Georgia State University | MASTER OF SCIENCE IN ANALYTICS

GSU Institute for Insight Hadoop Cluster New User Manual

Fall 2016

Chris DeBellis
Padhu Sahadevan
Taj Pirzada

Institute for Insight

Conventions used in this manual

Commands are shown monospaced
Shell commands begin with $
`$` is the prompt. Do not enter $.
<Enter> is implied to run commands

Hive commands begin with hive>
`hive>` is the prompt. Do not type hive>.

Scala commands begin with scala>
`scala>` is the prompt. Do not type scala>.

PySpark commands begin with pyspark>
`pyspark>` is the prompt. Do not type pyspark>.

About this manual

This manual is intended to serve as an introductory guide to using the GSU Institute for Insight Hadoop Cluster. At the time of this writing, the cluster consisted of one head node and 3 computing nodes. This manual may become obsolete as the cluster configuration changes.

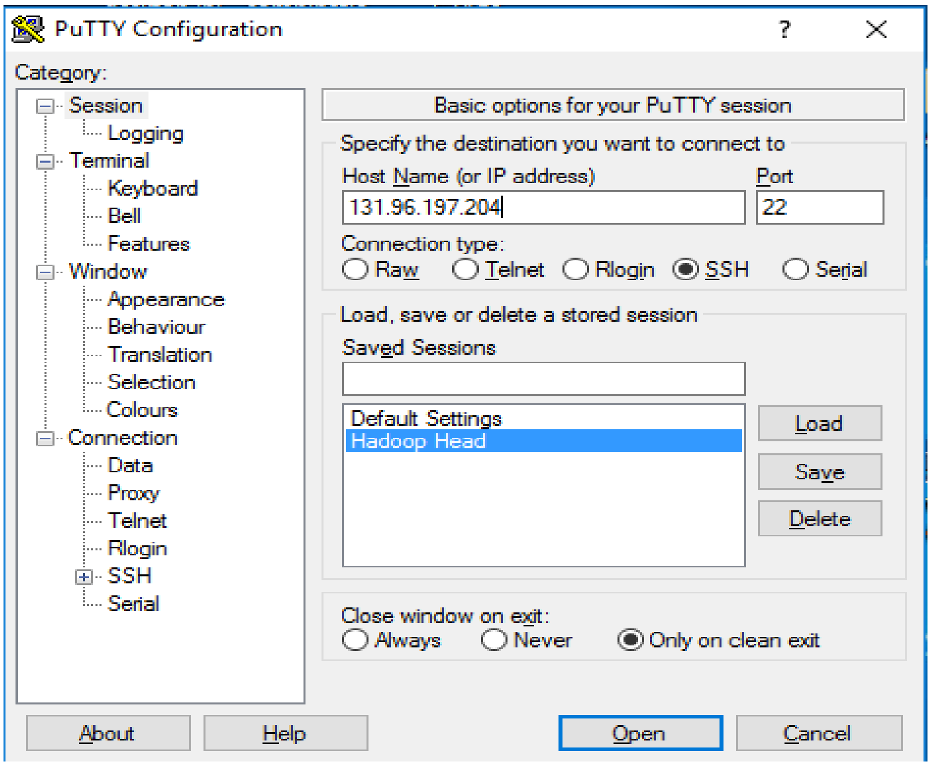
The dataset referenced in this manual is the blood donor information provided by the American Red Cross for MSA 8300 Value through Analytics course taught in Fall 2016. The dataset consisted of 3 primary files.

File	**Number of rows**
allstates.csv	38,955,037
donor_summary912016.csv	14,288,393
donor_deferral912016.csv	4,591,619

Accessing the Hadoop Cluster

<u>Windows</u>

Install and Open "PuTTY" , type in the server IP address



Specify the destination you want to connect to

Host Name (or IP address): 131.96.197.204

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

Default Settings

Hadoop Head

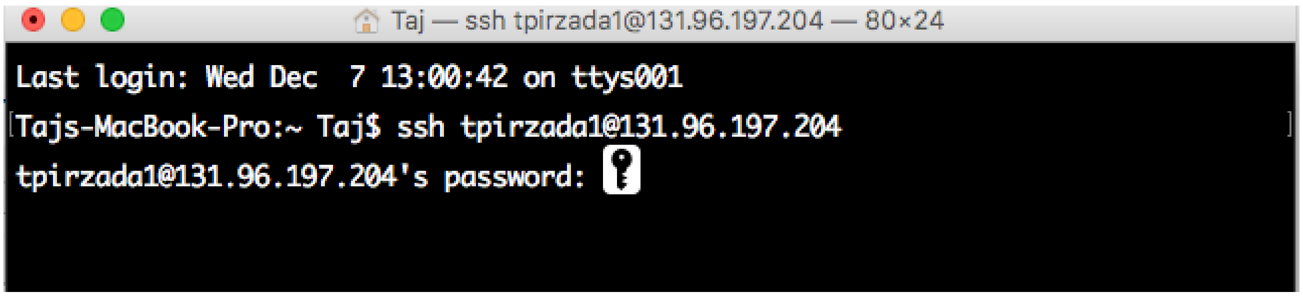
Close window on exit: Always Never [✓ Only on clean exit]

<u>Mac</u>

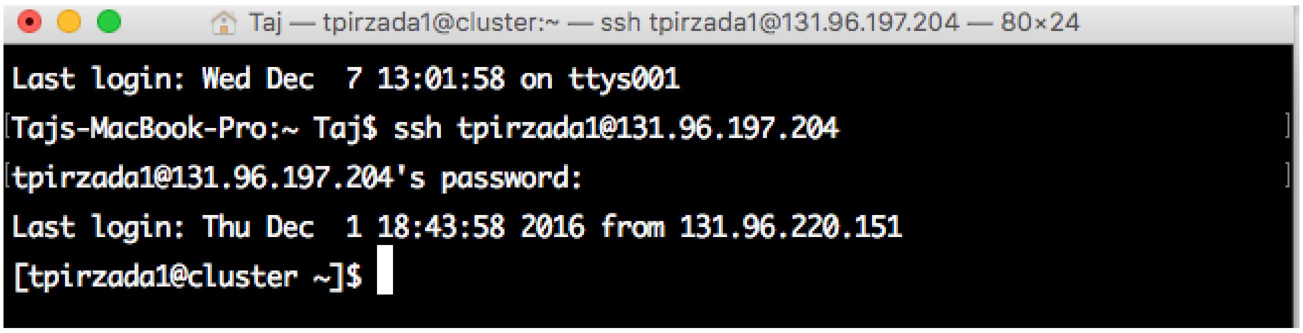
Launch Terminal. If not on GSU network, please VPN into the network using your YourCampusID and Password.

You will be looking at your home folder on your local drive. Type in the Secure Shell command: ssh username@131.96.197.204

You will be prompted for a password. Use the password assigned by the cluster admin.



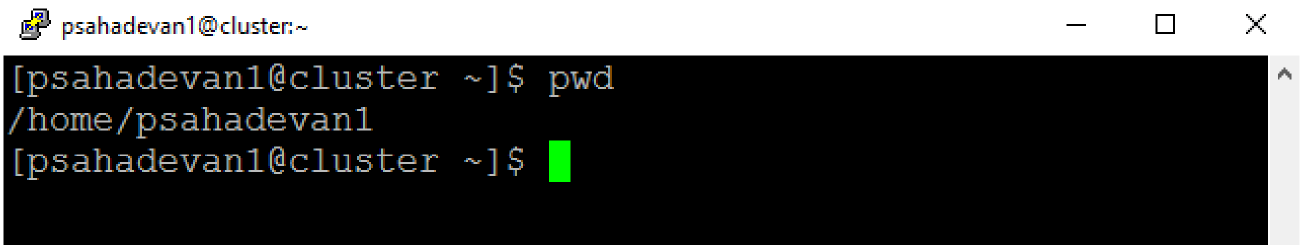
```
Last login: Wed Dec  7 13:00:42 on ttys001
Tajs-MacBook-Pro:~ Taj$ ssh tpirzada1@131.96.197.204
tpirzada1@131.96.197.204's password:
```


Taj — tpirzada1@cluster:~ — ssh tpirzada1@131.96.197.204 — 80×24

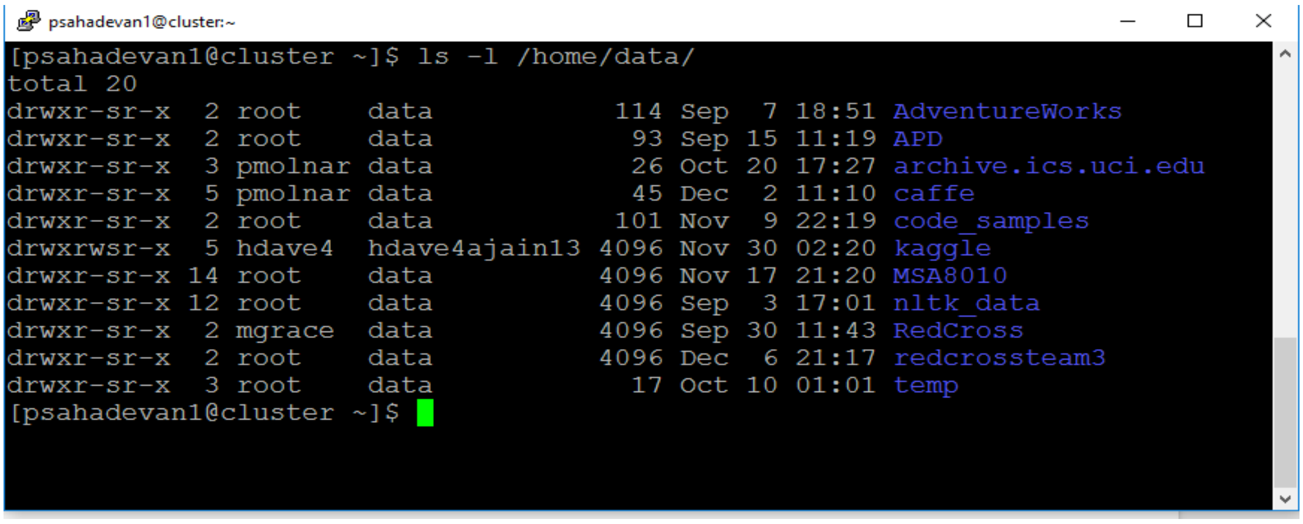
```
Last login: Wed Dec  7 13:01:58 on ttys001
Tajs-MacBook-Pro:~ Taj$ ssh tpirzada1@131.96.197.204
tpirzada1@131.96.197.204's password:
Last login: Thu Dec  1 18:43:58 2016 from 131.96.220.151
[tpirzada1@cluster ~]$
```

You are now logged in to the hadoop cluster.

The default working directory is **/home/YourCampusID**. The storage limit on these user folders is 2 GB. If you want to store data files larger than 2 GB, use one of the common folders like **/home/data/**.



```
[psahadevan1@cluster ~]$ pwd
/home/psahadevan1
[psahadevan1@cluster ~]$
```



```
[psahadevan1@cluster ~]$ ls -l /home/data/
total 20
drwxr-sr-x  2 root    data          114 Sep  7 18:51 AdventureWorks
drwxr-sr-x  2 root    data           93 Sep 15 11:19 APD
drwxr-sr-x  3 pmolnar data           26 Oct 20 17:27 archive.ics.uci.edu
drwxr-sr-x  5 pmolnar data           45 Dec  2 11:10 caffe
drwxr-sr-x  2 root    data          101 Nov  9 22:19 code_samples
drwxrwsr-x  5 hdave4  hdave4ajain13 4096 Nov 30 02:20 kaggle
drwxr-sr-x 14 root    data         4096 Nov 17 21:20 MSA8010
drwxr-sr-x 12 root    data         4096 Sep  3 17:01 nltk_data
drwxr-sr-x  2 mgrace  data         4096 Sep 30 11:43 RedCross
drwxr-sr-x  2 root    data         4096 Dec  6 21:17 redcrossteam3
drwxr-sr-x  3 root    data           17 Oct 10 01:01 temp
[psahadevan1@cluster ~]$
```

Now that you have logged into the cluster, you can run shell commands. If you are not familiar with unix commands, you can refer to this link for quick reference http://mally.stanford.edu/~sr/computing/basic-unix.html

The Hadoop Ecosystem consists of several different technologies. This manual provides an overview of a few of the more useful tools.

Scala

Scala is an object-oriented, functional programming language. The official Scala website is https://www.scala-lang.org/index.html

It is possible to run commands in Scala. To enter the Scala command line:

```
$ spark-shell
```

You can enter scala commands from the scala command prompt

```
scala>
```

Scala can be used to create complex programs and manipulate large datasets. Just to familiarize yourself with Scala, here is a sample routine to calculate pi using the scala language. If you copy and paste this into the command scala command line:

```
scala>
val NUM_SAMPLES=100000
val count = sc.parallelize(1 to NUM_SAMPLES).map{i =>
val x = Math.random()
val y = Math.random()
if (x*x + y*y < 1) 1 else 0
}.reduce(_ + _)
println("Pi is roughly " + 4.0 * count / NUM_SAMPLES)
```

It will return:

```
Pi is approximately 3.1408
```

Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. Hive is one way to store and query large datasets. Hive provides a SQL like interface to store and access data stored in HDFS using HiveQL (Hive Query Language). The official Hive website is https://hive.apache.org/

Enter Hive by typing

```
$ hive
```

Most hive commands follow standard SQL conventions. When entering HiveQL commands from the hive command line, you have to terminate the commands with a semicolon (;). Start by creating a table. Enter the command below to create a table for all_states_new.

```
hive> CREATE TABLE IF NOT EXISTS all_states_new
      (arc_id                   int
      ,donation_dt              date
      ,appt_ind                 int
      ,walk_in_ind              int
      ,prodctv_proc_ind         int
      ,bps                    string
      ,weight                 string
      ,blood_pressure         string
      ,pulse                  string
      ,temp                   string
      ,first_donat_ind        int
      ,deferral_ind           int
      ,donation_ind           int
      ,sponsor_name           string
      ,sponsor_category       string
      ,site_zip               int
      ,donation_type          string
      ,unknown                int
      ,check                  int
      ,zipcode                int
      ,ziptype                string
      ,cityname               string
      ,citytype               string
      ,countyname             string
      ,countyfips             int
      ,statename              string
      ,stateabbr              string
      ,statefips              int
      ,msacode                int
      ,areacode               string
      ,timezone               string
      ,utc                    int
      ,dst                    string
      ,latitude               double
      ,longitude              double
      ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

The create table command above includes the ROW FORMAT options. This is necessary in order to properly load data into the table directly from a csv file. If you don't include the ROW FORMAT options here, the data will not be loaded properly. It was our experience that although all rows would load, all the values were NULL.

To confirm the table schema, use the following command:

```
hive> desc all_states_new;
```

Loading from a CSV file in HDFS into a Hive Table:

HDFS (Hadoop Distributed File System) is a Java-based file system that provides scalable and reliable data storage, and it was designed to span large clusters of commodity servers.

The first step in using HDFS is to load files from the cluster to HDFS.

```
$ hdfs dfs -put /home/data/RedCross/allstates.csv /user/data/allstates.csv
```

Then load the file from HDFS to Hive using the command:

```
hive> LOAD DATA INPATH '/user/data/allstates.csv' INTO TABLE all_state_new;
```

The file will be moved automatically from HDFS to the hive warehouse by the load command. This helps to reduce redundant data. You can now access the data by querying the all_states_new table in hive.

To confirm the data loaded:

```
hive> select count(*) from all_states_new;

Query ID = cdebellis1_20161202121056_fa630b94-2bfb-4c0a-a5f7-129af088e75b
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.

Status: Running (Executing on YARN cluster with App id
application_1477280399907_0445)

--------------------------------------------------------------------------------
      VERTICES    STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED  39          39          0      0      0      0
Reducer 2 ......   SUCCEEDED        1           1      0      0      0      0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 16.68 s
--------------------------------------------------------------------------------
OK
3117887
Time taken: 22.247 seconds, Fetched: 1 row(s)
```

Adding columns:

```
hive> alter table all_states_new add columns (date1 date);

hive> update donor_deferral912016 set date1
=cast(to_date(from_unixtime(unix_timestamp(deferral_start_date,'yyyy/MM/dd')))
as date);
```

Dealing with Dates:

```
hive> create table donor_deferral_new as select arc_id, deferral_start_date,
deferral_end_date,to_date(from_unixtime(unix_timestamp(deferral_start_date,'yyyy
/MM/dd'))) as date1 from donor_deferral912016;

hive> create table donor_deferral_new as select arc_id,
cast(to_date(from_unixtime(unix_timestamp(deferral_start_date,'yyyy/MM/dd'))) as
date) as
deferral_start_date,cast(to_date(from_unixtime(unix_timestamp(deferral_end_date,
'yyyy/MM/dd'))) as date) as deferral_end_date from donor_deferral912016;
```

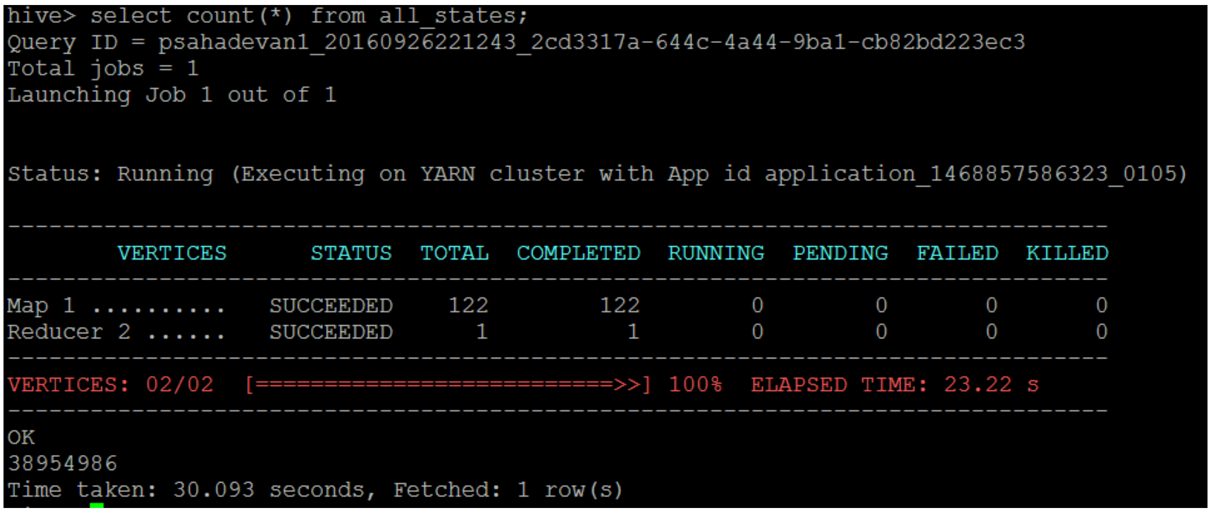
```
hive> select count(*) from all_states A inner join donor_summary B where
A.first_donat_ind =1 and datediff(A.donation_dt,B.birth_dt)/365 between 16 and
21;
```

```
hive> select count(*) from all_states1 A where A.first_donat_ind =1 and
datediff(A.donation_dt,A.birth_dt)/365 between 16 and 21;
```

```
hive> select count(*) from (select arc_id, count(*) from all_states1 group by
arc_id having count(*) > 1);
```

Hive uses Map Reduce:

```
hive> select count(*) from all_states;
```



```
hive> select count(*) from all_states;
Query ID = psahadevan1_20160926221243_2cd3317a-644c-4a44-9ba1-cb82bd223ec3
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1468857586323_0105)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED    122        122        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 23.22 s
--------------------------------------------------------------------------------
OK
38954986
Time taken: 30.093 seconds, Fetched: 1 row(s)
```

Displaying Column Labels:

By default Hive does not provide column names when returning query results. This can be enabled by setting the hive.cli.print.header parameter. This parameter is session based. You have to enter it each time you start the hive command line.

Here is an example of a query that does not return column headers because hive.cli.print.header is not set.

```
hive> select arc_id, donation_dt from all_states limit 5;

8109202    2006-02-13
2262134    2006-04-24
0853154    2006-04-24
7453794    2006-07-24
1431957    2006-07-24
```

Now turn on column headers by entering the command below.

```
hive> set hive.cli.print.header = true;
```

Here is the same query that returns column headers because hive.cli.print.header is now enabled.

```
hive> select arc_id, donation_dt as donation_date from all_states limit 5;

arc_id,    donation_date
8109202    2006-02-13
2262134    2006-04-24
0853154    2006-04-24
7453794    2006-07-24
1431957    2006-07-24
```

Exporting Hive Query Results to Parquet Format (Directory) on HDFS

```
hive> insert overwrite directory '/user/data/all_states/'
      STORED AS PARQUET select * from all_states1;
```

Hive Query examples

Select number of donations by age at the time of donation:

```
hive> select round(datediff(donation_dt, birth_dt)/365), count(*) from
      all_states1 group by round(datediff(donation_dt, birth_dt)/365);
```

Records with negative age:

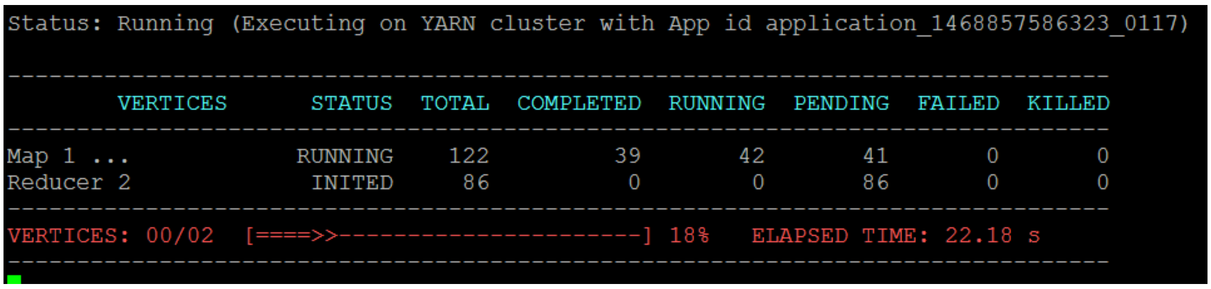
```
hive> select count(*) from all_states1 where datediff(donation_dt, birth_dt)<0;
```

Check the years where donation date precedes the birth date using the query below:

```
hive> select distinct year(donation_dt) from all_states1 where
      datediff(donation_dt, birth_dt) <0;

hive> select year(donation_dt), count(*) from all_states1 where
      datediff(donation_dt, birth_dt) <0 group by year(donation_dt);
```

Notice the number of reducers- it is 86; The “select count(*) “ queries will typically have one reducer (refer to the previous screenshot in the document). This “select distinct “ query requires multiple reducers. A good intuition of how map and reduce programming paradigm works.



```
Status: Running (Executing on YARN cluster with App id application_1468857586323_0117)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ...            RUNNING    122         39       42       41       0       0
Reducer 2             INITED     86          0        0       86       0       0
--------------------------------------------------------------------------------
VERTICES: 00/02  [===>>---------------------] 18%   ELAPSED TIME: 22.18 s
--------------------------------------------------------------------------------
```