1. 安装jdk。
1.1 在用户主目录下面建立jdk文件夹
```
[ghostfire@turing]$ mkdir jdk
```

1.2 解压缩 jdk.tar.gz 文件到jdk文件夹中
```
[ghostfire@turing]$ tar -zxvf ./jdk.tar.gz ./jdk/
```

1.3 将java路径加入path中
```
[ghostfire@turing]$ vim ～/.bashrc
```
在文件末尾添加如下内容并保存。
```
    export JAVA_HOME=~/jdk/jdk1.7.0_09
    export CLASSPATH=.:$JAVA_HOME/lib/*jar
    export PATH=$JAVA_HOME/bin:$PATH
```
执行如下命令，使得我们设置的path能够马上生效。
```
[ghostfire@turing]$ source ~/.bashrc
```

1.5 检测jdk是否安装成功
```
[ghostfire@turing]$ java -version
```

2. 验证并安装ssh
2.1 首先检查是否安装了ssh
```
[ghostfire@turing]$ which ssh
[ghostfire@turing]$ which sshd
[ghostfire@turing]$ which ssh-keygen
```

2.2 如果提示没有安装或者无任何内容显示，执行如下命令安装ssh
```
[ghostfire@turing]$ sudo apt-get install openssh-client
[ghostfire@turing]$ sudo apt-get install openssh-server
```

2.3 检测sshd服务是否启动
```
[ghostfire@turing]$ ps aux | grep sshd
```
结果中若显示sshd(注意显示 grep
sshd不算)，则sshd服务成功启动,否则执行如下命令启动sshd服务
```
[ghostfire@turing]$ sudo /etc/init.d/ssh start
```
注意在有些版本下,命令可能是 sudo /etc/init.d/sshd start

3.生成ssh秘钥对
3.1 生成ssh公钥
```
[ghostfire@turing]$ ssh-keygen -t rsa
```
待输入的地方全部回车选择默认
执行完毕后，会在 ~/.ssh/下面生成私钥 id_rsa,公钥id_rsa.pub

3.2 公钥添加
```
[ghostfire@turing]$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
[ghostfire@turing]$ chmod 600 ~/.ssh/authorized_keys
```

3.3 检测公钥是否配置完成
```
[ghostfire@turing]$ ssh localhost
```
如果配置成功，则不需要密码就可以通过ssh进入localhost

4.安装hadoop
4.1 在用户主目录下建立hadoop文件夹
```
[ghostfire@turing]$ mkdir hadoop
```

4.2 解压缩hadoop-1.0.4.tar.gz
```
[ghostfire@turing]$ tar -zxvf ./hadoop-1.0.4.tar.gz ./hadoop
```

4.3 将hadoop路径加入path
```
[ghostfire@turing]$ vim ～/.bashrc
```
在文件末尾添加如下内容并保存。
```
    export HADOOP_HOME=~/hadoop/hadoop-1.0.4
    export PATH=$HADOOP_HOME/bin:$PATH
```
执行如下命令，使得我们设置的path能够马上生效。
```
[ghostfire@turing]$ source ~/.bashrc
```

4.4 配置hadoop-env.sh
修改~/hadoop/hadoop-1.0.4/conf/hadoop-env.sh
在该文件最后一行添加
```
export JAVA_HOME=~/jdk/jdk1.7.0_09
```

5. 配置单机模式
对conf目录下面的配置文件不做修改即为单机模式

6. 配置伪分布模式
6.1 修改core-site.xml文件，内容如下
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

6.2 修改mapred-site.xml文件，内容如下
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
    </property>
</configuration>
```

6.3 修改hdfs-site.xml文件，内容如下
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
<configuration>
```

6.4 将localhost添加到hadoop conf目录下面的masters文件中
```
[ghostfire@turing]$ echo "localhost" >> masters
```

6.5 将localhost添加到hadoop conf目录下面的slaves文件中
```
[ghostfire@turing]$ echo "localhost" >> slaves
```

7. 格式化HDFS
```
[ghostfire@turing]$ ~/hadoop/hadoop-1.0.4/bin/hadoop namenode -format
```

8. 启动hadoop
```
[ghostfire@turing]$ ~/hadoop/hadoop-1.0.4/bin/start-all.sh
```

9. 检测hadoop是否成功启动
```
[ghostfire@turing]$ jps
TaskTracker
SecondaryNameNode
NameNode
DateNode
JobTracker
```

10. 在HDFS中添加文件和目录
```
[ghostfire@turing]$ hadoop fs -mkdir /user/[你的用户名]/wordcount/input
```
上面的命令本质上是递归创建的,但在有的版本上是不支持的,那么需要你依次执行如下命令
```
[ghostfire@turing]$ hadoop fs -mkdir /user
[ghostfire@turing]$ hadoop fs -mkdir /user/[你的用户名]
[ghostfire@turing]$ hadoop fs -mkdir /user/[你的用户名]/wordcount
[ghostfire@turing]$ hadoop fs -mkdir /user/[你的用户名]/wordcount/input
```

在当前目录下面创建若干个文本文件，每个文件里面添加若干个英文单词，比如
```
[ghostfire@turing]$ cat input1.txt
no zuo no die
you can you up
[ghostfire@turing]$ cat input2.txt
you can you up
no zuo no die
```

将文本文件从本地目录上传到HDFS中
```
[ghostfire@turing]$ hadoop fs -put ./input1.txt /user/[你的用户名]/wordcount/input
[ghostfire@turing]$ hadoop fs -put ./input2.txt /user/[你的用户名]/wordcount/input
```

查看文件上传是否成功
```
[ghostfire@turing]$ hadoop fs -lsr /
```

11. 在当前目录下新建一个WordCount.java文件
```
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.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.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
```

```
public class WordCount {

  public static class TokenizerMapper
       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 {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       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 val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

12. 编译WordCount.java
[ghostfire@turing]$ mkdir wordcount
[ghostfire@turing]$ cp ./WordCount.java ./wordcount
[ghostfire@turing]$ cd ./wordcount
[ghostfire@turing]$ mkdir classes
[ghostfire@turing]$ javac -classpath
/home/[你的用户名]/hadoop/hadoop-1.0.4/hadoop-core-1.0.4.jar:/home/[你的用户名]/hadoop/hadoop-1.0.4/lib/commons-cli-1.2.jar -d ./classes/ ./WordCount.java
(注意,如果有同学用的是hadoop-2以上版本的,classpath和上面的区别非常大,具体请参考如下几个链接
http://stackoverflow.com/questions/19223288/hadoop-2-1-0-beta-javac-compile-error
http://stackoverflow.com/questions/19488894/compile-hadoop-2-2-0-job)

[ghostfire@turing]$ jar -cvf ./WordCount.jar -C ./classes .
(注意,打包的时候一定不能忘记了上面命令最后的点号)

13. 运行Hadoop作业
[ghostfire@turing]$ hadoop jar ~/wordcount/WordCount.jar WordCount
/user/[你的用户名]/wordcount/input /user/[你的用户名]/wordcount/output
如果提示你说输出文件夹已经存在，那么则执行如下命令删除
[ghostfire@turing]$ hadoop fs -rmr /user/[你的用户名]/wordcount/output

14. 获得运行结果
[ghostfire@turing]$ hadoop fs -ls /user/[你的用户名]/wordcount/output
[ghostfire@turing]$ hadoop fs -get /user/[你的用户名]/wordcount/output/[文件名] ./

15. 关闭hadoop集群
[ghostfire@turing]$ stop-all.sh