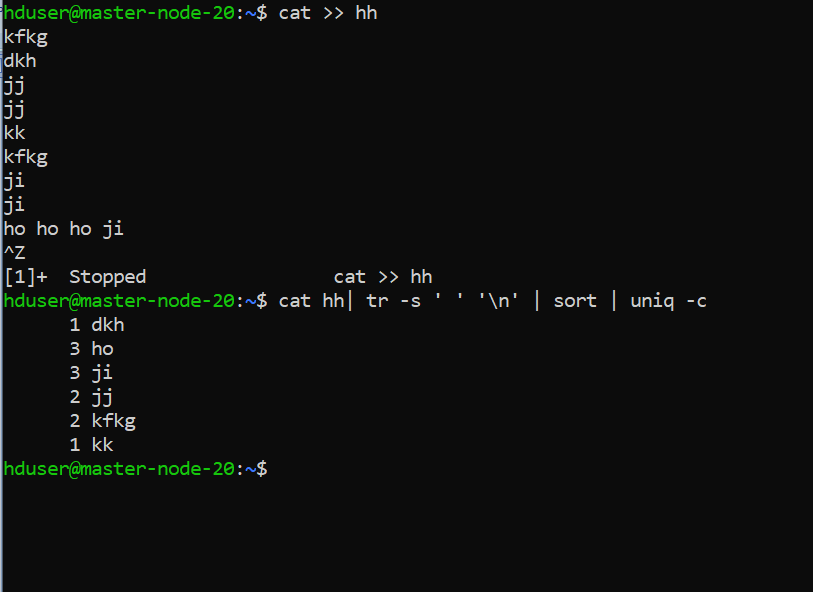
Unix command **cat sample.txt | tr -s ' ' '\n' | sort | uniq -c** step by step:

1. **cat sample.txt**:
   * **cat** is a command that concatenates and displays the content of files. In this case, it's used to display the content of the file **sample.txt**.
2. **tr -s ' ' '\n'**:
   * **tr** is the translate or delete characters command. Here, it is used to replace spaces (' ') with newline characters ('\n').
   * The option **-s** squeezes multiple occurrences of the specified set of characters into a single character. In this case, it squeezes multiple spaces into a single space.
3. **sort**:
   * **sort** is a command that sorts lines of text. In this case, it sorts the lines alphabetically.
4. **uniq -c**:
   * **uniq** is a command that filters out adjacent matching lines in a sorted file. The option **-c** is used to prefix lines by the number of occurrences.
   * So, **uniq -c** counts the number of occurrences of each unique line (word in this case,
   * **cat sample.txt**: Display the content of the file.

* **tr -s ' ' '\n'**: Replace spaces with newline characters, squeezing multiple spaces into a single space.
* **sort**: Sort the lines alphabetically.
* **uniq -c**: Count the occurrences of each unique line (word) and display the count along with the line.

The overall effect of this command is to count the occurrences of each unique word in the **sample.txt** file. It's a simple example of using a combination of Unix commands to perform a basic text processing task.

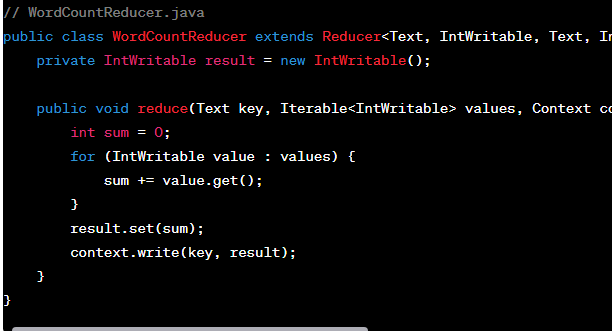


Using Hadoop:

Now, let's consider the same task using Hadoop for processing a larger dataset, which could be a collection of many text files.

Assuming you have Hadoop set up and the data is stored in Hadoop Distributed File System (HDFS), you could use a MapReduce program to perform the word count.







# Using Unix tools

cat file1.txt file2.txt file3.txt | wc –l

// LineCountMapper.java

public class LineCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text("totalLines");

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

context.write(word, one);

}

}

// LineCountReducer.java

public class LineCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable value : values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

// LineCountDriver.java

public class LineCountDriver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "line count");

job.setJarByClass(LineCountDriver.class);

job.setMapperClass(LineCountMapper.class);

job.setCombinerClass(LineCountReducer.class);

job.setReducerClass(LineCountReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path("input\_directory"));

FileOutputFormat.setOutputPath(job, new Path("output\_directory"));

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

}

}