University of Glasgow | School of Computing Science

CSC3005 Laboratory/Tutorial 8: Big Data Hadoop Streaming

Acknowledgement: Jacob Abraham

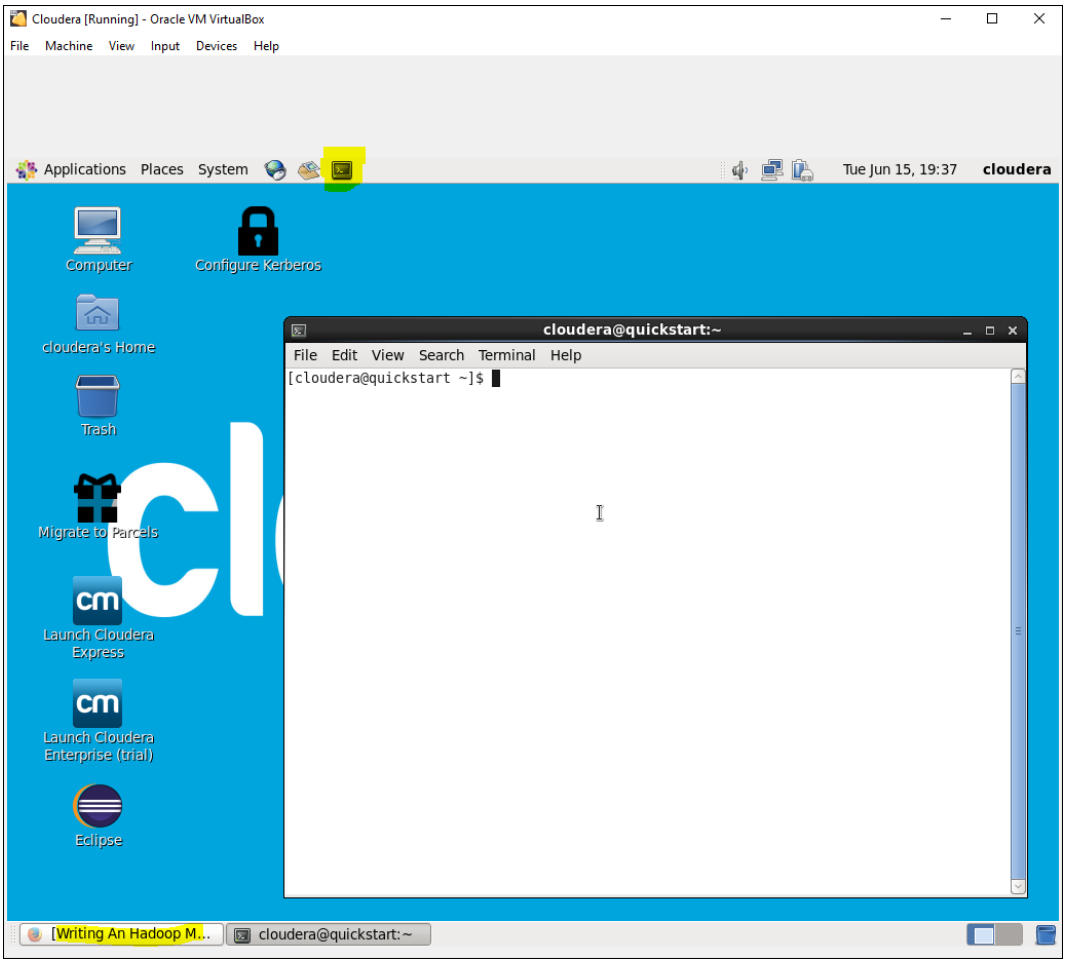
1. Environment

Oracle VM VirtualBox, Cloudera QuickStart VM, Hadoop, Hadoop-streaming API, HDFS, Python, PySpark, and YARN

2. Create required text files and python scripts

1. **Open a terminal window in Cloudera**
   a. Minimize all web browser windows.
   b. Click the Terminal icon from top menu.

Screen shot:

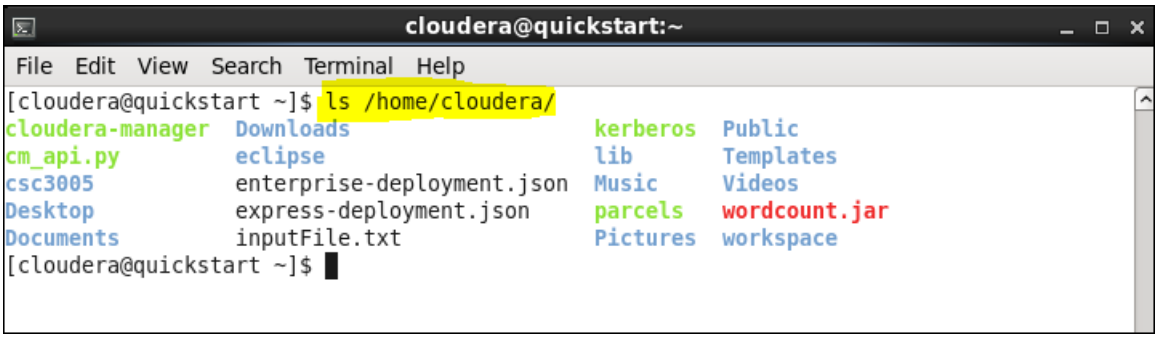


2. **Check the contents of the default folder**
   [cloudera@quickstart ~]$ ls

   Or

   [cloudera@quickstart ~]$ ls /home/cloudera/

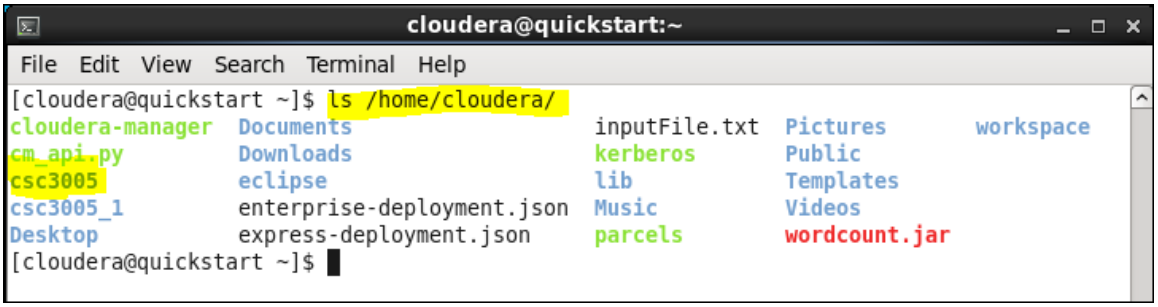
   Screen shot:



3. **Create a directory "csc3005" to hold python scripts and simple text files**
   [cloudera@quickstart ~]$ mkdir /home/cloudera/csc3005

4. **Check the "csc3005" directory is created successfully**
   [cloudera@quickstart ~]$ ls /home/cloudera/

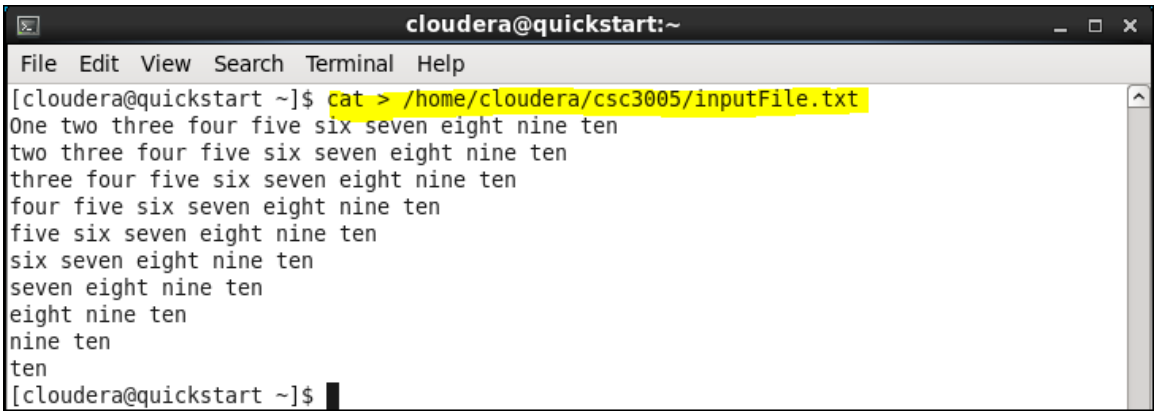
   Screen shot:



5. **Create a text file "inputFile.txt" to hold words and save it under the directory "/home/cloudera/csc3005"**
   Note:
   a. After typing the command "cat > /home/cloudera/csc3005/inputFile.txt", copy (Ctrl + c) the contents in Appendix 1 and paste (do not use "Ctrl + v", right click and select Paste from the context menu)
   b. To end typing and save, press "Ctrl + d"

   [cloudera@quickstart ~]$ cat > /home/cloudera/csc3005/inputFile.txt

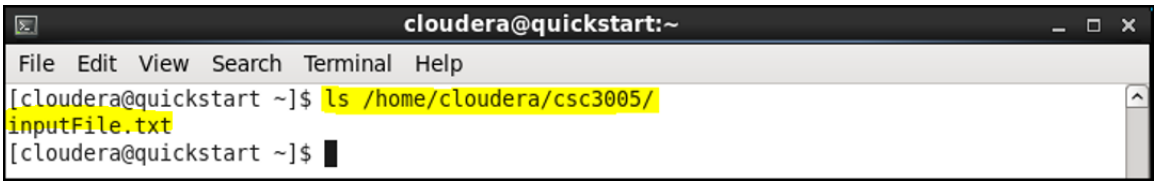
Screen shot:



6. **Check the file "inputFile.txt" is successfully created**

   [cloudera@quickstart ~]$ ls /home/cloudera/csc3005/

Screen shot:



7. **Create a mapper python script file "mapper.py"**

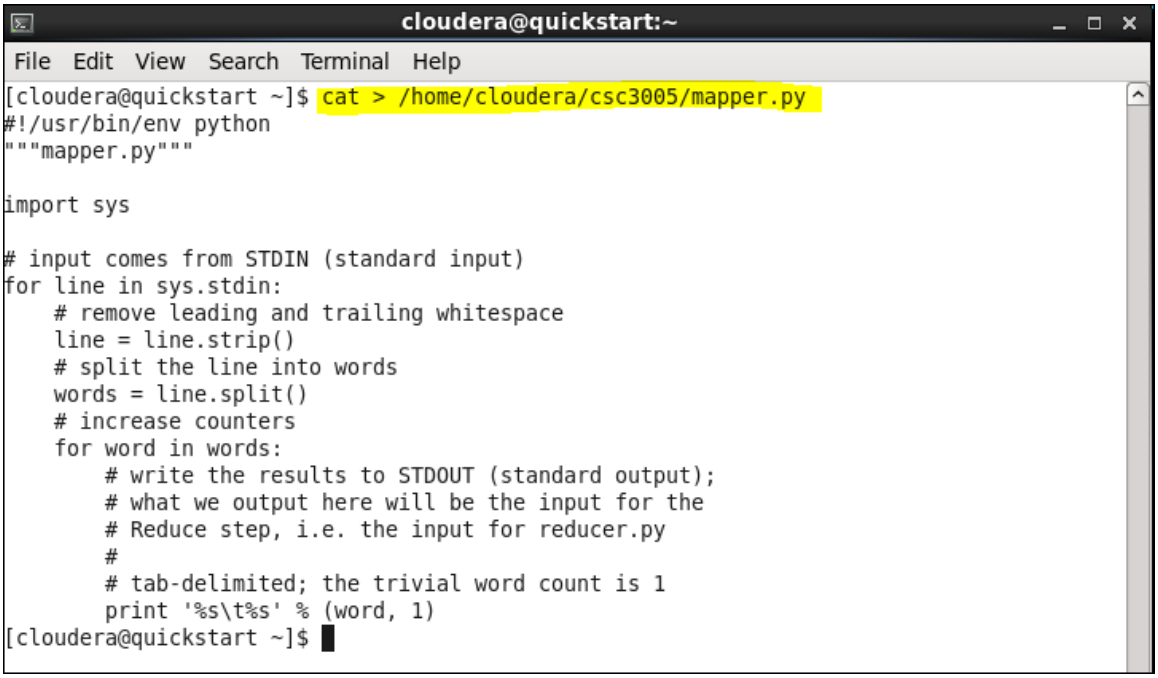
   Note:

   a. After typing the command "cat > /home/cloudera/csc3005/mapper.py", copy (Ctrl + c) the contents in Appendix 2 and paste (do not use "Ctrl + v", right click and select Paste from the context menu)

   b. To end typing and save, press "Ctrl + d"

   [cloudera@quickstart ~]$ cat > /home/cloudera/csc3005/mapper.py

Screen shot:



8. **Create a reducer python script file "reducer.py"**

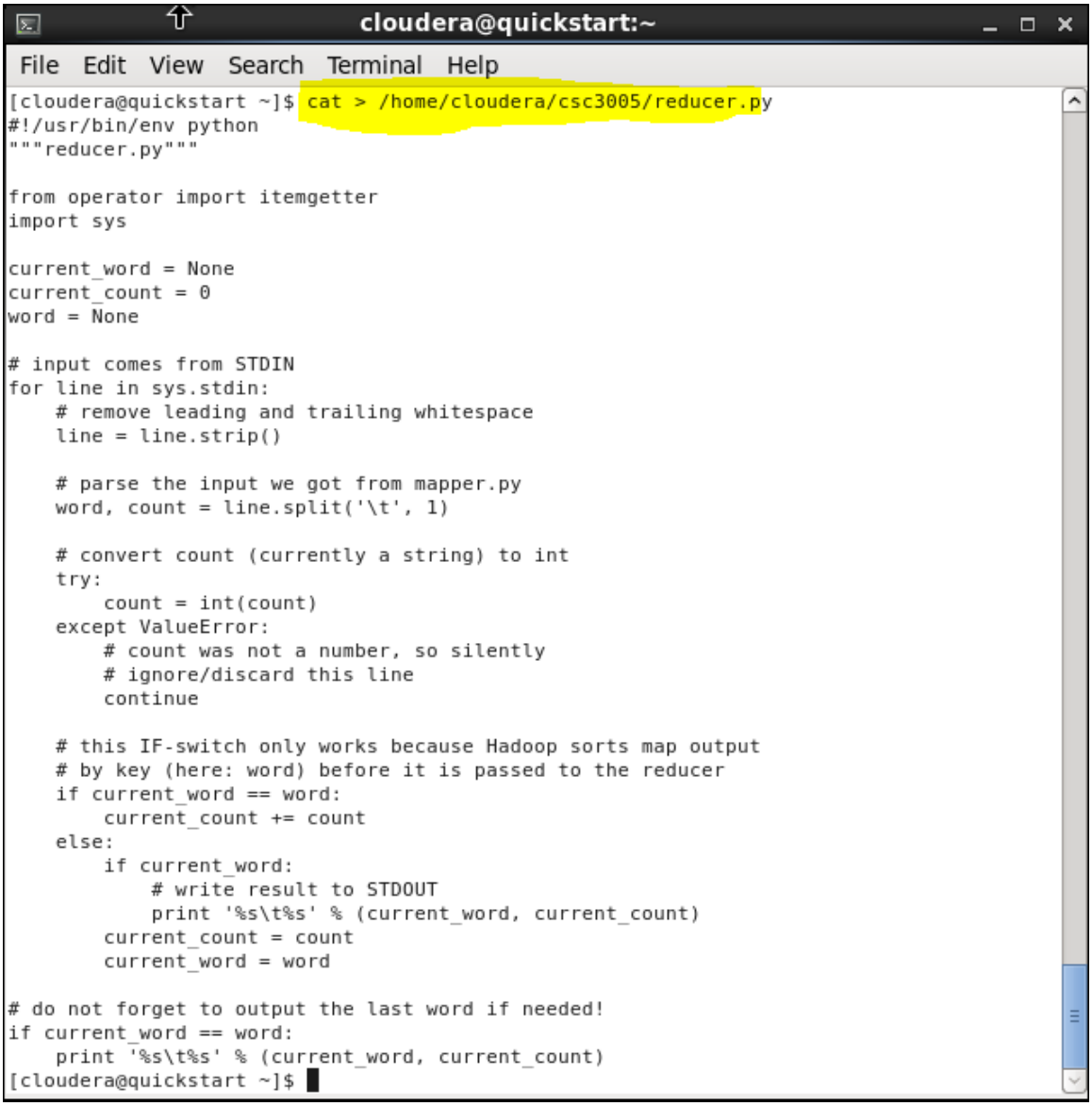
   Note:

   a. After typing the command "cat > /home/cloudera/csc3005/reducer.py", copy (Ctrl + c) the contents in Appendix 3 and paste (do not use "Ctrl + v", right click and select Paste from the context menu)

   b. To end typing and save, press "Ctrl + d"

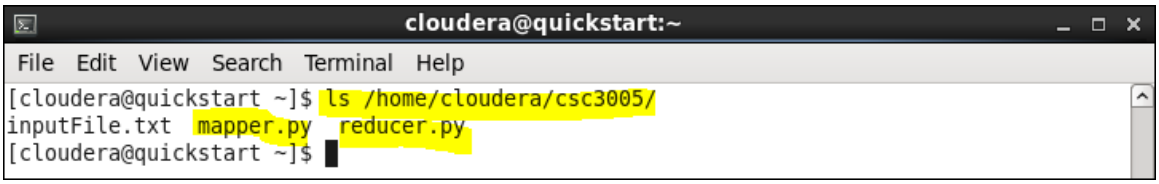
   [cloudera@quickstart ~]$ cat > /home/cloudera/csc3005/reducer.py

Screen shot:


cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ cat > /home/cloudera/csc3005/reducer.py
#!/usr/bin/env python
"""reducer.py"""
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()
# parse the input we got from mapper.py
word, count = line.split('\t', 1)
# convert count (currently a string) to int
try:
count = int(count)
except ValueError:
count was not a number, so silently
ignore/discard this line
continue
# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
current_count += count
else:
if current_word:
# write result to STDOUT
print '%s\t%s' % (current_word, current_count)
current_count = count
current_word = word
# do not forget to output the last word if needed!
if current_word == word:
print '%s\t%s' % (current_word, current_count)
[cloudera@quickstart ~]$

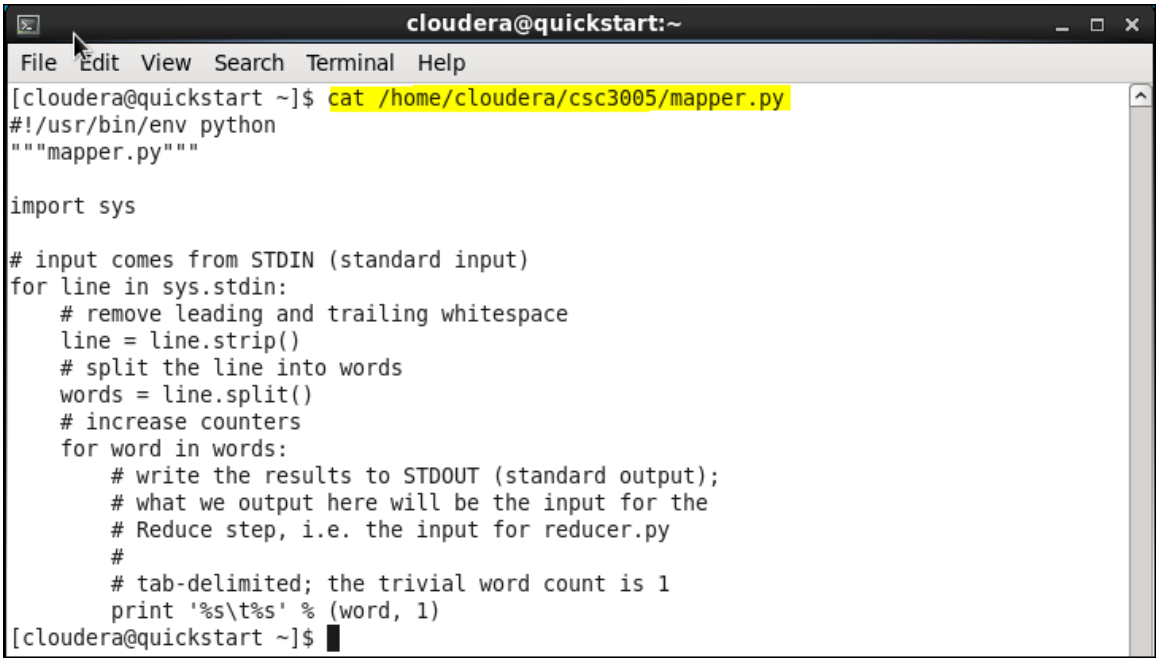
9. **Check both "mapper.py" and "reducer.py" are successfully created**
   [cloudera@quickstart ~]$ ls /home/cloudera/csc3005/

Screen shot:



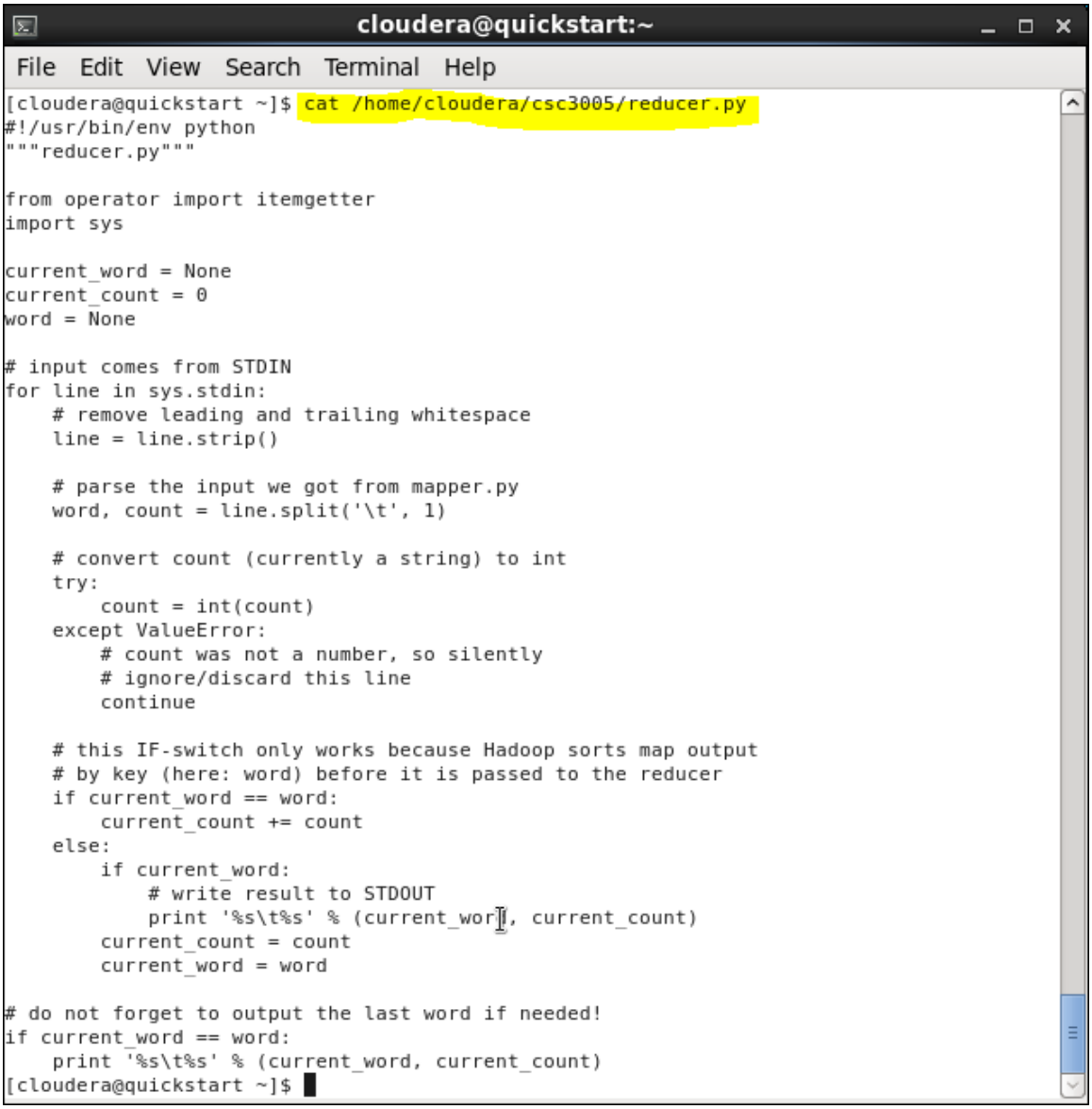
10. **Check the contents of "mapper.py"**
    [cloudera@quickstart ~]$ cat /home/cloudera/csc3005/mapper.py

Screen shot:



11. **Check the contents of "reducer.py"**
    [cloudera@quickstart ~]$ cat /home/cloudera/csc3005/reducer.py

Screen shot:

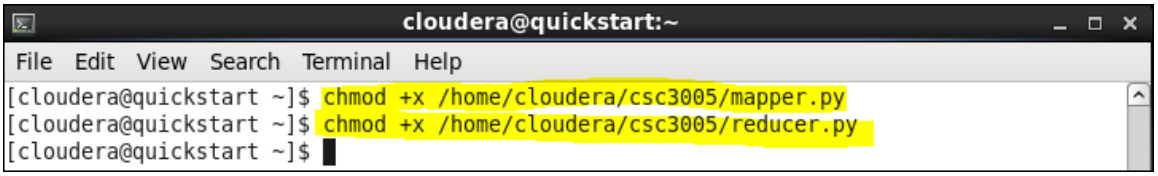


12. **Assign execution permission to "mapper.py" and "reducer.py"**

[cloudera@quickstart ~]$ chmod +x /home/cloudera/csc3005/mapper.py

[cloudera@quickstart ~]$ chmod +x /home/cloudera/csc3005/reducer.py

Screen shot:



3. Test mapper and reducer without hadoop-mapreduce

1. **Test the mapper**

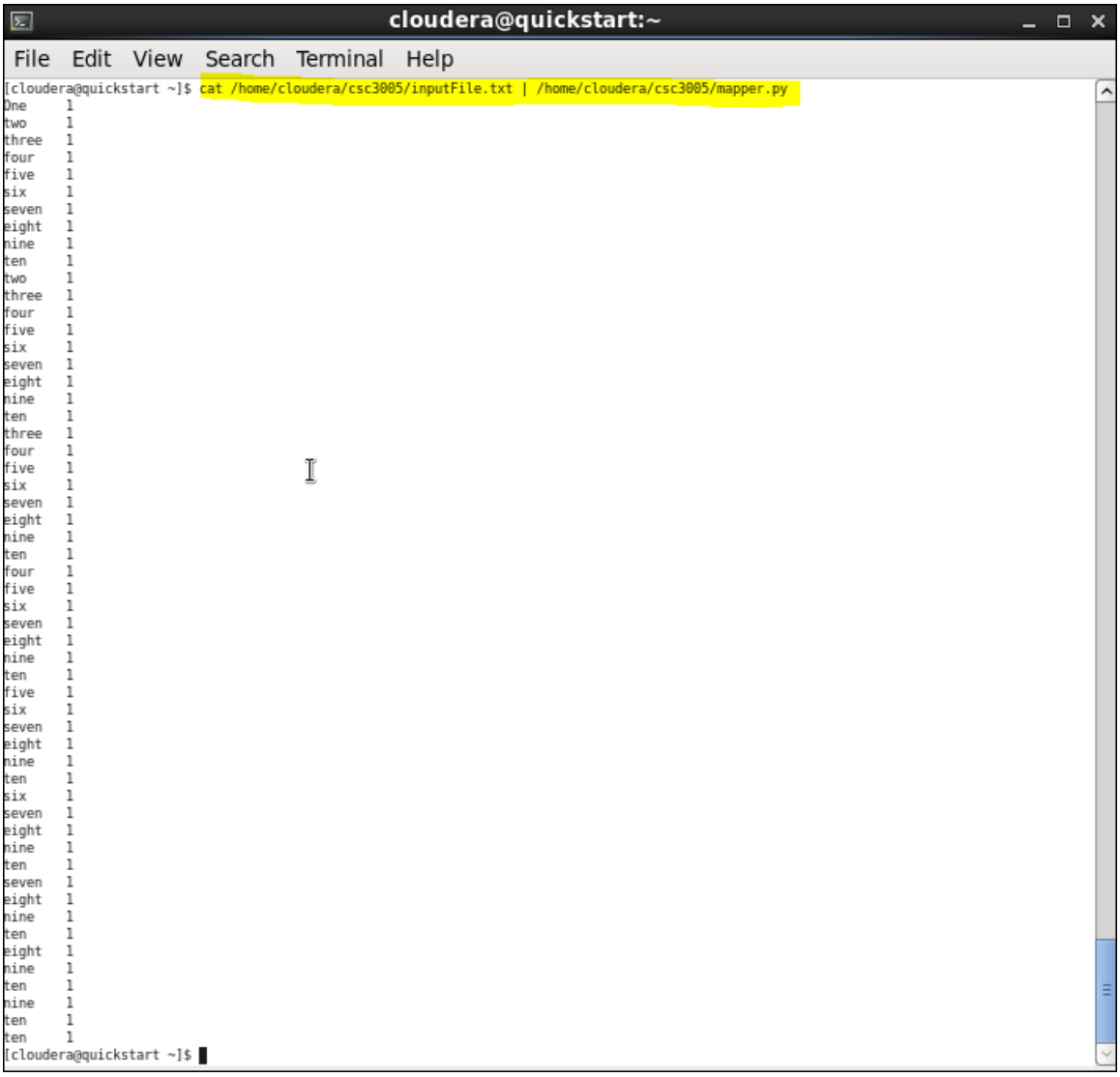
   Note:

   a. The text in "inputFile.txt" is fed (pipped through "|") to the mapper

   b. The mapper will read data from STDIN, split it into words and output a list of lines mapping words to their (intermediate) counts to STDOUT. The Map script will not compute an (intermediate) sum of a word's occurrences though. Instead, it will output <word> 1 tuples immediately – even though a specific word might occur multiple times in the input.

   [cloudera@quickstart ~]$ cat /home/cloudera/csc3005/inputFile.txt \
   > | /home/cloudera/csc3005/mapper.py

   Screen shot:



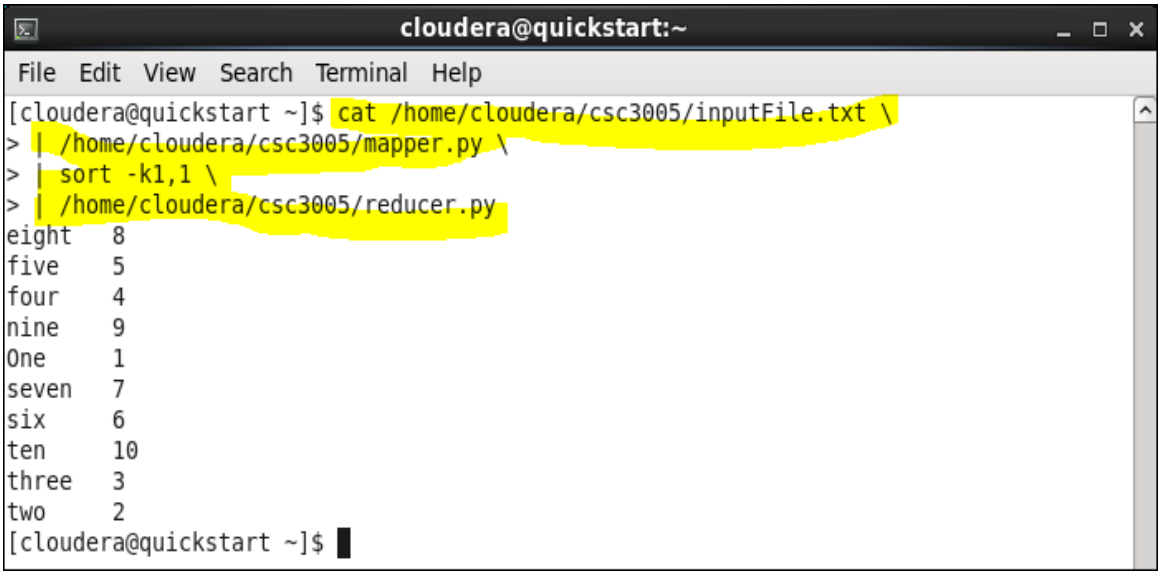
2. **Test the mapper and reducer together**

Note:

1. The text in "inputFile.txt" is fed (pipped through "|") to the mapper
2. The mapper will read data from STDIN, split it into words and output a list of lines mapping words to their (intermediate) counts to STDOUT. The Map script will not compute an (intermediate) sum of a word's occurrences though. Instead, it will output <word> 1 tuples immediately – even though a specific word might occur multiple times in the input.
3. The reducer reads the data (<word> 1 tuples output from mapper) from STDIN and outputs the final sum count sorted by the words.

```
[cloudera@quickstart ~]$ cat /home/cloudera/csc3005/inputFile.txt \
> | /home/cloudera/csc3005/mapper.py \
> | sort -k1,1 \
> | /home/cloudera/csc3005/reducer.py
```

Screen shot:

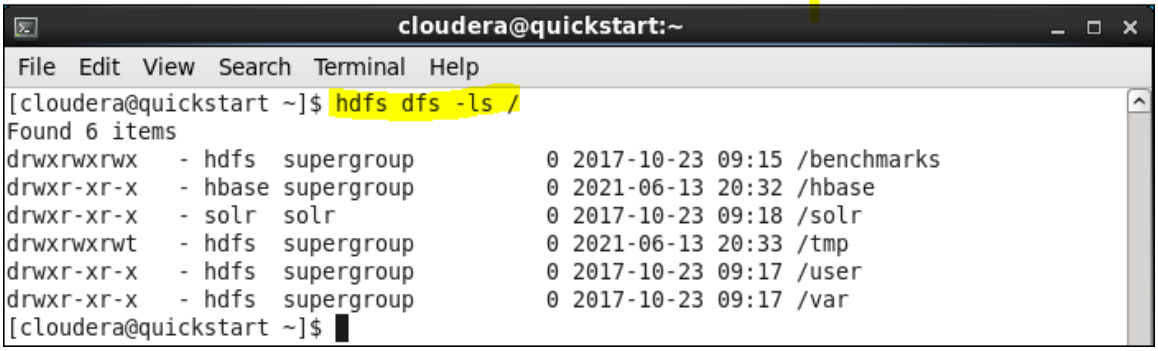


4. Test mapper and reducer with hadoop-mapreduce

1. **Check the contents of the default hdfs directory**

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
```

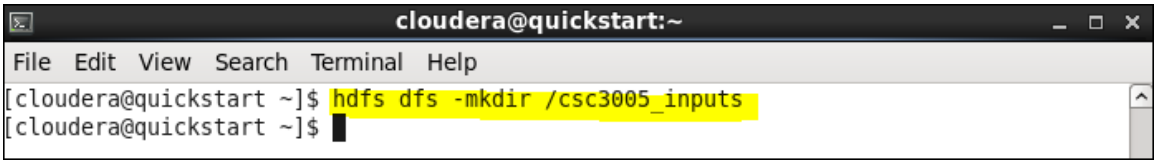
Screen shot:



2. **Create a new directory "csc3005_inputs" in hdfs to store "inputFile.txt" which is in local (non-hdfs) directory "/home/cloudera/csc3005"**

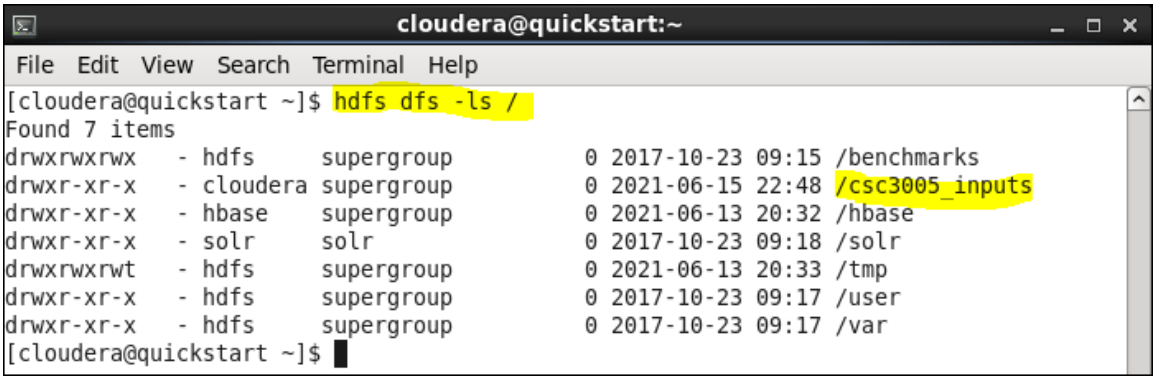
   [cloudera@quickstart ~]$ hdfs dfs -mkdir /csc3005_inputs

   Screen shot:



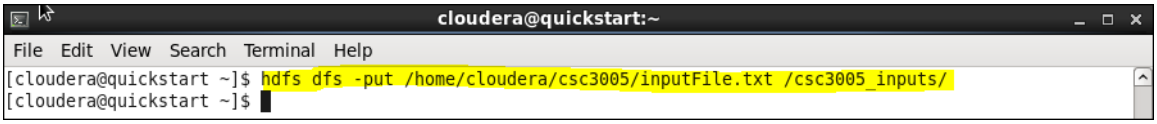
3. **Verify that the directory "" is created in hdfs**
   [cloudera@quickstart ~]$ hdfs dfs -ls /

   Screen shot:



4. **Copy "InputFile.txt" from local directory "/home/cloudera/csc3005" to hdfs directory "/csc3005_inputs"**
   [cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/csc3005/inputFile.txt /csc3005_inputs/

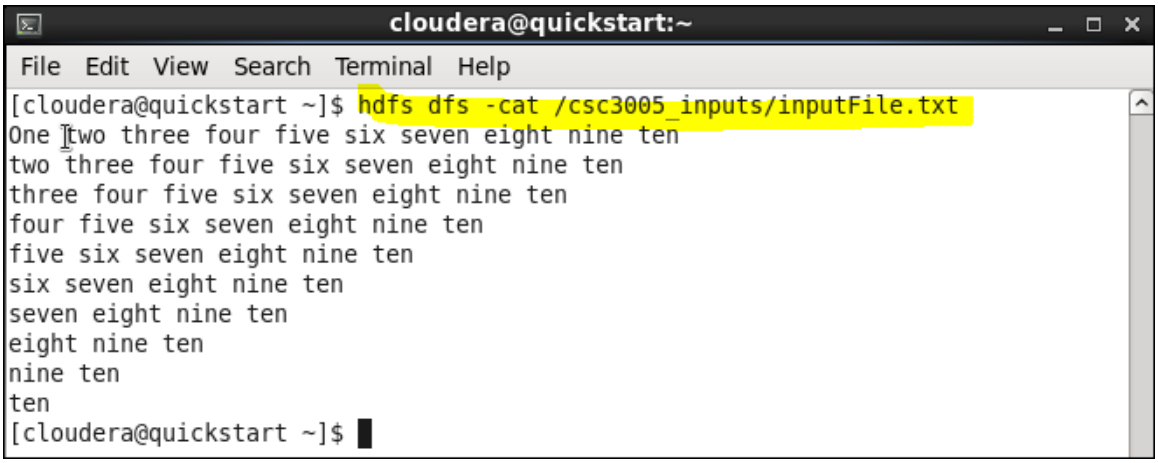
Screen shot:



5. **View "inputFile.txt" that is copied to hdfs directory "/csc3005_inputs"**

   [cloudera@quickstart ~]$ hdfs dfs -cat /home/cloudera/csc3005/inputFile.txt

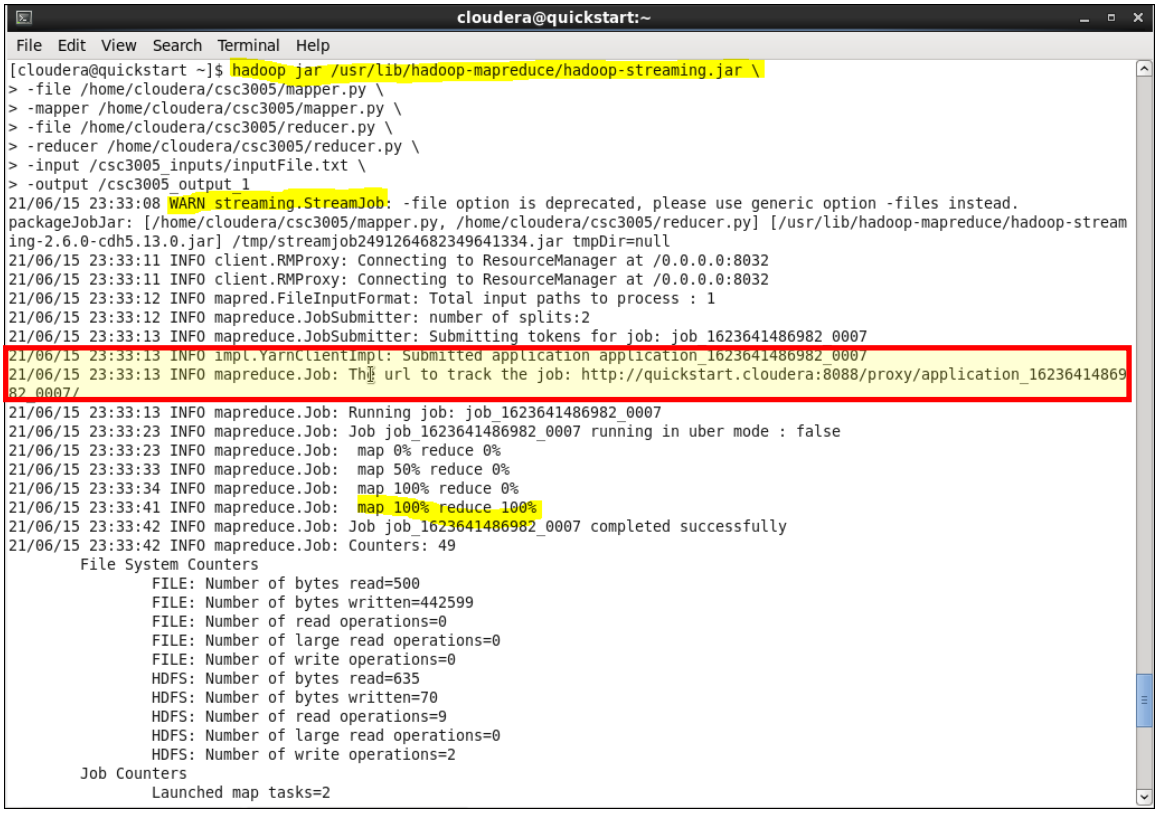
   Screen shot:

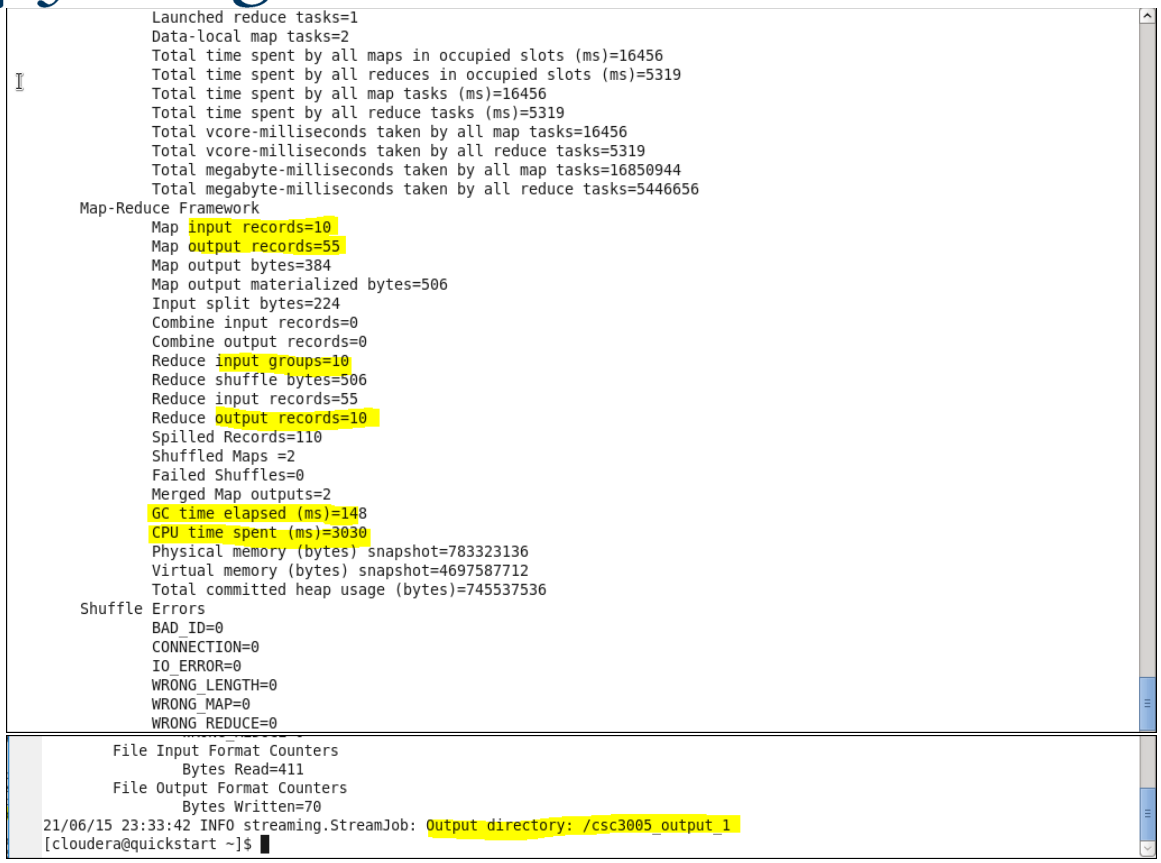


6. **Run the python code (mapper and reducer) on Hadoop as a MapReduce job**

   [cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
   > -file /home/cloudera/csc3005/mapper.py \
   > -mapper /home/cloudera/csc3005/mapper.py \
   > -file /home/cloudera/csc3005/reducer.py \
   > -reducer /home/cloudera/csc3005/reducer.py \
   > -input /csc3005_inputs/inputFile.txt \
   > -output /csc3005_output_1

Screen shot:

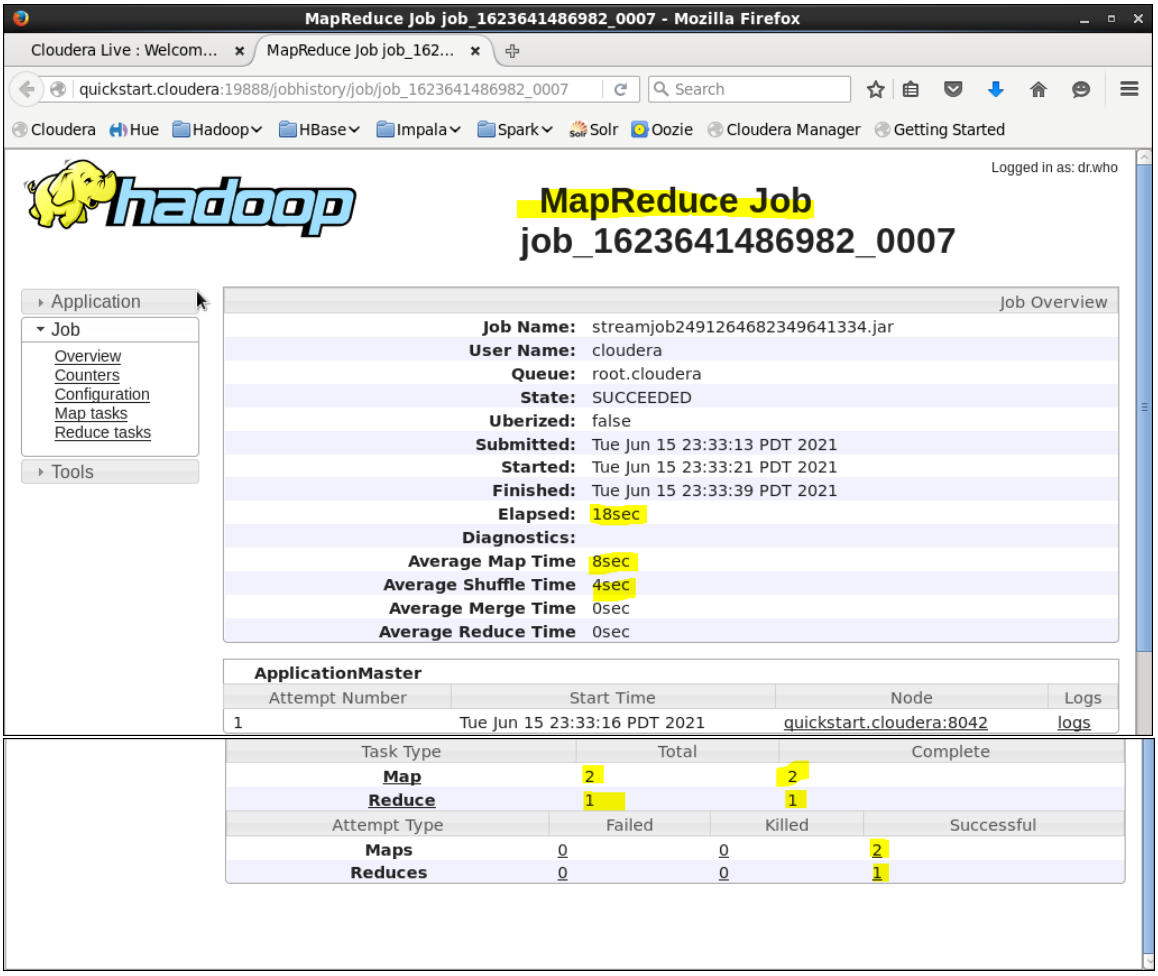

cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
> -file /home/cloudera/csc3005/mapper.py \
> -mapper /home/cloudera/csc3005/mapper.py \
> -file /home/cloudera/csc3005/reducer.py \
> -reducer /home/cloudera/csc3005/reducer.py \
> -input /csc3005_inputs/inputFile.txt \
> -output /csc3005_output_1
21/06/15 23:33:08 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [/home/cloudera/csc3005/mapper.py, /home/cloudera/csc3005/reducer.py] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob2491264682349641334.jar tmpDir=null
21/06/15 23:33:11 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/15 23:33:11 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/15 23:33:12 INFO mapred.FileInputFormat: Total input paths to process : 1
21/06/15 23:33:12 INFO mapreduce.JobSubmitter: number of splits:2
21/06/15 23:33:13 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1623641486982_0007
21/06/15 23:33:13 INFO impl.YarnClientImpl: Submitted application application_1623641486982_0007
21/06/15 23:33:13 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1623641486982_0007/
21/06/15 23:33:13 INFO mapreduce.Job: Running job: job_1623641486982_0007
21/06/15 23:33:23 INFO mapreduce.Job: Job job_1623641486982_0007 running in uber mode : false
21/06/15 23:33:23 INFO mapreduce.Job: map 0% reduce 0%
21/06/15 23:33:33 INFO mapreduce.Job: map 50% reduce 0%
21/06/15 23:33:34 INFO mapreduce.Job: map 100% reduce 0%
21/06/15 23:33:41 INFO mapreduce.Job: map 100% reduce 100%
21/06/15 23:33:42 INFO mapreduce.Job: Job job_1623641486982_0007 completed successfully
21/06/15 23:33:42 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=500
FILE: Number of bytes written=442599
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=635
HDFS: Number of bytes written=70
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=2



Note:

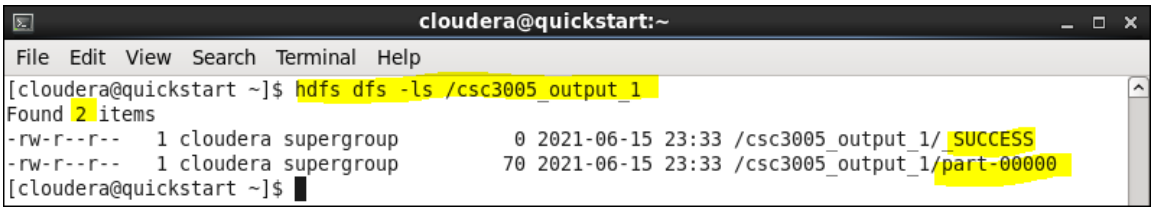
The output provides a basic web interface URL to track the job to view statistics and information http://quickstart.cloudera:19888/jobhistory/job/job_1623641486982_0007

Screen shot:



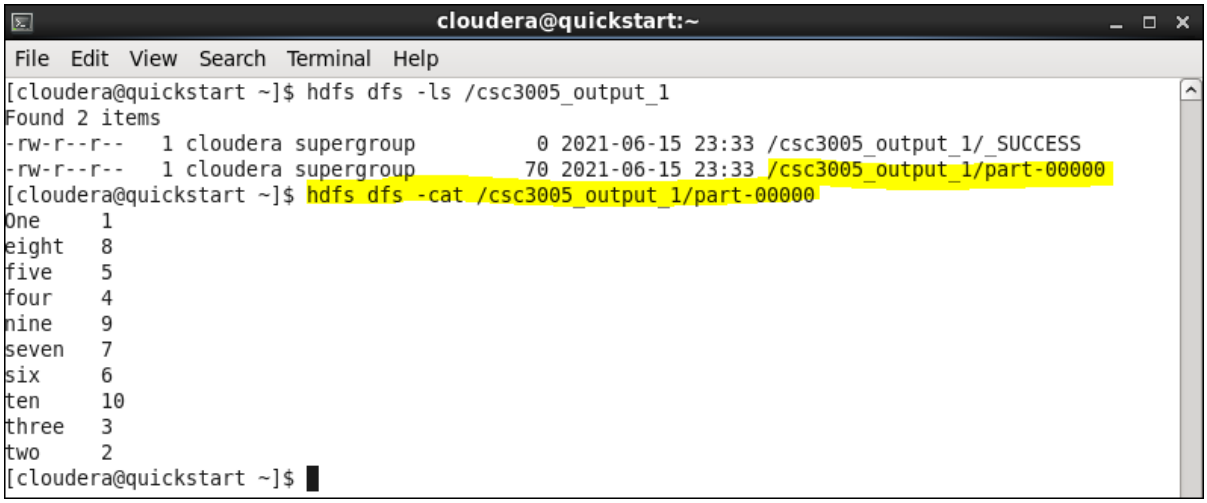
7. **Check the contents of the hdfs folder "/csc3005_output_1" which stores all output files:**
[cloudera@quickstart ~]$ hdfs dfs -ls /csc3005_output_1

Screen shot:



8. **View the result of the MapReduce job from hdfs file "/csc3005_output_1/part-00000":**
[cloudera@quickstart ~]$ hdfs dfs -cat /csc3005_output_1/part-00000

Screen shot:



9. Test advanced-mapper and advanced-reducer on Hadoop as a MapReduce job

Note:

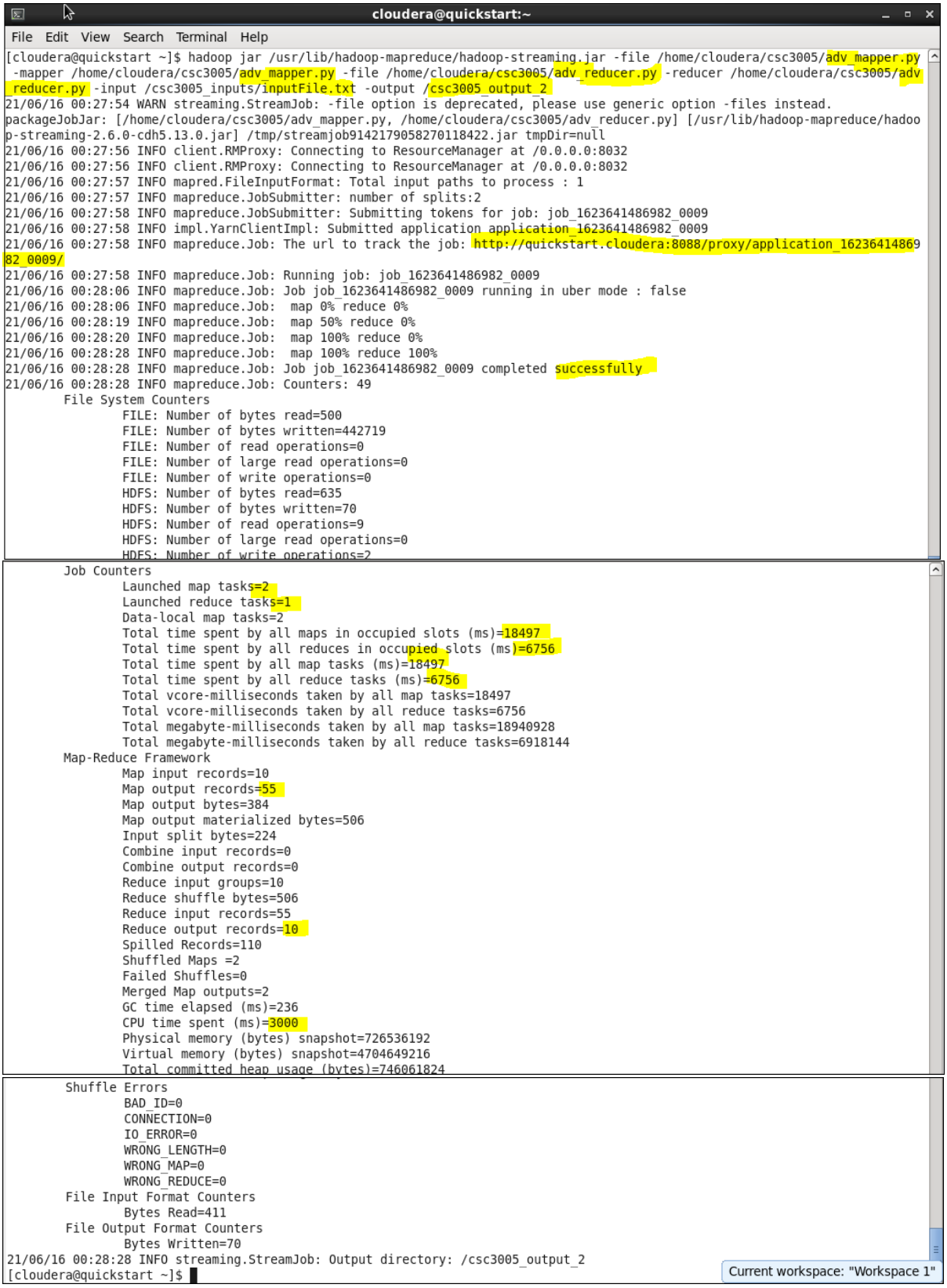
a. The advanced-mapper and advanced-reducer python codes are in Appendix 4 and Appendix 5 respectively

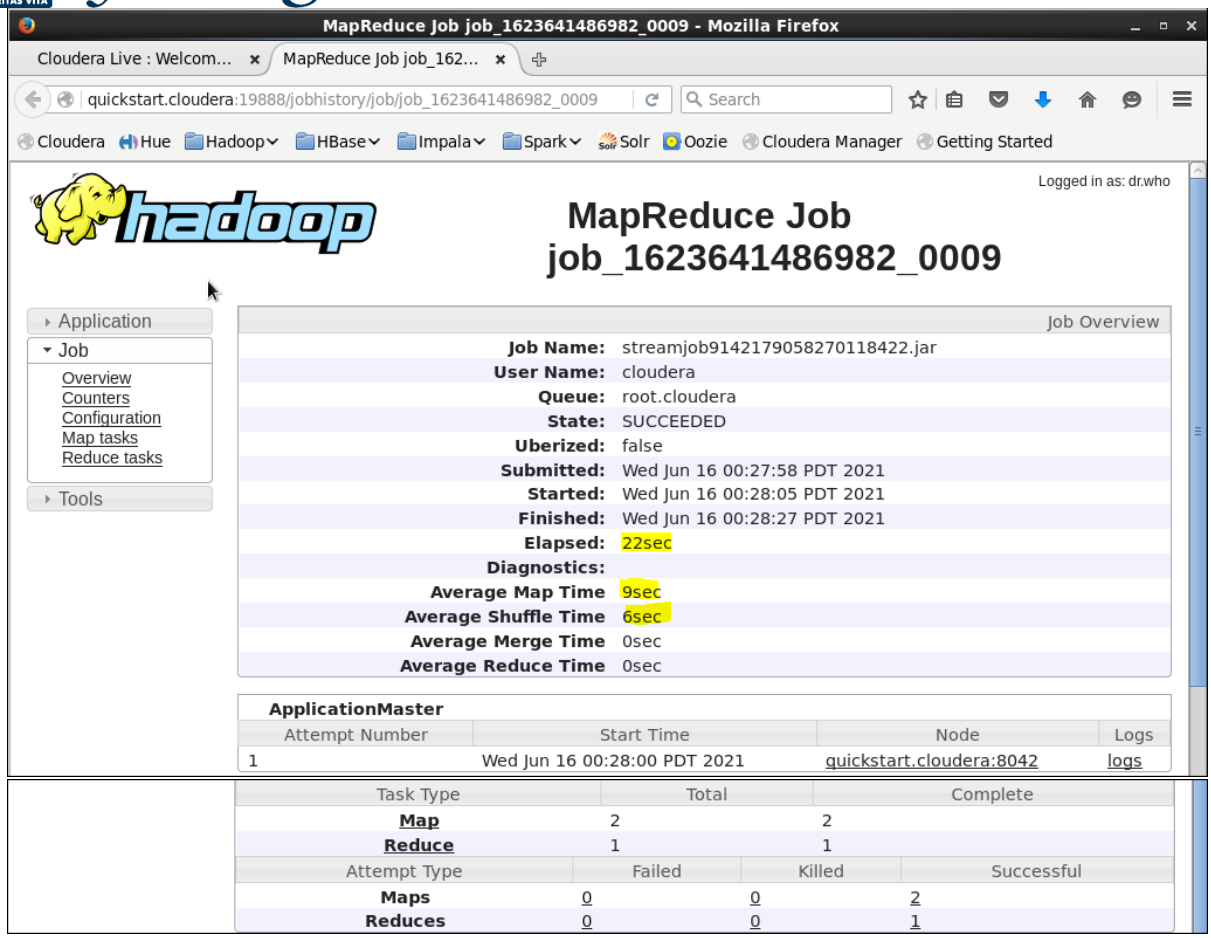
b. The advanced versions use iterators and generators (functions that create iterators, for example with Python's yield statement). These have the advantage that an element of a sequence is not produced until it is needed. This can help a lot in terms of computational expensiveness or memory consumption depending on the task at hand.

c. With the advanced version, naive test command "cat DATA | ./mapper.py | sort -k1,1 | ./reducer.py" will not work correctly anymore because some functionality is intentionally outsourced to Hadoop.

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
> -file /home/cloudera/csc3005/adv_mapper.py \
> -mapper /home/cloudera/csc3005/adv_mapper.py \
> -file /home/cloudera/csc3005/adv_reducer.py \
> -reducer /home/cloudera/csc3005/adv_reducer.py \
> -input /csc3005_inputs/inputFile.txt \
> -output /csc3005_output_2
```

Screen shot:


cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file /home/cloudera/csc3005/adv_mapper.py -mapper /home/cloudera/csc3005/adv_mapper.py -file /home/cloudera/csc3005/adv_reducer.py -reducer /home/cloudera/csc3005/adv_reducer.py -input /csc3005_inputs/inputFile.txt -output /csc3005_output_2
21/06/16 00:27:54 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [/home/cloudera/csc3005/adv_mapper.py, /home/cloudera/csc3005/adv_reducer.py] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob9142179058270118422.jar tmpDir=null
21/06/16 00:27:56 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/16 00:27:56 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/16 00:27:57 INFO mapred.FileInputFormat: Total input paths to process : 1
21/06/16 00:27:57 INFO mapreduce.JobSubmitter: number of splits:2
21/06/16 00:27:58 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1623641486982_0009
21/06/16 00:27:58 INFO impl.YarnClientImpl: Submitted application application_1623641486982_0009
21/06/16 00:27:58 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1623641486982_0009/
21/06/16 00:27:58 INFO mapreduce.Job: Running job: job_1623641486982_0009
21/06/16 00:28:06 INFO mapreduce.Job: Job job_1623641486982_0009 running in uber mode : false
21/06/16 00:28:06 INFO mapreduce.Job: map 0% reduce 0%
21/06/16 00:28:19 INFO mapreduce.Job: map 50% reduce 0%
21/06/16 00:28:20 INFO mapreduce.Job: map 100% reduce 0%
21/06/16 00:28:28 INFO mapreduce.Job: map 100% reduce 100%
21/06/16 00:28:28 INFO mapreduce.Job: Job job_1623641486982_0009 completed successfully
21/06/16 00:28:28 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=500
FILE: Number of bytes written=442719
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=635
HDFS: Number of bytes written=70
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=18497
Total time spent by all reduces in occupied slots (ms)=6756
Total time spent by all map tasks (ms)=18497
Total time spent by all reduce tasks (ms)=6756
Total vcore-milliseconds taken by all map tasks=18497
Total vcore-milliseconds taken by all reduce tasks=6756
Total megabyte-milliseconds taken by all map tasks=18940928
Total megabyte-milliseconds taken by all reduce tasks=6918144
Map-Reduce Framework
Map input records=10
Map output records=55
Map output bytes=384
Map output materialized bytes=506
Input split bytes=224
Combine input records=0
Combine output records=0
Reduce input groups=10
Reduce shuffle bytes=506
Reduce input records=55
Reduce output records=10
Spilled Records=110
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=236
CPU time spent (ms)=3000
Physical memory (bytes) snapshot=726536192
Virtual memory (bytes) snapshot=4704649216
Total committed heap usage (bytes)=746061824
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=411
File Output Format Counters
Bytes Written=70
21/06/16 00:28:28 INFO streaming.StreamJob: Output directory: /csc3005_output_2
[cloudera@quickstart ~]$
Current workspace: "Workspace 1"



5. Test advanced mapper and reducer on larger text files on Hadoop as a MapReduce job

Note:

a. Python codes for advanced mapper and advanced reducer are in Appendix 4 and Appendix 5 respectively.

b. Download example large input data text (Plain Text UTF-8)files from ebooks from Project Gutenberg:

- https://www.gutenberg.org/ebooks/20417
- https://www.gutenberg.org/ebooks/5000
- https://www.gutenberg.org/ebooks/4300

c. Follow steps in Section 4 to complete the word count task.

Appendix 1: inputFile.txt

```
One two three four five six seven eight nine ten

two three four five six seven eight nine ten

three four five six seven eight nine ten

four five six seven eight nine ten

five six seven eight nine ten

six seven eight nine ten

seven eight nine ten

eight nine ten

nine ten

ten
```

6. Appendix 2: mapper.py

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

import sys

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

7. Appendix 3: reducer.py

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

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

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

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

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

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

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

8. Appendix 4: adv_mapper.py

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

import sys

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

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

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

9. Appendix 5: adv_reducer.py

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

from itertools import groupby
from operator import itemgetter
import sys

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

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

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

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