

TABLE OF CONTENTS

EXP. NO	DATE	TITLE OF THE EXPERIMENT	PAGE NO.	SIGNATURE
1		Download and installing Hadoop.		
2		Hadoop implementation of file management task.		
3		Run a basic word count program to understand MapReduce paradigms		
4		Data Preprocessing and Table creation		
5		SPARK - Commands for Data Frames .		
6		PYSPARK- JOIN and SQL Commands		

Ex No 1:

Download and Installing Hadoop

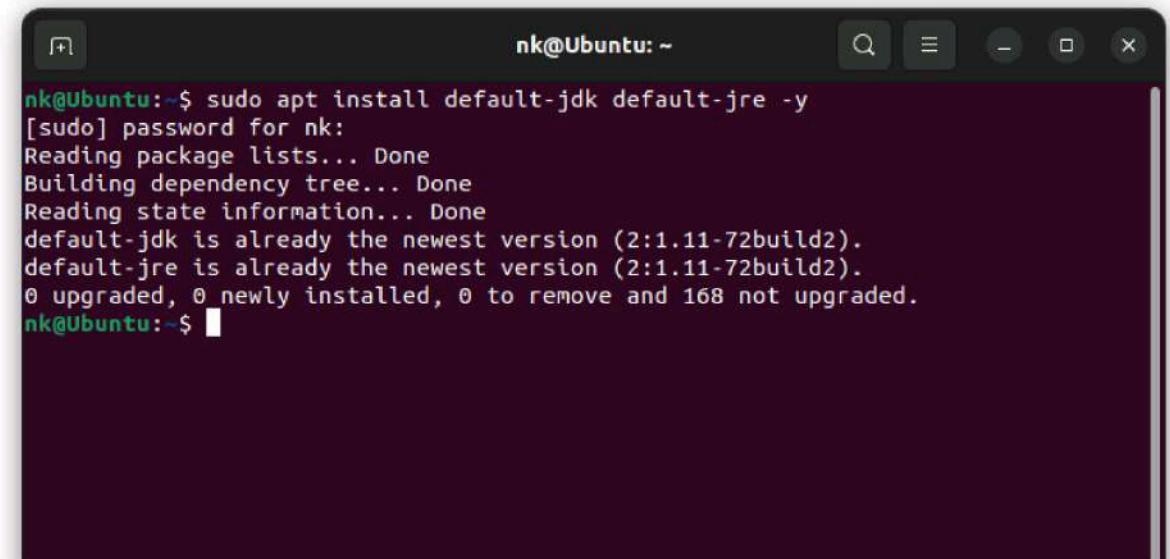
Date:

HADOOP COMMANDS

1. Install Java

1.1 Install the latest version of Java.

```
$ sudo apt install default-jdk default-jre -y
```

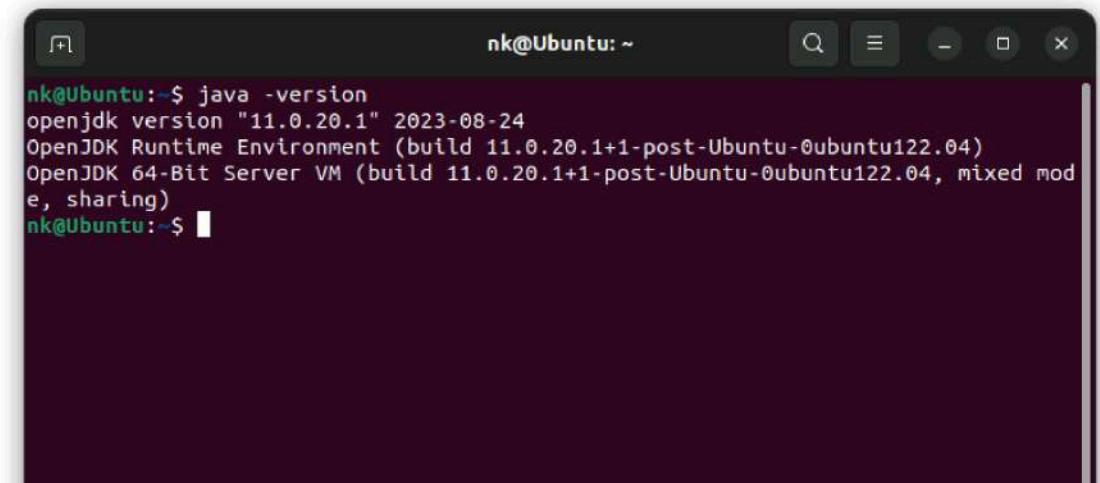


The screenshot shows a terminal window titled "nk@Ubuntu: ~". The user has run the command `sudo apt install default-jdk default-jre -y`. The output indicates that both packages are already at their newest versions (2:1.11-72build2). It also shows that 0 packages were upgraded, 0 were newly installed, 0 were removed, and 168 were not upgraded. The terminal window has a dark background and light-colored text.

```
nk@Ubuntu:~$ sudo apt install default-jdk default-jre -y
[sudo] password for nk:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
default-jdk is already the newest version (2:1.11-72build2).
default-jre is already the newest version (2:1.11-72build2).
0 upgraded, 0 newly installed, 0 to remove and 168 not upgraded.
nk@Ubuntu:~$
```

1.2 Verify the installed version of Java.

```
$ java -version
```



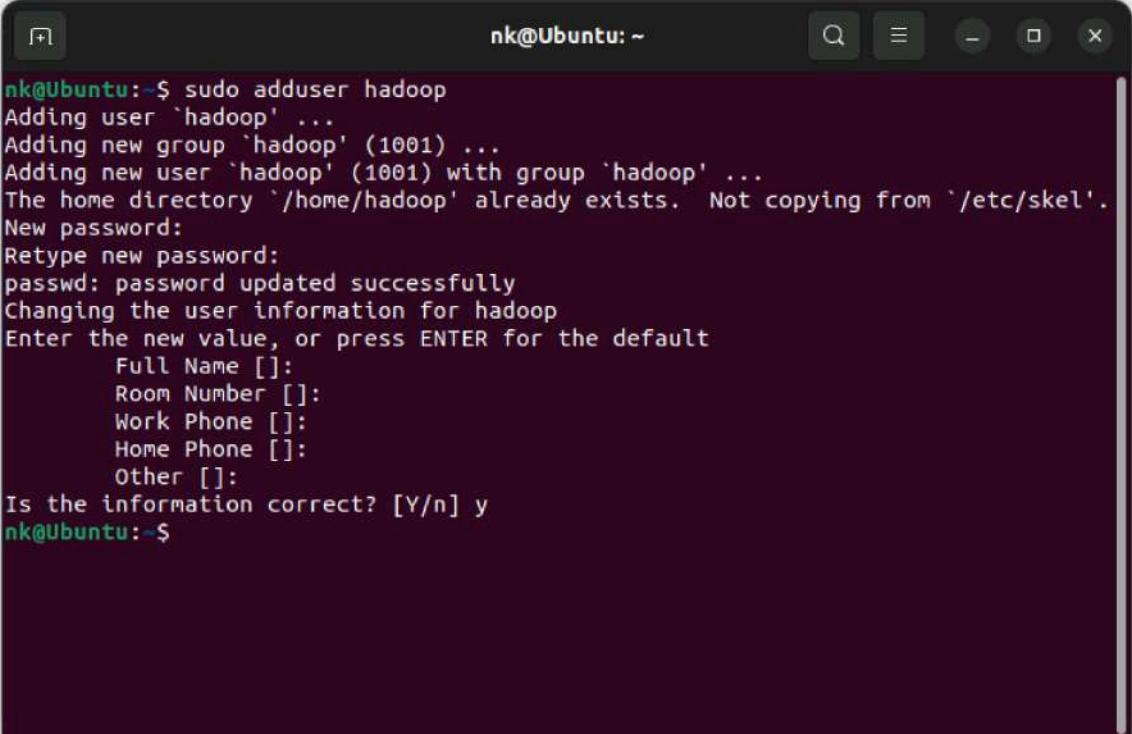
The screenshot shows a terminal window titled "nk@Ubuntu: ~". The user has run the command `java -version`. The output displays the Java version (openjdk 11.0.20.1), build date (2023-08-24), and the OpenJDK Runtime Environment and 64-Bit Server VM details. The terminal window has a dark background and light-colored text.

```
nk@Ubuntu:~$ java -version
openjdk version "11.0.20.1" 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
nk@Ubuntu:~$
```

2. Create Hadoop User and Configure Password-less SSH

2.1. Add a new user hadoop.

```
$ sudo adduser hadoop
```

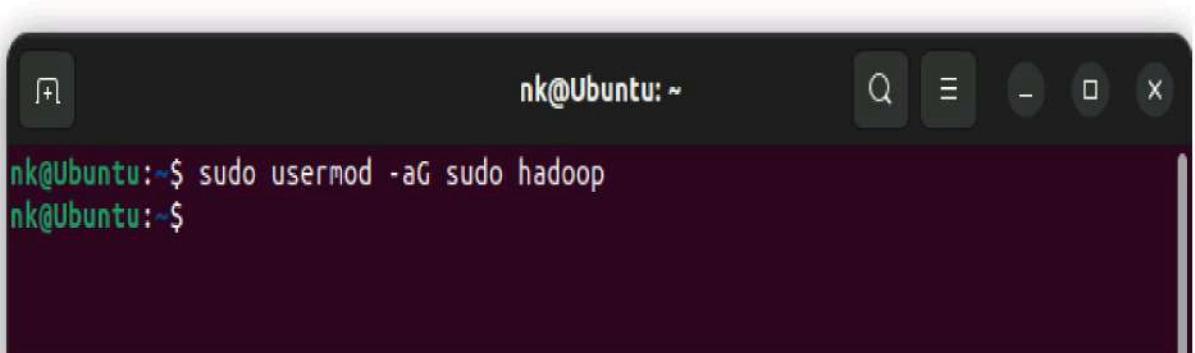


The screenshot shows a terminal window titled "nk@Ubuntu: ~". The command \$ sudo adduser hadoop is run, and the terminal displays the following output:

```
nk@Ubuntu:~$ sudo adduser hadoop
Adding user `hadoop' ...
Adding new group `hadoop' (1001) ...
Adding new user `hadoop' (1001) with group `hadoop' ...
The home directory `/home/hadoop' already exists. Not copying from `/etc/skel'.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
nk@Ubuntu:~$
```

2.2 Add the hadoop user to the sudo group.

```
$ sudo usermod -aG sudo Hadoop
```

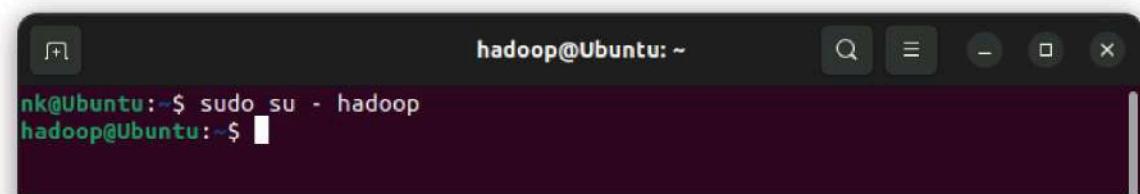


The screenshot shows a terminal window titled "nk@Ubuntu: ~". The command \$ sudo usermod -aG sudo hadoop is run, and the terminal displays the following output:

```
nk@Ubuntu:~$ sudo usermod -aG sudo hadoop
nk@Ubuntu:~$
```

2.2 Switch to the created user.

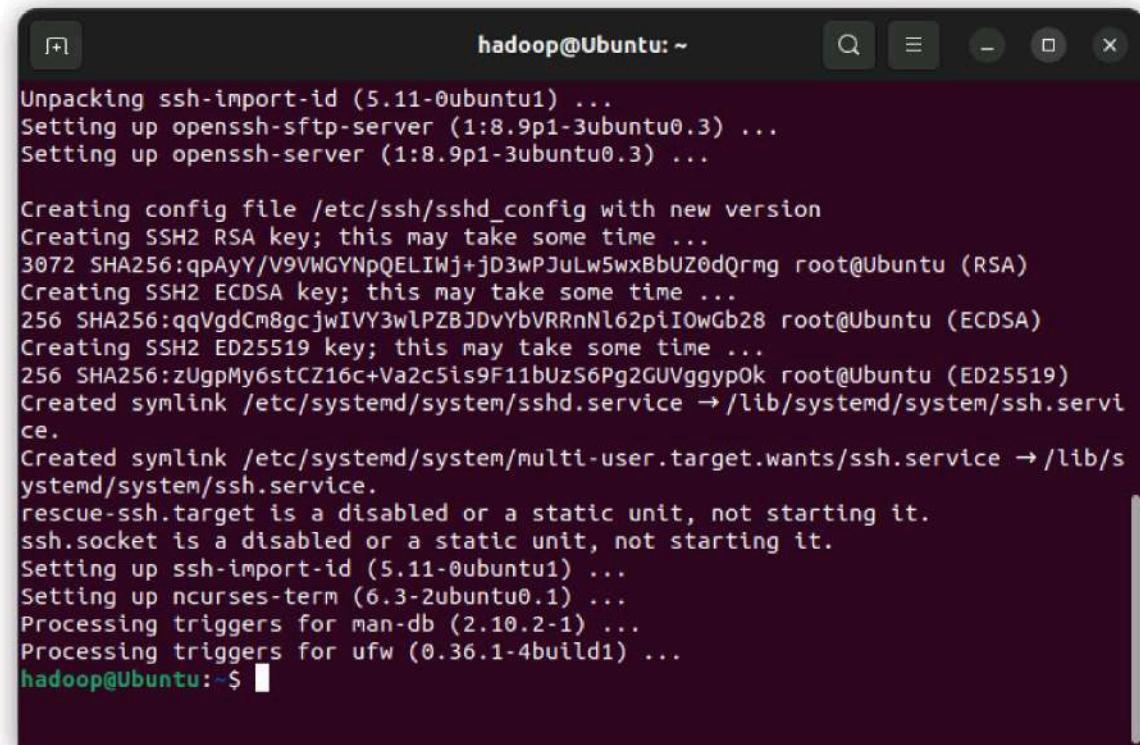
\$ sudo su – hadoop



```
nk@Ubuntu:~$ sudo su - hadoop
hadoop@Ubuntu:~$
```

2.3 Install the OpenSSH server and client.

\$ sudo apt install openssh-server openssh-client –y

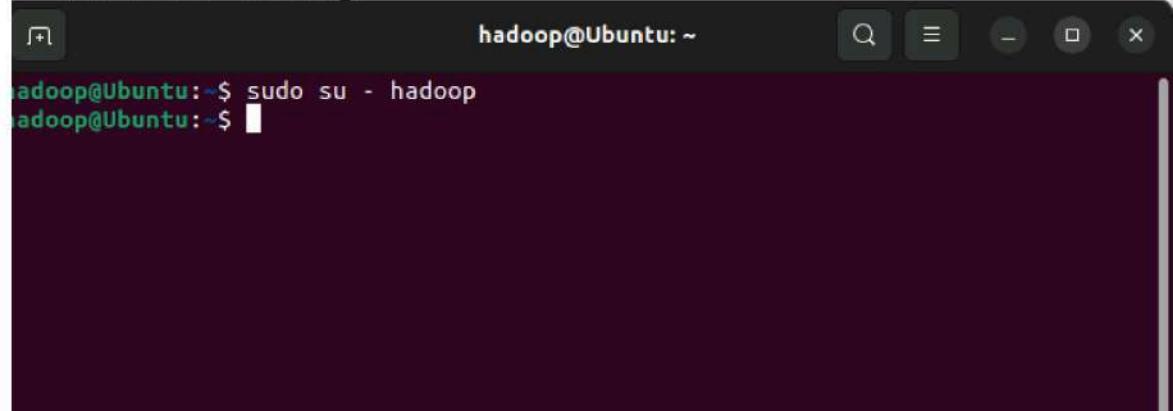


```
Unpacking ssh-import-id (5.11-0ubuntu1) ...
Setting up openssh-sftp-server (1:8.9p1-3ubuntu0.3) ...
Setting up openssh-server (1:8.9p1-3ubuntu0.3) ...

Creating config file /etc/ssh/sshd_config with new version
Creating SSH2 RSA key; this may take some time ...
3072 SHA256:qpAyY/V9VWGYNpQELIWj+jD3wPJUlw5wxBbUZ0dQrmg root@Ubuntu (RSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:qqVgdCm8gcjwIVY3wLPZBJDvYbVRNnL62piI0wGb28 root@Ubuntu (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:zUgpMy6stCZ16c+Va2c5is9F11bUzs6Pg2GUvgyp0k root@Ubuntu (ED25519)
Created symlink /etc/systemd/system/sshd.service → /lib/systemd/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants/ssh.service → /lib/systemd/system/ssh.service.
rescue-ssh.target is a disabled or a static unit, not starting it.
ssh.socket is a disabled or a static unit, not starting it.
Setting up ssh-import-id (5.11-0ubuntu1) ...
Setting up ncurses-term (6.3-2ubuntu0.1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for ufw (0.36.1-4build1) ...
hadoop@Ubuntu:~$
```

2.4 Switch to the created user.

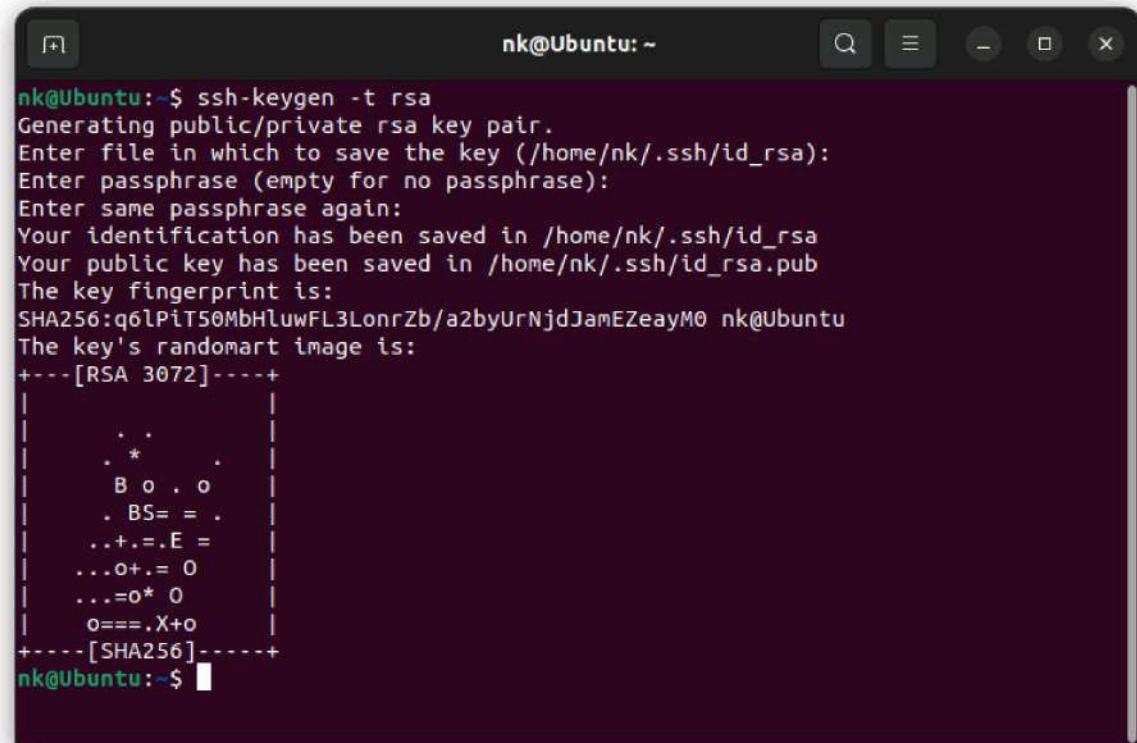
\$ sudo su – hadoop



```
hadoop@Ubuntu:~$ sudo su - hadoop
hadoop@Ubuntu:~$
```

2.5 Generate public and private key pairs.

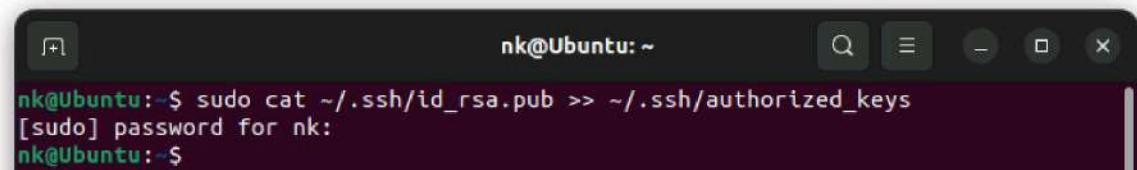
```
$ ssh-keygen -t rsa
```



```
nk@Ubuntu:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/nk/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nk/.ssh/id_rsa
Your public key has been saved in /home/nk/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:q6lPiT50MbHluwFL3LonrZb/a2byUrNjdJamEZeayM0 nk@Ubuntu
The key's randomart image is:
+---[RSA 3072]----+
| . .
| . *
| B o . o
| . BS= = .
| ..+.=E =
| ...o+=O
| ...=o* O
| o==..X+o
+---[SHA256]----+
nk@Ubuntu:~$
```

2.6 Add the generated public key from id_rsa.pub to authorized_keys.

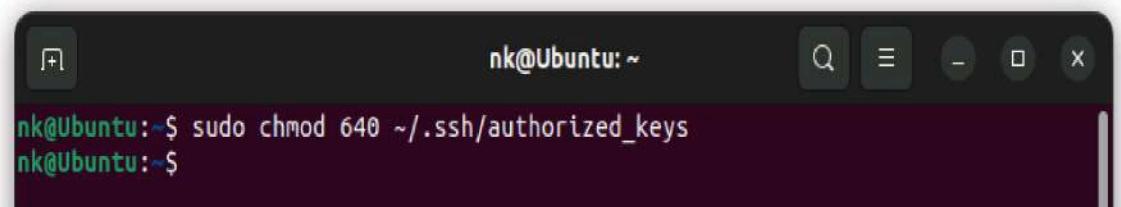
```
$ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```



```
nk@Ubuntu:~$ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
[sudo] password for nk:
nk@Ubuntu:~$
```

2.7 Change the permissions of the authorized_keys file.

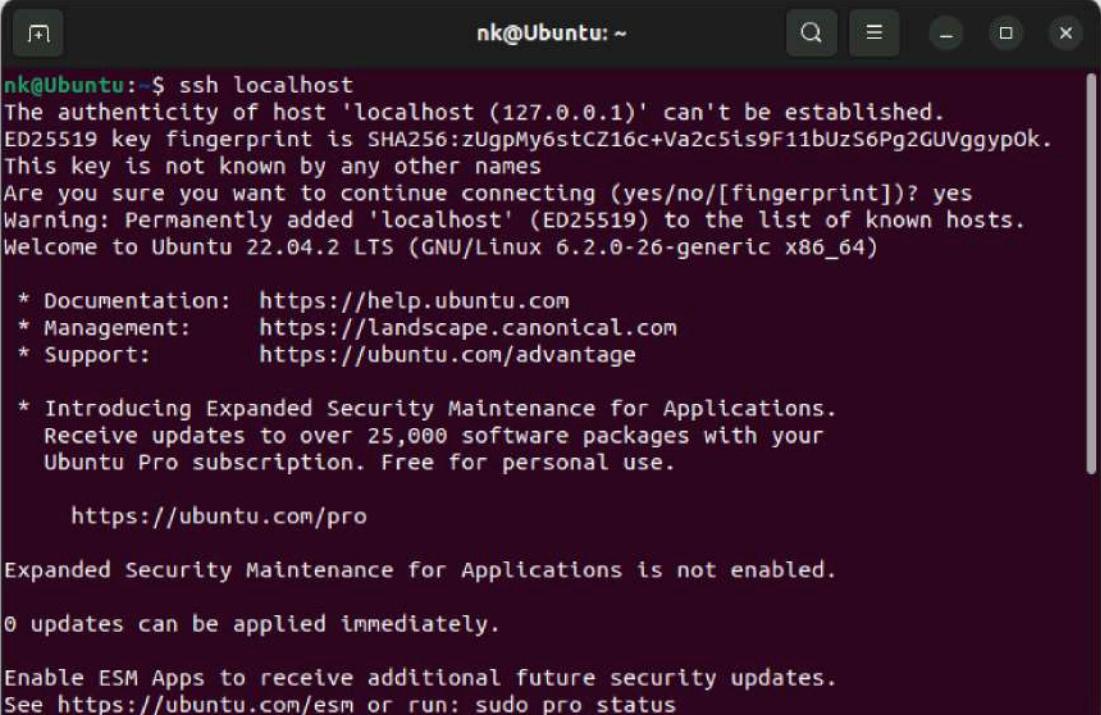
```
$ sudo chmod 640 ~/.ssh/authorized_keys
```



```
nk@Ubuntu:~$ sudo chmod 640 ~/.ssh/authorized_keys
nk@Ubuntu:~$
```

2.8 Verify if Password less SSH is functional

\$ ssh localhost



```
nk@Ubuntu:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ED25519 key fingerprint is SHA256:zUgpMy6stCZ16c+Va2c5is9F11bUzS6Pg2GUVggpok.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 6.2.0-26-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

   https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

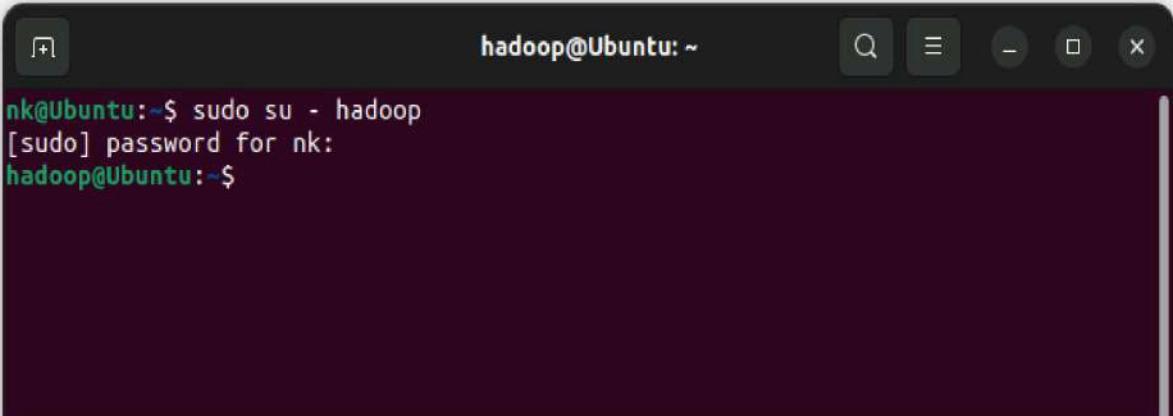
0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
```

3. Install Apache Hadoop

3.1 Log in with hadoop user.

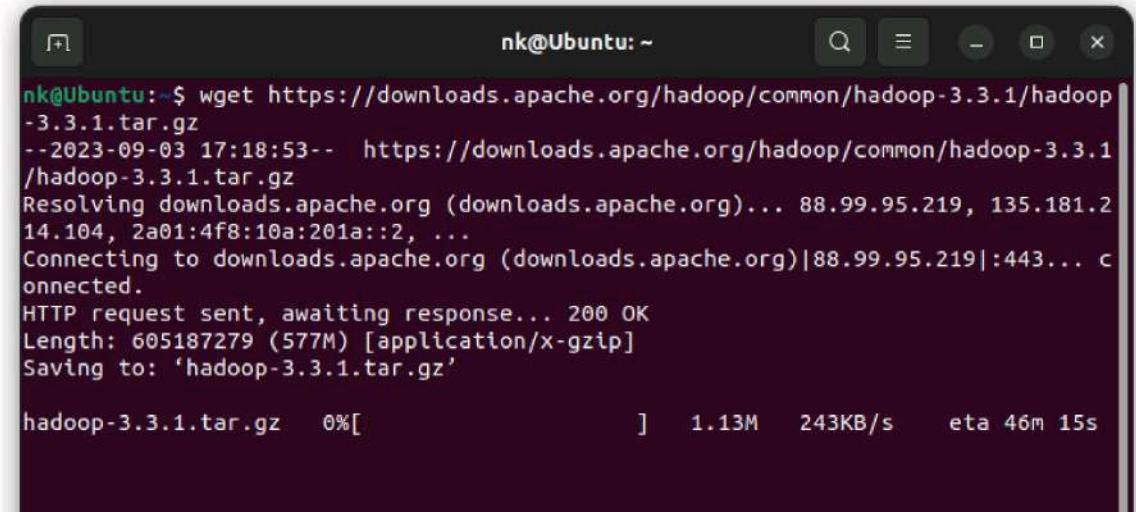
\$ sudo su – hadoop



```
nk@Ubuntu:~$ sudo su - hadoop
[sudo] password for nk:
hadoop@Ubuntu:~$
```

3.2 Download the latest stable version of Hadoop.

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
```



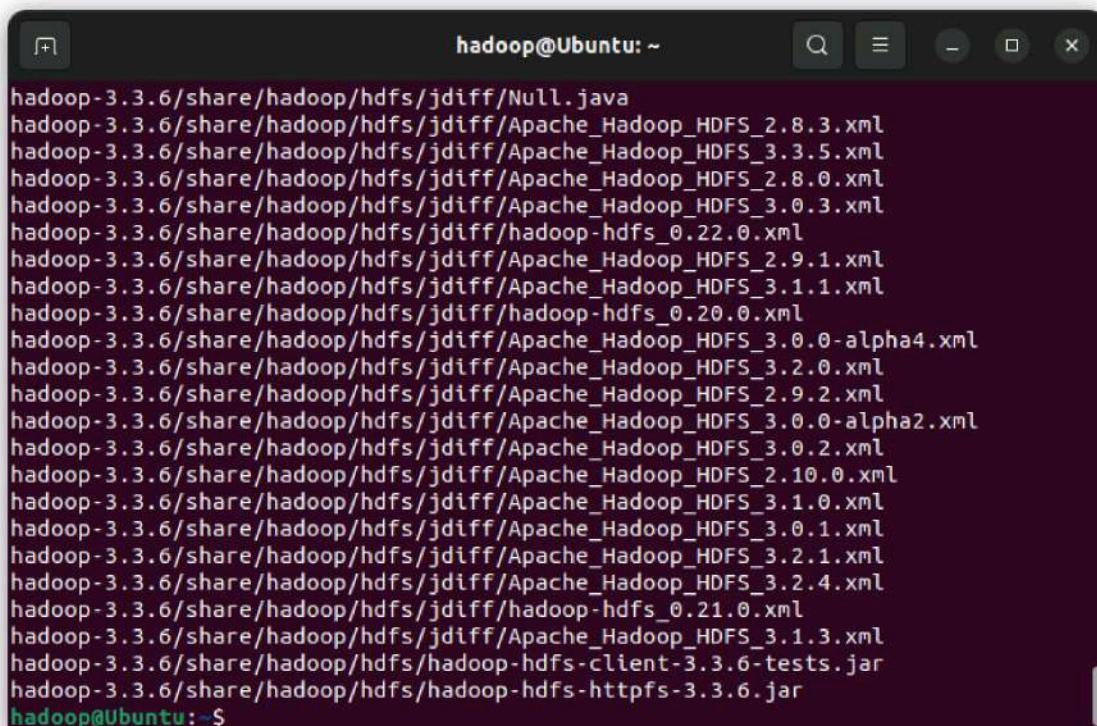
A screenshot of a terminal window titled "nk@Ubuntu: ~". The window shows the command \$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz being run. The output of the wget command is displayed, showing the progress of the download from the Apache mirror site. The terminal has a dark background with light-colored text and standard window controls at the top.

```
nk@Ubuntu:~$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
--2023-09-03 17:18:53-- https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.95.219, 135.181.2.14.104, 2a01:4f8:10a:201a::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.95.219|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 605187279 (577M) [application/x-gzip]
Saving to: 'hadoop-3.3.1.tar.gz'

hadoop-3.3.1.tar.gz    0%[          ]      1.13M   243KB/s eta 46m 15s
```

3.3 Extract the downloaded file.

```
$ tar -xvzf hadoop-3.3.1.tar.gz
```

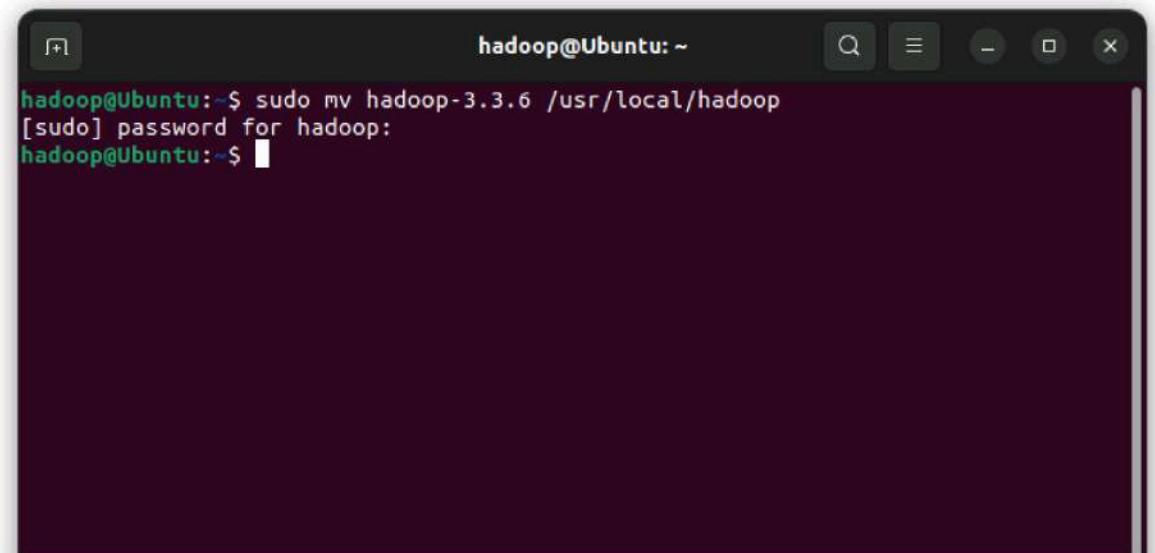


A screenshot of a terminal window titled "hadoop@Ubuntu: ~". The window shows the command \$ tar -xvzf hadoop-3.3.1.tar.gz being run. The output of the tar command is displayed, listing many XML configuration files extracted from the Hadoop distribution. The terminal has a dark background with light-colored text and standard window controls at the top.

```
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Null.java
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_2.8.3.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.3.5.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_2.8.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.0.3.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/hadoop-hdfs_0.22.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_2.9.1.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.1.1.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/hadoop-hdfs_0.20.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.0.0-alpha4.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.2.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_2.9.2.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.0.0-alpha2.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.0.2.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_2.10.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.1.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.0.1.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.2.1.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.2.4.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/hadoop-hdfs_0.21.0.xml
hadoop-3.3.6/share/hadoop/hdfs/jdiff/Apache_Hadoop_HDFS_3.1.3.xml
hadoop-3.3.6/share/hadoop/hdfs/hadoop-hdfs-client-3.3.6-tests.jar
hadoop-3.3.6/share/hadoop/hdfs/hadoop-hdfs-https-3.3.6.jar
hadoop@Ubuntu:~$
```

3.4 Move the extracted directory to the /usr/local/ directory.

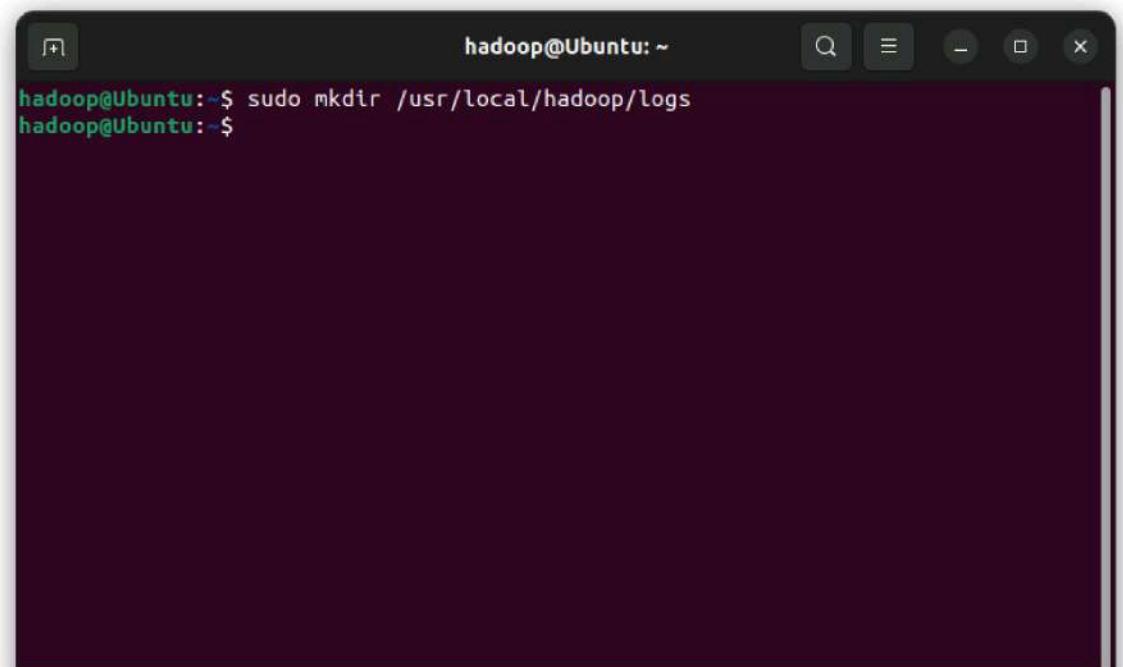
```
$ sudo mv hadoop-3.3.1 /usr/local/hadoop
```



```
hadoop@Ubuntu:~$ sudo mv hadoop-3.3.1 /usr/local/hadoop
[sudo] password for hadoop:
hadoop@Ubuntu:~$
```

3.5 Create a directory to store system logs.

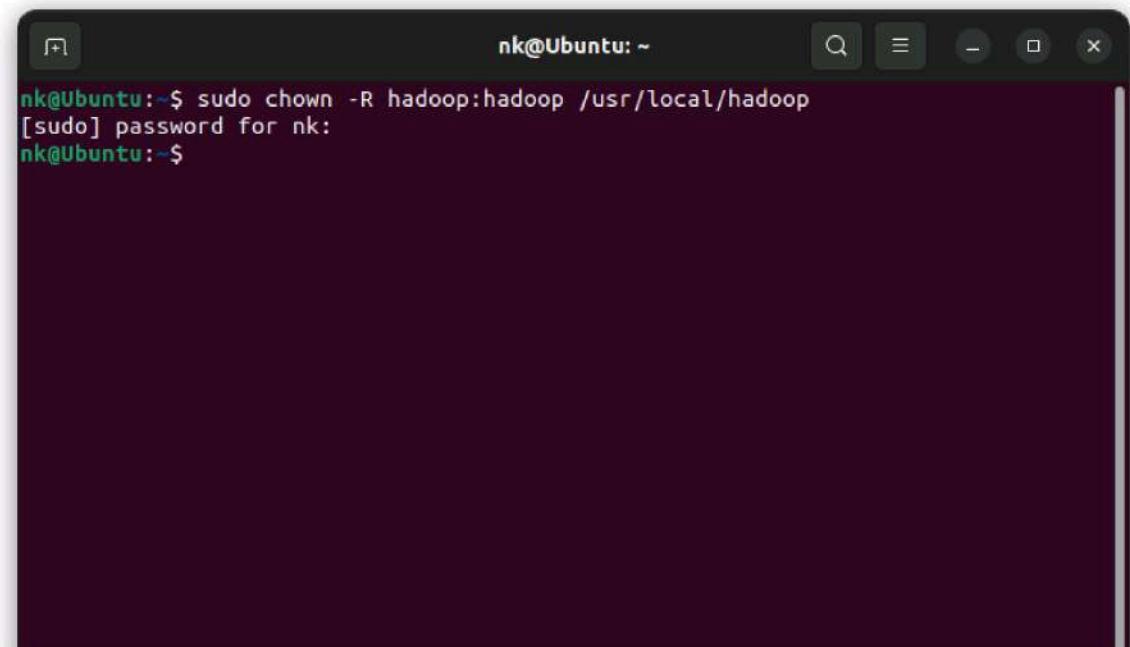
```
$ sudo mkdir /usr/local/hadoop/logs
```



```
hadoop@Ubuntu:~$ sudo mkdir /usr/local/hadoop/logs
hadoop@Ubuntu:~$
```

3.6 Change the ownership of the hadoop directory.

```
$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```



```
nk@Ubuntu:~$ sudo chown -R hadoop:hadoop /usr/local/hadoop
[sudo] password for nk:
nk@Ubuntu:~$
```

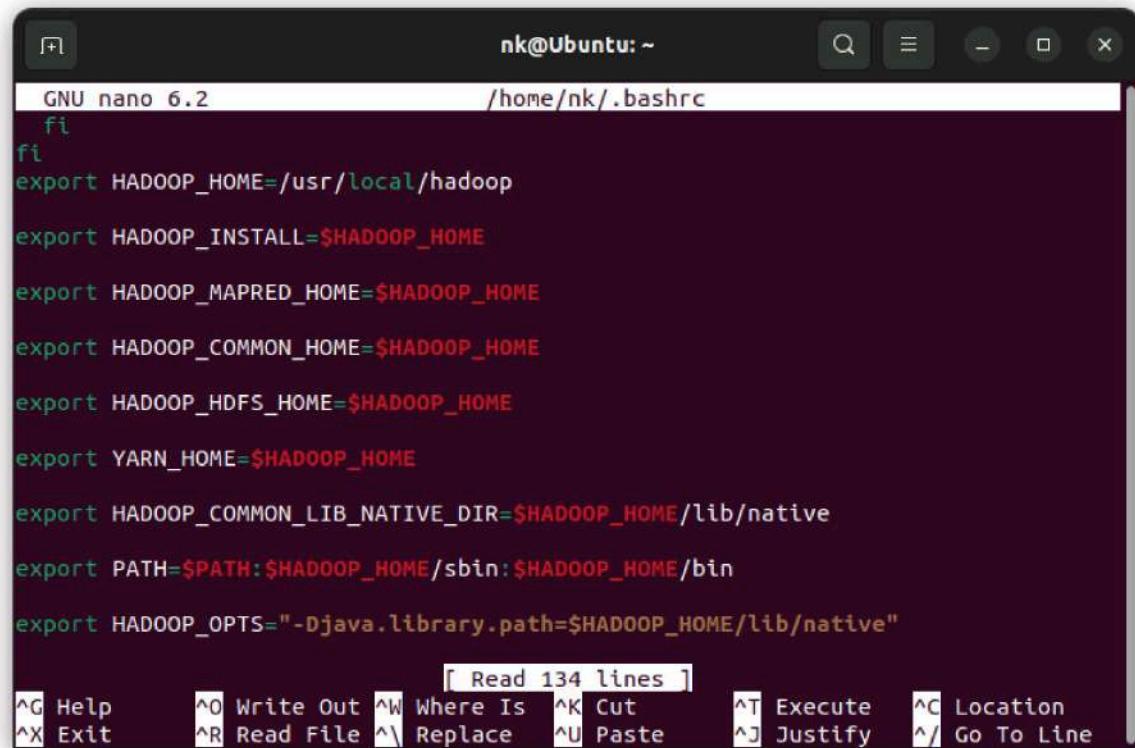
4 Configure Hadoop

4.1 Edit file `~/.bashrc` to configure the Hadoop environment variables.

```
$ sudo nano ~/.bashrc
```

Add the following lines to the file. Save and close the

```
file.  
export HADOOP_HOME=/usr/local/hadoop  
export HADOOP_INSTALL=$HADOOP_HOME  
export  
HADOOP_MAPRED_HOME=$HADOOP_HOME  
export  
HADOOP_COMMON_HOME=$HADOOP_HOME  
export  
HADOOP_HDFS_HOME=$HADOOP_HOME  
export YARN_HOME=$HADOOP_HOME  
export  
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/  
nat  
iveexport  
PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin  
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```



```
GNU nano 6.2                               /home/nk/.bashrc
fi
fi
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"

[ Read 134 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute   ^C Location
^X Exit      ^R Read File  ^\ Replace   ^U Paste    ^J Justify   ^/ Go To Line
```

Activate the environment variables.

```
$ source ~/.bashrc
```



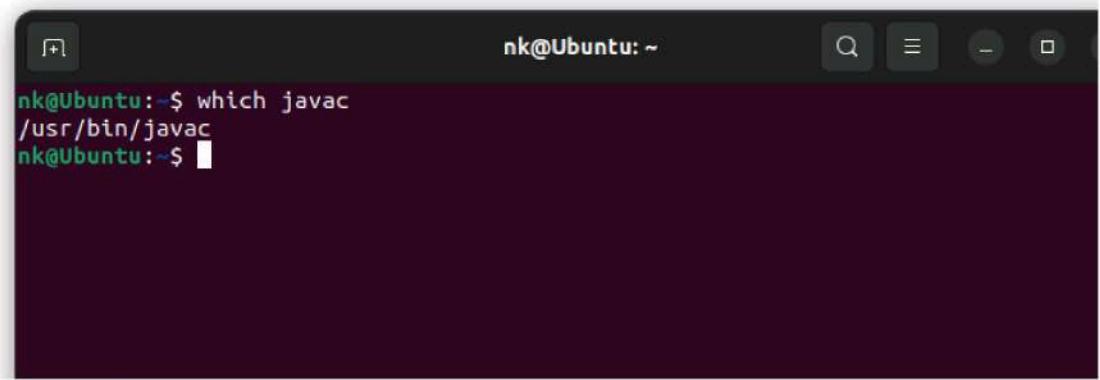
```
nk@Ubuntu:~$ source ~/.bashrc
nk@Ubuntu:~$
```

5 Configure Java Environment Variables

Hadoop has a lot of components that enable it to perform its core functions. To configure these components such as YARN, HDFS, MapReduce, and Hadoop-related project settings, you need to define Java environment variables in `hadoop-env.sh` configuration file.

5.1 Find the Java path.

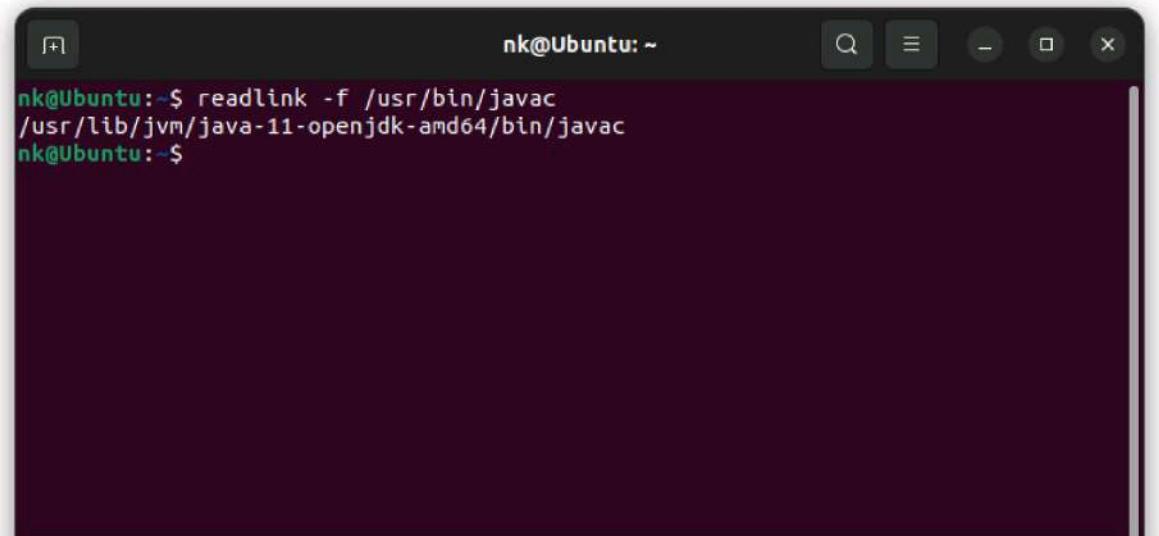
```
$ which javac
```



```
nk@Ubuntu:~$ which javac
/usr/bin/javac
nk@Ubuntu:~$
```

5.2 Find the OpenJDK directory.

\$ readlink -f /usr/bin/javac



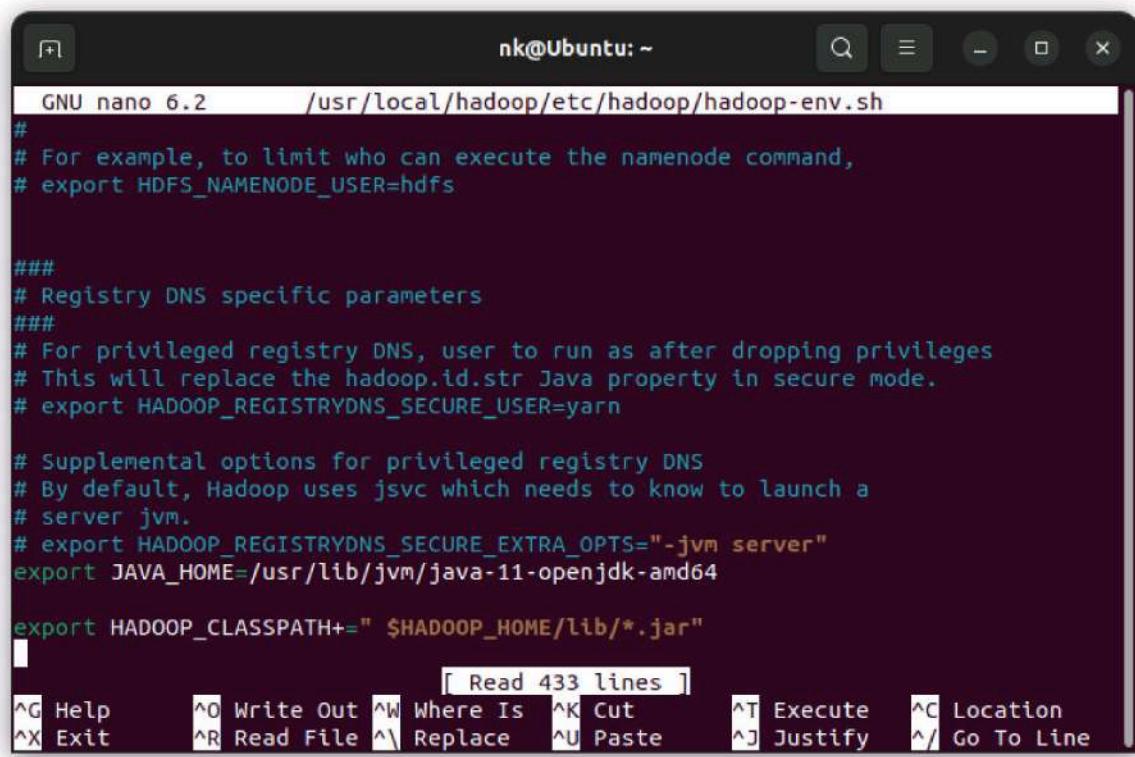
```
nk@Ubuntu:~$ readlink -f /usr/bin/javac
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac
nk@Ubuntu:~$
```

5.3 Edit the hadoop-env.sh file.

\$ sudo nano \$HADOOP_HOME/etc/hadoop/hadoop-env.sh

Add the following lines to the file. Then, close and save the file.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_CLASSPATH+="
$HADOOP_HOME/lib/*.jar"
```



```
GNU nano 6.2      /usr/local/hadoop/etc/hadoop-env.sh
#
# For example, to limit who can execute the namenode command,
# export HDFS_NAMENODE_USER=hdfs

###  
# Registry DNS specific parameters  
###  
# For privileged registry DNS, user to run as after dropping privileges  
# This will replace the hadoop.id.str Java property in secure mode.  
# export HADOOP_REGISTRYDNS_SECURE_USER=yarn

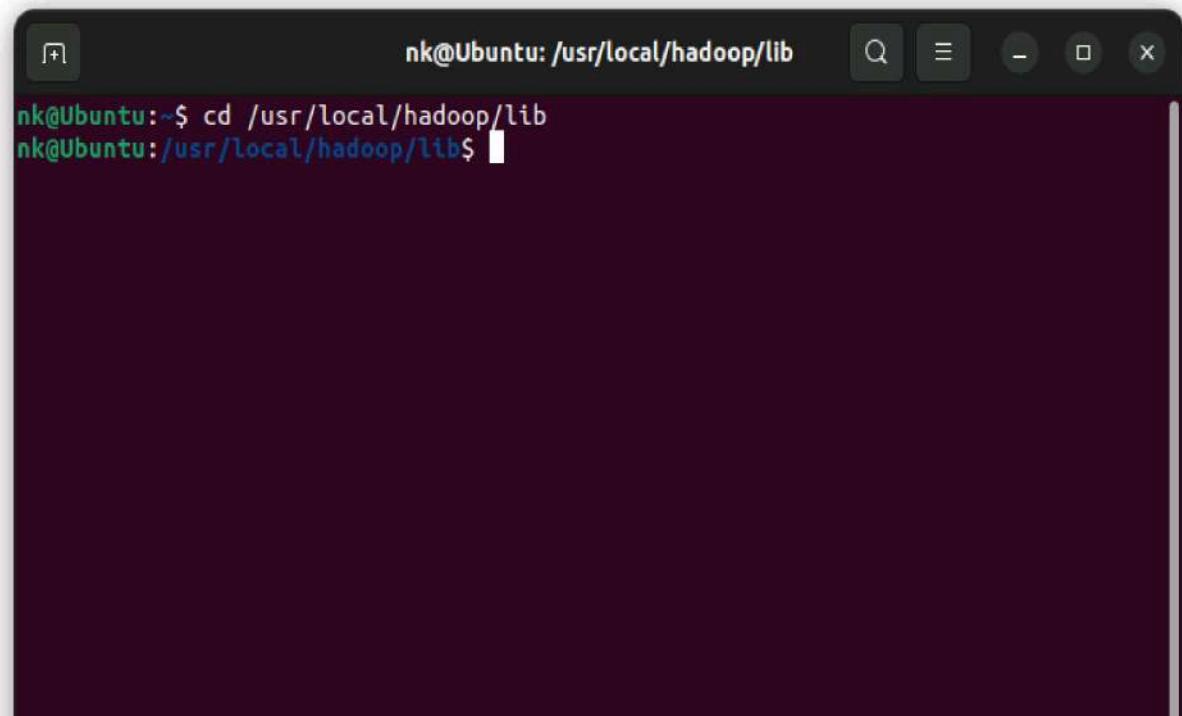
# Supplemental options for privileged registry DNS  
# By default, Hadoop uses jsvc which needs to know to launch a  
# server jvm.  
# export HADOOP_REGISTRYDNS_SECURE_EXTRA_OPTS="-jvm server"  
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64

export HADOOP_CLASSPATH+=" $HADOOP_HOME/lib/*.jar"
[ Read 433 lines ]
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

5.4 Browse to the hadoop lib directory.

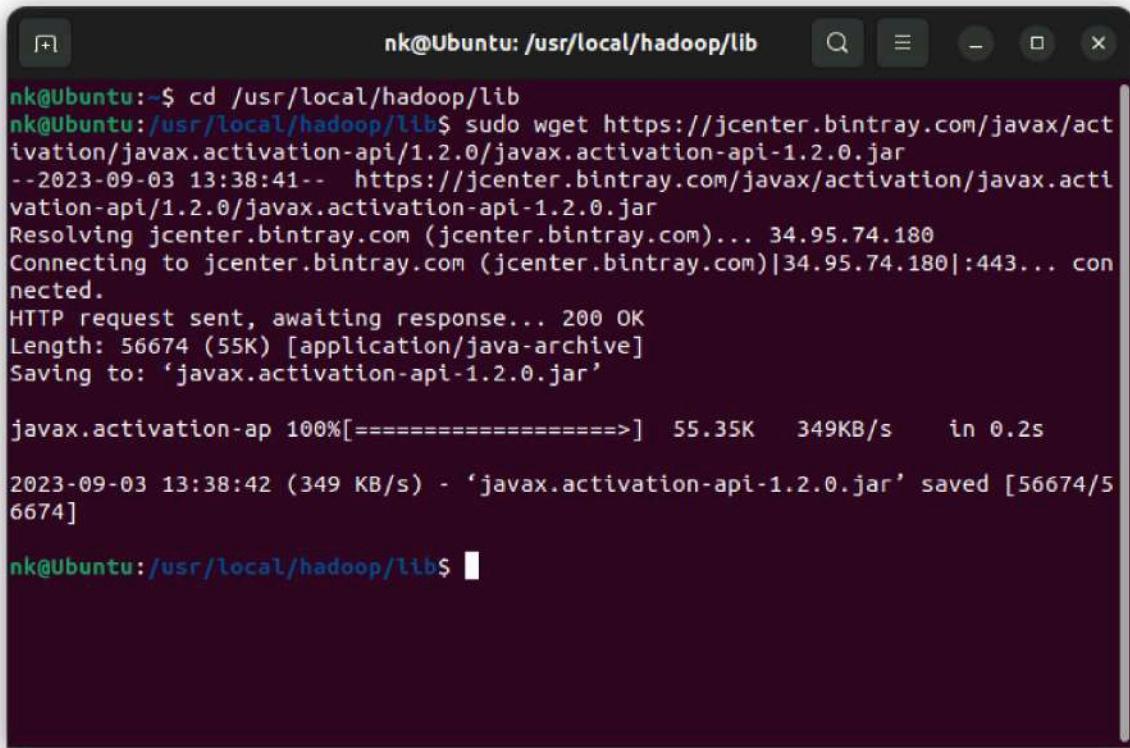
```
$ cd /usr/local/hadoop/lib
```



```
nk@Ubuntu:~$ cd /usr/local/hadoop/lib
nk@Ubuntu:/usr/local/hadoop/lib$ [
```

5.5 Download the Javax activation file.

```
$ sudo wget https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
```



The screenshot shows a terminal window titled "nk@Ubuntu: /usr/local/hadoop/lib". The user runs the command "sudo wget https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar". The output shows the progress of the download, including the URL, connection details, HTTP response, file length, and save location. The download completes successfully at 349 KB/s in 0.2s, saving the file to "/usr/local/hadoop/lib/javax.activation-api-1.2.0.jar".

```
nk@Ubuntu:~$ cd /usr/local/hadoop/lib
nk@Ubuntu:/usr/local/hadoop/lib$ sudo wget https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
--2023-09-03 13:38:41-- https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
Resolving jcenter.bintray.com (jcenter.bintray.com)... 34.95.74.180
Connecting to jcenter.bintray.com (jcenter.bintray.com)|34.95.74.180|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 56674 (55K) [application/java-archive]
Saving to: 'javax.activation-api-1.2.0.jar'

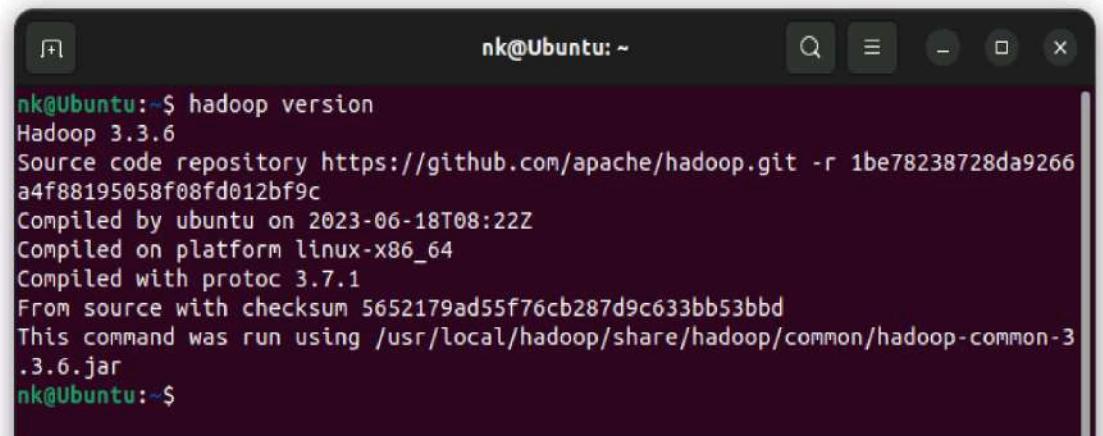
javax.activation-ap 100%[=====] 55.35K 349KB/s in 0.2s

2023-09-03 13:38:42 (349 KB/s) - 'javax.activation-api-1.2.0.jar' saved [56674/56674]

nk@Ubuntu:/usr/local/hadoop/lib$
```

5.6 Verify the Hadoop version.

```
$ hadoop version
```



The screenshot shows a terminal window titled "nk@Ubuntu: ~". The user runs the command "hadoop version". The output displays detailed information about the Hadoop version, including the version number (3.3.6), source code repository, compilation date, platform, and protoc version.

```
nk@Ubuntu:~$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.6.jar
nk@Ubuntu:~$
```

5.7 Edit the core-site.xml configuration file to specify the URL for your NameNode.
Add the following lines. Save and close the file.

```
<configuration>

<property>

    <name>fs.default.name</name>

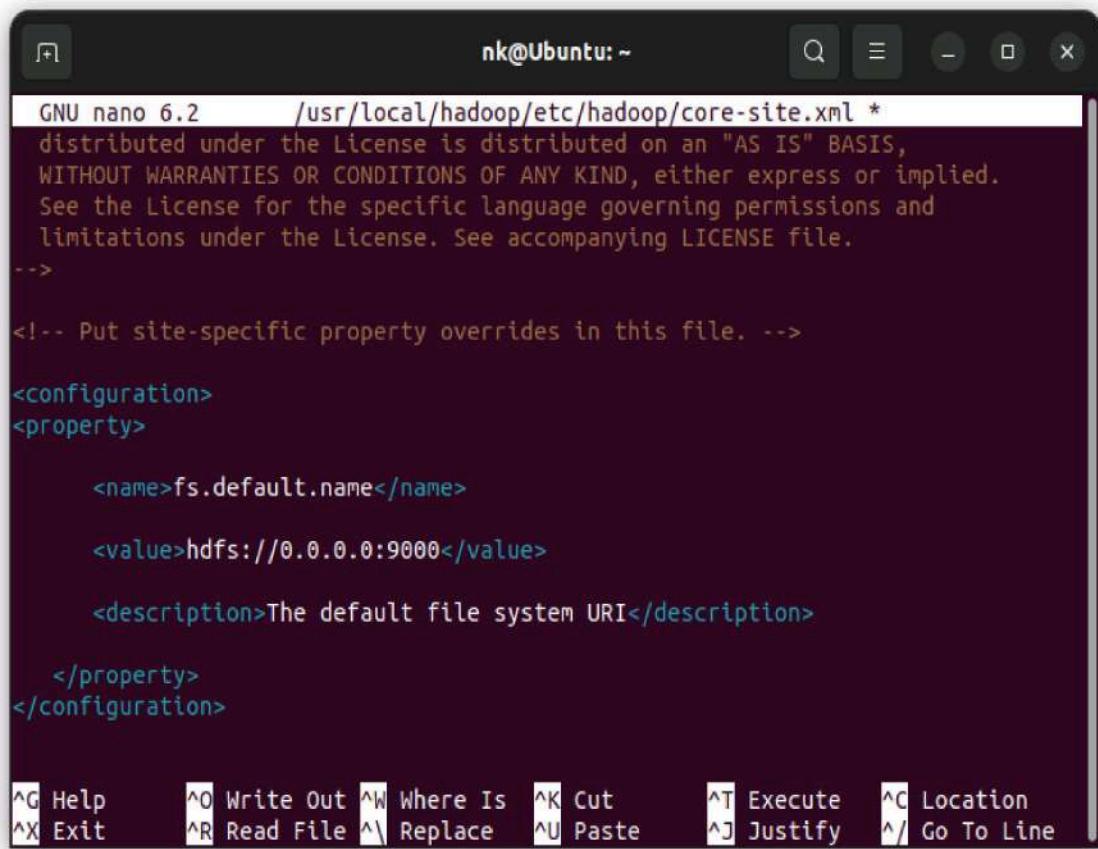
    <value>hdfs://0.0.0.0:9000</value>

    <description>The default file system URI</description>

</property>

</configuration>
```

\$ sudo nano \$HADOOP_HOME/etc/hadoop/core-site.xml



The screenshot shows a terminal window titled "nk@Ubuntu: ~" running the nano text editor. The file being edited is "/usr/local/hadoop/etc/hadoop/core-site.xml". The screen displays the XML configuration code from the previous step. At the bottom of the screen, there is a menu bar with various keyboard shortcut keys for navigating and modifying the file.

```
GNU nano 6.2      /usr/local/hadoop/etc/hadoop/core-site.xml *
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.

-->

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

<configuration>
<property>

    <name>fs.default.name</name>

    <value>hdfs://0.0.0.0:9000</value>

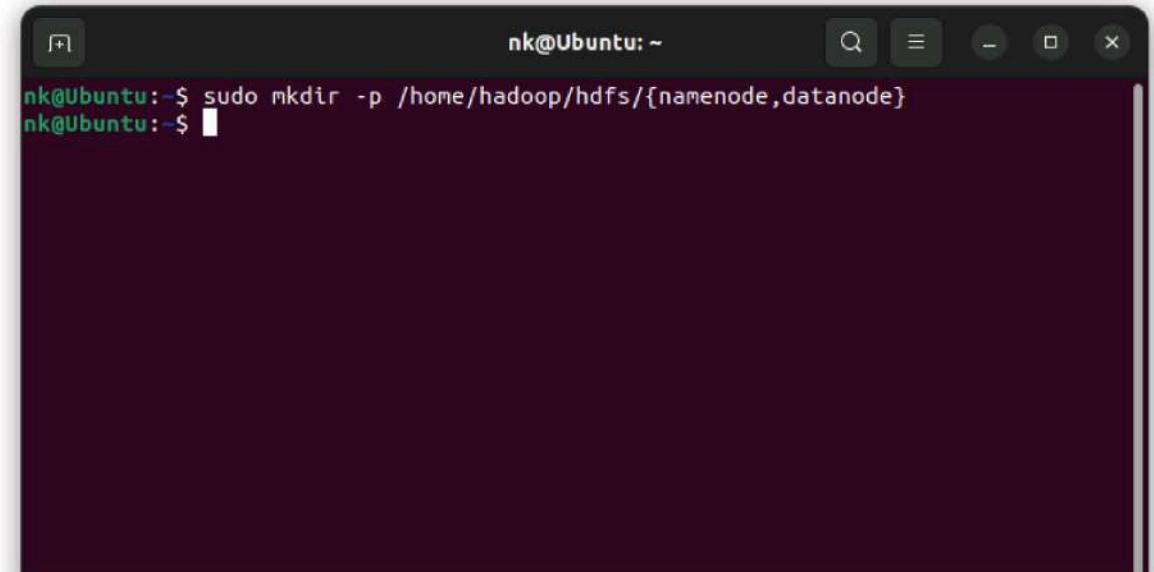
    <description>The default file system URI</description>

</property>
</configuration>

^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute  ^C Location
^X Exit      ^R Read File  ^L Replace   ^U Paste    ^J Justify  ^/ Go To Line
```

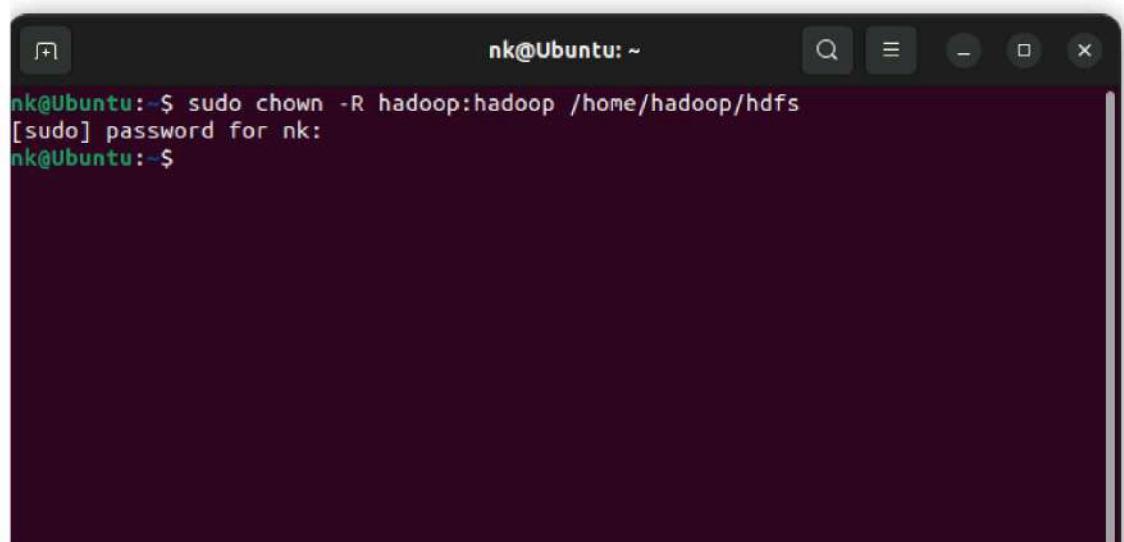
5.8 Create a directory for storing node metadata and change the ownership to hadoop.

a) **\$ sudo mkdir -p /home/hadoop/hdfs/{namenode,datanode}**



```
nk@Ubuntu:~$ sudo mkdir -p /home/hadoop/hdfs/{namenode,datanode}
nk@Ubuntu:~$
```

b) **\$ sudo chown -R hadoop:hadoop /home/hadoop/hdfs**



```
nk@Ubuntu:~$ sudo chown -R hadoop:hadoop /home/hadoop/hdfs
[sudo] password for nk:
nk@Ubuntu:~$
```

5.9 Edit hdfs-site.xml configuration file to define the location for storing node metadata, fs-image file. Add the following lines. Close and save the file.

```
<configuration>

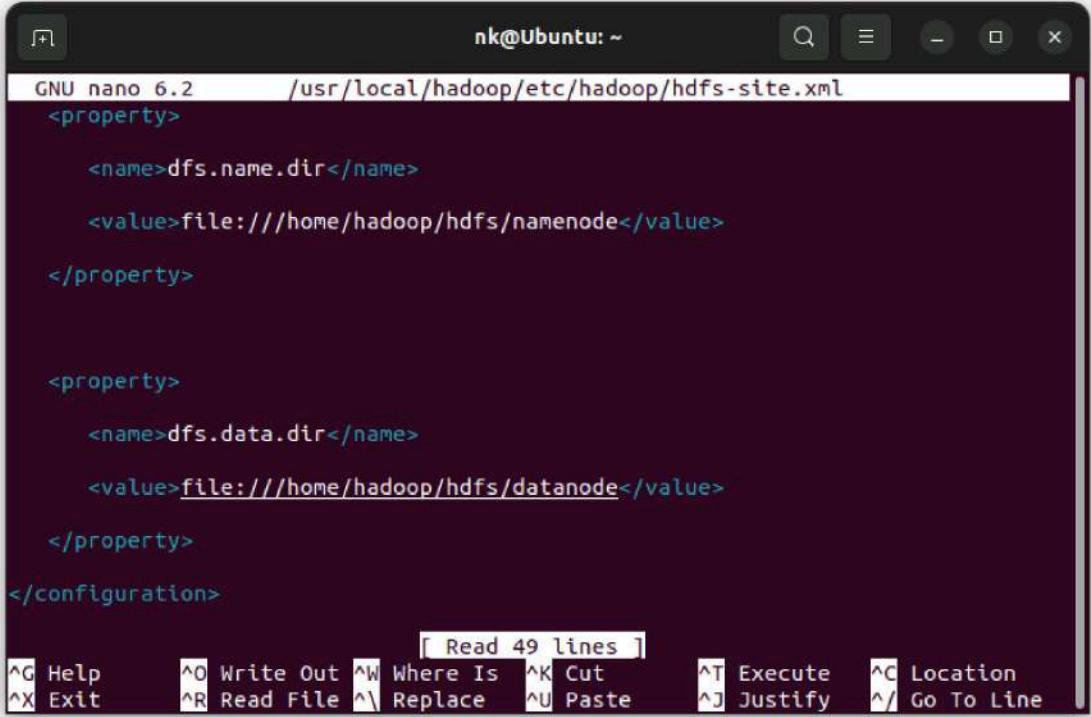
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

<property>
  <name>dfs.name.dir</name>
  <value>file:///home/hadoop/hdfs/namenode</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>file:///home/hadoop/hdfs/datanode</value>
</property>

</configuration>
```

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



The screenshot shows a terminal window titled "nk@Ubuntu: ~". The command "sudo nano \$HADOOP_HOME/etc/hadoop/hdfs-site.xml" is run, and the file content is displayed in the nano editor. The configuration file contains the three properties defined in the previous code block. The nano status bar at the bottom shows various keyboard shortcuts and the message "[Read 49 lines]".

```
GNU nano 6.2      /usr/local/hadoop/etc/hadoop/hdfs-site.xml
<property>
  <name>dfs.name.dir</name>
  <value>file:///home/hadoop/hdfs/namenode</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>file:///home/hadoop/hdfs/datanode</value>
</property>

</configuration>
```

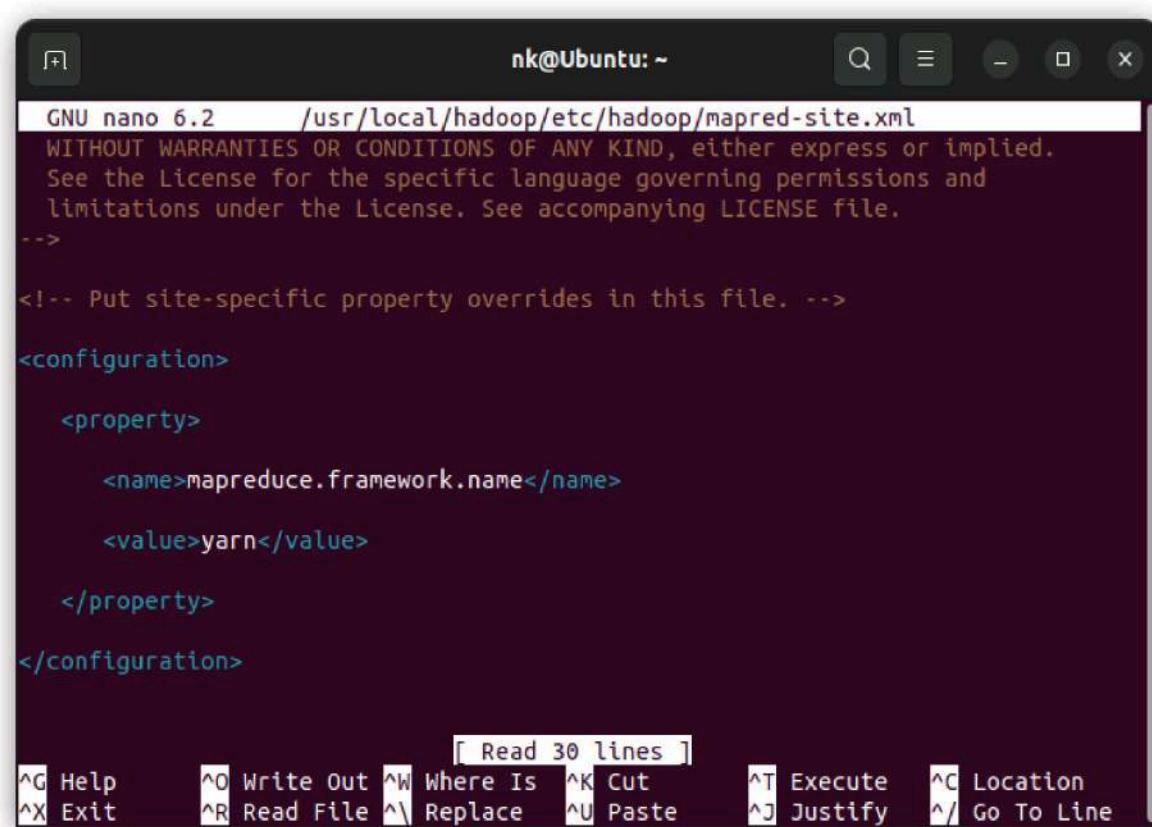
[Read 49 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

5.10 Edit mapred-site.xml configuration file to define MapReduce values. Add the following lines. Save and close the file.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

\$ sudo nano \$HADOOP_HOME/etc/hadoop/mapred-site.xml



The screenshot shows a terminal window titled "nk@Ubuntu: ~" running the nano text editor. The file being edited is "/usr/local/hadoop/etc/hadoop/mapred-site.xml". The terminal displays the XML configuration code. At the bottom, there is a status bar with various keyboard shortcuts for file operations like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, and Go To Line. The status bar also indicates "[Read 30 lines]".

```
GNU nano 6.2      /usr/local/hadoop/etc/hadoop/mapred-site.xml
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>

^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute  ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste    ^J Justify  ^/ Go To Line
```

5.11 Edit the yarn-site.xml configuration file and define YARN-related settings.

Add the following lines. Save and close the file.

```
<configuration>

    <property>

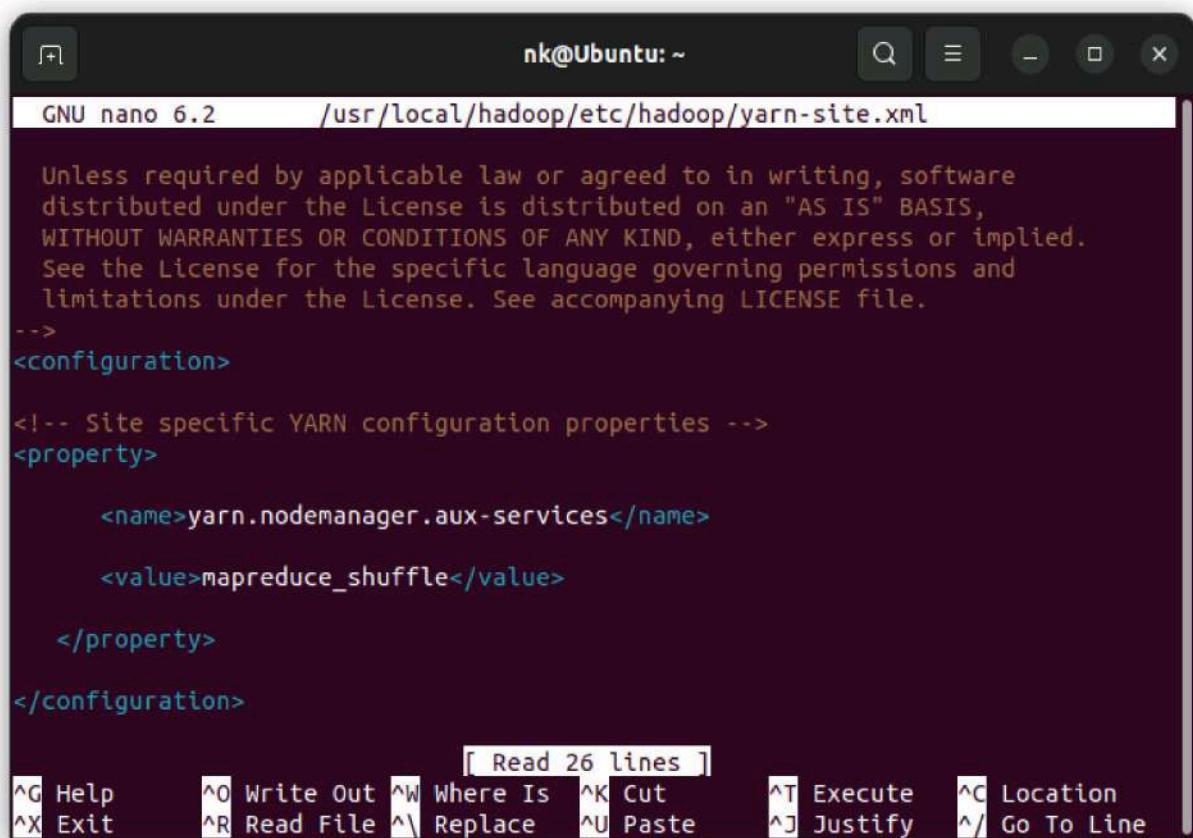
        <name>yarn.nodemanager.aux-services</name>

        <value>mapreduce_shuffle</value>

    </property>

</configuration>
```

\$ sudo nano \$HADOOP_HOME/etc/hadoop/yarn-site.xml



The screenshot shows a terminal window titled "nk@Ubuntu: ~". The command "sudo nano \$HADOOP_HOME/etc/hadoop/yarn-site.xml" is run, opening the file in the nano text editor. The file contains the configuration snippet from the previous step. The terminal window has a dark theme with light-colored text. The bottom of the window shows nano's command-line interface with various keyboard shortcuts for file operations like Help (^G), Exit (^X), Write Out (^O), Read File (^R), Where Is (^W), Replace (^R), Cut (^K), Paste (^U), Execute (^T), Justify (^J), Location (^C), and Go To Line (^L). A status bar at the bottom indicates "[Read 26 lines]".

```
GNU nano 6.2      /usr/local/hadoop/etc/hadoop/yarn-site.xml

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

-->
<configuration>

<!-- Site specific YARN configuration properties -->
<property>

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

    <value>mapreduce_shuffle</value>

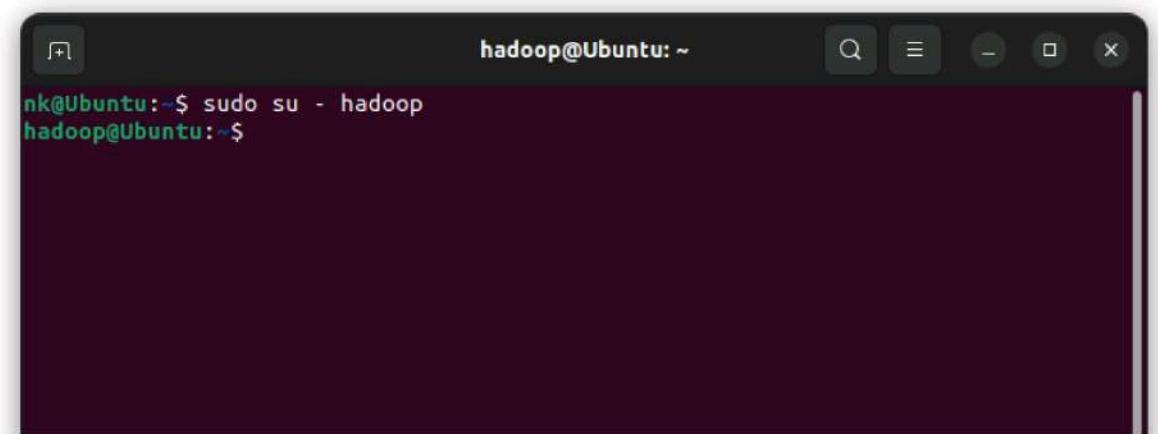
</property>

</configuration>

[ Read 26 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste     ^J Justify   ^/ Go To Line
```

5.12 Log in with hadoop user.

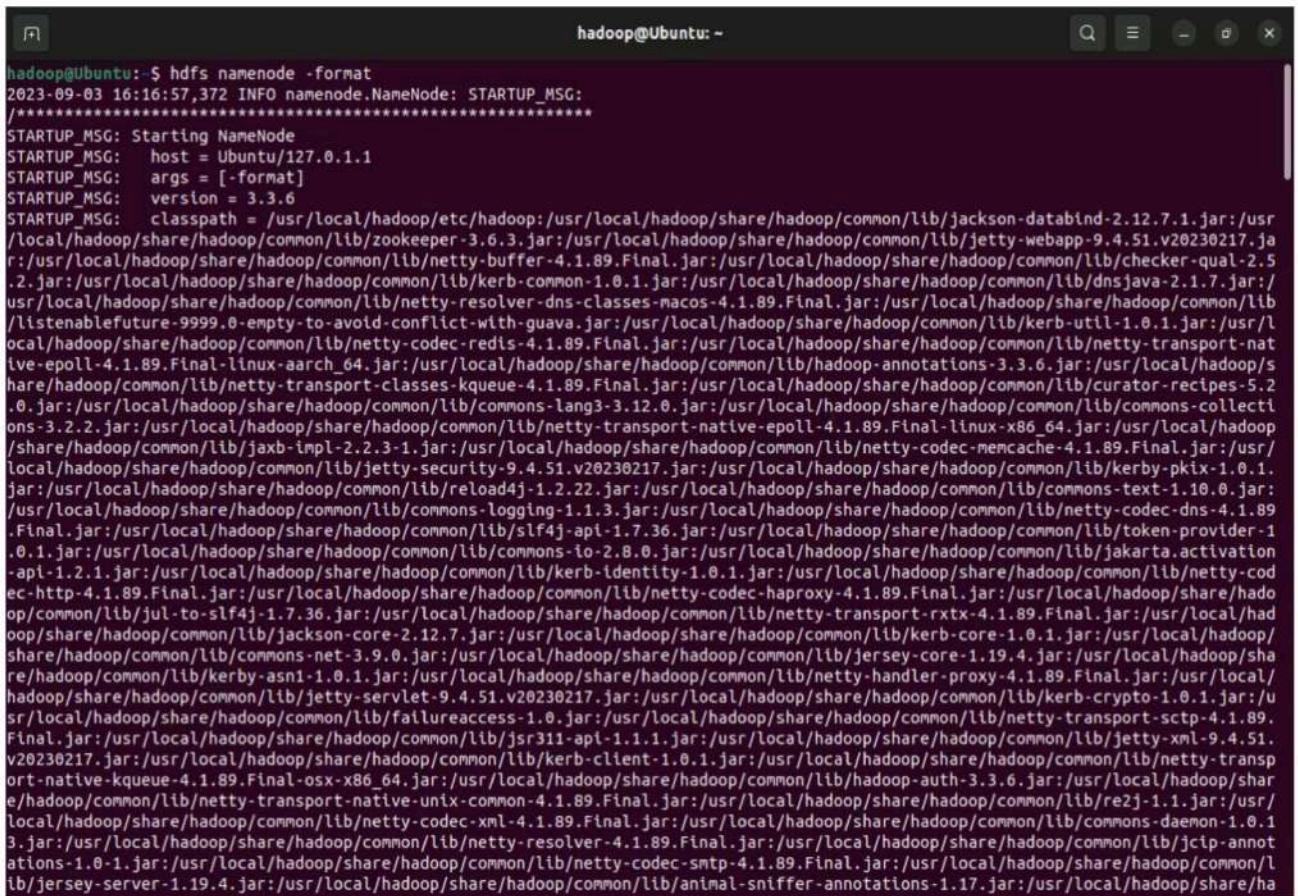
\$ sudo su - hadoop



```
nk@Ubuntu:~$ sudo su - hadoop
hadoop@Ubuntu:~$
```

5.13 Validate the Hadoop configuration and format the HDFS NameNode.

\$ hdfs namenode –format

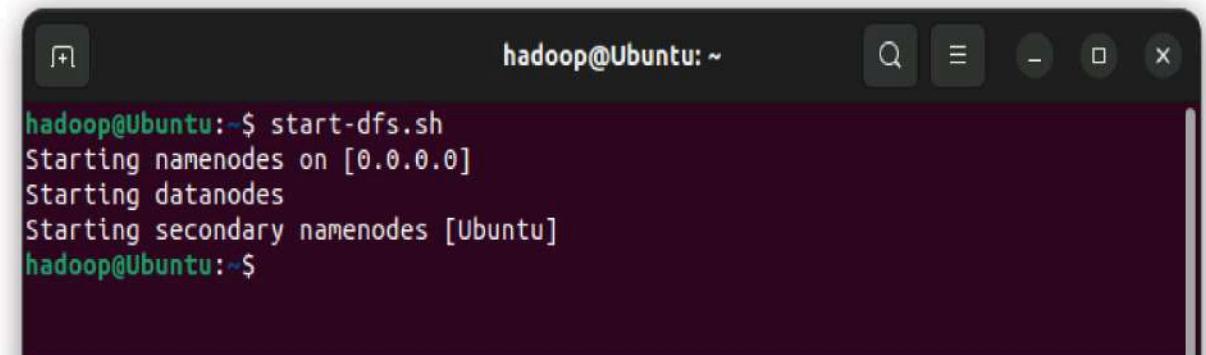


```
hadoop@Ubuntu:~$ hdfs namenode -format
2023-09-03 16:16:57,372 INFO namenode.NameNode: STARTUP_MSG:
*****STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = Ubuntu/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.3.6
STARTUP_MSG: classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/jackson-databind-2.12.7.1.jar:/usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.6.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-webapp-9.4.51.v20230217.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-buffer-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/checker-qual-2.5.2.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-common-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/dnsjava-2.1.7.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-resolver-dns-classes-macos-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-util-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-redis-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-native-epoll-4.1.89.Final-linux-aarch_64.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-annotations-3.3.6.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-recipes-5.2.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-lang3-3.12.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-collections-3.2.2.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-native-epoll-4.1.89.Final-linux-x86_64.jar:/usr/local/hadoop/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-memcache-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-security-9.4.51.v20230217.jar:/usr/local/hadoop/share/hadoop/common/lib/kerby-pkix-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/reload4j-1.2.22.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-text-1.10.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-logging-1.1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-dns-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-api-1.7.36.jar:/usr/local/hadoop/share/hadoop/common/lib/token-provider-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-net-3.9.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-core-1.19.4.jar:/usr/local/hadoop/share/hadoop/common/lib/jakarta.activation-api-1.2.1.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-identity-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-haproxy-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jul-to-slf4j-1.7.36.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-rxtx-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-core-2.12.7.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-core-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-net-3.9.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-core-1.19.4.jar:/usr/local/hadoop/share/hadoop/common/lib/kerby-asn1-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-handler-proxy-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-servlet-9.4.51.v20230217.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-crypto-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/failureaccess-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-sctp-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jsr311-api-1.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-xml-9.4.51.v20230217.jar:/usr/local/hadoop/share/hadoop/common/lib/kerb-client-1.0.1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-native-kqueue-4.1.89.Final-osx-x86_64.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-3.3.6.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-transport-native-unix-common-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/re2j-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-xml-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-daemon-1.0.1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-resolver-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jcip-annotations-1.0-1.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-codec-smtp-4.1.89.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-server-1.19.4.jar:/usr/local/hadoop/share/hadoop/common/lib/animal-sniffer-annotations-1.17.jar:/usr/local/hadoop/share/ha
```

6. Start the Apache Hadoop Cluster

6.1 Start the NameNode and DataNode.

\$ start-dfs.sh

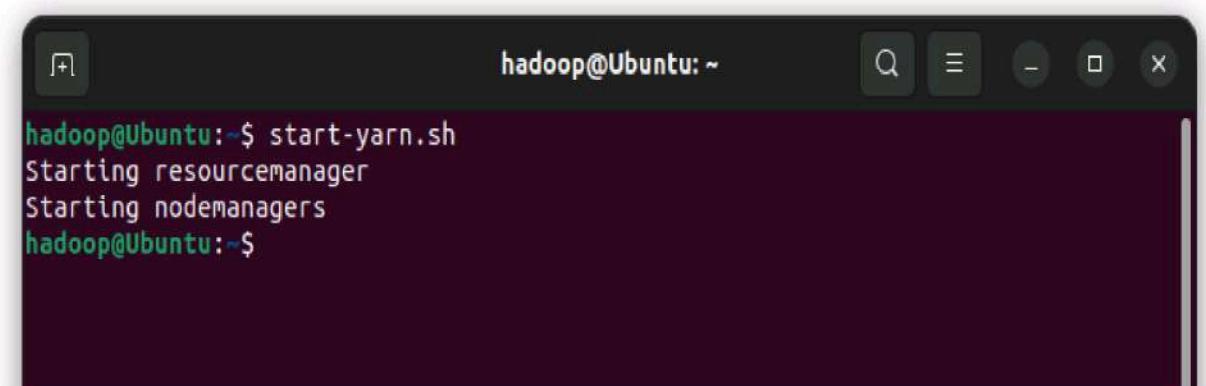


```
hadoop@Ubuntu:~$ start-dfs.sh
Starting namenodes on [0.0.0.0]
Starting datanodes
Starting secondary namenodes [Ubuntu]
hadoop@Ubuntu:~$
```

A screenshot of a terminal window titled "hadoop@Ubuntu: ~". The window shows the command "start-dfs.sh" being run, followed by the output which includes "Starting namenodes on [0.0.0.0]", "Starting datanodes", and "Starting secondary namenodes [Ubuntu]". The terminal has a dark background with light-colored text and standard window controls at the top.

6.2 Start the YARN resource and node managers.

\$ start-yarn.sh

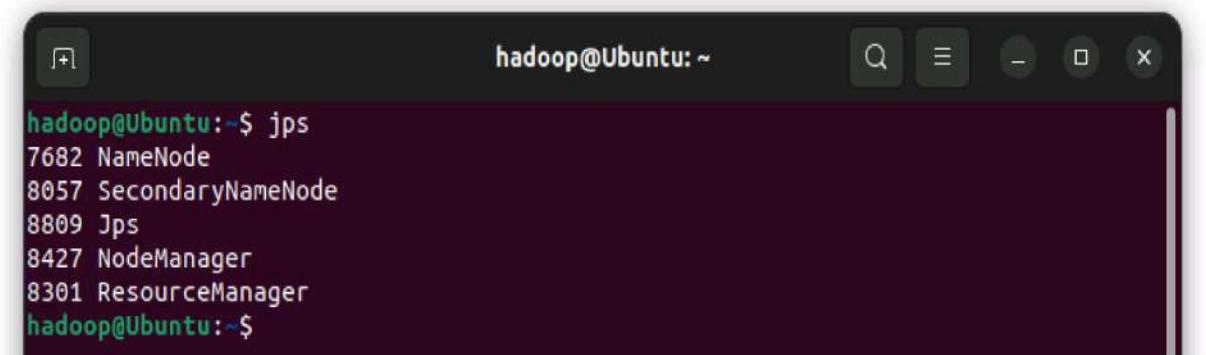


```
hadoop@Ubuntu:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@Ubuntu:~$
```

A screenshot of a terminal window titled "hadoop@Ubuntu: ~". The window shows the command "start-yarn.sh" being run, followed by the output which includes "Starting resourcemanager" and "Starting nodemanagers". The terminal has a dark background with light-colored text and standard window controls at the top.

Verify all the running components.

\$ jps



```
hadoop@Ubuntu:~$ jps
7682 NameNode
8057 SecondaryNameNode
8809 Jps
8427 NodeManager
8301 ResourceManager
hadoop@Ubuntu:~$
```

A screenshot of a terminal window titled "hadoop@Ubuntu: ~". The window shows the command "jps" being run, followed by the output which lists several Java processes: "NameNode" (pid 7682), "SecondaryNameNode" (pid 8057), "Jps" (pid 8809), "NodeManager" (pid 8427), and "ResourceManager" (pid 8301). The terminal has a dark background with light-colored text and standard window controls at the top.

7. Access Apache Hadoop Web Interface

You can access the Hadoop NameNode on your browser via <http://server-IP:9870>
For example:

http://127.0.0.2:9870

The screenshot shows a web browser window titled "Namenode information". The address bar displays "localhost:9870/dfshealth.html#tab-overview". The navigation bar includes links for "Hadoop", "Overview", "Datanodes", "Datanode Volume Failures", "Snapshot", "Startup Progress", and "Utilities". The main content area is titled "Overview '0.0.0.0:9000' (✓active)". It contains a table with the following data:

Started:	Sun Sep 03 16:34:05 +0530 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 13:52:00 +0530 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-2f70bb2d-fe5f-453f-b322-e6ceae53afb
Block Pool ID:	BP-19373805-127.0.1.1-1693738993729

Below this, the "Summary" section provides system status information:

- Security is off.
- Safemode is off.
- 1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).
- Heap Memory used 91.3 MB of 299 MB Heap Memory. Max Heap Memory is 2.71 GB.
- Non Heap Memory used 48.44 MB of 51.81 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

A progress bar at the bottom indicates "Configured Capacity" at 0 B.

RESULT

Thus the program has been executed successfully.

Ex No: 2

Hadoop Implementation of File Management Task

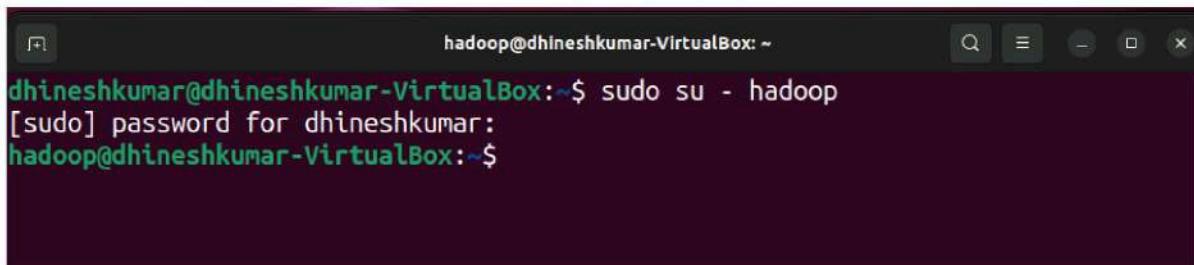
Date:

COMMANDS

1. Starting Hadoop Environment

- After successful installation of Hadoop, log in into the created account from terminal.

\$ sudo su – hadoop



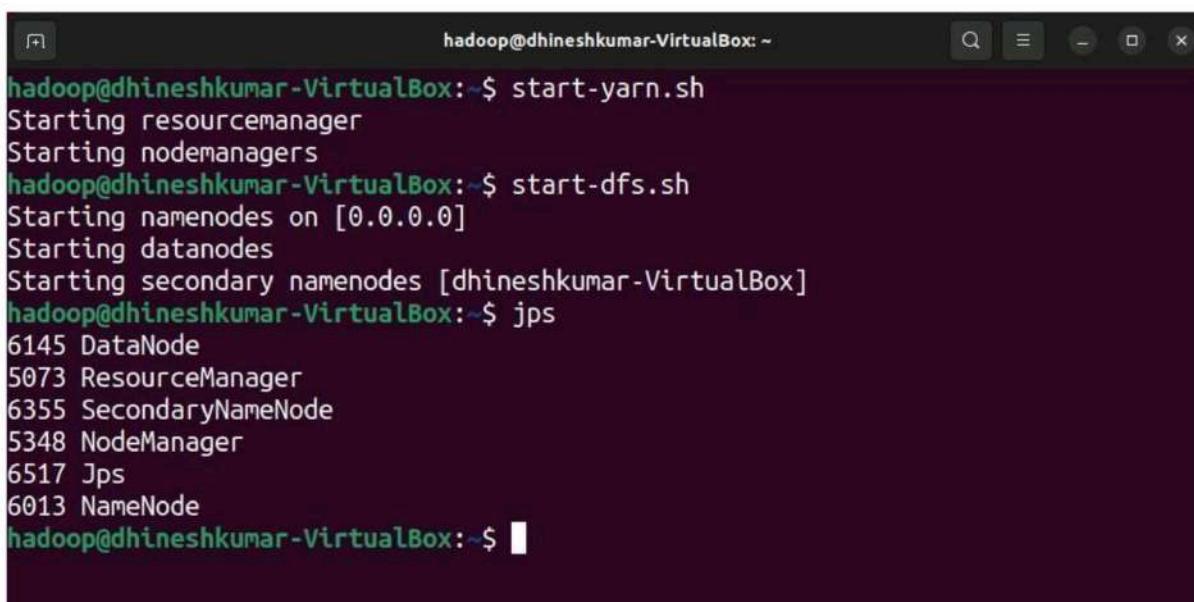
```
hadoop@dhineshkumar-VirtualBox:~$ sudo su - hadoop
[sudo] password for dhineshkumar:
hadoop@dhineshkumar-VirtualBox:~$
```

- Start yarn and dfs by using the following commands.

\$ start-yarn.sh

\$ start-dfs.sh

Check all the running components by using **\$ jps** command.



```
hadoop@dhineshkumar-VirtualBox:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@dhineshkumar-VirtualBox:~$ start-dfs.sh
Starting namenodes on [0.0.0.0]
Starting datanodes
Starting secondary namenodes [dhineshkumar-VirtualBox]
hadoop@dhineshkumar-VirtualBox:~$ jps
6145 DataNode
5073 ResourceManager
6355 SecondaryNameNode
5348 NodeManager
6517 Jps
6013 NameNode
hadoop@dhineshkumar-VirtualBox:~$
```

- Find the IP address of the local machine to open the Hadoop environment in a browser.

\$ ifconfig

```
hadoop@dhineshkumar-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fe80::2a16:ddb:5361:4f0c prefixlen 64 scopeid 0x20<link>
          ether 08:00:27:e4:cd:91 txqueuelen 1000 (Ethernet)
            RX packets 485239 bytes 727343459 (727.3 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 26501 bytes 1676557 (1.6 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Local Loopback)
            RX packets 1763 bytes 268343 (268.3 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1763 bytes 268343 (268.3 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

127.0.0.1 is the IP address of the local machine. View Hadoop in browser by going to the following link. <http://127.0.0.1:9870> (Here replace the IP with your system's IP)

Overview '0.0.0.0:9000' (active)

Started:	Mon Sep 18 21:41:38 +0530 2023
Version:	3.3.1, ra3b9c37a397ad4188041dd80621bdeefc46885f2
Compiled:	Tue Jun 15 10:43:00 +0530 2021 by ubuntu from (HEAD detached at release-3.3.1-RC3)
Cluster ID:	CID-695c8052-50b5-4bcb-8f56-cad6a9977e99
Block Pool ID:	BP-1509660294-127.0.1.1-1694539263356

Summary

Security is off.
Safemode is off.

1 files and directories, 0 blocks (0 replicated blocks; 0 erasure coded block groups) = 1 total filesystem object(s).

Heap Memory used 40.06 MB of 93 MB Heap Memory. Max Heap Memory is 484 MB.

Non Heap Memory used 52.77 MB of 56.15 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	33.66 GB
----------------------	----------

- In the browser go to **Utilities > Browse the file system**

The screenshot shows a web browser window with the URL `127.0.0.1:9870/explorer.html#/`. The page title is "Browse Directory". The main content area displays a table header with columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. Below the header, a message says "No data available in table". At the bottom of the table area, it says "Showing 0 to 0 of 0 entries". There are navigation buttons for "Previous" and "Next". A footer at the bottom left says "Hadoop, 2021."

The root directory is empty initially. Any directory or file can be created from the terminal.

2. Creating a new Directory named Dhinesh in hadoop.

\$ hadoop fs -mkdir /Dhinesh

The output on the browser would look like,

The screenshot shows the same browser interface as before, but now it lists a single entry: "Dhinesh". The table row for "Dhinesh" has the following details: Permission (drwxr-xr-x), Owner (hadoop), Group (supergroup), Size (0 B), Last Modified (Sep 18 22:42), Replication (0), Block Size (0 B), and Name (Dhinesh). The footer at the bottom left still says "Hadoop, 2021."

```

hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x - hadoop supergroup          0 2023-09-18 22:42 /Dhinesh
hadoop@dhineshkumar-VirtualBox:~$
```

Can also see the created directory from the terminal using **\$ hadoop fs -ls /** command

3. Creating a new file in home of hadoop user account using VIM editor

```
hadoop@dhineshkumar-VirtualBox: ~
```

```
GNU nano 6.2          sample.txt
```

```
Hello,  
This is Big Data lab.
```

```
[ Read 3 lines ]
```

```
^G Help      ^O Write Out  ^W Where Is   ^K Cut       ^T Execute    ^C Location  
^X Exit      ^R Read File   ^\ Replace    ^U Paste     ^J Justify    ^/ Go To Line
```

```
$ nano sample.txt
```

Save the file by,

1. **CTRL + X**
2. **Choose Y (Option Yes)**
3. **Press Enter key**

```
hadoop@dhineshkumar-VirtualBox: ~
```

```
hadoop@dhineshkumar-VirtualBox: ~$ ls
```

```
Desktop  Downloads      hdfs  Pictures  sample.txt  Templates  
Documents  hadoop-3.3.1.tar.gz  Music  Public    snap        Videos
```

```
hadoop@dhineshkumar-VirtualBox: ~$
```

Check the availability of created file using \$ ls command.

4. Moving file from local machine to hadoop user

```
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -copyFromLocal /home/hadoop/sample.txt /Dhinesh
hadoop@dhineshkumar-VirtualBox:~$
```

\$ hadoop fs -copyFromLocal /home/hadoop/sample.txt /Dhinesh

127.0.0.1:9870/explorer.html#/Dhinesh

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/Dhinesh	Go!	File	Up	Download	Open		
Show 25 entries	Search:						
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	31 B	Sep 18 22:59	1	128 MB	sample.txt

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2021.

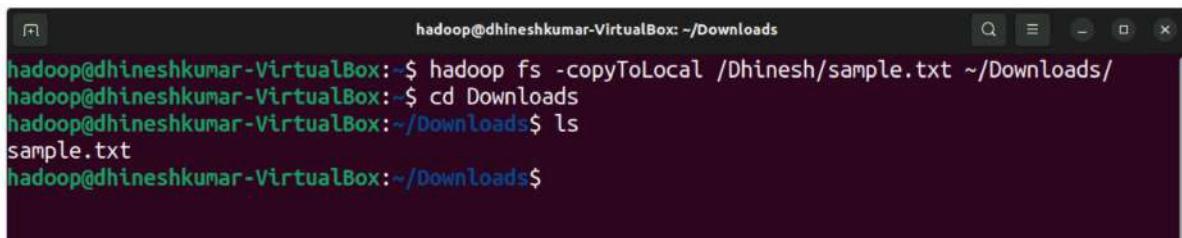
Check the moved file from browser.

```
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -ls /Dhinesh
Found 1 items
-rw-r--r-- 1 hadoop supergroup 31 2023-09-18 22:59 /Dhinesh/sample.txt
hadoop@dhineshkumar-VirtualBox:~$
```

Check the file creation using **\$ hadoop fs -ls /Dhinesh**

5. Retrieving file from hadoop

```
$ hadoop fs -copyToLocal /Dhinesh/sample.txt ~/Downloads/
```

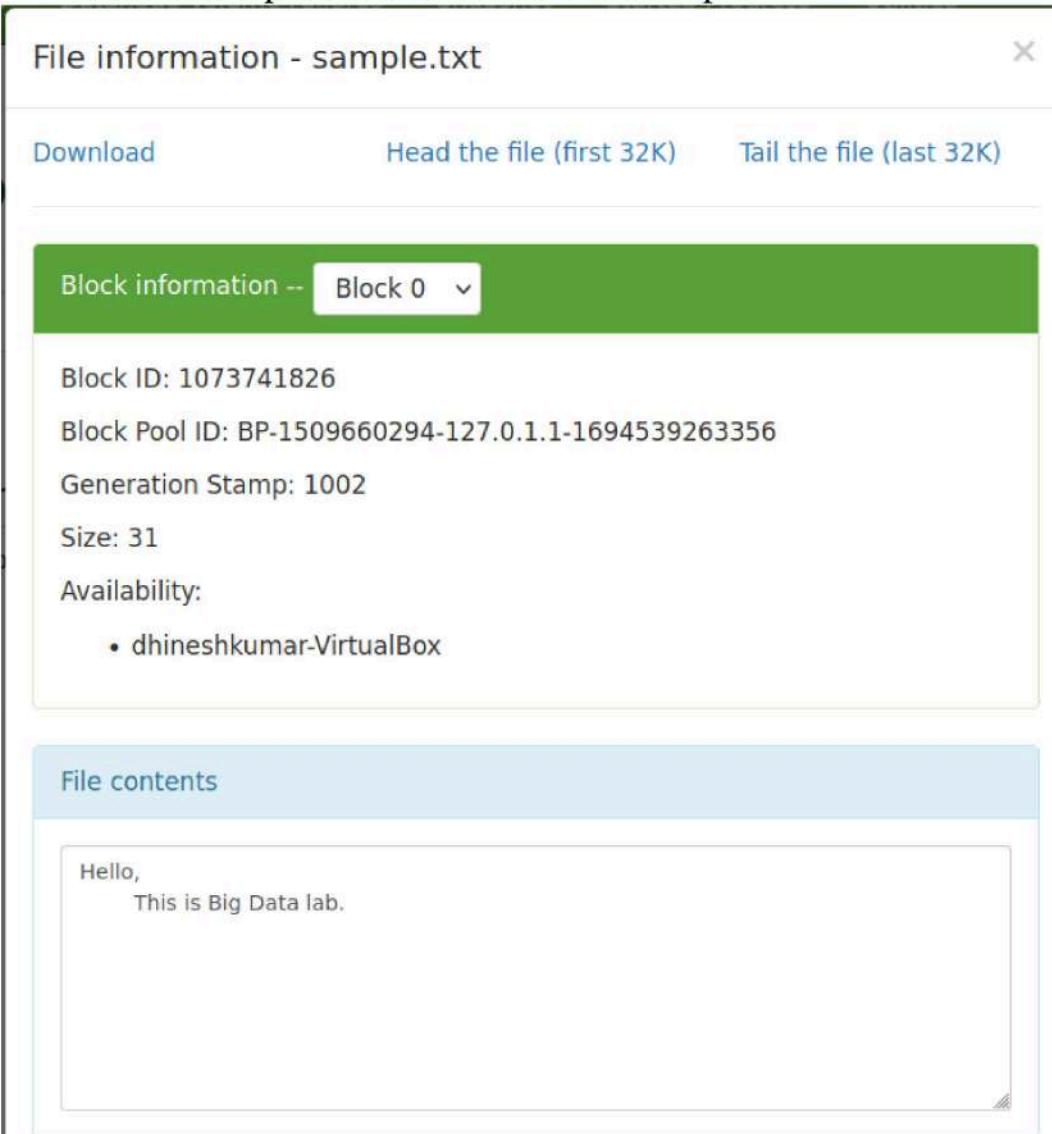


```
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -copyToLocal /Dhinesh/sample.txt ~/Downloads/
hadoop@dhineshkumar-VirtualBox:~$ cd Downloads
hadoop@dhineshkumar-VirtualBox:~/Downloads$ ls
sample.txt
hadoop@dhineshkumar-VirtualBox:~/Downloads$
```

Check the file transfer by going to Downloads folder and listing the available items using `$ ls`

6. File details

Click on sample.txt in the browser from step 4.



File information - sample.txt

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073741826
Block Pool ID: BP-1509660294-127.0.1.1-1694539263356
Generation Stamp: 1002
Size: 31
Availability:

- dhineshkumar-VirtualBox

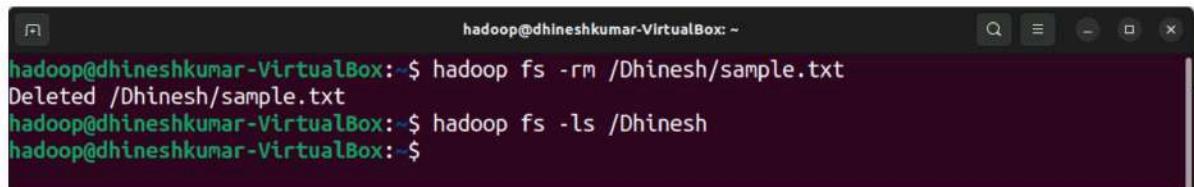
File contents

Hello,
This is Big Data lab.

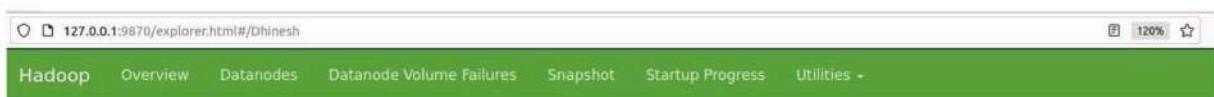
7. Deleting a file

\$ hadoop fs -rm /Dhinesh/sample.txt (to delete the file)

\$ hadoop fs -ls /Dhinesh (to check contents of Dhinesh directory)



```
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -rm /Dhinesh/sample.txt
Deleted /Dhinesh/sample.txt
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -ls /Dhinesh
hadoop@dhineshkumar-VirtualBox:~$
```



Browse Directory

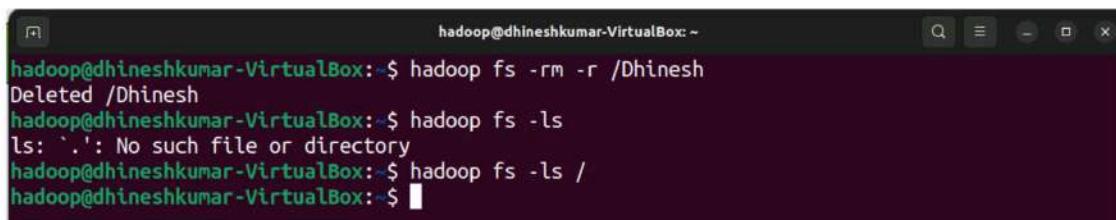
Browse Directory								
/Dhinesh								
Show 25 entries								
Permission								
□	Owner	Group	Size	Last Modified	Replication	Block Size	Name	IT
No data available in table								
Showing 0 to 0 of 0 entries								
Previous Next								

The browser also shows that the Dhinesh directory is empty.

8. Deleting the directory

\$ hadoop fs -rm -r /Dhinesh (to delete the directory)

\$ hadoop fs -ls / (to check the contents of root directory)



```
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -rm -r /Dhinesh
Deleted /Dhinesh
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -ls
ls: `.': No such file or directory
hadoop@dhineshkumar-VirtualBox:~$ hadoop fs -ls /
hadoop@dhineshkumar-VirtualBox:~$
```

In browser the following error message is shown.

The screenshot shows a web browser window with the URL `127.0.0.1:9870/explorer.html#/Dhinesh`. The page title is "Browse Directory". A red error message box contains the text: "Path does not exist on HDFS or WebHDFS is disabled. Please check your path or enable WebHDFS". Below the message is a search bar with the path `/Dhinesh` and a "Go!" button. To the right of the search bar are four small icons: a folder, a file, a list, and a refresh symbol.

Hadoop, 2021.

9. Stopping the hadoop resources.

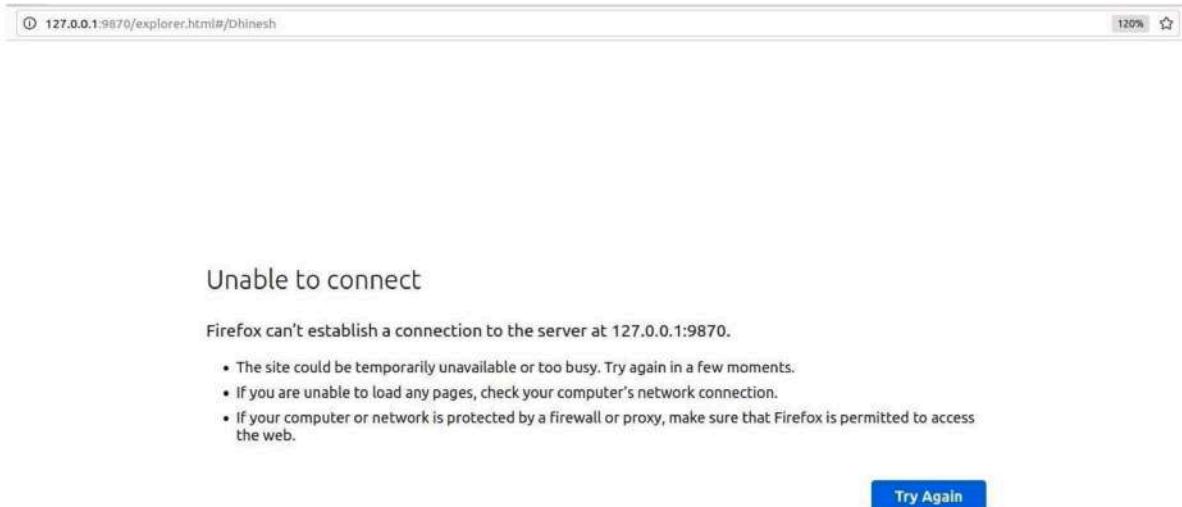
\$ stop-yarn.sh

\$ stop-dfs.sh

\$ jps (Check the running components)

```
hadoop@dhineshkumar-VirtualBox:~$ stop-yarn.sh
Stopping nodemanagers
Stopping resourcemanager
hadoop@dhineshkumar-VirtualBox:~$ stop-dfs.sh
Stopping namenodes on [0.0.0.0]
Stopping datanodes
Stopping secondary namenodes [dhineshkumar-VirtualBox]
hadoop@dhineshkumar-VirtualBox:~$ jps
10512 Jps
hadoop@dhineshkumar-VirtualBox:~$
```

The browser's interactive hadoop dashboard will be closed.



RESULT

Thus the program has been executed successfully.

Ex No 3 Run a basic word count program to understand MapReduce paradigms

Date:

PROGRAM

WordCounter.java

```
package count;

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

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

WordCountMapper.java

```
package count;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class WordCountMapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text
value,OutputCollector<Text,IntWritable> output,
                    Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        /*
        String line = value.toString().toLowerCase(); // Convert to lowercase
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();

```

```
// Remove punctuation symbols (only keep letters and numbers)
token = token.replaceAll("[^a-zA-Z0-9]", "");

if (!token.isEmpty()) { // Check if token is !empty after removing punctuation
    word.set(token);
    output.collect(word, one);
}

/*
}
}
}
```

WordCountReducer.java

```
package count;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class WordCountReducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output,
                    Reporter reporter) throws IOException {
        int sum=0;
        while (values.hasNext()) {
            sum+=values.next().get();
        }
    }
}
```

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

Close the eclipse and execute the below hadoop commands in the terminal.

COMMANDS

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

```
$ jps
```

```
$ hadoop fs -mkdir /WordCount
```

```
$ hadoop fs -mkdir /WordCount/Input
```

```
$ hadoop fs -put '/home/hadoop/eclipse-workspace/WordCount /input.txt'  
/WordCount /Input
```

```
$ hadoop jar '/home/hadoop/eclipse-workspace/WordCount /count.jar' /  
WordCount /Input / WordCount /Output
```

```
$ hadoop dfs -cat /MatrixMultiplicationTutorial/Output/*
```

(If any other issues are encountered, then the installation is not properly done.)

OUTPUT



```
hadoop@kirthika-VirtualBox: ~
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=144
File Output Format Counters
Bytes Written=183
hadoop@kirthika-VirtualBox:~$ hadoop dfs -cat /WordCount/Output/*
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Big      1
Get      1
Hello    1
Welcome  1
a        1
an       1
big      1
concepts 1
data     2
entertainment 1
experience 1
gives    1
great    1
in       1
is       1
lab      1
learning 1
life     1
new      2
of       1
subject  1
the      1
things   1
to       1
hadoop@kirthika-VirtualBox:~$
```

RESULT

Thus the program has been executed successfully.

Ex No :4

Data Preprocessing and Table Creation

Date:

Step 1. Data Preprocessing

1.1 Create directories in hdfs to store the csv files

```
hadoop fs -mkdir /user/cloudera/covidAnalysis
```

1.2 Copy the files from local to hdfs

```
hadoop fs -copyFromLocal covid/covid19.csv /user/cloudera/covidAnalysis
```

```
hadoop fs -copyFromLocal covid/statewiseTesting.csv /user/cloudera/covidAnalysis
```

```
[root@quickstart Desktop]# hadoop fs -ls /user/cloudera/covidAnalysis
Found 2 items
-rw-r--r-- 1 root cloudera 85487 2023-07-16 01:18 /user/cloudera/covidAnalysis/covid19.csv
-rw-r--r-- 1 root cloudera 77259 2023-07-16 01:18 /user/cloudera/covidAnalysis/statewiseTesting.csv
```

1.3 Visualize the content of the files in hdfs

```
[root@quickstart Desktop]# hadoop fs -cat /user/cloudera/covidAnalysis/covid19.csv |head
530,1/4/2020,Andhra Pradesh,1,0,83
531,1/4/2020,Andaman and Nicobar Islands,0,0,10
532,1/4/2020,Assam,0,0,1
533,1/4/2020,Bihar,0,1,23
534,1/4/2020,Chandigarh,0,0,16
535,1/4/2020,Chhattisgarh,2,0,9
536,1/4/2020,Delhi,6,2,152
537,1/4/2020,Goa,0,0,5
538,1/4/2020,Gujarat,5,6,82
539,1/4/2020,Haryana,21,0,43
```

```
[root@quickstart Desktop]# hadoop fs -cat /user/cloudera/covidAnalysis/statewiseTesting.csv |head
1,4/1/2020,Andaman and Nicobar Islands,1403,1210,12
2,4/2/2020,Andaman and Nicobar Islands,2879,,27
3,4/27/2020,Andaman and Nicobar Islands,2848,,33
4,5/1/2020,Andaman and Nicobar Islands,3754,,33
5,5/16/2020,Andaman and Nicobar Islands,6677,,33
6,5/19/2020,Andaman and Nicobar Islands,6965,,33
7,5/20/2020,Andaman and Nicobar Islands,7082,,33
8,5/21/2020,Andaman and Nicobar Islands,7167,,33
9,5/22/2020,Andaman and Nicobar Islands,7263,,33
10,5/23/2020,Andaman and Nicobar Islands,7327,,33
```

1.4 Creation of Tables in Mysql

```
mysql> create table IF NOT EXISTS StateTesting(
->     seq int not null primary key,
->     date varchar(50),
->     state varchar(50) not null,
->     totalSamples int ,
->     negative int,
->     positive int
-> );
Query OK, 0 rows affected (0.12 sec)
```

```
mysql> create table IF NOT EXISTS covidIndia(
->     sno int not null primary key,
->     date varchar(50),
->     state varchar(50) not null,
->     cured int ,
->     deaths int,
->     confirmed int
-> );
Query OK, 0 rows affected (0.02 sec)
```

1.5 Sqoop Export of data from hdfs to Mysql

```
sqoop export \
--connect jdbc:mysql://quickstart.cloudera:3306/test_db \
--username root \
--password cloudera \
--table StateTesting \
--lines-terminated-by ',' \
--export-dir /user/cloudera/covidAnalysis/statewiseTesting.csv

      bytes written=0
23/07/16 01:23:17 INFO mapreduce.ExportJobBase: Transferred 92.1865 KB in 52.1837 seconds (1.7666 KB/sec)
23/07/16 01:23:17 INFO mapreduce.ExportJobBase: Exported 1922 records.
```

```
sqoop export \
--connect jdbc:mysql://quickstart.cloudera:3306/test_db \
--username root \
--password cloudera \
--table covidIndia \
--lines-terminated-by ',' \
--export-dir /user/cloudera/covidAnalysis/covid19.csv
```

```
23/07/16 01:25:31 INFO mapreduce.ExportJobBase: Transferred 100.1777 KB in 45.3055 seconds (2.2112 KB/sec)
23/07/16 01:25:31 INFO mapreduce.ExportJobBase: Exported 2390 records.
```

1.6 Verify in Mysql

```
mysql> select count(*) from covidIndia;
+-----+
| count(*) |
+-----+
|    2390 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from StateTesting;
+-----+
| count(*) |
+-----+
|    1922 |
+-----+
1 row in set (0.00 sec)
```

1.7 Delete data from hdfs

Step 2- Sqoop Import

Command:

```
[root@quickstart Desktop]# sqoop-import \
> --connect jdbc:mysql://quickstart.cloudera:3306/test_db \
> --username root \
> --password cloudera \
> --table StateTesting \
> --incremental append \
> --check-column seq \
> --last-value 0 \
> --warehouse-dir /user/cloudera/covidAnalysis/sqoopImport
```

Command:

```
[root@quickstart Desktop]# sqoop-import \
> --connect jdbc:mysql://quickstart.cloudera:3306/test_db \
> --username root \
> --password cloudera \
> --table covidIndia \
> --incremental append \
> --check-column sno \
> --last-value 0 \
> --warehouse-dir /user/cloudera/covidAnalysis/sqoopImport
```

Output:

```
[root@quickstart Desktop]# hadoop fs -ls /user/cloudera/covidAnalysis/sqoopImport
Found 2 items
drwxr-xr-x  - root cloudera      0 2023-07-16 06:39 /user/cloudera/covidAnalysis/sqoopImport/StateTesting
drwxr-xr-x  - root cloudera      0 2023-07-16 06:43 /user/cloudera/covidAnalysis/sqoopImport/covidIndia
[root@quickstart Desktop]# hadoop fs -ls /user/cloudera/covidAnalysis/sqoopImport/StateTesting
Found 4 items
-rw-r--r--  1 root supergroup  19443 2023-07-16 06:39 /user/cloudera/covidAnalysis/sqoopImport/StateTesting/part-m-00000
-rw-r--r--  1 root supergroup  19266 2023-07-16 06:39 /user/cloudera/covidAnalysis/sqoopImport/StateTesting/part-m-00001
-rw-r--r--  1 root supergroup  18589 2023-07-16 06:39 /user/cloudera/covidAnalysis/sqoopImport/StateTesting/part-m-00002
-rw-r--r--  1 root supergroup  19815 2023-07-16 06:39 /user/cloudera/covidAnalysis/sqoopImport/StateTesting/part-m-00003
```

Step 3- Create Hive External Tables on top of data in HDFS

3.1 Connect to hive and select/create a database

```
[root@quickstart clouderaj]# beeline -u jdbc:hive2://
scan complete in 2ms
Connecting to jdbc:hive2://
Connected to: Apache Hive (version 1.1.0-cdh5.13.0)
Driver: Hive JDBC (version 1.1.0-cdh5.13.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.1.0-cdh5.13.0 by Apache Hive
0: jdbc:hive2://> show databases;
OK
+-----+
| database_name |
+-----+
| default       |
| mydb          |
| testdb         |
+-----+
3 rows selected (1.234 seconds)
0: jdbc:hive2://> use testdb;
OK
No rows affected (0.095 seconds)
```

3.2 Creation of Hive External Tables on top of data in HDFS

StateTesting table

```
0: jdbc:hive2://> create external table if not exists stateTesting(
. . . . . > seq int,
. . . . . > date string,
. . . . . > state string,
. . . . . > totalSamples int,
. . . . . > negative int,
. . . . . > positive int
. . . . . >
. . . . . > row format delimited
. . . . . > fields terminated by ','
. . . . . > stored as textfile
. . . . . > location '/user/cloudera/covidAnalysis/sqoopImport/StateTesting';|
```

```
0: jdbc:hive2://> select * from stateTesting LIMIT 5;
OK
+-----+-----+-----+-----+-----+-----+
| statetesting.seq | statetesting.date | statetesting.state | statetesting.totalsamples | statetesting.negative | statetesting.positive |
+-----+-----+-----+-----+-----+-----+
| 1 | 4/17/2020 | Andaman and Nicobar Islands | 1403 | 1210 | 12 |
| 2 | 4/24/2020 | Andaman and Nicobar Islands | 2679 | NULL | 27 |
| 3 | 4/27/2020 | Andaman and Nicobar Islands | 2848 | NULL | 33 |
| 4 | 5/1/2020 | Andaman and Nicobar Islands | 3754 | NULL | 33 |
| 5 | 5/16/2020 | Andaman and Nicobar Islands | 6677 | NULL | 33 |
+-----+-----+-----+-----+-----+-----+
```

Similarly, we do for covidIndia Table

Step 4- Create Optimized External tables in Hive

Optimizations Applied-

- File format used- ORC for Quicker and Efficient Reads
- Compression Codec used- Snappy for Fast Compression
- Partitioning on State Column
- Bucketing on Date Column

4.1 Create directories in hdfs for the Dynamically created Partitions:

```
hadoop fs -mkdir /user/cloudera/covidAnalysis/partitions_testing
hadoop fs -mkdir /user/cloudera/covidAnalysis/partitions_covidIndia
```

4.2 Enabling Dynamic Partitioning and Bucketing in Hive:

```
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.enforce.bucketing = true;
```

```
0: jdbc:hive2://> set hive.exec.dynamic.partition = true;
No rows affected (0.32 seconds)
0: jdbc:hive2://> set hive.exec.dynamic.partition.mode = nonstrict;
No rows affected (0.007 seconds)
0: jdbc:hive2://> set hive.enforce.bucketing = true;
No rows affected (0.015 seconds)
```

4.3 Optimized External tables creation in Hive :

```
0: jdbc:hive2://> create external table testing_OP
. . . . . > (seq INT,
. . . . . > date DATE,
. . . . . > totalSamples INT,
. . . . . > negative INT,
. . . . . > positive INT)
. . . . . > PARTITIONED BY (state STRING)
. . . . . > CLUSTERED BY (date) into 4 BUCKETS
. . . . . > STORED AS ORC
. . . . . > LOCATION '/user/cloudera/covidAnalysis/partitions_testing'
. . . . . > TBLPROPERTIES('orc.compress' = 'SNAPPY');
OK

0: jdbc:hive2://> create external table covidIndia_OP
. . . . . > (sno INT,
. . . . . > date DATE,
. . . . . > cured INT,
. . . . . > deaths INT,
. . . . . > confirmed INT)
. . . . . > PARTITIONED BY (state STRING)
. . . . . > CLUSTERED BY (date) into 4 BUCKETS
. . . . . > STORED AS ORC
. . . . . > LOCATION '/user/cloudera/covidAnalysis/partitions_covidIndia'
. . . . . > TBLPROPERTIES('orc.compress' = 'SNAPPY');
OK
No rows affected (0.966 seconds)
```

Step 5: Load data to the optimized hive tables from normal hive tables.

The date in stateTesting table is of the format M/dd/yyyy . We need to

```
0: jdbc:hive2://> INSERT into TABLE testing_OP
. . . . . > PARTITION (state)
. . . . . > SELECT
. . . . . > seq,from_unixtime(unix_timestamp(date,'M/dd/yyyy'), 'yyyy-MM-dd'),
. . . . . > totalSamples,negative,positive,state
. . . . . > FROM stateTesting;
```

convert it to yyyy-MM-dd format.

The date in covidIndia table is of the format dd-MM-yyyy . We need

```
0: jdbc:hive2://> INSERT OVERWRITE TABLE covidIndia_OP
. . . . . > PARTITION (state)
. . . . . > SELECT
. . . . . > sno,from_unixtime(unix_timestamp(date,'dd-MM-yyyy'), 'yyyy-MM-dd'),
. . . . . > cured,deaths,confirmed,state
. . . . . > FROM covidIndia;
```

to convert it to yyyy-MM-dd format.

Verification:

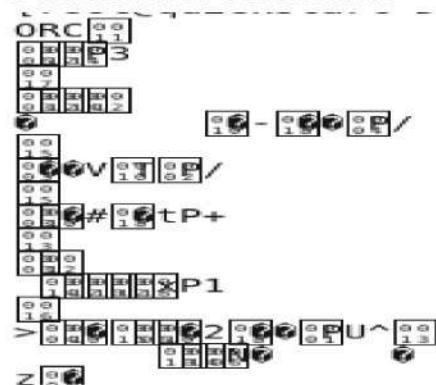
Partitions based on state

```
[root@quickstart Desktop]# hadoop fs -ls /user/cloudera/covidAnalysis/partitions_covidIndia
Found 38 items
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Andaman and Nicobar Isl
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Andhra Pradesh
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Arunachal Pradesh
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Assam
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Bihar
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Cases being reassigned
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Chandigarh
drwxr-xr-x - root cloudera          0 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Chhattisgarh
```

4 Buckets in each partition

```
[root@quickstart Desktop]# hadoop fs -ls /user/cloudera/covidAnalysis/partitions_covidIndia/state=Odisha
Found 4 items
-rw-r-xr-x 1 root cloudera      728 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Odisha/000000_0
-rw-r-xr-x 1 root cloudera      729 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Odisha/000001_0
-rw-r-xr-x 1 root cloudera      674 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Odisha/000002_0
-rw-r-xr-x 1 root cloudera      701 2023-07-16 09:04 /user/cloudera/covidAnalysis/partitions_covidIndia/state=Odisha/000003_0
```

Data stored in ORC format



Step 6-Inner Join two tables in Hive and get a consolidated table.

Performing Map side join on two columns ‘date’ and ‘state’.

Here it is assumed that the State_Testing table is small enough to fit in memory

6.1 Set the below Properties

```
set hive.auto.convert.join = false;
set hive.ignore.mapjoin_hint = false;
```

6.2 Execute Inner Join as Mapside join (testing_OP is the smaller table)

```
SELECT /*+ MAPJOIN(T)
*/ T.state, T.date, T.totalSamples, T.negative, T.positive, C.cured, C.deaths,
C.confirmed FROM testing_OP T JOIN covidIndia_OP C
ON (C.state = T.state) AND (C.date = T.date);
```

Joined Table Snapshot

t.state	t.date	t.totalsamples	t.negative	t.positive	c.cured	c.deaths	c.confirmed
Tamil Nadu	2020-06-09	621171	585678	34914	17527	286	33229
Tamil Nadu	2020-06-01	503339	479268	23495	12757	173	22333
Telangana	2020-04-19	14962	14104	858	186	18	844
Telangana	2020-05-16	23388	NULL	1551	959	34	1454
Telangana	2020-04-28	19063	NULL	1009	321	26	1004
Telangana	2020-04-29	19278	NULL	1016	367	26	1012
Tripura	2020-06-02	29066	28595	471	173	0	420
Tripura	2020-05-05	5850	5820	30	2	0	29
Tripura	2020-04-22	3215	3123	2	1	0	2
Tripura	2020-05-01	4828	4825	3	2	0	2
Tripura	2020-05-30	26376	26105	271	171	0	251
Tripura	2020-05-27	23264	23032	232	165	0	207
Tripura	2020-05-23	18737	18546	191	152	0	175

6.3 Create a consolidated table after join

```
CREATE TABLE covid_details AS
SELECT /*+ MAPJOIN(T)
*/T.state, T.date, T.totalSamples, T.negative, T.positive, C.cured, C.deaths,
C.confirmed FROM testing_OP T JOIN covidIndia_OP C
ON (C.state = T.state) AND (C.date = T.date);
```

Count:

_c0
1849

Step 7- Analysis

- Ideally, the number of samples tested positive and number of covid cases confirmed must be the same. See which state/states have more consistent data collection like The number of positive samples (table1) match mostly with number of confirmed cases(table2), for which state.**

Ideally, the number of samples tested positive and number of covid cases confirmed must be the same. So we find the percentage of records in each state where the data is consistent.

```
select state,avg(case when positive = confirmed then 1 else 0 end)*100 as
accuracy_percent
from covid_details
group by state
order by accuracy_percent;
```

state	accuracy_percent
Haryana	0.0
Telangana	0.0
Rajasthan	0.0
Punjab	1.5384615384615385
Uttar Pradesh	1.5384615384615385
Tamil Nadu	1.5384615384615385
Gujarat	1.5873015873015872
Jammu and Kashmir	1.5873015873015872
Delhi	3.225806451612903
Karnataka	4.545454545454546
Madhya Pradesh	7.5757575757576
Kerala	8.450704225352112
Jharkhand	11.864406779661017
Assam	13.043478260869565
Bihar	17.46031746031746
Arunachal Pradesh	18.867924528301888
Puducherry	20.754716981132077
Odisha	21.21212121212121
Tripura	23.91304347826087
Chandigarh	27.419354838709676
Nagaland	30.434782608695656
Andhra Pradesh	30.64516129032258
Himachal Pradesh	30.64516129032258
West Bengal	34.32835820895522
Uttarakhand	35.38461538461539
Chhattisgarh	42.857142857142854
Ladakh	46.42857142857143
Manipur	56.666666666666664
Goa	57.89473684210527
Dadra and Nagar Haveli	70.0
Sikkim	70.58823529411765
Maharashtra	72.72727272727273
Meghalaya	76.92307692307693
Mizoram	86.4406779661017
Andaman and Nicobar Islands	88.88888888888889

35 rows selected (73.028 seconds)

According to the above results, we can conclude the top 5 states with accurate covid testing for the given 2 months are - **Sikkim, Maharashtra, Meghalaya, Mizoram and Andaman and Nicobar Islands**

While the worst-performing ones are - **Haryana, Telangana, Rajasthan, Punjab, Uttar Pradesh, Tamil Nadu and Gujarat**

2. For every state, find the total number of confirmed cases reported and also the total number of positive samples tested, in the entire duration of 2 months, starting with the state with the highest cases.

If we scan the data we can see the data is cumulative for example TotalSamples shows cumulative total, Confirmed field shows cumulative value. So the value in each day is actually the total number till that day. Hence we only need to find the maximum number.

```
select state,max(positive) as totalPositives , max(confirmed) as
totalConfirmed from covid_details
group by state
order by totalConfirmed desc;
```

state	totalpositives	totalconfirmed
Maharashtra	90787	90787
Tamil Nadu	36841	34914
Delhi	32810	31309
Gujarat	21554	21014
Uttar Pradesh	11610	11335
Rajasthan	11600	11245
Madhya Pradesh	10049	9849
West Bengal	9328	8985
Karnataka	6041	5921
Bihar	5583	5459
Haryana	5438	5209
Andhra Pradesh	4126	5070
Jammu and Kashmir	4507	4346
Odisha	3250	3140
Assam	2937	2776
Punjab	2805	2719
Kerala	2162	2096
Uttarakhand	1560	1537
Telangana	1551	1454
Jharkhand	1423	1411
Chhattisgarh	1262	1240
Tripura	897	864
Himachal Pradesh	451	445
Goa	387	359
Chandigarh	328	323
Manipur	311	304
Nagaland	128	127
Puducherry	156	127
Ladakh	108	103
Arunachal Pradesh	61	57
Meghalaya	44	43
Mizoram	88	42
Andaman and Nicobar Islands	33	33
Dadra and Nagar Haveli	27	22
Sikkim	12	13

RESULT

Thus the program has been executed successfully.

Dataframe Basic Operations (Python)



[Import notebook](#)

Creating a SparkSession

- The SparkSession is the entry point to programming with Spark SQL.
- It allows you to create DataFrames, register DataFrames as tables, execute SQL over tables, cache tables, and read parquet files.
- **SparkSession.builder**: The builder attribute is a class attribute of SparkSession that provides a way to configure and create a SparkSession instance.
- **appName("Example App")**: The appName method sets the name of the Spark application. This name will appear in the Spark web UI and can help you identify your application among others running on the same cluster.
- **config("spark.some.config.option", "some-value")**: The config method allows you to set various configuration options for the Spark session. In `spark.some.config.option`, "" is a placeholder for an actual configuration key, and "some-value" is the value for that configuration. You can set multiple configuration options by chaining multiple config calls.
- **getOrCreate()**: The getOrCreate method either retrieves an existing SparkSession if one already exists or creates a new one if it does not. This ensures that you do not accidentally create multiple SparkSession instances in your application.

Note: In Databricks, you do not need to create or override the SparkSession as it is automatically created for each notebook or job executed against the cluster. Databricks manages the SparkSession and SparkContext for you, ensuring optimal configuration and resource usage.

```
from pyspark.sql import SparkSession  
spark = SparkSession.builder.appName("Spark DataFrames").config("spark.some.config.option", "some- value").getOrCreate()
```

Creating DataFrame

1. From Python a List of Tuples

```
%python
# List of tuples
data = [("John", 25), ("Doe", 30), ("Jane", 22)]

# Creating DataFrame
df_list = spark.createDataFrame(data, ["Name", "Age"])

# Display the DataFrame
df_list.show()
```

```
▶ df_list: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long]
+---+---+
|Name|Age|
+---+---+
|John| 25|
| Doe| 30|
|Jane| 22|
+   +   +
```

1. From a List of Dictionaries

```
%python
# List of dictionaries
data = [{"Name": "Alice", "Id": 1}, {"Name": "Bob", "Id": 2}, {"Name": "Cathy", "Id": 3}]

# Creating DataFrame
df_dict = spark.createDataFrame(data)

# Display the DataFrame df_dict.show()
```

2. From a List of Rows

```
%python
from pyspark.sql import Row

# List of Rows
data = [ Row(Name="Cathy", Id=1),
         Row(Name="David", Id=2),
         Row(Name="Eva", Id=3),
         Row(Name="Frank", Id=4)]

# Creating DataFrame
df_row = spark.createDataFrame(data)

# Display the DataFrame
df_row.show()
```

```
▶ df_row: pyspark.sql.dataframe.DataFrame = [Name: string, Id: long]
+----+---+
| Name| Id|
+----+---+
|Cathy| 1|
|David| 2|
| Eva| 3|
|Frank| 4|
+----+---+
```

1. Creating a DataFrame from an RDD

```
%python
# Import necessary modules from
pyspark.sql import Row

# Create an RDD
rdd = spark.sparkContext.parallelize([
    Row(Name="Alice", Age=25),
    Row(Name="Bob", Age=30),
    Row(Name="Cathy", Age=22),
    Row(Name="David", Age=35),
    Row(Name="Eva", Age=28),
    Row(Name="Frank", Age=40)
])

# Convert RDD to DataFrame
df_rdd = spark.createDataFrame(rdd)

# Display the DataFrame
df_rdd.show()
```

```
▶ df_rdd: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long]
+-----+---+
| Name|Age|
+-----+---+
|Alice| 25|
| Bob| 30|
|Cathy| 22|
|David| 35|
| Eva| 28|
|Frank| 40|
+-----+---+
```

2. Reading external file

spark.read: This is the entry point for reading data in Spark. It returns a DataFrameReader object that is used to read data from various sources.

.format("csv"): Specifies the format of the data source. In this case, it indicates that the data is in CSV (Comma- Separated Values) format.

.option("header", "true"): This option tells Spark that the first row of the CSV file contains the column names. If this option is set to false, Spark will treat the first row as data. "true" means that the CSV file has a header row.

.option("inferSchema", "true"): This option tells Spark to automatically infer the data types of each column in the CSV file. If this option is set to false, all columns will be read as strings (default behavior). "true" means that Spark will try to infer the schema (data types) of the columns based on the data.

```
.load("/FileStore/tables/retail_db/customers"):
```

This method specifies the path to the CSV file or directory containing CSV files that you want to read.

```
customer_df=spark.read.format("csv").option("header","true").option("inferSchema","true").load("dbfs:/FileStore/tables/customers_300mb.csv")
```

```
▶ [customer_df: pyspark.sql.dataframe.DataFrame = [customer_id: integer, name: string ... 5 more fields]]
```

3. Using StructType & StructField

```
%python
#employee data and schemas
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, FloatType, DateType from
datetime import date

# Create dummy data as a list of lists emp_data = [
[1, 101, "John Doe", 30, "M", 60000.0, date(2020, 1, 15)],
[2, 102, "Jane Smith", 25, "F", 65000.0, date(2019, 3, 10)],
[3, 101, "Mike Johnson", 35, "M", 70000.0, date(2018, 5, 20)],
[4, 103, "Emily Davis", 28, "F", 72000.0, date(2021, 7, 30)],
[5, 102, "Robert Brown", 40, "M", 80000.0, date(2017, 9, 25)],
[6, 101, "Linda Wilson", 32, "F", 68000.0, date(2020, 11, 5)],
[7, 103, "David Lee", 29, "M", 75000.0, date(2019, 12, 15)]]

# Define the schema
emp_schema = StructType([
    StructField("empid", StringType(), True),
    StructField("deptid", IntegerType(), True), StructField("name",
    StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("gender", StringType(), True), StructField("salary",
    FloatType(), True), StructField("hiredate", DateType(), True)
])
# Create DataFrame
df = spark.createDataFrame(emp_data, emp_schema)
#df = spark1.createDataFrame(data = emp_data, schema = emp_schema)

# Display the DataFrame df.show()
```

```

▶ df: pyspark.sql.dataframe.DataFrame = [empid: string, deptid: integer ... 5 more fields]
+-----+-----+-----+-----+-----+
|empid|deptid|      name|age|gender| salary| hiredate|
+-----+-----+-----+-----+-----+
| 1| 101| John Doe| 30| M|60000.0|2020-01-15|
| 2| 102| Jane Smith| 25| F|65000.0|2019-03-10|
| 3| 101|Mike Johnson| 35| M|70000.0|2018-05-20|
| 4| 103| Emily Davis| 28| F|72000.0|2021-07-30|
| 5| 102|Robert Brown| 40| M|80000.0|2017-09-25|
| 6| 101|Linda Wilson| 32| F|68000.0|2020-11-05|
| 7| 103| David Lee| 29| M|75000.0|2019-12-15|
+-----+-----+-----+-----+-----+

```

Basic DataFrame Operation

1. show() & display()

In Databricks, show() and display() are used to visualize DataFrames, but they have different functionalities:

show(): This is a method available on Spark DataFrames that prints the first n rows to the console. It is useful for quick inspection of data but does not provide rich formatting or interactivity. You can specify the number of rows to display, and it defaults to 20 rows if not specified.

display(): This is a Databricks-specific function that provides a rich, interactive view of the DataFrame. It is more suitable for use within notebooks as it allows for better visualization, including sorting, filtering, and graphical representation of data.

```

+-----+-----+-----+-----+-----+-----+
|customer_id|      name| city| state|country|registration_date|is_active|
+-----+-----+-----+-----+-----+-----+
| 0|Customer_0| Pune|Maharashtra| India| 2023-01-19| true|
| 1|Customer_1| Pune|West Bengal| India| 2023-08-10| true|
| 2|Customer_2| Delhi|Maharashtra| India| 2023-08-05| true|
| 3|Customer_3|Mumbai| Telangana| India| 2023-06-04| true|
| 4|Customer_4| Delhi| Karnataka| India| 2023-03-15| false|
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```
customer_df.show(5)
```

```
customer_df.display()
```

```
#display(customer_df)
```

Table ▾ +

Q Y E D

	customer_id	name	city	state	country	registration_date	is_active
1	0	Customer_0	Pune	Maharashtra	India	2023-01-19	true
2	1	Customer_1	Pune	West Bengal	India	2023-08-10	true
3	2	Customer_2	Delhi	Maharashtra	India	2023-08-05	true
4	3	Customer_3	Mumbai	Telangana	India	2023-06-04	true
5	4	Customer_4	Delhi	Karnataka	India	2023-03-15	false
6	5	Customer_5	Kolkata	West Bengal	India	2023-08-19	true
7	6	Customer_6	Kolkata	Tamil Nadu	India	2023-04-21	false
8	7	Customer_7	Mumbai	Telangana	India	2023-05-23	true
9	8	Customer_8	Pune	Tamil Nadu	India	2023-07-17	true
10	9	Customer_9	Delhi	Karnataka	India	2023-06-02	true
11	10	Customer_10	Hyderabad	Delhi	India	2023-02-23	true
12	11	Customer_11	Delhi	West Bengal	India	2023-11-08	true
13	12	Customer_12	Delhi	Delhi	India	2023-06-27	false
14	13	Customer_13	Pune	Maharashtra	India	2023-02-03	true
15	14	Customer_14	Chennai	Karnataka	India	2023-04-06	true

	customer_id	name	city	state	country	registration_date	is_active
14	13	Customer_13	Pune	Maharashtra	India	2023-02-03	true
15	14	Customer_14	Chennai	Karnataka	India	2023-04-06	true
16	15	Customer_15	Hyderabad	West Bengal	India	2023-03-31	true
17	16	Customer_16	Chennai	Maharashtra	India	2023-04-26	true
18	17	Customer_17	Pune	Delhi	India	2023-04-14	false
19	18	Customer_18	Chennai	Maharashtra	India	2023-02-04	false
20	19	Customer_19	Chennai	Karnataka	India	2023-01-22	true
21	20	Customer_20	Pune	Karnataka	India	2023-02-19	false
22	21	Customer_21	Hyderabad	Tamil Nadu	India	2023-09-16	true
23	22	Customer_22	Delhi	Karnataka	India	2023-12-27	true
24	23	Customer_23	Chennai	Tamil Nadu	India	2023-01-22	true
25	24	Customer_24	Pune	West Bengal	India	2023-09-11	true
26	25	Customer_25	Hyderabad	West Bengal	India	2023-08-22	true
27	26	Customer_26	Kolkata	West Bengal	India	2023-07-21	false
28	27	Customer_27	Pune	Gujarat	India	2023-02-09	false



10,000+ rows | Truncated data | 0.89s runtime

2. Columns & Prinschema()

In Spark, columns and printSchema() are used to inspect the structure of a DataFrame, but they serve different purposes:

- **columns:** This attribute returns a list of the column names in the DataFrame.
- **printSchema():** This method prints the schema of the DataFrame, including column names and data types, in a tree format.

```
customer_df.columns
```

```
root
|-- customer_id: integer (nullable = true)
|-- name: string (nullable = true)
|-- city: string (nullable = true)
|-- state: string (nullable = true)
|-- country: string (nullable = true)
|-- registration_date: date (nullable = true)
```

```
customer_df.printSchema()
```

```
['customer_id', 'name',
 'city',
 'state',
 'country',
 'registration_date',
 'is_active']
```

3. Select specific columns

```
customer_df.select("name","city").show()
```

Customer_5	Kolkata
Customer_6	Kolkata
Customer_7	Mumbai
Customer_8	Pune
Customer_9	Delhi
Customer_10	Hyderabad
Customer_11	Delhi
Customer_12	Delhi
Customer_13	Pune
Customer_14	Chennai
Customer_15	Hyderabad
Customer_16	Chennai
Customer_17	Pune
Customer_18	Chennai
Customer_19	Chennai

+-----+-----+

only showing top 20 rows

4. Filter rows

```
customer_df.filter(customer_df.city=="Hyderabad").show()
```

	25 Customer_25 Hyderabad West Bengal	India	2023-08-22	true	
	34 Customer_34 Hyderabad Telangana	India	2023-10-20	true	
	37 Customer_37 Hyderabad	Gujarat	India	2023-03-13	false
	38 Customer_38 Hyderabad	Karnataka	India	2023-06-19	false
	21 Customer_21 Hyderabad Tamil Nadu	India	2023-09-16	true	
	25 Customer_25 Hyderabad West Bengal	India	2023-08-22	true	
	34 Customer_34 Hyderabad	Telangana	India	2023-10-20	true
	37 Customer_37 Hyderabad 38	Gujarat	India	2023-03-13	false
	Customer_38 Hyderabad	Karnataka	India	2023-06-19	false
	40 Customer_40 Hyderabad Maharashtra	India	2023-07-29	false	
	44 Customer_44 Hyderabad Telangana 84	India	2023-08-18	false	
	Customer_84 Hyderabad Maharashtra	India	2023-04-08	false	
	100 Customer_100 Hyderabad Maharashtra	India	2023-12-30	false	
	110 Customer_110 Hyderabad Maharashtra	India	2023-03-14	false	
	118 Customer_118 Hyderabad	Gujarat	India	2023-01-27	false
	134 Customer_134 Hyderabad West Bengal	India	2023-06-25	true	
	137 Customer_137 Hyderabad Tamil Nadu	India	2023-03-11	true	
	138 Customer_138 Hyderabad	Delhi	India	2023-12-26	true
	149 Customer_149 Hyderabad	Karnataka	India	2023-09-21	false
	150 Customer_150 Hyderabad Maharashtra	India	2023-11-10	false	
	171 Customer_171 Hyderabad West Bengal	India	2023-12-24	true	
	173 Customer_173 Hyderabad	Gujarat	India	2023-05-30	false

+-----+-----+-----+-----+-----+

only showing top 20 rows

134 Customer_134 Hyderabad West Bengal India	2023-06-25	true
137 Customer_137 Hyderabad Tamil Nadu India	2023-03-11	true
138 Customer_138 Hyderabad Delhi India	2023-12-26	true
149 Customer_149 Hyderabad Karnataka India	2023-09-21	false
150 Customer_150 Hyderabad Maharashtra India	2023-11-10	false
171 Customer_171 Hyderabad West Bengal India	2023-12-24	true
173 Customer_173 Hyderabad Gujarat India	2023-05-30	false

only showing top 20 rows

5. Create or replace new column

The **withColumn** method is used to create a new column or replace an existing column in a DataFrame.

```
df.withColumn("name","defination")
```

```
%python
from pyspark.sql.functions import col, concat, lit

# col: A function to reference a column in a DataFrame.
# concat: A function to concatenate multiple columns or strings. # lit: A function to
create a column with a literal value.

# Example: Adding a new column
df_with_new_column = customer_df.withColumn("full name", concat(col("name"), lit(" Singh")))

# Display the DataFrame
df_with_new_column.show()
```

df_with_new_column: pyspark.sql.dataframe.DataFrame = [customer_id: integer, name: string ... 6 more fields]
2 Customer_2 Delhi Maharashtra India 2023-08-05 true Customer_2 Singh
3 Customer_3 Mumbai Telangana India 2023-06-04 true Customer_3 Singh
4 Customer_4 Delhi Karnataka India 2023-03-15 false Customer_4 Singh
5 Customer_5 Kolkata West Bengal India 2023-08-19 true Customer_5 Singh
6 Customer_6 Kolkata Tamil Nadu India 2023-04-21 false Customer_6 Singh
7 Customer_7 Mumbai Telangana India 2023-05-23 true Customer_7 Singh
8 Customer_8 Pune Tamil Nadu India 2023-07-17 true Customer_8 Singh
9 Customer_9 Delhi Karnataka India 2023-06-02 true Customer_9 Singh
10 Customer_10 Hyderabad Delhi India 2023-02-23 true Customer_10 Singh
11 Customer_11 Delhi West Bengal India 2023-11-08 true Customer_11 Singh
12 Customer_12 Delhi Delhi India 2023-06-27 false Customer_12 Singh
13 Customer_13 Pune Maharashtra India 2023-02-03 true Customer_13 Singh
14 Customer_14 Chennai Karnataka India 2023-04-06 true Customer_14 Singh
15 Customer_15 Hyderabad West Bengal India 2023-03-31 true Customer_15 Singh
16 Customer_16 Chennai Maharashtra India 2023-04-26 true Customer_16 Singh
17 Customer_17 Pune Delhi India 2023-04-14 false Customer_17 Singh
18 Customer_18 Chennai Maharashtra India 2023-02-04 false Customer_18 Singh
19 Customer_19 Chennai Karnataka India 2023-01-22 true Customer_19 Singh

only showing top 20 rows

withColumnRenamed

- The **withColumnRenamed** method is used to rename a single column in a DataFrame.

```
%python
# Example: Renaming a column
df_renamed_column = df_with_new_column.withColumnRenamed("full name", "Full Name")

# Display the DataFrame df_renamed_column.show()
```

	customer_id	name	city	state	country	date	is_customer
2 Customer_2	Customer_2	Delhi	Maharashtra	India		2023-08-05	true Customer_2 Singh
3 Customer_3	Customer_3	Mumbai	Telangana	India		2023-06-04	true Customer_3 Singh
4 Customer_4	Customer_4	Delhi	Karnataka	India		2023-03-15	false Customer_4 Singh
5 Customer_5	Customer_5	Kolkata	West Bengal	India		2023-08-19	true Customer_5 Singh
6 Customer_6	Customer_6	Kolkata	Tamil Nadu	India		2023-04-21	false Customer_6 Singh
7 Customer_7	Customer_7	Mumbai	Telangana	India		2023-05-23	true Customer_7 Singh
8 Customer_8	Customer_8	Pune	Tamil Nadu	India		2023-07-17	true Customer_8 Singh
9 Customer_9	Customer_9	Delhi	Karnataka	India		2023-06-02	true Customer_9 Singh
10 Customer_10	Customer_10	Hyderabad	Delhi	India		2023-02-23	true Customer_10 Singh
11 Customer_11	Customer_11	Delhi	West Bengal	India		2023-11-08	true Customer_11 Singh
12 Customer_12	Customer_12	Delhi	Delhi	India		2023-06-27	false Customer_12 Singh
13 Customer_13	Customer_13	Pune	Maharashtra	India		2023-02-03	true Customer_13 Singh
14 Customer_14	Customer_14	Chennai	Karnataka	India		2023-04-06	true Customer_14 Singh
15 Customer_15	Customer_15	Hyderabad	West Bengal	India		2023-03-31	true Customer_15 Singh
16 Customer_16	Customer_16	Chennai	Maharashtra	India		2023-04-26	true Customer_16 Singh
17 Customer_17	Customer_17	Pune	Delhi	India		2023-04-14	false Customer_17 Singh
18 Customer_18	Customer_18	Chennai	Maharashtra	India		2023-02-04	false Customer_18 Singh
19 Customer_19	Customer_19	Chennai	Karnataka	India		2023-01-22	true Customer_19 Singh

only showing top 20 rows

6. Dropping a Column

The drop method is used to remove one or more columns from a DataFrame.

```
# Dropping a single column
df_dropped_column = df_renamed_column.drop("Full Name")

# Display the DataFrame df_dropped_column.show()
```

	customer_id	name	city	state	country	date	is_customer
2 Customer_2	Customer_2	Delhi	Maharashtra	India		2023-08-05	true
3 Customer_3	Customer_3	Mumbai	Telangana	India		2023-06-04	true
4 Customer_4	Customer_4	Delhi	Karnataka	India		2023-03-15	false
5 Customer_5	Customer_5	Kolkata	West Bengal	India		2023-08-19	true
6 Customer_6	Customer_6	Kolkata	Tamil Nadu	India		2023-04-21	false
7 Customer_7	Customer_7	Mumbai	Telangana	India		2023-05-23	true
8 Customer_8	Customer_8	Pune	Tamil Nadu	India		2023-07-17	true
9 Customer_9	Customer_9	Delhi	Karnataka	India		2023-06-02	true
10 Customer_10	Customer_10	Hyderabad	Delhi	India		2023-02-23	true
11 Customer_11	Customer_11	Delhi	West Bengal	India		2023-11-08	true

18 Customer_18 Chennai Maharashtra India	2023-02-04	false
19 Customer_19 Chennai Karnataka India	2023-01-22	true

Dropping Multiple Columns

```
%python
# Dropping multiple columns
df_dropped_columns = df_renamed_column.drop("name", "country")
```

```
# Dropped columns shown below
df_dropped_columns.show()
```

3 Mumbai Telangana	2023-06-04	true Customer_3 Singh false
4 Delhi Karnataka	2023-03-15	Customer_4 Singh true
5 Kolkata West Bengal	2023-08-19	Customer_5 Singh false
6 Kolkata Tamil Nadu	2023-04-21	Customer_6 Singh true
7 Mumbai Telangana Pune	2023-05-23	Customer_7 Singh true
8 Tamil Nadu	2023-07-17	Customer_8 Singh true
9 Delhi Karnataka	2023-06-02	Customer_9 Singh
10 Hyderabad Delhi	2023-02-23	true Customer_10 Singh
11 Delhi West Bengal	2023-11-08	true Customer_11 Singh
12 Delhi Delhi	2023-06-27	false Customer_12 Singh
13 Pune Maharashtra	2023-02-03	true Customer_13 Singh
14 Chennai Karnataka	2023-04-06	true Customer_14 Singh
15 Hyderabad West Bengal	2023-03-31	true Customer_15 Singh
16 Chennai Maharashtra	2023-04-26	true Customer_16 Singh
17 Pune Delhi	2023-04-14	false Customer_17 Singh
18 Chennai Maharashtra	2023-02-04	false Customer_18 Singh
19 Chennai Karnataka	2023-01-22	true Customer_19 Singh

+ + + + + +
only showing top 20 rows

7. Removing Duplicate Rows

```
%python
# Removing duplicate rows
df_distinct = df_renamed_column.distinct()

# Display the DataFrame df_distinct.show()
```

```
df_distinct: pyspark.sql.dataframe.DataFrame = [customer_id: integer, name: string ... 6 more fields]
5| Customer_5| Kolkata|West Bengal| India| 2023-08-19| true| Customer_5 Singh|
6| Customer_6| Kolkata| Tamil Nadu| India| 2023-04-21| false| Customer_6 Singh|
```

3 Customer_3	Mumbai	Telangana	India	2023-06-04	true Customer_3 Singh
16 Customer_16	Chennai	Maharashtra	India	2023-04-26	true Customer_16 Singh
12 Customer_12	Delhi	Delhi	India	2023-06-27	false Customer_12 Singh
20 Customer_20	Pune	Karnataka	India	2023-02-19	false Customer_20 Singh
11 Customer_11	Delhi	West Bengal	India	2023-11-08	true Customer_11 Singh
4 Customer_4	Delhi	Karnataka	India	2023-03-15	false Customer_4 Singh
19 Customer_19	Chennai	Karnataka	India	2023-01-22	true Customer_19 Singh
7 Customer_7	Mumbai	Telangana	India	2023-05-23	true Customer_7 Singh
14 Customer_14	Chennai	Karnataka	India	2023-04-06	true Customer_14 Singh
1 Customer_1	Pune	West Bengal	India	2023-08-10	true Customer_1 Singh
13 Customer_13	Pune	Maharashtra	India	2023-02-03	true Customer_13 Singh

only showing top 20 rows

Aggregation

- Will cover in detail tomorrow

city	count
Bangalore 661013	
Chennai 660249	
Mumbai 661241	
Ahmedabad 660218	
Kolkata 660174	
Pune 660737	
Delhi 661025	
Hyderabad 662281	

PySpark SQL Joins comes with more optimizations by default (thanks to DataFrames), but there are still some p

Understanding how to effectively utilize PySpark joins is essential for conducting comprehensive data analysis # In this PySpark

SQL Join, you will learn different Join syntaxes and use different Join types on two or more

```
# PySpark Join Syntax #
PySpark Join Types #
Inner Join DataFrame
# Full Outer Join DataFrame #
Left Outer Join DataFrame #
Right Outer Join DataFrame #
Left Anti Join DataFrame #
Left Semi Join DataFrame #
Self Join DataFrame
# Using SQL Expression
```

Result:

This program has been executed successfully.

EXNO. :6

PYSPARK- JOIN and SQL Commands

Date:

PySpark Join is used to combine two DataFrames and by chaining these you can join multiple DataFrames; it

PySpark Joins are wider transformations that involve data shuffling across the network.

1. PySpark Join Syntax

PySpark SQL join has a below syntax and it can be accessed directly from DataFrame.

```
# Syntax  
#join(self, other, on=None, how=None)
```

#join() operation takes parameters as below and returns DataFrame.

```
# param other: Right side of the join  
# param on: a string for the join column name  
# param how: default inner. Must be one of inner, cross, outer, full, full_outer, left, left_outer, right, right
```

You can also write Join expression by adding where() and filter() methods on DataFrame and can have Join on m

2. PySpark Join Types

Below are the different Join Types PySpark supports.

# Join #	String Equivalent SQL Join
inner #	INNER JOIN
outer, #	full, fullouter, full_outer
left, #right,	leftouter, left_outer
# cross #	rightouter, right_outer
anti,	RIGHT JOIN
# semi,	leftanti, left_anti
	leftsemi, left_semi

Before diving into PySpark SQL Join illustrations, let's initiate "emp" and "dept" DataFrames.

The emp DataFrame contains the "emp_id" column with unique values, while the dept DataFrame contains the "dep" # Additionally, the "emp_dept_id" from "emp" refers to the "dept_id" in the "dept" dataset.

```
import findspark findspark.init()
```

```
# Prepare Data  
import pyspark  
from pyspark.sql import SparkSession  
  
spark = SparkSession.builder.appName('Joins').master('local[*]').getOrCreate()  
emp = [(1, "Smith", -1, "2018", "10",  
        "M", 3000),  
       (2, "Rose", 1, "2010", "20", "M", 4000),  
       (3, "Williams", 1, "2010", "10", "M", 1000),  
       (4, "Jones", 2, "2005", "10", "F", 2000),  
       (5, "Brown", 2, "2010", "40", "", -1),  
       (6, "Brown", 2, "2010", "50", "", -1)  
]
```

```
empColumns = ["emp_id", "name", "superior_emp_id", "year_joined", "emp_dept_id", "gender", "salary"]
```

```
empDF = spark.createDataFrame(data = emp, schema = empColumns) empDF.printSchema()  
empDF.show(truncate = False)
```

```
root
```

```
-- emp_id: long (nullable = true)  
-- name: string (nullable = true)  
-- superior_emp_id: long (nullable = true)  
-- year_joined: string (nullable = true)  
-- emp_dept_id: string (nullable = true)  
-- gender: string (nullable = true)  
-- salary: long (nullable = true)
```

```
+-----+-----+-----+-----+-----+-----+  
|emp_id|name |superior_emp_id|year_joined|emp_dept_id|gender|salary|  
+-----+-----+-----+-----+-----+-----+
```

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1	Smith	-1	2018	10	M	3000
2	Rose	1	2010	20	M	4000
3	Williams	1	2010	10	M	1000
4	Jones	2	2005	10	F	2000
5	Brown	2	2010	40		-1
6	Brown	2	2010	50		-1

```
dept = [("Finance", 10),  
        ("Marketing", 20),  
        ("Sales", 30),  
        ("IT", 40)  
    ]
```

```
deptColumns = ["dept_name", "dept_id"]  
  
deptDF = spark.createDataFrame(data = dept, schema = deptColumns)  
  
deptDF.printSchema()  
deptDF.show(truncate = False)
```

```
root
```

```
-- dept_name: string (nullable = true)  
-- dept_id: long (nullable = true)
```

```
+-----+  
|dept_name|dept_id|  
+-----+  
|Finance |10     |  
|Marketing|20     |  
|Sales   |30     |  
|IT      |40     |  
+-----+
```

3. How Join Works?

```
# PySpark's join operation combines data from two or more Datasets based on a  
common column or key. # It is a fundamental operation in PySpark and is similar to  
SQL joins.  
  
# Common Key:  
# =====  
  
# In order to join two or more datasets we need a common key or a column on which you want to  
join. This key is # Partitioning:  
# =====  
  
# PySpark Datasets are distributed and partitioned across multiple nodes  
in a cluster. # Ideally, data with the same join key should be located in the  
same partition.  
# If the Datasets are not already partitioned on the join key, PySpark may perform a shuffle  
operation to redis # Shuffling can be an expensive operation, especially for large Datasets.  
  
# Join Type Specification:  
# =====  
  
# We can specify the type of join like inner join, full join, left join, etc., by specifying on "how"  
parameter # This parameter determines which rows should be included or excluded in the  
resulting Dataset.  
  
# Join Execution:  
# =====  
  
# PySpark performs the join by comparing the values in the common key column  
between the Datasets. # Inner Join:  
# =====  
  
# Returns only the rows with matching keys in both DataFrames.
```

```

# Left Join:
# =====

# Returns all rows from the left DataFrame and matching rows from the right DataFrame. #

Right Join:
# =====

# Returns all rows from the right DataFrame and matching rows from the left DataFrame. # Full

Outer Join:
# =====

# Returns all rows from both DataFrames, including matching and non-matching rows. #

Left Semi Join:
# =====

# Returns all rows from the left DataFrame where there is a match in the right DataFrame. # Left

Anti Join:
# =====

# Returns all rows from the left DataFrame where there is no match in the right DataFrame.

```

4. PySpark Inner Join DataFrame

The default join in PySpark is the inner join, commonly used to retrieve

```

# Inner Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "inner") \
    .show(truncate = False)

```

data from two or more DataFrames based on a shared key. An Inner join combines two DataFrames based on the key (common column) provided and results in rows where there is a matching found. Rows from both DataFrames are dropped with a non-matching key.

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		-1	IT	40

This example drops “emp_dept_id” with value 50 from “emp” And “dept_id” with value 30 from “dept” datasets. Following is the result of the above Join statement.

5. PySpark Left Outer Join

Left a.k.a Leftouter join returns all rows from the left dataset regardless of match found on the right dataset when join expression doesn't match, it assigns null for that record and drops records from right where match not found.

```
# Left Outer Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "left") \
.show(truncate = False)

empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "leftouter") \
.show(truncate = False)
```

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
5	Brown	2	2010	40		-1	IT	40
6	Brown	2	2010	50		-1	NULL	NULL

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
5	Brown	2	2010	40		-1	IT	40
6	Brown	2	2010	50		-1	NULL	NULL

In our dataset, the record with “emp_dept_id” 50 does not have a corresponding entry in the “dept” dataset, resulting in null values in the “dept” columns (dept_name & dept_id). Additionally, the entry with “dept_id” 30 from the “dept” dataset is omitted from the results. Above is the outcome of the provided join expression.

6. Right Outer Join

Right a.k.a Rightouter join is opposite of left join, here it returns all rows from the right dataset regardless of match found on the left dataset, when join expression doesn’t match, it assigns null for that record and drops records from left where match not found.

```
# Right Outer Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "right") \
    .show(truncate = False)
```

```
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "rightouter") \
    .show(truncate = False)
```

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
4	Jones	2	2005	10	F	2000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
1	Smith	-1	2018	10	M	3000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40	-1	IT	40	

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
4	Jones	2	2005	10	F	2000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
1	Smith	-1	2018	10	M	3000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40	-1	IT	40	

In our example, the dataset on the right, containing “dept_id” with a value of 30, does not have a corresponding record in the left dataset “emp”. Consequently, this record contains null values for the columns from “emp”. Additionally, the record with “emp_dept_id” value 50 is dropped

as no match was found in the left dataset. Below is the result of the aforementioned join expression.

7. PySpark Full Outer Join

Outer a.k.a full, fullouter join in PySpark combines the results of both left and right outer joins, ensuring that all records from both DataFrames are

```
# Full Outer Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "outer") \
    .show(truncate = False)

empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "full") \
    .show(truncate = False)

empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "fullouter") \
    .show(truncate = False)
```

included in the resulting DataFrame. It includes all rows from both DataFrames and fills in missing values with nulls where there is no match. In other words, it merges the DataFrames based on a common key, but retains all rows from both DataFrames, even if there's no match. This join type is useful when you want to preserve all the information from both datasets, regardless of whether there's a match on the key or not.

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40	-1	IT	40	
6	Brown	2	2010	50	-1	NULL	NULL	

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40	-1	IT	40	
6	Brown	2	2010	50	-1	NULL	NULL	

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40	-1	IT	40	
6	Brown	2	2010	50	-1	NULL	NULL	

This code snippet performs a full outer join between two PySpark DataFrames, empDF and deptDF, based on the condition that emp_dept_id from empDF is equal to dept_id from deptDF. In our “emp” dataset, the “emp_dept_id” with a value of 50 does not have a corresponding record in the “dept” dataset, resulting in null values in the “dept” columns. Similarly, the “dept_id” 30 does not have a record in the “emp” dataset, hence you observe null values in the “emp” columns.

Above Code is the output of the provided join example.

8. Left Semi Join

A Left Semi Join in PySpark returns only the rows from the left DataFrame (the first DataFrame mentioned in the join operation) where

```
# Left Semi Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "leftsemi") \
.show(truncate = False)
```

there is a match with the right DataFrame (the second DataFrame). It does not include any columns from the right DataFrame in the resulting DataFrame. This join type is useful when you only want to filter rows from the left DataFrame based on whether they have a matching key in the right DataFrame. Left Semi Join can also be achieved by selecting only the columns from the left dataset from the result of the inner join

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1	Smith	-1	2018	10	M	3000
3	Williams	1	2010	10	M	1000
4	Jones	2	2005	10	F	2000
2	Rose	1	2010	20	M	4000
5	Brown	2	2010	40		-1

9. Left Anti Join

A Left Anti Join in PySpark returns only the rows from the left DataFrame (the first DataFrame mentioned in the join operation) where there is no

```
# Left Anti Join
empDF.join(deptDF, empDF.emp_dept_id == deptDF.dept_id, "leftanti") \
.show(truncate = False)
```

match with the right DataFrame (the second DataFrame). It excludes any rows from the left DataFrame that have a corresponding key in the right DataFrame. This join type is useful when you want to filter out rows from the left DataFrame that have matching keys in the right DataFrame.

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
6	Brown	2	2010	50		-1

10. PySpark Self Join

Joins are not complete without a self join, Though there is no self-join

```
# Self Join
from pyspark.sql.functions import col
empDF.alias("emp1").join(empDF.alias("emp2"),\
    col("emp1.superior_emp_id") == col("emp2.emp_id"), "inner")\
    .select(col("emp1.emp_id"), col("emp1.name"), \
    col("emp2.emp_id").alias("superior_emp_id"), \
    col("emp2.name").alias("superior_emp_name"))\
    .show(truncate = False)
```

type available in PySpark, we can use any of the above-explained join type s to join DataFrame to itself. below example use inner self join.

emp_id	name	superior_emp_id	superior_emp_name
2	Rose	1	Smith
3	Williams	1	Smith
4	Jones	2	Rose
5	Brown	2	Rose
6	Brown	2	Rose

11. Using SQL Expression

Alternatively, you can also use SQL query to join DataFrames/tables in

```
# Using spark.sql
empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF = spark.sql("SELECT * FROM EMP e, DEPT d WHERE e.emp_dept_id == d.dept_id")\
    .show(truncate = False)

joinDF2 = spark.sql("SELECT * FROM EMP e INNER JOIN DEPT d ON e.emp_dept_id == d.dept_id")\
    .show(truncate = False)
```

PySpark. To do so, first, create a temporary view using `createOrReplaceTempView()`, then use the `spark.sql()` to execute the join query.

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		-1	IT	40

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		-1	IT	40

12. PySpark SQL Join on multiple DataFrames

When you need to join more than two tables, you either use SQL expression after creating a temporary view on the DataFrame or use the result of join operation to join with another DataFrame like chaining them. for example# Join on multiple dataframes df1.join(df2, df1.id1 == df2.id2, "inner") \ .join(df2, df1.id1 == df3.id3, "inner")

13. PySpark SQL Join Complete Example

```
import pyspark

from pyspark.sql import SparkSession
from pyspark.sql.functions import col
spark =
SparkSession.builder.appName('Join').master('local[*]').getOrCreate()
emp = [(1,"Smith",-1,"2018","10","M",3000), \
```

```

(2,"Rose",1,"2010","20","M",4000), \
(3,"Williams",1,"2010","10","M",1000), \
(4,"Jones",2,"2005","10","F",2000), \
(5,"Brown",2,"2010","40","", -1), \
(6,"Brown",2,"2010","50","", -1) \
]

empColumns = ["emp_id", "name", "superior_emp_id",
               "year_joined", \ "emp_dept_id", "gender",
               "salary"]

empDF = spark.createDataFrame(data=emp, schema =
empColumns) empDF.printSchema()
empDF.show(truncate=False)

dept = [("Finance",10), \
        ("Marketing",20), \
        ("Sales",30), \
        ("IT",40) \
]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema =
deptColumns) deptDF.printSchema()
deptDF.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer") \
.show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"full") \
.show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"fullouter") \
.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left") \
.show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftouter") \
.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right") \
.show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"rightouter") \
.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi") \
.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftanti") \

```

```

.show(truncate=False)

empDF.alias("emp1").join(empDF.alias("emp2"), \
    col("emp1.superior_emp_id") == \
    col("emp2.emp_id"),"inner") \
.select(col("emp1.emp_id"),col("emp1.name"), \
    col("emp2.emp_id").alias("superior_emp_id"), \
    col("emp2.name").alias("superior_emp_name")) \
.show(truncate=False)

empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF = spark.sql("select * from EMP e, DEPT d where e.emp_dept_id == d.dept_id") \
.show(truncate=False)

joinDF2 = spark.sql("select * from EMP e INNER JOIN DEPT d ON e.emp_dept_id == \
d.dept_id") \
.show(truncate=False)

root
|-- emp_id: long (nullable = true)
|-- name: string (nullable = true)
|-- superior_emp_id: long (nullable = true)
|-- year_joined: string (nullable = true)
|-- emp_dept_id: string (nullable = true)
|-- gender: string (nullable = true)
|-- salary: long (nullable = true)

+-----+-----+-----+-----+-----+-----+
|emp_id|name |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+-----+-----+-----+-----+-----+-----+
|1     |Smith |-1           |2018      |10        |M     |3000  |
|2     |Rose  |1           |2010      |20        |M     |4000  |
|3     |Williams|1         |2010      |10        |M     |1000  |
|4     |Jones |2           |2005      |10        |F     |2000  |
|5     |Brown|2           |2010      |40        |      |-1    |
|6     |Brown|2           |2010      |50        |      |-1    |
+-----+-----+-----+-----+-----+-----+

```

root

```
-- dept_name: string (nullable = true)
-- dept_id: long (nullable = true)
```

dept_name	dept_id
Finance	10
Marketing	20
Sales	30
IT	40

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		1	IT	40

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2010	20	M	4000	Marketing	20
2	Rose	1	2010	40		1	Sales	30
NULL	NULL	NULL	2010	50		1	IT	40
5	Brown	2	NULL	+	+	+	NULL	+ NULL
6	Brown	2	NULL	+	+	+	+ NULL	+ NULL

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40		1	IT	40
6	Brown	2	2010	50		1	NULL	NULL

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	Sales	30
5	Brown	2	2010	40		1	IT	40
6	Brown	2	2010	50		1	NULL	NULL

2010	20	M	4000	Marketing	20	
NULL	NULL	NULL	NULL	Sales	30	
2010	40		1	IT	40	
2010	50		1	NULL	NULL	
+	+	+	+	+	+	+

+-----+ emp_id name superior_emp_id year_joined emp_dept_id gender salary dept_name dept_id +-----+
1 Smith -1 2018 10 M 3000 Finance 10
2 Rose 1 2010 20 M 4000 Marketing 20
3 Williams 1 2010 10 M 1000 Finance 10
4 Jones 2 2005 10 F 2000 Finance 10
5 Brown 2 2010 40 -1 IT 40
6 Brown 2 2010 50 -1 NULL NULL

+-----+ emp_id name superior_emp_id year_joined emp_dept_id gender salary dept_name dept_id +-----+
1 Smith -1 2018 10 M 3000 Finance 10
2 Rose 1 2010 20 M 4000 Marketing 20
3 Williams 1 2010 10 M 1000 Finance 10
4 Jones 2 2005 10 F 2000 Finance 10
5 Brown 2 2010 40 -1 IT 40
6 Brown 2 2010 50 -1 NULL NULL

+-----+ emp_id name superior_emp_id year_joined emp_dept_id gender salary dept_name dept_id +-----+
4 Jones 2 2005 10 F 2000 Finance 10
3 Williams 1 2010 10 M 1000 Finance 10
1 Smith 1 2018 10 M 3000 Finance 10
2 Rose 1 2010 20 M 4000 Marketing 20
NULL NULL NULL NULL NULL NULL Sales 30
5 Brown 2 2010 40 -1 IT 40

+-----+ emp_id name superior_emp_id year_joined emp_dept_id gender salary dept_name dept_id +-----+
4 Jones 2 2005 10 F 2000 Finance 10
3 Williams 1 2010 10 M 1000 Finance 10
1 Smith 1 2018 10 M 3000 Finance 10
2 Rose 1 2010 20 M 4000 Marketing 20
NULL NULL NULL NULL NULL NULL Sales 30
5 Brown 2 2010 40 -1 IT 40

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1	Smith	-1	2018	10	M	3000
3	Williams	1	2010	10	M	1000
4	Jones	2	2005	10	F	2000
2	Rose	1	2010	20	M	4000
5	Brown	2	2010	40		-1

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
6	Brown	2	2010	50		-1

emp_id	name	superior_emp_id	superior_emp_name
2	Rose	1	Smith
3	Williams	1	Smith
4	Jones	2	Rose
5	Brown	2	Rose
6	Brown	2	Rose

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		-1	IT	40

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
5	Brown	2	2010	40		-1	IT	40

Result:

Thus we learned all Pyspark commands.

