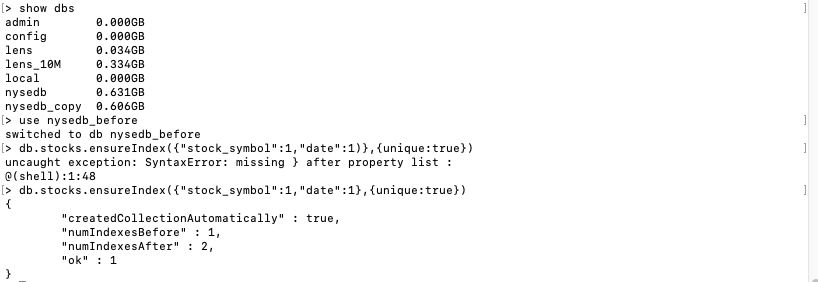
**Assignment 3**

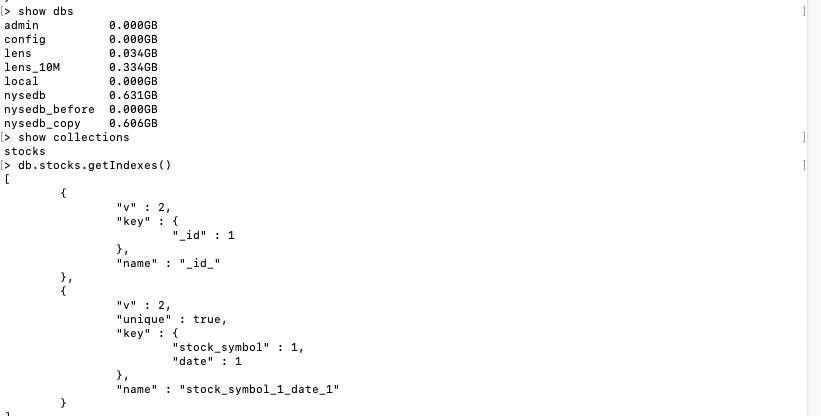
**PART 2 - MongoDB indexing**  
Most of the time, you’ll want to declare your indexes before putting your application into production. This allows indexes to be built incrementally, as the data is inserted. But there are two cases where you might choose to build an index after the fact. The first case occurs when you need to import a lot of data before switching into production. For instance, you might be migrating an application to MongoDB and need to seed the database with user information from a data warehouse. You could create the indexes on your user data in advance, but doing so after you have imported the data will ensure an ideally balanced and compacted index from the start. This will also minimize the net time to build the index. Use the NYSE dataset to declare your indexes before putting your application into production.

1. Create the index for “nysedb\_before” db first

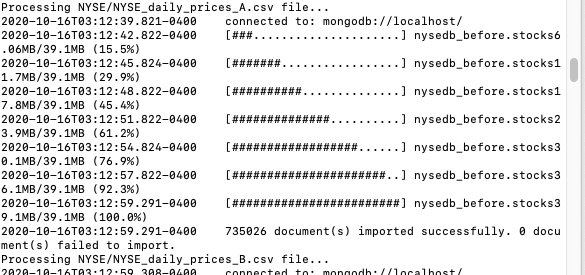


1. and then import all the data to the database.

Before import data



After import data:

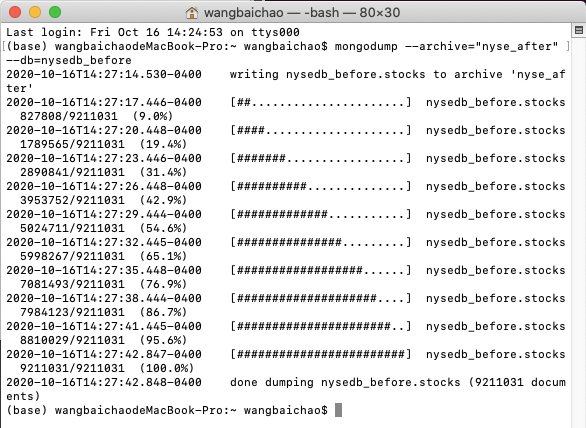




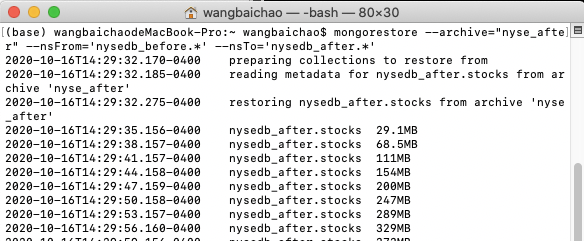
It will take more time to import the dataset if I created the index before importing.

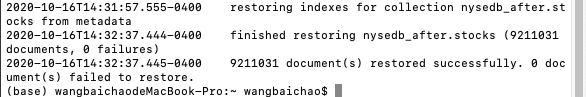
**PART 3 - MongoDB Indexing**  
Insert the NYSE dataset into a new database. You may use the existing NYSE database created before. Now, create indexes on existing data sets.

1. Use [mongodump](https://docs.mongodb.com/database-tools/mongodump/" \l "bin.mongodump" \o "(in database-tools vmaster)) to dump the nysedb \_before database to an archive nysedb\_after

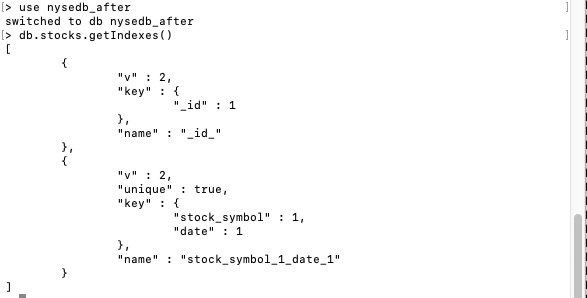


1. Use [mongorestore](https://docs.mongodb.com/database-tools/mongorestore/" \l "bin.mongorestore" \o "(in database-tools vmaster)) with [--nsFrom](https://docs.mongodb.com/database-tools/mongorestore/#cmdoption-mongorestore-nsfrom) and [--nsTo](https://docs.mongodb.com/database-tools/mongorestore/#cmdoption-mongorestore-nsto) to restore (with database name change) from the archive:

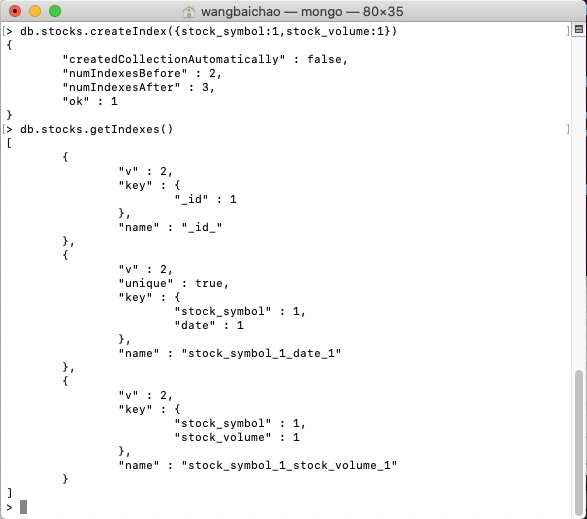




After copy:



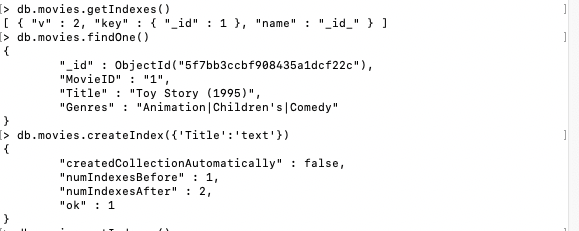
1. Add index to stock\_symbol and stock\_volume:

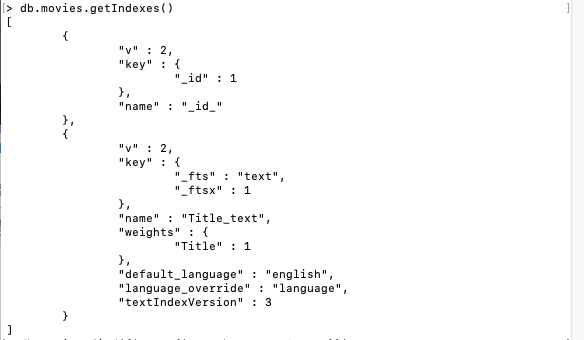


1 indicating that it should be in ascending order.

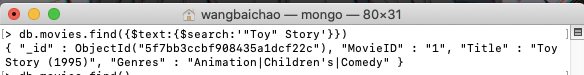
**PART 4 - MongoDB** **Text search (refer to chapter 9).  
Write and execute one query to perform each of the followings on any collection of your choice.**

1. Specify and word matches instead of or word matches.
2. Change to movies lens db,and create a text index for title:

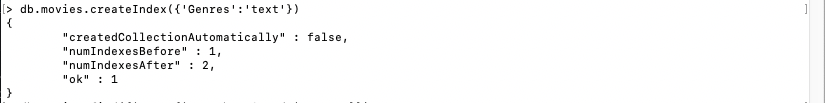




1. Specify “Toy” must be in the result document



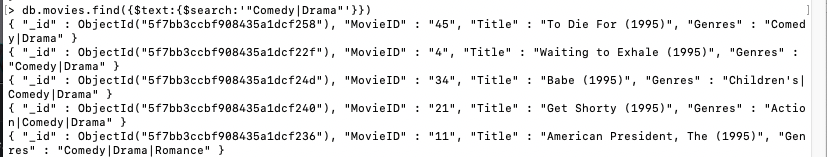
1. Perform exact phrase matches.
2. Create index for genres:



1. “Waiting to” phrase must in the result

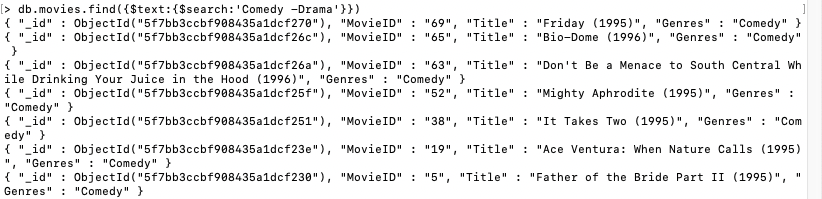


3) “Comedy|Drama” must in the result

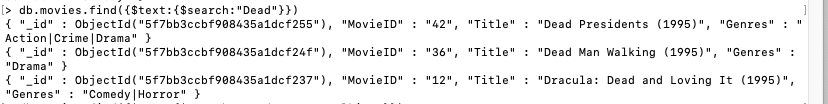


1. Exclude documents with certain words.

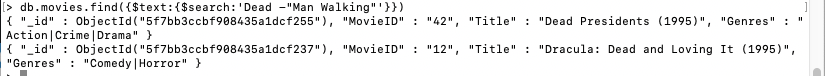
Search movies in Comedy genre and not in Drama



1. Exclude documents with certain phrases.
2. Serch movies which’s title include “Dead”



1. Search movies’ title include “Dead” but exclude “Man Walking”



**PART 5 - PROGRAMMING ASSIGNMENT**

Write a Java (could be a console app, swing app, web app, etc of your choice - will only run once to import the data into MongoDB) program to read the following file, and insert into 3 different collections (movies, ratings, tags). - MovieLens 10M Stable benchmark dataset. 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. [[http://grouplens.org/datasets/movielens/ (Links to an external site.)](http://grouplens.org/datasets/movielens/)]  
Once the data are inserted into MongoDB, do the followings using MapReduce: Write a MapReduce to do the followings:

Import tags,movies,ratings to lens\_10M

1. **import** java.io.File;
2. **import** java.io.FileNotFoundException;
3. **import** java.util.Map;
4. **import** java.util.Scanner;
5. **import** java.util.function.Consumer;
6. **import** com.mongodb.\*;
7. **import** com.mongodb.MongoClient;
8. **import** com.mongodb.client.MongoCollection;
9. **import** com.mongodb.client.MongoDatabase;
10. **import** com.mongodb.MongoClientURI;
11. **import** com.mongodb.ServerAddress;
12. **import** com.mongodb.MongoCredential;
13. **import** com.mongodb.MongoClientOptions;
14. **import** org.bson.Document;
16. **public** **class** MovieImport {
17. **private** **static** **final** String host = "localhost";
18. **private** **static** **final** **int** port = 27017;
19. **private** **static** **final** String databaseName = "lens\_10M";

22. **public** **static** **int** importToMongo(String collectionName, String fileFullPath, String header, String delimeter) {
23. MongoClient client = **new** MongoClient(host, port);
24. MongoDatabase database = client.getDatabase(databaseName);
25. MongoCollection<Document> collection = database.getCollection(collectionName);
26. Document document = **new** Document();
28. **int** counter = 0;
29. Scanner data;
30. **try** {
31. data = **new** Scanner(**new** File(fileFullPath));
32. **if** (header == **null**)
33. header = data.nextLine();
34. String[] names = header.split(delimeter);
36. **while** (data.hasNextLine()) {
37. String[] columns = data.nextLine().split(delimeter);
38. **for** (**int** i = 0; i < names.length; i++) {
39. document.put(names[i], columns[i]);
41. }
42. collection.insertOne(document);
43. document.clear();
44. counter++;
45. }
46. data.close();
47. } **catch** (FileNotFoundException e) {
48. counter = -1;
49. e.printStackTrace();
50. }
52. client.close();
54. **return** counter;

57. }
59. **private** **static** **void** loadTags() {
60. String header = "UserID::MovieID::Tag::Timestamp";
61. String collection = "tags";
62. **int** n = importToMongo(collection, "/Users/wangbaichao/Desktop/INFO7250/Assignments/assignment3/ml-10M100K/tags.dat", header, "::");
63. System.out.println("Successfully imported: " + n + " documents to " + collection);
64. }
66. **private** **static** **void** loadRatings() {
67. String header = "UserID::MovieID::Rating::Timestamp";
68. String collection = "ratings";
69. **int** n = importToMongo(collection, "/Users/wangbaichao/Desktop/INFO7250/Assignments/assignment3/ml-10M100K/ratings.dat", header, "::");
70. System.out.println("Successfully imported: " + n + " documents to " + collection);
71. }


75. **private** **static** **void** loadMovies() {
76. String header = "MovieID::Title::Year::Genres";
77. String collection = "movies";
78. **int** n = importToMongo(collection, "/Users/wangbaichao/Desktop/INFO7250/Assignments/assignment3/ml-10M100K/movies.dat", header, "::");
79. System.out.println("Successfully imported: " + n + " documents to " + collection);
80. }
82. **public** **static** **void** loadDataToMongo() {
84. loadTags();
85. loadMovies();
86. loadRatings();
87. }
89. **public** **static** **void** main(String[] args) {
90. loadDataToMongo();
92. }
93. }

Import result:



- Number of Movies released per year (Movies Collection)

Map:

var map1 = function () { emit(this.Year,{'MovieID':this.MovieID,'count':1}); }

Reduce:

var reduce1 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

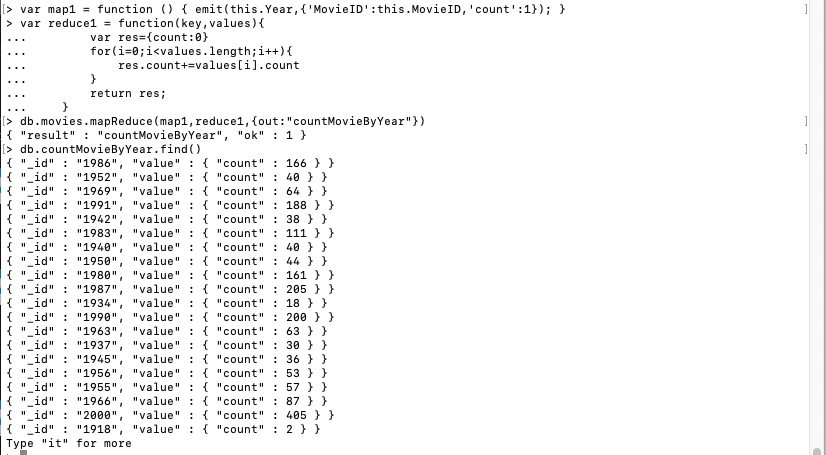
res.count+=values[i].count

}

return res;

}

Result:



- Number of Movies per genre (Movies Collection)

Map:

var map2 = function () { emit(this.Genres,{'MovieID':this.MovieID,'count':1}); }

Reduce:

var reduce2 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

res.count+=values[i].count

}

return res;

}

Result:



- Number of Movies per rating (Ratings Collection)

Map:

var map3 = function () { emit(this.Rating,{'MovieID':this.MovieID,'count':1}); }

Reduce:

var reduce3 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

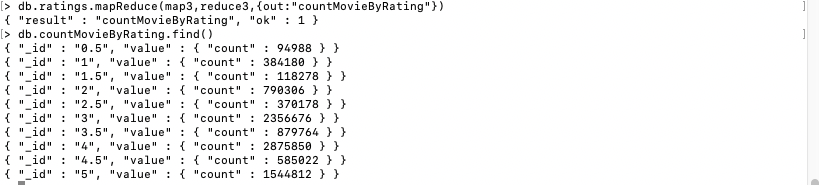
res.count+=values[i].count

}

return res;

}

Result:



- Number of times each movie was tagged (Tags Collection)

Map:

var map4 = function () { emit(this.MovieID,{'Tag':this.Tag,'count':1}); }

Reduce:

var reduce4 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

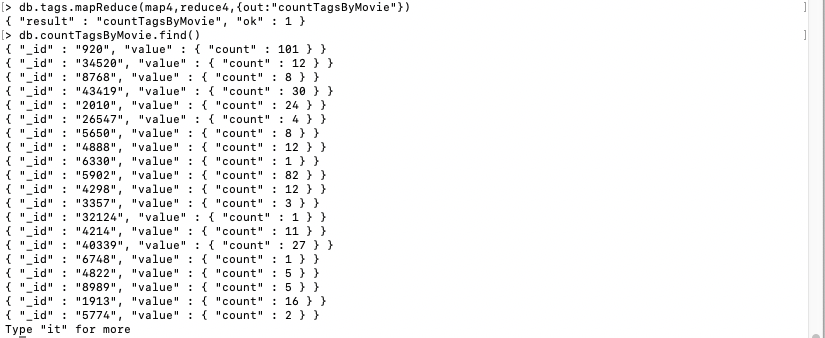
res.count+=values[i].count

}

return res;

}

Result:



**PART 6 - PROGRAMMING ASSIGNMENT [**[**access-1.log**](https://northeastern.instructure.com/courses/7136/files/608290/download?wrap=1)**[Preview the document](https://northeastern.instructure.com/courses/7136/files/608290/download?wrap=1)]**Write a Java (could be a console app - will only run once to import the data into MongoDB) program to read the access.log file (attached), and insert into access collection.  Once the data are inserted into MongoDB, do the followings using MapReduce:

Import to log db:

1. **import** java.io.File;
2. **import** java.io.FileNotFoundException;
3. **import** java.util.Map;
4. **import** java.util.Scanner;
5. **import** java.util.function.Consumer;
6. **import** com.mongodb.\*;
7. **import** com.mongodb.MongoClient;
8. **import** com.mongodb.client.MongoCollection;
9. **import** com.mongodb.client.MongoDatabase;
10. **import** com.mongodb.MongoClientURI;
11. **import** com.mongodb.ServerAddress;
12. **import** com.mongodb.MongoCredential;
13. **import** com.mongodb.MongoClientOptions;
14. **import** **static** com.mycompany.mongomoview.MovieImport.importToMongo;
15. **import** java.util.Arrays;
16. **import** org.bson.Document;
17. **public** **class** LogImport {
18. **private** **static** **final** String host = "localhost";
19. **private** **static** **final** **int** port = 27017;
20. **private** **static** **final** String databaseName = "log";
22. **public** **static** **int** importToMongo(String collectionName, String fileFullPath, String header, String delimeter) {
23. MongoClient client = **new** MongoClient(host, port);
24. MongoDatabase database = client.getDatabase(databaseName);
25. MongoCollection<Document> collection = database.getCollection(collectionName);
26. Document document = **new** Document();
28. **int** counter = 0;
29. Scanner data;
30. **try** {
31. data = **new** Scanner(**new** File(fileFullPath));
32. **if** (header == **null**)
33. header = data.nextLine();
34. String[] names = header.split(delimeter);

37. **while** (data.hasNextLine()) {
38. String nextLine = data.nextLine();
40. String[] columns = Spliter(nextLine);
41. **for** (**int** i = 0; i < names.length; i++) {
42. document.put(names[i], columns[i]);
44. }
45. collection.insertOne(document);
46. document.clear();
47. counter++;
48. }
49. data.close();
50. } **catch** (FileNotFoundException e) {
51. counter = -1;
52. e.printStackTrace();
53. }
55. client.close();
57. **return** counter;

60. }
62. **public** **static** String[] Spliter(String nextLine){
63. String[] split1 = nextLine.split(" - - \\[");
64. String ip = split1[0];
65. String[] split2 = split1[1].split("/",2);
66. String day = split2[0];
67. String[] split3 =split2[1].split("/",2);
68. String month = split3[0];
69. String[] split31=split3[1].split(":",2);
70. String year = split31[0];
71. String[] split4 =split31[1].split("\\] \"");
72. String time = split4[0];
73. String[] split5 =split4[1].split("\" ");
74. String methodAndPath=split5[0];
75. String[] split6=split5[1].split(" ");
76. String statusCode=split6[0];
77. String areaCode=split6[1];
79. String[] res = **new** String[8];
80. res[0]=ip;
81. res[1]=day;
82. res[2]=month;
83. res[3]=year;
84. res[4]=time;
85. res[5]=methodAndPath;
86. res[6]=statusCode;
87. res[7]=areaCode;
89. **return** res;
90. }

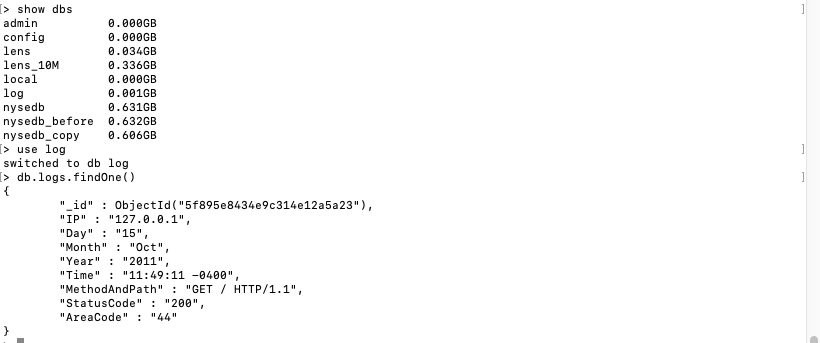
93. **private** **static** **void** loadLogs() {
94. String header = "IP Day Month Year Time MethodAndPath StatusCode AreaCode";
95. String collection = "logs";
96. **int** n = importToMongo(collection, "/Users/wangbaichao/Desktop/INFO7250/Assignments/assignment3/access-1.log", header, " ");
97. System.out.println("Successfully imported: " + n + " documents to " + collection);
98. }

101. **public** **static** **void** loadDataToMongo() {
103. loadLogs();
105. }
107. **public** **static** **void** main(String[] args) {
108. loadDataToMongo();
109. //System.out.println(Arrays.toString(Spliter("127.0.0.1 - - [15/Oct/2011:11:49:11 -0400] \"GET / HTTP/1.1\" 200 44")));
111. }
112. }

Import successful



Import result:



- Number of times any webpage was visited by the same IP address.

Map:

var map5 = function () { emit(this.IP,{'\_id':this.\_id,'count':1}); }

Reduce:

var reduce5 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

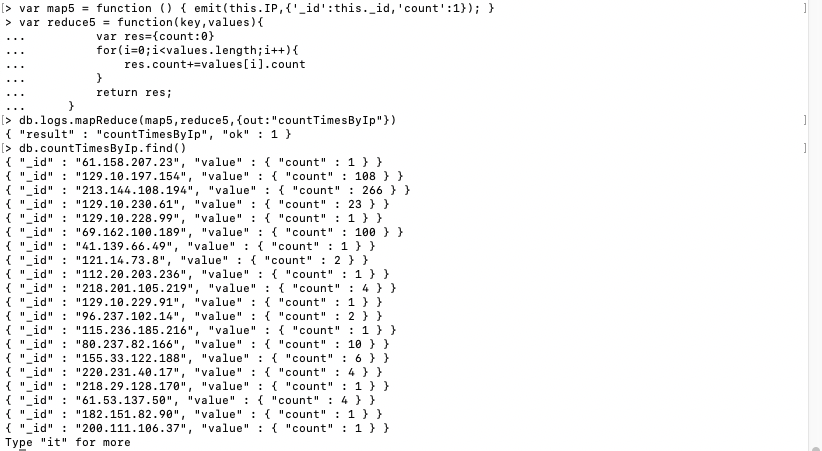
res.count+=values[i].count

}

return res;

}

Result:



- Number of times any webpage was visited each month.

Map:

var map6 = function () { emit(this.Month,{'\_id':this.\_id,'count':1}); }

Reduce:

var reduce6 = function(key,values){

var res={count:0}

for(i=0;i<values.length;i++){

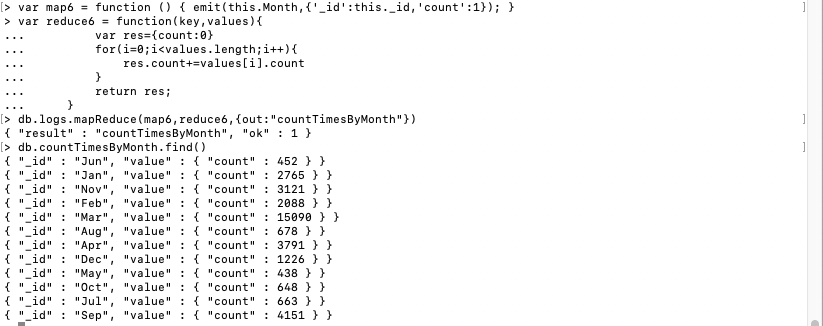
res.count+=values[i].count

}

return res;

}

Result:



**PART 7. Programming Assignment**

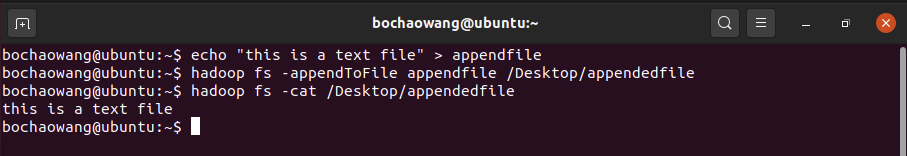
All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

Usage: hadoop [--config confdir] [--loglevel loglevel] [COMMAND] [GENERIC\_OPTIONS] [COMMAND\_OPTIONS]

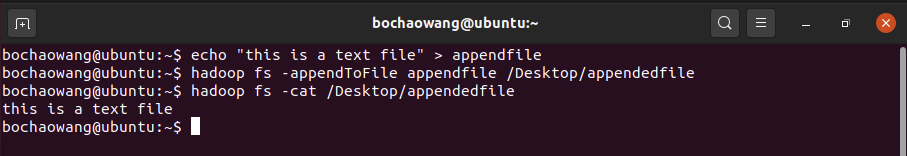
Execute each hadoop command once, and place the screenshots into a word file. If a command cannot be executed for any reason (such as, a distributed environment is needed), you may write the definition of the command, and skip execution.

[http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html (Links to an external site.)](http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html)

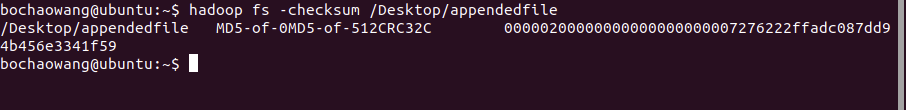
1) appendToFile



2) cat

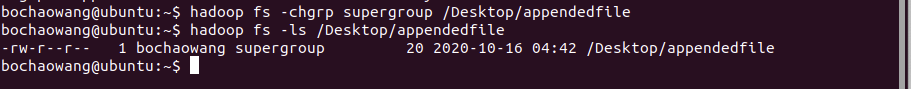


1. checksum



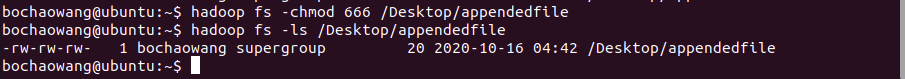
Returns the checksum information of a file.

1. chgrp



Change group asscociation of files

1. chomod



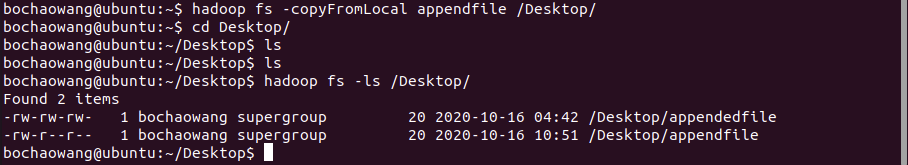
Change permission of files

1. chown



Change the owner of files

1. copyFromLocal

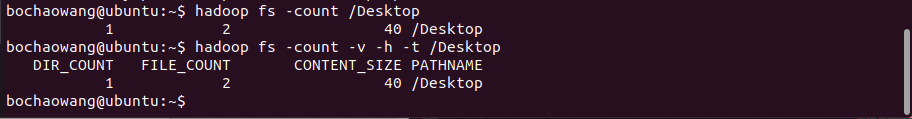


Similar to the fs -put command, except that the source is restricted to a local file reference.

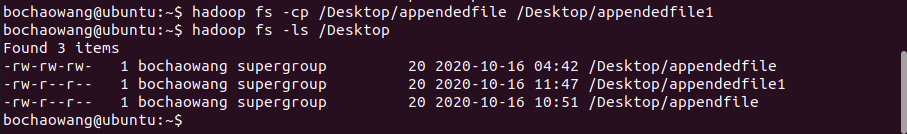
1. copyToLocal



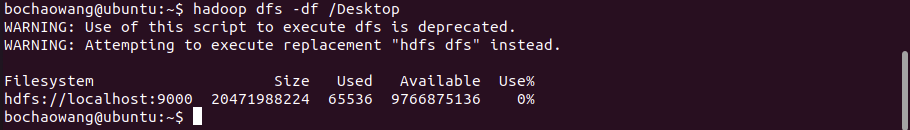
1. count



1. cp

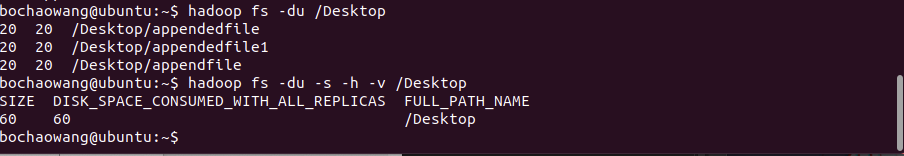


1. df



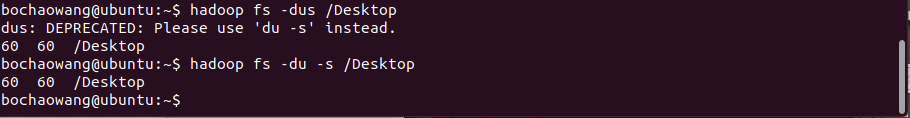
Displays free space.

1. du



Displays sizes of files and directories contained in the given directory or the length of a file in case its just a file.

1. dus



1. expunge

Permanently delete files in checkpoints older than the retention threshold from trash directory, and create new checkpoint.



1. find



Finds all files that match the specified expression and applies selected actions to them.

1. get



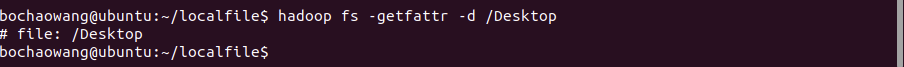


1. getfacl



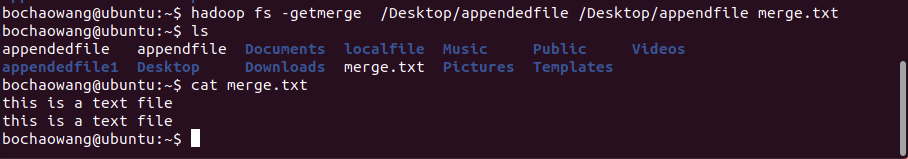
Displays the Access Control Lists (ACLs) of files and directories.

1. getfattr



Displays the extended attribute names and values (if any) for a file or directory.

1. getmerge



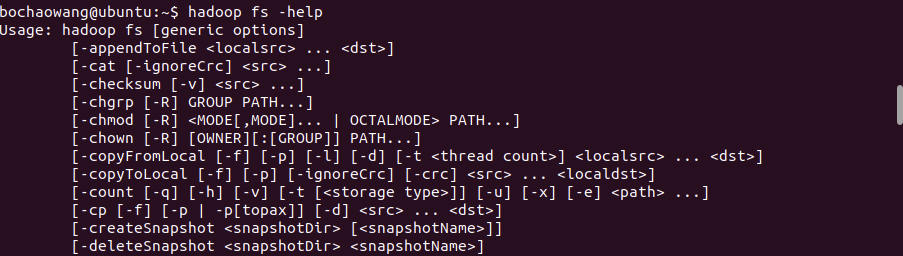
Takes a source directory and a destination file as input and concatenates files in src into the destination local file.

1. head

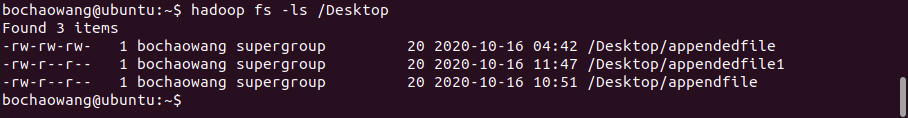


Displays first kilobyte of the file to stdout.

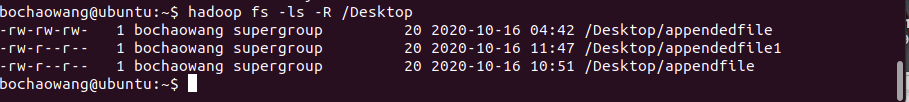
1. help



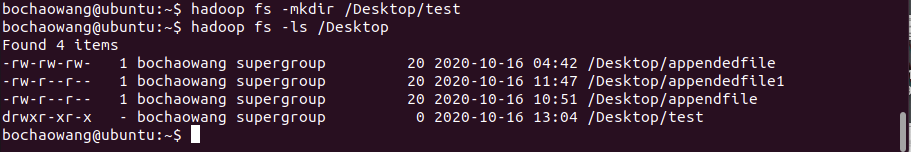
1. ls



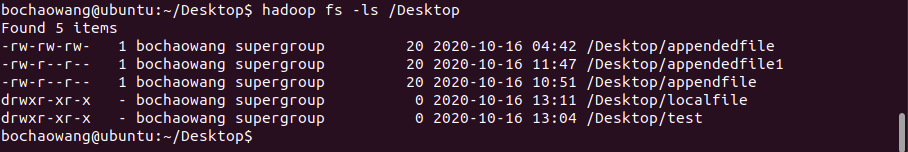
1. lsr



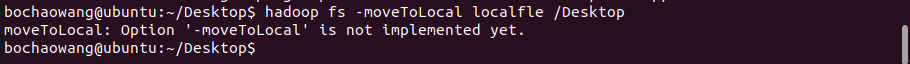
1. mkdir



1. moveFromLocal

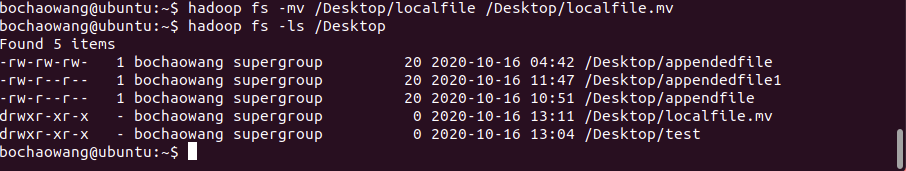


1. moveToLocal

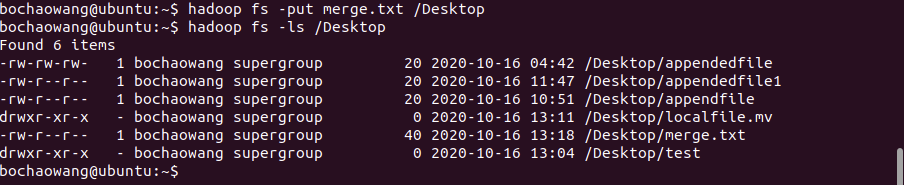


Displays a “Not implemented yet” message.

1. mv

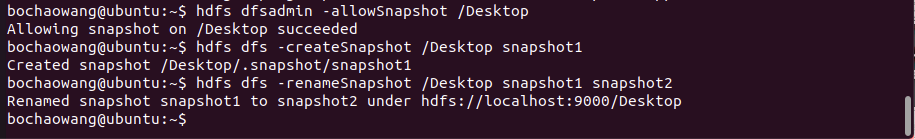


1. put

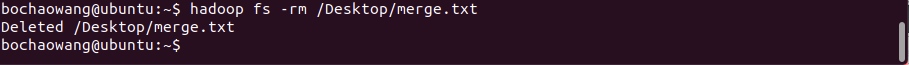


Copy single src, or multiple srcs from local file system to the destination file system.

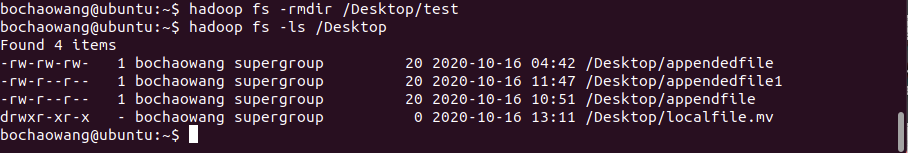
1. RenameSnapshot



1. rm



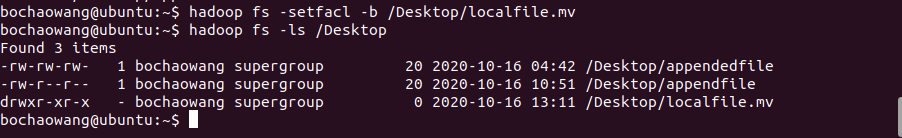
1. rmdir



1. rmr

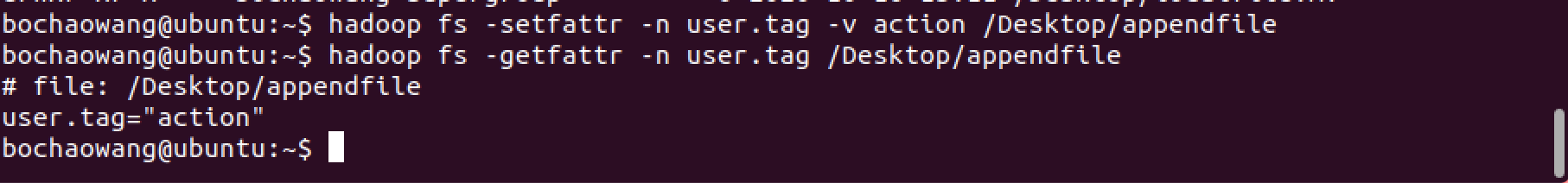


1. setfacl



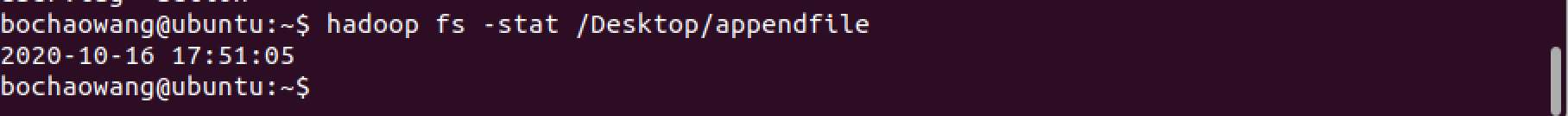
Sets Access Control Lists (ACLs) of files and directories.

1. setfattr

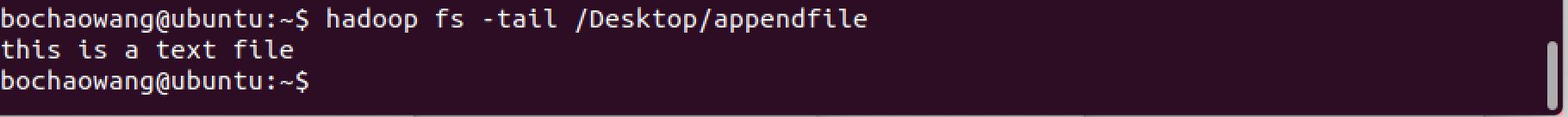


Sets Access Control Lists (ACLs) of files and directories.

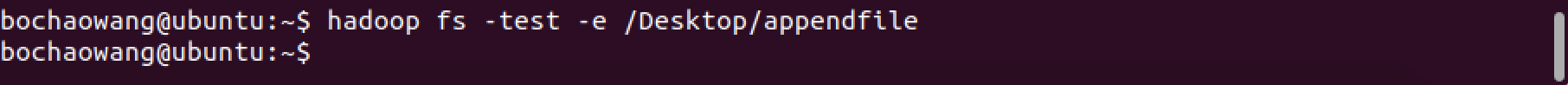
1. stat



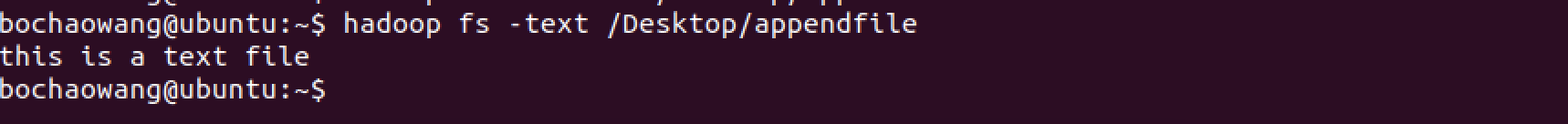
1. tail



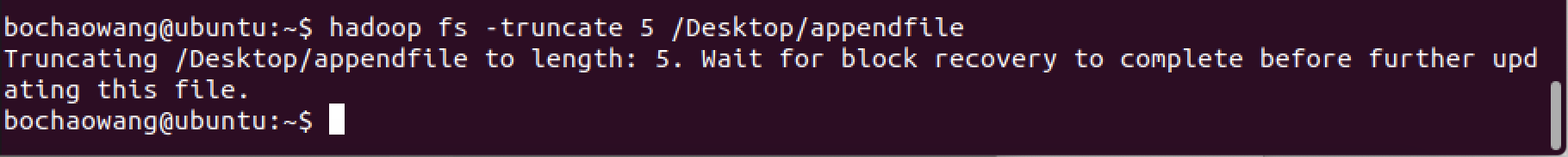
1. test



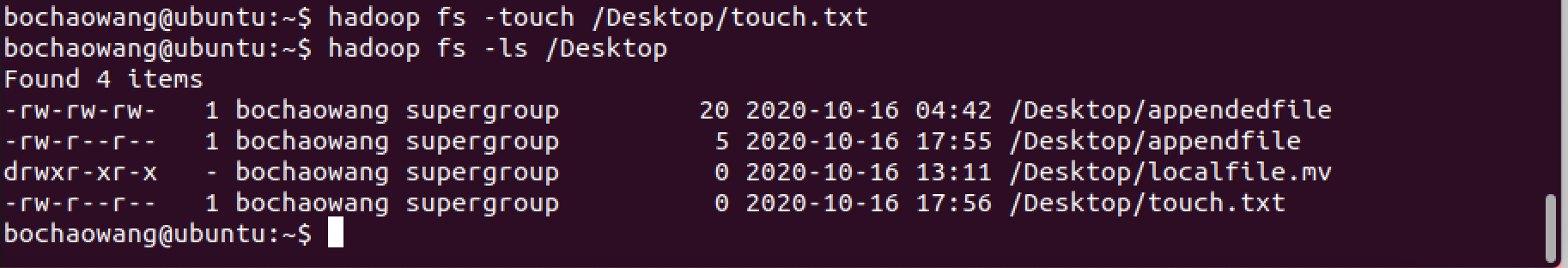
1. text



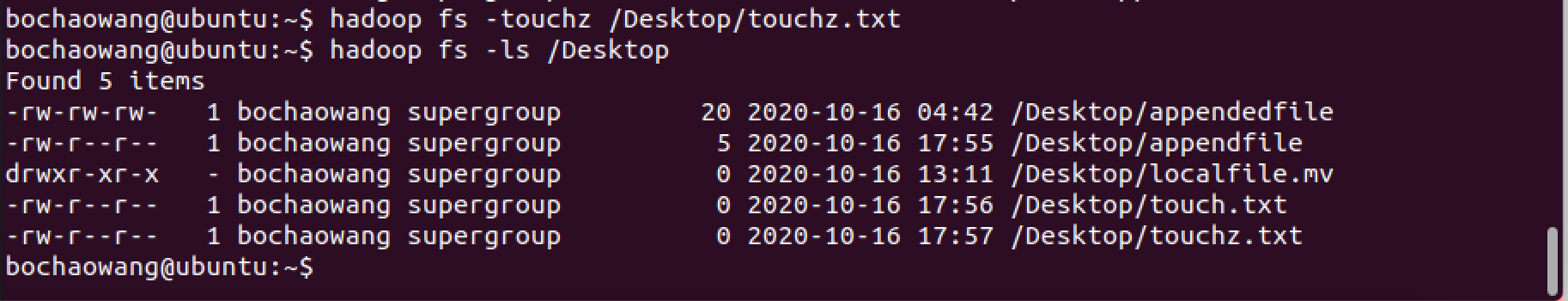
1. truncate



1. touch

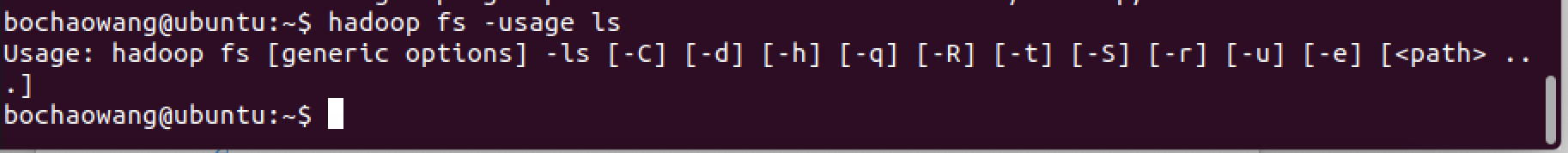


1. touchz



Create a file of zero length.

1. usage



**PART 8. HDFS Command Assignment - Submit screenshots**  
Copy all the files (A to Z) from NYSE dataset into HDFS.

1) copy all the files(A to Z)

FILES=NYSE/NYSE\_daily\_prices\_\*.csv

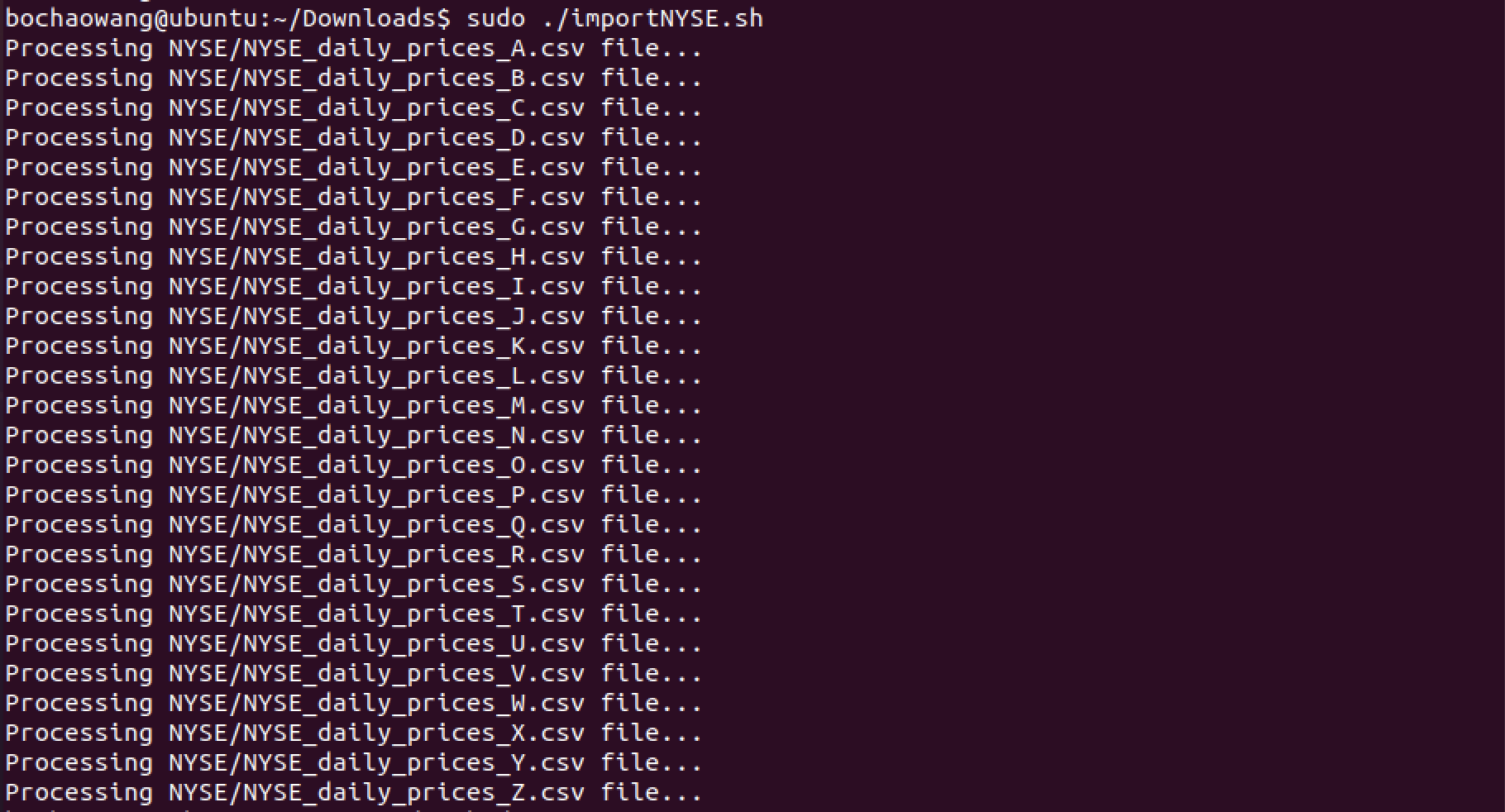
for f in $FILES

do

echo "Processing $f file..."

sudo /usr/local/bin/hadoop-3.3.0/bin/hadoop fs -appendToFile $f /movies/$f

done





2) Merge all files together

!/bin/bash

FILES=NYSE/NYSE\_daily\_prices\_\*.csv

for f in $FILES

do

echo "Processing $f file..."

cat $f >> NYSE\_all.csv

done

h

