



Co-funded by the  
Erasmus+ Programme  
of the European Union

Establishing Modern Master-level Studies in Information Systems  
561592-EPP-1-2015-1- FR-EPPKA2-CBHE-JP

## **“MIS and Data Warehousing”**

**Content module 4.**

**BIG DATA technology usage in DATA WAREHOUSING**

**Methodical instructions for laboratory work**

by Dr. Vyacheslav V. Kovtun (VNTU)

# Lab Work 01 – Downloading and Installing Hadoop

## Downloading and Installing the Cloudera Virtual Machine

### Learning Goals

In this activity, you will:

- Download and Install VirtualBox.
- Download and Install Cloudera Virtual Machine (VM) Image.
- Launch the Cloudera VM.

Hardware Requirements: (A) Quad Core Processor (VT-x or AMD-V support recommended), 64-bit; (B) 8 GB RAM; (C) 20 GB disk free. How to find your hardware information: Open System by clicking the Start button, right-clicking Computer, and then clicking Properties. Most computers with 8 GB RAM purchased in the last 3 years will meet the minimum requirements. You will need a high speed internet connection because you will be downloading files up to 4 Gb in size.

### Instructions

Please use the following instructions to download and install the Cloudera Quickstart VM with VirtualBox before proceeding to the Getting Started with the Cloudera VM Environment video. The screenshots are from a Mac but the instructions should be the same for Windows. Please see the discussion boards if you have any issues.

1. **Install VirtualBox.** Go to <https://www.virtualbox.org/wiki/Downloads> to download and install VirtualBox for your computer. The course uses Virtualbox 5.1.X, so we recommend clicking [VirtualBox 5.1 builds](#) on that page and downloading the older package for ease of following instructions and screenshots. However, it shouldn't be too different if you choose to use or upgrade to VirtualBox 5.2.X. For Windows, select the link "**VirtualBox 5.1.X for Windows hosts x86/amd64**" where 'X' is the latest version.

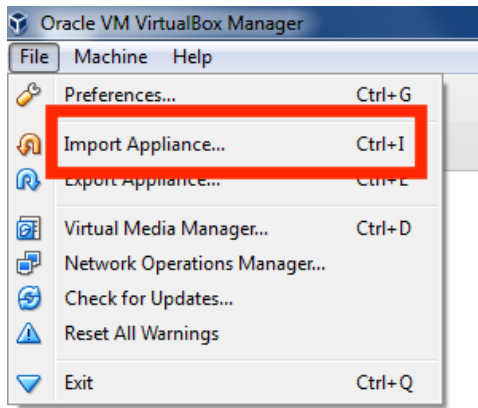
2. **Download the Cloudera VM.** Download the Cloudera VM from [https://downloads.cloudera.com/demo\\_vm/virtualbox/cloudera-quickstart-vm-5.4.2-0-virtualbox.zip](https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.4.2-0-virtualbox.zip). The VM is over 4GB, so will take some time to download.

3. **Unzip the Cloudera VM:**

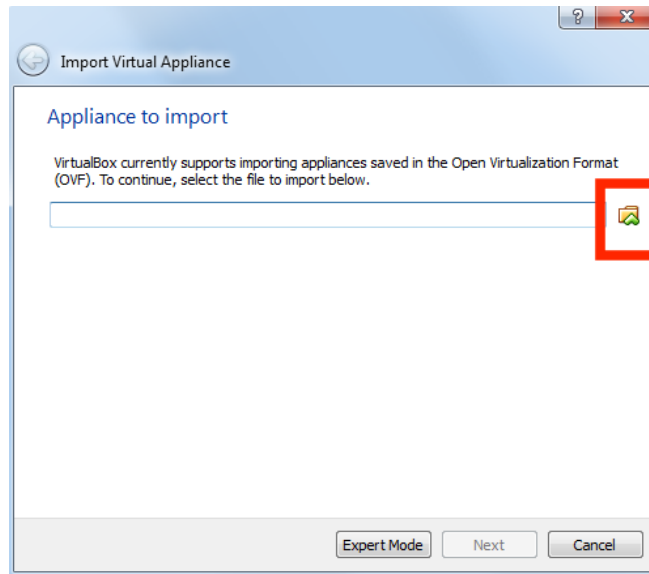
Right-click cloudera-quickstart-vm-5.4.2-0-virtualbox.zip and select "Extract All..."

4. **Start VirtualBox.**

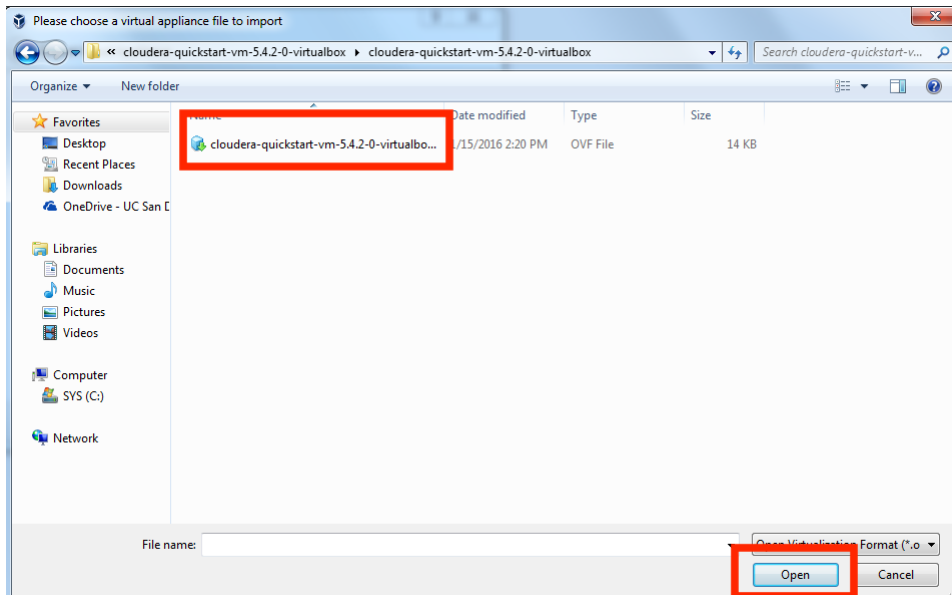
5. **Begin importing.** Import the VM by going to File -> Import Appliance



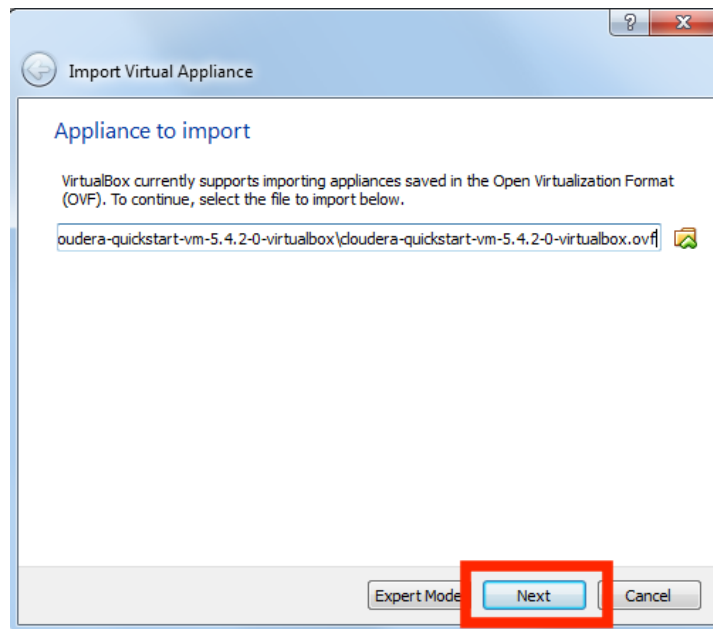
6. Click the Folder icon.



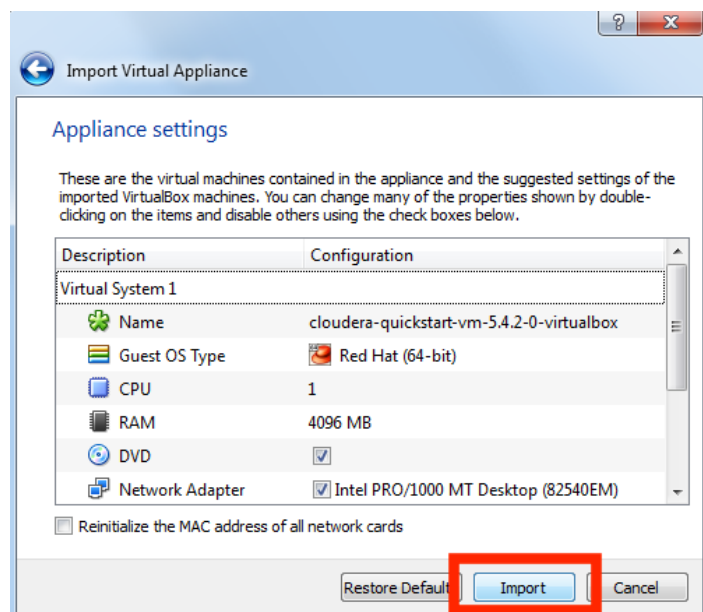
7. Select the cloudera-quickstart-vm-5.4.2-0-virtualbox.ovf from the Folder where you unzipped the VirtualBox VM and click Open.



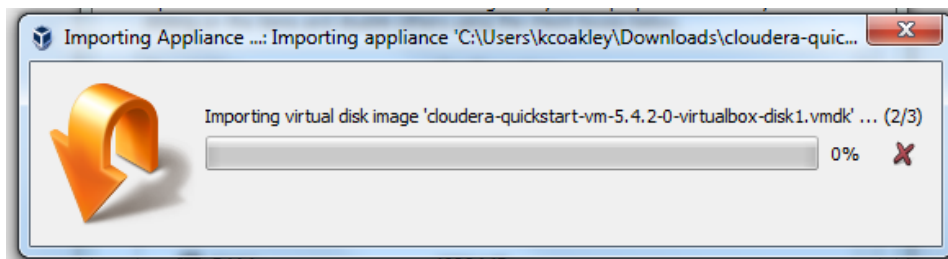
8. Click Next to proceed.



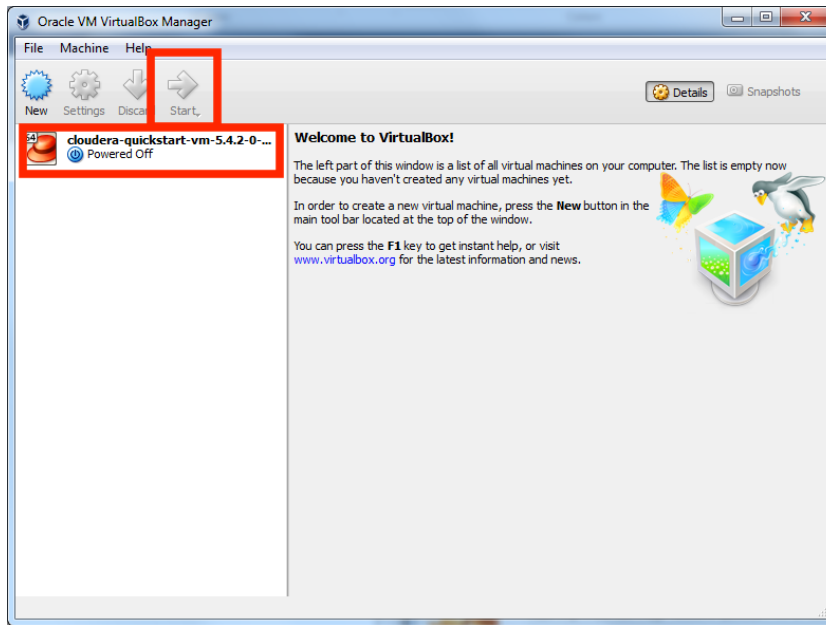
9. Click Import.



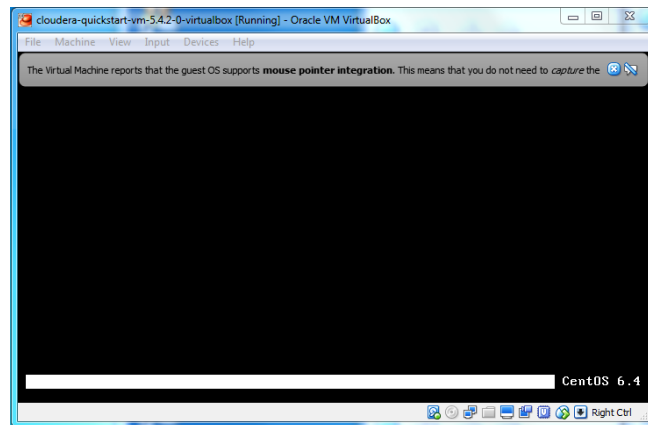
10. The virtual machine image will be imported. This can take several minutes.



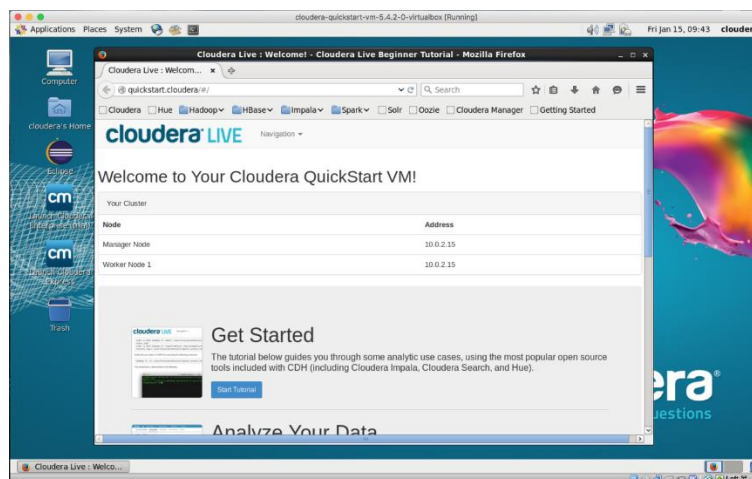
11. **Launch Cloudera VM.** When the importing is finished, the quickstart-vm-5.4.2-0 VM will appear on the left in the VirtualBox window. Select it and click the Start button to launch the VM.



12. **Cloudera VM booting.** It will take several minutes for the Virtual Machine to start. The booting process takes a long time since many Hadoop tools are started.



13. **The Cloudera VM desktop.** Once the booting process is complete, the desktop will appear with a browser.



## Copy your data into HDFS

### Learning Goals

By the end of this activity, you will be able to:

- Interact with Hadoop using the command-line application.
- Copy files into and out of the Hadoop Distributed File System (HDFS).

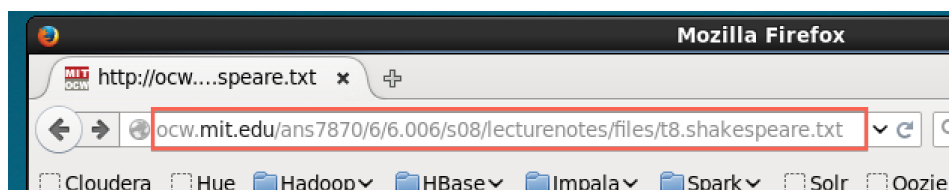
### Instructions

1. **Open a browser.** Open the browser by click on the browser icon on the top left of the screen.

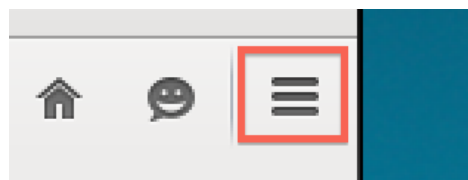


2. **Download the Shakespeare.** We are going to download a text file to copy into HDFS. Enter the following link in the

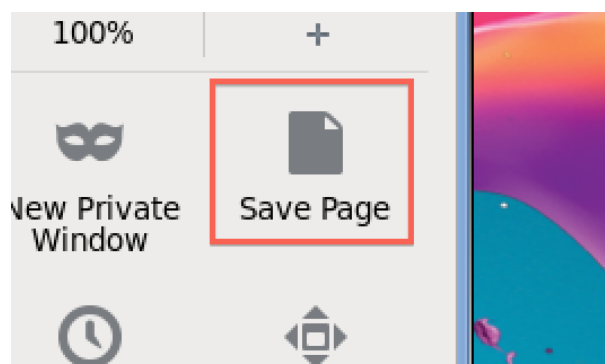
browser: <http://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt>



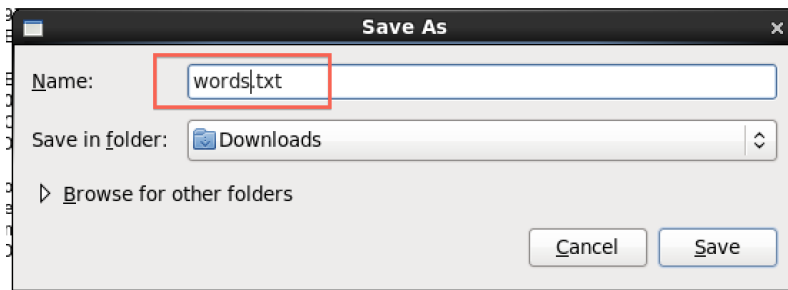
Once the page is loaded, click on the Open menu button.



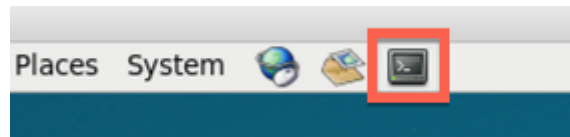
Click on Save Page



Change the output to words.txt and click Save.



2. **Open a terminal shell.** Open a terminal shell by clicking on the square black box on the top left of the screen.



Run `cd Downloads` to change to the Downloads directory.

```
[cloudera@quickstart ~]$ cd Downloads/  
[cloudera@quickstart Downloads]$
```

Run `ls` to see that `words.txt` was saved.

```
[cloudera@quickstart Downloads]$ ls  
words.txt
```

3. **Copy file to HDFS.** Run `hadoop fs -copyFromLocal words.txt` to copy the text file to HDFS.

```
[cloudera@quickstart Downloads]$ hadoop fs -copyFromLocal words.txt  
[cloudera@quickstart Downloads]$
```

4. **Verify file was copied to HDFS.** Run `hadoop fs -ls` to verify the file was copied to HDFS.

```
[cloudera@quickstart Downloads]$ hadoop fs -ls  
Found 1 items  
-rw-r--r-- 1 cloudera cloudera 5458199 2016-02-12 15:14 words.txt  
[cloudera@quickstart Downloads]$
```

5. **Copy a file within HDFS.** You can make a copy of a file in HDFS. Run `hadoop fs -cp words.txt words2.txt` to make a copy of `words.txt` called `words2.txt`

```
[cloudera@quickstart Downloads]$ hadoop fs -cp words.txt words2.txt  
[cloudera@quickstart Downloads]$
```

We can see the new file by running `hadoop fs -ls`

```
[cloudera@quickstart Downloads]$ hadoop fs -ls  
Found 2 items  
-rw-r--r-- 1 cloudera cloudera 5458199 2016-02-12 15:14 words.txt  
-rw-r--r-- 1 cloudera cloudera 5458199 2016-02-12 15:15 words2.txt  
[cloudera@quickstart Downloads]$
```

6. **Copy a file from HDFS.** We can also copy a file from HDFS to the local file system.

Run `hadoop fs -copyToLocal words2.txt` . to copy `words2.txt` to the local directory.

```
[cloudera@quickstart Downloads]$ hadoop fs -copyToLocal words2.txt  
[cloudera@quickstart Downloads]$ █
```

Let's run `ls` to see that the file was copied to see that `words2.txt` is there.

```
[cloudera@quickstart Downloads]$ ls  
words2.txt  words.txt  
[cloudera@quickstart Downloads]$ █
```

7. **Delete a file in HDFS.** Let's the delete `words2.txt` in HDFS. Run `hadoop fs -rm words2.txt`

```
[cloudera@quickstart Downloads]$ hadoop fs -rm words2.txt  
16/02/12 15:17:01 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele  
tion interval = 0 minutes, Emptier interval = 0 minutes.  
Deleted words2.txt  
[cloudera@quickstart Downloads]$ █
```

Run `hadoop fs -ls` to see that the file is gone.

```
[cloudera@quickstart Downloads]$ hadoop fs -ls  
Found 1 items  
-rw-r--r--  1 cloudera cloudera    5458199 2016-02-12 15:14 words.txt  
[cloudera@quickstart Downloads]$ █
```

## Run the WordCount program instructions

### Learning Goals

By the end of this activity, you will be able to:

- Execute the WordCount application.
  - Copy the results from WordCount out of HDFS.
1. Open a terminal shell. Start the Cloudera VM in VirtualBox, if not already running, and open a terminal shell. Detailed instructions for these steps can be found in the previous Readings.
  2. **See example MapReduce programs.** Hadoop comes with several example MapReduce applications. You can see a list of them by running `hadoop jar /usr/jars/hadoop-examples.jar`. We are interested in running WordCount.



```
[cloudera@quickstart ~]$ hadoop jar /usr/jars/hadoop-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
```

The output says that WordCount takes the name of one or more input files and the name of the output directory. Note that these files are in HDFS, not the local file system.

3. **Verify input file exists.** In the previous Reading, we downloaded the complete works of Shakespeare and copied them into HDFS. Let's make sure this file is still in HDFS so we can run WordCount on it. Run `hadoop fs -ls`

```
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--  1 cloudera cloudera    5458199 2016-02-12 15:14 words.txt
[cloudera@quickstart Downloads]$ █
```

4. **See WordCount command line arguments.** We can learn how to run WordCount by examining its command-line arguments. Run `hadoop jar /usr/jars/hadoop-examples.jar wordcount`.

```
[cloudera@quickstart ~]$ hadoop jar /usr/jars/hadoop-examples.jar wordcount
Usage: wordcount <in> [<in>...] <out>
```

5. **Run WordCount.** Run WordCount for words.txt: `hadoop jar /usr/jars/hadoop-examples.jar wordcount words.txt out`

```
[cloudera@quickstart Downloads]$ hadoop jar /usr/jars/hadoop-examples.jar wordcount words.txt out
16/02/12 15:27:34 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/12 15:27:35 INFO input.FileInputFormat: Total input paths to process : 1
16/02/12 15:27:35 INFO mapreduce.JobSubmitter: number of splits:1
```

As WordCount executes, the Hadoop prints the progress in terms of Map and Reduce. When the WordCount is complete, both will say 100%.

```
16/02/12 15:27:46 INFO mapreduce.Job: map 0% reduce 0%
16/02/12 15:27:54 INFO mapreduce.Job: map 100% reduce 0%
16/02/12 15:28:02 INFO mapreduce.Job: map 100% reduce 100%
16/02/12 15:28:02 INFO mapreduce.Job: Job job_1455318527581_0001 completed successfully
```

6. **See WordCount output directory.** Once WordCount is finished, let's verify the output was created. First, let's see that the output directory, *out*, was created in HDFS by running *hadoop fs -ls*

```
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 2 items
drwxr-xr-x  - cloudera cloudera          0 2016-02-12 15:28 out
-rw-r--r--  1 cloudera cloudera    5458199 2016-02-12 15:14 words.txt
[cloudera@quickstart Downloads]$ _
```

We can see there are now two items in HDFS: *words.txt* is the text file that we previously created, and *out* is the directory created by WordCount.

7. **Look inside output directory.** The directory created by WordCount contains several files. Look inside the directory by running *hadoop -fs ls out*

```
[cloudera@quickstart Downloads]$ hadoop fs -ls out
Found 2 items
-rw-r--r--  1 cloudera cloudera          0 2016-02-12 15:28 out/_SUCCESS
-rw-r--r--  1 cloudera cloudera    717768 2016-02-12 15:28 out/part-r-00000
[cloudera@quickstart Downloads]$ _
```

The file *part-r-00000* contains the results from WordCount. The file *\_SUCCESS* means WordCount executed successfully.

8. **Copy WordCount results to local file system.** Copy *part-r-00000* to the local file system by running *hadoop fs -copyToLocal out/part-r-00000 local.txt*

```
[cloudera@quickstart Downloads]$ hadoop fs -copyToLocal out/part-r-00000 local.txt
[cloudera@quickstart Downloads]$
```

9. **View the WordCount results.** View the contents of the results: *more local.txt*

```
[cloudera@quickstart Downloads]$ more local.txt
```

Each line of the results file shows the number of occurrences for a word in the input file. For example, *Accuse* appears four times in the input, but *Accusing* appears only once.

```
Accost- 1
Account 1
Accountant      1
Accounted      1
Accoutred      1
Accurs'd       2
Accurs'd,      1
Accursed       4
Accusativo,    2
Accuse 4
Accusing       1
Acheron 2
Acheron,       1
Aches 1
Achiev'd      1
```

## TODO

1. Follow all steps described above installing VM with Hadoop.
2. Write down all HDFS commands typing “hadoop fs”.
3. Choose two examples of mapreduce application from the list (ask your teacher) and run them on different input file(s) to prove benefits from usage of mapreduce application for big data.
4. Prepare a report for this lab work.

# Lab Work 02 – How to Create a Simple MapReduce Application

## Learning Goals

In this lab work, you will

- Download and install Eclipse IDE
- Create Java projects for Hadoop
- Learn the structure of MapReduce application

## Install Java 8

Download Java 8 from the link: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

a. Set environmental variables:

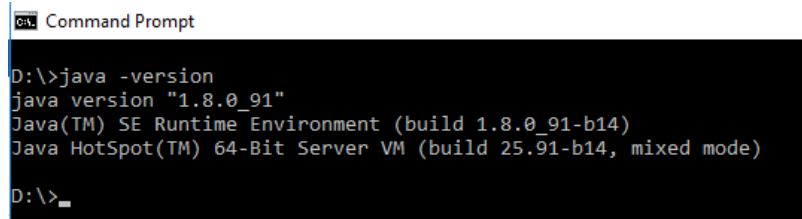
i. User variable:

- Variable: JAVA\_HOME
- Value: C:\java

ii. System variable:

- Variable: PATH
- Value: C:\java\bin

b. Check on cmd, see below:



```
Command Prompt
D:\>java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)
D:\>
```

## Install Eclipse

Download it from the link: <https://eclipse.org/downloads/> and extract it into C drive.

a. Set environmental variables:

i. User variable:

- Variable: ECLIPSE\_HOME
- Value: C:\eclipse

ii. System variable:

- Variable: PATH
- Value: C:\eclipse\bin

b. **Download “hadoop2x-eclipse-plugin-master.”** You will see three Jar files on the path “hadoop2x-eclipse-plugin-master\release.” Copy these three jar files and paste them into “C:\eclipse\dropins.”

c. **Download “slf4j-1.7.21.”** Copy Jar files from this folder and paste them to “C:\eclipse\plugins”. This step may create errors; when you will execute Eclipse, you will see errors like org.apa.....jar file in multiple places. So, now delete these files from all the places except one.

## Download Hadoop

This step is optional if you've already installed Hadoop.

Actually, you will need jar extra libraries from Hadoop folders.

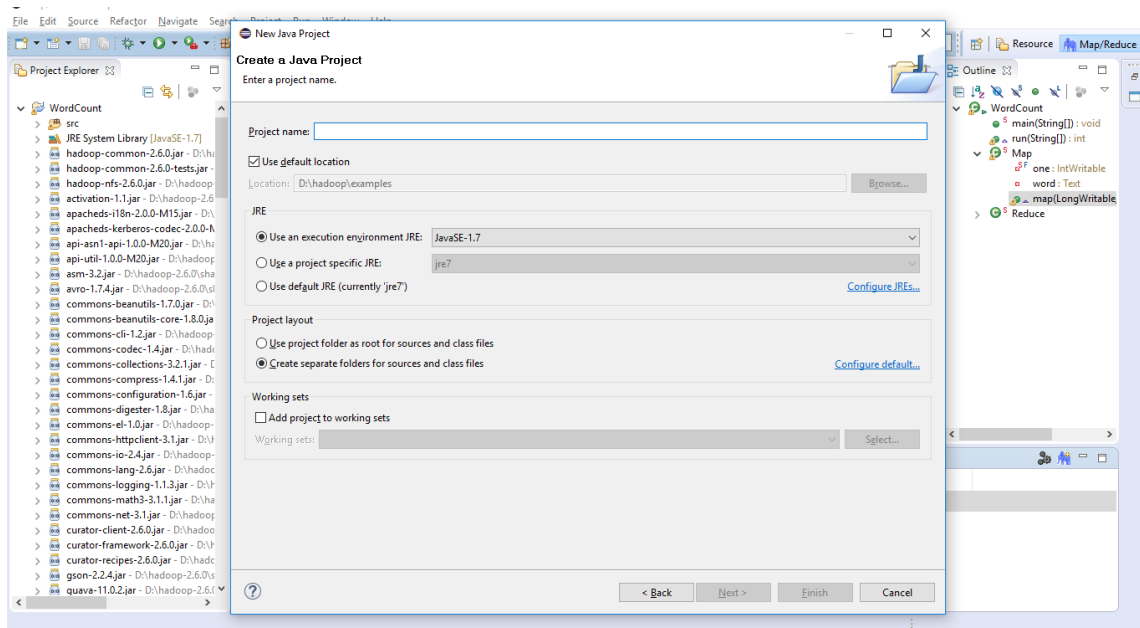
Download Hadoop 3.0.0 from the link:

<http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz>

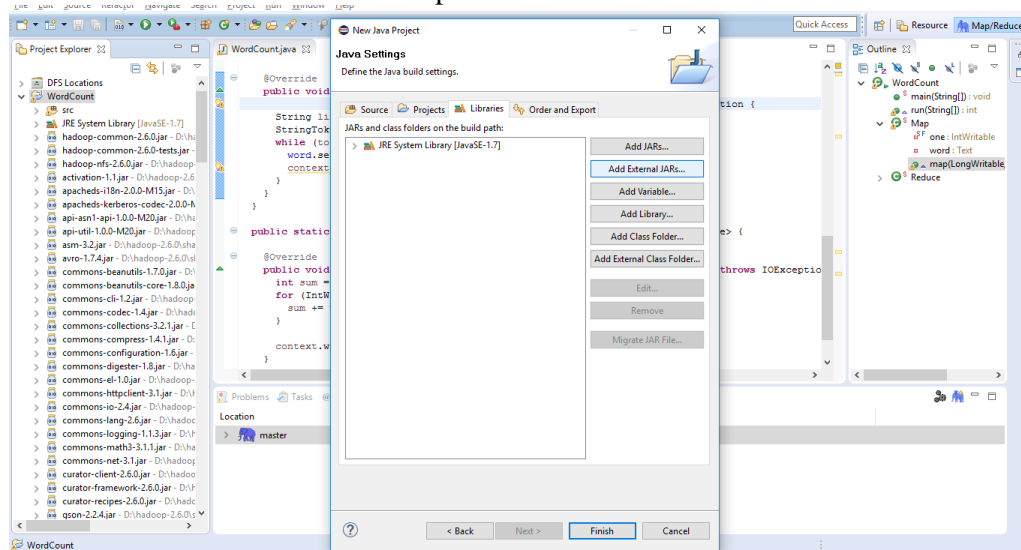
and put extracted files into your local drive (not C:\).

## How to create a new MapReduce project in Eclipse

1. Open Eclipse
2. Click File -> New Project -> Java project



3. Click next and add external Jars for MapReduce.

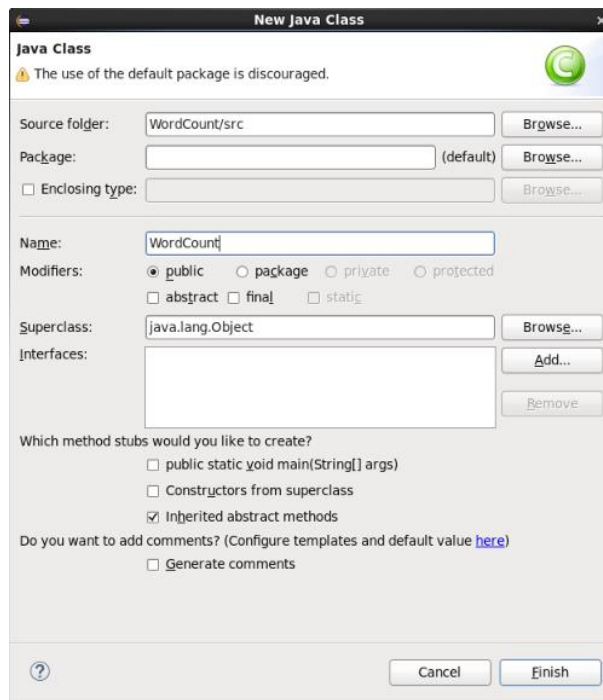


Copy all the Jar files from the locations "D:\hadoop-3.0.0\"

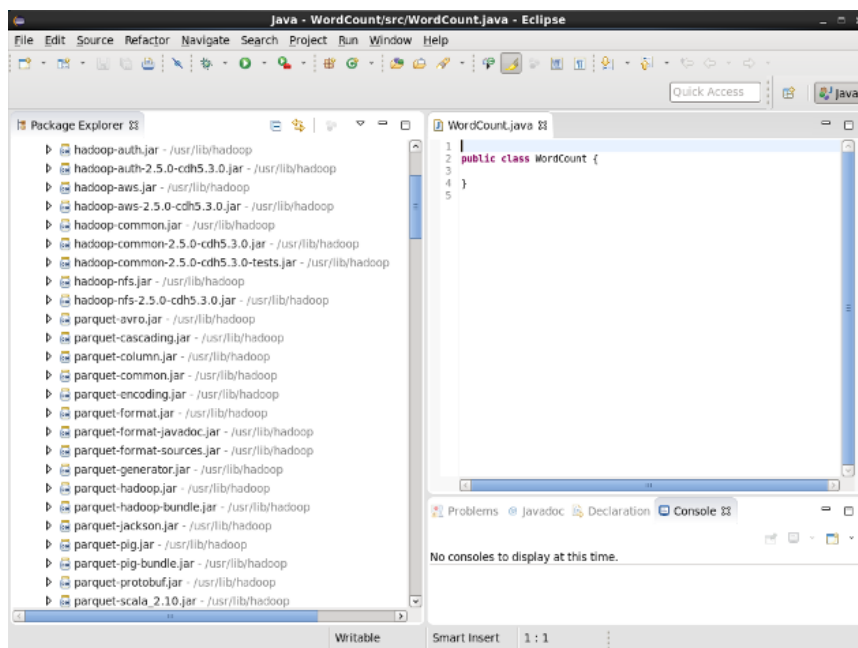
- \share\hadoop\common\lib
- \share\hadoop\mapreduce
- \share\hadoop\mapreduce\lib
- share\hadoop\yarn
- \share\hadoop\yarn\lib

#### 4. Create class file

Right click on source, New -> Class:



Click "Finish".



5. Paste the following code (taken from Hadoop examples):

```

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

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

    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Usage: wordcount <in> [<in>...] <out>");
            System.exit(2);
        }
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        for (int i = 0; i < otherArgs.length - 1; ++i) {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
        }
        FileOutputFormat.setOutputPath(job,

```

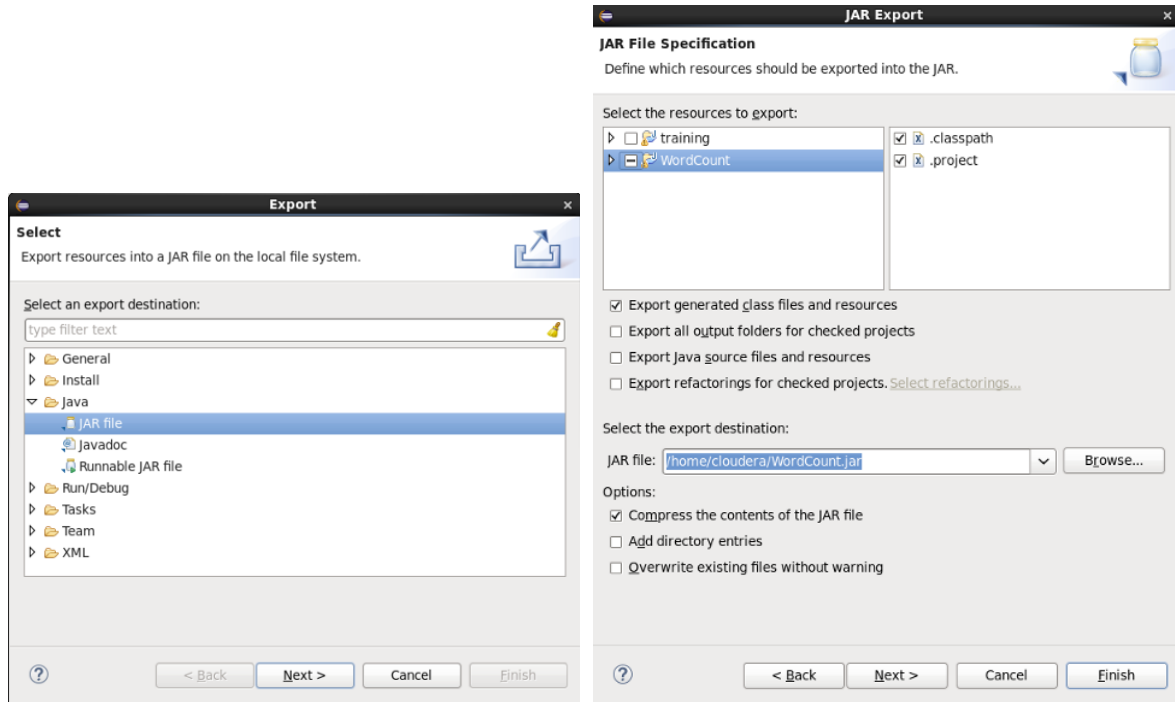
```

        new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
}

```

6. Now we want to export it as a jar.

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



Prepare input file wordcount.txt and copy it into HDFS.

7. Run MapReduce job.

```
>hadoop jar WordCount.jar WordCount wordcount.txt out
```

Application is running

```

> hadoop jar WordCount.jar WordCount wordcount.txt out
INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-
yarn/staging/user/.staging/job_1517319107770_0014
INFO input.FileInputFormat: Total input files to process : 1
INFO mapreduce.JobSubmitter: number of splits:1
INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated.
Instead, use yarn.system-metrics-publisher.enabled
INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1517319107770_0014
INFO mapreduce.JobSubmitter: Executing with tokens: []
INFO conf.Configuration: resource-types.xml not found
INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
INFO impl.YarnClientImpl: Submitted application application_1517319107770_0014
INFO mapreduce.Job: The url to track the job:
http://localhost:8088/proxy/application_1517319107770_0014/
INFO mapreduce.Job: Running job: job_1517319107770_0014
INFO mapreduce.Job: Job job_1517319107770_0014 running in uber mode : false

```



INFO mapreduce.Job: map 0% reduce 0%  
INFO mapreduce.Job: map 100% reduce 0%  
INFO mapreduce.Job: map 100% reduce 100%  
INFO mapreduce.Job: Job job\_1517319107770\_0014 completed successfully  
INFO mapreduce.Job: Counters: 53

#### File System Counters

FILE: Number of bytes read=60  
FILE: Number of bytes written=415397  
FILE: Number of read operations=0  
FILE: Number of large read operations=0  
FILE: Number of write operations=0  
HDFS: Number of bytes read=146  
HDFS: Number of bytes written=38  
HDFS: Number of read operations=8  
HDFS: Number of large read operations=0  
HDFS: Number of write operations=2

#### Job Counters

Launched map tasks=1  
Launched reduce tasks=1  
Data-local map tasks=1  
Total time spent by all maps in occupied slots (ms)=3205  
Total time spent by all reduces in occupied slots (ms)=3406  
Total time spent by all map tasks (ms)=3205  
Total time spent by all reduce tasks (ms)=3406  
Total vcore-milliseconds taken by all map tasks=3205  
Total vcore-milliseconds taken by all reduce tasks=3406  
Total megabyte-milliseconds taken by all map tasks=3935740  
Total megabyte-milliseconds taken by all reduce tasks=4182568

#### Map-Reduce Framework

Map input records=1  
Map output records=4  
Map output bytes=46  
Map output materialized bytes=60  
Input split bytes=117  
Combine input records=4  
Combine output records=4  
Reduce input groups=4  
Reduce shuffle bytes=60  
Reduce input records=4  
Reduce output records=4  
Spilled Records=8  
Shuffled Maps =1  
Failed Shuffles=0  
Merged Map outputs=1  
GC time elapsed (ms)=128  
CPU time spent (ms)=967  
Physical memory (bytes) snapshot=379265024  
Virtual memory (bytes) snapshot=478769152  
Total committed heap usage (bytes)=156237824  
Peak Map Physical memory (bytes)=279134208  
Peak Map Virtual memory (bytes)=337829888  
Peak Reduce Physical memory (bytes)=185753600  
Peak Reduce Virtual memory (bytes)=236560384

#### 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=29
File Output Format Counters
  Bytes Written=38
```

8. Get the result to local disk from HDFS: `hadoop fs -copyToLocal out`.

9. Examine the result in part-r-00000 file.

## TODO

1. Follow all steps described.
2. According to your variant, try to create your own Java project for one of Hadoop samples (see lab work #1).
3. Examine the structure of java file, learning details of implementation.
4. Prepare a report for this lab work.

# Lab Work 03 Querying Relational Data with Postgres

By this end of this activity, you will be able to:

- View table and column definitions, and perform SQL queries in the Postgres shell
- Query the contents of SQL tables
- Filter table rows and columns
- Combine two tables by joining on a column

## Downloading datasets

Step 1. **Start the Cloudera VM.** Most of the Hands On exercises in this course use the Cloudera Virtual Machine, so we will download the datasets onto the VM. Start the VM in VirtualBox and perform the remaining steps in the VM.

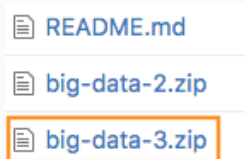
Step 2. **Open a web browser.** Open a web browser by clicking on the web browser icon in the top toolbar.



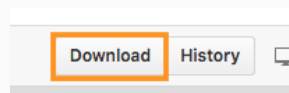
In the web browser, enter the following for the URL:

`http://github.com/words-sdsc/coursera`

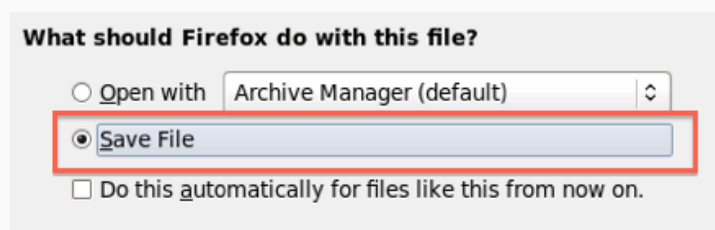
Step 3. **Download the datasets.** Click on *big-data-3.zip*:



Click on the *Download* button:

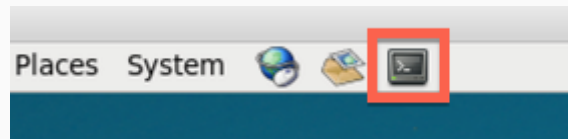


In the dialog, select *Save File*:



Click *OK*, and the file *big-data-3.zip* file will be downloaded to the Downloads directory.

Step 4. **Uncompress the datasets.** Open a terminal shell by clicking on the terminal shell icon in the top toolbar.



In the terminal, run:

```
cd Downloads
unzip -o big-data-3.zip
```

Step 5. **Install tools.** Change directories to *big-data-3* and run *setup.sh* to install tools and libraries.

```
cd big-data-3
./setup.sh
```

During the setup process, Anaconda will give you a series of prompts. First, press *enter* to continue the installation:

```
Welcome to Anaconda3 4.0.0 (by Continuum Analytics, Inc.)
In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> █
```

Next, read and accept the license:

```
Do you approve the license terms? [yes|no]
>>> █
```

Next, press *enter* to accept the default installation location:

```
Anaconda3 will now be installed into this location:
/home/cloudera/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below
```

```
[/home/cloudera/anaconda3] >>> █
```

Next, enter *yes* when it asks if you want to prepend the install location to PATH:

```
installation finished.
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /home/cloudera/.bashrc ? [yes|no]
[no] >>> yes
```

The setup of tools and datasets should continue.

Finally, source `$HOME/.bashrc`:

```
source $HOME/.bashrc
```

## Querying Data

Step 1. **Open a terminal window and start Postgres shell.** Open a terminal window by clicking on the square black box on the top left of the screen.



Next, start the Postgres shell by running `psql`:

```
[cloudera@quickstart big-data-3]$ psql
psql (8.4.20)
Type "help" for help.

cloudera=# █
```

Step 2. **View table and column definitions.** We can list the tables in the database with the `\d` command:

```
cloudera=# \d
          List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+-----
 public | adclicks | table | cloudera
 public | buyclicks | table | cloudera
 public | gameclicks | table | cloudera
(3 rows)
```

The database contains three tables: *adclicks*, *buyclicks*, and *gameclicks*. We can see the column definitions of the *buyclicks* table by running `\d buyclicks`:

```
cloudera=# \d buyclicks
          Table "public.buyclicks"
   Column   |          Type          | Modifiers
-----+-----+-----
 timestamp | timestamp without time zone | not null
 txid       | integer                | not null
 usersessionid | integer                | not null
 team       | integer                | not null
 userid     | integer                | not null
 buyid      | integer                | not null
 price      | double precision       | not null
```

This shows that the *buyclicks* table has seven columns, and what each column name and data type is.

Step 3. **Query table.** We can run the following command to view the contents of the *buyclicks* table:

```
select * from buyclicks;
```

The *select \** means we want to query all the columns, and *from buyclicks* denotes which table to query. Note that all query commands in the Postgres shell must end with a semi-colon.

The result of the query is:

timestamp	txid	usersessionid	team	userid	buyid	price
2016-05-26 15:36:54	6004	5820	9	1300	2	3
2016-05-26 15:36:54	6005	5775	35	868	4	10
2016-05-26 15:36:54	6006	5679	97	819	5	20
2016-05-26 16:36:54	6067	5665	18	121	2	3
2016-05-26 17:06:54	6093	5709	11	2222	5	20
2016-05-26 17:06:54	6094	5798	77	1304	5	20
2016-05-26 18:06:54	6155	5920	9	1027	5	20
2016-05-26 18:06:54	6156	5697	35	2199	2	3
2016-05-26 18:36:54	6183	5893	64	1544	5	20
2016-05-26 18:36:54	6184	5697	35	2199	1	2
2016-05-26 19:36:54	6243	5659	13	1623	4	10

You can hit `<space>` to scroll down, and `q` to quit.

Step 4. **Filter rows and columns.** We can query only the *price* and *userid* columns with the following command:

```
select price, userid from buyclicks;
```

The result of this query is:

price	userid
3	1300
10	868
20	819
3	121
20	2222
20	1304
20	1027
3	2199
20	1544

We can also query rows that match a specific criteria. For example, the following command queries only rows with a price greater than 10:

```
select price, userid from buyclicks where price > 10;
```

The result is:

```

price | userid
-----+-----
20    | 819
20    | 2222
20    | 1304
20    | 1027
20    | 1544
20    | 1065
20    | 2221

```

Step 5. **Perform aggregate operations.** The SQL language provides many aggregate operations. We can calculate the average price:

```

cloudera=# select avg(price) from buyclicks;
          avg
-----
7.26399728537496
(1 row)

```

We can also calculate the total price:

```

cloudera=# select sum(price) from buyclicks;
          sum
-----
21407
(1 row)

```

The complete list of aggregate functions for Postgres 8.4 (the version installed on the Cloudera VM) can be found here: <https://www.postgresql.org/docs/8.4/static/functions-aggregate.html>

Step 6. **Combine two tables.** We combine the contents of two tables by matching or joining on a single column. If we look at the definition of the *adclicks* table:

```

cloudera=# \d adclicks
Table "public.adclicks"
-----+-----+-----
Column          | Type                                | Modifiers
-----+-----+-----
timestamp       | timestamp without time zone        | not null
txid            | integer                             | not null
usersessionid   | integer                             | not null
teamid         | integer                             | not null
userid         | integer                             | not null
adid           | integer                             | not null
adcategory     | character varying(11)              | not null

```

We see that *adclicks* also has a column named *userid*. The following query combines the *adclicks* and *buyclickstable*s on the *userid* column in both tables:

```

select adid, buyid, adclicks.userid
from adclicks join buyclicks on adclicks.userid = buyclicks.userid;

```

This query shows the columns *adid* and *userid* from the *adclicks* table, and the *buyid* column from the *buyclickstable*. The *from adclicks join buyclicks* denotes that we want to combine these two tables, and *on adclicks.userid = buyclicks.userid* denotes which two columns to use when the tables are combined.

The result of the query is:

adid	buyid	userid
2	5	611
2	4	611
2	4	611
2	5	611
2	4	611
2	1	611
21	1	1874
21	1	1874
21	3	1874
21	1	1874
21	2	1874

Enter `\q` to quit the Postgres shell.

## TODO

1. Get familiar with Postgres by visiting its site <https://www.postgresql.org/> and reading the following article "Abbas Butt. Powering Big Data Processing in Postgres with Apache Spark <https://www.enterprisedb.com/blog/powering-big-data-processing-postgres-apache-spark>"
2. Answer the questions:
  - how many records in all tables?
  - describe the scheme of "adclicks" table
  - what categories are present in "adcatagory" column of "adclicks" table? What category is most frequent one?
3. Make a list of aggregate functions for Postgres. Write down an example of usage of aggregate function other than 'avg' or 'sum'.
4. Try to combine all three tables in one query.