Assignment No. 1
Linux Command
Create File in Linux

```
vi firstfile
```

Show File Contain in Linux

```
cat firstfile.txt
```

Hadoop Command
1. Create a Directory in HDFS
To create a directory in HDFS:

```
hdfs dfs –mkdir firstdir
```

2. List Files in a Directory in HDFS

```
hdfs dfs –ls
```

3. Upload Files from Local File System to HDFS
Use the -put command to upload files from your local file system to HDFS.

```
hdfs dfs –put firstfile.txt /user/cloudera/firstdir
```

4. List Files in a Directory
Use the -ls command to list the files in an HDFS directory.

```
hdfs dfs –ls /user/cloudera/firstdir/
```

5. Display the Contents of a File
Use the -cat command to display the contents of a file in HDFS.

```
hdfs dfs -cat /user/cloudera/firstfile.txt
```

6. Delete a File in HDFS
Use the -rm command to remove a file from HDFS.

```
hdfs dfs -rm /user/cloudera/firstdir/firstfile1.txt
```

Assignment No. 2
Implement a Java program to interact with HDFS (reading and writing files).

```
import java.io.File;
public class filehand {
public static void main(String []args) throws IOException
{
File obj1=new File("/home/cloudera/Desktop/Firstfile.txt");
if (obj1.createNewFile())
System.out.print("File is Created");
else
System.out.print("File is Already exist");
FileWrite w1=new FileWriter("/home/cloudera/Desktop/Firstfile.txt");
w1.write("Welcome my first file is write");
w1.close();
Scanner r1=new Scanner(obj1);
while(r1.hasNextLine())
{
String data=r1.nextLine();
System.out.println(data);
}  }  }
```

Assignment No. 3

Use Hadoop's built-in commands to manage files and directories.

1. Create Directories in HDFS.

```
hdfs dfs –mkdir firstdir
```

2. Upload Files from Local File System to HDFS

Use the -put command to upload files from your local file system to HDFS.

```
hdfs dfs –put firstfile.txt /user/cloudera/firstdir
```

3. List Files in a Directory

Use the -ls command to list the files in an HDFS directory.

```
hdfs dfs –ls /user/cloudera/firstdir/
```

5. Display the Contents of a File

Use the -cat command to display the contents of a file in HDFS.

```
hdfs dfs -cat /user/cloudera/firstfile.txt
```

6. Copy Files from HDFS to Local File System

Use the -get command to copy files from HDFS to your local file system.

```
hdfs dfs -get /user/cloudera/firstdir/firstfile.txt/home/cloudera/lindir
```

7. Delete a File in HDFS

Use the -rm command to remove a file from HDFS.

```
hdfs dfs -rm /user/cloudera/firstdir/firstfile1.txt
```

7. Delete a Directory in HDFS

Use the -rm -r command to delete a directory and its contents from HDFS.

```
hdfs dfs -rm -r /user/cloudera/firstdir (For Non Empty Directory
hdfs dfs -rm /user/cloudera/firstdir (For Empty Directory )
```

Assignment No. 4

Implement Map Side Join and Reduce Side Join.

(Write hadoop code to implement Map Reduce application count number of word in file)

```
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);
}
}
}
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 (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
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);
}
}
```

File Link

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-
core/MapReduceTutorial.html#Source_Code

Step 1: Export Java Eclipse Project Jar File to Cloudera
Step 2. Make firstfile.txt file vi editor ->Write data
Step 3: Perform Below commands on terminal
Command Map Reduce Code
1) Transfer all local file to hadoop

```
Hdfs dfs –put firstfile.txt /user/cloudera
Hdfs dfs –put WordCount.jar /user/cloudera
```

2) Run Java Jar File for Map Reduce Operation

```
hadoop jar WordCount.jar WordCount firstfile.txt outputfile
```

3) List outputfile

```
hdfs dfs –ls /user/cloudera/outputfile
```

4) Show outputfile

```
hdfs dfs –cat /user/cloudera/outputfile/part-r-00000
```

Assignment No. 6
Pipeline multiple Map Reduce jobs

Example: Pipelining Two Jobs

Job 1:Word Count(Word frequency count)

This first job counts the occurrences of each word in the input text files.

```
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 {

String[] words = value.toString().split("\\s+"); for
(String wordStr : words) {
word.set(wordStr);
context.write(word, one);
}
}
}
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 (IntWritable val : values) {
sum += val.get();
}
result.set(sum); context.write(key,
result);
}
}
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);
}
}
```

Job 2:Filter Words with Frequency GreaterThan 2

The secondjobprocesses the output ofthe firstjob to filter and only output words that have a frequency greater than 2

```
public class FilterWords {
public static class FilterMapper extends Mapper<Object, Text, Text,
IntWritable> {
private IntWritable count = new IntWritable();
public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {

String[] fields = value.toString().split("\t");
String word = fields[0];
int wordCount = Integer.parseInt(fields[1]);
// Output only words with count greater than 2 if
(wordCount > 2) {
count.set(wordCount); context.write(new
Text(word), count);
}
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "filter words");
job.setJarByClass(FilterWords.class);
job.setMapperClass(FilterMapper.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0])); // Input path
from the first job's output
FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output
path
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

 Output format: The output of each job should be compatible with the input format of the next
job. For instance, ifthe output of Job 1 is a simple key-value pair(word and count),Job 2 should be able to process that format directly.
Step 1: Export Java Eclipse Project Jar File to Cloudera Step 2. Make firstfile.txt file vi editor ->Write data Step 3: Perform Below commands on terminal
Command MapReduce Code

1) Transfer all local file to hadoop
```
Hdfs dfs –put firstfile.txt /user/cloudera Hdfs
dfs –put PipLine1.jar /user/cloudera
```
2) Run First job of Java Jar File for Map Reduce Operation
```
hadoop jar PipLine1.jar wordcount firstfile.txt outpip1
```
3) Run Second job of Java Jar File for Map Reduce Operation
```
hadoop jar PipLine1.jar FilterWords outpip1 outpip2
```
4) List outputfile
```
hdfs dfs –ls /user/cloudera/outpip2
```
5) Show outputfile
```
hdfs dfs –cat /user/cloudera/outpip2/part-r-00000
```

Assignment No. 7
Create and use UDFs in Pig Latin scripts (Write hadoop code to convert
username in uppercase)

Step 1: Open Java Eclipse -> Make New Project->(PigAss7) Add External Libarary->pig , hadoop
,hadoop 0.20_map_reduce->finish
Step 2: Add class in project -> PigUDF -> Write below code (Create the UDF)
```
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
```

```
public class PigUDF extends EvalFunc<String> {
public String exec(Tuple tuple)throws IOException {
if(tuple ==null) {
return null;
}
String user=(String)tuple.get(0);
String city=(String)tuple.get(1);
int score=(Integer)tuple.get(2);
return user+”,”+city.toUpperCase()+”,”+score;
}
}
```

Step 3: Export java project in to jar file ->PigAss7.jar
Step 4: Open terminal create cust_us.txt file using following command

```
vi cust_us
ex. User1,dhule,3
```

Step 5: Open terminal to create pig file using following command.

```
vi toupper.pig
```

Write following code in file

```
REGISTER /home/cloudera/PigAss7.jar
DEFINE toupper PigUDF();

usa = LOAD ‘/home/cloudera/cust_us.txt’ Using PigStorage(‘,’)
AS (user:chararray,city:chararray,score:int);
usa_upper = FOREACH usa GENERATE toupper(user,city,score);
DUMP usa_upper;
```

Step 6: Run Pig script

```
pig –x local toupper.pig
```

Assignment No. 7
Create and use UDFs in Pig Latin scripts (Write hadoop code to convert username in uppercase)

Step 1: Open Java Eclipse -> Make New Project->(PigAss7) Add External Libarary->pig , hadoop
,hadoop 0.20_map_reduce->finish
Step 2: Add class in project -> PigUDF -> Write below code (Create the UDF)

```
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
public class PigUDF extends EvalFunc<String> {
public String exec(Tuple tuple)throws IOException {
```

```
if(tuple ==null) {
return null;
}
String user=(String)tuple.get(0);
String city=(String)tuple.get(1);
int score=(Integer)tuple.get(2);
return user+”,”+city.toUpperCase()+”,”+score;
}
}
```

Step 3: Export java project in to jar file ->PigAss7.jar
Step 4: Open terminal create cust_us.txt file using following command

```
vi cust_us
```

Step 5: Open terminal to create pig file using following command.

```
vi toupper.pig
```

Write following code in file

```
REGISTER /home/cloudera/PigAss7.jar
DEFINE toupper PigUDF();

usa = LOAD ‘/home/cloudera/cust_us.txt’ Using PigStorage(‘,’)
AS (user:chararray,city:chararray,score:int);
usa_upper = FOREACH usa GENERATE toupper(user,city,score);
DUMP usa_upper;
```

Step 6: Run Pig script

```
pig –x local toupper.pig
```

Assignment No 9
Implement and execute HiveQL queries to perform data retrieval and manipulation.
1. Setting up Hive

```
Hive
```

2. Creating a Hive Table

```
CREATE TABLE employees (
emp_id INT,
emp_name STRING,
emp_salary DOUBLE,
emp_department STRING
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

3. Data Manipulation

a. Inserting Data into the Table:
INSERT INTO TABLE employees VALUES (1001, 'John Doe', 60000.00,'Engineering');
b. Updating Data:
INSERT OVERWRITE TABLE employees
SELECT emp_id, emp_name, CASE
WHEN emp_salary < 50000 THEN emp_salary * 1.1
ELSE emp_salary
END AS emp_salary,
emp_department
FROM employees;
c. Deleting Data:
INSERT OVERWRITE TABLE employees
SELECT * FROM employees WHERE emp_id != 1001;
4. Basic Data Retrieval Queries
a. Select all records:
SELECT * FROM employees;
b. Select specific columns:
SELECT emp_name, emp_salary FROM employees;
c. Select with a WHERE clause:
SELECT * FROM employees WHERE emp_salary > 50000;

Assignment No 10
Perform operations like joins, group by, and aggregations in Hive.
1. Creating a Hive Table
CREATE TABLE employees (
emp_id INT,
emp_name STRING,
emp_salary DOUBLE,
emp_department STRING
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
CREATE TABLE departments (
dept_id INT,
dept_name STRING,
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
Insert 5 record on each table by using insert command
2. Joins
SELECT e.emp_name, d.dept_name

```
FROM employees e
JOIN departments d
ON e.emp_department = d.dept_id;
```

3. Aggregate Functions

a. Count the number of employees:

```
SELECT COUNT(*) FROM employees;
```

b. Find the average salary:

```
SELECT AVG(emp_salary) FROM employees;
```

c. Find the highest salary:

```
SELECT MAX(emp_salary) FROM employees;
```

4. Group By

Group by department and get average salary per department:

```
SELECT emp_department, AVG(emp_salary) FROM employees GROUP BY emp_department;
```