

Running A MapReduce WordCount in Eclipse and Terminal

Cheng Chen
cchen224@uic.edu

Reading Instructions

1. Text following dollar sign (\$) in this tutorial is command in Terminal. The following two examples are for changing directory to home folder and listing files respectively.

```
$ cd ~  
$ ls -l
```

2. Shaded text following vi/vim command is what you need to attach at the end of the target file in vi/vim editor. If you are not familiar with vi/vim editor, you can also use text editor by opening the target file in your Finder/File Browser. Below is an example for adding 'bbb' into aaa.txt file under home directory.

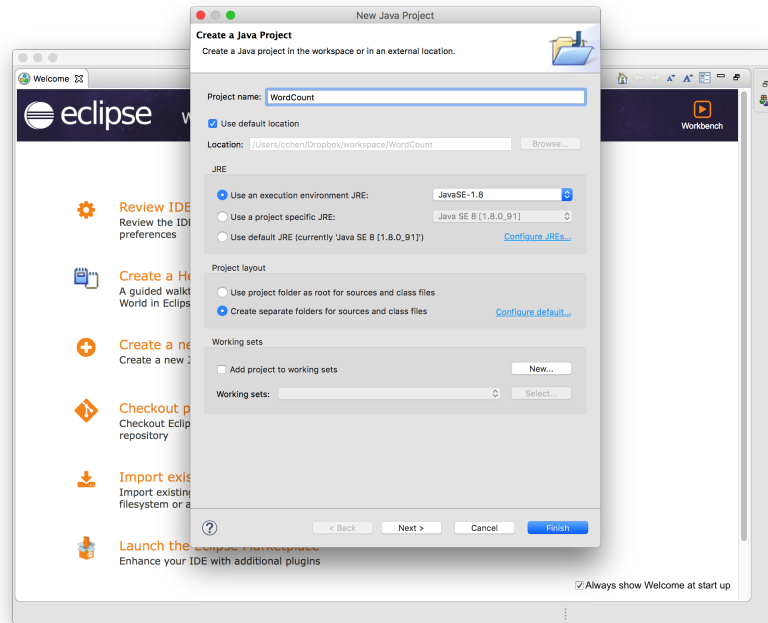
```
$ vi ~/aaa.txt  
  
bbb
```

3. Steps mentioning **Edit** some file and followed by a framed text is same as previous instruction: adding the framed text into the target file by using vi/vim/text editor. Sometimes the target file doesn't exist, then you may create a new one.
4. Some clarification for file path:
 - Absolute path is the path starting with '/';
 - Relative path is the path starting with some folder/file;
 - Path starting with \$ is pre-defined path in environment variable (e.g. \$HADOOP_HOME);
 - ~ is your home folder.
5. Blue text contains hyperlink that you can click. For example, [course website](#)
6. Feel free to [email TA](#) or post threads on Discussion Board if you still have questions after reviewing the reading instructions.

1 Run WordCount in Eclipse

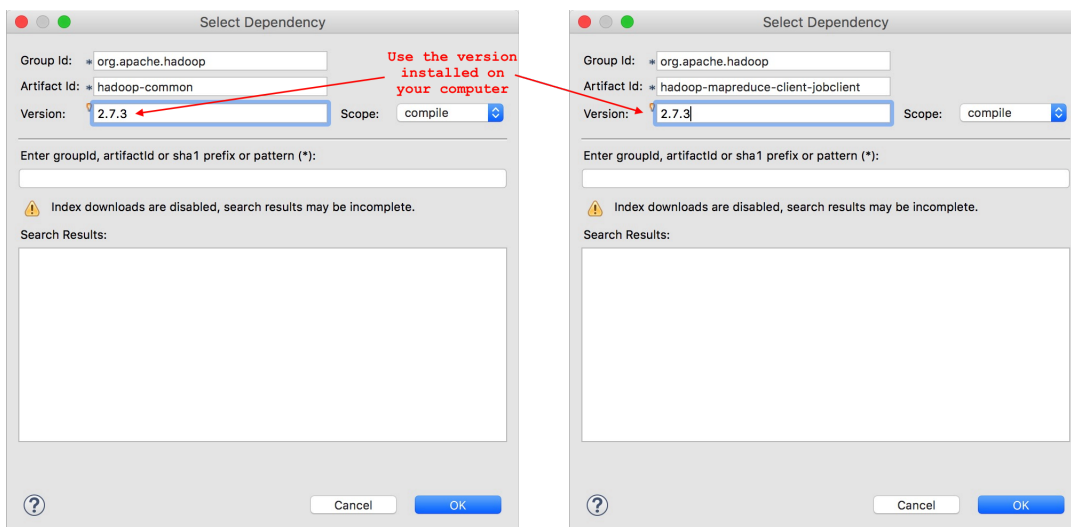
1.1 Open Eclipse and Create a Java Project

1. Click File → New → Java Project.
2. Name your project (e.g. WordCount) and click Finish



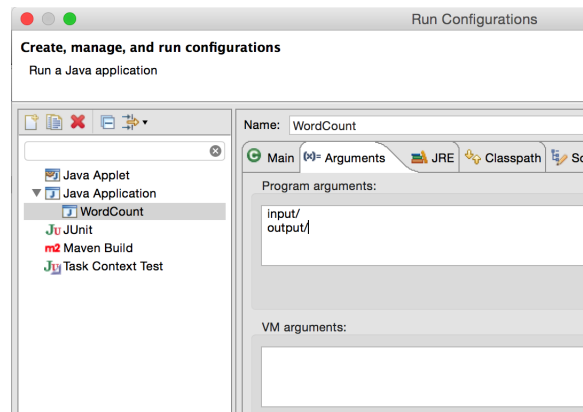
1.2 Convert to Maven Project and Configure

1. Right click your project → Configure → Convert to Maven Project. Then click Finish.
2. You will find pom.xml file generated after Step 1 and open it in Eclipse.
3. Click Dependencies at the bottom and add two hadoop dependencies, **hadoop-common** and **hadoop-mapreduce-client-jobclient** with proper Hadoop version (use 2.6.0 for CDH 5.8). Then save the pox.xml file (press cmd+s or ctrl+s).

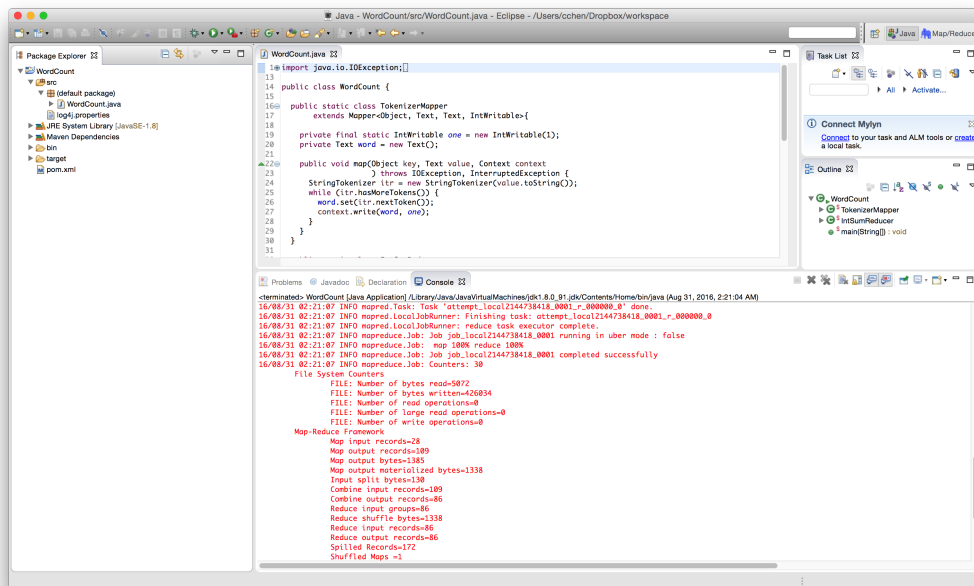


1.3 Execution

1. Create a java file under src and write your own Hadoop MapReduce program.
See code in [Appendix](#) or official [MapReduce Tutorial](#) website.
2. Copy `$HADOOP_HOME/etc/hadoop/log4j.properties` into your src folder.
(For CDH 5.8, `$HADOOP_HOME` is `/usr/lib/hadoop`)
3. Right click your java file → Run As → Run Configurations.... Click on Arguments, then enter your parameters for input and output folder paths, which are corresponding to line 57 and 58 in WordCount example. Note that the paths can be either relative path under project folder or absolute path outside of the project.



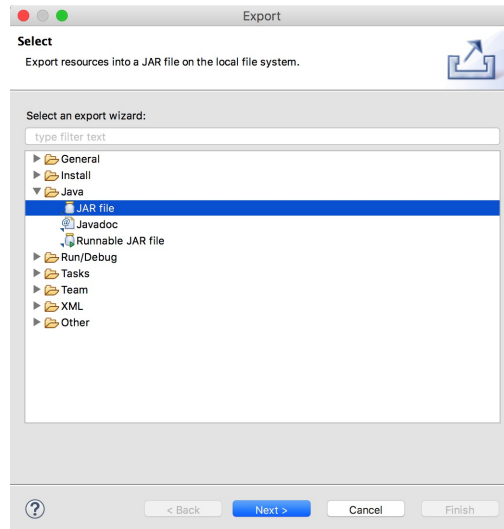
4. Click Run and check what you have in your output directory.



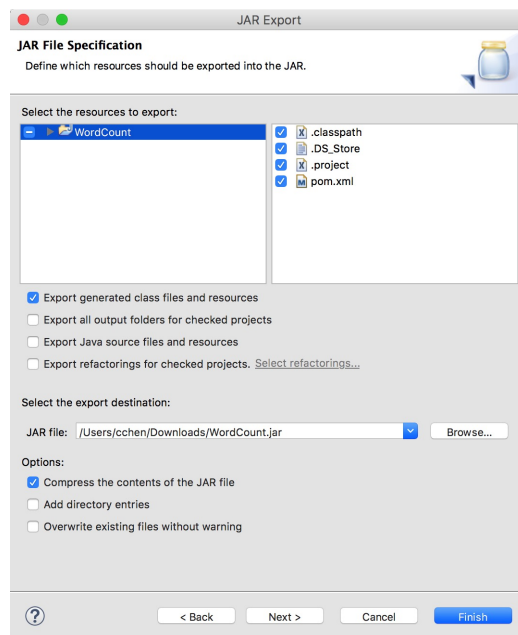
2 Run WordCount in Terminal

2.1 Create JAR file

1. Right click on the project → Export
2. In the pop-up dialog, expand the Java node and select JAR file. Click Next.



3. Enter your export destination (e.g. `/User/yourname/Downloads/WordCount.jar`) and click Finish.



2.2 Execution

1. Start hadoop using start-all.sh command mentioned in the Hadoop Installation Tutorial. (Hadoop is automatically started in CDH)

2. Create a directory in hadoop file system

```
$ hadoop fs -mkdir /user/hadoop/
```

3. Copy your local input file into the file system.

```
$ hadoop fs -put <your_local_filepath> /user/hadoop/  
$ hadoop fs -ls /user/hadoop/
```

4. Run WordCount. Use <class_name>, <input_path>, <output_path> as parameters.

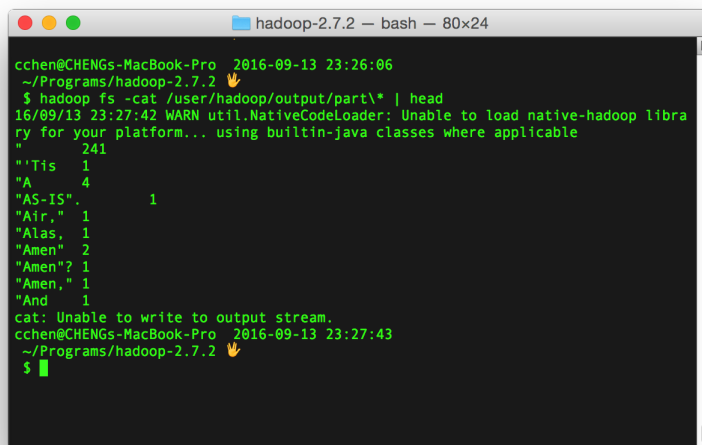
```
$ hadoop jar <JAR_filepath> WordCount /user/hadoop/pg100.txt /user/hadoop/output
```

5. Check output

```
$ hadoop fs -cat /user/hadoop/output/part\* | head
```

6. Copy output to local filesystem

```
$ hadoop fs -get /user/hadoop/output/part\* <your_directory>
```



7. To view the jobs logs, open the browser and point it to <http://localhost:8088>. You can also click on the job and check status.



All Applications

Logged in as: dr.who

Cluster

About Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	1	0	1	2 GB	8 GB	0 B	1	8	0	1	0	0	0	0

Scheduler Metrics

Capacity Scheduler

Scheduling Resource Type

Minimum Allocation

Maximum Allocation

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Show 20 entries

ID

User

Name

Application Type

Queue

StartTime

FinishTime

State

FinalStatus

Progress

Tracking UI

Blacklisted Nodes

application_147387729778_0001

cchen

word count

MAPREDUCE

default

Wed Sep 14 19:24:05 -0500 2016

Wed Sep 14 19:24:23 -0500 2016

FINISHED

SUCCEEDED

History

0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

A WordCount v1.0

```
1  import java.io.IOException;
2  import java.util.StringTokenizer;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17     extends Mapper<Object, Text, Text, IntWritable>{
18
19         private final static IntWritable one = new IntWritable(1);
20         private Text word = new Text();
21
22         public void map(Object key, Text value, Context context
23             ) throws IOException, InterruptedException {
24             StringTokenizer itr = new StringTokenizer(value.toString());
25             while (itr.hasMoreTokens()) {
26                 word.set(itr.nextToken());
27                 context.write(word, one);
28             }
29         }
30     }
31
32     public static class IntSumReducer
33     extends Reducer<Text, IntWritable, Text, IntWritable> {
34         private IntWritable result = new IntWritable();
35
36         public void reduce(Text key, Iterable<IntWritable> values,
37             Context context
38             ) throws IOException, InterruptedException {
39             int sum = 0;
40             for (IntWritable val : values) {
41                 sum += val.get();
42             }
43             result.set(sum);
44             context.write(key, result);
45         }
46     }
47
48     public static void main(String[] args) throws Exception {
49         Configuration conf = new Configuration();
50         Job job = Job.getInstance(conf, "word count");
51         job.setJarByClass(WordCount.class);
52         job.setMapperClass(TokenizerMapper.class);
53         job.setCombinerClass(IntSumReducer.class);
54         job.setReducerClass(IntSumReducer.class);
55         job.setOutputKeyClass(Text.class);
56         job.setOutputValueClass(IntWritable.class);
57         FileInputFormat.addInputPath(job, new Path(args[0]));
58         FileOutputFormat.setOutputPath(job, new Path(args[1]));
59         System.exit(job.waitForCompletion(true) ? 0 : 1);
60     }
61 }
```