# INFO 208 Quiz #1 — Hadoop Lab Exploration Component

Your Name: Venkata Anil Kumar Thota

The following Lab items should be carried out by you using your **Hortonworks HDP Sandbox** VM. Remember, you were asked to keep your HDP Sandbox VM for later use — both in this course and and for continuing education beyond this course.

The purpose of this **Lab Exploration Component to Quiz #1** is to determine how well you understood the Hadoop Labs that you did in the first weeks of this course and how well you can apply the concepts and approaches you learned there to simple situations that are different but somewhat similar.

**Open up the area after each question / section** in this MS Word file and type in (or paste) your answer(s) and your comments. Mail the resulting file to me at [glen.mules@sjsu.edu](mailto:glen.mules@sjsu.edu)

Your email **Subject** line should be:

**INFO 208 Quiz**

And the attached MS Word file sent by email should be named:

**INFO\_208\_HadoopQuiz-*YourLastName*.docx**

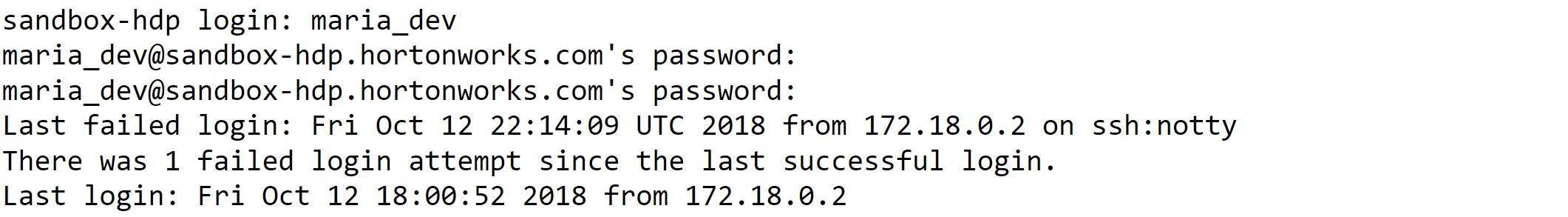
The file naming is important as it causes my email software to sort your *Quiz Response Email* into a directory for me so that I don’t lose amongst my other daily email.

**Basic Lab** (100% of your grade for the Lab Exploration Component):

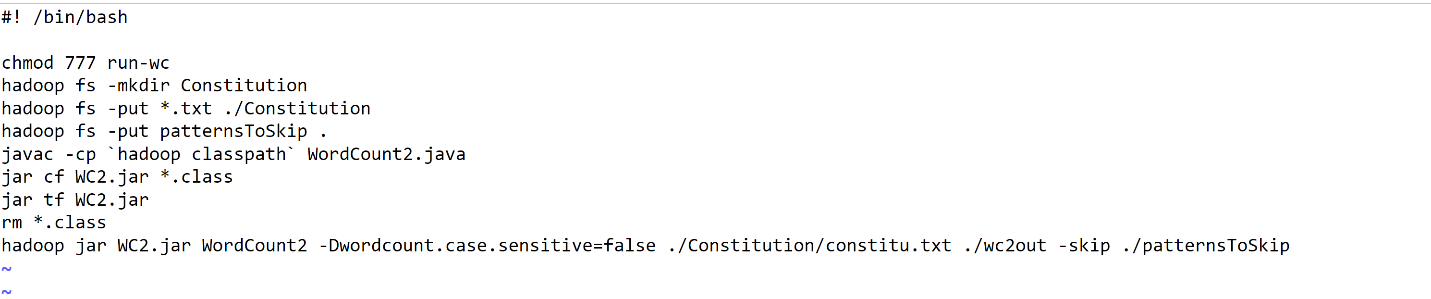
1. Locate a plain-text copy of the US Constitution. One place you can find it is: <https://www.constitution.org/cons/constitu.txt>
2. Using the **wordcount2** MapReduce program that you have already compiled (in Hadooop Labs #8), run **wordcount2** on the US Constitution text file. Find the top 20 non-trivial words (i.e., eliminate such words as: a, the, and, when, …
3. Remember your output directory must not exist or must be empty when you run wordcount2.
4. Explain *what you did*, *where you did it*, *how you ran it*, etc. with all the statements that you needed to use. The ***why*** you did something is even more important than ***what*** you did.

**Output: -**

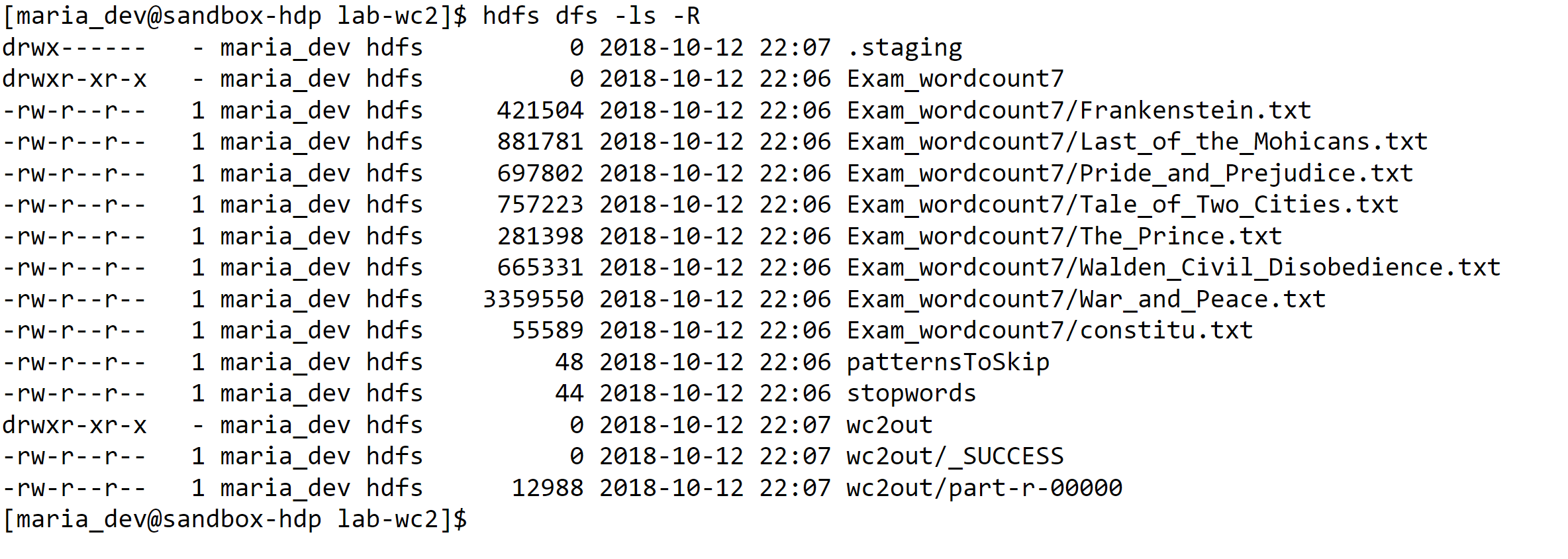
* Opened web shell and logged in using the maria\_dev credentials.

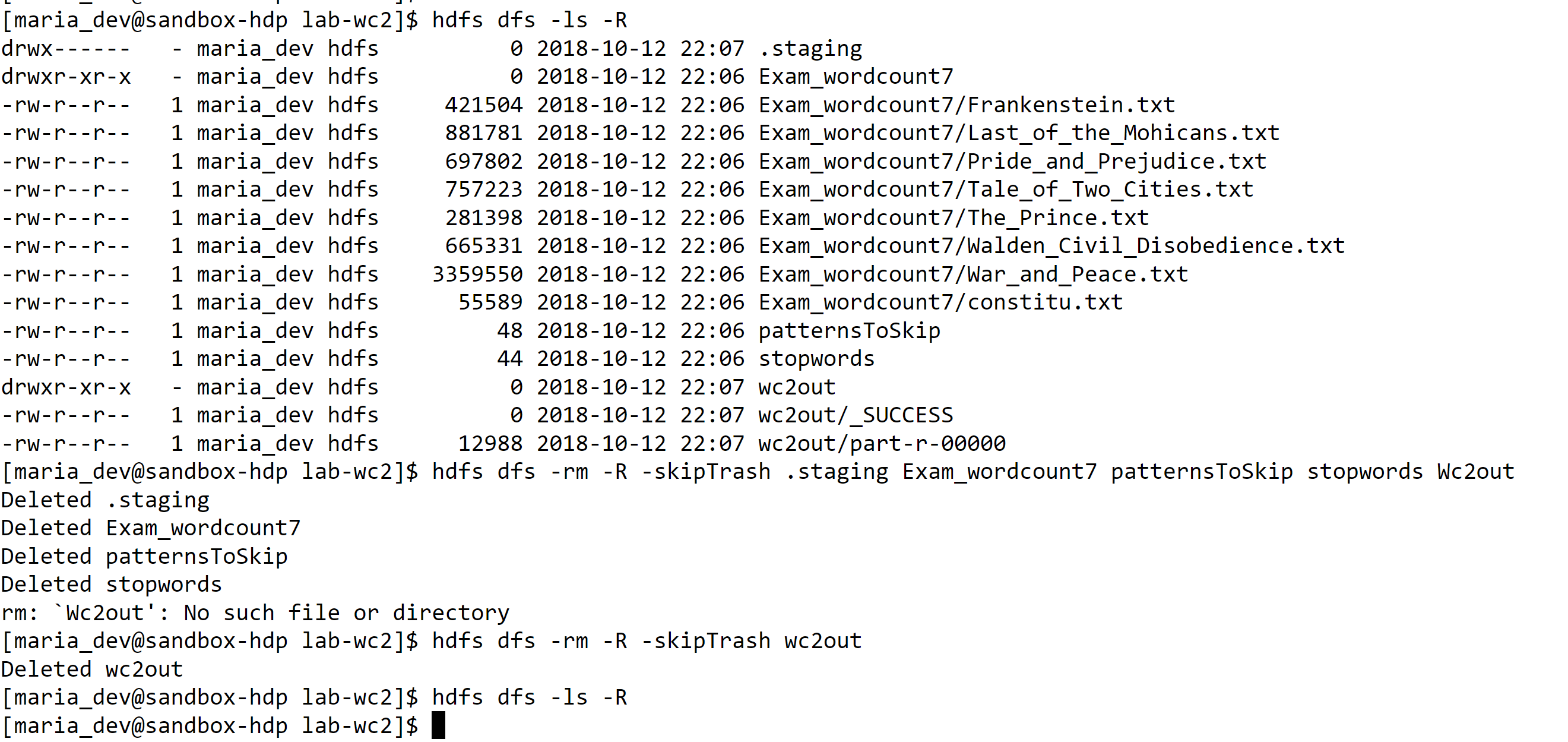


* As I already extracted tar file, made changes in the run-wc. Opened run-wc using “vi” and made changes in the insert mode, I saved the changes by clicking “:wq!” also before this step I downloaded constitu.txt file from the given link using the wget command.

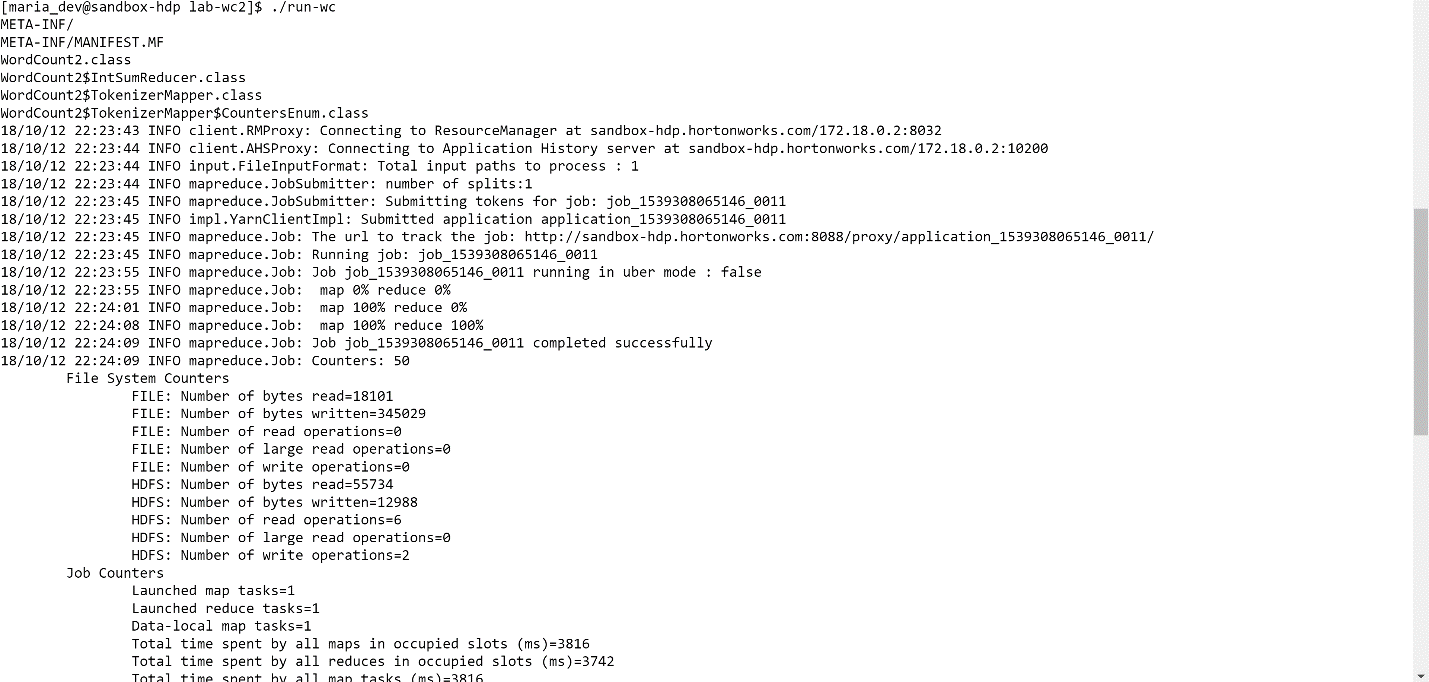


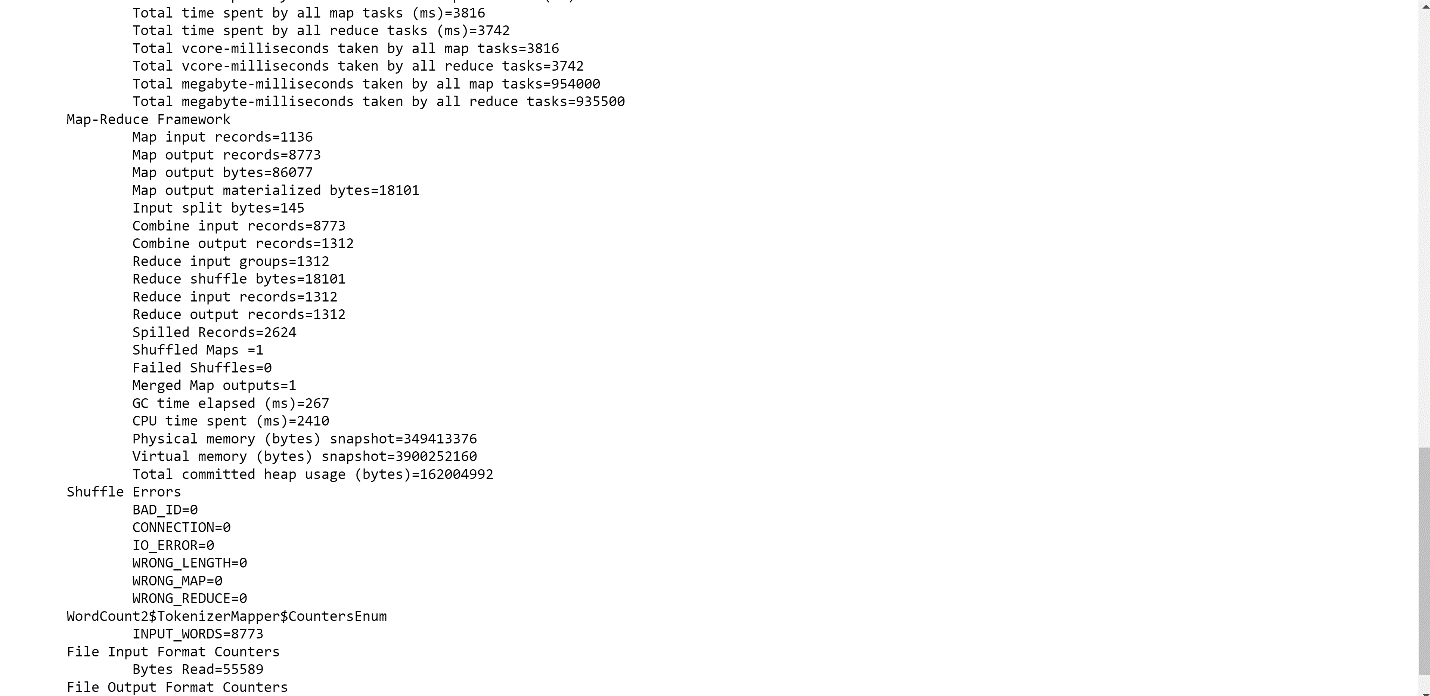
* run-wc was made executable. It has read,write and execute permissions for owner,group,and others. Then I checked whether maria\_dev has any files in the HDFS. There are few files of previous program which I did lastweek. Deleted all the files using the “rm” command. Below are the screenshots.

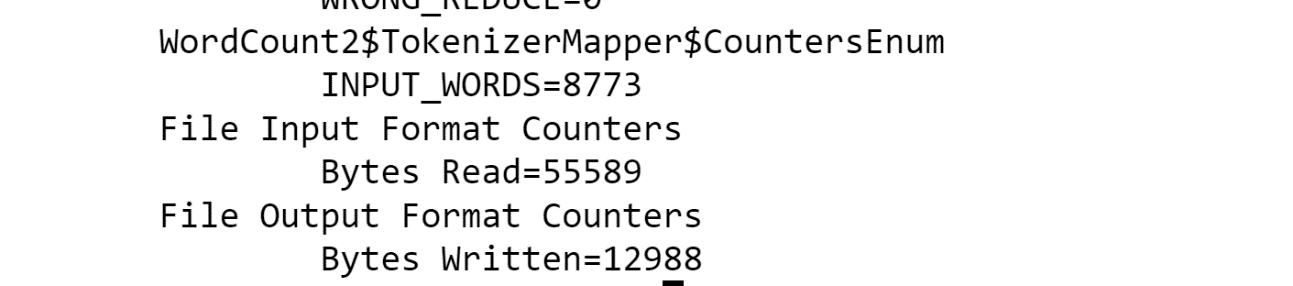




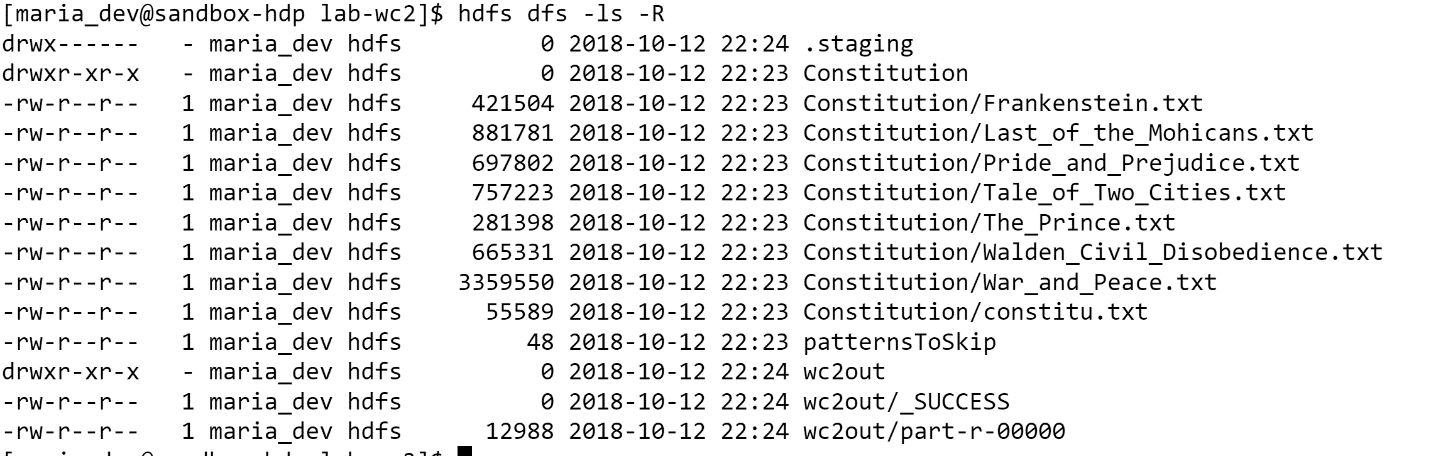
* Finally I started the program by using “./” which gives the current directory of the executable file. JVM has started and it took almost 30 seconds to complete the processing.

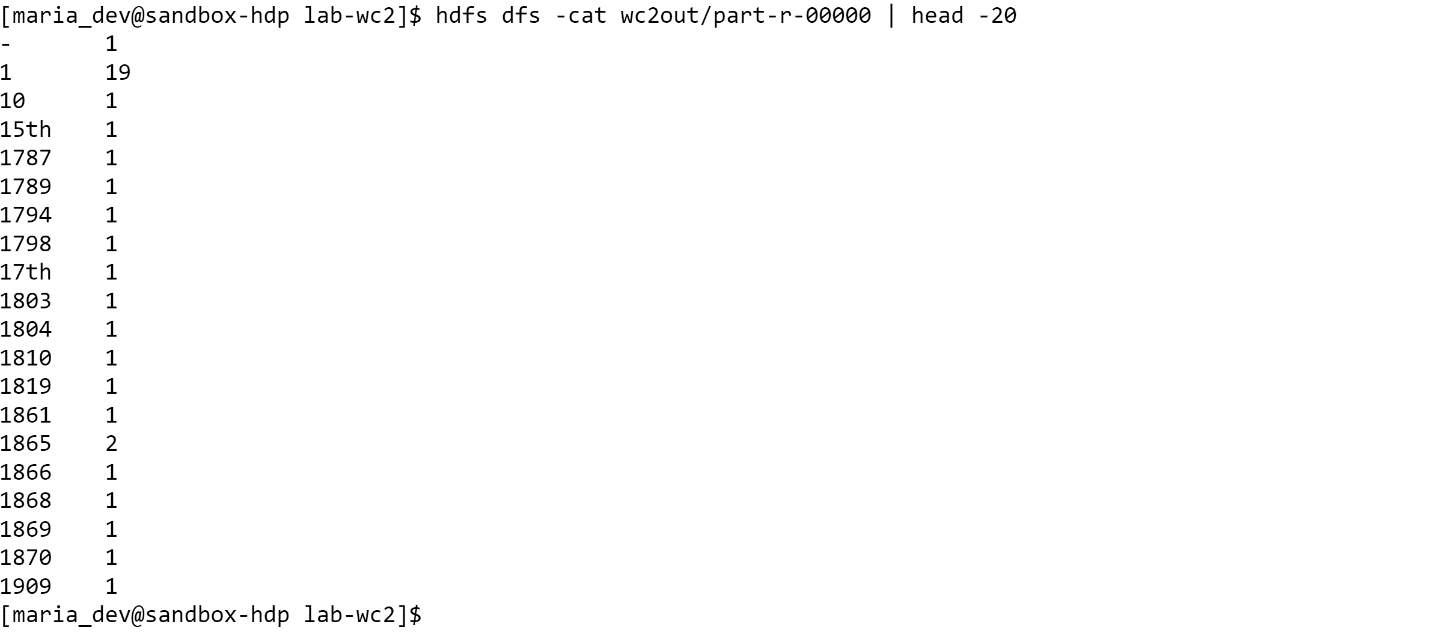


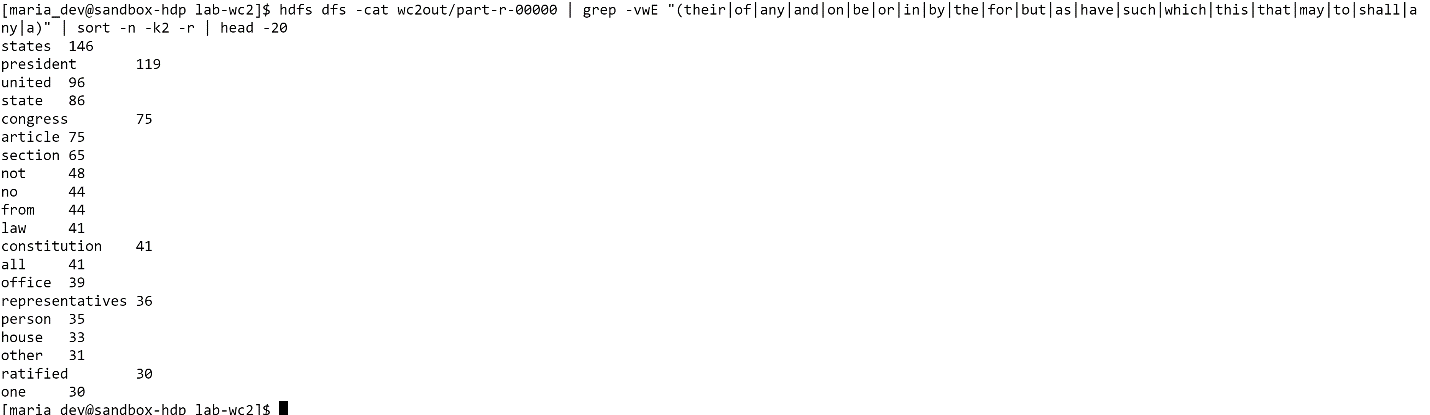




* I checked the files which are available now in HDFS. Explored the wc2out/part-0000 file by using head, tail and more commands. Using the grep and sort I got the top 20 Non-Trivial words.







* Java project is created, and the code is exported in the form of jar file. The Jar file is moved to our Hadoop system and it is extracted. Wordcount2.java file contains the java code. Patternstoskip file has all the punctuations which are need to be eliminated. Even I tried to keep all the stop words in the patterntoskip file and eliminate them in the output, but the problem which I faced here is those stop words are even eliminated in the other Non-trivial words. We should make sure that the Files in the HDFS are removed before running the program because it will get confused where to store the output file and it throws an Error.

**Lab for Extra Credit** (beyond the 100% of grade for the Lab Exploration Component — for up to 20% extra). ***This is optional extra***. No knowledge of programming in Java is actually needed to complete this.

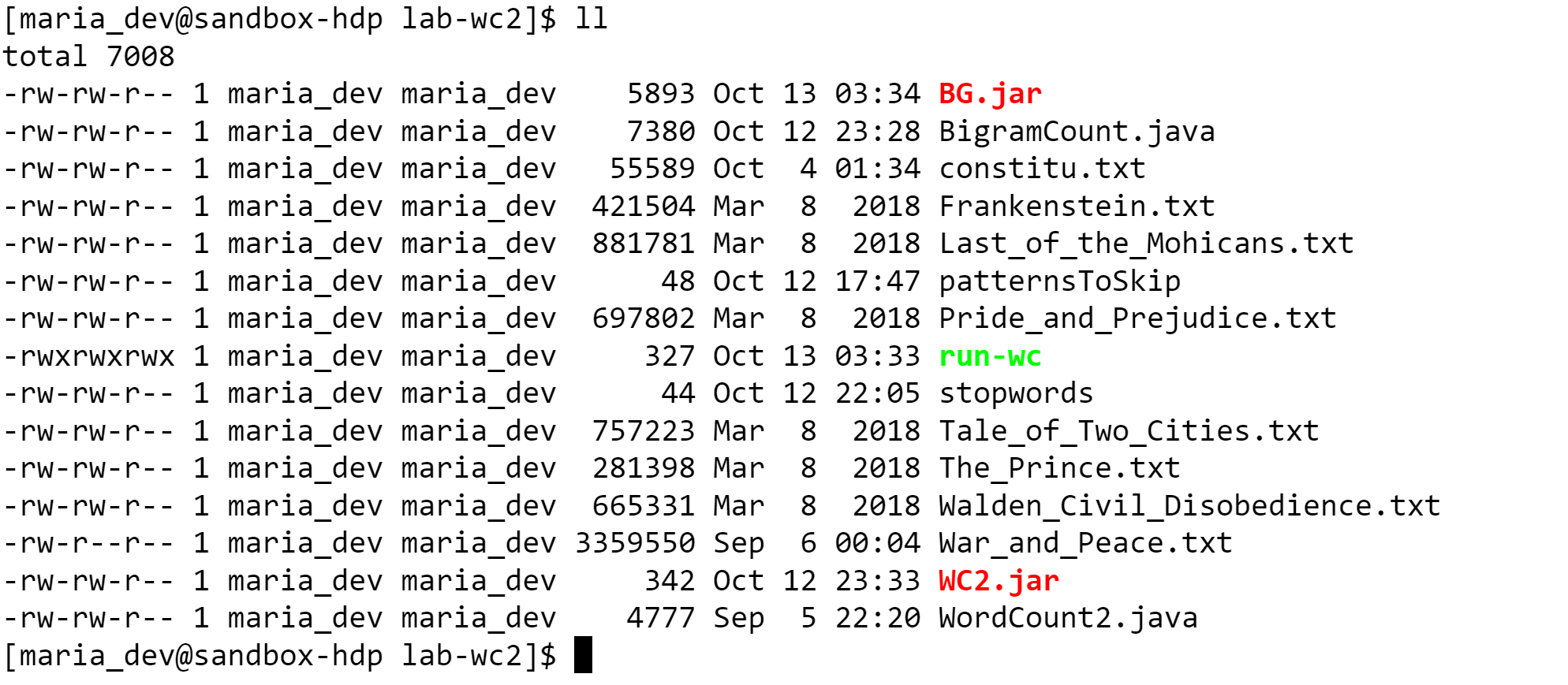
1. Find a program on the Internet that does pairings of words (called a 2-gram or bi-gram) in the text of the text files analyzed. They exist — probably programmed in Java for Hadoop/MapReduce.

Bigrams are simply *sequences of two consecutive words*. For example, the previous sentence contains the following bigrams: "Bigrams are", "are simply", "simply sequences", "sequence of", etc.

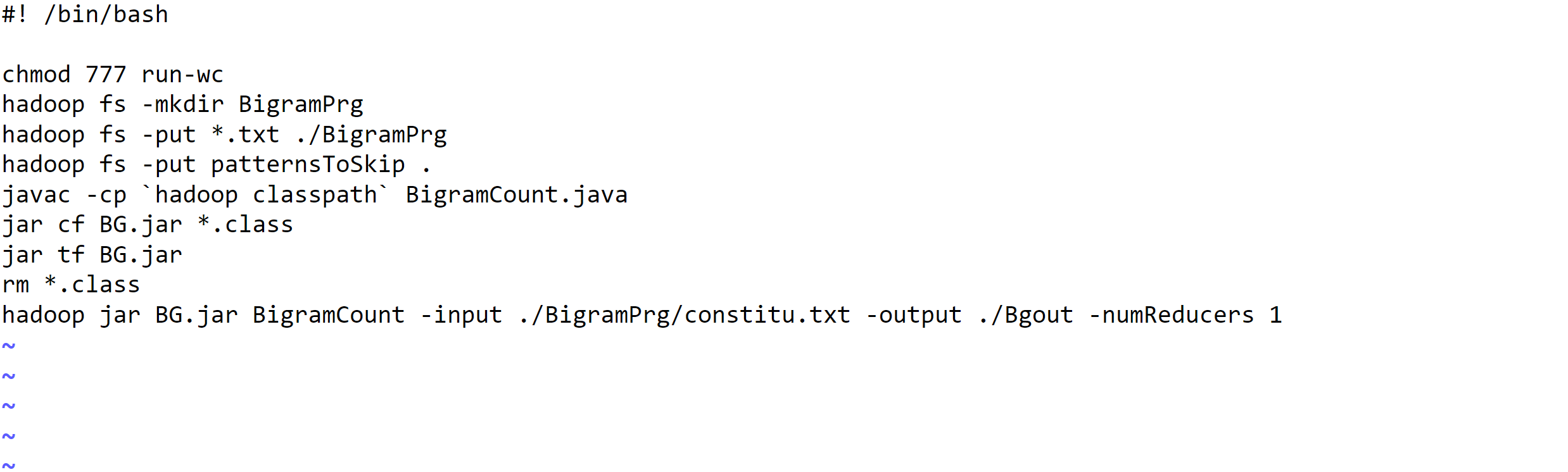
1. Using the approach that was used with wordcount2, *compile* the Java program and have it run against the Gutenberg documents that were loaded in Lab #8 *or* the US Constitution document that you downloaded above.
2. Find the most common bi-grams in your target text. The first 20 or so that do not use non-trivial / stop-words will be sufficient

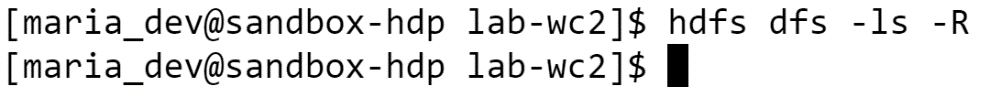
**Output:-**

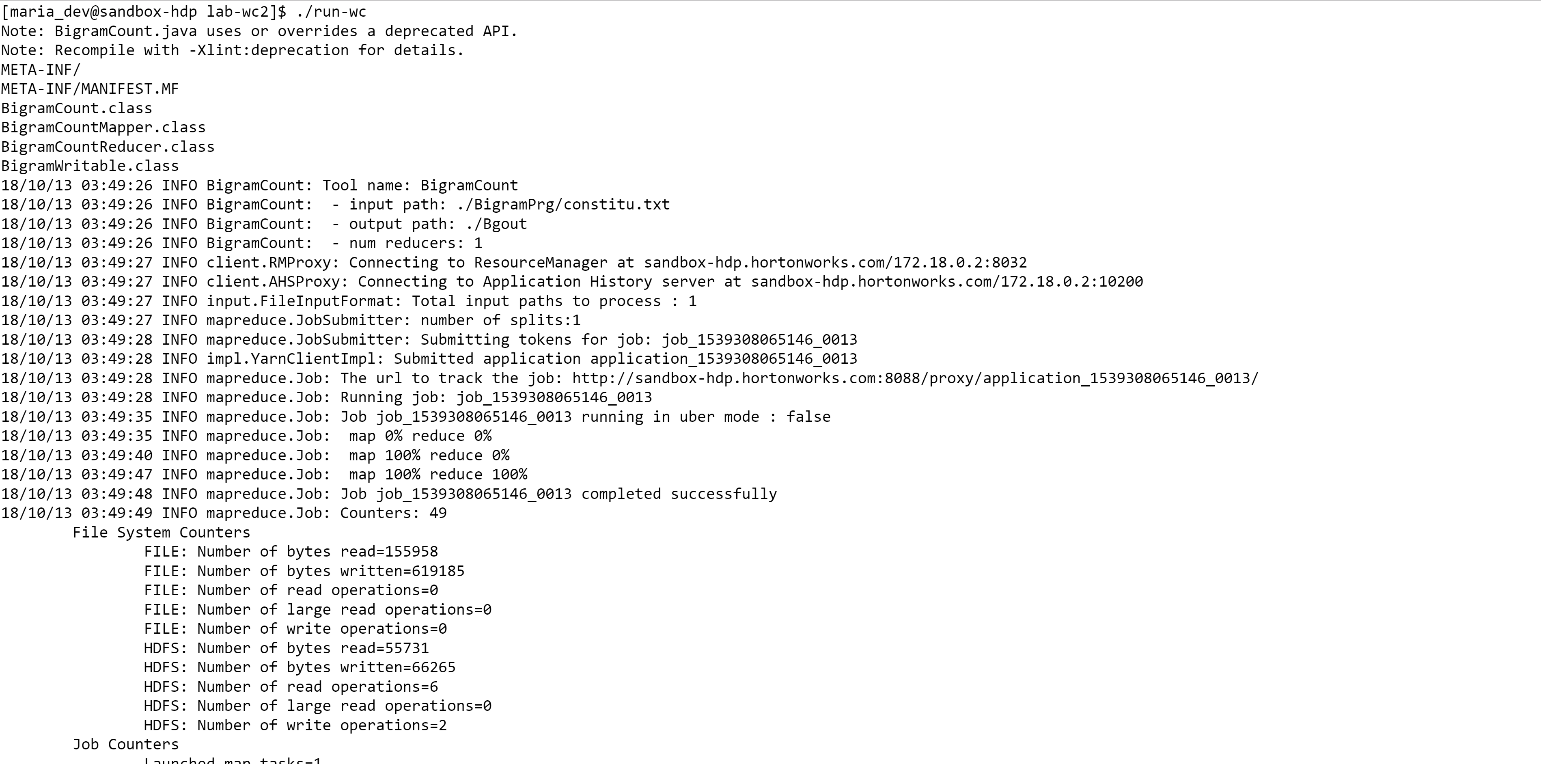
* Downloaded the Java File from the internet. Then I copied the file from my local machine to the Hadoop system using the scp command.
* After that I did changes to the run-wc file and made it executable.

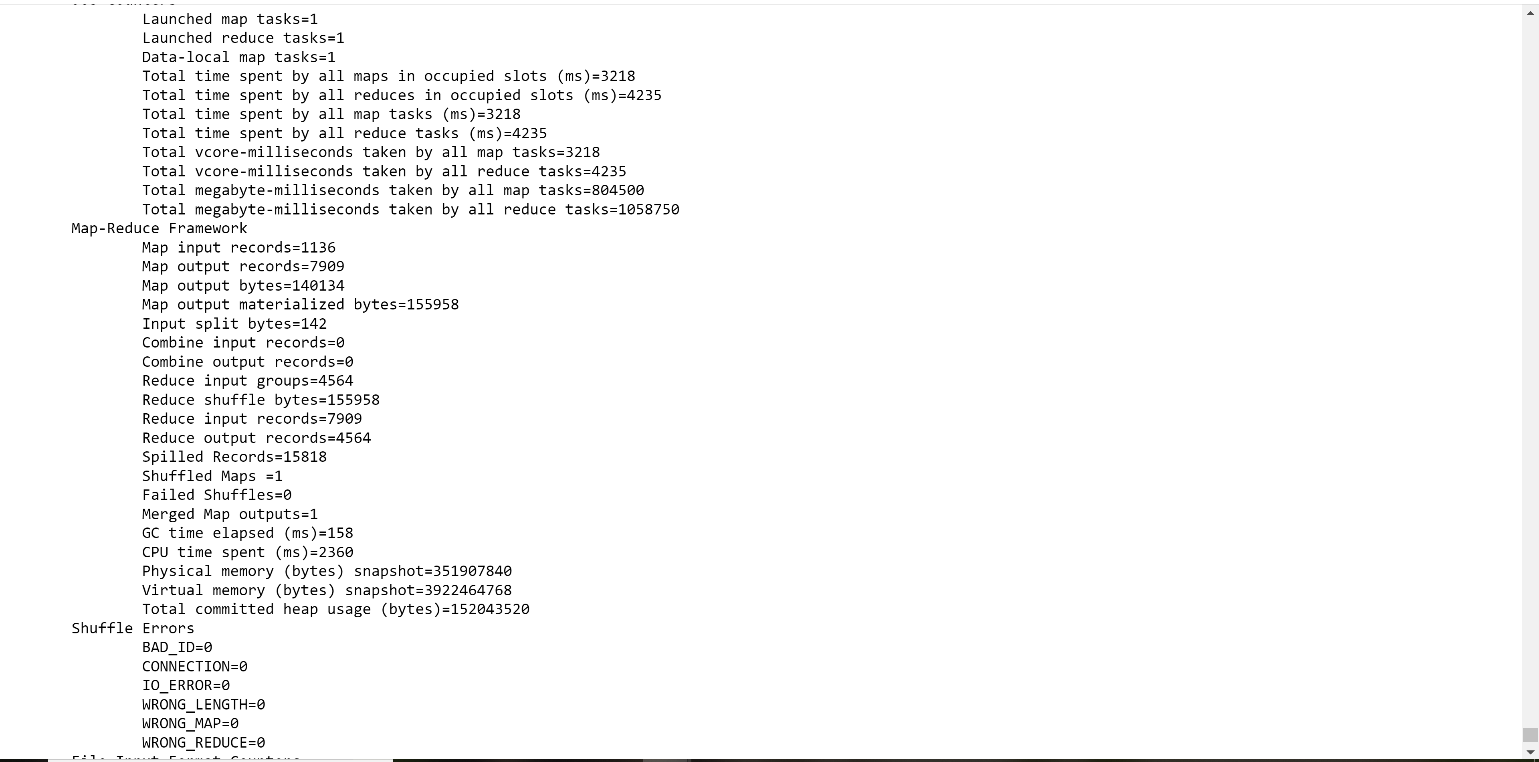


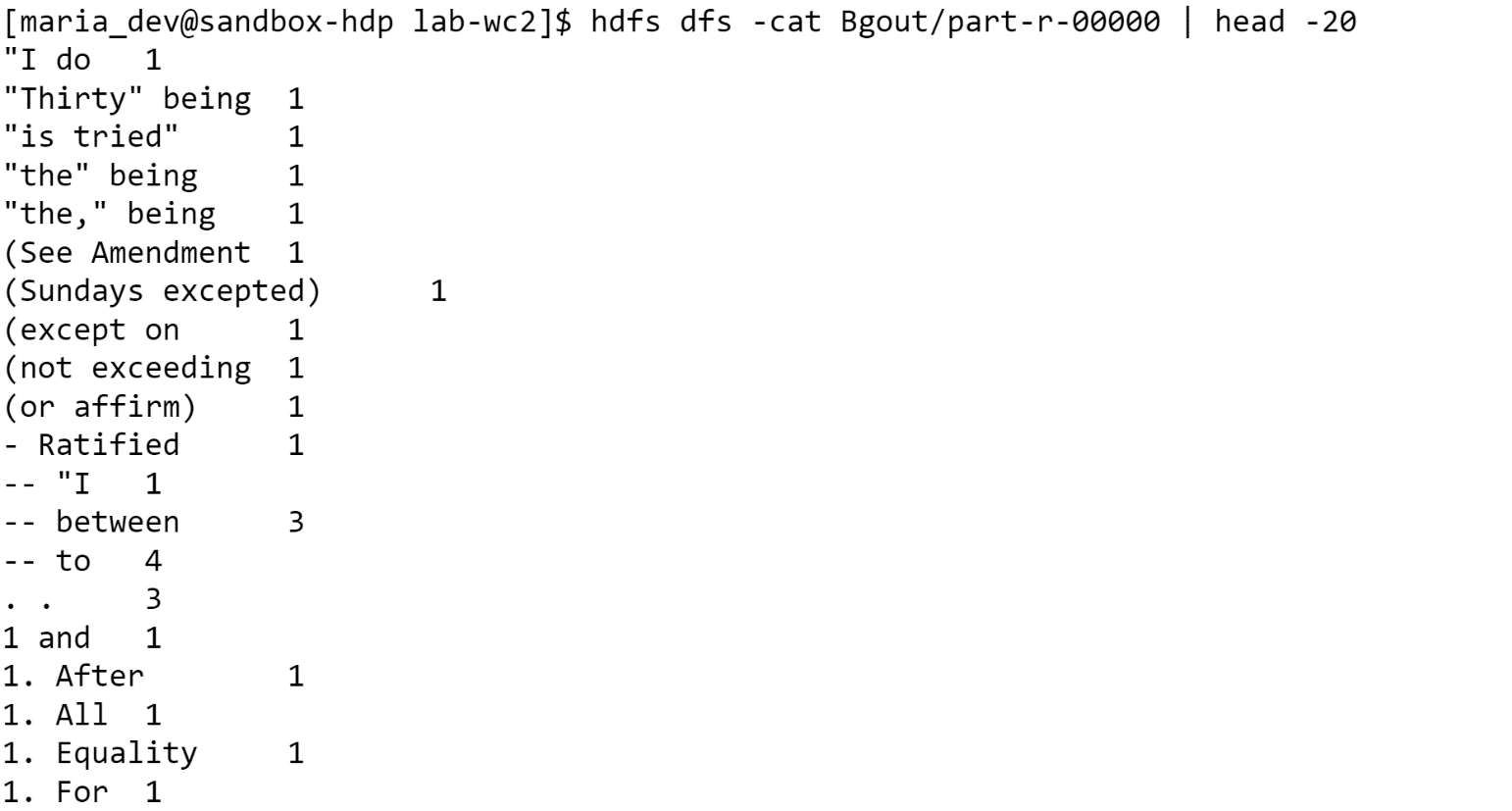
* Done Changes to run-wc File. Removed the Files in the HDFS and ran the program against the constitution File. Got the top 20 words(Without using the Non-trivial Words).











**Java Code:-**

import java.io.\*;

import java.util.\*;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import java.util.Arrays;

import org.apache.commons.cli.CommandLine;

import org.apache.commons.cli.CommandLineParser;

import org.apache.commons.cli.GnuParser;

import org.apache.commons.cli.HelpFormatter;

import org.apache.commons.cli.OptionBuilder;

import org.apache.commons.cli.Options;

import org.apache.commons.cli.ParseException;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

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.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import org.apache.log4j.Logger;

public class BigramCount extends Configured implements Tool {

private static final Logger LOG = Logger.getLogger(BigramCount.class);

private BigramCount() {

}

private static final String INPUT = "input";

private static final String OUTPUT = "output";

private static final String NUM\_REDUCERS = "numReducers";

@SuppressWarnings({ "static-access" })

public int run(String[] args) throws Exception {

Options options = new Options();

options.addOption(OptionBuilder.withArgName("path").hasArg()

.withDescription("input path").create(INPUT));

options.addOption(OptionBuilder.withArgName("path").hasArg()

.withDescription("output path").create(OUTPUT));

options.addOption(OptionBuilder.withArgName("num").hasArg()

.withDescription("number of reducers").create(NUM\_REDUCERS));

CommandLine cmdline;

CommandLineParser parser = new GnuParser();

try {

cmdline = parser.parse(options, args);

} catch (ParseException exp) {

System.err.println("Error parsing command line: "

+ exp.getMessage());

return -1;

}

if (!cmdline.hasOption(INPUT) || !cmdline.hasOption(OUTPUT)) {

System.out.println("args: " + Arrays.toString(args));

HelpFormatter formatter = new HelpFormatter();

formatter.setWidth(120);

formatter.printHelp(this.getClass().getName(), options);

ToolRunner.printGenericCommandUsage(System.out);

return -1;

}

String inputPath = cmdline.getOptionValue(INPUT);

String outputPath = cmdline.getOptionValue(OUTPUT);

int reduceTasks = cmdline.hasOption(NUM\_REDUCERS) ? Integer

.parseInt(cmdline.getOptionValue(NUM\_REDUCERS)) : 1;

LOG.info("Tool name: " + BigramCount.class.getSimpleName());

LOG.info(" - input path: " + inputPath);

LOG.info(" - output path: " + outputPath);

LOG.info(" - num reducers: " + reduceTasks);

Job job = new Job(getConf());

job.setJobName(BigramCount.class.getSimpleName());

job.setJarByClass(BigramCount.class);

job.setNumReduceTasks(reduceTasks);

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setMapOutputKeyClass(BigramWritable.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setOutputFormatClass(TextOutputFormat.class);

job.setMapperClass(BigramCountMapper.class);

job.setReducerClass(BigramCountReducer.class);

Path outputDir = new Path(outputPath);

FileSystem.get(getConf()).delete(outputDir, true);

long startTime = System.currentTimeMillis();

job.waitForCompletion(true);

System.out.println("Job Finished in "

+ (System.currentTimeMillis() - startTime) / 1000.0

+ " seconds");

return 0;

}

public static void main(String[] args) throws Exception {

ToolRunner.run(new BigramCount(), args);

}

}

class BigramCountMapper extends Mapper<LongWritable, Text, BigramWritable, IntWritable> {

private static final IntWritable ONE = new IntWritable(1);

private static final BigramWritable BIGRAM = new BigramWritable();

@Override

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

String prev = null;

StringTokenizer itr = new StringTokenizer(line);

while (itr.hasMoreTokens()) {

String cur = itr.nextToken();

if (prev != null) {

BIGRAM.set(new Text(prev),new Text(cur));

context.write(BIGRAM, ONE);

}

prev = cur;

}

}

}

class BigramCountReducer extends Reducer<BigramWritable, IntWritable, Text, IntWritable> {

private final static IntWritable SUM = new IntWritable();

@Override

public void reduce(BigramWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

Iterator<IntWritable> iter = values.iterator();

while (iter.hasNext()) {

sum += iter.next().get();

}

SUM.set(sum);

context.write(new Text(key.toString()), SUM);

}

}

class BigramWritable implements WritableComparable<BigramWritable> {

private Text leftBigram;

private Text rightBigram;

public BigramWritable() {

}

public BigramWritable(Text left, Text right) {

this.leftBigram = left;

this.rightBigram = right;

}

public void readFields(DataInput in) throws IOException {

leftBigram = new Text(in.readUTF());

rightBigram = new Text(in.readUTF());

}

public void write(DataOutput out) throws IOException {

out.writeUTF(leftBigram.toString());

out.writeUTF(rightBigram.toString());

}

public void set(Text prev, Text cur) {

leftBigram = prev;

rightBigram = cur;

}

@Override

public String toString() {

return leftBigram.toString() + " " + rightBigram.toString();

}

@Override

public int hashCode() {

return leftBigram.hashCode() + rightBigram.hashCode();

}

@Override

public boolean equals(Object o) {

if (o instanceof BigramWritable) {

BigramWritable bigram = (BigramWritable) o;

return leftBigram.equals(bigram.leftBigram)

&& rightBigram.equals(bigram.rightBigram);

}

return false;

}

public int compareTo(BigramWritable tp) {

int cmp = leftBigram.compareTo(tp.leftBigram);

if (cmp != 0) {

return cmp;

}

return rightBigram.compareTo(tp.rightBigram);

}

}