STEP-BY-STEP GUIDE FOR SETTING UP AND RUNNING HADOOP, STARTING FROM START-DFS.SH AND CREATING A WORDCOUNT PROGRAM.

Step 1: Start Hadoop Services

1. Start HDFS (Hadoop Distributed File System)

- Navigate to your Hadoop installation directory:

```
cd $HADOOP_HOME
```

- Start the HDFS services:

```
sbin/start-dfs.sh
```

- Verify that the services are running:

```
jps
```

You should see processes like:

```
NameNode

DataNode

SecondaryNameNode
```

---

2. Start YARN (If Required)

- Start YARN services for managing MapReduce jobs:

```
sbin/start-yarn.sh
```

- Verify that the ResourceManager and NodeManager are running:

```
jps
```

You should see processes like:

```
ResourceManager

NodeManager
```

- Access the Hadoop web interfaces for monitoring:
  - **HDFS Web UI**: http://localhost:9870/
  - **YARN ResourceManager UI**: http://localhost:8088/

Step 2: Prepare Input Data

1. Create a Sample Input File

- Use echo to create a sample input.txt file:

```
echo -e "Hadoop is great\nHadoop is powerful\nHadoop is scalable" > input.txt
```

- Verify the content of the file:

```
cat input.txt
```

2. Create Input Directory in HDFS

- Check if the /input directory already exists in HDFS:

```
hdfs dfs -ls /
```

- If it doesn't exist, create it:

```
hdfs dfs -mkdir /input
```

3. Add Input Data to HDFS

- Copy your input.txt file to the /input directory in HDFS:

```
hdfs dfs -put input.txt /input
```

- If input.txt already exists, Delete it using following command.

```
hdfs dfs -rm /input/input.txt
```

- Verify the file was uploaded successfully:

```
hdfs dfs -ls /input
```

Step 3: Compile the WordCount Program

1. Set Up a Project Directory

- Create a project directory:

```
mkdir ~/Desktop/Project
cd ~/Desktop/Project
```

2. Write the Java Code

- Create the following files in the project directory:
  - WordCountMapper.java
  - WordCountReducer.java

- WordCountDriver.java

Example Code:

WordCountMapper.java:

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

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {

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

  private Text word = new Text();

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

    String[] words = value.toString().split("\\s+");

    for (String str : words) {

      word.set(str);

      context.write(word, one);

    }

  }

}
```

WordCountReducer.java:

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

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class WordCountReducer 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));
    }
}
```

WordCountDriver.java:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCountDriver {
  public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCountDriver.class);
      job.setMapperClass(WordCountMapper.class);
      job.setReducerClass(WordCountReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
    }
}
```

---

3. Compile the Code

- Compile all .java files:

```
mkdir wordcount_classes
javac -classpath $(hadoop classpath) -d wordcount_classes WordCount*.java
```

4. Create the JAR File

- Package the compiled .class files into a JAR:

```
jar cf WordCount.jar -C wordcount_classes/ .
```

- Verify the JAR contents:

```
jar tf WordCount.jar
```

Step 4: Run the WordCount Program

1. Remove Old Output Directory

- Delete the /output directory in HDFS if it exists:

```
hdfs dfs -rm -r /output
```

2. Execute the Program

- Run the Hadoop job:

```
hadoop jar WordCount.jar WordCountDriver /input /output
```

3. View the Output

- Check the output files generated in HDFS:

```
hdfs dfs -ls /output
```

- Display the results:

```
hdfs dfs -cat /output/part-r-00000
```

Step 5: Troubleshooting

Common Errors and Fixes

1. **Could not find or load main class**:
    - Ensure the JAR file contains all .class files and the Driver class is set properly.
2. **Input/Output directory already exists**:
    - Remove the old directory:

      hdfs dfs -rm -r /output
3. **Check logs**:
    - Visit the YARN ResourceManager web interface http://localhost:8088/ for job logs.

Let me know if you face any issues!