https://towardsdatascience.com/installing-hadoop-3-2-1-single-node-cluster-on-windows-10-ac258dd48aef

towardsdatascience.com

Installing Hadoop 3.2.1 Single node cluster on Windows 10

Hadi Fadlallah
9-11 minutes

While working on a project two years ago, I wrote a step-by-step guide to install Hadoop 3.1.0 on Ubuntu 16.04 operating system. Since we are currently working on a new project where we need to install a Hadoop cluster on Windows 10, I decided to write a guide for this process.

This article is a part of a series that we are publishing on TowardsDataScience.com that aims to illustrate how to install Big Data technologies on Windows operating system.

Other published articles in this series:

- Installing Apache Pig 0.17.0 on Windows 10
- Installing Apache Hive 3.1.2 on Windows 10

1. Prerequisites

First, we need to make sure that the following prerequisites are installed:

1. Java 8 runtime environment (JRE): Hadoop 3 requires a Java 8 installation. I prefer using the offline installer.

2. Java 8 development Kit (JDK)

3. To unzip downloaded Hadoop binaries, we should install 7zip.

4. I will create a folder “E:\hadoop-env” on my local machine to store downloaded files.

2. Download Hadoop binaries

The first step is to download Hadoop binaries from the official website. The binary package size is about 342 MB.

Figure 1 — Hadoop binaries download link

After finishing the file download, we should unpack the package using 7zip int two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:

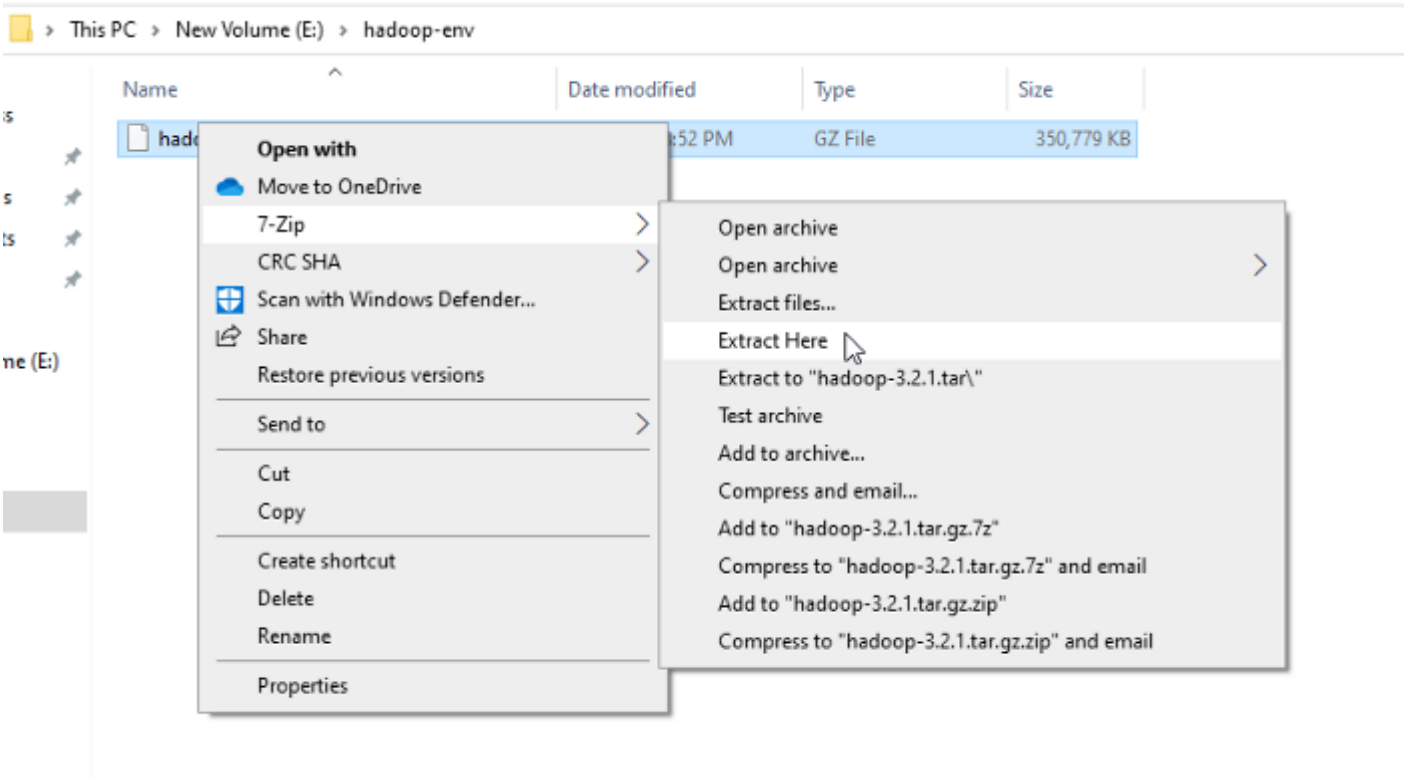


Figure 2 — Extracting hadoop-3.2.1.tar.gz package using 7zip

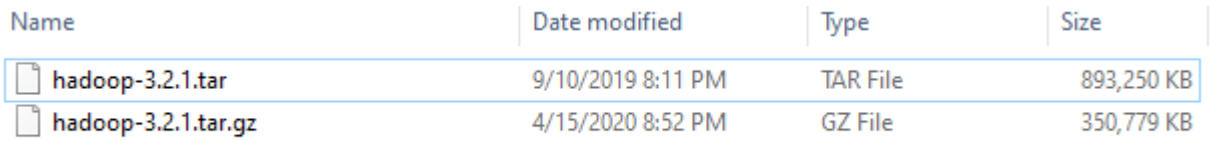


Figure 3 — Extracted hadoop-3.2.1.tar file

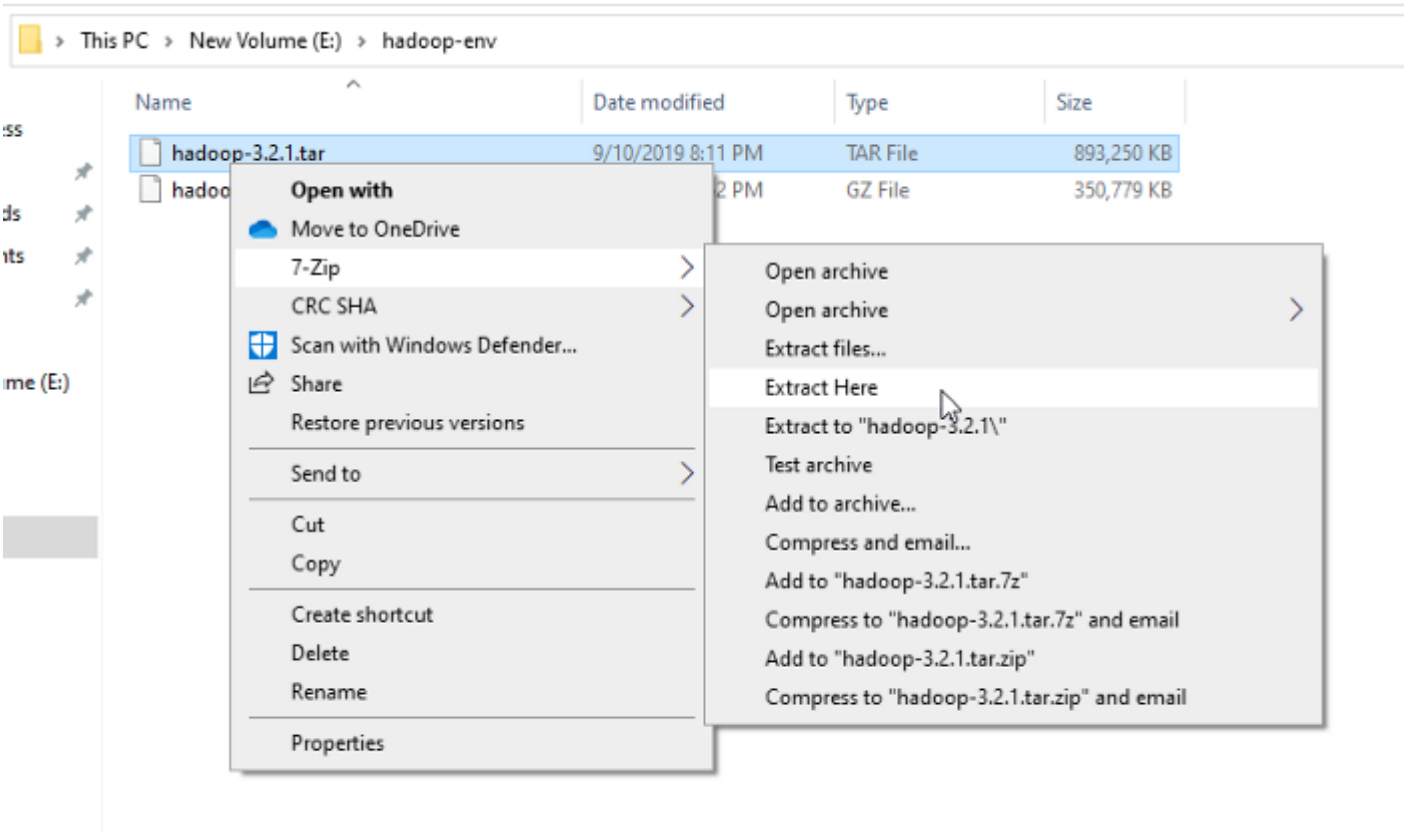


Figure 4 — Extracting the hadoop-3.2.1.tar file

The tar file extraction may take some minutes to finish. In the end, you may see some warnings about symbolic link creation. Just ignore these warnings since they are not related to windows.

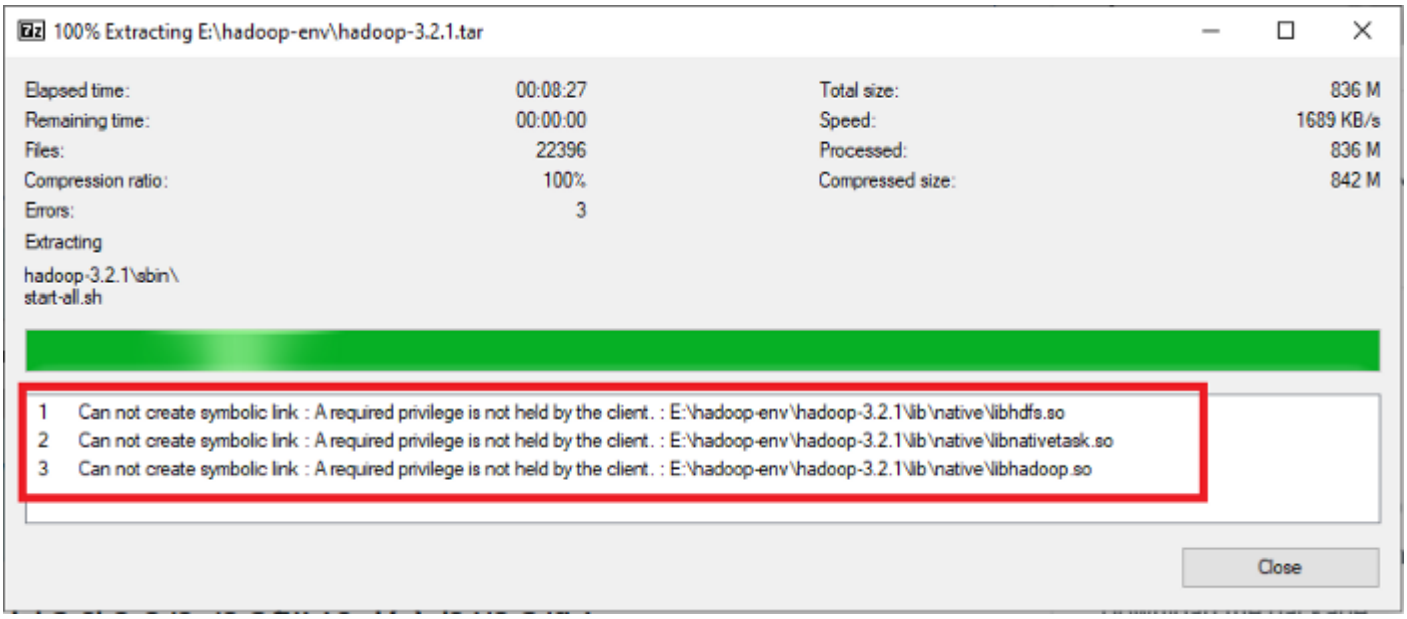
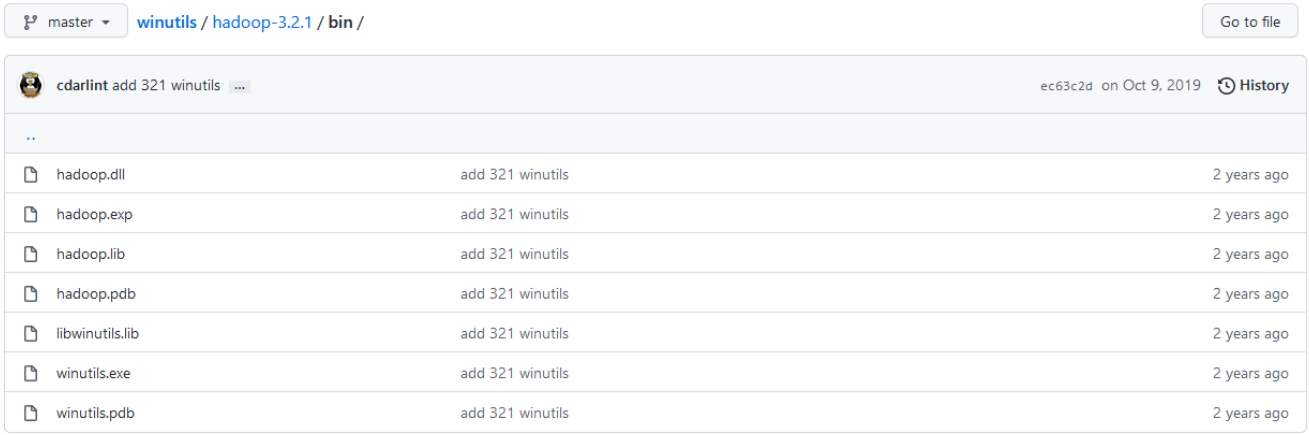


Figure 5 — Symbolic link warnings

After unpacking the package, we should add the Hadoop native IO libraries, which can be found in the following GitHub repository: https://github.com/cdarlint/winutils.

Since we are installing Hadoop 3.2.1, we should download the files located in https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.1/bin and copy them into the “hadoop-3.2.1\bin” directory.



3. Setting up environment variables

After installing Hadoop and its prerequisites, we should configure the environment variables to define Hadoop and Java default paths.

To edit environment variables, go to Control Panel > System and Security > System (or right-click > properties on My Computer icon) and click on the “Advanced system settings” link.

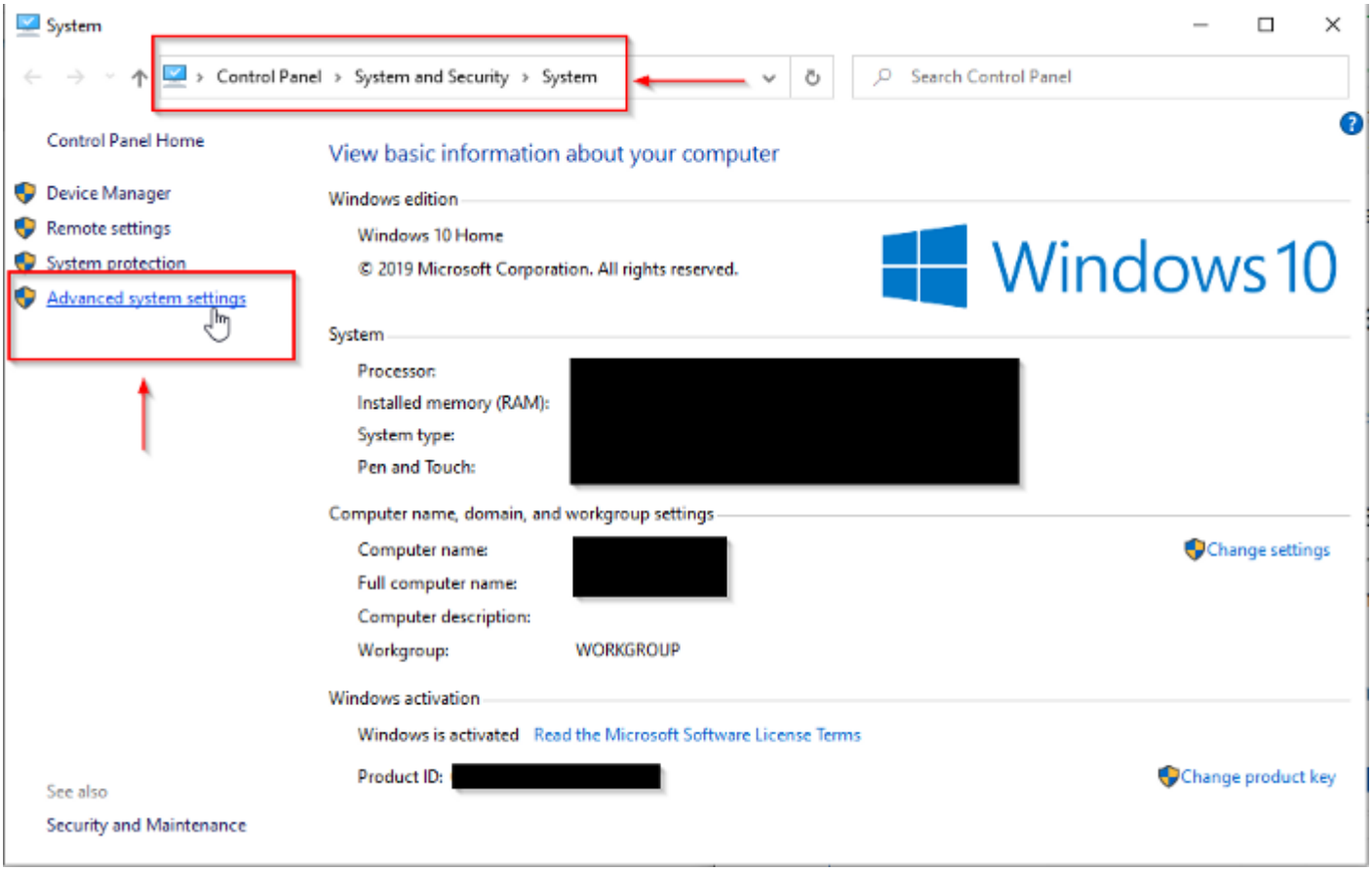


Figure 6 — Opening advanced system settings

When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.

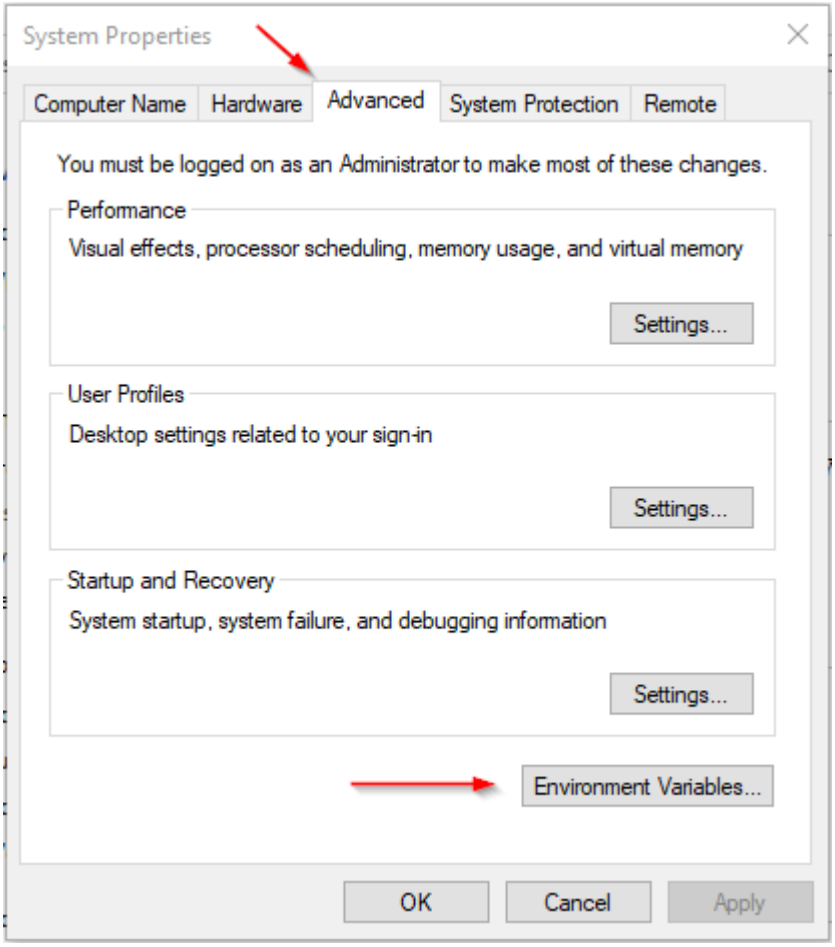


Figure 7 — Advanced system settings dialog

In the “Environment Variables” dialog, press the “New” button to add a new variable.

> Note: In this guide, we will add user variables since we are configuring Hadoop for a single user. If you are looking to configure Hadoop for multiple users, you can define System variables instead.

There are two variables to define:

1. JAVA_HOME: JDK installation folder path

2. HADOOP_HOME: Hadoop installation folder path

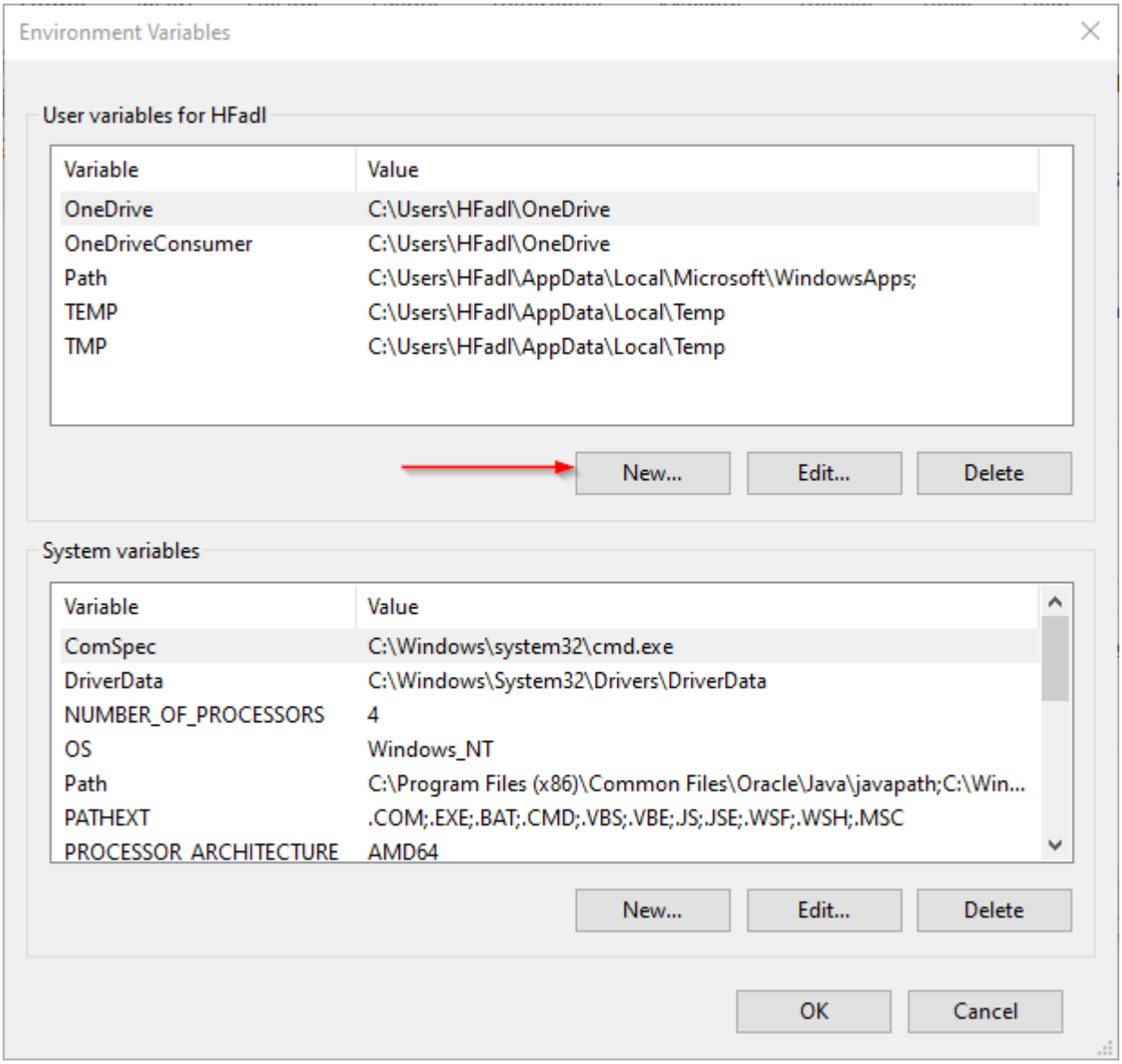


Figure 8 — Adding JAVA_HOME variable

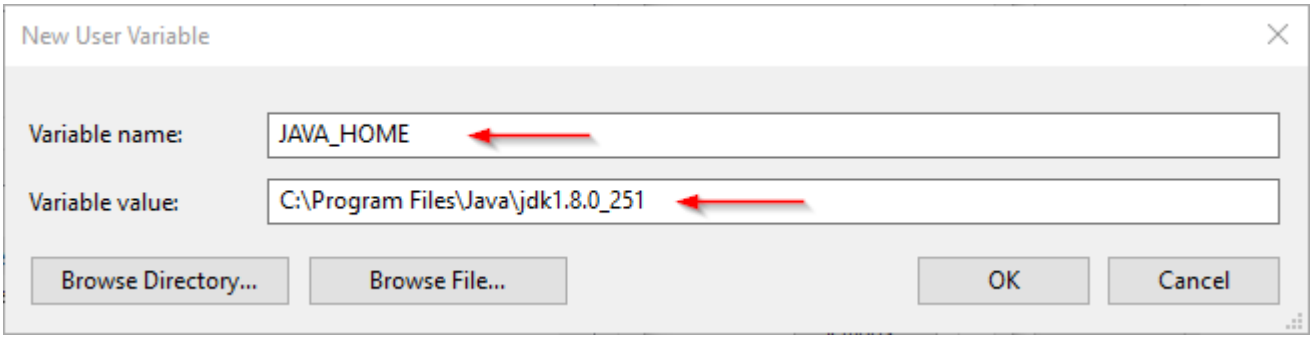


Figure 9 — Adding HADOOP_HOME variable

Now, we should edit the PATH variable to add the Java and Hadoop binaries paths as shown in the following screenshots.

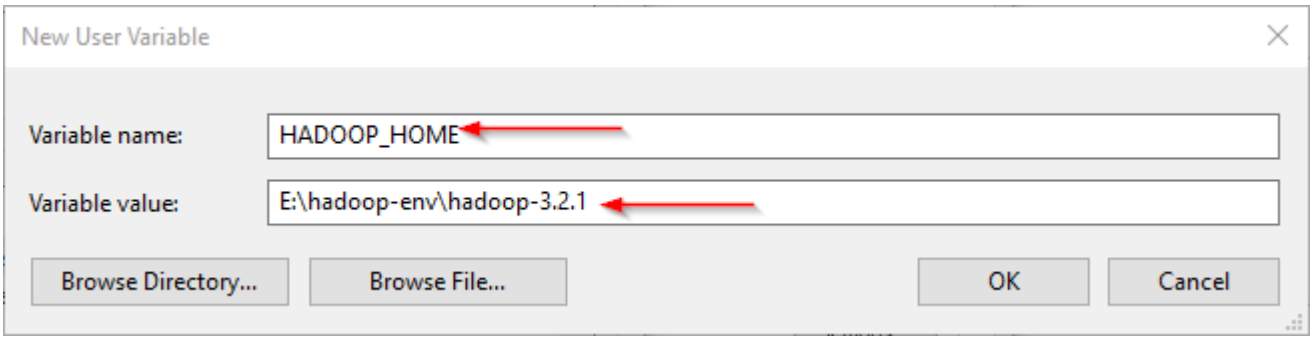


Figure 10 — Editing the PATH variable

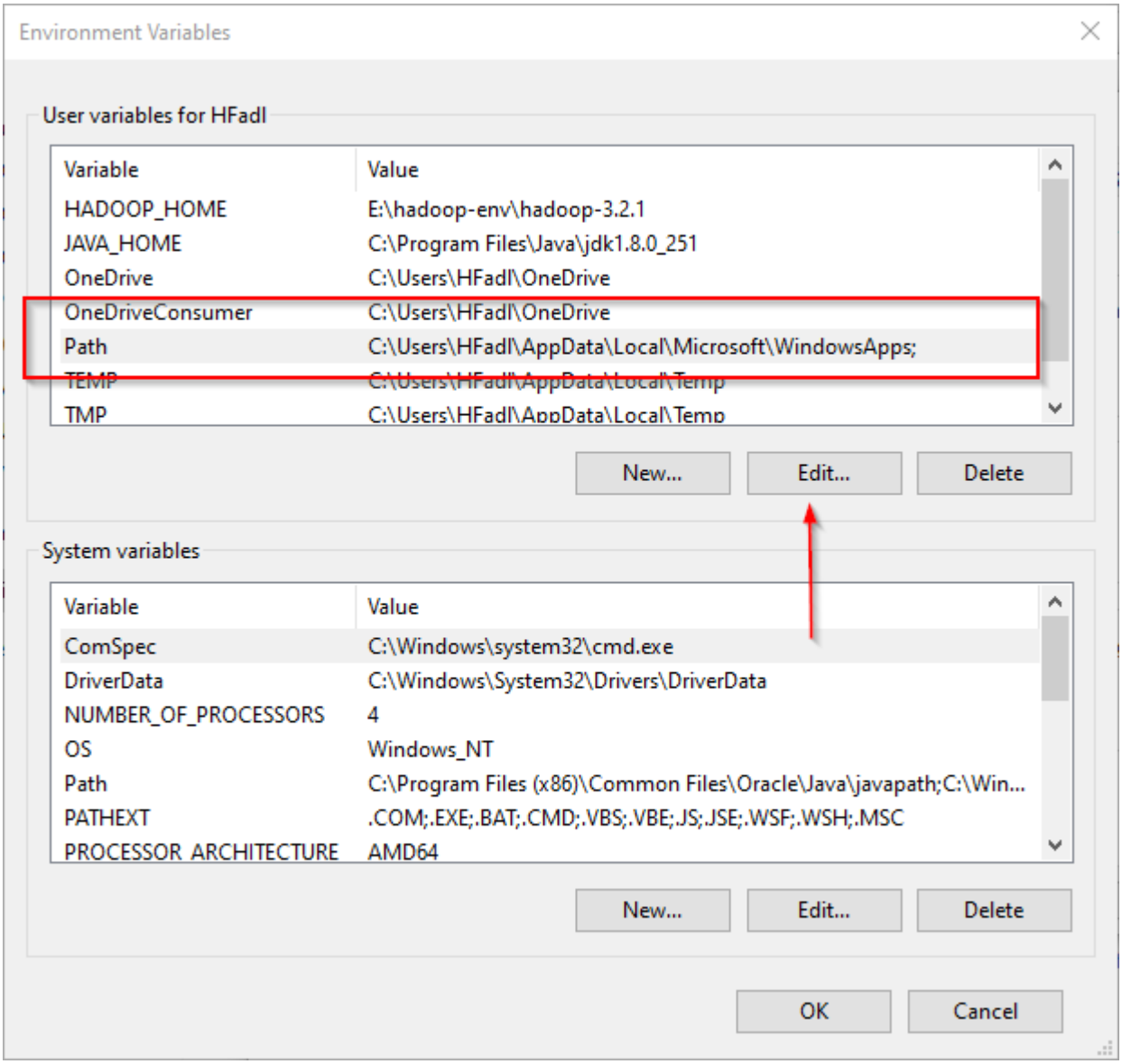


Figure 11 — Editing PATH variable

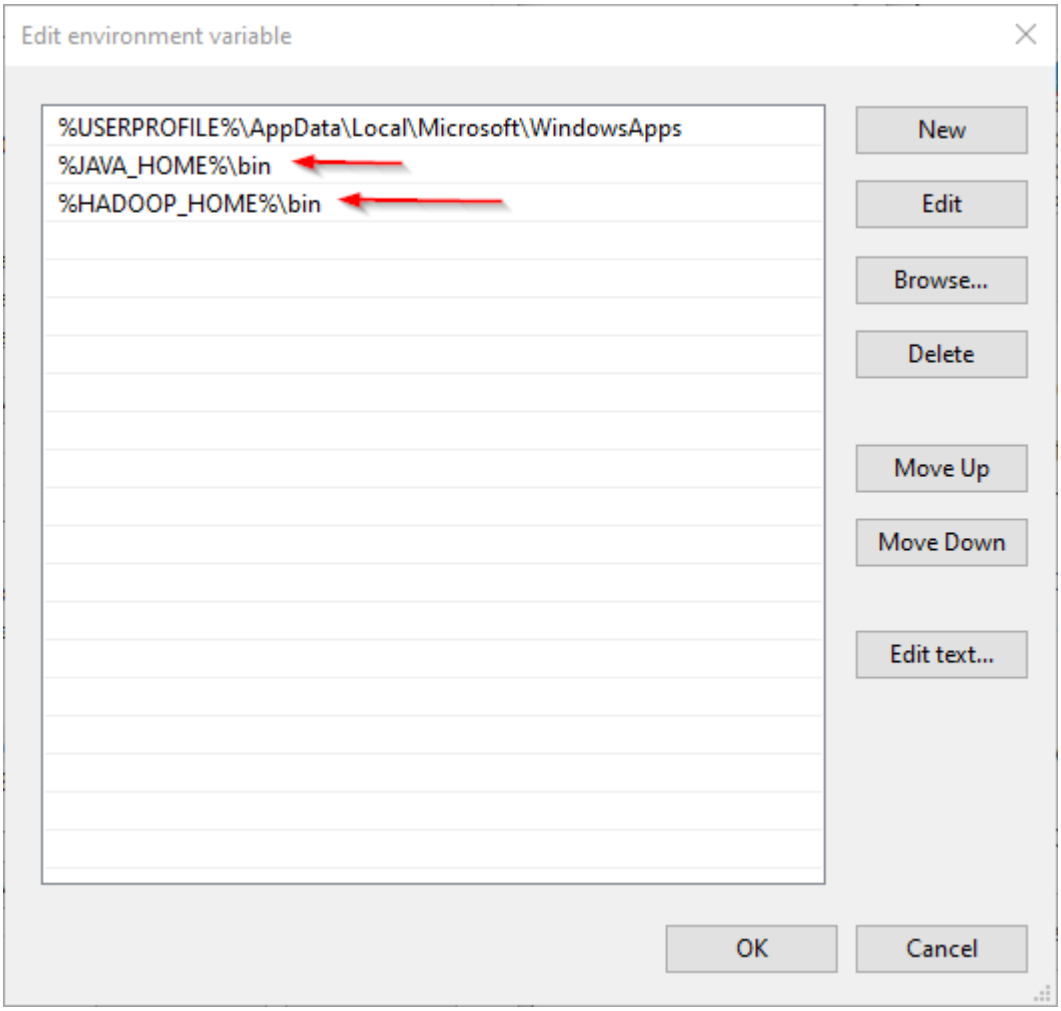


Figure 12— Adding new paths to the PATH variable

3.1. JAVA_HOME is incorrectly set error

Now, let's open PowerShell and try to run the following command:

```
hadoop -version
```

In this example, since the JAVA_HOME path contains spaces, I received the following error:

```
JAVA_HOME is incorrectly set
```

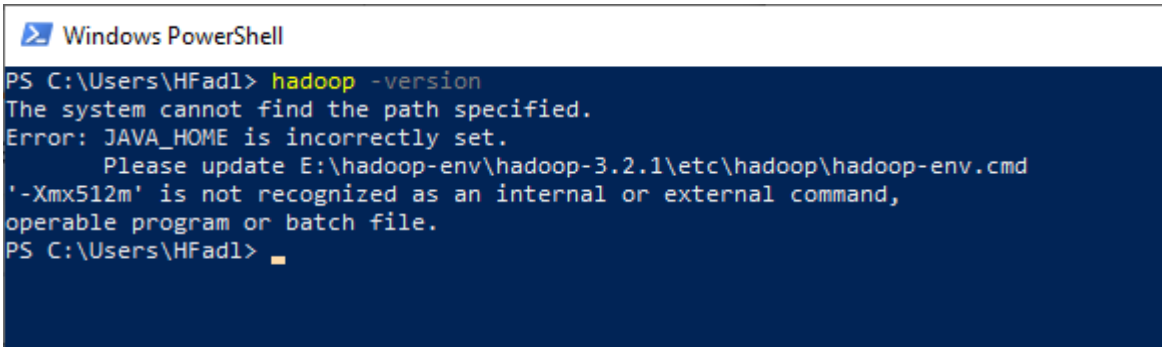


Figure 13 — JAVA_HOME error

To solve this issue, we should use the windows 8.3 path instead. As an example:

- Use “Progra~1” instead of “Program Files”
- Use “Progra~2” instead of “Program Files(x86)”

After replacing “Program Files” with “Progra~1”, we closed and reopened PowerShell and tried the same command. As shown in the screenshot below, it runs without errors.

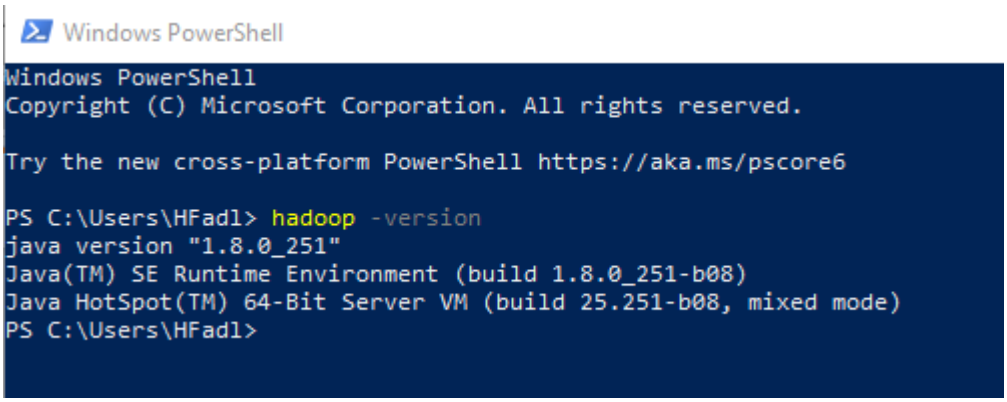


Figure 14 — hadoop -version command executed successfully

4. Configuring Hadoop cluster

There are four files we should alter to configure Hadoop cluster:

1. %HADOOP_HOME%\etc\hadoop\hdfs-site.xml
2. %HADOOP_HOME%\etc\hadoop\core-site.xml
3. %HADOOP_HOME%\etc\hadoop\mapred-site.xml
4. %HADOOP_HOME%\etc\hadoop\yarn-site.xml

4.1. HDFS site configuration

As we know, Hadoop is built using a master-slave paradigm. Before altering the HDFS configuration file, we should create a directory to store all master node (name node) data and another one to store data (data node). In this example, we created the following directories:

- E:\hadoop-env\hadoop-3.2.1\data\dfs\namenode
- E:\hadoop-env\hadoop-3.2.1\data\dfs\datanode

Now, let's open "hdfs-site.xml" file located in "%HADOOP_HOME%\etc\hadoop" directory, and we should add the following properties within the <configuration></configuration> element:

```
<property><name>dfs.replication</name><value>1</value></
property><property><name>dfs.namenode.name.dir</name><value>file:///E:/hadoop-env/
hadoop-3.2.1/data/dfs/namenode</value></
property><property><name>dfs.datanode.data.dir</name><value>file:///E:/hadoop-env/
hadoop-3.2.1/data/dfs/datanode</value></property>
```

Note that we have set the replication factor to 1 since we are creating a single node cluster.

4.2. Core site configuration

Now, we should configure the name node URL adding the following XML code into the <configuration></configuration> element within "core-site.xml":

```
<property><name>fs.default.name</name><value>hdfs://localhost:9820</value></
property>
```

4.3. Map Reduce site configuration

Now, we should add the following XML code into the <configuration></configuration> element within "mapred-site.xml":

```
<property><name>mapreduce.framework.name</name><value>yarn</
value><description>MapReduce framework name</description></property>
```

4.4. Yarn site configuration

Now, we should add the following XML code into the <configuration></configuration> element within "yarn-site.xml":

```
<property><name>yarn.nodemanager.aux-services</name><value>mapreduce_shuffle</
value><description>Yarn Node Manager Aux Service</description></property>
```

5. Formatting Name node

After finishing the configuration, let's try to format the name node using the following command:

```
hdfs namenode -format
```

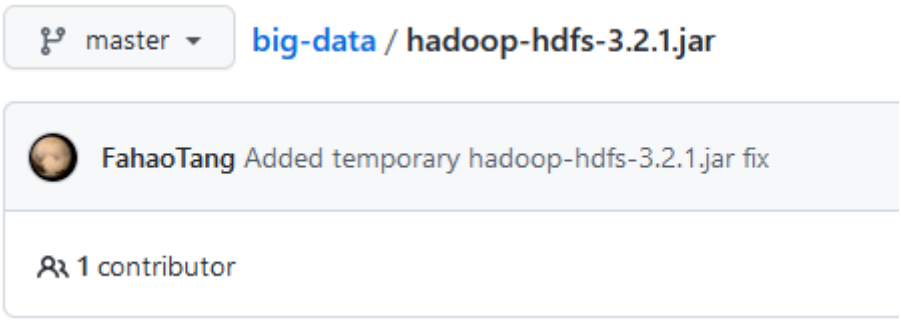
Due to a bug in the Hadoop 3.2.1 release, you will receive the following error:

```
2020–04–17 22:04:01,503 ERROR namenode.NameNode: Failed to start
namenode.java.lang.UnsupportedOperationExceptionat
java.nio.file.Files.setPosixFilePermissions(Files.java:2044)at
org.apache.hadoop.hdfs.server.common.Storage$StorageDirectory.clearDirectory(Storag
e.java:452)at
```

```
org.apache.hadoop.hdfs.server.namenode.NNStorage.format(NNStorage.java:591)at
org.apache.hadoop.hdfs.server.namenode.NNStorage.format(NNStorage.java:613)at
org.apache.hadoop.hdfs.server.namenode.FSImage.format(FSImage.java:188)at
org.apache.hadoop.hdfs.server.namenode.NameNode.format(NameNode.java:1206)at
org.apache.hadoop.hdfs.server.namenode.NameNode.createNameNode(NameNode.java:1649)a
t org.apache.hadoop.hdfs.server.namenode.NameNode.main(NameNode.java:1759)2020-04-
17 22:04:01,511 INFO util.ExitUtil: Exiting with status 1:
java.lang.UnsupportedOperationException2020-04-17 22:04:01,518 INFO
namenode.NameNode: SHUTDOWN_MSG:
```

This issue will be solved within the next release. For now, you can fix it temporarily using the following steps (reference):

1. Download hadoop-hdfs-3.2.1.jar file from the following link.



2. Rename the file name hadoop-hdfs-3.2.1.jar to hadoop-hdfs-3.2.1.bak in folder %HADOOP_HOME%\share\hadoop\hdfs
3. Copy the downloaded hadoop-hdfs-3.2.1.jar to folder %HADOOP_HOME%\share\hadoop\hdfs

Now, if we try to re-execute the format command (Run the command prompt or PowerShell as administrator), you need to approve file system format.

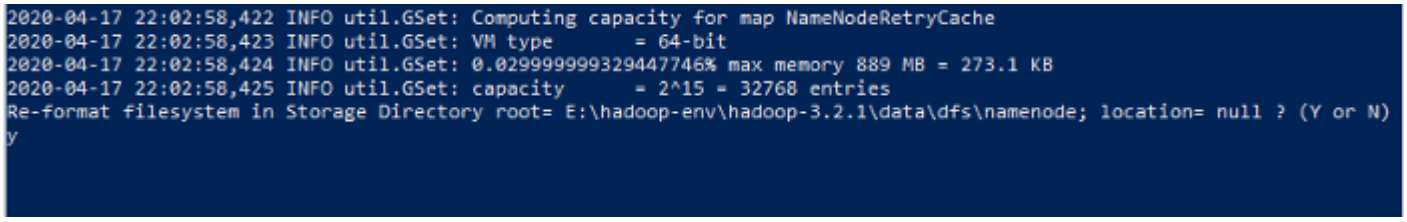


Figure 15 — File system format approval

And the command is executed successfully:

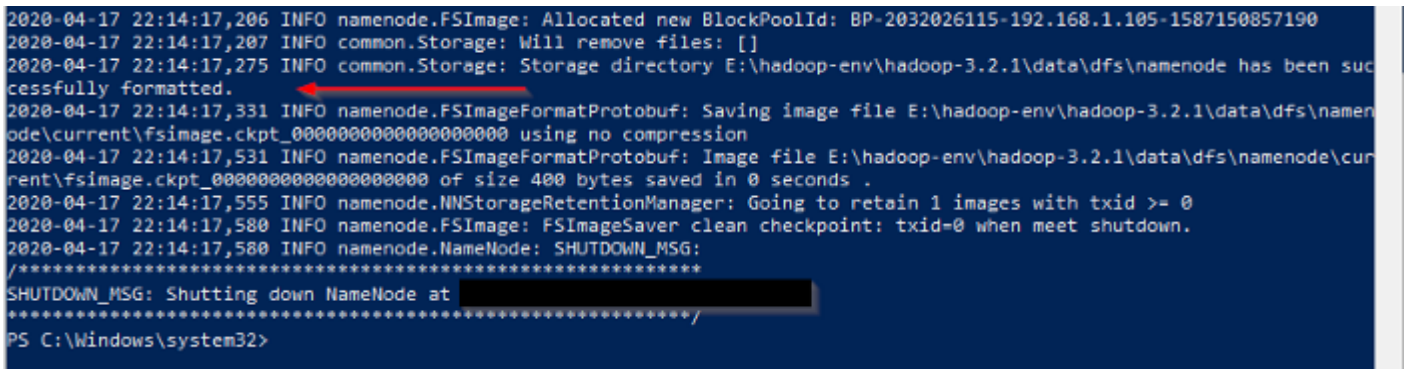


Figure 16 — Command executed successfully

6. Starting Hadoop services

Now, we will open PowerShell, and navigate to “%HADOOP_HOME%\sbin” directory. Then we will run the following command to start the Hadoop nodes:

```
.\start-dfs.cmd
```

Figure 17 — StartingHadoop nodes

Two command prompt windows will open (one for the name node and one for the data node) as follows:

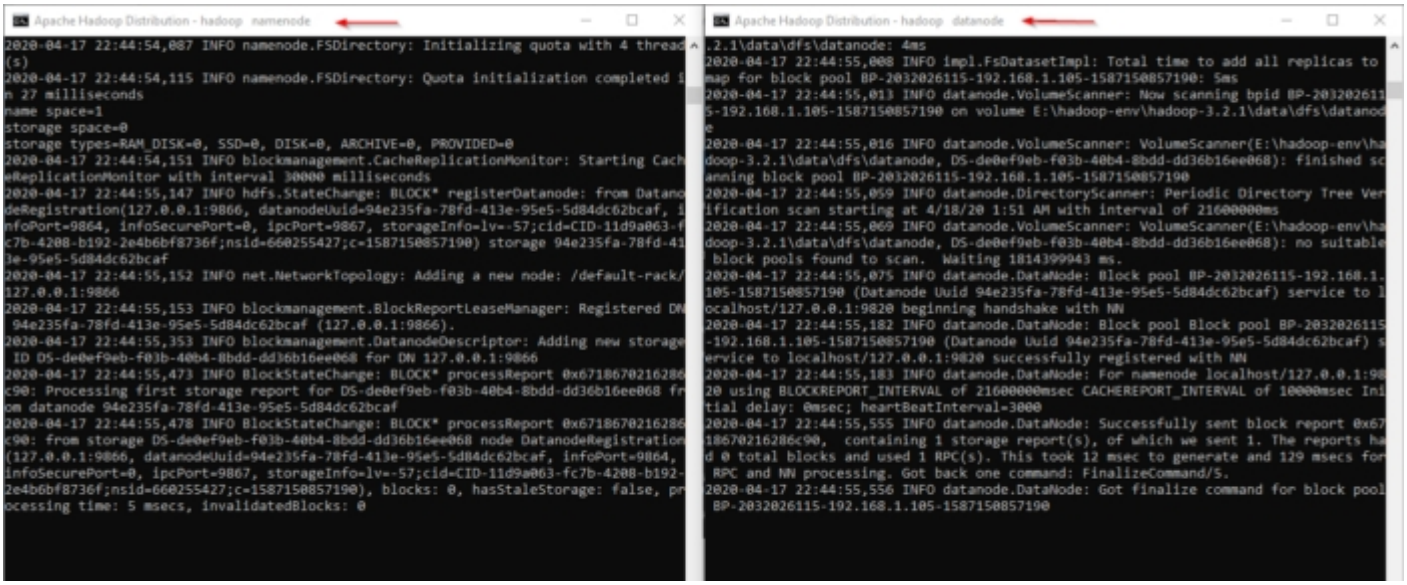


Figure 18 — Hadoop nodes command prompt windows

Next, we must start the Hadoop Yarn service using the following command:

```
./start-yarn.cmd
```

Figure 19 — Starting Hadoop Yarn services

Two command prompt windows will open (one for the resource manager and one for the node manager) as follows:

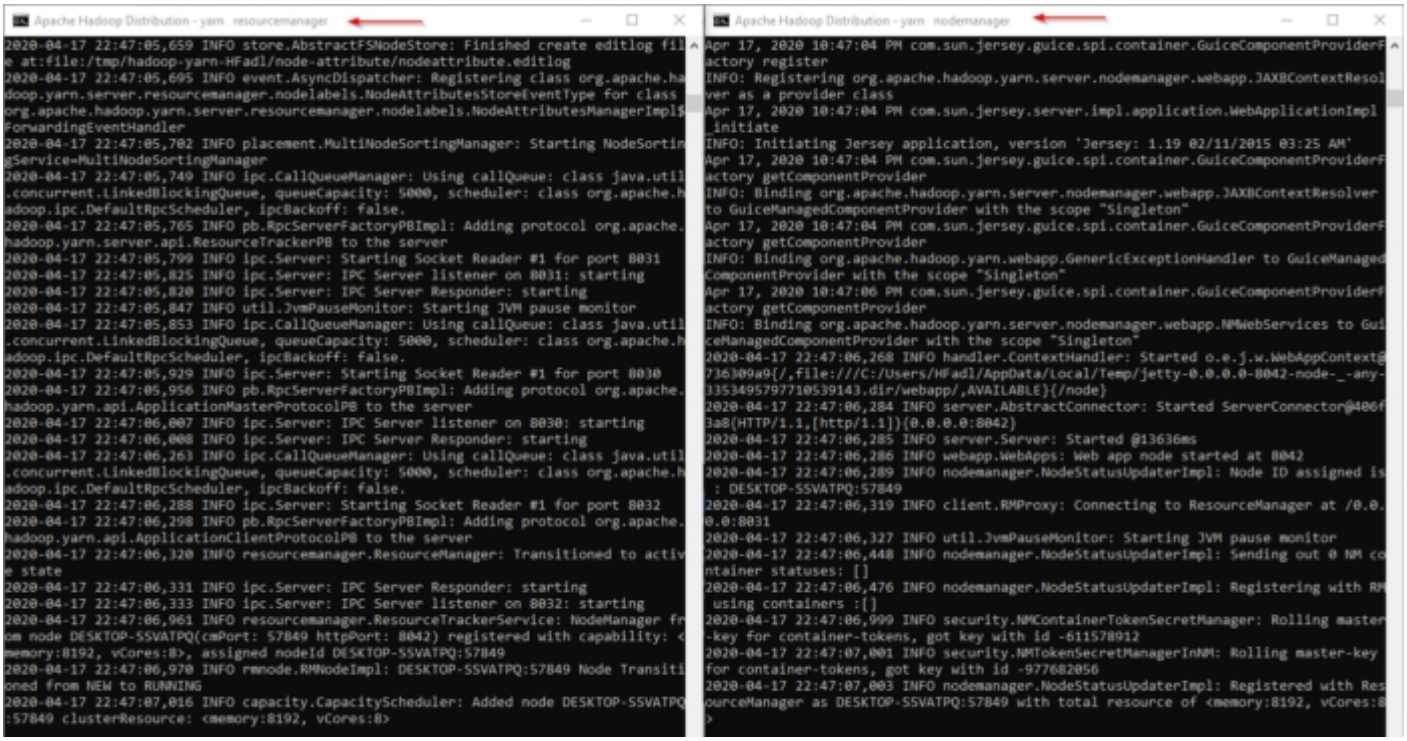


Figure 20— Node manager and Resource manager command prompt windows

To make sure that all services started successfully, we can run the following command:

```
jps
```

It should display the following services:

```
14560 DataNode
4960 ResourceManager
5936 NameNode
768 NodeManager
14636 Jps
```

```
PS E:\hadoop-env\hadoop-3.2.1\sbin> jps
14560 DataNode
4960 ResourceManager
5936 NameNode
768 NodeManager
14636 Jps
PS E:\hadoop-env\hadoop-3.2.1\sbin>
```

Figure 21 — Executing jps command

7. Hadoop Web UI

There are three web user interfaces to be used:

- Name node web page: http://localhost:9870/dfshealth.html

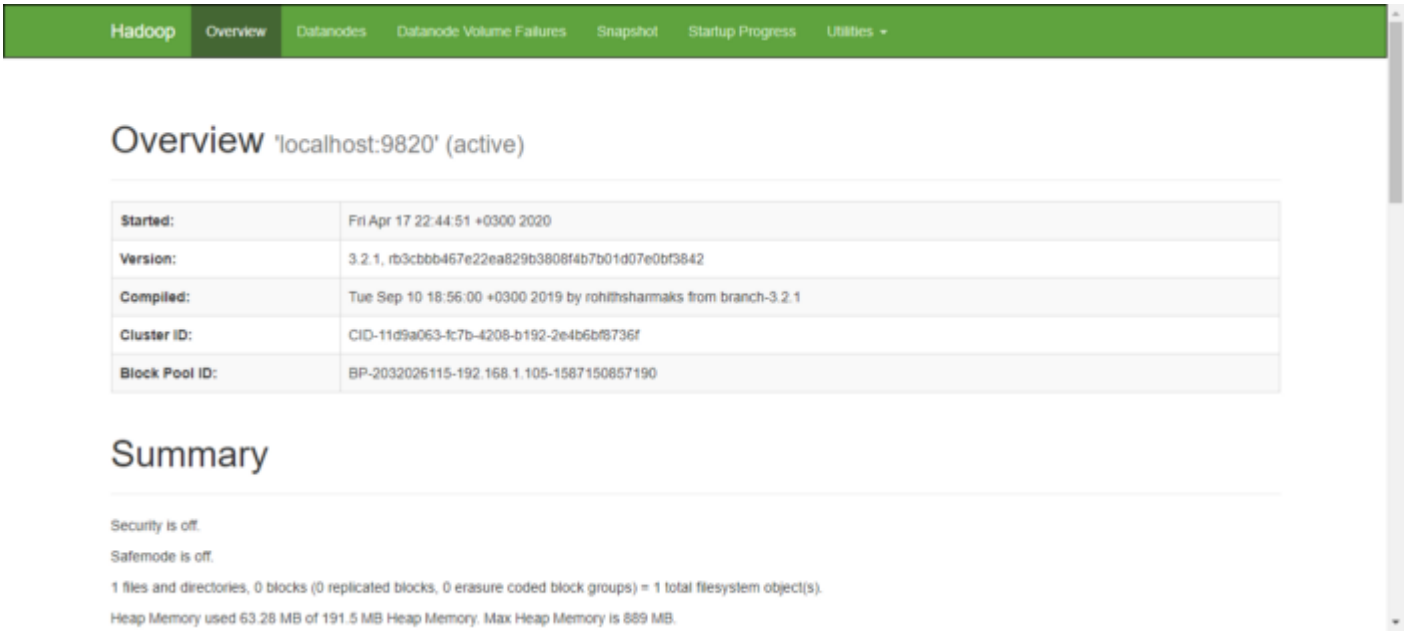


Figure 22 — Name node web page

- Data node web page: http://localhost:9864/datanode.html

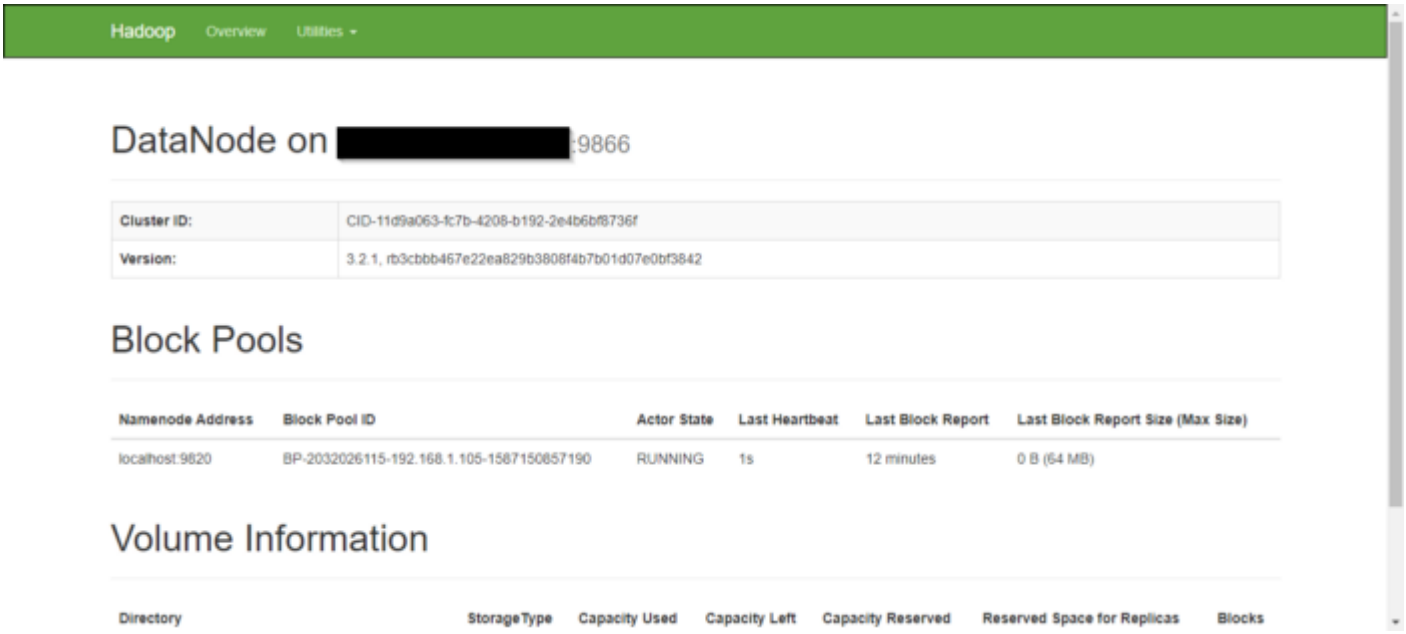


Figure 23 — Data node web page

- Yarn web page: http://localhost:8088/cluster

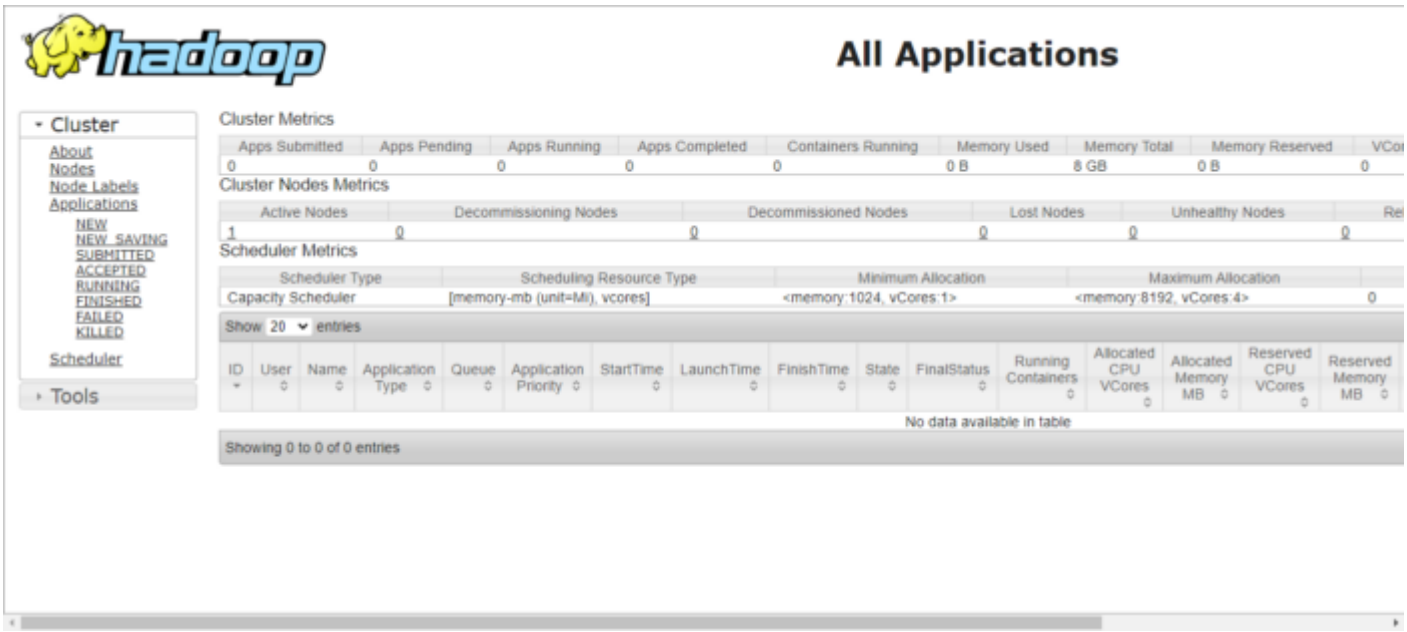


Figure 24 — Yarn web page

8. References

1. Hadi Fadlallah, Installing Hadoop 3.1.0 multi-node cluster on Ubuntu 16.04 Step by Step, TowardsDataScience.com
2. Jozef Jarosciak, How to install a Hadoop single node cluster on Windows 10
3. Raymond Tang, Install Hadoop 3.2.1 on Windows 10 Step by Step Guide, kontext.tech

4. Stack overflow Q/A website

Win 10 + hadoop 3.2.1 (build from sources)

1. run Command Prompt in admin mode
2. execute etc\hadoop\hadoop-env.cmd
3. run sbin\start-dfs.cmd
4. run sbin\start-yarn.cmd
5. now try to run your job

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>test</groupId>
  <artifactId>WCexample</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core -->
  <dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-core</artifactId>
      <version>3.2.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client -->
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>3.2.1</version>
    </dependency>
  </dependencies>

</project>
```

```
package test.wcexample;
```

```
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.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WC_Mapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    @Override
    public void map(LongWritable key, Text
value,OutputCollector<Text,IntWritable> output,
           Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer  tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }

}
```

```
package test.wcexample;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

```
package test.wcexample;
```

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class WC_Runner {
    public static void main(String[] args) throws IOException {
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

```
package tokenizing;

import java.util.StringTokenizer;

public class Tokenizing {
    public static void main(String[] args) {

        String txt="Arnaki, Arnaki *-(){]aspro kai paxy!";

        txt=txt.replaceAll("\\p{Punct}", "");

        StringTokenizer  tokenizer = new StringTokenizer(txt);
        while (tokenizer.hasMoreTokens()){
            System.out.println(tokenizer.nextToken());
        }
    }
}
```

```
hadoop jar .\WCexample.jar test.wcexample.WC_Runner /gutenberg/pg4300.txt
/gutenberg_output

hadoop jar .\WCexample.jar test.wcexample.WC_Runner /test/data.txt /r_output
```

https://hadoop.apache.org/docs/r3.2.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Michael G. Noll

Writing An Hadoop MapReduce Program In Python

In this tutorial I will describe how to write a simple MapReduce program for Hadoop in the Python programming language.

Motivation

Even though the Hadoop framework is written in Java, programs for Hadoop need not to be coded in Java but can also be developed in other languages like Python or C++ (the latter since version 0.14.1). However, Hadoop's documentation and the most prominent Python example on the Hadoop website could make you think that you *must* translate your Python code using Jython into a Java jar file. Obviously, this is not very convenient and can even be problematic if you depend on Python features not provided by Jython. Another issue of the Jython approach is the overhead of writing your Python program in such a way that it can interact with Hadoop – just have a look at the example in `$HADOOP_HOME/src/examples/python/WordCount.py` and you see what I mean.

That said, the ground is now prepared for the purpose of this tutorial: writing a Hadoop MapReduce program in a more Pythonic way, i.e. in a way you should be familiar with.

What we want to do

We will write a simple MapReduce program (see also the MapReduce article on Wikipedia) for Hadoop in Python but *without* using Jython to translate our code to Java jar files.

Our program will mimick the WordCount, i.e. it reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occured, separated by a tab.

Note: You can also use programming languages other than Python such as Perl or Ruby with the "technique" described in this tutorial.

Prerequisites

You should have an Hadoop cluster up and running because we will get our hands dirty. If you don't have a cluster yet, my following tutorials might help you to build one. The tutorials are tailored to Ubuntu Linux but the information does also apply to other Linux/Unix variants.

- Running Hadoop On Ubuntu Linux (Single-Node Cluster) – How to set up a *pseudo-distributed*, *single-node* Hadoop cluster backed by the Hadoop Distributed File System (HDFS)
- Running Hadoop On Ubuntu Linux (Multi-Node Cluster) – How to set up a *distributed*, *multi-node* Hadoop cluster backed by the Hadoop Distributed File System (HDFS)

Python MapReduce Code

The "trick" behind the following Python code is that we will use the Hadoop Streaming API (see also the corresponding wiki entry) for helping us passing data between our Map and Reduce code via `STDIN` (standard input) and `STDOUT` (standard output). We will simply use Python's `sys.stdin` to read input data and print our own output to `sys.stdout`. That's all we need to do because Hadoop Streaming will take care of everything else!

Map step: mapper.py

Save the following code in the file `/home/hduser/mapper.py`. It will read data from `STDIN`, split it into words and output a list of lines mapping words to their (intermediate) counts to `STDOUT`. The Map script will not compute an (intermediate) sum of a word's occurrences though. Instead, it will output `<word> 1` tuples immediately – even though a specific word might occur multiple times in the input. In our case we let the subsequent Reduce step do the final sum count. Of course, you can change this behavior in your own scripts as you please, but we will keep it like that in this tutorial because of didactic reasons. :-)

Make sure the file has execution permission (`chmod +x /home/hduser/mapper.py` should do the trick) or you will run into problems.

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Reduce step: reducer.py

Save the following code in the file `/home/hduser/reducer.py`. It will read the results of `mapper.py` from `STDIN` (so the output format of `mapper.py` and the expected input format of `reducer.py` must match) and sum the occurrences of each word to a final count, and then output its results to `STDOUT`.

Make sure the file has execution permission (`chmod +x /home/hduser/reducer.py` should do the trick) or you will run into problems.

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this 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)
```

Test your code (cat data | map | sort | reduce)

I recommend to test your `mapper.py` and `reducer.py` scripts locally before using them in a MapReduce job. Otherwise your jobs might successfully complete but there will be no job result data at all or not the results you would have expected. If that happens, most likely it was you (or me) who screwed up.

Here are some ideas on how to test the functionality of the Map and Reduce scripts.

```
# Test mapper.py and reducer.py locally first

# very basic test
hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py
foo     1
foo     1
quux    1
labs    1
foo     1
bar     1
quux    1

hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py |
sort -k1,1 | /home/hduser/reducer.py
bar     1
foo     3
labs    1
quux    2

# using one of the ebooks as example input
# (see below on where to get the ebooks)
hduser@ubuntu:~$ cat /tmp/gutenberg/20417-8.txt | /home/hduser/mapper.py
 The     1
 Project 1
 Gutenberg       1
 EBook   1
 of      1
 [...]
 (you get the idea)
```

Running the Python Code on Hadoop

Download example input data

We will use three ebooks from Project Gutenberg for this example:

- The Outline of Science, Vol. 1 (of 4) by J. Arthur Thomson
- The Notebooks of Leonardo Da Vinci
- Ulysses by James Joyce

Download each ebook as text files in `Plain Text UTF-8` encoding and store the files in a local temporary directory of choice, for example `/tmp/gutenberg`.

```
hduser@ubuntu:~$ ls -l /tmp/gutenberg/
total 3604
-rw-r--r-- 1 hduser hadoop  674566 Feb  3 10:17 pg20417.txt
-rw-r--r-- 1 hduser hadoop 1573112 Feb  3 10:18 pg4300.txt
-rw-r--r-- 1 hduser hadoop 1423801 Feb  3 10:18 pg5000.txt
hduser@ubuntu:~$
```

Copy local example data to HDFS

Before we run the actual MapReduce job, we must first copy the files from our local file system to Hadoop's HDFS.

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal /tmp/gutenberg
/user/hduser/gutenberg
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2010-05-08 17:40 /user/hduser/gutenberg
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser/gutenberg
Found 3 items
-rw-r--r--   3 hduser supergroup     674566 2011-03-10 11:38
/user/hduser/gutenberg/pg20417.txt
-rw-r--r--   3 hduser supergroup    1573112 2011-03-10 11:38
/user/hduser/gutenberg/pg4300.txt
-rw-r--r--   3 hduser supergroup    1423801 2011-03-10 11:38
/user/hduser/gutenberg/pg5000.txt
hduser@ubuntu:/usr/local/hadoop$
```

Run the MapReduce job

Now that everything is prepared, we can finally run our Python MapReduce job on the Hadoop cluster. As I said above, we leverage the Hadoop Streaming API for helping us passing data between our Map and Reduce code via `STDIN` and `STDOUT`.

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-
*streaming*.jar \
-file /home/hduser/mapper.py    -mapper /home/hduser/mapper.py \
-file /home/hduser/reducer.py   -reducer /home/hduser/reducer.py \
-input /user/hduser/gutenberg/* -output /user/hduser/gutenberg-output
```

If you want to modify some Hadoop settings on the fly like increasing the number of Reduce tasks, you can use the `-D` option:

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-
*streaming*.jar -D mapred.reduce.tasks=16 ...
```

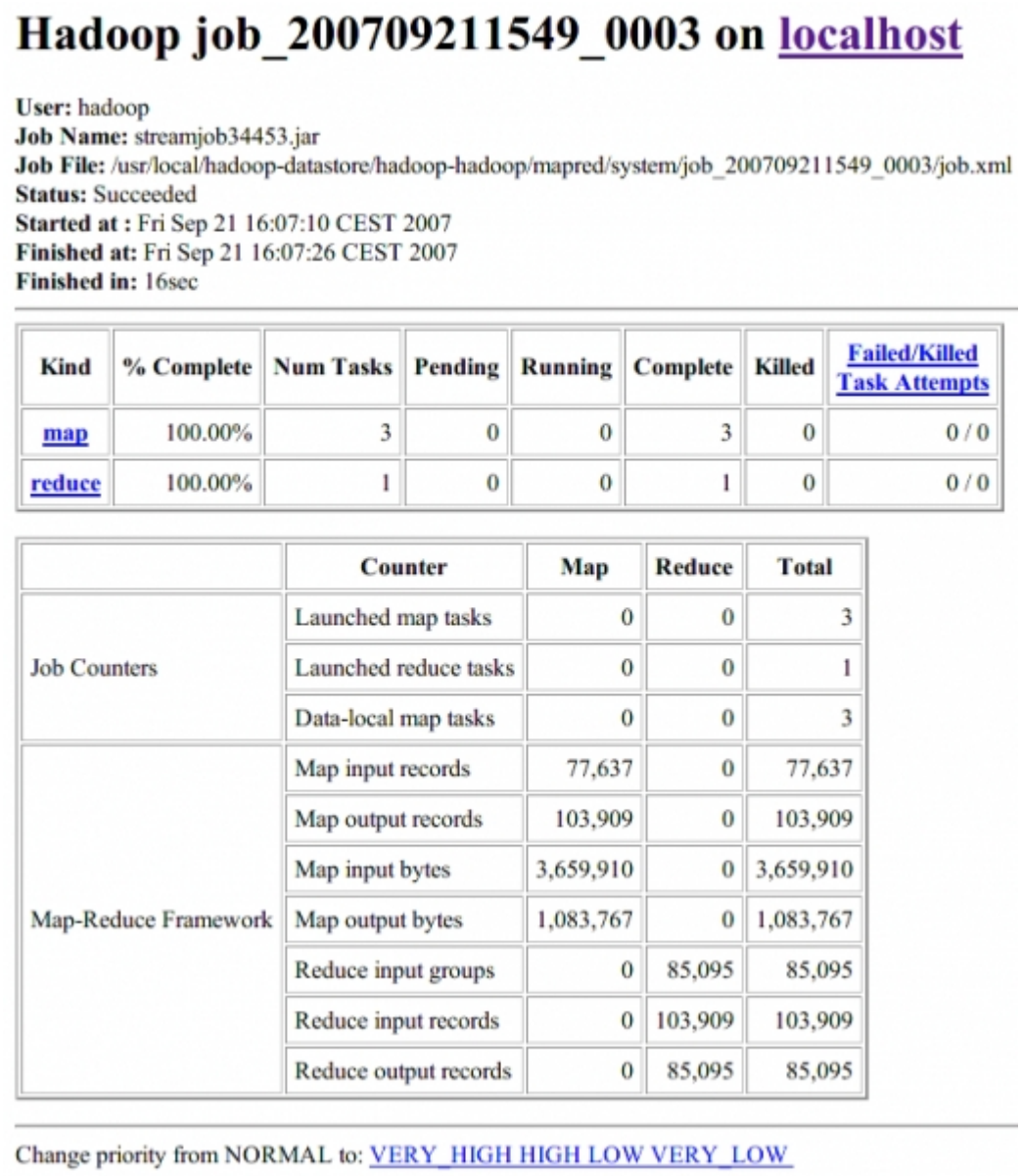
Note about `mapred.map.tasks`: Hadoop does not honor mapred.map.tasks beyond considering it a hint. But it accepts the user specified `mapred.reduce.tasks` and doesn't manipulate that. You cannot force `mapred.map.tasks` but can specify `mapred.reduce.tasks`.

The job will read all the files in the HDFS directory `/user/hduser/gutenberg`, process it, and store the results in the HDFS directory `/user/hduser/gutenberg-output`. In general Hadoop will create one output file per reducer; in our case however it will only create a single file because the input files are very small.

Example output of the previous command in the console:

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-
*streaming*.jar -mapper /home/hduser/mapper.py -reducer /home/hduser/reducer.py -
input /user/hduser/gutenberg/* -output /user/hduser/gutenberg-output
 additionalConfSpec_:null
 null=@@@userJobConfProps_.get(stream.shipped.hadoopstreaming
 packageJobJar: [/app/hadoop/tmp/hadoop-unjar54543/]
 [] /tmp/streamjob54544.jar tmpDir=null
 [...] INFO mapred.FileInputFormat: Total input paths to process : 7
 [...] INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
 [...] INFO streaming.StreamJob: Running job: job_200803031615_0021
 [...]
 [...] INFO streaming.StreamJob:  map 0%  reduce 0%
 [...] INFO streaming.StreamJob:  map 43%  reduce 0%
 [...] INFO streaming.StreamJob:  map 86%  reduce 0%
 [...] INFO streaming.StreamJob:  map 100%  reduce 0%
 [...] INFO streaming.StreamJob:  map 100%  reduce 33%
 [...] INFO streaming.StreamJob:  map 100%  reduce 70%
 [...] INFO streaming.StreamJob:  map 100%  reduce 77%
 [...] INFO streaming.StreamJob:  map 100%  reduce 100%
 [...] INFO streaming.StreamJob: Job complete: job_200803031615_0021
 [...] INFO streaming.StreamJob: Output: /user/hduser/gutenberg-output
hduser@ubuntu:/usr/local/hadoop$
```

As you can see in the output above, Hadoop also provides a basic web interface for statistics and information. When the Hadoop cluster is running, open http://localhost:50030/ in a browser and have a look around. Here’s a screenshot of the Hadoop web interface for the job we just ran.



Hadoop job_200709211549_0003 on localhost

User: hadoop
Job Name: streamjob34453.jar
Job File: /usr/local/hadoop-datastore/hadoop-hadoop/mapred/system/job_200709211549_0003/job.xml
Status: Succeeded
Started at : Fri Sep 21 16:07:10 CEST 2007
Finished at: Fri Sep 21 16:07:26 CEST 2007
Finished in: 16sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	3	0	0	3	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	3
	Launched reduce tasks	0	0	1
	Data-local map tasks	0	0	3
Map-Reduce Framework	Map input records	77,637	0	77,637
	Map output records	103,909	0	103,909
	Map input bytes	3,659,910	0	3,659,910
	Map output bytes	1,083,767	0	1,083,767
	Reduce input groups	0	85,095	85,095
	Reduce input records	0	103,909	103,909
	Reduce output records	0	85,095	85,095

Change priority from NORMAL to: VERY_HIGH HIGH LOW VERY_LOW

Figure 1: A screenshot of Hadoop's JobTracker web interface, showing the details of the MapReduce job we just ran

Check if the result is successfully stored in HDFS directory `/user/hduser/gutenberg-output`:

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser/gutenberg-output
Found 1 items
/user/hduser/gutenberg-output/part-00000     &lt;r 1&gt;   903193  2007-09-21 13:00
hduser@ubuntu:/usr/local/hadoop$
```

You can then inspect the contents of the file with the `dfs -cat` command:

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -cat /user/hduser/gutenberg-
output/part-00000
"(Lo)cra"       1
"1490   1
```

```
"1498," 1
"35"    1
"40,"   1
"A      2
"AS-IS".        2
"A_     1
"Absoluti       1
[...]
hduser@ubuntu:/usr/local/hadoop$
```

Note that in this specific output above the quote signs (") enclosing the words have not been inserted by Hadoop. They are the result of how our Python code splits words, and in this case it matched the beginning of a quote in the ebook texts. Just inspect the `part-00000` file further to see it for yourself.

Improved Mapper and Reducer code: using Python iterators and generators

The Mapper and Reducer examples above should have given you an idea of how to create your first MapReduce application. The focus was code simplicity and ease of understanding, particularly for beginners of the Python programming language. In a real-world application however, you might want to optimize your code by using Python iterators and generators (an even better introduction in PDF).

Generally speaking, iterators and generators (functions that create iterators, for example with Python's `yield` statement) have the advantage that an element of a sequence is not produced until you actually need it. This can help a lot in terms of computational expensiveness or memory consumption depending on the task at hand.

Note: The following Map and Reduce scripts will only work "correctly" when being run in the Hadoop context, i.e. as Mapper and Reducer in a MapReduce job. This means that running the naive test command "cat DATA | ./mapper.py | sort -k1,1 | ./reducer.py" will not work correctly anymore because some functionality is intentionally outsourced to Hadoop.

Precisely, we compute the sum of a word's occurrences, e.g. `("foo", 4)`, only if by chance the same word (`foo`) appears multiple times in succession. In the majority of cases, however, we let the Hadoop group the (key, value) pairs between the Map and the Reduce step because Hadoop is more efficient in this regard than our simple Python scripts.

mapper.py

```
#!/usr/bin/env python
"""A more advanced Mapper, using Python iterators and generators."""

import sys

def read_input(file):
    for line in file:
        # split the line into words
        yield line.split()
```

```
def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        for word in words:
            print '%s%s%d' % (word, separator, 1)

if __name__ == "__main__":
    main()
```

reducer.py

```
#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their group:
    #   current_word - string containing a word (the key)
    #   group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"]
items
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count) for current_word, count in group)
            print "%s%s%d" % (current_word, separator, total_count)
        except ValueError:
            # count was not a number, so silently discard this item
            pass

if __name__ == "__main__":
    main()
```

Related Links

From yours truly:

- Running Hadoop On Ubuntu Linux (Single-Node Cluster)
- Running Hadoop On Ubuntu Linux (Multi-Node Cluster)

MapReduce Word Count Example

In MapReduce word count example, we find out the frequency of each word. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

Pre-requisite

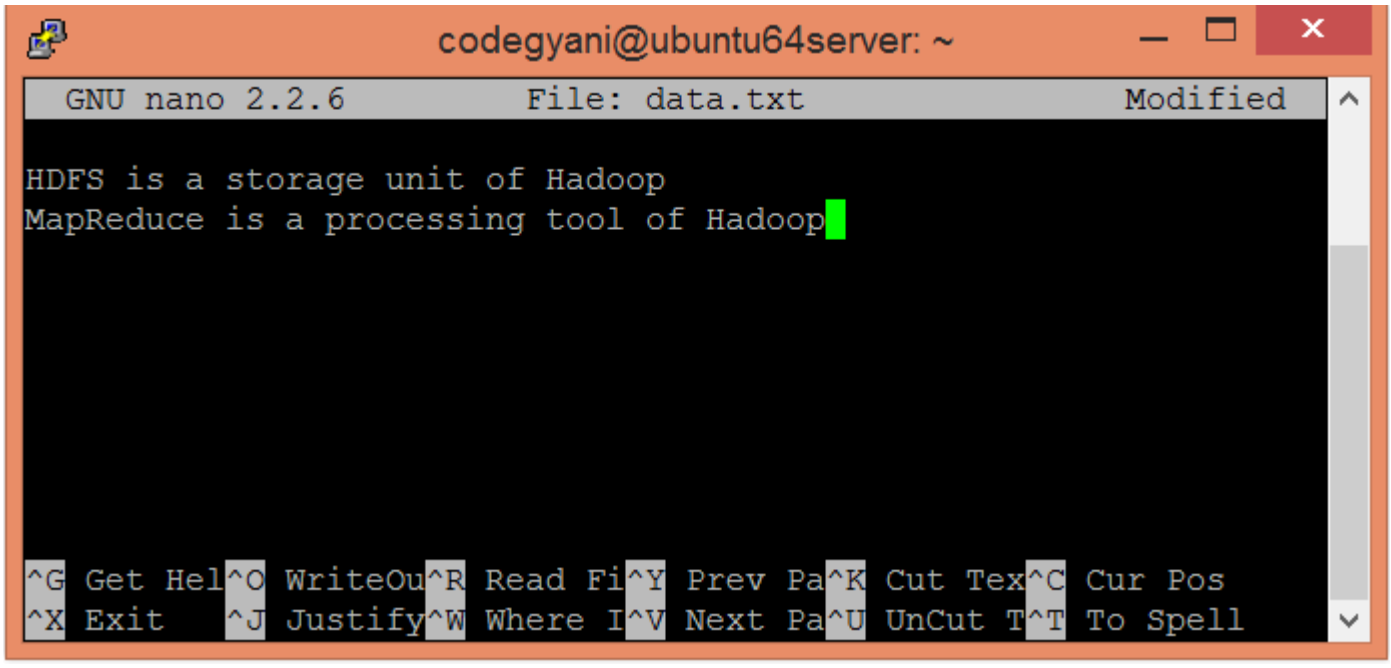
- **Java Installation -** Check whether the Java is installed or not using the following command.
  java -version
- **Hadoop Installation -** Check whether the Hadoop is installed or not using the following command.
  hadoop version

If any of them is not installed in your system, follow the below link to install it.

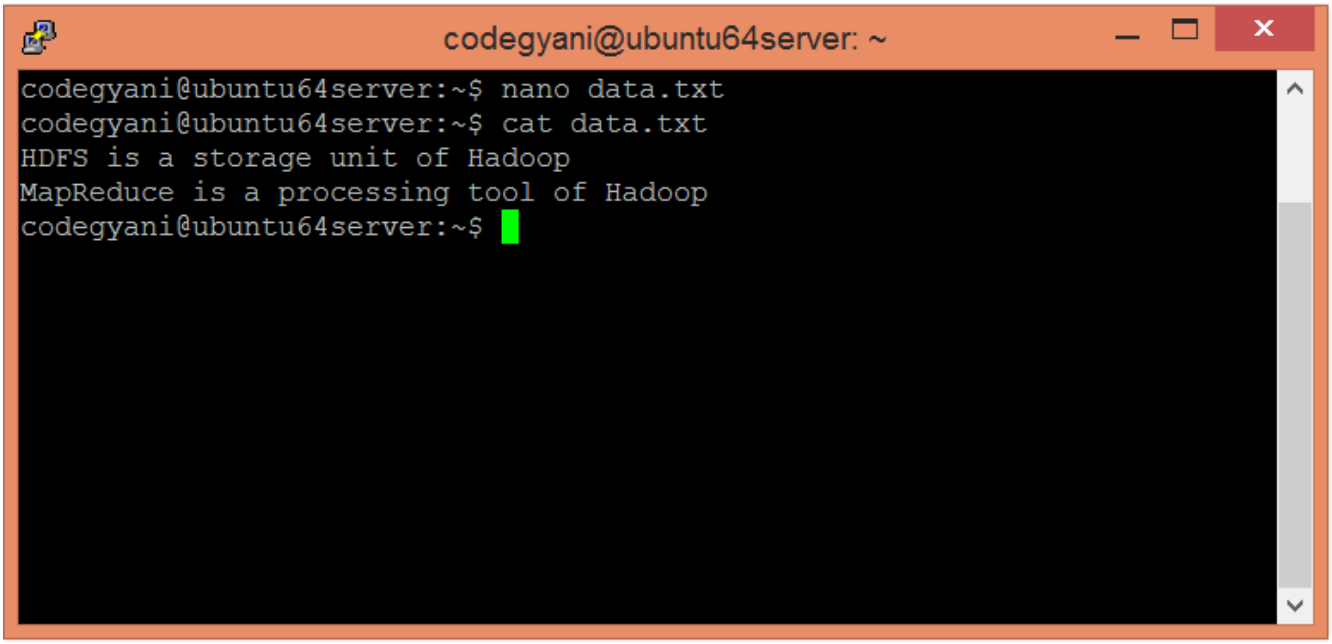
www.javatpoint.com/hadoop-installation

Steps to execute MapReduce word count example

- Create a text file in your local machine and write some text into it.
  $ nano data.txt

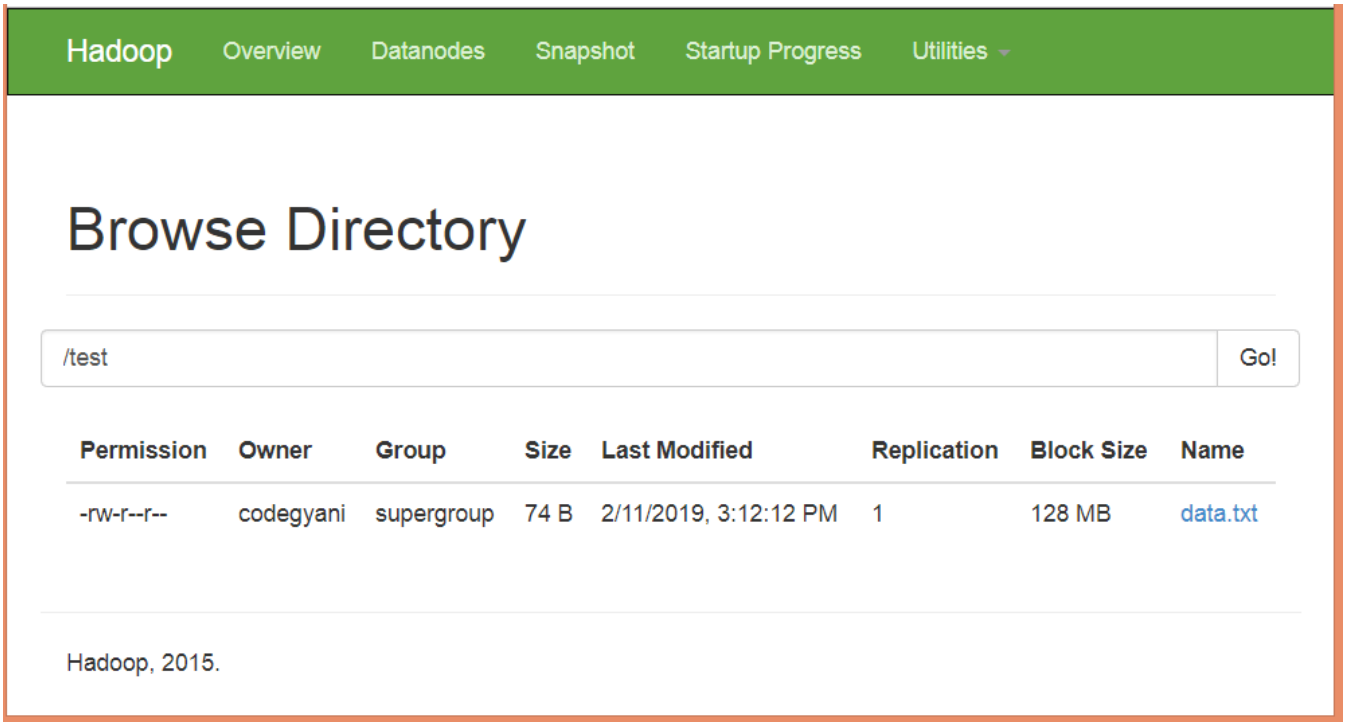


- Check the text written in the data.txt file.
  $ cat data.txt



In this example, we find out the frequency of each word exists in this text file.

- Create a directory in HDFS, where to kept text file.
  $ hdfs dfs -mkdir /test
- Upload the data.txt file on HDFS in the specific directory.
  $ hdfs dfs -put /home/codegyani/data.txt /test



- Write the MapReduce program using eclipse.

File: WC_Mapper.java

```
1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import java.util.StringTokenizer;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.LongWritable;
7. import org.apache.hadoop.io.Text;
8. import org.apache.hadoop.mapred.MapReduceBase;
9. import org.apache.hadoop.mapred.Mapper;
10.import org.apache.hadoop.mapred.OutputCollector;
11.import org.apache.hadoop.mapred.Reporter;
12.public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Te
   xt,IntWritable>{
13.    private final static IntWritable one = new IntWritable(1);
14.    private Text word = new Text();
15.    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
16.           Reporter reporter) throws IOException{
17.        String line = value.toString();
18.        StringTokenizer  tokenizer = new StringTokenizer(line);
19.        while (tokenizer.hasMoreTokens()){
20.            word.set(tokenizer.nextToken());
21.            output.collect(word, one);
22.        }
23.    }
24.
25.}
```

File: WC_Reducer.java

```
1. package com.javatpoint;
2.     import java.io.IOException;
3.     import java.util.Iterator;
4.     import org.apache.hadoop.io.IntWritable;
5.     import org.apache.hadoop.io.Text;
6.     import org.apache.hadoop.mapred.MapReduceBase;
7.     import org.apache.hadoop.mapred.OutputCollector;
8.     import org.apache.hadoop.mapred.Reducer;
9.     import org.apache.hadoop.mapred.Reporter;
10.
11.    public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,T
   ext,IntWritable> {
```

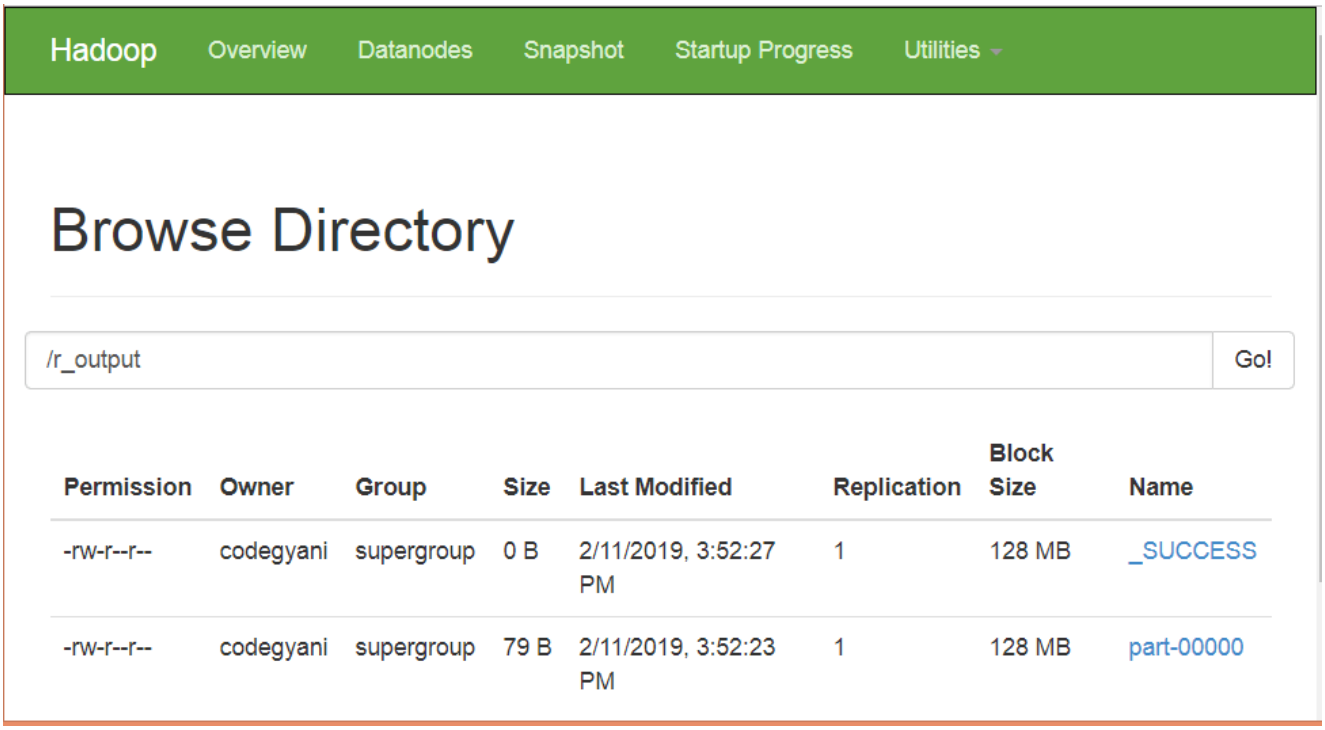
```
12.    public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable
    > output,
13.     Reporter reporter) throws IOException {
14.    int sum=0;
15.    while (values.hasNext()) {
16.    sum+=values.next().get();
17.    }
18.    output.collect(key,new IntWritable(sum));
19.    }
20.    }
```

File: WC_Runner.java

```
1.  package com.javatpoint;
2.
3.     import java.io.IOException;
4.     import org.apache.hadoop.fs.Path;
5.     import org.apache.hadoop.io.IntWritable;
6.     import org.apache.hadoop.io.Text;
7.     import org.apache.hadoop.mapred.FileInputFormat;
8.     import org.apache.hadoop.mapred.FileOutputFormat;
9.     import org.apache.hadoop.mapred.JobClient;
10.    import org.apache.hadoop.mapred.JobConf;
11.    import org.apache.hadoop.mapred.TextInputFormat;
12.    import org.apache.hadoop.mapred.TextOutputFormat;
13.    public class WC_Runner {
14.        public static void main(String[] args) throws IOException{
15.            JobConf conf = new JobConf(WC_Runner.class);
16.            conf.setJobName("WordCount");
17.            conf.setOutputKeyClass(Text.class);
18.            conf.setOutputValueClass(IntWritable.class);
19.            conf.setMapperClass(WC_Mapper.class);
20.            conf.setCombinerClass(WC_Reducer.class);
21.            conf.setReducerClass(WC_Reducer.class);
22.            conf.setInputFormat(TextInputFormat.class);
23.            conf.setOutputFormat(TextOutputFormat.class);
24.            FileInputFormat.setInputPaths(conf,new Path(args[0]));
25.            FileOutputFormat.setOutputPath(conf,new Path(args[1]));
26.            JobClient.runJob(conf);
27.        }
28.    }
```

Download the source code.

- Create the jar file of this program and name it **countworddemo.jar.**
- Run the jar file
  hadoop jar /home/codegyani/wordcountdemo.jar com.javatpoint.WC_Runner /test/data.txt /r_output
- The output is stored in /r_output/part-00000



- Now execute the command to see the output.
  hdfs dfs -cat /r_output/part-00000


codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ hdfs dfs -cat /r_output/part-00000
HDFS 1
Hadoop 2
MapReduce 1
a 2
is 2
of 2
processing 1
storage 1
tool 1
unit 1
codegyani@ubuntu64server:~$

HDFS Commands

- Difficulty Level : Easy
- Last Updated : 04 Apr, 2019

HDFS is the primary or major component of the Hadoop ecosystem which is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files. To use the HDFS commands, first you need to start the Hadoop services using the following command:

```
sbin/start-all.sh
```

To check the Hadoop services are up and running use the following command:

```
jps
```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ jps
2546 SecondaryNameNode
2404 DataNode
2295 NameNode
2760 ResourceManager
2874 NodeManager
4251 Jps
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

Commands:

1. **ls:** This command is used to list all the files. Use *lsr* for recursive approach. It is useful when we want a hierarchy of a folder.

   Syntax:

   ```
   bin/hdfs dfs -ls  <path>
   ```

   Example:

   ```
   bin/hdfs dfs -ls /
   ```

   It will print all the directories present in HDFS. bin directory contains executables so, *bin/hdfs* means we want the executables of hdfs particularly *dfs*(Distributed File System) commands.

   ```
   suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /
   19/01/31 10:35:11 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
   able
   Found 4 items
   -rw-r--r--   1 suraj supergroup   13965969 2019-01-31 00:13 /input
   drwxr-xr-x   - suraj supergroup          0 2019-01-31 01:30 /output
   drwx------   - suraj supergroup          0 2019-01-31 00:15 /tmp
   drwxr-xr-x   - suraj supergroup          0 2019-01-30 23:44 /user
   ```

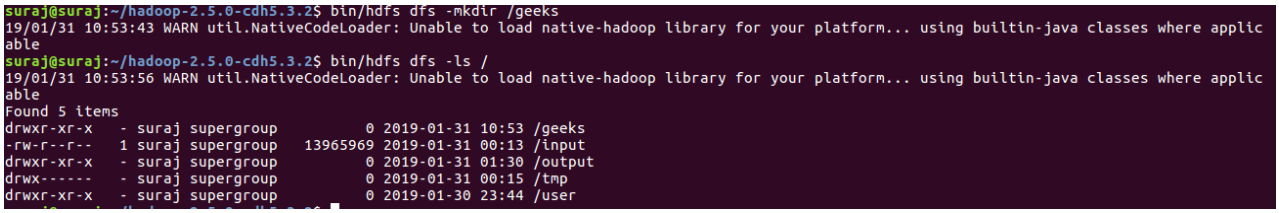
2. **mkdir**: To create a directory. In Hadoop *dfs* there is no home directory by default. So let's first create it.

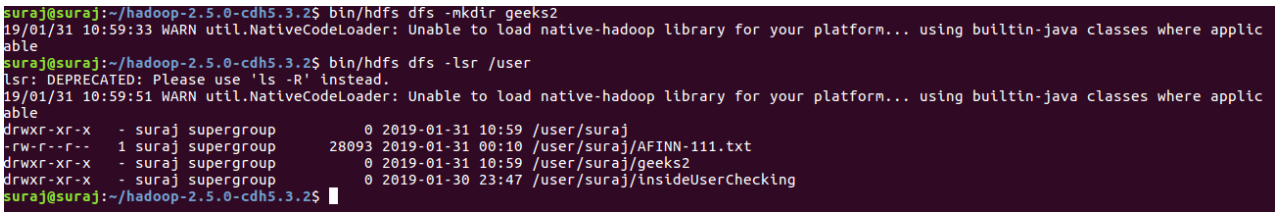
Syntax:

```
bin/hdfs dfs -mkdir <folder name>

creating home directory:

hdfs/bin -mkdir /user
hdfs/bin -mkdir /user/username -> write the username of your computer
```

Example:

```
bin/hdfs dfs -mkdir  /geeks  =>  '/' means absolute path
bin/hdfs dfs -mkdir  geeks2  =>   Relative path -> the folder will be
                                  created relative to the home directory.
```





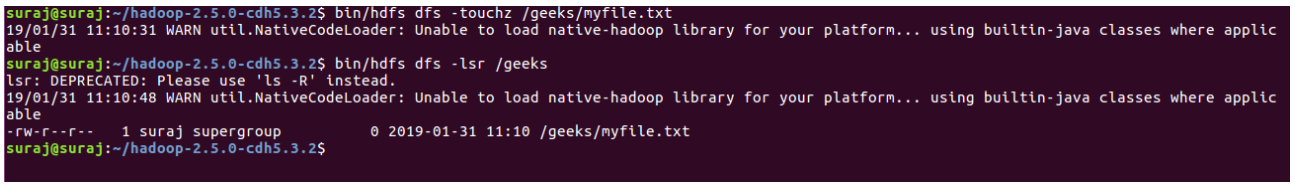
3. **touchz**: It creates an empty file.

   Syntax:

   ```
   bin/hdfs dfs  -touchz  <file_path>
   ```

   Example:

   ```
   bin/hdfs dfs -touchz  /geeks/myfile.txt
   ```



4. **copyFromLocal (or) put:** To copy files/folders from local file system to hdfs store. This is the most important command. Local filesystem means the files present on the OS.

   Syntax:

```
bin/hdfs dfs -copyFromLocal <local file path>  <dest(present on hdfs)>
```

Example: Let's suppose we have a file *AI.txt* on Desktop which we want to copy to folder *geeks* present on hdfs.

```
bin/hdfs dfs -copyFromLocal ../Desktop/AI.txt /geeks
```

```
(OR)
```

```
bin/hdfs dfs -put ../Desktop/AI.txt /geeks
```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -put ../Desktop/AI.txt /geeks
19/01/31 11:31:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -lsr /geeks
lsr: DEPRECATED: Please use 'ls -R' instead.
19/01/31 11:31:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
-rw-r--r--   1 suraj supergroup        205 2019-01-31 11:31 /geeks/AI.txt
-rw-r--r--   1 suraj supergroup          0 2019-01-31 11:10 /geeks/myfile.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

5. **cat:** To print file contents.

   Syntax:

   ```
   bin/hdfs dfs -cat <path>
   ```

   Example:

   ```
   // print the content of AI.txt present
   // inside geeks folder.
   bin/hdfs dfs -cat /geeks/AI.txt ->
   ```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -cat /geeks/AI.txt
19/01/31 11:33:25 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
In computer science, artificial intelligence, sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to t
he natural intelligence displayed by humans and other animals
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

6. **copyToLocal (or) get:** To copy files/folders from hdfs store to local file system.

   Syntax:

   ```
   bin/hdfs dfs -copyToLocal  <<srcfile(on hdfs)> <local file dest>
   ```

   Example:

   ```
   bin/hdfs dfs -copyToLocal  /geeks   ../Desktop/hero
   ```

   ```
   (OR)
   ```

   ```
   bin/hdfs dfs -get /geeks/myfile.txt  ../Desktop/hero
   ```

   myfile.txt from *geeks* folder will be copied to folder *hero* present on *Desktop*.

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -get /geeks/myfile.txt  ../Desktop/hero
19/01/31 11:43:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ ls ../Desktop/hero
myfile.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

Note: Observe that we don't write *bin/hdfs* while checking the things present on local filesystem.

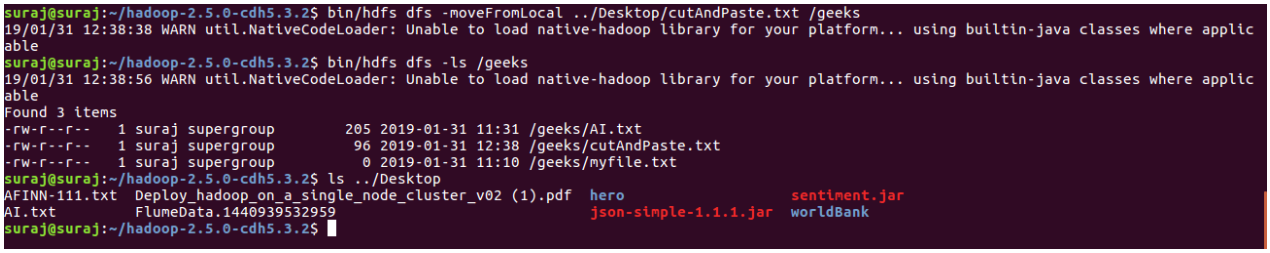
7. **moveFromLocal:** This command will move file from local to hdfs.

   Syntax:

   ```
   bin/hdfs dfs -moveFromLocal <local src>   <dest(on hdfs)>
   ```

   Example:

   ```
   bin/hdfs dfs -moveFromLocal  ../Desktop/cutAndPaste.txt   /geeks
   ```



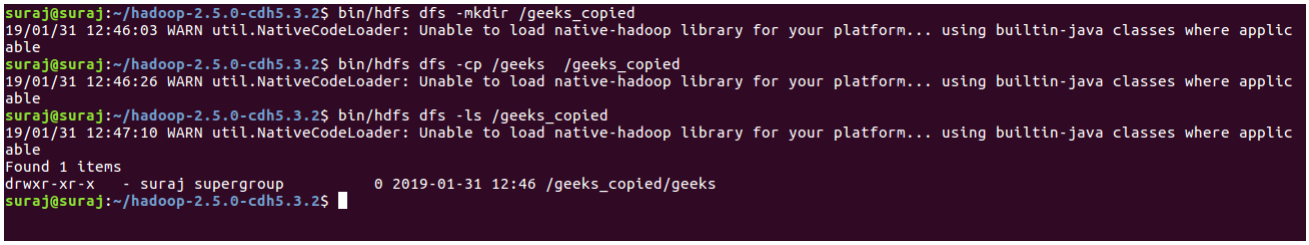
8. **cp:** This command is used to copy files within hdfs. Lets copy folder *geeks* to *geeks_copied*.

   Syntax:

   ```
   bin/hdfs dfs -cp  <src(on hdfs)>  <dest(on hdfs)>
   ```

   Example:

   ```
   bin/hdfs -cp /geeks  /geeks_copied
   ```



9. **mv:** This command is used to move files within hdfs. Lets cut-paste a file *myfile.txt* from *geeks* folder to *geeks_copied*.

   Syntax:

   ```
   bin/hdfs dfs -mv  <src(on hdfs)> <src(on hdfs)>
   ```

   Example:

   ```
   bin/hdfs  -mv  /geeks/myfile.txt  /geeks_copied
   ```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /geeks
19/01/31 12:48:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
Found 3 items
-rw-r--r--   1 suraj supergroup        205 2019-01-31 11:31 /geeks/AI.txt
-rw-r--r--   1 suraj supergroup         96 2019-01-31 12:38 /geeks/cutAndPaste.txt
-rw-r--r--   1 suraj supergroup          0 2019-01-31 11:10 /geeks/myfile.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -mv /geeks/myfile.txt  /geeks_copied
19/01/31 12:49:27 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /geeks
19/01/31 12:49:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
Found 2 items
-rw-r--r--   1 suraj supergroup        205 2019-01-31 11:31 /geeks/AI.txt
-rw-r--r--   1 suraj supergroup         96 2019-01-31 12:38 /geeks/cutAndPaste.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /geeks_copied
19/01/31 12:50:00 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
Found 2 items
drwxr-xr-x   - suraj supergroup          0 2019-01-31 12:46 /geeks_copied/geeks
-rw-r--r--   1 suraj supergroup          0 2019-01-31 11:10 /geeks_copied/myfile.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

10. **rmr:** This command deletes a file from HDFS *recursively*. It is very useful command when you want to delete a *non-empty directory*.

    Syntax:

    ```
    bin/hdfs dfs -rmr <filename/directoryName>
    ```

    Example:

    ```
    bin/hdfs dfs -rmr  /geeks_copied -> It will delete all the content inside the
                                        directory then the directory itself.
    ```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /geeks_copied
19/01/31 12:58:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
Found 2 items
drwxr-xr-x   - suraj supergroup          0 2019-01-31 12:46 /geeks_copied/geeks
-rw-r--r--   1 suraj supergroup          0 2019-01-31 11:10 /geeks_copied/myfile.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -rmr /geeks_copied
rmr: DEPRECATED: Please use 'rm -r' instead.
19/01/31 12:58:40 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
19/01/31 12:58:42 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /geeks_copied
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -ls /geeks_copied
19/01/31 12:59:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
ls: `/geeks_copied': No such file or directory
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

11. **du:** It will give the size of each file in directory.

    Syntax:

    ```
    bin/hdfs dfs -du  <dirName>
    ```

    Example:

    ```
    bin/hdfs dfs -du /geeks
    ```

```
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$ bin/hdfs dfs -du /geeks
19/01/31 13:01:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
205  205  /geeks/AI.txt
96   96   /geeks/cutAndPaste.txt
suraj@suraj:~/hadoop-2.5.0-cdh5.3.2$
```

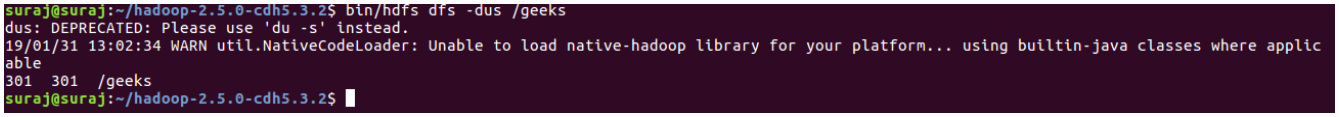
12. **dus:**: This command will give the total size of directory/file.

    Syntax:

    ```
    bin/hdfs dfs -dus  <dirName>
    ```

Example:

```
bin/hdfs dfs -dus /geeks
```



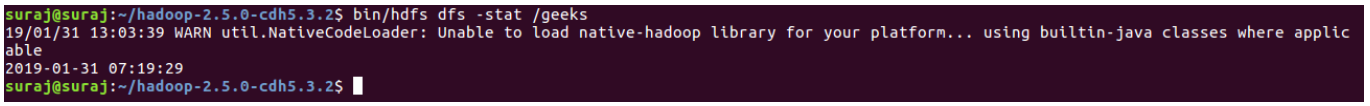
13. **stat:** It will give the last modified time of directory or path. In short it will give stats of the directory or file.

    Syntax:

    ```
    bin/hdfs  dfs -stat    <hdfs file>
    ```

    Example:

    ```
    bin/hdfs dfs -stat /geeks
    ```



14. **setrep:** This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs *core-site.xml*).

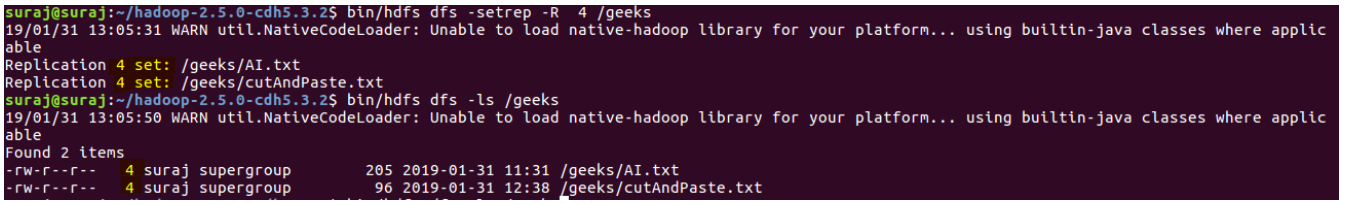
    Example 1: To change the replication factor to 6 for *geeks.txt* stored in HDFS.

    ```
    bin/hdfs dfs -setrep -R -w 6 geeks.txt
    ```

    Example 2: To change the replication factor to 4 for a directory *geeksInput* stored in HDFS.

    ```
    bin/hdfs dfs -setrep -R  4 /geeks
    ```

    Note: The **-w** means wait till the replication is completed. And **-R** means recursively, we use it for directories as they may also contain many files and folders inside them.



Note: There are more commands in HDFS but we discussed the commands which are commonly used when working with Hadoop. You can check out the list of *dfs* commands using the following command:

```
bin/hdfs dfs
```

```
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
        [-tail [-f] <file>]
        [-test -[defsz] <path>]
        [-text [-ignoreCrc] <src> ...]
```