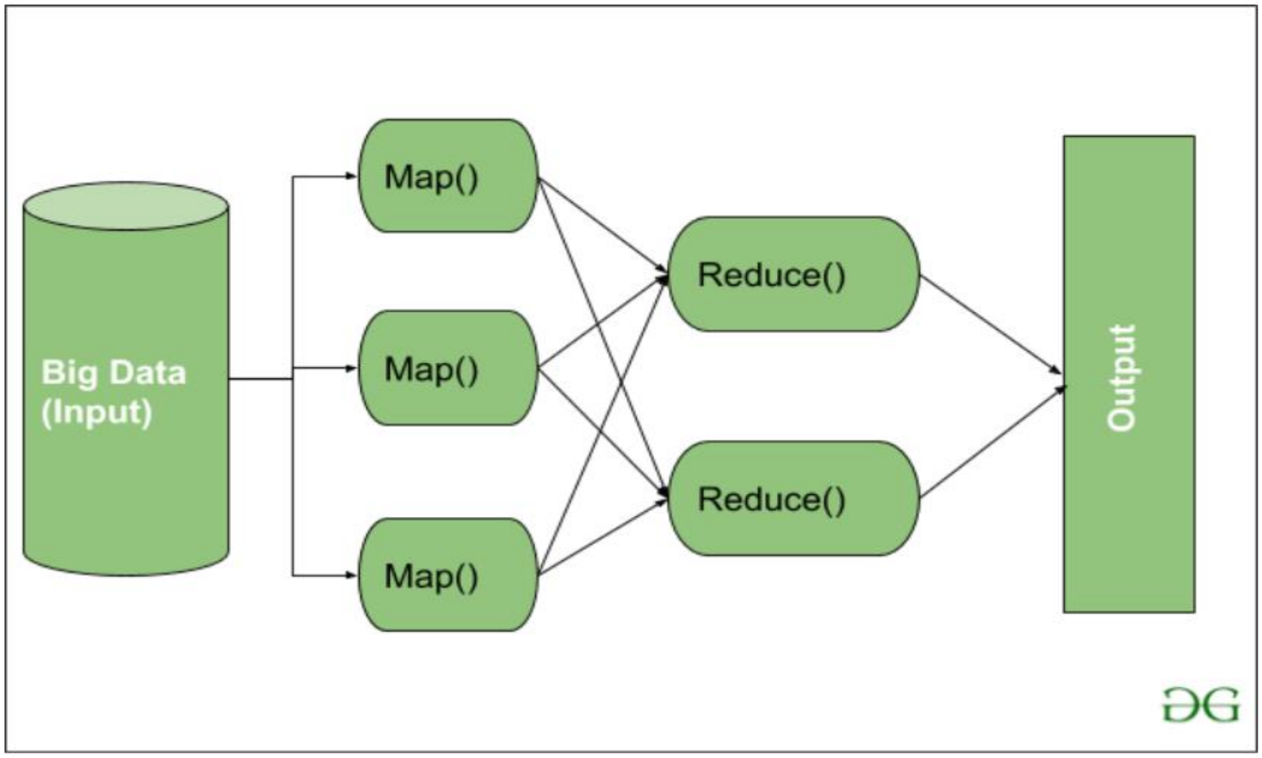
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common


Distributed Processing
Map Reduce
Distributed storage
HDFS
YARN
Hadoop common
Yet Another Resource Negotiator(Job scheduling and Resource Manager)
Java Library and utilities(Java Scripts)

1. MapReduce

MapReduce nothing but just like an Algorithm or a data structure that is based on the YARN framework. The major feature of MapReduce is to perform the distributed processing in parallel in a Hadoop cluster which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. MapReduce has mainly 2 tasks which are divided phase-wise:

In first phase, **Map** is utilized and in next phase **Reduce** is utilized.



Here, we can see that the *Input* is provided to the Map() function then it's *output* is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.

As we can see that an Input is provided to the Map(), now as we are using Big Data. The Input is a set of Data. The Map() function here breaks this DataBlocks into **Tuples** that are nothing but a key-value pair. These key-value pairs are now sent as input to the Reduce(). The Reduce() function then combines this broken Tuples or key-value pair based on its Key value and form set of Tuples, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.

The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

Let's understand the *Map Task* and *Reduce Task* in detail.

Map Task:

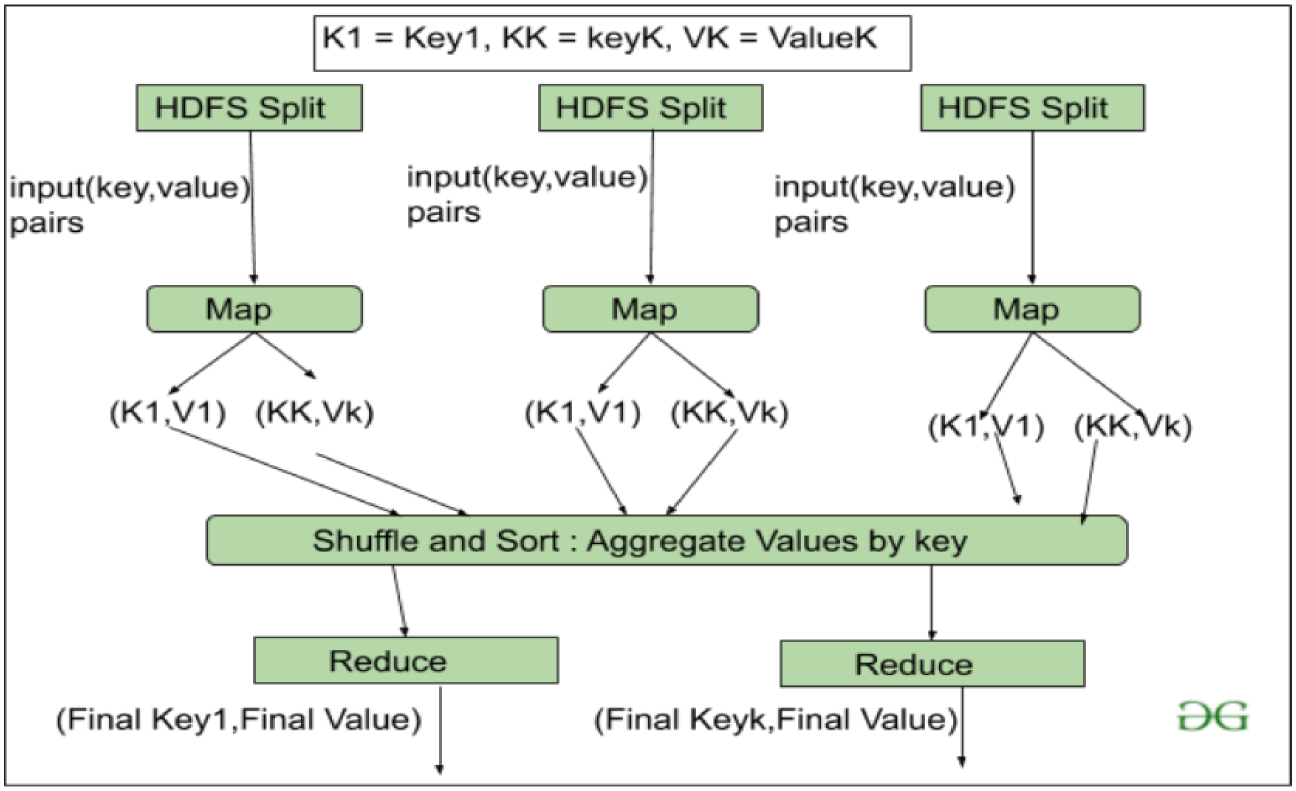
- **RecordReader** The purpose of *recordreader* is to break the records. It is responsible for providing key-value pairs in a Map() function. The key is actually is its locational information and value is the data associated with it.
- **Map:** A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.
- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map is combined with the help of this combiner. Using a combiner is not necessary as it is optional.
- **Partitionar:** Partitional is responsible for fetching key-value pairs generated in the Mapper Phases. The partitioner generates the shards corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers)).*

Reduce Task

- **Shuffle and Sort:** The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*. Using the Shuffling process the system can sort the data using its key value.

  Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.

- **Reduce:** The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-values depending on its key element.
- **Output Format:** Once all the operations are performed, the key-value pairs are written into the file with the help of record writer, each record in a new line, and the key and value in a space-separated manner.



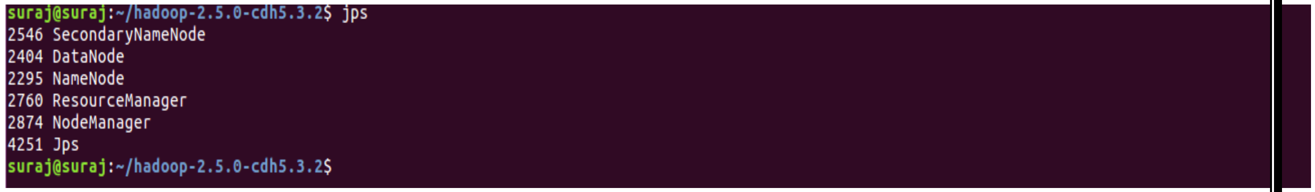
HDFS is the primary or major component of the Hadoop ecosystem which is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files. To use the HDFS commands, first you need to start the Hadoop services using the following command:

2)Aim: To perform the basic HDFS commands in Hadoop

Resources Required:Oracle virtual box,Hadoop

Procedure:

- startCDH.sh
- jps



```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ jps
2546 SecondaryNameNode
2404 DataNode
2295 NameNode
2760 ResourceManager
2874 NodeManager
4251 Jps
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

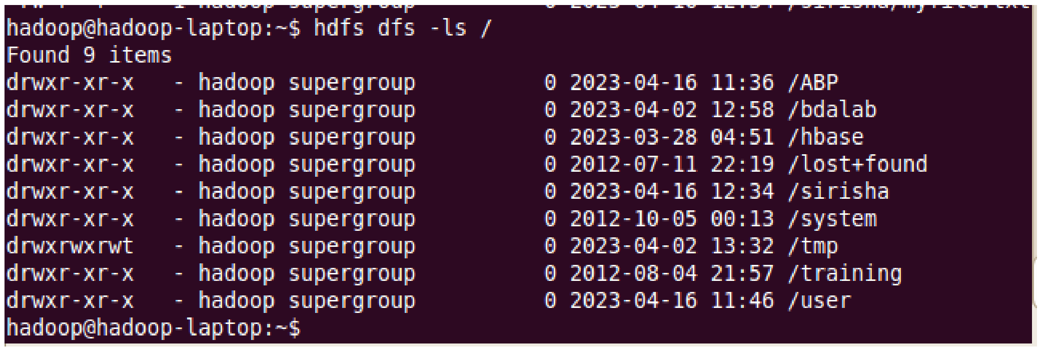
Commands:

ls: This command is used to list all the files. Use *lsr* for recursive approach. It is useful when we want a hierarchy of a folder.

Syntax:

```
hdfsdfs -ls  <path>
```

- `bin/hdfsdfs -ls /`



```
hadoop@hadoop-laptop:~$ hdfs dfs -ls /
Found 9 items
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 11:36 /ABP
drwxr-xr-x   - hadoop supergroup          0 2023-04-02 12:58 /bdalab
drwxr-xr-x   - hadoop supergroup          0 2023-03-28 04:51 /hbase
drwxr-xr-x   - hadoop supergroup          0 2012-07-11 22:19 /lost+found
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 12:34 /sirisha
drwxr-xr-x   - hadoop supergroup          0 2012-10-05 00:13 /system
drwxrwxrwt   - hadoop supergroup          0 2023-04-02 13:32 /tmp
drwxr-xr-x   - hadoop supergroup          0 2012-08-04 21:57 /training
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 11:46 /user
hadoop@hadoop-laptop:~$
```

2.mkdir: To create a directory. In Hadoop *dfs* there is no home directory by default. So let's first create it.

Syntax:

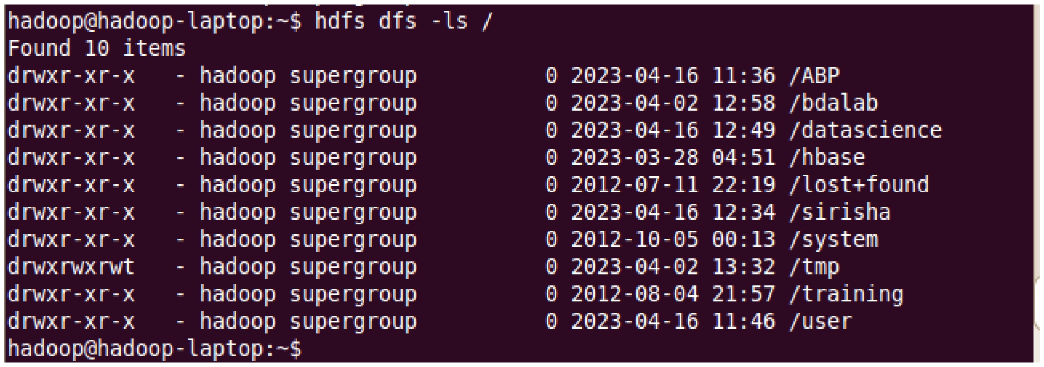
```
hdfsdfs -mkdir<folder name>


creating home directory:

hdfs/bin -mkdir /user

hdfs/bin -mkdir /user/username -> write the username of your computer
```

Example:

- `hdfsdfs -mkdir  /datascience  =>  '/' means absolute path`

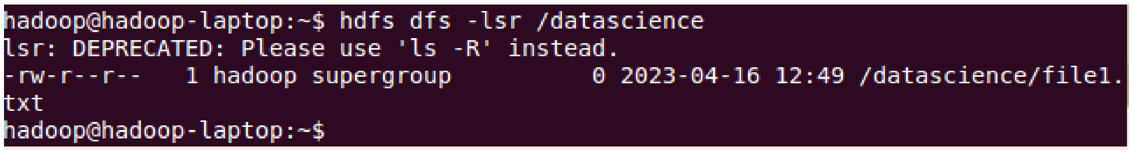
3.touchz: It creates an empty file.

- **Syntax:**
- **`hdfsdfs  -touchz<file_path>`**
- `hdfsdfs -touchz /datascience/file1.txt`
- `hdfsdfs -ls /`



```
hadoop@hadoop-laptop:~$ hdfs dfs -ls /
Found 10 items
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 11:36 /ABP
drwxr-xr-x   - hadoop supergroup          0 2023-04-02 12:58 /bdalab
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 12:49 /datascience
drwxr-xr-x   - hadoop supergroup          0 2023-03-28 04:51 /hbase
drwxr-xr-x   - hadoop supergroup          0 2012-07-11 22:19 /lost+found
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 12:34 /sirisha
drwxr-xr-x   - hadoop supergroup          0 2012-10-05 00:13 /system
drwxrwxrwt   - hadoop supergroup          0 2023-04-02 13:32 /tmp
drwxr-xr-x   - hadoop supergroup          0 2012-08-04 21:57 /training
drwxr-xr-x   - hadoop supergroup          0 2023-04-16 11:46 /user
hadoop@hadoop-laptop:~$
```

- `hdfsdfs -lsr /datascience`



```
hadoop@hadoop-laptop:~$ hdfs dfs -lsr /datascience
lsr: DEPRECATED: Please use 'ls -R' instead.
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 12:49 /datascience/file1.
txt
hadoop@hadoop-laptop:~$
```

4.copyFromLocal (or) put: To copy files/folders from local file system to hdfs store. This is the most important command. Local filesystem means the files present on the OS.

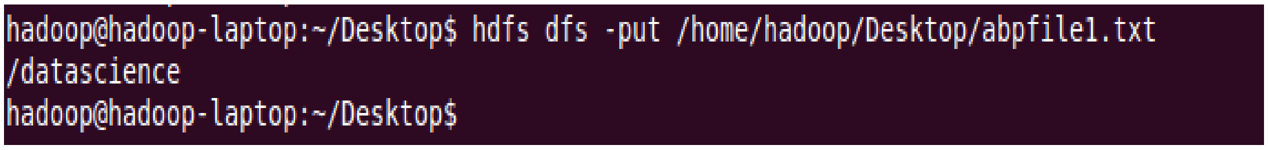
Syntax:

- **`hdfsdfs -copyFromLocal<local file path><dest(present on hdfs)>`**

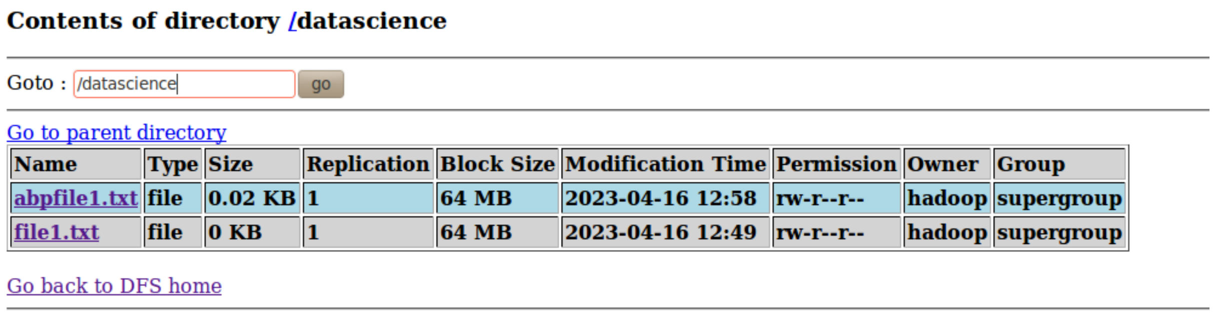
Example: Let's suppose we have a file *abpfile1.txt* on Desktop which we want to copy to folder *datascience* present on hdfs.

```
hdfsdfs -copyFromLocal ../Desktop/abpfile1.txt /datascience


(OR)
```

- `Pwd    to know the present working directory`
- `Cd Desktop`
- `ls`
- `hdfsdfs -put /home/hadoop/Desktop/abpfile1.txt /datascience`



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -put /home/hadoop/Desktop/abpfile1.txt
/datascience
hadoop@hadoop-laptop:~/Desktop$
```



Contents of directory /datascience

Goto : /datascience go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
abpfile1.txt	file	0.02 KB	1	64 MB	2023-04-16 12:58	rw-r--r--	hadoop	supergroup
file1.txt	file	0 KB	1	64 MB	2023-04-16 12:49	rw-r--r--	hadoop	supergroup

Go back to DFS home

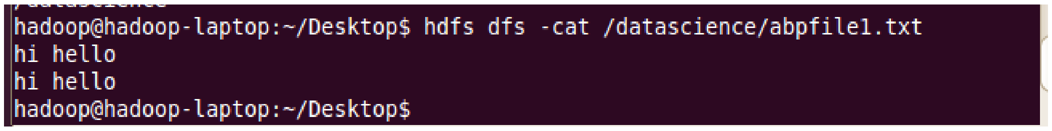
5.cat: To print file contents.

Syntax:

```
hdfsdfs -cat <path>
```

Example:

```
// print the content of abpfile1.txt present

// inside datascience folder.

hdfsdfs -cat /datascience/abpfile1.txt ->
```



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -cat /datascience/abpfile1.txt
hi hello
hi hello
hadoop@hadoop-laptop:~/Desktop$
```

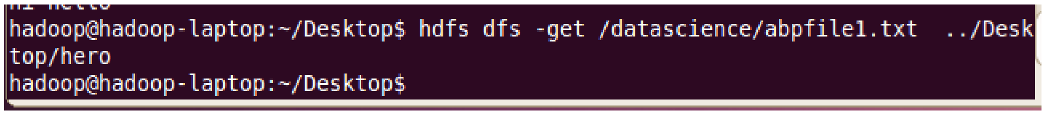
6.copyToLocal (or) get: To copy files/folders from hdfs store to local file system.

Syntax:

```
hdfsdfs -copyToLocal<<srcfile(on hdfs)><local file dest>
```

Example:

```
hdfsdfs -copyToLocal  /geeks   ../Desktop/hero

(OR)

hdfsdfs -get /datascience/abpfile1.txt  ../Desktop/hero
```



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -get /datascience/abpfile1.txt  ../Desk
top/hero
hadoop@hadoop-laptop:~/Desktop$
```

abpfile1.txt from *datascience* folder will be copied to folder *hero* present on *Desktop*.

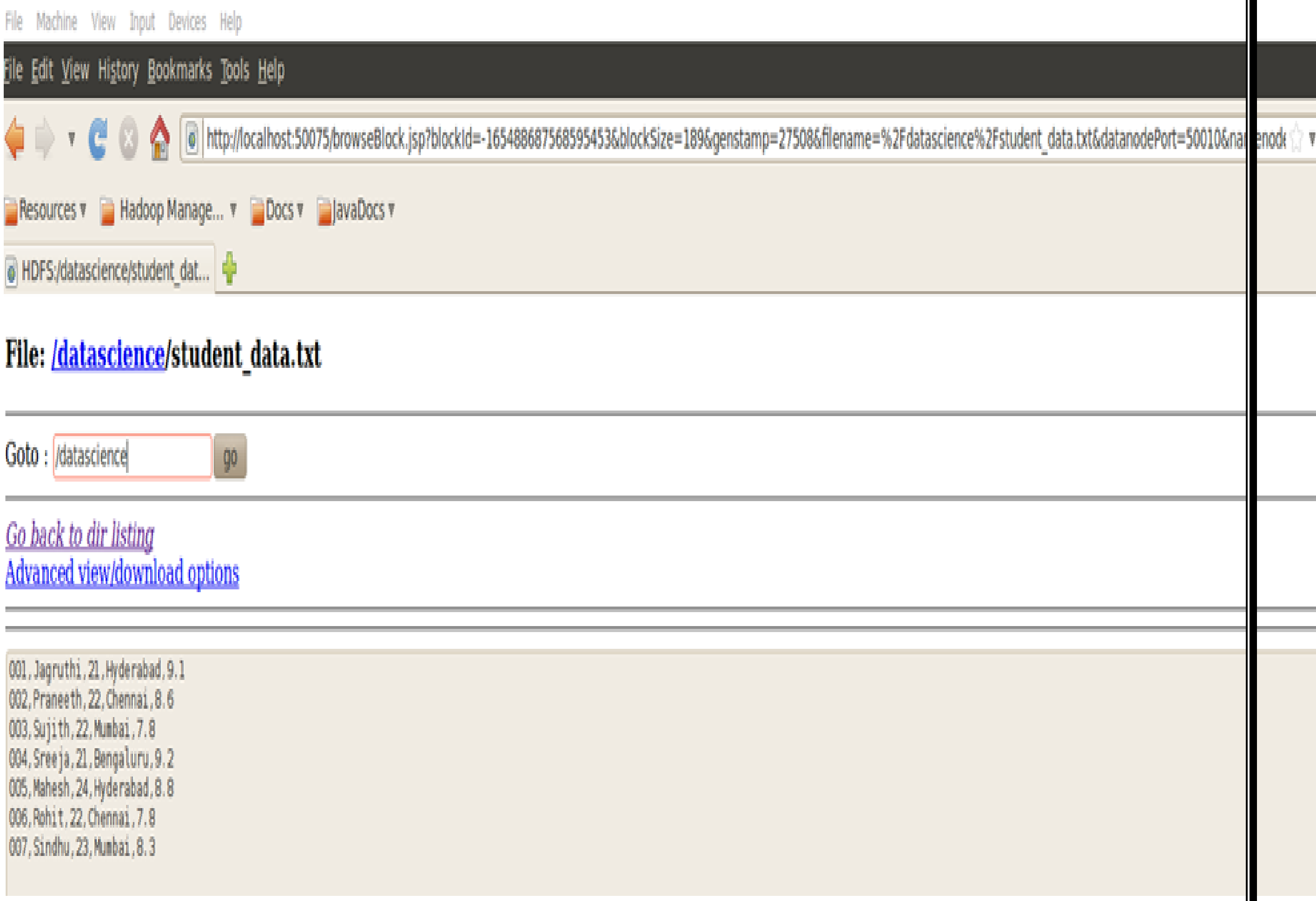
7.moveFromLocal: This command will move file from local to hdfs.

Syntax:

```
hdfsdfs -moveFromLocal<local src><dest(on hdfs)>
```

Example:

- ➢ `hdfsdfs -moveFromLocal  ../Desktop/student_data.txt /datascience`


File Machine View Input Devices Help
File Edit View History Bookmarks Tools Help
http://localhost:50075/browseBlock.jsp?blockId=-1654886875685954538&blockSize=189&genstamp=27508&filename=%2Fdatascience%2Fstudent_data.txt&datanodePort=50010&na
Resources Hadoop Manage... Docs JavaDocs
HDFS:/datascience/student_dat...

File: /datascience/student_data.txt

Goto : /datascience go

Go back to dir listing
Advanced view/download options

```
001,Jagruthi,21,Hyderabad,9.1
002,Praneeth,22,Chennai,8.6
003,Sujith,22,Mumbai,7.8
004,Sreeja,21,Bengaluru,9.2
005,Mahesh,24,Hyderabad,8.8
006,Rohit,22,Chennai,7.8
007,Sindhu,23,Mumbai,8.3
```

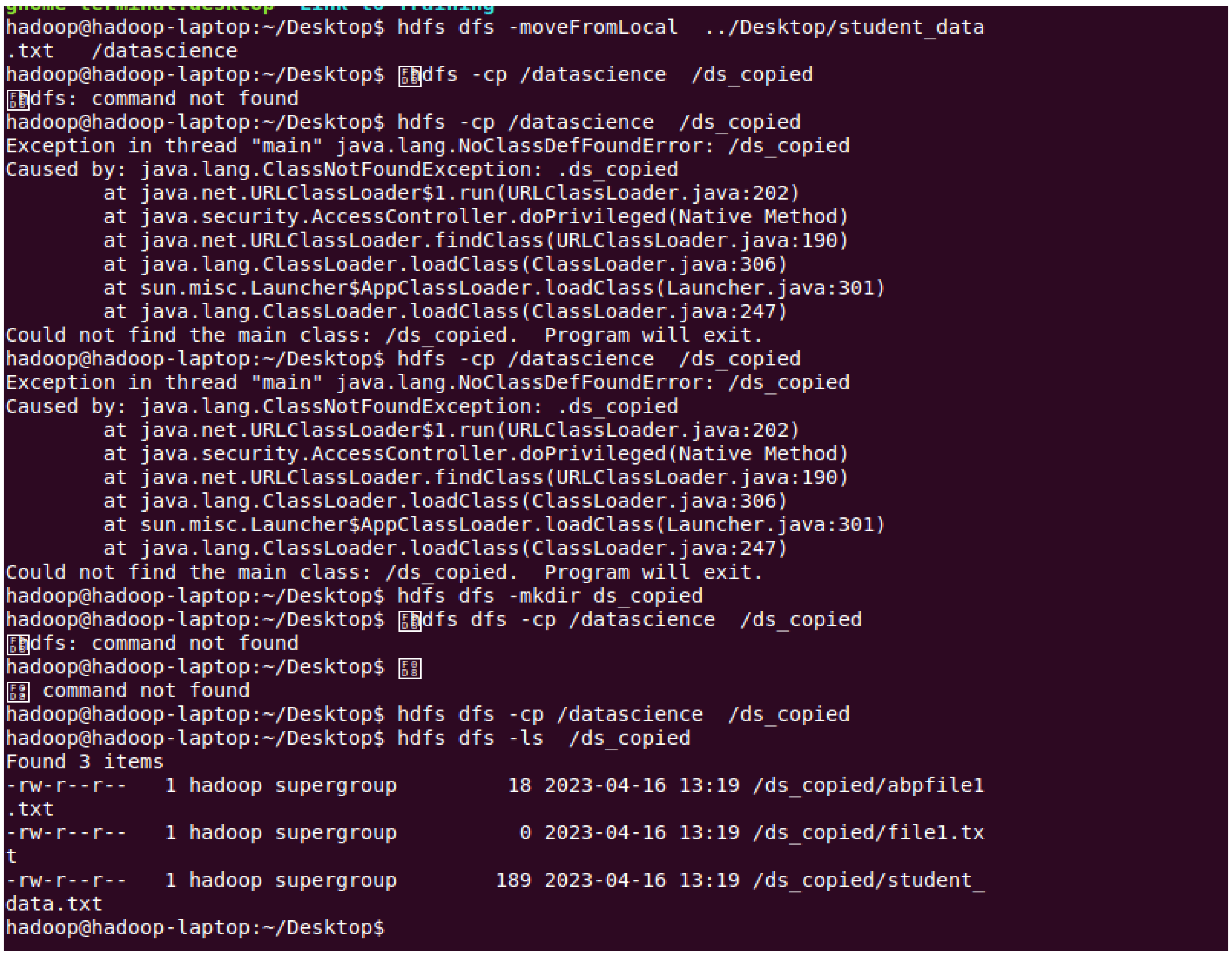
8.cp: This command is used to copy files within hdfs. Lets copy folder *datascience* to *ds_copied*.

Syntax:

- ➢ `hdfsdfs -cp  <src(on hdfs)><dest(on hdfs)>`

Example:

- ➢ **hdfsdfs -mkdirds_copied**
- ➢ **hdfsdfs-cp /datascience /ds_copied**
- ➢ **hdfsdfs -ls /ds_copied**

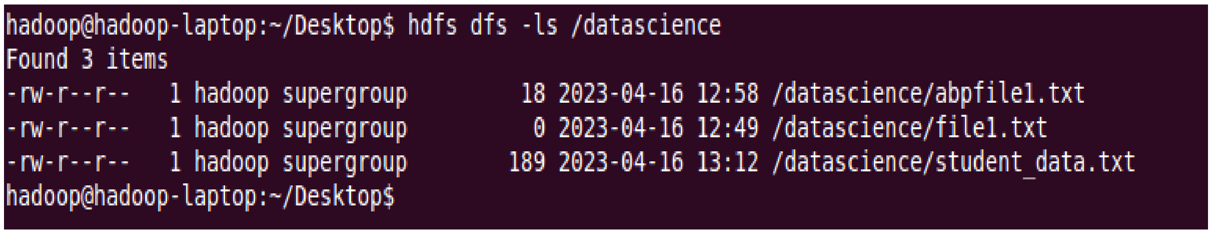


```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -moveFromLocal  ../Desktop/student_data
.txt   /datascience
hadoop@hadoop-laptop:~/Desktop$ hdfs -cp /datascience  /ds_copied
hdfs: command not found
hadoop@hadoop-laptop:~/Desktop$ hdfs -cp /datascience  /ds_copied
Exception in thread "main" java.lang.NoClassDefFoundError: /ds_copied
Caused by: java.lang.ClassNotFoundException: .ds_copied
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: /ds_copied.  Program will exit.
hadoop@hadoop-laptop:~/Desktop$ hdfs -cp /datascience  /ds_copied
Exception in thread "main" java.lang.NoClassDefFoundError: /ds_copied
Caused by: java.lang.ClassNotFoundException: .ds_copied
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: /ds_copied.  Program will exit.
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -mkdir ds_copied
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -cp /datascience  /ds_copied
hdfs: command not found
hadoop@hadoop-laptop:~/Desktop$ 
 command not found
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -cp /datascience  /ds_copied
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls  /ds_copied
Found 3 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 13:19 /ds_copied/abpfile1
.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:19 /ds_copied/file1.tx
t
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:19 /ds_copied/student_
data.txt
hadoop@hadoop-laptop:~/Desktop$
```

9.mv: This command is used to move files within hdfs. Lets cut-paste a file *abpfile1.txt* from *datascience* folder to *ds_copied*.

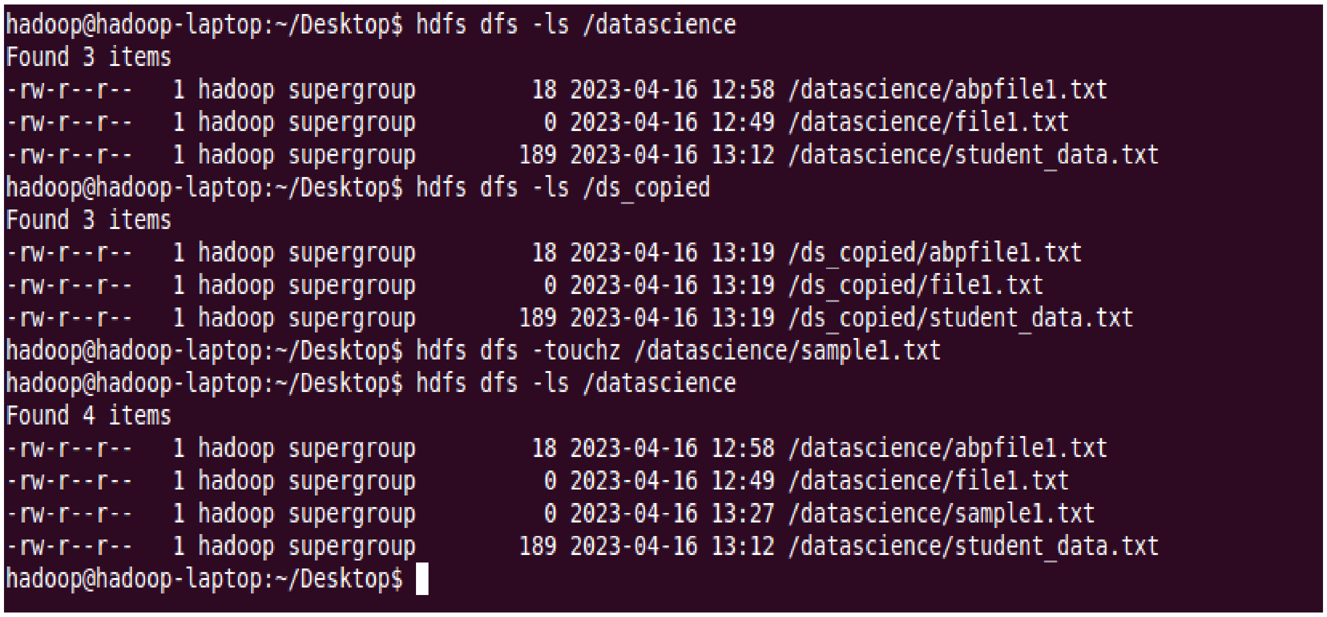
Syntax:

```
hdfsdfs -mv  <src(on hdfs)><src(on hdfs)>
```



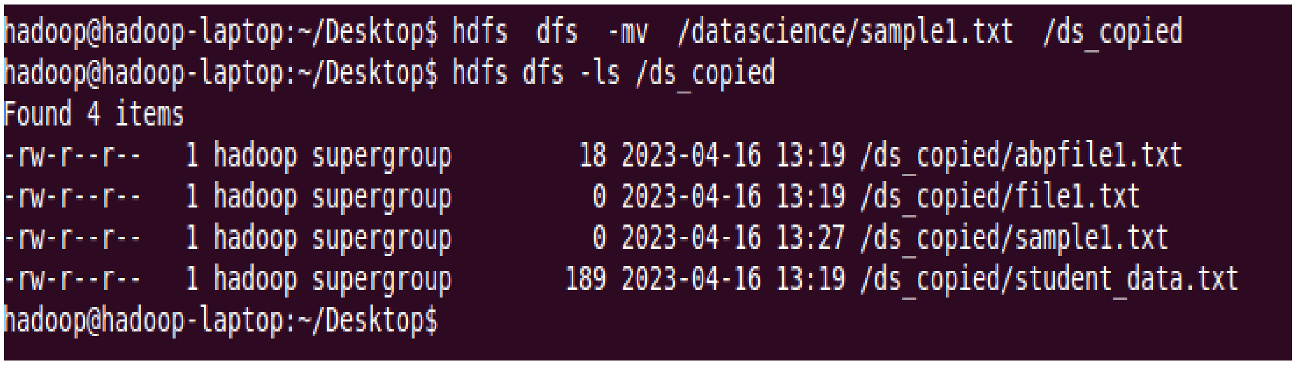
```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /datascience
Found 3 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 12:58 /datascience/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 12:49 /datascience/file1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:12 /datascience/student_data.txt
hadoop@hadoop-laptop:~/Desktop$
```

➢ `hdfsdfs -touchz /datascience/sample1.txt`
➢ `hdfsdfs -ls /datascience`



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /datascience
Found 3 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 12:58 /datascience/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 12:49 /datascience/file1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:12 /datascience/student_data.txt
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /ds_copied
Found 3 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 13:19 /ds_copied/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:19 /ds_copied/file1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:19 /ds_copied/student_data.txt
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -touchz /datascience/sample1.txt
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /datascience
Found 4 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 12:58 /datascience/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 12:49 /datascience/file1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:27 /datascience/sample1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:12 /datascience/student_data.txt
hadoop@hadoop-laptop:~/Desktop$ 
```

- `hdfsdfs  -mv  /datascience/sample1.txt  /ds_copied`
- `hdfsdfs -ls /ds_copied`



```
hadoop@hadoop-laptop:~/Desktop$ hdfs  dfs  -mv  /datascience/sample1.txt  /ds_copied
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /ds_copied
Found 4 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 13:19 /ds_copied/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:19 /ds_copied/file1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:27 /ds_copied/sample1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:19 /ds_copied/student_data.txt
hadoop@hadoop-laptop:~/Desktop$
```

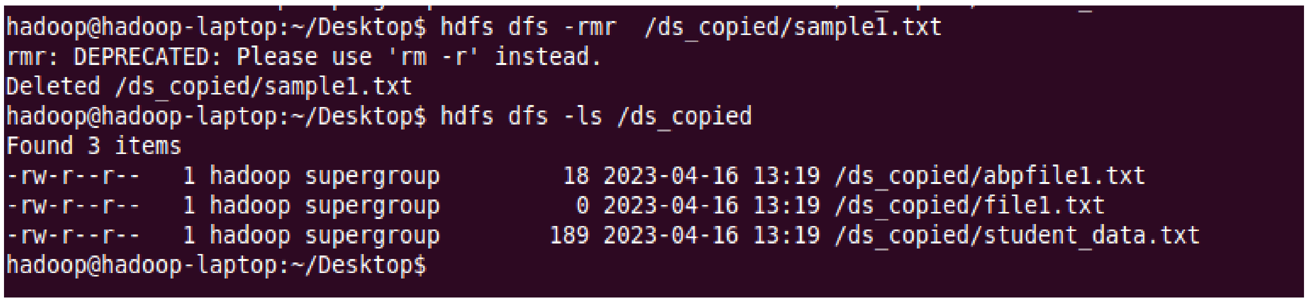
10.rmr: This command deletes a file from HDFS *recursively*. It is very useful command when you want to delete a *non-empty directory*.

Syntax:

`hdfsdfs -rmr<filename/directoryName>`

Example:

- `hdfsdfs -rmr  /ds_copied/sample1.txt`
- `hdfsdfs -ls /ds_copied`

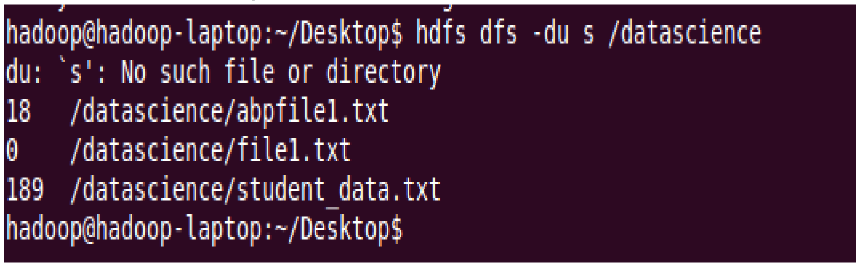


```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -rmr  /ds_copied/sample1.txt
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted /ds_copied/sample1.txt
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -ls /ds_copied
Found 3 items
-rw-r--r--   1 hadoop supergroup         18 2023-04-16 13:19 /ds_copied/abpfile1.txt
-rw-r--r--   1 hadoop supergroup          0 2023-04-16 13:19 /ds_copied/file1.txt
-rw-r--r--   1 hadoop supergroup        189 2023-04-16 13:19 /ds_copied/student_data.txt
hadoop@hadoop-laptop:~/Desktop$
```

11.dus:: This command will give the total size of directory/file.
Syntax:
`hdfsdfs -dus<dirName>`

Example:

- ➢ `hdfsdfs -du s /datascience`



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -du s /datascience
du: `s': No such file or directory
18   /datascience/abpfile1.txt
0    /datascience/file1.txt
189  /datascience/student_data.txt
hadoop@hadoop-laptop:~/Desktop$
```

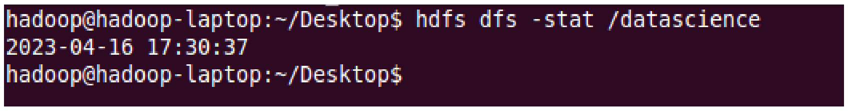
- ➢

12.stat: It will give the last modified time of directory or path. In short it will give stats of the directory or file.
Syntax:

- ➢ `hdfsdfs -stat    <hdfs file>`

Example:

- ➢ `hdfsdfs -stat /datascience`



```
hadoop@hadoop-laptop:~/Desktop$ hdfs dfs -stat /datascience
2023-04-16 17:30:37
hadoop@hadoop-laptop:~/Desktop$
```

13. setrep: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs *core-site.xml*).

Example 1: To change the replication factor to 6 for *geeks.txt* stored in HDFS.

- ➢ `hdfsdfs -setrep -R -w 6 datascience`

Example 2: To change the replication factor to 4 for a directory *geeksInput* stored in HDFS.

- ➢ `hdfsdfs -setrep -R  4 /datascience`

Note: The **-w** means wait till the replication is completed. And **-R** means recursively, we use it for directories as they may also contain many files and folders inside them

3) **Write a program to create word count java program code using mapreduce in hadoop framework**

HADOOP MAP-REDUCE PROGRAM

1. Switch to the Hadoop user (user containing installed Hadoop instance)

2. Open the Terminal, and type the following command:

    nano ~/.bashrc

```
hadoop@MSI:~$ nano ~/.bashrc
```

3. At the bottom of the file, paste the following line. Please ensure JAVA_HOME and HADOOP_HOME env variable is set before this command:

export HADOOP_CLASSPATH=$(find $HADOOP_HOME -name '*.jar' | xargs echo | tr ' ' ':')

```
export HADOOP_CLASSPATH=$(find $HADOOP_HOME -name '*.jar' | xargs echo | tr ' ' ':')
```

4. Save and exit the file.

5. In the terminal, type the following command:

    source ~/.bashrc

```
hadoop@MSI:~$   source ~/.bashrc
```

6. Now, start all Hadoop daemons:

    start-all.sh

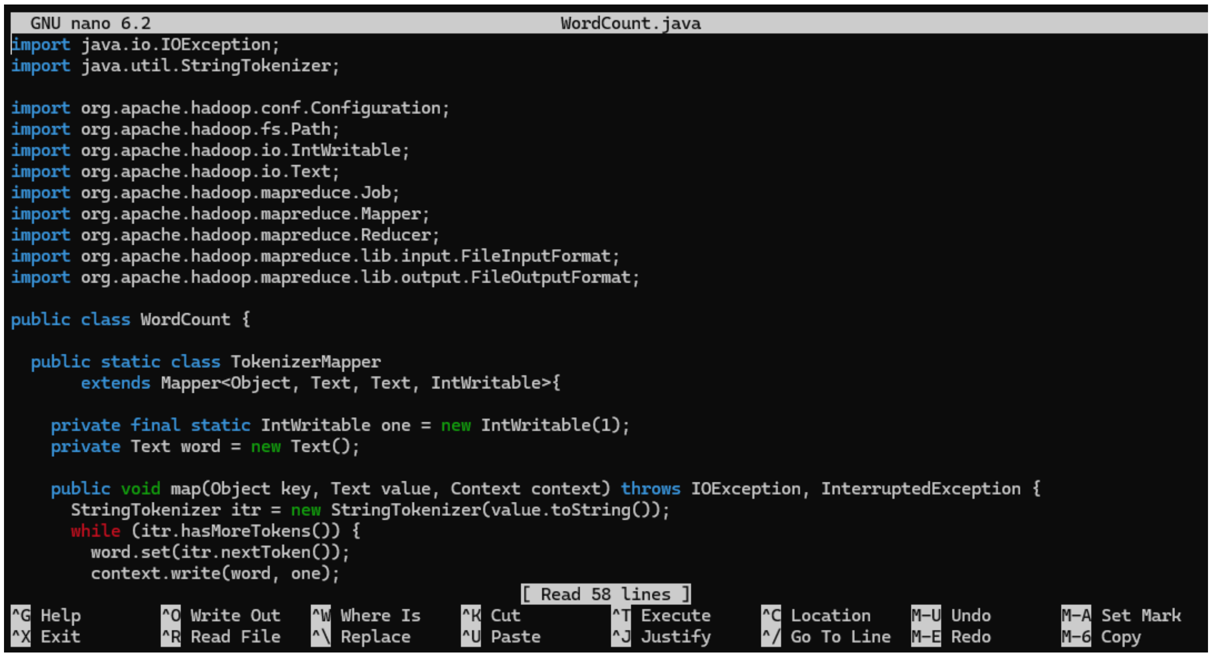
7. open a new java file using nano

    nano WordCount.java

```
hadoop@MSI:~$ nano WordCount.java
```

8. The nano text editor will open. Type the following java code for WordCount program:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount          //driver class
{    //Mapper class
     public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
     {  private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws IOException,
        InterruptedException
        {  StringTokenizeritr = new StringTokenizer(value.toString());
           while (itr.hasMoreTokens())
           {     word.set(itr.nextToken());
                 context.write(word, one);
           }
        }
     } //End Mapper class

     //Reducer class
     public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>
     {  private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
        IOException, InterruptedException
        {  int sum = 0;
           for (IntWritableval : values)
           {     sum += val.get();
           }
           result.set(sum);
           context.write(key, result);
        }
     } //End Reducer class
     //Driver class main()
     public static void main(String[] args) throws Exception
     {  Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
     } //End main()
} //End WordCount class
```

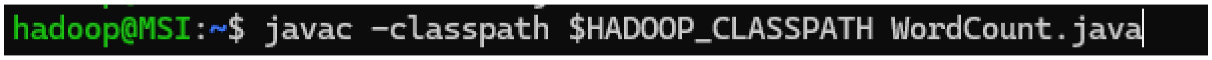


```
  GNU nano 6.2                                   WordCount.java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
                                           [ Read 58 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute  ^C Location   M-U Undo   M-A Set Mark
^X Exit      ^R Read File  ^\ Replace   ^U Paste    ^J Justify  ^/ Go To Line M-E Redo   M-6 Copy
```

9. Save and exit the nano editor

10. In the terminal, type the following command:

    javac -classpath $HADOOP_CLASSPATH WordCount.java

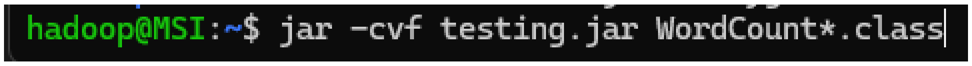


```
hadoop@MSI:~$ javac -classpath $HADOOP_CLASSPATH WordCount.java
```

11. The above line should execute without any errors.

12. Type the following command to compile into JAR file

    jar -cvf testing.jar WordCount*.class



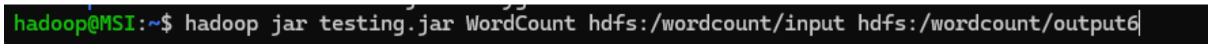
```
hadoop@MSI:~$ jar -cvf testing.jar WordCount*.class
```

13. The “testing.jar” file is created

14. Create one input folder in HDFS. In the input folder, create a testing text file to serve as input for the mapreduce program

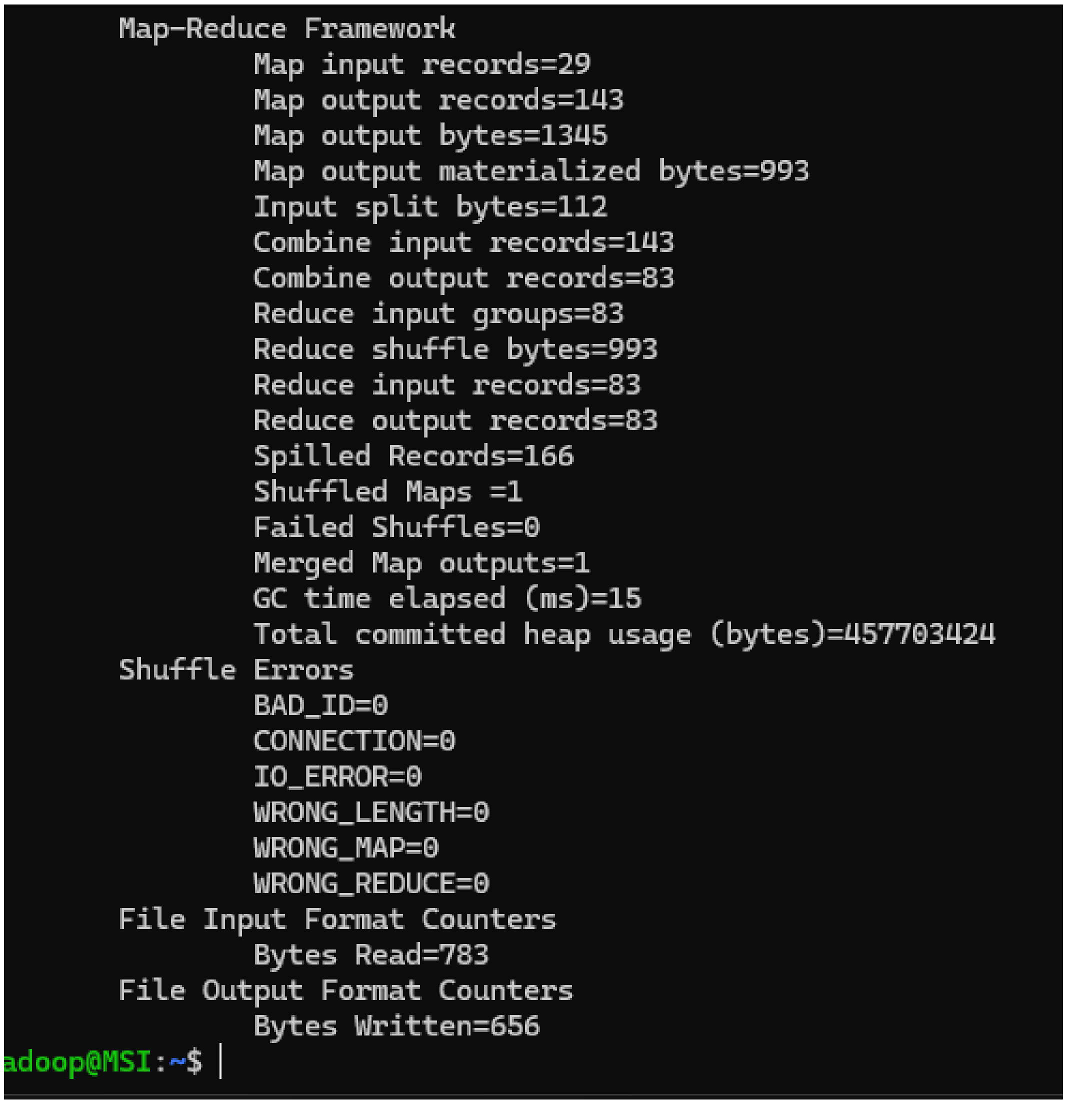
15. Run the following command to execute the mapreduce program. Replace the first argument with the input folder path on HDFS:

    hadoop jar testing.jar WordCounthdfs:/wordcount/input hdfs:/wordcount/output



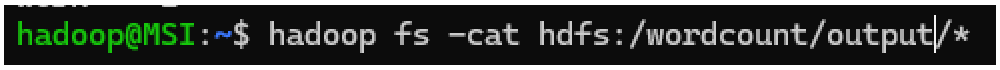
```
hadoop@MSI:~$ hadoop jar testing.jar WordCount hdfs:/wordcount/input hdfs:/wordcount/output6
```

16. The command should execute without errors:


```
        Map-Reduce Framework
                Map input records=29
                Map output records=143
                Map output bytes=1345
                Map output materialized bytes=993
                Input split bytes=112
                Combine input records=143
                Combine output records=83
                Reduce input groups=83
                Reduce shuffle bytes=993
                Reduce input records=83
                Reduce output records=83
                Spilled Records=166
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=15
                Total committed heap usage (bytes)=457703424
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=783
        File Output Format Counters
                Bytes Written=656
adoop@MSI:~$ 
```

17. To view the output, type the following command:

```
hadoop fs -cat hdfs:/wordcount/output/*
```

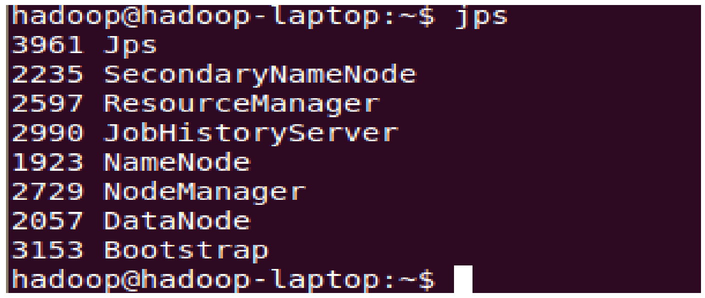

```
hadoop@MSI:~$ hadoop fs -cat hdfs:/wordcount/output/*
```

The output of the program will be visible.

4)Hbase commands

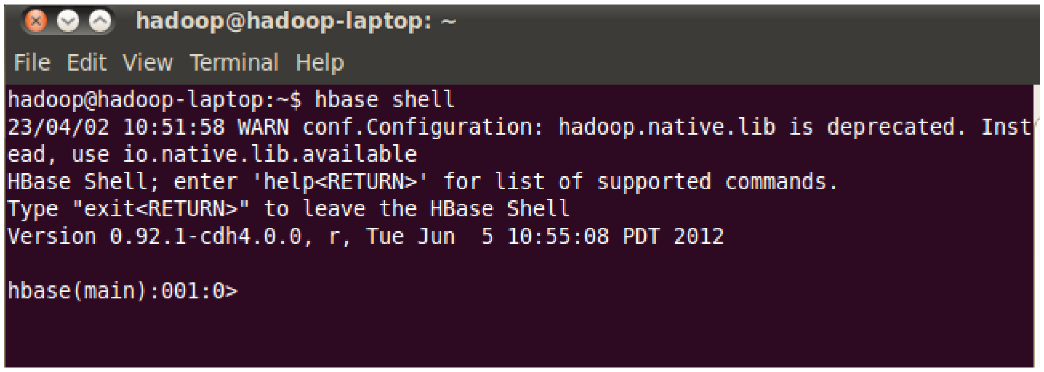
Step 1:First go to terminal and type **StartCDH.sh**

Step 2:Next type**jps** command in the terminal



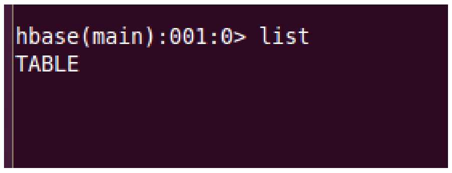
```
hadoop@hadoop-laptop:~$ jps
3961 Jps
2235 SecondaryNameNode
2597 ResourceManager
2990 JobHistoryServer
1923 NameNode
2729 NodeManager
2057 DataNode
3153 Bootstrap
hadoop@hadoop-laptop:~$
```

Step 3:Type**hbase shell**



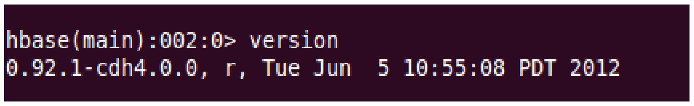
```
hadoop@hadoop-laptop:~$ hbase shell
23/04/02 10:51:58 WARN conf.Configuration: hadoop.native.lib is deprecated. Inst
ead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.1-cdh4.0.0, r, Tue Jun  5 10:55:08 PDT 2012

hbase(main):001:0>
```

List will gives you list of tables in Hbase



```
hbase(main):001:0> list
TABLE
```

Step 5:hbase(main):001:0>**version**



```
hbase(main):002:0> version
0.92.1-cdh4.0.0, r, Tue Jun  5 10:55:08 PDT 2012
```

Version will gives you the version of hbase

Create Table Syntax

CREATE 'name_space:table_name', 'column_family’

hbase(main):011:0> create 'newtbl','knowledge'

```
hbase(main):011:0>describe 'newtbl'

hbase(main):011:0>status

1 servers, 0 dead, 15.0000 average load
```

HBase – Using PUT to Insert data to Table

To insert data into the HBase table use PUT command, this would be similar to insert statement on RDBMS but the syntax is completely different. In this article I will describe how to insert data into HBase table with examples using PUT command from the HBase shell.

HBase PUT command syntax

Below is the syntax of PUT command which is used to insert data (rows and columns) into a HBase table.

HBase PUT command syntax

Below is the syntax of PUT command which is used to insert data (rows and columns) into a HBase table.

```
put '<name_space:table_name>', '<row_key>' '<cf:column_name>', '<value>'

hbase(main):015:0>put 'newtbl','r1','knowledge:sports','cricket'

0 row(s) in 0.0150 seconds

hbase(main):016:0>put 'newtbl','r1','knowledge:science','chemistry'

0 row(s) in 0.0040 seconds

hbase(main):017:0>put 'newtbl','r1','knowledge:science','physics'

0 row(s) in 0.0030 seconds

hbase(main):018:0>put 'newtbl','r2','knowledge:economics','macroeconomics'

0 row(s) in 0.0030 seconds

hbase(main):019:0>put 'newtbl','r2','knowledge:music','songs'

0 row(s) in 0.0170 seconds
```

```
hbase(main):020:0>scan 'newtbl'

ROW                COLUMN+CELL

 r1          column=knowledge:science, timestamp=1678807827189, value =physics

 r1          column=knowledge:sports, timestamp=1678807791753, value= cricket

 r2          column=knowledge:economics, timestamp=1678807854590, value=macroeconomics

 r2          column=knowledge:music, timestamp=1678807877340, value=songs

2 row(s) in 0.0250 secondsdisable
```

To retrieve only the row1 data

```
hbase(main):023:0>get 'newtbl', 'r1'
```

output

```
COLUMN              CELL

knowledge:science    timestamp=1678807827189, value=physics

knowledge:sports     timestamp=1678807791753, value=cricket

2 row(s) in 0.0150 seconds.
```

```
hbase(main):025:0> disable 'newtbl'
0 row(s) in 1.2760 seconds
```

Verification

After disabling the table, you can still sense its existence through **list** and **exists** commands. You cannot scan it. It will give you the following error.

```
hbase(main):028:0> scan 'newtbl'
ROW        COLUMN + CELL
ERROR: newtbl is disabled.
```

is_disabled

This command is used to find whether a table is disabled. Its syntax is as follows.

```
hbase>is_disabled 'table name'
```

```
hbase(main):031:0>is_disabled 'newtbl'
true
0 row(s) in 0.0440 seconds
```

disable_all

This command is used to disable all the tables matching the given regex. The syntax for **disable_all** command is given below.

```
hbase>disable_all 'r.*'
```

Suppose there are 5 tables in HBase, namely raja, rajani, rajendra, rajesh, and raju. The following code will disable all the tables starting with **raj.**

```
hbase(main):002:07>disable_all 'raj.*'
raja
rajani
rajendra
rajesh
raju
Disable the above 5 tables (y/n)?
y
5 tables successfully disabled
```

Enabling a Table using HBase Shell

Syntax to enable a table:

```
enable 'newtbl'
```

Example

Given below is an example to enable a table.

```
hbase(main):005:0> enable 'newtbl'
0 row(s) in 0.4580 seconds
```

Verification

After enabling the table, scan it. If you can see the schema, your table is successfully enabled.

```
hbase(main):006:0> scan 'newtbl'
```

is_enabled

This command is used to find whether a table is enabled. Its syntax is as follows:

```
hbase>is_enabled 'table name'
```

The following code verifies whether the table named **emp** is enabled. If it is enabled, it will return true and if not, it will return false.

```
hbase(main):031:0>is_enabled 'newtbl'
true
0 row(s) in 0.0440 seconds
```

describe

This command returns the description of the table. Its syntax is as follows:

```
hbase> describe 'table name'
```

```
hbase(main):006:0> describe 'newtbl'
  DESCRIPTION
```

ENABLED

5)Hive Data Definition Language(DDL) Commands
ImplementData Definition Language (DDL) Commandsfor databases in Hadoop Hiveframework using Cloudera.

DDL Commands for Databases

1) **CREATE** database Statement is used to create a database in Hive. A database in Hive is a namespace or a collection or catalog of tables.

   Syntax: **CREATE DATABASE|SCHEMA [IF NOT EXISTS] database_name**

   [COMMENT database_comment]

   [LOCATION hdfs_path]

   [WITH DBPROPERTIES (property_name=property_value, ...)];

   [ ] are optional clauses. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named employee. If everything went good, you will see a 'OK' message, else you will see relevant error message.

   Simple creation

   ```
   hive> CREATE DATABASE facultycse;

   OK

   Time taken: 0.033 seconds

   hive> CREATE DATABASE facultyece;
   ```

   Full creation

   ```
   hive> CREATE DATABASE IF NOT EXISTS employee COMMENT ‘this is employee
         database’ LOCATION ‘/user/hive/warehouse/hivedir/’ WITH DBPROPERTIES
         (‘creator’=‘Bhanu’, ‘date’=‘2020-12-07’);
   ```

2) SHOW databases statement lists all the databases present in the metastore.

Syn: **SHOW (DATABASES/SCHEMAS) [LIKE ‘wildcards'];**

- Wildcards in the regular expression can only be '*' for any character(s) or '|' for a choice. Examples are 'employees', 'emp*', 'emp*|*ees', all of which will match the database named 'employees’:

hive> SHOW DATABASES; default employee facultycse facultyece	hive> SHOW DATABASES LIKE ‘*ee’; employee
	hive> SHOW DATABASES LIKE ‘fac*’; facultycse facultyece

3) DESCRIBE database statement in Hive shows the name of Database in Hive, its comment (if set), its location, its owner name, owner type and its properties.

Syn: **DESCRIBE DATABASE/SCHEMA [EXTENDED] db_name;**

- EXTENDED can be used to get the database properties.

```
hive>DESCRIBE DATABASE facultycse;

facultycse hdfs://quickstart.cloudera:8020/user/hive/warehouse/faculty.db cloudera USER

hive>DESCRIBE DATABASE EXTENDED employee;

employee this is employee database hdfs://quickstart.cloudera:8020/user/hive/warehouse/
cloudera  USER  {date=2020-12-07, creator=Bhanu};
```

4)USE database statement in Hive is used to select the specific database for a session on which all subsequent HiveQL statements would be executed.

Syn: **USE db_name;**

hive> USE employee;

OK

5)DROP database statement in Hive is used to Drop (delete) the database. The default behavior is RESTRICT which means that the database is dropped only when it is empty. To drop the database with tables, we can use CASCADE.

Syn: **DROP (DATABASE|SCHEMA) [IF EXISTS] db_name [RESTRICT|CASCADE];**

hive>DROP DATABASE facultyece;

OK

hive> DROP DATABASE IF EXISTS facultycse CASCADE;

OK

6) ALTER database statement in Hive is used to change the metadata associated with the database in Hive.

Syntax for changing Database Properties:

ALTER (DATABASE|SCHEMA) db_name SET DBPROPERTIES (property_name=property_value, ...);

hive> ALTER DATABASE employee SET DBPROPERTIES (‘creator’=‘Bhanu Prasad’, ‘date’=‘07-12-2020’);

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ cloudera USER {date= **07-12-2020**, creator=**Bhanu Prasad**};

Syn for changing Database owner:

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;

hive> ALTER DATABASE employee SET OWNER **USERclient**;

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ **client USER**{date= 07-12-2020, creator=Bhanu Prasad};

hive> ALTER DATABASE employee SET OWNER **ROLE Admin**;

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ **Admin ROLE** {date= 07-12-2020, creator=Bhanu Prasad};

6)ImplementData Definition Language (DDL) Commands for tables in Hadoop Hive framework using Cloudera.

<u>DDL Commands for Tables</u>

1) **CREATE TABLE** statement in Hive is used to create a table with the given name. If a table or view already exists with the same name, then the error is thrown. We can use IF NOT EXISTS to skip the error.

Syn: **CREATE TABLE [IF NOT EXISTS] [db_name.] table_name [(col_namedata_type [COMMENT col_comment], ... [COMMENT col_comment])]**

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]

[LOCATION hdfs_path];

hive> CREATE TABLE IF NOT EXISTS employee.emptable (emp_id STRING COMMENT ‘This is Employee ID’, emp_name STRING COMMENT ‘This is Employee Name’, emp_sal FLOAT COMMENT ‘This is Employee Salary’)

COMMENT ‘This table contains Employees Data’

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ‘,’

STORED AS TEXTFILE;

2) **SHOW** tables statement in Hive lists all the base tables and views in the current database.

   Syn: **SHOW TABLES [IN database_name];**

   hive> SHOW TABLES IN employee;

   OK

   emptable

3) **DESCRIBE** table statement in Hive shows the lists of columns for the specified table.

   Syn: **DESCRIBE [EXTENDED|FORMATTED] [db_name.] table_name[.col_name ([.field_name])];**

   hive> DESCRIBE employee.emptable;

   emp_id string This is Employee ID

   emp_name string This is Employee Name

   emp_sal float This is Employee Salary

   hive> DESCRIBE EXTENDED employee.emptable;

   hive> DESCRIBE FORMATTED employee.emptable;

4) **ALTER** table statement in Hive enables you to change the structure of an existing table, rename the table, add columns to the table, change the table properties, etc.

   Syntax for Rename a table:

   ALTER TABLE table_name RENAME TO new_table_name;

   hive> ALTER TABLE employee.emptable RENAME TO employee.facultytable;

   Syn to Add columns to a table:

   ALTER TABLE table_name ADD COLUMNS (column1, column2) ;

   hive> ALTER TABLE employee.facultytable ADD COLUMNS (emp_post string COMMENT ‘This is employee post’, emp_age INT COMMENT ‘This is employee age’);

   Syn to set table properties:

   ALTER TABLE table_name SET TBLPROPERTIES

(‘property_key’=’property_new_value’);

hive> ALTER TABLE employee.facultytable SET TBLPROPERTIES (‘table for’=’faculty data’);

5) **DROP**tablestatement in Hive deletes the data for a particular table and remove all metadata associated with it from Hive metastore.

- ➢ If PURGE is not specified, then the data is actually moved to the .Trash/current directory.
- ➢ If PURGE is specified, then data is lost completely.

Syn: **DROP TABLE [IF EXISTS] table_name [PURGE];**

hive> DROP TABLE IF EXISTS employee.emptable PURGE;

OK

6) **TRUNCATE** table statement in Hive removes all the rows from the table or partition.

Syn: **TRUNCATE TABLE table_name;**

hive> TRUNCATE TABLE employee.emptable;

OK

7)Hive Data Manipulation Language(DML) Commands
ImplementData Manipulation Language (DML) Commandsfortablesin Hadoop Hiveframework using Cloudera.

DML Commands for Tables

1) **LOAD** statement in Hive is used to copy/move data files into the locations corresponding to Hive tables.

Syn: **LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)];**

LOCAL keyword = file path in the local filesystem.

LOCAL not specified = file path in the hdfs

OVERWRITE contents of the target table (or partition) will be deleted and replaced by the files otherwise contents are added to the table

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/HiveDir/emptextdata' INTO
TABLE employee.facultytable;

OK
```

<u>emptextdata contents</u>

```
1,bob,25000.00,asstprof,35,male

2,mary,35000.00,assocprof,38,female

3,mike,50000.00,prof,45,male
```

2) **SELECT** statement in Hive is similar to the SELECT statement in SQL used for retrieving data from the database.

Syn: **SELECT * FROM tablename;** //displays all records

```
hive> SELECT * FROM employee.facultytable;
1    bob     25000.00    asstprof              35     male
2    mary    35000.00    assocprof    38       female
3    mike    50000.00    prof         45       male
```

SELECT col1,col2 FROM tablename; //Retrieves only specified columns data

```
hive> SELECT emp_name,emp_salary FROM employee.facultytable;
bob         25000.00
mary        35000.00
mike        50000.00
```

3) **a) INSERT INTO** statement appends the data into existing data in the table or partition.

Syn: **INSERT INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)] VALUES (col1value,col2value,…)**

```
hive> INSERT INTO TABLE employee.facultytable VALUES (4, 'jessy', 45000.00,
```

‘assocprof’, 40, ‘female’);

hive> SELECT * FROM employee.facultytable;

```
4   jessy   45000.00    assocprof   40     female
1   bob     25000.00    asstprof           35     male
2   mary    35000.00    assocprof   38     female
3   mike    50000.00    prof        45     male
```

b) INSERT OVERWRITE table overwrites the existing data in the table or partition.

Syn: **INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [IF NOT EXISTS]] select_statement FROM from_statement;**

4) **DELETE** statement in Hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause.

Syn: **DELETE FROM tablename [WHERE expression];**

hive> DELETE FROM employee.facultytable WHERE emp_age=38;

hive> SELECT * FROM employee.facultytable;

```
4   jessy   45000.00    assocprof   40     female
1   bob     25000.00    asstprof           35     male
3   mike    50000.00    prof        45     male
```

5)**UPDATE** statement in Hive deletes the table data. If the WHERE clause is specified, then it updates the column of the rows that satisfy the condition in WHERE clause. Partitioning and Bucketing columns cannot be updated.

Syn: **UPDATE tablename SET column = value [, column = value ...] [WHERE expression];**

hive> UPDATE employee.facultytable SET emp_name = ‘mike tyson’ WHERE emp_age=45;

hive> SELECT * FROM employee.facultytable;

```
4   jessy   45000.00    assocprof   40     female
```

```
1   bob      25000.00      asstprof           35      male
3   mike tyson      50000.00      prof        45      male
```

6)EXPORT statement exports the table or partition data along with the metadata to the specified output location in the HDFS. Metadata is exported in a _metadata file, and data is exported in a subdirectory ‘data.’

Syn: **EXPORT TABLE tablename [PARTITION (part_column="value"[, ...])] TO 'export_target_path' [FOR replication('eventid')];**

hive> EXPORT TABLE employee.drivertable TO ‘/user/hive/warehouse’;

7)IMPORT command imports the data from a specified location to a new table or already existing table.

Syn: **IMPORT [[EXTERNAL] TABLE new_or_original_tablename [PARTITION (part_column="value"[, ...])]] FROM 'source_path' [LOCATION 'import_target_path’];**

hive> IMPORT TABLE employee.importedtable FROM ‘/user/hive/warehouse’;