Understanding Pig

About this Lab

Objective: To understand Pig scripts and relations

During this Lab: Perform the following steps

File locations: `~/data`

<h4>1. Start the Grunt Shell</h4>

Review the contents of the file `pigdemo.txt` - if you need to you may get it here.

Now, start the Grunt shell:

```
pig
```

Notice that the output includes where the logging for your Pig session will go as well as a statement about connecting to your Hadoop filesystem:

```
[main] INFO org.apache.pig.Main - Logging error messages t
o: /root/devph/labs/demos/pig_1377892197767.log
[main] INFO org.apache.pig.backend.hadoop.executionengine.
 HExecutionEngine - Connecting to hadoop file system at: h
dfs://[server or ip]:8020
```

Now look at the `help` command:

```
grunt> help
```

> *Note: Some of the path given in the above example given, may difer from your lab environment's.*

<!–STEP–>

<h4>2. Make a New Directory</h4>

Notice you can run HDFS commands easily from the Grunt shell. For example, run the `ls` command:

```
grunt> ls
```

Make a new directory named `demos`:

```
grunt> mkdir demos
```

Use `copyFromLocal` to copy the `pigdemo.txt` file into the `demos` folder:

```
grunt> copyFromLocal pigdemo.txt demos/
```

Verify the file was uploaded successfully:

```
grunt> ls demos hdfs://[server or ip]:8020/user/[user-name
]/demos/pigdemo.txt
```

Change the present working directory to `demos`:

```
grunt> cd demos
grunt> pwd hdfs://[server or ip]:8020/user/[user-name]/dem
os
```

> *Note: your particular location por file may be different*

View the contents using the `cat` command:

```
grunt> cat pigdemo.txt
```

Output:

```
SD  Rich
NV  Barry
CO  George
CA  Ulf
IL  Danielle
OH  Tom
CA  manish
CA  Brian
CO  Mark
```

<!–STEP–>

<h4>3. Define a Relation</h4>

Define the `employees` relation, using a schema:

```
grunt> employees = LOAD 'pigdemo.txt' AS (state, name);
```

Demonstrate the `describe` command, which describes what a relation looks like: `grunt> describe employees;`

```
employees: {state: bytearray,name: bytearray}
```

> *Note Fields have a data type, and we will discuss data types later in this unit. Notice that the default data type of a field (if you do not specify one) is bytearray.*

Let’s view the records in the employees relation:

```
grunt> DUMP employees;
```

> *Note this requires a MapReduce job to execute, and the result is a collection of tuples:*

```
(SD,Rich)
(NV,Barry)
(CO,George)
(CA,Ulf)
(IL,Danielle)
(OH,Tom)
(CA,manish)
(CA,Brian)
(CO,Mark)
```

<!-STEP->

<img src="https://user-

images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>4. Filter the Relation by a Field</h4>

Let’s filter the employees whose state field equals CA:

```
grunt> ca_only = FILTER employees BY (state=='CA');
grunt> DUMP ca_only;
```

The output is still tuples, but only the records that match the filter appear:

```
(CA,Ulf)
(CA,manish)
(CA,Brian)
```

<!–STEP–>

<h4>5. Create a Group</h4>

Define a relation that groups the employees by the state field:

```
grunt> emp_group = GROUP employees BY state;
```

Bags represent groups in Pig. A bag is an unordered collection of tuples:

```
grunt> describe emp_group;
emp_group: {group: bytearray,employees: {(state: bytearray
,name: bytearray)}}

```

All records with the same state will be grouped together, as shown by the output of the `emp_group` relation:

```
grunt> DUMP emp_group;
```

The output is:

```
(CA,{(CA,Ulf),(CA,manish),(CA,Brian)})
(CO,{(CO,George),(CO,Mark)})
(IL,{(IL,Danielle)})
(NV,{(NV,Barry)})
(OH,{(OH,Tom)})
(SD,{(SD,Rich)})
```

> *Note Tuples are displayed in parentheses. Curly braces represent bags.*

<!-STEP->

<h4>6. The STORE Command</h4>

The `DUMP` command dumps the contents of a relation to the console. The `STORE` command sends the output to a folder in HDFS. For example:

```
grunt> STORE emp_group INTO 'emp_group';
```

> *Note at the end of the MapReduce job that no records are output to the console.*

Verify that a new folder is created:

```
grunt> ls

hdfs://[server or ip]:8020/user/[user-name]/demos/emp_grou
p<dir>
hdfs://[server or ip]:8020/user/[user-name]/demos/pigdemo.
txt<r 1>89
```

View the contents of the output file:

```
grunt> cat emp_group/part-r-00000
CA      {(CA,Ulf),(CA,manish),(CA,Brian)}
CO      {(CO,George),(CO,Mark)}
IL      {(IL,Danielle)}
NV      {(NV,Barry)}
OH      {(OH,Tom)}
SD      {(SD,Rich)}
```

> *Note that the fields of the records (which in this case is the state field followed by a bag) are separated by a tab character, which is the default delimiter in Pig. Use the PigStorage object to specify a different delimiter*

```
grunt> STORE emp_group INTO 'emp_group_csv' USING PigStora
ge(',');
```

To view the results:

```
grunt > ls
grunt > cat emp_group_csv/part-r-00000
```

<!–STEP–>

<img src="https://user-

images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>7. Show All Aliases</h4>

The aliases command shows a list of currently defined aliases:

```
grunt> aliases;
aliases: [ca_only, emp_group, employees]
```

There will be a couple of additional numeric aliases created by the system for internal use. Please ignore them.

<!–STEP–>

<h4>8. Monitor the Pig Jobs</h4>

Point your browser to the JobHistory UI at `http://[server or ip]:19888`.

View the list of jobs, which should contain the MapReduce jobs that were executed from your Pig Latin code in the Grunt shell.

Notice you can view the log files of the ApplicationMaster and also each map and reduce task.

> *Note Three commands trigger a logical plan to be converted to a physical plan and execute as a MapReduce job:*

```
STORE, DUMP, and ILLUSTRATE.
```

Result

You are finished! Pig’s pretty powerful.