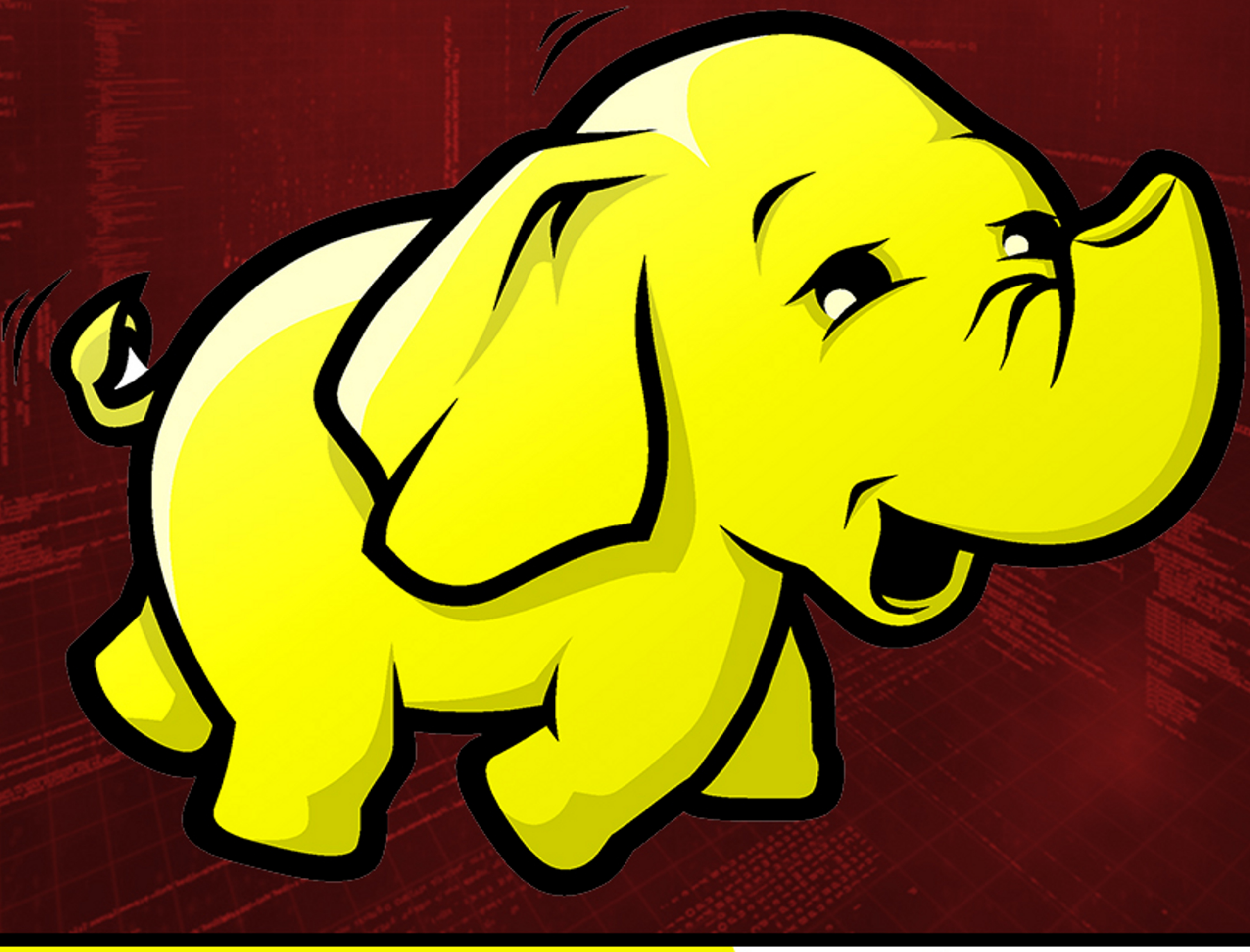
# APACHE HADOOP COOKBOOK

Hot Recipes for Apache Hadoop



RAMAN JHAJJ

Java Code Geeks

Java 2 Java Developers Resource Center

# Apache Hadoop Cookbook

# Contents

# Preface

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework.

Hadoop has become the de-facto tool used for Distributed computing. For this reason we have provided an abundance of tutorials here at Java Code Geeks, most of which can be found here: https://examples.javacodegeeks.com/category/enterprise-java/apache-hadoop/

In this ebook, we provide a compilation of Hadoop based examples that will help you kick-start your own web projects. We cover a wide range of topics, from installation and configuration, to distributed caching and streaming. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

# About the Author

Ramaninder has graduated from the Department of Computer Science and Mathematics of Georg-August University, Germany and currently works with a Big Data Research Center in Austria. He holds M.Sc in Applied Computer Science with specialization in Applied Systems Engineering and minor in Business Informatics.

He is also a Microsoft Certified Processional with more than 5 years of experience in Java, C#, Web development and related technologies. Currently, his main interests are in Big Data Ecosystem including batch and stream processing systems, Machine Learning and Web Applications.

# Chapter 1

# “Hello World” Example

## 1.1 Introduction

Hadoop is an Apache Software Foundation project. It is the open source version inspired by Google MapReduce and Google File System.

It is designed for distributed processing of large data sets across a cluster of systems often running on commodity standard hardware.

Hadoop is designed with an assumption that all hardware fails sooner or later and the system should be robust and able to handle the hardware failures automatically.

Apache Hadoop consists of two core components, they are:

- Distributed File System called Hadoop Distributed File System or HDFS for short.
- Framework and API for MapReduce jobs.

In this example, we are going to demonstrate the second component of Hadoop framework called MapReduce and we will do so by Word Count Example (Hello World program of the Hadoop Ecosystem) but first we shall understand what MapReduce actually is.

MapReduce is basically a software framework or programming model, which enable users to write programs so that data can be processed parallelly across multiple systems in a cluster. MapReduce consists of two parts Map and Reduce.

- Map: Map task is performed using a `map()` function that basically performs filtering and sorting. This part is responsible for processing one or more chunks of data and producing the output results which are generally referred as intermediate results. As shown in the diagram below, map task is generally processed in parallel provided the maping operation is independent of each other.
- Reduce: Reduce task is performed by `reduce()` function and performs a summary operation. It is responsible for consolidating the results produced by each of the Map task.

## 1.2 Hadoop Word-Count Example

Word count example is the "Hello World" program of the Hadoop and MapReduce. In this example, the program consists of MapReduce job that counts the number of occurrences of each word in a file. This job consists of two parts Map and Reduce. The Map task maps the data in the file and counts each word in data chunk provided to the map function. The outcome of the this task is passed to reduce which combines the data and output the final result on the disk.

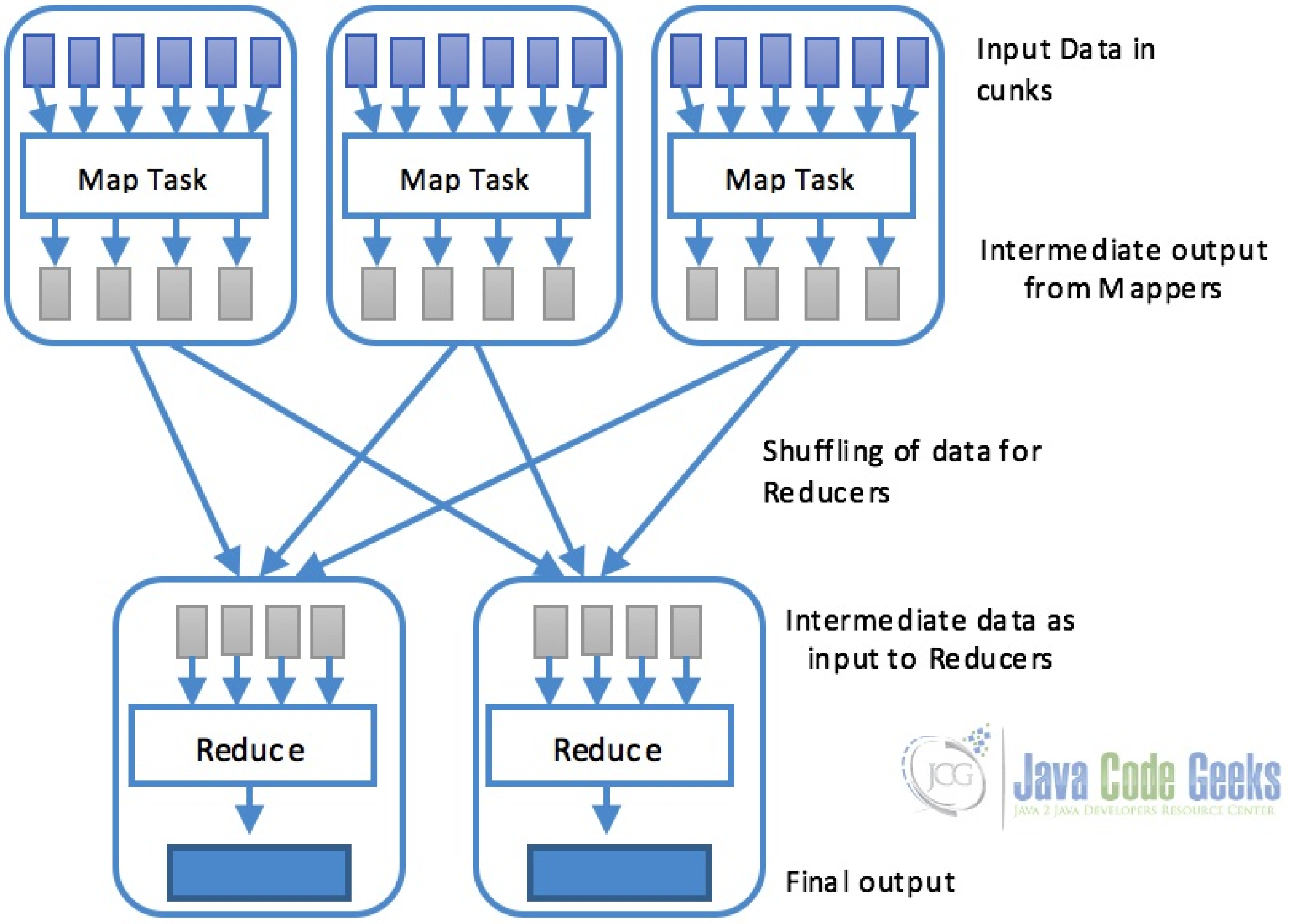

Input Data in cunks
Map Task
Map Task
Map Task
Intermediate output from Mappers
Shuffling of data for Reducers
Intermediate data as input to Reducers
Reduce
Reduce
Final output
Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Figure 1.1: Basic working of Map and Reduce Tasks in a MapReduce Job

### 1.2.1 Setup

We shall use Maven to setup a new project for Hadoop word count example. Setup a maven project in Eclipse and add the following Hadoop dependency to the `pom.xml`. This will make sure we have the required access to the Hadoop core library.

pom.xml

```
<dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-core</artifactId>
        <version>1.2.1</version>
    </dependency>
```

After adding the dependency, we are ready to write our word count code.

### 1.2.2 Mapper Code

The mapper task is responsible for tokenizing the input text based on space and create a list of words, then traverse over all the tokens and emit a key-value pair of each word with a count of one, for example, <Hello, 1>. Following is the `MapClass`, it needs to extends the MapReduce Mapper class and overrides the `map()` method. This method will receive a chunk of the input data to be processed. When this method is called the value parameter of the function will tokenize the data into words and context will write the intermediate output which will then be sent to one of the reducers.

MapClass.java

```
package com.javacodegeeks.examples.wordcount;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapClass extends Mapper<LongWritable, Text, Text, IntWritable>{

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

    @Override
    protected void map(LongWritable key, Text value,
                        Context context)
                        throws IOException, InterruptedException {

                String line = value.toString();
                StringTokenizer st = new StringTokenizer(line," ");

                while(st.hasMoreTokens()){
                        word.set(st.nextToken());
                        context.write(word,one);
                }

        }
}
```

## 1.2.3 Reducer Code

Following code snippet contains `ReduceClass` which extends the MapReduce Reducer class and overwrites the `reduce()` function. This function is called after the map method and receives keys which in this case are the word and also the corresponding values. Reduce method iterates over the values, adds them and reduces to a single value before finally writing the word and the number of occurrences of the word to the output file.

ReduceClass.java

```
package com.javacodegeeks.examples.wordcount;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReduceClass extends Reducer{

        @Override
        protected void reduce(Text key, Iterable values,
                        Context context)
                        throws IOException, InterruptedException {
```

```
                int sum = 0;
                Iterator valuesIt = values.iterator();

                while(valuesIt.hasNext()){
                        sum = sum + valuesIt.next().get();
                }

                context.write(key, new IntWritable(sum));
        }
}
```

### 1.2.4 Putting it all together, The Driver Class

So now when we have our map and reduce classes ready, it is time to put it all together as a single job which is done in a class called driver class. This class contains the `main()` method to setup and run the job. Following code checks for the correct input arguments which are the paths of the input and output files. Followed by setting up and running the job. At the end, it informs the user if the job is completed successfully or not. The resultant file with the word counts and the corresponding number of occurrences will be present in the provided output path.

WordCount.java

```
package com.javacodegeeks.examples.wordcount;
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.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;

public class WordCount extends Configured implements Tool{

        public static void main(String[] args) throws Exception{
                int exitCode = ToolRunner.run(new WordCount(), args);
                System.exit(exitCode);
        }

        public int run(String[] args) throws Exception {
                if (args.length != 2) {
                        System.err.printf("Usage: %s needs two arguments, input and output
files\n", getClass().getSimpleName());
                        return -1;
                }

                Job job = new Job();
                job.setJarByClass(WordCount.class);
                job.setJobName("WordCounter");

                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                job.setMapperClass(MapClass.class);
                job.setReducerClass(ReduceClass.class);
                int returnValue = job.waitForCompletion(true) ? 0:1;

                if(job.isSuccessful()) {
                        System.out.println("Job was successful");
                } else if(!job.isSuccessful()) {
                        System.out.println("Job was not successful");
                }

                return returnValue;
        }
}
```

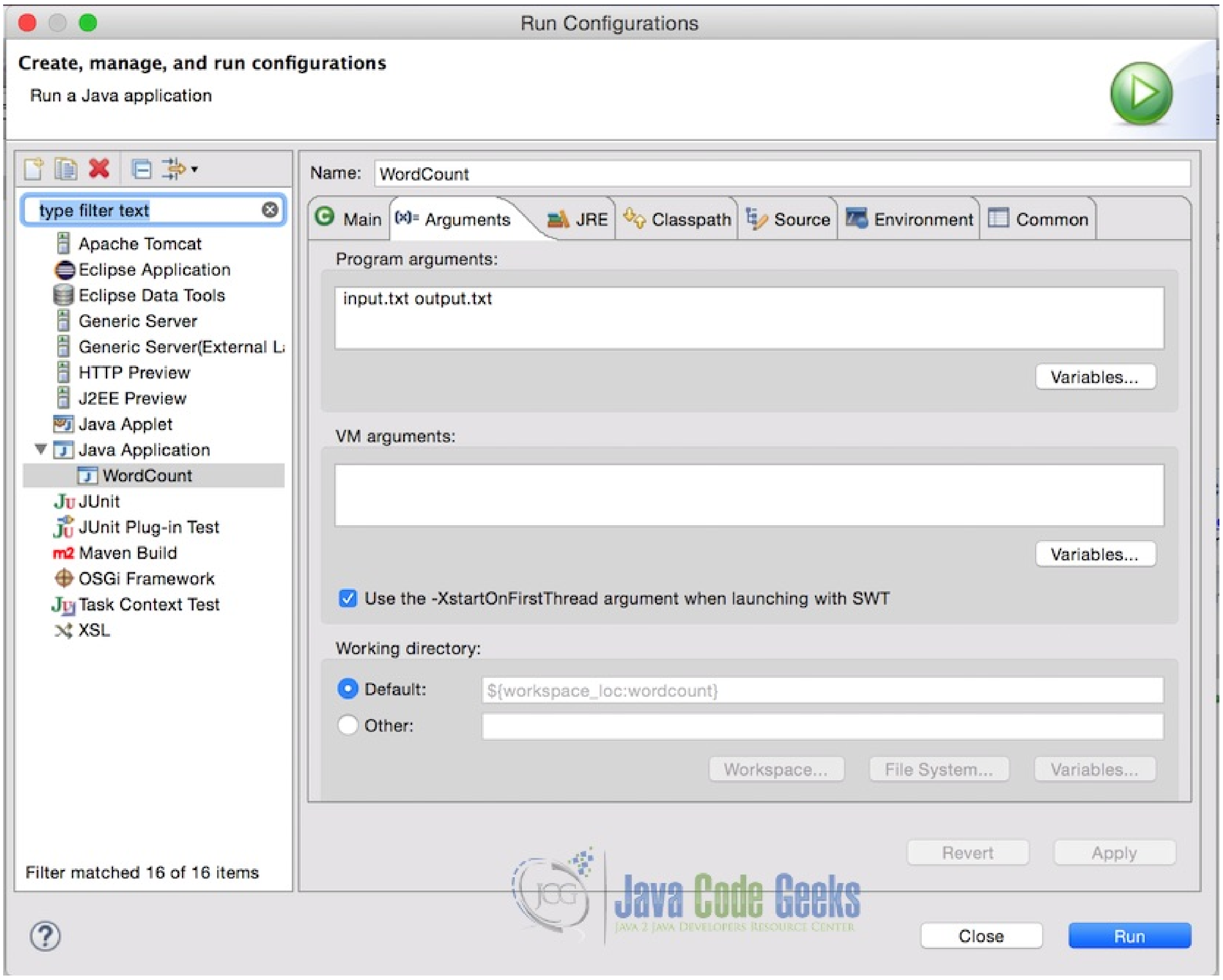
## 1.3 Running the example

To test the code implementation. We can run the program for testing purpose from Eclipse itself. First of all, create an input.txt file with dummy data. For the testing purpose, We have created a file with the following text in the project root.

input.txt

```
This is the example text file for word count example also knows as hello world example of ↩
    the Hadoop ecosystem.
This example is written for the examples article of java code geek
The quick brown fox jumps over the lazy dog.
The above line is one of the most famous lines which contains all the english language ↩
    alphabets.
```

In Eclipse, Pass the input file and output file name in the project arguments. Following is how the arguments looks like. In this case, the input file is in the root of the project that is why just filename is required, but if your input file is in some other location, you should provide the complete path.



Name: WordCount

Program arguments:

```
input.txt output.txt
```

VM arguments:

[✓] Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

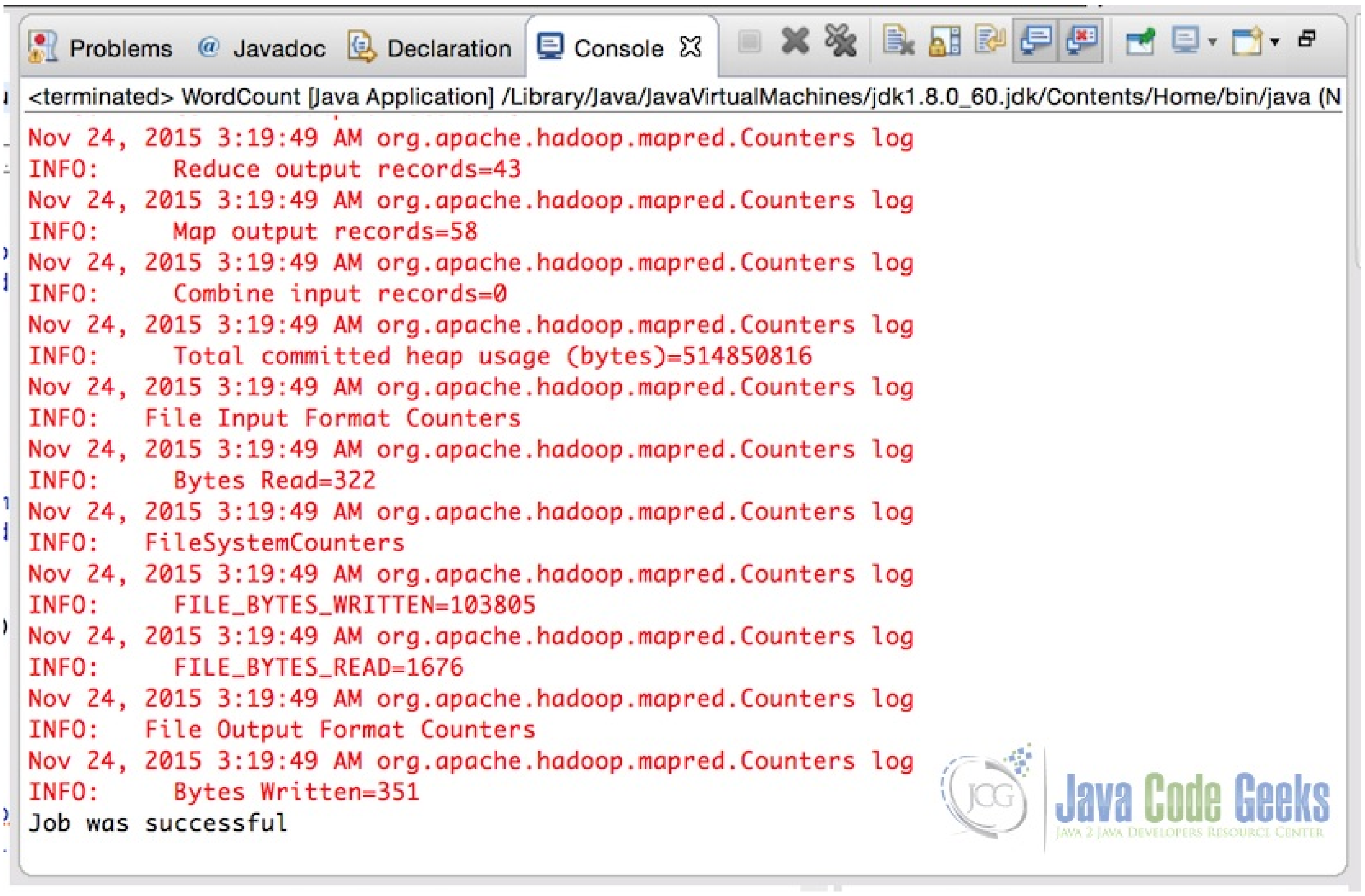
[● Default:] ${workspace_loc:wordcount}

Other:

Figure 1.2: Run Configuration of Eclipse Project

**Note:** Make sure the output file does not exist already. If it does, the program will throw an error.

After setting the arguments, simply run the application. Once the application is successfully completed, console will show the output



```
<terminated> WordCount [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (N
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Reduce output records=43
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Map output records=58
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Combine input records=0
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Total committed heap usage (bytes)=514850816
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   File Input Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Read=322
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   FileSystemCounters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_WRITTEN=103805
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_READ=1676
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   File Output Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Written=351
Job was successful
```

Figure 1.3: Console output in Eclipse

We are specifically interested in the last line:

```
Job was successful
```

That indicates the successful execution of the MapReduce job. This means that the output file is written in the destination provided in the arguments. Following is how the output file of the provided input looks like.

output

```
Hadoop  1
The     2
This    2
above   1
all     1
alphabets.      1
also    1
article 1
as      1
brown   1
code    1
contains        1
count   1
dog.    1
ecosystem.      1
english 1
example 4
examples        1
```

```
famous  1
file    1
for     2
fox     1
geek    1
hello   1
is      3
java    1
jumps   1
knows   1
language        1
lazy    1
line    1
lines   1
most    1
of      3
one     1
over    1
quick   1
text    1
the     6
which   1
word    1
world   1
written 1
```

## 1.4 Download the complete source code

This was an example of Word Count(Hello World) program of Hadoop MapReduce.

**Download** You can download the full source code of this example here: **WordCountExample**

# Chapter 2

# How to Install Apache Hadoop on Ubuntu

In this example, we will see the details of how to install Apache Hadoop on an Ubuntu system.

We will go through all the required steps starting with the required pre-requisites of Apache Hadoop followed by how to configure Hadoop and we will finish this example by learning how to insert data into Hadoop and how to run an example job on that data.

## 2.1 Introduction

The example will describe all the required steps for installing a single-node Apache Hadoop cluster on Ubuntu 15.10. Hadoop is a framework for distributed processing of application on large clusters of commodity hardware. It is written in Java and follows the MapReduce computing paradigm.

## 2.2 Prerequisites

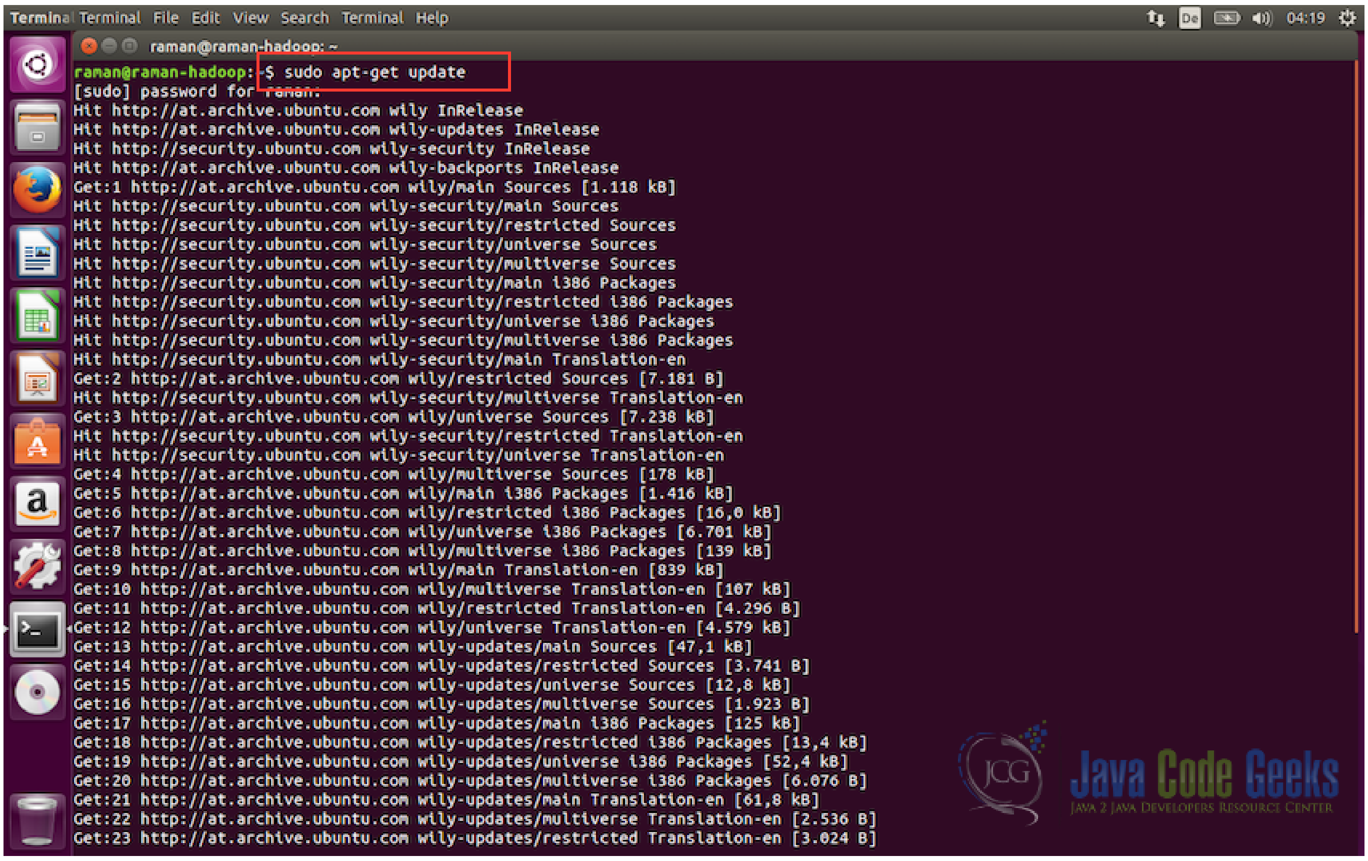
Following are the prerequisites of running Apache Hadoop on Ubuntu. Follow the steps to get all the prerequisites in place.

### 2.2.1 Installing Java

As Apache Hadoop is written in Java, it needs latest Java to be installed in the system. To install Java, first of all update the source list

```
#Update the source list
sudo apt-get update
```

It should update all the existing packages as shown in the screeenshot below.



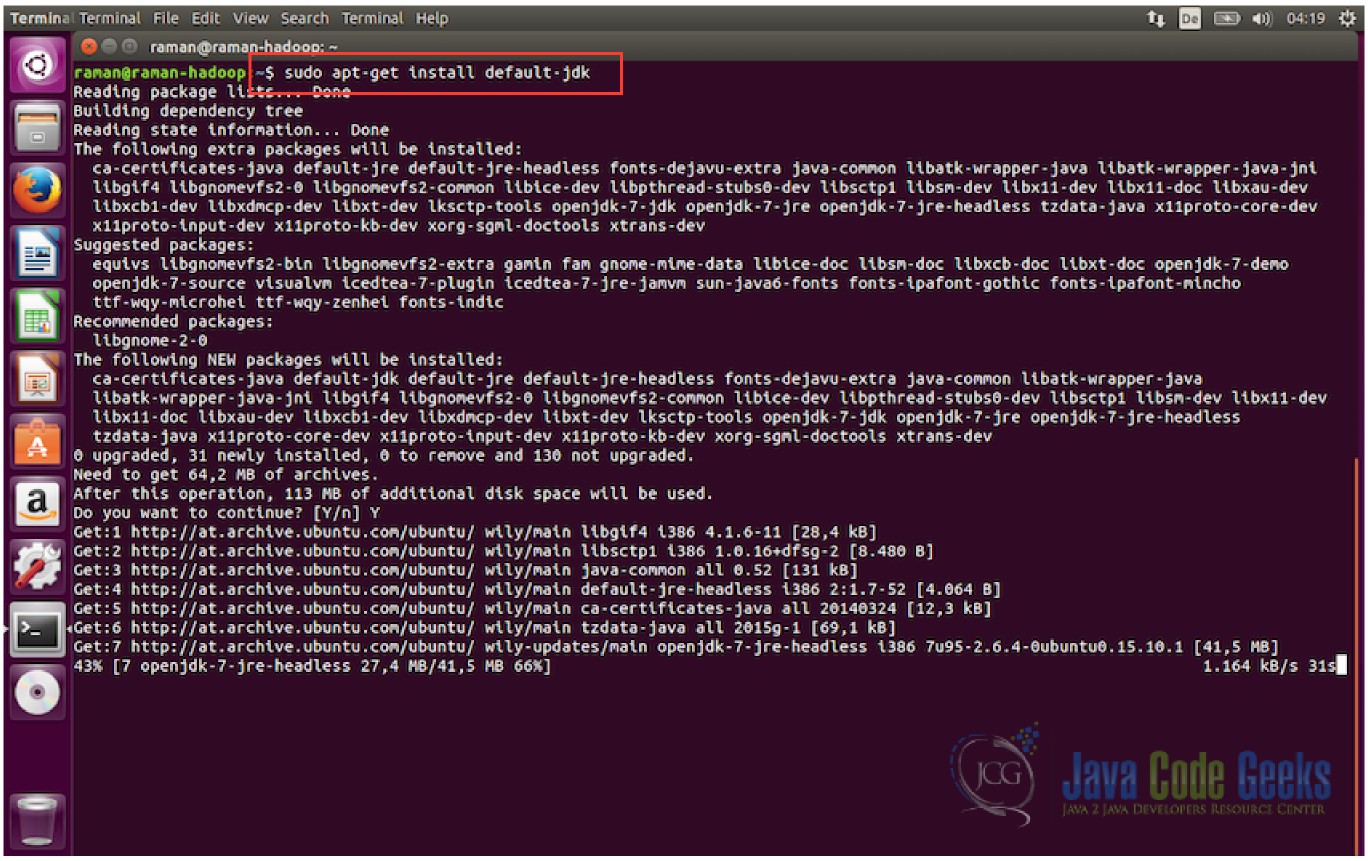
```
raman@raman-hadoop:~$ sudo apt-get update
[sudo] password for raman:
Hit http://at.archive.ubuntu.com wily InRelease
Hit http://at.archive.ubuntu.com wily-updates InRelease
Hit http://security.ubuntu.com wily-security InRelease
Hit http://at.archive.ubuntu.com wily-backports InRelease
Get:1 http://at.archive.ubuntu.com wily/main Sources [1.118 kB]
Hit http://security.ubuntu.com wily-security/main Sources
Hit http://security.ubuntu.com wily-security/restricted Sources
Hit http://security.ubuntu.com wily-security/universe Sources
Hit http://security.ubuntu.com wily-security/multiverse Sources
Hit http://security.ubuntu.com wily-security/main i386 Packages
Hit http://security.ubuntu.com wily-security/restricted i386 Packages
Hit http://security.ubuntu.com wily-security/universe i386 Packages
Hit http://security.ubuntu.com wily-security/multiverse i386 Packages
Hit http://security.ubuntu.com wily-security/main Translation-en
Get:2 http://at.archive.ubuntu.com wily/restricted Sources [7.181 B]
Hit http://security.ubuntu.com wily-security/multiverse Translation-en
Get:3 http://at.archive.ubuntu.com wily/universe Sources [7.238 kB]
Hit http://security.ubuntu.com wily-security/restricted Translation-en
Hit http://security.ubuntu.com wily-security/universe Translation-en
Get:4 http://at.archive.ubuntu.com wily/multiverse Sources [178 kB]
Get:5 http://at.archive.ubuntu.com wily/main i386 Packages [1.416 kB]
Get:6 http://at.archive.ubuntu.com wily/restricted i386 Packages [16,0 kB]
Get:7 http://at.archive.ubuntu.com wily/universe i386 Packages [6.701 kB]
Get:8 http://at.archive.ubuntu.com wily/multiverse i386 Packages [139 kB]
Get:9 http://at.archive.ubuntu.com wily/main Translation-en [839 kB]
Get:10 http://at.archive.ubuntu.com wily/multiverse Translation-en [107 kB]
Get:11 http://at.archive.ubuntu.com wily/restricted Translation-en [4.296 B]
Get:12 http://at.archive.ubuntu.com wily/universe Translation-en [4.579 kB]
Get:13 http://at.archive.ubuntu.com wily-updates/main Sources [47,1 kB]
Get:14 http://at.archive.ubuntu.com wily-updates/restricted Sources [3.741 B]
Get:15 http://at.archive.ubuntu.com wily-updates/universe Sources [12,8 kB]
Get:16 http://at.archive.ubuntu.com wily-updates/multiverse Sources [1.923 B]
Get:17 http://at.archive.ubuntu.com wily-updates/main i386 Packages [125 kB]
Get:18 http://at.archive.ubuntu.com wily-updates/restricted i386 Packages [13,4 kB]
Get:19 http://at.archive.ubuntu.com wily-updates/universe i386 Packages [52,4 kB]
Get:20 http://at.archive.ubuntu.com wily-updates/multiverse i386 Packages [6.076 B]
Get:21 http://at.archive.ubuntu.com wily-updates/main Translation-en [61,8 kB]
Get:22 http://at.archive.ubuntu.com wily-updates/multiverse Translation-en [2.536 B]
Get:23 http://at.archive.ubuntu.com wily-updates/restricted Translation-en [3.024 B]
```

Figure 2.1: Update Source List

Now install the default jdk using the following command.

```
# The OpenJDK project is the default version of Java
sudo apt-get install default-jdk
```

The OpenJDK is the default version of Java for Ubuntu Linux. It should be successfully installed with the `apt-get` command.

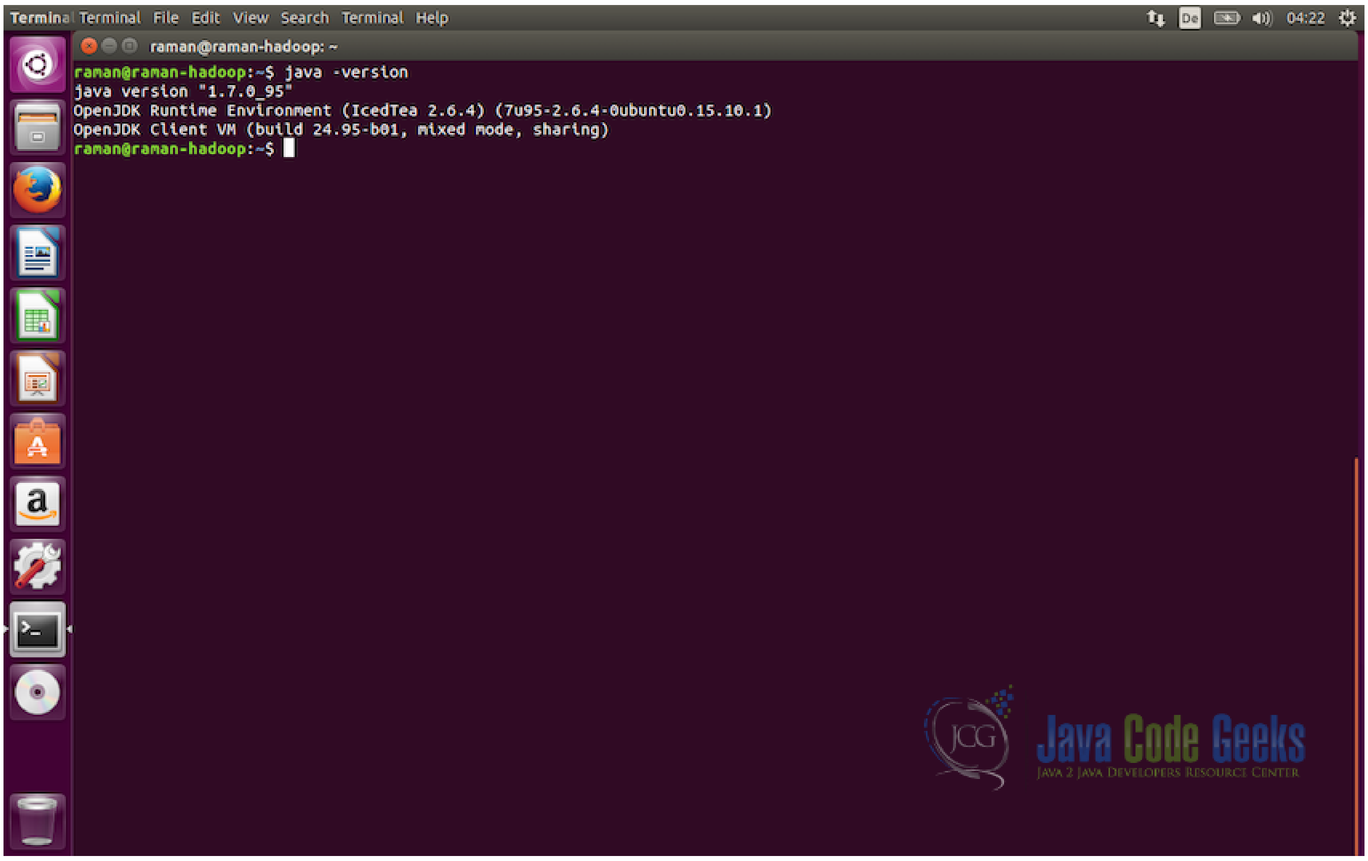


```
raman@raman-hadoop:~$ sudo apt-get install default-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
  libgif4 libgnomevfs2-0 libgnomevfs2-common libice-dev libpthread-stubs0-dev libsctp1 libsm-dev libx11-dev libx11-doc libxau-dev
  libxcb1-dev libxdmcp-dev libxt-dev lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless tzdata-java x11proto-core-dev
  x11proto-input-dev x11proto-kb-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  equivs libgnomevfs2-bin libgnomevfs2-extra gamin fam gnome-mime-data libice-doc libsm-doc libxcb-doc libxt-doc openjdk-7-demo
  openjdk-7-source visualvm icedtea-7-plugin icedtea-7-jre-jamvm sun-java6-fonts fonts-ipafont-gothic fonts-ipafont-mincho
  ttf-wqy-microhei ttf-wqy-zenhei fonts-indic
Recommended packages:
  libgnome-2-0
The following NEW packages will be installed:
  ca-certificates-java default-jdk default-jre default-jre-headless fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni libgif4 libgnomevfs2-0 libgnomevfs2-common libice-dev libpthread-stubs0-dev libsctp1 libsm-dev libx11-dev
  libx11-doc libxau-dev libxcb1-dev libxdmcp-dev libxt-dev lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless
  tzdata-java x11proto-core-dev x11proto-input-dev x11proto-kb-dev xorg-sgml-doctools xtrans-dev
0 upgraded, 31 newly installed, 0 to remove and 130 not upgraded.
Need to get 64,2 MB of archives.
After this operation, 113 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://at.archive.ubuntu.com/ubuntu/ wily/main libgif4 i386 4.1.6-11 [28,4 kB]
Get:2 http://at.archive.ubuntu.com/ubuntu/ wily/main libsctp1 i386 1.0.16+dfsg-2 [8.480 B]
Get:3 http://at.archive.ubuntu.com/ubuntu/ wily/main java-common all 0.52 [131 kB]
Get:4 http://at.archive.ubuntu.com/ubuntu/ wily/main default-jre-headless i386 2:1.7-52 [4.064 B]
Get:5 http://at.archive.ubuntu.com/ubuntu/ wily/main ca-certificates-java all 20140324 [12,3 kB]
Get:6 http://at.archive.ubuntu.com/ubuntu/ wily/main tzdata-java all 2015g-1 [69,1 kB]
Get:7 http://at.archive.ubuntu.com/ubuntu/ wily-updates/main openjdk-7-jre-headless i386 7u95-2.6.4-0ubuntu0.15.10.1 [41,5 MB]
43% [7 openjdk-7-jre-headless 27,4 MB/41,5 MB 66%]                                                          1.164 kB/s 31s
```

Figure 2.2: Installing Java

The `default-jdk` installs the version `1.7` of Java. Version `1.7` will be fine to run Hadoop but if you would like, you can explicitely install version `1.8` also.

```
#Java Version
java -version
```



```
raman@raman-hadoop:~$ java -version
java version "1.7.0_95"
OpenJDK Runtime Environment (IcedTea 2.6.4) (7u95-2.6.4-0ubuntu0.15.10.1)
OpenJDK Client VM (build 24.95-b01, mixed mode, sharing)
raman@raman-hadoop:~$
```

Figure 2.3: Java Version

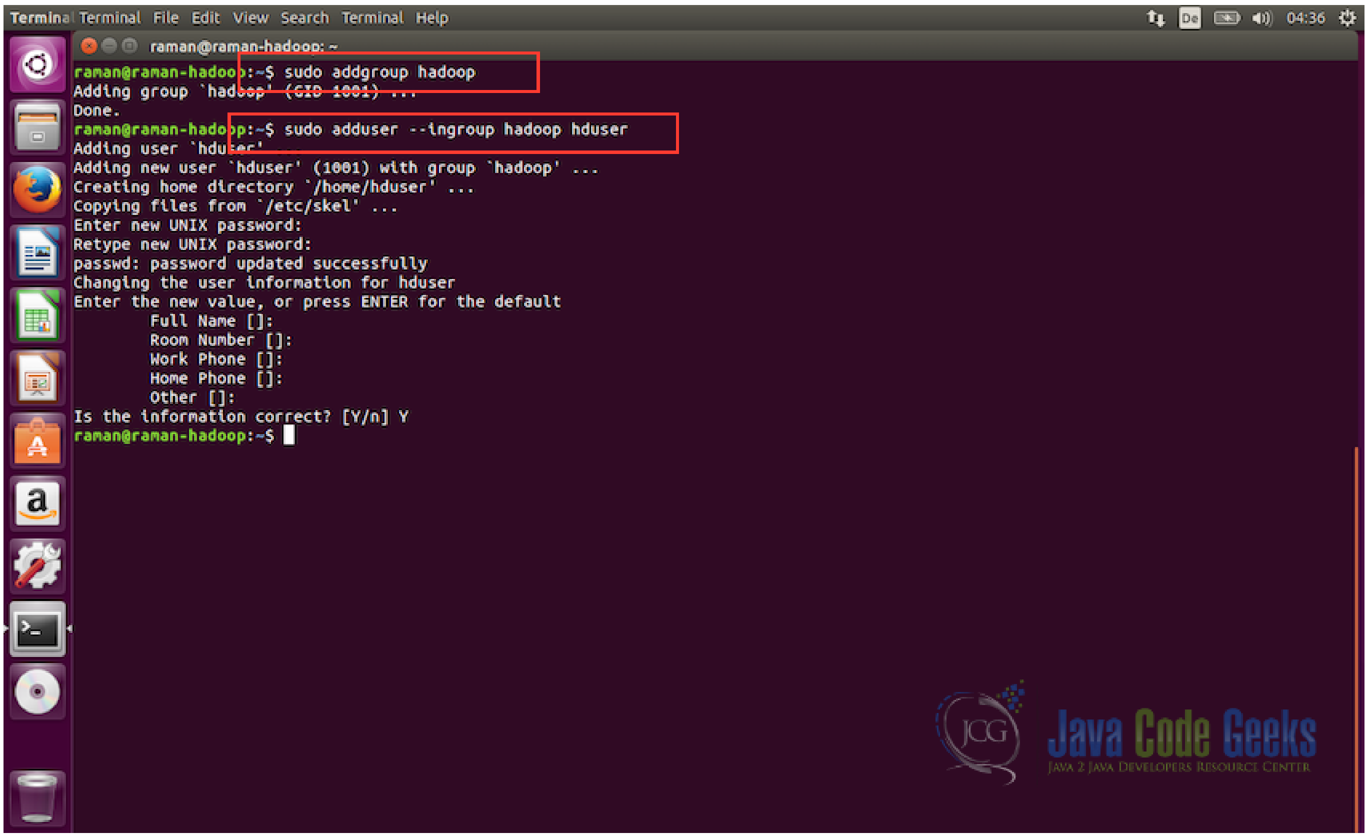
This completes the first prerequisite of the Apache Hadoop. Next we will move to creating a dedicated user which Hadoop can use for execution of its tasks.

### 2.2.2 Creating a Dedicated User

Hadoop needs a separate dedicated user for execution. With a complete control over the Hadoop executables and data folders. To create a new user, use the following command in the terminal.

```
#create a user group for hadoop
sudo addgroup hadoop

#create user hduser and add it to the hadoop usergroup
sudo adduser --ingroup hadoop hduser
```

The first command creates a new group with the name "hadoop" and the second command creates a new user "hduser" and assigns it to the "hadoop" group. We have kept all the user data like "First Name", "Phone Number" etc empty. You can keep it empty or assign values to the account as per your choice.



```
raman@raman-hadoop:~$ sudo addgroup hadoop
Adding group `hadoop' (GID 1001) ...
Done.
raman@raman-hadoop:~$ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
raman@raman-hadoop:~$
```

Figure 2.4: Creating dedicated user for Hadoop

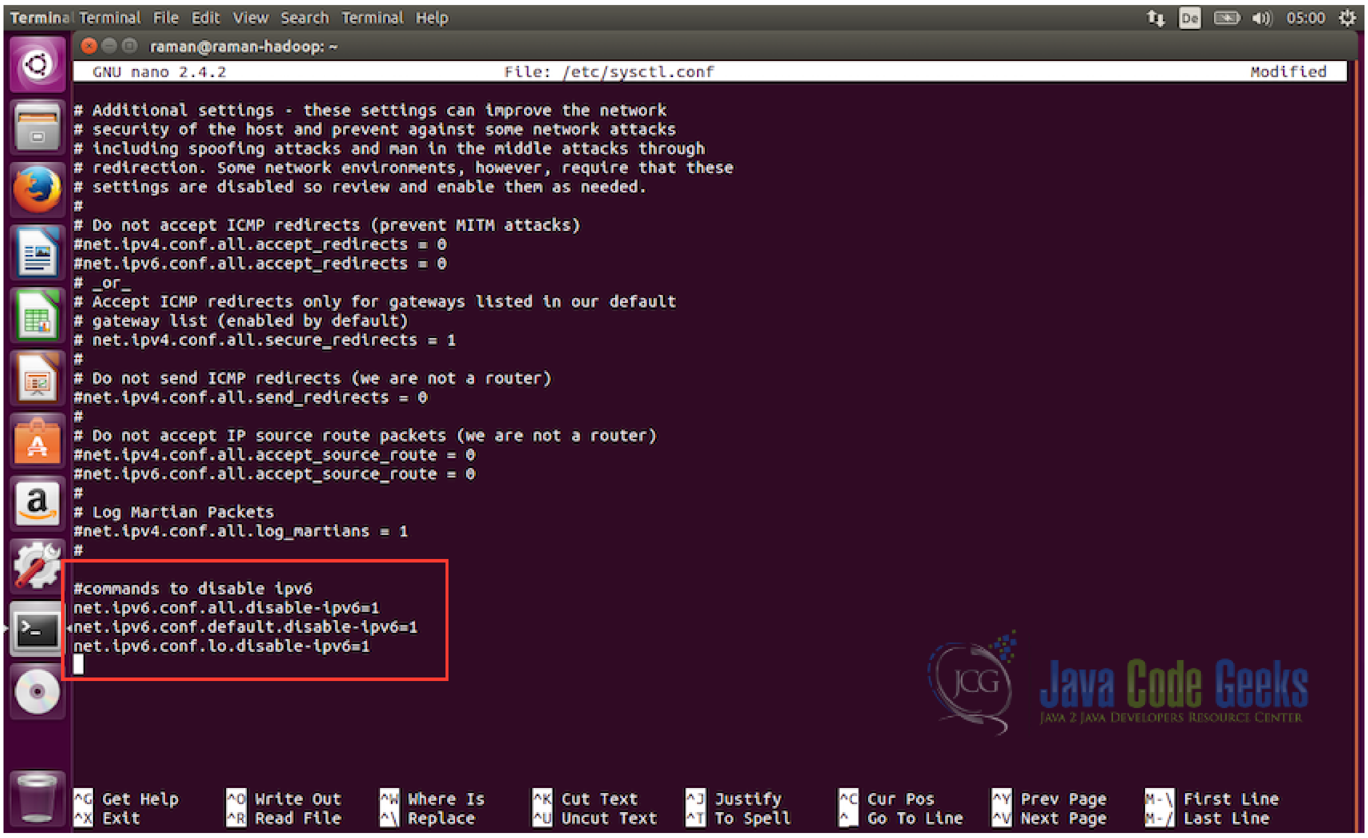
### 2.2.3 Disable ipv6

Next step is to disable ipv6 on all the machines. Hadoop is set to use ipv4 and that is why we need to disable ipv6 before creating a hadoop cluster. Open `/etc/sysctl.conf` as root using nano(or any other editor of your choice)

```
sudo nano /etc/sysctl.conf
```

and add the following lines at the end of the file.

```
#commands to disable ipv6
net.ipv6.conf.all.disable-ipv6=1
net.ipv6.conf.default.disable-ipv6=1
net.ipv6.conf.lo.disable-ipv6=1
```



```
GNU nano 2.4.2                    File: /etc/sysctl.conf                         Modified

# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
# _or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#

#commands to disable ipv6
net.ipv6.conf.all.disable-ipv6=1
net.ipv6.conf.default.disable-ipv6=1
net.ipv6.conf.lo.disable-ipv6=1

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

Figure 2.5: Disabling ipv6

Save the file using `ctrl+X` and then `Yes` when it prompts for saving the file. After this, to check if the ipv6 is properly disabled we can use the following command:

```
cat /proc/sys/net/ipv6/conf/all/disable-ipv6
```

it should return 0 or 1 as an output and we want it to be 1 as it symbolizes that the ipv6 is disable

### 2.2.4 Installing SSH and Setting up certificate

Hadoop requires SSH access to manage its remote nodes as well as node on local machine. For this example, we need to configure SSH access to localhost.

So, we will make sure we have SSH up and running and set up the public key access to allow it to login without a password. We will set up SSH certificate for allowing a password less authentication. Use the following commands to do the required steps.

`ssh` has two main components:

- ssh: The command we use to connect to remote machines - the client.
- sshd: The daemon that is running on the server and allows clients to connect to the server.

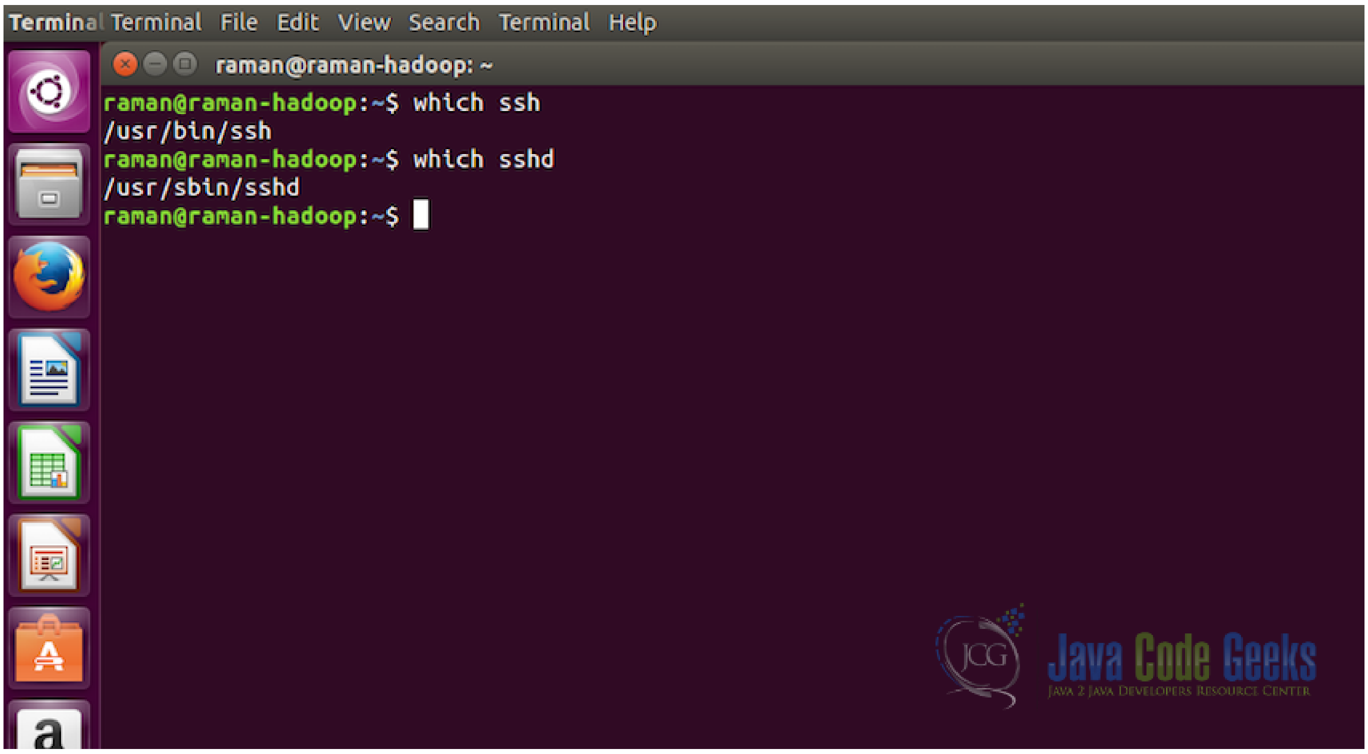
SSH is pre-enabled on ubuntu but to make sure `sshd` is enables we need to install `ssh` first using the following command.

```
#installing ssh
sudo apt-get install ssh
```

To make sure everything is setup properly, use the following commands and make sure the output is similar to the one displayed in the screenshot.

```
#Checking ssh
which ssh

#Checking sshd
which sshd
```

Both the above commands should show the path of the folder where `ssh` and `sshd` is installed as shown in the screenshot below. This is to make sure that both are present in the system.


Terminal Terminal File Edit View Search Terminal Help
raman@raman-hadoop: ~
JCG Java Code Geeks
Java 2 Java Developers Resource Center

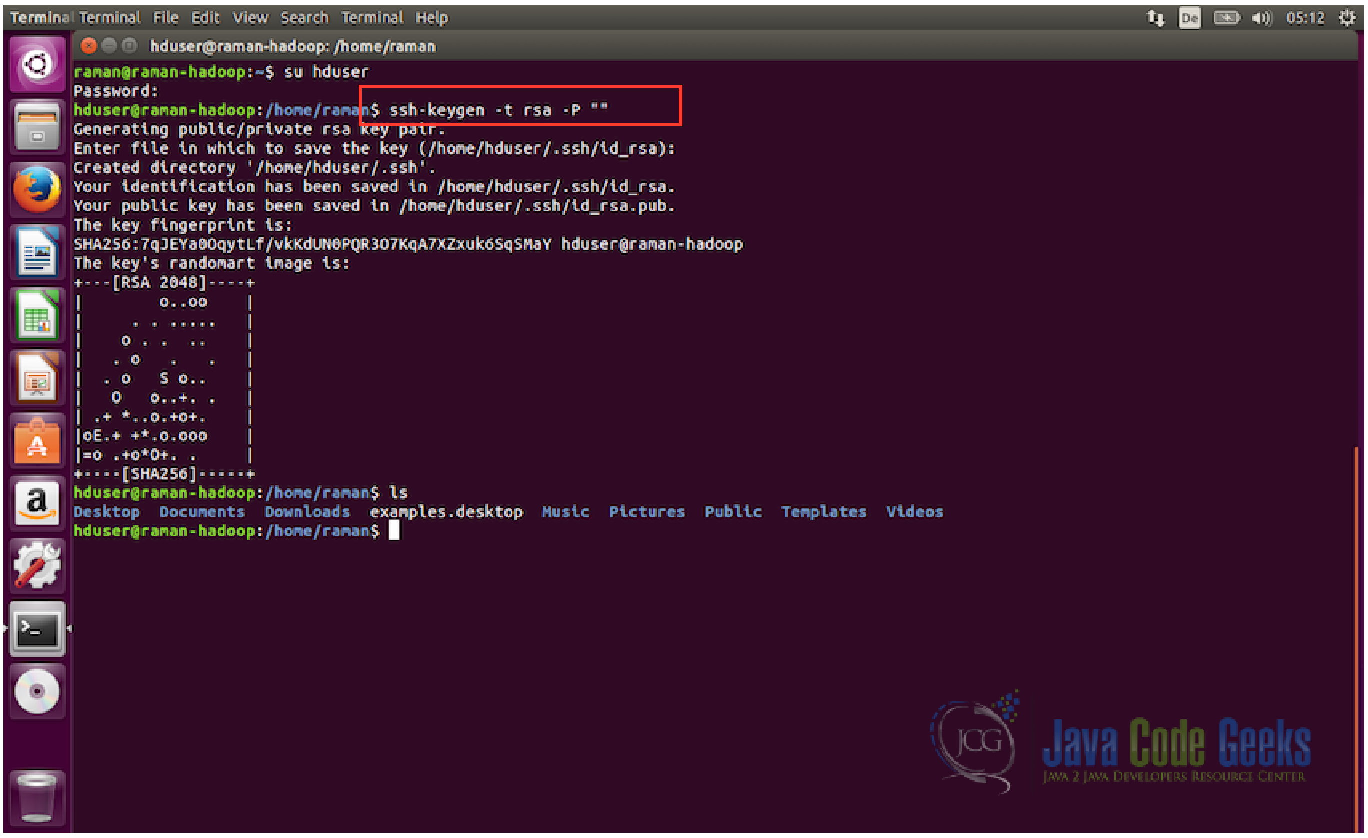
```
raman@raman-hadoop:~$ which ssh
/usr/bin/ssh
raman@raman-hadoop:~$ which sshd
/usr/sbin/sshd
raman@raman-hadoop:~$
```

Figure 2.6: Checking ssh and sshd

Now, in order to generate the `ssh` certificate we will switch to the `hduser` user. In the following command, we are keeping password empty while generating the key for ssh, you can give it some password if you would like to.

```
#change to user hduser
su hduser

#generate ssh key
ssh-keygen -t rsa -P ""
```

The second command will create an RSA key-pair for the machine. The password for this key will be empty as mentioned in the command. It will ask for the path to store the key with default path being $HOME/.ssh/id-rsa.pub, just press enter when prompted to keep the same path. If you plan to change the path then remember it as it will be needed in the next step.



```
raman@raman-hadoop:~$ su hduser
Password:
hduser@raman-hadoop:/home/raman$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7qJEYa0OqytLf/vkKdUN0PQR3O7KqA7XZxuk6SqSMaY hduser@raman-hadoop
The key's randomart image is:
+---[RSA 2048]----+
|         o..oo   |
|      . . .....  |
|     o . .  ..   |
|    . o    .   . |
|   . o   S o..   |
|    O   o..+. .  |
|  .+ *..o.+o+.   |
|oE.+ +*.o.ooo    |
|=o .+o*O+. .     |
+----[SHA256]-----+
hduser@raman-hadoop:/home/raman$ ls
Desktop  Documents  Downloads  examples.desktop  Music  Pictures  Public  Templates  Videos
hduser@raman-hadoop:/home/raman$ 
```

Figure 2.7: Generating ssh key

Enable SSH access to the machine with the key created in the previous step. For this, we have to add the key to the authorized keys list of the machine.

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

We can check if ssh works as following, is the `ssh` to localhost is succesful witout password prompt, then the certificate is properly enabled.

```
ssh localhost
```

By now, we are done with all the prerequisites for the Apache Hadoop. We will check how to setup Hadoop in the next section.

## 2.3 Installing Apache Hadoop

After all the prerequisites, we are ready to install Apache Hadoop on our Ubuntu 15.10 machine.

### 2.3.1 Download Apache Hadoop

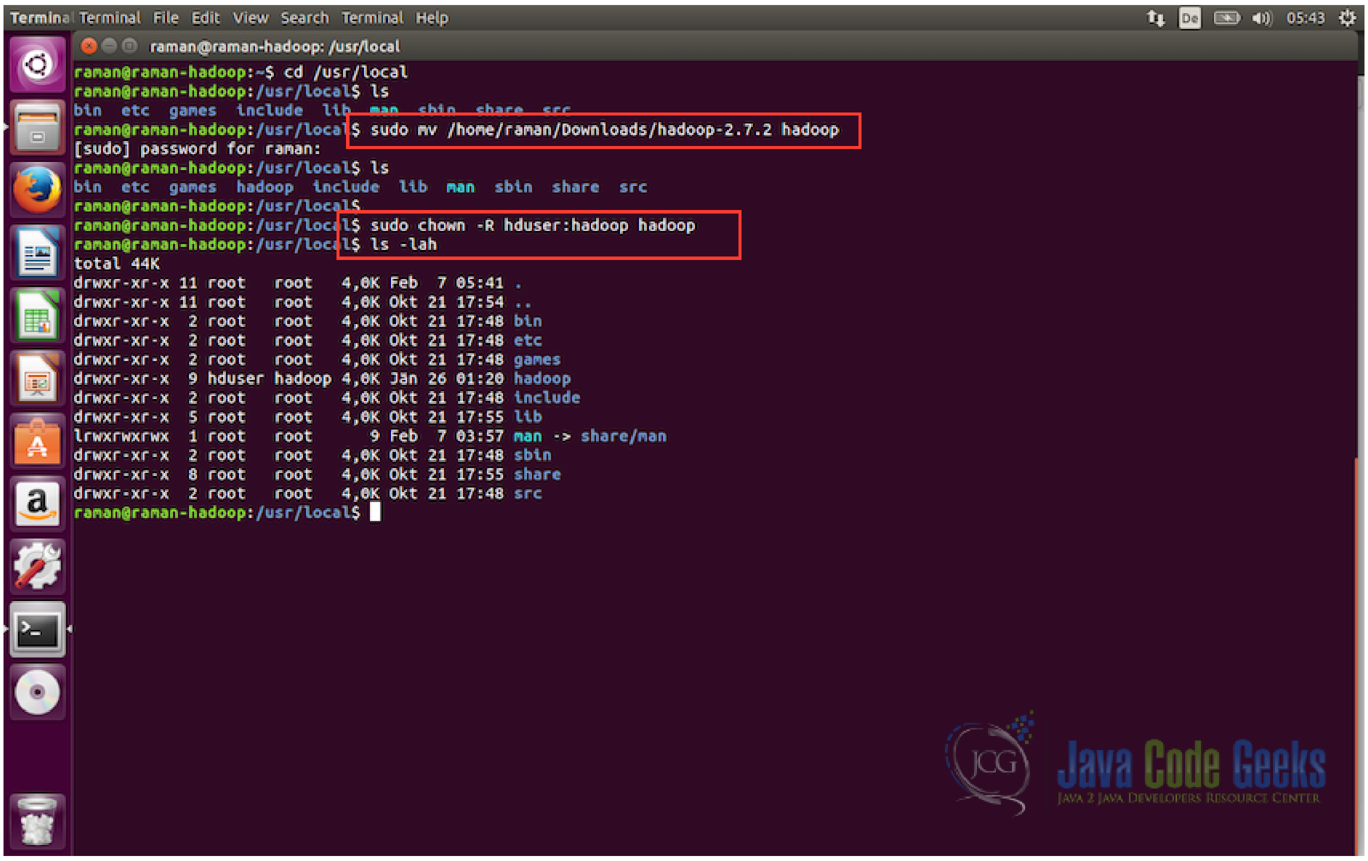
- Download Hadoop from Apache Mirrors at www.apache.org/dyn/closer.cgi/hadoop/core. It can be downloaded manually or using `wget` command.
- After download finishes, extract hadoop folder and move it to `/usr/local/hadoop` and finally change the owner of the folder to `hduser` and `hadoop` group.

```
#Change to the directory
cd /usr/local

#move hadoop files to the directory
sudo mv /home/hadoop1/Downloads/hadoop-2.7.1 hadoop

#change the permissions to the hduser user.
sudo chown -R hduser:hadoop
```

We can now check the permissions of the hadoop folder using the command:

```
ls -lah
```

This command shows the list of content in the `/usr/local/` directory along with the metadata. Hadoop fodler should have `hduser` as the owner and `hadoop` as the user group as shown in the screenshot below.



```
raman@raman-hadoop:~$ cd /usr/local
raman@raman-hadoop:/usr/local$ ls
bin  etc  games  include  lib  man  sbin  share  src
raman@raman-hadoop:/usr/local$ sudo mv /home/raman/Downloads/hadoop-2.7.2 hadoop
[sudo] password for raman:
raman@raman-hadoop:/usr/local$ ls
bin  etc  games  hadoop  include  lib  man  sbin  share  src
raman@raman-hadoop:/usr/local$
raman@raman-hadoop:/usr/local$ sudo chown -R hduser:hadoop hadoop
raman@raman-hadoop:/usr/local$ ls -lah
total 44K
drwxr-xr-x 11 root   root   4,0K Feb  7 05:41 .
drwxr-xr-x 11 root   root   4,0K Okt 21 17:54 ..
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 bin
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 etc
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 games
drwxr-xr-x  9 hduser hadoop 4,0K Jän 26 01:20 hadoop
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 include
drwxr-xr-x  5 root   root   4,0K Okt 21 17:55 lib
lrwxrwxrwx  1 root   root      9 Feb  7 03:57 man -> share/man
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 sbin
drwxr-xr-x  8 root   root   4,0K Okt 21 17:55 share
drwxr-xr-x  2 root   root   4,0K Okt 21 17:48 src
raman@raman-hadoop:/usr/local$
```

Figure 2.8: Placing hadoop in required folder and assigning dedicated user as owner of hadoop

## 2.3.2 Updating bash

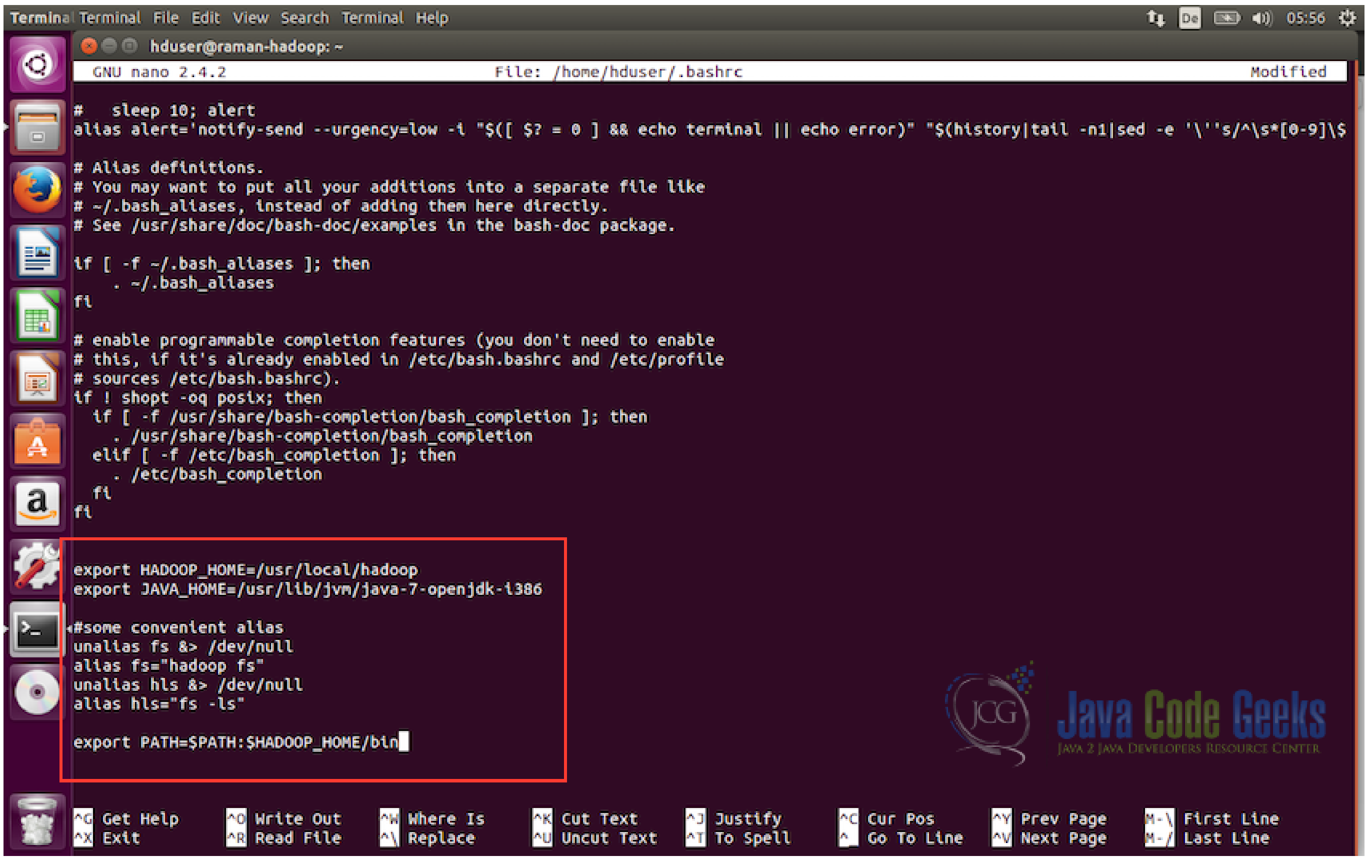
Update the `bashrc` file for the user **hduser**.

```
su - hduser
nano $HOME/.bashrc
```

At the end of the file, add the following lines.

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386

#Some convenient aliases
unalias fs &amp;> /dev/null
alias fs="hadoop fs"
unalias hls &amp;> /dev/null
alias hls="fs -ls"

export PATH=$PATH:$HADOOP_HOME/bin
```

The block of convenient aliases is optional and can be omitted. `JAVA_HOME`, `HADOOP_HOME` and `PATH` are the only compulsary requirements.



```
  GNU nano 2.4.2                File: /home/hduser/.bashrc                        Modified

#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\$

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi


export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386

#some convenient alias
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

export PATH=$PATH:$HADOOP_HOME/bin


^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

Figure 2.9: Updating .bashrc file

### 2.3.3 Configuring Hadoop

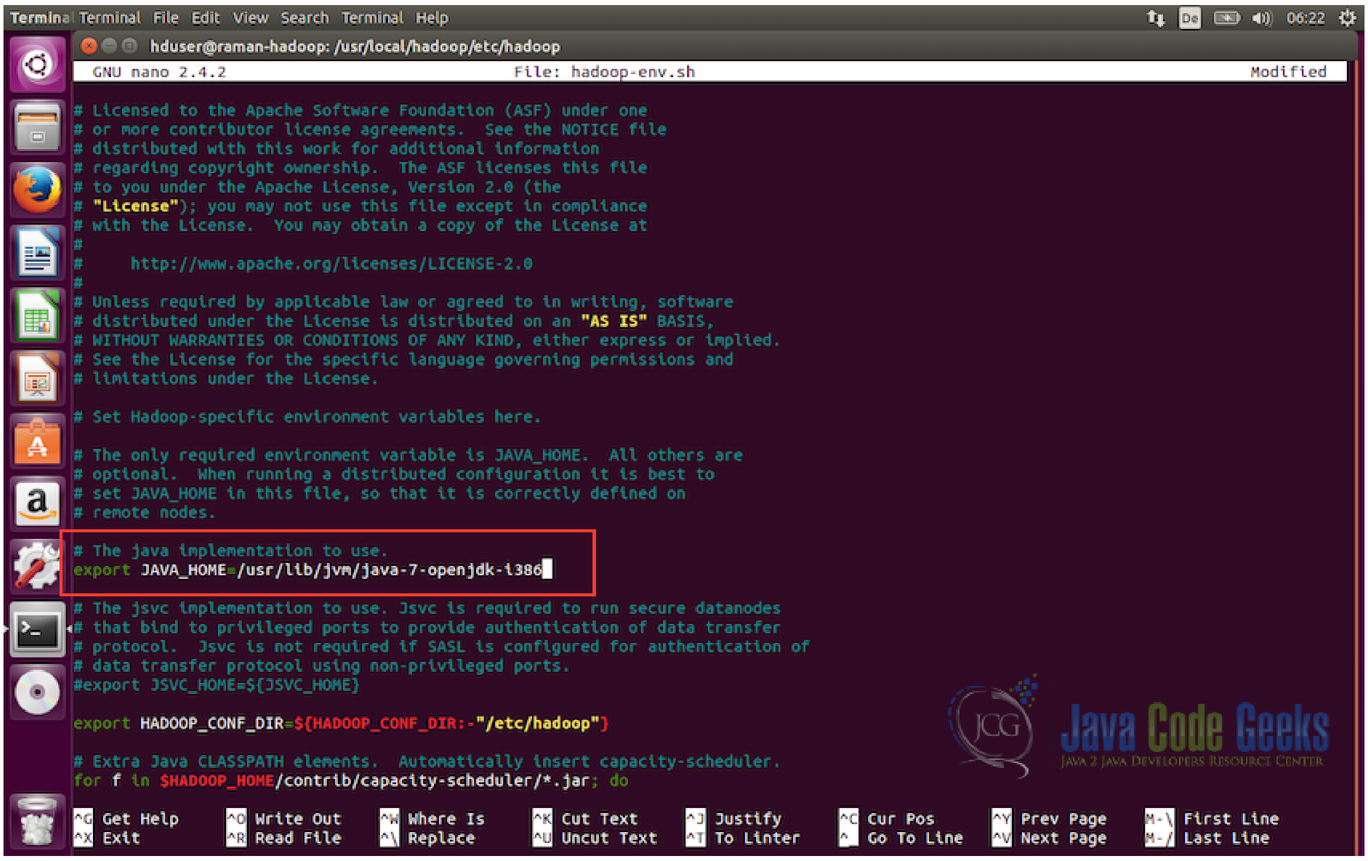
In this step, we will configure the Hadoop.

Open `hadoop-env.sh` in `/usr/local/hadoop/etc/hadoop/` and set the `JAVA_HOME` variable as shown below:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
```

and save the file using `ctrl+X` and then `Yes`.

**Note:** The path to java should be the path where the java is present in the system. By default it should be in the `/usr/lib` folder, but make sure it is the correct path as per your system. Also, make sure the version of java is correct which you want to use. Following screenshot shows where it need to be modified in the `hadoop-env.sh`.



```
GNU nano 2.4.2                    File: hadoop-env.sh                              Modified

# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}

# Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
```

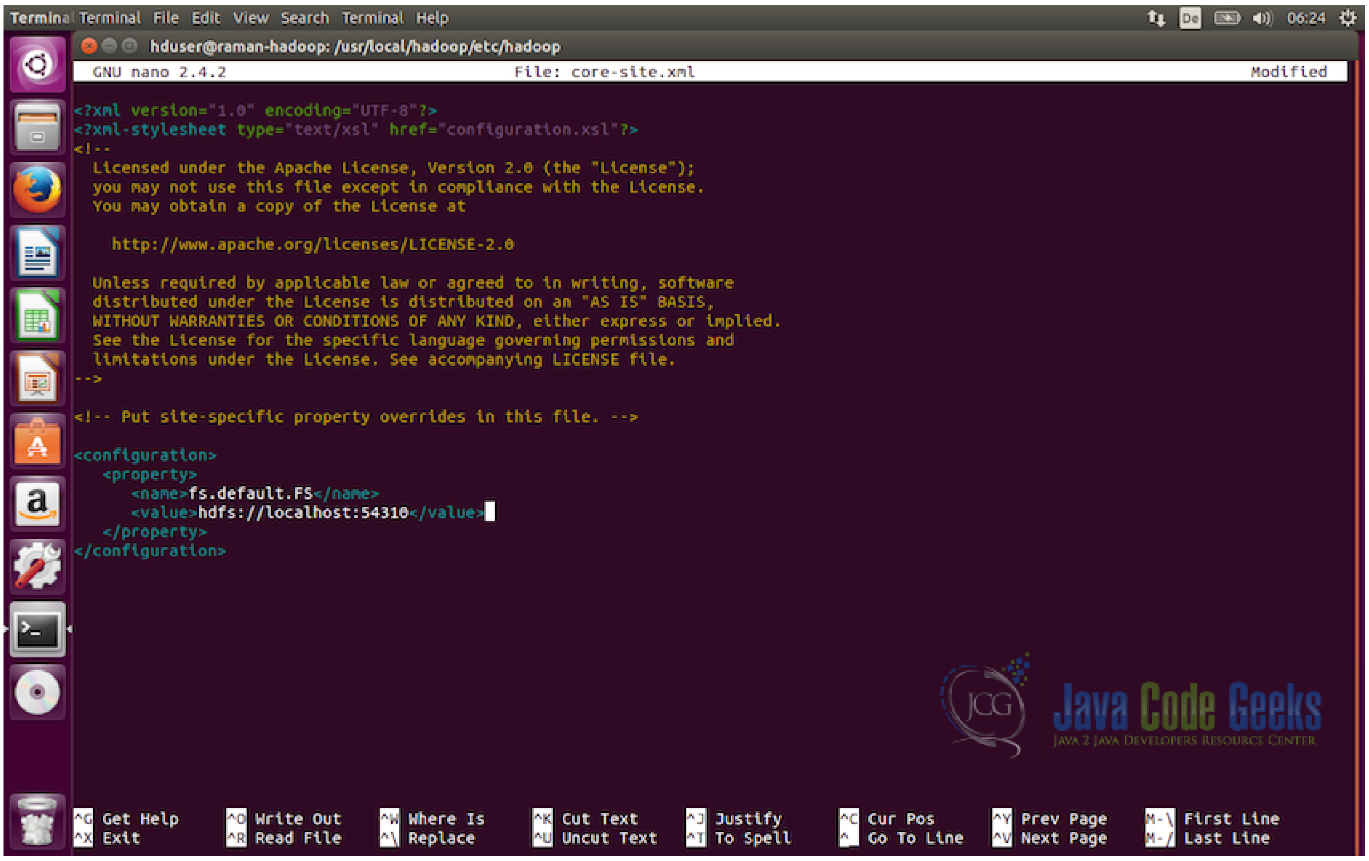
Terminal Terminal File Edit View Search Terminal Help
06:22
hduser@raman-hadoop: /usr/local/hadoop/etc/hadoop
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page M-\ First Line
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line ^V Next Page M-/ Last Line

Figure 2.10: Updating hadoop-env.sh file

Next, we will configure the `core-site.xml` in the folder `/usr/local/hadoop/etc/hadoop/` and add the following property

```
<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:54310</value>
   </property>
</configuration>
```

This tells the system where the default file system should be running on the system.

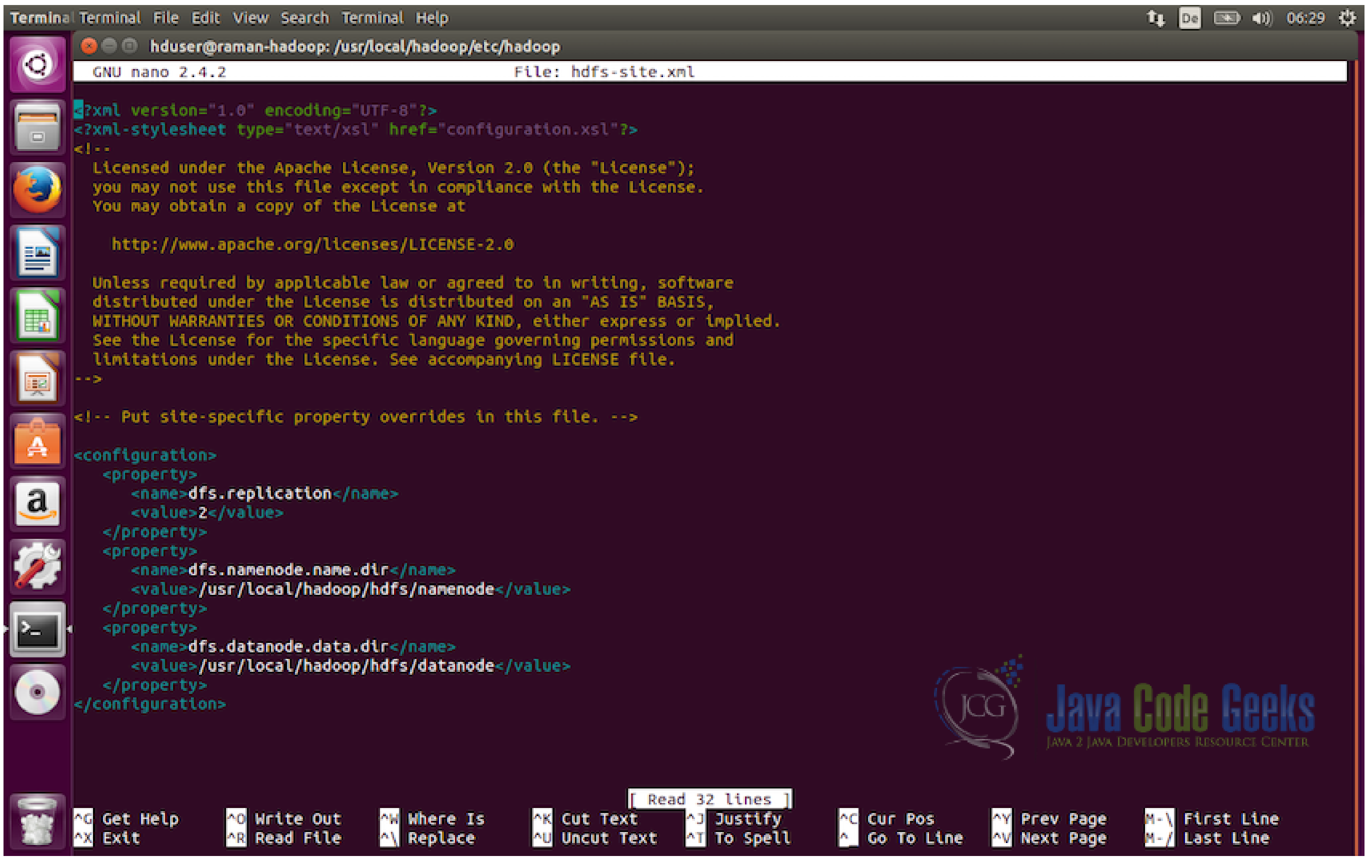

Terminal Terminal File Edit View Search Terminal Help
06:24
hduser@raman-hadoop: /usr/local/hadoop/etc/hadoop

```
  GNU nano 2.4.2                   File: core-site.xml                          Modified

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
      <name>fs.default.FS</name>
      <value>hdfs://localhost:54310</value>
   </property>
</configuration>

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

Figure 2.11: Updating core-site.xml

Next we need to update `hdfs-site.xml`. This file is used to specify the directories which will be used as the `namenode` and the `datanode`.

```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>2</value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>/usr/local/hadoop/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>/usr/local/hadoop/hdfs/datanode</value>
   </property>
</configuration>
```

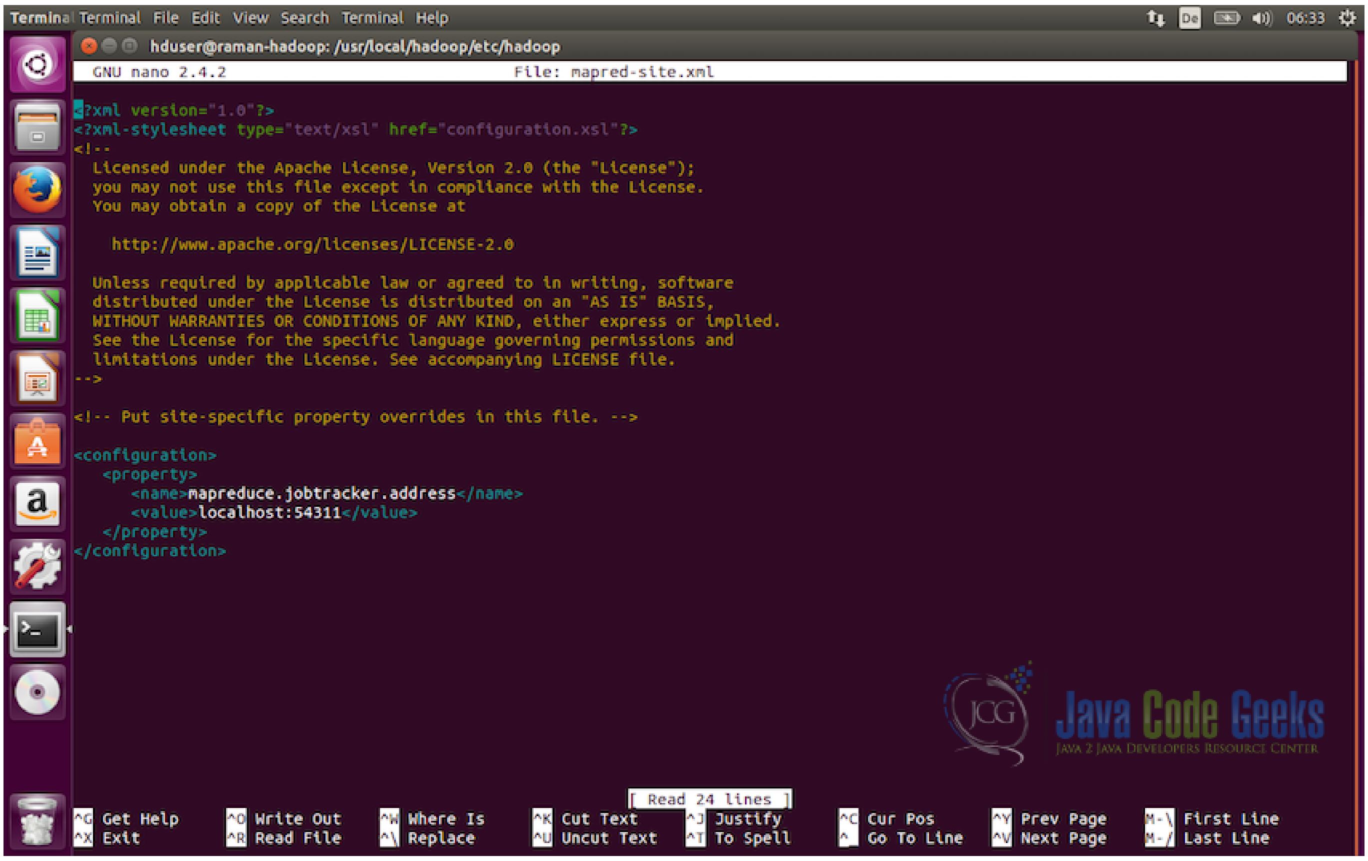

Terminal Terminal File Edit View Search Terminal Help
06:29
hduser@raman-hadoop: /usr/local/hadoop/etc/hadoop

```
  GNU nano 2.4.2                             File: hdfs-site.xml

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
      <name>dfs.replication</name>
      <value>2</value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>/usr/local/hadoop/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>/usr/local/hadoop/hdfs/datanode</value>
   </property>
</configuration>

                                          [ Read 32 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page   M-\ First Line
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  ^V Next Page   M-/ Last Line
```

Figure 2.12: Updating hdfs-site.xml

Now, we will update `mapred-site.xml` file. The folder `/usr/local/hadoop/etc/hadoop/` contains the file `mapred-site.xml.template`. Rename this file to `mapred-site.xml` before modification.

```
<configuration>
   <property>
      <name>mapreduce.jobtracker.address</name>
      <value>localhost:54311</value>
   </property>
</configuration>
```


Terminal Terminal File Edit View Search Terminal Help
06:33
hduser@raman-hadoop: /usr/local/hadoop/etc/hadoop

```
  GNU nano 2.4.2                          File: mapred-site.xml

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
      <name>mapreduce.jobtracker.address</name>
      <value>localhost:54311</value>
   </property>
</configuration>

                                       [ Read 24 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

Figure 2.13: Updating mapred-site.xml

### 2.3.4 Formatting the Hadoop Filesystem

We are now done with all the configuration, so before starting the cluster we need to format the namenode. To do so, use the following command on the terminal.

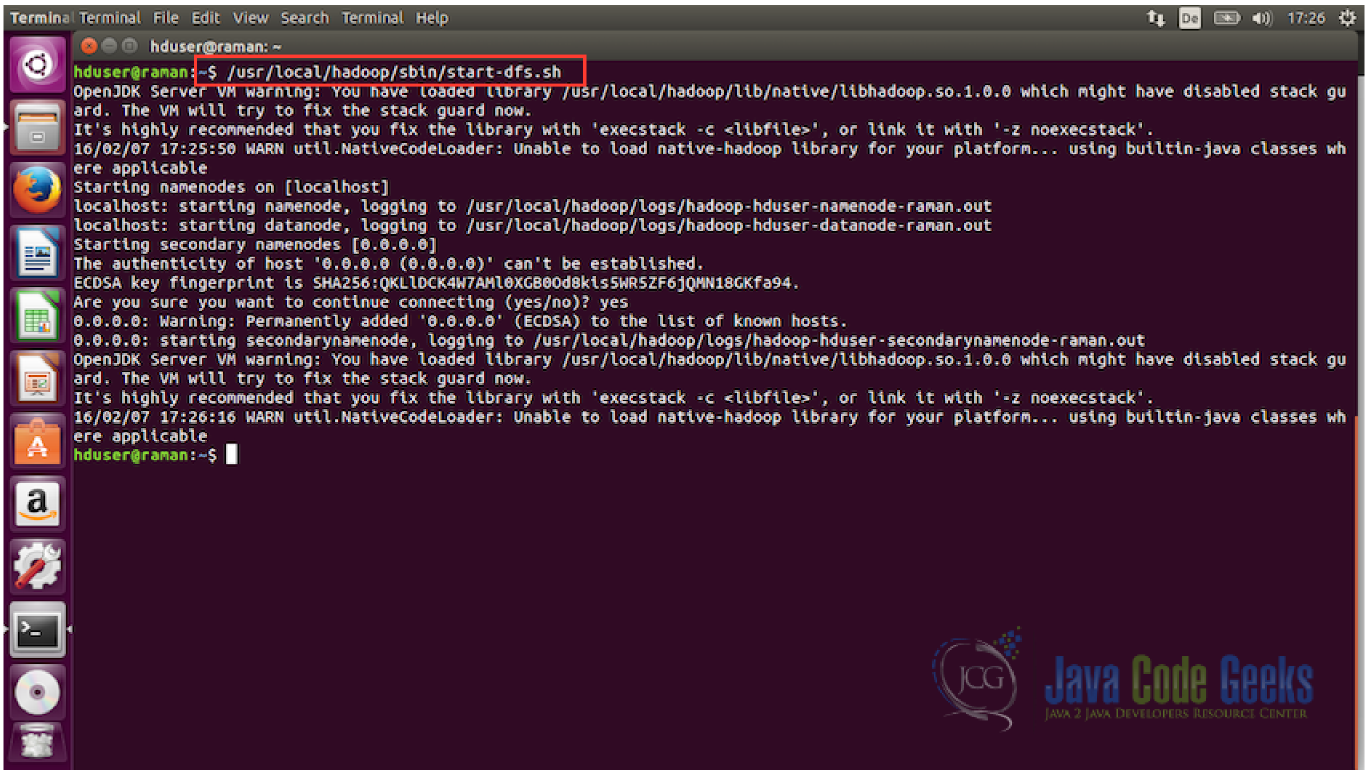
```
hdfs namenode -format
```

This command should be executed without any error on the console output. If it is executed without any errors, we are good to start the Apache Hadoop instance on our Ubuntu system.

### 2.3.5 Starting Apache Hadoop

Now it is time to start the Hadoop. Following is the command to do so:

```
/usr/local/hadoop/sbin/start-dfs.sh
```



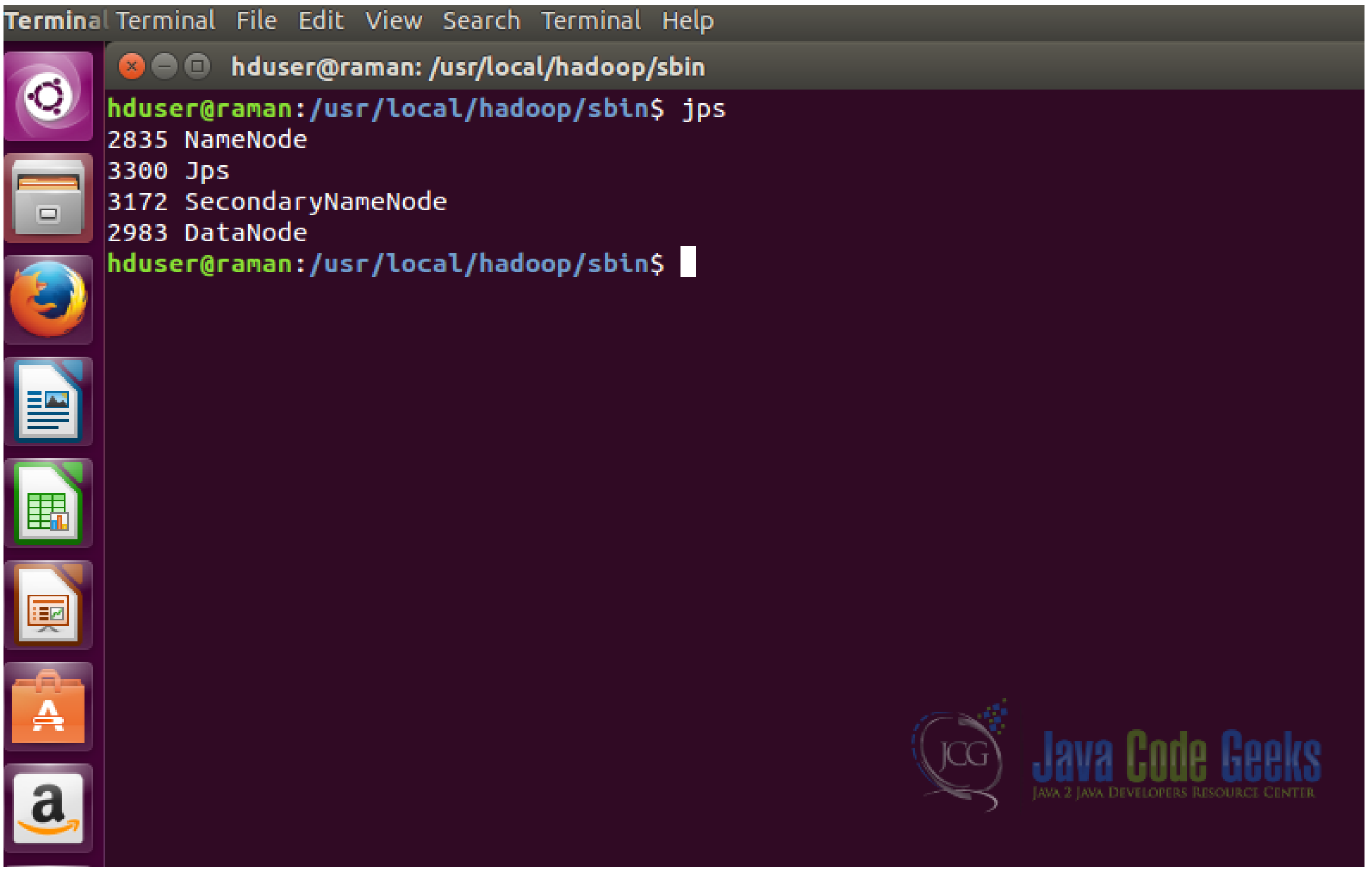
```
hduser@raman:~$ /usr/local/hadoop/sbin/start-dfs.sh
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 17:25:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-raman.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-raman.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:QKLlDCK4W7AMl0XGB0Od8kis5WR5ZF6jQMN18GKfa94.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-raman.out
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 17:26:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
hduser@raman:~$
```

Figure 2.14: Starting Hadoop

Once the dfs starts without any error, we can check if everything is working fine using the command `jps`

```
cd /usr/local/hadoop/sbin

#Checking the status of the Hadoop components
jps
```

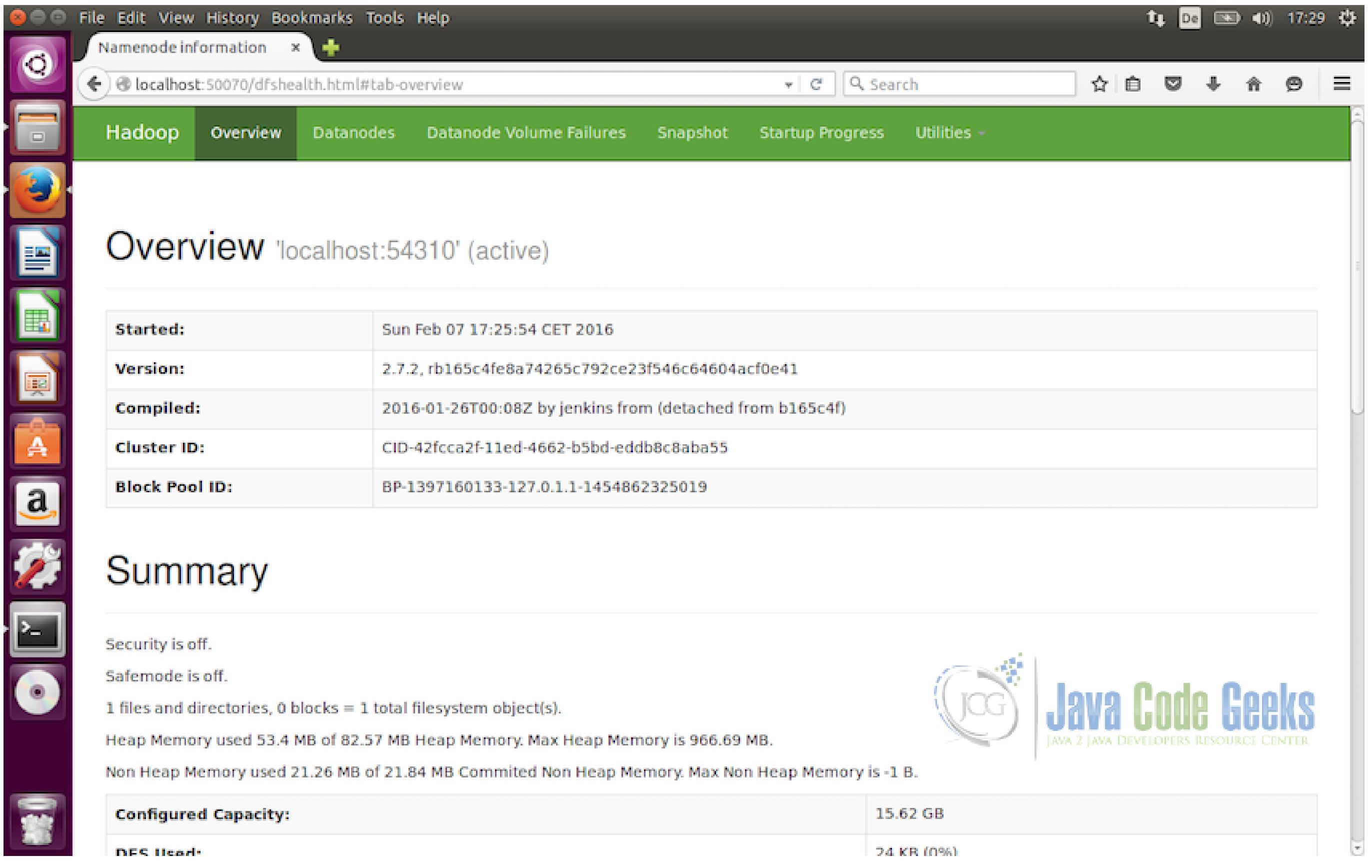
This command displays all the components of Hadoop which are running properly, we should see atleast a **Namenode** and a **Datanode** as shown in the screenshot below.


Terminal Terminal File Edit View Search Terminal Help
hduser@raman: /usr/local/hadoop/sbin

```
hduser@raman:/usr/local/hadoop/sbin$ jps
2835 NameNode
3300 Jps
3172 SecondaryNameNode
2983 DataNode
hduser@raman:/usr/local/hadoop/sbin$
```

Figure 2.15: jps command

Other options is to check the status of Apache Hadoop using the web interface for the Namenode on `https://localhost:50070`.



Overview 'localhost:54310' (active)

| | |
|---|---|
| **Started:** | Sun Feb 07 17:25:54 CET 2016 |
| **Version:** | 2.7.2, rb165c4fe8a74265c792ce23f546c64604acf0e41 |
| **Compiled:** | 2016-01-26T00:08Z by jenkins from (detached from b165c4f) |
| **Cluster ID:** | CID-42fcca2f-11ed-4662-b5bd-eddb8c8aba55 |
| **Block Pool ID:** | BP-1397160133-127.0.1.1-1454862325019 |

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

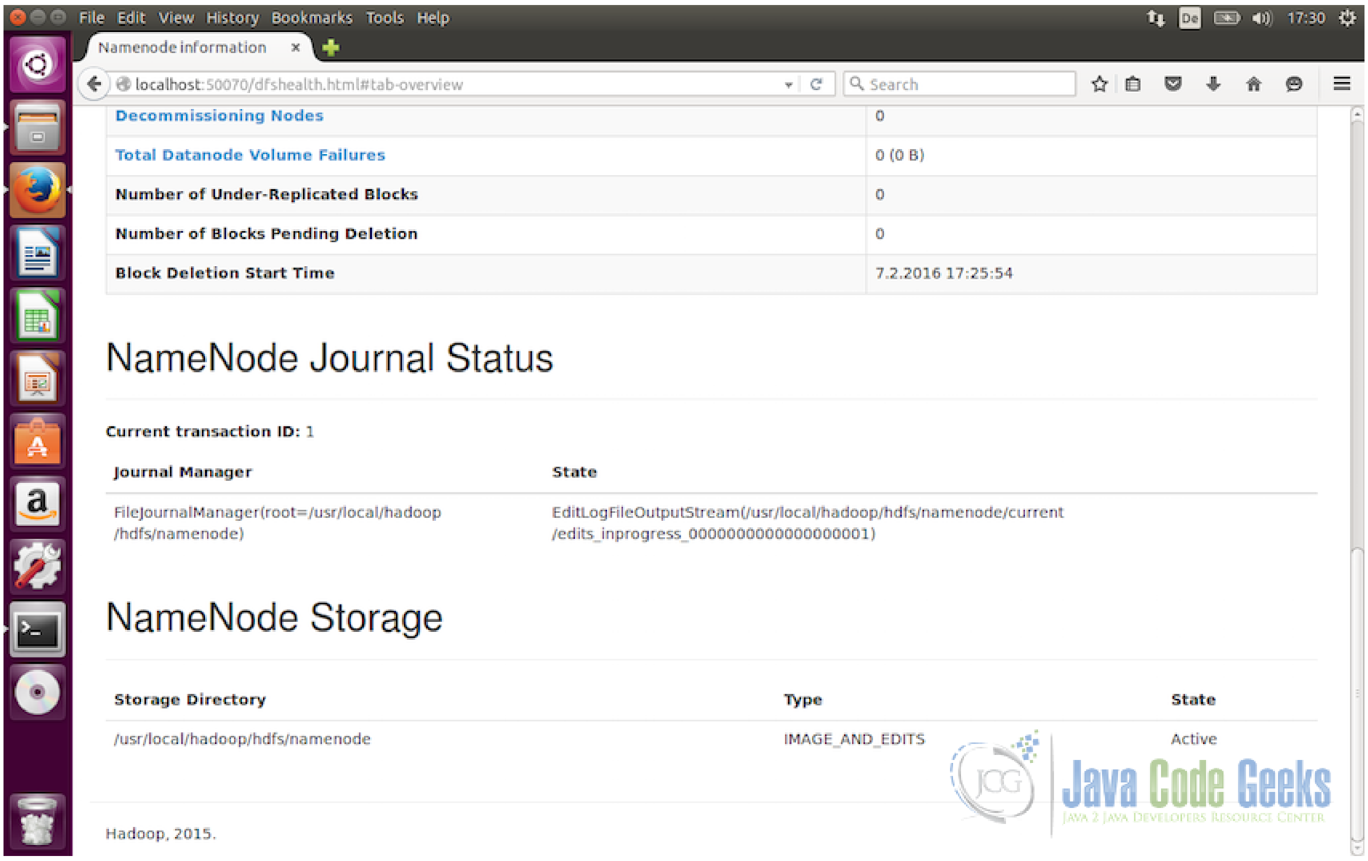
Heap Memory used 53.4 MB of 82.57 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 21.26 MB of 21.84 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

| | |
|---|---|
| **Configured Capacity:** | 15.62 GB |
| **DFS Used:** | 24 KB (0%) |

Figure 2.16: Apache Hadoop web interface

Following screenshot displays the details of Namenode in the web interface


File Edit View History Bookmarks Tools Help
Namenode information
localhost:50070/dfshealth.html#tab-overview
Search

| | |
|---|---|
| Decommissioning Nodes | 0 |
| Total Datanode Volume Failures | 0 (0 B) |
| Number of Under-Replicated Blocks | 0 |
| Number of Blocks Pending Deletion | 0 |
| Block Deletion Start Time | 7.2.2016 17:25:54 |

NameNode Journal Status

Current transaction ID: 1

| Journal Manager | State |
|---|---|
| FileJournalManager(root=/usr/local/hadoop/hdfs/namenode) | EditLogFileOutputStream(/usr/local/hadoop/hdfs/namenode/current/edits_inprogress_0000000000000000001) |

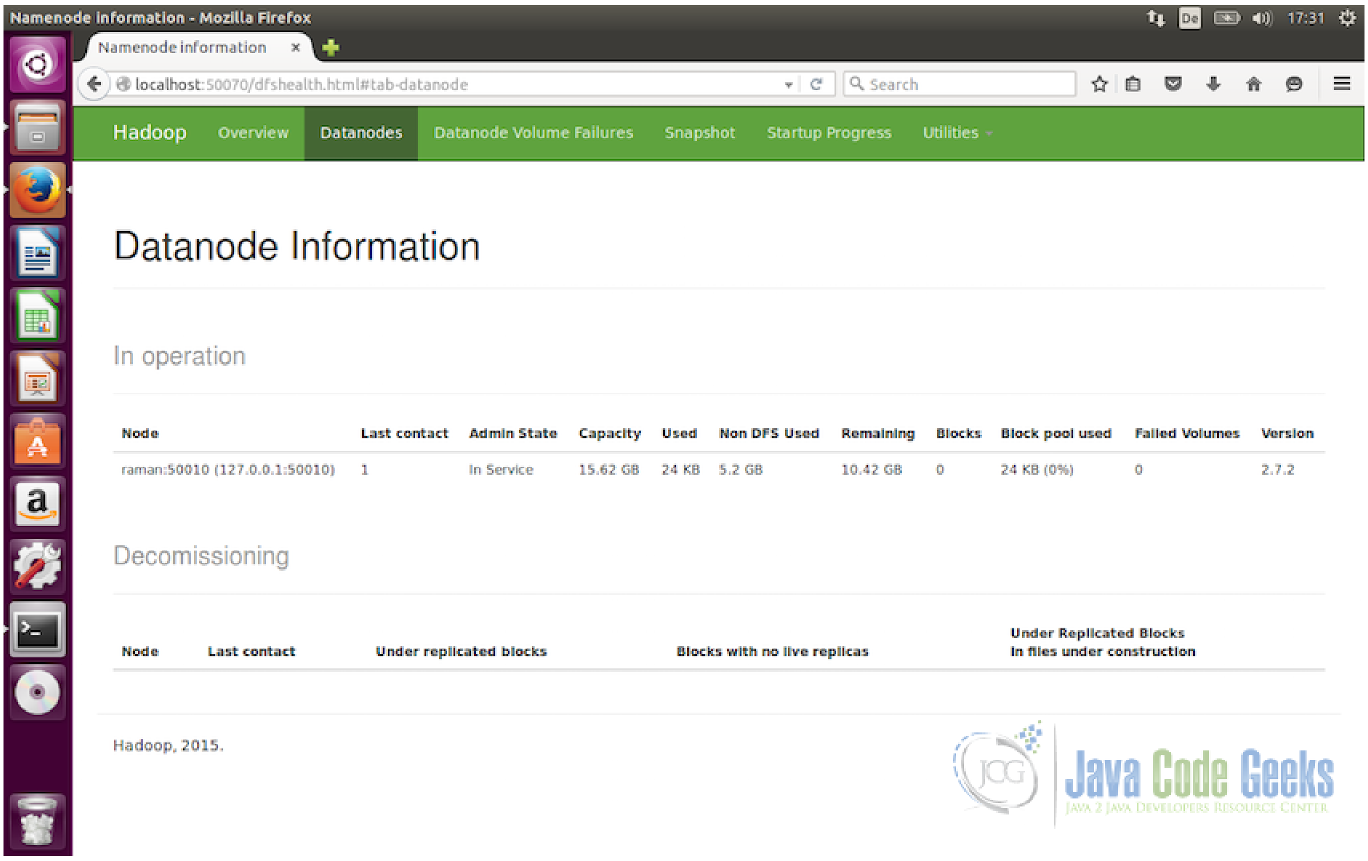
NameNode Storage

| Storage Directory | Type | State |
|---|---|---|
| /usr/local/hadoop/hdfs/namenode | IMAGE_AND_EDITS | Active |

Hadoop, 2015.

Figure 2.17: Namenode in Hadoop Web Interface

and the following screenshot shows the Datanode details in the Hadoop web interface



Datanode Information

In operation

| Node | Last contact | Admin State | Capacity | Used | Non DFS Used | Remaining | Blocks | Block pool used | Failed Volumes | Version |
|---|---|---|---|---|---|---|---|---|---|---|
| raman:50010 (127.0.0.1:50010) | 1 | In Service | 15.62 GB | 24 KB | 5.2 GB | 10.42 GB | 0 | 24 KB (0%) | 0 | 2.7.2 |

Decomissioning

| Node | Last contact | Under replicated blocks | Blocks with no live replicas | Under Replicated Blocks In files under construction |
|---|---|---|---|---|

Hadoop, 2015.

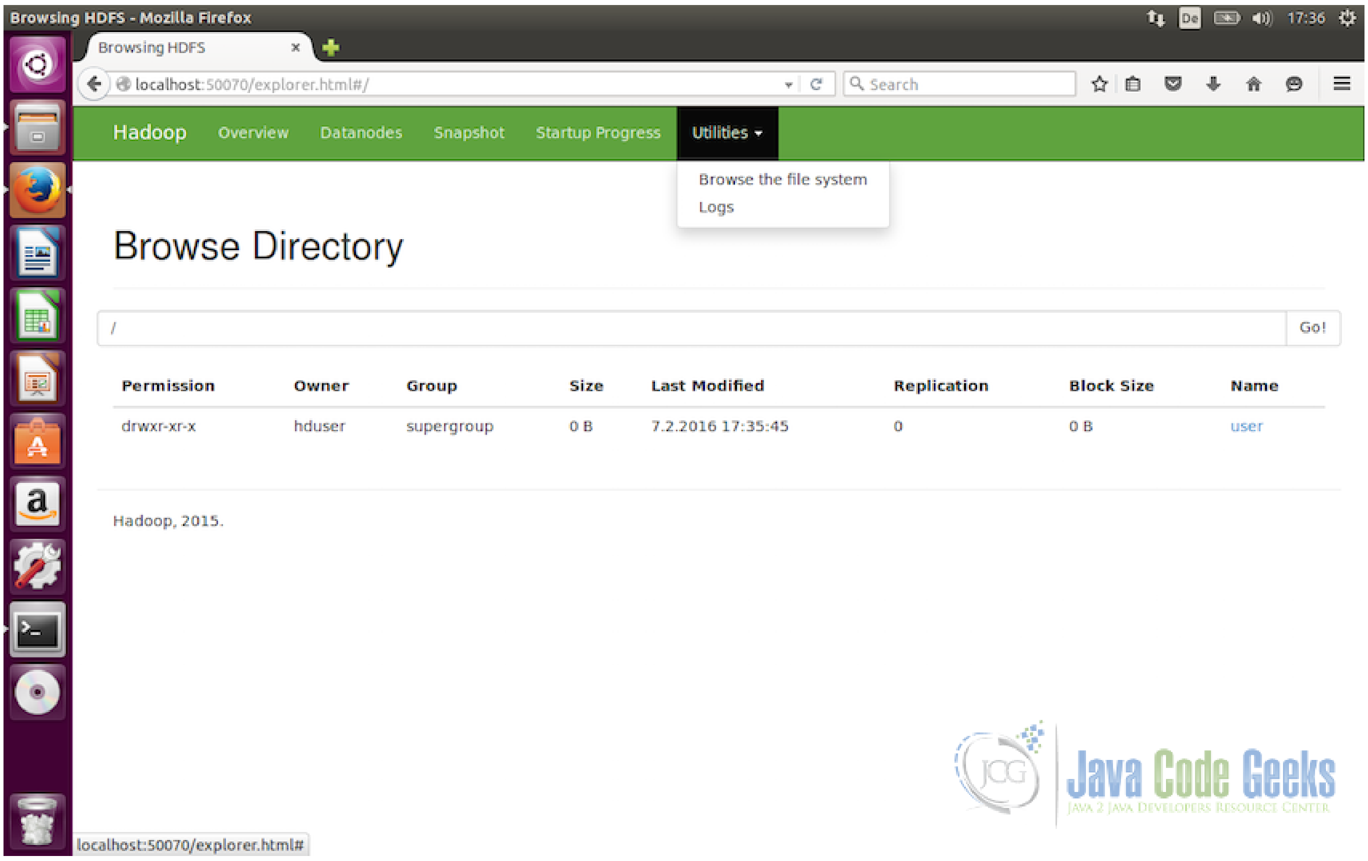
Figure 2.18: Datanode in Hadoop Web Interface

### 2.3.6 Testing MapReduce Job

First of all, lets make the required HDFS directories and copy some input data for testing purpose

```
#Make the required directories in HDFS
bin/hdfs dfs -mkdir /user
bin/hdfs dfs -mkdir /user/hduser
```

These directories can be accessed from the web interface also. To do so, go to the web interface, from the menu select ‘Utilities’ and from dropdown select ‘Browse the file system’



Browse Directory

/

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:35:45 | 0 | 0 B | user |

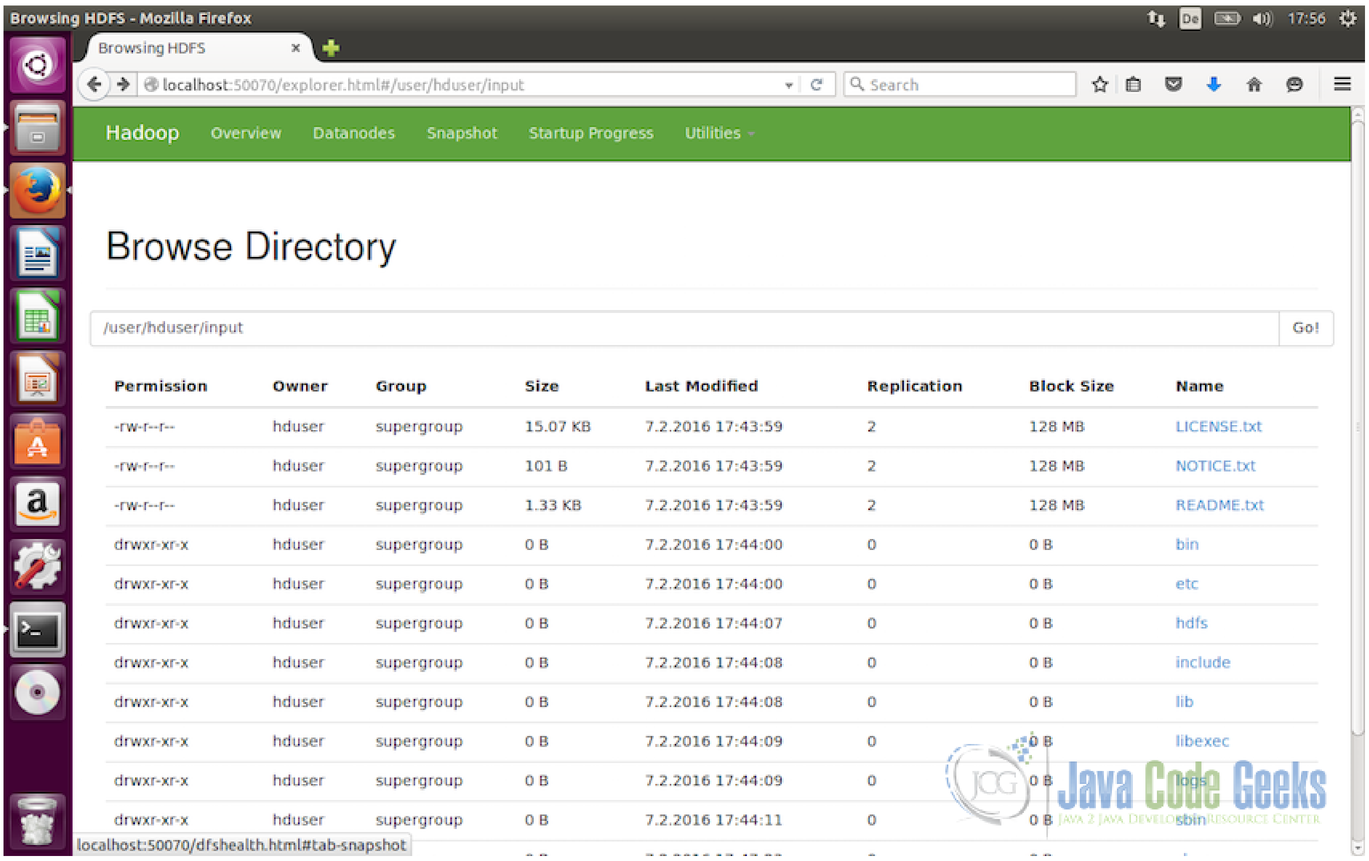
Hadoop, 2015.

Figure 2.19: Browse HDFS File System

Now, we can add some dummy files to the directory which we will use for the testing purpose. Let us pass the all the files from `etc/hadoop` folder.

```
#Copy the input files into the distributed file system
/usr/local/hadoop/bin/hdfs dfs -put /usr/local/hadoop/etc/hadoop input
```

Following screenshot shows the files added to the directories `/user/hduser/input` in the web interface


Browsing HDFS - Mozilla Firefox
Browsing HDFS
localhost:50070/explorer.html#/user/hduser/input
Search
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Go!
localhost:50070/dfshealth.html#tab-snapshot

Browse Directory

/user/hduser/input

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| -rw-r--r-- | hduser | supergroup | 15.07 KB | 7.2.2016 17:43:59 | 2 | 128 MB | LICENSE.txt |
| -rw-r--r-- | hduser | supergroup | 101 B | 7.2.2016 17:43:59 | 2 | 128 MB | NOTICE.txt |
| -rw-r--r-- | hduser | supergroup | 1.33 KB | 7.2.2016 17:43:59 | 2 | 128 MB | README.txt |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:00 | 0 | 0 B | bin |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:00 | 0 | 0 B | etc |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:07 | 0 | 0 B | hdfs |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:08 | 0 | 0 B | include |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:08 | 0 | 0 B | lib |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:09 | 0 | 0 B | libexec |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:09 | 0 | 0 B | logs |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:11 | 0 | 0 B | sbin |

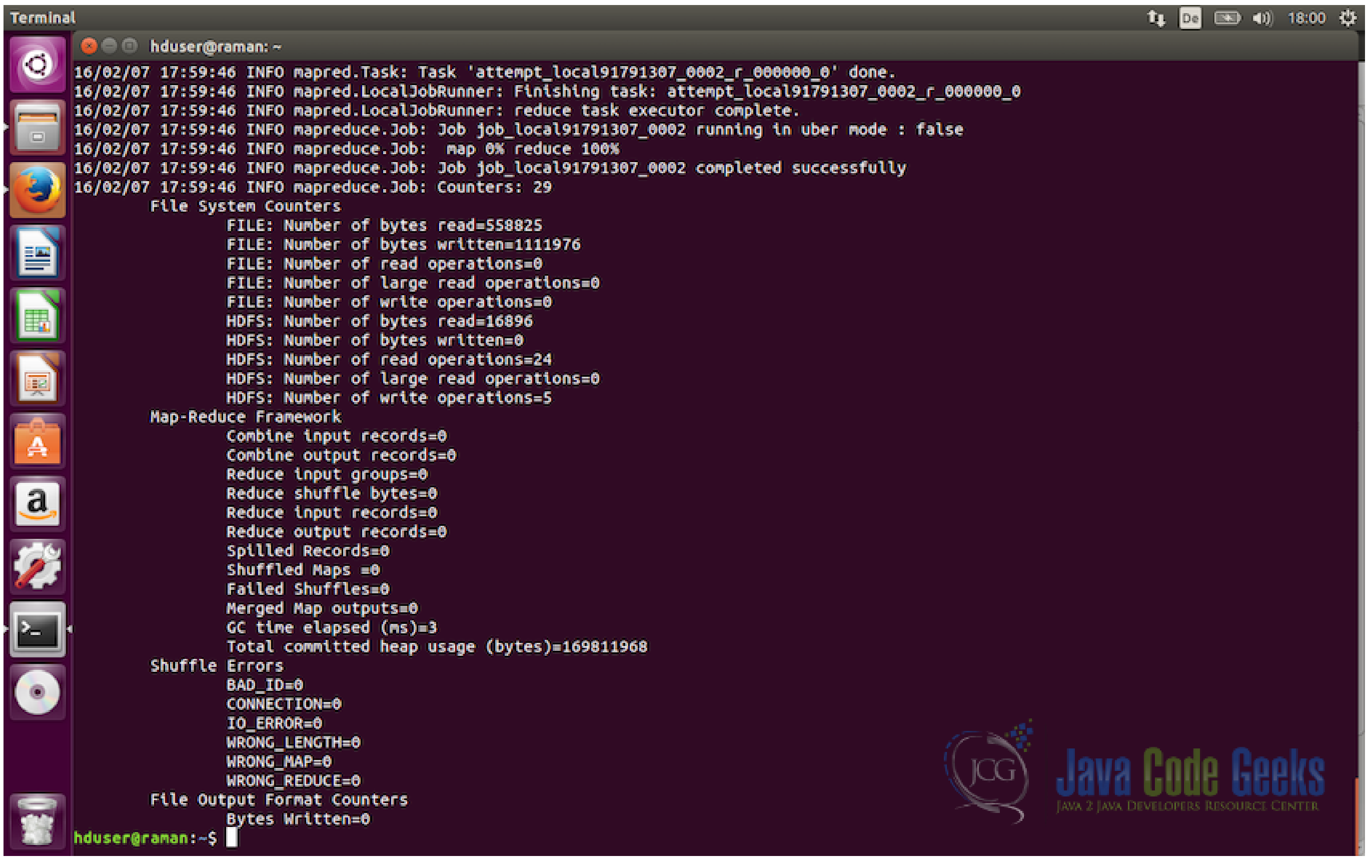
Figure 2.20: Browse HDFS File System

Run the MapReduce example job included in the Hadoop package using the following command:

```
/usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce- ↩
    example-2.7.1.jar grep input output 'dfs[a-z.]+'
```

**Note:** For details on how MapReduce example works, refer to the article "Hadoop Hello World Example"

Following screenshot shows the output log of the test example:



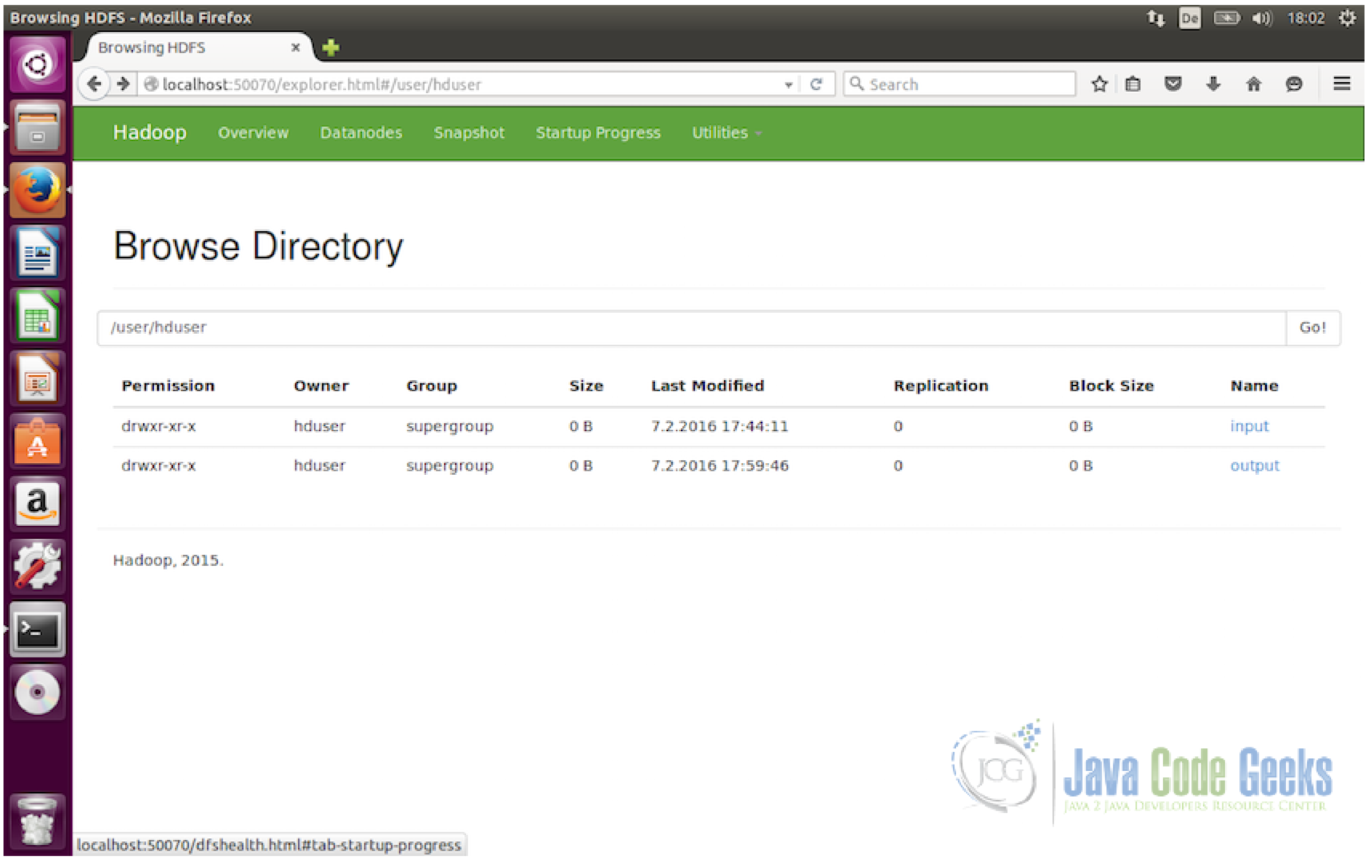
```
hduser@raman: ~
16/02/07 17:59:46 INFO mapred.Task: Task 'attempt_local91791307_0002_r_000000_0' done.
16/02/07 17:59:46 INFO mapred.LocalJobRunner: Finishing task: attempt_local91791307_0002_r_000000_0
16/02/07 17:59:46 INFO mapred.LocalJobRunner: reduce task executor complete.
16/02/07 17:59:46 INFO mapreduce.Job: Job job_local91791307_0002 running in uber mode : false
16/02/07 17:59:46 INFO mapreduce.Job:  map 0% reduce 100%
16/02/07 17:59:46 INFO mapreduce.Job: Job job_local91791307_0002 completed successfully
16/02/07 17:59:46 INFO mapreduce.Job: Counters: 29
        File System Counters
                FILE: Number of bytes read=558825
                FILE: Number of bytes written=1111976
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=16896
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=5
        Map-Reduce Framework
                Combine input records=0
                Combine output records=0
                Reduce input groups=0
                Reduce shuffle bytes=0
                Reduce input records=0
                Reduce output records=0
                Spilled Records=0
                Shuffled Maps =0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=3
                Total committed heap usage (bytes)=169811968
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=0
hduser@raman:~$
```

Figure 2.21: Wordcount example console output

We can now view the output file using the command

```
/usr/local/hadoop/bin/hdfs dfs -cat output/*
```

or using the web interface also as displayed in the screenshot below:


Browsing HDFS - Mozilla Firefox
Browsing HDFS
localhost:50070/explorer.html#/user/hduser
Search
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Go!
localhost:50070/dfshealth.html#tab-startup-progress
Java Code Geeks

Browse Directory

/user/hduser

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:44:11 | 0 | 0 B | input |
| drwxr-xr-x | hduser | supergroup | 0 B | 7.2.2016 17:59:46 | 0 | 0 B | output |

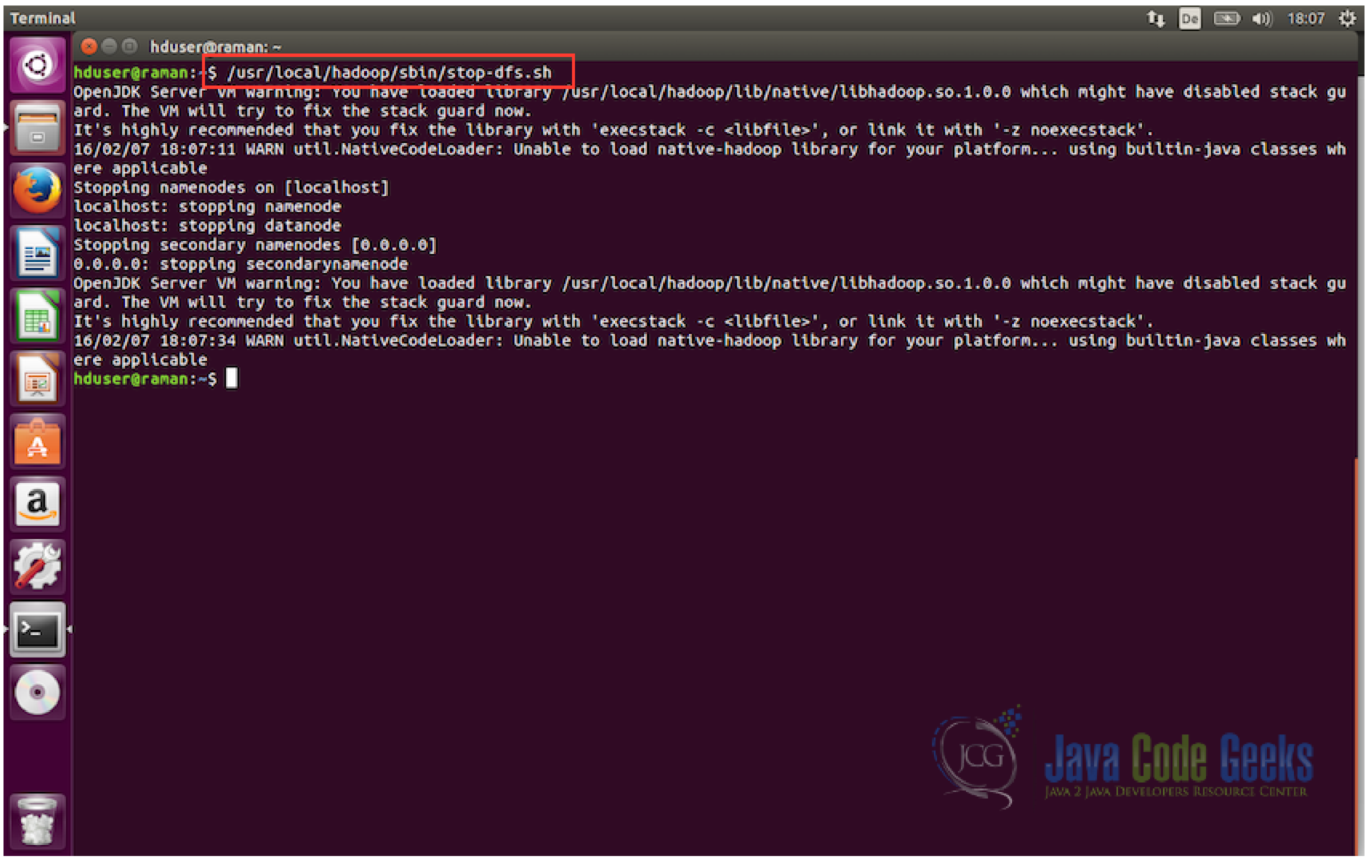
Hadoop, 2015.

Figure 2.22: Output folder in hdfs

### 2.3.7 Stopping Apache Hadoop

We can now stop the dfs(distributed format system) using the following command:

```
/usr/local/hadoop/sbin/stop-dfs.sh
```



```
hduser@raman:~$ /usr/local/hadoop/sbin/stop-dfs.sh
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 18:07:11 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 18:07:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
hduser@raman:~$
```

Figure 2.23: Stopping Apache Hadoop

## 2.4 Conclusion

This brings us to the end of the example. By now, we have Apache Hadoop Installed on our Ubuntu system and we know how to add data to the Hadoop and how to execute the job on the added data. After this, you can play around with Hadoop. You may also like to follow the example https://examples.javacodegeeks.com/enterprise-java/apache-hadoop/apache-hadoop-fs-commands-example/ to know some of the common Hadoop File System commands.

# Chapter 3

# FS Commands Example

In this example, we will go through most important commands which you may need to know to handle Hadoop File System(FS).

We assume the previous knowledge of what Hadoop is and what Hadoop can do? How it works in distributed fashion and what Hadoop Distributed File System(HDFS) is? So that we can go ahead and check some examples of how to deal with the Hadoop File System and what are some of the most important commands. Following are two examples which can help you if you are not well aware about Apache Hadoop:

- Hadoop "Hello World" Example
- How to set up Hadoop Cluster using Virtual Machines

Let us get started, as said in this example we will see top and the most frequently used Hadoop File System(fs) commands which will be useful to manage files and data in HDFS clusters.

## 3.1 Introduction

The Hadoop File System(FS) provides various shell like commands by default which can be used to interact with the Hadoop Distributed File System(HDFS) or any other supported file system using the Hadoop Shell. Some of the most common commands are the once used for operations like creating directories, copying a file, viewing the file content, changing ownership or permissions on the file.

## 3.2 Common Commands

In this section, we will see the usage and the example of most common Hadoop FS Commands.

### 3.2.1 Create a directory

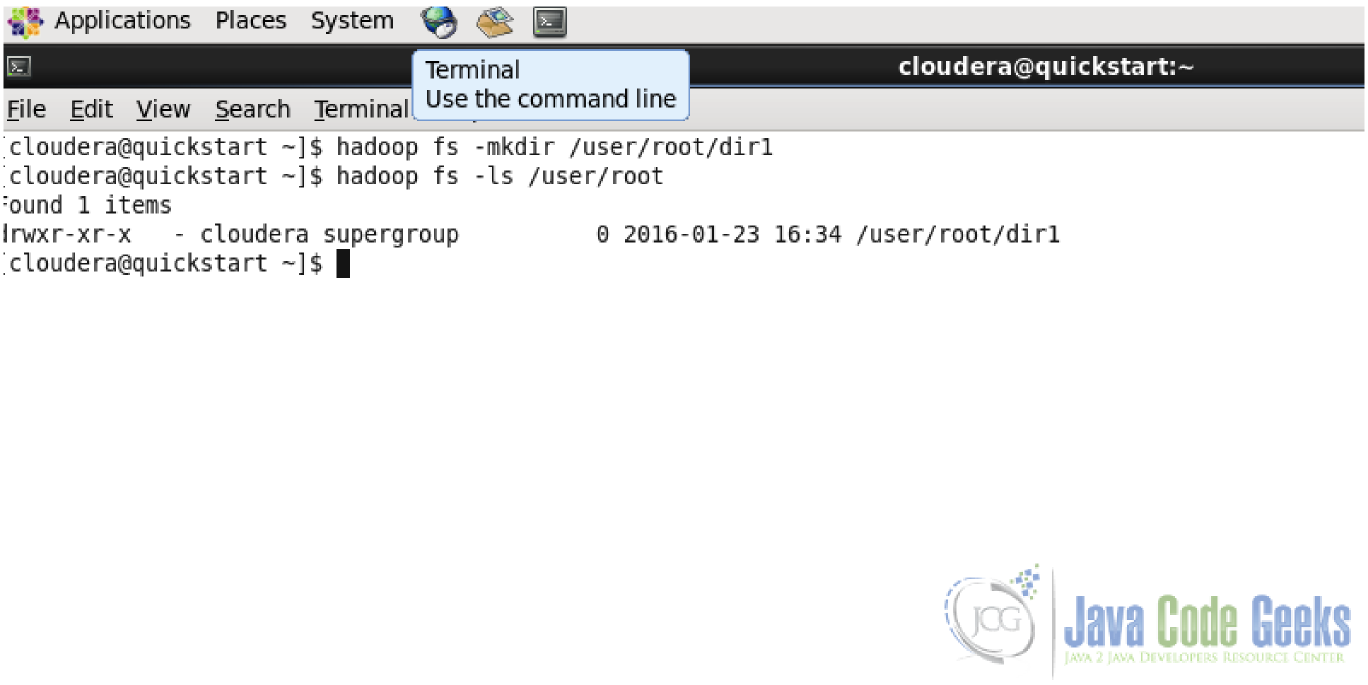
Usage:

```
hadoop fs -mkdir <paths>
```

Example:

```
hadoop fs -mkdir /user/root/dir1
```

Command in the second line is for listing the content of a particular path. We will see this command in the next sub-section. We can see in the screenshot that `dir1` is created



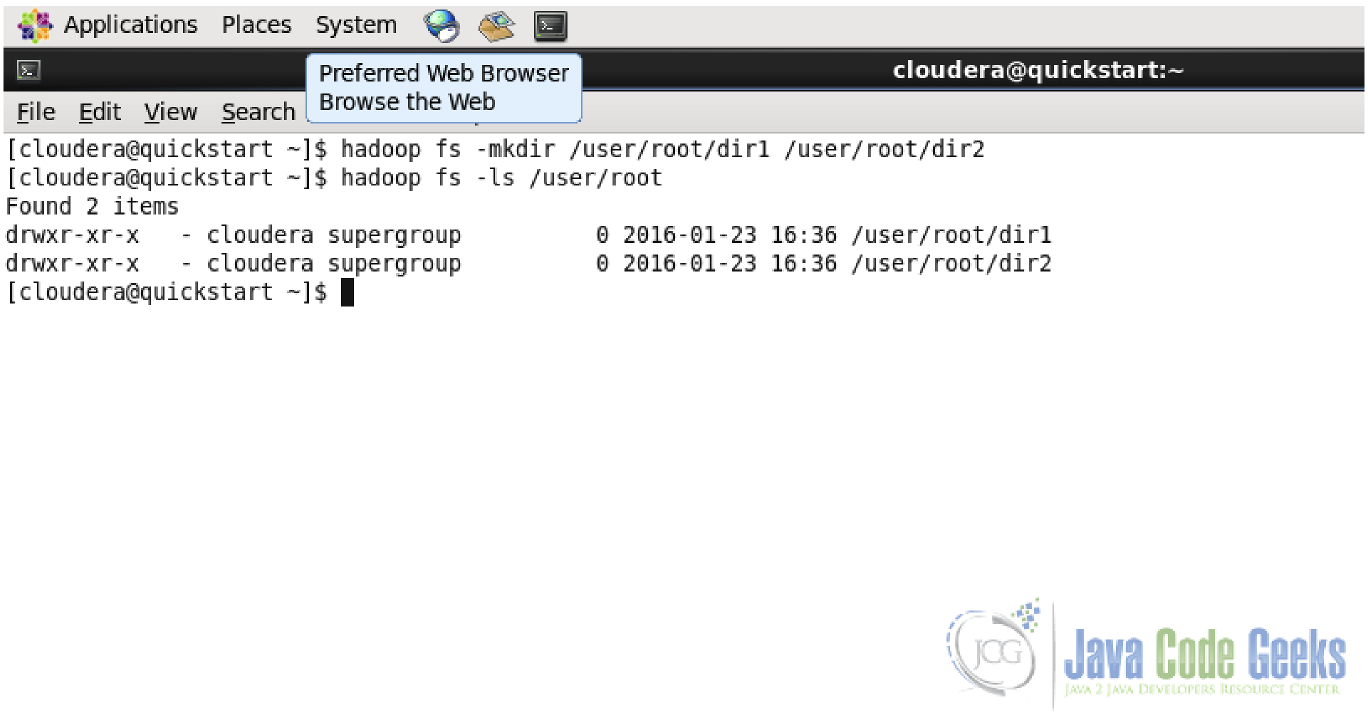
```
[cloudera@quickstart ~]$ hadoop fs -mkdir /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root
Found 1 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:34 /user/root/dir1
[cloudera@quickstart ~]$
```

Figure 3.1: Create Directory in Hadoop FS

Creating multiple directories with single command

```
hadoop fs -mkdir /user/root/dir1 /user/root/dir2
```

As shown in the above example, to create multiple directories in one go just pass multiple path and directory names separated by space.



```
[cloudera@quickstart ~]$ hadoop fs -mkdir /user/root/dir1 /user/root/dir2
[cloudera@quickstart ~]$ hadoop fs -ls /user/root
Found 2 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir1
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir2
[cloudera@quickstart ~]$
```

Figure 3.2: Make multiple directories with single command

### 3.2.2 List the content of the directory

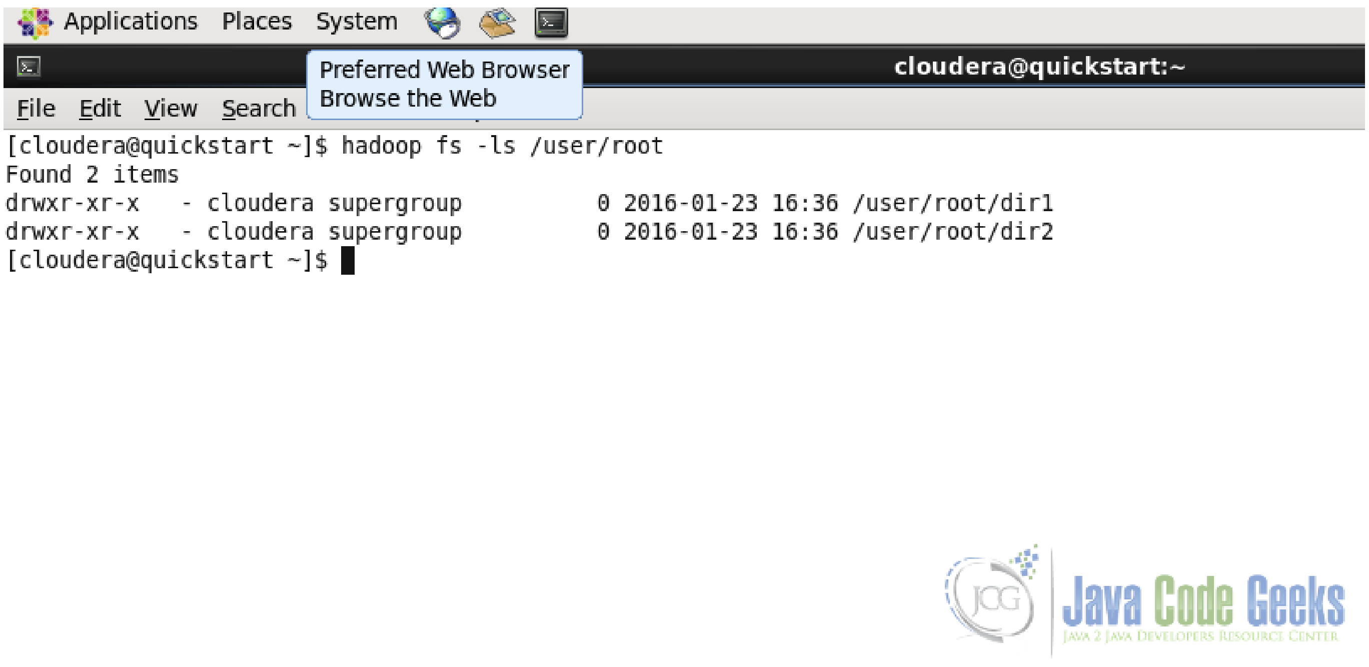
Usage:

```
hadoop fs -ls <paths>
```

Example:

```
hadoop fs -ls /user/root/
```

The command is similar to the `ls` command of the unix shell.



```
[cloudera@quickstart ~]$ hadoop fs -ls /user/root
Found 2 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir1
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir2
[cloudera@quickstart ~]$ 
```

Figure 3.3: Listing the files and directories

### 3.2.3 Upload a file in HDFS

Command is used to copy one or multiple files from local system to the Hadoop File System.

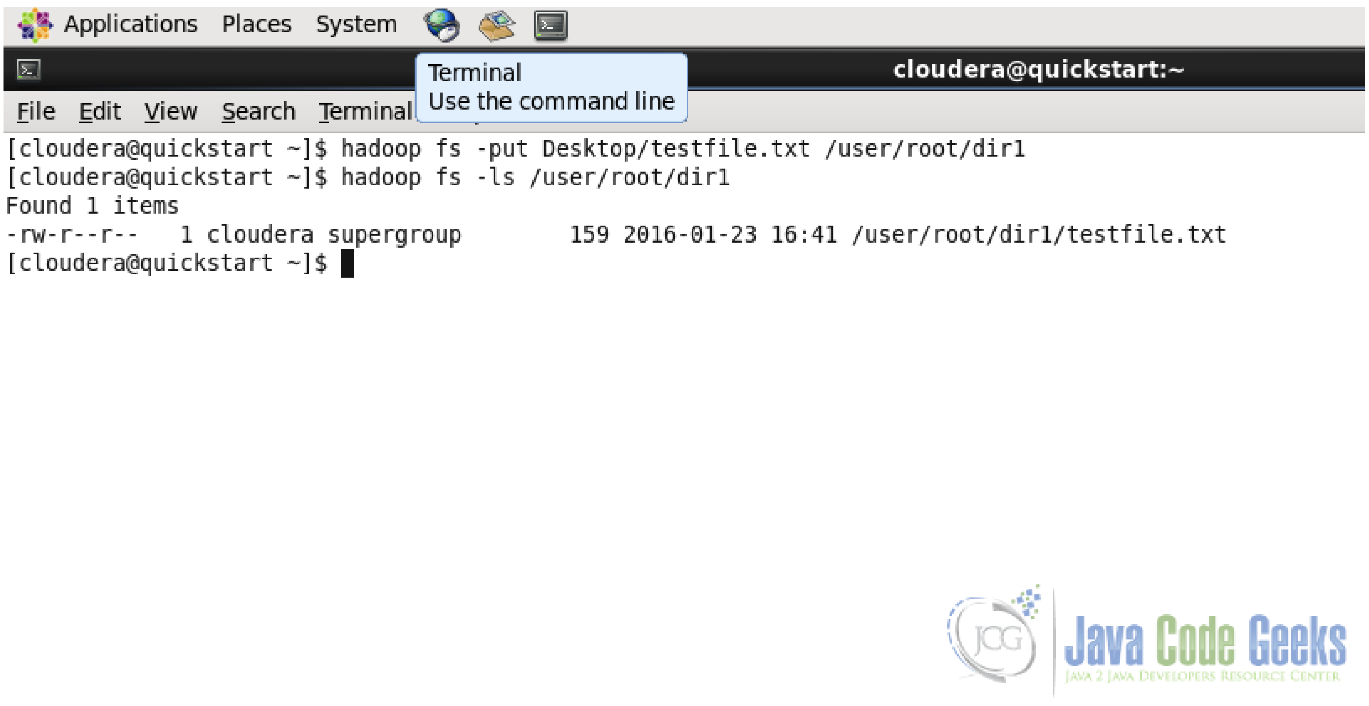
Usage:

```
hadoop fs -put <local_files> ... <hdfs_path>
```

Example:

```
hadoop fs -put Desktop/testfile.txt /user/root/dir1/
```

In the screenshot below, we `put` the file `testfile.txt` from `Desktop` of the Local File System to the Hadoop File System at the destiantion `/user/root/dir1`



```
[cloudera@quickstart ~]$ hadoop fs -put Desktop/testfile.txt /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir1
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:41 /user/root/dir1/testfile.txt
[cloudera@quickstart ~]$
```

Figure 3.4: Uploading the file to Hadoop FS

### 3.2.4 Download a file from HDFS

Download the file from HDFS to the local file system.

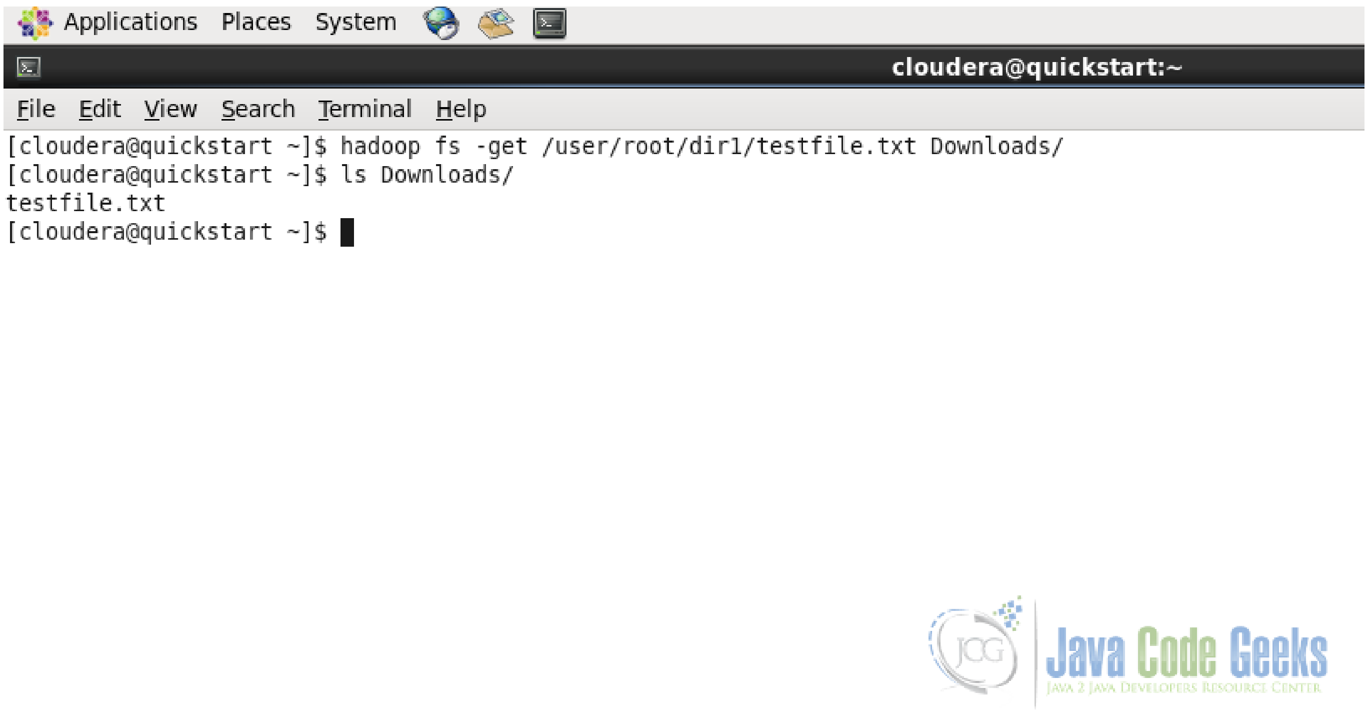
Usage:

```
hadoop fs -get <hdfs_paths> <local_path>
```

Example:

```
hadoop fs -get /user/root/dir1/testfile.txt Downloads/
```

As with the `put` command, `get` command gets or downloads the file from Hadoop File System to the Local File System in the `Downloads` folder.


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

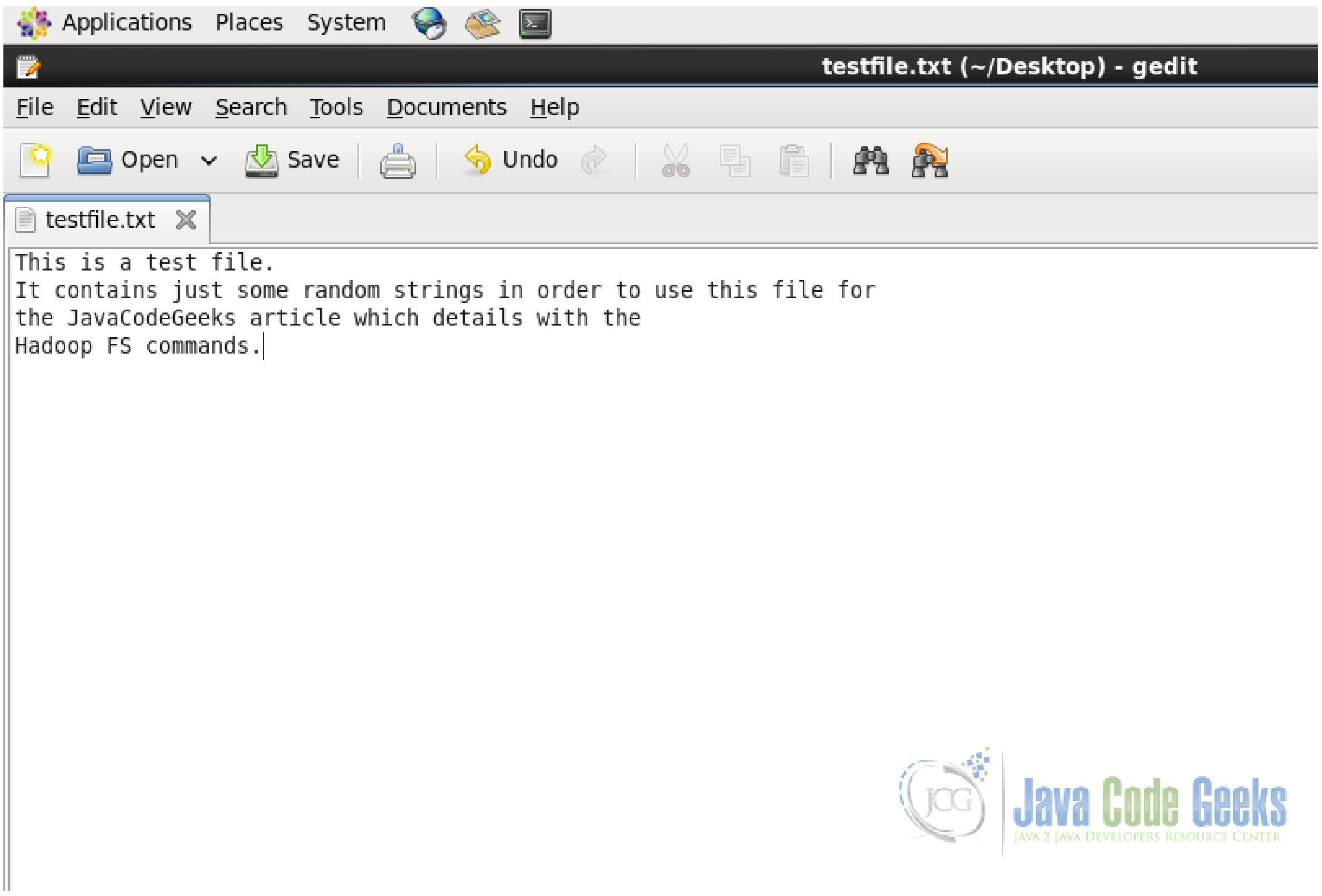
```
[cloudera@quickstart ~]$ hadoop fs -get /user/root/dir1/testfile.txt Downloads/
[cloudera@quickstart ~]$ ls Downloads/
testfile.txt
[cloudera@quickstart ~]$ 
```

Figure 3.5: Download the file from Hadoop FS

### 3.2.5 View the file content

For viewing the content of the file, `cat` command is available in the Hadoop File System. It is again similar to the one available in the unix shell.

Following is the content of the file which is uploaded to the Hadoop file system at the path `/user/root/dir1/` in the previous steps.



```
This is a test file.
It contains just some random strings in order to use this file for
the JavaCodeGeeks article which details with the
Hadoop FS commands.
```

Figure 3.6: Testfile.txt

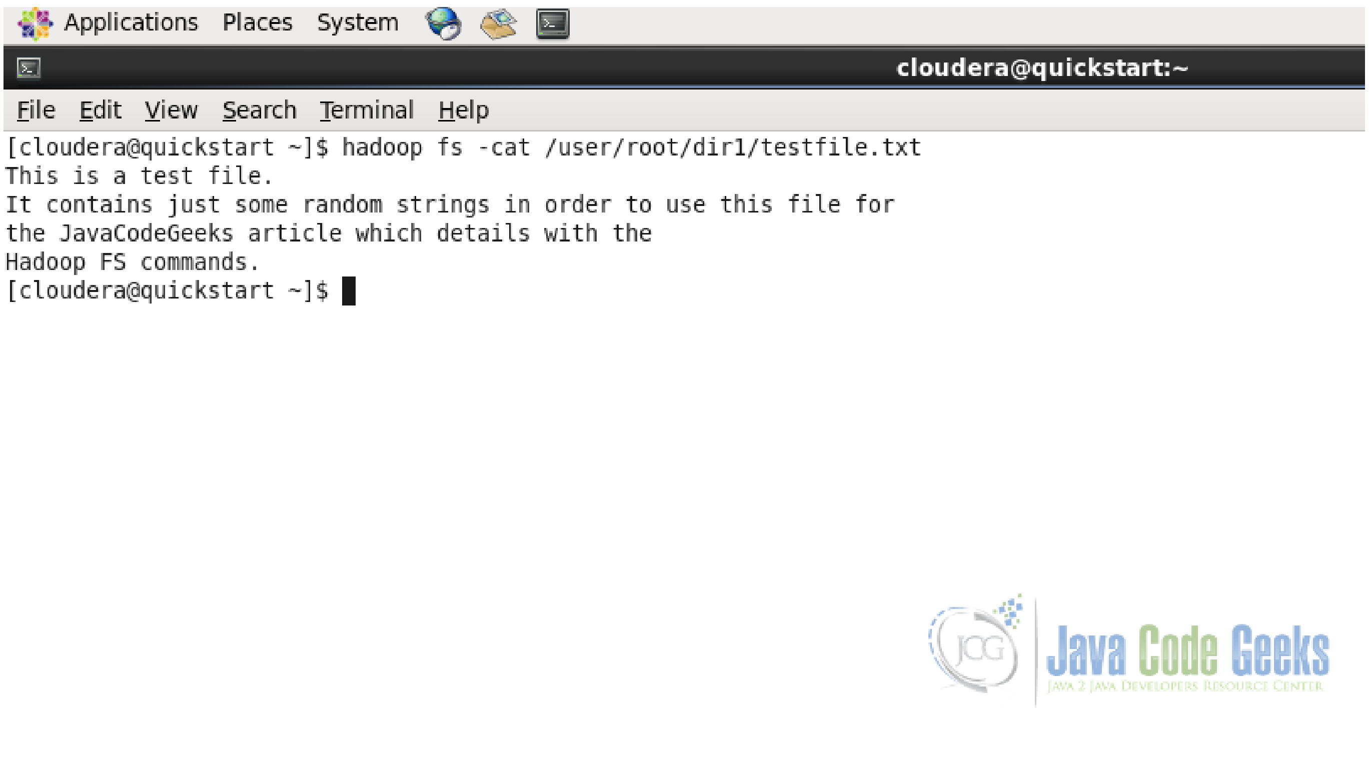
Usage:

```
hadoop fs -cat <paths>
```

Example:

```
hadoop fs -cat /user/root/dir1/testfile.txt
```

We can see that the content displayed in the screenshot below is same as the content in the `testfile.txt`


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/root/dir1/testfile.txt
This is a test file.
It contains just some random strings in order to use this file for
the JavaCodeGeeks article which details with the
Hadoop FS commands.
[cloudera@quickstart ~]$
```

Figure 3.7: Hadoop FS cat command

### 3.2.6 Copying a file

Copying a file from one place to another within the Hadoop File System is same syntax as `cp` command in unix shell.

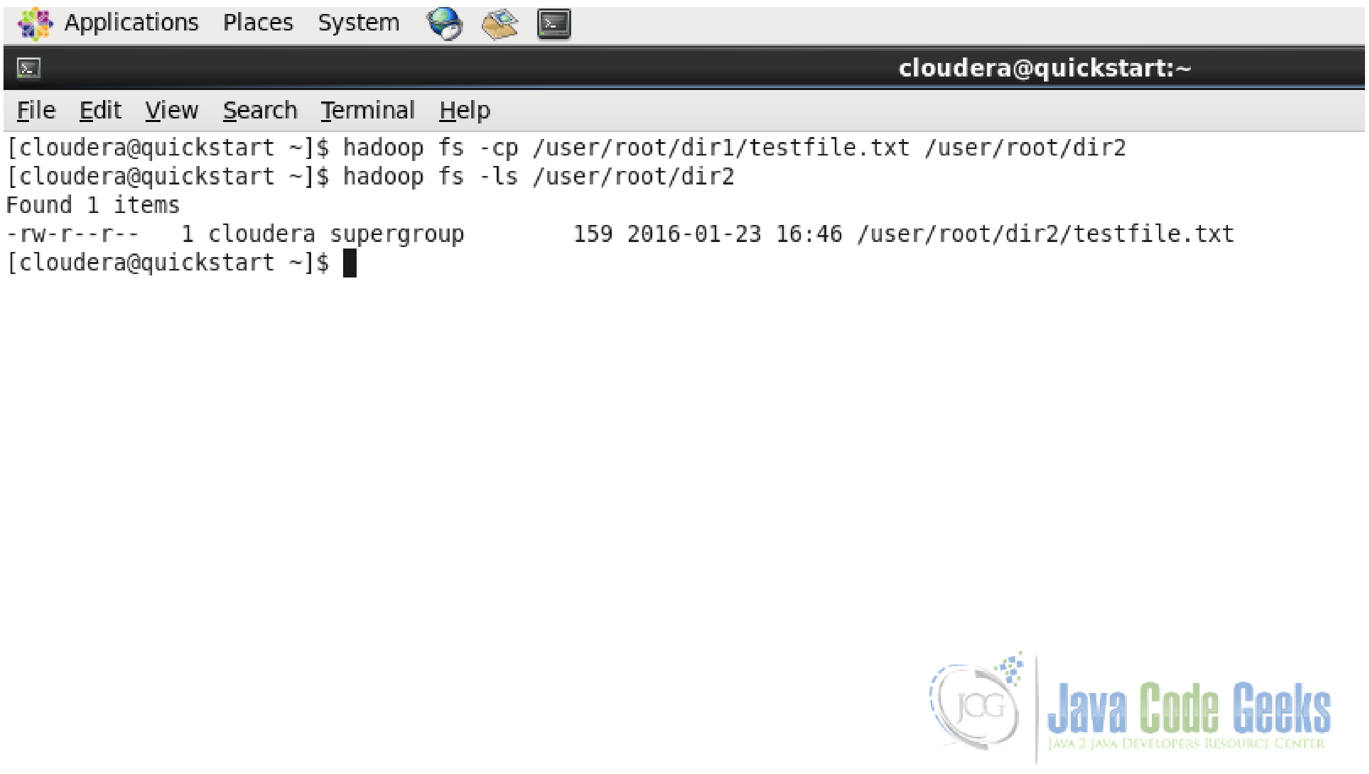
Usage:

```
hadoop fs -cp <source_path> ... <destination_path>
```

Example:

```
hadoop fs -cp /user/root/dir1/testfile.txt /user/root/dir2
```

In copying file from source to destination, we can provide multiple files in source also.


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help
Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

```
[cloudera@quickstart ~]$ hadoop fs -cp /user/root/dir1/testfile.txt /user/root/dir2
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir2
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:46 /user/root/dir2/testfile.txt
[cloudera@quickstart ~]$
```

Figure 3.8: Copying Hadoop FS file from one place to another

### 3.2.7 Moving file from source to destination

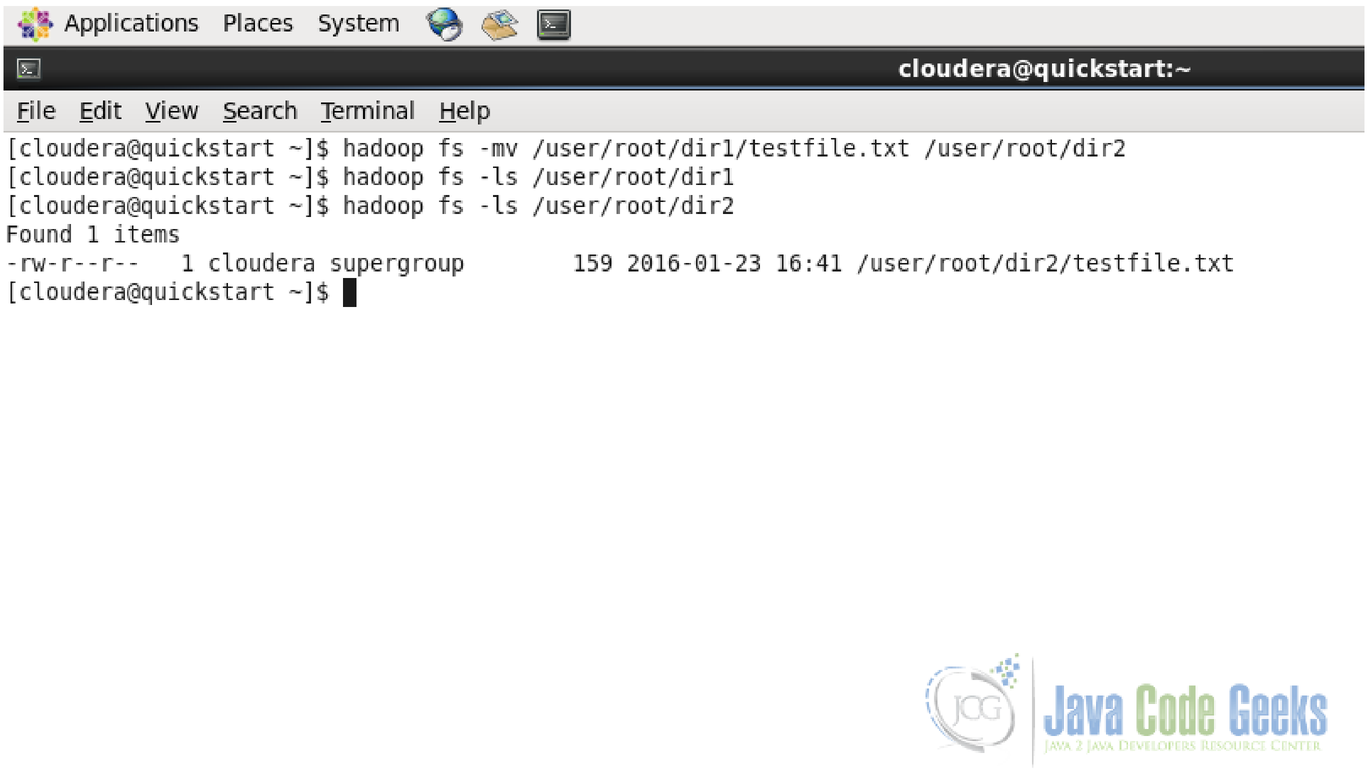
Following is the syntax and the example to move the file from one directory to another within Hadoop File System.

Usage:

```
hadoop fs -mv <source_path> <destination_path>
```

Example:

```
hadoop fs -mv /user/root/dir1/testfile.txt /user/root/dir2
```


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -mv /user/root/dir1/testfile.txt /user/root/dir2
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir2
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:41 /user/root/dir2/testfile.txt
[cloudera@quickstart ~]$
```

Figure 3.9: Moving file from one path to another

### 3.2.8 Removing the file or the directory from HDFS

Removing a file or directory from the Hadoop File System is similar to the unix shell. It also have two alternatives, `-rm` and `-rm -r`

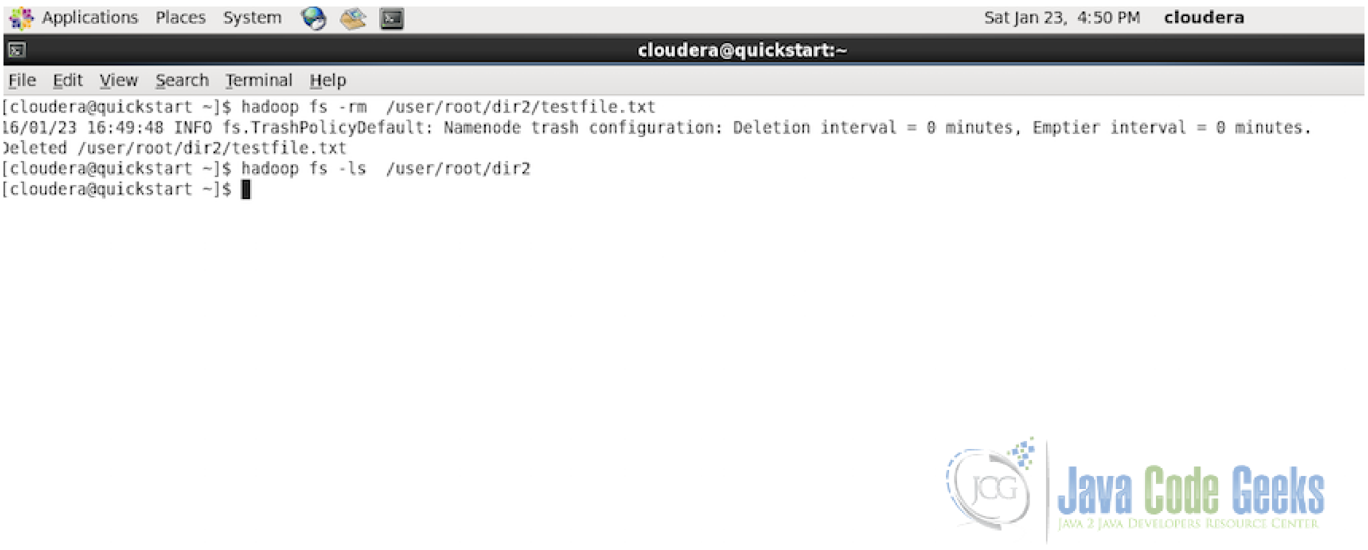
Usage:

```
hadoop fs -rm <path>
```

Example:

```
hadoop fs -rm /user/root/dir2/testfile.txt
```

The above command will only delete the particular file or in case of directory, only if it is empty. But if we want to delete a directory which contains other file, we have a recursive version of the remove command also.


Applications Places System
Sat Jan 23, 4:50 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -rm  /user/root/dir2/testfile.txt
16/01/23 16:49:48 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/root/dir2/testfile.txt
[cloudera@quickstart ~]$ hadoop fs -ls  /user/root/dir2
[cloudera@quickstart ~]$ 
```

Figure 3.10: Removing file from Hadoop FS

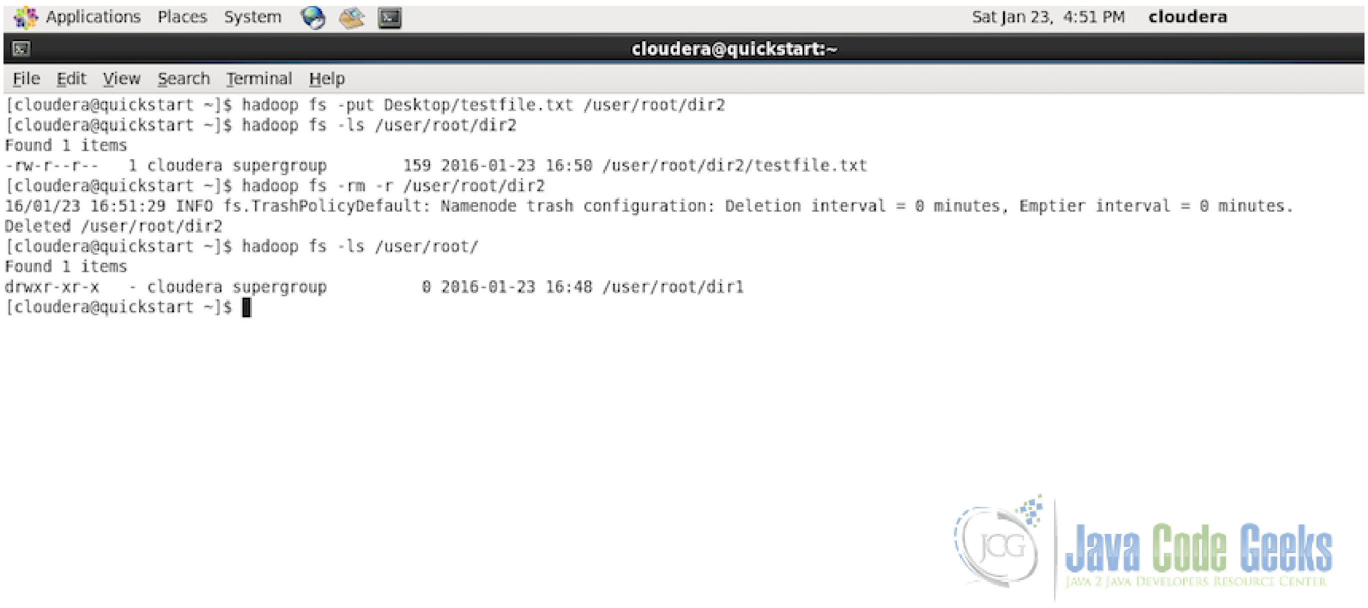
In case, we want to delete a directory which contains files, `-rm` will not be able to delete the directory. In that case we can use recursive option for removing all the files from the directory following by removing the directory when it is empty. Below is the example of the recursive operation:

Usage:

```
hadoop fs -rm -r <path>
```

Example:

```
hadoop fs -rm -r /user/root/dir2
```


Applications Places System
Sat Jan 23, 4:51 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -put Desktop/testfile.txt /user/root/dir2
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir2
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:50 /user/root/dir2/testfile.txt
[cloudera@quickstart ~]$ hadoop fs -rm -r /user/root/dir2
16/01/23 16:51:29 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/root/dir2
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/
Found 1 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:48 /user/root/dir1
[cloudera@quickstart ~]$ 
```

Figure 3.11: Removing the file recursively

### 3.2.9 Displaying the tail of a file

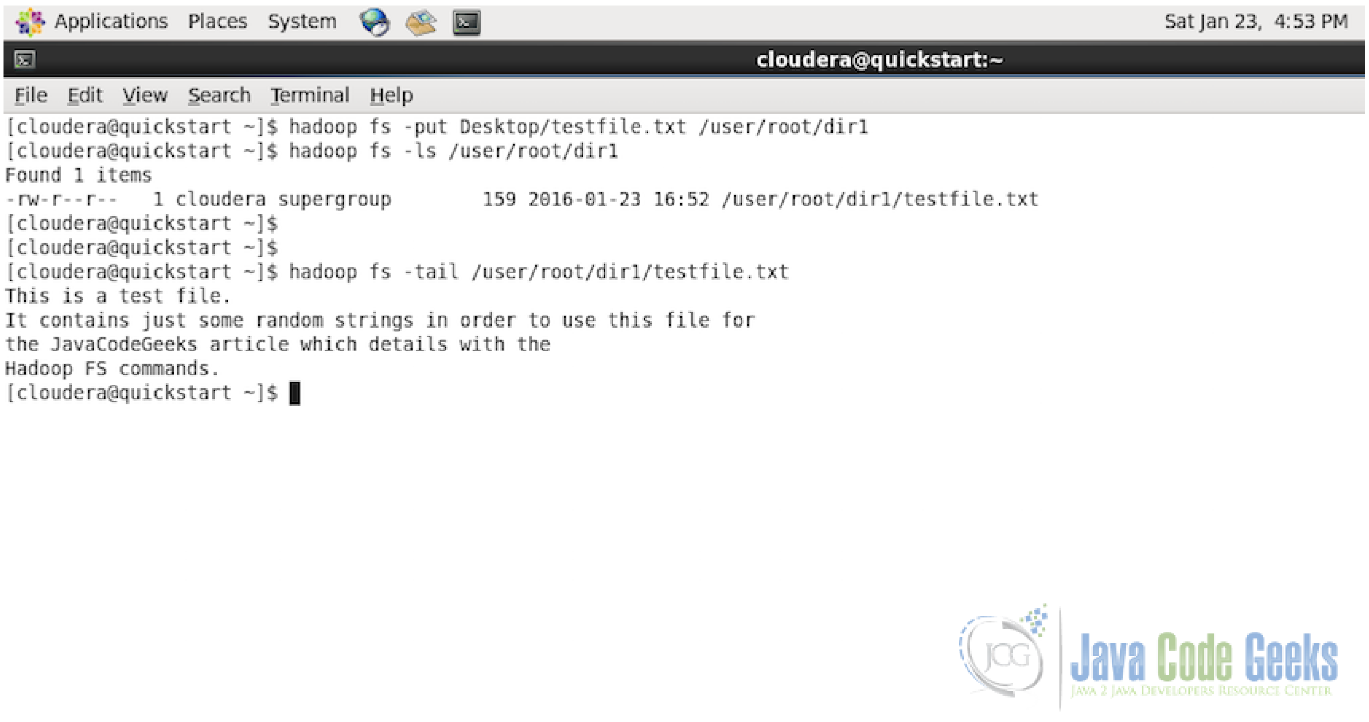
The command is exactly similar to the unix tail command.

Usage:

```
hadoop fs -tail <path>
```

Example:

```
hadoop fs -tail /user/root/dir1/testfile.txt
```


Applications Places System
Sat Jan 23, 4:53 PM
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -put Desktop/testfile.txt /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir1
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:52 /user/root/dir1/testfile.txt
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ hadoop fs -tail /user/root/dir1/testfile.txt
This is a test file.
It contains just some random strings in order to use this file for
the JavaCodeGeeks article which details with the
Hadoop FS commands.
[cloudera@quickstart ~]$
```

Figure 3.12: Tail command for Hadoop FS file.

### 3.2.10 Displaying the aggregate length of a particular file

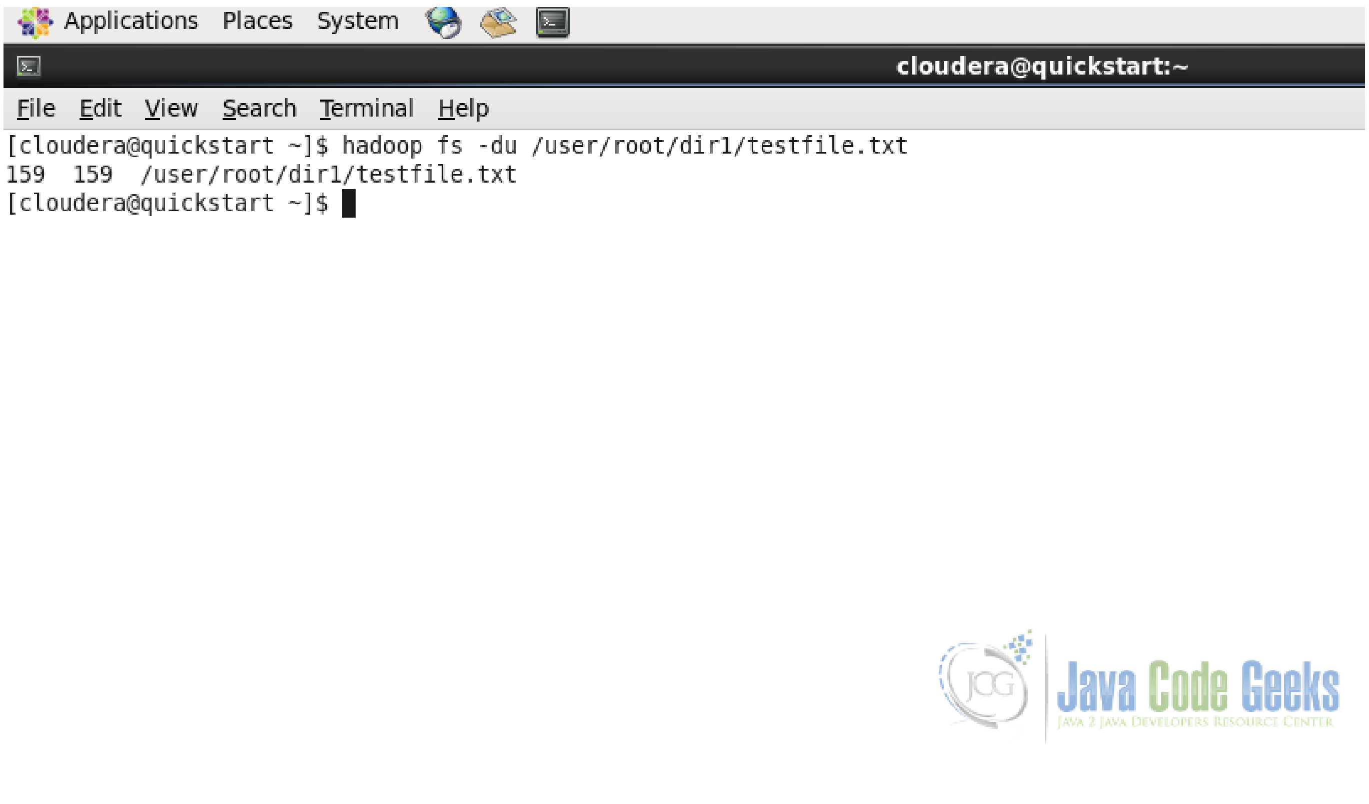
In order to check the aggregate length of the content in a file, we can use `-du.` command as below. If the path is of the file, then the length of the file is shown and if it is the path to the directory, then the aggregated size of the content if shown is shown including all files and directories.

Usage:

```
hadoop fs -du <path>
```

Example:

```
hadoop fs -du /user/root/dir1/testfile.txt
```


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -du /user/root/dir1/testfile.txt
159  159  /user/root/dir1/testfile.txt
[cloudera@quickstart ~]$ 
```

Figure 3.13: Hadoop Fs Aggregated Length

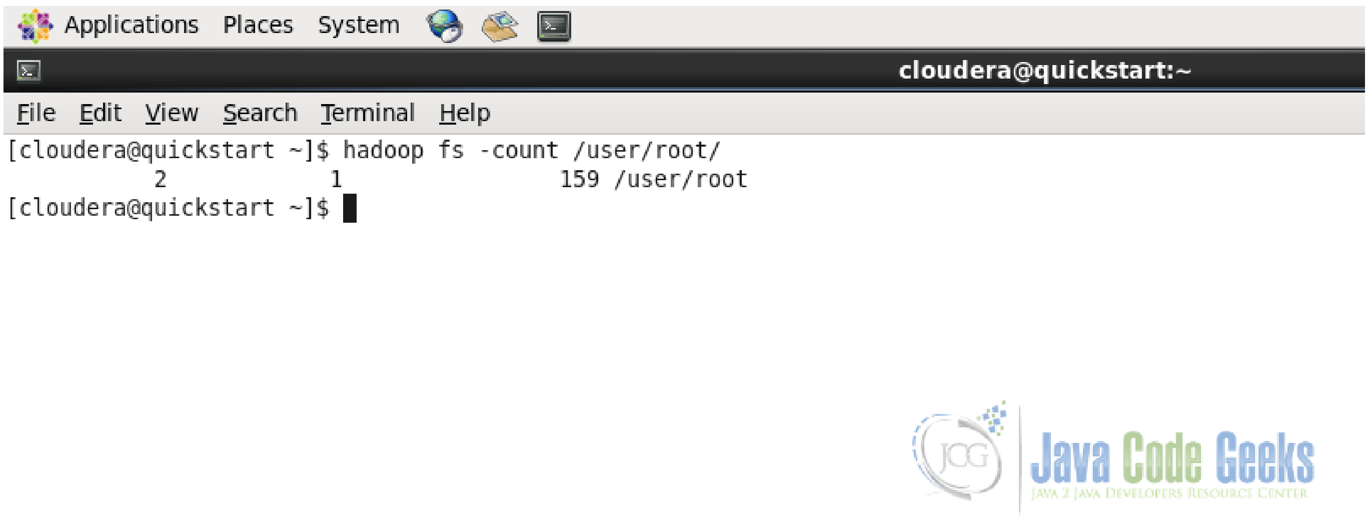
### 3.2.11 Count the directories and files

This command is to count the number of files and directories under the specified path. As in the following screenshot, the output shows the number of directories i.e. 2, number of files i.e. 1, the total content size which is 159 bytes and the path to which these stats belong to.

```
hadoop fs -count <path>
```

Example:

```
hadoop fs -count /user/root/
```


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -count /user/root/
           2            1                159 /user/root
[cloudera@quickstart ~]$
```

Figure 3.14: Count command output

### 3.2.12 Details of space in the file system

To get all the space related details of the Hadoop File System we can use `df` command. It provides the information regarding the amount of space used and amount of space available on the currently mounted filesystem

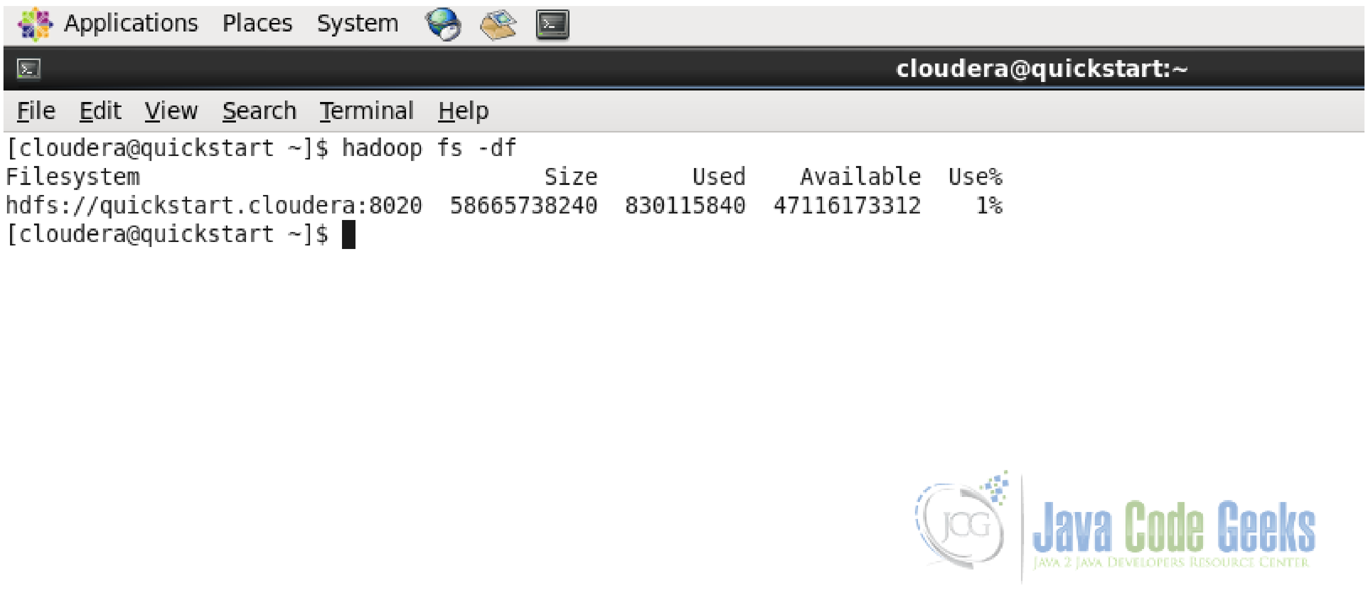
```
hadoop fs -df <path>
```

Command can be used without the path URI or with the path URI, when used without the path URI, it provides the information regarding the whole file system. When path URI id provided it provides the information specific to the path.

Example:

```
hadoop fs -df
hadoop fs -df /user/root
```

Following screenshot displays the Filesystem, Size of the filesystem, Used Space, Available Space and the Used percentage.


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -df
Filesystem                               Size       Used    Available  Use%
hdfs://quickstart.cloudera:8020  58665738240  830115840  47116173312    1%
[cloudera@quickstart ~]$
```

Figure 3.15: DF command output

## 3.3 Conclusion

This brings us to the conclusion of the example. These Hadoop File System commands will help you in getting a head start in dealing with the files and directories in the Hadoop Ecosystem.

# Chapter 4

# Cluster Setup Example

## 4.1 Introduction

Apache Hadoop is designed for a multi-machine cluster setup. Though it is possible to run on single machine also for testing purpose but real implementation is for multi-machine clusters. Even if we want to try out multi-machine setup we will need multiple systems which are connected to each other over the network which is not possible always; what if you do not have multiple systems to try out Hadoop Cluster?

Virtual Machines comes to rescue here. Using multiple Virtual Machines we can setup Hadoop Cluster using a single system. So, in this example, we will discuss how to setup Apache Hadoop Cluster using Virtual Machines.

## 4.2 Requirements

- VirtualBox (or any other VM environment)
- Lubuntu 15.04 (or any other linux flavor of your preference)
- VBox Guest Additions image file (VBoxGuestAdditions.iso)

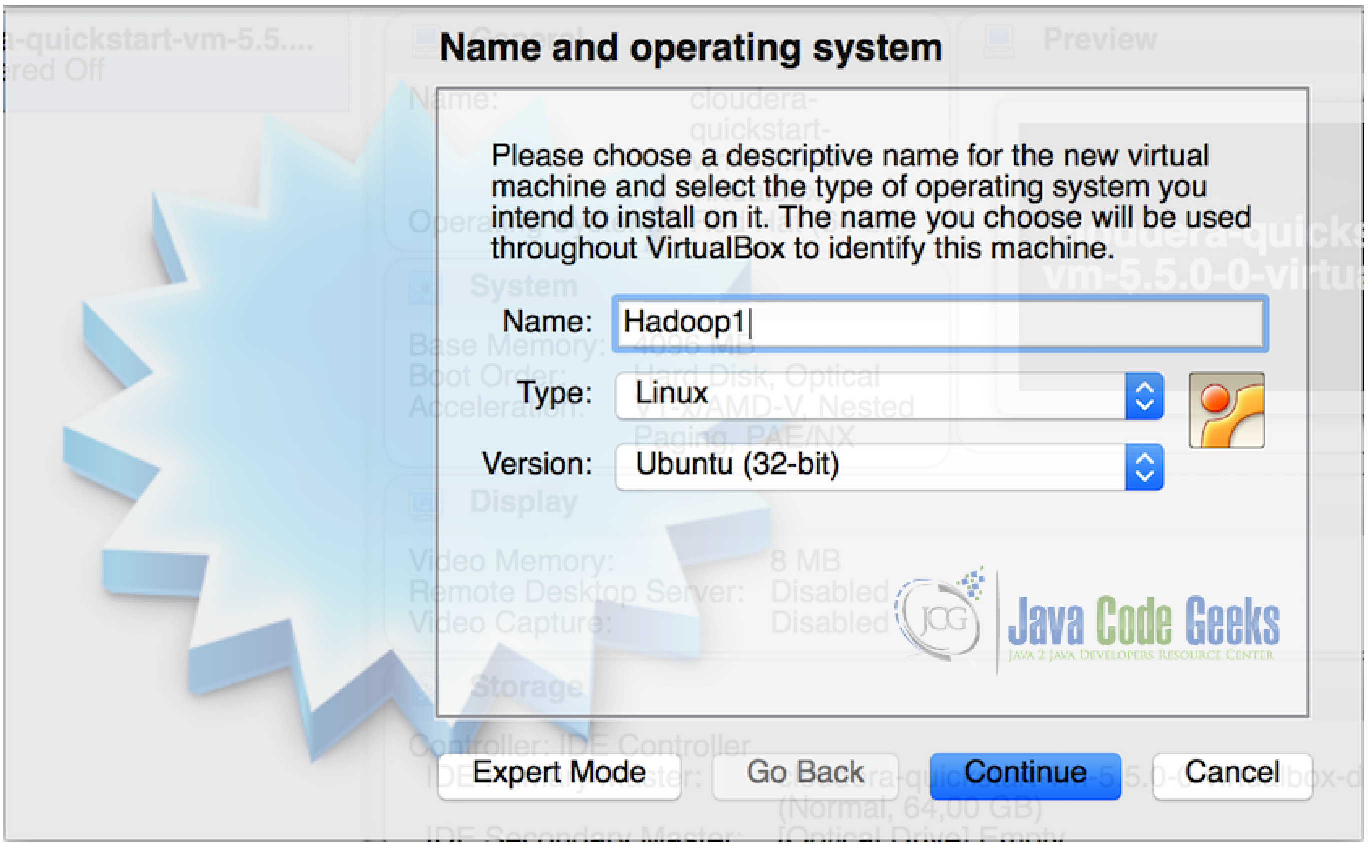
I personally prefer Lubuntu as it has lightweight LXDE Desktop GUI and it strips all the additional components which are present in Ubuntu and is a good option for virtual machines.

## 4.3 Preparing Virtual Machine

In this section we will go through steps to prepare virtual machines which we will use for cluster later in the example.

### 4.3.1 Creating VM and Installing Guest OS

Create a virtual machine(VM) in VirtualBox and assign minimum 2GB of memory and 15GB of storage to the virtual machine. Name the first VM as Hadoop1.



Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

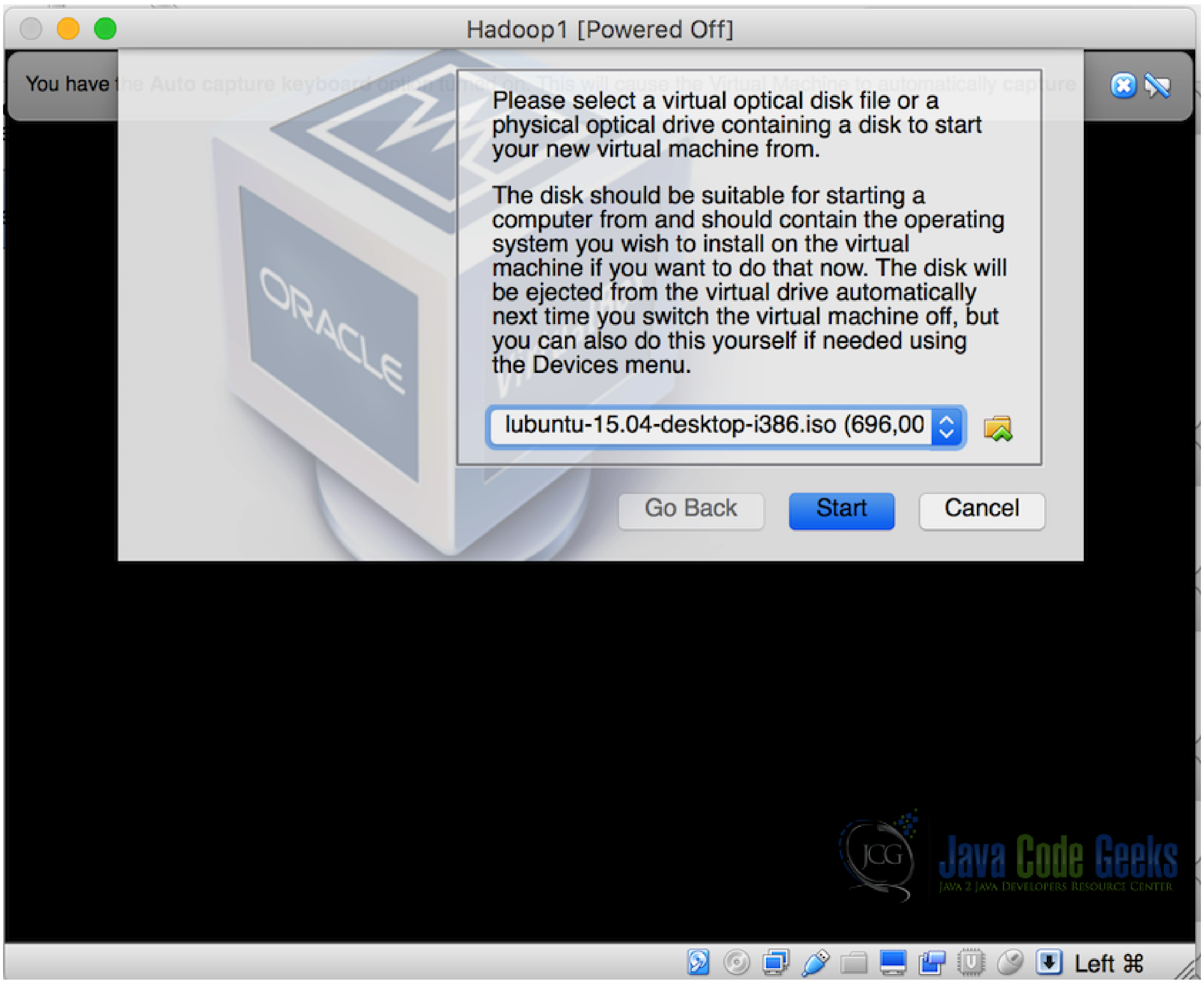
Name: Hadoop1

Type: Linux

Version: Ubuntu (32-bit)

Figure 4.1: Creating Virtual Machine in VirtualBox

Once the VM is created, install Lubuntu in the VM and complete the setup, we will get a working virtual machine after this.


Hadoop1 [Powered Off]

You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

lubuntu-15.04-desktop-i386.iso (696,00

Figure 4.2: Installing Lubuntu in created VM

Installation of the OS might take some time.

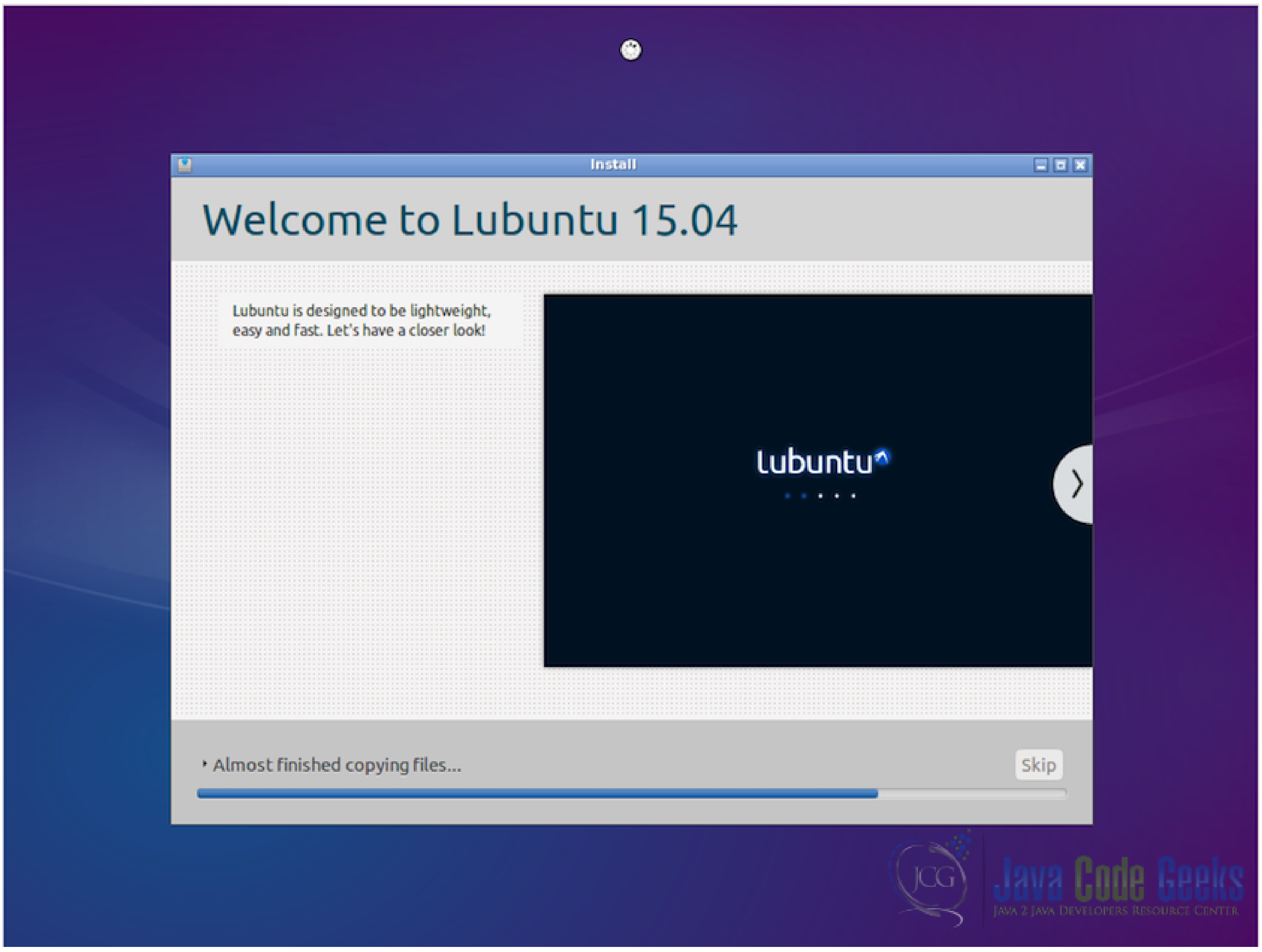

Install
Welcome to Lubuntu 15.04
Lubuntu is designed to be lightweight, easy and fast. Let's have a closer look!
lubuntu
Almost finished copying files...
Skip

Figure 4.3: Lubuntu installation in progress

### 4.3.2 Installing Guest Additions

Next step is to install Guest Additions in the VM. Guest Additions are additional setup needed for the VM to perform well. It consist of device drivers and system applications that optimize the guest operating system for better performance and usability. This is one of the important and needed step whenever creating a virtual machine, one thing it allows the Guest OS to detect the size of the screen(which helps in running the VM full screen) and also enabling guest operating system to have a shared folder with the host operating system if needed. Following are the steps which need to be performed for installing guest additions in the Guest OS:

First of all, prepare the system for building external kernel modules which can be done by running the following command in the terminal and installing DKMS(DKMS provides support for installing supplementary versions of kernel modules):

```
sudo apt-get install dkms
```

Insert `VBoxGuestAdditions.iso` CD file into Linux guest virtual CD-ROM drive.

Now open the terminal and change the directory to the CD-ROM drive and then execute the following command:

```
sh ./VBoxLinuxAdditions.run
```

**Note:** At this point reboot the system and move on to the next step where we will configure the network settings for the virtual machine.

## 4.4 Creating Cluster of Virtual Machines

In this section we will see how to configure the network for the virtual machines to act as single cluster machines, how to clone the first machine to others which will save all the time as we do not need to perform previous steps on all the machine individually.

### 4.4.1 VM Network settings

Go to Virtualbox preferences menu and select *Preferences* from the dropdown menu.

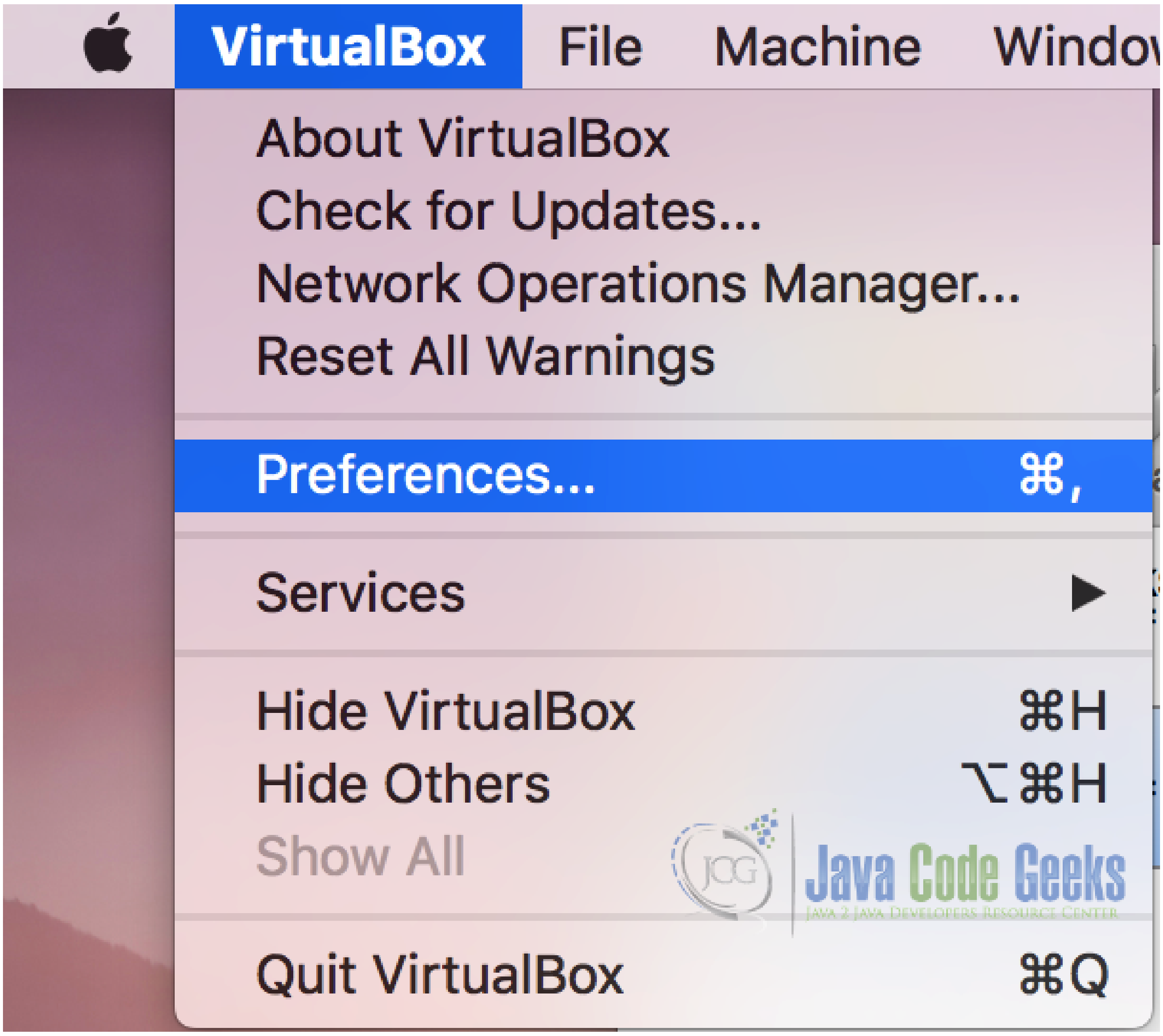
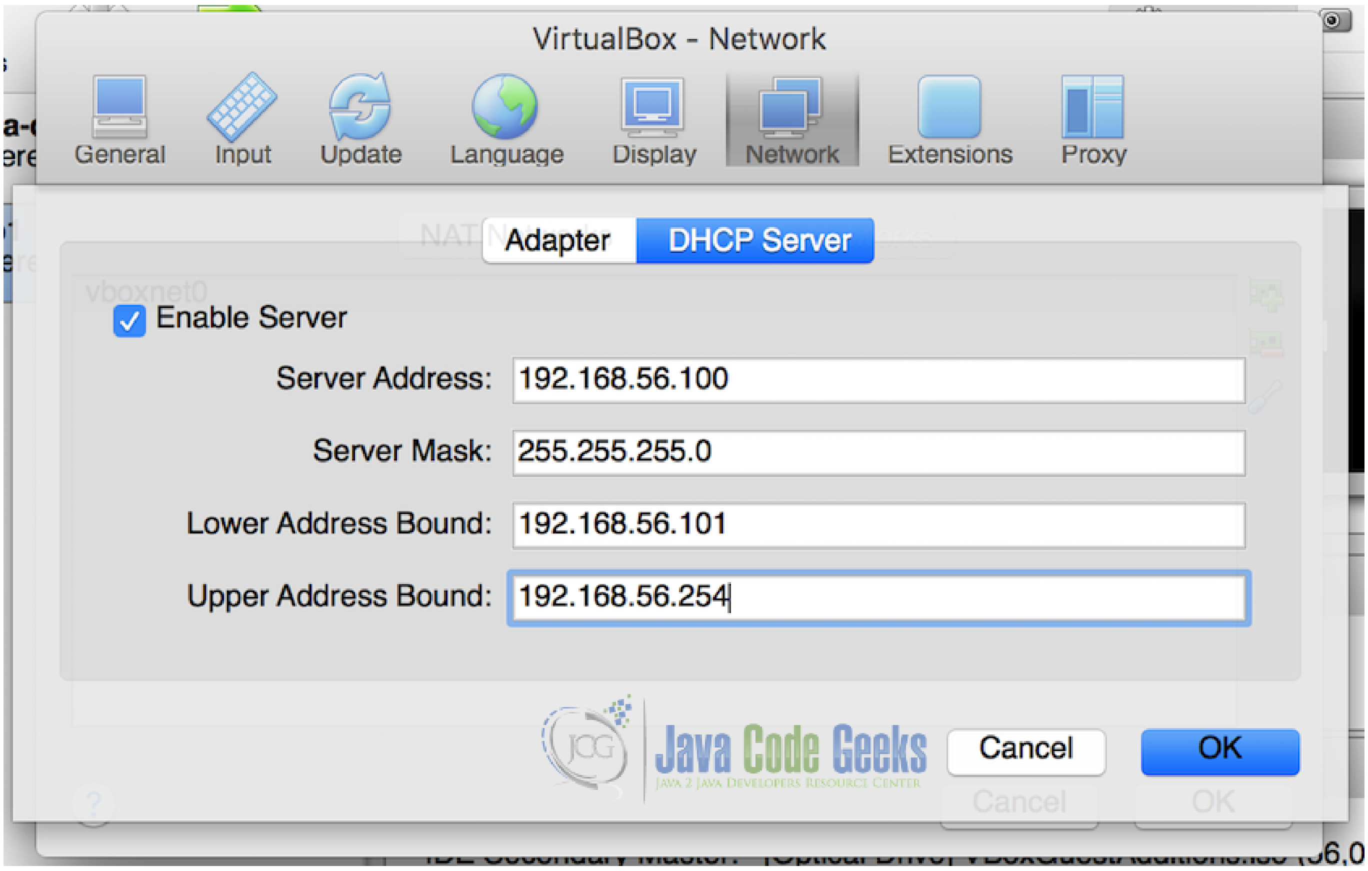


Figure 4.4: VirtualBox Preferences Menu

In *Preferences* menu, select *Network*. In network preferences, select *Host-only Networks* and click on *Add Driver*. Driver will be added to the list. Double click the driver and it will open a popup for DHCP server settings, insert DHCP server settings as shown in the screenshot below.



☑ Enable Server

Server Address: 192.168.56.100

Server Mask: 255.255.255.0

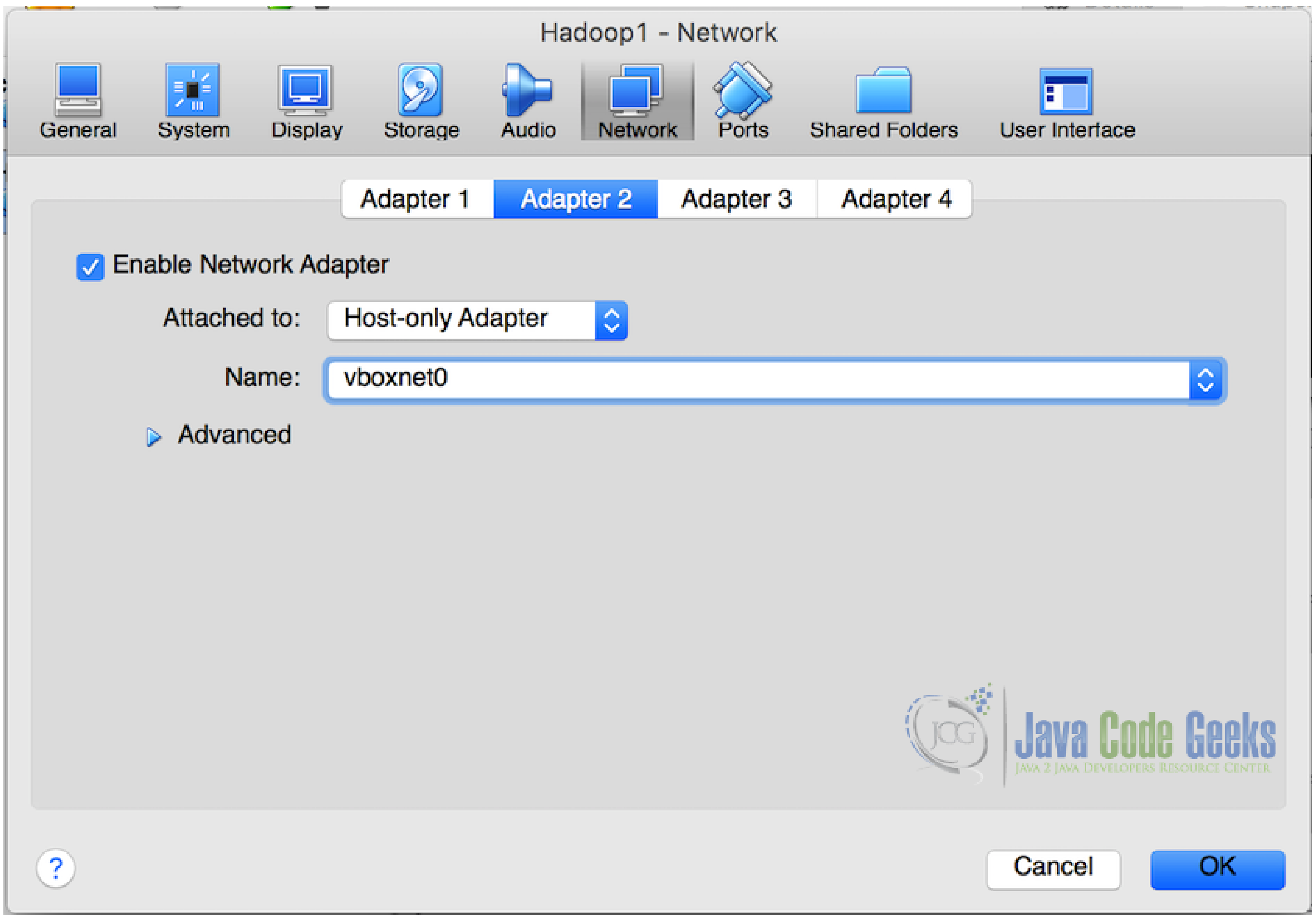
Lower Address Bound: 192.168.56.101

Upper Address Bound: 192.168.56.254

Figure 4.5: DHCP Server Settings

We will set the lower bound and upper bound of the network to be *192.168.56.101* and *192.168.56.254*, all the machines will have the IPs assigned from this range only. **Do not forget the check *Enable Server***

Once the network settings are done and DHCP server ready, in the VirtualBox Manager, right-click on the virtual machine and from the list and select *Settings* from the dropdown. From the settings popup, select *Network* and then *Adapter2'Check 'Enable Network Adapter* and then in *Attached to* dropdown choose *Host-only adapter*. In second dropdown, names of all the adapters will be available including the one we created in the previous step. Select that from the dropwdown, in our example it is names as *vboxnet0*. This will attach the virtual machine to this particualr network.



Enable Network Adapter ✓

Attached to: Host-only Adapter

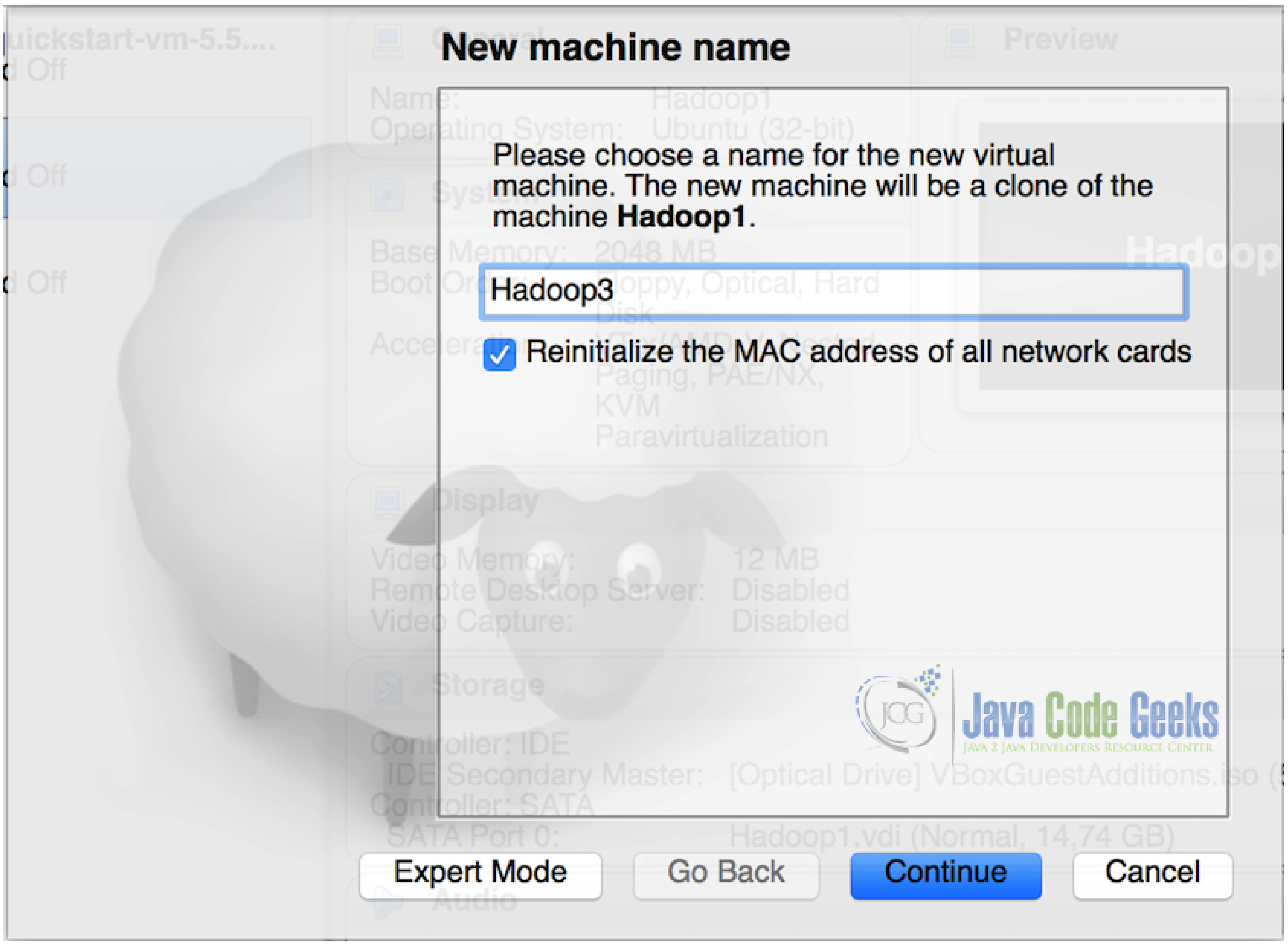
Name: vboxnet0

Figure 4.6: Virtual Machine Settings

### 4.4.2 Cloning the Virtual Machine

Now we have a virtual machine ready and we cna not clone this virtual machine to create identical machines, this saves us from the hassle of all the previous steps and we can easily have multiple virtual machines with the same configuration as the one they are cloned from.

Right-click on the virtual machine and from the dropdown select *Clone*. In the clone popup, rename the VM to *Hadoop2* and select *Reinitialize the MAC address of all the network cards* and click Continue.


New machine name

Please choose a name for the new virtual machine. The new machine will be a clone of the machine **Hadoop1**.

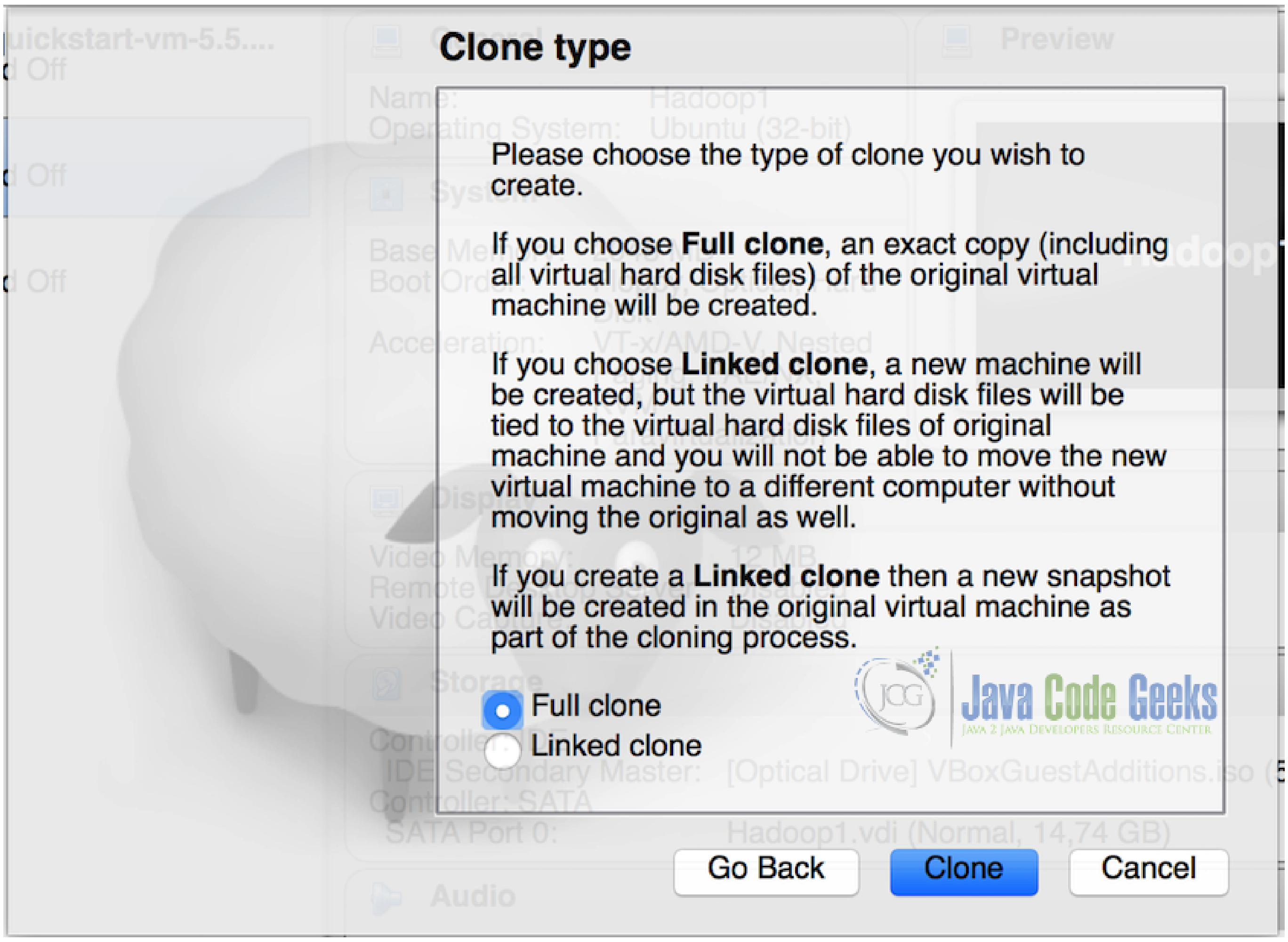
Hadoop3

✓ Reinitialize the MAC address of all network cards

Figure 4.7: Cloning the Virtual Machine

**Note:** Reinitializing the MAC address make sure that the new Virtual Machine will have a different MAC address for the network card.

In the next screen, select *Full Clone* option and click on *Clone*.



**Clone type**

Please choose the type of clone you wish to create.

If you choose **Full clone**, an exact copy (including all virtual hard disk files) of the original virtual machine will be created.

If you choose **Linked clone**, a new machine will be created, but the virtual hard disk files will be tied to the virtual hard disk files of original machine and you will not be able to move the new virtual machine to a different computer without moving the original as well.

If you create a **Linked clone** then a new snapshot will be created in the original virtual machine as part of the cloning process.

[✓ Full clone]
Linked clone

Figure 4.8: Full clone of the Virtual Machine

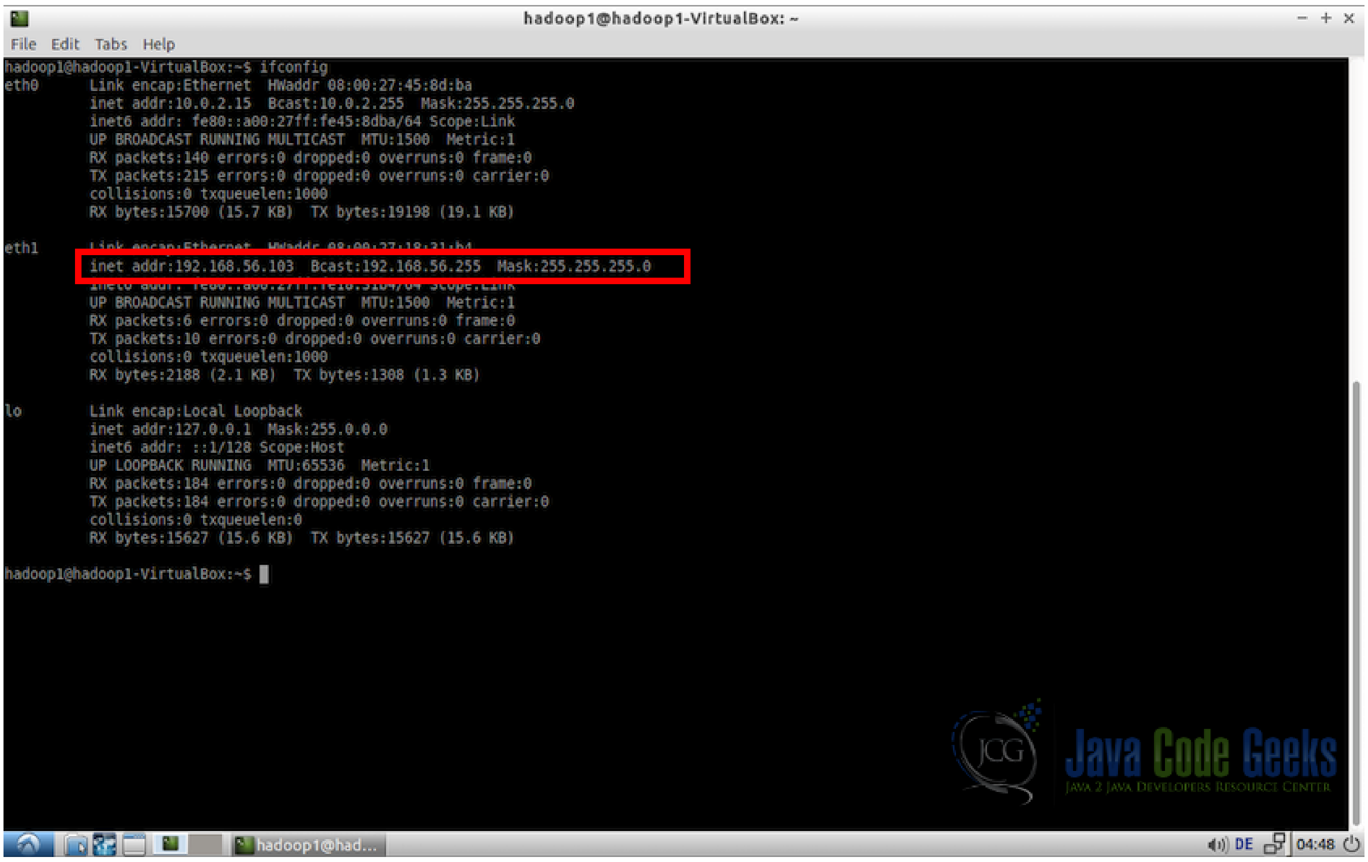
### 4.4.3 Testing the network IPs assigned to VMs

So now we have 2 machines on the same network. We have to test if both the machines are connected to the network adapter we setup for the cluster. Following are the steps to do so:

Start both the virtual machines and in terminals use the following command:

```
ifconfig
```

This will show the network configuration of the machine. We will notice that the IP assigned is in the range 192.168.56.101 and 192.168.56.254 (i.e. between lower address bound and upper address bound assigned to the DHCP network)



hadoop1@hadoop1-VirtualBox: ~

```
hadoop1@hadoop1-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:45:8d:ba
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe45:8dba/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:140 errors:0 dropped:0 overruns:0 frame:0
          TX packets:215 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15700 (15.7 KB)  TX bytes:19198 (19.1 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:18:31:b4
          inet addr:192.168.56.103  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe18:31b4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2188 (2.1 KB)  TX bytes:1308 (1.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:184 errors:0 dropped:0 overruns:0 frame:0
          TX packets:184 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:15627 (15.6 KB)  TX bytes:15627 (15.6 KB)

hadoop1@hadoop1-VirtualBox:~$
```

Figure 4.9: IP configuration of the virtual machine

**Note:** Perform the same task for both the machines and confirm everything is fine.

### 4.4.4 Converting to Static IPs for VMs

There will be one problem with this configuration though. IPs are allocated randomly to the systems and may change in the future reboots. Hadoop need static IPs to access the machines in the cluster, so we need to fix the IPs of the machines to be static always and assign specific IPs for both the machines. The following steps need to be performed on both the machines.

Go to `/etc/networks` in the terminal and edit the file `interfaces` as a root.

```
#Go to networks directory
cd /etc/networks
#Edit the file 'interfaces'
sudo nano interfaces
```

Add the following lines at the end of the interfaces file.

```
auto eth1
iface eth1 inet static

#Assign a static ip to the virtual machine
address 192.168.56.101
netmast 255.255.255.0
network 192.168.56.0

#Mention the broadcast address, get this address using ifconfig commmand
#in this case, is it 192.168.56.255
broadcast 192.168.56.255
```

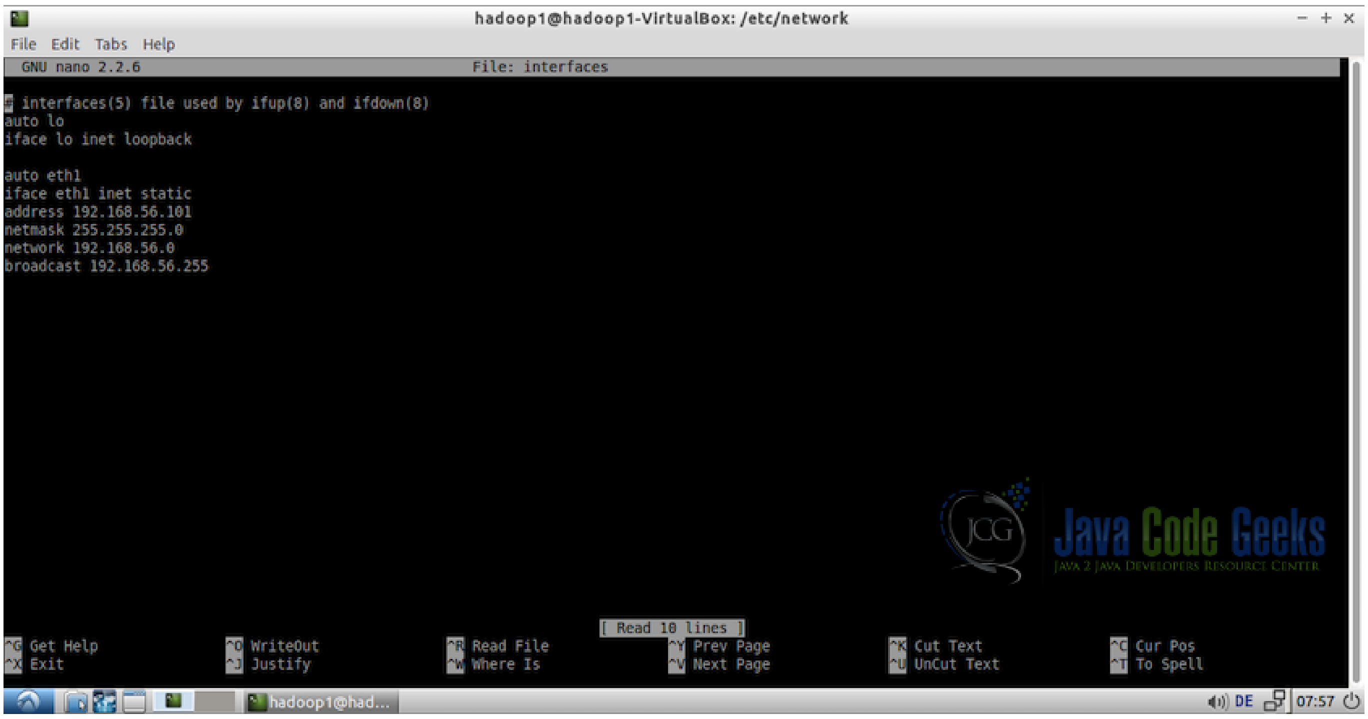

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth1
iface eth1 inet static
address 192.168.56.101
netmask 255.255.255.0
network 192.168.56.0
broadcast 192.168.56.255
```

Figure 4.10: Interfaces file

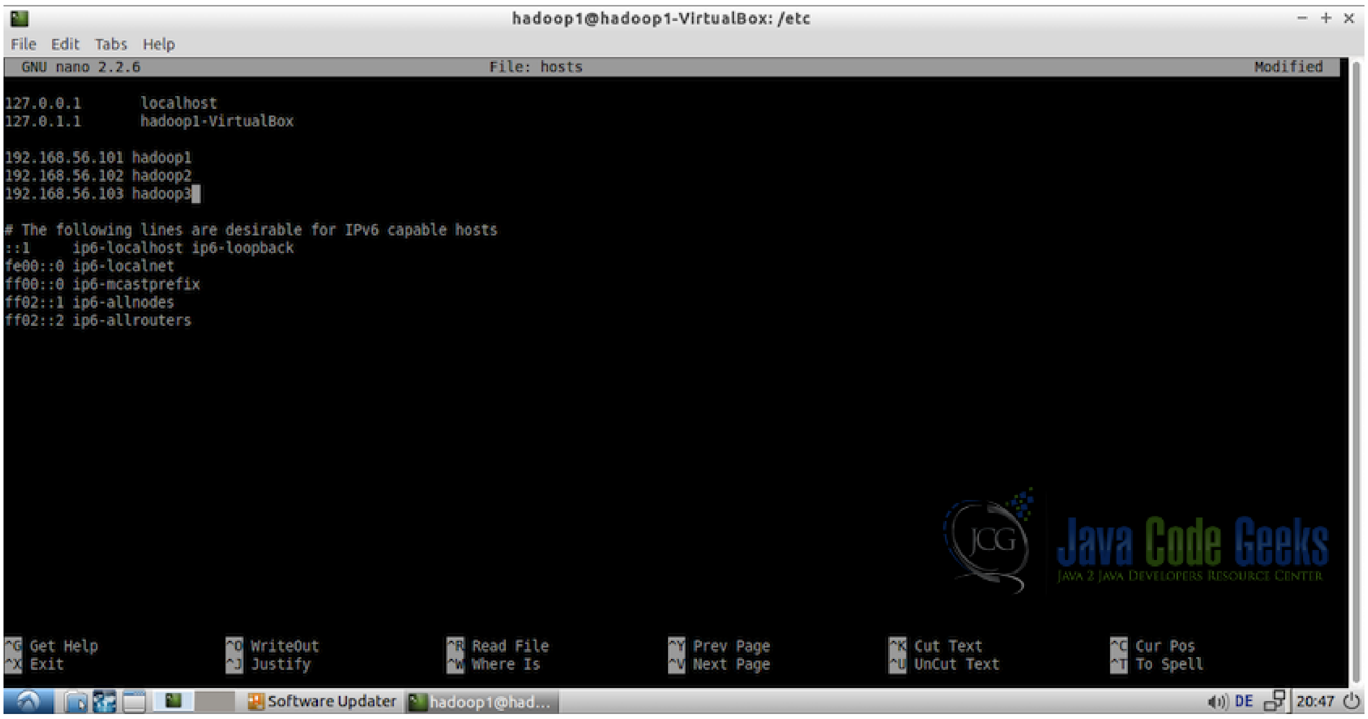
On each machine, edit the file `/etc/hosts` as root and add the hosts. For example:

```
#Edit file using nano editor
sudo nano /etc/hosts
```

Add following hosts:

```
192.168.56.101 hadoop1
192.168.56.102 hadoop2
```

**Note:** IPs should be same as assigned in the previous step.



```
127.0.0.1       localhost
127.0.1.1       hadoop1-VirtualBox

192.168.56.101 hadoop1
192.168.56.102 hadoop2
192.168.56.103 hadoop3

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Figure 4.11: Hosts file in the virtual machine

Reboot all the machines

## 4.5 Hadoop prerequisite settings

Following are the prerequisite settings for hadoop setup. Remember all the settings need to be done in all the machines which will be added to cluster(2 machines in this example)

### 4.5.1 Creating User

Create hadoop users in all the machines. For that open the terminal and enter the following commands:

```
#create a user group for hadoop
sudo addgroup hadoop

#create user hduser and add it to the hadoop usergroup
sudo adduser --ingroup hadoop hduser
```

### 4.5.2 Disable ipv6

Next step is to disable ipv6 on all the machines. Hadoop is set to use ipv4 and that is why we need to disable ipv6 before creating a hadoop cluster. Open `/etc/sysctl.conf` as root using nano

```
sudo nano /etc/sysctl.conf
```

and add the following lines at the end of the file.

```
#commands to disable ipv6
net.ipv6.conf.all.disable-ipv6=1
net.ipv6.conf.default.disable-ipv6=1
net.ipv6.conf.lo.disable-ipv6=1
```

After this, to check if the ipv6 is properly disable, use the following command

```
cat /proc/sys/net/ipv6/conf/all/disable-ipv6
```

it will return 0 or 1 as an output and we want it to be 1 as it symbolizes that the ipv6 is disabled.

### 4.5.3 Connecting the machines (SSH Access)

Now, we have to make sure that the machines are able to reach each other over the network using static IP addresses and SSH. For this example, we will consider `hadoop1` machine as the master node and `hadoop1` and `hadoop2` both as the slave nodes. So we have to make sure:

hadoop1(master) should be able to connect to itself using

```
ssh hadoop1
```

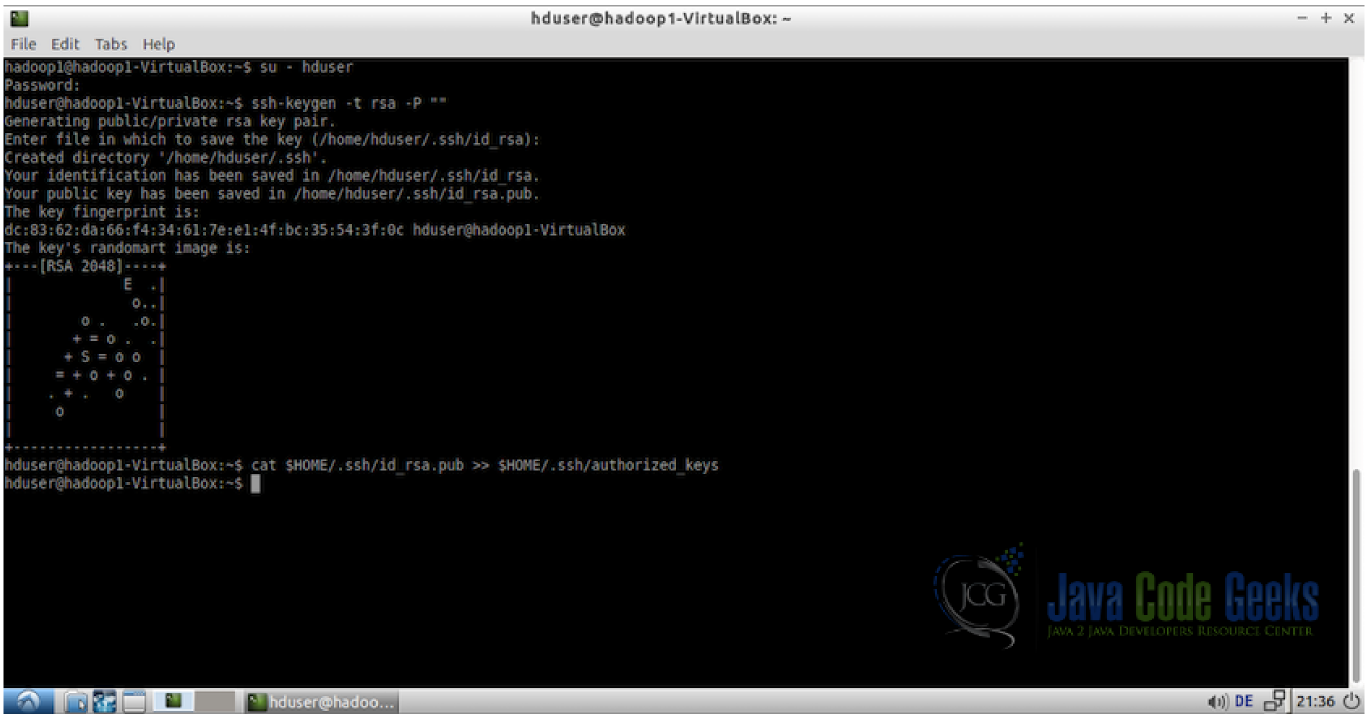
It should be able to connect to other VM using

```
ssh hduser@hadoop2
```

To achieve this, we have to generate SSH key in each machine. So login to `hadoop1` and following the steps mentioned below in the terminal:

Switch to the user `hduser` and generate the SSH public keys:

```
#change to user hduser
su - hduser

#generate ssh key
ssh-keygen -t rsa -P ""
```



```
hadoop1@hadoop1-VirtualBox:~$ su - hduser
Password:
hduser@hadoop1-VirtualBox:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
dc:83:62:da:66:f4:34:61:7e:e1:4f:bc:35:54:3f:0c hduser@hadoop1-VirtualBox
The key's randomart image is:
+--[RSA 2048]----+
|            E  .|
|             o..|
|       o .   .o.|
|      + = o .  .|
|     + S = o o  |
|    = + o + o . |
|   . + .   o    |
|    o           |
|                |
+-----------------+
hduser@hadoop1-VirtualBox:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
hduser@hadoop1-VirtualBox:~$ 
```

Figure 4.12: SSH Keygenration

The second command will create an RSA key-pair for the machine. The password for this key will be empty as mentioned in the command. It will ask for the path to store the key with default path being `$HOME/.ssh/id-rsa.pub`, just press enter when prompted to keep the same path. If you plan to change the path then remember it as it will be needed in the next step.

Enable SSH access to the machine with the key created in the previous step. For this, we have to add the key to the authorized keys list of the machine.

```
cat $HOME/.ssh/id-rsa.pub >> $HOME/.ssh/authorized_keys
```

Now we have to add the `hduser@hadoop1`'s public SSH key (master node) to the authorized keys file of the `hduser@hadoop2` machine. This can be done using the following commands on the terminal of `hadoop1`:

```
ssh-copy-id -i $HOME/.ssh/id-ras.pub hduser@hadoop2
```

This will prompt for the password for the user `hduser@hadoop2`

Test the SSH connections from `hadoop1` to itself and also to `hadoop2` to make sure everything is fine using:

```
ssh hadoop1
```

This will connect `hadoop1` to itself, if connected successfully, exit the connection and try to connect to the `hadoop2` machine

```
ssh hduser@hadoop2
```

This should also connect successfully.

## 4.6 Hadoop Setup

So, we are at the step where we have completed all the initial setup and now we are ready to setup hadoop on the cluster.

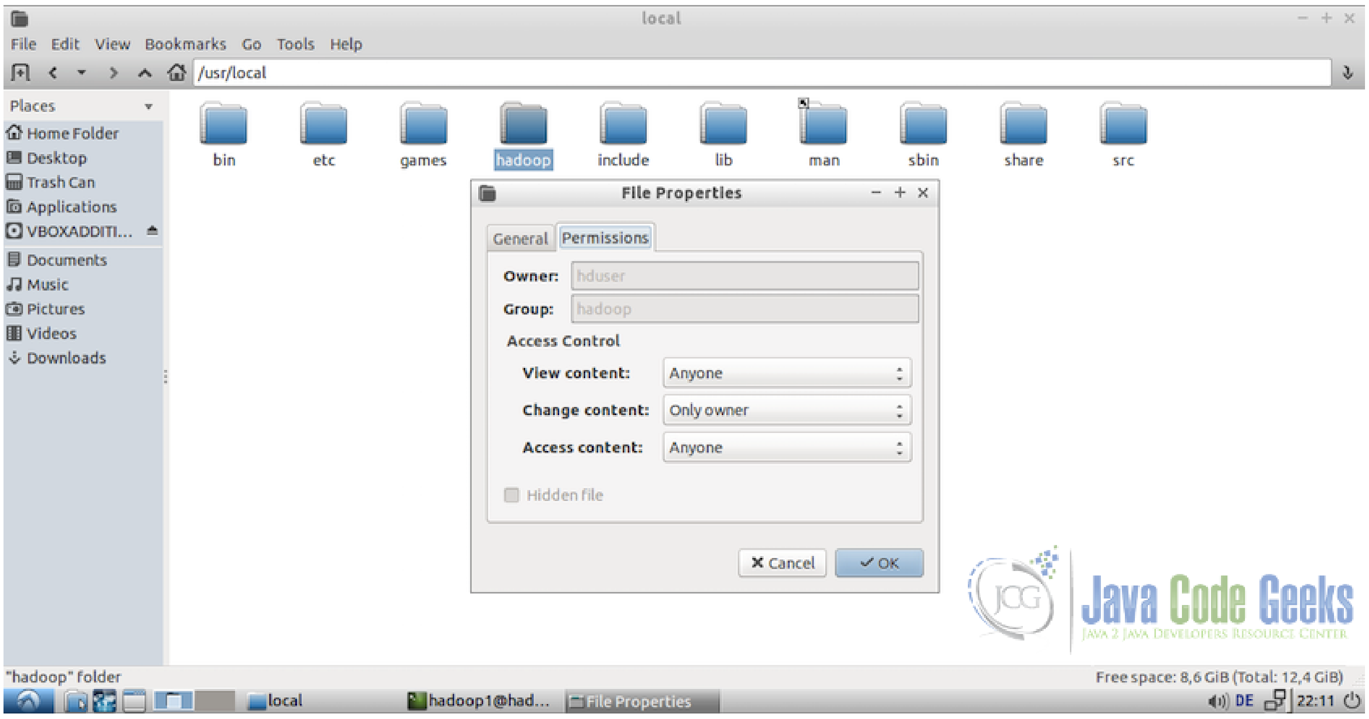
### 4.6.1 Download Hadoop

Download Hadoop from Apache Mirrors at www.apache.prg/dyn/closer.cgi/hadoop/core

After download finishes, extract hadoop folder and move it to `/usr/local/hadoop` and finally change the owner of the folder to `hduser` and `hadoop` group.

```
#Change to the directory
cd /usr/local

#move hadoop files to the directory
sudo mv /home/hadoop1/Downloads/hadoop-2.7.1 hadoop

#change the permissions to the hduser user.
sudo chown -R hduser:hadoop hadoop
```

We can check the permissions in the folder setting to confirm if they are fine.



Owner: hduser
Group: hadoop
Access Control
View content: Anyone
Change content: Only owner
Access content: Anyone
Hidden file

Figure 4.13: Folder settings to check permissions

### 4.6.2 Update bashrc

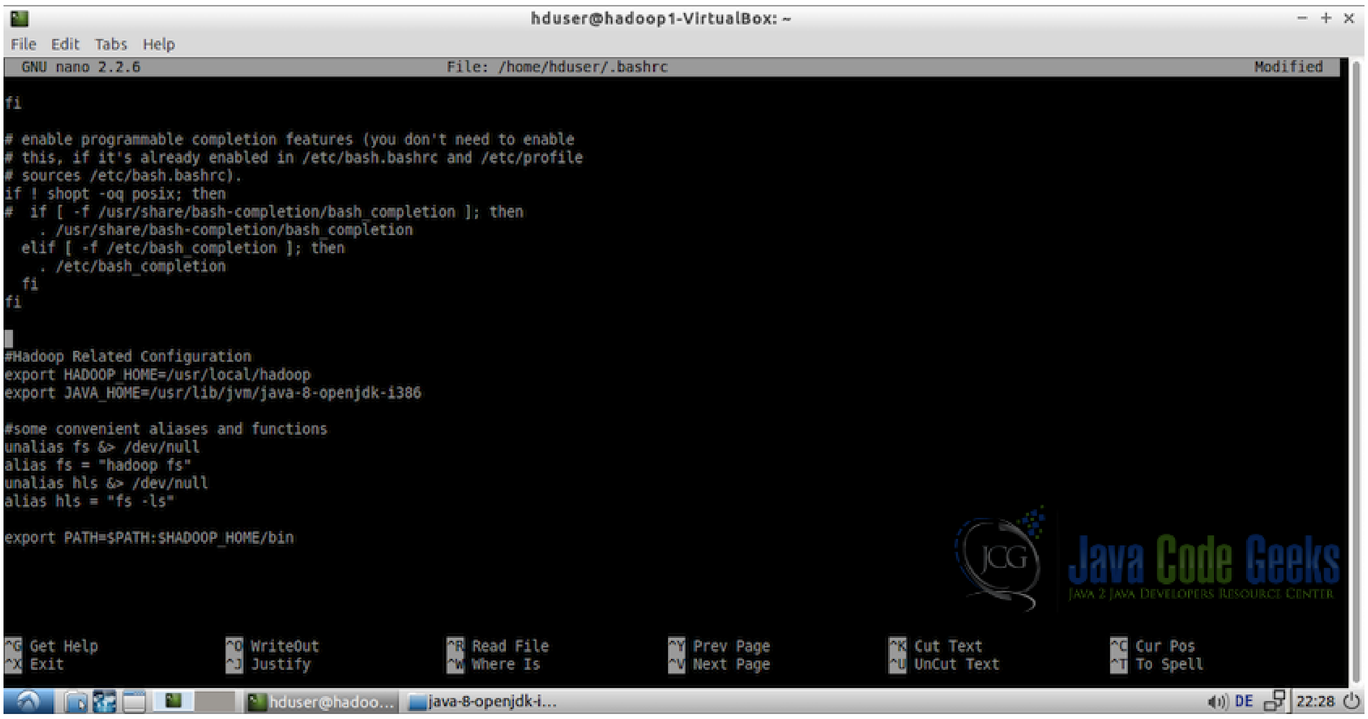
Update the bashrc file for the user hduser.

```
su - hduser
nano $HOME/.bashrc
```

At the end of the file, add the folloeing lines.

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386

#Some convenient aliases
unalias fs &amp;> /dev/null
```

```
alias fs="hadoop fs"
unalias hls &amp;> /dev/null
alias hls="fs -ls"

export PATH=$PATH:$HADOOP_HOME/bin
```



```
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
#  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi


#Hadoop Related Configuration
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386

#some convenient aliases and functions
unalias fs &> /dev/null
alias fs = "hadoop fs"
unalias hls &> /dev/null
alias hls = "fs -ls"

export PATH=$PATH:$HADOOP_HOME/bin
```

Figure 4.14: Updating bashrc file of user hduser

### 4.6.3 Configuring Hadoop

Now, it is the time to configure the hadoop setup. Following are the steps which need to be followed:

This need to be performed on all the machines. Open `hadoop-env.sh` in `/usr/local/hadoop/etc/hadoop/` and set the `JAVA_HOME` variable as shown below:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386
```

Next, we will configure the `core-site.xml` in the folder `/usr/local/hadoop/etc/hadoop/` and add the following property

```
<configuration>
      <property>
         <name>fs.default.FS</name>
         <value>hdfs://hadoop1:54310</value>
      </property>
   </configuration>
```

This will also need to be edited in all the machine but all the `value` fields should point to the master node only which is `hadoop1` in this example. So for both the machines, same property with same name and value need to be added.

Next we need to update `hdfs-site.xml` on all master and slave nodes

```
        <configuration>
      <property>
```

```
            <name>dfs.replication</name>
            <value>2</value>
        </property>
        <property>
            <name>dfs.namenode.name.dir</name>
            <value>/usr/local/hadoop/hdfs/namenode</value>
        </property>
        <property>
            <name>dfs.datanode.data.dir</name>
            <value>/usr/local/hadoop/hdfs/datanode</value>
        </property>
    </configuration>
```

Now, we will update `mapred-site.xml` file. It need to be edited **only on master node**

```
<configuration>
        <property>
            <name>mapreduce.jobtracker.address</name>
            <value>hadoop1:54311</value>
        </property>
    </configuration>
```

The last configuration will be in the file `slaves` in the folder `/usr/local/hadoop/etc/hadoop`. Add the host names or the ip addresses of the slave nodes

```
hadoop1
hadoop2
```

As `hadoop1` acts as both master and slave so we will add both the host names.

### 4.6.4 Formatting the Namenode

We are now done with all the configuration, so before starting the cluster we need to format the namenode. To do so, use the following command on the hadoop1(master) node terminal

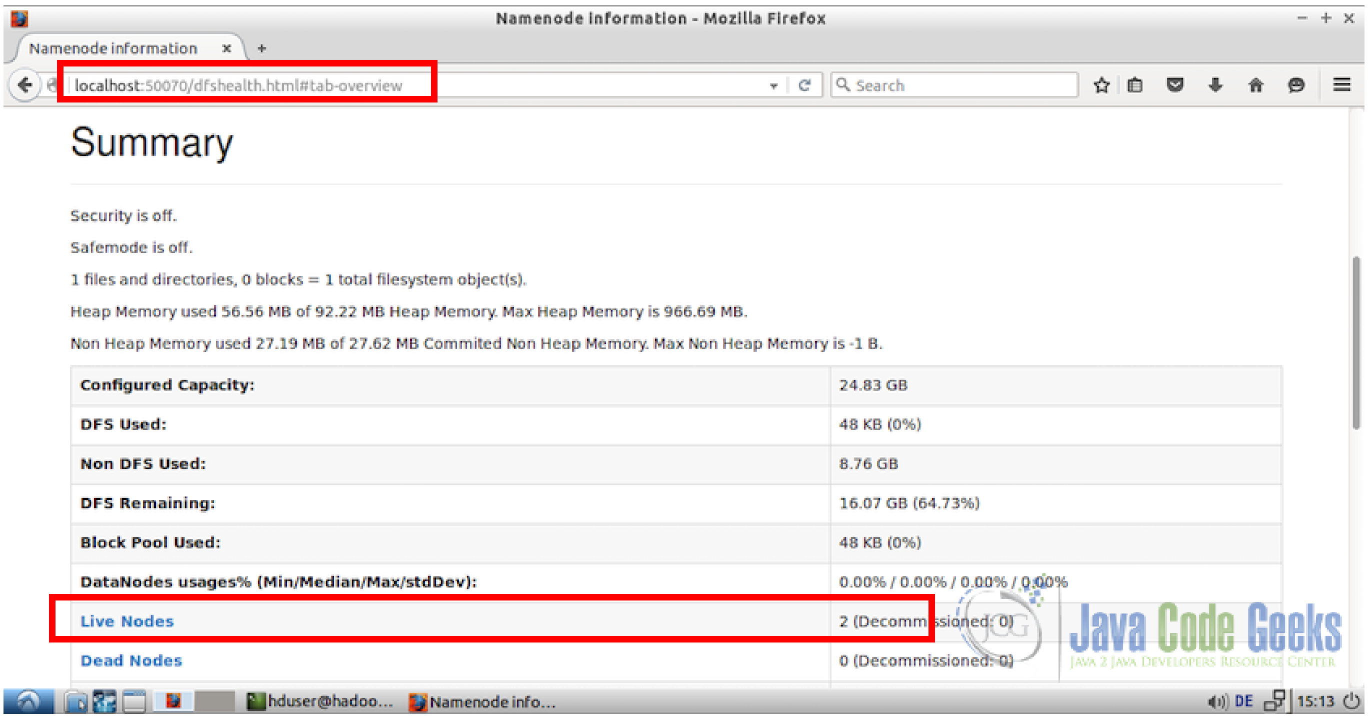
```
hdfs namenode -format
```

### 4.6.5 Start the Distributed Format System

Now it is time to start the distributed format system and start running the cluster. Following is the command to do so:

```
/usr/local/hadoop/sbin/start-dfs.sh
```

Once the dfs starts without any error, we can browse the web interface for the Namenode on `https://localhost:50070` on the master node



Namenode information - Mozilla Firefox

localhost:50070/dfshealth.html#tab-overview

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 56.56 MB of 92.22 MB Heap Memory. Max Heap Memory is 966.69 MB.

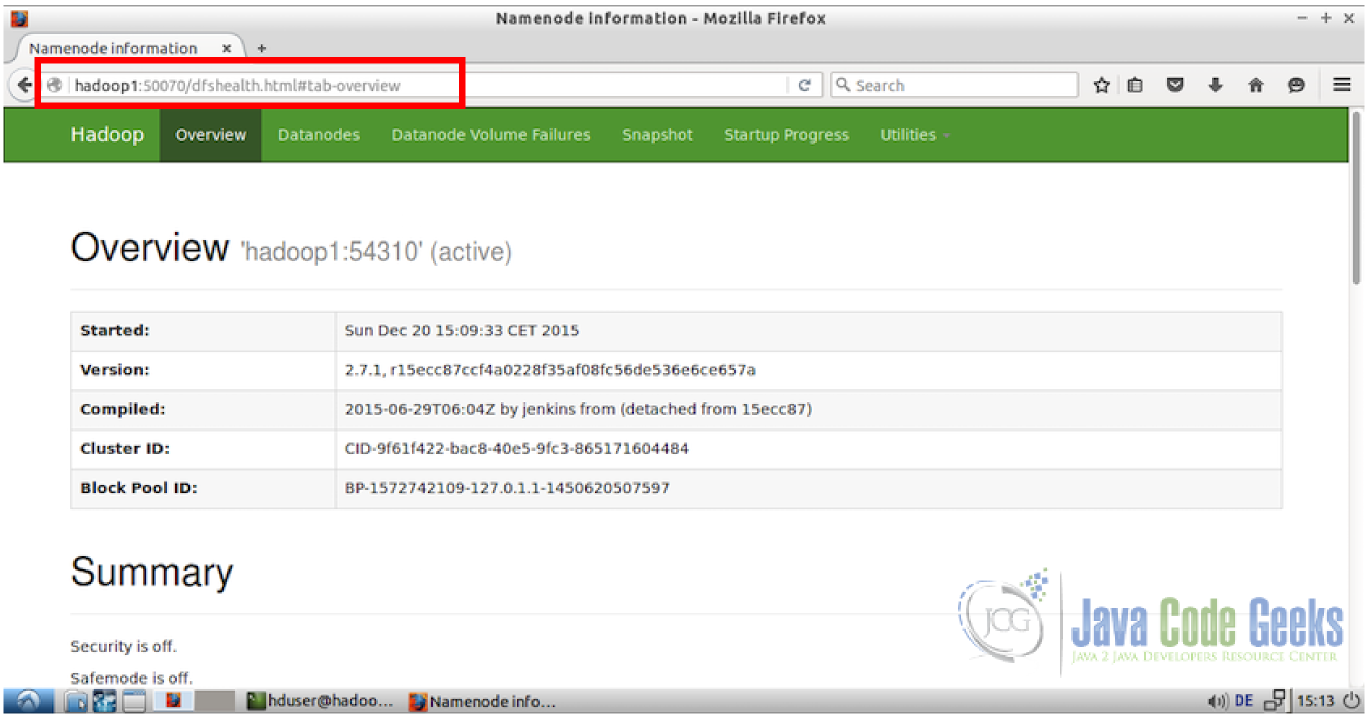
Non Heap Memory used 27.19 MB of 27.62 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

| | |
|---|---|
| **Configured Capacity:** | 24.83 GB |
| **DFS Used:** | 48 KB (0%) |
| **Non DFS Used:** | 8.76 GB |
| **DFS Remaining:** | 16.07 GB (64.73%) |
| **Block Pool Used:** | 48 KB (0%) |
| **DataNodes usages% (Min/Median/Max/stdDev):** | 0.00% / 0.00% / 0.00% / 0.00% |
| **Live Nodes** | 2 (Decommissioned: 0) |
| **Dead Nodes** | 0 (Decommissioned: 0) |

Figure 4.15: Hadoop Web Interface from Master Node

If you notice on the bottom of screenshot, there are two live node at the time what confirms that our cluster has two properly working nodes.

We can also access the web interface from any of the slave nodes but for those we have to use the master hostname or ip address. For example, from hadoop2(slave node) we can use the address `https://hadoop1:50070` to access the web interface.



Namenode information - Mozilla Firefox

hadoop1:50070/dfshealth.html#tab-overview

Overview 'hadoop1:54310' (active)

| | |
|---|---|
| **Started:** | Sun Dec 20 15:09:33 CET 2015 |
| **Version:** | 2.7.1, r15ecc87ccf4a0228f35af08fc56de536e6ce657a |
| **Compiled:** | 2015-06-29T06:04Z by jenkins from (detached from 15ecc87) |
| **Cluster ID:** | CID-9f61f422-bac8-40e5-9fc3-865171604484 |
| **Block Pool ID:** | BP-1572742109-127.0.1.1-1450620507597 |

Summary

Security is off.

Safemode is off.

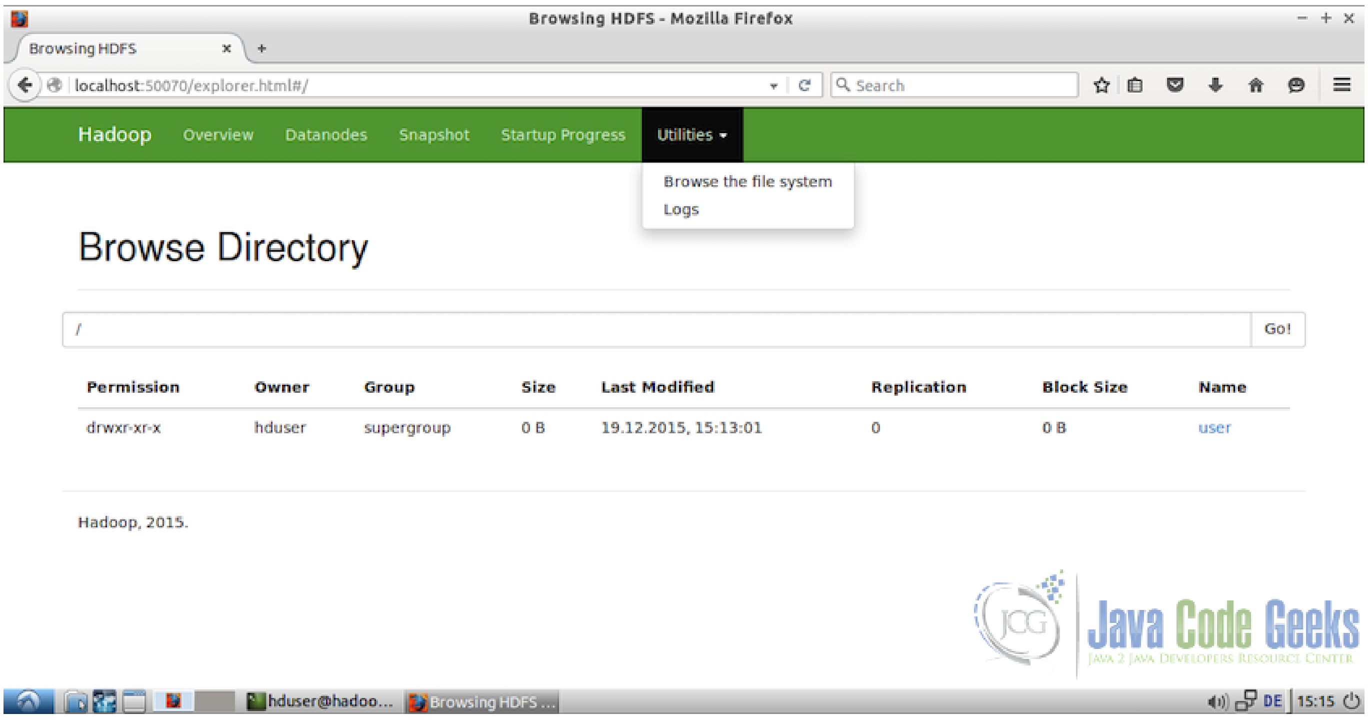
Figure 4.16: Hadoop Web Interface from the Slave node

### 4.6.6 Testing MapReduce Job

First of all, lets make the required HDFS directories and copy some input data for testing purpose

```
#Make the required directories
bin/hdfs dfs -mkdir /user
bin/hdfs dfs -mkdir /user/hduser
```

These directories can be accessed from the web interface also. To do so, go to the web interface, from the menu select *Utilities* and from dropdown select *Browse the file system*



Browse Directory

/

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| drwxr-xr-x | hduser | supergroup | 0 B | 19.12.2015, 15:13:01 | 0 | 0 B | user |

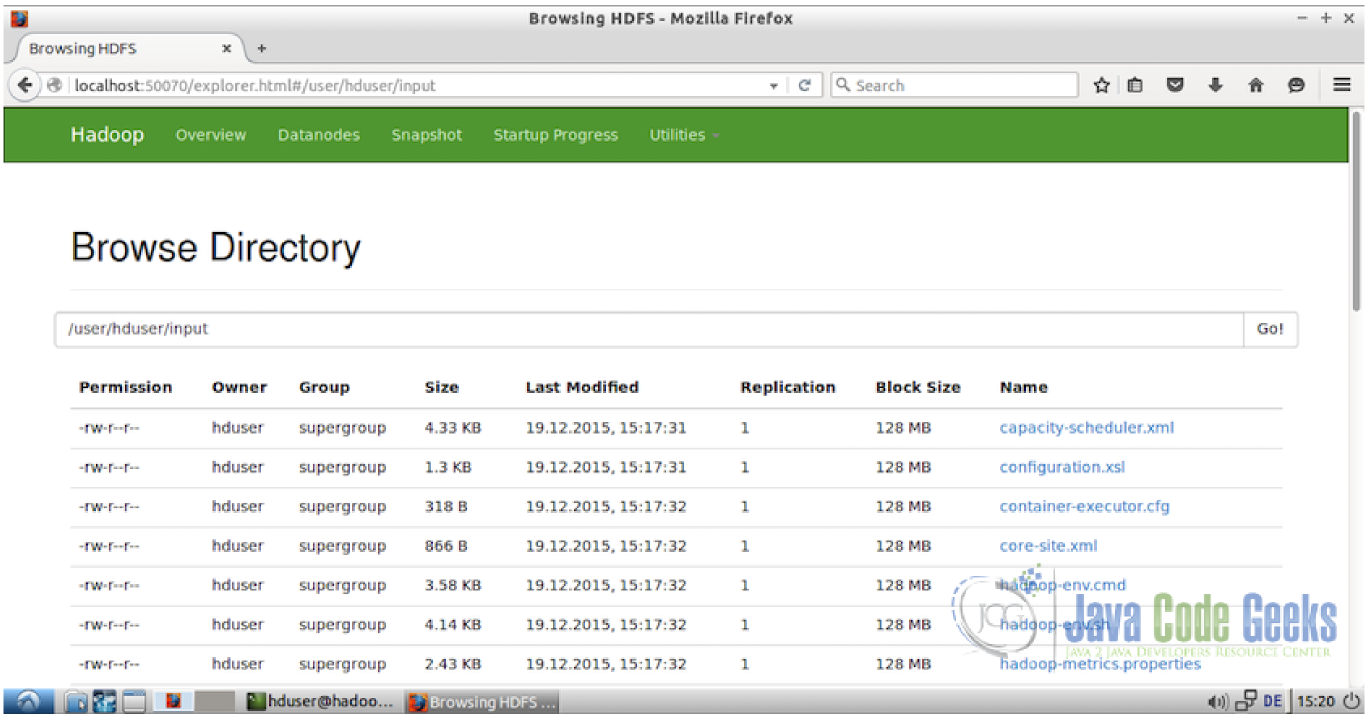
Hadoop, 2015.

Figure 4.17: Accessing directories in HDFS using web interface

Now, we can add some dummy files to the directory which we will use for the testing purpose. Lets ass the all the files from `etc/hadoop` folder

```
#Copy the input files into the distributed file system
/usr/local/hadoop/bin/hdfs dfs -put /usr/local/hadoop/etc/hadoop input
```

Following screenshot shows the files added to the directories `/user/hduser/input`



Browse Directory

/user/hduser/input

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| -rw-r--r-- | hduser | supergroup | 4.33 KB | 19.12.2015, 15:17:31 | 1 | 128 MB | capacity-scheduler.xml |
| -rw-r--r-- | hduser | supergroup | 1.3 KB | 19.12.2015, 15:17:31 | 1 | 128 MB | configuration.xsl |
| -rw-r--r-- | hduser | supergroup | 318 B | 19.12.2015, 15:17:32 | 1 | 128 MB | container-executor.cfg |
| -rw-r--r-- | hduser | supergroup | 866 B | 19.12.2015, 15:17:32 | 1 | 128 MB | core-site.xml |
| -rw-r--r-- | hduser | supergroup | 3.58 KB | 19.12.2015, 15:17:32 | 1 | 128 MB | hadoop-env.cmd |
| -rw-r--r-- | hduser | supergroup | 4.14 KB | 19.12.2015, 15:17:32 | 1 | 128 MB | hadoop-env.sh |
| -rw-r--r-- | hduser | supergroup | 2.43 KB | 19.12.2015, 15:17:32 | 1 | 128 MB | hadoop-metrics.properties |

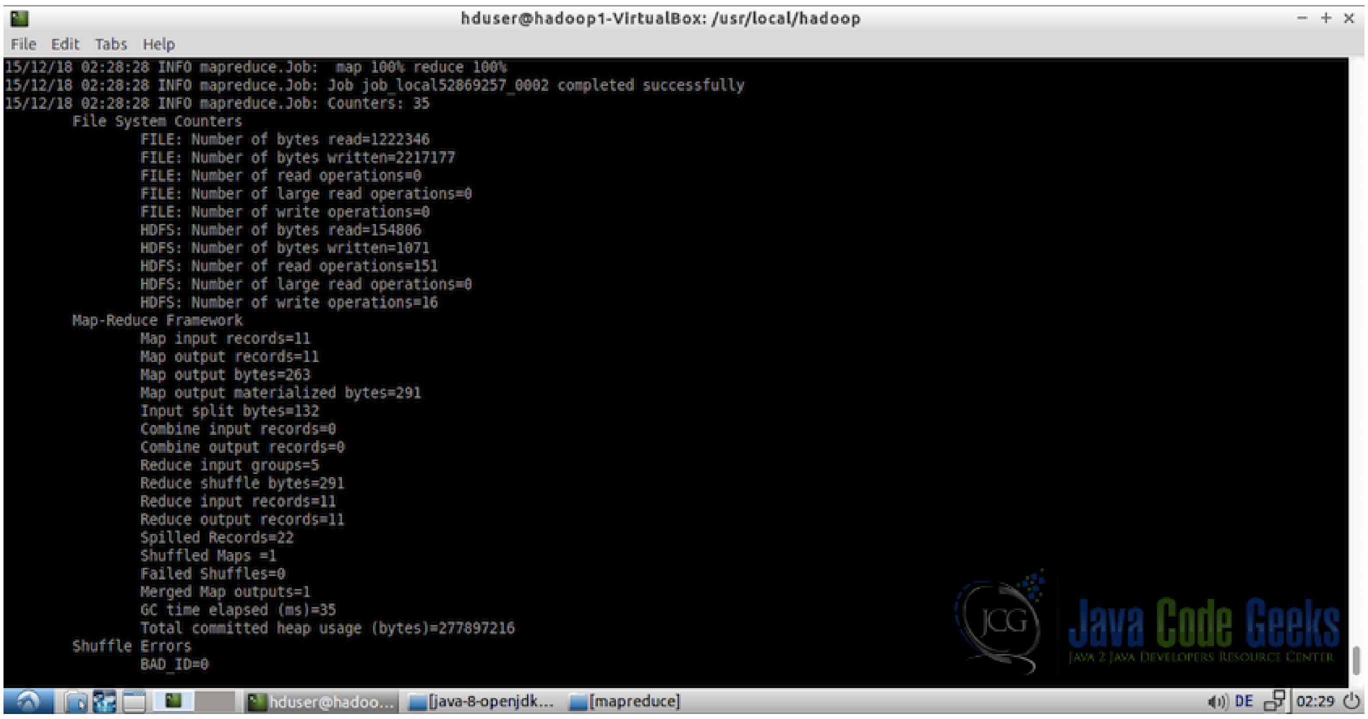
Figure 4.18: Browsing files in the HDFS

Run the MapReduce included in the hadoop package using the following command

```
/usr/local/hadoop/bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-example-2.7.1.jar ↩
    grep input output 'dfs[a-z.]+'
```

**Note:** For details on how MapReduce example works, refer to the article "Hadoop Hello World Example"

Following screenshot shows the output log of the test example:



```
15/12/18 02:28:28 INFO mapreduce.Job:  map 100% reduce 100%
15/12/18 02:28:28 INFO mapreduce.Job: Job job_local52869257_0002 completed successfully
15/12/18 02:28:28 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=1222346
                FILE: Number of bytes written=2217177
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=154806
                HDFS: Number of bytes written=1071
                HDFS: Number of read operations=151
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=16
        Map-Reduce Framework
                Map input records=11
                Map output records=11
                Map output bytes=263
                Map output materialized bytes=291
                Input split bytes=132
                Combine input records=0
                Combine output records=0
                Reduce input groups=5
                Reduce shuffle bytes=291
                Reduce input records=11
                Reduce output records=11
                Spilled Records=22
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=35
                Total committed heap usage (bytes)=277897216
        Shuffle Errors
                BAD_ID=0
```

Figure 4.19: Output of the test MapReduce example

We can now view the output file using

```
/usr/local/hadoop/bin/hdfs dfs -cat output/*
```

### 4.6.7 Stopping the Distributed Format System

We can now stop the dfs(distributed format system) using the following command:

```
/usr/local/hadoop/sbin/stop-dfs.sh
```

This brings us to the end of the setup and initial testing.

## 4.7 Conclusion

This brings us to the conclusion of this example. Hope this makes it a little more clear about how to set up Hadoop cluster on multiple machines. In case, a cluster need to be setup on multiple physical machines instead of virtual machines, the instructions are similar except steps containing 4.1 VM Network settings and 4.2 Cloning the Virtual Machine. For physical machines cluster, we can perform all other steps on the machines and everything should work smoothly.

## 4.8 Download configuration files

The configuration files which are modified and used for this example can be downloaded from here. Keep in mind that the modification done in these configuration files can be different based on the user network and other setting and may need to be changes accordingly. The package contains:

- hosts file

- sysctl.conf file
- Hadoop 1 folder (contains master node files)
  - core-site.xml
  - hdfs-site.xml
  - mapred-site.xml
  - slaves
- Hadoop 2 folder (contains slave note files)
  - core-site.xml
  - hdfs-site.xml

**Download** You can download all the above mentioned files from this example here: **HadoopClusterSetup**

# Chapter 5

# Distcp Example

In this example, we are going to show you how to copy large files in inter/intra-cluster setup of Hadoop using distributed copy tool.

## 5.1 Introduction

**DistCP** is the shortform of Distributed Copy in context of Apache Hadoop. It is basically a tool which can be used in case we need to copy large amount of data/files in inter/intra-cluster setup. In the background, DisctCP uses MapReduce to distribute and copy the data which means the operation is distributed across multiple available nodes in the cluster. This makes it more efficient and effective copy tool.

DistCP takes a list of files(in case of multiple files) and distribute the data between multiple Map tasks and these map tasks copy the data portion assigned to them to the destination.

## 5.2 Syntax and Examples

In this section, we will check the syntax of DistCP along with some examples.

### 5.2.1 Basic

Following is the basic syntac of distCp command.

```
hadoop distcp hdfs://namenode:port/source hdfs://namenode:port/destination
```

Following the `distcp` first argument should be the fully qualified address of the source including the namenode and the port number. Second argument should be the destination address. The basic syntax of `distcp` is quite easy and symple. It handles all the distribution and copying automatically using MapReduce.

If copying between the same cluster, the namenode and the port number of both source and destination will be same and in case of different cluster both will be different.

Example of basic `distcp`:

```
hadoop distcp hdfs://quickstart.cloudera:8020/user/access_logs hdfs://quickstart.cloudera ↩
    :8020/user/destination_access_logs
```

Following is the log of the command execution:

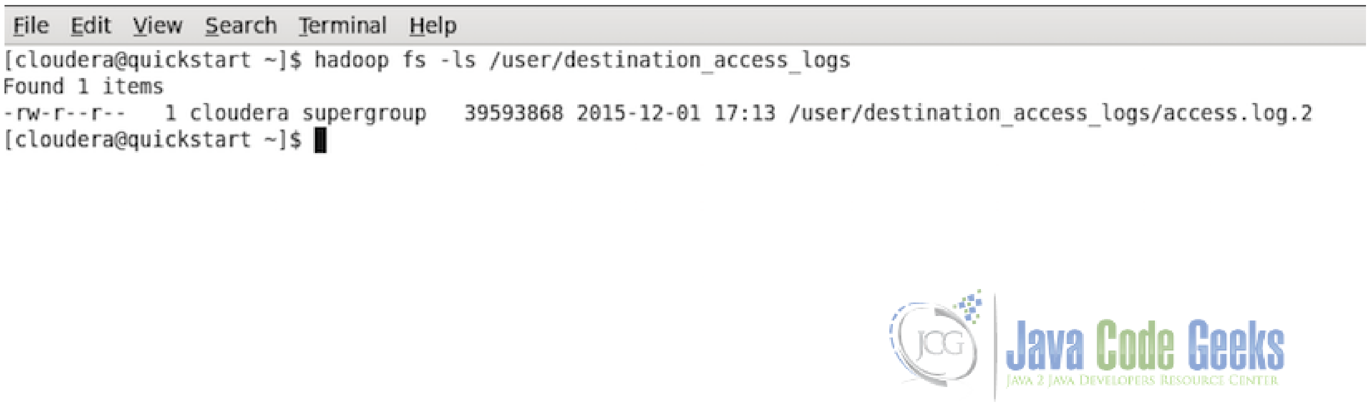
```
15/12/01 17:13:07 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=false,  ↩
    syncFolder=false, deleteMissing=false, ignoreFailures=false, maxMaps=20,  ↩
    sslConfigurationFile='null', copyStrategy='uniformsize', sourceFileListing=null,  ↩
    sourcePaths=[hdfs://quickstart.cloudera:8020/user/access_logs], targetPath=hdfs:// ↩
    quickstart.cloudera:8020/user/destination_access_logs, targetPathExists=false,  ↩
    preserveRawXattrs=false, filtersFile='null'}
15/12/01 17:13:07 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/12/01 17:13:08 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 2; dirCnt = 1
15/12/01 17:13:08 INFO tools.SimpleCopyListing: Build file listing completed.
15/12/01 17:13:08 INFO Configuration.deprecation: io.sort.mb is deprecated. Instead, use  ↩
    mapreduce.task.io.sort.mb
15/12/01 17:13:08 INFO Configuration.deprecation: io.sort.factor is deprecated. Instead,  ↩
    use mapreduce.task.io.sort.factor
15/12/01 17:13:08 INFO tools.DistCp: Number of paths in the copy list: 2
15/12/01 17:13:08 INFO tools.DistCp: Number of paths in the copy list: 2
15/12/01 17:13:08 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/12/01 17:13:09 INFO mapreduce.JobSubmitter: number of splits:2
15/12/01 17:13:09 INFO mapreduce.JobSubmitter: Submitting tokens for job:  ↩
    job_1449017643353_0001
15/12/01 17:13:10 INFO impl.YarnClientImpl: Submitted application  ↩
    application_1449017643353_0001
15/12/01 17:13:10 INFO mapreduce.Job: The url to track the job: https://quickstart.cloudera ↩
    :8088/proxy/application_1449017643353_0001/
15/12/01 17:13:10 INFO tools.DistCp: DistCp job-id: job_1449017643353_0001
15/12/01 17:13:10 INFO mapreduce.Job: Running job: job_1449017643353_0001
15/12/01 17:13:20 INFO mapreduce.Job: Job job_1449017643353_0001 running in uber mode :  ↩
    false
15/12/01 17:13:20 INFO mapreduce.Job:  map 0% reduce 0%
15/12/01 17:13:32 INFO mapreduce.Job:  map 50% reduce 0%
15/12/01 17:13:34 INFO mapreduce.Job:  map 100% reduce 0%
15/12/01 17:13:34 INFO mapreduce.Job: Job job_1449017643353_0001 completed successfully
15/12/01 17:13:35 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=228770
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=39594819
                HDFS: Number of bytes written=39593868
                HDFS: Number of read operations=28
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=7
        Job Counters
                Launched map tasks=2
                Other local map tasks=2
                Total time spent by all maps in occupied slots (ms)=20530
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=20530
                Total vcore-seconds taken by all map tasks=20530
                Total megabyte-seconds taken by all map tasks=21022720
        Map-Reduce Framework
                Map input records=2
                Map output records=0
                Input split bytes=276
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=94
                CPU time spent (ms)=1710
                Physical memory (bytes) snapshot=257175552
                Virtual memory (bytes) snapshot=3006455808
                Total committed heap usage (bytes)=121503744
        File Input Format Counters
                Bytes Read=675
        File Output Format Counters
                Bytes Written=0
        org.apache.hadoop.tools.mapred.CopyMapper$Counter
                BYTESCOPIED=39593868
                BYTESEXPECTED=39593868
                COPY=2
```

Line number 35 in the log indicate the number of map tasks executed, which is 2 in this case.

To check if the copy was successful, we can run the following command in HDFS:

```
hadoop fs -ls /user/destination_access_logs
```

Below is the output if the copy was successful and data is present in the destination folder:



```
[cloudera@quickstart ~]$ hadoop fs -ls /user/destination_access_logs
Found 1 items
-rw-r--r--   1 cloudera supergroup   39593868 2015-12-01 17:13 /user/destination_access_logs/access.log.2
[cloudera@quickstart ~]$ 
```

Figure 5.1: Screenshot displaying the out of the hadoop fs command

**Note:** When the files are copied between the two different clusters, HDFS version on both the clusters should be same or in case of different versions, the higher version should be backward compatible.

### 5.2.2 Multiple Sources

In case there are multiple file sources and it need to go to the same destination sources, then all the sources can be passed as the arguments as shown in the example syntax below:

```
hadoop distcp hdfs://namenode:port/source1 hdfs://namenode:port/source2 hdfs://namenode: ↩
    port/source3 hdfs://namenode:port/destination
```

So the files from all the three sources will be copied to the destination specified.

There is another alternative if there are many sources and writing long command becomes an issue. Following is the alternative approach:

```
hadoop distcp -f hdfs://namenode:port/sourceListFile hdfs://namenode:port/destination
```

where, the `sourceListFile` is a simple file containing the list of all the sources. In this case, the source list file need to be passed with the flag `-f` which indicates that the source is not the file to be copied but a file which contains all the sources.

**Note:** When `distcp` is used with multiple sources, in case if the sources collide, `distcp` will abort the copy with an error message. But in case of collisions at the destination, copying is not aborted but the collision is resolved as per the options specified. If no options are specified, default is that the files already existing at the destination are skipped.

### 5.2.3 Update and Overwrite Flag

As the names indicate, `update` will update the files at the destination folder but only if the update conditions are met. Conditions for update to be performed are that update checks id the destination have the same file name, if the file size and content are same as the source file, if everything is same then the files are not updated but if different the files are updated from the source to destination.

`overwrite` will overwrite the files at the destination id the destination have same file name, if so, then the file will be overwritten.

```
hadoop distcp -update hdfs://namenode:port/source hdfs://namenode:port/destination
```

```
hadoop distcp -overwrite hdfs://namenode:port/source hdfs://namenode:port/destination
```

### 5.2.4 Ignore Failures Flag

In `distcp` is any map task fails, it stops the other map tasks also and the copying process halts completely with an error. In case, there is the requirement to continue copying other chunks of data even if one or more map tasks fails we have an ignore failures flag i.e. `-i`.

```
hadoop distcp -i hdfs://namenode:port/source hdfs://namenode:port/destination
```

### 5.2.5 Maximum Map Tasks

If the user wants to specify the maximum number of map tasks which can be assigned for `distcp` execution, there is another flag `-m <max_num>`.

```
hadoop distcp -m 5 hdfs://namenode:port/source hdfs://namenode:port/destination
```

This example command will assign maximum of 5 map tasks to the `distcp` command.

Example of setting maximum map tasks in `distcp`:

```
hadoop distcp -m 1 hdfs://quickstart.cloudera:8020/user/access_logs hdfs://quickstart. ↩
    cloudera:8020/user/destination_access_logs_3
```

Here we limit the map task to be 1. From the above example log output we know that default map tasks for this file data is 2.

Below is the log of the command execution:

```
15/12/01 17:19:33 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=false,  ↩
    syncFolder=false, deleteMissing=false, ignoreFailures=false, maxMaps=1,  ↩
    sslConfigurationFile='null', copyStrategy='uniformsize', sourceFileListing=null,  ↩
    sourcePaths=[hdfs://quickstart.cloudera:8020/user/access_logs], targetPath=hdfs:// ↩
    quickstart.cloudera:8020/user/destination_access_logs_3, targetPathExists=false,  ↩
    preserveRawXattrs=false, filtersFile='null'}
15/12/01 17:19:33 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/12/01 17:19:34 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 2; dirCnt = 1
15/12/01 17:19:34 INFO tools.SimpleCopyListing: Build file listing completed.
15/12/01 17:19:34 INFO Configuration.deprecation: io.sort.mb is deprecated. Instead, use  ↩
    mapreduce.task.io.sort.mb
15/12/01 17:19:34 INFO Configuration.deprecation: io.sort.factor is deprecated. Instead,  ↩
    use mapreduce.task.io.sort.factor
15/12/01 17:19:34 INFO tools.DistCp: Number of paths in the copy list: 2
15/12/01 17:19:34 INFO tools.DistCp: Number of paths in the copy list: 2
15/12/01 17:19:34 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/12/01 17:19:35 INFO mapreduce.JobSubmitter: number of splits:1
15/12/01 17:19:35 INFO mapreduce.JobSubmitter: Submitting tokens for job:  ↩
    job_1449017643353_0003
```

```
15/12/01 17:19:35 INFO impl.YarnClientImpl: Submitted application ↩
    application_1449017643353_0003
15/12/01 17:19:35 INFO mapreduce.Job: The url to track the job: https://quickstart.cloudera ↩
    :8088/proxy/application_1449017643353_0003/
15/12/01 17:19:35 INFO tools.DistCp: DistCp job-id: job_1449017643353_0003
15/12/01 17:19:35 INFO mapreduce.Job: Running job: job_1449017643353_0003
15/12/01 17:19:44 INFO mapreduce.Job: Job job_1449017643353_0003 running in uber mode :  ↩
    false
15/12/01 17:19:44 INFO mapreduce.Job:  map 0% reduce 0%
15/12/01 17:19:52 INFO mapreduce.Job:  map 100% reduce 0%
15/12/01 17:19:52 INFO mapreduce.Job: Job job_1449017643353_0003 completed successfully
15/12/01 17:19:52 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=114389
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=39594404
                HDFS: Number of bytes written=39593868
                HDFS: Number of read operations=20
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=5
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=5686
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=5686
                Total vcore-seconds taken by all map tasks=5686
                Total megabyte-seconds taken by all map tasks=5822464
        Map-Reduce Framework
                Map input records=2
                Map output records=0
                Input split bytes=138
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=45
                CPU time spent (ms)=1250
                Physical memory (bytes) snapshot=123002880
                Virtual memory (bytes) snapshot=1504280576
                Total committed heap usage (bytes)=60751872
        File Input Format Counters
                Bytes Read=398
        File Output Format Counters
                Bytes Written=0
        org.apache.hadoop.tools.mapred.CopyMapper$Counter
                BYTESCOPIED=39593868
                BYTESEXPECTED=39593868
                COPY=2
```

Map tasks in this example is maximum 1 as indicated in the line 34 of the above log.

## 5.3 Final Notes

In this example, we saw the use of `distcp` command in Apache Hadoop to copy large amount of data. For more help and details about `distcp` command and all the options available, use the following command to check the built-in help:

```
hadoop distcp
```

# Chapter 6

# Distributed File System Explained

In this example, we will discuss Apache Hadoop Distributed File System(HDFS), its components and the architecture in detail. HDFS is one of the core components of Apache Hadoop ecosystem also.

## 6.1 Introduction

Apache Hadoop provides a distributed filesystem and a framework for the transformation of large data sets using the MapReduce paradigm. HDFS is designed to store very large data sets reliably while running on commodity hardware. It is fault-tolerant and provides high thoughput access to the data stored. While the interface of HDFS is patterned after the Unix filesystem, but it relaxes a few POSIX requirements to improve the performance of the application it targets to address and to provide streaming access to the data stored in the file system.

## 6.2 HDFS Design

Following are the properties of HDFS which makes it different from other file systems and which make HDFS capable of handling very large amount of data reliably.

### 6.2.1 System failures

HDFS is designed to work on a set of commodity hardware. System failures are considered a norm. As there are a large number of components on which HDFS rely on, considering these components have a non-trivial probability of failure will also result in one component or the other failing all the time. So HDFS is designed to detect the failures and perform automatic recovery in order to provide required performance is one of the core properties of HDFS.

### 6.2.2 Can handle large amount of data

HDFS is designed to be used with applications which depend on a large amount of data. This data can be in gigabytes, terabytes or petabytes also. So HDFS is tuned to support such large data sets and to scale to a large cluster of systems to store this data without compromising with the data thoughput.

### 6.2.3 Coherency Model

HDFS is tuned to address the applications which require to write data once or at maximum only a few times and read the data a lot more. As these applications are assumed to rely on "write once read many times" model, it simplifies the data coherency issues and allow HDFS to provide high thoughput data access.

### 6.2.4 Portability

HDFS is designed to be portable across heterogeneous hardware and software platforms. Which makes the adaptation of HDFS very easy and it became the platform of choice for the application dependent on distributed large set of data.

## 6.3 HDFS Nodes

There are two main components of HDFS **NameNode** and **DataNode**.

### 6.3.1 NameNode

HDFS follows a master-slave architecture in which NameNode is node which acts as the master node. One HDFS cluster consists of only one NameNode. The main functionality of NameNode is to manage the file system namespace and control the client authentication to the files stored in the HDFS cluster. It also handles the mapping of the data stored in different DataNodes.

### 6.3.2 DataNode

DataNode are the nodes which as the name indicates stores the actual data in the cluster. There are multiple DataNodes in the cluster, usually the number of DataNodes is same as the node of hardware nodes in the cluster. DataNode serve the read and write requests from the clients and also handles operation related to blocks of data like creation of blocks, deletion and replication of blocks.

## 6.4 HDFS Architecture

In this section we will understand the basic architecture of Hadoop Distributed File System(HDFS).

### 6.4.1 Working of NameNode and DataNode

HDFS is block-structured file system, that means all the individual files are divided into small blocks of data having a fixed block size. These blocks are then stored across the cluster of machines in the DataNodes. The NameNode handles the functions like opening, closing and renaming files or directories. NameNode as mentioned above also handles the mapping of the data in the cluster that means NameNode keeps track of which block of data is stored on which DataNode and how the replication of this data is handled.

### 6.4.2 HDFS Namespace

HDFS namespace defines how the data is stored and accessed in the cluster. HDFS supports the traditional hierarchical organization of the files and directories. It also supports almost all the required functions to handle the namespace operations like creation or removal of files or directories, moving files/Directories from one place to another etc.

As we discussed in section 3, NameNode is the component which maintains the HDFS file system namespace. Any operation on the data like creation or deletion of files, displacement of files or directories are maintained in the NameNode.

### 6.4.3 Data Replication

As HDFS is designed to store large amount of data reliably and securely on a set of commodity hardware. As this hardware is prone to easy failure, HDFS need to handle the data in a way that it can be retrieved easily in the event of hardware failure of one or more systems. HDFS uses data replication as the strategy to provide fault-tolerance feature. The application using the HDFS can configure the replication factor as well as the block size of data as per the requirement.

Now the question arises how the replication is decided, what if all the replicas are in a single rack in the cluster and the whole rack fails. HDFS tries to maintain the rack aware replication strategy which in fact needs a lot of tuning and experience. A simple but non-optimal policy is to place each replica of the block on a unique rack so that in case of a whole rack failure. Atleast the replica os the block is safe in another rack.

In most of the production systems the replication factor of three is used. In those cases. HDFS uses a slight different version of unique rack policy. It usually places one replica on a node in the local rack, another on a node on a completely different remote rack and the third one on a different node on the remote rack. This policy improves the write speed by cutting the inter-rack transfer time while writing on two different racks instead of three. This provides us backup in case of node failures and also in case of rack failures. This policy improves the write performance without and compromising data reliability.

### 6.4.4 Failures

The main objective and goal of Hadoop Distributed File System(HDFS) is to provide access to data reliably even in case of failures. As failures are more of norm in the commodity hardware cluster than an exception, HDFS needs a strategy to handle the failures. The three common types of failures are:

- NameNode failure
- DataNode failure
- Network partitions

Each and every DataNode in the cluster sends a periodic message to NameNode, this message is called heartbeat. This heartbeat conveys to NameNode that the particular DataNode is working fine and is live. Now in case of DataNode failures, there will be no heartbeats from the DataNode to the NameNode. Similarly in case of network partition also a subset of DataNodes may loose its connection to the NameNode and stops sending Heartbeats. Once the NameNode stop getting heartbeats from a particular DataNode or a set of DataNodes, it declares those nodes to be dead and then start the procedure to check the damage which include checking if all the blocks which are in dead DataNodes still have the sufficient number of replicas, if not then it starts the process to create re-replicas to attain the minimum number of replicas configured in the applicaion.

The NameNode failures are more serious as NameNode system is the only single point of failure for the complete HDFS cluster. If the NameNode system fails, the whole cluster is useless and it needs a manual intervention and another NameNode need to be setup.

### 6.4.5 Data Accessibility

Now in order to allow applications to access the data stored in an HDFS cluster, it provides a Java API for applications to use. A C language wrapper is also provided over the Java API if C language need to be used.

Besides Java and C API, HDFS also provides an option to access the HDFS data through web browser over the TCP port which can be configured in the settings of HDFS.

Third accessibility option is to use the file system shell. HDFS also provides a command line interface called FS Shell that let a user interact with the data in HDFS. The syntax for this command line interface is similar to the Linux shell commands. For example:

```
#To make a new directory
hadoop fs -mkdir /user1/project1

#List the content of the file
hadoop fs -ls /user1/project1

#Upload a file from local system to HDFS
hadoop fs -put Desktop/textfile.txt /user1/project1
```

For more examples and explanation of the FS Shell commands, you can check the article Apache Hadoop FS Commands Example

## 6.5 Configuring HDFS

Configuration of HDFS is very easy and it does not take much time to set up HDFS cluster. All the configuration files for HDFS are by default included in the Hadoop package and can be directly configured.

**Note:** We assume that the Hadoop package is already downloaded, unzipped and placed in the desired directory. In this article, we will discuss just the required configurations for HDFS. For detailed articles on how to setup Hadoop and Hadoop cluster. Follows the following tutorials:

- How to Install Apache Hadoop on Ubuntu
- Apache Hadoop Cluster Setup Example(with Virtual Machines)

### 6.5.1 Configuring HDFS

The HDFS is configured using the set of XML files which are by default present in the Hadoop configuration directory. This configuration directory is present in the root of the Hadoop folder and is named `conf`.

First of all, we will modify the file `conf/hadoop-sites.xml` and we need to setup three properties in this file i.e. `fs.default.name`, `dfs.data.dir`, `dfs.name.dir`

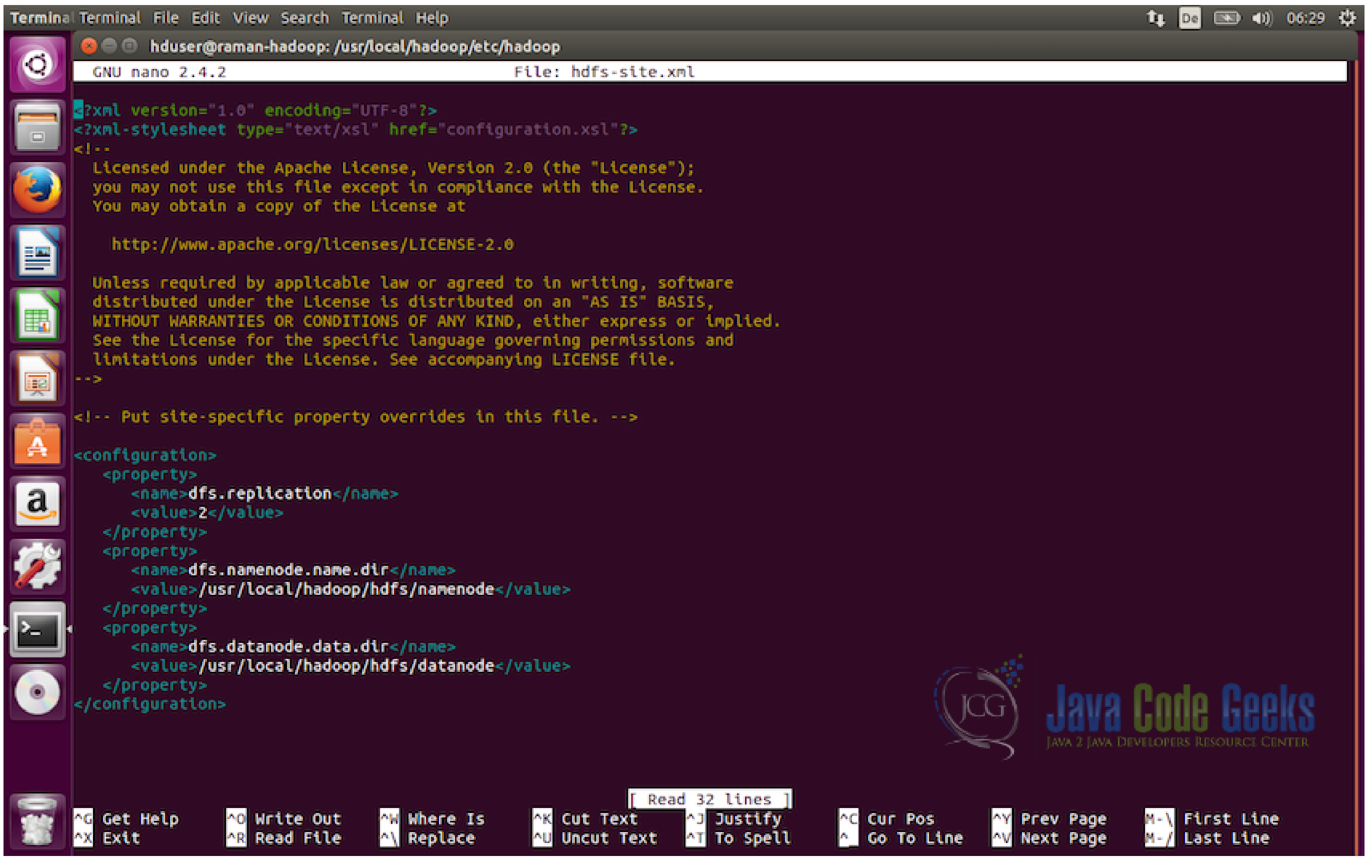
To modify the file open the file in the editor and add the following lines of code:

```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>2</value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>/usr/local/hadoop/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>/usr/local/hadoop/hdfs/datanode</value>
   </property>
</configuration>
```

First configuration we set here is `dfs.replication` which sets the replication factor to be used by the distributed file system. In this case we have set it up to be two.

The next configuration is to define the NameNode path i.e. `dfs.namenode.name.dir` and the value here need to be the directory to store the namenode information.

The third and the last configuration we need to set up is defining the path for the DataNode i.e. `dfs.datanode.data.dir` which will define the path to the directory to store the datanode information.


Terminal Terminal File Edit View Search Terminal Help
06:29
hduser@raman-hadoop: /usr/local/hadoop/etc/hadoop

```
  GNU nano 2.4.2                         File: hdfs-site.xml

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
      <name>dfs.replication</name>
      <value>2</value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>/usr/local/hadoop/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>/usr/local/hadoop/hdfs/datanode</value>
   </property>
</configuration>

                                            [ Read 32 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page   M-\ First Line
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  ^V Next Page   M-/ Last Line
```

Figure 6.1: Updating hdfs-site.xml

**Note: Make sure the directory where the namenode and datanode directory will be created and data will be stored is owned by the user which will run Hadoop. So that the user have read and write permission in the directory.**

### 6.5.2 Formating NameNode

Now the next step is to format the NameNode that we just configured. Following command is used to format the NameNode:

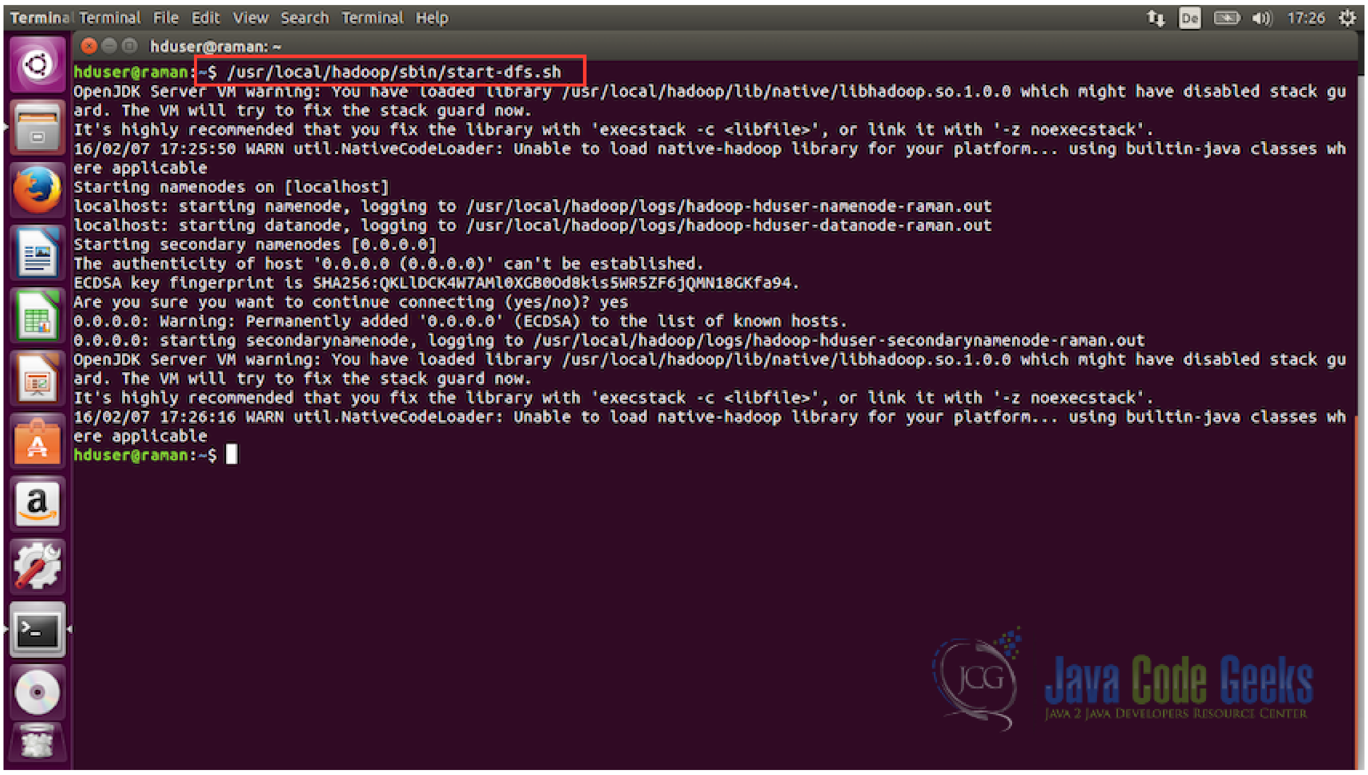
```
hdfs namenode -format
```

This command should be executed without any error on the console output. If it is executed without any errors, we are good to start the Apache Hadoop instance on our Ubuntu system.

### 6.5.3 Starting the HDFS

Now we are ready to start the Hadoop File System. To start HDFS, use the following command to run the `start-dfs.sh` file:

```
/usr/local/hadoop/sbin/start-dfs.sh
```



```
hduser@raman:~$ /usr/local/hadoop/sbin/start-dfs.sh
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 17:25:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-raman.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-raman.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:QKLlDCK4W7AMl0XGB0Od8kis5WR5ZF6jQMN18GKfa94.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-raman.out
OpenJDK Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack gu
ard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/02/07 17:26:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
hduser@raman:~$
```

Figure 6.2: Starting HDFS

Once this script is executed without any errors, HDFS will be up and running.

## 6.6 Interacting with HDFS using Shell

Now we will see some commands which are necessary to interact with HDFS using shell. In this section we will see just the basic introductory commands and will use only the command line interface. The commands which communicate with the cluster are present in the script `bin/hadoop`. This script loads the Hadoop package with the Java Virtual Machine(JVM), followed by the execution of the user command.

### 6.6.1 Creating a directory

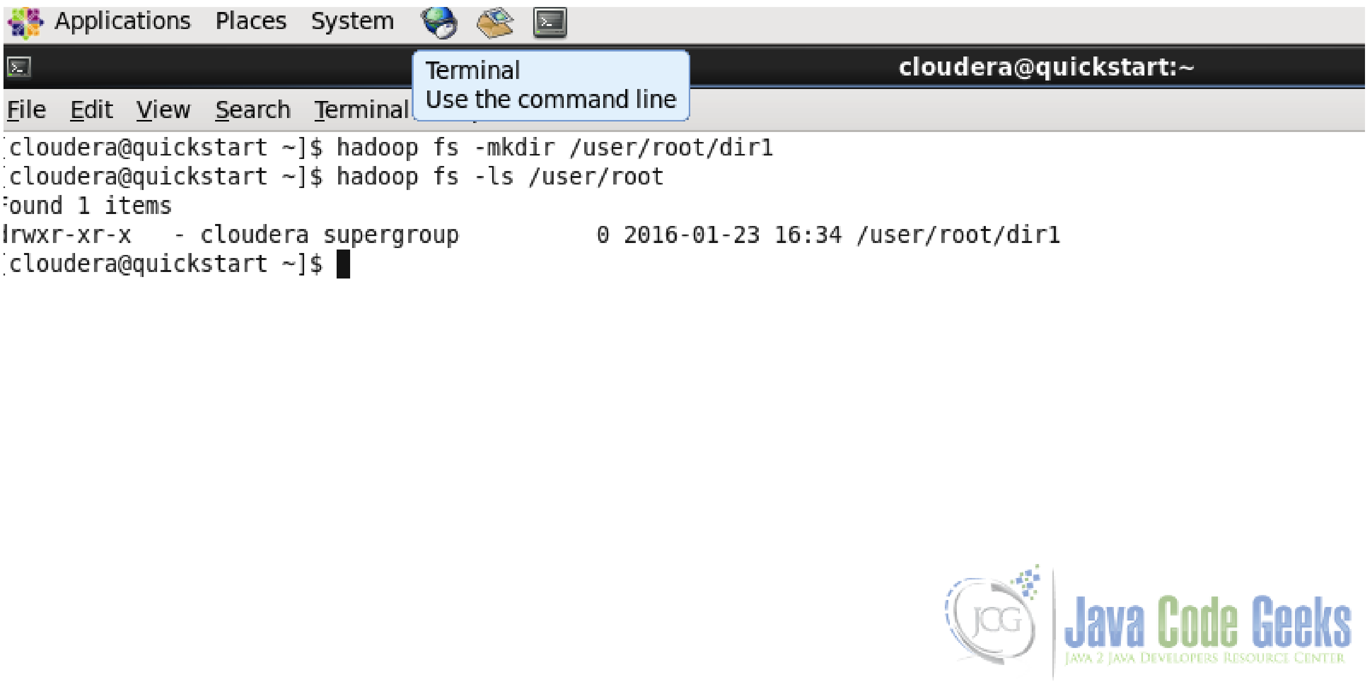
Usage:

```
hadoop fs -mkdir
```

Example:

```
hadoop fs -mkdir /user/root/dir1
```

Command in the second line is for listing the content of a particular path. We will see this command in the next sub-section. We can see in the screenshot that `dir1` is created



```
[cloudera@quickstart ~]$ hadoop fs -mkdir /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root
Found 1 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:34 /user/root/dir1
[cloudera@quickstart ~]$
```

Figure 6.3: Create Directory in Hadoop FS

### 6.6.2 List the content of the directory

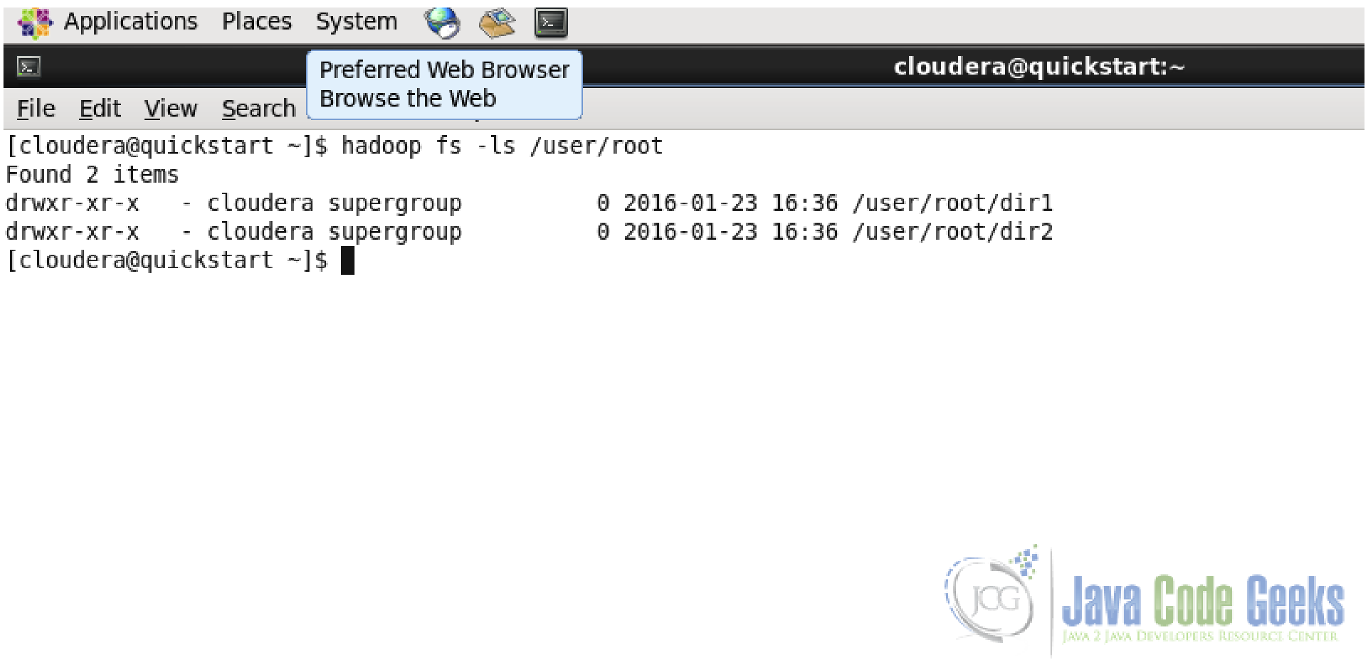
Usage:

```
hadoop fs -ls
```

Example:

```
hadoop fs -ls /user/root/
```

The command is similar to the `ls` command of the unix shell.



```
[cloudera@quickstart ~]$ hadoop fs -ls /user/root
Found 2 items
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir1
drwxr-xr-x   - cloudera supergroup          0 2016-01-23 16:36 /user/root/dir2
[cloudera@quickstart ~]$
```

Figure 6.4: Listing the files and directories

### 6.6.3 Upload a file in HDFS

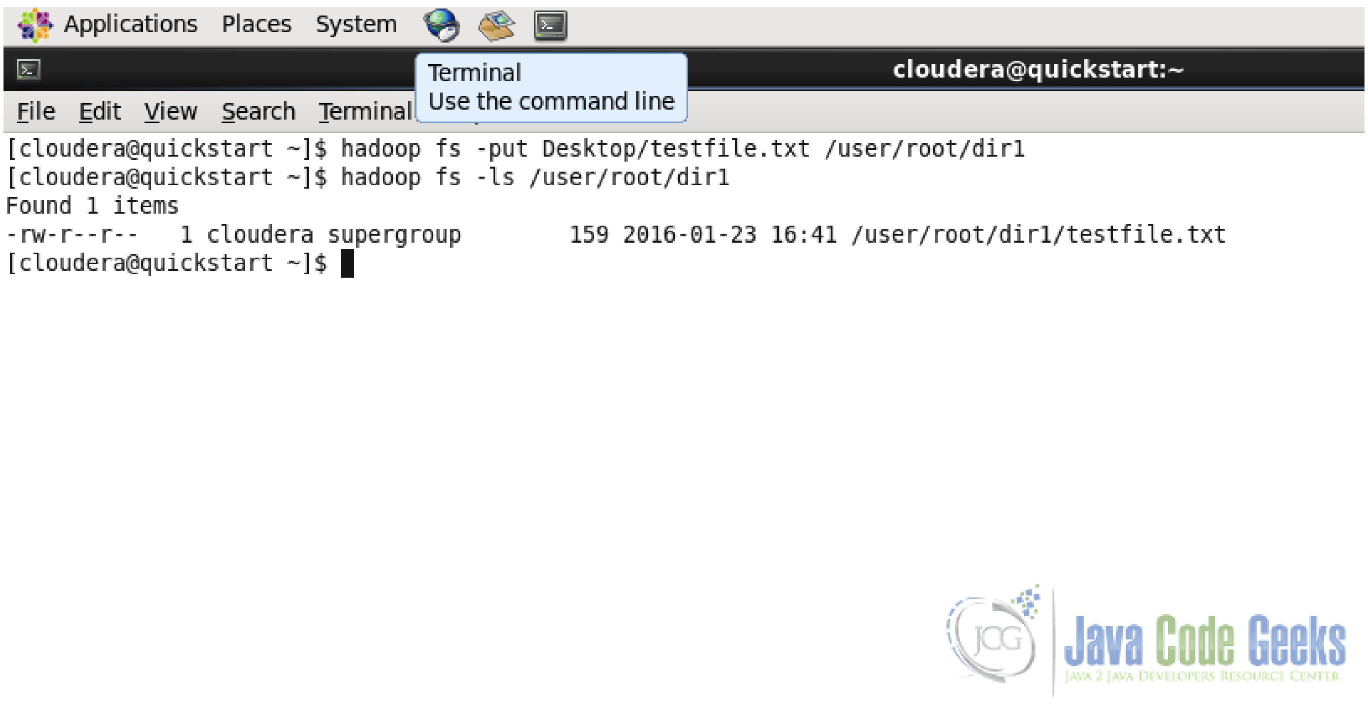
Command is used to copy one or multiple files from local system to the Hadoop File System. Usage:

```
hadoop fs -put  ...
```

Example:

```
hadoop fs -put Desktop/testfile.txt /user/root/dir1/
```

In the screenshot below, we `put` the file `testfile.txt` from `Desktop` of the Local File System to the Hadoop File System at the destination `/user/root/dir1`



```
[cloudera@quickstart ~]$ hadoop fs -put Desktop/testfile.txt /user/root/dir1
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/dir1
Found 1 items
-rw-r--r--   1 cloudera supergroup        159 2016-01-23 16:41 /user/root/dir1/testfile.txt
[cloudera@quickstart ~]$
```

Figure 6.5: Uploading the file to Hadoop FS

### 6.6.4 Download a file from HDFS

Download the file from HDFS to the local file system.

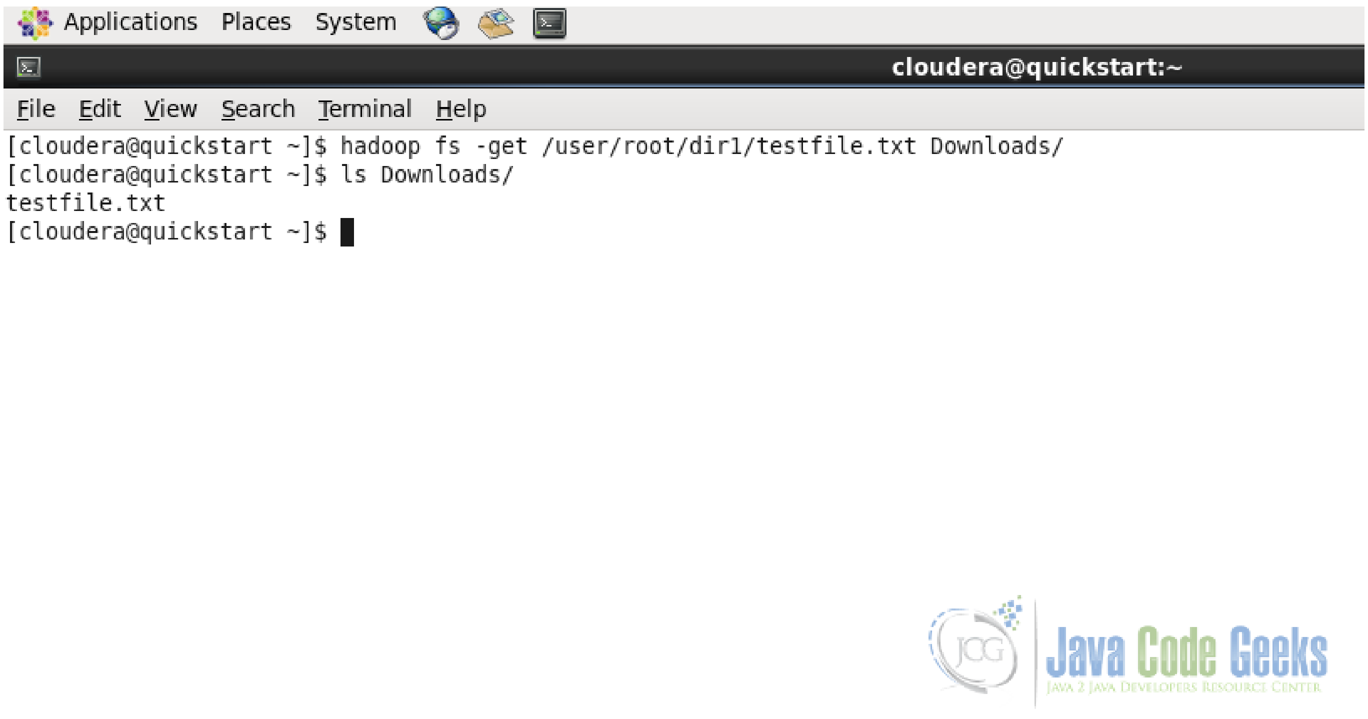
Usage:

```
hadoop fs -get
```

Example:

```
hadoop fs -get /user/root/dir1/testfile.txt Downloads/
```

As with the put command, get command `gets` or downloads the file from Hadoop File System to the Local File System in the `Downloads` folder.


Applications Places System
cloudera@quickstart:~
File Edit View Search Terminal Help
Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

```
[cloudera@quickstart ~]$ hadoop fs -get /user/root/dir1/testfile.txt Downloads/
[cloudera@quickstart ~]$ ls Downloads/
testfile.txt
[cloudera@quickstart ~]$ 
```

Figure 6.6: Download the file from Hadoop FS

**Note:** For details about the file system commands and for example of other important commands, refer to the article Apache Hadoop FS Commands Example or you can check the complete documentation of shell commands on the Apache Hadoop website in the documentation here: File System Shell Commands and HDFS Commands Guide

## 6.7 Interacting with HDFS using MapReduce

As we discussed that HDFS is a base component of Hadoop and MapReduce. Hadoop MapReduce jobs fetch data from the HDFS and stores the final result data in the HDFS.

Hadoop also provides a Java API using which we can execute HDFS functionality in out Java Application is required. In this section, we will see how to consume Java API in java code.

```
package com.javacodegeeks.examples.HDFSJavaApi;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.Path;

/**
 * Example application to show how the HDFS file system Java API works
 *
 * @Author Raman Jhajj
 */
public class App
{
        public static final String filename ="dummy.txt";
        public static final String message = "This is the dummy text for test the write to ↩
            file operation of HDFS";

    public static void main( String[] args ) throws IOException
    {
        //Get the file system instance
        Configuration configuration = new Configuration();
        FileSystem fs = FileSystem.get(configuration);

        Path filenamePath = new Path(filename);

        try {
                if(fs.exists(filenamePath)) {
                        //Delete Example
                        fs.delete(filenamePath, true);
                }

                //Write example
                FSDataOutputStream out = fs.create(filenamePath);
                out.writeUTF(message);
                out.close();

                //Read example
                FSDataInputStream in = fs.open(filenamePath);
                String messageIn = in.readUTF();
                System.out.println(messageIn);
                in.close();

                //Rename the file
                if(fs.exists(filenamePath)) {
                        Path renameFilenamePath = new Path("renamed_" + filename);
                        fs.rename(filenamePath, renameFilenamePath);
                }

        } catch(IOException ex) {
                System.out.println("Error: " + ex.getMessage());
        }
    }
}
```

This code above creates a file named `dummy.txt`, writes dummy message into this file.

- Line no. 24-25 creates an abstract `FileSystem` object with `Configuration` object. Configuration object uses default parameters in this case as we have not defined any parameters.
- Line no. 30-33 checks if the file already exists in HDFS and if it does exist, it tries deleting the file. This example introduces us to two methods available in the file system `exists()` and `delete()`
- Line no.35-38 writes the file into HDFS on the provided path, followed by writing the dummy messages in the file. This introduces to another method about how to write files in HDFS.
- Line no. 40-44 reads the file which we just wrote in the previous code lines and write the content of the file on the console. This code example does not provide much useful work, it is just designed to get the basic understanding of how reading and writing files works in HDFS using Java API.
- Line no. 47-50 checks if the file exists in HDFS and if it does, renames the file from `dummy.txt` to `renamed_dummy.txt`

For further reading, you can check the HDFS API JavaDoc on HDFS API JavaDoc

## 6.8 Conclusion

This brings us to the conclusion of the article. We discussed the basics of Hadoop Distributed File System(HDFS) starting with the design, followed by the understanding of the HDFS architecture. Then we saw how to configure and start the HDFS node and finally we discussed how to interact with the running HDFS cluster using the shell command line and HDFS Java API. I hope this gives the basic explanation about HDFS and its building blocks.

## 6.9 Download the code

Download the Eclipse project containing the code used to understand the HDFS Java API in this example.

**Download** You can download the full source code of this example here: **HDFSJavaApi**

# Chapter 7

# Distributed Cache Example

In this example article, we will go through Apache Hadoop Distributed Cache and will understand how to use it with MapReduce Jobs.

## 7.1 Introduction

Distributed Cache as the name indicates is the caching system to store files or data which is required frequently and this mechanism is distributed in nature as all other components of Hadoop are.

It can cache **read-only** text files, archives, jar files etc. Which are needed by the application. So if there is a file which is needed by let us say map tasks. So it needs to be present on all the machines which will run Map tasks, This is what distributed cache is used for.

## 7.2 Working

Application which needs to use distributed cache to distribute a file should make sure that the file is available and can be accessed via urls. Urls can be either `hdfs://` or `https://`.

Now once the file is present on the mentioned url and user mention it to be a cache file to the distributed cache API, the Map-Reduce framework will copy the necessary files on all the nodes before initiation of the tasks on those nodes.

**Notes:** In case the files provided are archives, these will be automatically unarchived on the nodes after transfer.

## 7.3 Implementation

For understanding how to use the distributed cache API we will see an example in which we will write a modified version of the word count program.

For the basic word count example and if you like to understand the basics of how MapReduce job works, please refer to the article Apache Hadoop Wordcount Example

In this program, we will provide an input file to the Map-Reduce job with the words we need to count but we will also provide another file which contains stop words which we need to remove from the input text before counting the word occurrences.

So let's start looking into the code:

### 7.3.1 The Driver Class

The driver class is the main entry point of the system and the class which set up the Map-Reduce job.

```
package com.javacodegeeks.examples.distributedcache;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.filecache.DistributedCache;
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;

/**
 * The entry point for the WordCount example,
 * which setup the Hadoop job with Map and Reduce Class
 *
 * @author Raman
 */
public class Driver extends Configured implements Tool{

        /**
         * Main function which calls the run method and passes the args using ToolRunner
         * @param args Two arguments input and output file paths
         * @throws Exception
         */
        public static void main(String[] args) throws Exception{
                int exitCode = ToolRunner.run(new Driver(), args);
                System.exit(exitCode);
        }

        /**
         * Run method which schedules the Hadoop Job
         * @param args Arguments passed in main function
         */
        public int run(String[] args) throws Exception {
                if (args.length != 3) {
                        System.err.printf("Usage: %s needs two arguments    files\n",
                                        getClass().getSimpleName());
                        return -1;
                }

                //Initialize the Hadoop job and set the jar as well as the name of the Job
                Job job = new Job();
                job.setJarByClass(Driver.class);
                job.setJobName("Word Counter With Stop Words Removal");

                //Add input and output file paths to job based on the arguments passed
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                //Set the MapClass and ReduceClass in the job
                job.setMapperClass(MapClass.class);
                job.setReducerClass(ReduceClass.class);

                DistributedCache.addCacheFile(new Path(args[2]).toUri(), job. ↩
                    getConfiguration());

                //Wait for the job to complete and print if the job was successful or not
                int returnValue = job.waitForCompletion(true) ? 0:1;

                if(job.isSuccessful()) {
                        System.out.println("Job was successful");
                } else if(!job.isSuccessful()) {
                        System.out.println("Job was not successful");
                }

                return returnValue;
        }
}
```

Above is the complete code of the driver class. You can see is the `main()` method we set up and initialize a Hadoop `Job()`. First of all this code checks for the arguments passed to the method. Arguments need to be 3 in number:

- Input text file path which containc the text for word count
- Output path for storing the output of the program
- File path and name containing the stop words which we will distribute through the Hadoop Distributed Cache

The code:

```
if (args.length != 3) {
        System.err.printf("Usage: %s needs two arguments    files\n",
                        getClass().getSimpleName());
        return -1;
}
```

checks for the number fo arguments and make sure we have the required number of arguments present otherwise it stops the program then and there.

After this the `Job` is initialized:

```
//Initialize the Hadoop job and set the jar as well as the name of the Job
Job job = new Job();
```

followed by all the necessary configuration settings including configuring the jar file, map and reduce classes, input and output methods and input and output paths.

Out main focus here is on the line number 61, which is:

```
DistributedCache.addCacheFile(new Path(args[2]).toUri(), job.getConfiguration());
```

This line of code calls the `DistributedCache` API and adds the cache file URL which we passed as the third argument to the program. Before passing this argument, it need to be converted to the path url. Second argument needs to be the configurations of the Hadoop job we are setting up.

The above code will set up the Hadoop Job and sets up the required file as the cache file in the Hadoop cluster. It is as easy as calling a single function. The main task is how to retrieve this cache file and how to use it to remove stop words from the processing text. That we will see in the `map` class in the following section.

### 7.3.2 Map Class

Map class contains the mapper method which is the main focus which contains the code regarding how to use the cache files in the MapReduce Tasks.

```
package com.javacodegeeks.examples.distributedcache;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.StringTokenizer;

import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * Map Class which extends MaReduce.Mapper class
 * Map is passed a single line at a time, it splits the line based on space
 * and generated the token which are output by map with value as one to be consumed
 * by reduce class
 * @author Raman
 */
public class MapClass extends Mapper{

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

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        try{
                Path[] stopWordsFiles = DistributedCache.getLocalCacheFiles(context. ↩
                    getConfiguration());
                if(stopWordsFiles != null &amp;&amp; stopWordsFiles.length > 0) {
                        for(Path stopWordFile : stopWordsFiles) {
                                readFile(stopWordFile);
                        }
                }
        } catch(IOException ex) {
                System.err.println("Exception in mapper setup: " + ex.getMessage());
        }
    }

    /**
     * map function of Mapper parent class takes a line of text at a time
     * splits to tokens and passes to the context as word along with value as one
     */
        @Override
        protected void map(LongWritable key, Text value,
                        Context context)
                        throws IOException, InterruptedException {

                String line = value.toString();
                StringTokenizer st = new StringTokenizer(line," ");

                while(st.hasMoreTokens()){
                        String wordText = st.nextToken();

                        if(!stopWords.contains(wordText.toLowerCase())) {
                                word.set(wordText);
                                context.write(word,one);
                        }
                }

        }

        private void readFile(Path filePath) {
                try{
                        BufferedReader bufferedReader = new BufferedReader(new FileReader( ↩
                            filePath.toString()));
                        String stopWord = null;
                        while((stopWord = bufferedReader.readLine()) != null) {
                                stopWords.add(stopWord.toLowerCase());
                        }
                } catch(IOException ex) {
                        System.err.println("Exception while reading stop words file: " + ex ↩
                            .getMessage());
                }
        }
}
```

Now this is where this code varies significantly from the standard word count MapReduce code. The map class contains a `setup` method which is the first method called when a node is setup to perform the map task.

```
@Override
    protected void setup(Context context) throws IOException, InterruptedException {
        try{
                Path[] stopWordsFiles = DistributedCache.getLocalCacheFiles(context. ↩
                    getConfiguration());
                if(stopWordsFiles != null &amp;&amp; stopWordsFiles.length > 0) {
                        for(Path stopWordFile : stopWordsFiles) {
                                readFile(stopWordFile);
                        }
                }
        } catch(IOException ex) {
                System.err.println("Exception in mapper setup: " + ex.getMessage());
        }
    }
```

So this is the place where we read the file stored in the distribute cache using the `DistributedCache` API and `getLocalCacheFiles()` method as shown in the line number 4 of the above code snippet. If you notice the methods return an array of the type `Path`. So for each file(we have only one in this case) we will call another method called `readFile()` and pass the path of the file to this method.

`readFile()` is the method which reads the content of the file and adds the stop words in the global `Set` of `stopWords`. The details of the method are in line numebr 67-77 of the Map class.

Now in the `map()` method, after splitting the lines into word tokens, we will check if a particular word is present in the stop words set, if it is present we skip that word and move to the next but if it is not a stop word then we pass it on to the context to be executed in the Reduce class as shown in the code snippet below:

```
StringTokenizer st = new StringTokenizer(line," ");

                while(st.hasMoreTokens()){
                        String wordText = st.nextToken();

                        if(!stopWords.contains(wordText.toLowerCase())) {
                                word.set(wordText);
                                context.write(word,one);
                        }
                }
```

### 7.3.3 Reduce Class

Reduce class in this article is exactly same as it is in the standard word count example, the `reduce()` method will contain only those words which are not stop words and reduce will count only the good words. Following is the code of the reduce class:

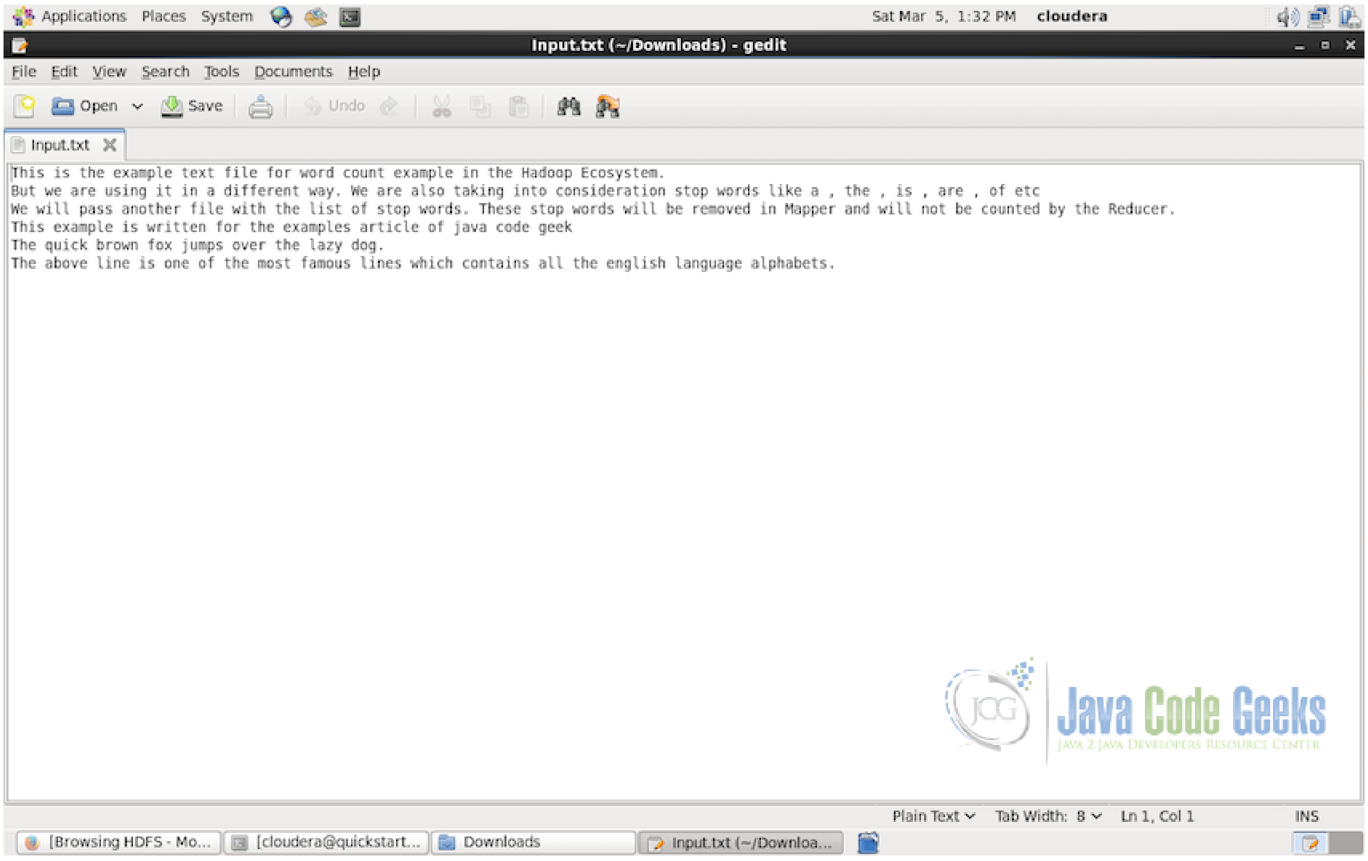
```
package com.javacodegeeks.examples.distributedcache;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * Reduce class which is executed after the map class and takes
 * key(word) and corresponding values, sums all the values and write the
 * word along with the corresponding total occurances in the output
 *
 * @author Raman
 */
public class ReduceClass extends Reducer{

        /**
         * Method which performs the reduce operation and sums
         * all the occurrences of the word before passing it to be stored in output
         */
        @Override
        protected void reduce(Text key, Iterable values,
                        Context context)
                        throws IOException, InterruptedException {

                int sum = 0;
                Iterator valuesIt = values.iterator();

                while(valuesIt.hasNext()){
                        sum = sum + valuesIt.next().get();
                }

                context.write(key, new IntWritable(sum));
        }
}
```

## 7.4 Executing the Hadoop Job

We will execute the MapReduce task we discussed in the previous section on the Hadoop cluster. But before we do so, we need two files

- Input file
- Stop Words file

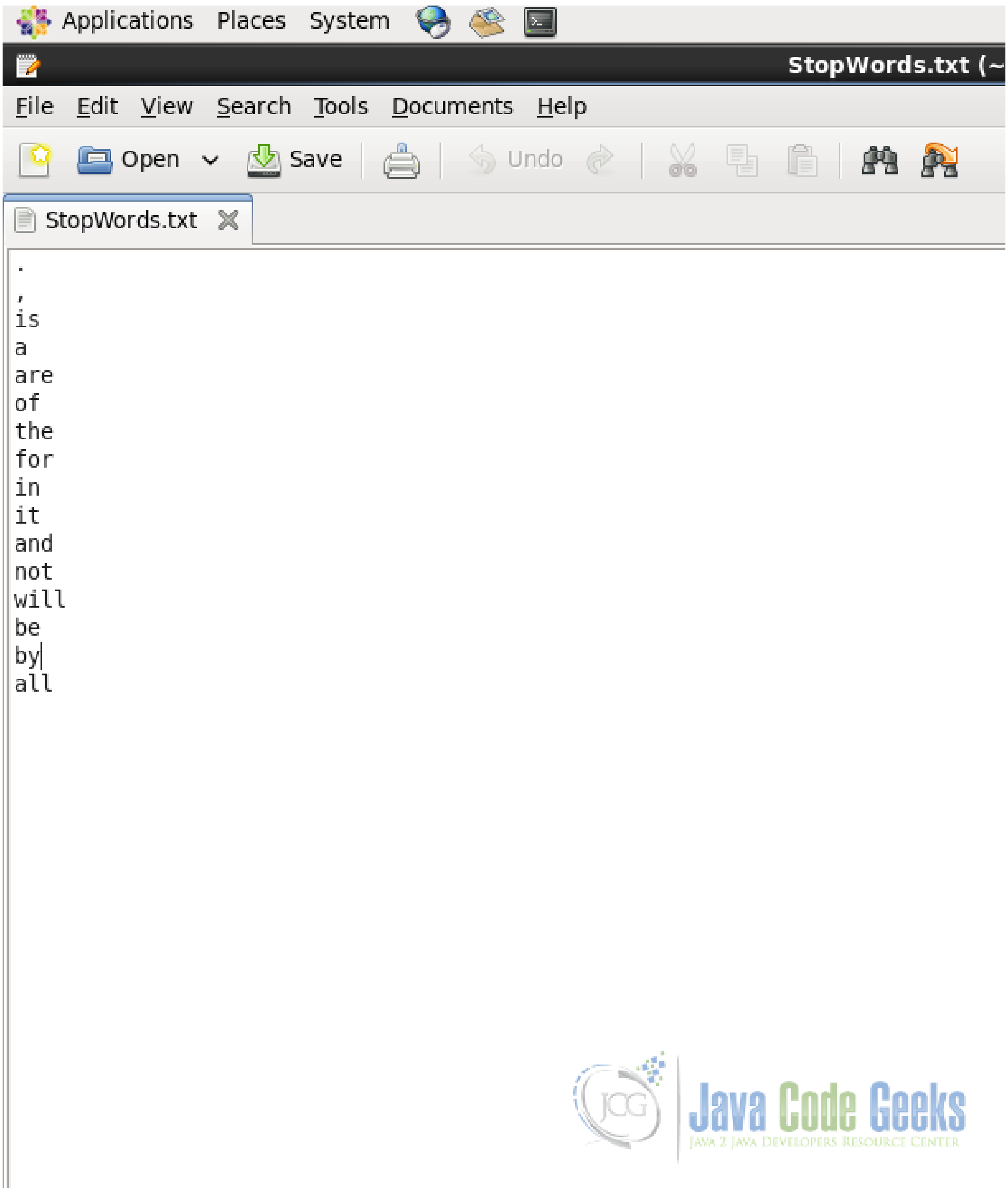
So following is the dummy text file which we will use for the example:


Applications Places System
Sat Mar 5, 1:32 PM cloudera
Input.txt (~/Downloads) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Input.txt
Plain Text Tab Width: 8 Ln 1, Col 1 INS
[Browsing HDFS - Mo... [cloudera@quickstart... Downloads Input.txt (~/Downloa...

```
This is the example text file for word count example in the Hadoop Ecosystem.
But we are using it in a different way. We are also taking into consideration stop words like a , the , is , are , of etc
We will pass another file with the list of stop words. These stop words will be removed in Mapper and will not be counted by the Reducer.
This example is written for the examples article of java code geek
The quick brown fox jumps over the lazy dog.
The above line is one of the most famous lines which contains all the english language alphabets.
```

Figure 7.1: Input.txt file to be processed

and following is the file containing stop words:

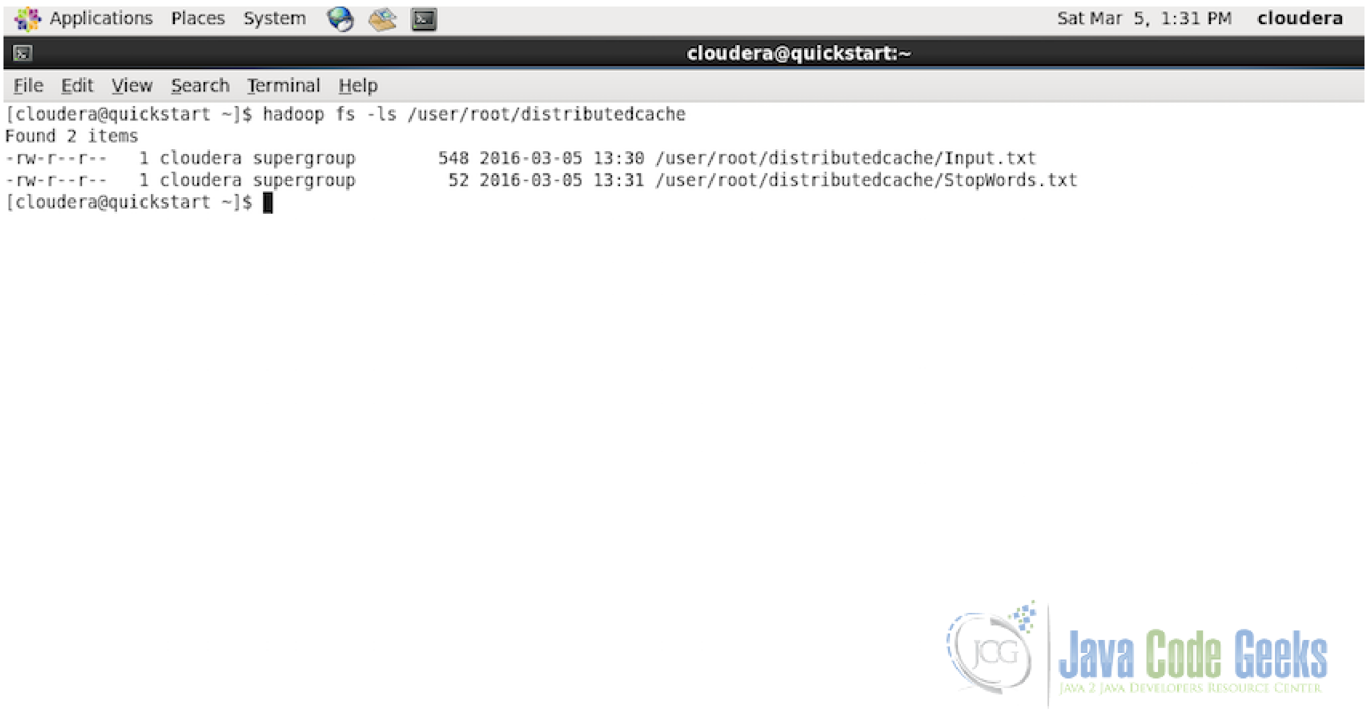


```
.
,
is
a
are
of
the
for
in
it
and
not
will
be
by
all
```

Figure 7.2: StopWords.txt file containing list of stop words

Make sure that both the files are present in the Hadoop Distributed File System. If you would like to read about the basics of HDFS and Hadoop File System including how to put files in HDFS, please refer to the article Apache Hadoop FS Commands

Example



Applications Places System
Sat Mar 5, 1:31 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

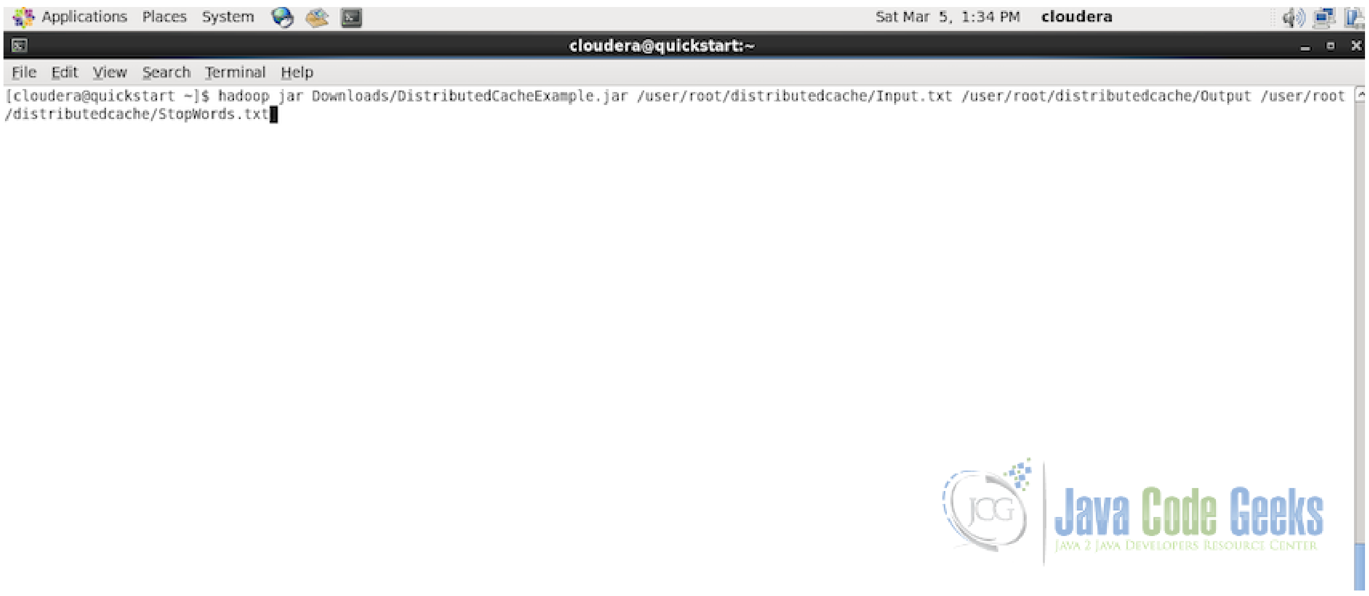
```
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/distributedcache
Found 2 items
-rw-r--r--   1 cloudera supergroup        548 2016-03-05 13:30 /user/root/distributedcache/Input.txt
-rw-r--r--   1 cloudera supergroup         52 2016-03-05 13:31 /user/root/distributedcache/StopWords.txt
[cloudera@quickstart ~]$
```

Figure 7.3: Listing the file present in HDFS

Now, to execute the Distributed Cache Example Task on the Hadoop Cluster, we have to submit the jar file along with the URLs of the input and stopwords files to the Hadoop Cluster. Following is the command to do so:

```
hadoop jar DistributedCacheExample.jar /user/root/distributedcache/Input.txt /user/root/ ↩
    distributedcache/Output /user/root/distributedcache/StopWords.txt
```

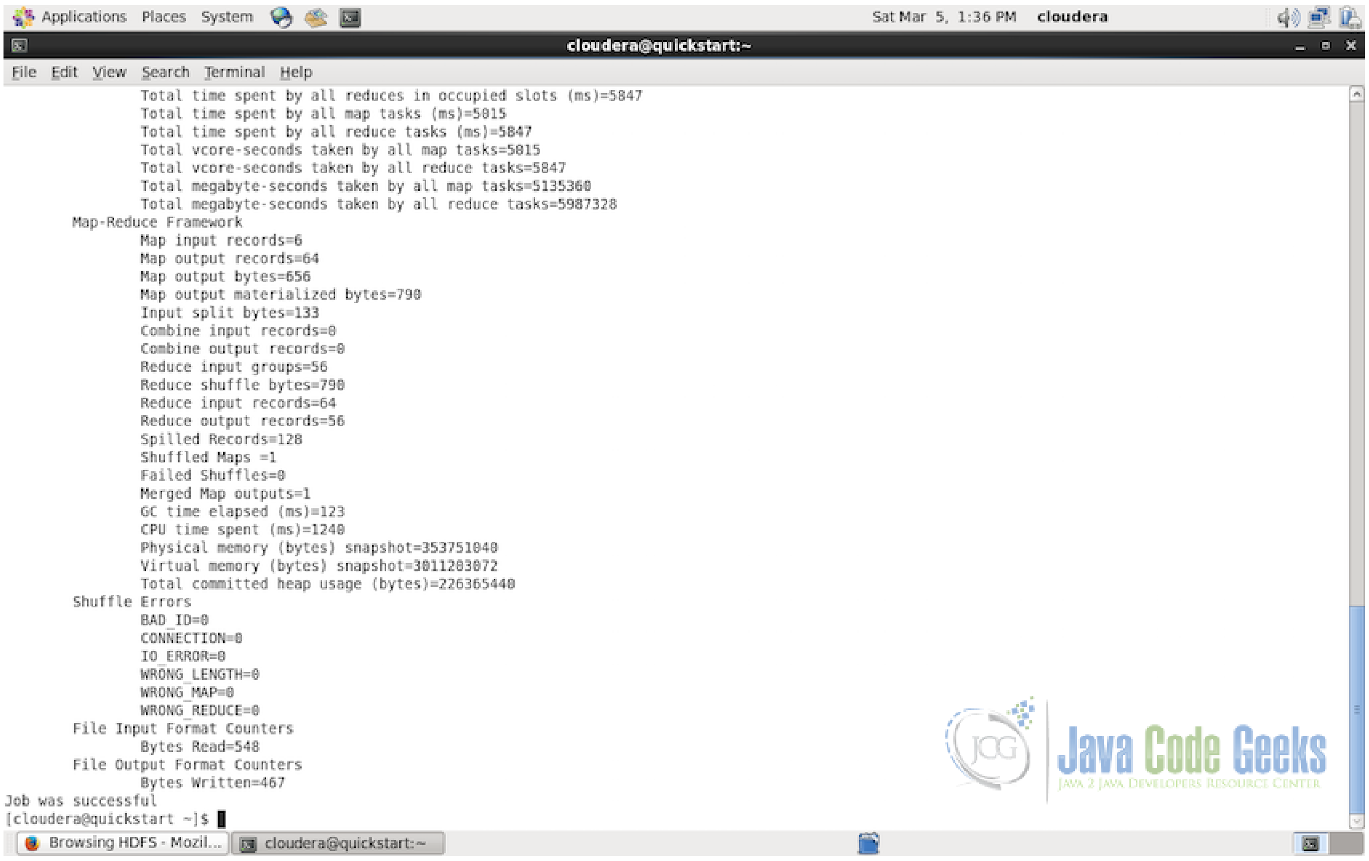
First argument mentions the input file to be used, second argument tells about the path where the output should be stored and the third argument tells the path of the stop words file.



```
[cloudera@quickstart ~]$ hadoop jar Downloads/DistributedCacheExample.jar /user/root/distributedcache/Input.txt /user/root/distributedcache/Output /user/root
/distributedcache/StopWords.txt
```

Figure 7.4: Command to submit hadoop job to the cluster

Once the job is successfully executed we will a console output something similar to:

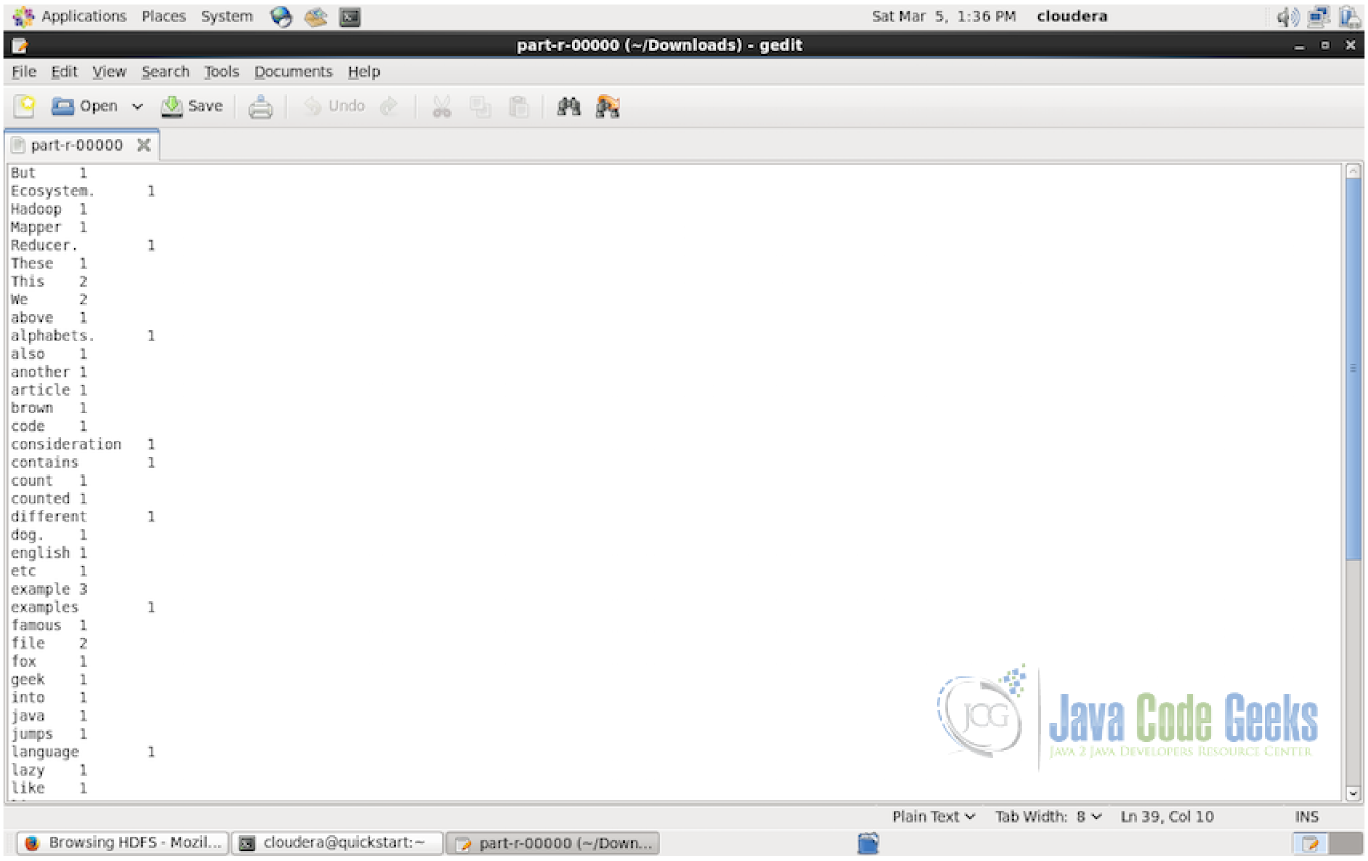


```
                Total time spent by all reduces in occupied slots (ms)=5847
                Total time spent by all map tasks (ms)=5815
                Total time spent by all reduce tasks (ms)=5847
                Total vcore-seconds taken by all map tasks=5815
                Total vcore-seconds taken by all reduce tasks=5847
                Total megabyte-seconds taken by all map tasks=5135360
                Total megabyte-seconds taken by all reduce tasks=5987328
        Map-Reduce Framework
                Map input records=6
                Map output records=64
                Map output bytes=656
                Map output materialized bytes=790
                Input split bytes=133
                Combine input records=0
                Combine output records=0
                Reduce input groups=56
                Reduce shuffle bytes=790
                Reduce input records=64
                Reduce output records=56
                Spilled Records=128
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=123
                CPU time spent (ms)=1240
                Physical memory (bytes) snapshot=353751040
                Virtual memory (bytes) snapshot=3011203072
                Total committed heap usage (bytes)=226365440
        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=548
        File Output Format Counters
                Bytes Written=467
Job was successful
[cloudera@quickstart ~]$
```

Figure 7.5: Console Output

Notice the last line which says "Job was successful". This is the line we printed from the Driver class on successful execution of the job. You can check the other details in the console output to know more about the job execution.

The output of the Hadoop job will be present on the HDFS path `/user/root/distributedcache/Output` in the `Output` folder as mentioned in the execution argument, this folder can be downloaded on the system from the HDFS. Following is how the output file looks like:


Applications Places System
Sat Mar 5, 1:36 PM cloudera
part-r-00000 (~/Downloads) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
part-r-00000
Plain Text Tab Width: 8 Ln 39, Col 10 INS
Browsing HDFS - Mozil... cloudera@quickstart:~ part-r-00000 (~/Down...

```
But     1
Ecosystem.      1
Hadoop  1
Mapper  1
Reducer.        1
These   1
This    2
We      2
above   1
alphabets.      1
also    1
another 1
article 1
brown   1
code    1
consideration   1
contains        1
count   1
counted 1
different       1
dog.    1
english 1
etc     1
example 3
examples        1
famous  1
file    2
fox     1
geek    1
into    1
java    1
jumps   1
language        1
lazy    1
like    1
```

Figure 7.6: Output file

## 7.5 Conclusion

In this example article, we talked about the Distributed Cache API of Apache Hadoop. We started with the introduction of what exactly distributed cache is and then understood the basic workflow of the distributed cache. Then we dived into the implementation section where we saw how we can use the Distributed Cache API to pass the common files, jars and other archives to the nodes executing the Hadoop Job.

## 7.6 Download the Eclipse Project

Complete code of the example and the dummy input and stop words text file can be useful for experimentation.

**Download** You can download the full source code of this example here: **DistributedCacheExample**

# Chapter 8

# Wordcount Example

In this example, we will demonstrate the **Word Count** example in Hadoop. Word count is the basic example to understand the Hadoop MapReduce paradigm in which we count the number of instances of each word in an input file and gives the list of words and the number of instances of the particular word as an output.

## 8.1 Introduction

Hadoop is an Apache Software Foundation project which is the open source equivalent of Google MapReduce and Google File System. It is designed for distributed processing of large data sets across a cluster of systems running on commodity standard hardware.

Hadoop is designed with an assumption that hardware failure is a norm rather an exception. All hardware fails sooner or later and the system should be robust and capable enough to handle the hardware failures gracefully.

## 8.2 MapReduce

Apache Hadoop consists of two core components, one being Hadoop Distributed File System(HDFS) and second is the Framework and APIs for MapReduce jobs.

In this example, we are going to demonstrate the second component of Hadoop framework called MapReduce. If you are interested in understanding the basics if HDFS, the article Apache Hadoop Distributed File System Explained may be of help. Before moving to the example of MapReduce paradigm, we shall understand what MapReduce actually is.

MapReduce is basically a software framework or programming paradigm, which enable users to write programs as separate components so that data can be processed parallelly across multiple systems in a cluster. MapReduce consists of two parts Map and Reduce.

- Map: Map task is performed using a `map()` function that basically performs filtering and sorting. This part is responsible for processing one or more chunks of data and producing the output results which are generally referred as intermediate results. As shown in the diagram below, map task is generally processed in parallel provided the mapping operation is independent of each other.
- Reduce: Reduce task is performed by `reduce()` function and performs a summary operation. It is responsible for consolidating the results produced by each of the Map task.

## 8.3 Word-Count Example

Word count program is the basic code which is used to understand the working of the MapReduce programming paradigm. The program consists of MapReduce job that counts the number of occurrences of each word in a file. This job consists of two parts `map` and `reduce`. The Map task maps the data in the file and counts each word in data chunk provided to the map function. The outcome of this task is passed to reduce task which combines and reduces the data to output the final result.

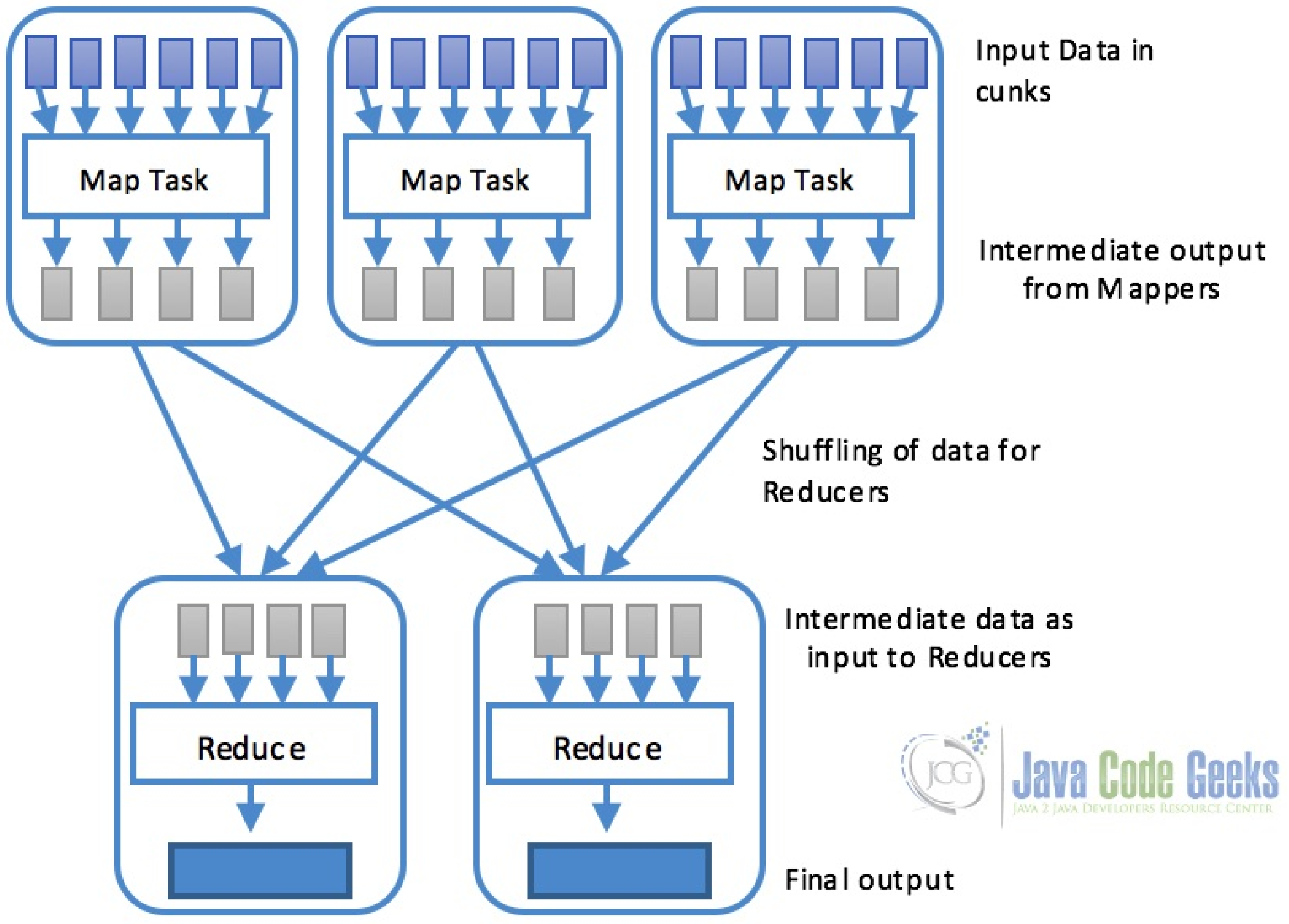

Input Data in cunks
Map Task
Map Task
Map Task
Intermediate output from Mappers
Shuffling of data for Reducers
Intermediate data as input to Reducers
Reduce
Reduce
Final output
Java Code Geeks

Figure 8.1: Working of Map and Reduce

### 8.3.1 Setup

We shall use Maven to setup a new project for Hadoop word count example. Setup a maven project in Eclipse and add the following Hadoop dependency to the `pom.xml`. This will make sure we have the required access to the Hadoop core library.

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-core</artifactId>
    <version>1.2.1</version>
</dependency>
```

After adding the dependency, we are ready to write our word count code.

### 8.3.2 Mapper Code

The mapper task is responsible for tokenizing the input text based on space and create a list of words, then traverse over all the tokens and emit a key-value pair of each word with a count of one. Following is the `MapClass`:

```
package com.javacodegeeks.examples.wordcount;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapClass extends Mapper{

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

    @Override
    protected void map(LongWritable key, Text value,
                        Context context)
                        throws IOException, InterruptedException {

        //Get the text and tokenize the word using space as separator.
                String line = value.toString();
                StringTokenizer st = new StringTokenizer(line," ");

        //For each token aka word, write a key value pair with
        //word and 1 as value to context
                while(st.hasMoreTokens()){
                        word.set(st.nextToken());
                        context.write(word,one);
                }

        }
}
```

Following is what exactly `map` task does:

- Line 13-14, defines static variable `one` with interger value 1 and `word` for storing the words.
- Line 22-23, In `map` method the input `Text` varoable is converted to `String` and Tokenized based on the space to get all the words in the input text.
- Line 27-30, For each word in the text, set the `word` variable and pass a key-value pair of `word` and integer value `one` to the `context`.

### 8.3.3 Reducer Code

Following code snippet contains `ReduceClass` which extends the MapReduce Reducer class and overwrites the `reduce()` function. This function is called after the map method and receives keys from the `map()` function corresponding to the specific key. Reduce method iterates over the values, adds them and reduces to a single value before finally writing the word and the number of occurrences of the word to the output file.

```
package com.javacodegeeks.examples.wordcount;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
public class ReduceClass extends Reducer{

        @Override
        protected void reduce(Text key, Iterable values,
                        Context context)
                        throws IOException, InterruptedException {

                int sum = 0;
                Iterator valuesIt = values.iterator();

        //For each key value pair, get the value and adds to the sum
        //to get the total occurances of a word
                while(valuesIt.hasNext()){
                        sum = sum + valuesIt.next().get();
                }

        //Writes the word and total occurances as key-value pair to the context
                context.write(key, new IntWritable(sum));
        }
}
```

Following is the workflow of `reduce` function:

- Lines 17-18, define a variable `sum` as interger with value 0 and `Iterator` over the values received by the reducer.
- Lines 22-24, Iterate over all the values and add the occurances of the words in `sum`
- Line 27, write the `word` and the `sum` as key-value pair in the `context`

### 8.3.4 The Driver Class

So now when we have our map and reduce classes ready, it is time to put it all together as a single job which is done in a class called driver class. This class contains the `main()` method to setup and run the job.

```
package com.javacodegeeks.examples.wordcount;

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.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;

public class WordCount extends Configured implements Tool{

        public static void main(String[] args) throws Exception{
                int exitCode = ToolRunner.run(new WordCount(), args);
                System.exit(exitCode);
        }

        public int run(String[] args) throws Exception {
                if (args.length != 2) {
                        System.err.printf("Usage: %s needs two arguments, input and output
files\n", getClass().getSimpleName());
                        return -1;
                }
```

```
            //Create a new Jar and set the driver class(this class) as the main class of ↩
                jar
            Job job = new Job();
                job.setJarByClass(WordCount.class);
                job.setJobName("WordCounter");

        //Set the input and the output path from the arguments
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setOutputFormatClass(TextOutputFormat.class);

        //Set the map and reduce classes in the job
                job.setMapperClass(MapClass.class);
                job.setReducerClass(ReduceClass.class);

        //Run the job and wait for its completion
                int returnValue = job.waitForCompletion(true) ? 0:1;

                if(job.isSuccessful()) {
                        System.out.println("Job was successful");
                } else if(!job.isSuccessful()) {
                        System.out.println("Job was not successful");
                }

                return returnValue;
        }
}
```

Following is the workflow of `main` function:

- Line 22-26, check if the required number of arguments are provided.
- Line 29-31, create a new `Job`, set the name of the job and the main class.
- Line 34-35, set the input and the output paths from the arguments.
- Line 37-39, set the key value type classes and the output format class. These classes need to be the same type whichw e use in the map and reduce for the output.
- Line 42-43, set the Map and Reduce classes in the `job`
- Line 46, execute the job and wait for its completion

## 8.4 Code Execution

There are two ways to execute the code we have written, first is to execute it within Eclipse IDE itself for the testing purpose and second is to execute in the Hadoop Cluster. We will see both ways in this section.

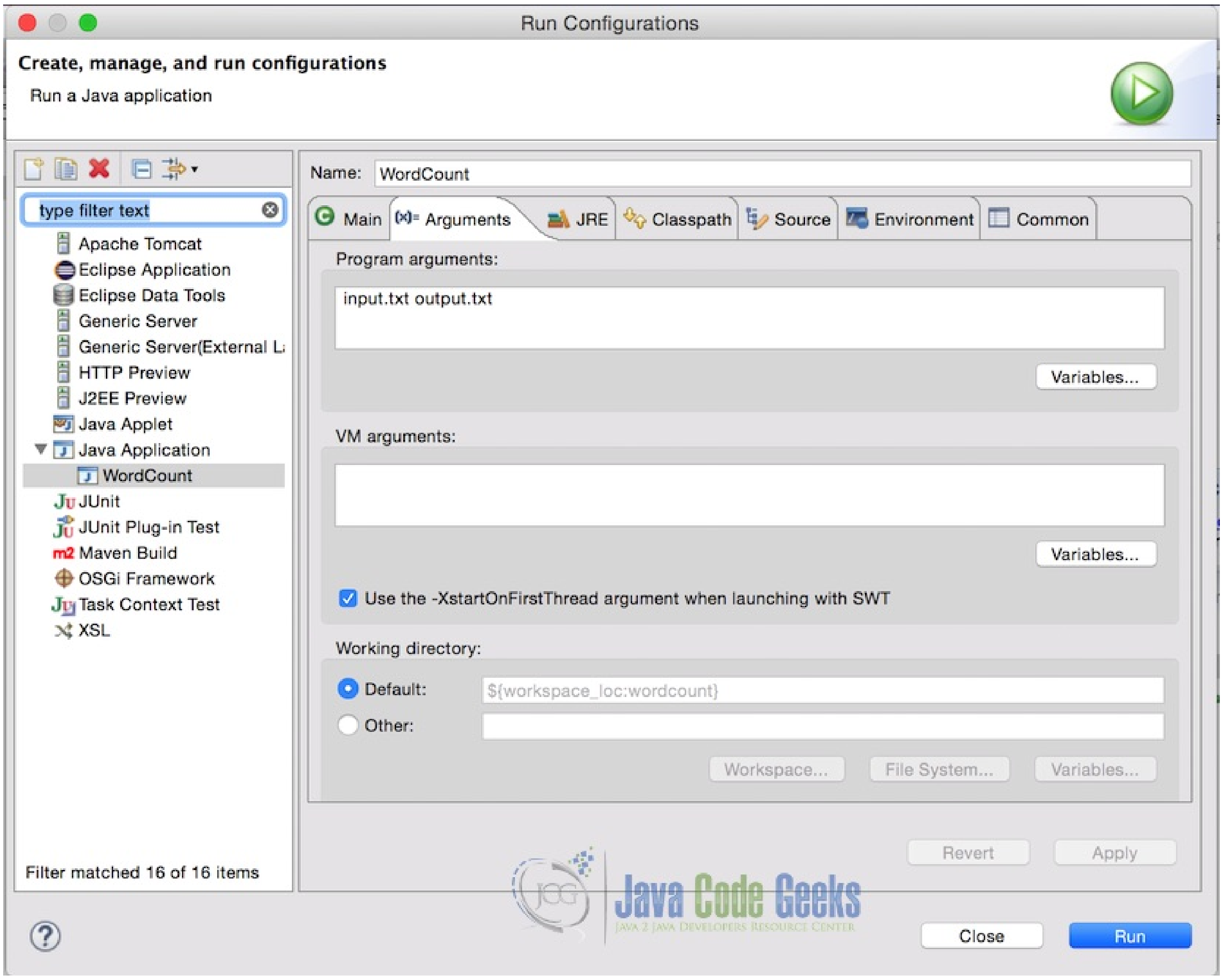
### 8.4.1 In Eclipse IDE

For executing the wordcount code in eclipse. First of all, create an input.txt file with dummy data. For the testing purpose, We have created a file with the following text in the project root.

```
This is the example text file for word count example also knows as hello world example of ↩
    the Hadoop ecosystem.
This example is written for the examples article of java code geek
```

```
The quick brown fox jumps over the lazy dog.
The above line is one of the most famous lines which contains all the english language ↩
    alphabets.
```

In Eclipse, Pass the input file and output file name in the project arguments. Following is how the arguments looks like. In this case, the input file is in the root of the project that is why just filename is required, but if your input file is in some other location, you should provide the complete path.


Run Configurations

Create, manage, and run configurations

Run a Java application

Name: WordCount

Program arguments:

```
input.txt output.txt
```

VM arguments:

[✓] Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

(●) Default: ${workspace_loc:wordcount}

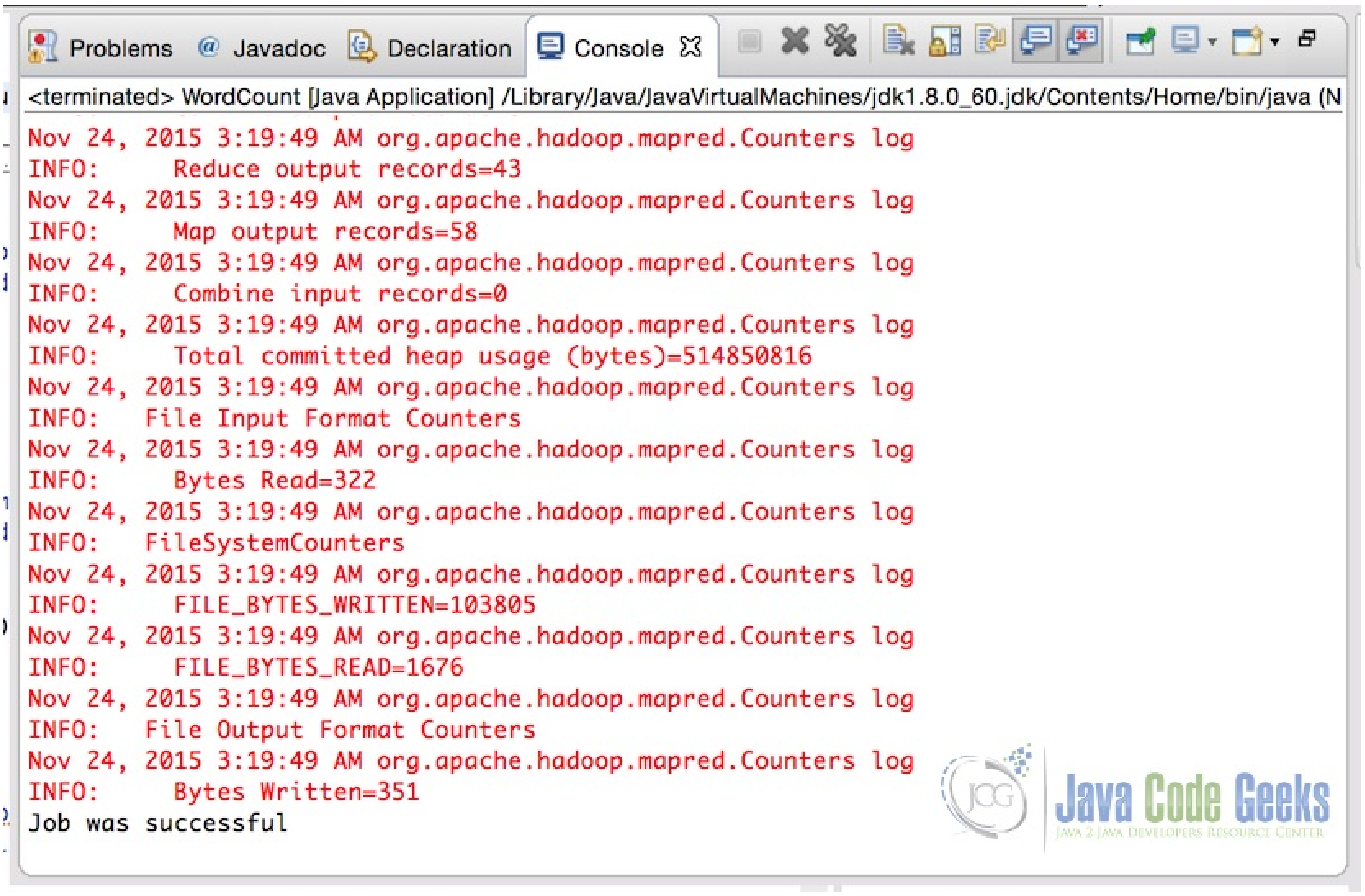
( ) Other:

Filter matched 16 of 16 items

Figure 8.2: Run Configuration of Eclipse Project

**Note:** Make sure the output file does not exist already. If it does, the program will throw an error.

After setting the arguments, simply run the application. Once the application is successfully completed, console will show the output.



```
<terminated> WordCount [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (N
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Reduce output records=43
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Map output records=58
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Combine input records=0
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Total committed heap usage (bytes)=514850816
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   File Input Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Read=322
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   FileSystemCounters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_WRITTEN=103805
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_READ=1676
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:   File Output Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Written=351
Job was successful
```

Figure 8.3: Console output in Eclipse

Below is the content of the output file:

```
Hadoop  1
The     2
This    2
above   1
all     1
alphabets.      1
also    1
article 1
as      1
brown   1
code    1
contains        1
count   1
dog.    1
ecosystem.      1
english 1
example 4
examples        1
famous  1
file    1
for     2
fox     1
geek    1
hello   1
is      3
```

```
java    1
jumps   1
knows   1
language        1
lazy    1
line    1
lines   1
most    1
of      3
one     1
over    1
quick   1
text    1
the     6
which   1
word    1
world   1
written 1
```

### 8.4.2 On Hadoop Cluster

For running the Wordcount example on hadoop cluster, we assume:

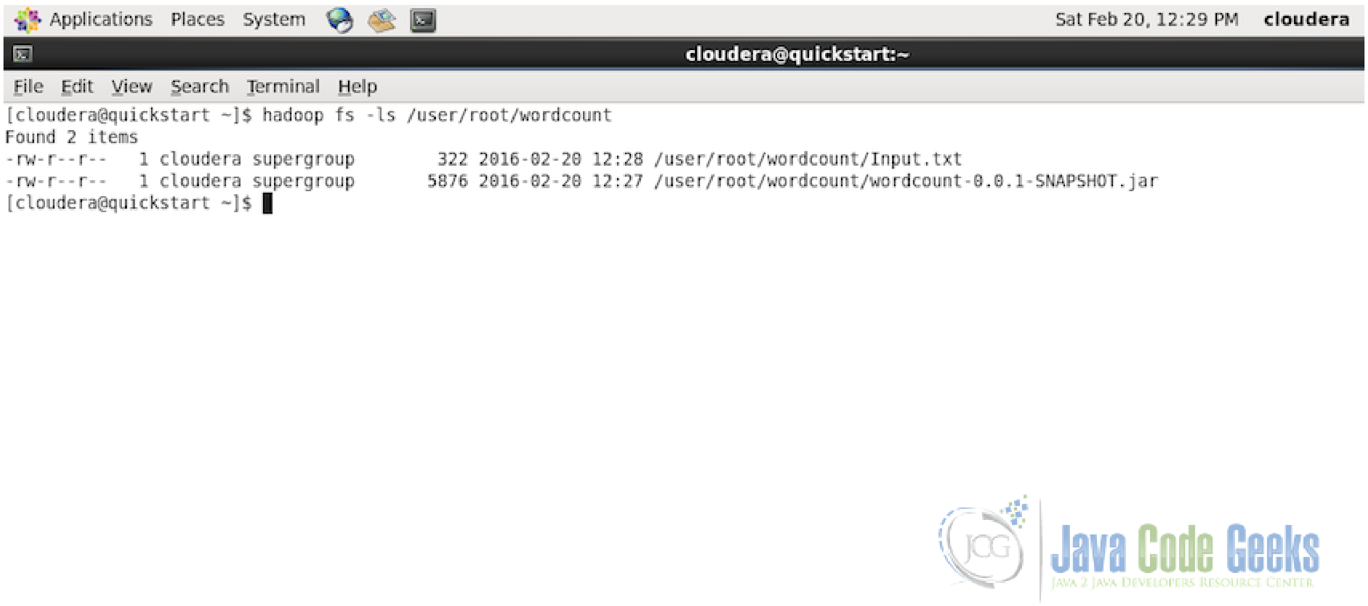
- Hadoop cluster is setup and running
- Input file is at the path `/user/root/wordcount/Input.txt` in the HDFS

In case, you need any help with setting up the hadoop cluster or Hadoop File System, please refer to the following articles:

- How to Install Apache Hadoop on Ubuntu
- Apache Hadoop Cluster Setup Example(with Virtual Machines)
- Apache Hadoop Distributed File System Explained
- Apache Hadoop FS Commands Example

Now, first of all make sure the `Input.txt` file is present at the path `/user/root/wordcount` using the command:

```
hadoop fs -ls /user/root/wordcount
```


Applications Places System
Sat Feb 20, 12:29 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

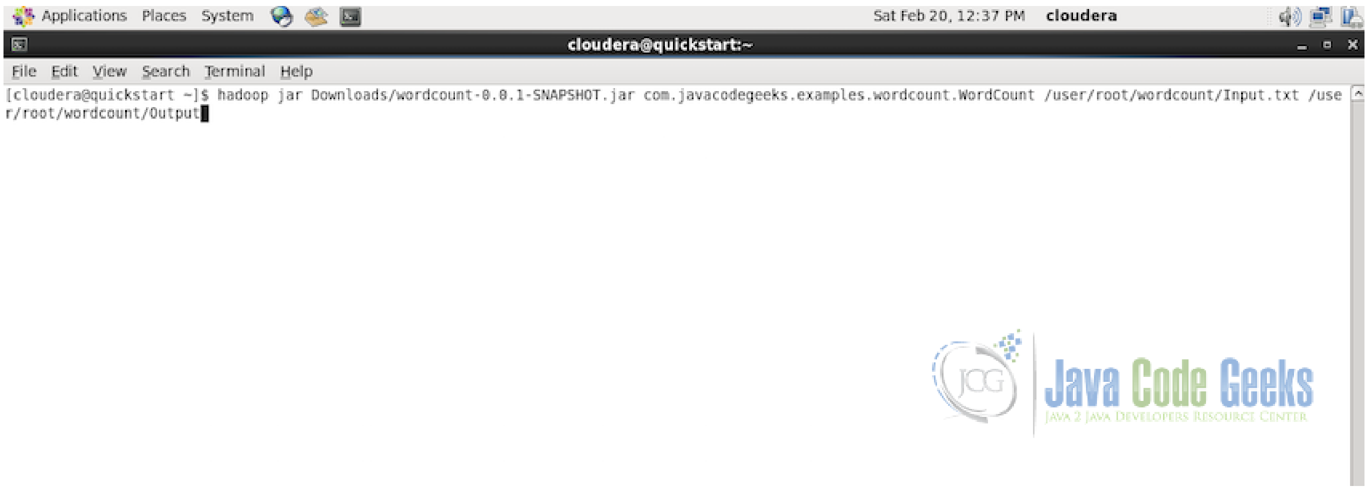
```
[cloudera@quickstart ~]$ hadoop fs -ls /user/root/wordcount
Found 2 items
-rw-r--r--   1 cloudera supergroup        322 2016-02-20 12:28 /user/root/wordcount/Input.txt
-rw-r--r--   1 cloudera supergroup       5876 2016-02-20 12:27 /user/root/wordcount/wordcount-0.0.1-SNAPSHOT.jar
[cloudera@quickstart ~]$
```

Figure 8.4: Confirm if Input file exists in the required folder

Now it is time to submit the MapReduce job. Use the following command for execution

```
hadoop jar Downloads/wordcount-0.0.1-SNAPSHOT.jar com.javacodegeeks.examples.wordcount. ←
    Wordcount /user/root/wordcount/Input.txt /user/root/wordcount/Output
```

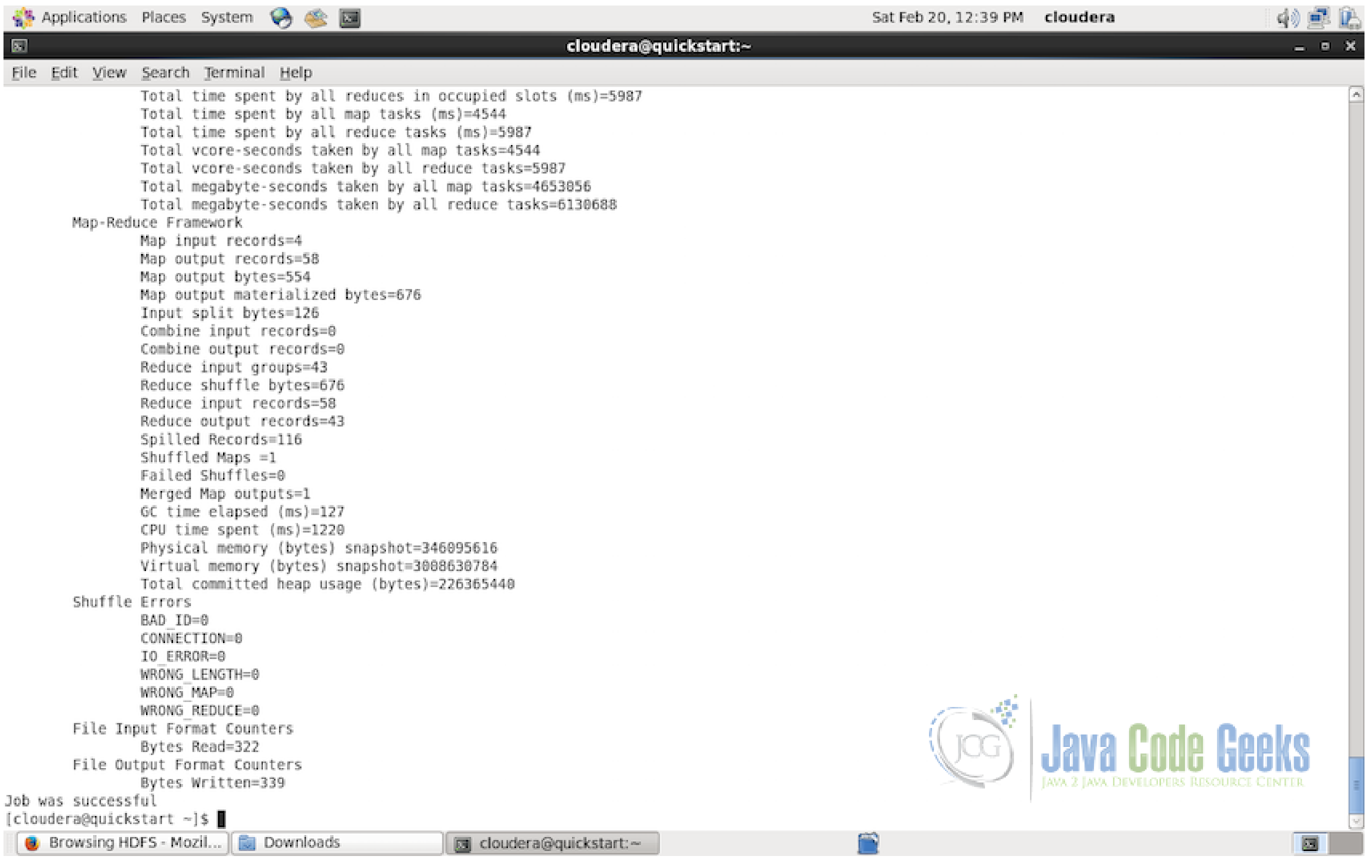
In the above code, jar file is in the `Downloads` folder and the Main class is at the path `com.javacodegeeks.examples.wordcount.Wordcount`


Applications Places System
Sat Feb 20, 12:37 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop jar Downloads/wordcount-0.0.1-SNAPSHOT.jar com.javacodegeeks.examples.wordcount.WordCount /user/root/wordcount/Input.txt /use
r/root/wordcount/Output
```

Figure 8.5: Jar execution command

Following should be the output of the execution. Console output's last line informs us that the job was successfully completed.


Applications Places System
Sat Feb 20, 12:39 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
Java Code Geeks
Java 2 Java Developers Resource Center
Browsing HDFS - Mozil...
Downloads
cloudera@quickstart:~

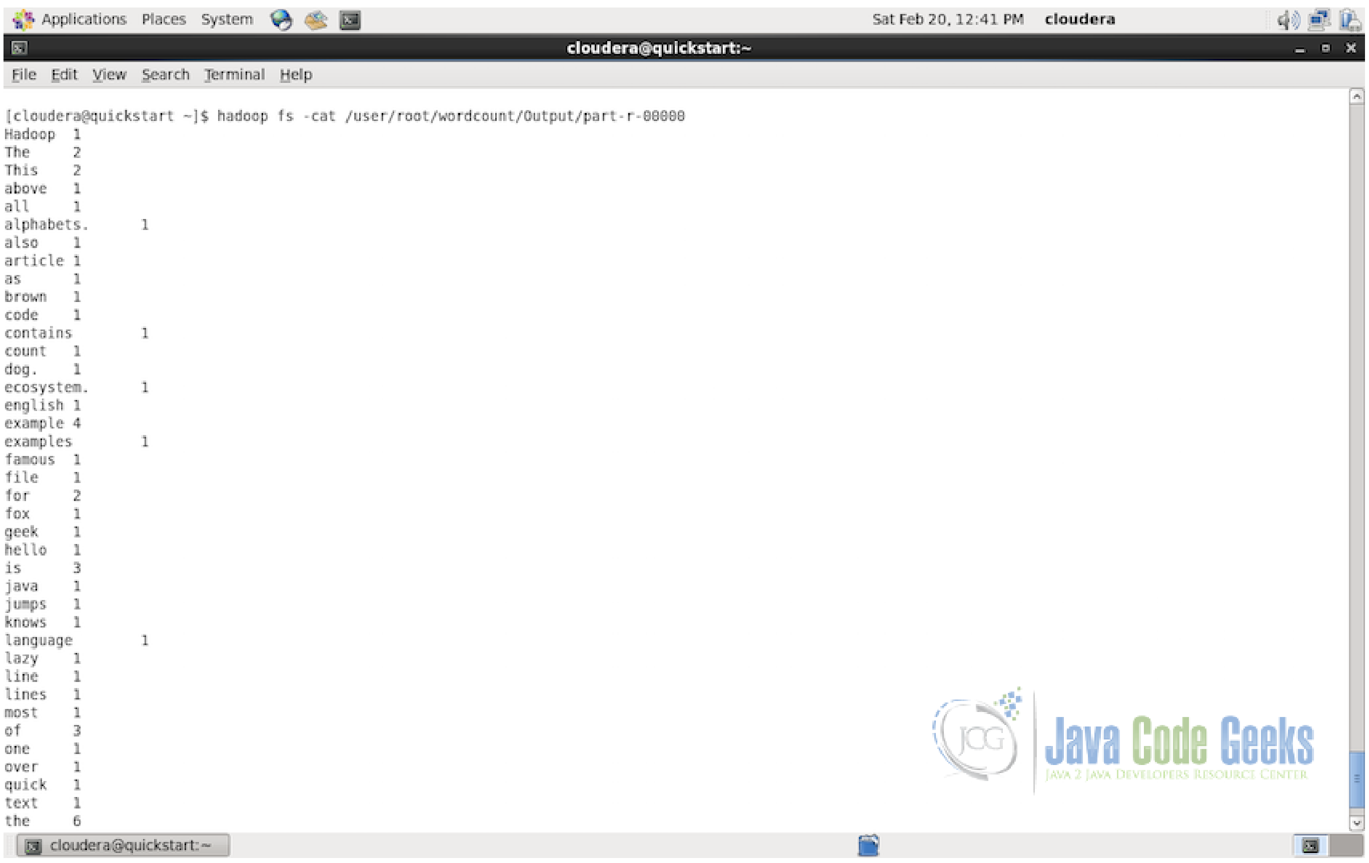
```
                Total time spent by all reduces in occupied slots (ms)=5987
                Total time spent by all map tasks (ms)=4544
                Total time spent by all reduce tasks (ms)=5987
                Total vcore-seconds taken by all map tasks=4544
                Total vcore-seconds taken by all reduce tasks=5987
                Total megabyte-seconds taken by all map tasks=4653056
                Total megabyte-seconds taken by all reduce tasks=6130688
        Map-Reduce Framework
                Map input records=4
                Map output records=58
                Map output bytes=554
                Map output materialized bytes=676
                Input split bytes=126
                Combine input records=0
                Combine output records=0
                Reduce input groups=43
                Reduce shuffle bytes=676
                Reduce input records=58
                Reduce output records=43
                Spilled Records=116
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=127
                CPU time spent (ms)=1220
                Physical memory (bytes) snapshot=346095616
                Virtual memory (bytes) snapshot=3008630784
                Total committed heap usage (bytes)=226365440
        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=322
        File Output Format Counters
                Bytes Written=339
Job was successful
[cloudera@quickstart ~]$ 
```

Figure 8.6: Console Output

Now we can read the output of the Wordcount map reduce job in the folder `/user/root/wordcount/Output/`. Use the following command to check the output in the console:

```
hadoop fs -cat /user/root/wordcount/Output/part-r-00000
```

Following screenshot display the content of the Output folder on the console.



```
[cloudera@quickstart ~]$ hadoop fs -cat /user/root/wordcount/Output/part-r-00000
Hadoop  1
The     2
This    2
above   1
all     1
alphabets.      1
also    1
article 1
as      1
brown   1
code    1
contains        1
count   1
dog.    1
ecosystem.      1
english 1
example 4
examples        1
famous  1
file    1
for     2
fox     1
geek    1
hello   1
is      3
java    1
jumps   1
knows   1
language        1
lazy    1
line    1
lines   1
most    1
of      3
one     1
over    1
quick   1
text    1
the     6
```

Figure 8.7: Output file

## 8.5 Conclusion

This example explains the MapReduce paradigm with respect to Apache Hadoop and how to write the word count example in MapReduce step by step. Next we saw how to execute the example in the eclipse for the testing purpose and also how to execute in the Hadoop cluster using HDFS for the input files. The article also provides links to the other useful articles for setting up Hadoop on Ubuntu, Setting up Hadoop Cluster, Understanding HDFS and Basic FS commands. We hope, this article serves the best purpose of explaining the basics of the Hadoop MapReduce and provides you with the solid base for understanding Apache Hadoop and MapReduce.

## 8.6 Download the Eclipse Project

Click on the following link to download the complete eclipse project of wordcount example.

**Download** You can download the full source code of this example here: **HadoopWordCount**

# Chapter 9

# Streaming Example

In this example, we will dive into the streaming component of Hadoop MapReduce. We will understand the basics of Hadoop Streaming and see an example using Python.

## 9.1 Introduction

Hadoop Streaming is the name which is quite misleading, here streaming has nothing to do with the continuous data streams or continuous data flow as it is understood generally. Hadoop Streaming is just a utility provided by the Hadoop MapReduce distribution which gives users the possibility to write MapReduce jobs in other programming languages like Python or Cpp etc which can make use of `stdin` and `stdout` to readin and writeout lines of text data. Support for Cpp is available on since the version 0.14.1

When we talk about using other programming languages, we do not mean that the code written in those languages need to be converted to the Java code. For example, if the original code is in Python, it is not required that the code be converted to Java using Jython or any similar utility. Direct Python code can run in Hadoop ecosystem using Hadoop Streaming.

## 9.2 Prerequisites and Assumptions

Following are the prerequisites/assumptions we made before diving into the details of Hadoop Streaming:

- It is assumed that you are familiar with Hadoop and MapReduce or atleast knows the basics of it. In case you need some basic understanding of those you can refer to the following articles.
  - Hadoop Hello World Example
  - Apache Hadoop Distributed File System Explained
  - Apache Hadoop Wordcount Example
- It is also assumed that you understand the basics of running and setting up a Hadoop Cluster or atleast a single instance for testing purpose. In case you need help with that, you can refer to the following articles.
  - How to Install Apache Hadoop on Ubuntu
  - Apache Hadoop Cluster Setup Example (with Virtual Machines)

Once we have all this prerequisites setup and clear, we can dive into the details of Hadoop Streaming and check out some examples.

## 9.3 Hadoop Streaming Workflow

For using Hadoop Streaming, both the mapper and reducer need to be executables and should be able to read input from `stdin` line by line and emit the output to `stdout`

Hadoop Streaming API will create and submit a MapReduce job from the executables defined for Mapper and Reducers. On initialization of each Map or Reduce task, a new process will be started with the corresponding executable.

For each input data, the mapper task takes the input line-by-line and feed the lines to the `stdin` of the mapper executable. After execution, the lines from `stdout` are taken by mapper and converted to key-value pair which will be the output of the mapper task and will be passed on to the reducer task.

In the similar fashion, reducer takes the key-value pair and convert it into lines and feed the reducer executable using `stdin`. After reducer is executed, it again takes the line from `stdout` and convert it into the key-value pair to be passed on as the final result.

**Note:** By default, the text in the lines upto first tab will be taken as the key and the rest of the line as value. In case, there is no tab character present in the line, the whole line will be taken as the key and value will be null. But this behavior is not binding and can be changed is required and the required behavior can be configured.

## 9.4 MapReduce Code in Python

As discussed in the section above, we will use Hadoop Streaming API to run Python Code on Hadoop. We will use `sys.stdin` and `sys.stdout` in Python to read in the data and write out the output data, everything else will be handled by the Streaming API itself.

### 9.4.1 Wordcount Example

Wordcount, as you might be knowing is the basic program which is used to explain the basics of the Hadoop MapReduce framework. In wordcount program, a bunch of text input is provided to the Mapper function which splits the lines of text into single words and pass these single words as key-value pair to the Reducer functions. Reducer received the input as the key-value pair and counts the number of instances of a particular word in the provided input text and output the key-value pairs with word as the key and the number of counts as the value. If you are not familiar with basics of wordcount program, please refer to the article Apache Hadoop Wordcount Example for the detailed explanation. In this article we will implement the same wordcount example but instead of Java we will use Python and will run the MapReduce job using Hadoop Streaming API

### 9.4.2 Mapper

The Mapper function in Python will read the line from `stdin`, split the line in the individual words and output the word as key-value pair with value as 1 and word as the key. For example, `<word,1>`

```python
#!/usr/bin/env python

import sys

# read the input from stdin
for line in sys.stdin:
    # trim any leading and trailing spaces
    line = line.strip()

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

    # for each word in words, output key-value pair
    for word in words:
        # outputs the result to stdout
        # MapReduce Streaming API will take this output
        # and feed as the input to the Reduce step

        # tab-delimited
        # word count is always one
        print '%s\t%s' % (word, 1)
```

Above is the Python code to perform the Map task, now save it as `mapper.py` and make sure we have read and run permission for the python file.

### 9.4.3 Reducer

The Reducer will take the input from the `mapper.py` through `stdin`. Reducer then sums the occurrence of each word and output the file reduced output in the form of key-value pair having the particular word as key and the total occurrences of the word as the value. For example, `<word, 5>`

```
#!/usr/bin/env python

from operator import itemgetter
import sys

#variable initialization
current_word = None
current_count = 0
word = None

# takes input stdin
for line in sys.stdin:
    # trim any leading and trailing spaces
    line = line.strip()

    # split the input from mapper.py and take the word and its count
    word, count = line.split('\t', 1)

    # convert count string to int
    try:
        count = int(count)
    except ValueError:
        # in case of exception
        # ignore the exception and discard the input line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    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)
```

Above is the Python code to perform the reduce task. Save this code in the file `reducer.py` and also make sure this file also have read and execution permission.

## 9.5 Testing the Python code

Before submitting the Python code as MapReduce job to the Hadoop cluster, it is preferred that we test the code to confirm that it works as excepted. Is it easy to make sure that the code works fine with a small input text before submitting to the cluster to parse large amount of data. We can perform the following two tests:

- First test will be to test the Mapper code. Execute the following command in the console. It will run the `mapper.py` script with the given input string and we can confirm that the output is as expected.

```
echo "the quick brown fox jumps over the lazy dog" | /home/cloudera/mapper.py
```

The output should be as shown in the screenshot below: .Test for mapper.py Test for mapper.py

* Now we can also test the Reducer code. Execute the following command in the console.

```
echo "the quick brown fox jumps over the lazy dog" | /home/cloudera/mapper.py | sort -k1,1 ↩
    | /home/cloudera/reducer.py
```

The output of the above command should be as shown in the screenshot below: .Test for reducer.py Test for reducer.py

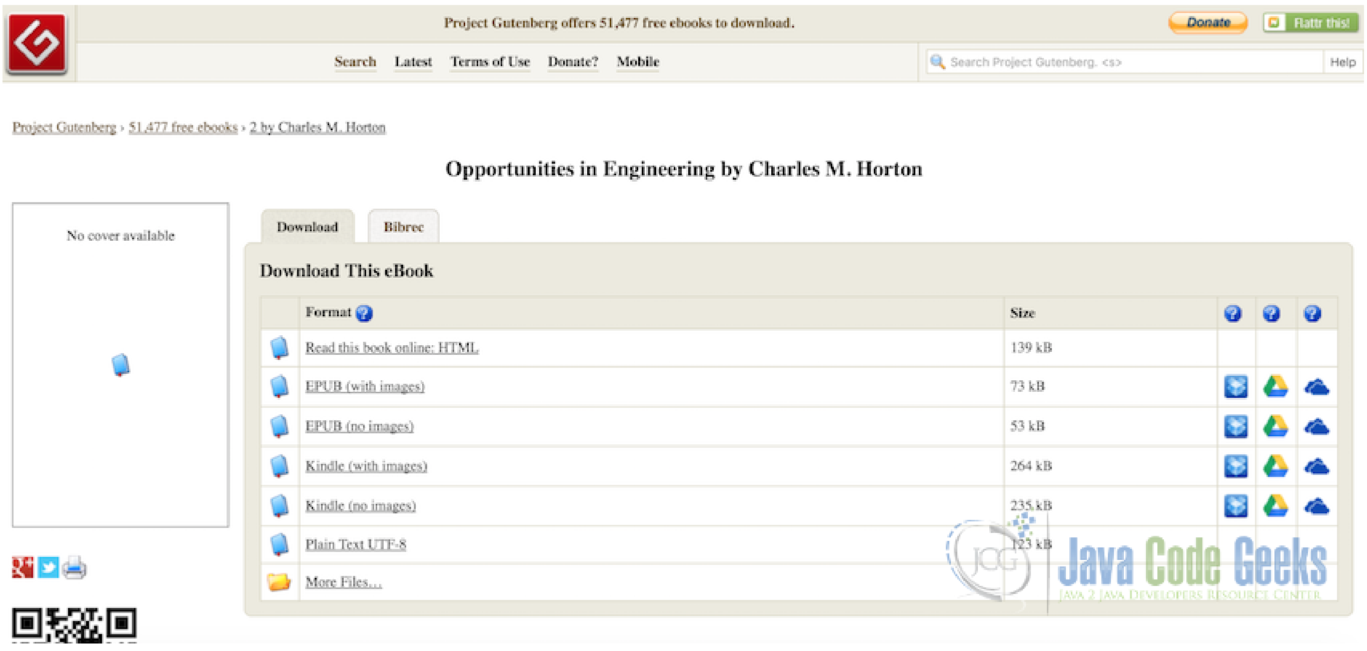
## 9.6 Submitting and Executing the Job on Hadoop cluster

In this section, we will learn how to run the Python MapReduce scripts on the Hadoop Cluster using Hadoop Streaming API.

### 9.6.1 Input Data

For this example, we will download a book from the Project Gutenberg which we will use as the input data for the MapReduce program. I have downloaded the book "Opportunities in Engineering by Charles M. Horton".

When you visit the webpage, you will find the book in many formats as shown in the screenshot below. Make sure to download the book in `Plain Text UTF-8` encoding format so that it can be easily read by the MapReduce program.



Project Gutenberg offers 51,477 free ebooks to download.

Search Latest Terms of Use Donate? Mobile

Project Gutenberg › 51,477 free ebooks › 2 by Charles M. Horton

Opportunities in Engineering by Charles M. Horton

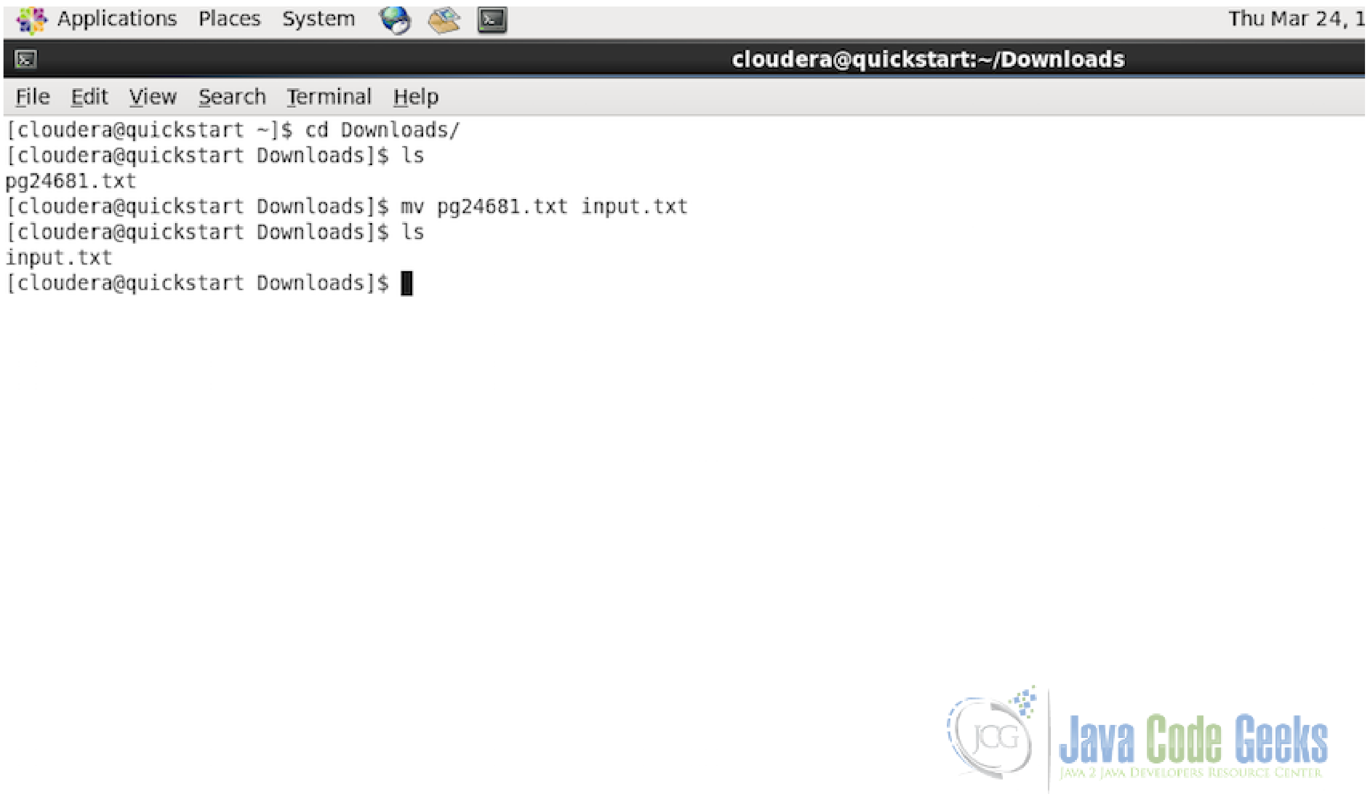
No cover available

Download This eBook

| Format | Size |
|---|---|
| Read this book online: HTML | 139 kB |
| EPUB (with images) | 73 kB |
| EPUB (no images) | 53 kB |
| Kindle (with images) | 264 kB |
| Kindle (no images) | 235 kB |
| Plain Text UTF-8 | 123 kB |
| More Files... | |

Figure 9.1: Project Gutenberg book download page

Once the book is downloaded, lets rename it to `input.txt` for easy reference


Applications Places System
Thu Mar 24, 1
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ cd Downloads/
[cloudera@quickstart Downloads]$ ls
pg24681.txt
[cloudera@quickstart Downloads]$ mv pg24681.txt input.txt
[cloudera@quickstart Downloads]$ ls
input.txt
[cloudera@quickstart Downloads]$ 
```

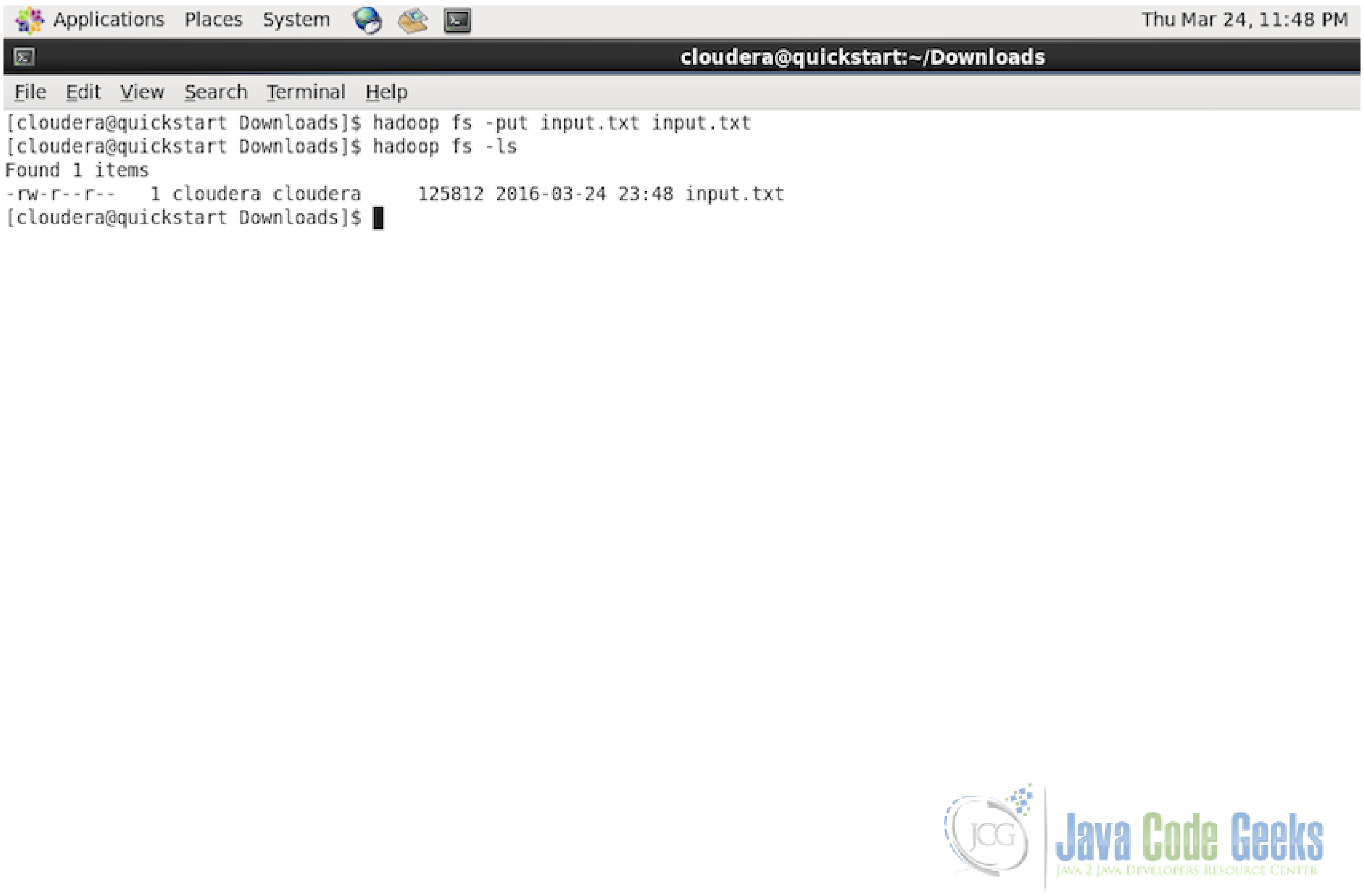
Figure 9.2: Renaming the file

### 9.6.2 Transferring input data to HDFS

MapReduce needs the input data to be present and accessible in the corresponding HDFS. So before we can run the MapReduce job, we need to transfer the book we just downloaded in the previous step to the HDFS. To do so, please use the following command:

```
hadoop fs -put input.txt input.txt
```

The above command, puts the `input.txt` file from the local system to the HDFS at the root location and with the name `input.txt` as shown in the screenshot below:


Applications Places System
Thu Mar 24, 11:48 PM
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help

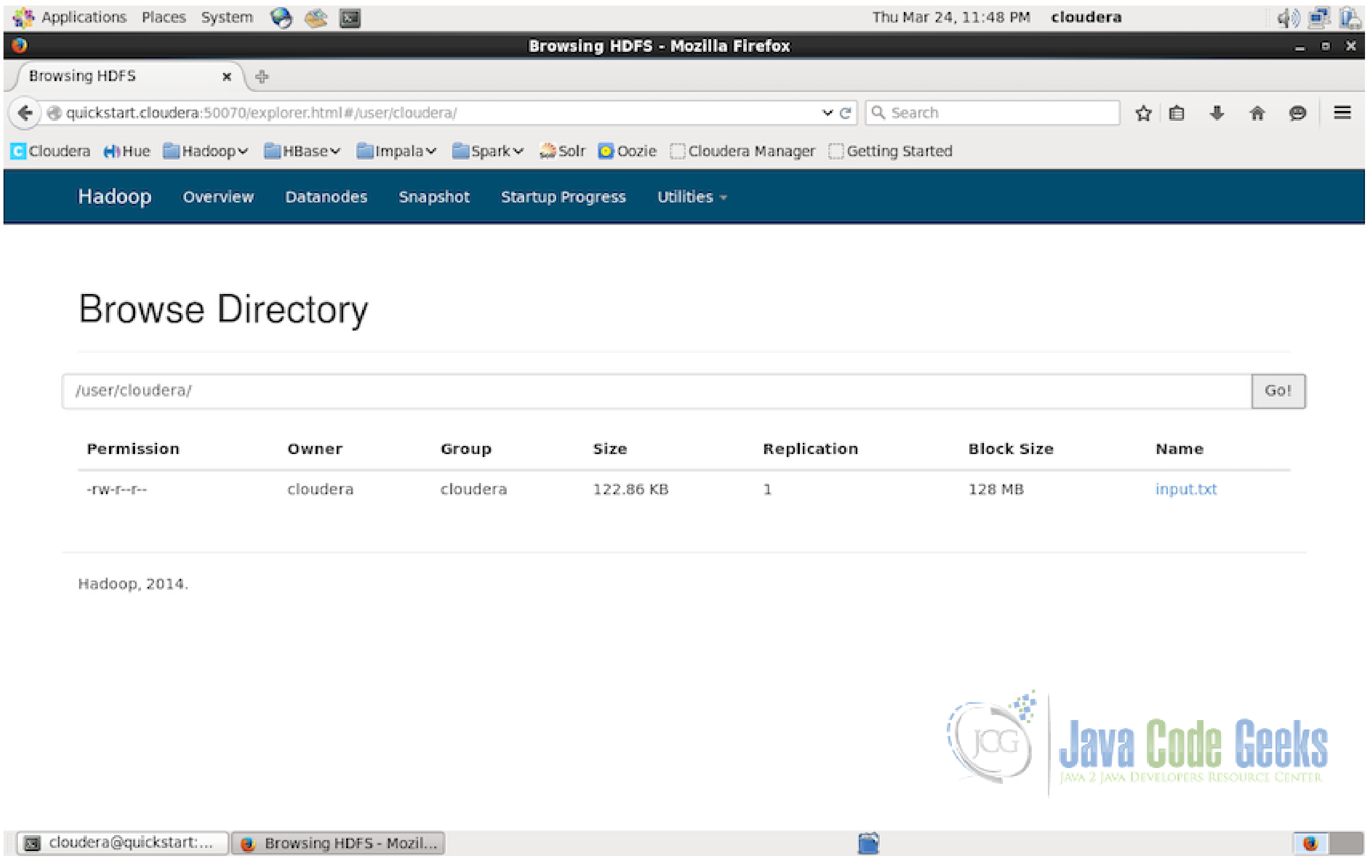
```
[cloudera@quickstart Downloads]$ hadoop fs -put input.txt input.txt
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera     125812 2016-03-24 23:48 input.txt
[cloudera@quickstart Downloads]$ 
```

Figure 9.3: Tranferring the input file from local to HDFS

You can check if the file is successfully transferred using the command:

```
hadoop fs -ls
```

or from the Hadoop User Panel


Applications Places System
Thu Mar 24, 11:48 PM cloudera
Browsing HDFS - Mozilla Firefox
Browsing HDFS
quickstart.cloudera:50070/explorer.html#/user/cloudera/
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Hadoop Overview Datanodes Snapshot Startup Progress Utilities
Go!
cloudera@quickstart:...
Browsing HDFS - Mozil...

Browse Directory

/user/cloudera/

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | cloudera | cloudera | 122.86 KB | 1 | 128 MB | input.txt |

Hadoop, 2014.

Figure 9.4: Hadoop User Panel

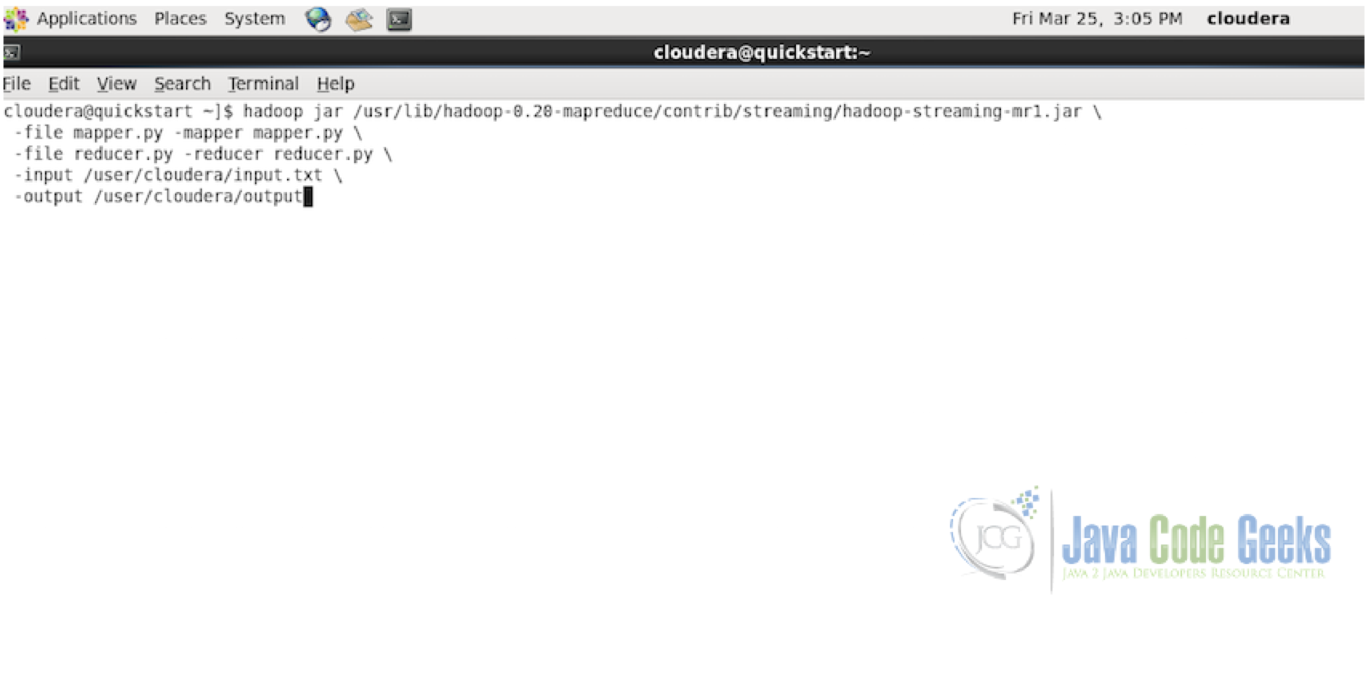
With the successful completion of this step we are now ready to submit the Python MapReduce job to Hadoop cluster.

### 9.6.3 Submitting the MapReduce Job

For running the job on Hadoop Cluster we will use the Streaming API so that the data can be passed between the Mapper and the Reducer using `stdin` and `stdout`. Following is the command used to submit and run the job:

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-mr1.jar \
-file mapper.py   -mapper mapper.py \
-file reducer.py   -reducer reducer.py \
-input /user/cloudera/input.txt \
-output /user/cloudera/output
```

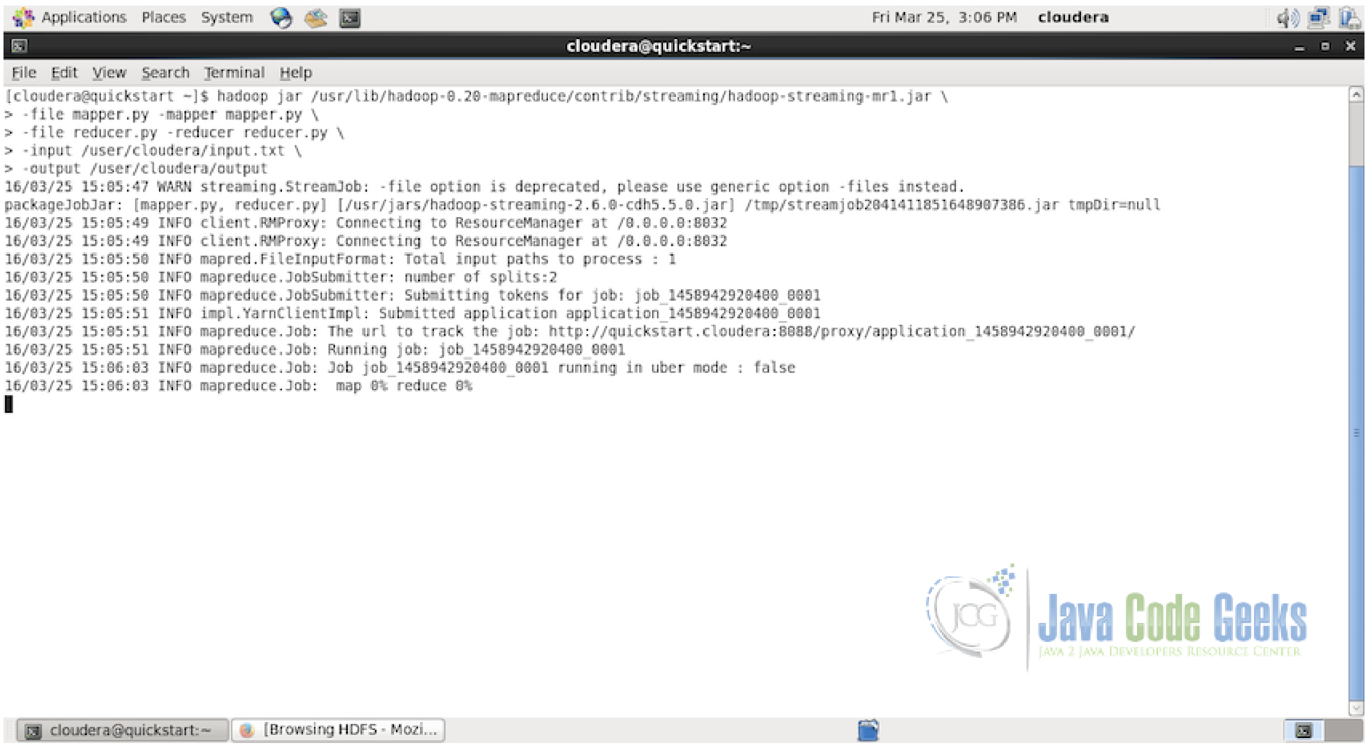
Following is the screenshot of complete command in the console:



```
cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-mr1.jar \
 -file mapper.py -mapper mapper.py \
 -file reducer.py -reducer reducer.py \
 -input /user/cloudera/input.txt \
 -output /user/cloudera/output
```

Figure 9.5: Submitting the job

If the job is successfully submitted and running, you will see the console out similar to the one in the screenshot below:

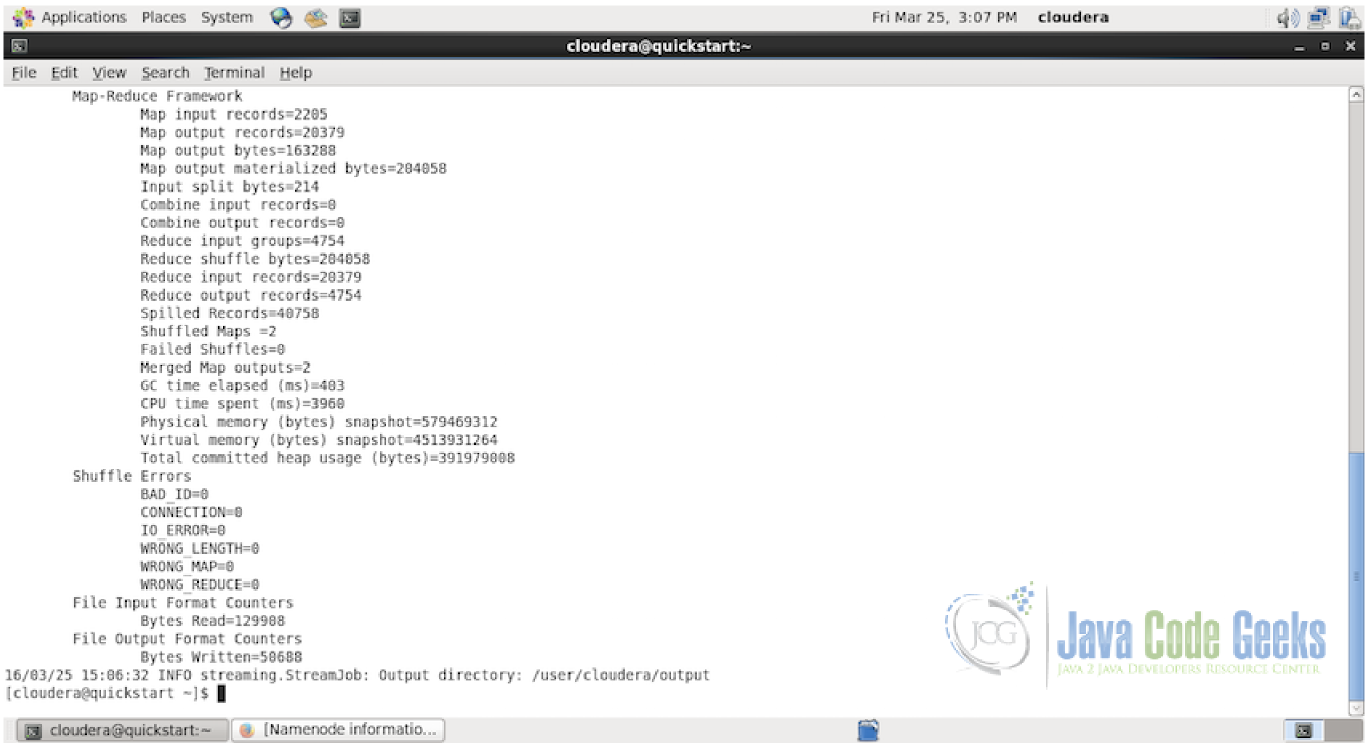


```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-mr1.jar \
> -file mapper.py -mapper mapper.py \
> -file reducer.py -reducer reducer.py \
> -input /user/cloudera/input.txt \
> -output /user/cloudera/output
16/03/25 15:05:47 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [mapper.py, reducer.py] [/usr/jars/hadoop-streaming-2.6.0-cdh5.5.0.jar] /tmp/streamjob2041411851648907386.jar tmpDir=null
16/03/25 15:05:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/25 15:05:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/25 15:05:50 INFO mapred.FileInputFormat: Total input paths to process : 1
16/03/25 15:05:50 INFO mapreduce.JobSubmitter: number of splits:2
16/03/25 15:05:50 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1458942920400_0001
16/03/25 15:05:51 INFO impl.YarnClientImpl: Submitted application application_1458942920400_0001
16/03/25 15:05:51 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1458942920400_0001/
16/03/25 15:05:51 INFO mapreduce.Job: Running job: job_1458942920400_0001
16/03/25 15:06:03 INFO mapreduce.Job: Job job_1458942920400_0001 running in uber mode : false
16/03/25 15:06:03 INFO mapreduce.Job:  map 0% reduce 0%
```

Figure 9.6: Console Log of Job Submission

Notice the console log assigned a job id to the MapReduce job and started running the job.

Once the job is finished without any exceptions or errors, you will see the following console log with the last line mentioning the path where the output of the job is stored.



```
        Map-Reduce Framework
                Map input records=2205
                Map output records=20379
                Map output bytes=163288
                Map output materialized bytes=204058
                Input split bytes=214
                Combine input records=0
                Combine output records=0
                Reduce input groups=4754
                Reduce shuffle bytes=204058
                Reduce input records=20379
                Reduce output records=4754
                Spilled Records=40758
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=403
                CPU time spent (ms)=3960
                Physical memory (bytes) snapshot=579469312
                Virtual memory (bytes) snapshot=4513931264
                Total committed heap usage (bytes)=391979008
        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=129988
        File Output Format Counters
                Bytes Written=50688
16/03/25 15:06:32 INFO streaming.StreamJob: Output directory: /user/cloudera/output
[cloudera@quickstart ~]$ 
```

Figure 9.7: Console output of the job

### 9.6.4 Understanding the Console Log

Successful execution of the MapReduce job will output a significant amount of log to the console. There are few important parts of the log which you should be aware of. Following is the complete console log of the execution of the above MapReduce job.

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop ↩
    -streaming-mr1.jar \
> -file mapper.py -mapper mapper.py \
> -file reducer.py -reducer reducer.py \
> -input /user/cloudera/input.txt \
> -output /user/cloudera/output
16/03/25 15:05:47 WARN streaming.StreamJob: -file option is deprecated, please use generic  ↩
    option -files instead.
packageJobJar: [mapper.py, reducer.py] [/usr/jars/hadoop-streaming-2.6.0-cdh5.5.0.jar] /tmp ↩
    /streamjob2041411851648907386.jar tmpDir=null
16/03/25 15:05:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/25 15:05:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/25 15:05:50 INFO mapred.FileInputFormat: Total input paths to process : 1
16/03/25 15:05:50 INFO mapreduce.JobSubmitter: number of splits:2
16/03/25 15:05:50 INFO mapreduce.JobSubmitter: Submitting tokens for job:  ↩
    job_1458942920400_0001
16/03/25 15:05:51 INFO impl.YarnClientImpl: Submitted application  ↩
    application_1458942920400_0001
16/03/25 15:05:51 INFO mapreduce.Job: The url to track the job: https://quickstart.cloudera ↩
    :8088/proxy/application_1458942920400_0001/
16/03/25 15:05:51 INFO mapreduce.Job: Running job: job_1458942920400_0001
16/03/25 15:06:03 INFO mapreduce.Job: Job job_1458942920400_0001 running in uber mode :  ↩
    false
16/03/25 15:06:03 INFO mapreduce.Job:  map 0% reduce 0%
16/03/25 15:06:20 INFO mapreduce.Job:  map 100% reduce 0%
16/03/25 15:06:32 INFO mapreduce.Job:  map 100% reduce 100%
16/03/25 15:06:32 INFO mapreduce.Job: Job job_1458942920400_0001 completed successfully
```

```
16/03/25 15:06:32 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=204052
                FILE: Number of bytes written=753127
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=130122
                HDFS: Number of bytes written=50688
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=29025
                Total time spent by all reduces in occupied slots (ms)=10319
                Total time spent by all map tasks (ms)=29025
                Total time spent by all reduce tasks (ms)=10319
                Total vcore-seconds taken by all map tasks=29025
                Total vcore-seconds taken by all reduce tasks=10319
                Total megabyte-seconds taken by all map tasks=29721600
                Total megabyte-seconds taken by all reduce tasks=10566656
        Map-Reduce Framework
                Map input records=2205
                Map output records=20379
                Map output bytes=163288
                Map output materialized bytes=204058
                Input split bytes=214
                Combine input records=0
                Combine output records=0
                Reduce input groups=4754
                Reduce shuffle bytes=204058
                Reduce input records=20379
                Reduce output records=4754
                Spilled Records=40758
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=403
                CPU time spent (ms)=3960
                Physical memory (bytes) snapshot=579469312
                Virtual memory (bytes) snapshot=4513931264
                Total committed heap usage (bytes)=391979008
        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=129908
        File Output Format Counters
                Bytes Written=50688
16/03/25 15:06:32 INFO streaming.StreamJob: Output directory: /user/cloudera/output
```

Following are the important parts of the whole console log:

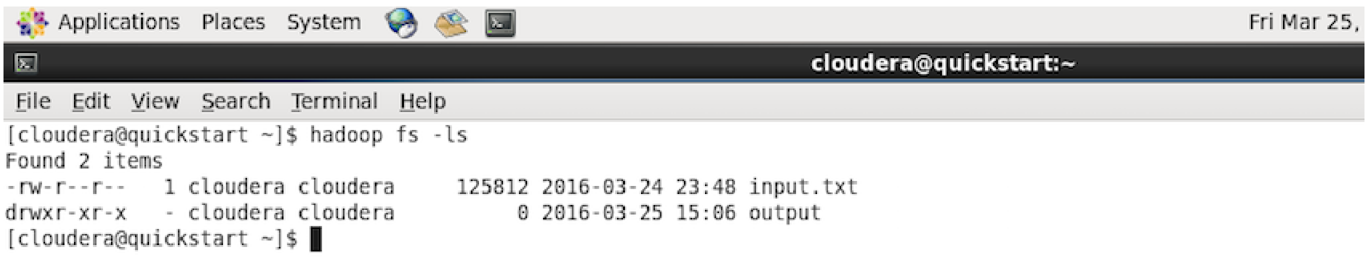
- Line 10: Logs the total number of paths to be processed for the job. Here we have only one input file so the path to be processed is also 1.

- Line 14: Logs the url which can be used to track the progess of the job in the web-browser
- Line 17-19: Logs the progress of the Map and the Reduce taks respectively.
- Line 20: Informs that the job is completed succesfully and after this the console log will display the stats of the job.
- Line 22-32:Displays the file system stats including the number of bytes read, total number of bytes written, number of read operations and number of write operations
- Line 33-44:Displays the Job stats including total number of map and reduce jobs launched (2 and 1 respectively in this case), total time spent by map tasks and reduce tasks for exections etc.
- Line 45-64:Display the MapReduce Framework stats including the Map and Reduce records processed, total CPU time spent in processing, amount of physical and virtual memory used etc.
- Line 77: Finally the line 77 logs the path where the output of the MapReduce job is stored.

### 9.6.5 MapReduce Job Output

So after the successful execution of the job, the output data is present at the provided path. We can check if the output directory is present using the command:

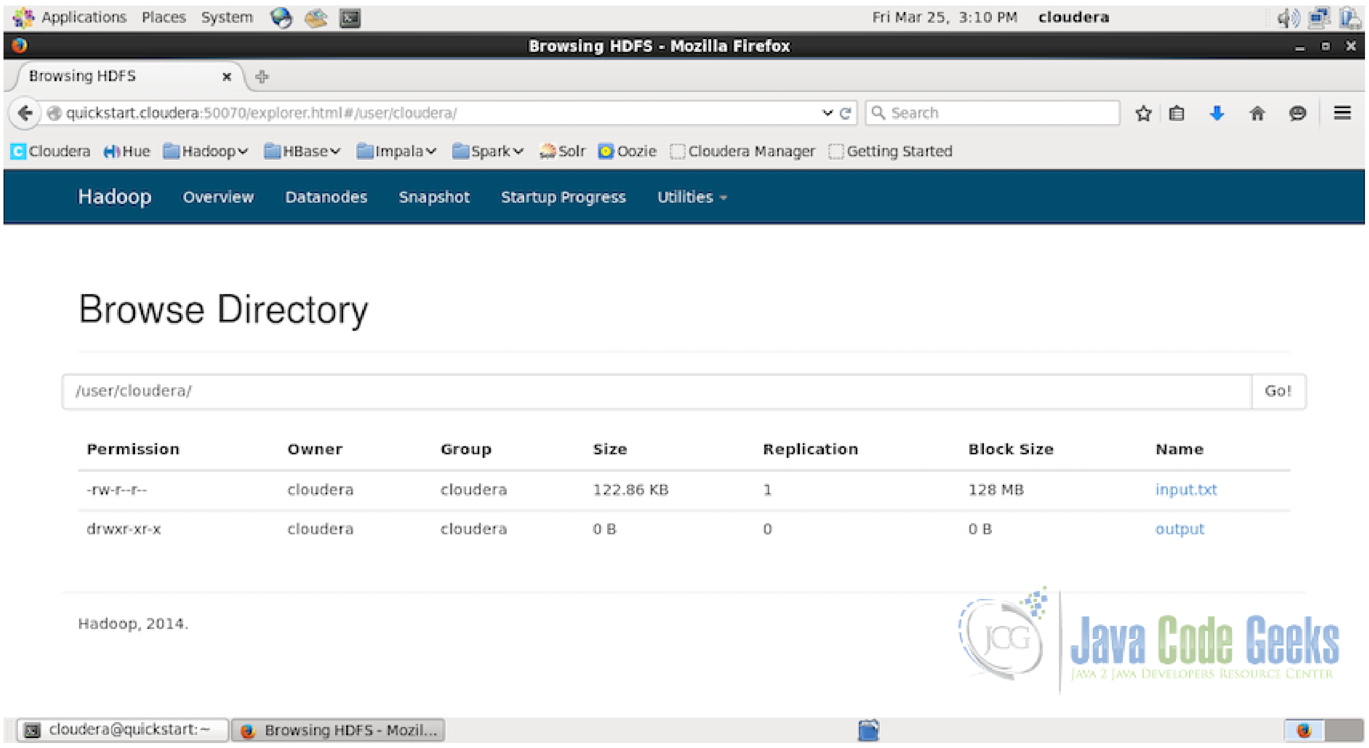
```
hadoop fs -ls
```


Applications Places System
Fri Mar 25,
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 cloudera cloudera     125812 2016-03-24 23:48 input.txt
drwxr-xr-x   - cloudera cloudera          0 2016-03-25 15:06 output
[cloudera@quickstart ~]$
```

Figure 9.8: Checking the output data

or through the Hadoop user interface:


Applications Places System
Fri Mar 25, 3:10 PM cloudera
Browsing HDFS - Mozilla Firefox
Browsing HDFS
quickstart.cloudera:50070/explorer.html#/user/cloudera/
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Hadoop Overview Datanodes Snapshot Startup Progress Utilities
cloudera@quickstart:~
Browsing HDFS - Mozil...

Browse Directory

/user/cloudera/ Go!

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | cloudera | cloudera | 122.86 KB | 1 | 128 MB | input.txt |
| drwxr-xr-x | cloudera | cloudera | 0 B | 0 | 0 B | output |

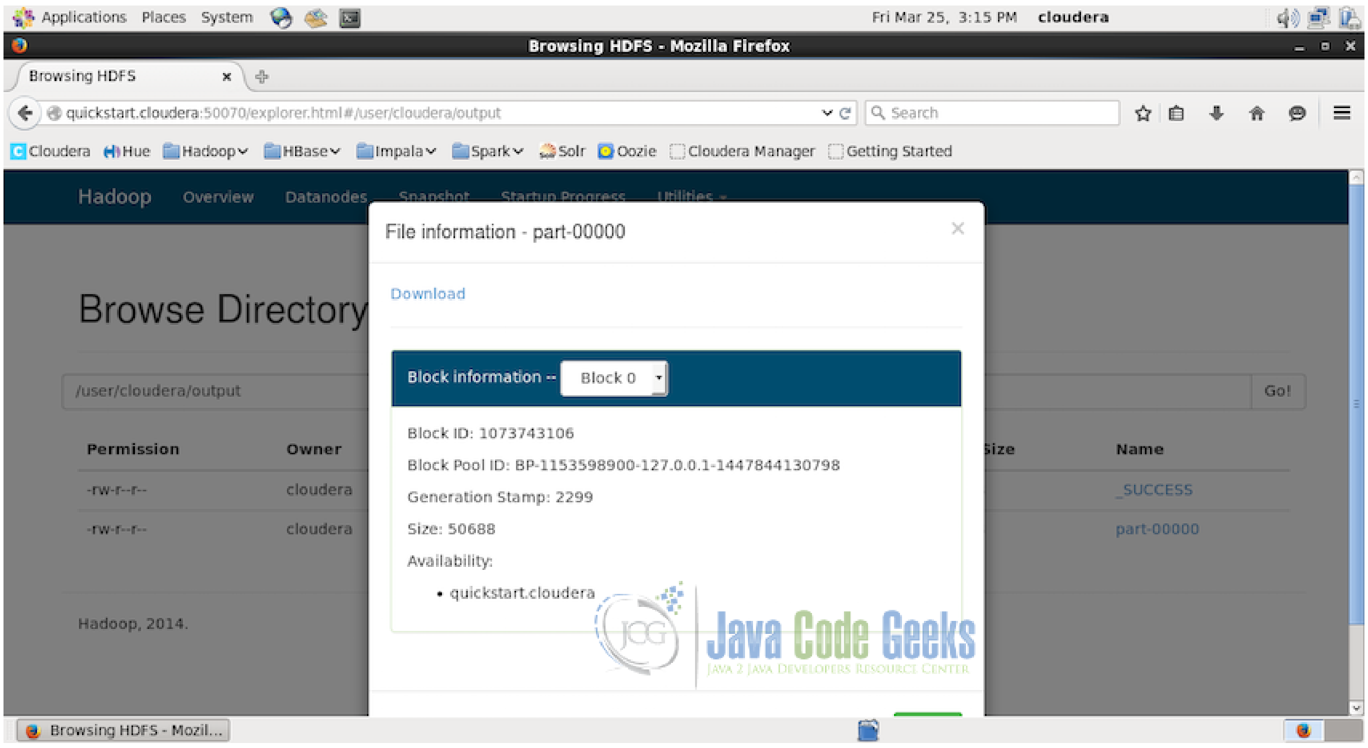
Hadoop, 2014.

Figure 9.9: Output file in UI

This output file can be downloaded either using the command:

```
hadoop fs -get output
```

or directly using the user interface:


Browsing HDFS - Mozilla Firefox
quickstart.cloudera:50070/explorer.html#/user/cloudera/output
Hadoop Overview Datanodes Snapshot Startup Progress Utilities
Browse Directory
/user/cloudera/output
Go!
Permission Owner Size Name
Download
Block information -- Block 0
Hadoop, 2014.

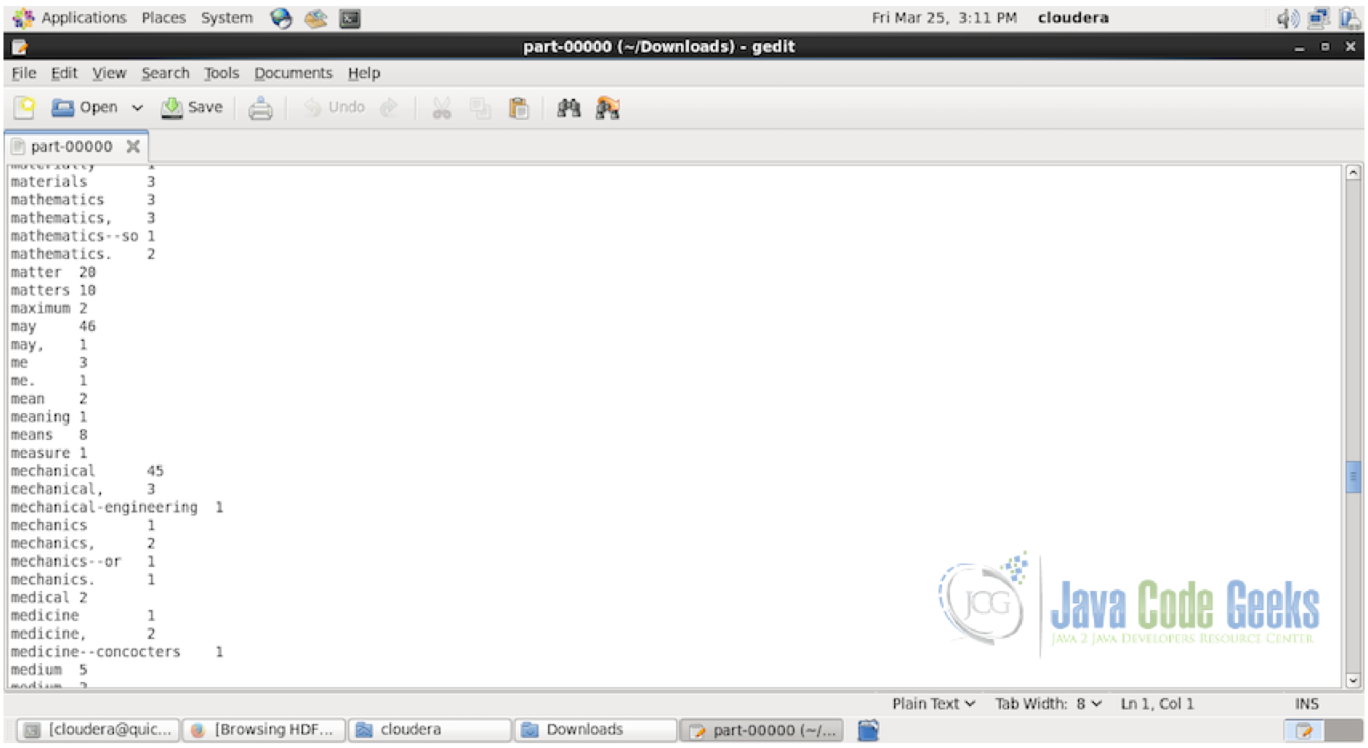
File information - part-00000

| Permission | Owner | Name |
|---|---|---|
| -rw-r--r-- | cloudera | _SUCCESS |
| -rw-r--r-- | cloudera | part-00000 |

Block ID: 1073743106
Block Pool ID: BP-1153598900-127.0.0.1-1447844130798
Generation Stamp: 2299
Size: 50688
Availability:
- quickstart.cloudera

Figure 9.10: Downloading the output

After downloading the `output` directly should have a text file with the name `part-00000` which contains the output of the job. Following is the screenshot of the part of the file:


part-00000 (~/Downloads) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
part-00000
Plain Text Tab Width: 8 Ln 1, Col 1 INS

```
materials       3
mathematics     3
mathematics,    3
mathematics--so 1
mathematics.    2
matter  28
matters 18
maximum 2
may     46
may,    1
me      3
me.     1
mean    2
meaning 1
means   8
measure 1
mechanical      45
mechanical,     3
mechanical-engineering  1
mechanics       1
mechanics,      2
mechanics--or   1
mechanics.      1
medical 2
medicine        1
medicine,       2
medicine--concocters    1
medium  5
```

Figure 9.11: Output file

## 9.7 Conclusion

This brings us to the end of the article, to conclude, we started by understanding the basic working of the Hadoop Streaming API and its complete workflow, we saw how MapReduce code can be written in Python and how the Streaming API can be used to run the jobs on the Hadoop Cluster.

We followed the theoretical understanding with the actually WordCount example in Python, we learnt how to submit the job using Streaming API, how to interpret the console log of the Hadoop job and finally how to get the output of the processed job for further usage.

## 9.8 Download the Source Code

The following download package contains the `mapper.py` and `reducer.py` scripts used in the article.

**Download** You can download the full source code of this example here: **HadoopStreaming_WordCount_Python**

# Chapter 10

# Zookeeper Example

In this example, we will explore Apache Zookeeper, starting with the introduction and then followed by the steps to setup the Zookeeper and to get it up and running.

## 10.1 Introduction

Apache Zookeeper is the building block of distributed systems. When a distributed system is designed there is always a need of developing and deploying something which can coordinate through the cluster. This is where Zookeeper comes into the picture. It is an open-source project maintained by Apache for maintenance and coordination of the distributed cluster. Some of the services provided by Zookeeper are:

- **Naming Service:** A name service is used to map a name to some sort of data which can then be accessed using this name. For example, DNS servers map to the ip address of the server and then client can access the server using that url name. In distributed systems we may need to check the status of servers or nodes using the name assigned to them. This can be done by using the naming service interface provided by default by Zookeeper.
- **Configuration Management:** Zookeeper also provides the option to manage the configuration of distributed system centrally. Configuration can be stored centrally on Zookeeper and any new node on joining the distributed system can pick the configuration from Zookeeper. This makes managing configuration quite easy and effort free.
- **Leader Election:** Distributed Systems usually needs an automatic fail-over strategy in case some nodes fails. Zookeeper provides an option to do so using leader election functionality.
- **Locking:** In every distributed system, there will be some shared resources and multiple services may need to access this. So to allow serialized access to this resource, a locking mechanism is required. Zookeeper provides this functionality.
- **Synchronization:** The access to the shared resources also need to the synchronized in the distributed setup. Zookeeper also provides a simple interface for this.

## 10.2 How Zookeeper Works?

Zookeeper follows a client-server model. In which clients are the machines in the cluster. These machines are also called nodes. These clients consume the service provided by the servers. Zookeeper coordinates the distributed system but it in itself is also a distributed system. The collection of Zookeeper servers in distributed mode is called Zookeeper ensemble.

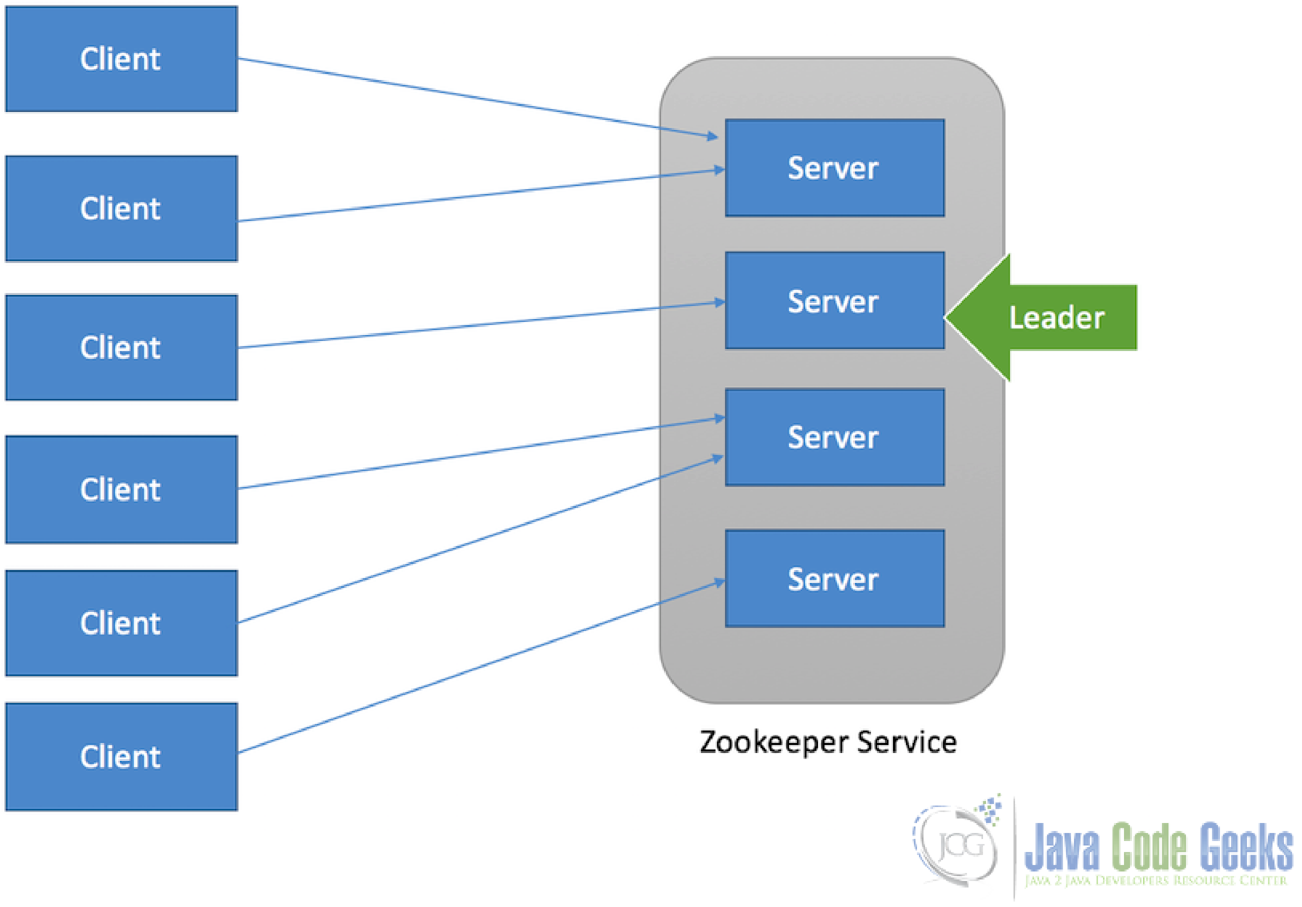


Figure 10.1: Zookeeper Client-Server Architecture

At any given time, one client can be connected to only one Zookeeper server but each zookeeper server can handle multiple clients at the time. Clients send pings(heartbeats) to the server periodically to let it know that it is alive and connected to the server. Zookeeper server also responds with an acknowledgement informing that it is alive and connected as well. The frequency of these pings/heartbeats can be set in the configuration file which we will see in the next section.

In case the client does not receive an acknowledgement from the server it is connected to within the specified time period, the client then tries to connect to the another server from the pool and on the successful connection the client session is transferred to the new Zookeeper server it is connected to.

Zookeeper follows a hierarchical system similar to the file system to store data in the nodes and it is called znodes. Znode is derived from "Zookeeper data nodes". Each znode acts as a directory and can have multiple sub-node and the hierarchy continues. To access the znodes also, Zookeeper follows the file path like structure. For example: the path to znode firstnode and the corresponsing sub nodes can look like this, `/firstnode/sub-node/sub-sub-node`

## 10.3 Zookeeper Setup

In this section, we will go through the steps to setup the Zookeeper server on the `localhost` for experimentation purpose. Zookeeper provides a single server in the package and can be directly run on the machine.

### 10.3.1 System Requirements

- Java, JDK 6 or later (We will use JDK 8)

- Minimum 2GB RAM
- Dual Core Processor
- Linux OS. Linux is supported as both development and production systems. Both Windows and MacOSX are only supported as the development system and not as production systems.

### 10.3.2 Install Java

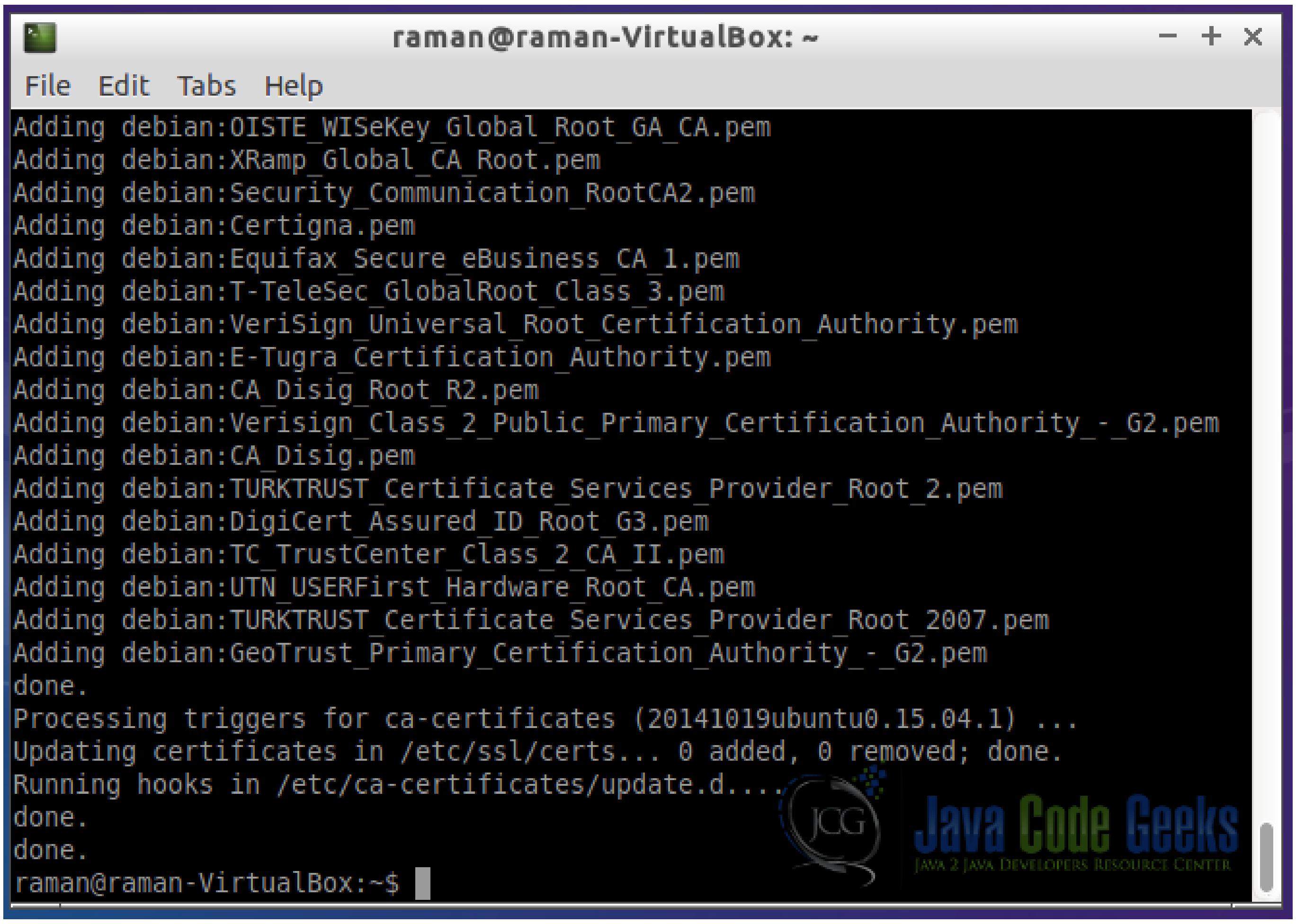
First of all, we will check if Java is installed on the system and if not, we need to install Java first. To check if Java is installed use:

```
java -version
```

If this returns the Java version number then Java is installed. Make sure it is atleast JDK 6 or higher. In case Java is not installed, we have to install it first. Use the following commands to install Java JDK 8.

```
sudo apt-get update
sudo apt-get intstall openjdk-8-jre-headless
```

Th first command will update all the packages already installed and the second command will install the OpenJDK 8. Following is the console output we get after running the above commands:



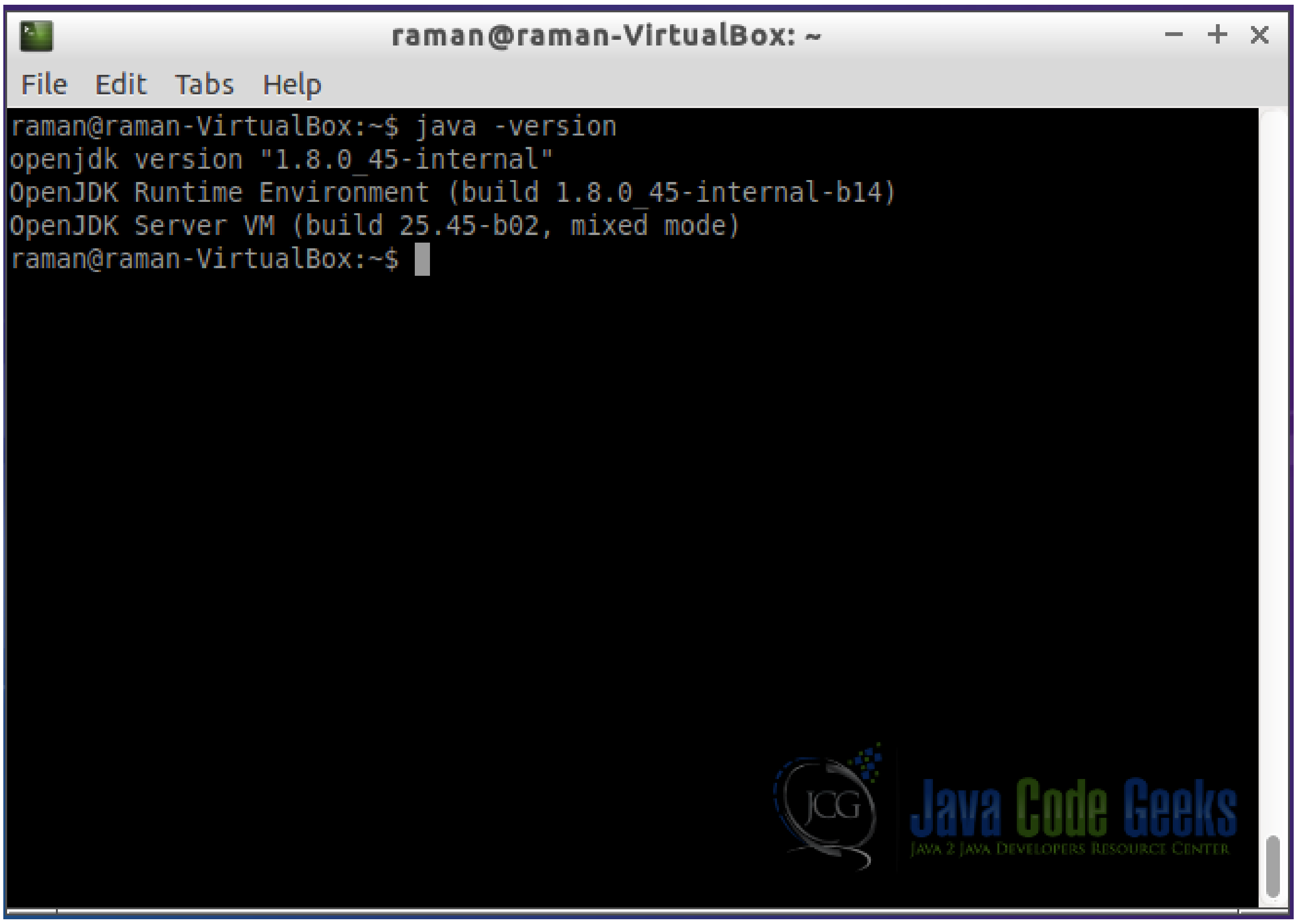
```
Adding debian:OISTE_WISeKey_Global_Root_GA_CA.pem
Adding debian:XRamp_Global_CA_Root.pem
Adding debian:Security_Communication_RootCA2.pem
Adding debian:Certigna.pem
Adding debian:Equifax_Secure_eBusiness_CA_1.pem
Adding debian:T-TeleSec_GlobalRoot_Class_3.pem
Adding debian:VeriSign_Universal_Root_Certification_Authority.pem
Adding debian:E-Tugra_Certification_Authority.pem
Adding debian:CA_Disig_Root_R2.pem
Adding debian:Verisign_Class_2_Public_Primary_Certification_Authority_-_G2.pem
Adding debian:CA_Disig.pem
Adding debian:TURKTRUST_Certificate_Services_Provider_Root_2.pem
Adding debian:DigiCert_Assured_ID_Root_G3.pem
Adding debian:TC_TrustCenter_Class_2_CA_II.pem
Adding debian:UTN_USERFirst_Hardware_Root_CA.pem
Adding debian:TURKTRUST_Certificate_Services_Provider_Root_2007.pem
Adding debian:GeoTrust_Primary_Certification_Authority_-_G2.pem
done.
Processing triggers for ca-certificates (20141019ubuntu0.15.04.1) ...
Updating certificates in /etc/ssl/certs... 0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....
done.
done.
raman@raman-VirtualBox:~$
```

Figure 10.2: Console output after installing Java JDK 8

To check if the installation was successful, again run the command

```
java -version
```

the output should be something similar to what is displayed in the following screenshot:

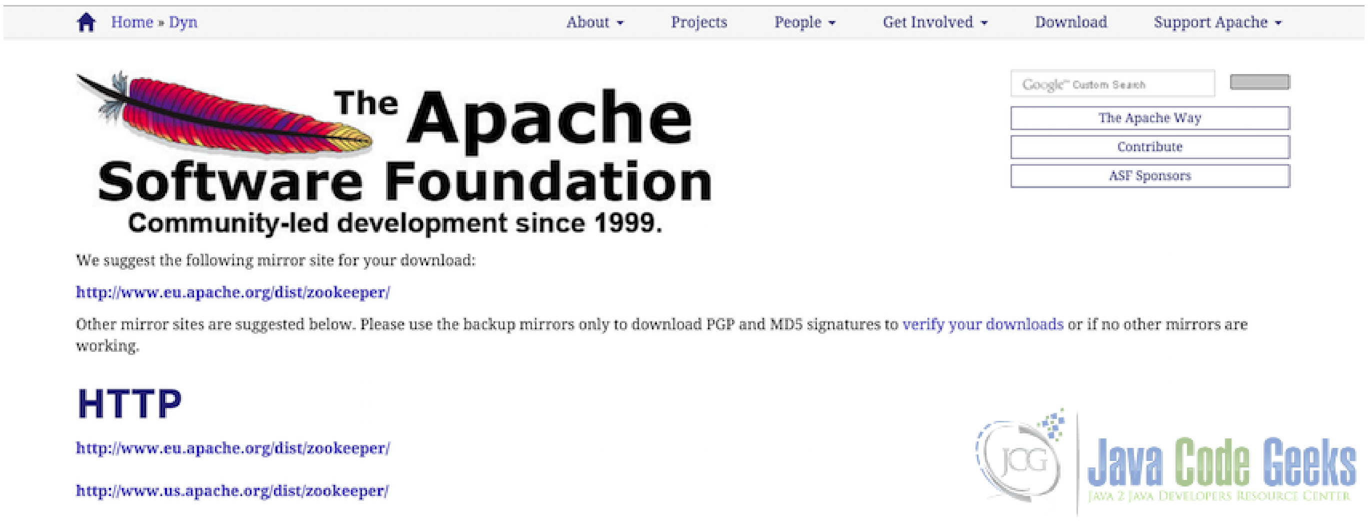


```
raman@raman-VirtualBox:~$ java -version
openjdk version "1.8.0_45-internal"
OpenJDK Runtime Environment (build 1.8.0_45-internal-b14)
OpenJDK Server VM (build 25.45-b02, mixed mode)
raman@raman-VirtualBox:~$
```

Figure 10.3: Console Output for Java Version

### 10.3.3 Download Zookeeper

Next step is to download the stable release version of Zookeeper from Resease site. Download manually the stable version from the Download section of the release site(at the time of writing, stable release is 3.4.6). We can use any of the mirrors mentioned in the site(as shown in the screenshot below) and unzip/untar to the desired folder.



We suggest the following mirror site for your download:

http://www.eu.apache.org/dist/zookeeper/

Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures to verify your downloads or if no other mirrors are working.

HTTP

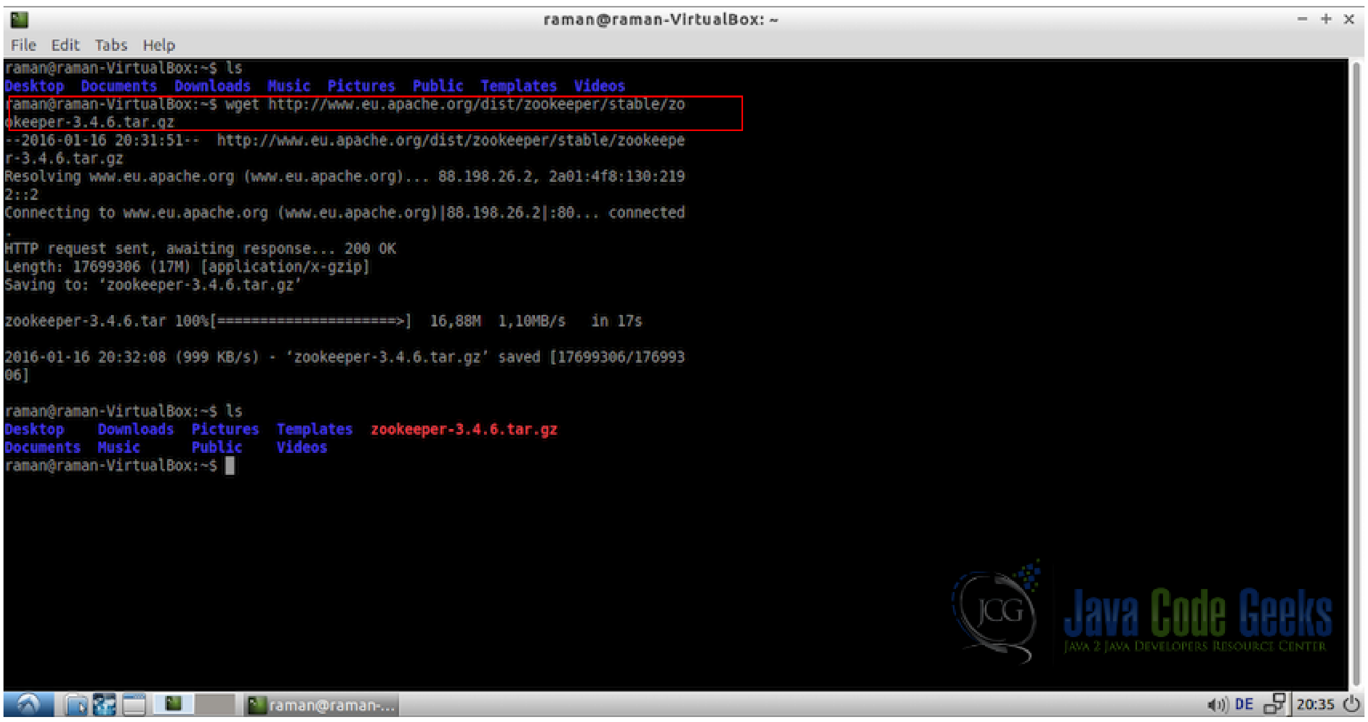
http://www.eu.apache.org/dist/zookeeper/

http://www.us.apache.org/dist/zookeeper/

Figure 10.4: Apache Zookeeper Mirrors

or use the following commands to download and untar.

```
wget https://www.eu.apache.org/dist/zookeeper/stable/zookeeper-3.4.6.tar.gz
tar -xvf zookeeper-3.4.6.tar.gz
cd zookeeper-3.4.6/
```



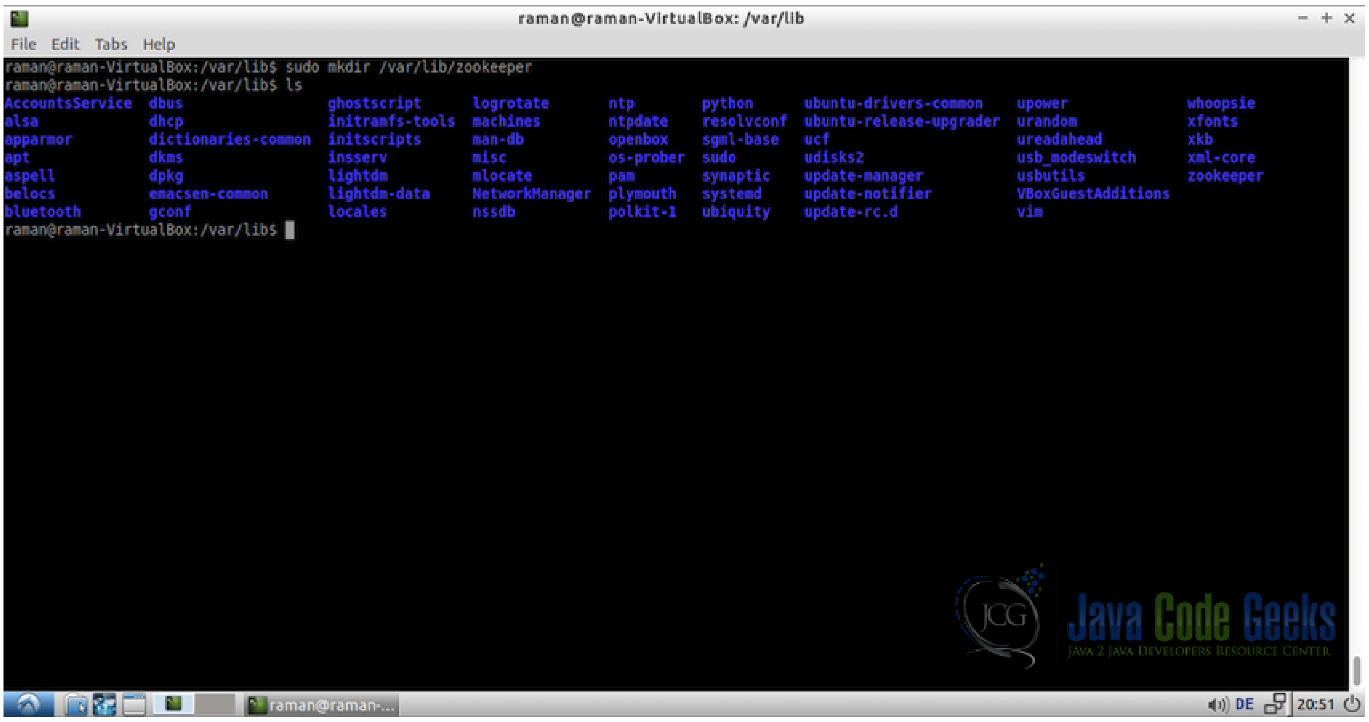
```
raman@raman-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
raman@raman-VirtualBox:~$ wget http://www.eu.apache.org/dist/zookeeper/stable/zo
okeeper-3.4.6.tar.gz
--2016-01-16 20:31:51--  http://www.eu.apache.org/dist/zookeeper/stable/zookeepe
r-3.4.6.tar.gz
Resolving www.eu.apache.org (www.eu.apache.org)... 88.198.26.2, 2a01:4f8:130:219
2::2
Connecting to www.eu.apache.org (www.eu.apache.org)|88.198.26.2|:80... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 17699306 (17M) [application/x-gzip]
Saving to: ‘zookeeper-3.4.6.tar.gz’

zookeeper-3.4.6.tar 100%[=====================>]  16,88M  1,10MB/s   in 17s

2016-01-16 20:32:08 (999 KB/s) - ‘zookeeper-3.4.6.tar.gz’ saved [17699306/176993
06]

raman@raman-VirtualBox:~$ ls
Desktop    Downloads  Pictures  Templates  zookeeper-3.4.6.tar.gz
Documents  Music      Public    Videos
raman@raman-VirtualBox:~$ 
```

Figure 10.5: Downloading the stable Zookeeper version

### 10.3.4 Data Directory

Next we need a directory to store the data related to the znodes and other zookeeper metadata. For that we will create a new directory in `/var/lib/` by the name `zookeeper`

```
sudo mkdir /var/lib/zookeeper
cd /var/lib
ls
```



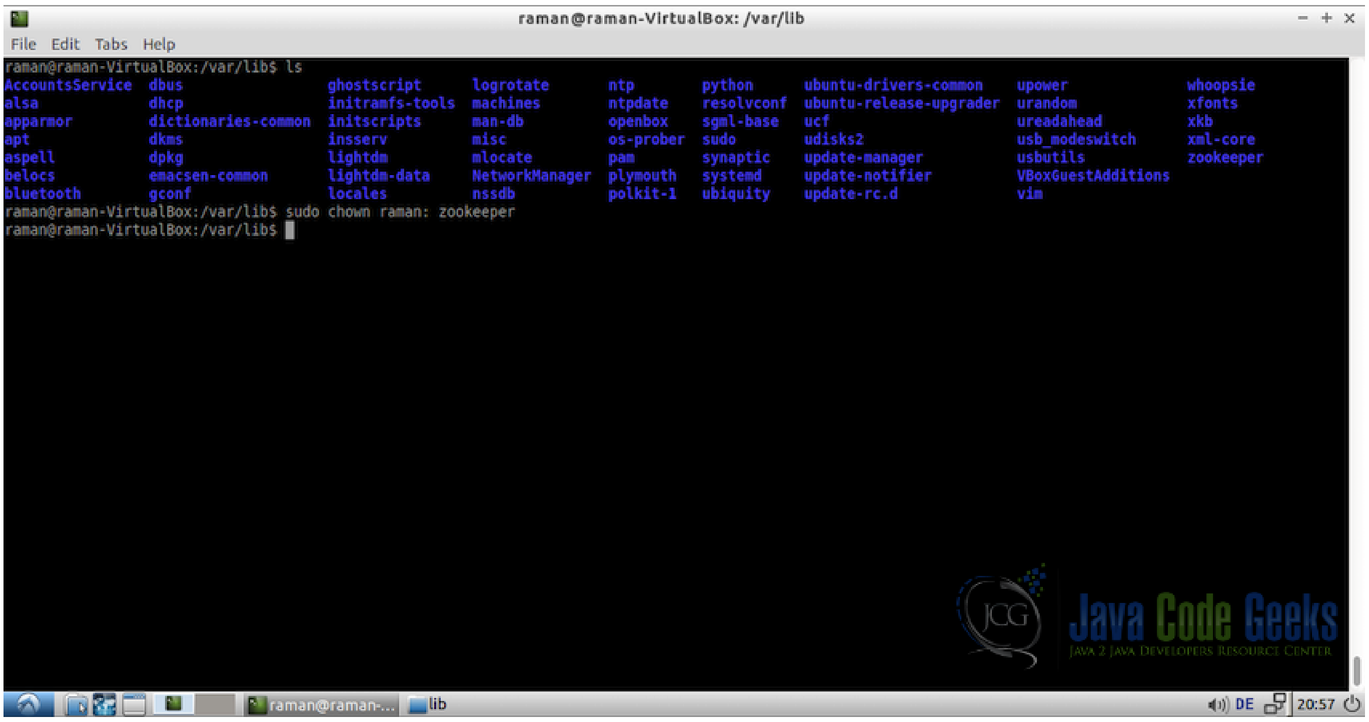
```
raman@raman-VirtualBox:/var/lib$ sudo mkdir /var/lib/zookeeper
raman@raman-VirtualBox:/var/lib$ ls
AccountsService  dbus                 ghostscript     logrotate       ntp         python      ubuntu-drivers-common    upower              whoopsie
alsa             dhcp                 initramfs-tools machines        ntpdate     resolvconf  ubuntu-release-upgrader  urandom             xfonts
apparmor         dictionaries-common  initscripts     man-db          openbox     sgml-base   ucf                      ureadahead          xkb
apt              dkms                 insserv         misc            os-prober   sudo        udisks2                  usb_modeswitch      xml-core
aspell           dpkg                 lightdm         mlocate         pam         synaptic    update-manager           usbutils            zookeeper
belocs           emacsen-common       lightdm-data    NetworkManager  plymouth    systemd     update-notifier          VBoxGuestAdditions
bluetooth        gconf                locales         nssdb           polkit-1    ubiquity    update-rc.d              vim
raman@raman-VirtualBox:/var/lib$
```

Figure 10.6: Make Zokeeper Data Directory

When this directory is created using `sudo`, it will by default be with `root` as the owner which we need to change to the user where Zookeeper will be running so that Zookeeper server can access the directory without any trouble. To change the user, run the following command from the folder `/var/lib`

```
cd /var/lib
sudo chown raman: zookeeper
```

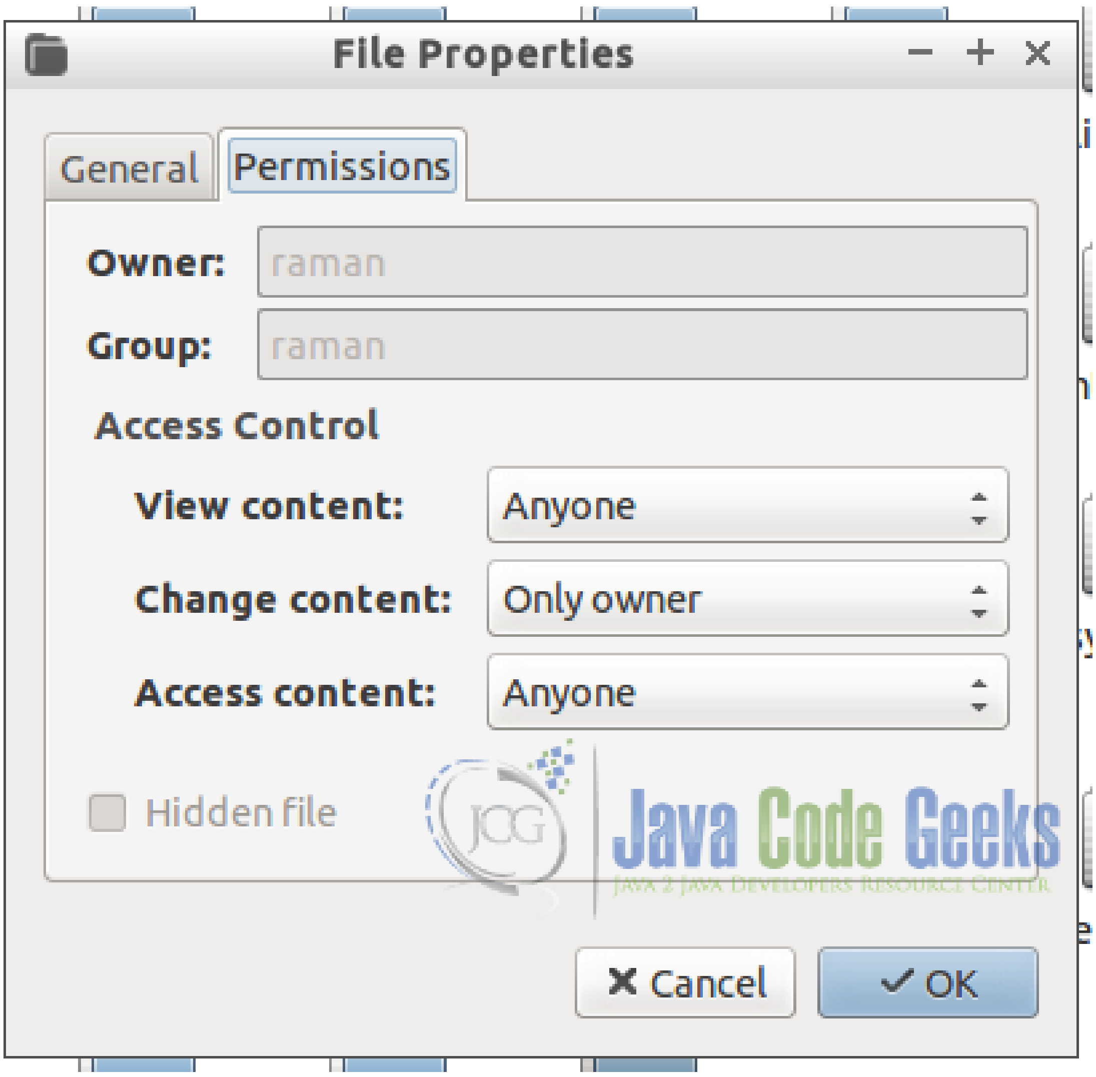
**Note:** There is a space between **:** and `zookeeper`. Here we are only mentioning the `raman` user as the owner of the directory and no user group(usergroup comes after **:**). So it will assign the default usergroup of the user to the directory `zookeeper`.



```
raman@raman-VirtualBox:/var/lib$ ls
AccountsService  dbus                ghostscript     logrotate       ntp          python       ubuntu-drivers-common   upower            whoopsie
alsa             dhcp                initramfs-tools machines        ntpdate      resolvconf   ubuntu-release-upgrader urandom           xfonts
apparmor         dictionaries-common initscripts     man-db          openbox      sgml-base    ucf                     ureadahead        xkb
apt              dkms                insserv         misc            os-prober    sudo         udisks2                 usb_modeswitch    xml-core
aspell           dpkg                lightdm         mlocate         pam          synaptic     update-manager          usbutils          zookeeper
belocs           emacsen-common      lightdm-data    NetworkManager  plymouth     systemd      update-notifier         VBoxGuestAdditions
bluetooth        gconf               locales         nssdb           polkit-1     ubiquity     update-rc.d             vim
raman@raman-VirtualBox:/var/lib$ sudo chown raman: zookeeper
raman@raman-VirtualBox:/var/lib$
```

Figure 10.7: Zookeeper user change command

To make sure that the owner is changed, go to the properties of the `/var/lib/zookeeper` directory and check the permissions. It should be assigned to user we set it in:


File Properties

General | Permissions

Owner: raman

Group: raman

Access Control

View content: Anyone

Change content: Only owner

Access content: Anyone

Hidden file

Cancel | OK

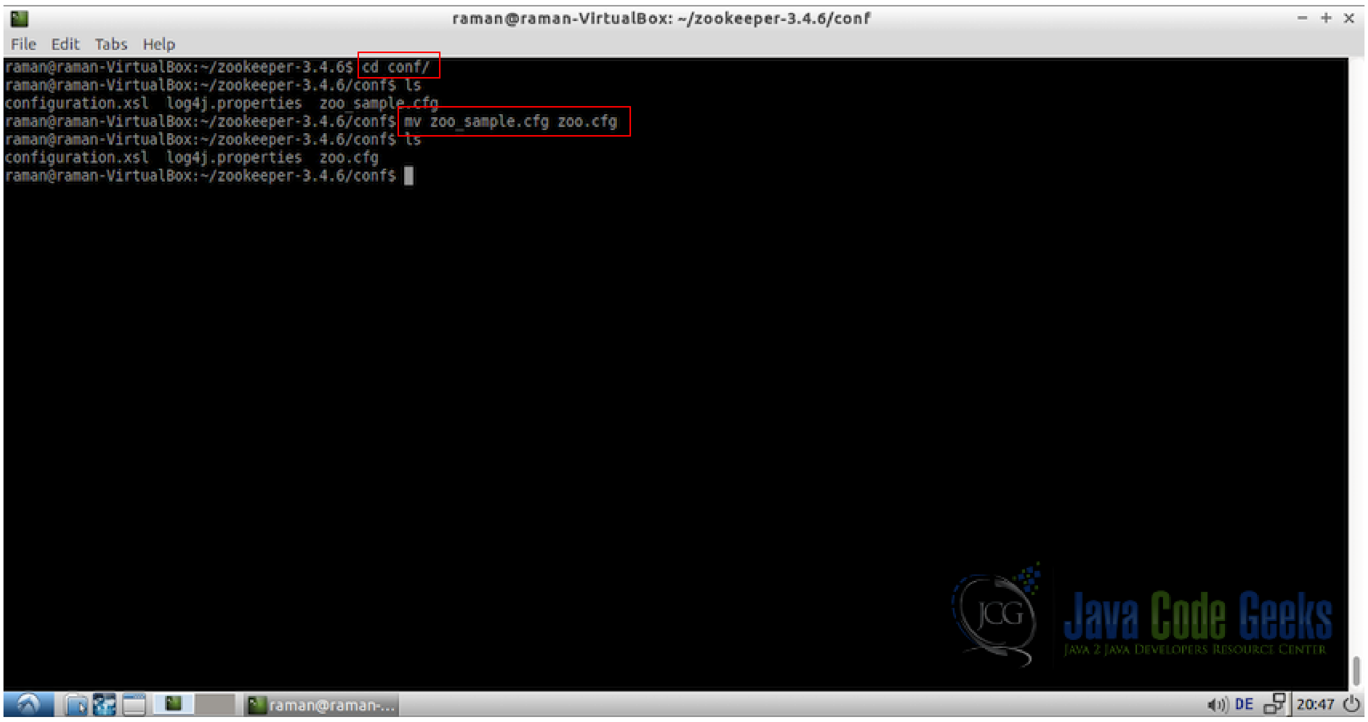
Figure 10.8: Zookeeper data folder properties

### 10.3.5 Configuration File

Now it is the time to make the required changes in the configurations of the Zookeeper server. It already contains the sample configuration file which we will use as the template. Sample configuration file is in the folder `zookeeper-3.4.6/conf/` and is named `zoo-sample.cfg`

First lets rename the file to `zoo.cfg`. Name of the file does not matter but there should be only one `.cfg` file present in the `conf` folder.

```
cd zookeeper-3.4.6/conf
mv zoo-sample.cfg zoo.cfg
```



```
raman@raman-VirtualBox:~/zookeeper-3.4.6$ cd conf/
raman@raman-VirtualBox:~/zookeeper-3.4.6/conf$ ls
configuration.xsl  log4j.properties  zoo_sample.cfg
raman@raman-VirtualBox:~/zookeeper-3.4.6/conf$ mv zoo_sample.cfg zoo.cfg
raman@raman-VirtualBox:~/zookeeper-3.4.6/conf$ ls
configuration.xsl  log4j.properties  zoo.cfg
raman@raman-VirtualBox:~/zookeeper-3.4.6/conf$
```

Figure 10.9: Renaming sample configuration file to the configuration file

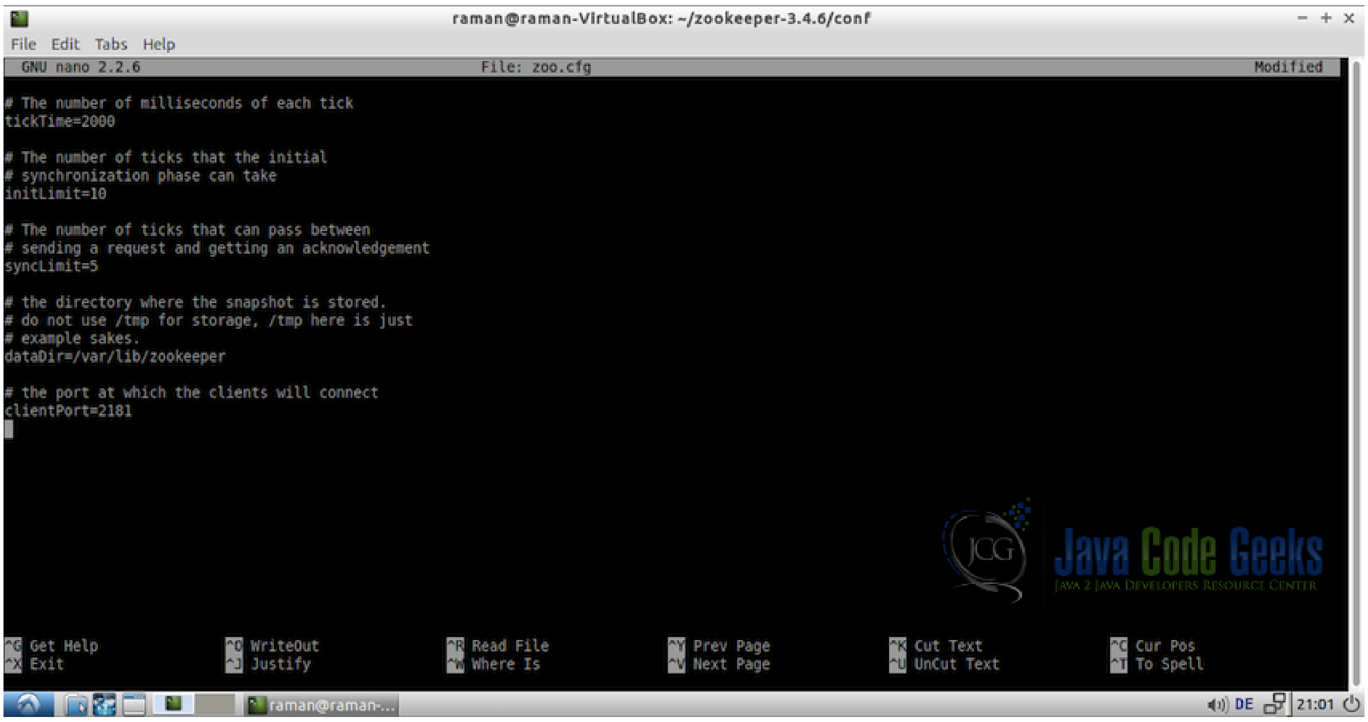
Now, let's edit this zoo.cfg file. In this example, we used the `nano` editor but you can use whichever editor you like to.

```
nano zoo.cfg
```

Make sure the file looks like what is in the screenshot below and contain the following settings:

```
tickTime = 2000
initLimit=10
syncLimit=5
dataDir=/var/lib/zookeeper
clientPort=2181
```

**Note:** `dataDir` should be set to the directory we created in the previous step i.e. `/var/lib/zookeeper`



```
raman@raman-VirtualBox: ~/zookeeper-3.4.6/conf
File  Edit  Tabs  Help
  GNU nano 2.2.6                  File: zoo.cfg                          Modified

# The number of milliseconds of each tick
tickTime=2000

# The number of ticks that the initial
# synchronization phase can take
initLimit=10

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5

# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/var/lib/zookeeper

# the port at which the clients will connect
clientPort=2181


^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure 10.10: zoo.cfg file

Let us have a short overview of what these configuration settings mean:

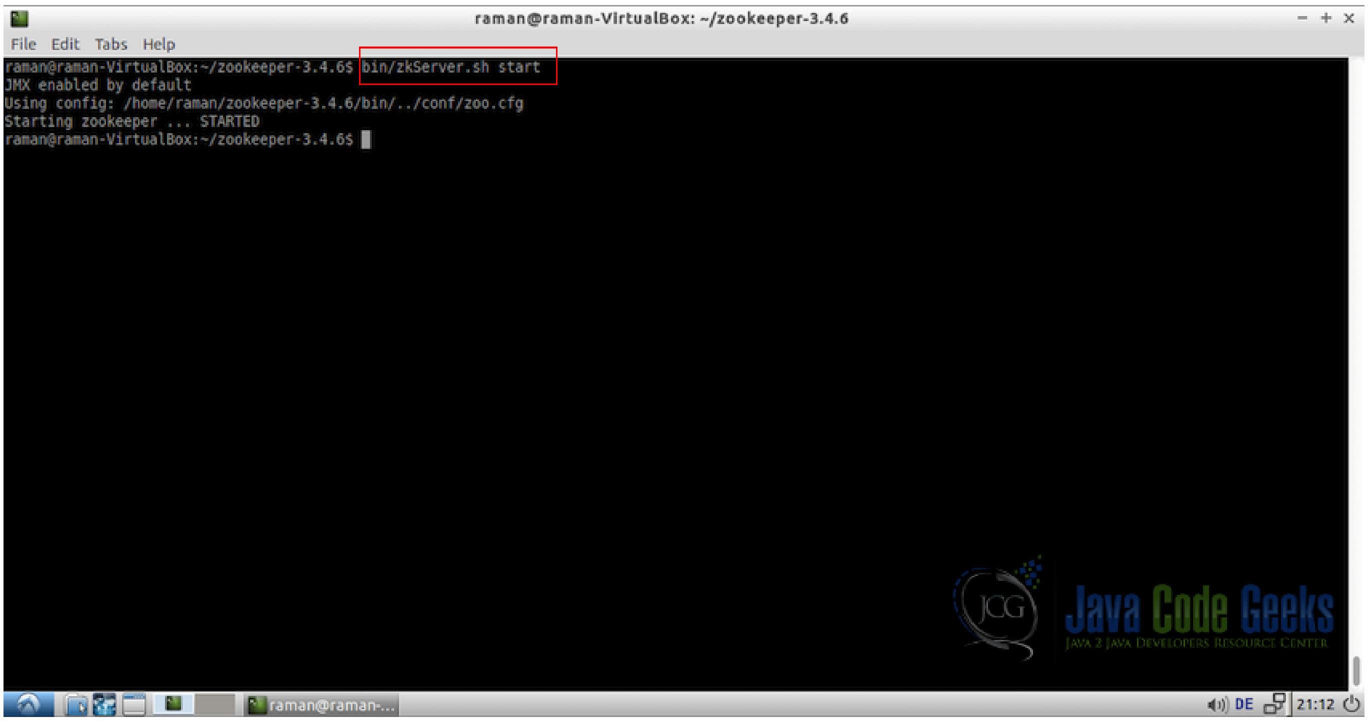
- tickTime: It is the time used by Zookeeper to do heartbeat with all the system nodes to check if all the nodes are alive and connected.
- initTime: The number of ticks that an initial synchronization phase can take.
- syncTime: The number of ticks that can pass between sending the request and getting an acknowledgement.
- dataDir: Directory to store in-memory database snapshots and the transaction logs by Zookeeper.
- clientPort: The port which will be used for the client connections.

### 10.3.6 Starting The Server

Now it is the time to start the Zookeeper server. Zookeeper comes with a script file to make it easy to start the server. The file is called `zkServer.sh`. So to start the server use the following code:

```
cd zookeeper-3.4.6/
bin/zkServer.sh start
```

It should display the console output similar to the following screenshot:



```
raman@raman-VirtualBox:~/zookeeper-3.4.6$ bin/zkServer.sh start
JMX enabled by default
Using config: /home/raman/zookeeper-3.4.6/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
raman@raman-VirtualBox:~/zookeeper-3.4.6$
```

Figure 10.11: Starting Zookeeper Server

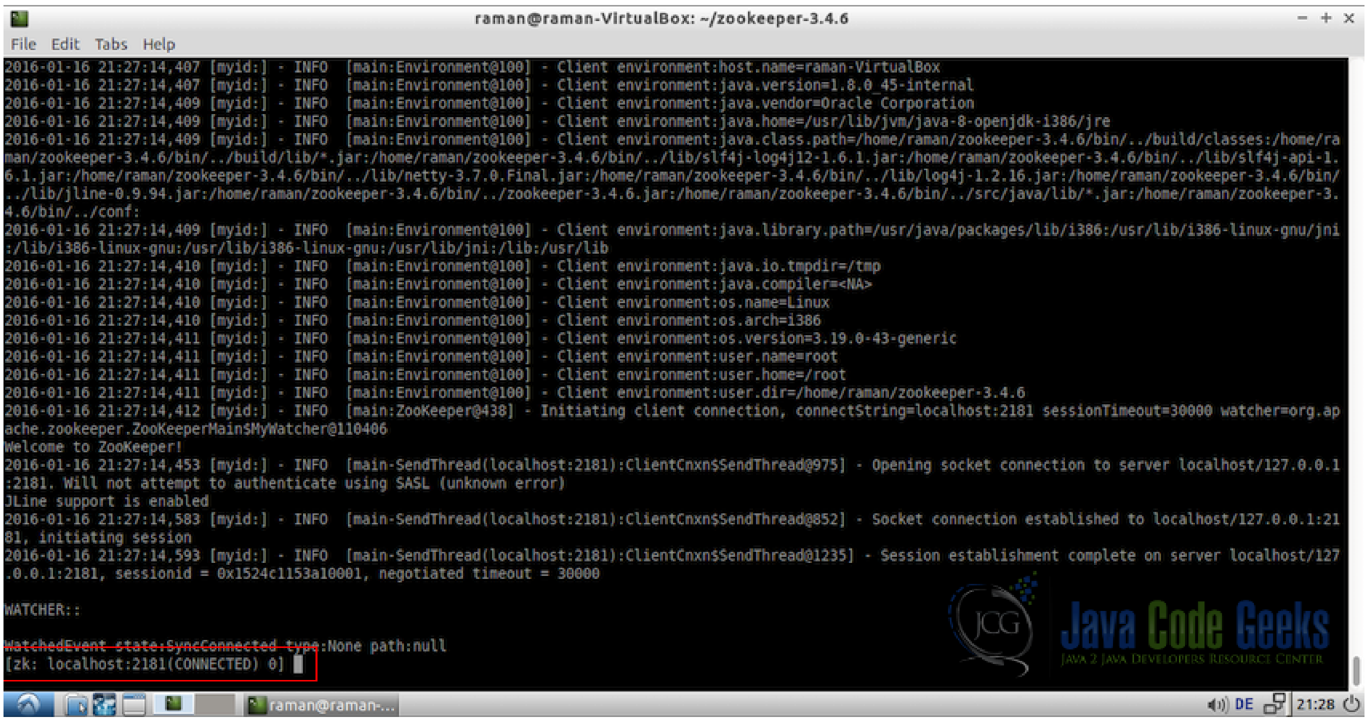
## 10.4 Zookeeper Server Basic Interaction

### 10.4.1 Starting The CLI

Once the Zookeeper server is running successfully, we can start the CLI(Command Line Interface) to interact with the server. Use the following command to do so:

```
cd zookeeper-3.4.6/
bin/zkCLi.sh -server
```

With this command, the console will go into the Zookeeper command line mode where we can use the Zookeeper specific commands to interact with the server.



```
2016-01-16 21:27:14,407 [myid:] - INFO  [main:Environment@100] - Client environment:host.name=raman-VirtualBox
2016-01-16 21:27:14,407 [myid:] - INFO  [main:Environment@100] - Client environment:java.version=1.8.0_45-internal
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=Oracle Corporation
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/usr/lib/jvm/java-8-openjdk-i386/jre
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.class.path=/home/raman/zookeeper-3.4.6/bin/../build/classes:/home/ra
man/zookeeper-3.4.6/bin/../build/lib/*.jar:/home/raman/zookeeper-3.4.6/bin/../lib/slf4j-log4j12-1.6.1.jar:/home/raman/zookeeper-3.4.6/bin/../lib/slf4j-api-1.
6.1.jar:/home/raman/zookeeper-3.4.6/bin/../lib/netty-3.7.0.Final.jar:/home/raman/zookeeper-3.4.6/bin/../lib/log4j-1.2.16.jar:/home/raman/zookeeper-3.4.6/bin/
../lib/jline-0.9.94.jar:/home/raman/zookeeper-3.4.6/bin/../zookeeper-3.4.6.jar:/home/raman/zookeeper-3.4.6/bin/../src/java/lib/*.jar:/home/raman/zookeeper-3.
4.6/bin/../conf:
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.library.path=/usr/java/packages/lib/i386:/usr/lib/i386-linux-gnu/jni
:/lib/i386-linux-gnu:/usr/lib/i386-linux-gnu:/usr/lib/jni:/lib:/usr/lib
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:java.io.tmpdir=/tmp
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:java.compiler=<NA>
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:os.name=Linux
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:os.arch=i386
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:os.version=3.19.0-43-generic
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.name=root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.home=/root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.dir=/home/raman/zookeeper-3.4.6
2016-01-16 21:27:14,412 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.ap
ache.zookeeper.ZooKeeperMain$MyWatcher@110406
Welcome to ZooKeeper!
2016-01-16 21:27:14,453 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@975] - Opening socket connection to server localhost/127.0.0.1
:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2016-01-16 21:27:14,583 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@852] - Socket connection established to localhost/127.0.0.1:21
81, initiating session
2016-01-16 21:27:14,593 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server localhost/127
.0.0.1:2181, sessionid = 0x1524c1153a10001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0]
```

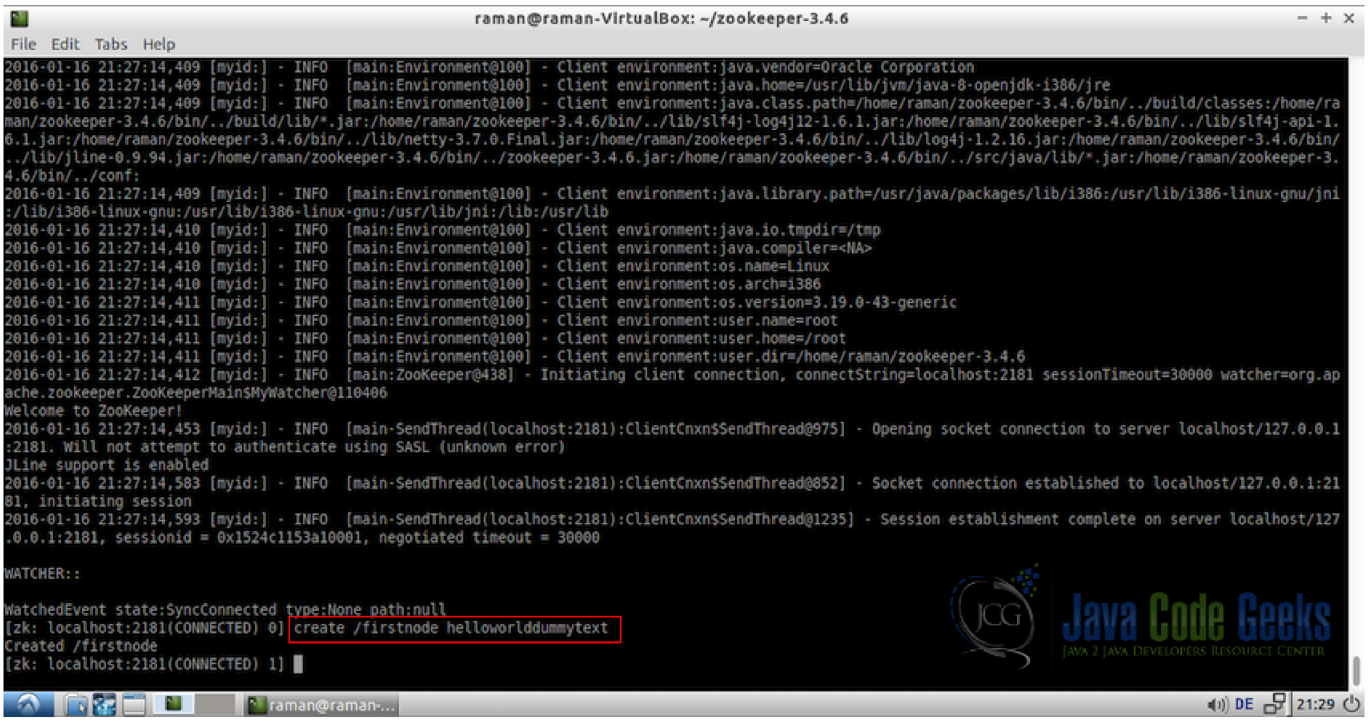
Figure 10.12: Zookeeper Command Line Interface

### 10.4.2 Creating The First Znode

Let us start by creating a new node. Following is the Zookeeper command to create a new znode with dummy data.

```
create /firstnode helloworlddummytext
```

Here `firstnode` is the name of the znode which will be created on the root path as indicated by / and `helloworlddummy text` is the dummy text stored in the znode memory.



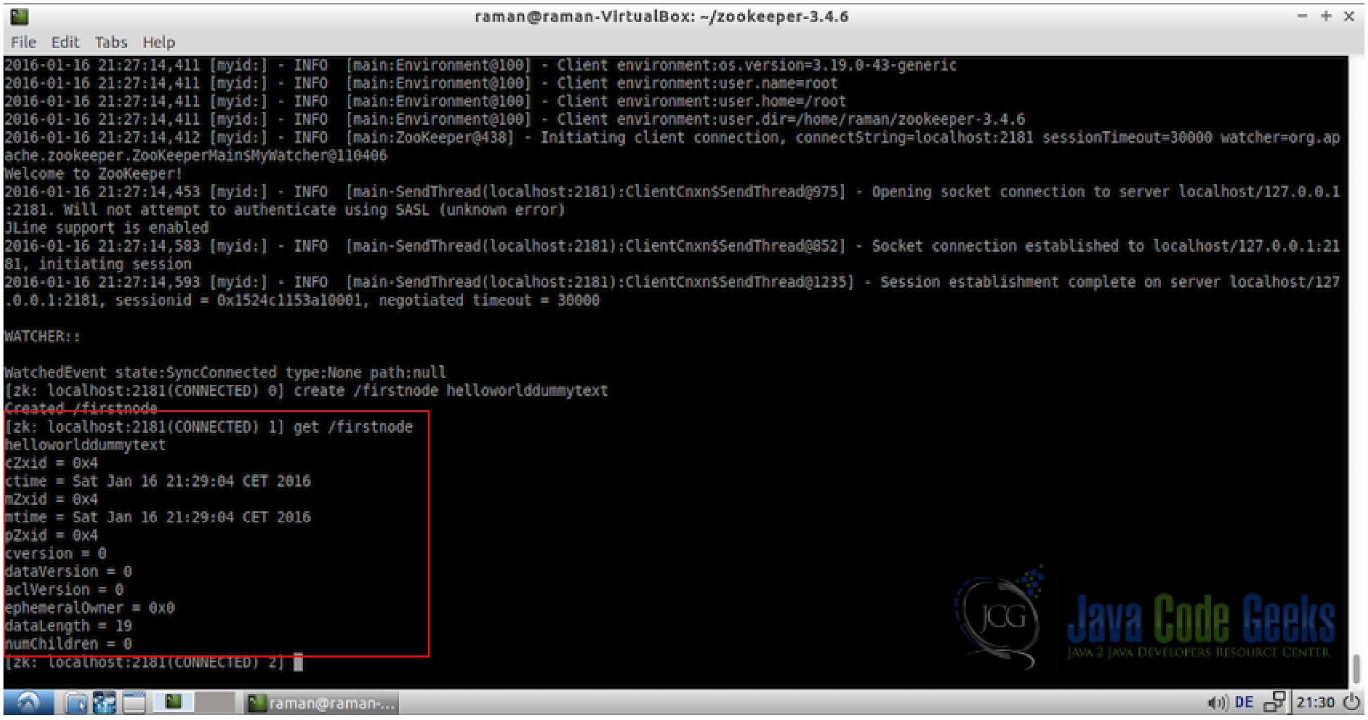
```
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=Oracle Corporation
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/usr/lib/jvm/java-8-openjdk-i386/jre
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.class.path=/home/raman/zookeeper-3.4.6/bin/../build/classes:/home/ra
man/zookeeper-3.4.6/bin/../build/lib/*.jar:/home/raman/zookeeper-3.4.6/bin/../lib/slf4j-log4j12-1.6.1.jar:/home/raman/zookeeper-3.4.6/bin/../lib/slf4j-api-1.
6.1.jar:/home/raman/zookeeper-3.4.6/bin/../lib/netty-3.7.0.Final.jar:/home/raman/zookeeper-3.4.6/bin/../lib/log4j-1.2.16.jar:/home/raman/zookeeper-3.4.6/bin/
../lib/jline-0.9.94.jar:/home/raman/zookeeper-3.4.6/bin/../zookeeper-3.4.6.jar:/home/raman/zookeeper-3.4.6/bin/../src/java/lib/*.jar:/home/raman/zookeeper-3.
4.6/bin/../conf:
2016-01-16 21:27:14,409 [myid:] - INFO  [main:Environment@100] - Client environment:java.library.path=/usr/java/packages/lib/i386:/usr/lib/i386-linux-gnu/jni
:/lib/i386-linux-gnu:/usr/lib/i386-linux-gnu:/usr/lib/jni:/lib:/usr/lib
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:java.io.tmpdir=/tmp
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:java.compiler=<NA>
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:os.name=Linux
2016-01-16 21:27:14,410 [myid:] - INFO  [main:Environment@100] - Client environment:os.arch=i386
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:os.version=3.19.0-43-generic
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.name=root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.home=/root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.dir=/home/raman/zookeeper-3.4.6
2016-01-16 21:27:14,412 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.ap
ache.zookeeper.ZooKeeperMain$MyWatcher@110406
Welcome to ZooKeeper!
2016-01-16 21:27:14,453 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@975] - Opening socket connection to server localhost/127.0.0.1
:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2016-01-16 21:27:14,583 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@852] - Socket connection established to localhost/127.0.0.1:21
81, initiating session
2016-01-16 21:27:14,593 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server localhost/127
.0.0.1:2181, sessionid = 0x1524c1153a10001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] create /firstnode helloworlddummytext
Created /firstnode
[zk: localhost:2181(CONNECTED) 1]
```

Figure 10.13: Create znode in Zookeeper

### 10.4.3 Retrieving Data From The First Znode

Similar to how we created a new znode, we can get back the details and data of the znode using the CLI(Command Line Interface). Following is the command for getting the data from znode.

```
get /firstnode
```



```
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:os.version=3.19.0-43-generic
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.name=root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.home=/root
2016-01-16 21:27:14,411 [myid:] - INFO  [main:Environment@100] - Client environment:user.dir=/home/raman/zookeeper-3.4.6
2016-01-16 21:27:14,412 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.ap
ache.zookeeper.ZooKeeperMain$MyWatcher@110406
Welcome to ZooKeeper!
2016-01-16 21:27:14,453 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@975] - Opening socket connection to server localhost/127.0.0.1
:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2016-01-16 21:27:14,583 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@852] - Socket connection established to localhost/127.0.0.1:21
81, initiating session
2016-01-16 21:27:14,593 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server localhost/127
.0.0.1:2181, sessionid = 0x1524c1153a10001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] create /firstnode helloworlddummytext
Created /firstnode
[zk: localhost:2181(CONNECTED) 1] get /firstnode
helloworlddummytext
cZxid = 0x4
ctime = Sat Jan 16 21:29:04 CET 2016
mZxid = 0x4
mtime = Sat Jan 16 21:29:04 CET 2016
pZxid = 0x4
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 19
numChildren = 0
[zk: localhost:2181(CONNECTED) 2]
```

Figure 10.14: Getting data from znode in Zookeeper

If you notice in the screenshot, along with the data we stored in the znode while creating, the server also returned some metadata related to this particular znode.

Some of the important fields in the metadata are:

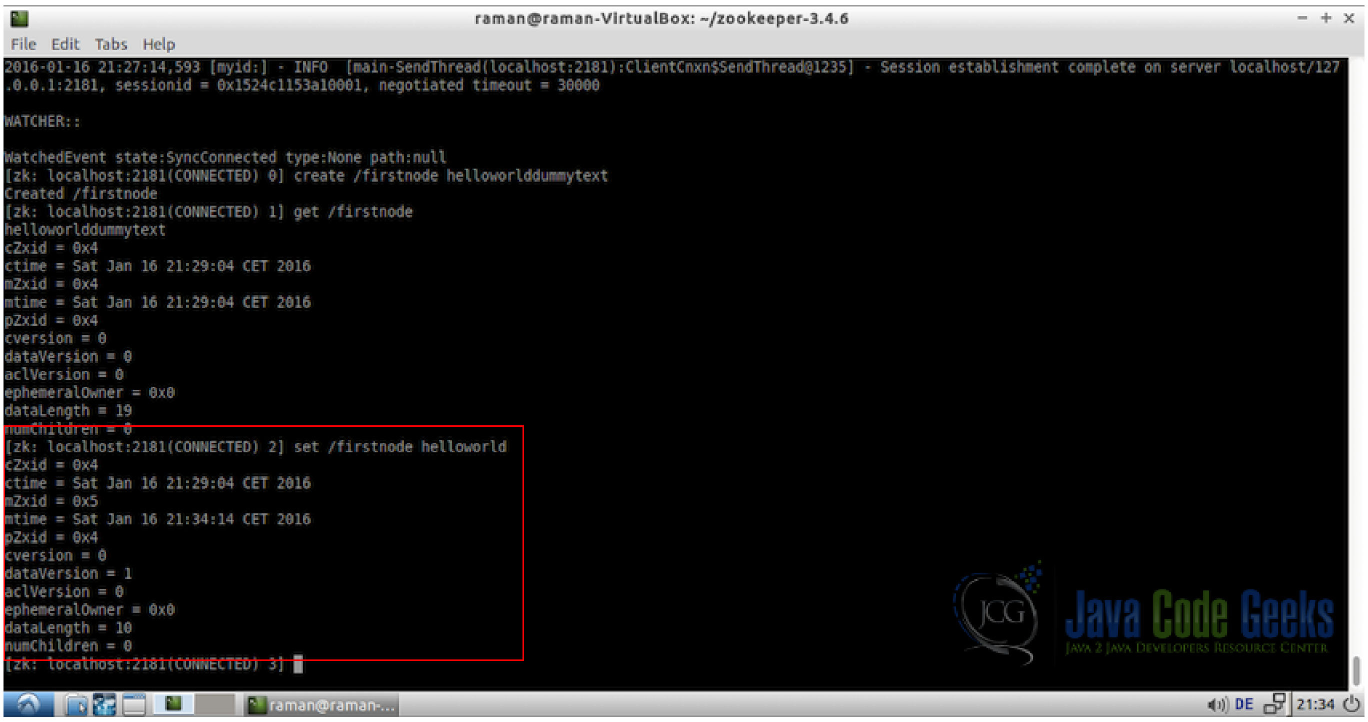
- **ctime:** Time when this znode was created.
- **mtime:** Last modified time.
- **dataVersion:** Version of the data which changes everytime data is modified
- **datalength:** Length of the data stored in the znode. In this case data is `helloworlddummydata` and the length is 19.
- **numchildren:** Number of children of this aprticualr znode.

### 10.4.4 Modifying Data in Znode

If we want to modify data in a particular node Zookeeper provides a command for that also. Following is how to modify the data in an existing znode:

```
set /firstnode helloworld
```

Where `firstnode` is the existing znode and `helloworld` is the new data which need to be written in the znode. Old data will be removed when new data is set.



```
2016-01-16 21:27:14,593 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server localhost/127
.0.0.1:2181, sessionid = 0x1524c1153a10001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] create /firstnode helloworlddummytext
Created /firstnode
[zk: localhost:2181(CONNECTED) 1] get /firstnode
helloworlddummytext
cZxid = 0x4
ctime = Sat Jan 16 21:29:04 CET 2016
mZxid = 0x4
mtime = Sat Jan 16 21:29:04 CET 2016
pZxid = 0x4
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 19
numChildren = 0
[zk: localhost:2181(CONNECTED) 2] set /firstnode helloworld
cZxid = 0x4
ctime = Sat Jan 16 21:29:04 CET 2016
mZxid = 0x5
mtime = Sat Jan 16 21:34:14 CET 2016
pZxid = 0x4
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10
numChildren = 0
[zk: localhost:2181(CONNECTED) 3]
```

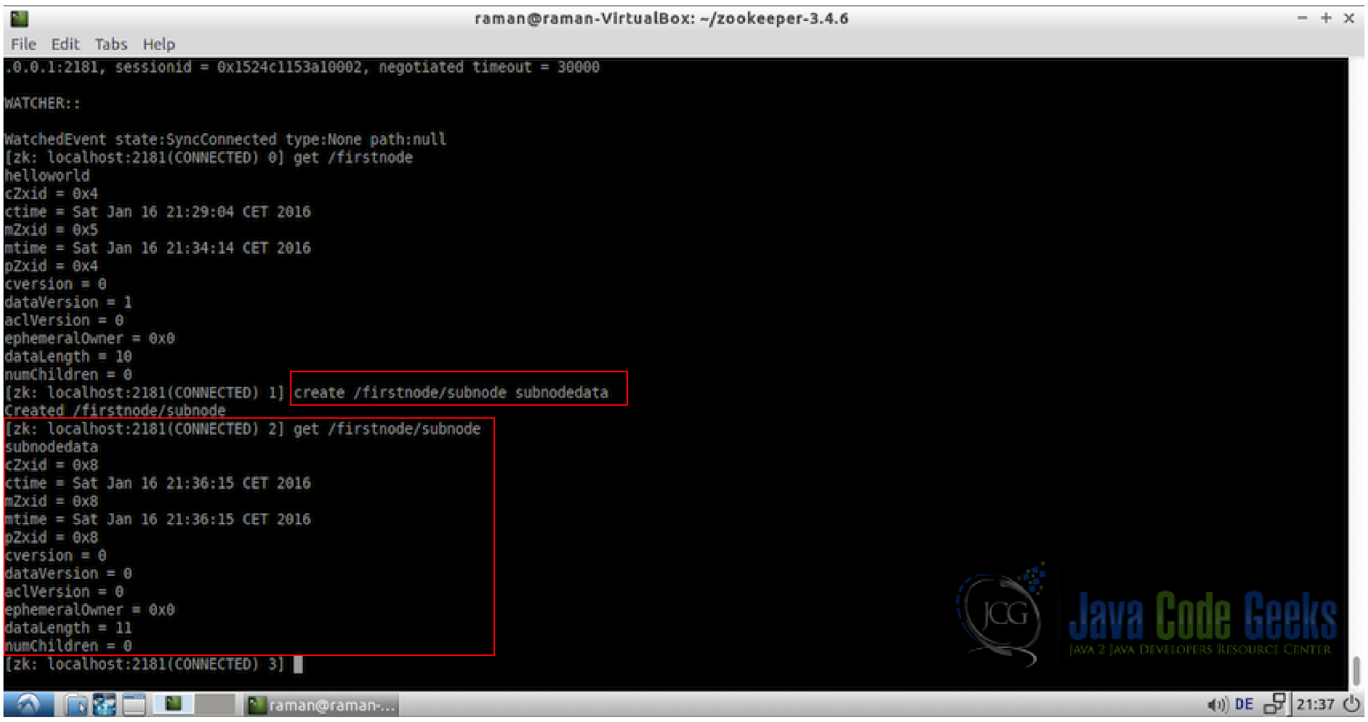
Figure 10.15: Modifying data in an existing znode

If you notice in the screenshot above `datalength`, `mtime`, and `dataversion` is also updated when a new value is set.

### 10.4.5 Creating A Subnode

Creating a subnode in an existing node is as easy as creating a new node. We just need to pass the full path for the new subnode.

```
create /firstnode/subnode subnodedata
get /firstnode/subnode
```



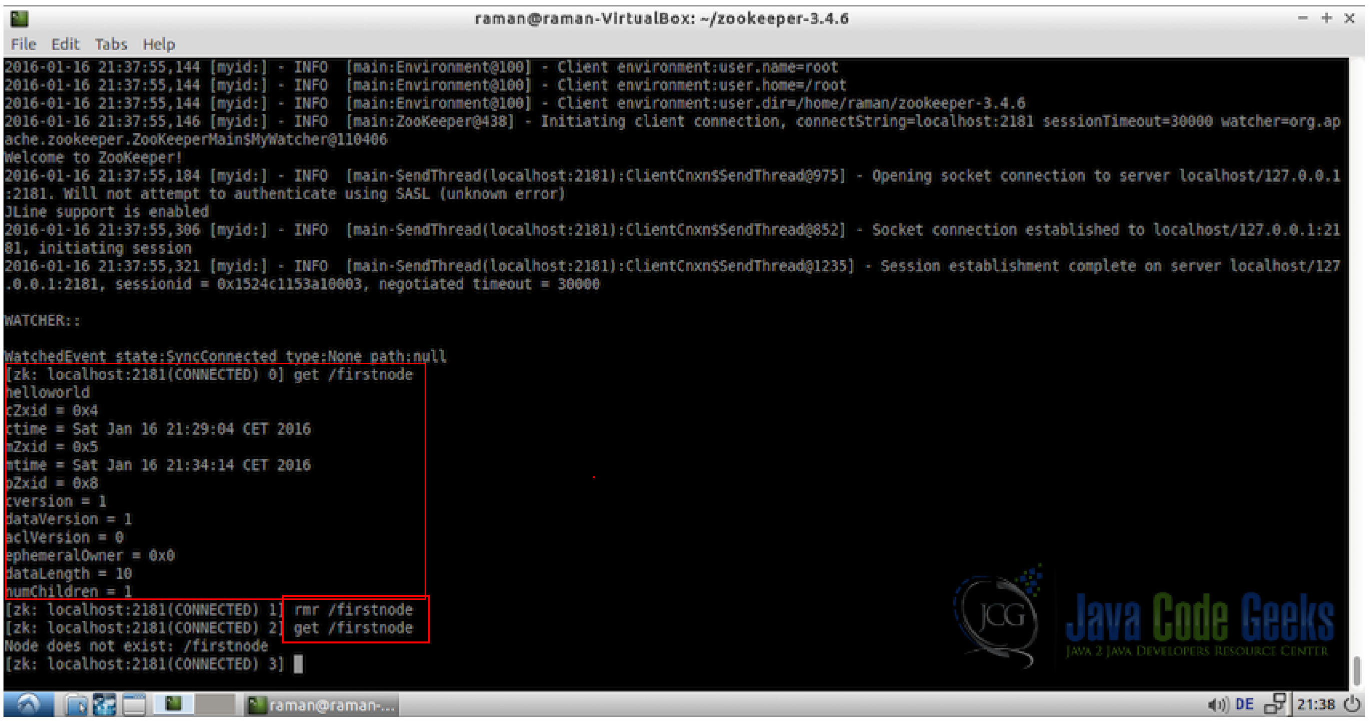
```
.0.0.1:2181, sessionid = 0x1524c1153a10002, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] get /firstnode
helloworld
cZxid = 0x4
ctime = Sat Jan 16 21:29:04 CET 2016
mZxid = 0x5
mtime = Sat Jan 16 21:34:14 CET 2016
pZxid = 0x4
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10
numChildren = 0
[zk: localhost:2181(CONNECTED) 1] create /firstnode/subnode subnodedata
Created /firstnode/subnode
[zk: localhost:2181(CONNECTED) 2] get /firstnode/subnode
subnodedata
cZxid = 0x8
ctime = Sat Jan 16 21:36:15 CET 2016
mZxid = 0x8
mtime = Sat Jan 16 21:36:15 CET 2016
pZxid = 0x8
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 11
numChildren = 0
[zk: localhost:2181(CONNECTED) 3]
```

Figure 10.16: Creating a subnode for an existing node

### 10.4.6 Removing A Node

Removing a node is quite easy using `rmr` command in the Zookeeper CLI. Removing a node also removing all its subnodes. Following is the code to remove `firstnode` which we created for this example:

```
rmr /firstnode
```



```
2016-01-16 21:37:55,144 [myid:] - INFO  [main:Environment@100] - Client environment:user.name=root
2016-01-16 21:37:55,144 [myid:] - INFO  [main:Environment@100] - Client environment:user.home=/root
2016-01-16 21:37:55,144 [myid:] - INFO  [main:Environment@100] - Client environment:user.dir=/home/raman/zookeeper-3.4.6
2016-01-16 21:37:55,146 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.ap
ache.zookeeper.ZooKeeperMain$MyWatcher@110406
Welcome to ZooKeeper!
2016-01-16 21:37:55,184 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@975] - Opening socket connection to server localhost/127.0.0.1
:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2016-01-16 21:37:55,306 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@852] - Socket connection established to localhost/127.0.0.1:21
81, initiating session
2016-01-16 21:37:55,321 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server localhost/127
.0.0.1:2181, sessionid = 0x1524c1153a10003, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] get /firstnode
helloworld
cZxid = 0x4
ctime = Sat Jan 16 21:29:04 CET 2016
mZxid = 0x5
mtime = Sat Jan 16 21:34:14 CET 2016
pZxid = 0x8
cversion = 1
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10
numChildren = 1
[zk: localhost:2181(CONNECTED) 1] rmr /firstnode
[zk: localhost:2181(CONNECTED) 2] get /firstnode
Node does not exist: /firstnode
[zk: localhost:2181(CONNECTED) 3]
```

Figure 10.17: Removing a node from Zookeeper

## 10.5 Conclusion

This brings us to the conclusion of this introductory example for Apache Zookeeper. In this example, we started with the introduction and the general architecture of Zookeeper followed by learning how to setup Zookeeper in a single machine. We also saw that using Zookeeper CLI to interface with the Zookeeper Service is also quite easy and command are present for all the basic interactions.

## 10.6 Download

**Download** You can download the configuration file `zoo.cfg` used in this example here: **Zookeeper Configuration**