**UNIVERSITY SCHOOL OF AUTOMATION & ROBOTICS (USAR)**

**Guru Gobind Singh Indraprastha University, East Delhi Campus, Surajmal Vihar, Delhi 110092**

**Big Data Analytics Lab**

(ARD-353)

**Submitted to: Submitted By:**

**Dr. Sanjay Kumar Singh Name: Pratham Kumar**

**Asst. Professor, USAR Batch: AI-DS (B1-A) Roll No: 00119011921**

**Index**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Program** | **Signature** |
| 1 | 1.1) Installation of Ubuntu using WSL/Virtual box/Dual Boot etc.  1.2) Installation of hadoop latest version in Ubuntu stand-alone mode. |  |
| 2 | Setting up a Single Node Cluster. Hadoop Installation: Psuedo Distributed Mode( Locally and YARN) |  |
| 3 | File Management tasks in Hadoop. Creation of folder, deletion of folder, put a file from local drive to hdfs, show the content of hdfs file, download the hdfs file to local drive, delete hdfs file. |  |
| 4 | Word count Map Reduce program using Java. |  |
| 5 | Word count Map Reduce program using python. |  |
| 6 | 6.1) Installing apache spark/pyspark.  6.2) WAP to create pyspark configuration, session and count number of words present in a given textfile. |  |
| 7 | WAP to implement various regression models and evaluate their performance using pyspark. |  |
| 8 | WAP to implement various classification models and evaluate their performance using pyspark. |  |
| 9 | WAP to implement various K-means clustering and find the best value of K using the elbow method using pyspark. |  |
| 10 | WAP to implement a recommender system using pyspark. |  |

# **Practical 1.1: Installation of Ubuntu using WSL.**

## **Aim:** To install Ubuntu using Windows Subsystem for Linux (WSL). Objectives:

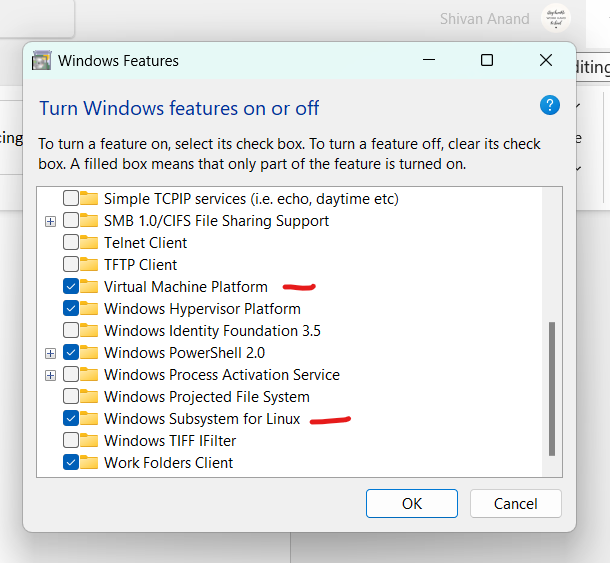
1. To understand the process of setting up WSL.
2. To install Ubuntu distribution within WSL.

## Requirements:

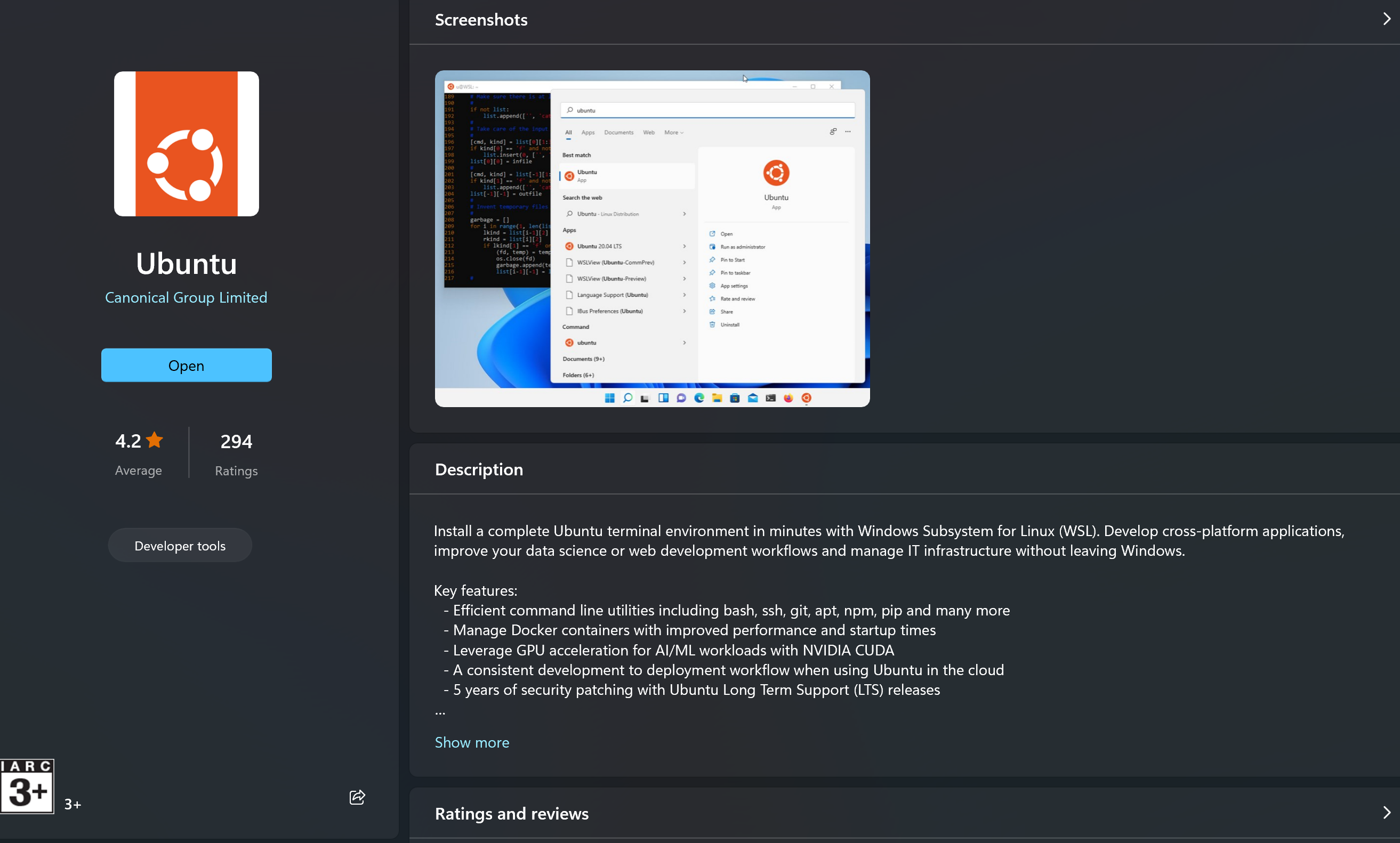
1. Windows 11 operating system or later.
2. Stable internet connection.
3. Access to the Microsoft Store.

## Procedure:

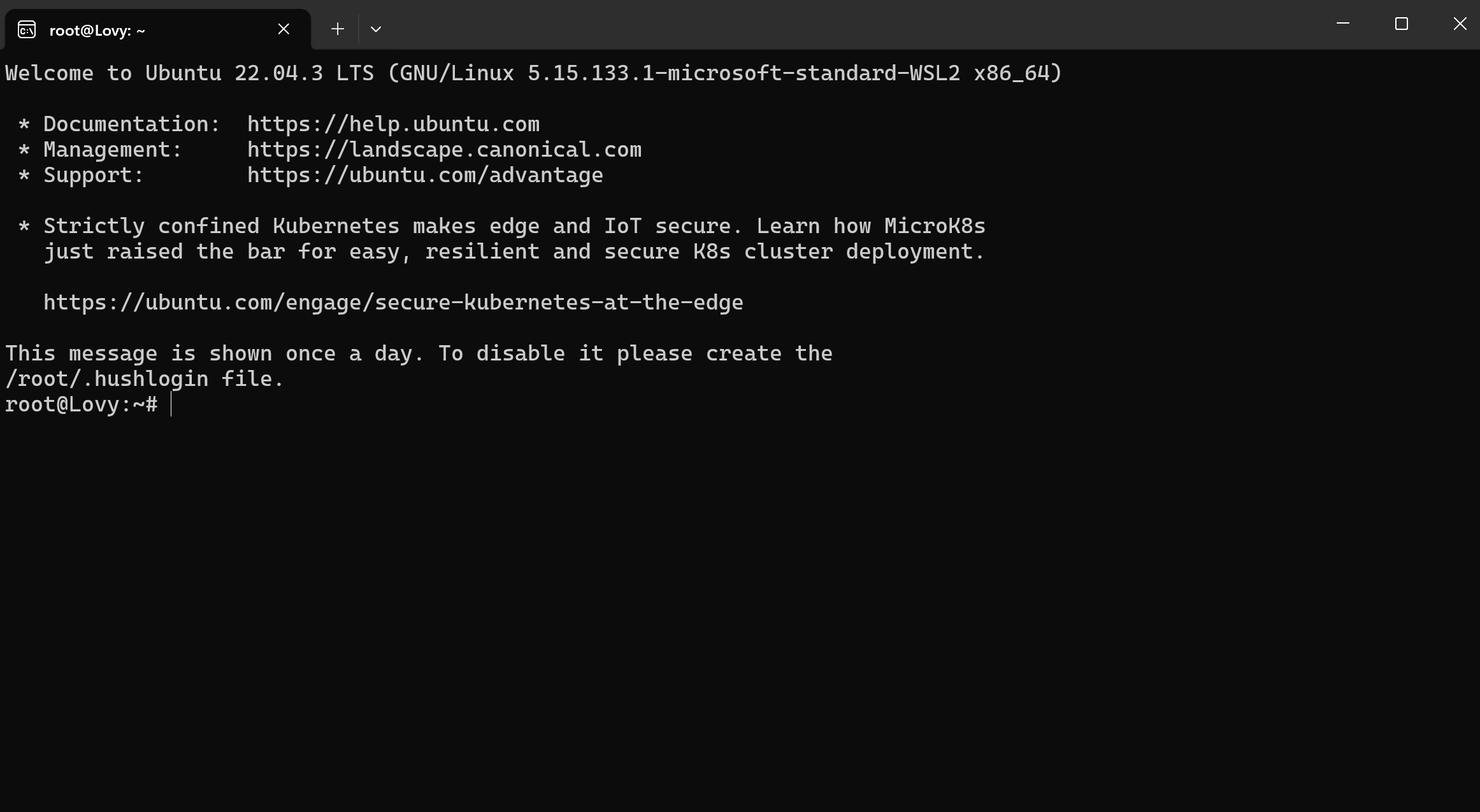
1. Open “Turn Windows features on or off” and check the checkbox for “Virtual Machine Platform” and “Windows Subsystem for Linux”.



1. Install Ubuntu 22.04.2 LTS from Microsoft Office.



1. Launch Ubuntu 22.04.2 LTS. Enter a username. This will create a local user account and you will be automatically logged in to Ubuntu 18.04 as this user. Enter a password for the user and enter a second time to confirm.



## Observations

1. The download and installation process may take some time depending on the speed of the internet connection.
2. During the installation, the progress will be displayed in the Microsoft Store.

## Results

1. Ubuntu has been successfully installed using WSL.
2. The Ubuntu terminal is accessible from the Start menu and can be used for various Linux operations.

## Conclusion

The successful installation of Ubuntu using WSL allows users to leverage the advantages of both Windows and Ubuntu operating systems, enabling a more versatile and comprehensive computing environment.

Practical 1.2: Installation of Hadoop in Ubuntu Stand-Alone Mode. Aim: To install the latest version of Hadoop in stand-alone mode on the Ubuntu operating system.

## Objectives:

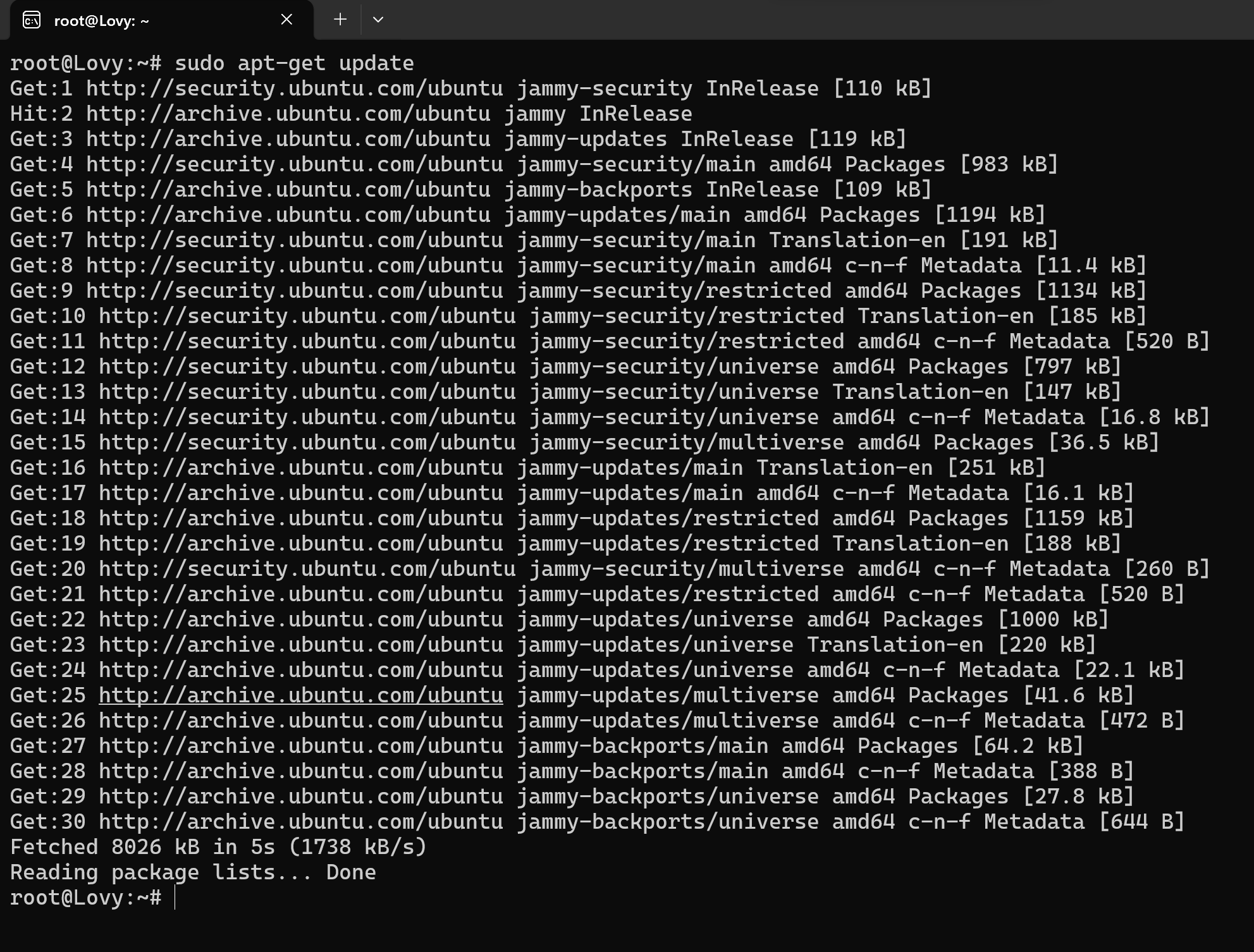
1. Setting up the environment for Hadoop installation.
2. Downloading the latest version of Hadoop.
3. Configuring the Hadoop environment variables.
4. Verifying the successful installation of Hadoop.

## Requirements:

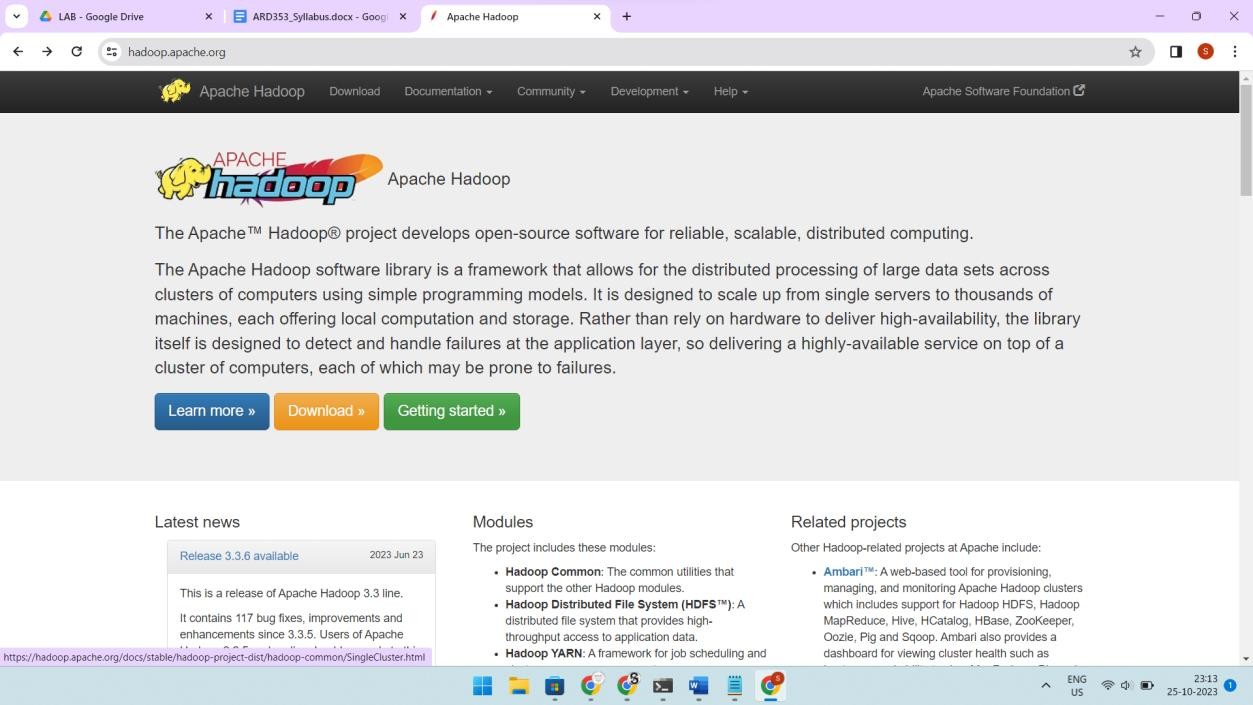
1. Ubuntu operating system installed using WSL or any other method.
2. Stable internet connection.
3. Basic understanding of the Ubuntu command line.

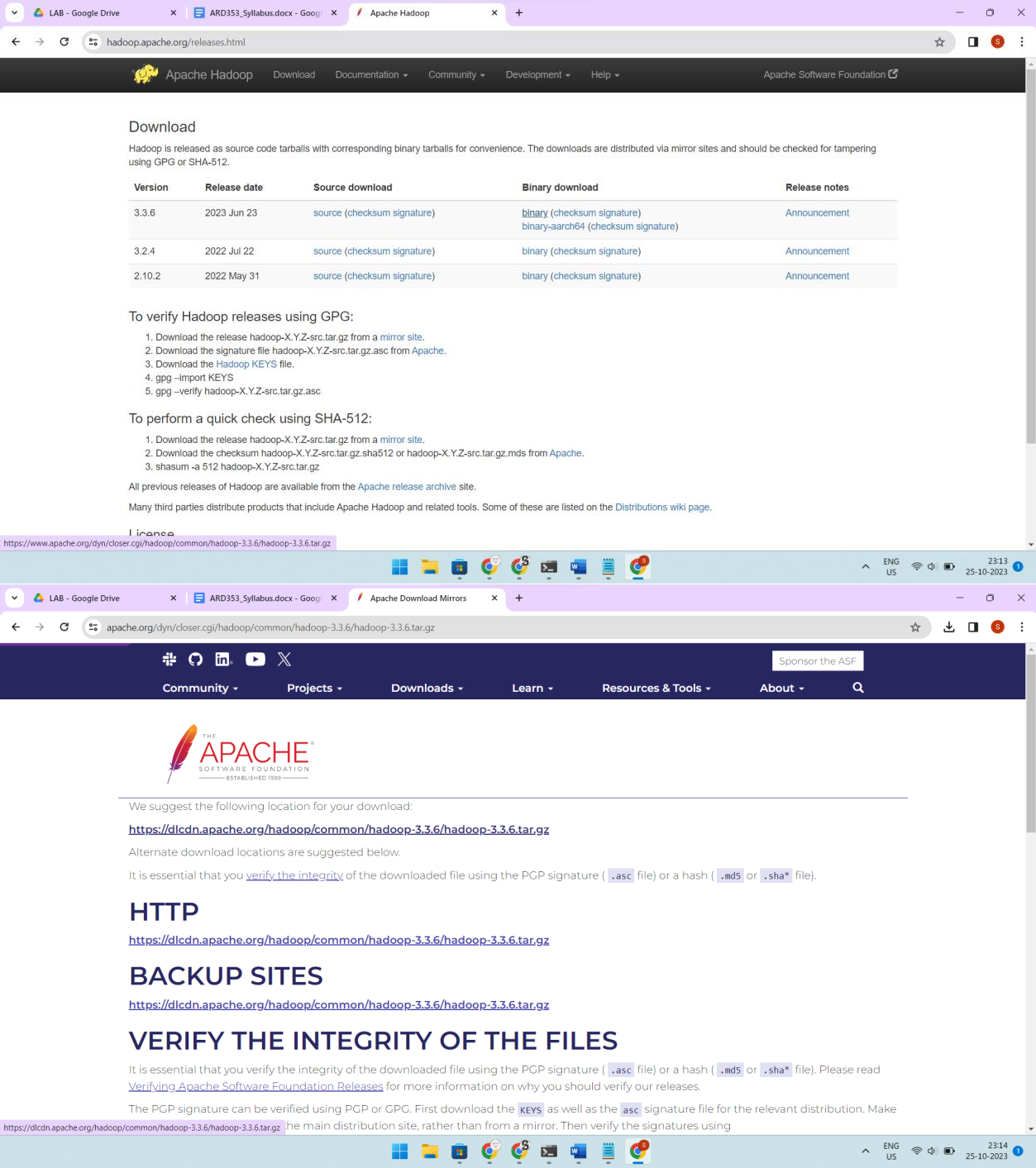
## Procedure:

1. Open the Ubuntu terminal.
2. Update the system packages using the command: `*sudo apt-get update*`.

­­­­­­­­­­­­­

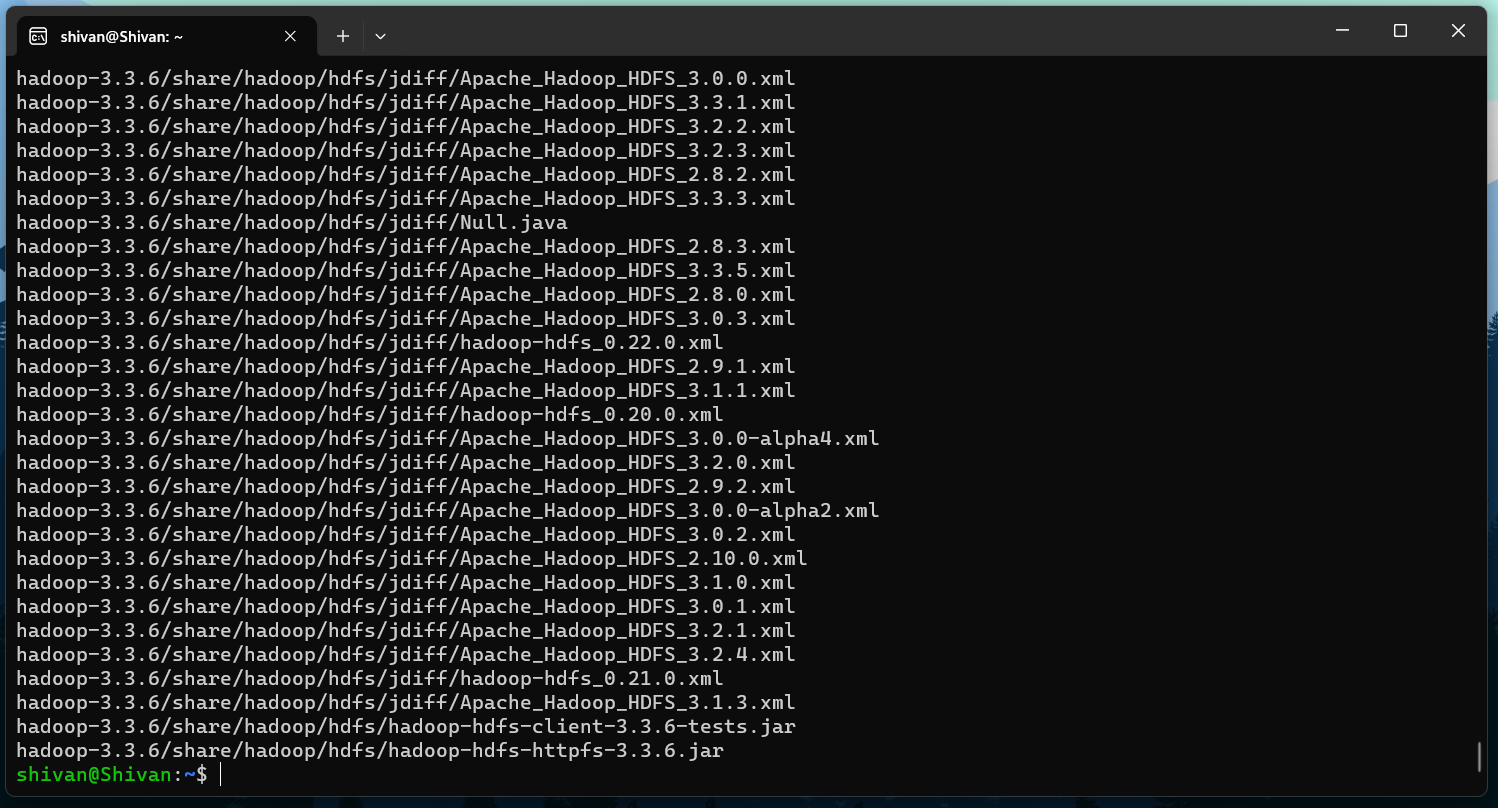
1. Go to <https://hadoop.apache.org/>, Click on “Getting Started”, then download the latest Binary download, in our case the latest Hadoop version is 3.3.6.



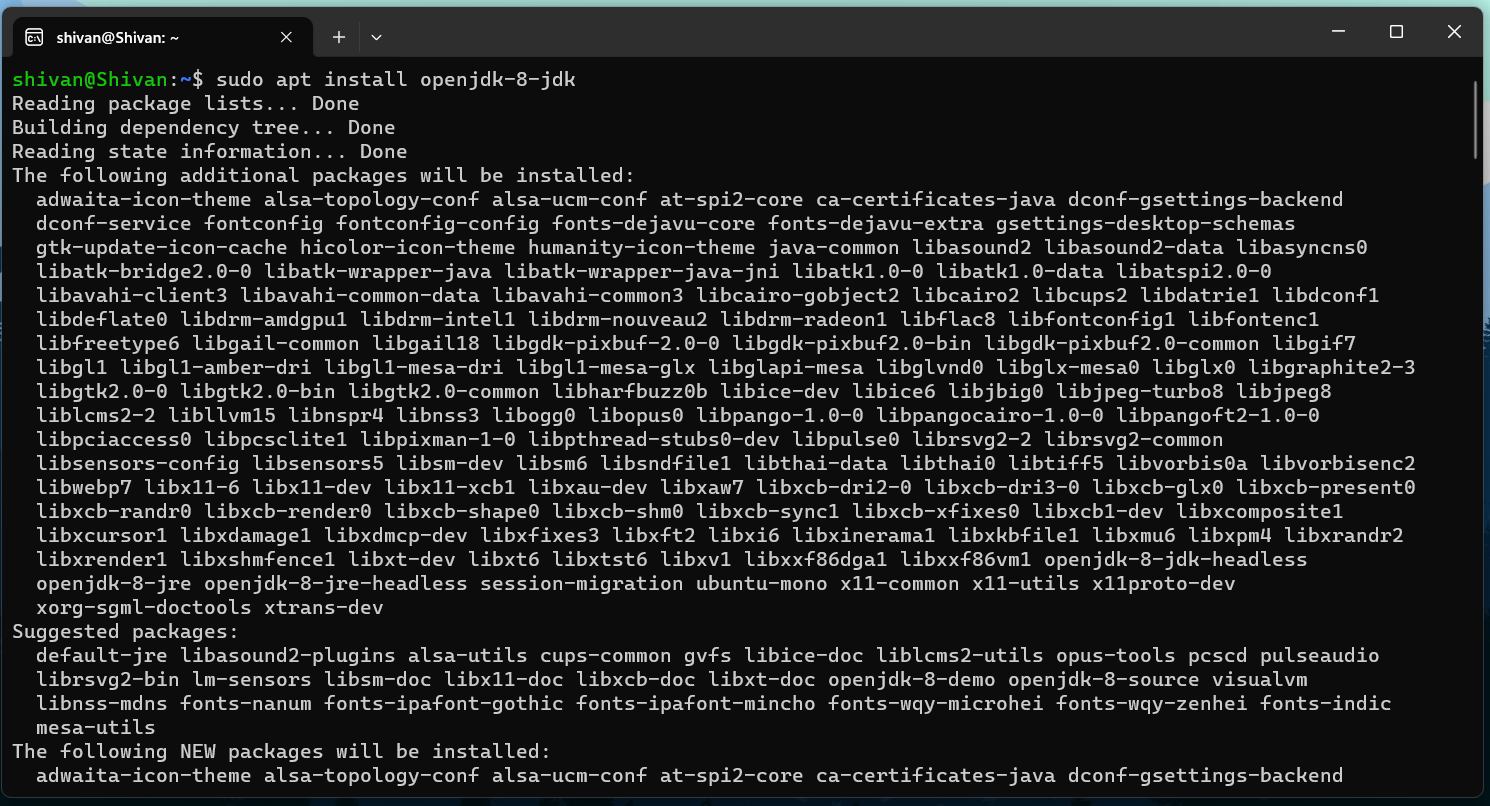


1. Extract the downloaded Hadoop package using the command: *` tar -xzvf*

*/mnt/c/Users/prath/Downloads/hadoop-3.3.6.tar.gz -C /home/prath/ `*.



1. Install Java JDK using the command: `*sudo apt install openjdk-8 -jdk*`.



1. Open .bashrc file using the command “*sudo nano .bashrc*”



and paste these commands:

*export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64 export PATH=$PATH:/usr/lib/jvm/java-8-openjdk-amd64/bin*

*export HADOOP\_HOME=~/hadoop-3.3.6/ export PATH=$PATH:$HADOOP\_HOME/bin export PATH=$PATH:$HADOOP\_HOME/sbin*

*export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME*

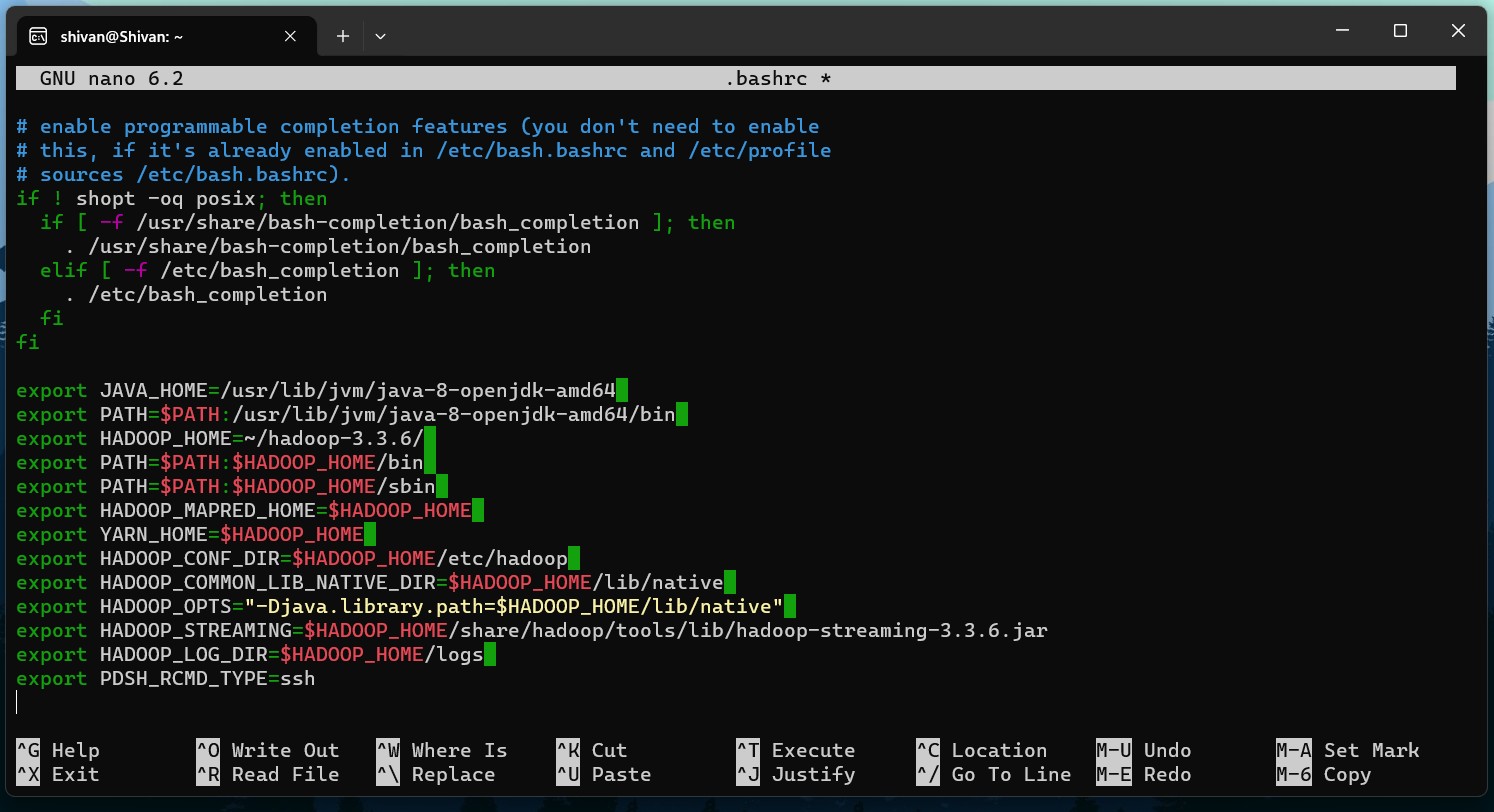
*export YARN\_HOME=$HADOOP\_HOME*

*export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop*

*export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib/native"*

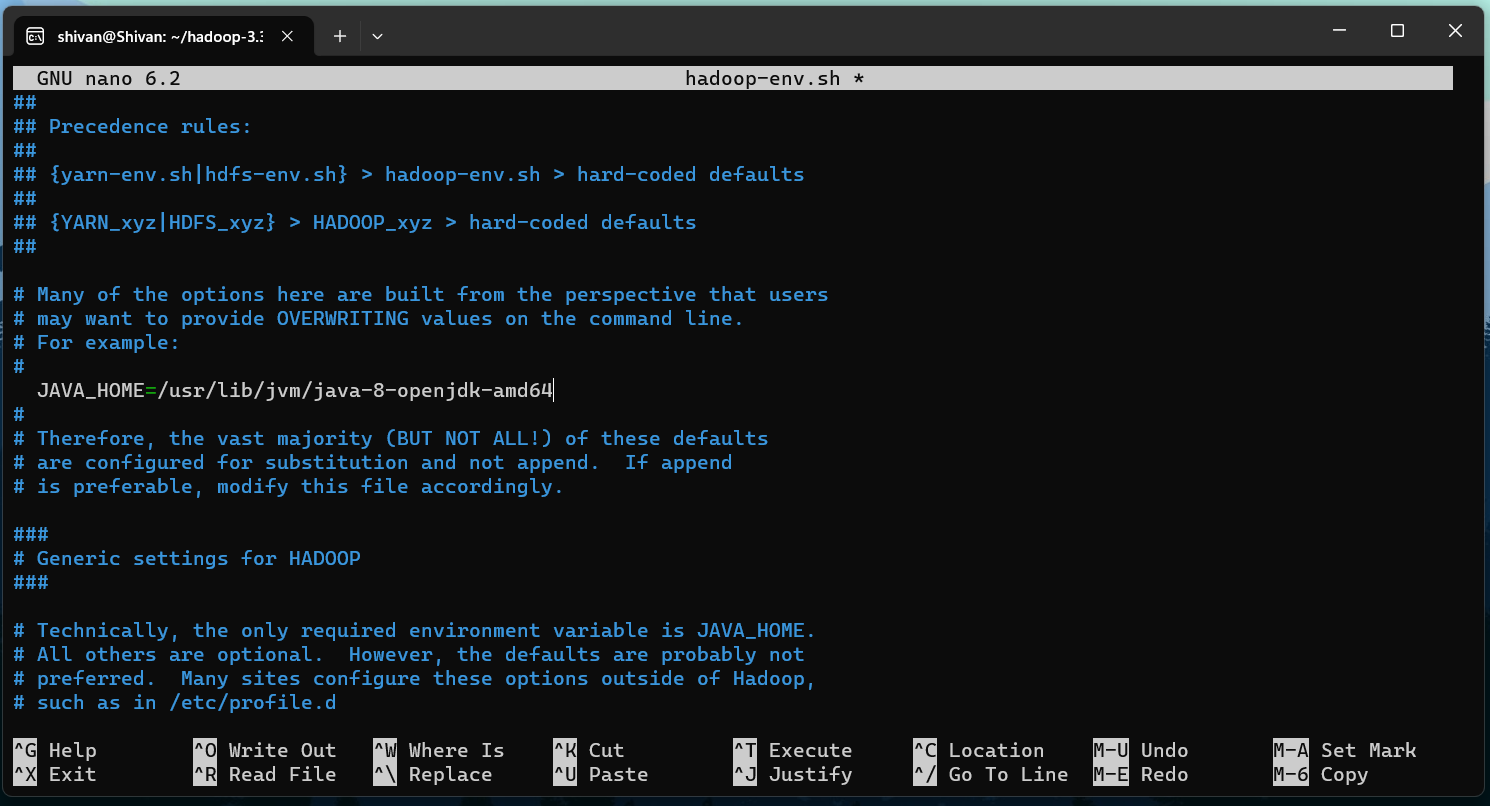
*export HADOOP\_STREAMING=$HADOOP\_HOME/share/hadoop/tools/lib/hadoop- streaming-3.3.6.jar*

*export HADOOP\_LOG\_DIR=$HADOOP\_HOME/logs export PDSH\_RCMD\_TYPE=ssh*



Press ctrl + O to write out and then ctrl + x to save the file

1. Go to hadoop-3.3.6/etc/hadoop
2. Set the JAVA\_HOME on Hadoop Environment by opening the file hadoop-env.sh using the command “*sudo nano hadoop-env.sh*” add the path of JAVA\_HOME and save the file as follows



## Observations:

1. The installation and configuration processes may take some time depending on the system specifications and internet speed.
2. Any errors or warnings during the installation process should be carefully noted for troubleshooting.

## Results:

Hadoop has been successfully installed in stand-alone mode on the Ubuntu operating system.

## Conclusion:

The successful installation of the latest version of Hadoop in stand-alone mode on Ubuntu provides users with a platform to explore the core functionalities of Hadoop and gain insights into distributed computing and data processing.

# **Practical 2: Setting up a Single Node Cluster and understanding its functionalities.**

## **Aim:** Setting up a Single Node Cluster. Hadoop Installation: Psuedo Distributed Mode( Locally and YARN)

Objectives:

1. Configuring Hadoop in Pseudo-Distributed Mode.
2. Understanding the setup of a Single Node Cluster.
3. Exploring the functionalities of the Hadoop ecosystem in a pseudo-distributed environment.

## Requirements:

1. Ubuntu operating system with Hadoop installed.
2. Basic understanding of Hadoop configuration files and settings.

## Procedure:

1. Go to hadoop-3.3.6/etc/hadoop
2. Open the file core-site.xml using the command “*sudo nano core-site.xml*”
3. Edit the `core-site.xml` file to define the Hadoop core parameters, which are:

*<configuration>*

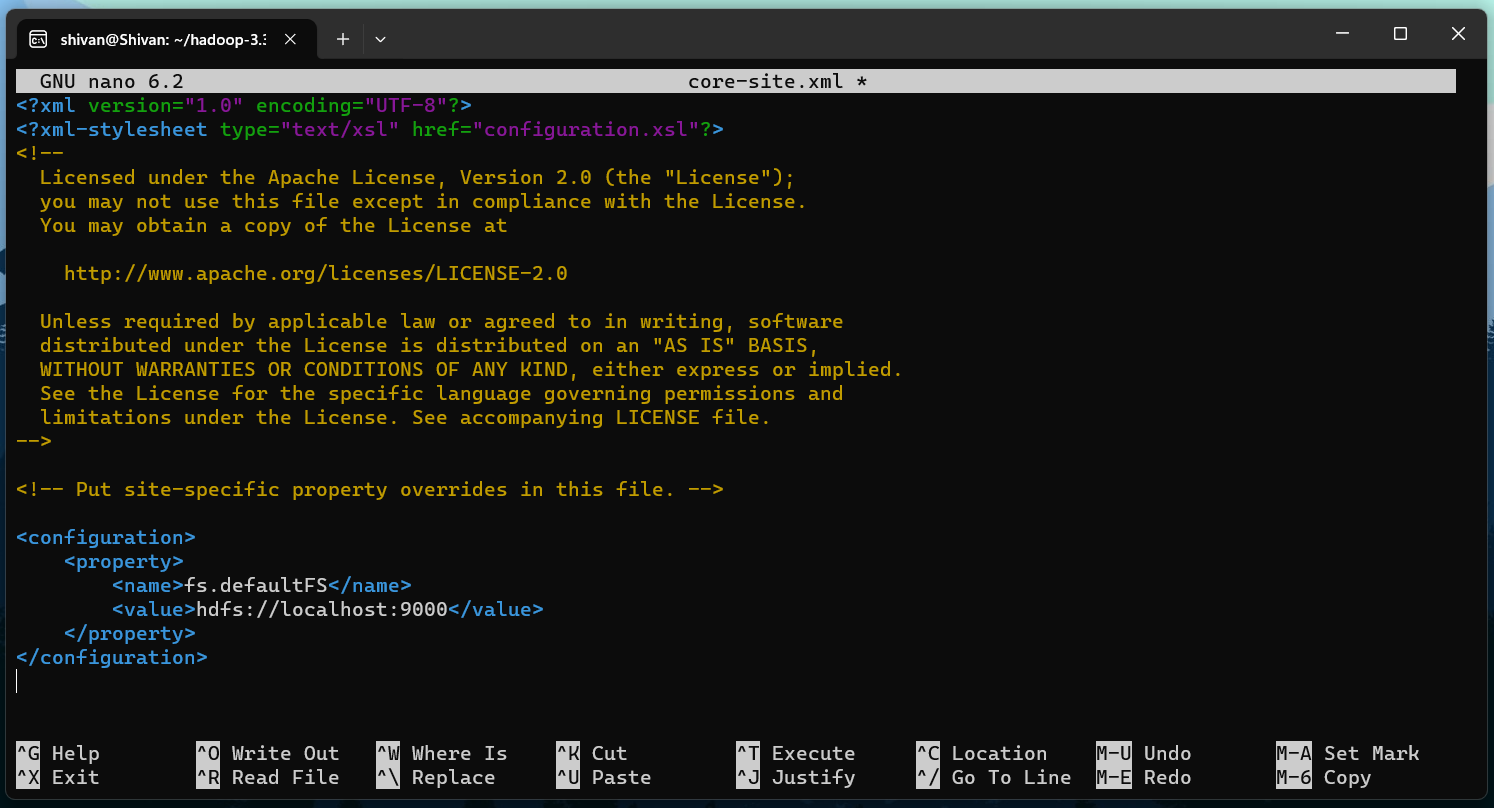
*<property>*

*<name>fs.defaultFS</name>*

*<value>hdfs://localhost:9000</value>*

*</property>*

*</configuration>*



Save the file.

1. Open the file hdfs-site.xml using the command “*sudo nano hdfs-site.xml*”
2. Modify the `hdfs-site.xml` file to configure HDFS settings, which are:

*<configuration>*

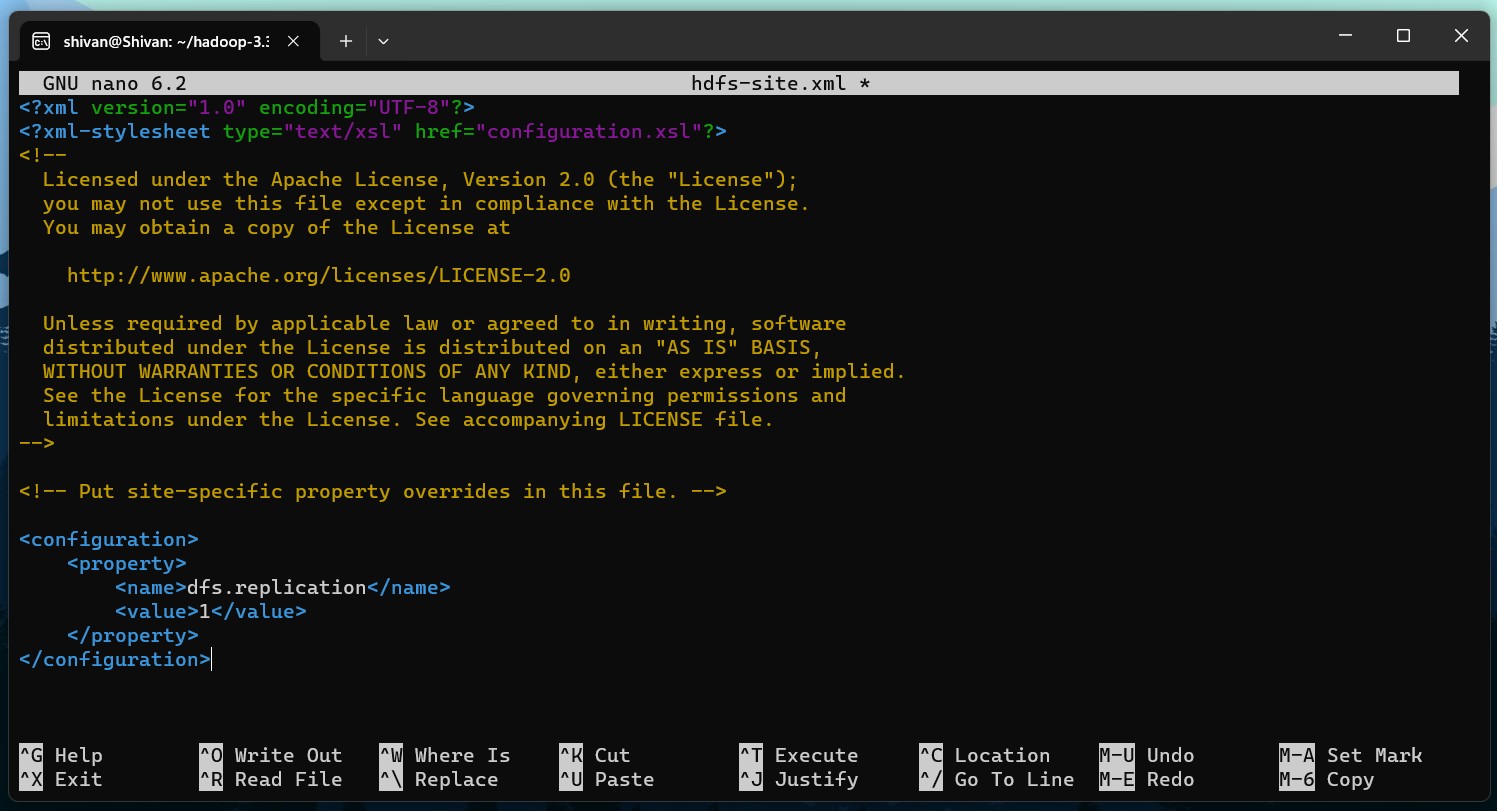
*<property>*

*<name>dfs.replication</name>*

*<value>1</value>*

*</property>*

*</configuration>*



Save the file.

1. Open the file mapred-site.xml using the command “*sudo nano mapred-site.xml*”
2. Configure the parameters of mapred-site.xml as follows:

*<configuration>*

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*

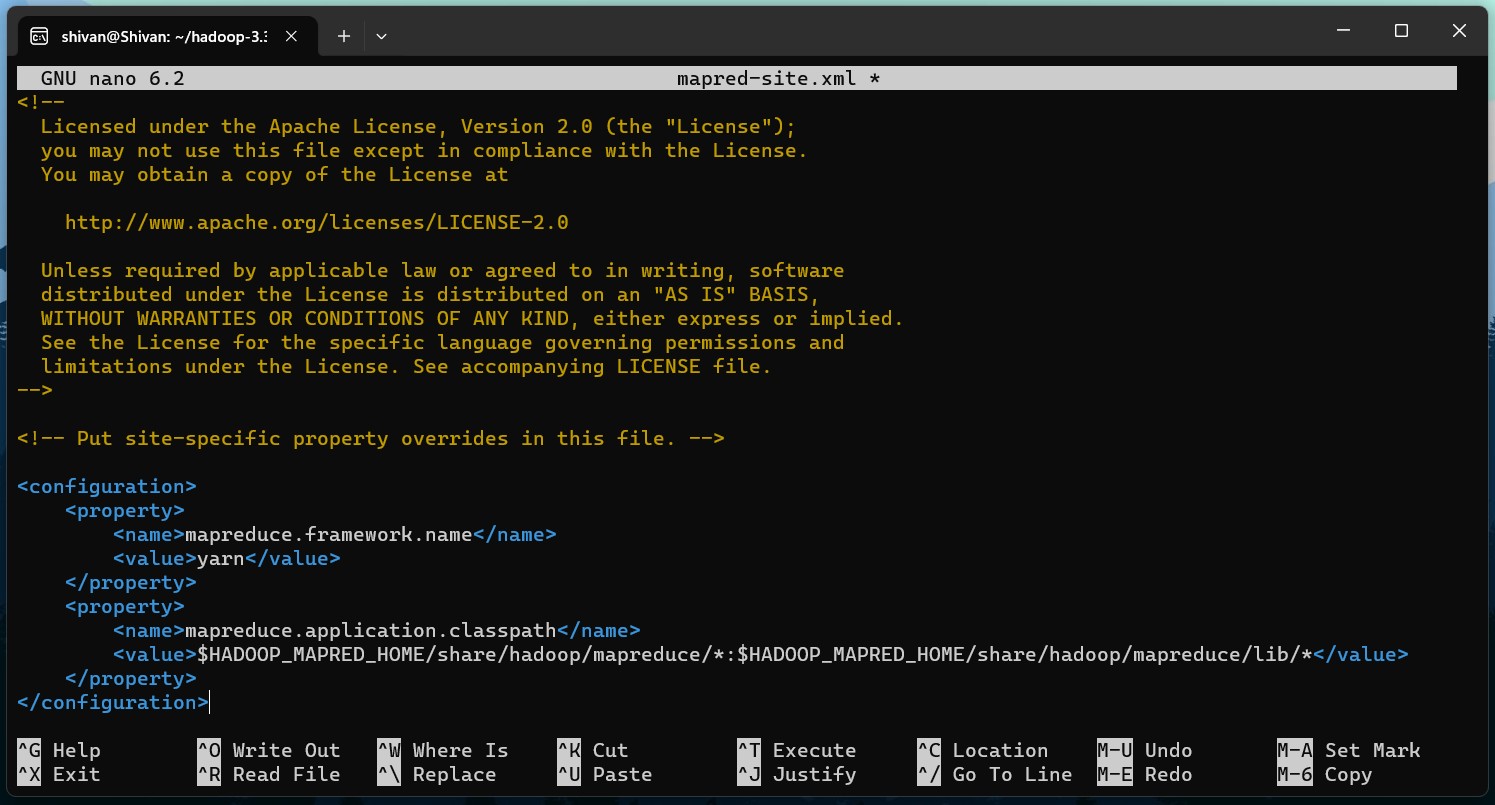
*<property>*

*<name>mapreduce.application.classpath</name>*

*<value>$HADOOP\_MAPRED\_HOME/share/hadoop/mapreduce/\*:$HADOOP\_MAPRE D\_HOME/share/hadoop/mapreduce/lib/\*</value>*

*</property>*

*</configuration>*



Save the file.

1. Open the file yarn-site.xml using the command “*sudo nano yarn-site.xml*”
2. Configure the `yarn-site.xml` file to set up the YARN settings. Configurations are:

*<configuration>*

*<property>*

*<name>yarn.nodemanager.aux-services</name>*

*<value>mapreduce\_shuffle</value>*

*</property>*

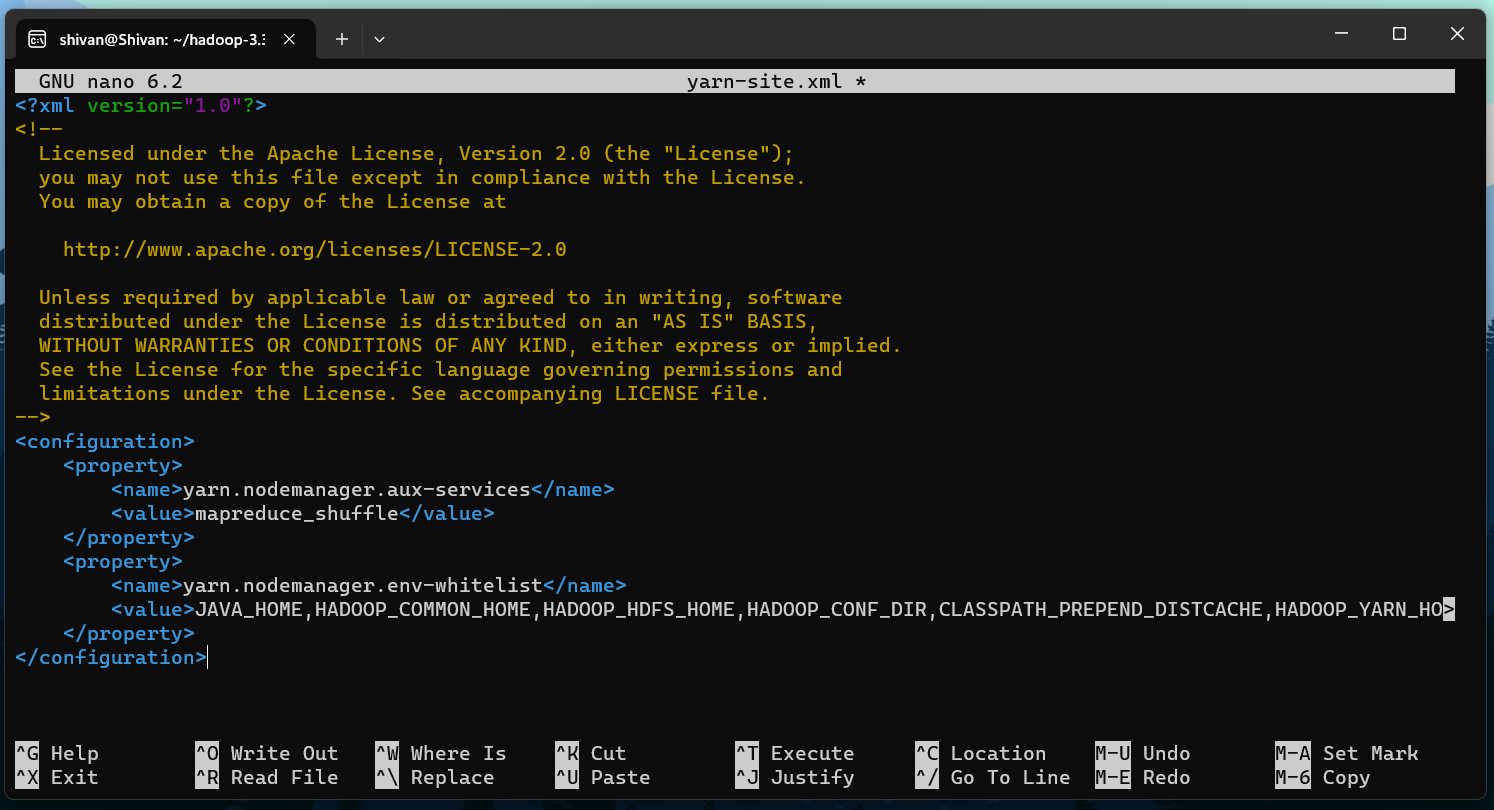
*<property>*

*<name>yarn.nodemanager.env-whitelist</name>*

*<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_ CONF\_DIR,CLASSPATH\_PREPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_HO ME,PATH,LANG,TZ,HADOOP\_MAPRED\_HOME</value>*

*</property>*

*</configuration>*

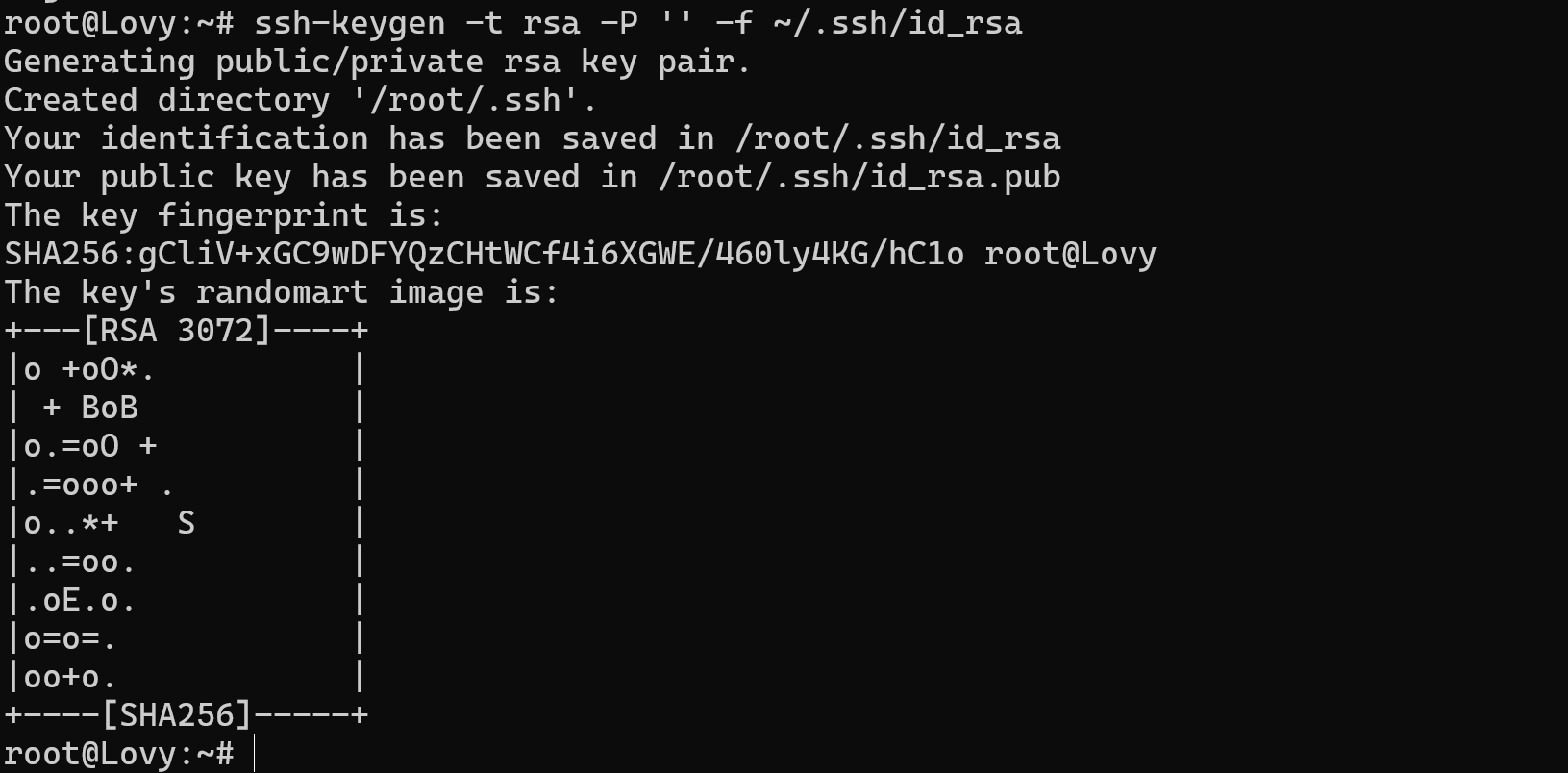


Save the file.

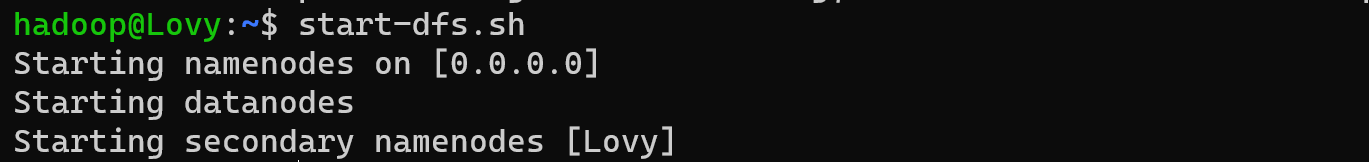
1. Now check that you can ssh to the localhost without a passphrase, using the command “*ssh localhost*”
2. Execute the following commands:

*ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa*

*cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys chmod 0600 ~/.ssh/authorized\_keys*

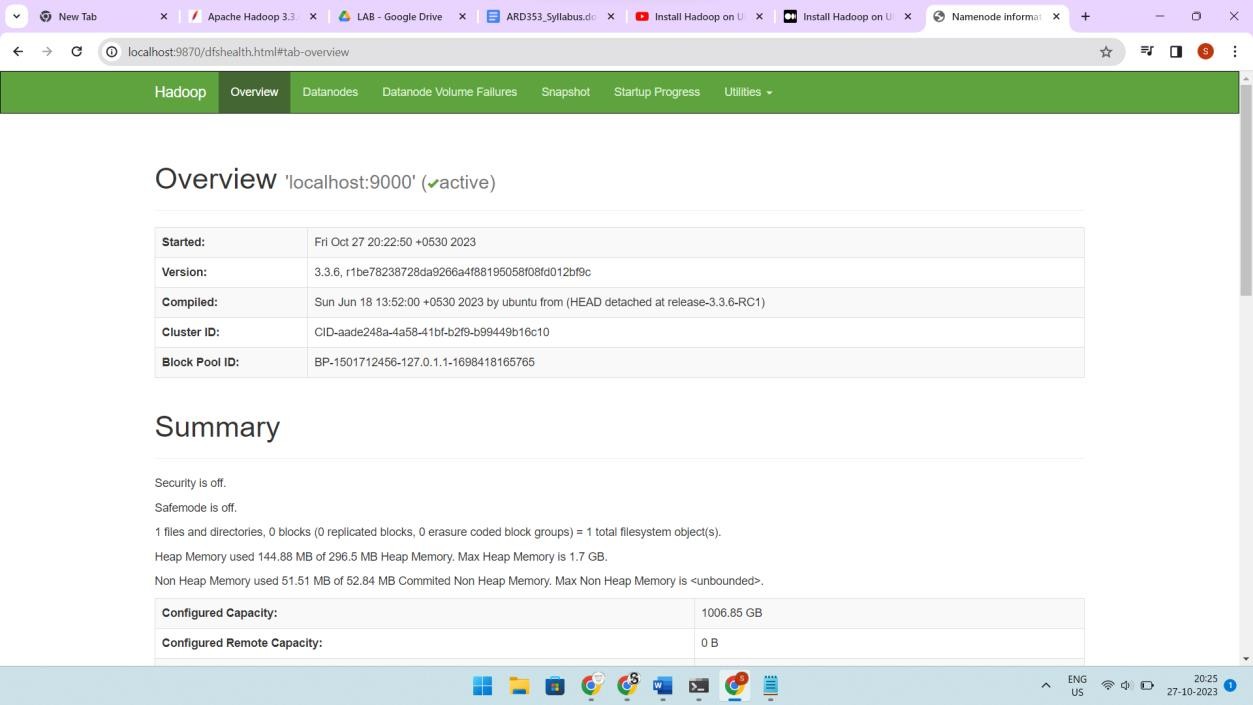


1. Format the filesystem using the command “*hdfs namenode -format*”
2. Start NameNode daemon and DataNode daemon using the command “*start-dfs.sh*”



1. Browse the web interface for the NameNode; by default it is available at: NameNode -

*http://localhost:9870/*

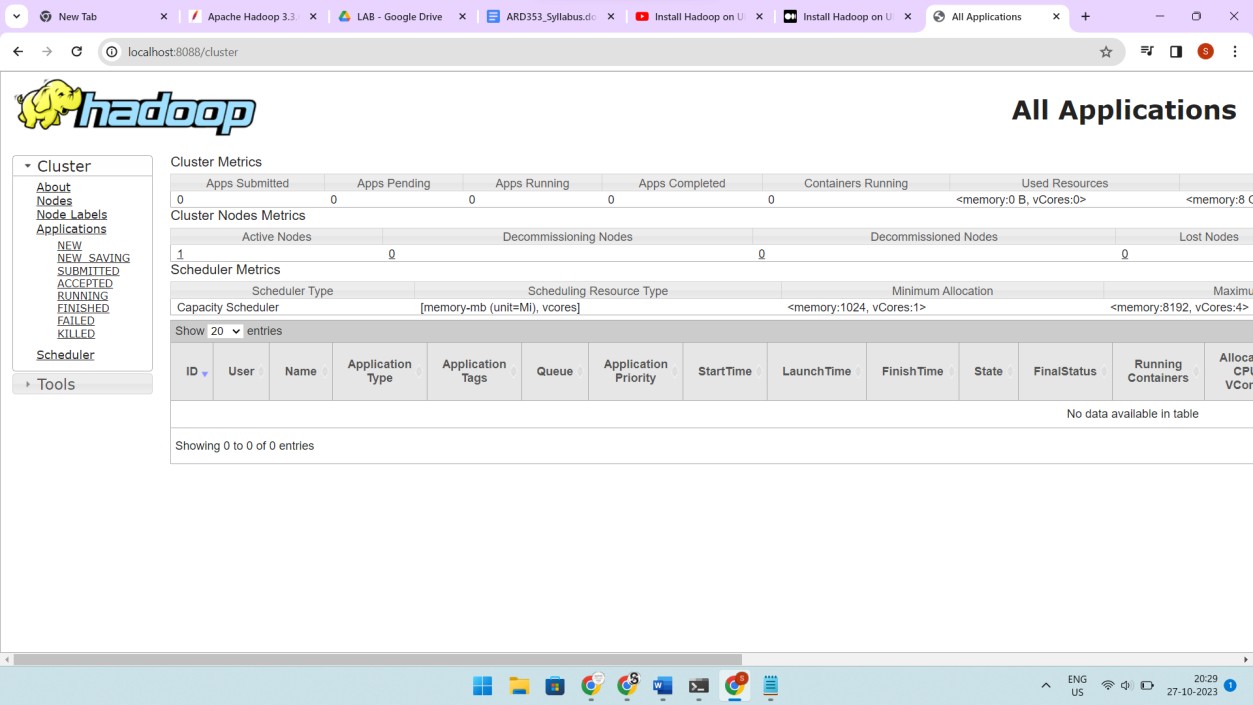


1. Start ResourceManager daemon and NodeManager daemon using the command “*start- yarn.sh*”



1. Browse the web interface for the ResourceManager; by default it is available at:

*http://localhost:8088/*



## Observations:

Pay attention to the log messages and error reports during the configuration and startup process.

## Results:

The Single Node Cluster has been successfully set up in Pseudo-Distributed Mode.

## Conclusion:

The successful setup of the Single Node Cluster in Pseudo-Distributed Mode facilitates the understanding of Hadoop's distributed architecture and operations, serving as a foundation for further exploration and learning in the field of Big Data processing and analytics.

# **Practical 3: Implementing file management tasks in Hadoop.**

## **Aim:** File Management tasks in Hadoop. Creation of folder, deletion of folder, put a file from local drive to hdfs, show the content of hdfs file, download the hdfs file to local drive, delete hdfs file.

Objectives:

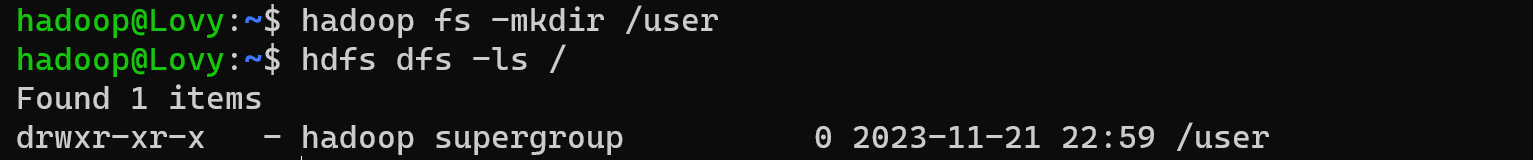
1. Creating folders in HDFS.
2. Deleting folders from HDFS.
3. Uploading files from the local drive to HDFS.
4. Displaying the contents of an HDFS file.
5. Downloading an HDFS file to the local drive.
6. Deleting an HDFS file.

## Requirements:

1. Hadoop installed and configured in Pseudo-Distributed or Standalone mode.
2. A sample file for uploading to HDFS.

## Procedure:

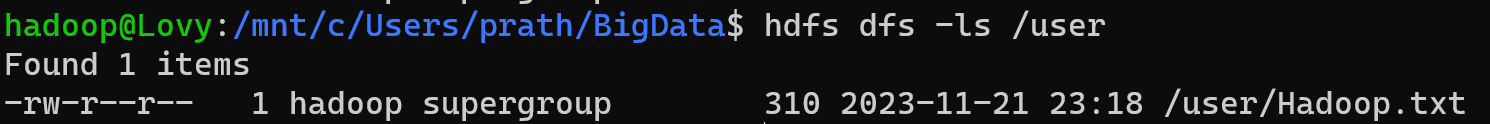
1. Use the “*hadoop fs -mkdir*” command to create a folder in HDFS.

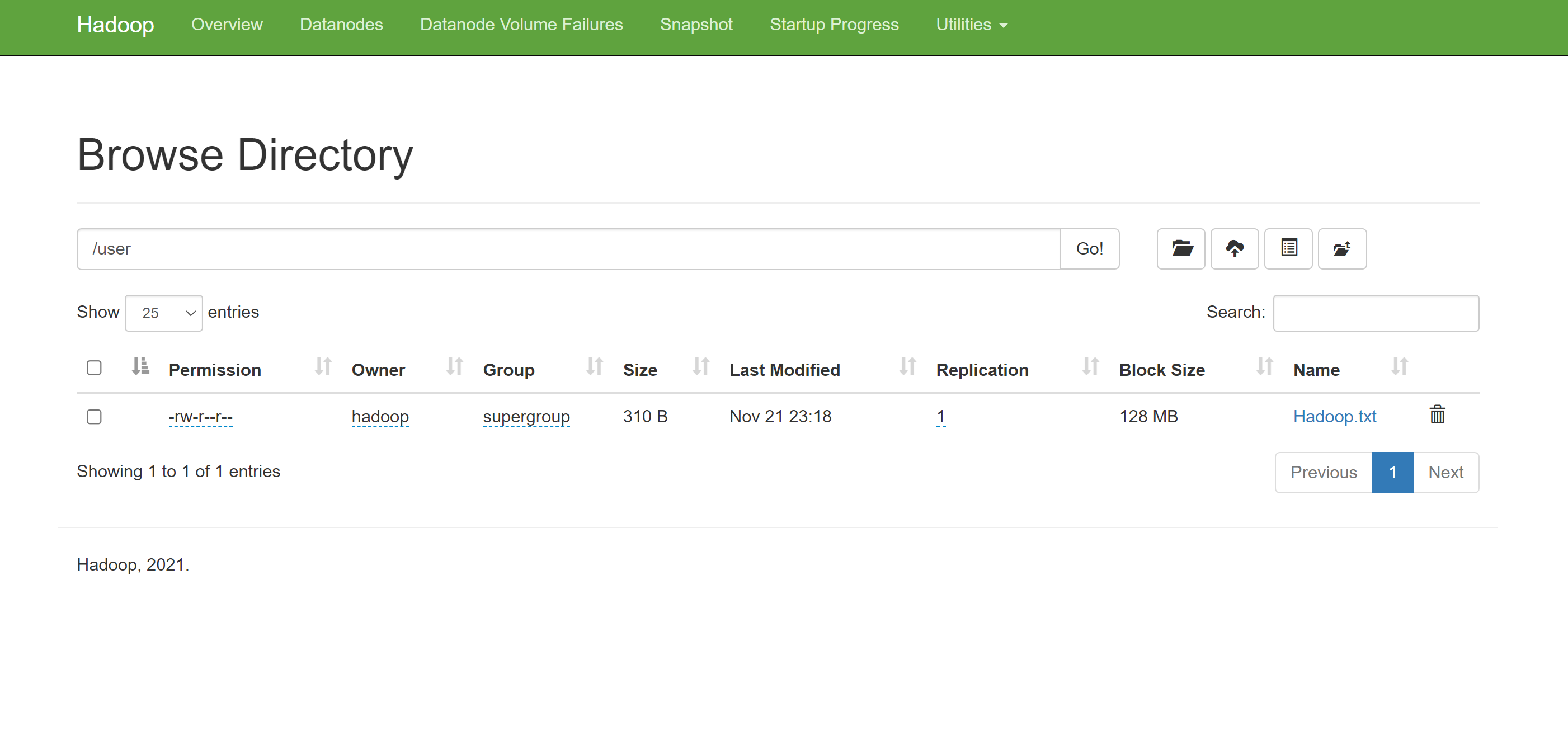


1. Upload a file from the local drive to HDFS using the “*hdfs dfs -put*” command.

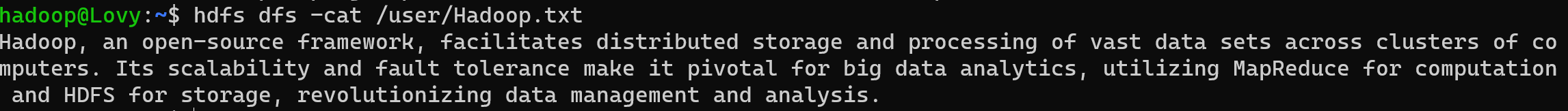








1. Display the contents of an HDFS file using the “*hdfs dfs -cat*” command.



1. Download an HDFS file to the local drive using the “*hdfs dfs -get*“ command.



1. Delete an HDFS file using the “*hdfs dfs -rm*” command.



1. Use the “*hdfs dfs -rm -r*” command to delete a folder from HDFS.



## Observations:

1. Note any errors or warnings during the file management tasks.
2. Monitor the file system changes in the Hadoop user interface.

## Results:

1. Folders have been created and deleted successfully in HDFS.
2. Files have been uploaded, downloaded, and deleted from HDFS.

## Conclusion:

The successful execution of file management tasks in Hadoop demonstrates the capabilities of HDFS in handling data storage, retrieval, and deletion, showcasing its role in managing Big Data effectively.

# **Practical 4: Writing a Word count MapReduce program in Java.**

## **Aim:** To implement a Word Count MapReduce program using Java.

## Objectives:

1. Understanding the basics of MapReduce programming paradigm.
2. Writing Mapper and Reducer functions for word counting.
3. Compiling and executing the Java program on the Hadoop cluster.

## Requirements:

1. Hadoop installed and configured in Pseudo-Distributed or Standalone mode.
2. Basic understanding of Java programming.
3. Text input files for testing the Word Count program.

## Procedure:

1. Write a Java file named WordCount.java containing the provided source 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);*

*}*

**

1. Set the JAVA\_HOME variable, using the command “*export JAVA\_HOME=/usr/lib/jvm/java-8- openjdk-amd64*”

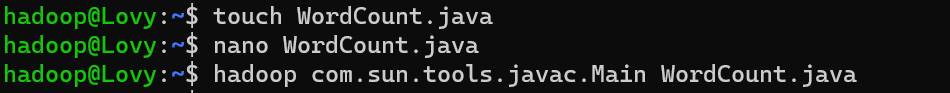


1. Set the HADOOP\_CLASSPATH variable, using the command “*export HADOOP\_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar*”

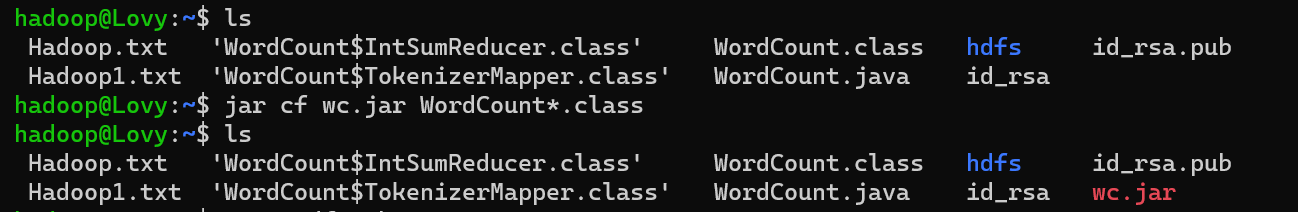


1. Compile the WordCount.java file and create a JAR file using the provided commands

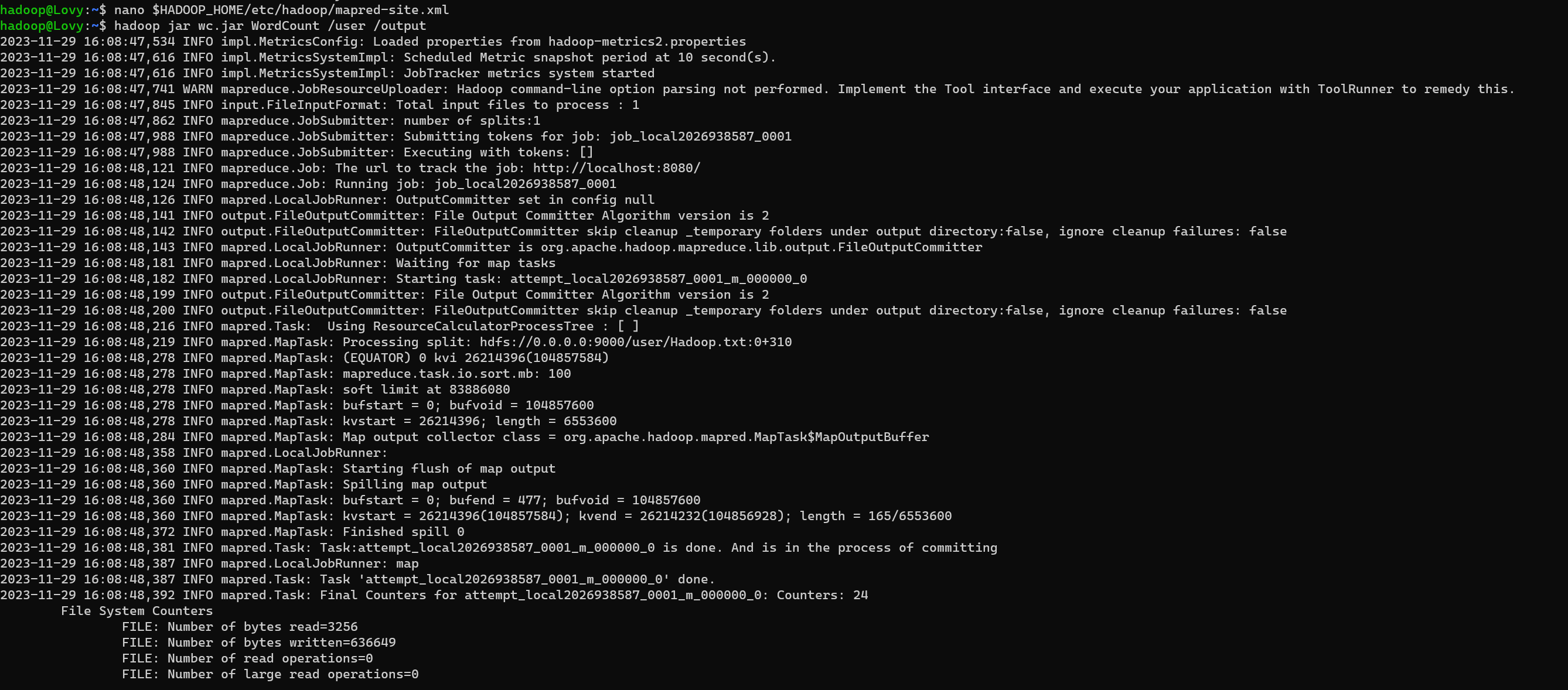
*hadoop com.sun.tools.javac.Main WordCount.java jar cf wc.jar WordCount\*.class*



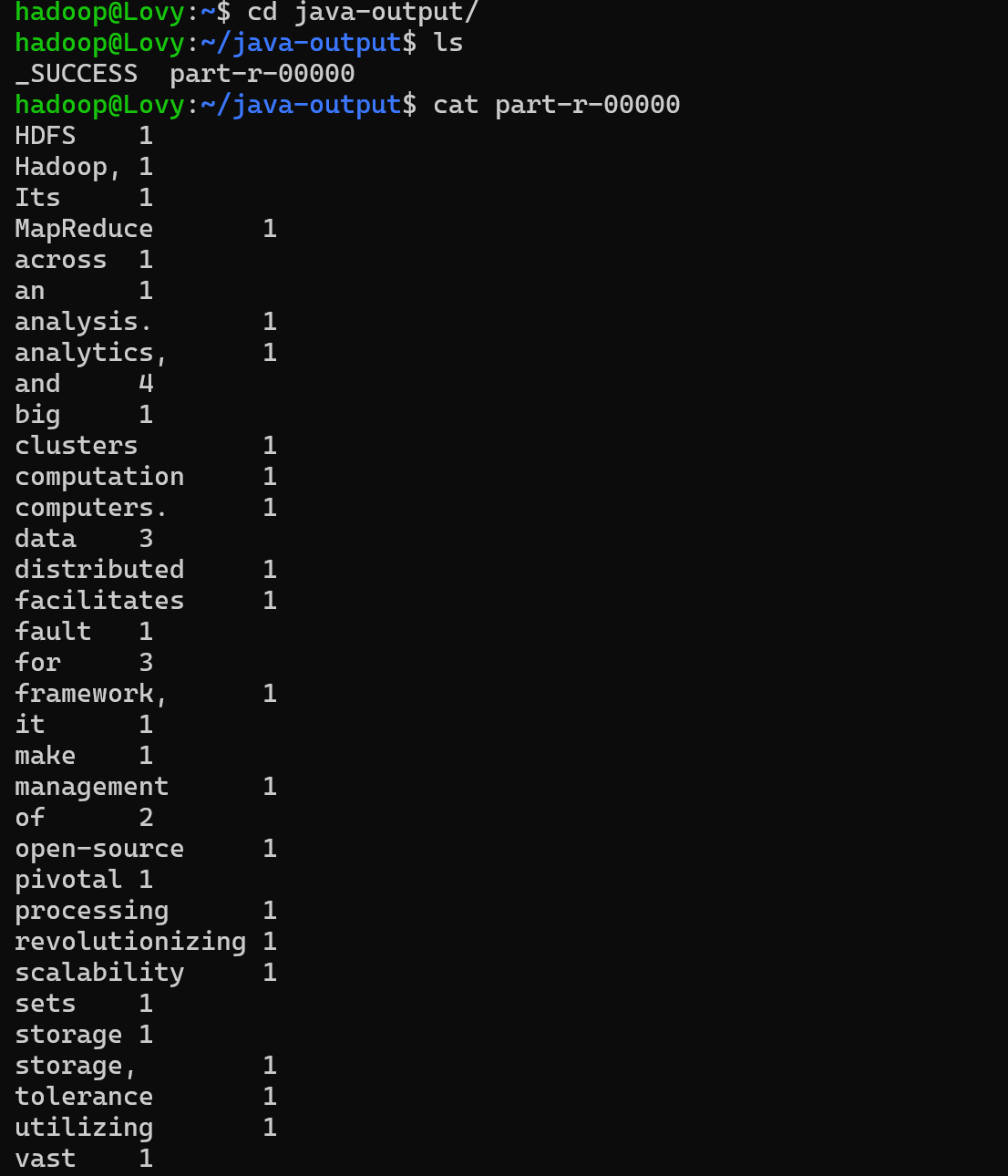
1. Prepare sample text files as input and store them in the Hadoop file system *(Refer Practical 3)*.



1. Execute the MapReduce application using the “*hadoop jar*” command with appropriate input and output paths.



1. Check the output using the “*hadoop fs -cat*” command to verify the word count results.



## Observations:

1. Observe the execution logs and any error messages for debugging purposes.
2. Compare the output with the expected results for accuracy verification.

## Results:

1. The Word Count program has successfully counted the occurrences of each word in the provided input text files.
2. The output displays the word counts for each unique word in the text.

## Conclusion:

The successful implementation of the Word Count MapReduce program in Java underscores the potential of Hadoop's distributed processing capabilities in handling data-intensive tasks, laying the groundwork for exploring more complex data processing operations in the Hadoop ecosystem.

# ­­Practical 5: Writing a Word count MapReduce program in Python.

## Aim: To implement a Word Count MapReduce program using Python. Objectives:

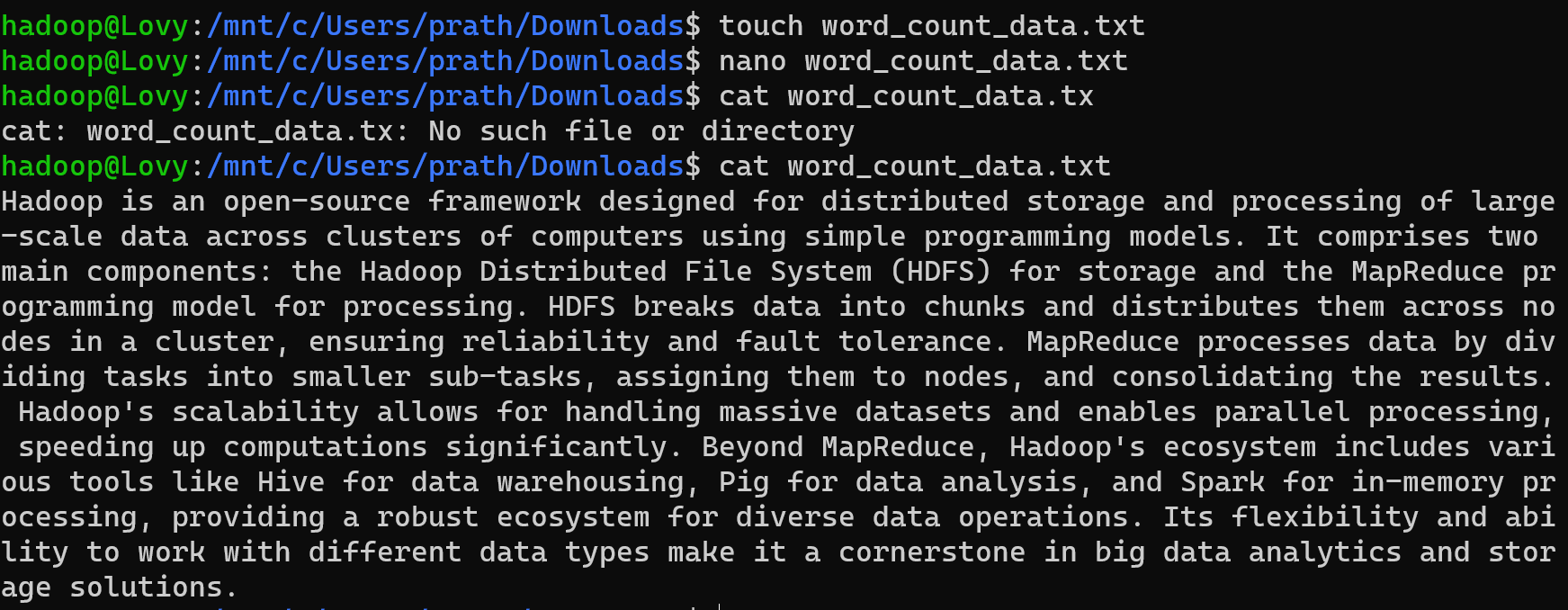
1. Implementing the concept of Hadoop Steaming
2. Understanding the fundamentals of the MapReduce programming model.
3. Implementing the Mapper and Reducer functions in Python for counting words.
4. Executing the Python program on the Hadoop cluster.

## Requirements:

1. Hadoop installed and configured in Pseudo-Distributed or Standalone mode.
2. Basic understanding of Python programming.

## Procedure:

1. Create a file with the name word\_count\_data.txt and add some data to it.



1. Create a mapper.py file that implements the mapper logic. It will read the data from STDIN and will split the lines into words, and will generate an output of each word with its individual count. Copy the below code to the mapper.py file

*#!/usr/bin/env python*

*# import sys because we need to read and write data to STDIN and STDOUT import sys*

*# reading entire line from STDIN (standard input) for line in sys.stdin:*

*# to remove leading and trailing whitespace line = line.strip()*

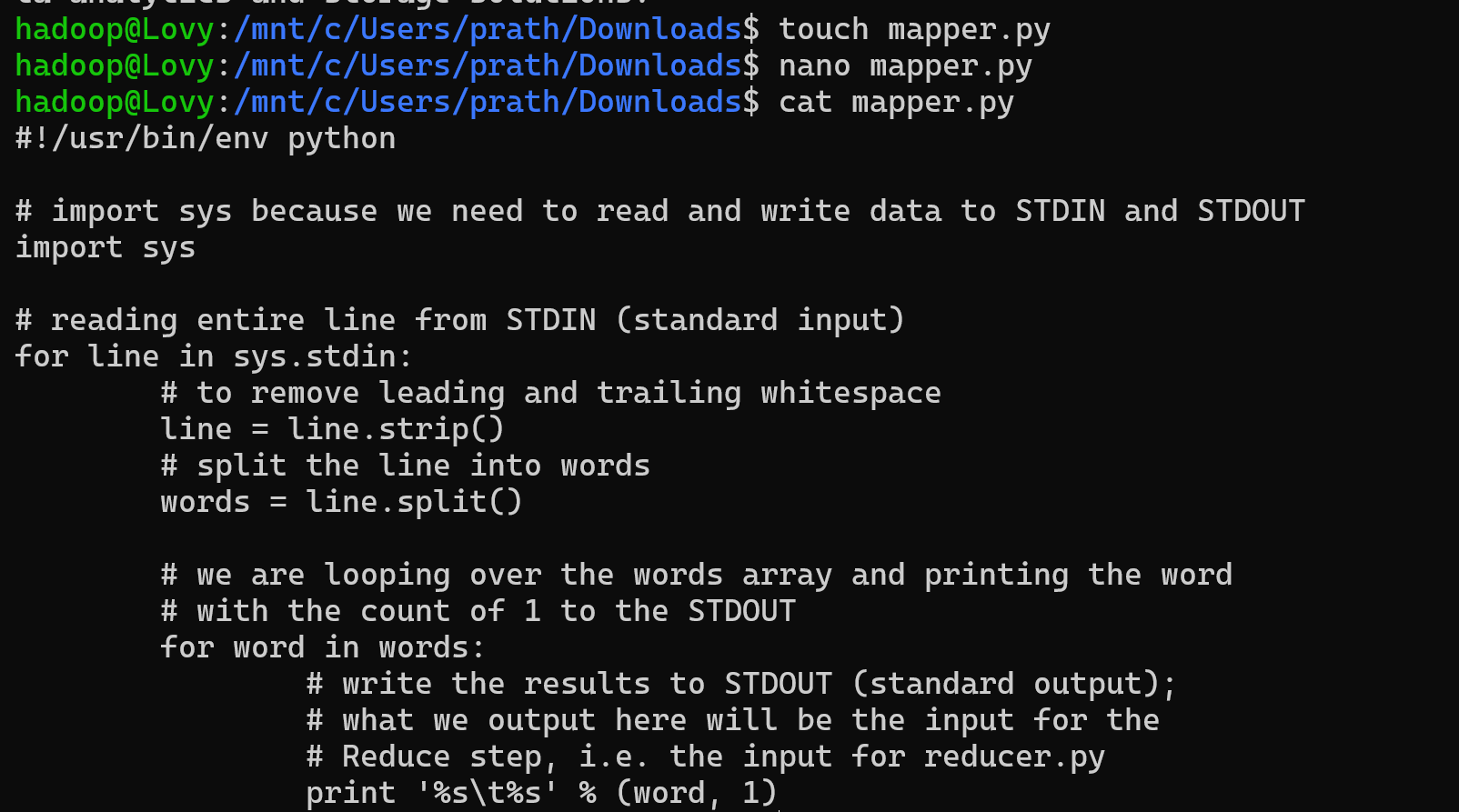
*# split the line into words words = line.split()*

*# we are looping over the words array and printing the word # with the count of 1 to the STDOUT*

*for word in words:*

*# write the results to STDOUT (standard output); # what we output here will be the input for the*

*# Reduce step, i.e. the input for reducer.py print ('%s\t%s' % (word, 1))*



1. Create a reducer.py file that implements the reducer logic. It will read the output of mapper.py from STDIN(standard input) and will aggregate the occurrence of each word and will write the final output to STDOUT. Copy the below code to the reducer.py file

*#!/usr/bin/env python*

*from operator import itemgetter import sys*

*current\_word = None current\_count = 0 word = None*

*# input comes from STDIN for line in sys.stdin:*

*line = line.strip()*

*word, count = line.split('\t', 1)*

*try:*

*count = int(count) except ValueError:*

*continue*

*if current\_word == word: current\_count += count*

*else:*

*if current\_word:*

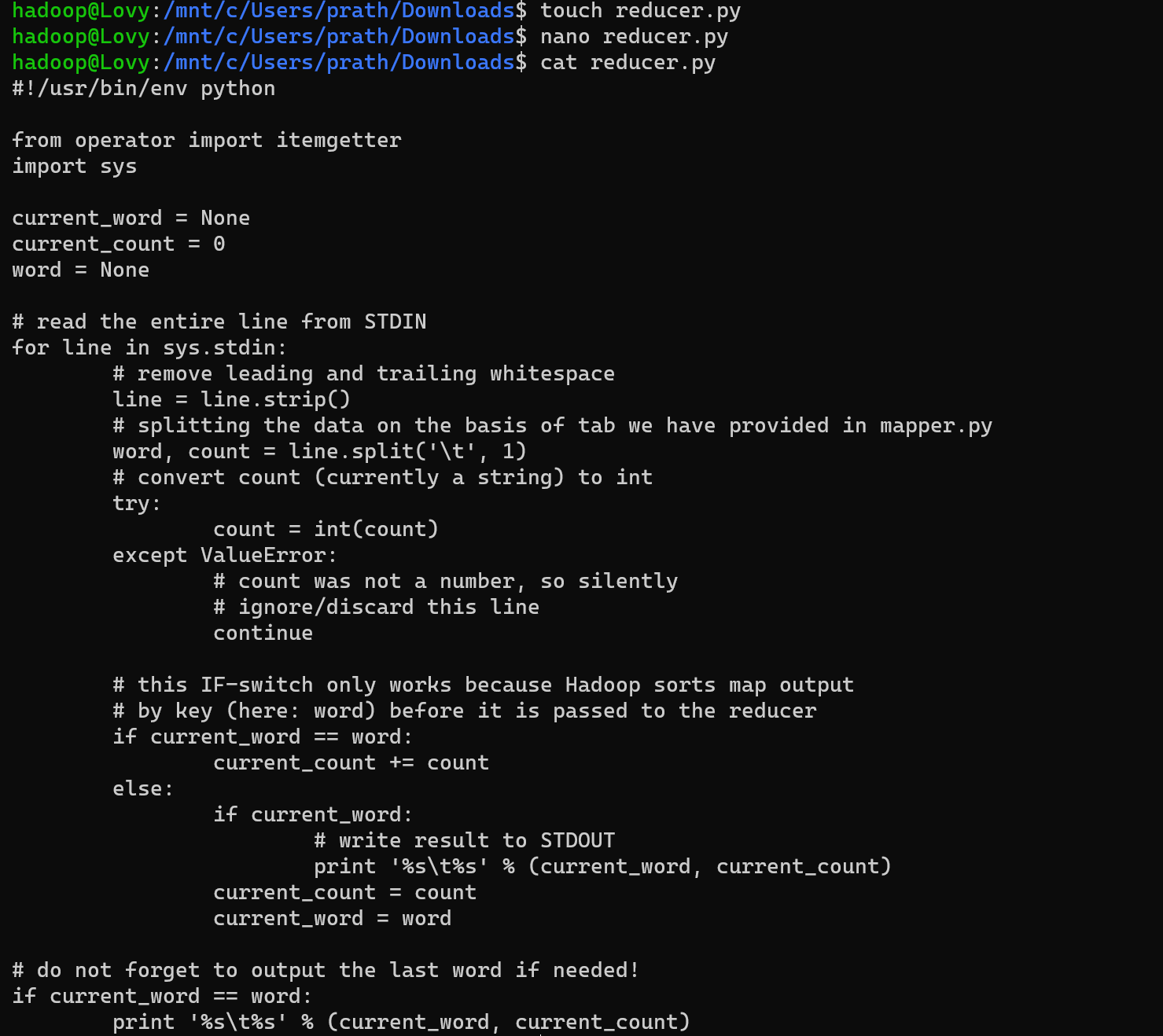
*# write result to STDOUT*

*print('%s\t%s' % (current\_word, current\_count)) current\_count = count*

*current\_word = word*

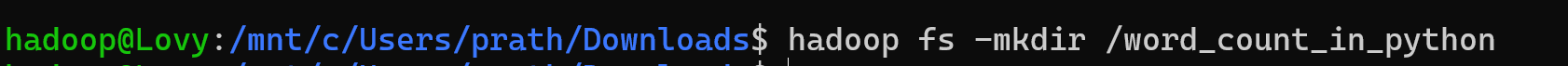
*# do not forget to output the last word if needed! if current\_word == word:*

*print('%s\t%s' % (current\_word, current\_count))*

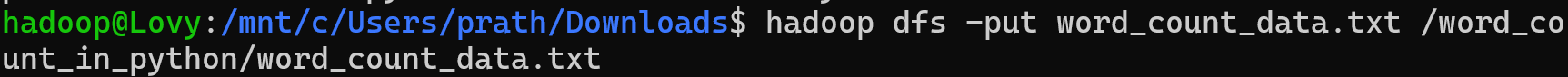


1. Now make a directory word\_count\_in\_python in our HDFS in the root directory that will store our word\_count\_data.txt file with the command “*hadoop fs -mkdir*

*/word\_count\_in\_python*”

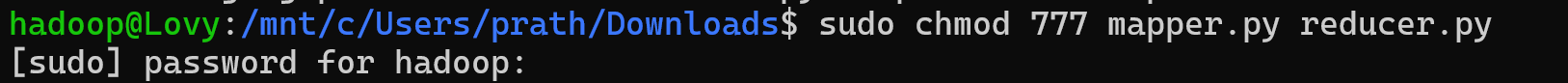


1. Copy word\_count\_data.txt to this folder in our HDFS



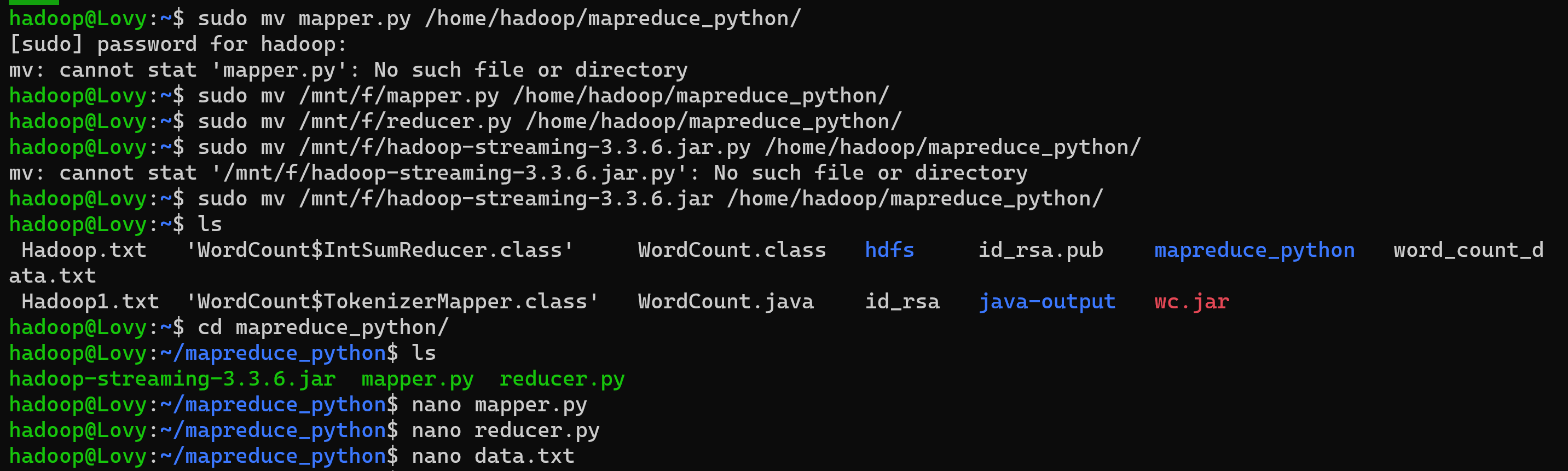
1. Give executable permission to the mapper.py and reducer.py with the help of the command

“*chmod 777 mapper.py reducer.py*”.



1. Now download the latest hadoop-streaming jar file from “*https://jar- download.com/artifacts/org.apache.hadoop/hadoop-streaming/3.3.6*”.





1. Run the python files with the help of the Hadoop streaming utility as shown below.

*hadoop jar hadoop-streaming-3.3.6.jar \*

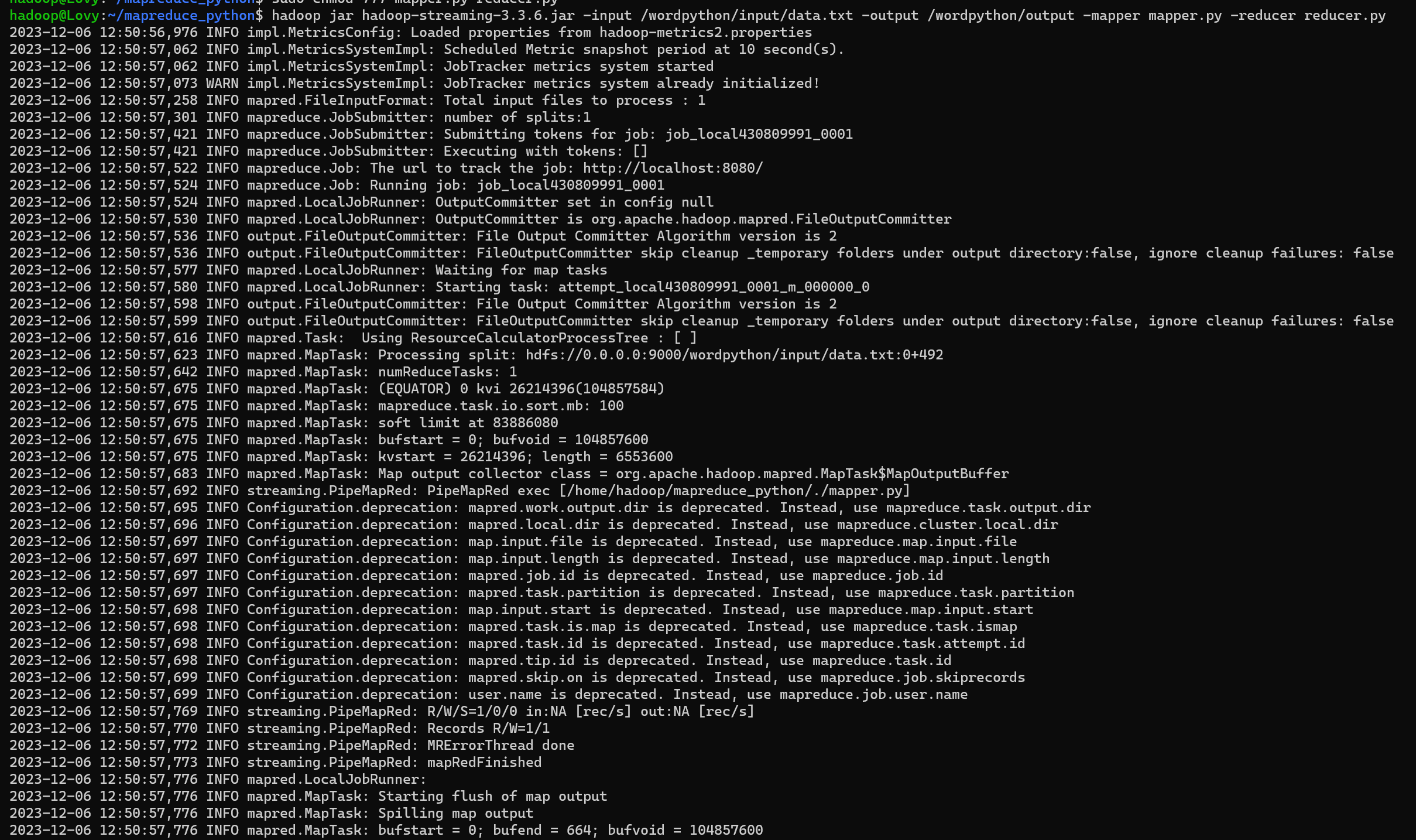
*-files mapper.py,reducer.py \*

*-mapper mapper.py \*

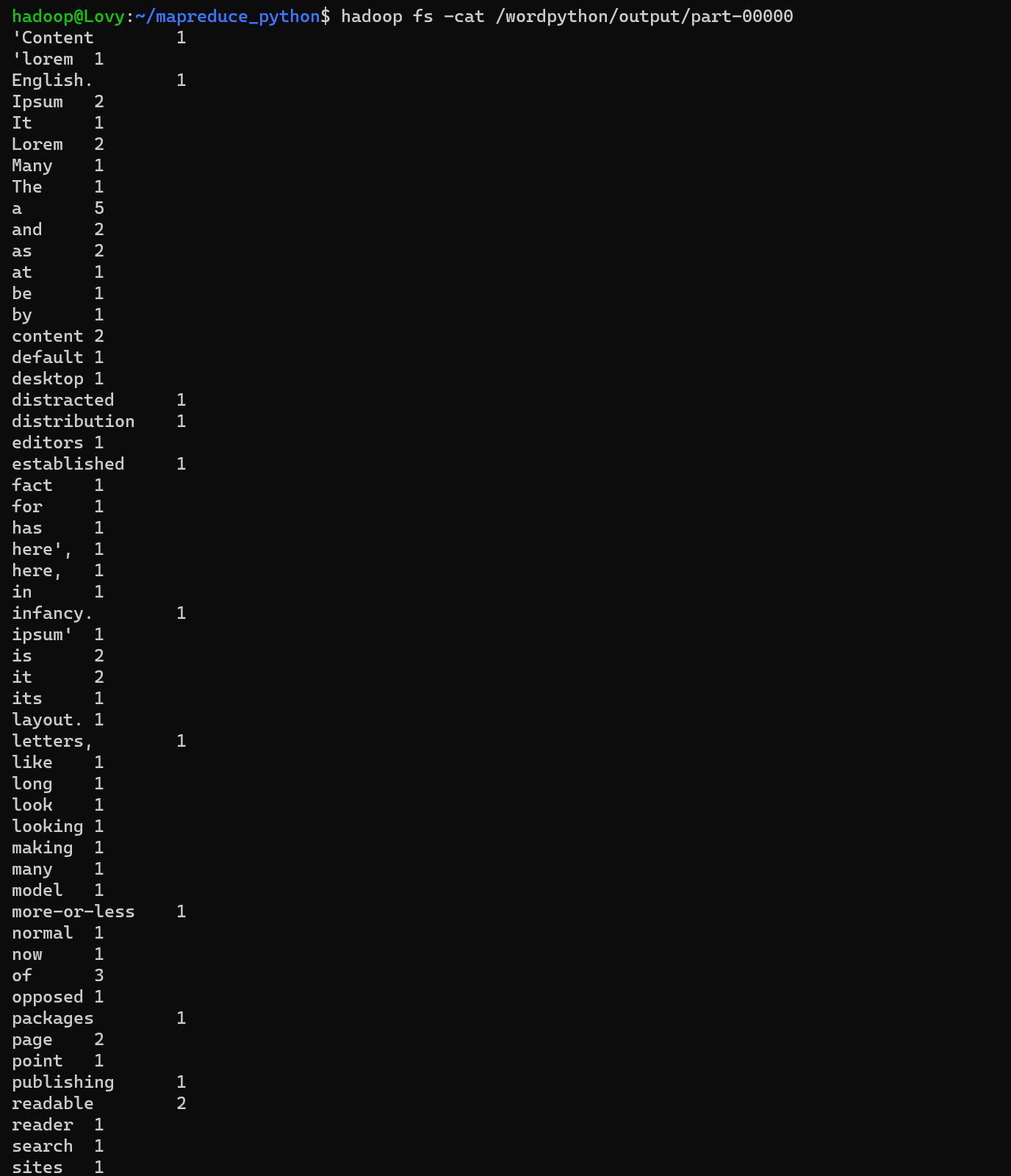
*-reducer reducer.py \*

*-input /user/shivan/word\_count\_data.txt \*

*-output /user/shivan/word\_count\_output*



1. Check the output using the “*hadoop fs -cat*” command to verify the word count results.



## Observations:

1. Monitor the execution logs and error messages to facilitate debugging, if necessary.
2. Compare the output generated by the Python program with the expected results.

## Results:

1. The Word Count MapReduce program in Python successfully counts the occurrences of each word in the provided input text files.
2. The output demonstrates the word counts for each unique word in the text.

## Conclusion:

The successful implementation of the Word Count MapReduce program in Python highlights the versatility of Hadoop's distributed computing capabilities, paving the way for exploring more intricate data processing tasks within the Hadoop ecosystem.