


Getting Familiar

Workshop Exercise: Getting Familiar With Hadoop

This project is designed to give you hands-on experience getting acquainted with the Hadoop tools.

The exercises for this training workshop will be conducted within a virtual machine running Ubuntu 8.10 (Intrepid Ibex). This virtual machine has been preconfigured as a self-contained "pseudo-distributed" Hadoop cluster. This means that it contains all the daemons required for distributed execution--just as if it were running on a cluster in a data center--but is actually operating by itself.

Decompress the `cloudera-training-vm.tar.bz2` archive and you should see a directory named `cloudera-training-0.2/`. In this directory is a file named `cloudera-training-0.2.vmx`. You should be able to double-click the `.vmx` file to launch the virtual machine.

 This assumes that you have already installed the pre-required software from VMWare. Windows and Linux users should download VMWare Player at <http://www.vmware.com/products/player/>. MacOSX users should purchase VMWare Fusion (\$50) or download the free 30-day trial at <http://www.vmware.com/products/fusion/>.

When you start the virtual machine, it should automatically log you into Gnome (XWindows). You are logged in as a user named `training`.

This user is a sudoer, so you can perform any actions necessary to configure your virtual machine. For the purposes of this training session, none are expected, but if you take this home, you might want to adjust things. You do not need a password to run commands via `sudo`. For reference, the user's password is also `training`.

Hadoop

Hadoop is already installed, configured, and running on the virtual machine. We are running a slightly-modified version of Hadoop 0.18.3, which is part of the Cloudera Hadoop Distribution (version 0.2.1). This contains Hadoop as

well as some related programs (Pig and Hive and some other support utilities).

Hadoop is installed in the `/usr/share/cloudera/hadoop` directory. We'll refer to this as `$HADOOP_HOME` throughout this document. If you start a terminal (double-click the icon on the desktop), `$HADOOP_HOME` will already be in your environment.

Most of your interaction with the system will be through a command-line wrapper called `$HADOOP_HOME/bin/hadoop`. The command `hadoop` is set as an alias to this longer name.

If you start a terminal and just run this program with no arguments, it will print a help message. To try this, run the following command:

```
training@cloudera-training:~$ hadoop
```

This wrapper program can access several different subsystems of Hadoop. Each of these subsystems has one or more commands associated with it. You'll need to become familiar with three of these to do most of your work.

This activity reviews the basics of this process.

HDFS Access

The first thing we'll want to do is use the Hadoop Distributed File System, HDFS. This is where data is stored in a way that makes it accessible to Hadoop MapReduce programs. The subsystem associated with the wrapper program is `fs`.

If you run:

```
training@cloudera-training:~$ hadoop fs
```

...you'll see a help message describing all the commands associated with this subsystem. Let's try to look at the current contents of the filesystem:

```
training@cloudera-training:~$ hadoop fs -ls /
```

This will show you the contents of the root directory. This should have two entries. One of these is `/user`. Individual users have a "home" directory

under this prefix, consisting of their username. (Your home directory is `/user/training`.) Try viewing the contents of the `/user` directory by running:

```
training@cloudera-training:~$ hadoop fs -ls /user
```

You can see your home directory in there. What happens if you run the following?

```
training@cloudera-training:~$ hadoop fs -ls /user/training
```

There's no files in there, so it silently exits. This is different than if you ran `hadoop fs -ls /foo`, which refers to a directory you know doesn't exist.

Note that the directory structure within HDFS has nothing to do with the directory structure of the local filesystem; they're completely separate namespaces.

Uploading Files

Besides browsing the existing filesystem, another important thing can do with the FsShell (as this subsystem is properly referred to), is to upload new data into it.

If you perform a "regular" `ls` command on your home directory in the local filesystem, you'll see a few files and directories, including two named `shakespeare.tar.gz` and `shakespeare-streaming.tar.gz`. Both of these files include the complete works of Shakespeare in text format, although they have been formatted slightly differently for convenience. We're going to work with the regular `shakespeare.tar.gz` for now.

Unzip this by running:

```
training@cloudera-training:~$ tar vzxvf shakespeare.tar.gz
```

This will inflate to a directory named `input/` containing several files. We can insert this directory into HDFS by running:

```
training@cloudera-training:~$ hadoop fs -put input /user/training/input
```

This will copy the local `input/` directory into a remote directory named `/user/training/input`. Try to read the contents of your HDFS home directory now, to confirm this:

```
training@cloudera-training:~$ hadoop fs -ls /user/training
```

You should see an entry for the `input` directory. Now, try the same `fs -ls` command but with no pathname argument:

```
training@cloudera-training:~$ hadoop fs -ls
```

You should get the same results back.

If you don't pass a directory name to the `-ls` command, it assumes you mean your home directory, `/user/$USER` a.k.a `/user/training`.

Relative paths

If you pass any relative (non-absolute) paths to FsShell commands (or use relative paths in MapReduce programs), they will implicitly be relative to this base directory. For example, you can see the contents of the uploaded `input` directory by running:

```
training@cloudera-training:~$ hadoop fs -ls input
```

You also *could have* uploaded the Shakespeare files into HDFS by running:

```
training@cloudera-training:~$ hadoop fs -put input input
```

Viewing Files

Now let's view one of the files in this directory:

```
training@cloudera-training:~$ hadoop fs -cat input/sonnets
```

It will then print the contents of the *sonnets* file to the terminal. This command is handy for viewing the output of your MapReduce programs. Very often, an individual output file of a MapReduce program will be very large, and you don't want to dump the entire thing to the terminal. In that case, you should pipe the output of the `-cat` command into either `head` (if you only want to see a couple lines) or `less` if you want the ability to control a scrolling buffer.

e.g.:

```
training@cloudera-training:~$ hadoop fs -cat input/sonnets | less
```

If you want to download a file and manipulate it in the local filesystem (e.g., with a text editor), the `-get` command takes two arguments, an HDFS path, and a local path, and copies the HDFS contents into the local filesystem.

Other Commands

There are several other commands associated with the FsShell subsystem; these can perform most common filesystem manipulations (`rm`, `mv`, `cp`, `mkdir`, etc.). Try playing around with a few of these if you'd like.

Running a MapReduce Program

The same wrapper program is used to launch MapReduce jobs. The source code for a job is contained in a compiled `.jar` file. Hadoop will load the JAR into HDFS and distribute it to the worker nodes, where the individual tasks of the MapReduce job will be executed.

Hadoop ships with some example MapReduce programs to run. One of these is a distributed *grep* application which reads through a set of source documents for strings which match a user-provided regular expression. Let's run this on the input documents you loaded in the previous segment:

```
training@cloudera-training:~$ hadoop jar $HADOOP_HOME/hadoop-0.18.3-patched-examples.jar grep input grep_output "[Ww]herefore"
```

The format of this command is `hadoop jar jarfilename [arguments to pass to program here]`

This program will search for all instances of the word "wherefore" in the Shakespeare corpus, regardless of initial capitalization, and report back the number of occurrences of each matching string. The strings which matched the regular expression argument, as well as their frequency of occurrence, will be written to files in the output directory, called `grep_output`. Note that this behavior is different than what the UNIX `grep` tool typically does.

Now, use `fs -ls` on the output directory to see the list of output files. Use `fs -cat` to view individual output files. (There should be one, named `part-00000`.)

If you'd like, try to run another MapReduce job, substituting a different regular expression for `"[Ww]herefore"` in the command string. If you would like to send your results to a different output directory, substitute that

output directory name for `grep_output` above. If you want to send your output to the `grep_output` directory again, you'll need to remove the existing directory named `grep_output` first. (Hadoop will refuse to run the job and stop with an error message if its output directory already exists.)

To remove the `grep_output` directory, run:

```
training@cloudera-training:~$ hadoop fs -rmr grep_output
```

The `-rmr` command recursively removes entities from HDFS.

Status pages

Hadoop also provides a web-based status tool to see what's occurring in your cluster. We'll cover this in more detail later, but for now, if you'd like to see what jobs have been run, open the web browser. The home page is set to point to the status page. Try starting a job and, while it's still running, refresh the status page. You'll see the running job listed; clicking on it will show you more intermediate status information.

Stopping MapReduce Jobs

Finally, an important capability is stopping jobs that are already running. This is useful if, for example, you accidentally introduced an infinite loop into your Mapper, etc. An important point to remember is that pressing `^C` to kill the current process (which is displaying the MapReduce job's progress) does **not** actually stop the job itself. The MapReduce job, once submitted to the Hadoop daemons, runs independently of any initiating process. Loosing the connection to the initiating process does not kill a MapReduce job.

Instead, you need to tell the Hadoop JobTracker to stop the job.

Start another `grep` job like you did in the previous section. While it's running, pop open another terminal, and run:

```
training@cloudera-training:~$ hadoop job -list
```

This will list the job id's of all running jobs. A job id looks something like `job_200902131742_0002`. Copy the job id, and then kill the job by running:

```
training@cloudera-training:~$ hadoop job -kill (jobid)
```

The JobTracker will kill the job, and the program running in the original terminal, reporting its progress, will inform you that the job has failed.

If you need to cancel a job and restart it (e.g., because you immediately thought of a bugfix after launching a job), make sure to properly kill the old job first.