Ex. No: 1 Basic Linux Commands

1. Display your current directory. (pwd)
2. List the contents of the root directory. (ls /)
3. List a long listing of the root directory. (ls / -all)
4. Stay where you are, and list the contents of /etc. (ls /etc)
5. Create a directory testdir in your home directory.

   mkdir /home/karthi/Desktop/sample
6. Remove the directory sample.

   mkdir /home/karthi/Desktop/sample
7. Create the files today.txt and yesterday.txt in folder all in desktop

   touch /home/karthi/Desktop/all/today.txt

   touch /home/karthi/Desktop/all/yesterday.txt
8. Move folder all to sample folder in desktop

   mv /home/karthi/Desktop/all /home/karthi/Desktop/sample
9. Move only the contents of Folder all to Desktop (today.txt and Yesterday.txt)

   mv /home/karthi/Desktop/sample/all/* /home/karthi/Desktop/
10. Remove folder all

    rm -r /home/karthi/Desktop/all
11. Display a list of all logged on users. who
12. Create a user account named serena, including a home directory and a description (or comment) that reads Serena. Do all this in one single command.

    sudo useradd -m -c 'Serena Williams' serena

    sudo useradd -p $(openssl passwd serena) -m serena
13. Use su to switch to another user account and display the home directory of serena

    su serena

    pwd

    The su command allows a user to run a shell as another user.
14. Delete the user Serena

sudo userdel -r serena

15. List the files with their access permissions for sample.txt Change the permission of file sample.txt in desktop to all users

    ls –l

    chmod 777 sample.txt (Gives read/write and execute permission)

    First rwx triplet represents the permissions for the user owner. The second triplet - group owner; third triplet defines permissions for all other users that are not the user owner and are not a member of the group owner.

16. Give only read permission for group users in sample.txt file

    chmod 747 sample.txt or chmod g-x-w sample.txt

17. Find the difference between su and sudo command

    The su command allows a user to run a shell as another user.

    The sudo program allows a user to start a program with the credentials of another user.

    To perform tasks as root, the first user is given all sudo rights via the /etc/sudoers. In fact all users that are members of the admin group can use sudo to run all commands as root.

18. Create the groups tennis

    sudo addgroup tennis

19. In one command, make serena a member of tennis

    sudo usermod -a -G tennis serena

20. Delete the group

    sudo usermod -a -G tennis serena

Evaluation 1 - Linux Commands

1. Find out the users who are currently logged in
2. Display the name of your home directory.
3. Create a directory SAMPLE under your home directory.
4. Create a sub-directory by name TRIAL under SAMPLE.
5. Change from home directory to TRIAL by using absolute and relative pathname.
6. Remove directory TRIAL.
7. Create files myfile and yourfile under SAMPLE Directory.
8. Remove SAMPLE directory with files by using a single command.
9. Is there any command available to get back a deleted file?
10. Login as root and create groups as dba with id 501 & stud with id 555
11. Create the following list of users

    User name UID GID Working Shell Secondary Comments Group

    Mac1 501 501 Bourne shell 555 Mac1 user
12. Identify the available memory in the system.
13. Login as a normal user
14. Create file test
15. Find the permissions of file test
16. Change the ownership of the file to MAC1
17. Switch to Super User Account
18. Change group of file test
19. Create a file testfile in testdir
20. Verify the ownership and the group of the testfile
21. Create three sample directories with some files to use with the tar command.
22. Use the tar command to backup all three directories into single tar file.
23. List the directory that contains binary files in your system
24. Difference between su and sudo– command.
25. List the directory that holds the configuration files.

Ex.No:2 Hadoop Installation

Hadoop 2.6.5 Installing on Ubuntu 16.04 (Single-Node Cluster)

Step 1: Update the OS

```
npprakash@npprakashhp:~$ sudo apt-get update
```

//*apt-get update* updates the package lists for upgrades for packages that need upgrading, as well as new packages that have just come to the repositories.

Step 2: Installing Java

```
npprakash@npprakashhp:~$ sudo apt-get install default-jdk
```

Step 2.1 Check the version

```
npprakash@npprakashhp:~$ java -version
```

```
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Step 3: Adding a dedicated Hadoop user

The next step is to create a dedicated user and group for our Hadoop installation. This allows all of the installation to be insulated from the rest of the environment, as well as enable tighter security measures to be enforced (in case you have a production environment). We will create a user hduser and a group hadoop, and add the user to the group. This can be done using the following commands.

```
npprakash@npprakashhp:~$ sudo addgroup hadoop

Adding group `hadoop' (GID 1001) ...
Done.

npprakash@npprakashhp:~$ sudo adduser --ingroup hadoop hduser

Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
```

```
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

Step 3.1 We can check if we create the **hadoop** group and **hduser** user:

```
npprakash@npprakashhp:~$ groups hduser

    hduser : hadoop sudo
```

Step 4 : Installing SSH

The hadoop control scripts rely on SSH to peform cluster-wide operations. For example, there is a script for stopping and starting all the daemons in the clusters. To work seamlessly, SSH needs to be setup to allow password-less login for the hadoop user from machines in the cluster. The simplest way to achive this is to generate a public/private key pair, and it will be shared across the cluster.

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus your local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost for the hduser user we created in the earlier.

We have to generate an SSH key for the hduser user.

```
npprakash@npprakashhp:~$ sudo apt-get install ssh
```

This will install ssh on our machine. If we get something similar to the following, we can think it is setup properly:

```
npprakash@npprakashhp:~$ which ssh

    /usr/bin/ssh

npprakash@npprakashhp:~$ which sshd

    /usr/sbin/sshd
```

Step 4.1 :

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

```
npprakash@npprakashhp:~$ su hduser

hduser@npprakashhp:/home/npprakash$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
```

```
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:/M18Dv+ku5js8npZvYi45Fr4F84SzoqXBUO5xAfo+/8 hduser@npprakashhp
The key's randomart image is:
+---[RSA 2048]----+
|      o.o        |
|     . = .       |
|    . o o        |
|     . =         |
|      . S       .|
|     .  .+ +    o|
|      ..=o* * .oo|
|      .+== *.B++ |
|     ..o+==EB*B+.|
+----[SHA256]-----+
```

Step 4.2 The following command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

```
hduser@npprakashhp:/home/npprakash$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys
```

```
Step 4.3 We can check if ssh works:
```

```
hduser@npprakashhp:/home/npprakash$  ssh localhost
```

```
hduser@localhost's password:

Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
232 packages can be updated.
117 updates are security updates.
Last login: Wed Nov  1 19:38:55 2017 from 127.0.0.1
```

Step 5: Install Hadoop

```
hduser@npprakashhp:~$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-
2.6.5/hadoop-2.6.5.tar.gz
```

```
hduser@npprakashhp:~$ tar xvzf hadoop-2.6.5.tar.gz
```

Step 5.1 : move the Hadoop installation to the **/usr/local/hadoop** directory. So, we should create the directory first:

```
hduser@npprakashhp:~$ sudo mkdir -p /usr/local/hadoop
```

Step 5.2 We can check again if **hduser** is not in **sudo** group:

```
hduser@npprakashhp:~$  sudo -v
Sorry, user hduser may not run sudo on laptop.
```

This can be resolved by logging in as a root user, and then add **hduser** to **sudo** group:

```
hduser@npprakashhp:~$ su npprakash
```

Password:
npprakash@npprakashhp:/home/hduser$

Now, the **hduser** has root priviledge, we can move the Hadoop installation to the **/usr/local/hadoop** directory without any problem:

npprakash@npprakashhp:/home/hduser$ sudo su hduser

```
hduser@npprakashhp:~/hadoop-2.6.5$ sudo mv * /usr/local/hadoop
hduser@npprakashhp:~/hadoop-2.6.5$ sudo chown -R hduser:hadoop /usr/local/hadoop
```

Step 6: Setup Configuration Files

Step 6.1 : Edit ~/.bashrc file

Before editing the **.bashrc** file in **hduser**'s home directory, we need to find the path where Java has been installed to set the **JAVA_HOME** environment variable using the following command:
hduser@npprakashhp:~$ update-alternatives --config java
There is only one alternative in link group java (providing /usr/bin/java): /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
Nothing to configure.
Now we can append the following to the end of **~/.bashrc**:

```
hduser@npprakashhp:~$  vim ~/.bashrc

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

hduser@laptop:~$ source ~/.bashrc
```

Step 6.2 : **2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh**
Adding the belwo statement in the **hadoop-env.sh** file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

```
hduser@npprakashhp:~$   vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Step 6.3 : **3. /usr/local/hadoop/etc/hadoop/core-site.xml**:

The **/usr/local/hadoop/etc/hadoop/core-site.xml** file contains configuration properties that Hadoop uses when starting up.

This file can be used to override the default settings that Hadoop starts with.

```
hduser@npprakashhp:~$ sudo mkdir -p /app/hadoop/tmp

hduser@npprakashhp:~$ sudo chown hduser:hadoop /app/hadoop/tmp
```

Open the file and enter the following in between the <configuration></configuration> tag:

```
hduser@npprakashhp:~$ vi /usr/local/hadoop/etc/hadoop/core-site.xml

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

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

Step 6.4 : **/usr/local/hadoop/etc/hadoop/mapred-site.xml**

By default, the /usr/local/hadoop/etc/hadoop/ folder contains
/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
file which has to be renamed/copied with the name mapred-site.xml:

```
hduser@npprakashhp:~$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml

The /usr/local/hadoop/etc/hadoop/mapred-site.xml file is used to specify which
framework is being used for MapReduce.
We need to enter the following content in between the
<configuration></configuration> tag:

hduser@npprakashhp:~$ vim usr/local/hadoop/etc/hadoop/mapred-site.xml

<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
 </property>
```

```
</configuration>
```

Step 6.5 : /usr/local/hadoop/etc/hadoop/hdfs-site.xml

The /usr/local/hadoop/etc/hadoop/hdfs-site.xml file needs to be configured for each host in the cluster that is being used.

It specifies the directories which will be used as the namenode and the datanode on that host.
Before editing this file, we need to create two directories which will contain the namenode and the datanode for this Hadoop installation.

This can be done using the following commands:

```
hduser@npprakashhp:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode

hduser@npprakashhp:~$  sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode

hduser@npprakashhp:~$ sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

Open the file and enter the following content in between the <configuration></configuration> tag:

```
hduser@npprakashhp:~$ vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>
```

Step 7. Format the New Hadoop Filesystem

Now, the Hadoop file system needs to be formatted so that we can start to use it. The **format** command should be issued with write permission since it creates **current** directory under **/usr/local/hadoop_store/hdfs/namenode** folder:

```
hduser@npprakashhp:~$ hadoop namenode -format
```

Important Note :

- ➔ **Note that hadoop namenode -format command should be executed once before we start using Hadoop.**

➔ **If this command is executed again after Hadoop has been used, it'll destroy all the data on the Hadoop file system.**

Step 8 : Starting Hadoop

Start NameNode daemon and DataNode daemon:

```
hduser@npprakashhp:/usr/local/hadoop/sbin$ start-dfs.sh
```

Start ResourceManager daemon and NodeManager daemon:

```
hduser@npprakashhp:/usr/local/hadoop/sbin$ start-yarn.sh
```

We can check if it's really up and running:

```
hduser@npprakashhp:/usr/local/hadoop/sbin$ jps
7040 NameNode
7956 Jps
7156 DataNode
7525 ResourceManager
7367 SecondaryNameNode
7834 NodeManager
```

Step 9: Stopping Hadoop

```
hduser@npprakashhp:/usr/local/hadoop/sbin$ stop-dfs.sh
hduser@npprakashhp:/usr/local/hadoop/sbin$ stop-yarn.sh
```

Step 10 : http://localhost:50070/ to check the hadoop User interface

Errors and Other Commands

Note : Sudoers Issue ::

edit /etc/sudoers

change the user privilege specficiation to hduser

hduser ALL=(ALL:ALL) ALL

or use:

at **root** the following command:

sudo usermod -a -G sudo hduser

Some important commands

To remove hduser:

deluser hduser

To remove hadoop group

deluser –-group hadoop

To uninstall hadoop

sudo rm -r -f location (/usr/local/hadoop)

To un install java

Remove all the Java related packages (Sun, Oracle, OpenJDK, IcedTea plugins, GIJ):

dpkg-query -W -f='${binary:Package}\n' | grep -E -e '^(ia32-)?(sun|oracle)-java' -e '^openjdk-' -e '^icedtea' -e '^(default|gcj)-j(re|dk)' -e '^gcj-(.*)-j(re|dk)' -e '^java-common' | xargs sudo apt-get -y remove
sudo apt-get -y autoremove

- **Purge config files:**

dpkg -l | grep ^rc | awk '{print($2)}' | xargs sudo apt-get -y purge

- **Remove Java config and cache directory:**

sudo bash -c 'ls -d /home/*/.java' | xargs sudo rm -rf

- **Remove manually installed JVMs:**

sudo rm -rf /usr/lib/jvm/*

- **Remove Java entries, if there is still any, from the *alternatives*:**

for g in ControlPanel java java_vm javaws jcontrol jexec keytool mozilla-javaplugin.so orbd pack200 policytool rmid rmiregistry servertool tnameserv unpack200 appletviewer apt extcheck HtmlConverter idlj jar jarsigner javac javadoc javah javap jconsole jdb jhat jinfo jmap jps jrunscript jsadebugd jstack jstat jstatd native2ascii rmic schemagen serialver wsgen wsimport xjc xulrunner-1.9-javaplugin.so; do sudo update-alternatives --remove-all $g; done

- **Search for possible remaining Java directories:**

sudo updatedb
sudo locate -b '\pack200'

If the command above produces any output like /path/to/jre1.6.0_34/bin/pack200 remove the directory that is parent of **bin**, like this: sudo rm -rf /path/to/jre1.6.0_34.

to check for Installations of Java

update-alternatives --config java

Ex. No 3 Hadoop Commands

Hadoop Commands *Hadoop Commands - 33 Frequently used HDFS shell commands*

Open a terminal window to the current working directory. The deafult directory is /user/hduser

1. Print the Hadoop version
hadoop version

2. List the contents of the root directory in HDFS
#
hadoop fs -ls /

hdfs dfs –ls /

3. Report the amount of space used and available on currently mounted file system
#
hadoop fs -df hdfs:/

4. Count the number of directories, files and bytes under the paths that match the specified file pattern (count the dir, file and bytes in specfied dir)
#
hadoop fs -count hdfs:/

5. Run a DFS file system checking utility checks all the files are health
#
hadoop fsck / - files

6. Run a cluster balancing utility
#
hadoop balancer

7. Create a new directory named “hadoop” below the /user/hduser directory in HDFS. Since you’re
currently logged in with the “hduser” user ID /user/hduser is your home directory in HDFS.
#
hadoop fs -mkdir /user/hduser/hadoop

8. Add a sample text file from the local directory named “data” to the new directory you created in HDFS during the previous step.
#
hadoop fs -put data/sample.txt /user/hduser/hadoop

9. List the contents of this new directory in HDFS.
#
hadoop fs -ls /user/hduser/hadoop

10. Add the entire local directory called “retail” to the /user/hduser directory in HDFS.
#

hadoop fs -put data/retail /user/hduser/hadoop

11. Since /user/hduser is your home directory in HDFS, any command that does not have an absolute path is interpreted as relative to that directory. The next command will therefore list your home directory, and should show the itmems you've just added there.
#
hadoop fs –ls

12. See how much space this directory occupies in HDFS.
#
hadoop fs -du -s -h hadoop/retail

13. Delete a file 'customers' from the "retail" directory.
#
hadoop fs -rm hadoop/retail/customers

14. Ensure this file is no longer in HDFS.
#
hadoop fs -ls hadoop/retail/customers

15. Delete all files from the "retail" directory using a wildcard.
#
hadoop fs -rm hadoop/retail/*

16. To empty the trash
#
hadoop fs -expunge

17. Finally, remove the entire retail directory and all of its contents in HDFS.
#
hadoop fs -rm -r hadoop/retail

18. List the hadoop directory again
#
hadoop fs -ls hadoop

19. Add the purchases.txt file from the local directory
named "/home/hduser/" to the hadoop directory you created in HDFS
#
hadoop fs -copyFromLocal /home/sam/purchases.txt hadoop/

20. To view the contents of your text file purchases.txt which is present in your hadoop directory.
#
hadoop fs -cat hadoop/purchases.txt

21. Add the purchases.txt file from "hadoop" directory which is present in HDFS directory to the directory "data" which is present in your local directory
#
hadoop fs -copyToLocal hadoop/purchases.txt /home/sam/data (this is your local disk)

22. cp is used to copy files between directories present in HDFS
#
hadoop fs -cp /user/hduser/*.txt /user/hduser/hadoop

23. '-get' command can be used alternaively to '-copyToLocal' command
#
hadoop fs -get hadoop/sample.txt /home/sam/Desktop

24. Display last kilobyte of the file "purchases.txt" to stdout.
#
hadoop fs -tail hadoop/purchases.txt

25. Default file permissions are 666 in HDFS, Use '-chmod' command to change permissions of a file
#
hadoop fs -ls hadoop/purchases.txt
sudo -u hdfs hadoop fs -chmod 600 hadoop/purchases.txt

26. Default names of owner and group are hduser,hduser, Use '-chown' to change owner name and group name simultaneously
#
hadoop fs -ls hadoop/purchases.txt
sudo -u hdfs hadoop fs -chown root:root hadoop/purchases.txt

27. Default name of group is hduser, Use '-chgrp' command to change group name
#
hadoop fs -ls hadoop/purchases.txt
sudo -u hdfs hadoop fs -chgrp hduser hadoop/purchases.txt

28. Move a directory from one location to other
#
hadoop fs -mv hadoop apache_hadoop

29. Default replication factor to a file is 3.
Use '-setrep' command to change replication factor of a file
#
hadoop fs -setrep -w 2 apache_hadoop/sample.txt

30. Copy a directory from one node in the cluster to another. Use '-distcp' command to copy,
-overwrite option to overwrite in an existing files -update command to synchronize both directories
#
hadoop fs -distcp hdfs://namenodeA/apache_hadoop hdfs://namenodeB/hadoop

31. Command to make the name node leave safe mode
#
hadoop fs -expunge
sudo -u hdfs dfsadmin -safemode leave

32. List all the hadoop file system shell commands
#
hadoop fs

33. Last but not least, always ask for help!
#
hadoop fs –help

Ex. No 4 **Map Reduce Programming**

Apache Hadoop : Creating Wordcount Java Project with Eclipse

Step 1: Create a JavaProject named as “WordCount”

File > New > Project > Java Project > Next.

"WordCount" as our project name and click "Finish"

Step 2: Getting references to hadoop libraries

We'll get bunch of refereces which refer to Hadoop libraries.

Right click on WordCount project and select "Properties":

Hit "Add External JARs...", then, File System > usr > lib > hadoop :

(1) /usr/local/hadoop/share/common
(2) /usr/local/hadoop/share/hdfs/
(3) /usr/local/hadoop/share/mapreduce
(4) /usr/local/hadoop/share/tools/
(5) /usr/local/hadoop/share/yarn

Step 3: Creating class files

Right click on source, New > Class:

Class Name as WordCount ad click finish.

Step 4: Copy the Code

Get the code from http://wiki.apache.org/hadoop/WordCount:

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

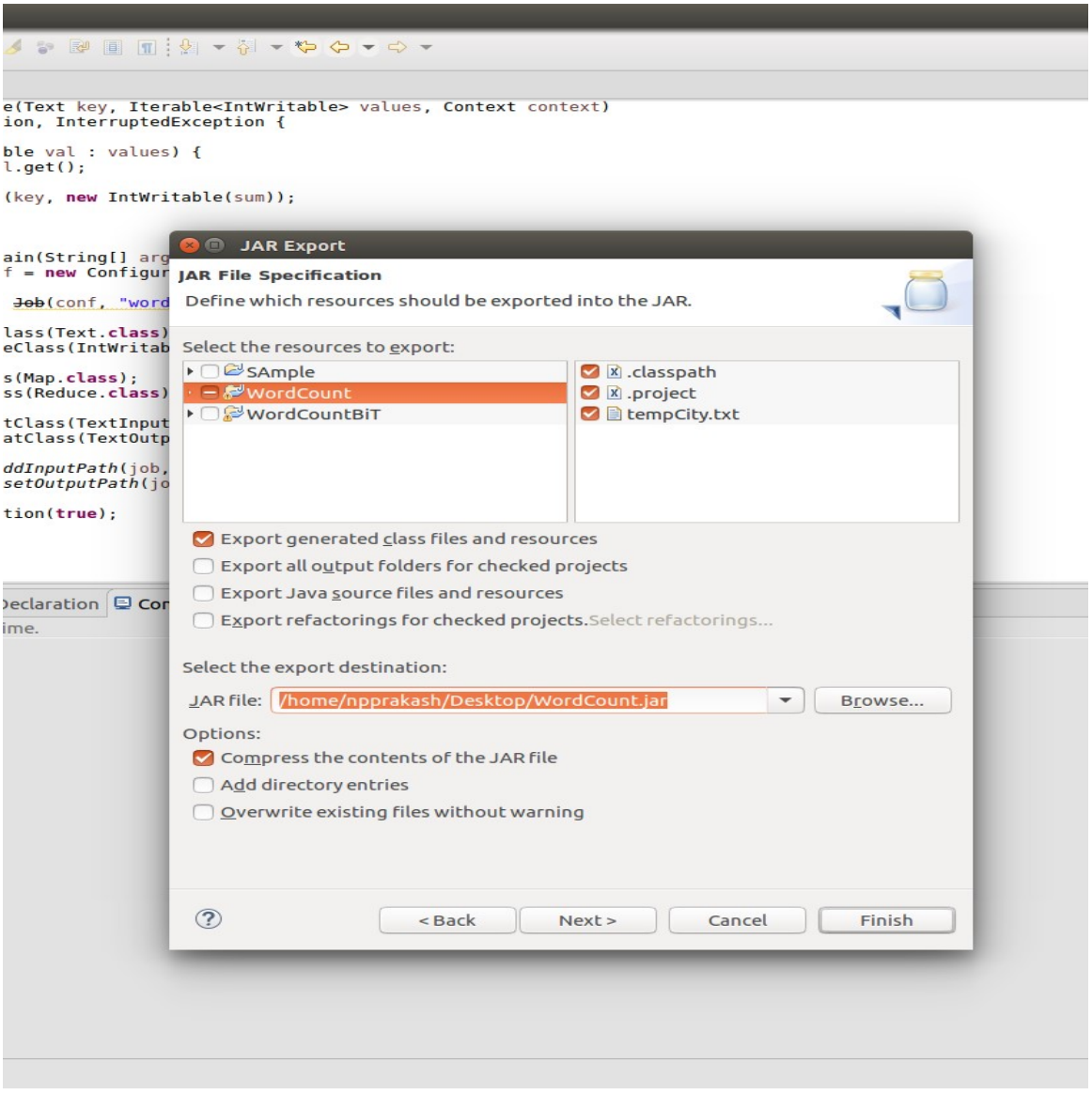
```
public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
          word.set(tokenizer.nextToken());
          context.write(word, one);
       }
   }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
     throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
          sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
 }

 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
       Job job = new Job(conf, "wordcount");
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setMapperClass(Map.class);
```

```
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
  }
}
```

Step 5: Exporting the Jar

Now we want to export it as a jar.

Right click on WordCount project and select "Export...":



Step 6: Create a text file named as wordcount.txt

hduser@npprakashhp:~$ cat /home/npprakash/wordcount.txt

Step 7: Move the local file into HDFS.

hduser@npprakashhp:~$ hdfs dfs -moveFromLocal /home/npprakash/Desktop/wordcount.txt /usr/local/hadoop/input/

Step 8: Run the jar file.

hduser@npprakashhp:~$ hadoop jar /home/npprakash/Desktop/WordCount.jar WordCount /usr/local/hadoop/input/wordcount.txt output

Step 9: List the files into the HDFS ouput directory.

hduser@npprakashhp:~$ hadoop fs -ls output/

Step 10 : Check the results

hduser@npprakashhp:~$ hadoop fs -cat output1/part-r-00000

To execute with package name:

hadoop jar /home/karthi/Desktop/WordCount_example.jar **wordcount.**WordCount /input/wc.txt output

where wordcount is the package name

Sample input file : wc.txt

this is a sample text

this is cat dog

Breaking down MapReduce Concepts

1: Data types

Hadoop uses a wrapper around java datatypes. For eg: IntWritable is used for declaring an integer instead of int.

Similarly, Conversion needed from Java to Hadoop:

* byte -> ByteWritable
* short -> ShortWritable
* long -> LongWritable
* float -> FloatWritable
* double -> DoubleWritable
* boolean -> BooleanWritable
* string -> Text

Explore more dataTypes : https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/io/
Also have a look at : http://stackoverflow.com/questions/19441055/why-does-hadoop-need-classes-like-text-or-intwritable-instead-of-string-or-integ

Using these hadoop datatypes are easy. Let us look at an example.

1* To create a IntWritable from a int.
Method: public IntWritable(int value)
Eg: IntWritable var1 = new IntWritable(5);

2* To set value to a declared IntWritable variable.
Method: public void set (int value)
Eg: IntWritable var1 = new IntWritable();
var1.set(5);

3* To get the value stored in the IntWritable variable.
Method: public int get()
Eg: var1.get(); // output will be 5;

Explore more methods at:
https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/io/IntWritable.html

Example 2: Finding the average and total salary of male and female employees

```
package temp;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class  salary{
/*
 * data schema(tab separated) :-100 Steven King M SKING 515.123.4567
 * 17-JUN-03 AD_PRES 25798.9 90 Sex at position 4th and salary at 9th
 * position
 */
public static class MapperClass extends  Mapper<LongWritable, Text, Text, FloatWritable> {
 public void map(LongWritable key, Text empRecord, Context con)
   throws IOException, InterruptedException {
  String[] word = empRecord.toString().split("\\t");
  String sex = word[3];
  try {
   Float salary = Float.parseFloat(word[8]);
   con.write(new Text(sex), new FloatWritable(salary));
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}

public static class ReducerClass extends   Reducer<Text, FloatWritable, Text, Text> {
 public void reduce(Text key, Iterable<FloatWritable> valueList,
```

```
  Context con) throws IOException, InterruptedException {
 try {
  Float total = (float) 0;
  int count = 0;
  for (FloatWritable var : valueList) {
   total += var.get();
   System.out.println("reducer " + var.get());
   count++;
  }
  Float avg = (Float) total / count;
  String out = "Total: " + total + " :: " + "Average: " + avg;
  con.write(key, new Text(out));
 } catch (Exception e) {
  e.printStackTrace();
 }
}
}

public static void main(String[] args) {
 Configuration conf = new Configuration();
 try {
  Job job = Job.getInstance(conf, "FindAverageAndTotalSalary");
  job.setJarByClass(salary.class);
  job.setMapperClass(MapperClass.class);
  job.setReducerClass(ReducerClass.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(FloatWritable.class);
  Path pathInput = new Path( "/user/employee_records.txt");
  Path pathOutputDir = new Path("/sal");
  FileInputFormat.addInputPath(job, pathInput);
  FileOutputFormat.setOutputPath(job, pathOutputDir);
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 } catch (IOException e) {
  e.printStackTrace();
 } catch (ClassNotFoundException e) {
```

```
  e.printStackTrace();
 } catch (InterruptedException e) {
  e.printStackTrace();
 }

}
}
```

Input Data File employee_records.txt

```
101  Neena  Kochhar  M  NKOCHHAR 515.123.4568 21-SEP-05  AD_VP
     18274.22        100 90
102  Lex  De Haan  M  LDEHAAN  515.123.4569 13-JAN-01  AD_VP
     18274.22        100 90
103  Alexander  Hunold  M  AHUNOLD  590.423.4567 03-JAN-06  IT_PROG
     9000  102  60
104  Bruce  Ernst  F  BERNST  590.423.4568 21-MAY-07  IT_PROG  6000
     103  60
105  David  Austin M  DAUSTIN  590.423.4569 25-JUN-05  IT_PROG  4800
     103  60
106  Valli  Pataballa  M  VPATABAL  590.423.4560 05-FEB-06  IT_PROG
     4800  103  60
107  Diana  Lorentz  M  DLORENTZ  590.423.5567 07-FEB-07  IT_PROG
     4200  103  60
```

2: Understanding the mapreduce components

Note: Start reading from the main method for better understanding.

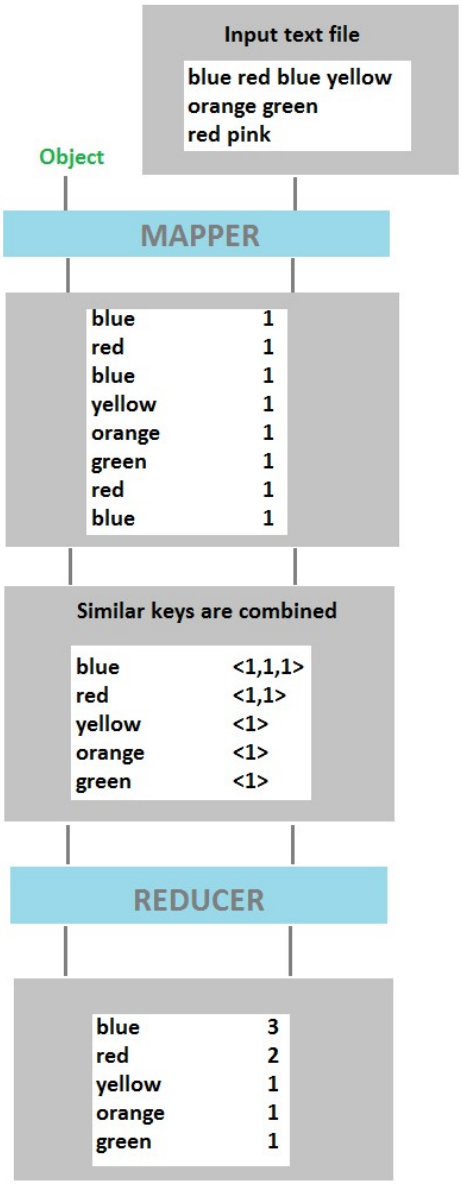
```
1  import java.io.IOException;
2  import java.util.StringTokenizer;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13
14 /* The following class has the implementation of Mapper,Combiner,Reducer and the main
15 method. For simplicity, we will use the same class for Reducer and Combiner.
16 */
17 public class WordCount {
18
19 /* Extend the custom class with Mapper<IpKey, IpValue, OpKey, OpValue >
20    IpKey ->InputKey to mapper class (Object in most cases)
21    IpValue ->InputValue to mapper class (Text in most cases)
22    OpKey ->OutputKey from mapper class (Will be the InputKey for Reducer)
23    OpValue ->OutputValue from mapper class(Will be InputValue for Reducer)
24 */
25   public static class TokenizerMapper
26        extends Mapper<Object, Text, Text, IntWritable>{
27
28     //Create new IntWritable object
29     private final static IntWritable one = new IntWritable(1);
30
31     //Create new Text object
32     private Text word = new Text();
33
34     /*User logic is placed inside map function.
35       Output (Key,value) pair is written to context in every stage,
```

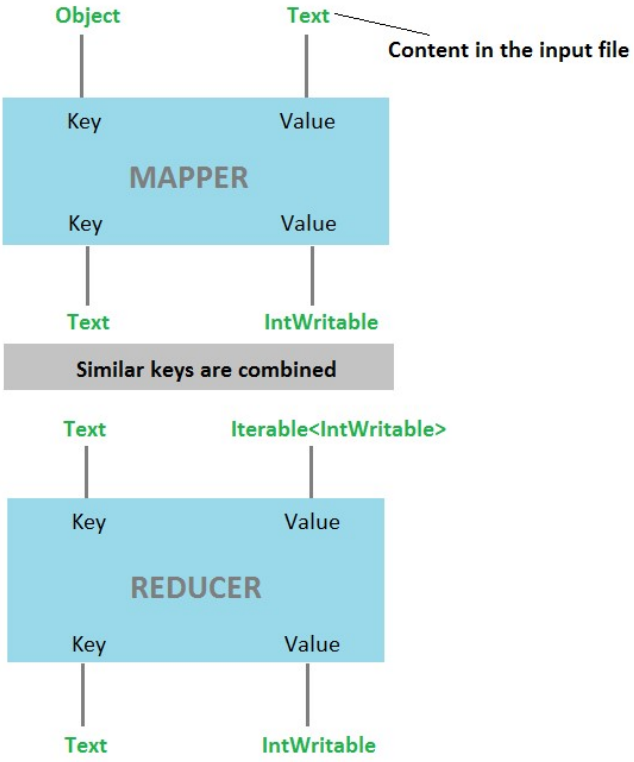
```
36      context facilitates the data movement from stage to another stage.
37      Eg: from mapper class to reducer class
38     */
39     public void map(Object key, Text value, Context context)
40         throws IOException, InterruptedException {
41
42     /*Splits strings based on a token. Words separated by spaces in a sentence are divided into
43  an array of words here. Iterate through the array of words and assign value '1'to every word.
44  Here word is the key and 1 is the value for the corresponding key.
45      */
46      StringTokenizer itr = new StringTokenizer(value.toString());
47      while (itr.hasMoreTokens()) {
48        word.set(itr.nextToken());
49
50        // context.write(key,value)
51        context.write(word, one);
52      }
53     }
54   }
55
56
57  /* Extend the custom class with Reducer<IpKey, IpValue, OpKey, OpValue >
58    IpKey ->InputKey to reducer class (OutputKey from mapper class)
59    IpValue ->InputValue to reducer class (OuputValue from mapper class)
60    OpKey ->OutputKey from reducer class (Will be written to file)
61    OpValue ->OutputValue from reducer class(Will be written to file)
62  */
63    public static class IntSumReducer
64        extends Reducer<Text,IntWritable,Text,IntWritable>{
65     private IntWritable result = new IntWritable();
66
67     /*User logic is placed inside reducefunction.
68       Output (Key,value) pair is written to context.
69       DataType of key in reduce is Text because outputKey from mapper is a Text. So the
70  dataType must be changed according to the mapper output.
```

```
71      DataType of Value is Iterable<IntWritable>because outputValue from mapper is a
72  IntWritable. As a particular key can have multiple values,
73  we adopt Iterable<InputValue DataType>as the standard.
74     */
75     public void reduce(Text key, Iterable<IntWritable>values,
76                        Context context
77                        ) throws IOException, InterruptedException {
78       int sum = 0;
79
80       //For every key, sum the value
81       for (IntWritable val : values) {
82         sum += val.get();
83       }
84       result.set(sum);
85       context.write(key, result);
86     }
87   }
88  public static void main(String[] args) throws Exception {
89     //Create an object of org.apache.hadoop.conf.Configuration
90     Configuration conf = new Configuration();
91
92     /* Create an object of org.apache.hadoop.mapreduce.Job.
93        Creates a new Jobwith a given jobName. The Job makes a copy of theConfiguration so
94  that any necessary internal modifications do not reflect on the incoming parameter.
95     */
96     Job job = Job.getInstance(conf, "word count");
97
98     // Set the class name in which main method resides.
99     job.setJarByClass(WordCount.class);
100    // Set the mapper class name
101    job.setMapperClass(TokenizerMapper.class);
102
103    //Set the combiner class name
104    job.setCombinerClass(IntSumReducer.class);
105
```

```
106     //Set the reducer class name
107     job.setReducerClass(IntSumReducer.class);
108
109     /*Output from reducer is the final result and Output is Key,Value pair.
110       Inform Hadoop about the dataType of Key and Value from reducer
111     */
112     job.setOutputKeyClass(Text.class);
113     job.setOutputValueClass(IntWritable.class);
114
115     /* The following command is used to run a mapreduce program.
116        hadoop jar wc.jar WordCount input_dir output_dir
117        hadoop jar <jar name><pgm class name><input path><output path>
118        The input and output path to the dataset are set as below
119     */
120     FileInputFormat.addInputPath(job, new Path(args[0]));
121     FileOutputFormat.setOutputPath(job, new Path(args[1]));
122
123     /* Wait for the program execution to complete.
124        Exit the program after job execution.
125     */
126     System.exit(job.waitForCompletion(true) ? 0 : 1);
127   }
128 }
129
130
131
132
133
134
135
136
137
```

3: Understanding the mapreduce workflow with an example


Input text file
blue red blue yellow
orange green
red pink
Object
MAPPER
blue 1
red 1
blue 1
yellow 1
orange 1
green 1
red 1
blue 1
Similar keys are combined
blue <1,1,1>
red <1,1>
yellow <1>
orange <1>
green <1>
REDUCER
blue 3
red 2
yellow 1
orange 1
green 1


Object
Text
Content in the input file
Key
Value
MAPPER
Key
Value
Text
IntWritable
Similar keys are combined
Text
Iterable<IntWritable>
Key
Value
REDUCER
Key
Value
Text
IntWritable