# Word Count With Two Text file

## Code

```
package two_file_wc;

import java.io.IOException;
import java.util.StringTokenizer;


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class two_file_wc {


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


    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        context.write(word, one);
```

```
            }
        }
    }


    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }


    public static void main(String[] args) throws Exception {


        if (args.length != 3) {
            System.err.println("Usage: WordCountTwoFiles <input path1> <input path2> <output
path>");
            System.exit(-1);
        }


        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Word Count for Two Files");


        job.setJarByClass(two_file_wc.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);


        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);


        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);


        // Add both input files
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileInputFormat.addInputPath(job, new Path(args[1]));


        Path outputPath = new Path(args[2]);
        FileOutputFormat.setOutputPath(job, outputPath);


        // Delete output folder if it exists
        outputPath.getFileSystem(conf).delete(outputPath, true);


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

## Code with explanation

**package two_file_wc;**
This defines the package name where your Java class belongs. It helps in organizing your Hadoop programs.

**import java.io.IOException;**
Imports the `IOException` class to handle input/output-related errors.

**import java.util.StringTokenizer;**
Imports `StringTokenizer` — used to split a line of text into individual words (tokens).

**import org.apache.hadoop.conf.Configuration;**
Used to access Hadoop configuration settings.

**import org.apache.hadoop.fs.Path;**
Represents a file or directory path in the Hadoop File System (HDFS).

**import org.apache.hadoop.io.IntWritable;**
A Hadoop data type for storing integer values (used instead of Java’s `int`).

**import org.apache.hadoop.io.LongWritable;**
A Hadoop data type for storing long integer values.

**import org.apache.hadoop.io.Text;**
A Hadoop data type for strings (used instead of Java’s `String`).

**import org.apache.hadoop.mapreduce.Job;**
Represents a MapReduce job configuration (including Mapper, Reducer, input, output paths, etc.).

**import org.apache.hadoop.mapreduce.Mapper;**
Base class for writing the Mapper logic.

**import org.apache.hadoop.mapreduce.Reducer;**
Base class for writing the Reducer logic.

**import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;**
Used to define the input file(s) or directory for the MapReduce job.

**import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;**
Specifies that the input will be plain text files, where each line is a record.

**import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;**
Used to define the output directory for the MapReduce job.

**import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;**
Specifies that the output will be stored as plain text.

---

**public class two_file_wc {**
This defines your main public class. It contains both the Mapper, Reducer, and main method.

---

## Mapper Class

**public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {**
Defines an inner static class named `Map` that extends Hadoop's `Mapper` class.
It takes input key-value pairs:

- Key → `LongWritable` (line offset)
- Value → `Text` (line content)
  And outputs key-value pairs:
- Key → `Text` (a word)
- Value → `IntWritable` (count 1)

---

**private final static IntWritable one = new IntWritable(1);**
Creates a constant value `1`, which represents one occurrence of a word.

**private Text word = new Text();**
Creates a reusable Text object to store each word.

---

**public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {**
This is the Mapper's main method — Hadoop calls it once for each line of input.

---

**String line = value.toString();**
Converts the line from `Text` to a regular Java `String`.

**StringTokenizer tokenizer = new StringTokenizer(line);**
Splits the line into individual words using spaces as delimiters.

---

**while (tokenizer.hasMoreTokens()) {**
Loops through all the words in the line.

---

**word.set(tokenizer.nextToken());**
Stores the next word in the `word` variable.

---

**context.write(word, one);**
Emits the word along with the count `1` to the Hadoop framework (Mapper output).

---

## Reducer Class

**public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {**
Defines a `Reduce` class that extends Hadoop's `Reducer`.
It takes input from Mapper:

- Key → `Text` (word)
- Values → list of `IntWritable` (counts)
  And outputs:
- Key → `Text` (word)
- Value → `IntWritable` (final total count)

---

**public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {**
This method is called once for each unique word key.

---

**int sum = 0;**
Initializes a counter for total occurrences of a word.

---

**for (IntWritable val : values) { sum += val.get(); }**
Iterates over all counts for that word and adds them up.

---

**context.write(key, new IntWritable(sum));**
Writes the final word and its total count to output.

---

## Main Method

**public static void main(String[] args) throws Exception {**
Main function — entry point of the program.

---

**if (args.length != 3) { ... }**
Checks whether exactly three arguments are given (two input files and one output directory).
If not, it prints a usage message and exits.

---

**Configuration conf = new Configuration();**
Creates a Hadoop configuration object to hold job settings.

---

**Job job = Job.getInstance(conf, "Word Count for Two Files");**
Creates and names a new MapReduce job instance.

---

**job.setJarByClass(two_file_wc.class);**
Specifies which class contains the job's main method (used to locate your JAR).

---

**job.setMapperClass(Map.class);**
Tells Hadoop which Mapper class to use.

**job.setReducerClass(Reduce.class);**
Tells Hadoop which Reducer class to use.

---

**job.setMapOutputKeyClass(Text.class);**
Defines the Mapper output key type (word).

**job.setMapOutputValueClass(IntWritable.class);**
Defines the Mapper output value type (count 1).

---

**job.setOutputKeyClass(Text.class);**
Defines the final output key type (word).

**job.setOutputValueClass(IntWritable.class);**
Defines the final output value type (total count).

---

**job.setInputFormatClass(TextInputFormat.class);**
Specifies that the input files are text files.

**job.setOutputFormatClass(TextOutputFormat.class);**
Specifies that the output will be text format.

---

**FileInputFormat.addInputPath(job, new Path(args[0]));**
Sets the first input file path.

**FileInputFormat.addInputPath(job, new Path(args[1]));**
Adds the second input file path. Both files will be processed together.

---

**Path outputPath = new Path(args[2]);**
Specifies the output directory path.

**FileOutputFormat.setOutputPath(job, outputPath);**
Sets where the result will be stored in HDFS.

---

**outputPath.getFileSystem(conf).delete(outputPath, true);**
Deletes the output directory if it already exists to prevent job failure.

---

**System.exit(job.waitForCompletion(true) ? 0 : 1);**
Runs the job and exits:

- Returns `0` if successful
- Returns 1 if it fails
-

## -: Commands :-

```
administrator@administrator:~$ hadoop jar
/home/administrator/HadoopJar/TwoFileWC.jar two_file_wc.two_file_wc
/data/TFWC/T1 /data/TFWC/T2 /data/TFWCoutput
```

- `administrator@administrator:~$ hadoop fs -ls /data/`

```
Found 2 items

drwxr-xr-x   - administrator supergroup          0 2025-10-07 16:08 /data/TFWC
drwxr-xr-x   - administrator supergroup          0 2025-10-07 16:12
/data/TFWCoutput
```

- `administrator@administrator:~$ hadoop fs -ls /data/TFWCoutput`

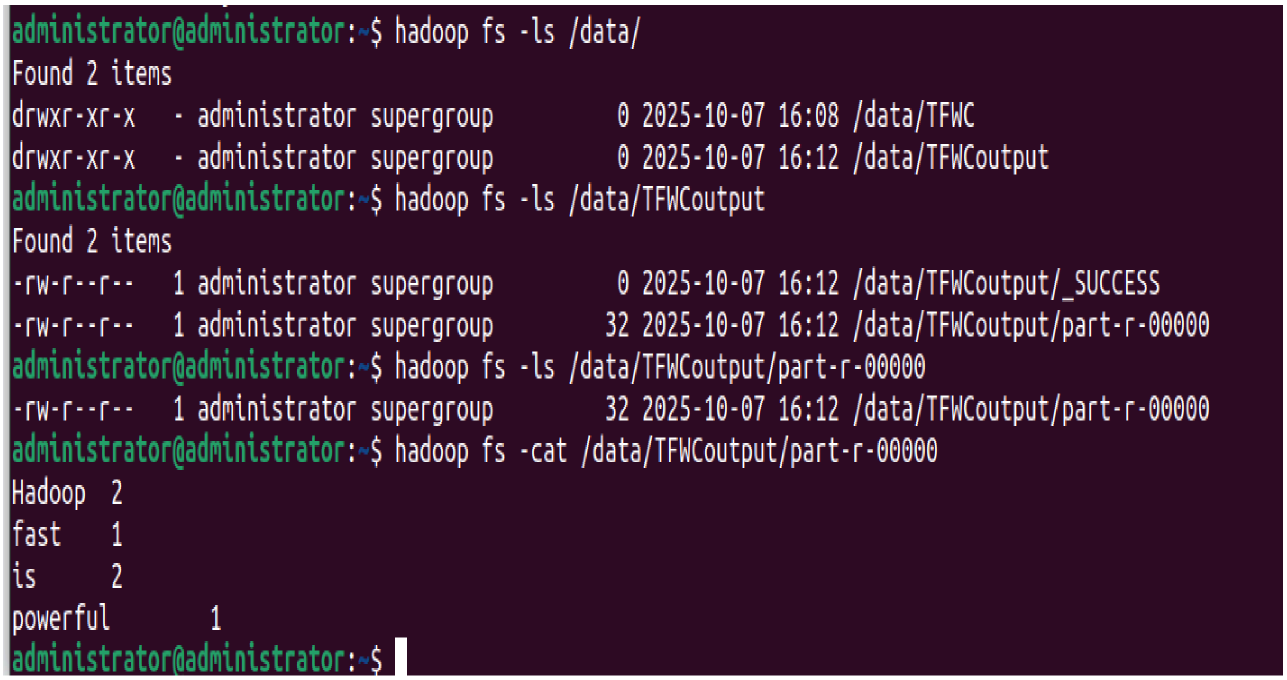
```
Found 2 items

-rw-r--r--   1 administrator supergroup          0 2025-10-07 16:12
/data/TFWCoutput/_SUCCESS
-rw-r--r--   1 administrator supergroup         32 2025-10-07 16:12
/data/TFWCoutput/part-r-00000
```

- `administrator@administrator:~$ hadoop fs -ls /data/TFWCoutput/part-r-00000`

```
-rw-r--r--   1 administrator supergroup         32 2025-10-07 16:12
/data/TFWCoutput/part-r-00000
```

- administrator@administrator:~$ hadoop fs -cat /data/TFWCoutput/part-r-00000

```
Hadoop2
fast   1
is     2
powerful     1
```



```
administrator@administrator:~$ hadoop fs -ls /data/
Found 2 items
drwxr-xr-x   - administrator supergroup          0 2025-10-07 16:08 /data/TFWC
drwxr-xr-x   - administrator supergroup          0 2025-10-07 16:12 /data/TFWCoutput
administrator@administrator:~$ hadoop fs -ls /data/TFWCoutput
Found 2 items
-rw-r--r--   1 administrator supergroup          0 2025-10-07 16:12 /data/TFWCoutput/_SUCCESS
-rw-r--r--   1 administrator supergroup         32 2025-10-07 16:12 /data/TFWCoutput/part-r-00000
administrator@administrator:~$ hadoop fs -ls /data/TFWCoutput/part-r-00000
-rw-r--r--   1 administrator supergroup         32 2025-10-07 16:12 /data/TFWCoutput/part-r-00000
administrator@administrator:~$ hadoop fs -cat /data/TFWCoutput/part-r-00000
Hadoop  2
fast    1
is      2
powerful        1
administrator@administrator:~$
```