

Hadoop

Steps to run JAR

1. `chmod 777 <your jar>.jar`
2. `hadoop fs -mkdir VoteCountInput`
3. `hadoop fs -cp votes.txt VoteCountInput`
4. `hadoop jar <your jar>.jar VoteCountInput VoteCountOutput`
5. `hadoop fs -ls VoteCountOutput`
6. `hadoop fs -cat VoteCountOutput/part-r-00000`

File & Folder commands

Command	Meaning
<code>hadoop fs -<Unix commands></code>	Main rule
<code>hadoop fs -ls</code>	Show content hadoop home folder
<code>hadoop fs put votes.txt</code>	Upload file to hadoop cluster (hdfs)
<code>hadoop fs -tail votes.txt</code>	Show last lines file
<code>hadoop fs -cat votes.txt</code>	Show full content
<code>hadoop fs -cp votes.txt votes2.txt</code>	Copy file
<code>hadoop fs -mv votes.txt newname.txt</code>	Move file
<code>hadoop fs -rm newname.txt</code>	Remove file
<code>hadoop fs -mkdir myinput</code>	Create folder
<code>hadoop fs -du votes.txt</code>	Get size of file / folder
<code>hadoop fs -du -s /.</code>	Get size of everything in the folder

Admin stuff

Command	Meaning
<code>hdfs dfsadmin</code>	Get all possible commands for <code>hdfs dfsadmin</code>
<code>hdfs dfsadmin -report</code>	Get dfs report

Configuration files

`/home/training/src/hadoop-common-project/hadoop-common/src/test/resources/`

Command	Meaning
<code>core-site.xml</code>	Configuration file of the Hadoop system <code>fs.s3.block.size</code> Storage block size
<code>hdfs-site.xml</code>	HDFS specific configuration parameters
<code>mapred-size.xml</code>	MapReduces configuration

Required JAR

1. `/usr/lib/hadoop/hadoop-common-2.0.0-cdh4.1.1.jar`
2. `/usr/lib/hadoop-mapreduce/hadoop-mapreduce-client-core-2.0.0-cdh4.1.1.jar`
3. `/usr/lib/hadoop-mapreduce/hadoop-mapreduce-jobclient-2.0.0-cdh4.1.1.jar`
4. `/usr/lib/hadoop/lib/commons-cli-1.2.jar`

5. /usr/lib/hadoop/lib/commons-logging-1.1.1.jar

Hive

Command	Meaning
hive	Load hive shell
quit;	Exit shell
hive -e "HiveQL query"	Execute query
hive -e "select sighted from ufodata limit 5;"	Sample of 5 values sighted column
hive -e "select reported from ufodata"	Validate the contents of column
hive -f commands.hql	Execute file

Load data in the database from hadoop filesystem.

```
LOAD DATA INPATH '/tmp/ufodata.tsv' OVERWRITE INTO TABLE ufodata;
```

Set right column separator

```
CREATE TABLE ufodata(sighted string, reported string, sighting_location
string, shape string, duration string, description string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

Create table from existing file

```
CREATE EXTERNAL TABLE states(abbreviation string, full_name string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/tmp/states';
```

Exporting query output

```
INSERT OVERWRITE DIRECTORY '/tmp/out'
SELECT t1.sighted, t1.reported, t1.shape, t2.full_name
FROM ufodata t1 JOIN states t2
ON (LOWER(t2.abbreviation) = LOWER(substr(t1.sighting_location, (LENGTH(t1.sighting_location) -1))));
```

- Output will be in /tmp/out
 - `hadoop fs -ls /tmp/out`
 - `hadoop fs -cat /tmp/out/000000_1 | head`

Making partitioned UFO sighting table

```
CREATE TABLE partufo(sighted string, reported string, sighting_location
string, shape string, duration string, description string)
PARTITIONED BY (year string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
INSERT OVERWRITE TABLE partufo partition (year)
SELECT sighted, reported, sighting_location, shape, duration,
description, SUBSTR(TRIM(sighted), 1,4) FROM ufodata;
```

- Location of the partitioned table
 - `/user/hive/warehouse/partufo/`

HiveQL vs SQL

[Source](#)

MySQL	HiveQL
SELECT from_columns FROM table WHERE conditions;	SELECT from_columns FROM table WHERE conditions;
SELECT * FROM table;	SELECT * FROM table;
SELECT * FROM table WHERE rec_name = "v";	SELECT * FROM table WHERE rec_name = "v";
SELECT * FROM TABLE WHERE r1 = "a" AND r2 = "b";	SELECT * FROM TABLE WHERE r1 = "a" AND r2 = "b";
SELECT column_name FROM table;	SELECT column_name FROM table;
SELECT DISTINCT column_name FROM table;	SELECT DISTINCT column_name FROM table;
SELECT c1, c2 FROM table ORDER BY c2;	SELECT c1, c2 FROM table ORDER BY c2;
SELECT c1, c2 FROM table ORDER BY c2 DESC;	SELECT c1, c2 FROM table ORDER BY c2 DESC;
SELECT COUNT(*) FROM table;	SELECT COUNT(*) FROM table;
SELECT own, COUNT(*) FROM table GROUP BY own;	SELECT own, COUNT(*) FROM table GROUP BY own;
SELECT MAX(col_name) AS label FROM table;	SELECT MAX(col_name) AS label FROM table;
SELECT p.na, comment FROM p, e WHERE p.na = e.na;	SELECT p.name, comment FROM p JOIN e ON (p.na = e.na)

MySQL	HiveQL
USE database;	USE database;
SHOW DATABASES;	SHOW DATABASES;
SHOW TABLES;	SHOW TABLES;
	SHOW TABLES '.*data';
DESCRIBE table;	DESCRIBE (FORMATTED EXTENDED) table;
CREATE DATABASE db_name;	CREATE DATABASE db_name;
DROP DATABASE db_name;	DROP DATABASE db_name (CASCADE);

Sqoop

Required packages - mysql-server - mysql

Setup database

```
mysql> create database hadooptest;
mysql> CREATE USER 'hadoopuser'@'localhost' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'hadoopuser'@'localhost' WITH GRANT OPTION;
mysql> flush privileges;
```

Load data in existing table

```
mysql> load data local infile 'employees.tsv'
into table employees
fields terminated by '\t' lines terminated by '\n';
```

Exporting data from MySQL to HDFS

```
sqoop import --connect jdbc:mysql://localhost/hadooptest
--username hadoopuser --password password --table employees
```

- `hadoop fs -ls employees`
- `hadoop fs -cat employees/part-m-00003`

Exporting data from MySQL into Hive `hive -e "show tables like 'employees'"`

```
sqoop import --connect jdbc:mysql://localhost/hadooptest
--username hadoopuser --password password --table employees
--hive-import --hive-table employees
```

Selective import

```
sqoop import --connect jdbc:mysql://localhost/hadooptest
--username hadoopuser --password password
--table employees --columns first_name,salary
--where "salary > 45000"
--hive-import --hive-table salary
```

Import data from Hadoop into MySQL

1. `hadoop fs -mkdir edata`
2. `hadoop fs -put newemployees.tsv edata/newemployees.tsv`

```
sqoop export --connect jdbc:mysql://localhost/hadooptest
--username hadoopuser --password password --table employees
--export-dir edata --input-fields-terminated-by '\t'
```

Importing Hive data into MySQL

1. Empty your table (`truncate employees`)
2. `hadoop fs -ls /user/hive/warehouse/employees`

```
sqoop export --connect jdbc:mysql://localhost/hadooptest
--username hadoopuser --password password --table employees
--export-dir /user/hive/warehouse/employees
--input-fields-terminated-by '\001'
--input-lines-terminated-by '\n'
```

Mongo

Command	Meaning
<code>mongo</code>	Load mongo shell
<code>show dbs</code>	Show current databases
<code>use movies</code>	Use the database <code>movies</code>
<code>db.getName()</code>	Get name current database
<code>show collections</code>	Show all tables

Create / Add data in MongoDB

```
db.comedy.insert({name:"Ted", year:2012})
db.comedy.save({name:"Ted", year:2012})
    save does an insert if there is no `_id`key in the object, else it does an update
```

Find

```
db.comedy.find();
    db.comedy.find({ "field" : { $gt: value } }); // greater than : field > value
    db.comedy.find({ "field" : { $lt: value } }); // less than : field < value
    db.comedy.find({ "field" : { $gte: value } }); // greater than or equal to : field >= value
```

```

db.comedy.find( { "field" : { $lte: value } } ); // less than or equal to : field <= value

db.comedy.find( { year: { $gt: 2007, $lt: 2011 } } )
db.comedy.find( { year: 2012 } )

db.comedy.find( { year: { $lt: 2012 } }, { name: true } ) // only show name column
db.comedy.find( { year: { $lt: 2012 } }, { name: false } ) // all columns except name

db.comedy.find( { year: { $ne: 2011 } } ) // $ne -> !=

db.comedy.find( { year: { $in: [2010, 2011, 2012] } } )
db.comedy.find( { year: { $nin: [2008, 2009] } } ) // $nin = not in

db.comedy.find( { $nor: [ { year: 2012 }, { name: 'The hangover' } ] } ) // all must be !=

db.comedy.find( { $or: [ { year: 2012 }, { name: 'The hangover' } ] } )
db.comedy.find( { year: 2012, $or: [ { name: 'Ted' }, { name: 'The hangover' } ] } )

db.comedy.find( { $and: [ { year: { $gt: 2010 } }, { year: { $lt: 2012 } } ] } )
db.comedy.find( { year: { $gt: 2010, $lt: 2012 } } )

```

Find dot notation

```

db.articles.insert( { title: 'MongoDB in Mongolia', meta: { author: 'Ghenghiz Khan', date: 1321958598538, likes: 75, tags: ['mongo', 'mongolia', 'ghenghiz'] }, comments: [ { by: 'Alex', text: 'Dude, it rocks' }, { by: 'Steve', text: 'The best article ever!' } ] } )

```

- article

```

- title
- meta
  * author
  * date
  * likes
  * tags (array)
- comments (array)
  * by
  * text

db.articles.find( { 'meta.author': 'Chad Muska' } )
db.articles.find( { 'meta.likes': { $gt: 10 } } )
db.articles.find( { 'meta.tags': 'mongolia' } )
db.articles.find( { 'comments.by': 'Steve' } )

```

Size

```

db.articles.find( { comments : { $size: 2 } } ) // exact 2 comments
db.articles.find( { tags : { $size: 2 } } ) // exact 2 tags

```

Regular Expression

```

db.comedy.find( { name: { $regex: /bill| ted/i } } )
db.comedy.find( { name: /The hangover.*i } );
db.comedy.find( { name: { $regex: 'The hangover.*', $options: 'i' } } );
db.comedy.find( { name: { $regex: /The hangover.*i, $nin: [ 'The Hangover Part II' ] } } );

```

Not

```
db.comedy.find({name: {$not: /The hangover.*/i}} );
```

Javascript expression

```
db.comedy.find('this.year > 2009 && this.name !== "Ted"')
=
db.comedy.find({year:{$gt: 2009}, name:{$ne:'Ted'}})

db.comedy.find({ $where: 'this.year > 2011'}) // $where expression lets you use SQL
db.comedy.find({name:'Ted', $where: 'this.year > 2010'})
```

Count

```
db.comedy.count()
db.comedy.count({year:{$gt:2009} } )
```

skip() and find()

```
db.comedy.find().limit(2) //limit output to 2

db.comedy.findOne()

db.comedy.find().skip(1).limit(2)
```

all

```
db.articles.find ({ 'meta.tags' : {$all: ['mongodb', 'mongo']}});
```

exists

```
db.articles.find({title: {$exists : true} } );
db.articles.find({titles: {$exists : true}});
db.articles.find({titles: {$exists : false}});
db.articles.find({ ' comments.by' : {$exists : true}});
```

sort()

```
db. articles.find().sort( { title : -1 } ); // sort by title, descending order
```

Update

```
db.comedy.update({name:"Ted"}, {$set:{director:'Seth MacFarlane', cast:['Mark Wahlberg', ' Mila Kunis']}})
db.comedy.update({name:"Ted"}, {$push:{cast:'Joel McHale'}}) // add into array

db.comedy.update({name:"Ted"}, {$pull:{cast:' Giovanni Ribisi '}}) //remove item in array
```

Delete

```
db.comedy.update({name:'Ted'}, {$unset:{cast:1}})

db.comedy.update({$unset:{cast:1}}, false, true)
false    -> upsert option
```

true -> multiple option, so if you want to delete from the whole collection

```
db.comedy.remove({name:' Ted'}) //delete all with name set to `Ted`
```

```
db.comedy.remove() // empty the collection
```

```
db.comedy.drop() // remove the collection
```

```
db.dropDatabase() // drop current database
```