HDFS Commands Lab

Getting Your Bearings

Let's start as we usually do by getting our bearings. From the shell of your Hadoop client, try these:

1. Type `$ hdfs dfs` and observe the output. Notice this command is the same as `$ hadoop fs` The general syntax of this command is `$ hdfs dfs <command>` where `<command>` is one of the many commands listed in the output.
2. Let's see what files are on HDFS: `$ hdfs dfs -ls`. It looks similar to the Linux `ls -l` command except it's displaying the files on the Hadoop file system not the local file system. You might be wondering *where* we are in the Hadoop file system? The root? Nope. The answer is the `ischool` home directory. Why? The current linux user is `ischool`
3. The *absolute path* to the `ischool` home directory on HDFS `/user/ischool` notice this is different from the linux file system default of `/home/ischool` Thus this command is the same as the previous. Type: `$ hdfs dfs -ls /user/ischool` The output should be the same as `$ hdfs dfs -ls`
4. Let's see what's in the root of the HDFS file system. Type `$ hdfs dfs -ls /` **NOTE:** For the Hadoop file system, there's no `pwd` or `cd` commands. That is because there is no shell or command prompt for HDFS. There is only executing Hadoop commands through the Linux command prompt (or any other Hadoop client for that matter).

Working With Files in HDFS

For next two sections of this walk-through we will use the `grades` data set in the `datasets` folder on your Hadoop client.

1. Let's begin by creating a directory in HDFS for this lab. All of our work will go in this directory, type: `% hdfs dfs -mkdir unit06lab1`
2. We use the `-put` command to copy files into HDFS. Let's load the `fall2015` grades into our `unit06lab1` folder on HDFS: `$ hdfs dfs -put datasets/grades/fall2015.tsv unit06lab1/fall2015.tsv`

3. Let's check to make sure we did that correctly, by listing the files in `unit06lab1` type: `$ hdfs dfs -ls unit06lab1` You should see this:

```
Found 1 items
-rw-r--r--   3 ischool ischool        122 2016-05-24 21:23 unit03lab1/fall2015.tsv
```

4. What happens if we repeat the same command? Will it overwrite the file on HDFS? Let's try it out: `$ hdfs dfs -put datasets/grades/fall2015.tsv unit06lab1/fall2015.tsv` You'll notice it does not work and we get an error: **put: 'fall2015.tsv': File exists**. HDFS will not overwrite files by design.
5. Suppose we want to view what's in the file on HDFS? For that we use the `-cat` command for this. Type: `$ hdfs dfs -cat unit06lab1/fall2015.tsv` to view the contents of the file on HDFS. There should be 5 lines in the file, output should look like this:

```
2015  Fall  IST101  1 A
2015  Fall  IST195  3 A
2015  Fall  IST233  3 B+
2015  Fall  SOC101  3 A-
2015  Fall  MAT221  3 C+
```

6. Let's make a copy of the file on HDFS, type: `$ hdfs dfs -cp unit06lab1/fall2015.tsv unit06lab1/grades.tsv` then do an `-ls` on `unit06lab1` to verify there are now two files like this:

```
Found 2 items
-rw-r--r--   3 ischool ischool        122 2016-05-24 21:23 unit03lab1/fall2015.tsv
-rw-r--r--   3 ischool ischool        122 2016-05-24 21:31 unit03lab1/grades.tsv
```

7. Now let's delete the `fall2015.tsv` on HDFS, type: `$ hdfs dfs -rm unit06lab1/fall2015.tsv` An `-ls` will now reveal a single file `grades.tsv` in the `unit06lab1` folder.
8. How do we download the file from HDFS back to the client? That's the `-get` command. Type: `$ hdfs dfs -get unit06lab1/grades.tsv grades.tsv` You will now notice the file `grades.tsv` is local. Verify with `$ cat grades.tsv`
9. Before we move on to the next step, let's remove the `grades.tsv` from HDFS. `$ hdfs dfs -rm unit06lab1/grades.tsv`
10. Not **really** deleting, is it? This is the second time we've deleted something, and you've probably noticed that when we delete a file from HDFS it is realling **moving** the file to the *Trash* folder. You can always get a listing of files in the `.Trash` folder to see what's there, type: `$ hdfs dfs -ls .Trash/Current/user/ischool` You'll see the two files we deleted are here.

NOTE: *HDFS Trash tips:* Files remain in the Trash folder for 24 hours. We can use the `-cp` or `-mv` commands to copy or move the files out of the trash. We can also add the `-skipTrash` option on the `-rm` command to delete the file immediately.

HDFS Best Practices

A common practice for HDFS storage is to place files of a common data set into the same folder. Thus the `fall2015.tsv`, `spring2015.tsv` and `fall2016.tsv` files should all be in the same `grades` folder as they represent the same thing. We will use this practice frequently in future labs.

1. Let's start by making a folder for the data set:
   `$ hdfs dfs -mkdir unit06lab1/grades`
2. Let's upload all the files into the grades folder:
   `$ hdfs dfs -put datasets/grades/* unit06lab1/grades` the asterisk `*` represents all the files in the source folder. There should be 3 files in the folder.
3. Verify the files are there with an `-ls` command:
   `$ hdfs dfs -ls unit06lab1/grades` you should see the three files:

```
Found 3 items
-rw-r--r--   3 ischool ischool        122 2016-05-30 18:41 unit03lab1/grades/fall2015.tsv
-rw-r--r--   3 ischool ischool        122 2016-05-30 18:41 unit03lab1/grades/fall2016.tsv
-rw-r--r--   3 ischool ischool        106 2016-05-30 18:41 unit03lab1/grades/spring2016.tsv
```

4. We know we can `-cat` one file from HDFS, but now that all the grades are in a folder we can few the contents of the folder as a single file:
   `$ hdfs dfs -cat unit06lab1/grades/*` For output, you should see all grades for Fall 2015, Spring 2016 and Fall 2016.
5. There's also an HDFS command to `-get` the files and return them as a single file, in this case `allgrades.tsv` `$ hdfs dfs -getmerge unit06lab1/grades/* allgrades.tsv`
   Since a MapReduce job creates one file per reducer, this command is useful for exporting that output from HDFS.
6. You can verify the merged file, which is now on the local file system is the contents of all the files on HDFS:
   `$ cat allgrades.tsv` should be the same as the `-cat` command before.

HDFS Block storage

We learned through class lecture that HDFS splits files into blocks and those blocks are then distributed to data nodes managed by HDFS. Let's explore how this works:

1. From your home directory, verify the data file `sr20160401.csv` is present in your `datasets` folder. Type: `$ ls -l datasets/nyc311` notice the size of the file is 3.9 MB.
2. As is customary to do, let's make an HDFS directory for this data set:
   `$ hdfs dfs -mkdir nyc311`
3. Let's try to split this file into 500 byte blocks as we upload it to HDFS, type:
   `$ hdfs dfs -D dfs.blocksize=500 -put datasets/nyc311/sr20160401.csv nyc311/`
   You'll get an error: **Specificed block size is less than configured minimum value (dfs.namenode.fs-limits.min-block-size): 500 < 1048576** This error tells the block size of 500 is lower than the configured size for HDFS which is currently set to 1048576.
4. Okay. Let's try a larger block size, like 1200500, try this:
   `$ hdfs dfs -D dfs.blocksize=1200500 -put datasets/nyc311/sr20160401.csv nyc311/` And we get another error: **Invalid values: dfs.bytes-per-checksum (=512) must divide block size (=1200500)** HDFS is telling us that the blocksize we specify must be at least 1048576 **and** a multiple of 512.
5. Let's do it one last time with a block size of 1048576, type:
   `$ hdfs dfs -D dfs.blocksize=1048576 -put datasets/nyc311/sr20160401.csv nyc311/`
6. Hooray! The command worked. Verify the file is in the `nyc311` folder on HDFS:
   `$ hdfs dfs -ls nyc311/`
7. Let's take a look at the blocks that make up this file, type:
   `$ hdfs fsck nyc311/sr20160401.csv` You should see **4** blocks. All the blocks are under-replicated. This makes sense since we only have a single node in our Hadoop cluster and a minimum of **3** are requried for replication. The blocks are implemented as files on the actual data node.

Test Yourself

1. Why are there no `pwd` or `cd` HDFS commands?
2. Explain what happens to a file in HDFS when you delete it?

3. What is the convention for storing files in HDFS, specifically which files belong in which folders?
4. What is the minimum block size in HDFS? Block sizes must be a multiple of which number?
5. What is the default replication factor in HDFS? Why are all blocks on your Hadoop cluster under-replicated?

Part 3: On Your own

Exercises

1. Upload all the `.json` files from `datasets/redditnews` into a `redditnews` folder on HDFS.
2. Upload the `clickstream` dataset to HDFS. Create a `clickstream` folder and then two folders `iplookup` and `logs` inside that folder. Upload all the `.log` files to the `logs` folder, and the `ip_lookup.csv` file to the `iplookups` folder.
3. The `datasets/customers` folder has two files in it. Examine the contents of each file and then devise a strategy for how these files should be uploaded to HDFS. Remember HDFS best practices as you decide how to process the files.