

WordCount v1.0 Source Code ([#topic_5_1](#))

You can find source code for three versions of WordCount at

<http://tiny.cloudera.com/hadoopTutorialSample>

(<http://tiny.cloudera.com/hadoopTutorialSample>). This topic examines and summarizes the code for WordCount version 1.0.

You can use an appropriate package for your domain, or keep the generic version.

```
package org.myorg;
```

The only standard Java classes you need to import are `IOException` and `regex.Pattern`. You use `regex.Pattern` to extract words from input files.

```
import java.io.IOException;
import java.util.regex.Pattern;
```

This application extends the class `Configured`, and implements the `Tool` utility class. You tell Hadoop what it needs to know to run your program in a configuration object. Then, you use `ToolRunner` to run your MapReduce application.

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

The `Logger` class sends debugging messages from inside the mapper and reducer classes. When you run the application, one of the standard `INFO` messages provides a URL you can use to track the job's success. Messages you pass to `Logger` are displayed in the map or reduce logs for the job on your Hadoop server.

```
import org.apache.log4j.Logger;
```

You need the `Job` class to create, configure, and run an instance of your MapReduce application. You extend the `Mapper` class with your own `Mapclass` and add your own processing instructions. The same is true for the `Reducer`: you extend it to create and customize your own `Reduce` class.

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

Use the `Path` class to access files in HDFS. In your job configuration instructions, you pass required paths using the `FileInputFormat` and `FileOutputFormat` classes.

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

Writable objects have convenience methods for writing, reading, and comparing values during map and reduce processing. You can think of the `Text` class as *StringWritable*, because it performs essentially the same functions as those for integer (`IntWritable`) and long integer (`LongWritable`) objects.

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
```

WordCount includes main and run methods, and the inner classes Map and Reduce. The class begins by initializing the logger.

```
public class WordCount extends Configured implements Tool {
```

```
    private static final Logger LOG = Logger.getLogger(WordCount.class);
```

The main method invokes ToolRunner, which creates and runs a new instance of WordCount, passing the command line arguments. When the application is finished, it returns an integer value for the status, which is passed to the System object on exit.

```
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new WordCount(), args);
        System.exit(res);
    }
```

The run method configures the job (which includes setting paths passed in at the command line), starts the job, waits for the job to complete, and then returns an integer value as the success flag.

```
    public int run(String[] args) throws Exception {
```

Create a new instance of the Job object. This example uses the Configured.getConf() method to get the configuration object for this instance of WordCount, and names the job object *wordcount*.

```
    Job job = Job.getInstance(getConf(), "wordcount");
```

Set the JAR to use, based on the class in use.

```
    job.setJarByClass(this.getClass());
```

Set the input and output paths for your application. You store your input files in HDFS, and then pass the input and output paths as command-line arguments at runtime.

```
    FileInputFormat.addInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

Set the map class and reduce class for the job. In this case, use the Map and Reduce inner classes defined in this class.

```
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
```

Use a Text object to output the key (in this case, the word being counted) and the value (in this case, the number of times the word appears).

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

Launch the job and wait for it to finish. The method syntax is `waitForCompletion(boolean verbose)`. When true, the method reports its progress as the Map and Reduce classes run. When false, the method reports progress up to, but not including, the Map and Reduce processes.

In Unix, 0 indicates success, and anything other than 0 indicates a failure. When the job completes successfully, the method returns 0. When it fails, it returns 1.

```
    return job.waitForCompletion(true) ? 0 : 1;
}
```

The Map class (an extension of Mapper) transforms key/value input into intermediate key/value pairs to be sent to the Reducer. The class defines several global variables, starting with an IntWritable for the value 1, and a Text object used to store each word as it is parsed from the input string.

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
```

Create a regular expression pattern you can use to parse each line of input text on word boundaries ("\\b"). Word boundaries include spaces, tabs, and punctuation.

```
private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");
```

Hadoop invokes the map method once for every key/value pair from your input source. This does not necessarily correspond to the intermediate key/value pairs output to the reducer. In this case, the map method receives the offset of the first character in the current line of input as the key, and a Text object representing an entire line of text from the input file as the value. It further parses the words on the line to create the intermediate output.

```
public void map(LongWritable offset, Text lineText, Context context)
    throws IOException, InterruptedException {
```

Convert the Text object to a string. Create the currentWord variable, which you use to capture individual words from each input string.

```
String line = lineText.toString();
Text currentWord = new Text();
```

Use the regular expression pattern to split the line into individual words based on word boundaries. If the word object is empty (for example, consists of white space), go to the next parsed object. Otherwise, write a key/value pair to the context object for the job.

```
for (String word : WORD_BOUNDARY.split(line)) {
    if (word.isEmpty()) {
        continue;
    }
    currentWord = new Text(word);
    context.write(currentWord, one);
}
}
```

The mapper creates a key/value pair for each word, composed of the word and the IntWritable value 1. The reducer processes each pair, adding one to the count for the current word in the key/value pair to the overall count of that word from all mappers. It then writes the result for that word to the reducer context object, and moves on to the next. When all of the intermediate key/value pairs are processed, the map/reduce task is complete. The application saves the results to the output location in HDFS.

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override public void reduce(Text word, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable count : counts) {
```

```

        sum += count.get();
    }
    context.write(word, new IntWritable(sum));
}
}
}

```



Categories: [Applications \(../categories/hub_applications.html\)](#) | [Hadoop \(../categories/hub_hadoop.html\)](#) | [MapReduce \(../categories/hub_mapreduce.html\)](#) | [Source code \(../categories/hub_source_code.html\)](#) | [Tutorial \(../categories/hub_tutorial.html\)](#) | [WordCount 1.0 \(../categories/hub_wordcount_1.0.html\)](#) | [All Categories \(../categories/hub.html\)](#)

- [About Cloudera \(http://www.cloudera.com/about-cloudera.html\)](http://www.cloudera.com/about-cloudera.html)
- [Resources \(http://www.cloudera.com/resources.html\)](http://www.cloudera.com/resources.html)
- [Contact \(http://www.cloudera.com/contact-us.html\)](http://www.cloudera.com/contact-us.html)
- [Careers \(http://www.cloudera.com/about-cloudera/careers.html\)](http://www.cloudera.com/about-cloudera/careers.html)
- [Press \(/about-cloudera/press-center.html\)](/about-cloudera/press-center.html)
- [Documentation \(/documentation.html\)](/documentation.html)

United States: +1 888 789 1488

Outside the US: +1 650 362 0488

© 2016 Cloudera, Inc. All rights reserved. [Apache Hadoop \(http://hadoop.apache.org\)](http://hadoop.apache.org) and associated open source project names are trademarks of the [Apache Software Foundation \(http://apache.org\)](http://apache.org). For a complete list of trademarks, [click here. \(/legal/terms-and-conditions.html\)](/legal/terms-and-conditions.html)

-  (<https://www.linkedin.com/company/cloudera>)
-  (<https://www.facebook.com/cloudera>)
-  (<https://twitter.com/cloudera>)
-  (</contact-us.html>)

[Terms & Conditions \(/legal/terms-and-conditions.html\)](/legal/terms-and-conditions.html) | [Privacy Policy \(/legal/privacy-policy.html\)](/legal/privacy-policy.html)

Page generated June 29, 2016.