Ajinkya Shinde

2019IMT-008

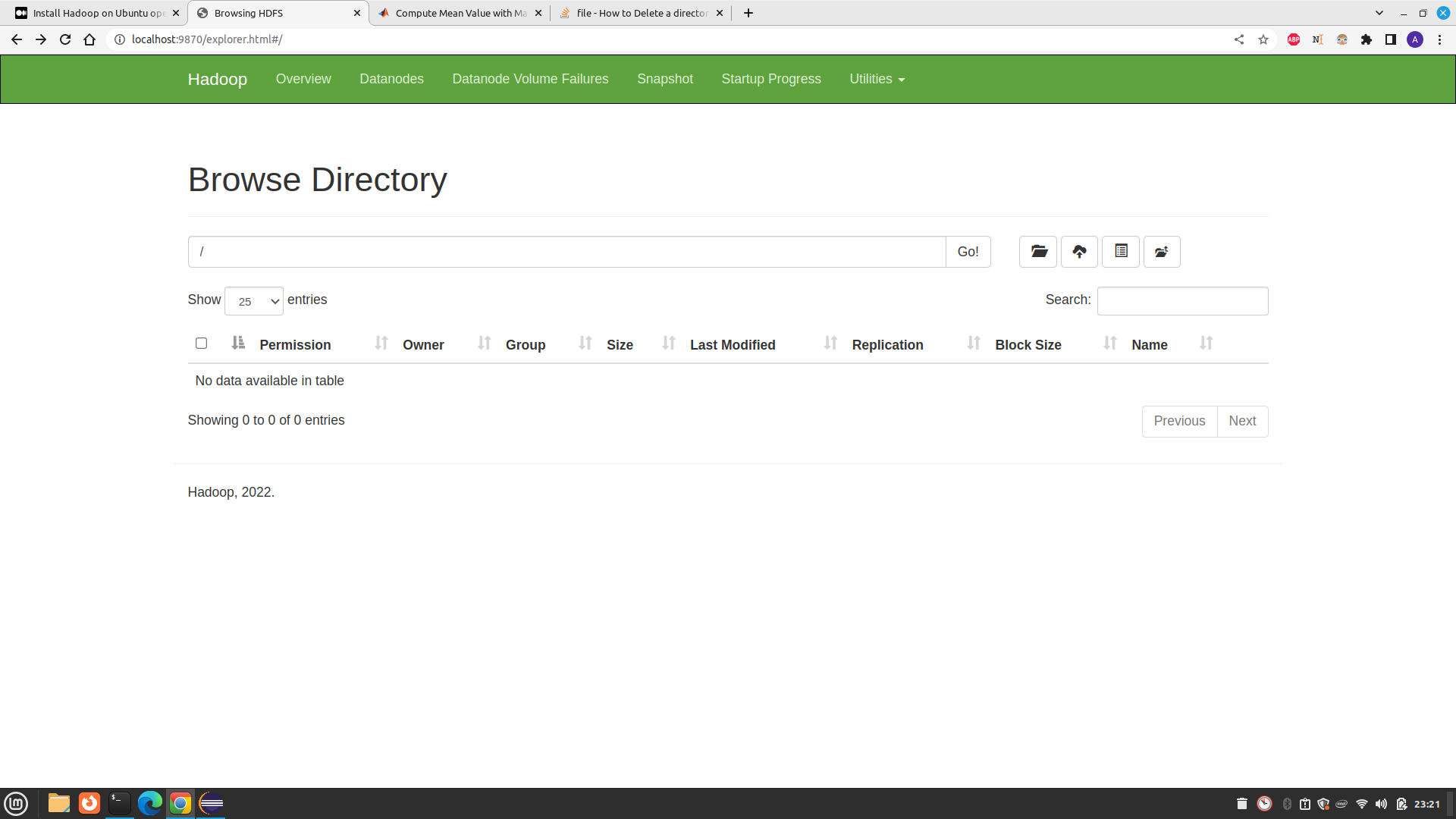
Bda lab

Assignment 1

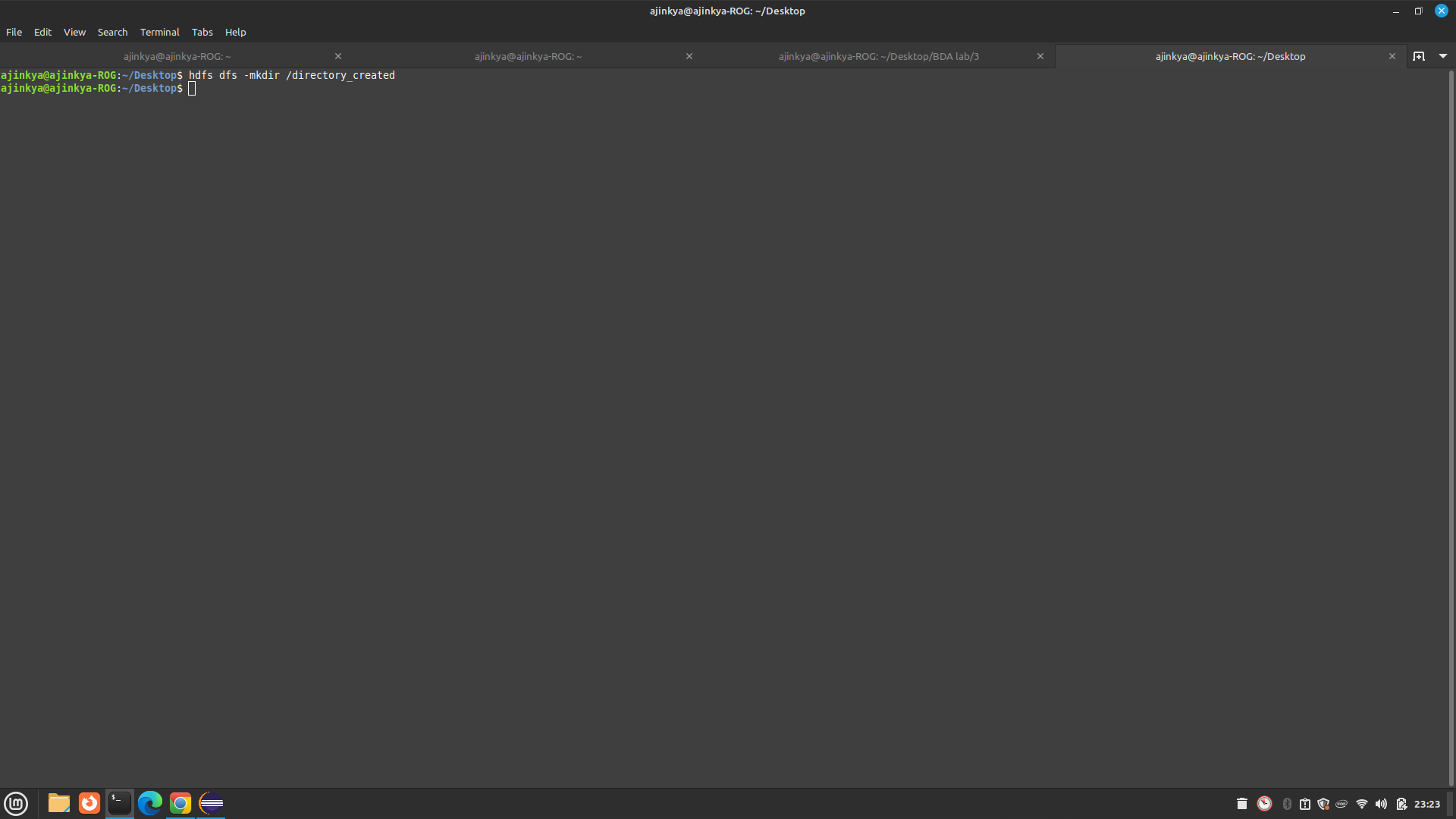
# Ques 1 Hadoop Implementation of file management tasks, such as Adding files and directories, Retrieving files and Deleting files.

**Making directories**

Before making directory



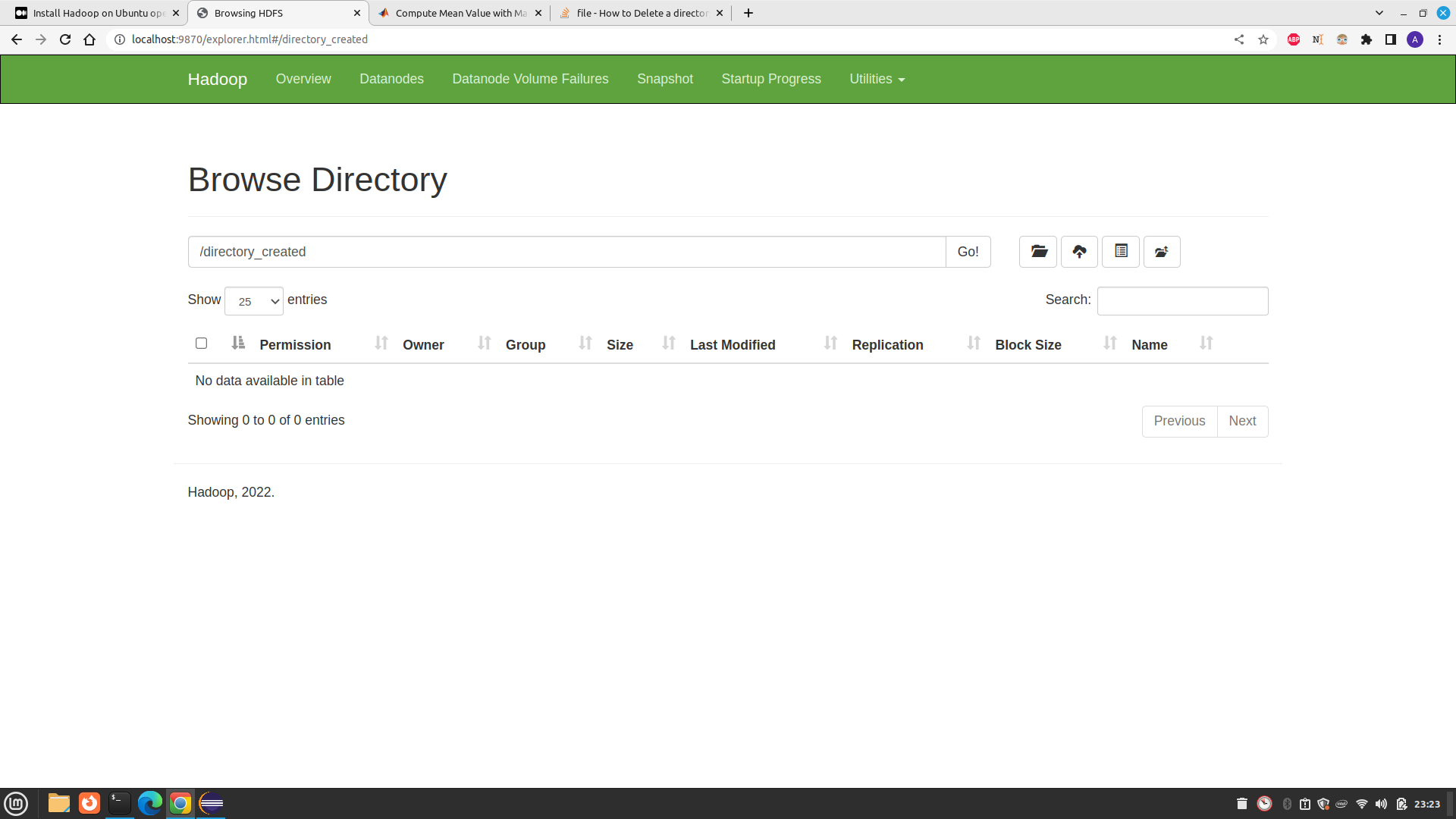
After making directory



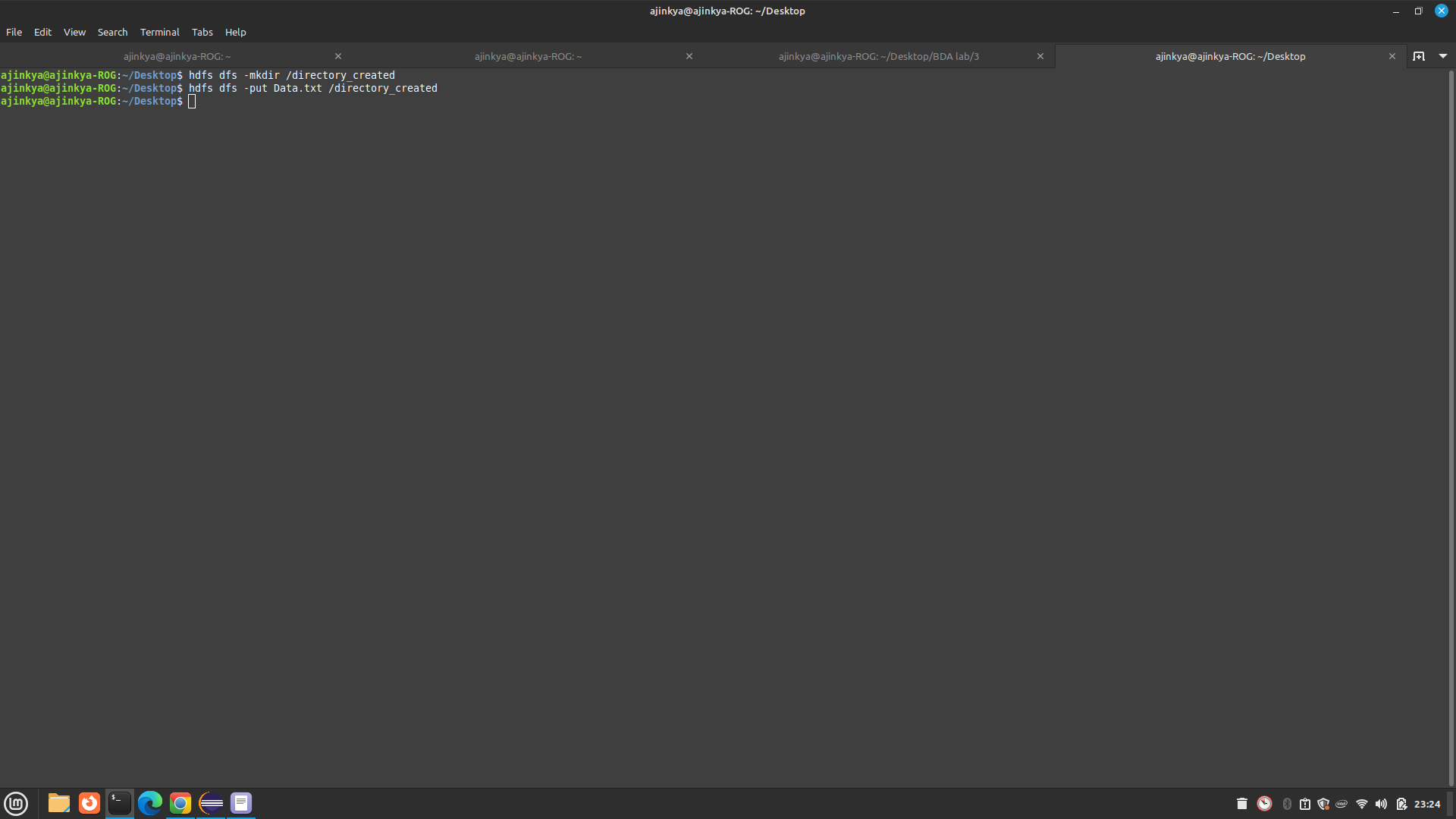


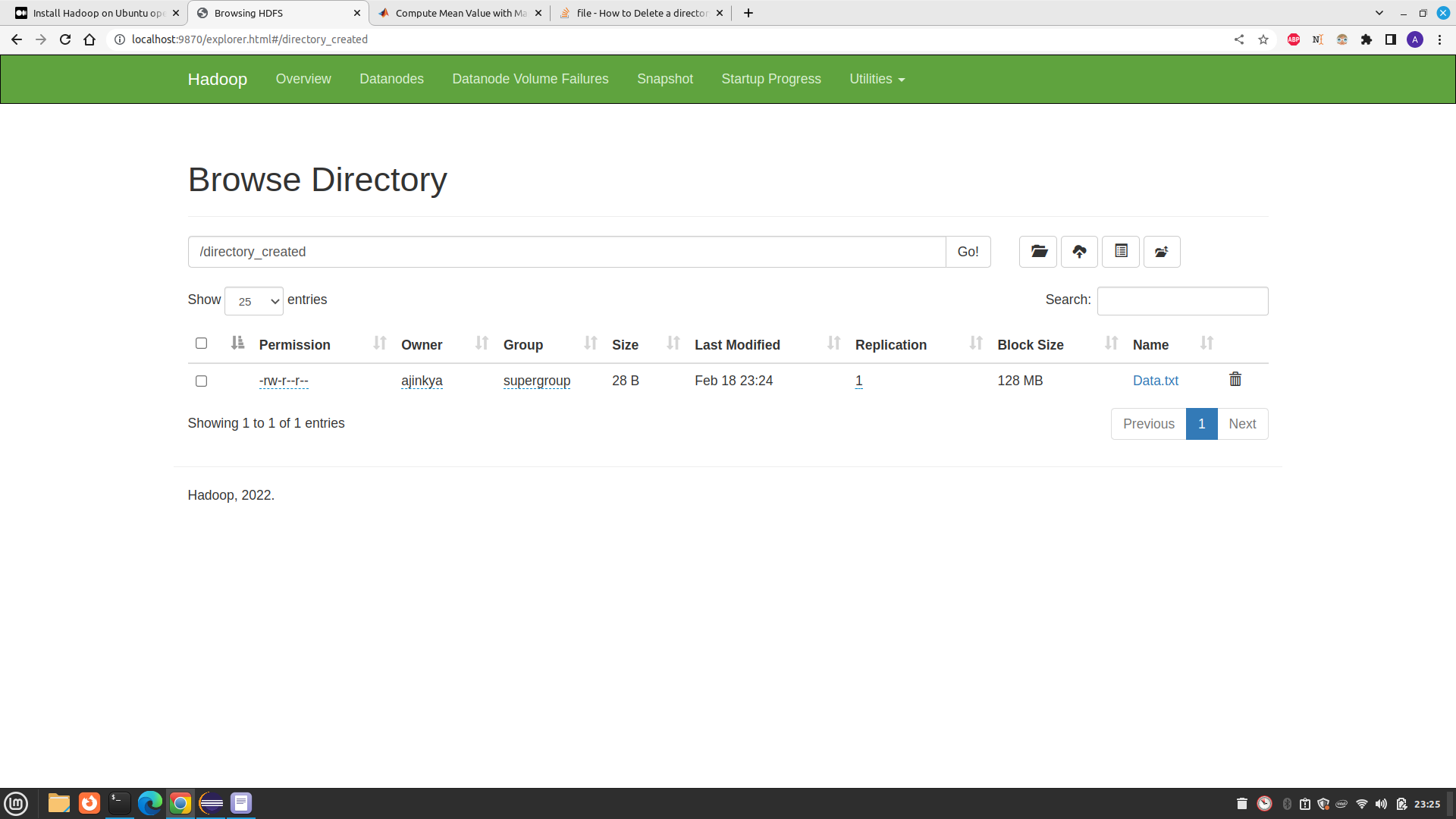
**Adding file**

Before adding file in the above created directory

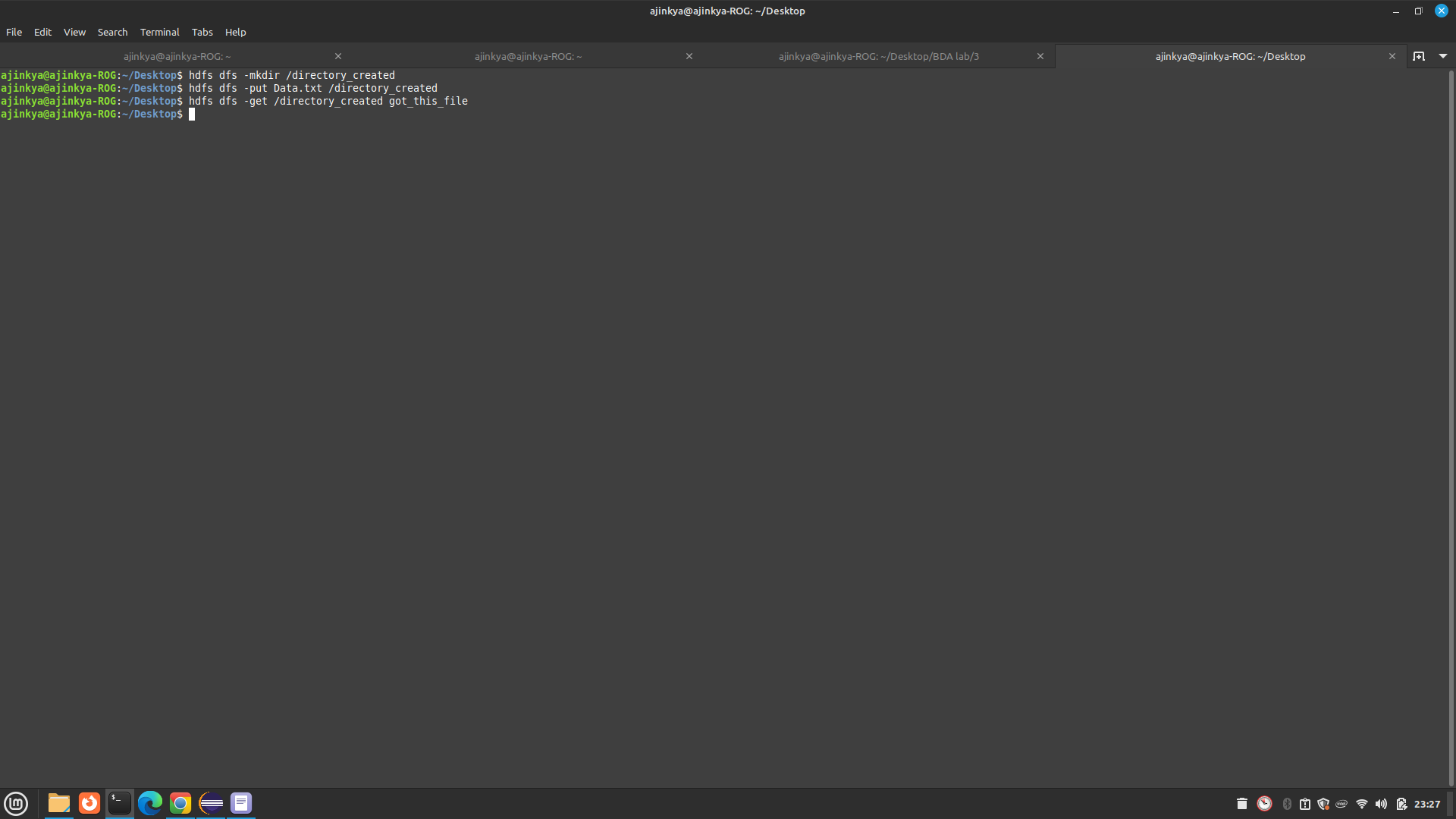


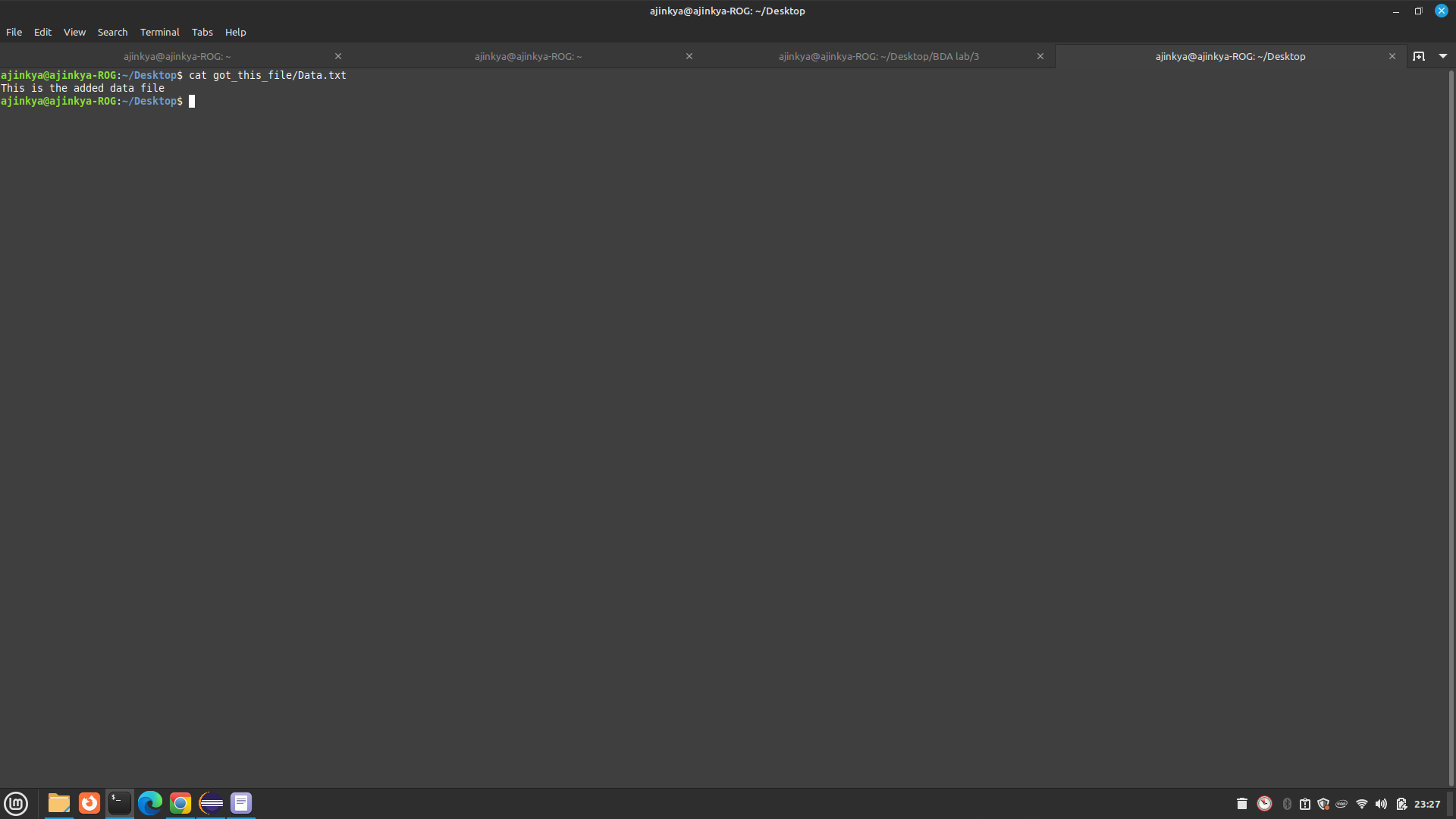
After add file to the directory



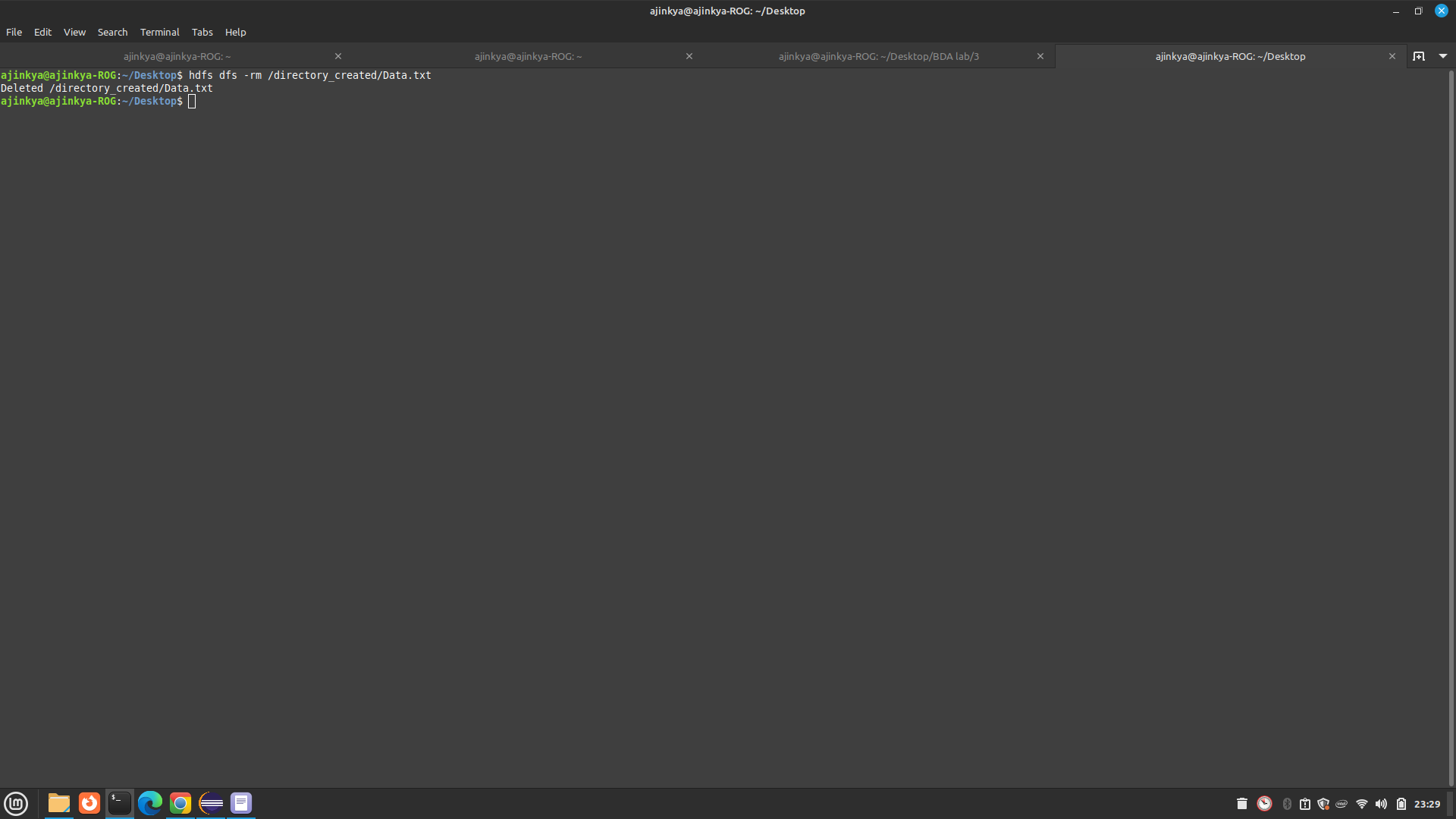


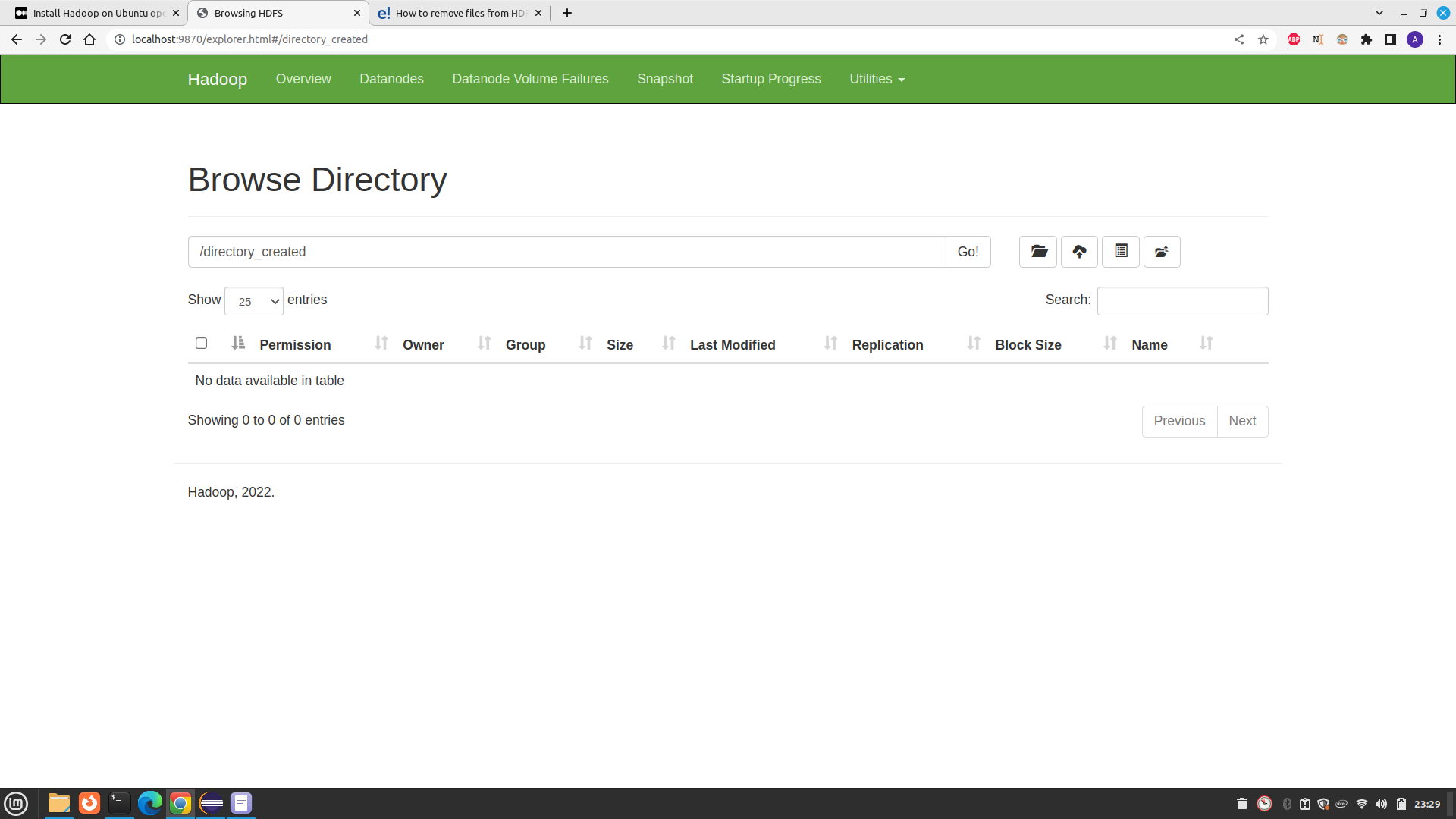
**Retrieving file**





**Deleting file in hadoop**





# Ques 2. Write a program to count the number of even and odd and finding their sum in Map Reduce.

# Ex Input : 1 2 3 4 5 6 7 8 9

# Output : Even 20 // sum of even numbers

# Even 4 // count of even numbers

# Odd 25 // sum of odd numbers

# Odd 5 // count of odd numbers

Code

Driver code

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class EODriver extends Configured implements Tool {

@Override

public int run(String[] args) throws Exception

{

if (args.length < 2)

{

System.out.println("Please enter valid arguments");

return -1;

}

JobConf conf = new JobConf(EODriver.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

conf.setMapperClass(EOMapper.class);

conf.setReducerClass(EOReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

// Main Method

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

{

int exitcode = ToolRunner.run(new EODriver(), args);

System.out.println(exitcode);

}

}

Mapper code

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class EOMapper extends MapReduceBase implements Mapper<LongWritable,

Text, Text, IntWritable> {

@Override

// Map function

public void map(LongWritable key, Text value, OutputCollector<Text,

IntWritable> output, Reporter rep)

throws IOException

{

// Splitting the line into spaces

String data[] = value.toString().split(" ");

for (String num : data)

{

int number = Integer.parseInt(num);

if (number % 2 == 1)

{

// For Odd Numbers

output.collect(new Text("ODD"), new IntWritable(number));

}

else

{

// For Even Numbers

output.collect(new Text("EVEN"),

new IntWritable(number));

}

}

}

}

Reducer Code

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class EOReducer extends MapReduceBase implements Reducer<Text,

IntWritable, Text, IntWritable> {

@Override

// Reduce Function

public void reduce(Text key, Iterator<IntWritable> value,

OutputCollector<Text, IntWritable> output, Reporter rep)

throws IOException

{

// For finding sum and count of even and odd

// you don't have to take different variables

int sum = 0, count = 0;

if (key.equals("ODD"))

{

while (value.hasNext())

{

IntWritable i = value.next();

// Finding sum and count of ODD Numbers

sum += i.get();

count++;

}

}

else

{

while (value.hasNext())

{

IntWritable i = value.next();

// Finding sum and count of EVEN Numbers

sum += i.get();

count++;

}

}

// First sum then count is printed

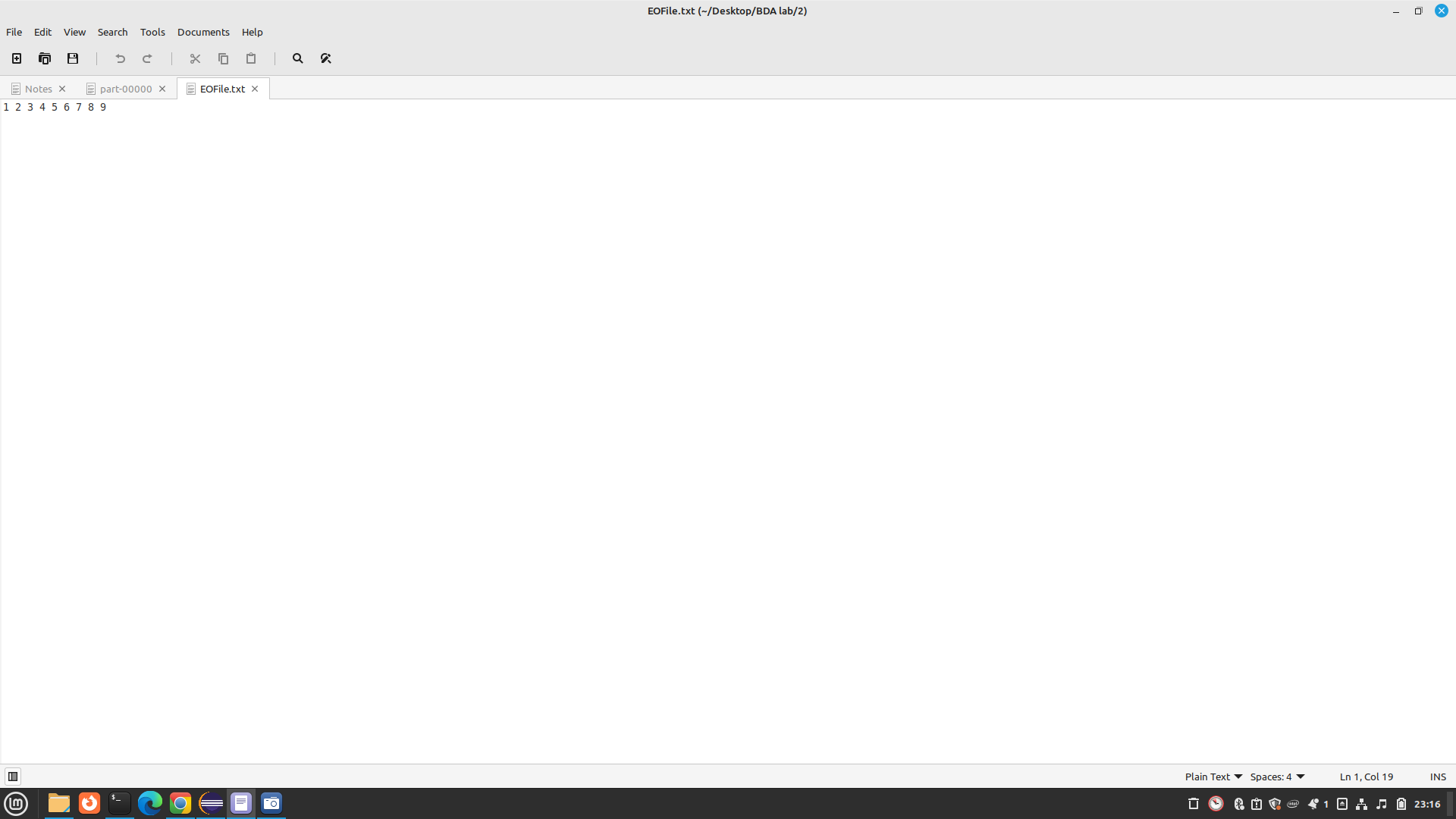
output.collect(key, new IntWritable(sum));

output.collect(key, new IntWritable(count));

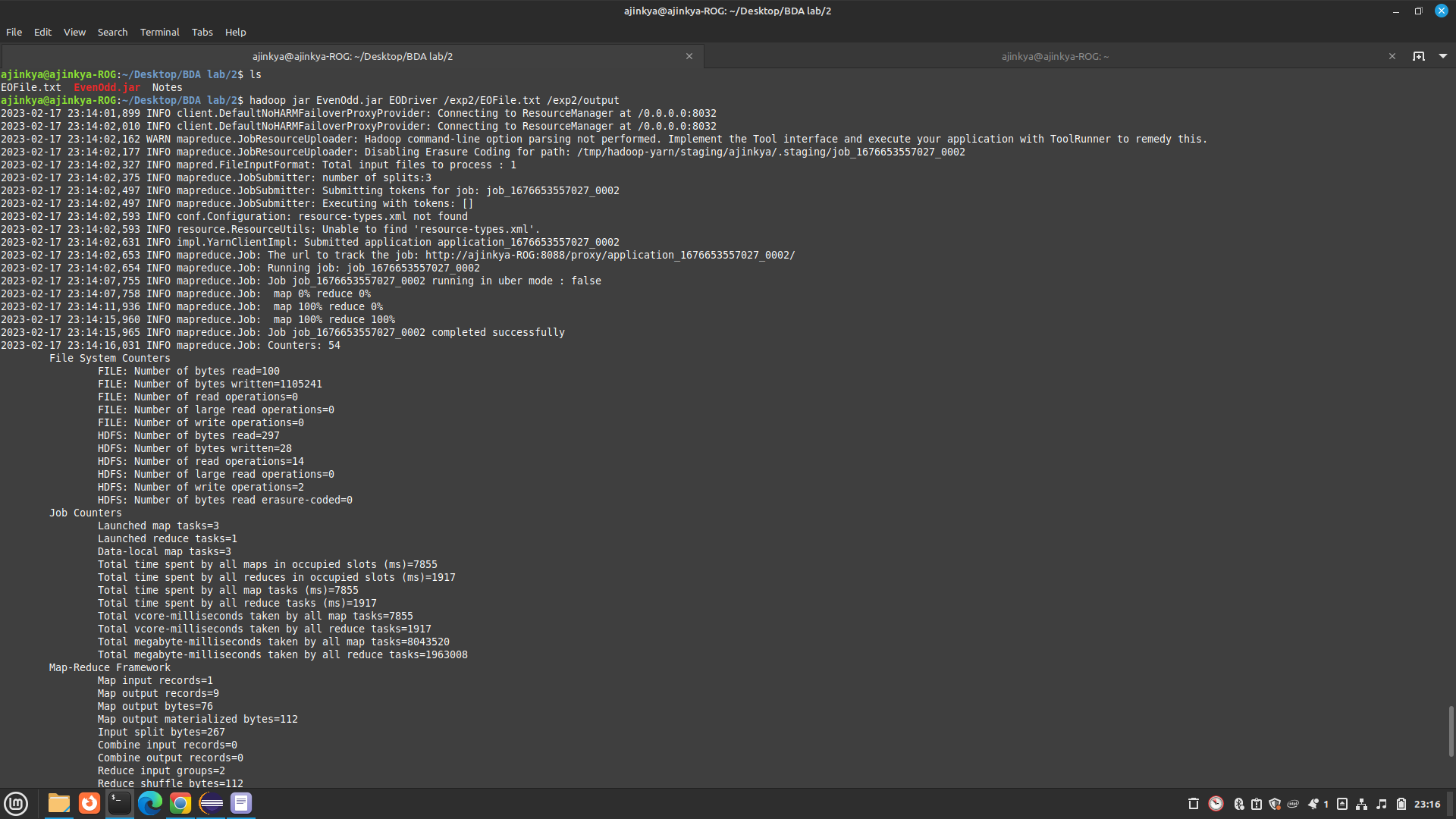
}

}

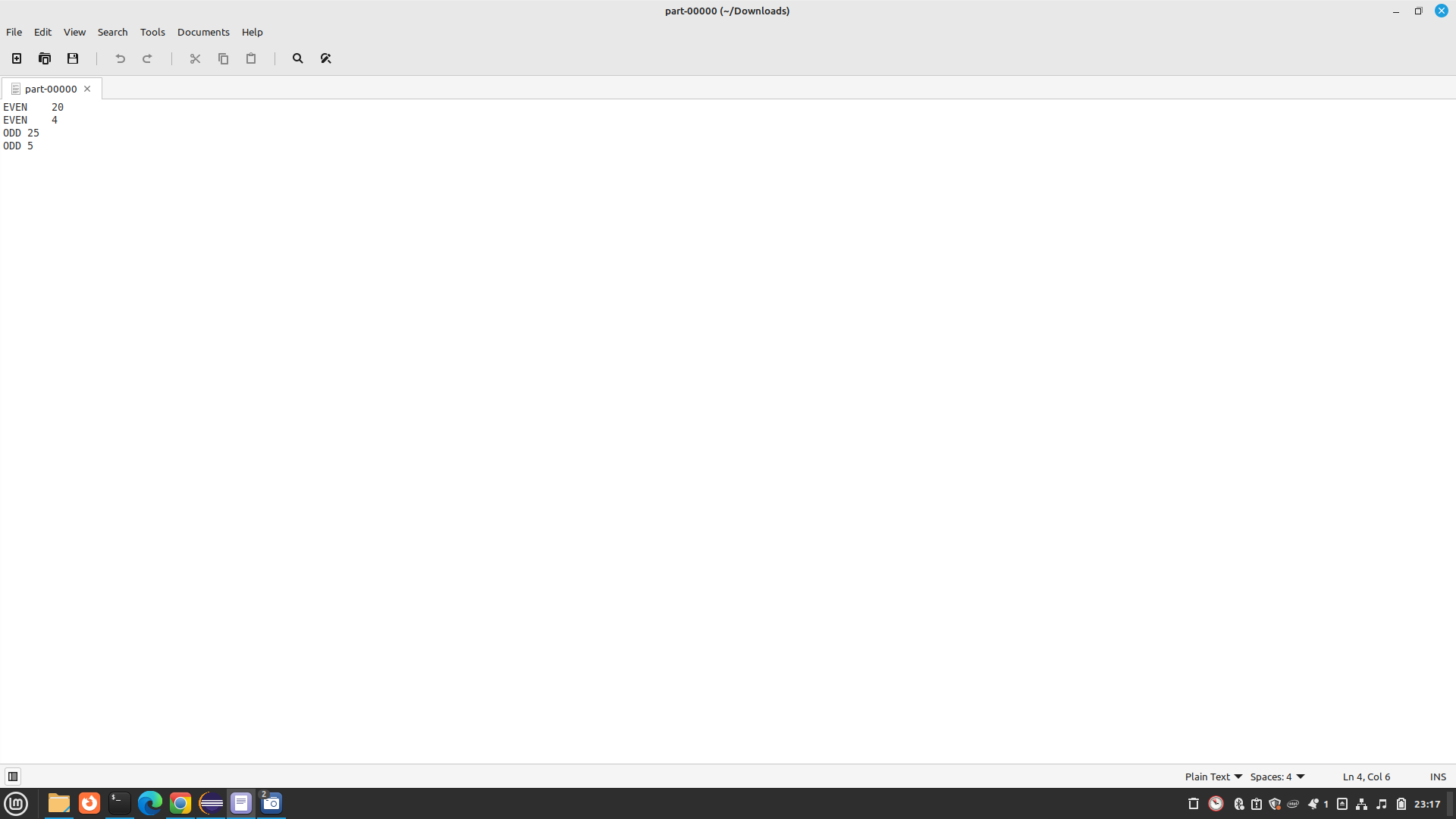
Input file



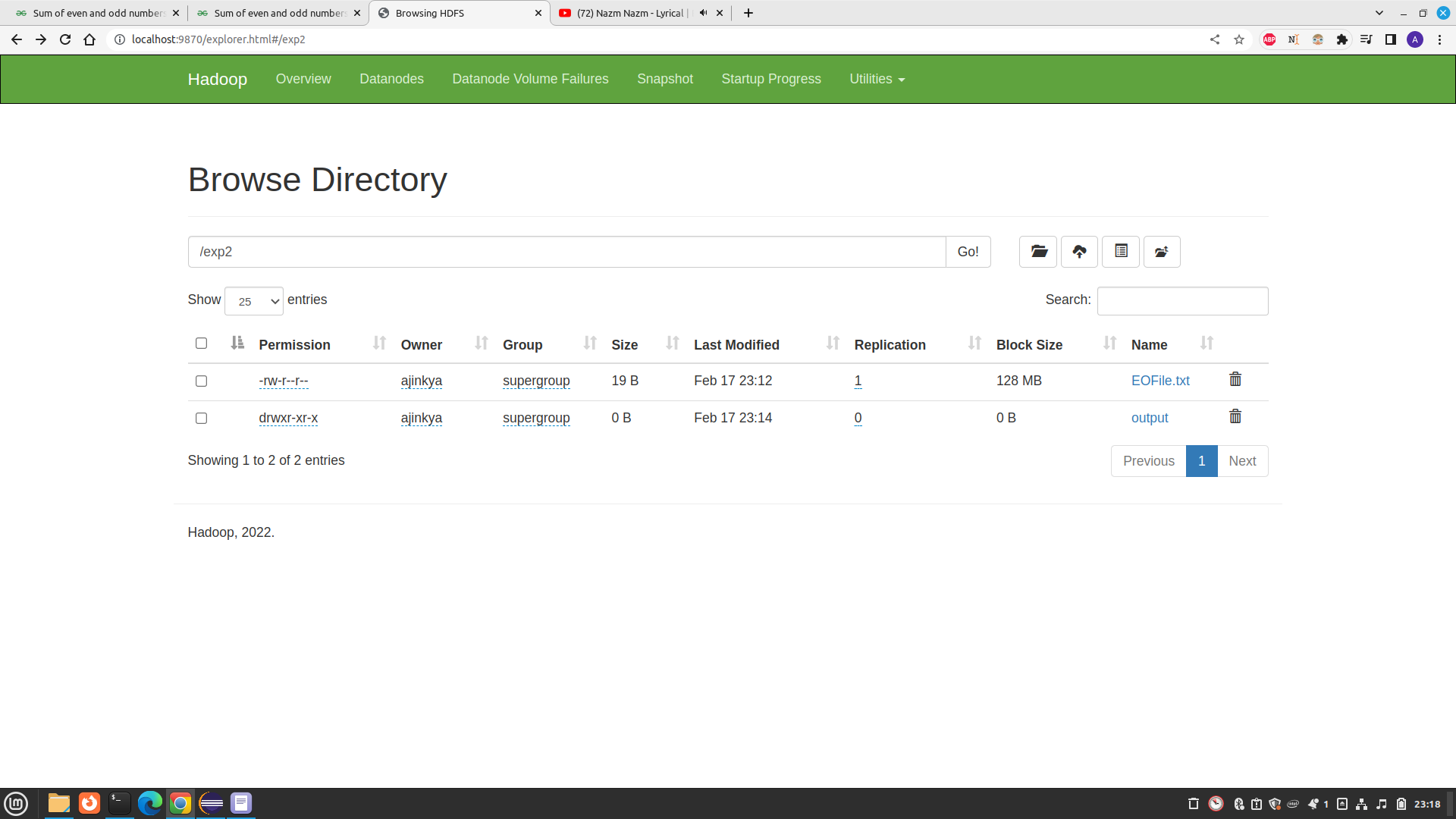
Running code

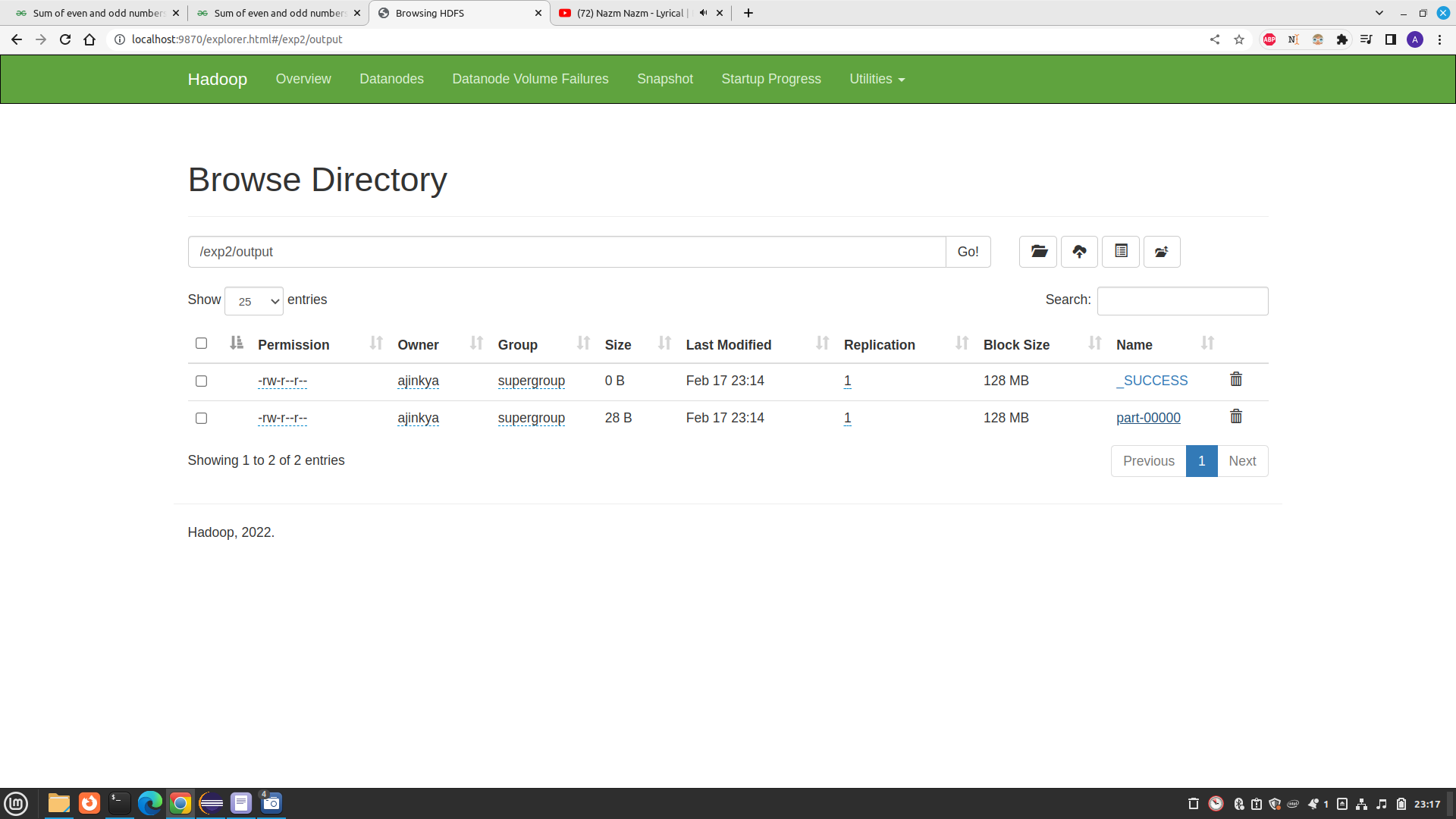


Output of code



Output shown by HDFS





# 3) Write a Map Reduce Code for Finding Average of numbers.

Code

**import** java.io.IOException;

**import** java.util.\*;

**import** org.apache.hadoop.fs.Path;

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

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

**public** **class** Average {

**private** **static** **int** *noOfClasses* = 10;//Used to update the maximum number of split in the mapper class. Value can be modified with a input argument when running the jar.

**public** **static** **class** Map **extends** MapReduceBase **implements** Mapper<LongWritable, Text, Text, Text> {

**private** Text rand\_class = **new** Text();

**public** **void** map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) **throws** IOException, NumberFormatException{

String line = value.toString();

**try** {

**if**(line!=**null** || line!="") {

/\*Text file should provide the numbers line by line\*/

Double.*valueOf*(line);

rand\_class.set("class-"+**new** Random().nextInt(*noOfClasses*));

output.collect(rand\_class, **new** Text(line));

}

}**catch**(NumberFormatException ex ){

**throw** **new** NumberFormatException("Invalid Number(="+line+") in the Input File");

}

}

}

**public** **static** **class** Combine **extends** MapReduceBase **implements** Reducer<Text, Text, Text, Text> {

**public** **void** reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) **throws** IOException {

**double** sum = 0;

**int** size = 0;

**while** (values.hasNext()) {

/\* Each values in the iterator of a class is scanned through and calculates the sum while counting the number of elements within the class \*/

sum += Double.*parseDouble*(values.next().toString());

size += 1;

}

/\*Sum per class is assigned to the 'Average' Key and the sum and the no of elements contributed to this sum will be given as the value\*/

output.collect(**new** Text("Aver"), **new** Text(sum + " " + size));

}

}

**public** **static** **class** Reduce **extends** MapReduceBase **implements** Reducer<Text, Text, Text, Text> {

**public** **void** reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) **throws** IOException {

String[] valSplit;

**int** sum = 0;

**double** size = 0;

**while** (values.hasNext()) {

valSplit = values.next().toString().split(" ");

sum += Double.*parseDouble*(valSplit[0]);

size += Integer.*parseInt*(valSplit[1]);

}

**double** avg = sum/size;

output.collect(**new** Text("Average"), **new** Text(avg+""));

}

}

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

JobConf conf = **new** JobConf(Average.**class**);

conf.setJobName("AverageProb");

conf.setOutputKeyClass(Text.**class**);

conf.setOutputValueClass(Text.**class**);

conf.setMapperClass(Map.**class**);

conf.setCombinerClass(Combine.**class**);

conf.setReducerClass(Reduce.**class**);

conf.setInputFormat(TextInputFormat.**class**);

conf.setOutputFormat(TextOutputFormat.**class**);

FileInputFormat.*setInputPaths*(conf, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(conf, **new** Path(args[1]));

**try** {

*noOfClasses* = Integer.*parseInt*(args[2]);

}**catch**(Exception e) {

*noOfClasses* = 10;

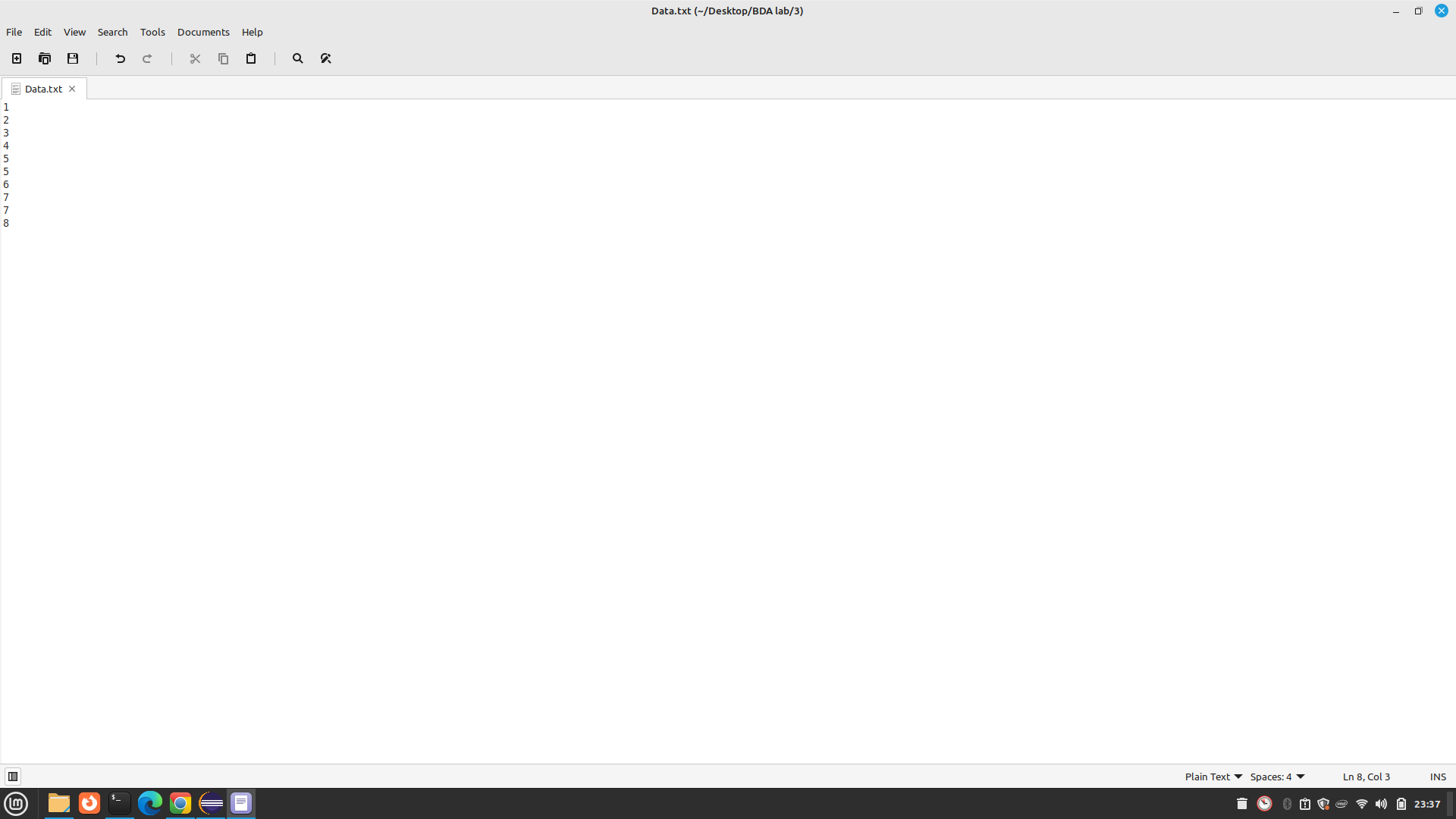
}

JobClient.*runJob*(conf);

}

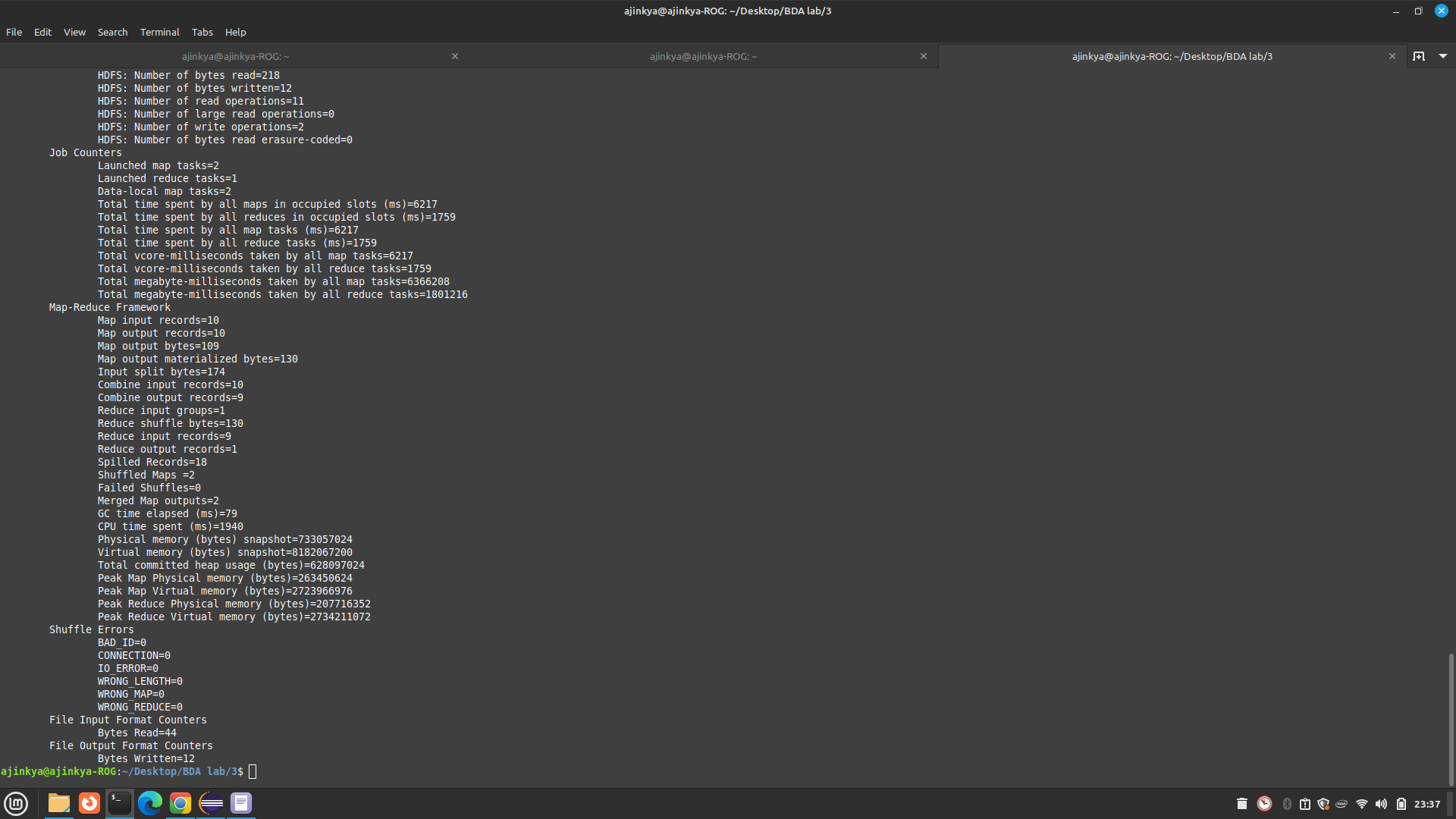
}

Data.txt

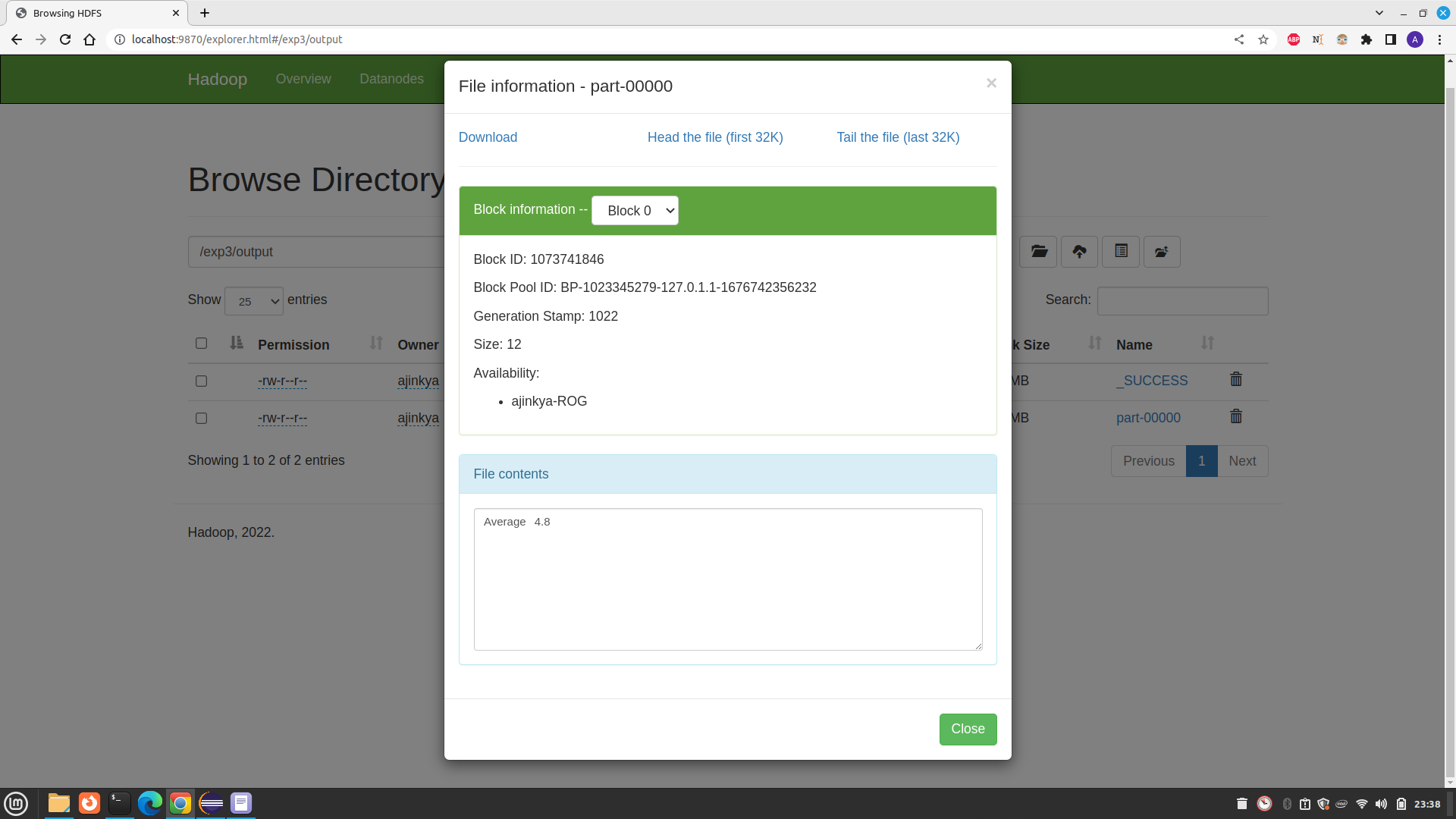


Running code





Output of the code



# Ques 4) Write a basic Word Count Map Reduce program and execute it understand Map Reduce Paradigm.

Code:

**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]));

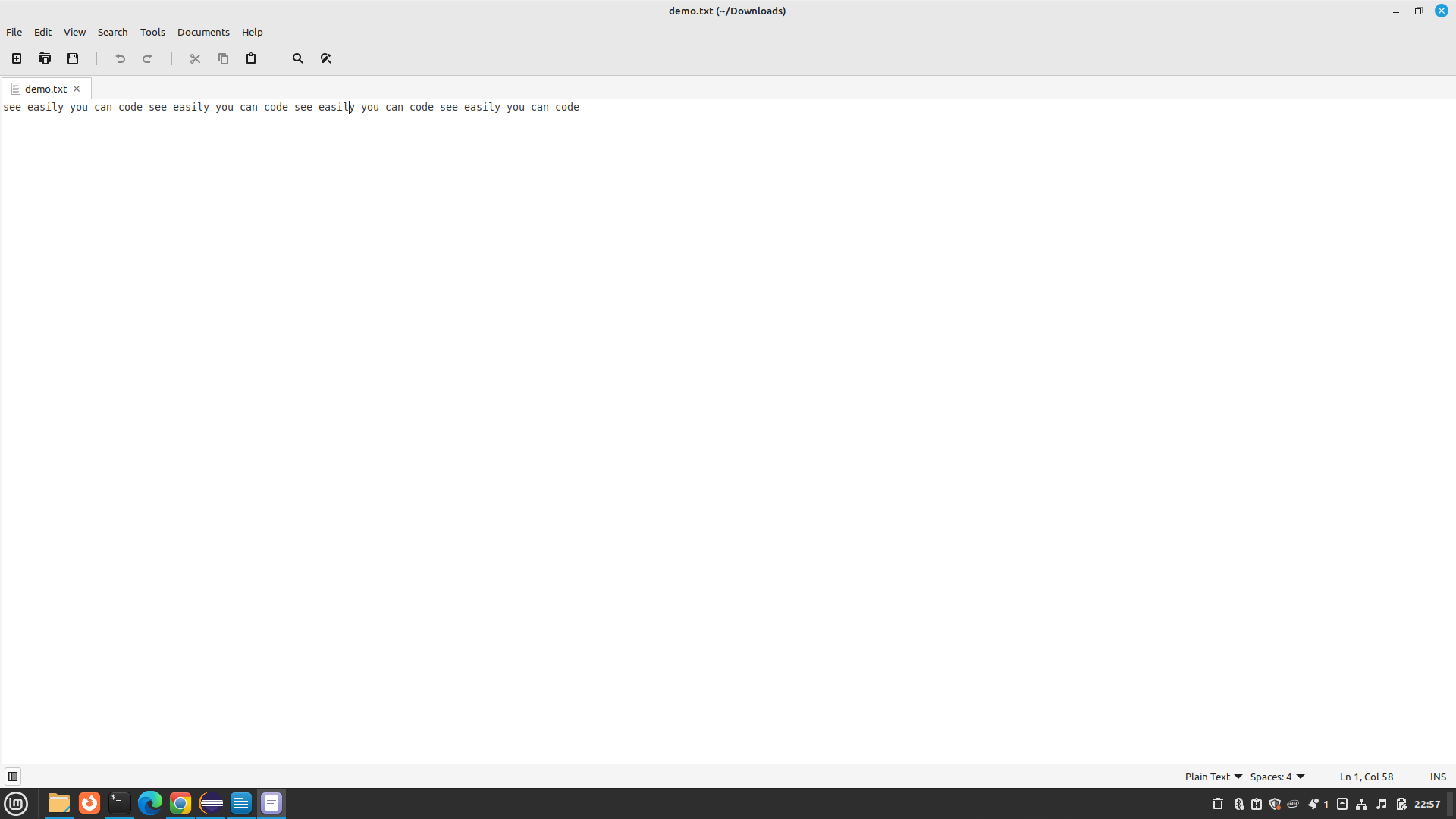
FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

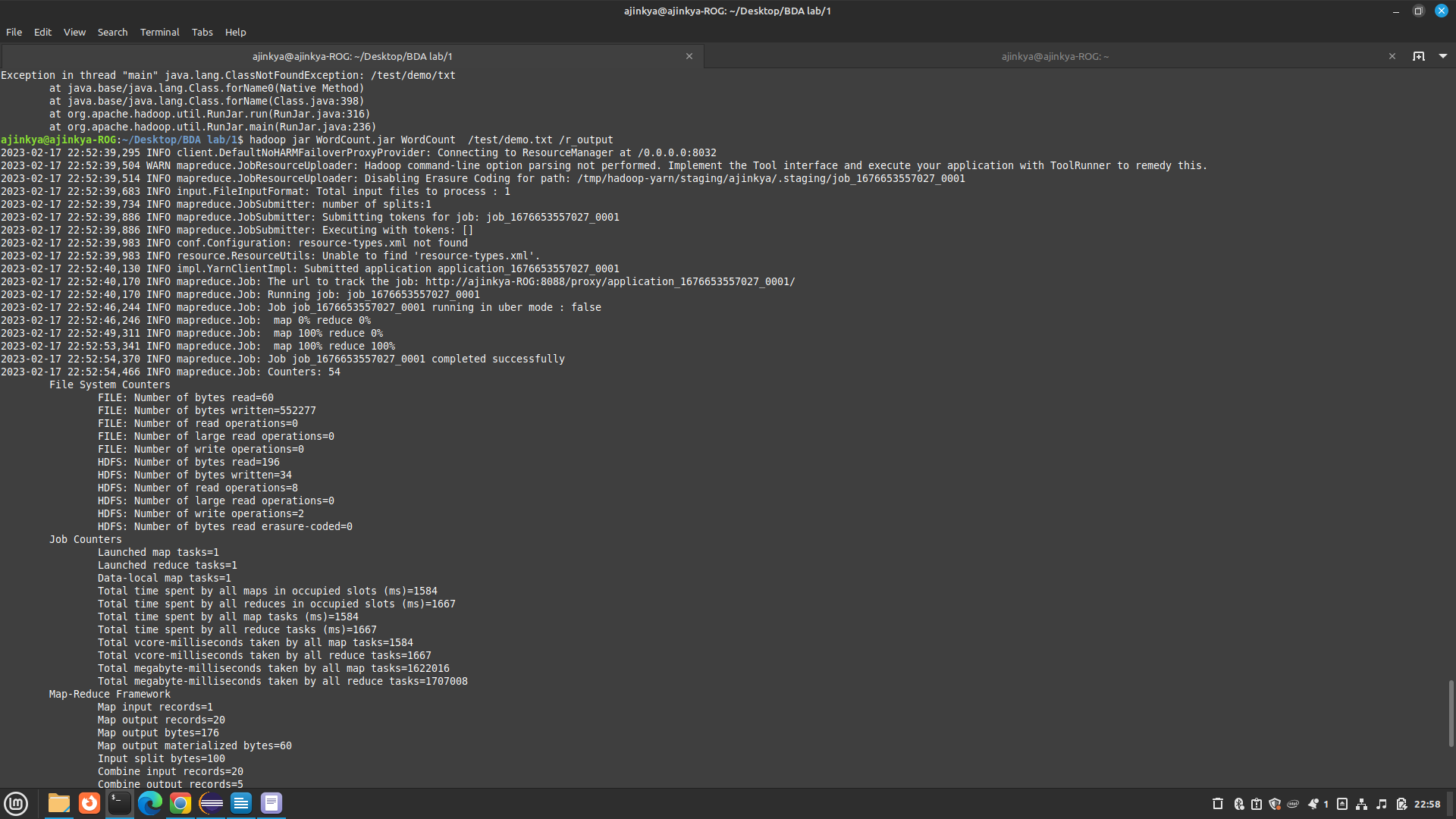
}

}

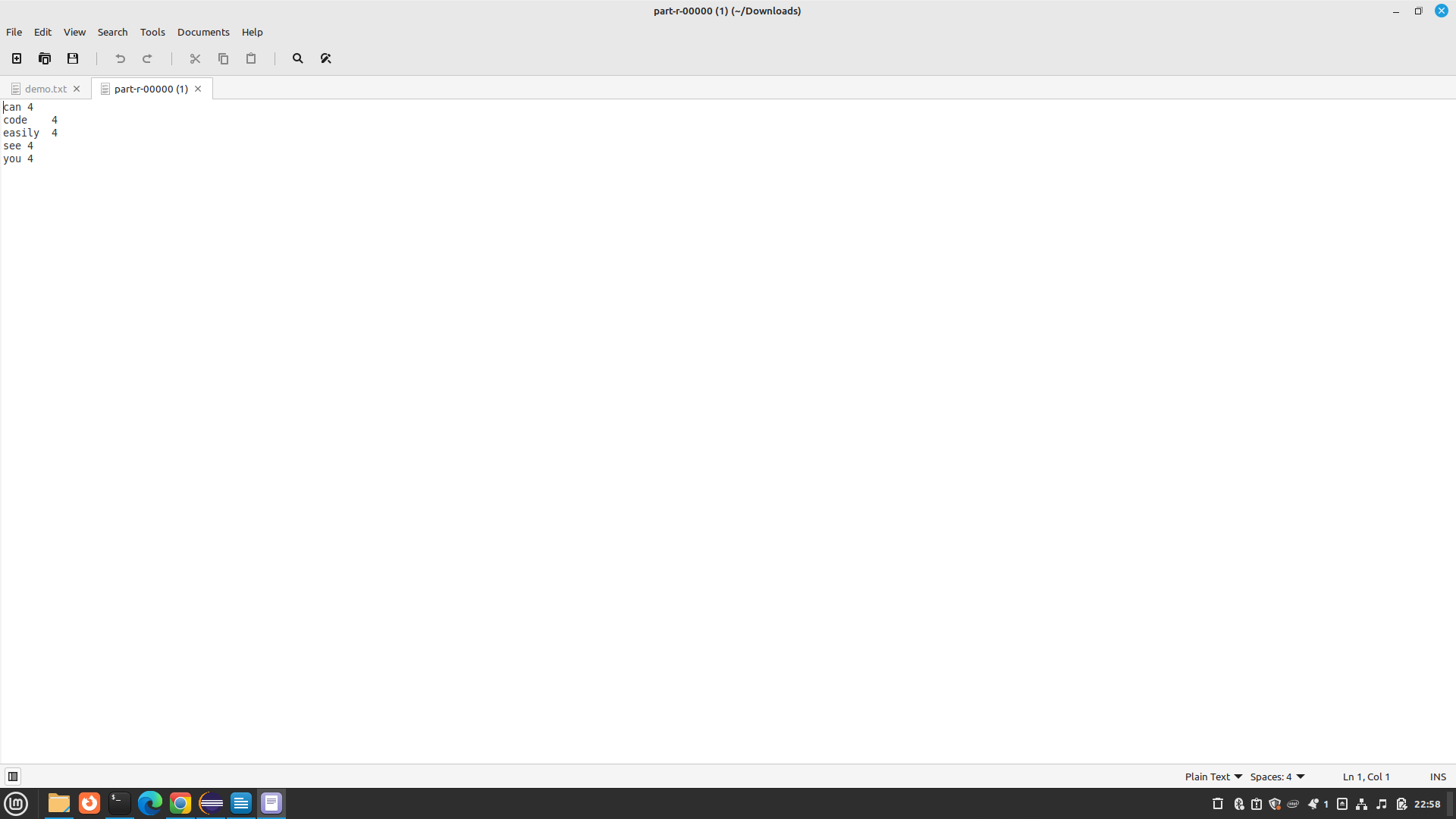
Input data

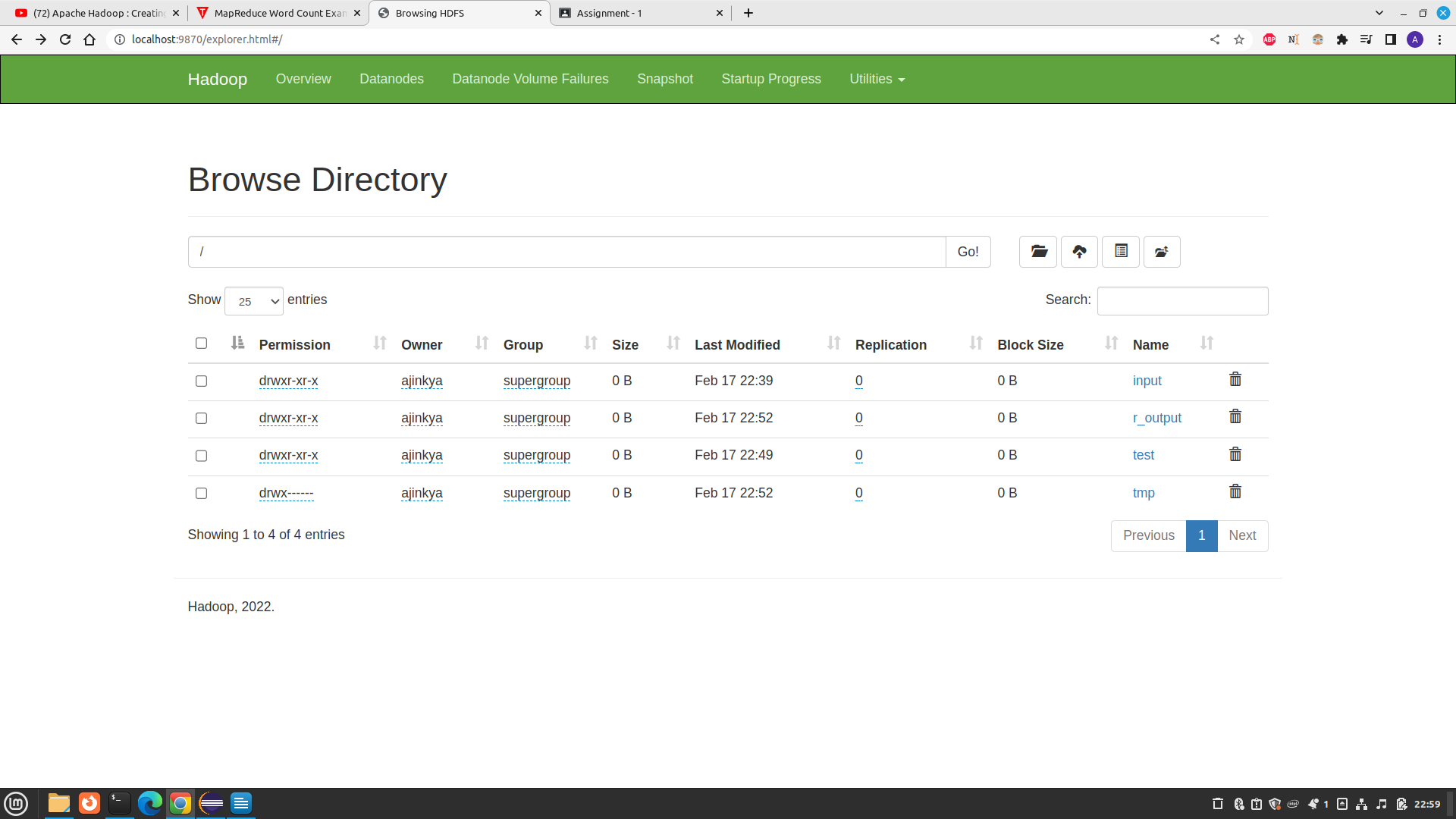


Runnig code



Output data

Directory after experiment



# Ques 5. Write a Map Reduce program to analyzed weather dataset and print whether the day is shinny or cool day.

// importing Libraries

**import** java.io.IOException;

**import** java.util.Iterator;

**import** org.apache.hadoop.fs.Path;

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

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

**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.mapreduce.lib.input.TextInputFormat;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.conf.Configuration;

**public** **class** HotOrCool {

**public** **static** **class** MaxTemperatureMapper **extends**

Mapper<LongWritable, Text, Text, Text> {

// the data in our data set with

// this value is inconsistent data

**public** **static** **final** **int** ***MISSING*** = 9999;

@Override

**public** **void** map(LongWritable arg0, Text Value, Context context)

**throws** IOException, InterruptedException {

// Convert the single row(Record) to

// String and store it in String

// variable name line

String line = Value.toString();

// Check for the empty line

**if** (!(line.length() == 0)) {

// from character 6 to 14 we have

// the date in our dataset

String date = line.substring(6, 14);

// similarly we have taken the maximum

// temperature from 39 to 45 characters

**float** temp\_Max = Float.*parseFloat*(line.substring(39, 45).trim());

// similarly we have taken the minimum

// temperature from 47 to 53 characters

**float** temp\_Min = Float.*parseFloat*(line.substring(47, 53).trim());

// if maximum temperature is

// greater than 30, it is a hot day

**if** (temp\_Max > 30.0) {

// Hot day

context.write(**new** Text("The Day is Shiny Day :" + date),

**new** Text(String.*valueOf*(temp\_Max)));

}

// if the minimum temperature is

// less than 15, it is a cold day

**if** (temp\_Min < 15) {

// Cold day

context.write(**new** Text("The Day is Cool Day :" + date),

**new** Text(String.*valueOf*(temp\_Min)));

}

}

}

}

**public** **static** **class** MaxTemperatureReducer **extends**

Reducer<Text, Text, Text, Text> {

**public** **void** reduce(Text Key, Iterator<Text> Values, Context context)

**throws** IOException, InterruptedException {

// putting all the values in

// temperature variable of type String

String temperature = Values.next().toString();

context.write(Key, **new** Text(temperature));

}

}

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

// reads the default configuration of the

// cluster from the configuration XML files

Configuration conf = **new** Configuration();

// Initializing the job with the

// default configuration of the cluster

Job job = **new** ~~Job~~(conf, "weather example");

// Assigning the driver class name

job.setJarByClass(HotOrCool.**class**);

// Key type coming out of mapper

job.setMapOutputKeyClass(Text.**class**);

// value type coming out of mapper

job.setMapOutputValueClass(Text.**class**);

// Defining the mapper class name

job.setMapperClass(MaxTemperatureMapper.**class**);

// Defining the reducer class name

job.setReducerClass(MaxTemperatureReducer.**class**);

// Defining input Format class which is

// responsible to parse the dataset

// into a key value pair

job.setInputFormatClass(TextInputFormat.**class**);

// Defining output Format class which is

// responsible to parse the dataset

// into a key value pair

job.setOutputFormatClass(TextOutputFormat.**class**);

// setting the second argument

// as a path in a path variable

Path OutputPath = **new** Path(args[1]);

// Configuring the input path

// from the filesystem into the job

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

// Configuring the output path from

// the filesystem into the job

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

// deleting the context path automatically

// from hdfs so that we don't have

// to delete it explicitly

OutputPath.getFileSystem(conf).~~delete~~(OutputPath);

// exiting the job only if the

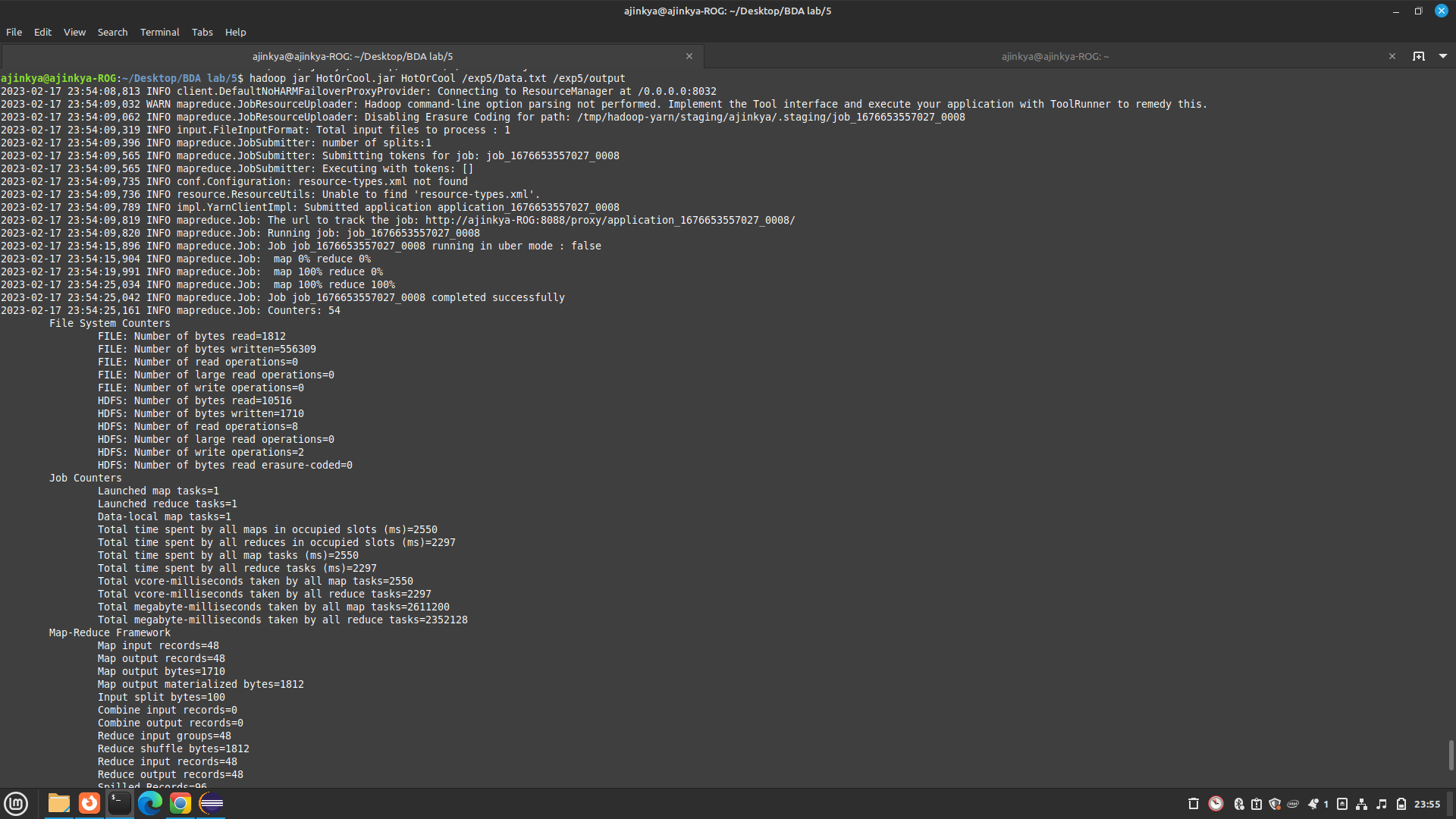
// flag value becomes false

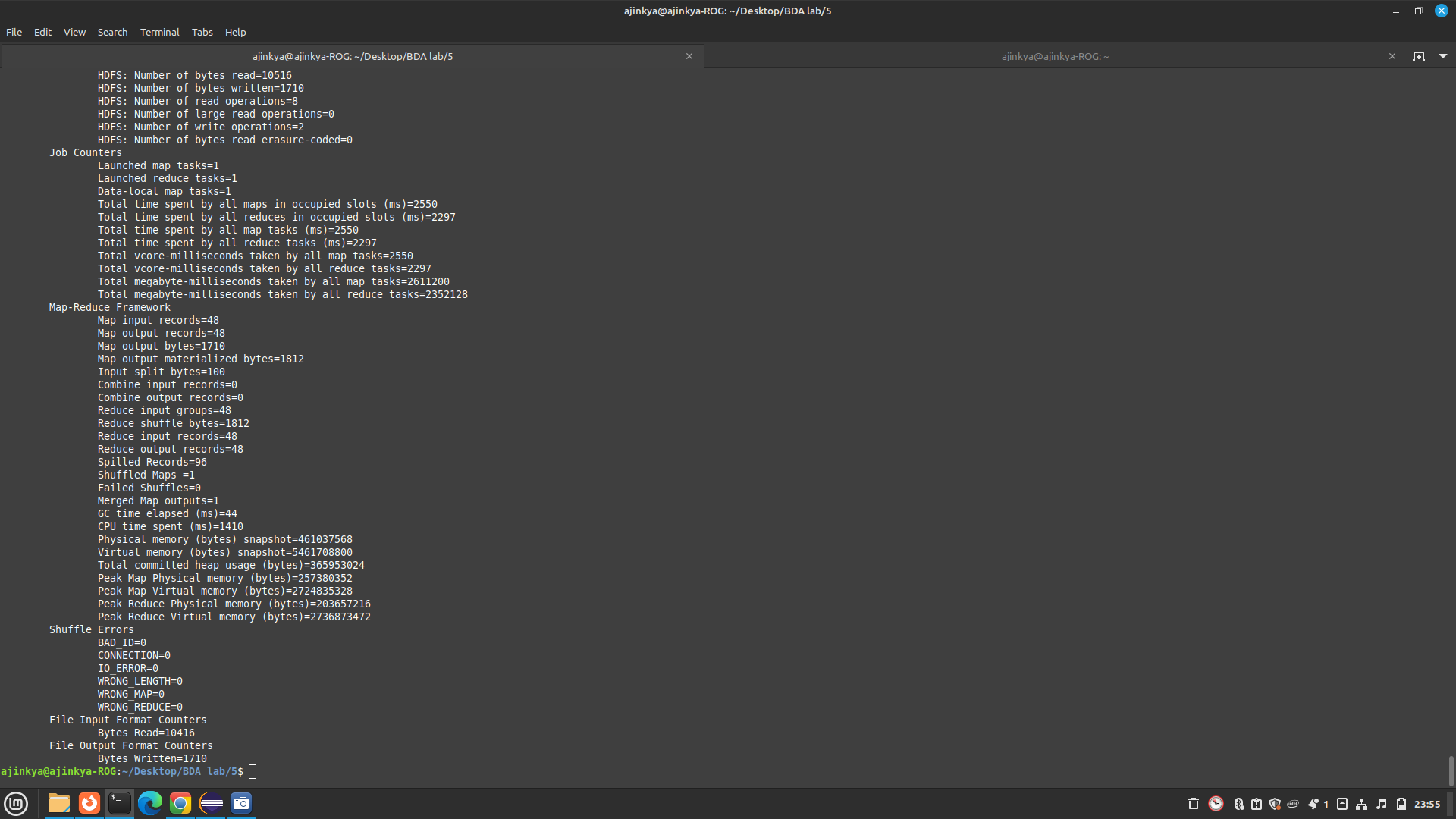
System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

}

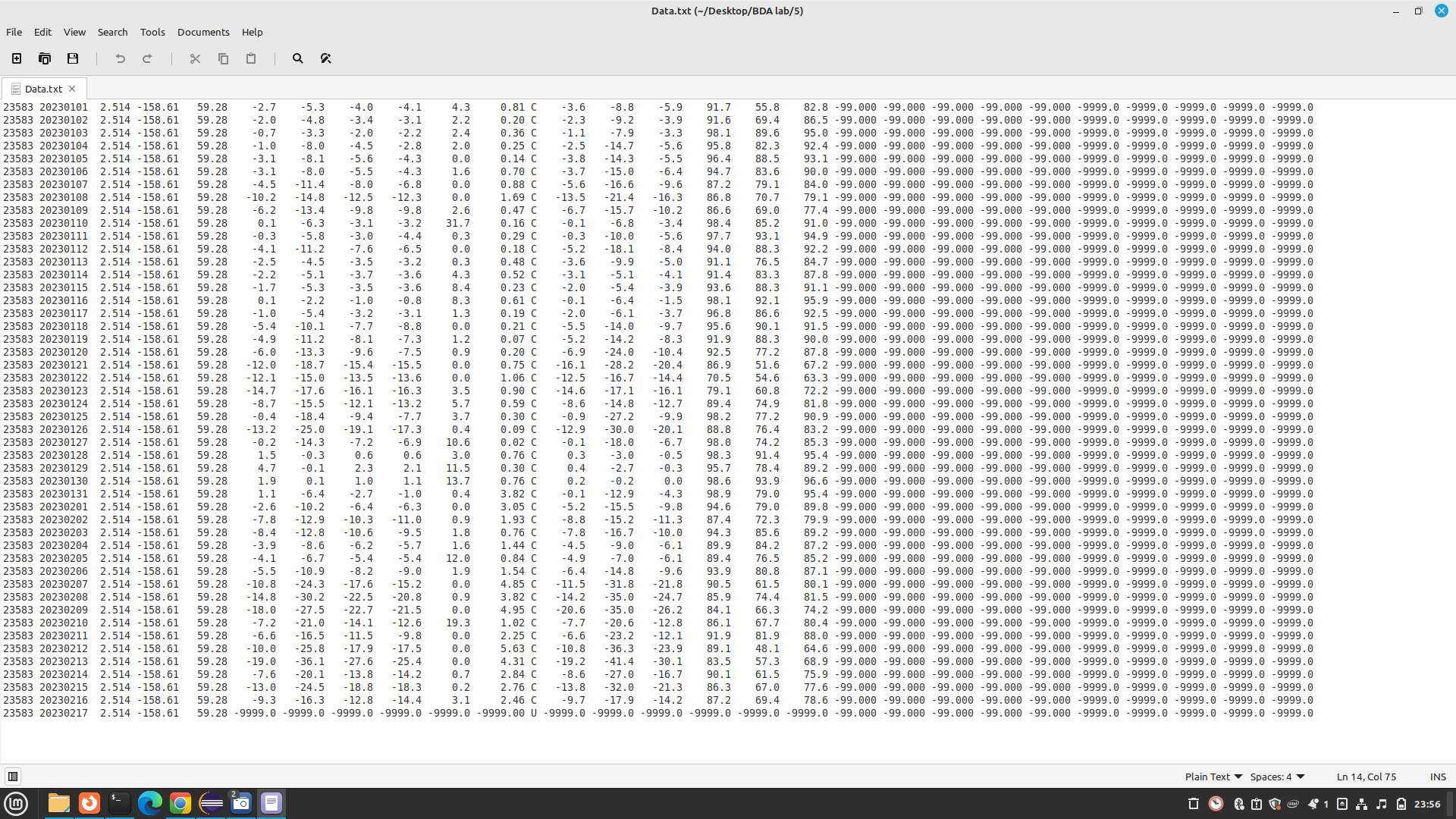
}

Runnig code

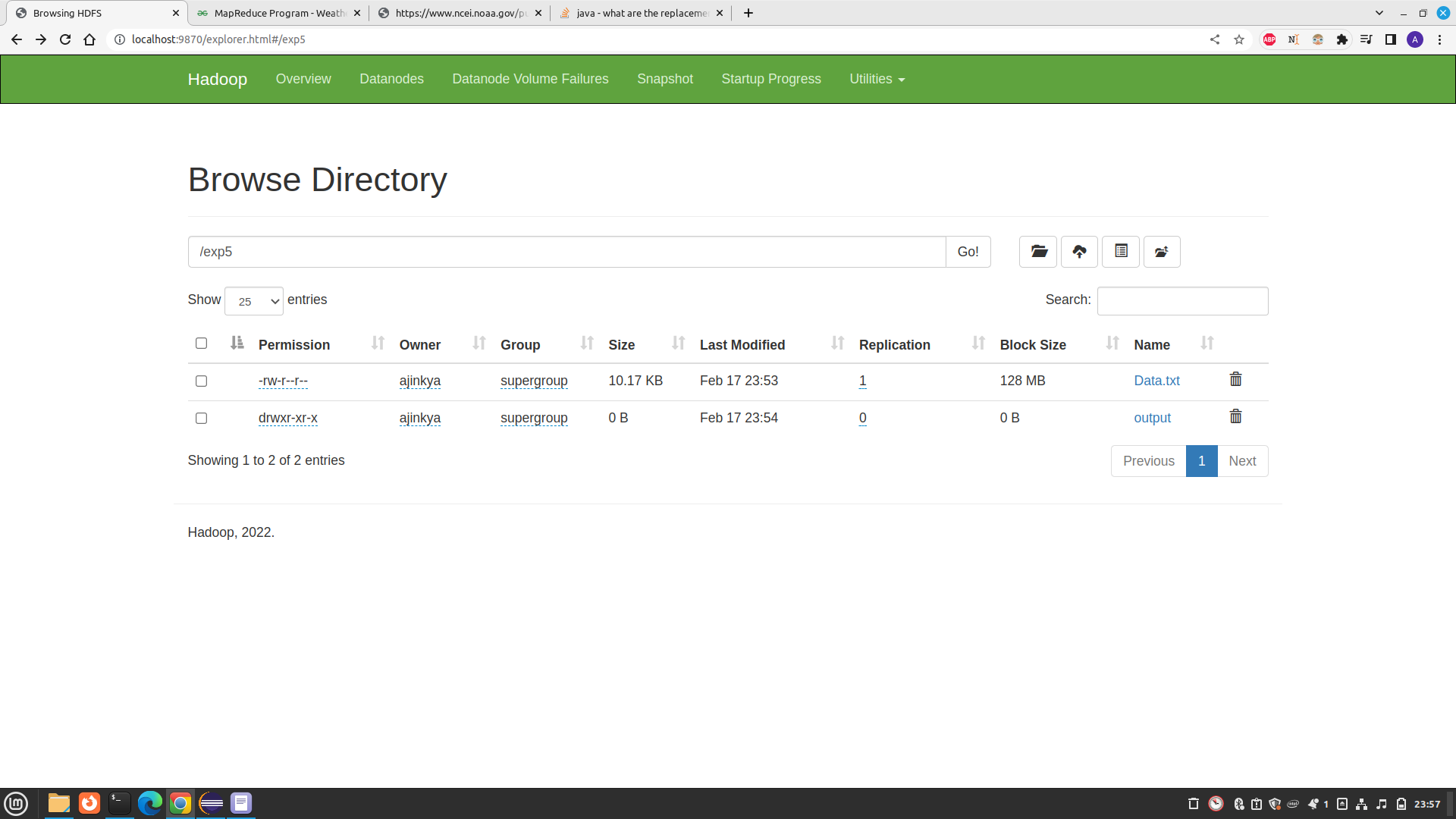


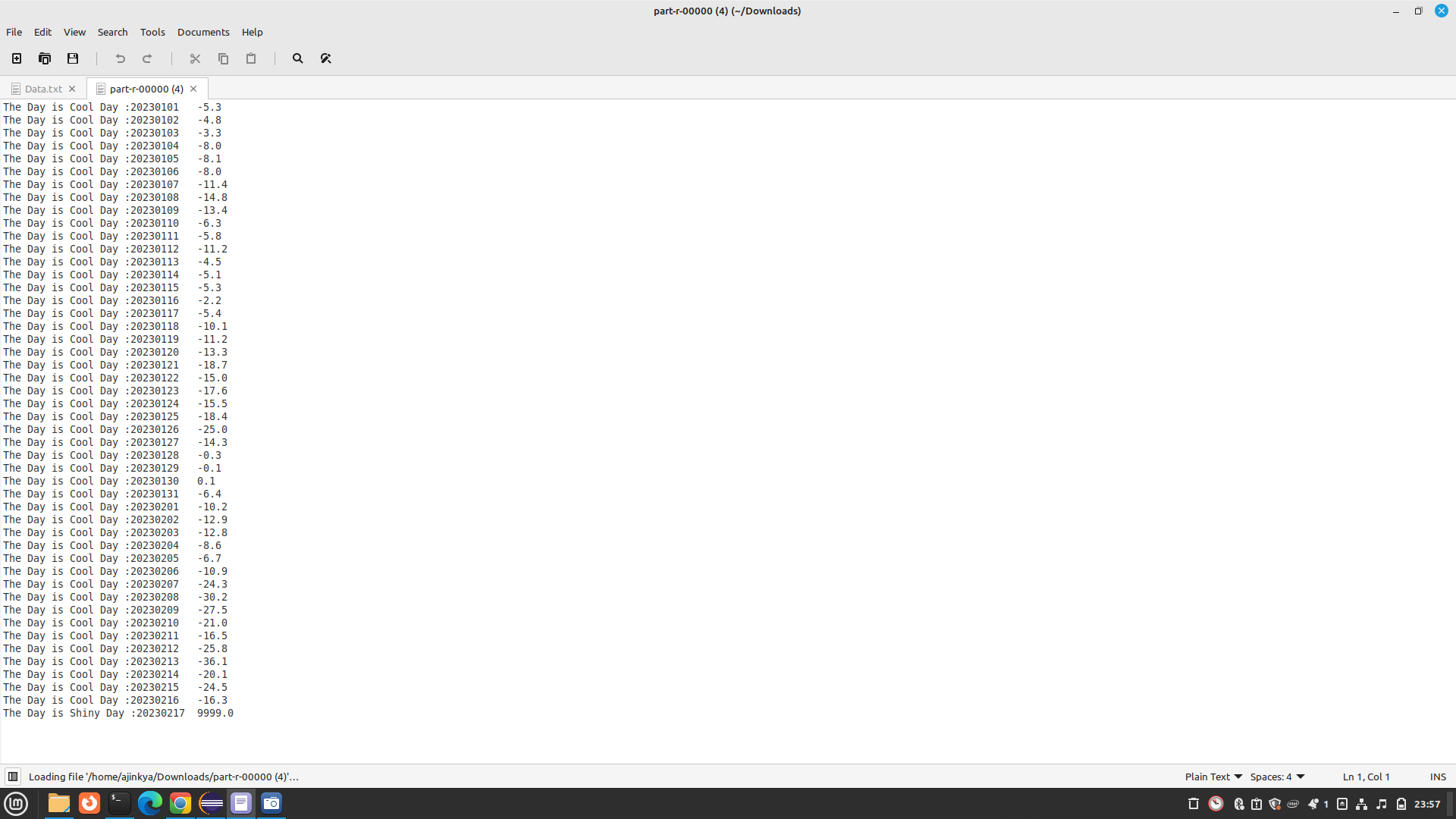


Input Data



Output folder

Output



# Ques 6. Analyzing the Titanic Disaster dataset, for finding the average age of male and female persons died in this disaster with Map Reduce Hadoop.\

# Code

Code

**import** java.io.IOException;

**import** org.apache.hadoop.fs.Path;

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

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

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

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

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

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

**public** **class** Average\_age {

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, IntWritable> {

**private** Text gender = **new** Text();

**private** IntWritable age = **new** IntWritable();

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

String line = value.toString();

String str[]=line.split(",");

**if**(str.length>6){

gender.set(str[4]);

**if**((str[1].equals("0")) ){

**if**(str[5].matches("\\d+")){

**int** i=Integer.*parseInt*(str[5]);

age.set(i);

}

}

}

context.write(gender, age);

}

}

**public** **static** **class** Reduce **extends** Reducer<Text,IntWritable, Text, IntWritable> {

**public** **void** reduce(Text key, Iterable<IntWritable> values, Context context)

**throws** IOException, InterruptedException {

**int** sum = 0;

**int** l=0;

**for** (IntWritable val : values) {

l+=1;

sum += val.get();

}

sum=sum/l;

context.write(key, **new** IntWritable(sum));

}

}

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

Configuration conf = **new** Configuration();

@SuppressWarnings("deprecation")

Job job = **new** ~~Job~~(conf, "Averageage\_survived");

job.setJarByClass(Average\_age.**class**);

job.setMapOutputKeyClass(Text.**class**);

job.setMapOutputValueClass(IntWritable.**class**);

// job.setNumReduceTasks(0);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(IntWritable.**class**);

job.setMapperClass(Map.**class**);

job.setReducerClass(Reduce.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

Path out=**new** Path(args[1]);

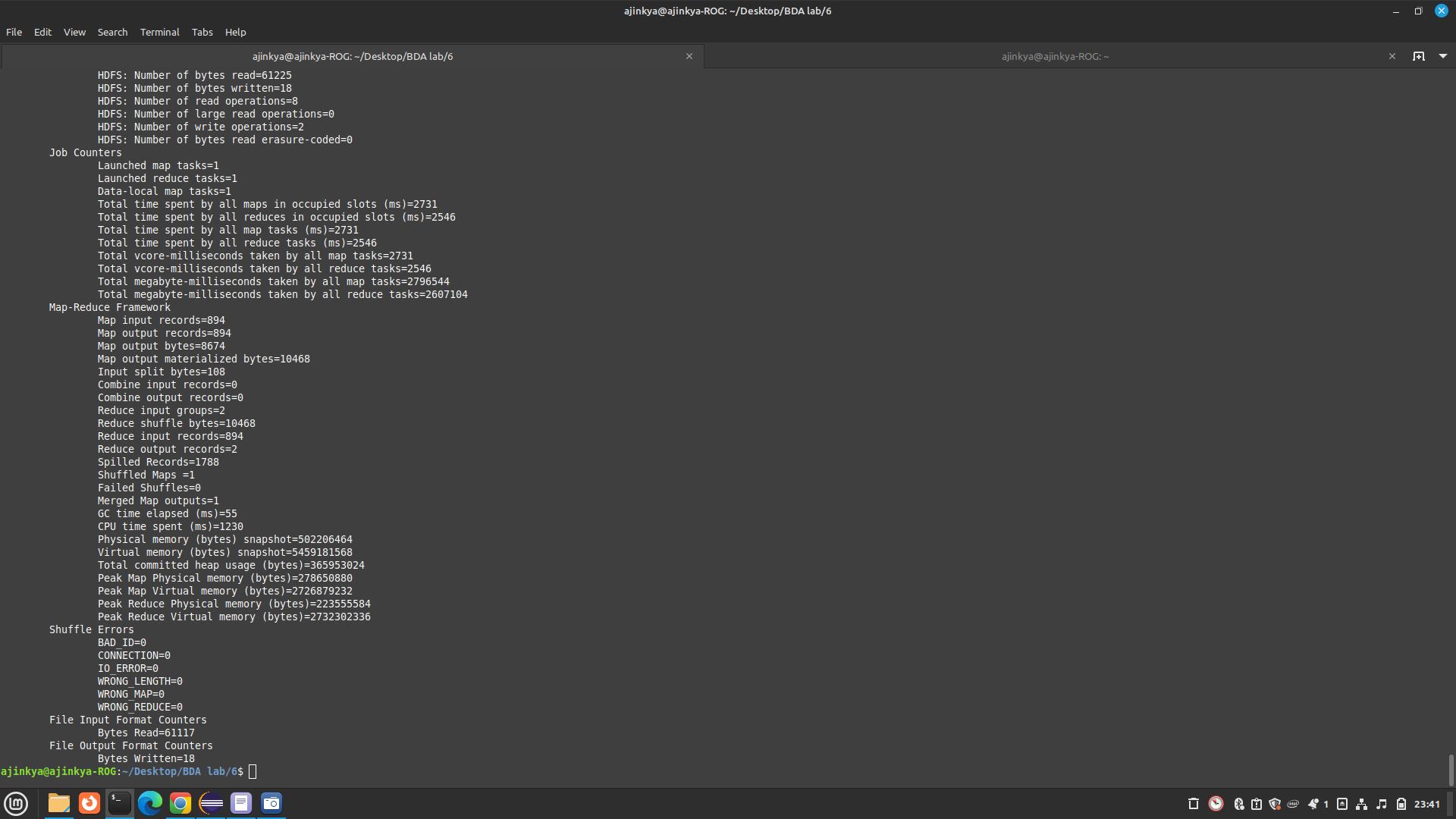
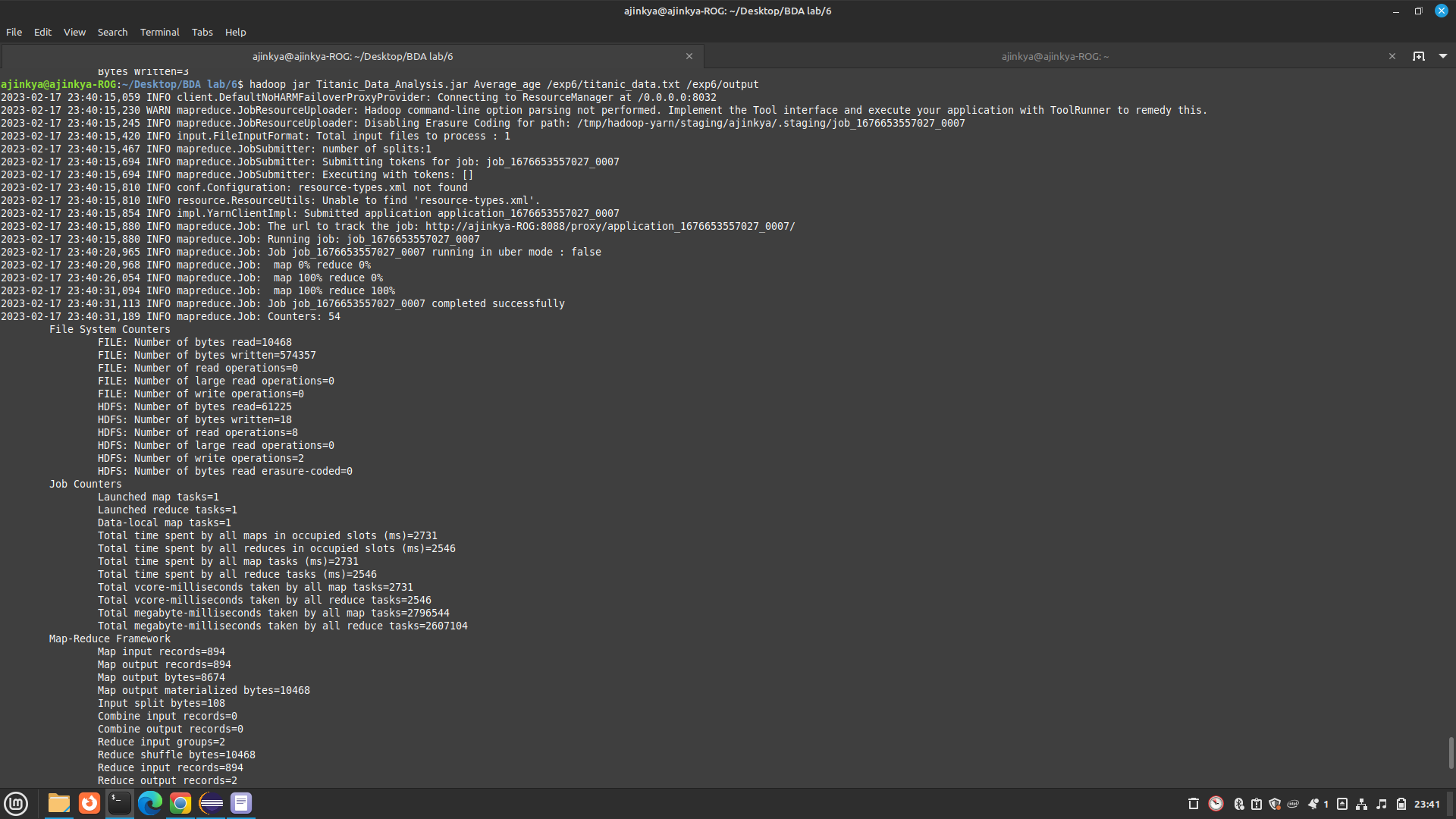
out.getFileSystem(conf).delete(out,**true**);

job.waitForCompletion(**true**);

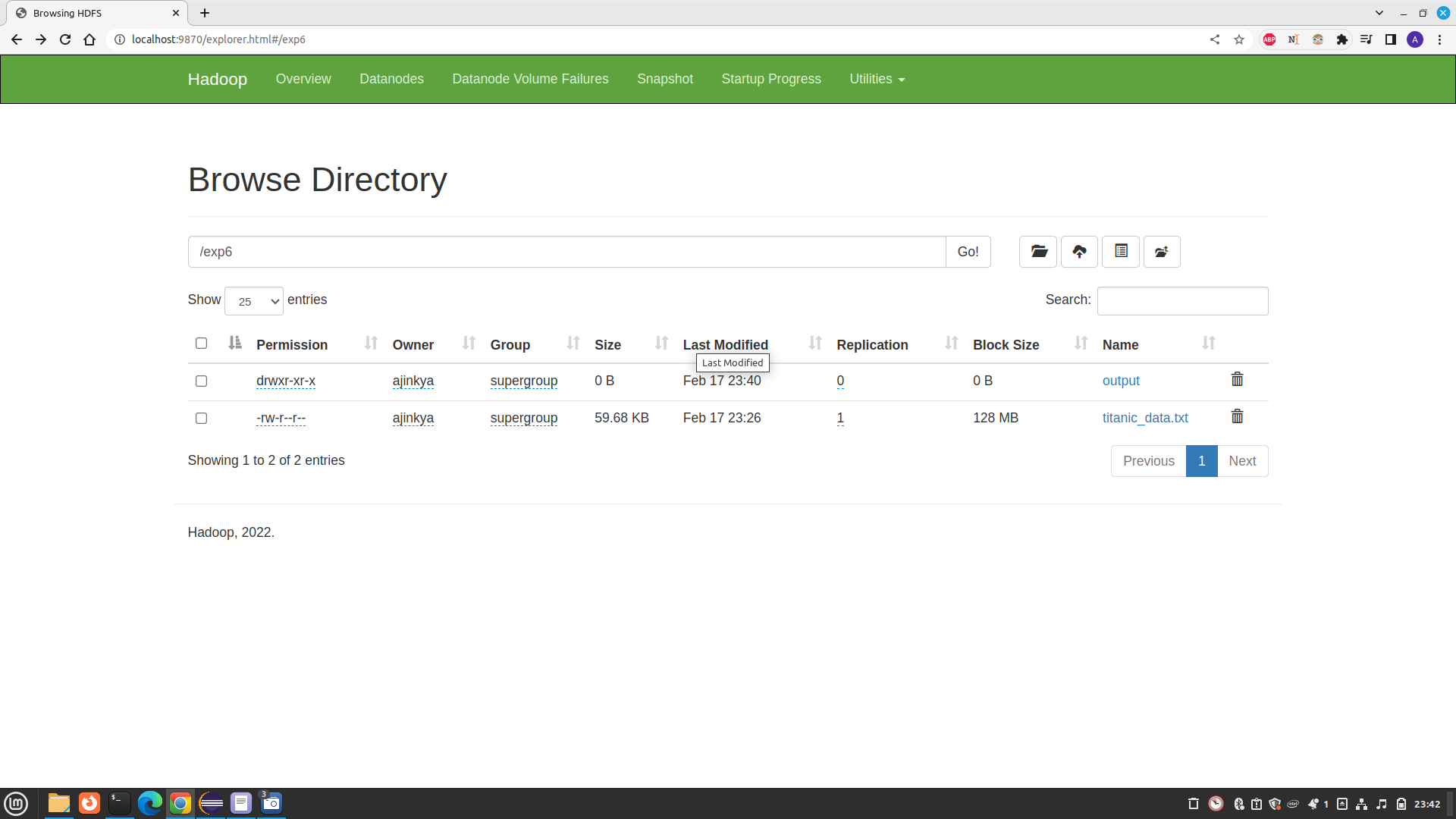
}

}

Running the code

Output



Directory structure