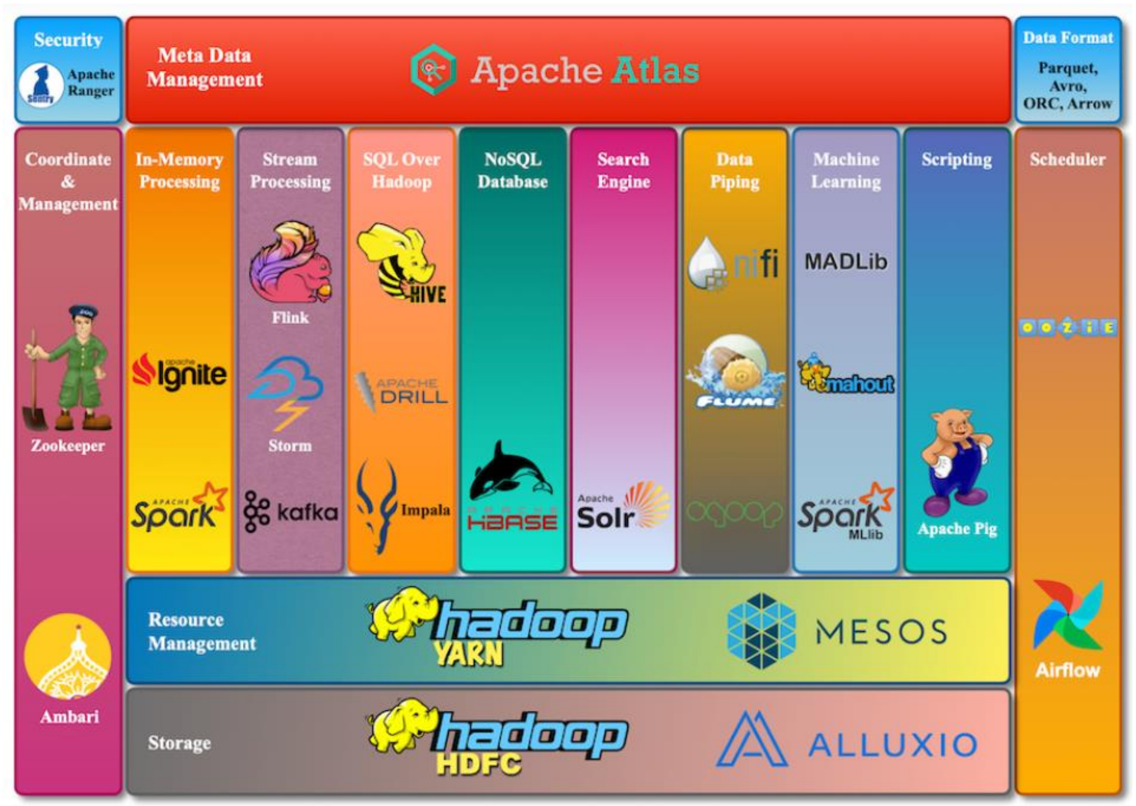
PROGRAM 1

Study of Big Data Framework and Tools: Apache Hadoop, Apache Spark, Apache Flink, Apache Mahout, and MOA.

Introduction to Big Data Analytic s: Big Data, Scalability and Parallel Processing, Designing Data Architecture, Data Sources, Quality, Pre-Processing and Storing, Data Storage and Analysis, Big Data processing platforms: HADOOP, SPARK, FLINK, and MOA, Challenges of Conventional Systems, Big Data Analytics Applications and Case Studies.



Hadoop is great for reliable, scalable, distributed calculations. However, it can also be exploited as common-purpose file storage. It can store and process petabytes of data. This solution consists of three key components:

- HDFS file system, responsible for the storage of data in the Hadoop cluster;
- MapReduce system, intended to process large volumes of data in a cluster;
- YARN, a core that handles resource management.

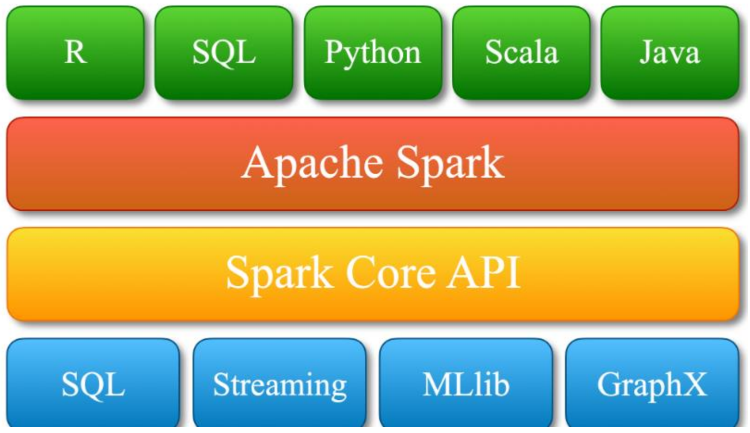
Hadoop can store and process many petabytes of info, while the fastest processes in Hadoop only take a few seconds to operate. It also forbids any edits to the data, already stored in the HDFS system during the processing.

Apache Spark Ecosystem

Apache Spark is a lightning-fast centralized analytics engine. It allows usages of more than 80 high-level operators to create applications. Apache Spark can be deployed on a stand-alone system or a cluster of nodes.

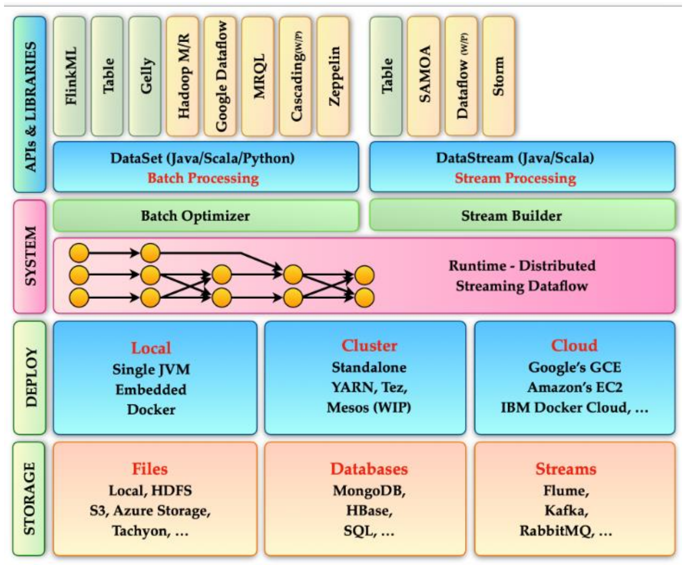
Apache Spark provides a robust set of components to perform different types of operations such as GraphX, Spark Streaming, Spark Core, SparkR, Spark SQL, and MLlib.

Now let us see each component of Apache Spark in the following section.



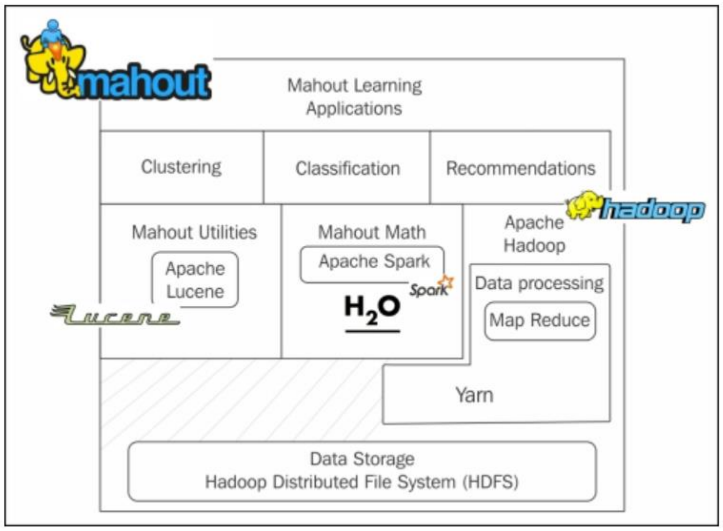
Apache Flink

Apache Flink is a stream processing framework, which is an open-source software delivered by Apache Software Foundation. Flink is a distributed processing tool that is used to process bounded data unbounded stream. We can deploy Apache Flink in almost all distributed computing frameworks and achieve an in-memory computation. Compared to Hadoop in which programs are divided into two parts and then process sequentially, all operations in the Apache Flink process in parallel make it lightning fast. Apache Flink can be easily deployed and run on various cluster management frameworks and the storage system which are Apache Kafka, Tez, YARN, and so on.



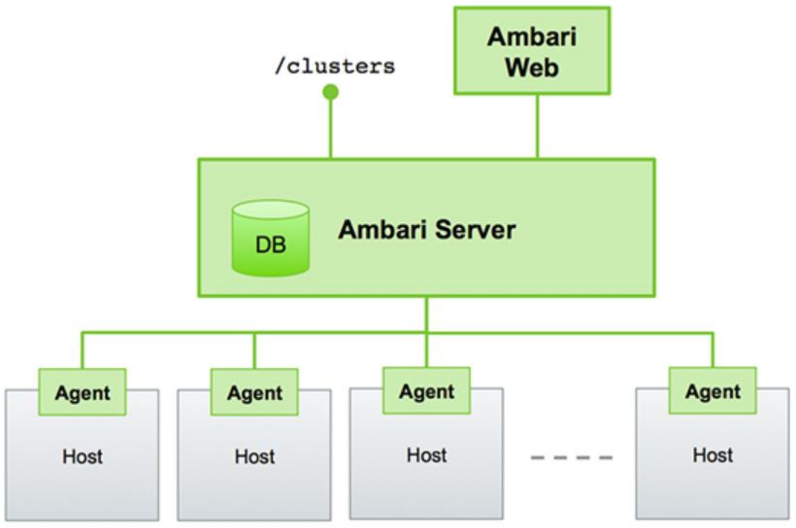
Apache Mahout

Apache Mahout uses Apache Hadoop, which is a distributed computing framework, to achieve scalability. The following figure clearly shows the place where Apache Hadoop fits into Apache Mahout:



Apache Mahout is a library of scalable machine-learning algorithms, implemented on top of Apache Hadoop and using the MapReduce paradigm. Machine learning is a discipline of artificial intelligence focused on enabling machines to learn without being explicitly programmed, and it is commonly used to improve future performance based on previous outcomes. The Apache Mahout™ project's goal is to build an environment for quickly creating scalable performant machine learning applications.

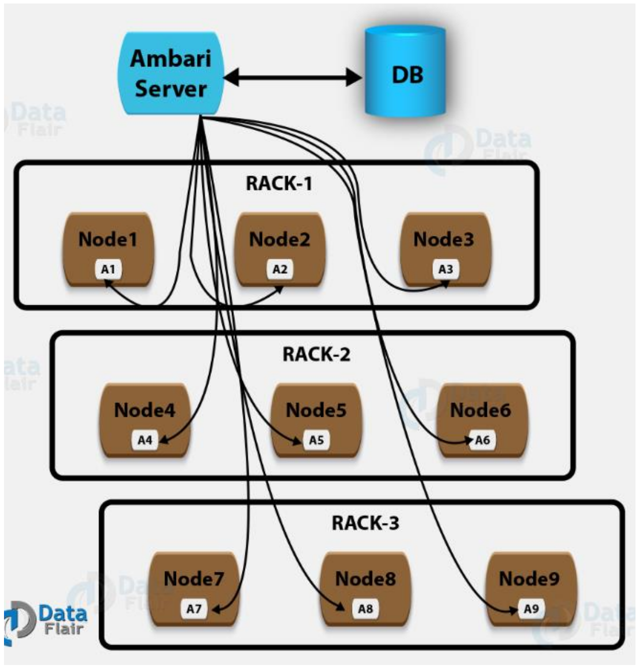
Apache Ambari



Basically, to access cluster information and perform cluster operations, Ambari Web calls the Ambari REST API (accessible from the Ambari Server). However, the application authenticates to the Ambari Server, after authenticating to Ambari Web.
Further, by using the REST API, communication between the browser and server occurs asynchronously.
In addition, there is a REST API in Ambari which is accessed by Web UI, that resets the session timeout. Hence, we can say Ambari Web sessions do not timeout automatically. And, after a period of inactivity, we can configure Ambari to timeout.

Working of Apache Ambari

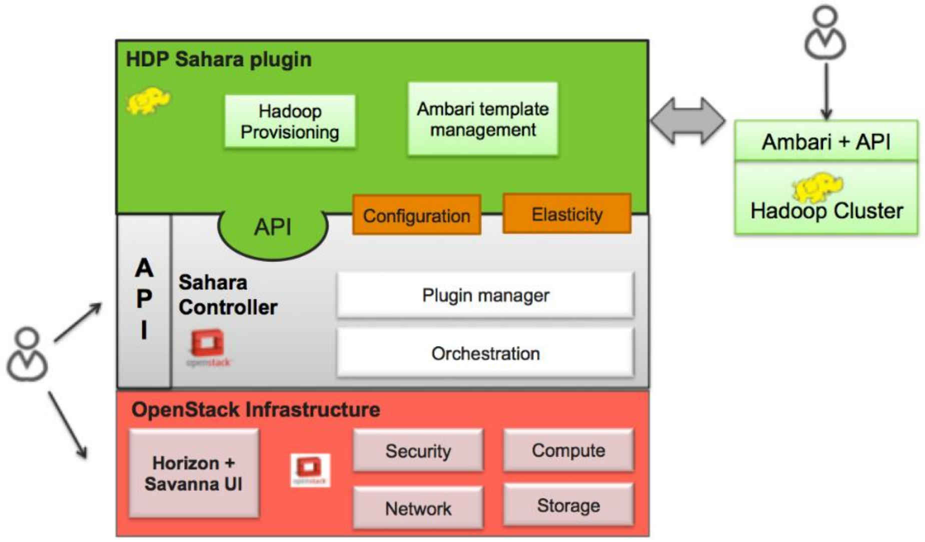
Ambari Architecture is of master/slave type architecture. So, to perform certain actions and report back the state of every action, the master node instructs the slave nodes.
Although, for keeping track of the state of the infrastructure, the master node is responsible. But for this process, a database server is used by the master node, that can be further configured during setup time.
Now, we can see the high-level architecture of Ambari by below diagram which also shows how Ambari works:



There are the following applications in Apache Ambari, at the core:

- Ambari server
- The Ambari agent
- Ambari web UI
- Database

Apache Ambari



The Hortonworks Data Platform (HDP) sahara plugin provides a way to provision HDP clusters on OpenStack using templates in a single click and in an easily repeatable fashion. As seen from the architecture diagram below, the sahara controller serves as the glue between Hadoop and OpenStack.

The HDP plugin mediates between the sahara controller and Apache Ambari in order to deploy and configure Hadoop on OpenStack. Core to the HDP Plugin is Apache Ambari which is used as the orchestrator for deploying HDP on OpenStack. The HDP plugin can make use of Ambari Blueprints for cluster provisioning.

HDP is a very popular big data platform or at the very least it was one of the two most popular on-prem Hadoop platforms. Hadoop is a collection of open-source applications which uses many servers to solve problems involving lots of structured and unstructured data. It is a framework based on distributed storage (HDFS) and the processing engine called MapReduce. HDFS basically breaks files into large blocks and distributes them across the data nodes in the cluster. HDFS by It then manipulates the data in each node in parallel and reduces the data set by sending the data to the Reducer program to get the final aggregated result. HDP is enterprise-ready, open-source Apache Hadoop distribution based on a centralized architecture. HDP addresses the complete needs of data at rest, powers real-time customer applications, and delivers robust big data analytics that accelerate decision making and innovation. The latest version HDP delivers new capabilities for the enterprise to enable agile application deployment, new machine learning/deep learning workloads, real-time data warehousing, & security and governance.

PROGRAM 2

Install Hadoop and Configure single

Steps to Install Hadoop 3 on Ubuntu 18.04.4

Step 1. Please download Hadoop 3.3.4 from the below link.

```
$wget https://archive.apache.org/dist/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
```

Step 2. Install Java 8 using the below command

```
$sudo apt-get update
$sudo apt-get install default-jdk
```

Check java is installed or not

Step 3. Create Hadoop Group and user

```
$sudo addgroup hadoop
$sudo adduser --ingroup hadoop hduser
$groups hduser
```

Step 4. Install SSH

```
$sudo apt-get install ssh
$check ssh and sshd
```

Step 5. Create and setup SSH

```
$su hduser
$ssh-keygen
$cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
$ssh localhost
```

Step 6. Install Hadoop Download Hadoop

```
$tar xvzf hadoop-3.3.4.tar.gz
$sudo -v
$su mainuser
```

```
$sudo adduser hduser sudo
$su hduser
$sudo mkdir -p /usr/local/hadoop
$cd hadoop-3.3.4
$sudo mv * /usr/local/hadoop
$sudo chown -R hduser:hadoop /usr/local/hadoop
```

Step 7. Setup Configuration Files

Open New Terminal Window

```
update-alternatives --config java
$sudo nano ~/.bashrc

                #HADOOP VARIABLES START
                export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-i486
                export HADOOP_INSTALL=/usr/local/hadoop
                export PATH=$PATH:$HADOOP_INSTALL/bin
                export PATH=$PATH:$HADOOP_INSTALL/sbin
                export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
                export HADOOP_COMMON_HOME=$HADOOP_INSTALL
                export HADOOP_HDFS_HOME=$HADOOP_INSTALL
                export YARN_HOME=$HADOOP_INSTALL
                export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
                export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
                #HADOOP VARIABLES END
$source ~/.bashrc

$sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
export JAVA_HOME=”JAVA PATH”
$sudo mkdir -p /app/hadoop/tmp
$sudo chown hduser:hadoop /app/hadoop/tmp
```

```
$sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
```

```
<configuration>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/app/hadoop/tmp</value>
                <description>A base for other temporarydirectories.</description>
        </property>

        <property>
                <name>fs.default.name</name>
                <value>hdfs://localhost:54310</value>
                <description>The name of the default file system. A URI whose scheme and authority
                        determine the FileSystem implementation. The uri's scheme determines the
                        config property (fs.SCHEME.impl) naming the FileSystem implementation
                        class. used to The uri's authority is determine the host, port, etc. for a
                        filesystem.</description>
        </property>
</configuration>
```

```
$sudo nano /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

```
<configuration>

        <property>

                <name>mapred.job.tracker</name>

                <value>localhost:54311</value>

                <description>The host and port that the MapReduce job tracker runs at. If "local", the

                        jobs are run in-process as a single map and reduce task.

                </description>

        </property>

</configuration>
```

```
$sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
$sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
$sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

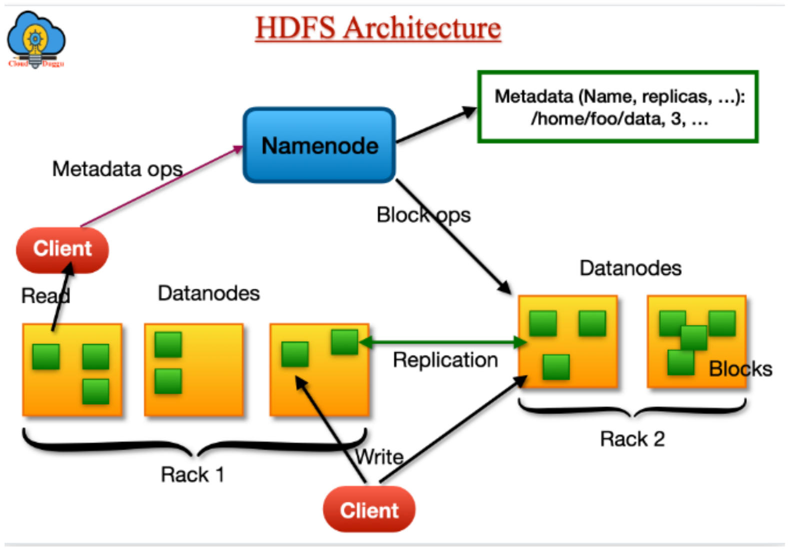
```
$sudo nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
                <description>Default block replication. The actual number of replications can be
                                specifie when the file is created. The default is used if replication is not
                                specified in create time.
                </description>
        </property>

        <property>
                <name>dfs.block.size</name>
                <value>1048576</value>
        </property>

        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
        </property>

        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
        </property>
</configuration>


$hadoop namenode -format
$su mainuser
$cd /usr/local/hadoop/sbin
$ls
$sudo su hduser
$ start-dfs.sh
$start-yarn.sh
$jps
$stop-yarn.sh
$stop-dfs.sh
```

PROGRAM 3

Learn HDFS and execute general and user commands.

The objective of this experiment is to provide a complete overview of Hadoop HDFS command references like writing down the file from local disk to HDFS and from HDFS to local disk and other useful commands as well.

HDFS Architecture

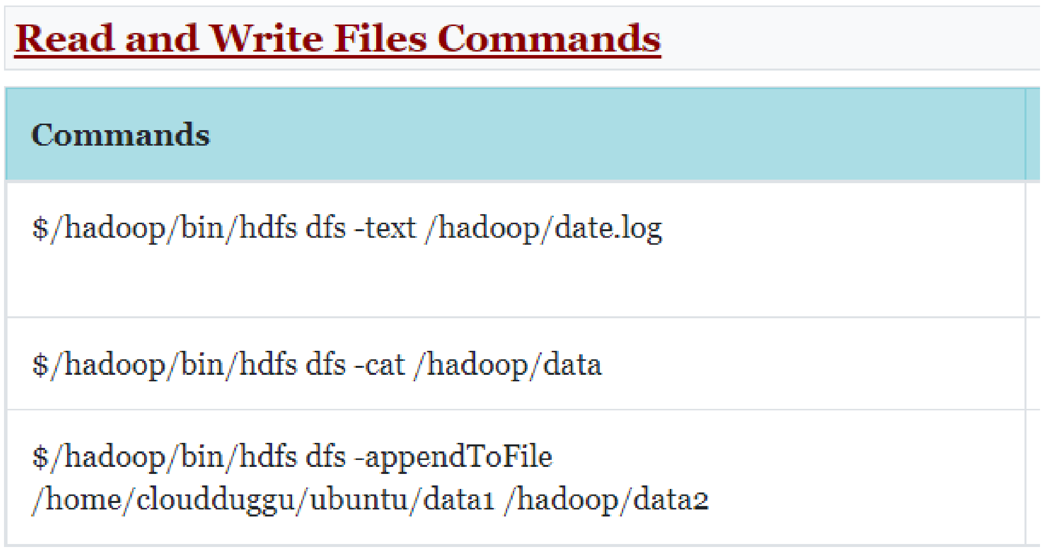


HDFS File Listing Commands

HDFS commands to manage file listing, read /write operations, upload/download files, file management, ownership management, filesystem management, and administration Commands.

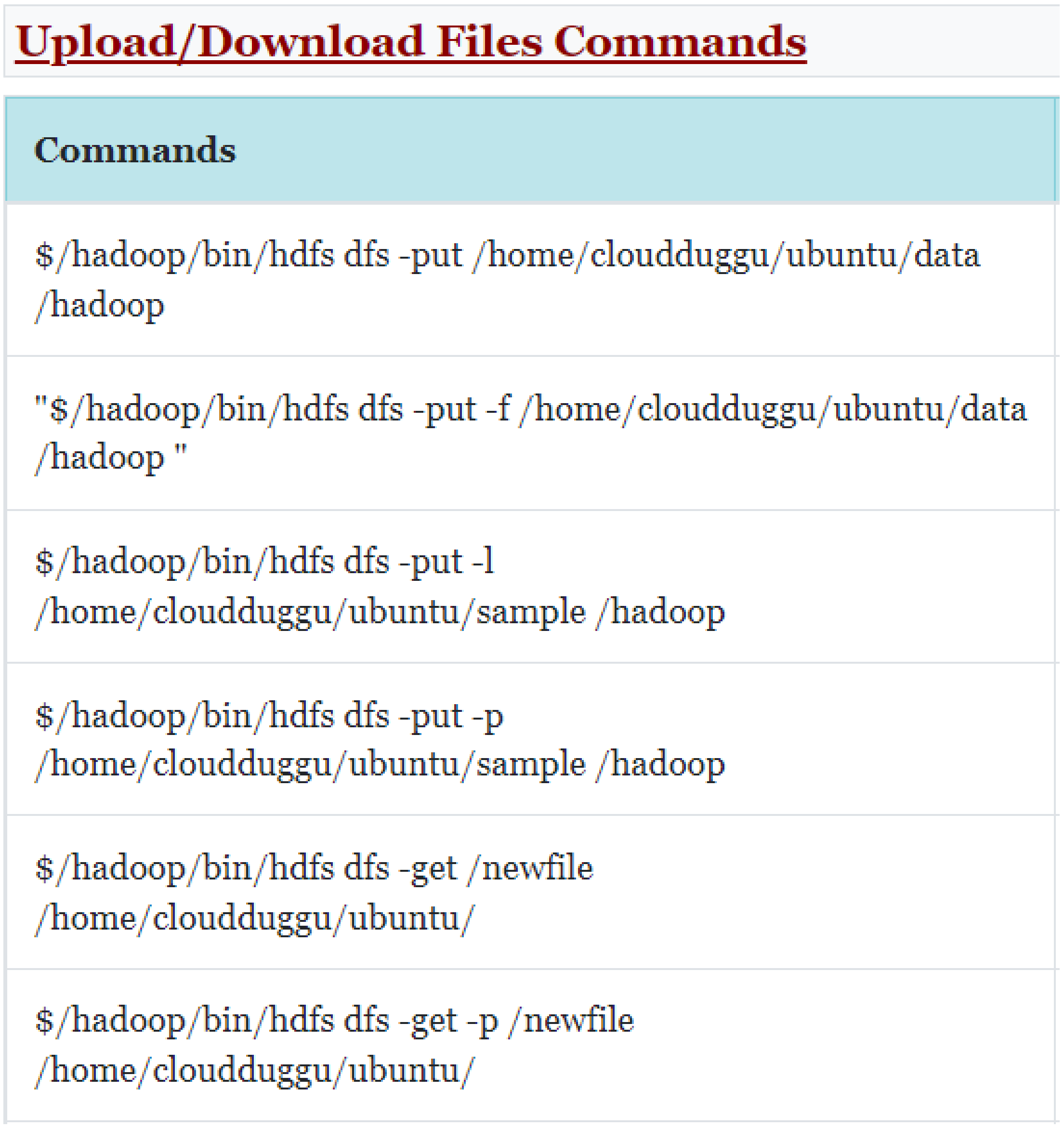
File Listing Commands

Commands
$/hadoop/bin/hdfs dfs -ls /
$/hadoop/bin/hdfs dfs -ls -d /hadoop
$/hadoop/bin/hdfs dfs -ls -h /data
$/hadoop/bin/hdfs dfs -ls -R /hadoop
$/hadoop/bin/hdfs dfs -ls /hadoop/word*



Read and Write Files Commands

Commands
$/hadoop/bin/hdfs dfs -text /hadoop/date.log
$/hadoop/bin/hdfs dfs -cat /hadoop/data
$/hadoop/bin/hdfs dfs -appendToFile /home/cloudduggu/ubuntu/data1 /hadoop/data2



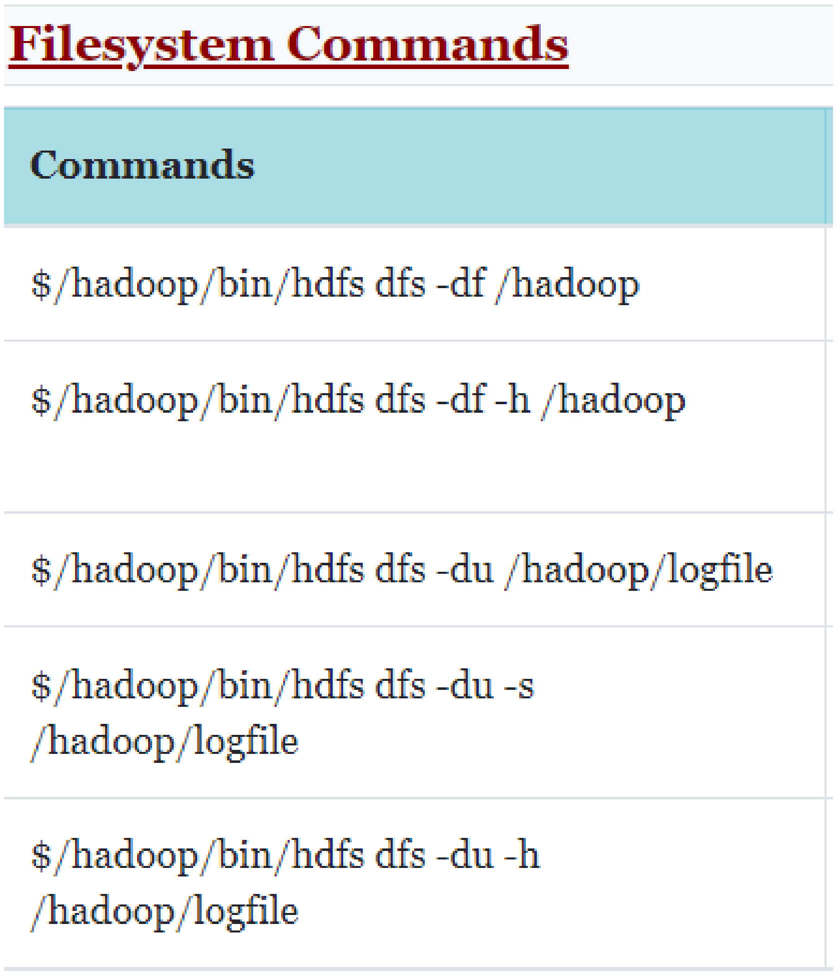
Upload/Download Files Commands

Commands
$/hadoop/bin/hdfs dfs -put /home/cloudduggu/ubuntu/data /hadoop
"$/hadoop/bin/hdfs dfs -put -f /home/cloudduggu/ubuntu/data /hadoop "
$/hadoop/bin/hdfs dfs -put -l /home/cloudduggu/ubuntu/sample /hadoop
$/hadoop/bin/hdfs dfs -put -p /home/cloudduggu/ubuntu/sample /hadoop
$/hadoop/bin/hdfs dfs -get /newfile /home/cloudduggu/ubuntu/
$/hadoop/bin/hdfs dfs -get -p /newfile /home/cloudduggu/ubuntu/

$/hadoop/bin/hdfs dfs -get /hadoop/*.log /home/cloudduggu/ubuntu/
$/hadoop/bin/hdfs dfs -copyFromLocal /home/cloudduggu/ubuntu/sample /hadoop
$/hadoop/bin/hdfs dfs -copyToLocal /newfile /home/cloudduggu/ubuntu/
$/hadoop/bin/hdfs dfs -moveFromLocal /home/cloudduggu/ubuntu/sample /hadoop

Ownership and Validation Commands

Commands
$/hadoop/bin/hdfs dfs -checksum /hadoop/logfile
$/hadoop/bin/hdfs dfs -chmod 755 /hadoop/logfile
$/hadoop/bin/hdfs dfs -chmod -r 755 /hadoop
$/hadoop/bin/hdfs dfs -chown ubuntu:ubuntu /hadoop
$/hadoop/bin/hdfs dfs -chown -r ubuntu:ubuntu /hadoop
$/hadoop/bin/hdfs dfs -chgrp ubuntu /hadoop
$/hadoop/bin/hdfs dfs -chgrp -r ubuntu /hadoop



Filesystem Commands

Commands
$/hadoop/bin/hdfs dfs -df /hadoop
$/hadoop/bin/hdfs dfs -df -h /hadoop
$/hadoop/bin/hdfs dfs -du /hadoop/logfile
$/hadoop/bin/hdfs dfs -du -s /hadoop/logfile
$/hadoop/bin/hdfs dfs -du -h /hadoop/logfile

PROGRAM 4

Write and execute Map-reduce word-count program in a single node

Step1. Type mapper.py and reducer.py using any editor

```
$sudo nano mapper.py
```

mapper.py

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

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

```
$sudo nano reducer.py
```

```
#!/usr/bin/env python
"""reducer.py"""
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None
```

```
# input comes from STDIN
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()
# parse the input we got from mapper.py
word, count = line.split('\t', 1)
# convert count (currently a string) to int
try:
        count = int(count)
except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
        current_count += count
else:
        if current_word:
                # write result to STDOUT
                print('%s\t%s' % (current_word, current_count))
                current_count = count
                current_word = wor

# do not forget to output the last word if needed!
if current_word == word:
        print('%s\t%s' % (current_word, current_count))
```

Step 2. Create input.txt file

```
$sudo nano input.txt
```

Step 3. Run the map-reduce word count program on UNIX platform

```
$cat input.txt | python3 mapper.py | sort | python3 reducer.py
```

Step 4. Run the map-reduce word count program on Hadoop

1. $sudo su hduser
2. $ start-dfs.sh
3. $start-yarn.sh
4. $jps
5. $hdfs dfs -put /input.txt / and

   $hdfs dfs -put /*.py(Path of mapper and reducer) /
6. hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar -files ./mapper.py,./reducer.py -mapper "python2 mapper.py" -reducer "python2 reducer.py" -input /5000-8.txt -output /0000009
7. Observe the result in localhost:port_number
8. Download outputfile and view it for wordcount

PROGRAM 5

Write and execute a Java/ python program to calculate the average salary of the employees in a company. (should refer to any pre-existing file or may generate their own).

Mapper:

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

# Input format: employee_id, name, salary

for line in sys.stdin:
    # Split the input line into fields
    fields = line.strip().split(",")

    # Check that the input line has the correct number of fields
    if len(fields) != 3:
        continue

    # Extract the salary from the input line
    salary = int(fields[2])

    # Emit the salary as the output key and a count of 1 as the output value
    print("{0}\t{1}".format(salary, 1))
```

Reducer:

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

import sys

# Initialize variables to hold the sum of salaries and the count of employees
total_salary = 0
employee_count = 0

# Process input from mapper
for line in sys.stdin:
    # Split the input line into key and value
    salary, count = line.strip().split("\t")

    # Update the sum of salaries and the count of employees
    total_salary += int(salary) * int(count)
    employee_count += int(count)

# Calculate the average salary
average_salary = total_salary / employee_count

# Output the result
print("Average salary: {0}".format(average_salary))
```

Input File:

```
1,John Doe,50000
2,Jane Smith,60000
3,Bob Johnson,70000
4,Susan Williams,55000
5,Tom Davis,65000
6,Emily Brown,75000
7,Michael Lee,45000
8,Kelly Green,55000
9,David Kim,80000
10,Michelle Wong,90000
```

Command to run:

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar \
  -input <input_path> \
  -output <output_path> \
  -mapper /path/to/mapper.py \
  -reducer /path/to/reducer.py
```

Output

```
Average salary: 64500.0
```

PROGRAM 6

Write and execute a Map-reduce Java/Python program for printing the maximum salary for a given input file.

Mapper:

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

import sys

for line in sys.stdin:
    # Remove leading and trailing whitespace
    line = line.strip()

    # Split the line into columns
    columns = line.split('\t')

    # Extract the salary column
    salary = columns[2]

    # Output the salary with a null key
    print("%s\t%s" % (None, salary))
```

Reducer:

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

import sys

# Initialize the maximum salary to 0
max_salary = 0

# Iterate over each line in the input
for line in sys.stdin:
    # Split the line into key and value

    key, value = line.strip().split('\t', 1)
```

```
        # Convert the value to an integer
        try:
            salary = int(value)
        except ValueError:
            continue

        # Update the maximum salary if this salary is higher
        if salary > max_salary:
            max_salary = salary

    # Output the final maximum salary
    print("Max Salary: %s" % max_salary)
```

Input file : employees.txt

```
Alice       Engineering         75000
Bob         Sales               60000
Charlie     Engineering         85000
David       Sales               45000
Eve         Marketing           90000
Frank       Engineering         100000
```

Command to run

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar \
    -file mapper.py \
    -file reducer.py \
    -mapper mapper.py \
    -reducer reducer.py \
    -input employees.txt \
    -output output
```

Output

```
Max Salary: 100000
```

PROGRAM 7

Write and execute a Map-reduce Java/python program to print year wise sales of a company from a given CSV file.

Mapper:

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

import sys
import csv

for line in csv.reader(iter(sys.stdin.readline, '')):
    year = line[0]
    sales = line[2]
    print(f"{year}\t{sales}")
```

Reducer:

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

current_year = None
total_sales = 0

for line in sys.stdin:
    year, sales = line.strip().split('\t')

    if current_year is None:
        current_year = year

    if year == current_year:
        total_sales += int(sales)
    else:
        print(f"{current_year}\t{total_sales}")
        current_year = year

        total_sales = int(sales)

if current_year is not None:
    print(f"{current_year}\t{total_sales}")
```

Input File: sales.csv

```
2018,Jan,10000
2018,Feb,12000
2018,Mar,15000
2019,Jan,20000
2019,Feb,22000
2019,Mar,25000
2020,Jan,30000
2020,Feb,32000
2020,Mar,35000
```

Command to Run:

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar \
    -file mapper.py \
    -file reducer.py \
    -mapper mapper.py \
    -reducer reducer.py \
    -input sales.csv \
    -output output
```

Output

```
2018 37000
2019 67000
2020 97000
```

PROGRAM 8

Execute a python program to implement inverted index

Mapper.py

```
import sys def mapper():
for line in sys.stdin:
        line = line.strip()
        document, words = line.split(" ", 1)
        word_list = words.split()
        for word in word_list:
                print(f"{word}\t{document}")
if  name== " main ": mapper()
```

Reducer.py

```
import sys def

reducer():
    current_word = None doc_list = []

    for line in sys.stdin: line =
        line.strip()
        word, document = line.split("\t", 1)

        if current_word == word: doc_list.append(document)
        else:
            if current_word:
                print(f"{current_word}\t{', '.join(set(doc_list))}") doc_list
            = [document]
            current_word = word

    if current_word:
        print(f"{current_word}\t{', '.join(set(doc_list))}")

if _name___== "_main_": reducer()
```

Input.txt

```
doc1 apple banana doc2
banana orange doc3 apple
mango doc4 banana apple
```

Output

```
apple: doc1, doc3, doc4 banana:
doc1, doc2, doc4 orange: doc2
mango: doc3
```

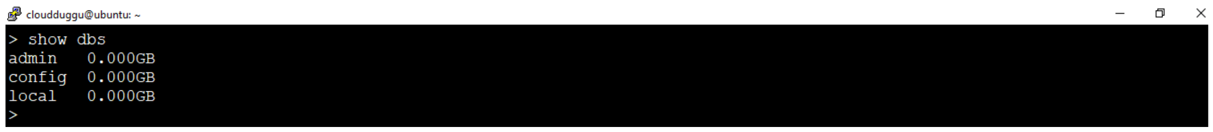
PROGRAM 9

Installation of MongoDB Compass, Implement NoSQL Database Operations: CRUD operations, Arrays using MongoDB

MongoDB Commands

Create Database

Step 1: Open MongoDB compass, connect to the localhost of the device.
Step 2: Open terminal and enter mongod and mongo cpmmands.
Step 3: show dbs (Use the **show dbs** command to check the list of databases that are already present in MongoDB.)

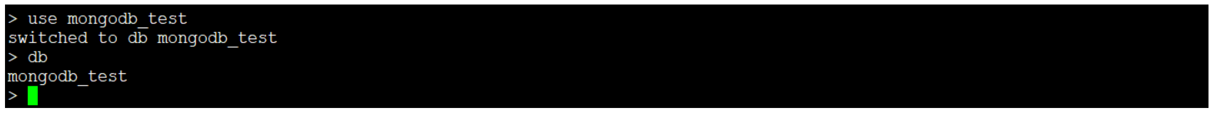


cloudduggu@ubuntu: ~

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
>
```

Step 4: The **use database name** command is used to **create a database in MongoDB**. If the database is already created then it will switch to that database otherwise it will create a new database.

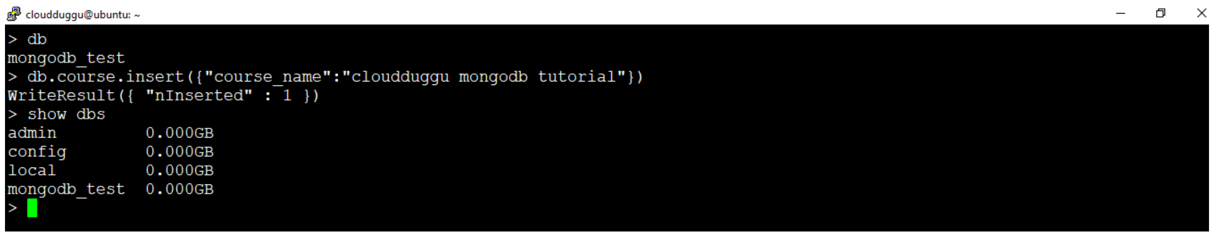
use mongodb_test



```
> use mongodb_test
switched to db mongodb_test
> db
mongodb_test
>
```

Step 5: If we run the **show dbs** command then newly database will not appear because there is no document present in that. So to display the database name we have to insert at least one document in the database using the below command.

> db.course.insert({"course_name":"cloudduggu mongodb tutorial"})
> show dbs



cloudduggu@ubuntu: ~

```
> db
mongodb_test
> db.course.insert({"course_name":"cloudduggu mongodb tutorial"})
WriteResult({ "nInserted" : 1 })
> show dbs
admin         0.000GB
config        0.000GB
local         0.000GB
mongodb_test  0.000GB
>
```

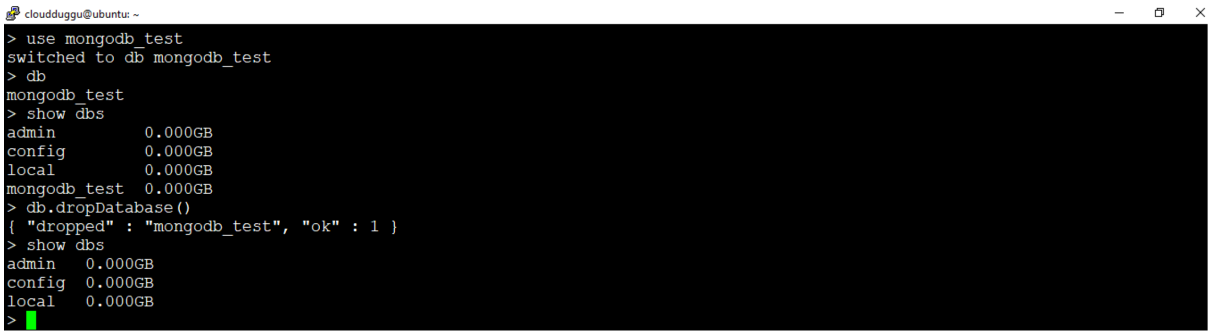
Step 6: drop Database() method

The **dropDatabase()** command is used to drop a selected database.

>db.dropdatabase()

We will use the **use mongodb_test** command to select the newest created database and then we will run the **db.dropDatabase()** command to drop the database.

> use mongodb_test
> show dbs
> db.dropdatabase()



cloudduggu@ubuntu: ~

```
> use mongodb_test
switched to db mongodb_test
> db
mongodb_test
> show dbs
admin         0.000GB
config        0.000GB
local         0.000GB
mongodb_test  0.000GB
> db.dropDatabase()
{ "dropped" : "mongodb_test", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

➢ **MongoDB insert Methods:**

In MongoDB, there is the following **Insert() Method** which is used to insert the documents in the collection.

- **db.collection.insertOne()**
- **db.collection.insertMany()**
- **db.collection.insert()**

1)db.collection.insertOne()

➢ The **insertOne()** method is used to insert a single document in the collection. In case the collection does not exist then MongoDB Insert methods will create one.

```
 > db.item_inventory.insertOne(
          { item_name: "canvas", item_quantity: 115, item_tags: ["cotton"], item_size: { height: 29, weight: 36.5, uom: "cm" } }
       )
```

2) db.collection.insertMany()

➢ The **insertMany()** method is used to insert many Documents in the Collection.

```
> db.item_inventory.insertMany(
[  { item_name: "journal",  item_quantity: 25,  item_tags: ["blank", "red"], item_size: { height: 14, weight : 21, uom: "cm" } },
      { item_name: "mat", item_quantity: 85, item_tags: ["gray"], item_size: { height: 27.9, weight: 35.5, uo
   m: "cm" } },

  { item_name: "laptop", item_quantity: 100, item_tags: ["black"], item_size: { height: 10, weight: 2, uom: "
cm" } },
  { item_name: "mouse", item_quantity: 110, item_tags: ["white"], item_size: { height: 3, weight: 4, uom: "c
m" } },
  { item_name: "mousepad", item_quantity: 25, item_tags: ["gel", "blue"], item_size: { height: 19, weight: 2
2.85, uom: "cm" } }
] )
```

3) db.collection.insert()

- The **insert()** method is used to insert a Document or multiple Documents in the Collection.

```
> db.products_detail.insert( { item_name: "box", item_quantity: 100 } )
```

PROGRAM 10

Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB.

MongoDB Limit and Sort Records

- **Limit () Method**
- **Sort () Method**

> db.inventory_records.insertMany([
{ item_name: "canvas",item_qty: 110,item_size: { height: 29, weight: 36.5, uom: "cm" }, status : "A" },{ item_name: "journal",item_qty: 35, item_size: { height: 15, weight: 22, uom: "cm" }, s tatus: "A" },{ item_name: "mat",item_qty: 95, item_size: { height: 23.9, weight: 32.5, uom: "cm" }, status: "A" },{ item_name: "mousepad",item_qty: 45, item_size: { height: 16, weight: 28.85, uo m: "cm" }, status: "P" },{ item_name: "notebook",item_qty: 60, item_size: { height: 7.5, weight: 10, uom: "in" }, status: "P" },{ item_name: "paper",item_qty: 200, item_size: { height: 9.5, weig ht: 12, uom: "in" }, status: "D" },{ item_name: "planner",item_qty: 75, item_size: { height: 25.85 , weight: 40, uom: "cm" }, status: "D" },{ item_name: "postcard",item_qty: 25, item_size: { heig ht: 15, weight: 17.25, uom: "cm" }, status: "A" },{ item_name: "sketchbook",item_qty: 90, item_ size: { height: 13, weight: 23, uom: "cm" }, status: "A" },{ item_name: "sketch_pad",item_qty: 8 5, item_size: { height: 25.85, weight: 32.5, uom: "cm" }, status: "A" }]);

1) Limit() Method

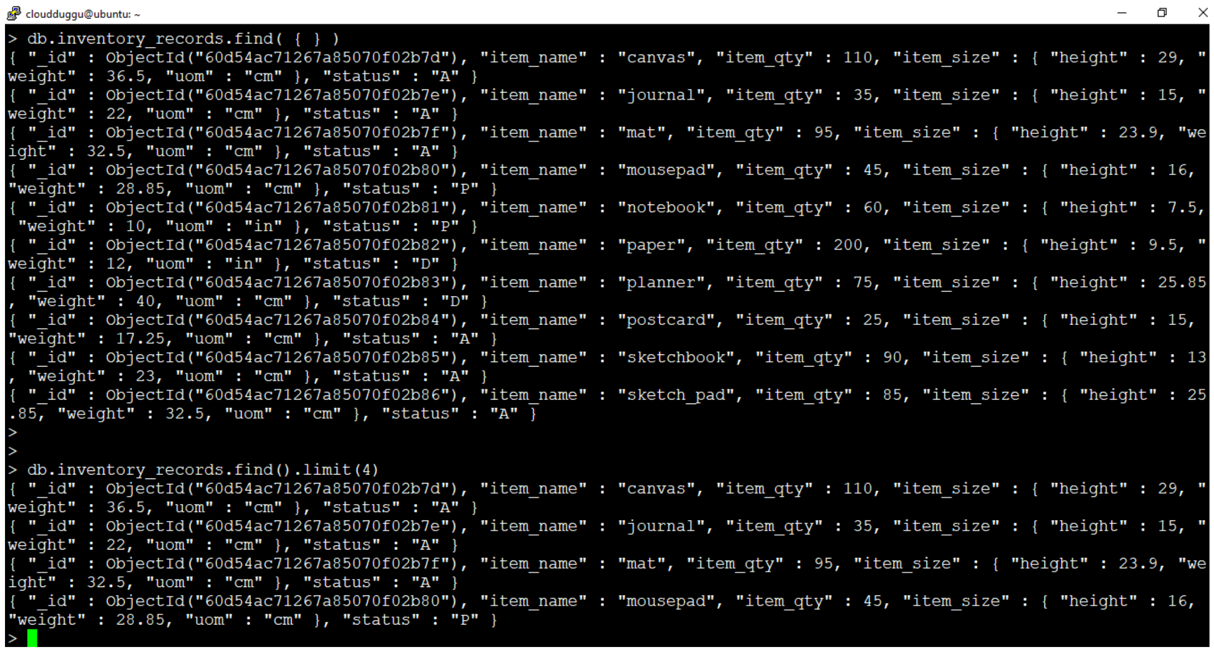
The MongoDB limit() method is used to limit the number of Documents that will be shown. It accepts a number that represents the number of Documents that will be projected.

Limit() Method
The MongoDB limit() method is used to limit the number of Documents that will be shown. It accepts a number that represents the number of Documents that will be projected.

Syntax:
The Following is the syntax of limit() method.
> db.collectionname.find().limit(NUMBER)

➢ **Example:**

> db.inventory_records.find({ })
> db.inventory_records.find().limit(4)



```
> db.inventory_records.find( { } )
{ "_id" : ObjectId("60d54ac71267a85070f02b7d"), "item_name" : "canvas", "item_qty" : 110, "item_size" : { "height" : 29, "
weight" : 36.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b7e"), "item_name" : "journal", "item_qty" : 35, "item_size" : { "height" : 15, "
weight" : 22, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b7f"), "item_name" : "mat", "item_qty" : 95, "item_size" : { "height" : 23.9, "we
ight" : 32.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b80"), "item_name" : "mousepad", "item_qty" : 45, "item_size" : { "height" : 16,
"weight" : 28.85, "uom" : "cm" }, "status" : "P" }
{ "_id" : ObjectId("60d54ac71267a85070f02b81"), "item_name" : "notebook", "item_qty" : 60, "item_size" : { "height" : 7.5,
 "weight" : 10, "uom" : "in" }, "status" : "P" }
{ "_id" : ObjectId("60d54ac71267a85070f02b82"), "item_name" : "paper", "item_qty" : 200, "item_size" : { "height" : 9.5, "
weight" : 12, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("60d54ac71267a85070f02b83"), "item_name" : "planner", "item_qty" : 75, "item_size" : { "height" : 25.85
, "weight" : 40, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("60d54ac71267a85070f02b84"), "item_name" : "postcard", "item_qty" : 25, "item_size" : { "height" : 15,
"weight" : 17.25, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b85"), "item_name" : "sketchbook", "item_qty" : 90, "item_size" : { "height" : 13
, "weight" : 23, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b86"), "item_name" : "sketch_pad", "item_qty" : 85, "item_size" : { "height" : 25
.85, "weight" : 32.5, "uom" : "cm" }, "status" : "A" }
>
>
> db.inventory_records.find().limit(4)
{ "_id" : ObjectId("60d54ac71267a85070f02b7d"), "item_name" : "canvas", "item_qty" : 110, "item_size" : { "height" : 29, "
weight" : 36.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b7e"), "item_name" : "journal", "item_qty" : 35, "item_size" : { "height" : 15, "
weight" : 22, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b7f"), "item_name" : "mat", "item_qty" : 95, "item_size" : { "height" : 23.9, "we
ight" : 32.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("60d54ac71267a85070f02b80"), "item_name" : "mousepad", "item_qty" : 45, "item_size" : { "height" : 16,
"weight" : 28.85, "uom" : "cm" }, "status" : "P" }
>
```

2) sort() Method

➢ The MongoDB sort() method is used to perform the sorting of the input Document and return their sorted order. The sort() method provides two types of sort order, the first one is 1 that sort the Document in ascending order, and the second one is -1 that sort the Document in descending order.
Syntax:

The Following is the syntax of sort() method.

> db.collectionname.find().sort{ fieldname_1: sort_order, fieldname_2: sort_order ... } }

➢ Example:
> db.inventory_records.find().sort({"item_qty": 1})

> db.inventory_records.find().sort({"item_qty": -1})

MongoDB Aggregation
MongoDB aggregation operations are used to process the data and return the processed records. The aggregation operations group the multiple Documents and perform various operations on that group of Documents to return a single value.

1)aggregate() Method

MongoDB uses the aggregate() method to perform the aggregation. Let us see the aggregate() method in detail with the below example.

Syntax:

The following is the syntax of aggregate() method.

```
> db.collectionname.aggregate(aggregate_operation)
```

```
> use mongodb_test
```

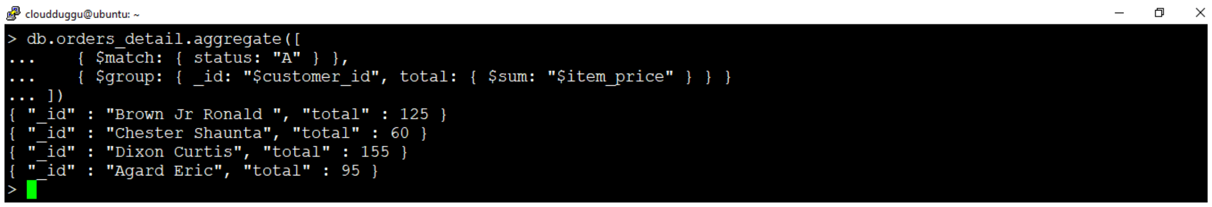
```
> db.orders_detail.insertMany([
   { _id: 101, customer_id: "Agard Eric", order_date: new Date("2019-03-
01"), item_price: 25,  items_detail: [ { sku: "oranges", quantity: 5, item_price: 2.5 }, { sku: "apples"
, quantity: 5, item_price: 2.5 } ], status: "A" },
   { _id: 102, customer_id: "Agard Eric", order_date: new Date("2019-03-
08"), item_price: 70,  items_detail: [ { sku: "oranges", quantity: 8, item_price: 2.5 }, { sku: "chocol
ates", quantity: 5, item_price: 10 } ], status: "A" },
   { _id: 103, customer_id: "Brown Jr Ronald ", order_date: new Date("2019-03-
08"), item_price: 50,  items_detail: [ { sku: "oranges", quantity: 10, item_price: 2.5 }, { sku: "pears"
, quantity: 10, item_price: 2.5 } ], status: "A" },
   { _id: 104, customer_id: "Brown Jr Ronald ", order_date: new Date("2019-03-
18"), item_price: 25,  items_detail: [ { sku: "oranges", quantity: 10, item_price: 2.5 } ], status: "A" }
,
   { _id: 105, customer_id: "Brown Jr Ronald ", order_date: new Date("2018-03-
19"), item_price: 50,  items_detail: [ { sku: "chocolates", quantity: 5, item_price: 10 } ], status: "A"
},
   { _id: 106, customer_id: "Chester Shaunta", order_date: new Date("2018-03-
19"), item_price: 35,  items_detail: [ { sku: "carrots", quantity: 10, item_price: 1.0 }, { sku: "apples"
, quantity: 10, item_price: 2.5 } ], status: "A" },
   { _id: 107, customer_id: "Chester Shaunta", order_date: new Date("2018-03-
20"), item_price: 25,  items_detail: [ { sku: "oranges", quantity: 10, item_price: 2.5 } ], status: "A" }
,
   { _id: 108, customer_id: "Dixon Curtis", order_date: new Date("2017-03-
20"), item_price: 75,  items_detail: [ { sku: "chocolates", quantity: 5, item_price: 10 }, { sku: "apple
s", quantity: 10, item_price: 2.5 } ], status: "A" },
   { _id: 109, customer_id: "Dixon Curtis", order_date: new Date("2017-03-
20"), item_price: 55,  items_detail: [ { sku: "carrots", quantity: 5, item_price: 1.0 }, { sku: "apples",
quantity: 10, item_price: 2.5 }, { sku: "oranges", quantity: 10, item_price: 2.5 } ], status: "A" },
   { _id: 110, customer_id: "Dixon Curtis", order_date: new Date("2017-03-
23"), item_price: 25,  items_detail: [ { sku: "oranges", quantity: 10, item_price: 2.5 } ], status: "A" }
])
```

2) $sum expression with aggregate() Method:

In the below example, in the first stage, we will use the $match expression to filter the Document of "orders_detail" Collection who has the status= "A", in the second stage we will use $group expression to group the Document by customer_id and calculate the sum of each customer_id.

Example:

```
> db.orders_detail.aggregate([
      { $match: { status: "A" } }, { $group: { _id: "$customer_id", total: { $sum: "$item_price" } } }
  ])
```



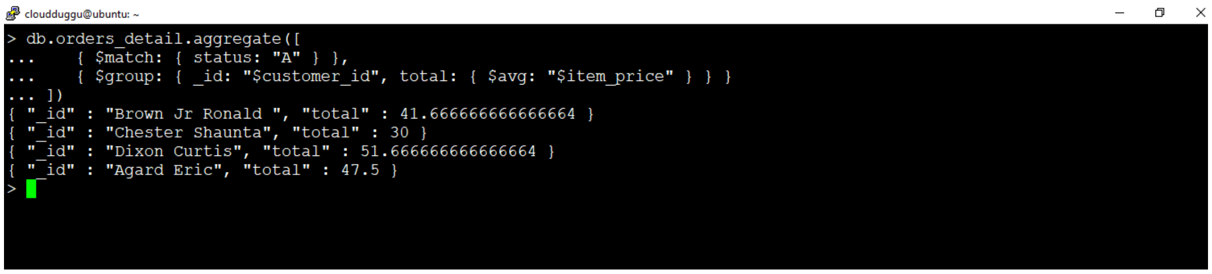
```
cloudduggu@ubuntu: ~
> db.orders_detail.aggregate([
...     { $match: { status: "A" } },
...     { $group: { _id: "$customer_id", total: { $sum: "$item_price" } } }
... ])
{ "_id" : "Brown Jr Ronald ", "total" : 125 }
{ "_id" : "Chester Shaunta", "total" : 60 }
{ "_id" : "Dixon Curtis", "total" : 155 }
{ "_id" : "Agard Eric", "total" : 95 }
>
```

3) $avg expression with aggregate() Method

The below example will fetch the average value of all matching Documents where status= "A".

Example:

```
> db.orders_detail.aggregate([
      { $match: { status: "A" } }, { $group: { _id: "$customer_id", total: { $avg: "$item_price" } } }
  ])
```



```
cloudduggu@ubuntu: ~
> db.orders_detail.aggregate([
...     { $match: { status: "A" } },
...     { $group: { _id: "$customer_id", total: { $avg: "$item_price" } } }
... ])
{ "_id" : "Brown Jr Ronald ", "total" : 41.666666666666664 }
{ "_id" : "Chester Shaunta", "total" : 30 }
{ "_id" : "Dixon Curtis", "total" : 51.666666666666664 }
{ "_id" : "Agard Eric", "total" : 47.5 }
>
```

4) $min expression with aggregate() Method

The below example will fetch the minimum value of all matching Documents where status= "A".

Example:

```
> db.orders_detail.aggregate([
      { $match: { status: "A" } }, { $group: { _id: "$customer_id", total: { $min: "$item_price" } } }
  ])
```

5) $max expression with aggregate() Method

The below example will fetch the maximum value of all matching Documents where status= "A".

Example:

```
> db.orders_detail.aggregate([
    { $match: { status: "A" } }, { $group: { _id: "$customer_id", total: { $max: "$item_price" } } }
  ])
```

6) $first expression with aggregate() Method

The $first expression returns the first Document from the group of Documents.

Example:

```
> db.orders_detail.aggregate([
    { $group: { _id: "$customer_id",  total: { $first: "$item_price" } } }
  ])
```

7) $last expression with aggregate() Method

The $last expression returns the last Document from the group of Documents.

Example:

```
> db.orders_detail.aggregate([
    { $group: { _id: "$customer_id",  total: { $last: "$item_price" } } }
  ])
```

MongoDB Indexing

1) db.collection_name.createIndex()
MongoDB createIndex() method is used to create an index on the Collection of Documents. MongoDB creates a default index on the _id field of Collection and provides the facility to create additional Indexes using the createIndex() method to support the queries and operation.

Syntax:

The following is the syntax of the createIndex() method. The 1 represents the ascending order and the -1 represents the descending order.

>db.collection_name.createIndex({keyname:1 or -1})

2) db.collection_name.getIndexes()
The MongoDB getIndexes() method is used to show the description of all Indexes of a Collection.

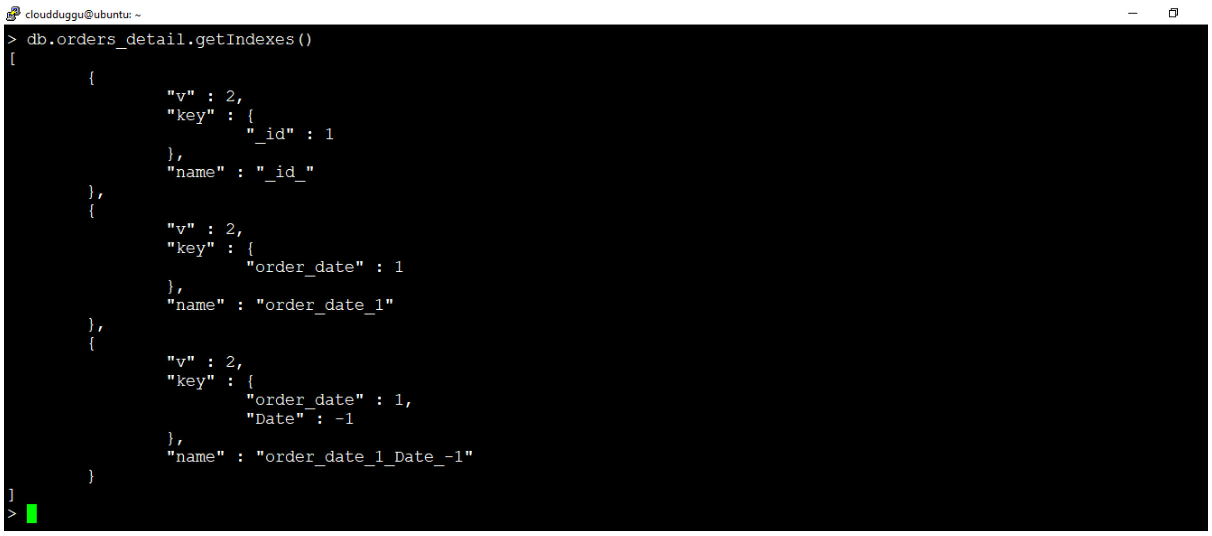
Syntax:

The following is the syntax of getIndexes() method.

>db.collectionname.getIndexes()

In the below example we can the Index detail of the Collection "orders_detail".

➢ Example:
> db.orders_detail.getIndexes()



```
> db.orders_detail.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "order_date" : 1
                },
                "name" : "order_date_1"
        },
        {
                "v" : 2,
                "key" : {
                        "order_date" : 1,
                        "Date" : -1
                },
                "name" : "order_date_1_Date_-1"
        }
]
>
```

3)
db.collection_name.dropIndex()
➢ MongoDB dropIndex() method is used to drop a specific index of a Document.
➢ Syntax:
The following is the syntax of dropIndex() method.

>db.collectionname.dropIndex({keyname: key_value})

Let us see the below example to drop an index that is created on Document "order_date" of "orders_detail" Collection.

Example:

> db.orders_detail.dropIndex({"order_date": 1})

4) db.collection_name.dropIndexes()
MongoDB dropIndexes() method is used to drop multiple indexes.

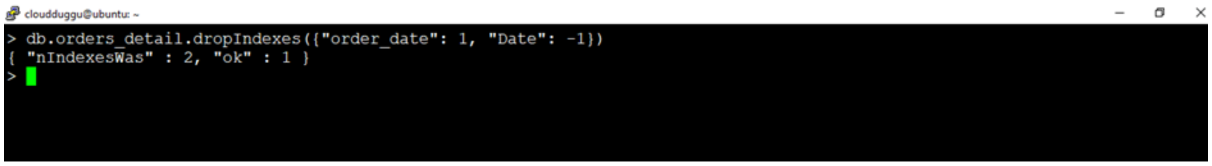
Syntax:

The following is the syntax of dropIndexes() method.

>db.collectionname.dropIndexes({keyname1: key_value1, keyname2: key_value2})

We will see an example to drop multiple indexes "order_date" =1, "Date" =-1 that is created above for "orders_detail" Collection.

➢ Example:
> db.orders_detail.dropIndexes({"order_date": 1, "Date": -1})



```
> db.orders_detail.dropIndexes({"order_date": 1, "Date": -1})
{ "nIndexesWas" : 2, "ok" : 1 }
>
```

PROGRAM 11

Install Hive, HBase run basic SQL commands.

Hive Installation

Step 1. Start all the Hadoop daemons

```
$start-all.sh
```

Step 2. Download the hive file by using the wget command

Step 3. Extract the jar file

```
$sudo tar -xzf apache-hive-3.1.2-bin.tar.gz
```

Step 4. Create a Hive Directory and change the directory

```
$sudo mkdir -p /usr/local/hive
$cd apache-hive-3.1.2-bin/
```

Step 5. Move the content of the downloaded file to the Hive directory

```
$sudo mv * /usr/local/hive/
```

Step 6. Update the bashrc file by setting up all the paths and refresh the bashrc

```
$sudo nano ~/.bashrc
        #HIVE VARIABLES START
        export HIVE_PATH=/usr/local/hive
        export PATH=$PATH:$HIVE_PATH/bin
        #HIVE VARIABLE END
$source ~/.bashrc
```

Step 7. Update configuration files by using the following steps

```
$cd /usr/local/hive/conf
$sudo cp hive-default.xml.template hive-site.xml
```

```
$sudo nano hive-site.xml
<property>
            <name>hive.metastore.warehouse.dir</name>
            <value>/user/hive/warehouse</value>
            <description>location of default database for the warehouse</description>
  </property>
  <property>
            <name>hive.metastore.warehouse.external.dir</name>
            <value/>
            <description>Default location for external tables created in the warehouse. If not set or
                                    null, then the normal warehouse location will be used as the default
                                     location.
            </description>
  </property>
  <property>
            <name>hive.metastore.uris</name>
            <value/>
            <description>Thrift URI for the remote metastore. Used by metastore client to connect to
                                     remote metastore.</description>
            </property>
$sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
            <property>
                        <name>hadoop.proxyuser.hduser.groups</name>
                        <value>*</value>
            </property>
            <property>
                        <name>hadoop.proxyuser.hduser.hosts</name>
                        <value>*</value>
            </property>
```

```
<property>
        <name>hadoop.proxyuser.server.hosts</name>
        <value>*</value>
</property>
<property>
        <name>hadoop.proxyuser.server.groups</name>
        <value>*</value>
</property>
```

```
$sudo nano $HIVE_PATH/bin/hive-config.sh
        export HADOOP_HOME=/usr/local/hadoop
```

Step 8. Create the following directories and change the mode

```
hdfs dfs -mkdir -p /user
hdfs dfs -mkdir -p /user/hive
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -mkdir /tmp
hdfs dfs -chmod g+w /user/hive/warehouse
hdfs dfs -chmod g+w /tmp
schematool -dbType derby -initSchema
```

Step 9. : **To launch hive shell and To launch hive server**

```
$hive
$hiveserver2
```

HIVE Commands

Step 1) Go to home/hadoop and open the terminal from this directory.

Step 2) Start hadoop daemons.

Step 3) After all daemons up, type hive and hit enter from home/hadoop.

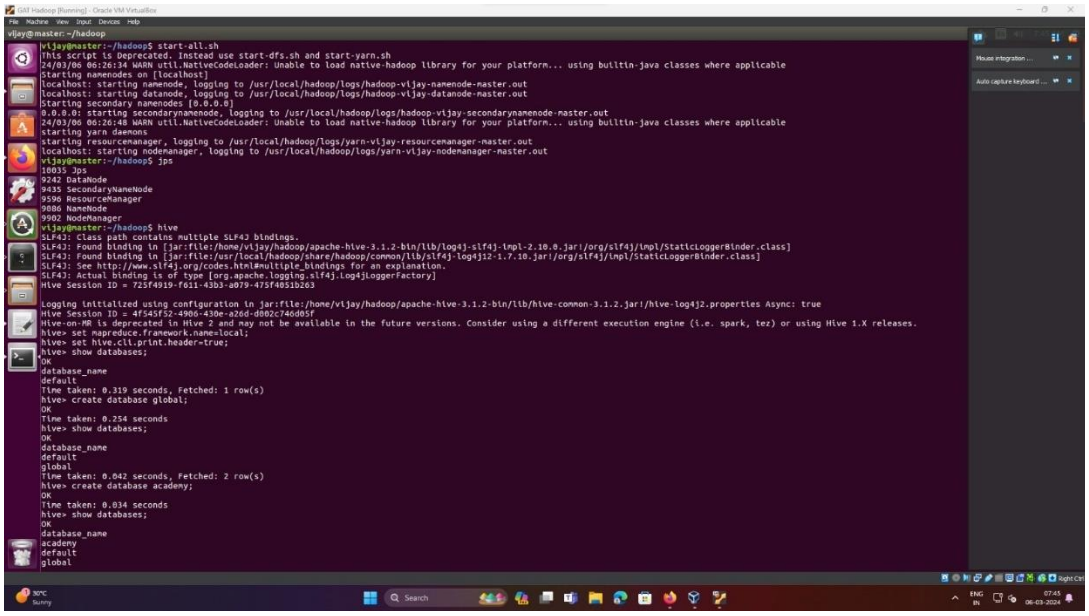
Step 4) Set mapreduce.framework.name=local;

Step 5) Set hive.cli.print.header=true // execute this within hive.//

Step 6) show databases;

Step 7) use database <databasename>

Step 8) show tables;



```
vijay@master:~/hadoop$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
24/03/06 06:26:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-vijay-namenode-master.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-vijay-datanode-master.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-vijay-secondarynamenode-master.out
24/03/06 06:26:48 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-vijay-resourcemanager-master.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-vijay-nodemanager-master.out
vijay@master:~/hadoop$ jps
10035 Jps
9242 DataNode
9435 SecondaryNameNode
9596 ResourceManager
9086 NameNode
9902 NodeManager
vijay@master:~/hadoop$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/vijay/hadoop/apache-hive-3.1.2-bin/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 725f4919-f611-43b3-a079-475f4051b263

Logging initialized using configuration in jar:file:/home/vijay/hadoop/apache-hive-3.1.2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = 4f545f52-4906-430e-a26d-d002c746d05f
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive> set mapreduce.framework.name=local;
hive> set hive.cli.print.header=true;
hive> show databases;
OK
database_name
default
Time taken: 0.319 seconds, Fetched: 1 row(s)
hive> create database global;
OK
Time taken: 0.254 seconds
hive> show databases;
OK
database_name
default
global
Time taken: 0.042 seconds, Fetched: 2 row(s)
hive> create database academy;
OK
Time taken: 0.034 seconds
hive> show databases;
OK
database_name
academy
default
global
```

create table order_products (order_id int,product_id int, item_id int, quantity int, list_price decimal(10,2), discount decimal (10,2)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

create table orders (order_id int,customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

create table orders (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

create table orders (order_id int, customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (year int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

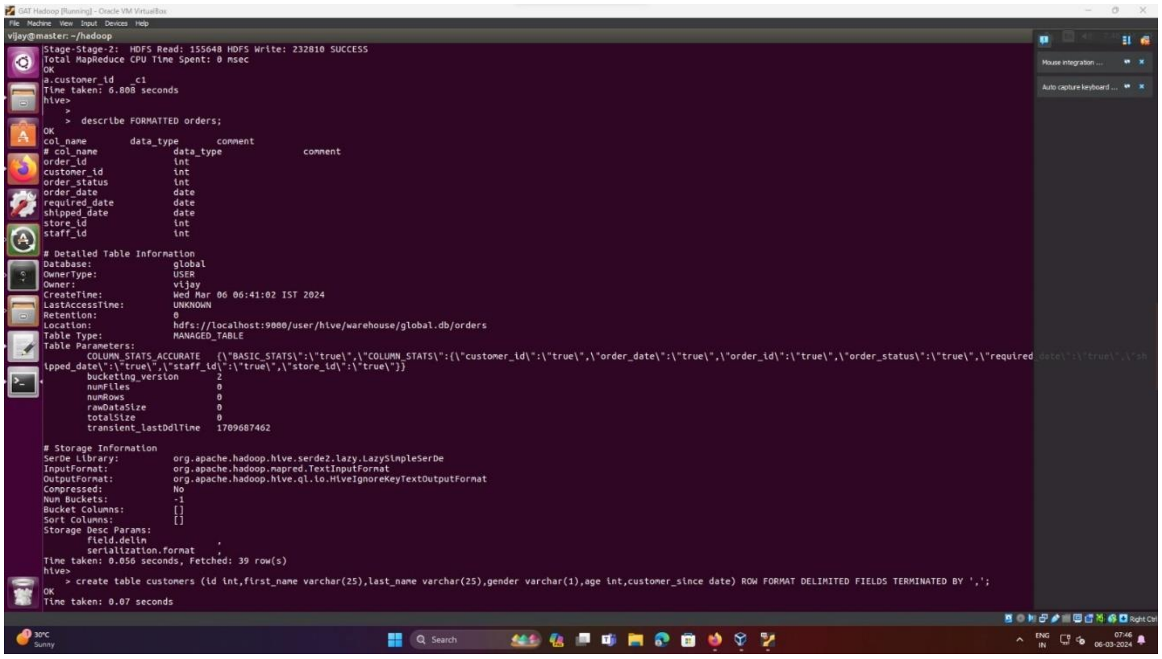
create table orders (order_id int, customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (year int) CLUSTERED BY (order_id) INTO 4 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

hive> load data local inpath '/home/vijay/hadoop/order_products.csv' into table order_products;

hive> load data local inpath '/home/vijay/hadoop/orders.csv' into table orders;

hive> select a.customer_id, sum(b.list_price * b.quantity) from orders a, order_products b where a.order_id = b.order_id GROUP BY a.customer_id;

hive> describe FORMATTED orders;



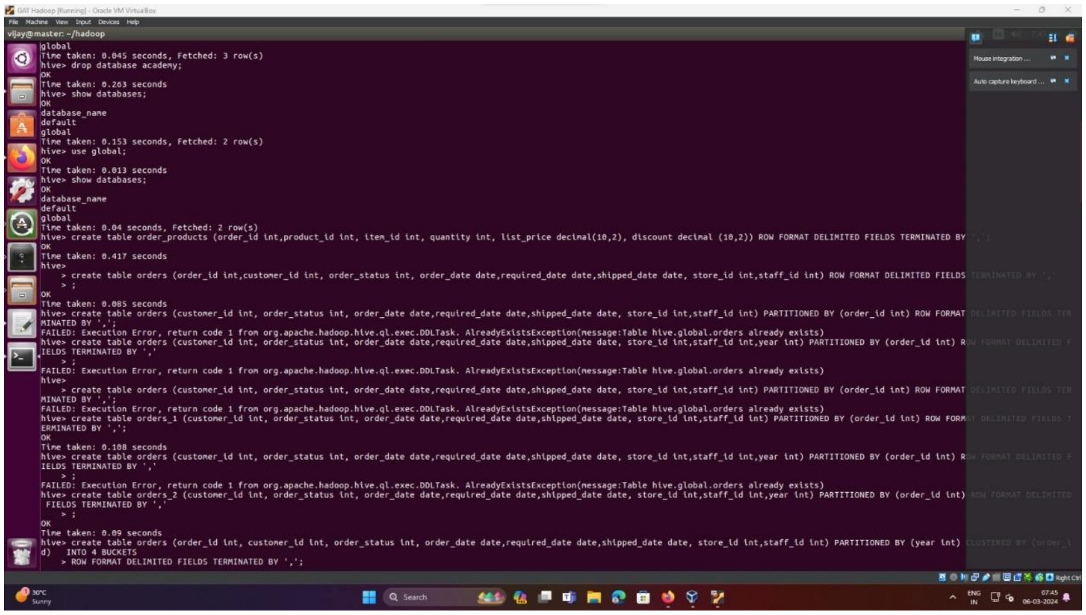
```
Stage-Stage-2:  HDFS Read: 155648 HDFS Write: 232810 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
a.customer_id   _c1
Time taken: 6.808 seconds
hive>
    >
    >  describe FORMATTED orders;
OK
col_name        data_type       comment
# col_name              data_type               comment
order_id                int
customer_id             int
order_status            int
order_date              date
required_date           date
shipped_date            date
store_id                int
staff_id                int

# Detailed Table Information
Database:               global
OwnerType:              USER
Owner:                  vijay
CreateTime:             Wed Mar 06 06:41:02 IST 2024
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://localhost:9000/user/hive/warehouse/global.db/orders
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":{\"customer_id\":\"true\",\"order_date\":\"true\",\"order_id\":\"true\",\"order_status\":\"true\",\"required_date\":\"true\",\"sh
ipped_date\":\"true\",\"staff_id\":\"true\",\"store_id\":\"true\"}}
        bucketing_version       2
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1709687462

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.056 seconds, Fetched: 39 row(s)
hive>
    > create table customers (id int,first_name varchar(25),last_name varchar(25),gender varchar(1),age int,customer_since date) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 0.07 seconds
```

create table customers (id int,first_name varchar(25),last_name varchar(25),gender varchar(1),age int,customer_since date) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

hive> load data local inpath '/home/vijay/hadoop/customers.csv' into table customers;

create table sales (id int,sales_date date,book_id int,cust_id int,quantity int,amount float) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

hive> load data local inpath '/home/vijay/hadoop/sales.csv' into table sales;

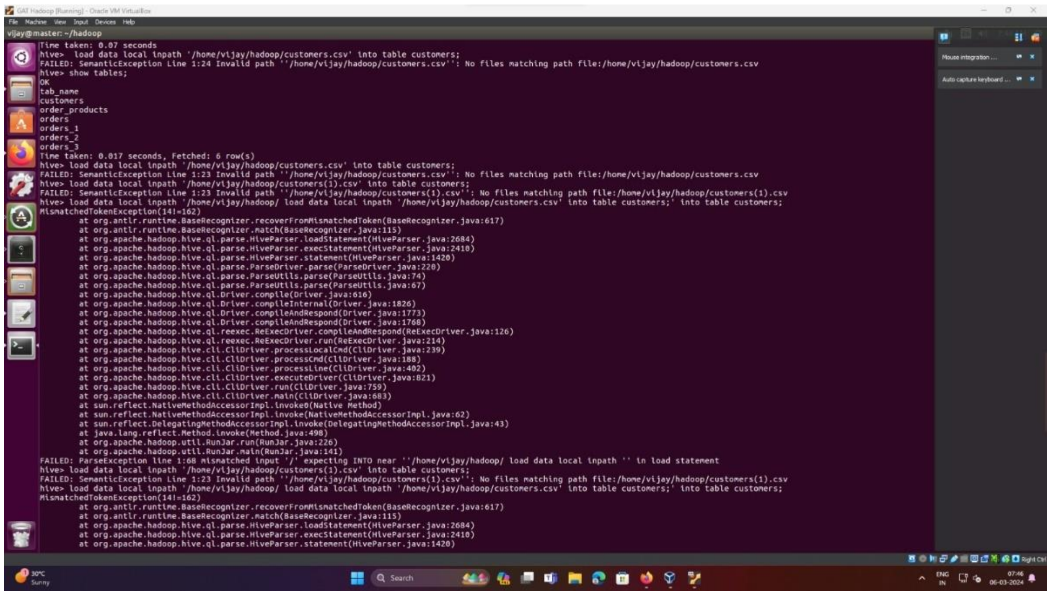


```
global
Time taken: 0.045 seconds, Fetched: 3 row(s)
hive> drop database academy;
OK
Time taken: 0.263 seconds
hive> show databases;
OK
database_name
default
global
Time taken: 0.153 seconds, Fetched: 2 row(s)
hive> use global;
OK
Time taken: 0.013 seconds
hive> show databases;
OK
database_name
default
global
Time taken: 0.04 seconds, Fetched: 2 row(s)
hive> create table order_products (order_id int,product_id int, item_id int, quantity int, list_price decimal(10,2), discount decimal (10,2)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 0.417 seconds
hive>
    > create table orders (order_id int,customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > ;
OK
Time taken: 0.085 seconds
hive> create table orders (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED FIELDS TER
MINATED BY ',';
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table hive.global.orders already exists)
hive> create table orders (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int,year int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED F
IELDS TERMINATED BY ','
    > ;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table hive.global.orders already exists)
hive>
    > create table orders (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED FIELDS TER
MINATED BY ',';
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table hive.global.orders already exists)
hive> create table orders_1 (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED FIELDS T
ERMINATED BY ',';
OK
Time taken: 0.108 seconds
hive> create table orders (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int,year int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED F
IELDS TERMINATED BY ','
    > ;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table hive.global.orders already exists)
hive> create table orders_2 (customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int,year int) PARTITIONED BY (order_id int) ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
    > ;
OK
Time taken: 0.09 seconds
hive> create table orders (order_id int, customer_id int, order_status int, order_date date,required_date date,shipped_date date, store_id int,staff_id int) PARTITIONED BY (year int) CLUSTERED BY (order_i
d)   INTO 4 BUCKETS
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

create table books (id int,name VARCHAR(50),author VARCHAR (50),genre VARCHAR(10),quantity int,price float) ROW

FORMAT DELIMITED FIELDS TERMINATED BY ','

hive> load data local inpath '/home/vijay/hadoop/books.csv' into table books;



```
Time taken: 0.07 seconds
hive>  load data local inpath '/home/vijay/hadoop/customers.csv' into table customers;
FAILED: SemanticException Line 1:24 Invalid path ''/home/vijay/hadoop/customers.csv'': No files matching path file:/home/vijay/hadoop/customers.csv
hive> show tables;
OK
tab_name
customers
order_products
orders
orders_1
orders_2
orders_3
Time taken: 0.017 seconds, Fetched: 6 row(s)
hive> load data local inpath '/home/vijay/hadoop/customers.csv' into table customers;
FAILED: SemanticException Line 1:23 Invalid path ''/home/vijay/hadoop/customers.csv'': No files matching path file:/home/vijay/hadoop/customers.csv
hive> load data local inpath '/home/vijay/hadoop/customers(1).csv' into table customers;
FAILED: SemanticException Line 1:23 Invalid path ''/home/vijay/hadoop/customers(1).csv'': No files matching path file:/home/vijay/hadoop/customers(1).csv
hive> load data local inpath '/home/vijay/hadoop/ load data local inpath '/home/vijay/hadoop/customers.csv' into table customers;' into table customers;
MismatchedTokenException(14!=162)
        at org.antlr.runtime.BaseRecognizer.recoverFromMismatchedToken(BaseRecognizer.java:617)
        at org.antlr.runtime.BaseRecognizer.match(BaseRecognizer.java:115)
        at org.apache.hadoop.hive.ql.parse.HiveParser.loadStatement(HiveParser.java:2684)
        at org.apache.hadoop.hive.ql.parse.HiveParser.execStatement(HiveParser.java:2410)
        at org.apache.hadoop.hive.ql.parse.HiveParser.statement(HiveParser.java:1420)
        at org.apache.hadoop.hive.ql.parse.ParseDriver.parse(ParseDriver.java:220)
        at org.apache.hadoop.hive.ql.parse.ParseUtils.parse(ParseUtils.java:74)
        at org.apache.hadoop.hive.ql.parse.ParseUtils.parse(ParseUtils.java:67)
        at org.apache.hadoop.hive.ql.Driver.compile(Driver.java:616)
        at org.apache.hadoop.hive.ql.Driver.compileInternal(Driver.java:1826)
        at org.apache.hadoop.hive.ql.Driver.compileAndRespond(Driver.java:1773)
        at org.apache.hadoop.hive.ql.Driver.compileAndRespond(Driver.java:1768)
        at org.apache.hadoop.hive.ql.reexec.ReExecDriver.compileAndRespond(ReExecDriver.java:126)
        at org.apache.hadoop.hive.ql.reexec.ReExecDriver.run(ReExecDriver.java:214)
        at org.apache.hadoop.hive.cli.CliDriver.processLocalCmd(CliDriver.java:239)
        at org.apache.hadoop.hive.cli.CliDriver.processCmd(CliDriver.java:188)
        at org.apache.hadoop.hive.cli.CliDriver.processLine(CliDriver.java:402)
        at org.apache.hadoop.hive.cli.CliDriver.executeDriver(CliDriver.java:821)
        at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:759)
        at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:683)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:226)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:141)
FAILED: ParseException line 1:68 mismatched input '/' expecting INTO near ''/home/vijay/hadoop/ load data local inpath '' in load statement
hive> load data local inpath '/home/vijay/hadoop/customers(1).csv' into table customers;
FAILED: SemanticException Line 1:23 Invalid path ''/home/vijay/hadoop/customers(1).csv'': No files matching path file:/home/vijay/hadoop/customers(1).csv
hive> load data local inpath '/home/vijay/hadoop/ load data local inpath '/home/vijay/hadoop/customers.csv' into table customers;' into table customers;
MismatchedTokenException(14!=162)
        at org.antlr.runtime.BaseRecognizer.recoverFromMismatchedToken(BaseRecognizer.java:617)
        at org.antlr.runtime.BaseRecognizer.match(BaseRecognizer.java:115)
        at org.apache.hadoop.hive.ql.parse.HiveParser.loadStatement(HiveParser.java:2684)
        at org.apache.hadoop.hive.ql.parse.HiveParser.execStatement(HiveParser.java:2410)
        at org.apache.hadoop.hive.ql.parse.HiveParser.statement(HiveParser.java:1420)
```

PROGRAM 12

Install PIG, run basic PIG commands. (DDL and DML)

Download Software Pig
https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz

Steps to Install Apache Pig version (0.17.0) on Ubuntu 18.04.4 LTS
Please follow the below steps to install Apache Pig.

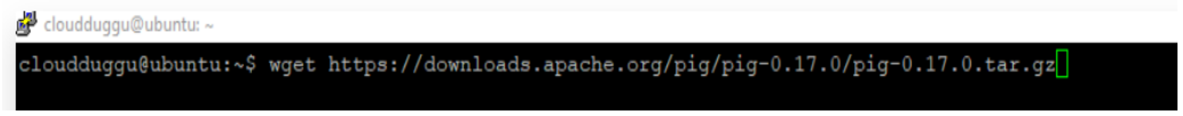
Step 1. Please verify if Hadoop is installed.

Step 2. Please verify if Java is installed.

Step 3. Please download Pig 0.17.0 from the below link.

On Linux: $wget https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz

On Windows: https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz

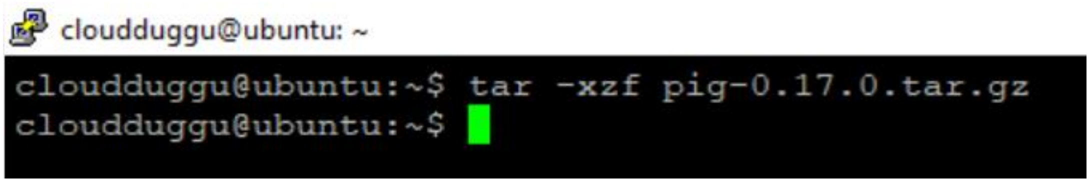

clouddugu@ubuntu: ~

```
cloudduggu@ubuntu:~$ wget https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
```

Step 4. Now we will extract the tar file by using the below command and rename the folder to pig to make it meaningful.

$tar -xzf pig-0.17.0.tar.gz

$mv pig-0.17.0 pig


cloudduggu@ubuntu: ~

```
cloudduggu@ubuntu:~$ tar -xzf pig-0.17.0.tar.gz
cloudduggu@ubuntu:~$
```

Step 5. Now edit the .bashrc file to update the environment variable of Apache Pig so that it can be accessed from any directory.

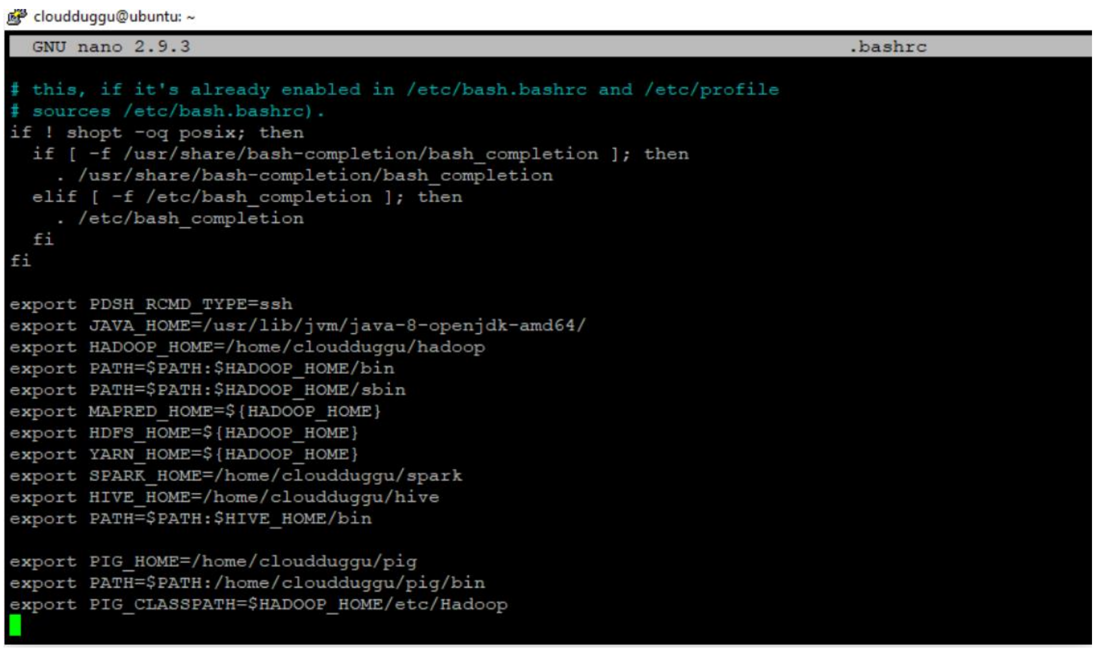
$nano .bashrc

Add below lines.

export PIG_HOME=/home/cloudduggu/pig

export PATH=$PATH:/home/cloudduggu/pig/bin

export PIG_CLASSPATH=$HADOOP_HOME/etc/Hadoop



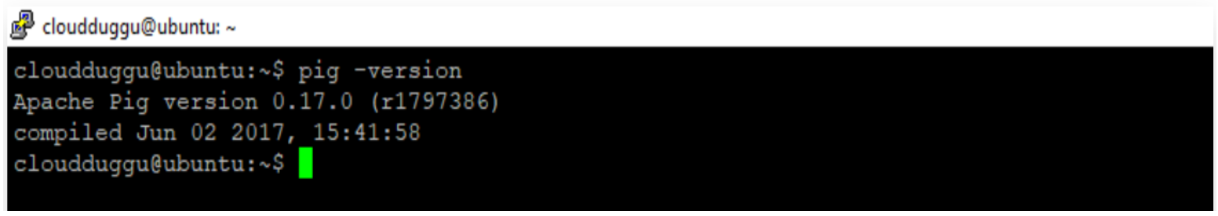
```
cloudduggu@ubuntu: ~
  GNU nano 2.9.3                                                        .bashrc

# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export PDSH_RCMD_TYPE=ssh
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
export HADOOP_HOME=/home/cloudduggu/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export MAPRED_HOME=${HADOOP_HOME}
export HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export SPARK_HOME=/home/cloudduggu/spark
export HIVE_HOME=/home/cloudduggu/hive
export PATH=$PATH:$HIVE_HOME/bin

export PIG_HOME=/home/cloudduggu/pig
export PATH=$PATH:/home/cloudduggu/pig/bin
export PIG_CLASSPATH=$HADOOP_HOME/etc/Hadoop
```

Save the changes by pressing CTRL + O and exit from the nano editor by pressing CTRL + X.

Step 6. Run source command to update changes in the same terminal.

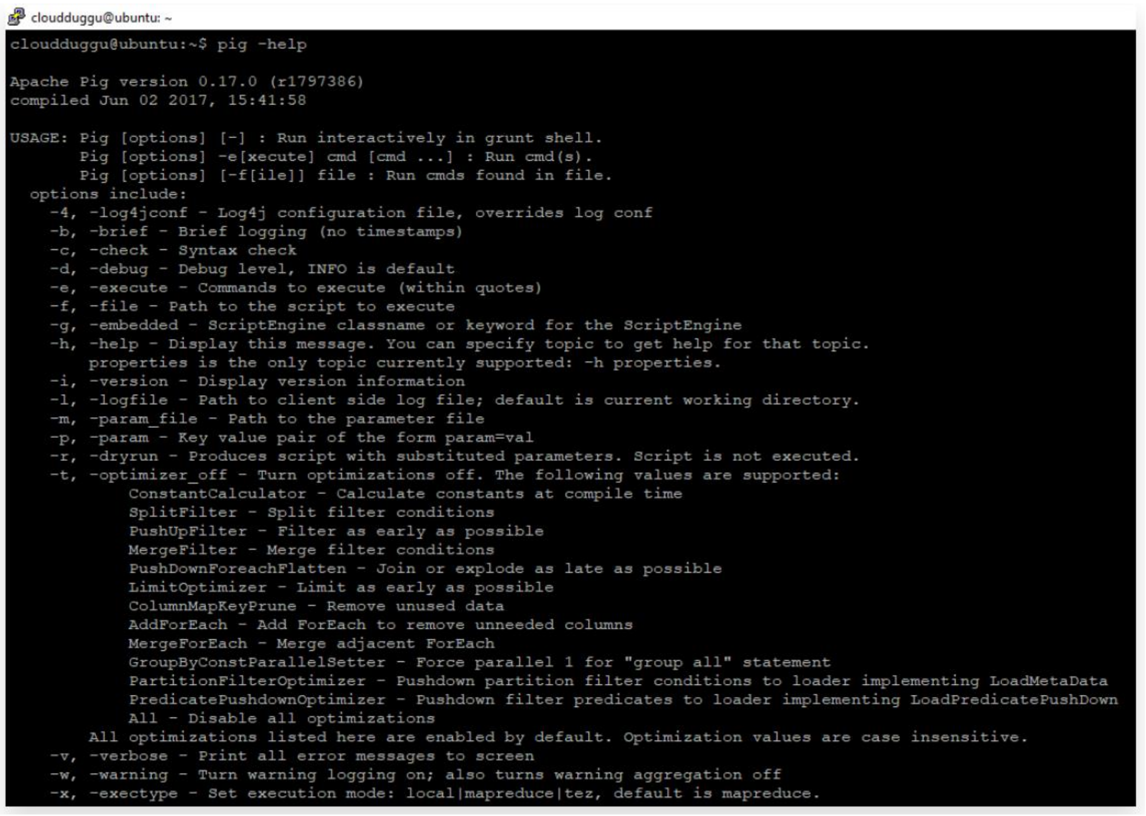
$source .bashrc

Step 7. Now run the pig version command to make sure that Pig is installed properly.



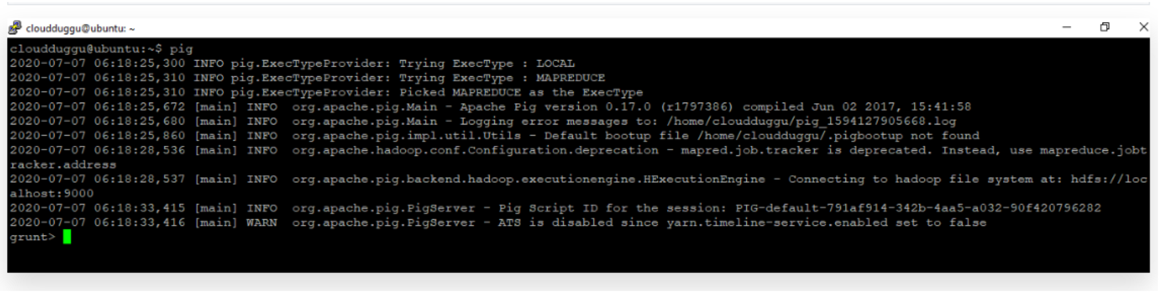
```
cloudduggu@ubuntu: ~
cloudduggu@ubuntu:~$ pig -version
Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
cloudduggu@ubuntu:~$
```

Step 8. Run pig help command to see all pig command options.



```
cloudduggu@ubuntu:~$ pig -help

Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58

USAGE: Pig [options] [-] : Run interactively in grunt shell.
       Pig [options] -e[xecute] cmd [cmd ...] : Run cmd(s).
       Pig [options] [-f[ile]] file : Run cmds found in file.
  options include:
    -4, -log4jconf - Log4j configuration file, overrides log conf
    -b, -brief - Brief logging (no timestamps)
    -c, -check - Syntax check
    -d, -debug - Debug level, INFO is default
    -e, -execute - Commands to execute (within quotes)
    -f, -file - Path to the script to execute
    -g, -embedded - ScriptEngine classname or keyword for the ScriptEngine
    -h, -help - Display this message. You can specify topic to get help for that topic.
        properties is the only topic currently supported: -h properties.
    -i, -version - Display version information
    -l, -logfile - Path to client side log file; default is current working directory.
    -m, -param_file - Path to the parameter file
    -p, -param - Key value pair of the form param=val
    -r, -dryrun - Produces script with substituted parameters. Script is not executed.
    -t, -optimizer_off - Turn optimizations off. The following values are supported:
            ConstantCalculator - Calculate constants at compile time
            SplitFilter - Split filter conditions
            PushUpFilter - Filter as early as possible
            MergeFilter - Merge filter conditions
            PushDownForeachFlatten - Join or explode as late as possible
            LimitOptimizer - Limit as early as possible
            ColumnMapKeyPrune - Remove unused data
            AddForEach - Add ForEach to remove unneeded columns
            MergeForEach - Merge adjacent ForEach
            GroupByConstParallelSetter - Force parallel 1 for "group all" statement
            PartitionFilterOptimizer - Pushdown partition filter conditions to loader implementing LoadMetaData
            PredicatePushdownOptimizer - Pushdown filter predicates to loader implementing LoadPredicatePushDown
            All - Disable all optimizations
        All optimizations listed here are enabled by default. Optimization values are case insensitive.
    -v, -verbose - Print all error messages to screen
    -w, -warning - Turn warning logging on; also turns warning aggregation off
    -x, -exectype - Set execution mode: local|mapreduce|tez, default is mapreduce.
```

Step 9. Now start Pig grunt shell (grunt shell is used to execute Pig Latin scripts).



```
cloudduggu@ubuntu:~$ pig
2020-07-07 06:18:25,300 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2020-07-07 06:18:25,310 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
2020-07-07 06:18:25,310 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2020-07-07 06:18:25,672 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2020-07-07 06:18:25,680 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/cloudduggu/pig_1594127905668.log
2020-07-07 06:18:25,860 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/cloudduggu/.pigbootup not found
2020-07-07 06:18:28,536 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobt
racker.address
2020-07-07 06:18:28,537 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://loc
alhost:9000
2020-07-07 06:18:33,415 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-791af914-342b-4aa5-a032-90f420796282
2020-07-07 06:18:33,416 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
grunt>
```

To start Pig Grunt type :

$pig -x local

It will start Pig Grunt shell:

grunt>

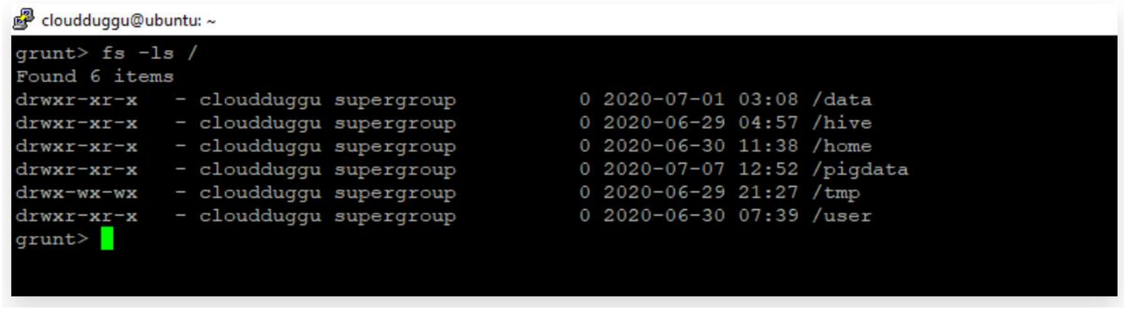
HDFS commands in Pig Grunt

Let us see few HDFS commands from the Pig Grunt shell.

fs -ls /
This command will print all directories present in HDFS "/".

Syntax:
grunt> fs subcommand subcommand_parameters;

Command:
grunt> fs -ls /



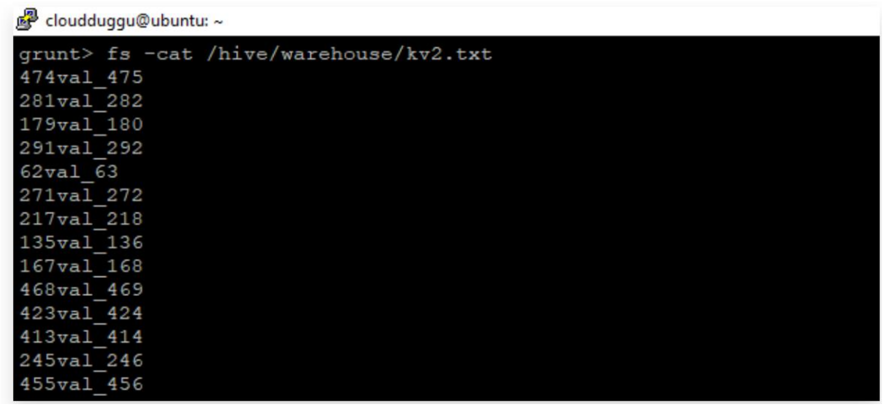
```
grunt> fs -ls /
Found 6 items
drwxr-xr-x   - cloudduggu supergroup          0 2020-07-01 03:08 /data
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-29 04:57 /hive
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-30 11:38 /home
drwxr-xr-x   - cloudduggu supergroup          0 2020-07-07 12:52 /pigdata
drwx-wx-wx   - cloudduggu supergroup          0 2020-06-29 21:27 /tmp
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-30 07:39 /user
grunt>
```

fs -cat
This command will print the content of a file present in HDFS.

Syntax:
grunt> fs subcommand subcommand_parameters

Command:
grunt> fs -cat /hive/warehouse/kv2.txt

Output:



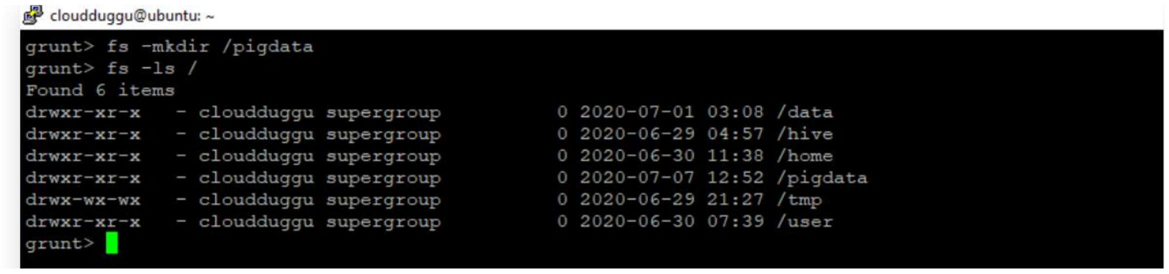
```
grunt> fs -cat /hive/warehouse/kv2.txt
474val_475
281val_282
179val_180
291val_292
62val_63
271val_272
217val_218
135val_136
167val_168
468val_469
423val_424
413val_414
245val_246
455val_456
```

fs -mkdir
This command will create a directory in HDFS.

Syntax:
grunt> fs subcommand subcommand_parameters

Command:
grunt> fs -mkdir /pigdata

Output:



cloudduggu@ubuntu: ~

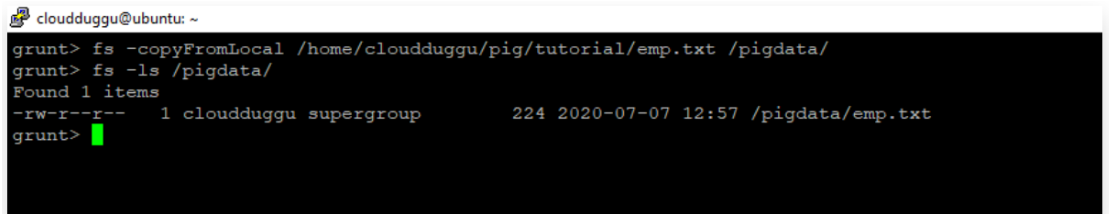
```
grunt> fs -mkdir /pigdata
grunt> fs -ls /
Found 6 items
drwxr-xr-x   - cloudduggu supergroup          0 2020-07-01 03:08 /data
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-29 04:57 /hive
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-30 11:38 /home
drwxr-xr-x   - cloudduggu supergroup          0 2020-07-07 12:52 /pigdata
drwx-wx-wx   - cloudduggu supergroup          0 2020-06-29 21:27 /tmp
drwxr-xr-x   - cloudduggu supergroup          0 2020-06-30 07:39 /user
grunt>
```

fs -copyFromLocal
This command will copy a file from the local system to HDFS.

Syntax:
grunt> fs subcommand subcommand_parameters

Command:
grunt> fs -copyFromLocal /home/cloudduggu/pig/tutorial/emp.txt /pigdata/

Output:



cloudduggu@ubuntu: ~

```
grunt> fs -copyFromLocal /home/cloudduggu/pig/tutorial/emp.txt /pigdata/
grunt> fs -ls /pigdata/
Found 1 items
-rw-r--r--   1 cloudduggu supergroup        224 2020-07-07 12:57 /pigdata/emp.txt
grunt>
```

Shell commands in Pig Grunt

We can use the Pig Grunt shell to run the basic shell command. We can invoke any shell commands using sh.

Let us see few Shell commands from the Pig Grunt shell. We cannot execute those commands which are part of the shell environment such as –cd.

sh ls

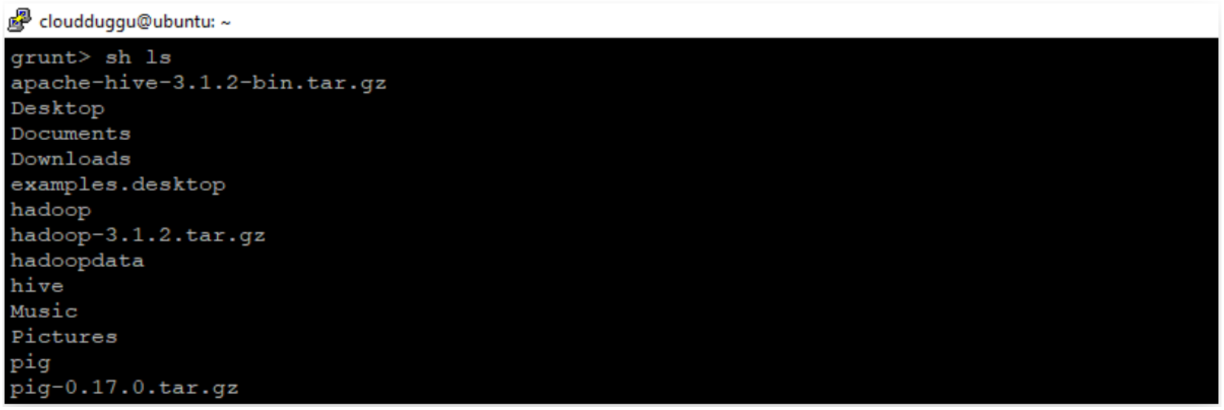
This command will list all directories/files.

Syntax:

grunt> sh subcommand subcommand_parameters

Command:

grunt> sh ls



```
cloudduggu@ubuntu: ~
grunt> sh ls
apache-hive-3.1.2-bin.tar.gz
Desktop
Documents
Downloads
examples.desktop
hadoop
hadoop-3.1.2.tar.gz
hadoopdata
hive
Music
Pictures
pig
pig-0.17.0.tar.gz
```

sh cat

This command will print the content of a file.

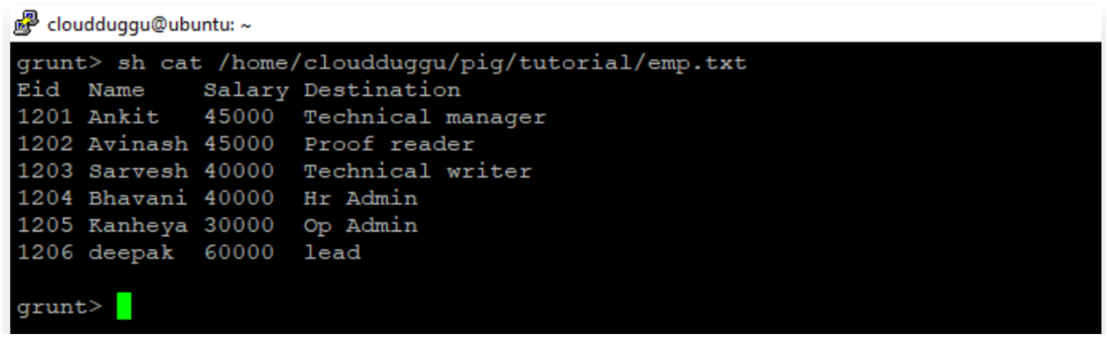
Syntax:

grunt> sh subcommand subcommand_parameters

Command:

grunt> sh cat

Output:



cloudduggu@ubuntu: ~

```
grunt> sh cat /home/cloudduggu/pig/tutorial/emp.txt
Eid  Name    Salary Destination
1201 Ankit   45000  Technical manager
1202 Avinash 45000  Proof reader
1203 Sarvesh 40000  Technical writer
1204 Bhavani 40000  Hr Admin
1205 Kanheya 30000  Op Admin
1206 deepak  60000  lead

grunt>
```

Clear Command

Clear command is used to clear the screen of the Grunt shell.

Syntax:
grunt> Clear

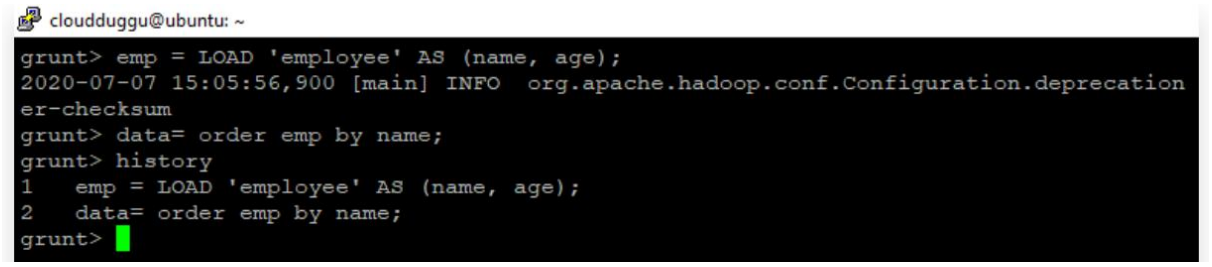
Command:
grunt> Clear

History Command
The history command is used to clear the screen of the Grunt shell.

Syntax:
grunt> history

Command:
grunt> history

Output:



cloudduggu@ubuntu: ~

```
grunt> emp = LOAD 'employee' AS (name, age);
2020-07-07 15:05:56,900 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation
er-checksum
grunt> data= order emp by name;
grunt> history
1   emp = LOAD 'employee' AS (name, age);
2   data= order emp by name;
grunt>
```

Set Command

The SET command is used to assign values to keys that are case sensitive. In case the SET command is used without providing arguments then all other system properties and configurations are printed.

Syntax:

```
grunt> set [key 'value']
```

Command:

```
grunt> SET debug 'on'
grunt> SET job.name 'my job'
grunt> SET default_parallel 100
```

EXEC Command

Exec command is used to execute Pig script from Grunt shell.

Please make sure the history server is running. You can verify in the JPS command output. This service “JobHistoryServer” should be running otherwise you can start it using the below command.

```
$ /home/cloudduggu/hadoop/sbin$./mr-jobhistory-daemon.sh start historyserver
```

Let us assume that we have a file name “emp.txt” which is present on HDFS /pigdata/ directory. Now we want to use this file and project its content using Pig script.

Content of “emp.txt”:

```
  201,Wick,Google
  203,John,Facebook
204,Partick,Instagram
205,Hema,Google
206,Holi,Facebook
207,Michael,Instagram
208,Michael,Instagram
209,Chung,Instagram
210,Anna,Instagram
```

Now we will create an “emp_script.pig” script file which will have the below statements to process data and put this file on the same location of HDFS that is /pigdata/.

Content of “emp_script.pig”:

```
employee = LOAD 'hdfs://localhost:9000/pigdata/emp.txt' USING PigStorage(',')
as (empid:int,empname:chararray,salary:int);
dump employee;
```

Now we will start the Pig Grunt shell and run the script.

Syntax:

```
grunt> exec [–param param_name = param_value] [–param_file file_name] [script]
```

Command:

```
$pig

grunt> exec hdfs:///pigdata/emp_script.pig
```

Kill Command

The kill command will attempt to kill any MapReduce jobs associated with the Pig job

Syntax:

```
grunt> kill JobId
```

Command:

```
grunt> kill job_500
```

Syntax:

```
grunt> run [–param param_name = param_value] [–param_file file_name] script
```

Command:

```
grunt> run hdfs:///pigdata/emp_script.pig
```

PROGRAM 13

Install Apache Spark (single and multi-node cluster) and learn to use basic commands and RDD creation and Implement SVM, Regression, classification etc. using Spark MLib.

Write and Execute word count program using spark shell.

Step 1. Install scala

```
$sudo apt install scala
```

Step 2. Check the version of scala

```
$scala -version
```

Step 3. Install python3·pip

```
$sudo apt install python3·pip
```

Step 4. Install pyspark

```
$pip install pyspark
```

Step 5. Download spark 3.3.1bin hadoop3.tgz from apache website

```
$sudo wget https://dlcdn.apache.org/spark/spark·3.3.1/spark 3.3.1bin hadoop3.tgz
```

Step 6. Unzip downloaded file

```
$sudo tar xvf spark-3.3.1bin-hadoop3.tgz
```

Step 7. Create a spark directory and move the content of the extracted file

```
$cd spark·3.3.1 bin/hadoop3

$sudo mv * /usr/local/spark
```

Step 8. Update bashrc and refresh it by using source command

```
$sudo nano -/.bashrc

        #SPARK Variables

        export PARK_HOME=/usr/local/spark export PATH=$PATH:SSPARK_HOME/bin export
        PATH=$PATH:SSPARK_HOME/sbin

        #SPARK Variables END

$source -/.bashrc
```

Step 2. Spark is ready to use, start spark-shell

$spark-shell

To stop shells for-all (ctrl+D)

$pyspark

Write and Execute word count program using spark shell.

Steps are used to learn how to perform wordcount using spark.

Step 1: Start the spark shell using the following command and wait for the prompt to appear

```
$spark-shell
```

Step 2: Create RDD from a file in HDFS, type the following on spark-shell, and press enter:

```
$var linesRDD = sc.textFile("/data/mr/wordcount/input/big.txt")
```

Step 3: Convert each record into word

```
$var wordsRDD = linesRDD.flatMap(_.split(" "))
```

Step 4: Convert each word into key-value pair

```
$var wordsKvRdd = wordsRDD.map((_, 1))
```

Step 5: Group By key and perform aggregation on each key:

```
$var wordCounts = wordsKvRdd.reduceByKey(_ + _ )
```

Step 6: Save the results into HDFS:

```
$wordCounts.saveAsTextFile("my_spark_shell_wc_output")
```