EXPERIMENT NO: 01

Title: Introduction of Big Data & Hadoop
Aim: Study of Big Data & Hadoop.

Theory:

Data:

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

Big Data:

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.

Examples of Big Data:

Following are some of the Big Data examples-

New York Stock Exchange:

The **New York Stock Exchange** generates about *one terabyte* of new trade data per day.

Social Media:

The statistic shows that *500+terabytes* of new data get ingested into the databases of social media site **Facebook**, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

Jet engine

A single **Jet engine** can generate *10+terabytes* of data in *30 minutes* of flight time. With many thousand flights per day, generation of data reaches up to many *Petabytes.*

Types Of Big Data:

Following are the types of Big Data:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

Structured:

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

Examples of Structured Data

An 'Employee' table in a database is an example of Structured Data

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000
7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000

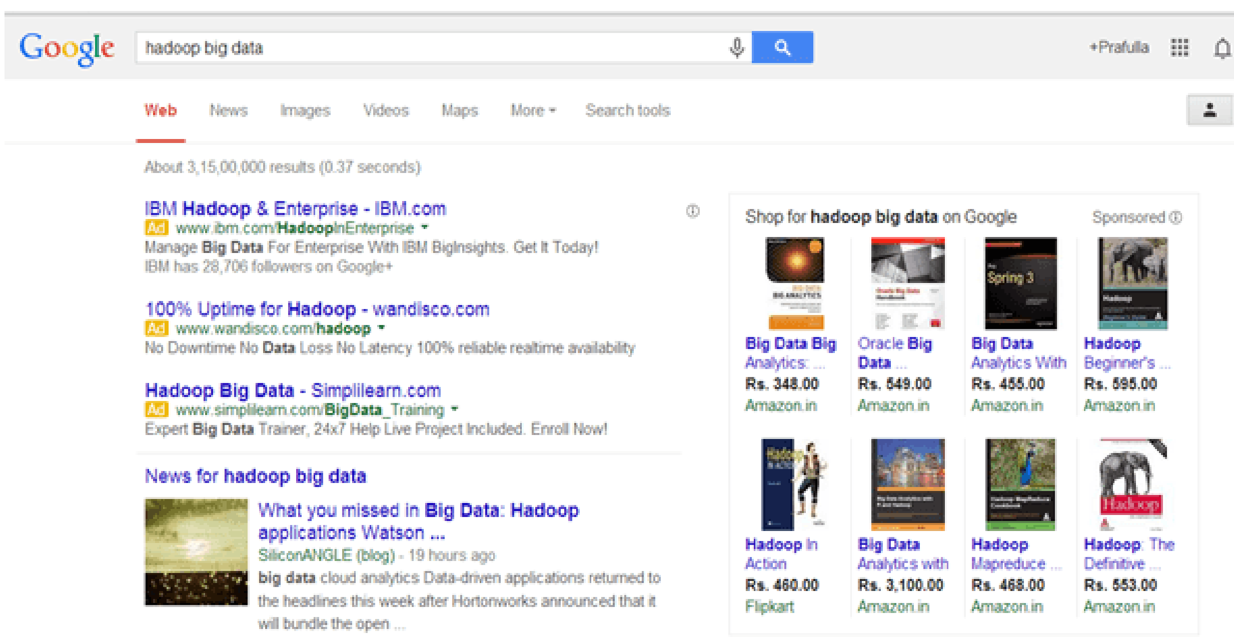
7699	Priya Sane	Female	Finance	550000

Unstructured:

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Examples of Un-structured Data:

The output returned by 'Google Search'



Semi-structured:

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in an XML file.

Examples Of Semi-structured Data

Personal data stored in an XML file-

```
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>
```

Characteristics of Big Data:

Big data can be described by the following characteristics:

- **Volume**
- **Variety**
- **Velocity**
- **Variability**

(i) Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, **'Volume'** is one characteristic which needs to be considered while dealing with Big Data.

***(ii) Variety* –** Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.

***(iii) Velocity* –** The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data. Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.

(iv) Variability **–** This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Benefits of Big Data Processing:

Ability to process Big Data brings in multiple benefits, such as-

- Businesses can utilize outside intelligence while taking decisions
- Improved customer service
- Early identification of risk to the product/services, if any
- Better operational efficiency

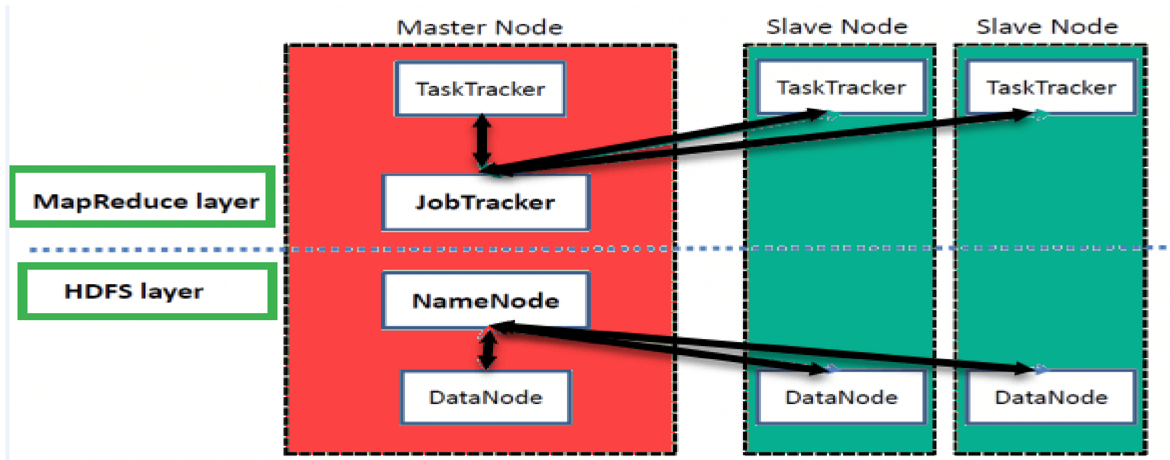
Apache Hadoop:

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in Hadoop, data resides in a distributed file system which is called as a **Hadoop Distributed File system**. The processing model is based on **'Data Locality'** concept wherein computational logic is sent to cluster nodes(server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

Hadoop Architecture:



High Level Hadoop Architecture

Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.

NameNode:
NameNode represented every files and directory which is used in the namespace
DataNode:
DataNode helps you to manage the state of an HDFS node and allows you to interacts with the blocks
MasterNode:
The master node allows you to conduct parallel processing of data using Hadoop MapReduce.
Slave node:
The slave nodes are the additional machines in the Hadoop cluster which allows you to store data to conduct complex calculations. Moreover, all the slave node comes with Task Tracker and a DataNode. This allows you to synchronize the processes with the NameNode and Job Tracker respectively.

In Hadoop, master or slave system can be set up in the cloud or on-premise

Features Of Hadoop:

- **Suitable for Big Data Analysis**

As Big Data tends to be distributed and unstructured in nature, HADOOP clusters are best suited for analysis of Big Data. Since it is processing logic (not the actual data) that flows to the computing nodes, less network bandwidth is consumed. This concept is called as **data locality concept** which helps increase the efficiency of Hadoop based applications.

• **Scalability**

HADOOP clusters can easily be scaled to any extent by adding additional cluster nodes and thus allows for the growth of Big Data. Also, scaling does not require modifications to application logic.

• **Fault Tolerance**

HADOOP ecosystem has a provision to replicate the input data on to other cluster nodes. That way, in the event of a cluster node failure, data processing can still proceed by using data stored on another cluster node.

Conclusion:

Sample Questions:

1) What is Hadoop.

2) Explain Hadoop Architecture.

3) Explain components of Haddop.

EXPERIMENT NO: 01-A

Title: Installation of Hadoop.	
Aim: Study of Installation procedure of Hadoop.	

Theory:

Procedure for Install Hadoop with Step by Step Configuration on Ubuntu:

This is 2 part process

- Part 1) Download and Install Hadoop
- Part 2) Configure Hadoop

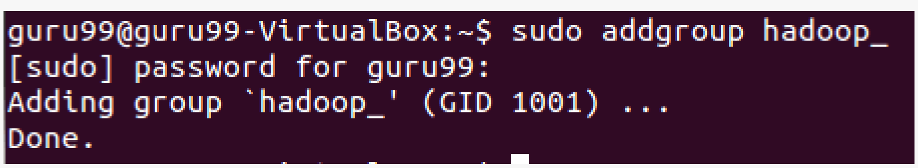
There are 2 **Prerequisites**

- You must have Ubuntu installed and running
- You must have Java Installed.

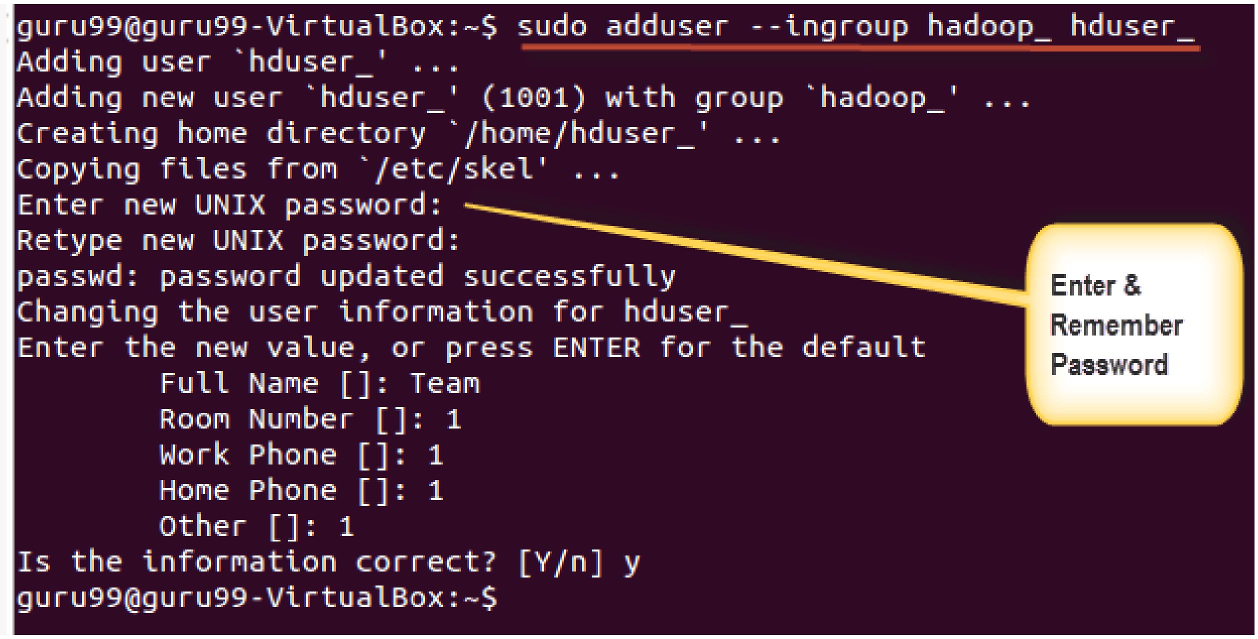
Part 1) Download and Install Hadoop

Step 1) Add a Hadoop system user using below command

sudo addgroup hadoop_



sudo adduser --ingroup hadoop_ hduser_

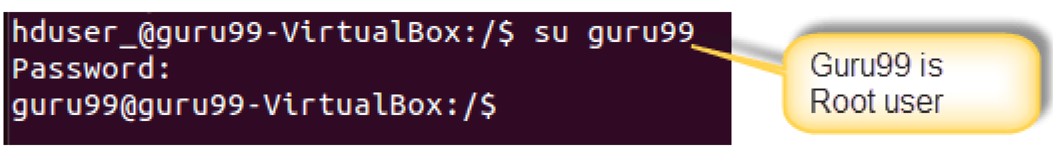


Enter your password, name and other details.

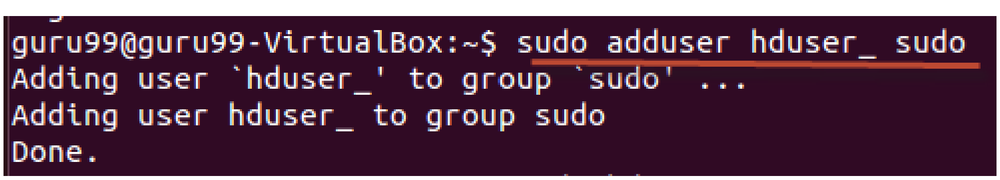
NOTE: There is a possibility of below-mentioned error in this setup and installation process.

"hduser is not in the sudoers file. This incident will be reported."

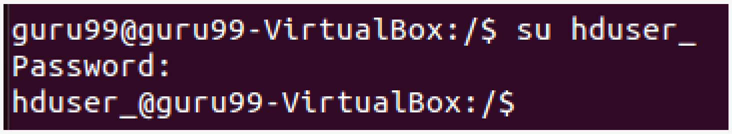
This error can be resolved by Login as a root user



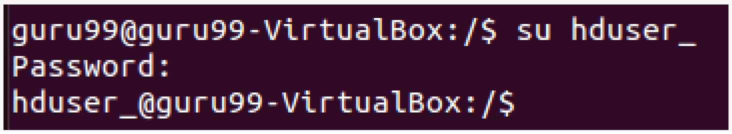
Execute the command



sudo adduser hduser_ sudo



Re-login as hduser_

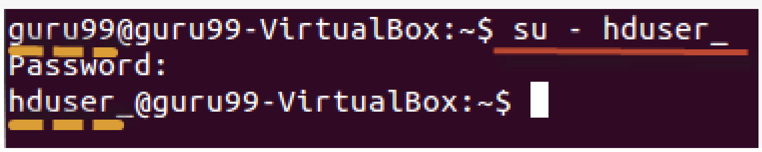


Step 2) Configure SSH

In order to manage nodes in a cluster, Hadoop requires SSH access

First, switch user, enter the following command

su - hduser_



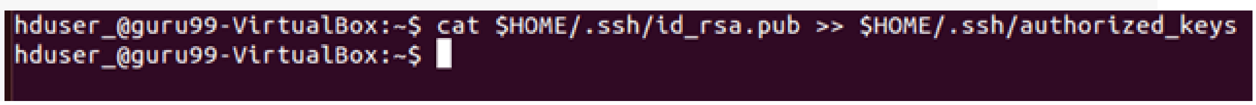
This command will create a new key.

ssh-keygen -t rsa -P ""

```
guru99@guru99-VirtualBox:~$ su - hduser_
Password:
hduser_@guru99-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:
07:e2:3f:7d:7d:d1:0d:9d:12:0e:e7:27:ab:47:4a:22 hduser_@guru99-VirtualBox
The key's randomart image is:
+--[ RSA 2048]----+
|            . o  |
|             = ...|
|      . .    =.o.|
|     . . .    =.o|
|      .ES... o .o|
|       ..oo +.  .|
|        o .o... .|
|         . ..  . |
|                 |
+-----------------+
hduser_@guru99-VirtualBox:~$
```


Enable SSH access to local machine using this key.

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



```
hduser_@guru99-VirtualBox:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
hduser_@guru99-VirtualBox:~$
```

Now test SSH setup by connecting to localhost as 'hduser' user.

```
ssh localhost
```

```
hduser_@guru99-VirtualBox:/$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is 4a:78:f7:93:32:0a:c1:b4:24:e2:a6:78:d7:cb:20:d6.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-29-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

hduser_@guru99-VirtualBox:~$
```

Note: Please note, if you see below error in response to 'ssh localhost', then there is a possibility that SSH is not available on this system-

```
hduser@guru99: ~
hduser@guru99:~$ ssh localhost
ssh: connect to host localhost port 22: Connection refused
hduser@guru99:~$
```

To resolve this -

Purge SSH using,

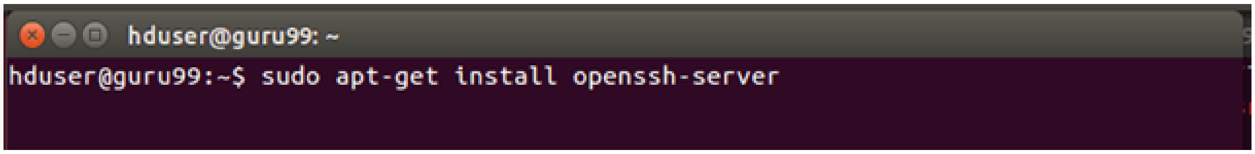
```
sudo apt-get purge openssh-server
```

It is good practice to purge before the start of installation

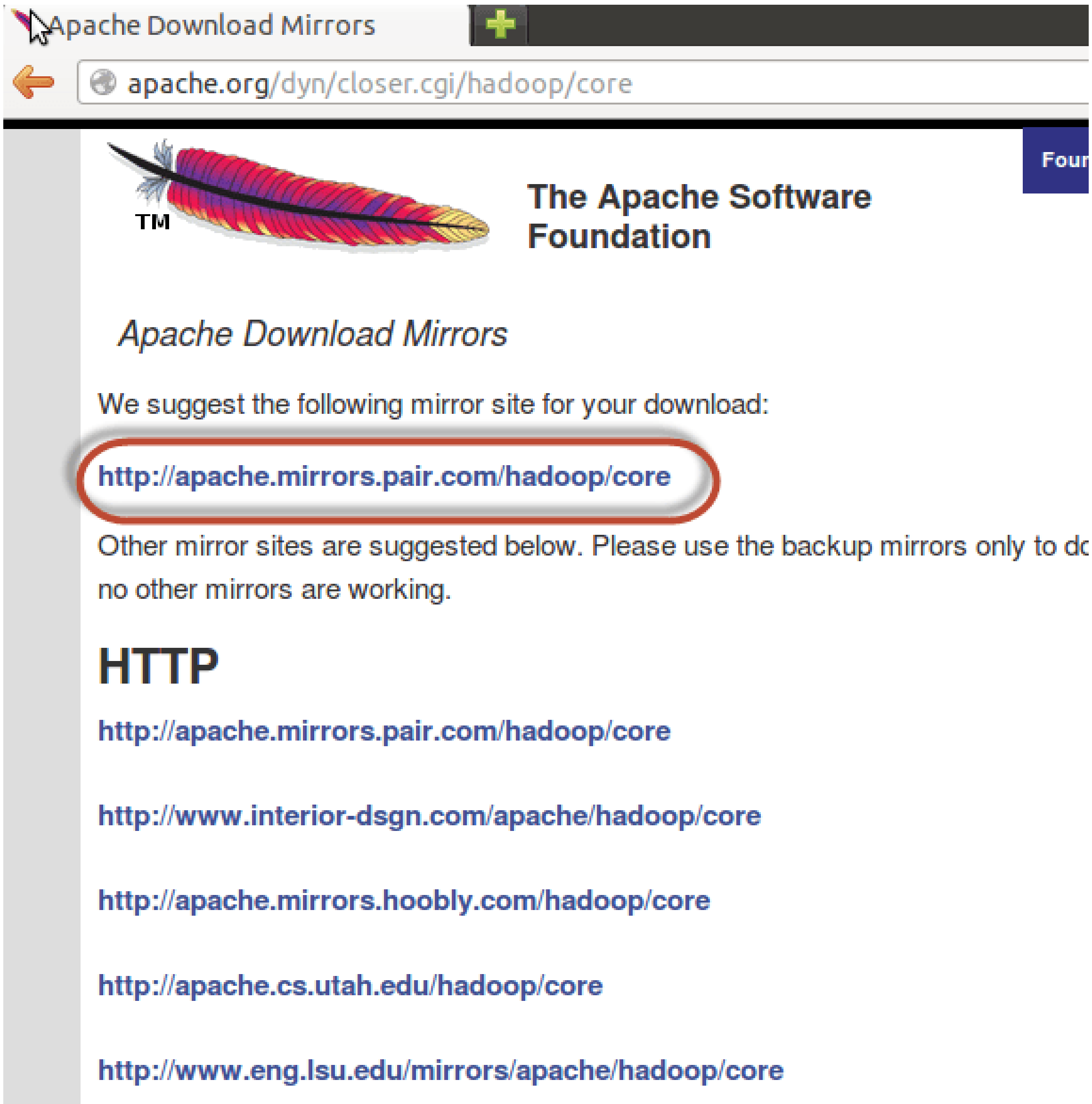
```
hduser@guru99:~$ sudo apt-get purge openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'openssh-server' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
hduser@guru99:~$
```

Install SSH using the command-

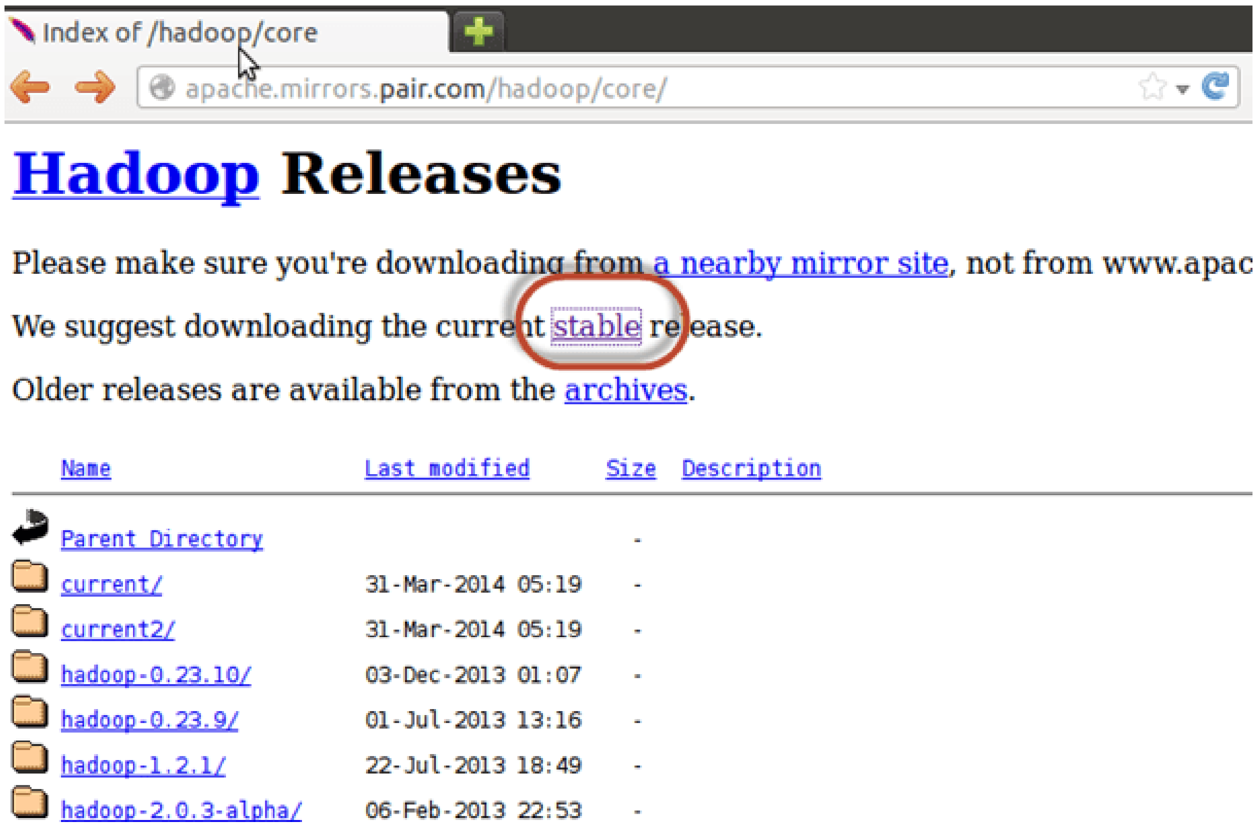
```
sudo apt-get install openssh-server
```



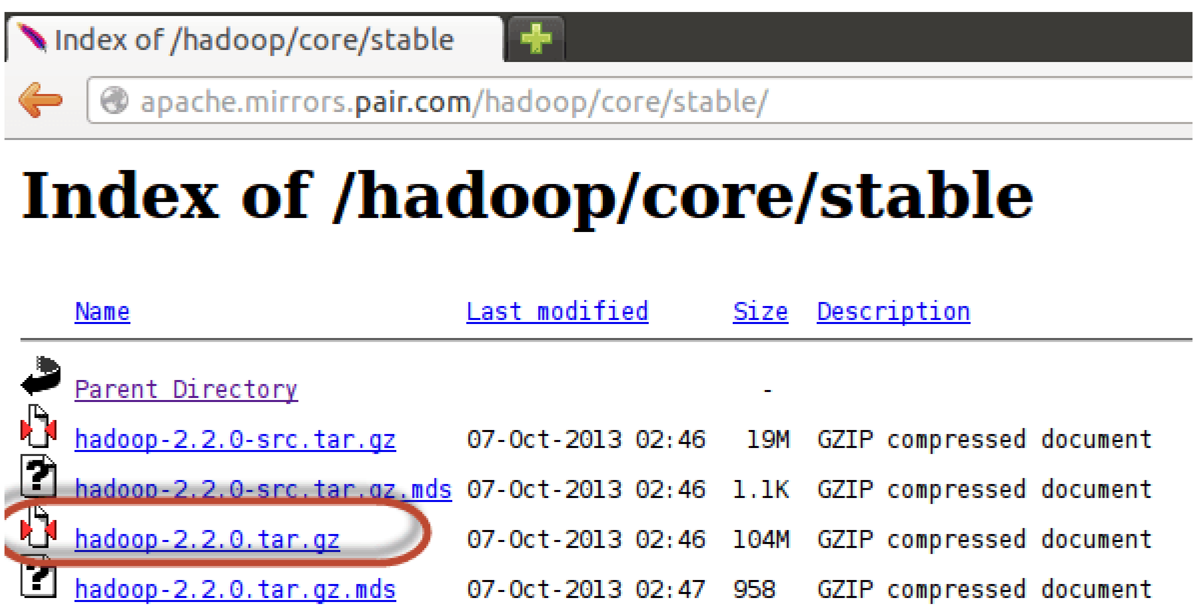
Step 3) Next step is to Download Hadoop



Select Stable



Select the tar.gz file (not the file with src)



Once a download is complete, navigate to the directory containing the tar file

```
hduser_@guru99-VirtualBox:~$ cd /home/guru99/Downloads
```

Enter,

```
sudo tar xzf hadoop-2.2.0.tar.gz
```

```
hduser_@guru99-VirtualBox:/home/guru99/Downloads$ sudo tar -xvf hadoop-2.2.0.tar.gz
```

Now, rename hadoop-2.2.0 as hadoop

```
sudo mv hadoop-2.2.0 hadoop
```

```
hduser_@guru99-VirtualBox:/home/guru99/Downloads$ sudo mv hadoop-2.2.0 hadoop
hduser_@guru99-VirtualBox:/home/guru99/Downloads$
```

```
sudo chown -R hduser_:hadoop_ hadoop
```

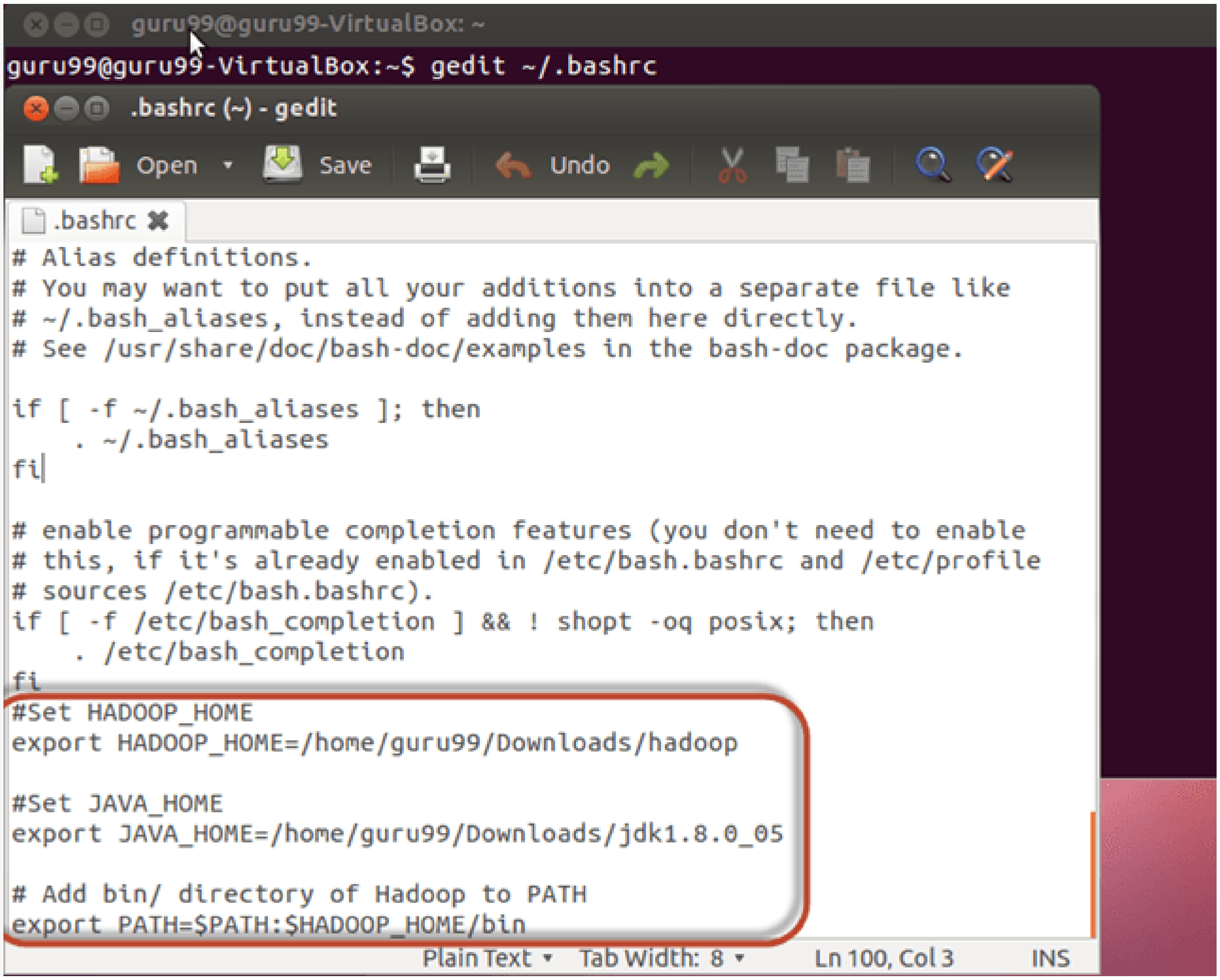
```
hduser_@guru99-VirtualBox:/home/guru99/Downloads$ sudo chown -R hduser_:hadoop_ hadoop
hduser_@guru99-VirtualBox:/home/guru99/Downloads$
```

Part 2) Configure Hadoop

Step 1) Modify **~/.bashrc** file

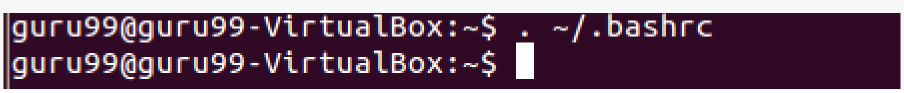
Add following lines to end of file **~/.bashrc**

```
#Set HADOOP_HOME
export HADOOP_HOME=<Installation Directory of Hadoop>
#Set JAVA_HOME
export JAVA_HOME=<Installation Directory of Java>
# Add bin/ directory of Hadoop to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```



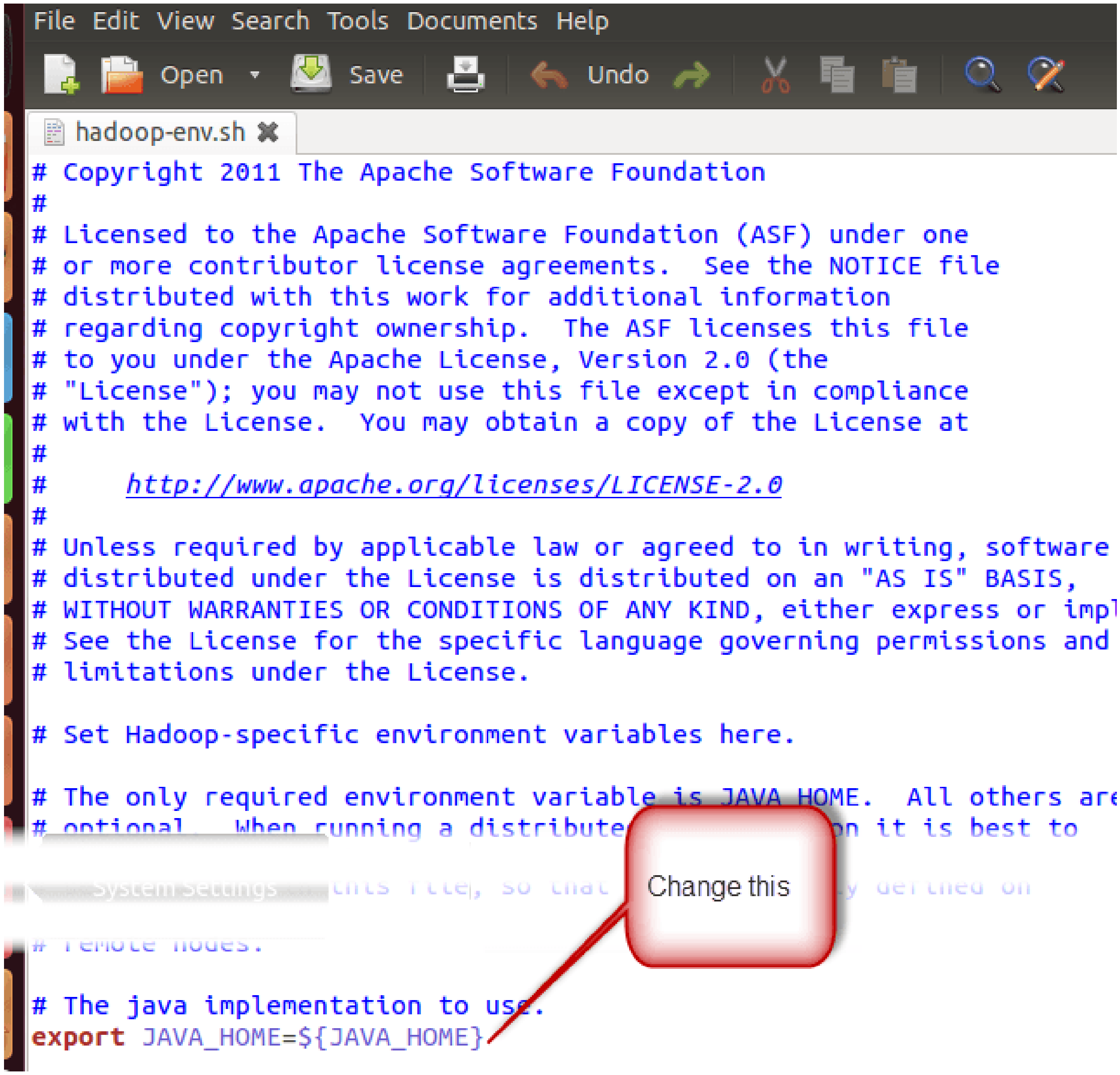
Now, source this environment configuration using below command

```
. ~/.bashrc
```

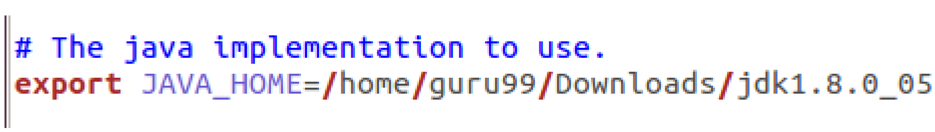


Step 2) Configurations related to HDFS

Set **JAVA_HOME** inside file **$HADOOP_HOME/etc/hadoop/hadoop-env.sh**



With



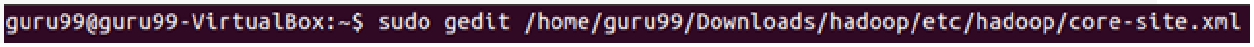
There are two parameters in **$HADOOP_HOME/etc/hadoop/core-site.xml** which need to be set-

1. 'hadoop.tmp.dir' - Used to specify a directory which will be used by Hadoop to store its data files.

2. 'fs.default.name' - This specifies the default file system.

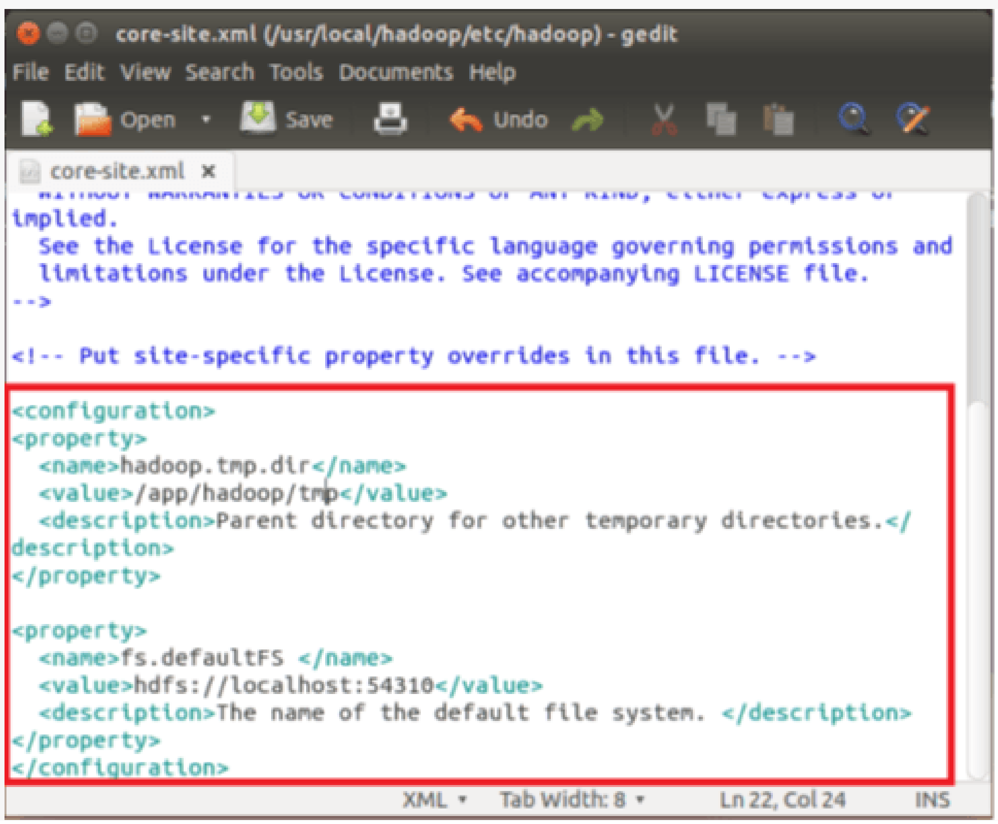
To set these parameters, open core-site.xml

```
sudo gedit $HADOOP_HOME/etc/hadoop/core-site.xml
```

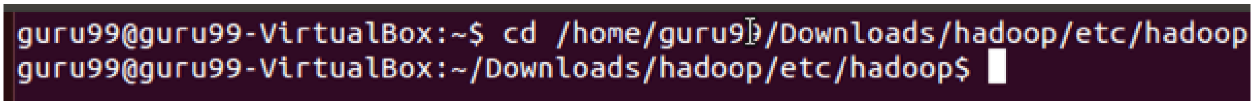


Copy below line in between tags <configuration></configuration>

```
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>Parent directory for other temporary directories.</description>
</property>
<property>
<name>fs.defaultFS </name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. </description>
</property>
```

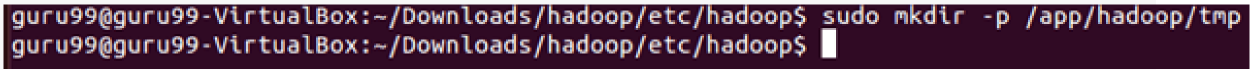


Navigate to the directory **$HADOOP_HOME/etc/Hadoop**



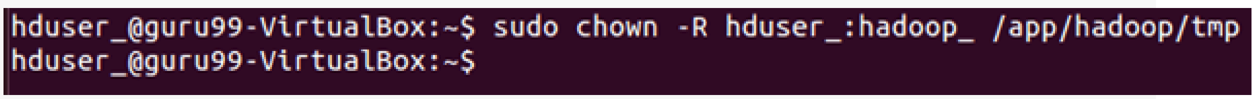
Now, create the directory mentioned in core-site.xml

sudo mkdir -p <Path of Directory used in above setting>

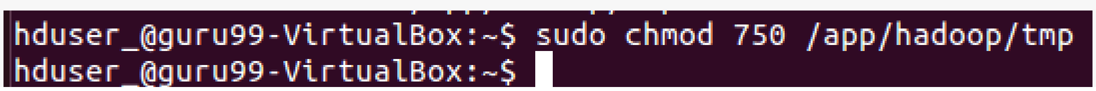


Grant permissions to the directory

sudo chown -R hduser_:Hadoop_ <Path of Directory created in above step>



sudo chmod 750 <Path of Directory created in above step>



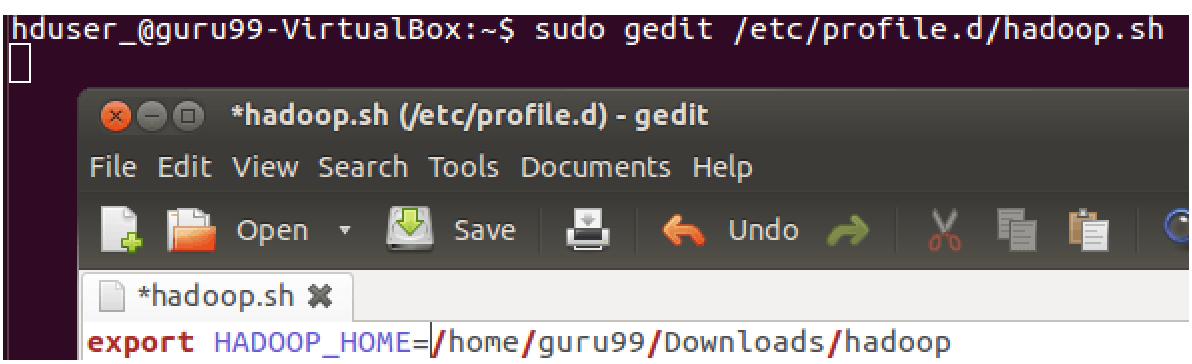
Step 3) Map Reduce Configuration

Before you begin with these configurations, lets set HADOOP_HOME path

sudo gedit /etc/profile.d/hadoop.sh

And Enter

export HADOOP_HOME=/home/guru99/Downloads/Hadoop

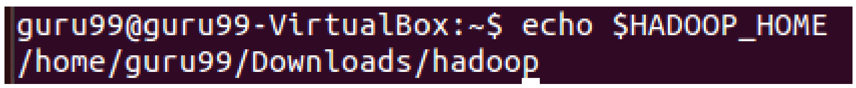


Next enter

```
sudo chmod +x /etc/profile.d/hadoop.sh
```

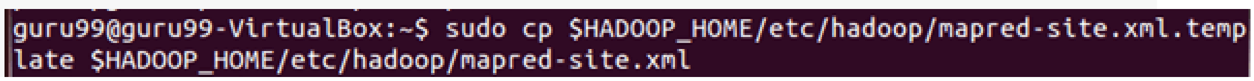
Exit the Terminal and restart again

Type echo $HADOOP_HOME. To verify the path



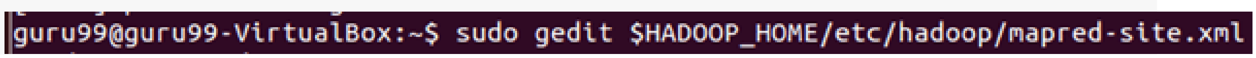
Now copy files

```
sudo cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template $HADO
OP_HOME/etc/hadoop/mapred-site.xml
```



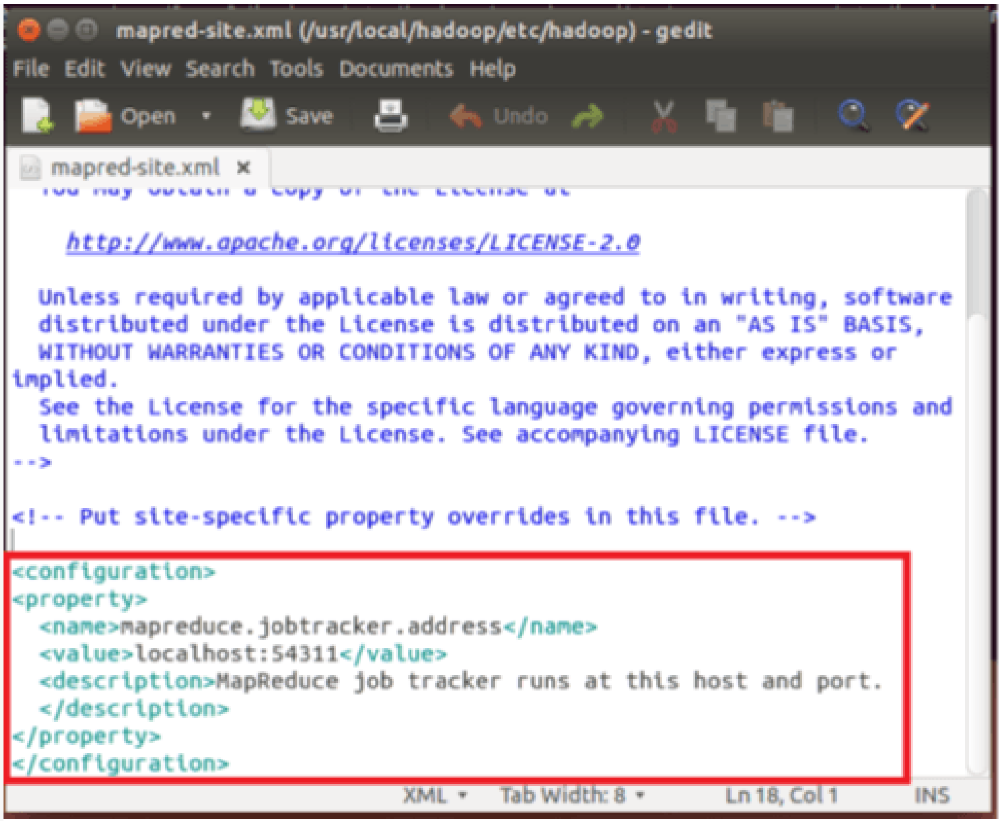
Open the **mapred-site.xml** file

```
sudo gedit $HADOOP_HOME/etc/hadoop/mapred-site.xml
```



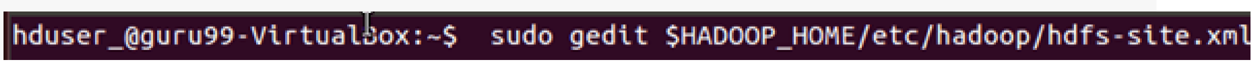
Add below lines of setting in between tags <configuration> and </configuration>

```
<property>
<name>mapreduce.jobtracker.address</name>
<value>localhost:54311</value>
<description>MapReduce job tracker runs at this host and port.
</description>
</property>
```



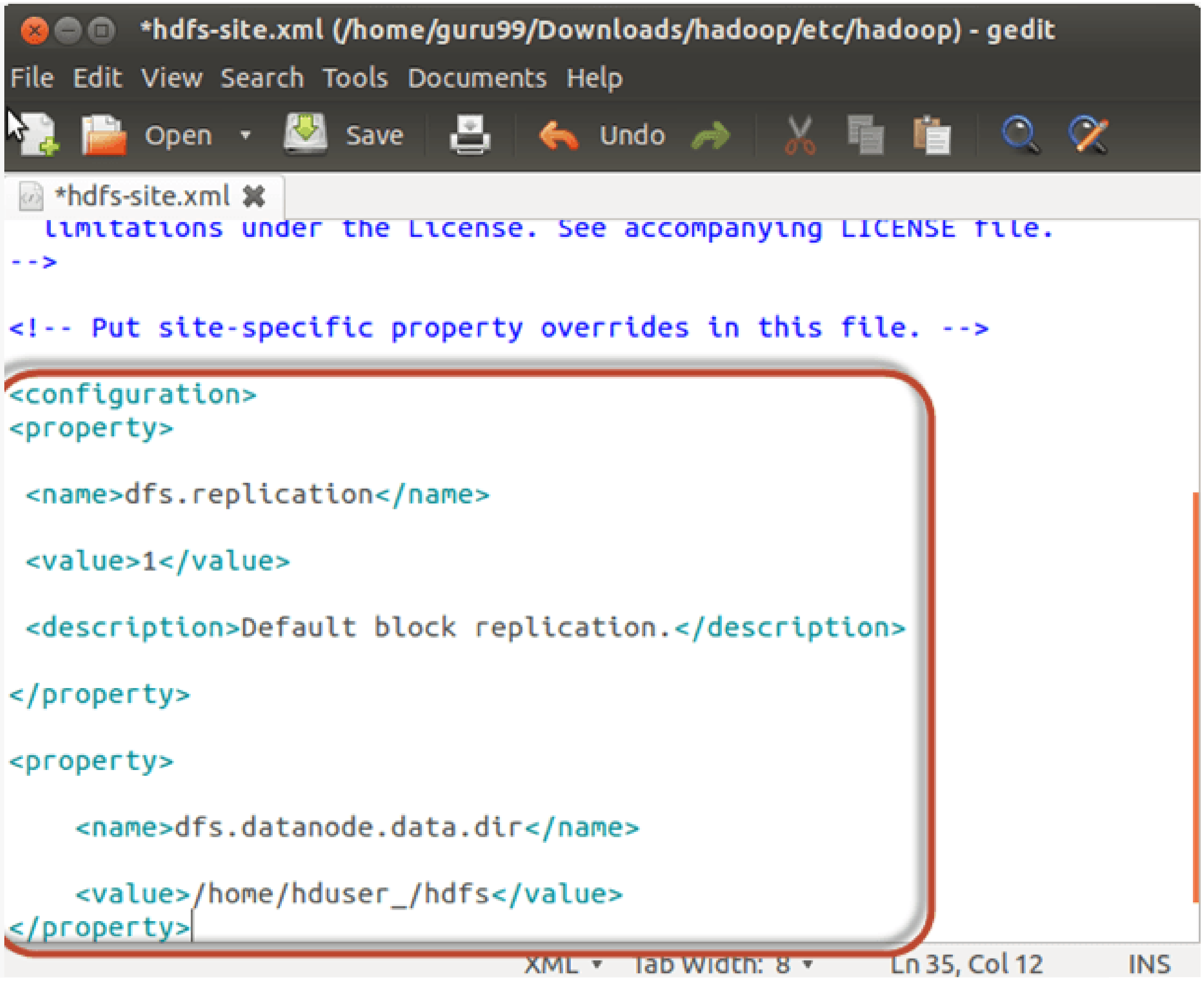
Open **$HADOOP_HOME/etc/hadoop/hdfs-site.xml** as below,

```
sudo gedit $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```



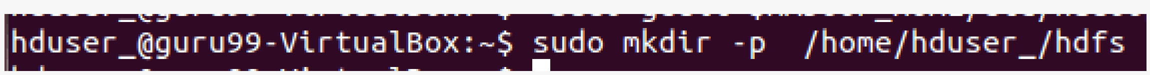
Add below lines of setting between tags <configuration> and </configuration>

```
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication.</description>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/home/hduser_/hdfs</value>
</property>
```



Create a directory specified in above setting-

sudo mkdir -p <Path of Directory used in above setting>
sudo mkdir -p /home/hduser_/hdfs



sudo chown -R hduser_:hadoop_ <Path of Directory created in above step>
sudo chown -R hduser_:hadoop_ /home/hduser_/hdfs

sudo chmod 750 <Path of Directory created in above step>
sudo chmod 750 /home/hduser_/hdfs

```
hduser_@guru99-VirtualBox:~$ sudo chmod 750 /home/hduser_/hdfs
```

Step 4) Before we start Hadoop for the first time, format HDFS using below command

```
$HADOOP_HOME/bin/hdfs namenode –format
```

```
hduser_@guru99-VirtualBox:~$  $HADOOP_HOME/bin/hdfs namenode -format
14/05/05 13:01:58 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = guru99-VirtualBox/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.2.0
STARTUP_MSG:   classpath = /home/guru99/Downloads/hadoop/etc/hadoop:/home/guru99
/Downloads/hadoop/share/hadoop/common/lib/activation-1.1.jar:/home/guru99/Downlo
ads/hadoop/share/hadoop/common/lib/netty-3.6.2.Final.jar:/home/guru99/Downloads/
hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/home/guru99/Downloads/ha
doop/share/hadoop/common/lib/xmlenc-0.52.jar:/home/guru99/Downloads/hadoop/share
/hadoop/common/lib/jsp-api-2.1.jar:/home/guru99/Downloads/hadoop/share/hadoop/co
mmon/lib/commons-collections-3.2.1.jar:/home/guru99/Downloads/hadoop/share/hadoo
p/common/lib/avro-1.7.4.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/li
b/jackson-core-asl-1.8.8.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/l
ib/commons-io-2.1.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/lib/jers
ey-core-1.9.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/lib/commons-co
dec-1.4.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/lib/mockito-all-1.
8.5.jar:/home/guru99/Downloads/hadoop/share/hadoop/common/lib/commons-cli-1.2.ja
r:/home/guru99/Downloads/hadoop/share/hadoop/common/lib/jets3t-0.6.1.jar:/home/g
uru99/Downloads/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/home/guru99/Dow
nloads/hadoop/share/hadoop/common/lib/commons-net-3.1.jar:/home/guru99/Downloads
/hadoop/share/hadoop/common/lib/commons-httpclient-3.1.jar:/home/guru99/Download
```

Step 5) Start Hadoop single node cluster using below command

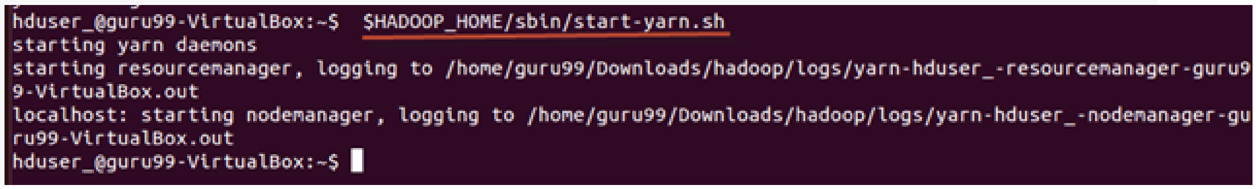
```
$HADOOP_HOME/sbin/start-dfs.sh
```

An output of above command

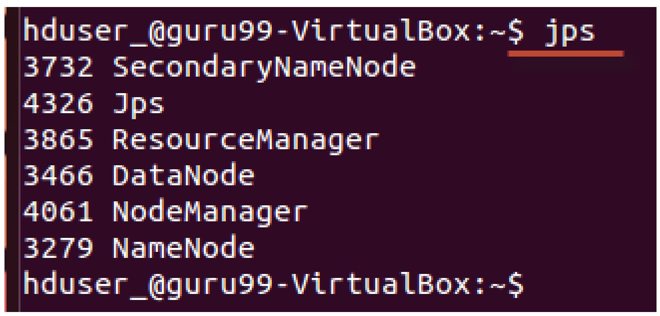
```
@guru99-VirtualBox: ~                                    1:06 PM  guru99
hduser_@guru99-VirtualBox:~$  $HADOOP_HOME/sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/guru99/Downloads/hadoop/logs/hadoop-hduser_-namenode-guru99
-VirtualBox.out
localhost: starting datanode, logging to /home/guru99/Downloads/hadoop/logs/hadoop-hduser_-datanode-guru99
-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is 4a:78:f7:93:32:0a:c1:b4:24:e2:a6:78:d7:cb:20:d6.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /home/guru99/Downloads/hadoop/logs/hadoop-hduser_-secondar
ynamenode-guru99-VirtualBox.out
hduser_@guru99-VirtualBox:~$
```

Enter Yes

$HADOOP_HOME/sbin/start-yarn.sh



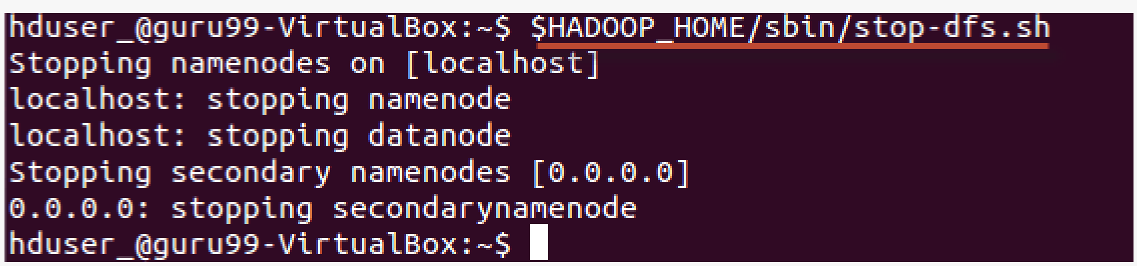
Using **'jps'** tool/command, verify whether all the Hadoop related processes are running or not.



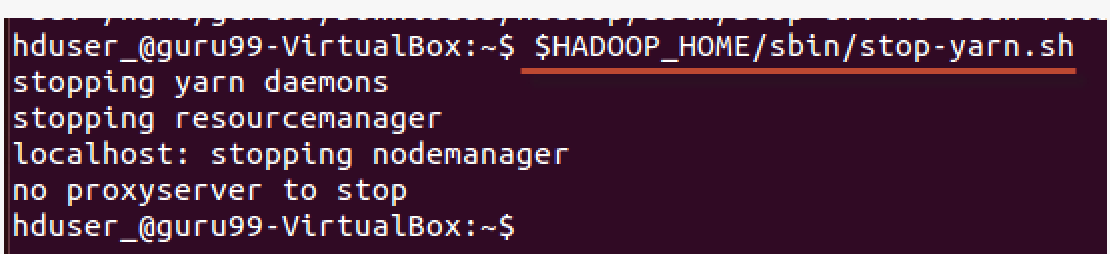
If Hadoop has started successfully then an output of jps should show NameNode, NodeManager, ResourceManager, SecondaryNameNode, DataNode.

Step 6) Stopping Hadoop

$HADOOP_HOME/sbin/stop-dfs.sh



$HADOOP_HOME/sbin/stop-yarn.sh



Conclusion:

Sample Questions:

1) Explain Hadoop distributed file system.

2) Explain Installation steps of Hadoop.

EXPERIMENT NO: 02

Title: Introduction of Hadddop MapReduce.
Aim: Study of Hadoop Mapreduce.

Theory:

MapReduce is a software framework and programming model used for processing huge amounts of data. **MapReduce** program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. The programs of Map Reduce in cloud computing are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

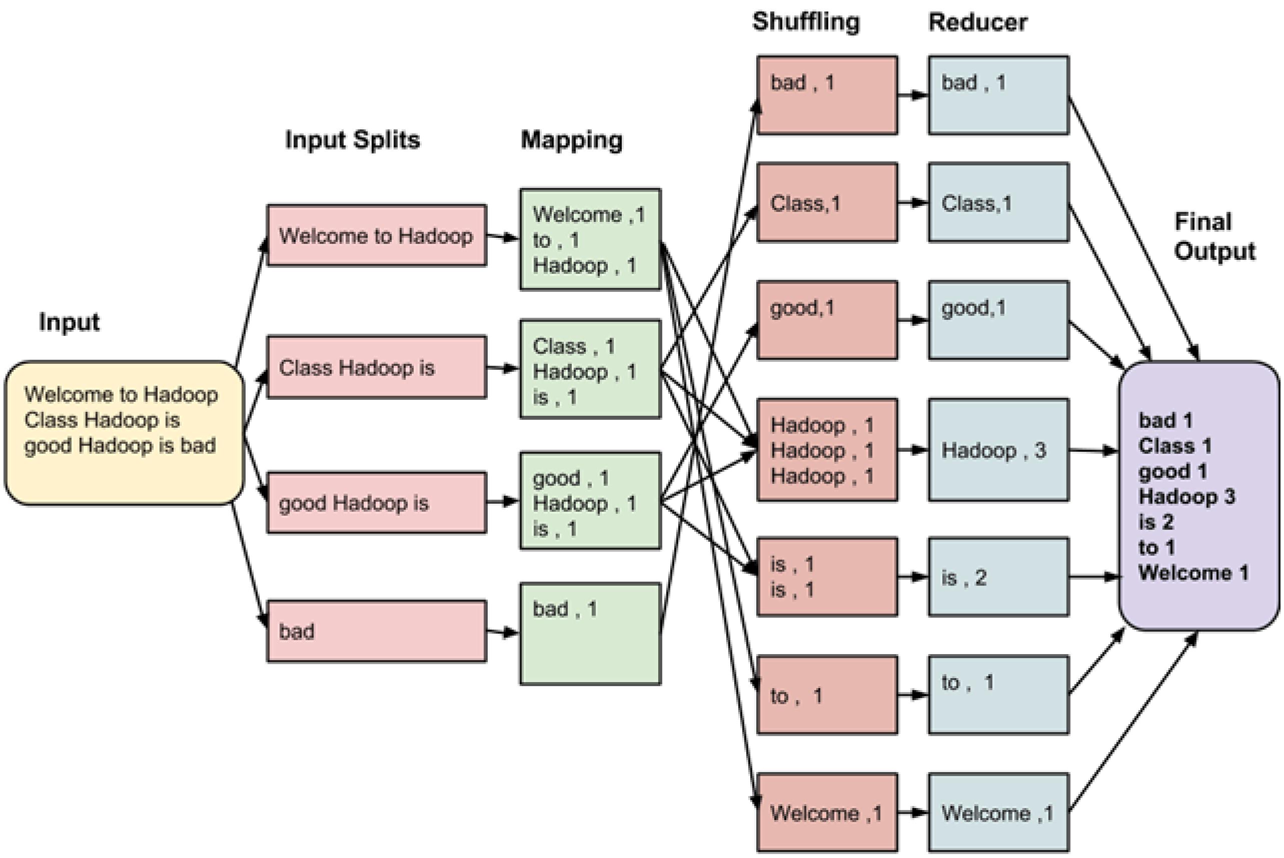
The input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

MapReduce Architecture in Big Data:

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

Now in this MapReduce tutorial, let's understand with a MapReduce example–

Consider you have following input data for your MapReduce in Big data Program



MapReduce Architecture

The final output of the MapReduce task is

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1

Welcome	1

The data goes through the following phases of MapReduce in Big Data

Input Splits:

An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map

Mapping

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

Shuffling

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

Reducing

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

MapReduce Architecture :

- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.

- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
- In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
- Reduce task doesn't work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
- On this machine, the output is merged and then passed to the user-defined reduce function.
- Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

How MapReduce Organizes Work?

Hadoop divides the job into tasks. There are two types of tasks:

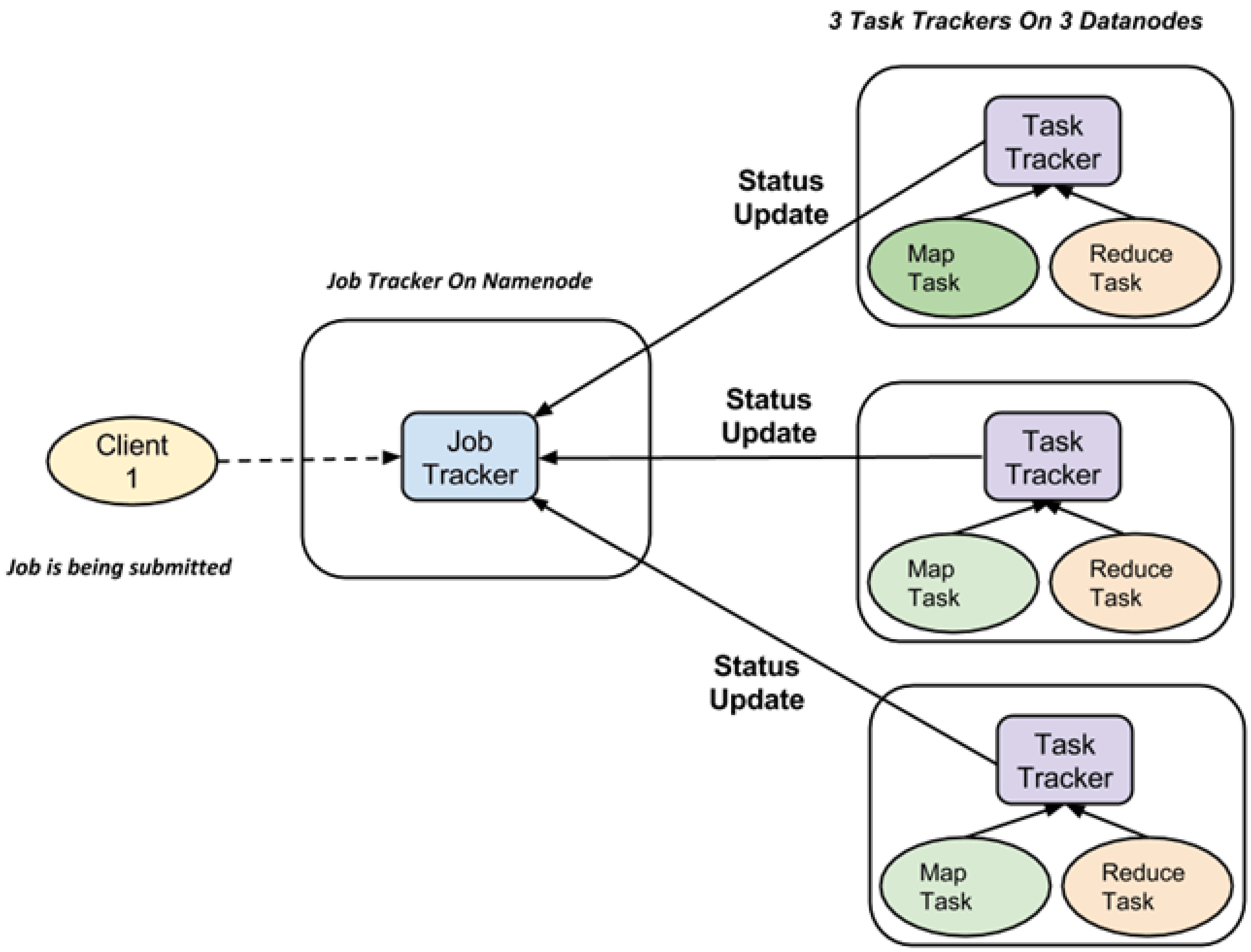
1. **Map tasks** (Splits & Mapping)
2. **Reduce tasks** (Shuffling, Reducing)

as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. **Jobtracker**: Acts like a **master** (responsible for complete execution of submitted job)
2. **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job

For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.



How Hadoop MapReduce Works

- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
- Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
- Task tracker's responsibility is to send the progress report to the job tracker.
- In addition, task tracker periodically sends **'heartbeat'** signal to the Jobtracker so as to notify him of the current state of the system.
- Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Conclusion:

Sample Questions:

1) What is MapReduce.

2) Explain Terminology of MapReduce.

EXPERIMENT NO: 03

Title: Building Hadddop MapReduce application
Aim: Understanding Mapreduce application working.

Theory:

First Hadoop MapReduce Program

	A	B	C	D	E	F	G	H	I	J	K	L
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917

Data of SalesJan2009

Ensure you have Hadoop installed. Before you start with the actual process, change user to 'hduser' (id used while Hadoop configuration, you can switch to the userid used during your Hadoop config).

```
su - hduser_
```

```
guru99@guru99-VirtualBox:~$ su - hduser_
Password:
hduser_@guru99-VirtualBox:~$
```

Step 1)

Create a new directory with name **MapReduceTutorial**

```
hduser_@guru99-VirtualBox:~$ sudo mkdir MapReduceTutorial
```

```
sudo mkdir MapReduceTutorial
```

Give permissions

```
sudo chmod -R 777 MapReduceTutorial
```

```
hduser_@guru99-VirtualBox:~$ sudo chmod -R 777 MapReduceTutorial
```

SalesMapper.java

```
package SalesCountry;


import java.io.IOException;


import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;


public class SalesMapper extends MapReduceBase implements Mapper <LongWritable, Text, T
ext, IntWritable> {
	private final static IntWritable one = new IntWritable(1);


	public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable> o
utput, Reporter reporter) throws IOException {


		String valueString = value.toString();
		String[] SingleCountryData = valueString.split(",");
		output.collect(new Text(SingleCountryData[7]), one);
```

```
        }
}
```

SalesCountryReducer.java

```
package SalesCountry;

import java.io.IOException;
import java.util.*;

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

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWrita
ble, Text, IntWritable> {

        public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Text,Int
Writable> output, Reporter reporter) throws IOException {
                Text key = t_key;
                int frequencyForCountry = 0;
                while (values.hasNext()) {
                        // replace type of value with the actual type of our value
                        IntWritable value = (IntWritable) values.next();
                        frequencyForCountry += value.get();

                }
```

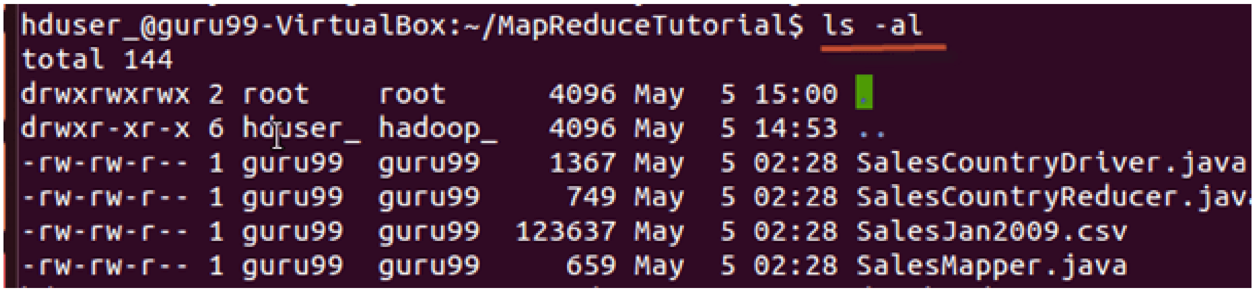
```
                    output.collect(key, new IntWritable(frequencyForCountry));
          }
}
```

SalesCountryDriver.java

```
package SalesCountry;


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


public class SalesCountryDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        // Create a configuration object for the job
        JobConf job_conf = new JobConf(SalesCountryDriver.class);


        // Set a name of the Job
        job_conf.setJobName("SalePerCountry");


        // Specify data type of output key and value
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);


        // Specify names of Mapper and Reducer Class
```

```
        job_conf.setMapperClass(SalesCountry.SalesMapper.class);
        job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

        // Specify formats of the data type of Input and output
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);

        // Set input and output directories using command line arguments,
        //arg[0] = name of input directory on HDFS, and arg[1] =  name of output directory to be cr
eated to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Check the file permissions of all these files

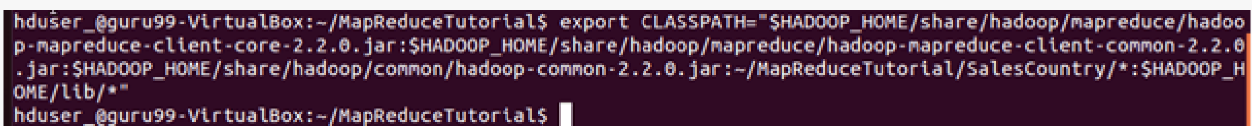


and if 'read' permissions are missing then grant the same-

Step 2)

Export classpath

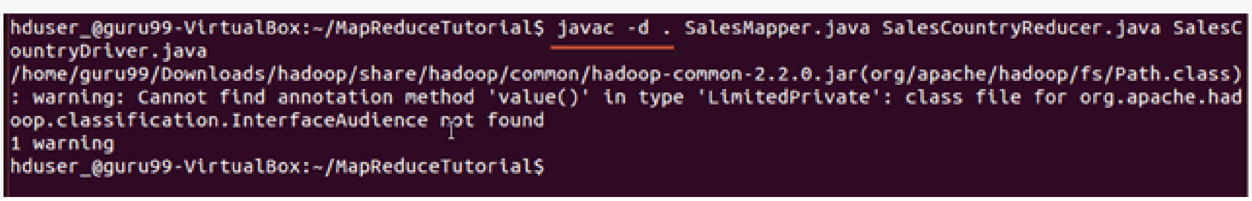
```
export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_HOME/lib/*"
```



Step 3)

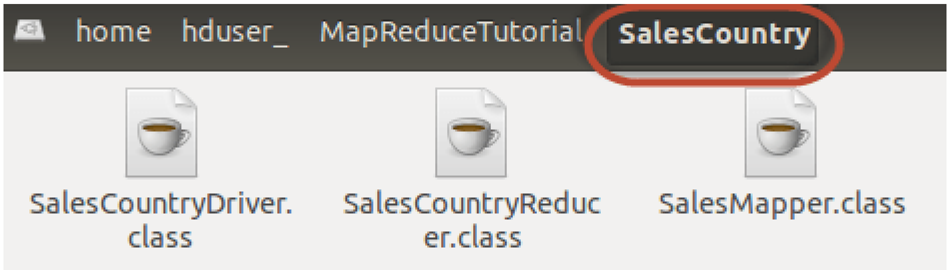
Compile Java files (these files are present in directory **Final-MapReduceHandsOn**). Its class files will be put in the package directory

```
javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java
```



This warning can be safely ignored.

This compilation will create a directory in a current directory named with package name specified in the java source file (i.e. **SalesCountry** in our case) and put all compiled class files in it.



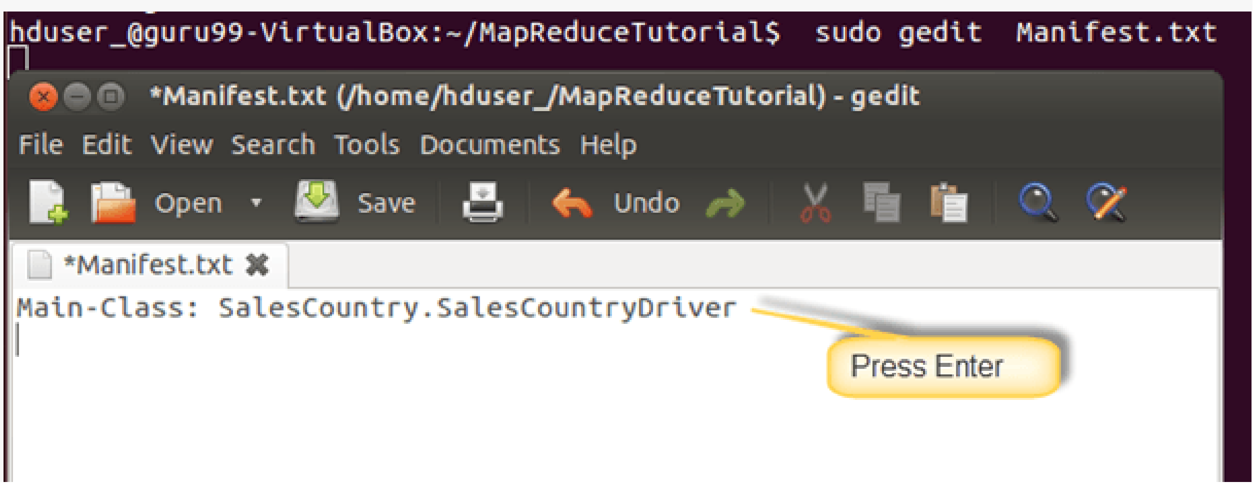
Step 4)

Create a new file **Manifest.txt**

```
sudo gedit Manifest.txt
```

add following lines to it,

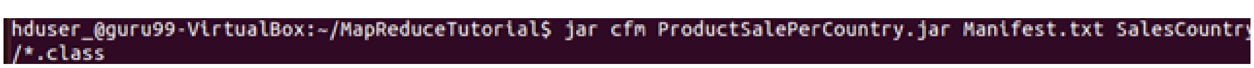
```
Main-Class: SalesCountry.SalesCountryDriver
```



SalesCountry.SalesCountryDriver is the name of main class. Please note that you have to hit enter key at end of this line.

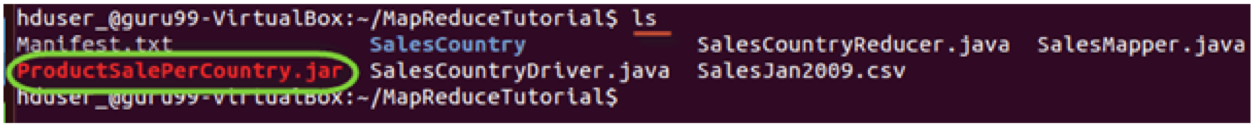
Step 5)

Create a Jar file



```
jar cfm ProductSalePerCountry.jar Manifest.txt SalesCountry/*.class
```

Check that the jar file is created



Step 6)

Start Hadoop

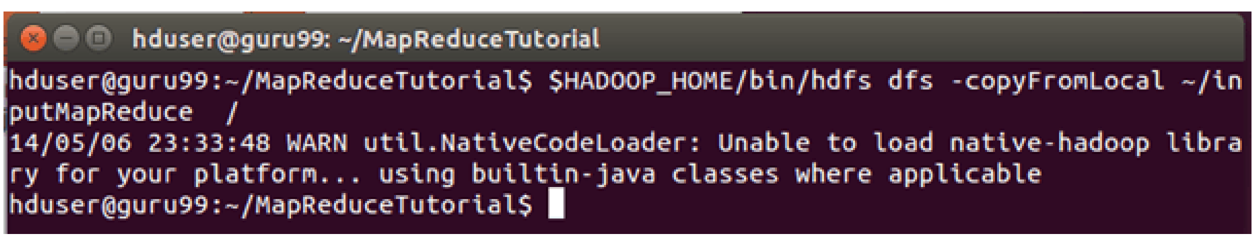
```
$HADOOP_HOME/sbin/start-dfs.sh

$HADOOP_HOME/sbin/start-yarn.sh
```

Step 7)

Copy the File **SalesJan2009.csv** into **~/inputMapReduce**

Now Use below command to copy **~/inputMapReduce** to HDFS.

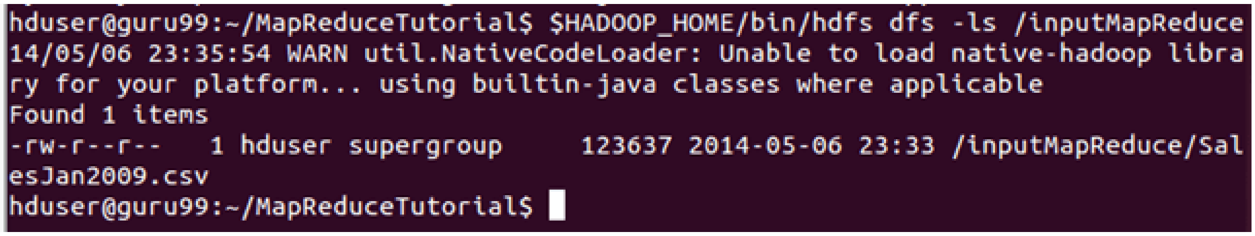
```
$HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/inputMapReduce /
```



We can safely ignore this warning.

Verify whether a file is actually copied or not.

```
$HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce
```



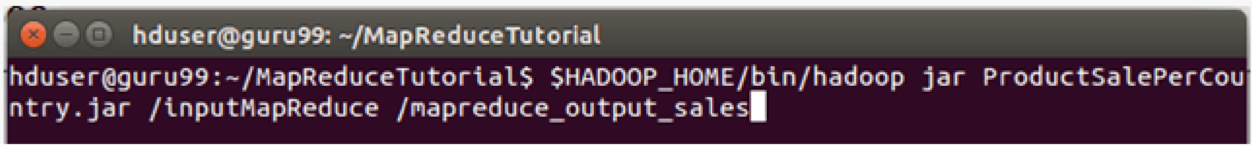
```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce
14/05/06 23:35:54 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 hduser supergroup     123637 2014-05-06 23:33 /inputMapReduce/Sal
esJan2009.csv
hduser@guru99:~/MapReduceTutorial$
```

Step 8)

Run MapReduce job

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_o

utput_sales



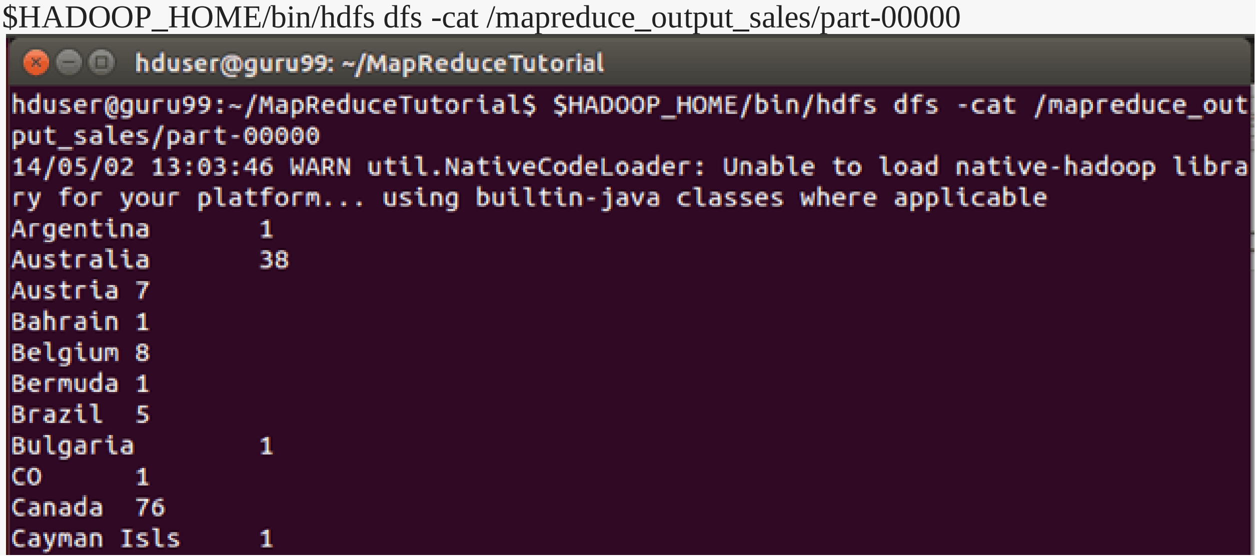
```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hadoop jar ProductSalePerCou
ntry.jar /inputMapReduce /mapreduce_output_sales
```

This will create an output directory named mapreduce_output_sales on HDFS. Contents of this directory will be a file containing product sales per country.

Step 9)

The result can be seen through command interface as,

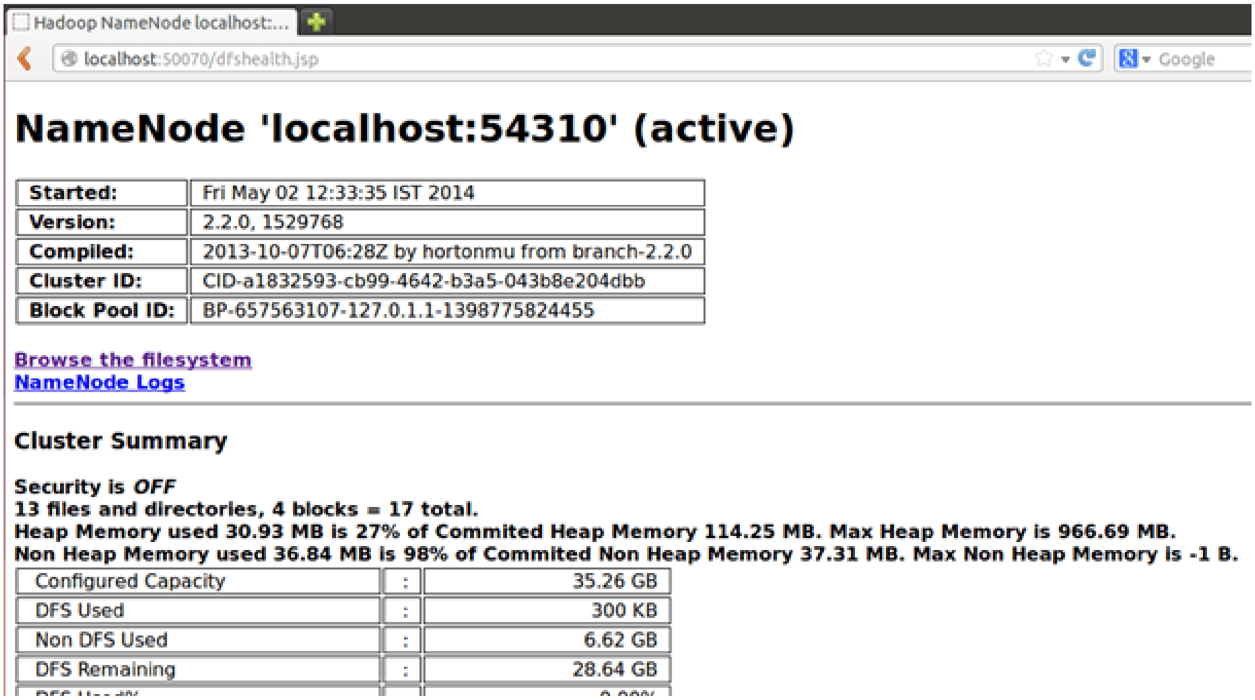
$HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_output_sales/part-00000



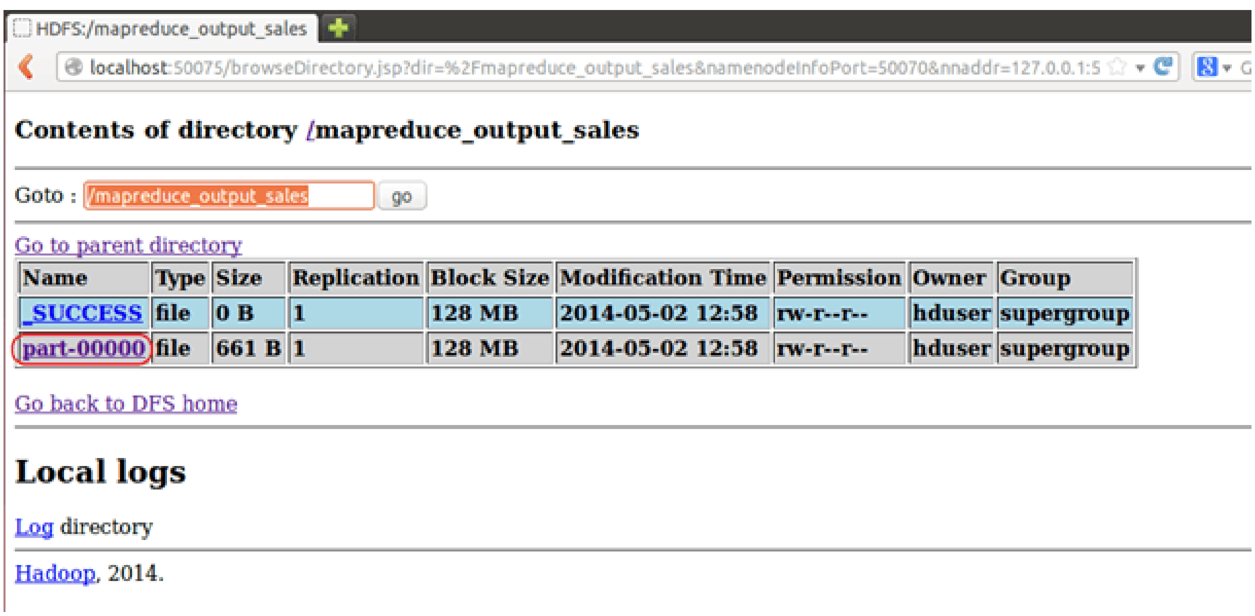
```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_out
put_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
```

Results can also be seen via a web interface as-

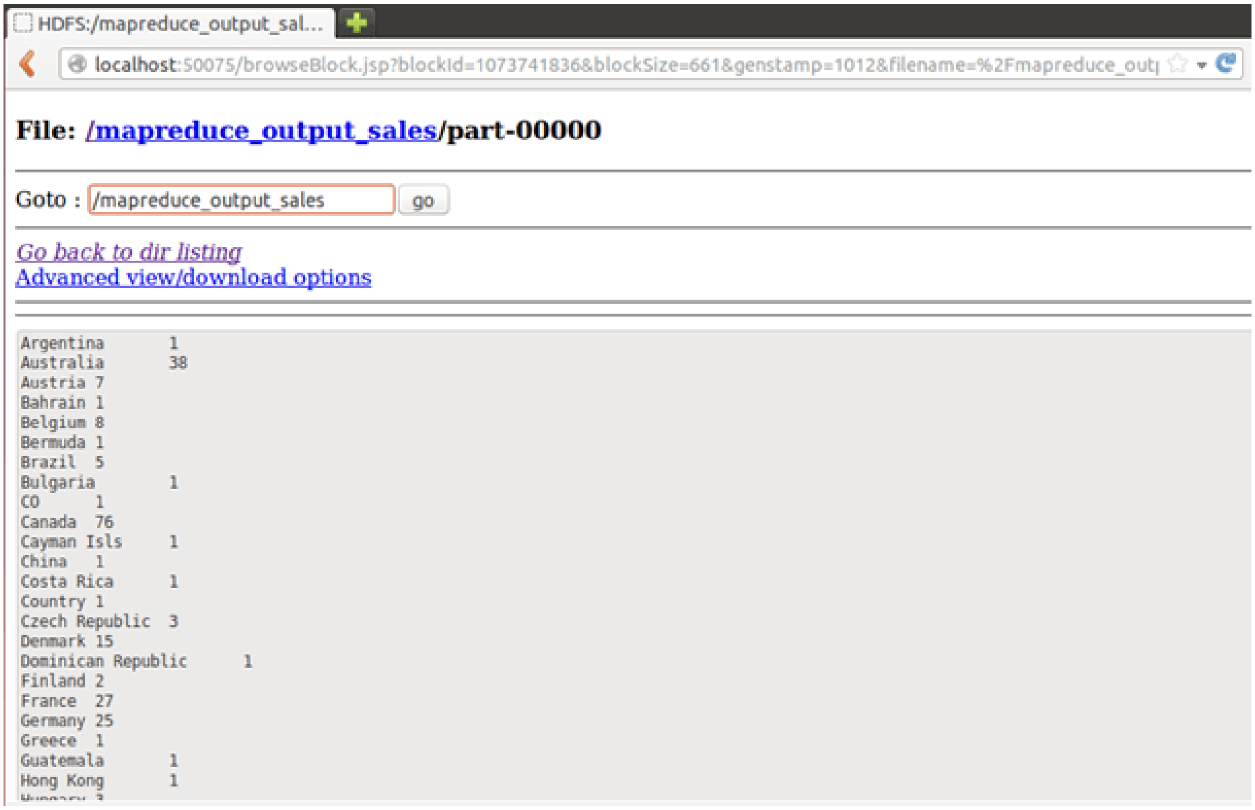
Open r in a web browser.



Now select **'Browse the filesystem'** and navigate to **/mapreduce_output_sales**



Open **part-r-00000**



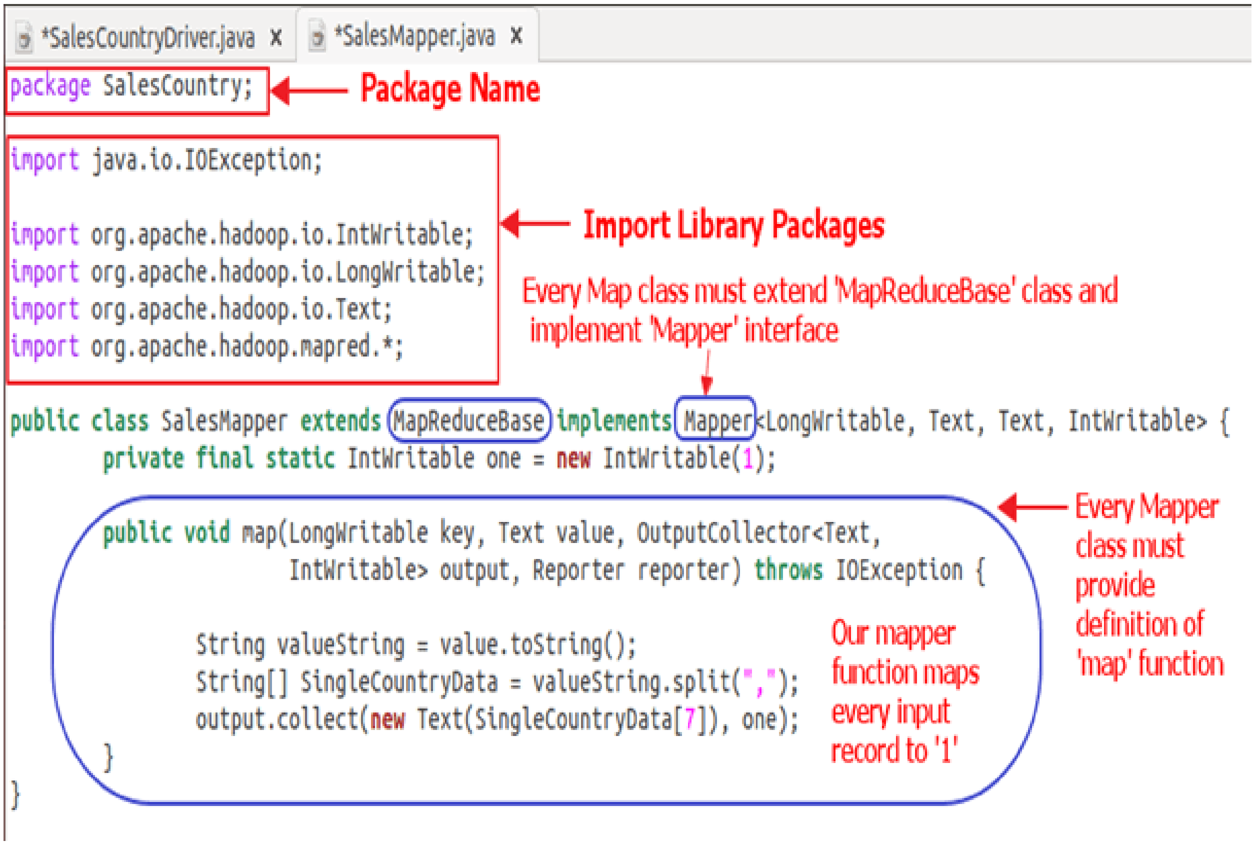
Explanation of SalesMapper Class

In this section, we will understand the implementation of **SalesMapper** class.

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of our package. Please note that output of compilation, **SalesMapper.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesMapper** class-



Sample Code Explanation:

1. SalesMapper Class Definition-

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

Every mapper class must be extended from **MapReduceBase** class and it must implement **Mapper** interface.

2. Defining 'map' function-

```
public void map(LongWritable key,

        Text value,

OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException
```

The main part of Mapper class is a **'map()'** method which accepts four arguments.

At every call to **'map()'** method, a **key-value** pair (**'key'** and **'value'** in this code) is passed.

'map()' method begins by splitting input text which is received as an argument. It uses the tokenizer to split these lines into words.

```
String valueString = value.toString();

String[] SingleCountryData = valueString.split(",");
```

Here, **','** is used as a delimiter.

After this, a pair is formed using a record at 7th index of array **'SingleCountryData'** and a value **'1'**.

```
output.collect(new Text(SingleCountryData[7]), one);
```

We are choosing record at 7th index because we need **Country** data and it is located at 7th index in array **'SingleCountryData'**.

Please note that our input data is in the below format (where **Country** is at 7th index, with 0 as a starting index)-

Transaction_date,Product,Price,Payment_Type,Name,City,State,**Country**,Account_Created,Last_Login,Latitude,Longitude

An output of mapper is again a **key-value** pair which is outputted using **'collect()'** method of **'OutputCollector'**.

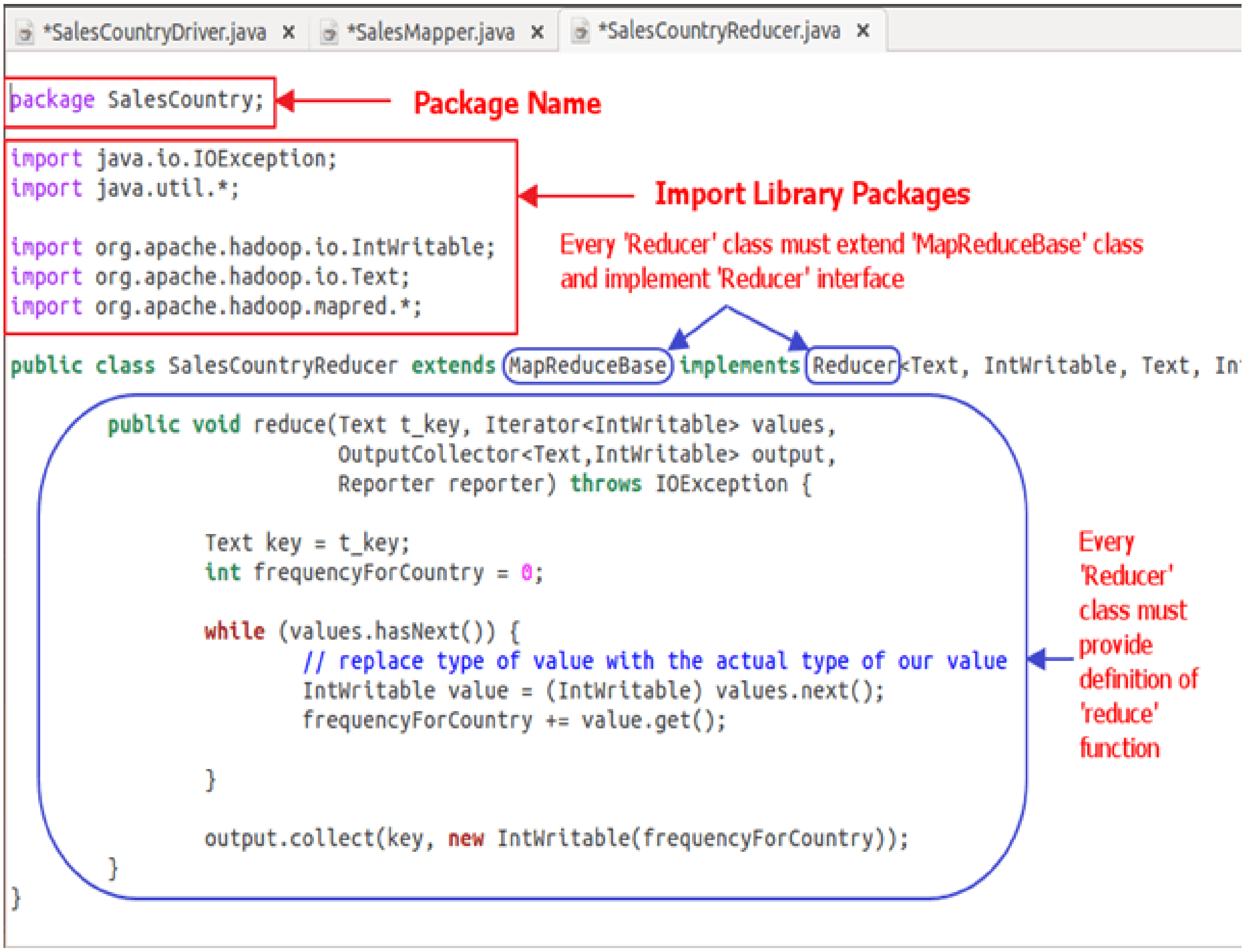
Explanation of SalesCountryReducer Class

In this section, we will understand the implementation of **SalesCountryReducer** class.

1. We begin by specifying a name of the package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryReducer.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesCountryReducer** class-



Code Explanation:

1. SalesCountryReducer Class Definition-

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

Here, the first two data types, **'Text'** and **'IntWritable'** are data type of input key-value to the reducer.

Output of mapper is in the form of <CountryName1, 1>, <CountryName2, 1>. This output of mapper becomes input to the reducer. So, to align with its data type, **Text** and **IntWritable** are used as data type here.

The last two data types, 'Text' and 'IntWritable' are data type of output generated by reducer in the form of key-value pair.

Every reducer class must be extended from **MapReduceBase** class and it must implement **Reducer** interface.

2. Defining 'reduce' function-

```
public void reduce( Text t_key,
            Iterator<IntWritable> values,
            OutputCollector<Text,IntWritable> output,
            Reporter reporter) throws IOException {
```

An input to the **reduce()** method is a key with a list of multiple values.

For example, in our case, it will be-

<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>,<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>.

This is given to reducer as **<United Arab Emirates, {1,1,1,1,1,1}>**

So, to accept arguments of this form, first two data types are used, viz., **Text** and **Iterator<IntWritable>**. **Text** is a data type of key and **Iterator<IntWritable>** is a data type for list of values for that key.

The next argument is of type **OutputCollector<Text,IntWritable>** which collects the output of reducer phase.

reduce() method begins by copying key value and initializing frequency count to 0.

```
Text key = t_key;
int frequencyForCountry = 0;
```

Then, using '**while**' loop, we iterate through the list of values associated with the key and calculate the final frequency by summing up all the values.

```
while (values.hasNext()) {

    // replace type of value with the actual type of our value

    IntWritable value = (IntWritable) values.next();

    frequencyForCountry += value.get();


}
```

Now, we push the result to the output collector in the form of **key** and obtained **frequency count**.

Below code does this-

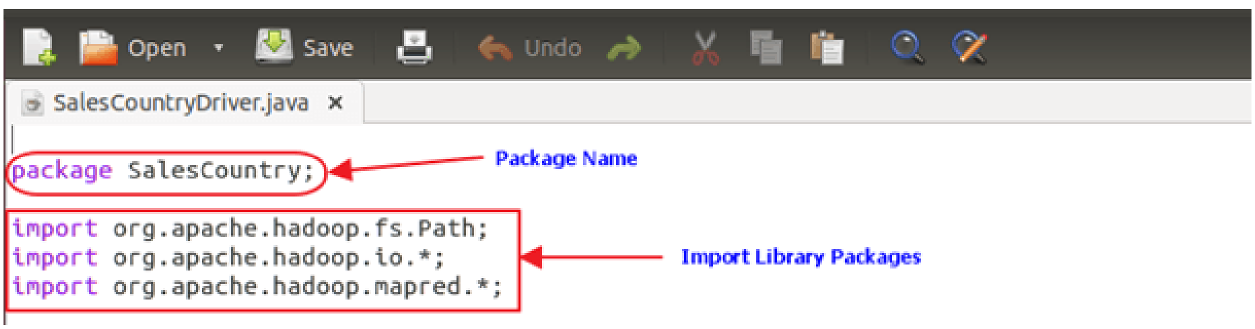
```
output.collect(key, new IntWritable(frequencyForCountry));
```

Explanation of SalesCountryDriver Class

In this section, we will understand the implementation of **SalesCountryDriver** class

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryDriver.class** will go into directory named by this package name: **SalesCountry**.

Here is a line specifying package name followed by code to import library packages.



2. Define a driver class which will create a new client job, configuration object and advertise Mapper and Reducer classes.

The driver class is responsible for setting our MapReduce job to run in Hadoop. In this class, we specify **job name, data type of input/output and names of mapper and reducer classes**.

SalesCountryDriver.java

```java
package SalesCountry;

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

public class SalesCountryDriver {
        public static void main(String[] args) {
                JobClient my_client = new JobClient();
                // Create a configuration object for the job
                JobConf job_conf = new JobConf(SalesCountryDriver.class);

                // Set a name of the Job
                job_conf.setJobName("SalePerCountry");

                // Specify data type of output key and value
                job_conf.setOutputKeyClass(Text.class);
                job_conf.setOutputValueClass(IntWritable.class);

                // Specify names of Mapper and Reducer Class
                job_conf.setMapperClass(SalesCountry.SalesMapper.class);
                job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

                // Specify formats of the data type of Input and output
                job_conf.setInputFormat(TextInputFormat.class);
                job_conf.setOutputFormat(TextOutputFormat.class);
```

Start of definition of SalesCountryDriver class

Entry point to the application

3. In below code snippet, we set input and output directories which are used to consume input dataset and produce output, respectively.

arg[0] and **arg[1]** are the command-line arguments passed with a command given in MapReduce hands-on, i.e.,

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales

```
        // Set input and output directories using command line arguments,
/inputMapReduce  -> //arg[0] = name of input directory on HDFS, and
/mapreduce_output -> //arg[1] = name of output directory to be created to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
                // Run the job
                JobClient.runJob(job_conf);   <- This code initiates Map-Reduce job
        } catch (Exception e) {
                e.printStackTrace();
        }
    }
```

4. Trigger our job

Below code start execution of MapReduce job-

```
try {
    // Run the job
    JobClient.runJob(job_conf);
} catch (Exception e) {
    e.printStackTrace();
}
```

Conclusion:

Sample Questions:

1) Describe mapper & reducer class function.

2) Explain Mapreduce application program.

EXPERIMENT NO: 04

Title: Introduction of Hadoop HIVE
Aim: Understanding of HIVE working environment.

Theory:

Hive is an ETL and Data warehousing tool developed on top of Hadoop Distributed File System (HDFS). Hive makes job easy for performing operations like

- Data encapsulation
- Ad-hoc queries
- Analysis of huge datasets

Important characteristics of Hive

- In Hive, tables and databases are created first and then data is loaded into these tables.
- Hive as data warehouse designed for managing and querying only structured data that is stored in tables.
- While dealing with structured data, Map Reduce doesn't have optimization and usability features like UDFs but Hive framework does. Query optimization refers to an effective way of query execution in terms of performance.
- Hive's SQL-inspired language separates the user from the complexity of Map Reduce programming. It reuses familiar concepts from the relational database world, such as tables, rows, columns and schema, etc. for ease of learning.
- Hadoop's programming works on flat files. So, Hive can use directory structures to "partition" data to improve performance on certain queries.
- A new and important component of Hive i.e. Metastore used for storing schema information. This Metastore typically resides in a relational database. We can interact with Hive using methods like
    - **Web GUI**
    - **Java Database Connectivity (JDBC) interface**
- Most interactions tend to take place over a command line interface (CLI). Hive provides a CLI to write Hive queries using Hive Query Language(HQL)
- Generally, HQL syntax is similar to the SQL syntax that most data analysts are familiar with. The Sample query below display all the records present in mentioned table name.
    - **Sample query** : Select * from <TableName>
- Hive supports four file formats those are **TEXTFILE, SEQUENCEFILE, ORC and RCFILE** (Record Columnar File).
- For single user metadata storage, Hive uses derby database and for multiple user Metadata or shared Metadata case Hive uses MYSQL.

Some of the key points about Hive:

- The major difference between HQL and SQL is that Hive query executes on Hadoop's infrastructure rather than the traditional database.
- The Hive query execution is going to be like series of automatically generated map reduce Jobs.
- Hive supports partition and buckets concepts for easy retrieval of data when the client executes the query.
- Hive supports custom specific UDF (User Defined Functions) for data cleansing, filtering, etc. According to the requirements of the programmers one can define Hive UDFs.

Hive Vs Relational Databases:-

By using Hive, we can perform some peculiar functionality that is not achieved in Relational Databases. For a huge amount of data that is in peta-bytes, querying it and getting results in seconds is important. And Hive does this quite efficiently, it processes the queries fast and produce results in second's time.

Let see now what makes Hive so fast.

Some key differences between Hive and relational databases are the following;

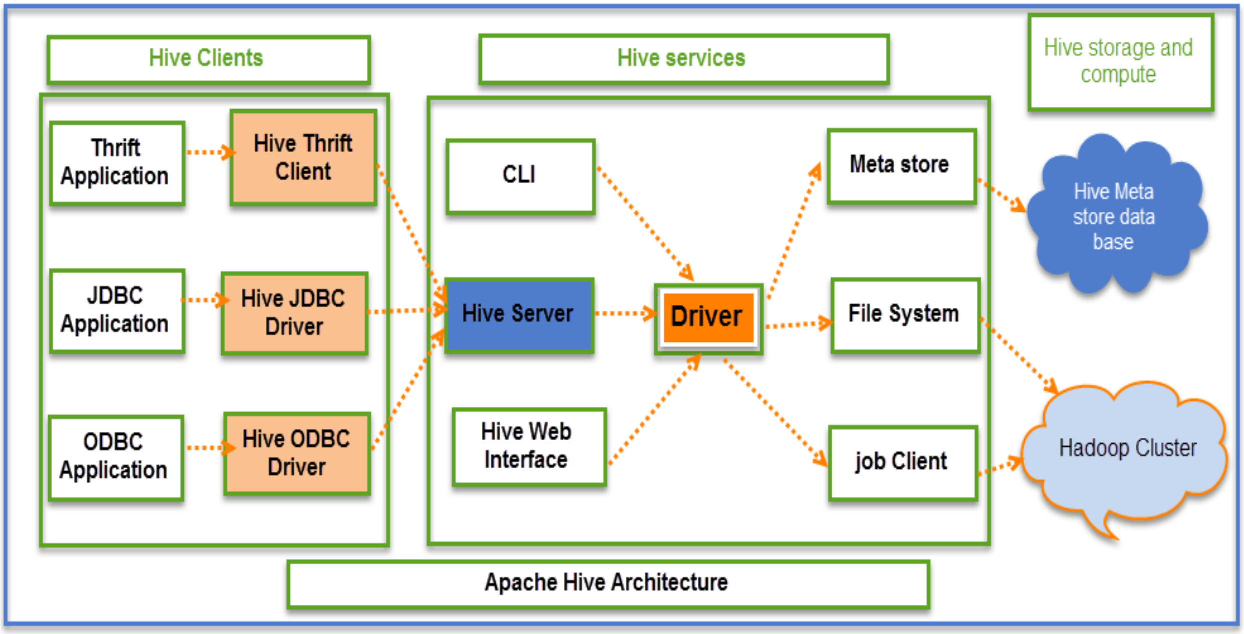
Relational databases are of "**Schema on READ and Schema on Write**". First creating a table then inserting data into the particular table. On relational database tables, functions like Insertions, Updates, and Modifications can be performed.

Hive is "**Schema on READ only**". So, functions like the update, modifications, etc. don't work with this. Because the Hive query in a typical cluster runs on multiple Data Nodes. So it is not possible to update and modify data across multiple nodes.(Hive versions below 0.13)

Also, Hive supports "**READ Many WRITE Once**" pattern. Which means that after inserting table we can update the table in the latest Hive versions.

NOTE: However the new version of Hive comes with updated features. Hive versions (Hive 0.14) comes up with Update and Delete options as new features

Hive Architecture:



The above screenshot explains the Apache Hive architecture in detail

Hive Consists of Mainly 3 core parts

1. **Hive Clients**
2. **Hive Services**
3. **Hive Storage and Computing**

Hive Clients:

Hive provides different drivers for communication with a different type of applications. For Thrift based applications, it will provide Thrift client for communication.

For Java related applications, it provides JDBC Drivers. Other than any type of applications provided ODBC drivers. These Clients and drivers in turn again communicate with Hive server in the Hive services.

Hive Services:

Client interactions with Hive can be performed through Hive Services. If the client wants to perform any query related operations in Hive, it has to communicate through Hive Services.

CLI is the command line interface acts as Hive service for DDL (Data definition Language) operations. All drivers communicate with Hive server and to the main driver in Hive services as shown in above architecture diagram.

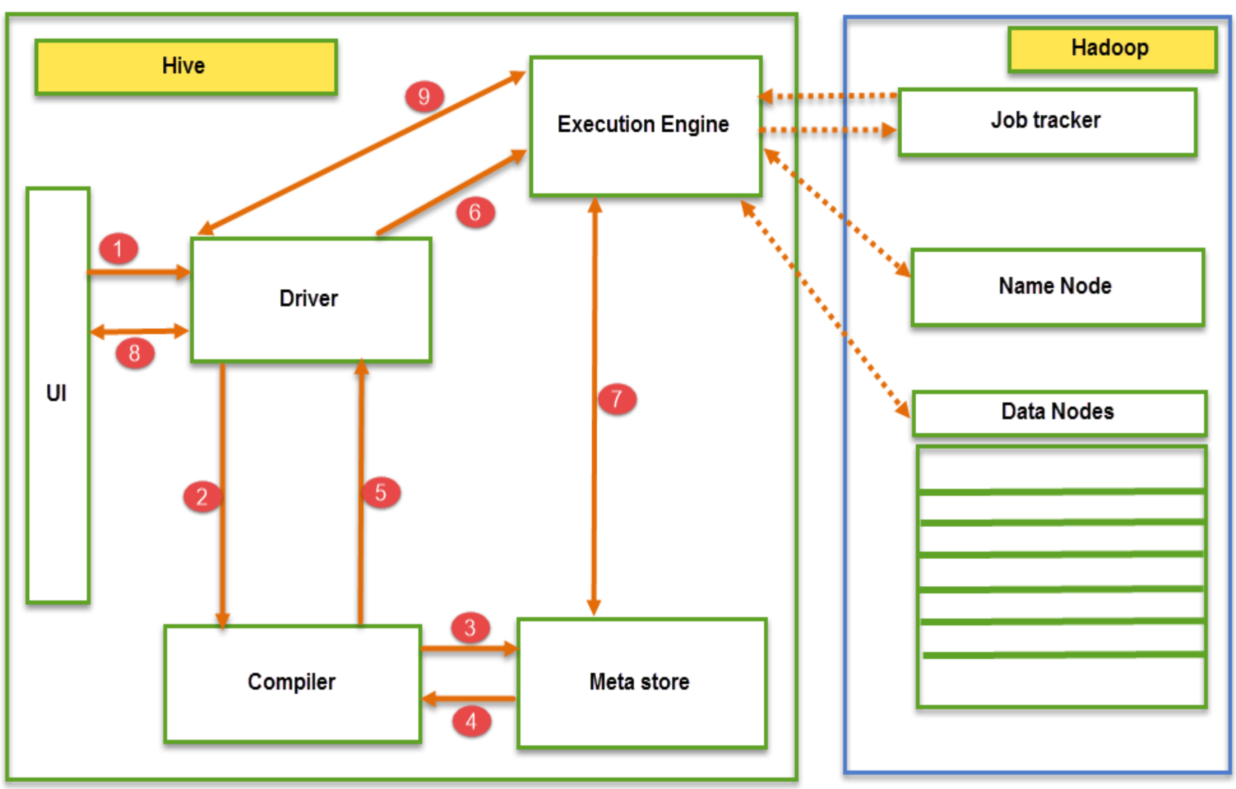
Driver present in the Hive services represents the main driver, and it communicates all type of JDBC, ODBC, and other client specific applications. Driver will process those requests from different applications to meta store and field systems for further processing.

Hive Storage and Computing:

Hive services such as Meta store, File system, and Job Client in turn communicates with Hive storage and performs the following actions

- Metadata information of tables created in Hive is stored in Hive "Meta storage database".
- Query results and data loaded in the tables are going to be stored in Hadoop cluster on HDFS.

Job exectution flow:



From the above screenshot we can understand the Job execution flow in Hive with Hadoop

The data flow in Hive behaves in the following pattern;

1. Executing Query from the UI(User Interface)

2. The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering
3. The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
4. Meta store sends metadata information back to compiler
5. Compiler communicating with Driver with the proposed plan to execute the query
6. Driver Sending execution plans to Execution engine
7. Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS operations.

    - EE should first contacts Name Node and then to Data nodes to get the values stored in tables.
    - EE is going to fetch desired records from Data Nodes. The actual data of tables resides in data node only. While from Name Node it only fetches the metadata information for the query.
    - It collects actual data from data nodes related to mentioned query
    - Execution Engine (EE) communicates bi-directionally with Meta store present in Hive to perform DDL (Data Definition Language) operations. Here DDL operations like CREATE, DROP and ALTERING tables and databases are done. Meta store will store information about database name, table names and column names only. It will fetch data related to query mentioned.
    - Execution Engine (EE) in turn communicates with Hadoop daemons such as Name node, Data nodes, and job tracker to execute the query on top of Hadoop file system

8. Fetching results from driver
9. Sending results to Execution engine. Once the results fetched from data nodes to the EE, it will send results back to driver and to UI (front end)

Hive Continuously in contact with Hadoop file system and its daemons via Execution engine. The dotted arrow in the Job flow diagram shows the Execution engine communication with Hadoop daemons.

Different modes of Hive

Hive can operate in two modes depending on the size of data nodes in Hadoop.

These modes are,

- **Local mode**
- **Map reduce mode**

When to use Local mode:

- If the Hadoop installed under pseudo mode with having one data node we use Hive in this mode
- If the data size is smaller in term of limited to single local machine, we can use this mode

- Processing will be very fast on smaller data sets present in the local machine

When to use Map reduce mode:

- If Hadoop is having multiple data nodes and data is distributed across different node we use Hive in this mode
- It will perform on large amount of data sets and query going to execute in parallel way
- Processing of large data sets with better performance can be achieved through this mode

In Hive, we can set this property to mention which mode Hive can work? By default, it works on Map Reduce mode and for local mode you can have the following setting.

Hive to work in local mode set

SET mapred.job.tracker=local;

From the Hive version 0.7 it supports a mode to run map reduce jobs in local mode automatically.

What is Hive Server2 (HS2)?

HiveServer2 (HS2) is a server interface that performs following functions:

- Enables remote clients to execute queries against Hive
- Retrieve the results of mentioned queries

From the latest version it's having some advanced features Based on Thrift RPC like;

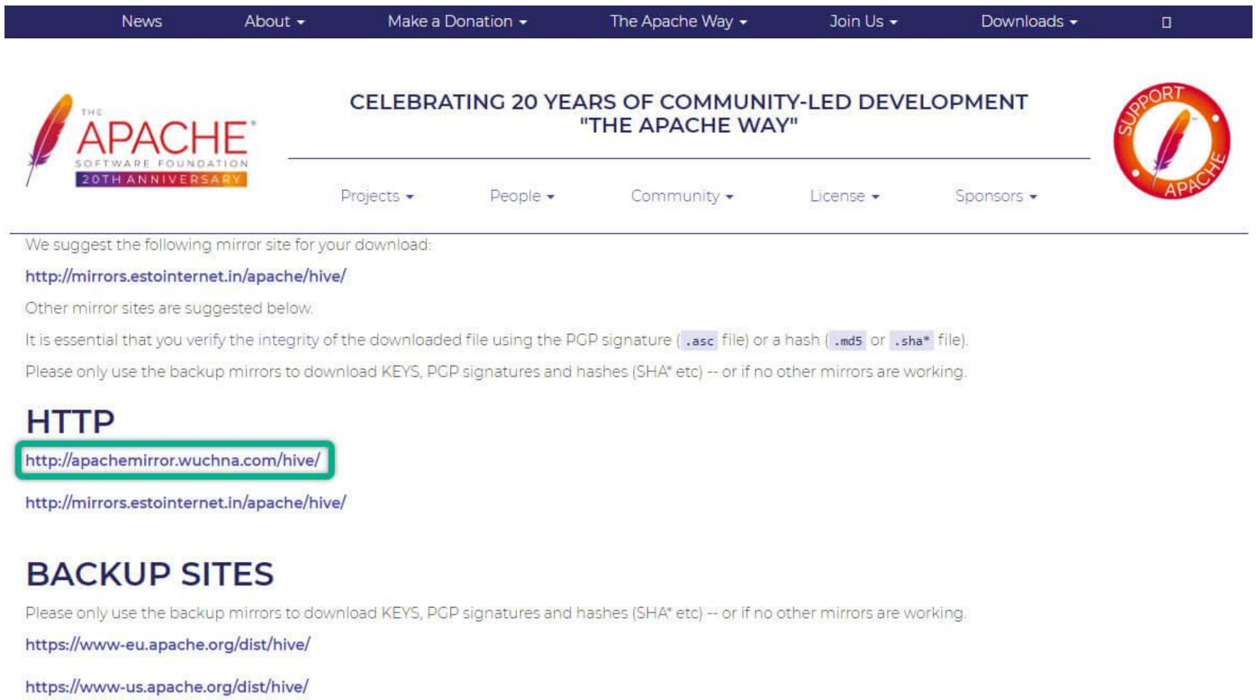
- Multi-client concurrency
- Authentication

How to Install Hive

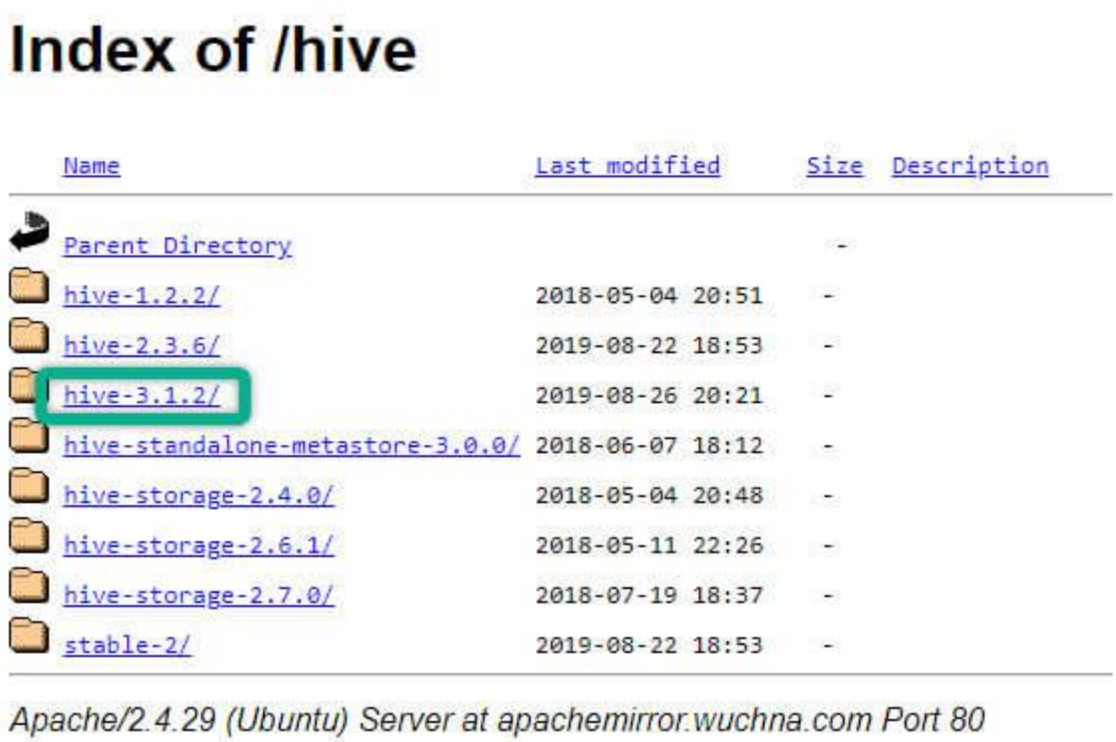
Step 1) Downloading and Installing Hive

For downloading Hive stable setup refer Apache URL as mentioned below

http://www.apache.org/dyn/closer.cgi/hive/. Go to the URL and select the apache mirror download link.



Select the Latest version of Hive. (In my current case it is hive – 3.1.2)

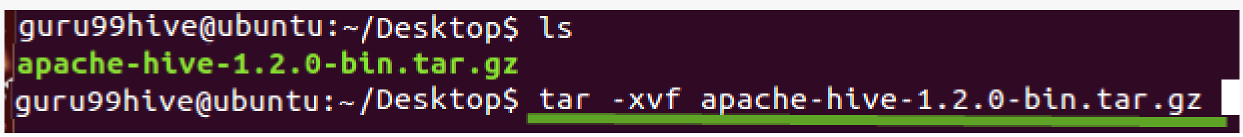


Click on the bin file and downloading will start.

Step 2) Extracting the tar file.

Go to the downloaded Tar file location ->extract the tar file by using the following command

```
tar –xvf  apache-hive-3.1.2-bin.tar.gz
```



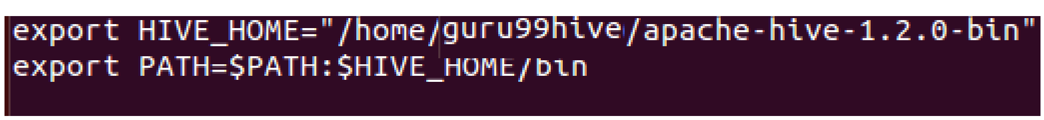
Step 3) Different Configuration properties to be placed in Apache Hive.

In this step, we are going to do two things

1. Placing Hive Home path in bashrc file
2. Placing Hadoop Home path location in hive-config.sh

1. Mention **Hive Pathin ~/.bashrc**

- Open the bashrc file as shown in above screenshot
- Mention Hive home path i.e., HIVE_HOME path in bashrc file and export it as shown in below



Code to be placed in bashrc

export HIVE_HOME="/home/guru99hive/apache-hive-1.2.0-bin"

export PATH=$PATH:$HIVE_HOME/bin

2. Exporting **Hadoop path in Hive-config.sh** (To communicate with the Hadoop eco system we are defining Hadoop Home path in hive config field)

   Open the hive-config.sh as shown in below

```
guru99hive@ubuntu:~/apache-hive-1.2.0-bin/bin$ nano hive-config.sh
```

Mention the HADOOP_HOME Path in hive-config.sh file as shown in below (HADOOP_HOME Path)

```
export HADOOP_HOME=/home/guru99hive/Hadoop_YARN/hadoop-2.2.0
```

Step 4) Creating Hive directories in Hadoop:

To communicate with Hadoop, we need to create directories in Hadoop as shown below.

```
guru99hive@ubuntu:~/Hadoop_YARN/hadoop-2.2.0/bin$ ./hadoop fs -mkdir /usr/hive/w
arehouse
```

Giving root permissions to create Hive folders in Hadoop.If it doesn't throw any error message, then it means that Hadoop has successfully given permissions to Hive folders.

```
guru99hive@ubuntu:~/Hadoop_YARN/hadoop-2.2.0/bin$ ./hadoop fs -chmod g+w /usr/hi
ve/warehouse
```

Step 5) Getting into Hive shell by entering **'. /hive'** command as shown in below.

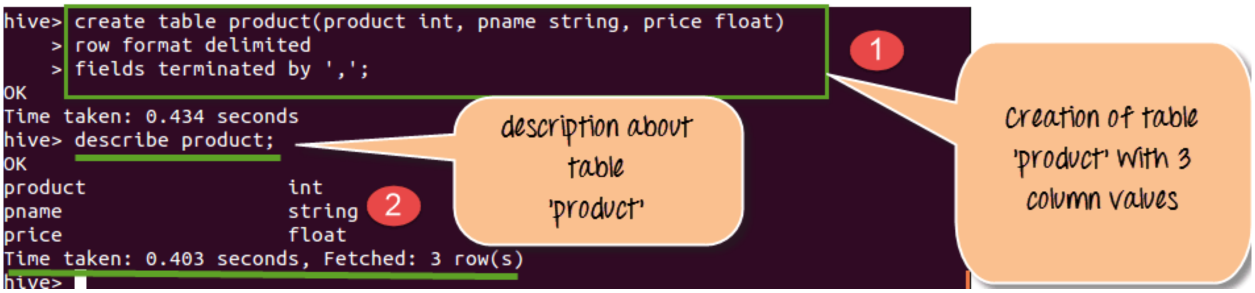
```
guru99hive@ubuntu:~/apache-hive-1.2.0-bin/bin$ ./hive
Logging initialized using configuration in jar:file:/home/datamatics/apache-hive
-1.2.0-bin/lib/hive-common-1.2.0.jar!/hive-log4j.properties
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /home/datamat
ics/Hadoop_YARN/hadoop-2.2.0/lib/native/libhadoop.so.1.0.0 which might have disa
bled stack guard. 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'.
hive>
```

Theory:

Hive shell commands

Here we are going to create sample table using Hive shell command "create" with column names.

Sample Code for creating data base in Hive



From the above screen shot we can observe the following:

1. Creation of Sample Table with column names in Hive

   - Here the table name is "product" with three column names **product, pname, and price**
   - The three column names denoted by their respective data type
   - All fields are terminated by coma ', '

2. Displaying Hive Table information

   - Using "describe" command we can able to see the table information present in Hive
   - Here it is displaying column names with their respective data types present in table schema
   - At the end, it will display time to perform this command and number of rows it fetched

Sample Code for creating data base in Hive (For self check)

1) Create table product(product int, pname string, price float)

Row format delimited

Fields terminated by ',';

2) describe product:

Data types in Hive

Data types are very important elements in Hive query language and data modeling. For defining the table column types, we must have to know about the data types and its usage.

The following gives brief overview of some data types present in Hive:

These are

- Numeric Types
- String Types
- Date/Time Types
- Complex Types

Numeric Types:

Type	Memory allocation
TINY INT	Its 1-byte signed integer (-128 to 127)
SMALL INT	2-byte signed integer (-32768 to 32767)
INT	4 –byte signed integer (-2,147,484,648 to 2,147,484,647)
BIG INT	8 byte signed integer
FLOAT	4 – byte single precision floating point number
DOUBLE	8- byte double precision floating point number
DECIMAL	We can define precision and scale in this Type

Creation and dropping of Database in Hive:

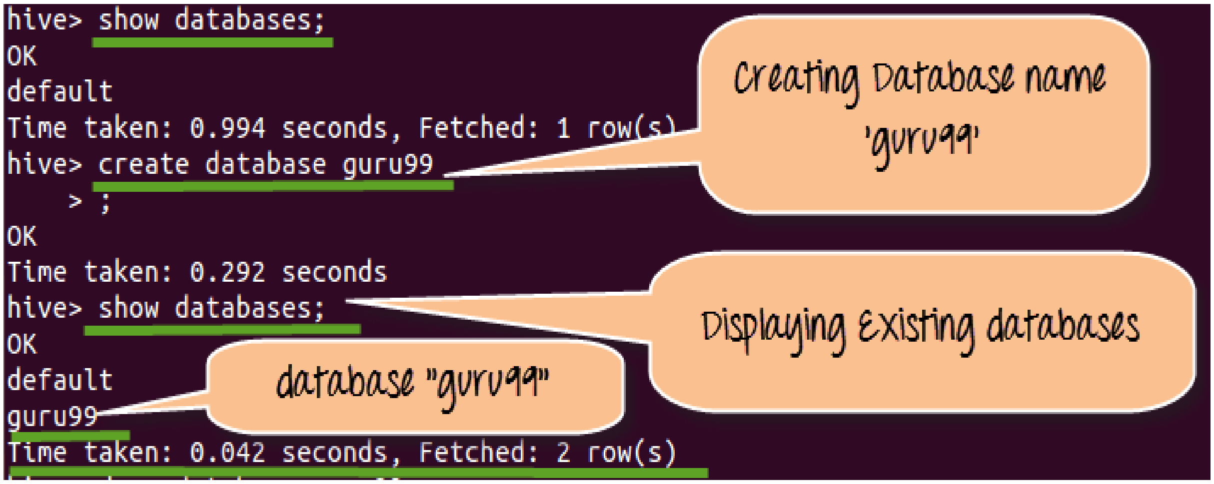
Create Database:

For creating database in Hive shell, we have to use the command as shown in syntax below:-

Syntax:

Create database <DatabaseName>

Example: -Create database "guru99"



From the above screen shot, we are doing two things

- Creating database "guru99" in Hive
- Displaying existing databases using "show" command
- In the same screen, Database "guru99" name is displayed at the end when we execute the show command. Which means Database "guru99" is successfully created.

Drop Database:

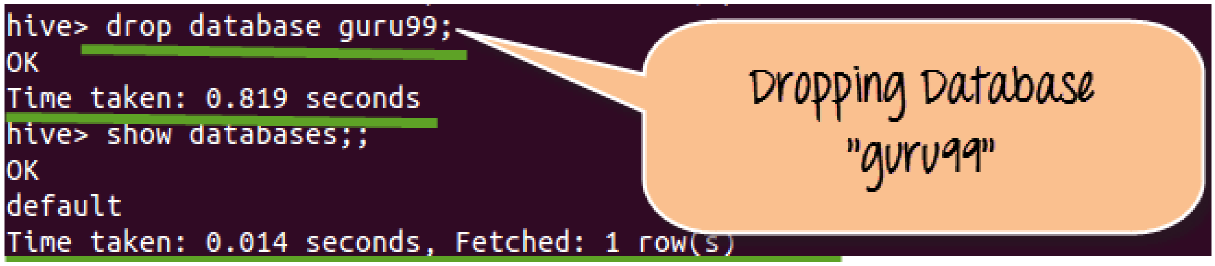
For Dropping database in Hive shell, we have to use the **"drop"** command as shown in syntax below:-

Syntax:

Drop database <DatabaseName>

Example:-

Drop database guru99



In the above screenshot, we are doing two things

- We are dropping database 'guru99' from Hive
- Cross checking the same with "show" command
- In the same screen, after checking databases with show command, database"guru99" does not appear inside Hive.
- So we can confirm now that database "guru99" is dropped

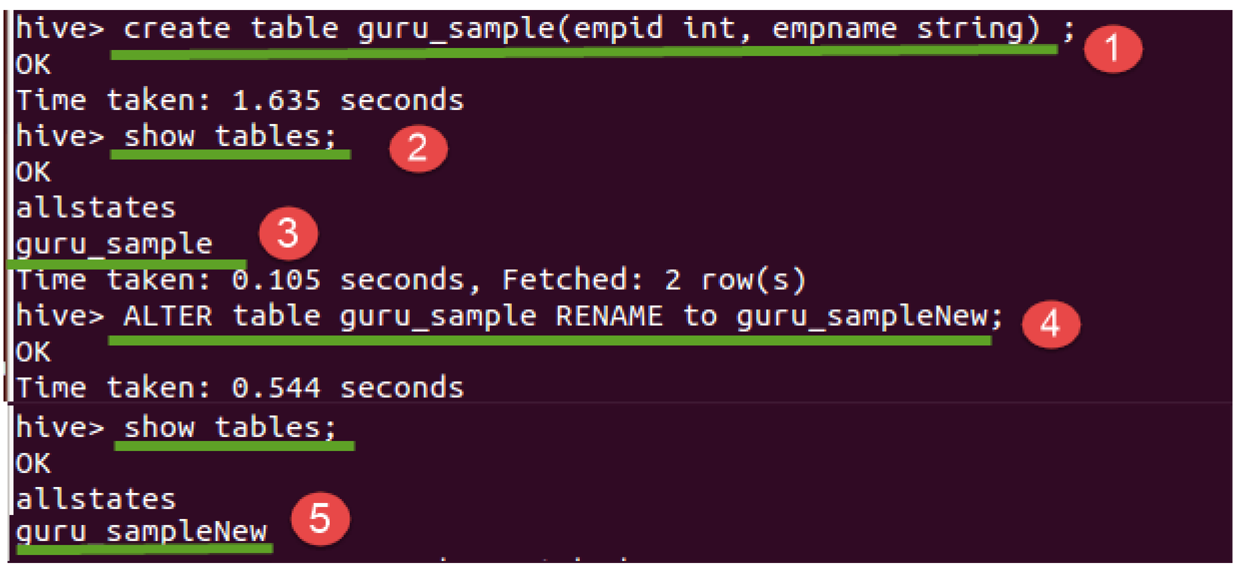
1. Creating table guru_sample with two column names such as "empid" and "empname"

2. Displaying tables present in guru99 database

3. Guru_sample displaying under tables

4. Altering table "guru_sample" as "guru_sampleNew"

5. Again when you execute "show" command, it will display the new name Guru_sampleNew



Dropping table guru_sampleNew:

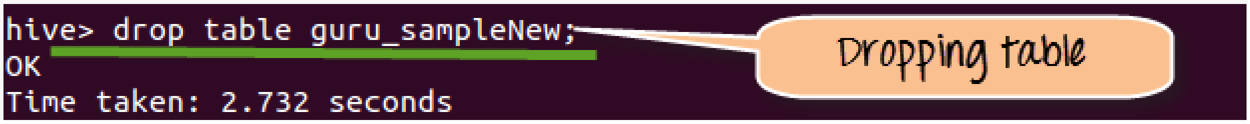


Table types and its Usage:

Coming to **Tables** it's just like the way that we create in traditional Relational Databases. The functionalities such as filtering, joins can be performed on the tables.

Hive deals with two types of table structures like **Internal and External** tables depending on the loading and design of schema in Hive.

Internal tables

- Internal Table is tightly coupled in nature.In this type of table, first we have to create table and load the data.
- We can call this one as **data on schema**.
- By dropping this table, both data and schema will be removed.
- The stored location of this table will be at /user/hive/warehouse.

When to Choose Internal Table:

- If the processing data available in local file system
- If we want Hive to manage the complete lifecycle of data including the deletion

Sample code Snippet for Internal Table

1. To create the internal table

```
Hive>CREATE TABLE guruhive_internaltable (id INT,Name STRING);
          Row format delimited
          Fields terminated by '\t';
```

2. Load the data into internal table

```
Hive>LOAD DATA INPATH '/user/guru99hive/data.txt' INTO table guruhive_internaltable;
```

3. Display the content of the table

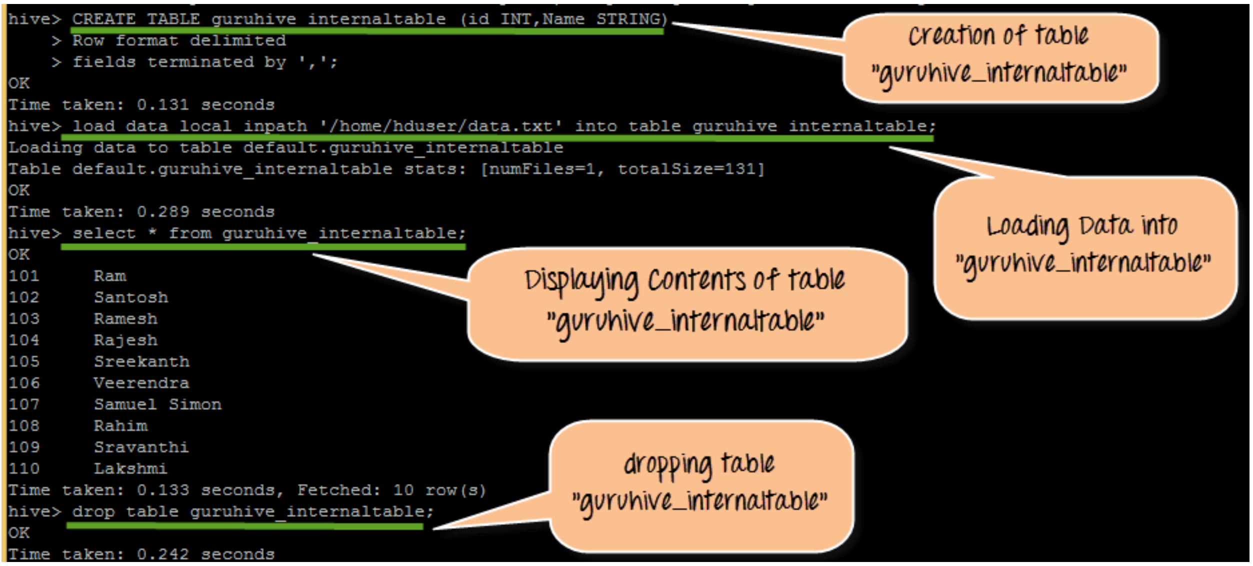
```
Hive>select * from guruhive_internaltable;
```

4. To drop the internal table

```
Hive>DROP TABLE guruhive_internaltable;
```

If you dropped the guruhive_internaltable, including its metadata and its data will be deleted from Hive.

From the following screenshot, we can observe the output



In above code and from screen shot we do following things,

- Create the internal table
- Load the data into internal table
- Display the content of the table
- To drop the internal table

External tables

- External Table is loosely coupled in nature. Data will be available in HDFS.The table is going to create on HDFS data.
- In other way, we can say like its creating **schema on data**.
- At the time of dropping the table it drops only schema, the data will be still available in HDFS as before.
- External tables provide an option to create multiple schemas for the data stored in HDFS instead of deleting the data every time whenever schema updates

When to Choose External Table:

- If processing data available in HDFS
- Useful when the files are being used outside of Hive

Sample code Snippet for External Table

1. Create External table

```
Hive>CREATE EXTERNAL TABLE guruhive_external(id INT,Name STRING)
        Row format delimited
        Fields terminated by '\t'
        LOCATION '/user/guru99hive/guruhive_external;
```

2. If we are not specifying the location at the time of table creation, we can load the data manually

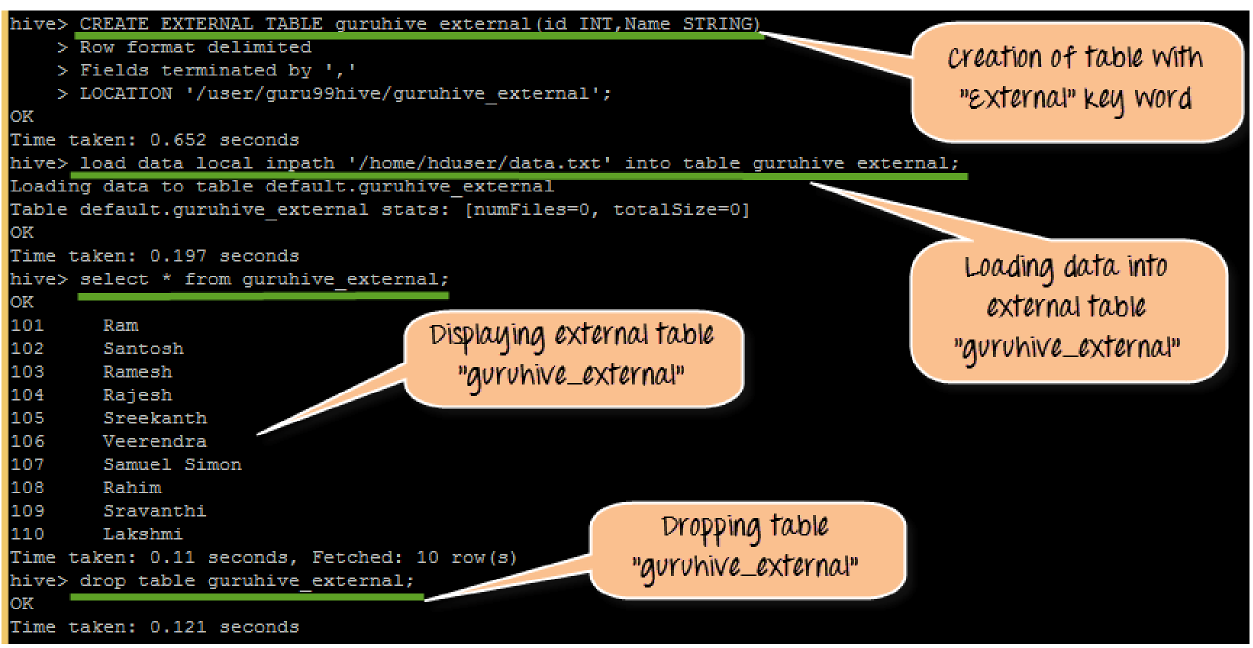
```
Hive>LOAD DATA INPATH '/user/guru99hive/data.txt' INTO TABLE guruhive_external;
```

3. Display the content of the table

```
Hive>select * from guruhive_external;
```

4. To drop the internal table

```
Hive>DROP TABLE guruhive_external;
```



Conclusion:

Sample Questions:

1) Explain Hadoop Hive.

2) Explain Hadoop job execution flow.

3) Explain Hadoop HIVE architechture.

4) Explain Characteristics of Hadoop HIVE.

5) Explain Hadoop HIVE DML commands.

6) Explain Hadoop HIVE DDL commands.

EXPERIMENT NO: 05

Title: Introduction of Apache Pig.
Aim: Understanding working environment of Apache Pig

Theory:

Pig is a high-level programming language useful for analyzing large data sets. Pig was a result of development effort at Yahoo!

In a MapReduce framework, programs need to be translated into a series of Map and Reduce stages. However, this is not a programming model which data analysts are familiar with. So, in order to bridge this gap, an abstraction called Pig was built on top of Hadoop.

Apache Pig enables people to focus more on **analyzing bulk data sets and to spend less time writing Map-Reduce programs.** Similar to Pigs, who eat anything, the Apache Pig programming language is designed to work upon any kind of data. That's why the name, Pig!

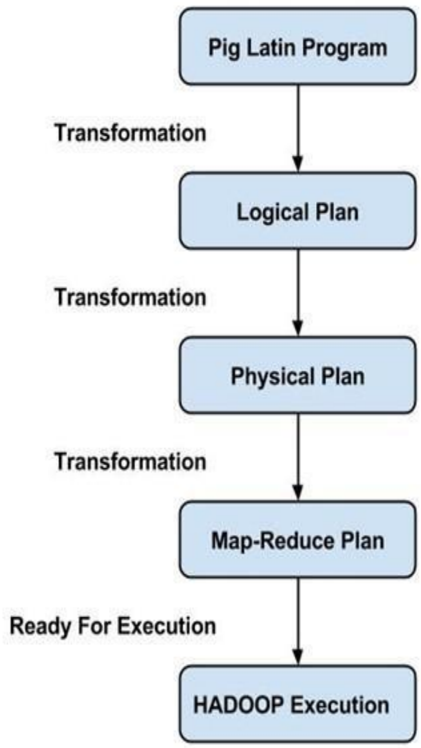
Pig Architecture

The Architecture of Pig consists of two components:

1. **Pig Latin,** which is a language
2. **A runtime environment,** for running PigLatin programs.

A Pig Latin program consists of a series of operations or transformations which are applied to the input data to produce output. These operations describe a data flow which is translated into an executable representation, by Hadoop Pig execution environment. Underneath, results of these transformations are series of MapReduce jobs which a programmer is unaware of. So, in a way, Pig in Hadoop allows the programmer to focus on data rather than the nature of execution.

PigLatin is a relatively stiffened language which uses familiar keywords from data processing e.g., Join, Group and Filter.



PIG Architecture

Execution modes:

Pig in Hadoop has two execution modes:

1. Local mode: In this mode, Hadoop Pig language runs in a single JVM and makes use of local file system. This mode is suitable only for analysis of small datasets using Pig in Hadoop
2. Map Reduce mode: In this mode, queries written in Pig Latin are translated into Map Reduce jobs and are run on a Hadoop cluster (cluster may be pseudo or fully distributed). MapReduce mode with the fully distributed cluster is useful of running Pig on large datasets.

How to Download and Install Pig

Now in this Apache Pig tutorial, we will learn how to download and install Pig:

Before we start with the actual process, ensure you have Hadoop installed. Change user to 'hduser' (id used while Hadoop configuration, you can switch to the userid used during your Hadoop config)

```
guru99@guru99-VirtualBox:~$ su - hduser_
Password:
hduser_@guru99-VirtualBox:~$
```

Step 1) Download the stable latest release of Pig Hadoop from any one of the mirrors sites available at

http://pig.apache.org/releases.html

Select **tar.gz** (and not **src.tar.gz)** file to download.

Step 2) Once a download is complete, navigate to the directory containing the downloaded tar file and move the tar to the location where you want to setup Pig Hadoop. In this case, we will move to /usr/local

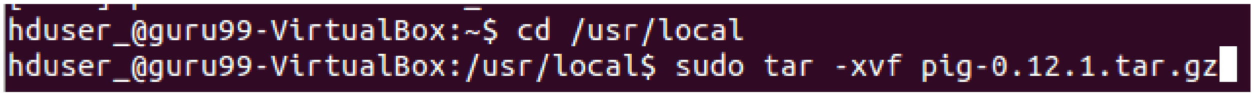
```
hduser_@guru99-VirtualBox:~$ sudo mv /home/guru99/Downloads/pig-0.12.1.tar.gz  /usr/local
[sudo] password for hduser_:
hduser_@guru99-VirtualBox:~$
```

Move to a directory containing Pig Hadoop Files

```
cd /usr/local
```

Extract contents of tar file as below

sudo tar -xvf pig-0.12.1.tar.gz

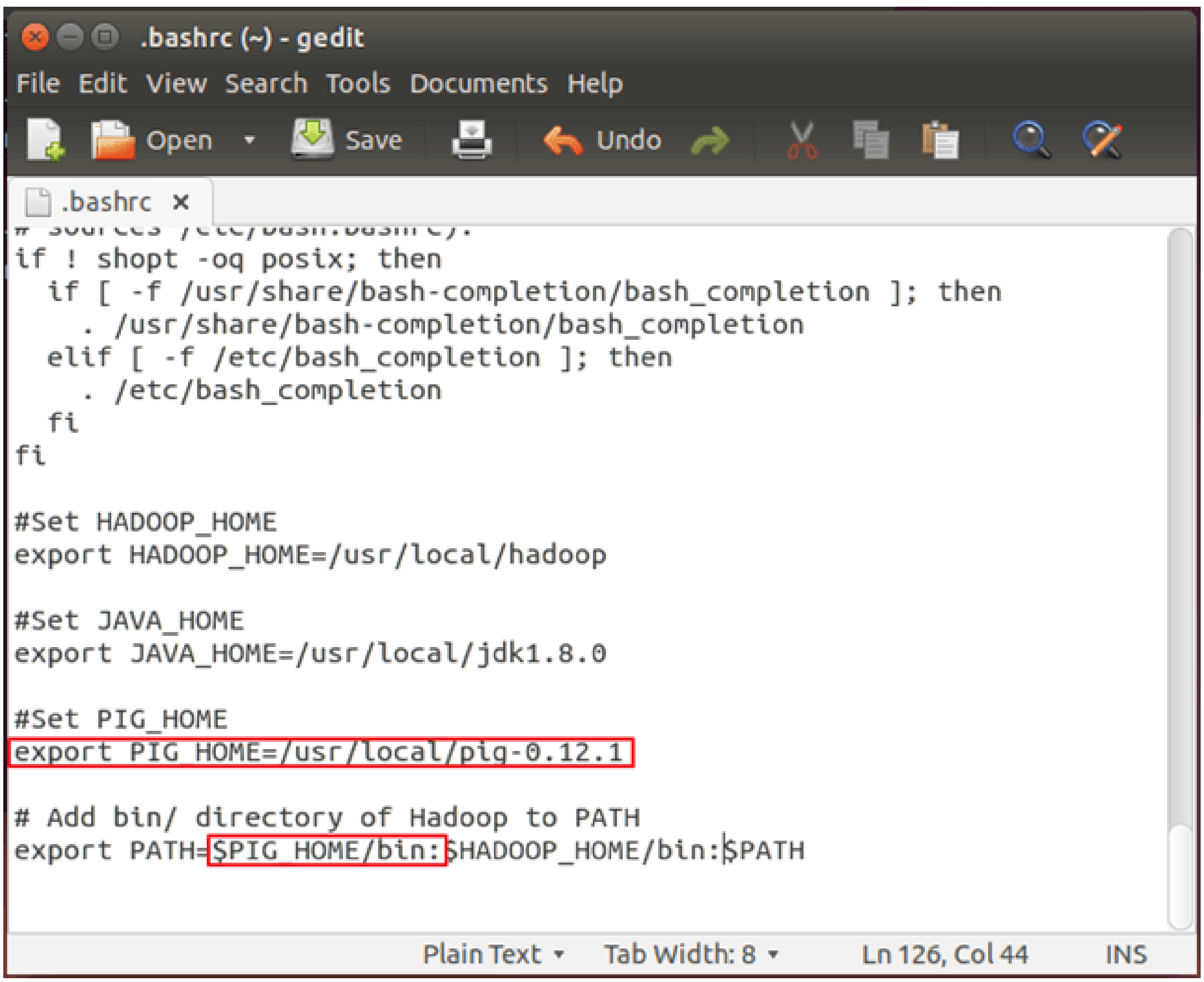


Step 3). Modify **~/.bashrc** to add Pig related environment variables

Open **~/.bashrc** file in any text editor of your choice and do below modifications-

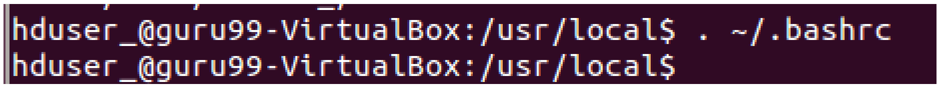
export PIG_HOME=<Installation directory of Pig>

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



Step 4) Now, source this environment configuration using below command

```
. ~/.bashrc
```



Step 5) We need to recompile **PIG** to support **Hadoop 2.2.0**

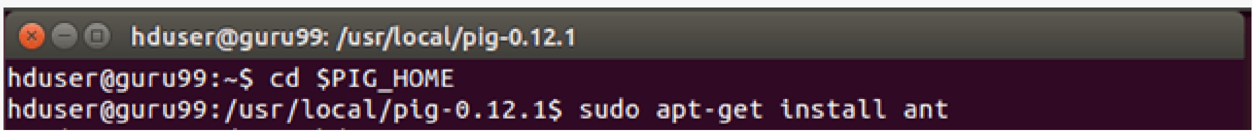
Here are the steps to do this-

Go to PIG home directory

```
cd $PIG_HOME
```

Install Ant

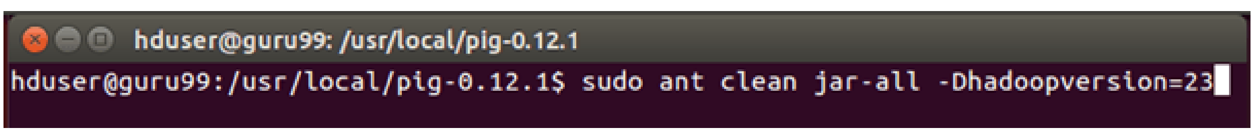
```
sudo apt-get install ant
```



Note: Download will start and will consume time as per your internet speed.

Recompile PIG

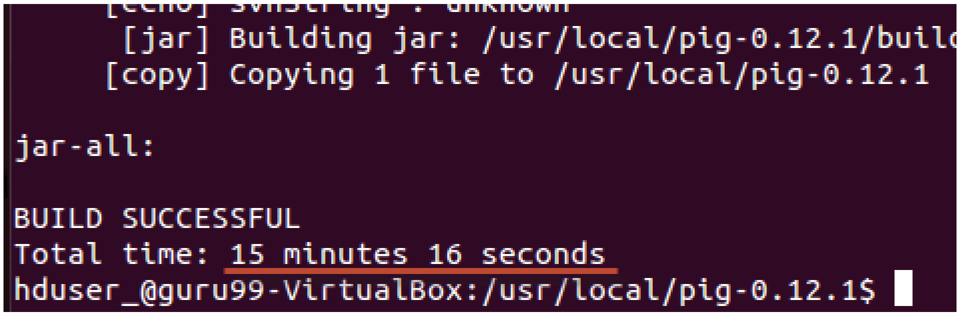
```
sudo ant clean jar-all -Dhadoopversion=23
```



Please note that in this recompilation process multiple components are downloaded. So, a system should be connected to the internet.

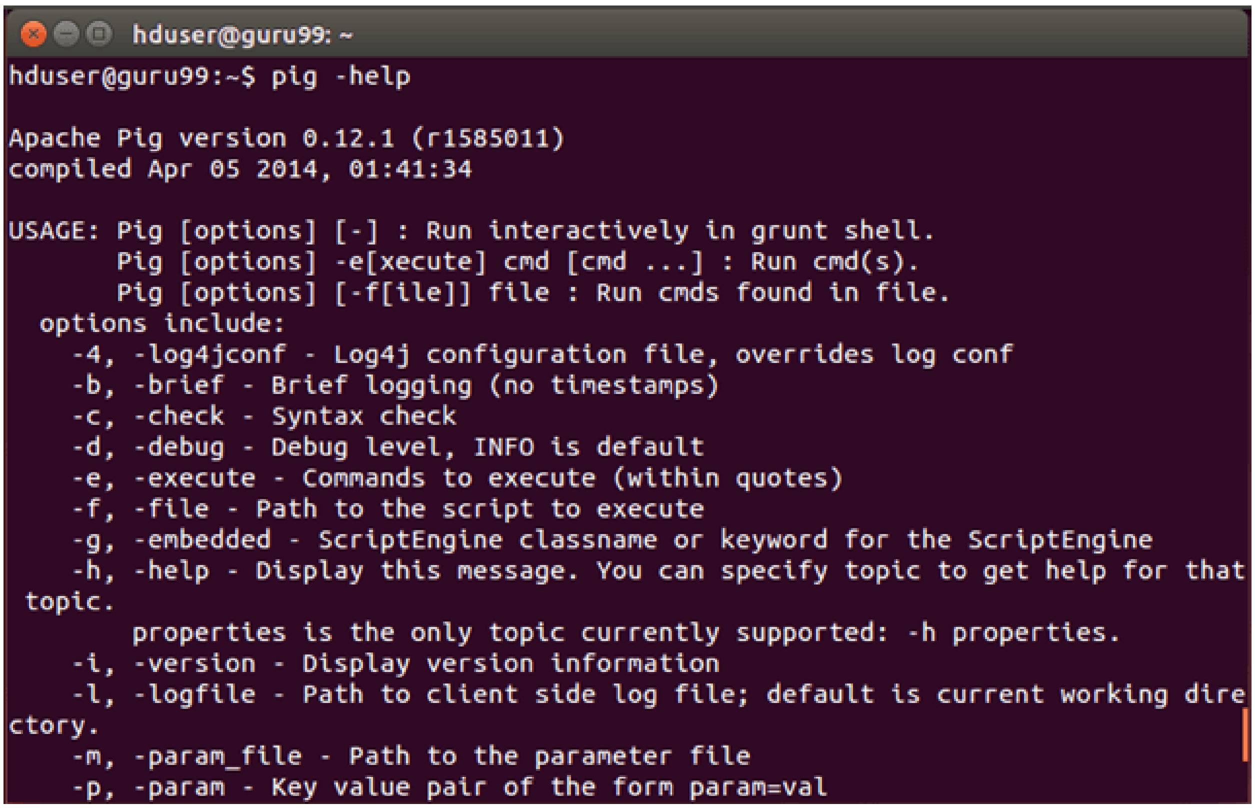
Also, in case this process stuck somewhere and you don't see any movement on command prompt for more than 20 minutes then press **Ctrl + c** and rerun the same command.

In our case, it takes 20 minutes



Step 6) Test the **Pig** installation using the command

```
pig -help
```



Example Pig Script

We will use Pig Scripts to find the Number of Products Sold in Each Country.

Input: Our input data set is a CSV file, SalesJan2009.csv

Step 1) Start Hadoop

$HADOOP_HOME/sbin/start-dfs.sh

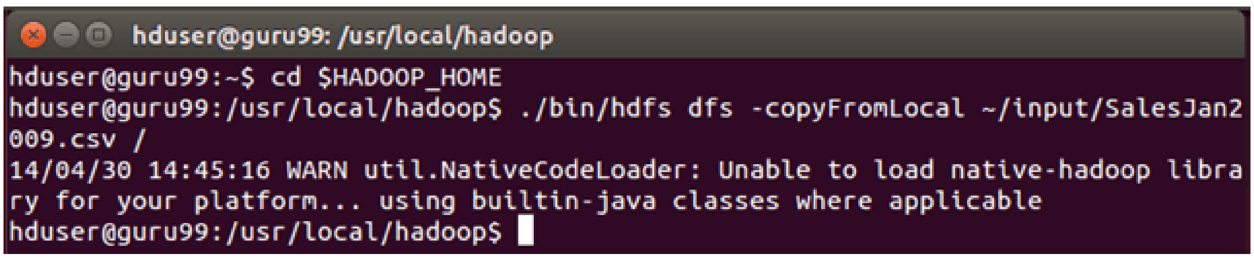
$HADOOP_HOME/sbin/start-yarn.sh

Step 2) Pig in Big Data takes a file from HDFS in MapReduce mode and stores the results back to HDFS.

Copy file **SalesJan2009.csv** (stored on local file system, **~/input/SalesJan2009.csv**) to HDFS (Hadoop Distributed File System) Home Directory

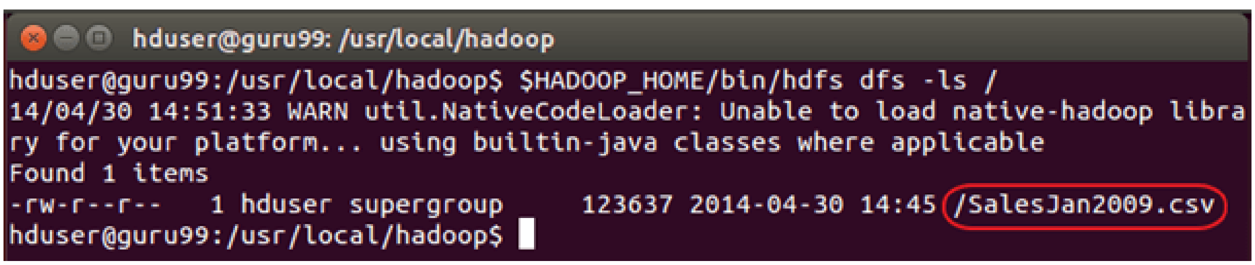
Here in this Apache Pig example, the file is in Folder input. If the file is stored in some other location give that name

$HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/input/SalesJan2009.csv /



Verify whether a file is actually copied or not.

$HADOOP_HOME/bin/hdfs dfs -ls /

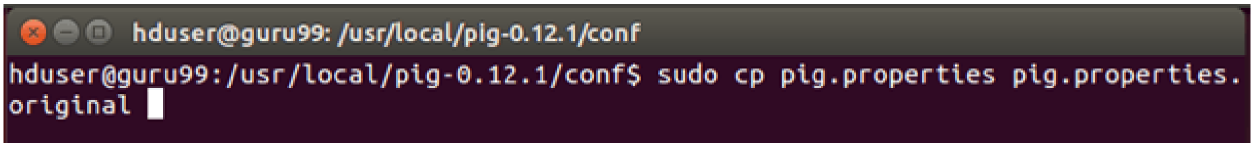


Step 3) Pig Configuration

First, navigate to $PIG_HOME/conf

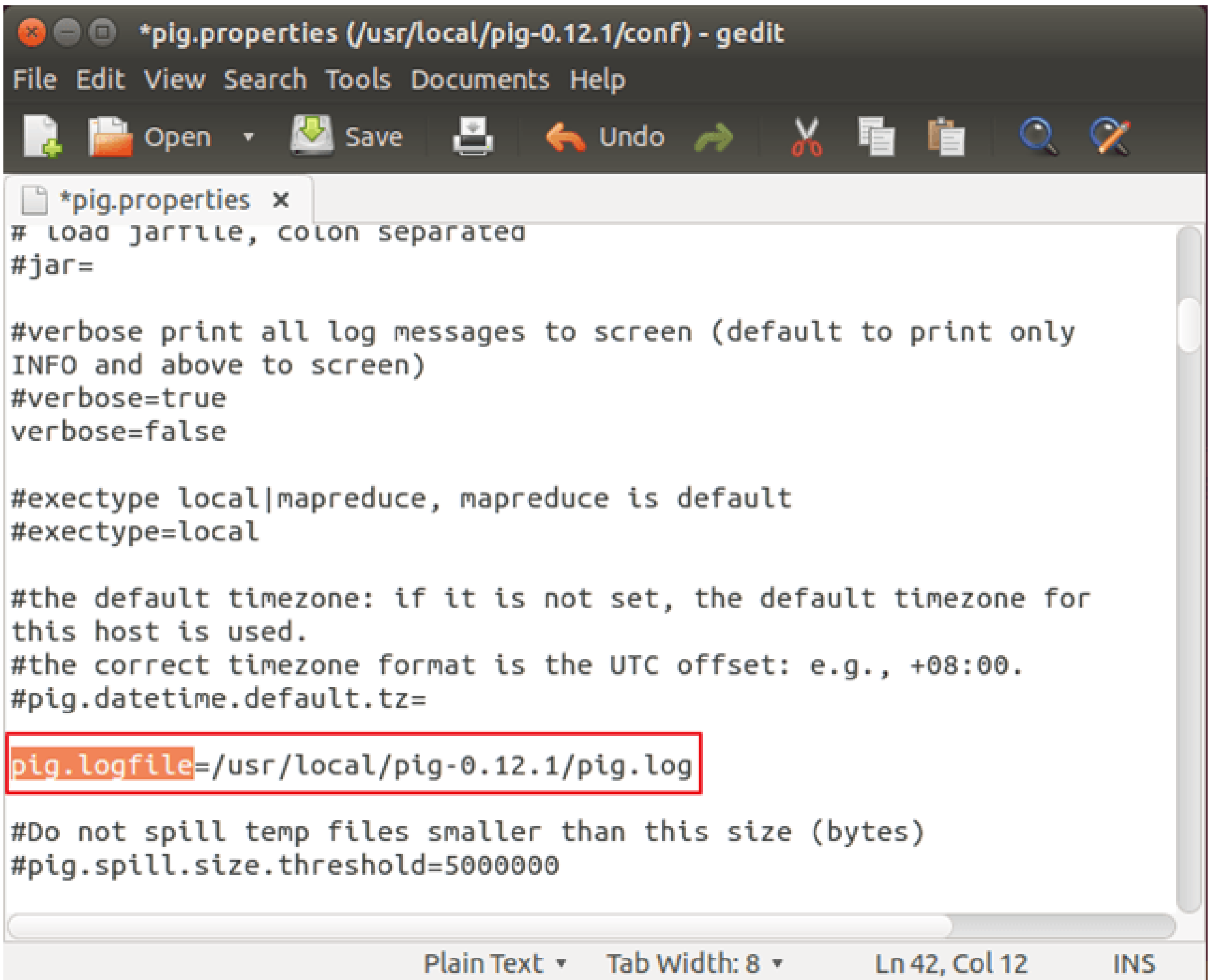
cd $PIG_HOME/conf

sudo cp pig.properties pig.properties.original



Open **pig.properties** using a text editor of your choice, and specify log file path using **pig.logfile**

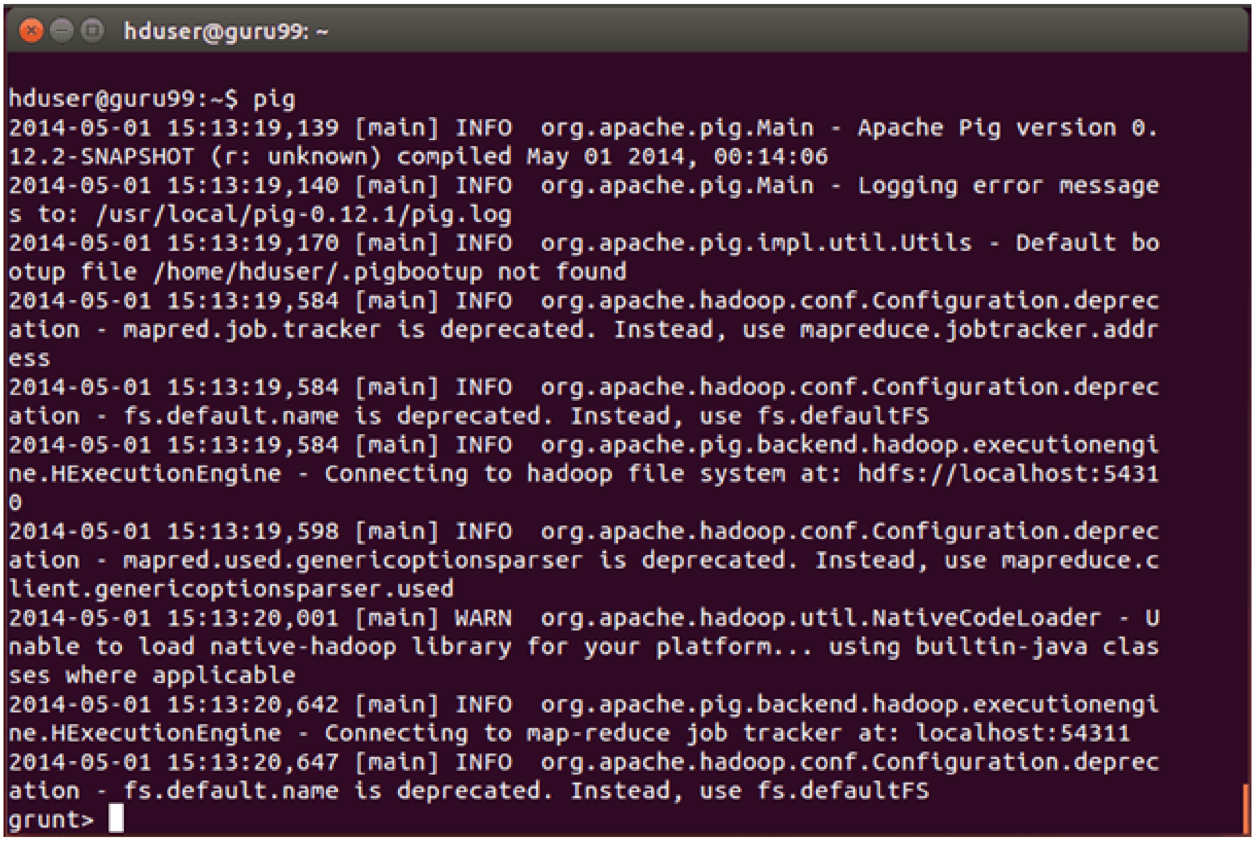
sudo gedit pig.properties



Loger will make use of this file to log errors.

Step 4) Run command 'pig' which will start Pig command prompt which is an interactive shell Pig queries.

pig



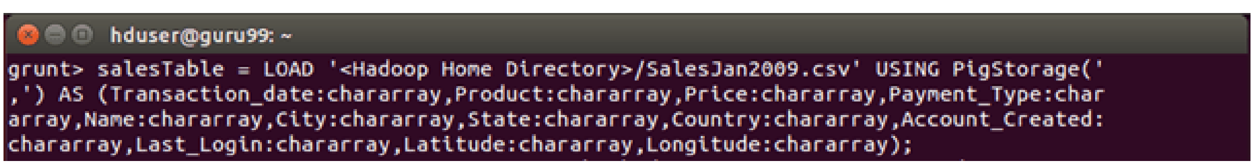
```
hduser@guru99: ~
hduser@guru99:~$ pig
2014-05-01 15:13:19,139 [main] INFO  org.apache.pig.Main - Apache Pig version 0.
12.2-SNAPSHOT (r: unknown) compiled May 01 2014, 00:14:06
2014-05-01 15:13:19,140 [main] INFO  org.apache.pig.Main - Logging error message
s to: /usr/local/pig-0.12.1/pig.log
2014-05-01 15:13:19,170 [main] INFO  org.apache.pig.impl.util.Utils - Default bo
otup file /home/hduser/.pigbootup not found
2014-05-01 15:13:19,584 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2014-05-01 15:13:19,584 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2014-05-01 15:13:19,584 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.HExecutionEngine - Connecting to hadoop file system at: hdfs://localhost:5431
0
2014-05-01 15:13:19,598 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.used.genericoptionsparser is deprecated. Instead, use mapreduce.c
lient.genericoptionsparser.used
2014-05-01 15:13:20,001 [main] WARN  org.apache.hadoop.util.NativeCodeLoader - U
nable to load native-hadoop library for your platform... using builtin-java clas
ses where applicable
2014-05-01 15:13:20,642 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.HExecutionEngine - Connecting to map-reduce job tracker at: localhost:54311
2014-05-01 15:13:20,647 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
```

Step 5) In Grunt command prompt for Pig, execute below Pig commands in order.

-- A. Load the file containing data.

salesTable = LOAD '/SalesJan2009.csv' USING PigStorage(',') AS (Transaction_date:chararray, Product:chararray,Price:chararray,Payment_Type:chararray,Name:chararray,City:chararray,State:chararray,Country:chararray,Account_Created:chararray,Last_Login:chararray,Latitude:chararray,Longitude:chararray);

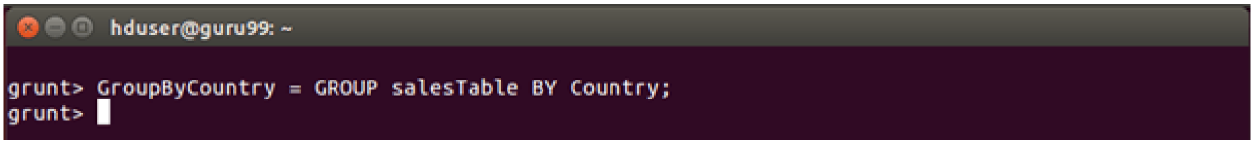
Press Enter after this command.



```
hduser@guru99: ~
grunt> salesTable = LOAD '<Hadoop Home Directory>/SalesJan2009.csv' USING PigStorage('
,') AS (Transaction_date:chararray,Product:chararray,Price:chararray,Payment_Type:char
array,Name:chararray,City:chararray,State:chararray,Country:chararray,Account_Created:
chararray,Last_Login:chararray,Latitude:chararray,Longitude:chararray);
```

-- B. Group data by field Country

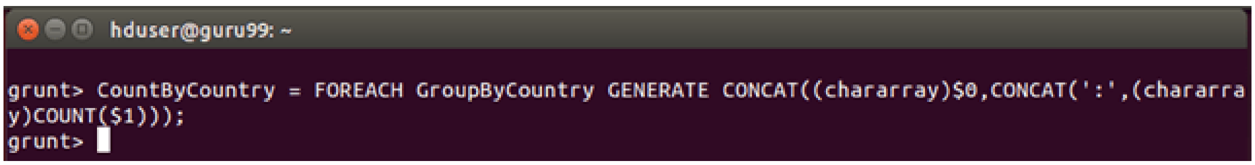
GroupByCountry = GROUP salesTable BY Country;



-- C. For each tuple in **'GroupByCountry'**, generate the resulting string of the form-> Name of Country: No. of products sold

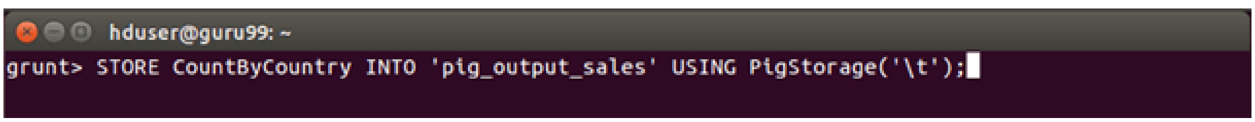
```
CountByCountry = FOREACH GroupByCountry GENERATE CONCAT((chararray)$0,CONC
AT(':',(chararray)COUNT($1)));
```

Press Enter after this command.



-- D. Store the results of Data Flow in the directory **'pig_output_sales'** on HDFS

```
STORE CountByCountry INTO 'pig_output_sales' USING PigStorage('\t');
```

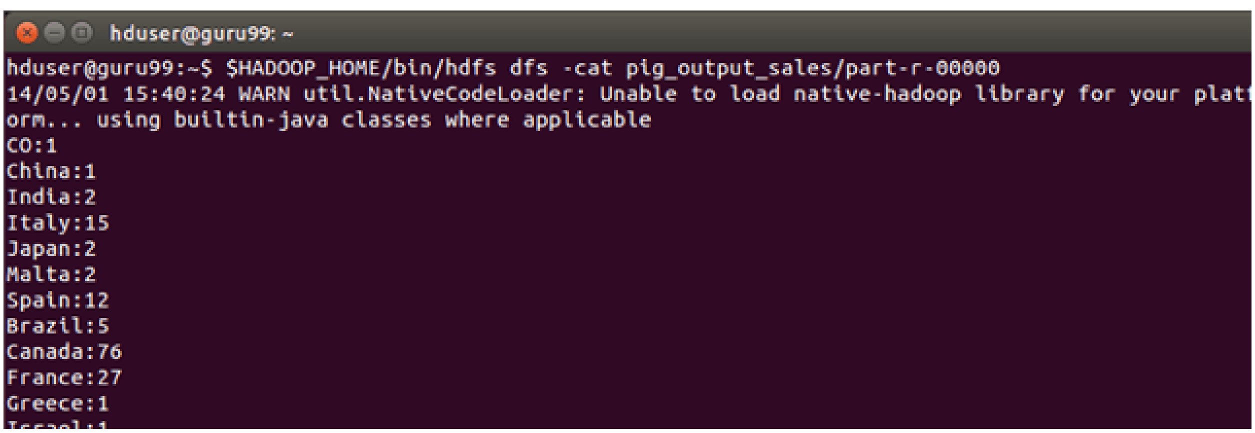


This command will take some time to execute. Once done, you should see the following screen

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMapTime   MaxReduceTime   Mi
nReduceTime     AvgReduceTime   MedianReducetime        Alias   Feature Outputs
job_local1064530209_0001        1       1       0       0       0       0       0       0       0C
ountByCountry,GroupByCountry,salesTable GROUP_BY,COMBINER       hdfs://localhost:54310/user/hduser_/pig_ou
tput_sales,

Input(s):
Successfully read 0 records from: "/SalesJan2009.csv"

Output(s):
Successfully stored 0 records in: "hdfs://localhost:54310/user/hduser_/pig_output_sales"

Counters:
Total records written : 0
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local1064530209_0001


2014-05-06 16:33:21,552 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduc
eLauncher - Success!
```

Step 6) Result can be seen through command interface as,

$HADOOP_HOME/bin/hdfs dfs -cat pig_output_sales/part-r-00000

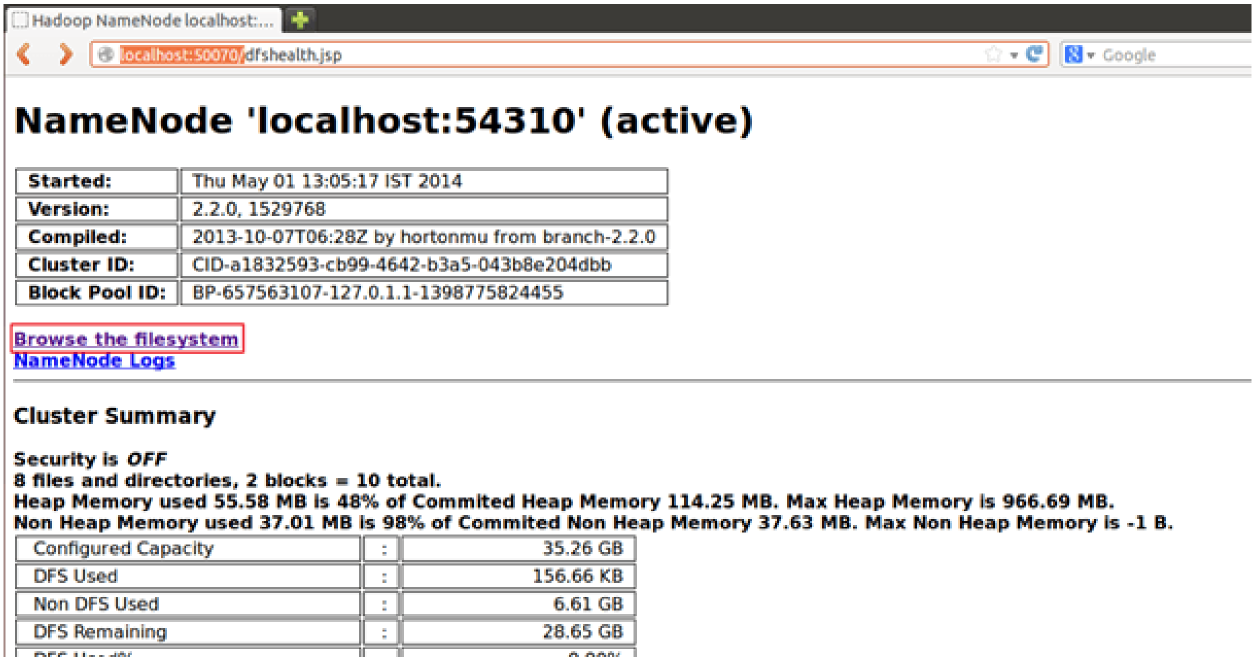


```
hduser@guru99: ~
hduser@guru99:~$ $HADOOP_HOME/bin/hdfs dfs -cat pig_output_sales/part-r-00000
14/05/01 15:40:24 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... using builtin-java classes where applicable
CO:1
China:1
India:2
Italy:15
Japan:2
Malta:2
Spain:12
Brazil:5
Canada:76
France:27
Greece:1
```

Results can also be seen via a web interface as-

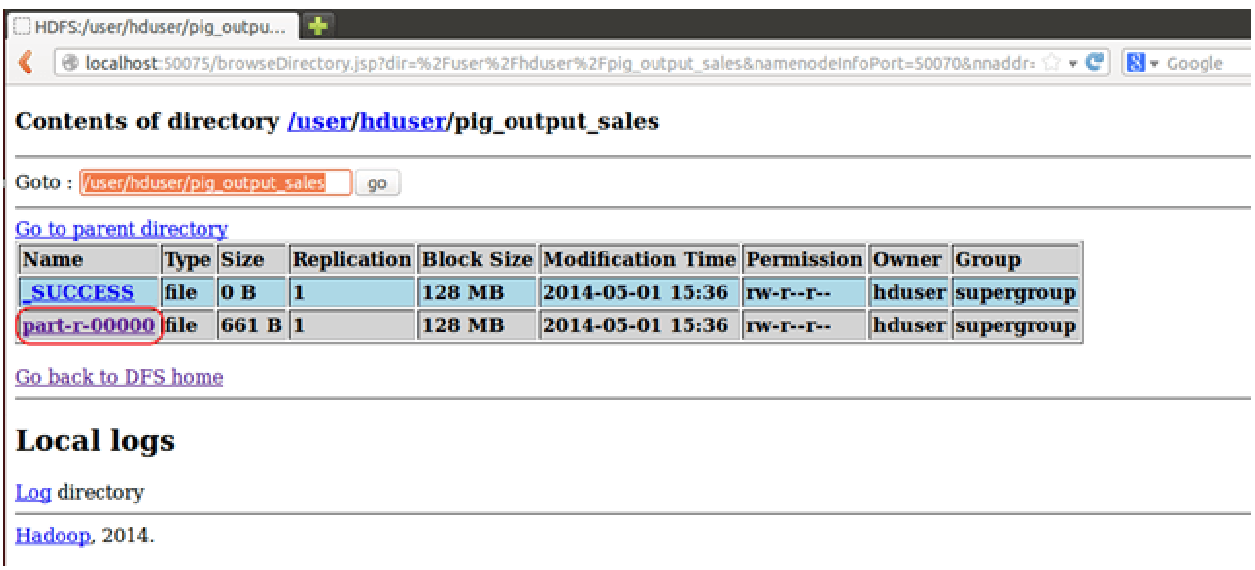
Results through a web interface-

Open http://localhost:50070/ in a web browser.

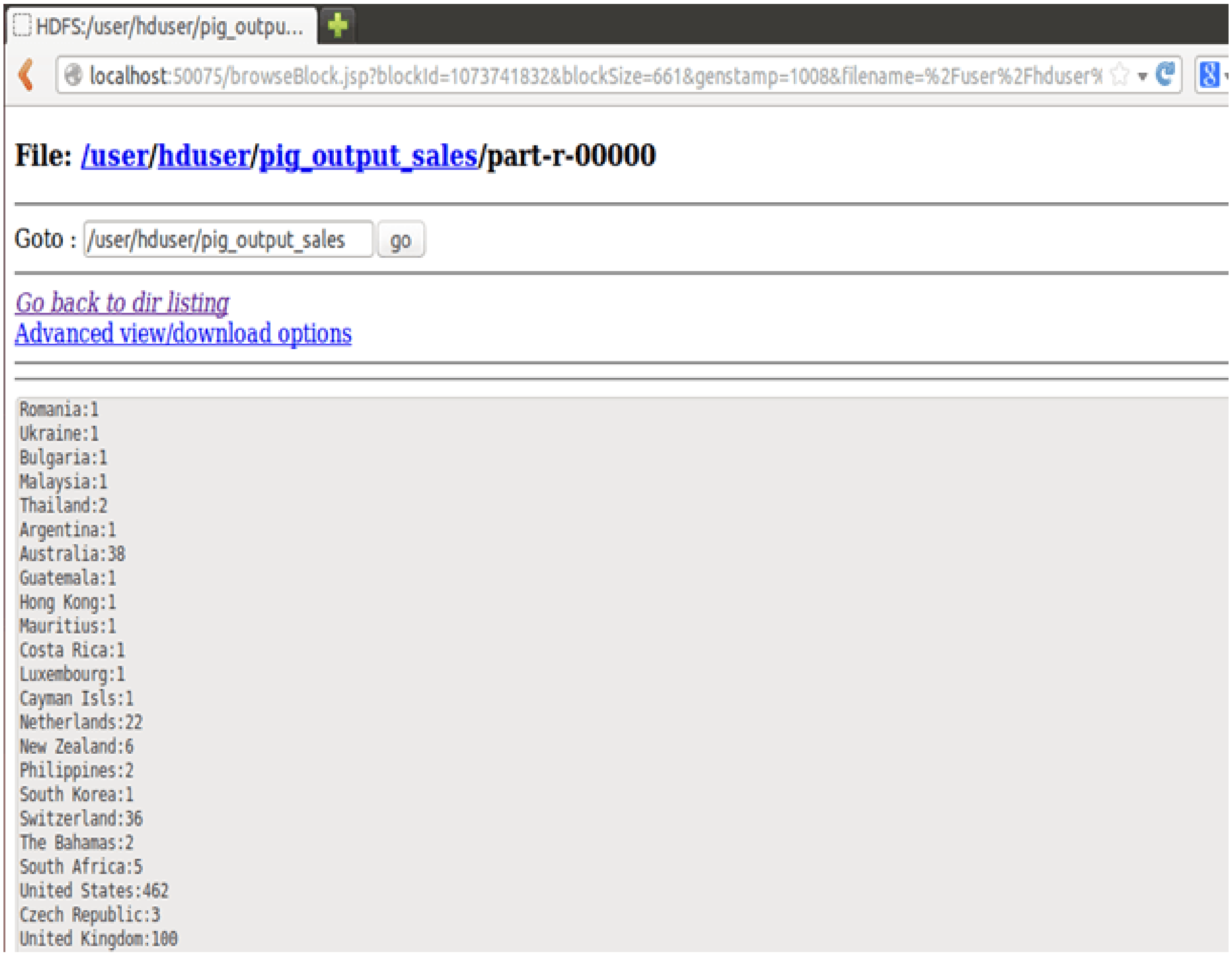


Now select **'Browse the filesystem'** and navigate

upto **/user/hduser/pig_output_sales**



Open **part-r-00000**



Conclusions:

Sample Questions:

1) What apache Pig.
2) Explain execution of apache pig.

EXPERIMENT NO: 06

Title: Study of Hadoop distributed file system.
Aim: Study of architecture & operations of HDFS

Theory:

HDFS:

HDFS is a distributed file system for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. Hadoop comes bundled with **HDFS** (**Hadoop Distributed File Systems**).

When data exceeds the capacity of storage on a single physical machine, it becomes essential to divide it across a number of separate machines. A file system that manages storage specific operations across a network of machines is called a distributed file system. HDFS is one such software.

HDFS Architecture:

HDFS cluster primarily consists of a **NameNode** that manages the file system **Metadata** and a **DataNodes** that stores the **actual data**.

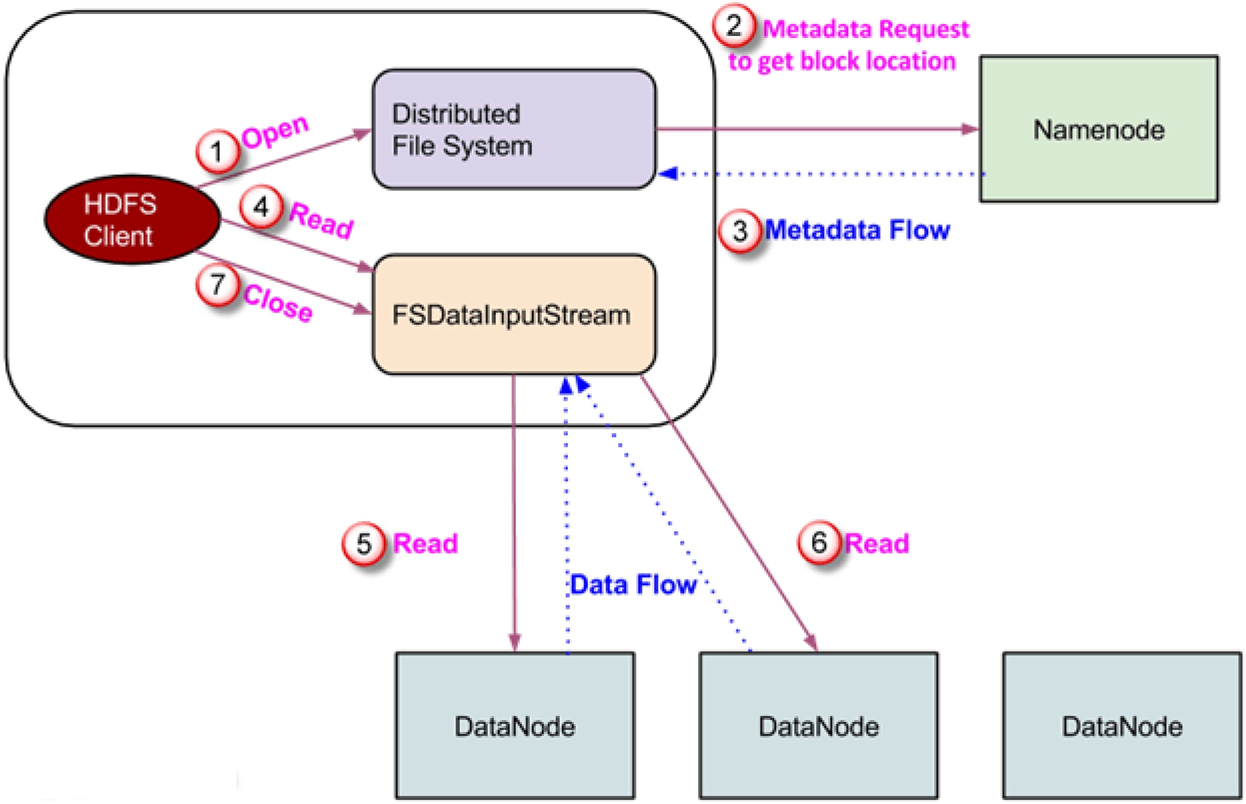
- **NameNode:** NameNode can be considered as a master of the system. It maintains the file system tree and the metadata for all the files and directories present in the system. Two files **'Namespace image'** and the **'edit log'** are used to store metadata information. Namenode has knowledge of all the datanodes containing data blocks for a given file, however, it does not store block locations persistently. This information is reconstructed every time from datanodes when the system starts.
- **DataNode:** DataNodes are slaves which reside on each machine in a cluster and provide the actual storage. It is responsible for serving, read and write requests for the clients.

Read/write operations in HDFS operate at a block level. Data files in HDFS are broken into block-sized chunks, which are stored as independent units. Default block-size is 64 MB.

HDFS operates on a concept of data replication wherein multiple replicas of data blocks are created and are distributed on nodes throughout a cluster to enable high availability of data in the event of node failure.

Read Operation In HDFS:

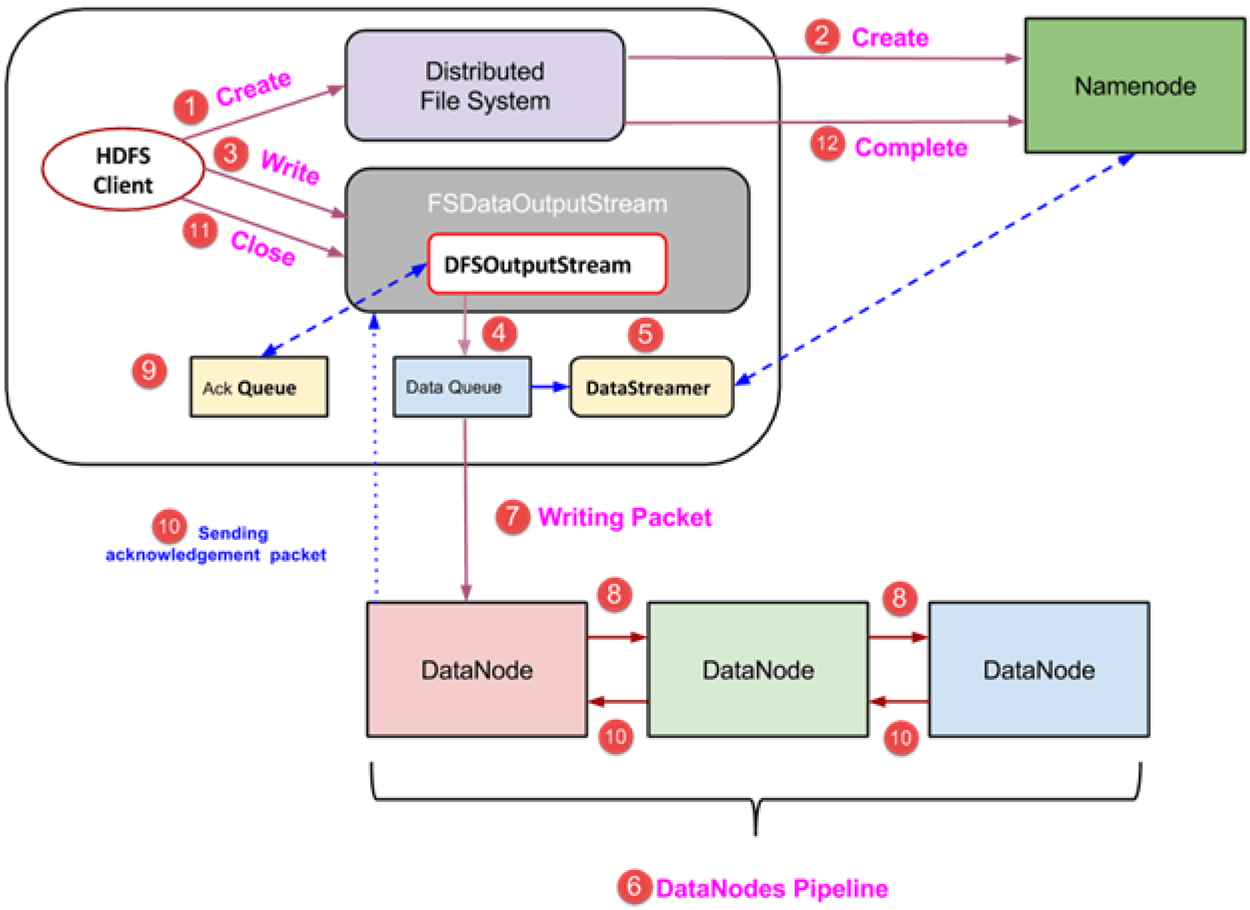
Data read request is served by HDFS, NameNode, and DataNode. Let's call the reader as a 'client'. Below diagram depicts file read operation in Hadoop.



1. A client initiates read request by calling **'open()'** method of FileSystem object; it is an object of type **DistributedFileSystem**.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few blocks of a file.
3. In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back.
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in the above diagram, a client invokes **'read()'** method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of a file.
5. Data is read in the form of streams wherein client invokes **'read()'** method repeatedly. This process of **read()** operation continues till it reaches the end of block.
6. Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once a client has done with the reading, it calls **a close()** method.

Write Operation In HDFS:

In this section, we will understand how data is written into HDFS through files.



1. A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in the above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file creates operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and a client has correct permissions to create a new file. If a file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to the client. Otherwise, the operation succeeds and a new record for the file is created by the NameNode.
3. Once a new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked (step 3 in the diagram).
4. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While the client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are enqueued into a queue which is called as **DataQueue**.
5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen a replication level of 3 and hence there are 3 DataNodes in the pipeline.
7. The DataStreamer pours packets into the first DataNode in the pipeline.

8. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in a pipeline.
9. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.
10. Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
11. After a client is done with the writing data, it calls a close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgment.
12. Once a final acknowledgment is received, NameNode is contacted to tell it that the file write operation is complete.

Conclusion:

Sample Questions:

1) Explain HDFS.

2) Describe read operation of HDFS.

3) Explain write operation of HDFS.

EXPERIMENT NO: 07

Title: Introduction of R Language and Study of datatypes, variables & logical operator in R	
Aim: Study of fundamentals of R Language.	

Theory:

R Software:

R is a programming language and free software developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithms, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred.

R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

Data analysis with R is done in a series of steps; programming, transforming, discovering, modeling and communicate the results

- **Program**: R is a clear and accessible programming tool
- **Transform**: R is made up of a collection of libraries designed specifically for data science
- **Discover**: Investigate the data, refine your hypothesis and analyze them
- **Model**: R provides a wide array of tools to capture the right model for your data
- **Communicate**: Integrate codes, graphs, and outputs to a report with R Markdown or build Shiny apps to share with the world

R used for:

- Statistical inference
- Data analysis
- Machine learning algorithm

Comparison R with other language:

Years ago, R was a difficult language to master. The language was confusing and not as structured as the other programming tools. To overcome this major issue, Hadley Wickham developed a collection of packages called tidyverse. The rule of the game changed for the best. Data manipulation become trivial and intuitive. Creating a graph was not so difficult anymore.

The best algorithms for machine learning can be implemented with R. Packages like Keras and TensorFlow allow to create high-end machine learning technique. R also has a package to perform Xgboost, one the best algorithm for Kaggle competition.

R can communicate with the other language. It is possible to call Python, Java, C++ in R. The world of big data is also accessible to R. You can connect R with different databases like Spark or Hadoop.

Finally, R has evolved and allowed parallelizing operation to speed up the computation. In fact, R was criticized for using only one CPU at a time. The parallel package lets you to perform tasks in different cores of the machine.

Data Types in R:

Following are the Data Types or Data Structures in R Programming:

- Scalars
- Vectors (numerical, character, logical)
- Matrices
- Data frames
- Lists

Basics types:

- 4.5 is a decimal value called **numerics**.
- 4 is a natural value called **integers**. Integers are also numerics.
- TRUE or FALSE is a Boolean value called **logical** binary operators in R.
- The value inside " " or ' ' are text (string). They are called **characters**.
- **Example 1:**

Example 1:

```
# Declare variables of different types

# Numeric

x <- 28

class(x)
```

Output:

```
## [1] "numeric"
```

Example 2:

```
# String

y <- "R is Fantastic"

class(y)
```

Output:

```
## [1] "character"
```

Example 3:

```
# Boolean

z <- TRUE

class(z)
```

Output:

```
## [1] "logical"
```

Variables:

Variables are one of the basic data types in R that store values and are an important component in R programming, especially for a data scientist. A variable in R data types can store a number, an object, a statistical result, vector, dataset, a model prediction basically anything R outputs. We can use that variable later simply by calling the name of the variable.

To declare variable data structures in R, we need to assign a variable name. The name should not have space. We can use _ to connect to words.

To add a value to the variable in data types in R programming, use <- or =.

Here is the syntax:

```
# First way to declare a variable:  use the `<-`

name_of_variable <- value

# Second way to declare a variable:  use the `=`

name_of_variable = value
```

In the command line, we can write the following codes to see what happens:

Example 1:

```
# Print variable x

x <- 42

x
```

Output:

```
## [1] 42
```

Example 2:

```
y  <- 10
y
```

Output:

```
## [1] 10
```

Example 3:

```
# We call x and y and apply a subtraction
x-y
```

Output:

```
## [1] 32
```

Vectors:

A vector is a one-dimensional array. We can create a vector with all the basic R data types we learnt before. The simplest way to build vector data structures in R, is to use the c command.

Example 1:

```
# Numerical
vec_num <- c(1, 10, 49)
vec_num
```

Output:

```
## [1]  1 10 49
```

Example 2:

```
# Character

vec_chr <- c("a", "b", "c")

vec_chr
```

Output:

```
## [1] "a" "b" "c"
```

Example 3:

```
# Boolean

vec_bool <-  c(TRUE, FALSE, TRUE)

vec_bool
```

Output:

```
##[1] TRUE FALSE TRUE
```

We can do arithmetic calculations on vector binary operators in R.

Example 4:

```
# Create the vectors

vect_1 <- c(1, 3, 5)

vect_2 <- c(2, 4, 6)

# Take the sum of A_vector and B_vector

sum_vect <- vect_1 + vect_2

# Print out total_vector

sum_vect
```

Output:

```
[1]  3  7 11
```

Example 5:

In R, it is possible to slice a vector. In some occasion, we are interested in only the first five rows of a vector. We can use the [1:5] command to extract the value 1 to 5.

```
# Slice the first five rows of the vector

slice_vector <- c(1,2,3,4,5,6,7,8,9,10)

slice_vector[1:5]
```

Output:

```
## [1] 1 2 3 4 5
```

Example 6:

The shortest way to create a range of value is to use the: between two numbers. For instance, from the above example, we can write c(1:10) to create a vector of value from one to ten.

```
# Faster way to create adjacent values

c(1:10)
```

Output:

```
## [1]  1  2  3  4  5  6  7  8  9 10
```

Arithmetic Operators

We will first see the basic arithmetic operators in R data types. Following are the arithmetic and boolean operators in R programming which stand for:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication

/	Division
^ or **	Exponentiation

Example 1:

```
# An addition

3 + 4
```

Output:

```
## [1] 7
```

You can easily copy and paste the above R code into Rstudio Console. The **output** is displayed after the character #. For instance, we write the code print('Guru99') the output will be ##[1] Guru99.

The ## means we print an output and the number in the square bracket ([1]) is the number of the display

The sentences starting with # **annotation**. We can use # inside an R script to add any comment we want. R won't read it during the running time.

Example 2:

```
# A multiplication

3*5
```

Output:

```
## [1] 15
```

Example 3:

```
# A division

(5+5)/2
```

Output:

```
## [1] 5
```

Example 4:

```
# Exponentiation

2^5
```

Output:

Example 5:

```
## [1] 32

# Modulo

28%%6
```

Output:

```
## [1] 4
```

Logical Operators:

With logical operators, we want to return values inside the vector based on logical conditions. Following is a detailed list of logical operators of data types in R programming

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x	y
x & y	x AND y
isTRUE(x)	Test if X is TRUE

Logical Operators in R

The logical statements in R are wrapped inside the []. We can add many conditional statements as we like but we need to include them in a parenthesis. We can follow this structure to create a conditional statement:

```
variable_name[(conditional_statement)]
```

With variable_name referring to the variable, we want to use for the statement. We create the logical statement i.e. variable_name > 0. Finally, we use the square bracket to finalize the logical statement. Below, an example of a logical statement.

Example 1:

```
# Create a vector from 1 to 10

logical_vector <- c(1:10)

logical_vector>5
```

Output:

```
## [1]FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

In the output above, R reads each value and compares it to the statement logical_vector>5. If the value is strictly superior to five, then the condition is TRUE, otherwise FALSE. R returns a vector of TRUE and FALSE.

Example 2:

In the example below, we want to extract the values that only meet the condition 'is strictly superior to five'. For that, we can wrap the condition inside a square bracket precede by the vector containing the values.

```
# Print value strictly above 5

logical_vector[(logical_vector>5)]
```

Output:

```
## [1]  6  7  8  9 10
```

Example 3:

```
# Print 5 and 6
```

```
logical_vector <- c(1:10)

logical_vector[(logical_vector>4) & (logical_vector<7)]
```

Output:

```
## [1] 5 6
```

Conclusion:

Sample Questions:

1) Describe R Environment.

2) Explain uses of R

3) Explain advantages of R.

4) Describe Data types in R.

5) Explain logical operator in R.

6) Explain variables in R.

EXPERIMENT NO: 08

Title: Study of Line graph in R
Aim: Understanding of data visualization in R.

Theory:

R – Line Graphs:

A line graph is a chart that is used to display information in the form of a series of data points. It utilizes points and lines to represent change over time. Line graphs are drawn by plotting different points on their X coordinates and Y coordinates, then by joining them together through a line from beginning to end. The graph represents different values as it can move up and down based on the suitable variable.

R – Line Graphs:

The plot() function in R is used to create the line graph.

Syntax: plot(v, type, col, xlab, ylab)

Parameters:

- v: This parameter is a contains only the numeric values
- type: This parameter has the following value:
- “p” : This value is used to draw only the points.
- “l” : This value is used to draw only the lines.
- “o”: This value is used to draw both points and lines
- xlab: This parameter is the label for x axis in the chart.
- ylab: This parameter is the label for y axis in the chart.
- main: This parameter main is the title of the chart.
- col: This parameter is used to give colors to both the points and lines.

Creating a Simple Line Graph:

Approach: In order to create a line graph:

It is created using the type parameter as “o” and input vector.

Below code to describe the line graph.

Example:

R

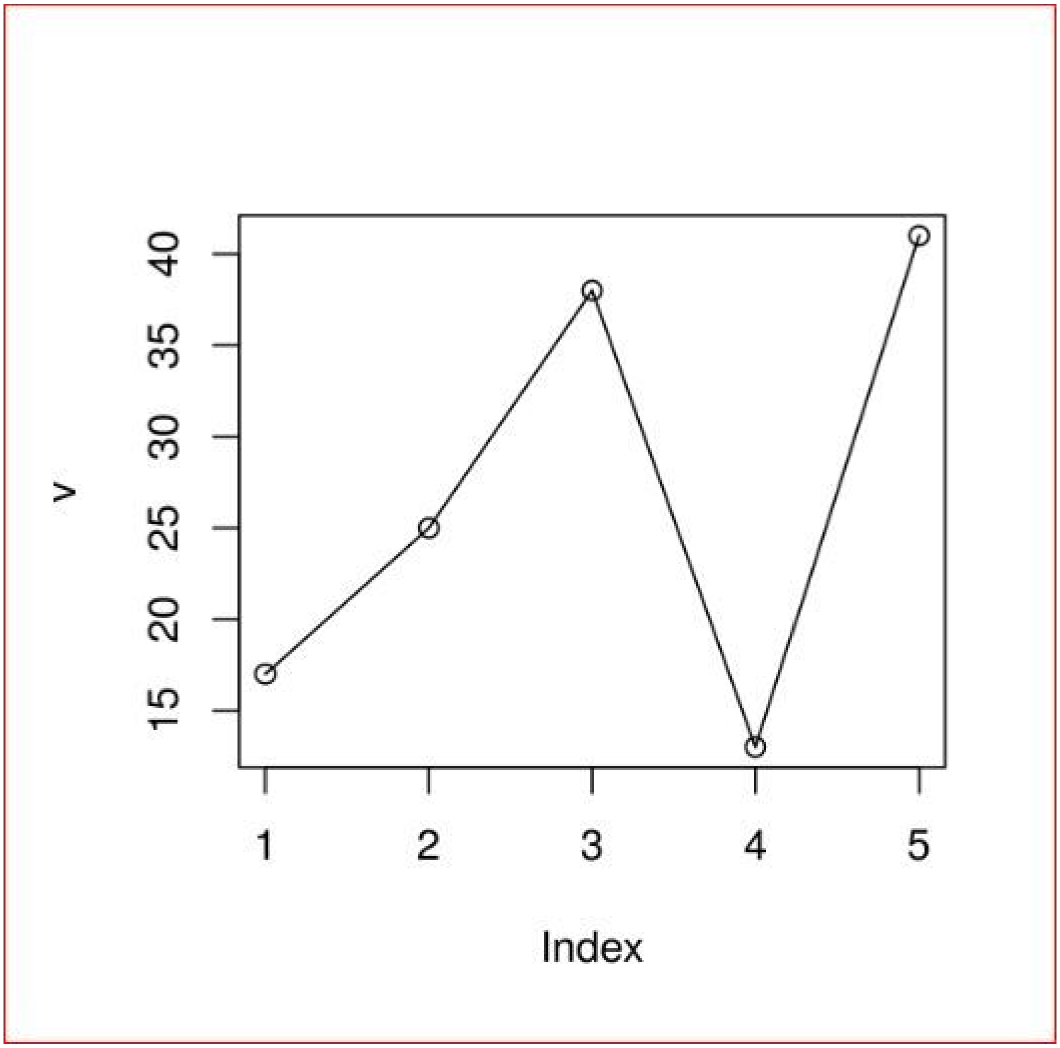
Create the data for the chart.

v <- c(17, 25, 38, 13, 41)

Plot the bar chart.

plot(v, type = "o")

Output:



Adding Title, Color and Labels in Line Graphs in R

Approach: To create a colored and labeled line chart.

Take all parameters which are required to make line chart by giving a title to the chart and add labels to the axes.

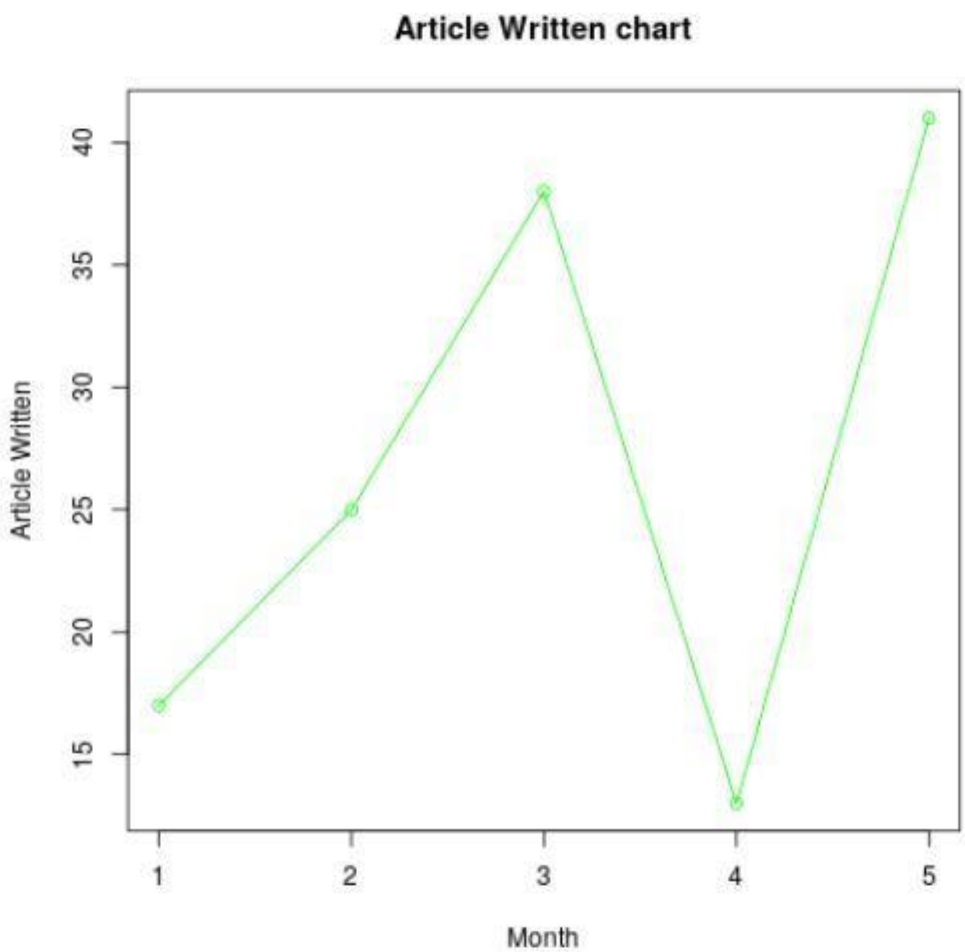
We can add more features by adding more parameters with more colors to the points and lines.

Example:

R

```
# Create the data for the chart.
v <- c(17, 25, 38, 13, 41)

# Plot the bar chart.
plot(v, type = "o", col = "green",
   xlab = "Month", ylab = "Article Written",
   main = "Article Written chart")
```

Output:



Multiple Lines in a Line Graph in R Programming Language

Approach: To create multiple line graphs.

In above example, we created line graphs by only one line in each graph.

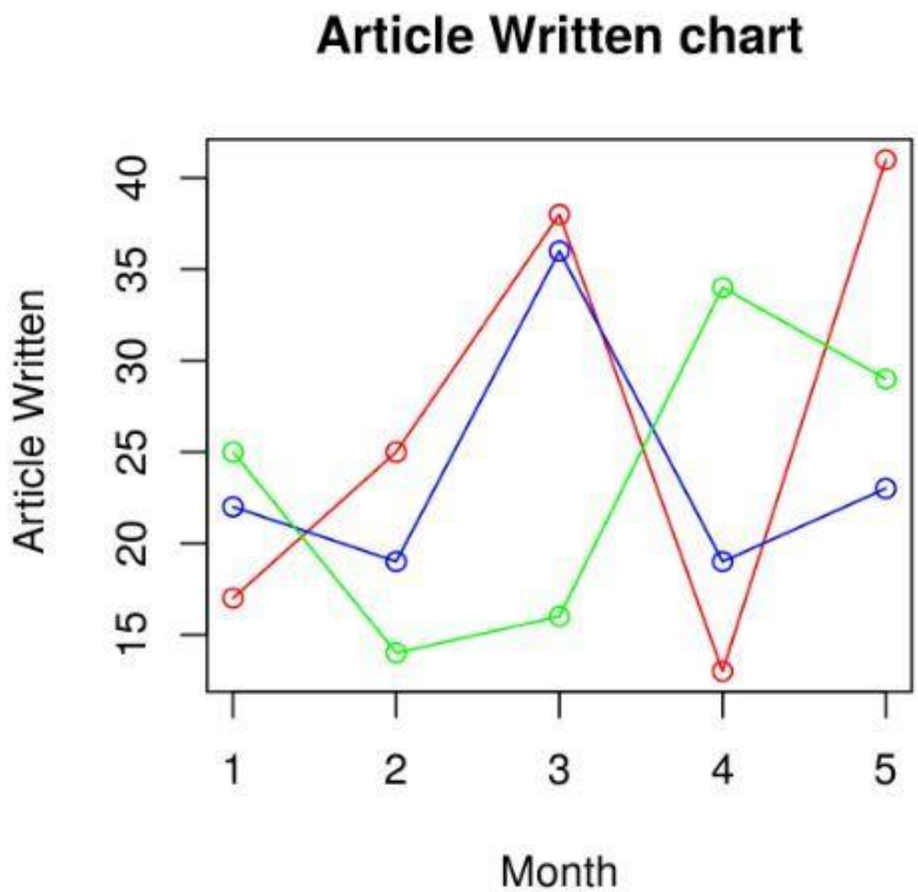
Now creating multiple lines to describe it more clearly.

Example:

R

```
# Create the data for the chart.
v <- c(17, 25, 38, 13, 41)
t <- c(22, 19, 36, 19, 23)
m <- c(25, 14, 16, 34, 29)

# Plot the bar chart.
plot(v, type = "o", col = "red",
    xlab = "Month", ylab = "Article Written ",
    main = "Article Written chart")

lines(t, type = "o", col = "blue")
lines(m, type = "o", col = "green")
```

Output: When we execute the above code, it shows the following result-



Conclusions:

Sample Questions:

1) Describe Data frame in R.

2) Explain line graph in R.

EXPERIMENT NO: 09

Title: Study of Bar chart in R
Aim: Understanding of data visualization in R.

Theory:

R – Bar Charts:

A bar chart is a pictorial representation of data that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. In other words, it is the pictorial representation of dataset. These data sets contain the numerical values of variables that represent the length or height.

R uses the function barplot() to create bar charts. Here, both vertical and Horizontal bars can be drawn.

Syntax:

barplot(H, xlab, ylab, main, names.arg, col)

Parameters:

- H: This parameter is a vector or matrix containing numeric values which are used in bar chart.
- xlab: This parameter is the label for x axis in bar chart.
- ylab: This parameter is the label for y axis in bar chart.
- main: This parameter is the title of the bar chart.
- names.arg: This parameter is a vector of names appearing under each bar in bar chart.
- col: This parameter is used to give colors to the bars in the graph.

Creating a Simple Bar Chart:

Approach: In order to create a Bar Chart:

A vector (H <- c(Values…)) is taken which contain numeral values to be used.

This vector H is plot using barplot().

Example:

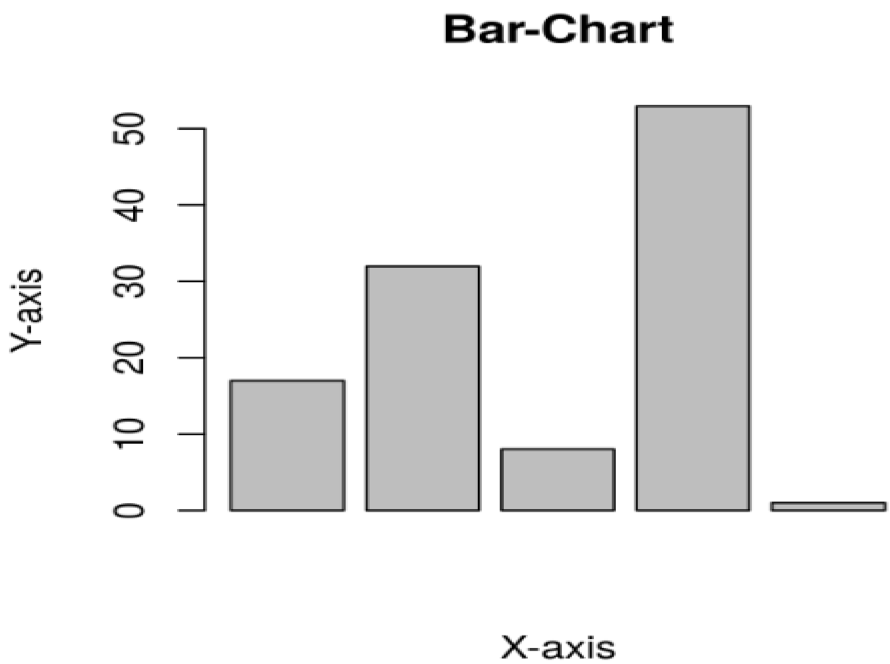
Create the data for the chart

A <- c(17, 32, 8, 53, 1)

Plot the bar chart

barplot(A, xlab = "X-axis", ylab = "Y-axis", main ="Bar-Chart")

Output:



Creating a Horizontal Bar Chart:

Approach: To create a horizontal bar chart:

Take all parameters which are required to make simple bar chart.

Now to make it horizontal new parameter is added.

barplot(A, horiz=TRUE)

Example: Creating a horizontal bar chart

Create the data for the chart

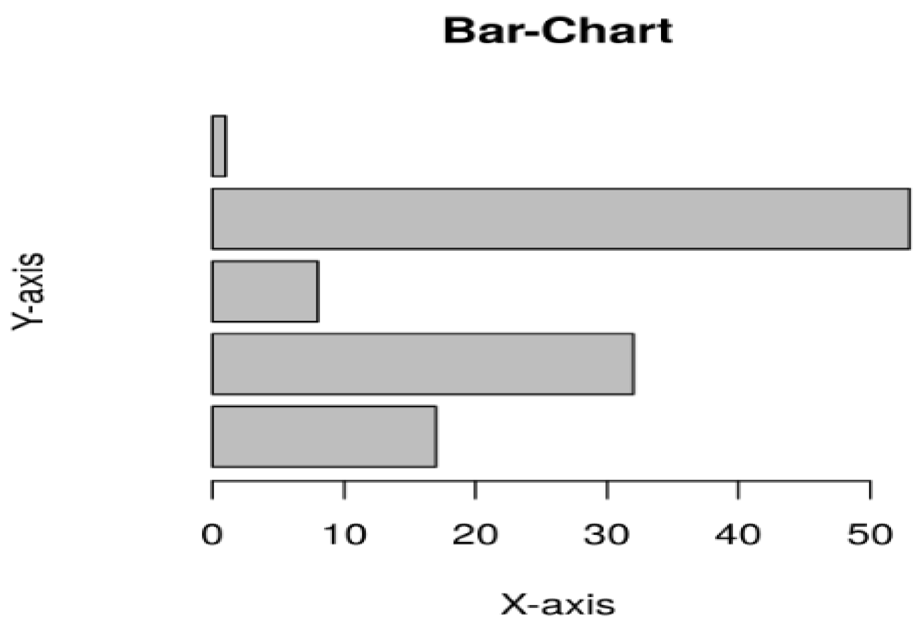
A <- c(17, 32, 8, 53, 1)

Plot the bar chart

barplot(A, horiz = TRUE, xlab = "X-axis",

ylab = "Y-axis", main ="Bar-Chart")

Output:



Adding Label, Title and Color in the BarChart:

Label, title and colors are some properties in the bar chart which can be added to the bar by adding and passing an argument.

Approach:

To add the title in bar chart.

barplot(A, main = title_name)

X-axis and Y-axis can be labeled in bar chart. To add the label in bar chart.

barplot(A, xlab= x_label_name, ylab= y_label_name)

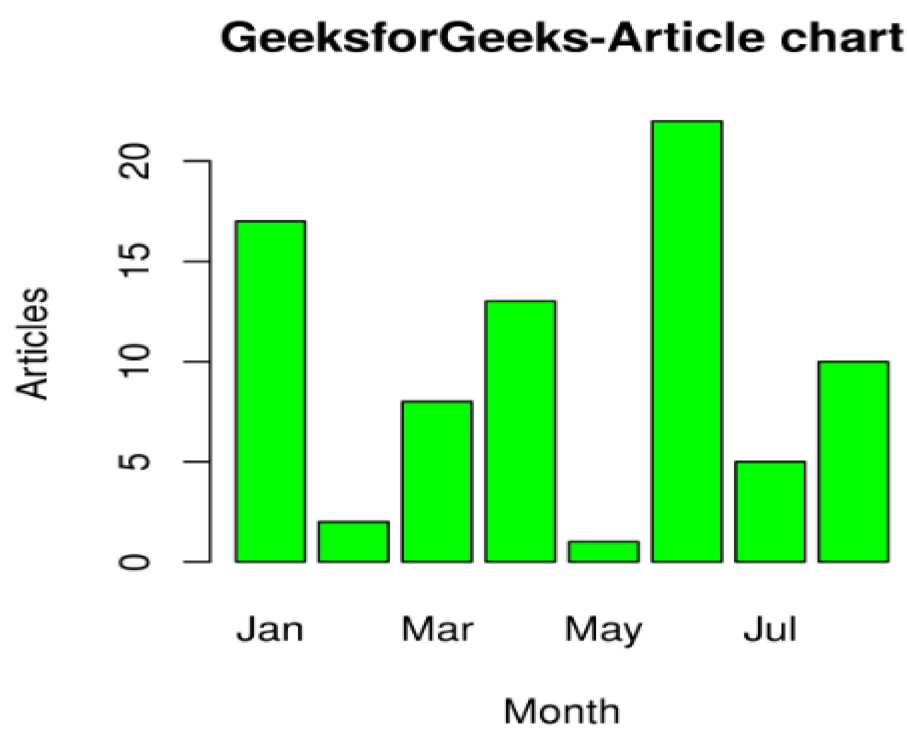
To add the color in bar chart.

barplot(A, col=color_name)

Example :

```
# Create the data for the chart
A <- c(17, 2, 8, 13, 1, 22)
B <- c("Jan", "feb", "Mar", "Apr", "May", "Jun")

# Plot the bar chart
barplot(A, names.arg = B, xlab ="Month",
        ylab ="Articles", col ="green",
        main ="GeeksforGeeks-Article chart")
```

Output:



Creating Stacked and Grouped Bar Chart:

The bar chart can be represented in two form group of bars and stacked.

Approach:

Take a vector value and make it matrix M which to be grouped or stacked. Making of matrix can be done by.

M <- matrix(c(values...), nrow = no_of_rows, ncol = no_of_column, byrow = TRUE)

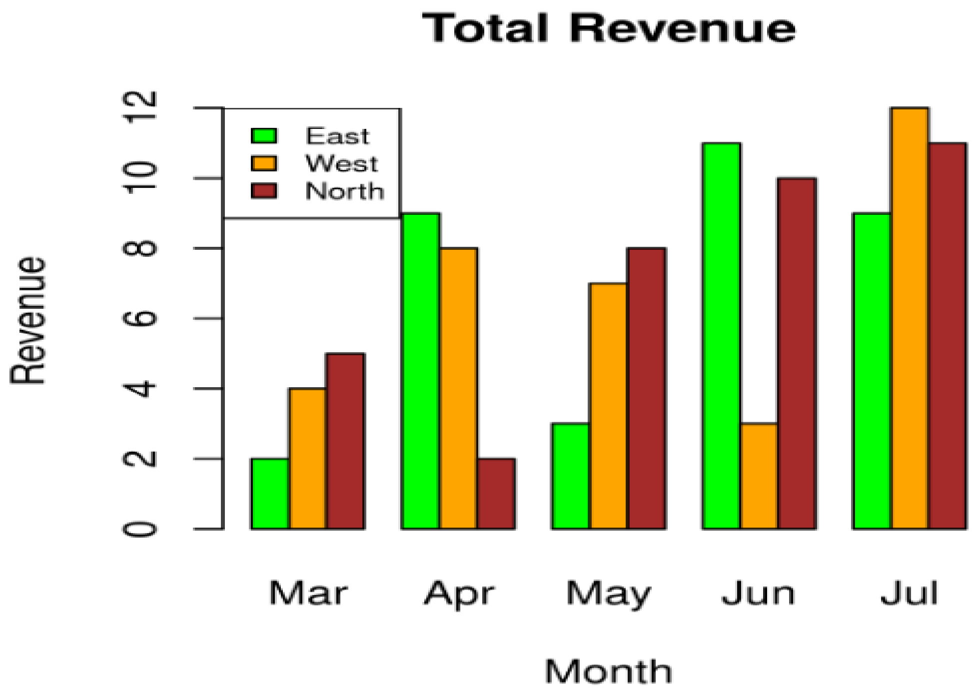
To display the bar explicitly we can use the beside parameter.

barplot(beside=TRUE)

Example 1:

```
colors = c("green", "orange", "brown")

months <- c("Mar", "Apr", "May", "Jun", "Jul")

regions <- c("East", "West", "North")


# Create the matrix of the values.

Values <- matrix(c(2, 9, 3, 11, 9, 4, 8, 7, 3, 12, 5, 2, 8, 10, 11),

           nrow = 3, ncol = 5, byrow = TRUE)


# Create the bar chart

barplot(Values, main = "Total Revenue", names.arg = months,

                 xlab = "Month", ylab = "Revenue",

                 col = colors, beside = TRUE)


# Add the legend to the chart

legend("topleft", regions, cex = 0.7, fill = colors)
```

Output:

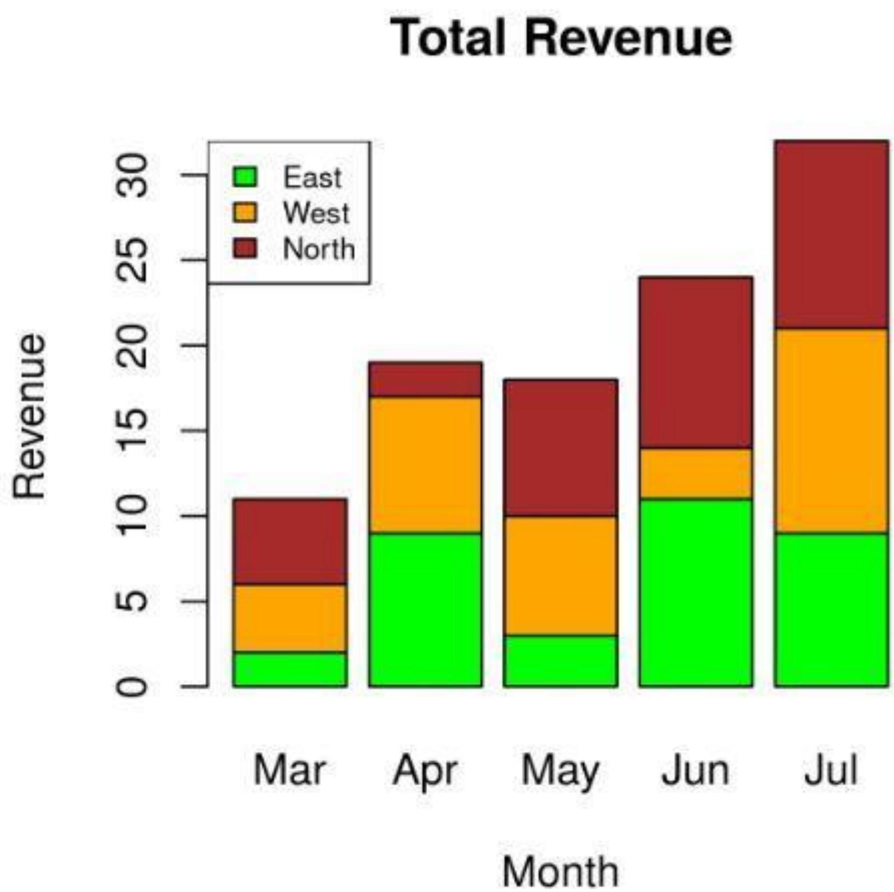


Example 2:

```
colors = c("green", "orange", "brown")

months <- c("Mar", "Apr", "May", "Jun", "Jul")

regions <- c("East", "West", "North")


# Create the matrix of the values.

Values <- matrix(c(2, 9, 3, 11, 9, 4, 8, 7, 3, 12, 5, 2, 8, 10, 11),

                 nrow = 3, ncol = 5, byrow = TRUE)


# Create the bar chart

barplot(Values, main = "Total Revenue", names.arg = months,
```

```
xlab = "Month", ylab = "Revenue", col = colors)


# Add the legend to the chart
legend("topleft", regions, cex = 0.7, fill = colors)
```

Output:



Conclusions:

Sample Questions:

1) Describe data visualization in R.

2) Explain bar charts in R.

EXPERIMENT NO: 10

Title: Study of Pie chart in R
Aim: Understanding of data visualization in R.

Theory:

R – Pie Charts:

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data. A circular chart cuts in a form of radii into segments describing relative frequencies or magnitude also known as a circle graph.

R – Pie Charts:

R Programming Language uses the function pie() to create pie charts. It takes positive numbers as a vector input.

Syntax: pie(x, labels, radius, main, col, clockwise)

Parameters:

- x: This parameter is a vector that contains the numeric values which are used in the pie chart.
- labels: This parameter gives the description to the slices in pie chart.
- radius: This parameter is used to indicate the radius of the circle of the pie chart.(value between -1 and +1).
- main: This parameter is represents title of the pie chart.
- clockwise: This parameter contains the logical value which indicates whether the slices are drawn clockwise or in anti clockwise direction.
- col: This parameter give colors to the pie in the graph.

Creating a simple pie chart:

To create a simple pie chart:

By using the above parameters, we can draw a pie chart.

It can be described by giving simple labels.

Example:

Create data for the graph.

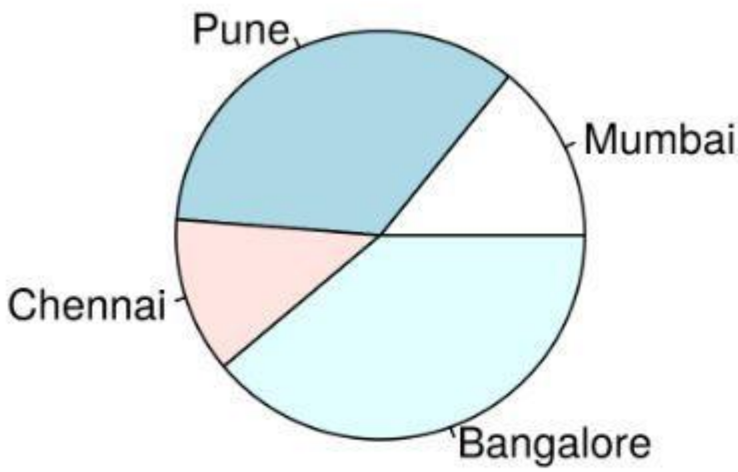
geeks<- c(23, 56, 20, 63)

labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")

Plot the chart.

pie(geeks, labels)

Output:



Pie chart including the title and colors:

To create color and title pie chart.

Take all parameters which are required to make pie chart by giving a title to the chart and add labels.

We can add more features by adding more parameters with more colors to the points.

Example:

Create data for the graph.

geeks<- c(23, 56, 20, 63)

```
labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")

# Plot the chart with title and rainbow
# color pallet.
pie(geeks, labels, main = "City pie chart",
          col = rainbow(length(geeks)))
```

Output:

Slice Percentage & Chart Legend:

To create chart legend and slice percentage, we can plot by doing the below methods.

There are two more properties of the pie chart:

slice percentage

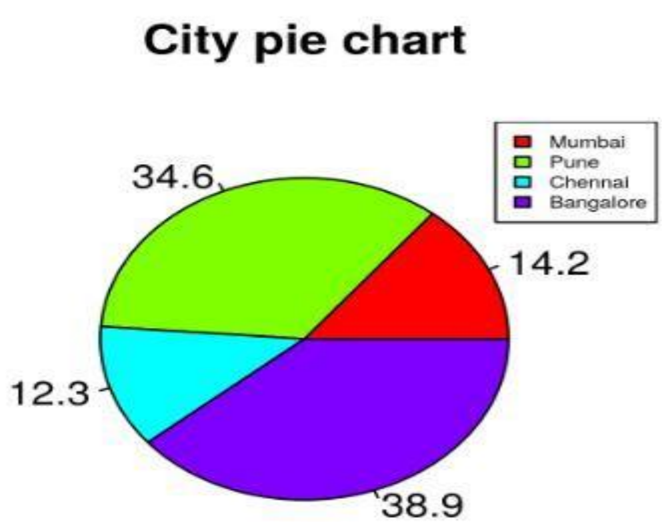
chart legend.

We can show the chart in the form of percentages as well as add legends.

Example:

```
# Create data for the graph.
geeks <- c(23, 56, 20, 63)
labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")

piepercent<- round(100 * geeks / sum(geeks), 1)

# Plot the chart.
pie(geeks, labels = piepercent,
   main = "City pie chart", col = rainbow(length(geeks)))
legend("topright", c("Mumbai", "Pune", "Chennai", "Bangalore"),
                  cex = 0.5, fill = rainbow(length(geeks)))
```

Output:

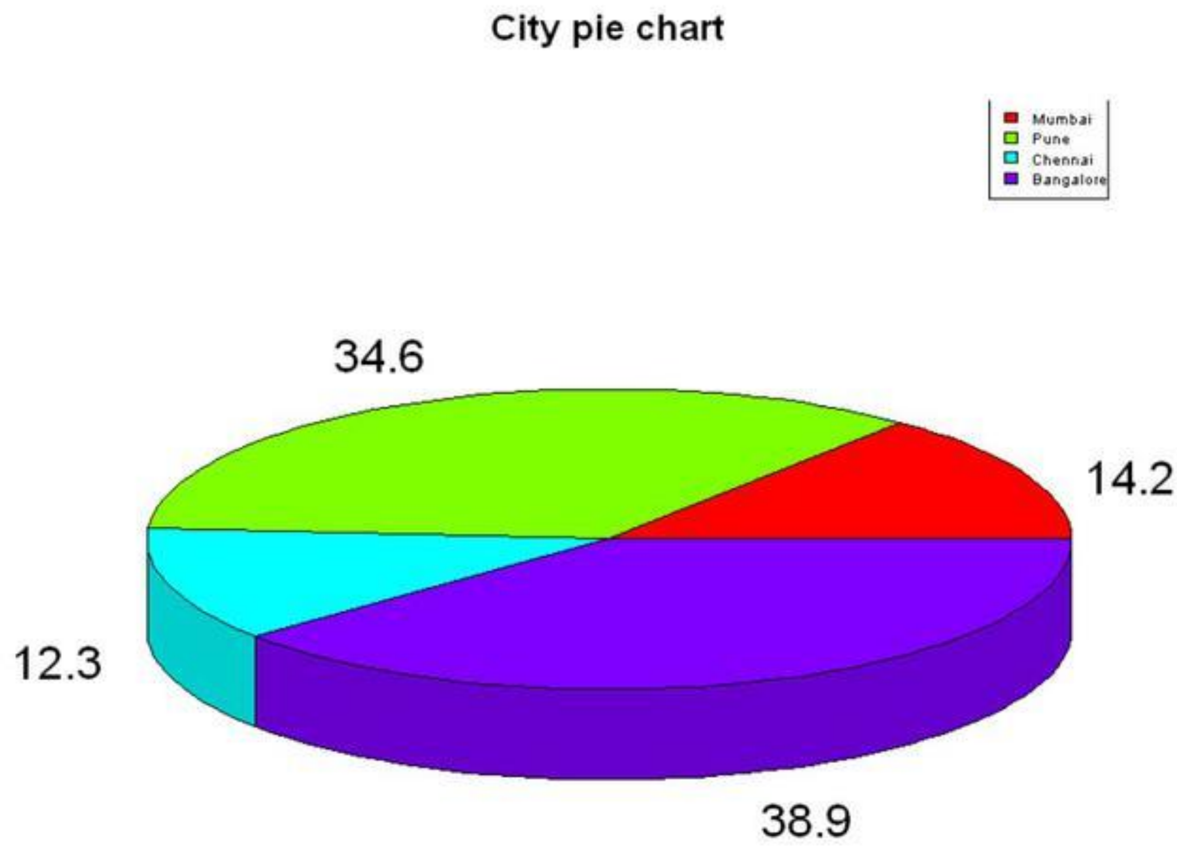


3D Pie Chart:

Here we are going to create a 3D Pie chart using plotrix package and then we will use pie3D() function to plot 3D plot.

```
# Get the library.

library(plotrix)


# Create data for the graph.

geeks <- c(23, 56, 20, 63)

labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")


piepercent<- round(100 * geeks / sum(geeks), 1)

# Plot the chart.

pie3D(geeks, labels = piepercent,

    main = "City pie chart", col = rainbow(length(geeks)))

legend("topright", c("Mumbai", "Pune", "Chennai", "Bangalore"),

                cex = 0.5, fill = rainbow(length(geeks)))
```

Output:



Conclusions:

Sample Questions:

1) Describe data visualization in R.

2) Explain Pie charts in R.

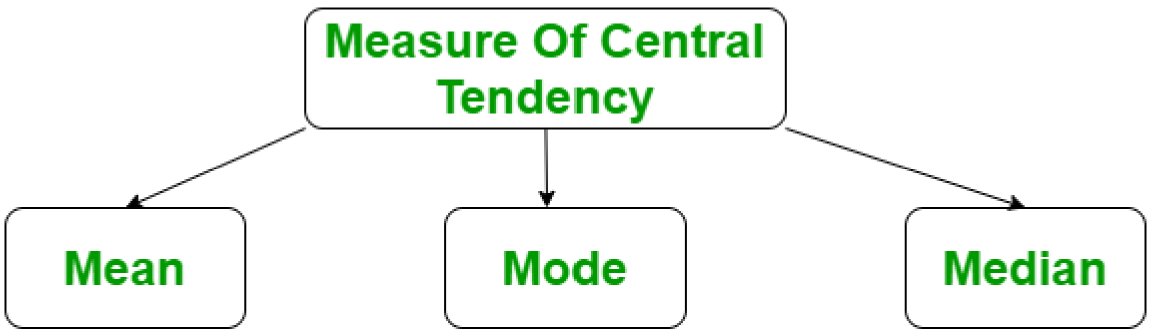
EXPERIMENT NO: 11

Title: Study of central tendency in R	
Aim: Understanding of statistics in R.	

Theory:

The measure of central tendency in R Language represents the whole set of data by a single value. It gives us the location of central points. There are three main measures of central tendency:

- Mean
- Median
- Mode



Prerequisite:

Before doing any computation, first of all, we need to prepare our data, save our data in external .txt or .csv files and it's a best practice to save the file in the current directory. After that import, your data into R as follow:

Get the CSV file here.

```
# R program to import data into R


# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors=F)

# Print the first 6 rows

print(head(myData))
```

Output:

```
  Product Age Gender Education MaritalStatus Usage Fitness Income Miles
1   TM195  18   Male        14        Single     3       4  29562   112
2   TM195  19   Male        15        Single     2       3  31836    75
3   TM195  19 Female        14     Partnered     4       3  30699    66
4   TM195  19   Male        12        Single     3       3  32973    85
5   TM195  20   Male        13     Partnered     4       2  35247    47
6   TM195  20 Female        14     Partnered     3       3  32973    66
```

Mean in R Programming Language:

It is the sum of observations divided by the total number of observations. It is also defined as average which is the sum divided by count.

$$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

Where, n = number of terms

Example:

```
# R program to illustrate
# Descriptive Analysis


# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors=F)


# Compute the mean value
mean = mean(myData$Age)
print(mean)
```

Output:

[1] 28.78889

Median in R Programming Language:

It is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is odd then the center element is median and if it is even then the median would be the average of two central elements.

Odd	Even
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

Where n = number of terms

Syntax: median(x, na.rm = False)

Where, X is a vector and na.rm is used to remove missing value

Example:

```
# R program to illustrate

# Descriptive Analysis


# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                 stringsAsFactors=F)


# Compute the median value

median = median(myData$Age)

print(median)
```

Output:

[1] 26

Mode in R Programming Language:

It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency of all data points is the same. Also, we can have more than one mode if we encounter two or more data points having the same frequency. There is no inbuilt function for finding mode in R, so we can create our own function for finding the mode or we can use the package called modest.

Creating user-defined function for finding Mode

There is no in-built function for finding mode in R. So let's create a user-defined function that will return the mode of the data passed. We will be using the table() method for this as it creates a categorical representation of data with the variable names and the frequency in the form of a table. We will sort the column Age column in descending order and will return the 1 value from the sorted values.

Example: Finding mode by sorting the column of the dataframe

```
# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                  stringsAsFactors=F)


mode = function(){

   return(sort(-table(myData$Age))[1])

}


mode()
```

Output:

```
25: -25
```

Using Modest Package:

We can use the modest package of the R. This package provides methods to find the mode of the univariate data and the mode of the usual probability distribution.

Example:

```
# R program to illustrate
# Descriptive Analysis

# Import the library
library(modest)

# Import the data using read.csv()
myData = read.csv("CardioGoodFitness.csv",
                  stringsAsFactors=F)

# Compute the mode value
mode = mfv(myData$Age)
print(mode)
```

Output:

```
[1] 25
```

Conclusions:

Sample Questions:

1) Describe mean, median and mode.
2) Explain measure of central tendency in R.

EXPERIMENT NO: 12

Title: Study of linear regression in R	
Aim: Understanding of statistics in R.	

Theory:

R - Linear Regression:

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is −

y = ax + b

Following is the description of the parameters used −

- y is the response variable.
- x is the predictor variable.
- a and b are constants which are called the coefficients.

Steps to Establish a Regression:

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

Carry out the experiment of gathering a sample of observed values of height and corresponding weight.

Create a relationship model using the lm() functions in R.

Find the coefficients from the model created and create the mathematical equation using these

Get a summary of the relationship model to know the average error in prediction. Also called residuals.

To predict the weight of new persons, use the predict() function in R.

Input Data

Below is the sample data representing the observations −

Values of height

151, 174, 138, 186, 128, 136, 179, 163, 152, 131

Values of weight.

63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function:

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for lm() function in linear regression is −

lm(formula,data)

Following is the description of the parameters used −

formula is a symbol presenting the relation between x and y.

data is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

Live Demo

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

print(relation)
```

When we execute the above code, it produces the following result −

Call:

lm(formula = y ~ x)

Coefficients:

(Intercept)   x

   -38.4551   0.6746

Get the Summary of the Relationship

Live Demo

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))

When we execute the above code, it produces the following result −

Call:

lm(formula = y ~ x)

Residuals:

    Min   1Q   Median   3Q   Max

-6.3002   -1.6629 0.0412   1.8944 3.9775

Coefficients:

            Estimate Std. Error t value Pr(>|t|)

```
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function:

Syntax

The basic syntax for predict() in linear regression is −

predict(object, newdata)

Following is the description of the parameters used −

object is the formula which is already created using the lm() function.

newdata is the vector containing the new value for predictor variable.

Predict the weight of new persons

Live Demo

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)
```

```
# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

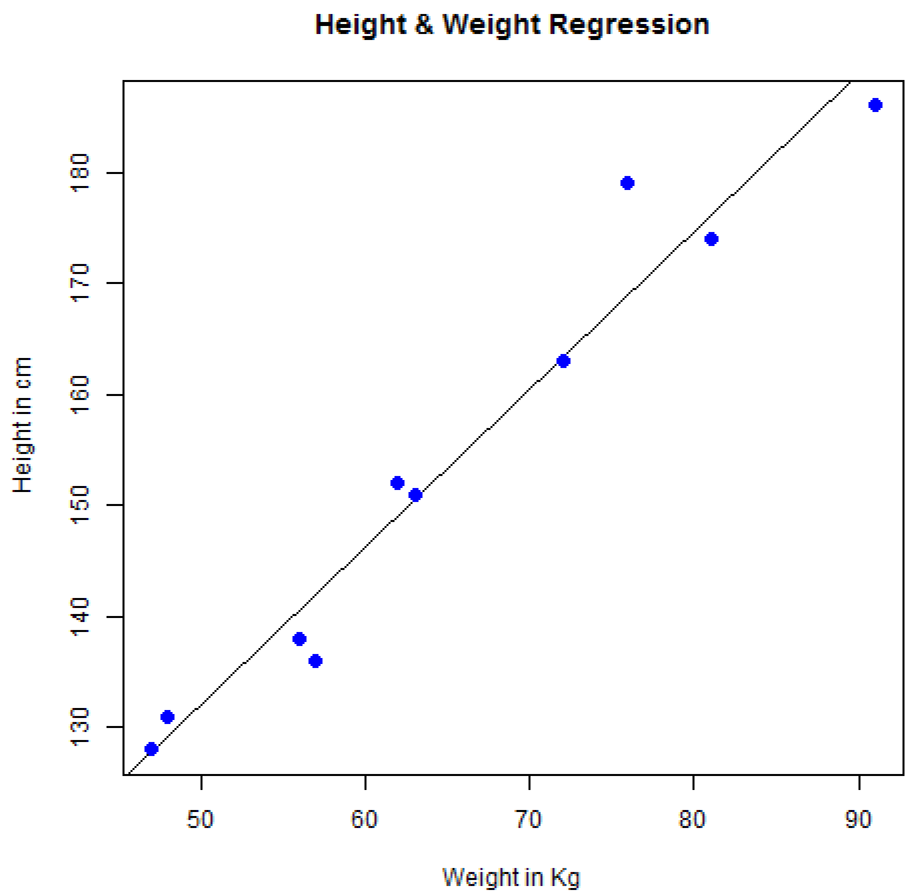
When we execute the above code, it produces the following result −

```
       1
76.22869
```

Visualize the Regression Graphically:

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")
```

When we execute the above code, it produces the following result –



Conclusions:

Sample Questions:

1) Describe linear regression.

2) Explain procedure to perform linear regression in R.